后端开发常问面试题集锦

——算法问题

一个支持 O(1)时间内完成 pop,push 和 max 的栈

这是一个面试题，跟同事交流得到的。解法十分巧妙，值得学习。

问题描述

要求设计一个栈，支持 pop,push 和 max 三种操作，每种操作都是 O(1)时间的。

问题分析

一般的栈，本身的 pop 和 push 的操作就是 O(1)的，可以考虑使用一个变量来存储最大值。问题在于，如果这个最大值被 pop 出去，这个变量就需要重新计算。如果通过遍历一遍来求出，则需要 O(n)的时间，达不到要求。此外，任何想通过一个排好序的序列来解决最大值的 pop 问题的方案，都有一个致命缺点，就是每次 push 的时候，需要进行插入。

实现一：借助第二个栈

这个方案是基于这样一个事实：如果 push 的元素小于最大值，则当前栈的最大值不会发生变化。举个例子：

输入序列为 2,1,0,5,4,3,6

当 push 和 pop1 和 0 时，栈的最大值不变，为 2。但当进一步输入 5，由于 5 大于 2，则 5 就是当前栈的最大值。继续 push 进来或者 pop 出去 4 和 3，5 都是最大值。当 5 被 pop 出去的时候，2 又变成最大值了。所以只要用一个栈来存储 2 和 5 就行了。每当 push 一个数，如果大于最大值栈的栈顶元素，则同时加入最大值栈。每当 pop 一个数时，如果跟最大值相等，就同样弹出。

代码如下：

```java
package com.twabs.prac;
public class NStack1 {
    private int[] stack = new int[100];
    private int[] maxs = new int[100];
    private int top = 0;
    private int mtop = 0;
    public void push(int el) {
        if (el >= maxs[mtop]) {
            maxs[++mtop] = el;
        }
        stack[++top] = el;
    }
    public int pop() {
        if (top <= 0) return -1;
        if (stack[top] == maxs[mtop]) {
            mtop--;
        }
        return stack[top--];
    }
    public int max() {
        return maxs[mtop];
    }
}
```

实现二：使用一个变量

使用第二个栈的主要原因在于我们需要在 pop 出最大值的时候，将第二大的值变成新的最大值。进一步考虑实现一的基础事实，新 push 的元素小于当前最大值，则最大值不变；否则最大值发生变化。在 pop 出元素的时候，这条事实也是成立的。关键在于最大值发生变化的时候，如何恢复次最大值？大小关系可以通过差值来进行存储，只要在 pop 和 push 的时候计算就行。

那差值能否帮助我们恢复次最大值呢？答案是肯定的。假设现在栈里面有 n-1 个元素，第 n 个元素进来时，我们再栈里面存贮的是(Sn - MAX(Sn-1,Sn-2,...,S1))。如果是正的，表明新的元素是新的最大值。这样在 pop 的时候，如果栈顶是正的，那么用当前最大值减去栈顶元素就可以得到新的最大值！如果是负数，则最大值不变。

这个想法的确很天才，可以将实现的空间复杂度降低到常熟级别。

代码如下：

```java
package com.twabs.prac;
public class NStack2 {
    private int[] stack = new int[100];
    private int top = 0;
    private int max = 0;
    public void push(int el) {
        stack[++top] = el - max;
        if (el > max) {
            max = el;
        }
    }
    public int pop() {
        int t = stack[top] + max;
        if (stack[top] > 0) {
            t = max;
            max -= stack[top--];
        } else {
            top--;
        }
        return t;
    }
    public int max() {
        return max;
    }
}
```

基本操作的威力

这个问题也生动的展示了，为了在数据结构中支持一种基本操作，数据结构本身所进行的改变和调整，同时这种改变和调整极大提高了基本操作的算法效率。