7，o(N)时间复杂度解决面试中遇到的问题上

2018年4月27日 19:49

- 简介
- 一些例题
  - 例1 名人问题 （社会名流问题）
  - 例2 Trapping in Rain Water
  - 例3 Container With Most Water
  - 例4 最大间隔问题
  - 例5 01相等的串
  - 例6 二进制矩阵最多1
- 总结
- 简介
- O(n)是什么
  - 注意n是什么
    - 规模 图:节点？边？
  - 扫一遍
  - 两头扫
  - 双重循环，但是内循环变量不减
  - 单调性
    - 队列
    - 堆栈

- 一些例题
  - 例1 名人问题 （社会名流问题）

```
/*
问题：输入为n×n的邻接矩阵。第i行第j列代表i是否认识j，(1表示认识,0表示不认识)，并且A认识B并不意味着B认识A,对角线上的元素不关心（自身）
    名人定义为他不认识任何人且所有人都认识他的人。请求出所有名人。

分析：最多有几个名人呢？只有1个，因为有两个的话，它们互相认识不认识？
```

笨方法：遍历i，检查每个j，看是否满足i不认识j但j认识i，时复为n^2

O(n)的方法：对于两个人i，j，如果i认识j，那么i明显不是名人，删掉i
如果i不认识j，则j也显然不是名人，删掉j。
最终剩余1个人，检查其是否为名人即可

实现1：用一个数组保存所有没检查人的编号。数组如何删除a[i]?不保证顺序的时候，只需让a[i] = a[--n]即可。
为什么呢?是因为此时，将数组结尾的元素放到了需要删除的位置，同时数组的长度也-1了

实现2：上面的空间也为N，那么可以将其优化为常数空间么。一头扫,ij一起扫，保证i<j，并且0-i-1没有名人，i-j-1也是没有名人的
如果i认识j，那么i肯定不是名人，删掉,令i=j,j = j+1
如果i不认识j，那么j肯定不是，j = j+1.
i，j始终为可能是名人的两人

实现3,两头扫的方法，让i = 0,j=n-1,i<j 则0-i-1是没名人，j+1,n-1是没有名人的。
如果i认识j，则删掉i，i++。
如果i认识j，则j--

```
*/
#include<bits/stdc++.h>
using namespace std;


class Solution1
{
public:
    int findP(int know[][4],int k)
    {
        int n = k;
        int a[n];//没检查过的人
        for(int i =0;i<n;i++)
            a[i] = i;
        while(n>1)//直到没检查的人只有1个了
        {
            if(know[a[0]][a[1]])//0位置上的人知道1位置上的人，那么0不是，删除
                a[0] = a[--n];//注意，这里应该是--n
            else//0位置不知道1位置，删除1
                a[1] = a[--n];
        }


        for(int i = 0;i<k;i++)//检查最后一个元素是否是名人，这里不应该是n了，应该是know.size()
        {
            if((i!=a[0])&&(know[a[0]][i] || !know[i][a[0]]))//非a[0]的人，一旦a[0]关注了，or 没有关注a[0]，return-1即可
                return -1;
        }
```

```
        return a[0];//名人
    }
};



class Solution2
{
public:
    int findP(int know[][4],int k)
    {
        int i = 0,j =1;//维持两个指针，j始终比i大
        for(;j<k;j++)//遍历j
        {
            if(know[i][j])//如果i认识j，i肯定不是名人，令i = j，j继续++
                i = j;
            //这里隐含了一个判断，else，如果i不认识j，那么j肯定不是名人，j = j+1. 相当于j之前的都肯定不是名人，
            //因此上面才能直接让i = j
        }
        for(j = 0;j<k;j++)//i为可能的名人，遍历其余，判断其是否是真的名人
        {
            if((j!=i)&&(know[i][j] || !know[j][i]))
                return -1;
        }
        return i;
    }
};


class Solution3
{
public:
    int findP(int know[][4],int n)
    {
        int i = 0,j = n-1;
        while(i<j)
        {
            if(know[i][j])//i知道j，i++
                i++;
            else
                j--;
        }
        for(j = 0;j<n;j++)//i为可能的名人，遍历其余，判断其是否是真的名人
        {
            if((j!=i)&&(know[i][j] || !know[j][i]))
                return -1;
        }
        return i;
    }
};
int main()
{
    int k[4][4] ={
```

```
            {1,0,0,0},
            {1,1,0,0},
            {1,0,1,0},
            {1,0,0,1}
        };
        //vector<vector<int> > know(k,k+4);
        Solution3 s;
        cout<<s.findP(k,4)<<endl;
        return 0;
    }
```

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

- 例2 Trapping in Rain Water
- 例2 Leetcode 42 给定每个块高度，求下雨后积水。图对应[0,1,0,2,1,0,1,3,2,1,2,1]


```
/*
问题：  Leetcode 42 给定每个块高度，求下雨后积水。 图对应[0,1,0,2,1,0,1,3,2,1,2,1]
https://leetcode.com/problems/trapping-rain-water/description/

分析：每一块积水高度（水高和原高加起来），其实等于它左边所有包括本身的最大值，和右边所有（包括本身）的最大值，里较小的。

    思路：利用前缀和后缀

*/

class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        int result = 0;
        vector<int> left(n),right(n);
        for(int i = 0;i<n;i++)//计算left,即包含自身在内，左边元素的最大值
            left[i] = i?max(height[i],left[i-1]):height[i];//i为0时就等于height[0]
        for(int i = n-1;i>=0;i--)//计算right，包含自身在内，右边元素的最大值
            right[i] = (i == n-1)?height[i]:max(height[i],right[i+1]);
```

```
        for(int i = 0;i<n;i++)
            result+=min(left[i],right[i]) - height[i];//取小的那个，再减去原高
        return result;
    }
};
```

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

- ## 例3 Container With Most Water

```
/*
问题：Leetcode 11 一个数组a[i]表示数轴上i的位置有一条高度为a[i]的竖直的线段，把两条线段当作一个容器左右边的高度，
问那两条线段组成的容器容积最大？
https://leetcode.com/problems/container-with-most-water/description/

分析：本质上是求max{min{a[i],a[j]}*(j-i)}, i<j
    思路:两头扫，i = 0,j = n-1,best = 0
    i<j
        best =max(best,min(a[i],a[j])*(j-i)
        if(a[i]<a[j]) i++;//i是较小的哪个，i++
        else j--;//j较小，j--

*/
class Solution {
public:
    int maxArea(vector<int>& height) {
        int i = 0,j =height.size()-1,best = 0;
        while(i<j)
        {
            best = max(best,min(height[i],height[j])*(j-i));//更新best
            if(height[i]<height[j]) i++;//更新索引，a[i]处小，i++
            else j--;//否则，j--
        }
        return best;
    }
};
```

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

- ## 例4 最大间隔问题

```
/*
问题： 给定数组a，求下标对儿（i, j）满足a[i] ≤ a[j]，并且j - i最大。

分析：假设目前最优解为d，那么对于j，则只需要检查i= j-d-1，即检查最优解范围外的部分，看会不会更优即可

    首先，我们记录下前缀最小值p[x] = min{p[0,..,x]}
    倒着循环j，对每个j看下a[j-d-1]是否<=a[j],如何判断?用p[j-d-1]是否<=a[j]，若是，肯定还能
```

往前移。否则，不用移了，前面所有的都比a[j]要大

建议画图分析
*/
```
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    int run(vector<int> a)
    {
        int n = a.size();
        int p[n];
        for(int i = 0;i<n;i++)
            p[i] = i?min(a[i],p[i-1]):a[i];
        int best = 0;
        for(int j = n-1;j>best;j--)//只需倒序检查最优解范围外的部分
        {
            while((j>best)&&(a[j]>=p[j-best-1]))    best++;//p指导了我们还能否再增大间隙best，能就一直增，不能就换j再试试
        }
        return best;
    }
};
int main()
{
    int a[7] = {1,2,4,5,7,2,2};
    vector<int> arr(a,a+7);
    Solution s;
    cout<<s.run(arr)<<endl;
    return 0;
}
```

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

-
  - 例5 01相等的串

```
/*
```
问题：给定一个01串，求它一个最长的子串满足0和1的个数相等。

分析：我们将0看作-1,1看作+1,统计一个前缀和。
若两个前置和相等，那么这两个前缀和之间的子串满足01个数相等。

若前缀和算出之后对其进行排序，那么需要n×logn

而这里不需要排序，
首先，前缀和的范围为-n,n,我们加上n将其变成0,2n. 然后只需要记录这个前缀和第一次出现的位置即可

本质：用hash代替排序

```
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int run(string s)
    {
        int n = s.size();
        vector<int> hash(n*2+1,-1);//下标从0-2n.记录第一次出现某前缀和的位置
        hash[n] = 0;//表示长度为n（-n） = 0出现的位置，最开始在0出现
        int sum = n;//统计目前为止的前缀和，初始化为0（+n）即n
        int best = 0;
        for(int i = 0;i<n;i++)
        {
            sum+=(s[i] == '0')?(-1):1;//统计目前位置的前缀和
            if(hash[sum]>=0)//表示该前缀和已出现过，当前位置减去第一次出现该前缀和的位置
                best = max(best,i+1-hash[sum]);
            else//还没出现过，记录该前缀和值第一次出现的位置
                hash[sum] = i+1;//存的是位置，而非下标

        }
        return best;

    }
};

int main()
{
    string s = "0101010110";
    Solution so;
    cout<<so.run(s)<<endl;
    return 0;
}
```

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

○ 例6 二进制矩阵最多1

/*
问题：给定n * n的01方阵，每一行都是降序的（即先连续的一段1，再连续的一段0），求1最多的那行中1的个数？

分析：我们可以二分出每一行01的分界线，这时时间复杂度为nlogn

优化：如果某个位置上为1,则往右，否则往下，(我们只有找到比该行更多的1才有意义),时间复杂度为N

```
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solve(vector<vector<int> > a)
    {
        int n = a.size();
        int best = 0;//每行1最多多少
        for(int i = 0;best<n&&i<n;i++)
        {

            while(best<n && a[i][best] == 1) ++best;

            //否则i++，统计下一行，若best处是1,继续往后统计，否则继续i++
        }
        return best;
    }
};
int main()
{
    int a[4][4]={
    {1,0,0,0},
    {1,0,0,0},
    {1,1,0,0},
    {1,0,0,0}
    };
    vector<vector<int> > arr(4,vector<int>(4));
    for(int i = 0;i<4;i++)
    {
        for(int j = 0;j<4;j++)
            arr[i][j] = a[i][j];
    }
    Solution s;
    cout<<s.solve(arr)<<endl;
    return 0;
}
```

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

- 总结
- 其他问题和算法
  - 最大子数组和
  - KMP（extend)

- Manacher
- 最大直方图（单调堆栈)
- 滑动窗口最大值（单调队列）
- 快排Partition过程
- 杨氏矩阵查找
  - 荷兰国旗问题
  - First Missing Positive
- 排列组合相关
  - Next/Previous permutation
- 树相关
  - 二叉树遍历、(最大、最小)深度、同构、镜像判断、平衡判断

- 多思考，多练习