计算机学院《算法设计与分析》第四次作业

20373363 李子涵

December 17, 2022

1 P、NP、NPC 问题

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. P 类问题为 NP 类问题的真子集。

无法判定。P 问题在多项式时间内可解，所以也多项式时间内可验证，$P \subseteq NP$。但是 $NP \subseteq P$ 无法判断，NP 是否等于 P 还是开放性问题。

2. 如果假设 $P \neq NP$，则 NP 完全问题可以在多项式时间内求解。

错误，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。

3. 若 SAT 问题可以用复杂度为 $O(n^9)$ 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。SAT 属于 NP 完全问题，如果 SAT 问题可以在多项式时间解决，则存在 NP 完全问题多项式时间可解，则所有 NP 完全问题为多项式可解。

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。虽然 SAT 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，则输出-1。请给出分析过程、伪代码以及算法复杂度。

2.1 分析过程

求无权有向图的最短路径，考虑广度优先搜索 BFS 算法。此题相对于基础 BFS 算法的区别在于:

1. 判定点是否已被遍历时，对于与前驱节点的不同边的颜色，需要分别考虑。故将点标记为**已遍历**并记录父节点信息时，需要按照边的**两种颜色分别记录**。

2. 判定后继节点是否可以被搜索的条件，除了此后继节点未被遍历，还需要考虑是否与上一条边满足颜色交错。所以每个点入队时，还需要额外记录节点与其前驱节点的边的颜色。此外，由于从起点出发时没有前驱节点及入边，故可以选择红色和蓝色两种颜色的边，所以初始化队列时需要两次加入起点，标记不同颜色。

由于广度优先搜索的队列性质，可使用贪心策略：第一次搜索到终点 v_y 且与其父节点符合颜色交错条件时，即可返回此最短路径；当广度优先搜索结束且未返回路径时，说明不存在这样的颜色交错路径，返回-1。

2.2 伪代码

Algorithm 1: 颜色交错最短路问题的广度优先算法

Input: 无权有向图 $G(V, E)$，起点 v_x，终点 v_y

Output: v_x 到 v_y 的最短路径

function $shortestPath(V, E, v_x, v_y)$**:**

  // 分别初始化红边和蓝边指向的节点的三个辅助数组，红色边指向的下标为 0，蓝色边指向的下标为 1

  for $v \in V$ **do**

    // 遍历状态数组$has[\ ]$为 0 表示未遍历，1 表示已遍历

    $has[v][0], has[v][1] \leftarrow 0, 0;$

    $pred[v][0], pred[v][1] \leftarrow NULL, NULL;$

    $dist[v][0], dist[v][1] \leftarrow \infty, \infty;$

  end

  $has[v_x][0], has[v_x][1], pred[v_x][0], pred[v_x][1], dist[v_x][0], dist[v_x][1] \leftarrow 1, 1, -1, -1, 0, 0;$

  // 源点入队

  $Q.Enqueue((v_x, 0)), Q.Enqueue((v_x, 1));$

  while 等待队列 Q 不为空 **do**

    // $(u, lastColor)$当前处理顶点 u 及其指向 u 的边的颜色

    $(u, lastColor) \leftarrow Q.Dequeue();$

    for $v \in G.Adj[u]$ **do**

      // $nextColor$是 u 到 v 的边的颜色

      $nextColor \leftarrow\ (< u, v > .color\ ==\ red)\ ?\ 0 : 1;$

      // 成功搜索条件：顶点没有被 $nextColor$ 的边检索过、检索符合颜色交错条件

      if $has[v][nextColor] == 0$ **and** $nextColor \neq lastColor$ **then**

        $has[v][nextColor] \leftarrow 1;$

        $pred[v][nextColor] \leftarrow u;$

        $dist[v][nextColor] \leftarrow dist[u][lastColor] + 1;$

        // 贪心策略：检索到终点返回，否则当前节点继续入队

        if $v == v_y$ **then**

          return $minPath(pred, v, nextColor), dist[v][nextColor];$

        end

        $Q.Enqueue((v, nextColor));$

      end

    end

  end

  return $-1;$

end

Algorithm 2: 输出最短颜色交错路径

Input: 父节点数组 $pred[\ ][2]$，起点 v_x, 终点 v_y，指向终点边的颜色 $color$

Output: v_x 到 v_y 的最短路径数组 $path[\ ]$

function $minPath(pred[\ ][2], v_x, v_y, color)$**:**
    let $path[\ ]$ **be new array**;
    $v \leftarrow v_y$;
    $path[0] \leftarrow v$;
    $path.length \leftarrow 1$;
    while $pred[v][color] \neq -1$ **do**
        $v \leftarrow pred[v][color]$;
        $color \leftarrow 1 - color$;
        $path[path.length++] = v$;
    end
    $path[path.length++] = v_x$;
    return $path.reverse()$;
end

2.3 时间复杂度分析

状态个数 $O(V)$, 每次搜索新状态时枚举当前状态出发的有向边, 总时间复杂度 $T(|V|,|E|) = O(|V| + |E|)$。

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 V' 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：$\{v3\}, \{v0, v1, v2, v3, v4\}, \{v2, v4\}$。最小的闭合子图仅包含 1 个顶点，为 $v3$。请给出分析过程、伪代码以及算法复杂度。

3.1 分析过程

首先考虑简单情况，假设 G 为有向无环图 DAG，则必存在出度为 0 的点为闭合子图，且由贪心策略可知，此出度为 0 的点即为最小闭合子图。因为其他出度不为 0 的点，均需要将其后继节点加入此节点的闭合子图。

对于可能存在环路的有向图，对图 G 计算强连通分量，使用强连通分量构造新的有向无环图 G'，将强连通分量作为 G' 的节点，E 中跨越两个强连通分量之间的边作为 G' 的边。

新图 G' 为有向无环图，只需要找到 G' 中出度为 0 的节点，这些节点代表的强连通分量中，包含最少原图节点的即为最小闭合子图。

3.2 伪代码

Algorithm 3: 最小闭合子图

Input: 有向图 $G(V, E)$

Output: 最小闭合子图的节点数

function $minGraph($ 有向图 $G(V, E))$**:**

  $\{s_1, s_2, ..., s_k\} \leftarrow scc(G)$;

  $V' = \{s_1, s_2, ..., s_k\}$;

  $E' = \{< s_a, s_b > \mid < u, v > \in E, u \in s_a, v \in s_b\}$;

  $ans = |V|$;

  for $s_u \in V'$ **do**

    $out \leftarrow |\{< s_u, s_v > \mid < s_u, s_v > \in E'\}|$;

    if $out == 0$ **then**

      $ans \leftarrow min\{ans, |s_u|\}$;

    end

  end

  return ans;

end

Algorithm 4: 计算强连通分量

Input: 有向图 $G(V, E)$

Output: 强连通分量

function $scc(G)$**:**

  $R \leftarrow \{\ \}$;

  $G^R \leftarrow G.reverse()$;

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

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

  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: 反向图执行 DFS

```
Input: 有向图 G(V, E)
Output: 反向图执行 DFS
function DFS(G):
    let color[1..|V|], L[1..|V|] be new arrays;
    for v ∈ V do
        color[v] ← WHITE;
    end
    for v ∈ V do
        if color[u] == 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: 按完成时刻从早到晚的顶点 L
function DFS − Visit(G, v):
    color[v] ← GRAY;
    for w ∈ G.Adj[v] do
        if color[w] == WHITE then
            L ← DFS − Visit(G, w);
        end
    end
    color[v] ← BLACK;
    L.append(v);
    return L;
end
```

3.3 时间复杂度分析

求解强连通分量复杂度 $O(|V| + |E|)$，贪心策略遍历所有强连通分量复杂度 $O(|V|)$，总时间复杂度 $O(|V| + |E|)$。

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_t 捕食者，t_i 表示被捕食者，根据生物学定义，食物网中不会存在环。

长度为 k 的食物链指包含 k 个动物的链：$a_1, a_2, ..., a_k$，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

n 个动物作为节点 V，m 个捕食关系作为边 E，构造有向无环图 $G(V,E)$。最大食物链可以表示为图上的一条路径。路径起点包含所有入度为 0 的点，终点包含所有出度为 0 的点。

求解路径数量可以使用深度优先搜索计算，注意到一个节点的所有后继捕食关系可能被前驱节点多次搜索，即存在重复的子问题，故考虑提高性能可以使用动态规划来缩减子问题规模。

易知捕食关系中不存在平行边，故考虑到达一个节点的所有路径数，只需要将所有前驱节点的路径数累加。动态规划设计如下：

1. 状态设计：使用 dp[i] 表示编号为 i 的节点为路径终点的路径数量
2. 状态转移：对于前驱节点 $pre[v]=\{u| <u,v> \in E, u\in V\}, dp[v]=\sum_{u\in pre[v]} dp[u]$。
3. 边界条件：所有路径起点（入度为 0 的点）的状态的路径数量为 0
4. 目标状态：所有路径终点（出度为 0 的点）的状态路径数目之和

再考虑遍历顺序，计算一个节点需要获取其所有前驱节点的状态，故遍历顺序使用图 G 的拓扑排序顺序即可。

4.2 伪代码

Algorithm 7: 食物链问题——动态规划求解

Input: 动物 n, 捕食关系 $m, (s_i, t_i)$

Output: 食物链数量

function $foodChain(n, m, (s_i, t_i))$**:**
  n **个动物作为节点** V m **个捕食关系** (s_i, t_i) **作为边** E**，构造有向无环图** $G(V,E)$;
  $topo \leftarrow topoSort(G)$;
  $ans \leftarrow 0$;
  for $i \leftarrow 0$ **to** n **do**
    $pre[topo[i]] \leftarrow \{u| <u, topo[i]> \in E, u \in V\}$;
    $out[topo[i]] \leftarrow |\{u| <topo[i], u> \in E, u \in V\}|$;
    $dp[topo[i]] \leftarrow \sum_{u\in pre[topo[i]]} dp[u]$;
    if $out[topo[i]] == 0$ **then**
      $ans \leftarrow ans + dp[topo[i]]$;
    end
  end
  return ans;
end

Algorithm 8: 拓扑排序

Input: 有向无环图 $G(V, E)$

Output: 顶点拓扑排序

function $topoSort(G)$**:**
    初始化空队列 Q，拓扑排序结果数组 ans;
    for $v \in V$ **do**
        if $v.in\_degree == 0$ **then**
            $Q.Dequeue()$;
        end
    end
    while $not Q.is\_empty()$ **do**
        $u \leftarrow Q.Dequeue()$;
        $ans.append(u)$;
    end
    for $v \in G.Adj(u)$ **do**
        $v.in\_degree \leftarrow v.in\_degree - 1$;
        if $v.in\_degree == 0$ **then**
            $Q.Enqueue(v)$;
        end
    end
    return ans;
end

4.3 时间复杂度分析

状态个数 $O(n)$，每个状态转移的复杂度为 $O(|pre[v]|)$，总时间复杂度为 $= O(m+n)$。

5 景区限流问题问题

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 1 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。

对于游客与景区的偏好关系，用 H 表示，则 H 可以表示如下形式，其中 (t_u, s_v) 表示游客 t_u 偏好景点 s_v。

5.1 分析过程

这是一个二分图最大匹配问题，修改约束条件为每个游客顶点至多关联一个景区顶点，每个景区顶点至多关联 l 个游客顶点。仍然采用匈牙利算法，依次检测左侧顶点，不断寻找交替路径进行增广：相邻景区顶点存在未达到最大游客量的景区顶点，直接匹配；如果相邻景区顶点都已达到最大游客量的匹配，则从已达到最大游客量匹配的多个游客顶点尝试寻找交替路径，增广成新匹配。

5.2 伪代码

Algorithm 9: 景区限流问题——二分图匹配

Input: 游客 T，景区 S，偏好关系 H，游客 t_u 偏好景点 s_v:(t_u, s_v)，每个景区最大游客量 l

Output: 到心仪景区游客的最大数量

```
function hungarian(T, S, H, l):
    游客 T，景区 S，偏好关系 H，游客 t_u 偏好景点 s_v : (t_u, s_v) 构造二分图 G(T, S, H);
    let matched[1..|S|][1..l], color[1..|S|] be new arrays;
    for u ∈ S do
        matched[u] ← { };
    end
    for v ∈ T do
        for u ∈ S do
            color[u] ← 0;
        end
        DFS − Find(G, color, matched, l, v);
    end
    ans ← 0;
    for u ∈ S do
        ans ← ans + matched[u].length;
    end
    return ans;
end
```

Algorithm 10: 寻找交替路径

Input: 二分图 $G(T, S, H)$, 匹配数组 $matched[\,][l]$，辅助数组 $color[\,]$, 景区最大游客量 l，顶点 v

Output: 是否存在从顶点 v 出发的交替路径

```
function DFS − Find(G, color, matched, l, v):
    for u ∈ G.Adj[v] do
        if color[u] == l then
            continue;
        end
        color[u] ← color[u] + 1;
        if matched[u].length < l) then
            matched[u].append(v);
            return true;
        end
        for t ∈ matched[u] do
            if DFS − Find(G, t) then
                matched[u].replace(t, v);
                return true;
            end
        end
    end
    return false;
end
```

5.3 时间复杂度分析

二分图遍历游客顶点进行匹配，总状态数为游客数目 n。每个状态中，初始化 $color$ 数组复杂度 $O(m)$,$DFS-Find$ 中每次搜索的规模是 kl。所以匹配的总复杂度为 $O(n*(m+kl))$

初始化 $matched$ 数组和计算 $matched$ 数组的复杂度都是 $O(m)$。

所以总时间复杂度为 $O(nm+nkl)$。