Intra KUET Programming Contest - 2020
Editorial

A. Piper's Sugar and Spice

Problem setter: Pias Roy
Alternate writer: Rudra Das

Prerequisites: Basic Geometry

This is a very tough problem. We couldn’t solve it.

B. Ticket Counter

Problem setter: S M Mohaiminul Islam Rafi
Alternate writer: Sourav Chakraborty

Prerequisites: Patience :p

This is a simulation problem. The most annoying part of this problem is the problem statement itself. We have two counters queues, separate for males and females. We can easily understand that it’s better to take customers on the first-come, first-serve principle. So we will maintain two queues sorted on the basis of arrival times. We can handle **U** in any convenient way like pushing **U** as a customer to its respective queue and check one the basis of arrival time if its **U** currently being processed. While creating the queues, we will also want to keep track of the type of ticket for each customer.

Now we can maintain two variables, one for tracking the current time and other for counting consecutive male customers. We will check the customer at the front of each queue and take the customer from a queue according to the rules, discard that customer, and set the current time to finish the second of his/her booking. So if the current customer is holding an AC ticket and the current time is T, then after serving that customer current time will be T+X. Here are the rules we should keep in mind.

- 1 female customer after every 2 consecutive male customers. The word consecutive is important. If both queues are empty between two male customers, they are still considered as consecutive.
- If it’s the turn of any queue which is empty, then the other queue is served.

Time Complexity: O($n\log_2 n$) for sorting, O(n) for iteration over the customers. Overall time complexity ~~ O($n\log_2 n$)

C. Show Off!

Problem setter: Muhiminul Islam Osim
Alternate writer: Md. Rifat Haider Chowdhury

Prerequisites: Constructive Algorithm, Implementation

At first, as you cannot make a_1 or a_n the highest pillar after the arrangement, it's a nice observation that you don't need to increase the height of a_1 or a_n in any time. So, a_2 needs to be $\max(a_1+1,a_2)$ to satisfy the 1^{st} property. Similarly, a_{n-1} needs to be $\max(a_n+1,a_{n-1})$. So, you can calculate $lcost_i$ as the cost needed to make a_1, a_2, …, a_i strictly increasing and same goes for $rcost_i$ to make a_i, a_{i+1}, …, a_n strictly decreasing. So, we check for each index i to be the highest pillar h after the arrangement if possible. The cost for an index i to satisfy the properties is $lcost_{i-1}$ + $rcost_{i+1}$ + (h - a_i) if and only if $a_i \leq h$, $lcost_{i-1} < h$ and $rcost_{i+1} < h$. You just check all the possibilities and take the minimum.

Time Complexity: O(n)

D. Batman Needs Your Help!

Problem setter: Al Noman
Alternate writer: Md. Rifat Haider Chowdhury, Toufique Imam Nuhash

Prerequisites: Hashing, Two Pointers, KMP

In order to create maximum length palindrome from given conditions, it is clear to see that possibilities are:

1. Say maximum length is x such that A.substring(0,x) == B.substring(m-x, m) , say string T . We can take a prefix palindrome from the rest of String A , say T_a, and take a suffix palindrome from the rest of String B, say T_b. We can then make our first two palindromes ans_1 = T + T_a + T and ans_2 = T + T_b + T
2. Another ans can be maximum length prefix palindrome from string A , say ans_3

3. Another ans can be maximum length suffix palindrome from string B , say ans_4

We take the best ans from ans_1, ans_2, ans_3 and ans_4 and print it.
For **the maximum length prefix palindrome** we can use **Hash / KMP** , or other methods that can compute maximum prefix palindrome in O(n).

Time Complexity: O(n)

E. Going Round in Circles

Problem setter: Redwanul Haque Sourav
Alternate writer: Tanvir Wahid, Rudra Das

Prerequisites: Observation, BitMask, Data Structures

Let’s take a binary string: 1101001. After the first cyclic left shift, it becomes: 1010011 After the second cyclic left shift, it becomes: 0100111 = 100111 After the third cyclic left shift, it becomes: 001111 = 1111 After the fourth cyclic left shift, it becomes: 1111

There’s no change after the third cyclic left shift. In fact, no matter how much cyclic left shift occurs here, the integer will not change.

It is easy to see after at most 32 cyclic left shifts an integer won’t change anymore.

So we can keep a segment tree on the numbers, with an extra flag in each node. This flag determines if all the numbers in the array in corresponding range has converged(Meaning it will not change anymore like 1111)

While updating we will descend from a node in segment tree if all the numbers have not converged in the range.

Total complexity: O(maximum bits in A_i in binary representation$\cdot n \cdot \log_2 n$)

F. Help Chokro!!!

Problem setter: Rudra Das
Alternate writer: Tanvir Wahid, Shaidul Mursalin Yead, Pias Roy

Prerequisites: Ternary Search, Constructive Algorithm

In this trouble you have to convert the array such that $A_1 = A_2 = \ldots = A_n = X$.Where X % k = 0.
So, X = d * K (where d is an integer value). But it can be easily observed that for the optimal answer, you need to just consider d in range $(\min(A_i)/k, (\max(A_i)+k-1)/k)$.
So, you can perform a ternary search in this range to find the optimal answer.
There are also some alternative ideas to solve this problem.

Time Complexity: $O(n\log_3 n)$

G. Short Statement Here

Problem setter: Tanvir Wahid

Prerequisites: Inclusion exclusion/mobius function, Sieve, Stack

Let's try to solve this problem using stack. Sort the array in non increasing order. Idea is, we'll run a loop from 1 to n and keep including current elements on the top of our stack. Before including it to the stack we do some operation. Let's say elements of our stack is as follows: (from top to bottom)
$s_1, s_2, s_3, s_4, s_5 \ldots$. ($s_1 \leq s_2 \leq s_3 \leq \ldots$ our array is sorted in non increasing order. So this condition will be true.)
Let's say our current element is a_i and s_3 is coprime to a_i. Then s_1, s_2 can't produce a better answer. Let's see an example. If elements of our stack is 8, 12, 13, 16 and our current element is 4 then 8 and 12 can't produce better answer than 4 * 13. So we can remove them from stack and we don't have to worry about them anymore.
Our solution is to for every a_i we pop the top element from the stack as long as we have integers in the stack which are coprime to a_i. Then push a_i to the stack. So, the next challenge is to know if there is a coprime number to a_i in

the stack. If there is, then how many. It can be done using inclusion exclusion/ mobius function.
We need to pre-calculate divisors of every integer from 1 to 10^6 using the idea of sieve technique and vector array. We also need a counter array. When we push a_i in the stack we increase count of all its divisors by 1. When we pop a_i from the stack we will decrease count of all its divisors by 1. Let's say, $a_i=p_1^{x1} * p_2^{x2} * p_3^{x3}$ where p_i are primes and x_i are their powers. So count of stack elements which are coprime to a_i = Size of stack - count(p_1) - count(p_2) - count(p_3) + count($p_1 * p_2$) + count($p_1 * p_3$) + count($p_2 * p_3$) - count($p_1 * p_2 * p_3$). In other word, $\sum$count((d_i)) * mobius(d_i) where d_i is divisor of a_i. If f(a_i) is the number of divisors of a_i then complexity is O($\sum f(a_i)$) . If you work with only unique numbers then complexity will be O(n ln n).

Time Complexity: O($\sum f(a_i)$) or O(n ln n)

H. Mr. Rifat Vs Evil Rudra

Problem setter: Sourav Chakraborty
Alternate writer: Muhiminul Islam Osim

Prerequisites: Data Structures, Hashing

If two points can not reach each other, they must be in different rectangles. Consider a Hash value to each rectangle and use a 2D bit or 2D segment tree to do the update and point query. If the two point query return the same value print yes otherwise no.

Time Complexity: O($q \cdot \log_2 n \cdot \log_2 m$)

I. Birthday Gift

Problem setter: Shaidul Mursalin Yead
Alternate writer: Sourav Chakraborty, Muhiminul Islam Osim, Rudra Das

Prerequisites: Binary Search, Dijkstra

This problem can be solved with binary search and dijkstra. First do a binary search over ***K*** (balloons) , for every mid-value of binary search ,only takes edges which ***K*** is greater than or equal to mid-value, then run Dijkstra with this edges, if it is possible to reach city ***n*** from city ***1*** with less than or equal to cost ***C,*** update the ***answer*** and lower bound of binary search with ***mid+1*** , else update upper bound with ***mid-1*** . Initially the answer is **0**.

Time Complexity: $O((N+M)\cdot \log_2 M \cdot \log_2 K)$

J. Minimum Time Travel

Problem setter: Toufique Imam Nuhash
Alternate writer: Rezaul Hoque, Al Noman

Prerequisites: Basic Geometry

It is easy to see that the minimum positive polygon area will be created by 3 points , triangle . As the number of points <=100 , we can go through all possible ways to create a triangle and find the minimum positive polygon area.

Time Complexity: $O(n^3)$.

K. Let There Be Equity

Problem setter: Shafiq Newaj Shovo
Alternate writer: Rezaul Hoque, Rudra Das

Prerequisites: Sorting

Here always the person with the present maximum perviledge value will always choose the present minimum free resource to maintain the Rule of equity.
So, this problem can be solved just using sorting.

Time Complexity: $O(n\log_2 n)$

L. Spoiler Alert

Problem setter: Rezaul Hoque
Alternate writer: Sourav Chakraborty

Prerequisites: Math, Pattern

After observing some small cases, you will find a series of 1, 2, 4, 6, 9, 12....
For any given N, the ans is $((N+1)^2)/4$.

Time complexity: O(1) for each test case

M. OverloadedOsim

Problem setter: Md.Rifat Haider Chowdhury
Alternate writer: Muhiminul Islam Osim

Prerequisites: Dynamic Programming, Math

First of all, we don't need to remove elements from the array, we just split the array in different groups.

One Observation is all newly added value in the array,

1. for all odd indexes, every newly added value will be the same.
2. for all even indexes, every newly added value will be the same.

For dynamic programming approach, our state will be,

- dp(idx, oddvalue, evenvalue)
  - Here, idx represents the current starting index of the array.
  - oddvalue represents the total newly added value to the odd indexes.
  - evenvalue represents the total newly added value to the even index.

- Now, we have our starting index (idx) in our parameter.So, we can create a new group from this index. For this, we can iterate through

for the last index for this group, and find the maximum in dp function.
if i is our last index,

- let Sum be the summation of the array from index idx to index i.
- Now update Sum by adding the proper amount of oddvalue and evenvalue. Hope that you can do it by yourself.
- let K = Sum % X
- Calculate the Cost adding the proper amount of K and-K. Hope that you can do it by yourself.
- Now the next call will be , dp(i + 1, newoddvalue, newevenvalue).
- if idx is odd then, newoddvalue = oddvalue + K and newevenvalue = evenvalue - K
- if idx is even then, newoddvalue = oddvalue - K and newoddvalue = oddvalue + K
- Now get the maximum of all, Cost + dp(i + 1, newodovalue, newevenvalue)

Now, another observation is oddvalue and evenvalue will be opposite of each other. like, oddvalue = -evenvalue. so we can get rid of one parameter.

Time Complexity: $O(n^2x)$