

# I++ OLPC Editorials

## 1) Pandu and the hostellers:

**Difficulty level:** Cake walk

**Setter:** lakshmi8

**Editorialist:** lakshmi8

**Tester1:** ara\_rossi46

**Tester2:** krishna95

This was the easiest question in the contest. You just have to do what the question says. Read the string. Create a new string with all the consonants removed from the original string. Check whether any anagram of the newly formed string is a palindrome. To check for palindrome, you just need a hash table of size 5 (as there are only 5 vowels) for counting the occurrence of the characters. If the string satisfies the following two conditions, it is a palindrome. Otherwise, it is not.

- i) For an odd length palindrome, only one character can occur odd number of times.
- ii) For an even length palindrome, all the characters must occur even number of times.

Setter Code: <http://ideone.com/imXX28>

Tester1 code: <http://ideone.com/wkE3e0>

Tester2 code: <http://ideone.com/CySHMF>

## 2) BST:

**Difficulty level:** Cake walk

**Setter:** ara\_rossi46

**Editorialist:** ara\_rossi46

**Tester:** lakshmi8

It's very easy to figure out from the problem statement that the biggest element gets added to every element and the smallest element does not add up to any element except for itself. The only extra thing which you need to do is to sort the given array to make your work simple as the array is now ordered from smallest to biggest element. Once this is done, your work is simple.

Ex: 5 3 1 4 2

After array is sorted ...

1 2 3 4 5

1 adds up once to sum, 2 twice, 3 thrice, 4 four times and 5 five times and this is the logic. Since the range of numbers is  $10^5$ , multiplication with integers causes overflow, long int is to be used. There is nothing else special with this problem

Setter code: <http://ideone.com/ySQbhT>

Tester code: <http://ideone.com/lqj5EA>

### 3) Minimum Abizons:

**Difficulty level:** Easy

**Setter:** ara\_rossi46

**Editorialist:** ara\_rossi46

**Tester:** lakshmi8

As you can see from the problem statement, the clear approach to this problem is greedy. All we have to find is whether the current student is a part of an increasing sequence or a decreasing sequence and assign the required cost for that student.


For Ex: 1 2 3 4 3 1

The cost assigned is

1 2 3 4 2 1


As you can see we must assign the least possible cost to every student. Participants themselves have coded better solution than mine.

What I have done is I take sequence by sequence. In the above example, I will process the first 4 elements (increasing sequence) and then the last 3 elements (decreasing sequence) and assign the least possible cost starting from the first element.

Thanks to those participants who have given a better solution than mine. I learnt something new .

Setter code: <http://ideone.com/hvoyVr>

Tester code: <http://ideone.com/N0gcRw>

lakshmi8's comment on this problem: ara\_rossi46 and me learnt a new way to solve this problem. This is the importance of problem setting. When you set a problem, you will get to know easy solutions from those who solve your problem. Thank you people for teaching us .

## 4) The Novel:

**Difficulty level:** Medium

**Setter:** lakshmi8

**Editorialist:** lakshmi8

This problem is also easy for those who know BFS/DFS already. A well written brute force solution is efficient enough to pass as the constraints in the problem are very low. A more efficient solution will be to use bfs 2 times. First bfs calculates the minimum time at which a cell gets caught up in fire. The 2<sup>nd</sup> bfs checks if it is possible to move from Source to Destination with the time information gathered in the 1<sup>st</sup> bfs. If it is possible to reach the destination, the 2<sup>nd</sup> bfs outputs the minimum time to do so. Otherwise, it outputs "IMPOSSIBLE".

The important constraints to be handled are:

- i) We cannot move to a cell which has a trap
- ii) We cannot move to a cell which is already caught up in fire
- iii) We cannot move to a cell which is about to get fired in the same minute as we enter that cell.
- iv) The fire in a cell spreads to the cell to its top, bottom, left and right. (Diagonal spread is not possible)
- v) The fire cannot spread to the exit point but it can spread to the entry point.

Check the code for better understanding.

Setter code: <http://ideone.com/OlryV2>

## 5) Meeting with Zora:

**Difficulty level:** Medium hard

**Setter:** lakshmi8

**Editorialist:** lakshmi8

This problem can be using Dynamic programming.

Let  $f(n, m)$  be the number of arrays of length 'n' with exactly 'm' inversions.

Case 1: '1' is the first number in the array.

The number of such arrays that start with the number '1' is  $f(n - 1, m)$ . This is because, no inversions can be formed with 1 as the starting element (as it is the smallest element). So, we have to use numbers from 2 to n. We can also use 1 to  $n - 1$  instead of 2 to n as we are considering consecutive numbers and it doesn't make a difference.

Case 2: '1' is the second number in the array.

If 1 is the second number, then  $f(n, m) = f(n - 1, m - 1)$  because whichever element is first, it will form an inversion with '1'.

Now it can be seen that there is an underlying pattern.

The pattern will be

$f(n, m) = f(n - 1, m - i)$  where 'i' ranges from 0 to  $n - 1$ .

A direct implementation of the above formula will result in a time complexity of  $O(n * n * m)$  which is too slow.

$O(n * m)$  implementation:

If we use pen and paper to reduce the formula, it becomes

$f(n, m) = f(n, m - 1) + f(n - 1, m) - f(n - 1, m - n)$ .

We can construct an  $n * m$  table to store the results.

Memory optimization: For computing a row in the table, we need the results of the previous row alone, so we need only 2 rows at any time. Refer the solution for the optimized implementation.

Setter code: <http://ideone.com/vz4NTn>

## 6) Help Zora:

**Difficulty level:** Hard

**Setter:** krishna95

**Editorialist:** krishna95

On seeing the question first, the naive solution that will come to our mind is to store the stack operations and to build the stack for each and every query. But the maximum number of queries possible is 1000000 and the maximum size of stack possible is 1000000 so this naive solution won't pass in the given time limit.

So we need to find a way to store the states of the stack after each operation. So that each query will just take linear time to print the contents of the stack. The next solution that will come to our mind is to create  $n$  stacks where the  $i$ th will store the state of the stack after  $i$ th operation. But this approach won't pass for the given memory limit.

So we need to find an approach which stores the state of all the stacks with minimum memory possible. We should avoid storing the same element again and again for example, in the above example if we store the states in 5 stacks, then the element 1 will be stored in all the 5 stacks. This will result in more memory. So we shall use a tree like structure where element repetition can be avoided so we can

store 1 only once and other elements can be added as the child nodes of 1. In that tree push operation denotes creation of new child to the current node and pop operation denotes moving the current pointer from current node to its parent (Here the stack grows downwards and shrinks upwards). So the node should have a pointer which should point to its parent. We should also maintain an array to store the address of top node of the stack after each operation.

Setter code: <http://ideone.com/bIDDC7>