Homework 8

蘇俊杰、劉韋成、曾守瑜

Problem1

(7.38) Given a directed acyclic graph $G = (V, E)$, find a simple (directed) path in G that has the maximum number of edges among all simple paths in G. The algorithm should run in linear time.

Problem1

Method 1: from top to bottom

1. Start from the vertex v whose indegree $= 0$.
2. Go through all the edges (v, w) and update the maximum number of edges of w.
3. Subtract 1 from $w.indegree$. If it becomes 0, put w into the vertices we will go through next.

Remember: A **directed acyclic graph (DAG)** may has more than 1 vertices whose indegree is 0.

Problem1

Algorithm MaximumEdgesPath($G = (V, E)$)

```
begin
    initialize v.length := 0 & v.pre := null for every vertex v in V;
    initialize v.indegree for every vertex v in V;
    Q := empty queue;
    S := empty stack;
    for all vertices v in V do
        if v.indegree = 0 then
            put v into Q;
    repeat
        remove v in Q;
        for all edge(v, w) do
            w.indegree := w.indegree − 1;
            if v.length + 1 > w.length then
                w.length := v.length + 1;
```

Problem1

```
                    w.pre := v;
                if w.indegree = 0 then
                    put w into Q;
        until Q is empty

        v := v of v.length whose length is the largest in V;
        while v.pre is not null do
            push v into S;
            v := v.pre;
        push v into S;

        return S;
    end
```

Time complexity: O(|V|+|E|).

Problem1

Method 2: from bottom to top

1. Use the conception of DFS. Start from the vertex v whose indegree $= 0$.
2. Run DFS from v. Update the maximum number of edges of w in postWORK for (v, w).

Remember: A **directed acyclic graph (DAG)** may has more than 1 vertices whose indegree is 0.

Problem1

Algorithm MaximumEdgesPath($G = (V, E)$)

begin
initialize $v.length := 0$ & $v.post := null$ for every vertex v in V;
initialize $v.indegree$ for every vertex v in V;
$Q :=$ empty queue;
for all vertices v in V **do**
if $v.indegree = 0$ **then**
$DFS\_MEP(G, v)$;

$v :=$ v of $v.length$ whose length is the largest in V;
while $v.post$ is not $null$ **do**
put v into Q;
$v := v.post$;
put v into Q;

return Q;
end

Problem1

procedure DFS_MEP(G, v)

begin
mark v;
for all edges (v, w) **do**
if w is unmarked **then**
DFS_MEP(G, w);
if $w.length + 1 > v.length$ **then**
$v.length := w.length + 1$;
$v.post :=$ w;
end

Time complexity: O(|V|+|E|).

Problem2

Dijkstra's algorithm for single-source shortest paths assumes that every edge of the input graph has a nonnegative weight. Suppose we are given a graph with negative weights on some edges, where the minimum weight of the edges is $-c$ for some $c > 0$. If we add c to the weight of every edge, then we obtain a new graph with nonnegative edge weights. We could then apply Dijkstra's algorithm to find the shortest paths for the new graph and thereafter subtract c from each edge of a path. Does this give us the shortest paths for the original graph? Please explain your answer.

Problem2

No, this method cannot give us the shortest paths for the original graph.

There are two kinds of explanations:

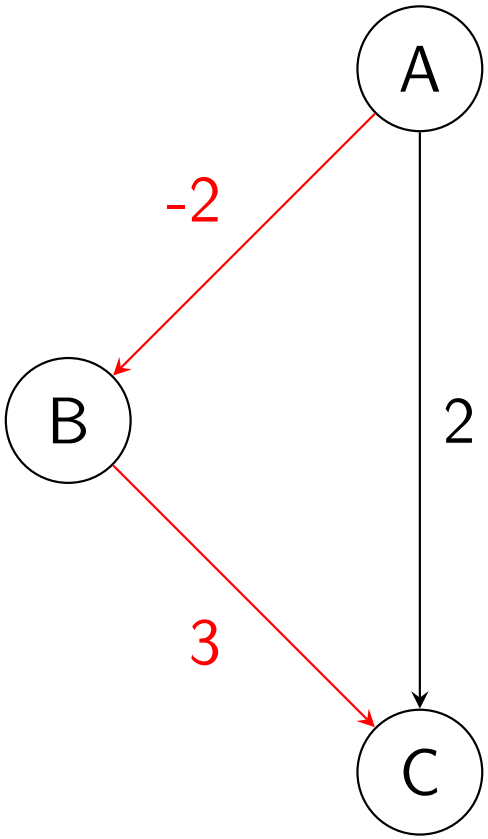
Problem2

Explanation 1:

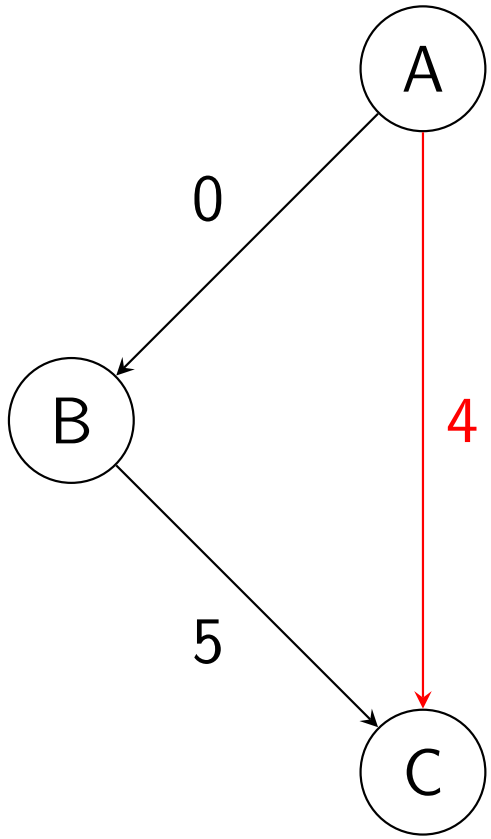
Each path has different number of edges. If there are two paths P_1 and P_2 whose number of edges are N_1 and N_2 separately. We assume that the weights of P_1 and P_2 are W_1 and W_2 separately and P_1 is the shortest path for the original graph ($W_1 < W_2$). So the weights of P_1 and P_2 of the the new graph with nonnegative edge weights we obtain after add c to the weight of every edge will be $W_1 + cN_1$ and $W_2 + cN_2$.

We cannot guarantee that $W_1 + cN_1 < W_2 + cN_2$ still holds, which means that if N_1 is large enough than N_2, P_2 may become the new shortest path in the new graph.

Problem2



Old graph



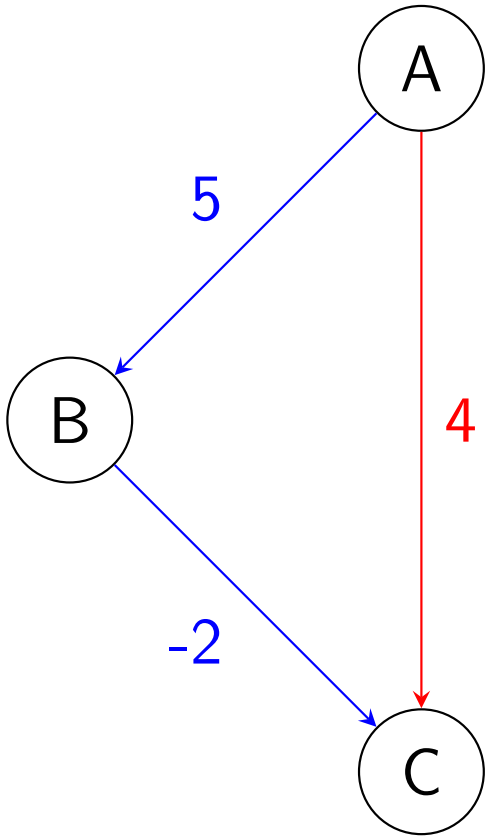
New graph (add 2)

Problem2

Explanation 2:

When we run Dijkstra's algorithm, there is a precondition guarantees that each time we search a new edge, the weight will not be reduced. So we can promise that when we search the weight in ascending order, the shortest path we find is always the shortest at that moment.

Problem2



When we run Dijkstra's algorithm, we will obtain the red path, but the shortest path is blue path.

Problem3

(7.9) Prove that if the costs of all edges in a given connected graph are distinct, then the graph has exactly one unique minimum-cost spanning tree.

Problem3

- 利用反證法
- **若存在有兩顆相異的 minimum-cost spanning trees MST_1(G) 與 MST_2(G)**
- MST_1(G) 與 MST_2(G) 的 edges 按 cost 大小排序
  - MST_1(G): e1, e2,$\cdots$, MST_2(G): e1$'$, e2$'$,$\cdots$
- 假設 e_i be the minimum cost edge in MST_1 but not MST_2(G).
- 假設 e_i' be the minimum cost edge in MST_2(G) but not MST_1 .
- 假設 $e_i < e_i$', then MST_2(G)$\cup \{e_i\}$ 會產生 cycle
- Let e_k' (that is in MST_2(G)) be the maximum cost edge of the cycle

Problem3

- $e'_1 < e'_2 < \ldots < e_i < \ldots < e'_i \ldots$
- 因為每個 edge 的 cost 都是相異的且 e_k' 是 maximum cost edge
- e_k' 不屬於任何 minimum-cost spanning trees ，但 e_k' 出現在 MST_2(G) 中
- MST_2(G) 不是 minimum-cost spanning tree
- 這違反假設 **若存在有兩顆相異的 minimum-cost spanning trees MST_1(G) 與 MST_2(G)**
- 故不可能有兩棵 minimum-cost spanning trees

Problem4

What is wrong with the following algorithm for computing the minimum-cost spanning tree of a given weighted undirected graph (assumed to be connected)?

> If the input is just a single-node graph, return the single node. Otherwise, divide the graph into two subgraphs, recursively compute their minimum-cost spanning trees, and then connect the two spanning trees with an edge between the two subgraphs that has the minimum weight.

Problem4

- 題目的演算法並沒有告知如何分割圖形
- 在切割圖形的時候，此演算法沒有確保切開的線沒有切到重要的線
- 此演算法也沒有告知如何處理切到相同 weight 的邊時如何處理
- 將所有的邊做排序再去想如何切割和串連
- 題目需要解釋 !!!! 有些人只畫一個圖

Problem5

(7.61) Let $G = (V, E)$ be a connected weighted undirected graph and T be a minimum-cost spanning tree (MCST) of G. Suppose that the cost of one edge $\{u, v\}$ in G is changed (*increased* or *decreased*); $\{u, v\}$ may or may not belong to T. Design an algorithm to either find a new MCST or to determine that T is still an MCST. The more efficient your algorithm is, the more points you will be credited for this problem. Explain why your algorithm is correct and analyze its time complexity.

Problem5

給定圖 G 以及 minimum-cost spanning tree T
若是 G 的某個 edge 的 cost 產生變化
如何找出新的 minimum-cost spanning tree ?

Problem5

首先很明顯地：

	變大	變小
在 T 中	要檢查	不用檢查
不在 T 中	不用檢查	要檢查

我們還需要 Lemma：
將 spanning tree 上沒有連線的兩點連在一起，則會形成一個 cycle
而對 minimum-cost spanning tree 而言，加上另外一個 edge 形成 cycle 時，這個 edge 一定是 cycle 當中 cost 最大的 (若否，則去除另外一個 edge 才能得到真正的 minimum-cost spanning tree)

Problem5

若 tree 上的 edge cost 升高
假設我們去除了這個 edge，就會將樹分成兩塊，分別連接 V_1, V_2 兩個頂點集合
找出所有連接著 V_1, V_2 的 edges，選出其中 cost 最小的

Problem5

Is this a minimum-cost spanning tree?
選定一個不在新樹中的 edge
若它與其他 edge 形成的 cycle 不包含新的 edge
那麼可以用原本的 mcst 套入 Lemma 得知這個 edge 不好；
若 cycle 包含新 edge
代表這個 edge 也是連接兩塊小樹的 edge
而這個 edge 之所以當初沒被選中就是因為 cost 比較大

Problem5

那這個 cycle 上最重的真的就是這個任意 edge 嗎？
設被移除掉的 edge 1 號在 V_1 的端點為 v_{11}，新添加上去的 edge 2 號在 V_1 的端點為 v_{12}，隨意挑的 edge 3 號在 V_1 的端點為 v_{13}
在 spanning tree 上不會有 cycle，所以這三個點之間的關連可以是：三點不在同一條路徑上、三點在同一個路徑上且三個點各在中間，共四種可能

Problem5

三點不在同一路徑
假設三點路徑分岔於 w 點
從原本的 MCST 來看，加入 2 號 edge 形成的 cycle （會包含 1 號 edge，因為它們都連接著 V_1, V_2 ）當中，2 號 edge 會最重
代表 2 號 edge 的 cost 大於 $v_{11} - w - v_{12}$ 路徑上所有的 edges
同理 3 號 edge 的 cost 大於 $v_{11} - w - v_{13}$ 的所有 edges
現在在新樹上加上 3 號 edge，它與 2 號 edge 形成的 cycle 當中，V_1 部分的 edges 是 $v_{12} - w - v_{13}$ 路徑上的 edges
已知 $v_{12} - w$ 這堆的 cost 會小於 2 號 edge，而 2 號 edge 的 cost 小於 3 號 edge 的 cost，$w - v_{13}$ 這堆的 cost 也會小於 3 號 edge
所以 3 號 edge 會比 V_1 的這堆 edges 要來得重

Problem5

三點在同一路徑且 v_{11} 在中間
2 號 edge 的 cost 大於 $v_{11} - v_{12}$ 路徑上所有的 edges
3 號 edge 的 cost 大於 $v_{11} - v_{13}$ 的所有 edges
現在在新樹上加上 3 號 edge，它與 2 號 edge 形成的 cycle 當中，V_1 部分的 edges 是 $v_{12} - v_{11} - v_{13}$ 路徑上的 edges
已知 $v_{12} - v_{11}$ 這堆的 cost 會小於 2 號 edge，而 2 號 edge 的 cost 小於 3 號 edge 的 cost，$v_{11} - v_{13}$ 這堆的 cost 也會小於 3 號 edge
所以 3 號 edge 會比 V_1 的這堆 edges 要來得重

Problem5

三點在同一路徑且 v_{12} 在中間
2 號 edge 的 cost 大於 $v_{11} - v_{12}$ 路徑上所有的 edges
3 號 edge 的 cost 大於 $v_{11} - v_{12} - v_{13}$ 的所有 edges
現在在新樹上加上 3 號 edge，它與 2 號 edge 形成的 cycle 當中，
V_1 部分的 edges 是 $v_{12} - v_{13}$ 路徑上的 edges
$v_{12} - v_{13}$ 這堆的 cost 會小於 3 號 edge
所以 3 號 edge 會比 V_1 的這堆 edges 要來得重

Problem5

三點在同一路徑且 v_{13} 在中間
2 號 edge 的 cost 大於 $v_{11} - v_{13} - v_{12}$ 路徑上所有的 edges
3 號 edge 的 cost 大於 $v_{11} - v_{13}$ 的所有 edges
現在在新樹上加上 3 號 edge，它與 2 號 edge 形成的 cycle 當中，V_1 部分的 edges 是 $v_{12} - v_{13}$ 路徑上的 edges
已知 $v_{12} - v_{13}$ 這堆的 cost 會小於 2 號 edge，而 2 號 edge 的 cost 小於 3 號 edge 的 cost，所以 3 號 edge 會比 V_1 的這堆 edges 要來得重

Problem5

結論是隨意挑選加入的 3 號 edge 不會比 V_1 當中的 edges 來得輕，對 V_2 也同理，而 3 號 edge 又比 2 號重，所以 3 號 edge 會被去掉

所以現在的這個樹確實已經是最好的了

Problem5

若 tree 外的 edge cost 降低
假設我們把這個 edge 加到 T，就會產生 cycle
找出 cycle 當中 cost 最高的去掉

Problem5

Is this a minimum-cost spanning tree?
將被去掉的 edge 設為 1 號，cost 降低而被加入的是 2 號
選定一個不在新樹中的 edge 3 號
我們可以用之前的方法推導出 3 號 edge 總會被移除，證明我們得到的新樹確實是 MCST

Problem5

```
function FindNewMinCostSpanningTree(G, T, e = (v, w))
    if e.cost increase and e in T then
        remove e from T
        do DFS on T from v and mark 1
        do DFS on T from w and mark 2
        for all (x, y) in E and x.mark ≠ y.mark do
            if (x, y) is smaller then
                newEdge := (x, y);
        add newEdge into T
    else if e.cost decrease and e not in T then
        add e into T
        find cycle and store the cycle in C
        for all (x, y) in C do
            if (x, y) is larger then
                removeEdge := (x, y);
        remove removeEdge from T
```

Problem5

令 T = (V', E')
其中 |V| = |V'|, |E'| = |V| - 1
複雜度分析：
兩個子樹的 DFS 總共花費 O(|V'|+|E'|) = O(|V|)
遍歷 G 的所有 edge 花費 O(|E|)

找尋 T 上的 cycle 花費 O(|V'|+|E'|) = O(|V|)
遍歷 cycle 上的 edge 則花費 O(|E'|) = O(|V|) (不是 O(|E|))

Increase: O(|V|+|E|), Decrease: O(|V|)
總和來看是 O(|V|+|E|)