Question 3.2 From CTCI
Minimum Stack Elements

October 25, 2017

Question

How would you design a stack which, in addition to push and pop, also has a function min which returns the minimum element? Push, pop and min should all operate in O(1) time .

Explanation and Algorithm

There are two approaches to this problem:

For the brute force solution, we can have each node in the stack contain the minimum value of the stack up to that point. When a node is created and pushed to the stack, we check if the value being added is smaller than the current minimum. If it is, then we set the new min to be this value and create the node with its min field set to the newly determined min. If not, then we keep the old min and push the node with its min field set to that value. In this way, we can find the minimum of the stack just by popping the top node. Keep in mind that you must write a function min that determines the minimum element in the stack.

The brute force implementation isn't very space efficient. If we have a stack with a large number of nodes, keeping track of the min for each node would be costly. Instead, we can use an additional stack to keep track of the mins. This will allow us to avoid duplicating the data.

Hints

1. For the brute force solution, what is the easiest way to keep track of the minimum of the stack? It involves using push().
2. Suppose we were to keep the current minimum stored in each value pushed to the stack. How would you determine the current minimum?
3. For the brute force solution, we would end up with duplication of data for the minimum value. How might you avoid this?

4. Consider implementing a separate stack that keeps track of the minimum values. How would you update the push(), pop(), and min() methods accordingly?

Code

```
/*Brute Force Implementation*/

public class StackWithMin extends Stack<NodeWithMin> {


   public void push(int value){

      // Determine if the new value added is the new minimum of the
          stack and add the node to the stack.
      int currMin = Math.min(value, min());
      super.push(new NodeWithMin(value, currMin));
   }


   public int min(){

      // Get the current minimum value of the stack.

      if(this.isEmpty()){
         return Integer.MAX_VALUE;
      }

      return peek().min;

   }


}

public class NodeWithMin{

   public int value;
   public int min;
   public NodeWithMin(int value, int min){

      this.value = value;
      this.min = min;

   }

}
```

```
/*Space Efficient Implementation*/

public class StackWithMin2 extends Stack<Integer>{

   Stack<Integer> minStack;

   public StackWithMin2(){

      minStack = new Stack<Integer>();

   }

   public void push (int value){

      // If the new value is the new minimum, push to minStack.
          Otherwise the current minimum still holds.
      if(value <= min()){
         minStack.push(value);
      }

      super.push(value);
   }

   public Integer pop(){

      int value = super.pop();

      // If the value being removed is the minimum value, then remove it
          from the minStack as well.
      if(value == min()){
         minStack.pop();
      }

      return value;
   }

   public int min(){

      // Get the current minimum value of the min stack.
      if(minStack.isEmpty()){
         return Integer.MAX_VALUE;
      } else {
         return minStack.peek();
      }

   }
```

```
}
```

Sources

Question, answer and other material taken from Cracking the Coding Interview 4th edition by Gayle Laakmann McDowell.