算法设计与分析第四次作业

杨博文 21373037

2023 年 12 月 27 日

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。

1. 任何 NP 完全问题都不存在多项式时间内的解法。

无法判定，现在还在研究。

2. P 类问题是 NP 类问题的真子集。

无法判定，若 $P \neq NP$，则 P 是 NP 的真子集，若 $P = NP$，则 P 和 NP 相等。

3. 对某问题 $X \in NP$ 而言，若可以证明规约式 $3 - SAT \leq pX$，则 $X \in NPC$。

正确。

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，反例：SAT 是 NPC 问题，但是对于任意满足 2-SAT 条件的输入，都可以在多项式时间内求解。

2. 节点寻找问题

现存在一张由 n 个节点，m 条边组成的有向图 G。图 G 中所有顶点按照 1 到 n 进行编号，第 i 条有向边起点为 s_i，终点为 t_i。令函数 $F(i)$ 表示由顶点 i 出发能够到达的最小的顶点编号（每一个顶点都可以到达它自己）。请设计一个算法计算图 G 中每一个顶点 i 对应的 $F(i)$ 值，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。

1. 核心思想

很显然我们可以通过 n 次 DFS 来实现要求，时间复杂度为 $O(n*(n+m))$，过于低效，因此我们使用强连通分量降低复杂度。我们使用反转图的强连通分量算法，第一遍对反向图的 DFS 不变，在第二遍对正向图的 DFS 中，实时维护每一个强连通分量里的最小值，值得注意的是，我们可能会遍历到已经遍历过的其他强连通分量节点，对此我们需要额外比较当前强连通分量 A 的最小值和已经遍历过的强连通分量 B 的最小值，那么每当我们遍历完一个强连通分量后，该强连通分量的最小值被唯一确定。那么第二次 DFS 后可以得到的每个强连通分量的最小值，那么内部所有节点的最小值均为该值。

2. 伪代码

Algorithm 1 节点寻找

Input: $G(V, E)$

Output: $F[1...n]$ // 结果数组

$R \leftarrow \{\}$

$G^R \leftarrow G.reverse()$ // 反转图 G

$L \leftarrow DFS(G^R)$

$color[1..n] \leftarrow WHITE$

for $i \leftarrow L.lengthto1$ **do**
    $u \leftarrow L[i]$
    if $color[u] = WHITE$ **then**
        $Scc_{min} \leftarrow MAX$
        $L_{scc}, Scc_{min} \leftarrow DFS - Visit - 2(G, u, Scc_{min})$
        for $k \in L_{scc}$ **do**
            $F[k] \leftarrow Scc_{min}$
        end
    end
end

Algorithm 2 DFS

Input: $G(V, m)$

$color[1..n], L[1..n]$

for $v \in V$ **do**
    $color[v] \leftarrow WHITE$
end

for $v \in V$ **do**
    if $color[v] = WHITE$ **then**
        $L^{'} \leftarrow DFS - Visit(G, v)$
        $L.append(L^{'})$
    end
end

return L

Algorithm 3 $DFS-Visit$

Input: G, v

Output: 按完成时刻从早到晚排列的顶点序列 L

$color[v] \leftarrow GRAY$

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

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

    $L \leftarrow DFS-Visit(G, w)$

  end

end

$color[v] \leftarrow BLACK$

$L.append(v)$ **return** L

Algorithm 4 $DFS-Visit-2$

Input: G, v, Scc_{min}

Output: 该强连通分量的最小值，以及该强连通分量内部节点

$color[v] \leftarrow GRAY$

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

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

    if $w < Scc_{min}$ **then**

      $Scc_{min} \leftarrow w$ `// 此处的 w 是该节点的编号`

    end

    $L, Scc_{min} \leftarrow DFS-Visit-2(G, w, Scc_{min})$

  else

    if $color[w] = BLACK$ & $F[w] < Scc_{min}$ **then**

      $Scc_{min} \leftarrow F[w]$

    end

  end

end

$color[v] \leftarrow BLACK$

$L.append(v)$

return L, Scc_{min}

3. 时间复杂度分析

第一遍对反转图 DFS 的时间复杂度是 $O(m+n)$，第二遍对原图 DFS 并实时计算结果的时间复杂度为 $O(m+n)+O(n)=O(m+n)$，因此该算法时间复杂度为 $O(m+n)$。

3. 二分图判定问题

二分图是指一个无向图 $G=(V,E)$, 它的顶点集可被分成两个互不相交子集，且同一个子集中任何两顶点间都没有边相连。(换言之，G 为二分图，当且仅当存在两个集合 V1, V2 满足 $V1 \cap V2 = V, V1 \cap V2 = \phi$，E 中每条边都连接了 V1 中某个点与 V2 中某个点。) 请设计一个基于广度优先搜索 (BFS) 的算法来判断无向图 G 是否为二分图，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。

1. 核心思想

根据二分图的定义，我们需要将所有节点划分为两个不相交的集合，集合内的点不能有边相连。我们可以通过 BFS，将一个节点和他所有邻接的节点被分配到不同集合，如此一旦出现矛盾，则不为二分图，若成功 BFS 完毕，则为二分图。为简化操作，我们为每个节点设置标记来表示它位于哪个集合。

2. 伪代码

Algorithm 5 二分图判定问题

Input: $G(V, E)$

Output: $True/False$

$visit[1...V] \leftarrow 0, Queue \leftarrow \{\}, flag[1...V] \leftarrow -1$

随便选择一起始点 $start$

$flag[start] \leftarrow 0$

$Queue.Enqueue(start)$

while *not* $Queue.is\_empty()$ **do**
    $u \leftarrow Q.Dequeue()$
    $visit[u] \leftarrow 1$
    for $v \in G.Adj(u)$ **do**
        if $visit[v] = 0$ **then**
            $Queue.Enqueue(v)$
            if $flag[u] = 1$ **then**
                $flag[v] = 0$
            else
                $flag[v] = 1$
            end
        else
            if $flag[u] = flag[v]$ **then**
                return $False$
            end
        end
    end
end
return $True$

3. 时间复杂度分析

该算法仅需要一遍改良版 BFS，时间复杂度为 $O(V + E)$。

4. 道路改建问题

在某一座城市的郊区存在 n 个村庄，它们通过一个由 m 条双向可通行的公路组成的道路网络彼此互通，第 i 条公路连接村庄 s_i 和 t_i，长度为 w_i 公里。为了支持各个村庄的经济发展，当地政府计划将其中的一些公路修建为高速公路。但是受限于地理因素，政府只会选择 n 1 条公路进行改建，使得完工后的高速公路网络满足：

1. 这 n 1 条高速公路恰好可以使得这些村庄彼此互通。

2. 最长的若干条（可以是一条或者多条）高速公路长度恰好为 K。

但是，修建高速公路的过程不能改变原公路的长度，因此，在现有的道路网络上可能无法找到合适的 n 1 条待改建公路，所以当地政府计划先改变一些现有公路的长度。将现有的任意一条公路扩充 1 公里或者缩短 1 公里都需要 1 单位成本，请设计一个算法帮忙计算最少需要多少单位成本改建现有的道路网络才能使得这个高速公路修建工程顺利开工。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。

1. 核心算法

题目要求一，需要我们找到一颗生成树；要求二，需要我们最长的生成树边恰好为 K。我们可以想到通过求解最小生成树来解决此问题，进一步观察最小生成树与该问题的异同点，如果找到了一棵最小生成树，此时最小生成树内的最长边 L 与 K 有下述三种情况：

1. L $<$ K: 此时我们寻找一个更为接近 K 的边来构成最短生成树，因此我们在非 MST 边中寻找一条更接近 K 的边，若找不到则只能将 L 扩充为 K，若找到则将边 $L^{'}$ 加入树中，此时 n 条边会产生环，我们删除一条在环上且不为 $L^{'}$ 的边即可，此时的单位成本等于 $L^{'}$ 与 K 的绝对差值，由于我们不要求给出新生成树，因此找到 $L^{'}$ 即可;

2. L $=$ K: 自然满足情况;

3. L $>$ K: 由于我们求得便是最小生成树，此时不存在比 L 更小的边，替换后仍满足生成树的要求，因此必须将 L 缩短为 K。自然的此时也许会暴露出新的最长边大于 K，因此我们需要将所有大于 K 的边缩短为 K。

我们使用并查集优化克鲁斯卡尔算法。

2. 伪代码

Algorithm 6 道路改建问题

Input: 带权无向图 $G(V, E, W)$

Output: 最小改建成本 $cost$

将边按照权重升序排序

为每个顶点建立不相交集

$T \leftarrow \{\}$

$cost \leftarrow 0, count \leftarrow 0$

for $(u, v) \in E$ **do**
  if $Find - Set(u) \neq Find - Set(v)$ **then**
    $T \leftarrow T \cup \{(u, v)\}$
    $Union - Set(u, v)$
    $count \leftarrow count + 1$
    if $w(u, v) > K$ **then**
      $cost \leftarrow cost + w(u, v) - K$
    end
  end
  if $count = n - 1$ & $w(u, v) < K$ **then**
    $t \leftarrow$ 从当前边直至末尾，二分查找最接近 K 的边的权重
    $cost \leftarrow |w(t) - K|$
    $Break$
  end
end

return $cost$

Algorithm 7 $Union - Set$

Input: 顶点 x, y

$a \leftarrow Find - Set(x)$

$b \leftarrow Find - Set(y)$

if $a.height \leq b.height$ **then**
  if $a.height = b.height$ **then**
    $b.height \leftarrow b.height + 1$
  end
  $a.parent \leftarrow b$
else
  $b.parent \leftarrow a$
end

Algorithm 8 $Find-Set$

Input: 顶点 x

Output: 所属连通分量父节点

while $x.parent \neq x$ **do**
| $x \leftarrow x.parent$

end

return x

3. 时间复杂度分析

我们根据课内所学认为 Kruskal 算法时间复杂度为 $O(|E|log|V|)$，构建 MST 中进行的额外操作无需额外时间复杂度，最后若出现最坏情况 $L_{max} < K$ 时，需要使用 $O(log|E|)$ 时间复杂度去二分查找，根据假设 $|E| = O(|V|^2)$，则 $O(log|E|) = O(2log|V|)$，最后该算法时间复杂度为 $O(|E|log|V|)$。

5. 新最短路径问题

给定一张由 n 个点，m 条有向边组成的有向图 G，其中第 i 条边起点为 s_i ，终点为 t_i，边权为 $w(s_i, t_i)$，保证 $w(s_i, t_i) > 0$，假设 G 中存在一个由 t 个顶点组成的路径 $P = [v_1, v_2, ..., v_t]$。我们在课上学到的有关路径长度的定义为 $dis(P) = \sum_{i=1}^{t-1} w(v_i, v_{i+1})$，即路径上每一条边的边权和。现给出一个新的路径长度定义，如下公式所示：

$$dis(P) = \sum_{i=1}^{t-1} w(v_i, v_{i+1}) - max_{i=1}^{t-1} w(v_i, v_{i+1}) + min_{i=1}^{t-1} w(v_i, v_{i+1})$$

特别的，如果 $t = 1$，那么 $dis(P) = 0$。在这个新路径长度定义下，请设计一个算法计算顶点 1 到图中所有顶点，即点 $1, 2, ..., n$ 的最短路径长度。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。

1. 核心思想

我们首先会想到使用迪杰斯特拉算法，进而观察迪杰斯特拉算法和本题目的异同点。本题目要求我们计算一种新的最短路径，定义为路径长度减去路径中最长边加上路径中最短边。我们首先考虑他的简化版，去掉最短边，仅为路径长度减去路径中的最长边。不难发现节点路径长度减去最长边应小于等于节点路径长度减去任意一条边，我们在进行传统迪杰斯特拉算法的同时，不仅要维护每个节点的路径长度的最小距离，还需要维护每个节点路径长度减去任意一条边的最小距离，这一步应当是最为关键的状态转化步骤，具体来说，由于我们记录了一个节点 u 的 $distu1$(记录传统路径长度最短值), $distu2$(记录路径长度减去任意一条边最短值) 两个状态，那么当我们遍历到其邻接节点 v 时，$distv1$ 仍采用传统方法更新，而 $distv2 = min(distu1, distu2 + w)$，而这是由 u 的节点两个状态两次遍历到 v 节点决定的。那么根据上述不等式，最终求得的新最短路径，自然是路径长度减去最长边的距离，符合要求。那么加上最短边同理，又需要额外维护一个新状态，结合二者情况，又需要额外维护一个新状态，最后我们使用 $dist[v][0/1][0/1]$ 来表示该节点是否减去一条边权值，是否加上一条边权重的最短路径，$dist[v][1][1]$ 为最终所求结果。

2. 伪代码

Algorithm 9 新最短路径问题

Input: $G(V,E,W)$

Output: $dis[v][1][1]$

$Eachof(dist[V][1][1]) \leftarrow -1$

$Eachof(vis[V][1][1]) \leftarrow 0$

初始化优先队列 Q

$dis[start][0][0] \leftarrow 0$

$Q.Enqueue(start,0,0,0)$ // 分别代表节点，状态 1，状态 2，当前最短距离

while *not* $Q.is\_empty()$ **do**
  $u, curS1, curS2, curD \leftarrow Q.Dequeue()$
  if $vis[u][cueS1][curS2] = 1$ **then**
    $Continue$
  end
  $vis[u][curS1][curS2] \leftarrow 1$
  for $(u,v) \in G.Adj(u)$ **do**
    for $i : 0 \rightarrow 2$ **do**
      for $j : 0 \rightarrow 2$ **do**
        $d \leftarrow curD + w(u,v)$
        $s1 \leftarrow curS1, s2 \leftarrow curS2$
        if $i = 1$ & $s1 = 0$ **then**
          $d \leftarrow d - w(u,v)$
          $s1 \leftarrow 1$
        end
        if $j = 1$ & $s2 = 0$ **then**
          $d \leftarrow d + w(u,v)$
          $s2 \leftarrow 1$
        end
        if $dis[v][s1][s2] = -1$ || $dis[v][s1][s2] > d$ **then**
          $dis[v][s1][s2] \leftarrow d$
          $Q.Enqueue(v, s1, s2, d)$
        end
      end
    end
  end
end
return $dis[V][1][1]$ // 遍历打印新最短距离即可

3. 时间复杂度分析

优先队列内是节点的四种状态，因此可以说将传统迪杰斯特拉算法的规模扩大了四倍，本质上该算法的时间复杂度仍为迪杰斯特拉算法的时间复杂度，即 $O(mlogn)$。