Reverse a stack using recursion.

Posted on February 5, 2019 | by Prashant Yadav

Posted in Algorithms, Stack | Tagged medium

An algorithm to reverse a stack using recursion.

We will implement an algorithm to reverse a stack using recursion in javascript. Everything will be written in ES6.

Example

```
Input:
5 4 3 2 1

Output:
1 2 3 4 5
```

Implementation

- We will be using two functions which will be calling one another recursively to reverse the stack.
- First function will be used to remove each item from the stack and pass it to the second function to add it at the top of the stack.
- Then the second function will check if stack is empty or not. If it is empty then add the items in the stack else call the first function.

Practically prepare for your JavaScript interview

JavaScript Revision
JavaScript-Concept Based Problems
Data Structures
Algorithms
Machine Coding
Web Fundamentals

Copy

```
//First function to reverse the stack
let reverseStack = (stack) => {
  //If stack has value then call it revcursively
  if(!stack.isEmpty()){
      let temp = stack.pop();
      reverseStack(stack);

      //Pass the element to second function to add it at top
      insertAtBottom(temp, stack);
  }
}

//Second function to add the items at the bottom
let insertAtBottom = (temp, stack) => {
  //If stack is empty then add the item
  if(stack.isEmpty()){
    stack.push(temp);
  }else{

    //Else call the same function recursively
    let x = stack.pop();
    insertAtBottom(temp, stack);
    stack.push(x);
  }
}
```

Copy

```
Input:
 let stack = new Stack();
 stack.push(1);
 stack.push(2);
 stack.push(3);
 stack.push(4);
 stack.push(5);

 reverseStack(stack); //call the function

 //Print the stack
 while(!stack.isEmpty()){
   console.log(stack.pop());
 }

Output:
1
2
3
4
5
```

Time complexity: O(n ^ 2).

Space complexity: O(n).

Time and Space complexity

- We are using two functions in which first function is calling itself recursively as well as the second function in each call. Also the second function is calling itself recursively, so Time complexity is O(n ^ 2).

- We are storing both the functions in call stack one after another, so Space complexity is O(2n) = O(n).

Recommended Posts:

Bubble sort using two stacks
3Sum closest
Check if binary tree has path sum
Print all subarrays with a given sum k in an array
Insertion sort algorithm in javascript
Print right view of a binary tree
Top view of a binary tree
Find least frequent number from an array
Program to print the diamond pattern
Count number of sub-string occurrence in a string

Prev Next

Comments

sr says:
July 22, 2020 At 2:19 Pm

hey prashant!
could u please explain the space complexity part…..i think it should be o(n) because at a tim e only one of the 2 functions will be on stack trace.

Reply

Prashant Yadav says:
July 22, 2020 At 6:54 Pm

Yes, thanks for pointing it out. Updated it.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Start typing...

Name*

Name

Email*

Email

POST COMMENT

Advertisements

■ ■ ■