Homework 8

Yu Hsiao Yu-Hsuan Wu

Question 1

1. Consider Dijkstra's algorithm for single-source shortest paths. The values of SP for all vertices may be stored in either an ordinary array or a min heap. How do these two implementations compare in terms of time complexity? Please explain.

Question 1

```
function DIJKSTRA(G, v)
    for all vertices w do
        w.mark := false;
        w.SP := ∞;
    end for
    v.SP := 0;
    while there exists an unmarked vertex do
        let w be an unmarked vertex such that w.SP is minimal;
        w.mark := true
        for all edges (w, z) such that z is unmarked do
            if w.SP + length(w, z) < z.SP then
                z.SP := w.SP + length(w, z);
            end if
        end for
    end while
end function
```

Question 1

The 2 key operations:

- select vertex v with minimum SP (and remove it from data structure)
- for each vertex v, update its neighbor's SP

Question 1

Using min heap:

- Remove minimum SP from heap: $O(\log |V|)$
- Insert the updated SP into heap: $O(\log |V|)$

The while loop repeat $|V|$ times and the update takes at most $|E|$ times.

Time Complexity: $O(|V| \times \log |V|) + O(|E| \times \log |V|)$

$$= O((|V| + |E|) \times \log |V|)$$

Question 1

Using array:

- Remove minimum SP from array: $O(|V|)$
- Insert the updated SP into array: $O(1)$

The while loop repeat $|V|$ times and the update takes at most $|E|$ times.

Time Complexity: $O(|V| \times |V|) + O(|E|)$

$$= O(|V|^2)$$

Question 2

2. (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.

Question 2

We will proof by **contradiction:**

- Suppose there exist two distinct minimum-cost spanning trees MST_1 and MST_2 for a given connect graph G.
- First, we sort the edges of MST_1 and MST_2 in ascending order of cost:
  - MST_1: $w(e_1) < w(e_2) < \cdots < w(e_{i-1}) < w(e_i) < \cdots$
  - MST_2: $w(e_1) < w(e_2) < \cdots < w(e_{i-1}) < w(e_j) < \cdots$
- Without loss of generality, assume that $w(e_i) < w(e_j)$; that is, e_i is the minimum cost edge that is in MST_1 but not in MST_2.
- Since MST_2 is an MST, $MST_2 \cup \{e_i\}$ will create a cycle c.
- Let's now take a look at c. Since $e_i \in MST_1$ and MST_1 is an MST, there must exist an edge $e_k \in c$ with a higher cost than e_i (and, of course, no smaller than e_j) and not in MST_1.

Question 2

- Note that $c \subset MST_2 \cup \{e_i\}$ and $e_k \in c$, so it is obvious that $e_k \in MST_2$ and $e_k \notin MST_1$.
- Now, consider $MST_2 \cup \{e_i\} \backslash \{e_k\}$. This is an MST that costs less than MST_2. Thus, we know MST_2 is not a minimum-cost spanning tree.
- By contradiction, we know that G has exactly one unique minimum-cost spanning tree. ■

Question 3

3. The well-known Kruskal's algorithm computes the minimum-cost spanning tree of a given connected weighted undirected graph with n vertices as follows:

   Initially, it treats the n vertices as a forest of n trees, each of a single node. It then examines the edges one by one in *increasing order* of their weights. If the edge under examination connects two different trees (i.e., the edge does not complete a cycle), it is included in the forest (causing the forest to evolve, eventually becoming a single tree).

   Please present the algorithm in adequate pseudocode utilizing the two operations of the Union-Find data structure. What is the time complexity of the algorithm? Please explain.

Question 3

The steps of **Kruskal's algorithm** are as follows:

1. Remove all self-loops.
2. Sort all edges in increasing order by their weights.
3. Pick the unchosen edge with the minimal weight if its appearance doesn't form any cycle; otherwise, move on to the next edge in the order.

Ultimately, all trees (the vertices) in the forest are connected, resulting in the desired MST.

Question 3

Algorithm KRUSKAL($G(V, E)$)
  begin
    $T := \emptyset$;
    $L_E :=$ sort E in ascending order;
    for v in V **do**
      MakeSet(v);
    end for
    for $i := 1$ to $|E|$ **do**
      $L_E[i] = \{u, v\}$;
      if Find(u) $\neq$ **Find**(v) **then**
        $T.add(L_E[i])$;
        Union(u, v);
      end if
      if $|T| = |V| - 1$ **then**
        return T;
      end if
    end for
  end
end Algorithm

Question 3

Theorem

*If both balancing and path compression are used, any sequence of m **Find** or **Union** operations (where $m \geq n$) takes $O(m \log^* n)$ steps.*

Time complexity:

- Sort edges: $O(|E| \log |E|)$.
- Union-Find: $O((2|E| + |V| - 1) \cdot \log^* |V|)$
  - Find: $2|E|$ operations
  - Union: $|V| - 1$ operations
  - Total: $2|E| + |V| - 1$ operations

Thus, we have $O(|E| \log |E|) + O((2|E| + |V| - 1) \cdot \log^* |V|)$. And since $\log^* |V|$ grows extremely slowly, the time complexity can also be written as $O(|E| \log |E|)$.

Question 4

4. 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 disjoint subgraphs arbitrarily (by removing the edges between the two subgraphs), recursively compute their minimum-cost spanning trees, and then connect the two spanning trees with an (earlier removed) edge between the two subgraphs that has the minimum weight.

Question 4

Some details are missing in the algorithm.

1. How to divide the graph?
2. How to break a tie when the edges have the same weight?

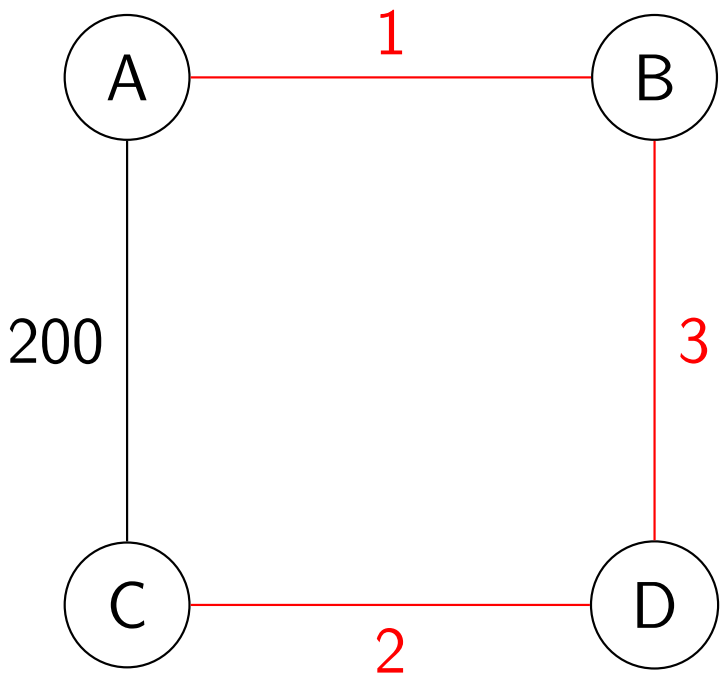
Question 4

Most importantly, the algorithm does not ensure preserving all the important edges of MCST.

It derives wrong result when at least two important edges are cut at the same time.

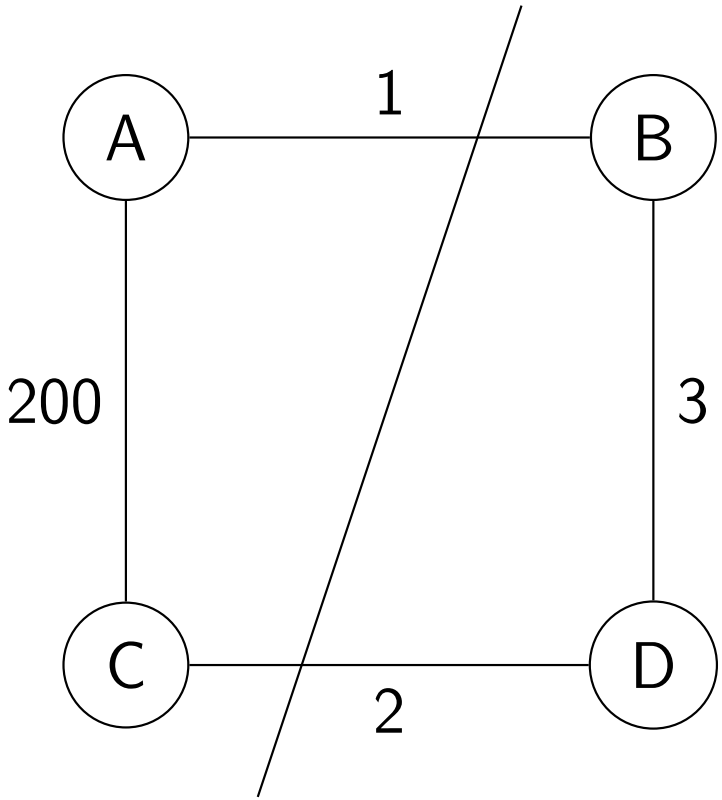
Question 4

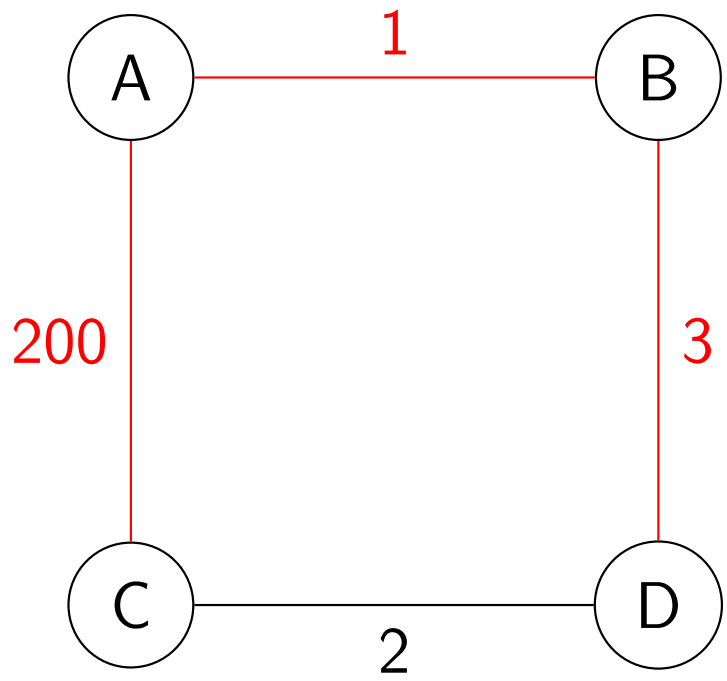
For example, edge $\overline{AB}$, $\overline{BD}$, and $\overline{CD}$ constitutes the MCST:



Question 4

But if we divide the graph by cutting through $\overline{AB}$ and $\overline{CD}$, we will obtain the tree shown in the bottom right figure. This is because we are forced to pick the smallest weight edge among the cuts, but actually both of them are crucial to the MCST.





Question 5

5. (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 *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.

Question 5

Given a spanning tree T of a graph. If we arbitrarily add an edge to T (creating a cycle) and the edge always has the largest cost of the cycle, T is an MCST.

Question 5

For the decreased edge $\{u, v\}$, there are two cases: $\{u, v\}$ is in T or not.

Obviously, if $\{u, v\}$ is in T, T must still be an MCST of G, so we only need to consider the situation that $\{u, v\}$ is not in T.

The idea is to add $\{u, v\}$ to T, which creates a cycle in the tree. After locating the cycle, we remove the edge with the largest cost in the cycle and obtain a new MCST.

Question 5

Algorithm FindNewMCST$(G(V, E), T, e\{u, v\})$

```
begin
    if e ∉ T then
        newEdge := e;
        add e to T;
        run DFS on T to find the cycle C
        for all edges {x, y} ∈ C do
            if cost({x, y}) > cost(newEdge) then
                newEdge := {x, y};
            end if
        end for
        remove newEdge from T;
    end if
end
end Algorithm
```

Question 5

Let $T = (V, E')$, where $E' \subseteq E$ and $|E'| = |V| - 1$.

Time complexity:

- finding cycle: $O(|V| + |E'|) = O(|V|)$.
- finding maximum cost edge in a cycle: $O(|E'|) = O(|V|)$.

Total time complexity: $O(|V|)$.