Learn how to implement two stack with an array

Posted on July 16, 2019 | by Prashant Yadav

Posted in Algorithms, Arrays, Stack | Tagged Easy

An algorithm to implement two stacks with a single array.

We are going to create a data structure called `twoStacks` which will be using only a single array to store the data but will act as two different stacks.

The `twoStacks` data structure will perform following operations.

- **push1(elm):** This will add data in the first stack.
- **push2(elm):** This will add data in the second stack.
- **pop1():** This will remove the data from the first stack.
- **pop2():** This will remove the data from the second stack.

Example

```
Input:
let stack = new twoStacks(10);

//Push data in first stack
stack.push1('Prashant');

//Push data in second stack
stack.push2('Yadav');

//Pop data from first stack
console.log(stack.pop1());

//Pop data from second stack
console.log(stack.pop2());

Output:
"Prashant"
"Yadav"
```

Practically prepare for your JavaScript interview

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

Implementation of two stack with an array.

There are two different ways in which we can implement this.

Method 1: By dividing the array in two equal halves

The most simplest way is to implement two stacks in an array is by dividing the array in two equal halves and using these halves as two different stacks to store the data.

This method works fine, however it is not space efficient because suppose we have two stack with **4** and **6** elements and our array is of **10** length. No if we divide our array in two equal halves then it is going to have two stacks of length **5**. If we push only **4** items in the first stack then it has one space vacant and when we try to push **6** items in the second stack it will overflow because it only has a capacity of **5**. We could have used the **1** vacant space of the first stack to store the data.

Method 2: A space efficient method.

This method is very space efficient and it does not overflow if there is space available in the array or any of the stack.

The concept we use here is we store the data on the two different ends in the array (from start and from end).

The first stack stores the data from the front that is at the index `0` and the second stack stores the data from the end that is the index `size-1`.

Both stack push and pop data from opposite ends and to prevent the overflow we just need to check if there is space in the array.

Copy

```
class twoStacks {

  //Initialize the size of the stack
  constructor(n){
    this.size = n;
    this.top1 = -1;
    this.top2 = n;
    this.arr = [];
  }

  //Push in stack1
  push1 = (elm) => {
    //Check if there is space in array
    //Push at the start of the array
    if(this.top1 < this.top2 - 1){
      this.arr[++this.top1] = elm;
    }else{
      console.log('Stack overflow');
      return false;
    }
  }

  //Push in stack2
  push2 = (elm) => {
    //Check if there is space in array
    //Push at the end of the array
    if(this.top1 < this.top2 - 1){
      this.arr[--this.top2] = elm;
    }else{
      console.log('Stack overflow');
      return false;
    }
  }

  //Pop from the stack 1
  pop1 = () => {
    //Check if stack1 has data
    //Remove it from the front of the stack
    if(this.top1 >= 0){
       let elm = this.arr[this.top1];
       this.top1--;
       return elm;
    }else{
       console.log('stack underflow');
       return false;
    }
  }

  //Pop from the stack 2
  pop2 = () => {
    //Check if stack2 has data
    //Remove it from the end of the array
    if(this.top2 < this.size){
       let elm = this.arr[this.top2];
       this.top2++;
       return elm;
    }else{
       console.log('stack underflow');

       return false;
    }
  }
}
```

Copy

```
Input:
let stack = new twoStacks(10);
//push in first stack
stack.push1('Prashant');

//push in second stack
stack.push2('Yadav');

//pop from first stack
console.log(stack.pop1());

//pop from second stack
console.log(stack.pop2());

Output:
"Prashant"
"Yadav"
```

Time Complexity

#	Access	Search	Insert	Delete
Average	Θ(N)	Θ(N)	Θ(1)	Θ(1)
Worst	O(N)	O(N)	O(1)	O(1)

Space Complexity

#	space
Worst	O(N)

Recommended Posts:

- Bubble sort using two stacks
- Merge two sorted linked list
- How to find loop in linked list
- Convert an array to string javascript
- Longest repeated subsequence
- Check if given number is armstrong in javascript
- Sort a stack using another stack
- Check if an array is palindrome in javascript
- What is the difference between an array and an object in JavaScript?
- Check if string contains a substring in javascript

Prev Next