ACM SUMMER CHALLENGE 2020

Editorial- Stacks and queues

Icha Icha tactics

Author: Shivangi

Fun fact: Well its not a fun fact but just a fact that Shivangi might end up marrying Kakashi some day.

This question is based on classic balanced parentheses problem of stacks Tutorial. The balanced parentheses problem is basically checking whether a given parentheses sequence is valid or not. For example “{}”,”{}{}”and ”{{}}” are valid while “{{”,”{{}” and “}}{{” are not valid. What we do for this problem is push “{” when we encounter it in the string and pop when we encounter “}” and the top element of the stack is “{” and the stack is not empty, if stack is empty before performing this operation then the string is not valid. If at last the stack is not empty then the string is not valid.

In this question we have to do the same thing but in this case we have an extra character “1” hence the balanced sequence is “{1}” in place of “{}”. So a bit of extra implementation with the original parentheses problem will be all for this question.

The Ant Man

Author: Jignesh

Fun fact: This question was intentionally kept easy in order to balance difficulty.

The ants entered in a straight line in a particular order, do we have a word for that? “Queue”. So basically we have to maintain two queues namely enter and exit, if any ant is ata position in the exit queue who haven’t appeared in the enter queue before that means it appears in the enter queue later than present position that means it overtook so we have to count that ant in punished ones.

Jake Peralta's Case

Author :Krunal (KR)

Fun Fact: The Problem Setter who made this editorial needed to confirm the right greedy approach before writing it.

Here we can remove only the front letters of the given string and the back letters of the stand by string that means we can replace them to queue and stack respectively for better convenience. We need lexicographically smallest *Ans* string that means we have to pop the front elements and push the new elements greedily.

For example if the current front of the S string is smallest letter in the whole string then we will first pop it from *S* push it in *P* and then again pop it from *P* and push it in *Ans*. But if not that means we have a smaller letter which we can put ahead of the letter we currently are on so we will just push it into stand by *P* stack and move forward.

Once we are done with the input string(queue *S* is empty) that means we have no other option and we have to push all stack *P* elements as it is in *Ans*.