计算机学院《算法设计与分析》第四次作业

December 28, 2024

1 对于下面的每个描述，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。

1. 如果一个问题是 NP-Hard 问题，那么它一定是 NP 问题。
2. P 问题 $\cap$ NPC 问题 $= \emptyset$。
3. 若 TSP 问题无法在多项式时间内被解决，则 3-SAT 问题也无法在多项式时间内被解决。
4. 对某问题 $X \in$ NP 而言，若可以证明规约式 3-SAT $\leq_p X$，则 $X \in$ NPC。

1.1 解答

1. 该说法是**错误**的。NP-Hard 问题的定义是所有 NP 问题都可以在多项式时间内规约到它，但 NP-Hard 问题不要求自身属于 NP 类，即它不一定能够在多项式时间内验证解。因此，NP-Hard 问题不一定是 NP 问题，故该说法是错误的。
2. 该说法是**无法判断**的。在计算复杂性理论中，P 类问题是指可以在多项式时间内由确定性算法解决的问题，而 NP 类问题则是那些解可以在多项式时间内由非确定性算法验证的问题。NP 完全（NP-Complete，NPC）问题是 NP 类中最复杂的问题，具有以下特性：1. 它们属于 NP 类；2. 所有 NP 问题都可以在多项式时间内规约到它们。如果一个问题同时属于 P 类和 NPC 类，那么所有 NP 问题都可以在多项式时间内解决，这意味着 P=NP。然而，P 是否等于 NP 仍是一个未解决的问题。在当前理论框架下，如果认为 P≠NP，则 P 类问题与 NPC 问题的交集是 $\emptyset$，即 P 类问题与 NPC 问题没有交集；否则有 P=NP=NPC。故此说法无法判断。
3. 该说法是**错误**的。在计算复杂性理论中，3-SAT 问题是 NP 完全（NP-Complete，NPC）问题，而旅行商问题（TSP）是 NP 困难（NP-Hard）问题。虽然两者都与 NP 类问题相关，但它们之间的复杂性并不直接可比。具体而言，3-SAT 问题的求解可以在多项式时间内验证解的正确性，但 TSP 问题并不一定具备这一特性。因此，就算 TSP 问题没有多项式时间内解决，3-SAT 仍有可能在多项式时间内解决。
4. 该说法是**正确**的。根据计算复杂性理论的定义，若某问题 $X \in$ NP，并且可以证明 3-SAT 问题可以通过多项式时间规约到问题 X（即 3-SAT $\leq_p X$），则说明 X 至少与 3-SAT 问题一样难，同时它也满足 NP 完全问题的两个条件：

   (a) $X \in$ NP，即其解可以在多项式时间内验证；

   (b) 所有 NP 问题（包括 3-SAT）都可以规约到它。

   因此，问题 X 是 NP 完全问题（即 $X \in$ NPC）。

2 最小点集问题

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 $u\in U$，满足图中存在一条从 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。

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。

2.1 问题分析

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，就意味着已经选择了其对应的强连通分量。

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 DAG 图。

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任选一个点加入点集 U 即为最小的点集 U。

故我们先利用 $kosaraju$ 算法通过 2 遍 dfs 求出强连通分量，再依次遍历 DAG 图中所有入度为 0 的点即可。

2.2 时间复杂度分析

求强连通分量的过程为 $O(|V|+|E|)$ ，而选出 DAG 中入度为 0 的点时间复杂度为 $O(|V|)$ 。故总的时间复杂度为 $T(|V|,|E|)=O(|V|+|E|)$。

2.3 最小点集伪代码

Algorithm 1: kosaraju(V, E)

Input: 给定的图 $G(V, E)$，及其反图 $G2(V, E)$

Output: 强连通分量集 s

```
1  function dfs1(u):
2      vis[u] ← true;
3      for v : G[u] do
4          if !vis[v] then
5              dfs1(v);
6          end
7      end
8      s.push(u);
9  end
10 function dfs2(u):
11     color[u] ← sccCnt;
12     for v : G2[u] do
13         if !color[v] then
14             dfs2(v);
15         end
16     end
17 end
18 function kosaraju():
19     sccCnt ← 0;
20     for i ← 1 to n do
21         if !vis[i] then
22             dfs1(i);
23         end
24     end
25     for i ← n to 1 do
26         if !color[s[i]] then
27             sccCnt ← sccCnt + 1;
28             dfs2(s[i]);
29         end
30     end
31 end
```

Algorithm 2: minPointSet(V, E)

Input: 给定的图 $G(V, E)$

Output: 所选出的最小点集 ans

```
kosaraju();      //求出 G 图的所有强连通分量 s_1, s_2, ··· , s_k
V ← s_1, s_2, ··· , s_k;
E ← < s_u, s_v > | < u, v >∈ E, u ∈ s_u, v ∈ s_v;
in[s_i]←|< s_u, s_i > | < s_u, s_i >∈ E |;
for i ← 1 to k do
    if in[s_i] = 0 then
        任取 s_i 中一点加入 ans;
    end
end
return ans
```

3 最小环问题

给定一个包含 n 个点的无向图，保证无重边无自环。边权使用矩阵 $w_{i,j}$ 表示 $(w_{i,j} > 0)$。

请设计一个高效的算法，计算图中最小环（最少包含三个节点）的最小边权和，请描述算法的核心思想，给出该算法伪代码并分析时间复杂度。

3.1 问题分析

为了计算最小环的权重，可以将一个环拆开。例如，对于环 $a-b-\cdots-c-a$，可以拆分为 $a-b$，$a-c$，以及 $b-\cdots-c$。这样仅需枚举节点 a，以及与节点 a 相连的节点 b 和 c，该环的权重为 $w_{a,b}+w_{a,c}+d_{b,c}$。

由于题目要求环最少包含 3 个节点，需要保证路径 $b-\cdots-c$ 不经过节点 a。

故我们利用 *Floyd* 算法，在进行三重循环的枚举时，先固定最外层节点编号 k，枚举内层循环 i 和 j 时，判断 k 是否与 i, j 直接相连，如果直接相连，则更新答案 $ans = \min(ans, w_{i,k} + w_{j,k} + d_{i,j})$。

更新完 ans 之后再判断 i, j 是否有经过 k 的最短路径，更新 $d_{i,j}$。

3.2 时间复杂度分析

三重循环，时间复杂度为 $O(n^3)$。

3.3 最小环伪代码

Algorithm 3: mincycle$(n, w[i][j])$

Input: 节点数 n，边权矩阵 $w_{i,j}$

Output: 一个正整数 ans，最小边权和

```
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if i = j or i = k or k = j then
                continue;
            end
            ans ← min(ans, w_{i,k} + w_{j,k} + d_{i,j});
        end
    end
    for i ← 1 to n do
        for j ← 1 to n do
            d_{i,j} ← min(d_{i,j}, d_{i,k} + d_{k,j});
        end
    end
end
return ans
```

4 最短路经过次数问题

给定一张由 n 个点组成的无向带权图 G，其所有边权都是正数，现指定一个起点 S 和一个终点 T。对于图中的每一个节点 i，请求出从 S 到 T 的所有最短路径中，一共有几条最短路径经过了节点 i。

请你设计一个高效算法求解本问题，描述算法的核心思想，并给出算法伪代码和对应的时间复杂度分析。

4.1 问题分析

有题，我们要求的其实是一个单源最短路径中经过各点的次数，由此我们很容易想到 $Dijkstra$ 算法，对于 $Dijkstra$ 算法，我们是否有办法求出单源到各点的最短路径数量呢，答案是肯定的。

我们知道 $Dijkstra$ 算法中，节点的入队顺序是按照 dist 数组从小到大依次入队的。也就是说，如果从 S 到 i 有多条最短路径，那这些最短路径上 i 的前驱节点一定是比 i 先入队的（排除边权位负的情况）。所以，我们只需要在每次松弛操作的时候，判断并更新最短路径数量即可。即：

$$count[neighbor] = \begin{cases} count[current],\ dist[current] + weight_{current,neighbor} < dist[neighbor] \\ count[neighbor] + count[current],\ dist[current] + weight_{current,neighbor} = dist[neighbor] \end{cases}$$

4.2 算法核心步骤

解决了上述问题之后，我们知道，只要分别以 S, T 为源，进行一次 $Dijkstra$ 算法，之后对于每个点 i，判断是否有 $distS[i] + distT[i] = dist_S[T]$，有：

$$res[i] = \begin{cases} countS[i] \times countT[i],\ distS[i] + distT[i] = dist_S[T] \\ 0 \end{cases}$$

4.3 时间复杂度分析

本题时间复杂度取决于 $Dijkstra$ 算法的复杂度，即 $O((n+m)\log n)$。

4.4 最短路经过次数伪代码

Algorithm 4: mod_Dijkstra($GraphG, S$)

Input: $GraphG, S$

Output: 最短路径距离序列 $dist$，最短路径数量序列 $count$

```
dist ← ∞ × n;
count ← 0 × n;
dist[source] ← 0;
count[S] ← 1;
pq ← [(0, S)];    //创建优先队列 (距离, 节点)
while pq do
    d, u ← heapq.heappop(pq);
    if d > dist[u] then
        continue;
    end
    for v, weight ∈ G[u] do
        newDist ← dist[u] + weight;
        if newDist < dist[v] then
            dist[v] ← newDist;
            count[v] ← count[u];
            heapq.heappush(pq, (newDist, v));
        end
        else if newDist = dist[v] then
            count[v] ← count[v] + count[u];
        end
    end
end
```

Algorithm 5: pathCounts($GraphG, S, T$)

Input: $GraphG, S, T$

Output: 路径计数序列 ans

```
distS, countS ← mod_Dijkstra(graph, S);
distT, countT ← mod_Dijkstra(graph, T);
for node u ∈ G do
    if distS[u] + distT[u] = distS[T] then
        ans[u] ← countS[u] × countT[u];
    end
end
return ans
```

5 扩张树问题

给定一个包含 n 个点的带权无向树 T，保证边权均为整数，要求你在 T 的基础上添加 $\frac{n(n-1)}{2}-n+1$ 条带权无向边，得到图 G，并且满足以下条件：

1. G 是一张完全图。

2. T 是 G 的唯一最小生成树。

3. G 的边权均为整数。

（完全图的边数为 $\frac{n(n-1)}{2}$，树的边数为 $n-1$，故添加的边数为两者之差）

求添加的 $\frac{n(n-1)}{2}-n+1$ 条无向边的边权之和的最小值。请你设计一个高效算法求解本问题，描述算法的核心思想，并给出算法伪代码及对应的时间复杂度分析。

5.1 问题分析

定义 $G(x,y)$ 为树上 x 到 y 路径中边权的最大值。

结论：非树边的最小边权

在最优方案里，对于任意一条非树边 (x,y)，其边权的最小值就是 $G(x,y)+1$。我们从两个方面来证明这个结论：

(x, y) 的边权不能更小

显然，如果 (x,y) 的边权比 $G(x,y)+1$ 还小，我们可以选择断开那条边权最大的边，然后连上 (x,y)。这同样也是一棵生成树，而且生成树的边权和小于等于之前的边权和。此时，生成树就不唯一了。因此，(x,y) 的边权不能比 $G(x,y)+1$ 更小。

(x, y) 的边权是合法的

(x,y) 的边权设置为 $G(x,y)+1$ 是合法的，即不会导致生成树的权值和变小或者生成树的唯一性变化。对于单条边的情况，这个结论是显然的。但是否可能出现有多条边同时加入生成树，导致结论发生变化呢?

我们考虑将多条边逐条加入原生成树，并相应地移除某些边，只有在强行删边之后使得生成树的某些 $G(x,y)$ 变小才会不合法。假设我们加入边 (u',v')，删掉边 (u,v)，与 u,v 同侧的 $G(x,y)$ 不会受到影响，由 $Kruskal$ 算法我们知道，只有在分别位于 u,v 两侧的点的 $G(x,y)$ 才会受影响，而 (u',v') 边权大于 (u,v)，所以对于所有的 $G(x,y)$，经过加边删边操作后，只会变大或不变，因此，这样的加边无法造成不合法的情况，所以将 (x,y) 的边权设置为 $G(x,y)+1$ 是合法的。

将边按照边权从小到大的方式排序，每条边对于总边权的贡献是是它连接的两棵子树的节点数的乘积，所有贡献加起来即为答案。我们用采用类 $Kruskal$ 算法来统计答案。

5.2 时间复杂度分析

由于我们用采用类 $Kruskal$ 算法来统计答案，故时间复杂度为 $O(n\log n)$（维护并查集的时间开销）。

5.3 扩张树伪代码

Algorithm 6: tree($n, edge$)

Input: 给定的最小生成树 $edge$

Output: 最小边权和 ans

```
1  pre[1..n] and num[1..n] ← 0;
2  function Init():
3      for i ← 1 to n do
4          pre[i] ← i;
5          num[i] ← 1;
6      end
7  end
8  function Find(x):
9      father ← x;
10     while pre[father] ≠ father do
11         father ← pre[father];
12     end
13     while x ≠ father do
14         now ← pre[x];
15         pre[x] ← father;
16         x ← now;
17     end
18     return father;
19 end
20 Init();
21 Sort(edge[1..n − 1]) by w in ascending order; // egde 中 x，y 为节点，w 为权重
22 ans ← 0;
23 for i ← 1 to n − 1 do
24     u ← edge[i].u;
25     v ← edge[i].v;
26     ans ← ans + (edge[i].w + 1) × (num[Find(u)] × num[Find(v)] − 1);
27     num[Find(v)] ← num[Find(v)] + num[Find(u)];
28     pre[Find(u)] ← Find(v);
29 end
30 return ans
```