Problem Set 8

1: 遍历整个 adjacency-list，记录每个 vertex 的 in-degree，同时以 in-degree 为指标将 vertices 分组 (e.g. in-degree=0 的一组，in-degree=1 的一组……)。每次 remove 一个 in-degree 为 0 的 vertex，不妨记为 u。记 adjacency-list 为 Adj。之后将 Adj[u] 中的 vertices 的 in-degree 减一，同时移到对应 in-degree 的组中。

不断 repeat 上述过程，直至 in-degree=0 的组体为空。

2: (a) 首先执行 SCC 过程寻找到所有 k 个 SCC，不妨记为 $C_1, C_2, \cdots, C_k$。然后再遍历所有边 E，如果发现边 (u,v)，且 $u \in C_i, v \in C_j$ $(i \neq j)$，同时在 component graph 中没有边 (C_i, C_j)，则添加边 (C_i, C_j)。

(b) 我们执行 the second DFS 的理论依据是：the node with maximum finish time (DFS on G^R) is guaranteed to be in source GCC in G^R / sink GCC in G。

Professor Bacon 的想法错用了如下相反命：

Do DFS on G^R, the node with minimum finish time is in sink GCC in G^R.

反例：

G: A → B ⇄ C

⇒ finish time: $C.f < A.f < B.f$

G^R: A(4/5) ← B(1/6) ⇄ C(2/3)

此时若从 C 开始 DFS，则 B, C 将在同一个 SCC 中，显然错误。

3: (a) 首先，对 G 做 Topo Sort，注意到 Topo Sort 后最末的 vertex (记为 v) 的 out-degree=0，即 cost[v]=P_v。我们定义一个临时数组 temp。

```
FillCost():
    temp=[]
    for u∈V:
        temp[u]=P_u  //初始化
    TopoSort(G)
Repeat:
    v=last vertex after TopoSort
    cost[v]=temp[v]
    for each (u,v)∈E:
        temp[u]=min(temp[u],temp[v])
        remove (u,v) from E
    remove v from V
Until V is empty
```

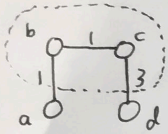
(b) 把该问题通过 SCC 方法化为(a).

首先 Compute SCC 并作出 Component Graph (Problem 2 (a) 给出 O(|V|+|E|) 的方法). 假设 SCC 为 $C_1, C_2, \cdots, C_k$. 对每个 C_i, 在其中 DFS 遍历一遍, 找到 $m_i = \min(\text{minimum}(p_u))$, ~~将 C_i 中所有 vertices 的 price 更新为 m_i~~. 此时相应的定义 component graph 中 C_i 的 price 为 m_i.

显然, component graph 为 DCG, 其中每一个 vertex 均有一个 price m. 在 component graph 上做问题(a)过程. 定义 cost'[C_i] is the cheapest SCC reachable from C_i, 填法 cost'[ ] 后, 即可填满 cost[ ]: for each $u \in C_i$, cost[u] = cost'[C_i].

4.0

5: (a) No. 考虑如下反例: 10



$G=(V, E)$

$V=\{a, b, c, d\}$

$E=\{(a,b), (b,c), (c,d)\}$ (权重 1, 1, 3)

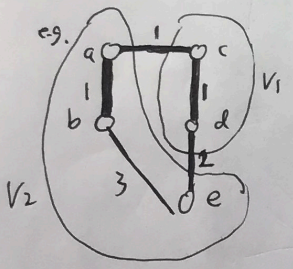
不妨: $A=\{(b,c)\}$

$S=\{b,c\}$, $V-S=\{a,d\}$

显然 (c,d) 是 safe edge crossing $(S, V-S)$, 但是 (c,d) 并不是 light edge.

(b) No. ~~如果 (a,b), (c,d) 两条~~

如果 cross V_1, V_2 的边中有多条属于 MST, 该算法必会丢弃部分, 此时不会生成 MST.



显然左图的 MST $=(V', E')$, where

$V'=\{a, b, c, d, e\}$, $E'=\{(a,b), (a,c), (c,d), (d,e)\}$.

如果初始划分 $V_1=\{a,b,e\}$, $V_2=\{c,d\}$, 最后生成的树的边集 $E''=\{(a,b), (a,c), (c,d), (b,e)\}$.

6: (a) Proof: 使用反证法. 不妨假设 G 有 2 个 MST, T_1, T_2. 它们的路径权重和均为 S. 互异的.

10

显然, 必 $\exists$ 一条边 $e_1 \in T_1$, s.t. $e_1 \notin T_2$. 将 e_1 添加到 T_2 中, 由于 T_2 为 MST. 则加入 e_1 后的 T_2 必然会形成一个环. 于是我们可以在这个环中找到一条边 $e_2 \neq e_1$, ~~s.t. 去掉 e_2 后~~ 且 (不妨假设 $e_2.w \geq e_1.w$), 去掉 e_2. 有树 $\{e_1\} \cup T_2 - \{e_2\}$ 也是 MST, 且其权重 ~~相等~~ 小于 S.

$\Rightarrow$ 这与 T_2 为 MST 矛盾

(b) 假设T''是G'的MST。~~并且T''≠T'，于是显然~~

T'不是G'的MST，

于是显然有T''.w < T'.w.

设 T''' = T - T'，~~显然~~ 显然 T''∪T''' 是原图G的连通树

⇒ (T''∪T''').w = T''.w + T'''.w < T'.w + (T-T').w = T.w

~~⇒ T''∪T''' 为原图的MST，又 T''∪T''' ≠ T~~

⇒ 与T为G的MST矛盾

⇒ T'是G'的MST

7: (a) 由原题意易知，~~题中~~ 边集E-T非空。取其中的一条 边记为 $e=(x,y)$，显然 $(x,y)\notin T$，将 (x,y) 添加到T中，形成的新图中必存在环。从该环中移除一条边 $(u,v)\neq(x,y)$，得到新的~~是~~生成树 $T''=T-\{(u,v)\}\cup\{(x,y)\}$，其权重 $w(T'')=w(T)-w(\{(u,v)\})+w(\{(x,y)\})$。

记 $\Delta w=w(\{(x,y)\})-w(\{(u,v)\})$ ⇒ 使 Δw 最小的 (x,y)、(u,v) 即能使 T'' 为 minimum spanning tree 的两条边。⇒ 存在 G中

(b) 对每个 vertex S 分别调用 COMPUTEMAX(T, S)。

```
COMPUTEMAX(T,S):
FIFOqueue Q:
Q.enque(S)
While !Q.empty():
    u = Q.dequeue()
    for (each edge (u,v) in T:
        if u==S:
            max(S,V) = (u,v).weight
        else:
            max(S,V) = maximum(max(S,u), (u,v).weight)
        Q.enque(v)
```

可以看出，单次COMPUTEMAX(T,S)时间复杂度为 O(|V|) ⇒ 总 time complexity 是 O(|V|²)

(c) ①首先用 Prim's/Kruskal's 算法在 O(|E|log|V|) 时间内算出图 G 的 MST，记为 T。

②利用(b)的算法求出 max(u,v) for all (u,v) pairs

③对 E-T 中每一条边 (m,n)，计算 (m,n).weight - max(m,n).

④取 (m,n) = minimum$_{(m,n)\in E-T}$ [(m,n).weight - max(m,n)] (依据(a)的结论，使ow最小)

⑤于是，记 max(m,n) 对应的 edge 为 (a,b)，于是 T-{a,b}∪{(m,n)} 为 second-best minimum spanning tree.