Editorial of WPC 1 Session 2013

Praveen Dhinwa

Indian Institute of Technology

praveendhinwa@gmail.com

August 18, 2013

Overview

Now is time to know people behind the scenes of the WPC. Thank you Pankaj Jindal for inspiring me to write an internal contest. Thanks ACA (specifically Harshvardhan Sharama) for providing the judge, PClub (specially Ankush Sachdeva) for providing the prize money.
Problem Setter for the contest is me, Problem testers are Utkarsh Lath, Bhupendra Kastore. Rizwan Hudda has tested all the problem descriptions and his guidelines about difficulty of the contest.
Many many thanks to all of you. It was really a nice experience to work with you guys :)

Problem A

- It was an easy problem. All you had to take care of taking input correctly. You can use gets(stringname) function in C++. You can also use getline (cin, stringname).
- After taking the input, you have to break the input string into tokens by using whitespace as delimeter. That can be done in O(n) time by iterating on the string and keeping track of last valid string with no spaces.
  In the process when you encounter any whitespace theIn if the size of previous string is non zero, then include that string in your set of tokens. Tricky cases are when the start or end of the input string is empty.
- For counting no of different strings, you can use two methods.
  - Use set $<$ string $>$ st;
    Then insert all the strings into the set using st.insert (stringname). As set only stores distinct elements. Use set.size() to find your answer.
  - Sort the strings and iterate over the strings. Keep track of last equal string.

Problem B

- It was the most easiest problem. You just have to observe that you just to have to check really less numbers than floor (n/2) to make them coprime to n. Hence use a simple bruteforce.
- future excercise: Prove the reason for this behaviour.

Problem C

- It was a hard problem. Noone could solve it during the contest.
- Range of answer is not too much. It can be atmost n $\sqrt{n}$. Excercise: prove it :).
- BruteForce for O(n^3) will surely get TLE
  So let us make some observations. First observation: Let us call the point which connects base and height of right angle triangle, special point. Now iterate over each point i and consider the point i to be the special point. Now consider all the points which have x coordinates equal to x[i], Let us say these points are stored in the array pointsEqualtoX[x[i]] and also consider the point which have y coordinates equal to y[i], Let us say these points are stored in the array pointsEqualtoY[y[i]]
- Now for fixing the special vertex, you have to find out how many of these points satisfy the base and height equal property. So if you try to directly search in the entire array pointsEqualtoX[x[i]] or pointsEqualtoY[y[i]]. It will be O(n^2) which will be TLE. Instead you chose by the smaller of pointsEqualtoX[x[i]] and pointsEqualtoY[y[i]],

Problem D

The problem has medium difficulty. You can use dynamic programming with bitmasks. For reading about bitmasks, you can refer to given link. link

dynamic programming idea

Consider dp[mask] denote the maximum area that you can create using all the sticks which are set to 1 in the mask. Now you can try to remove any three valid sticks and do the computation recursively.
Complexity O $(2^n\ n^3)$. Sample code snippets given here. TODO

Greedy Solution

Some solutions used greedy solution by sorting the sticks lengths and keep taking only the highest length sticks whenever it is possible. Tricky case for those was: 4 sticks of length 25 26 24 50

Problem E

hi

The problem has medium difficulty. You can use dynamic programming here also.

dynamic programming for checking palindromes

isPalindrome[i][j] denotes whether s[i, j] (substring of string s from index i to j inclusive) is a palindrome or not.
if s[i, j] has length 1, then it is surely a palindrome.
if it has length, then check if the two characters are equal or not.
In all the remaining cases, last s[i] and s[j] both should be equal for a s[i,j] to be palindromes and for checking s[i + 1, j - 1] is palindrome or not, use recursive computation.
Complexity $O(n^2)$.

dynamic programming for minimum number of palindromes a string can be broken

we can use a similar solution to above solution. let dp[i] = no of min

Problem G

- It was a medium problem.
- You have to identify that problem asks for the $\log_2(totalcost) + 1$. As costs are given in the powers of 2 always. This is a great hint that product of all costs will finally also be power of 2. As we know total number of divisors of 2^n are n + 1.
- Now consider one very important property of log.
  $\log(cost_1 * cost_2 * .. * cost_n) =$
  $\log_2(cost_1) + \log_2(cost_1) + \log_2(cost_2) + .. + \log_2(cost_n)$
  Now replace all the roads of cost having C by log (C).
  Now problem converts into finding Minumum Spanning Tree (MST) of a graph. (it is a tree with minimum edge weight).
  Finding MST can be done by either by Prim's algorithm or by Kruskal's algorithm.
- complexity O(m log(m)).

Problem H

- It was a medium problem.
- problem recudes into 2xy = n So if n is odd, then no of solutions zero. otherwise find solutions of xy = n / 2 obeying the properties given in the problem.
- As x should not be prime, and y can be anything. You have to find such divisors of n which are not prime. You can do it in this way.
- Generate all the factors by recursive algorithms and check whether they are prime or not. Sample solution for generating factors recursively: TODO
- complexity O($\sqrt{n}$).

Problem I

- It was a medium graph theory problem.
- Note the following important property. If you can swap a and b and b and c, then you can also swap a and c. This property is called Hence this problem reduces into finding connected set of such swaps because in all those sets you can swap the numbers easily. For finding connected components, Use dfs or bfs.
- complexity O(n^2 + n + m).

Problem J

- It was a medium string theory problem.
- It has two solutions
  - consider the two strings to be a and b.
    Answer is "YES" iff ab = ba
    otherwise answer is NO.
    Exercise: try to prove yourself. :)
  - Without loss of generality let us assume that length (a) ¡= length (b). Now answer to this problem is YES if a = p^m for some p and m and b = p^n for some p and n. Note that length of smallest p will be length(p) = gcd (length (a), length(b)). and check whether the two strings are of the form of p^m or p^n or not. This step can be done in O(n).
- complexity O(length(a) + length(b)).

Problem L

- It was a hard geometry problem.
- It is a standard problem. For finding minimum distance pairs, you can use divide and conquor algorithm.
  For finding maximum pair distance, you can first find out convex hull of the points. Then use rotating callipers technique to solve the problem. All the things here are standard and a good amount of google search can lead to good resoures. I will udpate the resources later. TODO
- complexity O(n * logn).

Problem M

- It was a hard number theory problem.
- answer to problem is $(-1)^N/2$ where N is number of solutions of $x^2 = 1$ mod n. Tricky test cases : 1 : answer is 0 4 : answer is 3
  Now after thinking for some time, you will realize that answer to problem can be -1 mod n which is same as (n -1) iff n = p^i such that i *gt* 1 and p is odd prime. Proof: Try out yourself :) I will update my proof later.
  Now finding whether a number is p^i is or not ?? First find out whether n is itself prime or not. Then for i = 2, again check if n is of type p^2 or not, if yes then also use primality testing algorithms to test the prime. For i = 3, n is p^3. Now p is in the range of 10^6. You can directly check whether a number is prime or not, even you check with the help of primality testing algorithms. Similarily you can do this for all i such that p^i ¡= n. A fast implementation of this algorithm will pass
- complexity O(c logn). where c is number of iterations in primality testing algorithms

Problem N

- easy
- Answer to problem is 1 if number of black balls is odd. otherwise it is 0.
- Explanation: As always parity of number of black balls remains constant during the removing of the balls.
  Assume there are x white and y black balls.
  if the two balls are of same color, then he would put 1 white ball.
  Now assume that both balls are white, then remaining white balls will be x - 2 + 1, black will be y.
  Now assume that both balls are black, then remaining white balls will be x + 1, black will be y - 2.
  if the two balls are of different color, then he would put 1 black ball.
  Now assume that both balls are black, then remaining white balls will be x - 1, black will be y - 1 + 1 = y
  Note that parity (value mod 2) of y remains constant.
  So if y is odd, it will remain upto the last step.
- complexity O(1)

Problem O

- easy
- If you have to search for numbers of from k, 2k in the given array and keep taking them.
  This greedy solution will give incorrect answer.
  If we think about this greedy process, let us take an example numbers are k, 2k, 4k. Then we can only take (k,2k) or (2k,4k). We can only make only one pair out of it.
  But take the case of k, 2k, 4k, 8k. If you decide to take 2k, 4k then you can make only 1 collections. But the answer in this case is 2, as you can make (k, 2k) and (4k, 8k) collections.
  So if you take numbers always in the sorted order, you would not make any error in such kind of cases.
  So finally algorithm is as follows: answer = 0;
  sort the array a
  set all the numbers to be not-taken.
  for val from 0 to n - 1
  search in the not-taken elements

Thank you all