CP Learning

Websites for learning

👉 To find any sequence use this website: https://oeis.org/

Short Tricks

▼ Count Odds Numbers Between range L-R

```
int countOdd(int L, int R){

    int N = (R - L) / 2;

    // if either R or L is odd
    if (R % 2 != 0 || L % 2 != 0)
        N += 1;

    return N;
}
```

▼ Find the Least Significant Bit of a Number OR the minimum value of power of 2 divide the N

```
int a = 16;
cout << (a & (-a) << endl;
// output << 16

int a = 60;
cout << (a & -a) << endl;
// 4

// So, 4 is the minimum value Power of 2 divide the 60

// We can use log2(); to find the bit number;
```

▼ Most greatest Number divisible by 'a' and less than 'n'

Identify the highest number divisible by a that is less than or equal to n

$$Result = \lfloor a/n \rfloor \times a$$

```
int n, a;
cout << (n/a) * a;

// n = 11, a = 5;
// 11 / 5 = 2
// 2 * 5 = 10
```

▼ log2 function

```
int s(int a) {
    int ans=0;
    while (a>1) {
        ans++;
        a=(a)/2;
    }
    return ans;
}

// ### Short Cut

int s(int a){
   return __lg(a);
}

// log2() in ceil

int s(int a) {
    int ans=0;
    while (a>1) {
        ans++;
        a=(a+1)/2;
    }
    return ans;
}

// ## Shortcut

int s(int a){
   return __lg(2 * a - 1);
}
```

▼ Bitmask Equation

👉 if a ^ b = x then b = x ^ a
if a|b = x then, b = x & ~a
if a & b = x, then be can't be unique value,
but, the maximum value of b can be,
b = x | ~a

▼ How to check segments that intersect

To determine when two line segments [a,b] and [c,d] intersect (on a 1D number line), you need to check whether the intervals **overlap**.

The condition for intersection is:

$$max(a,c) \le min(b,d)$$

if there are many segments use multiset.

▼ How calculate nC2 in real time using Map

If we have N values, and we want to find nC2 using map,

```
map<int,int> mp;
int ans = 0;

for(int i = 0; i < n; i++){
int x; cin >> x;
ans += mp[x]++;
}
```

▼ Lambda Expression for DFS

```
function<void(int)> dfs = [&](int x){
 if(x == 0) return;
 cout << x << endl;
 dfs(x - 1);
};

dfs(10);

// function<return_type(arguments type)>
```

▼ Priority Queue Comparator

```
auto cmp = [](pair<int, int>& a, pair<int, int>& b) {
    return a.second > b.second;
};
```

```
priority_queue<
    pair<int, int>,
    vector<pair<int, int>>,
    decltype(cmp)
> pq(cmp);
```

▼ Count how many odd multiples of k are in [L, R]

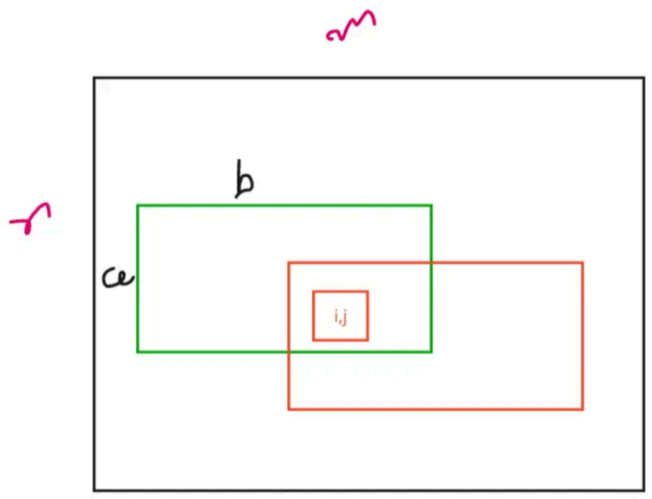
```
// Count how many odd multiples of k are in [L, R]
ll countOddMultiples(ll L, ll R, ll k) {
    // First multiple of k ≥ L
    ll start = (L + k - 1) / k * k;
    // First odd multiple ≥ L
    if (start % 2 == 0) start += k;
    if (start > R) return 0;

    // Last multiple of k ≤ R
    ll end = R / k * k;
    if (end % 2 == 0) end -= k;
    if (end < L) return 0;

    return ((end - start) / (2 * k)) + 1;
}
```

▼ Unordered_map Pair declaration

```
struct pair_hash{
    size_t operator()(const pair<int, int>& p) const {
        return hash<int>()(p.first) ^ (hash<int>()(p.second) << 1);
    }
};
//unordered_map<pair<int,int>, int, pair_hash> m;
```

2D Array

▼ If we have n*m matrix and we have a*b rectangle, and if we put this rectangle to the matrix, so that it does not overflow, we have to find, how many rectangles will cover the every cells.



```
int n = 3;
    int m = 3;
    int a = 2, b = 2;
   for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){

      int lbx = max(0, i - b + 1);
      int ubx = min(i, n - b);
      int lby = max(0, j - a + 1);
      int uby = min(j, m - a);
      cout << (ubx - lbx + 1) * (uby - lby + 1) << " ";
    }
    cout << endl;
   }
```

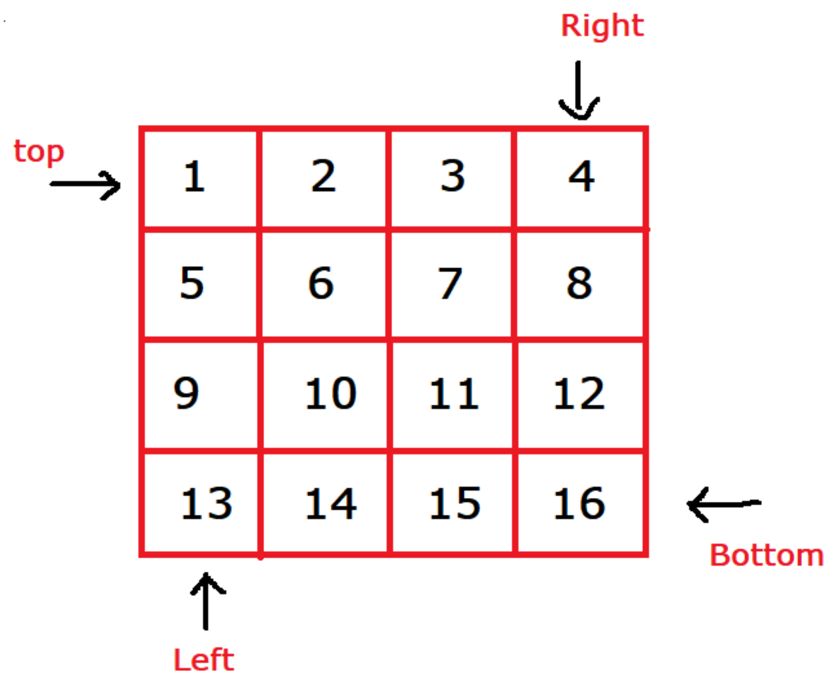
▼ If we have a rectangle of N * M, and we want know how many sub-rectangle are there of area of K

```
int countSubrectangles(int n, int m, int k) {
    int count = 0;
    for (int a = 1; a * a <= k; ++a) {
        if (k % a == 0) {
            int b = k / a;
            // (a, b) subrectangle
            if (a <= n && b <= m) {
                count += (n - a + 1) * (m - b + 1);
            }
            // (b, a) subrectangle, only if a != b
            if (a != b && b <= n && a <= m) {
                count += (n - b + 1) * (m - a + 1);
            }
        }
    }
    return count;
  }
```

PDF

attachment:b50936c0-b8fa-4f89-826d-f6cd642ddb5e:subrectangle_count.pdf

▼ **Spiral Traversal of Matrix**



```
#include <bits/stdc++.h>

using namespace std;

vector<int> printSpiral(vector<vector<int>> mat) {

  // Define ans array to store the result.
  vector<int> ans;

  int n = mat.size(); // no. of nows
  int m = mat[0].size(); // no. of columns

  // Initialize the pointers reqd for traversal.
  int top = 0, left = 0, bottom = n - 1, right = m - 1;
  // Loop until all elements are not traversed.
  while (top <= bottom && left <= right) {

    // For moving left to right
    for (int i = left; i <= right; i++)
      ans.push_back(mat[top][i]);

    top++;

    // For moving top to bottom.
    for (int i = top; i <= bottom; i++)
      ans.push_back(mat[i][right]);

    right--;

    // For moving right to left.
    if (top <= bottom) {
      for (int i = right; i >= left; i--)
       ans.push_back(mat[bottom][i]);

      bottom--;
    }

    // For moving bottom to top.
    if (left <= right) {
      for (int i = bottom; i >= top; i--)
        ans.push_back(mat[i][left]);

      left++;
    }
  }
  return ans;
}

int main() {

  //Matrix initialization.
  vector<vector<int>> mat   {{1, 2, 3, 4},
                             {5, 6, 7, 8},
                             {9, 10, 11, 12},
                             {13, 14, 15, 16}};

  vector<int> ans = printSpiral(mat);

  for(int i = 0;i<ans.size();i++){

      cout<<ans[i]<<" ";
  }
  cout<<endl;

  return 0;
}
```

Math

▼ Sequence

Arithmetic Progression

▼ a + ad + a2d + a3d + ...

$$a + a \cdot d + a \cdot 2d + a \cdot 3d + ...$$

Nth Term (General Term):

$$a_n = a + (n - 1) \times d$$

where:

- a is the first term,
- d is the common difference, and
- n is the term number.

Sum of the First N Terms:

$$S_n = 2n \cdot (2a + (n - 1) \cdot d)$$

$$Sn = \frac{n}{2} \cdot (a + a_n)$$

- S_n is the sum of the first n terms, n
- a is the first term,
- a_n is the nth term, and
- d is the common difference.

Geometric Progression

▼ a + rd + r^2d + ...

Nth Term (General Term):

$$a_n = a \cdot r^{n-1}$$

where:

- aaa is the first term,

- rrr is the common ratio, and
- nnn is the term number.

Sum of the First N Terms:

$$S_n = a \cdot \frac{r^n - 1}{r - 1} \quad ; r > 1$$

$$S_n = a \cdot \frac{1 - r^n}{1 - r} \quad ; r < 1$$

▼ **Pythagorean Triplets**

Any Pytagorean Triplets can be generated by these equation, where m > n.
$a = m^2 - n^2, b = 2mn, c = m^2 + n^2, (m > n)$

Number Theory

▼ Bezout's Identity

💡 For any two integers a and b , **not both zero**, there exist integers x and y such that: gcd(a,b)=ax+by

▼ Modular Arithmetic

📌 You can only change a number **X** into **Y** if and only if either:

1. x≡y(mod k) or
2. x+y≡ 0 (mod k)

See solution C: https://codeforces.com/blog/entry/145439

▼ McNugget Theorem

👉 If you have two **relatively prime(gcd(m,n) = 1)** positive integers m and n, the largest number you **cannot** express as a nonnegative combination of m and n is:

mn - m - n

For **two** positive coprime pack sizes m and n, the number of positive integers **not** representable as
am + bn with a,b ≥ 0 is

(m-1)(n-1) / 2

▼ Catalan Number

https://cp-algorithms.com/combinatorics/catalan-numbers.html

http://www.geometer.org/mathcircles/catalan.pdf

```
const int MOD = (int)1e9 + 7;
const int MAX = (int)1e5;
int catalan[MAX];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

▼ GCD Sum function

link: https://forthright48.com/gcd-sum-function/#more-502

$$g(n) = gcd(1,n) + gcd(2,n) + gcd(3,n) + \cdots + gcd(n,n) = \sum_{i=1}^{n} gcd(i,n)$$

$$\begin{aligned} g(6) &= gcd(1,6) + gcd(2,6) + g(3,6) + gcd(4,6) + gcd(5,6) + gcd(6,6) && (1)\\ &= 1+2+3+2+1+6 && (2)\\ &= 15 && (3)\end{aligned}$$

In short, there is a direct formula for calculating the value of **g(n)**.

If the prime factorization of n is $p_1^{a_1} \times p_2^{a_2} \times \dots p_k^{a_k}$

$$g(n) = \prod_{i=0}^{k} (a_i+1)p_i^{a_i} – a_i p^{a_i - 1}$$

```
int const MAXN = (int)1e7 + 5;

vector<int> spf(MAXN + 1, 1);

//from
//https://www.geeksforgeeks.org/dsa/prime-factorization-using-sieve-olog-n-multiple-queri
es/
```

```
// Time Complexity : O(nloglogn)

void sieve()
{

    spf[0] = 0;
    for (int i = 2; i <= MAXN; i++) {
        if (spf[i] == 1) {
            for (int j = i; j <= MAXN; j += i) {
                if (spf[j]== 1)
                    spf[j] = i;
            }
        }
    }
}

//log(n)
map<int,int> getFactorization(int x)
{
    map<int,int> mp;
    while (x != 1) {
        mp[spf[x]]++;
        x = x / spf[x];
    }
    return mp;
}

int _pow(int base, int power){
  int res = 1;
  while(power){
     if(power&1){
      res = res * base;
      power--;
     }else{
      base = base * base;
      power =  power / 2;
     }
  }
  return res;
}

int gcdSumFunction(int n){
   map<int,int> pf = getFactorization(n);

   int ans = 1;

   for(auto i: pf){
      ans *= ((i.second  + 1) * _pow(i.first, i.second)) - ((i.second) * _pow(i.first, i.second - 1));
   }
  return ans;
}
```

String

▼ Find Lexicographically largest subsequence

```
vector<int> ans;
for (ll i = 0; i < n; i++) {
    while (!ans.empty() && (str[i] > str[ans.back()])) ans.pop_back();
    ans.push_back(i);
 }
```

Miscellaneous

▼ **MEX (minimal excluded) of a sequence**

https://cp-algorithms.com/sequences/mex.html

▼ Generate All Palicdromic Numbers upto N

URL: https://www.geeksforgeeks.org/generate-palindromic-numbers-less-n/

```
vector<int> pnums;

 int createPalindrome(int input, int b, bool isOdd)
{
   int n = input;
   int palin = input;

   if (isOdd)
      n /= b;

   while (n > 0)
   {
      palin = palin * b + (n % b);
      n /= b;
   }
   return palin;
}
void generatePalindromes(int n)
{
   int number;
   for (int j = 0; j < 2; j++)
   {
      int i = 1;
      while ((number = createPalindrome(i, 10, j % 2)) < n)
```

```
        {
            pnums.push_back(number);
            i++;
        }
    }

    // Have to sort, bcz, generates numbers ar not sorted;

    sort(all(pnums));
}
```

Bitmask

▼ **Sum of XOR of all possible subsets of a given array**

if the length of the array is N, and the OR of each element of array is X,

the sum = X * (2 ^ N - 1)

Example problem: https://codeforces.com/problemset/problem/1614/C

```
int subsetXORSum(vector<int> &arr) {
    int n = arr.size();
    int bits = 0;

    // Finding bitwise OR of all elements
    for (int i=0; i < n; ++i)
        bits |= arr[i];

    int ans = bits * (1<<N-1);

    return ans;
}
```

▼ CF Blog

https://codeforces.com/blog/entry/72437

▼ BitMask Equations

💡 **Some properties of bitwise operations:**

- a|b = a⊕b + a&b
- a⊕(a&b) = (a|b)⊕b
- b⊕(a&b) = (a|b)⊕a
- (a&b)⊕(a|b) = a⊕b

Addition:

- a+b = a|b + a&b
- a+b = a⊕b + 2(a&b)

Subtraction:

- a-b = (a⊕(a&b))-((a|b)⊕a)
- a-b = ((a|b)⊕b)-((a|b)⊕a)
- a-b = (a⊕(a&b))-(b⊕(a&b))
- a-b = ((a|b)⊕b)-(b⊕(a&b))

Combinatorics

▼ Pigeonhole Principle

📌 if i have, N elements, and k box. and if i put, element to boxex. them, there will be minimum, ceil(N/k) element in a single box.
i.e N= mk + 1
m = how many object per box (minimum)

✅ Theorem:
Every sequence of **n^2 +1** distinct real numbers contains a subsequence of length **n + 1** that is either strictly increasing or strictly decreasing.
n = 3; 3^2 + 1 = 10;
1, 10, 100, 3, 4, 9, 11, 20, 23, 33, 200
there is always, subsequence of length 3 + 1 which is strictly increasing or strictly decreasing.

▼ Intersection points of regular N' gons

Intersection points of regular N' gons: (N must be odd)

$$(nC4) = n(n-1)(n-2)(n-3)/24$$

Number of diagonals:

$$nC2 - n = n(n-3)/2$$

Tricks

▼ **Logarithmic Subarray Aggregator (LSA Trick)**

https://youkn0wwho.academy/topic-list/logarithmic_bruteforce

Problem: https://codeforces.com/gym/106057/problem/C

Sol: https://github.com/MHKhanCou/CoU-IUPC-2025-Regional/blob/main/problemC__Prime_Dominion_(lca).cpp

editorial: https://codeforces.com/gym/106057/attachments/download/33229/CoU_IUPC_2025_Editorial.pdf

Graph

▼ Topological sorting

Topological sorting:

DFS: https://www.youtube.com/watch?v=5lZ0iJMrUMk

BFS: https://www.youtube.com/watch?v=73sneFXuTEg

https://codeforces.com/gym/105981/problem/E

https://codeforces.com/contest/2143/problem/C

```
// Including necessary header file
#include <bits/stdc++.h>
using namespace std;

// We mainly take input graph as a set of edges. This function is
// mainly a utility function to convert the edges to an adjacency
// list
vector<vector<int>> constructadj(int V,vector<vector<int>> &edges){

    // Graph represented as an adjacency list
    vector<vector<int> > adj(V);

    // Constructing adjacency list
    for (auto i : edges) {
        adj[i[0]].push_back(i[1]);
    }

    return adj;
}

// Function to return list containing vertices in
// Topological order.
vector<int> topologicalSort(int V, vector<vector<int> >& edges)
{
    vector<vector<int>> adj = constructadj(V,edges);

    // Vector to store indegree of each vertex
    vector<int> indegree(V);
    for (int i = 0; i < V; i++) {
        for (auto it : adj[i]) {
            indegree[it]++;
        }
    }
    // Queue to store vertices with indegree 0
    queue<int> q;
    for (int i = 0; i < V; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }
    vector<int> result;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        result.push_back(node);

        // Decrease indegree of adjacent vertices as the
        // current node is in topological order
        for (auto it : adj[node]) {
            indegree[it]--;

            // If indegree becomes 0, push it to the queue
            if (indegree[it] == 0)
                q.push(it);
        }
    }

    // Check for cycle
    if (result.size() != V) {
        cout << "Graph contains cycle!" << endl;
        return {};
    }

    return result;
}

int main()
{

    // Number of nodes
    int V = 6;

    // Edges
    vector<vector<int> > edges
        = {{0, 1}, {1, 2}, {2, 3},
           {4, 5}, {5, 1}, {5, 2}};

    vector<int> result = topologicalSort(V, edges);

    // Displaying result
    for (auto i : result) {
        cout << i << " ";
    }

    return 0;
}
```