Algorithmic Problem Solving Final Report

Mukunda Jajoo
muj4340@rit.edu
Rochester Institute of Technology

Faculty Sponsor
Dr. Ivona Bezáková

Table of Contents

Introduction

This document details the report of the Algorithmic Problem Solving Independent Study. The goal of this study was to study advanced methods of algorithmic design, analysis, and implementation of efficient solutions to solve problems. The problems for the study were proposed and agreed upon by taking inputs from all the students in the group in consultation with the sponsoring faculty. Problems of varying levels of difficulty were selected from online competitive judging software like LeetCode, HackerRank, CodeChef and Google Code Jam. The target number of problems to be solved was 30 which we complete successfully passing all the test cases and discuss in the following sections below. Further, we solve a few extra problems we find interesting.

For each problem, we include the following,

- The source (a link to the problem source)
- Problem description (*as it appears on the source*)
- The approaches taken to solve the problem
- A link to our code on GitHub and the online platform submission
- The run time analysis of the solutions
- Screenshots of the test cases cleared

Project Resources

Online Judge Platforms used:

- Leetcode (https://leetcode.com/)
- HackerRank (https://www.hackerrank.com/)
- CodeChef (https://www.codechef.com/)
- Google Code Jam (https://codingcompetitions.withgoogle.com/codejam)

Username and Profiles:

- Leetcode - user1524a / https://leetcode.com/user1524a
- HackerRank - MukundaJ / https://www.hackerrank.com/MukundaJ
- CodeChef - chefmukunda / https://www.codechef.com/users/chefmukunda
- Google Code Jam - mukundaj / jajoo.mukunda@gmail.com

GitHub Repository - https://github.com/MukundaJ/APS.git

Problem 1

Bigger is Greater
https://www.hackerrank.com/challenges/bigger-is-greater/problem

Problem Description

Lexicographical order is often known as alphabetical order when dealing with strings. A string is greater than another string if it comes later in a lexicographically sorted list.
Given a word, create a new word by swapping some or all of its characters. This new word must meet two criteria:

- *It must be greater than the original word*
- *It must be the smallest word that meets the first condition*

For example, given the word, the next largest word is.
Complete the function biggerIsGreater below to create and return the new string meeting the criteria. If it is not possible, return no answer.

Function Description

Complete the biggerIsGreater function in the editor below. It should return the smallest lexicographically higher string possible from the given string or no answer.
biggerIsGreater has the following parameter(s):

- *w: a string*

Input Format

The first line of input contains, the number of test cases.
Each of the next lines contains.
Constraints

- $1 \leq T \leq 10^5$
- $1 \leq w \leq 100$
- w *will contain only letters in the range ascii[a..z].*

Output Format

For each test case, output the string meeting the criteria. If no answer exists, print no answer.

Approach Description

This is a string problem that asks us to find the next lexicographically greater string, if possible, by only rearranging characters of a given an input string.

A brute force approach to the problem would be to list out all the arrangements of the string, then we sort this list having all the arrangements which would take $\mathcal{O}(klog(k))$ time where k is the length of the list (We observe that a string of length n has n! such arrangements so k would be n!). Then we simply scan across the sorted list to find the index of our input string, the next string in the list different from the input string would then be our required solution (i.e. the next lexicographically greater string formed by rearranging characters of the given string). A quick optimization would be to search for our input string in the sorted list using binary search although this still. However, this approach is still extremely slow, costing $\mathcal{O}(n!log(n!))$ time.

We can do better, much better in fact. The most important observation of this approach is that we want to 'increase' the string as little as possible. Just like counting up in numbers, we try to modify the leftmost elements of the string leaving the rightmost ones unchanged.

First, we identify the longest non-increasing suffix in the string, since this suffix is at the highest possible arrangement already, we cannot make the next greater arrangement by modifying it. This takes us $\mathcal{O}(n)$ time, where n is the length of the input string.
If the entire string is non-decreasing going left to right in the string, then this is the lexicographically largest string that can be made from those characters (e.g. 'cbaa'), in which case we return 'no answer' since the string is already at the highest permutation it can be.
If not, we record the first decreasing element (i.e. the one just before / to the left of the start of the suffix) and call it the pivot.
There has to be one such element since we already take care of the case where the entire string is non-decreasing. Let us call everything that is not a part of the suffix as the prefix (Observe that the last element of the prefix is the pivot).

Second, we try to minimize the prefix by finding an element in the suffix that is just greater than the pivot, then we swap the pivot and this element thus minimizing the prefix.

Finally, we need to sort the suffix so that it is as low as possible. What's even better is that we can completely avoid sorting and just reverse the suffix by leveraging the fact that the previous replacement/swap in the suffix follows the non-increasing order. This results in the next lexicographically permutation that we need.

The following blog is a helpful resource to visualize the problem.

Run Time Analysis

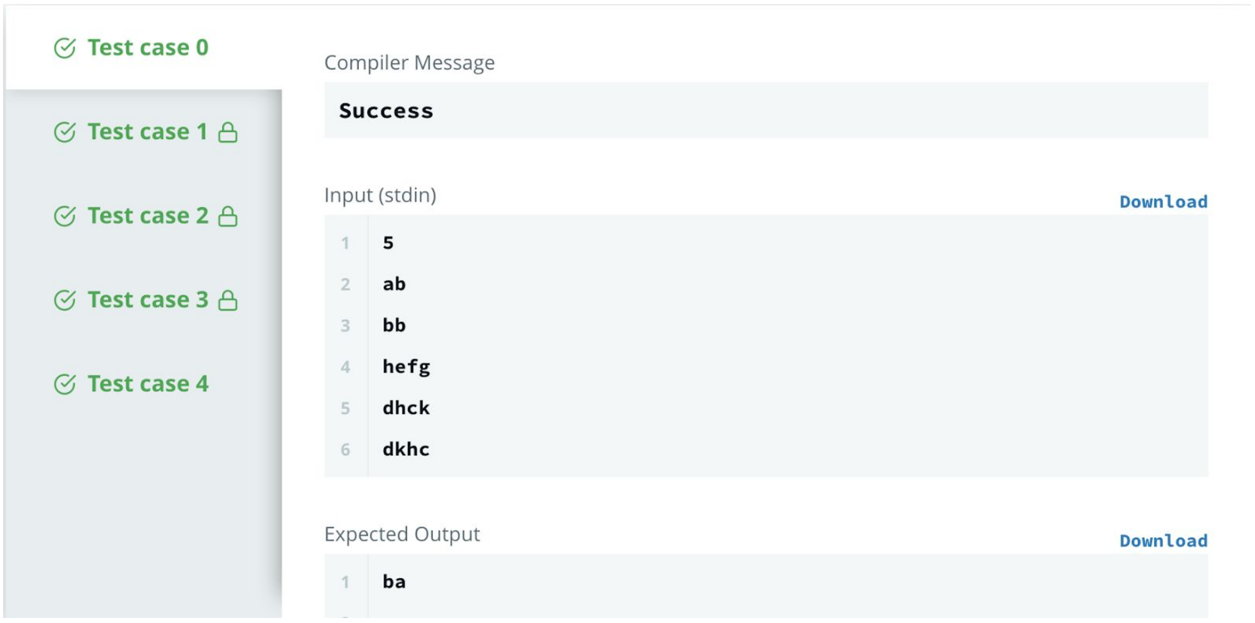
Time Complexity - $\mathcal{O}(|w|)$
Space Complexity - $\mathcal{O}(|w|)$

where w is the input string.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Bigger%20is%20Greater/bigger_is_greater.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
5
ab
bb
hefg
dhck
dkhc
```

Expected Output

```
ba
```

https://www.hackerrank.com/challenges/bigger-is-greater/submissions/code/128498588

Problem 2

Chef and Bipartite Graphs

https://www.codechef.com/problems/ICPC16F

Problem Description

Chef is very interested in studying biparite graphs. Today he wants to construct a bipartite graph with n vertices each, on the two parts, and with total number of edges equal to m. The vertices on the left are numbered from 1 to n. And the vertices on the right are also numbered from 1 to n. He also wants the degree of every vertex to be greater than or equal to d, and to be lesser than or equal to D. ie. for all v, d ≤ deg(v) ≤ D
Given four integers, n, m, d, D, you have to help Chef in constructing some bipartite graph satisfying this property. If there does not exist any such graph, output -1.

Input

First line of the input contains an integer T denoting the number of test cases. T test cases follow.
The only line of each test case contains four space-separated integers n, m, d, D.

Output

For each test case, output -1 if there is no bipartite graph satisfying this property. Otherwise, output m lines, each of the lines should contain two integers u, v denoting that there is an edge between vertex u on the left part and vertex v on the right part. If there can be multiple possible answers, you can print any. Note that the bipartite graph should not have multi-edges.

Constraints

- *1 ≤ T, n ≤ 100*
- *1 ≤ d ≤ D ≤ n*
- *0 ≤ m ≤ n * n*

Approach Description

Given the minimum and maximum degree of all the nodes in the bipartite graph,
Chef asks us to construct, if possible, a bipartite graph that has m edges.
It is easy to see that,
If the total number of edges, m, is
less than n * d (i.e. minimum # edges that could be made) or
more than n * D (i.e. maximum # edges that can be made),
No such graph exists.

This forms the base case of our approach.
If this condition is fulfilled, there definitely exists a bipartite graph that can be formed.
Since any graph is accepted, we first start by equally ‘distributing’ the edges among all the vertices. The resulting degree of every vertex after this step is floor(m / n), where m is the total number edges needed and n is the number of vertices/nodes on either side of the bipartite graph. Let’s call this min_deg. We make an edge between any vertex min_deg apart (in node number).
Now we only need to account for the remaining m % n edges. To do this, we simply make edges between two vertices min_deg apart.
During this process, it is always guaranteed that all the nodes have at least d edges, since min_deg >= d and none of the edges would have more than D edges since we ‘distribute’ the edges equally and it is guaranteed that we can accommodate all the edges (due to the check at the beginning).

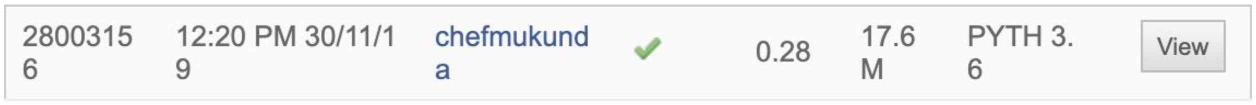
Run Time Analysis

Time Complexity - $\mathcal{O}(m)$
Space Complexity - $\mathcal{O}(m)$
Where m is the total number of edges required to be made.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Bipartite%20Graphs/bipartite_graphs.py

Test Cases



28003156	12:20 PM 30/11/19	chefmukunda	✓	0.28	17.6M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28003156

Problem 3

Forming a Magic Square

https://www.hackerrank.com/challenges/magic-square-forming/problem

Problem Description

We define a magic square to be a n x n matrix of distinct positive integers from 1 to n^2 *where the sum of any row, column, or diagonal of length n is always equal to the same number: the magic constant.*
You will be given a 3 x 3 matrix s of integers in the inclusive range [1, 9]. We can convert any digit a to any other digit b in the range [1, 9] at cost of |a - b|. Given s, convert it into a magic square at minimal cost. Print this cost on a new line.
Note: The resulting magic square must contain distinct integers in the inclusive range [1, 9].
For example, we start with the following matrix s:
5 3 4
1 5 8
6 4 2

We can convert it to the following magic square:
8 3 4
1 5 9
6 7 2

This took three replacements at a cost of |5 - 8| + |8 - 9| + |4 - 7| = 7.

Function Description

Complete the formingMagicSquare function in the editor below. It should return an integer that represents the minimal total cost of converting the input square to a magic square.
formingMagicSquare has the following parameter(s):

- *s: a 3 x 3 array of integers*

Input Format

Each of the lines contains three space-separated integers of row s[i].

Constraints

- $s[i][j] \in [1, 9]$

Output Format

Print an integer denoting the minimum cost of turning matrix s into a magic square.

Approach Description

The problem defines a magic square to be an n x n matrix where, the sum of both the diagonals, all the rows, and all the columns is equal to the magic constant.
https://en.wikipedia.org/wiki/Magic_square
Given this information, it is easy to verify if a matrix is a magic square or not, which takes $\mathcal{O}(n^2)$ time.
Now the problem states that, given a matrix of size 3 x 3, we need to find the minimum cost to make it a magic square.
A simple approach would be to find all magic squares of the same size and compare the difference with each of them and return the minimum cost required to convert the given matrix to a magic square. To do this, we list all the permutations of numbers from 1 up to 9 which would lead to n! of them, then making it a matrix by starting a new row after every 3 elements as shown below.
[1, 2, 3, 4, 5, 6, 7, 8, 9] =>
[[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]]

Now that we have all possible magic square of the required size, we simply calculate the cost to convert the given input matrix to all the known magic squares and return the minimum cost as required.
We can also leverage the fact that the input size and thus the known magic squares are fixed ahead of time for the sake of this problem. This allows us to simply store all the 8 possible magic squares of order 3 and simply calculate each of the 8 costs to make the input a magic square if it is not.

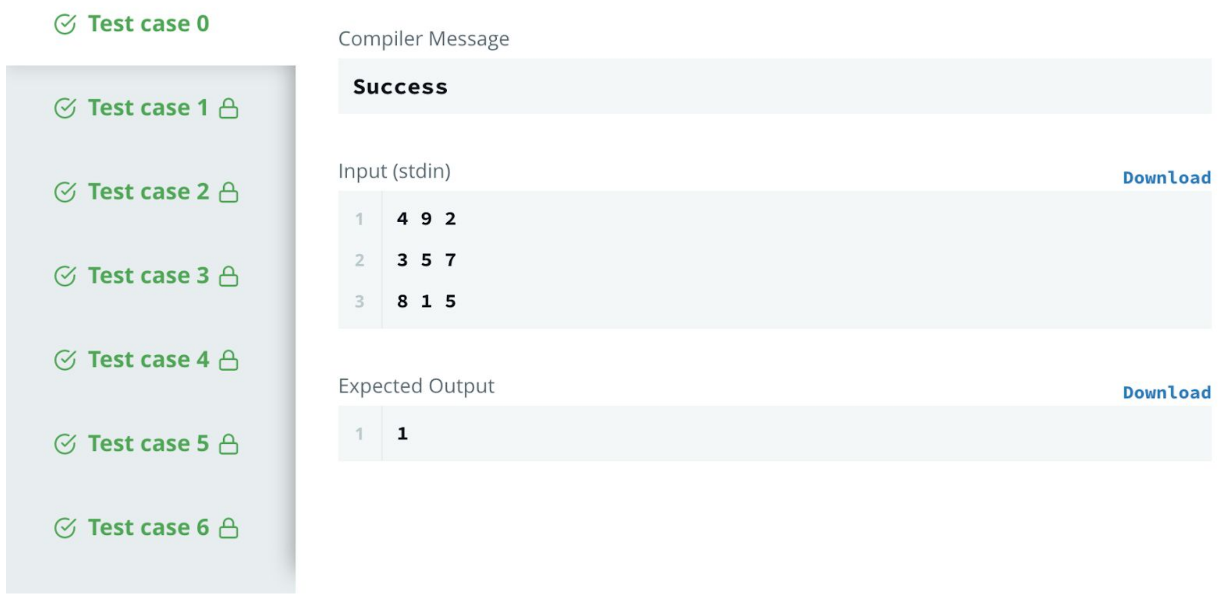
Run Time Analysis

Time Complexity - $\mathcal{O}(1)$ since n is constant, else $\mathcal{O}(n^2)$
Space Complexity - $\mathcal{O}(1)$
This is since the size of our input is fixed and making our storage and number of operations constant.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Forming%20Magic%20Square/forming_magic_square.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
4 9 2
3 5 7
8 1 5
```

Expected Output

```
1
```

https://www.hackerrank.com/challenges/magic-square-forming/submissions/code/132339603

Problem 4

Number of Factors
https://www.codechef.com/problems/NUMFACT

Problem Description

Alice has learnt factorization recently. Bob doesn't think she has learnt it properly and hence he has decided to quiz her. Bob gives Alice a very large number and asks her to find out the number of factors of that number. To make it a little easier for her, he represents the number as a product of N numbers. Alice is frightened of big numbers and hence is asking you for help. Your task is simple. Given N numbers, you need to tell the number of distinct factors of the product of these N numbers.

Input

First line of input contains a single integer T, the number of test cases.
Each test starts with a line containing a single integer N.
The next line consists of N space-separated integers (A_i).

Output

For each test case, output on a separate line the total number of factors of the product of given numbers.

Constraints

$1 \leq T \leq 100$
$1 \leq N \leq 10$
$2 \leq A_i \leq 1000000$

Approach Description

The question gives us N integers and asks us to find the number of distinct factors of the product of all the numbers.
As simple brute force approach would find the product of all these numbers and simply count all the numbers that exactly divide it from 1 up-to-the product.
An optimization to the above approach would be to count only up-to (product / 2), this would yield us all the distinct factors excluding the number itself.
A further optimization would be to count only up-to the square root of the product and count every pair [i, product / i] where i divides the product exactly.
However, all these approaches are still very slow since they involve calculating the product of all the given N integers.

An important observation from elementary mathematics is that the number of distinct factors of a number can b simply be calculated from its prime factorization.
Now we simply find all the prime factors of each of the N integers, this allows us to find the complete/overall prime factorization of the total product of the N integers themselves.
To find the prime factors of each number, we use the sieve technique, we repeatedly divide the number 2 till it is no longer even keeping count of the factors, then we try dividing it every odd number up-to-the square root of the odd number obtained dividing it by 2 repeatedly.
This gives us the prime factorization of the number. This method is similar to the sieve of eratosthenes taking O(Alog(A)) time. Next, since we need to do this for all the numbers, it makes out complexity $\mathcal{O}(Alog(log(A)) + Nlog(A))$.

Finally repeating this for all the integers, the combined prime factorization for all of them is equivalent to the prime factorization of the product of these integers by definition.
Now that we have the prime factorization of the overall product, we find the count of all its distinct factors as the product of exponents of all the prime factors increased by 1. This results from the factor tree method in elementary (https://en.wikipedia.org/wiki/Integer_factorization) mathematics.

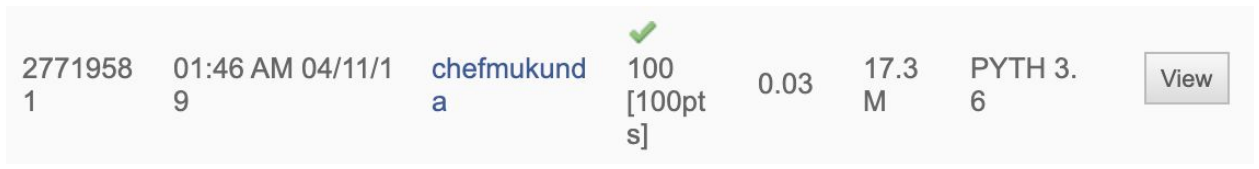
Run Time Analysis

Time Complexity - $\mathcal{O}(Alog(log(A)) + Nlog(A))$
Space Complexity - $\mathcal{O}(A)$
Where A, is the maximum number of the N integers, we need to find prime factors of the numbers.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Number%20of%20Factors/number_of_factors.py

Test Cases



27719581	01:46 AM 04/11/19	chefmukunda	✓ 100 [100pts]	0.03	17.3M	PYTH 3.6	View

https://www.codechef.com/viewsolution/27719581

Problem 5

Trouble Sort

https://codingcompetitions.withgoogle.com/codejam/round/00000000000000cb/00000000000079cb

Problem Description

Deep in Code Jam's secret algorithm labs, we devote countless hours to wrestling with one of the most complex problems of our time: efficiently sorting a list of integers into non-decreasing order. We have taken a careful look at the classic bubble sort algorithm, and we are pleased to announce a new variant.

The basic operation of the standard bubble sort algorithm is to examine a pair of adjacent numbers and reverse that pair if the left number is larger than the right number. But our algorithm examines a group of three adjacent numbers, and if the leftmost number is larger than the rightmost number, it reverses that entire group. Because our algorithm is a "triplet bubble sort", we have named it Trouble Sort for short.

```
TroubleSort(L): // L is a 0-indexed list of integers
  let done := false
  while not done:
    done = true
    for i := 0; i < len(L)-2; i++:
      if L[i] > L[i+2]:
        done = false
        reverse the sublist from L[i] to L[i+2], inclusive
```

For example, for L = 5 6 6 4 3, Trouble Sort would proceed as follows:

- *First pass:*
    - *inspect 5 6 6, do nothing: 5 6 6 4 3*
    - *inspect 6 6 4, see that 6 > 4, reverse the triplet: 5 4 6 6 3*
    - *inspect 6 6 3, see that 6 > 3, reverse the triplet: 5 4 3 6 6*
- *Second pass:*
    - *inspect 5 4 3, see that 5 > 3, reverse the triplet: 3 4 5 6 6*
    - *inspect 4 5 6, do nothing: 3 4 5 6 6*
    - *inspect 5 6 6, do nothing: 3 4 5 6 6*
- *Then the third pass inspects the three triplets and does nothing, so the algorithm terminates.*

We were looking forward to presenting Trouble Sort at the Special Interest Group in Sorting conference in Hawaii, but one of our interns has just pointed out a problem: it is possible that Trouble Sort does not correctly sort the list! Consider the list 8 9 7, for example.

We need your help with some further research. Given a list of N integers, determine whether Trouble Sort will successfully sort the list into non-decreasing order. If it will not, find the index (counting starting from 0) of the first sorting error after the algorithm has finished: that is, the first value that is larger than the value that comes directly after it when the algorithm is done.

Input

The first line of the input gives the number of test cases, T. T test cases follow. Each test case consists of two lines: one line with an integer N, the number of values in the list, and then another line with N integers V_i, the list of values.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is OK if Trouble Sort correctly sorts the list, or the index (counting starting from 0) of the first sorting error, as described above.

Limits

$1 \leq T \leq 100$.
$0 \leq V_i \leq 10^9$, for all i.
Memory limit: 1GB.

Approach Description

The problem proposes and presents a new way/algorithm to sort numbers similar to bubble sort. The problem proposes that to sort the array, each time we look at three continuous numbers in the array and compare the left-most and the right-most element if the left-most element is larger it reverses the entire group of three numbers. The problem asks us to examine the given algorithm and answer if it would correctly sort a given list or not. A trivial solution would be to implement the Trouble sort Algorithm and simply check if it correctly sorts the given list. However, the proposed algorithm runs in O(n^2) time, although a correct solution, this times out on the test cases indicating we can do better. We analyze the trouble sort algorithm to expose a fundamental flaw in it. The algorithm only compares even indexed elements with other even indexed elements and the same for odd indexed elements. This implies that after the algorithm is done running, all the elements at even and odd indices are sorted among themselves (i.e. all the values at even indices and all the values at odd indices are sorted among themselves). So to recreate the output of this algorithm, we can simply run a known, faster sorting algorithm on the even elements and the odd elements separately, then we interleave these sorted groups with each other, picking an even indexed element followed by an odd one. Finally, we can simply check for the first out of place value in this resultant array to check if it is sorted or not. This brings down our complexity to O(nlog(n)). Sorting even and odd indexed elements takes O(nlog(n)) + O(nlog(n)) = O(nlog(n)) time. Checking if the list is sorted takes O(n) time. Overall, O(nlog(n)) + O(n) = O(nlog(n)). The space complexity depends on the sorting algorithm used.

The python sort function we use sorts the list in place without any additional space, making our space complexity O(1).

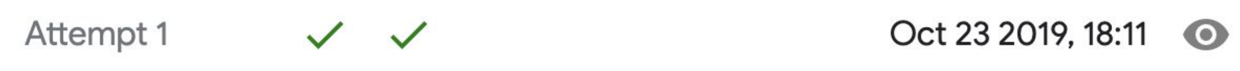
Run Time Analysis

Time Complexity - $\mathcal{O}(nlog(n))$
Space Complexity - $\mathcal{O}(1)$
Where n is the size of the input array.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Trouble%20Sort/trouble_sort.py

Test Cases


Attempt 1 ✓ ✓ Oct 23 2019, 18:11

Problem 6

String Similarity

https://www.hackerrank.com/challenges/string-similarity/problem

Problem Description

For two strings A and B, we define the similarity of the strings to be the length of the longest prefix common to both strings. For example, the similarity of strings "abc" and "abd" is 2, while the similarity of strings "aaa" and "aaab" is 3.
Calculate the sum of similarities of a string S with each of its suffixes.

Input Format

The first line contains the number of test cases t.
Each of the next t lines contains a string to process, s.

Constraints

- $1 \leq t \leq 10$
- $1 \leq |s| \leq 100000$
- *s is composed of characters in the range ascii[a-z]*

Output Format

Output t lines, each containing the answer for the corresponding test case.

Approach Description

The question defines similarity as the length of the longest common prefix of two strings.
It asks us to find the sum of the similarities of the given input string with all of its suffixes.

We attempt a brute force solution to the problem by listing out all the suffixes of the string and then calculate it's similarity with the original input string itself. However, this approach takes (n^2) time which is still to slow.
Finally, we optimize this approach by using the popular Z-function to find the similarity of each suffix storing it as we go. The following blog was helpful in understanding the Z-function. We are able to calculate the z array, where z[i] is the length of the longest common prefix between s and the suffix of s starting at i.
We can simply iterate through every position in the string, i, and update z[i] for each one.
This still leads to an (n^2) algorithm.

Turn out we can optimize this by simple case analysis and by making use of the previously computed values. We define a segment as those substrings that coincide with a prefix of s. To achieve this, we keep the l, r indices of the rightmost segment match, where one could imagine r as a boundary till which we have scanned our input string s.
Iterating through the array, we find that,
If i > r, we compute z[i] using our trivial algorithm.
Otherwise, we use the previous values of z already to initialize z[i].
We observe that the substrings s[l:r] match with s[0:r - l]. So we take z[i - l] as the initial approximation for z[i] in this case.
Notice how in each case we move one step forward in processing the string. So we never loop more than $\mathcal{O}(n)$ times. Leading to a linear time solution for our problem.

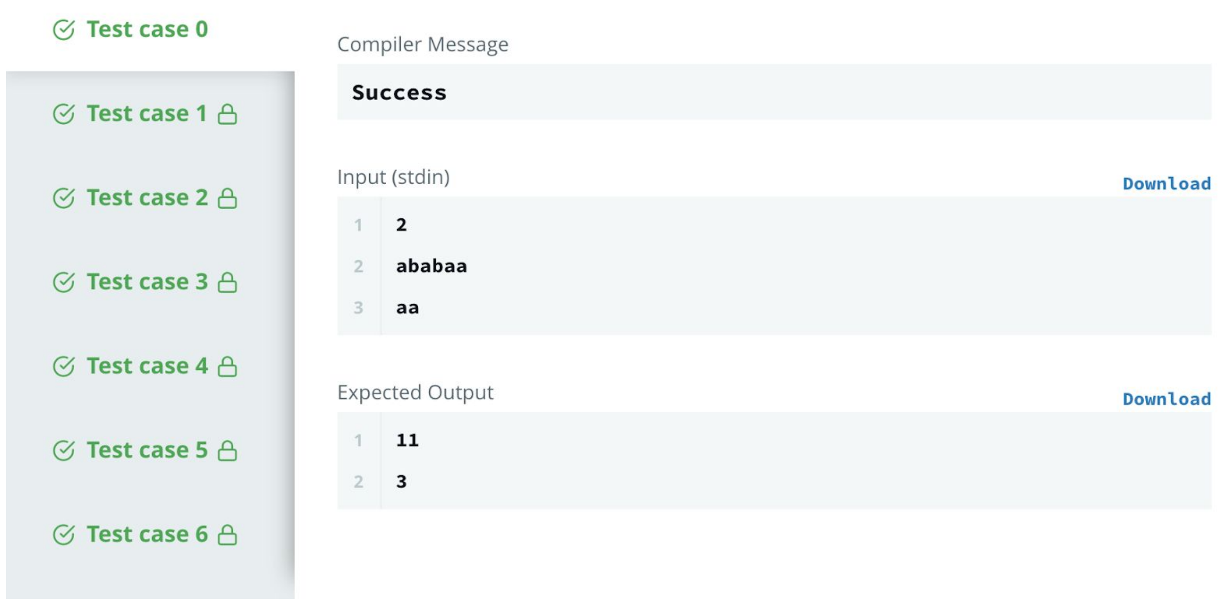
Run Time Analysis

Time Complexity - $\mathcal{O}(n)$
Space Complexity - $\mathcal{O}(n)$

Github

https://github.com/MukundaJ/APS/blob/master/Problems/String%20Similarity/string_similarity.py

Test Cases


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin)

```
2
ababaa
aa
```

Expected Output

```
11
3
```

https://www.hackerrank.com/challenges/string-similarity/submissions/code/131870415

Problem 7

Delivery Man

https://www.codechef.com/LTIME19/problems/TADELIVE

Problem Description

Andy and Bob are the only two delivery men of Pizza-chef store. Today, the store received N orders. It's known that the amount of tips may be different when handled by different delivery man. More specifically, if Andy takes the i^{th} order, he would be tipped A_i dollars and if Bob takes this order, the tip would be B_i dollars.
They decided that they would distribute the orders among themselves to maximize the total tip money. One order will be handled by only one person. Also, due to time constraints, Andy cannot take more than X orders and Bob cannot take more than Y orders. It is guaranteed that X + Y is greater than or equal to N, which means that all the orders can be handled by either Andy or Bob.
Please find out the maximum possible amount of total tip money after processing all the orders.

Input

- *The first line contains three integers N, X, Y.*
- *The second line contains N integers. The i^{th} integer represents A_i.*
- *The third line contains N integers. The i^{th} integer represents B_i.*

Output

- *Print a single integer representing the maximum tip money they would receive.*

Constraints

All test:
- $1 \le N \le 10^5$
- $1 \le X, Y \le N; X + Y \ge N$
- $1 \le A_i, B_i \le 10^4$

10 points:
- $1 \le N \le 20$

30 points:
- $1 \le N \le 5000$

60 points:
- $1 \le N \le 10^5$

Approach Description

A simple brute force approach would be to find the tips for all possible arrangements of bob and andy going at different times and then find the maximum tip of this list.
However, there would be an exponential number of arrangements there could be possible.
Clearly, this would be a very slow approach.

In the problem, we need to be able to maximize the tips that andy and bob together can collect.
So we must consider who to send to make a higher tip at any particular time.
We construct an array what holds tuples of the form ((min(a, b), max(a, b), (a - b))).
We sort this array based on the absolute value of the difference i.e. |a - b|.
This takes us $\mathcal{O}(nlog(n))$ time.

Next using a simple case analysis we iterate through this list.

```
If the difference is zero, we can send either andy or bob for the order.
if Andy had a higher tip,
    If Andy can take this order, add his tip.
    Else let Bob take the order and add his tip.
Else Bob had a higher tip
    If Bob can take this order, add his tip.
    Else let Andy take the order and add his tip.
```

The above takes $\mathcal{O}(n)$ time.
This is a fairly intuitive way to approach the problem and take over all $\mathcal{O}(nlog(n))$ time.

We may still slightly optimize the above approach by first,
We simply let Andy or Bob take the tip at a time based simply on who has a higher tip at that time.
Next, we need to account for either andy or bob is 'overutilized''
If Andy was over-utilized,
find the min diff in tips where he contributed and take Bob's
contribution there instead.
This works because when we subtract the diff, the min tip is taken for
order instead.
Put simply, if either of them is overutilized, we take the others contribution at that time instead.
For this, we only need a sorted array of both Andy's and Bob's tips so we only sort a smaller array. This still takes an overall of $\mathcal{O}(nlog(n))$ time, but slightly faster.

Run Time Analysis

Time Complexity - $\mathcal{O}(nlog(n))$
Space Complexity - $\mathcal{O}(min(X+Y, N))$

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Delivery%20Man/delivery_man.py

Test Cases

N/A
Platform link broken / problem closed for submission.

Problem 8

Array Transform

https://www.codechef.com/problems/ARRAYTRM

Problem Description

Given n numbers, you can perform the following operation any number of times: Choose any subset of the numbers (possibly empty), none of which are 0. Decrement the numbers in the subset by 1, and increment the numbers not in the subset by K.
Is it possible to perform operations such that exactly n - 1 numbers become 0?

Input

*The first line contains the number of test cases T. 2*T lines follow, 2 for each case. The first line of a test case contains the numbers n and K. The next line contains n numbers, a_1...a_n.*

Output

Output T lines, one corresponding to each test case. For a test case, output "YES" if there is a sequence of operations as described, and "NO" otherwise.

Sample Input :

```
3
2 1
10 10
3 2
1 2 2
3 2
1 2 3
```

Sample Output :

```
YES
YES
NO
```

Constraints

1 <= T <= 1000
2 <= n <= 100
1 <= K <= 10
0 <= a_i <= 1000

Approach Description

The question asks us if we can make exactly n - 1 elements zero using only the operation as described in the question as selecting a subset of the array (whose elements are not 0) then we decrement the numbers in the subset by 1 and increment the remaining numbers by k.

If there is just 1 or 2 (i.e. n == 1 or 2) element, then 0 or 1 (i.e. n - 1 == 0 or 1) elements should be 0, so the all unit length arrays satisfy the condition, and for 2 length arrays, we can keep selecting one number to decrement to 0 trivially.
We notice that, decreasing one number say x, in the subset and increasing the other not in the subset say y, after m operations we would have,
x - 1 * m = y + k * m => m = (x - y) / (k + 1)

Otherwise, we find the remainder on modulating each of the numbers by (k + 1), essentially mapping them to a number space of [0, k]. We then count how many elements have the same remainder (i.e. get mapped to the same number when transformed to a [0, k] space). If n - 1 or more numbers have the same remainder/mapping, then it is possible to transform the array.

The intuition behind the approach is when we to transform all the numbers to a restricted number space based on k as described above it is the same as increment any number subset with k and decrement the numbers in our chosen subset, similar to smaller arrays case.
Now If the largest value of these mappings is n - 1 or greater, it means that at least those n - 1 numbers can be made 0, by simply decreasing them by 1.
Otherwise, there is no possible way to make exactly n - 1 of the elements 0 using only the given operations.

We can see that an O(1) space solution is now easy to do, by simply

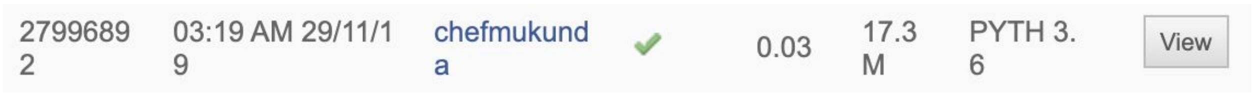
Run Time Analysis

Time Complexity - $\mathcal{O}(n)$
Space Complexity - $\mathcal{O}(n)$/ O(1)
Since we, go through all numbers and also construct a frequency map that takes O(n) time and also finds the maximum value in the frequency map.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Array%20Transform/array_transfrom.py

Test Cases



27996892	03:19 AM 29/11/19	chefmukunda	✓	0.03	17.3M	PYTH 3.6	View

https://www.codechef.com/viewsolution/27996892

Problem 9

Bead Ornaments
https://www.hackerrank.com/challenges/beadornaments/problem

Problem Description

There are N colors of beads. You have b_i beads of the i^{th} color. You want to make an ornament by joining all the beads together. You create the ornament by using the following algorithm:

- *Step #1 Arrange all the beads in any order such that beads of the same color are placed together.*
- *Step #2 The ornament initially consists of only the first bead from the arrangement.*
- *Step #3 For each subsequent bead in order, join it to a bead of the same color in the ornament. If there is no bead of the same color, it can be joined to any bead in the ornament.*

All beads are distinct, even if they have the same color. How many different ornaments can be formed by following the above algorithm? Two ornaments are considered different if two beads are joined by a thread in one configuration, but not in the other.

Update/clarification

Think of the bead formation as a tree and not as a straight line. Any number of beads can be connected to a bead.

Input Format

The first line contains the number of test cases T. T test cases follow. Each test case contains N on the first line - the number of colors of beads. The next line contains N integers, where the i^{th} integer b_i denotes the number of beads of the i^{th} color.

Constraints

- $1 \leq T \leq 20$
- $1 \leq N \leq 10$
- $1 \leq b_i \leq 30$

Output Format

Output T lines, one for each test case. All answers should be output modulo 1000000007.

Approach Description

The update asks us to think of the ornaments a tree.
We are also told that each colored bead, even those of the same color are considered to be distinct.
We use caleys formula to calculate the number of trees we can make with n labeled nodes.
Now we know that there are n^{n-2} such arrangements.
https://en.wikipedia.org/wiki/Cayley%27s_formula

Next we need to find the number of ways these trees can be connected among themselves.
Connecting only single beads of each color, there would be only caley(N) number of ornaments.
But we need to find the number of ways to be able to arrange the different colors as well.

Say there were only two colors 0 and 1.
Then the answer would be = arrange(0) * beads(0) * arrange(1) * beads(1)
The number of arrangements is given to us by caleys formula.
Which makes the above formula
beads(0)^(beads(0) - 1) * beads(0)^(beads(0) - 1)

This leads us to the formula to arrange n beads in the same way as,
(beads(0)^(beads(0) - 1))*...* …* (beads(n)^(beads(n) - 1)) * (beads(0) + … +beads(n))^(n - 2)

Now we can simply calculate the number of bead ornaments that can be made using the above formula.
We apply this to the given input with the number of beads of each color and return the result.
The following blog was extremely helpful in solving this question.

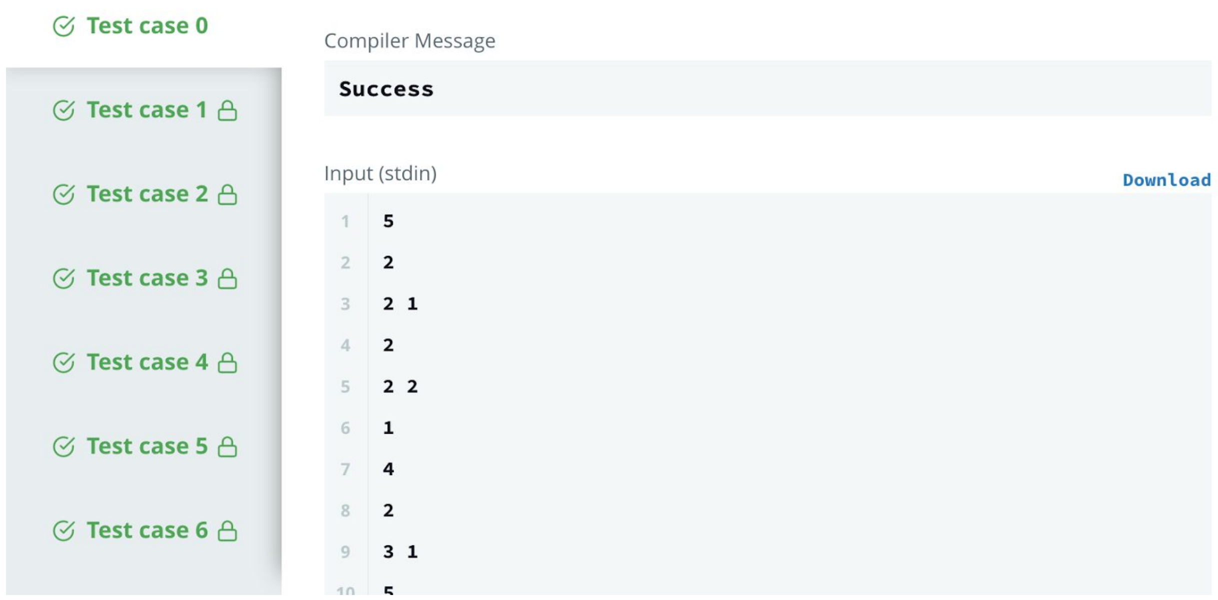
Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(1)
Where n is the number of different colors.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Bead%20Ornaments/bead_ornaments.py

Test Cases



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

```
5
2
2 1
2
2 2
1
4
2
3 1
5
```

https://www.hackerrank.com/challenges/beadornaments/submissions/code/132525906

Problem 10

Alien Rhyme

https://codingcompetitions.withgoogle.com/codejam/round/0000000000051635/0000000000104e05

Problem Description

During some extraterrestrial exploration, you found evidence of alien poetry! Your team of linguists has determined that each word in the alien language has an accent on exactly one position (letter) in the word; the part of the word starting from the accented letter is called the accent-suffix. Two words are said to rhyme if both of their accent-suffixes are equal. For example, the words PROL and TARPOL rhyme if the accented letter in both is the O or the L, but they do not rhyme if the accented letters are the Rs, or the R in PROL and the P in TARPOL, or the O in PROL and the L in TARPOL.

You have recovered a list of N words that may be part of an alien poem. Unfortunately, you do not know which is the accented letter for each word. You believe that you can discard zero or more of these words, assign accented letters to the remaining words, and then arrange those words into pairs such that each word rhymes only with the other word in its pair, and with none of the words in other pairs.

You want to know the largest number of words that can be arranged into pairs in this way.

Input

The first line of the input gives the number of test cases, T. T test cases follow. Each test case starts with a line with a single integer N. Then, N lines follow, each of which contains a string W_i of uppercase English letters, representing a distinct word. Notice that the same word can have different accentuations in different test cases.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the size of the largest subset of words meeting the criteria described above.

Limits

$1 \le T \le 100$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \le$ length of $W_i \le 50$, for all i.
W_i consists of uppercase English letters, for all i.
$W_i \ne W_j$, for all $i \ne j$. (Words are not repeated within a test case.)

Approach Description

The problem asks us to find pairs of words whose suffixes match.
We want to find out the largest/maximum number of words that can be made into such pairs.

A naive way would be to just brute force all the and try all the set of all possible ways to arrange the words into pairs.
For every possible pair, we then try all the suffixes that can be matched.
We take care that none of the matched suffixes in any pair are the same.
Finally, we find the maximum set (one with the greatest size) and this would be our result.
While this brute force solution does work for the first test case, it times out on the second one.

We need to have a more efficient way of calculating these groupings of the words into pairs.
An important observation is that any we must match pairs that have the longest accent size in common. This would allow us to match more pairs if possible which may have shorter matching suffixes.

To do this,
First
We use 2 dictionaries to hold the count of each suffix and one to map the suffix to a list of words it is common to.
We build these dictionaries for all the suffixes of all the words which takes us $\mathcal{O}(n^2)$ time
Next, to us only the suffixes that have two or more words in common are useful to us,
We discard all the other suffixes with a frequency of less than 2

Next, we sort all the useful suffixes based on the length of each suffix. This takes us O(nlog(n)) time.
Finally, we iterate over all the useful suffixes,
For every useful suffix,

```
    (we already have a mapping of all the suffixes to a list of words)
    We find the words that haven't been used yet in the suffixes list of words
    (This is just an O(1) lookup in a set of used words)
    If there are 2 or more unused/available words, we match the first 2
    (any 2 can be matched here)
        We update the number of words matched by 2
        And make the matched words now unavailable.
```

There are more space-efficient ways to implement the same using Tries, where we may simply add all the words in a reversed order and essentially follow the same process of matching words with the largest suffixes first. However, our approach solves both the test cases successfully.

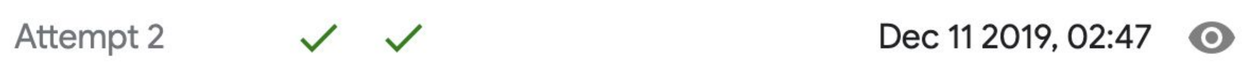
Run Time Analysis

Time Complexity - $\mathcal{O}(n^2)$
Space Complexity - $\mathcal{O}(n^2)$
Where n is the total number of input words.
Since we find and store all the suffixes of all the words.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Alien%20Rhyme/alien_rhyme.py

Test Cases


Attempt 2 ✓ ✓ Dec 11 2019, 02:47

Problem 11

Marbles

https://www.codechef.com/problems/MARBLES

Problem Description

Rohit dreams he is in a shop with an infinite amount of marbles. He is allowed to select n marbles. There are marbles of k different colors. From each color, there are also infinitely many marbles. Rohit wants to have at least one marble of each color, but still, there are a lot of possibilities for his selection. In his effort to make a decision he wakes up. Now he asks you how many possibilities for his selection he would have had. Assume that marbles of equal color can't be distinguished, and the order of the marbles is irrelevant.

Input

The first line of input contains a number T <= 100 that indicates the number of test cases to follow. Each test case consists of one line containing n and k, where n is the number of marbles Rohit selects and k is the number of different colors of the marbles. You can assume that 1<=k<=n<=1000000.

Output

For each test case print the number of possibilities that Rohit would have had. You can assume that this number fits into a signed 64-bit integer.

Approach Description

This is a n Choose k problem, where you need to select at least one marble of every color.
This would be equivalent to the sum of all possible values of n - k Choose i, where i is in the range [1..n-k] for each marble.
Mathematically this works out to (n - 1) Choose (k - 1).

if k > n or k < 0 or n < 0, there are 0 ways to make the choices,
Otherwise, we just calculate the value of (n - 1) C (k - 1).

A small optimization to the problem is where we observe that n Choose k and n Choose (n - k)
Are the same, we calculate only up to minimum of (k, n - k) when calculating n Choose k.
This takes us $\mathcal{O}(min(n-k,k))$

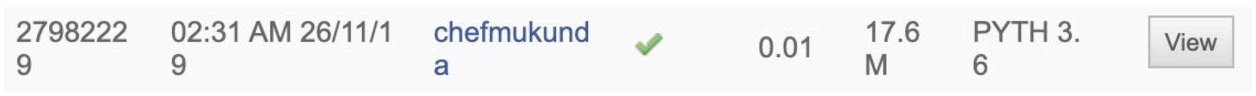
Run Time Analysis

Time Complexity - $\mathcal{O}(min(n-k,k))$
Space Complexity - $\mathcal{O}(1)$

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Marbles/marbles.py

Test Cases



27982229	02:31 AM 26/11/19	chefmukunda	✓	0.01	17.6M	PYTH 3.6	View

https://www.codechef.com/viewsolution/27982229

Problem 12

Best Time to Buy and Sell Stock IV
https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/

Problem Description

Say you have an array for which the i-th element is the price of a given stock on day i.
Design an algorithm to find the maximum profit. You may complete at most k transactions.

Note: You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Approach Description

A brute force approach for this problem is to list out all possible sets k profitable transactions and find the set which has the highest possible profit.
However, this would be exponential and too slow.

Consider the case where k is 1, we would make the transaction with minimum buy price and the maximum sale price.
For k = 2, we try to minimize our buying cost,
When making the second transaction, we only need to pay its cost - our current profit.
We try and extend this approach to k transaction with the following pseudo-code explaining our approach in the #comments.

```
#Array to hold
#1. profit when making at most i transactions.
#2. minimum cost to be able to buy a stock at time i, initally all zero.
profits, costs = [0] * (k + 1), [float('inf')] * (k + 1)
# For all the prices at time i,
for p in prices:
  # For all the k transactions,
  for i in range(1, k + 1):
     # Minimize the cost,
     # currentPrice - previous profit represents how much we actually pay
     # out of pocket (since some of the cost is covered by the profit
     # we made earlier).
     costs[i] = min(costs[i], p - profits[i - 1])
     # Maximize the profit,
     # currentPrice - cost[i] = max profit that we can making at most i
     # transactions.
     profits[i] = max(profits[i], p - costs[i])
# Return maximum profit that can be made by at most k transactions.
return profits[k]
```

This is a simple dynamic programming approach where, at each step, we essentially find the minimum cost of the i transactions and the maximum profit by making i transactions.
This takes us $\mathcal{O}(Nk)$ time where N is the number of prices and k are then number of transactions we are allowed.

This still, approach, however, times out some of the test cases.
If the number of transactions is greater than n // 2, we can make all the possible transactions.

Run Time Analysis

Time Complexity - $\mathcal{O}(Nk)$
Space Complexity - $\mathcal{O}(k)$ to hold the dp array.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20IV/buy_n_sell.py

Test Cases

211 / 211 test cases passed.
Runtime: **100 ms**
Memory Usage: **13.3 MB**

Status: Accepted
Submitted: **2 weeks, 2 days ago**

https://leetcode.com/submissions/detail/281492807/

Problem 13

Super Six Substrings

https://www.hackerrank.com/contests/hourrank-18/challenges/super-six-substrings/problem

Problem Description

David loves numeric strings. He considers a substring of a numeric sequence to be super if both of the following conditions are satisfied:

- *The substring's integer representation is divisible by 6.*
- *It does not contain leading zeroes (e.g., 6 is super, but 06 is not).*

In addition, he considers the one-character substring 0 to be super.
For example, s = “606” has five super substrings: 6, 0, 6, 60, and 606.
Given s, find and print the total number of super substrings in s.
Note: The length of the numbers represented by s and its substrings are likely to be outside of the bounds of what numeric data types can represent.
Input Format
A single numeric string denoting s.
Constraints

- $1 \leq |s| \leq 10^5$

Output Format
Print an integer denoting the number of super substrings.

Approach Description

An easy approach to this problem is to just find all the substrings of the given input string. Then we check to see if that string is divisible by 6 or doesn’t have leading 0’s if it doesn’t we increment the count of our found super six substrings by 1.
This approach, however, takes $\mathcal{O}(n^2)$ time.

We need a better approach.
We realize that any even number that is divisible by 3 will be divisible by 6.
We define a function T(i, m), which is the number of substrings that start at i
(i.e. belong to s[i:], where s is our input string), and m is the sum of their digits modulated by 3 (% 3), we always ensure that the number it represents is an even one.
We can see that the number of super substrings would be the sum of all the values of T(i, 0) Where i belongs to [0:len(s)].

Using T(i, m) we find to find all the even substrings that starting at i + 1 and the sum of their digits modulo 3 is (s[i] + m).
We also need to account for the case where (s[i] + m) is itself divisible by 6.
So, we get the following recurrence relation
T(i, m) = T(i + 1, s[i] + m) + ((s[i] + m) % 6 == 0)
This yields us an O(n) solution and is easy to implement recursively.

However, we can optimize it to be more space-efficient by accounting for the fact that anything % 3 can have only 3 values (i.e. 0, 1, 2).

First, we Iterate through s, and memorize every values of sum of all the prefixes % 3 and store it in an array say a (size 3). This array essentially stores, the number of prefixes that have 0 modulo at i = 0, 1 modulo at i = 1 and 2 at i = 2 (doing only so for even substrings).
The pusedo code is as below

```
for i in range(n):
  digit = int(s[i])
  x = (x + digit) % 3
  if digit % 2 == 0: a[x] += 1
```

Finally, we iterate through s again and
notice that is s[i] == '0' only one string is valid starting at i i.e. s[i] itself.
Otherwise, we find all the prefixes beginning at i, having the same
sum modulo % 3 value, for which we can use our precomputed array a.
We demonstrates the psuedo code below,

```
for i in range(n):
  if s[i] == '0':
     count += 1
  else:
     count += a[x]
# Accounting for the prefix already counted by now.
  digit = int(s[i])
  x = (x + digit) % 3
  if digit % 2 == 0: a[x] -= 1
```

This allows us to use constant space and takes O(2n) = O(n) time.

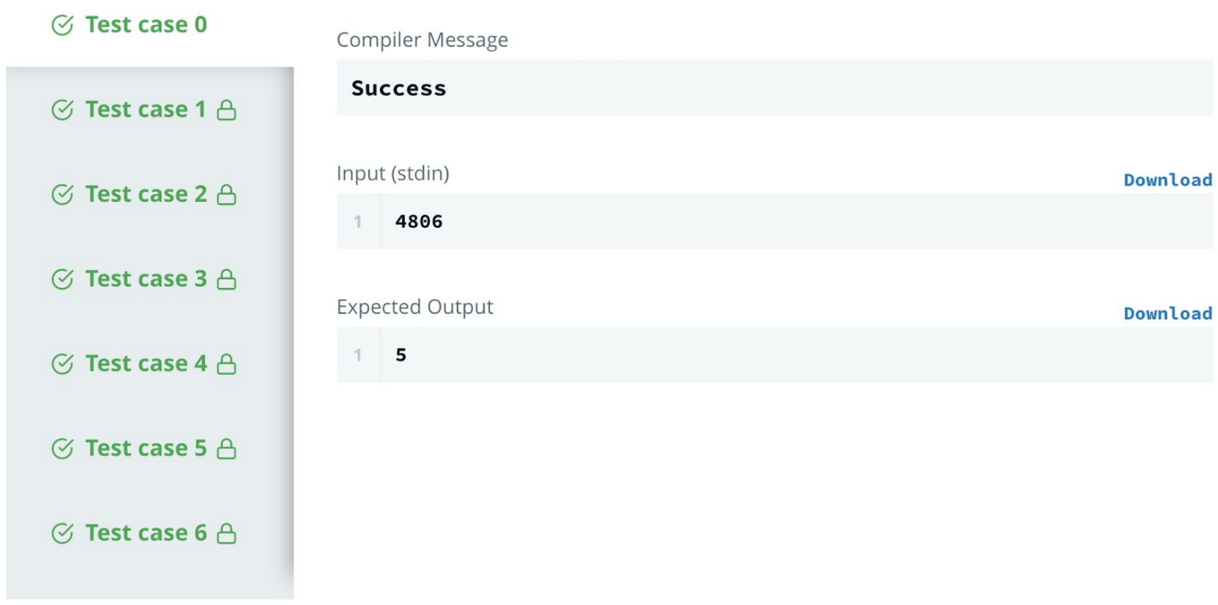
Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(1)

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Super%20Six%20Substrings/super_six_substrings.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
4806
```

Expected Output

```
5
```

https://www.hackerrank.com/contests/hourrank-18/challenges/super-six-substrings/submissions/code/1317937042

Problem 14

Robot Programming Strategy

https://codingcompetitions.withgoogle.com/codejam/round/00000000000516b9/0000000000134c90

Problem Description

After many sleepless nights, you have finally finished teaching a robotic arm to make the hand gestures required for the Rock-Paper-Scissors game. Now you just need to program it to compete in the upcoming robot tournament!
In this tournament, each robot uses a program that is a series of moves, each of which must be one of the following: R (for "Rock"), P (for "Paper"), or S (for "Scissors"). Paper beats Rock and loses to Scissors; Rock beats Scissors and loses to Paper; Scissors beats Paper and loses to Rock.
When two robots face off in a match, the first robot to play a winning move wins. To start, each robot plays the first move of its program. If the two moves are different, one of the moves beats the other and thus one of the robots wins the match. If the moves are the same, each robot plays the next move in its program, and so on.
Whenever a robot has reached the end of its program and needs its next move, it returns to the start of its program. So, for example, the fifth move of a robot with the program RSSP would be R. If a match goes on for over a googol (10^{100}) of moves, the judges flip a fair coin to determine the winner.
Once a match is over, the winning robot resets, so it has no memory of that match. In its next match, it starts by playing the first move of its program, and so on.
The tournament is played in K rounds and has a single-elimination "bracket" structure. There are $N = 2^K$ robots in total, numbered 0 through N - 1. In the first round, robot 0 plays a match against robot 1, robot 2 plays a match against robot 3, and so on, up to robots N - 2 and N - 1. The losers of those matches are eliminated from the tournament. In the second round, the winner of the 0-1 match faces off against the winner of the 2-3 match, and so on. Once we get to the K-th round, there is only one match, and it determines the overall winner of the tournament.
All of the other contestants are so confident that they have already publicly posted their robots' programs online. However, the robots have not yet been assigned numbers, so nobody knows in advance who their opponents will be. Knowing all of the other programs, is it possible for you to write a program that is guaranteed to win the tournament, no matter how the robot numbers are assigned?

Input

The first line of the input gives the number of test cases, T; T test cases follow. Each test case begins with one line containing an integer A: the number of adversaries (other robots) in the tournament. Then, there are A more lines; the i-th of these contains a string C_i of uppercase letters that represent the program of the i-th opponent's robot.

Output

For each test case, output one line containing Case #x: y. If there is a string of between 1 and 500 characters that is guaranteed to win the tournament, as described above, then y should be the string of uppercase letters representing that program. Otherwise, y should be IMPOSSIBLE, in uppercase letters.

Limits

1 ≤ T ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1GB.
Each character in C_i is uppercase R, P, or S, for all i.
$A = 2^K - 1$ for some integer $K \geq 1$.

Approach Description

The question asks us to make a strategy where, given the moves of all the opponents in advance, we must win at the game of rock paper and scissor (R, P, S).

A simple case analysis leads us to the solution.
At each time step, we look at the set moves of all the opponents,

If all three rock paper and scissors are in the set, there is no move we can make that would defeat everyone so it is impossible to make a winning strategy and we return.
If there is only 1 in the set, we pick the move that defeats everyone.
Else only 2 moves are in the set, we pick the move that ties with one and wins over the others. Put simply, we pick find the missing move in the set and pick the move that it would have otherwise defeated, now it is guaranteed that we either win or tie.

This goes on for all the time steps and we look at all the opponents move at each time step so it takes us $\mathcal{O}(Nk)$ where k is the longest move an opponent makes.

Run Time Analysis

Time Complexity - $\mathcal{O}(Nk)$ where k is the longest move an opponent makes
N is the total number of opponents
Space Complexity - $\mathcal{O}(k)$ where k is the time steps

 where N is the number of opponents k is the longest move any opponent makes (or the maximum number of rounds in the worst case).

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Robot%20Programming%20Strategy/robot_programming_strategy.py

Test Cases

Attempt 2 ✓ ✓ Nov 28 2019, 22:40

Problem 15

The Coin Change Problem
https://www.hackerrank.com/challenges/coin-change/problem

Problem Description

You are working at the cash counter at a fun-fair, and you have different types of coins available to you in infinite quantities. The value of each coin is already given. Can you determine the number of ways of making change for a particular number of units using the given types of coins?
For example, if you have 4 types of coins, and the value of each type is given as 8, 3, 1, 2 respectively, you can make change for 3 units in three ways: {1, 1, 1}, {1, 2} and {3}.

Function Description

Complete the getWays function in the editor below. It must return an integer denoting the number of ways to make change.
getWays has the following parameter(s):

- *n: an integer, the amount to make change for*
- *c: an array of integers representing available denominations*

Input Format

The first line contains two space-separated integers describing the respective values of n and m, where:
n is the number of units
m is the number of coin types
The second line contains m space-separated integers describing the respective values of each coin type: c = [c[0], c[1] … c[m - 1]] (the list of distinct coins available in infinite amounts).

Constraints

- $1 \leq c[i] \leq 50$
- $1 \leq n \leq 250$
- $1 \leq m \leq 50$
- *Each c[i] is guaranteed to be distinct.*

Hints

Solve overlapping subproblems using [Dynamic Programming]() (DP):
You can solve this problem recursively but will not pass all the test cases without optimizing to eliminate the [overlapping subproblems](). Think of a way to store and reference previously

*computed solutions to avoid solving the same subproblem multiple times. * Consider the degenerate cases:*
*- How many ways can you make change for 0 cents? - How many ways can you make change for > 0 cents if you have no coins? * If you're having trouble defining your solutions store, then think about it in terms of the base case (n = 0). - The answer may be larger than a 32-bit integer.*

Output Format

Print a long integer denoting the number of ways we can get a sum of n from the given infinite supply of m types of coins.

Approach Description

An exponential solution the problem would be one where we list all possible amounts smaller than the given amount. However, this is clearly an exponential solution and does now solve the problem in the required time constraints.

The hint in the question is a helpful guide to the solution.
We notice that there is only 1 way to make up an amount of 0, by not selecting anything.
Next, we notice that the number of ways to make up 0 + denomination would be the number of ways we can make to make up 0 + (already known number of ways to make 0 + denomination).

The code in itself is very easy to follow,
We explain each step with comments below.

```
def change(amount: int, coins: [int]) -> int:
        # DP array to hold the number of ways an amount i can be made up, where i is
        # the index of the array.
        # Base case, there is only 1 way to make an amount = 0,
        # i.e. select nothing.
        combinations = [1] + [0] * (amount - 1)

        # For all available denominations,
        for denomination in coins:
          # For all the amounts, up to the given amount,
          for amt in range(amount):
             # we check if this denomination can make up something <= amount?
             # if yes, then the number of ways to make up (amt + denomination) =
             # number of ways to make up amt,
             # so we accumulate the total number of ways to make up any amount i.
             if amt + denomination <= amount:
                combinations[amt + denomination] += combinations[amt]

        # return the total number of ways to make up the amount.
        return combinations[amount]
```

Basically for all the given denominations,
We loop through the range [0, amount],
And follow the same intuition as described in the previous paragraph
And keep updating the DP array.
This take us $\mathcal{O}(Amount * Number of denominations)$

Once done, every index i of the array now holds the total number of ways to make up the amount i using the denominations provided.

Finally, we return the number of ways to make up the given amount

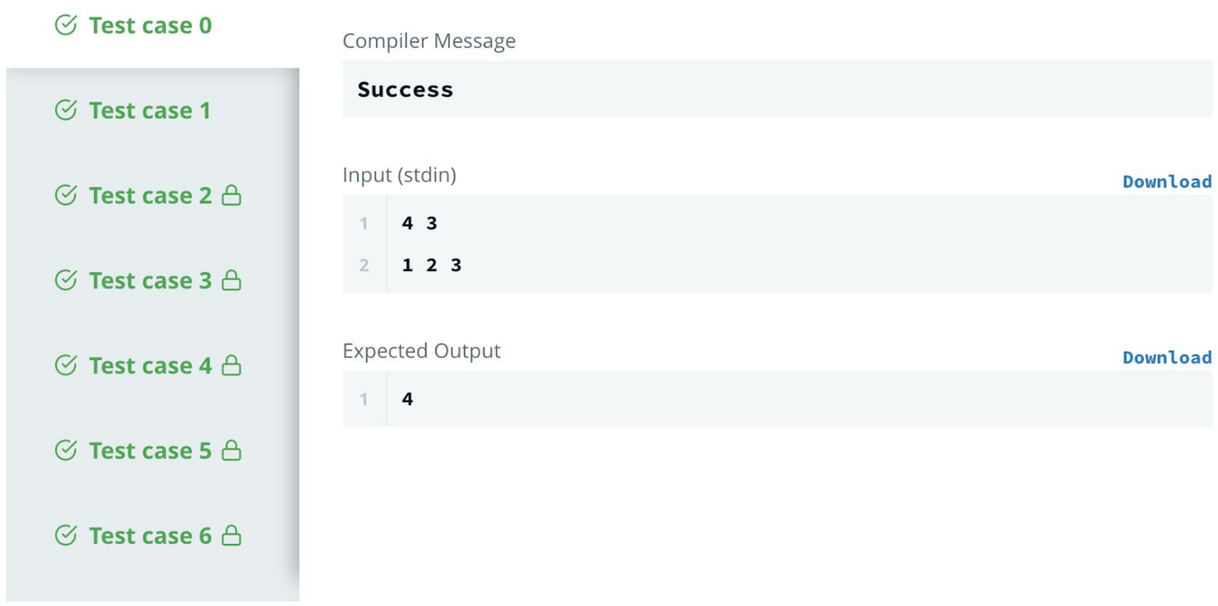
Run Time Analysis

Time Complexity - $\mathcal{O}(N * k)$ where N is the total amount to be made and k is the number of denominations provided to us
Space Complexity - $\mathcal{O}(N)$ where N is the total amount to be made up.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Coin%20Change/coin_change.py

Test Cases


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin)

Download

```
4 3
1 2 3
```

Expected Output

Download

```
4
```

https://www.hackerrank.com/challenges/coin-change/submissions/code/133709842

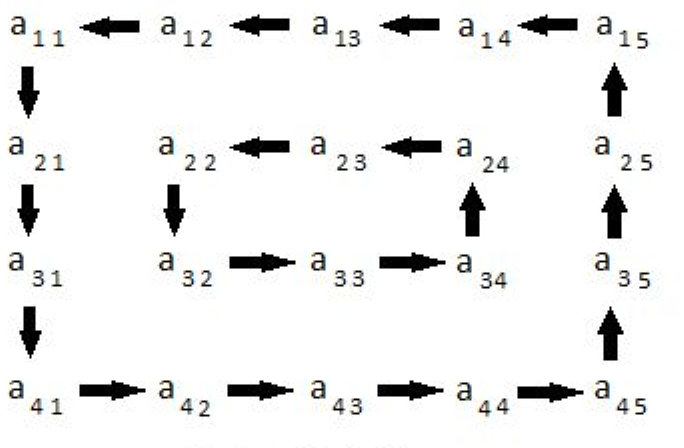
Problem 16

Matrix Layer Rotation

https://www.hackerrank.com/challenges/matrix-rotation-algo/problem

Problem Description

You are given a 2D matrix of dimension m x n and a positive integer r. You have to rotate the matrix r times and print the resultant matrix. Rotation should be in anti-clockwise direction. Rotation of a 4 x 5 matrix is represented by the following figure. Note that in one rotation, you have to shift elements by one step only.



Matrix Rotation

It is guaranteed that the minimum of m and n will be even.
As an example rotate the Start matrix by 2:

```
Start       First          Second
 1 2 3 4       2  3  4  5      3  4  5  6
12 1 2 5  ->   1  2  3  6 ->   2  3  4  7
11 4 3 6      12  1  4  7       1  2  1  8
10 9 8 7      11 10  9  8     12 11 10  9
```

Function Description

Complete the matrixRotation function in the editor below. It should print the resultant 2D integer array and return nothing.
matrixRotation has the following parameter(s):

- *matrix: a 2D array of integers*

- *r: an integer that represents the rotation factor*

Input Format

The first line contains three space-separated integers m, n and r, the number of rows and columns in matrix, and the required rotation.
The next m lines contain n space-separated integers representing the elements of a row of matrix.

Constraints

$2 \leq \mathrm{m}, \mathrm{n} \leq 300$
$1 \leq r \leq 10^9$
$min(m, n)\%2 = 0$
$1 \leq a_{ij} \leq 10^8 \text{ where } i \in [1 \ldots m] \text{ and } j \in [1 \ldots n]$

Output Format

Print each row of the rotated matrix as space-separated integers on separate lines.

Approach Description

First, we try to do exactly what the question says.
We find rings in the matrix and denote a ring by using 4 pointers to the left, right top and bottom of the ring. using left, right, top and bottom, we can denote the vertices of a 'layer'/ring as (top, left), (top, right), (bottom, left), (bottom, right).
Then for every ring, we do the following

```
        Do r times :
                Shift the elements of the top edge of the 'layer'/ring leftwards.
                Shift the elements of the left edge of the 'layer'/ring downwards.
                Shift the elements of the bottom edge of the 'layer'/ring rightwards.
                Shift the elements of the right edge of the 'layer'/ring upwards.
        After each iteration in the loop, they would move on to the next/inner
        'layer'/ring of the matrix and repeat.
```

We know there are no more rings when the condition left < right and top < bottom is not met.

While this approach is correct, it still times out on a few test cases.
We notice that this may be because we don't actually need to rotate each ring r times.
Since after some rotations, the ring remains unchanged. We find the modulo value (r % 2 * (right - left + bottom - top)), rotating the ring these many times would suffice.

This approach then passes all the test cases.

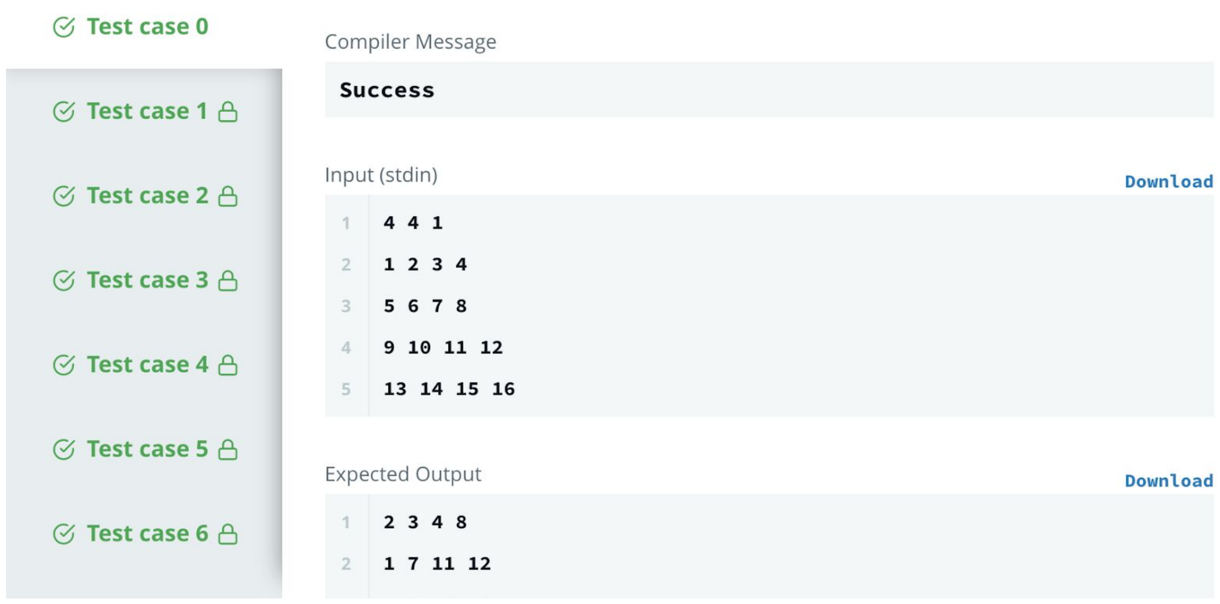
Run Time Analysis

Time Complexity - $\mathcal{O}((m+n)^2 * min(m,n))$
Space Complexity - O(1)

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Matrix%20Layer%20Rotation/matrix_layer_rotation.py

Test Cases


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin)

```
4 4 1
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Expected Output

```
2 3 4 8
1 7 11 12
```

https://www.hackerrank.com/challenges/matrix-rotation-algo/submissions/code/132432321

Problem 17

Bytelandian Gold Coins
https://www.codechef.com/problems/COINS

Problem Description

In Byteland they have a very strange monetary system.
Each Bytelandian gold coin has an integer number written on it. A coin n can be exchanged in a bank into three coins: n/2, n/3, and n/4. But these numbers are all rounded down (the banks have to make a profit).
You can also sell Bytelandian coins for American dollars. The exchange rate is 1:1. But you can not buy Bytelandian coins.
You have one gold coin. What is the maximum amount of American dollars you can get for it?

Input

The input will contain several test cases (not more than 10). Each test case is a single line with a number n, 0 <= n <= 1 000 000 000. It is the number written on your coin.

Output

For each test case output a single line, containing the maximum amount of American dollars you can make.

Approach Description

The thought behind this approach is simple.
We need to recursively calculate the maximum number of coins that can be made using an amount.
We observe the base cases when the amount is 0 and 1,
The maximum American dollars we can make are 0 and 1 respectively.
Calculating trivially, we see that this is case for all amounts < 12.
The pseudocode for the problem is as follows,

```
def _helper(amt: int) -> int:
  # If we already have the max we can make using that amount, return it, Base cases
  if amt in memo:
     return memo[amt]
  # Other wise find the max amount we can make for it using the given number of ways.
  else:
     memo[amt] = max(amt, _helper(amt // 2) + _helper(amt // 3) + _helper(amt // 4))
     return memo[amt]
```

Memo is initally a dictionary of the form {0: 0, 1: 1}
Although memo sould also be set to {0: 0, 1: 1 … 11: 11} upto amount 12, since we know by calculating trivally, but even our 2 base cases work just fine.

The maximum we can get for the coin at any step would be as follows,
max(amount, max_we_can_make_from(amount / 2) + max_we_can_make_from(amount / 3) + max_we_can_make_from(amount / 4))

And this is the recursive relation we implement in our function.

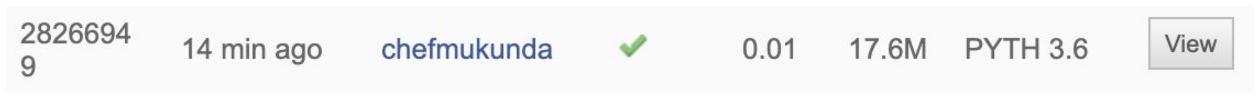
Run Time Analysis

Time Complexity - O(log(n))
Space Complexity - O(log(n))
Where n is the value of the gold coin.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Bytelandian%20Gold/bytelandian_gold.py

Test Cases



28266949	14 min ago	chefmukunda	✓	0.01	17.6M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28266949

Problem 18

The Next Palindrome
https://www.codechef.com/problems/PALIN

Problem Description

A positive integer is called a palindrome if its representation in the decimal system is the same when read from left to right and from right to left. For a given positive integer K of not more than 1000000 digits, write the value of the smallest palindrome larger than K to output. Numbers are always displayed without leading zeros.

Input

The first line contains integer t, the number of test cases. Followed by t lines containing integers K.

Output

For each K, output the smallest palindrome larger than K.

Approach Description

The problem asks us to find the next palindrome that is greater than the given number K.

First,
We check, if all the digits of k are 9 (like 9 or 99, or 999), then the next palindrome would always be obtained by adding 2 to it.
9 -> 11, 99 -> 101, 999 -> 1001 and so on.

Next, we iterate through the array and replace elements on the left half with that on the right half.
The pseudo code for this is as follows,

```
 for i in range(n // 2):
   num[~i] = num[i]
```

Where, ~i = n - i - 1

Now the number is a palindrome.
We check this number against the original number,
If this palindromic number is greater than the original number, we are done!
And we return this number as our result.
If not, go to the middle of the string and expand from the center towards the ends.

Note: if the input string is of odd length, the middle points would be at n // 2 and n // 2
If it is an even length string, the middle points would be n // 2 and (n // 2 + 1)
Since the center of the even length palindrome is between the two mids.
Consider
i = n // 2 + (0 if n % 2 else -1)
j = n // 2

Next expanding about the center using i and j we move towards the ends of the string.
If we keep seeing 9 as we go to the start of the string (i.e. leftward, using i),
We make num[i] = num[j] = ‘0’ i.e. i and its corresponding palindromic index are made ‘0’ and move i behind and j forward in the string.
Else if we don’t see a 9,
We simply increase the leftmost element at i (i.e. num[i]) by 1, this guarantees the new palindrome will definitely be greater than K, and we also set its corresponding palindromic index j, to the same value. Now we can simply return as we now have a palindrome greater than K. (Note we must and will stop and since we are guaranteed not to have all 9s in the string as we already handle this case)

The pseudo code for the above is as follows,

```
# We start exploring from the center to the edges.
while True:
  # If any of the number travelling leftward is 9,
  # make it zero and the corresponding mirror index zero too.
  if num[i] == '9':
     num[i] = num[j] = '0'
     i, j = i - 1, j + 1
  # Other-wise, we just add one to the leftward number, making it greater,
  # set its corresponding palindromic index to the same value.
  else:
     num[i] = num[j] = str(int(num[i]) + 1)
     return ''.join(num)
```

This way we are always guaranteed,

1. The resulting number is a palindrome (since we make all changes symmetrically)
2. The resulting number is greater than the input number (since we ever only increase the number, never decrease)

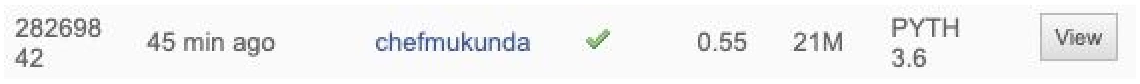
Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(n)
Where n is the length of the input string.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/The%20Next%20Palindrome/next_palindrome.py

Test Cases



28269842	45 min ago	chefmukunda	✓	0.55	21M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28269842

Problem 19

Cryptopangrams
https://codingcompetitions.withgoogle.com/codejam/round/0000000000051705/000000000008830b

Problem Description

On the Code Jam team, we enjoy sending each other pangrams, which are phrases that use each letter of the English alphabet at least once. One common example of a pangram is "the quick brown fox jumps over the lazy dog". Sometimes our pangrams contain confidential information — for example, CJ QUIZ: KNOW BEVY OF DP FLUX ALGORITHMS — so we need to keep them secure.
We looked through a cryptography textbook for a few minutes, and we learned that it is very hard to factor products of two large prime numbers, so we devised an encryption scheme based on that fact. First, we made some preparations:

- *We chose 26 different prime numbers, none of which is larger than some integer N.*
- *We sorted those primes in increasing order. Then, we assigned the smallest prime to the letter A, the second smallest prime to the letter B, and so on.*
- *Everyone on the team memorized this list.*

Now, whenever we want to send a pangram as a message, we first remove all spacing to form a plaintext message. Then we write down the product of the prime for the first letter of the plaintext and the prime for the second letter of the plaintext. Then we write down the product of the primes for the second and third plaintext letters, and so on, ending with the product of the primes for the next-to-last and last plaintext letters. This new list of values is our ciphertext. The number of values is one smaller than the number of characters in the plaintext message.
For example, suppose that N = 103 and we chose to use the first 26 odd prime numbers because we worry that it is too easy to factor even numbers. Then A = 3, B = 5, C = 7, D = 11, and so on, up to Z = 103. Also, suppose that we want to encrypt the CJ QUIZ... pangram above, so our plaintext is CJQUIZKNOWBEVYOFDPFLUXALGORITHMS. Then the first value in our ciphertext is 7 (the prime for C) times 31 (the prime for J) = 217; the next value is 1891, and so on, ending with 3053.
We will give you a ciphertext message and the value of N that we used. We will not tell you which primes we used, or how to decrypt the ciphertext. Do you think you can recover the plaintext anyway?

Input

The first line of the input gives the number of test cases, T. T test cases follow; each test case consists of two lines. The first line contains two integers: N, as described above, and L, the *length of the list of values in the ciphertext. The second line contains L integers: the list of values in the ciphertext.*

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is a string of L + 1 uppercase English alphabet letters: the plaintext.

Limits

1 ≤ T ≤ 100.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.
25 ≤ L ≤ 100.
The plaintext contains each English alphabet letter at least once.

Approach Description

A brute force solution would be to just find all the primes numbers up to N and try and divide the given ciphers by them. This is, however, is too slow to pass the test cases.

From the problem, it is easy to see that every consecutive number in the test case would have in common a prime factor. This way we can simply keep finding the greatest common divisor of every consecutive pair of numbers and get all the primes.
Next, we can sort these prime numbers to be able to map them to alphabets This takes us constant time, where there are n prime numbers since in our case are guaranteed to be 26 so as to map to each alphabet.

This, however, does not work, we miss to account for cases where there may be repeated pairs like ABABAB and BABABA, both these would have the same ciphers in the start (i.e. the product of A and B) and we would get the cipher itself as the gcd.
So we pick the first cipher and find the next dissimilar cipher to it.
There is guaranteed to be an end to the pattern since the phrase must use all 26 alphabets as it is a pangram by definition.
Now that we have 2 dissimilar ciphers, we find their common prime factor and now we can continue to decipher the remainder of the sequence after that as before.
We record all the primes we encounter to build the prime to alphabet mapping.
Since we already have the original order in which the primes occurred in the text, and its mapping, we can now easily decode the cryptogram.

Run Time Analysis

Time Complexity - O(log(n) + |L|)
Space Complexity - O(1)

Where |L| length of the cipher

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Cryptopangrams/cryptopangrams.py

Test Cases

Attempt 10 ✓ ✓ Oct 21 2019, 20:38

Problem 20

Yet Another KMP Problem

https://www.hackerrank.com/challenges/kmp-problem/problem

Problem Description

This challenge uses the famous KMP algorithm. It isn't really important to understand how KMP works, but you should understand what it calculates.

A KMP algorithm takes a string, S, of length N as input. Let's assume that the characters in S are indexed from 1 to N; for every prefix of S, the algorithm calculates the length of its longest valid border in linear complexity. In other words, for every i (where $1 \leq i \leq N$) it calculates the largest l (where $0 \leq l \leq i-1$) such that for every p (where $1 \leq p \leq l$) there is S[p] = S[i - l + p].

Here is an implementation example of KMP:

```
kmp[1] = 0;
for (i = 2; i <= N; i = i + 1){
   l = kmp[i - 1];
   while (l > 0 && S[i] != S[l + 1]){
      l = kmp[l];
   }
   if (S[i] == S[l + 1]){
      kmp[i] = l + 1;
   }
   else{
      kmp[i] = 0;
   }
}
```

Given a sequence $x_1, x_2 x_{26}$, construct a string, S, that meets the following conditions:

1. *The frequency of letter 'a' in S is exactly x_1, the frequency of letter 'b' in S is exactly x_2, and so on.*
2. *Let's assume characters of S are numbered from 1 to N, where $\sum_{i=1}^{n} x_i = N$. We apply the KMP algorithm to S and get a table, kmp, of size p. You must ensure that the sum of kmp[i] for all i is minimal.*

If there are multiple strings which fulfill the above conditions, print the lexicographically smallest one.

Input Format

A single line containing 26 space-separated integers describing sequence x.

Constraints

- *The sum of all x_i will be a positive integer $\leq 10^5$.*

Output Format
Print a single string denoting S.

Approach Description

We that the KMP algorithm calculates the kmp array where for every i in the array, it finds the prefix that is also the suffix of the string, commonly known as the border, in linear time.

Given the frequency of every alphabet in a string, we are asked to construct a string that minimizes the sum of this kmp array.
To do this, we must ensure that the border is as small as possible for every i.

We do this by case analysis.
First,
If the string contains only 1 character, we simply return a string with that character repeated as many times as its given frequency.
Second,
If there are more than 1 characters in the string to be made,
Find the character that has the minimum frequency (Let us call the character with the least frequency _min) and is the smallest lexicographically in case of a tie.
Next, if the _min is not the smallest lexicographical character in the string or there is only 1 of it,
We make a string of the form _min concatenated with all the other characters repeated as many times as their frequency, in lexicographical order.
Third,
If the _min character is also the lexicographically smallest character,
We Start the string with _min repeated twice (We would have at-least 2 since we checked for it being earlier) Then we interleave it with the remaining characters till the _min character is exhausted as (# _min # _min ...)
Finally, the string would be of the form _min_min#_min#_min#_min….###
Here # are characters repeated as many times as their frequency, in lexicographical order.

All three branches take at most O(n) time

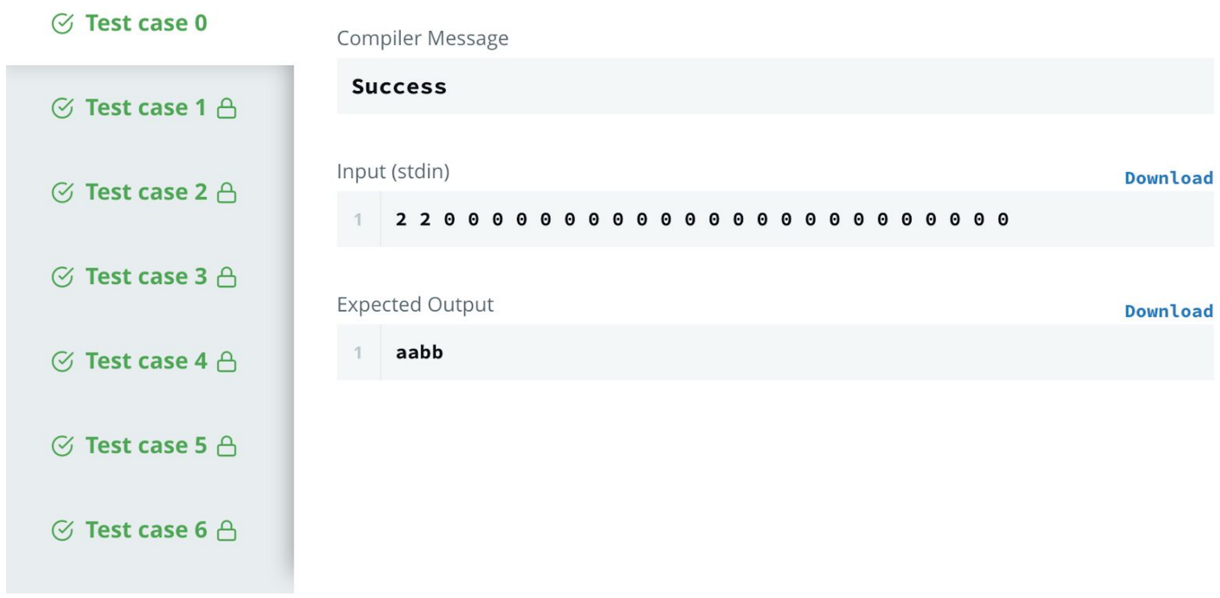
Run Time Analysis

Time Complexity - O(n) where n is the size of the input (size of given input string)
Space Complexity - O(1)

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Yet%20Another%20KMP%20Problem/another_kmp.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Expected Output

```
aabb
```

https://www.hackerrank.com/challenges/kmp-problem/submissions/code/132630884

Problem 21

Fogone Solution
https://codingcompetitions.withgoogle.com/codejam/round/0000000000051705/0000000000088231

Problem Description
Someone just won the Code Jam lottery, and we owe them N jamcoins! However, when we tried to print out an oversized check, we encountered a problem. The value of N, which is an integer, includes at least one digit that is a 4... and the 4 key on the keyboard of our oversized check printer is broken.
Fortunately, we have a workaround: we will send our winner two checks for positive integer amounts A and B, such that neither A nor B contains any digit that is a 4, and A + B = N. Please help us find any pair of values A and B that satisfy these conditions.

Input
The first line of the input gives the number of test cases, T. T test cases follow; each consists of one line with an integer N.

Output
For each test case, output one line containing Case #x: A B, where x is the test case number (starting from 1), and A and B are positive integers as described above.
It is guaranteed that at least one solution exists. If there are multiple solutions, you may output any one of them. (See "What if a test case has multiple correct solutions?" in the Competing section of the [FAQ](). This information about multiple solutions will not be explicitly stated in the remainder of the 2019 contest.)

Limits
1 ≤ T ≤ 100.
Time limit: 10 seconds per test set.
Memory limit: 1GB.
At least one of the digits of N is a 4.

Approach Description
The problem is simple in fact that we can make the broken digit 4 using 3 and 1.
So we follow a simple process,

We make two lists one containing all the digits in N say A and one with all 0’s with the same length as A say B.
We iterate through A and any time we spot a 4, we make it a 3 and make the corresponding digit at B a 1 so now they add up to a 4 and we can represent that digit when adding A and B.
Once this is done, we remove all the leading zeros from B is any
and return A and B

Run Time Analysis

Time Complexity - O(n), where n is the number of digits in the input integer N.
Space Complexity - O(n), where n is the size required to represent the integer N.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Forgone%20Solution/forgone.py

Test Cases

Attempt 5 ✓ ✓ ✓ Oct 20 2019, 18:31

Problem 22

Ordering the Soldiers

https://www.codechef.com/problems/ORDERS

Problem Description

In Byteland it is always the military officer's main worry to order his soldiers on parade correctly. Luckily, ordering soldiers is not really such a problem. If a platoon consists of n men, all of them have different rank (from 1 - lowest to n - highest) and on parade, they should be lined up from left to right in increasing order of rank.

Sounds simple, doesn't it? Well, Sgt Johnny thought the same, until one day he was faced with a new command. He soon discovered that his elite commandos preferred to do the fighting and leave the thinking to their superiors. So, when at the first rollcall the soldiers lined up in fairly random order it was not because of their lack of discipline, but simply because they couldn't work out how to form a line in correct order of ranks. Sgt Johnny was not at all amused, particularly as he soon found that none of the soldiers even remembered his own rank. Over the years of service, every soldier had only learned which of the other soldiers were his superiors. But Sgt Johnny was not a man to give up easily when faced with a true military challenge. After a moment's thought, a solution of brilliant simplicity struck him and he issued the following order: "men, starting from the left, one by one, do: (step forward; go left until there is no superior to the left of you; get back in line).". This did indeed get the men sorted in a few minutes. The problem was solved... for the time being.

The next day, the soldiers came in exactly the same order as the day before and had to be rearranged using the same method. History repeated. After some weeks, Sgt Johnny managed to force each of his soldiers to remember how many men he passed when going left, and thus make the sorting process even faster.

If you know how many positions each man has to walk to the left, can you try to find out what order of ranks the soldiers initially line up in?

Input

The first line of input contains an integer t<=50, the number of test cases. It is followed by t test cases, each consisting of 2 lines. The first line contains a single integer n (1<=n<=200000). The second line contains n space-separated integers w_i, denoting how far the i-th soldier in line must walk to the left when applying Sgt Johnny's algorithm.

Output

For each test case, output a single line consisting of n space-separated integers - the ranks of the soldiers, given from left to right in their initial arrangement.

Approach Description

To initially approach this we simply simulate the algorithm mentioned above.
Since each of them know how many positions to move to the left, we can get the final position of the soldiers as follows

```
final_pos = []
for i in range(n): final_pos.insert(i - w[i], i)
```

The final_pos array to holds the arrangement after the soldiers move the required steps.
The for loop simulates the moving of a soldier to their final position,
the index of the array represents the rank of the soldier,
and the value is the position where the soldier was initially.
The for loop and the insert in the loop take overall $\mathcal{O}(n^2)$ time

Next, we build an array of the initial position of the soldiers as follows,

```
initial_order = [_ for _ in range(n)]
for i in range(n): initial_order[final_pos[i]] = i + 1
```

The initial_order array stores the initial order the soldier arrived in.
After moving, the soldiers are now in increasing order of rank,
and we know where they were initially from the values in final_pos.
The value at final_pos[i] is where soldier ranked i was initially standing.

We can, however, do this in one $\mathcal{O}(n^2)$ loop
We, already know the final rank position of all the soldiers and how far left they need to move,

```
# Array holding the final ranking position.
ranks = [i for i in range(1, n + 1)]

# The initial order would then be,
# rank at (end of ranks array - steps moved to left).
for i in range(n - 1, -1, -1): w[i] = ranks.pop(i - w[i])
```

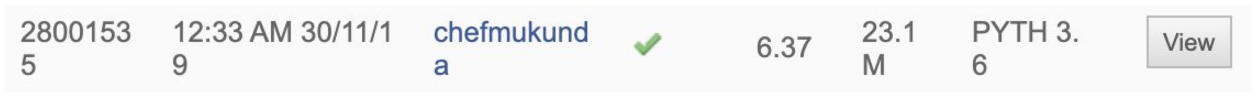
Run Time Analysis

Time Complexity - $\mathcal{O}(n^2)$
Space Complexity - $\mathcal{O}(n)$
where n is the number of soldiers

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Ordering%20the%20Soldiers/ordering_soldiers.py

Test Cases



28001535	12:33 AM 30/11/19	chefmukunda	✓	6.37	23.1M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28001535

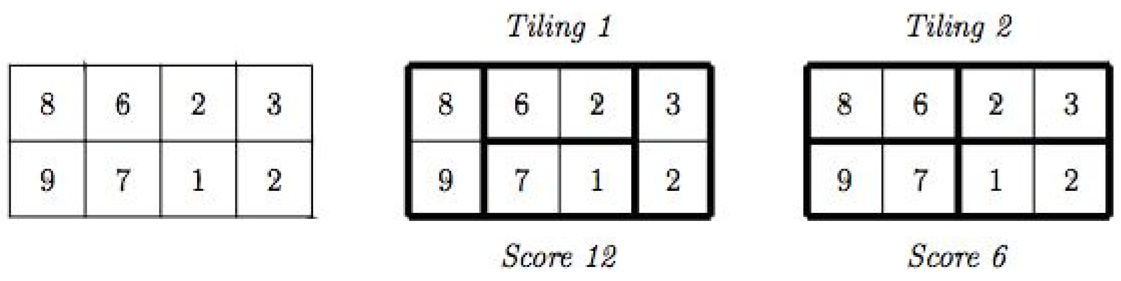
Problem 23

Domino Solitaire

https://www.codechef.com/problems/DOMSOL

Problem Description

In Domino Solitaire, you have a grid with two rows and N columns. Each square in the grid contains an integer A. You are given a supply of rectangular 2 × 1 tiles, each of which exactly covers two adjacent squares of the grid. You have to place tiles to cover all the squares in the grid such that each tile covers two squares and no pair of tiles overlap.
The score for a tile is the difference between the bigger and the smaller number that are covered by the tile. The aim of the game is to maximize the sum of the scores of all the tiles.
Here is an example of a grid, along with two different tilings and their scores.The score for Tiling 1 is 12 = (9 − 8) + (6 − 2) + (7 − 1) + (3 − 2) while the score for Tiling 2 is 6 = (8 − 6) + (9 − 7) + (3 − 2) + (2 − 1). There are other tilings possible for this grid, but you can check that Tiling 1 has the maximum score among all tilings. Your task is to read the grid of numbers and compute the maximum score that can be achieved by any tiling of the grid.



Your task is to read the grid of numbers and compute the maximum score that can be achieved by any tiling of the grid.

Input

The first line contains one integer N, the number of columns in the grid. This is followed by 2 lines describing the grid. Each of these lines consists of N integers, separated by blanks.

Output

A single integer indicating the maximum score that can be achieved by any tiling of the given grid.

Approach Description

For this problem, we follow a simple DP approach.
The intuition is similar to the coin change problem.

For only 1 column, we can only place one tile i.e. vertically.
For columns 0 & 1, we can place two tiles either both vertically or
both horizontally only (since no overlap is allowed).

Then we simply use these base results to calculate the maximum up to the next column,
At each time we have 2 choices,
Take everything up to the N-1 column and place a vertical tile
Or take everything up to the N-2 column and place 2 horizontal tiles

The code explaining each line is as follows,

```
# Array to hold maximum score made up to column i in the grid, where i is
# represents by the column number (from 0 up to N).
score = [0] * N

# For only 1 column, we can only place one tile i.e. vertically.
score[0] = abs(grid[0][0] - grid[1][0])

# For columns 0 & 1, we can place two tiles either both vertically or
# both horizontally only (since no overlap is allowed),
# we calculate the max score as follows,
# First term represents placing two tiles horizontally,
# The second one represents placing two tiles vertically,
score[1] = max(score[0] + abs(grid[0][1] - grid[1][1]),
           abs(grid[0][0] - grid[0][1]) + abs(grid[1][0] - grid[1][1]))

# We calculate the upto score for the following columns by using these base
# results.
for i in range(2, N):
  # First term represents that everything upto col i - 1 is occupied,
  # So we only place a single vertical tile at column i.
  # Second term represents the case where everything upto column i - 2 is
  # occupied so we can place 2 horizontal tiles at each row covering
  # columns i - 1 and i.
  score[i] = max(score[i - 1] + abs(grid[0][i] - grid[1][i]),
             score[i - 2] + abs(grid[0][i] - grid[0][i - 1]) +
             abs(grid[1][i] - grid[1][i - 1]))
```

The above takes O(N) time and space.
We see here in the DP relation that we only need results upto the past 2 columns.
So we can reduce our space complexity simply by storing only the past 2 results instead of the all of them.

It may be easier to think of the problem in the reverse way.
Consider that we have the maximal arrangement.
Now to remove something from the last column of the board,
We can either remove 1 vertical tile, in which case the arrangement of everything up to column N-1 would have been optimal
Or
We can either remove 2 horizontal tiles, in which case the arrangement of everything up to column N-2 would have been optimal.
We can carry on the process till we reach column 1 or 0, our base cases!
Whose maximal arrangements we already know.

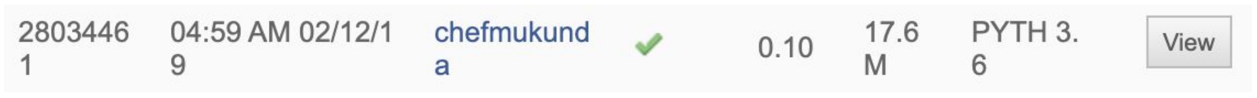
Run Time Analysis

Time Complexity - O(N)
Space Complexity - O(1)

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Domino%20Solitaire/domino_solitaire.py

Test Cases



28034461	04:59 AM 02/12/19	chefmukunda	✓	0.10	17.6M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28034461

Problem 24

Password Cracker
https://www.hackerrank.com/challenges/password-cracker/problem

Problem Description

There are N users registered on a website CuteKittens.com. Each of them has a unique password represented by pass[1], pass[2], ..., pass[N]. As this a very lovely site, many people want to access those awesomely cute pics of the kittens. But the adamant admin does not want the site to be available to the general public, so only those people who have passwords can access it.
Yu, being an awesome hacker finds a loophole in the password verification system. A string which is a concatenation of one or more passwords, in any order, is also accepted by the password verification system. Any password can appear 0 or more times in that string. Given access to each of the n passwords, and also have a string
$loginAttempt, determinewhetherthisstringbeacceptedbythepasswordverificationsystemofthewebsite. Ifallofthe$
loginAttempt$ string can be created by concatenating password strings, it is accepted.
For example, if there are 3 users with passwords = [abra, ka, dabra], then some of the valid combinations are abra (passwords[1]), kaabra (passwords[2] + passwords[1]), kadabraka (passwords[2] + passwords[3] + passwords[1]), kadabraabra(passwords[2] + passwords[3] + passwords[1]) and so on. Supplying abra ka dabra, concatenated, passes authentication.

Function Description

Complete the passwordCracker function in the editor below. It should return the passwords as a single string in the order required for the password to be accepted, each separated by a space. If it is not possible to form the string, return the string WRONG PASSWORD.
passwordCracker has the following parameters:
- passwords: a list of password strings
- loginAttempt: the string to attempt to create
Input Format
The first line contains an integer t, the total number of test cases.
Each of the next t sets of three lines is as follows:
- The first line of each test case contains n, the number of users with passwords.
- The second line contains n space-separated strings, passwords[i], that represent the passwords of each user.
- The third line contains a string, loginAttempt, which Yu must test for acceptance.

Constraints

- $1 \le t \le 10$
- $1 \le n \le 10$
- $passwords[i] \ne passwords[j], 1 \le i < j \le N$
- $1 \le |passwords[i]| \le 10$, where $i \in [1, n]$
- $1 < |loginAttempt| \le 2000$
- loginAttempt and passwords[i] contain only lowercase latin characters ('a'-'z').

Output Format

For each valid string, Yu has to print the actual order of passwords, separated by space, whose concatenation results into loginAttempt. If there are multiple solutions, print any of them. If loginAttempt can't be accepted by the password verification system, then print WRONG PASSWORD.

Approach Description

For this problem, we use a dfs/backtracking with memoization approach to prevent any overlapping calculations.
The intuition is that we keep trying known passwords to match with the start of the loginAttempt password, each time.
The pseudo-code is as follows,
If the loginAttempt is already in the memo array, we return the answer.
If the loginAttempt is already a valid password, we store it in the memo and return the result.
Otherwise, For all the known passwords,
    we try and match the prefix of the current loginAttempt to the password, if it matches, we call password cracker on the remainder of the loginAttempt and save the result in the memo.
    If the result is not False, we store it in the memo and return the result.

Finally, if none of the passwords match, we return WRONG PASSWORD.

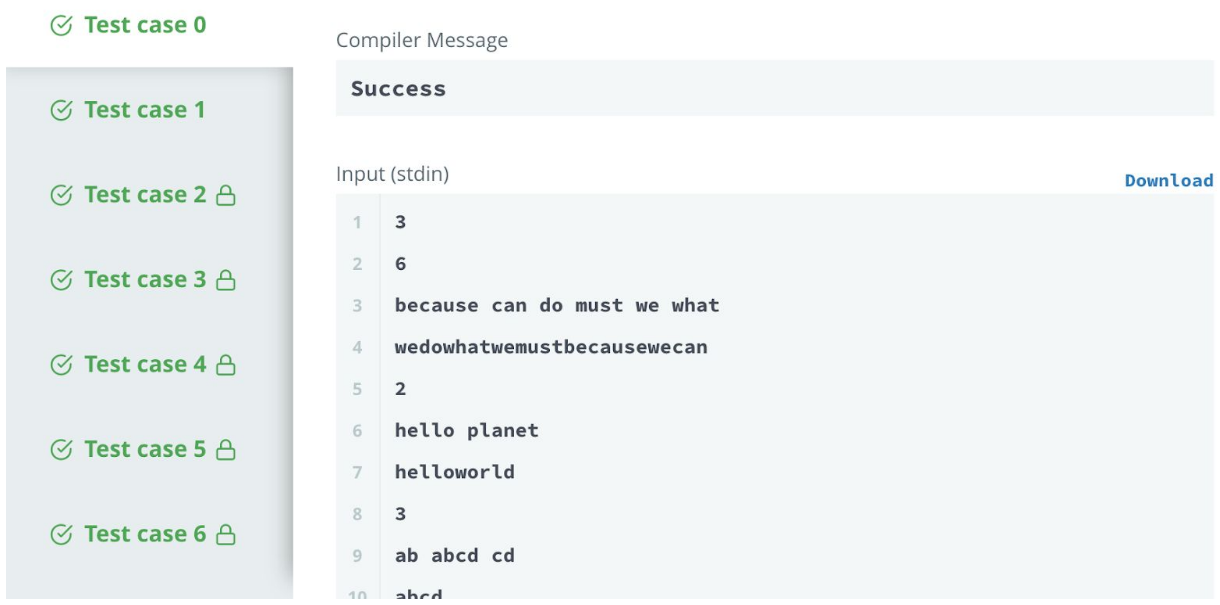
Run Time Analysis

Time Complexity - $\mathcal{O}(b^d)$
Space Complexity - $\mathcal{O}(b)$
Where b is the number of passwords and d is the length of the loginAttempt password.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Password%20Cracker/password_cracker.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
3
6
because can do must we what
wedowhatwemustbecausewecan
2
hello planet
helloworld
3
ab abcd cd
abcd
```

https://www.hackerrank.com/challenges/password-cracker/submissions/code/133964300

Problem 25

Waffle Choppers

https://codingcompetitions.withgoogle.com/codejam/round/0000000000007883/000000000003005a

Problem Description

The diners at the Infinite House of Pancakes have gotten tired of circular pancakes, so the chefs are about to offer a new menu option: waffles! As a publicity stunt, they have made one large waffle that is a grid of square cells with R rows and C columns. Each cell of the waffle is either empty or contains a single chocolate chip.
Now it is time for the chefs to divide up the waffle among their hungry diners. A horizontal cut runs along the entire gridline between two of the rows; a vertical cut runs along the entire gridline between two of the columns. For efficiency's sake, one chef will make exactly H different horizontal cuts and another chef will make exactly V different vertical cuts. This will conveniently create one piece for each of the (H + 1) × (V + 1) diners. The pieces will not necessarily all be of equal sizes, but that is fine; market research has shown that the diners do not care about that. What the diners do care about is the number of chocolate chips they get, so each piece must have exactly the same number of chocolate chips. Can you determine whether the chefs can accomplish this goal using the given numbers of horizontal and vertical cuts?

Input

The first line of the input gives the number of test cases, T; T test cases follow. Each begins with one line containing four integers R, C, H, and V: the number of rows and columns in the waffle, and the exact numbers of horizontal and vertical cuts that the chefs must make. Then, there are R more lines of C characters each; the j-th character in the i-th of these lines represents the cell in the i-th row and the j-th column of the waffle. Each character is either @, which means the cell has a chocolate chip, or ., which means the cell is empty.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is POSSIBLE if the chefs can accomplish the goal as described above, or IMPOSSIBLE if they cannot.

Limits

1 ≤ T ≤ 100.
Time limit: 6 seconds per test set.
Memory limit: 1GB.

Approach Description

We need to be able to make the given number of horizontal and vertical cuts to be able to distribute the number of chips in the waffle equally in each part.
Making h horizontal cuts and v vertical cuts divides the waffle into p = (h + 1)*(v + 1) parts.

We iterate through each cell of the waffle and count the total number of chips in it.
This take us O(r*c) time.
If the number of chips is not exactly divisible by p, our task is impossible.

Next, we iterate through each cell of the waffle and count the number of chips in each row,
If the number of chips in any row exceed h_avg = average number of chips per horizontal slice,
Our task fails.
Otherwise, we record a horizontal cut as soon as we find h_avg number of chips

We do the same now but iterating column-wise to account for the vertical cuts.
Next, we iterate through each cell of the waffle and count the number of chips in each column,
If the number of chips in any column exceed v_avg = average number of chips per vertical slice,
Our task fails.
Otherwise, we record a vertical cut as soon as we find v_avg number of chips.

Now finally we must ensure that each of the p parts has exactly (total number of chips / p) chips in each part,
This can be done simply by iterating through the entire waffle and counting the number of chips in each part, if any of parts do not have the exact number of chips, our task ahs failed.

If all the above checks pass, it is possible to cut up the waffle with the given number of vertical and horizontal cuts.

Run Time Analysis

Time Complexity - O(r * c)
Space Complexity - O(h + v)
Where r and c are the number of rows and columns in the waffle.
And v, h are the number of vertical and horizontal cuts to be made.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Waffle%20Choppers/waffle_choppers.py

Test Cases

Attempt 1 ✓ ✓ Dec 1 2019, 03:10

Problem 26

Chief Hopper

https://www.hackerrank.com/challenges/chief-hopper/problem

Problem Description

Chief's bot is playing an old DOS-based game. There is a row of buildings of different heights arranged at each index along a number line. The bot starts at building 0 and at a height of 0. You must determine the minimum energy his bot needs at the start so that he can jump to the top of each building without his energy going below zero.
Units of height relate directly to units of energy. The bot's energy level is calculated as follows:

- *If the bot's botEnergy is less than the height of the building, his newEnergy = botEnergy - (height - botEnergy)*
- *If the bot's botEnergy is greater than the height of the building, his newEnergy = botEnergy + (height - botEnergy)*

For example, building heights are given as h = [2, 3, 4, 3, 2]. If the bot starts with botEnergy = 4, we get the following table:

botEnergy	*height*	*delta*
4	*2*	*+2*
6	*3*	*+3*
9	*4*	*+5*
14	*3*	*+11*
25	*2*	*+23*
48		

That allows the bot to complete the course, but may not be the minimum starting value. The minimum starting botEnergy, in this case, is 3.

Function Description

Complete the chiefHopper function in the editor below. It should return an integer that represents the minimum starting botEnergy that will allow completion of the course.
chiefHopper has the following parameter(s):

- *arr: an array of integers that represent building heights*

Input Format
The first line contains an integer n, the number of buildings.
The next line contains n space-separated integers h[1] … h[n] representing the heights of the buildings.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq h[i] \leq 10^5, i \in [1, n]$

Output Format

Print a single integer representing minimum units of energy required to complete the game.

Approach Description

The intuition for this problem comes from elementary physics equations of energy.
If the bot started at the minimum possible energy,
The botEnergy at the end should be e = zero.
Now we can simply calculate the initial energy of the bot by traversing backward in the building heights array and update the initial energy e as e = Ceil((e + height) / 2), where ceil is the ceiling function.

We use ceil since,
On each reaching height, h the bots botenergy would be 2*botEnergy - h.
So we solve it backwards initial energy to be zero
e=(e+h)/2 or (e+h)/2+1 as per the equations given in the problem statement.
depending on the value of e+h, here even and odd respectively.
Using ceil allows us to account for both the cases eliminating the need to check if e+h is even or odd.

The psuedo code is as follows,

```
# For the minimum initial energy, the bot mush have 0 energy left at the end
# of the last building.
e = 0
# Now we calculate backwards how much initial energy would be required.
# if e < h, e = 2e - h
for height in arr[::-1]:
        e = (e + height + 1) // 2
return e
```

This ‘greedy’ approach takes O(n) time and passes all the tests successfully.

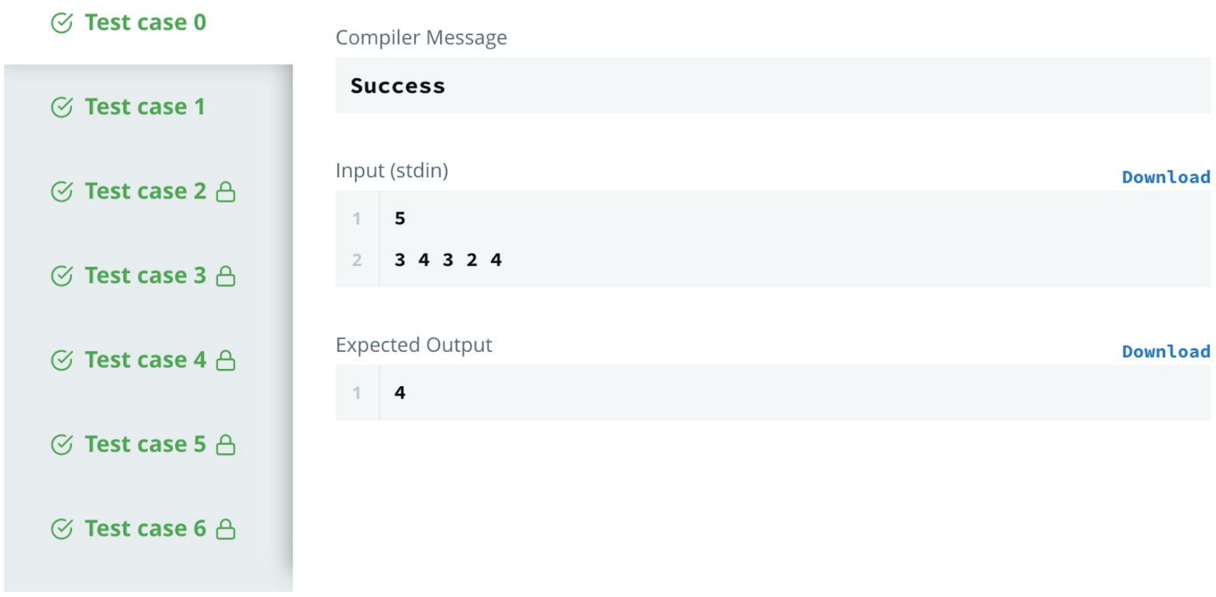
Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(1)
Where n is the number of buildings which the bot has to jump over.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Chief%20Hopper/chief_hopper.py

Test Cases



Compiler Message

```
Success
```

Input (stdin)

```
5
3 4 3 2 4
```

Expected Output

```
4
```

https://www.hackerrank.com/challenges/chief-hopper/submissions/code/132341326

Problem 27

Vogon Zoo

https://www.codechef.com/problems/VOGOZO

Problem Description

On the icy planet Zorg, the Vogons are putting together a zoo. One cage will house a collection of Kubudu dragons. Unlike the limited number of blood types found in other creatures, Kubudu dragons have a large variety of blood types. Each dragon's blood type is fixed when it is born and is given by a positive integer.
These blood types determine how the dragons interact with each other. Two dragons whose blood types are close to each other will quickly start fighting and eventually destroy each other. At any given time, there is a threshold K such that it is safe to put two Kubudu dragons in the same cage only if their blood types differ by K or more.
A batch of freshly hatched Kubudu dragons has arrived. Each dragon has had its blood tested and is tagged with its blood type. The Vogon zookeeper would like to determine the size of the largest collection of dragons from this batch that can safely be placed in a single cage. Given the dragons' fighting tendencies, this means that for each pair of dragons in the cage, their blood types should differ by at least K.
For instance, suppose that K is 3 and there are 12 Kubudu dragons whose blood types are 1, 5, 6, 1, 8, 3, 12, 2, 13, 7, 9 and 11. From this batch, the maximum number of dragons that can be placed safely in the same cage is 4—for example, the dragons with blood types: 6, 12, 2 and 9.
You will be given the blood types of N Kubudu dragons and the threshold K. Your task is to compute the size of the largest group of dragons from this collection that can safely be placed in the same cage.

Input

The first line of input has two space-separated integers N and K, where N is the number of Kubudu dragons and K is the threshold below which they fight, as described above. The second line of input consists of N space-separated integers, the blood types of the N dragons.

Output

A single integer, the size of the largest collection of dragons that can be safely placed in the same cage.
Test data
In all cases, $1 \leq N \leq 10^6$. In 30% of the inputs, $1 \leq N \leq 5000$. The blood types of the dragons lie in the range 1 to 10^7

Approach Description

In the problem, we are given the blood types of all the kubudu dragons and asked to place the maximum number of kubudu dragons in a cage whose blood types differ by at least k.

To approach this problem,
We first sort the kubudu dragons by their given blood types, and this takes us O(nlog(n)) time where n is the number of kubudu dragons.
Now we simply find the next dragon that can be placed with the first dragon of the smallest blood type.
We do this since taking a dragon with any higher blood type would only reduce the number of dragons we can fit in the same cage.
When the next dragon is found we search for another dragon k bigger in blood type with the new dragon and so on.
We keep a variable to keep the count of the dragons currently in the cage.
The pseudo-code for the above is as follows.

```
total, curr = 1, blood_types[0]

for blood_type in blood_types:
  if blood_type >= curr + K:
        total, curr = total + 1, blood_type
```

This takes us O(n) time in the after sorting the array.
And constant space.

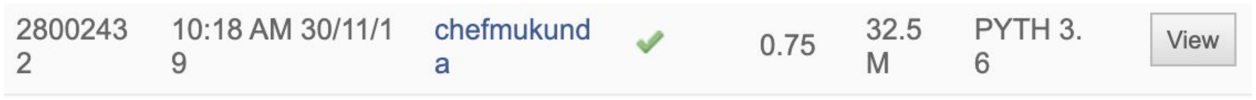
Run Time Analysis

Time Complexity - O(nlog(n))
Space Complexity - O(1)
where n is the number of kubudu dragons.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Vogon%20Zoo/vogon_zoo.py

Test Cases



28002432	10:18 AM 30/11/19	chefmukunda	✓	0.75	32.5M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28002432

Problem 28

Cultural Programme

https://www.codechef.com/problems/CULPRO

Problem Description

The Siruseri Welfare Association has organized a Cultural Programme for families in the area, to build up community spirit. The Cultural Programme is being held at the Siruseri Community Hall. Inside the hall, children perform songs and dances. Outside, local restaurants have set up stalls selling snacks.

As is typical on such an occasion, members of the audience drift in and out of the hall during the programme. An observant office-bearer of the Siruseri Welfare Association notes down the times at which people enter and leave the hall. Suppose that is N. He wants to know the maximum size of the audience was during the course of the programme.

For convenience, he writes down each time as a single integer - A_i, the number of minutes from the start of the programme for the ith person. Also, the door of the hall is narrow, so at any time, either one person can enter or one person can leave the hall, but not both. Thus, each entry and exit time that is noted down is distinct.

For example, suppose the observations noted down are the following. Each line denotes the entry time and exit time of one person. (The identity of the person is not important—the same person may enter and leave the hall many times. For instance, in the example below, it might well be that the entries and exits recorded at Serial Nos 2 and 5 refer to the same person.)

Your task is to read the list of entry and exit times and compute the maximum size of the audience during the programme.

Input

The first line of input is a single integer, N, the number of entries and exits recorded. This is followed by N lines of input. Each of these lines consists of two integers, separated by a space, describing one entry and exit. The first integer is the entry time and the second integer is the exit time..

Output

A single integer, denoting the maximum size of the audience during the performance.

Approach Description

The problem asks us to find the maximum number of people in a hall at any time, given the entry and exit times each time a person enters or leaves the hall.

For this, we would need to know as soon as an event takes place, either entry or exit.
We combine the arrays holding the entry and exit times into one and sort it.
This takes us O(2n*log(2n)) time since the resulting array is of size 2n which is asymptotically the same as O(nlog(n)).

We keep a set of the entry times to know if an event was an entry or an exit since it can’t be both as guaranteed by the problem.

We keep two variables, one to keep a track of the current number of people in the hall, and another to keep a track of the maximum number of people we have seen till that time.
Now we iterate through this sorted array and find the maximum number of people at any time as follows,

```
for record in sorted(entries + exits):
   people = people + 1 if record in entry_set else people - 1
   max_people = max(max_people, people)
```

And simply return the maximum number of people.

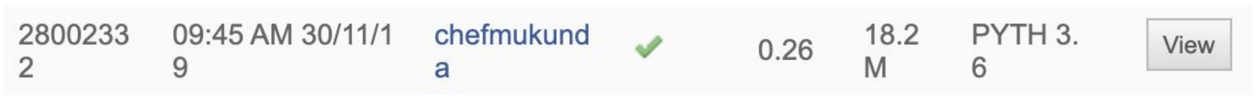
Run Time Analysis

Time Complexity - O(nlog(n))
Space Complexity - O(n)

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Cultural%20Programme/cultural_programme.py

Test Cases



28002332	09:45 AM 30/11/19	chefmukunda	✓	0.26	18.2M	PYTH 3.6	View

https://www.codechef.com/viewsolution/28002332

Problem 29

Reaching Points

https://leetcode.com/problems/reaching-points/

Problem Description

A move consists of taking a point (x, y) and transforming it to either (x, x+y) or (x+y, y).
Given a starting point (sx, sy) and a target point (tx, ty), return True if and only if a sequence of moves exists to transform the point (sx, sy) to (tx, ty). Otherwise, return False.
Examples:
Input: sx = 1, sy = 1, tx = 3, ty = 5
Output: True
Explanation:
One series of moves that transforms the starting point to the target is:
(1, 1) -> (1, 2)
(1, 2) -> (3, 2)
(3, 2) -> (3, 5)

Input: sx = 1, sy = 1, tx = 2, ty = 2
Output: False

Input: sx = 1, sy = 1, tx = 1, ty = 1
Output: True

Note:
- *sx, sy, tx, ty will all be integers in the range [1, 10^9].*

Approach Description

A recursive solution for this problem is intuitive.
In the case where tx < sx or ty < sy, we return false
As we can only increase sx or sy to tx and ty and not decrease them, so no transformation is possible.
As a base cases we take
1. When condition where both sx == tx and sy == ty, we return true
2. When sx == tx, we return (ty - sy) % tx == 0
3. When sy == ty, we return (tx - sx) % ty == 0

Is sx = tx, then ty must be of the form sy + n*tx
Similarly when sx = tx, then tx must have been of the form sx + n*ty

If tx > ty
Then we call the function again on (sx, sy) and (tx % ty, ty)
Else If tx < ty
Then we call the function again on (sx, sy) and (tx, ty % tx)

Consider making only (x, y) -> (x, y +x) transformations,
(sx, sy) -> (sx, sy + n * sx)
Consider making only (x, y) -> (x + y, y) transformations,
(sx, sy) -> (sx + n * sy, sy)
Where n is the number of transformations made, these observations help us make our base cases.

Run Time Analysis

Time Complexity - O(max(tx, ty))
Space Complexity - O(max(tx, ty)) / O(1)
depending on recursive (due to stack space) or iterative solution.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Reaching%20Points/reaching_points.py

Test Cases

190 / 190 test cases passed.
Status: Accepted
Runtime: **24 ms**
Memory Usage: **12.6 MB**
Submitted: **1 hour, 33 minutes ago**

https://leetcode.com/submissions/detail/285583667/

Problem 30

You Can Go Your Own Way

https://codingcompetitions.withgoogle.com/codejam/round/0000000000051705/00000000000881da

Problem Description

You have just entered the world's easiest maze. You start in the northwest cell of an N by N grid of unit cells, and you must reach the southeast cell. You have only two types of moves available: a unit move to the east, and a unit move to the south. You can move into any cell, but you may not make a move that would cause you to leave the grid.
You are excited to be the first in the world to solve the maze, but then you see footprints. Your rival, Labyrinth Lydia, has already solved the maze before you, using the same rules described above!
As an original thinker, you do not want to reuse any of Lydia's moves. Specifically, if her path includes a unit move from some cell A to some adjacent cell B, your path cannot also include a move from A to B. (However, in that case, it is OK for your path to visit A or visit B, as long as you do not go from A to B.) Please find such a path.
In the following picture, Lydia's path is indicated in blue, and one possible valid path for you is indicated in orange:

Input

The first line of the input gives the number of test cases, T. T test cases follow; each case consists of two lines. The first line contains one integer N, giving the dimensions of the maze, as described above. The second line contains a string P of 2N - 2 characters, each of which is either uppercase E (for east) or uppercase S (for south), representing Lydia's valid path through the maze.

Output

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is a string of 2N - 2 characters each of which is either uppercase E (for east) or uppercase S (for south), representing your valid path through the maze that does not conflict with Lydia's path, as described above. It is guaranteed that at least one answer exists.

Limits

1 ≤ T ≤ 100.
Time limit: 15 seconds per test set.
Memory limit: 1GB.
P contains exactly N - 1 E characters and exactly N - 1 S characters.

Approach Description

The problem asks us to find a path in the given maze which does not overlap with the with an already traversed path.

A simple brute force solution would be to just list out all the possible paths to reach from the northwest cell to the southeast cell. As soon as we find a way that is not the same as the path taken by Lydia, we stop and return the path.
While this is sufficient to pass the first test case, it times out on the other 2.

We notice, that we need to take N-1 steps southward and N-1 steps eastward, making a total of 2N-2 steps. The order in which we take the steps does not matter at all.
A simple idea is that, since we already know Lydia's moves, we invert and follow them!

We maintain a dictionary to invert the given moves, East to South and South to East.
We try and understand why this would not reuse any of Lydia's moves.
At any time, the next move made by Lydia at any cell [x][y] would be the (x + y + 1)th term in her moves. Since our move is always inverted, our move at the (x + y + 1)th index is guaranteed to put us in a different (inverted) path.

Another approach that works is, where we consider all the cells in the maze as nodes on a graph and define successors as per our given moves. We then traverse the graph using algorithms like Depth-first search or Breadth first search to find the path from the northwest cell to the southeast cell. We only omit traveling along an edge if that was a path taken by Lydia.
This would take us $\mathcal{O}(n^2)$ time where 2N - 2 is the number of moves needed to be made to reach the goal (basically N is the order of the given square matrix).

Our most optimal solution takes only O(n) time,
Going through all the moves that Lydia makes and simply inverting them.

Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(n)
Where n is the order of the square matrix/maze.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/You%20Can%20Go%20Your%20Own%20Way/own_way.py

Test Cases

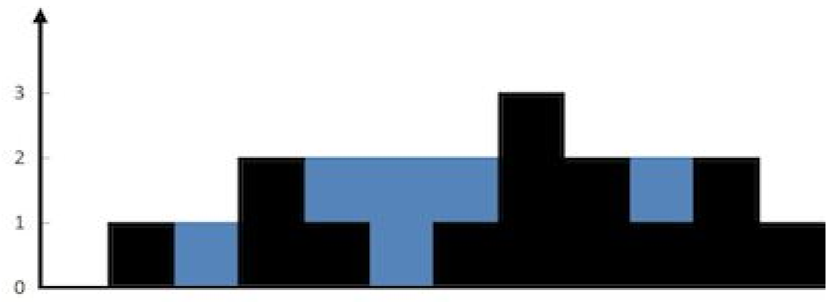
Attempt 3 ✓ ✓ ✓ Dec 12 2019, 16:59

Problem 31

Trapping Rain Water
https://leetcode.com/problems/trapping-rain-water/

Problem Description

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rainwater (blue section) are being trapped. Thanks, Marcos for contributing this image!
Example:
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6

Approach Description

A simple solution to the problem would be to find the for every element in the array, the maximum level of water it can trap after it rains. The water that it can trap is the minimum(leftMaxHeight, rightMaxHeight) - its own height.
The psuedo code would be as:

```
Rain = 0
For every height in the array:
        # these variables hold the left and right maximum at each index/height
        Lmax, rmax = 0, 0
        Iterate from the start upto current height to find lmax
        Iterate from the current height till the end of the array to find rmax
        ans = ans + min(lmax, rmax) - height
```

This takes $\mathcal{O}(n^2)$ time.

An optimization to this approach would be that we could store the leftMaxHeight and the rightMaxHeights for every index in an array and then simply loop over the heights to find the total amount of water trapped.
The pseudo code would be as,

```
For i in range(1, len(heights)):
    Lmax[i] = max(height[i], lmax[i - 1])
    Rmax[~i] = max(height[~i], lmax[~(i - 1)])
    # ~i = n - i - 1, where n is the length of the list.

For i in range(1, len(heights)):
    Ans = ans + min(lmax[i], rmax[i]) - heights[i]
```

This asymptotically takes us only O(n) time.

We observe that at any point in the man, the amount of rainwater trapped by it depends on the minimum of lmax and rmax.
So in the above approach, as long as lmax[i] < rmax[i] the rain amount depends on lmax[i]
And similarly for rmax as well.
Thus, as we can see that as long as there is a larger height at the other end, we can keep traversing in our current direction, as soon as we find a height greater, we start traversing from the reverse direction and so on till both our left and right pointers meet.
So we now need only 2 pointers and can compute the result in a single pass
The pseudo code is as follows,

```
Ans = 0
Left, right = 0, n - 1     # where n is the size of the elevation map
lmax , rmax = -1, -1
While left < right:
    If height[left] < height[right]:
        If height[left] >= lMax:
            lMax = height[left]
        Else:
            Ans += lmax - height[left]
        Left += 1
    Else:
        If height[right] >= rMax:
            rMax = height[right]
        Else:
            Ans += rmax - height[right]
        right -= 1
```

Notice at each step we the one pointers left and right move toward the other. We are guaranteed to take at most n steps where n is the size of the elevation map.

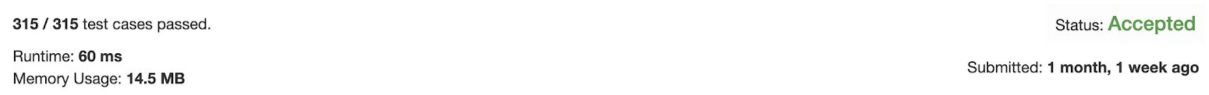
Run Time Analysis

Time Complexity - O(n)
Space Complexity - O(1)
Where n is the size of the elevation map.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Trapping%20Rain%20Water/trapping_rain.py

Test Cases



315 / 315 test cases passed.

Runtime: **60 ms**
Memory Usage: **14.5 MB**

Status: Accepted

Submitted: **1 month, 1 week ago**

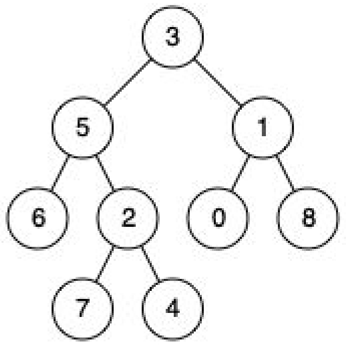
https://leetcode.com/submissions/detail/275193294/

Problem 32

Lowest Common Ancestor of a Binary Tree
https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

Problem Description

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself)."
Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]



Example 1:
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
Example 2:
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

Note:

- *All of the nodes' values will be unique.*
- *p and q are different and both values will exist in the binary tree.*

Approach Description

Given two node in a binary tree, we need to be able to find the lowest common ancestor of the two nodes.
A recursive solution is intuitive to think of,
We travel the tree in a depth-first manner and as soon as we encounter either p or q in our path we return a boolean flag (or even just 0 or 1). Using these flags, we know if we encountered either p or q in our path from the current node. The lowest common ancestor would be one where both the branches return true, or is either p or q and the corresponding branch returns true.
The pseudo-code for the algorithm is as follows,
We start traversing the tree in a depth-first manner from the root of the tree.
If this node is either p or q, we set flag a to 1.
We do the same on the left and the right halves of the tree and collect the results.
If at any point any two of, (the flag a, and the answers of the left, right branch) are true this means we have found the lowest common ancestor of the given nodes p and q.

Another approach that is fairly straight forward to think of is,
We can get the parent pointers of p and q by traversing the tree from the root to p and q.
Now that we have the parent pointers, we can traverse back from p and q,
Where ever this path first intersects/meets that node is the lowest common ancestor of p and q.

The pseudo-code for the above approach would be as,
We start traversing the tree in a depth-first manner from the root of the tree until we find both p and q.
We record the parents of each node as we traverse in a parents dictionary.
Now we travel backward from p along the parent pointers and store the path in an ancestors set till we reach the root.
Next we start traversing back from q as long as we don’t encounter a node in its path that is also in the ancestors set. As soon as we do encounter this node (which we are guaranteed to, atleast at the root), we return it, since it is the lowest common ancestor of p and q.

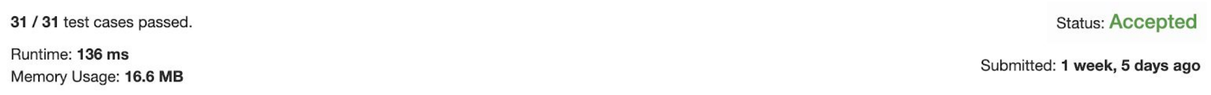
Run Time Analysis

Time Complexity - O(N)
Space Complexity - O(N)
Where N is the is the number of nodes in the binary tree.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Lowest%20Common%20Ancestor/lowest_common_ancestor.py

Test Cases



31 / 31 test cases passed.

Status: Accepted

Runtime: **136 ms**
Memory Usage: **16.6 MB**

Submitted: **1 week, 5 days ago**

https://leetcode.com/submissions/detail/282762160/

Problem 33

Merge k Sorted Lists

https://leetcode.com/problems/merge-k-sorted-lists/

Problem Description

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.
Example:
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6

Approach Description

It is easy to see a brute force solution,
Where we simply traverse all the linked lists and collect all their values in a list.
We the sort this list and create a new linked list out of it.
This would take us O(Nlog(N)) time and O(N) space where N is the total number of nodes in all the lists.

A simple optimization to the above is that we compare k nodes which are the current heads of the linked lists and add the minimum of those to our merged list, we can find the minimum in O(k) time where k is the number of linked lists given.
This takes us O(Nk) time where N is the total number of nodes in all the lists and k is the number of linked lists given. This could also be done by performing k-1 merges between 2 lists.

In the above, it is easy to see that we can improve the time taken to find the minimum each time by simply putting all the k nodes in a priority queue and then popping the queue as needed. This reduces our time complexity to O(Nlog(k)) where N is the total number of nodes in all the lists and k is the number of linked lists given.

An even more elegant solution to the problem comes from the basic solution of being able to merge 2 sorted lists. We already know how to merge 2 sorted lists!
At each step, we pair up 2 lists at interval distance apart.
For the first time interval = 1, so we pair up k lists and merge every pair.

Now we are left with only k/2 lists.
We repeat the process by doubling the interval
And pair up the the k/2 lists and merge each pair to be left with k/4 lists only and so on.
It is easy to see that since only half the lists are left each time, this would take only log(k) time,
At each step, we compare all the nodes of the linked lists to be merged so we do O(n) work at each step.
This makes the overall complexity as O(Nlog(k)) where N is the total number of nodes in all the lists and k is the number of linked lists given.

Run Time Analysis

Time Complexity - O(Nlog(k))
Space Complexity - O(1)
Where n is the size of the elevation map.

Github

https://github.com/MukundaJ/APS/blob/master/Problems/Merge%20k%20Sorted%20Lists/merge_k_sorted.py

Test Cases

131 / 131 test cases passed.
Runtime: **136 ms**
Memory Usage: **17.2 MB**

Status: Accepted
Submitted: **1 month, 1 week ago**

https://leetcode.com/submissions/detail/276069480/

Appendix

Table 1 below details the time we spent on task.

Table 1

Problem	Individual Brainstorming(hrs)	Group Discussion (hrs)	Implementation (hrs)	Report (hrs)	Total (hrs)
Selecting Problems	4	4	0	0	8
Project Compilation	1	1	1	1	4
Bigger is Greater	2	0	2	0.75	4.75
Chef and Bipartite Graphs	2	0	2	0.5	4.5
Forming a Magic Square	3	0.25	1	0.5	4.75
Number of Factors	2	0	1	0.5	3.5
Trouble Sort	2	0	2	0.75	4.75
String Similarity	3	0	2	0.5	5.5
Delivery Man	2	0	2	0.5	4.5
Array Transform	2	0	1	0.5	3.5
Bead Ornaments	5	0.5	0.5	0.5	6.5
Alien Rhyme	2	0.5	4	0.75	7.25
Marbles	1	0	1	0.25	2.25
Best Time to Buy and Sell Stock IV	2	0	1	0.75	3.75
Super Six Substrings	3	0.25	2	0.5	5.75
Robot Programming Strategy	1	0.25	1	0.5	2.75
The Coin Change Problem	1	0	2	0.5	3.5
Matrix Layer Rotation	2	0	1.5	0.5	4
Bytelandian Gold Coin	2	0	2	0.5	4.5
The Next Palindrome	2	0	1	0	3
Cryptopanagrams	2	0	2	0.5	4.5
Yet Another KMP Problem	3	0.25	1.5	0.5	5.25
Foregone Solutions	1	0	2	0.5	3.5
Ordering the Soldiers	1	0	1	0.5	2.5
Domino Solitaire	2	0.25	1	0.75	4

Password Cracker	4	0.5	2	0.5	7
Waffle Choppers	3	0	1	0.5	4.5
Chief Hopper	1	0	2	0.5	3.5
Vogon Zoo	2	0	2	0.5	4.5
Cultural Programme	1	0	1	0.5	2.5
Reaching Points	2	0	2	0.5	4.5
You Can Go Your Own Way	3	0.25	1	0.5	4.75
Trapping Rain water	2	0	2	0.5	4.5
Lowest Common Ancestor of a Binary Tree	1	0	1	0.5	2.5
Merge K Sorted Lists.	2	0	1	0.5	3.5
Total	**74**	**8**	**52.5**	**18**	**152.5**

Table 2 lists all our group members.

Table 2

Name	**Email**
Mukunda Jajoo	muj4340@rit.edu
Nikhil Pandey	np7803@rit.edu
Abhiram Bharadwaj	ab8136@rit.edu
Manasi Gund	mg9546@rit.edu
Priyanshu Srivastava	ps6808@rit.edu

All (30/30) of the chosen questions were successfully solved, passing all the test-cases. Further, we include 3 more interesting questions we solved (which we solved due to the poor availability of some of the chosen problems on the platform at times Ex- Problem 7).

In total, about 153 hours were devoted to this Independent Study.