Problem Set Analysis

ICPC Dhaka Regional Preliminary Contest, 2021 Hosted by: BUBT

Problem A: Omicron Juice

Author: Md Mahbubul Hasan
Contributor: Nafis Sadique, Sabit anwar Zahin
Analysis: This is the easiest problem of the set. We just need to check if the sum of A, B and C is divisible by 3 or not.
Comment: We always face one dilemma when setting a giveaway. In the preliminary there are lots of new teams, so we want some easy problems (not necessarily only one) so that they can keep engaged for a significant amount of time during the competition. On the other hand it causes the queue size to increase significantly since there are lots of submissions in a very small amount of time. We also think that maybe the give away should be a bit difficult so that submission time gets a little bit distributed not causing a submission spike. But on the other hand a little bit difficult for average might mean very difficult for inexperienced teams. But this time the issue was not this… see problem B.

Problem B: Omicron Juice Again

Author: Md Mahbubul Hasan
Contributor: Nafis Sadique, Sabit Anwar Zahin,
Analysis: First of all, we can move the juice by K amount. So the value of A%K, B%K, C%K will never change and they must be the same to have Peaceful morning. Let's subtract A%K from A (and similarly for B and C). Now we can consider the "K amount" as a unit amount. We can add/subtract K from A, B or C like the Problem A. Lets divide A by K (similarly for B and C). Now the problem is completely reduced to Problem A. Just check whether their sum is divisible by 3.

Comment: (Continuing from Problem A) However, for this problem the issue was a bit different. It had 200,000 lines of input, and many contestants submitted TLE solutions. Some resubmitted their TLE solutions while they did not receive the verdict from the initial submission. Also we suspect using endl causes some issue in Codemarshal. All these yielded a huge submission queue. Initially we were thinking that it's similar to the previous long queue issues, but the TLE scenario made it more complicated and frustrating. After discussing a while and considering many different options/scenarios we decided to reduce the data set and reduce the time limit. This caused some side effects like rejudging a huge amount of solutions making the CM slow for a moment but the queue cleared quite fast and we had a better contest experience.

Problem C: Bitonic Beaver

Author: Pritom Kundu
Contributor: Rafid Bin Mostofa, Md. Nafis Sadique
Analysis: A sequence is bitonic if and only if its smallest element is at the start or the end and the sequence stays bitonic if the smallest element is removed. Or in other way, you can pick the elements in zig-zag fashion, either from the front or back, how many ways you can do that.

It means all the sequences must have a common element at one of the ends of the array. So let's consider three cases.

Case 1: Let's create a set of "the end of the array numbers" for all the given arrays. If they do not have a common element then the answer is 0.

Case 2: Similarly if those sets have exactly one element in common then we can remove them and call the counting subroutine for the remaining of the arrays.

Case 3: Since the number of elements of those sets is at most two, the remaining case is when there are exactly two common elements across all the sets. This is a difficult case. First let's handle the easy subcase, when the remaining arrays are all equal. In such a case the answer is 2^(length of the arrays). Since each time we can take the smallest element from one end or the other. If all the arrays are not equal, let's first reverse some of them if necessary to make sure the first element of all the arrays are the same. Now check the length of the common prefix and length of the common suffix. Say they are P, S respectively. We can say that the resulting count is equal to ncr(P + S, P) times the answer for the remaining portion of the arrays. Why? Suppose you start taking elements from only prefix. After taking P elements, you are stuck. Unless you take S elements from the suffix you can not take any element from the prefix end. Similarly unless you take P elements from the prefix you can not take any element from the suffix end. That means, you first need to take P and S elements from the ends, only then you can approach the remaining arrays. Counting how many ways you can take P and S elements from the end is an easy counting problem and this number is: ncr(P + S, P).

Problem D: Lazy Squirrel

Author: S. M. Shaheen Sha
Contributor: Hasnain Heickal, Pritom Kundu, Md. Muhiminul Islam Osim
Analysis: This problem can be solved by DFS and Hashing. The main challenge of this problem is finding the nodes in which the string that we got by concatenating the letters from root to that node is palindrome. For this we can run DFS from root. We maintain two hash values, one for the string obtained from root to that node(Hash_1), and another one for the reversed string(Hash_2). Suppose a node is in now L'th level, V is the character on that node, then new hash values will be,

Hash_1 = Hash_1 + Base^(L-1) * V
Hash_2 = Hash_2 * Base + V

If these two values are equal then it's a palindrome. Rest of the part is to do simple math.

P = Number of nodes in which Squirrel will be happy
Q = Number of Node in that tree
G = GCD(P, Q)
So, Final result = (P / G) “/” (Q / G)

Problem E: Pairdrome

Author: Aminul Haq
Contributor: Arghya Pal, Sabit Anwar Zahin, Md. Mahamudur Rahaman Sajib, Rafid Bin Mostofa
Analysis: Let sub_a and sub_b be the substrings of the given input X and Y respectively. There can be three cases, |sub_a| < |sub_b| or |sub_a| = |sub_b| or |sub_a| > |sub_b|. (Here |S| denotes the length of any string S.)

Let's discuss for the case where |sub_a| < |sub_b|. If their concatenation is a palindrome, sub_a definitely is a reverse of a suffix of sub_b, and the rest of sub_b is a palindrome. For example, let sub_a = “xy”, sub_b = “zyzyx”, so here the reverse of sub_a (yx) is a suffix of sub_b and the rest of sub_b “zyz” is a palindrome.

So we need to pre-calculate how many palindromic substrings have ended at each position of string Y. This can be done by some palindrome algorithm (Manacher/palindromic tree etc).

After doing this, we have to count the matchings from sub_a with a suffix of sub_b and update our answer, this can be solved with suffix array/automata along with another data structure (depending on the implementation).

The other two cases can be handled in a similar way.

Problem F: A Lonely Queen

Author: Tanzir Pial
Contributor: Rafid Bin Mostofa, Sabit Anwar Zahin, Raihat Zaman Neloy
Analysis: This is a standard Dijkstra problem. But if your state is only the cell you are currently at, then you will get TLE as there can be (n x m) x (n+m) edges. Your state should be your current cell and your current direction. Direction can have 9 values, 8 for 8 different directions and one for indicating you have ended the previous move and can start a new move from the current cell. This way you can either proceed in your previous direction without incurring any further cost or you can go from directional state to “end of previous direction” state. You can also go from “end of previous direction” to a new direction and add the cost of moving towards that direction. Handling the tunnel or the obstacle is a minor detail here.

Problem G: Alice, Bob and Tree

Author: Mohammad Ashraful Islam
Contributor: Md. Muhiminul Islam Osim, S. M. Shaheen Sha

Analysis: Given a tree and a formula $\sum_{i=1}^{N} ((W_{root} - W_i) \times (-1)^{Distance(root, i)})$.

Observation 1: For a particular node i we do NOT need to know the $Distance(root, i)$, we only need to know whether the distance is odd or even.
Observation 2: Let's bicolor the tree. That is, let's color them with white and black color so that adjacent pairs of nodes do not have the same color. After bicoloring, all the nodes with the same color have even distance between them.

So, for a particular node X colored as WHITE, the formula can be rewritten as $((White\ Count \times W_X) - \sum_{i=1}^{White\ Count} W_i) + (\sum_{i=1}^{Black\ Count} W_i - (Black\ Count \times W_X))$.
Vice-versa for BLACK colored node.

Problem H: Update Query Problem

Author: Md Mahamudur Rahaman Sajib
Contributor: Ashiqul Islam, Raihat Zaman Neloy, Nafis Sadique
Analysis: Let's try to solve the problem for a simple case. For example S = P[L1…R1] = "abbaca", T = Q[L2…R2] = "aacbaa", we need to find number of distinct string we can get by concatenating prefix of S and suffix of T.
Let's try to solve this problem for a fixed length L. We need to count how many distinct strings we can create if string length = L.
S = a b b a c a
T = a a c b a a

For example L = 6
empty prefix chosen from S and T[0 : 5] is chosen as suffix from T
S = [] a b b a c a
T = [a a c b a a]
---
a a c b a a

S[0 : 0] chosen from S and T[1 : 5] is chosen as suffix from T
S = [a] b b a c a
T = a [a c b a a]
---
a a c b a a

S[0 : 1] chosen from S and T[2 : 5] is chosen as suffix from T

S = [a b] b a c a
T = a a [c b a a]
---
a b c b a a

S[0 : 2] chosen from S and T[3 : 5] is chosen as suffix from T
S = [a b b] a c a
T = a a c [b a a]
---
a b b b a a

S[0 : 3] chosen from S and T[4 : 5] is chosen as suffix from T
S = [a b b a] c a
T = a a c b [a a]
---
a b b a a a

S[0 : 4] chosen from S and T[5 : 5] is chosen as suffix from T
S = [a b b a c] a
T = a a c b a [a]
---
a b b a c a
S[0 : 5] chosen from S and empty is chosen as suffix from T from T
S = [a b b a c a]
T = a a c b a a []
---
a b b a c a

If we observe carefully we can get that if S[0 : i] is chosen as prefix from S and suffix T[i + 1 : Length(T) - 1] is chosen from T as suffix and if S[i] is not equal to T[i] then we are getting a new string otherwise we are getting a string which previously occurred. So each time we get a match at i't position we are getting a string which has previously occurred.

Let's generalize this solution, if prefix S[0 : i] is taken and T[j + 1 : Length(T) - 1] is taken as suffix then if S[i] = T[j] then we won't count the string as it is previously counted.

So the number of strings which are previously counted is equal to the number of (i, j) pairs such that S[i] = T[j]. We can count that easily by counting the frequency of characters.
freqS[x] = frequency of character x in string S.
freqT[x] = frequency of character x in string T.

number of strings which previously occurred = $\sum_{c='a'}^{'z'} freqS[c] * freqT[c]$

number of distinct strings = $(Length(S) + 1)(Length(T) + 1) - \sum_{c='a'}^{'z'} freqS[c] * freqT[c]$

Now as we have seen, the solution only depends on the frequency of characters. So the update and query part is easy if we can maintain this frequency with segment trees. We will maintain a segment tree for each character from ‘a’ to ‘z’ for both string P and Q. Each segment tree will be built on a binary array. For character C we can build an array such that if C is occuring in an index in P then element on that index will be 1 otherwise 0 (same for Q). Then we will build the segment tree on that binary array and with that segment tree we will try to maintain range sum.

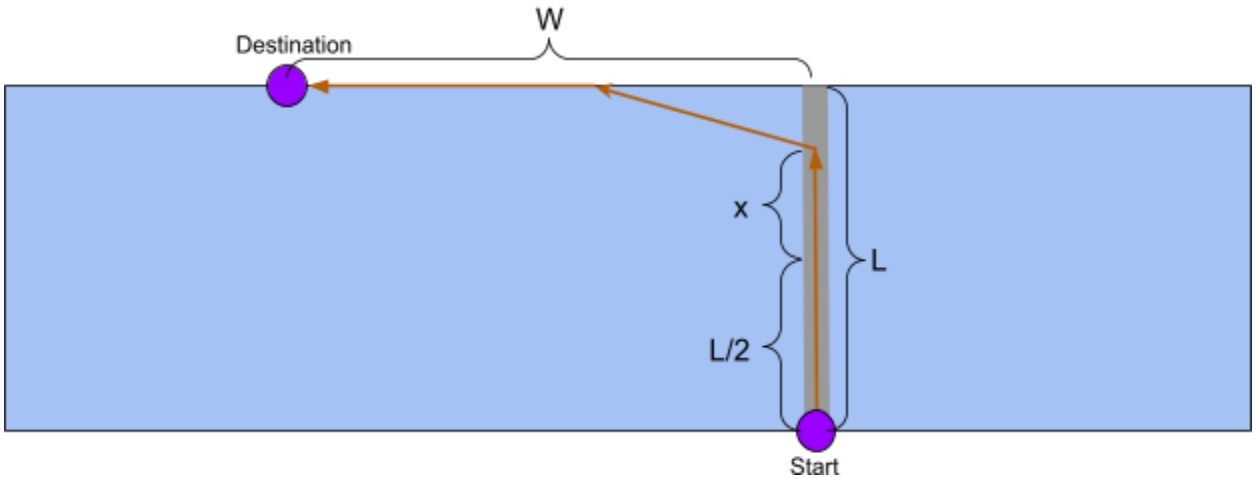
For operation 1:
1 L1 R1 C1
We will update the segment tree of character C1 of string P and assign 1 to all indexes from L1 to R1. For all other segment trees of other characters of string P, we will assign 0. Time complexity of operation 1 is O(26 * log(n)).

For operation 2:
2 L2 R2 C2
We will update the segment tree of character C2 of string Q and assign 1 to all indexes from L2 to R2. For all other segment trees of other characters of string Q, we will assign 0. Time complexity of operation 2 is O(26 * log(n)).

For operation 3:
3 L1 R1 L2 R2
For each character from ‘a’ to ‘z’ we will do a range sum query in the respective segment tree and count frequencies. Time complexity of operation 3 is O(26 * log(n)). So time complexity is O(M * 26 * log(N)).

Problem I: Hovercraft

Author: Shahriar Manzoor
Contributor: Derek Kisman, Rafid Bin Mostofa, Md. Muhiminul Islam Osim
Analysis:



It is obvious that for minimum path length the journey through water must begin after crossing more than half of the bridge. Suppose this journey begins after crossing L/2+x distance. Then

the total path length becomes $L + W - (\sqrt{2Lx} - 2x)$. So now with the help of some basic derivation the value of x can be determined for the minimum value of total distance covered.