北京航空航天大学《算法设计与分析》第三次作业

1 分蛋糕问题 (20 分)

给定 n 块体积不同的蛋糕，其体积分别用 $a_1, \ldots, a_n$ 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 $s_1, \ldots, s_k$ $(1 \le s_1 < \cdots < s_k \le n)$，则这次分配的不公平度为

$$\max\{a_{s_1}, \ldots, a_{s_k}\} - \min\{a_{s_1}, \ldots, a_{s_k}\}$$

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。

1.1 算法核心思想

对于该问题可以采用贪心算法的思想进行分析，对于 n 块体积不同的蛋糕，我们首先按照体积从小到大将其排序，然后从中选择 k 块，使得这 k 块蛋糕中最大的体积和最小的体积之差最小，那么不难证明这 k 块蛋糕一定是连续的，我们只需要从第一块开始计算连续 k 块蛋糕的最后一块和第一块的体积之差，如果体积差小于已经算出的最小体积差，则更新最小的体积差，直到分析完所有的蛋糕。

对于选取的 k 块蛋糕的连续性证明如下：

假设我们目前选取了一个包含连续 k 块蛋糕的序列，为 $S = \{a_{s_1}, \ldots, a_{s_k}\}$，其中 $s_1 < \cdots < s_k$，并且有另一个序列 $S' = \{a_{s'_1}, \ldots, a_{s'_k}\}$，其中 $s'_1 < \cdots < s'_k$，且 s'_1 与 s_1 相等，S' 序列是一个以 s_1 开头的体积极差最小的序列。因为 S 序列中 s_1 到 s_k 连续，且 S' 序列以 s_1 开头，因此必然有 $s'_k \ge s_k$，又因为蛋糕的体积按照从小到大排序，那么又有 $a_{s'_k} \ge a_{s_k}$。而 S 序列体积的极差为 $a_{s_k} - a_{s_1}$，S' 序列体积的极差为 $a_{s'_k} - a_{s_1}$，因此 $a_{s'_k} - a_{s_1} \ge a_{s_k} - a_{s_1}$，因此 S' 序列的体积极差不小于 S 序列的体积极差，因此 S 序列是一个以 s_1 开头的体积极差最小的序列。

1.2 算法伪代码

Algorithm 1: 分蛋糕问题

Input: 正整数 n，为蛋糕数量，数组 $A[n]$，为蛋糕的体积数组

Output: 长度为 k 的数组 Out，为选取的蛋糕的编号，正整数 $Fair$，为最小的不公平度

```
function n, A:
    let id[n] be a new one-dimension array//记录初始时刻蛋糕的编号
    let Out[k] be a new one-dimension array//记录选取的蛋糕的编号
    Fair = ∞;//初始化最小不公平度为无穷大
    FairBegin = 0;//初始化最小不公平度序列的起始位置
    for i = 0 → n − 1 do
        id[i] = i + 1;
    end
    Sort(A, id);//对蛋糕的体积进行排序，同时调整相应蛋糕的编号
    for i = 0 → n − k do
        FairTemp = A[i + k − 1] − A[i];//计算当前序列的不公平度
        if FairTemp < Fair then
            Fair = FairTemp;//更新最小不公平度
            FairBegin = i;//更新最小不公平度序列的起始位置
        end
    end
    for i = 0 → k − 1 do
        Out[i] = id[FairBegin + i];//记录选取的蛋糕的编号
    end
    return Out, Fair;
end
```

1.3 时间复杂度分析

- 排序时间复杂度为 $O(n\log n)$
- 初始化编号序列和计算最小不公平度的时间复杂度为 $O(n)$
- 记录选取蛋糕编号的时间复杂度为 $O(k)$
- **总时间复杂度为** $O(n\log n)$

2 分糖果问题 (20 分)

假设 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 $\{2,2\}$。

2.1 算法核心思想

本题采用贪心算法的思想进行分析，由于我们知道糖果数量的总和和每一袋糖果的数量，但是不知道取出的糖果数量具体是多少，因此我们可以假设每次取出的都是目前所有袋子中糖果数最多的那一袋，直到取出的糖果数量大于糖果总数的一半。此时取出的糖果袋数即为最少需要留下的糖果袋数。

下面证明这一思路的正确性：

首先将 n 袋糖果按照里面装的糖果数量从大到小进行排序，设排序后的糖果数量为 $a_1, a_2, \cdots, a_n$，则有 $a_1 \geq a_2 \geq \cdots \geq a_n$。其中必然有一个 i 满足 $\sum_{i=0}^{i-1} a_i \leq \frac{\sum_{k=0}^{n-1} a_k}{2} \&\& \sum_{i=0}^{i} a_i > \frac{\sum_{k=0}^{n-1} a_k}{2}$。对于任何的 $p < i$，这 p 袋糖果总数一定满足 $sum(p) \leq \sum_{j=0}^{p-1} a_j \leq \sum_{j=0}^{i-1} a_j < \sum_{j=0}^{n-1} a_j$，即哥哥留下的糖果数一定少于妹妹，因此 i 是满足条件的最小值。

2.2 算法伪代码

Algorithm 2: 分糖果问题

Input: 正整数 n，为糖果袋数；长度为 n 的数组 $number$，为每袋糖果的数量

Output: 正整数 min，为需要留下的最少糖果袋数

```
function main(n):
    Sort(number);//按照糖果数量从大到小对糖果袋进行排序
    sum = 0;//初始化糖果总数
    for i = 0 → n − 1 do
        sum = sum + number[i];//计算糖果总数
    end
    sum = sum/2;
    candyCount = 0;
    for min = 0 → n − 1 do
        candyCount = candyCount + number[min];//计算取出的糖果数量
        if candyCount > sum then
            break;
        end
    end
    return min;
end
```

2.3 时间复杂度分析

- 排序时间复杂度为 $O(n \log n)$
- 计算糖果总数时间复杂度为 $O(n)$
- 计算留下的最少袋数的时间复杂度为 $O(n)$
- **总时间复杂度为** $O(n \log n)$

3 最大收益问题 (20 分)

假设某公司有一台机器，在每天结束时，该机器产出的收益为 X_1 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 X_2 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 $X_1 + K \times X_2$ 元。

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

3.1 算法核心思想

本题采用贪心算法的思想进行分析，对于第 i 天，如果这一天选择升级机器，那么这一次升级在接下来的 $n-i+1$ 天（包含这一天）多产生的收益为 $(n-i+1)X_2$ 元，而这次升级消耗的成本为 U 元，因此只要 $(n-i+1)X_2 > U$，就选择升级机器，否则选择不升级，直到第 n 天结束，此时计算出来的总收益就是该公司拥有总资金的最大值。

下面证明该想法的正确性：

在 n 天中，一定存在一个 m 满足 $m(n-m+1)X_2 \leq U$ 且 $(m-1)(n-m+2)X_2 > U$，同时存在一个 s 满足 $C+(s-1)X_1 \geq U$，s 为有足够资金升级机器的第一天。我们假设第一种决策方案为第 s 天到第 $m-1$ 天均选择升级机器（如果 $m \geq s$），以后均选择不升级。同时存在一种最优决策方案为在第 s 天到第 n 天中选择 k 天进行升级，记选择的这 k 天为 $t_1, t_2, \ldots, t_k$，那么对于 $1 \leq j \leq k \&\& a_j \leq (m-1)$，一定有 $a_j \geq (s+j-1)$，也就是有 $(n-a_j+1)X_2 \leq (n-(s+j-1)+1)X_2$，即对于 $1 \leq j \leq k \&\& a_j \leq (m-1)$ 这部分来说，其升级所产生的收益一定小于第一种决策方案的总收益，而当 $a_j \geq m$ 时，有 $(n-a_j+1)X_2 \leq (n-m+1)X_2 \leq U$，因此对于这部分来说，其升级产生的收益为负，因此该最佳方案的总收益不会大于方案一的收益，从而证明我们的方案一为最佳方案。

3.2 算法伪代码

Algorithm 3: 最大收益问题

Input: 正整数 n，为总天数；正整数 U，为升级机器的费用；正整数 X_1，为机器初始收益；正整数 X_2，为升级一次每天的附加收益；非负整数 C，为初始资金

Output: 正整数 $maxMoney$，为 n 天后该公司总资金的最大值

```
function main(n, U, X1, X2, C):
    maxMoney = C;//初始化总资金
    for i = 1 → n do
        if maxMoney ≥ U&&(n − i + 1)X2 > U then
            maxMoney = maxMoney + X1 + (n − i + 1)X2 − U;
            //升级机器
        end
        else
            maxMoney = maxMoney + X1;
            //不升级机器
        end
    end
    return maxMoney;
end
```

3.3 时间复杂度分析

- 更新总资金的时间复杂度为 $O(n)$
- **总时间复杂度为** $O(n)$

4 哈密顿路径问题 (20 分)

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。

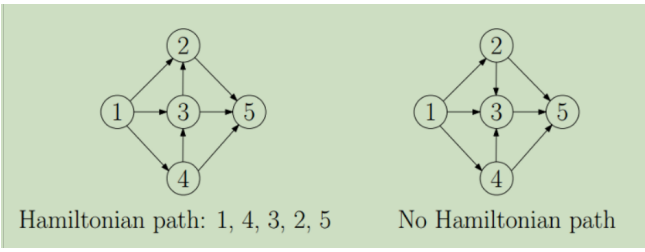

Figure 1: 哈密顿路径

如图 1 所示，左侧图包含一条哈密顿路径 $1 \to 4 \to 3 \to 2 \to 5$，右侧图则不包含哈密顿路径。

给定有向无环图 $G = (V, E)$，请设计一个高效算法来寻找图 G 的哈密顿路径。如果不存在哈密顿路径，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。

4.1 算法核心思想

对于该问题，我们可以对给定的有向无环图进行拓扑排序，如果拓扑排序的结果中存在两个相邻的节点，其在图中不存在一条边，那么这个图一定不存在哈密顿路径，否则这个图一定存在哈密顿路径，并且该哈密顿路径即为拓扑排序的结果。

下面证明算法的正确性：

如果拓扑排序结果中每一个顶点都和下一个顶点之间存在边，那么显然这个结果就是一条哈密顿路径。

否则，对于图中没有边相连的顶点 u 和 v，因为拓扑排序的性质，如果这两个顶点之间存在一条包含其他顶点的路径，那么在拓扑序列中该顶点一定位于这两个顶点之间，因此这两个顶点之间不存在任何相连的路径，也从而说明该图不存在哈密顿路径。

4.2 算法伪代码

Algorithm 4: 哈密顿路径问题（主函数）

Input: 图 G，为有向无环图

Output: 正整数数组 $hamil$，为哈密顿路径序列，如果不存在哈密顿路径，则返回 -1

```
1  function main(G):
2      topoArray = new int[G.V];//初始化拓扑排序序列
3      topoArray = topoSort(G);//对图进行拓扑排序
4      for i = 0 → G.V − 2 do
5          if G[topoArray[i]][topoArray[i + 1]] == 0 then
6              return −1;
7              //如果拓扑排序序列中存在两个相邻的顶点，其在图中不存在一条边，那么这个图一
                 定不存在哈密顿路径
8          end
9      end
10     hamil = topoArray;//哈密顿路径即为拓扑排序的结果
11     return hamil;
12 end
```

Algorithm 5: 哈密顿路径问题（拓扑排序函数）

```
Input: 图 G，为有向无环图；数组 topoArray，为拓扑排序序列
function topoSort(G, topoArray):
    Q = new Queue;//初始化空队列 Q
    for v ∈ G.V do
        if v.inDegree == 0 then
            Q.push(v);//将入度为 0 的顶点加入队列
        end
    end
    while not Q.isEmpty do
        u = Q.pop();//从队列中取出一个顶点
        topoArray.push(u);//将该顶点加入拓扑排序序列
        for v ∈ G.Adj[u] do
            v.inDegree = v.inDegree − 1;//将该顶点的所有邻接点的入度减一
            if v.inDegree == 0 then
                Q.push(v);//将入度为 0 的顶点加入队列
            end
        end
    end
    return;
end
```

4.3 时间复杂度分析

- 拓扑排序的时间复杂度为 $O(|V|+|E|)$
- 判断 Hamiltonian 路径是否存在的时间复杂度为 $O(|V|)$
- **总时间复杂度为** $O(|V|+|E|)$

5 马的遍历问题 (20 分)

给定一个 $n \times m$ 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，并且不允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。

5.1 算法核心思想

本题可以使用宽度优先搜索的方法进行计算，马的每一次移动最多有八种可能性，由于使用的是宽度优先搜索，因此这八个目标点中仍未被访问的点与源点之间的最少步数一定是当前步数加一，更新这些未被访问的点的最少步数，然后将他们加入队列，直到队列为空，此时所有的点都被访问过，$D[n][m]$ 中的 $D[i][j]$ 就是马从 (x, y) 移动到 (i, j) 最少需要的步数。

5.2 算法伪代码

Algorithm 6: 马的遍历问题

Input: 正整数 n, m，为棋盘的长宽；正整数 x, y，为马的初始位置

Output: 正整数数组 $D[n][m]$，为马从 (x, y) 移动到棋盘中任一点最少需要的步数

```
1  function main(n, m, x, y):
2      D = new int[n][m];//初始化距离矩阵
3      visited = new int[n][m];//初始化访问矩阵
4      Q = new Queue;//初始化空队列 Q
5      Q.push((x, y));//将初始位置加入队列
6      visited[x][y] = 1;//将初始位置标记为已访问
7      while not Q.isEmpty do
8          u = Q.pop();//从队列中取出一个顶点
9          for v ∈ u.next do
10             //u.next 包含马的所有可能目标，即在棋盘内的 (x+2,y+1),(x+2,y-1),(x-
                2,y+1),(x-2,y-1),(x+1,y+2),(x+1,y-2),(x-1,y+2),(x-1,y-2)
11             if visited[v.x][v.y] == 0 then
12                 D[v.x][v.y] = D[u.x][u.y] + 1;//更新距离矩阵
13                 visited[v.x][v.y] = 1;//将该顶点标记为已访问
14                 Q.push(v);//将该顶点加入队列
15             end
16         end
17     end
18     return D;
19 end
```

5.3 时间复杂度分析

- 由于每个点只需要更新一次，因此计算距离矩阵的时间复杂度为 $O(nm)$
- **总时间复杂度为** $O(nm)$