O(n)时间解决的面试题（上）

七月算法　曹鹏

2015年5月9日

提纲

- 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

例1 名人问题

- 例1 有n个人他们之间认识与否用邻接矩阵表示 (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,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？只有1个！（有两个的话，他们认识不认识？）
  - $O(n^2)$的笨方法，遍历i，检查每个j，是否满足i不认识j且j认识i。

例1 续

- O(n)的方法
  - 对于两个人i和j
    - 如果i认识j, 则i显然不是名人，删掉i
    - 如果i不认识j，则j显然不是名人，删掉j
  - 最终剩余一个人，检查他是否是名人
- 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 只要a[i] = a[--n]即可

例1 续2

- 伪代码：时间O(n)，空间O(n)

```
for (int i = 0; i < n; ++i) {
        a[i] = i;
}
while (n > 1) {
        if (known[a[0]][a[1]]) {
                a[0] = a[--n];
        }
        else {
                a[1] = a[--n];
        }
}
for (int i = 0; i < n; ++i) {
        if ((a[0] != i) && (known[a[0]][i] || !known[i][a[0]])) {
                return -1;
        }
}
return a[0];
```

例1 续3

- 实现2 优化、优化、再优化
  - 能否O(1)空间？
  - “一头扫”
    - i < j
    - [0..i – 1]没有名人
    - [i + 1..j – 1]没有名人
    - 如果i认识j, 删掉i
      - i = j, j = j + 1
    - 如果i不认识j,删掉j
      - j = j + 1

```
int i = 0, j = 1;
for (; j < n; ++j) {
        if (known[i][j]) {
                i = j;
        }
}
for (j = 0; j < n; ++j) {
        if ((i != j) & ((known[i][j] || !known[j][i])) {
                return -1;
        }
}
return i;
```

例1 续4

- 实现3 “两头扫”
  - i = 0， j = n – 1
  - i < j
    - [0..i – 1]没有名人
    - [j + 1..n]没有名人
    - 如果i认识j，删掉i
      - ++i
    - 如果i不认识j,删掉 j
      - --j

```
int i = 0, j = n - 1;
while (i < j) {
        if (known[i][j]) {
                ++i;
        }
        else {
                --j;
        }
}
for (j = 0; j < n; ++j) {
        if ((i != j) & ((known[i][j] || !known[j][i])) {
                return -1;
        }
}
return i;
```

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]
  - 分析：每一块和水高度等于它左面（包括本身）的最大值和右边（包括本身）的最大值里较小的

例2 续

- 利用“前缀”和“后缀”

```
class Solution {
public:
    int trap(vector<int> &A) {
        int n = A.size();
        int result = 0;
        vector<int> left(n),right(n);
        for (int i = 0; i < n; ++i) {
            left[i] = i?max(left[i - 1], A[i]):A[i];
        }
        for (int i = n - 1; i >= 0; --i) {
            right[i] = (i == n - 1)?A[i]:max(right[i + 1], A[i]);
        }
        for (int i = 0; i < n; ++i) {
            result += min(left[i], right[i]) - A[i];
        }
        return result;
    }
};
```

例3 Container With Most Water

- 例3 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？
  - 本质是求 i < j, max{min{a[i], a[j]} * (j – i)}
  - 算法两头扫：
    - 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; else --j;

例3 续1

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        int best = 0;
        int n = height.size();
        for (int i = 0, j = n - 1; i < j;) {
            best = max(best, min(height[i], height[j]) * (j - i));
            if (height[i] < height[j]) {
                ++i;
            }
            else {
                --j;
            }
        }
        return best;
    }
};
```

例3 续2

- 证明：算法一定扫过最优解
  - 关键：如果一边移动到了最优解，另一边还没到最优解，没到的那边高度一定比最优解中较低的边低！（道理：因为x轴上宽度更宽）
    - 无论高或者低的那边先到最优解，根据我们的“关键点”，另外那边一定比它还要低，算法会一直移动另外那边到最优解，而高的这边保持不动。

例4 最大间隔问题

- 例4 给定数组a，求下标对i, j满足a[i] ≤ a[j]，并且j – i最大。
- 分析：
  - 假设目前最优解是d, 对于j，至少要检查i = j – d – 1才可能更优
  - 记录前缀最小值p[x] = min{a[0..x]}
  - 倒着循环j,对于每个j看一下p[j – d – 1]是否<= a[j]，用p“引导”
  - 如果前面都比a[j]大，则这个j得不到更优的解

例4 续

- 对best的理解

```
int  run(vector<int> &a) {
        int n = a.size();
        vector<int> p(n);
        for (int i = 0; i < n; ++i) {
                p[i] = ((i == 0) || (a[i] < p[i - 1]))?a[i]:p[i - 1];
        }
        int best = 0;
        for (int j = n - 1; j > best; --j) {
                while ((j > best) && (a[j] >= p[j - best - 1])) {
                        ++best;
                }
        }
        return best;
}
```

例5 01相等的串

- 例5 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：把0看成-1，1当作+1，还记得“前缀和”么？
  - 需要两个前缀和相等，则这两个前缀和之间的子串满足0的个数和1的个数相等。
  - 对前缀和排序？ O(nlogn)
  - 优化——不需要排序
    - 前缀和范围是[-n..n]，我们加上n之后就是[0..2n]，只要记录第一次出现的位置

例5 续

本质：
用hash代替排序。
而当hash值是比较小的非负整数时，可以用做数组下标

```
int run(char *s) {
        int n = strlen(s);
        vector<int> have((n << 1) | 1, -1);
        have[n] = 0;
        int sum = n;
        int best = 0;
        for (int i = 0; i < n; ++i) {
                sum += (s[i] == '0')?(-1):(+1);
                if (have[sum] >= 0) {
                        best = max(best, i - have[sum] + 1);
                }
                else {
                        have[sum] = i + 1;
                }
        }
        return best;
}
```

例6 二进制矩阵中1的个数

- 例6 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：
    - 算法1 数出每一行的1…… 复杂度$O(n^2)$
    - 算法2 二分出每一行0和1的分界线 复杂度 O(nlogn)
    - 算法3
      - 如果某个位置时1，则向右，是0则向下 （我们只需要找到比本行更多的1才有意义！）

例6 续

```
int run(vector<vector<char> &a) {
int n = a.size();
int best = 0;
        for (int i = 0; (best < n) && (i < n); ++i) {
                while ((best < n) && (a[i][best] == '1')) {
                        ++best;
                }
        }
        return best;
}
```

时间复杂度O(n)

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