Implement queue using two stack

Posted on April 14, 2020 | by Prashant Yadav

Posted in Algorithms, Queue, Stack | Tagged medium

Learn how to implement queue using two stack in javascript.

We will create the queue data structure using two different stacks instead of using an array or linkedlist.

Following is the list of operations we will be adding in our queue.

- **enqueue**: Adds the element in the queue.
- **dequeue**: Removes the element from the queue.
- **peek**: Returns the top element.

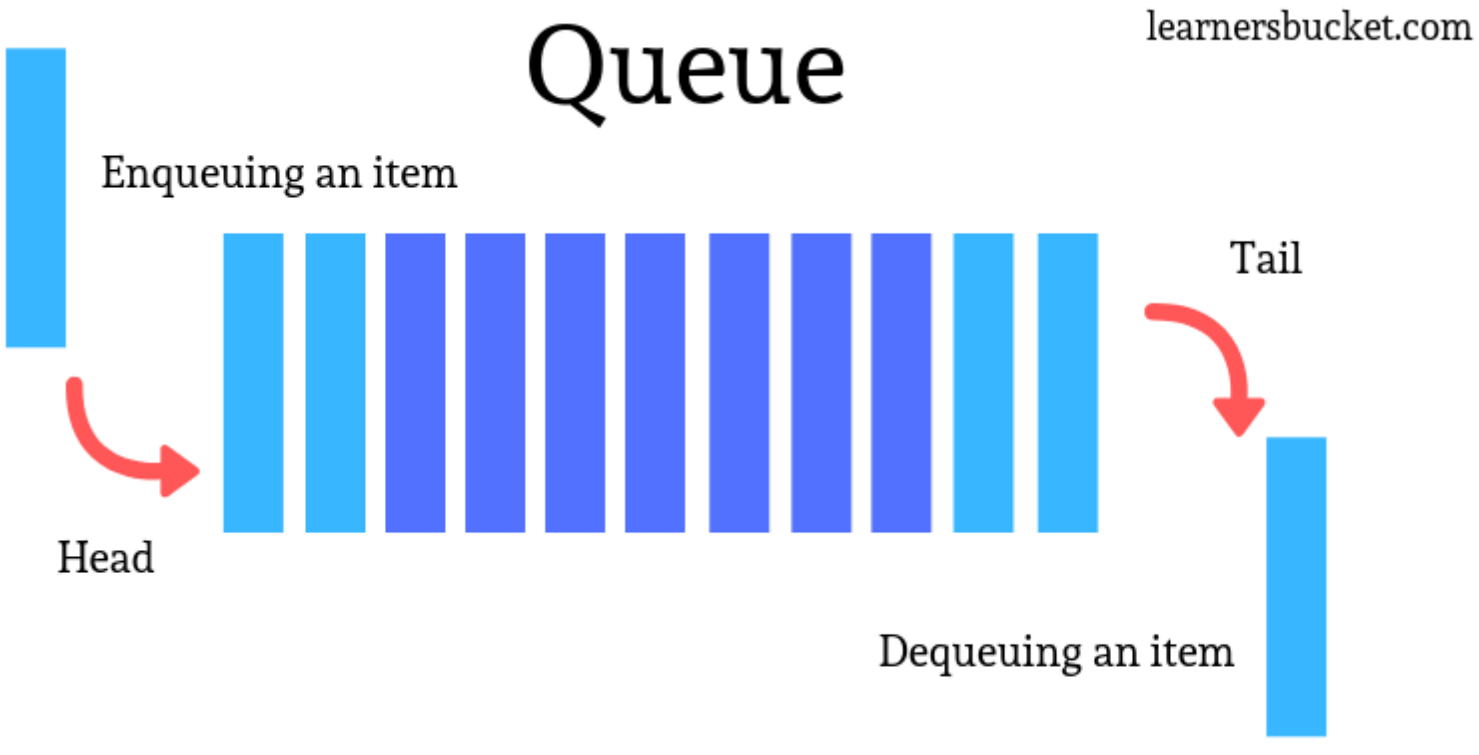
There are two different approaches which can be used to create queue using stack.
1. By making the enqueue operation costly.
2. Making the dequeue operation costly.

As you know that generally the enqueue and dequeue operation of queue is performed in O(1) time.

But as we are using stack rather than array or linked list, either one of the operation will be running in O(n) time. It is up to us to decide which way we want to proceed.

Practically prepare for your JavaScript interview

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



Implement queue using stack by making the enqueue operation costly.

Every time an item is added to the queue first copy all the items from the stack1 to the stack2 then add the new element to the stack2.

Then again copy all the items back from stack2 to stack1.

This way we keep FIFO principal in place by keeping the first element at front and last element at end.

Copy

```
//Enqueue costly
class QueueUsingStack {
  constructor() {
    this.stack1 = new Stack();
    this.stack2 = new Stack();
  }

  enqueue = (x) => {
    while(!this.stack1.isEmpty()){
      this.stack2.push(this.stack1.pop());
    }

    this.stack2.push(x);

    while(!this.stack2.isEmpty()){
      this.stack1.push(this.stack2.pop());
    }
  }

  dequeue = () => {
    return this.stack1.pop();
  }

  peek = () => {
    return this.stack1.peek();
  }

  size = () => {
    return this.stack1.size();
  }

  isEmpty = () => {
    return this.stack1.isEmpty();
  }

  clear = () => {
    this.stack1 = new Stack();
    this.stack2 = new Stack();
  }
}
```

```
Input:
const queue = new QueueUsingStack();
queue.enqueue(10);
queue.enqueue(20);
queue.enqueue(30);
queue.enqueue(40);
queue.enqueue(50);
console.log(queue.peek());
console.log(queue.dequeue());
console.log(queue.peek());
console.log(queue.dequeue());
console.log(queue.peek());

Output:
10
10
20
20
30
```

Time Complexity

#	Enqueue	Dequeue	Peek
Worst	O(N)	O(1)	O(1)

This is the most efficient way because once we have stored the data in organized way all other operations works perfectly, we don’t need to change them.

Making the dequeue costly

In this we will store the element in LIFO order using stack, but while removing the element we will have to copy all the elements into another stack so that the first element is at the top and can be removed.

And then copy all the elements back to the first stack.

The only problem with this way is that for the `peek` operation also we have to do the same process, this makes two operations to run in O(n) time which makes it highly inefficient.

It is always to better to store the data in organized way so that all the later process don’t get affected.

Copy

```
class QueueUsingStack {
  constructor() {
    this.stack1 = new Stack();
    this.stack2 = new Stack();
  }

  enqueue = (x) => {
    this.stack1.push(x);
  }

  dequeue = () => {
    while(!this.stack1.isEmpty()){
      this.stack2.push(this.stack1.pop());
    }

    const item = this.stack2.pop();

    while(!this.stack2.isEmpty()){
      this.stack1.push(this.stack2.pop());
    }

    return item;
  }

  peek = () => {
    while(!this.stack1.isEmpty()){
      this.stack2.push(this.stack1.pop());
    }

    const item = this.stack2.peek();

    while(!this.stack2.isEmpty()){
      this.stack1.push(this.stack2.pop());
    }

    return item;
  }

  size = () => {
    return this.stack1.size();
  }

  isEmpty = () => {
    return this.stack1.isEmpty();
  }

  clear = () => {
    this.stack1 = new Stack();
    this.stack2 = new Stack();
  }
}
```

```
Input:
const queue = new QueueUsingStack();
queue.enqueue(10);
queue.enqueue(20);
queue.enqueue(30);
queue.enqueue(40);
queue.enqueue(50);
console.log(queue.peek());
console.log(queue.dequeue());
console.log(queue.peek());
console.log(queue.dequeue());
console.log(queue.peek());

Output:
10
10
20
20
30
```

Time Complexity

#	Enqueue	Dequeue	Peek
Worst	O(N)	O(1)	O(N)

Recommended Posts:

Reverse a sublist of linked list

Check if given binary tree is full.

Find the LCM of two numbers in javascript

Tree traversal in Javascript

Find all anagrams substring in a string.

Print all the quadruplets with given | 4 sum algorithm

Merge sort a linked list

Program to print the pyramid pattern

Find the biggest perfect square in an array

Algorithm to merge two sorted array

Prev Next

Handcrafted with ♥somewhere in **Mumbai**