Homework 8

Yu-Ju Teng Ling-Hsuan Chen

Problem 1

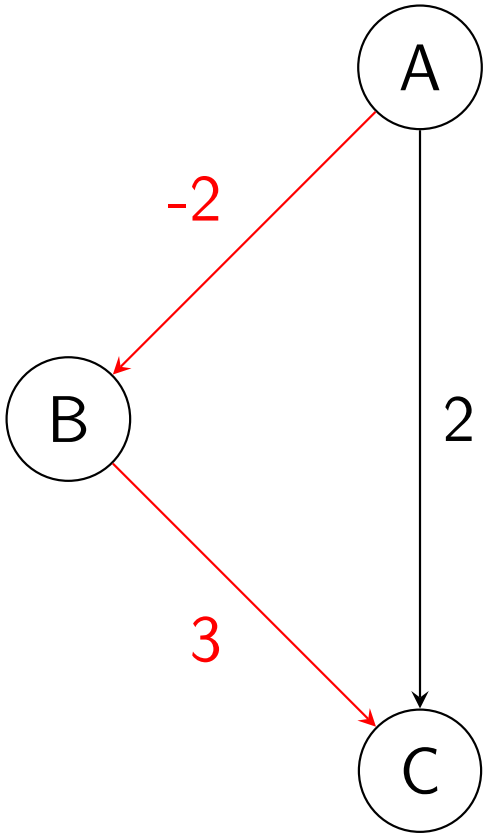
1. Dijkstra's algorithm for single-source shortest paths assumes that every edge of the input graph has a nonnegative weight. Suppose we are given a graph with negative weights on some edges, where the minimum weight of the edges is $-c$ for some $c > 0$. If we add c to the weight of every edge, then 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.

Problem1

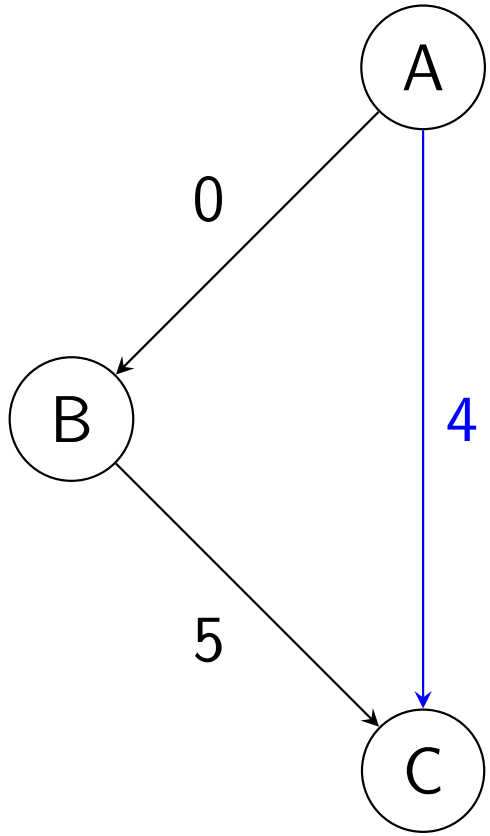
No, this method cannot give us the shortest paths for the original graph.

1. Assume each path has a different number of edges.
2. If there are two paths P_1 and P_2 with the number of edges N_1 and N_2 respectively, and assuming that the weights of P_1 and P_2 are W_1 and W_2 respectively (with P_1 being the shortest path in the original graph, i.e., $W_1 < W_2$), then in the new graph with nonnegative edge weights obtained after adding c to the weight of every edge:
   - The weight of P_1 in the new graph will be $W_1 + cN_1$.
   - The weight of P_2 in the new graph will be $W_2 + cN_2$.
3. It cannot be guaranteed that $W_1 + cN_1 < W_2 + cN_2$ still holds. This means that if N_1 is large enough compared to N_2, P_2 may become the new shortest path in the new graph.

Problem 1



Old graph



New graph (add 2)

The shortest path will change from the red path to the blue path.

Problem 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.

Problem 2

Proof by contradiction:

- Suppose there exist two distinct minimum-cost spanning trees $MST_1(G)$ and $MST_2(G)$.
- Sort the edges of $MST_1(G)$ and $MST_2(G)$ in ascending order of cost.
  - $MST_1(G)$: $\{e_1 < e_2, \cdots\}$, $MST_2(G)$: $\{e'_1 < e'_2, \cdots\}$
- Let e_i be the minimum cost edge in MST_1 but not in MST_2.
- Let e'_j be the minimum cost edge in MST_2 but not in MST_1.
- Assume $e_i < e'_j$, then $MST_2(G) \cup \{e_i\}$ will create a cycle.
- Let e'_k be the maximum cost edge of the cycle.
- Since all edge costs are distinct and e'_k is the maximum cost edge in a cycle, e'_k does not belong to any minimum spanning tree. But e'_k is in MST_2

$\rightarrow$ MST_2 is not minimum-cost spanning tree (Contradiction).

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

Problem 3

Kruskal's algorithm:

1. Step 1 : Remove all self-loops.
2. Step 2 : Sort all the edges in increasing order of their weights.
3. Step 3 : Choose the edge with minimal weight if its appearance won't form any cycle.

In the end, each tree in the forest is connected.
There's only one tree left, which is the MCST.

Problem 3

Algorithm Kruskal's algorithm$(G(V, E))$

begin

$T := \emptyset$

Sort all edges E by increasing order;

for v in V **do**;

MakeSet(v);

for each edge {u, v} in sorted E **do**

if Find$(u) \neq$ **Find**(v) **then**

T.add({u, v});

Union(u, v);

return T

end

Problem 3

Time complexity :

- Sort all edges : $|E| \log |E|$.
- Union-Find :
  - We do m **find** operations, and the total time complexity of this part is $m \log^* |V| = (2|E| + |V| - 1) \log^* |V|$.

The overall time complexity is

$$\begin{aligned} & O(|E| \log |E| + (2|E| + |V|) \log^* |V|) \\ = \quad & O(|E| \log |E|). \end{aligned}$$

Problem 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.

Problem 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?
3. How to ensure that the cut dividing the edges does not include important edges?

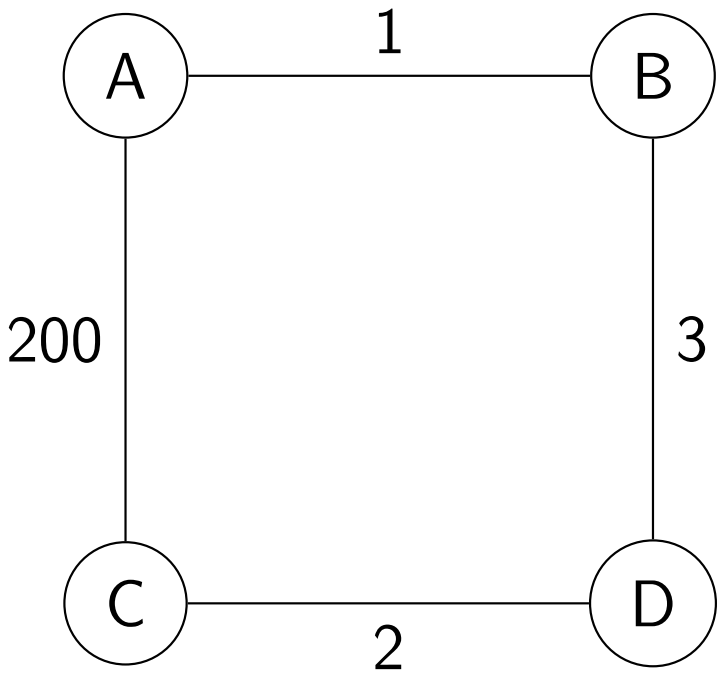
You need to explain the reasons above with details.

Problem 4

For example:

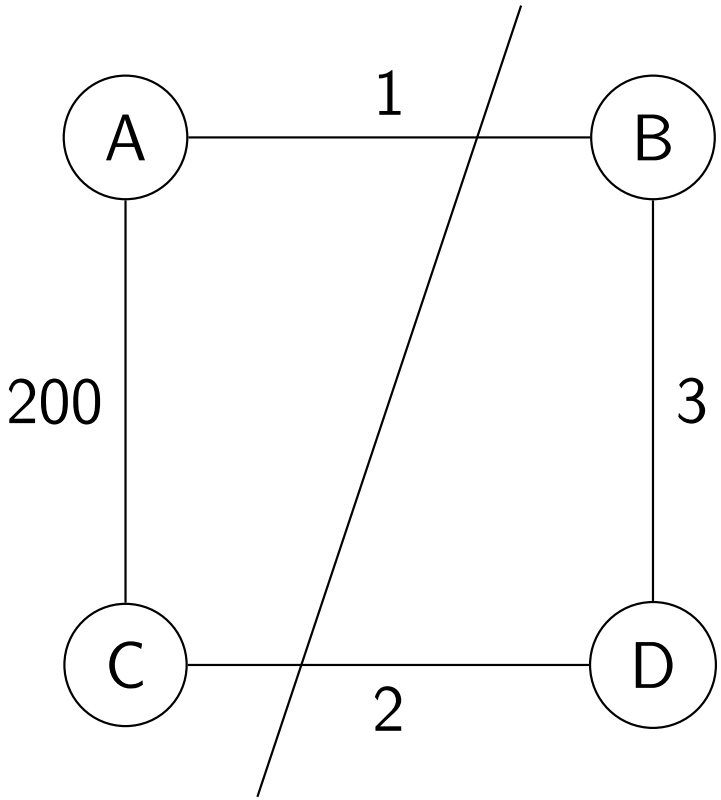
The algorithm does not explain how to divide the graph.

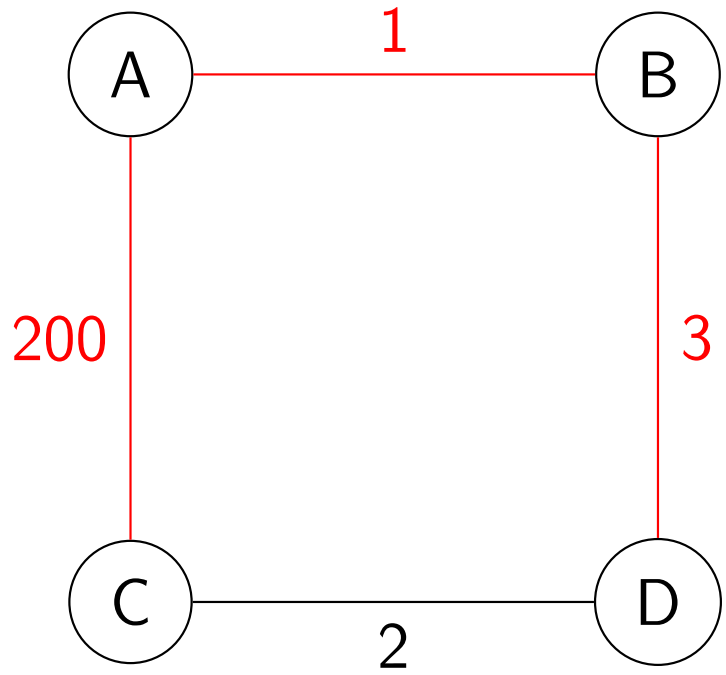
e.q.



Problem 4

If we divide the graph like the left figure shows, we will obtain the MCST likes the right figure, which is obviously not the MCST we expected:





Problem 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 *increased*; $\{u, v\}$ 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.

Problem 5

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

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

The idea is to remove $\{u, v\}$ from T and divide T into two subtrees T_1 and T_2. For all edges connecting T_1 and T_2, find the edge with the minimal cost, and use this edge to connect T_1 and T_2 to form a new MCST.

Problem 5

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

begin

if $e \in T$ **then**

newEdge := e ;

remove e from T ;

run DFS on T from u and assign $x.mark := 1$
for all searched vertices x ;

run DFS on T from v and assign $x.mark := 2$
for all searched vertices x ;

for all edges $\{x,y\} \in E$ and $x.mark \neq y.mark$ **do**

if cost$(\{x,y\})$ < cost(newEdge) **then**

newEdge := $\{x,y\}$;

add newEdge into T;

end

Problem 5

Time complexity:

Let $T = (V, E')$, where $E' \subseteq E$ and $|E'| = |V| - 1$.
Time complexity of two DFSs: $O(|V| + |E'|) = O(|V|)$.
Time complexity of finding all connecting edges $\{x, y\}$: $O(|E|)$.

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

Problem 5

Why the algorithm works?

Use the theorem in slides [Basic Graph Algorithms, p.31]:

Theorem

Let V_1 and V_2 be a partition of V and $E(V_1, V_2)$ be the set of edges connecting nodes in V_1 to nodes in V_2. The edge with the minimum weight in $E(V_1, V_2)$ must be in the minimum-cost spanning tree of G.

In our algorithm, when the edge $\{u, v\}$ is increased, all the edges in G but not in T that connecting the same two subtrees T_1 and T_2 should be rechecked, because they have larger costs than $\{u, v\}$ before, but not sure now.