计算机学院《算法设计与分析》
(2020 年秋季学期)

第四次作业参考答案

1 对下面的每个描述，请判断其是正确或错误，或无法判断正误。对于你判为错误的描述，请说明它为什么是错的。(每小题 5 分，共 20 分)

1. $NP\text{-}hard \subseteq NP$；
2. 对某问题 $X \in NP$ 而言，若可以证明规约式 $3\text{-}SAT \leq_p X$，则 $X \in NPC$；
3. $P \neq NP$；
4. 所有 NP 完全问题均无法在多项式时间内被解决。

解：

1. 错误，$NP\text{-}hard$ 问题未必在多项式时间内可以验证。
2. 正确。
3. 无法判定。
4. 无法判定。

2 最小点集问题 (20 分)

给定一个包含 n 个点的连通有向图 $G = (V, E)$，节点编号为 $1, 2, \cdots, n$，请设计算法找出最小的点集 $U \subseteq V$，使得对所有点 $v \in V$，均存在某点 $u \in U$，使得图中存在一条从 u 到 v 的路径。如果这样的点集有多个，求出任意一个即可。此外，请分析该算法的时间复杂度。

例如，给定一个包含 $n = 6$ 个点的图，边集 $E = \{(1,2),(1,3),(3,6),(4,5),(5,3)\}$。可以发现，在该图中从 1 出发可到达 $2, 3, 6$，从 4 出发可到达 $3, 5, 6$。因此，选择点集 $U = \{1, 4\}$ 即可满足条件。

解：

1. 问题分析

注意到对于任意一点 u，其可以到达的点集，等于其所在的强连通分量可达的点集。换言之，选择一个点加入 U 集，意味着选择其对应的强连通分量。

2. 强连通分量性质

若将一个强连通分量视作一个单一的节点，两个强连通分量之间有连边则相应节点之间有边，这样收缩出来的新图 $G' = (V', E')$ 是有向无环图 (DAG)。

3. 贪心策略

注意到该 DAG 中每个入度为零的点都必须选择。故仅需要选择该 DAG 中入度为零的点即可以到达图中的所有点。即在这些点对应在原先的强连通分量中各选一个点加入点集 U 即为最小的点集 U。

4. 时间复杂度分析

注意到整个求强连通分量的过程是 $O(|V| + |E|)$ 的，而选出 DAG 中入度为 0 的点也是 $O(|V|)$ 的。故总的时间复杂度为 $T(|V|, |E|) = O(|V| + |E|)$。

参考伪代码如1所示。

Algorithm 1 $minset(V, E)$

Input: 给定的图 $G(V, E)$
Output: 所选出的最小点集

求出 G 图的所有强连通分量 $\{s_1, s_2, \cdots, s_k\}$
$V' \leftarrow \{s_1, s_2, \cdots, s_k\}$
$E' \leftarrow \{< s_u, s_v > | < u, v >\in E, u \in s_u, v \in s_v\}$
$in[s_i] \leftarrow |\{< s_u, s_i > | < s_u, s_i >\in E'\}|$
$ANS \leftarrow \emptyset$
for $i : 1 \rightarrow k$ **do**
  if $in[s_i] = 0$ **then**
    任取 s_i 中一个点加入 ANS
  end if
end for
return ANS

3 项目排序问题 (20 分)

公司有 n 个项目和 m 个小组，项目和小组都从 1 开始编号。每个项目可能有两种情况：一是没有归属，二是由某个小组负责。我们用 $group[i]$ 代表第 i 个项目所属的小组，如果这个项目无需任何小组负责，那么 $group[i] = 0$。

现需要对这些项目制定完成顺序，并返回排序后的项目列表，要求如下：

1. 同一小组的项目，排序后在列表中彼此相邻；
2. 项目之间存在一定的依赖关系，用 $pre[i]$ 表示。其含义为在进行第 i 个项目前，应该完成的项目集合。

如果存在多个解决方案，只需要返回其中任意一个即可；如果没有合适的解决方案，就返回一个空列表。请设计算法解决该问题并分析其时间复杂度。

例如，给定 $n = 8$ 个项目，$m = 2$ 个小组，项目归属 $group = [0, 0, 2, 1, 1, 2, 1, 0]$，其含义为第 3，第 6 个项目由小组 2 负责；第 4，第 5，第 7 个项目由小组 1 负责；其他项目无需任何小组负责。项目依赖关系如下：$pre[2] = \{7\}, pre[3] = \{6\}, pre[4] = \{7\}, pre[5] = \{4, 7\}$。第 1 和第 6 个项目不依赖于任何其他项目，因此这里略去。

在此情况下，一个可行的项目完成顺序为 $[7, 4, 5, 2, 6, 3, 1, 8]$，其中项目 $4, 5, 7$ 同属小组 1，项目 $3, 6$ 同属小组 2，在列表中相邻；项目 7 在 2 前，6 在 3 前，7 在 4 前，$4, 7$ 在 5 前，满足项目依赖关系。

解：

1. 问题分析

注意到一个组的项目应当连续的完成，故项目之间的依赖关系有两种：

1. 同一个组之间的依赖关系；
2. 不同组之间的依赖关系。

故问题可以分成组内的完成顺序排序，和组与组之间的完成顺序排序两个步骤。

2. 拓扑排序

对每个组 A 内部的任意一个 i，有若干条有向边 $\{< u, i > |u \in pre[i] \cap group[u] = A\}$。这样构成的有向图 G_A 做一次拓扑排序，得到组内的完成顺序排序 S_A。

而对于不同组之间的依赖关系，可以将每个组视作一个节点，而 $group[i] = 0$ 的节点各自为一个节点（不能将无小组负责的所有节点视作同一个组）。在此条件下的依赖关系同样会构成一个有向图 G'，在此图之上再做一次拓扑排序，得到每个组的出现顺序。

3. 无解情形

上述环节中的任意一次拓扑排序失败，则证明依赖关系存在环路，无法排序。

4. 方案输出

首先根据 G' 的拓扑排序结果，得到每个组的出现顺序，再将每个组 A 的节点按照 S_A 的顺序，依次排序即可。

5. 时间复杂度分析

首先，记依赖关系的总数为 T。

注意到每个组内部拓扑排序的总需要遍历的节点数为总项目数，需要的考察的依赖关系为依赖关系总数，故总时间复杂度为 $O(n+T)$。

而在组与组之间的拓扑排序需要考虑的节点为总组数以及不为没有组负责的项目数，需要考察的依赖关系为依赖关系总数，故总时间复杂度为 $O(n+m+T)$。

故总的时间复杂度为 $T=(n,m,T)=O(n+m+T)$。

参考伪代码如2所示。

Algorithm 2 $tasksort(n, m, group[1..n], pre[1..n])$

Input: n 个项目和 m 个小组，以及对应的小组编号 $group[1..n]$，以及依赖关系集合。

Output: 排序结果

$V_i \leftarrow \{u|group[u]=i\}$
$E_i \leftarrow \emptyset$
$V' \leftarrow \{n+1,\cdots,n+m\} \cup \{i|group[i]=0\}$
$E' \leftarrow \emptyset$
for $i:1\rightarrow n$ **do**
  for $u: pre[i]$ **do**
    if $group[u]=group[i] \wedge group[u]>0$ **then**
      将 $<u,i>$ 加入 $E_{group[i]}$ 集合
    else
      $T_u \leftarrow \begin{cases} n+group[u] & \text{if } group[u]\neq 0 \\ u & \text{if } group[u]=0 \end{cases}$
      $T_i \leftarrow \begin{cases} n+group[i] & \text{if } group[i]\neq 0 \\ i & \text{if } group[i]=0 \end{cases}$
      将 $<T_u,T_i>$ 加入 E' 的集合
    end if
  end for
end for
对 $G_i(V_i,E_i)$ 求拓扑序 S_i
对 $G'(V',E')$ 求拓扑序 S'
$ANS \leftarrow \emptyset$
if S_i 中任意一个不存在，或者 S' 不存在 **then**
  return ANS
end if
for $e \in S'$ **do**
  if $e \leq n$ **then**
    将 e 加入 ANS 最末尾
  else
    for $x: S_{e-n}$ **do**
      将 x 加入 ANS 最末尾
    end for
  end if
end for
return ANS

4 方程式求解问题 (20 分)

给定 n 个变量 $a_i(1\leq i\leq n)$，和 m 个方程式，其中第 k 个方程式以三元组 $(i_k,j_k,v_k)(1\leq i_k,j_k\leq n)$ 的形式给出，其含义为 $a_{i_k}/a_{j_k}=v_k$。请设计算法，根据已知的 m 个方程式求解目标式子 $a_x/a_y(1\leq x,y\leq n)$ 的值，并分析该算法的时间复杂度。

可以假设输入总是有效的，且除法运算中不会出现除数为 0 的情况。

例如：对给定的 3 个变量 a_1,a_2,a_3，已知 $m=2$ 个方程式 $a_1/a_2=3, a_2/a_3=2$（对应的三元组分别为 $(1,2,3)$ 和 $(2,3,2)$），现要求取目标式子 a_1/a_3 的值 $(x=1,y=3)$。则根据上述两个

方程式可推出，$a_1/a_3 = 6$。

参考伪代码如3所示。

解：

1. 问题分析

注意到如果有 $a_{i_k}/a_{j_k} = v_k$，这相当于 i_k 到 j_k 与一条权值为 v_k 的有向边，j_k 到 i_k 有一条权值为 $\frac{1}{v_k}$ 的有向边。故此时如果点 a_x 到点 a_y 有路径，则 a_x/a_y 为路径上所有边的边权的乘积，否则不可以求解目标 a_x/a_y。

2. 搜索算法

故可以考虑直接从 a_x 开始广度优先遍历整张图，即 p_k 表示从 a_x 到 a_k 的所有边权的乘积，当遍历至 a_y 时，搜索结束。

3. 时间复杂度分析

整个搜索时，所有点和边最多被遍历一次，而每条边对应了一个方程式，每个点对应了一个变量，故总的时间复杂度为 $T(n,m) = O(n+m)$。

参考伪代码如3所示。

Algorithm 3 $solve(n, m, (i_k, j_k, v_k), a_x, a_y)$

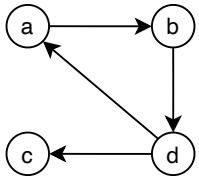
```
E[i] ← {(j, v)|∃k s.t. (i, j, v) = (i_k, j_k, v_k) ∪ (j, i, 1/v) = (i_k, j_k, v_k)}
vis[i] ← 0
将 a_x 加入队列 Q
vis[a_x] ← 0
p[a_x] ← 1
while 队列 Q 不为空 do
    将 Q 队列首取出赋值给 now
    for (nxt, v) : E[now] do
        if vis[nxt] = 0 then
            vis[nxt] ← 1
            p[nxt] ← p[now] × v
            将 nxt 加入队列 Q
        end if
    end for
end while
if vis[a_y] = 0 then
    return 不可求解 a_x/a_y
else
    return p[a_y]
end if
```

5 传递闭包问题 (20 分)

给定一个包含 n 个节点的有向图 $G = (V, E)$，其传递闭包定义为一个 n 阶布尔矩阵 $T = \{t_{ij}\}$，其中矩阵第 i 行 $(i \le i \le n)$ 第 j 列 $(1 \le j \le n)$ 的元素 t_{ij} 表示图中是否存在从 i 到 j 的路径。如果从第 i 个顶点到第 j 个顶点之间存在一条有向路径，则 t_{ij} 为 1；否则 t_{ij} 为 0。

现给定一个有向图的邻接矩阵 A，请设计算法求出其传递闭包 T 并分析该算法的时间复杂度。

例如，对于有向图



其邻接矩阵为 $A=\begin{pmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&0&1&0\end{pmatrix}$，求出的传递闭包为 $T=\begin{pmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\1&1&1&1\end{pmatrix}$。

解：

1. 问题思路

求解传递闭包可以用 `Floyd-Warshall` 算法，即设 $D[i][j][k]$ 表示从 i 到 j 只经过 $1..k$ 中的节点为中间节点的是否可以到达。

则有：

1. 若 $D[i][j][k-1]=1$ 则 $D[i][j][k]=1$
2. 若 $D[i][k][k-1]=1 \wedge D[k][j][k-1]=1$ 则可以经过 k 为中间节点连通 i,j，即 $D[i][j][k]=1$

2. 时间复杂度分析

故，需要考察 $D[1..n][1..n][1..n]$ 的所有状态才能求得传递闭包 $T[1..n][1..n]=D[1..n][1..n][n]$。而每个状态需要考察的都是上述两种选择，故总时间复杂度为 $T(n)=O(n^3)$

参考伪代码如4所示。

Algorithm 4 $transitive(n, A[1..n][1..n])$

```
D[1..n][1..n] = A[1..n][1..n]
for k : 1 → n do
    for i : 1 → n do
        for j : 1 → n do
            if D[i][k] = 1 ∧ D[k][j] = 1 then
                D[i][j] = 1
            end if
        end for
    end for
end for
return D[1..n][1..n]
```