HW3

<u>Problem 1</u>

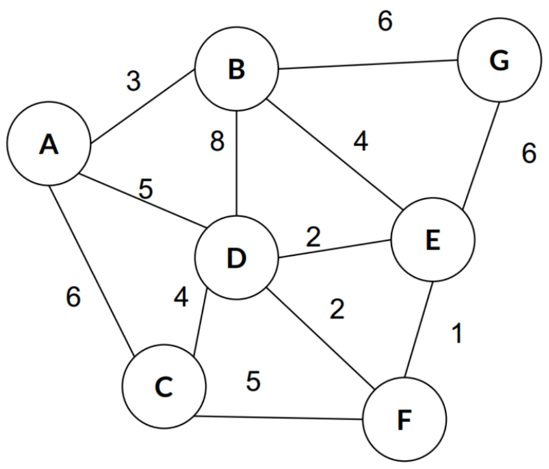
[5 points] You are given a minimum spanning tree T in a graph G = (V,E). Suppose we remove an edge from G creating a new graph G1. Assuming that G1 is still connected, devise a linear time algorithm to find a MST in G1.

Solution: Given a graph G and its minimum spanning tree T, if an edge e is removed to form a new graph G1, the algorithm verifies if e is in T. If it isn't, T remains the MST for G1; if it is, it finds the smallest edge in G1 connecting the two components resulting from removing e from T, and adds this edge to T, creating the MST for G1.

We would take O(E) time to verify if removed edge is in the MST or not. Similarly, we would take O(V) time to traverse the vertices to check for minimum cost edge that belongs to G1 and connects the two components. Therefore, overall the algorithm would take O(V+E) time or linear time.

<u>Problem 2</u>

[15 points] Considering the following graph G:



1. In graph G, if we use Kruskal's Algorithm to find the MST, what is the third edge added to the solution? Select all correct answers.

Solution: c. A-B

2. In graph G, if we use Prim's Algorithm to find MST starting at A, what is the second edge added to the solution?

Solution: b. B-E

3. What is the cost of the MST in the Graph?

Solution: d. 21

Problem 3

[20 points] A new startup FastRoute wants to route information along a path in a communication network, represented as a graph. Each vertex represents a router and each edge a wire between routers. The wires are weighted by the maximum bandwidth they can support. How can we find a path with maximum bandwidth between s and t? As you would expect, the bandwidth of a path is the minimum of the bandwidths of the edges on that path; the minimum edge is the bottleneck. Show the steps you take to solve this problem.

Solution: We can solve this problem by using a modified version of Prim's algorithm. Basically the only changes that we would make to the algorithm is that we would use a max heap instead of a min heap and we would store the root node value as ∞ and all other nodes as 0.

Implementation:

```
Initially S = {s} and d(s) = ∞
    for all other nodes d(u) = 0
While S != V
    v = Extract-Max(Q)
    S = S ∪ {v}
    for each vertex u ∈ Adj(v)
        if d(u) < e(u, v)
            Increase-Key (Q, u, e(u, v))
    end for
end while
```

The overall time complexity for this approach would be similar to Prim's which would be O(m log n), where m is the total number of wires and n is the total number of routers.

Problem 4

[20 points] A network of n servers under your supervision is modeled as an undirected graph G = (V,E) where a vertex in the graph corresponds to a server in the network and an edge models a link between the two servers corresponding to its incident vertices. Assume G is connected. Each edge is labeled with a positive integer that represents the cost of maintaining the link it models. Further, there is one server (call its corresponding vertex as S) that is not reliable and likely to fail. Due to a budget cut, you decide to remove a subset of the links while still ensuring connectivity. That is, you decide to remove a subset of E so that the remaining graph is a spanning tree. Further, to ensure that the failure of S does not affect the rest of the network, you also require that S is connected to exactly one other vertex in the remaining graph. Design an algorithm that given G and the edge costs efficiently decides if it is possible to remove a subset of E, such that the remaining graph is a spanning tree where S is connected to

exactly one other vertex and (if possible) finds a solution that minimizes the sum of maintenance costs of the remaining edges.

Solution: To solve this problem, we will apply the Kruskal's algorithm with slight modifications. Kruskal's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph. It adds edges in increasing order of cost if they connect disjoint components, avoiding cycles.

However, in this problem, we don't want server S to be connected to more than one vertex, so we will handle this server differently.

Here is the step-by-step solution:
- Remove all edges connected to server S from the graph.
- Apply Kruskal's algorithm to the remaining graph to find a minimum spanning tree. If the graph becomes disconnected after removing S, it is not possible to solve the problem, so stop here.
- Among the edges connected to S that were removed in step 1, add back the edge with the smallest cost to the minimum spanning tree obtained in step 2.

This algorithm works because it always selects the smallest cost edge, ensuring that the sum of the maintenance costs of the remaining edges is minimized.

The time complexity of this algorithm is the same as Kruskal's algorithm, which is O(E log E) if we use a fast union-find data structure. The reason for this is because the most time-consuming step is sorting the edges, which takes O(E log E) time, while all other steps take linear time in the number of edges or vertices.

Problem 5
[20 points] Prove or disprove the following:

- **T is a spanning tree on an undirected graph G = (V, E). Edge costs in G are NOT guaranteed to be unique. If every edge in T belongs to SOME minimum cost spanning trees in G, then T is itself a minimum cost spanning tree.**

- **Consider two positively weighted graphs G = (V, E, w) and G′ = (V, E, w′) with the same vertices V and edges E such that, for any edge e in E, we have w′(e) = w(e)2 For any two vertices u, v in V , any shortest path between u and v in G′ is also a shortest path in G.**

Solution:
(1) True. If every edge in a spanning tree T of a graph G belongs to some minimum cost spanning tree (MST), then the cost of T is minimized, making T an MST. If T wasn't an MST, there would exist a lower-cost tree T' where at least one edge of T wouldn't be

included, contradicting our initial assumption. Therefore, T must be an MST if all its edges belong to some MST of G, as any contradiction leads to T having a non-MST edge.

(2) False. Suppose there are two nodes a and b between u and v such that for graph G
e(s, a) = 1
e(a, b) = 1
e(b, t) = 1
e(s, t) = 2
Here the shorts path is the edge e(s, t)
However, for G' the edges would be
e(s, a) = 1
e(a, b) = 1
e(b, t) = 1
e(s, t) = 4
Making the shortest path s→a→b→t for the graph G' and thereby contradicting the statement.

<u>**Problem 6**</u>
[20 points] You have been tasked with organizing a conference that will bring together participants from different countries, each with their own native language. In order to facilitate communication and ensure that all participants can understand each other, you need to hire a set of interpreters. However, due to budget constraints, you can only hire a limited number of interpreters. There are n different languages spoken by the participants and m available interpreters. Each interpreter is fluent in exactly two languages and can provide simultaneous interpretation between them. Each interpreter has a specific hiring cost associated with them. Your goal is to determine the minimum cost of hiring a subset of interpreters such that every participant can understand each other, either directly or through a chain of interpreters. Design an efficient algorithm to solve this problem and find the minimum cost of hiring interpreters for the conference.

G: Graph representing the interpreters and their language pairs
V : Set of vertices representing the interpreters
E: Set of edges representing the language pairs between interpreters

Solution:
This problem can be thought of as a Minimum Spanning Tree (MST) problem, where vertices represent languages, edges represent interpreters fluent in the two languages that the edges connect, and edge weights represent hiring costs. We need to find an MST that connects all vertices (languages) with minimum total edge weight (cost).

Here's an outline of the algorithm, using Kruskal's algorithm, which is an efficient algorithm for finding MSTs:

1. Construct a graph G where each vertex represents a language, each edge represents an interpreter between two languages, and edge weight represents the interpreter's cost.

2. Sort all edges in the graph based on their weights in ascending order.

3. Add the least weight edge to the Minimum Spanning Tree (MST), checking for cycles using a disjoint-set data structure and discarding if it forms a cycle.

4. Repeat step 3 until the MST has (n-1) edges or all edges are examined, indicating all languages can't be interpreted if fewer than (n-1) edges.

5. The total weight of the edges in the MST represents the minimum cost to hire the interpreters.

This algorithm runs in O(ElogE) or O(ElogV) time where E is the number of edges (interpreters) and V is the number of vertices (languages). This is because we need to sort all edges first (which takes O(ElogE) time), and for each edge, we need to do a union-find operation (which takes O(logV) time).