3，栈和队列

2018年4月27日 16:42

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
  - 超级重要例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（也是线性）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)

3/21 julyedu.com

面试题总体分析

- 堆栈
  - 基本理解
  - 应用：DFS
    - 深度优先——按深度遍历
    - 递归转非递归
- 队列
  - 基本理解
  - 应用：BFS

□ 广度优先——按层序遍历

例1 元素出入栈顺序合法性判断

```
/*
问题： 给定一些元素的入栈顺序和出栈顺序（不是入完再出那种，而是随时入随时出），问是否是可能的？（假设所有元素都不相同）
分析：直接用栈模拟该操作，如果栈顶元素和当前要出栈的元素相等的话，则必须出栈

    否则就入栈。注意判断两个vector size是否一样


//之后我分析了下，发现确实可行，想出这个方法的人是天才吧


*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    bool isPossible(vector<int> in,vector<int> out)
    {
        stack<int> s;
        for(int i = 0,j=0;j<out.size();j++)//遍历出栈队列
        {
            while(s.empty() || s.top() != out[j])//如果栈为空or出栈元素和栈顶不等，入栈。注意这里是while
            {
                if(i>=in.size()) return false;
                s.push(in[i++]);
            }
            s.pop();//出栈元素等于栈顶元素
        }
        return true;
    }
};
int main()
{
    int in_arr[6] = {3,4,5,1,2,3};
    int out_arr[6] = {3,5,4,3,2,1};
    vector<int> in(in_arr,in_arr+6);
    vector<int> out(out_arr,out_arr+6);
    Solution so;
    cout<<so.isPossible(in,out)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

例2 用两个队列实现一个堆栈

```
/*
问题：如何用两个队列实现一个堆栈？

分析：两个队列来回倒，保证一个队列是空的，用空队列来存贮除队尾外的所有元素

    比如，q1非空，q2为空，要出栈，实际上要出的是q1的最后一个元素，我们将q1的元素一个一个放入q2,剩下最后一个，让其出队即可。

    入栈：push(x)：O(1)
        if(!q1.empty()) q1.push(x);
        else q2.push(x)
    出栈：pop():O(N)
        if(!q1.empty())
        {
            while(q1.size()>1)//剩最后一个元素
            {
                q2.push(q1.front());
                q1.pop();
            }
            q1.pop();
        }
        else
        {
            while(q2.size()>1)//剩最后一个元素
            {
                q1.push(q2.front());
                q2.pop();
            }
            q2.pop();
        }
*/
```

来自 <http://tool.oschina.net/highlight>

例3 用两个堆栈实现一个队列

```
/*
问题：用两个堆栈实现一个队列

分析：s1负责入队，s2负责出队。入的话直接入s1,出的话如果s2非空，则先从s2出，否则，将s1全部元素亚入s2

push(x):O(1)
    s1.push(x);

pop(x):平均O(1)，因为每个元素出入两个栈各1次
    if(!s2.empty())
        s2.pop();
    else
    {
        while(!s1.empty())
        {
            s2.push(s1.top());
            s1.pop();
```

```
        }
    }
*/
```

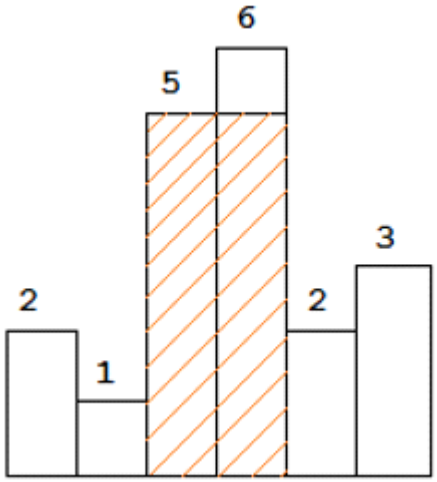
来自 <http://tool.oschina.net/highlight>

例4 支持查找最小元素的堆栈

```
/*
问题：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

分析：方法1（笨方法）：用两个堆，1个正常使用，另一个一直是空，
    getmin的时候，将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将元素一个一个倒回s1, O(n)

    方法2：还是两个堆，一个维护原来的值，s2维护当前最小值（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.pop());//如果非空，且x大于s2栈顶，那么直接push栈顶
        else s2.push(x);//否则push x

    pop():O(1)
        s1.pop();
        s2.pop();

    getMin():O(1);
        return s2.top();

    方法3：s2没必要放一样多的元素
    push(x):
        s1.push(x);
        if(s2.empty()|| x<=s2.top()) s2.push(x);//这里为何要等于的时候也放，是因为下面我们看到，pop最小值的情况下要出栈。

    pop():
        if(s1.top() == s2.top()) s2.pop();//pop最小值的情况下，s2也pop
        s1.pop();
*/
```

来自 <http://tool.oschina.net/highlight>

例5 最大直方图



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）	

/*
https://leetcode.com/problems/largest-rectangle-in-histogram/description/
问题： 给出一个直方图，求最大面积矩形 （Leetcode 84）

分析：用堆栈分析每一块板能延伸到的左右边界（要求，这个延伸范围内的板高都大于等于当前），为让所有元素出栈，再人为的在最后加一块高为0的板

每块板i的高度h(i)入栈，高度比栈顶大入栈，该板的左边界定为之前栈顶元素的编号

高度比栈顶小，栈顶元素出栈，确定右边界为当前板号i。

左边界确定为当前板的下边板的编号（如果栈为空，则-1），有板比当前板小出栈，右边界确定为当前

让其出栈的板号

时间复杂度为0(n)

//这个思路太精妙了
*/

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        int n = heights.size(),res = 0;
        stack<int> s;
        for(int i = 0;i<n;++i)//遍历每块板
        {
            while(!s.empty()&&heights[s.top()]>=heights[i])//如果栈非空，且栈顶元素高于等于当前元素，右边界确定为i
            //原题这里>=和>都通过了，我猜是因为测试用例板高不重复，否则理论上应该是大于。
            {
                int h = heights[s.top()];
                s.pop();
                res = max(res,(i-1-(s.empty()?(-1):s.top()))*h);//右边界延伸到i-1，注意i是不满足高于之前板这个条件的。
            }
            s.push(i);//否则栈空or当前板高于之前板，直接入栈 板编号
        }
        while(!s.empty())//处理剩下的没出栈的，相当于在最后加了块高为0的板
        {
            int h = heights[s.top()];
            s.pop();
            res = max(res,(n-1-(s.empty()?(-1):s.top()))*h);//这些板的右边界都延伸到了n-1，这里s.top是栈顶元素的下一块板的编号，即左边界
        }
        return res;
    }
};
```

来自 <http://tool.oschina.net/highlight>

例6 滑动窗口最大值

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入队，b[0] = 5

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

```
/*
问题：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-(k-2).. i-(k-k)])
    若i<k-1,认为负数下标对应值是无穷小

分析：即分析以i之前k个元素范围到i 这个范围的最大值，作为b[i]处的值，

    方法1：用一个最大堆存放这k个数，时间复杂度为n*logk,因为n个元素，然后每出入一次堆都为logk的时间复杂度
    如果计算好了b[i-1]，那么如何计算b[i]呢？
    首先， a[i-k]（即a[i-1 - (k-1)出堆
    然后a[i]入堆
    然后b[i] = 堆顶即可

    但这个时间复杂度还不是最理想的，我们可以做到线性
    方法2：我们可以使用一个双端队列，
    如果队尾的数<=将要入队的数a[i],则扔掉队尾的数，将a[i]放入队尾
    否则，如果入队元素小于队尾，不仍队尾的，直接诶放入队尾即可
    这样一来，队列的元素从队头到队尾是单减的，队头永远是窗口最大值
    队头何时过期呢？i>队头元素索引+k-1 比如队头为a[0],k=3,i= 3时，就过期了
    时间复杂度O(n)
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    vector<int> solution(vector<int> a,int k)
    {
        int n = a.size();
        vector<int> b(n,0);
        deque<int> q;//双端队列，存的是下标
        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();//队尾太小
            q.push_back(i);//入队尾
            b[i] = a[q.front()];
        }
        return b;
    }
};


int main()
{
    int arr[6] = {4,2,7,6,5,8};
    vector<int> a(arr,arr+6);
    Solution so;
    vector<int> b = so.solution(a,3);
    for(vector<int>::iterator iter = b.begin();iter!=b.end();iter++)
        cout<<*iter<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

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