Dash
All
Articles
Videos
Problems
Quiz
Prev
Next

Stack with getMin() in O(1)

Design a Data Structure SpecialStack that supports all the stack operations like push(), pop(), isEmpty(), isFull() and an additional operation **getMin()** which should return minimum element from the SpecialStack. All these operations of SpecialStack must have a time and space complexity of O(1).
Note: To implement SpecialStack, you should only use standard Stack data structure and no other data structure like arrays, lists, etc

Example:

> ***Input:*** *Consider the following SpecialStack*
>
> *16 –> TOP*
> *15*
> *29*
> *19*
> *18*
>
> *When getMin() is called it should return 15,*
> *which is the minimum element in the current stack.*
>
> *If we do pop two times on stack, the stack becomes*

> *29 –> TOP*
> *19*
> *18*
>
> *When getMin() is called, it should return 18*
> *which is the minimum in the current stack.*

Approach: To solve the problem follow the below idea:

> *We define a variable **minEle** that stores the current minimum element in the stack. Now the interesting part is, how to handle the case when the minimum element is removed. To handle this, we push "2x – minEle" into the stack instead of x so that the previous minimum element can be retrieved using the current minEle and its value stored in the stack*

Follow the given steps to implement the stack operations:

Push(x): Insert x at the top of the stack

- If the stack is empty, insert x into the stack and make minEle equal to x.
- If the stack is not empty, compare x with minEle. Two cases arise:
  - If x is greater than or equal to minEle, simply insert x.
  - If x is less than minEle, insert (2*x – minEle) into the stack and make minEle equal to x.
    For example, let the previous minEle be 3. Now we want to insert 2. We update minEle as 2 and insert 2*2 – 3 = 1 into the stack

Pop(): Removes an element from the top of the stack

- Remove the element from the top. Let the removed element be y. Two cases arise:
  - If y is greater than or equal to minEle, the minimum element in the stack is still minEle.

- If y is less than minEle, the minimum element now becomes (2*minEle – y), so update (minEle = 2*minEle – y). This is where we retrieve the previous minimum from the current minimum and its value in the stack.
  For example, let the element to be removed be 1 and minEle be 2. We remove 1 and update minEle as 2*2 – 1 = 3

Important Points:

> - *Stack doesn't hold the actual value of an element if it is minimum so far.*
> - *The actual minimum element is always stored in the minEle variable*

Below is the illustration of the above approach:

> ***Push(x)***
>
> | Number Inserted | Present Stack | minELe |
> |---|---|---|
> | 3 | 3 | 3 |
> | 5 | 5 3 | 3 |
> | 2 | 1 5 3 | 2 |
> | 1 | 0 1 5 3 | 1 |
> | 1 | 1 0 1 5 3 | 1 |
> | -1 | -3 1 0 1 5 3 | -1 |

- *Number to be Inserted: 3, Stack is empty, so insert 3 into stack and minEle = 3.*
- *Number to be Inserted: 5, Stack is not empty, 5> minEle, insert 5 into stack and minEle = 3.*
- *Number to be Inserted: 2, Stack is not empty, 2< minEle, insert (2*2-3 = 1) into stack and minEle = 2.*
- *Number to be Inserted: 1, Stack is not empty, 1< minEle, insert (2*1-2 = 0) into stack and minEle = 1.*
- *Number to be Inserted: 1, Stack is not empty, 1 = minEle, insert 1 into stack and minEle = 1.*
- *Number to be Inserted: -1, Stack is not empty, -1 < minEle, insert (2*-1 – 1 = -3) into stack and minEle = -1.*

Pop()

Number Removed	Orignal Number	Present Stack	minEle
-	-	-3 1 0 1 5 3	-1
-3	-1	1 0 1 5 3	1
1	1	0 1 5 3	1
0	1	1 5 3	2
1	2	5 3	3
5	5	3	3

- *Initially the minimum element minEle in the stack is -1.*

- *Number removed: -3, Since -3 is less than the minimum element the original number being removed is minEle which is -1, and the new minEle = 2*-1 – (-3) = 1*
- *Number removed: 1, 1 == minEle, so number removed is 1 and minEle is still equal to 1.*
- *Number removed: 0, 0< minEle, original number is minEle which is 1 and new minEle = 2*1 – 0 = 2.*
- *Number removed: 1, 1< minEle, original number is minEle which is 2 and new minEle = 2*2 – 1 = 3.*
- *Number removed: 5, 5> minEle, original number is 5 and minEle is still 3*

Below is the implementation of the above approach:

C++ Java

```
// Java program to implement a stack that supports
// getMinimum() in O(1) time and O(1) extra space.
import java.util.*;

// A user defined stack that supports getMin() in
// addition to push() and pop()
class MyStack {
    Stack<Integer> s;
    Integer minEle;

    // Constructor
    MyStack() { s = new Stack<Integer>(); }

    // Prints minimum element of MyStack
    void getMin()
```

```
{
    // Get the minimum number in the entire stack
    if (s.isEmpty())
        System.out.println("Stack is empty");

    // variable minEle stores the minimum element
    // in the stack.
    else
        System.out.println("Minimum Element in the "
                           + " stack is: " + minEle);
}

// prints top element of MyStack
void peek()
{
    if (s.isEmpty()) {
        System.out.println("Stack is empty ");
        return;
    }

    Integer t = s.peek(); // Top element.

    System.out.print("Top Most Element is: ");

    // If t < minEle means minEle stores
    // value of t.
    if (t < minEle)
        System.out.println(minEle);
    else
```

```
        System.out.println(t);
}

// Removes the top element from MyStack
void pop()
{
    if (s.isEmpty()) {
        System.out.println("Stack is empty");
        return;
    }

    System.out.print("Top Most Element Removed: ");
    Integer t = s.pop();

    // Minimum will change as the minimum element
    // of the stack is being removed.
    if (t < minEle) {
        System.out.println(minEle);
        minEle = 2 * minEle - t;
    }

    else
        System.out.println(t);
}

// Insert new number into MyStack
void push(Integer x)
{
    if (s.isEmpty()) {
```

```
            minEle = x;
            s.push(x);
            System.out.println("Number Inserted: " + x);
            return;
        }

        // If new number is less than original minEle
        if (x < minEle) {
            s.push(2 * x - minEle);
            minEle = x;
        }

        else
            s.push(x);

        System.out.println("Number Inserted: " + x);
    }
};

// Driver Code
public class Main {
    public static void main(String[] args)
    {
        MyStack s = new MyStack();

        // Function calls
        s.push(3);
        s.push(5);
        s.getMin();
```

```
        s.push(2);
        s.push(1);
        s.getMin();
        s.pop();
        s.getMin();
        s.pop();
        s.peek();
    }
}
```

Output

```
Number Inserted: 3
Number Inserted: 5
Minimum Element in the stack is: 3
Number Inserted: 2
Number Inserted: 1
Minimum Element in the stack is: 1
Top Most Element Removed: 1
Minimum Element in the stack is: 2
Top Most Element Removed: 2
Top Most Element is: 5
```


How does this approach work?

When the element to be inserted is less than minEle, we insert “2x – minEle”. The important thing to note is, that 2x – minEle will always be less than x (proved below), i.e., new minEle and while popping out this element we will see that

something unusual has happened as the popped element is less than the minEle. So we will be updating minEle.

> *How 2*x – minEle is less than x in push()?*
>
> *x < minEle which means x – minEle < 0*
>
> *// Adding x on both sides*
> *x – minEle + x < 0 + x*
> *2*x – minEle < x*
> *We can conclude 2*x – minEle < new minEle*

While popping out, if we find the element(y) less than the current minEle, we find the new minEle = 2*minEle – y

> *How previous minimum element, prevMinEle is, 2*minEle – y*
> *in pop() is y the popped element?*
>
> *// We pushed y as 2x – prevMinEle. Here*
> *// prevMinEle is minEle before y was inserted*
>
> *y = 2*x – prevMinEle*
>
> *// Value of minEle was made equal to x*
> *minEle = x .*
>
> *new minEle = 2 * minEle – y*
> *= 2*x – (2*x – prevMinEle)*
> *= prevMinEle // This is what we wanted*

Design a stack that supports getMin() in O(1) time and O(1) extra space by creating a MinStack class:

To solve the problem follow the below idea:

> *Create a class node that has two variables Val and min. Val will store the actual value that we are going to insert in the stack, whereas min will store the min value so far seen up to that node*

Below is the implementation of the above approach:

C++ Java

```
// Java program for the above approach

import java.io.*;
import java.util.*;
class MinStack {
    Stack<Node> s;

    class Node {
        int val;
        int min;
        public Node(int val, int min)
        {
            this.val = val;
```

```
            this.min = min;
        }
    }

    /** initialize your data structure here. */
    public MinStack() { this.s = new Stack<Node>(); }
    public void push(int x)
    {
        if (s.isEmpty()) {
            this.s.push(new Node(x, x));
        }
        else {
            int min = Math.min(this.s.peek().min, x);
            this.s.push(new Node(x, min));
        }
    }
    public int pop() { return this.s.pop().val; }
    public int top() { return this.s.peek().val; }
    public int getMin() { return this.s.peek().min; }
}

// Driver code
class GFG {

    public static void main(String[] args)
    {
        MinStack s = new MinStack();

        // Function calls
```

```
        s.push(-1);
        s.push(10);
        s.push(-4);
        s.push(0);
        System.out.println(s.getMin());
        System.out.println(s.pop());
        System.out.println(s.pop());
        System.out.println(s.getMin());
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz

Output

```
-4
0
-4
-1
```

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.