计算机学院《算法设计与分析》第四次作业

计算机学院　20373673　于敬凯

December 19, 2022

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/无法判断的描述，请说明它为什么是错误/无法判断的。

1.1　P 类问题为 NP 类问题的真子集。

无法判定。$P = NP$ 目前仍是 open problem。

1.2　如果假设 $P = NP$，则 NP 完全问题可以在多项式时间内求解。

错误。反证法：如果 NP 完全问题可以在多项式时间内求解，则所有 NP 问题可以在多项式时间内求解，与假设 $P \neq NP$ 矛盾，故证毕。

1.3　若 SAT 问题可以用复杂度为 $O(n9)$ 的算法来解决，则所有的 NP 完全问题都可以在多项式时间内被解决

正确。

1.4　对于一个 NP 完全问题，其所有种类的输入均需要用指数级的时间求解。

错误。SAT 问题是 NP 完全问题，但是对于所有满足 $2-SAT$ 条件的输入，都可以在多项式时间内求解。

2　颜色交错最短路问题

给定一个无权有向图 $G =< V, E >$（所有边长度为 1），其中 $V = \{v_0, v_1, ..., v_{n-1}\}$，且这个图中的每条边不是红色就是蓝色（$\forall e \in E, e.color = red$ 或 $e.color = blue$），图 G 中可能存在自环或平行边。

现给定图中两点 v_x, v_y，请设计算法求出一条从 v_x 到 v_y，且红色和蓝色边交替出现的最短路径。如果不存在这样的路径，请输出 -1。请给出分析过程，伪代码和算法复杂度。

2.1　分析过程

本题中，当搜索到某个点 i 的时候，既可能是从一条红边访问而来，也可能是从一条蓝边访问而来，因此需要分别对这两种情况使用广度优先搜索。

使用元组 $(i, 0)$ 和 $(i, 1)$ 分别表示访问到该点时是从红边访问到的和访问到该点时是从蓝边访问到的。

以 n 表示图 G 中的点的数量；初始化二维数组 $vis[n][2]$ 为 0 表示 n 个点都处于未访问状态，$vis[i][0] = 1$ 表示节点 i 被红边访问过，$vis[i][1] = 1$ 表示节点 i 被蓝边访问过；初始化一维数组 $nxt[n]$，每个元素 $nxt[i]$ 为节点 i 的相邻节点集合；初始化二维数组 $prev[n][2]$ 为 -1 表示所有节点的前置访问节点都是非法的，$prev[i][k] = j$ 表示节点 i 被以 k 表示的颜色边访问时的前置节点编号。

初始时，将 $(v_x, 0)$ 和 $(v_x, 1)$ 都加入队列 q 中，表示从起点开始可以走蓝边或红边；将 $vis[v_x][0]$ 和 $vis[v_x][1]$ 均置 1 表示已经访问过。每次取出队列 q 的队首元素 $i, state$，对 $nxt[i]$ 的每一个节点 j，若 $vis[j][1 - state] = 0$，说明可以通过 $1 - state$ 代表的颜色边访问节点 j，则将其加入队列 q 并置 $vis[j][1 - state]$ 为 1 表示已访问。如此循环直至访问到 v_y 或队列为空。由广度优先搜索和贪心思想知，第一次访问到 v_y 时即为从 v_x 出发的最短路，又由于题目要求找到一条即可，因此可以直接停止算法。

2.2 伪代码

Algorithm 1: 颜色交错最短路问题

Input: 无权有向图 $G =< V, E >$，节点数量 n，起点 v_x，终点 v_y

Output: 若可达输出最短路径，若不可达输出 -1

```
1  function main(G =< V, E >, n, v_x, v_y):
       // 初始化访问数组vis全为0
2      vis[n][2] ← {0};
       // 初始化相邻节点数组nxt表示相邻节点集合
3      nxt[n] ← 由G < V, E > 初始化相邻节点集合数组;
       // 初始化搜索队列q
4      q ← new Queue();
       // 初始化前置节点数组
5      prev[n][2] ← {−1};
       // 将(v_x, 0), (v_x, 1)入队
6      q.Enqueue((v_x, 0)), q.Enqueue((v_x, 1));
       // 标记v_x已访问
7      vis[v_x][0] ← 1, vis[v_x][1] ← 1;
8      while not q.empty() do
           // 取出队首元素
9          (i, state) ← q.Dequeue();
           // 遍历节点i的相邻节点集合
10         for j : nxt[i] do
               // 若节点j没有被1 − state颜色的边访问过
11             if vis[j][1 − state] = 0 then
                   // 将该点和访问该点的边的颜色加入队列
12                 q.Enqueue((j, 1 − state));
                   // 设置前置访问节点
13                 prev[j][1 − state] ← i;
                   // 将其标记为已访问
14                 vis[j][1 − state] ← 1;
15                 if j = v_y then
                       // 访问到终点v_y说明找到了一条最短路
16                     return getPath(prev, v_y, 1 − state);
17                 end
18             end
19         end
20     end
       // 运行至此说明没有找到最短路，返回−1
21     return −1;
22 end
```

Algorithm 2: 输出最短路

Input: 前置节点数组 $prev[][]$，终点节点 v_y，终点状态 $state$

Output: 最短路

```
function getPath(prev[][], v_y, state):
    // 初始化访问节点序列
    vis[] ← [];
    // 赋值当前访问节点
    node ← v_y;
    while node ≠ −1 do
        // 将node加入节点序列
        vis.add(node);
        // 获取当前节点前置节点
        node ← prev[node][state];
        // 设置状态为相反状态
        state ← 1 − state;
    end
    // 将节点序列倒置即为答案
    return vis.reverse();
end
```

2.3 算法复杂度

总状态个数为 $O(|V|)$ 级别，每次搜索新的状态都是在枚举点所连接的边，因此总的时间复杂度为 $T(|V|,|E|) = O(|V| + |E|)$。

3 最小闭合子图问题

对于一个有向图 $G =< V, E >$，其闭合子图是指一个顶点集为 $V \subseteq V'$ 的子图，且保证点集 V' 中的所有出边都还指向该点集。换言之，V' 需满足对所有边 $(u, v) \in E$，如果点 u 在集合 V' 中，则点 v 也一定在集合中。

现给定一个包含 n 个点的有向图 $G =< V, E >$，请设计算法求出该图中的闭合子图至少应包含几个顶点，并分析其时间复杂度。

例如，给定如下图所示的包含 5 个顶点的图，其闭合子图可能为：$\{v_3\}, \{v_0, v_1, v_2, v_3, v_4\}, \{v_2, v_4\}$。最小的闭合子图仅包含 1 个顶点，为 v_3。请给出分析过程，伪代码和算法复杂度。

3.1 分析过程

由题意知，对于图中任何一点，如果其在闭合子图中，那么其所有出边节点都在闭合子图中，因此其强连通分量也在该闭合子图中。

考虑将每个强连通分量视作单一的点，不同的强连通分量包含的节点之间有边则这两个强连通分量有边，如此得到的收缩后的图 G' 是一个有向无环图，在此基础上选择闭合子图，若选择任意节点，需要将其所有后继节点（如果有）都选入闭合子图，直至其所有后继节点都在闭合子图中或不再有没有加入闭合子图的节点。

由 SCC 的性质可知，收缩后的图 G' 中一定存在没有后继节点的强连通分量；由贪心策略可知，求解强连通分量后，可以直接选择没有后继节点的强连通分量中包含原图节点个数最少的作为答案。

3.2 伪代码

Algorithm 3: 最小闭合子图问题

Input: 有向图 $G =< V, E >$，节点数量 n

Output: 闭合子图中至少应包含的节点数量

function $main(G =< V, E >, n)$**:**

    // 获取G中所有强连通分量构成的集合

    $R \leftarrow SCC(G)$;

    // 计算强连通分量之间的连边

    $E' \leftarrow \{< s_u, s_v > | < u, v >\in E, u \in s_u, v \in s_v\}$;

    // 计算强连通分量的后继节点数量

    $out[s_i] \leftarrow \{< s_i, s_u > | < s_i, s_u >\in E'\}.size()$;

    // 初始化答案变量为节点数量n

    $ans \leftarrow n$;

    for $r \in R$ **do**

        if $out[r] = 0$ **then**

            $ans \leftarrow min\{ans, r.size()\}$;

        end

    end

    return ans;

end

Algorithm 4: 求强连通分量

Input: 有向图 $G =< V, E$

Output: 强连通分量集合

function $SCC(G =< V, E >)$**:**

    $R \leftarrow \{\}$;

    // 构造反向图

    $G^R \leftarrow G.reverse$;

    // 对反向图G^R进行DFS

    $L \leftarrow DFS(G^R)$;

    $color[1...V] \leftarrow WHITE$;

    for $i \leftarrow L.length()$ **downto** 1 **do**

        $u \leftarrow L[i]$;

        if $color[u] = WHITE$ **then**

            $L_{scc} \leftarrow DFS - Visit(G, u)$;

            $R \leftarrow R \cup set(L_{scc})$;

        end

    end

    return R;

end

Algorithm 5: DFS

Input: 有向图 $G =< V, E >$

Output:

```
function DFS(G =< V, E >):
    colorlinks[v] ← GRAY;
    for u ∈ V do
        color[v] ← WHITE;
    end
    for u ∈ V do
        if color[v] = WHITE then
            L' ← DFS − Visit(G, v);
            L.append(L');
        end
    end
    return L;
end
```

Algorithm 6: $DFS - Visit$

Input: 有向图 $G =< V, E >$，顶点 v

Output: 按照完成时刻从早到晚排列的顶点 L

```
function DFS − Visit(G =< V, E >, v):
    colorlinks[v] ← GRAY;
    for w ∈ G.Adj[v] do
        if color[w] = WHITE then
            L ← DFS − Visit(G, w);
        end
    end
    color[v] ← BLACK;
    // 向L结尾追加顶点v
    L.append(v);
    return L;
end
```

3.3 算法复杂度

求解强连通分量的过程是 $O(|V| + |E|)$；贪心选取遍历所有的强连通分量，时间复杂度为 $O(|V|)$。综上总时间复杂度为 $O(|V| + |E|)$。

4 食物链问题

给定一个食物网，包含 n 个动物，m 个捕食关系，第 i 个捕食关系使用 (s_i, t_i) 表示，s_i 捕食者，t_i 被捕食者，根据生物学定义，食物网中不会存在环。

长度为 k 的食物链包含 k 个动物的链：$a_1, a_2, \cdots a_k$，其中，a_i 会捕食 a_{i+1}，一个食物链为最大食物链当且仅当 a_1 不会被任何动物捕食，且 a_k 不会捕食任何动物。

请设计一个高效算法计算食物网中最大食物链的数量，并给出分析过程、伪代码以及算法复杂度。

4.1 分析过程

由题中 n 个动物和 m 个捕食关系，以 n 个动物构造节点集合 V，以 m 个捕食关系构造边集 E，以 (V,E) 构造图 $G=<V,E>$。

最大食物链等价于 G 上的一条路径，其中路径的起点的入度为 0，终点的出度为 0。

由于涉及重叠子问题，因此考虑使用动态规划。

状态定义：$dp[i]$ 表示编号为 i 的节点为路径终点的数量。

状态转移方程：对于前驱节点 $pre[v]=\{u|<u,v>\in E,u\in V\}$，$dp[v]=\sum_{u\in pre[v]}dp[u]$

边界条件：对于所有入度为 0 的点 u，$dp[u]=1$。

遍历顺序：以图 G 拓扑排序顺序进行遍历以确保状态转移时其依赖的状态均已正确计算完毕。

4.2 伪代码

Algorithm 7: 食物链问题

Input: 动物数量 n，捕食关系数量 m

Output: 最大食物链数量

```
function main(n, m):
    // 由动物数量n和捕食关系m构造有向图G =< V, E >
    V ← {1, 2, ...n};
    E ← {< u, v > |存在捕食关系(u, v)};
    // 计算有向图G的拓扑序
    topo ← top_sort(G);
    // 初始化答案变量为0
    ans ← 0;
    // 遍历拓扑序进行动态规划的状态计算与状态转移
    for i ← to n do
        // 获取当前节点
        node ← topo[i];
        pre[node] ← {u| < u, node >∈ E, u ∈ V};
        out[node] ← {u| < node, u >∈ E, u ∈ V}.size();
        dp[node] ← Σ_{u∈pre[node]} dp[u];
        if out[node] = 0 then
            ans ← ans + dp[node];
        end
    end
    return ans;
end
```

Algorithm 8: 拓扑排序

Input: 图 $G =< E, V >$

Output: 图的拓扑序列

```
function top_sort(G =< E, V >):
    // 初始化空队列Q
    Q ← new Queue();
    for u ∈ V do
        if u.in_degree = 0 then
            Q.Enqueue(v);
        end
    end
    // 初始化答案序列为空
    L[] ← [];
    while not Q.is_empty() do
        u ← Q.Dequeue();
        L.add(u);
        for v ∈ G.Adj(u) do
            v.in_degree ← v.in_degree − 1;
            if v.in_degree = 0 then
                Q.Enqueue(v);
            end
        end
    end
    return L;
end
```

4.3 算法复杂度分析

拓扑排序的时间复杂度是 $O(n+m)$；动态规划时每个节点 v 状态转移的复杂度为 $|pre[v]|$，因此动态规划部分的时间复杂度为 $O(n+m)$，总时间复杂度为 $O(n+m)$。

5 景区限流问题

已知某市有热门景区 m 个，可以表示为集合 $S = \{s_1, s_2, ..., s_m\}$，有游客 n 人，可以表示为 $T = \{t_1, t_2, ..., t_n\}$。每名游客有 k 个心仪的景区，但每个景区最多容纳 l 人，问最多有多少人能够去到自己心仪的景区，并给出分析过程、伪代码以及算法复杂度。

5.1 分析过程

本题是一个最大二分图匹配类型的问题。

首先，将每个景点 s_i 用 l 个不同的节点 $s_{i_1}, s_{i_2}, ..., s_{i_l}$ 取代；然后，将每个游客 t_u 偏好景点 s_u 元组用 l 个元组 $(t_u, s_{u_1}), (t_u, s_{u_2}), ...(t_u, s_{u_l})$ 取代。

因此，相当于有 n 个游客，$l \times m$ 个不同的景区，每个游客偏好 $k \times l$ 个景区，每个景区只能容纳 1 个人。

5.2 伪代码

Algorithm 9: 景区限流问题

Input: 景区数量 m，景区集合 $S = \{s_1, s_2, ..., s_m\}$，游客数量 n，游客集合 $T = \{t_1, t_2, ...t_n\}$，游客与景区偏好关系矩阵 H

Output: 能去自己心仪的景区的最多人数

function $main(m, S, n, T, H)$**:**
  // 将每个景区以l个不同景区取代
  for $i \leftarrow 1$ **to** m **do**
    $S \leftarrow S - \{s_i\} \cup \{s_{i_j} | j = 1, 2, ..., l\}$;
  end
  // 将游客与新景区的偏好构建为边集E
  $E \leftarrow \{\}$;
  for $i \leftarrow$ **to** n **do**
    $E.append(\{(t_i, s) | (t_i, s_{ij}) \in H \ and \ s = s_{ij_p} \ and \ p = 1, 2, ..., k\})$;
  end
  // 初始化匹配数组$matched$
  for $t \in T$ **do**
    $matched[t] \leftarrow NULL$;
  end
  for $s \in S$ **do**
    // 初始化$color$数组
    for $t \in T$ **do**
      $color[u] \leftarrow WHITE$;
    end
    $DFS - Find(G =< V, E >, v)$;
  end
  // 初始化答案变量
  $ans \leftarrow 0$;
  for $i \leftarrow 1$ **to** n **do**
    if $matched[i]$ *not null* **then**
      $ans \leftarrow ans + 1$;
    end
  end
  return ans;
end

Algorithm 10: 深度优先搜索寻找交替路径

Input: 图 $G =< V, E >$，节点 v

Output: 是否存在从顶点 v 出发的交替路径

```
1  function DFS − Find(G =< V, E >, v):
       // 深度优先搜索寻找以顶点v出发的交替路径
2      for u ∈ G.Adj[v] do
3          if color[u] = BLACK then
4              continue;
5          end
6          color ← BLACK;
7          if matched[u] = NULL or DFS − Find(G, matched[u]) then
8              matched[u] ← v;
9              return true;
10         end
11     end
12     return false;
13 end
```

5.3 算法复杂度

由上述分析和伪代码知，$|V| = n + ml, |E| = nkl$，因此算法复杂度为 $O(|V| \times |E|) = O(n^2kl + nmkl^2)$