栈和队列面试题精讲

七月算法　**曹鹏**

2015年4月23日

提纲

- 线性表简介
- 面试题总体分析
- 一些例题
  - 例1 元素出入栈顺序合法性判断
  - 例2 用两个队列实现一个堆栈
  - 例3 用两个堆栈实现一个队列
  - 例4 支持查询最小值的堆栈
  - 例5 单调堆栈——最大直方图
  - 例6 单调队列——滑动窗口最大值
- 总结

线性表简介

- 堆栈和队列统称线性表
  - 简单的线性结构
  - 数组和链表可以实现这两种数据结构
- 堆栈
  - 后进先出 (Last In First Out)
- 队列
  - 先进先出 (First In First Out)

面试题总体分析

- 堆栈
  - 基本理解
  - DFS
    - 深度优先——按深度遍历
    - 递归转非递归
- 队列
  - 基本理解
  - BFS
    - 广度优先——按层序遍历

例1 元素出入栈顺序合法性判断

- 例1 给定一些元素的入栈顺序和出栈顺序，问是否可能？（假设所有元素都不相同）
  - 分析： 模拟堆栈即可，如果当前要出栈的元素恰好在栈顶，则必须出栈，否则就入栈。（注意判断两个vector size一样）

```
bool isPossible(vector<int> &in, vector<int> &out) {
        for (int i = 0, j = 0; j < out.size(); ++j) {
                while (s.empty() || s.top() != out[j]) {
                                if (i >= in.size()) return false;
                                                s.push(in[i++]);
                }
                s.pop();
                 }
                 return true;
}
```

例2 用两个队列实现一个堆栈

- 例2 如何用两个队列实现一个堆栈？
  - 队列无论怎么折腾，元素顺序不会改变！
  - 两个队列来回倒，保证一个队列是空的，用空队列临时存储除队尾外所有元素
    - 例如q1非空，q2是空的，要出“栈”，实际上要出的是q1里面最后一个元素，我们把q1里面元素一个一个放入q2里面（所有元素的顺序不会变化），直到剩下一个，再让它出队即可

例2 续

- 入“栈”:维护一个队列是空的 : O(1)

```
push(x) :
    if (!q1.empty()) q1.push(x);
    else q2.push(x);
```

- 出“栈”:用一个队列临时存放元素 : O(n)

```
pop() :
    if (!q1.empty()) {
        while (q1.size() > 1) {
            q2.push(q1.front());
            q1.pop();
        }
        q1.pop();
    }
    else {  //类似操作}
```

例3 用两个堆栈实现一个队列

- 例3 如何堆栈实现一个队列？
  - s1负责“入队”，s2负责“出队”（反向）
  - 入队直接入到s1里
  - 要出队如果s2非空，则先从s2出，否则把s1里面全部元素压入s2中
  - 理解：
    - s1负责存放入队元素
    - s2负责出队并反向
    - 每个元素实际上反向了两次，出入一次s1,出入一次s2

例3 续

- push(x): O(1)

  s1.push(x)
- pop: 均摊O(1) 每个元素出入两个栈各1次

```
if (s2.empty()) {
        while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
         }
 }
s2.pop();
```

例4 支持查找最小元素的堆栈

- 一个堆栈除了支持push, pop以外还要支持一个操作getMin得到当前堆栈里所有元素的最小值
  - 方法1（笨）
    - 用两个堆栈，s1和s2，s1正常使用，s2一直是空的
    - getMin的时候，把s1的元素一个一个弹出到s2，每弹出一个，顺便求当前的最小值，然后再从s2把元素一个一个弹回到s1，也清空了s2: O(n)

例4 续1

- 方法2
  - 用两个堆栈，s1维护原来的值，s2维护最小值 它们元素个数一样多

```
push(x):  O(1)
        s1.push(x);
        if (!s2.empty() && s2.top() < x) s2.push(s2.top());
         else s2.push(x);
pop(): O(1)
        s1.pop();
        s2.pop();
getMin : O(1)
         return s2.top();
```

例4 续2

- 方法3 思路不变，s2真的需要存储那么多值么？ 假设之前入过一个最小值，s2的顶端存了许多相同的最小值

```
push(x):  O(1)
 s1.push(x);
 if (s2.empty() || s2.top() >=  x) s2.push(x);
pop : O(1)
    if (s1.top() == s2.top()) s2.pop();
    s1.pop();
```

例5 最大直方图

- 例5 给出一个直方图，求最大面积矩形（Leetcode 84）
  - 用堆栈计算每一块板能延伸到的左右边界
  - 对每一块板

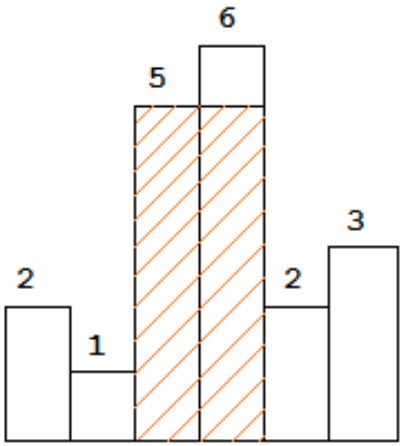
    堆栈顶矮，这一块左边界确定，入栈

    堆栈顶高，堆栈顶右边界确定，出栈，计算面积

    入栈时左边界确定

    出栈时右边界确定

    堆栈里元素是递增的
  - 本质：中间的短板没有用！
  - 复杂度 O(n)



例5 续1

H	0	1	2	3	4	5	6
值	2	1	5	6	2	3	0

新数	堆栈 (顶->底)	说明
H[0] = 2	{2}	2入栈,左边界(-1)
H[1] = 1	{1}	2出栈，右边界（1），1入栈，左边界(-1)
H[2] = 5	{5,1}	5入栈，左边界（1）
H[3] = 6	{6，5，1}	6入栈，左边界（2）
H[4] = 2	{2,1}	6，5出栈，右边界（4），2入栈左边界（1）
H[5] = 3	{3,2,1}	3入栈，左边界(4)
H[6] = 0		3,2,1,出栈 右边界（6）

例5 续2

```
class Solution {
public:
    int largestRectangleArea(vector<int> &height) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
        int n = height.size(),result = 0;
        stack<int> s;
        for (int i = 0; i < n; ++i) {
            while ((!s.empty()) && (height[s.top()] >= height[i])) {
                int h = height[s.top()];
                s.pop();
                result = max(result, (i - 1 - (s.empty()?(-1):s.top())) * h);

            }
            s.push(i);
        }
        while (!s.empty()) {
            int h = height[s.top()];
            s.pop();
            result = max(result, (n - 1 - (s.empty()?(-1):s.top())) * h);
        }
        return result;

    }
};
```

例6 滑动窗口最大值

- 给定一个数组a[0..n],还有一个值k, 计算数组b[i] = max(a[i – k + 1.. i]) 注意认为负数下标对应值是无穷小
  - 方法1: 用一个最大堆存放最近的k个数
    - 计算好b[i – 1]后
    - a[i – k]出堆， 如何找到a[i – k]?
    - a[i]入堆
    - b[i] = 堆顶
    - 时间复杂度O(nlogk),

例6 续1

方法2

- 如果同时存在一个旧的数x,和一个新的数y并且x≤y，则x永远不会是我们要的解。因为：
  - “窗口”朝右滑动
  - x先离开窗口
  - y进入窗口后x与y总是同时存在，直到x离开
  - x没用了.....利用这个性质？
  - 双端队列，队头存旧的数，队尾存新的数
  - 如果队尾的数≤将要入队的数a[i]，则扔掉队尾的数
  - 队列里的从队头到队尾是单减的，队头永远是窗口最大值
  - 考虑：
    - 队头何时过期？
    - 时间复杂度？ O(n) :每个元素出入队一次

例6 续2

- K = 3

a	0	1	2	3	4	5
值	5	1	3	4	2	6

新数	队列（头->尾）	说明
a[0] = 5	{5}	5入队, b[0] = 5
a[1] = 1	{5,1}	1比5小直接入队, b[1] = 5
a[2] = 3	{5,3}	1太小了,被扔掉， 3入队, b[2] = 5
a[3] = 4	{4}	5过期了， 被扔掉。3比4小， 被扔掉,b[3] = 4
a[4] = 2	{4,2}	2比4小， 入队, b[4] = 4
a[5] = 6	{6}	6最大， 把2和4都扔掉, b[5] = 6

例6 续3

- 实现：for (int i = 0; i < n; ++i) {

  while (!q.empty() && q.front() <= i – k) q.pop_front(); //过期

  while (!q.empty() && a[q.back()] <= a[i]) q.pop_back(); //扔队尾

  q.push(i); //入队

  b[i] = a[q.front()];

}

- 理解：
  - 旧的数比较大，因为“过期”而“不得不”出队
  - 存放a数组的“下标”而没存放具体值
- 扩展
  - 如果输入是一个流，我们必须自己保存“时间戳”，决定过期。

总结

- 理解队列堆栈的基本概念
  - n个左右括号的出入栈顺序有多少种？（Catalan数）
- 熟悉队列、堆栈的应用
  - 递归和非递归的转化 dfs
  - Bfs搜索
- 维护队列和堆栈的单调性 *
  - 利用顺序