Data Structures and Algorithms (II)	Spring 2007

Voluntary Exercise 2

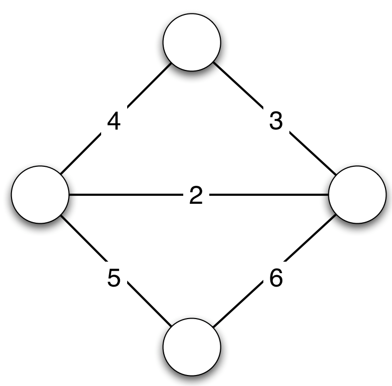
Tue, Mar 27 **_due Mon, Apr 09_**

Problem 1. (*See CLRS Problem 23-1*) Let $G = (V, E)$ be an undirected, connected graph with weight function $w : E \to R$, and suppose that $|E| \geq |V|$ and all edge weights are distinct.
A ***second-best minimum spanning tree*** is any minimum-weight spanning tree among all spanning trees except the one(s) of minimum weight. ♣

Solution.

a. We'll show that the minimum spanning tree (MST) be unique. First we assume by contradiction that T and T' are two different minimum spanning trees. And there is at least one edge $(u, v) \in T'$ and $(u, v) \notin T$. Then, $T \cup (u, v)$ must form a cycle including (u, v). Let (x, y) be the heaviest edge in the cycle. According to the cycle property, (x, y) can't be contained in any MST. If $(x, y) \in T$(in T' resp.), then $T(T'$ resp.) is not a MST, which contradicts to our assumption. Therefore the uniqueness of MST is proved.

   The second-best MST is not unique as shown by the following example:



   In the above graph, the best MST consists of edges of weights 2, 3, 5. There are two 2nd-best MSTs, one having edges of weights 2, 4, 5 and the other one having edges of weights 2, 3, 6.

b. Let T_2 be a 2-nd best MST. Let $(u, v) \in T - T_2$. Then, $T_2 \cup (u, v)$ contains a cycle where one of the edges in the cycle is not in T (cycle property). Let this edge by (x, y). Then, we must have $w(x, y) > w(u, v)$, for otherwise, we could replace (x, y) in T_2 by (u, v) to get a MST better than T_2 . Now, we note that $S = T_2 - (x, y) \cup (u, v)$ is also a spanning tree since (u, v) and (x, y) are in the same cycle.

In addition, $w(S) < w(T_2)$. So, S is a best MST. By the uniqueness of MST, we see that $S = T$, therefore T and T_2 differ with only one edge.

c. The main observation is that the path between two vertices u and v in the MST is unique. The algorithm is as follows:

Algorithm 1 *For each $u \in V$, perform a BFS or DFS to find the maximum weight between u and every other $v \in V$. Since T is a spanning tree, the BFS Tree or DFS Tree has only tree edges. When we visit edge (x, y)(from x to y), the max-weight edge in the path from root u to y is found as following:*

$$max[x, y] = \begin{cases} max[u, x] & \text{if } w(max[u, x]) > w[x, y]; \quad (1) \\ (x, y) & \text{otherwise.} \quad (2) \end{cases}$$

For each u, the time for BFS or DFS is $O(|V|)$ because a tree has only $|V| - 1$ edges. So, the total time is $O(|V|^2)$.

d. From part b, we know that we can replace one edge (u, v) in the MST by another edge (x, y) to get a 2nd-best MST. How do we determine these two edges?

Note that if we replace (u, v) by (x, y), then

$$w(T_2) = w(T) - w(u, v) + w(x, y) = w(T) + [w(x, y) - w(u, v)] \tag{3}$$

If we know which (x, y) to add to T_2, then (u, v) must be the max-weight edge in the path from x to y (which can minimize the second part of the equation above), which can be found by part c. So, we get the following algorithm:

Algorithm 2 *(1) Find MST by using Prim's Algorithm.*

(2) Find max-weight edges as in part c. Let $max[x, y]$ denote the max-weight edge in the path from x to y in tree T.

(3) Find an edge $(x, y) \in E - T$ that minimizes $w(x, y) - w(max[x, y])$.

(4) Output $T_2 = (T - max[x, y]) \cup (x, y)$.

The step (1) can be done in time $O(|V|^2 \lg |V|)$ by using min-heap, and step (2) can be done in time $O(|V|^2)$. There are $E = O(|V|^2)$ edges to be considered in step (3), and so step (3) takes time $O(|V|^2)$. The total runtime is $O(|V|^2 \lg |V|)$.

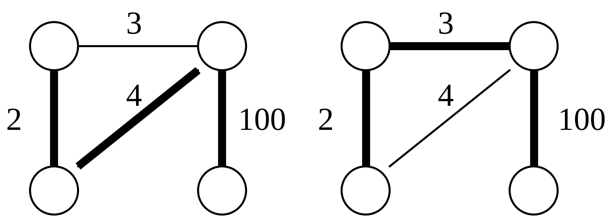
If we use Fibonacci heap to realize the priority queue in the Prim's Algorithm, the running-time will be guaranteed in $O(|V|^2)$.

■

Problem 2

(From 守壹, 愷陽)

a. 題目是：Let T be a spanning tree of G whose largest edge weight is minimum over all spanning trees of G ⇔ T is a minimum spanning tree.



=> 這個方向是錯的。如左圖，雖然兩個 tree 的 largest edge weight 都是 100，但是左邊那顆不是 minimum spanning tree

<= 這個方向是對的。假設有一個 MST T，他的 largest edge weight $w(e)$不是 minimum over all spanning trees。 表示說有另一個 spanning tree T^*, 他的 largest edge weight $y < w(e)$。所以 $w(e)$比在 T^*中的每一個邊的 weight 都還要重。而當我們把 e 放到 T^*，會產生一個 cycle，而 e 在此 cycle 中是 weight 最重的 edge，但是這個和 cycle property 一個 cycle 最重的 edge 不可能在 MST 中矛盾，所以若 T 是 MST，則 T is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G.

b. We claim that 如果在原本的 graph 上的 MST 是 T，加了新的點以後，在考慮新的 graph 的 MST T'時，那些原本在舊的 graph 上的 non tree edge 不可能是 T'的一個邊。因爲那些 non tree edge，在舊的 graph 上是某一個 cycle 的最重邊，但是因爲新的 graph 我們只有加入新的點和邊，所以那些 cycle 仍然存在在新的 graph 中，所以那些 non tree edge，由 cycle property，不可能在 T' 上。所以我們現在需要考慮新的 graph 中的 edge 就變少了。$m = (n - 1) + n = O(n)$。其中 $n - 1$ 是在 T 上的那些邊，而 n 是指加入的新的點，他最多會新加 n 個 edge 到新的 graph 裡面。而 Kruskal algorithm 是 $O(m\log n)$，所以要 update 到新的 MST 需要花 $O(n\log n)$。

Problem 3. Give an efficient algorithm to find a minimum weight cycle in a weighted, undirected, connected graph $G = (V, E)$ in which all edge weights are non-negative. Analyze its running time.

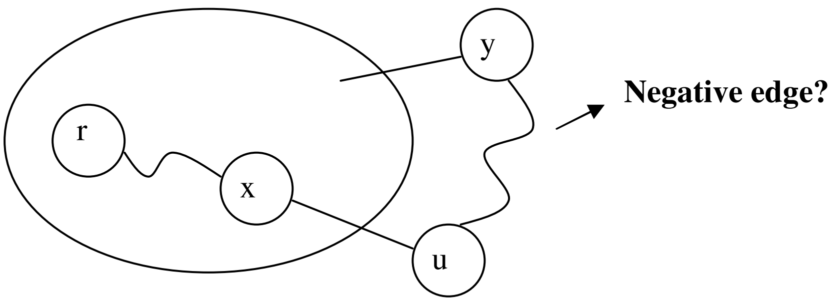
Solution.

For each edge (u,v) in G, remove edge (u,v) from G but keep the end vertices. Run Dijkstra's algorithm to find shortest path from u to v. If the shortest path from u to v exists, add (u,v) to this path will form a cycle. Then we can find the cycle with minimum weight among them. This algorithm runs in $O(E(E + V\lg V))$.

Another approach is running Dijkstra's algorithm for each vertex v, and for each edge (x,y) not in the shortest path tree, add (x,y) to the tree will form a cycle. Since we have the distance from v to all vertices, we can find the cycle with minimum weight by adding non-tree edges in $O(V^2 + E)$ time. So the total running time of this algorithm is $\mathrm{O}(V(E + V\lg V + V^2))$.

Problem 4.

Solution:



Assume we choose u to be the next vertex, then $d[u]$ must be equal to $d(u)$, the shortest distance from r to u. By contradiction, there might be some negative edges on the path from another vertex y to u, then we will evaluate new distance of u, $d'[u]$. The negative edges may cause $d'[u]$ smaller than $d[u]$, so $d[u]$ is not the shortest distance from r to u.

But if the negative edges all incident to r, it's impossible that any negative edge on the path from y to u. We always evaluate $d[u]$ correctly.