第七次算法作业

1. 假设有 n 个职业摔跤手，并且有一个给出竞争关系的 r 对摔跤手的链表。请给出一个时间为 O(n+r)的算法来判断是否可以将某些摔跤手划分为“娃娃脸”型，而剩下的划分为“高跟鞋”型，使得所有的竞争关系均只存在于娃娃脸型和高跟鞋型选手之间。如果可以进行这种划分，则算法还应生成一种这样的划分。

算法步骤:

可以将所有的摔跤选手看为一个点，存在竞争关系意味着两点之间有一条无向边。因此算法可以实现为:

1、 从每个顶点开始进行 BFS，直到覆盖所有的顶点 V
2、 检查每个顶点的距离 V.d，奇数则为娃娃脸，偶数则为高跟鞋
3、 检查每条边是否为娃娃脸与高跟鞋之间的竞争，如果有一条不是返回 false，否则返回 true。

算法伪代码描述为:

```
WRESRLER-PARTITION(G):
for each vertex u ∈ G.V – {s}
    u.color = WHITE
    u.d = INFINITE, U.p = NIL
s.color = GRAY
s.d = 0, s.p = NIL, s.t = ‘BABYFACE’
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v. d = u.d + 1, v.p = u
            if v.d % 2 == 0 //如果为偶数，划分为 BABYFACE，否则 HIGHHEEL
                v.t = ‘BABYFACE’
            else
                v.t = ‘HIGHHEEL’
            ENQUEUE(Q, v)
        else if v.t ≠ NIL && v.t == u.t //检查已有的划分是否正确，不正确则返回 FALSE
            return FALSE
    u.color = BLACK
return TRUE
```

复杂度分析:

执行 BFS 的复杂度为 O(n+r)，划分类型复杂度为 O(n)，判断边复杂度为 O(r)，因此总的复杂度为 O(n+r)。

2. 给定有向图 G=(V, E)，如果对于所有的结点对$\mathbf{u}, \mathbf{v} \in \mathbf{V}$，我们有$\mathbf{u} \rightsquigarrow \mathbf{v}$或者$\mathbf{v} \rightsquigarrow \mathbf{u}$，则 G 是半连通的。请给出一个有效算法来判断图 G 是否是半连通的。证明算法的正确性并分析其运行时间。

算法步骤:

1. 计算图中的强连通分量，调用 STRONGLY-CONNECTED-COMPONENTS(G)
2. 将第一步的强连通子图转变为有向无环分量图 G'
3. 对生成的分量图 G'进行拓扑排序，形成点的线性排序(v_1, v_2, ……, v_{k-1}, v_k,)
4. 检查(v_1, v_2)，(v_2, v_3)，……，（v_{k-1}，v_k）这些边在是否在 G'中，如果在则 G 为半连通图，否则 G 不是半连通图。

算法伪码:

```
VERIFY-SEMI-GRAPH(G)
    STRONGLY-CONNECTED-COMPONENTS(G)
    Convert G to G'
    let v be a new array
    V = TOPOLOGICAL-SORT(G')
    for i = 1 to V.length-1
        if G[V[i]][V[i+1]] ∉ G'
            return FALSE
    return TRUE
```

证明:

因为每个强连接分量图都有 u⇝v 或者 v⇝u，因此需要证明：当且仅当 G'中包含一条路径连接所有的点，图 G 才是半连通的。

（1）假设 G'中存在这样的路径，那么对于任意的点 u，v，它们之间存在一条路径，有 u⇝v 或者 v⇝u，因此该图是半连通图。

（2）如果不能找到这样一条路径，图不是半连通的。设拓扑排序后的点对(u,v)，如果该点对不在 G'中，说明不存在一条路径，使得 u⇝v，而由拓扑排序的定义可知，不存在一条路径，使得 v⇝u，所以改图不是半连通图。

复杂度分析:

第一步的计算强连通分量的复杂度为$\Theta(V+E)$，第二步生成分量图的复杂度为$O(V+E)$，因为最坏情况下分量图有最多 V 个点和 E 条边，所以第三步拓扑排序的复杂度为$O(V+E)$，第四部的复杂度为$O(V+E)$，所以最终算法复杂度为$O(V+E)$。

3. 假定图 G 的一棵最小生成树已经被计算出来，如果在图中加入一个新结点及其相关的新边，我们需要多少时间对最小生成树进行更新?

我们需要$O(V\lg V)$的时间对最小生成树更新。

令 T 为图 G=(V,E)的最小生成树，G'=(V', E')是 G 的子图，令 T'=E-T，那么一定存在一个 G'的最小生成树不包括 T'中的任意一条边。这是因为对于任意的 u, v ∈ V，如果 Kruskal 算法在 G 上运行后这些向量是属于同一集合的，那么在 G' 上运行 Kruskal 算法后仍属于同一个集合。

令 G1=（V1，E1）是 G（V，E）加入一个结点和一条相关的边构成的图，T 是 G 的最小生成树，G2=（V1，E2），E2=T∪（E1 - E）。根据上一步的证明，存在一个 G1 的最小生成树不包括 E-T 的所有边，也就是说存在一个 G1 的最小生成树只包含 T 和 E1-E 中的边，也就是 E2 中的边。因此 G2 的最小生成树也是 G1 的最小生成树。

计算 G2 的最小生成树，使用 Prim 算法和斐波那契堆，V1 = V+1， E2<=2V+1，因此建立 G2 的时间复杂度为 O（V），计算最小生成树的算法复杂度为$\mathrm{O(E2 + V1lgV1) = O(VlgV)}$，因此总分的算法复杂度为$\mathrm{O(VlgV)}$

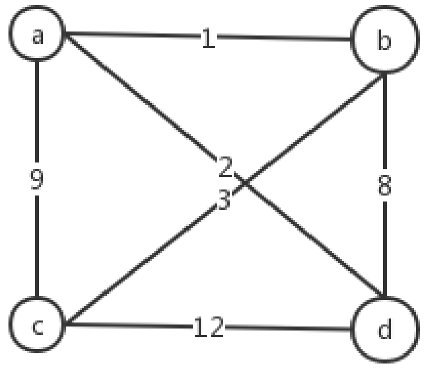
4. 次优最小生成树

a. 证明：最小生成树是唯一的，但次优最小生成树不是唯一的。

首先证明：最小生成树是唯一的。

假设存在两个权重相等但不同的最小生成树 $T_1=\{a_1, a_2,\cdots,a_{n-1}\}$，$T_2=\{b_1,b_2,\cdots,b_{n-1}\}$，其中$\mathrm{a_1} < \mathrm{a_2} < \cdots < a_{n-1}, b_1 < b_2 < \cdots < b_{n-1}$，设 k 为第一个下标使得$\mathrm{a}_k \neq \mathrm{b}_k$，不失一般性，设$\mathrm{a}_k < \mathrm{b}_k$，将$\mathrm{a}_k$加入$\mathrm{T_2}$，则$\mathrm{T_2}$中出现一个环，在这个环中必然存在$\mathrm{b}_i > \mathrm{a}_k$，将$\mathrm{b}_i$从图中移除，形成一棵有更小权重值的生成树，这与$\mathrm{T_2}$是最小生成树相矛盾。因此，最小生成树是唯一的。

然后证明：次优最小生成树不是唯一的。



如图所示，上图的次优最小生成树是 ac，ad，ab 或者 ab，bc，bd，因此次优最小生成树不是唯一的。

b. 设 T 为 G 的一棵最小生成树，证明：图 G 包含边$\mathbf{(u, v) \in T}$和边$\mathbf{(x, y) \notin T}$，使得$\mathbf{T - \{(u, v) \cup (x, y)\}}$是 G 的一棵次优最小生成树。

因为最小生成树有$|\mathrm{V}| - 1$条边，任意的次优最小生成树一定至少有一条边不在最小生成树中，如果只有一条边不在最小生成树中，那么这种情况下就是$\mathrm{T - \{(u, v) \cup (x, y)\}}$是 G 的次优最小生成树。

因此只需证明如果替换最小生成树的两条边（或两条边以上），那么生成的一定不是次优最小生成树。令 T 为 G 的最小生成树，假设存在一个次优最小生成树$\mathrm{T'}$至少替换了 T 的两条边，令$\mathrm{(u, v)}$为$\mathrm{T - T'}$中权重最小的一条边，把$\mathrm{(u, v)}$加入 $\mathrm{T'}$中会形成一个环，这个环必然包含了$\mathrm{T' - T}$中的边，记为$\mathrm{(x, y)}$。使用$\mathrm{(u, v)}$替换$\mathrm{(x, y)}$，因为$\mathrm{(u, v)}$的权重小于$\mathrm{(x, y)}$小于的权重，因此这个生成树的权重小于$\mathrm{T'}$，这与$\mathrm{T'}$是次优最小生成树矛盾。所以如果至少替换最小生成树的两条边，那得到的一定不是次优最小生成树。

综上所述。图 G 必然包含边$\mathrm{(u, v) \in T}$和边$\mathrm{(x, y) \notin T}$，使得$\mathrm{T - \{(u, v) \cup (x, y)\}}$是 G 的一

棵次优最小生成树。

c. 设 T 为 G 的一棵最小生成树，对于任意两个结点$u, v \in V$，设$max[u, v]$表示数 T 中从结点 u 到结点 v 的简单路径上最大权重的边，请给出一个$O(V^2)$时间复杂度的算法来计算$max[u, v]$。

算法伪代码:

```
BFS-MAX(G, T, w)
let M be a new array
for each u ∈ G.V
    Q = Ø
    ENQUEUE(Q, s)
    while Q ≠ Ø
        x = DEQUEUE(Q)
        for each v ∈ G.adj[x]
            if M[u,v]==NULL && v ≠ u
                if x == u || weight(x, v) > M[u,x]
                    M[u, v] = (x, v)
                else M[u, v] = M[u, x]
                ENQUEUE(Q, v)
return M
```

d. 给出一个高效算法计算 G 的次优最小生成树。

由 b 可知，把最小生成树的一条边替换可能会生成一个次优最小生成树。由 c 可知，如果我们能直接判断[u,v]之间权重的最大值 M[u,v]，我们只需要找到一条边$(u, v) \notin T$并且使得$weight(max[u, v]) - weight(u, v)$最小。因此算法为:

1. 计算最小生成树
2. 计算 c 中定义的 M 数组
3. 找到一条边$(u, v) \notin T$使得$weight(max[u, v]) - weight(u, v)$最小
4. 次优最小生成树为$T - \{max[u, v]\} U \{(u, v)\}$

算法伪代码为:

```
FIND-SUB-MINIMUM-SPANNING-TREE
T = PRIM(G, w, r)
M = BFS-MAX(G, T, w)
min = INFNITY
result = NULL
for each edge in G ∉ T
    temp = w(M[u,v])-w(u,v)
    if temp < min
        min = temp
        result = (u,v)
return T-{max[u, v]}U{(u, v)}
```

第一步算法复杂度为 O(V^2)，第二步算法复杂度为 O(V^2)，第三步算法复杂度为 O(V^2)，因此总的算法复杂度为 O(V^2)。