算法设计与分析第三次作业

杨博文 21373037

2023 年 11 月 30 日

1. 分蛋糕问题

给定 n 块体积不同的蛋糕，其体积分别用 a1，$\cdots$, an 表示。现要从中挑选出 k(k < n) 块蛋糕分给同学们。不妨记选出的蛋糕的编号为 $s_1, \cdots, s_k (1 \leq s_1 < \cdots < s_k \leq n)$，则这次分配的不公平度为

$$\max(a_{s1}, \cdots, a_{sk}) - \min(a_{s1}, \cdots, a_{sk})$$

请设计一个尽可能高效的算法制定蛋糕的选取方案，使得选出蛋糕的不公平度最小，请描述算法的核心思想，必要时给出证明，给出算法伪代码并分析其对应的时间复杂度。

1. 核心思想

首先将蛋糕的体积由小到大排序，排序后的数组称为 $a^{'}, s^{'}$(排序后的下标到原下标的映射关系)，则我们连续的在排好序的数组中取 k 个值 $a_i^{'}, a_{i+1}^{'}, ...a_{i+k-1}^{'}$，则不公平度为 $a_{i+k-1}^{'} - a_i^{'}$，我们只需遍历数组找到所有长度为 k 的子串的最小不公平度即可。

2. 简单证明

我们证明不公平度最小的情况一定出现在排好序的长度为 k 的子串中。

如果我们选择相邻的 k 个，那么这 k 个蛋糕的体积一定是连续的一段，这样可以确保最大值和最小值之差最小。

使用反证法，如果我们选择不相邻的 k 个，记为 $a_{s1}, ..., a_{sk} (s_1 < s_2 ... < s_k)$，我们固定其下界 a_{s1}，将其他元素聚拢，得到 $a_{s1}, a_{s_1+1}, ..., a_{s_1+k-1}$，由于有序性，显然后者的最大最小值之差更小，因此任何一种不相邻的情况都可以优化为相邻的情况。

3. 伪代码

Algorithm 1 分蛋糕问题

Input: $a[1...n], k$

Output: 下标集合 $subscript[1..k]$

对体积数组 $a[n]$ 由小到大进行排序并记录原数组下标，得到 $a^{'}, s^{'}$。

$min \leftarrow \infty, start \leftarrow 0$

for $i : 1 \rightarrow n - k + 1$ **do**

  if $a^{'}[i + k - 1] - a^{'}[i] < min$ **then**

    $min \leftarrow a^{'}[i + k - 1] - a^{'}[i], start \leftarrow i$

  end

end

for $i : start \rightarrow start + k - 1$ **do**

  $subscript.append(s^{'}[i])$ 加入进下标集合

end

4. 时间复杂度分析

数组排序时间复杂度为 $O(nlogn)$。遍历数组求最小值时间复杂度为 $O(n)$，该算法时间复杂度为 $O(nlogn)$。

2. 分糖果问题

F 同学手中有 n 袋糖果，其中第 i 袋中有 a_i 颗糖。F 同学希望在不拆开糖果袋的前提下把一些糖果分给他的妹妹，但是他希望分完之后留给自己的糖果总数要大于妹妹得到的糖果总数。请设计一个算法帮 F 同学计算他最少要给自己留多少袋糖果才能满足要求，请描述算法的核心思想，必要时给出证明，给出算法伪代码并分析其对应的时间复杂度。例如，F 同学持有 3 袋糖果，分别装有 2, 1, 2 颗糖果，这时他至少需要给自己留 2 袋糖果，这两袋糖果的数量可以是 1, 2，或者是 2, 2。

1. 核心思想

首先将糖果数组 a 由小到大排序。想让自己留的糖果袋数尽可能少，且总数比妹妹的多，说明自己需要留下装糖果更多的袋子，即数组下标由大至小 (糖果数由多至少) 依次划分给哥哥，直至哥哥的糖果总量大于全部的二分之一。

2. 简单证明

证明哥哥选的 k 袋不是容量由大到小的前 k 袋时一定不是最优解。(同假设哥哥选的 k 袋不是容量由大到小的前 k 袋时是最优解，后证明存在不劣于最优解的解)

假设哥哥最后拿的糖果袋为 $a_{s_1}, a_{s_2}, ..., a_{s_k}$，共 k 袋，若去掉一袋后仍满足约束，则从容量由小到大去除直至恰好满足约束。此时若该 k 袋不为容量由大到小的前 k 袋，则找到当前未被选中的最大糖果数的袋子 a_x，将此时容量最小的一袋 a_y 替换为 a_x，相比之前多出 $a_y - a_x$ 个糖果，在始终满足哥哥大于妹妹的前提下，可从容量最小的袋子开始去除... 如此反复操作，直至哥哥选的 $k^{'}$ 袋是容量由大到小的前 $k^{'}$ 袋，此时 $k^{'} \leq k$，是最优解。

3. 伪代码

Algorithm 2 分糖果问题

Input: $a[1..n]$

Output: cnt

将数组 a 由小至大排序，结果仍记为 a

$total \leftarrow 0, brototal \leftarrow 0$

for $i : 1 \rightarrow n$ **do**
  $total \leftarrow total + a[i]$
end

for $i : n \rightarrow 1$ **do**
  $cnt \leftarrow cnt + 1$
  $brototal \leftarrow brototal + a[i]$
  if $brototal > total - brototal$ **then**
    $break$
  end
end

4. 时间复杂度分析

数组排序时间复杂度为 $O(nlogn)$，两次遍历数组时间复杂度为 $O(n)$，该算法时间复杂度为 $O(nlogn)$。

3. 最大收益问题

某公司有一台机器，在每天结束时，该机器产出的收益为 X1 元。在每天开始时，若当前剩余资金大于等于 U 元，则可以支付 U 元来升级该机器（每天最多只能升级一次）。从升级之日起，该机器每天可以多产出 X2 元的收益。即是说，在执行 K 次升级之后，这台机器每天的产出为 X1 + K Œ X2 元。

该公司初始资金为 C 元，请你设计算法求出 n 天之后该公司拥有的总资金的最大值。请描述算法的核心思想，必要时给出证明，给出算法伪代码并分析其对应的时间复杂度。

1. 核心思想

在一天起始可以选择升级机器与否,我们要判断升级机器所花的钱和升级机器后预期赚到的钱的多少,由于我们已知共 n 天，则第 k 天 (k <= n) 时，若选择升级机器需要花费 U 元，而此次升级机器共可以多赚 $X2 * (n - k + 1)$ 元，因此当 $U < X2 * (n - k + 1)$ 时，我们升级机器是赚的，反之亏。进一步发现 X2，U，n 皆为定值，函数 $X2 * (n - k + 1) - U$ 单调递减，存在某一天 m 作为分界，自 m 天之前升级皆赚，之后升级皆亏，且显然在第 i 天升级与在第 j 天升级 (i < j < m) 相比，在第 k 天 (j < k) 时，第 i 天升级的资金会大于第 j 天升级的资金。

同时需要考虑 m 天之前存在资金不够升级的情况，我们的选择为 m 天之前有钱升级就升级，钱不够则不升，下做证明。

2. 简单证明

证明在分界 m 之前每天有钱升级便升级。假设存在一个最优解，其升级机器的时机不是按照上述策略来选择。其升级时机为 $s_1^{'}, s_2^{'}...s_{k'}^{'}(s_1^{'} < s_2^{'}...)$，而按照上述规则升级时机为 $s_1, s_2...s_k(s_1 < s_2...)$，倘若 $k^{'} \leq k$，那么我们可以通过依次交换升级时机 $s_1^{'} \rightarrow s_1...s_{k'}^{'} \rightarrow s_{k'}$，根据核心思想中的推导，可转化为一个按照上述贪心策略的解，总资金不会减少。

因此该问题转化为证明：按照上述贪心算法策略能升级的天数一定是最多的，不存在任一种策略前 m 天可升级超过 k 次，即证明 $k^{'} \leq k$。

假设存在一种策略前 m 天可升级 k+1 次，升级时机为 $s_1^{'}, s_2^{'}...s_k^{'}, s_{k+1}^{'}(s_1^{'} < s_2^{'}...)$，根据上述交换升级时机的方式，我们可将升级时机转变为 $s_1, s_2...s_k, s_{k+1}^{'}(s_k < s_k^{'} < s_{k+1}^{'})$，此时在第 $s_{k+1}^{'}$ 天时总资金不会减少，且前 k 次升级与贪心策略一致，根据假设贪心策略无资金进行第 k+1 次升级，而该策略在第 $s_{k+1}^{'}$ 天实际的资金不高于贪心策略，因此必然无资金进行第 k+1 次升级，与假设矛盾，故最多 k 次升级。

3. 伪代码

Algorithm 3 最大收益问题

Input: $X1, X2, U, n, C$

Output: $Cnow$

$m \leftarrow \lfloor -U/X2 + n + 1 \rfloor, updateCnt \leftarrow 0, Cnow \leftarrow C$

for $i : 1 \rightarrow n$ **do**

  if $i \leq m$ & $Cnow \geq U$ **then**

    $Cnow \leftarrow Cnow - U$

    $updateCnt \leftarrow updateCnt + 1$

  end

  $Cnow \leftarrow Cnow + X1 + updateCnt * X2$

end

4. 时间复杂度分析

仅遍历 n 天，时间复杂度为 $O(n)$。

4. 哈密顿路径问题

哈密顿路径 (Hamiltonian path) 是指图中每个节点都仅经过一次且必须经过一次的路径。对于一般的图结构来说，求解哈密顿路径的问题是 NP 难问题。然而，在有向无环图上寻找哈密顿路径的问题是存在多项式时间的解法的。

如图 1 所示，左侧图包含一条哈密顿路径 1 4 3 2 5，右侧图则不包含哈密顿路径。给定一个有向无环图 G = (V, E)，请设计一个高效算法来寻找图 G 的一条哈密顿路径，如不存在哈密顿路径则返回 1，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度

1. 核心算法

哈密顿路径指每个节点都仅经过一次且必须经过一次的路径，类似一笔画，可以考虑通过拓扑排序变种解决。

首先我们已知该图为有向无环图，即一定可以进行拓扑排序，根据拓扑排序的广度优先策略我们可知起始至少存在一个顶点的入度为 0，且每次删去一个入度为 0 的点后会出现新的入度为 0 的点，直至顶点删空。我们首先计算各顶点的入度 $s_{v_i}(v_i \in V)$，选择一入度为 0 的点为起始点。

进而观察哈密顿回路和拓扑排序的不同，对于当前图 G(V, E) 而言，若存在大于 1 个入度为 0 的点，则意味着我们一定无法找到哈密顿回路，因为入度为 0 的点只能作为起点，而起点只有一个，因此在采用拓扑排序不断删除节点的同时，实时计算当前图中各顶点的入度，若存在大于 1 个入度为 0 的顶点，则无法构成哈密顿回路，反之可构成。换句话说，当拓扑排序中记录入度为 0 的节点的队列长度大于 1 时，则无法构成哈密顿回路。构建已删除顶点的数组，若能够构成哈密顿回路，则正序输出数组内顶点即为一条哈密顿回路。

2. 伪代码

Algorithm 4 哈密顿路径问题

Input: 图 $G(V,E)$

Output: $(v_1,v_2,...v_n) or -1$

初始化空队列 Q, 数组 $Path$

for $v \in V$ **do**
  if $v.in\_degree = 0$ **then**
    $Q.Enqueue(v)$
  end
end
while *not* $Q.is\_empty()$ **do**
  if $Q.size() > 1$ **then**
    return -1
  end
  $u \leftarrow Q.Dequeue()$
  $Path.append(u)$
  for $v \in G.Adj(u)$ **do**
    // 由 u 指向 v
    $v.in\_degree \leftarrow v.in\_degree - 1$
    if $v.in\_degree = 0$ **then**
      $Q.Enqueue(v)$
    end
  end
end
顺序输出 $Path$ 数组即为所求

3. 时间复杂度分析

初始化队列需要遍历 $|V|$ 个节点，时间复杂度为 $O(|V|)$；后主循环时间复杂度同理拓扑排序，时间复杂度为 $O(\sum_{v\in V}(1+|Adj[v]|)) = O(|V|+|E|)$，该算法时间复杂度为 $O(|V|+|E|)$.

5. 马的遍历问题

给定一个 nŒm 的中国象棋棋盘，规定其左下角的格点为坐标原点，坐标系 x 轴和 y 轴分别沿右方以及上方延伸。现在点 (x, y) 上有一个马，该棋子与中国象棋的马移动规则相同，并且不许移动到棋盘范围外。请设计一个高效算法，计算一个距离矩阵 D[n][m]，D[i][j] 表示马从 (x, y) 移动到 (i, j) 最少需要几步，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

1. 核心思想

由于求最小需要步数，考虑使用 BFS 遍历，因为 BFS 具有遍历到的每个点一定是步数最少的性质，步骤如下。

1. 首先我们初始化矩阵 $D[n][m]$ 内每个元素为 ∞，$D[i][j]$ 记录从 (x,y) 到 (i,j) 最小需要步数；初始化遍历到的顶点队列，将 (x,y) 入队，$D[x][y] \leftarrow 0$；初始化记录是否遍历到的数组 $vis[n][m]$ 各元素为 $false$，将 $vis[x][y] \leftarrow true$。

2. 出队一顶点 v，将 v 的马能访问到的全部未访问过的节点加入队列，步数加一，并标记为已访问。

3. 重复第二步，直至队列为空。

2. 伪代码

Algorithm 5 马的遍历问题

Input: x, y, n, m

Output: $D[n][m]$

$d \in D[n][m] \leftarrow \infty$, $D[x][y] \leftarrow 0$

$v \in vis[n][m] \leftarrow false$, $vis[x][y] \leftarrow true$

初始化队列 Q，$Q.Enqueue((x, y))$

while *not* $Q.is\_empty()$ **do**

// 某顶点 (i, j) 的马可能跳的下一个位置有 8 种情况，$(i+1, j+2), (i+2, j+1), (i-1, j+2), (i-2, j+1), (i+1, j-2), (i+2, j-1), (i-1, j-2), (i-2, j-1)$

$(a, b) \leftarrow Q.Dequeue()$

计算马在当前顶点下一步可跳到的顶点集合 $Next$

for $(i, j) \in Next$ & $vis[i][j] = false$ & (i, j) 没有越界 **do**

$vis[i][j] \leftarrow true$

$D[i][j] \leftarrow D[a][b] + 1$

$Q.Enqueue((i, j))$

end

end

3. 时间复杂度分析

初始化各数组遍历全部顶点，时间复杂度为 $O(mn)$，BFS 算法中，外层遍历全部顶点 mn，而内层循环不再依赖于图的邻接矩阵，因为本问题的图特殊，一个顶点的邻接顶点仅有固定的八个顶点，因此内层循环为常数复杂度，主 BFS 时间复杂度为 $O(mn)$，该算法时间复杂度为 $O(mn)$。