CENG2001- 2020 – HOMEWORK - Report

04-Ocak-2021

Name : Hasan Ali Özkan
ID : 180709020

Challenge :Maximum Element

Example Input: (10, (1 97), 2, (1 20), 2, (1 26), (1 20), 2, 3, (1 91), 3)

Time Complexity: Pushing element is O(1)
Popping element is O(n) // n is the size of the stack.
Printing element is O(1)

In my 'MyStack' class I defined some methods to the stack which are custom.For example, the 'push' method checks whether the element to be added is greater than the current 'max' variable. If the element to be added is greater than 'max' then simply sets this element to 'max' otherwise it simply pushes the element into the stack.Another custom method is 'pop'.T he pop method removes the top most element in the stack but it checks if the element to be popped is the max element in the stack. If it is , It looks for the second maximum element in the stack then sets this element to 'max' otherwise it simply pops the topmost element into the stack.'printMax' method just print the 'max' variable which Is the maximum element in the stack.

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int number = scan.nextInt();
        MyStack stack = new MyStack();
        while (number>0){
            int ch = scan.nextInt();
            if (ch == 1){
                int value = scan.nextInt();
                stack.push(value);
            }
            else if(ch == 2){
                stack.pop();
            }
            else if (ch == 3){
                stack.printMax();
            }
            number--;
        }


    }
}
class MyStack {
    private int max = Integer.MIN_VALUE;
    private LinkedList<Integer> stack = new LinkedList<>();

    public void push(int data){
        stack.add(data);
        if (data>this.max)
            this.max = data;
    }
    public int pop(){
        int ret = Integer.MIN_VALUE;
        if (!stack.isEmpty()){
            ret = stack.removeLast();
            if(ret == max){
                max = Integer.MIN_VALUE;
                for (int i = 0; i <stack.size() ; i++) {
                    if (stack.get(i) > max)
                        max = stack.get(i);
                }
            }
        }
        return ret;
    }

    public void printMax(){
        System.out.println(this.max);
    }
}
```

Challenge : Equal Stacks

Example Input: (5,3,4),(3,2,1,1,1),(4,3,2),(1,1,4,1)

Time Complexity: O(n)

Instead of keeping the height of each cell separately, I assign the current height for cell into the stacks.I use stack class because with stack its so easy. Then I'm just checking that the top elements in the stacks are equal to all three stacks.I pop the element in the highest stack until all three top most element equal in the stacks.

```
25      public static int equalStacks( List<Integer> h1,  List<Integer> h2,  List<Integer> h3) {
26         Stack<Integer> stackh1 = new Stack<>();
27         Stack<Integer> stackh2 = new Stack<>();
28         Stack<Integer> stackh3 = new Stack<>();
29         int height1=0,height2=0,height3=0;
30         for (int i = h1.size()-1;i>=0 ;i--) {
31             height1 += h1.get(i);
32             stackh1.push(height1);
33         }
34         for (int i = h2.size()-1;i>=0 ;i--) {
35             height2 += h2.get(i);
36             stackh2.push(height2);
37         }
38         for (int i = h3.size()-1;i>=0 ;i--) {
39             height3 += h3.get(i);
40             stackh3.push(height3);
41         }
42
43
44         while (true){
45             if (stackh1.isEmpty() || stackh2.isEmpty() || stackh3.isEmpty())
46                 return 0;
47             height1 = stackh1.peek();
48             height2 = stackh2.peek();
49             height3 = stackh3.peek();
50
51             if (height1 == height2 && height2 == height3)
52                 return height1;
53
54
55             if (height1>=height2 && height1>=height3)
56                 stackh1.pop();
57             else if (height2>=height1 && height2>=height3)
58                 stackh2.pop();
59             else stackh3.pop();
60         }
61
62     }
63
64 }
```

Challenge : Balanced Brackets
Time Complexity: O(n) // n is the length of the s.
Example Input:

3
{[()]}
{[(])}
{{[[(())]]}}

```
// Complete the isBalanced function below.
static String isBalanced(String s) {
    Stack<Character> brackets = new Stack<Character>();
    for (int i = 0; i <s.length() ; i++) {
        if (s.charAt(i) == '[' || s.charAt(i) == '(' || s.charAt(i) =='{'){
            brackets.push(s.charAt(i));
            continue;
    }
        if(brackets.isEmpty())
            return "NO";
        if (s.charAt(i) ==')' && brackets.peek() == '(')
            brackets.pop();
        else if (s.charAt(i) ==']' && brackets.peek() == '[')
            brackets.pop();
        else if (s.charAt(i) =='}' && brackets.peek() == '{')
            brackets.pop();
        else
            return"NO";
    }
    if (brackets.isEmpty())
        return "YES";
return "NO";
}
```

I used the stack.In my opinion this is the best approach. I don't traverse any huge list I am just pushing the open parenthesis, open square , and open curly brackets. I am checking the current element is not pushable(that means it is close parenthesis or close square or close curly brackets.) then if it is matching the element which is the top-most element in the stack. If it is I am doing the same thing the next element if all elements are match then simply return the 'YES' otherwise return the 'NO'. Also I am checking if all elements are pushable then simply return 'NO'.

Challenge : Queue using two stack

Time Complexity: Enqueu is O(1)
Dequeu is O(n) // n is the number of element in the queue which is created with two stacks.
Printing is O(n) //The n same as the dequeue.

Example Input: 10,(1,42),2,(1,14),3,(1,28),3,(1,60),(1,78),2,2

```
public class Solution {
 public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        Stack<Integer> stack2 = new Stack<>();
        Scanner scan = new Scanner(System.in);
        int number = scan.nextInt();
        while (number>0){
             int ch = scan.nextInt();
             if(ch == 1 ){
                 int value = scan.nextInt();
                 stack1.push(value);
             }
             else if (ch == 2 ){
                 if (stack2.isEmpty()){
                     while (!stack1.isEmpty()){
                         stack2.push(stack1.pop());
                     }
                 }
                 if(!stack2.isEmpty()){
                     stack2.pop();
                 }
             }
             else if (ch == 3){
                 if (stack2.isEmpty()){
                     while (!stack1.isEmpty()){
                         stack2.push(stack1.pop());
                     }
                 }
                     System.out.println(stack2.peek());
             }
            number--;


        }
    }
}
```

I used two stacks because it's required. The logic of the queue is FIFO but the logic of the stack is LIFO so I have to use 2 stacks to reverse the order. Every enqueue process I simply add the current value in to the stack1.Every dequeue process I checked the emptiness of the stack2 if it is empty I push the all elements into the stack to from stack1.This process same as the printing the first element in the queue which is created with two stacks.

Challenge : QHEAP1
Time Complexity: Adding an element is O(logn)
Removing an element is O(nlogn) // because first we search for the element.
Printing is O(1)
Example Input: 5,(1,4),(1,9),3,(2,4),3

```
4  public class Solution {
5
6      public static void main(String[] args) {
7          Scanner scan = new Scanner(System.in);
8          int number = scan.nextInt();
9          Queue<Integer> heap = new PriorityQueue<>();
10         while (number>0){
11             int ch = scan.nextInt();
12             if (ch == 1 ){
13                 int insert = scan.nextInt();
14                 heap.add(insert);
15             }
16             else if (ch == 2){
17                 int remove = scan.nextInt();
18                 heap.remove(remove);
19             }
20             else if (ch == 3){
21                 System.out.println(heap.peek());
22             }
23             number--;
24         }
25     }
26 }
```

In Java, the priority queue is implemented using heap data structures. That's why I thought I should use priority queue.The code is very simple I just add an element into the heap variable and it puts the variable on the right place.Removing is the same I remove the element from the heap first search the element then remove it.Printing is the simplest just print the first element into the heap variable.

Challenge : Jessie And Cookies
Time Complexity: O(nlogn) // adding and removing takes logn time but the while loop takes n time.
Example Input: (6,7),(1,2,3,9,10,12)

```
12      static int cookies(int k, int[] A) {
13          Queue<Integer> pqueue = new PriorityQueue<Integer>();
14          for (int i = 0; i <A.length ; i++) {
15              pqueue.add(A[i]);
16          }
17          int operations = 0;
18          while(pqueue.size()>1){
19              if (pqueue.peek()>=k)
20                  return operations;
21              else {
22                  int cookie = pqueue.remove() + pqueue.remove()*2;
23                  pqueue.add(cookie);
24                  operations++;
25              }
26          }
27          if (pqueue.size() > 0 && pqueue.peek() >= k)
28              return operations;
29
30          return -1;
31
32      }
```

I use priority queue because the priority queue is implemented using heap data structures which is I want to efficiency. Until one cookie left in my priority queue I am checking the first cookie sweetness is greater than or equal to Jessie's want if it is I simply return the number of operations I have applied to make this happen.Otherwise I take the least sweetness cookie and take twice the second at least sweetness cookie and add to it. Then I add the result to my priority queue.And I increase the number of operations.If my queue size is 1 and the sweetness of cookie in the queue is greater than Jessie's want then I simply return number of opertions.Otherwise return -1 that means this is impossible.