Interview Questions

算法

Queue with two stacks(两个栈实现一个队列)

一个栈用来入队，一个栈用来出队，出队时将入队栈中的元素全部导入到出队栈。
注意：如果“出队”栈不为空，此时有元素入队，直接将元素压入”入队”栈，省去了导入环节。
入队的时间复杂度为$O(1)$，出队如果需要导入时间复杂度为$O(n)$，不需要导入复杂度为$O(1)$，均摊复杂度是**常数**$O(1)$。

```java
import edu.princeton.cs.algs4.Queue;
import edu.princeton.cs.algs4.Stack;

public class QueueWithTwoStacks {
    private Stack<Integer> stack1;//enqueue
    private Stack<Integer> stack2;//dequeue
    private int N;//队列中元素数
    public QueueWithTwoStacks() {
        stack1 = new Stack<Integer>();
        stack2 = new Stack<Integer>();
        N=0;
    }
    public void Enqueue(int i){
        stack1.push(i);
        N++;
    }
    public int Dequeue(){
        int orginStack1Size=stack1.size();
        if (!stack2.isEmpty()) {
            N--;
            return stack2.pop();
        }
        for (int i=0;i<orginStack1Size;i++){
            stack2.push(stack1.pop());
        }
        N--;
        return stack2.pop();
    }
    public int size(){
        return N;
    }
    public boolean isEmpty(){
        return N==0;
    }
    public static void main(String[] args){
        QueueWithTwoStacks queueWithTwoStacks=new QueueWithTwoStack
s();
        queueWithTwoStacks.Enqueue(1);
        queueWithTwoStacks.Enqueue(2);
        queueWithTwoStacks.Enqueue(3);
        System.out.println(queueWithTwoStacks.Dequeue());
        queueWithTwoStacks.Enqueue(4);
        System.out.println(queueWithTwoStacks.Dequeue());
    }
}
```

Stack with max

找到栈中最大的元素，第一反应也是复杂度最大的想法即是暴力法，遍历栈中的元素进行比较返回最大元素，无论数组还是链表形成的stack，复杂度均为**线性**$O(n)$
如果我在push中就进行比较呢？在push新元素时可以以**常数**$O(1)$时间计算最大/最小值，由于需要记录Stack中最大值，次大值，需要另开辟个栈来保存每次push、pop操作后原先栈的最大值。

```
import edu.princeton.cs.algs4.Stack;

public class StackWithMax {
    private Stack<Integer> stack1;
    private Stack<Integer> stack2;
    private int max = Integer.MIN_VALUE;

    public StackWithMax() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }

    public void Push(int i) {
        stack1.push(i);
        if (i > max)
            max = i;
        stack2.push(max);
    }

    public void Pop() {
        int i = stack1.pop();
        stack2.pop();
        max = stack2.peek();
    }

    public int getMax() {
        return max;
    }

    public static void main(String[] args) {
        StackWithMax stackWithMax = new StackWithMax();
        stackWithMax.Push(-10);
        stackWithMax.Push(10);
        stackWithMax.Push(5);
        stackWithMax.Push(20);

    }
}
```

自己构造Stack:

```java
import java.util.NoSuchElementException;

public class StackWithMax_mystack {
    private int max = Integer.MIN_VALUE;
    private int N = 0;
    private Node first;
    private Node second;

    private class Node {
        int element;
        Node next;
    }

    public void push(int i) {
        Node oldfirst = first;
        first = new Node();
        first.element = i;
        first.next = oldfirst;
        if (first.element > max)
            max = first.element;
        Node oldsecond = second;
        second = new Node();
        second.element = max;
        second.next = oldsecond;
        N++;

    }

    public void pop() {
        if (isEmpty()) throw new NoSuchElementException("Stack Unde
rflow");
        first = first.next;
        second = second.next;
        if (second == null)
            max = Integer.MIN_VALUE;
        else max = second.element;
        N--;
    }

    public boolean isEmpty() {
        return first == null;
    }

    public int top() {
        return first.element;
    }

    public int getMax() {
```

```
            return max;
        }

        public static void main(String[] args) {
        }
    }
```

正好Leetcode题目Min Stack

```java
public class StackWithMin_mystack {

    /**
     * initialize your data structure here.
     */
    private int min = Integer.MAX_VALUE;
    private Node first;
    private Node second;

    private class Node {
        private int element;
        private Node next;
    }


    public void push(int x) {
        Node oldfirst = first;
        first = new Node();
        first.element = x;
        first.next = oldfirst;
        if (x < min) min = x;
        Node oldsecond = second;
        second = new Node();
        second.element = min;
        second.next = oldsecond;

    }

    public void pop() {
        int ele = first.element;
        first = first.next;
        second = second.next;
        if (second == null) min = Integer.MAX_VALUE;
        else min = second.element;

    }

    public int top() {
        return first.element;
    }

    public int getMin() {
        return min;
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
```

```
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```