Problem on Stack

Relevel
by Unacademy

Problem - 1 (15 min)

Given a string s of brackets `( { [ ] } )` , you need to determine if the provided string S is valid or not?

Condition for the string to be valid:

1. Each opening bracket should have there corresponding closing bracket
2. Order of the opening and the closing bracket should be maintained

Sample Input-1:

(({}))

Output:

True

Explaination:

For each opening bracket with have there corresponding closing backet

Sample Input-2:

(({)

Output:

False

Explaination:

Closing and opening of the brackets are not matching

Approach:

1. Traversal of the string form left to right
2. On encounter of an open bracket, we will move it to the stack datastructure.
3. On encounter of an top bracket we will check the top most element of the stack, if both the elements are same then we will pop it out of the stack and continue else we will return false
4. Once the iteration is completed then the stack should be empty as each opening bracket will have there corresponding closing bracket. If not then return false else true.

Code:

Output

True

```
function validateString(s) {
          //Stack to store the opening bracket
    const openingBrackets = [];
    //Traverse through each character of the string
    for (let idx = 0; idx < s.length; idx++) {
        //Check if left bracket is encountered
        if (s[idx] === '{' || s[idx] === '(' || s[idx] === '[')
            openingBrackets.push(s[idx]);
        //Else pop the element from the stack after validate the following points
        // 1. check the type of closing bracket
        // 2. check if it is not the last element of the string
        // 3. check if the top most element of the stack is same as the closing bracket
        else if (s[idx] === ')' && openingBrackets.length !== 0 &&
openingBrackets[openingBrackets.length - 1] === '(') {
            openingBrackets.pop();
        } else if (s[idx] === '}' && openingBrackets.length !== 0 &&
openingBrackets[openingBrackets.length - 1] === '{') {
            openingBrackets.pop();
        } else if (s[idx] === ']' && openingBrackets.length !== 0 &&
openingBrackets[openingBrackets.length - 1] === '[') {
            openingBrackets.pop();
```

```
        }
        else {
            return false;
        }
    }
    return openingBrackets.length === 0;
}
console.log(validateString("(()){}"));
```

Problem - 2 (25 min)

In solar system we have asteroids moving at constant speed. We are given an array that contains the integer values. The integer value determines the size of the asteroid which is while the sign of the interger determines the direction. Negative sign indicates it is moving towards left while positive indicates marble moves towards right. You need to find the state of the asteroid after the collision. In collision the asteroids with the higher size destroys the one with the lower size and the collision will happen only when they are moving in the same direction.

Sample Input-1:
[11,16,-20]
Output:
[-20]

Explaination:
11 and 16 will not collide with each other as they are moving in same direction, but on collision of 11 and -20, -20 will remain because of its greater size and on collision of 16 and -20, -20 will remain. Hence the final asteroid will be -20

Sample Input-2:
[5,4,-2]

Output:
[5,4]

Explaination:
-2 on collision with 5 and 4 will get destroyed, while 5 and 4 will never collide as they are moving in the same direction.

Sample Input-2:
[-5,5]
Output:
[-5,5]

Explaination:
-5 and 5 are travelling in opposite direction right from start, so wont collide
Approach

Approach:
We will traverse through each asteroid in the asteroid array and for each positive asteroid we will add it to the stack and when a negative value is encountered we will remove all the asteroids from the stack whose size is less than the abs value of the negative asteroid.
Then simply add it to the result only if the top most value of the stack is not same and is negative. Once the traversing is completed return the resultant stack.

Code:

Output:
[10]

```
function onCollision(asteroids){
  let resultStack = [];
  //Traverse through each asteroid in the array
asteroids.forEach(x=>{
                    //If moving in right direction then store it in stack
        if(x>0)
               resultStack.push(x);

        //If found an asteroid moving in the left direction then check for collision
```

```
        else{
          //If the stack  not empty then keep poping till the time the top most
element is less than the size of the current element

while(resultStack.length!=0 && resultStack[resultStack.length-1]>0
          && resultStack[resultStack.length-1]<Math.abs(x))
                    resultStack.pop();
          //Insert the element in the stack only if the last most element is less than 0
          if (resultStack.length == 0 || resultStack[resultStack.length-1] < 0 )
          resultStack.push(x);
          else if(resultStack[resultStack.length-1] == Math.abs(x))
               resultStack.pop();
        }

      })
      return resultStack;
}

console.log(onCollision([10,2,-5]))
```

Problem 3: (25 min)

Given a string S made up of characters `a` abd `b`. You need to count the minimum deletion needed to make the string balanced.
A string is said to be balanced only if there doesnt exists a pair (i,j) where i<j and s[i]=’b’ and s[j]=’a’

Example Input-1
aababbab
Output:
2

Explaination:
On removing the element at position 3 and 6 then the balanced string will be aabbbb and count 2

Approach:
Transverse throught the list of element in reverse order and for each position where a is placed left of b remove it.

1. Traverse through the element in reverse order
2. If the element is first element then put it in stackand continue
3. Check if the current element is b and the top of the stack is a, if yes then remove the top most element and increment the counter, else push the element in stack

Code:

```
function deletionsCount(s){
    let res = [];
    let count = 0;
    for(let idx=s.length-1;idx>0;idx--){
      if(res.length==0){
          res.push(s.charAt(idx));
      }
      else if(res[res.length-1]=='a' && s.charAt(idx)=='b')
          {
              count+=1;
            res.pop();
        }
      else{
                    res.push(s.charAt(idx));
      }
    }
    return count;
}

console.log(deletionsCount("baaabbba"))
```

Problem-4: (25 min)

Given a string S of numbers and an integer N, you need to find the minimum possible integer that can be formed after removing N digits from the string S

Example Input-1:
S = 10300 N=1
200

Example Input-2:
S = 121198 N=2
1118

Approach:
Traverse through each character of the string S and at each character Find the minimum value character among position of character and the following n+1 character of String S and place it in the resultant string.

CODE:

```
function findMinimumInteger(S,k)
{
    let resultantStack = [];

    // Store the final string in stack
    for (let ch of S)
    {
        while (resultantStack.length>0 && k > 0 &&
        resultantStack[resultantStack.length-1].charCodeAt(0) > ch.charCodeAt(0))
        {
            resultantStack.pop();
            k -= 1;
        }

        if(resultantStack.length > 0 || ch !== '0')
            resultantStack.push(ch);
    }

    // Remove largest value from top of stack
    while(resultantStack.length > 0 && k--)
```

```
        resultantStack.pop();
    if (resultantStack.length == 0)
        return "0";
          //Convert the number in stack to string
          let res_string = ""
    while(resultantStack.length > 0)
         {
             res_string = resultantStack[resultantStack.length - 1]+res_string;
             resultantStack.pop();

         }
    return res_string;
}

// Driver code
let str = "765028321"
let k = 5
console.log(findMinimumInteger(str, k));
```

Problem-5 : (25 min)

A frog likes to jump from one wall to another. But the only condition is that the next wall should be higher than the previous one if it is not, then the frog will stop. The stamina needed by the frog to cover the walls is equavilalent to the xor of the wall height traversed by the frog. Now given the height of the buildings you need to find the maximum stamina needed, provided the frog starts from any building.

Input-1: 1,2,4,9,5
Output: 15

Explaination:
2^4^9 = 15

CODE:

```
function getMaxStamina(n,height){
  let stk=[]
  let compuations = height
  for (let i = n-1;i >-1; i--){
    while (stk.length > 0 && height[stk[stk.length-1]] < height[i])
          stk.pop()
      if (stk.length > 0)
            compuations[i] ^= compuations[stk[stk.length-1]]

      stk.push(i)
  }
  return Math.max(...compuations)

}

let n=5;
let height=[1,2,3,8,6]
console.log(getMaxStamina(n,height))
```

Problem-6 (20 min)

Given two mathematical expressions you have to determine if both of them are same or not?

Note:There will be 26 operands and each of them will be unique in an expression

Input-1:

124*+

Output:

9

Explanation:

2*4 = 8

1+8 = 9

Input-2:

6784/+-

Output:

-3

Explanation:
8/4 = 2
7+2 = 9
6 - 9 = -3

Approach:
Given the postfix expression we will use stack to store the integers and then as an operator is received well pop last two element f
the stack compute the operation and push it back in stack uptill the last value of the expression.

Code:

```
function computeexp(exp)
{
        let stack=[];
        for(let idx=0;idx<exp.length;idx++)
        {
            let ch=exp[idx];

            if(! isNaN( parseInt(ch) ))
            stack.push(ch.charCodeAt(0) - '0'.charCodeAt(0));
```

```
else
{
    let v1 = stack.pop();
    let v2 = stack.pop();

    switch(ch)
    {
        case '-':
        stack.push(v2- v1);
        break;

        case '+':
        stack.push(v2+v1);
        break;

        case '/':
        stack.push(v2/v1);
        break;

        case '*':
```

```
                    stack.push(v2*v1);
                    break;
                }
            }
        }
        return stack.pop();
}

let exp="331*+";
console.log(computeexp(exp));
```

MCQ:

1. What is the value of the postfix expression 4 3 2 1 + – *

a) 0
b) 1
c) 2
d) 3

2. What's the maximum number of parenthesis for the following expression appear on the stack at a given time?
(()(())(()))

a) 1
b) 2
c) 3
d) 4

3. Pushing an element in a stack of size 5 which is already filled will result in?

a) Overflow

b) Underflow

c) Error

d) None of the above.

4. Which datastructure can be used for an implicit implementation of recursion?

a) Stack

b) Queue

c) Linked List

d) Array

5. Convert the following expression to postfix using stack

a+b+c

a)abc++

b)a++bc

c)++abc

d) None of the above

Assignment Question:

1. Given a stack S, you need to reverse the element in the stack using another empty stack S'
   Input: 2,4,5,7,8
   Output: 8,7,5,4,2

1. Given a stack, find the max and minimum value stored inside it

Thank you