Tsinghua University

Homework #4

Student name:

Course: 算法分析与设计 – Professor: 王振波

Due date: *2020* 年 *10* 月 *13* 日

Exercises 4.8,4.18

4.8 Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

假设 T, T' 均为最小生成树。那么一定存在边 $e \in T - T'$，在图 T 中去掉 e 之后变成了两个不相交的连通图，其顶点分别为 S_1, S_2，则在 T' 中 S_1, S_2 之间存在一条边 e'，且 $e \neq e'$，但是由最小生成树的割性质即可得出矛盾。 ■

4.18 Your friends want to use the Web site to determine the fastest way to travel through the directed graph from their starting point to their intended destination. (You should assume that they start at time 0, and that all predictions made by the Web site are completely correct.) Give a polynomial-time algorithm to do this, where we treat a single query to the Web site (based on a specific edge e and a time t) as taking a single computational step.

设起点为 s，终点为 f，我们如下进行操作：

1: 初始化：设 V 为除去 s 的所有顶点，S 为标记点集合。$S = \{s\}$，对每个顶点 v，我们存储一个时间 $t(v)$ 和前一点 $pre(v)$。$t(s) = 0, pre(s) = \phi$
2: **while** $f \notin S$ **do**
3: 选择 V 中与 S 相连的节点 v 使得 $t'(v) = min_{e=(u,v):u \in S} f_e(t(u))$ 最小
4: 把 v 加入到 S 中，并且从 V 中删除。存储 $t(v) = t'(v), pre(v) = u$(u 是上式取最小值的 u)。
5: **end while**

类似于 Dijkstra 算法，我们可知此算法为 $O(m \log n)$，其中 m 为边数，n 为节点数。我们证明对每个 v，算法产生的 $t(v)$ 都是最短时间：

反之，存在某点 y，使得 $s = a_0 \to a_1 \to \cdots \to a_m = y$ 花费的时间 $t < t(y)$。我们不妨设 y 是所有这些点中第一个加入到 S 中的 (那么在 y 之前加入到 S 的点均是最快路径)。若在加入 y 时，$a_1, \cdots, a_{m-1}$ 都在 S 中，那么我们算法的 $t(y) \le t$，矛盾；若在加入 y 时，$a_1, \cdots, a_i$ 在 S 中，a_{i+1} 不在 S 中 (i 可以为 0)，那么可知 $t \ge f_{(a_i,a_{i+1})}(t(a_i)) \ge f_{(pre(y),y)}(t(pre(y))$，那么可见我们算法产生了一条用时更短的路径，这与假设矛盾！ ∎

5.1 You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values-so there are $2n$ values total-and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the n^{th} smallest value.

However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k^{th} smallest value that it contains. since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(\log n)$ queries.

我们想办法把原问题转化成更小规模的问题。

设这两个数据库分别为 $A(i), B(i), i = 1 : n$，其中 $A(1) \le A(2) \le \cdots \le A(n), B(1) \le B(2) \le \cdots \le B(n)$ 我们首先在两个数据库中询问 $k = \lceil \frac{n}{2} \rceil$，得到 $A(k), B(k)$：若 $A(k) < B(k)$(反之完全类似)，我们可知 $B(k)$ 比 A 中前 k 个数大，也比 B 中前 $k-1$ 个数大，则 $B(k)$ 至少是第 $k + k = 2\lceil \frac{n}{2} \rceil \ge n$ 小的数，所以第 n 小的数应该在 $B(k)$ 之前 (含)；而 $A(k)$ 比 A 中 $k-1$ 个数大，且最多在 B 中比 $k-1$ 个数大，那么 $A(k)$ 至多是第 $k + k - 1 = 2k - 1 \le n$ 小的数，所以第 n 小的数应该在 $A(k)$ 之后 (含)。

为了使剩下的两个子列 $A_1 = \{A(k), \cdots, A(n)\}, B_1 = \{B(1), \cdots, B(k)\}$ 个数相等，我们可以注意到当 n 为奇数时，已然相等；当 n 为偶数时，由于 $2k - 1 < n$，所以一定不包含 $A(k)$，我们可以去掉 A_1 中的 $A(k)$，于是可以令 $A_1 = \{A(\lceil \frac{n+1}{2} \rceil), \cdots, A(n)\}, B_1 = \{B(1), \cdots, B(k)\}$，容易发现 A, B 的中位数也是 A_1, B_1 的中位数。于是问题的规模减小了一半。

我们给出如下递归算法 1，其中参数 x, y, n 代表我们在序列 $A(x+1), \cdots, A(x+n); B(y+1), \cdots, B(y+n)$ 中寻找中位数。

根据算法可知：每次递归规模减半，而且每次递归需要两次询问和一次比较，所以总的运行时间为 $O(\log n)$ ∎

```
1: function MEDIAN(x, y, n)
2:     if n = 1 then return min{A(x + 1), B(y + 1)}
3:     end if
4:     k = ⌈n/2⌉
5:     if A(k + n) < B(k + n) then
6:         return MEDIAN(x + ⌈(n−1)/2⌉, y, k)
7:     else
8:         return MEDIAN(x, y + ⌈(n−1)/2⌉, k)
9:     end if
10: end function
```

Algorithm 1: median(x,y,n)

5.2 Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a significant inversion if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.

事实上，这跟传统的计算逆序数的算法只多了一个步骤，那就是在计数时略有差别。我们首先将序列分为两半，这样就变成了两个子序列和他们之间的强逆序数的计数，而他们之间的强逆序数和前一个子列和后一个子列的两倍之间的逆序数相同，所以我们只需要对逆序数的算法稍作修改 (算法 2)：

函数 merge-and-count(A, B) 把两个排好序的列表排序成新列表 C，并计算了强逆序数；主函数 Sort-and-Count(L) 采用分而治之的方法进行递归，并返回了强逆序数和排好序的列表 L。可以看出该算法与归并算法同阶，均为 $O(n \log n)$ ∎

```
function MERGE-AND-COUNT(A, B)
    初始化 count = 0, i = j = 1, n = |A| = |B|, C = Φ
    while i ≤ n and j ≤ n do      //排序
        if a_i ≤ b_j then
            把 a_i 加入到链表 C 的末尾，i = i + 1
        else
            把 b_j 加入到链表 C 的末尾，j = j + 1
        end if
    end while
    把 i, j 中小于等于 n 的对应 A 或 B 取出，将其 j 号元素及其之后的加入到 C 中；
    再次初始化：i = j = 1
    while i ≤ n and j ≤ n do      //计数
        if a_i ≤ 2b_j then
            i = i + 1
        else
            j = j + 1, count = count + n − i + 1
        end if
    end while
    return count, C
end function

function SORT-AND-COUNT(L)
    if L 中只有一个元素 then return r = 0, L
    else
        把 L 分为两个部分 A, B，其中 A 包含前 ⌈|L|/2⌉ 个元素，B 包含后 ⌊|L|/2⌋ 个元素
        (r_A, A)=SORT-AND-COUNT(A)
        (r_B, B)=SORT-AND-COUNT(B)
        (r, L)=MERGE-AND-COUNT(A,B)
        return r = r + r_A + r_B, L
    end if
end function
```

Algorithm 2: 强逆序数算法