

Contents

1 A Better Way To Approach Competitive Programming	13
Source	15
2 AKS Primality Test	16
Source	24
3 Algorithm Library C++ Magicians STL Algorithm	25
Source	31
4 Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)	32
Source	35
5 Array range queries over range queries	36
Source	45
6 Array with GCD of any of its subset belongs to the given array	46
Source	52
7 BFS using STL for competitive coding	53
Source	56
8 Bit Tricks for Competitive Programming	57
Source	60
9 Bits manipulation (Important tactics)	61
Source	64
10 Bitwise Hacks for Competitive Programming	65
Source	70
11 Burst Balloon to maximize coins	71
Source	72
12 C qsort() vs C++ sort()	73
Source	76
13 C++ tricks for competitive programming (for C++ 11)	77
Source	80

14 C++: Methods of code shortening in competitive programming	81
Source	85
15 Check if a M-th fibonacci number divides N-th fibonacci number	86
Source	90
16 Check if any permutation of a large number is divisible by 8	91
Source	99
17 Check if concatenation of two strings is balanced or not	100
Source	107
18 Check if it is possible to convert one string into another with given constraints	108
Source	110
19 Check if the large number formed is divisible by 41 or not	111
Source	118
20 Check if the last element of array is even or odd after performing a operation p times	119
Source	124
21 Cin-Cout vs Scanf-Printf	125
Source	128
22 Cleaning the room	129
Source	132
23 Combinatorics on ordered trees	133
Source	143
24 Common mistakes to be avoided in Competitive Programming in C++ Beginners	144
Source	151
25 Competitive Programming: Conquering a given problem	152
Source	154
26 Container with Most Water	155
Source	160
27 Count inversions of size k in a given array	161
Source	167
28 Count number of primes in an array	168
Source	170
29 Count number of right triangles possible with a given perimeter	171
Source	173

30 Count strings with consonants and vowels at alternate position	174
Source	176
31 Count unique subsequences of length K	177
Source	180
32 Data Type Ranges and their macros in C++	181
Source	183
33 Difference between the summation of numbers whose frequency of all digits are same and different	184
Source	186
34 Digit DP Introduction	187
Source	192
35 Distinct Prime Factors of Array Product	193
Source	196
36 Dynamic Disjoint Set Data Structure for large range values	197
Source	202
37 Dynamic Programming Wildcard Pattern Matching Linear Time and Constant Space	203
Source	209
38 Element which occurs consecutively in a given subarray more than or equal to K times	210
Source	213
39 Euler tour of Binary Tree	214
Source	217
40 Extended Mo's Algorithm with O(1) time complexity	218
Source	229
41 Fast I/O for Competitive Programming	230
Source	233
42 Fast I/O in Java in Competitive Programming	234
Source	241
43 Find $(a^b) \% m$ where 'b' is very large	242
Source	248
44 Find Nth term (A matrix exponentiation example)	249
Source	252
45 Find if it is possible to reach the end through given transitions	253
Source	255

46 Find if neat arrangement of cups and shelves can be made	256
Source	263
47 Find the Largest Cube formed by Deleting minimum Digits from a number	264
Source	267
48 Find the arrangement of queue at given time	268
Source	273
49 Find the minimum time after which one can exchange notes	274
Source	280
50 Find the number of operations required to make all array elements Equal	281
Source	282
51 Find two numbers from their sum and XOR	283
Source	288
52 First occurrence of a digit in a given fraction	289
Source	294
53 Formatted output in Java	295
Source	298
54 Frequency Measuring Techniques for Competitive Programming	299
Source	306
55 Generating Test Cases (generate() and generate_n() in C++)	307
Source	309
56 Graph implementation using STL for competitive programming Set 1 (DFS of Unweighted and Undirected)	310
Source	312
57 Graph implementation using STL for competitive programming Set 2 (Weighted graph)	313
Source	315
58 How can competitive programming help you get a job?	316
Source	318
59 How to become a master in competitive programming?	319
Source	320
60 How to begin with Competitive Programming?	321
Source	327
61 How to get rid of Java TLE problem	328
Source	333

62 How to overcome Time Limit Exceed(TLE)?	334
Source	335
63 How to prepare for ACM – ICPC?	336
Source	341
64 How to prepare for Facebook Hacker Cup?	342
Source	346
65 How to prepare for Google Asia Pacific University (APAC) Test ?	347
Source	350
66 How to read Competitive Programming Questions?	351
Source	354
67 How to read content of GeeksforGeeks in an organized way?	355
Source	357
68 Inclusion Exclusion principle and programming applications	358
Source	365
69 Input/Output from external file in C/C++, Java and Python for Competitive Programming	366
Source	368
70 Input/Output from external file in C/C++, Java and Python for Competitive Programming Set 2	369
Source	371
71 Introduction to Programming Languages	372
Source	374
72 Java tricks for competitive programming (for Java 8)	375
Source	379
73 Knowing the complexity in competitive programming	380
Source	381
74 LCA for general or n-ary trees (Sparse Matrix DP approach < O(nlogn), O(logn)>)	382
Source	388
75 Largest connected component on a grid	389
Source	399
76 Largest number with one swap allowed	400
Source	405
77 Longest substring having K distinct vowels	406
Source	411

78 Making elements of two arrays same with minimum increment/decrement	412
Source	416
79 Matrix Exponentiation	417
Source	425
80 Maximize the bitwise OR of an array	426
Source	432
81 Maximize the number of segments of length p, q and r	433
Source	438
82 Maximize the total profit of all the persons	439
Source	441
83 Maximum Possible Product in Array after performing given Operations	442
Source	445
84 Maximum difference between groups of size two	446
Source	448
85 Maximum elements that can be made equal with k updates	449
Source	456
86 Maximum number of customers that can be satisfied with given quantity	457
Source	460
87 Maximum sum increasing subsequence from a prefix and a given element after prefix is must	461
Source	466
88 Minimum adjacent swaps to move maximum and minimum to corners	467
Source	473
89 Minimum cost path from source node to destination node via an intermediate node	474
Source	478
90 Minimum difference between groups of size two	479
Source	481
91 Minimum digits to remove to make a number Perfect Square	482
Source	491
92 Minimum number of elements to be removed to make XOR maximum	492
Source	494
93 Minimum number using set bits of a given number	495
Source	498

94 Minimum operations required to make all the elements distinct in an array	499
Source	500
95 Minimum steps to reach end from start by performing multiplication and mod operations with array elements	501
Source	503
96 Minimum total cost incurred to reach the last station	504
Source	511
97 Modulo 10^{9+7} (1000000007)	512
Source	515
98 Modulo power for large numbers represented as strings	516
Source	518
99 Multi Source Shortest Path in Unweighted Graph	519
Source	526
100 Multistage Graph (Shortest Path)	527
Source	530
101 NZEC error in Python	531
Source	533
102 Nth non-Square number	534
Source	537
103 Number of Larger Elements on right side in a string	538
Source	541
104 Number of Transpositions in a Permutation	542
Source	548
105 Number of days until all chocolates become unhealthy	549
Source	555
106 Number of digits in N factorial to the power N	556
Source	559
107 Number of digits in the nth number made of given four digits	560
Source	563
108 Number of distinct prime factors of first n natural numbers	564
Source	572
109 Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)	573
Source	578

110 Number of horizontal or vertical line segments to connect 3 points	579
Source	586
111 Number of integral solutions for equation $x = b*(\text{sumofdigits}(x)^a) + c$	587
Source	591
112 Number of odd and even results for every value of x in range [min, max] after performing N steps	592
Source	603
113 Number of prime pairs in an array	604
Source	606
114 Number of quadrilaterals possible from the given points	607
Source	615
115 Number of terms in Geometric Series with given conditions	616
Source	619
116 Number of ways to change the XOR of two numbers by swapping the bits	620
Source	627
117 Ordered Set and GNU C++ PBDS	628
Source	631
118 Pair of arrays with equal sum after removing exactly one element from each	632
Source	635
119 Pair with minimum absolute difference after solving each query	636
Source	640
120 Pairs involved in Balanced Parentheses	641
Source	645
121 Pairs with GCD equal to one in the given range	646
Source	649
122 Palindromic Tree Introduction & Implementation	650
Source	663
123 Passing the Assignment	664
Source	671
124 Permutation of a string with maximum number of characters greater than its adjacent characters	672
Source	675
125 Possible timings	676

Source	680
126 Practice for cracking any coding interview	681
Source	693
127 Prefix Sum Array – Implementation and Applications in Competitive Programming	694
Source	700
128 Print all subsequences of a string Iterative Method	701
Source	706
129 Program to find last two digits of 2^n	707
Source	722
130 Project Euler	723
Source	728
131 Python Input Methods for Competitive Programming	729
Source	732
132 Python Tricks for Competitive Coding	733
Source	735
133 Python in Competitive Programming	736
Source	738
134 Queries to find distance between two nodes of a Binary tree	739
Source	741
135 Queries to find distance between two nodes of a Binary tree – $O(\log n)$ method	742
Source	750
136 Queries to find maximum product pair in range with updates	751
Source	755
137 Querying the number of distinct colors in a subtree of a colored tree using BIT	756
Source	762
138 Quick ways to check for Prime and find next Prime in Java	763
Source	765
139 Range Minimum Query (Square Root Decomposition and Sparse Table)	766
Source	774
140 Remove the forbidden strings	775
Source	782
141 Represent a number as sum of minimum possible pseudobinary numbers	783

Source	789
142 Searching in a map using std::map functions in C++	790
Source	796
143 Sequence Alignment problem	797
Source	808
144 Smallest number with sum of digits as N and divisible by 10^N	809
Source	814
145 Some important shortcuts in Competitive Programming	815
Source	816
146 Sqrt (or Square Root) Decomposition Technique Set 1 (Introduction)	817
Source	823
147 Sqrt (or Square Root) Decomposition Set 2 (LCA of Tree in $O(\sqrt{height})$ time)	824
Source	834
148 String transformation using XOR and OR	835
Source	842
149 Sum of all prime divisors of all the numbers in range L-R	843
Source	850
150 Sum of decimal equivalent of all possible pairs of Binary representation of a Number	851
Source	854
151 Sum of elements in range L-R where first half and second half is filled with odd and even numbers	855
Source	866
152 Sum of elements of all partitions of number such that no element is less than K	867
Source	872
153 Sum of $f(a[i], a[j])$ over all pairs in an array of n integers	873
Source	876
154 Sum of range in a series of first odd then even natural numbers	877
Source	883
155 Test Case Generation Set 1 (Random Numbers, Arrays and Matrices)	884
Source	888
156 Test Case Generation Set 2 (Random Characters, Strings and Arrays of Random Strings)	889

Source	892
157 Test Case Generation Set 3 (Unweighted and Weighted Trees)	893
Source	900
158 Test Case Generation Set 4 (Random directed / undirected weighted and unweighted Graphs)	901
Source	909
159 Test Case Generation Set 5 (Generating random Sorted Arrays and Palindromes)	910
Source	913
160 The Google Foo Bar Challenge	914
161 My Experience with the Google Foo Bar Challenge	915
162 The Challenge	918
Source	920
163 The painter's partition problem Set 2	921
Source	930
164 Tiling with Dominoes	931
Source	937
165 Tips and Tricks for Competitive Programmers Set 1 (For Beginners)	938
Source	940
166 Tips and Tricks for Competitive Programmers Set 2 (Language to be used for Competitive Programming)	941
Source	944
167 Top 10 Algorithms and Data Structures for Competitive Programming	945
Source	949
168 Traversal of tree with k jumps allowed between nodes of same height	950
Source	955
169 Trick for modular division ((x1 * x2 xn) / b) mod (m)	956
Source	960
170 Triplet with no element divisible by 3 and sum N	961
Source	968
171 Two Dimensional Segment Tree Sub-Matrix Sum	969
Source	976
172 Using Chinese Remainder Theorem to Combine Modular equations	977
Source	980

173 Vantieghems Theorem for Primality Test	981
Source	982
174 Variation in Nim Game	983
Source	990
175 Water Connection Problem	991
Source	997
176 Water drop problem	998
Source	1000
177 What coding habits improve timing in coding contest?	1001
Source	1002
178 What to do at the time of Wrong Answer (WA)?	1003
Source	1004
179 Why is programming important for first year or school students?	1005
Source	1007
180 Why is python best suited for Competitive Coding?	1008
Source	1012
181 Writing C/C++ code efficiently in Competitive programming	1013
Source	1018
182 Writing code faster in C++ STL	1019
Source	1020
183 getchar_unlocked() – faster input in C/C++ for Competitive Programming	1021
Source	1022

Chapter 1

A Better Way To Approach Competitive Programming

A Better Way To Approach Competitive Programming - GeeksforGeeks

This article helps to all those who want to begin with Competitive Programming. The only prerequisite one need is the knowledge of a programming language.

Now, Let us find a better approach to Competitive Programming. Please note:

1. One should read the proper Input and Output format because most of the beginners make mistakes of having extra print statements in the output. So please be careful to the output format. **Example** – “Please Enter Next Number :” and “Output is : .
2. Analyze the problem constraints before writing code because most of the time you have written the code that is brute force and will not run in the given time limit constraint.

Following are the some of the issues that you might face if you are a beginner.:

1. **Time Limit** : It is given in seconds. It gives you the insight about the order of the correct solution. Consider following situation. Time Limit = 1 Sec
Let us Assume 1 sec approximately can perform 10^8 operations.
If you write a program that is of order $O(N^2)$ and the problem has T test cases. Then total order of your program becomes $O(T \cdot N^2)$.
If $T \leq 1000$ and $N \leq 1000$, then (ignoring hidden constants in asymptotic notations) your code may not be accepted in worst case as $1000 \cdot 1000 \cdot 1000$ is 10^9 operations which means 10 seconds.
To avoid TLE, always think of the worst test cases that are possible for the problem and analyze your code in that situation.
2. **Run Time Error** : It is the one of the most encountered problem by the beginners. The main reason could be :

- (a) Segmentation Fault : It is the illegal accessing of memory address.
e.g int[] array = new int[100]; System.out.println(array[101]);
 - (b) Declaration of an array of more than 10^8 .
 - (c) Divide & Taking Modulus with 0 .
 - (d) You should know how to use GDB that will help you to correct your run time error.
3. **Compilation Error** : It is one of the errors that is frequently faced when programming in C/C++.
 4. **Wrong Answer** : Whenever you encounter WA, write a brute force code & make sure that it is perfect. Now generate test cases using random function in C++. Run your code on these test cases and match the output. Now think of the corner cases that will help you to find the problem in your algorithm.
 5. **IntOverflow** : Many times unknowingly you will exceed the maximum value that can be stored in primitive type int. e.g. Constraints : $0 < \text{num1}, \text{num2} \leq 10^9$

```
#include<iostream.h>
int main()
{
    int num1, num2;
    cin >> num1 >> num2;
    int answer = num1 + num2;

    // error if num1 and num2 are large
    // numbers
    cout<< answer;

    return 0;
}
```

The above program won't run correct always because ($2 * 10^9$) it may exceed the maximum value that can be stored in INTS. i.e 10^9 . So you will have to use long longint or unsigned int primitive data type for answer in such a case.

6. **Comparing Doubles** :

```
int main()
{
    float a ;
    cin << a;
    if (a == 10)
        cout << "YES";
}
```

float and double data types don't have infinite precision . **Beware** (6/15 digit precision for them respectively). So always use a margin of (~0.0000001) in comparing. Example

—

```
if (abs(a -10) < (0.0000001))
{
    cout << "YES";
}
```

Other Useful Points :

Sometimes when you are stuck. Check running time of other accepted code and analyze about the order of solution is required and the amount of memory that is allowed.

1. 4 MB ~ integer array of size 10^6 (assuming int takes 4 bytes) or 2-d array of size $10^3 \times 10^3$

Standard Memory limits in most of the problem are of order of 256MB.

If you have to allocate large array, then it is NOT a good idea to do allocation inside a function as memory is allocated and released for every test case, and memory is allocated on function call stack (stack size is limited at many places). Thus if you have to make an array of large size, make it global.

I hope this article was of help to you and hope that you will now be able to attempt questions being better prepared and thereby perform better, quicker.

Source

<https://www.geeksforgeeks.org/overcoming-common-problems-in-competitive-programming/>

Chapter 2

AKS Primality Test

AKS Primality Test - GeeksforGeeks

There are several primality test available to check whether the number is prime or not like **Fermat's Theorem**, **Miller-Rabin** Primality test and a lot more. But problem with all of them is they are all probabilistic in nature. So, here comes one another method i.e **AKS primality test (Agrawal-Kayal-Saxena primality test)** and it is **deterministically correct** for any general number.

Features of AKS primality test :

1. The AKS algorithm can be used to verify the primality of any general number given.
2. The maximum running time of the algorithm can be expressed as a polynomial over the number of digits in the target number.
3. The algorithm is guaranteed to distinguish deterministically whether the target number is prime or composite.
4. The correctness of AKS is not conditional on any subsidiary unproven hypothesis.

The AKS primality test is based upon the following theorem: An integer n greater than 2 is prime if and only if the polynomial congruence relation

$$(x + \frac{1}{n})^n \equiv x^n + \frac{1}{n} \pmod{n}$$

holds for some **a coprime to n**. Here x is just a formal symbol .

The AKS test evaluates the equality by making complexity dependent on the size of r . This is expressed as

$$(x + \frac{1}{n})^n \equiv x^n + \frac{1}{n} \pmod{n}$$

which can be expressed in simpler term as

$$(x + \frac{1}{n})^n = (x^n + f(n)) + g(n) \pmod{n}$$

for some polynomials f and g .

This congruence can be checked in polynomial time when r is polynomial to the digits of n . The AKS algorithm evaluates this congruence for a large set of a values, whose size is polynomial to the digits of n . The proof of validity of the AKS algorithm shows that one can find r and a set of a values with the above properties such that if the congruences hold

then n is a power of a prime. The brute force approach would require the expansion of the $(x - a)^n$ polynomial and a reduction (mod n) of the resulting $n + 1$ coefficients.

As a should be co-prime to n . So, to implement this algorithm we can check by taking $a = 1$, but for large values of n we should take large values of a .

The algorithm is based on the condition that if n is any number, then it is prime if,

$(x - 1)^n - (x^n - 1)$ is divisible by n .

Checking for $n = 3$:

$$\begin{aligned} & (x-1)^3 - (x^3 - 1) \\ &= (x^3 - 3x^2 + 3x - 1) - (x^3 - 1) \\ &= -3x^2 + 3x \end{aligned}$$

As all the coefficients are divisible by n i.e. 3, so 3 (n) is prime. As the number increases, size increases.

The code here is based on this condition and can check primes till 64.

Below is the implementation of above approach:

C++

```
// C++ code to check if number is prime. This
// program demonstrates concept behind AKS
// algorithm and doesn't implement the actual
// algorithm (This works only till n = 64)
#include <bits/stdc++.h>
using namespace std;

// array used to store coefficients .
long long c[100];

// function to calculate the coefficients
// of  $(x - 1)^n - (x^n - 1)$  with the help
// of Pascal's triangle .
void coef(int n)
{
    c[0] = 1;
    for (int i = 0; i < n; c[0] = -c[0], i++) {
        c[1 + i] = 1;

        for (int j = i; j > 0; j--)
            c[j] = c[j - 1] - c[j];
    }
}

// function to check whether
// the number is prime or not
bool isPrime(int n)
```

```

{
    // Calculating all the coefficients by
    // the function coef and storing all
    // the coefficients in c array .
    coef(n);

    // subtracting c[n] and adding c[0] by 1
    // as ( x - 1 )^n - ( x^n - 1 ), here we
    // are subtracting c[n] by 1 and adding
    // 1 in expression.
    c[0]++, c[n]--;
}

// checking all the coefficients whether
// they are divisible by n or not.
// if n is not prime, then loop breaks
// and (i > 0).
int i = n;
while (i-- && c[i] % n == 0)
;

// Return true if all coefficients are
// divisible by n.
return i < 0;
}

// driver program
int main()
{
    int n = 37;
    if (isPrime(n))
        cout << "Prime" << endl;
    else
        cout << "Not Prime" << endl;
    return 0;
}

```

Java

```

// Java code to check if number is prime. This
// program demonstrates concept behind AKS
// algorithm and doesn't implement the actual
// algorithm (This works only till n = 64)

class GFG {
    // array used to store coefficients .
    static long c[] = new long[100];

    // function to calculate the coefficients

```

```

// of (x - 1)^n - (x^n - 1) with the help
// of Pascal's triangle .
static void coef(int n)
{
    c[0] = 1;
    for (int i = 0; i < n; c[0] = -c[0], i++) {
        c[1 + i] = 1;

        for (int j = i; j > 0; j--)
            c[j] = c[j - 1] - c[j];
    }
}

// function to check whether
// the number is prime or not
static boolean isPrime(int n)
{
    // Calculating all the coefficients by
    // the function coef and storing all
    // the coefficients in c array .
    coef(n);

    // subtracting c[n] and adding c[0] by 1
    // as ( x - 1 )^n - ( x^n - 1 ), here we
    // are subtracting c[n] by 1 and adding
    // 1 in expression.
    c[0]++;
    c[n]--;

    // checking all the coefficients whether
    // they are divisible by n or not.
    // if n is not prime, then loop breaks
    // and (i > 0).
    int i = n;
    while ((i--) > 0 && c[i] % n == 0)
        ;

    // Return true if all coefficients are
    // divisible by n.
    return i < 0;
}
// Driver code
public static void main(String[] args)
{
    int n = 37;
    if (isPrime(n))
        System.out.println("Prime");
    else

```

```
        System.out.println("Not Prime");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 code to check if
# number is prime. This
# program demonstrates concept
# behind AKS algorithm and
# doesn't implement the actual
# algorithm (This works only
# till n = 64)

# array used to
# store coefficients .
c = [0] * 100;

# function to calculate the
# coefficients of  $(x - 1)^n -$ 
#  $(x^n - 1)$  with the help
# of Pascal's triangle .
def coef(n):
    c[0] = 1;
    for i in range(n):
        c[1 + i] = 1;
        for j in range(i, 0, -1):
            c[j] = c[j - 1] - c[j];
    c[0] = -c[0];

# function to check whether
# the number is prime or not
def isPrime(n):

    # Calculating all the coefficients
    # by the function coef and storing
    # all the coefficients in c array .
    coef(n);

    # subtracting c[n] and adding
    # c[0] by 1 as  $(x - 1)^n -$ 
    #  $(x^n - 1)$ , here we are
    # subtracting c[n] by 1 and
    # adding 1 in expression.
    c[0] = c[0] + 1;
    c[n] = c[n] - 1;
```

```

# checking all the coefficients
# whether they are divisible by
# n or not. if n is not prime,
# then loop breaks and (i > 0).
i = n;
while (i > -1 and c[i] % n == 0):
    i = i - 1;

# Return true if all coefficients
# are divisible by n.
return True if i < 0 else False;

# Driver Code
n = 37;
if (isPrime(n)):
    print("Prime");
else:
    print("Not Prime");

# This code is contributed by mits

```

C#

```

// C# code to check if number is prime. This
// program demonstrates concept behind AKS
// algorithm and doesn't implement the actual
// algorithm (This works only till n = 64)
using System;

class GFG {

    // array used to store coefficients .
    static long []c = new long[100];

    // function to calculate the coefficients
    // of  $(x - 1)^n - (x^n - 1)$  with the help
    // of Pascal's triangle .
    static void coef(int n)
    {
        c[0] = 1;

        for (int i = 0; i < n; c[0] = -c[0], i++)
        {
            c[1 + i] = 1;

            for (int j = i; j > 0; j--)
                c[j] = c[j - 1] - c[j];
        }
    }
}

```

```

        }

    }

    // function to check whether
    // the number is prime or not
    static bool isPrime(int n)
    {

        // Calculating all the coefficients by
        // the function coef and storing all
        // the coefficients in c array .
        coef(n);

        // subtracting c[n] and adding c[0] by 1
        // as ( x - 1 )^n - ( x^n - 1 ), here we
        // are subtracting c[n] by 1 and adding
        // 1 in expression.
        c[0]++;
        c[n]--;

        // checking all the coefficients whether
        // they are divisible by n or not.
        // if n is not prime, then loop breaks
        // and (i > 0).
        int i = n;
        while ((i--) > 0 && c[i] % n == 0)
            ;

        // Return true if all coefficients are
        // divisible by n.
        return i < 0;
    }

    // Driver code
    public static void Main()
    {
        int n = 37;
        if (isPrime(n))
            Console.WriteLine("Prime");
        else
            Console.WriteLine("Not Prime");
    }
}

// This code is contributed by anuj_67.

```

PHP

```

<?php
// PHP code to check if number
// is prime. This program
// demonstrates concept behind
// AKS algorithm and doesn't
// implement the actual
// algorithm (This works only
// till n = 64)

// array used to
// store coefficients .
global $c;

// function to calculate
// the coefficients
// of  $(x - 1)^n -$ 
//  $(x^n - 1)$  with the help
// of Pascal's triangle .
function coef($n)
{
    $c[0] = 1;
    for ($i = 0; $i < $n; $c[0] = -$c[0], $i++)
    {
        $c[1 + $i] = 1;

        for ($j = $i; $j > 0; $j--)
            $c[$j] = $c[$j - 1] - $c[$j];
    }
}

// function to check whether
// the number is prime or not
function isPrime($n)
{
    global $c;

    // Calculating all the
    // coefficients by the
    // function coef and
    // storing all the
    // coefficients in c array .
    coef($n);

    // subtracting c[n] and
    // adding c[0] by 1 as
    //  $(x - 1)^n - (x^n - 1)$ ,
    // here we are subtracting c[n]
    // by 1 and adding 1 in expression.

```

```
// $c[0]++; $c[$n]--;
// checking all the coefficients whether
// they are divisible by n or not.
// if n is not prime, then loop breaks
// and (i > 0).
$i = $n;
while ($i-- && $c[$i] % $n == 0)

// Return true if all
// coefficients are
// divisible by n.
return $i < 0;
}

// Driver Code
$n = 37;
if (isPrime($n))
    echo "Not Prime", "\n";
else
    echo "Prime", "\n";

// This code is contributed by aj_36
?>
```

Output:

Prime

References:

https://en.wikipedia.org/wiki/AKS_primality_test
https://rosettacode.org/wiki/AKS_test_for_primes#C
<https://www.youtube.com/watch?v=HvMSRWTE2mI>

Improved By : [jit_t](#), [vt_m](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/aks-primality-test/>

Chapter 3

Algorithm Library | C++ Magicians STL Algorithm

Algorithm Library | C++ Magicians STL Algorithm - GeeksforGeeks

For all those who aspire to excel in competitive programming, only having a knowledge about containers of STL is of less use till one is not aware what all STL has to offer. STL has an ocean of algorithms, for all < algorithm > library functions : Refer [here](#).

Some of the most used algorithms on vectors and most useful one's in Competitive Programming are mentioned as follows :

Non-Manipulating Algorithms

1. **sort(first_iterator, last_iterator)** – To sort the given vector.
2. **reverse(first_iterator, last_iterator)** – To reverse a vector.
3. ***max_element (first_iterator, last_iterator)** – To find the maximum element of a vector.
4. ***min_element (first_iterator, last_iterator)** – To find the minimum element of a vector.
5. **accumulate(first_iterator, last_iterator, initial value of sum)** – Does the summation of vector elements

```
// A C++ program to demonstrate working of sort(),
// reverse()
#include <algorithm>
#include <iostream>
#include <vector>
#include <numeric> //For accumulate operation
using namespace std;
```

```

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23, 42, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is: ";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // Sorting the Vector in Ascending order
    sort(vect.begin(), vect.end());

    cout << "\nVector after sorting is: ";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // Reversing the Vector
    reverse(vect.begin(), vect.end());

    cout << "\nVector after reversing is: ";
    for (int i=0; i<6; i++)
        cout << vect[i] << " ";

    cout << "\nMaximum element of vector is: ";
    cout << *max_element(vect.begin(), vect.end());

    cout << "\nMinimum element of vector is: ";
    cout << *min_element(vect.begin(), vect.end());

    // Starting the summation from 0
    cout << "\nThe summation of vector elements is: ";
    cout << accumulate(vect.begin(), vect.end(), 0);

    return 0;
}

```

Output:

```

Vector before sorting is: 10 20 5 23 42 15
Vector after sorting is: 5 10 15 20 23 42
Vector before reversing is: 5 10 15 20 23 42
Vector after reversing is: 42 23 20 15 10 5
Maximum element of vector is: 42
Minimum element of vector is: 5
The summation of vector elements is: 115

```

6. **count(first_iterator, last_iterator,x)** – To count the occurrences of x in vector.

7. **find(first_iterator, last_iterator, x)** – Points to last address of vector ((name_of_vector).end()) if element is not present in vector.

```
// C++ program to demonstrate working of count()
// and find()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23, 42, 20, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Occurrences of 20 in vector : ";

    // Counts the occurrences of 20 from 1st to
    // last element
    cout << count(vect.begin(), vect.end(), 20);

    // find() returns iterator to last address if
    // element not present
    find(vect.begin(), vect.end(), 5) != vect.end()?
        cout << "\nElement found":
        cout << "\nElement not found";

    return 0;
}
```

Output:

```
Occurrences of 20 in vector: 2
Element found
```

8. **binary_search(first_iterator, last_iterator, x)** – Tests whether x exists in sorted vector or not.
9. **lower_bound(first_iterator, last_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’.
10. **upper_bound(first_iterator, last_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

```
// C++ program to demonstrate working of lower_bound()
// and upper_bound().
```

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Sort the array to make sure that lower_bound()
    // and upper_bound() work.
    sort(vect.begin(), vect.end());

    // Returns the first occurrence of 20
    auto q = lower_bound(vect.begin(), vect.end(), 20);

    // Returns the last occurrence of 20
    auto p = upper_bound(vect.begin(), vect.end(), 20);

    cout << "The lower bound is at position: ";
    cout << q-vect.begin() << endl;

    cout << "The upper bound is at position: ";
    cout << p-vect.begin() << endl;

    return 0;
}
```

Output:

```
The lower bound is at position: 3
The upper bound is at position: 5
```

Some Manipulating Algorithms

11. **arr.erase(position to be deleted)** – This erases selected element in vector and shifts and resizes the vector elements accordingly.
12. **arr.erase(unique(arr.begin(),arr.end()),arr.end())** – This erases the duplicate occurrences in sorted vector in a single line.

```
// C++ program to demonstrate working of erase()
#include <algorithm>
#include <iostream>
#include <vector>
```

```

using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is :";
    for (int i=0; i<6; i++)
        cout << vect[i]<< " ";

    // Delete second element of vector
    vect.erase(vect.begin()+1);

    cout << "\nVector after erasing the element: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // sorting to enable use of unique()
    sort(vect.begin(), vect.end());

    cout << "\nVector before removing duplicate "
         " occurrences: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // Deletes the duplicate occurrences
    vect.erase(unique(vect.begin(), vect.end()), vect.end());

    cout << "\nVector after deleting duplicates: ";
    for (int i=0; i< vect.size(); i++)
        cout << vect[i] << " ";

    return 0;
}

```

Output:

```

Vector before erasing the element:5 20 5 23 20 20
Vector after erasing the element: 5 5 23 20 20
Vector before removing duplicate occurrences: 5 5 20 20 23
Vector after deleting duplicates: 5 20 23

```

13. **next_permutation(first_iterator, last_iterator)** – This modified the vector to its next permutation.

14. **prev_permutation(first_iterator, last_iterator)** – This modified the vector to its previous permutation.

```
// C++ program to demonstrate working of next_permutation()
// and prev_permutation()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Given Vector is:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // modifies vector to its next permutation order
    next_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing next permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    prev_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing prev permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    return 0;
}
```

Output:

```
Given Vector is:
5 10 15 20 20 23 42 45
Vector after performing next permutation:
5 10 15 20 20 23 45 42
Vector after performing prev permutation:
5 10 15 20 20 23 42 45
```

14. **distance(first_iterator,desired_position)** – It returns the distance of desired position from the first iterator. This function is very useful while finding the index.

```
// C++ program to demonstrate working of distance()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Return distance of first to maximum element
    cout << "Distance between first to max element: ";
    cout << distance(vect.begin(),
                      max_element(vect.begin(), vect.end()));
    return 0;
}
```

Output:

```
Distance between first to max element: 7
```

More – [STL Articles](#)

Source

<https://www.geeksforgeeks.org/c-magicians-stl-algorithms/>

Chapter 4

Array algorithms in C++ STL (`all_of`, `any_of`, `none_of`, `copy_n` and `iota`)

Array algorithms in C++ STL (`all_of`, `any_of`, `none_of`, `copy_n` and `iota`) - GeeksforGeeks

From C++11 onwards, some new and interesting algorithms are added in STL of C++. These algorithms operate on an array and are useful in saving time during coding and hence useful in competitive programming as well.

`all_of()`

This function operates on whole range of array elements and can save time to run a loop to check each elements one by one. It checks for a given property on every element and returns true when each element in range satisfies specified property, else returns false.

```
// C++ code to demonstrate working of all_of()
#include<iostream>
#include<algorithm> // for all_of()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, -6};

    // Checking if all elements are positive
    all_of(ar, ar+6, [](int x) { return x>0; })?
        cout << "All are positive elements" :
        cout << "All are not positive elements";

    return 0;
}
```

}

Output:

```
All are not positive elements
```

In the above code, -6 being a negative element negates the condition and returns false.

any_of()

This function checks for a given range if there's even one element satisfying a given property mentioned in function. Returns true if at least one element satisfies the property else returns false.

```
// C++ code to demonstrate working of any_of()
#include<iostream>
#include<algorithm> // for any_of()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, -6};

    // Checking if any element is negative
    any_of(ar, ar+6, [](int x){ return x<0; })?
        cout << "There exists a negative element" :
        cout << "All are positive elements";

    return 0;
}
```

Output:

```
There exists a negative element
```

In above code, -6 makes the condition positive.

none_of()

This function returns true if none of elements satisfies the given condition else returns false.

```
// C++ code to demonstrate working of none_of()
#include<iostream>
```

```
#include<algorithm> // for none_of()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, 6};

    // Checking if no element is negative
    none_of(ar, ar+6, [](int x){ return x<0; })?
        cout << "No negative elements" :
        cout << "There are negative elements";

    return 0;
}
```

Output:

```
No negative elements
```

Since all elements are positive, the function returns true.

copy_n()

copy_n() copies one array elements to new array. This type of copy creates a deep copy of array. This function takes 3 arguments, source array name, size of array and the target array name.

```
// C++ code to demonstrate working of copy_n()
#include<iostream>
#include<algorithm> // for copy_n()
using namespace std;
int main()
{
    // Initializing array
    int ar[6] = {1, 2, 3, 4, 5, 6};

    // Declaring second array
    int ar1[6];

    // Using copy_n() to copy contents
    copy_n(ar, 6, ar1);

    // Displaying the copied array
    cout << "The new array after copying is : ";
    for (int i=0; i<6 ; i++)
        cout << ar1[i] << " ";
```

```
    return 0;  
}
```

Output:

```
The new array after copying is : 1 2 3 4 5 6
```

In the above code, the elements of ar are copied in ar1 using copy_n()

iota()

This function is used to assign continuous values to array. This function accepts 3 arguments, the array name, size, and the starting number.

```
// C++ code to demonstrate working of iota()  
#include<iostream>  
#include<numeric> // for iota()  
using namespace std;  
int main()  
{  
    // Initializing array with 0 values  
    int ar[6] = {0};  
  
    // Using iota() to assign values  
    iota(ar, ar+6, 20);  
  
    // Displaying the new array  
    cout << "The new array after assigning values is : ";  
    for (int i=0; i<6 ; i++)  
        cout << ar[i] << " ";  
  
    return 0;  
}
```

Output:

```
The new array after assigning values is : 20 21 22 23 24 25
```

In the above code, continuous values are assigned to array using iota().

Improved By : [mohitw16](#)

Source

<https://www.geeksforgeeks.org/useful-array-algorithms-in-c-stl/>

Chapter 5

Array range queries over range queries

Array range queries over range queries - GeeksforGeeks

Given an array of size n and a give set of commands of size m. The commands are enumerated from 1 to m. These commands can be of the following two types of commands:

1. **Type 1** [l r ($1 \leq l \leq r \leq n$)] : Increase all elements of the array by one, whose indices belongs to the range [l, r]. In these queries of the index is inclusive in the range.
2. **Type 2** [l r ($1 \leq l \leq r \leq m$)] : Execute all the commands whose indices are in the range [l, r]. In these queries of the index is inclusive in the range. It's guaranteed that r is strictly less than the enumeration/number of the current command.

Note : The array indexing is from 1 as per the problem statement.

Example 1

```
Input : 5 5
        1 1 2
        1 4 5
        2 1 2
        2 1 3
        2 3 4
Output : 7 7 0 7 7
```

Explanation of Example 1 :

Our array initially is of size 5 whose each element has been initialized to 0. So now the question states that we have 5 queries for the above example.

1. Query 1 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 2 so after the execution of the first our array turns down to be 1 1 0 0 0 .
2. Query 2 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 4 and 5 so after the execution of the first our array turns down to be 1 1 0 1 1 .
3. Query 3 is of type 2 : As stated in the definition of this type of query we will execute the queries stated in the range i.e. we will operate the queries instead of the array. The range given is 1 and 2 so we will execute queries 1 and 2 again i.e. we will use repetitive approach for the type 2 queries so we will execute query 1 again and our array will be 2 2 0 1 1 . Now when we execute the query we will execute query 2 and our resultant array will be 2 2 0 2 2 .
4. Query 4 is of type 2 : As stated in the definition of this type of query we will execute the queries stated in the range i.e. we will operate the queries instead of the array. The range given is 1 and 3 so we will execute queries 1, 2 and 3 again i.e. using repetitive approach queries 1, 2 and 3 will be executed. After the execution of the query 1 again the array will be 3 3 0 2 2 . After the execution of the query 2 again the array will be 3 3 0 3 3 . Now due to query 3 inclusive in the range we will execute query 3 the resultant array will be 4 4 0 4 4 . As explained above.
5. Query 5 is of type 2 : The last query will execute the 3rd and 4th query which has been explained above. After the execution of the 3rd query our array will be 5 5 0 5 5 . And after the execution of the 4th query i.e. execution of query 1, 2 and 3 our array will be 7 7 0 7 7 The above is the desired result

Example 2

```
Input : 1 2
        1 1 1
        1 1 1
Output : 2
```

Explanation of the example 2:

Our array initially is of size 1 whose each element has been initialized to 0. So now the question states that we have 2 queries for the above example.

1. Query 1 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 1 so after the execution of the first our array turns down to be 1 .
2. Query 2 is of type 1 : As stated above we will simply increment the array indices by 1 the given indices are 1 and 1 so after the execution of the first our array turns down to be 2 . This gives us the desired result

Method 1 :

This method is the brute force method where by simple recursion is applied on the type 2 queries and for type 1 queries simple increment in the array index is performed.

```

// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
using namespace std;

// Function to execute type 1 query
void type1(int arr[], int start, int limit)
{
    // incrementing the array by 1 for type
    // 1 queries
    for (int i = start; i <= limit; i++)
        arr[i]++;
}

// Function to execute type 2 query
void type2(int arr[], int query[][3], int start, int limit)
{
    for (int i = start; i <= limit; i++) {

        // If the query is of type 1 function
        // call to type 1 query
        if (query[i][0] == 1)
            type1(arr, query[i][1], query[i][2]);

        // If the query is of type 2 recursive call
        // to type 2 query
        else if (query[i][0] == 2)
            type2(arr, query, query[i][1], query[i][2]);
    }
}

// Driver code
int main()
{
    // Input size of array and number of queries
    int n = 5, m = 5;
    int arr[n + 1];
    for (int i = 1; i <= n; i++)
        arr[i] = 0;

    // Build query matrix
    int temp[15] = { 1, 1, 2, 1, 4, 5, 2,
                    1, 2, 2, 1, 3, 2, 3, 4 };
    int query[5][3];
    int j = 0;
    for (int i = 1; i <= m; i++) {
        query[i][0] = temp[j++];
        query[i][1] = temp[j++];
    }
}

```

```

        query[i][2] = temp[j++];
    }

    // Perform queries
    for (int i = 1; i <= m; i++)
        if (query[i][0] == 1)
            type1(arr, query[i][1], query[i][2]);
        else if (query[i][0] == 2)
            type2(arr, query, query[i][1], query[i][2]);

    // printing the result
    for (int i = 1; i <= n; i++)
        cout << arr[i] << " ";

    return 0;
}

```

Output:

7 7 0 7 7

The Time complexity of the above code is $O(2^m)$

Method 2 :

In this method we use an extra array for creating the record array to find the number of time a particular query is being executed and after creating the record array we simply execute the queries of type 1 and the contains of the record array is simply added to the main array the and this would give us the resultant array.

```

// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
using namespace std;

// Function to create the record array
void record_sum(int record[], int l, int r,
                int n, int adder)
{
    for (int i = l; i <= r; i++)
        record[i] += adder;
}

// Driver Code
int main()
{
    int n = 5, m = 5;
    int arr[n];

```

```

// Build query matrix
memset(arr, 0, sizeof arr);
int query[5][3] = { { 1, 1, 2 }, { 1, 4, 5 },
                    { 2, 1, 2 }, { 2, 1, 3 },
                    { 2, 3, 4 } };
int record[m];
memset(record, 0, sizeof record);

for (int i = m - 1; i >= 0; i--) {

    // If query is of type 2 then function
    // call to record_sum
    if (query[i][0] == 2)
        record_sum(record, query[i][1] - 1,
                   query[i][2] - 1, m, record[i] + 1);

    // If query is of type 1 then simply add
    // 1 to the record array
    else
        record_sum(record, i, i, m, 1);

}

// for type 1 queries adding the contains of
// record array to the main array record array
for (int i = 0; i < m; i++) {
    if (query[i][0] == 1)
        record_sum(arr, query[i][1] - 1,
                   query[i][2] - 1, n, record[i]);
}

// printing the array
for (int i = 0; i < n; i++)
    cout << arr[i] << ' ';

return 0;
}

```

Output :

7 7 0 7 7

The Time complexity of the above code is $O(n^2)$

Method 3 :

This method has been made more efficient by applying [square root decomposition](#) to the record array.

```

// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
#define max 10000
using namespace std;

// For prefix sum array
void update(int arr[], int l)
{
    arr[l] += arr[l - 1];
}

// This function is used to apply square root
// decomposition in the record array
void record_func(int block_size, int block[],
                 int record[], int l, int r, int value)
{
    // traversing first block in range
    while (l < r && l % block_size != 0 && l != 0) {
        record[l] += value;
        l++;
    }
    // traversing completely overlapped blocks in range
    while (l + block_size <= r + 1) {
        block[l / block_size] += value;
        l += block_size;
    }
    // traversing last block in range
    while (l <= r) {
        record[l] += value;
        l++;
    }
}
// Function to print the resultant array
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int n = 5, m = 5;
    int arr[n], record[m];
    int block_size = sqrt(m);
    int block[max];
    int command[5][3] = { { 1, 1, 2 }, { 1, 4, 5 },

```

```

        { 2, 1, 2 }, { 2, 1, 3 },
        { 2, 3, 4 } };
memset(arr, 0, sizeof arr);
memset(record, 0, sizeof record);
memset(block, 0, sizeof block);

for (int i = m - 1; i >= 0; i--) {

    // If query is of type 2 then function
    // call to record_func
    if (command[i][0] == 2) {
        int x = i / (block_size);
        record_func(block_size, block, record,
                    command[i][1] - 1, command[i][2] - 1,
                    (block[x] + record[i] + 1));
    }
    // If query is of type 1 then simply add
    // 1 to the record array
    else
        record[i]++;
}

// Merging the value of the block in the record array
for (int i = 0; i < m; i++) {
    int check = (i / block_size);
    record[i] += block[check];
}

for (int i = 0; i < m; i++) {
    // If query is of type 1 then the array
    // elements are over-written by the record
    // array
    if (command[i][0] == 1) {
        arr[command[i][1] - 1] += record[i];
        if ((command[i][2] - 1) < n - 1)
            arr[(command[i][2])] -= record[i];
    }
}

// The prefix sum of the array
for (int i = 1; i < n; i++)
    update(arr, i);

// Printing the resultant array
print(arr, n);
return 0;
}

```

Output :

7 7 0 7 7

Method 4 :

This method has been made more efficient by applying [Binary Indexed Tree](#) or Fenwick Tree by creating two binary indexed tree for query 1 and query 2 respectively.

```
// CPP program to perform range queries over range
// queries.
#include <bits/stdc++.h>
using namespace std;

// Updates a node in Binary Index Tree (BITree) at given index
// in BITree. The given value 'val' is added to BITree[i] and
// all of its ancestors in tree.
void updateBIT(int BITree[], int n, int index, int val)
{
    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse all ancestors and add 'val'
    while (index <= n) {

        // Add 'val' to current node of BI Tree
        BITree[index] = (val + BITree[index]);

        // Update index to that of parent in update View
        index = (index + (index & (-index)));
    }
    return;
}

// Constructs and returns a Binary Indexed Tree for given
// array of size n.
int* constructBITree(int n)
{
    // Create and initialize BITree[] as 0
    int* BITree = new int[n + 1];
    for (int i = 1; i <= n; i++)
        BITree[i] = 0;

    return BITree;
}

// Returns sum of arr[0..index]. This function assumes
// that the array is preprocessed and partial sums of
// array elements are stored in BITree[]
```

```

int getSum(int BITree[], int index)
{
    int sum = 0;
    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse ancestors of BITree[index]
    while (index > 0) {

        // Add element of BITree to sum
        sum = (sum + BITree[index]);

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
}

// Function to update the BITree
void update(int BITree[], int l, int r, int n, int val)
{
    updateBIT(BITree, n, l, val);
    updateBIT(BITree, n, r + 1, -val);
    return;
}

// Driver code
int main()
{
    int n = 5, m = 5;
    int temp[15] = { 1, 1, 2, 1, 4, 5, 2, 1, 2,
                    2, 1, 3, 2, 3, 4 };
    int q[5][3];
    int j = 0;
    for (int i = 1; i <= m; i++) {
        q[i][0] = temp[j++];
        q[i][1] = temp[j++];
        q[i][2] = temp[j++];
    }

    // BITree for query of type 2
    int* BITree = constructBITree(m);

    // BITree for query of type 1
    int* BITree2 = constructBITree(n);

    // Input the queries in a 2D matrix
    for (int i = 1; i <= m; i++)

```

```

cin >> q[i][0] >> q[i][1] >> q[i][2];

// If query is of type 2 then function call
// to update with BITree
for (int i = m; i >= 1; i--)
    if (q[i][0] == 2)
        update(BITree, q[i][1] - 1, q[i][2] - 1, m, 1);

for (int i = m; i >= 1; i--) {
    if (q[i][0] == 2) {
        long int val = getSum(BITree, i - 1);
        update(BITree, q[i][1] - 1, q[i][2] - 1, m, val);
    }
}

// If query is of type 1 then function call
// to update with BITree2
for (int i = m; i >= 1; i--) {
    if (q[i][0] == 1) {
        long int val = getSum(BITree, i - 1);
        update(BITree2, q[i][1] - 1, q[i][2] - 1,
               n, (val + 1));
    }
}

for (int i = 1; i <= n; i++)
    cout << (getSum(BITree2, i - 1)) << " ";

return 0;
}

```

Output :

7 7 0 7 7

The Time complexity of Method 3 and Method 4 is $O(\log n)$.

Source

<https://www.geeksforgeeks.org/array-range-queries-range-queries/>

Chapter 6

Array with GCD of any of its subset belongs to the given array

Array with GCD of any of its subset belongs to the given array - GeeksforGeeks

Given a set of N elements such that $N \leq 10^5$, task is to generate an array such that the GCD of any subset of the generated array lies in the given set of elements. The generated array should **not be more than thrice the length of the set of the GCD**.

Prerequisite : [GCD of an Array](#) | [Subset of Array](#)

Examples :

Input : 3
1 2 7
Output : 1 1 2 1 7

Input : 4
2 4 6 12
Output : 4 6 12

Input : 5
2 5 6 7 11
Output : No array can be build

Explanation :

Calculate the [GCD of an array](#) or in this case a set. Now, first sort the given set of GCD. If the GCD of this set is equal to the minimum number of the given set, then just by putting this GCD between each number. But, if this GCD is not the minimum element of the given

set, then unfortunately “no array can be build”.

C++

```
// C++ implementation to generate the
// required array
#include <bits/stdc++.h>
using namespace std;

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to find gcd of
// array of numbers
int findGCD(vector<int> arr, int n)
{
    int result = arr[0];
    for (int i = 1; i < n; i++)
        result = gcd(arr[i], result);
    return result;
}

// Function to generate the array
// with required constraints.
void compute(vector<int> arr, int n)
{
    vector<int> answer;

    // computing GCD of the given set
    int GCD_of_array = findGCD(arr, n);

    // Solution exists if GCD of array is equal
    // to the minimum element of the array
    if(GCD_of_array == arr[0])
    {
        answer.push_back(arr[0]);
        for(int i = 1; i < n; i++)
        {
            answer.push_back(arr[0]);
            answer.push_back(arr[i]);
        }
    }

    // Printing the built array
```

```
        for (int i = 0; i < answer.size(); i++)
            cout << answer[i] << " ";
    }
    else
        cout << "No array can be build";
}

// Driver function
int main()
{

    // Taking in the input and initializing
    // the set STL set in cpp has a property
    // that it maintains the elements in
    // sorted order, thus we do not need
    // to sort them externally
    int n = 3;
    int input[] = {2, 5, 6, 7, 11};
    set<int> GCD(input, input + n);
    vector<int> arr;
    set<int>::iterator it;

    for(it = GCD.begin(); it != GCD.end(); ++it)
        arr.push_back(*it);

    // Calling the computing function.
    compute(arr, n);

    return 0;
}
```

Java

```
// Java implementation
// to generate the
// required array
import java.io.*;
import java.util.*;

class GFG
{
// Function to return
// gcd of a and b
static int gcd(int a,
              int b)
{
    if (a == 0)
        return b;
```

```
        return gcd(b % a, a);
    }

    // Function to find gcd
    // of array of numbers
    public static int findGCD(ArrayList<Integer>
                                arr, int n)
    {
        int result = arr.get(0);
        for (int i = 1; i < n; i++)
            result = gcd(arr.get(i),
                         result);
        return result;
    }

    // Function to generate
    // the array with required
    // constraints.
    public static void compute(ArrayList<Integer>
                                arr, int n)
    {
        ArrayList<Integer> answer =
            new ArrayList<Integer>();

        // computing GCD of
        // the given set
        int GCD_of_array = findGCD(arr, n);

        // Solution exists if GCD
        // of array is equal to the
        // minimum element of the array
        if(GCD_of_array == arr.get(0))
        {
            answer.add(arr.get(0));
            for(int i = 1; i < n; i++)
            {
                answer.add(arr.get(0));
                answer.add(arr.get(i));
            }
        }

        // Printing the
        // built array
        for (int i = 0;
             i < answer.size(); i++)
            System.out.print(answer.get(i) + " ");
    }
    else
        System.out.print("No array " +
```

```
        "can be build");
}

// Driver Code
public static void main(String args[])
{

    // Taking in the input and
    // initializing the set STL
    // set in cpp has a property
    // that it maintains the
    // elements in sorted order,
    // thus we do not need to
    // sort them externally
    int n = 3;
    Integer input[] = {2, 5, 6, 7, 11};
    HashSet<Integer> GCD = new HashSet<Integer>
        (Arrays.asList(input));
    ArrayList<Integer> arr =
        new ArrayList<Integer>();

    for (int v : GCD)
        arr.add(v);

    // Calling the
    // computing function.
    compute(arr, n);
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
// C# implementation
// to generate the
// required array
using System;
using System.Collections.Generic;

class GFG
{
    // Function to return
    // gcd of a and b
    static int gcd(int a, int b)
    {
        if (a == 0)
```

```
        return b;
        return gcd(b % a, a);
    }

    // Function to find gcd
    // of array of numbers
    static int findGCD(List<int> arr,
                        int n)
    {
        int result = arr[0];
        for (int i = 1; i < n; i++)
            result = gcd(arr[i],
                         result);
        return result;
    }

    // Function to generate
    // the array with required
    // constraints.
    static void compute(List<int> arr,
                        int n)
    {
        List<int> answer = new List<int>();

        // computing GCD of
        // the given set
        int GCD_of_array = findGCD(arr, n);

        // Solution exists if GCD
        // of array is equal to the
        // minimum element of the array
        if(GCD_of_array == arr[0])
        {
            answer.Add(arr[0]);
            for(int i = 1; i < n; i++)
            {
                answer.Add(arr[0]);
                answer.Add(arr[i]);
            }

            // Printing the
            // built array
            for (int i = 0; i < answer.Count; i++)
                Console.Write(answer[i] + " ");
        }
        else
            Console.WriteLine("No array " +
                            "can be build");
    }
}
```

```
}

// Driver Code
static void Main()
{
    // Taking in the input and
    // initializing the set STL
    // set in cpp has a property
    // that it maintains the
    // elements in sorted order,
    // thus we do not need to
    // sort them externally
    int n = 3;
    int []input= new int[]{2, 5, 6, 7, 11};
    HashSet<int> GCD = new HashSet<int>(input);
    List<int> arr = new List<int>();

    foreach (int b in GCD)
        arr.Add(b);

    // Calling the
    // computing function.
    compute(arr, n);
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

No array can be build

Time Complexity : $O(n \log(n))$, where n is the size of array given.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/array-gcd-subset-belongs-given-array/>

Chapter 7

BFS using STL for competitive coding

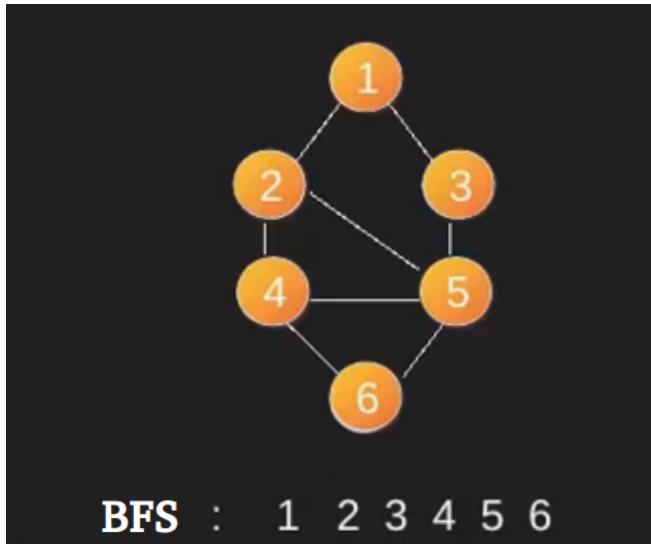
BFS using STL for competitive coding - GeeksforGeeks

A STL based simple implementation of BFS using `queue` and `vector` in STL. The adjacency list is represented using vectors of vector.

In BFS, we start with a node.

- 1) Create a queue and enqueue source into it.
Mark source as visited.
- 2) While queue is not empty, do following
 - a) Dequeue a vertex from queue. Let this be f.
 - b) Print f
 - c) Enqueue all not yet visited adjacent of f and mark them visited.

Below is an example BFS starting from source vertex 1. Note that there can be multiple BFSs possible for a graph (even from a particular vertex).



For more details of BFS, refer this post .

The code here is simplified such that it could be used in competitive coding.

```
// A Quick implementation of BFS using
// vectors and queue
#include <bits/stdc++.h>
#define pb push_back

using namespace std;

vector<bool> v;
vector<vector<int>> g;

void edge(int a, int b)
{
    g[a].pb(b);

    // for undirected graph add this line
    // g[b].pb(a);
}

void bfs(int u)
{
    queue<int> q;

    q.push(u);
    v[u] = true;

    while (!q.empty()) {
```

```

int f = q.front();
q.pop();

cout << f << " ";

// Enqueue all adjacent of f and mark them visited
for (auto i = g[f].begin(); i != g[f].end(); i++) {
    if (!v[*i]) {
        q.push(*i);
        v[*i] = true;
    }
}
}

// Driver code
int main()
{
    int n, e;
    cin >> n >> e;

    v.assign(n, false);
    g.assign(n, vector<int>());

    int a, b;
    for (int i = 0; i < e; i++) {
        cin >> a >> b;
        edge(a, b);
    }

    for (int i = 0; i < n; i++) {
        if (!v[i])
            bfs(i);
    }

    return 0;
}

```

Input:

```

8 10
0 1
0 2
0 3
0 4
1 5
2 5
3 6

```

```
4 6  
5 7  
6 7
```

Output:
0 1 2 3 4 5 6 7

Source

<https://www.geeksforgeeks.org/bfs-using-stl-competitive-coding/>

Chapter 8

Bit Tricks for Competitive Programming

Bit Tricks for Competitive Programming - GeeksforGeeks

In competitive programming or in general some problems seems difficult but can be solved very easily with little bit magic. We have discussed some tricks in below previous post.

[Bitwise Hacks for Competitive Programming](#)

We have considered below facts in this article –

- 0 based indexing of bits from left to right.
- Setting i-th bit means, turning i-th bit to 1
- Clearing i-th bit means, turning i-th bit to 0

1) Clear all bits from LSB to ith bit

```
mask = ~((1 << i+1 ) - 1);
x &= mask;
```

Logic: To clear all bits from LSB to i-th bit, we have to AND x with mask having LSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0. Now we can simply take complement of mask to get all first i bits to 0 and remaining to 1.

Example-

x = 29 (00011101) and we want to clear LSB to 3rd bit, total 4 bits
mask -> 1 << 4 -> 16(00010000)
mask -> 16 - 1 -> 15(00001111)
mask -> ~mask -> 11110000
x & mask -> 16 (00010000)

2) Clearing all bits from MSB to i-th bit

```
mask = (1 << i) - 1;
x &= mask;
```

Logic: To clear all bits from MSB to i-th bit, we have to AND x with mask having MSB to i-th bit 0. To obtain such mask, first left shift 1 i times. Now if we minus 1 from that, all the bits from 0 to i-1 become 1 and remaining bits become 0.

Example-

x = 215 (11010111) and we want to clear MSB to 4th bit, total 4 bits
 mask -> 1 << 4 -> 16(00010000)
 mask -> 16 - 1 -> 15(00001111)
 x & mask -> 7(00000111)

3) Divide by 2

```
x >>= 1;
```

Logic: When we do arithmetic right shift, every bit is shifted to right and blank position is substituted with sign bit of number, 0 in case of positive and 1 in case of negative number. Since every bit is a power of 2, with each shift we are reducing the value of each bit by factor of 2 which is equivalent to division of x by 2.

Example-

x = 18(00010010)
 x >> 1 = 9 (00001001)

4) Multiplying by 2

```
x <<= 1;
```

Logic: When we do arithmetic left shift, every bit is shifted to left and blank position is substituted with 0 . Since every bit is a power of 2, with each shift we are increasing the value of each bit by a factor of 2 which is equivalent to multiplication of x by 2.

Example-

x = 18(00010010)
 x << 1 = 36 (00100100)

5) Upper case English alphabet to lower case

```
ch |= ' ';
```

Logic: The bit representation of upper case and lower case English alphabets are –

A -> 01000001 a -> 01100001

B -> 01000010	b -> 01100010
C -> 01000011	c -> 01100011
.	.
.	.
Z -> 01011010	z -> 01111010

As we can see if we set 5th bit of upper case characters, it will be converted into lower case character. We have to prepare a mask having 5th bit 1 and other 0 (00100000). This mask is bit representation of space character (' '). The character 'ch' then ORed with mask.

Example-

```
ch = 'A' (01000001)
mask = ' ' (00100000)
ch | mask = 'a' (01100001)
```

Please refer [Case conversion \(Lower to Upper and Vice Versa\)](#) for details.

6) Lower case English alphabet to upper case

```
ch &= '_';
```

Logic: The bit representation of upper case and lower case English alphabets are –

A -> 01000001	a -> 01100001
B -> 01000010	b -> 01100010
C -> 01000011	c -> 01100011
.	.
.	.
Z -> 01011010	z -> 01111010

As we can see if we clear 5th bit of lower case characters, it will be converted into upper case character. We have to prepare a mask having 5th bit 0 and other 1 (10111111). This mask is bit representation of underscore character ('_'). The character 'ch' then AND with mask.

Example-

```
ch = 'a' (01100001)
mask = '_' (11011111)
ch | mask = 'A' (01000001)
```

Please refer [Case conversion \(Lower to Upper and Vice Versa\)](#) for details.

7) Count set bits in integer

```
int countSetBits(int x)
{
    int count = 0;
    while (x)
    {
```

```
    x &= (x-1);
    count++;
}
return count;
}
```

Logic: This is [Brian Kernighan's algorithm](#).

8) Find log base 2 of 32 bit integer

```
int log2(int x)
{
    int res = 0;
    while (x >= 1)
        res++;
    return res;
}
```

Logic: We right shift x repeatedly until it becomes 0, meanwhile we keep count on the shift operation. This count value is the $\log_2(x)$.

9) Checking if given 32 bit integer is power of 2

```
int isPowerof2(int x)
{
    return (x && !(x & x-1));
}
```

Logic: All the power of 2 have only single bit set e.g. 16 (00010000). If we minus 1 from this, all the bits from LSB to set bit get toggled, i.e., $16-1 = 15$ (00001111). Now if we AND x with $(x-1)$ and the result is 0 then we can say that x is power of 2 otherwise not. We have to take extra care when $x = 0$.

Example

$x = 16(00010000)$
 $x - 1 = 15(00001111)$
 $x \& (x-1) = 0$
so 16 is power of 2

Please refer [this](#) article for more bit hacks.

Source

<https://www.geeksforgeeks.org/bit-tricks-competitive-programming/>

Chapter 9

Bits manipulation (Important tactics)

Bits manipulation (Important tactics) - GeeksforGeeks

Prerequisites : [Bitwise operators in C](#), [Bitwise Hacks for Competitive Programming](#), [Bit Tricks for Competitive Programming](#)

1. Compute XOR from 1 to n (direct method) :

```
// Direct XOR of all numbers from 1 to n
int computeXOR(int n)
{
    if (n % 4 == 0)
        return n;
    if (n % 4 == 1)
        return 1;
    if (n % 4 == 2)
        return n + 1;
    else
        return 0;
}
```

Input: 6

Output: 7

Refer [Compute XOR from 1 to n](#) for details.

2. We can quickly calculate the total number of combinations with numbers smaller than or equal to with a number whose sum and XOR are equal. Instead of using looping (Brute force method), we can directly find it by a mathematical trick i.e.

```
// Refer Equal Sum and XOR for details.  
Answer = pow(2, count of zero bits)
```

3. How to know if a number is a power of 2?

```
// Function to check if x is power of 2  
bool isPowerOfTwo(int x)  
{  
    // First x in the below expression is  
    // for the case when x is 0  
    return x && !(x & (x - 1));  
}
```

Refer [check if a number is power of two](#) for details.

4. Find XOR of all subsets of a set. We can do it in O(1) time. The answer is always 0 if given set has more than one elements. For set with single element, the answer is value of single element. Refer [XOR of the XOR's of all subsets](#) for details.
5. We can quickly find number of leading, trailing zeroes and number of 1's in a binary code of an integer in C++ using GCC. It can be done by using inbuilt function i.e.

```
Number of leading zeroes: builtin_clz(x)  
Number of trailing zeroes : builtin_ctz(x)  
Number of 1-bits: __builtin_popcount(x)
```

Refer [GCC inbuilt functions](#) for details.

6. Convert binary code directly into an integer in C++.

```
// Conversion into Binary code//  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    auto number = 0b011;  
    cout << number;  
    return 0;  
}
```

Output: 3

7. The Quickest way to swap two numbers:

```
a ^= b;  
b ^= a;  
a ^= b;
```

Refer [swap two numbers](#)for details.

8. Simple approach to flip the bits of a number: It can be done by a simple way, just simply subtract the number from the value obtained when all the bits are equal to 1 . For example:

```
Number : Given Number  
Value : A number with all bits set in given number.  
Flipped number = Value - Number.
```

```
Example :  
Number = 23,  
Binary form: 10111;  
After flipping digits number will be: 01000;  
Value: 11111 = 31;
```

9. We can find the most significant set bit in O(1) time for a fixed size integer. For example below cocde is for 32 bit integer.

```
int setBitNumber(int n)  
{  
    // Below steps set bits after  
    // MSB (including MSB)  
  
    // Suppose n is 273 (binary  
    // is 100010001). It does following  
    // 100010001 | 010001000 = 110011001  
    n |= n>>1;  
  
    // This makes sure 4 bits  
    // (From MSB and including MSB)  
    // are set. It does following  
    // 110011001 | 001100110 = 111111111  
    n |= n>>2;  
  
    n |= n>>4;  
    n |= n>>8;  
    n |= n>>16;  
  
    // Increment n by 1 so that  
    // there is only one set bit  
    // which is just before original  
    // MSB. n now becomes 1000000000
```

```
n = n + 1;

// Return original MSB after shifting.
// n now becomes 100000000
return (n >> 1);
}
```

Refer [Find most significant set bit of a number](#) for details.

10. We can quickly check if bits in a number are in alternate pattern (like 101010). We compute $n \wedge (n >> 1)$. If n has an alternate pattern, then $n \wedge (n >> 1)$ operation will produce a number having set bits only. ' \wedge ' is a bitwise XOR operation. Refer [check if a number has bits in alternate pattern](#) for details.

Source

<https://www.geeksforgeeks.org/bits-manipulation-important-tactics/>

Chapter 10

Bitwise Hacks for Competitive Programming

Bitwise Hacks for Competitive Programming - GeeksforGeeks

It is recommended to refer [Interesting facts about Bitwise Operators](#) as a prerequisite.

1. How to set a bit in the number ‘num’ :

If we want to set a bit at nth position in number ‘num’ ,it can be done using ‘OR’ operator(|).

- First we left shift ‘1’ to n position via ($1 << n$)
- Then, use ‘OR’ operator to set bit at that position.’OR’ operator is used because it will set the bit even if the bit is unset previously in binary representation of number ‘num’.

```
#include<iostream>
using namespace std;
// num is the number and pos is the position
// at which we want to set the bit.
void set(int & num,int pos)
{
    // First step is shift '1', second
    // step is bitwise OR
    num |= (1 << pos);
}
int main()
{
    int num = 4, pos = 1;
    set(num, pos);
    cout << (int)(num) << endl;
    return 0;
}
```

Output:

6

We have passed the parameter by ‘call by reference’ to make permanent changes in the number.

2. How to unset/clear a bit at n'th position in the number ‘num’ :

Suppose we want to unset a bit at nth position in number ‘num’ then we have to do this with the help of ‘AND’ (&) operator.

- First we left shift ‘1’ to n position via ($1 \ll n$) than we use bitwise NOT operator ‘~’ to unset this shifted ‘1’.
- Now after clearing this left shifted ‘1’ i.e making it to ‘0’ we will ‘AND’(&) with the number ‘num’ that will unset bit at nth position position.

```
#include <iostream>
using namespace std;
// First step is to get a number that has all 1's except the given position.
void unset(int &num,int pos)
{
    //Second step is to bitwise and this number with given number
    num &= ~(1 << pos);
}
int main()
{
    int num = 7;
    int pos = 1;
    unset(num, pos);
    cout << num << endl;
    return 0;
}
```

Output:

5

3. Toggling a bit at nth position :

Toggling means to turn bit ‘on’(1) if it was ‘off’(0) and to turn ‘off’(0) if it was ‘on’(1) previously. We will be using ‘XOR’ operator here which is this ‘^’. The reason behind ‘XOR’ operator is because of its properties.

- Properties of ‘XOR’ operator.
 - $1 \hat{\wedge} 1 = 0$
 - $0 \hat{\wedge} 0 = 0$

$$\begin{aligned}- 1 \hat{\wedge} 0 &= 1 \\ - 0 \hat{\wedge} 1 &= 1\end{aligned}$$

- If two bits are different then ‘XOR’ operator returns a set bit(1) else it returns an unset bit(0).

```
#include <iostream>
using namespace std;
// First step is to shift 1, Second step is to XOR with given number
void toggle(int &num, int pos)
{
    num ^= (1 << pos);
}
int main()
{
    int num = 4;
    int pos = 1;
    toggle(num, pos);
    cout << num << endl;
    return 0;
}
```

Output:

6

4. Checking if bit at nth position is set or unset:

It is quite easily doable using ‘AND’ operator.

- Left shift ‘1’ to given position and then ‘AND’(&).

```
#include <iostream>
using namespace std;

bool at_position(int num, int pos)
{
    bool bit = num & (1<<pos);
    return bit;
}

int main()
{
    int num = 5;
    int pos = 0;
    bool bit = at_position(num, pos);
    cout << bit << endl;
    return 0;
}
```

Output:

1

Observe that we have first left shifted ‘1’ and then used ‘AND’ operator to get bit at that position. So if there is ‘1’ at position ‘pos’ in ‘num’, then after ‘AND’ our variable ‘bit’ will store ‘1’ else if there is ‘0’ at position ‘pos’ in the number ‘num’ than after ‘AND’ our variable bit will store ‘0’.

Some more quick hacks:

- **Inverting every bit of a number/1's complement:**

If we want to invert every bit of a number i.e change bit ‘0’ to ‘1’ and bit ‘1’ to ‘0’.We can do this with the help of ‘~’ operator. For example : if number is num=00101100 (binary representation) so ‘~num’ will be ‘11010011’.

This is also the ‘1s complement of number’.

```
#include <iostream>
using namespace std;
int main()
{
    int num = 4;

    // Inverting every bit of number num
    cout << (~num);
    return 0;
}
```

Output:

-5

- **Two's complement of the number:** 2's complement of a number is 1's complement + 1.

So formally we can have 2's complement by finding 1s complement and adding 1 to the result i.e ($\sim\text{num}+1$) or what else we can do is using ‘-‘ operator.

```
#include <iostream>
using namespace std;
int main()
{
    int num = 4;
    int twos_complement = -num;
    cout << "This is two's complement " << twos_complement << endl;
    cout << "This is also two's complement " << (~num+1) << endl;
    return 0;
}
```

Output:

```
This is two's complement -4
This is also two's complement -4
```

- **Stripping off the lowest set bit :**

In many situations we want to strip off the lowest set bit for example in Binary Indexed tree data structure, counting number of set bit in a number.

We do something like this:

```
X = X & (X-1)
```

But how does it even work ?

Let us see this by taking an example, let $X = 1100$.

$(X-1)$ inverts all the bits till it encounter lowest set ‘1’ and it also invert that lowest set ‘1’.
 $X-1$ becomes 1011. After ‘ANDing’ X with $X-1$ we get lowest set bit stripped.

```
#include <iostream>
using namespace std;
void strip_last_set_bit(int &num)
{
    num = num & (num-1);
}
int main()
{
    int num = 7;
    strip_last_set_bit(num);
    cout << num << endl;
    return 0;
}
```

Output:

6

- **Getting lowest set bit of a number:**

This is done by using expression ‘ $X \& (-X)$ ’ Let us see this by taking an example: Let $X = 00101100$. So $\sim X$ (1’s complement) will be ‘11010011’ and 2’s complement will be $(\sim X + 1)$ or $-X$ i.e ‘11010100’. So if we ‘AND’ original number ‘ X ’ with its two’s complement which is ‘ $-X$ ’, we get lowest set bit.

```
00101100  
& 11010100  
-----  
00000100
```

```
#include <iostream>  
using namespace std;  
int lowest_set_bit(int num)  
{  
    int ret = num & (-num);  
    return ret;  
}  
int main()  
{  
    int num = 10;  
    int ans = lowest_set_bit(num);  
    cout << ans << endl;  
    return 0;  
}
```

Output:

2

[Bit Tricks for Competitive Programming](#)

Refer [BitWise Operators Articles](#) for more articles on Bit Hacks.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/bitwise-hacks-for-competitive-programming/>

Chapter 11

Burst Balloon to maximize coins

Burst Balloon to maximize coins - GeeksforGeeks

We have been given N balloons, each with a number of coins associated with it. On bursting a balloon i, the number of coins gained is equal to $A[i-1]*A[i]*A[i+1]$. Also, balloons i-1 and i+1 now become adjacent. Find the maximum possible profit earned after bursting all the balloons. Assume an extra 1 at each boundary.

Examples:

```
Input : 5, 10
Output : 60
Explanation - First Burst 5, Coins = 1*5*10
            Then burst 10, Coins+= 1*10*1
            Total = 60
```

```
Input : 1, 2, 3, 4, 5
Output : 110
```

A recursive solution is discussed [here](#). We can solve this problem using dynamic programming.

First, consider a sub-array from indices Left to Right(inclusive).

If we assume the balloon at index Last to be the last balloon to be burst in this sub-array, we would say the coined gained to be $A[\text{left}-1]*A[\text{last}]*A[\text{right}+1]$.

Also, the total Coin Gained would be this value, plus $\text{dp}[\text{left}][\text{last} - 1] + \text{dp}[\text{last} + 1][\text{right}]$, where $\text{dp}[i][j]$ means maximum coin gained for sub-array with indices i, j.

Therefore, for each value of Left and Right, we need find and choose a value of Last with maximum coin gained, and update the dp array.

Our Answer is the value at $\text{dp}[1][N]$.

Python3

```
# Python program burst balloon problem.
```

```
def getMax(A):
    N = len(A)
    A = [1] + A + [1] # Add Bordering Balloons
    dp = [[0 for x in range(N + 2)] for y in range(N + 2)] # Declare DP Array

    for length in range(1, N + 1):
        for left in range(1, N-length + 2):
            right = left + length - 1

                # For a sub-array from indices left, right
                # This innermost loop finds the last balloon burst
            for last in range(left, right + 1):
                dp[left][right] = max(dp[left][right], \
                                      dp[left][last-1] + \
                                      A[left-1]*A[last]*A[right + 1] + \
                                      dp[last + 1][right])

    return(dp[1][N])

# Driver code
A = [1, 2, 3, 4, 5]
print(getMax(A))
```

Output:

110

Source

<https://www.geeksforgeeks.org/burst-balloon-to-maximize-coins/>

Chapter 12

C qsort() vs C++ sort()

C qsort() vs C++ sort() - GeeksforGeeks

Standard C library provides qsort function that can be used for sorting an array. Following is the prototype of qsort() function.

```
// Sort an array of any type. The parameters are, base
// address of array, size of array and pointer to
// comparator function
void qsort (void* base, size_t num, size_t size,
           int (*comparator)(const void*, const void*));
```

It requires a pointer to the array, the number of elements in the array, the size of each element and a comparator function. We have discussed qsort comparator in detail [here](#).

C++ Standard Library provides a similar function sort() that originated in the STL. We have discussed C++ sort [here](#). Following are prototypes of C++ sort() function.

```
// To sort in default or ascending order.
template
void sort(T first, T last);

// To sort according to the order specified
// by comp.
template
void sort(T first, T last, Compare comp);
```

The order of equal elements is not guaranteed to be preserved. C++ provides std::stable_sort that can be used to preserve order.

Comparison to qsort and sort()

1. Implementation details:

As the name suggests, qsort function uses QuickSort algorithm to sort the given array, although the C standard does not require it to implement quicksort.

C++ sort function uses introsort which is a hybrid algorithm. Different implementations use different algorithms. The GNU Standard C++ library, for example, uses a 3-part hybrid sorting algorithm: introsort is performed first (introsort itself being a hybrid of quicksort and heap sort) followed by an insertion sort on the result.

2. Complexity :

The C standard doesn't talk about its complexity of qsort. The new C++11 standard requires that the complexity of sort to be $O(N \log N)$ in the worst case. Previous versions of C++ such as C++03 allow possible worst case scenario of $O(N^2)$. Only average complexity was required to be $O(N \log N)$.

3. Running time:

STL's sort ran faster than C's qsort, because C++'s templates generate optimized code for a particular data type and a particular comparison function.

STL's sort runs 20% to 50% faster than the hand-coded quicksort and 250% to 1000% faster than the C qsort library function. C might be the fastest language but qsort is very slow.

When we tried to sort one million integers on C++14, Time taken by C qsort() was 0.247883 sec and time taken by C++ sort() was only 0.086125 sec

```
// C++ program to demonstrate performance of
// C qsort and C++ sort() algorithm
#include <bits/stdc++.h>
using namespace std;

// Number of elements to be sorted
#define N 1000000

// A comparator function used by qsort
int compare(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

// Driver program to test above functions
int main()
{
    int arr[N], dupArr[N];

    // seed for random input
    srand(time(NULL));

    // to measure time taken by qsort and sort
    clock_t begin, end;
    double time_spent;
```

```
// generate random input
for (int i = 0; i < N; i++)
    dupArr[i] = arr[i] = rand()%100000;

begin = clock();
qsort(arr, N, sizeof(int), compare);
end = clock();

// calculate time taken by C qsort function
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C qsort() - "
    << time_spent << endl;

time_spent = 0.0;

begin = clock();
sort(dupArr, dupArr + N);
end = clock();

// calculate time taken by C++ sort
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C++ sort() - "
    << time_spent << endl;

return 0;
}
```

Output :

```
Time taken by C qsort() - 0.247883
Time taken by C++ sort() - 0.086125
```

C++ sort() is blazingly faster than qsort() on equivalent data due to inlining. sort() on a container of integers will be compiled to use std::less::operator() by default, which will be inlined and sort() will be comparing the integers directly. On the other hand, qsort() will be making an indirect call through a function pointer for every comparison which compilers fails to optimize.

4. Flexibility:

STL's sort works for all data types and for different data containers like C arrays, C++ vectors, C++ deques, etc and other containers that can be written by the user. This kind of flexibility is rather difficult to achieve in C.

5. Safety:

Compared to qsort, the templated sort is more type-safe since it does not require access to data items through unsafe void pointers, as qsort does.

References:

<http://theory.stanford.edu/~amitp/rants/c++-vs-c>
[https://en.wikipedia.org/wiki/Sort_\(C%2B%2B\)](https://en.wikipedia.org/wiki/Sort_(C%2B%2B))

Source

<https://www.geeksforgeeks.org/c-qsort-vs-c-sort/>

Chapter 13

C++ tricks for competitive programming (for C++ 11)

C++ tricks for competitive programming (for C++ 11) - GeeksforGeeks

We have discussed some tricks in below post. In this post, some more tricks are discussed.

[Writing C/C++ code efficiently in Competitive programming](#)

Although practice is the only way that ensures increased performance in programming contests but having some tricks up your sleeve ensures an upper edge and fast debugging.

1) Checking if the number is even or odd without using the % operator:

Although this trick is not much better than using % operator but is sometimes efficient (with large numbers). Use & operator:

```
if (num & 1)
    cout << "ODD";
else
    cout << "EVEN";
```

Example:

num = 5

Binary: “101 & 1” will be 001, so true

num = 4

Binary: “100 & 1” will be 000, so false.

2) Fast Multiplication or Division by 2

Multiplying by 2 means shifting all the bits to left and dividing by 2 means shifting to the right.

Example : 2 (Binary 10): shifting left 4 (Binary 100) and right 1 (Binary 1)

```
n = n << 1; // Multiply n with 2
n = n >> 1; // Divide n by 2
```

3) Swapping of 2 numbers using XOR:

This method is fast and doesn't require the use of 3rd variable.

```
// A quick way to swap a and b  
a ^= b;  
b ^= a;  
a ^= b;
```

4) Avoiding use of strlen():

```
// Use of strlen() can be avoided by:  
for (i=0; s[i]; i++)  
{  
}  
// loop breaks when the character array ends.
```

5) Use of emplace_back() (Discussed [here](#), [here](#) and [here](#))

Instead of push_back() in STL emplace_back can be used because it is much faster and instead of allocating memory somewhere else and then appending it directly allocates memory in the container.

6) Inbuilt GCD function: C++ has inbuilt GCD function and there is no need to explicitly code it. Syntax: __gcd(x, y);

7) Using Inline functions: We can create inline functions and use them without having to code them up during the contest. Examples: (a) function for sieve, (b) function for palindrome.

8) Maximum size of the array: We must be knowing that the maximum size of array declared inside the main function is of the order of 10^6 but if you declare array globally then you can declare its size upto 10^7 .

9) Calculating the most significant digit: To calculate the most significant digit of any number log can be directly used to calculate it.

```
Suppose the number is N then  
Let double K = log10(N);  
now K = K - floor(K);  
int X = pow(10, K);  
X will be the most significant digit.
```

10) Calculating the number of digits directly: To calculate number of digits in a number, instead of looping you can efficiently use log :

```
Number of digits in N =floor(log10(N)) + 1;
```

11) Efficient trick to know if a number is a power of 2 sing the normal technique of division the complexity comes out to be $O(\log N)$, but it can be solved using $O(v)$ where v are the number of digits of number in binary form.

```
/* Function to check if x is power of 2*/
bool isPowerOfTwo (int x)
{
    /* First x in the below expression is
       for the case when x is 0 */
    return x && !(x&(x-1));
}
```

12) C++11 has in built algorithms for following:

```
// are all of the elements positive?
all_of(first, first+n, ispositive());

// is there at least one positive element?
any_of(first, first+n, ispositive());

// are none of the elements positive?
none_of(first, first+n, ispositive());
```

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and iota\)](#) for details.

13) Copy Algorithm: used to copy elements from one container to another.

```
int source[5] = {0, 12, 34, 50, 80};
int target[5];
// copy 5 elements from source to target
copy_n(source, 5, target);
```

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and iota\)](#) for details.

14) The Iota Algorithm The algorithm iota() creates a range of sequentially increasing values, as if by assigning an initial value to *first, then incrementing that value using prefix ++. In the following listing, iota() assigns the consecutive values {10, 11, 12, 13, 14} to the array arr, and {'a', 'b', 'c'} to the char array c[].

```
int a[5] = {0};
char c[3] = {0};

// changes a to {10, 11, 12, 13, 14}
iota(a, a+5, 10);
iota(c, c+3, 'a'); // {'a', 'b', 'c'}
```

Please refer [Array algorithms in C++ STL \(all_of, any_of, none_of, copy_n and itoa\)](#) for details.

15) Initialization in Binary form: In C++ 11 assignments can also be made in binary form.

```
// C++ code to demonstrate working of
// "binary" numbers
#include<iostream>
using namespace std;
int main()
{
    auto number = 0b011;
    cout << number;
    return 0;
}
```

Output :

3

16) Use of “and” Though not a very productive one, this tip helps you to just use conditional operator and instead of typing &.

```
// C++ code to demonstrate working of "and"
#include<iostream>
using namespace std;
int main()
{
    int a = 10;
    if (a < 20 and a > 5)
        cout << "Yes";
    return 0;
}
```

Output :

Yes

Improved By : [bugsanderrors](#)

Source

<https://www.geeksforgeeks.org/c-tricks-competitive-programming-c-11/>

Chapter 14

C++: Methods of code shortening in competitive programming

C++: Methods of code shortening in competitive programming - GeeksforGeeks

Shortcode is ideal in competitive programming because programs should be written as fast as possible. Because of this, competitive programmers often define shorter names for datatypes and other parts of the code.

We here discuss the method of code shortening in C++ specifically.

Type names

Using the command *typedef* it is possible to give a shorter name to a datatype.

For example, the name long long is long, so we can define a shorter name ll:

```
typedef long long ll;
```

After this, the code

```
long long a = 123456789;
long long b = 987654321;
cout << a * b << "\n";
```

can be shorted as follows:

```
ll a = 123456789;
ll b = 987654321;
cout << a * b << "\n";
```

The command *typedef* can also be used with more complex types. For example, the following code gives the name vi for a vector of integers and the name pi for a pair that contains two integers,

```
typedef vector<int> vi;
typedef pair<int> pi;
```

Macros

Another way to shorten code is to define **macros**. A macro means that certain strings in the code will be changed before the compilation. In C++, macros are defined using the `#define` keyword.

For example, we can define the following macros:

```
#define F first; #define S second; #define PB push_back; #define MP make_pair;
```

After this, the code

```
v.push_back(make_pair(y1, x1));
v.push_back(make_pair(y2, x2));
int d = v[i].first+v[i].second;
```

can be shortened as follows;

```
v.PB(MP(y1, x1));
v.PB(MP(y2, x2));
int d = v[i].F+v[i].S;
```

A macro can also have parameters which makes it possible to shorten loops and others structures. For example, we can define the following macro:

```
#define REP(i, a, b) for (int i=a; i<=b; i++)
```

After this, the code

```
for (int i=1; i<=n; i++){      search(i); }
```

can be shortened as follows:

```
REP(i, 1, n){      search(i); }
```

A version of template given below

This can be used in competitive programming for faster coding.

```
#include <bits/stdc++.h> // Include every standard library
using namespace std;

typedef long long LL;
typedef pair<int, int> pii;
typedef pair<LL, LL> pll;
typedef pair<string, string> pss;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<pii> vii;
typedef vector<LL> vl;
typedef vector<vl> vvl;

double EPS = 1e-9;
int INF = 1000000005;
long long INFF = 10000000000000000005LL;
double PI = acos(-1);
```

```
int dirx[8] = { -1, 0, 0, 1, -1, -1, 1, 1 };
int diry[8] = { 0, 1, -1, 0, -1, 1, -1, 1 };

#ifndef TESTING
#define DEBUG fprintf(stderr, "====TESTING====\n")
#define VALUE(x) cerr << "The value of " << #x << " is " << x << endl
#define debug(...) fprintf(stderr, __VA_ARGS__)
#else
#define DEBUG
#define VALUE(x)
#define debug(...)
#endif

#define FOR(a, b, c) for (int(a) = (b); (a) < (c); ++(a))
#define FORN(a, b, c) for (int(a) = (b); (a) <= (c); ++(a))
#define FORD(a, b, c) for (int(a) = (b); (a) >= (c); --(a))
#define FORSQ(a, b, c) for (int(a) = (b); (a) * (a) <= (c); ++(a))
#define FORC(a, b, c) for (char(a) = (b); (a) <= (c); ++(a))
#define FOREACH(a, b) for (auto&(a) : (b))
#define REP(i, n) FOR(i, 0, n)
#define REPN(i, n) FORN(i, 1, n)
#define MAX(a, b) a = max(a, b)
#define MIN(a, b) a = min(a, b)
#define SQR(x) ((LL)(x) * (x))
#define RESET(a, b) memset(a, b, sizeof(a))
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define ALL(v) v.begin(), v.end()
#define ALLA(arr, sz) arr, arr + sz
#define SIZE(v) (int)v.size()
#define SORT(v) sort(ALL(v))
#define REVERSE(v) reverse(ALL(v))
#define SORTA(arr, sz) sort(ALLA(arr, sz))
#define REVERSEA(arr, sz) reverse(ALLA(arr, sz))
#define PERMUTE next_permutation
#define TC(t) while (t--)

inline string IntToString(LL a)
{
    char x[100];
    sprintf(x, "%lld", a);
    string s = x;
    return s;
}

inline LL StringToInt(string a)
```

```
{  
    char x[100];  
    LL res;  
    strcpy(x, a.c_str());  
    sscanf(x, "%lld", &res);  
    return res;  
}  
  
inline string GetString(void)  
{  
    char x[1000005];  
    scanf("%s", x);  
    string s = x;  
    return s;  
}  
  
inline string uppercase(string s)  
{  
    int n = SIZE(s);  
    REP(i, n)  
    if (s[i] >= 'a' && s[i] <= 'z')  
        s[i] = s[i] - 'a' + 'A';  
    return s;  
}  
  
inline string lowercase(string s)  
{  
    int n = SIZE(s);  
    REP(i, n)  
    if (s[i] >= 'A' && s[i] <= 'Z')  
        s[i] = s[i] - 'A' + 'a';  
    return s;  
}  
  
inline void OPEN(string s)  
{  
#ifndef TESTING  
    freopen((s + ".in").c_str(), "r", stdin);  
    freopen((s + ".out").c_str(), "w", stdout);  
#endif  
}  
  
// end of Sektor_jr template v2.0.3 (BETA)  
  
int main()  
{  
    freopen("A.in", "r", stdin);  
    freopen("output.txt", "w", stdout);
```

```
int a, b;
fin >> a >> b;
fout << a + b << endl;
return 0;
}
```

Source

<https://www.geeksforgeeks.org/c-methods-of-code-shortening-in-competitive-programming/>

Chapter 15

Check if a M-th fibonacci number divides N-th fibonacci number

Check if a M-th fibonacci number divides N-th fibonacci number - GeeksforGeeks

Given two numbers M and N, the task is to check if the M-th and N-th [Fibonacci](#) numbers perfectly divide each other or not.

Examples:

Input: M = 3, N = 6

Output: Yes

$F(3) = 2$, $F(6) = 8$ and $F(6) \% F(3) = 0$

Input: M = 2, N = 9

Output: No

A **naive approach** will be to find the N-th and M-th Fibonacci numbers and check if they are perfectly divisible or not.

An **efficient approach** is to use the [Fibonacci property](#) to determine the result. If m perfectly divides n, then F_m also perfectly divides F_n , else it does not.

Exception: When N is 2, it is always possible as Fib_2 is 1, which divides every other Fibonacci number.

Below is the implementation of the above approach:

C++

```
// C++ program to check if  
// M-th fibonacci divides N-th fibonacci
```

```
#include <bits/stdc++.h>
using namespace std;

void check(int n, int m)
{
    // exceptional case for F(2)
    if (n == 2 || m == 2 || n % m == 0) {
        cout << "Yes" << endl;
    }
    // if none of the above cases,
    // hence not divisible
    else {
        cout << "No" << endl;
    }
}

// Driver Code
int main()
{
    int m = 3, n = 9;
    check(n, m);

    return 0;
}
```

Java

```
// Java program to check
// if M-th fibonacci
// divides N-th fibonacci
import java.io.*;

class GFG
{
static void check(int n, int m)
{
    // exceptional case for F(2)
    if (n == 2 || m == 2 ||
        n % m == 0)
    {
        System.out.println( "Yes");
    }

    // if none of the above cases,
    // hence not divisible
    else
    {
        System.out.println( "No");
    }
}
```

```
        }
    }

// Driver Code
public static void main (String[] args)
{
    int m = 3, n = 9;
    check(n, m);
}
}

// This code is contributed
// by anuj_67.
```

Python 3

```
# Python 3 program to
# check if M-th fibonacci
# divides N-th fibonacci

def check(n, m):

    # exceptional case for F(2)
    if (n == 2 or m == 2 or
        n % m == 0) :
        print( "Yes" )

    # if none of the above
    # cases, hence not divisible
    else :
        print( "No" )

# Driver Code
m = 3
n = 9
check(n, m)

# This code is contributed
# by Smitha
```

C#

```
// C# program to check
// if M-th fibonacci
// divides N-th fibonacci
using System;
```

```
class GFG
{
static void check(int n, int m)
{
    // exceptional case for F(2)
    if (n == 2 || m == 2 ||
        n % m == 0)
    {
        Console.WriteLine( "Yes");
    }

    // if none of the above cases,
    // hence not divisible
    else
    {
        Console.WriteLine( "No");
    }
}

// Driver Code
public static void Main ()
{
    int m = 3, n = 9;
    check(n, m);
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
<?php
// PHP program to check
// if M-th fibonacci
// divides N-th fibonacci

function check($n, $m)
{
    // exceptional case for F(2)
    if ($n == 2 || $m == 2 ||
        $n % $m == 0)
    {
        echo "Yes" , "\n";
    }

    // if none of the above cases,
    // hence not divisible
```

```
    else
    {
        echo "No" ,"\n";
    }
}

// Driver Code
$m = 3; $n = 9;
check($n, $m);

// This code is contributed
// by anuj_67.
?>
```

Output:

Yes

Time Complexity: O(1).

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/check-if-a-m-th-fibonacci-number-divides-n-th-fibonacci-number/>

Chapter 16

Check if any permutation of a large number is divisible by 8

Check if any permutation of a large number is divisible by 8 - GeeksforGeeks

Given a large number N and the task is to check if any permutation of a large number is divisible by 8.

Examples:

Input: N = 31462708

Output: Yes

Many of permutation of number N like
34678120, 34278160 are divisible by 8.

Input: 75

Output: No

A **naive approach** is to generate all permutations of the number N and check if($N \% 8 == 0$) and return true if any of the permutations is divisible by 8.

An **efficient approach** is to use the fact that if the last three digits of a number are divisible by 8, then the number is also divisible by 8. Below are the required steps:

- Use a hash-table to count the occurrences of all digits in the given number.
- Traverse for all three-digit numbers which are divisible by 8.
- Check if the digits in the three-digit number are in the hash-table.
- If yes a number can be formed by permutation where the last three digits combine to form the three-digit number.
- If none of the three-digit numbers can be formed, return false.

Below is the implementation of the above approach:

C++

```
// C++ program to check if any permutation of
// a large number is divisible by 8 or not
#include <bits/stdc++.h>
using namespace std;

// Function to check if any permutation
// of a large number is divisible by 8
bool solve(string n, int l)
{

    // Less than three digit number
    // can be checked directly.
    if (l < 3) {
        if (stoi(n) % 8 == 0)
            return true;

        // check for the reverse of a number
        reverse(n.begin(), n.end());
        if (stoi(n) % 8 == 0)
            return true;
        return false;
    }

    // Stores the Frequency of characters in the n.
    int hash[10] = { 0 };
    for (int i = 0; i < l; i++)
        hash[n[i] - '0']++;

    // Iterates for all three digit numbers
    // divisible by 8
    for (int i = 104; i < 1000; i += 8) {

        int dup = i;

        // stores the frequency of all single
        // digit in three-digit number
        int freq[10] = { 0 };
        freq[dup % 10]++;
        dup = dup / 10;
        freq[dup % 10]++;
        dup = dup / 10;
        freq[dup % 10]++;

        dup = i;

        // check if the original number has
```

```
// the digit
if (freq[dup % 10] > hash[dup % 10])
    continue;
dup = dup / 10;

if (freq[dup % 10] > hash[dup % 10])
    continue;
dup = dup / 10;

if (freq[dup % 10] > hash[dup % 10])
    continue;

return true;
}

// when all are checked its not possible
return false;
}

// Driver Code
int main()
{
    string number = "31462708";
    int l = number.length();

    if (solve(number, l))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// Java program to check if
// any permutation of a large
// number is divisible by 8 or not
import java.util.*;

class GFG
{
    // Function to check if any
    // permutation of a large
    // number is divisible by 8
    public static boolean solve(String n, int l)
    {

        // Less than three digit number
```

```
// can be checked directly.  
if (l < 3)  
{  
    if (Integer.parseInt(n) % 8 == 0)  
        return true;  
  
    // check for the reverse  
    // of a number  
    n = new String((new StringBuilder()).append(n).reverse());  
  
    if (Integer.parseInt(n) % 8 == 0)  
        return true;  
    return false;  
}  
  
// Stores the Frequency of  
// characters in the n.  
int []hash = new int[10];  
for (int i = 0; i < l; i++)  
    hash[n.charAt(i) - '0']++;  
  
// Iterates for all  
// three digit numbers  
// divisible by 8  
for (int i = 104; i < 1000; i += 8)  
{  
    int dup = i;  
  
    // stores the frequency of  
    // all single digit in  
    // three-digit number  
    int []freq = new int[10];  
    freq[dup % 10]++;  
    dup = dup / 10;  
    freq[dup % 10]++;  
    dup = dup / 10;  
    freq[dup % 10]++;  
  
    dup = i;  
  
    // check if the original  
    // number has the digit  
    if (freq[dup % 10] >  
        hash[dup % 10])  
        continue;  
    dup = dup / 10;  
  
    if (freq[dup % 10] >
```

```
        hash[dup % 10])
        continue;
    dup = dup / 10;

    if (freq[dup % 10] >
        hash[dup % 10])
        continue;

    return true;
}

// when all are checked
// its not possible
return false;
}

// Driver Code
public static void main(String[] args)
{
    String number = "31462708";

    int l = number.length();

    if (solve(number, l))
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed
// by Harshit Saini
```

Python3

```
# Python3 program to check if
# any permutation of a large
# number is divisible by 8 or not

# Function to check if any
# permutation of a large
# number is divisible by 8
def solve(n, l):

    # Less than three digit
    # number can be checked
    # directly.
    if l < 3:
```

```
if int(n) % 8 == 0:  
    return True  
  
# check for the reverse  
# of a number  
n.sort(reverse = True)  
  
if int(n) % 8 == 0:  
    return True  
return False  
  
# Stores the Frequency of  
# characters in the n.  
hash = 10 * [0]  
for i in range(0, 1):  
    hash[int(n[i]) - 0] += 1;  
  
# Iterates for all  
# three digit numbers  
# divisible by 8  
for i in range(104, 1000, 8):  
    dup = i  
  
    # stores the frequency  
    # of all single digit  
    # in three-digit number  
    freq = 10 * [0]  
    freq[int(dup % 10)] += 1;  
    dup = dup / 10  
    freq[int(dup % 10)] += 1;  
    dup = dup / 10  
    freq[int(dup % 10)] += 1;  
  
    dup = i  
  
    # check if the original  
    # number has the digit  
    if (freq[int(dup % 10)] >  
        hash[int(dup % 10)]):  
        continue;  
    dup = dup / 10;  
  
    if (freq[int(dup % 10)] >  
        hash[int(dup % 10)]):  
        continue;  
    dup = dup / 10  
  
    if (freq[int(dup % 10)] >
```

```
hash[int(dup % 10)]):
    continue;

return True

# when all are checked
# its not possible
return False

# Driver Code
if __name__ == "__main__":
    number = "31462708"

    l = len(number)

    if solve(number, l):
        print("Yes")
    else:
        print("No")

# This code is contributed
# by Harshit Saini
```

PHP

```
<?php
error_reporting(0);
// PHP program to check
// if any permutation of
// a large number is
// divisible by 8 or not

// Function to check if
// any permutation of a
// large number is divisible by 8
function solve($n, $l)
{
    // Less than three digit
    // number can be checked
    // directly.
    if ($l < 3)
    {
        if (intval($n) % 8 == 0)
            return true;

        // check for the
```

```
// reverse of a number
strrev($n);
if (intval($n) % 8 == 0)
    return true;
return false;
}

// Stores the Frequency of
// characters in the n.
$hash[10] = array(0);
for ($i = 0; $i < $l; $i++)
    $hash[$n[$i] - '0']++;

// Iterates for all three
// digit numbers divisible by 8
for ($i = 104;
     $i < 1000; $i += 8)
{
    $dup = $i;

    // stores the frequency of
    // all single digit in
    // three-digit number
    $freq[10] = array(0);
    $freq[$dup % 10]++;
    $dup = $dup / 10;
    $freq[$dup % 10]++;
    $dup = $dup / 10;
    $freq[$dup % 10]++;

    $dup = $i;

    // check if the original
    // number has the digit
    if ($freq[$dup % 10] >
        $hash[$dup % 10])
        continue;
    $dup = $dup / 10;

    if ($freq[$dup % 10] >
        $hash[$dup % 10])
        continue;
    $dup = $dup / 10;

    if ($freq[$dup % 10] >
        $hash[$dup % 10])
        continue;
```

```
        return true;
    }

    // when all are checked
    // its not possible
    return false;
}

// Driver Code
$number = "31462708";
$l = strlen($number);

if (solve($number, $l))
    echo "Yes";
else
    echo "No";

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

Output:

Yes

Time Complexity: O(L), where L is the number of digits in the number.

Improved By : [Harshit Saini](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/check-if-any-permutation-of-a-large-number-is-divisible-by-8/>

Chapter 17

Check if concatenation of two strings is balanced or not

Check if concatenation of two strings is balanced or not - GeeksforGeeks

Given two bracket sequences S1 and S2 consisting of ‘(‘ and ‘)’. The task is to check if the string obtained by concatenating both the sequences is balanced or not. Concatenation can be done by $s1+s2$ or $s2+s1$.

Examples:

Input: $s1 = ")(())()$ ", $s2 = "((())()$ "

Output: Balanced

$s2 + s1 = "((())())()$ ", which
is a balanced parenthesis sequence.

Input: $s1 = "((()))"$, $s2 = "())()$ "

Output: Not balanced

$s1 + s2 = "((())())()$ " \rightarrow Not balanced

$s2 + s1 = "())()((())()$ " \rightarrow Not balanced

A **naive** solution is to first concatenate both sequences and then check if the resultant sequence is balanced or not using a stack. First, check if $s1 + s2$ is balanced or not. If not, then check if $s2 + s1$ is balanced or not. To check if a given sequence of brackets is balanced or not using a stack, the following algorithm can be used.

1. Declare a character stack S.
2. Now traverse the expression string exp.
 - If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.
 - If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from the stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.

3. After complete traversal, if there is some starting bracket left in stack then “not balanced”.

Below is the implementation of above approach:

C++

```
// CPP program to check if sequence obtained
// by concatenating two bracket sequences
// is balanced or not.
#include <bits/stdc++.h>
using namespace std;

// Check if given string is balanced bracket
// sequence or not.
bool isBalanced(string s)
{

    stack<char> st;

    int n = s.length();

    for (int i = 0; i < n; i++) {

        // If current bracket is an opening
        // bracket push it to stack.
        if (s[i] == '(')
            st.push(s[i]);

        // If current bracket is a closing
        // bracket then pop from stack if
        // it is not empty. If stack is empty
        // then sequence is not balanced.
        else {
            if (st.empty())
                return false;
        }

        else
            st.pop();
    }

    // If stack is not empty, then sequence
    // is not balanced.
    if (!st.empty())
        return false;
}
```

```
        return true;
    }

// Function to check if string obtained by
// concatenating two bracket sequences is
// balanced or not.
bool isBalancedSeq(string s1, string s2)
{

    // Check if s1 + s2 is balanced or not.
    if (isBalanced(s1 + s2))
        return true;

    // Check if s2 + s1 is balanced or not.
    return isBalanced(s2 + s1);
}

// Driver code.
int main()
{
    string s1 = ")()((()))";
    string s2 = "((())(";

    if (isBalancedSeq(s1, s2))
        cout << "Balanced";
    else
        cout << "Not Balanced";

    return 0;
}
```

Output:

Balanced

Time complexity: O(n)

Auxiliary Space: O(n)

An **efficient** solution is to check if given sequences can result in balanced parenthesis sequence without using a stack, i.e., in constant extra space.

Let the concatenated sequence is s. There are two possibilities: either $s = s_1 + s_2$ is balanced or $s = s_2 + s_1$ is balanced. Check for both possibilities whether s is balanced or not.

- If s is balanced, then the number of opening brackets in s should always be greater than or equal to the number of closing brackets in S at any instant of traversing it. This is because if at any instant number of closing brackets in s is greater than the

number of opening brackets, then the last closing bracket will not have a matching opening bracket (that is why the count is more) in s.

- If the sequence is balanced then at the end of traversal, the number of opening brackets in s is equal to the number of closing brackets in s.

Below is the implementation of above approach:

C++

```
// C++ program to check if sequence obtained
// by concatenating two bracket sequences
// is balanced or not.
#include <bits/stdc++.h>
using namespace std;

// Check if given string is balanced bracket
// sequence or not.
bool isBalanced(string s)
{

    // To store result of comparison of
    // count of opening brackets and
    // closing brackets.
    int cnt = 0;

    int n = s.length();

    for (int i = 0; i < n; i++) {

        // If current bracket is an
        // opening bracket, then
        // increment count.
        if (s[i] == '(')
            cnt++;

        // If current bracket is a
        // closing bracket, then
        // decrement count and check
        // if count is negative.
        else {
            cnt--;
            if (cnt < 0)
                return false;
        }
    }

    // If count is positive then
```

```
// some opening brackets are
// not balanced.
if (cnt > 0)
    return false;

return true;
}

// Function to check if string obtained by
// concatenating two bracket sequences is
// balanced or not.
bool isBalancedSeq(string s1, string s2)
{

    // Check if s1 + s2 is balanced or not.
    if (isBalanced(s1 + s2))
        return true;

    // Check if s2 + s1 is balanced or not.
    return isBalanced(s2 + s1);
}

// Driver code.
int main()
{
    string s1 = ")()((()))";
    string s2 = "((())(";

    if (isBalancedSeq(s1, s2))
        cout << "Balanced";
    else
        cout << "Not Balanced";

    return 0;
}
```

Java

```
// Java program to check if
// sequence obtained by
// concatenating two bracket
// sequences is balanced or not.
import java.io.*;

class GFG
{

    // Check if given string
```

```
// is balanced bracket
// sequence or not.
static boolean isBalanced(String s)
{

    // To store result of comparison
    // of count of opening brackets
    // and closing brackets.
    int cnt = 0;
    int n = s.length();
    for (int i = 0; i < n; i++)
    {

        // If current bracket is
        // an opening bracket,
        // then increment count.
        if (s.charAt(i) == '(')
        {
            cnt = cnt + 1;
        }

        // If current bracket is a
        // closing bracket, then
        // decrement count and check
        // if count is negative.
        else
        {
            cnt = cnt - 1;
            if (cnt < 0)
                return false;
        }
    }

    // If count is positive then
    // some opening brackets are
    // not balanced.
    if (cnt > 0)
        return false;

    return true;
}

// Function to check if string
// obtained by concatenating
// two bracket sequences is
// balanced or not.
static boolean isBalancedSeq(String s1,
                             String s2)
```

```
{  
  
// Check if s1 + s2 is  
// balanced or not.  
if (isBalanced(s1 + s2))  
    return true;  
  
// Check if s2 + s1 is  
// balanced or not.  
return isBalanced(s2 + s1);  
}  
  
// Driver code  
public static void main(String [] args)  
{  
    String s1 = ")()((()));"  
    String s2 = "((())(";  
  
    if (isBalancedSeq(s1, s2))  
    {  
        System.out.println("Balanced");  
    }  
    else  
    {  
        System.out.println("Not Balanced");  
    }  
}  
}  
  
// This code is contributed  
// by Shivi_Agarwal
```

Python3

```
# Python3 program to check  
# if sequence obtained by  
# concatenating two bracket  
# sequences is balanced or not.  
  
# Check if given string  
# is balanced bracket  
# sequence or not.  
def isBalanced(s):  
  
    # To store result of  
    # comparison of count  
    # of opening brackets  
    # and closing brackets.
```

```
cnt = 0
n = len(s)

for i in range(0, n):
    if (s[i] == '('):
        cnt = cnt + 1
    else :
        cnt = cnt - 1
        if (cnt < 0):
            return False
    if (cnt > 0):
        return False

return True

def isBalancedSeq(s1, s2):

    if (isBalanced(s1 + s2)):
        return True

    return isBalanced(s2 + s1)

# Driver code
a = ")()((()))";
b = "((())(";

if (isBalancedSeq(a, b)):
    print("Balanced")
else:
    print("Not Balanced")

# This code is contributed
# by Shivi_Agarwal
```

Output:

Balanced

Time complexity: O(n)
Auxiliary Space: O(1)

Improved By : [Shivi_Agarwal](#)

Source

<https://www.geeksforgeeks.org/check-if-concatenation-of-two-strings-is-balanced-or-not/>

Chapter 18

Check if it is possible to convert one string into another with given constraints

Check if it is possible to convert one string into another with given constraints - Geeks-forGeeks

Given two strings contains three characters i.e ‘A’, ‘B’ and ‘#’ only. Check is it possible to convert first string into another string by performing following operations on string first.

- 1- ‘A’ can move towards Left only
- 2- ‘B’ can move towards Right only
- 3- Neither ‘A’ nor ‘B’ cross each other

If it is possible then print “Yes” otherwise “No”.

Examples:

Input : str1=” #A#B#B# “, str2=” A###B#B ”

Output :Yes

Explanation :

‘A’ in str1 is right to the ‘A’ in str2 so ‘A’ of str1 can move easily towards the left because there is no ‘B’ on its left positions and for first ‘B’ in str1 is left to the ‘B’ in str2 so ‘B’ of str2 can move easily towards the right because there is no ‘A’ on its right positions and it is same for next ‘B’ so str1 can be easily converted into str2.

Input :str1=” #A#B# “, str2=” #B#A# ”

Output :No

Explanation :

Here first ‘A’ in str1 is left to the ‘A’ in str2 and according to the condition ‘A’ can’t move towards right. so str1 can’t be converted into str2.

Method :

1-Length of Both string must be same

- 2-No. of A's and B's in both the strings must be equal
3-Order of A and B in both the strings should be same(for ex: if 'A' is coming before 'B'in string second then the same sequence must be follow on string first)

```
// C++ Program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to check is it possible to convert
// first string into another string or not.
bool isItPossible(string str1, string str2, int m, int n)
{

    // To Check Length of Both String is Equal or Not
    if (m != n)
        return false;

    // To Check Frequency of A's and B's are
    // equal in both strings or not.
    if (count(str1.begin(), str1.end(), 'A') !=
        count(str2.begin(), str2.end(), 'A') ||
        count(str1.begin(), str1.end(), 'B') !=
        count(str2.begin(), str2.end(), 'B'))
        return false;

    // Start traversing
    for (int i = 0; i < m; i++) {
        if (str1[i] != '#') {
            for (int j = 0; j < n; j++) {

                // To Check no two elements cross each other.
                if ((str2[j] != str1[i]) && str2[j] != '#')
                    return false;

                if (str2[j] == str1[i]) {
                    str2[j] = '#';

                    // To Check Is it Possible to Move
                    // towards Left or not.
                    if (str1[i] == 'A' && i < j)
                        return false;

                    // To Check Is it Possible to Move
                    // towards Right or not.
                    if (str1[i] == 'B' && i > j)
                        return false;
                }
            }
        }
    }

    break;
}
```

```
        }
    }
}

return true;
}

// Drivers code
int main()
{
    string str1 = "A#B#";
    string str2 = "A##B";

    int m = str1.length();
    int n = str2.length();

    isItPossible(str1, str2, m, n) ? cout << "Yes\n"
                                    : cout << "No\n";

    return 0;
}
```

Output:

Yes

Time Complexity : $O(n^2)$

Source

<https://www.geeksforgeeks.org/check-if-it-is-possible-to-convert-one-string-into-another/>

Chapter 19

Check if the large number formed is divisible by 41 or not

Check if the large number formed is divisible by 41 or not - GeeksforGeeks

Given the first two digits of a large number *digit1* and *digit2*. Also given a number *c* and the length of the actual large number. The next n-2 digits of the large number are calculated using the formula **digit[i] = (digit[i - 1]*c + digit[i - 2]) % 10**. The task is to check whether the number formed is divisible by 41 or not.

Examples:

Input: digit1 = 1 , digit2 = 2 , c = 1 , n = 3

Output: YES

The number formed is 123

which is divisible by 41

Input: digit1 = 1 , digit2 = 4 , c = 6 , n = 3

Output: NO

A **naive approach** is to form the number using the given formula. Check if the number formed is divisible by 41 or not using % operator. But since the number is very large, it will not be possible to store such a large number.

Efficient Approach : All the digits are calculated using the given formula and then the *associative* property of *multiplication* and *addition* is used to check if it is divisible by 41 or not. A number is divisible by 41 or not means (number % 41) equals 0 or not.

Let X be the large number thus formed, which can be written as.

$$\begin{aligned} X &= (\text{digit}[0] * 10^{n-1}) + (\text{digit}[1] * 10^{n-2}) + \dots + (\text{digit}[n-1] * 10^0) \\ X &= (((\text{digit}[0] * 10 + \text{digit}[1]) * 10 + \text{digit}[2]) * 10 + \text{digit}[3]) \dots * 10 + \text{digit}[n-1] \\ X \% 41 &= (((((\text{digit}[0] * 10 + \text{digit}[1]) \% 41) * 10 + \text{digit}[2]) \% 41) * 10 + \text{digit}[3]) \% 41 \dots * 10 + \text{digit}[n-1]) \% 41 \end{aligned}$$

Hence after all the digits are calculated, below algorithm is followed:

1. Initialize the first digit to *ans*.
2. Iterate for all n-1 digits.
3. Compute ans at every ith step by $(ans * 10 + digit[i]) \% 41$ using associative property.
4. Check for the final value of ans if it divisible by 41 r not.

Below is the implementation of the above approach.

C++

```
// C++ program to check a large number
// divisible by 41 or not
#include <bits/stdc++.h>
using namespace std;

// Check if a number is divisible by 41 or not
bool DivisibleBy41(int first, int second, int c, int n)
{
    // array to store all the digits
    int digit[n];

    // base values
    digit[0] = first;
    digit[1] = second;

    // calculate remaining digits
    for (int i = 2; i < n; i++)
        digit[i] = (digit[i - 1] * c + digit[i - 2]) % 10;

    // calculate answer
    int ans = digit[0];
    for (int i = 1; i < n; i++)
        ans = (ans * 10 + digit[i]) % 41;

    // check for divisibility
    if (ans % 41 == 0)
        return true;
    else
        return false;
}

// Driver Code
int main()
{

    int first = 1, second = 2, c = 1, n = 3;
```

```
    if (DivisibleBy41(first, second, c, n))
        cout << "YES";
    else
        cout << "NO";
    return 0;
}
```

Java

```
// Java program to check
// a large number divisible
// by 41 or not
import java.io.*;

class GFG
{
// Check if a number is
// divisible by 41 or not
static boolean DivisibleBy41(int first,
                             int second,
                             int c, int n)
{
    // array to store
    // all the digits
    int digit[] = new int[n];

    // base values
    digit[0] = first;
    digit[1] = second;

    // calculate remaining
    // digits
    for (int i = 2; i < n; i++)
        digit[i] = (digit[i - 1] * c +
                    digit[i - 2]) % 10;

    // calculate answer
    int ans = digit[0];
    for (int i = 1; i < n; i++)
        ans = (ans * 10 +
               digit[i]) % 41;

    // check for
    // divisibility
    if (ans % 41 == 0)
        return true;
    else
```

```
        return false;
}

// Driver Code
public static void main (String[] args)
{
    int first = 1, second = 2, c = 1, n = 3;

    if (DivisibleBy41(first, second, c, n))
        System.out.println("YES");
    else
        System.out.println("NO");
}
}

// This code is contributed
// by akt_mit
```

Python3

```
# Python3 program to check
# a large number divisible
# by 41 or not

# Check if a number is
# divisible by 41 or not
def DivisibleBy41(first,
                  second, c, n):

    # array to store
    # all the digits
    digit = [0] * n

    # base values
    digit[0] = first
    digit[1] = second

    # calculate remaining
    # digits
    for i in range(2,n):
        digit[i] = (digit[i - 1] * c +
                    digit[i - 2]) % 10

    # calculate answer
    ans = digit[0]
    for i in range(1,n):
        ans = (ans * 10 + digit[i]) % 41
```

```
# check for
# divisibility
if (ans % 41 == 0):
    return True
else:
    return False

# Driver Code
first = 1
second = 2
c = 1
n = 3

if (DivisibleBy41(first,
                   second, c, n)):
    print("YES")
else:
    print("NO")

# This code is contributed
# by Smita
```

C#

```
// C# program to check
// a large number divisible
// by 41 or not
using System;

class GFG
{

    // Check if a number is
    // divisible by 41 or not
    static bool DivisibleBy41(int first,
                             int second,
                             int c, int n)
    {
        // array to store
        // all the digits
        int []digit = new int[n];

        // base values
        digit[0] = first;
        digit[1] = second;

        // calculate
        // remaining
```

```
// digits
for (int i = 2; i < n; i++)
    digit[i] = (digit[i - 1] * c +
                digit[i - 2]) % 10;

// calculate answer
int ans = digit[0];
for (int i = 1; i < n; i++)
    ans = (ans * 10 +
           digit[i]) % 41;

// check for
// divisibility
if (ans % 41 == 0)
    return true;
else
    return false;
}

// Driver Code
public static void Main ()
{
    int first = 1,
        second = 2,
        c = 1, n = 3;

    if (DivisibleBy41(first, second, c, n))
        Console.WriteLine("YES");
    else
        Console.WriteLine("NO");
}
}

// This code is contributed
// by Smita
```

PHP

```
<?php
// PHP program to check a
// large number divisible
// by 41 or not

// Check if a number is
// divisible by 41 or not
function DivisibleBy41($first, $second, $c, $n)
{
    // array to store
```

```
// all the digits
$digit[$n] = range(1, $n);

// base values
$digit[0] = $first;
$digit[1] = $second;

// calculate remaining digits
for ($i = 2; $i < $n; $i++)
    $digit[$i] = ($digit[$i - 1] * $c +
                  $digit[$i - 2]) % 10;

// calculate answer
$ans = $digit[0];
for ($i = 1; $i < $n; $i++)
    $ans = ($ans * 10 + $digit[$i]) % 41;

// check for divisibility
if ($ans % 41 == 0)
    return true;
else
    return false;
}

// Driver Code
$first = 1;
$second = 2;
$c = 1;
$n = 3;

if (DivisibleBy41($first, $second, $c, $n))
    echo "YES";
else
    echo "NO";

// This code is contributed by Mahadev.
?>
```

Output:

YES

Time Complexity: O(N)
Auxiliary Space: O(N)

Improved By : [jit_t](#), [Smitha Dinesh Semwal](#), [Mahadev99](#)

Source

<https://www.geeksforgeeks.org/check-if-the-number-is-divisible-by-41-or-not-when-first-two-digits-are-given/>

Chapter 20

Check if the last element of array is even or odd after performing a operation p times

Check if the last element of array is even or odd after performing a operation p times -
GeeksforGeeks

Given is an array of n non-negative integers. The operation is to insert a number in the array which is strictly greater than current sum of the array. After performing the operation p times, find whether the last element of the array is even or odd.

Examples:

```
Input : arr[] = {2, 3}
        P = 3
Output : EVEN
For p = 1, Array sum = 2 + 3 = 5.
So, we insert 6.
For p = 2, Array sum = 5 + 6 = 11.
So, we insert 12.
For p = 3, Array sum = 11 + 12 = 23.
So, we insert 24 (which is even).
```

```
Input : arr[] = {5, 7, 10}
        p = 1
Output : ODD
For p = 1, Array sum = 5 + 7 + 10 = 22.
So, we insert 23 (which is odd).
```

Naive Approach: First find the sum of the given array. This can be done in a single loop. Now make an another array of size P + N. This array will denote the element to be

inserted, and last element will be our required answer. At any step, if parity of the sum of the elements of array is “even”, parity of inserted element will be “odd”.

Efficient Approach: Let us say that sum of the array is even, next inserted element will be odd. Now sum of array will be odd, so next inserted element will be even, now sum of array becomes odd, so we insert an even number, and so on. We can generalize that if sum of array is even, then for P = 1, last inserted number will be odd, otherwise it will be even. Now, consider the case in which sum of the array is odd. The next inserted element will be even, now sum of array will become odd, so next inserted element will be even, now sum of array will be odd, add another even number and so on. We can generalize that last inserted number is always even in this case.

Below is the implementation of the above approach:

C++

```
// CPP program to check whether the last
// element of the array is even or odd
// after performing the operation p times.
#include <bits/stdc++.h>
using namespace std;

string check_last(int arr[], int n, int p)
{
    int sum = 0;

    // sum of the array.
    for (int i = 0; i < n; i++)
        sum = sum + arr[i];

    if (p == 1) {

        // if sum is even
        if (sum % 2 == 0)
            return "ODD";
        else
            return "EVEN";
    }
    return "EVEN";
}

// driver code
int main()
{
    int arr[] = { 5, 7, 10 }, p = 1;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << check_last(arr, n, p) << endl;
    return 0;
}
```

Python3

```
# Python3 code to check whether the last
# element of the array is even or odd
# after performing the operation p times.

def check_last (arr, n, p):
    _sum = 0

    # sum of the array.
    for i in range(n):
        _sum = _sum + arr[i]
    if p == 1:

        # if sum is even
        if _sum % 2 == 0:
            return "ODD"
        else:
            return "EVEN"
    return "EVEN"

# driver code
arr = [5, 7, 10]
p = 1
n = len(arr)
print(check_last (arr, n, p))

# This code is contributed by "Abhishek Sharma 44"
```

Java

```
// Java program to check whether the last
// element of the array is even or odd
// after performing the operation p times.
import java.util.*;

class Even_odd{

    public static String check_last(int arr[], int n, int p)
    {
        int sum = 0;

        // sum of the array.
        for (int i = 0; i < n; i++)
            sum = sum + arr[i];

        if (p == 1) {
```

```
// if sum is even
if (sum % 2 == 0)
    return "ODD";
else
    return "EVEN";
}

return "EVEN";
}

public static void main(String[] args)
{
    int arr[] = { 5, 7, 10 }, p = 1;
    int n = 3;
    System.out.print(check_last(arr, n, p));
}
}

//This code is contributed by rishabh_jain
```

C#

```
// C# program to check whether the last
// element of the array is even or odd
// after performing the operation p times.
using System;

class GFG {

    public static string check_last(int []arr, int n, int p)
    {
        int sum = 0;

        // sum of the array.
        for (int i = 0; i < n; i++)
            sum = sum + arr[i];

        if (p == 1) {

            // if sum is even
            if (sum % 2 == 0)
                return "ODD";
            else
                return "EVEN";
        }

        return "EVEN";
    }
}
```

```
// Driver code
public static void Main()
{
    int []arr = { 5, 7, 10 };
    int p = 1;
    int n = arr.Length;

    Console.WriteLine(check_last(arr, n, p));
}
}

//This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to check whether the last
// element of the array is even or odd
// after performing the operation p times.

function check_last($arr, $n, $p)
{
    $sum = 0;

    // sum of the array.
    for ( $i = 0; $i < $n; $i++)
        $sum = $sum + $arr[$i];

    if ($p == 1) {

        // if sum is even
        if ($sum % 2 == 0)
            return "ODD";
        else
            return "EVEN";
    }
    return "EVEN";
}

// Driver Code
$arr = array(5, 7, 10);
$p = 1;
$n = count($arr);
echo check_last($arr, $n, $p);

// This code is contributed by anuj_67.
?>
```

Output:

ODD

Time Complexity: $O(n)$

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-last-element-array-even-odd-performing-operation-p-times/>

Chapter 21

Cin-Cout vs Scanf-Printf

Cin-Cout vs Scanf-Printf - GeeksforGeeks

Regular competitive programmers face common challenge when input is large and the task of reading such an input from stdin might prove to be a bottleneck. Such problem is accompanied with “Warning: large I/O data”.

Let us create a dummy input file containing a line with 16 bytes followed by a newline and having 1000000 such lines, making a file of 17MB should be good enough.

```
// Creating a dummy file of size 17 MB to compare
// performance of scanf() and cin()
$ yes 1111111111111111 | head -1000000 > tmp/dummy
```

Let us compare the time taken to read the file from stdin (get the file from disk to stdin using redirection) by using scanf() versus cin.

```
// Filename : cin_test.cc to test the
// We redirect above created temp file
// of 17 MB to stdin when this program
// is run.
#include<iostream>
using namespace std;

int main()
{
    char buffer[256];
    while (cin >> buffer)
    {
    }
    return 0;
}
```

Output of above program when dummy file is redirected to stdin.

```
$ g++ cin_test.cc -o cin_test
$ time ./cin_test < /tmp/dummy
real      0m2.162s
user      0m1.696s
sys   0m0.332s

// Filename : scanf_test.c to see
// performance of scanf()
// We redirect above created temp file
// of 17 MB to stdin when this program
// is run.
#include<cstdlib>
#include<cstdio>
int main()
{
    char buffer[256];
    while (scanf("%s", buffer) != EOF)
    {
    }
    return 0;
}
```

Output of above program when dummy file is redirected to stdin.

```
$ g++ scanf_test.cc -o scanf_test
$ time ./scanf_test < /tmp/dummy
real      0m0.426s
user      0m0.248s
sys   0m0.084s
```

Well, the above results are consistent with our observations.

Why is scanf faster than cin?

On a high level both of them are wrappers over `read()` system call, just syntactic sugar. The only visible difference is that `scanf()` has to explicitly declare the input type, whereas `cin` has the redirection operation overloaded using templates. This does not seem like a good enough reason for a performance hit of 5x.

It turns out that `iostream` makes use of `stdio`'s buffering system. So, `cin` wastes time synchronizing itself with the underlying C-library's `stdio` buffer, so that calls to both `scanf()` and `cin` can be interleaved.

The good thing is that `libstdc++` provides an option to turn off synchronization of all the `iostream` standard streams with their corresponding standard C streams using

```
std::ios::sync_with_stdio(false);
```

and *cin* becomes faster than *scanf()* as it should have been.

A detailed article on [Fast Input Output in Competitive Programming](#)

```
// Filename : cin_test_2.cc to see
// performance of cin() with stdio sync
// disabled using sync_with_stdio(false).
#include<iostream>
using namespace std;

int main()
{
    char buffer[256];
    ios_base::sync_with_stdio(false);

    while (cin >> buffer)
    {

    }
    return 0;
}
```

Running the program :

```
$ g++ cin_test_2.cc -o cin_test_2
$ time./cin_test_2 </tmp/dummy
real    0m0.380s
user    0m0.240s
sys     0m0.028s
```

- Like with all things, there is a caveat here. With synchronization turned off, using *cin* and *scanf()* together will result in an undefined mess.
- With synchronization turned off, the above results indicate that *cin* is 8-10% faster than *scanf()*. This is probably because *scanf()* interprets the format arguments at runtime and makes use of variable number of arguments, whereas *cin* does this at compile time.

Now wondering, how fast can it be done?

```
// Redirecting contents of dummy file to null
// device (a special device that discards the
// information written to it) using command line.
$time cat /tmp/dummy > /dev/null
```

```
real    0m0.185s
user    0m0.000s
sys     0m0.092s
```

Wow! This is fast!!!

Source

<https://www.geeksforgeeks.org/cincout-vs-scanfprintf/>

Chapter 22

Cleaning the room

Cleaning the room - GeeksforGeeks

Given a room with square grids having '*' and '.' representing untidy and normal cells respectively.

You need to find whether room can be cleaned or not.

There is a machine which helps you in this task, but it is capable of cleaning only normal cell. Untidy cells cannot be cleaned with machine, until you have cleaned the normal cell in its row or column. Now, see to it whether room can be cleaned or not.

The input is as follows :

First line contains n the size of the room. The next n lines contains description for each row where the row[i][j] is '*' if it is more untidy than others else it is '.' if it is normal cell.

Examples:

```
Input : 3
      .**
      .**
      .**
Output :Yes, the room can be cleaned.
      1 1
      2 1
      3 1
Input :4
      ****
      ...*
      ..*.
      ..*.
      ..*.
Output : house cannot be cleaned.
```

Approach :

The minimum number of cells can be n . It is the only answer possible as it need to have

an element of type ‘*’ in every different row and column. If particular column and a given row contain ‘*’ in all the cells then, it is known that the house cannot be cleaned. Traverse every row and find the ‘*’ that can be used for the machine. Use this step two times, check every column for every row and then check for every row for every column. Then check if any of the two gives answer as n. If yes then house can be cleaned otherwise not. This approach will give us the minimum answer required.

In the first example the machine will clean cell (1, 1), (2, 1), (3, 1) in order to clean the entire room.

In the second example every cell in the $\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}$ row has ‘*’ and every cell in $\begin{smallmatrix} 3 & 4 \\ 1 & 2 \end{smallmatrix}$ column contains ‘*’, therefore the house cannot be cleaned. $\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}$ row cannot be cleaned in any way.

C++

```
// CPP code to find whether
// house can be cleaned or not
#include <bits/stdc++.h>
using namespace std;

// Matrix A stores the string
char A[105][105];

// ans stores the pair of indices
// to be cleaned by the machine
vector<pair<int, int> > ans;

// Function for printing the
// vector of pair
void print()
{
    cout << "Yes, the house can be"
        << " cleaned." << endl;

    for (int i = 0; i < ans.size(); i++)
        cout << ans[i].first << " "
            << ans[i].second << endl;
}

// Function performing calculations
int solve(int n)
{
    // push every first cell in
    // each row containing '.'
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (A[i][j] == '.') {
                ans.push_back(make_pair(i + 1, j + 1));
            }
        }
    }
}
```

```
        break;
    }
}

// If total number of cells are
// n then house can be cleaned
if (ans.size() == n) {
    print();
    return 0;
}

ans.clear();

// push every first cell in
// each column containing '.'
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (A[j][i] == '.') {
            ans.push_back(make_pair(i + 1, j + 1));
            break;
        }
    }
}

// If total number of cells are
// n then house can be cleaned
if (ans.size() == n) {
    print();
    return 0;
}
cout << "house cannot be cleaned."
     << endl;
}

// Driver function
int main()
{
    int n = 3;
    string s = "";
    s += ".**";
    s += ".**";
    s += ".**";
    int k = 0;

    // Loop to insert letters from
    // string to array
    for (int i = 0; i < n; i++) {
```

```
    for (int j = 0; j < n; j++)
        A[i][j] = s[k++];
}
solve(n);
return 0;
}
```

Output:

Yes, the house can be cleaned.

1 1
2 1
3 1

Source

<https://www.geeksforgeeks.org/cleaning-the-room/>

Chapter 23

Combinatorics on ordered trees

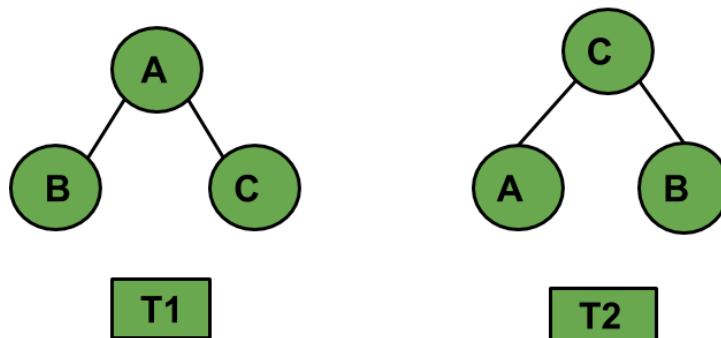
Combinatorics on ordered trees - GeeksforGeeks

An **ordered tree** is an oriented tree in which the children of a node are somehow ordered. It is a rooted tree in which an ordering is specified for the children of each vertex. This is called a “plane tree” because an ordering of the children is equivalent to an embedding of the tree in the plane, with the root at the top and the children of each vertex lower than that vertex.

Ordered tree can be further specified as labelled ordered tree and unlabelled ordered tree.

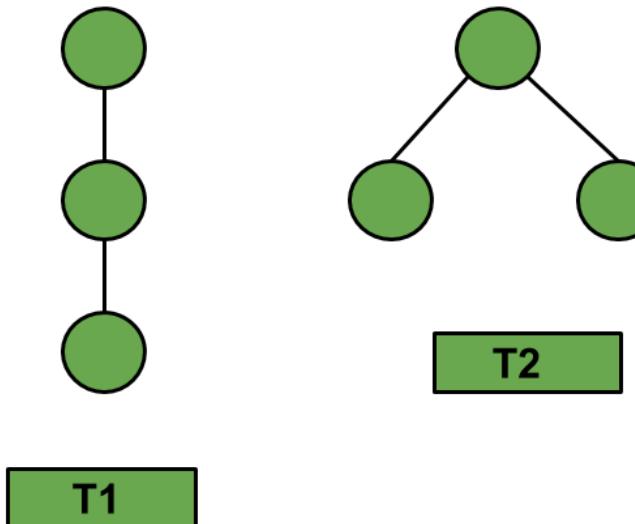
Prerequisite : [Catalan Numbers | Binomial Coefficient](#).

Labelled ordered trees : A labeled tree is a tree where each vertex is assigned a unique number from 1 to n.



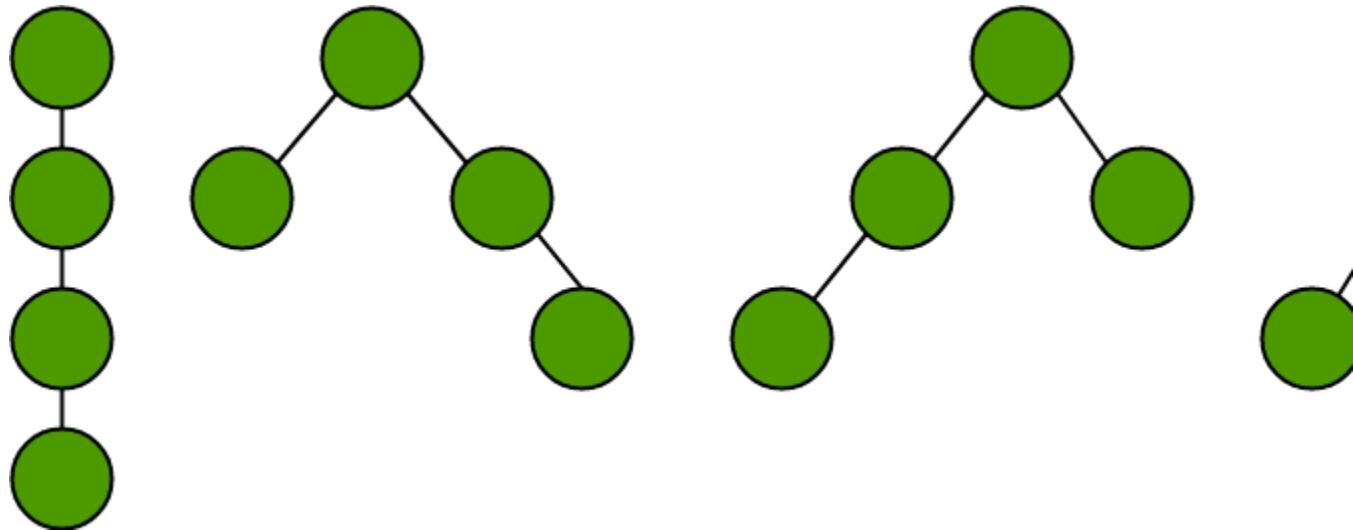
If T1 and T2 are ordered trees. Then, $T1 \neq T2$ else $T1 = T2$.

Unlabelled ordered trees : An unlabelled tree is a tree where every vertex is unlabelled. Given below are the possible unlabelled ordered tree having 3 vertices.



The total number of unlabelled ordered trees having n nodes is equal to the $(n - 1)$ -th [Catalan Number](#).

Given below are the possible unlabelled ordered trees having 4 nodes. This diagram will work as a reference example for the next few results.



1. Number of trees with exactly k leaves.

Let us consider, we have a 'n' edges . Then, the solution for the total possible ordered trees having 'k' leaves is given by :

$$L_n(k) = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$$

2. Total number of nodes of degree d in these trees.

Let us consider, we have a ‘n’ edges . Then, the solution for the total number of nodes having degree ‘d’ is given by :

$$D_n(d) = \binom{2n - 1 - d}{n - 1}$$

3. Number of trees in which the root has degree r.

Let us consider, we have a ‘n’ edges . Then, the solution for the total possible ordered trees whose root has degree ‘r’ is given by :

$$R_n(r) = \frac{r}{n} \binom{2n - 1 - r}{n - 1}$$

Below is the implementation of above combinatorics functions using Binomial Coefficient :

C++

```
// CPP code to find the number of ordered trees
// with given number of edges and leaves
#include <bits/stdc++.h>
using namespace std;

// Function returns value of
// Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
    int C[n + 1][k + 1] = { 0 };
    ...
```

```

int i, j;

// Calculate value of Binomial
// Coefficient in bottom up manner
for (i = 0; i <= n; i++) {
    for (j = 0; j <= min(i, k); j++) {

        // Base Cases
        if (j == 0 || j == i)
            C[i][j] = 1;

        // Calculate value using
        // previously stored values
        else
            C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
    }
}

return C[n][k];
}

// Function to calculate the number
// of trees with exactly k leaves.
int k_Leaves(int n, int k)
{
    int ans = (binomialCoeff(n, k) * binomialCoeff(n, k - 1)) / n;
    cout << "Number of trees having 4 edges"
        << " and exactly 2 leaves : " << ans << endl;
    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
int numberOfNodes(int n, int d)
{
    int ans = binomialCoeff(2 * n - 1 - d, n - 1);
    cout << "Number of nodes of degree 1 in"
        << " a tree having 4 edges : " << ans << endl;
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
int rootDegreeR(int n, int r)
{
    int ans = r * binomialCoeff(2 * n - 1 - r, n - 1);
    ans = ans / n;
    cout << "Number of trees having 4 edges"

```

```

        << " where root has degree 2 : " << ans << endl;
    return 0;
}

// Driver program to test above functions
int main()
{
    // Number of trees having 3
    // edges and exactly 2 leaves
    k_Leaves(3, 2);

    // Number of nodes of degree
    // 3 in a tree having 4 edges
    number0fNodes(3, 1);

    // Number of trees having 3
    // edges where root has degree 2
    rootDegreeR(3, 2);

    return 0;
}

```

Java

```

// java code to find the number of ordered
// trees with given number of edges and
// leaves
import java.io.*;

class GFG {

    // Function returns value of
    // Binomial Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int [][]C = new int[n+1] [k+1];
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= Math.min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

```

```

        // Calculate value using
        // previously stored values
        else
            C[i][j] = C[i - 1][j - 1]
                      + C[i - 1][j];
        }
    }

    return C[n][k];
}

// Function to calculate the number
// of trees with exactly k leaves.
static int k_Leaves(int n, int k)
{
    int ans = (binomialCoeff(n, k) *
               binomialCoeff(n, k - 1)) / n;
    System.out.println( "Number of trees "
        + "having 4 edges and exactly 2 "
        + "leaves : " + ans) ;
    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
static int numberOfWorkNodes(int n, int d)
{
    int ans = binomialCoeff(2 * n - 1 - d,
                           n - 1);
    System.out.println("Number of nodes "
        +"of degree 1 in a tree having 4 "
        + "edges : " + ans);
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
static int rootDegreeR(int n, int r)
{
    int ans = r * binomialCoeff(2 * n
                               - 1 - r, n - 1);
    ans = ans / n;
    System.out.println("Number of trees "
        + "having 4 edges where root has"
        + " degree 2 : " + ans);
    return 0;
}

```

```

// Driver program to test above functions

public static void main (String[] args)
{
    // Number of trees having 3
    // edges and exactly 2 leaves
    k_Leaves(3, 2);

    // Number of nodes of degree
    // 3 in a tree having 4 edges
    number0fNodes(3, 1);

    // Number of trees having 3
    // edges where root has degree 2
    rootDegreeR(3, 2);
}
}

// This code is contributed by anuj_67.

```

C#

```

// C# code to find the number of ordered
// trees with given number of edges and
// leaves
using System;

class GFG {

    // Function returns value of
    // Binomial Coefficient C(n, k)
    static int binomialCoeff(int n, int k)
    {
        int [,]C = new int[n+1,k+1];
        int i, j;

        // Calculate value of Binomial
        // Coefficient in bottom up manner
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= Math.Min(i, k); j++)
            {

                // Base Cases
                if (j == 0 || j == i)
                    C[i,j] = 1;

                // Calculate value using

```

```

        // previously stored values
        else
            C[i,j] = C[i - 1,j - 1]
                      + C[i - 1,j];
        }
    }

    return C[n,k];
}

// Function to calculate the number
// of trees with exactly k leaves.
static int k_Leaves(int n, int k)
{
    int ans = (binomialCoeff(n, k) *
               binomialCoeff(n, k - 1)) / n;
    Console.WriteLine( "Number of trees "
                      + "having 4 edges and exactly 2 "
                      + "leaves : " + ans) ;
    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
static int number0fNodes(int n, int d)
{
    int ans = binomialCoeff(2 * n - 1 - d,
                           n - 1);
    Console.WriteLine("Number of nodes "
                      +"of degree 1 in a tree having 4 "
                      + "edges : " + ans);
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
static int rootDegreeR(int n, int r)
{
    int ans = r * binomialCoeff(2 * n
                               - 1 - r, n - 1);
    ans = ans / n;
    Console.WriteLine("Number of trees "
                      + "having 4 edges where root has"
                      + " degree 2 : " + ans);
    return 0;
}

// Driver program to test above functions

```

```
public static void Main ()
{
    // Number of trees having 3
    // edges and exactly 2 leaves
    k_Leaves(3, 2);

    // Number of nodes of degree
    // 3 in a tree having 4 edges
    numberOfNodes(3, 1);

    // Number of trees having 3
    // edges where root has degree 2
    rootDegreeR(3, 2);
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP code to find the number of ordered
// trees with given number of edges and
// leaves

// Function returns value of Binomial
// Coefficient C(n, k)
function binomialCoeff($n, $k)
{
    $C = array(array());
    $i; $j;

    // Calculate value of Binomial
    // Coefficient in bottom up manner
    for ($i = 0; $i <= $n; $i++) {
        for ($j = 0; $j <= min($i, $k); $j++)
        {

            // Base Cases
            if ($j == 0 or $j == $i)
                $C[$i][$j] = 1;

            // Calculate value using
            // previously stored values
            else
                $C[$i][$j] = $C[$i - 1][$j - 1]
```

```

        + $C[$i - 1][$j];
    }
}

return $C[$n][$k];
}

// Function to calculate the number
// of trees with exactly k leaves.
function k_Leaves( $n, $k)
{
    $ans = (binomialCoeff($n, $k) *
            binomialCoeff($n, $k - 1)) / $n;

    echo "Number of trees having 4 edges and ",
         "exactly 2 leaves : " , $ans ,"\n";

    return 0;
}

// Function to calculate total number of
// nodes of degree d in these trees.
function numberOfNodes( $n, $d)
{
    $ans = binomialCoeff(2 * $n - 1 - $d, $n - 1);
    echo "Number of nodes of degree 1 in"
         , " a tree having 4 edges : " , $ans," \n" ;
    return 0;
}

// Function to calculate the number of
// trees in which the root has degree r.
function rootDegreeR( $n, $r)
{
    $ans = $r * binomialCoeff(2 * $n - 1 - $r,
                               $n - 1);
    $ans = $ans / $n;
    echo "Number of trees having 4 edges"
         , " where root has degree 2 : " , $ans ;
    return 0;
}

// Driver program to test above functions
// Number of trees having 3
// edges and exactly 2 leaves
k_Leaves(3, 2);

// Number of nodes of degree

```

```
// 3 in a tree having 4 edges  
numberOfNodes(3, 1);  
  
// Number of trees having 3  
// edges where root has degree 2  
rootDegreeR(3, 2);  
  
// This code is contributed by anuj_67.  
?>
```

Output:

```
Number of trees having 4 edges and exactly 2 leaves : 3  
Number of nodes of degree 1 in a tree having 4 edges : 6  
Number of trees having 4 edges where root has degree 2 : 2
```

Time Complexity : $O(n^k)$.
Auxiliary Space : $O(n^k)$.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/combinatorics-ordered-trees/>

Chapter 24

Common mistakes to be avoided in Competitive Programming in C++ | Beginners

Common mistakes to be avoided in Competitive Programming in C++ | Beginners - Geeks-forGeeks

1. Not using of 1LL or 1ll when needed

```
// A program shows problem if we
// don't use 1ll or 1LL
#include <iostream>
using namespace std;
int main()
{
    int x = 1000000;
    int y = 1000000;

    // This causes overflow even
    // if z is long long int
    long long int z = x*y;

    cout << z;
    return 0;
}
```

Output: -727379968

You might get some other negative value as output. So what is the problem here? The ints are not promoted to long long before multiplication, they remain ints and

their product as well. Then the product is cast to long long, but we are late now as overflow has already occurred. Having one of x or y as long long should work, as the other would be promoted. We can also use 1LL (or 1ll). LL is the suffix for long long, which is 64-bit on most C/C++ implementations. So 1LL, is a 1 of type long long.

```
// C++ program to show that use of 1ll  
// fixes the problem in above code.  
#include <iostream>  
using namespace std;  
int main()  
{  
    int x = 1000000;  
    int y = 1000000;  
  
    long long int z = 1LL*x*y;  
  
    cout << z;  
    return 0;  
}
```

Output: 1000000000000

Here is another place where this trick can help you.

```
// Another problematic code that doesn't  
// use 1LL or 1ll  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    // we should use 1LL or 1ll here  
    // instead of 1. The correct statement  
    // is "long long int z = 1LL << 40;"  
    long long int z = 1 << 40;  
    cout << z;  
    return 0;  
}
```

Output: 0

2. Not using `cin.ignore()` with `getline`

```
// A program that shows problem if we
```

```
// don't use cin.ignore()
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;
    string s;
    for(int i = 0; i<n; ++i)
    {
        getline(cin, s);
        cout << s.length() << " ";
        cout << s << endl;
    }
    return 0;
}
```

Input:

```
4
a b
c d
e f
g h
Output:
0
3 a b
3 c d
3 e f
```

So what is the problem here?

This has little to do with the input you provided yourself but rather with the default behavior getline() exhibits. When you provided your input for the integer n (cin >> n), you not only submitted the following, but also an implicit newline was appended to the stream:

```
"4\n"
```

A newline is always appended to your input when you select Enter or Return when submitting from a terminal. It is also used in files for moving toward the next line. The newline is left in the buffer after the extraction into n until the next I/O operation where it is either discarded or consumed. When the flow of control reaches getline(), the newline will be discarded, but the input will cease immediately. The reason this happens is because the default functionality of this function dictates that it should (it attempts to read a line and stops when it finds a newline).

Because this leading newline inhibits the expected functionality of your program, it follows that it must be skipped or ignored somehow. One option is to call `cin.ignore()` after the first extraction. It will discard the next available character so that the newline is no longer intrusive.

cin.ignore(n, delim);

This extracts characters from the input sequence and discards them, until either n characters have been extracted, or one compares equal to delim.

```
// C++ program to show that use of cin.ignore()
// fixes the problem in above code.
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;
    string s;
    cin.ignore(1, '\n');
    for (int i = 0; i<n; ++i)
    {
        getline(cin, s);
        cout << s.length() << " ";
        cout << s << endl;
    }
    return 0;
}
```

Input:

```
4
a b
c d
e f
g h
Output:
3 a b
3 c d
3 e f
3 g h
```

But what if we don't know how many lines of input are going to be there? We can use this then:

```
// C++ program to handle cases when we
// don't know how many lines of input
// are going to be there
#include <iostream>
```

```
using namespace std;
int main()
{
    string s;
    while (getline(cin, s))
    {
        if (s.empty())
            break;
        cout << s << endl;
    }
    return 0;
}
```

Input:

```
a b
c d
e f
g h
```

Output:

```
a b
c d
e f
g h
```

3. **A problem when taking remainders:** In a lot of problems, you have to print your solution modulo some large prime (for example $10^9 + 7$).

```
// Below program shows problem if we
// don't use don't take remainders
// properly.
#include <iostream>
#define mod 1000000007
using namespace std;

int main()
{
    long long int x, y, z;
    cin >> x >> y >> z;
    z = (z + x*y)%mod; // not good practice
    cout << z;
    return 0;
}
```

Since $z + x*y$ might not fit into long long, the above code can cause problems. The better way is to do this:

```
// Program to demonstrate proper
// ways of taking remainders.
#include <iostream>
#define mod 1000000007
using namespace std;

int main()
{
    long long int x, y, z;
    cin >> x >> y >> z;

    // good practice
    z = ((z%mod) + ((x%mod)*(y%mod))%mod) % mod;
    cout << z;
    return 0;
}
```

This can save you a lot Wrong Answers so it is better to take mod after every computation that might exceed long long. The test cases are generally designed to make sure you have handled overflow cases properly.

Why does this work?

Because $(z + x*y)\%mod$ is the same as $((z\%mod) + ((x\%mod)*(y\%mod))\%mod)\%mod$.

4. **cin and cout might cause TLE:** In a lot of programs, the cause of TLE is generally based on your algorithm. For example, if $n = 10^6$ and your algorithm runs in $O(n^2)$ then this won't pass a 1 second Time Limit. But say, you have found an algorithm that runs in $O(n)$ for $n = 10^6$. This should pass the 1 second Time Limit.

What if this is failing?

One possible reason is that you are using cin and cout for I/O over multiple test cases. You can use scanf or printf for the same. You can also use some custom Fast I/O function for the same based on something like getchar() and putchar().

scanf and printf are faster than cin and cout. See [this](#) for more details.

Custom Fast I/O functions:

```
// C++ program to demonstrate fast input and output

// use long long x = fast_input(); to read in x
inline long long int fast_input(void)
{
    char t;
    long long int x=0;
    long long int neg=0;
    t = getchar();
    while ((t<48 || t>57) && t!='-')
        t = getchar();
    if (t == '-') //handle negative input
    {
        neg = 1;
        t = getchar();
    }
    if (t >='0' && t <='9')
        x = t - '0';
    else
        x = t - 'A' + 10;
    while ((t>47 && t<58) && t!= '-')
        t = getchar();
    if (t == '-')
        neg = -1;
    if (neg == 1)
        x = -x;
}
```

```
        t = getchar();
    }
    while (t>=48 && t<=57)
    {
        x = (x<<3) + (x<<1) + t - 48;

        // x<<3 means 8*x and x<<1 means 2*x so we
        // have x = 10*x+(t - 48)
        t = getchar();
    }

    if (neg)
        x = -x;
    return x;
}

// use fast_output(x, 0); to print x and a newline
// use fast_output(x, 1); to print x and a ' ' after
// the x
inline void fast_output(long long int x, int mode)
{
    char a[20];
    long long int i=0, j;
    a[0] = '0';
    if (x < 0)
    {
        putchar('-');
        x = -x;
    }
    if (x==0)
        putchar('0');
    while (x)
    {
        // convert each digit to character and
        // store in char array
        a[i++] = x%10 + 48;
        x /= 10;
    }

    // print each character from the array
    for (j=i-1; j>=0; j--)
        putchar(a[j]);

    if (mode == 0)
        putchar('\n');
    else putchar(' ');
}
```

Related Articles :

- [A Better Way To Approach Competitive Programming](#)
- [C++ tricks for competitive programming \(for C++ 11\)](#)
- [Writing C/C++ code efficiently in Competitive programming](#)

Source

<https://www.geeksforgeeks.org/common-mistakes-avoided-competitive-programming-c-beginners/>

Chapter 25

Competitive Programming: Conquering a given problem

Competitive Programming: Conquering a given problem - GeeksforGeeks

Programming is the process of developing and implementing sets of instructions to enable a computer to do a certain task. The programming contests like ACM ICPC, Google CodeJam, and IEEE Extreme etc. are delightful playgrounds for the exploration of intelligence of programmers.

From knowing the contest at first to being at ACM ICPC Amritapuri Regionals, I have learnt a lot and would like to share some tips for the coders to tackle the contest problems.

During a real time contest, teams consisting of three students and one computer are to solve as many of the given problems as possible within 5 hours. The team with the most problems solved wins, where ‘solved’ means producing the right outputs for a set of test inputs (trivial and secret). Though the individual skills of the team members are important, in order to be a top team, it is necessary to make use of synergy within the team.

However, to make full use of a strategy, it is also important that your individual skills are as honed as possible. You do not have to be a genius as practicing can take you quite far. As far as feel, there are three factors crucial for being a good programming team:

1. Knowledge of standard algorithms and the ability to find an appropriate algorithm for every problem in the set
2. Ability to code an algorithm into a working program
3. Having a cooperative strategy with your teammates

What is an Algorithm?

[Algorithm](#) is a step-by-step sequence of instructions for the computer to follow.

To be able to do something competitive in programming contests, you need to know a lot of [well-known algorithms](#) and ability to identify which algorithms is suitable for a particular

problem (if the problem is straightforward), or which combinations or variants of algorithms (if the problem is a bit more complex).

Ability to quickly identify problem types

In all programming contests, there are only three types of problems:

1. I haven't seen this one before.
2. I have seen this type before, but haven't or can't solve it.
3. I have solved this type before.

In programming contests, you will be dealing with a set of problems, rather than a single problem. The ability to quickly identify problems into the above mentioned context classifications (haven't seen, have seen, have solved) will be one of key factor to do well in programming contests. As far as my knowledge and repository, a problem is generally classified amongst given categories:

1. **Mathematics** : Prime Number, Big Integer, Permutation, Number Theory, Factorial, Fibonacci, Sequences, Modulus
2. **Dynamic Programming** : Longest Common Subsequence, Longest Increasing Subsequence, Edit Distance, 0/1 Knapsack, Coin Change, Matrix Chain Multiplication, Max Interval Sum
3. **Graph Traversal** : Flood Fill, Floyd Warshall, MST, Max Bipartite Matching, Network Flow, Articulation Point
4. **Sorting**: Bubble Sort, Quick Sort, Merge Sort, Selection Sort, Radix Sort, Bucket Sort
5. **Searching** : Complete Search, Brute Force, Binary Search
6. **String Processing** : String Matching, Pattern Matching
7. **AdHoc Problems**: Trivial Problems

Ability to analyse your algorithm

You have identified your problem. You think you know how to solve it. The question that you must ask now is simple: Given the maximum input bound (usually given in problem description), can my algorithm, with the complexity that I can compute, pass the time limit given in the programming contest.

Usually, there are more than one way to solve a problem. However, some of them may be incorrect and some of them is not fast enough. However, the rule of thumb is: Brainstorm many possible algorithms – then pick the stupidest that works!

It is useful to memorize the following ordering for quickly determine which algorithm perform better asymptotically: **constant < log n < n < n log n < n^2 < n^3 < 2^n < n!**

Coding the Problem

Follow the principle KISS: Keep it Simple and Smart, keeping all versions working and stepwise refinement of code.

After you have coded with the best algorithm, matching time/space complexity and satisfying the test cases (sample test cases are trivial, so never measure code correctness according to them and try tricky cases too), then submit the solution – '**ACCEPTED**'

Identifying a tricky test case to get the opponent down is as important as solving the problem where your mind works more at the boundary conditions.

To summarise, these are some key points:

1. Read all the problems, select the problem in order : easy to tough (if everyone is solving a selected problem, do have a look at it)
2. Sketch the algorithms, complexity, conditions, data structures and tricky details
3. Brainstorm other possible algorithms (if any) – then pick the stupidest that works
4. Do all the Math and select the best algorithm.
5. Code it, as fast as possible, and it must be correct.
6. Try to break the algorithm – look out for boundary test cases.

The main mantra to succeed a problem is to keep calm, manage your time and have a strategic approach, rest your experience and practice plays the part.

Keep **Practicing !!**

Source

<https://www.geeksforgeeks.org/competitive-programming-conquering-a-given-problem/>

Chapter 26

Container with Most Water

Container with Most Water - GeeksforGeeks

Given n non-negative integers $\{h_1, h_2, \dots, h_n\}$ where each represents a point at coordinate (i, h_i) , n vertical lines are drawn such that the two endpoints of line i is at (i, h_i) and $(i, 0)$.

Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is 2D plane we are working with for simplicity).

Note : You may not slant the container.

Examples :

Input : [1, 5, 4, 3]

Output : 6

Explanation :

5 and 3 are distance 2 apart.

So the size of the base = 2.

Height of container = $\min(5, 3) = 3$.

So total area = $3 * 2 = 6$

Input : [3, 1, 2, 4, 5]

Output : 12

Explanation :

5 and 3 are distance 4 apart.

So the size of the base = 4.

Height of container = $\min(5, 3) = 3$.

So total area = $4 * 3 = 12$

Approach :

Note 1 : When you consider a_1 and a_N , then the area is $(N-1) * \min(a_1, a_N)$.

Note 2 : The base $(N-1)$ is the maximum possible.

- This implies that if there was a better solution possible, it will definitely have the Height greater than $\min(a_1, a_N)$.
~~Base & Height > ($N - 1$) * min(a_1, a_N)~~
- We know that, Base $\min(a_1, a_N)$
 This means that we can discard $\min(a_1, a_N)$ from our set and look to solve this problem again from the start.
- If $a_1 < a_N$, then the problem reduces to solving the same thing for a_2, a_N .
- Else, it reduces to solving the same thing for a_1, a_{N-1}

C++

```
// C++ code for Max
// Water Container
#include<iostream>
using namespace std;

int maxArea(int A[], int len)
{
    int l = 0;
    int r = len - 1;
    int area = 0;

    while (l < r)
    {
        // Calculating the max area
        area = max(area, min(A[l],
                              A[r]) * (r - l));

        if (A[l] < A[r])
            l += 1;

        else
            r -= 1;
    }
    return area;
}

// Driver code
int main()
{
```

```
int a[] = {1, 5, 4, 3};  
int b[] = {3, 1, 2, 4, 5};  
  
int len1 = sizeof(a) / sizeof(a[0]);  
cout << maxArea(a, len1);  
  
int len2 = sizeof(b) / sizeof(b[0]);  
cout << endl << maxArea(b, len2);  
}  
  
// This code is contributed by Smitha Dinesh Semwal
```

Java

```
// Java code for Max  
// Water Container  
import java.util.*;  
  
class Area{  
  
    public static int maxArea(int A[], int len)  
    {  
        int l = 0;  
        int r = len - 1;  
        int area = 0;  
  
        while (l < r)  
        {  
            // Calculating the max area  
            area = Math.max(area,  
                            Math.min(A[l], A[r]) * (r - l));  
  
            if (A[l] < A[r])  
                l += 1;  
  
            else  
                r -= 1;  
        }  
        return area;  
    }  
  
    public static void main(String[] args)  
    {  
        int a[] = {1, 5, 4, 3};  
        int b[] = {3, 1, 2, 4, 5};  
  
        int len1 = 4;  
        System.out.print( maxArea(a, len1)+"\n" );  
    }  
}
```

```
        int len2 = 5;
        System.out.print( maxArea(b, len2) );
    }
}

// This code is contributed by rishabh_jain
```

Python

```
# Python code for Max
# Water Container
def maxArea( A):
    l = 0
    r = len(A) -1
    area = 0

    while l < r:
        # Calculating the max area
        area = max(area, min(A[l],
                              A[r]) * (r - l))

        if A[l] < A[r]:
            l += 1
        else:
            r -= 1
    return area

# Driver code
a = [1, 5, 4, 3]
b = [3, 1, 2, 4, 5]

print(maxArea(a))
print(maxArea(b))
```

C#

```
// C# code for Max
// Water Container
using System;

class Area{

    public static int maxArea(int []A, int len)
    {
        int l = 0;
        int r = len -1;
```

```
int area = 0;

while (l < r)
{
    // Calculating the max area
    area = Math.Max(area,
                    Math.Min(A[l], A[r]) * (r - l));

    if (A[l] < A[r])
        l += 1;

    else
        r -= 1;
}
return area;
}

// Driver code
public static void Main()
{
    int []a = {1, 5, 4, 3};
    int []b = {3, 1, 2, 4, 5};

    int len1 = 4;
    Console.WriteLine( maxArea(a, len1));

    int len2 = 5;
    Console.WriteLine( maxArea(b, len2) );
}
}

// This code is contributed by Vt_M
```

PHP

```
<?php
// PHP code for Max
// Water Container
function maxArea($A, $len)
{
    $l = 0;
    $r = $len - 1;
    $area = 0;

    while ($l < $r)
    {
        // Calculating the max area
        $area = max($area, min($A[$l],
```

```
$A[$r]) * ($r - $l));  
  
if ($A[$l] < $A[$r])  
    $l += 1;  
  
else  
    $r -= 1;  
}  
return $area;  
}  
  
// Driver code  
$a = array(1, 5, 4, 3);  
$b = array(3, 1, 2, 4, 5);  
  
$len1 = sizeof($a) / sizeof($a[0]);  
echo maxArea($a, $len1). "\n";  
  
$len2 = sizeof($b) / sizeof($b[0]);  
echo maxArea($b, $len2);  
  
// This code is contributed by mits  
?>
```

Output :

```
6  
12
```

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/container-with-most-water/>

Chapter 27

Count inversions of size k in a given array

Count inversions of size k in a given array - GeeksforGeeks

Given an array of n distinct integers a_1, a_2, \dots, a_n and an integer k. Find out the number of sub-sequences of a such that $a_1 > a_2 > \dots > a_k$, and $i < j_1 < j_2 < \dots < j_k < n$. In other words output the total number of inversions of length k.

Examples:

Input : a[] = {9, 3, 6, 2, 1}, k = 3

Output : 7

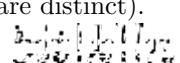
The seven inversions are {9, 3, 2}, {9, 3, 1}, {9, 6, 2}, {9, 6, 1}, {9, 2, 1}, {3, 2, 1} and {6, 2, 1}.

Input : a[] = {5, 6, 4, 9, 2, 7, 1}, k = 4

Output : 2

The two inversions are {5, 4, 2, 1}, {6, 4, 2, 1}.

We have already discussed counting inversion of length three [here](#). This post will generalise the approach using [Binary Indexed Tree](#) and [Counting inversions using BIT](#). More on Binary Indexed Trees can be found from the actual paper that Peter M. Fenwick published, [here](#).

1. To begin with, we first convert the given array to a permutation of elements $1, 2, 3, \dots, n$ (Note that this is always possible since the elements are distinct).
2. The approach is to maintain a set of k Fenwick Trees. Let it be denoted by 

where $\text{BIT}[l][x]$, keeps track of the number of l – length sub-sequences that start with x .

3. We iterate from the end of the converted array to the beginning. For every converted array element x , we update the Fenwick tree set, as: For each $i \leq l \leq k$, each sub-sequence of length l that start with a number less than x is also a part of sequences of length $k+1-l$.

The final result is found by sum of occurrences of $\text{BIT}[k][x]$.

The implementation is discussed below:

C++

```
// C++ program to count inversions of size k using
// Binary Indexed Tree
#include <bits/stdc++.h>
using namespace std;

// It is beneficial to declare the 2D BIT globally
// since passing it into functions will create
// additional overhead
const int K = 51;
const int N = 100005;
int BIT[K][N] = { 0 };

// update function. "t" denotes the t'th Binary
// indexed tree
void updateBIT(int t, int i, int val, int n)
{
    // Traversing the t'th BIT
    while (i <= n) {
        BIT[t][i] = BIT[t][i] + val;
        i = i + (i & (-i));
    }
}

// function to get the sum.
// "t" denotes the t'th Binary indexed tree
int getSum(int t, int i)
{
    int res = 0;

    // Traversing the t'th BIT
    while (i > 0) {
        res = res + BIT[t][i];
        i = i - (i & (-i));
    }
}
```

```

        }
        return res;
    }

// Converts an array to an array with values from 1 to n
// and relative order of smaller and greater elements
// remains same. For example, {7, -90, 100, 1} is
// converted to {3, 1, 4, 2 }
void convert(int arr[], int n)
{
    // Create a copy of arr[] in temp and sort
    // the temp array in increasing order
    int temp[n];
    for (int i = 0; i < n; i++)
        temp[i] = arr[i];
    sort(temp, temp + n);

    // Traverse all array elements
    for (int i = 0; i < n; i++) {

        // lower_bound() Returns pointer to the
        // first element greater than or equal
        // to arr[i]
        arr[i] = lower_bound(temp, temp + n,
                             arr[i]) - temp + 1;
    }
}

// Returns count of inversions of size three
int getInvCount(int arr[], int n, int k)
{
    // Convert arr[] to an array with values from
    // 1 to n and relative order of smaller and
    // greater elements remains same. For example,
    // {7, -90, 100, 1} is converted to {3, 1, 4, 2 }
    convert(arr, n);

    // iterating over the converted array in
    // reverse order.
    for (int i = n - 1; i >= 0; i--) {
        int x = arr[i];

        // update the BIT for l = 1
        updateBIT(1, x, 1, n);

        // update BIT for all other BITS
        for (int l = 1; l < k; l++) {
            updateBIT(l + 1, x, getSum(l, x - 1), n);
    }
}

```

```
        }
    }

    // final result
    return getSum(k, n);
}

// Driver program to test above function
int main()
{
    int arr[] = { 5, 6, 4, 9, 3, 7, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 4;
    cout << "Inversion Count : " << getInvCount(arr, n, k);
    return 0;
}
```

Java

```
// Java program to count
// inversions of size k using
// Binary Indexed Tree
import java.io.*;
import java.util.Arrays;
import java.util.ArrayList;
import java.lang.*;
import java.util.Collections;

class GFG
{

    // It is beneficial to declare
    // the 2D BIT globally since
    // passing it into functions
    // will create additional overhead
    static int K = 51;
    static int N = 100005;
    static int BIT[][] = new int[K][N];

    // update function. "t" denotes
    // the t'th Binary indexed tree
    static void updateBIT(int t, int i,
                          int val, int n)
    {

        // Traversing the t'th BIT
        while (i <= n)
        {
```

```
        BIT[t][i] = BIT[t][i] + val;
        i = i + (i & (-i));
    }
}

// function to get the sum.
// "t" denotes the t'th
// Binary indexed tree
static int getSum(int t, int i)
{
    int res = 0;

    // Traversing the t'th BIT
    while (i > 0)
    {
        res = res + BIT[t][i];
        i = i - (i & (-i));
    }
    return res;
}

// Converts an array to an
// array with values from
// 1 to n and relative order
// of smaller and greater
// elements remains same.
// For example, {7, -90, 100, 1}
// is converted to {3, 1, 4, 2 }
static void convert(int arr[], int n)
{
    // Create a copy of arr[] in
    // temp and sort the temp
    // array in increasing order
    int temp[] = new int[n];
    for (int i = 0; i < n; i++)
        temp[i] = arr[i];
    Arrays.sort(temp);

    // Traverse all array elements
    for (int i = 0; i < n; i++)
    {

        // lower_bound() Returns
        // pointer to the first
        // element greater than
        // or equal to arr[i]
        arr[i] = Arrays.binarySearch(temp,
                                     arr[i]) + 1;
    }
}
```

```
        }
    }

// Returns count of inversions
// of size three
static int getInvCount(int arr[],
                      int n, int k)
{

    // Convert arr[] to an array
    // with values from 1 to n and
    // relative order of smaller
    // and greater elements remains
    // same. For example, {7, -90, 100, 1}
    // is converted to {3, 1, 4, 2 }
    convert(arr, n);

    // iterating over the converted
    // array in reverse order.
    for (int i = n - 1; i >= 0; i--)
    {
        int x = arr[i];

        // update the BIT for l = 1
        updateBIT(1, x, 1, n);

        // update BIT for all other BITs
        for (int l = 1; l < k; l++)
        {
            updateBIT(l + 1, x,
                      getSum(l, x - 1), n);
        }
    }

    // final result
    return getSum(k, n);
}

// Driver Code
public static void main(String[] args)
{
    int arr[] = { 5, 6, 4, 9,
                 3, 7, 2, 1 };
    int n = arr.length;
    int k = 4;
    System.out.println("Inversion Count : " +
                       getInvCount(arr, n, k));
}
```

```
}
```

```
}
```

Output:

Inversion Count : 11

Time Complexity $\mathcal{O}(n^2 \log(n))$

Auxiliary Space $\mathcal{O}(n^2)$

It should be noted that this is not the only approach to solve the problem of finding k-inversions. Obviously, any problem solvable by BIT is also solvable by Segment Tree. Besides, we can use Merge-Sort based algorithm, and C++ policy based data structure too. Also, at the expense of higher time complexity, Dynamic Programming approach can also be used.

Source

<https://www.geeksforgeeks.org/count-inversions-of-size-k-in-a-given-array/>

Chapter 28

Count number of primes in an array

Count number of primes in an array - GeeksforGeeks

Given an array arr[] of N positive integers. The task is to write a program to count the number of prime elements in the given array.

Examples:

```
Input: arr[] = {1, 3, 4, 5, 7}
Output: 3
There are three primes, 3, 5 and 7
```

```
Input: arr[] = {1, 2, 3, 4, 5, 6, 7}
Output: 4
```

Naive Approach: A simple solution is to traverse the array and keep [checking for every element if it is prime](#) or not and keep the count of the prime elements at the same time.

Efficient Approach: Generate all primes upto maximum element of the array using [sieve of Eratosthenes](#) and store them in a hash. Now traverse the array and find the count of those elements which are prime using the hash table.

Below is the implementation of above approach:

C++

```
// CPP program to find count of
// primes in given arary.
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find count of prime
int primeCount(int arr[], int n)
{
    // Find maximum value in the array
    int max_val = *max_element(arr, arr+n);

    // USE SIEVE TO FIND ALL PRIME NUMBERS LESS
    // THAN OR EQUAL TO max_val
    // Create a boolean array "prime[0..n]". A
    // value in prime[i] will finally be false
    // if i is Not a prime, else true.
    vector<bool> prime(max_val + 1, true);

    // Remaining part of SIEVE
    prime[0] = false;
    prime[1] = false;
    for (int p = 2; p * p <= max_val; p++) {

        // If prime[p] is not changed, then
        // it is a prime
        if (prime[p] == true) {

            // Update all multiples of p
            for (int i = p * 2; i <= max_val; i += p)
                prime[i] = false;
        }
    }

    // Find all primes in arr[]
    int count = 0;
    for (int i = 0; i < n; i++)
        if (prime[arr[i]])
            count++;

    return count;
}

// Driver code
int main()
{

    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << primeCount(arr, n);

    return 0;
}
```

}

PHP

Output:

4

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/count-number-of-primes-in-an-array/>

Chapter 29

Count number of right triangles possible with a given perimeter

Count number of right triangles possible with a given perimeter - GeeksforGeeks

Given a perimeter P, the task is to find the number of right triangles possible with perimeter equal to p.

Examples:

Input: P = 12
Output: number of right triangles = 1
The only right angle possible is with sides
hypotenuse = 5, perpendicular = 4 and base = 3.

Input: p = 840
Output: number of right triangles = 8

So the aim is to find the number of solutions which satisfy equations $a + b + c = p$ and $a^2 + b^2 = c^2$.

A **naive** approach is to run two loops for a (1 to $p/2$) and b ($a+1$ to $p/3$) then make $c=p-a-b$ and increase count by one if $a^2 + b^2 = c^2$. This will take $\mathcal{O}(n^2)$ time.

An **efficient** approach can be found by little algebraic manipulation :

$$\begin{aligned}
 & \text{Let } p = a + b + c \\
 & \text{Then } a^2 + b^2 = c^2 \\
 & \text{or, } 2ab = p^2 - 2c^2 \\
 & \text{or, } 2ab = p^2 - 2(p-a-b)^2 \\
 & \text{or, } 2ab = 2p^2 - 2(p-a-b)^2 - 2a^2 - 2b^2 \\
 & \text{or, } 2ab = 2p^2 - 2(p-a-b)^2 - 2(a+b)^2
 \end{aligned}$$

Since $a + c > b$ or, $p - b > b$ or, $b < p/2$. Thus iterating b from 1 to $p/2$, calculating a and storing only the whole number a would give all solutions for a given p. There are no right triangles are possible for odd p as right angle triangles follow the [Pythagoras theorem](#). Use a list of pairs to store the values of a and band return the count at the end.

Below is the implementation of the above approach.

```

# python program to find the number of
# right triangles with given perimeter

# Function to return the count
def countTriangles(p):

    # making a list to store (a, b) pairs
    store = []

    # no triangle if p is odd
    if p % 2 != 0 : return 0
    else :
        count = 0
        for b in range(1, p // 2):

            a = p / 2 * ((p - 2 * b) / (p - b))
            inta = int(a)
            if (a == inta ):

                # make (a, b) pair in sorted order
                ab = tuple(sorted((inta, b)))

                # check to avoid duplicates
                if ab not in store :

```

```
count += 1
# store the new pair
store.append(ab)
return count

# Driver Code
p = 12
print("number of right triangles = "+str(countTriangles(p)))
```

Output:

```
number of right triangles = 8
```

Time complexity: O(P)

Source

<https://www.geeksforgeeks.org/count-number-of-right-triangles-possible-with-a-given-perimeter/>

Chapter 30

Count strings with consonants and vowels at alternate position

Count strings with consonants and vowels at alternate position - GeeksforGeeks

Given a string **str**. The task is to find all possible number of strings that can be obtained by replacing the “\$” with alphabets in the given string.

Note: Alphabets should be placed in such a way that the string is always alternating in vowels and consonants, and the string must always start with a consonant. It is assumed that such a string is always possible, i.e. there is no need to care about the characters other than “\$”.

Examples:

Input: str = "y\$s"
Output: 5
\$ can be replaced with any of the 5 vowels.
So, there can be 5 strings.

Input: str = "s\$\$e\$"
Output: 2205

Approach: It is given that the string will start with a consonant. So, if ‘\$’ is at even position(considering 0-based indexing) then there should be a consonant else there should be a vowel. Also, given that there is no need to care about the characters other than “\$”, i.e., characters other than “\$” are placed correctly in the string maintaining the alternating consonant and vowel sequence. Let us understand the problem with an example.

str = “s\$\$e\$”

Here we have to find the number of ways to form a string with given constraints.

- First occurrence of \$ is at 2nd position i.e. 1st index, so we can use 5 vowels.

- Second occurrence of \$ is at 3rd position, so we can use 21 consonants.
- Third occurrence of \$ is at 5th position, so we can use 21 consonants.

So, total number of ways to form above string is = **5*21*21 = 2205**

Below is the implementation of the above approach:

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the count of strings
int countStrings(string s)
{
    // Variable to store the final result
    long sum = 1;

    // Loop iterating through string
    for (int i = 0; i < s.size(); i++) {

        // If '$' is present at the even
        // position in the string
        if (i % 2 == 0 && s[i] == '$')

            // 'sum' is multiplied by 21
            sum *= 21;

        // If '$' is present at the odd
        // position in the string
        else if (s[i] == '$')

            // 'sum' is multiplied by 5
            sum *= 5;
    }

    return sum;
}

// Driver code
int main()
{
    // Let the string 'str' be s$$e$
    string str = "s$$e$";

    // Print result
    cout << countStrings(str) << endl;
    return 0;
}
```

Output:

2205

Source

<https://www.geeksforgeeks.org/count-strings-with-consonants-and-vowels-at-alternate-position/>

Chapter 31

Count unique subsequences of length K

Count unique subsequences of length K - GeeksforGeeks

Given an array of N numbers and an integer K. The task is to print the number of unique subsequences possible of length K.

Examples:

```
Input : a[] = {1, 2, 3, 4}, k = 3
Output : 4.
Unique Subsequences are:
{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}
```

```
Input: a[] = {1, 1, 1, 2, 2, 2 }, k = 3
Output : 4
Unique Subsequences are
{1, 1, 1}, {1, 1, 2}, {1, 2, 2}, {2, 2, 2}
```

Approach: There is a well-known formula how many subsequences of fixed length K can be chosen from N unique objects. But the problem here has several differences. One among them is the order in subsequences is important and must be preserved as in the original sequence. For such a problem there can be no ready combinatorics formula because the results depend on the order of the original array.

The main idea is to deal recurrently by the length of the subsequence. On each recurrent step, move from the end to the beginning and count the unique combinations using the count of shorter unique combinations from the previous step. More strictly on every step j we keep an array of length N and every element in the place p means how many unique subsequences with length j we found to the right of the element in place i, including i itself.

Below is the implementation of the above approach.

```
#include <bits/stdc++.h>
using namespace std;

// Function which returns the number of
// unique subsequences of length K
int solution(vector<int>& A, int k)
{
    // size of the vector
    // which does is constant
    const int N = A.size();

    // bases cases
    if (N < k || N < 1 || k < 1)
        return 0;
    if (N == k)
        return 1;

    // Prepare arrays for recursion
    vector<int> v1(N, 0);
    vector<int> v2(N, 0);
    vector<int> v3(N, 0);

    // initiate separately for k = 1
    // initiate the last element
    v2[N - 1] = 1;
    v3[A[N - 1] - 1] = 1;

    // initiate all other elements of k = 1
    for (int i = N - 2; i >= 0; i--) {

        // initialize the front element
        // to vector v2
        v2[i] = v2[i + 1];

        // if element v[a[i]-1] is 0
        // then increment it in vector v2
        if (v3[A[i] - 1] == 0) {
            v2[i]++;
            v3[A[i] - 1] = 1;
        }
    }

    // iterate for all possible values of K
    for (int j = 1; j < k; j++) {

        // fill the vectors with 0
        fill(v3.begin(), v3.end(), 0);
```

```

// fill(v1.begin(), v1.end(), 0)
// the last must be 0 as from last no unique
// subarray can be formed
v1[N - 1] = 0;

// Iterate for all index from which unique
// subsequences can be formed
for (int i = N - 2; i >= 0; i--) {

    // add the number of subsequence formed
    // from the next index
    v1[i] = v1[i + 1];

    // start with combinations on the
    // next index
    v1[i] = v1[i] + v2[i + 1];

    // Remove the elements which have
    // already been counted
    v1[i] = v1[i] - v3[A[i] - 1];

    // Update the number used
    v3[A[i] - 1] = v2[i + 1];
}

// prepare the next iteration
// by filling v2 in v1
v2 = v1;
}

// last answer is stored in v1
return v1[0];
}

// Function to push the vector into an array
// and print all the unique subarrays
void solve(int a[], int n, int k)
{
    vector<int> v;

    // fill the vector with a[]
    v.assign(a, a + n);

    // Function call to print the count
    // of unique subsequences of size K
    cout << solution(v, k);
}

```

```
// Driver Code
int main()
{
    int a[] = { 1, 2, 3, 4 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 3;
    solve(a, n, k);

    return 0;
}
```

Output:

4

Source

<https://www.geeksforgeeks.org/count-unique-subsequences-of-length-k/>

Chapter 32

Data Type Ranges and their macros in C++

Data Type Ranges and their macros in C++ - GeeksforGeeks

Most of the times, in competitive programming, there is a need to assign the variable, the maximum or minimum value that data type can hold, but remembering such a large and precise number comes out to be a difficult job. Therefore, C++ has certain macros to represent these numbers, so that these can be directly assigned to the variable without actually typing the whole number. List of some of them are mentioned below.

Data Type	Range	Macro for min value
char	-128 to +127	CHAR_MIN
short char	-128 to +127	SCHAR_MIN
unsigned char	0 to 255	0
short int	-32768 to +32767	SHRT_MIN
unsigned short int	0 to 65535	0
int	-2147483648 to +2147483647	INT_MIN
unsigned int	0 to 4294967295	0
long int	-9223372036854775808 to +9223372036854775807	LONG_MIN
unsigned long int	0 to 18446744073709551615	0
long long int	-9223372036854775808 to +9223372036854775807	LLONG_MIN
unsigned long long int	0 to 18446744073709551615	0
float	1.17549e-38 to 3.40282e+38	FLT_MIN
float(negative)	-1.17549e-38 to -3.40282e+38	-FLT_MIN
double	2.22507e-308 to 1.79769e+308	DBL_MIN
double(negative)	-2.22507e-308 to -1.79769e+308	-DBL_MIN

```
// C++ code to demonstrate the macros for data types
```

```
#include<iostream>
#include<limits.h> // for int,char macros
#include<float.h> // for float,double macros
using namespace std;
int main()
{
    // Displaying ranges with the help of macros
    cout << "char ranges from : " << CHAR_MIN << " to " << CHAR_MAX;
    cout << "\n\nshort char ranges from : " << SCHAR_MIN << " to " << SCHAR_MAX;
    cout << "\n\nunsigned char ranges from : " << 0 << " to " << UCHAR_MAX;

    cout << "\n\n\nshort int ranges from : " << SHRT_MIN << " to " << SHRT_MAX;
    cout << "\n\nunsigned short int ranges from : " << 0 << " to " << USHRT_MAX;
    cout << "\n\nint ranges from : " << INT_MIN << " to " << INT_MAX;
    cout << "\n\nunsigned int ranges from : " << 0 << " to " << UINT_MAX;
    cout << "\n\nlong int ranges from : " << LONG_MIN << " to " << LONG_MAX;
    cout << "\n\nunsigned long int ranges from : " << 0 << " to " << ULONG_MAX;
    cout << "\n\nlong long int ranges from : " << LLONG_MIN << " to " << LLONG_MAX;
    cout << "\n\nunsigned long long int ranges from : " << 0 << " to " << ULLONG_MAX;

    cout << "\n\n\nfloat ranges from : " << FLT_MIN << " to " << FLT_MAX;
    cout << "\n\nnegative float ranges from : " << -FLT_MIN << " to " << -FLT_MAX;
    cout << "\n\ndouble ranges from : " << DBL_MIN << " to " << DBL_MAX;
    cout << "\n\nnegative double ranges from : " << -DBL_MIN << " to " << +DBL_MAX;

    return 0;
}
```

Output:

```
char ranges from : -128 to 127

short char ranges from : -128 to 127

unsigned char ranges from : 0 to 255

short int ranges from : -32768 to 32767

unsigned short int ranges from : 0 to 65535

int ranges from : -2147483648 to 2147483647

unsigned int ranges from : 0 to 4294967295
```

```
long int ranges from : -9223372036854775808 to 9223372036854775807
unsigned long int ranges from : 0 to 18446744073709551615
long long int ranges from : -9223372036854775808 to 9223372036854775807
unsigned long long int ranges from : 0 to 18446744073709551615

float ranges from : 1.17549e-38 to 3.40282e+38
negative float ranges from : -1.17549e-38 to -3.40282e+38
double ranges from : 2.22507e-308 to 1.79769e+308
negative double ranges from : -2.22507e-308 to 1.79769e+308
```

Source

<https://www.geeksforgeeks.org/data-type-ranges-and-their-macros-in-c/>

Chapter 33

Difference between the summation of numbers whose frequency of all digits are same and different

Difference between the summation of numbers whose frequency of all digits are same and different - GeeksforGeeks

Given an array of N integers, find the difference between the summation of numbers whose frequency of all digits are same and different. For e.g. 8844, 1001, 56, 77, 34764673 are the examples of numbers where all digits have the same frequency. Similarly, 545, 44199, 76672, 202 are the examples of numbers whose frequency of digits are not the same.

Examples:

Input: $a[] = \{24, 787, 2442, 101, 1212\}$
Output: 2790
 $(2442 + 24 + 1212) - (787 + 101) = 2790$

Input: $a[] = \{12321, 786786, 110022, 47, 22895\}$
Output: 861639

Approach: Traverse for every element in the array. Keep a count of all digits in a map. Once all digits are traversed in the element, check if the map contains the same frequency for all digits. If it contains the same frequency, then add it to same else add it to diff. After the array has been traversed completely, return the difference of both the elements.

Below is the implementation of the above approach:

```
// C++ Difference between the
// summation of numbers
```

```
// in which the frequency of
// all digits are same and different
#include <bits/stdc++.h>
using namespace std;

// Function that returns the difference
int difference(int a[], int n)
{

    // Stores the sum of same
    // and different frequency digits
    int same = 0;
    int diff = 0;

    // traverse in the array
    for (int i = 0; i < n; i++) {
        // duplicate of array element
        int num = a[i];
        unordered_map<int, int> mp;

        // traverse for every digit
        while (num) {
            mp[num % 10]++;
            num = num / 10;
        }

        // iterator pointing to the
        // first element in the array
        auto it = mp.begin();

        // count of the smallest digit
        int freqdigit = (*it).second;
        int flag = 0;

        // check if all digits have same frequency or not
        for (auto it = mp.begin(); it != mp.end(); it++) {
            if ((*it).second != freqdigit) {
                flag = 1;
                break;
            }
        }

        // add to diff if not same
        if (flag)
            diff += a[i];
        else
            same += a[i];
    }
}
```

```
        return same - diff;  
    }  
  
    // Driver Code  
    int main()  
    {  
        int a[] = { 24, 787, 2442, 101, 1212 };  
        int n = sizeof(a) / sizeof(a[0]);  
        cout << difference(a, n);  
        return 0;  
    }
```

Output:

2790

Source

<https://www.geeksforgeeks.org/difference-between-the-summation-of-numbers-whose-frequency-of-all-digits-are-same-and-different/>

Chapter 34

Digit DP | Introduction

Digit DP | Introduction - GeeksforGeeks

Prerequisite : [How to solve a Dynamic Programming Problem ?](#)

There are many types of problems that ask to count the number of integers ‘x’ between two integers say ‘a’ and ‘b’ such that x satisfies a specific property that can be related to its digits.

So, if we say $G(x)$ tells the number of such integers between 1 to x (inclusively), then the number of such integers between a and b can be given by $G(b) - G(a-1)$. This is when Digit DP (Dynamic Programming) comes into action. All such integer counting problems that satisfy the above property can be solved by digit DP approach.

Key Concept

- Let given number x has n digits. The main idea of digit DP is to first represent the digits as an array of digits $t[]$. Let's say a we have $t_n t_{n-1} t_{n-2} \dots t_2 t_1$ as the decimal representation where t_i ($0 < i \leq n$) tells the i-th digit from the right. The leftmost digit t_n is the most significant digit.
- Now, after representing the given number this way we generate the numbers less than the given number and simultaneously calculate using DP, if the number satisfy the given property. We *start generating integers having number of digits = 1 and then till number of digits = n. Integers having less number of digits than n can be analyzed by setting the leftmost digits to be zero.*

Example Problem :

Given to integers **a** and **b**. Your task is to print the sum of all the digits appearing in the integers between a and b.

For example if $a = 5$ and $b = 11$, then answer is 38 ($5 + 6 + 7 + 8 + 9 + 1 + 0 + 1 + 1$)

Constraints : $1 \leq a < b \leq 10^{18}$

Now we see that if we have calculated the answer for state having $n-1$ digits, i.e., $t_{n-1} t_{n-2} \dots t_2 t_1$ and we need to calculate answer for state having n digit $t_n t_{n-1} t_{n-2} \dots t_2 t_1$. So,

clearly, we can use the result of the previous state instead of re-calculating it. Hence, it follows the overlapping property.

Let's think for a state for this DP

Our DP state will be **dp(idx, tight, sum)**

1) idx

- It tells about the index value from right in the given integer

2) tight

- This will tell if the current digits range is restricted or not. If the current digit's range is not restricted then it will span from 0 to 9 (inclusively) else it will span from 0 to digit[idx] (inclusively).

Example: consider our limiting integer to be 3245 and we need to calculate G(3245)
 index : 4 3 2 1
 digits : 3 2 4 5

Unrestricted range:

Now suppose the integer generated till now is : 3 1 * * (* is empty place, where digits are to be inserted to form the integer).

```
index : 4 3 2 1
digits : 3 2 4 5
generated integer: 3 1 _ _
```

here, we see that index 2 has unrestricted range. Now index 2 can have digits from range 0 to 9(inclusively).

For unrestricted range **tight = 0**

Restricted range:

Now suppose the integer generated till now is : 3 2 * * (* is an empty place, where digits are to be inserted to form the integer).

```
index : 4 3 2 1
digits : 3 2 4 5
generated integer: 3 2 _ _
```

here, we see that index 2 has a restricted range. Now index 2 can only have digits from range 0 to 4 (inclusively)

For unrestricted range **tight = 1**

3) sum

- This parameter will store the sum of digits in the generated integer from msd to idx.
- Max value for this parameter sum can be $9 \times 18 = 162$, considering 18 digits in the integer

State Relation

The basic idea for state relation is very simple. We formulate the dp in top-down fashion. Let's say we are at the **msd** having index idx. So initially sum will be 0.

Therefore, we will fill the digit at index by the digits in its range. Let's say its range is from 0 to k ($k \leq 9$, depending on the tight value) and fetch the answer from the next state having index = idx-1 and sum = previous sum + digit chosen.

```
int ans = 0;
for (int i=0; i<=k; i++) {
    ans += state(idx-1, newTight, sum+i)
}

state(idx,tight,sum) = ans;
```

How to calculate the newTight value?

The new tight value from a state depends on its previous state. If tight value form the previous state is 1 and the digit at idx chosen is $\text{digit}[idx]$ (i.e the digit at idx in limiting integer) , then only our new tight will be 1 as it only then tells that the number formed till now is prefix of the limiting integer.

```
// digitTaken is the digit chosen
// digit[idx] is the digit in the limiting
//           integer at index idx from right
// previousTight is the tight value form previous
//           state

newTight = previousTight & (digitTake == digit[idx])
```

C++ code for the above implementation:

```
// Given two integers a and b. The task is to print
// sum of all the digits appearing in the
// integers between a and b
#include "bits/stdc++.h"
using namespace std;

// Memoization for the state results
long long dp[20][180][2];
```

```

// Stores the digits in x in a vector digit
long long getDigits(long long x, vector <int> &digit)
{
    while (x)
    {
        digit.push_back(x%10);
        x /= 10;
    }
}

// Return sum of digits from 1 to integer in
// digit vector
long long digitSum(int idx, int sum, int tight,
                    vector <int> &digit)
{
    // base case
    if (idx == -1)
        return sum;

    // checking if already calculated this state
    if (dp[idx][sum][tight] != -1 and tight != 1)
        return dp[idx][sum][tight];

    long long ret = 0;

    // calculating range value
    int k = (tight)? digit[idx] : 9;

    for (int i = 0; i <= k ; i++)
    {
        // calculating newTight value for next state
        int newTight = (digit[idx] == i)? tight : 0;

        // fetching answer from next state
        ret += digitSum(idx-1, sum+i, newTight, digit);
    }

    if (!tight)
        dp[idx][sum][tight] = ret;
}

// Returns sum of digits in numbers from a to b.
int rangeDigitSum(int a, int b)
{
    // initializing dp with -1
}

```

```

        memset(dp, -1, sizeof(dp));

        // storing digits of a-1 in digit vector
        vector<int> digitA;
        getDigits(a-1, digitA);

        // Finding sum of digits from 1 to "a-1" which is passed
        // as digitA.
        long long ans1 = digitSum(digitA.size()-1, 0, 1, digitA);

        // Storing digits of b in digit vector
        vector<int> digitB;
        getDigits(b, digitB);

        // Finding sum of digits from 1 to "b" which is passed
        // as digitB.
        long long ans2 = digitSum(digitB.size()-1, 0, 1, digitB);

        return (ans2 - ans1);
    }

    // driver function to call above function
    int main()
    {
        long long a = 123, b = 1024;
        cout << "digit sum for given range : "
            << rangeDigitSum(a, b) << endl;
        return 0;
    }
}

```

Output:

```
digit sum for given range : 12613
```

Time Complexity:

There are total $\text{idx} \times \text{sum} \times \text{tight}$ states and we are performing 0 to 9 iterations to visit every state. Therefore, The Time Complexity will be **$O(10 \times \text{idx} \times \text{sum} \times \text{tight})$** . Here, we observe that **tight** = 2 and **idx** can be max 18 for 64 bit unsigned integer and moreover, the sum will be max $9 \times 18 \sim 200$. So, overall we have $10 \times 18 \times 200 \times 2 \sim 10^5$ iterations which can be easily executed in **0.01 seconds**.

The above problem can also be solved using simple recursion without any memoization. The recursive solution for the above problem can be found [here](#). We will be soon adding more problems on digit dp in our future posts.

Improved By : [sauravtygg](#)

Source

<https://www.geeksforgeeks.org/digit-dp-introduction/>

Chapter 35

Distinct Prime Factors of Array Product

Distinct Prime Factors of Array Product - GeeksforGeeks

Given an array of integers. Let us say P is the product of elements of the array. Find the number of distinct prime factors of product P.

Examples:

Input : 1 2 3 4 5

Output : 3

Explanation: Here $P = 1 * 2 * 3 * 4 * 5 = 120$. Distinct prime divisors of 120 are 2, 3 and 5. So, the output is 3.

Input : 21 30 15 24 16

Output : 4

Explanation: Here $P = 21 * 30 * 15 * 24 * 16 = 3628800$. Distinct prime divisors of 3628800 are 2, 3, 5 and 7. So, the output is 4.

Naive Approach :

The simple solution for the problem would be to multiply every number in the array and then find the number of distinct prime factors of the product.

But this method can lead to integer overflow.

Better Approach :

To avoid the overflow instead of multiplying the numbers we can find the prime factors of each element separately and store the prime factors in a set or a map for unique factors.

C++

```
// C++ program to count distinct prime
// factors of a number.
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to count the number of distinct prime
// factors of product of array
int Distinct_Prime_factors(vector<int> a)
{
    // use set to store distinct factors
    unordered_set<int> m;

    // iterate over every element of array
    for (int i = 0; i < a.size(); i++) {
        int sq = sqrt(a[i]);

        // from 2 to square root of number run
        // a loop and check the numbers which
        // are factors.
        for (int j = 2; j <= sq; j++) {
            if (a[i] % j == 0) {

                // if j is a factor store it in the set
                m.insert(j);

                // divide the number with j till it
                // is divisible so that only prime factors
                // are stored
                while (a[i] % j == 0) {
                    a[i] /= j;
                }
            }
        }

        // if the number is still greater than 1 then
        // it is a prime factor, insert in set
        if (a[i] > 1) {
            m.insert(a[i]);
        }
    }

    // the number of unique prime factors will
    // be the size of the set
    return m.size();
}

// Driver Function
int main()
{
    vector<int> a = { 1, 2, 3, 4, 5 };
    cout << Distinct_Prime_factors(a) << '\n';
}
```

```
    return 0;
}
```

Java

```
// Java program to count distinct
// prime factors of a number.
import java.util.*;

class GFG
{
    // Function to count the number
    // of distinct prime factors of
    // product of array
    static int Distinct_Prime_factors(Vector a)
    {
        // use set to store distinct factors
        HashSet m = new HashSet();

        // iterate over every element of array
        for (int i = 0; i < a.size(); i++) { int sq = (int)Math.sqrt(a.get(i)); // from 2 to square
            root of number // run a loop and check the numbers // which are factors. for (int j = 2;
            j <= sq; j++) { if (a.get(i) % j == 0) { // if j is a factor store // it in the set m.add(j);
            // divide the number with j // till it is divisible so // that only prime factors // are stored
            while (a.get(i) % j == 0) { a.set(i, a.get(i) / j); } } } // if the number is still greater //
            than 1 then it is a prime factor, // insert in set if (a.get(i) > 1)
        {
            m.add(a.get(i));
        }
    }

    // the number of unique prime
    // factors will be the size of the set
    return m.size();
}

// Driver Code
public static void main(String args[])
{
    Vector a = new Vector();
    a.add(1);
    a.add(2);
    a.add(3);
    a.add(4);
    a.add(5);
    System.out.println(Distinct_Prime_factors(a));
}
}

// This code is contributed by Arnab Kundu
```

Output :

3

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/distinct-prime-factors-of-array-product/>

Chapter 36

Dynamic Disjoint Set Data Structure for large range values

Dynamic Disjoint Set Data Structure for large range values - GeeksforGeeks

Prerequisites:

- [Disjoint Set Data Structure](#)
- [Set](#)
- [Unordered_Map](#)

[Disjoint Set data structure](#) is used to keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets.

In this article, we will learn about constructing the same Data Structure dynamically. This data structure basically helps in situation where we cannot simply use arrays for creating disjoint sets because of large inputs of order 10^9 .

To illustrate this, consider the following problem. We need to find the total number of [connected components in the Graph](#) when the total Number of Vertices can be up to 10^9 .

Examples:

```
Input : Edges : { { 1, 2 },
                  { 2, 3 },
                  { 4, 5 } }

Output : 2
Explanation: {1, 2, 3} forms a component and
{4, 5} forms another component.
```

The idea to solve this problem is, we will maintain two hash tables (implemented using [unordered_maps](#) in C++). One for *parent* and other for *degree*. Parent[V] will give the

parent of the component which the Vertex V is part of and Degree will give the number of vertices in that component.

Initially, both Parent and Degree will be empty. We will keep inserting vertices to the maps as sequentially.

See the code and the explanation simultaneously for a better understanding. Below are the methods used in the code to solve the above problem:

1. **getParent(V):** This method will give the parent of the vertex V. Here we recursively find the parent of the vertex V(see code), meanwhile we assign all the vertex in that component to have the same parent.(In disjoint set data structure all the vertex in the same component have the same parent.)
2. **Union():** When we add a edge and the two vertexes are of different components we call the Union() method to join both the component. Here the parent of the component formed after joining both the components will be the parent of the component among the two which had more number of vertexes before the union. The degree of the new component is updated accordingly.
3. **getTotalComponent():** Vertex in the same component will have the same parent. We use [unordered_set \(STL\)](#) to count the total number of components. As we have maintained the Data Structure as Dynamic so, there can be any vertex which has not been added to any of the components hence they are different component alone. So the total number of components will be given by,

```
Total no of Component = Total Vertices - Number of Vertices
                        in parent (Map) + Number of Component
                        formed from the Vertexes inserted
                        in the graph.
```

Below is the implementation of above idea:

```
// Dynamic Disjoint Set Data Structure
// Union-Find

#include <bits/stdc++.h>
using namespace std;

int N;
int Edges[3][2];

// Dynamic Disjoint Set Data Structure
struct DynamicDisjointSetDS {

    // We will add the vertex to the edge
    // only when it is asked to i.e. maintain
    // a dynamic DS.
```

```
unordered_map<int, int> parent, degree;

// Total number of Vertex in the Graph
int N;

// Constructor
DynamicDisjointSetDS(int n)
{
    N = n;
}

// Get Parent of vertex V
int getParent(int vertex)
{
    // If the vertex is already inserted
    // in the graph
    if (parent.find(vertex) != parent.end()) {

        if (parent[vertex] != vertex) {
            parent[vertex] =
                getParent(parent[vertex]);
            return parent[vertex];
        }
    }
}

// if the vertex is operated for the first
// time
else {

    // insert the vertex and assign its
    // parent to itself
    parent.insert(make_pair(vertex, vertex));

    // Degree of the vertex
    degree.insert(make_pair(vertex, 1));
}

return vertex;
}

// Union by Rank
void Union(int vertexA, int vertexB)
{
    // Parent of Vertex A
    int x = getParent(vertexA);

    // Parent of Vertex B
```

```
int y = getParent(vertexB);

// if both have same parent
// Do Nothing
if (x == y)
    return;

// Merging the component
// Assigning the parent of smaller Component
// as the parent of the bigger Component.
if (degree[x] > degree[y]) {
    parent[y] = x;
    degree[x] = degree[x] + degree[y];
}
else {
    parent[x] = y;
    degree[y] = degree[y] + degree[x];
}

// Count total Component in the Graph
int GetTotalComponent()
{
    // To count the total Component formed
    // from the inserted vertex in the Graph
    unordered_set<int> total;

    // Iterate through the parent
    for (auto itr = parent.begin();
         itr != parent.end(); itr++) {

        // Add the parent of each Vertex
        // to the set
        total.insert(getParent(itr->first));
    }

    // Total Component = Total Vertices -
    // Number of Vertices in the parent +
    // Number of Components formed from
    // the Vertices inserted in the Graph
    return N - parent.size() + total.size();
};

// Solve
void Solve()
{
```

```
// Declaring the Dynamic Disjoint Set DS
DynamicDisjointSetDS dsu(N);

// Traversing through the Edges
for (int i = 0; i < 3; i++) {

    // If the Vertexes in the Edges
    // have same parent do nothing
    if (dsu.getParent(Edges[i][0]) ==
        dsu.getParent(Edges[i][1])) {
        continue;
    }

    // else Do Union of both the Components.
    else {
        dsu.Union(Edges[i][0], Edges[i][1]);
    }
}

// Get total Components
cout << dsu.GetTotalComponent();
}

// Driver Code
int main()
{
    // Total Number of Vertexes
    N = 5;

    /* Edges
     * 1 <--> 2
     * 2 <--> 3
     * 4 <--> 5      */

    Edges[0][0] = 1;
    Edges[0][1] = 2;
    Edges[1][0] = 2;
    Edges[1][1] = 3;
    Edges[2][0] = 4;
    Edges[2][1] = 3;

    // Solve
    Solve();

    return 0;
}
```

Output:

2

Note : If the number of vertexes are even larger we can implement the same code just by changing the data type from int to long long.

Source

<https://www.geeksforgeeks.org/dynamic-disjoint-set-data-structure-for-large-range-values/>

Chapter 37

Dynamic Programming | Wildcard Pattern Matching | Linear Time and Constant Space

Dynamic Programming | Wildcard Pattern Matching | Linear Time and Constant Space - GeeksforGeeks

Given a text and a wildcard pattern, find if wildcard pattern is matched with text. The matching should cover the entire text (not partial text).

The wildcard pattern can include the characters ‘?’ and ‘*’

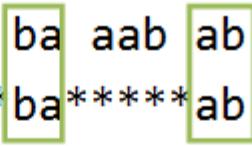
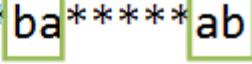
‘?’ – matches any single character

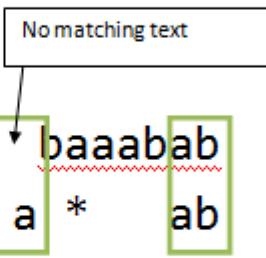
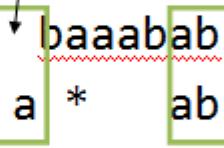
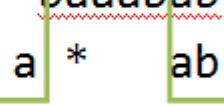
‘*’ – Matches any sequence of characters (including the empty sequence)

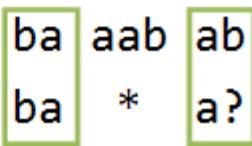
Prerequisite : [Dynamic Programming | Wildcard Pattern Matching](#)

Examples:

```
Text = "baaabab",
Pattern = "*****ba*****ab", output : true
Pattern = "baaa?ab", output : true
Pattern = "ba*a?", output : true
Pattern = "a*ab", output : false
```

Input = 
Pattern = ****
Output : true


Input = 
Pattern = 
Output : false

Input = 
Pattern = 
Output : true

Each occurrence of '?' character in wildcard pattern can be replaced with any other character and each occurrence of '*' with a sequence of characters such that the wildcard pattern becomes identical to the input string after replacement.

We have discussed a solution [here](#) which has $O(m \times n)$ time and $O(m \times n)$ space complexity.

For applying the optimization, we will at first note the **BASE CASE** which involves :

If the length of the pattern is zero then answer will be true only if the length of the text with which we have to match the pattern is also zero.

ALGORITHM | (STEP BY STEP)

Step – (1) : Let i be the marker to point at the current character of the text.

Let j be the marker to point at the current character of the pattern.

Let index_txt be the marker to point at the character of text on which we encounter '*' in pattern.

Let index_pat be the marker to point at the position of '*' in the pattern.

NOTE : WE WILL TRAVERSE THE GIVEN STRING AND PATTERN USING A WHILE LOOP

Step – (2) : At any instant if we observe that $txt[i] == pat[j]$, then we increment both i and j as no operation needs to be performed in this case.

Step – (3) : If we encounter $\text{pat}[j] == '?'$, then it resembles the case mentioned in step – (2) as ‘?’ has the property to match with any single character.

Step – (4) : If we encounter $\text{pat}[j] == '*'$, then we update the value of index_txt and index_pat as ‘*’ has the property to match any sequence of characters (including the empty sequence) and we will increment the value of j to compare next character of pattern with the current character of the text.(As character represented by i has not been answered yet).

Step – (5) : Now if $\text{txt}[i] == \text{pat}[j]$, and we have encountered a ‘*’ before, then it means that ‘*’ included the empty sequence, else if $\text{txt}[i] != \text{pat}[j]$, a character needs to be provided by ‘*’ so that current character matching takes place, then i needs to be incremented as it is answered now but the character represented by j still needs to be answered, therefore, $j = \text{index_pat} + 1$, $i = \text{index_txt} + 1$ (as ‘*’ can capture other characters as well), $\text{index_txt}++$ (as current character in text is matched).

Step – (6) : If step – (5) is not valid, that means $\text{txt}[i] != \text{pat}[j]$, also we have not encountered a ‘*’, that means it is not possible for the pattern to match the string. (return false).

Step – (7) : Check whether j reached its final value or not, then return the final answer.

Let us see the above algorithm in action, then we will move to the coding section :-

```
text = "baaabab"
pattern = "*****ba*****ab"
```

NOW APPLYING THE ALGORITHM

```
Step – (1) : i = 0 (i -> 'b')
j = 0 (j -> '*')
index_txt = -1
index_pat = -1
```

NOTE : LOOP WILL RUN TILL i REACHES ITS FINAL VALUE OR THE ANSWER BECOMES FALSE MIDWAY.

FIRST COMPARISON :-

As we see here that $\text{pat}[j] == '*'$, therefore directly jumping on to step – (4).

```
Step – (4) : index_txt = i (index_txt -> 'b')
index_pat = j (index_pat -> '*')
j++ (j -> '*')
```

```
After four more comparisons : i = 0 (i -> 'b')
j = 5 (j -> 'b')
index_txt = 0 (index_txt -> 'b')
index_pat = 4 (index_pat -> '*')
```

SIXTH COMPARISON :-

As we see here that $\text{txt}[i] == \text{pat}[j]$, but we already encountered ‘*’, therefore using step – (5).

```
Step – (5) : i = 1 (i -> 'a')
j = 6 (j -> 'a')
index_txt = 0 (index_txt -> 'b')
index_pat = 4 (index_pat -> '*')
```

SEVENTH COMPARISON :-

Step – (5) : i = 2 (i → ‘a’)
j = 7 (j → ‘*’)
index_txt = 0 (index_txt → ‘b’)
index_pat = 4 (index_pat → ‘*’)

EIGHTH COMPARISON :-

Step – (4) : i = 2 (i → ‘a’)
j = 8 (j → ‘*’)
index_txt = 2 (index_txt → ‘a’)
index_pat = 7 (index_pat → ‘*’)

After four more comparisons : i = 2 (i → ‘a’)
j = 12 (j → ‘a’)
index_txt = 2 (index_txt → ‘a’)
index_pat = 11 (index_pat → ‘*’)

THIRTEENTH COMPARISON :-

Step – (5) : i = 3 (i → ‘a’)
j = 13 (j → ‘b’)
index_txt = 2 (index_txt → ‘a’)
index_pat = 11 (index_pat → ‘*’)

FOURTEENTH COMPARISON :-

Step – (5) : i = 3 (i → ‘a’)
j = 12 (j → ‘a’)
index_txt = 3 (index_txt → ‘a’)
index_pat = 11 (index_pat → ‘*’)

FIFTEENTH COMPARISON :-

Step – (5) : i = 4 (i → ‘b’)
j = 13 (j → ‘b’)
index_txt = 3 (index_txt → ‘a’)
index_pat = 11 (index_pat → ‘*’)

SIXTEENTH COMPARISON :-

Step – (5) : i = 5 (i → ‘a’)
j = 14 (j → end)
index_txt = 3 (index_txt → ‘a’)
index_pat = 11 (index_pat → ‘*’)

SEVENTEENTH COMPARISON :-

Step – (5) : i = 4 (i → ‘b’)
j = 12 (j → ‘a’)
index_txt = 4 (index_txt → ‘b’)
index_pat = 11 (index_pat → ‘*’)

EIGHTEENTH COMPARISON :-

Step - (5) : i = 5 (i -> 'a')
j = 12 (j -> 'a')
index_txt = 5 (index_txt -> 'a')
index_pat = 11 (index_pat -> '*')

NINETEENTH COMPARISON :-

Step - (5) : i = 6 (i -> 'b')
j = 13 (j -> 'b')
index_txt = 5 (index_txt -> 'a')
index_pat = 11 (index_pat -> '*')

TWENTIETH COMPARISON :-

Step - (5) : i = 7 (i -> end)
j = 14 (j -> end)
index_txt = 5 (index_txt -> 'a')
index_pat = 11 (index_pat -> '*')

NOTE : NOW WE WILL COME OUT OF LOOP TO RUN STEP - 7.

Step - (7) : j is already present at its end position, therefore answer is true.

Below is the implementation of above optimized approach.

```
// C++ program to implement wildcard
// pattern matching algorithm
#include <bits/stdc++.h>
using namespace std;

// Function that matches input text
// with given wildcard pattern
bool strmatch(char txt[], char pat[],
              int n, int m)
{
    // empty pattern can only
    // match with empty string.
    // Base Case :
    if (m == 0)
        return (n == 0);

    // step-1 :
    // initailze markers :
    int i = 0, j = 0, index_txt = -1,
        index_pat = -1;

    while (i < n) {

        // For step - (2, 5)
        if (txt[i] == pat[j]) {
            i++;
        }
    }
}
```

```
j++;
}

// For step - (3)
else if (j < m && pat[j] == '?') {
    i++;
    j++;
}

// For step - (4)
else if (j < m && pat[j] == '*') {
    index_txt = i;
    index_pat = j;
    j++;
}

// For step - (5)
else if (index_pat != -1) {
    j = index_pat + 1;
    i = index_txt + 1;
    index_txt++;
}

// For step - (6)
else {
    return false;
}
}

// For step - (7)
while (j < m && pat[j] == '*') {
    j++;
}

// Final Check
if (j == m) {
    return true;
}

return false;
}

// Driver code
int main()
{
    char str[] = "baaabab";
    char pattern[] = "*****ba*****ab";
    // char pattern[] = "ba*****ab";
}
```

```
// char pattern[] = "ba*ab";
// char pattern[] = "a*ab";

if (strmatch(str, pattern,
            strlen(str), strlen(pattern)))
    cout << "Yes" << endl;
else
    cout << "No" << endl;

char pattern2[] = "a*****ab";
if (strmatch(str, pattern2,
            strlen(str), strlen(pattern2)))
    cout << "Yes" << endl;
else
    cout << "No" << endl;

return 0;
}
```

Output:

```
Yes
No
```

Time complexity of above solution is $O(m)$. Auxiliary space used is $O(1)$.

Source

<https://www.geeksforgeeks.org/dynamic-programming-wildcard-pattern-matching-linear-time-constant-space/>

Chapter 38

Element which occurs consecutively in a given subarray more than or equal to K times

Element which occurs consecutively in a given subarray more than or equal to K times - GeeksforGeeks

Given an array of size N and Q queries, each query consists of L, R, and K(consider 1 based-indexing for L and R). The task is to find an element for each query that occurs consecutively in the subarray [L, R] more than or equal to K times. K will always be greater than $\text{floor}((R-L+1)/2)$. If no such element exists, print -1.

Examples:

Input: arr[] = [1, 3, 3, 3, 4]

Q = 1

L = 1, R = 5, K = 3

Output: 3

The element 3 occurs 3 times consecutively in the range [1, 5]

Input: arr[] = [3, 2, 1, 1, 2, 2, 2, 2]

Q = 2

L = 2, R = 6, K = 3

L = 3, R = 8, K = 4

Output:

-1

2

Approach: Create two auxiliary arrays *left* and *right* of size n. The *Left* array will store the count of elements from the start that occurs consecutively in the array. The *Right* array

will store count of elements from back that occurs consecutively in the array. Since it is given in the question that k will always be greater than $\text{floor}((R-L+1)/2)$. Hence if any such element exists in the given range, it is always lie in the index **mid**. Mathematically, $\min\{ \text{right}[mid] + \text{mid} - 1, r - 1 \} - \max\{ \text{mid} - \text{left}[mid] + 1, l - 1 \} + 1$ will give the range of the element in the subarray L-R which lies at the index mid. If the range exceeds or is equal to k, then return the $a[\text{mid}]$ element. If it is not then return -1.

Below is the implementation of above approach :

```
// CPP program to find the element for each
// query that occurs consecutively in the
// subarray [L, R] more than or equal to K times.
#include <bits/stdc++.h>
using namespace std;

/// Function to find the element for each
/// query that occurs consecutively in the
/// subarray [L, R] more than or equal to K times.
int findElement(int arr[], int left[], int right[],
                int n, int l, int r, int k)
{
    // find mid point of subarray [L, R]
    int mid = (l - 1 + r - 1) / 2;

    // find starting and ending index of range
    int s = max(mid - left[mid] + 1, l - 1);
    int e = min(right[mid] + mid - 1, r - 1);

    int range = e - s + 1;

    // compare this range with k
    // if it is greater than or
    // equal to k, return element
    if (range >= k)
        return arr[mid];
    // if not then return -1
    else
        return -1;
}

// function to answer query in range [L, R]
int answerQuery(int arr[], int n, int l, int r, int k)
{
    int left[n], right[n];

    // store count of elements that occur
    // consecutively in the array [1, n]
    int count = 1;
    for (int i = 0; i < n - 1; i++) {
```

```
if (arr[i] == arr[i + 1]) {
    left[i] = count;
    count++;
}
else {
    left[i] = count;
    count = 1;
}
}
left[n - 1] = count;

// store count of elements that occur
// consecutively in the array [n, 1]
count = 1;
for (int i = n - 1; i > 0; i--) {
    if (arr[i] == arr[i - 1]) {
        right[i] = count;
        count++;
    }
    else {
        right[i] = count;
        count = 1;
    }
}
right[0] = count;

// Function to return the element
return findElement(arr, left, right, n, l, r, k);
}

// Driver Code
int main()
{
    int a[] = { 3, 2, 1, 1, 2, 2, 2, 2 };
    int n = sizeof(a) / sizeof(a[0]);

    // 1st query
    int L = 2, R = 6, k = 3;
    cout << answerQuery(a, n, L, R, k) << "\n";

    // 2nd query
    L = 3, R = 8, k = 4;
    cout << answerQuery(a, n, L, R, k);
}
```

Output:

-1
2

Time complexity: $O(N)$ to pre-compute the left and right array and $O(1)$ to answer every query.

Auxiliary Space: $O(N)$

Source

<https://www.geeksforgeeks.org/element-which-occurs-consecutively-in-a-given-subarray-more-than-or-equal-to-k-times/>

Chapter 39

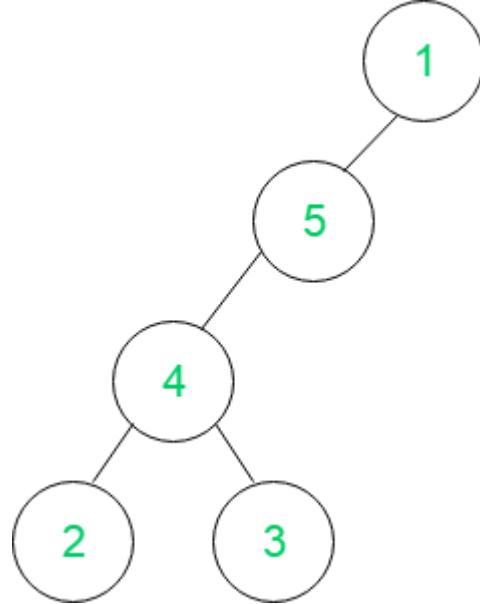
Euler tour of Binary Tree

Euler tour of Binary Tree - GeeksforGeeks

Given a binary tree where each node can have at most two child nodes, the task is to find the Euler tour of the binary tree. Euler tour is represented by a pointer to the topmost node in the tree. If the tree is empty, then value of root is NULL.

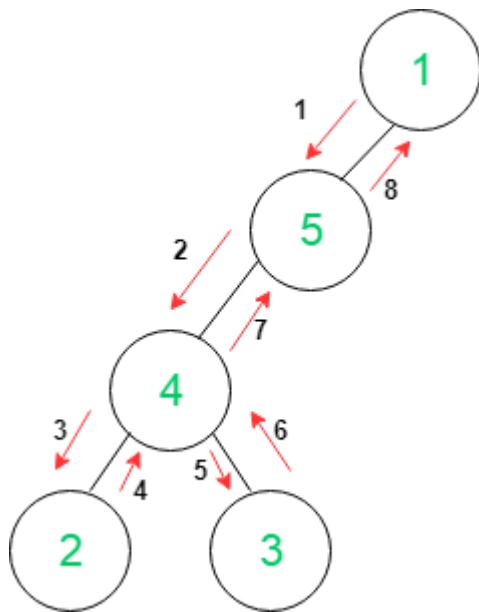
Examples:

Input :



Output: 1 5 4 2 4 3 4 5 1

Approach:



- (1) First, start with root node 1, **Euler[0]=1**
- (2) Go to left node i.e, node 5, **Euler[1]=5**
- (3) Go to left node i.e, node 4, **Euler[2]=4**
- (4) Go to left node i.e, node 2, **Euler[3]=2**
- (5) Go to left node i.e, NULL, go to parent node 4 **Euler[4]=4**
- (6) Go to right node i.e, node 3 **Euler[5]=3**
- (7) No child, go to parent, node 4 **Euler[6]=4**
- (8) All child discovered, go to parent node 5 **Euler[7]=5**
- (9) All child discovered, go to parent node 1 **Euler[8]=1**

[Euler tour of tree](#) has been already discussed where it can be applied to N-ary tree which is represented by adjacency list. If a Binary tree is represented by the classical structured way by links and nodes, then there need to first convert the tree into adjacency list representation and then we can find the Euler tour if we want to apply method discussed in the original post. But this increases the space complexity of the program. Here, In this post, a generalized space-optimized version is discussed which can be directly applied to binary trees represented by structure nodes.

This method :

- (1) Works without the use of Visited arrays.
- (2) Requires exactly $2*N-1$ vertices to store Euler tour.

```

// C++ program to find euler tour of binary tree
#include <bits/stdc++.h>
using namespace std;

/* A tree node structure */
struct Node {
    int data;
    struct Node* left;
    
```

```
    struct Node* right;
};

/* Utility function to create a new Binary Tree node */
struct Node* newNode(int data)
{
    struct Node* temp = new struct Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Find Euler Tour
void eulerTree(struct Node* root, vector<int> &Euler)
{
    // store current node's data
    Euler.push_back(root->data);

    // If left node exists
    if (root->left)
    {
        // traverse left subtree
        eulerTree(root->left, Euler);

        // store parent node's data
        Euler.push_back(root->data);
    }

    // If right node exists
    if (root->right)
    {
        // traverse right subtree
        eulerTree(root->right, Euler);

        // store parent node's data
        Euler.push_back(root->data);
    }
}

// Function to print Euler Tour of tree
void printEulerTour(Node *root)
{
    // Stores Euler Tour
    vector<int> Euler;

    eulerTree(root, Euler);

    for (int i = 0; i < Euler.size(); i++)
```

```
        cout << Euler[i] << " ";
}

/* Driver function to test above functions */
int main()
{
    // Constructing tree given in the above figure
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);

    // print Euler Tour
    printEulerTour(root);

    return 0;
}
```

Output:

```
1 2 4 2 5 2 1 3 6 8 6 3 7 3 1
```

Time Complexity: $O(2*N-1)$ where N is number of nodes in the tree.
Auxiliary Space : $O(2*N-1)$ where N is number of nodes in the tree.

Source

<https://www.geeksforgeeks.org/euler-tour-binary-tree/>

Chapter 40

Extended Mo's Algorithm with O(1) time complexity

Extended Mo's Algorithm with O(1) time complexity - GeeksforGeeks

Given an array of n elements and q range queries (range sum in this article) with no updates, task is to answer these queries with efficient time and space complexity. The time complexity of a range query after applying square root decomposition comes out to be $O(\sqrt{n})$. This square-root factor can be decreased to a constant linear factor by applying square root decomposition on the block of the array which was decomposed earlier.

Prerequisite: [Mo's Algorithm | Prefix Array](#)

Approach :

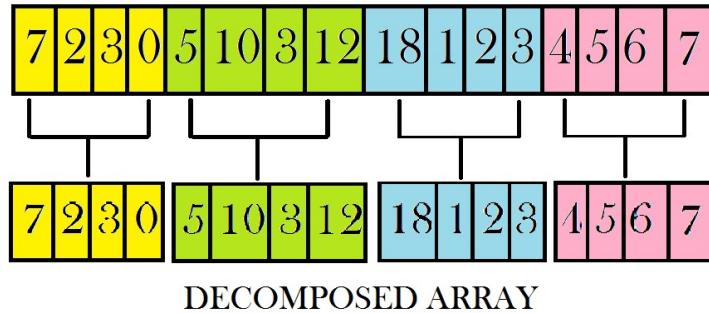
As we apply [square root decomposition](#) to the given array, querying a range-sum comes in $O(\sqrt{n})$ time.

Here, calculate the sum of blocks which are in between the blocks under consideration(cross blocks), which takes $O(\sqrt{n})$ iterations.

Initial Array :

7	2	3	0	5	10	3	12	18	1	2	3	4	5	6	7
---	---	---	---	---	----	---	----	----	---	---	---	---	---	---	---

Decomposition of array into blocks :



And the calculation time for the sum on the starting block and ending block both takes $O(\sqrt{n})$ iterations.

Which will leaves us per query time complexity of :

$$\begin{aligned}
 &= O(\sqrt{n}) + O(\sqrt{n}) + O(\sqrt{n}) \\
 &= 3 * O(\sqrt{n}) \\
 &\sim O(\sqrt{n})
 \end{aligned}$$

Here, we can reduce the runtime complexity of our query algorithm cleverly by calculating blockwise prefix sum and using it to calculate the sum accumulated in the blocks which lie between the blocks under consideration. Consider the code below :

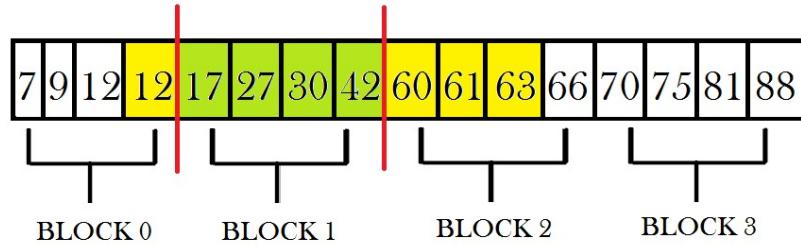
```
interblock_sum[x1][x2] = prefixData[x2 - 1] - prefixData[x1];
```

Time taken for calculation of above table is :

$$\begin{aligned}
 &= O(\sqrt{n}) * O(\sqrt{n}) \\
 &\sim O(n)
 \end{aligned}$$

NOTE : We haven't taken the sum of blocks x_1 & x_2 under consideration as they might be carrying partial data.

Prefix Array :



Suppose we want to query for the sum for range from 4 to 11, we consider the sum between block 0 and block 2 (excluding the data contained in block 0 and block 1), which can be calculated using the sum in the green coloured blocks represented in the above image.

$$\text{Sum between block 0 and block 2} = 42 - 12 = 30$$

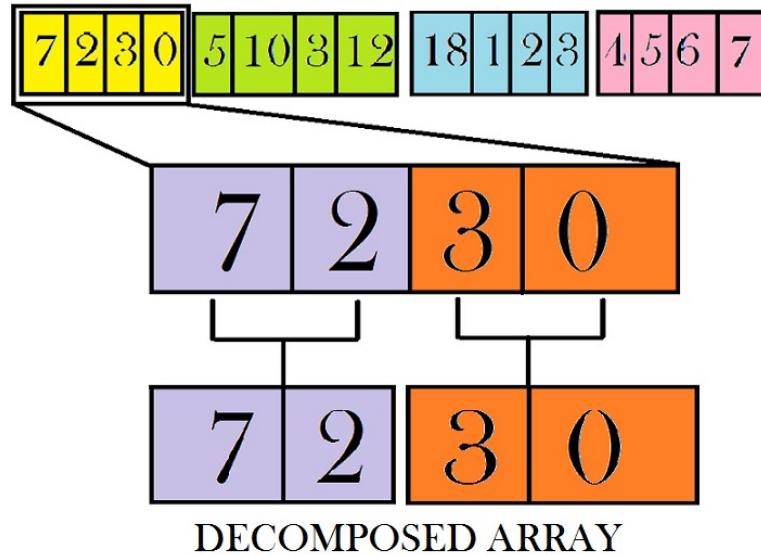
For calculation of rest of the sum present in the yellow blocks, consider the prefix array at the decomposition level-2 and repeat the process again.

Here, observe that we have reduced our time complexity per query significantly, though our runtime remains similar to our last approach :

Our new time complexity can be calculated as :

$$\begin{aligned} &= O(\sqrt{n}) + O(1) + O(\sqrt{n}) \\ &= 2 * O(\sqrt{n}) \\ &\sim O(\sqrt{n}) \end{aligned}$$

Square-root Decomposition at Level-2 :



Further, we apply square root decomposition again on every decomposed block retained from the previous decomposition. Now at this level, we have approximately $\sqrt{\sqrt{n}}$ sub-blocks in each block which were decomposed at last level. So, we need to run a range query on these blocks only two times, one time for starting block and one time for ending block.

Precalcuation Time taken for level 2 decomposition :

No of blocks at level 1 $\sim \sqrt{n}$

No of blocks at level 2 $\sim \sqrt{\sqrt{n}}$

Level-2 Decomposition Runtime of a level-1 decomposed block :

$$= O(\sqrt{n})$$

Overall runtime of level-2 decomposition over all blocks :

$$= O(\sqrt{n}) * O(\sqrt{n}) \\ \sim O(n)$$

Now, we can query our level-2 decomposed blocks in $O(\sqrt{\sqrt{n}})$ time.

So, we have reduced our overall time complexity from $O(\sqrt{n})$ to $O(\sqrt{\sqrt{n}})$

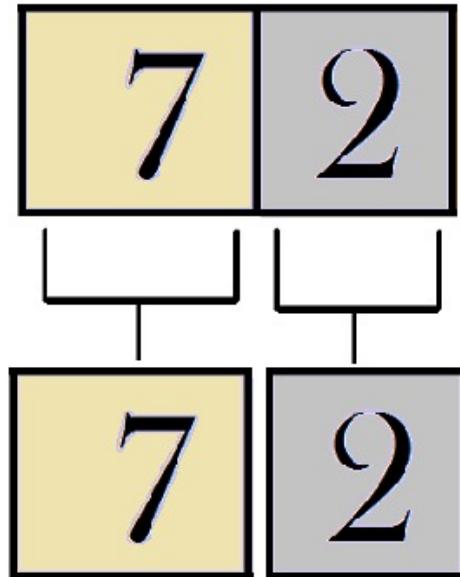
Time complexity taken in querying edge blocks :

$$= O(\sqrt{\sqrt{n}}) + O(1) + O(\sqrt{\sqrt{n}}) \\ = 2 * O(\sqrt{\sqrt{n}}) \\ \sim O(\sqrt{\sqrt{n}})$$

Total Time complexity can be calculated as :

$$\begin{aligned}
 &= O(\sqrt{\sqrt{n}}) + O(1) + O(\sqrt{\sqrt{n}}) \\
 &= 2 * O(\sqrt{\sqrt{n}}) \\
 &\sim O(\sqrt{\sqrt{n}})
 \end{aligned}$$

Square-root Decomposition at Level-3 :



DECOMPOSED ARRAY

Using this method we can decompose our array again and again recursively **d times** to reduce our time complexity to a factor of **constant linearity**.

$O(d * n^{1/(2^d)}) \sim O(k)$, as d increases this factor converges to a constant linear term

The code presented below is a representation of triple square root decomposition where d = 3:

$$O(q * d * n^{1/(2^3)}) \sim O(q * k) \sim O(q)$$

[where q represents number of range queries]

```
// CPP code for offline queries in
// approx constant time.
#include<bits/stdc++.h>
using namespace std;

int n1;

// Structure to store decomposed data
typedef struct
{
    vector<int> data;
    vector<vector<int>> rdata;
    int blocks;
    int blk_sz;
}sqrtD;

vector<vector<sqrtD>> Sq3;
vector<sqrtD> Sq2;
sqrtD Sq1;

// Square root Decomposition of
// a given array
sqrtD decompose(vector<int> arr)
{
    sqrtD sq;
    int n = arr.size();
    int blk_idx = -1;
    sq.blk_sz = sqrt(n);
    sq.data.resize((n/sq.blk_sz) + 1, 0);

    // Calculation of data in blocks
    for (int i = 0; i < n; i++)
    {
        if (i % sq.blk_sz == 0)
        {
            blk_idx++;
        }
        sq.data[blk_idx] += arr[i];
    }

    int blocks = blk_idx + 1;
    sq.blocks = blocks;

    // Calculation of prefix data
    int prefixData[blocks];
    prefixData[0] = sq.data[0];
    for(int i = 1; i < blocks; i++)
    {
```

```

        prefixData[i] =
            prefixData[i - 1] + sq.data[i];
    }

    sq.rdata.resize(blocks + 1,
                    vector<int>(blocks + 1));

    // Calculation of data between blocks
    for(int i = 0 ;i < blocks; i++)
    {
        for(int j = i + 1; j < blocks; j++)
        {
            sq.rdata[i][j] = sq.rdata[j][i] =
                prefixData[j - 1] - prefixData[i];
        }
    }

    return sq;
}

// Sqaure root Decompostion at level3
vector<vector<sqrtD>> tripleDecompose(sqrtD sq1,
                                             sqrtD sq2, vector<int> &arr)
{
    vector<vector<sqrtD>> sq(sq1.blocks,
                                vector<sqrtD>(sq1.blocks));

    int blk_idx1 = -1;

    for(int i = 0; i < sq1.blocks; i++)
    {
        int blk_ldx1 = blk_idx1 + 1;
        blk_idx1 = (i + 1) * sq1.blk_sz - 1;
        blk_idx1 = min(blk_idx1,n1 - 1);

        int blk_idx2 = blk_ldx1 - 1;

        for(int j = 0; j < sq2.blocks; ++j)
        {
            int blk_ldx2 = blk_idx2 + 1;
            blk_idx2 = blk_ldx1 + (j + 1) *
                sq2.blk_sz - 1;
            blk_idx2 = min(blk_idx2, blk_idx1);

            vector<int> ::iterator it1 =
                arr.begin() + blk_ldx2;
            vector<int> ::iterator it2 =
                arr.begin() + blk_idx2 + 1;
        }
    }
}

```

```

        vector<int> vec(it1, it2);
        sq[i][j] = decompose(vec);
    }
}
return sq;
}

// Sqaure root Decompostion at level2
vector<sqrtD> doubleDecompose(sqrtD sq1,
                                 vector<int> &arr)
{
    vector<sqrtD> sq(sq1.blocks);
    int blk_idx = -1;
    for(int i = 0; i < sq1.blocks; i++)
    {
        int blk_ldx = blk_idx + 1;
        blk_idx = (i + 1) * sq1.blk_sz - 1;
        blk_idx = min(blk_idx, n1 - 1);
        vector<int> ::iterator it1 =
            arr.begin() + blk_ldx;
        vector<int> ::iterator it2 =
            arr.begin() + blk_idx + 1;
        vector<int> vec(it1, it2);
        sq[i] = decompose(vec);
    }

    return sq;
}

// Sqaure root Decompostion at level1
void singleDecompose(vector<int> &arr)
{
    sqrtD sq1 = decompose(arr);
    vector<sqrtD> sq2(sq1.blocks);
    sq2 = doubleDecompose(sq1, arr);

    vector<vector<sqrtD>> sq3(sq1.blocks,
                                vector<sqrtD>(sq2[0].blocks));

    sq3 = tripleDecompose(sq1, sq2[0], arr);

    // ASSIGNMENT TO GLOBAL VARIABLES
    Sq1 = sq1;
    Sq2.resize(sq1.blocks);
    Sq2 = sq2;
    Sq3.resize(sq1.blocks,
              vector<sqrtD>(sq2[0].blocks));
    Sq3 = sq3;
}

```

```

}

// Function for query at level 3
int queryLevel3(int start,int end, int main_blk,
                int sub_main_blk, vector<int> &arr)
{
    int blk_sz= Sq3[0][0].blk_sz;

    // Element Indexing at level2 decompostion
    int nstart = start - main_blk *
        Sq1.blk_sz - sub_main_blk * Sq2[0].blk_sz;
    int nend = end - main_blk *
        Sq1.blk_sz - sub_main_blk * Sq2[0].blk_sz;

    // Block indexing at level3 decompostion
    int st_blk = nstart / blk_sz;
    int en_blk = nend / blk_sz;

    int answer =
        Sq3[main_blk][sub_main_blk].rdata[st_blk][en_blk];

    // If start and end point dont lie in same block
    if(st_blk != en_blk)
    {
        int left = 0, en_idx = main_blk * Sq1.blk_sz +
                    sub_main_blk * Sq2[0].blk_sz +
                    (st_blk + 1) * blk_sz -1;

        for(int i = start; i <= en_idx; i++)
        {
            left += arr[i];
        }

        int right = 0, st_idx = main_blk * Sq1.blk_sz +
                        sub_main_blk * Sq2[0].blk_sz +
                        (en_blk) * blk_sz;

        for(int i = st_idx; i <= end; i++)
        {
            right += arr[i];
        }

        answer += left;
        answer += right;
    }
    else
    {
        for(int i = start; i <= end; i++)

```

```

    {
        answer += arr[i];
    }
}

return answer;
}

// Function for splitting query to level two
int queryLevel2(int start, int end, int main_blk,
                vector<int> &arr)
{
    int blk_sz = Sq2[0].blk_sz;

    // Element Indexing at level1 decompostion
    int nstart = start - (main_blk * Sq1.blk_sz);
    int nend = end - (main_blk * Sq1.blk_sz);

    // Block indexing at level2 decompostion
    int st_blk = nstart / blk_sz;
    int en_blk = nend / blk_sz;

    // Interblock data level2 decompostion
    int answer = Sq2[main_blk].rdata[st_blk][en_blk];

    if(st_blk == en_blk)
    {
        answer += queryLevel3(start, end, main_blk,
                               st_blk, arr);
    }
    else
    {
        answer += queryLevel3(start, (main_blk *
                                       Sq1.blk_sz) + ((st_blk + 1) *
                                                       blk_sz) - 1, main_blk, st_blk, arr);

        answer += queryLevel3((main_blk * Sq1.blk_sz) +
                               (en_blk * blk_sz), end, main_blk, en_blk, arr);
    }
}

return answer;
}

// Function to return answer according to query
int Query(int start,int end,vector<int>& arr)
{
    int blk_sz = Sq1.blk_sz;
    int st_blk = start / blk_sz;
    int en_blk = end / blk_sz;
}

```

```
// Interblock data level1 decompostion
int answer = Sq1.rdata[st_blk][en_blk];

if(st_blk == en_blk)
{
    answer += queryLevel2(start, end, st_blk, arr);
}
else
{
    answer += queryLevel2(start, (st_blk + 1) *
                           blk_sz - 1, st_blk, arr);
    answer += queryLevel2(en_blk * blk_sz, end,
                           en_blk, arr);
}

// returning final answer
return answer;
}

// Driver code
int main()
{
    n1 = 16;

    vector<int> arr = {7, 2, 3, 0, 5, 10, 3, 12,
                       18, 1, 2, 3, 4, 5, 6, 7};

    singleDecompose(arr);

    int q = 5;
    pair<int, int> query[q] = {{6, 10}, {7, 12},
                                {4, 13}, {4, 11}, {12, 16}};

    for(int i = 0; i < q; i++)
    {
        int a = query[i].first, b = query[i].second;
        printf("%d\n", Query(a - 1, b - 1, arr));
    }

    return 0;
}
```

Output:

44
39
58

51

25

Time Complexity : $O(q * d * n^{1/(2^3)})$ $O(q * k)$ $O(q)$

Auxiliary Space : $O(k * n)$ $O(n)$

Note : This article is to only explain the method of decomposing the square root to further decomposition.

Improved By : [vaibhav2992](#)

Source

<https://www.geeksforgeeks.org/extended-mos-algorithm-o1-time-complexity/>

Chapter 41

Fast I/O for Competitive Programming

Fast I/O for Competitive Programming - GeeksforGeeks

In competitive programming, it is important to read input as fast as possible so we save valuable time.

You must have seen various problem statements saying: “**Warning:** Large I/O data, be careful with certain languages (though most should be OK if the algorithm is well designed)”. Key for such problems is to use Faster I/O techniques.

It is often recommended to use scanf/printf instead of cin/cout for a fast input and output. However, you can still use cin/cout and achieve the same speed as scanf/printf by including the following two lines in your main() function:

```
ios_base::sync_with_stdio(false);
```

It toggles on or off the synchronization of all the C++ standard streams with their corresponding standard C streams if it is called before the program performs its first input or output operation. Adding ios_base::sync_with_stdio (false); (which is true by default) before any I/O operation avoids this synchronization. It is a static member of function of std::ios_base.

```
cin.tie(NULL);
```

tie() is a method which simply guarantees the flushing of std::cout before std::cin accepts an input. This is useful for interactive console programs which require the console to be updated constantly but also slows down the program for large I/O. The NULL part just returns a NULL pointer.

Moreover, you can include the standard template library (STL) with a single include:

```
#include <bits/stdc++.h>
```

So your template for competitive programming could look like this:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    return 0;
}
```

It is recommended to use `cout << "\n";` instead of `cout << endl;`. `endl` is slower because it forces a flushing stream, which is usually unnecessary (See [this](#) for details). (You'd need to flush if you were writing, say, an interactive progress bar, but not when writing a million lines of data.) Write '`\n`' instead of `endl`.

We can test our input and output methods on the problem [INTEST – Enormous Input Teston SPOJ](#). Before further reading, I would suggest you to solve the problem first.

Solution in C++ 4.9.2

Normal I/O : The code below uses `cin` and `cout`. The solution gets accepted with a runtime of 2.17 seconds.

```
// A normal IO example code
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, k, t;
    int cnt = 0;
    cin >> n >> k;
    for (int i=0; i<n; i++)
    {
        cin >> t;
        if (t % k == 0)
            cnt++;
    }
    cout << cnt << "\n";
    return 0;
}
```

Fast I/O However, we can do better and reduce the runtime a lot by adding two lines. The program below gets accepted with a runtime of 0.41 seconds.

```
// A fast I/O program
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // added the two lines below
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, k, t;
    int cnt = 0;
    cin >> n >> k;
    for (int i=0; i<n; i++)
    {
        cin >> t;
        if (t % k == 0)
            cnt++;
    }
    cout << cnt << "\n";
    return 0;
}
```

Now, talking about competitive contests like ACM ICPC, Google CodeJam, TopCoder Open, here is an exclusive code to read integers in the fastest way.

```
void fastscan(int &number)
{
    //variable to indicate sign of input number
    bool negative = false;
    register int c;

    number = 0;

    // extract current character from buffer
    c = getchar();
    if (c=='-')
    {
        // number is negative
        negative = true;
    }

    // extract the next character from the buffer
    c = getchar();
}

// Keep on extracting characters if they are integers
// i.e ASCII Value lies from '0'(48) to '9' (57)
for (; (c>47 && c<58); c=getchar())
```

```
number = number *10 + c - 48;

// if scanned input has a negative sign, negate the
// value of the input number
if (negative)
    number *= -1;
}

// Function Call
int main()
{
    int number;
    fastscan(number);
    cout << number << "\n";
    return 0;
}
```

[getchar_unlocked\(\)](#) for faster input in C for competitive programming

Source

<https://www.geeksforgeeks.org/fast-io-for-competitive-programming/>

Chapter 42

Fast I/O in Java in Competitive Programming

Fast I/O in Java in Competitive Programming - GeeksforGeeks

Using Java in competitive programming is not something many people would suggest just because of its slow input and output, and well indeed it is slow.

In this article, we have discussed some ways to get around the difficulty and change the verdict from TLE to (in most cases) AC.

For all the Programs below

Input:

```
7 3
1
51
966369
7
9
999996
11
```

Output:

```
4
```

1. **Scanner Class** – (easy, less typing, but not recommended very slow, refer [this](#) for reasons of slowness): In most of the cases we get TLE while using scanner class. It uses built-in nextInt(), nextLong(), nextDouble methods to read the desired object after initiating scanner object with input stream.(eg System.in). The following program many a times gets time limit exceeded verdict and therefore not of much use.

```
// Working program using Scanner
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Scanner;
public class Main
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();
        int k = s.nextInt();
        int count = 0;
        while (n-- > 0)
        {
            int x = s.nextInt();
            if (x%k == 0)
                count++;
        }
        System.out.println(count);
    }
}
```

2. **BufferedReader** – (fast, but not recommended as it requires lot of typing): The Java.io.BufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. With this method we will have to parse the value every time for desired type. Reading multiple words from single line adds to its complexity because of the use of StringTokenizer and hence this is not recommended. This gets accepted with a running time of approx 0.89 s.but still as you can see it requires a lot of typing all together and therefore method 3 is recommended.

```
// Working program using BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Main
{
    public static void main(String[] args) throws IOException
    {

        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in));
    }
}
```

```

 StringTokenizer st = new StringTokenizer(br.readLine());
 int n = Integer.parseInt(st.nextToken());
 int k = Integer.parseInt(st.nextToken());
 int count = 0;
 while (n-- > 0)
 {
     int x = Integer.parseInt(br.readLine());
     if (x%k == 0)
         count++;
 }
 System.out.println(count);
}
}

```

3. **Userdefined FastReader Class-** (which uses bufferedReader and StringTokenizer): This method uses the time advantage of BufferedReader and StringTokenizer and the advantage of user defined methods for less typing and therefore a faster input altogether. This gets accepted with a time of 1.23 s and this method is *very much recommended* as it is easy to remember and is fast enough to meet the needs of most of the question in competitive coding.

```

// Working program with FastReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.StringTokenizer;

public class Main
{
    static class FastReader
    {
        BufferedReader br;
        StringTokenizer st;

        public FastReader()
        {
            br = new BufferedReader(new
                InputStreamReader(System.in));
        }

        String next()
        {
            while (st == null || !st.hasMoreElements())
            {
                try
                {

```

```
        st = new StringTokenizer(br.readLine());
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
return st.nextToken();
}

int nextInt()
{
    return Integer.parseInt(next());
}

long nextLong()
{
    return Long.parseLong(next());
}

double nextDouble()
{
    return Double.parseDouble(next());
}

String nextLine()
{
    String str = "";
    try
    {
        str = br.readLine();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    return str;
}
}

public static void main(String[] args)
{
    FastReader s=new FastReader();
    int n = s.nextInt();
    int k = s.nextInt();
    int count = 0;
    while (n-- > 0)
    {
```

```
        int x = s.nextInt();
        if (x%k == 0)
            count++;
    }
    System.out.println(count);
}
}
```

4. **Using Reader Class:** There is yet another fast way through the problem, I would say the fastest way but is not recommended since it requires very cumbersome methods in its implementation. It uses `inputDataStream` to read through the stream of data and uses `read()` method and `nextInt()` methods for taking inputs. This is by far the fastest ways of taking input but is difficult to remember and is cumbersome in its approach. Below is the sample program using this method.

This gets accepted with a surprising time of just 0.28 s. Although this is ultra fast, it is clearly not an easy method to remember.

```
// Working program using Reader Class
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.StringTokenizer;

public class Main
{
    static class Reader
    {
        final private int BUFFER_SIZE = 1 << 16;
        private DataInputStream din;
        private byte[] buffer;
        private int bufferPointer, bytesRead;

        public Reader()
        {
            din = new DataInputStream(System.in);
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }

        public Reader(String file_name) throws IOException
        {
            din = new DataInputStream(new FileInputStream(file_name));
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }
    }
}
```

```
public String readLine() throws IOException
{
    byte[] buf = new byte[64]; // line length
    int cnt = 0, c;
    while ((c = read()) != -1)
    {
        if (c == '\n')
            break;
        buf[cnt++] = (byte) c;
    }
    return new String(buf, 0, cnt);
}

public int nextInt() throws IOException
{
    int ret = 0;
    byte c = read();
    while (c <= ' ')
        c = read();
    boolean neg = (c == '-');
    if (neg)
        c = read();
    do
    {
        ret = ret * 10 + c - '0';
    } while ((c = read()) >= '0' && c <= '9');

    if (neg)
        return -ret;
    return ret;
}

public long nextLong() throws IOException
{
    long ret = 0;
    byte c = read();
    while (c <= ' ')
        c = read();
    boolean neg = (c == '-');
    if (neg)
        c = read();
    do {
        ret = ret * 10 + c - '0';
    }
    while ((c = read()) >= '0' && c <= '9');
    if (neg)
        return -ret;
    return ret;
}
```

```
}

public double nextDouble() throws IOException
{
    double ret = 0, div = 1;
    byte c = read();
    while (c <= ' ')
        c = read();
    boolean neg = (c == '-');
    if (neg)
        c = read();

    do {
        ret = ret * 10 + c - '0';
    }
    while ((c = read()) >= '0' && c <= '9');

    if (c == '.')
    {
        while ((c = read()) >= '0' && c <= '9')
        {
            ret += (c - '0') / (div *= 10);
        }
    }

    if (neg)
        return -ret;
    return ret;
}

private void fillBuffer() throws IOException
{
    bytesRead = din.read(buffer, bufferPointer = 0, BUFFER_SIZE);
    if (bytesRead == -1)
        buffer[0] = -1;
}

private byte read() throws IOException
{
    if (bufferPointer == bytesRead)
        fillBuffer();
    return buffer[bufferPointer++];
}

public void close() throws IOException
{
    if (din == null)
        return;
```

```
        din.close();
    }
}

public static void main(String[] args) throws IOException
{
    Reader s=new Reader();
    int n = s.nextInt();
    int k = s.nextInt();
    int count=0;
    while (n-- > 0)
    {
        int x = s.nextInt();
        if (x%k == 0)
            count++;
    }
    System.out.println(count);
}
}
```

Suggested Read: [Enormous Input Test](#)designed to check the fast input handling of your language.Timings of all programs are noted from SPOJ.

Source

<https://www.geeksforgeeks.org/fast-io-in-java-in-competitive-programming/>

Chapter 43

Find $(a^b) \% m$ where 'b' is very large

Find $(a^b) \% m$ where 'b' is very large - GeeksforGeeks

Given three numbers a , b and m where $1 \leq a \leq m \leq 10^6$. Given very large ' b ' containing up to 10^6 digits and m is a prime number, the task is to find $(a^b) \% m$.

Examples:

Input: a = 2, b = 3, m = 17

Output: 8

$$2^{\wedge} 3 \% 17 = 8$$

Input: a = 3, b = 10000000000000000000000000000000, m = 1000000007

Output: 835987331

Approach: According to Fermat's little theorem,

$a^{(p-1)} \bmod p = 1$, When p is prime.

From this, as of the problem, M is prime, express $A^{\wedge}B \bmod M$ as follows:

$$A^{\wedge}B \bmod M = (A^{\wedge}(M-1) * A^{\wedge}(M-1) * \dots * A^{\wedge}(M-1) * A^{\wedge}(x)) \bmod M$$

Where x is $B \bmod M-1$ and $A^{\wedge (M-1)}$ continues $B/(M-1)$ times

Now, from Fermat's Little Theorem,

$$A^{\wedge (M-1)} \bmod M = 1.$$

Hence,

$$A^B \bmod M = (1 * 1 * \dots * 1 * A^{\wedge}(x)) \bmod M$$

Hence mod B with M-1 to reduce the number to a smaller one and then use `power()` method to compute $(a^b) \% m$.

Below is the implementation of the above approach:

C++

```
for (int i = 0; i < b.length(); i++)
    remainderB = (remainderB * 10 + b[i] - '0') % (MOD - 1);

cout << power(a, remainderB, MOD) << endl;
return 0;
}
```

Java

```
long remainderB = 0;
long MOD = 1000000007;

// Reduce the number B to a small
// number using Fermat Little
for (int i = 0; i < b.length(); i++)
    remainderB = (remainderB * 10 +
                  b.charAt(i) - '0') %
                (MOD - 1);

System.out.println(power(a, remainderB, MOD));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Reduce the number B
# to a small number
# using Fermat Little
for i in range(len(b)):
    remainderB = ((remainderB * 10 +
    ord(b[i]) - 48) %
    (MOD - 1));

print(power(a, remainderB, MOD));
# This code is contributed by mits

C#

// C# program to find
//  $(a^b) \% m$  for b very large.
using System;

class GFG
{

    // Function to find power
    static long power(long x,
                      long y, long p)
    {
        // Initialize result
        long res = 1;

        // Update x if it is more
        // than or equal to p
        x = x % p;

        while (y > 0)
        {
            // If y is odd, multiply
            // x with the result
            if ((y & 1) > 0)
                res = (res * x) % p;

            // y must be even now
            y = y >> 1; // y = y/2
            x = (x * x) % p;
        }
        return res;
    }

    // Driver Code
    public static void Main ()
    {
```

PHP

```
<?php
// PHP program to find
// (a^b)%m for b very large.

// Function to find power
function power($x, $y, $p)
{
    $res = 1; // Initialize result

    // Update x if it is
    // more than or equal to p
    $x = $x % $p;

    while ($y > 0)
    {
        // If y is odd, multiply
        // x with the result
        if ($y & 1)
            $res = ($res * $x) % $p;

        // y must be even now
        $y = $y >> 1; // y = y/2
        $x = ($x * $x) % $p;
    }
}
```

Output:

835987331

Improved By : vt_m, Mithun Kumar

Source

<https://www.geeksforgeeks.org/find-abm-where-b-is-very-large/>

Chapter 44

Find Nth term (A matrix exponentiation example)

Find Nth term (A matrix exponentiation example) - GeeksforGeeks

We are given a recursive function that describes Nth terms in form of other terms. In this article we have taken specific example.

Given : $T_n = T_{n-1} + T_{n-2}$
Given : $T_0 = 0$
 $T_1 = 1$

Now you are given n, and you have to find out nth term using above formula.

Examples:

Input : n = 2
Output : 5

Input : n = 3
Output : 13

Prerequisite :

Basic Approach: This problem can be solved by simply just iterating over the n terms. Every time you find a term, using this term find next one and so on. But time complexity of this problem is of order O(n).

Optimized Approach

All such problem where a term is a function of other terms in linear fashion. Then these can be solved using Matrix (Please refer : [Matrix Exponentiation](#)). First we make transformation matrix and then just use matrix exponentiation to find Nth term.

Step by Step method includes:

Step 1. Determine k the number of terms on which T(i) depends.

In our example $T(i)$ depends on two terms so, $k = 2$

Step 2. Determine initial values

As in this article $T_0=1$, $T_1=1$ are given.

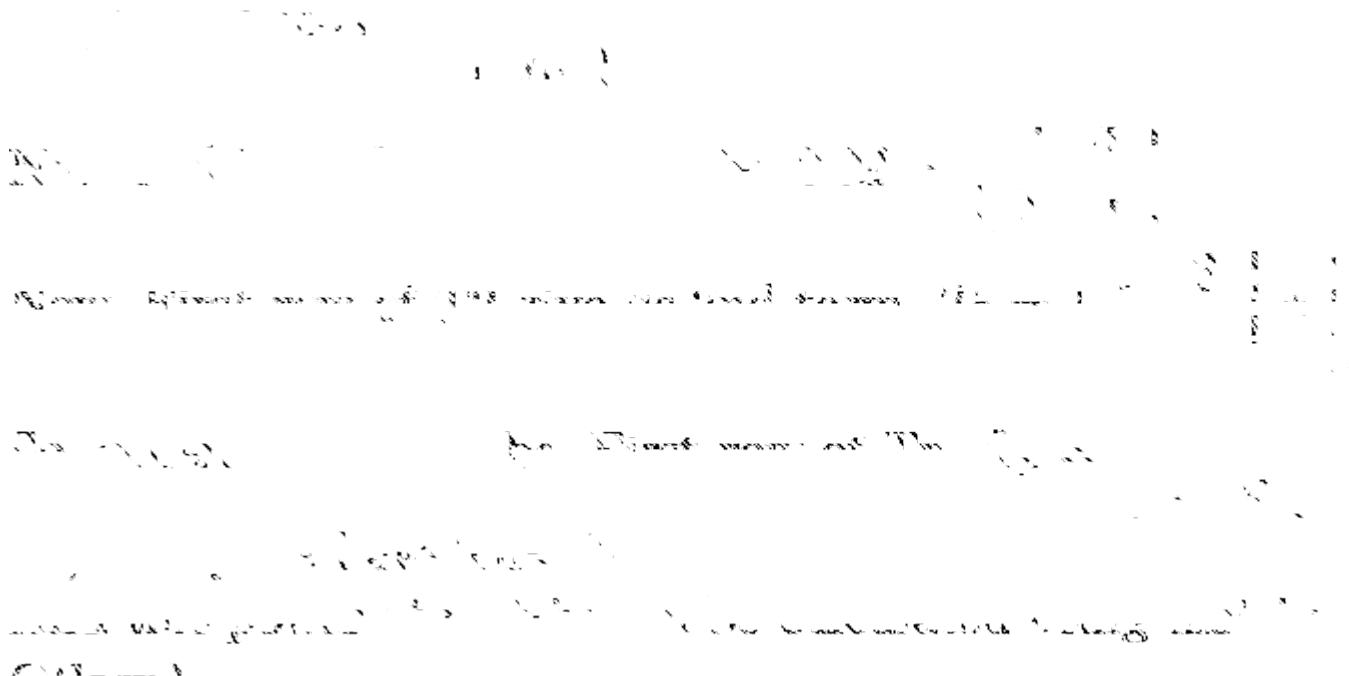
Step 3. Determine TM, the transformation matrix.

This is the most important step in solving recurrence relation. In this step, we have to make matrix of dimension $k \times k$.

Such that

$T(i) = TM^*(\text{initial value vector})$

Here **initial value vector** is vector that contains intial value.we name this vector as **initial**.



Below is C++ program to implement above approach

```
// CPP program to find n-th term of a recursive
// function using matrix exponentiation.
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000009

#define ll long long int

ll power(ll n)
```

```
{
    if (n <= 1)
        return 1;

    // This power function returns first row of
    // {Transformation Matrix}^n-1*Initial Vector
    n--;

    // This is an identity matrix.
    ll res[2][2] = { 1, 0, 0, 1 };

    // this is Transformation matrix.
    ll tMat[2][2] = { 2, 3, 1, 0 };

    // Matrix exponentiation to calculate power of {tMat}^n-1
    // store res in "res" matrix.
    while (n) {

        if (n & 1) {
            ll tmp[2][2];
            tmp[0][0] = (res[0][0] * tMat[0][0] +
                          res[0][1] * tMat[1][0]) % MOD;
            tmp[0][1] = (res[0][0] * tMat[0][1] +
                          res[0][1] * tMat[1][1]) % MOD;
            tmp[1][0] = (res[1][0] * tMat[0][0] +
                          res[1][1] * tMat[1][0]) % MOD;
            tmp[1][1] = (res[1][0] * tMat[0][1] +
                          res[1][1] * tMat[1][1]) % MOD;
            res[0][0] = tmp[0][0];
            res[0][1] = tmp[0][1];
            res[1][0] = tmp[1][0];
            res[1][1] = tmp[1][1];
        }
        n = n / 2;
        ll tmp[2][2];
        tmp[0][0] = (tMat[0][0] * tMat[0][0] +
                      tMat[0][1] * tMat[1][0]) % MOD;
        tmp[0][1] = (tMat[0][0] * tMat[0][1] +
                      tMat[0][1] * tMat[1][1]) % MOD;
        tmp[1][0] = (tMat[1][0] * tMat[0][0] +
                      tMat[1][1] * tMat[1][0]) % MOD;
        tmp[1][1] = (tMat[1][0] * tMat[0][1] +
                      tMat[1][1] * tMat[1][1]) % MOD;
        tMat[0][0] = tmp[0][0];
        tMat[0][1] = tmp[0][1];
        tMat[1][0] = tmp[1][0];
        tMat[1][1] = tmp[1][1];
    }
}
```

```
// res store {Transformation matrix}^n-1
// hence will be first row of res*Initial Vector.
return (res[0][0] * 1 + res[0][1] * 1) % MOD;
}

// Driver code
int main()
{
    ll n = 3;
    cout << power(n);
    return 0;
}
```

Output:

13

Time Complexity : O(Log n)

The same idea is used to [find n-th Fibonacci number in O\(Log n\)](#)

Source

<https://www.geeksforgeeks.org/find-nth-term-a-matrix-exponentiation-example/>

Chapter 45

Find if it is possible to reach the end through given transitions

Find if it is possible to reach the end through given transitions - GeeksforGeeks

Given, n points on X-axis and the list of allowed transition between the points. Find if it is possible to reach the end from starting point through these transitions only.

Note: If there is a transition between points x1 and x2, then you can move from point x to any intermediate points between x1 and x2 or directly to x2.

Examples:

```
Input : n = 5 ,  
Transitions allowed: 0 -> 2  
                    2 -> 4  
                    3 -> 5  
Output : YES  
Explanation : We can move from 0 to 5 using the  
allowed transitions. 0->2->3->5
```

```
Input : n = 7 ,  
Transitions allowed: 0 -> 4  
                    2 -> 5  
                    6 -> 7  
Output : NO  
Explanation : We can't move from 0 to 7 as there is  
no transition between 5 and 6.
```

The idea to solve this problem is to first sort this list according to first element of the pairs. Then start traversing from the second pair of the list and check if the first element of this pair is in between second element of previous pair and second element of current pair or not. This condition is used to check if there is a path between two consecutive pairs.

At the end check if the point we have reached is the destination point and the point from which we have started is start point. If so, print YES otherwise print NO.

```
// C++ implementation of above idea
#include<bits/stdc++.h>
using namespace std;

// function to check if it is possible to
// reach the end through given points
bool checkPathPairs(int n, vector<pair<int, int>> vec)
{
    // sort the list of pairs
    // according to first element
    sort(vec.begin(), vec.end());

    int start = vec[0].first;

    int end=vec[0].second;

    // start traversing from 2nd pair
    for (int i=1; i<n; i++)
    {
        // check if first element of current pair
        // is in between second element of previous
        // and current pair
        if (vec[i].first > end)
            break;

        end=max(end, vec[i].second);
    }

    return (n <= end && start==0);
}

// Driver code
int main()
{
    vector<pair<int, int>> vec;
    vec.push_back(make_pair(0,4));
    vec.push_back(make_pair(2,5));
    vec.push_back(make_pair(6,7));

    if (checkPathPairs(7, vec))
        cout << "YES";
    else
        cout << "NO";

    return 0;
}
```

}

Output:

NO

Source

<https://www.geeksforgeeks.org/find-possible-reach-end-given-transitions/>

Chapter 46

Find if neat arrangement of cups and shelves can be made

Find if neat arrangement of cups and shelves can be made - GeeksforGeeks

Given three different types of cups ($a[]$) and saucers ($b[]$), and n number of shelves, find if neat arrangement of cups and shelves can be made.

Arrangement of the cups and saucers will be neat if it follows the below rules:

- No shelf can contain both cups and saucers
- There can be no more than 5 cups in any shelf
- There can be no more than 10 saucers in any shelf

Examples:

```
Input : a[] = {3, 2, 6}
        b[] = {4, 8, 9}
        n = 10
Output : Yes
Explanation :
Total cups = 11, shelves required = 3
Total saucers = 21, shelves required = 3
Total required shelves = 3 + 3 = 6,
which is less than given number of
shelves n. So, output is Yes.
```

```
Input : a[] = {4, 7, 4}
        b[] = {3, 9, 10}
        n = 2
Output : No
```

Approach : To arrange the cups and the saucers, find out the total number of cups a and total number of saucers b . Since, there cannot be more than 5 cups in the same shelf, therefore find out the maximum number of shelves required for cup by the formula

$$\lceil \frac{a}{5} \rceil$$

and the maximum number of shelves required for saucers by using the formula

$$\lceil \frac{b}{10} \rceil$$

If sum of these two values is equal to or less than n then the arrangement is possible otherwise not.

Below is the implementation of above approach :

C++

```
// C++ code to find if neat
// arrangement of cups and
// shelves can be made
#include<bits/stdc++.h>
using namespace std;

// Function to check arrangement
void canArrange(int a[], int b[], int n)
{
    int suma = 0, sumb = 0;

    // Calculating total number
    // of cups
    for(int i = 0; i < 2; i++)
        suma += a[i];

    // Calculating total number
    // of saucers
    for(int i = 0; i < 2; i++)
        sumb += b[i];

    // Adding 5 and 10 so that if the
    // total sum is less than 5 and
    // 10 then we can get 1 as the
    // answer and not 0
    int na = (suma + 5 - 1) / 5;
    int nb = (sumb + 10 - 1) / 10;

    if(na + nb <= n)
        cout << "Yes";
    else
        cout << "No";
}
```

```
// Driver code
int main()
{
    // Number of cups of each type
    int a[] = {3, 2, 6};

    // Number of saucers of each type
    int b[] = {4, 8, 9};

    // Number of shelves
    int n = 10;

    // Calling function
    canArrange(a, b, n);
    return 0;
}
```

Java

```
// Java code to find if neat
// arrangement of cups and
// shelves can be made
import java.io.*;

class Gfg
{
    // Function to check arrangement
    public static void canArrange(int a[], int b[],
                                   int n)
    {
        int suma = 0, sumb = 0;

        // Calculating total number
        // of cups
        for(int i = 0; i < 2; i++)
            suma += a[i];

        // Calculating total number
        // of saucers
        for(int i = 0; i < 2; i++)
            sumb += b[i];

        // Adding 5 and 10 so that if
        // the total sum is less than
        // 5 and 10 then we can get 1
        // as the answer and not 0
        int na = (suma + 5 - 1) / 5;
```

```
int nb = (sumb + 10 - 1) / 10;

if(na + nb <= n)
    System.out.println("Yes");
else
    System.out.println("No");
}

// Driver function
public static void main(String args[])
{
    // Number of cups of each type
    int a[] = {3, 2, 6};

    // Number of saucers of each type
    int b[] = {4, 8, 9};

    // Number of shelves
    int n = 10;

    // Calling function
    canArrange(a, b, n);
}
}
```

Python 3

```
# Python code to find if neat
# arrangement of cups and
# shelves can be made

import math

# Function to check arrangement
def canArrange( a, b, n):
    suma = 0
    sumb = 0

    # Calculating total number
    # of cups
    for i in range(0, len(a)):
        suma += a[i]

    # Calculating total number
    # of saucers
    for i in range(0, len(b)):
        sumb += b[i]
```

```
# Adding 5 and 10 so that if
# the total sum is less than
# 5 and 10 then we can get 1
# as the answer and not 0
na = (suma + 5 - 1) / 5
nb = (sumb + 10 - 1) / 10

if(na + nb <= n):
    print("Yes")
else:
    print("No")

# driver function

#Number of cups of each type
a = [3, 2, 6]

# Number of saucers of each type
b = [4, 8, 9]

# Number of shelves
n = 10

#Calling function
canArrange(a ,b ,n)

# This code is contributed by Gitanjali.
```

C#

```
// C# code to find if neat
// arrangement of cups and
// shelves can be made
using System;

class Gfg {

    // Function to check arrangement
    public static void canArrange(int []a, int []b,
                                  int n)
    {

        int suma = 0, sumb = 0;

        // Calculating total number
        // of cups
        for(int i = 0; i < 2; i++)

```

```
suma += a[i];

// Calculating total number
// of saucers
for(int i = 0; i < 2; i++)
    sumb += b[i];

// Adding 5 and 10 so that if
// the total sum is less than
// 5 and 10 then we can get 1
// as the answer and not 0
int na = (suma + 5 - 1) / 5;
int nb = (sumb + 10 - 1) / 10;

if(na + nb <= n)
    Console.WriteLine("Yes");
else
    Console.WriteLine("No");
}

// Driver function
public static void Main()
{
    // Number of cups of each type
    int []a = {3, 2, 6};

    // Number of saucers of each type
    int []b = {4, 8, 9};

    // Number of shelves
    int n = 10;

    // Calling function
    canArrange(a, b, n);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find if neat
// arrangement of cups and
// shelves can be made
```

```
// Function to check arrangement
function canArrange($a, $b, $n)
{
    $summa = 0; $sumb = 0;

    // Calculating total number
    // of cups
    for( $i = 0; $i < 2; $i++)
        $summa += $a[$i];

    // Calculating total number
    // of saucers
    for( $i = 0; $i < 2; $i++)
        $sumb += $b[$i];

    // Adding 5 and 10 so that if the
    // total sum is less than 5 and
    // 10 then we can get 1 as the
    // answer and not 0
    $na = ($summa + 5 - 1) / 5;
    $nb = ($sumb + 10 - 1) / 10;

    if($na + $nb <= $n)
        echo "Yes";
    else
        echo "No";
}

// Driver code

// Number of cups of each type
$a = array(3, 2, 6);

// Number of saucers of each type
$b = array(4, 8, 9);

// Number of shelves
$n = 10;

// Calling function
canArrange($a, $b, $n);

// This code is contributed by vt_m.
?>
```

Output:

Yes

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-neat-arrangement-cups-shelves-can-made/>

Chapter 47

Find the Largest Cube formed by Deleting minimum Digits from a number

Find the Largest Cube formed by Deleting minimum Digits from a number - GeeksforGeeks

Given a number n, the task is to find the largest perfect cube that can be formed by deleting minimum digits(possibly 0) from the number.

X is called a perfect cube if $X = Y^3$ for some Y.

Examples:

```
Input : 4125
Output : 125
Explanation
125 = 53. We can form 125 by deleting digit 4 from 4125
```

```
Input : 876
Output :8
Explanation
8 = 23. We can form 8 by deleting digits 7 and 6 from 876
```

We can generate cubes of all numbers till from 1 to $N^{1/3}$ (We don't consider 0 as 0 is not considered as a perfect cube). We iterate the cubes from largest to the smallest.

Now if we look at the number n given to us, then we know that this number contains only $\log(n) + 1$ digits, thus we can efficiently approach the problem if we treat this number n as a string hereafter.

While iterating on the perfect cubes, we check if the perfect cube is a subsequence of the number n when its represented as a string. If this is the case then the deletions required for changing the number n to the current perfect cube is:

```
No of deleted digits = No of digits in number n -  
                        Number of digits in current  
                        perfect cube
```

Since we want the largest cube number we traverse the array of preprocessed cubes in reverse order.

```
/* C++ code to implement maximum perfect cube  
   formed after deleting minimum digits */  
#include <bits/stdc++.h>  
using namespace std;  
  
// Returns vector of Pre Processed perfect cubes  
vector<string> preProcess(long long int n)  
{  
    vector<string> preProcessedCubes;  
    for (int i = 1; i * i * i <= n; i++) {  
        long long int iThCube = i * i * i;  
  
        // convert the cube to string and push into  
        // preProcessedCubes vector  
        string cubeString = to_string(iThCube);  
        preProcessedCubes.push_back(cubeString);  
    }  
    return preProcessedCubes;  
}  
  
/* Utility function for findLargestCube().  
   Returns the Largest cube number that can be formed */  
string findLargestCubeUtil(string num,  
                           vector<string> preProcessedCubes)  
{  
    // reverse the preProcessed cubes so that we  
    // have the largest cube in the beginning  
    // of the vector  
    reverse(preProcessedCubes.begin(), preProcessedCubes.end());  
  
    int totalCubes = preProcessedCubes.size();  
  
    // iterate over all cubes  
    for (int i = 0; i < totalCubes; i++) {  
        string currCube = preProcessedCubes[i];  
  
        int digitsInCube = currCube.length();  
        int index = 0;  
        int digitsInNumber = num.length();  
        for (int j = 0; j < digitsInNumber; j++) {
```

```
// check if the current digit of the cube
// matches with that of the number num
if (num[j] == currCube[index])
    index++;

if (digitsInCube == index)
    return currCube;
}
}

// if control reaches here, the its
// not possible to form a perfect cube
return "Not Possible";
}

// wrapper for findLargestCubeUtil()
void findLargestCube(long long int n)
{
    // pre process perfect cubes
    vector<string> preProcessedCubes = preprocess(n);

    // convert number n to string
    string num = to_string(n);

    string ans = findLargestCubeUtil(num, preProcessedCubes);

    cout << "Largest Cube that can be formed from "
        << n << " is " << ans << endl;
}

// Driver Code
int main()
{
    long long int n;
    n = 4125;
    findLargestCube(n);

    n = 876;
    findLargestCube(n);

    return 0;
}
```

Output:

Largest Cube that can be formed from 4125 is 125

Chapter 47. Find the Largest Cube formed by Deleting minimum Digits from a number

Largest Cube that can be formed from 876 is 8

Time Complexity of the above algorithm is $O(N^{1/3}\log(N))$ $\log(N)$ is due to the fact that the number of digits in N are $\log(N) + 1$.

Source

<https://www.geeksforgeeks.org/find-largest-cube-formed-deleting-minimum-digits-number/>

Chapter 48

Find the arrangement of queue at given time

Find the arrangement of queue at given time - GeeksforGeeks

n people are standing in a queue to buy entry ticket for the carnival. People present there strongly believe in chivalry. Therefore, at time = t, if a man at position x, finds a woman standing behind him then he exchanges his position with her and therefore, at time = t+1, woman is standing at position x while man is standing behind her.

Given the total number of people standing in a queue as n, particular instant of time as t and the initial arrangement of the queue in the form of a string containing 'M' representing man at position i and 'W' representing woman is at position i, find out the arrangement of the queue at time = t.

Examples :

Input : n = 6, t = 2

BBGBBG

Output: GBBGBB

Explanation:

At t = 1, 'B' at position 2 will swap with 'G' at position 3 and 'B' at position 5 will swap with 'G' at position 6. String after t = 1 changes to "BGBBGB". Now at t = 2, 'B' at position = 1 will swap with 'G' at position = 2 and 'B' at position = 4 will swap with 'G' at position 5. String changes to "GBBGBB". Since, we have to display arrangement at t = 2, the current arrangement is our answer.

Input : n = 8, t = 3
BBGBGBGB
Output: GGBGBBBB

Approach:

Traverse the entire string at every moment of time from 1 to t and if we find pairwise “BG” then swap them and move to check the next pair.

Below is the implementation of above approach:

C++

```
// CPP program to find the arrangement
// of queue at time = t
#include <bits/stdc++.h>
using namespace std;

// prints the arrangement at time = t
void solve(int n, int t, string s)
{
    // Checking the entire queue for
    // every moment from time = 1 to
    // time = t.
    for (int i = 0; i < t; i++)
        for (int j = 0; j < n - 1; j++)

            /*If current index contains 'B'
             and next index contains 'G'
             then swap*/
            if (s[j] == 'B' && s[j + 1] == 'G') {
                char temp = s[j];
                s[j] = s[j + 1];
                s[j + 1] = temp;
                j++;
            }

    cout << s;
}

// Driver function for the program
int main()
{
    int n = 6, t = 2;
    string s = "BBGBBG";
    solve(n, t, s);
    return 0;
}
```

Java

```
// Java program to find the arrangement
// of queue at time = t
import java.io.*;

class Geek {

    // prints the arrangement at time = t
    static void solve(int n, int t, char s[])
    {
        // Checking the entire queue for
        // every moment from time = 1 to
        // time = t.
        for (int i = 0; i < t; i++)
            for (int j = 0; j < n - 1; j++)

                /*If current index contains 'B'
                 and next index contains 'G'
                 then swap.*/
                if (s[j] == 'B' && s[j + 1] == 'G') {
                    char temp = s[j];
                    s[j] = s[j + 1];
                    s[j + 1] = temp;
                    j++;
                }

        System.out.print(s);
    }

    // Driver function
    public static void main(String args[])
    {
        int n = 6, t = 2;
        String s = "BBGBBG";
        char str[] = s.toCharArray();
        solve(n, t, str);
    }
}
```

Python3

```
# Python program to find
# the arrangement of
# queue at time = t

# prints the arrangement
```

```
# at time = t
def solve(n, t, p) :

    s = list(p)

    # Checking the entire
    # queue for every
    # moment from time = 1
    # to time = t.
    for i in range(0, t) :

        for j in range(0, n - 1) :

            # If current index
            # contains 'B' and
            # next index contains
            # 'G' then swap
            if (s[j] == 'B' and
                s[j + 1] == 'G') :

                temp = s[j];
                s[j] = s[j + 1];
                s[j + 1] = temp;
                j = j + 1

    print (''.join(s))

# Driver code
n = 6
t = 2
p = "BBGBBG"
solve(n, t, p)

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
// C# program to find the arrangement
// of queue at time = t
using System;

class Geek {

    // prints the arrangement at time = t
    static void solve(int n, int t, char[] s)
    {
        // Checking the entire queue for
```

```
// every moment from time = 1 to
// time = t.
for (int i = 0; i < t; i++)
    for (int j = 0; j < n - 1; j++)

        /*If current index contains 'B'
        and next index contains 'G'
        then swap.*/
        if (s[j] == 'B' && s[j + 1] == 'G')
        {
            char temp = s[j];
            s[j] = s[j + 1];
            s[j + 1] = temp;
            j++;
        }

        Console.WriteLine(s);
    }

// Driver function
public static void Main(String[] args)
{
    int n = 6, t = 2;
    String s = "BBGBBG";
    char []str = s.ToCharArray();
    solve(n, t, str);
}
}

// This code is contributed by parashar...
```

PHP

```
<?php
// PHP program to find
// the arrangement of
// queue at time = t

// prints the arrangement
// at time = t
function solve($n, $t, $s)
{
    // Checking the entire
    // queue for every
    // moment from time = 1
    // to time = t.
    for ($i = 0; $i < $t; $i++)
    {
```

```
for ($j = 0;
      $j < $n - 1; $j++)
{
    /*If current index
     contains 'B' and
     next index contains
     'G' then swap*/
    if ($s[$j] == 'B' &&
        $s[$j + 1] == 'G')
    {
        $temp = $s[$j];
        $s[$j] = $s[$j + 1];
        $s[$j + 1] = $temp;
        $j++;
    }
}
echo ($s);
}

// Driver code
$n = 6; $t = 2;
$s = "BBGBBG";
solve($n, $t, $s);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

GBBGBB

Improved By : [parashar](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/find-arrangement-queue-given-time/>

Chapter 49

Find the minimum time after which one can exchange notes

Find the minimum time after which one can exchange notes - GeeksforGeeks

Given n number of cashiers exchanging the money. At the moment, $\frac{\text{number of people}}{\text{cashier}}$ cashier had $\frac{\text{notes}}{\text{people}}$ number of people in front of him. The $\frac{\text{person}}{\text{line}}$ person in the line to $\frac{\text{cashier}}{\text{people}}$ cashier had $\frac{\text{notes}}{\text{people}}$ notes.

Find, how much early can one exchange his notes.

Time taken by the cashiers:

- The cashier took 5 seconds to scan a single note.
- After the cashier scanned every note for the customer, he took 15 seconds to exchange the notes.

Examples:

Input : n = 5
k[] = 10 10 10 10 10
m1[] = 6 7 8 6 8 5 9 8 10 5
m2[] = 9 6 9 8 7 8 8 10 8 5
m3[] = 8 7 7 8 7 5 6 8 9 5
m4[] = 6 5 10 5 5 10 7 8 5 5
m5[] = 10 9 8 7 6 9 7 9 6 5

Output : 480

Explanation: The cashier takes 5 secs for every note of each customer, therefore add $5*m[i][j]$. Each cashier spends 15 seconds for every customer, therefore add $15*k[]$ to the answer. The minimum time obtained after calculating the time taken by each cashier is our answer. Cashier m4 takes the minimum time i.e. 480.

Input : n = 1
k[] = 1
m1[] = 100
Output : 515

Approach : Calculate the total time for every cashier and minimum time obtained among all the cashier's time is the desired answer.

Below is the implementation of above approach:

C++

```
// CPP code to find minimum
// time to exchange notes
#include <bits/stdc++.h>
using namespace std;

// Function to calculate minimum
// time to exchange note
void minTimeToExchange(int k[], int m[][10],
                      int n)
{
    int min = INT_MAX;

    // Checking for every cashier
    for (int i = 0; i < n; i++)
    {
        // Time for changing the notes
        int temp = k[i] * 15;

        // Calculating scanning time
        // for every note
        for (int j = 0; j < k[i]; j++)
        {
            temp += m[i][j] * 5;
        }

        // If value in temp is minimum
        if (temp < min)
            min = temp;
    }

    cout << min;
}

// Driver function
int main()
{
```

```
// number of cashiers
int n = 5;

// number of customers with
// each cashier
int k[] = {10, 10, 10, 10, 10};

// number of notes with each customer
int m[][] = {{6, 7, 8, 6, 8, 5, 9, 8, 10, 5},
             {9, 6, 9, 8, 7, 8, 8, 10, 8, 5},
             {8, 7, 7, 8, 7, 5, 6, 8, 9, 5},
             {6, 5, 10, 5, 5, 10, 7, 8, 5, 5},
             {10, 9, 8, 7, 6, 9, 7, 9, 6, 5}};

// Calling function
minTimeToExchange(k, m, n);

return 0;
}
```

Java

```
// Java code to find minimum time to exchange
// notes
import java.io.*;

public class GFG {

    // Function to calculate minimum
    // time to exchange note
    static void minTimeToExchange(int []k,
                                  int [][]m, int n)
    {

        int min = Integer.MAX_VALUE;

        // Checking for every cashier
        for (int i = 0; i < n; i++)
        {
            // Time for changing the notes
            int temp = k[i] * 15;

            // Calculating scanning time
            // for every note
            for (int j = 0; j < k[i]; j++)
            {
                temp += m[i][j] * 5;
            }
        }
    }
}
```

```
// If value in temp is minimum
if (temp < min)
    min = temp;
}

System.out.println(min);
}

// Driver function
static public void main (String[] args)
{

    // number of cashiers
    int n = 5;

    // number of customers with
    // each cashier
    int []k = {10, 10, 10, 10, 10};

    // number of notes with each customer
    int [][]m = {
        {6, 7, 8, 6, 8, 5, 9, 8, 10, 5},
        {9, 6, 9, 8, 7, 8, 8, 10, 8, 5},
        {8, 7, 7, 8, 7, 5, 6, 8, 9, 5},
        {6, 5, 10, 5, 5, 10, 7, 8, 5, 5},
        {10, 9, 8, 7, 6, 9, 7, 9, 6, 5}};

    // Calling function
    minTimeToExchange(k, m, n);
}
}

// This code is contributed by vt_m.
```

C#

```
// C# code to find minimum
// time to exchange notes
using System;

public class GFG {

    // Function to calculate minimum
    // time to exchange note
    static void minTimeToExchange(int []k,
                                  int [,]m, int n)
{
```

```
int min = int.MaxValue;

// Checking for every cashier
for (int i = 0; i < n; i++)
{
    // Time for changing the notes
    int temp = k[i] * 15;

    // Calculating scanning time
    // for every note
    for (int j = 0; j < k[i]; j++)
    {
        temp += m[i,j] * 5;
    }

    // If value in temp is minimum
    if (temp < min)
        min = temp;
}

Console.WriteLine(min);
}

// Driver function
static public void Main (){
    // number of cashiers
    int n = 5;

    // number of customers with
    // each cashier
    int []k = {10, 10, 10, 10, 10};

    // number of notes with each customer
    int [,]m = {{6, 7, 8, 6, 8, 5, 9, 8, 10, 5},
                {9, 6, 9, 8, 7, 8, 8, 10, 8, 5},
                {8, 7, 7, 8, 7, 5, 6, 8, 9, 5},
                {6, 5, 10, 5, 5, 10, 7, 8, 5, 5},
                {10, 9, 8, 7, 6, 9, 7, 9, 6, 5}};

    // Calling function
    minTimeToExchange(k, m, n);
}

// This code is contributed by vt_m.
```

```
<?php
// PHP code to find minimum
// time to exchange notes

// Function to calculate minimum
// time to exchange note
function minTimeToExchange($k, $m,
                           $n)
{
    $min = PHP_INT_MAX;

    // Checking for every cashier
    for ( $i = 0; $i < $n; $i++)
    {

        // Time for changing the notes
        $temp = $k[$i] * 15;

        // Calculating scanning time
        // for every note
        for ($j = 0; $j < $k[$i]; $j++)
        {
            $temp += $m[$i][$j] * 5;
        }

        // If value in temp is minimum
        if ($temp < $min)
            $min = $temp;
    }

    echo $min;
}

// Driver Code
// number of cashiers
$n = 5;

// number of customers with
// each cashier
$k = array(10, 10, 10, 10, 10);

// number of notes with
// each customer
$m = array(array(6, 7, 8, 6, 8, 5, 9, 8, 10, 5),
           array(9, 6, 9, 8, 7, 8, 8, 10, 8, 5),
           array(8, 7, 7, 8, 7, 5, 6, 8, 9, 5),
           array(6, 5, 10, 5, 5, 10, 7, 8, 5, 5),
           array(10, 9, 8, 7, 6, 9, 7, 9, 6, 5));
```

```
// Calling function  
minTimeToExchange($k, $m, $n);  
  
// This code is contributed by anuj_67.  
?>
```

Output:

480

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-minimum-time-one-can-exchange-notes/>

Chapter 50

Find the number of operations required to make all array elements Equal

Find the number of operations required to make all array elements Equal - GeeksforGeeks

Given an array of **N** integers, the task is to find the number of operations required to make all elements in the array equal. In one operation we can distribute equal weights from the maximum element to the rest of the array elements. If it is not possible to make the array elements equal after performing the above operations then print -1.

Examples:

Input: arr = [1, 6, 1, 1, 1];

Output: 4

Explanation: Since arr becomes [2, 2, 2, 2, 2] after distribution from max element.

Input : arr = [2, 2, 3];

Output : -1

Explanation: Here arr becomes [3, 3, 1] after distribution.

Algorithm:

- Declare temporary variable to store number of times operation is performed.
- Find maximum element of the given array and store its index value.
- Check if all the elements are equal to the maximum element after n subtractions.
- Again check that each element is equal to other elements and return n.

Below is the implementation of above approach:

```
# Python program to find the number
# of operations required to make
# all array elements Equal

# Function to find maximum
# element of the given array
def find_n(a):
    j, k = 0, 0

    x = max(a)
    for i in range(len(a)):
        if(a[i] == x):
            s = i
            break

    for i in a:
        if(i != x and i <= min(a) and i != '\0'):
            a[j] += 1
            a[s] -= 1
            x -= 1
            k += 1
            j += 1
        elif(i != '\0'):
            j += 1

    for i in range(len(a)):
        if(a[i] != x):
            k = -1
            break

    return k

# Driver Code
a = [1, 6, 1, 1, 1]
print (find_n(a))
```

Output:

4

Time complexity: O(n)

Source

<https://www.geeksforgeeks.org/find-the-number-of-operations-required-to-make-all-array-elements-equal/>

Chapter 51

Find two numbers from their sum and XOR

Find two numbers from their sum and XOR - GeeksforGeeks

Given the sum and xor of two numbers **X** and **Y** s.t. sum and xor we need to find the numbers minimizing the value of **X**.

Examples :

```
Input : sum = 17
        xor = 13
Output : X = 2
         Y = 15
```

```
Input : sum = 1870807699
        xor = 259801747
Output : X = 805502976
         Y = 1065304723
```

```
Input : sum = 1639
        xor = 1176
Output : No such numbers exist
```

Let the summation be **S** and xor be **xo**. The summation and xor operations both are commutative in nature. This means that for a corresponding set bit in **X** if we swap it with an unset bit at same position in **Y**, the sum and xor would remain unaffected. The only affect it would make is that it will reduce the value of **X** and increase the value of **Y**.

For example, **X** = 7 and **Y** = 10, **X+Y** = **S** = 17 and **X[^]Y** = **xo** = 13.
Binary representation of **X** = 0111

Binary representation of Y = 1010

Now, for a given position in X which is 1, if for the same position in Y is 0 and we swap it;

New binary representation of X = 0010

New binary representation of Y = 1111

New value of X = 2

New Value of Y = 15

Their sum and xor are still the same.

Now, let's use the above piece of information to exploit our problem statement because following the above steps, we have successfully minimized the value of X. We can even conclude from the above proof that $\mathbf{Y} = \mathbf{x}_0 + \mathbf{X}$

Now, we know that $\mathbf{X} + \mathbf{Y} = \mathbf{S}$ and we also know that $\mathbf{Y} = \mathbf{X} + \mathbf{x}_0$, now if we equate the given equations, we get the results are

$$\begin{aligned} \mathbf{X} &= \frac{\mathbf{S} - \mathbf{Y}}{2} \\ \mathbf{Y} &= \mathbf{X} + \mathbf{x}_0 \end{aligned}$$

Using the above equations we can easily compute the required values of X and Y.

C++

```
// CPP program to find two numbers with
// given Sum and XOR such that value of
// first number is minimum.
#include <iostream>
using namespace std;

// Function that takes in the sum and XOR
// of two numbers and generates the two
// numbers such that the value of X is
// minimized
void compute(unsigned long int S,
             unsigned long int Xo)
{
    // If sum becomes less than Xor
    // in this case, no solution shall
    // exist.
    if (S < Xo)
        cout << "No such numbers exist";
    else {

        // Application of the derived formula!
        int x = (S - Xo) / 2;
        cout << "X = " << x << endl;
        cout << "Y = " << (x + Xo) << endl;
    }
}

// Driver function
int main()
```

```
{  
    unsigned long int S = 17, Xo = 13;  
    compute(S, Xo);  
    return 0;  
}
```

Java

```
// Java program to find two numbers  
// with given Sum and XOR such that  
// value of first number is minimum.  
class GFG{  
  
    // Function that takes in the  
    // sum and XOR of two numbers  
    // and generates the two numbers  
    // such that the value of X is  
    // minimized  
    static void compute(int S, int Xo)  
    {  
  
        // If sum becomes less than  
        // Xor in this case, no solution  
        // shall exist.  
        if (S < Xo)  
            System.out.printf  
                ("No such numbers exist");  
        else {  
  
            // Application of the derived  
            // formula!  
            int x = (S - Xo) / 2;  
            System.out.printf  
                ( "X = %d\n", x);  
            System.out.printf  
                ( "Y = %d", (x + Xo));  
        }  
    }  
  
    // Driver function  
    public static void main(String[] args)  
    {  
        int S = 17, Xo = 13;  
  
        compute(S, Xo);  
    }  
}
```

```
// This code is contributed by  
// Smitha Dinesh Semwal.
```

Python3

```
# Python program to find  
# two numbers with  
# given Sum and XOR such  
# that value of  
# first number is minimum.  
  
# Function that takes in  
# the sum and XOR  
# of two numbers and  
# generates the two  
# numbers such that the  
# value of X is  
# minimized  
def compute(S, Xo):  
  
    # If sum becomes less than Xor  
    # in this case, no solution shall  
    # exist.  
    if (S < Xo):  
        print("No such numbers exist")  
    else :  
  
        # Application of the derived formula!  
        x = int((S - Xo) / 2)  
        print("X =",x)  
        print("Y =", (x + Xo))  
  
# Driver function  
S = 17  
Xo = 13  
compute(S, Xo)  
  
# This code is contributed by  
# Smitha Dinesh Semwal
```

C#

```
// C# program to find two numbers  
// with given Sum and XOR such that  
// value of first number is minimum.  
using System;
```

```
class GFG
{
    // Function that takes in the
    // sum and XOR of two numbers
    // and generates the two numbers
    // such that the value of X is
    // minimized
    public static void compute(int S,
                               int Xo)
    {
        // If sum becomes less than
        // Xor in this case, no
        // solution shall exist.
        if (S < Xo)
            Console.WriteLine("No such " +
                "numbers exist");
        else
        {
            // Application of the
            // derived formula!
            int x = (S - Xo) / 2;
            Console.WriteLine("X = "+ x);
            Console.WriteLine("Y = "+ (x + Xo));
        }
    }

    // Driver Code
    static public void Main ()
    {
        int S = 17, Xo = 13;

        compute(S, Xo);
    }
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP program to find two numbers
// with given Sum and XOR such that
// value of first number is minimum.
```

```
// Function that takes in the sum
// and XOR of two numbers and
// generates the two numbers such
// that the value of X is minimized
function compute($S, $Xo)
{
    // If sum becomes less than
    // Xor in this case, no
    // solution shall exist.
    if ($S < $Xo)
        echo "No such numbers exist";
    else
    {

        // Application of the
        // derived formula!
        $x = ($S - $Xo) / 2;
        echo "X = " , $x , "\n";
        echo "Y = " , ($x + $Xo) , "\n";
    }
}

// Driver Code
$S = 17;
$Xo = 13;
compute($S, $Xo);

// This code is contributed by ajit
?>
```

Output :

```
X = 2
Y = 15
```

Time complexity of the above approach 

Improved By : [jit_t](#), Rajnish Ranjan

Source

<https://www.geeksforgeeks.org/find-two-numbers-sum-xor/>

Chapter 52

First occurrence of a digit in a given fraction

First occurrence of a digit in a given fraction - GeeksforGeeks

Given three integers a, b and c, find the first occurrence of c in a/b after the decimal point. If it does not exists, print -1.

Examples:

```
Input : a = 2 b = 3 c = 6
Output : 1
Explanation:
0.666666.. so 6 occurs at first place
of a/b after decimal point
```

```
Input : a = 1 b = 4 c = 5
Output : 2
Explanation:
1 / 4 = 0.25 which gives 5's position
to be 2.
```

A **naive approach** will be to perform the division and keep the decimal part and iterate and check if the given number exists or not. This will not work well when divisions such as 2/3 is done as it yields 0.666666666, but in programming language it will round it off to 0.666667 so we get a 7 also which does not exists in the original a/b

An **efficient approach** will be the mathematical one, if we modulate every-time a by b and multiply it with 10 we get the integers after the decimal part every-time. The number of modulations required will be b as it will have a maximum of b integers after decimal point. So, we compare it with c and get our desired value if it is present.

Below is the implementation of the above approach :

C++

```
// CPP program to find first occurrence
// of c in a/b
#include <bits/stdc++.h>
using namespace std;

// function to print the first digit
int first(int a, int b, int c)
{
    // reduce the number to its mod
    a %= b;

    // traverse for every decimal places
    for (int i = 1; i <= b; i++)
    {
        // get every fraction places
        // when (a*10/b)/c
        a = a * 10;

        // check if it is equal to
        // the required integer
        if (a / b == c)
            return i;

        // mod the number
        a %= b;
    }
    return -1;
}

// driver program to test the above function
int main()
{
    int a = 1, b = 4, c = 5;
    cout << first(a, b, c);
    return 0;
}
```

Java

```
// Java program to find first occurrence
// of c in a/b
import java.util.*;
import java.lang.*;

public class GfG{
```

```
// Function to print the first digit
public static int first(int a, int b, int c)
{
    // Reduce the number to its mod
    a %= b;

    // Traverse for every decimal places
    for (int i = 1; i <= b; i++)
    {
        // Get every fraction places
        // when (a*10/b)/c
        a = a * 10;

        // Check if it is equal to
        // the required integer
        if (a / b == c)
            return i;

        // Mod the number
        a %= b;
    }
    return -1;
}

// Driver function
public static void main(String argc[]){
    int a = 1, b = 4, c = 5;
    System.out.println(first(a, b, c));
}
```

/* This code is contributed by Sagar Shukla */

Python3

```
# Python3 program to find first occurrence
# of c in a/b

# function to print the first digit
def first( a , b , c ):

    # reduce the number to its mod
    a %= b

    # traverse for every decimal places
    for i in range(1, b + 1):

        # get every fraction places
```

```
# when (a*10/b)/c
a = a * 10

# check if it is equal to
# the required integer
if int(a / b) == c:
    return i

# mod the number
a %= b

return -1

# driver code to test the above function
a = 1
b = 4
c = 5
print(first(a, b, c))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to find first occurrence
// of c in a/b
using System;

public class GfG{

    // Function to print the first digit
    public static int first(int a, int b, int c)
    {

        // Reduce the number to its mod
        a %= b;

        // Traverse for every decimal places
        for (int i = 1; i <= b; i++)
        {

            // Get every fraction places
            // when (a*10/b)/c
            a = a * 10;

            // Check if it is equal to
            // the required integer
            if (a / b == c)
                return i;
        }
    }
}
```

```
// Mod the number
a %= b;
}

return -1;
}

// Driver function
public static void Main() {

    int a = 1, b = 4, c = 5;

    Console.WriteLine(first(a, b, c));
}
}

/* This code is contributed by vt_m */
```

PHP

```
<?php
// PHP program to find first
// occurrence of c in a/b

// function to print
// the first digit
function first( $a, $b, $c)
{

    // reduce the number
    // to its mod
    $a %= $b;

    // traverse for every
    // decimal places
    for ($i = 1; $i <= $b; $i++)
    {

        // get every fraction places
        // when (a*10/b)/c
        $a = $a * 10;

        // check if it is equal to
        // the required integer
        if ($a / $b == $c)
            return $i;
    }
}
```

```
// mod the number
$a %= $b;
}
return -1;
}

// Driver Code
$a = 1; $b = 4; $c = 5;
echo first($a, $b, $c);

// This code is contributed by anuj_67.
?>
```

Output:

2

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/first-occurrence-of-a-digit-in-a-given-fraction/>

Chapter 53

Formatted output in Java

Formatted output in Java - GeeksforGeeks

Sometimes in Competitive programming, it is essential to print the output in a given specified format. Most users are familiar with printf function in C. Let us see discuss how we can format the output in Java:

Formatting output using System.out.printf()

This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments.

```
// A Java program to demonstrate working of printf() in Java
class JavaFormatter1
{
    public static void main(String args[])
    {
        int x = 100;
        System.out.printf("Printing simple integer: x = %d\n", x);

        // this will print it upto 2 decimal places
        System.out.printf("Formatted with precision: PI = %.2f\n", Math.PI);

        float n = 5.2f;

        // automatically appends zero to the rightmost part of decimal
        System.out.printf("Formatted to specific width: n = %.4f\n", n);

        n = 2324435.3f;

        // here number is formatted from right margin and occupies a
        // width of 20 characters
        System.out.printf("Formatted to right margin: n = %20.4f\n", n);
    }
}
```

Output:

```
Printing simple integer: x = 100
Formatted with precision: PI = 3.14
Formatted to specific width: n = 5.2000
Formatted to right margin: n =          2324435.2500
```

[System.out.format\(\)](#) is equivalent to printf() and can also be used.

Formatting using DecimalFormat class:

DecimalFormat is used to format decimal numbers.

```
// Java program to demonstrate working of DecimalFormat
import java.text.DecimalFormat;

class JavaFormatter2
{
    public static void main(String args[])
    {
        double num = 123.4567;

        // prints only numeric part of a floating number
        DecimalFormat ft = new DecimalFormat("####");
        System.out.println("Without fraction part: num = " + ft.format(num));

        // this will print it upto 2 decimal places
        ft = new DecimalFormat("#.##");
        System.out.println("Formatted to Give precision: num = " + ft.format(num));

        // automatically appends zero to the rightmost part of decimal
        // instead of #, we use digit 0
        ft = new DecimalFormat("#.000000");
        System.out.println("appended zeroes to right: num = " + ft.format(num));

        // automatically appends zero to the leftmost of decimal number
        // instead of #, we use digit 0
        ft = new DecimalFormat("00000.00");
        System.out.println("formatting Numeric part : num = "+ft.format(num));

        // formatting money in dollars
        double income = 23456.789;
        ft = new DecimalFormat("$###,###.##");
```

```
        System.out.println("your Formatted Dream Income : " + ft.format(income));
    }
}
```

Output:

```
Without fraction part: num = 123
Formatted to Give precision: num = 123.46
appended zeroes to right: num = 123.456700
formatting Numeric part : num = 00123.46
your Formatted Dream Income : $23,456.79
```

Formatting dates and parsing using SimpleDateFormat class:

This class is present in java.text package.

```
// Java program to demonstrate working of SimpleDateFormat
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

class Formatter3
{
    public static void main(String args[]) throws ParseException
    {
        // Formatting as per given pattern in the argument
        SimpleDateFormat ft = new SimpleDateFormat("dd-MM-yyyy");
        String str = ft.format(new Date());
        System.out.println("Formatted Date : " + str);

        // parsing a given String
        str = "02/18/1995";
        ft = new SimpleDateFormat("MM/dd/yyyy");
        Date date = ft.parse(str);

        // this will print the date as per parsed string
        System.out.println("Parsed Date : " + date);
    }
}
```

Output:

```
Formatted Date : 09-08-2018
Parsed Date : Sat Feb 18 00:00:00 UTC 1995
```

References:

<https://docs.oracle.com/javase/tutorial/essential/io/formatting.html>
<https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>
<http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html>

Improved By : [Mayur_jain](#)

Source

<https://www.geeksforgeeks.org/formatted-output-in-java/>

Chapter 54

Frequency Measuring Techniques for Competitive Programming

Frequency Measuring Techniques for Competitive Programming - GeeksforGeeks

Measuring frequency of elements in an array is a really handy skill and is required a lot of competitive coding problems. We, in lot of problems are required to measure frequency of various elements like numbers, alphabets, symbols, etc. as a part of our problem.

Naive method

```
Input : arr[] = {10, 20, 20, 10, 10, 20, 5, 20}
Output : 10 3
          20 4
          5 1

Input : arr[] = {10, 20, 20}
Output : 10 2
          20 1
```

We run two loops. For every item count number of times it occurs. To avoid duplicate printing, keep track of processed items.

```
// CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n)
{
```

```
// Mark all array elements as not visited
vector<int> visited(n, false);

// Traverse through array elements and
// count frequencies
for (int i = 0; i < n; i++) {

    // Skip this element if already processed
    if (visited[i] == true)
        continue;

    // Count frequency
    int count = 1;
    for (int j = i + 1; j < n; j++) {
        if (arr[i] == arr[j]) {
            visited[j] = true;
            count++;
        }
    }
    cout << arr[i] << " " << count << endl;
}
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Optimized methods :

Measuring frequencies when elements are limited by value

If our input array has small values, we can use array elements as index in a count array and increment count. In below example, elements are maximum 10.

```
Input : arr[] = {5, 5, 6, 6, 5, 6, 1, 2, 3, 10, 10}
        limit = 10
Output : 1 1
```

```
2 1
3 1
5 3
6 3
10 2

// CPP program to count frequencies of array items
// having small values.
#include <bits/stdc++.h>
using namespace std;

void countFreq(int arr[], int n, int limit)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through array elements and
    // count frequencies (assuming that elements
    // are limited by limit)
    for (int i = 0; i < n; i++)
        count[arr[i]]++;

    for (int i = 0; i <= limit; i++)
        if (count[i] > 0)
            cout << i << " " << count[i] << endl;
}

int main()
{
    int arr[] = {5, 5, 6, 6, 5, 6, 1, 2, 3, 10, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    int limit = 10;
    countFreq(arr, n, limit);
    return 0;
}
```

Output:

```
1 1
2 1
3 1
5 3
6 3
10 2
```

```
// CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;

const int limit = 255;

void countFreq(string str)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through string characters and
    // count frequencies
    for (int i = 0; i < str.length(); i++)
        count[str[i]]++;

    for (int i = 0; i <= limit; i++)
        if (count[i] > 0)
            cout << (char)i << " " << count[i] << endl;
}

int main()
{
    string str = "GeeksforGeeks";
    countFreq(str);
    return 0;
}
```

Output:

```
G 2
e 4
f 1
k 2
o 1
r 1
s 2
```

Measuring frequencies when elements are in limited range

For example consider a string containing only upper case alphabets. Elements of string are limited in range from 'A' to 'Z'. The idea is to subtract smallest element ('A' in this example) to get index of the element.

```
// CPP program to count frequencies of array items
```

```
#include <bits/stdc++.h>
using namespace std;

const int limit = 25;

void countFreq(string str)
{
    // Create an array to store counts. The size
    // of array is limit+1 and all values are
    // initially 0
    vector<int> count(limit+1, 0);

    // Traverse through string characters and
    // count frequencies
    for (int i = 0; i < str.length(); i++)
        count[str[i] - 'A']++;

    for (int i = 0; i <= limit; i++)
        if (count[i] > 0)
            cout << (char)(i + 'A') << " " << count[i] << endl;
}

int main()
{
    string str = "GEEKSFORGEEKS";
    countFreq(str);
    return 0;
}
```

Output:

```
E 4
F 1
G 2
K 2
O 1
R 1
S 2
```

Measuring frequencies if no range and no limit

The idea is to use hashing ([unordered_map in C++](#) and [HashMap in Java](#)) to get frequencies.

```
// CPP program to count frequencies of array items
#include <bits/stdc++.h>
```

```
using namespace std;

void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse through map and print frequencies
    for (auto x : mp)
        cout << x.first << " " << x.second << endl;
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
5 1
10 3
20 4
```

Output:

```
5 1
10 3
20 4
```

Time Complexity : O(n)
Auxiliary Space : O(n)

In above efficient solution, how to print elements in same order as they appear in input?

```
// CPP program to count frequencies of array items
#include <bits/stdc++.h>
using namespace std;
```

```
void countFreq(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Traverse through array elements and
    // count frequencies
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // To print elements according to first
    // occurrence, traverse array one more time
    // print frequencies of elements and mark
    // frequencies as -1 so that same element
    // is not printed multiple times.
    for (int i = 0; i < n; i++) {
        if (mp[arr[i]] != -1)
        {
            cout << arr[i] << " " << mp[arr[i]] << endl;
            mp[arr[i]] = -1;
        }
    }
}

int main()
{
    int arr[] = { 10, 20, 20, 10, 10, 20, 5, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    countFreq(arr, n);
    return 0;
}
```

Output:

```
10 3
20 4
5 1
```

Output:

```
10 3
20 4
5 1
```

Time Complexity : $O(n)$
Auxiliary Space : $O(n)$

In Java, we can get elements in same order using [LinkedHashMap](#). Therefore we do not need an extra loop.

Lot of problems are based on frequency measurement and will be a cheesecake if we know how to calculate frequency of various elements in a given array. For example try the given below problems which are based on frequency measurement:

1. [Anagrams](#)
2. [Sorting Elements of an Array by Frequency](#)
3. [Single Number](#)

Source

<https://www.geeksforgeeks.org/frequency-measurement-techniques-for-competitive-programming/>

Chapter 55

Generating Test Cases (`generate()` and `generate_n()` in C++)

Generating Test Cases (`generate()` and `generate_n()` in C++) - GeeksforGeeks

Generating test cases for array programs can be a cumbersome process. But the `generate` and `generate_n` functions in the STL (Standard Template Library), come handy to populate the array with random values.

- **`generate()`**

The `generate` functions assigns random values provided by calling the generator function ‘gen’ to the elements in the range $[begin, end)$. Notice that `begin` is included in the range but `end` is NOT included.

Following code demonstrates the implementation of `generate` :

```
// C++ program to demonstrate generate function in STL
#include <bits/stdc++.h>
using namespace std;

// function to generate random numbers in range [0-999] :
int randomize()
{
    return (rand() % 1000);
}

int main ()
{
    // for different values each time we run the code
    srand(time(NULL));
```

```
vector<int> vect(10); // declaring the vector

// Fill all elements using randomize()
generate(vect.begin(), vect.end(), randomize);

// displaying the content of vector
for (int i=0; i<vect.size(); i++)
    cout << vect[i] << " " ;

return 0;
}
```

Output :

```
832 60 417 710 487 260 920 803 576 58
```

NOTE : The output would be different each time we run the code because of strand. If we remove strand, we would get the same set of random numbers every time we run the code.

- **generate_n()**

The generate_n does the same job as generate upto n elements starting from the element pointed to by the begin iterator.

The following code demonstrates the working of generate_n :

```
// C++ program to demonstrate generate_n() function in STL
#include <bits/stdc++.h>
using namespace std;

// function to generate random numbers in range [0-999] :
int randomize()
{
    return (rand() % 1000);
}

int main ()
{
    // for different values each time we run the code
    srand(time(NULL));

    vector<int> vect(10); // declaring the vector

    // Fill 6 elements from beginning using randomize()
    generate_n(vect.begin(), 6, randomize);
```

```
// displaying the content of vector
for (int i=0; i<vect.size(); i++)
    cout << vect[i] << " " ;

    return 0;
}
```

Output :

```
177 567 15 922 527 4 0 0 0 0
```

NOTE : Here also, the output would be different each time we run the code because of strand. If we remove strand, we would get the same set of random numbers every time we run the code.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/generating-test-cases-generate-and-generate_n-in-c/

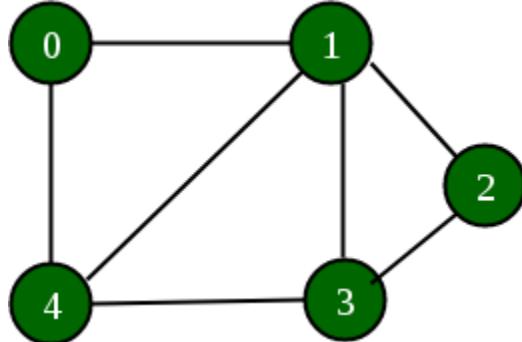
Chapter 56

Graph implementation using STL for competitive programming | Set 1 (DFS of Unweighted and Undirected)

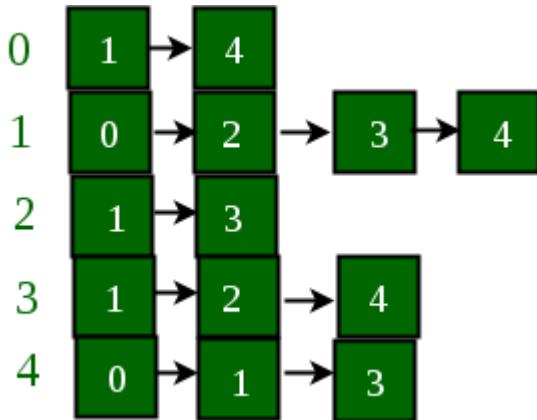
Graph implementation using STL for competitive programming | Set 1 (DFS of Unweighted and Undirected) - GeeksforGeeks

We have introduced Graph basics in [Graph and its representations](#). In this post, a different STL based representation is used that can be helpful to quickly implement graph using [vectors](#). The implementation is for adjacency list representation of graph.

Following is an example undirected and unweighted graph with 5 vertices.



Below is adjacency list representation of the graph.



We use vector in STL to implement graph using adjacency list representation.

- **vector** : A sequence container. Here we use it to store adjacency lists of all vertices.
We use vertex number as index in this vector.

The idea is to represent graph as an array of vectors such that every vector represents adjacency list of a vertex. Below is complete STL based C++ program for [DFS Traversal](#).

```

// A simple representation of graph using STL,
// for the purpose of competitive programming
#include<bits/stdc++.h>
using namespace std;

// A utility function to add an edge in an
// undirected graph.
void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

// A utility function to do DFS of graph
// recursively from a given vertex u.
void DFSUtil(int u, vector<int> adj[],
             vector<bool> &visited)
{
    visited[u] = true;
    cout << u << " ";
    for (int i=0; i<adj[u].size(); i++)
        if (visited[adj[u][i]] == false)
            DFSUtil(adj[u][i], adj, visited);
}

// This function does DFSUtil() for all
  
```

```
// unvisited vertices.  
void DFS(vector<int> adj[], int V)  
{  
    vector<bool> visited(V, false);  
    for (int u=0; u<V; u++)  
        if (visited[u] == false)  
            DFSUtil(u, adj, visited);  
}  
  
// Driver code  
int main()  
{  
    int V = 5;  
  
    // The below line may not work on all  
    // compilers. If it does not work on  
    // your compiler, please replace it with  
    // following  
    // vector<int> *adj = new vector<int>[V];  
    vector<int> adj[V];  
  
    // Vertex numbers should be from 0 to 4.  
    addEdge(adj, 0, 1);  
    addEdge(adj, 0, 4);  
    addEdge(adj, 1, 2);  
    addEdge(adj, 1, 3);  
    addEdge(adj, 1, 4);  
    addEdge(adj, 2, 3);  
    addEdge(adj, 3, 4);  
    DFS(adj, V);  
    return 0;  
}
```

Output :

0 1 2 3 4

Below are related articles:

[Graph implementation using STL for competitive programming | Set 2 \(Weighted graph\)](#)
[Dijkstra's Shortest Path Algorithm using priority_queue of STL](#)
[Dijkstra's shortest path algorithm using set in STL](#)
[Kruskal's Minimum Spanning Tree using STL in C++](#)
[Prim's algorithm using priority_queue in STL](#)

Source

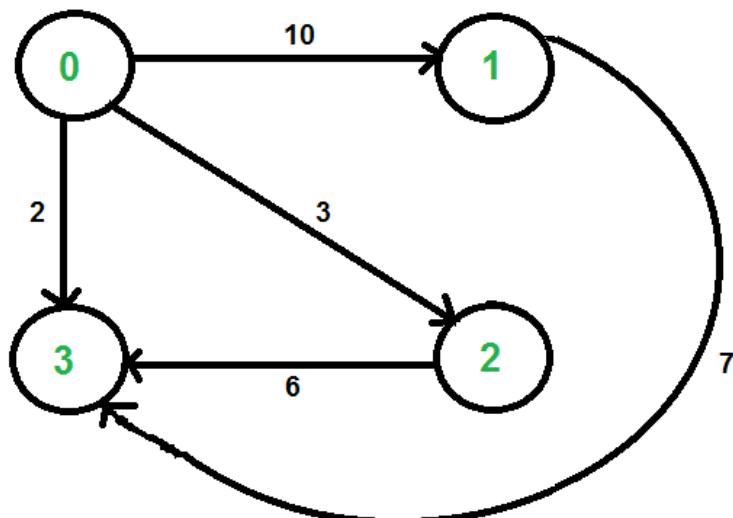
<https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-1-dfs-of-unweight>

Chapter 57

Graph implementation using STL for competitive programming | Set 2 (Weighted graph)

Graph implementation using STL for competitive programming | Set 2 (Weighted graph) - GeeksforGeeks

In [Set 1](#), unweighted graph is discussed. In this post, weighted graph representation using STL is discussed. The implementation is for adjacency list representation of weighted graph.



We use two STL containers to represent graph:

- **vector** : A sequence container. Here we use it to store adjacency lists of all vertices. We use vertex number as index in this vector.
- **pair** : A simple container to store pair of elements. Here we use it to store adjacent vertex number and weight of edge connecting to the adjacent.

The idea is to use a vector of pair vectors. Below code implements the same.

```
// C++ program to represent undirected and weighted graph
// using STL. The program basically prints adjacency list
// representation of graph
#include <bits/stdc++.h>
using namespace std;

// To add an edge
void addEdge(vector<pair<int, int>> adj[], int u,
             int v, int wt)
{
    adj[u].push_back(make_pair(v, wt));
    adj[v].push_back(make_pair(u, wt));
}

// Print adjacency list representation of graph
void printGraph(vector<pair<int,int>> adj[], int V)
{
    int v, w;
    for (int u = 0; u < V; u++)
    {
        cout << "Node " << u << " makes an edge with \n";
        for (auto it = adj[u].begin(); it!=adj[u].end(); it++)
        {
            v = it->first;
            w = it->second;
            cout << "\tNode " << v << " with edge weight ="
                << w << "\n";
        }
        cout << "\n";
    }
}

// Driver code
int main()
{
    int V = 5;
    vector<pair<int, int>> adj[V];
    addEdge(adj, 0, 1, 10);
    addEdge(adj, 0, 4, 20);
    addEdge(adj, 1, 2, 30);
    addEdge(adj, 1, 3, 40);
```

```
    addEdge(adj, 1, 4, 50);
    addEdge(adj, 2, 3, 60);
    addEdge(adj, 3, 4, 70);
    printGraph(adj, V);
    return 0;
}
```

Output:

```
Node 0 makes an edge with
    Node 1 with edge weight =10
    Node 4 with edge weight =20

Node 1 makes an edge with
    Node 0 with edge weight =10
    Node 2 with edge weight =30
    Node 3 with edge weight =40
    Node 4 with edge weight =50

Node 2 makes an edge with
    Node 1 with edge weight =30
    Node 3 with edge weight =60

Node 3 makes an edge with
    Node 1 with edge weight =40
    Node 2 with edge weight =60
    Node 4 with edge weight =70

Node 4 makes an edge with
    Node 0 with edge weight =20
    Node 1 with edge weight =50
    Node 3 with edge weight =70
```

Improved By : [Ritesh Ghorse](#)

Source

<https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-2-weighted-graph/>

Chapter 58

How can competitive programming help you get a job?

How can competitive programming help you get a job? - GeeksforGeeks

Let us first know about competitive programming!

Competitive Programming

Competitive programming is a brain game which take place on the Internet or a local network in which programmers have to code according to the given constraints. Here programmers are referred as competitive coders. Many top notch companies like Google, Facebook host contests like Codejam and Hackercup respectively. Those who perform well in these contests are recognised by these companies and get offers to work with these tech giants.

- **Publicly demonstrate your skills**

Competitive programmers are known for their problem solving skills. Like developers show their skills by making different projects, competitive programmers show their talent by taking part in different challenges which sites like Codeforces, Codechef, Topcoder, Hackerrank, Hackerearth and many more host frequently. Competitive programmers build their name and earn fame on these sites and as they perform good, people start to recognize them.

- **Prepare you for a Technical Interview**

As you get used to solving harder and harder problem in contests, you will easily be able to answer questions asked in technical interview. Competitive programming also increases your problem solving speed which provides a edge to you over other applicants.

- **Makes you desirable Candidate for major Companies**

Big companies like Apple, Google and Facebook want talented and smart people to work with them. So these companies keep an eye on those programmers who

outperform worldwide in the contests which take place at world level. One such contest is ACM ICPC, it is like olympics for a competitive programmer. You will definitely get an opportunity to work with these companies if you perform well in world level contests.

- **Makes you more faster and focussed**

You will become faster in every aspect of your life. You start finish your tasks quickly in your real life as well. This is an excellent skill which you develop. It helps you become more focussed as your code gets accepted only when your all test cases passes. So you start developing a habit of analyzing each and every factor which can affect your code. Hence in life also you don't miss any factor which remains unconsidered easily.

- **Helps you solve Complicated Problems**

While solving a question in competitive programming, most of the time you get wrong answer and you face a failure. By solving lots of questions, you will overcome the fear of failure. Competitive Programmers perform under pressure and take out a solution which builds their real life problem facing skills.

For example : You are opening a business, then you won't have fear of failure. You will handle any situation which comes in your way and will overcome it easily.

- **Guaranteed Brain Exercise**

Many a times, we come across a condition when we think that I have not done anything productive today. By solving 2 or 3 problems and getting correct answer for that helps you feel motivated. You will feel that yes I have applied my brain in solving these problems, which boosts your motivation.

- **Teaches you how to work in Teams**

Many contests take place at individual level and many contests involve team participation. You give contest in a group of 2 or 3. So you start to learn how to approach a situation in a group. Some person has good dynamic programming, some code faster, some think of a solution faster. In this way you learn to divide the work in team.

It helps when you are working a company and doing work in a project.

- **It's FUN!**

One of the biggest factor is that competitive programming gives you a real time fun. Many people play football, cricket. They get recognition from various people that this person is very good in that particular sport. In the same way when you do Competitive Programming, you compete at the world level, among your peers. You start getting fame and recognition from people that this person's algorithmic approach is fantastic. It feels nice when you hear these kind of words. So it is thrilling to do Competitive Programming.

Various platform where you can showcase your skills

1. Codeforces
2. Codechef
3. Topcoder
4. Hackerearth
5. Hackerrank

Various yearly top competitions

1. ACM ICPC
2. Codejam
3. Hackercup

So guys, start competitive programming today if you have not geared up. It definitely helps you get a good job.

Related Article :

[Practice for cracking any coding interview](#)

Source

<https://www.geeksforgeeks.org/how-can-competitive-programming-help-you-get-a-job/>

Chapter 59

How to become a master in competitive programming?

How to become a master in competitive programming? - GeeksforGeeks

There are many people for whom programming is like a haunted dream. Programming is nothing but an art of talking with machines and telling them what to do, when to do, and why to do. Most of the students hear this word in high school. For many of them programming starts with ‘C’ and ends at ‘C’. There’s no problem in choosing any specific programming language but getting stuck at some well-known codes or the codes required only for the exam clarification is futile. The coming era is the age of technology. And here comes in picture the art called “competitive programming”. **Competitive programming** is an advanced form of programming which deals with real world problems. Here we see our code ruling the world. But writing such a code requires dexterity with passion.

We know that a code is basically our logic behind any problem in high level language. But logic alone is not sufficient in writing a perfect code. It requires deeper understanding of technical terms like complexity, syntax, and the art of creating big solutions through shortest codes possible. All this can be achieved only through practice. But if practice fuses with a good guidance, it explodes into a masterpiece. This goal can be achieved through the following simple steps:-

1. Get thorough understanding

First of all study all the concepts of the programming language deeply. Always use standard books. Today many online platforms are available where geeks from all around the world share their knowledge and try to make the concepts easier. One such platform is “geeksforgeeks” where you can find some of the most useful articles.

2. Follow a hierarchical approach

Try to start coding using simpler problems. Before directly writing the code, first make a flowchart of the logic being used. This will increase the number of correct codes which will not only sharpen your skills but also boost up your confidence.

3. Implementation in real life

Once you get used to with the codes and the basic programming try to make codes that solve your daily life problems. These may include report card of any student, ticket reservation system, library management system, etc. They will make you feel like a software developer.

4. Truncate the code

Now the next step is shortening the code. Suppose you make a simple code for library management system. Now try to abbreviate it such that the same task can be accomplished in a much simpler and shorter way. You can first just see the problem and make your own code. Now see the optimal solution to learn how it can be reduced. This is the most important and the transition phase from basic programming to competitive programming.

5. Be a fighter

Now start participating in coding competitions. The competitions may be in your school, online or at national level. Here you will find guys like you competing and beating each other. Here, you have to write the optimal solution and that too within the shortest time possible. Obviously, since it's a competition so it's survival of the fittest. Such a healthy competitive environment makes the learning rate faster and involves learn by fun mechanism. Apart from this you will also get a rating according your successful submissions of the code and the competitions you win which strengthens your professional profile.

6. Start spreading the “GYAN”

Once you become a code-keeda, don't keep your knowledge to yourself. Spread it. Share it with your juniors, your peers and through the world. There is a popular proverb that says, “gyan baantne se badhta hai” – “Your knowledge increases more if you share it with others”

7. Be updated

“Success is not a destination you arrive at, it is a manner of living.” So, always be updated of the new technologies and the new amendments in the field of coding. It will help you produce better and serve better.

Thus following the above sapt-karma (seven steps) you can surely become a master in competitive programming and can serve the society in a much better and a much advanced way. And believe me guys when you see the power of these codes in real world applications, that is, when you see robots working on the commands you write, trains moving as per your codes, a nation's defence mechanism being secured by your codes, it feels like heaven. So go ahead guys, be a programmer and contribute in making India a Digital India, and the whole world the Digital World.

Source

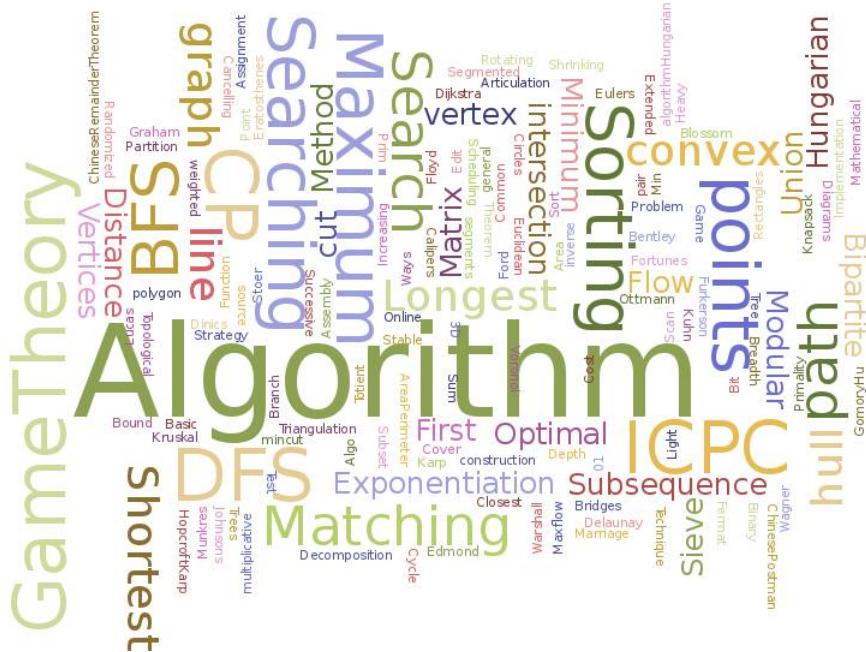
<https://www.geeksforgeeks.org/how-to-become-a-master-in-competitive-programming/>

Chapter 60

How to begin with Competitive Programming?

How to begin with Competitive Programming? - GeeksforGeeks

At the very beginning to competitive programming, barely anyone knows the coding style to be followed. Below is an example to help understand that style.



Let us consider below problem statement as an example.

Problem Statement:

Linear Search: Given an integer array and an element x, find if element is present in array or not. If element is present, then print index of its first occurrence. Else print -1.

Input:

First line contains an integer, the number of test cases ‘T’. Each test case should be an integer. Size of the array ‘N’ in the second line. In the third line, input the integer elements of the array in a single line separated by space. Element X should be inputted in the fourth line, i.e., after entering the elements of array. Repeat the above steps second line onwards for multiple test cases.

Output:

Print the output in a separate line returning the index of the element X. If the element is not present, then print -1.

Constraints:

$1 \leq T \leq 100$
 $1 \leq N \leq 100$
 $1 \leq \text{Arr}[i] \leq 100$

Example Input and Output for Your Program:

Input:

2
4
1 2 3 4
3
5
10 90 20 30 40
40

Output:

2
4

Explanation:

There are 2 test cases (Note 2 at the beginning of input)

Test Case 1: Input: arr[] = {1, 2, 3, 4},
Element to be searched = 3.
Output: 2
Explanation: 3 is present at index 2.

Test Case 2: Input: arr[] = {10, 90, 20, 30, 40},
Element to be searched = 40.
Output: 4
Explanation: 40 is present at index 4.

C

```
// A Sample C program for beginners with Competitive Programming
```

```
#include<stdio.h>

// This function returns index of element x in arr[]
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
    {
        // Return the index of the element if the element
        // is found
        if (arr[i] == x)
            return i;
    }

    //return -1 if the element is not found
    return -1;
}

int main()
{
    // Note that size of arr[] is considered 100 according to
    // the constraints mentioned in problem statement.
    int arr[100], x, t, n, i;

    // Input the number of test cases you want to run
    scanf("%d", &t);

    // One by one run for all input test cases
    while (t--)
    {
        // Input the size of the array
        scanf("%d", &n);

        // Input the array
        for (i=0; i<n; i++)
            scanf("%d",&arr[i]);

        // Input the element to be searched
        scanf("%d", &x);

        // Compute and print result
        printf("%d\n", search(arr, n, x));
    }
    return 0;
}
```

C++

```
// A Sample C++ program for beginners with Competitive Programming
#include<iostream>
using namespace std;

// This function returns index of element x in arr[]
int search(int arr[], int n, int x)
{
    for (int i = 0; i < n; i++)
    {
        // Return the index of the element if the element
        // is found
        if (arr[i] == x)
            return i;
    }

    // return -1 if the element is not found
    return -1;
}

int main()
{
    // Note that size of arr[] is considered 100 according to
    // the constraints mentioned in problem statement.
    int arr[100], x, t, n;

    // Input the number of test cases you want to run
    cin >> t;

    // One by one run for all input test cases
    while (t--)
    {
        // Input the size of the array
        cin >> n;

        // Input the array
        for (int i=0; i<n; i++)
            cin >> arr[i];

        // Input the element to be searched
        cin >> x;

        // Compute and print result
        cout << search(arr, n, x) << endl;
    }
    return 0;
}
```

Python

```
# A Sample Python program for beginners with Competitive Programming

# Returns index of x in arr if it is present,
# else returns -1
def search(arr, x):
    n = len(arr)
    for j in range(0,n):
        if (x == arr[j]):
            return j
    return -1

# Input number of test cases
t = int(raw_input())

# One by one run for all input test cases
for i in range(0,t):

    # Input the size of the array
    n = int(raw_input())

    # Input the array
    arr = map(int, raw_input().split())

    # Input the element to be searched
    x = int(raw_input())

    print(search(arr, x))

    # The element can also be searched by index method
    # But you need to handle the exception when element is not found
    # Uncomment the below line to get that working.
    # arr.index(x)
```

Java

```
// A Sample Java program for beginners with Competitive Programming
import java.util.*;
import java.lang.*;
import java.io.*;

class LinearSearch
{
    // This function returns index of element x in arr[]
    static int search(int arr[], int n, int x)
    {
        for (int i = 0; i < n; i++)
        {
            // Return the index of the element if the element
```

```
// is found
if (arr[i] == x)
    return i;
}

// return -1 if the element is not found
return -1;
}

public static void main (String[] args)
{
    // Note that size of arr[] is considered 100 according to
    // the constraints mentioned in problem statement.
    int[] arr = new int[100];

    // Input the number of test cases you want to run
    Scanner sc = new Scanner(System.in);
    int t = sc.nextInt();

    // One by one run for all input test cases
    while (t > 0)
    {
        // Input the size of the array
        int n = sc.nextInt();

        // Input the array
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();

        // Input the element to be searched
        int x = sc.nextInt();

        // Compute and print result
        System.out.println(search(arr, n, x));

        t--;
    }
}
```

Common mistakes by beginners

1. Program should not print any extra character. Writing a statement like `printf("Enter value of n")` would cause rejection on any platform.
2. Input and output format specifications must be read carefully. For example, most of the problems expect a new line after every output. So if we don't write `printf("\n")` or equivalent statement in a loop that runs for all test cases, the program would be rejected.

How to Begin Practice?

You can begin with above problem itself. Try submitting one of the above solutions [here](#).

Now you know how to write your first program in Competitive Programming Environment, you can start with [School Practice Problems for Competitive Programming](#) or [Basic Practice Problems for Competitive Programming](#).

How to Begin Study?

[Top 10 Algorithms and Data Structures for Competitive Programming](#).

Visit [here](#) to decide which category suits you more.

See [this](#) for more FAQs for beginners.

How to prepare for ACM – ICPC?

Source

<https://www.geeksforgeeks.org/how-to-begin-with-competitive-programming/>

Chapter 61

How to get rid of Java TLE problem

How to get rid of Java TLE problem - GeeksforGeeks

It happens many times that you have written correct Java code with as much optimization as needed according to the constraints. But, you get TLE .

This happens due to the time taken by Java to take input and write output using Scanner class which is slow as compared to BufferedReader and StringBuffer class. Read in detail about Scanner Class [here](#).

Have a look at some tips to get rid of this TLE issue (when your logic is correct obviously)?

Tip 1 : Avoid using [Scanner Class](#) and try to use [BufferedReader class](#).

Tip 2 : Try to use [StringBuffer class](#) in case you have to print large number of data.

Let's take a problem from [GeeksforGeeks practice](#) and solve the TLE issue:

Problem : [Segregate an Array of 0s, 1s and 2s](#)

In short, problem is, given an array of 0s, 1s and 2s. We have to segregate all the 0s in starting of array, all the 1s in mid of the array, and all the 2s in last of the array.

Examples:

Input : 1 1 2 0 0 2 1
Output : 0 0 1 1 2 2

Approach : [Segregate array of 0s, 1s and 2s](#)

Below is the implementation of above Approach :

```
// Program to segregate the
```

```
// array of 0s, 1s and 2s
import java.util.*;
import java.lang.*;
import java.io.*;
class GFG {
    public static void main(String[] args)
    {
        // Using Scanner class to take input
        Scanner sc = new Scanner(System.in);

        // Number of testcase input
        int t = sc.nextInt();

        // Iterating through all the testcases
        while (t-- > 0) {

            // Input n, i.e. size of array
            int n = sc.nextInt();

            int arr[] = new int[n];

            // Taking input of array elements
            for (int i = 0; i < n; i++)
                arr[i] = sc.nextInt();

            // Calling function to segregate
            // input array
            segregateArr(arr, n);

            // printing the modified array
            for (int i = 0; i < n; i++) {
                System.out.print(arr[i] + " ");
            }

            System.out.println();
        }
    }

    // Function to segregate 0s, 1s and 2s
    public static void segregateArr(int arr[], int n)
    {
        /*
        low : to keep left index
        high : to keep right index
        mid : to get middle element
        */
        int low = 0, high = n - 1, mid = 0;
```

```
// Iterating through the array and
// segregating elements
while (mid <= high) {

    // If element at mid is 0
    // move it to left
    if (arr[mid] == 0) {
        int temp = arr[low];
        arr[low] = arr[mid];
        arr[mid] = temp;
        low++;
        mid++;
    }

    // If element at mid is 1
    // nothing to do
    else if (arr[mid] == 1) {
        mid++;
    }

    // If element at mid is 2
    // move it to last
    else {
        int temp = arr[mid];
        arr[mid] = arr[high];
        arr[high] = temp;
        high--;
    }
}
}
```

According to our expectations, it should pass all the testcases and get accepted on [GeeksforGeeks practice](#). But, when we submit this code on GeeksforGeeks IDE, it shows TLE.

Your program took more time than expected. 
Expected Time Limit < 5.472sec
Hint : Please optimize your code and submit again.
Actual ExecTime: 0.44 Calculated TimeLimit: 5.472 [Need Help ?](#)

This signifies that we have exceeded the time limit as expected. Not an issue, let's use the tips [given above](#).

1. Use BufferedReader to take input.
2. Use StringBuffer to save and print output.

Approach : Segregate array of 0s, 1s and 2s

Below is the implementation of Java code for segregating 0s, 1s and 2s

```
// Java program to segregate
// array of 0s, 1s and 2s
import java.io.*;
import java.util.*;

class GFG {
    // Driver Code
    public static void main(String[] args) throws IOException
    {

        // Using BufferedReader class to take input
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        // taking input of number of testcase
        int t = Integer.parseInt(br.readLine());

        while (t-- > 0) {
            // n : size of array
            int n = Integer.parseInt(br.readLine());

            // Declaring array
            int arr[] = new int[n];

            // to read multiple integers line
            String line = br.readLine();
            String[] strs = line.trim().split("\\s+");

            // array elements input
            for (int i = 0; i < n; i++)
                arr[i] = Integer.parseInt(strs[i]);

            // Calling Functions to segregate Array elements
            segregateArr(arr, n);

            // Using string buffer to append each output in a string
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < n; i++)
                sb.append(arr[i] + " ");

            // finally printing the string
            System.out.println(sb);
        }
    }

    // Function to segregate 0s, 1s and 2s
```

```
public static void segregateArr(int arr[], int n)
{
    /*
    low : to keep left index
    high : to keep right index
    mid : to get middle element
    */
    int low = 0, high = n - 1, mid = 0;

    // Iterating through the array and
    // segregating elements
    while (mid <= high) {

        // If element at mid is 0
        // move it to left
        if (arr[mid] == 0) {
            int temp = arr[low];
            arr[low] = arr[mid];
            arr[mid] = temp;
            low++;
            mid++;
        }

        // If element at mid is 1
        // nothing to do
        else if (arr[mid] == 1) {
            mid++;
        }

        // If element at mid is 2
        // move it to last
        else {
            int temp = arr[mid];
            arr[mid] = arr[high];
            arr[high] = temp;
            high--;
        }
    }
}
```

Correct Answer. ✓

Execution Time: 1.26

Actual ExecTime: 1.25 Calculated Timelimit: 5.472

Next Suggested Problem: Convert time from 12 hour to 24 hour format

Great! You have leveled up.

Java TLE issue? Seems pretty much simple :). [You may try it now.](#)

Source

<https://www.geeksforgeeks.org/how-to-get-rid-of-java-tle-problem/>

Chapter 62

How to overcome Time Limit Exceed(TLE)?

How to overcome Time Limit Exceed(TLE)? - GeeksforGeeks

Many programmers always argue that the problems in Competitive Programming always end up with TLE(Time Limit Exceed). The Main problem of this error is that it will not allow you to know that your solution would reach to correct solution or not!

Why TLE comes?

- **Online Judge Restrictions:** TLE comes because the Online judge have some restriction that it will not allow to process the instruction after certain Time limit given by Problem setter usually (1 sec).
- **Server Configuration:** The exact time taken by the code depends on the speed of server,architecture of server,OS and certainly on the complexity of the algorithm. So different servers like practice, codechef, SPOJ etc may have different execution speed. By estimating the maximum value of N (N is total number of instructions of your whole code), you can roughly estimate the TLE would occur or not in 1 sec.

MAX value of N	Time complexity
10^8	$O(N)$ Border case
10^7	$O(N)$ Might be accepted
10^6	$O(N)$ Perfect
10^5	$O(N * \log N)$
10^3	$O(N ^ 2)$
10^2	$O(N ^ 3)$
10^9	$O(\log N)$ or $\text{Sqrt}(N)$

So after analyzing this chart you can roughly estimate your Time complexity and make your code within the upper bound limit.

- **Method of reading input and writing output is too slow:** Sometimes methods used by a programmer for input output may cause TLE.

OverCome Time Limit Errors

- **Change methods of Input-Output:** You must choose proper input-output functions and data structure which would help you in optimization.
 - In C++, do not use cin/cout – use scanf and printf instead.
 - In Java, do not use a Scanner – use a BufferedReader instead.
 - In Python, you could try speeding up your solutions by adding the following two lines to the start of your file:

```
import psyco
psyco.full()
```
- **Bounds of loops may be reduced :** Read the bounds in the input carefully before writing your program, and try to figure out which inputs will cause your program to run the slowest. For example if a problem tells you that $N \leq 100000$ or $N \leq 1000000$, and your program has nested loops each which go up to N , your program will never be fast enough.
- **Optimize your Algorithm:** If nothing works after all this, then you should try changing the algorithm or the approach you are using to solve your problem. Generally , the internal test cases are designed in such a way that you will be able to clear all of them only if you choose the best possible algorithm.
- **Look for Suggestions given:** Although this should be the last step but you must look at the comments given below any problem in which other programmers might have given a hint on how the problem can be solved in a better and more efficient way. And even when you overcome TLE try more exhaustive and corner test cases against your program to check the performance.

Ultimately, with experience you'll surely come to know what to do and what not to avoid TLEs. The more you code the more you get to know about how to compete TLE.

Practice Now

Source

<https://www.geeksforgeeks.org/overcome-time-limit-exceedtle/>

Chapter 63

How to prepare for ACM – ICPC?

How to prepare for ACM - ICPC? - GeeksforGeeks

ACM ICPC (Association for Computing Machinery – International Collegiate Programming Contest) is a world-wide annual multi-tiered programming contest being organized for over thirteen years. The contest is sponsored by IBM.

This article focuses on what all topics that are important for the competitive programming and should especially be studied in order to train yourself for upcoming ACM-ICPC contest.

Rules of the Contest – World final Rules for 2017 Click [here](#)

Indian Participants – Codechef conducts all the Indian Regionals. Click [here](#) to know about team formation, reimbursements etc

ICPC for Schools by **CodeChef** – This competition serves as a gateway for the school students to participate in ACM ICPC contest along with ICPC college participants held across India. It is an idea conceived by CodeChef and supported by Amrita University.

A sample ICPC Problem : A usual ICPC problem has the following features:

1. **Problem statement:** describing the problem and what output is to be generated.
2. **Input:** Make sure that you read this section with complete attention as missing out any minor detail may land you in wrong answer zone.
3. **Output:** Just like above, this one also should be read carefully.
4. **Constraints:** These can include constraints on input, time, memory, code size, etc.
5. **Time limit:** See if your algorithm can work in this range. If not, time to change it!
6. **Memory limit:** If you are fond of allocating memory for every small thing, it's a good time that you changed it.

Preparing for ACM-ICPC

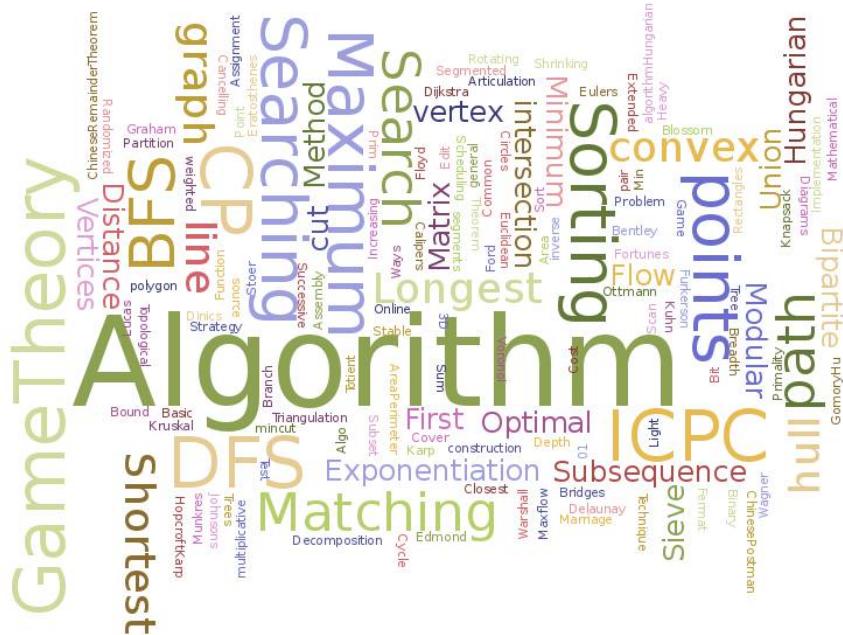
First and foremost Step: PRACTICE – Following are the resources that can be referred for practicing the ACM-ICPC alike contests and problems. For all these OJs, begin with

the problems with maximum submissions and check other solutions to check how you may improve. Do Participate in their monthly contests to remain up to the mark.

- ACM-ICPC Past Problems – [ICPC Archive](#), Practice at [Codechef](#)
 - [TopCoder](#) – Proceed by increasing problem levels gradually
 - [Codeforces](#) -List of [Problem Sets](#)
 - [Codechef](#) – Beginners can start with [Codechef Beginners](#) and proceed further
 - [SPOJ](#) – Move from easy to tough problems
 - [USACO](#) – Excellent training resource
 - [uvaOnline Judge](#) – Huge repository of problems
 - [Hackerrank](#) – Practice problems topic wise and participate in preparatory series
 - [Hackerearth](#) – Participate in preparatory series
 - [Practice](#) – Good for beginners. Has problems ranging from difficulty level School to Hard.
 - [List](#) of various Competitive Programming Contests available online all through the year

What to study?

Knowing just the basics of programming won't be fruitful for aspirants of ACM ICPC. One needs to have a thorough knowledge of advanced algorithms used as well. Following Topics list out the necessary Topics and Algorithms that one must surely know to improve and stand a chance in the actual competition.



Elementary data structures: To begin with competitive programming, one must master the Data Structures. Following is the list of most commonly used data structures:

- Array
- Stack
- Queue
- String
- Heap
- Hash
- Extensive list of Data structures

Advanced Data Structures

[Priority queues](#), union-find sets, (augmented) interval trees, (augmented) balanced BSTs and binary indexed trees

- Binary Indexed Tree or Fenwick tree
- Segment Tree ([RMQ](#), [Range Sum](#) and [Lazy Propagation](#))
- K-D tree (See [insert](#), [minimum](#) and [delete](#))
- Union Find Disjoint Set ([Cycle Detection](#) and [By Rank](#) and [Path Compression](#))
- Tries
- Interval Tree

[More Advanced Data Structures.](#)

Sorting and Searching : Concentrate to learn the basic concepts and also get familiar with all the library functions available.

- Binary Search
- Quick Sort
- Merge Sort
- Order Statistics

String manipulation : Strings make programming problems interesting and difficult too and probably that's the reason they are used extensively in such contests. Learning library functions for String actually proves very helpful (C++ : See [this](#) and [this](#), [String in Java](#)).

- KMP algorithm
- Rabin karp
- Z's algorithm
- Aho Corasick String Matching

Choosing the right Language : C++ is till date most preferred language followed by Java when it comes to programming contests but you should always choose a language you are comfortable with. Being CONFIDENT in any language is most important.

Standard Template Library : A quintessential especially for those using C++ as a language for coding

- Power up C++ STL by Topcoder – [Part 1](#), [Part 2](#)
- [C++ Magicians – STL Algorithms](#)

Dynamic Programming

- Longest Common Subsequence
- Longest Increasing Subsequence
- Edit Distance
- Minimum Partition
- Ways to Cover a Distance
- Longest Path In Matrix
- Subset Sum Problem
- Optimal Strategy for a Game
- 0-1 Knapsack Problem
- Assembly Line Scheduling
- Optimal Binary Search Tree

[All DP Algorithms](#)

BackTracking

- Rat in a Maze
- N Queen Problem
- Subset Sum
- m Coloring Problem
- Hamiltonian Cycle

[More articles on Backtracking](#)

Greedy Algorithms

- Activity Selection Problem
- Kruskal's Minimum Spanning Tree Algorithm
- Huffman Coding
- Efficient Huffman Coding for Sorted Input
- Prim's Minimum Spanning Tree Algorithm

[More articles on Greedy Algorithms](#)

Graph Algorithms : One of the most important topic which you can not ignore if preparing for ACM – ICPC.

- Breadth First Search (BFS)
- Depth First Search (DFS)
- Shortest Path from source to all vertices **Dijkstra**
- Shortest Path from every vertex to every other vertex **Floyd Warshall**
- Minimum Spanning tree **Prim**
- Minimum Spanning tree **Kruskal**
- Topological Sort
- Johnson's algorithm
- Articulation Points (or Cut Vertices) in a Graph

- Bridges in a graph

All Graph Algorithms

Basic Mathematics

Arithmetic : Programmers must know how integers and real numbers are represented internally and should be able to code high-precision numbers. Bit manipulation tricks and knowing library functions for number basic arithmetic would be very helpful.

Number theory : Knowing some of these concepts would save a lot of time and efforts while programming in the contests.

- Modular Exponentiation
- Modular multiplicative inverse
- Primality Test | Set 2 (Fermat Method)
- Euler's Totient Function
- Sieve of Eratosthenes
- Convex Hull
- Basic and Extended Euclidean algorithms
- Segmented Sieve
- Chinese remainder theorem
- Lucas Theorem

Combinatorics : Although directly might not seem to be important, Combinatorics is important to estimate asymptotic complexity of algorithms.

- Analysis of Algorithms
- Combinatorial Game Theory | Set 1 (Introduction)

Geometrical Algorithms

- Convex Hull
- Graham Scan
- Line Intersection
- Matrix Exponentiation and this
- Online construction of 3-D convex hull
- Bentley Ottmann algorithm to list all intersection points of n line segments
- Rotating Calipers Technique
- Area/Perimeter of Union of Rectangles
- Closest pair of points
- Area of Union of Circles
- Delaunay Triangulation of n points
- Voronoi Diagrams of n points using Fortune's algorithm
- Point in a polygon problem

Network Flow Algorithms

- Maxflow Ford Furkerson Algo and Edmond Karp Implementation
- Min cut
- Stable Marriage Problem
- Dinic's Algorithm for Maximum Flow and [Wiki](#)
- Minimum Cost Flow Problem
- Successive Shortest path Algorithm
- Cycle Cancelling algorithm
- Maximum weighted Bipartite Matching (Kuhn Munkres algorithm/Hungarian Method)
 - Hungarian Algorithm [Wiki](#)
 - Hungarian Algorithm for Assignment Problem
 - Maximum Bipartite Matching
- Stoer Wagner min-cut algorithm
- Maximum matching in general graph (Blossom Shrinking)
- Gomory-Hu Trees
- Chinese Postman problem (Please see [this](#) too)
- Hopcroft-Karp Algorithm for Maximum Matching

[All Articles on Geometric Algorithms](#)

More Advanced Stuff

[Bit Algorithms](#) , [Randomized Algorithms](#) , [Branch and Bound](#) , [Mathematical Algorithms](#) , [Heavy Light Decomposition](#), [A* Search](#)

Informative Articles that you may like to read

- An Awesome list for Competitive Programming – [Codeforces](#)
- What are the algorithms required to solve all C++ problems in Contests ? – [Quora](#)
- Best books and sites to prepare for ACM-ICPC – [Quora](#)
- Introduction to Programming Contest – [Stanford.edu](#)
- World Final Problems of ACM-ICPC – [icpc.kattis](#)

References:

[Programming Camp Syllabus](#)

Source

<https://www.geeksforgeeks.org/how-to-prepare-for-acm-icpc/>

Chapter 64

How to prepare for Facebook Hacker Cup?

How to prepare for Facebook Hacker Cup? - GeeksforGeeks

Facebook hacker cup is an annual algorithmic programming contest organized by Facebook. Be it students, professionals or experts it attracts numerous programming enthusiasts from all around the globe. Top contenders are eligible for the interview call from Facebook for Software Developer role.



What is the process?

Facebook Hacker cup is particularly known for its different environment used for Judging and the variety of problems. It is conducted in many rounds where the difficulty of algorithmic challenges keep on increasing.

[Registration Link](#)

What are the prizes?(May vary year to year)

- 1st Place: \$10,000 USD
- 2nd Place: \$2,000 USD
- 3rd Place: \$1,000 USD
- 4th-25th Place: \$100 USD

Organized in the month of **January** Facebook Hacker cup is conducted in 4 series round:

1. **Qualification round:** Is the easiest round in which at least 1 problem needs to be solved successfully in order to advance in the next round. This round lasts for 72 hours.
2. **Round 1 :** The selected candidates participate in round 1 which lasts for 24 hours and must gain at least certain number of points (decided accordingly every year) to qualify for the round 2. This round is fairly difficult than the qualification round.
3. **Round 2:** Candidates selected from the round 1 advance to participate in round 2 and compete in 3-hour format contest. Top 200 participants advance to Round 3 and top 500 participants are awarded with Hackercup T-shirt.
4. **Round 3:** Top 200 participants compete in this 3-hour format contest and top 25 qualify for Onsite Final. From now on the problem set gets tough.
5. **Onsite Final:** Top 25 participants from all over the globe compete for winning the title and trophy of Facebook hacker cup at its headquarters. Problems are quite challenging and are good enough to make the contestants sweat head to toe.

And the 1st person on the leader board bags the title and basks in glory.

Format and Environment

The judging format of Facebook hacker cup is quite different from other annual programming contests like ACM-ICPC or IOI.

- When the contest begins users are required to login in the website.
- After logging in they are presented with the problem set. After you think you have solved a problem and sure about its correctness, you have to **download** an input test file.
- As soon as the input test file is downloaded a timer of 6 minutes commences and in that window of time you are required to run the input test file over your code and form a test file. Within 6 minutes you need to submit both the code and the output text file.
- You can submitted more than once and only the last correct submission will be used for evaluation.
- Once the timer expires you will be unable to submit the solution for that problem again. Time penalty is the sum of submission times of a problem.

How to prepare?

Facebook Hacker cup is particularly famous for its innovative and mind tickling algorithmic challenges. Inclined more towards mathematics and combination of various concepts hacker cups tests knowledge, implementation, accuracy, speed, conceptuality and almost everything by its different rounds.

You need to be fast in order to survive further rounds and innovative to survive the initial long timed rounds.

These are the main topics which should be done thoroughly as problems are generally asked from more than 1 topics combined.

Number Theory

1. Euclidian and extended Euclidian algorithm
2. Modular Arithmetic and modular inverse
3. Prime generation (Sieve and Segmented Sieve)
4. Fermat's theorem
5. Euler Totient function
6. Miller Rabin primality test
7. Chinese remainder Theorem
8. Lucas theorem.

Greedy Algorithms

1. Activity selection problem
2. Kruskal's Algorithm
3. Prim's Algorithm.

Binary search

1. TopCoder-Binary Search
2. Binary Search
3. Ubiquitous Binary Search – Get grasp of discrete and continuous binary search.

Data Structures

1. Linked lists
2. Binary search tree
3. Binary Indexed Tree or Fenwick tree
4. Segment Tree (RMQ, Range Sum and Lazy Propagation)
5. Red-Black trees
6. Hashing

Extensive list of Data structures

Graph Algorithms

1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Shortest Path from source to all vertices **Dijkstra**
4. Shortest Path from every vertex to every other vertex **Floyd Warshall**
5. Minimum Spanning tree **Prim**
6. Minimum Spanning tree **Kruskal**
7. Topological Sort
8. Johnson's algorithm
9. Articulation Points (or Cut Vertices) in a Graph
10. Bridges in a graph

[All Graph Algorithms](#)

String Algorithms

Learning library functions for String actually proves very helpful (C++ : See [this](#) and [this](#), String in Java).

1. KMP algorithm
2. Rabin karp
3. Z's algorithm
4. Aho Corasick String Matching
5. Suffix Arrays
6. Trie
7. Finite Automata

Dynamic programming

1. Dynamic Programming – GeeksforGeeks
2. Dynamic Programming – Codechef

Dynamic programming is quite important and can be infused and asked with various other topics. Some different types of DP concepts are :

Classic DP

1. Longest Common Subsequence
2. Longest Increasing Subsequence
3. Edit Distance
4. Minimum Partition
5. Ways to Cover a Distance
6. Longest Path In Matrix
7. Subset Sum Problem
8. Optimal Strategy for a Game
9. 0-1 Knapsack Problem
10. Assembly Line Scheduling

[All DP Algorithms](#)

Computational geometry

1. Convex hull algorithms
2. Geometric Algorithms

All in all Facebook Hacker cup is a very challenging contest and a person needs gigantic amount of training and perseverance and all the standard topics need to be etched and understood.

Practice is the only way to do so!

Extra Points:

1. Practice on [Codeforces](#) (specially the [GYM](#)section) and [TopCoder Arena](#). This will truly help in basic understanding.
2. Facebook Hacker Cup's problems have a different style than Codeforces and Topcoder, probably the best comparison would be with [Google Code Jam](#) who have a similar format.
3. Go through previous Facebook – HackerCup questions and get familiar with the format of the contest.

Source

<https://www.geeksforgeeks.org/prepare-facebook-hacker-cup/>

Chapter 65

How to prepare for Google Asia Pacific University (APAC) Test ?

How to prepare for Google Asia Pacific University (APAC) Test ? - GeeksforGeeks

Google Asia Pacific University Test, also referred to as Google APAC, is perhaps the best opportunity for a student enrolled in higher education institutes in the **APAC** region to mark a place in the big shot in the IT Sector – Google.

Registration details

1. Go to the [Google APAC Website](#) and click on participate.
2. Once you login from your Google account, you will be forwarded to the registration form for Google APAC 2017.
3. Fill all the details like name, country, gender, university details, etc. You will need to know your **University ID**. Contact your college for this.
4. Fill in the details about any internship or project experience that you have, any awards/honours that you have received carefully. Be precise and to – the point.
5. Provide your **resume link**. Preferably, your resume should be of only one page length with one or two lines about every project that you have worked upon, internship experience etc.
6. Once you have filled all the details, click on I accept. Register!

Once you have completed all the registration formalities, you can take the test on your preferred date. The schedule is given below:

Schedule for Google APAC -2017 Test

Date	Time (IST)	Duration	Description
Sunday, June 26, 2016	10:30 IST	3hr	Practice Round
Sunday, July 10, 2016	10:30 IST	3hr	Round A

Sunday, August 28, 2016	10:30 IST	3hr	Round B
Sunday, September 18, 2016	10:30 IST	3hr	Round C
Sunday, October 16, 2016	10:30 IST	3hr	Round D
Sunday, November 6, 2016	10:30 IST	3hr	Round E

Note:

- There is no restriction on the number of rounds you can sit in. Only the best score will be taken into account.
- Shortlisted students may be called for interviews for internships as well as full time roles after reviewing their form entries and performance in the APAC tests.

Selection Process

Each APAC round will have 4 problems of varying difficulty. There are two kinds of input that you can test your solution for, small and large. Solving a problem for small input will give you lesser points as compared to the large one but it will be easier to solve too.

The solutions are graded according to [Google's terms](#) and finally the leaderboard is prepared. The students are then sorted according to their position in the leaderboard and their registration form entries and may be called for interviews for internship or full time job.

There are usually 4 interview rounds with questions ranging from subjects like Programming, Algorithms, Data Structures, and Operating System.

What to study??

To crack the Google APAC University test, one **must have a good grasp on algorithms and data structures**. Without working hard on data structures and algorithms , its not possible to crack the Google APAC test. These are some of the **important topics** that one should try before going for Google APAC Exam:

1. Start your preparation with the basic data structures. You should know how to implement them according to the problem given.

- [Linked list](#)
- [Binary search tree](#)
- [Array](#)
- [Stack](#)
- [Queue](#)
- You can also refer to this [extensive list of articles on Data structures](#) for more information.

2. Next comes a bit more advanced stuff.

- Graph algorithms
 - [Breadth First Search \(BFS\)](#)

- Depth First Search (DFS)
- Shortest Path from source to all vertices ****Dijkstra****
- Bridges in a graph
- Articulation Points (or Cut Vertices) in a Graph
- Johnson’s algorithm
- Topological Sort
- Minimum Spanning tree ****Kruskal****
- Minimum Spanning tree ****Prim****

- Hashing
- Heaps
- Searching and sorting
 - Binary Search
 - Quick Sort
 - Merge Sort
 - Counting Sort

String Manipulation

- KMP algorithm
- Rabin karp
- Z’s algorithm
- Aho Corasick String Matching

- Number theory concepts
 - Sieve of Eratosthenes
 - Segmented Sieve
 - Wilson’s Theorem
 - Prime Factorisation
 - Pollard’s rho algorithm

3. Once you are done with this, you can move on to more advanced algorithms like:

- Convex Hull
- Graham Scan
- Line Intersection
- Interval Tree
- Matrix Exponentiation
- Maxflow Ford Fulkerson Algo and Edmond Karp Implementation
- Min cut
- Stable Marriage Problem
- Hopcroft-Karp Algorithm for Maximum Matching

What to practice?

If you are a beginner, you can start your preparation from the **basic** and **easy** programming challenges and then can slowly move on to more difficult ones.

- [GeeksforGeeks practice](#) section provides an excellent collection of programming challenges sorted according to category and difficulty level.
- Also, make sure that you try out [previous APAC Tests](#).

Source

<https://www.geeksforgeeks.org/how-to-prepare-for-google-asia-pacific-university-test/>

Chapter 66

How to read Competitive Programming Questions?

How to read Competitive Programming Questions? - GeeksforGeeks

Competitive Programming is considered as a sport in the field of computer science. The culture of this sport is growing day by day and a lot of people are also considering this as a career choice. So, to help the participants with improving their efficiency in this sport, in this post, we have tried to cover all possible things, needed to keep in mind while reading a

competitive programming question.



Business vector created by Freepik

The most important thing that matters in competitive programming is:

How quickly you are able to solve a problem?

Time is considered as a prime factor in tie-breaking for ranking participants with the same scores in programming contests.

Again, to save time one must understand the problem as fast as possible which depends on *how you are reading a programming question?*.

Now to cope-up with this, the basic strategy that a lot of *sports programmer* follows is they read problems differently based on their difficulty level. Note that most of the coding contests have problems based on the below difficulty levels:

- **Cake-walk:** Problems in this category are considered very easy and based on simple implementation. This problem also matters a lot in tie-breaking as most of the programmers submit their solutions to the cake-walk problem in the first 5 mins of the contest. The basic strategy to solve this problem as fast as possible is to save time by avoiding reading the complete question. Try to look at the test cases at first and the Input and Output format which are explained in the problem statement. Most of the times, this comes to be more than sufficient to solve a cake-walk problem.
- **Easy:** Problems in this category are easy but not as easy as cake-walk problems. These problems are not based on any Data Structures and Algorithm or any tougher concepts. These problems aim to test the analytical skills of the programmer. Most of the times, these problems based on any mathematical concept or string implementations etc. While reading this problem try to avoid the story part of the problem statement if it has any. Reading this problem once will be sufficient to solve it but you must be careful enough at the same time to not miss any concept.
- **Medium and Hard:** We had put both medium and hard in the same section as one must be very careful while reading these questions as they contain a lot corner cases and include in-depth concepts of data-structures and algorithms like [segment trees](#), [Binary Indexed Trees](#), [Dynamic Programming](#), [Persistent Data Structures](#) etc.

Some of the steps which budding programmers can follow to solve these problems are:

1. **Read the question at least twice:** The first step of turning a problem into a solution is to understand the problem itself. If you are able to understand the question at first read, we would suggest you to read it once again, there can be a hidden test case set by the problem setter. If you are not able to understand the question, read it as many times you want until and unless you figure out what the problems need to be solved. Once you get the logic, its all about coding it in a language of your choice.
2. **Mark out the minute details:** You can write down the names of the variables you need to declare, some formulas mentioned in the question so that you don't forget in the long run. You must set aside the constraint of each variable on a piece of paper or remember it since this may lead to an error in the future.

Marking out the details will help you avoid silly errors and focus on the logic and save your precious time in coding competitions. If you get a long question and you have to write a long code for it you might not remember the details of each and every variable so you can refer to the list you have set aside for reference and continue coding at the same pace.

3. **Refer to the question while coding:** You can open 2 tabs, one for the coding and one for the question part. Some coding sites allow you to do that and are beneficial to see the exact details while coding.
4. **Read the question again before submitting:** It may feel boring to read the same question again and again but if you are missing an important detail this might make you see it. Once you miss it, the game is over. So, you must go through the question once again before submitting your solution.

Most of you must have participated in several coding competitions and had your fair of success. While coding sometimes, everyone come across errors due to little things. Missing a variable to declare, leaving a formula, not reading the question entirely, ignoring the constraint on a variable and the list goes on. The above details may come in use to avoid these basic mistakes.

Many coding contests also **penalize participants for submitting a wrong solution**. So, it is recommended to be confident about your solution before you submit it.

Source

<https://www.geeksforgeeks.org/how-to-read-competitive-programming-questions/>

Chapter 67

How to read content of GeeksforGeeks in an organized way?

How to read content of GeeksforGeeks in an organized way? - GeeksforGeeks

We have been trying to organize content of geeksforgeeks. We added below pages for this purpose.

Algorithms and Data Structures:

1. [Algorithms](#)
2. [Data Structures](#)

Languages:

3. [C](#)
4. [C++](#)
5. [Java](#)
6. [Python](#)
7. [SQL](#)
8. [PHP](#)
9. [JavaScript](#)
10. [Program output](#)

Interview:

11. [Company preparation](#)
12. [Top Topics](#)
13. [Practice company questions](#)

- 14. Interview experiences
- 15. Experienced Interview
- 16. Internship Interview
- 17. Competitive programming
- 18. Software Design patterns
- 19. Multiple choice Quizzes

Students:

- 20. Computer Science Projects
- 21. Placements course
- 22. Puzzles
- 23. Geek on the Top
- 24. GBlog
- 25. School Programming

GATE, ISRO, and UGC NET exam preparation:

- 26. GATE CS Notes
- 27. GATE CS Corner
- 28. Last minute notes
- 29. ISRO CS previous year solved papers
- 30. Previous years UGC NET CS solved papers
- 31. UGC NET CS Notes Paper-II
- 32. UGC NET CS Notes Paper-III

Computer Science subjects:

- 33. Mathematics
- 34. Operating Systems
- 35. Computer Network
- 36. DBMS
- 37. Computer Organization
- 38. Microprocessor
- 39. Theory Of Computation
- 40. Compiler Design
- 41. Digital Logic
- 42. Software Engineering

Other subjects:

- 43. Web Technology
- 44. Advanced Topics
- 45. Machine Learning
- 46. Computer Graphics
- 47. Difference between

Courses and Test exams:

- 48. [Geek Classes](#)
- 49. [Sudo Placement](#)
- 50.

The above pages contain all related article at one place, but do not provide proper navigation through articles. GeeksforGeeks is quite limited in functionalities which motivated the team to design [GeeksforGeeks Courses](#). The idea of <https://practice.geeksforgeeks.org/courses>, is to allow users to see/navigate content in an organized manner, to enroll into different courses and track their progress. We will be adding more features over time.

We are also planning to cover more areas/topics which are not currently covered on GeeksforGeeks. Stay tuned!

Source

<https://www.geeksforgeeks.org/how-to-read-content-of-geeksforgeeks-in-an-organized-way/>

Chapter 68

Inclusion Exclusion principle and programming applications

Inclusion Exclusion principle and programming applications - GeeksforGeeks

Sum Rule – If a task can be done in one of \mathcal{W}_1 ways or one of \mathcal{W}_2 ways, where none of the set of \mathcal{W}_1 ways is the same as any of the set of \mathcal{W}_2 ways, then there are $\mathcal{W}_1 + \mathcal{W}_2$ ways to do the task.

The sum-rule mentioned above states that if there are multiple sets of ways of doing a task, there shouldn't be any way that is common between two sets of ways because if there is, it would be counted twice and the enumeration would be wrong.

The principle of **inclusion-exclusion** says that in order to count only unique ways of doing a task, we must add the number of ways to do it in one way and the number of ways to do it in another and then subtract the number of ways to do the task that are common to both sets of ways.

The principle of inclusion-exclusion is also known as the **subtraction principle**. For two sets of ways A_1 and A_2 , the enumeration would like-

Below are some examples to explain the application of inclusion-exclusion principle:

- **Example 1:** How many binary strings of length 8 either start with a '1' bit or end with two bits '00'?

Solution: If the string starts with one, there are 7 characters left which can be filled in $2^7 = 128$ ways.

If the string ends with '00' then 6 characters can be filled in $2^6 = 64$ ways.

Now if we add the above sets of ways and conclude that it is the final answer, then it would be wrong. This is because there are strings with start with '1' and end with

'00' both, and since they satisfy both criteria they are counted twice.

So we need to subtract such strings to get a correct count.

Strings that start with '1' and end with '00' have five characters that can be filled in

$$2^5 = 32 \text{ ways.}$$

So by the inclusion-exclusion principle we get-

$$\text{Total strings} = 128 + 64 - 32 = 160$$

- **Example 2:** How many numbers between 1 and 1000, including both, are divisible by 3 or 4?

$$\text{Solution: Number of numbers divisible by 3} = \left| A_1 \right| = \left| \{ \text{Numbers between 1 and 1000 which are divisible by 3} \} \right| = 333$$

$$\text{Number of numbers divisible by 4} = \left| A_2 \right| = \left| \{ \text{Numbers between 1 and 1000 which are divisible by 4} \} \right| = 250$$

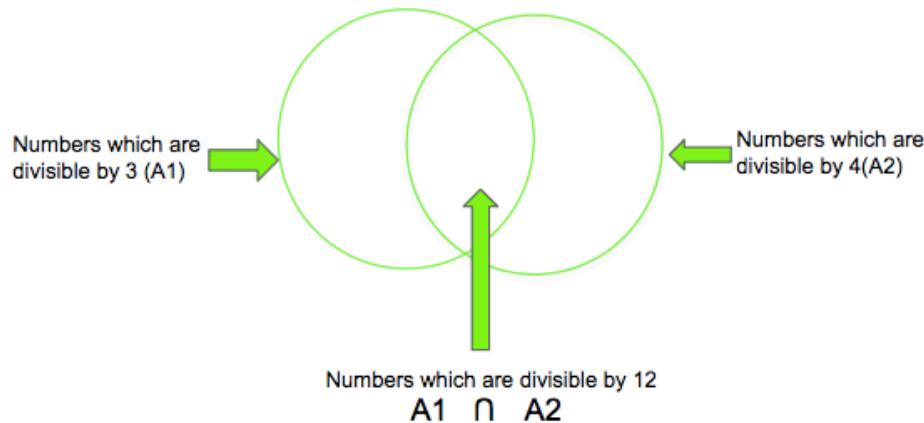
$$\text{Number of numbers divisible by 3 and 4} = \left| A_1 \cap A_2 \right| = \left| \{ \text{Numbers between 1 and 1000 which are divisible by 12} \} \right| = 83$$

$$\text{Therefore, number of numbers divisible by 3 or 4} = \left| A_1 \cup A_2 \right| = 333 + 250 - 83 = 500$$

Implementation

Problem 1: How many numbers between 1 and 1000, including both, are divisible by 3 or 4?

The Approach will be the one discussed above, we add the number of numbers that are divisible by 3 and 4 and subtract the numbers which are divisible by 12.



C++

```
// CPP program to count the
```

```
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
#include <bits/stdc++.h>
using namespace std;

// function to count the divisors
int countDivisors(int N, int a, int b)
{
    // Counts of numbers
    // divisible by a and b
    int count1 = N / a;
    int count2 = N / b;

    // inclusion-exclusion
    // principle applied
    int count3 = (N / (a * b));

    return count1 + count2 - count3;
}

// Driver Code
int main()
{
    int N = 1000, a = 3, b = 4;
    cout << countDivisors(N, a, b);
    return 0;
}
```

Java

```
// Java program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
import java.io.*;

class GFG
{

    // function to count the divisors
    public static int countDivisors(int N,
                                    int a,
                                    int b)
    {
        // Counts of numbers
        // divisible by a and b
        int count1 = N / a;
```

```
int count2 = N / b;

// inclusion-exclusion
// principle applied
int count3 = (N / (a * b));

return count1 + count2 - count3;
}

// Driver Code
public static void main (String[] args)
{
    int N = 1000, a = 3, b = 4;
    System.out.println (countDivisors(N, a, b));
}
}

// This code is contributed by m_kit
```

C#

```
// C# program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4
using System;

class GFG
{

    // function to count
    // the divisors
    public static int countDivisors(int N,
                                    int a,
                                    int b)
    {
        // Counts of numbers
        // divisible by a and b
        int count1 = N / a;
        int count2 = N / b;

        // inclusion-exclusion
        // principle applied
        int count3 = (N / (a * b));

        return count1 + count2 - count3;
    }
}
```

```
// Driver Code
static public void Main ()
{
    int N = 1000, a = 3, b = 4;
    Console.WriteLine(countDivisors(N, a, b));
}
}

// This code is contributed by aj_36
```

PHP

```
<?php
// PHP program to count the
// number of numbers between
// 1 and 1000, including both,
// that are divisible by 3 or 4

// function to count the divisors
function countDivisors($N, $a, $b)
{
    // Counts of numbers
    // divisible by a and b
    $count1 = $N / $a;
    $count2 = $N / $b;

    // inclusion-exclusion
    // principle applied
    $count3 = ($N / ($a * $b));

    return $count1 + $count2 - $count3;
}

// Driver Code
$N = 1000; $a = 3; $b = 4;
echo countDivisors($N, $a, $b);

// This code is contributed by aj_36
?>
```

Output :

500

.

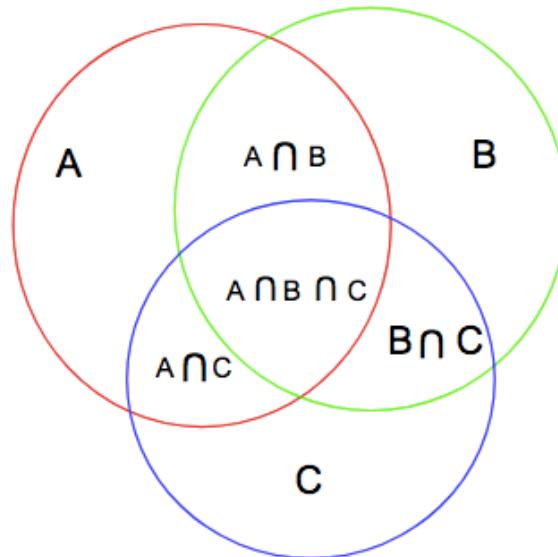
Problem 2: Given N prime numbers and a number M, find out how many numbers from 1 to M are divisible by any of the N given prime numbers.

Examples :

Input: N numbers = {2, 3, 5, 7} M = 100
Output: 78

Input: N numbers = {2, 5, 7, 11} M = 200
Output: 69

The approach for this problem will be to generate all the possible combinations of numbers using N prime numbers using power set in 2^N . For each of the given prime numbers P_i among N, it has M/P_i multiples. Suppose M=10, and we are given with 3 prime numbers(2, 3, 5), then the total count of multiples when we do $10/2 + 10/3 + 10/5$ is 11. Since we are counting 6 and 10 twice, the count of multiples in range 1-M comes 11. Using inclusion-exclusion principle, we can get the correct number of multiples. The inclusion-exclusion principle for three terms can be described as:



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Similarly, for every N numbers, we can easily find the total number of multiples in range 1 to M by applying the formula for an intersection of N numbers. The numbers that are formed by multiplication of an odd number of prime numbers will be added and

the numbers formed by multiplication of even numbers will thus be subtracted to get the total number of multiples in the range 1 to M.

Using power set we can easily get all the combination of numbers formed by the given prime numbers. To know if the number is formed by multiplication of odd or even numbers, simply count the number of set bits in all the possible combinations (**1-1<<N**).

Using power sets and adding the numbers created by combinations of odd and even prime numbers we get 123 and 45 respectively. Using **inclusion-exclusion principle** we get the number of numbers in range 1-M that is divided by any one of N prime numbers is **(odd combinations-even combinations) = (123-45) = 78**.

Below is the implementation of the above idea:

C++

```
// CPP program to count the
// number of numbers in range
// 1-M that are divisible by
// given N prime numbers
#include <bits/stdc++.h>
using namespace std;

// function to count the number
// of numbers in range 1-M that
// are divisible by given N
// prime numbers
int count(int a[], int m, int n)
{
    int odd = 0, even = 0;
    int counter, i, j, p = 1;
    int pow_set_size = (1 << n);

    // Run from counter 000..0 to 111..1
    for (counter = 1; counter < pow_set_size;
         counter++)
    {
        p = 1;
        for (j = 0; j < n; j++)
        {

            // Check if jth bit in the
            // counter is set If set
            // then pront jth element from set
            if (counter & (1 << j))
            {
                p *= a[j];
            }
        }
    }
}
```

```
// if set bits is odd, then add to
// the number of multiples
if (_builtin_popcount(counter) & 1)
    odd += (100 / p);
else
    even += 100 / p;
}

return odd - even;
}
// Driver Code
int main()
{
    int a[] = { 2, 3, 5, 7 };
    int m = 100;
    int n = sizeof(a) / sizeof(a[0]);
    cout << count(a, m, n);
    return 0;
}
```

Output:

78

Time Complexity : $O(2^N \cdot N)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/inclusion-exclusion-principle-and-programming-applications/>

Chapter 69

Input/Output from external file in C/C++, Java and Python for Competitive Programming

Input/Output from external file in C/C++, Java and Python for Competitive Programming
- GeeksforGeeks

In Competitive Programming, most of the time we need to enter input for checking our code manually. But it would become cumbersome if we have a questions like Graphs, strings, or bulky entry of input data because it will definitely time out or the chances of typing incorrect data would be increased.

We can easily get rid of problem by simply saving the test cases to the specified file at the desired location according to our easiness. These are useful in competitions like **Facebook Hackercup**, **Google Codejam**. These competitions do not provide the online judge like environment rather than they asked you to upload input and output files.

For Input/Output, we basically use these two type of file access modes i.e.,

- “**r**”: It means read, as it opens the file for input operation but the file must exist at specified location.
- “**w**”: It means write, as it creates an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.

C/C++

In C/C++ we can use freopen for standard input and output. The syntax of this function as:-

```
FILE * freopen(const char * filename, const char * mode, FILE * stream );
```

- **filename:** It refers to name of the file that we want to open.
- **mode:** Discussed above.
- **stream:** Pointer to a FILE object that identifies the stream to be reopened.

We can also use Symbolic constant **ONLINE_JUDGE** for debugging purpose. As most of the online judges add this flag at the end of the command line before executing our code. For example in C++11, it would be compiled as

```
-std = c++0x -O2 -static -s -lm -DONLINE_JUDGE
```

So we can take advantage of that by using [C preprocessor directive](#).

```
// Below is C/C++ code for input/output
#include<stdio.h>

int main()
{
#ifndef ONLINE_JUDGE

    // For getting input from input.txt file
    freopen("input.txt", "r", stdin);

    // Printing the Output to output.txt file
    freopen("output.txt", "w", stdout);

#endif
    return 0;
}
```

JAVA

In Java, we can use [BufferedReader class](#) for the fast Input and [PrintWriter class](#) for formatted representation to the output along with FileReader and FileWriter class.

- **FileReader(String filename):** This constructor creates a new FileReader, and instructs the parser to read file from that directory. The file must exist in that specified location.
- **FileWriter(String fileName):** This constructor creates a FileWriter object, to the specified location.

```
// Java program For handling Input/Output
import java.io.*;
class Main
{
    public static void main(String[] args) throws IOException
    {
        // BufferedReader Class for Fast buffer Input
```

```
BufferedReader br = new BufferedReader(  
        new FileReader("input.txt"));  
  
// PrintWriter class prints formatted representations  
// of objects to a text-output stream.  
PrintWriter pw=new PrintWriter(new  
        BufferedWriter(new FileWriter("output.txt")));  
  
// Your code goes Here  
  
pw.flush();  
}  
}  
// Thanks to Saurabh Kumar Prajapati for providing this java Code
```

PYTHON

In python we first import the module sys(system), after that We will use open() function which returns the file object, that are commonly used with two arguments: open(filename, mode).

- The first argument is a string containing the filename.
- The second argument is another string (mode) containing a few characters describing the way in which the file will be used.

```
# Below is Pythone code for input/output  
  
import sys  
# For getting input from input.txt file  
sys.stdin = open('input.txt', 'r')  
  
# Printing the Output to output.txt file  
sys.stdout = open('output.txt', 'w')
```

Source

<https://www.geeksforgeeks.org/inputoutput-external-file-cc-java-python-competitive-programming/>

Chapter 70

Input/Output from external file in C/C++, Java and Python for Competitive Programming | Set 2

Input/Output from external file in C/C++, Java and Python for Competitive Programming | Set 2 - GeeksforGeeks

Prerequisites : [Input/Output from external file in C/C++, Java and Python for Competitive Programming](#)

In above post, we saw a way to have standard input/output from external file using file handling. In this post, we will see a very easy way to do this. Here we will compile and run our code from terminal or cmd using input and output streams.

Windows Environment

For **Windows**, it's the case of redirection operators to redirect command input and output streams from the default locations to different locations.

1. **Redirecting command input (<)** : To redirect command input from the keyboard to a file or device, use the < operator.
2. **Redirecting command output (>)** : To redirect command output from the Command Prompt window to a file or device, use the > operator.

Run the code

```
a.exe < input_file > output_file  
  
// A simple C++ code (test_code.cpp) which takes
```

```
// one string as input and prints the same string
// to output
#include <iostream>
using namespace std;

int main()
{
    string S;
    cin >> S;
    cout << S;

    return 0;
}
```

Input from input.txt:

GeeksForGeeks

Compile & run:

```
g++ test_code.cpp
a.exe < input.txt > output.txt
```

Output in output.txt:

GeeksForGeeks

Linux Environment

I/O Redirection in linux: Input and output in the Linux environment are distributed across three streams. These streams are:

1. **standard input (stdin):** The standard input stream typically carries data from a user to a program. Programs that expect standard input usually receive input from a device, such as a keyboard, but using < we can redirect input from the text file.
2. **standard output (stdout):** Standard output writes the data that is generated by a program. When the standard output stream is not redirected, it will output text to the terminal. By using > we can redirect output to a text file.
3. **standard error (stderr):** Standard error writes the errors generated by a program that has failed at some point in its execution. Like standard output, the default destination for this stream is the terminal display.

Linux includes redirection commands for each stream.

Overwrite : Commands with a single bracket overwrite the destination's existing contents.

- > – standard output

- < – standard input
- 2> – standard error

Append : Commands with a double bracket do not overwrite the destination's existing contents.

- >> – standard output
- << – standard input
- 2>> – standard error

Here, we will use Overwrite's command because we don't need to append the output i.e we only want the output of one single code.

Compile C++ code

```
g++ file_name.cpp
```

Run the code

```
./a.out < input_file > output_file
```

We can similarly give standard input/output from text files for C or Java by first compiling the code and while running the code we give input/output files in given format. For languages like python which don't require compilation, we do the following

```
python test_code.py < input.txt > output.txt
```

Related Article: [Python Input Methods for Competitive Programming](#)

References :

- Microsoft
- Digital Ocean

Source

<https://www.geeksforgeeks.org/inputoutput-external-file-cc-java-python-competitive-programming-set-2/>

Chapter 71

Introduction to Programming Languages

Introduction to Programming Languages - GeeksforGeeks

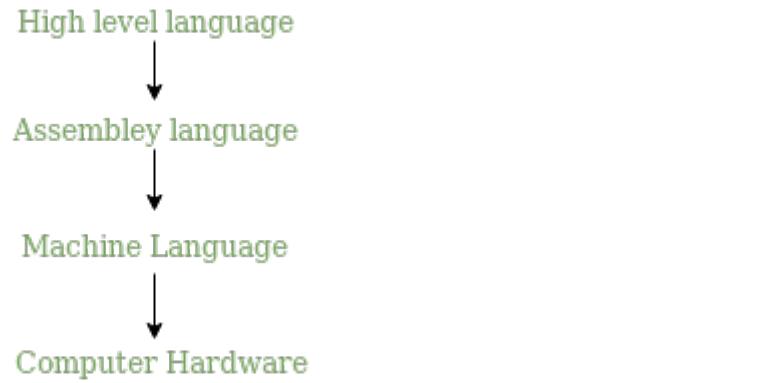
A computer is a computational device which is used to process the data under the control of a computer program. Program is a sequence of instruction along with data. While executing the program, raw data is processed into a desired output format. These computer programs are written in a programming language which are high level languages. High level languages are nearly human languages which are more complex than the computer understandable language which are called machine language, or low level language.

Basic example of a computer program written in C programming language:

```
#include<stdio.h>
int main(void)
{
    printf("C is a programming language");
    return 0;
}
```

Between high-level language and machine language there are assembly language also called symbolic machine code. Assembly language are particularly computer architecture specific. Utility program (**Assembler**) is used to convert assembly code into executable machine code. High Level Programming Language are portable but require Interpretation or compiling to convert it into a machine language which is computer understood.

Hierarchy of Computer language –



There have been many programming language some of them are listed below:

C	Python	C++
C#	R	Ruby
COBOL	ADA	Java
Fortran	BASIC	Altair BASIC
True BASIC	Visual BASIC	GW BASIC
QBASIC	PureBASIC	PASCAL
Turbo Pascal	GO	ALGOL
LISP	SCALA	Swift
Rust	Prolog	Reia
Racket	Scheme	Shimula
Perl	PHP	Java Script
CoffeeScript	VisualFoxPro	Babel
Logo	Lua	Smalltalk
Matlab	F	F#
Dart	Datalog	dbase
Haskell	dylan	Julia
ksh	metro	Mumps
Nim	OCaml	pick
TCL	D	CPL
Curry	ActionScript	Erlang
Clojure	DarkBASIC	Assembly

Most Popular Programming Languages –

- C
- Python
- C++
- Java
- SCALA
- C#
- R
- Ruby

- Go
- Swift
- JavaScript

Characteristics of a programming Language –

- A programming language must be simple, easy to learn and use, have good readability and human recognizable.
- Abstraction is a must-have Characteristics for a programming language in which ability to define the complex structure and then its degree of usability comes.
- A portable programming language is always preferred.
- Programming language's efficiency must be high so that it can be easily converted into a machine code and executed consumes little space in memory.
- A programming language should be well structured and documented so that it is suitable for application development.
- Necessary tools for development, debugging, testing, maintenance of a program must be provided by a programming language.
- A programming language should provide single environment known as Integrated Development Environment(IDE).
- A programming language must be consistent in terms of syntax and semantics.

Source

<https://www.geeksforgeeks.org/introduction-to-programming-languages/>

Chapter 72

Java tricks for competitive programming (for Java 8)

Java tricks for competitive programming (for Java 8) - GeeksforGeeks

Although practice is the only way that ensures increased performance in programming contests but having some tricks up your sleeve ensures an upper edge and fast debugging.

1) Checking if the number is even or odd without using the % operator:

Although this trick is not much better than using % operator but is sometimes efficient (with large numbers). Use & operator:

```
System.out.println((a & 1) == 0 ? "EVEN" : "ODD");
```

Example:

num = 5

Binary: “101 & 1” will be 001, so true

num = 4

Binary: “100 & 1” will be 000, so false.

2) Fast Multiplication or Division by 2

Multiplying by 2 means shifting all the bits to left and dividing by 2 means shifting to the right.

Example : 2 (Binary 10): shifting left 4 (Binary 100) and right 1 (Binary 1)

```
n = n << 1; // Multiply n with 2  
n = n >> 1; // Divide n by 2
```

3) Swapping of 2 numbers using XOR:

This method is fast and doesn't require the use of 3rd variable.

```
// A quick way to swap a and b  
a ^= b;  
b ^= a;  
a ^= b;
```

4) Faster I/O:

Refer [here](#) for Fast I/O in java

5) For String manipulations:

Use [StringBuffer](#) for string manipulations, as [String](#) in java is immutable. Refer [here](#).

6) Calculating the most significant digit: To calculate the most significant digit of any number log can be directly used to calculate it.

```
Suppose the number is N then  
Let double K = Math.log10(N);  
now K = K - Math.floor(K);  
int X = (int) Math.pow(10, K);  
X will be the most significant digit.
```

7) Calculating the number of digits directly: To calculate number of digits in a number, instead of looping we can efficiently use log :

```
No. of digits in N = Math.floor(Math.log10(N)) + 1;
```

8) Inbuilt GCD Method: Java has inbuilt GCD method in [BigInteger class](#). It returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val). Returns 0 if this==0 && val==0.

Syntax :
public BigInteger gcd(BigInteger val)
Parameters :
val - value with which the GCD is to be computed.
Returns :
GCD(abs(this), abs(val))

```
// Java program to demonstrate how  
// to use gcd method of BigInteger class  
  
import java.math.BigInteger;  
  
class Test  
{  
    public static int gcd(int a, int b)
```

```
{  
    BigInteger b1 = BigInteger.valueOf(a);  
    BigInteger b2 = BigInteger.valueOf(b);  
    BigInteger gcd = b1.gcd(b2);  
    return gcd.intValue();  
}  
  
public static long gcd(long a, long b)  
{  
    BigInteger b1 = BigInteger.valueOf(a);  
    BigInteger b2 = BigInteger.valueOf(b);  
    BigInteger gcd = b1.gcd(b2);  
    return gcd.longValue();  
}  
  
// Driver method  
public static void main(String[] args)  
{  
    System.out.println(gcd(3, 5));  
    System.out.println(gcd(1000000000L, 600000000L));  
}  
}
```

Output:

```
1  
200000000
```

9) checking for a prime number: Java has inbuilt `isProbablePrime()` method in [BigInteger class](#). It returns true if this BigInteger is probably prime(with some certainty), false if it's definitely composite.

```
BigInteger.valueOf(1235).isProbablePrime(1)
```

10) Efficient trick to know if a number is a power of 2 The normal technique of division the complexity comes out to be $O(\log N)$, but it can be solved using $O(v)$ where v are the number of digits of number in binary form.

```
/* Method to check if x is power of 2*/  
static boolean isPowerOfTwo (int x)  
{  
    /* First x in the below expression is  
       for the case when x is 0 */  
    return x!=0 && ((x&(x-1)) == 0);  
}
```

11) Sorting Algorithm:

`Arrays.sort()` used to sort elements of a array.
`Collections.sort()` used to sort elements of a collection.

For primitives, `Arrays.sort()` uses dual pivot quicksort algorithms.

12) Searching Algorithm:

`Arrays.binarySearch()`([SET 1](#) | [SET2](#)) used to apply binary search on an sorted array.
`Collections.binarySearch()` used to apply binary search on a collection based on comparators.

13) Copy Algorithm:

`Arrays.copyOf()` and `copyOfRange()` copy the specified array.
`Collections.copy()` copies specified collection.

14) Rotation and Frequency We can use `Collections.rotate()` to rotate a collection or an array by a specified distance. You can also use `Collections.frequency()` method to get frequency of specified element in a collection or an array.

15) Most data structures are already implemented in the Collections Framework.
For example : Stack, LinkedList, HashSet, HashMaps, Heaps etc.

16) Use Wrapper class functions for getting radix conversions of a number Some-time you require radix conversion of a number. For this you can use wrapper classes.

```
// Java program to demonstrate use of wrapper
// classes for radix conversion

class Test
{
    // Driver method
    public static void main(String[] args)
    {
        int a = 525;
        long b = 12456545645L;

        String binaryA = Integer.toString(a, 2);
        System.out.println("Binary representation" +
                           " of A : " + binaryA);
        String binaryB = Long.toString(b, 2);
        System.out.println("Binary representation" +
                           " of B : " + binaryB);
        String octalA = Integer.toString(a, 8);
        System.out.println("Octal representation" +
                           " of A : " + octalA);
        String octalB = Long.toString(b, 8);
        System.out.println("Octal representation" +
                           " of B : " + octalB);
    }
}
```

Output:

```
Binary representation of A : 1000001101
Binary representation of B : 101110011001110111100110101101101
Octal representation of A : 1015
Octal representation of B : 134635746555
```

17) **NullPointerException(Why ?)** Refer [here](#) and [here](#) to avoid it.

Source

<https://www.geeksforgeeks.org/java-tricks-competitive-programming-java-8/>

Chapter 73

Knowing the complexity in competitive programming

Knowing the complexity in competitive programming - GeeksforGeeks

Prerequisite: [Time Complexity Analysis](#)

Generally, while doing competitive programming problems on various sites, the most difficult task faced is writing the code under desired complexity otherwise the program will get a TLE (**Time Limit Exceeded**). A naive solution is almost never accepted. So how to know, what complexity is acceptable?

The answer to this question is directly related to the number of operations that are allowed to perform within a second. Most of the sites these days allow **10⁸ operations per second**, only a few sites still allow 10⁷ operations. After figuring out the number of operations that can be performed, search for the right complexity by looking at the constraints given in the problem.

Example:

Given an array A[] and a number x, check for a pair in A[] with the sum as x.
where N is:

- 1) $1 \leq N \leq 103$
- 2) $1 \leq N \leq 105$
- 3) $1 \leq N \leq 108$

For Case 1

A naive solution that is using two for-loops works as it gives us a complexity of $O(N^2)$, which even in the worst case will perform 10^6 operations which are well under 10^8 . Ofcourse $O(N)$ and $O(NlogN)$ is also acceptable in this case.

For Case 2

We have to think of a better solution than $O(N^2)$, as in worst case, it will perform 10^{10}

operations as N is 10^5 . So complexity acceptable for this case is either $O(N \log N)$ which is approximately 10^6 ($10^5 * \sim 10$) operations well under 10^8 or $O(N)$.

For Case 3

Even $O(N \log N)$ gives us TLE as it performs $\sim 10^9$ operations which are over 10^8 . So the only solution which is acceptable is $O(N)$ which in worst case will perform 10^8 operations.

The code for the given problem can be found on : <https://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whether-or-not-there-exist-t>

Source

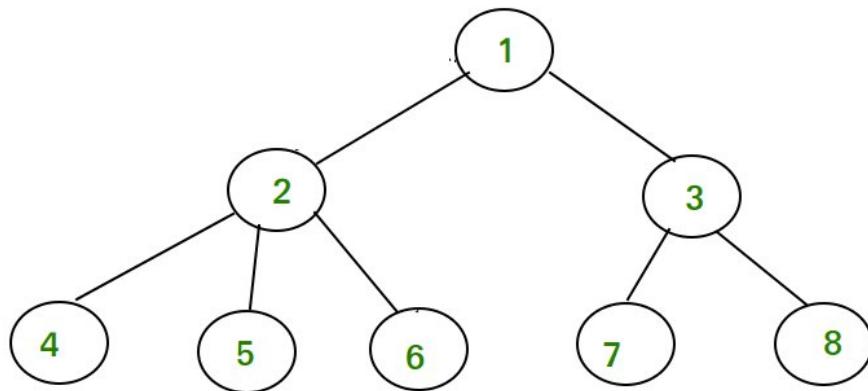
<https://www.geeksforgeeks.org/knowing-the-complexity-in-competitive-programming/>

Chapter 74

LCA for general or n-ary trees (Sparse Matrix DP approach < O(nlogn), O(logn)>)

LCA for general or n-ary trees (Sparse Matrix DP approach < O(nlogn), O(logn)>) - Geeks-forGeeks

In previous posts, we have discussed how to calculate the Lowest Common Ancestor (LCA) for a binary tree and a binary search tree ([this](#), [this](#) and [this](#)). Now let's look at a method that can calculate LCA for any tree (not only for binary tree). We use Dynamic Programming with Sparse Matrix Approach in our method. This method is very handy and fast when you need to answer multiple queries of LCA for a tree.



$$\begin{aligned} \text{LCA}(4, 6) &= 2 \\ \text{LCA}(5, 7) &= 1 \end{aligned}$$

Pre-requisites : -

- 1) [DFS](#)
- 2) Basic DP knowledge ([This](#) and [this](#))
- 3) [Range Minimum Query \(Square Root Decomposition and Sparse Table\)](#)

Naive Approach:- $O(n)$

The naive approach for this general tree LCA calculation will be the same as the naive approach for the LCA calculation of Binary Tree (this naive approach is already well described [here](#).

The C++ implementation for the naive approach is given below :-

```
/* Program to find LCA of n1 and n2 using one DFS on
   the Tree */
#include <iostream>
#include <vector>
using namespace std;

// Maximum number of nodes is 100000 and nodes are
// numbered from 1 to 100000
#define MAXN 100001

vector < int > tree[MAXN];
int path[3][MAXN]; // storing root to node path

// storing the path from root to node
void dfs(int cur, int prev, int pathNumber, int ptr,
         int node, bool &flag)
{
    for (int i=0; i<tree[cur].size(); i++)
    {
        if (tree[cur][i] != prev and !flag)
        {
            // pushing current node into the path
            path[pathNumber][ptr] = tree[cur][i];
            if (tree[cur][i] == node)
            {
                // node found
                flag = true;

                // terminating the path
                path[pathNumber][ptr+1] = -1;
                return;
            }
            dfs(tree[cur][i], cur, pathNumber, ptr+1,
                 node, flag);
        }
    }
}
```

```
}  
  
// This Function compares the path from root to 'a' & root  
// to 'b' and returns LCA of a and b. Time Complexity : O(n)  
int LCA(int a, int b)  
{  
    // trivial case  
    if (a == b)  
        return a;  
  
    // setting root to be first element in path  
    path[1][0] = path[2][0] = 1;  
  
    // calculating path from root to a  
    bool flag = false;  
    dfs(1, 0, 1, 1, a, flag);  
  
    // calculating path from root to b  
    flag = false;  
    dfs(1, 0, 2, 1, b, flag);  
  
    // runs till path 1 & path 2 matches  
    int i = 0;  
    while (path[1][i] == path[2][i])  
        i++;  
  
    // returns the last matching node in the paths  
    return path[1][i-1];  
}  
  
void addEdge(int a,int b)  
{  
    tree[a].push_back(b);  
    tree[b].push_back(a);  
}  
  
// Driver code  
int main()  
{  
    int n = 8; // Number of nodes  
    addEdge(1,2);  
    addEdge(1,3);  
    addEdge(2,4);  
    addEdge(2,5);  
    addEdge(2,6);  
    addEdge(3,7);  
    addEdge(3,8);  
}
```

```

cout << "LCA(4, 7) = " << LCA(4,7) << endl;
cout << "LCA(4, 6) = " << LCA(4,6) << endl;
return 0;
}

```

Output:

```

LCA(4, 7) = 1
LCA(4, 6) = 2

```

Sparse Matrix Approach ($O(n\log n)$) pre-processing, $O(\log n)$ – query

Pre-computation :- Here we store the 2^i th parent for every node, where $0 \leq i < \text{LEVEL}$, here “LEVEL” is a constant integer that tells the maximum number of 2^i th ancestor possible.

Therefore, we assume the worst case to see what is the value of the constant LEVEL. In our worst case every node in our tree will have at max 1 parent and 1 child or we can say it simply reduces to a linked list.

So, in this case $\text{LEVEL} = \text{ceil}(\log(\text{number of nodes}))$.

We also pre-compute the height for each node using one dfs in $O(n)$ time.

```

int n          // number of nodes
int parent[MAXN][LEVEL] // all initialized to -1

parent[node][0] : contains the  $2^0$ th(first)
parent of all the nodes pre-computed using DFS

// Sparse matrix Approach
for node -> 1 to n :
    for i-> 1 to LEVEL :
        if ( parent[node][i-1] != -1 ) :
            parent[node][i] =
                parent[ parent[node][i-1] ][i-1]

```

Now , as we see the above dynamic programming code runs two nested loop that runs over their complete range respectively.

Hence, it can be easily be inferred that its asymptotic Time Complexity is $O(\text{number of nodes} * \text{LEVEL}) \sim O(n * \text{LEVEL}) \sim O(n \log n)$.

Return LCA(u,v) :-

1) First Step is to bring both the nodes at the same height. As we have already pre-computed the heights for each node. We first calculate the difference in the heights of u and v (let's say $v \geq u$). Now we need the node 'v' to jump h nodes above. This can be easily done in $O(\log h)$ time (where h is the difference in the heights of u and v) as we have already stored

the 2^i parent for each node. This process is exactly same as calculating $x \wedge y$ in $O(\log y)$ time. (See the code for better understanding).

2) Now both u and v nodes are at same height. Therefore now once again we will use 2^i jumping strategy to reach the first Common Parent of u and v.

Pseudo-code:

```
For i-> LEVEL to 0 :
    If parent[u][i] != parent[v][i] :
        u = parent[u][i]
        v = parent[v][i]
```

C++ implementation of the above algorithm is given below:

```
// Sparse Matrix DP approach to find LCA of two nodes
#include <bits/stdc++.h>
using namespace std;
#define MAXN 100000
#define level 18

vector <int> tree[MAXN];
int depth[MAXN];
int parent[MAXN][level];

// pre-compute the depth for each node and their
// first parent( $2^0$ th parent)
// time complexity :  $O(n)$ 
void dfs(int cur, int prev)
{
    depth[cur] = depth[prev] + 1;
    parent[cur][0] = prev;
    for (int i=0; i<tree[cur].size(); i++)
    {
        if (tree[cur][i] != prev)
            dfs(tree[cur][i], cur);
    }
}

// Dynamic Programming Sparse Matrix Approach
// populating  $2^i$  parent for each node
// Time complexity :  $O(n\log n)$ 
void precomputeSparseMatrix(int n)
{
    for (int i=1; i<level; i++)
    {
        for (int node = 1; node <= n; node++)
        {
            if (parent[node][i-1] != -1)
```

```
        parent[node][i] =
            parent[parent[node][i-1]][i-1];
    }
}

// Returning the LCA of u and v
// Time complexity : O(log n)
int lca(int u, int v)
{
    if (depth[v] < depth[u])
        swap(u, v);

    int diff = depth[v] - depth[u];

    // Step 1 of the pseudocode
    for (int i=0; i<level; i++)
        if ((diff>>i)&1)
            v = parent[v][i];

    // now depth[u] == depth[v]
    if (u == v)
        return u;

    // Step 2 of the pseudocode
    for (int i=level-1; i>=0; i--)
        if (parent[u][i] != parent[v][i])
    {
        u = parent[u][i];
        v = parent[v][i];
    }

    return parent[u][0];
}

void addEdge(int u,int v)
{
    tree[u].push_back(v);
    tree[v].push_back(u);
}

// driver function
int main()
{
    memset(parent,-1,sizeof(parent));
    int n = 8;
    addEdge(1,2);
    addEdge(1,3);
```

```
addEdge(2,4);
addEdge(2,5);
addEdge(2,6);
addEdge(3,7);
addEdge(3,8);
depth[0] = 0;

// running dfs and precalculating depth
// of each node.
dfs(1,0);

// Precomputing the  $2^i$  th ancestor for every node
precomputeSparseMatrix(n);

// calling the LCA function
cout << "LCA(4, 7) = " << lca(4,7) << endl;
cout << "LCA(4, 6) = " << lca(4,6) << endl;
return 0;
}
```

Output:

```
LCA(4,7) = 1
LCA(4,6) = 2
```

Time Complexity: The time complexity for answering a single LCA query will be $O(\log n)$ but the overall time complexity is dominated by precalculation of the 2^i th ($0 \leq i \leq \text{level}$) ancestors for each node. Hence, the overall asymptotic Time Complexity will be $O(n * \log n)$ and Space Complexity will be $O(n \log n)$, for storing the data about the ancestors of each node.

Source

<https://www.geeksforgeeks.org/lca-for-general-or-n-ary-trees-sparse-matrix-dp-approach-onlogn-ologn/>

Chapter 75

Largest connected component on a grid

Largest connected component on a grid - GeeksforGeeks

Given a grid with different colors in a different cell, each color represented by a different number. The task is to find out the largest connected component on the grid. Largest component grid refers to a maximum set of cells such that you can move from any cell to any other cell in this set by only moving between side-adjacent cells from the set.

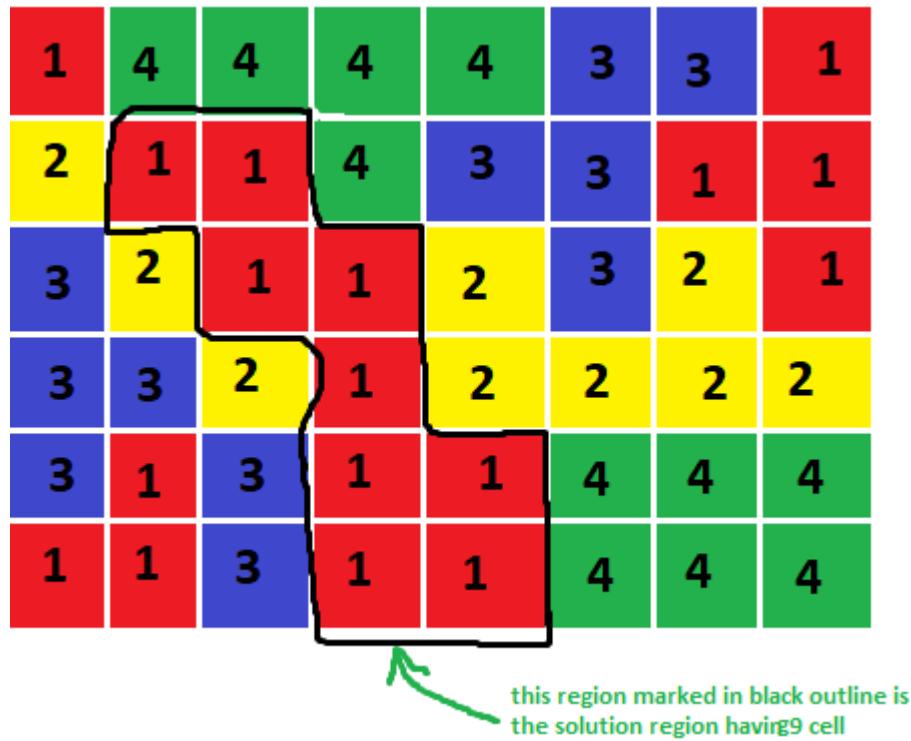
Examples:

Input :

1	4	4	4	4	3	3	1
2	1	1	4	3	3	1	1
3	2	1	1	2	3	2	1
3	3	2	1	2	2	2	2
3	1	3	1	1	4	4	4
1	1	3	1	1	4	4	4

Grid of different colors

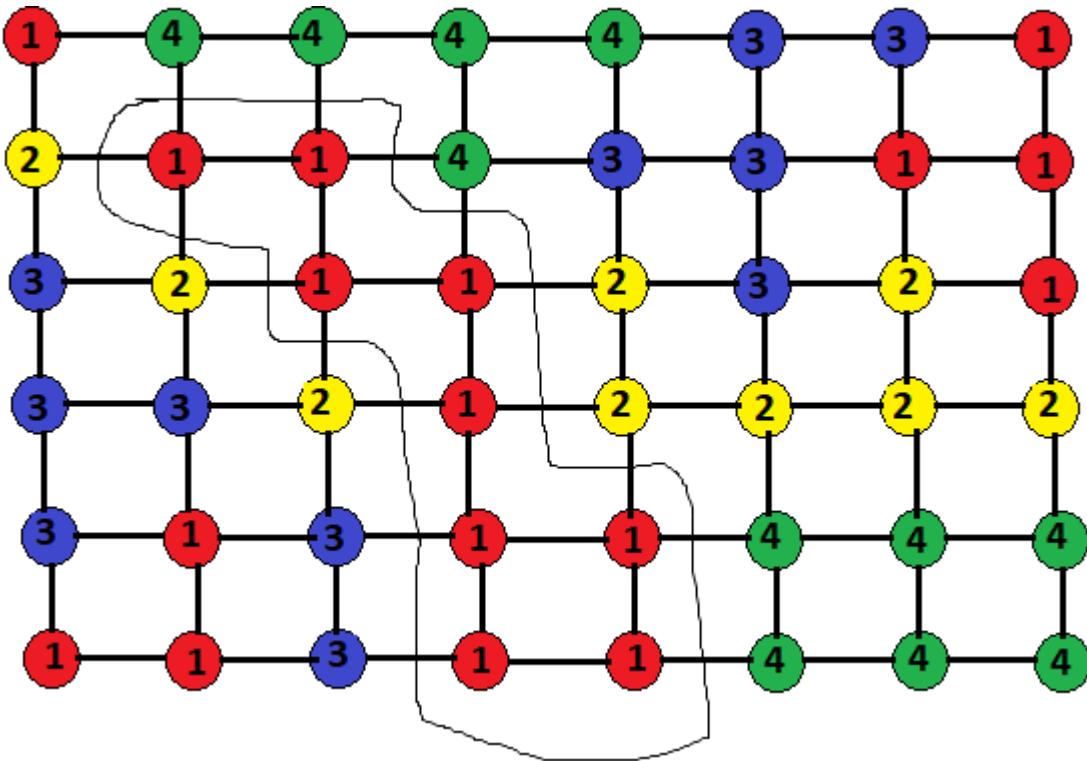
Output : 9



Largest connected component of grid

Approach :

The approach is to visualize the given grid as a graph with each cell representing a separate node of the graph and each node connected to four other nodes which are to immediately up, down, left, and right of that grid. Now doing a **BFS** search for every node of the graph, find *all the nodes connected to the current node with same color value as the current node*. Here is the graph for above example :



Graph representation of grid

At every cell (i, j) , a BFS can be done. The possible moves from a cell will be either to **right, left, top or bottom**. Move to only those cells which are in range and are of the same color. If the same nodes have been visited previously, then the largest component value of the grid is stored in **result[][]** array. Using memoization, reduce the number of BFS on any cell. **visited[][]** array is used to mark if the cell has been visited previously and count stores the count of the connected component when a BFS is done for every cell. Store the maximum of the count and print the resultant grid using **result[][]** array.

Below is the illustration of the above approach:

C++

```
// CPP program to print the largest
// connected component in a grid
#include <bits/stdc++.h>
using namespace std;

const int n = 6;
const int m = 8;
```

```

// stores information about which cell
// are already visited in a particular BFS
int visited[n][m];

// result stores the final result grid
int result[n][m];

// stores the count of cells in the largest
// connected component
int COUNT;

// Function checks if a cell is valid i.e it
// is inside the grid and equal to the key
bool is_valid(int x, int y, int key, int input[n][m])
{
    if (x < n && y < m && x >= 0 && y >= 0) {
        if (visited[x][y] == false && input[x][y] == key)
            return true;
        else
            return false;
    }
    else
        return false;
}

// BFS to find all cells in
// connection with key = input[i][j]
void BFS(int x, int y, int i, int j, int input[n][m])
{
    // terminating case for BFS
    if (x != y)
        return;

    visited[i][j] = 1;
    COUNT++;

    // x_move and y_move arrays
    // are the possible movements
    // in x or y direction
    int x_move[] = { 0, 0, 1, -1 };
    int y_move[] = { 1, -1, 0, 0 };

    // checks all four points connected with input[i][j]
    for (int u = 0; u < 4; u++)
        if (is_valid(i + y_move[u], j + x_move[u], x, input))
            BFS(x, y, i + y_move[u], j + x_move[u], input);
}

```

```

// called every time before a BFS
// so that visited array is reset to zero
void reset_visited()
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            visited[i][j] = 0;
}

// If a larger connected component
// is found this function is called
// to store information about that component.
void reset_result(int key, int input[n][m])
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (visited[i][j] && input[i][j] == key)
                result[i][j] = visited[i][j];
            else
                result[i][j] = 0;
        }
    }
}

// function to print the result
void print_result(int res)
{
    cout << "The largest connected "
        << "component of the grid is :" << res << "\n";

    // prints the largest component
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (result[i][j])
                cout << result[i][j] << " ";
            else
                cout << ". ";
        }
        cout << "\n";
    }
}

// function to calculate the largest connected
// component
void computeLargestConnectedGrid(int input[n][m])
{
    int current_max = INT_MIN;

```

```

for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        reset_visited();
        COUNT = 0;

        // checking cell to the right
        if (j + 1 < m)
            BFS(input[i][j], input[i][j + 1], i, j, input);

        // updating result
        if (COUNT >= current_max) {
            current_max = COUNT;
            reset_result(input[i][j], input);
        }
        reset_visited();
        COUNT = 0;

        // checking cell downwards
        if (i + 1 < n)
            BFS(input[i][j], input[i + 1][j], i, j, input);

        // updating result
        if (COUNT >= current_max) {
            current_max = COUNT;
            reset_result(input[i][j], input);
        }
    }
}
print_result(current_max);
}

// Drivers Code
int main()
{
    int input[n][m] = { { 1, 4, 4, 4, 4, 3, 3, 1 },
                        { 2, 1, 1, 4, 3, 3, 1, 1 },
                        { 3, 2, 1, 1, 2, 3, 2, 1 },
                        { 3, 3, 2, 1, 2, 2, 2, 2 },
                        { 3, 1, 3, 1, 1, 4, 4, 4 },
                        { 1, 1, 3, 1, 1, 4, 4, 4 } };

    // function to compute the largest
    // connected component in the grid
    computeLargestConnectedGrid(input);
    return 0;
}

```

Java

```
// Java program to print the largest
// connected component in a grid
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
static final int n = 6;
static final int m = 8;

// stores information about which cell
// are already visited in a particular BFS
static final int visited[][] = new int [n] [m];

// result stores the final result grid
static final int result[][] = new int [n] [m];

// stores the count of
// cells in the largest
// connected component
static int COUNT;

// Function checks if a cell
// is valid i.e it is inside
// the grid and equal to the key
static boolean is_valid(int x, int y,
                      int key,
                      int input[][])
{
    if (x < n && y < m &&
        x >= 0 && y >= 0)
    {
        if (visited[x] [y] == 0 &&
            input[x] [y] == key)
            return true;
        else
            return false;
    }
    else
        return false;
}

// BFS to find all cells in
// connection with key = input[i] [j]
static void BFS(int x, int y, int i,
                int j, int input[][])
{
```

```

// terminating case for BFS
if (x != y)
    return;

visited[i][j] = 1;
COUNT++;

// x_move and y_move arrays
// are the possible movements
// in x or y direction
int x_move[] = { 0, 0, 1, -1 };
int y_move[] = { 1, -1, 0, 0 };

// checks all four points
// connected with input[i][j]
for (int u = 0; u < 4; u++)
    if ((is_valid(i + y_move[u],
                  j + x_move[u], x, input)) == true)
        BFS(x, y, i + y_move[u],
             j + x_move[u], input);
}

// called every time before
// a BFS so that visited
// array is reset to zero
static void reset_visited()
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            visited[i][j] = 0;
}

// If a larger connected component
// is found this function is
// called to store information
// about that component.
static void reset_result(int key,
                         int input[][])
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (visited[i][j] == 1 &&
                input[i][j] == key)
                result[i][j] = visited[i][j];
            else
                result[i][j] = 0;
        }
    }
}

```

```

        }
    }
}

// function to print the result
static void print_result(int res)
{
    System.out.println ("The largest connected " +
                        "component of the grid is :" +
                        res );

    // prints the largest component
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (result[i][j] != 0)
                System.out.print(result[i][j] + " ");
            else
                System.out.print(". ");
        }
        System.out.println();
    }
}

// function to calculate the
// largest connected component
static void computeLargestConnectedGrid(int input[][])
{
    int current_max = Integer.MIN_VALUE;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            reset_visited();
            COUNT = 0;

            // checking cell to the right
            if (j + 1 < m)
                BFS(input[i][j], input[i][j + 1],
                     i, j, input);

            // updating result
            if (COUNT >= current_max)
            {
                current_max = COUNT;
                reset_result(input[i][j], input);
            }
        }
    }
}

```

```

        }
        reset_visited();
        COUNT = 0;

        // checking cell downwards
        if (i + 1 < n)
            BFS(input[i][j],
                input[i + 1][j], i, j, input);

        // updating result
        if (COUNT >= current_max)
        {
            current_max = COUNT;
            reset_result(input[i][j], input);
        }
    }
    print_result(current_max);
}
// Driver Code
public static void main(String args[])
{
    int input[][] = {{1, 4, 4, 4, 4, 3, 3, 1},
                    {2, 1, 1, 4, 3, 3, 1, 1},
                    {3, 2, 1, 1, 2, 3, 2, 1},
                    {3, 3, 2, 1, 2, 2, 2, 2},
                    {3, 1, 3, 1, 1, 4, 4, 4},
                    {1, 1, 3, 1, 1, 4, 4, 4}};

    // function to compute the largest
    // connected component in the grid
    computeLargestConnectedGrid(input);
}
}

// This code is contributed by Subhadeep

```

Output:

The largest connected component of the grid is :9

```

. . . . .
. 1 1 . . .
. . 1 1 . . .
. . . 1 . . .
. . . 1 1 . .
. . . 1 1 . .

```

Improved By : [tufan_gupta2000](#)

Source

<https://www.geeksforgeeks.org/largest-connected-component-on-a-grid/>

Chapter 76

Largest number with one swap allowed

Largest number with one swap allowed - GeeksforGeeks

Given a positive integer, find the largest number that could be generated by swapping only two digits at most once.

Examples:

Input: 2736
Output : 7236
Explanation:
If we swap the number 2 and the number
7 then the generated number would be
the largest number.

Input : 432
Output : 432
Explanation:
Here, no swap is required. The given
number is already largest.

Approach 1 (Trying every pair):

We convert the number to a string. For each digit of the number, we will swap with positions (i, j) and store it as temp check if the temp is larger than number. If yes then swap back to restore the original number.

C++

```
// code to find largest number with
// given conditions.
```

```
#include <bits/stdc++.h>
using namespace std;

// function to find the largest number
// with given conditions.
int largestNum(int num)
{
    // converting the number to the string
    string num_in_str = to_string(num);
    string temp = num_in_str;

    // swamping each digit
    for (int i = 0; i < num_in_str.size(); i++) {
        for (int j = i + 1; j < num_in_str.size(); j++) {

            // Swapping and checking for the larger
            swap(num_in_str[i], num_in_str[j]);
            if (stoi(num_in_str) > stoi(temp))
                temp = num_in_str;

            // Reverting the changes
            swap(num_in_str[i], num_in_str[j]);
        }
    }

    return stoi(temp);
}

// driver function
int main()
{
    int num = 432;
    cout << largestNum(num) << endl;
    num = 2736;
    cout << largestNum(num) << endl;
    num = 4596;
    cout << largestNum(num) << endl;
    return 0;
}
```

Java

```
// Java code to find largest number
// with given conditions.
public class LargestNumber{

    static String swap(String str, int i, int j)
```

```
{  
    char ch[] = str.toCharArray();  
    char temp = ch[i];  
    ch[i] = ch[j];  
    ch[j] = temp;  
    String c=String.valueOf(ch);  
    return c;  
}  
  
// function to find the largest number  
// with given conditions.  
static int largestNum(int num)  
{  
    // converting the number to the string  
    String num_in_str = ""+num;  
    String temp = num_in_str;  
  
    // swamping each digit  
    for (int i = 0; i < num_in_str.length(); i++) {  
        for (int j = i + 1; j < num_in_str.length();  
             j++) {  
  
            // Swapping and checking for the larger  
            num_in_str= swap(num_in_str,i,j);  
            if (temp.compareTo(num_in_str)<0)  
                temp = num_in_str;  
  
            // Reverting the changes  
            num_in_str=swap(num_in_str,i,j);  
        }  
    }  
  
    return Integer.parseInt(temp);  
}  
  
// Driver code  
public static void main(String[] s)  
{  
    int num = 423;  
    System.out.println(largestNum(num));  
    num = 2736;  
    System.out.println(largestNum(num));  
    num = 4596;  
    System.out.println(largestNum(num));  
}  
}  
  
// This code is contributed by Prerena Saini
```

Python3

```
# python3 code to find the largest
# number with given conditions.

# function to find the largest number
def largestNum(num):

    # converting the number to the list
    num_to_str = list(str(num))
    temp = num_to_str[:]

    # swaping each digit and check for
    # the largest number
    for i in range(len(num_to_str)):
        for j in range(i + 1, len(num_to_str)):

            // Swapping current pair
            num_to_str[i], num_to_str[j] = num_to_str[j], num_to_str[i]
            if num_to_str > temp:
                temp = num_to_str[:]

            # Reverting above change before next iteration
            num_to_str[i], num_to_str[j] = num_to_str[j], num_to_str[i]

    # returning the largest number.
    return int("".join(temp))

# main function
def main():
    A = int(432)
    print(largestNum(A))
    A = int(2736)
    print(largestNum(A))
    A = int(4596)
    print(largestNum(A))

# calling main function
if __name__=="__main__":
    main()
```

Output:

```
432
7236
9546
```

Approach 2 (Efficient) :

We will scan the number from backward direction. In the scan, if the ith digit is the largest by far, store it and its index or if the current digit is smaller than the largest digit recorded by far, this digit and the largest digit are the best suitable for swap.

Below is the c++ code for the above approach.

```
// code to find largest number with
// given conditions.
#include <bits/stdc++.h>
using namespace std;

// function to find the largest number
// with given conditions.
int largestNum(int num)
{
    int max_digit = -1;
    int max_digit_indx = -1;
    int l_indx = -1;
    int r_indx = -1;

    // converting the number to string
    string num_in_str = to_string(num);
    for (int i = num_in_str.size() - 1; i >= 0; i--) {

        // current digit is the largest by far
        if (num_in_str[i] > max_digit) {
            max_digit = num_in_str[i];
            max_digit_indx = i;
            continue;
        }

        // best digit for swap if there is no more
        // such situation on the left side
        if (num_in_str[i] < max_digit) {
            l_indx = i;
            r_indx = max_digit_indx;
        }
    }

    // check for is nummber already in order
    if (l_indx == -1)
        return num;

    swap(num_in_str[l_indx], num_in_str[r_indx]);

    return stoi(num_in_str);
```

```
}

// driver function
int main()
{
    int num = 789;
    cout << largestNum(num) << endl;
    num = 49658;
    cout << largestNum(num) << endl;
    num = 2135;
    cout << largestNum(num) << endl;
    return 0;
}
```

Output:

```
987
94658
5132
```

Source

<https://www.geeksforgeeks.org/largest-number-with-one-swap-allowed/>

Chapter 77

Longest substring having K distinct vowels

Longest substring having K distinct vowels - GeeksforGeeks

Given a string s we have to find the length of the longest substring of s which contain exactly K distinct vowels.

Note : Consider uppercase and lowercase characters as two different characters.

Examples:

Input : s = “tHeracEBetwEEEntheTwo”, k = 1

Output : 14

Explanation : Longest substring with only 1 vowel is “cEBetwEEEntheTw” and its length is 14.

Input : s = “artyebui”, k = 2

Output : 6

Explanation : Longest substring with only 2 vowel is “rtyebu”

Brute-Force Approach : For each substring, we check for the criteria for K distinct vowel and check the length. Finally, the largest length will be the result.

Efficient Approach : Here we maintain the count of vowels occurring in the substring. Till K is not zero, we count the distinct vowel occurring in the substring. As K becomes negative, we start deleting the first vowel of the substring we have found till that time, so that it may be possible that new substring(larger length) is possible afterward. As we delete the vowel we decrease its count so that in new substring may contain that vowel occurring in the later part of the string. And as K is 0 we get the length of the substring.

Below is the implementation of the above approach

C++

```
// CPP program to find the longest substring
// with k distinct vowels.
#include <bits/stdc++.h>
using namespace std;

#define MAX 128

// Function to check whether a character is
// vowel or not
bool isVowel(char x)
{
    return (x == 'a' || x == 'e' || x == 'i' ||
            x == 'o' || x == 'u' || x == 'A' ||
            x == 'E' || x == 'I' || x == 'O' ||
            x == 'U');
}

int KDistinctVowel(char s[], int k)
{
    // length of string
    int n = strlen(s);

    // array for count of characters
    int c[MAX];
    memset(c, 0, sizeof(c));

    // Initialize result to be
    // negative
    int result = -1;

    for (int i = 0, j = -1; i < n; ++i) {

        int x = s[i];

        // If letter is vowel then we
        // increment its count value
        // and decrease the k value so
        // that if we again encounter the
        // same vowel character then we
        // don't consider it for our result
        if (isVowel(x)) {
            if (++c[x] == 1) {

                // Decrementing the K value
                --k;
            }
        }
    }
}
```

```
// Till k is 0 above if condition run
// after that this while loop condition
// also become active. Here what we have
// done actually is that, if K is less
// than 0 then we eliminate the first
// vowel we have encountered till that
// time . Now K is incremented and k
// becomes 0. So, now we calculate the
// length of substring from the present
// index to the deleted index of vowel
// which result in our results.
while (k < 0) {

    x = s[++j];
    if (isVowel(x)) {

        // decresing the count
        // so that it may appear
        // in another substring
        // appearing after this
        // present substring
        if (--c[x] == 0) {

            // incrementing the K value
            ++k;
        }
    }
}

// Checking the maximum value
// of the result by comparing
// the length of substring
// whenever K value is 0 means
// K distinct vowel is present
// in substring
if (k == 0)
    result = max(result, i - j);
}
return result;
}

// Driver code
int main(void)
{
    char s[] = "tHeracEBetwEEEntheTwo";
    int k = 1;
    cout << KDistinctVowel(s, k);
    return 0;
}
```

}

C#

```
// C# program to find the longest substring
// with k distinct vowels.
using System;

class GFG {

    static int MAX = 128;

    // Function to check whether a character is
    // vowel or not
    static bool isVowel(char x)
    {
        return (x == 'a' || x == 'e' || x == 'i' ||
                x == 'o' || x == 'u' || x == 'A' ||
                x == 'E' || x == 'I' || x == 'O' ||
                x == 'U');
    }

    static int KDDistinctVowel(string s, int k)
    {
        // length of string
        int n = s.Length;

        // array for count of characters
        int []c = new int[MAX];
        Array.Clear(c, 0, c.Length);

        // Initialize result to be
        // negative
        int result = -1;

        for (int i = 0, j = -1; i < n; ++i) {

            char x = s[i];

            // If letter is vowel then we
            // increment its count value
            // and decrease the k value so
            // that if we again encounter the
            // same vowel character then we
            // don't consider it for our result
            if (isVowel(x)) {
                if (++c[x] == 1) {
```

```
// Decrementing the K value
--k;
}

// Till k is 0 above if condition run
// after that this while loop condition
// also become active. Here what we have
// done actually is that, if K is less
// than 0 then we eliminate the first
// vowel we have encountered till that
// time . Now K is incremented and k
// becomes 0. So, now we calculate the
// length of substring from the present
// index to the deleted index of vowel
// which result in our results.
while (k < 0) {

    x = s[++j];
    if (isVowel(x)) {

        // decresing the count
        // so that it may appear
        // in another substring
        // appearing after this
        // present substring
        if (--c[x] == 0) {

            // incrementing the K value
            ++k;
        }
    }
}

// Checking the maximum value
// of the result by comparing
// the length of substring
// whenever K value is 0 means
// K distinct vowel is present
// in substring
if (k == 0) {
    result = Math.Max(result, i - j);
}
}

return result;
}

// Driver code
```

```
static void Main()
{
    string s = "tHeracEBetwEEEntheTwo";
    int k = 1;
    Console.WriteLine(KDistinctVowel(s, k));
}

// This code is contributed Manish Shaw
// (manishshaw1)
```

Output:

7

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/longest-substring-having-k-distinct-vowels/>

Chapter 78

Making elements of two arrays same with minimum increment/decrement

Making elements of two arrays same with minimum increment/decrement - GeeksforGeeks

Given two arrays of same size, we need to convert the first array into another with minimum operations. In an operation, we can either increment or decrement an element by one. Note that orders of appearance of elements do not need to be same.

Here to convert one number into another we can add or subtract 1 from it.

Examples :

Input : a = { 3, 1, 1 }, b = { 1, 2, 2 }

Output : 2

Explanation : Here we can increase any 1 into 2 by 1 operation and 3 to 2 in one decrement operation. So a[] becomes {2, 2, 1} which is a permutation of b[].

Input : a = { 3, 1, 1 }, b = { 1, 1, 2 }

Output : 1

Algorithm :

1. First sort both the arrays.
2. After sorting we will run a loop in which we compare the first and second array elements and calculate the required operation needed to make first array equal to second.

Below is implementation of the above approach

C++

```
// CPP program to find minimum increment/decrement  
// operations to make array elements same.
```

```
#include <bits/stdc++.h>
using namespace std;

int MinOperation(int a[], int b[], int n)
{
    // sorting both arrays in
    // ascending order
    sort(a, a + n);
    sort(b, b + n);

    // variable to store the
    // final result
    int result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for (int i = 0; i < n; ++i) {
        if (a[i] > b[i])
            result = result + abs(a[i] - b[i]);

        else if (a[i] < b[i])
            result = result + abs(a[i] - b[i]);
    }

    return result;
}

// Driver code
int main()
{
    int a[] = { 3, 1, 1 };
    int b[] = { 1, 2, 2 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << MinOperation(a, b, n);
    return 0;
}
```

Java

```
// Java program to find minimum
// increment/decrement operations
// to make array elements same.
```

```
import java.util.Arrays;
import java.io.*;

class GFG
{
    static int MinOperation(int a[],
                           int b[],
                           int n)
    {
        // sorting both arrays
        // in ascending order
        Arrays.sort(a);
        Arrays.sort(b);

        // variable to store
        // the final result
        int result = 0;

        // After sorting both arrays
        // Now each array is in non-
        // decreasing order. Thus,
        // we will now compare each
        // element of the array and
        // do the increment or decrement
        // operation depending upon the
        // value of array b[].
        for (int i = 0; i < n; ++i)
        {
            if (a[i] > b[i])
                result = result +
                    Math.abs(a[i] - b[i]);

            else if (a[i] < b[i])
                result = result +
                    Math.abs(a[i] - b[i]);
        }

        return result;
    }

    // Driver code
    public static void main (String[] args)
    {
        int a[] = {3, 1, 1};
        int b[] = {1, 2, 2};
        int n = a.length;
        System.out.println(MinOperation(a, b, n));
    }
}
```

```
}
```

```
}
```

```
// This code is contributed
```

```
// by akt_mit
```

PHP

```
<?php
// PHP program to find minimum
// increment/decrement operations
// to make array elements same.
function MinOperation($a, $b, $n)
{
    // sorting both arrays in
    // ascending order

    sort($a);
    sort($b);

    // variable to store
    // the final result
    $result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for ($i = 0; $i < $n; ++$i)
    {
        if ($a[$i] > $b[$i])
            $result = $result + abs($a[$i] -
                $b[$i]);

        else if ($a[$i] < $b[$i])
            $result = $result + abs($a[$i] -
                $b[$i]);
    }

    return $result;
}

// Driver code
$a = array ( 3, 1, 1 );
```

```
$b = array ( 1, 2, 2 );
$n = sizeof($a);
echo MinOperation($a, $b, $n);

// This code is contributed by ajit
?>
```

Output :

2

Time Complexity : $O(n \log n)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/making-elements-of-two-arrays-same-with-minimum-incrementdecrement/>

Chapter 79

Matrix Exponentiation

Matrix Exponentiation - GeeksforGeeks

This is one of the [most used techniques in competitive programming](#). Let us first consider below simple question.

What is the minimum time complexity to find n'th Fibonacci Number?

We can find n'th Fibonacci Number in O(Log n) time using Matrix Exponentiation. Refer method 4 of [this](#) for details. In this post, a general implementation of Matrix Exponentiation is discussed.

For solving the matrix exponentiation we are assuming a linear recurrence equation like below:

$$F(n) = a*F(n-1) + b*F(n-2) + c*F(n-3) \quad \text{for } n \geq 3 \\ \dots \dots \dots \text{Equation (1)}$$

where a, b and c are constants.

For this recurrence relation it depends on three previous values.

Now we will try to represent Equation (1) in terms of matrix.

$$\begin{bmatrix} \text{First Matrix} \end{bmatrix} = \begin{bmatrix} \text{Second matrix} \end{bmatrix} * \begin{bmatrix} \text{Third Matrix} \end{bmatrix} \\ \begin{vmatrix} F(n) \end{vmatrix} = \text{Matrix 'C'} * \begin{vmatrix} F(n-1) \end{vmatrix} \\ \begin{vmatrix} F(n-1) \end{vmatrix} \quad | \quad F(n-2) \\ \begin{vmatrix} F(n-2) \end{vmatrix} \quad | \quad F(n-3) \quad |$$

Dimension of the first matrix is 3 x 1 .

Dimension of third matrix is also 3 x 1.

So the dimension of the second matrix must be 3 x 3
[For multiplication rule to be satisfied.]

Now we need to fill the Matrix 'C'.

So according to our equation.

$$F(n) = a*F(n-1) + b*F(n-2) + c*F(n-3)$$

$$F(n-1) = F(n-1)$$

$$F(n-2) = F(n-2)$$

$$C = \begin{bmatrix} a & b & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Now the relation between matrix becomes :

[First Matrix]	[Second matrix]	[Third Matrix]
$ F(n) $	$= a \ b \ c * 1 \ 0 \ 0 $	$ F(n-1) $
$ F(n-1) $		$ F(n-2) $
$ F(n-2) $		$ F(n-3) $

Lets assume the initial values for this case :-

$$F(0) = 0$$

$$F(1) = 1$$

$$F(2) = 1$$

So, we need to get $F(n)$ in terms of these values.

So, for $n = 3$ Equation (1) changes to

$ F(3) $	$= a \ b \ c * 1 \ 0 \ 0 $	$ F(2) $
$ F(2) $		$ F(1) $
$ F(1) $		$ F(0) $

Now similarly for $n = 4$

$ F(4) $	$= a \ b \ c * 1 \ 0 \ 0 $	$ F(3) $
$ F(3) $		$ F(2) $
$ F(2) $		$ F(1) $

$$\begin{array}{c} \dots \quad 2 \text{ times} \dots \\ | F(4) | = | a \ b \ c | * | a \ b \ c | * | a \ b \ c | * | F(2) | \\ | F(3) | \quad | 1 \ 0 \ 0 | \quad | 1 \ 0 \ 0 | \quad | F(1) | \\ | F(2) | \quad | 0 \ 1 \ 0 | \quad | 0 \ 1 \ 0 | \quad | F(0) | \end{array}$$

So for n , the Equation (1) changes to

$$\begin{array}{c} \dots \quad n-2 \text{ times} \dots \\ | F(n) | = | a \ b \ c | * | a \ b \ c | * \dots * | a \ b \ c | * | F(2) | \\ | F(n-1) | \quad | 1 \ 0 \ 0 | \quad | 1 \ 0 \ 0 | \quad | 1 \ 0 \ 0 | \quad | F(1) | \\ | F(n-2) | \quad | 0 \ 1 \ 0 | \quad | 0 \ 1 \ 0 | \quad | 0 \ 1 \ 0 | \quad | F(0) | \end{array}$$

$$\begin{array}{c} | F(n) | = [| a \ b \ c |]^{(n-2)} * | F(2) | \\ | F(n-1) | \quad [| 1 \ 0 \ 0 |] \quad | F(1) | \end{array}$$

$$| F(n-2) | \quad [| 0 1 0 |] \quad | F(0) |$$

So we can simply multiply our Second matrix n-2 times and then multiply it with the third matrix to get the result. Multiplication can be done in $(\log n)$ time using Divide and Conquer algorithm for power (See [this](#) or [this](#))

Let us consider the problem of finding n'th term of a series defined using below recurrence.

```
n'th term,
F(n) = F(n-1) + F(n-2) + F(n-3), n >= 3
Base Cases :
F(0) = 0, F(1) = 1, F(2) = 1
```

We can find n'th term using following :

Putting a = 1, b = 1 and c = 1 in above formula

$$\begin{array}{l} | F(n) | = [| 1 1 1 |]^{(n-2)} * | F(2) | \\ | F(n-1) | \quad [| 1 0 0 |] \quad | F(1) | \\ | F(n-2) | \quad [| 0 1 0 |] \quad | F(0) | \end{array}$$

Below is the implementation of above idea.

C++

```
// C++ program to find value of f(n) where f(n)
// is defined as
//      F(n) = F(n-1) + F(n-2) + F(n-3), n >= 3
// Base Cases :
//      F(0) = 0, F(1) = 1, F(2) = 1
#include<bits/stdc++.h>
using namespace std;

// A utility function to multiply two matrices
// a[][] and b[][], Multiplication result is
// stored back in b[][]
void multiply(int a[3][3], int b[3][3])
{
    // Creating an auxiliary matrix to store elements
    // of the multiplication matrix
    int mul[3][3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            mul[i][j] = 0;
            for (int k = 0; k < 3; k++)
                mul[i][j] += a[i][k] * b[k][j];
        }
    }
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            b[i][j] = mul[i][j];
    }
}
```

```

        mul[i][j] += a[i][k]*b[k][j];
    }
}

// storing the multiplication result in a[][]
for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
        a[i][j] = mul[i][j]; // Updating our matrix
}

// Function to compute F raise to power n-2.
int power(int F[3][3], int n)
{
    int M[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}};

    // Multiply it with initial values i.e with
    // F(0) = 0, F(1) = 1, F(2) = 1
    if (n==1)
        return F[0][0] + F[0][1];

    power(F, n/2);

    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);

    // Multiply it with initial values i.e with
    // F(0) = 0, F(1) = 1, F(2) = 1
    return F[0][0] + F[0][1];
}

// Return n'th term of a series defined using below
// recurrence relation.
// f(n) is defined as
// f(n) = f(n-1) + f(n-2) + f(n-3), n>=3
// Base Cases :
// f(0) = 0, f(1) = 1, f(2) = 1
int findNthTerm(int n)
{
    int F[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}} ;

    return power(F, n-2);
}

// Driver code
int main()
{

```

```
int n = 5;

cout << "F(5) is " << findNthTerm(n);

return 0;
}
```

Java

```
// JAVA program to find value of f(n) where
// f(n) is defined as
// F(n) = F(n-1) + F(n-2) + F(n-3), n >= 3
// Base Cases :
// F(0) = 0, F(1) = 1, F(2) = 1
import java.io.*;

class GFG {

    // A utility function to multiply two
    // matrices a[][] and b[][]
    // Multiplication result is
    // stored back in b[][]
    static void multiply(int a[][], int b[][])
    {
        // Creating an auxiliary matrix to
        // store elements of the
        // multiplication matrix
        int mul[][] = new int[3][3];
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                mul[i][j] = 0;
                for (int k = 0; k < 3; k++)
                    mul[i][j] += a[i][k]
                                * b[k][j];
            }
        }

        // storing the multiplication
        // result in a[][]
        for (int i=0; i<3; i++)
            for (int j=0; j<3; j++)

                // Updating our matrix
                a[i][j] = mul[i][j];
    }
}
```

```

// Function to compute F raise to
// power n-2.
static int power(int F[][] , int n)
{
    int M[][] = {{1, 1, 1}, {1, 0, 0},
                 {0, 1, 0}};

    // Multiply it with initial values
    // i.e with F(0) = 0, F(1) = 1,
    // F(2) = 1
    if (n == 1)
        return F[0][0] + F[0][1];

    power(F, n / 2);

    multiply(F, F);

    if (n%2 != 0)
        multiply(F, M);

    // Multiply it with initial values
    // i.e with F(0) = 0, F(1) = 1,
    // F(2) = 1
    return F[0][0] + F[0][1];
}

// Return n'th term of a series defined
// using below recurrence relation.
// f(n) is defined as
// f(n) = f(n-1) + f(n-2) + f(n-3), n>=3
// Base Cases :
// f(0) = 0, f(1) = 1, f(2) = 1
static int findNthTerm(int n)
{
    int F[][] = {{1, 1, 1}, {1, 0, 0},
                 {0, 1, 0}} ;

    return power(F, n-2);
}

// Driver code
public static void main (String[] args) {

    int n = 5;

    System.out.println("F(5) is "
                       + findNthTerm(n));
}

```

```

}

//This code is contributed by vt_m.

C#

    // C# program to find value of f(n) where
    // f(n) is defined as
    // F(n) = F(n-1) + F(n-2) + F(n-3), n >= 3
    // Base Cases :
    // F(0) = 0, F(1) = 1, F(2) = 1
    using System;

    class GFG {

        // A utility function to multiply two
        // matrices a[][] and b[] []. Multiplication
        // result is stored back in b[] []
        static void multiply(int[, ] a, int[, ] b)
        {

            // Creating an auxiliary matrix to store
            // elements of the multiplication matrix
            int[, ] mul = new int[3, 3];

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    mul[i, j] = 0;
                    for (int k = 0; k < 3; k++)
                        mul[i, j] += a[i, k] * b[k, j];
                }
            }

            // storing the multiplication result
            // in a[] []
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)

                    // Updating our matrix
                    a[i, j] = mul[i, j];
        }

        // Function to compute F raise to power n-2.
        static int power(int[, ] F, int n)
        {

            int[, ] M = { { 1, 1, 1 }, { 1, 0, 0 },
                         { 0, 1, 0 } };

```

```
// Multiply it with initial values i.e
// with F(0) = 0, F(1) = 1, F(2) = 1
if (n == 1)
    return F[0, 0] + F[0, 1];

power(F, n / 2);

multiply(F, F);

if (n % 2 != 0)
    multiply(F, M);

// Multiply it with initial values i.e
// with F(0) = 0, F(1) = 1, F(2) = 1
return F[0, 0] + F[0, 1];
}

// Return n'th term of a series defined
// using below recurrence relation.
// f(n) is defined as
// f(n) = f(n-1) + f(n-2) + f(n-3), n>=3
// Base Cases :
// f(0) = 0, f(1) = 1, f(2) = 1
static int findNthTerm(int n)
{
    int[,] F = { { 1, 1, 1 }, { 1, 0, 0 },
                { 0, 1, 0 } };

    return power(F, n - 2);
}

// Driver code
public static void Main()
{
    int n = 5;

    Console.WriteLine("F(5) is "
                      + findNthTerm(n));
}

// This code is contributed by vt_m.
```

Output :

F(5) is 7

Time Complexity of this solution : $O(\log n)$

This article is contributed by **Abhiraj Smit**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/matrix-exponentiation/>

Chapter 80

Maximize the bitwise OR of an array

Maximize the bitwise OR of an array - GeeksforGeeks

Given an array of N integers. The bitwise OR of all the elements of the array has to be maximized by performing one task. The task is to multiply any element of the array **at-most k times** with a given integer x.

Examples :

Input: a = {1, 1, 1}, k = 1, x = 2

Output: 3

Explanation: Any possible choice of doing one element of the array will result the same three numbers 1, 1, 2.

So, the result is $1 \mid 1 \mid 2 = 3$.

Input: a = {1, 2, 4, 8}, k = 2, x = 3

Output: 79

Approach: Precompute the prefix and suffix OR arrays.

In one iteration, multiply an element with x^k and do Bitwise OR it with prefix OR i.e. Bitwise OR of all previous elements and suffix OR i.e. Bitwise OR of all next elements and return the maximum value after all iterations.

C++

```
// C++ program to maximize the Bitwise
// OR Sum in given array
#include <bits/stdc++.h>
using namespace std;

// Function to maximize the bitwise
// OR sum
```

```
int maxOR(long long arr[], int n, int k, int x)
{
    long long preSum[n + 1], suffSum[n + 1];
    long long res, pow = 1;

    // Compute x^k
    for (int i = 0; i < k; i++)
        pow *= x;

    // Find prefix bitwise OR
    preSum[0] = 0;
    for (int i = 0; i < n; i++)
        preSum[i + 1] = preSum[i] | arr[i];

    // Find suffix bitwise OR
    suffSum[n] = 0;
    for (int i = n - 1; i >= 0; i--)
        suffSum[i] = suffSum[i + 1] | arr[i];

    // Find maximum OR value
    res = 0;
    for (int i = 0; i < n; i++)
        res = max(res, preSum[i] | (arr[i] * pow) | suffSum[i + 1]);

    return res;
}

// Drivers code
int main()
{
    long long arr[] = { 1, 2, 4, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2, x = 3;

    cout << maxOR(arr, n, k, x) << "\n";

    return 0;
}
```

Java

```
// Java program to maximize the Bitwise
// OR Sum in given array
import java.io.*;

class GFG {

    // Function to maximize the bitwise OR sum
```

```
public static long maxOR(long arr[], int n,
                        int k, int x)
{
    long preSum[] = new long[n + 1];
    long suffSum[] = new long[n + 1];
    long res = 0, pow = 1;

    // Compute x^k
    for (int i = 0; i < k; i++)
        pow *= x;

    // Find prefix bitwise OR
    preSum[0] = 0;
    for (int i = 0; i < n; i++)
        preSum[i + 1] = preSum[i] | arr[i];

    // Find suffix bitwise OR
    suffSum[n] = 0;
    for (int i = n - 1; i >= 0; i--)
        suffSum[i] = suffSum[i + 1] | arr[i];

    // Find maximum OR value
    res = 0;
    for (int i = 0; i < n; i++)
        res = Math.max(res, preSum[i] |
                       (arr[i] * pow) | suffSum[i + 1]));

    return res;
}

// Drivers code
public static void main(String args[])
{
    long arr[] = { 1, 2, 4, 8 };
    int n = 4;
    int k = 2, x = 3;

    long ans = maxOR(arr, n, k, x);
    System.out.println(ans);
}
}

// This code is contributed by Jaideep Pyne
```

Python 3

```
# Python 3 program to maximize the Bitwise
# OR Sum in given array
```

```
# Function to maximize the bitwise
# OR sum
def maxOR(arr, n, k, x):

    preSum = [0] * (n + 1)
    suffSum = [0] * (n + 1)
    pow = 1

    # Compute x^k
    for i in range(0 ,k):
        pow *= x

    # Find prefix bitwise OR
    preSum[0] = 0
    for i in range(0, n):
        preSum[i + 1] = preSum[i] | arr[i]

    # Find suffix bitwise OR
    suffSum[n] = 0
    for i in range(n-1, -1, -1):
        suffSum[i] = suffSum[i + 1] | arr[i]

    # Find maximum OR value
    res = 0
    for i in range(0 ,n):
        res = max(res, preSum[i] |
                  (arr[i] * pow) | suffSum[i + 1]))

    return res

# Drivers code
arr = [1, 2, 4, 8 ]
n = len(arr)
k = 2
x = 3
print(maxOR(arr, n, k, x))

# This code is contributed by Smitha

C#
// C# program to maximize the Bitwise
// OR Sum in given array
using System;

class GFG {
```

```
// Function to maximize the bitwise OR sum
public static long maxOR(long []arr, int n,
                         int k, int x)
{
    long []preSum = new long[n + 1];
    long []suffSum = new long[n + 1];
    long res = 0, pow = 1;

    // Compute x^k
    for (int i = 0; i < k; i++)
        pow *= x;

    // Find prefix bitwise OR
    preSum[0] = 0;
    for (int i = 0; i < n; i++)
        preSum[i + 1] = preSum[i] | arr[i];

    // Find suffix bitwise OR
    suffSum[n] = 0;
    for (int i = n - 1; i >= 0; i--)
        suffSum[i] = suffSum[i + 1] | arr[i];

    // Find maximum OR value
    res = 0;
    for (int i = 0; i < n; i++)
        res = Math.Max(res, preSum[i] |
                       (arr[i] * pow) | suffSum[i + 1]);

    return res;
}

// Drivers code
public static void Main()
{
    long []arr = { 1, 2, 4, 8 };
    int n = 4;
    int k = 2, x = 3;

    long ans = maxOR(arr, n, k, x);
    Console.WriteLine(ans);
}
}

// This code is contributed by Smitha
```

PHP

```
<?php
```

```
// PHP program to maximize the
// Bitwise OR Sum in given array

// Function to maximize
// the bitwise OR sum

function maxOR($arr, $n, $k, $x)
{
    $res; $pow = 1;

    // Compute x^k
    for ($i = 0; $i < $k; $i++)
        $pow *= $x;

    // Find prefix bitwise OR
    $preSum[0] = 0;
    for ($i = 0; $i < $n; $i++)
        $preSum[$i + 1] = $preSum[$i] |
                           $arr[$i];

    // Find suffix bitwise OR
    $suffSum[$n] = 0;
    for ($i = $n - 1; $i >= 0; $i--)
        $suffSum[$i] = $suffSum[$i + 1] |
                       $arr[$i];

    // Find maximum OR value
    $res = 0;
    for ($i = 0; $i < $n; $i++)
        $res = max($res, $preSum[$i] |
                   ($arr[$i] * $pow) |
                   $suffSum[$i + 1]);

    return $res;
}

// Driver Code
$arr = array(1, 2, 4, 8);
$n = sizeof($arr);
$k = 2; $x = 3;

echo maxOR($arr, $n, $k, $x), "\n";

// This code is contributed by jit_t
?>
```

Output :

79

Improved By : [jaideeppyne1997](#), [jit_t](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/maximize-the-bitwise-or-of-an-array/>

Chapter 81

Maximize the number of segments of length p, q and r

Maximize the number of segments of length p, q and r - GeeksforGeeks

Given a rod of length L, the task is to cut the rod in such a way that the total number of segments of length p, q and r is maximized. The segments can only be of length p, q, and r.

Examples:

Input: l = 11, p = 2, q = 3, r = 5

Output: 5

Segments of 2, 2, 2, 2 and 3

Input: l = 7, p = 2, q = 5, r = 5

Output: 2

Segments of 2 and 5

Approach: [Dynamic Programming](#) is used to solve this problem. Initialize dp[] array to 0. Iterate till the length of the rod. For every i, a cut of p, q and r if possible is done. Initialize $ans[i+p] = \max(ans[i+p], 1 + ans[i])$, $ans[i+q] = \max(ans[i+q], 1 + ans[i])$ and $ans[i+r] = \max(ans[i+r], 1 + ans[i])$ for all the possible cuts. $ans[i]$ will be 0 if a cut at i-th index is not possible. $ans[l]$ will give the maximum number of cuts possible.

Below is the implementation of the above approach:

C++

```
// C++ program to maximize the number
// of segments of length p, q and r
#include <bits/stdc++.h>
using namespace std;
```

```
// Function that returns the maximum number
// of segments possible
int findMaximum(int l, int p, int q, int r)
{
    // Array to store the cut at each length
    int dp[l + 1];

    // All values with -1
    memset(dp, -1, sizeof(dp));

    // if length of rod is 0 then total cuts will be 0
    // so, initialize the dp[0] with 0
    dp[0] = 0;

    for (int i = 0; i <= l; i++) {

        // if certain length is not possible
        if (dp[i] == -1)
            continue;

        // if a segment of p is possible
        if (i + p <= l)
            dp[i + p] = max(dp[i + p], dp[i] + 1);

        // if a segment of q is possible
        if (i + q <= l)
            dp[i + q] = max(dp[i + q], dp[i] + 1);

        // if a segment of r is possible
        if (i + r <= l)
            dp[i + r] = max(dp[i + r], dp[i] + 1);
    }

    // return value corresponding to length l
    return dp[l];
}

// Driver Code
int main()
{
    int l = 11, p = 2, q = 3, r = 5;

    // Calling Function
    int ans = findMaximum(l, p, q, r);
    cout << ans;

    return 0;
}
```

}

Java

```
// Java program to maximize
// the number of segments
// of length p, q and r
import java.io.*;

class GFG
{

    // Function that returns
    // the maximum number
    // of segments possible
    static int findMaximum(int l, int p,
                           int q, int r)
    {

        // Array to store the
        // cut at each length
        int dp[] = new int[l + 1];

        // All values with -1
        for(int i = 0; i < l + 1; i++)
            dp[i] = -1;

        // if length of rod is 0
        // then total cuts will
        // be 0 so, initialize
        // the dp[0] with 0
        dp[0] = 0;

        for (int i = 0; i <= l; i++)
        {

            // if certain length
            // is not possible
            if (dp[i] == -1)
                continue;

            // if a segment of
            // p is possible
            if (i + p <= l)
                dp[i + p] = Math.max(dp[i + p],
                                     dp[i] + 1);
        }
    }
}
```

```
// if a segment of
// q is possible
if (i + q <= l)
    dp[i + q] = Math.max(dp[i + q],
                          dp[i] + 1);

// if a segment of
// r is possible
if (i + r <= l)
    dp[i + r] = Math.max(dp[i + r],
                          dp[i] + 1);
}

// return value corresponding
// to length l
return dp[l];
}

// Driver Code
public static void main (String[] args)
{
    int l = 11, p = 2,
        q = 3, r = 5;

    // Calling Function
    int ans = findMaximum(l, p, q, r);
    System.out.println( ans);
}
}

// This code is contributed
// by anuj_67.
```

C#

```
// C# program to maximize
// the number of segments
// of length p, q and r
using System;

class GFG
{

    // Function that returns
    // the maximum number
    // of segments possible
    static int findMaximum(int l, int p,
                           int q, int r)
```

```
{  
  
    // Array to store the  
    // cut at each length  
    int []dp = new int[l + 1];  
  
    // All values with -1  
    for(int i = 0; i < l + 1; i++)  
        dp[i] = -1;  
  
    // if length of rod is 0  
    // then total cuts will  
    // be 0 so, initialize  
    // the dp[0] with 0  
    dp[0] = 0;  
  
    for (int i = 0; i <= l; i++)  
    {  
  
        // if certain length  
        // is not possible  
        if (dp[i] == -1)  
            continue;  
  
        // if a segment of  
        // p is possible  
        if (i + p <= l)  
            dp[i + p] = Math.Max(dp[i + p],  
                                  dp[i] + 1);  
  
        // if a segment of  
        // q is possible  
        if (i + q <= l)  
            dp[i + q] = Math.Max(dp[i + q],  
                                  dp[i] + 1);  
  
        // if a segment of  
        // r is possible  
        if (i + r <= l)  
            dp[i + r] = Math.Max(dp[i + r],  
                                  dp[i] + 1);  
    }  
  
    // return value corresponding  
    // to length l  
    return dp[l];  
}
```

```
// Driver Code
public static void Main ()
{
    int l = 11, p = 2,
        q = 3, r = 5;

    // Calling Function
    int ans = findMaximum(l, p, q, r);
    Console.WriteLine(ans);
}

// This code is contributed
// by anuj_67.
```

Output:

5

Time Complexity: O(N)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximize-the-number-of-segments-of-length-p-q-and-r/>

Chapter 82

Maximize the total profit of all the persons

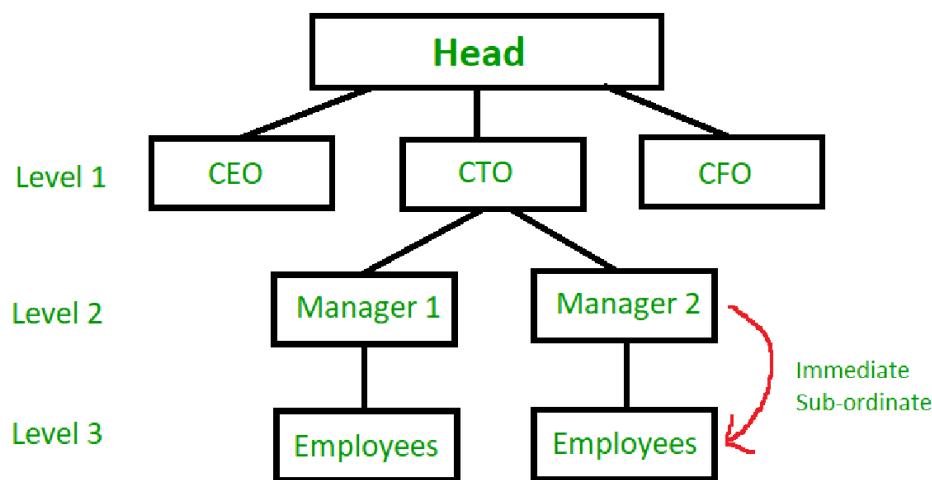
Maximize the total profit of all the persons - GeeksforGeeks

There is a hierarchical structure in an organization. A party is to be organized. No two immediate subordinates can come to the party. A profit is associated with every person. You have to maximize the total profit of all the persons who come to the party.

Hierarchical Structure

In a hierarchical organization, all employees(except the one at the head) are subordinates of some other employee.

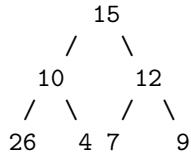
Employees only directly report to their immediate superior. This allows for a flexible and efficient structure.



For purposes of this problem, this structure may be imagined as a tree, with each employee as its node.

Examples:

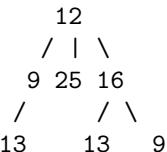
Input:



Output: The Maximum Profit would be $15+12+26+9 = 62$

The Parent 15 chooses sub-ordinate 12, 10 chooses 26 and 12 chooses 9.

Input:



Output: $12+25+13+13 = 63$

Approach:

Given the profit from each employee, we have to find the maximum sum such that no two employees(Nodes) with the same superior(Parent) are invited. This can be achieved if each employee selects the subordinate with the maximum contribution to go.

In the program, the hierarchy of the company is implemented in the form of a dictionary, with the key being a unique employee ID, and data being an array of the form [Profit associated with this employ, [List of immediate Sub-ordinates]].

For each Employee, the subordinate with the highest profit associated is added to the total profit. Further, the Employee at the head is always invited.

```

def getMaxProfit(hier):
    # The head has no competition and therefore invited
    totSum = hier[1][0]
    for i in hier:
        currentSuperior = hier[i]
        selectedSub = 0
        # select the subordinate with maximum
        # value of each superior
        for j in currentSuperior[1]:
            if(hier[j][0] > selectedSub):
                selectedSub = hier[j][0]
        totSum += selectedSub
    return totSum

# driver function
  
```

```
def main():
    # Define the Organization as a Dictionary
    # Index : [Profit, List of Employees]
        # Same as example 1 given above
    # 1:15          /          \
    # 2:10          3:12          \
    #      /      \      /      \
    # 4:26  5:4  6:7  7:9

    organization = {1:[15, [2, 3]],
                    2:[10, [4, 5]], 3:[12, [6, 7]],
                    4:[26, []], 5:[4, []], 6:[7, []], 7:[9, []]}
    maxProfit = getMaxProfit(organization)
    print(maxProfit)

main()
```

Output:

62

Source

<https://www.geeksforgeeks.org/maximize-the-total-profit-of-all-the-persons/>

Chapter 83

Maximum Possible Product in Array after performing given Operations

Maximum Possible Product in Array after performing given Operations - GeeksforGeeks

Given an array with size N. You are allowed to perform two types of operations on the given array as described below:

1. Choose some position i and j , such that (i is not equals to j), replace the value of $a[j]$ with $a[i]*a[j]$ and remove the number from the i^{th} cell.
2. Choose some position i and remove the number from the i^{th} cell (This operation can be performed at-most once and at any point of time, not necessarily in the beginning).

The task is to perform exactly $N-1$ operations with the array in such a way that the only number that remains in the array is maximum possible. This number can be rather large, so instead of printing it, print the sequence of operations which leads to this maximum number.

The output should contain exactly $N-1$ lines:

- If the operation is of the first type then print **1 i j**.
- If the operation is of the second type then print **2 i**.

Note: The array is considered to have 1-based indexing.

Examples:

```
Input : a[] = { 5, -2, 0, 1, -3 }
Output : 2 3
```

```
1 1 2  
1 2 4  
1 4 5
```

Explanation:

Step 1: $a[3]$ is removed.
Step 2: $a[2] = a[2]*a[1] = -10$; $a[1]$ is removed.
Step 3: $a[4] = a[4]*a[2] = -10$; $a[2]$ is removed.
Step 4: $a[5] = a[5]*a[4] = 30$; $a[4]$ is removed.
So, the maximum product is 30.

```
Input : a[] = { 0, 0, 0, 0 }  
Output : 1 1 2  
         1 2 3  
         1 3 4
```

Approach : There are several cases in the problem. Let the number of zeroes in the array be **cntZero** and the number of negative elements be **cntNeg**. Also let **maxNeg** be the position of the maximum negative element in the array, or -1 if there are no negative elements in the array.

Let the answer part be the product of all the numbers which will be in the answer and the removed part be the product of all the numbers which will be removed by the second type of operation.

The cases are as follows:

1. The first case is when **cntZero=0** and **cntNeg=0**. Then the answer part is the product of all the numbers in the array. The removed part is empty.
2. The second case is when **cntNeg** is odd. Then the answer part is the product of all the numbers in the array except all zeroes and $a[\text{maxNeg}]$. The removed part is the product of all zeroes and $a[\text{maxNeg}]$.
3. The third case is when **cntNeg** is even. Then the answer part is the product of all the numbers in the array except all zeroes. The removed part is the product of all zeroes in the array (be careful in case **cntNeg=0** and **cntZero=n**).

Below is the implementation of the above idea:

```
// CPP program for maximum possible product  
// with given array of numbers  
#include <bits/stdc++.h>  
using namespace std;  
  
// Function that prints operations in each step  
void MaximumProduct(int a[], int n)  
{  
    int cntneg = 0;  
    int cntzero = 0;
```

```
int used[n] = { 0 };
int pos = -1;

// count number of zeros and negative numbers
for (int i = 0; i < n; ++i) {
    if (a[i] == 0) {
        used[i] = 1;
        cntzero++;
    }

    if (a[i] < 0) {
        cntneg++;

        // To get negative number which
        // is nearest to zero, that is maximum
        // negative number
        if (pos == -1 || abs(a[pos]) > abs(a[i]))
            pos = i;
    }
}

// if number of negative number are odd then
// remove negative number at position pos
if (cntneg % 2 == 1)
    used[pos] = 1;

// initial condition
if (cntzero == n || (cntzero == n - 1 &&
                      cntneg == 1)) {
    for (int i = 0; i < n - 1; ++i)
        cout << 1 << " " << i + 1 << " "
                                      << i + 2 << endl;
    return;
}

int lst = -1;
for (int i = 0; i < n; ++i) {
    if (used[i]) {
        if (lst != -1)
            cout << 1 << " " << lst + 1 << " "
                                      << i + 1 << endl;
        lst = i;
    }
}

// perform second type operation
if (lst != -1)
    cout << 2 << " " << lst + 1 << endl;
```

```
lst = -1;

// for remaining numbers
for (int i = 0; i < n; ++i) {
    // if it is not removed yet
    if (!used[i]) {
        if (lst != -1)
            cout << 1 << " " << lst + 1 << " "
                << i + 1 << endl;
        lst = i;
    }
}

// Driver code
int main()
{
    int a[] = { 5, -2, 0, 1, -3 };

    int n = sizeof(a) / sizeof(a[0]);

    MaximumProduct(a, n);

    return 0;
}
```

Output:

```
2 3
1 1 2
1 2 4
1 4 5
```

Source

<https://www.geeksforgeeks.org/maximum-possible-product-in-array-after-performing-given-operations/>

Chapter 84

Maximum difference between groups of size two

Maximum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is maximum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {1, 4, 9, 6}
Output : 10
Groups formed will be (1, 4) and (6, 9),
the difference between highest sum group
(6, 9) i.e 15 and lowest sum group (1, 4)
i.e 5 is 10.
```

```
Input : arr[] = {6, 7, 1, 11}
Output : 11
Groups formed will be (1, 6) and (7, 11),
the difference between highest sum group
(7, 11) i.e 18 and lowest sum group (1, 6)
i.e 7 is 11.
```

Simple Approach: We can solve this problem by making all possible combinations and checking each set of combination difference between the group with highest sum and with the lowest sum. A total of $n*(n-1)/2$ such groups would be formed ($nC2$).

Time Complexity: $O(n^3)$, because it will take $O(n^2)$ to generate groups and to check against each group n iterations will be needed thus overall it takes $O(n^3)$ time.

Efficient Approach: We can use the greedy approach. Sort the whole array and our result is sum of last two elements minus sum of first two elements.

C++

```
// CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll CalculateMax(ll arr[], int n)
{
    // Sorting the whole array.
    sort(arr, arr + n);

    int min_sum = arr[0] + arr[1];
    int max_sum = arr[n-1] + arr[n-2];

    return abs(max_sum - min_sum);
}

// Driver code
int main()
{
    ll arr[] = { 6, 7, 1, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << CalculateMax(arr, n) << endl;
    return 0;
}
```

PHP

```
<?php
// PHP program to find minimum
// difference between groups of
// highest and lowest sums.
function CalculateMax($arr, $n)
{
    // Sorting the whole array.
    sort($arr);

    $min_sum = $arr[0] +
               $arr[1];
```

```
$max_sum = $arr[$n - 1] +  
$arr[$n - 2];  
  
return abs($max_sum -  
$min_sum);  
}  
  
// Driver code  
$arr = array (6, 7, 1, 11 );  
$n = sizeof($arr);  
echo CalculateMax($arr, $n), "\n" ;  
  
// This code is contributed by ajit  
?>
```

Output:

11

Time Complexity: O (n * log n)

Further Optimization :

Instead of sorting, we can find maximum two and minimum two in linear time and reduce time complexity to O(n).

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/maximum-difference-groups-size-two/>

Chapter 85

Maximum elements that can be made equal with k updates

Maximum elements that can be made equal with k updates - GeeksforGeeks

Given an array and a value k. We have to find the maximum number of equal elements possible for the array so that we can increase the elements of the array by incrementing a total of at-most k.

Examples:

Input : array = { 2, 4, 9 }, k = 3

Output : 2

We are allowed to do at most three increments. We can make two elements 4 by increasing 2 by 2. Note that we can not make two elements 9 as converting 4 to 9 requires 5 increments.

Input : array = { 5, 5, 3, 1 }, k = 5

Output : 3

Explanation: Here 1st and 2nd elements are equal. Then we can increase 3rd element 3 upto 5. Then k becomes $(k-2) = 3$. Now we can't increase 1 to 5 because k value is 3 and we need 4 for the updation. Thus equal elements possible are 3. Here we can also increase 1 to 5. Then also we have 3 because we can't update 3 to 5.

Input : array = { 5, 5, 3, 1 }, k = 6

Output : 4

Naive Approach: In the naive approach we have an algorithm in $O(n^2)$ time in which we check for each element how many other elements can be incremented so that they will become equal to them.

Efficient Approach: In this approach, first we will sort the array. Then we maintain two arrays. First is prefix sum array which stores the prefix sum of the array and another is $\maxx[]$ array which stores the maximum element found till every point, i.e., $\max[i]$ means

maximum element from 1 to i. After storing these values in prefix[] array and maxx[] array, we do the binary search from 1 to n(number of elements of the array) to calculate how many elements which can be incremented to make them equal. In the binary search, we use one function in which we determine *what is the number of elements can be incremented to make them equal to a single value.*

C++

```
// C++ program to find maximum elements that can
// be made equal with k updates
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the maximum number of
// equal elements possible with atmost K increment
// of values .Here we have done sliding window
// to determine that whether there are x number of
// elements present which on increment will become
// equal. The loop here will run in fashion like
// 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
bool ElementsCalculationFunc(int pre[], int maxx[],
                               int x, int k, int n)
{
    for (int i = 0, j = x; j <= n; j++, i++) {

        // It can be explained with the reasoning
        // that if for some x number of elements
        // we can update the values then the
        // increment to the segment (i to j having
        // length -> x) so that all will be equal is
        // (x*maxx[j]) this is the total sum of
        // segment and (pre[j]-pre[i]) is present sum
        // So difference of them should be less than k
        // if yes, then that segment length(x) can be
        // possible return true
        if (x * maxx[j] - (pre[j] - pre[i]) <= k)
            return true;
    }
    return false;
}

void MaxNumberOfElements(int a[], int n, int k)
{
    // sort the array in ascending order
    sort(a, a + n);
    int pre[n + 1]; // prefix sum array
    int maxx[n + 1]; // maximum value array
```

```
// Initializing the prefix array
// and maximum array
for (int i = 0; i <= n; ++i) {
    pre[i] = 0;
    maxx[i] = 0;
}

// set the first element of both
// array
maxx[0] = a[0];
pre[0] = a[0];
for (int i = 1; i < n; i++) {

    // Calculating prefix sum of the array
    pre[i] = pre[i - 1] + a[i];

    // Calculating max value upto that position
    // in the array
    maxx[i] = max(maxx[i - 1], a[i]);
}

// Binary search applied for
// computation here
int l = 1, r = n, ans;
while (l < r) {
    int mid = (l + r) / 2;

    if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n)) {
        ans = mid;
        l = mid + 1;
    }
    else
        r = mid - 1;
}

// printing result
cout << ans << "\n";
}

int main()
{
    int arr[] = { 2, 4, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    MaxNumberOfElements(arr, n, k);
    return 0;
}
```

Java

```
// java program to find maximum elements that can
// be made equal with k updates

import java.util.Arrays;
public class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static boolean ElementsCalculationFunc(int pre[],
                                           int maxx[], int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if (x * maxx[j] - (pre[j] - pre[i]) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int a[], int n, int k)
    {
        // sort the array in ascending order
        Arrays.sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
            pre[i] = 0;
            maxx[i] = 0;
```

```
}

// set the first element of both
// array
maxx[0] = a[0];
pre[0] = a[0];
for (int i = 1; i < n; i++) {

    // Calculating prefix sum of the array
    pre[i] = pre[i - 1] + a[i];

    // Calculating max value upto that position
    // in the array
    maxx[i] = Math.max(maxx[i - 1], a[i]);
}

// Binary search applied for
// computation here
int l = 1, r = n, ans=0;
while (l < r) {

    int mid = (l + r) / 2;

    if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n))
    {
        ans = mid;
        l = mid + 1;
    }
    else
        r = mid - 1;
}

// printing result
System.out.print((int)ans + "\n");
}

public static void main(String args[]) {

    int arr[] = { 2, 4, 9 };
    int n = arr.length;
    int k = 3;

    MaxNumberOfElements(arr, n, k);

}
}
```

```
// This code is contributed by Sam007

C#

// C# program to find maximum elements that can
// be made equal with k updates
using System;

class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static bool ElementsCalculationFunc(int []pre,
                                         int []maxx, int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if ((x * maxx[j] - (pre[j] - pre[i])) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int []a, int n, int k)
    {
        // sort the array in ascending order
        Array.Sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
```

```
    pre[i] = 0;
    maxx[i] = 0;
}

// set the first element of both
// array
maxx[0] = a[0];
pre[0] = a[0];
for (int i = 1; i < n; i++) {

    // Calculating prefix sum of the array
    pre[i] = pre[i - 1] + a[i];

    // Calculating max value upto that position
    // in the array
    maxx[i] = Math.Max(maxx[i - 1], a[i]);
}

// Binary search applied for
// computation here
int l = 1, r = n, ans=0;
while (l < r) {

    int mid = (l + r) / 2;

    if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n))
    {
        ans = mid;
        l = mid + 1;
    }
    else
        r = mid - 1;
}

// printing result
Console.Write ((int)ans + "\n");
}

// Driver code
public static void Main()
{
    int []arr = { 2, 4, 9 };
    int n = arr.Length;
    int k = 3;

    MaxNumberOfElements(arr, n, k);
}
```

```
}
```

```
// This code is contributed by Sam007
```

Output:

2

Time Complexity : **O(nlog(n))**
Space Complexity : **O(n)**

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/maximum-elements-can-made-equal-k-updates/>

Chapter 86

Maximum number of customers that can be satisfied with given quantity

Maximum number of customers that can be satisfied with given quantity - GeeksforGeeks

A new variety of rice has been brought in supermarket and being available for the first time, the quantity of this rice is limited. Each customer demands the rice in two different packaging of size a and size b. The sizes a and b are decided by staff as per the demand. Given the size of the packets a and b, the total quantity of rice available d and the number of customers n, find out maximum number of customers that can be satisfied with the given quantity of rice.

Display the total number of customers that can be satisfied and the index of customers that can be satisfied.

Note: If a customer orders 2 3, he requires 2 packets of size a and 3 packets of size b. Assume indexing of customers starts from 1.

Input:

The first line of input contains two integers n and d; next line contains two integers a and b. Next n lines contain two integers for each customer denoting total number of bags of size a and size b that customer requires.

Output:

Print maximum number of customers that can be satisfied and in next line print the space separated indexes of satisfied customers.

Examples:

```
Input : n = 5, d = 5
        a = 1, b = 1
        2 0
```

```
3 2
4 4
10 0
0 1
Output : 2
      5 1

Input : n = 6, d = 10000000000
        a = 9999, b = 10000
        10000 9998
        10000 10000
        10000 10000
        70000 70000
        10000 10000
        10000 10000
Output : 5
      1 2 3 5 6
```

Explanation:

In first example, the order of customers according to their demand is:

Customer ID	Demand
5	1
1	2
2	5
3	8
4	10

From this, it can easily be concluded that only customer 5 and customer 1 can be satisfied for total demand of $1 + 2 = 3$. Rest of the customer cannot purchase the remaining rice, as their demand is greater than available amount.

Approach:

In order to meet the demand of maximum number of customers we must start with the customer with minimum demand so that we have maximum amount of rice left to satisfy remaining customers. Therefore, sort the customers according to the increasing order of demand so that maximum number of customers can be satisfied.

Below is the implementation of above approach:

```
// CPP program to find maximum number
// of customers that can be satisfied
#include <bits/stdc++.h>
using namespace std;

vector<pair<long long, int> > v;
```

```
// print maximum number of satisfied
// customers and their indexes
void solve(int n, int d, int a, int b,
           int arr[][])
{
    // Creating an vector of pair of
    // total demand and customer number
    for (int i = 0; i < n; i++) {
        int m = arr[i][0], t = arr[i][1];
        v.push_back(make_pair((a * m + b * t),
                              i + 1));
    }

    // Sorting the customers according
    // to their total demand
    sort(v.begin(), v.end());

    vector<int> ans;

    // Taking the first k customers that
    // can be satisfied by total amount d
    for (int i = 0; i < n; i++) {
        if (v[i].first <= d) {
            ans.push_back(v[i].second);
            d -= v[i].first;
        }
    }

    cout << ans.size() << endl;
    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << " ";
}

// Driver program
int main()
{
    // Initializing variables
    int n = 5;
    long d = 5;
    int a = 1, b = 1;
    int arr[][] = {{2, 0},
                  {3, 2},
                  {4, 4},
                  {10, 0},
                  {0, 1}};

    solve(n, d, a, b, arr);
    return 0;
}
```

}

Output:

2
5 1

Source

<https://www.geeksforgeeks.org/maximum-number-customers-can-satisfied-given-quantity/>

Chapter 87

Maximum sum increasing subsequence from a prefix and a given element after prefix is must

Maximum sum increasing subsequence from a prefix and a given element after prefix is must
- GeeksforGeeks

Given an array of n positive integers, write a program to find the maximum sum of increasing subsequence from prefix till i-th index and also including a given kth element which is after i, i.e., k > i .

Examples :

Input : arr[] = {1, 101, 2, 3, 100, 4, 5}
i-th index = 4 (Element at 4th index is 100)
K-th index = 6 (Element at 6th index is 5.)
Output : 11
So we need to calculate the maximum sum of subsequence (1 101 2 3 100 5) such that 5 is necessarily included in the subsequence, so answer is 11 by subsequence (1 2 3 5).

Input : arr[] = {1, 101, 2, 3, 100, 4, 5}
i-th index = 2 (Element at 2nd index is 2)
K-th index = 5 (Element at 5th index is 4.)
Output : 7
So we need to calculate the maximum sum of subsequence (1 101 2 4) such that 4 is necessarily included in the subsequence, so answer is 7 by subsequence (1 2 4).

Prerequisite : [Maximum Sum Increasing Subsequence](#)

Simple Approach:

1. Construct a new array containing elements till ith index and the kth element.
2. Recursively calculate all the increasing subsequences.
3. Discard all the subsequences not having kth element included.
4. Calculate the maximum sum from the left over subsequences and display it.

Time Complexity: $O(2^n)$

Efficient Approach: Use a dynamic approach to maintain a table $dp[][]$. The value of $dp[i][k]$ stores the maximum sum of increasing subsequence till ith index and containing the kth element.

C++

```
// CPP program to find maximum sum increasing
// subsequence till i-th index and including
// k-th index.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll pre_compute(ll a[], ll n, ll index, ll k)
{
    ll dp[n][n] = { 0 };

    // Initializing the first row of the dp[] [] .
    for (int i = 0; i < n; i++) {
        if (a[i] > a[0])
            dp[0][i] = a[i] + a[0];
        else
            dp[0][i] = a[i];
    }

    // Creating the dp[] [] matrix.
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (a[j] > a[i] && j > i) {
                if (dp[i - 1][i] + a[j] > dp[i - 1][j])
                    dp[i][j] = dp[i - 1][i] + a[j];
                else
                    dp[i][j] = dp[i - 1][j];
            }
            else
                dp[i][j] = dp[i - 1][j];
        }
    }
}
```

```
// To calculate for i=4 and k=6.  
return dp[index][k];  
}  
  
int main()  
{  
    ll a[] = { 1, 101, 2, 3, 100, 4, 5 };  
    ll n = sizeof(a) / sizeof(a[0]);  
    ll index = 4, k = 6;  
    printf("%lld", pre_compute(a, n, index, k));  
    return 0;  
}
```

Java

```
// Java program to find maximum sum increasing  
// subsequence till i-th index and including  
// k-th index.  
class GFG {  
  
    static int pre_compute(int a[], int n,  
                          int index, int k)  
    {  
        int dp[][] = new int[n][n];  
  
        // Initializing the first row of  
        // the dp[][] .  
        for (int i = 0; i < n; i++) {  
            if (a[i] > a[0])  
                dp[0][i] = a[i] + a[0];  
            else  
                dp[0][i] = a[i];  
        }  
  
        // Creating the dp[][] matrix.  
        for (int i = 1; i < n; i++)  
        {  
            for (int j = 0; j < n; j++)  
            {  
                if (a[j] > a[i] && j > i)  
                {  
                    if (dp[i - 1][i] + a[j] >  
                        dp[i - 1][j])  
                        dp[i][j] = dp[i - 1][i]  
                                  + a[j];  
                    else  
                        dp[i][j] = dp[i - 1][j];  
                }  
            }  
        }  
    }  
}
```

```
        }
        else
            dp[i][j] = dp[i - 1][j];
    }
}

// To calculate for i=4 and k=6.
return dp[index][k];
}

// Driver code
public static void main(String[] args)
{
    int a[] = { 1, 101, 2, 3, 100, 4, 5 };
    int n = a.length;
    int index = 4, k = 6;
    System.out.println(
        pre_compute(a, n, index, k));
}
}

// This code is contributed by Smitha.
```

C#

```
// C# program to find maximum
// sum increasing subsequence
// till i-th index and including
// k-th index.
using System;

class GFG
{
    static int pre_compute(int []a, int n,
                          int index, int k)
    {
        int [,]dp = new int[n, n];

        // Initializing the first
        // row of the dp[][] .
        for (int i = 0; i < n; i++)
        {
            if (a[i] > a[0])
                dp[0, i] = a[i] + a[0];
            else
                dp[0, i] = a[i];
        }
    }
}
```

```
// Creating the dp[][] matrix.
for (int i = 1; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (a[j] > a[i] && j > i)
        {
            if (dp[i - 1, i] + a[j] >
                dp[i - 1, j])
                dp[i, j] = dp[i - 1, i] +
                            a[j];
            else
                dp[i, j] = dp[i - 1, j];
        }
        else
            dp[i, j] = dp[i - 1, j];
    }
}

// To calculate for i=4 and k=6.
return dp[index, k];
}

// Driver code
static public void Main ()
{
    int []a = {1, 101, 2,
               3, 100, 4, 5};
    int n = a.Length;
    int index = 4, k = 6;
    Console.WriteLine(pre_compute(a, n,
                                 index, k));
}
```

// This code is contributed by @ajit

Output:

11

Time Complexity : $O(n^2)$

Note: This approach is very useful if you have to answer multiple such queries of i and k because using the pre calculated dp matrix you can answer such query in $O(1)$ time.

To try similar problem, give this article a read: [Maximum product of an increasing subsequence](#)

Improved By : [Smitha Dinesh Semwal, jit_t](#)

Source

<https://www.geeksforgeeks.org/maximum-sum-increasing-subsequence-from-a-prefix-and-a-given-element-after-prefix-is-must/>

Chapter 88

Minimum adjacent swaps to move maximum and minimum to corners

Minimum adjacent swaps to move maximum and minimum to corners - GeeksforGeeks

Given N number of elements, find the minimum number of swaps required so that the maximum element is at the beginning and the minimum element is at last with the condition that only swapping of adjacent elements is allowed.

Examples:

Input: $a[] = \{3, 1, 5, 3, 5, 5, 2\}$

Output: 6

Step 1: Swap 5 with 1 to make the array as $\{3, 5, 1, 3, 5, 5, 2\}$

Step 2: Swap 5 with 3 to make the array as $\{5, 3, 1, 3, 5, 5, 2\}$

Step 3: Swap 1 with 3 at its right to make the array as $\{5, 3, 3, 1, 5, 5, 2\}$

Step 4: Swap 1 with 5 at its right to make the array as $\{5, 3, 3, 5, 1, 5, 2\}$

Step 5: Swap 1 with 5 at its right to make the array as $\{5, 3, 3, 5, 5, 1, 2\}$

Step 6: Swap 1 with 2 at its right to make the array as $\{5, 3, 3, 5, 5, 2, 1\}$

After performing 6 swapping operations 5 is at the beginning and 1 at the end

Input: $a[] = \{5, 6, 1, 3\}$

Output: 2

The approach will be to find the index of the largest element(let l). Find the index of the leftmost largest element if largest element appears in the array more than once. Similarly, find the index of the rightmost smallest element(let r). There exists two cases to solve this problem.

1. **Case 1:** If $l < r$: Number of swaps = $l + (n-r-1)$

2. **Case 2:** If $l > r$: Number of swaps = $l + (n-r-2)$, as one swap has already been performed while swapping the larger element to the front

C++

```
// CPP program to count Minimum number
// of adjacent /swaps so that the largest element
// is at beginning and the smallest element
// is at last
#include <bits/stdc++.h>
using namespace std;

// Function that returns the minimum swaps
void solve(int a[], int n)
{
    int maxx = -1, minn = a[0], l = 0, r = 0;
    for (int i = 0; i < n; i++) {

        // Index of leftmost largest element
        if (a[i] > maxx) {
            maxx = a[i];
            l = i;
        }

        // Index of rightmost smallest element
        if (a[i] <= minn) {
            minn = a[i];
            r = i;
        }
    }
    if (r < l)
        cout << l + (n - r - 2);
    else
        cout << l + (n - r - 1);
}

// Driver Code
int main()
{
    int a[] = { 5, 6, 1, 3 };
    int n = sizeof(a)/sizeof(a[0]);
    solve(a, n);
    return 0;
}
```

Java

```
// Java program to count Minimum number
```

```
// of swaps so that the largest element
// is at beginning and the
// smallest element is at last
import java.io.*;
class GFG {
    // Function performing calculations
    public static void minimumSwaps(int a[], int n)
    {
        int maxx = -1, minn = a[0], l = 0, r = 0;
        for (int i = 0; i < n; i++) {

            // Index of leftmost largest element
            if (a[i] > maxx) {
                maxx = a[i];
                l = i;
            }

            // Index of rightmost smallest element
            if (a[i] <= minn) {
                minn = a[i];
                r = i;
            }
        }
        if (r < l)
            System.out.println(l + (n - r - 2));
        else
            System.out.println(l + (n - r - 1));
    }

    // Driver Code
    public static void main(String args[]) throws IOException
    {
        int a[] = { 5, 6, 1, 3 };
        int n = a.length;
        minimumSwaps(a, n);
    }
}
```

Python3

```
# Python3 program to count
# Minimum number of adjacent
# swaps so that the largest
# element is at beginning and
# the smallest element is at last.
def minSwaps(arr):
    '''Function that returns
    the minimum swaps'''
```

```
n = len(arr)
maxx, minn, l, r = -1, arr[0], 0, 0

for i in range(n):

    # Index of leftmost
    # largest element
    if arr[i] > maxx:
        maxx = arr[i]
        l = i

    # Index of rightmost
    # smallest element
    if arr[i] <= minn:
        minn = arr[i]
        r = i

if r < l:
    print(l + (n - r - 2))
else:
    print(l + (n - r - 1))

# Driver code
arr = [5, 6, 1, 3]

minSwaps(arr)

# This code is contributed
# by Tuhin Patra
```

C#

```
// C# program to count Minimum
// number of swaps so that the
// largest element is at beginning
// and the smallest element is at last
using System;

class GFG
{
    // Function performing calculations
    public static void minimumSwaps(int []a,
                                    int n)
    {
        int maxx = -1, l = 0,
            minn = a[0], r = 0;
        for (int i = 0; i < n; i++)
    }
```

```
{  
  
    // Index of leftmost  
    // largest element  
    if (a[i] > maxx)  
    {  
        maxx = a[i];  
        l = i;  
    }  
  
    // Index of rightmost  
    // smallest element  
    if (a[i] <= minn)  
    {  
        minn = a[i];  
        r = i;  
    }  
    if (r < l)  
        Console.WriteLine(l + (n - r - 2));  
    else  
        Console.WriteLine(l + (n - r - 1));  
}  
  
// Driver Code  
public static void Main()  
{  
  
    int []a = { 5, 6, 1, 3 };  
    int n = a.Length;  
  
    // Calling function  
    minimumSwaps(a, n);  
}  
}  
  
// This code is contributed by anuj_67.
```

PHP

```
<?php  
// PHP program to count Minimum  
// number of adjacent /swaps so  
// that the largest element is  
// at beginning and the smallest  
// element is at last  
  
// Function that returns
```

```
// the minimum swaps
function solve($a, $n)
{
    $maxx = -1; $minn = $a[0];
    $l = 0; $r = 0;
    for ($i = 0; $i < $n; $i++)
    {

        // Index of leftmost
        // largest element
        if ($a[$i] > $maxx)
        {
            $maxx = $a[$i];
            $l = $i;
        }

        // Index of rightmost
        // smallest element
        if ($a[$i] <= $minn)
        {
            $minn = $a[$i];
            $r = $i;
        }
    }

    if ($r < $l)
        echo $l + ($n - $r - 2);
    else
        echo $l + ($n - $r - 1);
}

// Driver Code
$a = array(5, 6, 1, 3);
$n = count($a);
solve($a, $n);

// This code is contributed
// by anuj_67.
?>
```

Output:

2

Time Complexity: O(N)

Improved By : [vt_m](#), [TuhinPatra](#)

Source

<https://www.geeksforgeeks.org/minimum-adjacent-swaps-to-move-maximum-and-minimum-to-corners/>

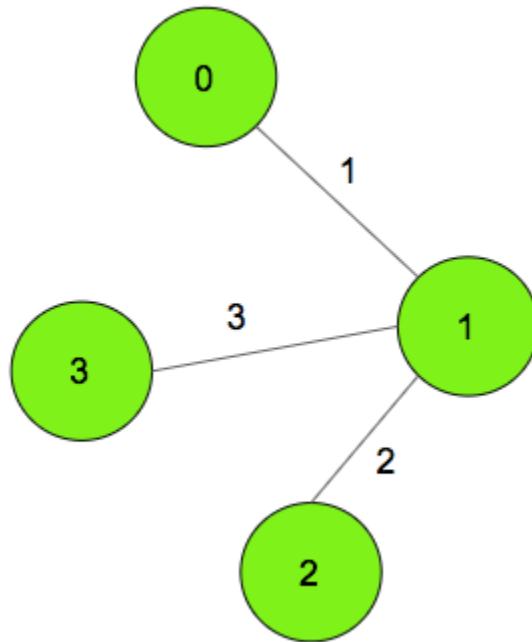
Chapter 89

Minimum cost path from source node to destination node via an intermediate node

Minimum cost path from source node to destination node via an intermediate node - Geeks-forGeeks

Given an undirected weighted graph. The task is to find the minimum cost of the path from source node to the destination node via an intermediate node.

Note: If an edge is traveled twice, only once weight is calculated as cost.



Examples:

Input: source = 0, destination = 2, intermediate = 3;

Output: 6

The minimum cost path 0->1->3->1->2

The edge (1-3) occurs twice in the path, but its weight is added only once to the answer.

Input: source = 0, destination = 2, intermediate = 1;

Output: 3

The minimum cost path is 0->1>2

Approach: Let suppose take a path P1 from Source to intermediate, and a path P2 from intermediate to destination. There can be some common edges among these 2 paths. Hence, the optimal path will always have the following form: for any node U, the walk consists of edges on the shortest path from Source to U, from intermediate to U, and from destination to U. Hence, if $\text{dist}(a, b)$ is the cost of shortest path between node a and b, the required minimum cost path will be $\min\{ \text{dist}(\text{Source}, U) + \text{dist}(\text{intermediate}, U) + \text{dist}(\text{destination}, U) \}$ for all U. The Minimum distance of all nodes from Source, intermediate, and destination can be found by doing [Dijkstra's Shortest Path algorithm](#) from these 3 nodes.

Below is the implementation of the above approach.

```
// CPP program to find minimum distance between  
// source and destination node and visiting
```

```
// of intermediate node is compulsory
#include <bits/stdc++.h>
using namespace std;
#define MAXN 100005

// to strore maped values of graph
vector<pair<int, int> > v[MAXN];

// to store distance of
// all nodes from the source node
int dist[MAXN];

// Dijkstra's algorithm to find
// shortest path from source to node
void dijkstra(int source, int n)
{
    // set all the vertices
    // distances as infinity
    for (int i = 0; i < n; i++)
        dist[i] = INT_MAX;

    // set all vertex as unvisited
    bool vis[n];
    memset(vis, false, sizeof vis);

    // make distance from source
    // vertex to source vertex is zero
    dist = 0;

    // // multiset do the job
    // as a min-priority queue
    multiset<pair<int, int> > s;

    // insert the source node with distance = 0
    s.insert({ 0, source });

    while (!s.empty()) {
        pair<int, int> p = *s.begin();
        // pop the vertex with the minimum distance
        s.erase(s.begin());

        int x = p.second;
        int wei = p.first;

        // check if the popped vertex
        // is visited before
        if (vis[x])
            continue;

        vis[x] = true;

        // update the distances of
        // adjacent vertices
        for (auto it : v[x]) {
            int adjNode = it.first;
            int adjWei = it.second;

            if (dist[adjNode] > dist[x] + adjWei) {
                dist[adjNode] = dist[x] + adjWei;
                s.insert({ dist[adjNode], adjNode });
            }
        }
    }
}
```

```
vis[x] = true;

for (int i = 0; i < v[x].size(); i++) {
    int e = v[x][i].first;
    int w = v[x][i].second;

    // check if the next vertex
    // distance could be minimized
    if (dist[x] + w < dist[e]) {

        dist[e] = dist[x] + w;

        // insert the next vertex
        // with the updated distance
        s.insert({ dist[e], e });
    }
}
}

// function to add edges in graph
void add_edge(int s, int t, int weight)
{
    v[s].push_back({ t, weight });
    v[t].push_back({ s, weight });
}

// function to find the minimum shortest path
int solve(int source, int destination,
          int intermediate, int n)
{
    int ans = INT_MAX;

    dijkstra(source, n);

    // store distance from source to
    // all other vertices
    int dsouce[n];
    for (int i = 0; i < n; i++)
        dsouce[i] = dist[i];

    dijkstra(destination, n);
    // store distance from destination
    // to all other vertices
    int ddestination[n];
    for (int i = 0; i < n; i++)
        ddestination[i] = dist[i];
```

```
dijkstra(intermediate, n);
// store distance from intermediate
// to all other vertices
int dintermediate[n];
for (int i = 0; i < n; i++)
    dintermediate[i] = dist[i];

// find required answer
for (int i = 0; i < n; i++)
    ans = min(ans, dsouce[i] + ddestination[i]
               + dintermediate[i]);

return ans;
}

// Driver code
int main()
{

    int n = 4;
    int source = 0, destination = 2, intermediate = 3;

    // add edges in graph
    add_edge(0, 1, 1);
    add_edge(1, 2, 2);
    add_edge(1, 3, 3);

    // function call for minimum shortest path
    cout << solve(source, destination, intermediate, n);

    return 0;
}
```

Output:

6

Time complexity: $O((N + M) * \log N)$, where N is number of nodes, M is number of edges.

Auxiliary Space: $O(N+M)$

Source

<https://www.geeksforgeeks.org/minimum-cost-path-from-source-node-to-destination-node-via-an-intermediate-node/>

Chapter 90

Minimum difference between groups of size two

Minimum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is minimum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {2, 6, 4, 3}
Output : 1
Groups formed will be (2, 6) and (4, 3),
the difference between highest sum group
(2, 6) i.e 8 and lowest sum group (3, 4)
i.e 7 is 1.
```

```
Input : arr[] = {11, 4, 3, 5, 7, 1}
Output : 3
Groups formed will be (1, 11), (4, 5) and
(3, 7), the difference between highest
sum group (1, 11) i.e 12 and lowest sum
group (4, 5) i.e 9 is 3.
```

Simple Approach: A simple approach would be to try against all combinations of array elements and check against each set of combination difference between the group with the highest sum and the one with lowest sum. A total of $n*(n-1)/2$ such groups would be formed ($nC2$).

Time Complexity : $O(n^3)$ To generate groups n^2 iterations will be needed and to check against each group n iterations will be needed and hence n^3 iterations will be needed in worst case.

Efficient Approach: Efficient approach would be to use the greedy approach. Sort the whole array and generate groups by selecting one element from the start of the array and one from the end.

```
// CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll calculate(ll a[], ll n)
{
    // Sorting the whole array.
    sort(a, a + n);

    // Generating sum groups.
    vector<ll> s;
    for (int i = 0, j = n - 1; i < j; i++, j--)
        s.push_back(a[i] + a[j]);

    ll mini = *min_element(s.begin(), s.end());
    ll maxi = *max_element(s.begin(), s.end());

    return abs(maxi - mini);
}

int main()
{
    ll a[] = { 2, 6, 4, 3 };
    int n = sizeof(a) / (sizeof(a[0]));
    cout << calculate(a, n) << endl;
    return 0;
}
```

Output:

1

Time Complexity: $O(n * \log n)$

Asked in: Inmobi

Reference: <https://www.hackerearth.com/problem/algorithm/project-team/>

Source

<https://www.geeksforgeeks.org/minimum-difference-between-groups-of-size-two/>

Chapter 91

Minimum digits to remove to make a number Perfect Square

Minimum digits to remove to make a number Perfect Square - GeeksforGeeks

Given a integer n, we need to find how many digits remove from the number to make it a perfect square.

Examples :

Input : 8314

Output: 81 2

Explanation: If we remove 3 and 4 number becomes 81 which is a perfect square.

Input : 57

Output : -1

The idea is to generate [all possible subsequences](#) and return optimal string using [set bits](#). Let's suppose we have a string 8314. And using set bits we form all possible subsequences i.e.,

8, 3, 83, 1, 81, 31, 831, 4, 84, 34, 834, 14, 814, 314, 8314.

After forming all possible subsequences, we check which one is the perfect square. And we return a perfect square number which has the minimum length.

In above example, three perfect squares are 1 4 and 81, so answer would be 81 because 81 has the max length 2.

C++

```
// C++ program to find required minimum digits
// need to remove to make a number perfect square
#include <bits/stdc++.h>
```

```
using namespace std;

// function to check minimum number of digits
// should be removed to make this number
// a perfect square
int perfectSquare(string s)
{
    // size of the string
    int n = s.size();

    // our final answer
    int ans = -1;

    // to store string which is perfect square.
    string num;

    // We make all possible subsequences
    for (int i = 1; i < (1 << n); i++) {
        string str = "";

        for (int j = 0; j < n; j++) {

            // to check jth bit is set or not.
            if ((i >> j) & 1) {
                str += s[j];
            }
        }

        // we do not consider a number with leading zeros
        if (str[0] != '0') {

            // convert our temporary string into integer
            int temp = 0;
            for (int j = 0; j < str.size(); j++)
                temp = temp * 10 + (int)(str[j] - '0');

            int k = sqrt(temp);

            // checking temp is perfect square or not.
            if (k * k == temp) {

                // taking maximum sized string
                if (ans < (int)str.size()) {
                    ans = (int)str.size();
                    num = str;
                }
            }
        }
    }
}
```

```
}

if (ans == -1)
    return ans;
else {

    // print PerfectSquare
    cout << num << " ";
    return n - ans;
}
}

// Driver code
int main()
{
    cout << perfectSquare("8314") << endl;
    cout << perfectSquare("753") << endl;
    return 0;
}
```

Java

```
// Java program to find required minimum digits
// need to remove to make a number perfect square
import java.io.*;
import java.lang.*;

public class GFG {

    // function to check minimum
    // number of digits should
    // be removed to make this
    // number a perfect square
    static int perfectSquare(String s)
    {
        // size of the string
        int n = s.length();

        // our final answer
        int ans = -1;

        // to store string which
        // is perfect square.
        String num = "";

        // We make all possible subsequences
        for (int i = 1; i < (1 << n); i++) {
            String str = "";
```

```
for (int j = 0; j < n; j++) {

    // to check jth bit is set or not.
    if (((i >> j) & 1) == 1) {
        str += s.charAt(j);
    }
}

// we do not consider a number
// with leading zeros
if (str.charAt(0) != '0') {

    // convert our temporary
    // string into integer
    int temp = 0;
    for (int j = 0; j <
                     str.length(); j++)
        temp = temp * 10 +
            (int)(str.charAt(j) - '0');

    int k = (int)Math.sqrt(temp);

    // checking temp is perfect
    // square or not.
    if (k * k == temp) {

        // taking maximum sized string
        if (ans < (int)str.length()) {
            ans = (int)str.length();
            num = str;
        }
    }
}

if (ans == -1)
    return ans;
else {

    // print PerfectSquare
    System.out.print(num + " ");
    return n - ans;
}
}

// Driver code
public static void main(String args[])

```

```
{  
    System.out.println(perfectSquare("8314"));  
    System.out.println(perfectSquare("753"));  
}  
}  
  
// This code is contributed by  
// Manish Shaw (manishshaw1)
```

Python3

```
# C++ program to find required minimum  
# digits need to remove to make a  
# number perfect square  
  
import math  
# function to check minimum number of  
# digits should be removed to make  
# this number a perfect square  
def perfectSquare(s) :  
  
    # size of the string  
    n = len(s)  
  
    # our final answer  
    ans = -1  
  
    # to store string which is  
    # perfect square.  
    num = ""  
  
    # We make all possible subsequences  
    for i in range(1, (1 << n)) :  
        str = ""  
  
        for j in range(0, n) :  
  
            # to check jth bit is  
            # set or not.  
            if ((i >> j) & 1) :  
                str = str + s[j]  
  
            # we do not consider a number  
            # with leading zeros  
            if (str[0] != '0') :  
  
                # convert our temporary  
                # string into integer
```

```
temp = 0;
for j in range(0, len(str)) :
    temp = (temp * 10 +
            (ord(str[j]) - ord('0')))

k = int(math.sqrt(temp))

# checking temp is perfect
# square or not.
if (k * k == temp) :

    # taking maximum sized
    # string
    if (ans < len(str)) :
        ans = len(str)
        num = str

if (ans == -1) :
    return ans
else :

    # print PerfectSquare
    print ("{} ".format(num), end="")
    return n - ans

# Driver code
print (perfectSquare("8314"))
print (perfectSquare("753"));

# This code is contributed by
# manishshaw1.
```

C#

```
// C# program to find required minimum digits
// need to remove to make a number perfect square
using System;
class GFG {

    // function to check minimum
    // number of digits should
    // be removed to make this
    // number a perfect square
    static int perfectSquare(string s)
    {
        // size of the string
        int n = s.Length;
```

```
// our final answer
int ans = -1;

// to store string which
// is perfect square.
string num = "";

// We make all possible subsequences
for (int i = 1; i < (1 << n); i++) {
    string str = "";

    for (int j = 0; j < n; j++) {

        // to check jth bit is set or not.
        if (((i >> j) & 1) == 1) {
            str += s[j];
        }
    }

    // we do not consider a number
    // with leading zeros
    if (str[0] != '0') {

        // convert our temporary
        // string into integer
        int temp = 0;
        for (int j = 0; j < str.Length; j++)
            temp = temp * 10 + (int)(str[j] - '0');

        int k = (int)Math.Sqrt(temp);

        // checking temp is perfect
        // square or not.
        if (k * k == temp) {

            // taking maximum sized string
            if (ans < (int)str.Length) {
                ans = (int)str.Length;
                num = str;
            }
        }
    }
}

if (ans == -1)
    return ans;
else {
```

```
// print PerfectSquare
Console.WriteLine(num + " ");
return n - ans;
}
}

// Driver code
public static void Main()
{
    Console.WriteLine(perfectSquare("8314"));
    Console.WriteLine(perfectSquare("753"));
}
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
<?php
// PHP program to find required
// minimum digits need to remove
// to make a number perfect square

// function to check minimum
// number of digits should be
// removed to make this number
// a perfect square
function perfectSquare($s)
{
    // size of the string
    $n = strlen($s);

    // our final answer
    $ans = -1;

    // to store string which
    // is perfect square.
    $num = "";

    // We make all possible
    // subsequences
    for ($i = 1; $i < (1 << $n); $i++)
    {
        $str = "";
        for ($j = 0; $j < $n; $j++)
        {
```

```
// to check jth bit
// is set or not.
if (($i >> $j) & 1)
{
    $str = $str.$s[$j];
}

// we do not consider a
// number with leading zeros
if ($str[0] != '0')
{
    // convert our temporary
    // string into integer
    $temp = 0;
    for ($j = 0; $j < strlen($str); $j++)
        $temp = $temp * 10 +
            (ord($str[$j]) - ord('0'));

    $k = (int)(sqrt($temp));

    // checking temp is perfect
    // square or not.
    if (($k * $k) == $temp)
    {

        // taking maximum sized string
        if ($ans < strlen($str))
        {
            $ans = strlen($str);
            $num = $str;
        }
    }
}

if ($ans == -1)
    return $ans;
else
{
    // print PerfectSquare
    echo ($num." ");
    return ($n - $ans);
}
}

// Driver code
echo (perfectSquare("8314"). "\n");
```

```
echo (perfectSquare("753") . "\n");  
// This code is contributed by  
// Manish Shaw (manishshaw1)  
?>
```

Output :

```
81 2  
-1
```

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/required-minimum-digits-remove-number-make-perfect-square/>

Chapter 92

Minimum number of elements to be removed to make XOR maximum

Minimum number of elements to be removed to make XOR maximum - GeeksforGeeks

Given a number N where, $1 \leq N \leq 10^{18}$. The task is to find the minimum number of elements to be deleted in between 1 to N such that the XOR obtained from the remaining elements is maximum.

Examples:

Input: $N = 5$
Output: 2

Input: 1000000000000000
Output: 1

Approach: Considering the following cases:

Case 1: When $N = 1$ or $N = 2$, then answer is 0. No need to remove any element.

Case 2: Now, we have to find a number which is power of 2 and greater than or equal to N .

Let's call this number be a .

So, if $N = 2^a$ or $N = a - 1$ then we will just remove $a - 1$.
Hence the answer is 1.

else if $N = 2^a - 1$, then answer is 0. No need to remove any element.

Case 3: Otherwise, if n is ~~odd~~, then answer is **1**.
else if n is ~~odd~~, then answer is **2**.

Below is the implementation of above approach:

```
// C++ implementation to find minimum number of
// elements to remove to get maximum XOR value
#include <bits/stdc++.h>
using namespace std;

unsigned int nextPowerOf2(unsigned int n)
{
    unsigned count = 0;

    // First n in the below condition
    // is for the case where n is 0
    if (n && !(n & (n - 1)))
        return n;

    while (n != 0) {
        n >>= 1;
        count += 1;
    }

    return 1 << count;
}

// Function to find minimum number of
// elements to be removed.
int removeElement(unsigned int n)
{

    if (n == 1 || n == 2)
        return 0;

    unsigned int a = nextPowerOf2(n);

    if (n == a || n == a - 1)
        return 1;

    else if (n == a - 2)
        return 0;

    else if (n % 2 == 0)
        return 1;

    else
}
```

```
        return 2;
}

// Driver code
int main()
{
    unsigned int n = 5;

    // print minimum number of elements
    // to be removed
    cout << removeElement(n);

    return 0;
}
```

Output:

2

Time complexity: O(logn)

Source

<https://www.geeksforgeeks.org/minimum-number-of-elements-to-be-removed-to-make-xor-maximum/>

Chapter 93

Minimum number using set bits of a given number

Minimum number using set bits of a given number - GeeksforGeeks

Given an unsigned number, find the minimum number that could be formed by using the bits of the given unsigned number.

Examples :

Input : 6

Output : 3

Binary representation of 6 is 0000....0110. Smallest number with same number of set bits 0000....0011.

Input : 11

Output : 7

Simple Approach:

1. Find binary representation of the number using simple decimal to binary representation technique.
2. Count number of set bits in the binary representation equal to ‘n’.
3. Create a binary representation with it’s ‘n’ least significant bits set to 1.
4. Convert the binary representation back to the number.

Efficient Approach:

1. Just measure the number of 1’s present in the bit representation of the number.
2. (Number of set bits raised to the power of 2) – 1 represents the minimized number.

C++

```
// An efficient C++ program to find  
// minimum number formed by bits of a given number.  
#include <bits/stdc++.h>
```

```
#define ll unsigned int
using namespace std;

// Returns minimum number formed by
// bits of a given number.
ll minimize(ll a)
{
    // __popcnt32(a) gives number of 1's
    // present in binary representation
    // of a.
    ll n = __popcnt32(a);

    return (pow(2, n) - 1);
}

// Driver function.
int main()
{
    ll a = 11;
    cout << minimize(a) << endl;
    return 0;
}
```

Java

```
// An efficient Java program to
// find minimum number formed
// by bits of a given number.
import java.io.*;

class GFG
{
    public static int __popcnt32(long number)
    {
        int count = 0;
        while (number > 0)
        {
            count += number & 1L;
            number >= 1L;
        }
        return count;
    }

    // Returns minimum number formed
    // by bits of a given number.
    static long minimize(long a)
    {
        // __popcnt32(a) gives number
```

```
// of 1's present in binary
// representation of a.
int n = __popcnt32(a);

return ((long)Math.pow(2, n) - 1);
}

// Driver Code.
public static void main(String args[])
{
    long a = 11;
    System.out.print(minimize(a));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
// An efficient C# program to
// find minimum number formed
// by bits of a given number.
using System;
using System.Linq;
using System.Collections.Generic;

class GFG
{
    // Returns minimum number formed
    // by bits of a given number.
    static long minimize(long a)
    {
        // __popcnt32(a) gives number
        // of 1's present in binary
        // representation of a.
        string binaryString = Convert.ToString(a, 2);
        int n = binaryString.Split(new [] {'0'},
            StringSplitOptions.RemoveEmptyEntries).Length + 1;

        return ((long)Math.Pow(2, n) - 1);
    }

    // Driver Code.
    static void Main()
    {
        long a = 11;
        Console.Write(minimize(a));
```

```
    }  
}  
  
// This code is contributed by  
// Manish Shaw(manishshaw1)
```

Output :

7

Note : The above code uses GCC specific functions. If we wish to write code for other compilers, we may use Count set bits in an integer.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/minimum-number-using-set-bits-given-number/>

Chapter 94

Minimum operations required to make all the elements distinct in an array

Minimum operations required to make all the elements distinct in an array - GeeksforGeeks

Given an array of N integers. If a number occurs more than once, choose any number y from the array and replace the x in the array to x+y such that x+y is not in the array. The task is to find the minimum number of operations to make the array a distinct one.

Examples:

Input: a[] = {2, 1, 2}

Output: 1

Let x = 2, y = 1 then replace 2 by 3.

Performing the above step makes all the elements in the array distinct.

Input: a[] = {1, 2, 3}

Output: 0

Approach: If a number appears more than once, then the summation of (occurrences-1) for all duplicate elements will be the answer. The main logic behind this is if x is replaced by x+y where y is the largest element in the array, then x is replaced by x+y which is the largest element in the array. Use a `map` to store the frequency of the numbers of array. Traverse in the map, and if the frequency of an element is more than 1, add it to the count by subtracting one.

Below is the implementation of the above approach:

```
// C++ program to find Minimum number
// of changes to make array distinct
#include <bits/stdc++.h>
```

```
using namespace std;

// Function that returns minimum number of changes
int minimumOperations(int a[], int n)
{

    // Hash-table to store frequency
    unordered_map<int, int> mp;

    // Increase the frequency of elements
    for (int i = 0; i < n; i++)
        mp[a[i]] += 1;

    int count = 0;

    // Traverse in the map to sum up the (occurrences-1)
    // of duplicate elements
    for (auto it = mp.begin(); it != mp.end(); it++) {
        if ((*it).second > 1)
            count += (*it).second - 1;
    }
    return count;
}

// Driver Code
int main()
{
    int a[] = { 2, 1, 2, 3, 3, 4, 3 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << minimumOperations(a, n);
    return 0;
}
```

Output:

3

Time Complexity: O(N)

Source

<https://www.geeksforgeeks.org/minimum-operations-required-to-make-all-the-elements-distinct-in-an-array/>

Chapter 95

Minimum steps to reach end from start by performing multiplication and mod operations with array elements

Minimum steps to reach end from start by performing multiplication and mod operations with array elements - GeeksforGeeks

Given start, end and an array of N numbers. At each step, start is multiplied with any number in the array and then mod operation with 100000 is done to get the new start. The task is to find the minimum steps in which end can be achieved starting from start.

Examples:

Input: start = 3 end = 30 a[] = {2, 5, 7}

Output: 2

Step 1: $3*2 = 6 \ \% \ 100000 = 6$

Step 2: $6*5 = 30 \ \% \ 100000 = 30$

Input: start = 7 end = 66175 a[] = {3, 4, 65}

Output: 4

Step 1: $7*3 = 21 \ \% \ 100000 = 21$

Step 2: $21*3 = 63 \ \% \ 100000 = 63$

Step 3: $63*65 = 4095 \ \% \ 100000 = 4095$

Step 4: $4095*65 = 266175 \ \% \ 100000 = 66175$

Approach: Since in the above problem the modulus given is 100000, therefore the maximum number of states will be 10^5 . All the states can be checked using simple **BFS**. Initialize an ans[] array with -1 which marks that the state has not been visited. ans[i] stores the number of steps taken to reach i from start. Initially push the start to the queue, then apply BFS.

Pop the top element and check if it is equal to the end, if it is then print the ans[end]. If the element is not equal to the topmost element, then multiply top element with every element in the array and perform a mod operation. If the multiplied element state has not been visited previously, then push it into the queue. Initialize ans[pushed_element] by ans[top_element] + 1. Once all the states are visited, and the state cannot be reached by performing every possible multiplication, then print -1.

Below is the implementation of the above approach:

```
// C++ program to find the minimum steps
// to reach end from start by performing
// multiplications and mod operations with array elements
#include <bits/stdc++.h>
using namespace std;

// Function that returns the minimum operations
int minimumMultiplications(int start, int end, int a[], int n)
{
    // array which stores the minimum steps
    // to reach i from start
    int ans[100001];

    // -1 indicated the state has not been visited
    memset(ans, -1, sizeof(ans));
    int mod = 100000;

    // queue to store all possible states
    queue<int> q;

    // initially push the start
    q.push(start % mod);

    // to reach start we require 0 steps
    ans[start] = 0;

    // till all states are visited
    while (!q.empty()) {

        // get the topmost element in the queue
        int top = q.front();

        // pop the topmost element
        q.pop();

        // if the topmost element is end
        if (top == end)
            return ans[end];

        // perform multiplication with all array elements
        for (int i = 0; i < n; i++) {
            int prod = (top * a[i]) % mod;
            if (ans[prod] == -1) {
                ans[prod] = ans[top] + 1;
                q.push(prod);
            }
        }
    }
}
```

```
for (int i = 0; i < n; i++) {  
    int pushed = top * a[i];  
    pushed = pushed % mod;  
  
    // if not visited, then push it to queue  
    if (ans[pushed] == -1) {  
        ans[pushed] = ans[top] + 1;  
        q.push(pushed);  
    }  
}  
}  
return -1;  
}  
  
// Driver Code  
int main()  
{  
    int start = 7, end = 66175;  
    int a[] = { 3, 4, 65 };  
    int n = sizeof(a) / sizeof(a[0]);  
  
    // Calling function  
    cout << minimumMultiplications(start, end, a, n);  
    return 0;  
}
```

Output:

4

Source

<https://www.geeksforgeeks.org/minimum-steps-to-reach-end-from-start-by-performing-multiplication-and-mod-ope>

Chapter 96

Minimum total cost incurred to reach the last station

Minimum total cost incurred to reach the last station - GeeksforGeeks

Given an array where each element denotes the number of chocolates corresponding to each station and to move from station i to station $i+1$, we get $A[i] - A[i+1]$ chocolates for free, if this number is negative, we lose that many chocolates also.

You can only move from station i to station $i+1$, if we have non-negative number of chocolates.

Given the cost of one chocolate p , the task to find the minimum cost incurred in reaching last station from the first station (station 1).

Note: Initially we have 0 chocolates.

Examples:

```
Input : arr[] = {1, 2, 3}    p = 10
Output : 30
Initial choc_buy = 1 (arr[0])
To reach index 0 to 1, arr[0] - arr[1] = -1,
so choc_buy +=1.
Similarly To reach index 2 to 1,
arr[1] - arr[2] = -1, so choc_buy +=1.
Total choc_buy = 3.
Total cost = choc_by * p = 3*10 = 30.
```

```
Input : arr[] = {10, 6, 11, 4, 7, 1}    p = 5
Output : 55
Initial choc_buy = 10 (arr[0])
To reach index 0 to 1, arr[0] - arr[1] = 4,
so curr_choc =4.
Similarly To reach index 2 to 1,
```

```
arr[1] - arr[2] = -5, so curr_choc=4+-5 = -1.  
choc_buy += abs(-1).  
So, similarly till last station, Total choc_buy = 11.  
Total cost = choc_by * p = 11*5 = 55.
```

Approach :

1. Initialize choc_buy with the first element of array and curr_choc =0.
2. Add arr[i]-arr[i+1] to curr_choc for every i.
 - a) Check if curr_choc becomes negative.
choc_buy += abs(arr[i]- arr[i+1])
curr_choc = 0
3. Return choc_buy * p.

C++

```
// C++ program to calculate  
// minimum cost for candies  
#include <bits/stdc++.h>  
using namespace std;  
  
// Function to find minimum cost  
// to be incurred  
int findMinCost(int arr[], int n, int choc_cost) {  
  
    // To reach first station, initial  
    // chocolates required  
    int choc_buy = arr[0];  
    int curr_choc = 0;  
  
    // Start traversing  
    for (int i = 0; i < n - 1; i++) {  
  
        // Find no. of chocolates  
        // lose or gain  
        int choc = arr[i] - arr[i + 1];  
  
        // Add into curr_coc  
        curr_choc += choc;  
  
        // if no. of chocolates becomes  
        // negative that means we have  
        // to buy that no. of chocolates  
        if (curr_choc < 0) {  
            choc_buy += abs(curr_choc);  
            curr_choc = 0;  
        }  
    }  
}
```

```
// Return cost required
return choc_buy * choc_cost;
}

// Drivers code
int main() {
    int arr[] = {10, 6, 11, 4, 7, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Price of each candy
    int p = 5;

    cout << findMinCost(arr, n, p);

    return 0;
}
```

Java

```
// Java program to calculate
// minimum cost for candies
import java.io.*;

class GFG
{
    // Function to find minimum cost
    // to be incurred
    static int findMinCost(int arr[], int n, int choc_cost)
    {

        // To reach first station, initial
        // chocolates required
        int choc_buy = arr[0];
        int curr_choc = 0;

        // Start traversing
        for (int i = 0; i < n - 1; i++)
        {

            // Find no. of chocolates
            // lose or gain
            int choc = arr[i] - arr[i + 1];

            // Add into curr_coc
            curr_choc += choc;

            // if no. of chocolates becomes
            // negative that means we have
```

```
// to buy that no. of chocolates
if (curr_choc < 0)
{
    choc_buy += Math.abs(curr_choc);
    curr_choc = 0;
}
}

// Return cost required
return choc_buy * choc_cost;
}

// Drivers code
public static void main (String[] args)
{
    int arr[] = {10, 6, 11, 4, 7, 1};
    int n = arr.length;

    // Price of each candy
    int p = 5;

    System.out.println ( findMinCost(arr, n, p));
}
}

// This code is contributed by vt_m
```

Python3

```
# Python 3 program to calculate
# minimum cost for candies

# Function to find minimum cost
# to be incurred
def findMinCost(arr, n, choc_cost):

    # To reach first station,
    # initial chocolates required
    choc_buy = arr[0]
    curr_choc = 0

    # Start traversing
    for i in range(0,n - 1):

        # Find no. of chocolates lose
        # or gain
        choc = arr[i] - arr[i + 1]
```

```
# Add into curr_coc
curr_choc += choc

# if no. of chocolates becomes
# negative that means we have
# to buy that no. of chocolates
if (curr_choc < 0):
    choc_buy += abs(curr_choc)
    curr_choc = 0

# Return cost required
return choc_buy * choc_cost

# Drivers code
arr = [10, 6, 11, 4, 7, 1]
n = len(arr)

# Price of each candy
p = 5

print(findMinCost(arr, n, p))

# This code is contributed by Smitha Dinesh Semwal

C#
// C# program to calculate
// minimum cost for candies
using System;

class GFG
{
    // Function to find minimum cost
    // to be incurred
    static int findMinCost(int []arr,
                           int n, int choc_cost)
    {

        // To reach first station, initial
        // chocolates required
        int choc_buy = arr[0];
        int curr_choc = 0;

        // Start traversing
        for (int i = 0; i < n - 1; i++)
        {
```

```
// Find no. of chocolates
// lose or gain
int choc = arr[i] - arr[i + 1];

// Add into curr_coc
curr_choc += choc;

// if no. of chocolates becomes
// negative that means we have
// to buy that no. of chocolates
if (curr_choc < 0)
{
    choc_buy += Math.Abs(curr_choc);
    curr_choc = 0;
}

// Return cost required
return choc_buy * choc_cost;
}

// Drivers code
public static void Main ()
{
    int []arr = {10, 6, 11, 4, 7, 1};
    int n = arr.Length;

    // Price of each candy
    int p = 5;

    Console.WriteLine( findMinCost(arr, n, p));
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to calculate
// minimum cost for candies

// Function to find minimum cost
// to be incurred
function findMinCost($arr, $n, $choc_cost)
{

    // To reach first station, initial
```

```
// chocolates required
$choc_buy = $arr[0];
$curr_choc = 0;

// Start traversing
for ($i = 0; $i < $n - 1; $i++) {

    // Find no. of chocolates
    // lose or gain
    $choc = $arr[$i] - $arr[$i + 1];

    // Add into curr_coc
    $curr_choc += $choc;

    // if no. of chocolates becomes
    // negative that means we have
    // to buy that no. of chocolates
    if ($curr_choc < 0) {
        $choc_buy += abs($curr_choc);
        $curr_choc = 0;
    }
}

// Return cost required
return $choc_buy * $choc_cost;
}

// Driver Code
$arr = array(10, 6, 11, 4, 7, 1);
$n = count($arr);

// Price of each candy
$p = 5;

echo findMinCost($arr, $n, $p);

// This code is contributed by Sam007
?>
```

Output:

55

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/minimum-total-cost-incurred-to-reach-the-last-station/>

Chapter 97

Modulo 10^{9+7} (1000000007)

Modulo 10^{9+7} (1000000007) - GeeksforGeeks

In most of the programming competitions, problems are required to answer the result in 10^{9+7} modulo. The reason behind this is to have problems for large integers so that only efficient algorithms can solve them in allowed limited time.

What is modulo operation:

The remainder obtained after the division operation on two operands is known as modulo operation. Operator for doing modulus operation is ‘%’. For ex: $a \% b = c$ which means when a is divided by b it gives the remainder c , $7\%2 = 1$, $17\%3 = 2$.

Why do we need modulo:

- The reason of taking Mod is to prevent integer overflows. The largest integer data type in C/C++ is *unsigned long long int* which is of 64 bit and can handle integer from 0 to $(2^{64} - 1)$. But in some problem where growth of the output is very high, this high range of unsigned long long may be insufficient.
Let in a 64 bit variable ‘A’, 2^{62} is stored and in another 64 bit variable ‘B’, 2^{63} is stored. When we multiply A and B, the system does not give a runtime error or exception. It just does some bogus computation and stores the bogus result because the bit size of result comes after multiplication overflows.
- In some of the problems to compute the result, modulo inverse are needed and this number helps very much because it is prime. Also this number should be large enough otherwise modular inverse techniques may fail in some situations.

Due to these reasons, problem setters ask to give the answer as a result of modulo of some number **N**.

There are certain criteria on which value of N depends:

1. It should just be large enough to fit in an largest integer data type i.e it makes sure that there is no over flow in result.

2. It should be a prime number because if we take mod of a number by Prime the result is generally spaced i.e. the results are very different results in comparison to mod the number by non-prime, that is why primes are generally used for mod.

10^{9+7} fulfills both the criteria. It is the first 10-digital prime number and fits in int data type as well. In fact any prime number less than 2^{30} will be fine in order to prevent possible overflows.

How modulo is used:

A few distributive properties of modulo are as follows:

1. $(a + b) \% c = ((a \% c) + (b \% c)) \% c$
2. $(a * b) \% c = ((a \% c) * (b \% c)) \% c$
3. $(a - b) \% c = ((a \% c) - (b \% c)) \% c$
4. $((a / b) \% c) = ((a \% c) / (b \% c)) \% c$

So, modulo is distributive over +, * and – but not over / [Please refer [Modular Division](#) for details]

NOTE: The result of $(a \% b)$ will always be less than b.

In case of computer programs, due to size of variable limitations we **perform modulo M at each intermediate stage** so that range overflow never occurs.

Example:

```
a = 145785635595363569532135132
b = 3151635135413512165131321321
c = 99987445522222200651351351
m = 1000000007
Print (a*b*c)%m.
```

Method 1:

```
First multiply all the number and then take modulo:
(a*b*c)%m = (459405448184212290893339835148809
515332440033400818566717735644307024625348601572) %
1000000007
a*b*c does not fit even in the unsigned long long
int due to which system drop some of its most
significant digits. Therefore, it gives wrong answer.
(a*b*c)%m = 798848767
```

Method 2:

Take modulo at each intermediate steps:

```
i = 1
i = (i*a) % m    // i = 508086243
i = (i*b) % m    // i = 144702857
i = (i*c) % m    // i = 798848767
i = 798848767
```

Method 2 always gives correct answer.

Function for finding factorial of large number using modulo but at different positions.

```
unsigned long long factorial(int n)
{
    const unsigned int M = 1000000007;
    unsigned long long f = 1;

    for (int i = 1; i <= n; i++)
        f = f * i; // WRONG APPROACH as
                    // f may exceed (2^64 - 1)

    return f % M;
}

unsigned long long factorial(int n)
{
    const unsigned int M = 1000000007;

    unsigned long long f = 1;
    for (int i = 1; i <= n; i++)
        f = (f*i) % M; // Now f never can
                        // exceed  $10^{9+7}$ 
    return f;
}
```

The same procedure can be followed for addition.

$(a + b + c) \% M$ is the same as $(((a + b) \% M) + c) \% M$.

Perform $\% M$ every time a number is added so as to avoid overflow.

Note: In most of the programming languages (like in C/C++) when you perform modular operation with negative numbers it gives negative result like $-5\%3 = -2$, but what the result comes after modular operation should be in the range 0 to $n-1$ means the $-5\%3 = 1$. So for this convert it into positive modular equivalent.

```
int mod(int a, int m)
{
    return (a%m + m) % m;
}
```

But the rules is different for division. To perform division in modulo arithmetic, we need to first understand the concept of modulo multiplicative inverse.

Modulo multiplicative Inverse(MMI):

The multiplicative inverse of a number y is z iff $(z * y) == 1$.

Dividing a number x by another number y is same as multiplying x with the multiplicative inverse of y .

$x / y == x * y^{-1} == x * z$ (where z is multiplicative inverse of y).

In normal arithmetic, the multiplicative inverse of y is a float value. Ex: Multiplicative inverse of 7 is 0.142..., of 3 is 0.333....

In mathematics, modular multiplicative inverse of an integer ‘ a ’ is an integer ‘ x ’ such that the product ax is congruent to 1 with respect to the modulus m .

$ax = 1 \pmod{m}$

The remainder after dividing ax by the integer m is 1.

Example:

If $M = 7$, the MMI of 4 is 2 as $(4 * 2) \% 7 == 1$,

If $M = 11$, the MMI of 7 is 8 as $(7 * 8) \% 11 == 1$,

If $M = 13$, the MMI of 7 is 2 as $(7 * 2) \% 13 == 1$.

Observe that the MMI of a number may be different for different M .

So, if we are performing modulo arithmetic in our program and we need the result of the operation x / y , we should NOT perform

$z = (x/y) \% M;$

instead we should perform

```
y_inv = findMMI(y, M);
z = (x * y_inv) % M;
```

Now one question remains.. How to find the MMI of a number n .

There exist two algorithms to find MMI of a number. First is the **Extended Euclidian algorithm** and the second using **Fermat's Little Theorem**.

You can find both the methods in the given link:

[Modular multiplicative inverse](#)

Finding modular multiplicative inverses also has practical applications in the field of cryptography, i.e. public-key cryptography and the [RSA Algorithm](#). A benefit for the computer implementation of these applications is that there exists a very fast algorithm (the extended Euclidean algorithm) that can be used for the calculation of modular multiplicative inverses.

References:

- <https://www.quora.com/What-exactly-is-print-it-modulo-10-9-+-7-in-competitive-programming-websites>
- <https://discuss.codechef.com/questions/4698/use-of-modulo-1000007-in-competitions>
- <https://codeaccepted.wordpress.com/2014/02/15/output-the-answer-modulo-109-7/>

Source

<https://www.geeksforgeeks.org/modulo-1097-1000000007/>

Chapter 98

Modulo power for large numbers represented as strings

Modulo power for large numbers represented as strings - GeeksforGeeks

Given two numbers **sa** and **sb** represented as strings, find $a^b \% \text{MOD}$ where MOD is $1e9 + 7$. The numbers a and b can contain upto 10^6 digits.

Examples:

Input : $sa = 2, sb = 3$

Output : 8

Output : 494234546

As **a** and **b** very large (may contain upto 10^6 digits each). So what we can do, we apply [Fermat's little theorem](#) and property of modulo to reduce **a** and **b**.

Reduce a:

As we know,

$$(ab) \% \text{ MOD} = ((a \% \text{ MOD})b) \% \text{ MOD}$$

Reduce by:

How to reduce b, We have already discuss in [Find \$\(a^b\)\%m\$](#) where 'b' is very large

Now finally we have both **a** and **b** are in range of $1 \leq a, b \leq 10^9 + 7$. Hence we can now use our modular exponentiation to calculate required answer.

```
// CPP program to find (a^b) % MOD where a and  
// b may be very large and represented as strings.
```

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long int
const ll MOD = 1e9 + 7;

// Returns modulo exponentiation for two numbers
// represented as long long int. It is used by
// powerStrings(). Its complexity is log(n)
ll powerLL(ll x, ll n)
{
    ll result = 1;
    while (n) {
        if (n & 1)
            result = result * x % MOD;
        n = n / 2;
        x = x * x % MOD;
    }
    return result;
}

// Returns modulo exponentiation for two numbers
// represented as strings. It is used by
// powerStrings()
ll powerStrings(string sa, string sb)
{
    // We convert strings to number

    ll a = 0, b = 0;

    // calculating a % MOD
    for (int i = 0; i < sa.length(); i++)
        a = (a * 10 + (sa[i] - '0')) % MOD;

    // calculating b % (MOD - 1)
    for (int i = 0; i < sb.length(); i++)
        b = (b * 10 + (sb[i] - '0')) % (MOD - 1);

    // Now a and b are long long int. We
    // calculate a^b using modulo exponentiation
    return powerLL(a, b);
}

int main()
{
    // As numbers are very large
    // that is it may contains upto
    // 10^6 digits. So, we use string.
```

```
string sa = "2", sb = "3";  
  
cout << powerStrings(sa, sb) << endl;  
return 0;  
}
```

Output:

8

Source

<https://www.geeksforgeeks.org/modulo-power-for-large-numbers-represented-as-strings/>

Chapter 99

Multi Source Shortest Path in Unweighted Graph

Multi Source Shortest Path in Unweighted Graph - GeeksforGeeks

Suppose there are n towns connected by m bidirectional roads. There are s towns among them with a police station. We want to find out the distance of each town from the nearest police station. If the town itself has one the distance is 0.

Example:

```
Input :  
Number of Vertices = 6  
Number of Edges = 9  
Towns with Police Station : 1, 5  
Edges:  
1 2  
1 6  
2 6  
2 3  
3 6  
5 4  
6 5  
3 4  
5 3
```

```
Output :  
1 0  
2 1  
3 1  
4 1  
5 0
```

6 1

Naive Approach: We can loop through the vertices and from each vertex run a BFS to find the closest town with police station from that vertex. This will take $O(V.E)$.

Naive approach implementation using BFS from each vertex:

```
// cpp program to demonstrate distance to
// nearest source problem using BFS
// from each vertex
#include <bits/stdc++.h>
using namespace std;
#define N 100000 + 1
#define inf 1000000

// This array stores the distances of the
// vertices from the nearest source
int dist[N];

// a hash array where source[i] = 1
// means vertex i is a source
int source[N];

// The BFS Queue
// The pairs are of the form (vertex, distance
// from current source)
deque<pair<int, int>> BFSQueue;

// visited array for remembering visited vertices
int visited[N];

// The BFS function
void BFS(vector<int> graph[], int start)
{
    // clearing the queue
    while (!BFSQueue.empty())
        BFSQueue.pop_back();

    // push_back starting vertices
    BFSQueue.push_back({ start, 0 });

    while (!BFSQueue.empty()) {

        int s = BFSQueue.front().first;
        int d = BFSQueue.front().second;
        visited[s] = 1;
        BFSQueue.pop_front();

        for (int v : graph[s]) {
            if (visited[v] == 0) {
                BFSQueue.push_back({ v, d + 1 });
                dist[v] = d + 1;
            }
        }
    }
}
```

```

// stop at the first source we reach during BFS
if (source[s] == 1) {
    dist[start] = d;
    return;
}

// Pushing the adjacent unvisited vertices
// with distance from current source = this
// vertex's distance + 1
for (int i = 0; i < graph[s].size(); i++)
    if (visited[graph[s][i]] == 0)
        BFSQueue.push_back({ graph[s][i], d + 1 });
}
}

// This function calculates the distance of each
// vertex from nearest source
void nearestTown(vector<int> graph[], int n,
                  int sources[], int S)
{

    // resetting the source hash array
    for (int i = 1; i <= n; i++)
        source[i] = 0;
    for (int i = 0; i <= S - 1; i++)
        source[sources[i]] = 1;

    // loop through all the vertices and run
    // a BFS from each vertex to find the distance
    // to nearest town from it
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            visited[j] = 0;
        BFS(graph, i);
    }

    // Printing the distances
    for (int i = 1; i <= n; i++)
        cout << i << " " << dist[i] << endl;
}

void addEdge(vector<int> graph[], int u, int v)
{
    graph[u].push_back(v);
    graph[v].push_back(u);
}

// Driver Code

```

```

int main()
{
    // Number of vertices
    int n = 6;

    vector<int> graph[n + 1];

    // Edges
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 6);
    addEdge(graph, 2, 6);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 6);
    addEdge(graph, 5, 4);
    addEdge(graph, 6, 5);
    addEdge(graph, 3, 4);
    addEdge(graph, 5, 3);

    // Sources
    int sources[] = { 1, 5 };

    int S = sizeof(sources) / sizeof(sources[0]);

    nearestTown(graph, n, sources, S);

    return 0;
}

```

Output:

```

1 0
2 1
3 1
4 1
5 0
6 1

```

Time Complexity: $O(V.E)$

Efficient Method A better method is to use the [Djikstra's algorithm](#) in a modified way. Let's consider one of the sources as the original source and the other sources to be vertices with 0 cost paths from the original source. Thus we push all the sources into the Djikstra Queue with distance = 0, and the rest of the vertices with distance = infinity. The minimum distance of each vertex from the original source now calculated using the Djikstra's Algorithm are now essentially the distances from the nearest source.

Explanation: The C++ implementation uses a [set of pairs](#) (distance from the source, vertex) sorted according to the distance from the source. Initially, the set contains the sources with distance = 0 and all the other vertices with distance = infinity.

On each step, we will go to the vertex with minimum distance(d) from source, i.e, the first element of the set (the source itself in the first step with distance = 0). We go through all it's adjacent vertices and if the distance of any vertex is $> d + 1$ we replace its entry in the set with the new distance. Then we remove the current vertex from the set. We continue this until the set is empty.

The idea is there cannot be a shorter path to the vertex at the front of the set than the current one since any other path will be a sum of a longer path (\geq it's length) and a non-negative path length (unless we are considering negative edges).

Since all the sources have a distance = 0, in the beginning, the adjacent non-source vertices will get a distance = 1. All vertices will get distance = distance from their nearest source.

Implementation of Efficient Approach:

```

// cpp program to demonstrate
// multi-source BFS
#include <bits/stdc++.h>
using namespace std;
#define N 100000 + 1
#define inf 1000000

// This array stores the distances of the vertices
// from the nearest source
int dist[N];

// This Set contains the vertices not yet visited in
// increasing order of distance from the nearest source
// calculated till now
set<pair<int, int> > Q;

// Util function for Multi-Source BFS
void multiSourceBFSUtil(vector<int> graph[], int s)
{
    set<pair<int, int> >::iterator it;
    int i;
    for (i = 0; i < graph[s].size(); i++) {
        int v = graph[s][i];
        if (dist[s] + 1 < dist[v]) {

            // If a shorter path to a vertex is
            // found than the currently stored
            // distance replace it in the Q
            it = Q.find({ dist[v], v });
            Q.erase(it);
            dist[v] = dist[s] + 1;
            Q.insert({ dist[v], v });
        }
    }

    // Stop when the Q is empty -> All
}

```

```

// vertices have been visited. And we only
// visit a vertex when we are sure that a
// shorter path to that vertex is not
// possible
if (Q.size() == 0)
    return;

// Go to the first vertex in Q
// and remove it from the Q
it = Q.begin();
int next = it->second;
Q.erase(it);

multiSourceBFSUtil(graph, next);
}

// This function calculates the distance of
// each vertex from nearest source
void multiSourceBFS(vector<int> graph[], int n,
                    int sources[], int S)
{
    // a hash array where source[i] = 1
    // means vertex i is a source
    int source[n + 1];

    for (int i = 1; i <= n; i++)
        source[i] = 0;
    for (int i = 0; i <= S - 1; i++)
        source[sources[i]] = 1;

    for (int i = 1; i <= n; i++) {
        if (source[i]) {
            dist[i] = 0;
            Q.insert({ 0, i });
        }
        else {
            dist[i] = inf;
            Q.insert({ inf, i });
        }
    }
}

set<pair<int, int> >::iterator itr;

// Get the vertex with lowest distance,
itr = Q.begin();

// currently one of the souces with distance = 0
int start = itr->second;

```

```
multiSourceBFSUtil(graph, start);

// Printing the distances
for (int i = 1; i <= n; i++)
    cout << i << " " << dist[i] << endl;
}

void addEdge(vector<int> graph[], int u, int v)
{
    graph[u].push_back(v);
    graph[v].push_back(u);
}

// Driver Code
int main()
{
    // Number of vertices
    int n = 6;

    vector<int> graph[n + 1];

    // Edges
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 6);
    addEdge(graph, 2, 6);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 6);
    addEdge(graph, 5, 4);
    addEdge(graph, 6, 5);
    addEdge(graph, 3, 4);
    addEdge(graph, 5, 3);

    // Sources
    int sources[] = { 1, 5 };

    int S = sizeof(sources) / sizeof(sources[0]);

    multiSourceBFS(graph, n, sources, S);

    return 0;
}
```

Output:

```
1 0
2 1
```

3 1
4 1
5 0
6 1

Time Complexity: $O(E \cdot \log V)$

Source

<https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/>

Chapter 100

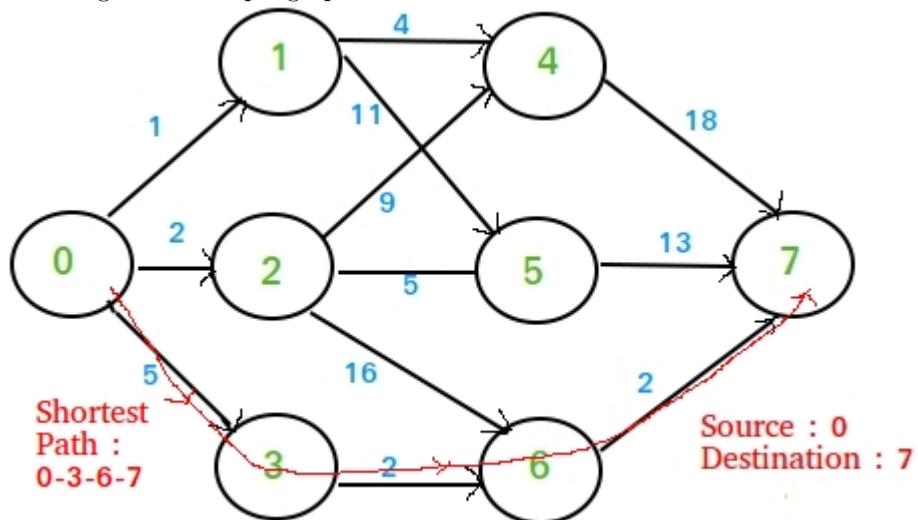
Multistage Graph (Shortest Path)

Multistage Graph (Shortest Path) - GeeksforGeeks

A **Multistage graph** is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only (In other words there is no edge between vertices of same stage and from a vertex of current stage to previous stage).

We are given a multistage graph, a source and a destination, we need to find shortest path from source to destination. By convention, we consider source at stage 1 and destination as last stage.

Following is an example graph we will consider in this article :-



Now there are various strategies we can apply :-

- The **Brute force** method of finding all possible paths between Source and Destination and then finding the minimum. That's the WORST possible strategy.
- **Dijkstra's Algorithm** of Single Source shortest paths. This method will find shortest paths from source to all other nodes which is not required in this case. So it will take a lot of time and it doesn't even use the SPECIAL feature that this MULTI-STAGE graph has.
- **Simple Greedy Method** – At each node, choose the shortest outgoing path. If we apply this approach to the example graph given above we get the solution as $1 + 4 + 18 = 23$. But a quick look at the graph will show much shorter paths available than 23. So the greedy method fails !
- The best option is Dynamic Programming. So we need to find **Optimal Substructure, Recursive Equations and Overlapping Sub-problems**.

Optimal Substructure and Recursive Equation :-

We define the notation :- $M(x, y)$ as the minimum cost to T (target node) from Stage x , Node y .

Shortest distance from stage 1, node 0 to destination, i.e., 7 is $M(1, 0)$.

```
// From 0, we can go to 1 or 2 or 3 to
// reach 7.
M(1, 0) = min(1 + M(2, 1),
                2 + M(2, 2),
                5 + M(2, 3))
```

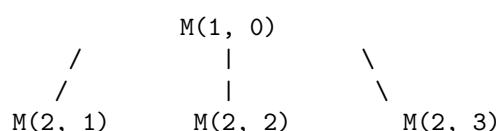
This means that our problem of $0 \rightarrow 7$ is now sub-divided into 3 sub-problems :-

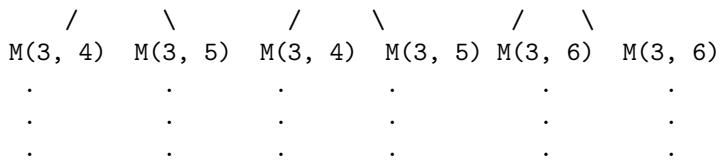
So if we have total ' n ' stages and target as T , then the stopping condition will be :-
 $M(n-1, i) = i \rightarrow T + M(n, T) = i \rightarrow T$

Recursion Tree and Overlapping Sub-Problems:-

So, the hierarchy of $M(x, y)$ evaluations will look something like this :-

In $M(i, j)$, i is stage number and j is node number





So, here we have drawn a very small part of the Recursion Tree and we can already see Overlapping Sub-Problems. We can largely reduce the number of $M(x, y)$ evaluations using Dynamic Programming.

Implementation details:

The below implementation assumes that nodes are numbered from 0 to $N-1$ from first stage (source) to last stage (destination). We also assume that the input graph is multistage.

```

// CPP program to find shortest distance
// in a multistage graph.
#include<bits/stdc++.h>
using namespace std;

#define N 8
#define INF INT_MAX

// Returns shortest distance from 0 to
// N-1.
int shortestDist(int graph[N][N]) {

    // dist[i] is going to store shortest
    // distance from node i to node N-1.
    int dist[N];

    dist[N-1] = 0;

    // Calculating shortest path for
    // rest of the nodes
    for (int i = N-2 ; i >= 0 ; i--)
    {

        // Initialize distance from i to
        // destination (N-1)
        dist[i] = INF;

        // Check all nodes of next stages
        // to find shortest distance from
        // i to N-1.
        for (int j = i ; j < N ; j++)
        {
            // Reject if no edge exists
            if (graph[i][j] == INF)

```

```
        continue;

        // We apply recursive equation to
        // distance to target through j.
        // and compare with minimum distance
        // so far.
        dist[i] = min(dist[i], graph[i][j] +
                      dist[j]);
    }
}

return dist[0];
}

// Driver code
int main()
{
    // Graph stored in the form of an
    // adjacency Matrix
    int graph[N][N] =
    {{INF, 1, 2, 5, INF, INF, INF},
     {INF, INF, INF, INF, 4, 11, INF},
     {INF, INF, INF, INF, 9, 5, 16, INF},
     {INF, INF, INF, INF, INF, INF, 2, INF},
     {INF, INF, INF, INF, INF, INF, INF, 18},
     {INF, INF, INF, INF, INF, INF, INF, 13},
     {INF, INF, INF, INF, INF, INF, INF, 2}};

    cout << shortestDist(graph);
    return 0;
}
```

Output:

9

Time Complexity : $O(n^2)$

Source

<https://www.geeksforgeeks.org/multistage-graph-shortest-path/>

Chapter 101

NZEC error in Python

NZEC error in Python - GeeksforGeeks

While coding in various competitive sites, many people must have have encountered NZEC error. NZEC (non zero exit code) as the name suggests occurs when your code is failed to return 0. When a code returns 0 it means it is successfully executed otherwise it will return some other number depending on the type of error.

When the program ends and it is supposed to return “0” to indicate if finished fine and not able to do so it cause NZEC. Of course there are more cases associated with NZEC.

Why does NZEC occur?(one example)

In python, generally multiple inputs are separated by commas and we read them using `input()` or `int(input())`, but most of the online coding platforms while testing gives input separated by space and in those cases `int(input())` is not able to read the input properly and shows error like NZEC.

How to resolve?

For Example, Think of a simple program where you have to read 2 integer and print them(in input file both integer are in same line). Suppose you have two integers as shown below:

23 45

Instead of using :

```
n = int(input())
k = int(input())
```

Use:

```
n, k = raw_input().split(" ")
n = int(n)
k = int(k)
```

to delimit input by white spaces.

Wrong code

```
n = int(input())
k = int(input())
print n," ",k
```

Input:

2 3

When you run the above code in [IDE](#) with above input you will get error:-

```
Traceback (most recent call last):
  File "b712edd81d4a972de2a9189fac8a83ed.py", line 1, in <module>
    n = int(input())
  File "", line 1
    2 3
               ^
SyntaxError: unexpected EOF while parsing
```

The above code will work fine when the input are in 2 two different lines. You can test yourself. To overcome this problem you need to use split.

Correct code

```
n, k = raw_input().split(" ")
n = int(n)
k = int(k)
print n," ",k
```

Input:

7 3

Output:

7 3

Some prominent reasons for NZEC error

1. Infinite Recursion or if you have run out of stack memory.
2. Input and output both are NOT exactly same as the test cases.
3. As the online platforms, test your program using a computer code which matches your output with the specified outputs exactly.
4. This type of error is also shown when your program is performing basic programming mistakes like dividing by 0.
5. Check for the values of your variables, they can be vulnerable to integer flow.

There could be some other reasons also for occurrence NZEC error, i have listed the frequent ones.

Source

<https://www.geeksforgeeks.org/nzec-error-python/>

Chapter 102

Nth non-Square number

Nth non-Square number - GeeksforGeeks

Given **n**, find the **nth** number which is not a perfect square among natural numbers (1, 2, 3, 4, 5, 6, ...)

Examples:

```
Input : 3
Output : 5
First three non-square numbers are 2, 3
and 5
```

```
Input : 5
Output : 7
```

```
Input : 16
Output : 20
```

Looking at the problem statement we can come up to a straight-forward brute-force approach. We can start from $n = 1$, and start to check if each of them is a perfect square or not. So we can come up to the nth non-square number.

However, the above approach is very slow as it searches for each in every number smaller than the target each time.

We can observe that the series under consideration is 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 17,

We can come upto the constant time formula for the nth number in this sequence, by inspection.

$$a_n = \frac{n(n+1)}{2} + 1$$

The correctness of the formula can be proved by the Principle of Mathematical Induction.

The implementation of the above formula is given below.

C++

```
// CPP program to find n-th non-square number.
#include <bits/stdc++.h>

using namespace std;

// function to find the nth Non-Square Number
int findNthNonSquare(int n)
{
    // conversion from int to long double is
    // necessary in order to preserve decimal
    // places after square root.
    long double x = (long double)n;

    // calculating the result
    long double ans = x + floor(0.5 + sqrt(x));

    return (int)ans;
}

// Driver code
int main()
{
    // initializing the term number
    int n = 16;

    // Print the result
    cout << "The " << n << "th Non-Square number is ";
    cout << findNthNonSquare(n);

    return 0;
}
```

Java

```
// Java program to find
// n-th non-square number.
import java.io.*;
import java.util.*;
import java.lang.*;

class GFG
{

    // function to find the
```

```
// nth Non-Square Number
static int findNthNonSquare(int n)
{
    // conversion from int to
    // long double is necessary
    // in order to preserve decimal
    // places after square root.
    double x = (double)n;

    // calculating the result
    double ans = x + Math.floor(0.5 +
        Math.sqrt(x));

    return (int)ans;
}

// Driver code
public static void main(String[] args)
{
    // initializing
    // the term number
    int n = 16;

    // Print the result
    System.out.print("The " + n +
                      "th Non-Square number is ");
    System.out.print(findNthNonSquare(n));
}
```

C#

```
// C# program to find
// n-th non-square number.
using System;

class GFG
{

    // function to find the
    // nth Non-Square Number
    static int findNthNonSquare(int n)
    {
        // conversion from int
        // to long double is
        // necessary in order
        // to preserve decimal
        // places after square
```

```
// root.  
double x = (double)n;  
  
// calculating the result  
double ans = x + Math.Floor(0.5 +  
    Math.Sqrt(x));  
  
return (int)ans;  
}  
  
// Driver code  
public static void Main()  
{  
    // initializing  
    // the term number  
    int n = 16;  
  
    // Print the result  
    Console.Write("The " + n +  
        "th Non-Square " +  
        "number is ");  
    Console.Write(findNthNonSquare(n));  
}  
}  
  
// This code is contributed  
// by anuj_67.
```

PHP

Output:

The 16th Non-Square number is 20

Time Complexity $\mathcal{O}(1)$.

Space Complexity $\mathcal{O}(1)$

Improved By : [vt_m](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/nth-non-square-number/>

Chapter 103

Number of Larger Elements on right side in a string

Number of Larger Elements on right side in a string - GeeksforGeeks

Given a string, find count of number of larger alphabets for every character of the string.

Examples:

```
Input : str = "abcd"
Output : 3 2 1 0
There are 3 greater characters on right of 'a',
2 greater characters on right of 'b', 1 greater
character on right of 'c' and 0 greater characters
on right of 'd'.
```

```
Input : geeks
Output : 2 2 2 1 0
```

A **naive approach** is to use two for loops. First will keep track of each alphabet in string and second loop will be used to find no of larger alphabet according to ASCII values.

C++

```
// CPP program to find counts of right greater
// characters for every character.
#include <bits/stdc++.h>
using namespace std;

void printGreaterCount(string str)
```

```
{  
    int len = str.length(), right[len] = { 0 };  
    for (int i = 0; i < len; i++)  
        for (int j = i + 1; j < len; j++)  
            if (str[i] < str[j])  
                right[i]++;
  
    for (int i = 0; i < len; i++)  
        cout << right[i] << " ";
}  
  
// Driver code  
int main()  
{  
    string str = "abcd";  
    printGreaterCount(str);  
    return 0;
}
```

PHP

```
<?php  
// PHP program to find counts  
// of right greater characters  
// for every character.  
  
function printGreaterCount($str)  
{  
    $len = strlen($str);  
    $right = array_fill(0, $len, 0);  
    for ($i = 0; $i < $len; $i++)  
    {  
        for ($j = $i + 1; $j < $len; $j++)  
            if ($str[$i] < $str[$j])  
                $right[$i]++;
    }
  
    for ($i = 0; $i < $len; $i++)  
        echo $right[$i] . " ";
}  
  
// Driver code  
$str = 'abcd';  
printGreaterCount($str);
  
// This code is contributed  
// by Abhinav96  
?>
```

Output:

3 2 1 0

Time Complexity : $O(N * N)$

An **efficient approach** is to traverse the string from right and keep track of counts of characters from right side. For every character that we traverse from right, we increment its count in count array and add counts of all greater characters to answer for this character.

C++

```
// C++ program to count greater characters on right
// side of every character.
#include <bits/stdc++.h>
using namespace std;
#define MAX_CHAR 26

void printGreaterCount(string str)
{
    int len = str.length();

    // Arrays to store result and character counts.
    int ans[len] = { 0 }, count[MAX_CHAR] = { 0 };

    // start from right side of string
    for (int i = len - 1; i >= 0; i--)
    {
        count[str[i] - 'a']++;
        for (int j = str[i] - 'a' + 1; j < MAX_CHAR; j++)
            ans[i] += count[j];
    }

    for (int i = 0; i < len; i++)
        cout << ans[i] << " ";
}

// Driver code
int main()
{
    string str = "abcd";
    printGreaterCount(str);
    return 0;
}
```

Output:

3 2 1 0

Time Complexity : $O(N)$

Source

<https://www.geeksforgeeks.org/number-of-larger-elements-on-right-side-in-a-string/>

Chapter 104

Number of Transpositions in a Permutation

Number of Transpositions in a Permutation - GeeksforGeeks

Permutation A permutation is an arrangement of elements. A permutation of n elements can be represented by an arrangement of the numbers $1, 2, \dots, n$ in some order. e.g. $5, 1, 4, 2, 3$.

Cycle notation A permutation can be represented as a composition of permutation cycles. A permutation cycle is a set of elements in a permutation which trade places with one another.

For e.g.

$$P = \{ 5, 1, 4, 2, 3 \}:$$

Here, 5 goes to 1, 1 goes to 2 and so on (according to their indices position):

$$5 \rightarrow 1$$

$$1 \rightarrow 2$$

$$2 \rightarrow 4$$

$$4 \rightarrow 3$$

$$3 \rightarrow 5$$

Thus it can be represented as a single cycle: $(5, 1, 2, 4, 3)$.

Now consider the permutation: $\{ 5, 1, 4, 3, 2 \}$. Here

$$5 \rightarrow 1$$

$$1 \rightarrow 2$$

$2 \rightarrow 5$ this closes 1 cycle.

The other cycle is

$$4 \rightarrow 3$$

$$3 \rightarrow 4$$

In cycle notation it will be represented as $(5, 1, 2) (4, 3)$.

Transpositions

Now all cycles can be decomposed into a composition of 2 cycles (transpositions). The

number of transpositions in a permutation is important as it gives the minimum number of 2 element swaps required to get this particular arrangement from the identity arrangement: 1, 2, 3, ... n. The parity of the number of such 2 cycles represents whether the permutation is even or odd.

For e.g.

The cycle (5, 1, 2, 4, 3) can be written as (5, 3)(5, 4)(5, 2)(5, 1). 4 transpositions (even).

Similarly,

(5, 1, 2) -> (5, 2)(5, 1)

(5, 1, 2)(4, 3) -> (5, 2)(5, 1)(4, 3). 3 transpositions (odd).

It is clear from the examples that the **number of transpositions from a cycle = length of the cycle – 1**.

Problem

Given a permutation of n numbers $P_1, P_2, P_3, \dots, P_n$. Calculate the number of transpositions in it.

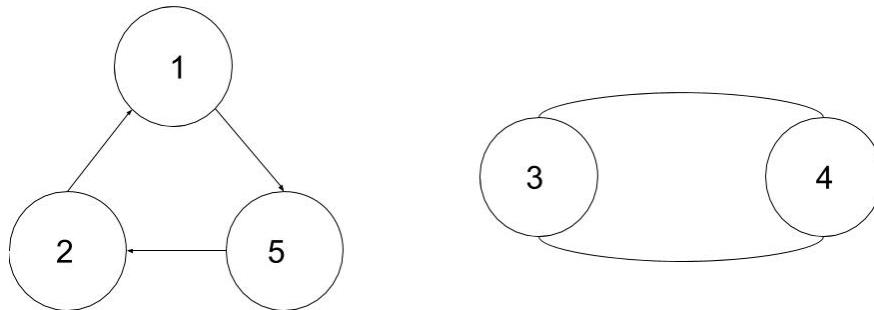
Example:

Input: 5 1 4 3 2

Output: 3

Approach: The permutation can be easily represented as a directed graph where the number of connected components gives the number of cycles. And (the size of each component – 1) gives the number of transpositions from that cycle.

Example Permutation : { 5, 1, 4, 3, 2 } -> (5, 1, 2)(4, 3)



Transpositions = 2

Total = 3

Transpositions = 1

Below is the implementation of above approach.

C++

```
// CPP Program to find the number of
// transpositions in a permutation
#include <bits/stdc++.h>
using namespace std;

#define N 1000001

int visited[N];

// This array stores which element goes to which position
int goesTo[N];

// For eg. in { 5, 1, 4, 3, 2 }
// goesTo[1] = 2
// goesTo[2] = 5
// goesTo[3] = 4
// goesTo[4] = 3
// goesTo[5] = 1

// This function returns the size of a component cycle
int dfs(int i)
{
    // If it is already visited
    if (visited[i] == 1)
        return 0;

    visited[i] = 1;
    int x = dfs(goesTo[i]);
    return (x + 1);
}

// This function returns the number
// of transpositions in the permutation
int noOfTranspositions(int P[], int n)
{
    // Initializing visited[] array
    for (int i = 1; i <= n; i++)
        visited[i] = 0;

    // building the goesTo[] array
    for (int i = 0; i < n; i++)
        goesTo[P[i]] = i + 1;

    int transpositions = 0;

    for (int i = 1; i <= n; i++) {
        if (visited[i] == 0) {
            int ans = dfs(i);
```

```
        transpositions += ans - 1;
    }
}
return transpositions;
}

// Driver Code
int main()
{
    int permutation[] = { 5, 1, 4, 3, 2 };
    int n = sizeof(permutation) / sizeof(permutation[0]);

    cout << noOfTranspositions(permutation, n);
    return 0;
}
```

Java

```
// Java Program to find the number of
// transpositions in a permutation
import java.io.*;

class GFG {

    static int N = 1000001;

    static int visited[] = new int[N];

    // This array stores which element
    // goes to which position
    static int goesTo[] = new int[N];

    // For eg. in { 5, 1, 4, 3, 2 }
    // goesTo[1] = 2
    // goesTo[2] = 5
    // goesTo[3] = 4
    // goesTo[4] = 3
    // goesTo[5] = 1

    // This function returns the size
    // of a component cycle
    static int dfs(int i)
    {

        // If it is already visited
        if (visited[i] == 1)
            return 0;
```

```

    visited[i] = 1;
    int x = dfs(goesTo[i]);
    return (x + 1);
}

// This function returns the number
// of transpositions in the
// permutation
static int noOfTranspositions(int P[],
                               int n)
{
    // Initializing visited[] array
    for (int i = 1; i <= n; i++)
        visited[i] = 0;

    // building the goesTo[] array
    for (int i = 0; i < n; i++)
        goesTo[P[i]] = i + 1;

    int transpositions = 0;

    for (int i = 1; i <= n; i++) {
        if (visited[i] == 0) {
            int ans = dfs(i);
            transpositions += ans - 1;
        }
    }
    return transpositions;
}

// Driver Code
public static void main (String[] args)
{
    int permutation[] = { 5, 1, 4, 3, 2 };
    int n = permutation.length ;

    System.out.println(
        noOfTranspositions(permutation, n));
}
}

// This code is contributed by anuj_67.

```

C#

```

// C# Program to find the number of
// transpositions in a permutation
using System;

```

```
class GFG {

    static int N = 1000001;

    static int []visited = new int[N];

    // This array stores which element
    // goes to which position
    static int []goesTo= new int[N];

    // For eg. in { 5, 1, 4, 3, 2 }
    // goesTo[1] = 2
    // goesTo[2] = 5
    // goesTo[3] = 4
    // goesTo[4] = 3
    // goesTo[5] = 1

    // This function returns the size
    // of a component cycle
    static int dfs(int i)
    {

        // If it is already visited
        if (visited[i] == 1)
            return 0;

        visited[i] = 1;
        int x = dfs(goesTo[i]);
        return (x + 1);
    }

    // This functio returns the number
    // of transpositions in the
    // permutation
    static int noOfTranspositions(int []P,
                                  int n)
    {
        // Initializing visited[] array
        for (int i = 1; i <= n; i++)
            visited[i] = 0;

        // building the goesTo[] array
        for (int i = 0; i < n; i++)
            goesTo[P[i]] = i + 1;

        int transpositions = 0;
```

```
for (int i = 1; i <= n; i++) {
    if (visited[i] == 0) {
        int ans = dfs(i);
        transpositions += ans - 1;
    }
}
return transpositions;
}

// Driver Code
public static void Main ()
{
    int []permutation = { 5, 1, 4, 3, 2 };
    int n = permutation.Length ;

    Console.WriteLine(
        noOfTranspositions(permutation, n));
}
}

// This code is contributed by anuj_67.
```

Output:

3

Time Complexity : $O(n)$

Auxiliary space : $O(n)$

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/number-of-transpositions-in-a-permutation/>

Chapter 105

Number of days until all chocolates become unhealthy

Number of days until all chocolates become unhealthy - GeeksforGeeks

Pablo has square chocolate Box of size $n \times n$ in which a variety of healthy chocolates are present denoted by 'H' initially but he finds out that some of the chocolates are rotten and are unhealthy denoted by 'U'. In one day the rotten chocolates make all its neighbouring chocolates as unhealthy. This goes on and on until all chocolates present in the chocolate box becomes Unhealthy to eat. Find out the number of days in which the whole chocolate box becomes Unhealthy.

(Note : It is guaranteed that atleast one of the chocolate is Unhealthy)

Examples:

```
Input : n = 3
        H H H
        H U H
        H H H
Output : 1
Only 1 day is required to turn all the
chocolates unhealthy in the chocolate box.
```

```
Input : n = 4
        H H H U
        H H H H
        H U H H
        H H H H
Output : 2
Explanation:
In first day chocolate at (0, 0), (0, 1),
(2, 3), (3, 3) will remain healthy and in
```

the second day all the chocolates will become unhealthy.

Asked in Amazon, Accolite, Arcesium.

Brute Force Approach:

Initialize a flag = 1. Use a while loop, inside that while search for an H (searching requires $O(n^2)$ time complexity if we are unable to find a H in the 2-D character array stop incrementing the day counter and set flag as 0 to break the loop.

Below is the implementation of above approach:

```
// CPP program to find number of days before
// all chocolates become unhealthy.
#include <bits/stdc++.h>
using namespace std;

// Validates out of bounds indexing
bool isValid(int i, int j, int n)
{
    if (i < 0 || j < 0 || i >= n || j >= n)
        return false;
    return true;
}

// function for returning number of days
int numdays(char arr[][][4], int n)
{
    int numdays = 0;

    while (true)
    {
        // Traverse matrix to look for unhealthy
        // chocolates and mark their neighbors.
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (arr[i][j] == 'U')

                    if (isValid(i - 1, j - 1, n) &&
                        arr[i - 1][j - 1] == 'H')
                        arr[i - 1][j - 1] = 'V';

                if (isValid(i - 1, j, n) &&
                    arr[i - 1][j] == 'H')
                    arr[i - 1][j] = 'V';
            }
        }
    }
}
```

```

        if (isValid(i - 1, j + 1, n) &&
            arr[i - 1][j + 1] == 'H')
            arr[i - 1][j + 1] = 'V';

        if (isValid(i, j - 1, n) &&
            arr[i][j - 1] == 'H')
            arr[i][j - 1] = 'V';

        if (isValid(i, j + 1, n) &&
            arr[i][j + 1] == 'H')
            arr[i][j + 1] = 'V';

        if (isValid(i + 1, j - 1, n) &&
            arr[i + 1][j - 1] == 'H')
            arr[i + 1][j - 1] = 'V';

        if (isValid(i + 1, j, n) &&
            arr[i + 1][j] == 'H')
            arr[i + 1][j] = 'V';

        if (isValid(i + 1, j + 1, n) &&
            arr[i + 1][j + 1] == 'H')
            arr[i + 1][j + 1] = 'V';
    }

    /*Here we are assigning the neighbours of U
    with the character V because we don't want
    these neighbours to be counted in that
    particular day. If we do not do so, in the
    next iteration that neighbour will also get
    counted which was supposed to be counted in
    the next day. */
}

}

// Mark chocolates unhealthy which are made
// unhealthy in current day.
bool Hflag = false;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (arr[i][j] == 'V')
        {
            arr[i][j] = 'U';
            Hflag = true;
        }
    }
}

```

```
}

// Check if there was any chocolate
// marked unhealthy in current day
if (Hflag)
    numdays++;
else
    break;
}
return numdays;
}

// Driver function
int main()
{
    int n = 4;
    char arr[4][4] = { 'H', 'H', 'H', 'U',
                      'H', 'H', 'H', 'H',
                      'H', 'U', 'H', 'H',
                      'H', 'H', 'H', 'H'
                    };
    int ans = numdays(arr, n);
    cout << "number of days taken : "
        << ans << "\n";
    return 0;
}
```

Output:

```
number of days taken : 2
```

Efficient Approach (Uses BFS)

In this approach, declare a queue which inputs pairs which corresponds to the index of the unhealthy chocolates and then as soon as the index (-1, -1) is reached we increment the numdays counter. This Solution is basically based on calculating levels in level order traversal (Iterative version) of a binary tree in which we push the initial indexes of the unhealthy chocolates instead of root node and incrementing numdays instead of level counter as soon as the index (-1, -1) is reached instead of NULL. As soon as the counter of the flag reaches 2 we break the loop denoting that queue has encountered two consecutive (-1, -1) pair.

Below is the implementation of above approach:

```
// CPP program using Efficient approach
// to find number of days
#include <bits/stdc++.h>
using namespace std;
```

```
// Validates out of bounds indexing
bool isValid(int i, int j, int n)
{
    if (i < 0 || j < 0 || i >= n || j >= n)
        return false;
    return true;
}

// function for returning number of days
int numdays(char arr[][][4], int n)
{
    int numdays = 0;
    int i, j;
    queue<pair<int, int> > q;

    // Initializing queue with initial
    // positions of unhealthy chocolates
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            if (arr[i][j] == 'U')
                q.push(make_pair(i, j));
        }

    q.push(make_pair(-1, -1));

    // (-1, -1) is used as a checkpoint
    // to count the number of days
    pair<int, int> temp;

    // temporary pair to store the indexes
    int flag = 0;
    while (!q.empty()) {
        temp = q.front();
        i = temp.first;
        j = temp.second;
        q.pop();
        if (i == -1 && j == -1) {
            flag++;
            q.push(make_pair(-1, -1));

            // pushing the respective
            // checkpoint
            if (flag == 2)
                break;
            numdays++;
        }
        else {
            flag = 0;
```

```
if (isValid(i - 1, j - 1, n) &&
    arr[i - 1][j - 1] == 'H') {
    q.push(make_pair(i - 1, j - 1));
    arr[i - 1][j - 1] = 'U';
}

if (isValid(i - 1, j, n) &&
    arr[i - 1][j] == 'H') {
    q.push(make_pair(i - 1, j));
    arr[i - 1][j] = 'U';
}

if (isValid(i - 1, j + 1, n) &&
    arr[i - 1][j + 1] == 'H') {
    q.push(make_pair(i - 1, j + 1));
    arr[i - 1][j + 1] = 'U';
}

if (isValid(i, j - 1, n) &&
    arr[i][j - 1] == 'H') {
    q.push(make_pair(i, j - 1));
    arr[i][j - 1] = 'U';
}

if (isValid(i, j + 1, n) &&
    arr[i][j + 1] == 'H') {
    q.push(make_pair(i, j + 1));
    arr[i][j + 1] = 'U';
}

if (isValid(i + 1, j - 1, n) &&
    arr[i + 1][j - 1] == 'H') {
    q.push(make_pair(i + 1, j - 1));
    arr[i + 1][j - 1] = 'U';
}

if (isValid(i + 1, j, n) &&
    arr[i + 1][j] == 'H') {
    q.push(make_pair(i + 1, j));
    arr[i + 1][j] = 'U';
}

if (isValid(i + 1, j + 1, n) &&
    arr[i + 1][j + 1] == 'H') {
    q.push(make_pair(i + 1, j + 1));
    arr[i + 1][j + 1] = 'U';
}
}
```

```
}

    return numdays - 1;
}

// Driver function
int main()
{
    int n = 4;
    char arr[4][4] = { 'H', 'H', 'H', 'U',
                      'H', 'H', 'H', 'H',
                      'H', 'H', 'U', 'H',
                      'H', 'H', 'H', 'H' };
    int ans = numdays(arr, n);
    cout << "number of days taken : "
        << ans << "\n";
    return 0;
}
```

Output:

```
number of days taken : 2
```

Source

<https://www.geeksforgeeks.org/number-of-days-until-all-chocolates-become-unhealthy/>

Chapter 106

Number of digits in N factorial to the power N

Number of digits in N factorial to the power N - GeeksforGeeks

Given a positive integer N, we have to find the total number of digits in the factorial of N

raised to the power N, i.e., $(N!)^N$

Examples:

Input: 4

Output: 6

Explanations: $= = 331776$.

Total number of digits in 331776 is 6.

Input: 5

Output: 11

Explanations: $= = 24883200000$

Total number of digits in 24883200000 is 11.

Input: 2

Output: 1

Input: 1000

Output: 2567605

The idea of the solution is described below.

Brute force method: By brute force, we can simply compute $N!$ in $O(N)$ time and then multiply it n times but that will be a very slow method and would

exceed both time and space because $(N!)^N$ would be a huge number.

Efficient method:

Let's look at it more closely. We can break $(N!)^N$ into simpler terms which are easy to compute. By taking common logarithm,

$$\text{we get } \frac{N!}{N!} \approx \log_{10}(N!) + \log_{10}(N!) + \dots + \log_{10}(N!).$$

and we know, $\log_{10}(1) + \log_{10}(2) + \log_{10}(3) + \dots + \log_{10}(N)$.
Therefore we can reduce further.

$$\frac{N!}{N!} \approx \log_{10}(1) + \log_{10}(2) + \dots + \log_{10}(N)$$

$$= N \cdot \left(\log_{10}(1) + \log_{10}(2) + \dots + \log_{10}\left(\frac{N}{2}\right) + \log_{10}\left(\frac{N}{2}+1\right) + \dots + \log_{10}(N) \right)$$

Now we can compute the answer easily in $O(N)$ time and without space limit exceeding.

So why is a valid answer to the problem?

We know that the total number of digits in a number N whose base is 10 can be easily found by taking $\text{ceil}(\log_{10}(N))$. That is exactly done in the approach described above.

The code implementation of the above idea is given below.

C++

```
// CPP program to find count of digits in N
// factorial raised to N
#include <bits/stdc++.h>
using namespace std;

int countDigits(int n)
{
    // we take sum of logarithms as explained
    // in the approach
    double ans = 0;
    for (int i = 1; i <= n; i++)
        ans += log10(i);

    // multiply the result with n
    ans = ans * n;
    return 1 + floor(ans);
}
```

```
int main()
{
    int n = 4;
    cout << countDigits(n) << "\n";
    return 0;
}
```

Java

```
// Java program to find
// count of digits in N
// factorial raised to N
import java.io.*;

class GFG
{
static int countDigits(int n)
{
    // we take sum of logarithms
    // as explained in the approach
    double ans = 0;
    for (int i = 1; i <= n; i++)
        ans += Math.log10(i);

    // multiply the
    // result with n
    ans = ans * n;
    return 1 + (int)Math.floor(ans);
}

// Driver Code
public static void main (String[] args)
{
    int n = 4;
    System.out.println(
        countDigits(n) + "\n");
}
}

// This code is contributed
// by anuj_67.
```

C#

```
// C# program to find
// count of digits in N
```

```
// factorial raised to N
using System;

class GFG
{
    static int countDigits(int n)
    {
        // we take sum of logarithms
        // as explained in the approach
        double ans = 0;
        for (int i = 1; i <= n; i++)
            ans += Math.Log10(i);

        // multiply the
        // result with n
        ans = ans * n;
        return 1 + (int)Math.Floor(ans);
    }

    // Driver Code
    public static void Main ()
    {
        int n = 4;
        Console.WriteLine(
            countDigits(n) + "\n");
    }
}

// This code is contributed
// by anuj_67.
```

Output:

6

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/number-digits-n-factorial-power-n/>

Chapter 107

Number of digits in the nth number made of given four digits

Number of digits in the nth number made of given four digits - GeeksforGeeks

Find the number of digits in the nth number constructed by using 6, 1, 4 and 9 as the only digits in the ascending order.

First few numbers constructed by using only 6, 1, 4 and 9 as digits in the ascending order would be: 1, 6, 4,
9, 11, 14, 16, 19, 41, 44, 46, 49, 61, 64, 66, 69, 91, 94, 96, 99, 111, 114, 116, 119 and so on.

Examples:

```
Input : 6
Output : 2
6th digit of the series is 14 which has 2 digits.
```

```
Input : 21
Output : 3
21st digit of the series is 111 which has 3 digits.
```

Simple Approach: This is a brute force approach.

1. Initialize a number to 1 and a counter to 0.
2. Check if the initialized number has only 6, 1, 4 or 9 as it's digits.
3. If it has only the mentioned digits then increase the counter by 1.
4. Increase the number and repeat the above steps until the counter is less than n.

Note: The value of n could be large and hence this approach can't work as it's not time efficient.

Efficient Approach: You can calculate the number of k digit numbers in O (1) time and they will be always be power of 4, for instance number of 1 digit numbers in the series would be 4, number of 2 digit numbers in the series would be 16 and so on.

1. Count all subsequent k digit numbers and keep adding them to a sum.
2. Break the loop when sum is greater than or equal to n.
3. Maintain a counter to keep track of the number of digits.
4. The value of the counter at the break of the loop will indicate the answer.

C++

```
// CPP program to count number of digits
// in n-th number made of given four digits.
#include <bits/stdc++.h>
using namespace std;

// Efficient function to calculate number
// of digits in the nth number constructed
// by using 6, 1, 4 and 9 as digits in the
// ascending order.
ll number_of_digits(ll n)
{
    ll i, res, sum = 0;

    // Number of digits increase after
    // every i-th number where i increases in
    // powers of 4.
    for (i = 4, res = 1;; i *= 4, res++) {
        sum += i;
        if (sum >= n)
            break;
    }
    return res;
}

// Driver Program.
int main()
{
    ll n = 21;
    cout << number_of_digits(n) << endl;
    return 0;
}
```

Java

```
// Java program to count
```

```
// number of digits in
// n-th number made of
// given four digits.
import java.io.*;

class GFG
{

    // Efficient function to
    // calculate number of digits
    // in the nth number constructed
    // by using 6, 1, 4 and 9 as
    // digits in the ascending order.
    static int number_of_digits(int n)
    {
        int i;
        int res;
        int sum = 0;

        // Number of digits increase
        // after every i-th number
        // where i increases in powers of 4.
        for (i = 4, res = 1;; i *= 4, res++)
        {
            sum += i;
            if (sum >= n)
                break;
        }
        return res;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 21;
        System.out.println(number_of_digits(n));
    }
}

// This code is contributed
// by akt_mit
```

PHP

```
<?php
// PHP program to count number
// of digits in n-th number
// made of given four digits.
```

```
// Efficient function to
// calculate number of digits
// in the nth number constructed
// by using 6, 1, 4 and 9 as
// digits in the ascending order.
function number_of_digits($n)
{
    $i; $res; $sum = 0;

    // Number of digits increase
    // after every i-th number
    // where i increases in
    // powers of 4.
    for ($i = 4, $res = 1;;
         $i *= 4, $res++)
    {
        $sum += $i;
        if ($sum >= $n)
            break;
    }
    return $res;
}

// Driver Code
$n = 21;
echo number_of_digits($n),"\n";

// This code is contributed by ajit
?>
```

Output:

3

Note: Since n could be really large we have used boost library, to know more about boost library give this article a read: <https://www.geeksforgeeks.org/advanced-c-boost-library/>

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/number-digits-nth-number-made-given-four-digits/>

Chapter 108

Number of distinct prime factors of first n natural numbers

Number of distinct prime factors of first n natural numbers - GeeksforGeeks

Given a number N, find the number of distinct prime factors for all numbers in the range [1, N].

Examples :

```
Input : N = 3
Output : 0 1 1
Number of distinct Prime Factors of 1 is 0
Number of distinct Prime Factors of 2 is 1
Number of distinct Prime Factors of 3 is 1
```

```
Input : 6
Output : 0 1 1 1 1 2
Number of distinct Prime Factors of 1 is 0
Number of distinct Prime Factors of 2 is 1
Number of distinct Prime Factors of 3 is 1
Number of distinct Prime Factors of 4 is 1
Number of distinct Prime Factors of 5 is 1
Number of distinct Prime Factors of 6 is 2
2, 3 and 5 are themselves prime. The only prime
factor of 4 is 2. The two prime factors of 6 are
2 and 3.
```

The idea is based on [Sieve of Erathosthenes](#). Whenever we mark number as prime, we also

increment count of prime factors in its multiples.

C++

```
// C++ program to find number of distinct prime factors
// for all number in range [1, N]
#include <bits/stdc++.h>
using namespace std;

void printDistinctPFs(int n)
{
    // array to store the number of distinct primes
    long long factorCount[n + 1];

    // true if index 'i' is a prime
    bool prime[n + 1];

    // initializing the number of factors to 0 and
    for (int i = 0; i <= n; i++) {
        factorCount[i] = 0;
        prime[i] = true; // Used in Sieve
    }

    for (int i = 2; i <= n; i++) {

        // condition works only when 'i' is prime,
        // hence for factors of all prime number,
        // the prime status is changed to false
        if (prime[i] == true) {

            // Number is prime
            factorCount[i] = 1;

            // number of factor of a prime number is 1
            for (int j = i * 2; j <= n; j += i) {

                // incrementing factorCount all
                // the factors of i
                factorCount[j]++;
            }

            // and changing prime status to false
            prime[j] = false;
        }
    }

    // Printing result
    for (int i = 1; i <= n; i++)
```

```
        cout << factorCount[i] << " ";
```

```
}
```

```
// Driver code
```

```
int main()
```

```
{
```

```
    int n = 20; // input
```

```
    printDistinctPFs(n);
```

```
    return 0;
```

```
}
```

Java

```
// Java program to find number
// of distinct prime factors
// for all number in range [1, N]
import java.io.*;

class GFG
{
static void printDistinctPFs(int n)
{
    // array to store the number
    // of distinct primes
    long factorCount[] = new long[n + 1];

    // true if index 'i' is a prime
    boolean prime[] = new boolean[n + 1];

    // initializing the number
    // of factors to 0 and
    for (int i = 0; i <= n; i++)
    {
        factorCount[i] = 0;
        prime[i] = true; // Used in Sieve
    }

    for (int i = 2; i <= n; i++)
    {

        // condition works only when
        // 'i' is prime, hence for
        // factors of all prime number,
        // the prime status is changed to false
        if (prime[i] == true)
        {

            // Number is prime
            factorCount[i]++;
            for (int j = 2 * i; j <= n; j += i)
                prime[j] = false;
        }
    }
}

// This code is contributed by AnkitRai01
```

```
factorCount[i] = 1;

    // number of factor of
    // a prime number is 1
    for (int j = i * 2; j <= n; j += i)
    {

        // incrementing factorCount
        // all the factors of i
        factorCount[j]++;
        // and changing prime
        // status to false
        prime[j] = false;
    }
}

// Printing result
for (int i = 1; i <= n; i++)
System.out.print( factorCount[i] + " ");

// Driver Code
public static void main (String[] args)
{
    int n = 20; // input
    printDistinctPFs(n);
}
```

//This code is contributed by anuj_67.

Python3

```
# Python3 program to find
# number of distinct prime
# factors for all number
# in range [1, N]
def printDistinctPFs(n):

    # array to store the
    # number of distinct primes
    factorCount = [];

    # true if index
    # 'i' is a prime
    prime = [];
```

```
# initializing the number
# of factors to 0 and
for i in range(n + 1):
    factorCount.append(0);
    prime.append(1);
    # Used in Sieve

for i in range(2, n + 1):

    # condition works only
    # when 'i' is prime,
    # hence for factors of
    # all prime number,
    # the prime status is
    # changed to false
    if (prime[i] == 1):

        # Number is prime
        factorCount[i] = 1;

        # number of factor of
        # a prime number is 1
        j = i * 2;
        while(j <= n):

            # incrementing factorCount
            # all the factors of i
            factorCount[j] = factorCount[j] + 1;

            # and changing prime
            # status to false
            prime[j] = 0;
            j = j + i;

    # Printing result
    for i in range(1, n + 1):
        print(factorCount[i] ,
              end = " ");

# Driver code
n = 20;

# input
printDistinctPFs(n);

# This code is contributed
# by mits
```

C#

```
// C# program to find number
// of distinct prime factors
// for all number in range [1, N]
using System;

class GFG
{
static void printDistinctPFs(int n)
{
    // array to store the number
    // of distinct primes
    long []factorCount = new long[n + 1];

    // true if index 'i' is a prime
    bool []prime = new bool[n + 1];

    // initializing the number
    // of factors to 0 and
    for (int i = 0; i <= n; i++)
    {
        factorCount[i] = 0;
        prime[i] = true; // Used in Sieve
    }

    for (int i = 2; i <= n; i++)
    {

        // condition works only
        // when 'i' is prime,
        // hence for factors of
        // all prime number, the
        // prime status is changed
        // to false
        if (prime[i] == true)
        {

            // Number is prime
            factorCount[i] = 1;

            // number of factor of
            // a prime number is 1
            for (int j = i * 2; j <= n; j += i)
            {

                // incrementing factorCount
                // all the factors of i
            }
        }
    }
}
```

```
        factorCount[j]++;
        // and changing prime
        // status to false
        prime[j] = false;
    }
}
}

// Printing result
for (int i = 1; i <= n; i++)
    Console.Write(factorCount[i] + " ");
}

// Driver Code
public static void Main ()
{
    int n = 20; // input
    printDistinctPFs(n);
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find number
// of distinct prime factors
// for all number in range [1, N]
function printDistinctPFs($n)
{
    // array to store the
    // number of distinct primes
    $factorCount = array();

    // true if index
    // 'i' is a prime
    $prime = array();

    // initializing the number
    // of factors to 0 and
    for ($i = 0; $i <= $n; $i++)
    {
        $factorCount[$i] = 0;
        $prime[$i] = true; // Used in Sieve
    }
}
```

```
for ($i = 2; $i <= $n; $i++)
{
    // condition works only
    // when 'i' is prime,
    // hence for factors of
    // all prime number,
    // the prime status is
    // changed to false
    if ($prime[$i] == true)
    {

        // Number is prime
        $factorCount[$i] = 1;

        // number of factor of
        // a prime number is 1
        for ($j = $i * 2;
            $j <= $n; $j += $i)
        {

            // incrementing factorCount
            // all the factors of i
            $factorCount[$j]++;
           

            // and changing prime
            // status to false
            $prime[$j] = false;
        }
    }
}

// Printing result
for ($i = 1; $i <= $n; $i++)
    echo $factorCount[$i] , " ";
}

// Driver code
$n = 20; // input
printDistinctPFs($n);

// This code is contributed
// by anuj_67.
?>
```

Output :

0 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 1 2

This is the most efficient way to calculate the number of prime factors for numbers in [1, N]. Here the data type of n, factorCount can be changed to solve problems with huge constraints.

Improved By : [vt_m, Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/number-of-distinct-prime-factors-of-first-n-natural-numbers/>

Chapter 109

Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)

Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)
- GeeksforGeeks

Prerequisites: [Fenwick Tree \(Binary Indexed Tree\)](#)

Given an array of N numbers, and a number of queries where each query will contain three numbers(l, r and k). The task is to calculate the number of array elements which are greater than K in the subarray[L, R].

Examples:

```
Input: n=6
      q=2
      arr[ ] = { 7, 3, 9, 13, 5, 4 }
      Query1: l=1, r=4, k=6
      Query2: l=2, r=6, k=8
```

```
Output: 3
        2
```

For the first query, [7, 3, 9, 13] represents the subarray from index 1 till 4, in which there are 3 numbers which are greater than k=6 that are {7, 9, 13}.

For the second query, there are only two numbers in the query range which are greater than k.

Naive Approach is to find the answer for each query by simply traversing the array from index l till r and keep adding 1 to the count whenever the array element is greater than k.
Time Complexity: $O(n^*q)$

A Better Approach is to use Merge Sort Tree. In this approach, build a Segment Tree with a vector at each node containing all the elements of the sub-range in a sorted order. Answer each query using the segment tree where Binary Search can be used to calculate how many numbers are present in each node whose sub-range lies within the query range which are greater than k.

Time complexity: $O(q * \log(n) * \log(n))$

An **Efficient Approach** is to solve the problem using offline queries and [Fenwick Trees](#). Below are the steps:

- First store all the array elements and the queries in the same array. For this, we can create a self-structure or class.
- Then sort the structural array in descending order (in case of collision the query will come first then the array element).
- Process the whole array of structure again, but before that create another BIT array (Binary Indexed Tree) whose $\text{query}(i)$ function will return the count of all the elements which are present in the array till i'th index.
- Initially, fill the whole array with 0.
- Create an answer array, in which the answers of each query are stored.
- Process the array of structure.
- If it is an array element, then update the BIT array with +1 from the index of that element.
- If it is a query, then subtract the $\text{query}(r) - \text{query}(l-1)$ and this will be the answer for that query which will be stored in answer array at the index corresponding to the query number.
- Finally output the answer array.

The key observation here is that since the array of the structure has been sorted in descending order. Whenever we encounter any query only the elements which are greater than 'k' comprises the count in the BIT array which is the answer that is needed.

Below is the explanation of structure used in the program:

Pos: stores the order of query. In case of array elements it is kept as 0.

L: stores the starting index of the query's subarray. In case of array elements it is also 0.

R: stores the ending index of the query's subarray. In case of array element it is used to store the position of element in the array.

Val: store 'k' of the query and all the array elements.

Below is the implementation of the above approach:

```
// C++ program to print the number of elements
// greater than k in a subarray of range L-R.
#include <bits/stdc++.h>
using namespace std;

// Structure which will store both
// array elements and queries.
struct node {
    int pos;
    int l;
    int r;
    int val;
};

// Boolean comparator that will be used
// for sorting the structural array.
bool comp(node a, node b)
{
    // If 2 values are equal the query will
    // occur first then array element
    if (a.val == b.val)
        return a.l > b.l;

    // Otherwise sorted in descending order.
    return a.val > b.val;
}

// Updates the node of BIT array by adding
// 1 to it and its ancestors.
void update(int* BIT, int n, int idx)
{
    while (idx <= n) {
        BIT[idx]++;
        idx += idx & (-idx);
    }
}

// Returns the count of numbers of elements
// present from starting till idx.
int query(int* BIT, int idx)
{
    int ans = 0;
    while (idx) {
        ans += BIT[idx];

        idx -= idx & (-idx);
    }
    return ans;
}
```

```
// Function to solve the queries offline
void solveQuery(int arr[], int n, int QueryL[],
                int QueryR[], int QueryK[], int q)
{
    // create node to store the elements
    // and the queries
    node a[n + q + 1];
    // 1-based indexing.

    // traverse for all array numbers
    for (int i = 1; i <= n; ++i) {
        a[i].val = arr[i - 1];
        a[i].pos = 0;
        a[i].l = 0;
        a[i].r = i;
    }

    // iterate for all queries
    for (int i = n + 1; i <= n + q; ++i) {
        a[i].pos = i - n;
        a[i].val = QueryK[i - n - 1];
        a[i].l = QueryL[i - n - 1];
        a[i].r = QueryR[i - n - 1];
    }

    // In-built sort function used to
    // sort node array using comp function.
    sort(a + 1, a + n + q + 1, comp);

    // Binary Indexed tree with
    // initially 0 at all places.
    int BIT[n + 1];

    // initially 0
    memset(BIT, 0, sizeof(BIT));

    // For storing answers for each query( 1-based indexing ).
    int ans[q + 1];

    // traverse for numbers and query
    for (int i = 1; i <= n + q; ++i) {
        if (a[i].pos != 0) {

            // call function to returns answer for each query
            int cnt = query(BIT, a[i].r) - query(BIT, a[i].l - 1);

            // This will ensure that answer of each query
            ans[a[i].pos] = cnt;
        }
    }
}
```

```
// are stored in order it was initially asked.  
ans[a[i].pos] = cnt;  
}  
else {  
    // a[i].r contains the position of the  
    // element in the original array.  
    update(BIT, n, a[i].r);  
}  
}  
}  
// Output the answer array  
for (int i = 1; i <= q; ++i) {  
    cout << ans[i] << endl;  
}  
}  
  
// Driver Code  
int main()  
{  
    int arr[] = { 7, 3, 9, 13, 5, 4 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    // 1-based indexing  
    int QueryL[] = { 1, 2 };  
    int QueryR[] = { 4, 6 };  
  
    // k for each query  
    int QueryK[] = { 6, 8 };  
  
    // number of queries  
    int q = sizeof(QueryL) / sizeof(QueryL[0]);  
  
    // Function call to get  
    solveQuery(arr, n, QueryL, QueryR, QueryK, q);  
  
    return 0;  
}
```

Output:

```
3  
2
```

Time Complexity: $O(N * \log N)$ where $N = (n+q)$

What is offline query?

In some questions, it is hard to answer queries in any random order. So instead of answering each query separately, store all the queries and then order them accordingly to calculate

answer for them efficiently. Store all the answers and then output it in the order it was initially given.

This technique is called *Offline Query*.

Note: Instead of Fenwick Tree, segment tree can also be used where each node of the segment tree will store the number of elements inserted till that iteration. The update and query functions will change, rest of the implementation will remain same.

Necessary Condition For Offline Query: This technique can be used only when the answer of one query does not depend on the answers of previous queries since after sorting the order of queries may change.

Improved By : [dhruvgupta167](#)

Source

<https://www.geeksforgeeks.org/number-of-elements-greater-than-k-in-the-range-l-to-r-using-fenwick-tree-offline-queries/>

Chapter 110

Number of horizontal or vertical line segments to connect 3 points

Number of horizontal or vertical line segments to connect 3 points - GeeksforGeeks

Given three points on the x-y coordinate plane. You need to find the no. of line segments formed by making a polyline passing through these points. (Line segment can be vertically or horizontally aligned to the coordinate axis)

Examples :

```
Input : A = {-1, -1}, B = {-1, 3}, C = {4, 3}
Output : 2
Explanation:
There are two segments in this polyline.
Input : A = {1, 1}, B = {2, 3} C = {3, 2}
Output : 3
```

The result is one if all points are on x axis or y axis. The result is 2 if points can form L shape. L shape is formed if any of the three points can be used as a joining point. Otherwise answer is 3.

C++

```
// CPP program to find number of horizontal (or vertical
// line segments needed to connect three points.
#include <iostream>
using namespace std;
```

```
// Function to check if the third point forms a
// rectangle with other two points at corners
bool isBetween(int a, int b, int c)
{
    return min(a, b) <= c && c <= max(a, b);
}

// Returns true if point k can be used as a joining
// point to connect using two line segments
bool canJoin(int x[], int y[], int i, int j, int k)
{
    // Check for the valid polyline with two segments
    return (x[k] == x[i] || x[k] == x[j]) &&
           isBetween(y[i], y[j], y[k]) ||
           (y[k] == y[i] || y[k] == y[j]) &&
           isBetween(x[i], x[j], x[k]);
}

int countLineSegments(int x[], int y[])
{
    // Check whether the X-coordinates or
    // Y-coordinates are same.
    if ((x[0] == x[1] && x[1] == x[2]) ||
        (y[0] == y[1] && y[1] == y[2]))
        return 1;

    // Iterate over all pairs to check for two
    // line segments
    else if (canJoin(x, y, 0, 1, 2) ||
             canJoin(x, y, 0, 2, 1) ||
             canJoin(x, y, 1, 2, 0))
        return 2;

    // Otherwise answer is three.
    else
        return 3;
}

// Driver code
int main()
{
    int x[3], y[3];
    x[0] = -1; y[0] = -1;
    x[1] = -1; y[1] = 3;
    x[2] = 4; y[2] = 3;
    cout << countLineSegments(x, y);
    return 0;
}
```

}

Java

```
// Java program to find number of horizontal
// (or vertical line segments needed to
// connect three points.
import java.io.*;

class GFG {

    // Function to check if the third
    // point forms a rectangle with
    // other two points at corners
    static boolean isBetween(int a, int b, int c)
    {
        return (Math.min(a, b) <= c &&
                c <= Math.max(a, b));
    }

    // Returns true if point k can be
    // used as a joining point to connect
    // using two line segments
    static boolean canJoin(int x[], int y[],
                           int i, int j, int k)
    {
        // Check for the valid polyline
        // with two segments
        return (x[k] == x[i] || x[k] == x[j]) &&
               isBetween(y[i], y[j], y[k]) ||
               (y[k] == y[i] || y[k] == y[j]) &&
               isBetween(x[i], x[j], x[k]);
    }

    static int countLineSegments(int x[], int y[])
    {
        // Check whether the X-coordinates or
        // Y-coordinates are same.
        if ((x[0] == x[1] && x[1] == x[2]) ||
            (y[0] == y[1] && y[1] == y[2]))
            return 1;

        // Iterate over all pairs to check for two
        // line segments
        else if (canJoin(x, y, 0, 1, 2) ||
                 canJoin(x, y, 0, 2, 1) ||
                 canJoin(x, y, 1, 2, 0))
            return 2;
    }
}
```

```
// Otherwise answer is three.  
else  
    return 3;  
}  
  
// Driver code  
public static void main (String[] args) {  
  
    int x[]={new int[3], y[]={new int[3];  
  
        x[0] = -1; y[0] = -1;  
        x[1] = -1; y[1] = 3;  
        x[2] = 4; y[2] = 3;  
  
        System.out.println(countLineSegments(x, y));  
    }  
  
}  
  
// This code is contributed by vt_m
```

Python3

```
# Python program to find number  
# of horizontal (or vertical  
# line segments needed to  
# connect three points.  
  
import math  
  
# Function to check if the  
# third point forms a  
# rectangle with other  
# two points at corners  
def isBetween(a, b, c) :  
  
    return min(a, b) <= c and c <= max(a, b)  
  
# Returns true if point k  
# can be used as a joining  
# point to connect using  
# two line segments  
def canJoin( x, y, i, j, k) :  
  
    # Check for the valid polyline
```

```
# with two segments
return (x[k] == x[i] or x[k] == x[j]) and isBetween(y[i], y[j], y[k]) or (y[k] == y[i] or y[k] == y[j])

def countLineSegments( x, y):

    # Check whether the X-coordinates or
    # Y-coordinates are same.
    if ((x[0] == x[1] and x[1] == x[2]) or
        (y[0] == y[1] and y[1] == y[2])):
        return 1

    # Iterate over all pairs to check for two
    # line segments
    elif (canJoin(x, y, 0, 1, 2) or
          canJoin(x, y, 0, 2, 1) or
          canJoin(x, y, 1, 2, 0)):
        return 2

    # Otherwise answer is three.
    else:
        return 3

#driver code
x= [-1,-1, 4]
y= [-1, 3, 3]

print(countLineSegments(x, y))

# This code is contributed by Gitanjali.
```

C#

```
// C# program to find number of horizontal
// (or vertical) line segments needed to
// connect three points.
using System;

class GFG {

    // Function to check if the third
    // point forms a rectangle with
    // other two points at corners
    static bool isBetween(int a, int b, int c)
    {
        return (Math.Min(a, b) <= c &&
                c <= Math.Max(a, b));
    }
}
```

```
// Returns true if point k can be
// used as a joining point to connect
// using two line segments
static bool canJoin(int[] x, int[] y,
                     int i, int j, int k)
{
    // Check for the valid polyline
    // with two segments
    return (x[k] == x[i] || x[k] == x[j])
        && isBetween(y[i], y[j], y[k])
        || (y[k] == y[i] || y[k] == y[j])
        && isBetween(x[i], x[j], x[k]);
}

static int countLineSegments(int[] x, int[] y)
{
    // Check whether the X-coordinates or
    // Y-coordinates are same.
    if ((x[0] == x[1] && x[1] == x[2]) ||
        (y[0] == y[1] && y[1] == y[2]))
        return 1;

    // Iterate over all pairs to check for two
    // line segments
    else if (canJoin(x, y, 0, 1, 2)
            || canJoin(x, y, 0, 2, 1)
            || canJoin(x, y, 1, 2, 0))
        return 2;

    // Otherwise answer is three.
    else
        return 3;
}

// Driver code
public static void Main()
{
    int[] x = new int[3];
    int[] y = new int[3];

    x[0] = -1;
    y[0] = -1;
    x[1] = -1;
    y[1] = 3;
    x[2] = 4;
```

```
y[2] = 3;  
  
Console.WriteLine(countLineSegments(x, y));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to find number  
// of horizontal (or vertical  
// line segments needed to  
// connect three points.  
  
// Function to check if the  
// third point forms a  
// rectangle with other  
// two points at corners  
function isBetween( $a, $b, $c)  
{  
    return min($a, $b) <= $c and  
           $c <= max($a, $b);  
}  
  
// Returns true if point k  
// can be used as a joining  
// point to connect using  
// two line segments  
function canJoin($x, $y, $i, $j, $k)  
{  
    // Check for the valid  
    // polyline with two segments  
    return ($x[$k] == $x[$i] or  
           $x[$k] == $x[$j]) and  
           isBetween($y[$i], $y[$j], $y[$k]) or  
           ($y[$k] == $y[$i] or  
            $y[$k] == $y[$j]) and  
           isBetween($x[$i], $x[$j], $x[$k]);  
}  
  
function countLineSegments( $x, $y)  
{  
    // Check whether the X-coordinates  
    // or Y-coordinates are same.  
    if (($x[0] == $x[1] and $x[1] == $x[2]) or
```

```
($y[0] == $y[1] and $y[1] == $y[2]))  
    return 1;  
  
    // Iterate over all pairs to  
    // check for two line segments  
    else if (canJoin($x, $y, 0, 1, 2) or  
            canJoin($x, $y, 0, 2, 1) ||  
            canJoin($x, $y, 1, 2, 0))  
        return 2;  
  
    // Otherwise answer is three.  
    else  
        return 3;  
}  
  
// Driver code  
$x = array();  
$y = array();  
$x[0] = -1; $y[0] = -1;  
$x[1] = -1; $y[1] = 3;  
$x[2] = 4; $y[2] = 3;  
echo countLineSegments($x, $y);  
  
// This code is contributed by anuj_67.  
?>
```

Output :

2

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-no-segments-polyline/>

Chapter 111

Number of integral solutions for equation $x = b * (\text{sumofdigits}(x) \hat{a}) + c$

Number of integral solutions for equation $x = b * (\text{sumofdigits}(x) \hat{a}) + c$ - GeeksforGeeks

Given a, b and c which are part of the equation $x = b * (\text{sumofdigits}(x) \hat{a}) + c$.

Where $\text{sumofdigits}(x)$ determines the sum of all digits of the number x. The task is to find out all integer solutions for x that satisfy the equation and print them in increasing order.

Given that, $1 \leq x \leq 10^9$

Examples:

Input: a = 3, b = 2, c = 8

Output: 10 2008 13726

Values of x are: 10 2008 13726. For 10, $s(x)$ is 1; Putting value of $s(x)$ in equation $b * (s(x) \hat{a}) + c$ we get 10, and as 10 lies in range $0 < x < 1e+9$, therefore 10 is a possible answer, similar for 2008 and 13726. No other value of x satisfies the equation for the given value of a, b and c

Input: a = 2, b = 2, c = -1

Output: 1 31 337 967

Values of x that satisfy the equation are: 1 31 337 967

Approach: $\text{sumofdigits}(x)$ can be in the range of $1 \leq s(x) \leq 81$ for the given range of x i.e $0 < x < 1e+9$. This is because value of x can be minimum 0 where $\text{sumofdigits}(x)=0$ and maximum 999999999 where $\text{sumofdigits}(x)$ is 81. So first iterate through 1 to 81 to find x from the given equation, then cross check if the sum of digits of the number found, is same as the value of $\text{sumofdigits}(x)$. If both are same then increase the counter and store the result in an array.

Below is the implementation of the above approach:

C++

```
// C++ program to find the numbers of
// values that satisfy the equation
#include <bits/stdc++.h>
using namespace std;

// This function returns the sum of
// the digits of a number
int getsum(int a)
{
    int r = 0, sum = 0;
    while (a > 0) {
        r = a % 10;
        sum = sum + r;
        a = a / 10;
    }
    return sum;
}

// This function creates
// the array of valid numbers
void value(int a, int b, int c)
{
    int co = 0, p = 0;
    int no, r = 0, x = 0, q = 0, w = 0;
    vector<int> v;

    for (int i = 1; i < 82; i++) {

        // this computes s(x)^a
        no = pow((double)i, double(a));

        // this gives the result of equation
        no = b * no + c;

        if (no > 0 && no < 1000000000) {
            x = getsum(no);

            // checking if the sum same as i
            if (x == i) {

                // counter to keep track of numbers
                q++;

                // resultant array
```

```
        v.push_back(no);
        w++;
    }
}

// prints the number
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
}
}

// Driver Code
int main()
{
    int a = 2, b = 2, c = -1;

    // calculate which value
    // of x are possible
    value(a, b, c);

    return 0;
}
```

PHP

```
<?php
// PHP program to find the numbers of
// values that satisfy the equation

// This function returns the sum of
// the digits of a number
function getsum($a)
{
    $r = 0;
    $sum = 0;
    while ($a > 0)
    {
        $r = $a % 10;
        $sum = $sum + $r;
        $a = (int)($a / 10);
    }
    return $sum;
}

// This function creates
// the array of valid numbers
function value($a, $b, $c)
```

```
{  
    $co = 0;  
    $p = 0;  
    $no;  
    $r = 0;  
    $x = 0;  
    $q = 0;  
    $w = 0;  
    $v = array();  
    $u = 0;  
  
    for ($i = 1; $i < 82; $i++)  
    {  
  
        // this computes s(x)^a  
        $no = pow($i, $a);  
  
        // this gives the result  
        // of equation  
        $no = $b * $no + $c;  
  
        if ($no > 0 && $no < 1000000000)  
        {  
            $x = getsum($no);  
  
            // checking if the  
            // sum same as i  
            if ($x == $i)  
            {  
  
                // counter to keep  
                // track of numbers  
                $q++;  
  
                // resultant array  
                $v[$u++] = $no;  
                $w++;  
            }  
        }  
    }  
  
    // prints the number  
    for ($i = 0; $i < $u; $i++)  
    {  
        echo $v[$i] . " ";  
    }  
}
```

```
// Driver Code
$a = 2;
$b = 2;
$c = -1;

// calculate which value
// of x are possible
value($a, $b, $c);

// This code is contributed
// by mits
?>
```

Output:

1 31 337 967

Time Complexity: O(N)

Auxiliary Space: O(N)

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/number-of-integral-solutions-for-equation-x-bsumofdigitxac/>

Chapter 112

Number of odd and even results for every value of x in range [min, max] after performing N steps

Number of odd and even results for every value of x in range [min, max] after performing N steps - GeeksforGeeks

Given a number N and the min and max range. Given N values of a and b respectively. The task is to count the number of even/odd results after performing a series of N operations as described below.

At every step, calculate:

$$y_N = a_N y_{N-1} + b_N.$$

Explanation:

- Step 1: $y_1 = a_1 x + b_1$
- Step 2: $y_2 = a_2 y_1 + b_2 \Rightarrow y_2 = a_2 a_1 x + a_2 b_1 + b_2$
- Step 3: $y_3 = a_3 y_2 + b_3 \Rightarrow y_3 = a_3 a_2 a_1 x + a_3 a_2 b_1 + a_3 b_2 + b_3$
- Step n: $y_n = a_n y_{n-1} + b_n$

To obtain the final results, take the values of y_0 as every value in the range [min, max]. For simplicity, we have assumed the value of y_0 to be x and replaced x with all possible values in the range [min, max] in the final equation to calculate results.

Examples:

Input: n = 2, min = 1, max = 4

a = 1, b = 2

a = 3, b = 4

Output: even = 2, odd = 2.

Step1: $y = 1x + 2 = x+2$

Step2: $y = 3(x+2) + 4 = 3x + 10$

Putting all values of in range [1, 4],
2 odd values and 2 even values are obtained.

Input: n = 1, min = 4, max = 60

a= 1, b = 2

Output: even = 29, odd = 28

A **naive approach** will be to store the values of a and b in an array and calculate the final result for each number in the specified range. If the result is even, count of even is incremented. Otherwise, count of odd is incremented.

An **Efficient Approach** which is used here is the basic concept that product of two numbers is even if any one of the numbers is even, otherwise odd, and the sum of two numbers is even only if both the numbers are even. Here, it is seen that at each step, a number is multiplied with x, and another constant is added to the product. The task is to check for the result to be even or odd. At the last step of calculation, check that if $a_1a_2a_3\dots a_n$ is even/odd, and $a_2a_3\dots a_n b_1 + a_3a_4\dots a_n b_2 + \dots + b_n$ is even/odd. Checking if $a_1a_2a_3\dots a_n$ is even/odd: If any a_i is even, the product will always be even, otherwise it will be odd. Checking if $a_2a_3\dots a_n b_1 + a_3a_4\dots a_n b_2 + \dots + b_n$ is even/odd:

Below table explains all the various possibilities for coefficients:

$a_2a_3\dots a_{i-1}b_1 + a_3a_4\dots a_{i-1}b_2 + \dots + b_{i-1}$	a_i	b_i	$a_2a_3\dots a_ib_1 + a_3a_4\dots a_ib_2 + \dots + b_i$
odd	odd	odd	even
odd	odd	even	odd
odd	even	odd	odd
odd	even	even	even
even	odd	odd	odd
even	odd	even	even
even	even	odd	odd
even	even	even	even

Below table explains all the various possibilities for $y = ax + b$:

x	a	b	y
odd	odd	odd	even
odd	odd	even	odd
odd	even	odd	odd

x	a	b	y
odd	even	even	even
even	odd	odd	odd
even	odd	even	even
even	even	odd	odd
even	even	even	even

Instead of traversing for all the numbers in the range [min, max], divide it into two parts to check whether the number in the range is even or odd as all the even inputs have the same result, and all the odd inputs have the same result. So, check for one case and multiply it by the number of even and odd in the range.

The above calculation is carried out, and the coefficient of x at the last step is checked.

- If it is even, then *aeven* is true, else false.
- If the constant is even, then *beven* is true, otherwise false.
- The coefficient of x is even of any one value of a is even in any one layer.
- The constant term after the last layer if executed is checked with the help of value of *beven*, and current a and b.

With the help of the table given above (first one), the constant at each layer is tested and the value of *beven* is updated accordingly.

Assume x is even, the value of even and odd is initialized.

1. If x is even, then ax will always be even, regardless of a. Hence, depending on the value of the constant term, the result will be even or odd.
2. If the constant is even, then the result is even, hence even is initialized by the number of even in the given range ($\text{max}/2 - (\text{min}-1)/2$) and odd is initialized by zero.
3. If constant is odd, then the result is odd, hence odd is initialized by the number of even in the given range ($\text{max}/2 - (\text{min}-1)/2$) and even is initialized by zero.

Assuming x is odd, the value of even and odd is updated.

1. If a is odd, ax is odd. If a is even, ax is even.
2. If ax and constant, both are odd or ax and constant, both are even, then the result is even, hence even is incremented by the number of odd in the given range (**max - min + 1 - number of even**).
3. If ax is even and constant is odd or ax is odd and constant is odd, then the result is odd, hence the number of odd is incremented by the number of odd in the given range (**max - min + 1 - number of even**).

Below is the implementation of the above approach:

C++

```
// C++ program to print
// Number of odd/even results for
// every value of x in range [min, end]
// after performing N steps
#include <bits/stdc++.h>
using namespace std;

// Function that prints the
// number of odd and even results
void count_even_odd(int min, int max, int steps[][])
{
    int a, b, even, odd;

    // If constant at layer i is even, beven is true,
    // otherwise false. If the coefficient of x at
    // layer i is even, aeven is true, otherwise false.
    bool beven = true, aeven = false;
    int n = 2;
    for (int i = 0; i < n; i++) {

        a = steps[i][0], b = steps[i][1];

        // If any of the coefficients at any layer is found
        // to be even, then the product of all the
        // coefficients will always be even.

        if (!(aeven || a & 1))
            aeven = true;

        // Checking whether the constant added after all
        // layers is even or odd.

        if (beven) {
            if (b & 1)
                beven = false;
        }
        else if (!(a & 1)) {
            if (!(b & 1))
                beven = true;
        }
        else {
            if (b & 1)
                beven = true;
        }
    }

    // Counting the number of even and odd.
```

```
// Assuming input x is even.
if (beven) {
    even = (int)max / 2 - (int)(min - 1) / 2;
    odd = 0;
}
else {
    even = (int)max / 2 - (int)(min - 1) / 2;
    odd = 0;
}

// Assuming input x is odd.
if (!(beven ^ aeven))
    even += max - min + 1 - (int)max / 2
        + (int)(min - 1) / 2;
else
    odd += max - min + 1 - (int)max / 2
        + (int)(min - 1) / 2;

// Displaying the counts.
cout << "even = " << even << ", odd = " << odd << endl;
}

// Driver Code
int main()
{
    int min = 1, max = 4;
    int steps[][][2] = { { 1, 2 },
                        { 3, 4 } };

    count_even_odd(min, max, steps);
    return 0;
}
```

Java

```
// Java program to print
// Number of odd/even
// results for every value
// of x in range [min, end]
// after performing N steps
import java.io.*;

class GFG
{

    // Function that prints
    // the number of odd and
    // even results
```

```
static void count_even_odd(int min,
                           int max,
                           int steps[][][])
{
    int a, b, even, odd;

    // If constant at layer i
    // is even, b even is true,
    // otherwise false. If the
    // coefficient of x at layer
    // i is even, a even is true,
    // otherwise false.
    boolean b even = true,
           a even = false;
    int n = 2;
    for (int i = 0; i < n; i++)
    {

        a = steps[i][0];
        b = steps[i][1];

        // If any of the coefficients
        // at any layer is found to be
        // even, then the product of
        // all the coefficients will
        // always be even.
        if (!(a even || (a & 1) > 0))
            a even = true;

        // Checking whether the
        // constant added after all
        // layers is even or odd.
        if (b even)
        {
            if ((b & 1) > 0)
                b even = false;
        }
        else if (!((a & 1) > 0))
        {
            if (!((b & 1) > 0))
                b even = true;
        }
        else
        {
            if ((b & 1) > 0)
                b even = true;
        }
    }
}
```

```
// Counting the number
// of even and odd.

// Assuming input x is even.
if (beven)
{
    even = (int)max / 2 -
           (int)(min - 1) / 2;
    odd = 0;
}
else
{
    even = (int)max / 2 -
           (int)(min - 1) / 2;
    odd = 0;
}

// Assuming input x is odd.
if (!beven ^ aeven)
    even += max - min + 1 -
            (int)max / 2 +
            (int)(min - 1) / 2;
else
    odd += max - min + 1 -
            (int)max / 2 +
            (int)(min - 1) / 2;

// Displaying the counts.
System.out.print("even = " + even +
                  ", odd = " + odd);
}

// Driver Code
public static void main (String[] args)
{
    int min = 1, max = 4;
    int steps[][] = {{1, 2},
                    {3, 4}};

    count_even_odd(min, max, steps);
}
}

// This code is contributed
// by anuj_67.
```

Python3

```
# Python3 program to print
# Number of odd/even results
# for every value of x in
# range [min, end] after
# performing N steps

# Function that prints
# the number of odd
# and even results
def count_even_odd(min, max, steps):

    # If constant at layer i
    # is even, beven is True,
    # otherwise False. If
    # the coefficient of x at
    # layer i is even, aeven
    # is True, otherwise False.
    beven = True
    aeiven = False
    n = 2
    for i in range(0, n) :
        a = steps[i][0]
        b = steps[i][1]

        # If any of the coefficients
        # at any layer is found to
        # be even, then the product
        # of all the coefficients
        # will always be even.
        if (not(aeiven or a & 1)):
            aeiven = True

        # Checking whether the
        # constant added after all
        # layers is even or odd.
        if (beven) :
            if (b & 1):
                beven = False

        elif (not(a & 1)) :
            if (not(b & 1)):
                beven = True

        else :
            if (b & 1):
                beven = True

    # Counting the number
```

```
# of even and odd.

# Assuming input x is even.
if (beven):
    even = (int(max / 2) -
             int((min - 1) / 2))
    odd = 0

else :
    even = (int(max / 2) -
             int((min - 1) / 2))
    odd = 0

# Assuming input x is odd.
if (not(beven ^ aeven)):
    even += (max - min + 1 -
              int(max / 2) + int((min - 1) / 2))
else:
    odd += (max - min + 1 -
             int(max / 2) + int((min - 1) / 2))

# Displaying the counts.
print("even = " , even ,
      ", odd = " , odd, sep = "")

# Driver Code
min = 1
max = 4
steps = [[1, 2],[3, 4]]
count_even_odd(min, max, steps)

# This code is contributed
# by Smitha
```

C#

```
// C# program to print
// Number of odd/even
// results for every value
// of x in range [min, end]
// after performing N steps
using System;

class GFG
{

    // Function that prints
    // the number of odd and
```

```
// even results
static void count_even_odd(int min,
                           int max,
                           int [,]steps)
{
    int a, b, even, odd;

    // If constant at layer i
    // is even, beven is true,
    // otherwise false. If the
    // coefficient of x at layer
    // i is even, aeven is true,
    // otherwise false.
    bool beven = true,
        aeven = false;
    int n = 2;
    for (int i = 0; i < n; i++)
    {

        a = steps[i, 0];
        b = steps[i, 1];

        // If any of the coefficients
        // at any layer is found
        // to be even, then the
        // product of all the
        // coefficients will always
        // be even.
        if (!(aeven || (a & 1) > 0))
            aeven = true;

        // Checking whether the
        // constant added after all
        // layers is even or odd.
        if (beven)
        {
            if ((b & 1) > 0)
                beven = false;
        }
        else if (!((a & 1) > 0))
        {
            if (!((b & 1) > 0))
                beven = true;
        }
        else
        {
            if ((b & 1) > 0)
                beven = true;
        }
    }
}
```

```
        }

    }

    // Counting the number
    // of even and odd.

    // Assuming input
    // x is even.
    if (beven)
    {
        even = (int)max / 2 -
               (int)(min - 1) / 2;
        odd = 0;
    }
    else
    {
        even = (int)max / 2 -
               (int)(min - 1) / 2;
        odd = 0;
    }

    // Assuming input
    // x is odd.
    if (!(beven ^ aeven))
        even += max - min + 1 -
                (int)max / 2 +
                (int)(min - 1) / 2;
    else
        odd += max - min + 1 -
               (int)max / 2 +
               (int)(min - 1) / 2;

    // Displaying the counts.
    Console.WriteLine("even = " + even +
                      ", odd = " + odd);
}

// Driver Code
public static void Main ()
{
    int min = 1, max = 4;
    int [,]steps = {{1, 2},
                    {3, 4}};

    count_even_odd(min, max, steps);
}
}
```

```
// This code is contributed  
// by anuj_67.
```

Output:

```
even = 2, odd = 2
```

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/number-of-odd-and-even-results-for-every-value-of-x-in-range-min-max-after-performing-n-steps/>

Chapter 113

Number of prime pairs in an array

Number of prime pairs in an array - GeeksforGeeks

Given an array. The task is to count the possible pairs which can be formed using the elements of the array where both of the elements in the pair are prime.

Note: Pairs such as (a, b) and (b, a) should not be considered different.

Examples:

```
Input: arr[] = {1, 2, 3, 5, 7, 9}
Output: 6
From the given array, prime pairs are
(2, 3), (2, 5), (2, 7), (3, 5), (3, 7), (5, 7)
```

```
Input: arr[] = {1, 4, 5, 9, 11}
Output: 1
```

A **naive approach** is to count all possible pairs in the array and check if both the elements in the pair are prime.

An **efficient approach** is to count a number of primes in the array using Sieve of Eratosthenes. Let it be C. Now, total number of possible pairs is equal to $C*(C-1)/2$.

Below is the implementation of the above approach:

```
// CPP program to find count of
// prime pairs in given array.
#include <bits/stdc++.h>
using namespace std;

// Function to find count of prime pairs
```

```
int pairCount(int arr[], int n)
{
    // Find maximum value in the array
    int max_val = *max_element(arr, arr+n);

    // USE SIEVE TO FIND ALL PRIME NUMBERS LESS
    // THAN OR EQUAL TO max_val
    // Create a boolean array "prime[0..n]". A
    // value in prime[i] will finally be false
    // if i is Not a prime, else true.
    vector<bool> prime(max_val + 1, true);

    // Remaining part of SIEVE
    prime[0] = false;
    prime[1] = false;
    for (int p = 2; p * p <= max_val; p++) {

        // If prime[p] is not changed, then
        // it is a prime
        if (prime[p] == true) {

            // Update all multiples of p
            for (int i = p * 2; i <= max_val; i += p)
                prime[i] = false;
        }
    }

    // Find all primes in arr[]
    int count = 0;
    for (int i = 0; i < n; i++)
        if (prime[arr[i]])
            count++;

    // return the count of
    // possible prime pairs
    // Number of unique pairs
    // with N elements is N*(N-1)/2
    return (count * (count - 1)) / 2;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << pairCount(arr, n);
    return 0;
}
```

Output:

6

Source

<https://www.geeksforgeeks.org/number-of-prime-pairs-in-an-array/>

Chapter 114

Number of quadrilaterals possible from the given points

Number of quadrilaterals possible from the given points - GeeksforGeeks

Given four points (x, y) in the cartesian coordinate. Find the possible no of quadrilaterals than can be formed by joining all the four points.

Examples:

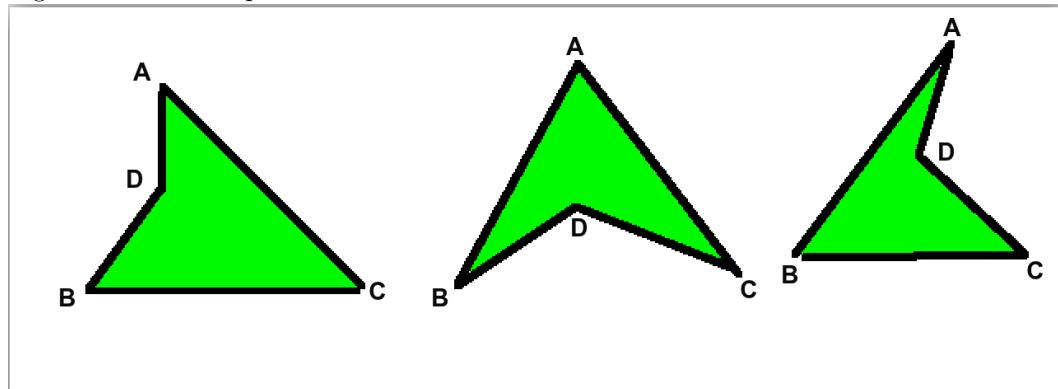
Input: $A=(0, 9)$, $B=(-1, 0)$, $C=(5, -1)$, $D=(5, 9)$

Output: Only one quadrilateral is possible (ABCD) in any orientation

Input: $A=(0, 9)$, $B=(-1, 0)$, $C=(5, -1)$, $D=(0, 3)$

Output: 3 quadrilaterals are possible (ABCD), (ADBC), (ABDC)

Figure for 2nd example:



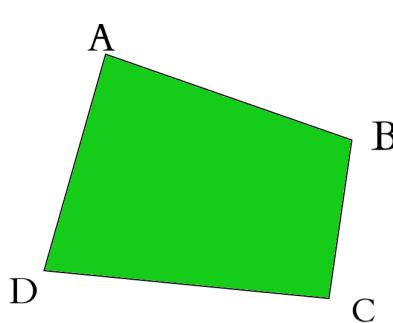
Approach:

1. We need to check whether any of the given points are same. **If yes, no of quadrilateral = 0**
2. Then we need to check whether any of the 3 points of the given 4 points are collinear or not. **If yes, no of quadrilateral=0.** Check [Program to check if three points are collinear](#) link to check collinearity of 3 points.
3. (a) if its a **convex quadrilateral** then there is only **one possible quadrilateral**.
 (b) if its a **concave quadrilateral** then there are **3 possible quadrilaterals**.

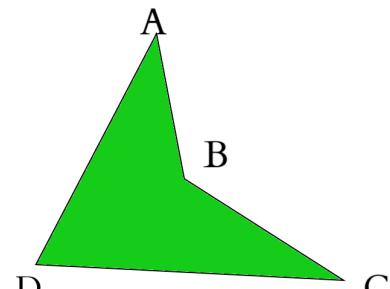
This can be determined by [How to check if two given line segments intersect?](#) of diagonals. In case of a **convex quadrilateral**, the diagonals will intersect whereas in case of a concave quadrilateral the diagonals won't intersect.

since we don't know the orientation of the points, we can't specifically determine the diagonals so all the distinct line segments(no common points in the two line segments) of the quadrilateral and determine whether they intersect or not.

Refer to the figure to understand how to determine the type of quadrilateral :



Convex quadrilateral



Concave quadrilateral

Convex quadrilateral:

line AB and line DC do not intersect
 line AD and line BC do not intersect
 line AC and line BD intersect
 so total no of intersection= 1

Concave quadrilateral

line AB and line DC do not intersect
 line AD and line BC do not intersect
 line AC and line BD intersect
 so total no of intersection= 0

if no of intersection = 1, its a convex quadrilateral so no of possibility= 1
if no of intersection = 0, its a concave quadrilateral so no of possibilities = 3

C++

```
#include <iostream>
using namespace std;

struct Point // points
{
    int x;
    int y;
};

// determines the orientation of points
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;
    return (val > 0) ? 1 : 2;
}

// check whether the distinct line segments intersect
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    return false;
}

// check if points overlap(similar)
bool similar(Point p1, Point p2)
{
    // it is same, we are returning false because
    // quadrilateral is not possible in this case
    if (p1.x == p2.x && p1.y == p2.y)
        return false;

    // it is not same, So there is a
```

```
// possibility of a quadrilateral
return true;
}

// check for collinearity
bool collinear(Point p1, Point p2, Point p3)
{
    int x1 = p1.x, y1 = p1.y;
    int x2 = p2.x, y2 = p2.y;
    int x3 = p3.x, y3 = p3.y;

    // it is collinear, we are returning false
    // because quadrilateral is not possible in this case
    if ((y3 - y2) * (x2 - x1) == (y2 - y1) * (x3 - x2))
        return false;

    // it is not collinear, So there
    // is a possibility of a quadrilateral
    else
        return true;
}

int no_of_quads(Point p1, Point p2, Point p3, Point p4)
{
    // ** Checking for cases where no quadrilateral = 0 **

    // check if any of the points are same
    bool same = true;
    same = same & similar(p1, p2);
    same = same & similar(p1, p3);
    same = same & similar(p1, p4);
    same = same & similar(p2, p3);
    same = same & similar(p2, p4);
    same = same & similar(p3, p4);

    // similar points exist
    if (same == false)
        return 0;

    // check for collinearity
    bool coll = true;
    coll = coll & collinear(p1, p2, p3);
    coll = coll & collinear(p1, p2, p4);
    coll = coll & collinear(p1, p3, p4);
    coll = coll & collinear(p2, p3, p4);

    // points are collinear
    if (coll == false)
```

```
    return 0;

/** Checking for cases where no of quadrilaterals= 1 or 3 **

int check = 0;

if (doIntersect(p1, p2, p3, p4))
    check = 1;
if (doIntersect(p1, p3, p2, p4))
    check = 1;
if (doIntersect(p1, p2, p4, p3))
    check = 1;

if (check == 0)
    return 3;
return 1;
}

// Driver code
int main()
{
    struct Point p1, p2, p3, p4;
    // A =(0, 9), B = (-1, 0), C = (5, -1), D=(5, 9)
    p1.x = 0, p1.y = 9;
    p2.x = -1, p2.y = 0;
    p3.x = 5, p3.y = -1;
    p4.x = 5, p4.y = 9;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(-1, 0), C=(5, -1), D=(0, 3)
    p1.x = 0, p1.y = 9;
    p2.x = -1, p2.y = 0;
    p3.x = 5, p3.y = -1;
    p4.x = 0, p4.y = 3;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(0, 10), C=(0, 11), D=(0, 12)
    p1.x = 0, p1.y = 9;
    p2.x = 0, p2.y = 10;
    p3.x = 0, p3.y = 11;
    p4.x = 0, p4.y = 12;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(0, 9), C=(5, -1), D=(0, 3)
    p1.x = 0, p1.y = 9;
    p2.x = 0, p2.y = 9;
    p3.x = 5, p3.y = -1;
    p4.x = 0, p4.y = 3;
```

```
    cout << no_of_quads(p1, p2, p3, p4) << endl;  
  
    return 0;  
}
```

Java

```
class GFG  
{  
    static class Point // points  
    {  
        int x;  
        int y;  
    }  
  
    // determines the orientation of points  
    static int orientation(Point p, Point q,  
                           Point r)  
    {  
        int val = (q.y - p.y) * (r.x - q.x) -  
                  (q.x - p.x) * (r.y - q.y);  
  
        if (val == 0)  
            return 0;  
        return (val > 0) ? 1 : 2;  
    }  
  
    // check whether the distinct  
    // line segments intersect  
    static boolean doIntersect(Point p1, Point q1,  
                               Point p2, Point q2)  
    {  
        int o1 = orientation(p1, q1, p2);  
        int o2 = orientation(p1, q1, q2);  
        int o3 = orientation(p2, q2, p1);  
        int o4 = orientation(p2, q2, q1);  
  
        if (o1 != o2 && o3 != o4)  
            return true;  
  
        return false;  
    }  
  
    // check if points overlap(similar)  
    static boolean similar(Point p1, Point p2)  
    {  
  
        // it is same, we are returning
```

```
// false because quadrilateral is
// not possible in this case
if (p1.x == p2.x && p1.y == p2.y)
    return false;

// it is not same, So there is a
// possibility of a quadrilateral
return true;
}

// check for collinearity
static boolean collinear(Point p1, Point p2,
                        Point p3)
{
    int x1 = p1.x, y1 = p1.y;
    int x2 = p2.x, y2 = p2.y;
    int x3 = p3.x, y3 = p3.y;

    // it is collinear, we are returning
    // false because quadrilateral is not
    // possible in this case
    if ((y3 - y2) *
        (x2 - x1) == (y2 - y1) *
        (x3 - x2))
        return false;

    // it is not collinear, So there
    // is a possibility of a quadrilateral
    else
        return true;
}

static int no_of_quads(Point p1, Point p2,
                      Point p3, Point p4)
{
    // Checking for cases where
    // no quadrilateral = 0

    // check if any of the
    // points are same
    boolean same = true;
    same = same & similar(p1, p2);
    same = same & similar(p1, p3);
    same = same & similar(p1, p4);
    same = same & similar(p2, p3);
    same = same & similar(p2, p4);
    same = same & similar(p3, p4);
```

```
// similar points exist
if (same == false)
    return 0;

// check for collinearity
boolean coll = true;
coll = coll & collinear(p1, p2, p3);
coll = coll & collinear(p1, p2, p4);
coll = coll & collinear(p1, p3, p4);
coll = coll & collinear(p2, p3, p4);

// points are collinear
if (coll == false)
    return 0;

// Checking for cases where
// no of quadrilaterals= 1 or 3

int check = 0;

if (doIntersect(p1, p2, p3, p4))
    check = 1;
if (doIntersect(p1, p3, p2, p4))
    check = 1;
if (doIntersect(p1, p2, p4, p3))
    check = 1;

if (check == 0)
    return 3;
return 1;
}

// Driver code
public static void main(String args[])
{
    Point p1, p2, p3, p4;
    p1 = new Point();
    p2 = new Point();
    p3 = new Point();
    p4 = new Point();

    // A =(0, 9), B = (-1, 0),
    // C = (5, -1), D=(5, 9)
    p1.x = 0; p1.y = 9;
    p2.x = -1; p2.y = 0;
    p3.x = 5; p3.y = -1;
    p4.x = 5; p4.y = 9;
    System.out.println(no_of_quads(p1, p2, p3, p4));
}
```

```
// A=(0, 9), B=(-1, 0),
// C=(5, -1), D=(0, 3)
p1.x = 0; p1.y = 9;
p2.x = -1; p2.y = 0;
p3.x = 5; p3.y = -1;
p4.x = 0; p4.y = 3;
System.out.println(no_of_quads(p1, p2, p3, p4));

// A=(0, 9), B=(0, 10),
// C=(0, 11), D=(0, 12)
p1.x = 0; p1.y = 9;
p2.x = 0; p2.y = 10;
p3.x = 0; p3.y = 11;
p4.x = 0; p4.y = 12;
System.out.println(no_of_quads(p1, p2, p3, p4));

// A=(0, 9), B=(0, 9),
// C=(5, -1), D=(0, 3)
p1.x = 0; p1.y = 9;
p2.x = 0; p2.y = 9;
p3.x = 5; p3.y = -1;
p4.x = 0; p4.y = 3;
System.out.println(no_of_quads(p1, p2, p3, p4));
}

}

// This code is contributed
// by Arnab Kundu
```

Output:

```
1
3
0
0
```

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/number-of-quadrilaterals-possible-from-the-given-points/>

Chapter 115

Number of terms in Geometric Series with given conditions

Number of terms in Geometric Series with given conditions - GeeksforGeeks

A geometric progression is a sequence of integers b_1, b_2, b_3, \dots , where for each $i > 1$, the respective term satisfies the condition $b_i = b_{i-1} * q$, where q is called the common ratio of the progression.

Given geometric progression b defined by two integers b_1 and q , and m “bad” integers a_1, a_2, \dots, a_m , and an integer l , write all progression terms one by one (including repetitive) while condition $|b_i| \leq l$ is satisfied ($|x|$ means absolute value of x).

Calculate how many numbers will be there in our sequence, or print “inf” in case of infinitely many integers.

Note: If a term equals one of the “bad” integers, skip it and move forward to the next term.

Examples:

```
Input : b1 = 3, q = 2, l = 30,
        m = 4
        6 14 25 48
Output : 3
The progression will be 3 12 24.
6 will also be there but because
it is a bad integer we won't include it
```

```
Input : b1 = 123, q = 1, l = 2143435
        m = 4
        123 11 -5453 141245
Output : 0
As value of q is 1, progression will
```

always be 123 and would become infinity
but because it is a bad integer we
won't include it and hence our value
will become 0

```
Input : b1 = 123, q = 1, l = 2143435
        m = 4
        5234 11 -5453 141245
Output : inf
In this case, value will be infinity
because series will always be 123 as
q is 1 and 123 is not a bad integer.
```

Approach:

We can divide our solution in different cases:

Case 1: If starting value of series is greater than the given limit, output is 0.

Case 2: If starting value of series or q is 0, there are three more cases:

Case 2.a: If 0 is not given as a bad integer, answer will become inf.

Case 2.b: If $b1 \neq 0$ but q is 0 and b1 is also not a bad integer, answer will become 1.

Case 2.c: If 0 is given as a bad integer and $b1 = 0$, answer will become 0.

Case 3: If $q = 1$ we will check if b1 is given as a bad integer or not. If it is then answer will be 0 else answer will be inf.

Case 4: If $q = -1$, check if b1 and $-b1$ is present or not, if they are present our answer will be 0 else our answer will be inf.

Case 5: If none of the above cases hold, simply run a loop for b1 till l and calculate the number of elements.

Below is the implementation of above approach:

```
// CPP program to find number of terms
// in Geometric Series
#include <bits/stdc++.h>
using namespace std;

// A map to keep track of the bad integers
map<int, bool> mapp;

// Function to calculate No. of elements
// in our series
void progression(int b1, int q, int l,
                 int m, int bad[])
{
    // Updating value of our map
    for (int i = 0; i < m; i++)
        mapp[bad[i]] = 1;

    // if starting value is greater
    // than our given limit
```

```
if (abs(b1) > 1)
    cout << "0";

// if q or starting value is 0
else if (q == 0 || b1 == 0)
{
    // if 0 is not a bad integer,
    // answer becomes inf
    if (mapp[0] != 1)
        cout << "inf";

    // if q is 0 and b1 is not and b1
    // is not a bad integer, answer becomes 1
    else if (mapp[0] == 1 && mapp[b1] != 1)
        cout << "1";

    else // else if 0 is bad integer and
          // b1 is also a bad integer,
          // answer becomes 0
        cout << "0";
}
else if (q == 1) // if q is 1
{
    // and b1 is not a bad integer,
    // answer becomes inf
    if (mapp[b1] != 1)
        cout << "inf";

    else // else answer is 0
        cout << "0";
}
else if (q == -1) // if q is -1
{
    // and either b1 or -b1 is not
    // present answer becomes inf
    if (mapp[b1] != 1 || mapp[-1 * b1] != 1)
        cout << "inf";

    else // else answer becomes 0
        cout << "0";
}
else // if none of the above case is true,
      // simply calculate the number of
      // elements in our series
{
    int co = 0;
    while (abs(b1) <= 1) {
        if (mapp[b1] != 1)
```

```
        co++;
        b1 *= 1LL * q;
    }
    cout << co;
}

// driver code
int main()
{
    // starting value of series,
    // number to be multiplied,
    // limit within which our series,
    // No. of bad integers given
    int b1 = 3, q = 2, l = 30, m = 4;

    // Bad integers
    int bad[4] = { 6, 14, 25, 48 };

    progression(b1, q, l, m, bad);

    return 0;
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/number-of-terms-in-geometric-series-with-given-conditions/>

Chapter 116

Number of ways to change the XOR of two numbers by swapping the bits

Number of ways to change the XOR of two numbers by swapping the bits - GeeksforGeeks

Given two binary strings s1 and s2. The XOR of them is X, the task is to find the number of ways to swap two-bit positions in string s1 such that XOR formed between new s1 and s2 is not same as X.

Examples:

Input: s1 = “01011”, s2 = “11001”

Output: 4

swap bits of index(1-based) (1, 4), (2, 3), (3, 4), or (3, 5) such that XOR value is changed.

Input: s1 = “011000”, s2 = “010011”

Output: 6

Approach:

1. Count the number of 1's and 0's in s1.
2. Traverse in the string s1, and check for two cases:
 - **0 and 0** in s1[i] and s2[i], as replacing 0 with 1, will change the XOR value.
 - **1 and 0** in s1[i] and s2[i], as replacing 1 with 0 will change the XOR value.
3. For the first case, the number of ways of replacement will be the number of ones-already used 1's.
4. For the second case, the number of ways of replacement will be the number of zeros-already used 0's.
5. summation of number of ways in both the cases will be the answer.

Below is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Function that returns the number of
// bit swaps such that xor is different
int countWays(string s1, string s2)
{
    int c1 = 0, c0 = 0;
    int n = s1.length();

    // traverse and count 1's and 0's
    for (int i = 0; i < n; i++) {
        if (s1[i] == '1')
            c1++;
        else
            c0++;
    }
    int used1 = 0, used0 = 0;
    int ways = 0;

    // traverse in the string
    for (int i = 0; i < n; i++) {

        // if both positions are 0
        if (s1[i] == '0' and s2[i] == '0') {

            // add the number of ones as
            // it will change the XOR
            ways += c1;

            // subtract the number of ones already used
            ways -= used1;

            // zeros have been used
            used0++;
        }

        // when 1 and 0, to change XOR, we have to
        // replace 1 by 0
        else if (s1[i] == '1' and s2[i] == '0') {

            // add number of 0's
            ways += c0;
        }
    }
}
```

```
// subtract number of 0's already used
ways -= used0;

// count 1's used
used1++;
}
}

// return the answer
return ways;
}

// Driver Code
int main()
{
    string s1 = "01011";
    string s2 = "11001";

    cout << countWays(s1, s2);
    return 0;
}
```

Java

```
// Java Program to find Number of
// ways to change the XOR of two
// numbers by swapping the bits
class GFG
{
// Function that returns the
// number of bit swaps such
// that xor is different
static int countWays(String s1,
                      String s2)
{
    int c1 = 0, c0 = 0;
    int n = s1.length();

    // traverse and count 1's and 0's
    for (int i = 0; i < n; i++)
    {
        if (s1.charAt(i) == '1')
            c1++;
        else
            c0++;
    }
    int used1 = 0, used0 = 0;
    int ways = 0;
```

```
// traverse in the String
for (int i = 0; i < n; i++)
{
    // if both positions are 0
    if (s1.charAt(i) == '0' &&
        s2.charAt(i) == '0')
    {
        // add the number of ones as
        // it will change the XOR
        ways += c1;

        // subtract the number of
        // ones already used
        ways -= used1;

        // zeros have been used
        used0++;
    }

    // when 1 and 0, to change XOR,
    // we have to replace 1 by 0
    else if (s1.charAt(i) == '1' &&
              s2.charAt(i) == '0')
    {
        // add number of 0's
        ways += c0;

        // subtract number of
        // 0's already used
        ways -= used0;

        // count 1's used
        used1++;
    }
}

// return the answer
return ways;
}

// Driver Code
public static void main(String[] args)
{
    String s1 = "01011";
```

```
String s2 = "11001";  
  
        System.out.println(countWays(s1, s2));  
    }  
}  
  
// This code is contributed  
// by Arnab Kundu
```

Python3

```
# Function that returns the number of  
# bit swaps such that xor is different  
def countWays(s1, s2):  
  
    c1 = 0  
    c0 = 0  
    n = len(s1)  
  
    # traverse and count 1's and 0's  
    for i in range(0,n) :  
        if (s1[i] == '1'):  
            c1+=1  
        else:  
            c0+=1  
  
    used1 = 0  
    used0 = 0  
    ways = 0  
  
    # traverse in the string  
    for i in range(0,n) :  
  
        # if both positions are 0  
        if (s1[i] == '0' and s2[i] == '0') :  
  
            # add the number of ones as  
            # it will change the XOR  
            ways += c1  
  
            # subtract the number of ones already used  
            ways -= used1  
  
            # zeros have been used  
            used0+=1  
  
        # when 1 and 0, to change XOR, we have to
```

```
# replace 1 by 0
elif (s1[i] == '1' and s2[i] == '0') :

    # add number of 0's
    ways += c0

    # subtract number of 0's already used
    ways -= used0

    # count 1's used
    used1+=1

# return the answer
return ways

# Driver Code
if __name__=='__main__':
    s1 = "01011"
    s2 = "11001"
    print(countWays(s1, s2))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# Program to find Number of
// ways to change the XOR of two
// numbers by swapping the bits
using System;

class GFG
{
// Function that returns the
// number of bit swaps such
// that xor is different
static int countWays(String s1,
                      String s2)
{
    int c1 = 0, c0 = 0;
    int n = s1.Length;

    // traverse and count 1's and 0's
    for (int i = 0; i < n; i++)
    {
        if (s1[i] == '1')
            c1++;
        else
            c0++;
    }
}
```

```
}

int used1 = 0, used0 = 0;
int ways = 0;

// traverse in the String
for (int i = 0; i < n; i++)
{

    // if both positions are 0
    if (s1[i] == '0' &&
        s2[i] == '0')
    {

        // add the number of ones as
        // it will change the XOR
        ways += c1;

        // subtract the number of
        // ones already used
        ways -= used1;

        // zeros have been used
        used0++;
    }

    // when 1 and 0, to change XOR,
    // we have to replace 1 by 0
    else if (s1[i] == '1' &&
              s2[i] == '0')
    {

        // add number of 0's
        ways += c0;

        // subtract number of
        // 0's already used
        ways -= used0;

        // count 1's used
        used1++;
    }
}

// return the answer
return ways;
}

// Driver Code
```

```
public static void Main(String[] args)
{
    String s1 = "01011";
    String s2 = "11001";

    Console.WriteLine(countWays(s1, s2));
}
}

// This code is contributed
// by Subhadeep Gupta
```

Output:

4

Time Complexity: O(N)

Improved By : [andrew1234](#), [tufan_gupta2000](#), [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/number-of-ways-to-change-the-xor-of-two-numbers-by-swapping-the-bits/>

Chapter 117

Ordered Set and GNU C++ PBDS

Ordered Set and GNU C++ PBDS - GeeksforGeeks

Prerequisite :Basic knowledge of [STL](#) and [Sets Data structure](#).

About ordered set

Ordered set is a [policy based data structure in g++](#) that keeps the **unique** elements in sorted order. It performs all the operations as performed by the set data structure in STL in $\log(n)$ complexity and performs two additional operations also in $\log(n)$ complexity .

- **order_of_key (k)** : Number of items strictly smaller than k .
- **find_by_order(k)** : K-th element in a set (counting from zero).

Required header files to implement ordered set and their description

For implementing `ordered_set` and GNU C++ library contains other Policy based data structures we need to include :

- // Common file
`include <ext/pb_ds/assoc_container.hpp>`
- // Including tree_order_statistics_node_update
`include <ext/pb_ds/tree_policy.hpp>`

The first one is used to include the associative containers or group of templates such as **set**, **multimap**, **map** etc. The tree-based data structures which we will be using below is present in this header file.

The second header file is used to include the *tree_order_statistics_node_update* which is explained below:

```
using namespace __gnu_pbds;
```

It is a namespace necessary for the **GNU based Policy based data structures**.

The tree based container has a concrete structure but the necessary structure required for the ordered set implementation is :

```
tree < int , null_type , less , rb_tree_tag , tree_order_statistics_node_update >
```

1. **int** : It is the type of the data that we want to insert (KEY).It can be integer, float or pair of int etc.
2. **null_type** : It is the mapped policy. It is null here to use it as a set.If we want to get map but not the set, as the second argument type must be used mapped type.
3. **less** : It is the basis for comparison of two functions.
4. **rb_tree_tag** : type of tree used. It is generally Red black trees because it takes $\log(n)$ time for insertion and deletion while other take linear time such as splay_tree.
5. **tree_order_statistics_node_update** : It is included in tree_policy.hpp and contains various operations for updating the node variants of a tree-based container, so we can keep track of metadata like the number of nodes in a subtree

Additional functions in the ordered set other than the set

Along with the previous operations of the set, it supports *two* main important operations

- **find_by_order(k)**: It returns to an iterator to the kth element (**counting from zero**) in the set in $O(\log n)$ time.To find the first element k must be zero.

Let us assume we have a set s : {1, 5, 6, 17, 88}, then :

`*s.find_by_order(2)` : 3rd element in the set i.e. 6
`*s.find_by_order(4)` : 5th element in the set i.e. 88

- **order_of_key(k)** : It returns to the number of items that are **strictly** smaller than our item k in $O(\log n)$ time.

Let us assume we have a set s : {1, 5, 6, 17, 88}, then :

`s.order_of_key(6)` : Count of elements **strictly** smaller than 6 is 2.
`s.order_of_key(25)` : Count of elements **strictly** smaller than 25 is 4.

Difference between set and ordered set

There is not so much difference between the set and ordered set although ordered set can be assumed as an extended version of set capable of performing some more advanced functions(stated above) that are extensively used in competitive programming.

NOTE : `ordered_set` is used here as a macro given to `tree<int, null_type, less, rb_tree_tag, tree_order_statistics_node_update>`. Therefore it can be given any name as *macro* other than `ordered_set` but generally in the world of competitive programming it is commonly referred as ordered set as it is a set with additional operations.

Practical applications:

Suppose we have a situation where the elements are inserted one by one in an array and after each insertion, we are given a range $[l, r]$ and we have to determine the number of elements in the array greater than equal to l and less than equal to r. Initially, the array is empty.

Examples:

Input : 5
 1 2
 1
 2 5
 2
 1 5

Output : 0
 1
 3

Explanation:

- 5 is inserted.
- Count of elements greater than equal to 1 and less than equal to 2 is 0.
- 1 is inserted.
- Count of elements greater than equal to 2 and less than equal to 5 is 1 i.e. 5.
- 2 is inserted.
- Count of elements greater than equal to 1 and less than equal to 5 is 3 i.e. 5, 1, 2.

Input : 1
 1 2
 2
 3 5
 5
 1 4
Output : 1
 0
 2

- 1 is inserted.
- Count of elements greater than equal to 1 and less than equal to 2 is 1 i.e 1.
- 2 is inserted.
- Count of elements greater than equal to 3 and less than equal to 5 is 0.
- 5 is inserted.
- Count of elements greater than equal to 1 and less than equal to 4 is 2 i.e. 1, 2.

2
2
2
5
1

Thus we can now solve the above problem easily i.e. count of elements between l and r can be found by:

`o_set.order_of_key(r+1) - o_set.order_of_key(l)`

NOTE : As the set contains only the **UNIQUE** elements, so to perform the operations on an array having repeated elements we can take the KEY as a pair of elements instead of integer in which the first element is our required element of the array and only the second element of the pair must be unique so that the whole pair is unique.

For more details refer to :

https://gcc.gnu.org/onlinedocs/libstdc++/manual/policy_data_structures.html

Source

<https://www.geeksforgeeks.org/ordered-set-gnu-c-pbds/>

Chapter 118

Pair of arrays with equal sum after removing exactly one element from each

Pair of arrays with equal sum after removing exactly one element from each - GeeksforGeeks

Given K arrays of different size. The task is to check if there exist any two arrays which have the same sum of elements after removing exactly one element from each of them. (Any element can be removed, but exactly *one* has to be removed). Print the indices of the array and the index of the removed elements if such pairs exist. If there are multiple pairs, print any one of them. If no such pairs exist, print -1.

Examples:

Input: k = 3

a1 = {8, 1, 4, 7, 1}

a2 = {10, 10}

a3 = {1, 3, 4, 7, 3, 2, 2}

Output: Array 1, index 4

Array 3, index 5

sum of Array 1{8, 1, 4, 7, 1} without index 4 is 20.

sum of Array 3{1, 3, 4, 7, 3, 2, 2} without index 5 is 20.

Input: k = 4

a1 = {2, 4, 6, 6}

a2 = {1, 2, 4, 8, 16}

a3 = {1, 3, 8}

a4 = {1, 4, 16, 64}

Output: -1

Brute Force:

For every pair of arrays, for each element, find the sum excluding that element and compare

it with the sum excluding each element one by one in the second array of the chosen pair. (Here $\max l$ denotes the maximum length of an array in the set).

Time Complexity :

$O(n^2 \log n)$ (Because we are using sorting)

Space Complexity :

$O(n^2)$ (Because we are using sorting)

Efficient Approach: Precompute all possible values of the sum obtained by removing one element from each of the arrays. Store the array index and element index which is removed with the computed sum. When these values are arranged in increasing order, it can easily be seen that if a solution exists, then both the sum values must be adjacent to the new arrangement. When two adjacent sum values are same, check if they belong to different arrays. If they do, print the array number and index of the element removed. If no such sum value is found, then no such pairs exist.

Below is the implementation of the above approach:

```
// C++ program to print the pair of arrays
// whose sum are equal after removing
// exactly one element from each
#include <bits/stdc++.h>
using namespace std;

// Function to print the pair of array and index
// of element to be removed to make sum equal
void printPair(vector<int> a[], int k)
{

    // stores the sum removing one element,
    // array number and index of removed element
    vector<pair<int, pair<int, int>> ans;

    // traverse in every array
    for (int i = 0; i < k; i++) {

        // length of array
        int l = a[i].size();

        int sum = 0;

        // compute total sum of array
        for (int j = 0; j < l; j++) {
            sum = sum + a[i][j];
        }

        // remove each element once and insert sum in
        // ans vector along with index
        for (int j = 0; j < l; j++) {
            ans.push_back({sum - a[i][j], {i + 1, j}});
        }
    }
}
```

```
        }

    }

    // sort the ans vector so that
    // same sum values after removing
    // a single element comes together
    sort(ans.begin(), ans.end());

    bool flag = false;

    // iterate and check if any adjacent sum are equal
    for (int p = 1; p < ans.size(); p++) {

        // check if the adjacent sum belong to different array
        // if the adjacent sum is equal
        if (ans[p - 1].first == ans[p].first
            && ans[p - 1].second.first != ans[p].second.first) {

            // first array number
            int ax = ans[p - 1].second.first;

            // element's index removed from first array
            int aidx = ans[p - 1].second.second;

            // second array number
            int bx = ans[p].second.first;

            // element's index removed from second array
            int bidx = ans[p].second.second;

            cout << "Array " << ax << ", index " << aidx << "\n";
            cout << "Array " << bx << ", index " << bidx << "\n";

            flag = true;
            break;
        }
    }

    // If no pairs are found
    if (!flag)
        cout << "No special pair exists\n";
}

// Driver Code
int main()
{
    // k sets of array
    vector<int> a[] = {
```

```
{ 8, 1, 4, 7, 1 },
{ 10, 10 },
{ 1, 3, 4, 7, 3, 2, 2 }
};

int k = sizeof(a) / sizeof(a[0]);

// Calling Function to print the pairs if any
printPair(a, k);

return 0;
}
```

Output:

```
Array 1, index 4
Array 3, index 5
```

Time Complexity : ~~$O(n^2)$~~ , or simply, $\Theta(n^2 \log n)$

Space Complexity : ~~$O(n^2)$~~ , or simply, $\Theta(n^2 \log n)$

Source

<https://www.geeksforgeeks.org/pair-of-arrays-with-equal-sum-after-removing-exactly-one-element-from-each/>

Chapter 119

Pair with minimum absolute difference after solving each query

Pair with minimum absolute difference after solving each query - GeeksforGeeks

Given Q queries and an empty list.

The queries can be of two types:

1. addToList(x) : Add x to your list.
2. removeFromList(x) : Remove x from your list.

The task is, after each query, you have to print the minimum value of $\text{abs}(\text{list}[i]-\text{list}[j])$ where, $0 \leq i \leq n$, $0 \leq j \leq n$ and $i \neq j$ and n is the total number of elements present in the list.

Examples:

Input : Q = 4

addToList(1), insert 1 in our set

addToList(5), insert 5 in our set

addToList(3), insert 3 in our set

removeFromList(3), delete 3 in our set

Output :

0, as we have only one element {1} in our set.

4, as we have {1, 5} minimum difference between all possible pairs is 4 (5-1)

2, as we have {1, 3, 5} minimum difference between all possible pairs is 2 (3-1)

4, as we have {1, 5} minimum difference between all possible pairs is 4 (5-1)

Method 1: The idea is to use set in C++ to store all of the elements so that insertion or deletion can be done in $O(\log(n))$ with keeping the elements in a sorted order. Now, to find the minimum difference, we have to iterate over the whole set and find the difference

between only adjacent elements as elements are in sorted order i.e, the minimum difference will always be contributed by adjacent elements. This can be done in $O(n)$ time complexity. So, the overall time complexity of this approach will be $(q * \log(n) + q * n)$.

Method 2:

We can use multiset to store and maintain all of the elements and map to store the difference between the adjacent elements in sorted order, where, map's key represents the difference between adjacent elements and the corresponding map's value represent the frequency of occurrence of this difference.

Below is the complete **algorithm**:

First of all, insert two sentinel value in the multi-set. Let's say $(-10^{9+7}$ and $10^{9+7})$ and initialize the map with the difference of these sentinel value, i.e $2 * 10^{9+7}$ and set its frequency to be 1.

Now we have two type of queries:

1. **Insertion Query:** For an insertion query, first insert the *value* in set. After inserting the element, we will have to also update the differences of adjacent elements in the map. To do this, find the *left* and *right* values of the newly inserted element in the set. Earlier when this new value was not inserted in the set, $\text{abs}(\text{right-left})$ is contributing to the difference term in the map. To update the map after inserting the new element, first remove the previous difference $\text{abs}(\text{right-left})$ as a new value is inserted between the elements left and right and add two differences to the map. i.e, $\text{abs}(\text{right-x})$ and $\text{abs}(\text{x-left})$.
2. **Deletion Query:** In case of deletion of a value from the set. First, find the *left* and *right* elements of the element needed to be deleted, say *x*. Then according to the above algorithm, reduce the frequency of $\text{abs}(\text{right-x})$ and $\text{abs}(\text{left-x})$ and increment the frequency of $\text{abs}(\text{right-left})$.

So, each query can be solved in $\log(n)$ time complexity. Therefore overall complexity will be $O(q * \log(n))$.

```
// C++ implementation of above approach
#include <bits/stdc++.h>
#define ll long long int
const ll MOD = 1e9 + 7;
using namespace std;

// Function to add an element into the list
void addToList(int x, multiset<ll>& s, map<ll, ll>& mp)
{
    // firstly insert value in set
    s.insert(x);

    // find left and right value of inserted value.
    ll left, right;

    // now here is logic explained below
```

```
left = *(--s.find(x));
right = *(++s.find(x));
mp[abs(left - x)]++;
mp[abs(right - x)]++;
mp[abs(left - right)]--;
if (mp[abs(left - right)] == 0)
    mp.erase(abs(left - right));
}

// Function to remove an element from the list
void removeFromList(int x, multiset<ll>& s, map<ll, ll>& mp)
{
    ll left, right;

    // Find left element of number that we want to delete.
    left = *(--s.find(x));

    // Find right element of number that we want to delete.
    right = *(++s.find(x));

    // remove x from set
    s.erase(s.find(x));

    // Again this will explain below in article.
    mp[abs(left - x)]--;
    if (mp[abs(left - x)] == 0)
        mp.erase(abs(left - x));
    mp[abs(right - x)]--;
    if (mp[abs(right - x)] == 0)
        mp.erase(abs(right - x));

    mp[abs(left - right)]++;
}

// Driver code
int main()
{
    int q = 4;

    // define set to store all values.
    multiset<ll> s;

    // define map to store difference in sorted
    map<ll, ll> mp;

    // initialize set with sentinel values
    s.insert(-MOD);
```

```
s.insert(MOD);

// maintain freq of differnce in map so, here intially
// difference b/w sentinel value and its freq is one.
mp[2 * MOD] = 1;

// 1st query 1 1 i.e, include 1 in our set
addToList(1, s, mp);

// As now we have only one element {-10^9+7, 1, 10^9+7}
// (except two are sentinel values)
// so minimum among all pair is zero
cout << 0 << endl;

// 2nd query 1 5 i.e, include 5 in our set.
addToList(5, s, mp);

// find smallest difference possible
// as it should be in begining of map
cout << mp.begin()->first << endl;

// 3rd query 1 3 i.e, include 3 in our set.
addToList(3, s, mp);

// find smallest difference possible
// as it should be in beginning of map
cout << mp.begin()->first << endl;

// 4th query 2 3 i.e, remove 3 from our list
removeFromList(3, s, mp);

cout << mp.begin()->first << endl;

return 0;
}
```

Output:

```
0
4
2
4
```

Explanation of each query:

1. **addToList(1):** As now we have only one element $\{-10^{9+7}, 1, 10^{9+7}\}$ (except two are sentinel values), so minimum among all pair is zero.

2. ***addToList(5)***: After inserting 1 and 5 to the set, set becomes $\{-10 \sim 9+7, 1, 5, 10 \sim 9+7\}$ and map: $mp[5-1]=1$ where key represents difference and its value represents frequency.
3. ***addToList(3)***: As here we are inserting 3. So, first of all, we find the left and right element of 3 in the set after insertion i.e. $left = 1$, $right = 5$. As 3 came in between 1 and 5 so we remove $map[5-1]-$. (as 3 came in between 1 and 5 so **5-1** will no longer minimum) And we include these differences $map[3-1]++$ and $map[5-3]++$ (because 3 came between 1 and 5, so possible minimum have two cases).
4. ***removeFromList(3)***: Similar as of above query, here we are deleting element. So, first of all, find the left and right element to 3 i.e. $left = 1$, $right = 5$. As we going to delete 3 which is in between 1 & 5 so minimum difference will be affected only of 3 and 5,

```
before deleting ==>{1, 3, 5},  
after deleting ==> {1, 5}.  
So, map[3-1]--, map[5-3]-- and add map[5-1]++.
```

Source

<https://www.geeksforgeeks.org/pair-with-minimum-absolute-difference-after-solving-each-query/>

Chapter 120

Pairs involved in Balanced Parentheses

Pairs involved in Balanced Parentheses - GeeksforGeeks

Given a string of brackets, task is to find the number of pairs of brackets involved in a balanced sequence in a given range.

Examples :

```
Input : ((())()
Range : 1 5
Range : 3 8
Output : 2
         2
Explanation : In range 1 to 5 ((()),
there are the two pairs. In range 3 to 8 (),
(), there are the two pairs.
```

```
Input : )()())
Range : 1 2
Range : 4 7
Output : 0
         1
Explanation : In range 1 to 2 )( there
is no any pair. In range 4 to 7 (()),
there is the only pair
```

Prerequisite : Segment Trees

Approach :

Here, in segment tree, for each node, keep some simple elements, like integers or sets or

vectors or etc.

For each node keep three integers :

1. t = Answer for the interval.
2. o = The number of opening brackets '(' remaining after deleting the brackets those who belong to the correct bracket sequence in this interval with length t.
3. c = The number of closing brackets ')' remaining after deleting the brackets those who belong to the correct bracket sequence in this interval with length t.

Now, having these variables, queries can be answered easily using segment tree.

Below is the implementation of above approach :

```
// CPP code to find the number of pairs
// involved in balanced parantheses
#include <bits/stdc++.h>
using namespace std;

// Our struct node
struct node {

    // three variables required
    int t, o, c;
};

// Declare array of nodes of very big
// size which acts as segment tree here.
struct node tree_arr[5 * 1000];

// To build a segment tree we pass 1 as
// 'id' 0 as 'l' and 1 as 'n'.
// Here, we consider query's interval as [x, y)
void build(int id, int l, int r, string s)
{
    /* this base condition is common to
       build any segment tree*/
    // This is the base
    // Only one element left
    if (r - l < 2) {

        // If that element is open bracket
        if (s[l] == '(')
            tree_arr[id].o = 1;

        // If that element is open bracket
        else
            tree_arr[id].c = 1;
        return;
    }

    // Next three lines are common
```

```

// for any segment tree.
int mid = (l + r) / 2;

// for left tree
build(2 * id, l, mid, s);

// for right tree
build(2 * id + 1, mid, r, s);

// Here we take minimum of left tree
// opening brackets and right tree
// closing brackets
int tmp = min(tree_arr[2 * id].o,
              tree_arr[2 * id + 1].c);

// we add that to our answer.
tree_arr[id].t = tree_arr[2 * id].t +
                 tree_arr[2 * id + 1].t + tmp;

// Remove the answer from opening brackets
tree_arr[id].o = tree_arr[2 * id].o +
                 tree_arr[2 * id + 1].o - tmp;

// Remove the answer from opening brackets
tree_arr[id].c = tree_arr[2 * id].c +
                 tree_arr[2 * id + 1].c - tmp;
}

// This will return the answer for each query.
// Here we consider query's interval as [x, y)
node segment(int x, int y, int id,
             int l, int r, string s)
{
    // If the given interval is out of range
    if (l >= y || x >= r) {
        struct node tem;
        tem.t = 0;
        tem.o = 0;
        tem.c = 0;
        return tem;
    }

    // If the given interval completely lies
    if (x <= l && r <= y)
        return tree_arr[id];

    // Next three lines are common for
    // any segment tree.

```

```
int mid = (l + r) / 2;

// For left tree
struct node a =
    segment(x, y, 2 * id, l, mid, s);

// For right tree
struct node b =
    segment(x, y, 2 * id + 1, mid, r, s);

// Same as made in build function
int temp;
temp = min(a.o, b.c);
struct node vis;
vis.t = a.t + b.t + temp;
vis.o = a.o + b.o - temp;
vis.c = a.c + b.c - temp;

return vis;
}

// Driver code
int main()
{
    string s = "((())(()";
    int n = s.size();

    // range for query
    int a = 3, b = 8;

    build(1, 0, n, s);

    // Here we consider query's interval as [a, b)
    // We subtract 1 from 'a' because indexes start
    // from 0.
    struct node p = segment(a-1, b, 1, 0, n, s);
    cout << p.t << endl;

    return 0;
}
```

Output:

Source

<https://www.geeksforgeeks.org/pairs-involved-balanced-parentheses/>

Chapter 121

Pairs with GCD equal to one in the given range

Pairs with GCD equal to one in the given range - GeeksforGeeks

Given a range i.e. L and R, the task is to find if we can form pairs such that GCD of every pair is 1. Every number in the range L-R should be involved exactly in one pair.

Examples:

Input: L = 1, R = 8

Output: Yes

{2, 7}, {4, 1}, {3, 8}, {6, 5}

All pairs have GCD as 1.

Input: L = 2, R = 4

Output: No

Approach: Since every number in the range(L, R) must be included exactly once in every pair. Hence if L-R is an even number, then it is not possible. If L-R is an odd number, then print all the adjacent pairs since adjacent pairs will always have GCD as 1.

Below is the implementation of the above approach:

C++

```
// C++ program to print all pairs
#include <bits/stdc++.h>
using namespace std;

// Function to print all pairs
bool checkPairs(int l, int r)
```

```
{  
    // check if even  
    if ((l - r) % 2 == 0)  
        return false;  
  
    /* We can print all adjacent pairs  
     * for (int i = l; i < r; i += 2) {  
     *     cout << "{" << i << ", " << i + 1 << "}, "  
     * } */  
  
    return true;  
}  
  
// Driver Code  
int main()  
{  
    int l = 1, r = 8;  
    if (checkPairs(l, r))  
        cout << "Yes";  
    else  
        cout << "No";  
    return 0;  
}
```

Java

```
// Java program to print all pairs  
class GFG  
{  
    // Function to print all pairs  
    static boolean checkPairs(int l, int r)  
{  
        // check if even  
        if ((l - r) % 2 == 0)  
            return false;  
  
        /* We can print all adjacent pairs  
         * for (int i = l; i < r; i += 2)  
         * {  
         *     System.out.print("{" + i + ", " + i + 1 + "}, ");  
         * } */  
  
        return true;  
    }  
  
    // Driver Code  
    public static void main(String[] args)  
    {
```

```
int l = 1, r = 8;
if (checkPairs(l, r))
    System.out.println("Yes");
else
    System.out.println("No");
}
}

// This code is contributed by mits
```

Python 3

```
# Python 3 program to print all pairs

# Function to print all pairs
def checkPairs(l, r) :

    # check if even
    if (l - r) % 2 == 0 :
        return False

    """ we can print all adjacent pairs
    for i in range(l,r,2) :
        print("{",i,",",i + 1, "},")
    """

    return True

# Driver Code
if __name__ == "__main__":
    l, r = 1, 8

    if checkPairs(l, r) :
        print("Yes")
    else :
        print("No")

# This code is contributed by ANKITRAI1
```

Output:

Yes

Time Complexity: O(N)

Auxiliary Space: O(1)

Improved By : [Mithun Kumar](#), [ANKITRAI1](#)

Source

<https://www.geeksforgeeks.org/pairs-with-gcd-equal-to-one-in-the-given-range/>

Chapter 122

Palindromic Tree | Introduction & Implementation

Palindromic Tree | Introduction & Implementation - GeeksforGeeks

We encounter various problems like Maximum length palindrome in a string, number of palindromic substrings and many more interesting problems on palindromic substrings . Mostly of these palindromic substring problems have some DP $O(n^2)$ solution (n is length of the given string) or then we have a complex algorithm like [Manacher's algorithm](#) which solves the Palindromic problems in linear time.

In this article, we will study an interesting Data Structure, which will solve all the above similar problems in much more simpler way. This data structure is invented by [Mikhail Rubinchik](#).

Features of Palindromic Tree : Online query and updation
Easy to implement
Very Fast

Structure of Palindromic Tree

Palindromic Tree's actual structure is **close to directed graph**. It is actually a merged structure of two Trees which share some common nodes(see the figure below for better understanding). Tree nodes store palindromic substrings of given string by storing their indices.

This tree consists of two types of edges :

- 1) Insertion edge (weighted edge)
- 2) Maximum Palindromic Suffix (un-weighted)

Insertion Edge :

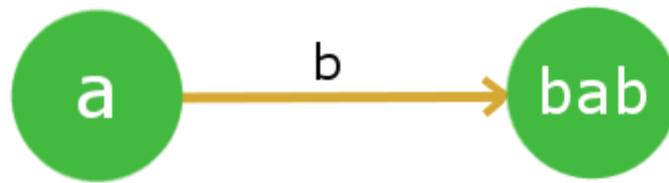
Insertion edge from a node **u** to **v** with some weight **x** means that the node v is formed by inserting x at the front and end of the string at u. As u is already a palindrome, hence the resulting string at node v will also be a palindrome.

x will be a single character for every edge. Therefore, a node can have max 26 insertion edges (considering lower letter string). We will use orange color for this edge in our pictorial representation.

Maximum Palindromic Suffix Edge:

As the name itself indicates that for a node this edge will point to its Maximum Palindromic Suffix String node. We will not be considering the complete string itself as the Maximum Palindromic Suffix as this will make no sense(self loops). For simplicity purpose, we will call it as Suffix edge(by which we mean maximum suffix except the complete string). It is quite obvious that every node will have only 1 Suffix Edge as we will not store duplicate strings in the tree. We will use Blue dashed edges for its Pictorial representation.

Insertion Edge



Maxi

aba

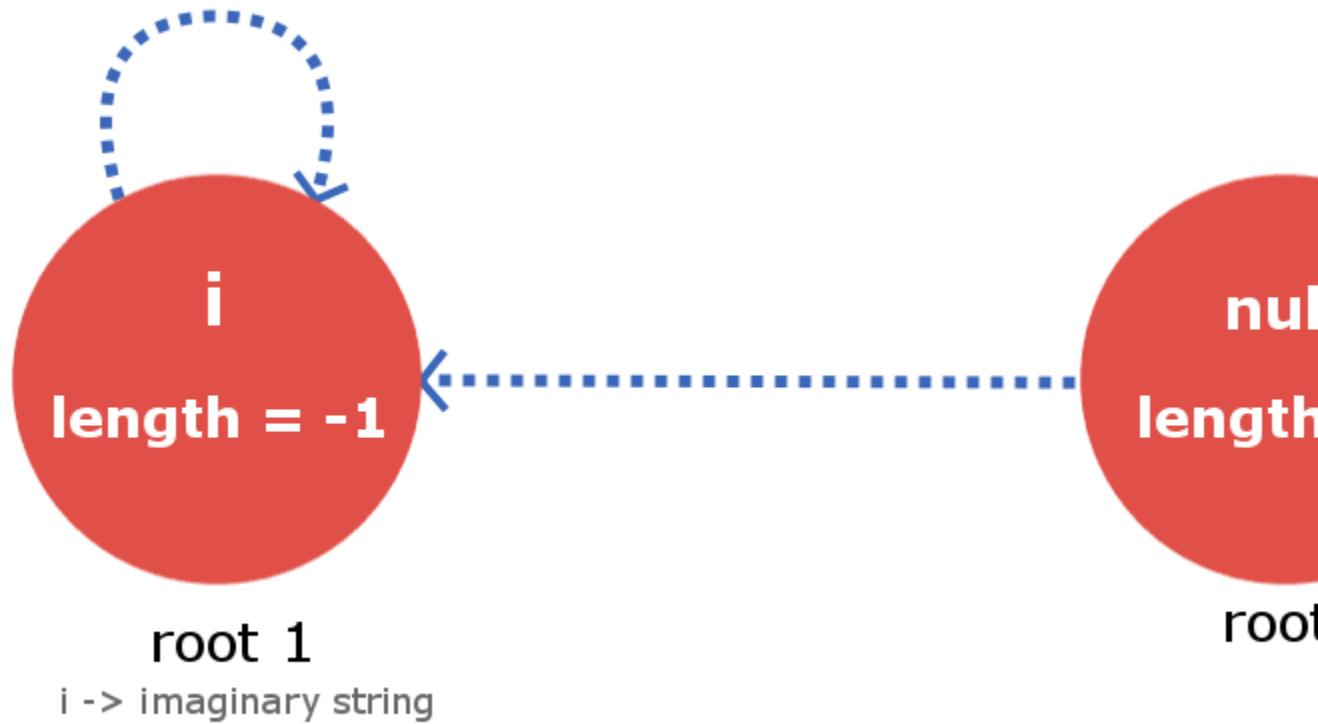
Root Nodes and their convention:

This tree/graph data structure will contain **2 root dummy nodes**. More, precisely consider it as roots of two separate trees, which are linked together.

Root-1 will be a dummy node which will describe a string for $length = -1$ (you can easily infer from the implementation point of view that why we used so). *Root-2* will be a node which will describe a null string of $length = 0$.

Root-1 has a suffix edge connected to itself(self-loop) as for any imaginary string of length -1 , its Maximum palindromic suffix will also be imaginary, so this is justified. Now *Root-2*

will also have its suffix edge connected to Root-1 as for a null string (length 0) there is no real palindromic suffix string of length less than 0.



Building the Palindromic Tree

To build a Palindromic Tree, we will simply insert the characters in our string one by one till we reach its end and when we are done inserting we will be with our palindromic tree which will contain all the distinct palindromic substrings of the given strings. All we need to ensure is that, at every insertion of a new character, our palindromic tree maintains the above discussed feature. Let's see how we can accomplish it.

Let's say we are given a string s with length l and we have inserted the string up till index k ($k < l-1$). Now, we need to insert the $(k+1)$ th character. Insertion of $(k+1)$ th character means insertion of a node that is longest palindrome ending at index $(k+1)$. So, the longest palindromic string will be of form (' $s[k+1]$ ' + "X" + ' $s[k+1]$ ') and X will itself be a palindrome. Now the fact is that the string X lies at index $< k+1$ and is palindrome. So, it will already exist in our palindromic tree as we have maintained the very basic property of it saying that it will contain all the distinct palindromic substrings.

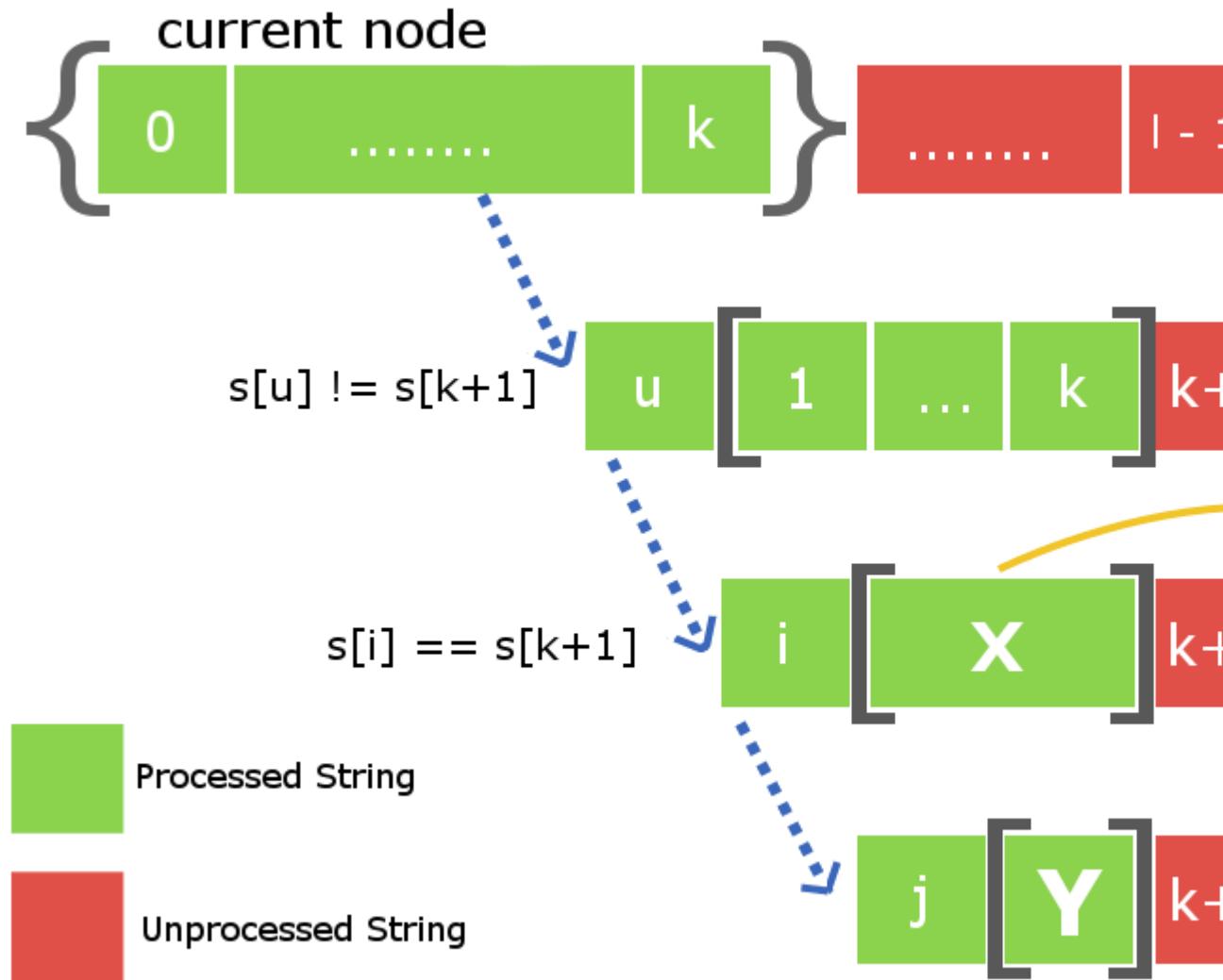
So, to insert the character $s[k+1]$, we only need to find the String X in our tree and direct the insertion edge from X with weight $s[k+1]$ to a new node, which contains $s[k+1]+X+s[k+1]$. The main job now is to find the string X in efficient time. As we know that we are storing the suffix link for all the nodes. Therefore to track the node with

string X we just need to move down the suffix link for the current node i.e the node which contains insertion of $s[k]$. See the below image for better understanding.

The current node in the below figure tells that it is the largest palindrome that ends at index k after processing all the indices from 0 to k. The blue dotted path is the link of suffix edges from current node to other processed nodes in the tree. String X will exist in one of these nodes that lie on this chain of suffix link. All we need is to find it by iterating over it down the chain.

To find the required node that contains the string X we will place the $k+1$ th character at the end of every node that lies in suffix link chain and check if first character of the corresponding suffix link string is equal to the $k+1$ th character.

Once, we find the X string we will direct an insertion edge with weight $s[k+1]$ and link it to the new node that contains largest palindrome ending at index $k+1$. The array elements between the brackets as described in the figure below are the nodes that are stored in the tree.



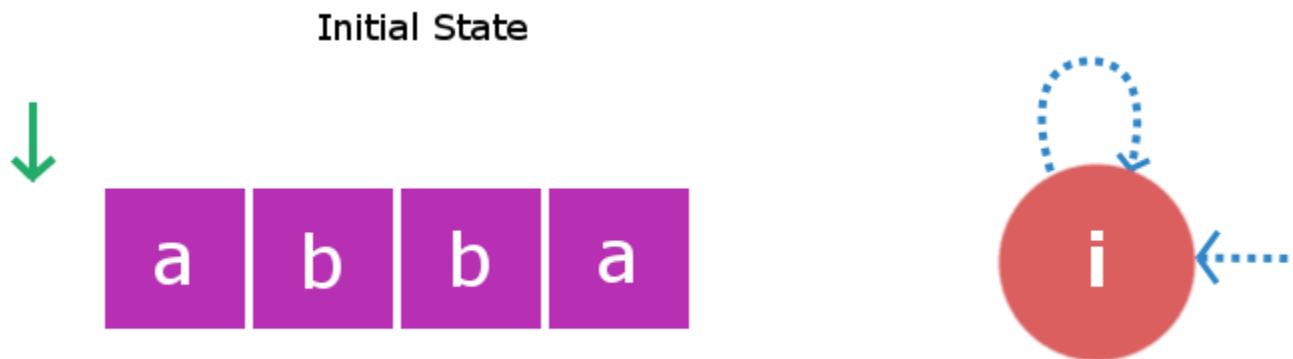
There is one more thing left that is to be done. As we have created a new node at this $s[k+1]$ insertion, therefore we will also have to connect it with its suffix link child. Once, again to do so we will use the above down the suffix link iteration from node X to find some new string Y such that $s[k+1] + Y + s[k+1]$ is a largest palindromic suffix for the newly created node. Once, we find it we will then connect the suffix link of our newly created node with the node Y.

Note : There are two possibilities when we find the string X. First possibility is that string $s[k+1]Xs[k+1]$ do not exist in the tree and second possibility is if it already exists in the tree. In first case we will proceed the same way but in second case we will not

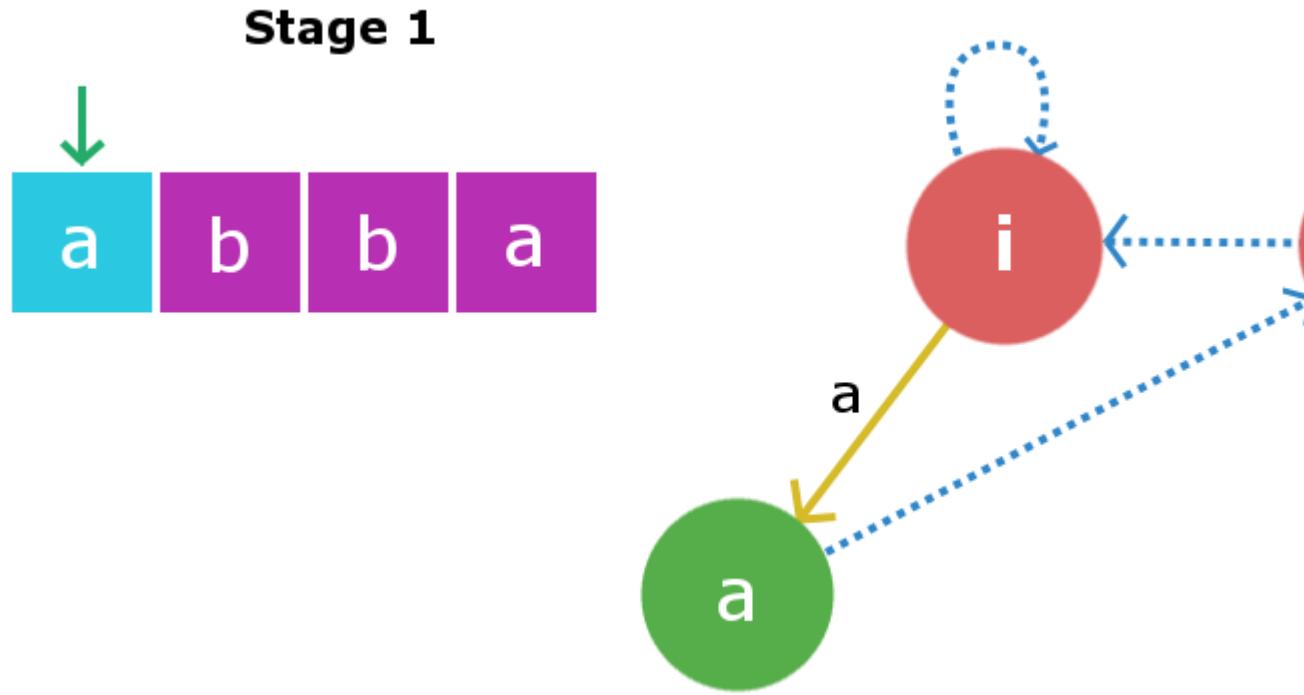
create a new node separately but will just link the insertion edge from X to already existing $S[k+1]+X+S[k+1]$ node in the tree. We also need not to add the suffix link because the node will already contain its suffix link.

Consider a string $s = \text{"abba"}$ with $length = 4$.

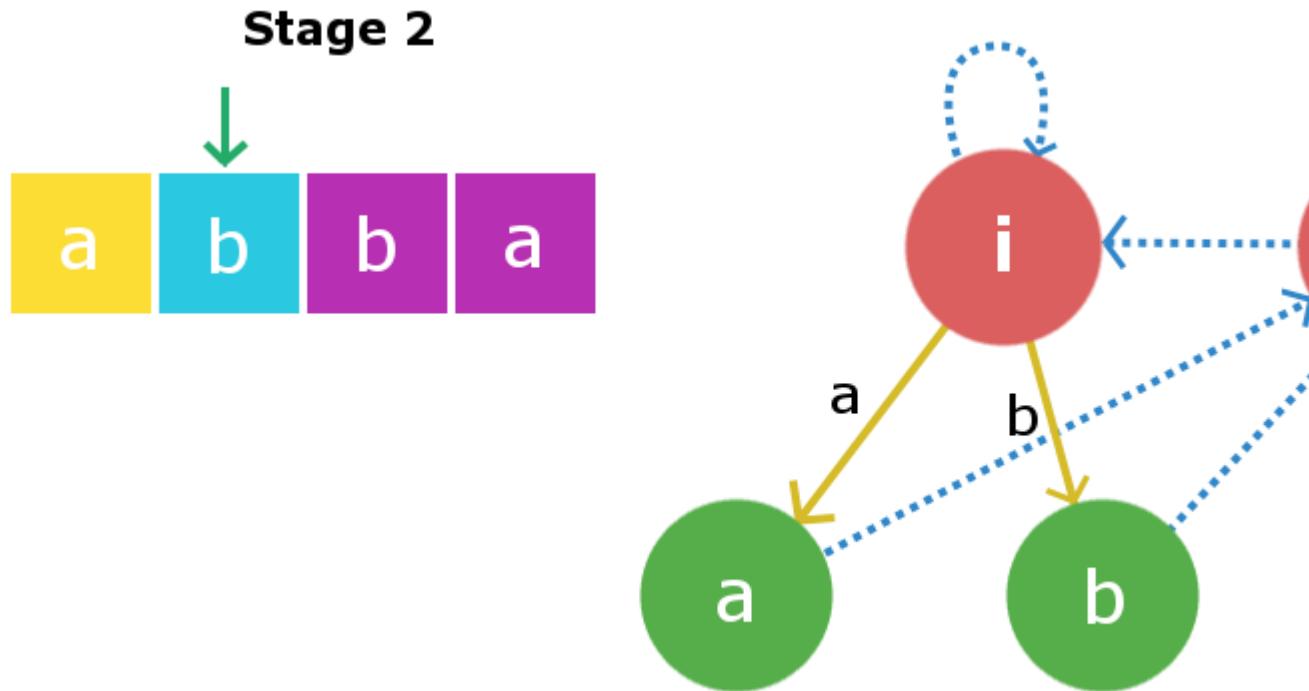
At initial state we will have our two dummy root nodes one with length -1 (some imaginary string **i**) and second a **null** string with length 0. At this point we haven't inserted any character in the tree. Root1 i.e root node with length -1 will be the current node from which insertion takes place.



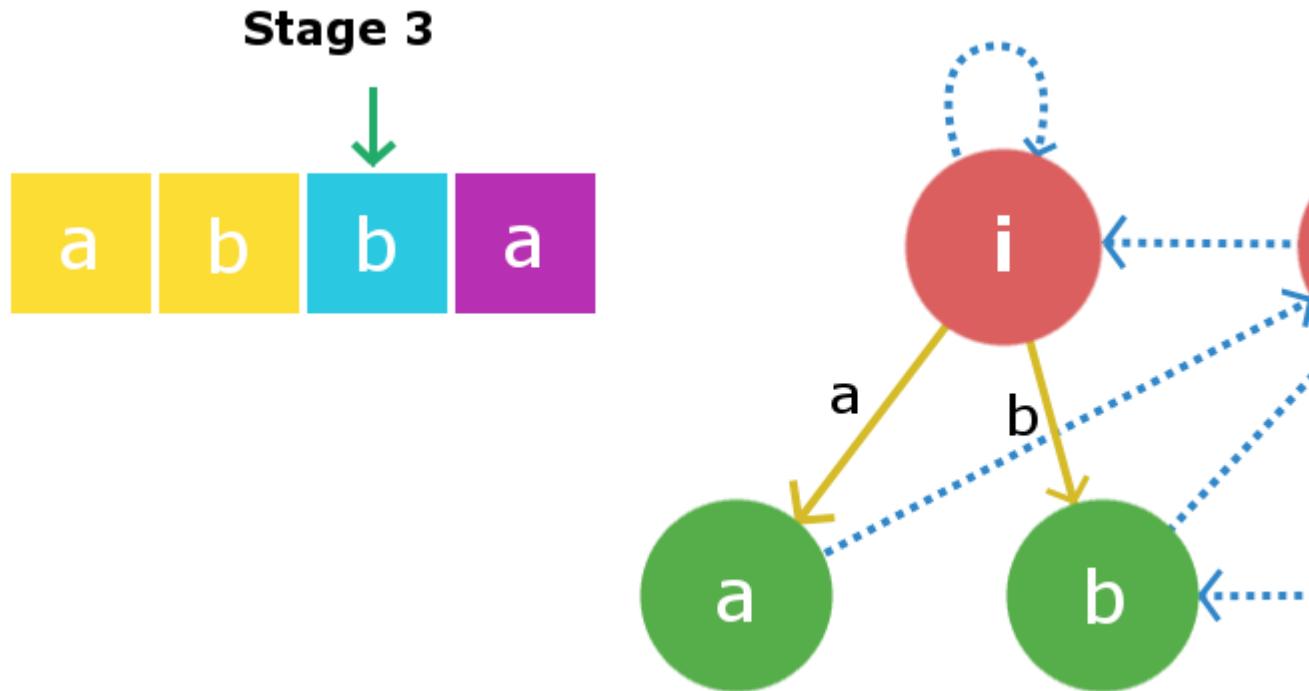
Stage 1: We will insert $s[0]$ i.e 'a'. We will start checking from the current node i.e Root1. Inserting 'a' at start and end of a string with length -1 will yield to a string with length 1 and this string will be "a". Therefore, we create a new node "a" and direct insertion edge from root1 to this new node. Now, largest palindromic suffix string for string of length 1 will be a null string so its suffix link will be directed to root2 i.e null string. Now the current node will be this new node "a".



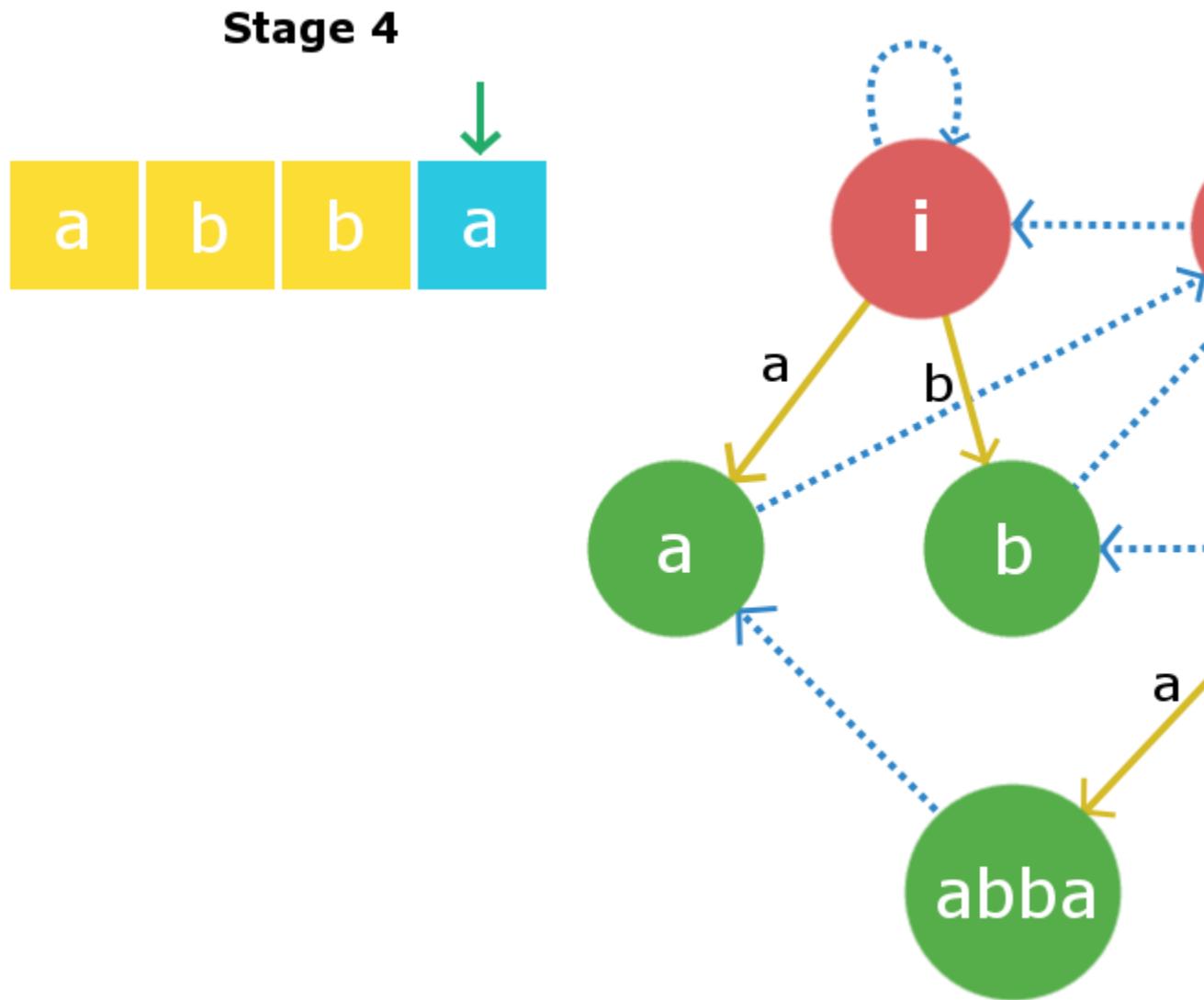
Stage 2: We will insert $s[1]$ i.e ‘b’. Insertion process will start from current node i.e “a” node. We will traverse the suffix link chain starting from current node till we find suitable X string , So here traversing the suffix link we again found root1 as X string. Once again inserting ‘b’ to string of length -1 will yield a string of of length 1 i.e string “b”. Suffix link for this node will go to null string as described in above insertion. Now the current node will be this new node “b”.



Stage 3: We will insert $s[2]$ i.e ‘b’. Once again starting from current node we will traverse its suffix link to find required X string. In this case it founds to be root2 i.e null string as adding ‘b’ at front and end of null string yields a palindrome “bb” of length 2. Therefore, we will create a new node “bb” and will direct the insertion edge from the null string to the newly created string. Now, the largest suffix palindrome for this current node will be node “b”. So, we will link the suffix edge from this newly created node to node “b”. Current node now becomes node “bb”.



Stage 4: We will insert $s[3]$ i.e 'a'. Insertion process begins with current node and in this case the current node itself is the largest X string such that $s[0] + X + s[3]$ is palindrome. Therefore, we will create a new node "abba" and link the insertion edge from the current node "bb" to this newly created node with edge weight 'a'. Now, the suffix the link from this newly created node will be linked to node "a" as that is the largest palindromic suffix.



The C++ implementation for the above implementation is given below :

```
// C++ program to demonstrate working of
// palindromic tree
#include <bits/stdc++.h>
using namespace std;

#define MAXN 1000

struct Node
```

```
{
    // store start and end indexes of current
    // Node inclusively
    int start, end;

    // stores length of substring
    int length;

    // stores insertion Node for all characters a-z
    int insertEdg[26];

    // stores the Maximum Palindromic Suffix Node for
    // the current Node
    int suffixEdg;
};

// two special dummy Nodes as explained above
Node root1, root2;

// stores Node information for constant time access
Node tree[MAXN];

// Keeps track the current Node while insertion
int currNode;
string s;
int ptr;

void insert(int idx)
{
    //STEP 1//

    /* search for Node X such that s[idx] X S[idx]
       is maximum palindrome ending at position idx
       iterate down the suffix link of currNode to
       find X */
    int tmp = currNode;
    while (true)
    {
        int curLength = tree[tmp].length;
        if (idx - curLength >= 1 and s[idx] == s[idx-curLength-1])
            break;
        tmp = tree[tmp].suffixEdg;
    }

    /* Now we have found X ....
     * X = string at Node tmp
     * Check : if s[idx] X s[idx] already exists or not*/
    if(tree[tmp].insertEdg[s[idx]-'a'] != 0)
}
}
```

```

{
    // s[idx] X s[idx] already exists in the tree
    currNode = tree[tmp].insertEdg[s[idx]-'a'];
    return;
}

// creating new Node
ptr++;

// making new Node as child of X with
// weight as s[idx]
tree[tmp].insertEdg[s[idx]-'a'] = ptr;

// calculating length of new Node
tree[ptr].length = tree[tmp].length + 2;

// updating end point for new Node
tree[ptr].end = idx;

// updating the start for new Node
tree[ptr].start = idx - tree[ptr].length + 1;

//STEP 2//

/* Setting the suffix edge for the newly created
Node tree[ptr]. Finding some String Y such that
s[idx] + Y + s[idx] is longest possible
palindromic suffix for newly created Node.*/

tmp = tree[tmp].suffixEdg;

// making new Node as current Node
currNode = ptr;
if (tree[currNode].length == 1)
{
    // if new palindrome's length is 1
    // making its suffix link to be null string
    tree[currNode].suffixEdg = 2;
    return;
}
while (true)
{
    int curLength = tree[tmp].length;
    if (idx-curLength >= 1 and s[idx] == s[idx-curLength-1])
        break;
    tmp = tree[tmp].suffixEdg;
}

```

```

// Now we have found string Y
// linking current Nodes suffix link with s[idx]+Y+s[idx]
tree[currNode].suffixEdg = tree[tmp].insertEdg[s[idx]-'a'];
}

// driver program
int main()
{
    // initializing the tree
    root1.length = -1;
    root1.suffixEdg = 1;
    root2.length = 0;
    root2.suffixEdg = 1;

    tree[1] = root1;
    tree[2] = root2;
    ptr = 2;
    currNode = 1;

    // given string
    s = "abcbab";
    int l = s.length();

    for (int i=0; i<l; i++)
        insert(i);

    // printing all of its distinct palindromic
    // substring
    cout << "All distinct palindromic substring for "
        << s << " : \n";
    for (int i=3; i<=ptr; i++)
    {
        cout << i-2 << " ) ";
        for (int j=tree[i].start; j<=tree[i].end; j++)
            cout << s[j];
        cout << endl;
    }

    return 0;
}

```

Output:

```

All distinct palindromic substring for abcbab :
1)a
2)b

```

- 3)c
- 4)bcb
- 5)abcba
- 6)bab

Time Complexity

The time complexity for the building process will be **O(k*n)**, here “n” is the length of the string and ‘k’ is the extra iterations required to find the string X and string Y in the suffix links every time we insert a character. Let’s try to approximate the constant ‘k’. We shall consider a worst case like $s = \text{"aaaaaaaaabcccccccccdeeeeeeeeef"}$. In this case for similar streak of continuous characters it will take extra 2 iterations per index to find both string X and Y in the suffix links , but as soon as it reaches some index i such that $s[i] \neq s[i-1]$ the left most pointer for the maximum length suffix will reach its rightmost limit. Therefore, for all i when $s[i] \neq s[i-1]$, it will cost in total n iterations(summing over each iteration) and for rest i when $s[i] == s[i-1]$ it takes 2 iteration which sums up over all such i and takes $2*n$ iterations. Hence, approximately our complexity in this case will be $O(3*n) \sim O(n)$. So, we can roughly say that the constant factor ‘k’ will be very less. Therefore, we can consider the overall complexity to be linear **O(length of string)**. You may refer the reference links for better understanding.

References :

- <http://codeforces.com/blog/entry/13959>
- <http://adilet.org/blog/25-09-14/>

Source

<https://www.geeksforgeeks.org/palindromic-tree-introduction-implementation/>

Chapter 123

Passing the Assignment

Passing the Assignment - GeeksforGeeks

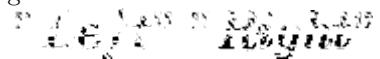
Rachel is submitting assignment from every student. She has given them a few minutes to arrange their assignments before submission. Just then Rahul remembers that his assignment is with Rohan. Rohan has to pass the assignment to Rahul. All the students are sitting in a straight line. He cannot pass the assignment in front of the teacher because then she will assume they have copied the assignment and will punish both of them. We have to help Rohan to find a way to pass the assignment to Rahul in such a way that he is not caught.

Assignment can be passed under some restrictions as given below:

- A student i can pass the assignment only to his immediate neighbor i.e $i-1$ and $i+1$.
- At any given time a student may pass the assignment to other student, accept the assignment from some other student or may keep the assignment to himself.
- Teacher is keeping watch over students from l_i to r_i including both l_i and r_i .
- If a student is being watched he cannot pass the note or accept the note, otherwise he is caught.
- If he keeps the assignment and he is being watched, he is not caught.

Given four inputs n, m, s, f where n is the total number of students, m is the number of steps during which teacher keeps vigil over students from l_i to r_i , s is the position of Rohan and f is the position of Rahul.

Each m query contains three inputs: The time at which she is watching, the leftmost child she is watching and the rightmost child she is watching.

We have to output a sequence of three words:  and  denoting the passing direction of current student.

Examples:

Input: n = 4, m = 4, s = 3, f = 4
1 2 4
2 1 2
3 3 4
4 2 3

Output: KeepRight

During time = 1, teacher is watching all the student from 2 to 4. Student 3 who has the assignment therefore cannot pass it to his neighbor, thus he keeps the assignment. During time = 2, teacher is watching student 1 and 2, therefore student 3 can pass the assignment to his right to student 4. Since student 4 is Rahul therefore our answer is "KeepRight"

Input: n = 10, m = 1, s = 1, f = 10
1 5 6

Output: RightRightRightRightRightRight
RightRightRightRight

During time = 1, teacher is watching student 5 and 6 therefore student 1 can easily pass the assignment to his right to student 2. After this teacher stops watching any student therefore they can keep on passing the assignment to their right until the assignment reaches Rahul. Therefore the answer is "RightRightRightRightRightRight
RightRightRightRight"

Approach:

At a given moment if the teacher is watching either the student currently holding the assignment or the student to whom the assignment is to be passed, then the student will keep it to himself else he will pass the assignment to his neighbor sitting towards Rahul.

Below is the implementation:

C++

```
// CPP program to find the way
#include <bits/stdc++.h>
using namespace std;

void solve(int n, int m, int s, int f,
           long t[], int l[], int r[])
```

```
{  
  
    int dir;  
    string val;  
  
    // If Rahul sits to right of Rohan  
    // then student will either pass  
    // it right or keep it to himself  
    if (s < f) {  
        dir = 1;  
        val = "Right";  
    }  
  
    // If Rahul sits to left of Rohan  
    // then student will either pass  
    // it left or keep it to himself  
    else {  
        dir = -1;  
        val = "Left";  
    }  
    string ans = "";  
  
    // current is keeping track of  
    // the position of assignment  
    // at a given time  
    int i = 0, current = s;  
  
    // tim variable keeping track of  
    // current time  
    long tim = 1;  
    while (1) {  
  
        // If time duration of query  
        // matches with the current time  
        if (i < m && tim == t[i]) {  
  
            // If teacher is watching the  
            // student having the assignment  
            // or the student to whom the  
            // assignment is to be passed  
            // then the student keeps it to  
            // itself  
            if ((current >= l[i] && current <= r[i]) ||  
                (current + dir >= l[i] && current + dir  
                 <= r[i])) {  
                ans += "Keep";  
                tim++;  
                i++;  
            }  
        }  
    }  
}
```

```
        continue;
    }
    i++;
}

// If the students are safe then student
// passes the assignment to his neighbor
current += dir;
ans += val;
tim++;

// If the assignment has reached Rahul
// then we break the loop
if (current == f)
    break;
}

cout << ans << endl;
}

// Driver function for the program
int main()
{
    int n = 4, m = 4, s = 3, f = 4;

    // matrix saving time for each query
    long t[m + 2] = { 1, 2, 3, 4 };

    // matrix saving leftmost child being
    // watched for each query
    int l[m + 2] = { 2, 1, 3, 2 };

    // matrix saving rightmost child
    // being watched for each query
    int r[m + 2] = { 4, 2, 4, 3 };

    solve(n, m, s, f, t, l, r);
    return 0;
}
```

Java

```
// Java program to find the way
import java.io.*;

class Hiding {
    static void solve(int n, int m, int s, int f,
                     long t[], int l[], int r[])
```

```
{  
    int dir;  
    String val;  
  
    // If Rahul sits to right of Rohan  
    // then student will either pass it  
    // right or keep it to himself  
    if (s < f) {  
        dir = 1;  
        val = "Right";  
    }  
  
    // If Rahul sits to left of Rohan then  
    // student will either pass it left  
    // or keep it to himself  
    else {  
        dir = -1;  
        val = "Left";  
    }  
  
    String ans = "";  
    int i = 0, current = s;  
  
    // Variable keeping track of current time  
    long tim = 1;  
    while (1 > 0) {  
  
        // If time duration of query  
        // matches with the current time  
        if (i < m && tim == t[i]) {  
  
            // If teacher is watching the student  
            // having the assignment or the  
            // student to whom the assignment is  
            // to be passed then the student  
            // keeps it to itself  
            if ((current >= l[i] && current <= r[i]) ||  
                (current + dir >= l[i] && current + dir  
                 <= r[i])) {  
                ans += "Keep";  
                tim++;  
                i++;  
                continue;  
            }  
            i++;  
        }  
  
        // If the students are safe then student
```

```
// passes the assignment to his neighbor
current += dir;
ans += val;
tim++;

// If the assignment has reached Rahul
// then we break the loop
if (current == f)
    break;
}
System.out.println(ans);
}

// Driver Program
public static void main(String args[])
{
    int n = 4, m = 4, s = 3, f = 4;

    // matrix saving time for each query
    long t[] = { 1, 2, 3, 4 };

    // matrix saving leftmost child
    // being watched
    int l[] = { 2, 1, 3, 2 };

    // matrix saving rightmost child
    // being watched
    int r[] = { 4, 2, 4, 3 };

    solve(n, m, s, f, t, l, r);
}
}
```

C#

```
// C# program to find the way
using System;

class Hiding
{
    static void solve(int n, int m, int s, int f,
                     long []t, int []l, int []r)
    {
        int dir;
        String val;

        // If Rahul sits to right of Rohan
        // then student will either pass it
```

```
// right or keep it to himself
if (s < f) {
    dir = 1;
    val = "Right";
}

// If Rahul sits to left of Rohan then
// student will either pass it left
// or keep it to himself
else {
    dir = -1;
    val = "Left";
}

String ans = "";
int i = 0, current = s;

// Variable keeping track of current time
long tim = 1;
while (1 > 0) {

    // If time duration of query
    // matches with the current time
    if (i < m && tim == t[i]) {

        // If teacher is watching the student
        // having the assignment or the
        // student to whom the assignment is
        // to be passed then the student
        // keeps it to itself
        if ((current >= l[i] && current <= r[i]) ||
            (current + dir >= l[i] && current +
             dir <= r[i]))
        {

            ans += "Keep";
            tim++;
            i++;
            continue;
        }
        i++;
    }

    // If the students are safe then student
    // passes the assignment to his neighbor
    current += dir;
    ans += val;
    tim++;
}
```

```
// If the assignment has reached Rahul
// then we break the loop
if (current == f)
    break;
}
Console.WriteLine(ans);
}

// Driver Program
public static void Main()
{
    int n = 4, m = 4, s = 3, f = 4;

    // matrix saving time for each query
    long []t = { 1, 2, 3, 4 };

    // matrix saving leftmost
    // child being watched
    int []l = { 2, 1, 3, 2 };

    // matrix saving rightmost
    // child being watched
    int []r = { 4, 2, 4, 3 };

    solve(n, m, s, f, t, l, r);
}
}

// This code is contributed by nitin mittal
```

Output:

KeepRight

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/passing-the-assignment/>

Chapter 124

Permutation of a string with maximum number of characters greater than its adjacent characters

Permutation of a string with maximum number of characters greater than its adjacent characters - GeeksforGeeks

Given a string str, the task is to print the maximum count of characters which are greater than both its left and right character in any permutation of the string.

Examples:

Input: str = “abc”

Output: 1

Permutations of the string with the count of maximal character in each string are:

abc – 0

acb – 1 Here a < c > b

bac – 0

bca – 1 Here b < c > a

cab – 0

cba – 0

Input: str = “geeks”

Output: 2

The string will be “egesk”

Observations:

- If the string' length is less than 3 then the answer will be 0 because no permutation is possible which satisfies the given condition.

- If the length of the given string is greater than or equal to 3 then assume that in the resulting string every other character is maximal character, that is there is exactly one character between any two consecutive maximal characters (otherwise we can remove all but the lowest one and add them to the end of the string).
- Assume for simplicity that this number is odd. Then, ideally, the string can have maximal characters in even positions i.e. at most $(n-1)/2$, where n is the length of the given string while the rest of the remaining characters in odd positions.
- First arrange all the characters in ascending order, place the first half characters at odd positions, and then fill the remaining even positions with the rest of the characters. In this way, all the characters in even positions will be those characters from which character at the left and the right position are smaller if there is no frequency of smaller character that is too high, start placing a character from an odd position, continue with the same character to even positions, and eventually reach a position next to the odd position from which we started to place the character. Here if the frequency of some smaller character in the string is too high then the count of maximal character will always be less than $(n-1)/2$.

Approach:

1. Calculate the frequency of each character in the given string.
2. Check the character which has the maximum frequency.
3. If the maximum frequency element is the smallest element in the given string then mark the flag as 0 otherwise mark the value of flag equal to 1.
4. The answer will be the minimum of $((n - 1) / 2, n - \text{max_freq} - \text{flag})$.

Below is the implementation of the above approach:

```
// C++ program to find maximum count
// of such characters which are greater
// its left and right character in
// any permutation of the string
#include <bits/stdc++.h>
using namespace std;

// function to find maximum maximal character in the string
int solve(int freq[])
{
    // to store sum of all frequency
    int n = 0;

    // to store maximum frequency
    int max_freq = 0;

    // frequency of the smallest element
    int first;

    // to check if the smallest
```

```
// element have amximum freqnqnc or not
int flag = 0;

// Iterate in the string and count frequency
for (int i = 0; i < 26; i++) {
    n += freq[i];

    // to store frequency of smallest element
    if (freq[i] != 0 && flag == 0) {
        first = freq[i];
        flag = 1;
    }

    // to store maximum frequency
    if (max_freq < freq[i])
        max_freq = freq[i];
}

// if sum of frequency of all element if 0
if (n == 0)
    return 0;

// if frequency of smallest character
// if largest frequency
if (first != max_freq)
    flag = 1;
else
    flag = 0;

return min((n - 1) / 2, n - max_freq - flag);
}

// Function that counts the frequency of
// each element
void solve(string s)
{
    // array to store the frequency of each character
    int freq[26];

    // initialize frequency of all character with 0
    memset(freq, 0, sizeof(freq));

    // loop to calculate freqnqnc of
    // each character in the given string
    for (int i = 0; i < s.length(); i++) {
        freq[s[i] - 'a']++;
    }
}
```

```
cout << solve(freq);
}

// Driver Code
int main()
{
    string s = "geeks";

    solve(s);
    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/permuatation-of-a-string-with-maximum-number-of-characters-greater-than-its-adjacent-characters/>

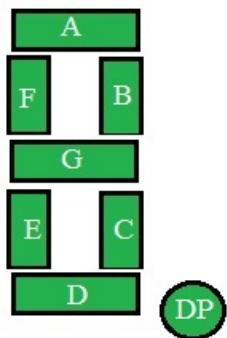
Chapter 125

Possible timings

Possible timings - GeeksforGeeks

Given a one/two digit timing, calculate the possibilities of occurrence of other timings(including the glowing one) with relevant to glowing segments, assuming some segments may not be glowing.

Displaying the numbers is done using seven segment display. It is guaranteed that the sticks currently displaying are working fine..



Examples:

Input : 78
Output :5

Input :05
Output :8

Explanation:

Example 1: 7 can be replaced by 5 different numbers 9, 3, 8, 0 and 7(if none of the

segment is broken) and **8** can be replaced by only **1** number i.e 8 itself(if none of the segment is broken), therefore the answer is **5*1=5**.

Example 2: **0** can be replaced by **2** numbers, 8 and 0, while **5** can be replaced by **4** different numbers. So, answer is **4*2=8**.

Approach :

Calculate for every number from **0–9** what all digits are possible by adding or removing exactly one rod from the display. Store this in an array and the answer will be product of the array value of both the digits of the input.

Below is the implementation of above approach:

C++

```
// CPP program to calculate possible
// number of timings
#include <bits/stdc++.h>
using namespace std;

// Array storing different numbers
// of digits a particular digit
// can be replaced with
int num[10] = { 2, 7, 2, 3, 3,
                 4, 2, 5, 1, 2 };

// Function performing calculations
void possibleTimings(string n)
{
    cout << num[n[0] - '0'] *
        num[n[1] - '0'] << endl;
}

// Driver function
int main()
{
    string n = "05";

    // Calling function
    possibleTimings(n);

    return 0;
}
```

Java

```
// Java program to calculate
// possible timings.
import java.io.*;
```

```
class Calci {  
  
    // Array storing different  
    // numbers of digits a particular  
    // digit can be replaced with  
    static int num[] = { 2, 7, 2, 3, 3,  
                        4, 2, 5, 1, 2 };  
  
    // Function performing calculations  
    public static void possibleTimings(String n)  
    {  
        System.out.println(num[(n.charAt(0) - '0')]  
                           * num[n.charAt(1) - '0']);  
    }  
  
    // Driver function  
    public static void main(String args[])  
    {  
        String n = "05";  
  
        // Calling function  
        possibleTimings(n);  
    }  
}
```

Python3

```
# python3 program to calculate possible  
# number of timings  
  
# Array storing different numbers  
# of digits a particular digit  
# can be replaced with  
num = [ 2,7,2,3,3,4,2,5,1,2 ]  
  
# Function performing calculations  
def possibleTimings(n):  
  
    print(num[int(n[0]) - int('0')] * num[int(n[1]) - int('0')])  
  
# Driver function  
n = "05"  
  
# Calling function  
possibleTimings(n)
```

```
# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
// C# program to calculate
// possible timings.
using System;

class Calci {

    // Array storing different
    // numbers of digits a particular
    // digit can be replaced with
    static int []num = { 2, 7, 2, 3, 3,
                        4, 2, 5, 1, 2 };

    // Function performing calculations
    public static void possibleTimings(string n)
    {
        Console.WriteLine(num[(n[0] - '0')]
                          * num[n[1] - '0']);
    }

    // Driver function
    public static void Main()
    {
        string n = "05";

        // Calling function
        possibleTimings(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to calculate
// possible number of timings

// Array storing different
// numbers of digits a
// particular digit can be
// replaced with
$num = array(2, 7, 2, 3, 3,
```

```
4, 2, 5, 1, 2);

// Function performing
// calculations
function possibleTimings($n)
{
    global $num;
    echo ($num[$n[0] - '0'] *
          $num[$n[1] - '0']). "\n";
}

// Driver Code
$n = "05";

// Calling function
possibleTimings($n);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

8

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/possible-timings/>

Chapter 126

Practice for cracking any coding interview

Practice for cracking any coding interview - GeeksforGeeks

Coding questions in this article are **difficulty wise ordered**. The idea of this post is to target two types of people.

1. **Competitive Programming Preparation (For Ist and IInd Year Students) :**
It is recommended to finish all questions from all categories except possibly Linked List, Tree and BST. However at least 10 questions from these categories should also be covered. If you have never done competitive programming before, it is strongly recommended to see [How to Begin with Competitive Programming](#) first. If you wish to get yourself prepared with a language first, you may first begin [C++ Track](#) or [Java Track](#)
2. **Interview preparation** It is recommended to cover all topics. In every every topic, you can start from questions according to your comfort level.

The [practice system](#) tells you exactly the test case where your code failed. In case you need more clarity about a question, you may use expected output button to see output for your given input. You can also view successful submissions of others in case you are stuck.

Topics :

- Mathematical
- Puzzles
- Arrays
- String
- Searching
- Sorting
- Hashing
- Matrix
- Recursion

- Divide & Conquer
- Linked List
- Doubly and Circular Linked Lists
- Stack

- Queue
- Prefix Sum and Sliding Window
- Bit Magic
- Tree
- Binary Search Tree
- Heap
- Graph
- Greedy Algorithms
- Dynamic Programming
- Backtracking
- Trie
- Misc Topics
- Important Links

Mathematical :

1. Print the pattern
2. Print table
3. Series AP
4. Series GP
5. Closest Number
6. Armstrong Numbers
7. Sum of digits of a number
8. Reverse digits
9. Print the Kth Digit
10. Binary number to decimal number
11. Jumping Numbers
12. GCD of two numbers
13. LCM of two numbers
14. Add two fractions
15. GCD of array
16. Factorial of a number
17. Compute nPr
18. Compute nCr
19. Largest prime factor
20. Perfect Numbers
21. Pair cube count
22. Find Nth root of M
23. Prime Number
24. Sieve of Eratosthenes
25. Sum of all prime numbers between 1 and N.
26. Pairs of prime numbers

Related Learning Resources : Mathematical Algorithms and Number Theory

Puzzles

1. Count Squares
2. 3 Divisors
3. Check if four points form a square
4. Check for power
5. Overlapping rectangles
6. Trailing zeroes in factorial
7. Angle between hour and minute hand
8. Number Of Open Doors
9. Triangular Numbers
10. Nth Even Fibonacci Number
11. Last two digit Fibonacci
12. Squares in a Matrix
13. Day of the week

Related Learning Resources : Puzzles

Arrays :

1. Array operations (Search, insert, delete)
2. Array alternate printing
3. Maximum and minimum in an array
4. Second largest in array
5. Sum of array elements
6. Reverse an Array
7. Rotate Array
8. Count of smaller elements
9. Remove duplicate elements from sorted Array
10. Count possible triangles
11. Leaders in an array
12. Minimum distance between two numbers
13. Sorted subsequence of size 3
14. Maximum Sub Array
15. Majority Element
16. Wave Array
17. Maximum Index
18. Max sum path in two arrays
19. Polynomial Addition
20. Product array puzzle
21. Find duplicates in a small ranged array
22. Find Missing And Repeating
23. Stock buy and sell

24. Trapping Rain Water
25. Pair with given sum in a sorted array
26. Chocolate Distribution Problem
27. Longest Consecutive subsequence
28. Three way partitioning

Related Learning Resources : Array Data Structure

String :

1. Check for palindrome
2. Check for anagram
3. Anagram Palindrome
4. Title case conversion
5. Sort the string
6. Merge two strings
7. Save Ironman
8. Good or Bad string
9. URLify a given string
10. Extract Maximum
11. Reverse words in a given string
12. Implement strstr
13. Check for subsequence
14. Check for rotation
15. Check if two strings are k-anagrams
16. Uncommon characters
17. Anagram Search
18. First repeating character
19. First non-repeating character
20. First non-repeating character in a stream
21. Longest Distinct characters in string
22. Longest Palindromic Substring
23. Find k-th character in string
24. Smallest window in a string containing all characters of another string
25. Add Binary Strings
26. Multiply two Strings
27. Nearest multiple of 10

Related Learning Resources : String Data Structure

Searching :

1. Linear Search
2. Facing the sun
3. Magnet Array Problem
4. Binary Search
5. Floor in a Sorted Array
6. Count occurrences in a sorted array

7. Search in a sorted and rotated
8. Find the missing number
9. Missing element of AP
10. Square root of a number
11. Find Transition Point in a Sorted Binary Array
12. Last index of One
13. Peak element
14. Allocate minimum number of pages
15. Common elements in three sorted
16. Smallest Positive missing number

Related Learning Resources : Searching Algorithms

Sorting :

1. Check if array is sorted
2. Sort a binary array
3. Sort an array of 0s, 1s and 2s
4. Bubble Sort
5. Insertion Sort
6. Selection Sort
7. Quick Sort
8. Merge Sort
9. Sort an array when two halves are sorted
10. Relative Sorting
11. Nuts and Bolts Problem
12. Triplet Sum in Array
13. Minimum Swaps to Sort
14. Sorting elements by frequency
15. Triplet Family
16. Count the triplets
17. Group Anagrams Together

Related Learning Resources : Sorting Algorithms

Hashing :

1. Count distinct elements
2. Array Subset of another array
3. Count frequencies of elements
4. Check if two arrays are equal or not
5. First element to occur k times
6. In First But Second
7. Non-Repeating Element
8. Winner of an election
9. Check for a pair with given sum
10. Count distinct pairs with difference k

11. Count pairs with given sum
12. Find all four sum numbers
13. A Simple Fraction
14. Largest Fibonacci Subsequence

Related Learning Resources : Hashing Data Structure

Matrix :

1. Transpose of Matrix
2. Print Matrix in snake Pattern
3. Print a given matrix in spiral form
4. Is Sudoku Valid
5. Count zeros in a sorted matrix
6. Squares in a Matrix
7. A Boolean Matrix Question
8. Search in row-wise and column-wise sorted
9. Find the row with maximum number of 1s
10. Count pairs Sum in matrices
11. Median In a Row-Wise sorted Matrix

Related Learning Resources : Matrix Data Structure

Recursion :

1. Print Pattern
2. Handshakes
3. Tower of Hanoi
4. Josephus problem
5. Recursively remove all adjacent duplicates
6. Possible words from Phone digits
7. Flood fill Algorithm
8. Permutations of a string

Related Learning Resources : Recursion

Divide & Conquer :

1. Write your own power function
2. Program for n-th Fibonacci Number
3. K-th element of two sorted Arrays
4. Median of two sorted arrays
5. Karatsuba Algorithm
6. The Painter's Partition Problem
7. Convex Hull

8. Counting inversions

Related Learning Resources : Divide and Conquer Algorithms

Linked List :

1. Print a Linked List
2. Length of a linked list
3. Node at a given index in linked list
4. Middle of a linked list
5. n-th node from end of a linked list
6. Delete a node
7. Remove every k'th node
8. Delete N nodes after M nodes of a linked list
9. Delete without head pointer
10. Rearrange a linked list
11. Segregate even and odd (Using only one traversal)
12. Reorder List
13. Insert in a Sorted List
14. Swap nodes in pairs
15. Reverse a linked list
16. Reverse a Linked List in groups of given size.
17. Check for palindrome
18. Flattening a linked list
19. Get intersection point
20. Remove duplicates from sorted list
21. Remove duplicates from unsorted lists
22. Sort a linked list of 0s, 1s and 2s.
23. Circular Linked List
24. Detect loop in a linked list
25. Find length of Loop
26. Remove loop in a linked list
27. Add two numbers represented by linked lists
28. Clone a linked list with random pointers
29. Add 1 to a number represented as linked list
30. Add two numbers represented as linked list
31. Multiply two linked lists
32. Merge two sorted linked lists
33. Merge Sort on Linked List
34. Intersection of Two Linked Lists
35. Union of Two Linked Lists

Related Learning Resources : Linked List Data Structure

Doubly and Circular Linked Lists

1. Insert a node in Doubly linked list
2. Delete node in Doubly Linked List

3. Circular Linked List Traversal
4. Split a Circular Linked List into two halves
5. Insert in Sorted way in a Sorted DLL
6. QuickSort on Doubly Linked List
7. Merge Sort on Doubly Linked List
8. Rotate doubly Linked List by P nodes
9. XOR Linked List

Related Learning Resources : [Doubly Linked List](#) and [Circular Linked List](#).

Stack

1. Implement Stack using Array
2. Implement Stack using Linked List
3. Check for balanced parenthesis
4. Reverse a stack
5. Implement two stacks in an array
6. Design a stack with getMin
7. The celebrity problem
8. Stock Span Problem
9. Next Greater Element
10. Next Smaller Element
11. Longest valid Parentheses

Related Learning Resources : [Stack Data Structure](#)

Queue and Dequeue

1. Implement Queue using Linked List
2. Implement Queue using Array
3. Implement Stack using Queue
4. Implement Queue using Stack
5. Reversing a Queue
6. Circular tour

Related Learning Resources : [Queue Data Structure](#)

Prefix Sum and Sliding Window

1. Equilibrium Point
2. Check if there is a subarray with 0 sum
3. Longest Sub-Array with Sum K
4. Longest subarray with sum divisible by K
5. Largest subarray with equal 1s and 0s
6. Longest common span with same number of 1s and 0s among two arrays

7. Find maximum sum in any subarray of size k
8. Count distinct elements in every window of size k
9. Check for subarray with given sum

Related Learning Resources : [Prefix Sum](#) and [Sliding Window](#)

Bit Magic

1. Check if a number is even or odd.
2. Number of bit flips
3. Game of XOR
4. Find bit at a position
5. Swap odd and even bits
6. Power of 2
7. Odd occurring element
8. Missing number in array
9. Index Of an Extra Element
10. Reverse Bits
11. Count set bits
12. Power Set

Related Learning Resources : [Bit Magic](#)

Tree

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Level order traversal
5. Find height of Binary Tree
6. Count Leaves in Binary Tree
7. Check for Children Sum Property
8. Mirror Tree
9. Check for Balanced Tree
10. Lowest Common Ancestor in a Binary Tree
11. Diameter of Binary Tree
12. Left View of Binary Tree
13. Right View of Binary Tree
14. Maximum path sum
15. Level order traversal line by line
16. Tree from Postorder and Inorder
17. Tree from Preorder and Inorder
18. Connect Nodes at Same Level
19. Zig-Zag level order traversal
20. Serialize and Deserialize a Binary Tree
21. Leaves to DLL
22. Binary Tree to Doubly Linked List
23. Binary Tree to Circular Doubly Linked List

Related Learning Resources : Tree Data Structure

Binary Search Tree

1. BST Search
2. BST Insert
3. BST Delete
4. Minimum in BST
5. Inorder Traversal and BST
6. Count BST nodes that lie in a given range
7. Add all greater values
8. Predecessor and Successor in BST
9. Closest Neighbor in BST
10. Lowest Common Ancestor in a BST
11. Convert Level Order Traversal to BST
12. Normal BST to Balanced BST
13. Pair with given sum in BST
14. Check for BST
15. Correct BST with two nodes swapped
16. Median of BST
17. k-th smallest element in BST
18. Unique BST's
19. Array to BST
20. Preorder Traversal and BST
21. Preorder to Postorder
22. Leaf nodes from preorder traversal
23. Triplet with 0 sum in BST
24. Merge two BST 's
25. Largest BST Subtree

Related Learning Resources : Binary Search Tree

Heap

1. Binary Heap Operations
2. Height of Heap
3. Heap Sort
4. Sort a Nearly Sorted Array
5. K Largest Elements
6. K-th largest element in a stream
7. Median of stream
8. Merge k sorted arrays

Related Learning Resources : Heap Data Structure

Graph

1. Print adjacency list
2. Breadth First Search
3. Depth First Search
4. Find whether path exist
5. Knight Walk
6. Snake and Ladder Problem
7. Bipartite Graph
8. Detect Cycle in an undirected graph
9. Detect Cycle in a directed graph
10. Find first n numbers with given set of digits
11. Rotten oranges
12. Topological sort
13. Shortest Source to Destination Path
14. Transitive closure of a Graph
15. Strongly Connected Components

Related Learning Resources : Graph Data Structure

Greedy Algorithms

1. Fractional Knapsack
2. Largest number with given sum
3. Activity Selection
4. N meetings in one room
5. Minimum Platforms
6. Minimum number of Coins
7. Job Sequencing Problem
8. Minimize the heights
9. Huffman Coding
10. Huffman Decoding
11. Minimum Spanning Tree
12. Dijkstra for Adjacency Matrix

Related Learning Resources : Greedy Algorithms

Dynamic Programming

1. Print first n Fibonacci Numbers.
2. Count ways to reach the n'th stair
3. Cutted Segments
4. Kadane's Algorithm
5. Stickler Thief
6. Minimum number of jumps
7. Total Decoding Messages
8. Min Cost Path
9. Coin Change
10. Longest Common Subsequence

11. Consecutive 1's not allowed
12. Edit Distance
13. Rod Cutting
14. Water Overflow
15. Maximum Tip Calculator
16. Longest Increasing Subsequence
17. Maximum sum increasing subsequence
18. Max length chain
19. 0 – 1 Knapsack Problem
20. Maximum Tip Calculator
21. Interleaved string
22. Longest Palindromic Subsequence
23. Wildcard Pattern Matching
24. Box Stacking
25. Longest Bitonic subsequence
26. Minimum sum partition
27. Largest square formed in a matrix
28. Word Break
29. Matrix Chain Multiplication
30. Special Keyboard
31. Egg Dropping Puzzle
32. Optimal Strategy for a Game

Related Learning Resources : Dynamic Programming

Backtracking

1. Rat Maze With Multiple Jumps
2. Coins and Game
3. Hamiltonian Path
4. Solve the Sudoku
5. Combination Sum – Part 2
6. Combination Sum
7. Subsets
8. Largest number in K swaps
9. M-Coloring Problem
10. Black and White

Related Learning Resources : Backtracking

Trie

1. Trie Search and Insert
2. Trie Delete
3. Unique rows in a binary matrix
4. Count of distinct substrings
5. Word Boggle

Related Learning Resources : [Trie Data Structure](#)

Misc Questions to test your overall learning

1. Longest common prefix
2. Implement Atoi
3. Two numbers with sum closest to zero
4. Smallest greater elements in whole array
5. Max rectangle
6. Find triplets with zero sum
7. Counting elements in two arrays
8. Merge K sorted linked lists
9. Maximum Difference
10. Circle of strings
11. All possible Word Breaks
12. Alien Dictionary
13. Design a tiny URL or URL shortener
14. Implement LRU Cache

Important Links :

1. [Sudo Placement](#) : For companies like Amazon, Microsoft, Adobe, .., etc
2. [Sudo Placement 2](#) : For companies like TCS, Infosys, Wipro, Cognizant, .. etc
3. Aptitude questions asked in round 1 : [Placements Course](#) designed for this purpose.
4. MCQs asked from different computer science subjects : [Subject-Wise Quizzes](#)
5. Interview theory and coding questions of all companies : [Company wise all practice questions](#).
6. Interview experiences of all companies : [Interview corner](#).
7. [Must Do Coding Questions for Companies](#) like Amazon, Microsoft, Adobe, ...
8. [Must Do Coding Questions Company-wise](#)

Source

<https://www.geeksforgeeks.org/practice-for-cracking-any-coding-interview/>

Chapter 127

Prefix Sum Array – Implementation and Applications in Competitive Programming

Prefix Sum Array - Implementation and Applications in Competitive Programming - Geeks-forGeeks

Given an array arr[] of size n, its prefix sum array is another array prefixSum[] of same size such that the value of prefixSum[i] is arr[0] + arr[1] + arr[2] ... arr[i].

Examples :

```
Input  : arr[] = {10, 20, 10, 5, 15}
Output : prefixSum[] = {10, 30, 40, 45, 60}
```

Explanation : While traversing the array, update the element by adding it with its previous element.
prefixSum[0] = 10,
prefixSum[1] = prefixSum[0] + arr[1] = 30,
prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.

To fill prefix sum array, we run through index 1 to last and keep on adding present element with previous value in prefix sum array.

Below is the implementation :

C++

```
// C++ program for Implementing
// prefix sum array
#include <bits/stdc++.h>
using namespace std;

// Fills prefix sum array
void fillPrefixSum(int arr[], int n, int prefixSum[])
{
    prefixSum[0] = arr[0];

    // Adding present element
    // with previous element
    for (int i = 1; i < n; i++)
        prefixSum[i] = prefixSum[i-1] + arr[i];
}

// Driver Code
int main()
{
    int arr[] = { 10, 4, 16, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int prefixSum[n];

    fillPrefixSum(arr, n, prefixSum);
    for (int i = 0; i < n; i++)
        cout << prefixSum[i] << " ";
}
```

Java

```
// Java Program for Implementing
// prefix sum array
class Prefix
{
    // Fills prefix sum array
    static void fillPrefixSum(int arr[], int n,
                             int prefixSum[])
    {
        prefixSum[0] = arr[0];

        // Adding present element
        // with previous element
        for( int i = 1; i < n; ++i )
            prefixSum[i] = prefixSum[i-1] + arr[i];
    }

    // Driver code
    public static void main(String[] args)
```

```
{  
    int arr[] = { 10, 4, 16, 20 };  
    int n = arr.length;  
    int prefixSum[] = new int[n];  
  
    fillPrefixSum(arr, n, prefixSum);  
  
    for (int i = 0; i < n; i++)  
        System.out.print(prefixSum[i] + " ");  
    System.out.println("");  
}  
  
}
```

// This Code is Contributed by Saket Kumar

Python3

```
# Python Program for Implementing  
# prefix sum array  
  
# Fills prefix sum array  
def fillPrefixSum(arr, n, prefixSum):  
  
    prefixSum[0] = arr[0]  
  
    # Adding present element  
    # with previous element  
    for i in range(1, n):  
        prefixSum[i] = prefixSum[i - 1] + arr[i]  
  
# Driver code  
arr =[10, 4, 16, 20 ]  
n = len(arr)  
  
prefixSum = [0 for i in range(n + 1)]  
  
fillPrefixSum(arr, n, prefixSum)  
  
for i in range(n):  
    print(prefixSum[i] , " ", end="")  
  
# This code is contributed  
# by Anant Agarwal.
```

C#

```
// C# Program for Implementing
```

```
// prefix sum array
class GFG
{
    // Fills prefix sum array
    static void fillPrefixSum(int []arr, int n,
                             int []prefixSum)
    {
        prefixSum[0] = arr[0];

        // Adding present element
        // with previous element
        for( int i = 1; i < n; ++i )
            prefixSum[i] = prefixSum[i - 1] + arr[i];
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 10, 4, 16, 20 };
        int n = arr.Length;
        int []prefixSum = new int[n];

        fillPrefixSum(arr, n, prefixSum);

        for (int i = 0; i < n; i++)
            Console.Write(prefixSum[i] + " ");
        Console.Write("");
    }
}

// This Code is Contributed by nitin mittal
```

PHP

```
<?php
// PHP program for
// Implementing prefix
// sum array

// Fills prefix sum array
function fillPrefixSum($arr,
                      $n)
{
    $prefixSum = array();
    $prefixSum[0] = $arr[0];
```

```
// Adding present element
// with previous element
for ($i = 1; $i < $n; $i++)
    $prefixSum[$i] = $prefixSum[$i - 1] +
        $arr[$i];

for ($i = 0; $i < $n; $i++)
    echo $prefixSum[$i] . " ";
}

// Driver Code
$arr = array(10, 4, 16, 20);
$n = count($arr);

fillPrefixSum($arr, $n);

// This code is contributed
// by Sam007
?>
```

Output:

10 14 30 50

Applications :

- **Equilibrium index of an array** : Equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes.
- **Find if there is a subarray with 0 sum** : Given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.
- **Maximum subarray size, such that all subarrays of that size have sum less than k** : Given an array of n positive integers and a positive integer k, the task is to find the maximum subarray size such that all subarrays of that size have sum of elements less than k.
- **Find the prime numbers which can written as sum of most consecutive primes** : Given an array of limits. For every limit, find the prime number which can be written as the sum of the most consecutive primes smaller than or equal to limit.
- **Longest Span with same Sum in two Binary arrays** : Given two binary arrays arr1[] and arr2[] of same size n. Find length of the longest common span (i, j) where j >= i such that arr1[i] + arr1[i+1] + + arr1[j] = arr2[i] + arr2[i+1] + + arr2[j].
- **Maximum subarray sum modulo m** : Given an array of n elements and an integer m. The task is to find the maximum value of the sum of its subarray modulo m i.e find the sum of each subarray mod m and print the maximum value of this modulo operation.
- **Maximum subarray size, such that all subarrays of that size have sum less than k** : Given an array of n positive integers and a positive integer k, the task is to find the

maximum subarray size such that all subarrays of that size have sum of elements less than k.

- **Maximum occurred integer in n ranges :** Given n ranges of the form L and R, the task is to find the maximum occurred integer in all the ranges. If more than one such integer exists, print the smallest one.
- **Minimum cost for acquiring all coins with k extra coins allowed with every coin :** You are given a list of N coins of different denominations. you can pay an amount equivalent to any 1 coin and can acquire that coin. In addition, once you have paid for a coin, we can choose at most K more coins and can acquire those for free. The task is to find the minimum amount required to acquire all the N coins for a given value of K.
- **Random number generator in arbitrary probability distribution fashion :** Given n numbers, each with some frequency of occurrence. Return a random number with probability proportional to its frequency of occurrence.

An Example Problem :

Consider an array of size n with all initial values as 0. Perform given ‘m’ add operations from index ‘a’ to ‘b’ and evaluate highest element in array. An add operation adds 100 to all elements from a to b (both inclusive).

Example :

```
Input : n = 5 // We consider array {0, 0, 0, 0, 0}
       m = 3.
       a = 2, b = 4.
       a = 1, b = 3.
       a = 1, b = 2.
Output : 300
```

Explanation :

```
After I operation -
A : 0 100 100 100 0
```

```
After II operation -
A : 100 200 200 100 0
```

```
After III operation -
A : 200 300 200 100 0
```

Highest element : 300

A simple approach is running a loop ‘m’ times. Inputting a and b and running a loop from a to b, adding all elements by 100.

Efficient approach using **Prefix Sum Array** :

- 1 : Run a loop for 'm' times, inputting 'a' and 'b'.
- 2 : Add 100 at index 'a' and subtract 100 from index 'b+1'.
- 3 : After completion of 'm' operations, compute the prefix sum array.
- 4 : Scan the largest element and we're done.

What we did was adding 100 at 'a' because this will add 100 to all elements while taking prefix sum array. Subtracting 100 from 'b+1' will reverse the changes made by adding 100 to elements from 'b' onward.

For better understanding :

After I operation -

A : 0 100 0 0 -100

Prefix Sum Array : 0 100 100 100 0

After II operation -

A : 100 100 0 -100 -100

Prefix Sum Array : 100 200 200 100 0

After III operation -

A : 200 100 -100 -100 -100

Prefix Sum Array : 200 300 200 100 0

Final Prefix Sum Array : 200 300 200 100 0

The required highest element : 300

Recent Articles on Prefix Sum Technique

Improved By : nitin mittal, Sam007

Source

<https://www.geeksforgeeks.org/prefix-sum-array-implementation-applications-competitive-programming/>

Chapter 128

Print all subsequences of a string | Iterative Method

Print all subsequences of a string | Iterative Method - GeeksforGeeks

Given a string s, print all possible subsequences of the given string in an iterative manner.
We have already discussed [Recursive method to print all subsequences of a string](#).
Examples:

Input : abc
Output : a, b, c, ab, ac, bc, abc

Input : aab
Output : a, b, aa, ab, aab

Approah 1 :

Here, we discuss much easier and simpler iterative approach which is similar to [Power Set](#).
We use bit pattern from [binary representation](#) of 1 to $2^{\text{length}(s)} - 1$.

input = "abc"

Binary representation to consider 1 to (2^3-1) , i.e 1 to 7.

Start from left (MSB) to right (LSB) of binary representation and append characters from input string which corresponds to bit value 1 in binary representation to Final subsequence string sub.

Example:

001 => abc . Only c corresponds to bit 1. So, subsequence = c.

101 => abc . a and c corresponds to bit 1. So, subsequence = ac.

```
binary_representation (1) = 001 => c
binary_representation (2) = 010 => b
binary_representation (3) = 011 => bc
binary_representation (4) = 100 => a
```

```
binary_representation (5) = 101 => ac
binary_representation (6) = 110 => ab
binary_representation (7) = 111 => abc
```

Below is the implementation of above approach:

```
// CPP program to print all Subsequences
// of a string in iterative manner
#include <bits/stdc++.h>
using namespace std;

// function to find subsequence
string subsequence(string s, int binary, int len)
{
    string sub = "";
    for (int j = 0; j < len; j++)

        // check if jth bit in binary is 1
        if (binary & (1 << j))

            // if jth bit is 1, include it
            // in subsequence
            sub += s[j];

    return sub;
}

// function to print all subsequences
void possibleSubsequences(string s){

    // map to store subsequence
    // lexicographically by length
    map<int, set<string> > sorted_subsequence;

    int len = s.size();

    // Total number of non-empty subsequence
    // in string is 2^len-1
    int limit = pow(2, len);

    // i=0, corresponds to empty subsequence
    for (int i = 1; i <= limit - 1; i++) {

        // subsequence for binary pattern i
        string sub = subsequence(s, i, len);

        // storing sub in map
        sorted_subsequence[sub.length()].insert(sub);
    }
}
```

```

for (auto it : sorted_subsequence) {

    // it.first is length of Subsequence
    // it.second is set<string>
    cout << "Subsequences of length = "
        << it.first << " are:" << endl;

    for (auto ii : it.second)

        // ii is iterator of type set<string>
        cout << ii << " ";

    cout << endl;
}
}

// driver function
int main()
{
    string s = "aabc";
    possibleSubsequences(s);
    return 0;
}

```

Output:

```

Subsequences of length = 1 are:
a b c
Subsequences of length = 2 are:
aa ab ac bc
Subsequences of length = 3 are:
aab aac abc
Subsequences of length = 4 are:
aabc

```

Time Complexity : $O(2^n \cdot l)$, where n is length of string to find subsequences and l is length of binary string.

Approach 2 :

Approach is to get the position of rightmost set bit and reset that bit after appending corresponding character from given string to the subsequence and will repeat the same thing till corresponding binary pattern has no set bits.

If input is s = “abc”

Binary representation to consider 1 to (2^3-1) , i.e 1 to 7.

001 => abc . Only c corresponds to bit 1. So, subsequence = c
101 => abc . a and c corresponds to bit 1. So, subsequence = ac.

Let us use Binary representation of 5, i.e 101.

Rightmost bit is at position 1, append character at beginning of sub = c ,reset position 1
=> 100

Rightmost bit is at position 3, append character at beginning of sub = ac ,reset position 3
=> 000

As now we have no set bit left, we stop computing subsequence.

Example :

```
binary_representation (1) = 001 => c
binary_representation (2) = 010 => b
binary_representation (3) = 011 => bc
binary_representation (4) = 100 => a
binary_representation (5) = 101 => ac
binary_representation (6) = 110 => ab
binary_representation (7) = 111 => abc
```

Below is the implementation of above approach :

C++

```
// CPP code all Subsequences of a
// string in iterative manner
#include <bits/stdc++.h>
using namespace std;

// function to find subsequence
string subsequence(string s, int binary)
{
    string sub = "";
    int pos;

    // loop while binary is greater than 0
    while(binary>0)
    {
        // get the position of rightmost set bit
        pos=log2(binary&-binary)+1;

        // append at beginning as we are
        // going from LSB to MSB
        sub=s[pos-1]+sub;

        // resets bit at pos in binary
        binary= (binary & ~(1 << (pos-1)));
    }

    return sub;
}
```

```
}

// function to print all subsequences
void possibleSubsequences(string s){

    // map to store subsequence
    // lexicographically by length
    map<int, set<string> > sorted_subsequence;

    int len = s.size();

    // Total number of non-empty subsequence
    // in string is 2^len-1
    int limit = pow(2, len);

    // i=0, corresponds to empty subsequence
    for (int i = 1; i <= limit - 1; i++) {

        // subsequence for binary pattern i
        string sub = subsequence(s, i);

        // storing sub in map
        sorted_subsequence[sub.length()].insert(sub);
    }

    for (auto it : sorted_subsequence) {

        // it.first is length of Subsequence
        // it.second is set<string>
        cout << "Subsequences of length = "
            << it.first << " are:" << endl;

        for (auto ii : it.second)

            // ii is iterator of type set<string>
            cout << ii << " ";

        cout << endl;
    }
}

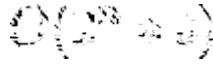
// driver function
int main()
{
    string s = "aabc";
    possibleSubsequences(s);

    return 0;
}
```

}

Output:

```
Subsequences of length = 1 are:  
a b c  
Subsequences of length = 2 are:  
aa ab ac bc  
Subsequences of length = 3 are:  
aab aac abc  
Subsequences of length = 4 are:  
aabc
```

Time Complexity:  $O(2^n \cdot b)$, where n is the length of string to find subsequence and b is the number of set bits in binary string.

Source

<https://www.geeksforgeeks.org/print-subsequences-string-iterative-method/>

Chapter 129

Program to find last two digits of 2^n

Program to find last two digits of 2^n - GeeksforGeeks

Given a number n, we need to find the last two digits of 2^n .

Examples:

Input : n = 7
Output : 28

Input : n = 72
Output : 96
 $2^{72} = 4722366482869645213696$

A **Naive Approach** is to find the value of 2^n iteratively or using [pow](#) function. Once the value of 2^n is calculated, find the last two digits and print it.

Note: This approach will only work for 2^n within certain range, as [overflow](#) occurs.

Below is the implementation of the above approach.

C++

```
// C++ code to find last 2 digits of 2^n
#include <bits/stdc++.h>
using namespace std;

// Find the first digit
int LastTwoDigit(long long int num)
{
```

```
// Get the last digit from the number
int one = num % 10;

// Remove last digit from number
num /= 10;

// Get the last digit from
// the number(last second of num)
int tens = num % 10;

// Take last digit to ten's position
// i.e. last second digit
tens *= 10;

// Add the value of ones and tens to
// make it complete 2 digit number
num = tens + one;

// return the first digit
return num;
}

// Driver program
int main()
{
    int n = 10;
    long long int num = 1;

    // pow function used
    num = pow(2, n);

    cout << "Last " << 2;

    cout << " digits of " << 2;

    cout << "^" << n << " = ";

    cout << LastTwoDigit(num) << endl;
    return 0;
}
```

Java

```
// Java code to find last 2 digits of  $2^n$ 

class Geeks {

    // Find the first digit
```

```
static long LastTwoDigit(long num)
{
    // Get the last digit from the number
    long one = num % 10;

    // Remove last digit from number
    num /= 10;

    // Get the last digit from
    // the number(last second of num)
    long tens = num % 10;

    // Take last digit to ten's position
    // i.e. last second digit
    tens *= 10;

    // Add the value of ones and tens to
    // make it complete 2 digit number
    num = tens + one;

    // return the first digit
    return num;
}

// Driver code
public static void main(String args[])
{
    int n = 10;
    long num = 1;

    // pow function used
    num = (long)Math.pow(2, n);

    System.out.println("Last 2 digits of  $2^{10} =$ 
                      +LastTwoDigit(num));

}
}

// This code is contributed by ankita_saini
```

Python 3

```
# Python 3 code to find
# last 2 digits of  $2^n$ 

# Find the first digit
```

```
def LastTwoDigit(num):

    # Get the last digit from the number
    one = num % 10

    # Remove last digit from number
    num //= 10

    # Get the last digit from
    # the number(last second of num)
    tens = num % 10

    # Take last digit to ten's position
    # i.e. last second digit
    tens *= 10

    # Add the value of ones and tens to
    # make it complete 2 digit number
    num = tens + one

    # return the first digit
    return num

# Driver Code
if __name__ == "__main__":
    n = 10
    num = 1

    # pow function used
    num = pow(2, n);

    print("Last " + str(2) + " digits of " +
          str(2) + "^" + str(n) +
          " = ", end = "")

    print(LastTwoDigit(num))

# This code is contributed
# by ChitraNayal

C#

// C# code to find last
// 2 digits of  $2^n$ 
using System;

class GFG
{
```

```
// Find the first digit
static long LastTwoDigit(long num)
{

    // Get the last digit
    // from the number
    long one = num % 10;

    // Remove last digit
    // from number
    num /= 10;

    // Get the last digit
    // from the number(last
    // second of num)
    long tens = num % 10;

    // Take last digit to
    // ten's position i.e.
    // last second digit
    tens *= 10;

    // Add the value of ones
    // and tens to make it
    // complete 2 digit number
    num = tens + one;

    // return the first digit
    return num;
}

// Driver code
public static void Main(String []args)
{
    int n = 10;
    long num = 1;

    // pow function used
    num = (long)Math.Pow(2, n);

    Console.WriteLine("Last 2 digits of  $2^{10} = " +$ 
                      LastTwoDigit(num));
}

// This code is contributed
// by Ankita_Saini
```

PHP

```
<?php
// PHP code to find last
// 2 digits of  $2^n$ 

// Find the first digit
function LastTwoDigit($num)
{
    // Get the last digit
    // from the number
    $one = $num % 10;

    // Remove last digit
    // from number
    $num /= 10;

    // Get the last digit
    // from the number(last
    // second of num)
    $tens = $num % 10;

    // Take last digit to
    // ten's position i.e.
    // last second digit
    $tens *= 10;

    // Add the value of ones
    // and tens to make it
    // complete 2 digit number
    $num = $tens + $one;

    // return the first digit
    return $num;
}

// Driver Code
$n = 10;
$num = 1;

// pow function used
$num = pow(2, $n);

echo ("Last " . 2);
echo (" digits of " . 2);
echo("^n " . $n . " = ");
```

```
echo( LastTwoDigit($num)) ;  
  
// This code is contributed  
// by Shivi_Aggarwal  
?>
```

Output:

```
Last 2 digits of  $2^{10} = 24$ 
```

Efficient approach: The efficient way is to keep only 2 digits after every multiplication. This idea is very similar to the one discussed in [Modular exponentiation](#) where a general way is discussed to find $(a^b) \% c$, here in this case c is 10^2 as the last two digits are only needed.

Below is the implementation of the above approach.

C++

```
// C++ code to find last 2 digits of  $2^n$   
#include <iostream>  
using namespace std;  
  
/* Iterative Function to calculate  $(x^y) \% p$  in  $O(\log y)$  */  
int power(long long int x, long long int y, long long int p)  
{  
    long long int res = 1; // Initialize result  
  
    x = x % p; // Update x if it is more than or  
    // equal to p  
  
    while (y > 0) {  
  
        // If y is odd, multiply x with result  
        if (y & 1)  
            res = (res * x) % p;  
  
        // y must be even now  
        y = y >> 1; // y = y/2  
        x = (x * x) % p;  
    }  
    return res;  
}  
  
// C++ function to calculate  
// number of digits in x
```

```
int numberOfDigits(int x)
{
    int i = 0;
    while (x) {
        x /= 10;
        i++;
    }
    return i;
}

// C++ function to print last 2 digits of  $2^n$ 
void LastTwoDigit(int n)
{
    cout << "Last " << 2;
    cout << " digits of " << 2;
    cout << "^" << n << " = ";

    // Generating  $10^2$ 
    int temp = 1;
    for (int i = 1; i <= 2; i++)
        temp *= 10;

    // Calling modular exponentiation
    temp = power(2, n, temp);

    // Printing leftmost zeros. Since  $(2^n) \% 2$ 
    // can have digits less than 2. In that
    // case we need to print zeros
    for (int i = 0; i < 2 - numberOfDigits(temp); i++)
        cout << 0;

    // If temp is not zero then print temp
    // If temp is zero then already printed
    if (temp)
        cout << temp;
}

// Driver program to test above functions
int main()
{
    int n = 72;
    LastTwoDigit(n);
    return 0;
}
```

Java

```
// Java code to find last
```

```
// 2 digits of  $2^n$ 
class GFG
{

    /* Iterative Function to
    calculate  $(x^y) \% p$  in  $O(\log y)$  */
    static int power(long x, long y,
                    long p)
    {
        int res = 1; // Initialize result

        x = x % p; // Update x if it is more
                    // than or equal to p

        while (y > 0)
        {

            // If y is odd, multiply
            // x with result
            long r = y & 1;

            if (r == 1)
                res = (res * (int)x) % (int)p;

            // y must be even now
            y = y >> 1; // y = y/2
            x = (x * x) % p;
        }
        return res;
    }

    // Java function to calculate
    // number of digits in x
    static int numberOfDigits(int x)
    {
        int i = 0;
        while (x != 0)
        {
            x /= 10;
            i++;
        }
        return i;
    }

    // Java function to print
    // last 2 digits of  $2^n$ 
    static void LastTwoDigit(int n)
    {
```

```
System.out.print("Last " + 2 +
                  " digits of " + 2 + "^");
System.out.print(n +" = ");

// Generating 10^2
int temp = 1;
for (int i = 1; i <= 2; i++)
    temp *= 10;

// Calling modular exponentiation
temp = power(2, n, temp);

// Printing leftmost zeros.
// Since  $(2^n) \% 2$  can have digits
// less than 2. In that case
// we need to print zeros
for (int i = 0;
     i < ( 2 - number0fDigits(temp)); i++)
    System.out.print(0 + " ");

// If temp is not zero then
// print temp. If temp is zero
// then already printed
if (temp != 0)
    System.out.println(temp);
}

// Driver Code
public static void main(String[] args)
{
    int n = 72;
    LastTwoDigit(n);
}
}

// This code is contributed
// by ChitraNayal
```

Python 3

```
# Python 3 code to find
# last 2 digits of  $2^n$ 

# Iterative Function to
# calculate  $(x^y) \% p$  in O(log y)
def power(x, y, p):

    res = 1 # Initialize result
```

```
x = x % p # Update x if it is more
           # than or equal to p

while (y > 0):

    # If y is odd, multiply
    # x with result
    if (y & 1):
        res = (res * x) % p

    # y must be even now
    y = y >> 1 # y = y/2
    x = (x * x) % p

return res

# function to calculate
# number of digits in x
def numberOfDigits(x):

    i = 0
    while (x):
        x // 10
        i += 1

    return i

# function to print
# last 2 digits of  $2^n$ 
def LastTwoDigit(n):

    print("Last " + str(2) +
          " digits of " + str(2), end = "")
    print("^\n" + str(n) + " = ", end = "")

    # Generating 10^2
    temp = 1
    for i in range(1, 3):
        temp *= 10

    # Calling modular exponentiation
    temp = power(2, n, temp)

    # Printing leftmost zeros.
    # Since  $(2^n)\%2$  can have digits
    # less then 2. In that case we
    # need to print zeros
```

```
for i in range(2 - number0fDigits(temp)):
    print(0, end = "")

# If temp is not zero then print temp
# If temp is zero then already printed
if temp:
    print(temp)

# Driver Code
if __name__ == "__main__":
    n = 72
    LastTwoDigit(n)

# This code is contributed
# by ChitraNayal
```

C#

```
// C# code to find last
// 2 digits of  $2^n$ 
using System;

class GFG
{

/* Iterative Function to calculate
    $(x^y) \% p$  in  $O(\log y)$  */
static int power(long x, long y,
                 long p)
{
    int res = 1; // Initialize result

    x = x % p; // Update x if it is more
                // than or equal to p

    while (y > 0)
    {

        // If y is odd, multiply
        // x with result
        long r = y & 1;

        if (r == 1)
            res = (res * (int)x) % (int)p;

        // y must be even now
        y = y >> 1; // y = y/2
        x = (x * x) % p;
    }
}
```

```
}  
return res;  
}  
  
// C# function to calculate  
// number of digits in x  
static int numberOfDigits(int x)  
{  
    int i = 0;  
    while (x != 0)  
    {  
        x /= 10;  
        i++;  
    }  
    return i;  
}  
  
// C# function to print  
// last 2 digits of  $2^n$   
static void LastTwoDigit(int n)  
{  
    Console.Write("Last " + 2 +  
                " digits of " + 2 + "^");  
    Console.Write(n + " = ");  
  
    // Generating  $10^2$   
    int temp = 1;  
    for (int i = 1; i <= 2; i++)  
        temp *= 10;  
  
    // Calling modular exponentiation  
    temp = power(2, n, temp);  
  
    // Printing leftmost zeros. Since  
    //  $(2^n) \% 2$  can have digits less  
    // than 2. In that case we need  
    // to print zeros  
    for (int i = 0;  
         i < (2 - numberOfDigits(temp)); i++)  
        Console.Write(0 + " ");  
  
    // If temp is not zero then print temp  
    // If temp is zero then already printed  
    if (temp != 0)  
        Console.Write(temp);  
}  
  
// Driver Code
```

```
public static void Main()
{
    int n = 72;
    LastTwoDigit(n);
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP code to find last
// 2 digits of  $2^n$ 

/* Iterative Function to
calculate  $(x^y) \% p$  in  $O(\log y)$  */
function power($x, $y, $p)
{
    $res = 1; // Initialize result

    $x = $x % $p; // Update x if it
                   // is more than or
                   // equal to p

    while ($y > 0)
    {

        // If y is odd, multiply
        // x with result
        if ($y & 1)
            $res = ($res * $x) % $p;

        // y must be even now
        $y = $y >> 1; // y = y/2
        $x = ($x * $x) % $p;
    }
    return $res;
}

// PHP function to calculate
// number of digits in x
function numberOfDigits($x)
{
    $i = 0;
    while ($x)
    {
```

```
$x /= 10;
$i++;
}
return $i;
}

// PHP function to print
// last 2 digits of  $2^n$ 
function LastTwoDigit($n)
{
    echo("Last " . 2);
    echo(" digits of " . 2);
    echo("^\n" . $n . " = ");

    // Generating  $10^2$ 
    $temp = 1;
    for ($i = 1; $i <= 2; $i++)
        $temp *= 10;

    // Calling modular
    // exponentiation
    $temp = power(2, $n, $temp);

    // Printing leftmost zeros.
    // Since  $(2^n) \% 2$  can have
    // digits less than 2. In
    // that case we need to
    // print zeros
    for ($i = 0;
        $i < 2 - number_of_digits($temp); $i++)
        echo (0);

    // If temp is not zero then
    // print temp. If temp is zero
    // then already printed
    if ($temp)
        echo ($temp);
}

// Driver Code
$n = 72;
LastTwoDigit($n);

// This code is contributed
// by Shivi_Agarwal
?>
```

Output:

Last 2 digits of $2^{72} = 96$

Time Complexity: $O(\log n)$

Improved By : [ankita_saini](#), [Shivi_Aggarwal](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/program-to-find-last-two-digits-of-2n/>

Chapter 130

Project Euler

Project Euler - GeeksforGeeks

What is Project Euler?

Project Euler is a series of challenging problems that require mathematical and programming skills. Somebody who enjoys learning new area of mathematics, project Euler is going to be a fun journey.

Where are the problems ?

The problems are right here in their official [archive](#).

Let's solve a problem from the archive and understand its complexity. Randomly I have chosen Problem no 116.

Problem 116 : Red, green or blue tiles

Problem Statement

A row of five black square tiles is chosen (length three), or blue (length four).

If red tiles are chosen there are exactly one way.



If green tiles are chosen there are exactly two ways.

And if blue tiles are chosen there are exactly one way.

Solution: [IT IS ADVISED TO TRY YOURSELF FIRST]

A red tile is of length 2, green is of length 3 and blue is of length 4.

Since, we need to count total ways for 50 units of black colored square tiles, say $k = 50$.

```
def E_116(i, k):
    ways = [1] * i + [0] * (k-i+1)
    for j in range(i, k+1):
        ways[j] += ways[j - 1] + ways[j - i]
    return ways[k] - 1
```

Here, we are initializing our function **E_116()** which holds the logic of the solution to the problem. The function **E_116()** has two parameters i = number of black coloured square tiles covered by the new coloured (red, green or blue) tiles and k = total number of black coloured square tiles.

In the function,

```
ways = [1] * i + [0] * (k-i+1)
```

In [17]: `ways = [1]*5`

In [18]: `ways`

Out[18]: `[1, 1, 1, 1, 1]`

In [19]: `type(ways)`

Out[19]: `list`

So, `ways` is a list which holds the total number of ways which the i length block can cover 50 black coloured square tiles. For e.g:

```
In [23]: def E_16(m, n):
    ...:     ways = [1] * m + [0]
    ...:     return ways
    ...:

In [24]: x = E_16(3,5)

In [25]: x
Out[25]: [1, 1, 1, 0, 0, 0]
```

In the above example $x = [1, 1, 1, 0, 0, 0]$, has 1 (x3) and 0 (x3), where 1 represents possible solution case and 0 represents failure. As we can compare for $i = 3$ and $k = 5$ from the question, we get total 3 possible ways. Hence there are 1 (x3) in the list, ways.

If green tiles are chosen there are three ways.



```
for j in range(i, k+1):
    ways[j] += ways[j - 1] + ways[j - i]
```

Using the for loop we are iterating from i to 50, we have written $k+1$ since iterating in the loop through `range()` will exclude the last case. In `ways[j] += ways[j - 1] + ways[j - i]`, we are updating the j th index of the list `ways` with the summation of $(j-1)$ and $(j+i)$ th index's value and also the j th index value (Since `+=`). These are the possible values of j , for $i=3$ and $k=5$.

```
In [31]: for j in range(3, 5+1)
    ...:     print(j)
    ...:
3
4
5
```

Finally we return our list as `return ways[k] - 1`. So, for $i=3$ and $k=5$, this gives the solution(i.e, Ans = 3)

```
In [32]: for j in range(3, 5+1):
    ...:     x[j] += x[j - 1]
    ...:

In [33]: x[5]-1
Out[33]: 3

In [34]: x
Out[34]: [1, 1, 1, 2, 3, 4]
```

Lastly in our code,

```
print("Number of black tiles =", k, "units")
print("Number of ways to fill:", E_116(2, k) + E_116(3, k) + E_116(4, k))
```

We print away our result using the `print()` function. The first statement prints ***Number of black tiles = 50*** and the second statement prints ***Number of ways to fill: 20492570929*** which is the desired answer to the problem.

```
# Project Euler Problem 116

k = 50
def E_116(i, k):
    ways = [1] * i + [0] * (k-i+1)
    for j in range(i, k+1):
        ways[j] += ways[j - 1] + ways[j - i]
    return ways[k] - 1

print("Number of black tiles =", k, "units")
print("Number of ways to fill:", E_116(2, k) + E_116(3, k) + E_116(4, k))
```

Resources:

- [projecteuler](#)
- [Java solution _ mathblog](#)

Source

<https://www.geeksforgeeks.org/project-euler/>

Chapter 131

Python Input Methods for Competitive Programming

Python Input Methods for Competitive Programming - GeeksforGeeks

Python is an amazingly user friendly language with the only flaw of being slow. In comparison to C, C++ and Java, it is quite slower. On Online coding platforms, if C/C++ limit provided is **X**. Usually, in Java time provided is **2X** and Python, it's **5X**.

To improve the speed of code execution for input/output intensive problems, languages have various input and output procedures.

An Example Problem :

Consider a question of finding the sum of **N** numbers inputted from the user.

Input a number **N**.

Input **N** numbers separated by a single space in a line.

Examples:

```
Input :  
5  
1 2 3 4 5  
Output :  
10
```

Different Python solutions for above Problem :

Normal Method Python: (Python 2.7)

1. `raw_input()` takes an optional prompt argument. It also strips the trailing newline character from the string it returns.
2. `print` is just a thin wrapper that formats the inputs (space between args and newline at the end) and calls the write function of a given object.

```
# basic method of input output
# input N
n = int(raw_input())

# input the array
arr = [int(x) for x in raw_input().split()]

# initialize variable
summation = 0

# calculate sum
for x in arr:
    summation += x

# print answer
print(summation)
```

A bit faster method using inbuilt stdin, stdout: (Python 2.7)

1. `sys.stdin` on the other hand is a **File Object**. It is like creating any other file object one could create to read input from the file. In this case, the file will be standard input buffer.
2. `stdout.write('D\n')` is faster than `print 'D'`.
3. Even faster is to write all once by `stdout.write("".join(list-comprehension))` but this makes memory usage dependent on size of input.

```
# import inbuilt standard input output
from sys import stdin, stdout

# suppose a function called main() and
# all the operations are performed
def main():

    # input via readline method
    n = stdin.readline()

    # array input similar method
    arr = [int(x) for x in stdin.readline().split()]

    #initialize variable
    summation = 0

    # calculate sum
    for x in arr:
        summation += x

    # could use inbuilt summation = sum(arr)

    # print answer via write
    # write method writes only
```

```
# string operations
# so we need to convert any
# data into string for input
stdout.write(str(summation))

# call the main method
if __name__ == "__main__":
    main()
```

Difference in time:

Timing summary (100k lines each)

```
Print : 6.040 s
Write to file : 0.122 s
Print with Stdout : 0.121 s
```

Adding a buffered pipe io: (Python 2.7)

1. Simply, **adding the buffered IO** code before your submission code to make the output faster.
2. The benefit of **io.BytesIO** objects is that they implement a common-ish interface (commonly known as a ‘file-like’ object). **BytesIO** objects have an internal pointer and for every call to read(n) the pointer advances.
3. The **atexit** module provides a simple interface to register functions to be called when a program closes down normally. The **sys** module also provides a hook, **sys.exitfunc**, but only one function can be registered there. The **atexit registry** can be used by multiple modules and libraries simultaneously.

```
# template begins
#####
#
# import libraries for input/ output handling
# on generic level
import atexit, io, sys

# A stream implementation using an in-memory bytes
# buffer. It inherits BufferedIOBase.
buffer = io.BytesIO()
sys.stdout = buffer

# print via here
@atexit.register
def write():
    sys.__stdout__.write(buffer.getvalue())
#
# template ends
```

```
# normal method followed
# input N
n = int(raw_input())

# input the array
arr = [int(x) for x in raw_input().split()]

# initialize variable
summation = 0

# calculate sum
for x in arr:
    summation += x

# print answer
print(summation)
```

While handling large amount of data usually, the normal method fails to execute within the time limit. The method 2 helps in maintaining the large amount of I/O data. The method 3 is fastest. Usually, handling of input data file greater than 2 or 3 MBs is helped via method 2 and 3.

Note : above mention codes are in Python 2.7, to use in Python 3.X versions. Simply replace the `raw_input()` with Python 3.X's `input()` syntax. Rest should work fine.

References:

- 1.[More About Input in Python 2.7](#)
- 2.[Output via sys library and other commands.](#)
- 3.[Input via sys library and other commands.](#)
4. [Python atexit Module docs.](#)

Source

<https://www.geeksforgeeks.org/python-input-methods-competitive-programming/>

Chapter 132

Python Tricks for Competitive Coding

Python Tricks for Competitive Coding - GeeksforGeeks

Python is one such programming language which makes everything easier and straight forward. Anyone who has dabbled in python for Competitive Coding gets somewhat addicted to its many features. Here is list of some of its cool features that that I've found most useful in a competitive coding environment.

1. The `most_common` function of the Counter Package.

This is probably the most useful function I've ever used and its always at the back of my mind while writing any python code. This function analyses a list/string and helps to return the top **n** entities in the list/string according to their number of occurrences in descending order where **n** is a number that is specified by the programmer. The individual entities are returned along with their number of occurrences in a **tuple** which can easily be referred/printed as and when required.

```
# Code to find top 3 elements and their counts
# using most_common
from collections import Counter

arr = [1, 3, 4, 1, 2, 1, 1, 3, 4, 3, 5, 1, 2, 5, 3, 4, 5]
counter = Counter(arr)
top_three = counter.most_common(3)
print(top_three)
```

Output:

```
[(1, 5), (3, 4), (4, 3)]
```

The output tuple clearly states that 1 has occurred 5 times, 3 has occurred 4 times, and 4 has occurred 3 times.

2. The **n-largest/n-smallest** function of the **heapq** Package.

This function helps to return the top **n** smallest/largest elements in any lists and here again **n** is a number specified by the programmer.

```
# Python code to find 3 largest and 4 smallest
# elements of a list.

import heapq

grades = [110, 25, 38, 49, 20, 95, 33, 87, 80, 90]
print(heapq.nlargest(3, grades))
print(heapq.nsmallest(4, grades))
```

Output:

```
[110, 95, 90]
[20, 25, 33, 38]
```

The first line of output gives 3 of the largest numbers present in the list `grades`. Similarly the second line of output prints out 4 of the smallest elements present in the list `grades`. Another **speciality** of this function is that it does not overlook repetitions. So in place of **n** if we were to place the length of the array the we would end up with the entire sorted array itself !!

3. Dictionary and concept of zipping Dictionaries

Dictionaries in python are truly fascinating in terms of the unique functionality that they offer. They are stored as a **Key and Value pair** in the form of an array like structure. Each value can be accessed by its corresponding key.

The `zip` function is used to join two lists together or we can even join the key and value pairs in a dictionary together as a single list. The application of this concept will be made clear in the following code snippet.

```
# Python code to demonstrate use of zip.

import heapq

stocks = {
    'Goog' : 520.54,
    'FB' : 76.45,
    'yhoo' : 39.28,
    'AMZN' : 306.21,
    'APPL' : 99.76
}

zipped_1 = zip(stocks.values(), stocks.keys())
```

```
# sorting according to values
print(sorted(zipped_1))

zipped_2 = zip(stocks.keys(), stocks.values())
print(sorted(zipped_2))
#sorting according to keys
```

Output:

```
[(39.28, 'yhoo'), (76.45, 'FB'), (99.76, 'APPL'), (306.21, 'AMZN'), (520.54, 'Goog')]
[('AMZN', 306.21), ('APPL', 99.76), ('FB', 76.45), ('Goog', 520.54), ('yhoo', 39.28)]
```

4. The Map function.

This function is a sneaky little shortcut that allows us to implement a simple function on a list of values in a very **Unconventional Manner**. The following example will give a simple application of this functionality. The function takes as parameters the function name and the name of the list the function needs to be applied upon.

```
# Python code to apply a function on a list
income = [10, 30, 75]

def double_money(dollars):
    return dollars * 2

new_income = list(map(double_money, income))
print(new_income)
```

Output:

```
[20, 60, 150]
```

Here, we just implemented a simple function which multiplies each list value by two and returns it as a new list.

Individually these functions might look innocent but will definitely come in handy in a **TIME LIMITED CODING ENVIRONMENT** in the sense that they offer large functionality in a **VERY** short amount of code. The functionalities discussed have very specific applications and act like a **SHORTCUT** or a **CHEAT-SHEET** in competitive coding. Having these useful tricks up your sleeve might just give someone the **COMPETITIVE EDGE** that they were looking for !!

Source

<https://www.geeksforgeeks.org/python-tricks-competitive-coding/>

Chapter 133

Python in Competitive Programming

Python in Competitive Programming - GeeksforGeeks

In 2017, when ACM allowed Python support for its prestigious competition, the ACM ICPC, a whole new community became interested in the sport of competitive programming. This meant more people coming back to the basics, learning algorithms that are the building blocks of complex packages they use to build their high level packages.

Unfortunately, not a lot of information exists out there on how to effectively use data structures and even the scoping rules of python which lead people to believe that python is subpar for competitive programming.

Today I'll show you how python sometimes is even more powerful than C++ or Java thanks to its amazing libraries, and how simple it actually can be.

Let me demonstrate with a simple example, look at the following snippets of code-

```
alphabets = ['a', 'b', 'c']
for item in alphabets:
    len(item)

alphabets = ['a', 'b', 'c']
fn = len
for item in alphabets:
    fn(item)
```

You might think I've assigned an alias to the function 'len' and it might not make a difference. So i wrote a performance testing function as follows.

```
import datetime
alphabets = [str(x) for x in range(10000000)]
a = datetime.datetime.now() # store initial time
```

```
for item in alphabets:  
    len(item)  
b = datetime.datetime.now() # store final time  
print (b-a).total_seconds() # results  
a = datetime.datetime.now()  
fn = len                      # function stored locally  
for item in alphabets:  
    fn(item)  
b = datetime.datetime.now()  
print (b-a).total_seconds()
```

I encourage you to try it on your systems.

Here's the output on mine on running the performance.py script.

```
k@kVato:~/Desktop$ python gfg.py  
0.883164  
0.474463  
k@kVato:~/Desktop$
```

Almost half!

Okay, now let's try to analyse why this happened. Reason? Lookup for a function is a costly operation.

In the second snippet, I stored the function directly in the scope of the function, so it doesn't matter how many times I call it, each time the runtime knows exactly where it has to look for the results.

Itertools

If you've been to codeforces, you know by now that the a lot of programming challenges involve backtracking. So today I'll tell you about a library to generate all permutations and combinations using a inbuilt library package which is extremely fast. Itertools. If you're looking to solve algorithmic challenges with python then itertools is library you must definitely explore.

To generate all permutations –

```
from itertools import permutations  
perm = permutations([1, 2, 3], 2)  
for i in list(perm):  
    print i  
  
# Answer->(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)
```

The combinations() functions behaves similarly I encourage the readers to try it on their own.

Python is a slow language only if your code is not leveraging the power of it successfully. Do not feel like you are at a disadvantage if you're a python coder, it's actually very neat and very quick!

Source

<https://www.geeksforgeeks.org/python-in-competitive-programming/>

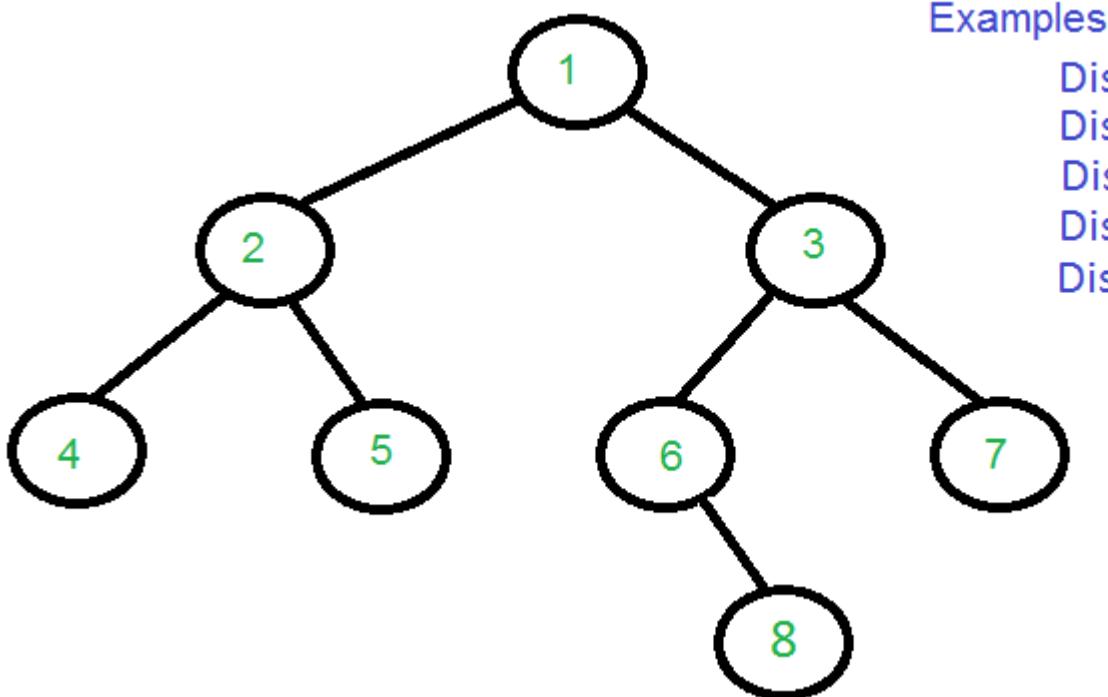
Chapter 134

Queries to find distance between two nodes of a Binary tree

Queries to find distance between two nodes of a Binary tree - GeeksforGeeks

Given a binary tree, the task is to find the distance between two keys in a binary tree, no parent pointers are given. The distance between two nodes is the minimum number of edges to be traversed to reach one node from other.

It has been already discussed in [this](#) for a single query in $O(\log n)$ time, here the task is to reduce multiple queries time to $O(1)$ by compromising with space complexity to $O(N \log n)$. In this post, we will use **Sparse table** instead of segment tree for finding the minimum in given range, which uses dynamic programming and bit manipulation to achieve $O(1)$ query time.



A sparse table will preprocess the minimum values given for an array in $N \log n$ space i.e. each node will contain chain of values of $\log(i)$ length where i is the index of the i th node in L array. Each entry in the sparse table says $M[i][j]$ will represent the index of the minimum value in the subarray starting at i having 2^j .

The distance between two nodes can be obtained in terms of lowest common ancestor.

$$\text{Dist}(n_1, n_2) = \text{Level}[n_1] + \text{Level}[n_2] - 2 * \text{Level}[\text{lca}]$$

This problem can be breakdown into **finding levels of each node**, **finding the Euler tour of binary tree** and **building sparse table** for LCA, these steps are explained below :

1. Find the levels of each node by applying [level order traversal](#).
2. Find the LCA of two nodes in the binary tree in $O(\log n)$ by Storing [Euler tour of tree](#) in array and computing two other arrays with the help of levels of each node and Euler tour.

These steps are shown below:

- (I) First, find [Euler Tour of binary tree](#).

Euler	1	2	4	2	5	2	1	3	6	8	6	3
	1	2	3	4	5	6	7	8	9	10	11	12
Euler tour of Binary Tree												

(II) Then, store levels of each node in Euler array.

L	0	1	2	1	2	1	0	1	2	3	2	1
	1	2	3	4	5	6	7	8	9	10	11	12
Levels of each node in Euler tour												

(III) Then, store First occurrences of all nodes of binary tree in Euler array.

H	1	2	8	3	5	9	13	10
	1	2	3	4	5	6	7	8
First occurrences of each node in Euler tree								

3. Then build sparse table on L array and find the minimum value say X in range ($H[A]$ to $H[B]$). Then use the index of value X as an index to Euler array to get LCA, i.e. $Euler[\text{index}(X)]$.

Let, A=8 and B=5.

(I) $H[8]=1$ and $H[5]=2$

(II) we get min value in L array between 1 and 2 as X=0, index=7

(III) Then, LCA= Euler[7], i.e LCA=1.

4. Finally, apply distance formula discussed above to get the distance between two nodes.

Source

<https://www.geeksforgeeks.org/queries-find-distance-two-nodes-binary-tree/>

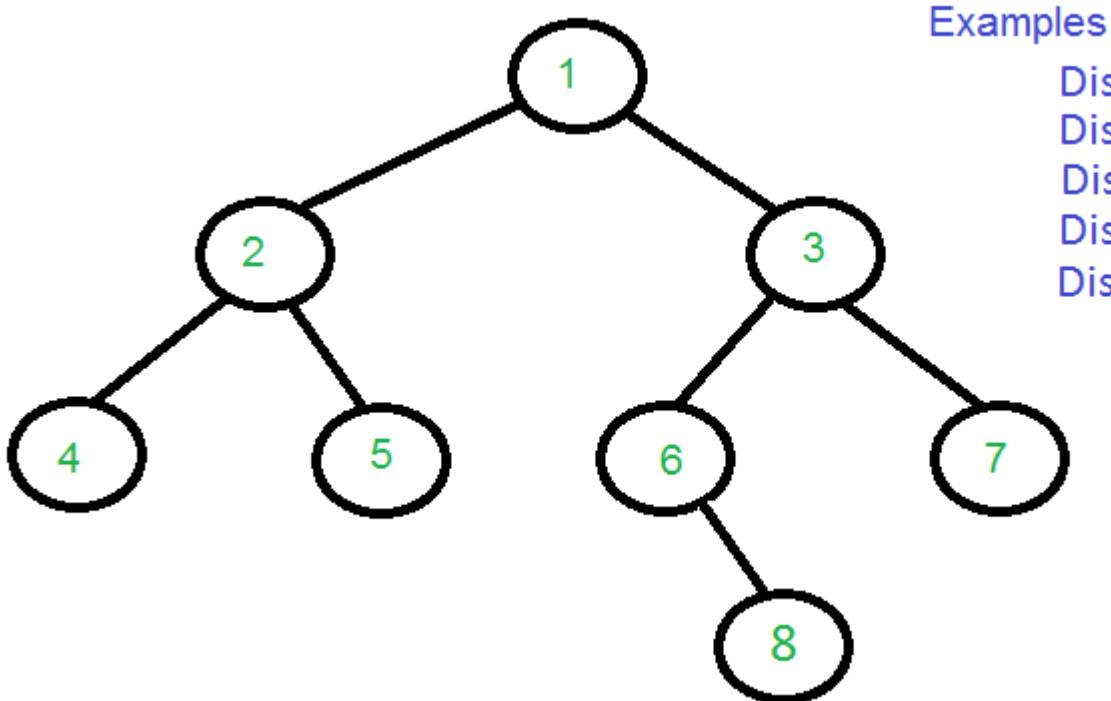
Chapter 135

Queries to find distance between two nodes of a Binary tree – O(logn) method

Queries to find distance between two nodes of a Binary tree - O(logn) method - GeeksforGeeks

Given a binary tree, the task is to find the distance between two keys in a binary tree, no parent pointers are given. Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.

This problem has been already discussed in [previous post](#) but it uses **three traversals** of the Binary tree, one for finding Lowest Common Ancestor(LCA) of two nodes(let A and B) and then two traversals for finding distance between LCA and A and LCA and B which has $O(n)$ time complexity. In this post, a method will be discussed that requires the **$O(\log(n))$** time to find LCA of two nodes.



The distance between two nodes can be obtained in terms of lowest common ancestor. Following is the formula.

$\text{Dist}(n_1, n_2) = \text{Dist}(\text{root}, n_1) + \text{Dist}(\text{root}, n_2) - 2 * \text{Dist}(\text{root}, \text{lca})$
 'n1' and 'n2' are the two given keys
 'root' is root of given Binary Tree.
 'lca' is lowest common ancestor of n1 and n2
 $\text{Dist}(n_1, n_2)$ is the distance between n1 and n2.

Above formula can also be written as:

$$\text{Dist}(n_1, n_2) = \text{Level}[n_1] + \text{Level}[n_2] - 2 * \text{Level}[\text{lca}]$$

This problem can be breakdown into:

1. Finding levels of each node
2. Finding the Euler tour of binary tree
3. Building segment tree for LCA,

These steps are explained below :

1. Find the levels of each node by applying [level order traversal](#).

2. Find the LCA of two nodes in binary tree in $O(\log n)$ by Storing Euler tour of Binary tree in array and computing two other arrays with the help of levels of each node and Euler tour.

These steps are shown below:

- (I) First, find Euler Tour of binary tree.

Euler	1	2	4	2	5	2	1	3	6	8	6	
	1	2	3	4	5	6	7	8	9	10	11	
Euler tour of Binary Tree												

Euler tour of binary tree in example

- (II) Then, store levels of each node in Euler array.

L	0	1	2	1	2	1	0	1	2	3	2	
	1	2	3	4	5	6	7	8	9	10	11	
Levels of each node in Euler tour												

- (III) Then, store First occurrences of all nodes of binary tree in Euler array. H stores the indices of nodes from Euler array, so that range of query for finding minimum can be minimized and thereby further optimizing the query time.

H	1	2	8	3	5	9	13	10	
	1	2	3	4	5	6	7	8	
First occurrences of each node in Euler tree									

3. Then **build segment tree on L array** and take the low and high values from H array that will give us the first occurrences of say Two nodes(A and B). Then, we query segment tree to find the minimum value say X in range ($H[A]$ to $H[B]$). Then we use the index of value X as index to Euler array to get LCA, i.e. Euler[index(X)].

Let, A = 8 and B = 5.

- (I) $H[8] = 1$ and $H[5] = 2$

(II) Querying on Segment tree, we get min value in L array between 1 and 2 as X=0, index=7

- (III) Then, LCA= Euler[7], i.e LCA = 1.

4. Finally, we apply distance formula discussed above to get distance between two nodes.

```
// C++ program to find distance between
// two nodes for multiple queries
#include <bits/stdc++.h>
#define MAX 100001
using namespace std;

/* A tree node structure */
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

/* Utility function to create a new Binary Tree node */
struct Node* newNode(int data)
{
    struct Node* temp = new struct Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Array to store level of each node
int level[MAX];

// Utility Function to store level of all nodes
void FindLevels(struct Node* root)
{
    if (!root)
        return;

    // queue to hold tree node with level
    queue<pair<struct Node*, int>> q;

    // let root node be at level 0
    q.push({root, 0});

    pair<struct Node*, int> p;

    // Do level Order Traversal of tree
    while (!q.empty()) {
        p = q.front();
        q.pop();

        // Node p.first is on level p.second
```

```
level[p.first->data] = p.second;

// If left child exists, put it in queue
// with current_level +1
if (p.first->left)
    q.push({ p.first->left, p.second + 1 });

// If right child exists, put it in queue
// with current_level +1
if (p.first->right)
    q.push({ p.first->right, p.second + 1 });
}

// Stores Euler Tour
int Euler[MAX];

// index in Euler array
int idx = 0;

// Find Euler Tour
void eulerTree(struct Node* root)
{
    // store current node's data
    Euler[++idx] = root->data;

    // If left node exists
    if (root->left) {

        // traverse left subtree
        eulerTree(root->left);

        // store parent node's data
        Euler[++idx] = root->data;
    }

    // If right node exists
    if (root->right) {
        // traverse right subtree
        eulerTree(root->right);

        // store parent node's data
        Euler[++idx] = root->data;
    }
}

// checks for visited nodes
```

```
int vis[MAX];

// Stores level of Euler Tour
int L[MAX];

// Stores indices of first occurrence
// of nodes in Euler tour
int H[MAX];

// Preprocessing Euler Tour for finding LCA
void preprocessEuler(int size)
{
    for (int i = 1; i <= size; i++) {
        L[i] = level[Euler[i]];

        // If node is not visited before
        if (vis[Euler[i]] == 0) {
            // Add to first occurrence
            H[Euler[i]] = i;

            // Mark it visited
            vis[Euler[i]] = 1;
        }
    }
}

// Stores values and positions
pair<int, int> seg[4 * MAX];

// Utility function to find minimum of
// pair type values
pair<int, int> min(pair<int, int> a,
                    pair<int, int> b)
{
    if (a.first <= b.first)
        return a;
    else
        return b;
}

// Utility function to build segment tree
pair<int, int> buildSegTree(int low, int high, int pos)
{
    if (low == high) {
        seg[pos].first = L[low];
        seg[pos].second = low;
        return seg[pos];
    }
}
```

```
int mid = low + (high - low) / 2;
buildSegTree(low, mid, 2 * pos);
buildSegTree(mid + 1, high, 2 * pos + 1);

seg[pos] = min(seg[2 * pos], seg[2 * pos + 1]);
}

// Utility function to find LCA
pair<int, int> LCA(int qlow, int qhigh, int low,
                     int high, int pos)
{
    if (qlow <= low && qhigh >= high)
        return {seg[pos], 0};

    if (qlow > high || qhigh < low)
        return {INT_MAX, 0};

    int mid = low + (high - low) / 2;

    return min(LCA(qlow, qhigh, low, mid, 2 * pos),
               LCA(qlow, qhigh, mid + 1, high, 2 * pos + 1));
}

// Function to return distance between
// two nodes n1 and n2
int findDistance(int n1, int n2, int size)
{
    // Maintain original Values
    int prevn1 = n1, prevn2 = n2;

    // Get First Occurrence of n1
    n1 = H[n1];

    // Get First Occurrence of n2
    n2 = H[n2];

    // Swap if low > high
    if (n2 < n1)
        swap(n1, n2);

    // Get position of minimum value
    int lca = LCA(n1, n2, 1, size, 1).second;

    // Extract value out of Euler tour
    lca = Euler[lca];

    // return calculated distance
    return level[prevn1] + level[prevn2] - 2 * level[lca];
}
```

```
}

void preProcessing(Node* root, int N)
{
    // Build Tree
    eulerTree(root);

    // Store Levels
    FindLevels(root);

    // Find L and H array
    preprocessEuler(2 * N - 1);

    // Build segment Tree
    buildSegTree(1, 2 * N - 1, 1);
}

/* Driver function to test above functions */
int main()
{
    int N = 8; // Number of nodes

    /* Constructing tree given in the above figure */
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);

    // Function to do all preprocessing
    preProcessing(root, N);

    cout << "Dist(4, 5) = " << findDistance(4, 5, 2 * N - 1) << "\n";
    cout << "Dist(4, 6) = " << findDistance(4, 6, 2 * N - 1) << "\n";
    cout << "Dist(3, 4) = " << findDistance(3, 4, 2 * N - 1) << "\n";
    cout << "Dist(2, 4) = " << findDistance(2, 4, 2 * N - 1) << "\n";
    cout << "Dist(8, 5) = " << findDistance(8, 5, 2 * N - 1) << "\n";

    return 0;
}
```

Output:

```
Dist(4, 5) = 2
```

```
Dist(4, 6) = 4  
Dist(3, 4) = 3  
Dist(2, 4) = 1  
Dist(8, 5) = 5
```

Time Complexity: $O(\log N)$

Space Complexity: $O(N)$

[Queries to find distance between two nodes of a Binary tree – O\(1\) method](#)

Source

<https://www.geeksforgeeks.org/queries-find-distance-two-nodes-binary-tree-ologn-method/>

Chapter 136

Queries to find maximum product pair in range with updates

Queries to find maximum product pair in range with updates - GeeksforGeeks

Given an array of N positive integers. The task is to perform the following operations according to the type of query given.

1. Print the maximum pair product in a given range. [L-R]
2. Update A_i with some given value.

Examples:

Input: A={1, 3, 4, 2, 5}

Queries:

Type 1: L = 0, R = 2.

Type 2: i = 1, val = 6

Type 1: L = 0, R = 2.

Output:

12

24

For the query1, the maximum product in a range [0, 2] is $3*4 = 12$.

For the query2, after an update, the array becomes [1, 6, 4, 2, 5]

For the query3, the maximum product in a range [0, 2] is $6*4 = 24$.

Naive Solution: The brute force approach is to traverse from L to R and check for every pair and then find the maximum product pair among them.

Time Complexity: $O(N^2)$ for every query.

A better solution is to find the first and the second largest number in the range L to R by traversing and then returning their product.

Time Complexity: $O(N)$ for every query.

A **efficient solution** is to use a [segment tree](#) to store the largest and second largest number in the nodes and then return the product of them.

Below is the implementation of above approach.

```
// C++ program to find the maximum
// product in a range with updates
#include <bits/stdc++.h>
using namespace std;
#define ll long long

// structure defined
struct segment {
    // l for largest
    // sl for second largest
    ll l;
    ll sl;
};

// function to perform queries
segment query(segment* tree, ll index,
              ll s, ll e, ll qs, ll qe)
{
    segment res;
    res.l = -1;
    res.sl = -1;
    // no overlapping case
    if (qs > e || qe < s || s > e) {

        return res;
    }

    // complete overlap case
    if (s >= qs && e <= qe) {

        return tree[index];
    }

    // partial overlap case
    ll mid = (s + e) / 2;

    // calling left node and right node
    segment left = query(tree, 2 * index,
                         s, mid, qs, qe);
    segment right = query(tree, 2 * index + 1,
                          mid + 1, e, qs, qe);

    // largest of (left.l, right.l)
```

```
ll largest = max(left.l, right.l);

// compute second largest
// second largest will be minimum
// of maximum from left and right node
ll second_largest = min(max(left.l, right.sl),
                        max(right.l, left.sl));

// store largest and
// second_largest in res
res.l = largest;
res.sl = second_largest;

// return the resulting node
return res;
}

// function to update the query
void update(segment* tree, ll index,
           ll s, ll e, ll i, ll val)
{
    // no overlapping case
    if (i < s || i > e) {
        return;
    }

    // reached leaf node

    if (s == e) {
        tree[index].l = val;
        tree[index].sl = INT_MIN;
        return;
    }

    // partial overlap

    ll mid = (s + e) / 2;

    // left subtree call
    update(tree, 2 * index, s, mid, i, val);

    // right subtree call
    update(tree, 2 * index + 1, mid + 1, e, i, val);

    // largest of (left.l, right.l)
    tree[index].l = max(tree[2 * index].l, tree[2 * index + 1].l);

    // compute second largest
```

```
// second largest will be
// minimum of maximum from left and right node

tree[index].sl = min(max(tree[2 * index].l, tree[2 * index + 1].sl),
                     max(tree[2 * index + 1].l, tree[2 * index].sl));
}

// Function to build the tree
void buildtree(segment* tree, ll* a, ll index, ll s, ll e)
{
    // tree is build bottom to up
    if (s > e) {
        return;
    }

    // leaf node
    if (s == e) {
        tree[index].l = a[s];
        tree[index].sl = INT_MIN;
        return;
    }

    ll mid = (s + e) / 2;

    // calling the left node
    buildtree(tree, a, 2 * index, s, mid);

    // calling the right node
    buildtree(tree, a, 2 * index + 1, mid + 1, e);

    // largest of (left.l, right.l)
    ll largest = max(tree[2 * index].l, tree[2 * index + 1].l);

    // compute second largest
    // second largest will be minimum
    // of maximum from left and right node
    ll second_largest = min(max(tree[2 * index].l, tree[2 * index + 1].sl),
                           max(tree[2 * index + 1].l, tree[2 * index].sl));

    // storing the largest and
    // second_largest values in the current node
    tree[index].l = largest;
    tree[index].sl = second_largest;
}

// Driver Code
int main()
{
```

```
// your code goes here
ll n = 5;

ll a[5] = { 1, 3, 4, 2, 5 };

// allocating memory for segment tree
segment* tree = new segment[4 * n + 1];

// buildtree(tree, a, index, start, end)
buildtree(tree, a, 1, 0, n - 1);

// query section
// storing the resulting node
segment res = query(tree, 1, 0, n - 1, 0, 2);

cout << "Maximum product in the range "
     << "0 and 2 before update: " << (res.l * res.sl);

// update section
// update(tree, index, start, end, i, v)
update(tree, 1, 0, n - 1, 1, 6);

res = query(tree, 1, 0, n - 1, 0, 2);

cout << "\nMaximum product in the range "
     << "0 and 2 after update: " << (res.l * res.sl);

return 0;
}
```

Output:

```
Maximum product in the range 0 and 2 before update: 12
Maximum product in the range 0 and 2 after update: 24
```

Time Complexity: $O(\log N)$ per query and $O(N)$ for building the tree.

Source

<https://www.geeksforgeeks.org/queries-to-find-maximum-product-pair-in-range-with-updates/>

Chapter 137

Querying the number of distinct colors in a subtree of a colored tree using BIT

Querying the number of distinct colors in a subtree of a colored tree using BIT - Geeks-forGeeks

Prerequisites : [BIT](#), [DFS](#)

Given a rooted tree T, with ‘n’ nodes, each node has a color denoted by the array color[] (color[i] denotes the color of ith node in form of an integer). Respond to ‘Q’ queries of the following type:

- **distinct u** – Print the number of distinct colored nodes under the subtree rooted under ‘u’

Examples:

```
      1
     / \
    2   3
   / \ | \
  4 5 6 7 8
   | \
  9 10 11

color[] = {0, 2, 3, 3, 4, 1, 3, 4, 3, 2, 1, 1}
Indexes   NA 1 2 3 4 5 6 7 8 9 10 11
(Node Values and colors start from index 1)

distinct 3 -> output should be '4'.
```

There are six different nodes in subtree rooted with 3, nodes are 3, 7, 8, 9, 10 and 11. These nodes have four distinct colors (3, 4, 2 and 1)

distinct 2 -> output should be '3'.
 distinct 7 -> output should be '3'.

Building a solution in steps:

1. Flatten the tree using DFS; store the visiting time and ending time for every node in two arrays, **vis_time[i]** stores the visiting time of the ith node while **end_time[i]** stores the ending time.
2. In the same DFS call, store the value of color of every node in an array **flat_tree[]**, at indices: **vis_time[i]** and **end_time[i]** for ith node.
 Note: size of the array **flat_tree[]** will be $2n$.

Now the problem is reduced to finding the number of distinct elements in the range $[vis_time[u], end_time[u]]$ in the array **flat_tree[]** for each query of the specified type. To do so, we will process the queries off-line(processing the queries in an order different than the one provided in the question, and storing the results, and finally printing the result for each in the order specified in the question).

Steps:

1. First, we pre-process the array **flat_tree[]**; we maintain a **table[]**(an array of vectors), **table[i]** stores the vector containing all the indices in **flat_tree[]** that have value i. That is, if **flat_tree[j] = i**, then **table[i]** will have one of its element **j**.
2. In BIT, we update '1' at ith index if we want the ith element of **flat_tree[]** to be counted in **query()** method. We now maintain another array **traverser[]**; **traverser[i]** contains the pointer to the next element of **table[i]** that is not marked in BIT yet.
3. We now update our BIT and set '1' at first occurrence of every element in **flat_tree[]** and increment corresponding **traverser[]** by '1'(if **flat_tree[i]** is occurring for the first time then **traverser[flat_tree[i]]** is incremented by '1') to point to the next occurrence of that element.
4. Now our **query(R)** function for BIT would return the number of distinct elements in **flat_tree[]** in the range $[1, R]$.
5. We sort all the queries in order of increasing **vis_time[]**, let l_i denote **vis_time[i]** and r_i denote the **end_time[i]**. Sorting the queries in increasing order of l_i gives us an edge, as when processing the ith query we won't see any query in future with its ' l ' smaller than l_i . So we can remove all the elements' occurrences up to $l_i - 1$ from BIT and add their next occurrences using the **traverser[]** array. And then **query(R)** would return the number of distinct elements in the range $[l_i, r_i]$.

```
// A C++ program implementing the above design
#include<bits/stdc++.h>
#define max_color 1000005
```

```
#define maxn 100005
using namespace std;

// Note: All elements of global arrays are
// initially zero
// All the arrays have been described above
int bit[maxn], vis_time[maxn], end_time[maxn];
int flat_tree[2 * maxn];
vector<int> tree[maxn];
vector<int> table[max_color];
int traverser[max_color];

bool vis[maxn];
int tim = 0;

//li, ri and index are stored in queries vector
//in that order, as the sort function will use
//the value li for comparison
vector< pair< pair<int, int>, int > > queries;

//ans[i] stores answer to ith query
int ans[maxn];

//update function to add val to idx in BIT
void update(int idx, int val)
{
    while ( idx < maxn )
    {
        bit[idx] += val;
        idx += idx & -idx;
    }
}

//query function to find sum(1, idx) in BIT
int query(int idx)
{
    int res = 0;
    while ( idx > 0 )
    {
        res += bit[idx];
        idx -= idx & -idx;
    }
    return res;
}

void dfs(int v, int color[])
{
    //mark the node visited
```

```
vis[v] = 1;

//set visiting time of the node v
vis_time[v] = ++tim;

//use the color of node v to fill flat_tree[]
flat_tree[tim] = color[v];

vector<int>::iterator it;
for (it=tree[v].begin(); it!=tree[v].end(); it++)
    if (!vis[*it])
        dfs(*it, color);

// set ending time for node v
end_time[v] = ++tim;

// setting its color in flat_tree[] again
flat_tree[tim] = color[v];
}

//function to add an edge(u, v) to the tree
void addEdge(int u, int v)
{
    tree[u].push_back(v);
    tree[v].push_back(u);
}

//function to build the table[] and also add
//first occurrences of elements to the BIT
void hashMarkFirstOccurrences(int n)
{
    for (int i = 1 ; i <= 2 * n ; i++)
    {
        table[flat_tree[i]].push_back(i);

        //if it is the first occurrence of the element
        //then add it to the BIT and increment traverser
        if (table[flat_tree[i]].size() == 1)
        {
            //add the occurrence to bit
            update(i, 1);

            //make traverser point to next occurrence
            traverser[flat_tree[i]]++;
        }
    }
}
```

```

//function to process all the queries and store thier answers
void processQueries()
{
    int j = 1;
    for (int i=0; i<queries.size(); i++)
    {
        //for each query remove all the ocurences before its li
        //li is the visiting time of the node
        //which is stored in first element of first pair
        for ( ; j < queries[i].first.first ; j++ )
        {
            int elem = flat_tree[j];

            //update(i, -1) removes an element at ith index
            //in the BIT
            update( table[elem] [traverser[elem] - 1], -1);

            //if there is another occurrence of the same element
            if ( traverser[elem] < table[elem].size() )
            {
                //add the occurrence to the BIT and
                //increment traverser
                update(table[elem] [ traverser[elem] ], 1);
                traverser[elem]++;
            }
        }

        //store the answer for the query, the index of the query
        //is the second element of the pair
        //And ri is stored in second element of the first pair
        ans[queries[i].second] = query(queries[i].first.second);
    }
}

// Count distinct colors in subtrees rooted with qVer[0],
// qVer[1], ...qVer[qn-1]
void countDistinctColors(int color[], int n, int qVer[], int qn)
{
    // build the flat_tree[], vis_time[] and end_time[]
    dfs(1, color);

    // add query for u = 3, 2 and 7
    for (int i=0; i<qn; i++)
        queries.push_back(make_pair(make_pair(vis_time[qVer[i]],
                                              end_time[qVer[i]]), i));

    // sort the queries in order of increasing vis_time

```

```
sort(queries.begin(), queries.end());

// make table[] and set '1' at first occurrences of elements
hashMarkFirstOccurrences(n);

// process queries
processQueries();

// print all the answers, in order asked
// in the question
for (int i=0; i<queries.size() ; i++)
{
    cout << "Distinct colors in the corresponding subtree"
    "is: " << ans[i] << endl;
}
}

//driver code
int main()
{
/*
          1
         / \
        2   3
       / \   | \
      4 5 6 7   8
       / \
      9 10 11
*/
int n = 11;
int color[] = {0, 2, 3, 3, 4, 1, 3, 4, 3, 2, 1, 1};

// add all the edges to the tree
addEdge(1, 2);
addEdge(1, 3);
addEdge(2, 4);
addEdge(2, 5);
addEdge(2, 6);
addEdge(3, 7);
addEdge(3, 8);
addEdge(7, 9);
addEdge(7, 10);
addEdge(7, 11);

int qVer[] = {3, 2, 7};
int qn = sizeof(qVer)/sizeof(qVer[0]);

countDistinctColors(color, n, qVer, qn);
```

```
    return 0;  
}
```

Output:

```
Distinct colors in the corresponding subtree is:4  
Distinct colors in the corresponding subtree is:3  
Distinct colors in the corresponding subtree is:3
```

Time Complexity: $O(Q * \log(n))$

Source

<https://www.geeksforgeeks.org/querying-the-number-of-distinct-colors-in-a-subtree-of-a-colored-tree-using-bit/>

Chapter 138

Quick ways to check for Prime and find next Prime in Java

Quick ways to check for Prime and find next Prime in Java - GeeksforGeeks

Many programming contest problems are somehow related Prime Numbers. Either we are required to check Prime Numbers, or we are asked to perform certain functions for all prime number between 1 to N. Example: Calculate the sum of all prime numbers between 1 and 1000000.

Java provides two function under [java.math.BigInteger](#) to deal with Prime Numbers.

isProbablePrime(int certainty): A method in [BigInteger class](#) to check if a given number is prime. For certainty = 1, it return true if BigInteger is prime and false if BigInteger is composite.

Below is Java program to demonstrate above function.

```
// A Java program to check if a number is prime using
// inbuilt function
import java.util.*;
import java.math.*;

class CheckPrimeTest
{
    //Function to check and return prime numbers
    static boolean checkPrime(long n)
    {
        // Converting long to BigInteger
        BigInteger b = new BigInteger(String.valueOf(n));

        return b.isProbablePrime(1);
    }
}
```

```
// Driver method
public static void main (String[] args)
                        throws java.lang.Exception
{
    long n = 13;
    System.out.println(checkPrime(n));
}
}
```

Output:

```
true
```

nextProbablePrime() : Another method present in BigInteger class. This functions returns the next Prime Number greater than current BigInteger.

Below is Java program to demonstrate above function.

```
// Java program to find prime number greater than a
// given number using built in method
import java.util.*;
import java.math.*;

class NextPrimeTest
{
    // Function to get nextPrimeNumber
    static long nextPrime(long n)
    {
        BigInteger b = new BigInteger(String.valueOf(n));
        return Long.parseLong(b.nextProbablePrime().toString());
    }

    // Driver method
    public static void main (String[] args)
                        throws java.lang.Exception
    {
        long n = 14;
        System.out.println(nextPrime(n));
    }
}
```

Output:

Source

<https://www.geeksforgeeks.org/quick-ways-to-check-for-prime-and-find-next-prime-in-java/>

Chapter 139

Range Minimum Query (Square Root Decomposition and Sparse Table)

Range Minimum Query (Square Root Decomposition and Sparse Table) - GeeksforGeeks

We have an array $\text{arr}[0 \dots n-1]$. We should be able to efficiently find the minimum value from index L (query start) to R (query end) where $0 \leq L \leq R \leq n-1$. Consider a situation when there are many range queries.

Example:

```
Input: arr[] = {7, 2, 3, 0, 5, 10, 3, 12, 18};  
       query[] = [0, 4], [4, 7], [7, 8]
```

```
Output: Minimum of [0, 4] is 0  
        Minimum of [4, 7] is 3  
        Minimum of [7, 8] is 12
```

A **simple solution** is to run a loop from L to R and find minimum element in given range. This solution takes $O(n)$ time to query in worst case.

Another approach is to use **Segment tree**. With segment tree, preprocessing time is $O(n)$ and time to for range minimum query is $O(\log n)$. The extra space required is $O(n)$ to store the segment tree. Segment tree allows updates also in $O(\log n)$ time.

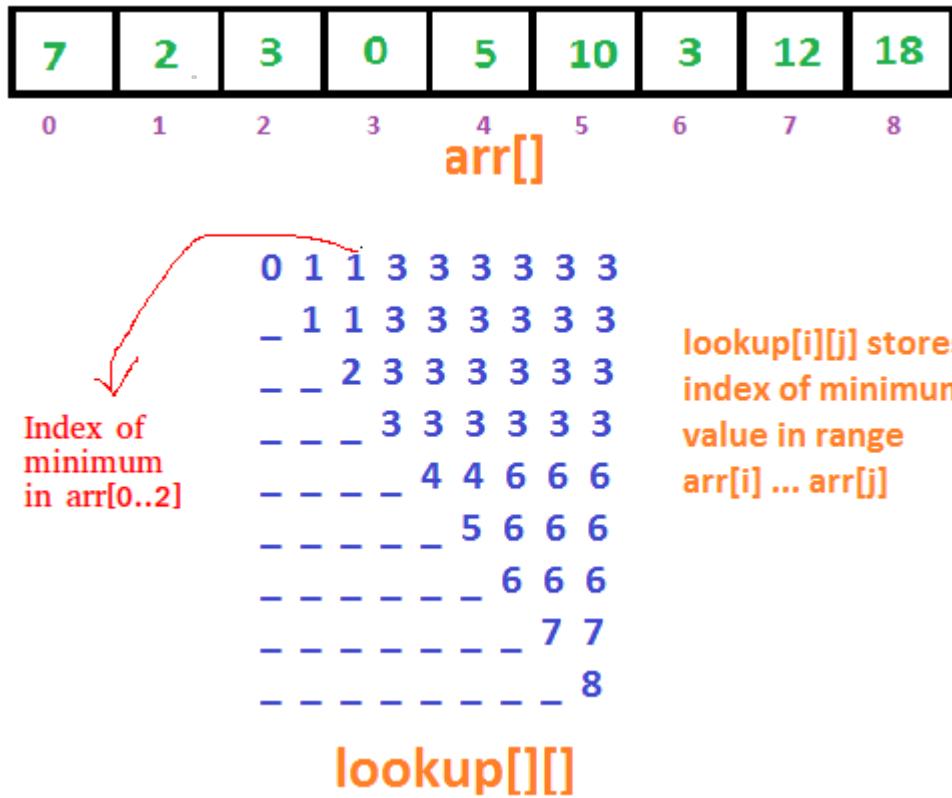
How to optimize query time when there are no update operations and there are many range minimum queries?

Below are different methods.

Method 1 (Simple Solution)

A Simple Solution is to create a 2D array $\text{lookup}[][]$ where an entry $\text{lookup}[i][j]$ stores the

minimum value in range $\text{arr}[i..j]$. Minimum of a given range can now be calculated in $O(1)$ time.



```
// C++ program to do range minimum query in O(1) time with O(n*n)
// extra space and O(n*n) preprocessing time.
#include<bits/stdc++.h>
using namespace std;
#define MAX 500

// lookup[i][j] is going to store index of minimum value in
// arr[i..j]
int lookup[MAX][MAX];

// Structure to represent a query range
struct Query
{
    int L, R;
};

// Fills lookup array lookup[n][n] for all possible values of
// query ranges
```

```

void preprocess(int arr[], int n)
{
    // Initialize lookup[][] for the intervals with length 1
    for (int i = 0; i < n; i++)
        lookup[i][i] = i;

    // Fill rest of the entries in bottom up manner
    for (int i=0; i<n; i++)
    {
        for (int j = i+1; j<n; j++)

            // To find minimum of [0,4], we compare minimum of
            // arr[lookup[0][3]] with arr[4].
            if (arr[lookup[i][j - 1]] < arr[j])
                lookup[i][j] = lookup[i][j - 1];
            else
                lookup[i][j] = j;
    }
}

// Prints minimum of given m query ranges in arr[0..n-1]
void RMQ(int arr[], int n, Query q[], int m)
{
    // Fill lookup table for all possible input queries
    preprocess(arr, n);

    // One by one compute sum of all queries
    for (int i=0; i<m; i++)
    {
        // Left and right boundaries of current range
        int L = q[i].L, R = q[i].R;

        // Print sum of current query range
        cout << "Minimum of [" << L << ", "
             << R << "] is " << arr[lookup[L][R]] << endl;
    }
}

// Driver program
int main()
{
    int a[] = {7, 2, 3, 0, 5, 10, 3, 12, 18};
    int n = sizeof(a)/sizeof(a[0]);
    Query q[] = {{0, 4}, {4, 7}, {7, 8}};
    int m = sizeof(q)/sizeof(q[0]);
    RMQ(a, n, q, m);
    return 0;
}

```

Output:

```
Minimum of [0, 4] is 0
Minimum of [4, 7] is 3
Minimum of [7, 8] is 12
```

This approach supports query in $O(1)$, but preprocessing takes $O(n^2)$ time. Also, this approach needs $O(n^2)$ extra space which may become huge for large input arrays.

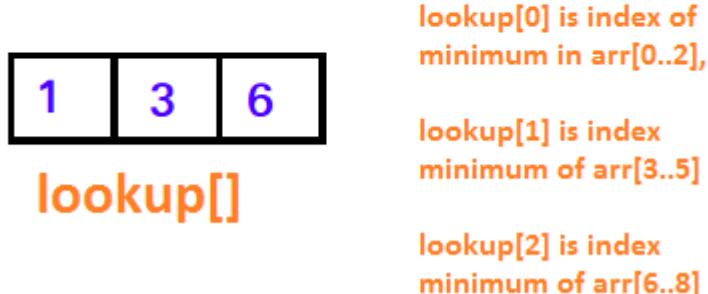
Method 2 (Square Root Decomposition)

We can use Square Root Decompositions to reduce space required in above method.

Preprocessing:

- 1) Divide the range $[0, n-1]$ into different blocks of \sqrt{n} each.
- 2) Compute minimum of every block of size \sqrt{n} and store the results.

Preprocessing takes $O(\sqrt{n} * \sqrt{n}) = O(n)$ time and $O(\sqrt{n})$ space.



Query:

- 1) To query a range $[L, R]$, we take minimum of all blocks that lie in this range. For left and right corner blocks which may partially overlap with given range, we linearly scan them to find minimum.

Time complexity of query is $O(\sqrt{n})$. Note that we have minimum of middle block directly accessible and there can be at most $O(\sqrt{n})$ middle blocks. There can be atmost two corner blocks that we may have to scan, so we may have to scan $2*O(\sqrt{n})$ elements of corner blocks. Therefore, overall time complexity is $O(\sqrt{n})$.

Refer [Sqrt \(or Square Root\) Decomposition Technique | Set 1 \(Introduction\)](#) for details.

Method 3 (Sparse Table Algorithm)

The above solution requires only $O(\sqrt{n})$ space, but takes $O(\sqrt{n})$ time to query. Sparse table method supports query time **$O(1)$** with extra space **$O(n \log n)$** .

The idea is to precompute minimum of all subarrays of size 2^j where j varies from 0 to **Log n**. Like method 1, we make a lookup table. Here $\text{lookup}[i][j]$ contains minimum of range starting from i and of size 2^j . For example $\text{lookup}[0][3]$ contains minimum of range $[0, 7]$ (starting with 0 and of size 2^3)

Preprocessing:

How to fill this lookup table? The idea is simple, fill in bottom up manner using previously computed values.

For example, to find minimum of range $[0, 7]$, we can use minimum of following two.

- a) Minimum of range $[0, 3]$
- b) Minimum of range $[4, 7]$

Based on above example, below is formula,

```
// If arr[lookup[0][2]] <= arr[lookup[4][2]],  
// then lookup[0][3] = lookup[0][2]  
If arr[lookup[i][j-1]] <= arr[lookup[i+2j-1-1][j-1]]  
    lookup[i][j] = lookup[i][j-1]  
  
// If arr[lookup[0][2]] > arr[lookup[4][2]],  
// then lookup[0][3] = lookup[4][2]  
Else  
    lookup[i][j] = lookup[i+2j-1-1][j-1]
```



0 1 3 3	lookup[i][j] contains index of minimum in range from arr[i] to arr[i + 2^j - 1]
1 1 3 3	
2 3 3 _	
3 3 3 _	
4 4 6 _	
5 6 6 _	
6 6 _ _	
7 7 _ _	
8 _ _ _	

lookup[][]

Query:

For any arbitrary range $[l, R]$, we need to use ranges which are in powers of 2. The idea is to use closest power of 2. We always need to do at most one comparison (compare minimum of two ranges which are powers of 2). One range starts with L and ends with “L + closest-power-of-2”. The other range ends at R and starts with “R – same-closest-power-of-2 + 1”. For example, if given range is (2, 10), we compare minimum of two ranges (2, 9) and (3, 10).

Based on above example, below is formula,

```
// For (2,10), j = floor(Log2(10-2+1)) = 3
j = floor(Log(R-L+1))

// If arr[lookup[0][3]] <= arr[lookup[3][3]],
// then RMQ(2,10) = lookup[0][3]
If arr[lookup[L][j]] <= arr[lookup[R-(int)pow(2,j)+1][j]]
    RMQ(L, R) = lookup[L][j]

// If arr[lookup[0][3]] > arr[lookup[3][3]],
// then RMQ(2,10) = lookup[3][3]
Else
```

```
RMQ(L, R) = lookup[R-(int)pow(2,j)+1][j]
```

Since we do only one comparison, time complexity of query is O(1).

Below is C++ implementation of above idea.

```
// C++ program to do range minimum query in O(1) time with
// O(n Log n) extra space and O(n Log n) preprocessing time
#include<bits/stdc++.h>
using namespace std;
#define MAX 500

// lookup[i][j] is going to store index of minimum value in
// arr[i..j]. Ideally lookup table size should not be fixed and
// should be determined using n Log n. It is kept constant to
// keep code simple.
int lookup[MAX][MAX];

// Structure to represent a query range
struct Query
{
    int L, R;
};

// Fills lookup array lookup[][] in bottom up manner.
void preprocess(int arr[], int n)
{
    // Initialize M for the intervals with length 1
    for (int i = 0; i < n; i++)
        lookup[i][0] = i;

    // Compute values from smaller to bigger intervals
    for (int j=1; (1<<j)<=n; j++)
    {
        // Compute minimum value for all intervals with size 2^j
        for (int i=0; (i+(1<<j)-1) < n; i++)
        {
            // For arr[2][10], we compare arr[lookup[0][3]] and
            // arr[lookup[3][3]]
            if (arr[lookup[i][j-1]] < arr[lookup[i + (1<<(j-1))][j-1]])
                lookup[i][j] = lookup[i][j-1];
            else
                lookup[i][j] = lookup[i + (1 << (j-1))][j-1];
        }
    }
}

// Returns minimum of arr[L..R]
```

```

int query(int arr[], int L, int R)
{
    // For [2,10], j = 3
    int j = (int)log2(R-L+1);

    // For [2,10], we compare arr[lookup[0][3]] and
    // arr[lookup[3][3]],
    if (arr[lookup[L][j]] <= arr[lookup[R - (1<<j) + 1][j]])
        return arr[lookup[L][j]];

    else return arr[lookup[R - (1<<j) + 1][j]];
}

// Prints minimum of given m query ranges in arr[0..n-1]
void RMQ(int arr[], int n, Query q[], int m)
{
    // Fills table lookup[n][Log n]
    preprocess(arr, n);

    // One by one compute sum of all queries
    for (int i=0; i<m; i++)
    {
        // Left and right boundaries of current range
        int L = q[i].L, R = q[i].R;

        // Print sum of current query range
        cout << "Minimum of [" << L << ", "
              << R << "] is " << query(arr, L, R) << endl;
    }
}

// Driver program
int main()
{
    int a[] = {7, 2, 3, 0, 5, 10, 3, 12, 18};
    int n = sizeof(a)/sizeof(a[0]);
    Query q[] = {{0, 4}, {4, 7}, {7, 8}};
    int m = sizeof(q)/sizeof(q[0]);
    RMQ(a, n, q, m);
    return 0;
}

```

Output:

```

Minimum of [0, 4] is 0
Minimum of [4, 7] is 3
Minimum of [7, 8] is 12

```

So sparse table method supports query operation in $O(1)$ time with $O(n \log n)$ preprocessing time and $O(n \log n)$ space.

This article is contributed by **Ruchir Garg**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [costom](#)

Source

<https://www.geeksforgeeks.org/range-minimum-query-for-static-array/>

Chapter 140

Remove the forbidden strings

Remove the forbidden strings - GeeksforGeeks

Given a string **W**, change the string in such a way that it does not contain any of the “forbidden” strings **S1** to **Sn** as one of its substrings. The rules that govern this change are as follows:

1. Case of the characters does not matter i.e “XyZ” is same as “xYZ”.
2. Change all the characters that are covered by the substrings in the original string **W** such that a particular letter **It** occurs maximum number of times in the changed string and the changed string is lexicographically the smallest string that can be obtained from all possible combinations.
3. Characters that are not within the forbidden substrings are not allowed to change.
4. The case of the changed character must be the same as the case of the original character at that position in string **W**.
5. If the letter **It** is part of the substring then it must be changed to some other character according to above rules.

Examples:

```
Input : n = 3
        s1 = "etr"
        s2 = "ed"
        s3 = "ied"
        W = "PEtrUnited"
        letter = "d"
Output : PDddUnitda
```

```
Input : n = 1
        s1 = "PetrDreamOh"
        W = "PetrDreamOh"
        letter = h
Output : HhhhhHhhhhHa
```

Explanation:

Example 1: First character P does not belong to any of the substrings therefore it is not changed. The next three characters form the substring “etr” and are changed to “Ddd”. The next four characters do not belong to any of the forbidden substrings and remain unchanged. The next two characters form the substring “ed” and are changed to “da” since d is The last character itself, it is replaced with another character ‘a’ such that the string is lexicographically the smallest.

Notice: “Etr” = “etr” and the changed substring “Ddd” has first character as ‘D’ since first letter of “Etr” is in uppercase.

Example 2: Since the entire string is a forbidden string, it is replaced with letter ‘h’ from first to second last character according to the case. The last character is ‘h’ therefore it is replaced with letter ‘a’ such that the string is lexicographically the smallest.

Approach:

Check for every character of the string W, if it is the beginning of the any of the substrings or not. Use another array to keep record of the characters of W that are part of the forbidden strings and needs to be changed.

The characters are changed according to the following rules:

1. If $W[i] = \text{letter}$ and $\text{letter} = 'a'$ then $W[i] = 'b'$
2. If $W[i] = \text{letter}$ and $\text{letter} \neq 'a'$ then $W[i] = 'a'$
3. If $W[i] \neq \text{letter}$ then $W[i] = \text{letter}$

The first and second condition will also be used when $W[i]$ is an uppercase character.

Below is the implementation of above approach:

C++

```
// CPP program to remove the forbidden strings
#include <bits/stdc++.h>
using namespace std;

// pre[] keeps record of the characters
// of w that need to be changed
bool pre[100];

// number of forbidden strings
int n;

// given string
string w;

// stores the forbidden strings
string s[110];

// letter to replace and occur
// max number of times
char letter;
```

```
// Function to check if the particula
// r substring is present in w at position
void verify(int position, int index)
{
    int l = w.length();
    int k = s[index].length();

    // If length of substring from this
    // position is greater than length
    // of w then return
    if (position + k > l)
        return;

    bool same = true;
    for (int i = position; i < position + k;
         i++) {

        // n and n1 are used to check for
        // substring without considering
        // the case of the letters in w
        // by comparing the difference
        // of ASCII values
        int n, n1;
        char ch = w[i];
        char ch1 = s[index][i - position];
        if (ch >= 'a' && ch <= 'z')
            n = ch - 'a';
        else
            n = ch - 'A';
        if (ch1 >= 'a' && ch1 <= 'z')
            n1 = ch1 - 'a';
        else
            n1 = ch1 - 'A';
        if (n != n1)
            same = false;
    }

    // If same == true then it means a
    // substring was found starting at
    // position therefore all characters
    // from position to length of substring
    // found need to be changed therfore
    // they needs to be marked
    if (same == true) {
        for (int i = position; i < position + k;
             i++)
            pre[i] = true;
    }
}
```

```
        return;
    }
}

// Function implementing logic
void solve()
{
    int l = w.length();
    int p = letter - 'a';

    for (int i = 0; i < 100; i++)
        pre[i] = false;

    // To verify if any substring is
    // starting from index i
    for (int i = 0; i < l; i++) {
        for (int j = 0; j < n; j++)
            verify(i, j);
    }

    // Modifying the string w
    // according to th rules
    for (int i = 0; i < l; i++) {
        if (pre[i] == true) {
            if (w[i] == letter)
                w[i] = (letter == 'a') ? 'b' : 'a';

            // This condition checks
            // if w[i]=upper(letter)
            else if (w[i] == 'A' + p)
                w[i] = (letter == 'a') ? 'B' : 'A';

            // This condition checks if w[i]
            // is any lowercase letter apart
            // from letter. If true replace
            // it with letter
            else if (w[i] >= 'a' && w[i] <= 'z')
                w[i] = letter;

            // This condition checks if w[i]
            // is any uppercase letter apart
            // from letter. If true then
            // replace it with upper(letter).
            else if (w[i] >= 'A' && w[i] <= 'Z')
                w[i] = 'A' + p;
        }
    }
    cout << w;
```

```
}

// Driver function for the program
int main()
{
    n = 3;
    s[0] = "etr";
    s[1] = "ed";
    s[2] = "ied";
    w = "PEtrUnited";
    letter = 'd';

    // Calling function
    solve();
    return 0;
}
```

Java

```
// Java program to remove forbidden strings
import java.io.*;

class rtf {

    // number of forbidden strings
    public static int n;

    // original string
    public static String z;

    // forbidden strings
    public static String s[] = new String[100];

    // to store original string
    // as character array
    public static char w[];

    // letter to replace and occur
    // max number of times
    public static char letter;

    // pre[] keeps record of the characters
    // of w that need to be changed
    public static boolean pre[] = new boolean[100];

    // Function to check if the particular
    // substring is present in w at position
    public static void verify(int position, int index)
```

```
{  
    int l = z.length();  
    int k = s[index].length();  
  
    // If length of substring from this  
    // position is greater than length  
    // of w then return  
    if (position + k > l)  
        return;  
  
    boolean same = true;  
    for (int i = position; i < position + k; i++) {  
  
        // n and n1 are used to check for  
        // substring without considering  
        // the case of the letters in w  
        // by comparing the difference  
        // of ASCII values  
        int n, n1;  
        char ch = w[i];  
        char ch1 = s[index].charAt(i - position);  
        if (ch >= 'a' && ch <= 'z')  
            n = ch - 'a';  
        else  
            n = ch - 'A';  
        if (ch1 >= 'a' && ch1 <= 'z')  
            n1 = ch1 - 'a';  
        else  
            n1 = ch1 - 'A';  
        if (n != n1)  
            same = false;  
    }  
  
    // If same == true then it means a substring  
    // was found starting at position therefore  
    // all characters from position to length  
    // of substring found need to be changed  
    // therfore they need to be marked  
    if (same == true) {  
        for (int i = position; i < position + k;  
             i++)  
            pre[i] = true;  
        return;  
    }  
}  
  
// Function performing calculations.  
public static void solve()
```

```
{  
    w = z.toCharArray();  
    letter = 'd';  
    int l = z.length();  
    int p = letter - 'a';  
    for (int i = 0; i < 100; i++)  
        pre[i] = false;  
  
    // To verify if any substring is  
    // starting from index i  
    for (int i = 0; i < l; i++) {  
        for (int j = 0; j < n; j++)  
            verify(i, j);  
    }  
  
    // Modifying the string w  
    // according to th rules  
    for (int i = 0; i < l; i++) {  
        if (pre[i] == true) {  
            if (w[i] == letter)  
                w[i] = (letter == 'a') ? 'b' : 'a';  
  
            // This conditon checks  
            // if w[i]=upper(letter)  
            else if (w[i] == (char)((int)'A' + p))  
                w[i] = (letter == 'a') ? 'B' : 'A';  
  
            // This conditon checks if w[i]  
            // is any lowercase letter apart  
            // from letter. If true replace  
            // it with letter  
            else if (w[i] >= 'a' && w[i] <= 'z')  
                w[i] = letter;  
  
            // This condition checks if w[i]  
            // is any uppercase letter apart  
            // from letter. If true then  
            // replace it with upper(letter).  
            else if (w[i] >= 'A' && w[i] <= 'Z')  
                w[i] = (char)((int)'A' + p);  
        }  
    }  
    System.out.println(w);  
}  
  
// Driver function for the program  
public static void main(String args[])  
{
```

```
n = 3;
s[0] = "etr";
s[1] = "ed";
s[2] = "ied";
z = "PEtrUnited";
solve();
}
}
```

Output:

PDddUnitda

Source

<https://www.geeksforgeeks.org/remove-forbidden-strings/>

Chapter 141

Represent a number as sum of minimum possible psuedobinary numbers

Represent a number as sum of minimum possible psuedobinary numbers - GeeksforGeeks

Given a number, you have to represent this number as sum of minimum number of possible *psuedobinary* numbers. A number is said to be *psuedobinary* number if its decimal number consists of only two digits (0 and 1). Example: 11,10,101 are all psuedobinary numbers.

Examples :-

Input : 44

Output : 11 11 11 11

Explanation : 44 can be represented as sum of minimum 4 psuedobinary numbers as 11+11+11+11

Input : 31

Output : 11 10 10

Explanation : 31 can be represented as sum of minimum 3 psuedobinary numbers as 11+10+10

The idea to do this is to first observe carefully that we need to calculate minimum number of possible psuedobinary numbers. To do this we find a new number m such that if for a place in given number n, the digit is non-zero then the digit in that place in m is 1 otherwise zero. For example if n = 5102, then m will be 1101. Then we will print this number m and subtract m from n. We will keep repeating these steps until n is greater than zero.

C++

```
// C++ program to represent a given
// number as sum of minimum possible
// psuedobinary numbers
#include<iostream>
using namespace std;

// function to represent a given
// number as sum of minimum possible
// psuedobinary numbers
void psuedoBinary(int n)
{
    // Repeat below steps until n > 0
    while (n > 0)
    {
        // calculate m (A number that has same
        // number of digits as n, but has 1 in
        // place of non-zero digits 0 in place
        // of 0 digits)
        int temp = n, m = 0, p = 1;
        while (temp)
        {
            int rem = temp % 10;
            temp = temp / 10;

            if (rem != 0)
                m += p;

            p *= 10;
        }
        cout << m << " ";

        // subtract m from n
        n = n - m;
    }
}

// Driver code
int main()
{
    int n = 31;

    psuedoBinary(n);

    return 0;
}
```

Java

```
// Java program to represent a given
// number as sum of minimum possible
// psuedobinary numbers

import java.util.*;
import java.lang.*;

class GFG
{
    public static void psuedoBinary(int n)
    {
        // Repeat below steps until n > 0
        while (n != 0)
        {
            // calculate m (A number that has same
            // number of digits as n, but has 1 in
            // place of non-zero digits 0 in place
            // of 0 digits)
            int temp = n, m = 0, p = 1;
            while(temp != 0)
            {
                int rem = temp % 10;
                temp = temp / 10;

                if (rem != 0)
                    m += p;

                p *= 10;
            }

            System.out.print(m + " ");

            // subtract m from n
            n = n - m;
        }
        System.out.println(" ");
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 31;
        psuedoBinary(n);
    }
}

// This code is contributed by Mohit Gupta_OMG
```

Python3

```
# Python3 program to represent
# a given number as sum of
# minimum possible psuedobinary
# numbers

# function to represent a
# given number as sum of
# minimum possible
# psuedobinary numbers
def psuedoBinary(n):

    # Repeat below steps
    # until n > 0
    while (n > 0):

        # calculate m (A number
        # that has same number
        # of digits as n, but
        # has 1 in place of non-zero
        # digits 0 in place of 0 digits)
        temp = n;
        m = 0;
        p = 1;
        while (temp):
            rem = temp % 10;
            temp = int(temp / 10);

            if (rem != 0):
                m += p;
            p *= 10;

        print(m,end=" ");

        # subtract m from n
        n = n - m;

    # Driver code
    n = 31;
    psuedoBinary(n);

# This code is contributed
# by mits.
```

C#

```
// C# program to represent a given
```

```
// number as sum of minimum possible
// psuedobinary numbers

using System;

class GFG
{
    public static void psuedoBinary(int n)
    {
        // Repeat below steps until n > 0
        while (n != 0)
        {
            // calculate m (A number that has same
            // number of digits as n, but has 1 in
            // place of non-zero digits 0 in place
            // of 0 digits)
            int temp = n, m = 0, p = 1;
            while(temp != 0)
            {
                int rem = temp % 10;
                temp = temp / 10;

                if (rem != 0)
                    m += p;

                p *= 10;
            }

            Console.WriteLine(m + " ");

            // subtract m from n
            n = n - m;
        }
        Console.WriteLine(" ");
    }

    // Driver code
    public static void Main()
    {
        int n = 31;
        psuedoBinary(n);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
<?php
// PHP program to represent a
// given number as sum of minimum
// possible psuedobinary numbers

// Function to represent a
// given number as sum of minimum
// possible psuedobinary numbers
function psuedoBinary($n)
{
    // Repeat below steps until n > 0
    while ($n > 0)
    {
        // calculate m (A number
        // that has same number of
        // digits as n, but has 1
        // in place of non-zero
        // digits 0 in place of 0
        // digits)
        $temp = $n; $m = 0; $p = 1;
        while ($temp)
        {
            $rem = $temp % 10;
            $temp = $temp / 10;

            if ($rem != 0)
                $m += $p;

            $p *= 10;
        }

        echo $m , " ";

        // subtract m from n
        $n = $n - $m;
    }
}

// Driver code
$n = 31;
psuedoBinary($n);

// This code is contributed
// by nitin mittal.
?>
```

Output :

11 10 10

Time Complexity : $O(\log n)$

Auxiliary Space : $O(1)$

Improved By : [nitin mittal](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/represent-number-sum-minimum-possible-psuedobinary-numbers/>

Chapter 142

Searching in a map using std::map functions in C++

Searching in a map using std::map functions in C++ - GeeksforGeeks

Usually, main purpose of using [map](#) stl container is for **efficient search operations and sorted order retrieval**. As map stores key-value pair, all the search operations take “ $O(\log(n))$ ” time (n is size of map). Different types of search functions exists in C++ language, each having different functions. In the context of competitive programming, this turns out to be useful when search operations are required and performs better than other containers. Some search operations are discussed below.

std::map::find()

find() is used to **search for the key-value** pair and accepts the “key” in its argument to find it. This function returns the pointer to the element if the element is found, else it returns the pointer pointing to the last position of map i.e “[map.end\(\)](#)” .

```
// C++ code to demonstrate the working of find()

#include<iostream>
#include<map> // for map operations
using namespace std;

int main()
{
    // declaring map
    // of char and int
    map<char, int > mp;

    // declaring iterators
    map<char, int>::iterator it ;
    map<char, int>::iterator it1 ;
```

```
// inserting values
mp['a']=5;
mp['b']=10;
mp['c']=15;
mp['d']=20;
mp['e']=30;

// using find() to search for 'b'
// key found
// "it" has address of key value pair.
it = mp.find('b');

if(it == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair present : "
        << it->first << "->" << it->second ;

cout << endl ;

// using find() to search for 'm'
// key not found
// "it1" has address of end of map.
it1 = mp.find('m');

if(it1 == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair present : "
        << it1->first << "->" << it1->second ;

}
```

Output:

```
Key-value pair present : b->10
Key-value pair not present in map
```

std::map::lower_bound()

`lower_bound()` is also used for the search operation but sometimes also returns a valid key-value pair even if it is not present in map . `lower_bound()` returns **address of key value pair**, if one is present in map, else returns the address to the smallest key greater than key mentioned in its arguments. If all keys are smaller than the key to be found, it points to “`map.end()`” .

```
// C++ code to demonstrate the working of lower_bound()
```

```
#include<iostream>
#include<map> // for map operations
using namespace std;

int main()
{
    // declaring map
    // of char and int
    map< char, int > mp;

    // declaring iterators
    map<char, int>::iterator it ;
    map<char, int>::iterator it1 ;
    map<char, int>::iterator it2 ;

    // inserting values
    mp['a']=5;
    mp['b']=10;
    mp['c']=15;
    mp['h']=20;
    mp['k']=30;

    // using lower_bound() to search for 'b'
    // key found
    // "it" has address of key value pair.
    it = mp.lower_bound('b');

    if(it == mp.end())
        cout << "Key-value pair not present in map" ;
    else
        cout << "Key-value pair returned : "
            << it->first << "->" << it->second ;

    cout << endl ;

    // using lower_bound() to search for 'd'
    // key not found
    // "it1" has address of next greater key.
    // key - 'h'
    it1 = mp.lower_bound('d');

    if(it1 == mp.end())
        cout << "Key-value pair not present in map" ;
    else
        cout << "Key-value pair returned : "
            << it1->first << "->" << it1->second ;
```

```
cout << endl;

// using lower_bound() to search for 'p'
// key not found
// "it2" has address of next greater key.
// all keys are smaller, hence returns mp.end()
it2 = mp.lower_bound('p');

if(it2 == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair returned : "
        << it2->first << "->" << it2->second ;

}
```

Output:

```
Key-value pair returned : b->10
Key-value pair returned : h->20
Key-value pair not present in map
```

std::map::upper_bound()

upper_bound() is also used for the search operation and **never returns the key-value pair searched for**. upper_bound() returns **address of key value pair coming exactly next to the searched key, if one is present in map**. If all keys are smaller than the key to be found, it points to “map.end()”

```
// C++ code to demonstrate the working of upper_bound()

#include<iostream>
#include<map> // for map operations
using namespace std;

int main()
{
    // declaring map
    // of char and int
    map< char, int > mp;

    // declaring iterators
    map<char, int>::iterator it ;
    map<char, int>::iterator it1 ;
    map<char, int>::iterator it2 ;
```

```
// inserting values
mp['a']=5;
mp['b']=10;
mp['c']=15;
mp['h']=20;
mp['k']=30;

// using upper_bound() to search for 'b'
// key found
// "it" has address of key value pair next to 'b' i.e 'c'.
it = mp.upper_bound('b');

if(it == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair returned : "
        << it->first << "->" << it->second ;

cout << endl ;

// using upper_bound() to search for 'd'
// key not found
// "it1" has address of next greater key.
// key - 'h'
it1 = mp.upper_bound('d');

if(it1 == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair returned : "
        << it1->first << "->" << it1->second ;

cout << endl;

// using upper_bound() to search for 'p'
// key not found
// "it2" has address of next greater key.
// all keys are smaller, hence returns mp.end()
it2 = mp.upper_bound('p');

if(it2 == mp.end())
    cout << "Key-value pair not present in map" ;
else
    cout << "Key-value pair returned : "
        << it2->first << "->" << it2->second ;

}
```

Output:

```
Key-value pair returned : c->15
Key-value pair returned : h->20
Key-value pair not present in map
```

std::map::equal_range

Yet another function to search in map, it **returns the range containing the searched key**. As map contains unique elements, range returned contains at most 1 element. This function **returns a iterator of pair**, whose 1st element points to `lower_bound()` of the searched key pair, and second element points to the `upper_bound()` of the searched key. If key is not present, both first element and second element point to the next greater element.

```
// C++ code to demonstrate the working of equal_range()

#include<iostream>
#include<map> // for map operations
using namespace std;

int main()
{
    // declaring map
    // of char and int
    map< char, int > mp;

    // declaring iterators
    pair<map<char, int>::iterator, map<char, int>::iterator> it;

    // inserting values
    mp['a']=5;
    mp['b']=10;
    mp['c']=15;
    mp['h']=20;
    mp['k']=30;

    // using equal_range() to search for 'b'
    // key found
    // 1st element of "it" has the address to lower_bound (b)
    // 2nd element of "it" has the address to upper_bound (c)
    it = mp.equal_range('b');

    cout << "The lower_bound of key is : "
        << it.first -> first << "->" << it.first -> second;
    cout << endl;
```

```
cout << "The upper_bound of key is : "
      << it.second -> first << "->" << it.second -> second;

cout << endl << endl ;

// using equal_range() to search for 'd'
// key not found
// Both elements of it point to next greater key
// key - 'h'
it = mp.equal_range('d');

cout << "The lower_bound of key is : "
      << it.first -> first << "->" << it.first -> second;
cout << endl;

cout << "The upper_bound of key is : "
      << it.second -> first << "->" << it.second -> second;

}

Output:
```

```
The lower_bound of key is : b->10
The upper_bound of key is : c->15
```

```
The lower_bound of key is : h->20
The upper_bound of key is : h->20
```

Source

<https://www.geeksforgeeks.org/searching-map-using-stdmap-functions-c/>

Chapter 143

Sequence Alignment problem

Sequence Alignment problem - GeeksforGeeks

Given as an input two strings, $X = x_1x_2 \dots x_n$, and $Y = y_1y_2 \dots y_m$, output the alignment of the strings, character by character, so that the net penalty is **minimised**. The penalty is calculated as:

1. A penalty of $\frac{P_{gap}}{gap}$ occurs if a gap is inserted between the string.
2. A penalty of $\frac{P_{mis}}{mis}$ occurs for mis-matching the characters of X and Y .

Examples:

Input : X = CG, Y = CA, p_gap = 3, p_xy = 7

Output : X = CG_, Y = C_A, Total penalty = 6

Input : X = AGGGCT, Y = AGGCA, p_gap = 3, p_xy = 2

Output : X = AGGGCT, Y = A_GGCA, Total penalty = 5

Input : X = CG, Y = CA, p_gap = 3, p_xy = 5

Output : X = CG, Y = CA, Total penalty = 5

A brief Note on the history of the problem

The Sequence Alignment problem is one of the fundamental problems of Biological Sciences, aimed at finding the similarity of two amino-acid sequences. Comparing amino-acids is of prime importance to humans, since it gives vital information on evolution and development. Saul B. Needleman and Christian D. Wunsch devised a dynamic programming algorithm to the problem and got it published in 1970. Since then, numerous improvements have been made to improve the time complexity and space complexity, however these are beyond the scope of discussion in this post.

Solution We can use dynamic programming to solve this problem. The feasible solution is to introduce gaps into the strings, so as to equalise the lengths. Since it can be easily proved

that the addition of extra gaps after equalising the lengths will only lead to increment of penalty.

Optimal Substructure

It can be observed from an optimal solution, for example from the given sample input, that the optimal solution narrows down to only **three** candidates.

1. $\text{A} \text{---} \text{B}$ and $\text{C} \text{---} \text{D}$.
2. $\text{A} \text{---} \text{B}$ and gap.
3. gap and $\text{C} \text{---} \text{D}$.

Proof of Optimal Substructure.

We can easily prove by contradiction. Let $\text{X}_1 \dots \text{X}_m$ be A^* and $\text{Y}_1 \dots \text{Y}_n$ be B^* . Suppose that the induced alignment of $\text{X}_1 \dots \text{X}_m$, $\text{Y}_1 \dots \text{Y}_n$ has some penalty P , and a competitor alignment has a penalty P' , with $P' < P$.

Now, appending $\text{A} \text{---} \text{B}$ and $\text{C} \text{---} \text{D}$, we get an alignment with penalty $P + P'$.

This contradicts the optimality of the original alignment of $\text{X}_1 \dots \text{X}_m$, $\text{Y}_1 \dots \text{Y}_n$. Hence, proved.

Let $\text{dp}[\text{m}][\text{n}]$ be the penalty of the optimal alignment of $\text{X}_1 \dots \text{X}_m$ and $\text{Y}_1 \dots \text{Y}_n$. Then, from the optimal substructure, $\text{dp}[\text{m}][\text{n}] = \min(\text{dp}[\text{m}-1][\text{n}] + 1, \text{dp}[\text{m}][\text{n}-1] + 1, \text{dp}[\text{m}-1][\text{n}-1])$.

The total minimum penalty is thus, $\text{dp}[\text{m}][\text{n}]$.

Reconstructing the solution

To Reconstruct,

1. Trace back through the filled table, starting $\text{dp}[\text{m}][\text{n}]$
2. When $\text{dp}[\text{m}][\text{n}]$
 -2a. if it was filled using case 1, go to $\text{dp}[\text{m}-1][\text{n}]$
 -2b. if it was filled using case 2, go to $\text{dp}[\text{m}][\text{n}-1]$
 -2c. if it was filled using case 3, go to $\text{dp}[\text{m}-1][\text{n}-1]$
3. if either $i = 0$ or $j = 0$, match the remaining substring with gaps.

Below is the implementation of the above solution.

C++

```
// CPP program to implement sequence alignment
// problem.
#include <bits/stdc++.h>
```

```
using namespace std;

// function to find out the minimum penalty
void getMinimumPenalty(string x, string y, int pxy, int pgap)
{
    int i, j; // initialising variables

    int m = x.length(); // length of gene1
    int n = y.length(); // length of gene2

    // table for storing optimal substructure answers
    int dp[n+m+1][n+m+1] = {0};

    // initialising the table
    for (i = 0; i <= (n+m); i++)
    {
        dp[i][0] = i * pgap;
        dp[0][i] = i * pgap;
    }

    // calculating the minimum penalty
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            if (x[i - 1] == y[j - 1])
            {
                dp[i][j] = dp[i - 1][j - 1];
            }
            else
            {
                dp[i][j] = min({dp[i - 1][j - 1] + pxy,
                                dp[i - 1][j] + pgap,
                                dp[i][j - 1] + pgap});
            }
        }
    }

    // Reconstructing the solution
    int l = n + m; // maximum possible length

    i = m; j = n;

    int xpos = l;
    int ypos = l;

    // Final answers for the respective strings
```

```

int xans[l+1], yans[l+1];

while ( !(i == 0 || j == 0))
{
    if (x[i - 1] == y[j - 1])
    {
        xans[xpos--] = (int)x[i - 1];
        yans[ypos--] = (int)y[j - 1];
        i--; j--;
    }
    else if (dp[i - 1][j - 1] + pxy == dp[i][j])
    {
        xans[xpos--] = (int)x[i - 1];
        yans[ypos--] = (int)y[j - 1];
        i--; j--;
    }
    else if (dp[i - 1][j] + pgap == dp[i][j])
    {
        xans[xpos--] = (int)x[i - 1];
        yans[ypos--] = (int)'_';
        i--;
    }
    else if (dp[i][j - 1] + pgap == dp[i][j])
    {
        xans[xpos--] = (int)'_';
        yans[ypos--] = (int)y[j - 1];
        j--;
    }
}
while (xpos > 0)
{
    if (i > 0) xans[xpos--] = (int)x[--i];
    else xans[xpos--] = (int)'_';
}
while (ypos > 0)
{
    if (j > 0) yans[ypos--] = (int)y[--j];
    else yans[ypos--] = (int)'_';
}

// Since we have assumed the answer to be n+m long,
// we need to remove the extra gaps in the starting
// id represents the index from which the arrays
// xans, yans are useful
int id = 1;
for (i = l; i >= 1; i--)
{
    if ((char)yans[i] == '_' && (char)xans[i] == '_')

```

```
{  
    id = i + 1;  
    break;  
}  
}  
  
// Printing the final answer  
cout << "Minimum Penalty in aligning the genes = "  
cout << dp[m][n] << "\n";  
cout << "The aligned genes are :\n";  
for (i = id; i <= l; i++)  
{  
    cout<<(char)xans[i];  
}  
cout << "\n";  
for (i = id; i <= l; i++)  
{  
    cout << (char)yans[i];  
}  
return;  
}  
  
// Driver code  
int main(){  
    // input strings  
    string gene1 = "AGGGCT";  
    string gene2 = "AGGCA";  
  
    // initialising penalties of different types  
    int misMatchPenalty = 3;  
    int gapPenalty = 2;  
  
    // calling the function to calculate the result  
    getMinimumPenalty(gene1, gene2,  
                      misMatchPenalty, gapPenalty);  
    return 0;  
}
```

Java

```
// Java program to implement  
// sequence alignment problem.  
import java.io.*;  
import java.util.*;  
import java.lang.*;  
  
class GFG  
{
```

```

// function to find out
// the minimum penalty
static void getMinimumPenalty(String x, String y,
                               int pxy, int pgap)
{
    int i, j; // initialising variables

    int m = x.length(); // length of gene1
    int n = y.length(); // length of gene2

    // table for storing optimal
    // substructure answers
    int dp[][] = new int[n + m + 1][n + m + 1];

    for (int[] x1 : dp)
        Arrays.fill(x1, 0);

    // initialising the table
    for (i = 0; i <= (n + m); i++)
    {
        dp[i][0] = i * pgap;
        dp[0][i] = i * pgap;
    }

    // calculating the
    // minimum penalty
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            if (x.charAt(i - 1) == y.charAt(j - 1))
            {
                dp[i][j] = dp[i - 1][j - 1];
            }
            else
            {
                dp[i][j] = Math.min(Math.min(dp[i - 1][j - 1] + pxy ,
                                              dp[i - 1][j] + pgap) ,
                                              dp[i][j - 1] + pgap );
            }
        }
    }

    // Reconstructing the solution
    int l = n + m; // maximum possible length

    i = m; j = n;
}

```

```

int xpos = 1;
int ypos = 1;

// Final answers for
// the respective strings
int xans[] = new int[l + 1];
int yans[] = new int[l + 1];

while ( !(i == 0 || j == 0))
{
    if (x.charAt(i - 1) == y.charAt(j - 1))
    {
        xans[xpos--] = (int)x.charAt(i - 1);
        yans[ypos--] = (int)y.charAt(j - 1);
        i--; j--;
    }
    else if (dp[i - 1][j - 1] + pxy == dp[i][j])
    {
        xans[xpos--] = (int)x.charAt(i - 1);
        yans[ypos--] = (int)y.charAt(j - 1);
        i--; j--;
    }
    else if (dp[i - 1][j] + pgap == dp[i][j])
    {
        xans[xpos--] = (int)x.charAt(i - 1);
        yans[ypos--] = (int)'_';
        i--;
    }
    else if (dp[i][j - 1] + pgap == dp[i][j])
    {
        xans[xpos--] = (int)'_';
        yans[ypos--] = (int)y.charAt(j - 1);
        j--;
    }
}
while (xpos > 0)
{
    if (i > 0) xans[xpos--] = (int)x.charAt(--i);
    else xans[xpos--] = (int)'_';
}
while (ypos > 0)
{
    if (j > 0) yans[ypos--] = (int)y.charAt(--j);
    else yans[ypos--] = (int)'_';
}

// Since we have assumed the
// answer to be n+m long,

```

```
// we need to remove the extra
// gaps in the starting id
// represents the index from
// which the arrays xans,
// yans are useful
int id = 1;
for (i = l; i >= 1; i--)
{
    if (((char)yans[i] == '_') &&
        (char)xans[i] == '_')
    {
        id = i + 1;
        break;
    }
}

// Printing the final answer
System.out.print("Minimum Penalty in " +
                  "aligning the genes = ");
System.out.print(dp[m][n] + "\n");
System.out.println("The aligned genes are :");
for (i = id; i <= l; i++)
{
    System.out.print((char)xans[i]);
}
System.out.print("\n");
for (i = id; i <= l; i++)
{
    System.out.print((char)yans[i]);
}
return;
}

// Driver code
public static void main(String[] args)
{
    // input strings
    String gene1 = "AGGGCT";
    String gene2 = "AGGCA";

    // initialising penalties
    // of different types
    int misMatchPenalty = 3;
    int gapPenalty = 2;

    // calling the function to
    // calculate the result
    getMinimumPenalty(gene1, gene2,
```

```

        misMatchPenalty, gapPenalty);
}
}

```

PHP

```

<?php
// PHP program to implement
// sequence alignment problem.

// function to find out
// the minimum penalty
function getMinimumPenalty($x, $y,
                           $pxy, $pgap)
{
    $i; $j; // initializing variables

    $m = strlen($x); // length of gene1
    $n = strlen($y); // length of gene2

    // table for storing optimal
    // substructure answers
    $dp[$n + $m + 1][$n + $m + 1] = array(0);

    // initialising the table
    for ($i = 0; $i <= ($n+$m); $i++)
    {
        $dp[$i][0] = $i * $pgap;
        $dp[0][$i] = $i * $pgap;
    }

    // calculating the
    // minimum penalty
    for ($i = 1; $i <= $m; $i++)
    {
        for ($j = 1; $j <= $n; $j++)
        {
            if ($x[$i - 1] == $y[$j - 1])
            {
                $dp[$i][$j] = $dp[$i - 1][$j - 1];
            }
            else
            {
                $dp[$i][$j] = min($dp[$i - 1][$j - 1] + $pxy ,
                                  $dp[$i - 1][$j] + $pgap ,
                                  $dp[$i][$j - 1] + $pgap );
            }
        }
    }
}

```

```

}

// Reconstructing the solution
$l = $n + $m; // maximum possible length

$i = $m; $j = $n;

$xpos = $l;
$ypos = $l;

// Final answers for
// the respective strings
// $xans[$l + 1]; $yans[$l + 1];

while ( !($i == 0 || $j == 0))
{
    if ($x[$i - 1] == $y[$j - 1])
    {
        $xans[$xpos--] = $x[$i - 1];
        $yans[$ypos--] = $y[$j - 1];
        $i--; $j--;
    }
    else if ($dp[$i - 1][$j - 1] +
              $pxy == $dp[$i][$j])
    {
        $xans[$xpos--] = $x[$i - 1];
        $yans[$ypos--] = $y[$j - 1];
        $i--; $j--;
    }
    else if ($dp[$i - 1][$j] +
              $pgap == $dp[$i][$j])
    {
        $xans[$xpos--] = $x[$i - 1];
        $yans[$ypos--] = '_';
        $i--;
    }
    else if ($dp[$i][$j - 1] +
              $pgap == $dp[$i][$j])
    {
        $xans[$xpos--] = '_';
        $yans[$ypos--] = $y[$j - 1];
        $j--;
    }
}
while ($xpos > 0)
{
    if ($i > 0) $xans[$xpos--] = $x[--$i];
    else $xans[$xpos--] = '_';
}

```

```

        }
        while ($ypos > 0)
        {
            if ($j > 0)
                $yans[$ypos--] = $y[--$j];
            else
                $yans[$ypos--] = '_';
        }

        // Since we have assumed the
        // answer to be n+m long,
        // we need to remove the extra
        // gaps in the starting
        // id represents the index
        // from which the arrays
        // xans, yans are useful
        $id = 1;
        for ($i = $l; $i >= 1; $i--)
        {
            if ($yans[$i] == '_' &&
                $xans[$i] == '_')
            {
                $id = $i + 1;
                break;
            }
        }

        // Printing the final answer
        echo "Minimum Penalty in ".
            "aligning the genes = ";
        echo $dp[$m][$n] . "\n";
        echo "The aligned genes are :\n";
        for ($i = $id; $i <= $l; $i++)
        {
            echo $xans[$i];
        }
        echo "\n";
        for ($i = $id; $i <= $l; $i++)
        {
            echo $yans[$i];
        }
        return;
    }

    // Driver code

    // input strings
    $gene1 = "AGGGCT";

```

```
$gene2 = "AGGCA";  
  
// initialising penalties  
// of different types  
$misMatchPenalty = 3;  
$gapPenalty = 2;  
  
// calling the function  
// to calculate the result  
getMinimumPenalty($gene1, $gene2,  
    $misMatchPenalty, $gapPenalty);  
  
// This code is contributed by Abhinav96  
?>
```

Output:

```
Minimum Penalty in aligning the genes = 5  
The aligned genes are :  
AGGGCT  
A_GGCA
```

Time Complexity : $\mathcal{O}(n^2)$

Space Complexity : $\mathcal{O}(n^2)$

Improved By : [AayushChaturvedi](#)

Source

<https://www.geeksforgeeks.org/sequence-alignment-problem/>

Chapter 144

Smallest number with sum of digits as N and divisible by 10^N

Smallest number with sum of digits as N and divisible by 10^N - GeeksforGeeks

Find the smallest number such that the sum of its digits is N and it is divisible by 10^N .

Examples :

Input : N = 5
Output : 500000
500000 is the smallest number divisible by 10^5 and sum of digits as 5.

Input : N = 20
Output : 29900000000000000000000000000000

Explanation

To make a number divisible by 10^N we need at least N zeros at the end of the number. To make the number smallest, we append exactly N zeros to the end of the number. Now, we need to ensure the sum of the digits is N. For this, we will try to make the length of the number as small as possible to get the answer. Thus we keep on inserting 9 into the number till the sum doesn't exceed N. If we have any remainder left, then we keep it as the first digit (most significant one) so that the resulting number is minimized.

The approach works well for all subtasks but there are 2 corner cases:

1. The first is that the final number may not fit into the data types present in C++/Java. Since we only need to output the number, we can use strings to store the answer.

2. The only corner case where the answer is 0 is $N = 0$.
3. There are no cases where the answer doesn't exist.

C++

```
// CPP program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by  $10^N$ .
#include <bits/stdc++.h>
using namespace std;

void digitsNum(int N)
{
    // If N = 0 the string will be 0
    if (N == 0)
        cout << "0\n";

    // If n is not perfectly divisible
    // by 9 output the remainder
    if (N % 9 != 0)
        cout << (N % 9);

    // Print 9 N/9 times
    for (int i = 1; i <= (N / 9); ++i)
        cout << "9";

    // Append N zero's to the number so
    // as to make it divisible by  $10^N$ 
    for (int i = 1; i <= N; ++i)
        cout << "0";

    cout << "\n";
}

// Driver Code
int main()
{
    int N = 5;
    cout << "The number is : ";
    digitsNum(N);
    return 0;
}
```

Java

```
// Java program to find smallest
// number to find smallest number
```

```
// with N as sum of digits and
// divisible by  $10^N$ .
import java.io.*;

class GFG
{

    static void digitsNum(int N)
    {
        // If N = 0 the string will be 0
        if (N == 0)
            System.out.println("0");

        // If n is not perfectly divisible
        // by 9 output the remainder
        if (N % 9 != 0)
            System.out.print((N % 9));

        // Print 9 N/9 times
        for (int i = 1; i <= (N / 9); ++i)
            System.out.print("9");

        // Append N zero's to the number so
        // as to make it divisible by  $10^N$ 
        for (int i = 1; i <= N; ++i)
            System.out.print("0");
        System.out.print(" ");

    }

    // Driver Code
    public static void main (String[] args)
    {
        int N = 5;
        System.out.print("The number is : ");
        digitsNum(N);
    }
}

// This code is contributed by vt_m
```

Python3

```
# Python program to find smallest
# number to find smallest number
```

```
# with N as sum of digits and
# divisible by  $10^N$ .

import math
def digitsNum(N):

    # If N = 0 the string will be 0
    if (N == 0) :
        print("0", end = "")

    # If n is not perfectly divisible
    # by 9 output the remainder
    if (N % 9 != 0):
        print (N % 9, end = "")

    # Print 9 N/9 times
    for i in range( 1, int(N / 9) + 1) :
        print("9", end = "")

    # Append N zero's to the number so
    # as to make it divisible by  $10^N$ 
    for i in range(1, N + 1) :
        print("0", end = "")

    print()

# Driver Code
N = 5
print("The number is : ",end="")
digitsNum(N)

# This code is contributed by Gitanjali.
```

C#

```
// C# program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by  $10^N$ .
using System;

class GFG
{

static void digitsNum(int N)
{
    // If N = 0 the string will be 0
```

```
if (N == 0)
Console.WriteLine("0");

// If n is not perfectly divisible
// by 9 output the remainder
if (N % 9 != 0)
    Console.WriteLine((N % 9));

// Print 9 N/9 times
for (int i = 1; i <= (N / 9); ++i)
    Console.WriteLine("9");

// Append N zero's to the number so
// as to make it divisible by  $10^N$ 
for (int i = 1; i <= N; ++i)
    Console.WriteLine("0");
    Console.WriteLine("");

}

// Driver Code
public static void Main ()
{
    int N = 5;
    Console.WriteLine("The number is : ");
    digitsNum(N);
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to find smallest
// number to find smallest number
// with N as sum of digits and
// divisible by  $10^N$ .

function digitsNum($N)
{
    // If N = 0 the string will be 0
    if ($N == 0)
        echo "0\n";
```

```
// If n is not perfectly divisible
// by 9 output the remainder
if ($N % 9 != 0)
    echo ($N % 9);

// Print 9 N/9 times
for ( $i = 1; $i <= ($N / 9); ++$i)
    echo "9";

// Append N zero's to the number so
// as to make it divisible by  $10^N$ 
for ($i = 1; $i <= $N; ++$i)
    echo "0";

echo "\n";
}

// Driver Code
$N = 5;
echo "The number is : ";
digitsNum($N);

// This code is contributed by ajit.
?>
```

Output :

```
The number is : 500000
```

Time Complexity : $O(N)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/smallest-number-sum-digits-n-divisible-10n/>

Chapter 145

Some important shortcuts in Competitive Programming

Some important shortcuts in Competitive Programming - GeeksforGeeks

The most common tools to save time in C/C++ are typedefs and macros. Unfortunately, these features are not available in many other languages like Java.

Here are some examples of our C/C++ code

Shortcuts:

```
// Shortcuts for "common" data types in contests
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
```

Infinite:

```
const int INF = 0x3f3f3f3f;
```

Setting up values in array:

```
// initialize DP memoization table with -1
memset(memo, -1, sizeof memo);

// to clear array of integers
memset(arr, 0, sizeof arr);
```

```
// Use a vector to creates a dynamic array  
// and initialize it in one statement.  
// Creates a vector of size N and values as 0.  
vector<int> v(N, 0); // OR vi v(N, 0);
```

The following shortcuts can save time in almost all languages including Java:

```
// to simplify: if (a) ans = b; else ans = c;  
ans = a ? b : c;  
  
// to simplify: ans = ans + val; and its variants  
ans += val;  
  
// index++; if (index >= n) index = 0;  
index = (index + 1) % n;  
  
// index--; if (index < 0) index = n - 1;  
index = (index + n - 1) % n;  
  
// for rounding to nearest integer  
int ans = (int)((double)d + 0.5);  
  
// min/max shortcut to update max/min  
ans = min(ans, new_computation);  
  
// To return true if a value matches a given  
// number, else return false  
// if (val == given_no) return true;  
// else return false;  
return (val == given_no)
```

This article is contributed by **Priyam kakati**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/important-shortcuts-competitive-programming/>

Chapter 146

Sqrt (or Square Root) Decomposition Technique | Set 1 (Introduction)

Sqrt (or Square Root) Decomposition Technique | Set 1 (Introduction) - GeeksforGeeks

Sqrt (or Square Root) Decomposition Technique is one of the most [common query optimization technique used by competitive programmers](#). This technique helps us to reduce Time Complexity by a factor of \sqrt{n} .

The key concept of this technique is to decompose given array into small chunks specifically of size \sqrt{n} .

Let's say we have an array of n elements and we decompose this array into small chunks of size \sqrt{n} . We will be having exactly \sqrt{n} such chunks provided that n is a perfect square. Therefore, now our array on n elements is decomposed into \sqrt{n} blocks, where each block contains \sqrt{n} elements (assuming size of array is perfect square).

Let's consider these chunks or blocks as an individual array each of which contains \sqrt{n} elements and you have computed your desired answer(according to your problem) individually for all the chunks. Now, you need to answer certain queries asking you the answer for the elements in range l to r (l and r are starting and ending indices of the array) in the original n sized array.

The **naive approach** is simply to iterate over each element in range l to r and calculate its corresponding answer. Therefore, the Time Complexity per query will be $O(n)$.

Sqrt Decomposition Trick : As we have already precomputed the answer for all individual chunks and now we need to answer the queries in range l to r . Now we can simply combine the answers of the chunks that lie in between the range l to r in the original array. So, if we see carefully here we are jumping \sqrt{n} steps at a time instead of jumping 1 step at a time as done in naive approach. Let's just analyze its Time Complexity and implementation considering the below problem :-

Problem :

Given an array of n elements. We need to answer q queries telling the sum of elements in range l to r in the array. Also the array is not static i.e the values are changed via some point update query.

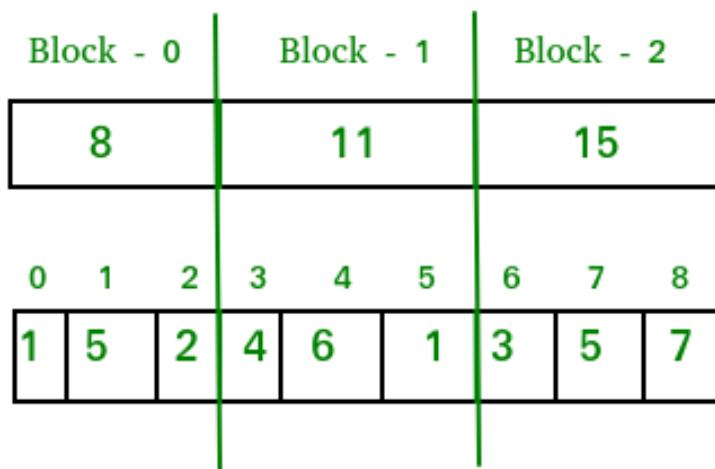
Range Sum Queries are of form : Q l r , where l is the starting index r is the ending index

Point update Query is of form : U idx val, where idx is the index to update val is the updated value

Let us consider that we have an array of 9 elements.

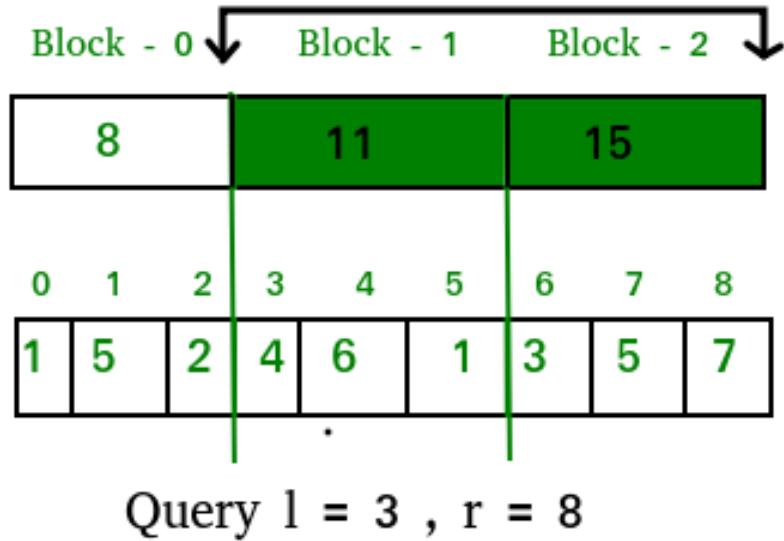
$A[] = \{1, 5, 2, 4, 6, 1, 3, 5, 7\}$

Let's decompose this array into $\sqrt{9}$ blocks, where each block will contain the sum of elements lying in it. Therefore now our decomposed array would look like this :



Till now we have constructed the decomposed array of $\sqrt{9}$ blocks and now we need to print the sum of elements in a given range. So first let's see two basic types of overlap that a range query can have on our array :-

Range Query of type 1 (Given Range is on Block Boundaries) :

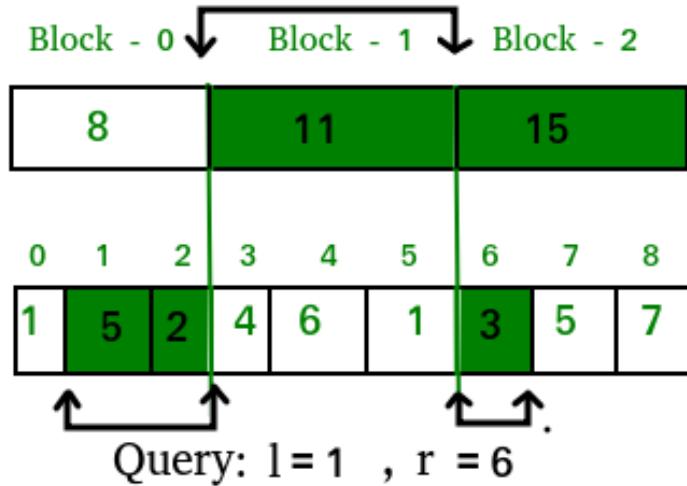


In this type the query, the range may totally cover the continuous sqrt blocks. So we can easily answer the sum of values in this range as the sum of completely overlapped blocks.

So answer for above query in the described image will be : ans = 11 + 15 = 26

Time Complexity: In the worst case our range can be 0 to n-1 (where n is the size of array and assuming n to be a perfect square). In this case all the blocks are completely overlapped by our query range. Therefore, to answer this query we need to iterate over all the decomposed blocks for the array and we know that the number of blocks = \sqrt{n} . Hence, the complexity for this type of query will be $O(\sqrt{n})$ in worst case.

Range Query of type 2 (Given Range is NOT on boundaries)



We can deal these type of queries by summing the data from the completely overlapped decomposed blocks lying in the query range and then summing elements one by one from the original array whose corresponding block is not completely overlapped by the query range.

So answer for above query in the described image will be : ans = 5 + 2 + 11 + 3 = 21

Time Complexity: Let's consider a query $[l = 1 \text{ and } r = n-2]$ (n is the size of the array and has a 0 based indexing). Therefore, for this query exactly $(\sqrt{n} - 2)$ blocks will be completely overlapped where as the first and last block will be partially overlapped with just one element left outside the overlapping range. So, the completely overlapped blocks can be summed up in $(\sqrt{n} - 2) \sim \sqrt{n}$ iterations, whereas first and last block are needed to be traversed one by one separately. But as we know that the number of elements in each block is at max \sqrt{n} , to sum up last block individually we need to make, $(\sqrt{n}-1) \sim \sqrt{n}$ iterations and same for the last block.

So, the overall Complexity = $O(\sqrt{n}) + O(\sqrt{n}) + O(\sqrt{n}) = O(3*\sqrt{N}) = O(\sqrt{n})$

Update Queries(Point update) :

In this query we simply find the block in which the given index lies, then subtract its previous value and add the new updated value as per the point update query.

Time Complexity : $O(1)$

Implementation :

The C++ implementation of the above trick is given below

```
// C++ program to demonstrate working of Square Root
// Decomposition.
```

```
#include "iostream"
#include "math.h"
using namespace std;

#define MAXN 10000
#define SQRSIZE 100

int arr[MAXN];           // original array
int block[SQRSIZE];     // decomposed array
int blk_sz;               // block size

// Time Complexity : O(1)
void update(int idx, int val)
{
    int blockNumber = idx / blk_sz;
    block[blockNumber] += val - arr[idx];
    arr[idx] = val;
}

// Time Complexity : O(sqrt(n))
int query(int l, int r)
{
    int sum = 0;
    while (l < r and l % blk_sz != 0 and l != 0)
    {
        // traversing first block in range
        sum += arr[l];
        l++;
    }
    while (l + blk_sz <= r)
    {
        // traversing completely overlapped blocks in range
        sum += block[l / blk_sz];
        l += blk_sz;
    }
    while (l <= r)
    {
        // traversing last block in range
        sum += arr[l];
        l++;
    }
    return sum;
}

// Fills values in input[]
void preprocess(int input[], int n)
{
    // initiating block pointer
```

```
int blk_idx = -1;

// calculating size of block
blk_sz = sqrt(n);

// building the decomposed array
for (int i=0; i<n; i++)
{
    arr[i] = input[i];
    if (i%blk_sz == 0)
    {
        // entering next block
        // incrementing block pointer
        blk_idx++;
    }
    block[blk_idx] += arr[i];
}

// Driver code
int main()
{
    // We have used separate array for input because
    // the purpose of this code is to explain SQRT
    // decomposition in competitive programming where
    // we have multiple inputs.
    int input[] = {1, 5, 2, 4, 6, 1, 3, 5, 7, 10};
    int n = sizeof(input)/sizeof(input[0]);

    preprocess(input, n);

    cout << "query(3,8) : " << query(3, 8) << endl;
    cout << "query(1,6) : " << query(1, 6) << endl;
    update(8, 0);
    cout << "query(8,8) : " << query(8, 8) << endl;
    return 0;
}
```

Output:

```
query(3,8) : 26
query(1,6) : 21
query(8,8) : 0
```

Note : The above code works even if n is not perfect square. In the case, the last block will contain even less number of elements than \sqrt{n} , thus reducing the number of iterations.

Let's say $n = 10$. In this case we will have 4 blocks first three block of size 3 and last block of size 1.

Source

<https://www.geeksforgeeks.org/sqrt-square-root-decomposition-technique-set-1-introduction/>

Chapter 147

Sqrt (or Square Root) Decomposition | Set 2 (LCA of Tree in O(sqrt(height)) time)

Sqrt (or Square Root) Decomposition | Set 2 (LCA of Tree in O(sqrt(height)) time) - Geeks-forGeeks

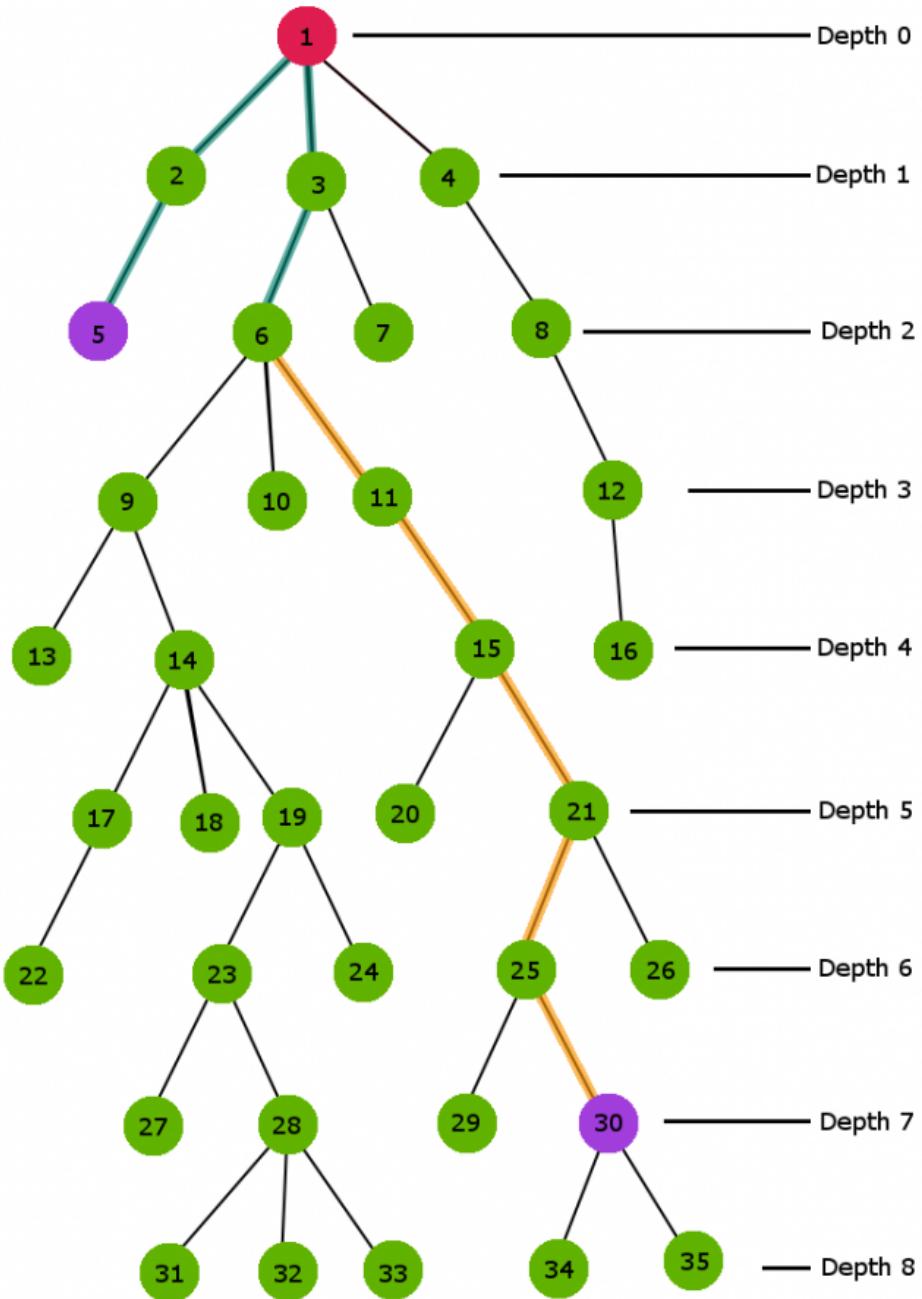
Prerequisite : [Introduction](#) and [DFS](#)

The task is to find LCA of two given nodes in a tree (not necessarily a Binary Tree). In previous posts, we have seen how to calculate [LCA using Sparse Matrix DP approach](#). In this post, we will see an optimization done on Naive method by sqrt decomposition technique that works well over the Naive Approach.

Naive Approach

To calculate the LCA of two nodes first of all we will bring both the nodes to same height by making the node with greater depth jump one parent up the tree till both the nodes are at same height. Once, both the nodes are at same height we can then start jumping one parent up for both the nodes simultaneously till both the nodes become equal and that node will be the LCA of the two originally given nodes.

Consider the below n-ary Tree with depth 9 and lets examine how naive approach works for this sample tree.



Here in the above Tree we need to calculate the LCA of node 6 and node 30

Clearly node 30 has greater depth than node 6. So first of all we start jumping one parent above for node 30 till we reach the depth value of node 6 i.e at depth 2.

The **orange colored path** in the above figure demonstrates the jumping sequence to reach the depth 2. In this procedure we just simply jump one parent above the current node.

Now both nodes are at same depth 2. Therefore, now both the nodes will jump one parent up till both the nodes become equal. This end node at which both the nodes become equal for the first time is our LCA.

The **blue color path** in the above figure shows the jumping route for both the nodes

C++ code for the above implementation:-

```

// Naive C++ implementation to find LCA in a tree
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
#define MAXN 1001

int depth[MAXN];           // stores depth for each node
int parent[MAXN];          // stores first parent for each node

vector < int > adj[MAXN];

void addEdge(int u,int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void dfs(int cur, int prev)
{
    // marking parent for each node
    parent[cur] = prev;

    // marking depth for each node
    depth[cur] = depth[prev] + 1;

    // propagating marking down the tree
    for (int i=0; i<adj[cur].size(); i++)
        if (adj[cur][i] != prev)
            dfs(adj[cur][i],cur);
}

void preprocess()
{
    // a dummy node
}

```

```

depth[0] = -1;

// precalclating 1)depth. 2)parent.
// for each node
dfs(1,0);
}

// Time Complexity : O(Height of tree)
// recursively jumps one node above
// till both the nodes become equal
int LCANaive(int u,int v)
{
    if (u == v)  return u;
    if (depth[u] > depth[v])
        swap(u, v);
    v = parent[v];
    return LCANaive(u,v);
}

// Driver function to call the above functions
int main(int argc, char const *argv[])
{
    // adding edges to the tree
    addEdge(1,2);
    addEdge(1,3);
    addEdge(1,4);
    addEdge(2,5);
    addEdge(2,6);
    addEdge(3,7);
    addEdge(4,8);
    addEdge(4,9);
    addEdge(9,10);
    addEdge(9,11);
    addEdge(7,12);
    addEdge(7,13);

    preprocess();

    cout << "LCA(11,8) : " << LCANaive(11,8) << endl;
    cout << "LCA(3,13) : " << LCANaive(3,13) << endl;

    return 0;
}

```

Output:

LCA(11,8) : 4

LCA(3,13) : 3

Time Complexity : We pre-calculate the depth for each node using one **DFS traversal in O(n)**. Now in worst case, the two nodes will be two bottom most node on the tree in different child branches of the root node. Therefore, in this case the root will be the LCA of both the nodes. Hence, both the nodes will have to jump exactly h height above, where h is the height of the tree. So, to answer each **LCA query Time Complexity will be $O(h)$** .

The Sqrt Decomposition Trick :

We categorize nodes of the tree into different groups according to their depth. Assuming the depth of the tree h is a perfect square. So once again like the [general sqrt decomposition approach](#) we will be having \sqrt{h} blocks or groups. Nodes from depth 0 to depth $\sqrt{h} - 1$ lie in first group; then nodes having depth \sqrt{h} to $2\sqrt{h} - 1$ lie in second group and so on till last node.

We keep track of the corresponding group number for every node and also depth of every node. This can be done by one single dfs on the tree (see the code for better understanding).

Sqrt trick :- In naive approach we were jumping one parent up the tree till both nodes aren't on the same depth. But here we perform group wise jump. To perform this group wise jump, we need two parameter associated with each node : 1) parent and 2) jump parent. Here **parent** for each node is defined as the first node above the current node that is directly connected to it, whereas **jump_parent** for each node is the node that is the first ancestor of the current node in the group just above the current node.

So, now we need to maintain 3 parameters for each node :

- 1) **depth**
- 2) **parent**
- 3) **jump_parent**

All these three parameters can be maintained in one dfs(refer to the code for better understanding)

Pseudo code for optimization process

```

LCAsqrt(u, v){

    // assuming v is at greater depth
    while (jump_parent[u] != jump_parent[v]){
        v = jump_parent[v];
    }

    // now both nodes are in same group
    // and have same jump_parent
    return LCAnaive(u,v);
}

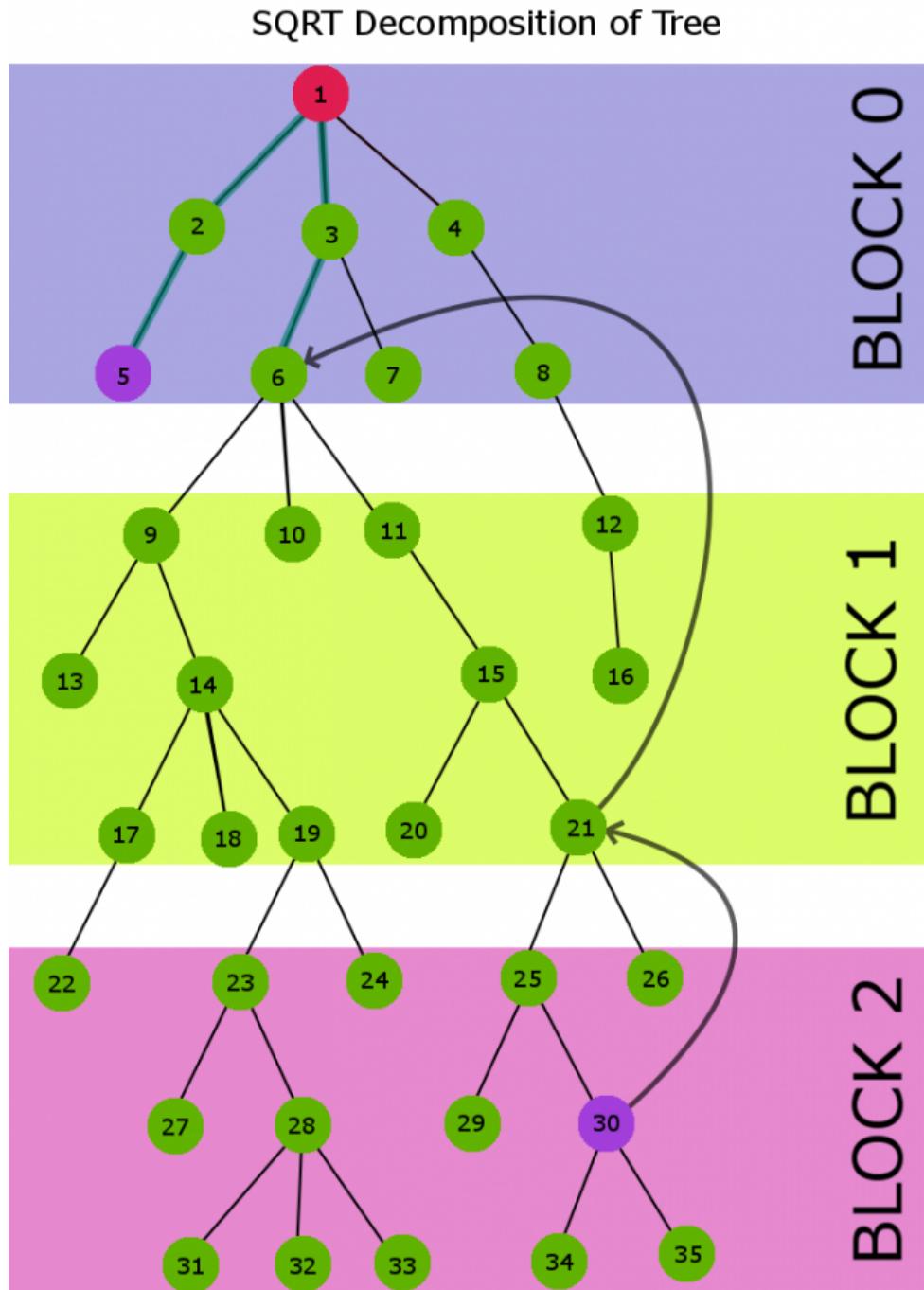
```

The key concept here is that first we bring both the nodes in same group and having same **jump_parent** by climbing decomposed blocks above the tree one by one and then when both

the nodes are in same group and have same jump_parent we use our naive approach to find LCA of the nodes.

This optimized group jumping technique reduces the iterating space by a factor of **sqrt(h)** and hence reduces the Time Complexity(refer below for better time complexity analysis)

Lets decompose the above tree in \sqrt{h} groups ($h = 9$) and calculate LCA for node 6 and 30.



In the above decomposed tree

Jump_parent[6] = 0	parent[6] = 3
Jump_parent[5] = 0	parent[5] = 2
Jump_parent[1] = 0	parent[1] = 0
Jump_parent[11] = 6	parent[11] = 6
Jump_parent[15] = 6	parent[15] = 11
Jump_parent[21] = 6	parent[21] = 15
Jump_parent[25] = 21	parent[25] = 21
Jump_parent[26] = 21	parent[26] = 21
Jump_parent[30] = 21	parent[30] = 25

Now at this stage Jump_parent for node 30 is 21 and Jump_parent for node 5 is 0, So we will climb to jump_parent[30] i.e to node 21

Now once again Jump_parent of node 21 is not equal to Jump_parent of node 5, So once again we will climb to jump_parent[21] i.e node 6

At this stage jump_parent[6] == jump_parent[5], So now we will use our naive climbing approach and climb one parent above for both the nodes till it reach node 1 and that will be the required LCA .

Blue path in the above figure describes jumping path sequence for node 6 and node 5.

The C++ code for the above description is given below:-

```
// C++ program to find LCA using Sqrt decomposition
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
#define MAXN 1001

int block_sz;           // block size = sqrt(height)
int depth[MAXN];       // stores depth for each node
int parent[MAXN];      // stores first parent for
                       // each node
int jump_parent[MAXN]; // stores first ancestor in
                       // previous block

vector < int > adj[MAXN];

void addEdge(int u,int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

```

int LCANaive(int u,int v)
{
    if (u == v)  return u;
    if (depth[u] > depth[v])
        swap(u,v);
    v = parent[v];
    return LCANaive(u,v);
}

// precalculating the required parameters
// associated with every node
void dfs(int cur, int prev)
{
    // marking depth of cur node
    depth[cur] = depth[prev] + 1;

    // marking parent of cur node
    parent[cur] = prev;

    // making jump_parent of cur node
    if (depth[cur] % block_sz == 0)

        /* if it is first node of the block
           then its jump_parent is its cur parent */
        jump_parent[cur] = parent[cur];

    else

        /* if it is not the first node of this block
           then its jump_parent is jump_parent of
           its parent */
        jump_parent[cur] = jump_parent[prev];

    // propogating the marking down the subtree
    for (int i = 0; i<adj[cur].size(); ++i)
        if (adj[cur][i] != prev)
            dfs(adj[cur][i], cur);
}

// using sqrt decomposition trick
int LCASQRT(int u, int v)
{
    while (jump_parent[u] != jump_parent[v])
    {
        if (depth[u] > depth[v])

```

```

// maintaining depth[v] > depth[u]
swap(u,v);

// climb to its jump parent
v = jump_parent[v];
}

// u and v have same jump_parent
return LCANaive(u,v);
}

void preprocess(int height)
{
    block_sz = sqrt(height);
    depth[0] = -1;

    // precalclating 1)depth. 2)parent. 3)jump_parent
    // for each node
    dfs(1, 0);
}

// Driver function to call the above functions
int main(int argc, char const *argv[])
{
    // adding edges to the tree
    addEdge(1,2);
    addEdge(1,3);
    addEdge(1,4);
    addEdge(2,5);
    addEdge(2,6);
    addEdge(3,7);
    addEdge(4,8);
    addEdge(4,9);
    addEdge(9,10);
    addEdge(9,11);
    addEdge(7,12);
    addEdge(7,13);

    // here we are directly taking height = 4
    // according to the given tree but we can
    // pre-calculate height = max depth
    // in one more dfs
    int height = 4;
    preprocess(height);

    cout << "LCA(11,8) : " << LCASQRT(11,8) << endl;
    cout << "LCA(3,13) : " << LCASQRT(3,13) << endl;
}

```

```
    return 0;
}
```

Output:

```
LCA(11,8) : 4
LCA(3,13) : 3
```

Note : The above code works even if height is not perfect square.

Now Lets see how the Time Complexity is changed by this simple grouping technique :

Time Complexity Analysis:

We have divided the tree into \sqrt{h} groups according to their depth and each group contain nodes having max difference in their depth equal to \sqrt{h} . Now once again take an example of worst case, let's say the first node 'u' is in first group and the node 'v' is in \sqrt{h} th group(last group). So, first we will make group jumps(single group jumps) till we reach group 1 from last group; This will take exactly $\sqrt{h} - 1$ iterations or jumps. So, till this step the Time Complexity is **$O(\sqrt{h})$** .

Now once we are in same group, we call the LCAnaive function. The Time complexity for LCA_Naive is $O(\sqrt{h})$, where h' is the height of the tree. Now, in our case value of h' will be \sqrt{h} , because each group has a subtree of at max \sqrt{h} height. So the complexity for this step is also $O(\sqrt{h})$.

Hence, the total Time Complexity will be **$O(\sqrt{h}) + \sqrt{h}) \sim O(\sqrt{h})$** .

Source

<https://www.geeksforgeeks.org/sqrt-square-root-decomposition-set-2-lca-tree-osqrth-time/>

Chapter 148

String transformation using XOR and OR

String transformation using XOR and OR - GeeksforGeeks

Given two binary strings. The task is to check if string s1 can be converted to string s2 by performing the given operations any number of times.

- Choose any two adjacent characters in a string s1 and replace one of them by $a \wedge b$ and the other by $a \vee b$ (a OR b).

Examples:

Input: S1 = “11”, S2 = “10”

Output: YES

Select two adjacent characters and replace $s2[0]$ by $s1[0] \wedge s1[1]$ and change $s2[1]$ by $s1[0] \vee s1[1]$

Input: S1 = “000”, S2 = “101”

Output: NO

Approach: Given below is a table which explains all the possibilities of XOR and OR operations.

X	Y	$X \wedge Y$	$X \vee Y$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

If the both the string consists of 0's only and their length is same, conversion is possible, as two adjacent zero will result in zeros only, irrespective of the operation done on it. If the both the string have 1's, follow the steps below to check if String1 can be converted to String2.

- Check if lengths are equal or not
- Check if both the strings have a minimum of one 1, as all the conversions are possible if both the strings have atleast 1 which can be seen in the table

If both of the above conditions are true, it is possible to convert String1 can be converted to String2.

Below is the implementation of above approach:

C++

```
// C++ program to check if string1 can be
// converted to string2 using XOR and OR operations
#include <bits/stdc++.h>
using namespace std;

// function to check if conversion is possible or not
bool solve(string s1, string s2)
{
    bool flag1 = 0, flag2 = 0;

    // if lengths are different
    if (s1.length() != s2.length())
        return false;

    int l = s1.length();

    // iterate to check if both strings have 1
    for (int i = 0; i < l; i++) {

        // to check if there is
        // even one 1 in string s1
        if (s1[i] == '1')
            flag1 = 1;

        // to check if there is even
        // one 1 in string s2
        if (s2[i] == '1')
            flag2 = 1;

        if (flag1 && flag2)
            return true;
    }
}
```

```
}

// if both string do not have a '1'.
return false;
}

// Driver code
int main()
{
    string s1 = "100101";
    string s2 = "100000";

    if (solve(s1, s2))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// Java program to check if
// string1 can be converted
// to string2 using XOR and
// OR operations
import java.io.*;
import java.util.*;

class GFG
{

    // function to check if
    // conversion is possible
    // or not
    static boolean solve(String s1,
                         String s2)
    {
        boolean flag1 = false,
              flag2 = false;

        // if lengths are different
        if (s1.length() != s2.length())
            return false;

        int l = s1.length();

        // iterate to check if
        // both strings have 1
```

```
for (int i = 0; i < l; i++)
{
    // to check if there is
    // even one 1 in string s1
    if (s1.charAt(i) == '1')
        flag1 = true;

    // to check if there is even
    // one 1 in string s2
    if (s2.charAt(i) == '1')
        flag2 = true;

    if (flag1 == true &&
        flag2 == true)
        return true;
}

// if both string do
// not have a '1'.
return false;
}

// Driver code
public static void main(String args[])
{
    String s1 = "100101";
    String s2 = "100000";

    if (solve(s1, s2) == true)
        System.out.print("Yes");
    else
        System.out.print("No");
}
}
```

Python3

```
# Python3 program to check
# if string1 can be converted
# to string2 using XOR and
# OR operations

# function to check if
# conversion is possible or not
def solve(s1, s2):
    flag1 = 0
    flag2 = 0
```

```
# if lengths are different
if (len(s1) != len(s2)):
    return False

l = len(s1)

# iterate to check if
# both strings have 1
for i in range (0, l):

    # to check if there is
    # even one 1 in string s1
    if (s1[i] == '1'):
        flag1 = 1;

    # to check if there is even
    # one 1 in string s2
    if (s2[i] == '1'):
        flag2 = 1

    # if both string
    # do not have a '1'.
    if (flag1 & flag2):
        return True
return False

# Driver code
s1 = "100101"
s2 = "100000"

if solve(s1, s2):
    print( "Yes")
else:
    print("No")

# This code is contributed
# by Shivi_Aggarwal

C#
// C# program to check if
// string1 can be converted
// to string2 using XOR and
// OR operations
using System;

class GFG
```

```
{\n\n// function to check if\n// conversion is possible\n// or not\nstatic bool solve(String s1,\n                  String s2)\n{\n    bool flag1 = false,\n        flag2 = false;\n\n    // if lengths are different\n    if (s1.Length != s2.Length)\n        return false;\n\n    int l = s1.Length;\n\n    // iterate to check if\n    // both strings have 1\n    for (int i = 0; i < l; i++)\n    {\n\n        // to check if there is\n        // even one 1 in string s1\n        if (s1[i] == '1')\n            flag1 = true;\n\n        // to check if there is even\n        // one 1 in string s2\n        if (s2[i] == '1')\n            flag2 = true;\n\n        if (flag1 == true &&\n            flag2 == true)\n            return true;\n    }\n\n    // if both string do\n    // not have a '1'.\n    return false;\n}\n\n// Driver code\npublic static void Main()\n{\n    String s1 = "100101";\n    String s2 = "100000";\n}
```

```
    if (solve(s1, s2) == true)
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
<?php
// PHP program to check if string1
// can be converted to string2
// using XOR and OR operations

// function to check if conversion
// is possible or not
function solve($s1, $s2)
{

    // if lengths are different
    if (strlen($s1) != strlen($s2))
        return false;

    $l = strlen($s1);

    // iterate to check if
    // both strings have 1
    for ($i = 0; $i < l; $i++)
    {

        // to check if there is
        // even one 1 in string s1
        if ($s1[$i] == '1')
            $flag1 = 1;

        // to check if there is even
        // one 1 in string s2
        if ($s2[$i] == '1')
            $flag2 = 1;

        if ($flag1 && $flag2)
            return true;
    }

    // if both string do
```

```
// not have a '1'.
return false;
}

// Driver code
\$s1 = "100101";
\$s2 = "100000";

if (solve(\$s1, \$s2))
    echo("Yes");
else
    echo("No");

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

Yes

Time Complexity: O(n) where n is length of input strings.

Improved By : [Shivi_Aggarwal](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/string-transformation-using-xor-and-or/>

Chapter 149

Sum of all prime divisors of all the numbers in range L-R

Sum of all prime divisors of all the numbers in range L-R - GeeksforGeeks

Given two integers L and R. The task is to find the sum of all prime factors of every number in the range[L-R].

Examples:

Input: l = 5, r = 10

Output: 17

5 is prime, hence sum of factors = 0

6 has prime factors 2 and 3, hence sum = 5

7 is prime, hence sum = 0

8 has prime factor 2, hence sum = 2

9 has prime factor 3, hence sum = 3

10 has prime factors 2 and 5, hence sum = 7

Hence, total sum = $5 + 2 + 3 + 7 = 17$

Input: l = 18, r = 25

Output: 45

18 has prime factors 2, 3 hence sum = 5

19 is prime, hence sum of factors = 0

20 has prime factors 2 and 5, hence sum = 7

21 has prime factors 3 and 7, hence sum = 10

22 has prime factors 2 and 11, hence sum = 13

23 is prime. hence sum = 0

24 has prime factors 2 and 3, hence sum = 5

25 has prime factor 5, hence sum = 5

Hence, total sum = $5 + 7 + 10 + 13 + 5 + 5 = 45$

A **naive** approach would be to start iterating through all numbers from 1 to r. For each iteration, start from 2 to i and find if i is divisible by that number, if it is divisible, we simply add i and proceed.

Below is the implementation of the above approach.

Java

```
// Java program to find the sum of prime
// factors of all numbers in range [L-R]
class gfg {
    static boolean isPrime(int n)
    {
        for (int i = 2; i * i <= n; i++) {

            // n has a factor, hence not a prime
            if (n % i == 0)
                return false;
        }
        // we reach here if n has no factors
        // and hence n is a prime number
        return true;
    }
    static int sum(int l, int r)
    {
        int sum = 0;

        // iterate from lower to upper
        for (int i = l; i <= r; i++) {

            // if i is prime, it has no factors
            if (isPrime(i))
                continue;
            for (int j = 2; j < i; j++) {

                // check if j is a prime factor of i
                if (i % j == 0 && isPrime(j))
                    sum += j;
            }
        }
        return sum;
    }
    public static void main(String[] args)
    {
        int l = 18, r = 25;
        System.out.println(sum(l, r));
    }
}
```

C#

```
// C# program to find the sum
```

```
// of prime factors of all
// numbers in range [L-R]
using System;

class GFG
{
    static bool isPrime(int n)
    {
        for (int i = 2;
              i * i <= n; i++)
        {

            // n has a factor,
            // hence not a prime
            if (n % i == 0)
                return false;
        }

        // we reach here if n has
        // no factors and hence n
        // is a prime number
        return true;
    }

    static int sum(int l, int r)
    {
        int sum = 0;

        // iterate from lower to upper
        for (int i = l; i <= r; i++)
        {

            // if i is prime, it
            // has no factors
            if (isPrime(i))
                continue;
            for (int j = 2; j < i; j++)
            {

                // check if j is a
                // prime factor of i
                if (i % j == 0 && isPrime(j))
                    sum += j;
            }
        }
        return sum;
    }
}
```

```
// Driver code
public static void Main()
{
    int l = 18, r = 25;
    Console.WriteLine(sum(l, r));
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
<?php
// PHP program to find the
// sum of prime factors of
// all numbers in range [L-R]
function isPrime($n)
{
    for ($i = 2; $i * $i <= $n; $i++)
    {
        // n has a factor, hence
        // not a prime
        if ($n % $i == 0)
            return false;
    }

    // we reach here if n has
    // no factors and hence n
    // is a prime number
    return true;
}

function sum1($l, $r)
{
    $sum = 0;

    // iterate from lower to upper
    for ($i = $l; $i <= $r; $i++)
    {

        // if i is prime, it
        // has no factors
        if (isPrime($i))
            continue;
        for ($j = 2; $j < $i; $j++)
        {
```

```
// check if j is a
// prime factor of i
if ($i % $j == 0 && isPrime($j))
    $sum += $j;
}
}
return $sum;
}

// Drive Code
$l = 18;
$r = 25;
echo sum1($l, $r);

// This code is contributed by mits
?>
```

Output:

45

Time Complexity: $O(N * N * \sqrt{N})$

An **efficient** approach is to modify [sieve of eratosthenes](#) slightly to [find sum of all prime divisors](#). Next, maintain a [prefix array](#) to keep the sum of sum of all prime divisors upto index i. Hence `pref_arr[r] - pref_arr[l-1]` would give the answer.

Below is the implementation of the above approach.

Java

```
// Java program to find the sum of prime
// factors of all numbers in range [L-R]
public class gfg {

    static int N = 10000;
    static long arr[] = new long[N];

    // function to compute the seive
    static void seive()
    {
        for (int i = 2; i * i < N; i++) {

            // i is prime
            if (arr[i] == 0) {

                // add i to all the multiples of i till N
                for (int j = 2; i * j < N; j++) {
```

```
        arr[i * j] += i;
    }
}
}

// function that returns the sum
static long sum(int l, int r)
{
    // Function call to compute seive
    seive();

    // prefix array to keep the sum of all arr[i] till i
    long[] pref_arr = new long[r + 1];
    pref_arr[0] = arr[0];

    // calculate the prefix sum of prime divisors
    for (int i = 1; i <= r; i++) {
        pref_arr[i] = pref_arr[i - 1] + arr[i];
    }

    // lower is the beginning of array
    if (l == 1)
        return (pref_arr[r]);

    // lower is not the beginning of the array
    else
        return (pref_arr[r] - pref_arr[l - 1]);
}

// Driver Code
public static void main(String[] args)
{
    int l = 5, r = 10;
    System.out.println(sum(l, r));
}
```

C#

```
// C# program to find the sum
// of prime factors of all
// numbers in range [L-R]
using System;

class GFG
{
```

```
static int N = 10000;
static long[] arr = new long[N];

// function to compute
// the seive
static void seive()
{
    for (int i = 2; i * i < N; i++)
    {

        // i is prime
        if (arr[i] == 0)
        {

            // add i to all the multiples
            // of i till N
            for (int j = 2;
                 i * j < N; j++)
            {
                arr[i * j] += i;
            }
        }
    }
}

// function that
// returns the sum
static long sum(int l, int r)
{

    // Function call to
    // compute seive
    seive();

    // prefix array to keep the
    // sum of all arr[i] till i
    long[] pref_arr = new long[r + 1];
    pref_arr[0] = arr[0];

    // calculate the prefix
    // sum of prime divisors
    for (int i = 1; i <= r; i++)
    {
        pref_arr[i] = pref_arr[i - 1] +
                      arr[i];
    }

    // lower is the beginning
```

```
// of array
if (l == 1)
    return (pref_arr[r]);

// lower is not the
// beginning of the array
else
    return (pref_arr[r] -
            pref_arr[l - 1]);
}

// Driver Code
public static void Main()
{
    int l = 5, r = 10;
    Console.WriteLine(sum(l, r));
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

Output:

17

Improved By : [Abby_akku](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/sum-of-all-prime-divisors-of-all-the-numbers-in-range-l-r/>

Chapter 150

Sum of decimal equivalent of all possible pairs of Binary representation of a Number

Sum of decimal equivalent of all possible pairs of Binary representation of a Number - GeeksforGeeks

Given a number N. The task is to find the sum of the decimal equivalent of all the pairs formed from the binary representation of the given number.

Examples:

Input: N = 4

Output: 4

Binary equivalent of 4 is 100.

All possible pairs are 10, 10, 00 and their decimal equivalent are 2, 2, 0 respectively.

So, $2 + 2 + 0 = 4$

Input: N = 11

Output: 13

All possible pairs are: 10, 11, 11, 01, 01, 11

Sum = $2 + 3 + 3 + 1 + 1 + 3 = 13$

Approach:

1. Find the binary equivalent of N and store it in a vector.
2. Run two loops to consider each and every pair formed from the bits of binary equivalent stored in the vector.
3. Find the decimal equivalent of all the pairs and add them.
4. Return the sum.

Below is the implementation of the above approach:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the sum
int sumOfPairs(int n)
{

    // Store the Binary equivalent of decimal
    // number in reverse order
    vector<int> v;
    int sum = 0;

    // Calculate binary equivalent of decimal number
    while (n > 0) {
        v.push_back(n % 2);
        n = n / 2;
    }

    // for correct binary representation
    reverse(v.begin(), v.end());

    // Consider every pair
    for (int i = 0; i < v.size() - 1; i++) {
        for (int j = i + 1; j < v.size(); j++)

        {

            // handles all combinations of 01
            if (v[i] == 0 && v[j] == 1)
                sum += 1;

            // handles all combinations of 11
            if (v[i] == 1 && v[j] == 1)
                sum += 3;

            // handles all combinations of 10
            if (v[i] == 1 && v[j] == 0)
                sum += 2;
        }
    }

    return sum;
}

// Driver code
```

```
int main()
{
    int N = 5;

    cout << sumOfPairs(N);

    return 0;
}
```

Java

```
// Java implementation of above approach
import java.util.*;

class GFG
{
    public static int sumOfPairs(int n)
    {
        // Store the Binary equivalent
        // of decimal number in reverse order
        ArrayList v = new ArrayList();
        int sum = 0;

        // Calculate binary equivalent
        // of decimal number
        while (n > 0)
        {
            v.add(n % 2);
            n = n / 2;
        }

        Collections.reverse(v);

        for (int i = 0; i < v.size() - 1; i++) {
            for (int j = i + 1; j < v.size(); j++) {
                // handles all combinations of 01 if (v.get(i) == 0 && v.get(j) == 1)
                sum += 1; // handles all combinations of 11 if (v.get(i) == 1 && v.get(j) == 1)
                sum += 3; // handles all combinations of 10 if (v.get(i) == 1 && v.get(j) == 0)
                sum += 2; }
            } }
        return sum;
    } }

public class Main {
    public static void main (String[] args) {
        int N = 5;
        System.out.print(sumOfPairs(N));
    }
}
```

for (int i = 0; i < v.size() - 1; i++) { for (int j = i + 1; j < v.size(); j++) { // handles all combinations of 01 if (v.get(i) == 0 && v.get(j) == 1) sum += 1; // handles all combinations of 11 if (v.get(i) == 1 && v.get(j) == 1) sum += 3; // handles all combinations of 10 if (v.get(i) == 1 && v.get(j) == 0) sum += 2; } } return sum; } // Driver Code public static void main (String[] args) { int N = 5; System.out.print(sumOfPairs(N)); } } // This code is contributed by Kirti_Mangal [tabby title = "Python 3"]

```
# Python3 program to find the sum

# Function to find the sum
def sumofPairs(n) :

    # Store the Binary equivalent of decimal
    # number in reverse order
    v = []
    sum = 0
```

```
# Calculate binary equivalent of decimal number
while n > 0 :
    v.append(n % 2)
    n = n // 2

# for correct binary representation
v.reverse()

# Consider every pair
for i in range(len(v) - 1) :

    for j in range(i + 1, len(v)) :

        # handles all combinations of 01
        if v[i] == 0 and v[j] == 1 :
            sum += 1

        # handles all combinations of 11
        if v[i] == 1 and v[j] == 1 :
            sum += 3

        # handles all combinations of 10
        if v[i] == 1 and v[j] == 0 :
            sum += 2

return sum

# Driver Code
if __name__ == "__main__":
    N = 5

    # function calling
    print(sumofPairs(N))

# This code is contributed by ANKITRAI1
```

Output:

6

Improved By : [ANKITRAI1](#), [Kirti_Mangal](#)

Source

<https://www.geeksforgeeks.org/sum-of-decimal-equivalent-of-all-possible-pairs-of-binary-representation-of-a-number/>

Chapter 151

Sum of elements in range L-R where first half and second half is filled with odd and even numbers

Sum of elements in range L-R where first half and second half is filled with odd and even numbers - GeeksforGeeks

Given a number N, create an array such the first half of the array is filled with odd numbers till N and second half of the array is filled with even numbers. Also given are L and R indices, the task is to print the sum of elements in the array in the range [L, R].

Examples:

Input: N = 12, L = 1, R = 11

Output: 66

The array formed thus is {1, 3, 5, 7, 9, 11, 2, 4, 6, 8, 10, 12}

The sum between index 1 and index 11 is 66 {1+3+5+7+9+11+2+4+6+8+10}

Input: N = 11, L = 3 and R = 7

Output: 34

The array formed is {1, 3, 5, 7, 9, 11, 2, 4, 6, 8, 10}

The sum between index 3 and index 7 is 34 {5+7+9+11+2}

A **naive approach** will be to form the array of size N and iterate from L to R and return the sum.

Java

```
// Java program to find the sum between L-R
// range by creating the array
```

```
// Naive Approach
import java.io.*;
public class GFG {

    // Function to find the sum between L and R
    static int rangesum(int n, int l, int r)
    {
        // array created
        int[] arr = new int[n];

        // fill the first half of array
        int c = 1, i = 0;
        while (c <= n) {
            arr[i++] = c;
            c += 2;
        }

        // fill the second half of array
        c = 2;
        while (c <= n) {
            arr[i++] = c;
            c += 2;
        }
        int sum = 0;

        // find the sum between range
        for (i = l - 1; i < r; i++) {
            sum += arr[i];
        }

        return sum;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int n = 12;
        int l = 1, r = 11;
        System.out.println(rangesum(n, l, r));
    }
}
```

C#

```
// C# program to find the sum
// between L-R range by creating
// the array Naive Approach
using System;
```

```
class GFG
{
    // Function to find the
    // sum between L and R
    static int rangesum(int n,
                        int l, int r)
    {
        // array created
        int[] arr = new int[n];

        // fill the first
        // half of array
        int c = 1, i = 0;
        while (c <= n)
        {
            arr[i++] = c;
            c += 2;
        }

        // fill the second
        // half of array
        c = 2;
        while (c <= n)
        {
            arr[i++] = c;
            c += 2;
        }
        int sum = 0;

        // find the sum
        // between range
        for (i = l - 1; i < r; i++)
        {
            sum += arr[i];
        }

        return sum;
    }

    // Driver Code
    public static void Main()
    {
        int n = 12;
        int l = 1, r = 11;
        Console.WriteLine(rangesum(n, l, r));
    }
}
```

}

```
// This code is contributed  
// by inder_verma.
```

Output:

66

Time complexity: O(N)

Auxiliary Space: O(N)

Efficient Approach: Without building the array the problem can be solved in O(1) complexity. Consider the middle point of the sequence thus formed. Then there can be 3 possibilities for L and R.

- l and r both are on the right of middle point.
- l and r both are on the left of middle point.
- l is on the left of the middle point and r is on the left of the middle point.

For 1st and 2nd case, just find the number which corresponds to the l-th position from the start or middle and the number which corresponds to the r-th position from start or middle. The below formula can be used to find out the sum:

```
sum = (no. of terms in the range)*(first term + last term)/2
```

For the third case, consider it as [L-mid] and [mid-R] and then apply case 1 and case 2 formula as mentioned above.

Below is the implementation of the above approach:

Java

```
// Java program to find the sum between L-R  
// range by creating the array  
// Efficient Approach  
import java.io.*;  
public class GFG {  
  
    // // Function to calculate the sum if n is even  
    static int sumeven(int n, int l, int r)  
    {  
        int sum = 0;  
        int mid = n / 2;
```

```
// both l and r are to the left of mid
if (r <= mid) {
    // first and last element
    int first = (2 * l - 1);
    int last = (2 * r - 1);

    // Total number of terms in
    // the sequence is r-l+1
    int no_of_terms = r - l + 1;

    // use of formula derived
    sum = ((no_of_terms) * ((first + last))) / 2;
}

// both l and r are to the right of mid
else if (l >= mid) {
    // // first and last element
    int first = (2 * (l - n / 2));
    int last = (2 * (r - n / 2));

    int no_of_terms = r - l + 1;

    // Use of formula derived
    sum = ((no_of_terms) * ((first + last))) / 2;
}

// left is to the left of mid and
// right is to the right of mid
else {
    // Take two sums i.e left and
    // right differently and add
    int sumleft = 0, sumright = 0;

    // first and last element
    int first_term1 = (2 * l - 1);
    int last_term1 = (2 * (n / 2) - 1);

    // total terms
    int no_of_terms1 = n / 2 - l + 1;

    // no of terms
    sumleft = ((no_of_terms1) * ((first_term1 + last_term1))) / 2;

    // The first even number is 2
    int first_term2 = 2;

    // The last element is given by 2*(r-n/2)
    int last_term2 = (2 * (r - n / 2));
    int no_of_terms2 = r - mid;
```

```
// formula applied
sumright = ((no_of_terms2) * ((first_term2 + last_term2))) / 2;
sum = (sumleft + sumright);
}

return sum;
}

// Function to calculate the sum if n is odd
static int sumodd(int n, int l, int r)
{

// take ceil value if n is odd
int mid = n / 2 + 1;
int sum = 0;

// // both l and r are to the left of mid
if (r <= mid) {
    // first and last element
    int first = (2 * l - 1);
    int last = (2 * r - 1);

    // number of terms
    int no_of_terms = r - l + 1;

    // formula
    sum = ((no_of_terms) * ((first + last))) / 2;
}

// // both l and r are to the right of mid
else if (l > mid) {

    // firts and last ter,
    int first = (2 * (l - mid));
    int last = (2 * (r - mid));

    // no of terms
    int no_of_terms = r - l + 1;

    // formula used
    sum = ((no_of_terms) * ((first + last))) / 2;
}

// If l is on left and r on right
else {

    // calculate separate sums
```

```
int sumleft = 0, sumright = 0;

// first half
int first_term1 = (2 * l - 1);
int last_term1 = (2 * mid - 1);

// calculate terms
int no_of_terms1 = mid - l + 1;
sumleft = ((no_of_terms1) * ((first_term1 + last_term1))) / 2;

// second half
int first_term2 = 2;
int last_term2 = (2 * (r - mid));
int no_of_terms2 = r - mid;
sumright = ((no_of_terms2) * ((first_term2 + last_term2))) / 2;

// add both halves
sum = (sumleft + sumright);
}

return sum;
}
// Function to find the sum between L and R
static int rangesum(int n, int l, int r)
{
    int sum = 0;

    // If n is even
    if (n % 2 == 0)
        return sumeven(n, l, r);

    // If n is odd
    else
        return sumodd(n, l, r);
}

// Driver Code
public static void main(String[] args)
{
    int n = 12;
    int l = 1, r = 11;
    System.out.println(rangesum(n, l, r));
}
```

C#

```
// C# program to find the sum
```

```
// between L-R range by creating
// the array Efficient Approach
using System;

class GFG
{

    // Function to calculate
    // the sum if n is even
    static int sumeven(int n,
                       int l, int r)
    {
        int sum = 0;
        int mid = n / 2;

        // both l and r are
        // to the left of mid
        if (r <= mid)
        {
            // first and last element
            int first = (2 * l - 1);
            int last = (2 * r - 1);

            // Total number of terms in
            // the sequence is r-l+1
            int no_of_terms = r - l + 1;

            // use of formula derived
            sum = ((no_of_terms) *
                   ((first + last))) / 2;
        }

        // both l and r are
        // to the right of mid
        else if (l >= mid)
        {
            // first and last element
            int first = (2 * (l - n / 2));
            int last = (2 * (r - n / 2));

            int no_of_terms = r - l + 1;

            // Use of formula derived
            sum = ((no_of_terms) *
                   ((first + last))) / 2;
        }

        // left is to the left of
    }
}
```

```
// mid and right is to the
// right of mid
else
{
    // Take two sums i.e left and
    // right differently and add
    int sumleft = 0, sumright = 0;

    // first and last element
    int first_term1 = (2 * l - 1);
    int last_term1 = (2 * (n / 2) - 1);

    // total terms
    int no_of_terms1 = n / 2 - l + 1;

    // no of terms
    sumleft = ((no_of_terms1) *
                ((first_term1 +
                  last_term1)) / 2;

    // The first even
    // number is 2
    int first_term2 = 2;

    // The last element is
    // given by 2*(r-n/2)
    int last_term2 = (2 * (r - n / 2));
    int no_of_terms2 = r - mid;

    // formula applied
    sumright = ((no_of_terms2) *
                ((first_term2 +
                  last_term2)) / 2;
    sum = (sumleft + sumright);
}

return sum;
}

// Function to calculate
// the sum if n is odd
static int sumodd(int n,
                  int l, int r)
{

    // take ceil value
    // if n is odd
    int mid = n / 2 + 1;
```

```
int sum = 0;

// both l and r are
// to the left of mid
if (r <= mid)
{
    // first and last element
    int first = (2 * l - 1);
    int last = (2 * r - 1);

    // number of terms
    int no_of_terms = r - l + 1;

    // formula
    sum = ((no_of_terms) *
           ((first + last))) / 2;
}

// both l and r are
// to the right of mid
else if (l > mid)
{
    // first and last ter,
    int first = (2 * (l - mid));
    int last = (2 * (r - mid));

    // no of terms
    int no_of_terms = r - l + 1;

    // formula used
    sum = ((no_of_terms) *
           ((first + last))) / 2;
}

// If l is on left
// and r on right
else
{
    // calculate separate sums
    int sumleft = 0, sumright = 0;

    // first half
    int first_term1 = (2 * l - 1);
    int last_term1 = (2 * mid - 1);

    // calculate terms
```

```
int no_of_terms1 = mid - l + 1;
sumleft = ((no_of_terms1) *
            ((first_term1 +
              last_term1))) / 2;

// second half
int first_term2 = 2;
int last_term2 = (2 * (r - mid));
int no_of_terms2 = r - mid;
sumright = ((no_of_terms2) *
            ((first_term2 +
              last_term2))) / 2;

// add both halves
sum = (sumleft + sumright);
}

return sum;
}

// Function to find the
// sum between L and R
static int rangesum(int n,
                     int l, int r)
{

    // If n is even
    if (n % 2 == 0)
        return sumeven(n, l, r);

    // If n is odd
    else
        return sumodd(n, l, r);
}

// Driver Code
public static void Main()
{
    int n = 12;
    int l = 1, r = 11;
    Console.WriteLine(rangesum(n, l, r));
}
}

// This code is contributed
// by chandan_jnu.
```

Output:

66

Time complexity: O(1)

Auxiliary Space: O(1)

Improved By : [inderDuMCA](#), [Chandan_Kumar](#)

Source

<https://www.geeksforgeeks.org/sum-of-elements-in-range-l-r-where-first-half-and-second-half-is-filled-with-odd-and-even-numbers/>

Chapter 152

Sum of elements of all partitions of number such that no element is less than K

Sum of elements of all partitions of number such that no element is less than K - Geeks-forGeeks

Given an integer N, the task is to find an aggregate sum of all integer partitions of this number such that each partition does not contain any integer less than K.

Examples:

Input: N = 6 and K = 2

Output: 24

In this case, there are 4 valid partitions.

- 1) {6}
- 2) {4, 2}
- 3) {3, 3}
- 4) {2, 2, 2}

Therefore, aggregate sum would be

$$6 + 4 + 2 + 3 + 3 + 2 + 2 + 2 = 24$$

Input: N = 10 and K = 3

Output: 50

Here, 5 valid partitions are:

- 1) {10}
- 2) {7, 3}
- 3) {6, 4}
- 4) {5, 5}
- 5) {3, 3, 4}

Aggregate sum in this case would be

$$10 + 7 + 3 + 6 + 4 + 5 + 5 + 3 + 3 + 4 = 50$$

Approach: This problem has a simple **recursive** solution. First, we need to count the total number of valid partitions of number N such that each partition contains integers greater than or equal to K. So we will iteratively apply our recursive solution to find valid partitions that have the minimum integer K, K+1, K+2, ..., N.

Our final answer would be **N * no of valid partitions** because each valid partition has a sum equal to N.

Following are some key ideas for designing recursive function to find total number of valid partitions.

- If $N < K$ then no partition is possible.
- If $N < 2K$ then only one partition is possible and that is the number N itself.
- We can find number partitions in a recursive manner that contains integers at least equal to 'i' ('i' can be from K to N) and add them all to get final answer.

Pseudo code for recursive function to find number of valid partitions:

```
f(N,K):
    if N < K
        return 0
    if N < 2K
        return 1
    Initialize answer = 1
    FOR i from K to N
        answer = answer + f(N-i,i)
    return answer
```

Below is the **Dynamic Programming** solution:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function that returns total number of valid
// partitions of integer N
long long int countPartitions(int n, int k)
{

    // Global declaration of 2D dp array
    // which will be later used for memoization
    long long int dp[201][201];

    // initializing 2D dp array with -1
```

```
// we will use this 2D array for memoization
for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < n + 1; j++) {
        dp[i][j] = -1;
    }
}

// if this subproblem is already previously
// calculated, then directly return that answer
if (dp[n][k] >= 0)
    return dp[n][k];

// if N < K, then no valid
// partition is possible
if (n < k)
    return 0;

// if N is between K to 2*K then
// there is only one
// partition and that is the number N itself
if (n < 2 * k)
    return 1;

// Initialize answer with 1 as
// the number N itself
// is always a valid partition
long long int answer = 1;

// for loop to iterate over K to N
// and find number of
// possible valid partitions recursively.
for (int i = k; i < n; i++)
    answer = answer + countPartitions(n - i, i);

// memoization is done by storing
// this calculated answer
dp[n][k] = answer;

// returning number of valid partitions
return answer;
}

// Driver code
int main()
{
    int n = 10, k = 3;

    // Printing total number of valid partitions
```

```
    cout << "Total Aggregate sum of all Valid Partitions: "
    << countPartitions(n, k) * n;

    return 0;
}
```

Java

```
// Java implementation of
// above approach
class GFG
{
// Function that returns
// total number of valid
// partitions of integer N
static long countPartitions(int n, int k)
{

    // Global declaration of 2D
    // dp array which will be
    // later used for memoization
    long[][] dp = new long[201][201];

    // initializing 2D dp array
    // with -1 we will use this
    // 2D array for memoization
    for (int i = 0; i < n + 1; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            dp[i][j] = -1;
        }
    }

    // if this subproblem is already
    // previously calculated, then
    // directly return that answer
    if (dp[n][k] >= 0)
        return dp[n][k];

    // if N < K, then no valid
    // partition is possible
    if (n < k)
        return 0;

    // if N is between K to 2*K
    // then there is only one
    // partition and that is
```

```
// the number N itself
if (n < 2 * k)
    return 1;

// Initialize answer with 1
// as the number N itself
// is always a valid partition
long answer = 1;

// for loop to iterate over
// K to N and find number of
// possible valid partitions
// recursively.
for (int i = k; i < n; i++)
    answer = answer +
        countPartitions(n - i, i);

// memoization is done by storing
// this calculated answer
dp[n][k] = answer;

// returning number of
// valid partitions
return answer;
}

// Driver code
public static void main(String[] args)
{
    int n = 10, k = 3;

    // Printing total number
    // of valid partitions
    System.out.println("Total Aggregate sum of " +
        "all Valid Partitions: " +
        countPartitions(n, k) * n);
}
}

// This code is contributed by mits
```

Output:

Total Aggregate sum of all Valid Partitions: 50

Time Complexity: $O(N^2)$

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/sum-of-elements-of-all-partitions-of-number-such-that-no-element-is-less-than-k/>

Chapter 153

Sum of $f(a[i], a[j])$ over all pairs in an array of n integers

Sum of $f(a[i], a[j])$ over all pairs in an array of n integers - GeeksforGeeks

Given an array of n integers, find the sum of $f(a[i], a[j])$ of all pairs (i, j) such that $(1 \leq i < j \leq n)$.

$f(a[i], a[j])$:

```
If |a[j]-a[i]| > 1
    f(a[i], a[j]) = a[j] - a[i]
Else // if |a[j]-a[i]| <= 1
    f(a[i], a[j]) = 0
```

Examples:

Input : 6 6 4 4
Output : -8
Explanation:
All pairs are: $(6 - 6) + (6 - 6) +$
 $(6 - 6) + (4 - 6) + (4 - 6) + (4 - 6) +$
 $(4 - 6) + (4 - 4) + (4 - 4) = -8$

Input: 1 2 3 1 3
Output: 4
Explanation: the pairs that add up are:
(3, 1), (3, 1) to give 4, rest all pairs
according to condition gives 0.

A **naive approach** is to iterate through all pairs and calculate $f(a[i], a[j])$ and summing it while traversing in two nested loops will give us our answer.

Time Complexity: $O(n^2)$

A **efficient approach** will be to use a map/hash function to keep a count of every occurring numbers and then traverse through the list. While traversing through the list, we multiply the count of numbers that are before it and the number itself. Then subtract this result with the pre-sum of the number before that number to get the sum of difference of all pairs possible with that number. To remove all pairs whose absolute difference is $<=1$, simply subtract the count of occurrence of (number-1) and (number+1) from the previously computed sum. Here we subtract count of (number-1) from the computed sum as it had been previously added to the sum, and we add (number+1) count since the negative has been added to the pre-computed sum of all pairs.

Time Complexity : $O(n)$

Below is the implementation of the above approach :

C++

```
// CPP program to calculate the
// sum of f(a[i], a[j])
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the sum
int sum(int a[], int n)
{
    // map to keep a count of occurrences
    unordered_map<int, int> cnt;

    // Traverse in the list from start to end
    // number of times a[i] can be in a pair and
    // to get the difference we subtract pre_sum.
    int ans = 0, pre_sum = 0;
    for (int i = 0; i < n; i++) {
        ans += (i * a[i]) - pre_sum;
        pre_sum += a[i];

        // if the (a[i]-1) is present then
        // subtract that value as f(a[i], a[i]-1)=0
        if (cnt[a[i] - 1])
            ans -= cnt[a[i] - 1];

        // if the (a[i]+1) is present then
        // add that value as f(a[i], a[i]-1)=0
        // here we add as a[i]-(a[i]-1)<0 which would
        // have been added as negative sum, so we add
        // to remove this pair from the sum value
    }
}
```

```
    if (cnt[a[i] + 1])
        ans += cnt[a[i] + 1];

    // keeping a counter for every element
    cnt[a[i]]++;
}
return ans;
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3, 1, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << sum(a, n);
    return 0;
}
```

Java

```
// Java program to calculate
// the sum of f(a[i], aj])
import java.util.*;
public class GfG {

    // Function to calculate the sum
    public static int sum(int a[], int n)
    {
        // Map to keep a count of occurrences
        Map<Integer, Integer> cnt = new HashMap<Integer, Integer>();

        // Traverse in the list from start to end
        // number of times a[i] can be in a pair and
        // to get the difference we subtract pre_sum
        int ans = 0, pre_sum = 0;
        for (int i = 0; i < n; i++) {
            ans += (i * a[i]) - pre_sum;
            pre_sum += a[i];

            // If the (a[i]-1) is present then subtract
            // that value as f(a[i], a[i]-1) = 0
            if (cnt.containsKey(a[i] - 1))
                ans -= cnt.get(a[i] - 1);

            // If the (a[i]+1) is present then
            // add that value as f(a[i], a[i]-1)=0
            // here we add as a[i]-(a[i]-1)<0 which would
            // have been added as negative sum, so we add
```

```
// to remove this pair from the sum value
if (cnt.containsKey(a[i] + 1))
    ans += cnt.get(a[i] + 1);

// keeping a counter for every element
if(cnt.containsKey(a[i])) {
    cnt.put(a[i], cnt.get(a[i]) + 1);
}
else {
    cnt.put(a[i], 1);
}
}
return ans;
}

// Driver code
public static void main(String args[])
{
    int a[] = { 1, 2, 3, 1, 3 };
    int n = a.length;
    System.out.println(sum(a, n));
}
}

// This code is contributed by Swetank Modi
```

Output :

4

Source

<https://www.geeksforgeeks.org/sum-fai-aj-pairs-array-n-integers/>

Chapter 154

Sum of range in a series of first odd then even natural numbers

Sum of range in a series of first odd then even natural numbers - GeeksforGeeks

The sequence first consists of all the odd numbers starting from 1 to n and then remaining even numbers starting 2 up to n. Let's suppose we have n as 1000. Then the sequence becomes 1 3 5 7....999 2 4 6....1000

We are given a range (L, R), we need to find sum of numbers of this sequence in given range.

Note: Here the range is given as (L, R) L and R are included in the range

Examples:

```
Input : n = 10
        Range 1 6
Output : 27
Explanation:
Sequence is 1 3 5 7 9 2 4 6 8 10
Sum in range (2, 6)
= 1 + 3 + 5 + 7 + 9 + 2
= 27
```

```
Input : n = 5
        Range 1 2
Output : 4
Explanation:
sequence is 1 3 5 2 4
sum = 1 + 3 = 4
```

The idea is to first find sum of numbers before left(excluding left), then find sum of numbers before right (including right). We get result as second sum minus first sum.

How to find sum till a limit?

We first count how many odd numbers are there, then we use formulas for [sum of odd natural numbers](#) and [sum of even natural numbers](#) to find the result.

How to find count of odd numbers?

- If n is odd then the number of odd numbers are $((n/2) + 1)$
- If n is even then number of odd numbers are $(n/2)$

By simple observation, we get the number of odd numbers is $\text{ceil}(n/2)$. So, the number of even numbers are $n - \text{ceil}(n/2)$.

- Sum of first N odd numbers is (N^2)
- Sum of first N even numbers is $(N^2) + N$

For a given number x how will we find the sum in the sequence from 1 to x?
let's suppose x is less than the number of odd numbers.

- Then we simply return $(x*x)$

If the x is greater then the number of odd numbers

var = x-odd;

That means we need first var even numbers

we return $(\text{odd}*\text{odd}) + (\text{var}*\text{var}) + \text{var};$

C++

```
// CPP program to find sum in the given range in
// the sequence 1 3 5 7.....N 2 4 6...N-1
#include <bits/stdc++.h>
using namespace std;

// For our convenience
#define ll long long

// Function that returns sum
// in the range 1 to x in the
// sequence 1 3 5 7.....N 2 4 6...N-1
ll sumTillX(ll x, ll n)
{
    // number of odd numbers
    ll odd = ceil(n / 2.0);

    if (x <= odd)
        return x * x;
```

```
// number of extra even
// numbers required
ll even = x - odd;

    return ((odd * odd) + (even * even) + even);
}

int rangeSum(int N, int L, int R)
{
    return sumTillX(R, N) - sumTillX(L-1, N);
}

// Driver code
int main()
{
    ll N = 10, L = 1, R = 6;
    cout << rangeSum(N, L, R);
    return 0;
}
```

Java

```
// Java program to find
// sum in the given
// range in the sequence
// 1 3 5 7.....N
// 2 4 6...N-1

class GFG {

    // Function that returns sum
    // in the range 1 to x in the
    // sequence 1 3 5 7.....N 2 4 6...N-1
    static double sumTillX(double x,
                           double n)
    {

        // number of odd numbers
        double odd = Math.ceil(n / 2.0);

        if (x <= odd)
            return x * x;

        // number of extra even
        // numbers required
        double even = x - odd;

        return ((odd * odd) + (even *
```

```
        even) + even);
    }

    static double rangeSum(double N,
                          double L,
                          double R)
    {
        return sumTillX(R, N) -
               sumTillX(L-1, N);
    }

    // Driver Code
    public static void main(String args[])
    {
        long N = 10, L = 1, R = 6;
        int n = 101;
        System.out.println((int)rangeSum(N, L, R));

    }
}

// This code is contributed by Sam007
```

Python 3

```
# Python 3 program to find sum in the
# given range in the sequence 1 3 5 7
# .....N 2 4 6...N-1
import math

# For our convenience
#define ll long long

# Function that returns sum in the
# range 1 to x in the sequence
# 1 3 5 7.....N 2 4 6...N-1
def sumTillX(x, n):

    # number of odd numbers
    odd = math.ceil(n / 2.0)

    if (x <= odd):
        return x * x;

    # number of extra even
    # numbers required
    even = x - odd;
```

```
return ((odd * odd) +
       (even * even) + even);

def rangeSum(N, L, R):
    return (sumTillX(R, N) -
            sumTillX(L-1, N));

# Driver code
N = 10
L = 1
R = 6
print(rangeSum(N, L, R))

# This code is contributed by
# Smitha

C#
// C# program to find sum in the given
// range in the sequence 1 3 5 7.....N
// 2 4 6...N-1
using System;

public class GFG {

    // Function that returns sum
    // in the range 1 to x in the
    // sequence 1 3 5 7.....N 2 4 6...N-1
    static double sumTillX(double x, double n)
    {

        // number of odd numbers
        double odd = Math.Ceiling(n / 2.0);

        if (x <= odd)
            return x * x;

        // number of extra even
        // numbers required
        double even = x - odd;

        return ((odd * odd) + (even * even)
                + even);
    }

    static double rangeSum(double N, double L,
```

```
        double R)
{
    return sumTillX(R, N) - sumTillX(L-1, N);
}

// Driver code
public static void Main()
{
    long N = 10, L = 1, R = 6;
    Console.WriteLine(rangeSum(N, L, R));
}
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP program to find sum
// in the given range in the
// sequence 1 3 5 7.....
// N 2 4 6...N-1

// Function that returns sum
// in the range 1 to x in the
// sequence 1 3 5 7.....
// N 2 4 6...N-1
function sumTillX($x, $n)
{

    // number of odd numbers
    $odd = ceil($n / 2.0);

    if ($x <= $odd)
        return $x * $x;

    // number of extra even
    // numbers required
    $even = $x - $odd;

    return ((($odd * $odd) +
            ($even * $even) +
            $even));
}

function rangeSum($N, $L, $R)
{
    return sumTillX($R, $N) -
```

```
        sumTillX($L-1, $N);  
    }  
  
    // Driver code  
    $N = 10; $L = 1; $R = 6;  
    echo(rangeSum($N, $L, $R));  
  
    // This code is contributed by Ajit.  
    ?>
```

Output:

27

Improved By : [Sam007](#), [jit_t](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/sum-of-range-in-a-series-of-first-odd-then-even-natural-numbers/>

Chapter 155

Test Case Generation | Set 1 (Random Numbers, Arrays and Matrices)

Test Case Generation | Set 1 (Random Numbers, Arrays and Matrices) - GeeksforGeeks

The test cases are extremely important part of any “Software/Project Testing Process”. Hence this Set will be very important for all the aspiring software developers. The following are C++ program to generate test cases.

Generating Random Numbers

```
// A C++ Program to generate test cases for
// random number
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the range of the test data generated
#define MAX 10000000

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases.in", "w", stdout);

    // For random values every time
```

```
    srand(time(NULL));

    for (int i=1; i<=RUN; i++)
        printf("%d\n", rand() % MAX);

    // Uncomment the below line to store
    // the test data in a file
    //fclose(stdout);
    return(0);
}
```

Generating Random Arrays

```
// A C++ Program to generate test cases for
// array filled with random numbers
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the range of the test data generated
#define MAX 10000000

// Define the maximum number of array elements
#define MAXNUM 100

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    //freopen ("Test_Cases_Random_Array.in", "w", stdout);

    //For random values every time
    srand(time(NULL));

    for (int i=1; i<=RUN; i++)
    {
        // Number of array elements
        int NUM = 1 + rand() % MAXNUM;

        // First print the number of array elements
        printf("%d\n", NUM);

        // Then print the array elements separated
        // by space
        for (int j=1; j<=NUM; j++)
```

```
    printf("%d ", rand() % MAX);

    printf("\n");
}

// Uncomment the below line to store
// the test data in a file
//fclose(stdout);
return(0);
}
```

Generating Random Matrix

```
// A C++ Program to generate test cases for
// matrix filled with random numbers
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 3

// Define the range of the test data generated
#define MAX 100000

// Define the maximum rows in matrix
#define MAXROW 10

// Define the maximum columns in matrix
#define MAXCOL 10

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Random_Matrix.in", "w", stdout);

    // For random values every time
    srand(time(NULL));

    for (int i=1; i<=RUN; i++)
    {
        // Number of rows and columns
        int row = 1 + rand() % MAXROW;
        int col = 1 + rand() % MAXCOL;

        // First print the number of rows and columns
        printf("%d %d\n", row, col);
```

```
// Then print the matrix
for (int j=1; j<=row; j++)
{
    for (int k=1; k<=col; k++)
        printf("%d ", rand() % MAX);
    printf("\n");
}
printf("\n");

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Library Functions Used

1. **rand()** Function–

- > Generate random numbers from the range 0 to RAND_MAX (32767)
- > Defined in <stdlib.h>/<cstdlib> header
- > If we want to assign a random number in the range – m to n [n >= m] to a variable var, then use–
var = m + (rand() % (n - m + 1));
- > This function is a run-time function. So the values – n, m must be declared before compiling just like we have to declare the size of array before compiling.
- > Call this function every time you want to generate a random number

2. **time()** Function

- > Return the number of seconds from [00:00:00 UTC, January 1, 1970]
- > Defined in <time.h> header

3. **srand(seed)**

- > Generates random number according to the seed passed to it.
- > If this function is not used and we use rand() function then every time we run the program the same random numbers gets generated.
- > To overcome the above limitation, we pass time(NULL) as a seed. Hence srand(time(NULL)) is used to generate random values every time the program is made to run.
- > Always use this at the beginning of the program, i.e- just after int main() {
- > No need to call this function every time you generate a random number
- > Defined in <stdlib.h>/<cstdlib> header

4. **freopen(“output.txt”, “w”, stdout)**

- > Writes (that's why we passed “w” as the second argument) all the data to output.txt file (The file must be in the same file as the program is in).
- > Used to redirect stdout to a file.

-> If the output.txt file is not created before then it gets created in the same file as the program is in.

5. **fclose(stdout)**

-> Closes the standard output stream file to avoid leaks.

-> Always use this at the end of the program, i.e- just before return(0) in the int main() function.

Source

<https://www.geeksforgeeks.org/test-case-generation-set-1-random-numbers-arrays-and-matrices/>

Chapter 156

Test Case Generation | Set 2 (Random Characters, Strings and Arrays of Random Strings)

Test Case Generation | Set 2 (Random Characters, Strings and Arrays of Random Strings)
- GeeksforGeeks

[Set 1 \(Random Numbers, Arrays and Matrices\)](#)

- Generating Random Characters

```
// A C++ Program to generate test cases for
// random characters
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the range of the test data generated
// Here it is 'a' to 'z'
#define MAX 25

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Random_Character.in", "w", stdout);

    // For random values every time
```

```
    srand(time(NULL));  
  
    for (int i=1; i<=RUN; i++)  
        printf("%c\n", 'a' + rand() % MAX);  
  
    // Uncomment the below line to store  
    // the test data in a file  
    // fclose(stdout);  
    return(0);  
}
```

- Generating Random Strings

```
// A C++ Program to generate test cases for  
// random strings  
#include<bits/stdc++.h>  
using namespace std;  
  
// Define the number of runs for the test data  
// generated  
#define RUN 100000  
  
// Define the range of the test data generated  
// Here it is 'a' to 'z'  
#define MAX 25  
  
// Define the maximum length of string  
#define MAXLEN 100  
  
int main()  
{  
    // Uncomment the below line to store  
    // the test data in a file  
    // freopen ("Test_Cases_Random_String.in", "w", stdout);  
  
    //For random values every time  
    srand(time(NULL));  
  
    int LEN;      // Length of string  
  
    for (int i=1; i<=RUN; i++)  
    {  
        LEN = 1 + rand() % MAXLEN;  
  
        // First print the length of string  
        printf("%d\n", LEN);  
  
        // Then print the characters of the string
```

```
        for (int j=1; j<=LEN; j++)
            printf("%c", 'a' + rand() % MAX);

        printf("\n");
    }

    // Uncomment the below line to store
    // the test data in a file
    // fclose(stdout);
    return(0);
}
```

- Generating Array of Random Strings

```
// A C++ Program to generate test cases for
// random strings
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 1000

// Define the range of the test data generated
// Here it is 'a' to 'z'
#define MAX 25

// Define the range of number of strings in the array
#define MAXNUM 20

// Define the maximum length of string
#define MAXLEN 20

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Array_of_Strings.in", "w", stdout);

    //For random values every time
    srand(time(NULL));

    int NUM; // Number of strings in array

    int LEN; // Length of string

    for (int i=1; i<=RUN; i++)
    {
```

```
NUM = 1 + rand() % MAXNUM;
printf("%d\n", NUM);

for (int k=1; k<=NUM; k++)
{
    LEN = 1 + rand() % MAXLEN;

    // Then print the characters of the string
    for (int j=1; j<=LEN; j++)
        printf("%c", 'a' + rand() % MAX);

    printf(" ");
}
printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Source

<https://www.geeksforgeeks.org/test-case-generation-set-2-random-characters-strings-and-arrays-of-random-strings/>

Chapter 157

Test Case Generation | Set 3 (Unweighted and Weighted Trees)

Test Case Generation | Set 3 (Unweighted and Weighted Trees) - GeeksforGeeks

Generating Random Unweighted Trees

- Since this is a tree, the test data generation plan is such that no cycle gets formed.
- The number of edges is one less than the number of vertices
- For each **RUN** we first print the number of vertices – **NUM** first in a new separate line and the next **NUM-1** lines are of the form **(a b)** where **a** is the parent of **b**

```
// A C++ Program to generate test cases for
// an unweighted tree
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the maximum number of nodes of the tree
#define MAXNODE 20

class Tree
{
    int V;      // No. of vertices

    // Pointer to an array containing adjacency listss
    list<int> *adj;
```

```

// used by isCyclic()
bool isCyclicUtil(int v, bool visited[], bool *rs);
public:
    Tree(int V); // Constructor
    void addEdge(int v, int w); // adds an edge
    void removeEdge(int v, int w); // removes an edge

    // returns true if there is a cycle in this graph
    bool isCyclic();
};

// Constructor
Tree::Tree(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Tree::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Tree::removeEdge(int v, int w)
{
    list<int>::iterator it;
    for (it=adj[v].begin(); it!=adj[v].end(); it++)
    {
        if (*it == w)
        {
            adj[v].erase(it);
            break;
        }
    }
    return;
}

// This function is a variation of DFSUtil() in
// https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
bool Tree::isCyclicUtil(int v, bool visited[], bool *recStack)
{
    if (visited[v] == false)
    {
        // Mark the current node as visited and part of
        // recursion stack
        visited[v] = true;
        recStack[v] = true;

```

```
// Recur for all the vertices adjacent to this vertex
list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i)
{
    if (!visited[*i] && isCyclicUtil(*i, visited, recStack))
        return true;
    else if (recStack[*i])
        return true;
}

recStack[v] = false; // remove the vertex from recursion stack
return false;
}

// Returns true if the graph contains a cycle, else false.
// This function is a variation of DFS() in
// https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
bool Tree::isCyclic()
{
    // Mark all the vertices as not visited and not part of recursion
    // stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for(int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle in different
    // DFS trees
    for (int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}

int main()
{
    set<pair<int, int>> container;
    set<pair<int, int>>::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Unweighted_Tree.in", "w", stdout);
}
```

```
//For random values every time
srand(time(NULL));

int NUM;      // Number of Vertices/Nodes

for (int i=1; i<=RUN; i++)
{
    NUM = 1 + rand() % MAXNODE;

    // First print the number of vertices/nodes
    printf("%d\n", NUM);
    Tree t(NUM);
    // Then print the edges of the form (a b)
    // where 'a' is parent of 'b'
    for (int j=1; j<=NUM-1; j++)
    {
        int a = rand() % NUM;
        int b = rand() % NUM;
        pair<int, int> p = make_pair(a, b);

        t.addEdge(a, b);

        // Search for a random "new" edge everytime
        while (container.find(p) != container.end()
               || t.isCyclic() == true)
        {
            t.removeEdge(a, b);

            a = rand() % NUM;
            b = rand() % NUM;
            p = make_pair(a, b);
            t.addEdge(a, b);
        }
        container.insert(p);
    }

    for (it=container.begin(); it!=container.end(); ++it)
        printf("%d %d\n", it->first, it->second);

    container.clear();
    printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
```

}

Generating Random **Weighted** Trees

- Since this is a tree, the test data generation plan is such that no cycle gets formed.
- The number of edges is one less than the number of vertices
- For each **RUN** we first print the number of vertices – **NUM** first in a new separate line and the next **NUM-1** lines are of the form (**a b wt**) where **a** is the parent of **b** and the edge has a weight of **wt**

```
// A C++ Program to generate test cases for
// an unweighted tree
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the maximum number of nodes of the tree
#define MAXNODE 20

// Define the maximum weight of edges
#define MAXWEIGHT 200

class Tree
{
    int V;      // No. of vertices

    // Pointer to an array containing adjacency lists
    list<int> *adj;

    // used by isCyclic()
    bool isCyclicUtil(int v, bool visited[], bool *rs);
public:
    Tree(int V);    // Constructor
    void addEdge(int v, int w);    // adds an edge
    void removeEdge(int v, int w); // removes an edge

    // returns true if there is a cycle in this graph
    bool isCyclic();
};

Tree::Tree(int V)
{
    this->V = V;
    adj = new list<int>[V];
```

```
}  
  
void Tree::addEdge(int v, int w)  
{  
    adj[v].push_back(w); // Add w to v's list.  
}  
  
void Tree::removeEdge(int v, int w)  
{  
    list<int>::iterator it;  
    for (it=adj[v].begin(); it!=adj[v].end(); it++)  
    {  
        if (*it == w)  
        {  
            adj[v].erase(it);  
            break;  
        }  
    }  
    return;  
}  
  
// This function is a variation of DFSUtil() in  
// https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/  
bool Tree::isCyclicUtil(int v, bool visited[], bool *recStack)  
{  
    if(visited[v] == false)  
    {  
        // Mark the current node as visited and part of  
        // recursion stack  
        visited[v] = true;  
        recStack[v] = true;  
  
        // Recur for all the vertices adjacent to this vertex  
        list<int>::iterator i;  
        for (i = adj[v].begin(); i != adj[v].end(); ++i)  
        {  
            if (!visited[*i] && isCyclicUtil(*i, visited,  
                                              recStack))  
                return true;  
            else if (recStack[*i])  
                return true;  
        }  
    }  
  
    // remove the vertex from recursion stack  
    recStack[v] = false;
```

```
    return false;
}

// Returns true if the graph contains a cycle, else false.
// This function is a variation of DFS() in
// https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
bool Tree::isCyclic()
{
    // Mark all the vertices as not visited and not part
    // of recursion stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for (int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle
    // in different DFS trees
    for (int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}

int main()
{
    set<pair<int, int> > container;
    set<pair<int, int> >::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Weighted_Tree1.in", "w", stdout);

    //For random values every time
    srand(time(NULL));

    int NUM;      // Number of Vertices/Nodes

    for (int i=1; i<=RUN; i++)
    {
        NUM = 1 + rand() % MAXNODE;

        // First print the number of vertices/nodes
        printf("%d\n", NUM);
        Tree t(NUM);
```

```
// Then print the edges of the form (a b wt)
// where 'a' is parent of 'b' and the edge has
// a weight of 'wt'
for (int j=1; j<=NUM-1; j++)
{
    int a = rand() % NUM;
    int b = rand() % NUM;
    pair<int, int> p = make_pair(a, b);

    t.addEdge(a, b);

    // Search for a random "new" edge everytime
    while (container.find(p) != container.end()
           || t.isCyclic() == true)
    {
        t.removeEdge(a, b);

        a = rand() % NUM;
        b = rand() % NUM;
        p = make_pair(a, b);
        t.addEdge(a, b);
    }
    container.insert(p);
}

for (it=container.begin(); it!=container.end(); ++it)
{
    int wt = 1 + rand() % MAXWEIGHT;
    printf("%d %d %d\n", it->first, it->second, wt);
}

container.clear();
printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Source

<https://www.geeksforgeeks.org/test-case-generation-set-3-unweighted-and-weighted-trees/>

Chapter 158

Test Case Generation | Set 4 (Random directed / undirected weighted and unweighted Graphs)

Test Case Generation | Set 4 (Random directed / undirected weighted and unweighted Graphs) - GeeksforGeeks

Generating Random Directed Unweighted Graphs

- Since this is a graph, the test data generation plan doesn't guarantee that a cycle gets formed or not.
- The number of edges – **NUMEDGE** is greater than zero and less than **NUM*(NUM-1)/2**, where NUM = Number of Vertices
- For each **RUN** we first print the number of vertices – **NUM** first in a new separate line and the next **NUMEDGE** lines are of the form (a b) where a is connected to b and the edge is directed from a to b (**a->b**)
- Each of the **NUMEDGE** lines will have distinct edges, for e.g – if **(1, 2)** is there in one of the **NUMEDGE** lines then it is guaranteed, that **(1, 2)** will not be there in the remaining **NUMEDGE-1** lines as this is a directed graph.

```
// A C++ Program to generate test cases for
// an unweighted directed graph
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5
```

```
// Define the maximum number of vertices of the graph
#define MAX_VERTICES 20

// Define the maximum number of edges
#define MAX_EDGES 200

int main()
{
    set<pair<int, int>> container;
    set<pair<int, int>>::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen ("Test_Cases_Directed_Unweighted_Graph.in", "w", stdout);

    //For random values every time
    srand(time(NULL));

    int NUM;      // Number of Vertices
    int NUMEDGE; // Number of Edges

    for (int i=1; i<=RUN; i++)
    {
        NUM = 1 + rand() % MAX_VERTICES;

        // Define the maximum number of edges of the graph
        // Since the most dense graph can have N*(N-1)/2 edges
        // where N = nnumber of vertices in the graph
        NUMEDGE = 1 + rand() % MAX_EDGES;

        while (NUMEDGE > NUM*(NUM-1)/2)
            NUMEDGE = 1 + rand() % MAX_EDGES;

        // First print the number of vertices and edges
        printf("%d %d\n", NUM, NUMEDGE);

        // Then print the edges of the form (a b)
        // where 'a' is connected to 'b'
        for (int j=1; j<=NUMEDGE; j++)
        {
            int a = 1 + rand() % NUM;
            int b = 1 + rand() % NUM;
            pair<int, int> p = make_pair(a, b);

            // Search for a random "new" edge everytime
            // Note - In a tree the edge (a, b) is same
            // as the edge (b, a)
```

```
        while (container.find(p) != container.end())
        {
            a = 1 + rand() % NUM;
            b = 1 + rand() % NUM;
            p = make_pair(a, b);
        }
        container.insert(p);
    }

    for (it=container.begin(); it!=container.end(); ++it)
        printf("%d %d\n", it->first, it->second);

    container.clear();
    printf("\n");

}
// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Generating Random Directed Weighted Graphs

- Since this is a graph, the test data generation plan doesn't guarantee that a cycle gets formed or not.
- The number of edges – NUMEDGE is greater than zero and less than **NUM*(NUM-1)/2**, where NUM = Number of Vertices
- For each **RUN** we first print the number of vertices – NUM first in a new separate line and the next NUMEDGE lines are of the form (a b wt) where a is connected to b and the edge is directed from a to b (**a->b**) and the edge has a weight of wt
- Each of the NUMEDGE lines will have distinct edges, for e.g – if (1, 2) is there in one of the NUMEDGE lines then it is guaranteed, that (1, 2) will not be there in the remaining **NUMEDGE-1** lines as this is a directed graph.

```
// A C++ Program to generate test cases for
// a weighted directed graph
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the maximum number of vertices of the graph
#define MAX_VERTICES 20
```

```
// Define the maximum number of edges
#define MAX_EDGES 200

// Define the maximum weight of edges
#define MAXWEIGHT 200

int main()
{
    set<pair<int, int>> container;
    set<pair<int, int>>::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases_Directed_Weighted_Graph.in",
    //         "w", stdout);

    // For random values every time
    srand(time(NULL));

    int NUM;      // Number of Vertices
    int NUMEDGE; // Number of Edges

    for (int i=1; i<=RUN; i++)
    {
        NUM = 1 + rand() % MAX_VERTICES;

        // Define the maximum number of edges of the graph
        // Since the most dense graph can have N*(N-1)/2 edges
        // where N = n number of vertices in the graph
        NUMEDGE = 1 + rand() % MAX_EDGES;

        while (NUMEDGE > NUM*(NUM-1)/2)
            NUMEDGE = 1 + rand() % MAX_EDGES;

        // First print the number of vertices and edges
        printf("%d %d\n", NUM, NUMEDGE);

        // Then print the edges of the form (a b)
        // where 'a' is connected to 'b'
        for (int j=1; j<=NUMEDGE; j++)
        {
            int a = 1 + rand() % NUM;
            int b = 1 + rand() % NUM;
            pair<int, int> p = make_pair(a, b);

            // Search for a random "new" edge every time
            // Note - In a tree the edge (a, b) is same
            // as the edge (b, a)
```

```

        while (container.find(p) != container.end())
        {
            a = 1 + rand() % NUM;
            b = 1 + rand() % NUM;
            p = make_pair(a, b);
        }
        container.insert(p);
    }

    for (it=container.begin(); it!=container.end(); ++it)
    {
        int wt = 1 + rand() % MAXWEIGHT;
        printf("%d %d %d\n", it->first, it->second, wt);
    }

    container.clear();
    printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}

```

Generating Random Undirected Unweighted Graphs

- Since this is a graph, the test data generation plan doesn't guarantee that a cycle gets formed or not.
- The number of edges – **NUMEDGE** is greater than zero and less than **NUM*(NUM-1)/2**, where **NUM** = Number of Vertices
- For each **RUN** we first print the number of vertices – **NUM** first in a new separate line and the next **NUMEDGE** lines are of the form **(a b)** where **a** is connected to **b**
- Each of the **NUMEDGE** lines will have distinct edges, for e.g – if **(1, 2)** is there in one of the **NUMEDGE** lines then it is guaranteed, that **(1, 2)** and **(2, 1)** both will not be there in the remaining **NUMEDGE-1** lines as this is an undirected graph.

```

// A C++ Program to generate test cases for
// an unweighted undirected graph
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

```

```
// Define the maximum number of vertices of the graph
#define MAX_VERTICES 20

// Define the maximum number of edges
#define MAX_EDGES 200

int main()
{
    set<pair<int, int>> container;
    set<pair<int, int>>::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases_Undirected_Unweighted_Graph.in",
    //         "w", stdout);

    // For random values every time
    srand(time(NULL));

    int NUM;      // Number of Vertices
    int NUMEDGE; // Number of Edges

    for (int i=1; i<=RUN; i++)
    {
        NUM = 1 + rand() % MAX_VERTICES;

        // Define the maximum number of edges of the graph
        // Since the most dense graph can have N*(N-1)/2 edges
        // where N = nnumber of vertices in the graph
        NUMEDGE = 1 + rand() % MAX_EDGES;

        while (NUMEDGE > NUM*(NUM-1)/2)
            NUMEDGE = 1 + rand() % MAX_EDGES;

        // First print the number of vertices and edges
        printf("%d %d\n", NUM, NUMEDGE);

        // Then print the edges of the form (a b)
        // where 'a' is connected to 'b'
        for (int j=1; j<=NUMEDGE; j++)
        {
            int a = rand() % NUM;
            int b = rand() % NUM;
            pair<int, int> p = make_pair(a, b);
            pair<int, int> reverse_p = make_pair(b, a);

            // Search for a random "new" edge everytime
            // Note - In a tree the edge (a, b) is same
```

```

// as the edge (b, a)
while (container.find(p) != container.end() ||
       container.find(reverse_p) != container.end())
{
    a = rand() % NUM;
    b = rand() % NUM;
    p = make_pair(a, b);
    reverse_p = make_pair(b, a);
}
container.insert(p);

for (it=container.begin(); it!=container.end(); ++it)
    printf("%d %d\n", it->first, it->second);

container.clear();
printf("\n");

}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}

```

Generating Random Undirected Weighted Graphs

- Since this is a graph, the test data generation plan doesn't guarantee that a cycle gets formed or not.
- The number of edges – **NUMEDGE** is greater than zero and less than **NUM*(NUM-1)/2**, where **NUM** = Number of Vertices
- For each **RUN** we first print the number of vertices – **NUM** first in a new separate line and the next **NUMEDGE** lines are of the form **(a b wt)** where a is connected to b and the edge has a weight of **wt**
- Each of the **NUMEDGE** lines will have distinct edges, for e.g – if **(1, 2)** is there in one of the **NUMEDGE** lines then it is guaranteed, that **(1, 2)** and **(2, 1)** both will not be there in the remaining **NUMEDGE-1** lines as this is an undirected graph.

```

// A C++ Program to generate test cases for
// an weighted undirected graph
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

```

```
// Define the maximum number of vertices of the graph
#define MAX_VERTICES 20

// Define the maximum number of edges
#define MAX_EDGES 200

// Define the maximum weight of edges
#define MAXWEIGHT 200

int main()
{
    set<pair<int, int>> container;
    set<pair<int, int>>::iterator it;

    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases_Undirected_Weighted_Graph.in",
    //         "w", stdout);

    //For random values every time
    srand(time(NULL));

    int NUM;      // Number of Vertices
    int NUMEDGE; // Number of Edges

    for (int i=1; i<=RUN; i++)
    {
        NUM = 1 + rand() % MAX_VERTICES;

        // Define the maximum number of edges of the graph
        // Since the most dense graph can have N*(N-1)/2 edges
        // where N = nnumber of vertices in the graph
        NUMEDGE = 1 + rand() % MAX_EDGES;

        while (NUMEDGE > NUM*(NUM-1)/2)
            NUMEDGE = 1 + rand() % MAX_EDGES;

        // First print the number of vertices and edges
        printf("%d %d\n", NUM, NUMEDGE);

        // Then print the edges of the form (a b)
        // where 'a' is connected to 'b'
        for (int j=1; j<=NUMEDGE; j++)
        {
            int a = rand() % NUM;
            int b = rand() % NUM;
            pair<int, int> p = make_pair(a, b);
            container.insert(p);
        }
    }
}
```

```
pair<int, int> reverse_p = make_pair(b, a);

// Search for a random "new" edge everytime
// Note - In a tree the edge (a, b) is same
// as the edge (b, a)
while (container.find(p) != container.end() ||
       container.find(reverse_p) != container.end())
{
    a = rand() % NUM;
    b = rand() % NUM;
    p = make_pair(a, b);
    reverse_p = make_pair(b, a);
}
container.insert(p);

for (it=container.begin(); it!=container.end(); ++it)
{
    int wt = 1 + rand() % MAXWEIGHT;
    printf("%d %d %d\n", it->first, it->second, wt);
}

container.clear();
printf("\n");

}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Source

<https://www.geeksforgeeks.org/test-case-generation-set-4-random-directed-undirected-weighted-and-unweighted-graphs/>

Chapter 159

Test Case Generation | Set 5 (Generating random Sorted Arrays and Palindromes)

Test Case Generation | Set 5 (Generating random Sorted Arrays and Palindromes) - GeeksforGeeks

Generating Random **Sorted Arrays**

We store the random array elements in an array and then sort it and print it.

```
// A C++ Program to generate test cases for
// array filled with random numbers
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the range of the test data generated
#define MAX 100000

// Define the maximum number of array elements
#define MAXNUM 100

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases_Random_Sorted_Array.in",
    //         "w", stdout);
```

```
// For random values every time
srand(time(NULL));

int NUM;      // Number of array elements

for (int i=1; i<=RUN; i++)
{
    int arr[MAXNUM];

    NUM = 1 + rand() % MAXNUM;

    // First print the number of array elements
    printf("%d\n", NUM);

    // Then print the array elements separated by
    // space
    for (int j=0; j<NUM; j++)
        arr[j] = rand() % MAX;

    // Sort the generated random array
    sort (arr, arr + NUM);

    // Print the sorted random array
    for (int j=0; j<NUM; j++)
        printf("%d ", arr[j]);

    printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

Generating Random **Palindromes**

- The test case generation plan generates odd as well as even length palindromes.
- The test case generation plan uses one of the most under-rated data structure- **Deque**
- Since a palindrome is read the same from left as well as right so we simply put the same random characters on both the left side (done using **push_front()**) and the right side (done using **push_back()**)

```
// A C++ Program to generate test cases for
// random strings
```

```
#include<bits/stdc++.h>
using namespace std;

// Define the number of runs for the test data
// generated
#define RUN 5

// Define the range of the test data generated
// Here it is 'a' to 'z'
#define MAX 25

// Define the maximum length of string
#define MAXLEN 50

int main()
{
    // Uncomment the below line to store
    // the test data in a file
    // freopen("Test_Cases_Palindrome.in", "w",
    //         stdout);

    // For random values every time
    srand(time(NULL));

    // A container for storing the palindromes
    deque<char> container;
    deque<char>::iterator it;

    int LEN;      // Length of string

    for (int i=1; i<=RUN; i++)
    {
        LEN = 1 + rand() % MAXLEN;

        // First print the length of string
        printf("%d\n", LEN);

        // If it is an odd-length palindrome
        if (LEN % 2)
            container.push_back('a' + rand() % MAX);

        // Then print the characters of the palindromic
        // string
        for (int j=1; j<=LEN/2; j++)
        {
            char ch = 'a' + rand() % MAX;
            container.push_back(ch);
            container.push_front(ch);
        }
    }
}
```

```
}

for (it=container.begin(); it!=container.end(); ++it)
    printf("%c",*it);

container.clear();
printf("\n");
}

// Uncomment the below line to store
// the test data in a file
// fclose(stdout);
return(0);
}
```

References : –

<http://spojtoolkit.com/TestCaseGenerator/>

Source

<https://www.geeksforgeeks.org/test-case-generation-set-5-generating-random-sorted-arrays-palindromes/>

Chapter 160

The Google Foo Bar Challenge

The Google Foo Bar Challenge - GeeksforGeeks

The Google Foo bar challenge has been known for the last 5 years or more as a secret process of hiring developers and programmers all over the world.

It is a secret process and the challenge consists of coding challenges of increasing difficulty as you go along.

Chapter 161

My Experience with the Google Foo Bar Challenge

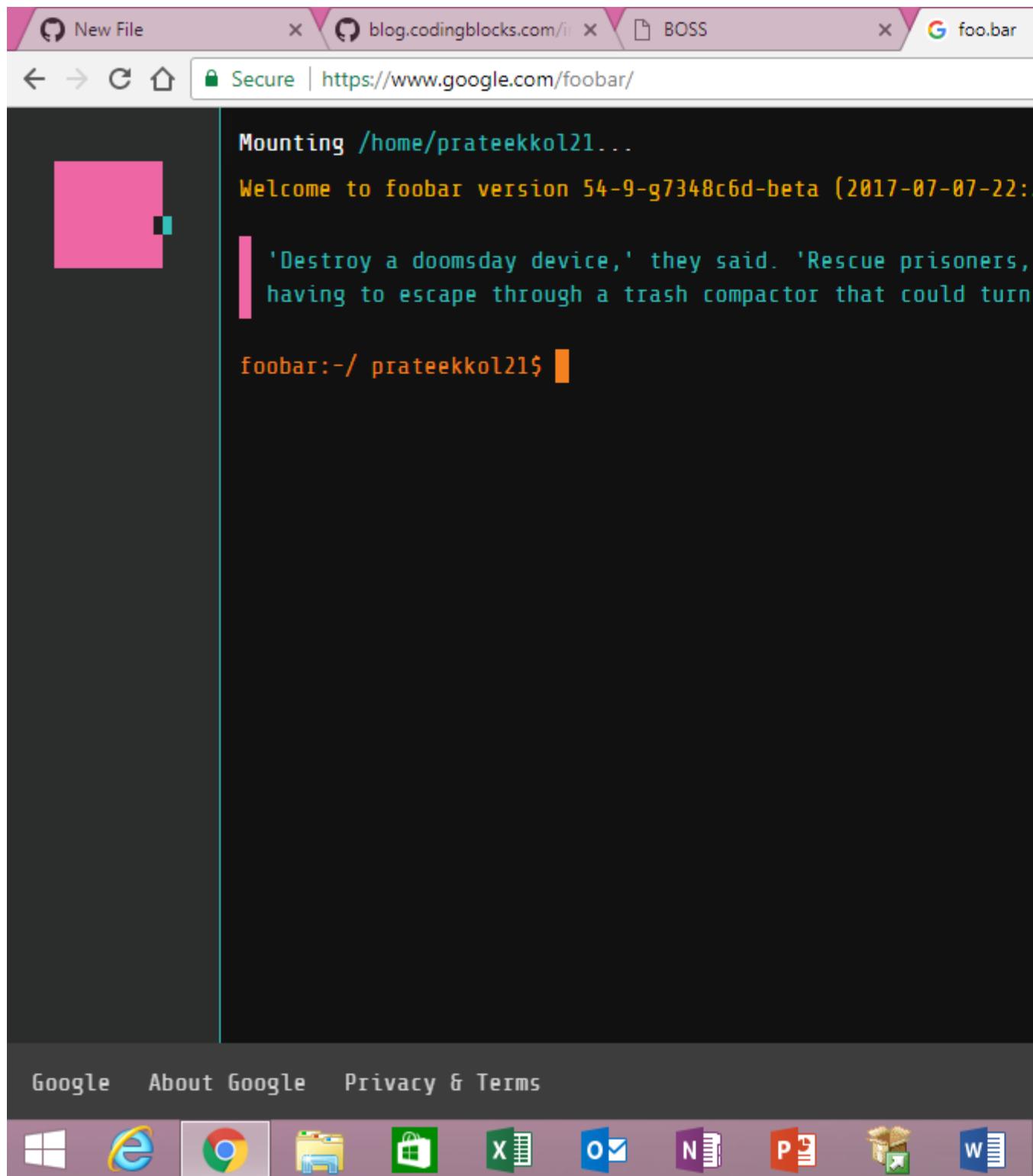
The Google foo bar page is not accessible to everyone. Google has a list of what the user goes searching for and if it finds it relevant to programming, it gives the user an opportunity to participate in the foo bar challenge.

The message reads like this.

**You are speaking our language.
Up for a challenge!**

In my case, it didn't go the traditional way. During the ICC matches a few months ago, Google had updated their doodle with a small cricket game with players as ants. Just out of curiosity, I went on for inspecting the page, when suddenly there was a comment under a 'li' tag.

"Up for a challenge. You are invited". And there was a link and I opened it and it redirected me to a new page 'Foo.bar'



Its kind of like a linux console where you get to solve the problems one by one upon opening a new problem directory.

Chapter 162

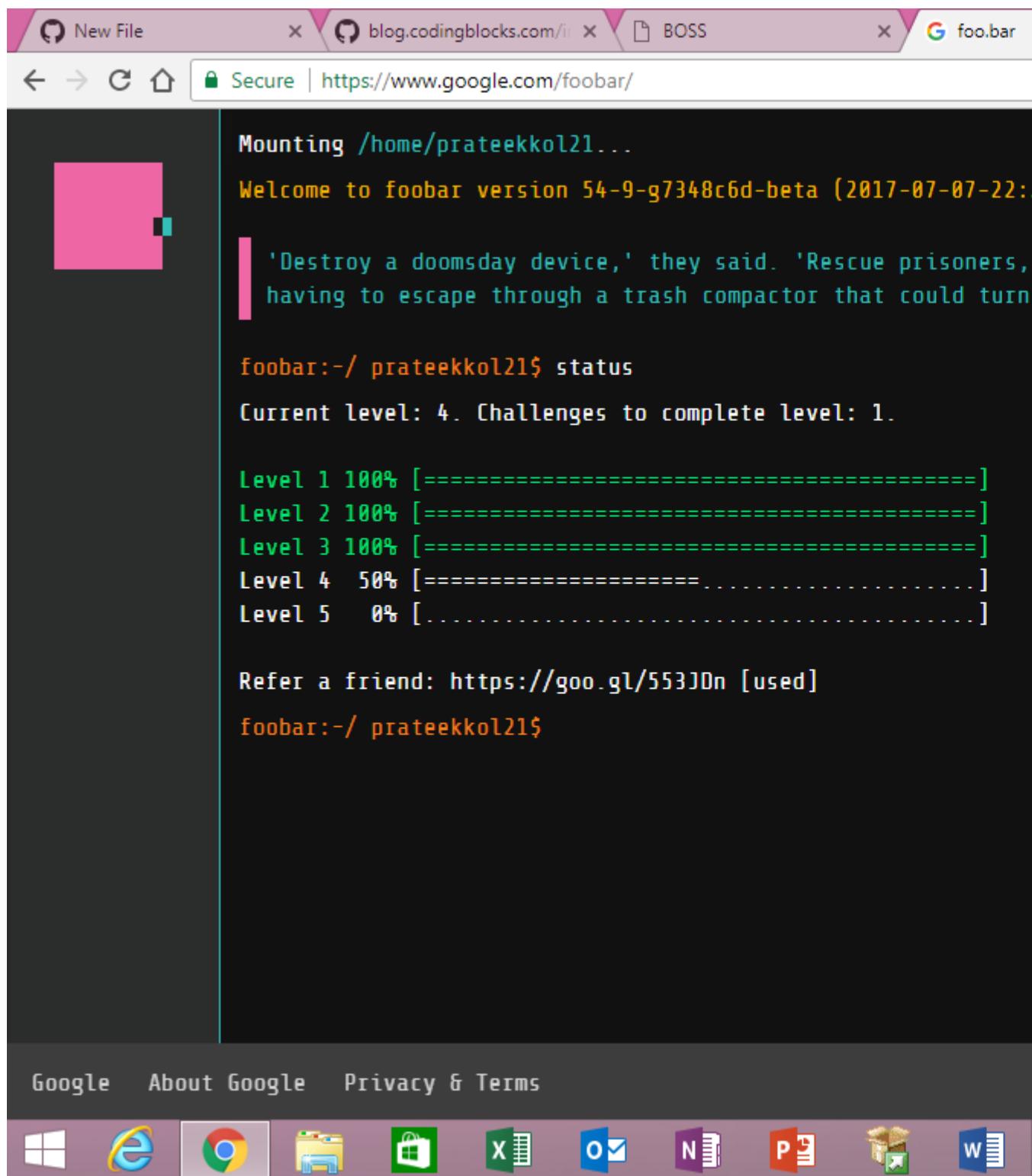
The Challenge

The challenge consists of 5 levels consisting of algorithm problems. I won't share the problems, neither the solutions as it would be unfair.

But it was really a great experience so far.

The first few levels were relatively easy, but as the levels peaked up, the difficulty gained heights.

Currently I am on level 4 and just have one more label to go.



Upon completing level 3 , I had to submit my personal details with a potential recruiter for a future interview.

The Google Foobar is presumably still hiring and its needed to submit solution either in Python or Java.

I don't know about the future upcomings for this, but to be honest I really enjoyed the challenges.

Looking forward to solve more.

Disclaimer : The information provided in this article is not found on any of the Google sites. However there is a [quora threads](#) about this.

Improved By : [Prateek Chanda](#)

Source

<https://www.geeksforgeeks.org/google-foo-bar-challenge/>

Chapter 163

The painter's partition problem | Set 2

The painter's partition problem | Set 2 - GeeksforGeeks

We have to paint n boards of length $\{A_1, A_2, \dots, A_n\}$. There are k painters available and each takes 1 unit time to paint 1 unit of board. The problem is to find the minimum time to get this job done under the constraints that any painter will only paint continuous sections of boards, say board $\{2, 3, 4\}$ or only board $\{1\}$ or nothing but not board $\{2, 4, 5\}$.

Examples :

Input : k = 2, A = {10, 10, 10, 10}

Output : 20.

Here we can divide the boards into 2 equal sized partitions, so each painter gets 20 units of board and the total time taken is 20.

Input : k = 2, A = {10, 20, 30, 40}

Output : 60.

Here we can divide first 3 boards for one painter and the last board for second painter.

In the [previous post](#) we discussed a dynamic programming based approach having time

complexity of $O(n^2k)$ and $O(n^2k)$ extra space.

In this post we will look into a more efficient approach using binary search. We know that the invariant of binary search has two main parts:

- * the target value would always be in the searching range.
- * the searching range will decrease in each loop so that the termination can be reached.

We also know that the values in this range must be in sorted order. Here our target value is the maximum sum of a contiguous section in the optimal allocation of boards. Now how can we apply binary search for this? We can fix the possible low to high range for the target value and narrow down our search to get the optimal allocation.

We can see that the highest possible value in this range is the sum of all the elements in the array and this happens when we allot 1 painter all the sections of the board. The lowest possible value of this range is the maximum value of the array max, as in this allocation we can allot max to one painter and divide the other sections such

that the cost of them is less than or equal to max and as close as possible to max. Now if we consider we use x painters in the above scenarios, it is obvious that as the value in the range increases, the value of x decreases and vice-versa. From this we can find the target value when $x=k$ and use a helper function to find x, the minimum number of painters required when the maximum length of section a painter can paint is given.

C++

```
// CPP program for painter's partition problem
#include <iostream>
using namespace std;

// return the maximum element from the array
int getMax(int arr[], int n)
{
    int max = INT_MIN;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
int numberOfWorkers(int arr[], int n, int maxlen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];
        if (total > maxlen) {
            numPainters++;
            total = arr[i];
        }
    }
    return numPainters;
}
```

```

        if (total > maxLen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters = number_of_painters(arr, n, mid);

        // find better optimum in lower half
        // here mid is included because we
        // may not get anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper half
        // here mid is excluded because it gives
        // required Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
    return lo;
}

// driver function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << partition(arr, n, k) << endl;
    return 0;
}

```

Java

```
// Java Program for painter's partition problem
import java.util.*;
import java.io.*;

class GFG
{
// return the maximum element from the array
static int getMax(int arr[], int n)
{
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
static int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
static int numberOfPainters(int arr[], int n, int maxlen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];

        if (total > maxlen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

static int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
```

```
int hi = getSum(arr, n);

while (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    int requiredPainters = numberOfWorkers(arr, n, mid);

    // find better optimum in lower half
    // here mid is included because we
    // may not get anything better
    if (requiredPainters <= k)
        hi = mid;

    // find better optimum in upper half
    // here mid is excluded because it gives
    // required Painters > k, which is invalid
    else
        lo = mid + 1;
}

// required
return lo;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    // Calculate size of array.
    int n = arr.length;
    int k = 3;
    System.out.println(partition(arr, n, k));
}
}

// This code is contributed by Sahil_Bansall
```

C#

```
// C# Program for painter's
// partition problem
using System;

class GFG
{

    // return the maximum
    // element from the array
```

```
static int getMax(int []arr, int n)
{
    int max = int.MinValue;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the
// elements in the array
static int getSum(int []arr, int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required
// painters for given
// maxlen which is the
// maximum length a painter
// can paint
static int numberOfPainters(int []arr,
                            int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++)
    {
        total += arr[i];

        if (total > maxLen)
        {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

static int partition(int []arr,
                    int n, int k)
{
```

```
int lo = getMax(arr, n);
int hi = getSum(arr, n);

while (lo < hi)
{
    int mid = lo + (hi - lo) / 2;
    int requiredPainters =
        numberOfPainters(arr, n, mid);

    // find better optimum in lower
    // half here mid is included
    // because we may not get
    // anything better
    if (requiredPainters <= k)
        hi = mid;

    // find better optimum in upper
    // half here mid is excluded
    // because it gives required
    // Painters > k, which is invalid
    else
        lo = mid + 1;
}

// required
return lo;
}

// Driver code
static public void Main ()
{
    int []arr = {1, 2, 3, 4, 5,
                6, 7, 8, 9};

    // Calculate size of array.
    int n = arr.Length;
    int k = 3;
    Console.WriteLine(partition(arr, n, k));
}
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP program for painter's
// partition problem
```

```
// return the maximum
// element from the array
function getMax($arr, $n)
{
    $max = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] > $max)
            $max = $arr[$i];
    return $max;
}

// return the sum of the
// elements in the array
function getSum($arr, $n)
{
    $total = 0;
    for ($i = 0; $i < $n; $i++)
        $total += $arr[$i];
    return $total;
}

// find minimum required painters
// for given maxlen which is the
// maximum length a painter can paint
function numberOfWorkers($arr, $n,
                         $maxLen)
{
    $total = 0; $numWorkers = 1;

    for ($i = 0; $i < $n; $i++)
    {
        $total += $arr[$i];

        if ($total > $maxLen)
        {

            // for next count
            $total = $arr[$i];
            $numWorkers++;
        }
    }

    return $numWorkers;
}

function partition($arr, $n, $k)
{
```

```
$lo = getMax($arr, $n);
$hi = getSum($arr, $n);

while ($lo < $hi)
{
    $mid = $lo + ($hi - $lo) / 2;
    $requiredPainters =
        numberOfPainters($arr,
                          $n, $mid);

    // find better optimum in
    // lower half here mid is
    // included because we may
    // not get anything better
    if ($requiredPainters <= $k)
        $hi = $mid;

    // find better optimum in
    // upper half here mid is
    // excluded because it
    // gives required Painters > k,
    // which is invalid
    else
        $lo = $mid + 1;
}

// required
return floor($lo);
}

// Driver Code
$arr = array(1, 2, 3,
             4, 5, 6,
             7, 8, 9);
$n = sizeof($arr);
$k = 3;

echo partition($arr, $n, $k) , "\n";

// This code is contributed by ajit
?>
```

Output :

17

For better understanding, please trace the example given in the program in pen and paper.

The time complexity of the above approach is $O(n^2 \log(\text{sum}(arr)))$.

References:

<https://articles.leetcode.com/the-painters-partition-problem-part-ii/>

<https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/>

Asked in: Google, Codenation.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/painters-partition-problem-set-2/>

Chapter 164

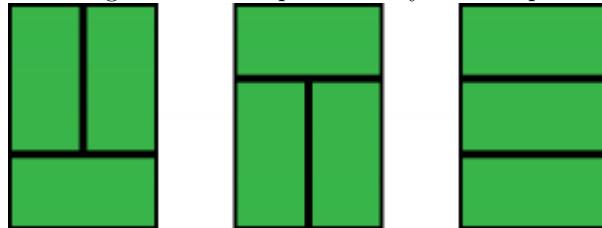
Tiling with Dominoes

Tiling with Dominoes - GeeksforGeeks

Given a $3 \times n$ board, find the number of ways to fill it with 2×1 dominoes.

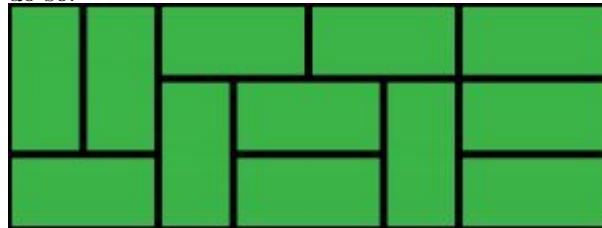
Example 1

Following are all the **3** possible ways to fill up a 3×2 board.



Example 2

Here is one possible way of filling a 3×8 board. You have to find all the possible ways to do so.



Examples :

Input : 2

Output : 3

Input : 8

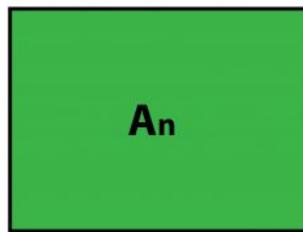
Output : 153

Input : 12
Output : 2131

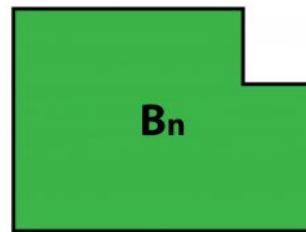
Defining Subproblems:

At any point while filling the board, there are three possible states that the last column can be in:

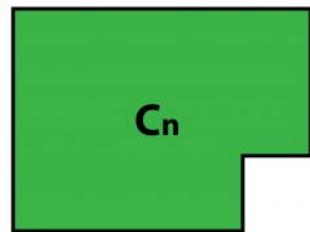
Possible states of the last column



Completely Filled



Top Corner Empty



Bottom Corner Empty

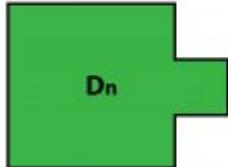
A_n = No. of ways to completely fill a $3 \times n$ board. (We need to find this)

B_n = No. of ways to fill a $3 \times n$ board with top corner in last column not filled.

C_n = No. of ways to fill a $3 \times n$ board with bottom corner in last column not filled.

Note: The following states are impossible to reach:

Following States are NOT possible



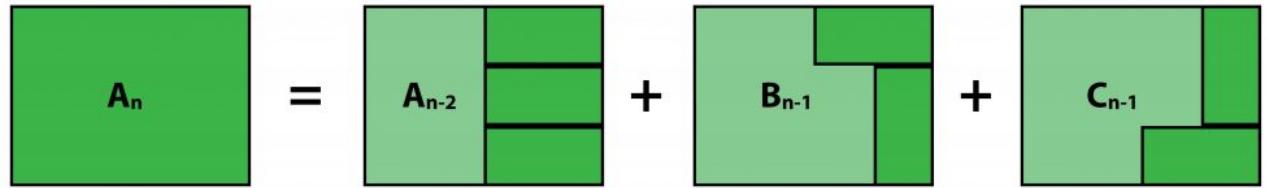
Finding Recurrences

Note: Even though B_n and C_n are different states, they will be equal for same ' n '. i.e

$$B_n = C_n$$

Hence, we only need to calculate one of them.

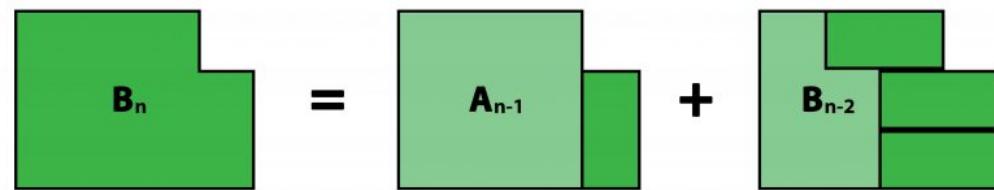
Calculating A_n :



$$A_n = A_{n-2} + B_{n-1} + C_{n-1}$$

$$A_n = A_{n-2} + 2 * (B_{n-1})$$

Calculating B_n:



$$B_n = A_{n-1} + B_{n-2}$$

Final Recursive Relations are:

Base Cases:

C++

```
// C++ program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
#include <iostream>
using namespace std;

int countWays(int n)
{
    int A[n + 1], B[n + 1];
```

```
A[0] = 1, A[1] = 0, B[0] = 0, B[1] = 1;
for (int i = 2; i <= n; i++) {
    A[i] = A[i - 2] + 2 * B[i - 1];
    B[i] = A[i - 1] + B[i - 2];
}

return A[n];
}

int main()
{
    int n = 8;
    cout << countWays(n);
    return 0;
}
```

Java

```
// Java program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
import java.io.*;

class GFG {

    static int countWays(int n)
    {
        int []A = new int[n+1];
        int []B = new int[n+1];
        A[0] = 1; A[1] = 0;
        B[0] = 0; B[1] = 1;
        for (int i = 2; i <= n; i++)
        {
            A[i] = A[i - 2] + 2 * B[i - 1];
            B[i] = A[i - 1] + B[i - 2];
        }

        return A[n];
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 8;
        System.out.println(countWays(n));
    }
}

// This code is contributed by anuj_67.
```

Python 3

```
# Python 3 program to find no. of ways
# to fill a 3xn board with 2x1 dominoes.

def countWays(n):

    A = [0] * (n + 1)
    B = [0] * (n + 1)
    A[0] = 1
    A[1] = 0
    B[0] = 0
    B[1] = 1
    for i in range(2, n+1):
        A[i] = A[i - 2] + 2 * B[i - 1]
        B[i] = A[i - 1] + B[i - 2]

    return A[n]

n = 8
print(countWays(n))

# This code is contributed by Smitha
```

C#

```
// C# program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
using System;

class GFG {

    static int countWays(int n)
    {
        int []A = new int[n+1];
        int []B = new int[n+1];
        A[0] = 1; A[1] = 0;
        B[0] = 0; B[1] = 1;
        for (int i = 2; i <= n; i++)
        {
            A[i] = A[i - 2] + 2 * B[i - 1];
            B[i] = A[i - 1] + B[i - 2];
        }

        return A[n];
    }
}
```

```
// Driver code
public static void Main ()
{
    int n = 8;
    Console.WriteLine(countWays(n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.

function countWays($n)
{
    $A = array();
    $B = array();
    $A[0] = 1; $A[1] = 0;
    $B[0] = 0; $B[1] = 1;
    for ( $i = 2; $i <= $n; $i++)
    {
        $A[$i] = $A[$i - 2] + 2 *
                  $B[$i - 1];
        $B[$i] = $A[$i - 1] +
                  $B[$i - 2];
    }

    return $A[$n];
}

// Driver Code
$n = 8;
echo countWays($n);

// This code is contributed by anuj_67.
?>
```

Output :

153

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/tiling-with-dominoes/>

Chapter 165

Tips and Tricks for Competitive Programmers | Set 1 (For Beginners)

Tips and Tricks for Competitive Programmers | Set 1 (For Beginners) - GeeksforGeeks

This article is a collection of various tips that would help beginners of Competitive programming to get an insight of things that should or shouldn't be done.

Competitive programming can only be improved by “PRACTICE, PRACTICE AND PRACTICE”. Try to solve as many questions you can solve on sites like practice.geeksforgeeks.org. This will enhance your mind to think more on algorithms.

Start with the beginner section, and when you feel comfortable with that, move on to higher level i.e. easy, medium, and hard and so on. Try to attempt all the questions yourself and don't see the solution before attempting it. Don't feel demotivated when you get wrong answers, it's just part of the learning. The more you practice the more you learn. Just be passionate about coding and practice.

Few Days before you begin:

1. **Learn -Practice-Repeat** -Try to learn a new concept on a daily basis. Solve questions daily, one or two if not more!! After going through a new algorithm or technique, we should immediately search for its applications and attempt problems. Like if you learn dynamic programming, try to finish up all its problems. Adapt the habit of reading which most of the youngsters don't have nowadays.
2. **Write before coding**– Implement all algorithms yourself rather than copying from someone else. Make yourself written notes while studying these concepts. Mathematics is great area to start competitive programming.
3. **Getting Edgy**- During practice always solve that problem that is just at the edge of your knowledge i.e., you don't exactly know how to solve the problem but you know what you should know to solve that problem. For example, you look at the problem

and you can tell that it's a simple graph problem but you do not know anything about graph.

4. **Trees, Graphs, Algos**– Make sure you are thorough with the concepts of trees, graphs and **important** algorithms as there is at least one question making use of their applications in every contest or a company hiring round.
5. **Short is sweet**– Long contest is good for learning but try to take part in more and more short contests. Short contest is the real competitive programming. We should make it a must habit to spend some short time during peak hours in a programming forum where top coders usually hangout sharing their insights and often get into discussions.
6. **Complexity is Complex**– Do not be obsessed with lower and lower execution time. Do not waste time on over-optimizing your solution. If the solution is accepted, move on to next problem. First just get into the habit of coding daily and then worry about complexities.
7. **Hard must come**– Some people say Stick to one website for practice while others believe you must taste all bunches. Whatever you decide , slowly but surely start solving**harder** problems.
8. **Target Job**– If you are frequently participating in the contests which are meant for jobs then make sure to read all **previous questions**, algorithms and related stuff to cut short your efforts as well as selection time.You must attempt previous **company**questions too.

Last day before the contest especially If you are attempting the contest for Getting Hired

9. Don't look for new problems because that may create panic in your mind.
10. Take sufficient amount of sleep the night before. Keep your mind relaxed and stay stress free

During the contest

1. **Be Attentive** – Most of the programmers when see a new question, they will hurry in typing it on system before pre planning or before writing logic to crack that task. Sometimes they will stick at a point in between of typing code in system and might need to start coding again. If we avoid typing in system before cracking the logic it will be helpful to save time. One should start with:
 - (a) Reading the problem statement at least twice
 - (b) Analyzing the problem statement
 - (c) Input output pattern should be kept in mind before submission and read problem many times to understand concept behind problem.
 - (d) Use pen and paper to develop the logic and then code
 - (e) Read the instructions of the contest carefully (**Time limit, Meaning of various symbols used in the contest page, Number of submissions allowed etc.**)
2. **Clock is ticking**– Keep an eye on the clock .If you are unable to solve a particular question, you very well have the option to go to next.

3. **Test the test cases**– If your code is not accepted, then go through your code again and check about variable declaration, complexity of the code, and try checking your code for multiple numbers of test cases.

After the contest

1. **Editorials are MUST**– After submission; even if your code is accepted then just don't jump on to the next question. Try to read [editorial](#) of the question , this will help you know better and efficient solution of that question.
2. **Geeks around may know better**- Examining codes written by other eminent coders will reveal great insights (if it is allowed). Even reviewing other's solution to a problem we have solved might expose some of the unique features of the problem and aids us in viewing the same problem from a different point of view. The important point here is that- you may come across different algorithms that are used to solve questions, learn those algorithms and make sure that you understand them.
3. **Practice**– Don't worry if you are unable to solve the questions there, it just means that you need more practice.
4. **Past teaches future**– It is a good practice to stick to a problem which we are unable to solve for at least 2 days. On reviewing the solution, we will understand where we deviated from the correct path and would aid our thinking process in future attempts. We should make a note of the problems which we were unable to solve and hence, we went for the solution. We should make sure to review the same problem after a couple of weeks and attempt to solve it entirely.
5. **Time is precious**– Preparation for anything is very important be it for a contest day, an exams or a project submission, which student mostly fail to do. Preparing at the last moment often fail short of the expectations. Give enough time to go through algorithms, sample problems and work upon your own strengths and weaknesses.

Happy Coding!!

This article is exclusively drafted by contributions of our Campus Geeks- **Rahul Agarwal, Aditya Chatterjee, Shubham Singh Rajput, Vineet Sethia, Saiteja Reddy, Shaily Seth, Mudit Maheshwari and Ajay Jain**.

[Tips and Tricks for Competitive Programmers | Set 2 \(Language to be used for Competitive Programming\)](#)

Source

<https://www.geeksforgeeks.org/tips-and-tricks-for-competitive-programmers-set-1-for-beginners/>

Chapter 166

Tips and Tricks for Competitive Programmers | Set 2 (Language to be used for Competitive Programming)

Tips and Tricks for Competitive Programmers | Set 2 (Language to be used for Competitive Programming) - GeeksforGeeks

This is a question asked quite often as in which language should be preferred to be efficient in competitive programming. It is something one should not worry about as what matters is the logic not the language. Most of the languages are more or less same ,but till now the most proffered language is C++ and here are the reasons.

Python

- **Simple and Easy:** Python is simple, easy to write (we need to type less), and has a huge collection of modules with almost all the functions you can imagine.
- **Data Types:** Python is generally preferred as [it does not have any upper limit on the memory of integers](#). Also, one does not need to specify which data type it is and things like this make it easier to code but at the same time makes it difficult to compile (in reference to time taken for compilation).
- **Slow in Execution:** Python programs are generally slower compared to Java (See [this](#)). Python is pretty much ruled out in the starting itself due to it's high time of execution.
- **Python is not allowed everywhere:** Python is not allowed in contests in various popular online competitive programming portals.

Now we are mostly left with Java, C, C++, now here it becomes difficult to compare and is mostly dependent on the user but let's discuss the good and the bad points of each of the them.

Java

- **STL vs containers:** STL in C++ is really well designed whereas some people love Java Containers more than anything. There are few situations where STL doesn't have a direct solution. For example priority_queue in STL doesn't support decrease key operation which is required for implementations of Dijkstra's shortest Path Algorithm and Prim's algorithm
- **Exception Handling in Java is incomparable:** Java code provides a stronger exception handling versus C++. For instance it's easier to trace an ArrayIndexOutOfBoundsException or a segmentation fault in Java. C++/C might give you wrong answers but Java is surely reliable in this context.
- **Time Limit Exceeds :** You might get TLE due to Java being slightly slower on the time limit side (Specially in SPOJ), Codeforces.
- **Big integer and Regular Expressions:** Java has some few advantages with respect to programming contests. BigInteger, Regular Expressions and geometry library are some of them.

Now let us proceed to C++.

C++ and C

- **C++ speed is comparable to C:** Many C programs are valid C++ programs as well – and such C programs run at identical speed when compiled
- **C++ does not force object oriented programming:** The C++ language contains some language extensions that facilitate object oriented programming and C++ does not force object oriented design anywhere – it merely allows it.
- **Parametrized types** The template keyword allows the programmer to write generic (type-agnostic) implementations of algorithms. Where in C, one could write a generic list implementation with an element like:

```
struct element_t
{
    struct element_t *next, *prev;
    void *element;
};
```

C++ allows one to write something like:

```
template <typename T>
struct element_t
{
```

```
    element_t<T> *next, *prev;
    T element;
};
```

- **A bigger standard library:** C++ allows the full use of the C standard library as well as C++ includes its own libraries including [Standard Template Library](#). The STL contains a number of useful templates, like the sort routine above. It includes useful common data structures such as lists, maps, sets, etc. Like the sort routine, the other STL routines and data structures are “tailored” to the specific needs the programmer has – all the programmer has to do is fill in the types.
For example, if we need to implement Binary Search for a problem, we will have to write our own function whereas in C++ [Binary Search STL routine](#) is defined as

```
binary_search(startaddress, endaddress, valuetofind)
```

C++ vs Java

- **Java Codes are longer** A programmer needs to write more when programming in Java
- **Java is verbose:** In C++, Input Output is simpler by just writing scanf/printf. In Java, you need the BufferedReader class, which again is tedious.
- **C++ STL vs Java Containers:** Most of the programmers find it easier to use STLs.
- **C++ is more Popular:** Be it the origin year or the comfort of use, but C++ outstands Java in terms of number of users that use the language.
- **C++ saves time :** It's well-known fact that Java is slower than C++. We generally need to compile and run programs many times to test them. It takes relatively much less time in C++. Therefore, in limited time contests, our time can be saved.

Wrapping it up, C++ is till date most preferred language followed by Java when it comes to programming contests but you should always choose a language you are comfortable with. Being CONFIDENT in any language is most important. Never choose a language which you have “just learnt” as it will be difficult to express yourself in that language.

For topics, start with cakewalk questions and then move to ad-hoc questions, then cover standard [algorithms](#) and [data structure](#). Finally learn to optimize your code. All this time emphasis at learning maths, for [mathematical algos](#) are important part of excelling Competitive programming.

Happy Coding!!

Reference: http://unthought.net/c%2B%2B/c_vs_c%2B%2B.html

This article is exclusively drafted by contributions of our Campus Geeks- **Rahul Agarwal, Aditya Chatterjee, Shubham Singh Rajput, Vineet Sethia, Saiteja Reddy, Shaily Seth, Mudit Maheshwari, Ajay Jain and Ruchir Garg.**

[Tips and Tricks for Competitive Programmers | Set 1 \(For Beginners\)](#)

If you are new to competitive programming, [this article](#) may help you to write your first code.

Source

<https://www.geeksforgeeks.org/tips-and-tricks-for-competitive-programmers-set-2-which-language-should-be-used->

Chapter 167

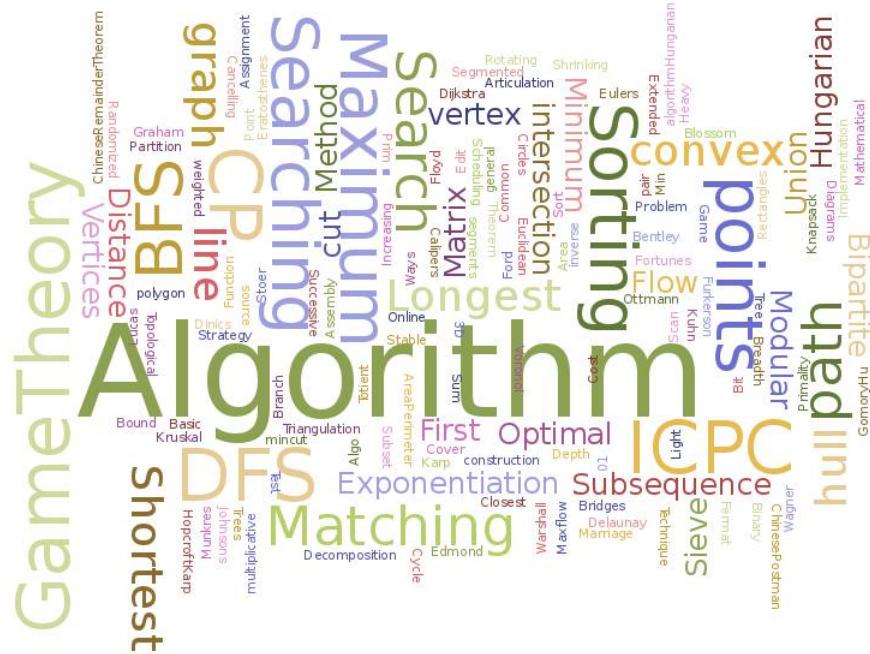
Top 10 Algorithms and Data Structures for Competitive Programming

Top 10 Algorithms and Data Structures for Competitive Programming - GeeksforGeeks

In this post “Important top 10 algorithms and data structures for competitive coding “.

Topics :

1. [Graph algorithms](#)
2. [Dynamic programming](#)
3. [Searching and Sorting:](#)
4. [Number theory and Other Mathematical](#)
5. [Geometrical and Network Flow Algorithms](#)
6. [Data Structures](#)



The below links cover all most important algorithms and data structure topics:

Graph Algorithms

1. Breadth First Search (BFS)
 2. Depth First Search (DFS)
 3. Shortest Path from source to all vertices **Dijkstra**
 4. Shortest Path from every vertex to every other vertex **Floyd Warshall**
 5. Minimum Spanning tree **Prim**
 6. Minimum Spanning tree **Kruskal**
 7. Topological Sort
 8. Johnson's algorithm
 9. Articulation Points (or Cut Vertices) in a Graph
 10. Bridges in a graph

All Graph Algorithms

Dynamic Programming

1. Longest Common Subsequence
 2. Longest Increasing Subsequence
 3. Edit Distance
 4. Minimum Partition

- 5. Ways to Cover a Distance
- 6. Longest Path In Matrix
- 7. Subset Sum Problem
- 8. Optimal Strategy for a Game
- 9. 0-1 Knapsack Problem
- 10. Assembly Line Scheduling

[All DP Algorithms](#)

Searching And Sorting

- 1. Binary Search
- 2. Quick Sort
- 3. Merge Sort
- 4. Order Statistics
- 5. KMP algorithm
- 6. Rabin karp
- 7. Z's algorithm
- 8. Aho Corasick String Matching
- 9. Counting Sort
- 10. Manacher's algorithm: [Part 1](#), [Part 2](#) and [Part 3](#)

All Articles on [Searching](#), [Sorting](#) and [Pattern Searching](#).

Number theory and Other Mathematical Prime Numbers and Prime Factorization

- 1. Primality Test | Set 1 (Introduction and School Method)
- 2. Primality Test | Set 2 (Fermat Method)
- 3. Primality Test | Set 3 (Miller–Rabin)
- 4. Sieve of Eratosthenes
- 5. Segmented Sieve
- 6. Wilson's Theorem
- 7. Prime Factorisation
- 8. Pollard's rho algorithm

Modulo Arithmetic Algorithms

- 1. Basic and Extended Euclidean algorithms
- 2. Euler's Totient Function
- 3. Modular Exponentiation
- 4. Modular Multiplicative Inverse

5. Chinese remainder theorem Introduction
6. Chinese remainder theorem and Modulo Inverse Implementation
7. $nCr \% m$ and [this](#).

Miscellaneous:

1. Counting Inversions
2. Counting Inversions using BIT
3. logarithmic exponentiation
4. Square root of an integer
5. Heavy light Decomposition, [this](#) and [this](#)
6. Matrix Rank
7. Gaussian Elimination to Solve Linear Equations
8. Hungarian algorithm
9. Link cut
10. Mo's algorithm and [this](#)
11. Factorial of a large number in C++
12. Factorial of a large number in Java+
13. Russian Peasant Multiplication
14. Catalan Number

[All Articles on Mathematical Algorithms](#)

Geometrical and Network Flow Algorithms

1. Convex Hull
2. Graham Scan
3. Line Intersection
4. Interval Tree
5. Matrix Exponentiation and [this](#)
6. Maxflow Ford Fulkerson Algo and Edmond Karp Implementation
7. Min cut
8. Stable Marriage Problem
9. Hopcroft-Karp Algorithm for Maximum Matching
10. Dinic's algo and [e-maxx](#)

[All Articles on Geometric Algorithms](#)

Data Structures

1. Binary Indexed Tree or Fenwick tree
2. Segment Tree (RMQ, Range Sum and Lazy Propagation)
3. K-D tree (See [insert](#), [minimum](#) and [delete](#))
4. Union Find Disjoint Set (Cycle Detection and By Rank and Path Compression)
5. Tries

6. Suffix array ([this](#), [this](#) and [this](#))
7. Sparse table
8. Suffix automata
9. Suffix automata II
10. LCA and RMQ

[All Articles on Advanced Data Structures.](#)

How to Begin?

Please see [How to begin with Competitive Programming?](#)

How to Practice?

Please see <https://practice.geeksforgeeks.org/>

What are top algorithms in Interview Questions?

[Top 10 algorithms in Interview Questions](#)

How to prepare for ACM – ICPC?

[How to prepare for ACM – ICPC?](#)

This is an initial draft. We will soon be adding more links and algorithms to this post. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/top-algorithms-and-data-structures-for-competitive-programming/>

Chapter 168

Traversal of tree with k jumps allowed between nodes of same height

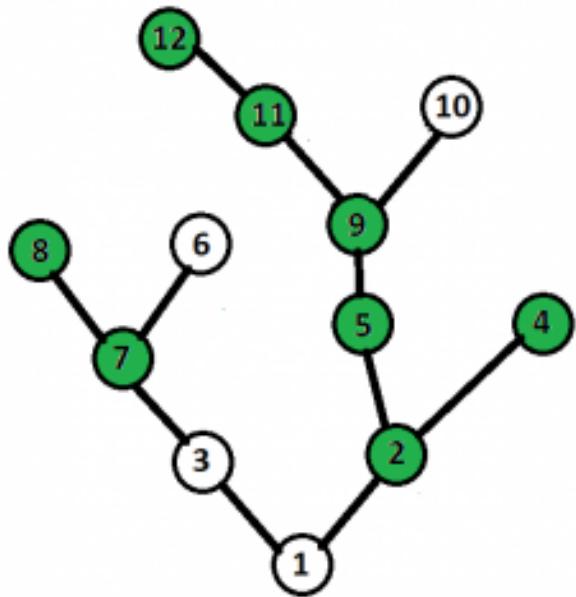
Traversal of tree with k jumps allowed between nodes of same height - GeeksforGeeks

Consider a tree with n nodes and root. You can jump from one node to any other node on the same height a maximum of k times on total jumps. Certain nodes of the tree contain a fruit, find out the maximum number of fruits he can collect.

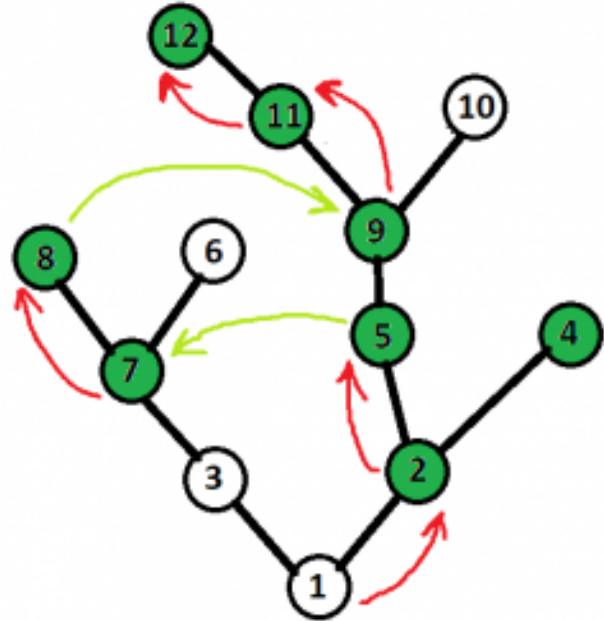
Example :

```
Input Tree :  
Number of Nodes = 12  
Number of jumps allowed : 2  
Edges:  
1 2  
1 3  
2 4  
2 5  
5 9  
9 10  
9 11  
11 12  
1 3  
3 7  
7 6  
7 8  
Nodes Containing Fruits : 2 4 5 7 8 9 11 12  
Output: 7
```

Tree for above testcase :



Explanation:



Approach: The idea is to use **DFS** to create a Height Adjacency List of the Nodes and to store the parents. Then use another dfs to compute the maximum no of special nodes that can be reached using the following dp state:

```
dp[current_node][j] = max( max{ dp[child_i][j], for all children of current_node },
                           max{ dp[node_at_same_height_i][j - 1],
                                 for all nodes at same height as current_node} )
```

Thus, **dp[Root_Node][Total_no_of_Jumps]** gives the answer to the problem.

Below is the implementation of above approach :

```
// Program to demonstrate tree traversal with
// ability to jump between nodes of same height
#include <bits/stdc++.h>
using namespace std;

#define N 1000

vector<int> H[N];

// Arrays declaration
```

```
int Fruit[N];
int Parent[N];
int dp[N][20];

// Function for DFS
void dfs1(vector<int> tree[], int s,
          int p, int h)
{
    Parent[s] = p;
    int i;
    H[h].push_back(s);
    for (i = 0; i < tree[s].size(); i++) {
        int v = tree[s][i];
        if (v != p)
            dfs1(tree, v, s, h + 1);
    }
}

// Function for DFS
int dfs2(vector<int> tree[], int s,
          int p, int h, int j)
{
    int i;
    int ans = 0;
    if (dp[s][j] != -1)
        return dp[s][j];

    // jump
    if (j > 0) {
        for (i = 0; i < H[h].size(); i++) {
            int v = H[h][i];
            if (v != s)
                ans = max(ans, dfs2(tree, v,
                                      Parent[v], h, j - 1));
        }
    }

    // climb
    for (i = 0; i < tree[s].size(); i++) {
        int v = tree[s][i];
        if (v != p)
            ans = max(ans, dfs2(tree, v, s, h + 1, j));
    }

    if (Fruit[s] == 1)
        ans++;
    dp[s][j] = ans;
}
```

```
    return ans;
}

// Function to calculate and
// return maximum number of fruits
int maxFruit(vector<int> tree[],
              int NodesWithFruits[],
              int n, int m, int k)
{
    // resetting dp table and Fruit array
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 20; j++)
            dp[i][j] = -1;
        Fruit[i] = 0;
    }

    // This array is used to mark
    // which nodes contain Fruits
    for (int i = 0; i < m; i++)
        Fruit[NodesWithFruits[i]] = 1;

    dfs1(tree, 1, 0, 0);
    int ans = dfs2(tree, 1, 0, 0, k);

    return ans;
}

// Function to add Edge
void addEdge(vector<int> tree[], int u, int v)
{
    tree[u].push_back(v);
    tree[v].push_back(u);
}

// Driver Code
int main()
{
    int n = 12; // Number of nodes
    int k = 2; // Number of allowed jumps

    vector<int> tree[N];

    // Edges
    addEdge(tree, 1, 2);
    addEdge(tree, 1, 3);
    addEdge(tree, 2, 4);
    addEdge(tree, 2, 5);
    addEdge(tree, 5, 9);
```

```
addEdge(tree, 9, 10);
addEdge(tree, 9, 11);
addEdge(tree, 11, 12);
addEdge(tree, 1, 3);
addEdge(tree, 3, 7);
addEdge(tree, 7, 6);
addEdge(tree, 7, 8);

int NodesWithFruits[] = { 2, 4, 5, 7, 8, 9, 11, 12 };

// Number of nodes with fruits
int m = sizeof(NodesWithFruits) / sizeof(NodesWithFruits[0]);

int ans = maxFruit(tree, NodesWithFruits, n, m, k);

cout << ans << endl;

return 0;
}
```

Output:

7

Time Complexity : $O(n \cdot n \cdot k)$ (worst case, eg: 2 level tree with the root having $n-1$ child nodes)

Source

<https://www.geeksforgeeks.org/traversal-tree-ability-jump-nodes-height/>

Chapter 169

Trick for modular division ($(x_1 * x_2 \dots x_n) / b$) mod (m)

Trick for modular division ($(x_1 * x_2 \dots x_n) / b$) mod (m) - GeeksforGeeks

Given integers $x_1, x_2, x_3\dots x_n, b$ and m , we are supposed to find the result of $((x_1*x_2\dots x_n)/b)\text{mod}(m)$.

Example 1 : Suppose that we are required to find $(55C5)\%(1000000007)$ i.e $((55*54*53*52*51)/120)\%1000000007$

Naive Method :

1. Simply calculate the product $(55*54*53*52*51)=$ say x ,
2. Divide x by 120 and then take its modulus with 1000000007

Using Modular Multiplicative Inverse :

Above method will work only when $x_1, x_2, x_3\dots x_n$ have small values.

Suppose we are required to find the result where $x_1, x_2, \dots x_n$ fall in the range of $\sim 1000000(10^6)$. So we will have to exploit the rule of modular mathematics which says :
 $(a*b)\text{mod}(m)=(a(\text{mod}(m))*b(\text{mod}(m)))\text{mod}(m)$

Note that the above formula is valid for modular multiplication. Similar formula for division does not exist.

i.e $(a/b)\text{mod}(m) \neq a(\text{mod}(m))/b(\text{mod}(m))$

1. So we are required to find out **modular multiplicative inverse** of b say i and then multiply ' i ' with a .
2. After this we will have to take the modulus of the result obtained.
i.e $((x_1*x_2\dots x_n)/b)\text{mod}(m)=((x_1*x_2\dots x_n)*i)\text{mod}(m)= ((x_1)\text{mod}(m) * (x_2)\text{mod}(m) * \dots (x_n)\text{mod}(m) * (i)\text{mod}(m))\text{mod}(m)$

Note : To find modular multiplicative inverse we can use [Extended Euclidian algorithm](#) or Fermat's Little Theorem.

Example 2 : Let us suppose that we have to find $(55555 \times 55554 \times 55553 \times 55552 \times 55551) / 120 \pmod{1000000007}$ i.e $((55555 \times 55554 \times 55553 \times 55552 \times 55551) / 120) \% 1000000007$.

```
// To run this code, we need to copy modular inverse
// from below post.
// https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

int main()
{
    // naive method-calculating the result in a single line
    long int naive_answer = ((long int)(55555 * 55554 *
        55553 * 55552 * 55551) / 120) % 1000000007;

    long int ans = 1;

    // modular_inverse() is a user defined function
    // that calculates inverse of a number
    long int i = modular_inverse(120, 100000007);

    // it will use extended Euclidian algorithm or
    // Fermat's Little Theorem for calculation.
    // MMI of 120 under division by 1000000007
    // will be 808333339
    for (int i = 0; i < 5; i++)
        ans = (ans * (55555 - i)) % 1000000007;

    ans = (ans * i) % 1000000007;
    cout << "Answer using naive method: " << naive_answer << endl;
    cout << "Answer using multiplicative"
        << " modular inverse concept: " << ans;

    return 0;
}
```

Input :

Output : Answer using naive method: -5973653
Answer using multiplicative modular inverse concept: 300820513

It is clear from the above example that the naive method will lead to overflow of data resulting in incorrect answer. Moreover, using modular inverse will give us the correct answer.

Without Using Modular Multiplicative Inverse :

But it is interesting to note that a slight change in code will discard the use of finding

modular multiplicative inverse.

C++

```
#include <iostream>
using namespace std;
int main()
{
    long int ans = 1;
    long int mod = (long int)1000000007 * 120;
    for (int i = 0; i < 5; i++)
        ans = (ans * (55555 - i)) % mod;
    ans = ans / 120;
    cout << "Answer using shortcut: " << ans;
    return 0;
}
```

Java

```
class GFG {

    public static void main(String[] args)
    {
        long ans = 1;
        long mod = (long)1000000007 * 120;

        for (int i = 0; i < 5; i++)
            ans = (ans * (55555 - i)) % mod;

        ans = ans / 120;
        System.out.println("Answer using"
                           + " shortcut: "+ ans);
    }
}

// This code is contributed by smitha.
```

Python3

```
ans = 1
mod = 1000000007 * 120

for i in range(0, 5) :
    ans = (ans * (55555 - i)) % mod

ans = int(ans / 120)
```

```
print("Answer using shortcut: ", ans)

# This code is contributed by Smitha.

C#

using System;

class GFG {

    public static void Main()
    {
        long ans = 1;
        long mod = (long)1000000007 * 120;

        for (int i = 0; i < 5; i++)
            ans = (ans * (55555 - i)) % mod;

        ans = ans / 120;
        Console.Write("Answer using "
                     + "shortcut: "+ ans);
    }
}

// This code is contributed by smitha.
```

PHP

```
<?php
$ans = 1;
$mod = 1000000007 * 120;

for ( $i = 0; $i < 5; $i++)
    $ans = ($ans * (55555 - $i)) %
          $mod;
$ans = $ans / 120;
echo "Answer using shortcut: " ,
      $ans;

// This code is contributed by anuj_67.
?>
```

Output :

```
Answer using shortcut: 300820513
```

Why did it work?

Let us consider $a = x_1 * x_2 * x_3 \dots * x_n$

We have to find $\text{ans} = (a/b) \% 1000000007$

1. Let result of $a \% (1000000007 * b)$ be y . To avoid overflow, we use modular multiplicative property. This can be represented as
$$a = (1000000007 * b)q + y \text{ where } y < (1000000007 * b) \text{ and } q \text{ is an integer}$$
2. Now dividing LHS and RHS by b , we get
$$\begin{aligned} y/b &= a/b - (1000000007 * b) * q/b \\ &= a/b - 1000000007 * q < 1000000007 \text{ (From 1)} \end{aligned}$$

Therefore, y/b is equivalent to $(a/b) \bmod (b * 1000000007)$.

Improved By : [Smitha Dinesh Semwal, vt_m](#)

Source

<https://www.geeksforgeeks.org/trick-modular-division-x1x2-xnbmodm/>

Chapter 170

Triplet with no element divisible by 3 and sum N

Triplet with no element divisible by 3 and sum N - GeeksforGeeks

Given an integer N greater than 2, the task is to print any combination of a, b and c such that:

- $a + b + c = N$
- a, b and c are not divisible by 3.

Examples:

Input: N = 233
Output: 77 77 79

Input: N = 3
Output: 1 1 1

A **naive approach** is to use three loops and check for the given condition.

Below is the implementation of the above approach:

C++

```
// C++ program to print a, b and c
// such that a+b+c=N
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to print a, b and c
void printCombination(int n)
{
    // first loop
    for (int i = 1; i < n; i++) {

        // check for 1st number
        if (i % 3 != 0) {

            // second loop
            for (int j = 1; j < n; j++) {

                // check for 2nd number
                if (j % 3 != 0) {

                    // third loop
                    for (int k = 1; k < n; k++) {

                        // Check for 3rd number
                        if (k % 3 != 0 && (i + j + k) == n) {
                            cout << i << " " << j << " " << k;
                            return;
                        }
                    }
                }
            }
        }
    }
}

// Driver Code
int main()
{
    int n = 233;

    printCombination(n);
    return 0;
}
```

Java

```
// Java program to print a,
// b and c such that a+b+c=N
import java.io.*;

class GFG
{
```

```
// Function to print a, b and c
static void printCombination(int n)
{
    // first loop
    for (int i = 1; i < n; i++)
    {

        // check for 1st number
        if (i % 3 != 0)
        {

            // second loop
            for (int j = 1; j < n; j++)
            {

                // check for 2nd number
                if (j % 3 != 0)
                {

                    // third loop
                    for (int k = 1; k < n; k++)
                    {

                        // Check for 3rd number
                        if (k % 3 != 0 && (i + j + k) == n)
                        {
                            System.out.println( i + " " +
                                j + " " + k);
                            return;
                        }
                    }
                }
            }
        }
    }
}

// Driver Code
public static void main (String[] args)
{
    int n = 233;

    printCombination(n);
}
}

// This code is contributed
```

```
// by anuj_67.
```

Python3

```
# Python3 program to print a, b  
# and c such that a+b+c=N
```

```
# Function to print a, b and c  
def printCombination(n):
```

```
# first loop
```

```
for i in range(1, n):
```

```
# check for 1st number
```

```
if (i % 3 != 0):
```

```
# second loop
```

```
for j in range(1, n):
```

```
# check for 2nd number
```

```
if (j % 3 != 0):
```

```
# third loop
```

```
for k in range(1, n):
```

```
# Check for 3rd number
```

```
if (k % 3 != 0 and
```

```
(i + j + k) == n):
```

```
print(i, j, k);
```

```
return;
```

```
# Driver Code
```

```
n = 233;
```

```
printCombination(n);
```

```
# This code is contributed
```

```
# by mits
```

```
C#
```

```
// C# program to print a,
```

```
// b and c such that a+b+c=N
```

```
class GFG
```

```
{
```

```
// Function to print a, b and c
```

```
static void printCombination(int n)
```

```
{
```

```
// first loop
```

```
for (int i = 1; i < n; i++)
```

```
{
```

```
// check for 1st number
```

```
if (i % 3 != 0)
{
    // second loop
    for (int j = 1; j < n; j++)
    {
        // check for 2nd number
        if (j % 3 != 0)
        {
            // third loop
            for (int k = 1; k < n; k++)
            {
                // Check for 3rd number
                if (k % 3 != 0 && (i + j + k) == n)
                {
                    System.Console.WriteLine(i + " " +
                                              j + " " + k);
                    return;
                }
            }
        }
    }
}

// Driver Code
static void Main ()
{
    int n = 233;

    printCombination(n);
}
}

// This code is contributed
// by mits
```

PHP

```
<?php
// PHP program to print a, b and c
// such that a+b+c=N

// Function to print a, b and c
```

```
function printCombination($n)
{
    // first loop
    for ($i = 1; $i < $n; $i++)
    {
        // check for 1st number
        if ($i % 3 != 0)
        {

            // second loop
            for ($j = 1; $j < $n; $j++)
            {

                // check for 2nd number
                if ($j % 3 != 0)
                {

                    // third loop
                    for ($k = 1; $k < $n; $k++)
                    {

                        // Check for 3rd number
                        if ($k % 3 != 0 &&
                            ($i + $j + $k) == $n)
                        {
                            echo $i , " " , $j , " " , $k;
                            return;
                        }
                    }
                }
            }
        }
    }

    // Driver Code
    $n = 233;

    printCombination($n);

    // This code is contributed
    // inder_verma
    ?>
```

Output:

1 2 230

Time Complexity: $O(n^3)$

Efficient approach:

1. Consider 1 as one of the three numbers.
2. The other two numbers will be:
 - **(2, n-3)** if $n-2$ is divisible by 3.
 - Otherwise **(1, n-2)** if $n-2$ is not divisible by 3.

Below is the implementation of the above approach:

C++

```
// C++ program to print a, b and c
// such that a+b+c=N
#include <bits/stdc++.h>
using namespace std;

// Function to print a, b and c
void printCombination(int n)
{
    cout << 1 << " ";

    // check if n-2 is divisible
    // by 3 or not
    if ((n - 2) % 3 == 0)
        cout << 2 << " " << n - 3;
    else
        cout << 1 << " " << n - 2;
}

// Driver Code
int main()
{
    int n = 233;

    printCombination(n);
    return 0;
}
```

Output:

1 2 230

Time Complexity: O(1)

Improved By : [inderDuMCA](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/triplet-with-no-element-divisible-by-3-and-sum-n/>

Chapter 171

Two Dimensional Segment Tree | Sub-Matrix Sum

Two Dimensional Segment Tree | Sub-Matrix Sum - GeeksforGeeks

Given a rectangular matrix $M[0...n-1][0...m-1]$, and queries are asked to find the sum / minimum / maximum on some sub-rectangles $M[a...b][e...f]$, as well as queries for modification of individual matrix elements (i.e $M[x][y] = p$).

We can also answer sub-matrix queries using [Two Dimensional Binary Indexed Tree](#).

In this article, We will focus on solving sub-matrix queries using two dimensional segment tree.Two dimensional segment tree is nothing but segment tree of segment trees.

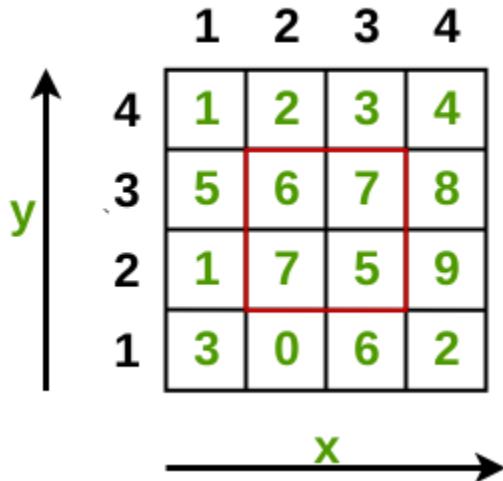
Prerequisite : [Segment Tree – Sum of given range](#)

Algorithm :

We will build a two-dimensional tree of segments by the following principle:

- 1 . In First step, We will construct an ordinary one-dimensional segment tree, working only with the first coordinate say ‘x’ and ‘y’ as constant. Here, we will not write number in inside the node as in the one-dimensional segment tree, but an entire tree of segments.
2. The second step is to combine the values of segmented trees. Assume that in second step instead of combining the elements we are combining the segment trees obtained from the step first.

Consider the below example. Suppose we have to find the sum of all numbers inside the highlighted red area



Step 1 : We will first create the segment tree of each strip of y- axis. We represent the segment tree here as an array where child node is $2n$ and $2n+1$ where $n > 0$.

Segment Tree for strip $y=1$

10	3	7	1	2	3	4
----	---	---	---	---	---	---

Segment Tree for Strip $y = 2$

26	11	15	5	6	7	8
----	----	----	---	---	---	---

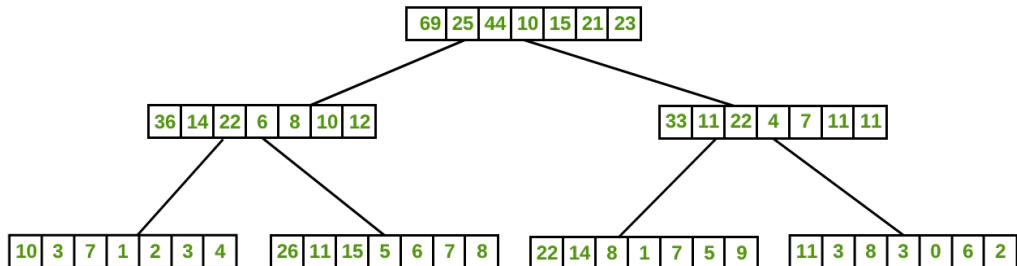
Segment Tree for Strip $y = 3$

22	14	8	1	7	5	9
----	----	---	---	---	---	---

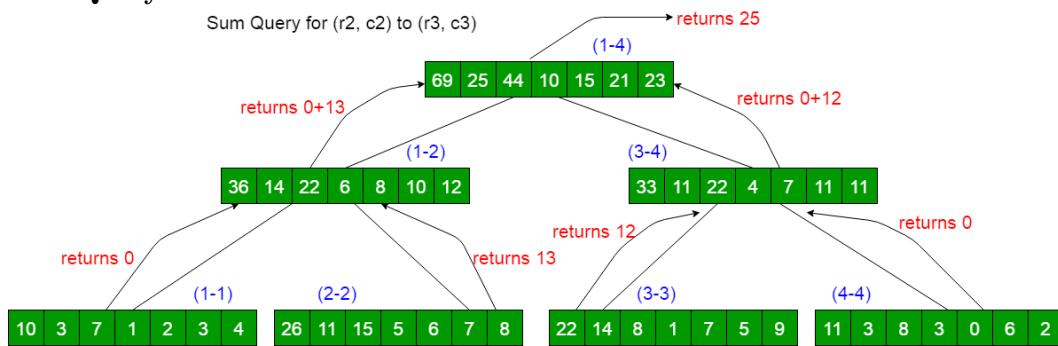
Segment Tree for Strip $y = 4$

11	3	8	3	0	6	2
----	---	---	---	---	---	---

Step 2: In this step, we create the segment tree for the rectangular matrix where the base node are the strips of y-axis given above. The task is to merge above segment trees.



Sum Query :



Thanks to **Sahil Bansal** for contributing this image.

Processing Query :

We will respond to the two-dimensional query by the following principle: first to break the query on the first coordinate, and then, when we reached some vertex of the tree of segments with the first coordinate and then we call the corresponding tree of segments on the second coordinate.

This function works in time **O(log n * log m)**, because it first descends the tree in the first coordinate, and for each traversed vertex of that tree, it makes a query from the usual tree of segments along the second coordinate.

Modification Query :

We want to learn how to modify the tree of segments in accordance with the change in the value of an element $M[x][y] = p$. It is clear that the changes will occur only in those vertices of the first tree of segments that cover the coordinate x , and for the trees of the segments corresponding to them, the changes will only occur in those vertices that cover the coordinate y . Therefore, the implementation of the modification request will not be very different from the one-dimensional case, only now we first descend the first coordinate, and then the second.

Output for the highlighted area will be 25.

Below is the implementation of above approach :

```

// C++ program for implementation
// of 2D segment tree.
#include <bits/stdc++.h>
using namespace std;

// Base node of segment tree.
int ini_seg[1000][1000] = { 0 };

// final 2d-segment tree.
int fin_seg[1000][1000] = { 0 };

// Rectangular matrix.
int rect[4][4] = {
    { 1, 2, 3, 4 },
    { 5, 6, 7, 8 },
    { 1, 7, 5, 9 },
    { 3, 0, 6, 2 },
};

// size of x coordinate.
int size = 4;

/*
 * A recursive function that constructs
 * Initial Segment Tree for array rect[][] = { }.
 * 'pos' is index of current node in segment
 * tree seg[]. 'strip' is the enumeration
 * for the y-axis.
 */
int segment(int low, int high,
            int pos, int strip)
{
    if (high == low) {
        ini_seg[strip][pos] = rect[strip][low];
    }
    else {
        int mid = (low + high) / 2;
        segment(low, mid, 2 * pos, strip);
        segment(mid + 1, high, 2 * pos + 1, strip);
        ini_seg[strip][pos] = ini_seg[strip][2 * pos] +
                            ini_seg[strip][2 * pos + 1];
    }
}

/*
 * A recursive function that constructs
 * Final Segment Tree for array ini_seg[][] = { }.

```

```

/*
int finalSegment(int low, int high, int pos)
{
    if (high == low) {

        for (int i = 1; i < 2 * size; i++)
            fin_seg[pos][i] = ini_seg[low][i];
    }
    else {
        int mid = (low + high) / 2;
        finalSegment(low, mid, 2 * pos);
        finalSegment(mid + 1, high, 2 * pos + 1);

        for (int i = 1; i < 2 * size; i++)
            fin_seg[pos][i] = fin_seg[2 * pos][i] +
                fin_seg[2 * pos + 1][i];
    }
}

/*
* Return sum of elements in range from index
* x1 to x2 . It uses the final_seg[][] array
* created using finalsegment() function.
* 'pos' is index of current node in
* segment tree fin_seg[][] .
*/
int finalQuery(int pos, int start, int end,
               int x1, int x2, int node)
{
    if (x2 < start || end < x1) {
        return 0;
    }

    if (x1 <= start && end <= x2) {
        return fin_seg[node][pos];
    }

    int mid = (start + end) / 2;
    int p1 = finalQuery(2 * pos, start, mid,
                        x1, x2, node);

    int p2 = finalQuery(2 * pos + 1, mid + 1,
                        end, x1, x2, node);

    return (p1 + p2);
}
*/

```

```

* This fuction calls the finalQuery fuction
* for elements in range from index x1 to x2 .
* This fuction queries the yth coordinate.
*/
int query(int pos, int start, int end,
          int y1, int y2, int x1, int x2)
{
    if (y2 < start || end < y1) {
        return 0;
    }

    if (y1 <= start && end <= y2) {
        return (finalQuery(1, 1, 4, x1, x2, pos));
    }

    int mid = (start + end) / 2;
    int p1 = query(2 * pos, start,
                   mid, y1, y2, x1, x2);
    int p2 = query(2 * pos + 1, mid + 1,
                   end, y1, y2, x1, x2);

    return (p1 + p2);
}

/* A recursive function to update the nodes
   which for the given index. The following
   are parameters : pos --> index of current
   node in segment tree fin_seg[][] . x ->
   index of the element to be updated. val -->
   Value to be change at node idx
*/
int finalUpdate(int pos, int low, int high,
                int x, int val, int node)
{
    if (low == high) {
        fin_seg[node][pos] = val;
    }
    else {
        int mid = (low + high) / 2;

        if (low <= x && x <= mid) {
            finalUpdate(2 * pos, low, mid, x, val, node);
        }
        else {
            finalUpdate(2 * pos + 1, mid + 1, high,
                        x, val, node);
        }
    }
}

```

```

        fin_seg[node][pos] = fin_seg[node][2 * pos] +
                            fin_seg[node][2 * pos + 1];
    }
}

/*
This function call the final update function after
visiting the yth coordinate in the segment tree fin_seg[][].
*/
int update(int pos, int low, int high, int x, int y, int val)
{
    if (low == high) {
        finalUpdate(1, 1, 4, x, val, pos);
    }
    else {
        int mid = (low + high) / 2;

        if (low <= y && y <= mid) {
            update(2 * pos, low, mid, x, y, val);
        }
        else {
            update(2 * pos + 1, mid + 1, high, x, y, val);
        }

        for (int i = 1; i < size; i++)
            fin_seg[pos][i] = fin_seg[2 * pos][i] +
                               fin_seg[2 * pos + 1][i];
    }
}

// Driver program to test above functions
int main()
{
    int pos = 1;
    int low = 0;
    int high = 3;

    // Call the ini_segment() to create the
    // initial segment tree on x- coordinate
    for (int strip = 0; strip < 4; strip++)
        segment(low, high, 1, strip);

    // Call the final function to built the 2d segment tree.
    finalSegment(low, high, 1);

    /*
Query:
* To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)

```

```
* update the value of index (3, 3)=100;
* To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)
*/
    cout << "The sum of the submatrix (y1, y2)->(2, 3), "
        << "(x1, x2)->(2, 3) is "
        << query(1, 1, 4, 2, 3, 2, 3) << endl;

// Function to update the value
update(1, 1, 4, 2, 3, 100);

cout << "The sum of the submatrix (y1, y2)->(2, 3), "
        << "(x1, x2)->(2, 3) is "
        << query(1, 1, 4, 2, 3, 2, 3) << endl;

return 0;
}
```

Output:

```
The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 25
The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 118
```

Time complexity :

Processing Query : $O(\log n * \log m)$
Modification Query: $O(2 * n * \log n * \log m)$
Space Complexity : $O(4 * m * n)$

Source

<https://www.geeksforgeeks.org/two-dimensional-segment-tree-sub-matrix-sum/>

Chapter 172

Using Chinese Remainder Theorem to Combine Modular equations

Using Chinese Remainder Theorem to Combine Modular equations - GeeksforGeeks

Given N modular equations:

A $x_1 \text{mod}(m_1)$

.

A $x_n \text{mod}(m_n)$

Find x in the equation A $x \text{mod}(m_1 * m_2 * m_3 .. * m_n)$

where m_i is prime, or a power of a prime, and i takes values from 1 to n.

The input is given as two arrays, the first being an array containing values of each x_i , and the second array containing the set of values of each prime. m_i

Output an integer for the value of x in the final equation.

Examples :

Consider the two equations

A 2mod(3)

A 3mod(5)

Input :

2 3

3 5

Output :

8

Consider the four equations,

A 3mod(4)

A 4mod(7)

```

A 1mod(9) (32)
A 0mod(11)
Input :
3 4 1 0
4 7 9 11
Output :
1243

```

Explanation :

We aim to solve these equations two at a time. We take the first two equations, combine it, and use that result to combine with the third equation, and so on. The process of combining two equations is explained as follows, by taking example 2 for reference:

1. A 3mod(4) and A 4mod(7) are the two equations that we are provided with at first.
Let the resulting equation be some $A_0 \equiv x_0 \pmod{m_1 * m_2}$.
 - A_0 is given by $m_1^{-1} * m_1 * x_0 + m_0^{-1} * m_0 * x_1$
where m_1^{-1} = modular inverse of m_1 modulo m_0 and m_0^{-1} = modular inverse of m_0 modulo m_1
 - We can calculate modular inverse using extended euclidean algorithm.
 - We find x_0 to be $A_0 \pmod{m_1 * m_2}$
 - We get our new equation to be A 11mod(28), where A is 95
2. We now try to combine this with equation 3, and by a similar method, we get A 235mod(252), where A = 2503
3. And finally, on combining this with equation 4, we get A 1243mod(2772) where A = 59455 and x = 1243

We observe that 2772 is rightly equal to $4 * 7 * 9 * 11$.

We have thus found the value of x for the final equation.

You can refer to [Extended Euclidean Algorithm](#) and [Modular multiplicative inverse](#) for extra information on these topics.

```

# Python 2.x program to combine modular equations
# using Chinese Remainder Theorem

# function that implements Extended euclidean
# algorithm
def extended_euclidean(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_euclidean(b % a, a)
        return (g, x - (b // a) * y, y)

# modular inverse driver function
def modinv(a, m):

```

```
g, x, y = extended_euclidean(a, m)
return x % m

# function implementing Chinese remainder theorem
# list m contains all the modulii
# list x contains the remainders of the equations
def crt(m, x):

    # We run this loop while the list of
    # remainders has length greater than 1
    while True:

        # temp1 will contain the new value
        # of A. which is calculated according
        # to the equation  $m_1' * m_1 * x_0 + m_0'$ 
        # *  $m_0 * x_1$ 
        temp1 = modinv(m[1],m[0]) * x[0] * m[1] + \
                modinv(m[0],m[1]) * x[1] * m[0]

        # temp2 contains the value of the modulus
        # in the new equation, which will be the
        # product of the modulii of the two
        # equations that we are combining
        temp2 = m[0] * m[1]

        # we then remove the first two elements
        # from the list of remainders, and replace
        # it with the remainder value, which will
        # be  $temp1 \% temp2$ 
        x.remove(x[0])
        x.remove(x[0])
        x = [temp1 % temp2] + x

        # we then remove the first two values from
        # the list of modulii as we no longer require
        # them and simply replace them with the new
        # modulii that we calculated
        m.remove(m[0])
        m.remove(m[0])
        m = [temp2] + m

        # once the list has only one element left,
        # we can break as it will only contain
        # the value of our final remainder
        if len(x) == 1:
            break

    # returns the remainder of the final equation
```

```
return x[0]

# driver segment
m = [4, 7, 9, 11]
x = [3, 4, 1, 0]
print crt(m, x)
```

Output
1243

This theorem and algorithm has excellent applications. One very useful application is in calculating ${}^nC_r \% m$ where m is not a prime number, and [Lucas Theorem](#) cannot be directly applied. In such a case, we can calculate the prime factors of m, and use the prime factors one by one as a modulus in our ${}^nC_r \% m$ equation which we can calculate using Lucas Theorem, and then combine the resulting equations together using the above shown Chinese Remainder Theorem.

Source

<https://www.geeksforgeeks.org/using-chinese-remainder-theorem-combine-modular-equations/>

Chapter 173

Vantieghems Theorem for Primality Test

Vantieghems Theorem for Primality Test - GeeksforGeeks

[Vantieghems Theorem](#) is a necessary and sufficient condition for a number to be prime.

It states that for a natural number n to be prime, the product of $2^i - 1$ where $2 \leq i \leq n$, is congruent to $1 \pmod{n}$.

In other words, a number n is prime if and only if.

$$\prod_{1 \leq i \leq n-1} (2^i - 1) \equiv 1 \pmod{n}$$

Examples:

- For $n = 3$, final product is $(2^1 - 1) * (2^2 - 1) = 1 * 3 = 3$. 3 is congruent to 3 mod 7. We get 3 mod 7 from expression $3 * (mod(2^3 - 1))$, therefore 3 is prime.
- For $n = 5$, final product is $1 * 3 * 7 * 15 = 315$. 315 is congruent to 5(mod 31), therefore 5 is prime.
- For $n = 7$, final product is $1 * 3 * 7 * 15 * 31 * 63 = 615195$. 615195 is congruent to 7(mod 127), therefore 7 is prime.
- For $n = 4$, final product $1 * 3 * 7 = 21$. 21 is not congruent to 4(mod 15), therefore 4 is composite.

Another way to state above theorem is, if $\{2^{n-i} - 1\}_{i=1}^{n-1}$ divides $\{2^n - 1\}_{i=1}^{n-1}$, then n is prime.

```
// CPP code to verify Vantieghem's Theorem
#include <bits/stdc++.h>
using namespace std;

void checkVantieghemsTheorem(int limit)
{
    long long unsigned prod = 1;
    for (long long unsigned n = 2; n < limit; n++) {

        // Check if above condition is satisfied
        if (((prod - n) % ((1LL << n) - 1)) == 0)
            cout << n << " is prime\n";

        // product of previous powers of 2
        prod *= ((1LL << n) - 1);
    }
}

// Driver code
int main()
{
    checkVantieghemsTheorem(10);
    return 0;
}
```

Output:

```
2 is prime
3 is prime
5 is prime
7 is prime
```

The above code does not work for values of n higher than 11. It causes overflow in prod evaluation.

Source

<https://www.geeksforgeeks.org/vantieghems-theorem-primality-test/>

Chapter 174

Variation in Nim Game

Variation in Nim Game - GeeksforGeeks

Prerequisites:

[Sprague Grunsky theorem](#)
[Grundy Numbers](#)

Nim is a famous game in which two players take turns removing items from distinct piles. During each turn, a player must remove one or more items from a single, non-empty pile. The winner of the game is whichever player removes the last item from the last non-empty pile.

Now, For each non-empty pile, either player can remove zero items from that pile and have it count as their move; however, this move can only be performed once per pile by either player.

Given the number of items in each pile, determine who will win the game; Player 1 or player 2. If player 1 starts the game and both plays optimally.

Examples:

```
Input : 3 [18, 47, 34]
Output : Player 2 wins
G = g(18)^g(47)^g(34) = (17)^(48)^(33) = 0
Grundy number(G), for this game is zero.
Player 2 wins.
```

```
Input : 3 [32, 49, 58]
Output : Player 1 wins
G = g(31)^g(50)^g(57) = (17)^(48)^(33) = 20
Grundy number(G), for this game is non-zero.
Player 1 wins.
```

Approach:

Grundy number for each pile is calculated based on the number of stones. To compensate

the zero move we will have to modify grundy values we used in standard nim game.

If pile size is odd; grundy number is size+1 and

if pile size is even; grundy number is size-1.

We XOR all the grundy number values to check if final Grundy number(G) of game is non zero or not to decide who is winner of game.

Explanation:

Grundy number of a state is the **smallest positive integer that cannot be reached in one valid move**.

So, we need to calculate **mex value** for each n, bottom up wise so that we can induce the grundy number for each n. where n is the pile size and valid move is the move that will lead the current player to winning state.

Winning state: A tuple of values from where the current player will win the game no matter what opponent does. (If $G \neq 0$)

Losing state: A tuple of values from where the current player will loose the game no matter what opponent does. (If $G = 0$)

For a given pile size n, we have two states:

- (1) n with no zero move available,
grundy number will same as standard nim game.
- (2) n with zero move available, we can
reach above state and other states with
zero move remaining.

For, $n = 0$, $g(0) = 0$, empty pile

For, $n = 1$, we can reach two states:

- (1) $n = 0$ (zero move not used)
- (2) $n = 1$ (zero move used)

Therefore, $g(1) = \text{mex}\{0, 1\}$ which implies
that $g(1)=2$.

For, $n = 2$, we can reach :

- (1) $n = 0$ (zero move not used) state
because this is a valid move.
 - (2) $n = 1$ is not a valid move, as it will
lead the current player into loosing state.
- Therefore, $g(2) = \text{mex}\{0\}$ which implies
that $g(2)=1$.

If we try to build a solution bottom-up
like this, it turns out that if n is even,
the grundy number is $n - 1$ and when it is odd,
the grundy is $n + 1$.

Below is the implementation of above approach:

C++

```
// CPP program for the variation
// in nim game
#include <bits/stdc++.h>
using namespace std;

// Function to return final
// grundy Number(G) of game
int solve(int p[], int n)
{
    int G = 0;
    for (int i = 0; i < n; i++) {

        // if pile size is odd
        if (p[i] & 1)

            // We XOR pile size+1
            G ^= (p[i] + 1);

        else // if pile size is even

            // We XOR pile size-1
            G ^= (p[i] - 1);
    }
    return G;
}

// driver program
int main()
{
    // Game with 3 piles
    int n = 3;

    // pile with different sizes
    int p[3] = { 32, 49, 58 };

    // Function to return result of game
    int res = solve(p, n);

    if (res == 0) // if G is zero
        cout << "Player 2 wins";
    else // if G is non zero
        cout << "Player 1 wins";

    return 0;
}
```

Java

```
// Java program for the variation
// in nim game
class GFG {

    // Function to return final
    // grundy Number(G) of game
    static int solve(int p[], int n)
    {

        int G = 0;
        for (int i = 0; i < n; i++) {

            // if pile size is odd
            if (p[i] % 2 != 0)

                // We XOR pile size+1
                G ^= (p[i] + 1);

            else // if pile size is even

                // We XOR pile size-1
                G ^= (p[i] - 1);
        }

        return G;
    }

    //Driver code
    public static void main (String[] args)
    {

        // Game with 3 piles
        int n = 3;

        // pile with different sizes
        int p[] = { 32, 49, 58 };

        // Function to return result of game
        int res = solve(p, n);

        if (res == 0) // if G is zero
            System.out.print("Player 2 wins");
        else // if G is non zero
            System.out.print("Player 1 wins");
    }
}
```

```
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program for the
# variation in nim game

# Function to return final
# grundy Number(G) of game
def solve(p, n):
    G = 0
    for i in range(n):

        # if pile size is odd
        if (p[i] % 2 != 0):

            # We XOR pile size+1
            G ^= (p[i] + 1)

        # if pile size is even
        else:

            # We XOR pile size-1
            G ^= (p[i] - 1)

    return G

# Driver code

# Game with 3 piles
n = 3

# pile with different sizes
p = [32, 49, 58]

# Function to return result of game
res = solve(p, n)

if (res == 0): # if G is zero
    print("Player 2 wins")

else: # if G is non zero
    print("Player 1 wins")

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program for the variation
// in nim game
using System;
class GFG {

    // Function to return final
    // grundy Number(G) of game
    static int solve(int[] p, int n)
    {

        int G = 0;
        for (int i = 0; i < n; i++) {

            // if pile size is odd
            if (p[i] % 2 != 0)

                // We XOR pile size+1
                G ^= (p[i] + 1);

            else // if pile size is even

                // We XOR pile size-1
                G ^= (p[i] - 1);
        }

        return G;
    }

    // Driver code
    public static void Main()
    {

        // Game with 3 piles
        int n = 3;

        // pile with different sizes
        int[] p = { 32, 49, 58 };

        // Function to return result of game
        int res = solve(p, n);

        if (res == 0) // if G is zero
            Console.WriteLine("Player 2 wins");

        else // if G is non zero
            Console.WriteLine("Player 1 wins");
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// php program for the variation
// in nim game

// Function to return final
// grundy Number(G) of game
function solve($p,$n)
{
    $G = 0;
    for ($i = 0; $i < $n; $i++)
    {
        // if pile size is odd
        if ($p[$i] & 1)

            // We XOR pile size+1
            $G ^= ($p[$i] + 1);

        else // if pile size is even

            // We XOR pile size-1
            $G ^= ($p[$i] - 1);
    }
    return $G;
}

// Driver Code
// Game with 3 piles
$n = 3;

// pile with different sizes
$p= array( 32, 49, 58 );

// Function to return result of game
$res = solve($p, $n);

if ($res == 0) // if G is zero
    echo "Player 2 wins";
else // if G is non zero
    echo "Player 1 wins";

// This code is contributed by mits
?>
```

Output:

Player 1 wins

Time Complexity: $O(n)$

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/varyation-nim-game/>

Chapter 175

Water Connection Problem

Water Connection Problem - GeeksforGeeks

Every house in the colony has at most one pipe going into it and at most one pipe going out of it. Tanks and taps are to be installed in a manner such that every house with one outgoing pipe but no incoming pipe gets a tank installed on its roof and every house with only an incoming pipe and no outgoing pipe gets a tap.'

Given two integers **n** and **p** denoting the number of houses and the number of pipes. The connections of pipe among the houses contain three input values: **a_i**, **b_i**, **d_i** denoting the pipe of diameter **d_i** from house **a_i** to house **b_i**, find out the efficient solution for the network.

The output will contain the number of pairs of tanks and taps **t** installed in first line and the next **t** lines contain three integers: house number of tank, house number of tap and the minimum diameter of pipe between them.

Examples:

```
Input : 4 2
        1 2 60
        3 4 50
```

```
Output :2
```

```
        1 2 60
        3 4 50
```

Explanation:

Connected components are:

1->2 and 3->4

Therefore, our answer is 2 followed by

1 2 60 and 3 4 50.

```
Input :9 6
        7 4 98
        5 9 72
        4 6 10
```

```
2 8 22
9 7 17
3 1 66
Output :3
2 8 22
3 1 66
5 6 10
Explanation:
Connected components are 3->1,
5->9->7->4->6 and 2->8.
Therefore, our answer is 3 followed by
2 8 22, 3 1 66, 5 6 10
```

Approach:

Perform DFS from appropriate houses to find all different connected components. The number of different connected components is our answer t.

The next t lines of the output are the beginning of the connected component, end of the connected component and the minimum diameter from the start to the end of the connected component in each line.

Since, tanks can be installed only on the houses having outgoing pipe and no incoming pipe, therefore these are appropriate houses to start DFS from i.e. perform DFS from such unvisited houses.

Below is the implementation of above approach:

C++

```
// C++ program to find efficient
// solution for the network
#include <bits/stdc++.h>
using namespace std;

// number of houses and number
// of pipes
int n, p;

// Array rd stores the
// ending vertex of pipe
int rd[1100];

// Array wd stores the value
// of diameters between two pipes
int wt[1100];

// Array cd stores the
// starting end of pipe
int cd[1100];
```

```

// Vector a, b, c are used
// to store the final output
vector<int> a;
vector<int> b;
vector<int> c;

int ans;

int dfs(int w)
{
    if (cd[w] == 0)
        return w;
    if (wt[w] < ans)
        ans = wt[w];
    return dfs(cd[w]);
}

// Function performing calculations.
void solve(int arr[][3])
{
    int i = 0;

    while (i < p) {

        int q = arr[i][0], h = arr[i][1],
            t = arr[i][2];

        cd[q] = h;
        wt[q] = t;
        rd[h] = q;
        i++;
    }

    a.clear();
    b.clear();
    c.clear();

    for (int j = 1; j <= n; ++j)

        /*If a pipe has no ending vertex
        but has starting vertex i.e is
        an outgoing pipe then we need
        to start DFS with this vertex.*/
        if (rd[j] == 0 && cd[j]) {
            ans = 1000000000;
            int w = dfs(j);

            // We put the details of component

```

```
// in final output array
a.push_back(j);
b.push_back(w);
c.push_back(ans);
}

cout << a.size() << endl;
for (int j = 0; j < a.size(); ++j)
    cout << a[j] << " " << b[j]
        << " " << c[j] << endl;
}

// driver function
int main()
{
    n = 9, p = 6;

    memset(rd, 0, sizeof(rd));
    memset(cd, 0, sizeof(cd));
    memset(wt, 0, sizeof(wt));

    int arr[][][3] = { { 7, 4, 98 },
                      { 5, 9, 72 },
                      { 4, 6, 10 },
                      { 2, 8, 22 },
                      { 9, 7, 17 },
                      { 3, 1, 66 } };

    solve(arr);
    return 0;
}
```

Java

```
// Java program to find efficient
// solution for the network
import java.util.*;

class GFG {

    // number of houses and number
    // of pipes
    static int n, p;

    // Array rd stores the
    // ending vertex of pipe
    static int rd[] = new int[1100];
```

```
// Array wd stores the value
// of diameters between two pipes
static int wt[] = new int[1100];

// Array cd stores the
// starting end of pipe
static int cd[] = new int[1100];

// arraylist a, b, c are used
// to store the final output
static List <Integer> a =
    new ArrayList<Integer>();

static List <Integer> b =
    new ArrayList<Integer>();

static List <Integer> c =
    new ArrayList<Integer>();

static int ans;

static int dfs(int w)
{
    if (cd[w] == 0)
        return w;
    if (wt[w] < ans)
        ans = wt[w];

    return dfs(cd[w]);
}

// Function to perform calculations.
static void solve(int arr[][])
{
    int i = 0;

    while (i < p)
    {

        int q = arr[i][0];
        int h = arr[i][1];
        int t = arr[i][2];

        cd[q] = h;
        wt[q] = t;
        rd[h] = q;
        i++;
    }
}
```

```

a=new ArrayList<Integer>();
b=new ArrayList<Integer>();
c=new ArrayList<Integer>();

for (int j = 1; j <= n; ++j)

    /*If a pipe has no ending vertex
    but has starting vertex i.e is
    an outgoing pipe then we need
    to start DFS with this vertex.*/
    if (rd[j] == 0 && cd[j]>0) {
        ans = 1000000000;
        int w = dfs(j);

        // We put the details of
        // component in final output
        // array
        a.add(j);
        b.add(w);
        c.add(ans);
    }

System.out.println(a.size());

for (int j = 0; j < a.size(); ++j)
    System.out.println(a.get(j) + " "
                      + b.get(j) + " " + c.get(j));
}

// main function
public static void main(String args[])
{
    n = 9;
    p = 6;

    // set the value of the araray
    // to zero
    for(int i = 0; i < 1100; i++)
        rd[i] = cd[i] = wt[i] = 0;

    int arr[] [] = { { 7, 4, 98 },
                    { 5, 9, 72 },
                    { 4, 6, 10 },
                    { 2, 8, 22 },
                    { 9, 7, 17 },
                    { 3, 1, 66 } };
    solve(arr);
}

```

```
    }  
}  
  
// This code is contributed by Arnab Kundu
```

Output:

```
3  
2 8 22  
3 1 66  
5 6 10
```

Improved By : [andrew1234](#), [ROSHANRK_1995](#)

Source

<https://www.geeksforgeeks.org/water-connection-problem/>

Chapter 176

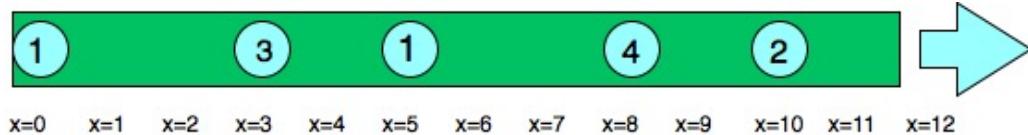
Water drop problem

Water drop problem - GeeksforGeeks

Consider a pipe of length L. The pipe has N water droplets at N different positions within it. Each water droplet is moving towards the end of the pipe($x=L$) at different rates. When a water droplet mixes with another water droplet, it assumes the speed of the water droplet it is mixing with. Determine the no of droplets that come out of the end of the pipe.

Refer to the figure below:

Numbers on circles indicates speed of water droplets



Examples:

Input: length = 12, position = [10, 8, 0, 5, 3],
speed = [2, 4, 1, 1, 3]

Output: 3

Explanation:

Droplets starting at $x=10$ and $x=8$ become a droplet, meeting each other at $x=12$ at time =1 sec.

The droplet starting at 0 doesn't mix with any other droplet, so it is a drop by itself.

Droplets starting at $x=5$ and $x=3$ become a single drop, mixing with each other at $x=6$ at time = 1 sec.

Note that no other droplets meet these drops before the end of the pipe, so the answer is 3.

Refer to the figure below

Numbers on circles indicates speed of water droplets.

Approach:

This problem uses greedy technique.

A drop will mix with another drop if two conditions are met:

1. If the drop is faster than the drop it is mixing with
2. If the position of the faster drop is behind the slower drop.

We use an array of [pairs](#) to store the position and the time that ith drop would take to reach the end of the pipe. Then we [sort](#) the array according to the position of the drops. Now we have a fair idea of which drops lie behind which drops and their respective time taken to reach the end. More time means less speed and less time means more speed. Now all the drops before a slower drop will mix with it. And all the drops after the slower drop will mix with the next slower drop and so on.

For example if the times to reach the end are- 12, 3, 7, 8, 1 (sorted according to positions)
0th drop is slowest, it won't mix with the next drop

1st drop is faster than the 2nd drop so they will mix and 2nd drop is faster than 3rd drop so all three will mix together. They cannot mix with the 4th drop because that is faster.
So we use a [stack](#) to maintain the local maxima of the times.

No of local maximal + residue(drops after last local maxima) = total no of drops

```
#include <bits/stdc++.h>
using namespace std;
int drops(int length, int position[], int speed[], int n)
{
    // stores position and time taken by a single
    // drop to reach the end as a pair
    vector<pair<int, double>> m(n);

    int i;
    for (i = 0; i < n; i++) {

        // calculates distance needs to be
        // covered by the ith drop
        int p = length - position[i];

        // inserts initial position of the
        // ith drop to the pair
        m[i].first = position[i];

        // inserts time taken by ith drop to reach
        // the end to the pair
        m[i].second = p * 1.0 / speed[i];
    }

    // sorts the pair according to increasing
    // order of their positions
    sort(m.begin(), m.end());
    int k = 0; // counter for no of final drops

    // stack to maintain the next slower drop
```

```
// which might coalesce with the current drop
stack<double> s;

// we traverse the array demo right to left
// to determine the slower drop
for (i = n - 1; i >= 0; i--)
{
    if (s.empty()) {
        s.push(m[i].second);
    }

    // checks for next slower drop
    if (m[i].second > s.top())
    {
        s.pop();
        k++;
        s.push(m[i].second);
    }
}

// calculating residual drops in the pipe
if (!s.empty())
{
    s.pop();
    k++;
}
return k;
}

// driver function
int main()
{
    int length = 12; // length of pipe
    int position[] = { 10, 8, 0, 5, 3 }; // position of droplets
    int speed[] = { 2, 4, 1, 1, 3 }; // speed of each droplets
    int n = sizeof(speed)/sizeof(speed[0]);
    cout << drops(length, position, speed, n);
    return 0;
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/water-drop-problem/>

Chapter 177

What coding habits improve timing in coding contest?

What coding habits improve timing in coding contest? - GeeksforGeeks

It is fun and fruitful to participate in coding challenges and hackathons offline or online. There are lots of prizes and incentives that you can win in these contests. You can showcase your skill and get awarded for it. Hackathons are generally held for 1 to 7 days or a fixed period of time, similar for coding challenges. The one thing that you must have come across is that you have to finish your code in the limited amount of time. Unless and until you compete efficiently and have a time management you will get outrun by the other challengers. You must work on different techniques to speed up your coding. You have to present your work correctly and in less time too.

Great coders & developers are highly reliable. They estimate the time needed to complete the task, communicating this and delivering on time.

These are some of the tips that you can follow to save your precious time while competing in a coding contest:

1. **Read the questions properly twice or thrice :** There are many loopholes set by the problem setter for the user to be trapped in. You have to escape through all of these loopholes. Unless and until you read the question twice or thrice you might get into trouble and you will lose your precious time in thinking what went wrong and correcting those errors. You can refer to this link [to know how to read competitive programming questions](#)

This article is written by me and it helps the reader to know how to read and attempt a coding question and improve his programming skill.

2. **Think and compile :** Sometimes you won't get the program right in just one attempt so you must be patient to know all the errors that you have done, you can't just remove an error and compile again. You have to go through the whole program again and analyze it correctly and stop wasting your limited time.
3. **Upgrade your logic and understanding :** You can run code from other efficient coders so that you can have an idea of what can be done using coding and how it can

be implemented faster and more logically. The program which you compiled in 100 lines of code, might have been compiled by some other efficient coder in 50 or fewer lines which saved him time for attempting other coding questions. It might have also taken less time to get the output through his code. Keep learning from other great programmers.

4. **Find and learn to use tools and languages efficiently :** Every language has new features for coders and developers to use and implement which makes their code more efficient. Discover these new commands, filters, and functions that can be used to solve a problem. Learn how to use it because developers who don't really know their tools waste a lot of the time in debugging and rewriting their code.
5. **Have a creative coding atmosphere :** Get ready with your tools before the competition because you might lose your innumerable seconds just because of some silly stuff like your net might not be connected, your browser is not working properly and so on. You might also get distracted due to these errors and cannot work on your logic. Plan out everything before you sit to conquer the challenge.
6. **Keep growing :** There is always room for improvement even after following these techniques. Don't be stagnant. Keep improving and **practicing** day by day.

Above techniques are my learning and point of views. You can suggest more techniques in the comment section which you have come up with by your self-learning and based on your programming experience. I hope you save some seconds the next time you code.

Related Articles :

1. [How to begin Competitive Programming?](#)
2. [Tips and Tricks for Competitive Programmers](#)

Source

<https://www.geeksforgeeks.org/what-coding-habits-improve-timing-in-coding-contest/>

Chapter 178

What to do at the time of Wrong Answer (WA)?

What to do at the time of Wrong Answer (WA)? - GeeksforGeeks

There is been always a case that a wrong answer gives too much pain rather than **TLE** (**Time Limit Exceed**), as in former you couldn't ascertain about which test case it is failing but in later you can estimate that for which value of N (Total instruction), it would show TLE.

So What to do at that time?

- **Read Question Carefully:** At the first time, when you got WA(Wrong Answer) then always be sure that you have read each and every word and fully understand the question because most of the time we skipped that particular part which is the base of whole question.
- **Check Input/Output formatting:** Mostly Programmers usually forget to add new line or white space according to the requirement of question. So before submitting the solution to Online judge, try to run your program in online compilers like code.geeksforgeeks.org or [ideone](http://ideone.com).
- **Check Algorithm/Logic of program:** Be sure that you are using correct logic that are covering all the test cases or not.
- **Corner test cases** Try to run your code on boundary test cases, if possible like 0, 1, 2 or N.

Avoid Silly Mistakes

- **Initialize variable:** Sometimes we forget to reinitialize variables, arrays after every test case T. For example-
 - Initialize the value of count variable to 0.
 - Setting all value of DP[] array to 0 or -1.

So input same or different test case atleast twice and check if correct output is obtained or not.

- **Data type Overflow:** Always keep the constraint given on value of N or other input in your mind, and make your program in the range of correct data type like int, long long in C/C++ or int, long in JAVA etc to avoid overflow.
- **Modular Problem:** In questions like Modular Arithmetic (answer % MOD),always ensure that answer is not getting a negative value, so try to use $(\text{answer} + \text{MOD}) \% \text{MOD}$, that will cover all possibilities.

Debugging

- **The most important part is debugging:** You can either use inbuilt debugger of Codeblock, Eclipse in C/C++ and JAVA respectively or you can print the variable after each and every line so that you can estimate that your program is running according to your requirement or not.
- **Use Assertion:** If you are going to write a plenty of lines in your code then using assert() is totally worth it. Click [here](#)to read more about how to use Assertion in Competitive programming.
- **Look for Suggestions given:** Although this should be the last step but you must look at the comments given below in which other programmers might have also facing the same problem and have given a hint on how to eradicate this issue.

Ultimately, always try to write a clean code with a small function that have well defined purpose.

Source

<https://www.geeksforgeeks.org/time-wrong-answer-wa/>

Chapter 179

Why is programming important for first year or school students?

Why is programming important for first year or school students? - GeeksforGeeks

Although computer programming was once seen as a skill reserved for geeks and computer nerds, it's now regarded as an essential ability for 21st century learners and is becoming a key component of many curriculums, even in primary schools. And as it is becoming essential to learning programming basics in school itself, you need to be ahead of those basics to prove yourself especially, if you are a budding CS engineer.

Thinking logically is undoubtedly essential and sharpens the chances of getting a elite job even if you are not studying in a Tier-1 college. Learning programming languages and CS basics is a part of your curriculum but what will boost your resume will definitely be **thorough knowledge of Data Structures and Algorithms**.

Learning Data Structures : When and Why?

While it is never too late to learn, but beginning early is always a boon. Learning data structures and algorithms during high school or in first year might be the best time to begin as it'll prepare you to make your placement interviews a cakewalk when you reach third year.

Benefits of beginning at high school or First-Year:

- **Just Coding is not enough, efficient Programming is the need of hour :** In your batch of 100, everyone knows how to code in C++ – then how would you stand out? Answer is simple, Be Efficient. To know how to code is basic and rising above the basics will add-up to you, everywhere. So start learning, how to use data structures and algorithms efficiently alongwith programming languages like C, C++, Java etc
- **Better the algorithms, better the programmer :** Learn to think before you learn to code. It is said in the industry that once you learn “how to think”, switching between different languages is just a matter of little time. So, concentrating earlier on thinking process is ought to give you an edge
- **Getting a Job off-campus might get easier :** This is one of the most important aspect and numerous Geeks prove this fact. There are loads of students who have got

great jobs just by learning the concepts in the right manner and at the right time. [Geeks on the top](#) prove this fact and can help you choose your way.

- **The art of Competitive Programming :** Competitive programming helps you to compete students all over the world to prove your programming skills and can immensely be helpful to get you a job through contests. But what you need to be is excellent in efficient coding and applying right data structures and algorithms at right place. [ACM-ICPC](#) provides a wonderful platform for engineers to showcase their coding skills and progress ahead.
- **Smart people and smart codes are always appreciated :** Good communication skills, impressive personality, nice college may all go into vain if you are not smart enough to analyse and efficiently solve a given problem in an interview. So, be smart physically and mentally too.
- **Right combination of Programming Language and Data Structures is a Big Plus :** While you are focusing on data structures and algorithms, please make sure to be well verse be at least one programming language like C++. Efficiently coding in one/ two programming language is sure shot the right track.

Consider a simple Interview scenario :

Interviewer : Perform sorting on array of N elements

Interviewee : Ok, I can do it using [Selection Sort](#)

The next question you are expected to be asked is :

Interviewer : Can you do it in a better way?

Interviewee : Thinks... Yes I can

Interviewer : How?

Interviewee : I can use [QuickSort](#). As its time complexity is better than Selection sort in average and best case $O(n \log n)$

Interviewer : What if all the elements of Array are already sorted in same order.(worst case of QuickSort)?

Interviewee : Then we might use [Insertion Sort](#) because in Insertion Sort, if element at position i is greater than all elements after it, no movements happen. So the outer loop will loop through all the elements without going inside making time complexity as $O(n)$

and so on.....

It can easily be observed here that practically being well verse with DS and Algos will make you prepared for future interviews and competitive programming challenges ahead.

How to Start ?

Getting started in learning a programming language isn't as daunting as it sounds, nor is it ever too late to learn. You can begin with free, open source sites. Some links that may help you:

- Learning [Data Structures](#) and [Algorithms](#)
- [How to begin with Competitive Programming?](#)
- [C , C++, Java, Python](#)

How Geek Classes* may help you?

While lot of students can learn on their own using e-learning, there would be a bigger lot who love to learn with fellow Engineers in a competitive yet healthy environment. GeekforGeeks has started initiative of class room learning through Geek Classes to help young high schoolers and budding Engineers to enhance their technical skills.

Benefits can be many fold :

1. **Direct Interaction with Industry People** : The advantages of learning directly from Industrial Professionals can be twosome. One, you'll get to know a new way of learning from trending industry trends aspects and two, you'll simple learn how to code efficiently.
2. **Collaborative learning with fellow Programmers** :Unlike individual learning, collaborative learning capitalize on evaluating one another's ideas, monitoring each other's work and getting more and more ways to do one task. More specifically, this model enhances the knowledge can be created within geeks where they actively interact by merely sharing.
3. **Personal attention and Progress Tracking** : If you are amongst those who might need personal attention to improvise, then land up straight in Geek Classes
4. **Internship to add up your skill set** : Of course you join the classes to learn but what if you get an opportunity of "On the Job Learning" too. Yes, at Geek Classes we try to offer internships to students as well, so as to increase their learning diameter. Interns at GeeksforGeeks are now working in firms like Google, Microsoft, Amazon, Samsung, PayTm etc.

*Currently Geek Classes are held in Noida only. We are in-process to begin on-line tutorials soon.

When in doubt, Ask an Expert

So while the thought of coding may make you a little nervous, it's not something to shy away from. Even having a little bit of an understanding is better than having none at all. Plus, it allows you to communicate with actual programmers, gives you an understanding of what it takes to program something and makes you better overall.

If you still have queries, you may please email at geeks.classes@gmail.com or Call Ayushmaan at 8375042560 to help you!!

Source

<https://www.geeksforgeeks.org/why-is-programming-important-for-first-year-or-school-students/>

Chapter 180

Why is python best suited for Competitive Coding?

Why is python best suited for Competitive Coding? - GeeksforGeeks

When it comes to **Product Based Companies**, they need good coders and one needs to clear the **Competitive Coding** round in order to reach the interview rounds. Competitive coding is one such platform which will test your mental ability and speed at the same time.

Who should read this?

Any programmer who still hasn't tried python for Competitive Coding MUST give this article a read. This should clear up any doubts one has before shifting to python. No matter how comfortable a programming language may seem to you right now Python is bound to feel even better. Python has a tendency of sticking to people like a bad habit !!

SPEED is a factor where python is second to none. The amount of code to be typed decreases drastically in comparison to conventional programming languages like C, C++, JAVA. Another most important point is that python arms its users with a **wide variety of functionality, packages and libraries** which act like a supplement to the programmer's mental ability.

Ultimately the best thing about python is that its very Simple and we need not waste much time on trivial matters like input,output etc. It helps shift our focus to the problem at hand.

Here I'm gonna list out some of my favourite features of python which I'm sure will encourage you to start trying python for Competitive Coding.

1. Variable Independence

Python doesn't require us to declare variables and their Data-Types before using them.

This also gives us the flexibility of range as long as its within reasonable limits of the Hardware i.e. no need to worry about integer and long integer. Type conversion is internally handled with flawless results.

Amazing Fact !!

For nested loops in python we can use the same variable name in both inner and outer for-loop variables without fear of inconsistent data or any errors !!

2. Common Functions like sorted, min, max, count etc.

The min/max function helps us to find the minimum/maximum element from a list. The Sorted function allows us to sort a list and **count function** helps us to count the number of occurrences of a particular element in a list.

The **best thing is that we can be rest assured that the python libraries use the best possible algorithms for each of the above operations.** For example the sorted function is a very special sorting algorithm called **TIMSORT** that has a worst case time complexity of $O(n \log n)$ which is the best a sorting algorithm can offer.

Reference:[Python sorting algorithm](#)

```
# Python code to demonstrate working of min(),
# max(), sorted() and count()
arr = [10, 76, 87, 45, 22, 87, 90, 87, 66, 84, 87]

print("Maximum = ",max(arr))
print("Minimum = ",min(arr))
print("The sorted array is = ",sorted(arr))
print('Number of occurrences of 87 is = ',arr.count(87))
```

Output:

```
('Maximum = ', 90)
('Minimum = ', 10)
('The sorted array is = ', [10, 22, 45, 66, 76, 84, 87, 87, 87, 90])
('Number of occurrences of 87 is = ', 4)
```

3. Lists in python combine the best aspects of arrays and linked lists.

Python lists provide the unique functionality of deleting specific elements while keeping the memory locations in a contiguous manner. **This feature renders the concept of Linked lists null and void.** Its like a linked list on STEROIDS ! Moreover Insertions can be performed at any desired locations.

```
# Python code to demonstrate list operations
arr = [00, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```

```
# deletion via index position
del arr[5]
print(arr)

# deletion via specifying particular element
arr.remove(22)
print(arr)

# insertion at any arbitrary position
arr[-1] = "A random number"
print(arr)

# concept of sub-lists
k = arr[:2]
print(k)
```

Output:

```
[0, 11, 22, 33, 44, 66, 77, 88, 99]
[0, 11, 33, 44, 66, 77, 88, 99]
[0, 11, 33, 44, 66, 77, 88, 'A random number']
[0, 11]
```

4. Unique list operations – Backtracking , Sub-Lists.

In case we are not sure about the list size then we can use the index position of -1 to access the last element. Similarly -2 can be used for second last element and so on. Thus we can back track a list. Also we don't have to specify any particular list size so it also works like a dynamic allocation array.

A specific portion of a list can be extracted without having to traverse the list as is seen in the above example. **A very astonishing fact about lists is that it can hold different datatypes.** Gone are the days where lists used to be a homogeneous collection of data elements!!

5. Functions can return more than one value.

Typically functions in other programming languages can return only one value but **in python we can return more than one value!!** as is seen in the following code snippet.

```
# Python code to demonstrate that a function
# can easily return multiple values.
def multi_return(*arr):
    k1 = arr[0]
    k2 = arr[1]
    return k1,k2

a,b = multi_return(11,22)
```

```
print(a, ' ', b)

a,b = multi_return(55,66,77,88,99)
print(a, ' ', b)
```

Output:

```
11    22
55    66
```

6. Flexible number of arguments to a function.

Arguemnts to a function may be passed in the form of a list whose size may vary every time we need to call the function. In the above example we first called the function with 2 arguements and then with 5 arguements!!

7. If else and for loops are much more User Friendly.

8. Code Indentation.

Python blocks of code are distinguished on the basis of their indentation. This provides better code readability and instills in us a good habit of indenting our code.

9. Concept of Sets and Dictionaries.

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Its like a list that doesn't allow duplicate elements.

A dictionary is like a list whose **values** can be accessed by user defined **keys** instead of conventional numeric index values.

```
# Python code to demonstrate use of dictionaries
# and sets.

a = {'a','b','c','d','e','a'}

# the second 'a' is dropped to avoid repetition
print(a)

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print("dict['Name']: ", dict['Name'])
print("dict['Age']: ", dict['Age'])
```

Output:

```
{'d', 'a', 'e', 'b', 'c'}
dict['Name']: Zara
dict['Age']: 7
```

10. Robust input statements.

In competitive coding we are often required to take 'n' space separated integers as input and preferably save them in a list/array. Python provides functionality to do it all in a single line of code.!!

```
# Python code to demonstrate how to take space
# separated inputs.
arr = [int(a) for a in input().strip().split(' ')]

print(arr)
```

Source

<https://www.geeksforgeeks.org/python-best-suited-competitive-coding/>

Chapter 181

Writing C/C++ code efficiently in Competitive programming

Writing C/C++ code efficiently in Competitive programming - GeeksforGeeks

First of all you need to know about [Template](#), [Macros](#) and [Vectors](#) before moving on the next phase!

- Templates are the foundation of generic programming, which involve writing code in a way that is independent of any particular type.
- A Macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro.
- Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

So we can use these powerful tools for writing our code in an effective way.

Some of the cool tricks that may be used in Competitive programming are given as follows:

1. **Using Range based for loop:** This is very cool feature in C++11 and would be considered best if you want to iterate from begin to end. This code shows how to use ranged for loops to iterate through an array and a vector:

```
// C++ program to demonstrate range based for
// loops for accessing vector and array elements
#include<iostream>
#include <vector>
using namespace std;

int main()
{
```

```
// Create a vector object that
// contains 5 elements
vector<int> vec = {0, 1, 2, 3, 4};

// Type inference by reference using auto.
// Range based loops are preferred when no
// modification is needed in value
for (const auto &value : vec)
    cout << value << ' ';

cout << '\n';

// Basic 5 element integer array
int array[] = {1, 2, 3, 4, 5};
for (const auto &value: array)
    cout << value << " ";

return 0;
}
```

Output:

```
0 1 2 3 4
1 2 3 4 5
```

2. **Initializer list:** This type is used to access the values in a C++ initialization list. Here the objects of this type are automatically constructed by the compiler from initialization list declarations, which is a list of comma-separated elements enclosed in braces.

```
#include<iostream>

template<typename T>
void printList(std::initializer_list<T> text)
{
    for (const auto & value: text)
        std::cout << value << " ";
}

// Driver program
int main()
{
    // Initialization list
    printList( {"One", "Two", "Three"} );
    return 0;
}
```

Output:

One Two Three

3. **Assigning Maximum or Minimum value:** This one is useful to avoid extra effort in writing max() or min() function.

```
#include<iostream>

// Call by reference is used in x
template<typename T, typename U>
static inline void amin(T &x, U y)
{
    if (y < x)
        x = y;
}

// call by reference is used in x
template<typename T, typename U>
static inline void amax(T &x, U y)
{
    if (x < y)
        x = y;
}

// Driver program to find the Maximum and Minimum value
int main()
{
    int max_val = 0, min_val = 1e5;
    int array[] = {4, -5, 6, -9, 2, 11};

    for (auto const &val: array)

        // Same as max_val = max (max_val, val)
        // Same as min_val = min (min_val, val)
        amax(max_val, val), amin (min_val, val);

    std::cout << "Max value = " << max_val << "\n"
           << "Min value = " << min_val;
    return 0;
}
```

Output:

```
Max value = 11
Min value = -9
```

4. **Fast Input/Output in C/C++:** In Competitive programming, you must read Input/Output as fast as possible to save valuable time.

```
#include <bits/stdc++.h>

template<typename T> void scan(T &x)
{
    x = 0;
    bool neg = 0;
    register T c = getchar();

    if (c == '-')
        neg = 1, c = getchar();

    while ((c < 48) || (c > 57))
        c = getchar();

    for ( ; c < 48||c > 57 ; c = getchar());
        for ( ; c > 47 && c < 58; c = getchar() )
            x= (x << 3) + ( x << 1 ) + ( c & 15 );

    if (neg) x *= -1;
}

template<typename T> void print(T n)
{
    bool neg = 0;

    if (n < 0)
        n *= -1, neg = 1;

    char snum[65];
    int i = 0;
    do
    {
        snum[i++] = n % 10 + '0';
        n /= 10;
    }

    while (n);
    --i;

    if (neg)
        putchar('-');
```

```
    while (i >= 0)
        putchar(snum[i--]);

    putchar('\n');
}

// Driver Program
int main()
{
    int value;

    // Taking input
    scan(value);

    // Printing output
    print(value);
    return 0;
}
```

Input: 756
Output: 756

To know more about fast input and output [Read this article](#).

5. **Using Macros as for loop:** Perhaps, it would not be good to use such macros as it would reduce the readability of code but for writing fast code you can take that risk!

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i,n) for (i = 0; i < n; ++i)
#define REP(i,k,n) for (i = k; i <= n; ++i)
#define REPR(i,k,n) for (i = k; i >= n; --i)

// Driver program to test above Macros
int main()
{
    int i;
    int array[] = {4, 5, 6, 9, 22, 11};
    int size= sizeof(array)/sizeof(array[0]);

    // Default 0 index based loop
    rep(i, size)
        cout << array[i] << " ";
    cout<<"\n";
```

```
// Starting index based loop
REP(i, 1, size-1)
    cout << array[i] << " ";
cout<<"\n";

// Reverse for loop
REPR(i, size-1,0)
    cout << array[i] << " ";
return 0;
}
```

Output

```
4 5 6 9 22 11
5 6 9 22 11
11 22 9 6 5 4
```

6. **Using “bits/stdc++.h”:** Instead of adding tons of #include lines, just use #include. The files includes all the header files you’ll need in competitive programming, saving a lot of your time.
7. **Containers:** Using various containers like vector, list, map etc enables one to use the pre-defined functions and reduces the size of code considerably (more often than not)
8. **Fast cin and cout:** If you use cin and cout for I/O, just add the following line just after the main().

```
std::ios_base::sync_with_stdio(false);
```

9. **auto:** Using auto to declare datatypes can save lot of time during programming contests. When a variable is defined as auto, compiler determines its type during compile-time.
10. **Libraries and pre-defined functions:** Using builtin functions such as __gcd(A,B), swap, __builtin_popcount(R), __builtin_clz(R) etc wherever that can be applied. Try to learn different functions available in [algorithm](#) library of C++. They are useful most of the times in programs

Ultimately, by using these smart tricks you can easily write code in a minimum amount of time and words.

Source

<https://www.geeksforgeeks.org/writing-cc-code-efficiently-in-competitive-programming/>

Chapter 182

Writing code faster in C++ STL

Writing code faster in C++ STL - GeeksforGeeks

You are wondering some time coder write his code in 2 min or less how? No kidding! Although this tip may not be very useful for competitions such as ICPC or IOI. But there are recent ICPCs ranks where rank i and rank $i + 1$ are just separated by few minutes. When you can solve the same number of problems as your competitor, it is now down to coding skill and ... typing speed.

Try this typing test at typingtest.com and follow the instructions there on how to improve your typing skill.

Many of them uses typedefs, shortcuts, or macros that are commonly used by competitive programmers to speed up the coding time. In this short section, we list down several examples as below.

```
// C++ STL shortcuts for
// typing codes fast
#include <bits/stdc++.h>

// Shortcuts for "common" data types in contests
typedef long long ll;
typedef vector<int> vi;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef set<int> si;
typedef map<string, int> msi;

// To simplify repetitions/loops, Note: define your
// loop style and stick with it!
#define REP(i, a, b) \
for(int i = int(a); i <= int(b); i++) // a to b, and variable i is local!
```

```
#define TRvi(c, it) \
for(vi::iterator it = (c).begin(); it != (c).end(); it++)
#define TRvii(c, it) \
for(vii::iterator it = (c).begin(); it != (c).end(); it++)
#define TRmsi(c, it) \
for(msi::iterator it = (c).begin(); it != (c).end(); it++)

#define INF 2000000000 // 2 billion

// If you need to recall how to use memset:
#define MEMSET_INF 127 // about 2B
#define MEMSET_HALF_INF 63 // about 1B

// memset(dist, MEMSET_INF, sizeof dist); // useful to initialize shortest path distances
// memset(dp_memo, -1, sizeof dp_memo); // useful to initialize DP memoization table
// memset(arr, 0, sizeof arr); // useful to clear array of integers

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // YOUR CODE GOES HERE

    return 0;
}
```

Here, `typedef` and `#define` are used to shorten the code and you can use as per your need in `main` method `ios_base::sync_with_stdio(false);` and `cin.tie(NULL);` use for fast io which is reduce the time of running code.

Related Articles:

- [Writing C/C++ code efficiently in Competitive programming](#)
- [How to begin with Competitive Programming?](#)
- [Tips and Tricks for Competitive Programmers | Set 1 \(For Beginners\)](#)

Source

<https://www.geeksforgeeks.org/writing-code-faster-in-c-stl/>

Chapter 183

getchar_unlocked() – faster input in C/C++ for Competitive Programming

getchar_unlocked() - faster input in C/C++ for Competitive Programming - GeeksforGeeks
[getchar_unlocked\(\)](#) is similar to getchar() with the exception that it is not thread safe. Below is an example code.

```
// A simple C program to demonstrate
// working of getchar_unlocked()
#include <stdio.h>
int main()
{
    // Syntax is same as getchar()
    char c = getchar_unlocked();

    printf("Entered character is %c", c);

    return 0;
}
```

Input: g
Output: Entered character is g

Following are some important points:

1. Since it is not thread safe, all overheads of mutual exclusion are avoided and it is faster than getchar().

2. Can be especially useful for competitive programming problems with “*Warning: Large I/O data, be careful with certain languages (though most should be OK if the algorithm is well designed)*”.
3. There is no issue with using `getchar_unlocked()` even in multithreaded environment as long as the thread using it is the only thread accessing file object
4. One more difference with `getchar()` is, it is not a C standard library function, but a POSIX function. It may not work on Windows based compilers.
5. *It is a known fact than `scanf()` is faster than `cin` and `getchar()` is faster than `scanf()` in general. `getchar_unlocked()` is faster than `getchar()`, hence fastest of all.*
6. Similarly, there are `getc_unlocked()` `putc_unlocked()`, and `putchar_unlocked()` which are non-thread-safe versions of `getc()`, `putc()` and `putchar()` respectively.

```
// A simple C program to demonstrate
// working of putchar_unlocked()
#include <stdio.h>
int main()
{
    // Syntax is same as getchar()
    char c = getchar_unlocked();

    putchar_unlocked(c);

    return 0;
}
```

Input: g
Output: g

As an exercise, the readers may try solutions given [here](#) with `getchar_unlocked()` and compare performance with `getchar()`.

This article is contributed by **Ayush Saluja**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/getchar_unlocked-faster-input-cc-competitive-programming/