算法与复杂性 第四次课程作业

UNIkeEN

2023 年 3 月 29 日

问题解答

1. 对于给定的二叉树，求其最小深度，即从根节点到最近的叶子的距离

> **解**
>
> 1. 使用深度优先搜索，搜索到叶结点后与当前最小值比较，得到最小深度，需要遍历整棵树。
> 2. 使用广度优先搜索，第一个遍历到的叶结点具有最小深度，最坏情况下需遍历整棵树。
>
> 设树共有 N 个结点，时间复杂度均为 $O(N)$，但使用广度优先搜索通常更优。

2. 设 G 是有向无环图，其所有路径最多含 k 条边，设计线性时间算法，将所有顶点分为 k+1 组，每一组中任意两个点之间不存在路径

> **解** DAG 中一定存在入度 0 的节点和出度为 0 的结点，最长路径出现在入度为 0 和出度为 0 的结点之间。
>
> 1. 首先取出 DAG 中出度为 0 的各个结点（遍历邻接表，若某一顶点指向边表的顶点为空，则取之），这些结点之间一定不存在路径，归为一组
> 2. 再次遍历邻接表，将剩余结点边表中已取出的结点删除。
> 3. 重复步骤 1，再次取出出度为 0 的各个结点，归为第二组，依此类推直至所有结点均被取出
>
> 最长路径长度为 k，则最长路径上有 $k+1$ 个结点，可知最终可取出 $k+1$ 组结点。
>
> 也可每次寻找入度为 0 的一批结点（类似拓扑排序），当图以出边邻接表给出时，算法的时间复杂度均为 $O(|V|+|E|)$，符合线性时间条件。

Algorithm 1 DAG 分组

```
1: procedure GroupDAG(G)
2:     创建一个空列表 groups
3:     while 存在未被分组的顶点 do
4:         创建一个空集合 group
```

```
        for 每个未被分组的顶点 v do
            if v 的出度为 0 then
                将 v 加入 group
            end if
        end for
        将 group 加入 groups
        for 未被分组的顶点 v do
            从 v 的邻接表中移除已分组的顶点
        end for
    end while
    return groups
end procedure
```

3. 给定一个有向图 $G(V,E)$，其中边的权重可以是 x, $2x$，或者 $3x$ (x 是一个正整数)，$O(|V|+|E|)$ 时间计算从源 s 到其它各个顶点的最小成本路径

解 使用 Dijkstra 算法不满足时间复杂度要求。本问边的权重情况较少，可以使用 BFS。

1. 建立数组 dist 记录源点到各点的当前最短距离，初始时 dist[s]=0，其余为无穷大
2. 从源点 s 起对图做广度优先搜索
3. 设当前结点为 u，遍历不做已遍历标记，而是遍历 u 的所有邻接结点 v（无论是否曾遍历过），如果 dist[u] + 边权重 < dist[v]，则更新 dist[v] = dist[u] + 边权重

完成遍历后，数组 dist 包含从源顶点 s 到其它各个顶点的最小成本路径长度。由于我们使用了广度优先搜索，使用邻接表存储图时算法的时间复杂度为 $O(|V|+|E|)$

还有一种方法是增加虚拟点，如果某边 (u,v) 权重为 $2x$，则添加虚拟点 t，将原边变为 (u,t) 和 (t,v)。同理，如果某边 (u,v) 权重为 $3x$ 则增加两个虚拟点。添加完后进行 BFS，遍历到某点（不含虚拟点）经历 BFS 的次数乘以 x 即为源点到该点的距离。

4. 给定一组英文单词，检查这些单词是否可以按照如下规则重新排列形成一个圆圈，该规则为：两个单词 X 和 Y，如果 X 的最后一个字符与 Y 第一个字符相同，则 Y 可以接在 X 后面

解 一种方式是将单词作为边构造图。

1. 首先按输入单词的首尾字母建立有向图，若首尾字母的结点已经存在，则连边，若首尾字母的结点不存在则增加结点后连边，允许重边和自环。
2. 如只需检验重排方案是否存在，则检查每个结点的入度和出度是否相等（欧拉回路的判定条件）且不只含有自环，若是则存在。
3. 如需输出重排方案，则使用深度优先搜索寻找图中的欧拉回路即可。

如使用邻接表存储图，时间复杂度为 $O(|V|+|E|)$，其中 E 为单词数量。

另一种方式是将单词作为点构造图，如果 Y 可接在 X 后面，则增加有向图 (X,Y)，然后使用深度优先搜索寻找图中的哈密顿回路。

5. 寻找有向图的根顶点（根顶点指图中的所有其它顶点都可以从根顶点到达）。一个图可以有多个根顶点（此时只要找到一个即可），也可能没有根顶点

解 图中可能存在环，故不能直接使用拓扑排序。

一种简单方式是将每个结点都作为深度或广度优先搜索的初始结点，如果可以遍历所有结点，则为根结点，算法终止。

另一种算法如下：

1. 建立一个布尔型数组（若使用邻接表存储，具体实现可将顶点表改为结构体），大小为顶点个数，用以记录各个顶点是否遍历过，初始值全部设为 false。
2. 遍历顶点，对于尚未被遍历的顶点，将其作为开始结点执行深度优先搜索，搜索过程只沿未遍历的顶点进行，将遍历到的点修改为已遍历（true）。
3. 重复第 2 步，若某次遍历后全部结点均已被遍历，将该次遍历的开始结点作为有向图的根顶点。
4. 重置布尔数组，使用深度优先或广度优先搜索检查根顶点是否能够到达其他所有顶点，若可以则返回，若不可以则没有根顶点。

Algorithm 2 寻找有向图的根顶点

```
1:  procedure FindRoot(G)
2:      创建一个布尔型数组 visited，大小为顶点个数，初始值设为 false
3:      root ← 空值
4:      for 每个顶点 v do
5:          if not visited[v] then
6:              root ← v
7:              DFS(G, v, visited)
8:          end if
9:      end for
10:     重置 visited 数组为 false
11:     reachable ← DFS(G, root)
12:     if 所有顶点都在 reachable 中 then
13:         return root
14:     else
15:         return “没有根顶点”
16:     end if
17: end procedure
```

6. 求 $n * m$ 棋盘上任意两点之间马能够走的最短路径长度

解 构造图 $G(E, V)$，其中 E 为棋盘上各格点，V 表示两格点可由马一步走到。（若马可从 u 一步走到 v，则 $(u, v) \in V$。

从需求的源点开始，构图与广度优先搜索同时进行，广度优先搜索得到最短路径长度。

若需输出任意两点之间最短路径长度的解，则在构造完图后使用 Floyd 算法即可。若是先任意确定起终点，则从起点开始构建马可一步走到的图，构图并广度优先搜索直至遇到终点。

7. 给定连通无向图 G，以及 3 条边 a, b, c，在线性时间内判断 G 中是否存在一个包含 a 和 b 但不含 c 的闭链

解

1. 删除边 c，在剩余的图中判断是否存在包含 a, b 的回路
2. 在剩余的无向图中划分双连通分支（这一方法是满足线性时间的），若 a, b 在同一个双连通分支里，则存在一条包含 a, b 但不包含 c 的回路

8. 设计线性时间算法求树的最大匹配

解 对树后序遍历，进行动态规划。

记函数 $dp(v, b)$ 为以 v 为根节点的子树中最大匹配的边数，其中 b 为

- 当 $b = 1$ 时，v 存在一条与子结点直接相连的边在最大匹配中
- 当 $b = 0$ 时，与 v 子结点相连的边都不在最大匹配中

对所有叶结点 w，其 $dp(w, 0) = dp(w, 1) = 0$

遍历到某点 v 时，设其子结点为 $u_1, u_2 ... u_n$ 有

$$dp(v, 0) = \sum_{i=1}^{n} dp(u_i, 1) \tag{1}$$

$$dp(v, 1) = \max \begin{cases} dp(w_1, 0) + dp(w_2, 1) + ... + dp(w_n, 1) + 1 \\ dp(w_1, 1) + dp(w_2, 0) + ... + dp(w_n, 1) + 1 \\ ... \\ dp(w_1, 1) + dp(w_2, 1) + ... + dp(w_n, 0) + 1 \end{cases} \tag{2}$$

由以上状态转移方程一直计算到根结点 v_0，解为 $\max(dp(v_0, 0), dp(v_0, 1))$

时间复杂度为 $O(|V| + |E|)$。

9. 无向图 G 的顶点覆盖是指顶点集合 U，G 中每条边都至少有一个顶点在此集合中。设计线性时间算法为树寻找一个顶点覆盖，并且使该点集的规模尽量小

解

1. 计算所有结点的度

2. 对于所有度数为 1 的节点，首先标记其相邻的点都在 U 中，然后删除与 U 中的点直接相邻的所有边

3. 重复上述步骤，直至所有边都被删除

选择合适的根节点所需时间复杂度为 $O(|E|)$，BFS 的时间复杂度为 $O(|E|+|V|)$，整体时间复杂度为 $O(|E|+|V|)$。

Algorithm 3 寻找最小顶点覆盖

```
1:  procedure FindU(G)
2:      创建一个空集合 U
3:      计算图 G 中所有顶点的度
4:      while 图 G 中存在边 do
5:          for 每个顶点 v do
6:              if v 的度数为 1 then
7:                  将 v 的相邻顶点 u 加入集合 U
8:                  删除与 u 直接相邻的所有边
9:                  更新图 G 中顶点的度数
10:             end if
11:         end for
12:     end while
13:     return U
14: end procedure
```