CTCI 3.2: Stack Min

December 6, 2017

Question

How would you design a stack which, in addition to push and pop, has a function min which returns the minimum element? Push, pop and min should all operate in 0(1) time.

Explanation and Algorithm

In this case, the minimum changes when a smaller element is added to the stack. Given this, we could think of our first solution: create an int value minValue in the stack class. When a minValue is popped from the stack, we search through the stack to find a new minimum. This isn't exactly right, thoguh, because this isn't O(1). Instead, we could keep track of the minimum at each state of the stack (i.e. whenever a value is popped or pushed). This can be done by having each node record the minimum beneath itself. To find this value, look at what the top element thinks is the min. When an element is pushed onto the stack, the element is given its current minimum, and sets its "local min" to be the min.

Hints

1. Recall the operations of the stack...push, pop, peek, and the boolean isEmpty. These are all going to be useful for keeping track of the min.
2. Lots of stuff to recall like classes, extending, and super classes. Because we're keeping track of the min at each state of the stack, we're going to need to come up with a class StackWithMin. This is like the typical Stack class, but handles keeping track of the min.
3. Think of edge cases. Within this class, we're going to come up with methods to push a value onto the stack and one for finding the min.
4. If you've figured out a general algorithm, think of one that could be even more efficient. How about keeping track of mins on a separate stack? Think of ways to build off the initial algorithm to do this.

Code

```
/*Answer 1: Smaller Stack */


public class StackWithMin extends Stack<NodeWithMin> {
   public void push(int value) {
    int newMin = Math.min(value, min());
    super.push(new NodeWithMin(value, newMin));
    }

   public int min(){
      if(this.isEmpty()){
         return Integer.MAX_VALUE; //Error val
      }else{
         return peek().min;
      }
   }
 }
    class NodeWithMin{
    public int value;
    public int minl
      public NodeWithMin(int v, int min){
         value = v;
            this.min = min;
      }
    }

 /*Answer 2: More Efficient for a Bigger Stack */


public class StackWithMin2 extends Stack<Integer> {
   Stack<Integer> s2;
    public StackWithMin2(){
      s2 = new Stack<Integer>();
    }

    public void push(int value){
      if(value <= min()){
         s2.push(value);
        }
      super.push(value);
    }

   public Integer pop(){
      int value = super.pop();
        if(value == min()){
         s2.pop();
        }
```

```
    return value;
    }

   public int min(){
      if(s2.isEmpty()){
         return Integer.MAX_VALUE;
        }else{
         return s2.peek();

        }
    }
}
```

Run time analysis

In this case, the runtime for each answer is O(1).