算法与复杂性 第三次课程作业

UNIkeEN

2023 年 3 月 29 日

问题解答

1. 证明二分查找是有序序列查找的最优算法

> **解** 有序序列查找，输出为元素在序列中的下标，可能的输出结果有 n 种。从信息论的角度，将这 n 种看作决策树的叶子结点，可知决策树高度不超过 $\log_2 n$，即 $\Omega(\log n)$
>
> 可知有序序列查找的时间复杂度不可能优于 $\Omega(\log n)$。二分查找的时间复杂度是 $O(\log n)$，可知其是有序序列查找的最优算法。

2. 对玻璃瓶做强度测试，设地面高度为 0，从 0 往上有刻度 1 到 n，相邻两个刻度间距离都相等，如果一个玻璃瓶从刻度 i 落到地面没有破碎，而在高度 i+1 处落到地面时碎了，则此类玻璃瓶的强度为 i。1）若只有一个样品供测试，如何得到该类玻璃瓶的强度；2）如果样品数量充足，如何用尽量少的测试次数得到强度；3）如果有两个样品，如何用尽量少的测试次数得到强度；4）如果有 k 个样品呢。

> **解**
>
> 1) 若只有一个样品供测试，则从高度 1 到高度 n 逐一进行摔落测试，当在高度 i 处摔碎时停止，得到强度为 $i-1$；若从高度 n 处落地仍未破碎，可知玻璃瓶强度大于等于 n。
>
> 2) 样品数量充足时，使用二分法测试。首先测试高度 $\lfloor \frac{n}{2} \rfloor$，若摔碎则测试 $\lfloor \frac{n}{4} \rfloor$，若未摔碎则测试 $\lfloor \frac{3n}{4} \rfloor$，以此类推
>
> 3) 若只有两个样品，则可将 1 到 n 划分成 $\lfloor \sqrt{n} \rfloor$ 个区间（第 k 个区间包含 $(k-1)\lfloor \sqrt{n} \rfloor \sim k\lfloor \sqrt{n} \rfloor$，最后一个区间包含 $(\lfloor \sqrt{n} \rfloor - 1)\lfloor \sqrt{n} \rfloor \sim n$）
>
> 第一个样品对各个区间的上限测试，以确定强度所在区间；第二个样品在确定的区间中由低到高逐高度测试，确定强度。时间复杂度为 $O(\sqrt{n})$。
>
> 4) 可将 1 到 n 划分为 $\sqrt[k]{n}$ 个区间，用第一个样品确定子区间后将子区间再划分为 $\sqrt[k]{n}$ 个区间，直至剩余 1 个样品时，对此时的子区间由低到高逐高度测试。时间复杂度为 $O(\sqrt[k]{n})$

3. 有 n 个数存放在两个有序数组中，如何找到这 n 个数的第 k 小的数

解 使用归并法，假设两个有序数组是递增排序。初始时两个指针指向数组头部，所指向数较小者，提取该数并后移一位（若所指数相同则任意移动一个指针，若某个指针已达数组末尾则移动另一指针）。移动 k 次后得到的数即为第 k 小的数。

若两个有序数组是递减排序，则指针可从后往前移动 k 次，也可从前往后移动 n-k 次，算法的复杂度为 $O(k)$。

Algorithm 1 归并取第 k 小的数

输入: 数组 $A[m], B[n]$；目标 k

输出: A, B 中第 k 小的数 ans

```
t ← 0, i ← 1, j ← 1
while t < k ∧ i ≤ m ∧ j ≤ n do
    if A[i] <= B[j] then
        ans ← A[i], i ← i + 1
    else
        ans ← B[j], j ← j + 1
    end if
    t ← t + 1
end while
while t < k ∧ i < m do
    ans ← A[i], i ← i + 1
    t ← t + 1
end while
while t < k ∧ j < n do
    ans ← B[j], j ← j + 1
    t ← t + 1
end while
```

4. 如何修改 KMP 算法，使之能够获得字符串 B 在字符串 A 中出现的次数

解 设置计数器 cnt，初始值为 0。

假设在字符串 A 中由 i 指示位置，在模式串 B（长度为 m）中由 j 指示位置。在 j 指到模式串末尾并完成一次匹配后，cnt、i 增加 1，j 回到 $next(m)+1$ 处继续向后匹配。

5. 序列 T 和 P 的最长公共子序列（LCS）L 定义为 T 和 P 的共同子序列中最长的一个，而最短公共超序列 (SCS)S 定义为所有以 T 和 P 为子序列的序列中最短的一个。设计高效算法求序列 T 和 P 的 LCS 和 SCS 的长度

解 子序列不同于子串，子序列不需要在原序列中占用连续的位置。基于这个思想，可以使用动态规划求解 LCS。

LCS(i, j) 为 T 前 i 位元素组成的序列与 P 前 j 位元素组成的序列的最长公共子序列，有

状态转移方程：

$$\mathrm{LCS}(i,j)=\begin{cases}\mathrm{LCS}(i-1,j-1)+1, & \mathrm{T}[i]=\mathrm{P}[j]\\ \max\{\mathrm{LCS}(i-1,j),\mathrm{LCS}(i,j-1)\}, & \mathrm{T}[i]\neq\mathrm{P}[j]\end{cases}$$

求最短公共超序列 SCS 的过程如下，设已知 T 和 P 的（LCS）L 为 Q，从 P 中删去 Q，将 P 的剩余部分插入 T，保证插入元素的相对位置不变，且其原本对于 Q 的相对位置对于 T 中的 Q 保持不变，插入后得到的新序列即为所求 SCS，该算法的时间复杂度为 $O(|T\times P|)$。

Algorithm 2 求最长公共子序列 LCS

输入: $T[m], P[n]$，假设 $m \geq n$
输出: l（T, P 的最长公共子序列的长度）

```
dp[0...1][0...n] ← {0}
for i ← 1 to m do
    for j ← 1 to n do
        if T[i] = P[j] then
            dp[i mod 2][j] ← dp[(i − 1) mod 2][j − 1] + 1
        else if dp[i mod 2][j − 1] ≥ dp[(i − 1) mod 2][j] then
            dp[i mod 2][j] ← dp[i mod 2][j − 1]
        else
            dp[i mod 2][j] ← dp[(i − 1) mod 2][j]
        end if
    end for
end for
return dp[m mod 2][n]
```

6. 有 $n=2^k$ 位选手参加一项单循环比赛，即每位选手都要与其他 $n-1$ 位选手比赛，且在 $n-1$ 天内每人每天进行一场比赛，1）设计算法以得到赛程安排；2）若比赛结果存放在一矩阵中，针对该矩阵设计 O(nlogn) 的算法，为各个选手赋予次序 P1,P2,⋯,Pn，使得 P1 打败 P2，P2 打败 P3，依此类推。

解

1) 本题可以使用递推法。考虑一个 $n\times n$ 的赛程矩阵，第 i 行 j 列的数字表示选手 i 在第 j 天的对手编号，要求即将 1 到 n 的数字填入矩阵，但每行和每列的元素不得重复。

考虑 $k=1$ 即只有 2 位选手的情况，矩阵可以是：

1	2
2	1

正式的比赛日从第 2 天（第二列）算起，即第 2 天选手 1 和 2 进行比赛。

$k=2$ 时，将矩阵划分为 4 块，左上和右下原样填入 $k=1$ 时的矩阵；左下和右上也填入 $k=1$ 的矩阵，但每个元素增加 2，如下：

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

$k = 3$ 时，将矩阵划分为 4 块，左上和右下原样填入 $k = 2$ 时的矩阵；左下和右上也填入 $k = 2$ 的矩阵，但每个元素增加 4，如下：

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

重复上述过程，将 k 规模即选手数为 2^k 的赛程矩阵分为四块，左上和右下原样填入 $k-1$ 规模的矩阵，左下和右上也填入 $k-1$ 规模的矩阵，但每个元素增加 2^{k-1}。

最后得到的矩阵即为赛程安排，从第 2 列起作为正式比赛日，第 i 行 j 列的数字表示选手 i 在第 $j-1$ 比赛日的对手编号。

2) 对于任意两个选手 i, j，定义若 i 赢了 j 则 i < j。任意两个人之间都可以按上述定义比较大小，该大小关系可以通过查询比赛结果的矩阵（可为二维数组）得到。使用一种基于比较的排序算法（如快速排序、归并排序），即可在 $O(n \log n)$ 的时间复杂度下完成排序，最后按照排序对选手进行编号即可。

7. G=(V,E) 是一个无向图，每个顶点的度数都为偶数，设计线性时间算法，给 G 中每条边一个方向，使每个顶点的入度等于出度。（请先简单说明算法思想，再给出伪代码，然后证明其时间复杂性符合要求）

解 无向图每个顶点的度数都为偶数，则该图一定存在欧拉回路，能经过每一条边且每条边仅经过一次。

使用 DFS 遍历所有的边，易得最终的遍历顺序即欧拉回路，沿该顺序给边添加方向即可。

8. 连连看游戏中用户可以把两个相同的图用线连到一起，如果连线拐的弯小于等于两个则表示可以消去。设计算法，判断指定的两个图形能否消去

解 假设当前连连看的地图状态使用矩阵存储，由于连线可以位于地图外，在矩阵外围增加一圈 padding，元素全部设为空。使用小于等于两次拐弯的线对目标点进行连接，一共有三种情况：直线、单次拐弯、两次拐弯。

1. 以目标两点（下图中黄色）为中心，寻找其上下左右的最大十字形空白区域（下图中蓝色区域）。
2. 如果目标两点处于同一行或同一列，首先尝试用直线连接，检查两点所连直线上是否全部为空（没有图案），如是则可以消去，算法结束，如否则进入步骤 4。（此时不可能只通过一次拐弯连接）
3. 如果目标两点不处于同行同列，首先尝试用一次拐弯连接，检查步骤 1 中生成的两处十字形空白区域是否相交，如是则可以消去，算法结束，如否则进入步骤 4。
4. 尝试用两次拐弯连接，寻找步骤 2 中生成的两处十字形空白区域，是否有处于同一行或同一列的部分，若有且这两部分所连直线上全部为空元素，则可以消去，如否则无法相连消去，算法结束。

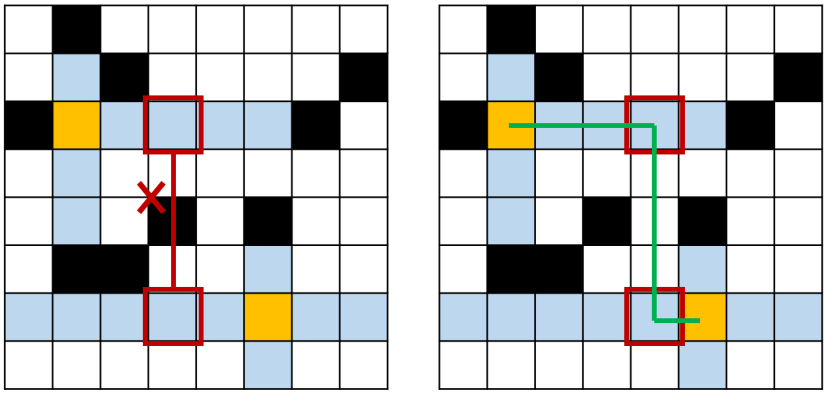

图 1: 两次拐弯判定示意图

9. 证明任意连通无向图中必然存在一个点，删除该点不影响图的连通性。用线性时间找到这个点。

解

- 证明：在无向图 G 中找到一条长度最长的简单路径 P。设路径 P 的两个端点为 u 和 v，删除 u 或 v 之一，不会影响图的连通性。

  由于路径 P 是最长的，所以在图 G 中，顶点 u 和 v 不可能有其他的相邻顶点属于路径 P。对于任意不在路径 P 中的顶点，它与路径 P 中的其他顶点的连接不会受到顶点 u（或 v）被删除的影响。而路径 P 中除 u（或 v）之外的其他顶点之间仍然存在路径连接。所以，子图仍然是连通的。

- 使用线性时间寻找此点，可以使用 DFS 的方法，一定存在搜索到某个结点时，该结点的所有相邻结点都已经被遍历过了，删除该结点一定不影响图的连通性，因为这一结点的所有相邻结点都可通过不经过此结点的其他路径与根结点连接。

  进行深度优先搜索的时间复杂度为 O(|V|+|E|)，是线性时间。

10. 给定一个连通的无向图和图中的一个顶点 v，构造一个有向图，使得有向图中的任何路径都通向

该特定顶点，设计其算法。

解 以顶点 v 为根结点，使用 DFS 或 BFS 对无向图 G 进行遍历，假设遍历过程中由点 u 遍历到 w，则在有向图 G' 中添加边 (w,u)，直至无向图 G 中的所有结点被遍历。

（题目示例图中不需要将无向图的所有边添加方向，如 (3,4) 和 (4,5)，如需给所有边加上方向，则在遍历时为点和边都进行标记，当某时刻遍历经过的边终点已被标记时，这条边可以随意地添加方向）