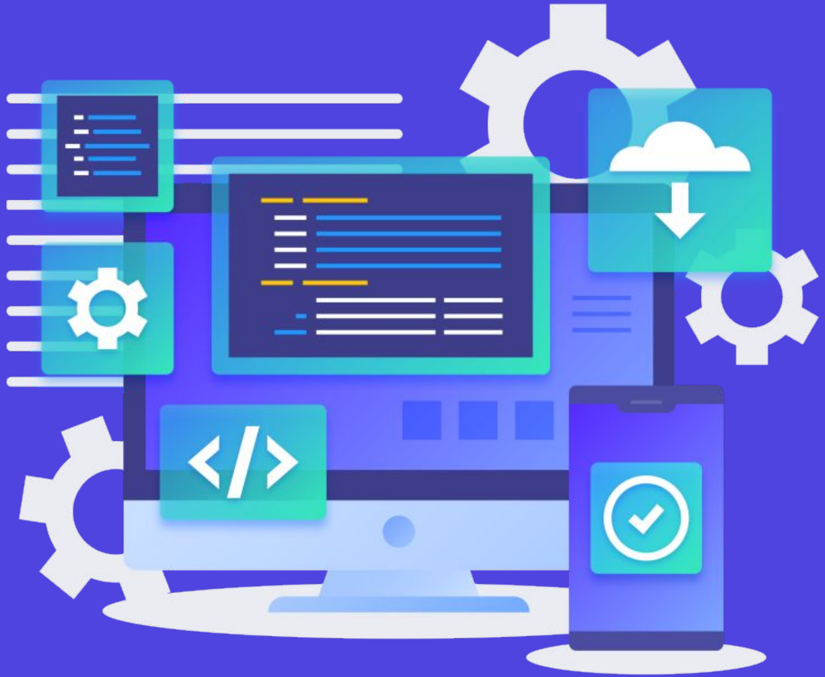
Stack Problems



Relevel
by Unacademy

Problems to be Covered

Problem-1 : K digit removal
Problem-2 : Arrange Students
Problem-3 : Water in storage
Problem-4 : Max Rectangular Plot
Problem-5 : Basic Computator

Problem-1: K digit removal

Given a string S determining a positive number and an integer k, you need to find the minimum possible digit that can be formed after removing k digits from the strings.

Input:
String = “8973132”
k=5

Output:
32

Explaination:
In the given string the minimum number possible after removal of 5 digit is 32.

Expected:
Time Complexity: O(N)
Space Complexity: O(N)

Approach:

We have to remove k digits from the string of size n, so the size of the resultant string would be n-k. So to do so we can use stack as follows:

1. If k>=n then return 0 else if k=0 return the number
2. Iterate throught the String S and push the character to stack if it is greater than the top most element of the stack.
3. Now again iterate over the string S and if the character is lower than the top most element of the stack then pop the ch from stack and decremenet the value of k unless we reach a condition where the condition fails.
4. If k is still more than 0 then pop element from stack k times.
5. Pop remaining element from stack to form an integer.

Code:

```
function getMinNumber(string,k){

    let n = string.length
    let stack = []
    for(let ch of string){
        while (stack.length>0 && k > 0
&& stack[stack.length - 1] > ch){
            stack.pop()
            k -= 1
        }

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


    while (stack.length>0 && k){
        k -= 1
        stack.pop()
    }
    if (stack.length == 0)
        return "0"

    while (stack.length>0){
        string=string.split('')
      string[n-1]=stack[stack.length-1];
      string=string.join('');
      stack.pop()
      n -= 1
    }

    return string.substring(n,)
}


let string = "8973132"
let k = 5
console.log(getMinNumber(string, k))
```

Problem-2: Arrange Students

For a school parade boy and girls have form a line. Boy is represented as B, and Girl is represented as G. The order in which they were supposed to stand was that the girl should stand at one end of line while boys on the other end. But few students has broke this order by standing anywhere in the line. Now your job is to help the teacher to identify minimum number such students who are violating the order and remove them from the line so that post there removal the order is maintained.

Example Input-1
BBGBGGBG

Output:
2

Explaination:
On removing the boys at position 3 and 6 then the order will be BBGGGG. Hence the count of student removed will be 2

Expected:
Time Complexity: O(N)
Space Complexity: O(N)

Approach:
Transverse throught the student list in reverse order and for each position where B is placed left of G remove it.

1. Traverse through the each student in reverse order
2. If the student is first student then put it in stack and continue
3. Check if the current student is G and the top of the stack is B, if yes then remove the top most student and increment the counter, else push the student 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]=='B' && s.charAt(idx)=='G')
         {
              count+=1;
            res.pop();
        }
      else{
                   res.push(s.charAt(idx));
      }
    }
    return count;
}

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

Problem-3: Water in storage

Given the list of integer indicating the height of walls you need to find the max volume of water that can be stored between them whenever it rains.
Note: the width of the wall is 1

Input:
4,0,4

Output:
4

Explaination:
Given two walls of height 4 each, max unit of water that can be stored is 4.

Input:
3,0,1,0,4

Output:
8

Explaination:
We can store 1 & 1 unit of water between wall 1 & 2 , 2&3 respectively. Then we can store 6 units of water between 1 & 3, hence the total unit of water that can be stored is 8.

Approach:
To solve this problem we can use stack to keep track of index of the bar that are bounded by the bigger left and the right bars using only single iteration.

1. Iterate over the the height of each bar
2. While the stack stores has an index of the wall and the current wall height is greater than the wall whose index is stored as the top most element of the stack perform the following
   2.1 Pop the item as last element from the stack and get its corresponding wall height
   2.2 Compute the distance between the current stack top position wall and current wall height
   2.3 Find the minimum height between top pos wall and the current wall
   2.4 Compute the max water that can be trapped i.e. distance*min_height
3. Add it to the stack

Code:

```
function maxStorage(height){
    let stack = []
    let maxUnit = 0
    for(let i=0;i<height.length;i++){
        while(stack.length != 0 && (height[stack[stack.length-1]] < height[i]) ){
            let last_element = height[stack.pop()]
            if(stack.length == 0)
                break
            let distance = i - stack[stack.length - 1] - 1
            let min_height =Math.min(height[stack[stack.length - 1]],height[i])-last_element
            maxUnit += distance * min_height
        }
        stack.push(i)
    }
    return maxUnit
}

let walls = [ 3,0,1,0,4]
console.log(maxStorage(walls))
```

Problem-4: Max Rectangular Plot

Sameer wants to buy a plot in a given locality. The locality is represented in form of a 2D matrix where each cell demonstrates a plot. In the locality if a plot occupied it is represented by 0 else it is marked as 1. Sameer being rich guy wants to buy the biggest possible plot, only condition is that the plot should be a complete rectangle.

Input:

```
0 1 1 0 0
1 1 1 1 1
1 1 1 1 1
1 1 0 0 1
```

Output:

```
10
```

Explaination:

The largest possible plot starts from 0,1 to 5,2 contains total of 10 plots Hence the output is 10.

Expected:

Time Complexity: O(RxC)

Space Complexity: O(C)

Where: R is length of row

And C is length of column

Approach:

We can solve the given problem by creating the histogram for each row and then compute the largest rectangular area of the histogram.

1. We need to iterate over all the rows
2. in all the cases where the row is not the first row, we need to update the row as matrix[i][j] is not equal to zero then matrix[i][j] = matrix[i-1][j]+matrix[i][j]
3. Compute the maximum rectangular area in the histogram by considering the ith row as the histogram heights.
4. Perform the last two steps for all the rows and print the max area for all the rows.

Code:

```
function maxHist(R, C, row)
    {
        let result = [];
        let top_val;
        let maxArea = 0;
        let area = 0;

        let i = 0;
        while (i < C) {
            if (result.length == 0
                || row[result[result.length
- 1]] <= row[i]) {
                result.push(i++);
            }

            else {
                top_val =
row[result[result.length - 1]];
                result.pop();
                area = top_val * i;
 if (result.length > 0) {
                    area = top_val * (i -
result[result.length - 1] - 1);
                }
                maxArea = Math.max(area,
maxArea);
            }
        }

        while (result.length > 0) {
            top_val = row[result[result.length -
1]];
            result.pop();
            area = top_val * i;
            if (result.length > 0) {
                area = top_val * (i -
result[result.length - 1] - 1);
            }

            max_area = Math.max(area, max_area);
        }
        return max_area;
    }
```

```
function biggestUbMatrix(A)
    {
              let R = A.length;
        let C = A[0].length;
        let maxRectangle = maxHist(R,
C, A[0]);

        for (let i = 1; i < R; i++) {
            for (let j = 0; j < C; j++)
{

                if (A[i][j] == 1) {
                    A[i][j] += A[i -
1][j];
                }
            }
 maxRectangle = Math.max(maxRectangle,
maxHist(R, C, A[i]));
        }

        return maxRectangle;
    }


      let matrix = [ [ 0, 1, 1, 0 ,1],
               [ 1, 1, 1, 1 ,1],
               [ 1, 1, 1, 1 ,1],
               [ 1, 1, 0, 0 ,1] ];
console.log(biggestUbMatrix(matrix));
```

Problem-5: Basic Computator

Given a string of expression, create a program that evaluates the string.
Note: The expression will contains only binary operations such as +,-,/ & * and can be in postfix, infix of prefix notations.

Input:
"212 * (65 + 12)"

Output:
16324

Input:
"123/0"

Output:
Cannot divide by zero

Expected:
Time Complexity: O(n)
Space Complexity: O(n)

Approach:
Consider the element in the expression as tokens. Iterate over the expression and check

1. If token is an integer pus it to the number stack
2. If token is a variable compute its value and push it to number stack
3. If token is a left parenthesis then push it to the symbol stack
4. If token is a right parenthesis then pop the symbol from the symbol stack and perform step 4.1 till the time we get left parenthesis

   4.1 If the pop element from symbol stack is an arithmetic operator then pop two numbers from the number stack and then compute the value and push it back to the number stack.
5. if the symbol is an arithmetic operator then we will pop two integer from the number stack, compute the expression and push the result back to the integer stack.

Perform these steps till the time symbol stack is not empty.

Code:

```
function compute(expression)
    {
        let tokens = expression.split(''); // Get the tokens in the expression
        let num_stack = []; //stack to store integer
        let sym_stack = []; //Stack to store symbols

        for (let i = 0; i < tokens.length; i++)
        {
            if (tokens[i] == ' ')
                continue;
            if (tokens[i] >= '0' && tokens[i] <= '9')
            {
                let sbuf = "";
                //If the consecutive tokens are integers then append the values
                while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9')
                    sbuf = sbuf + tokens[i++];
                //Convert it to string and store it in integer stack
                num_stack.push(parseInt(sbuf, 10));
```

```
                    i--;
                }
                //If bracket open then
directly push it into stack
                else if (tokens[i] ==
'(')

sym_stack.push(tokens[i]);
                //If bracket is close
then compute the expression by
poping values from integers stack
                else if (tokens[i] ==
')')
                {
                    while
(sym_stack[sym_stack.length - 1] !=
'(')
                    {

num_stack.push(computeOpertation(sym
_stack.pop(),
```

```
num_stack.pop(),

num_stack.pop()));
                }
                sym_stack.pop();
            }

            else if (tokens[i] == '+'
||
                     tokens[i] == '-'
||
                     tokens[i] == '*'
||
                     tokens[i] == '/')
            {
                while (sym_stack.length
> 0 &&

hasPrecedence(tokens[i],

sym_stack[sym_stack.length - 1]))
```

Thank You!