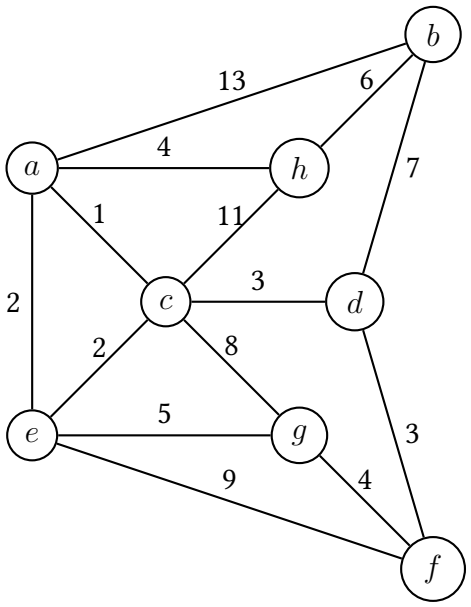
CS6033 Design and Analysis of Algorithms I
Homework Assignment 3 Part 1
10.10.2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)
vt2182 (Venu Vardhan Reddy Tekula)

---

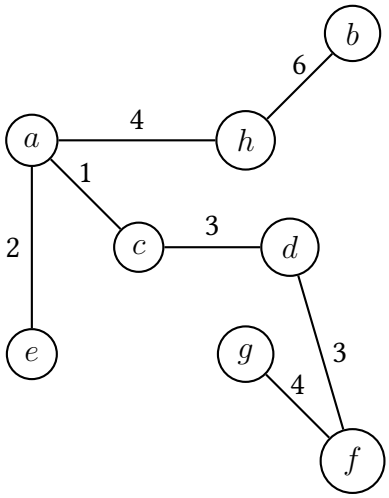
1. (10 points) Consider the given graph.



   (a) (3 points) What is the the cost of a minimum spanning tree?

   Ans:

   The cost of the minimum spanning tree for the given graph is 23.

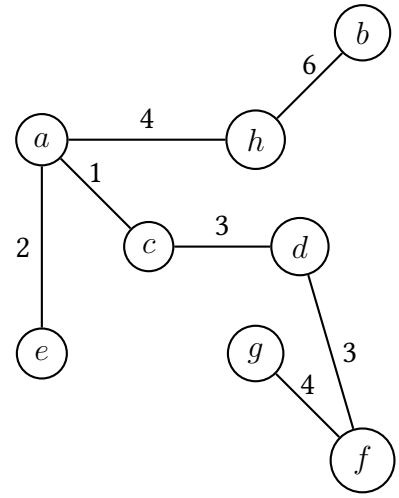


(b) (2 points) How many minimum spanning trees does it have?

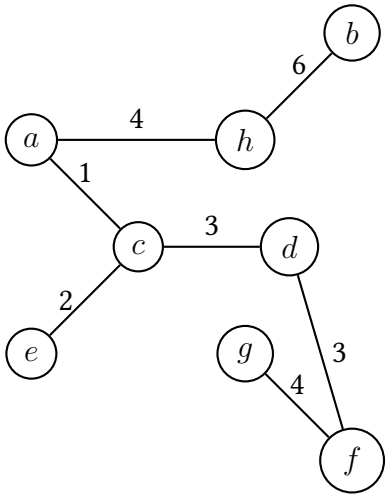
Ans:

There will be two minimum spanning trees for the given graph.

(i.)



(ii.)



(c) (5 points) Run Kruskal's algorithm on the graph above. In what order are the edges added to the MST? For the first three edges in this sequence, give a cut that justifies its addition

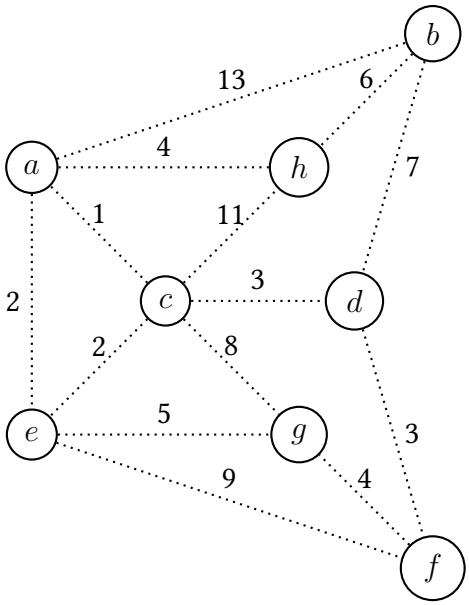
Ans:

$MST = 0$

Unvisited edges = $\{a, b, c, d, e, f, g, h\}$

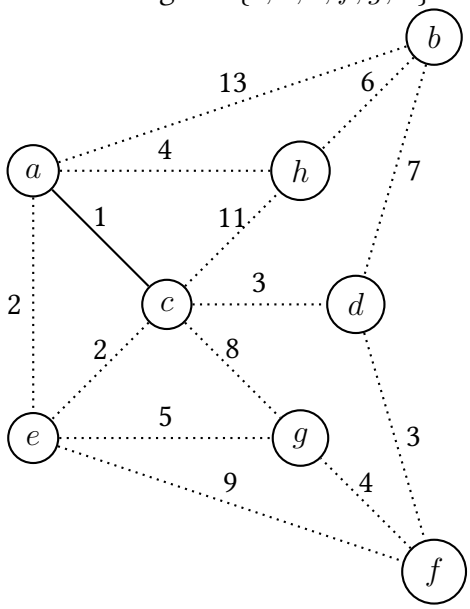
Sort the edges in ascending (increasing) order of the weights of the edges.

$ac - 1$, $ae - 2$, $ce - 2$, $cd - 3$, $df - 3$, $ah - 4$, $gf - 4$, $eg - 5$, $bh - 6$, $bf - 7$, $cg - 8$, $ef - 9$, $ch - 11$, $ab - 13$.

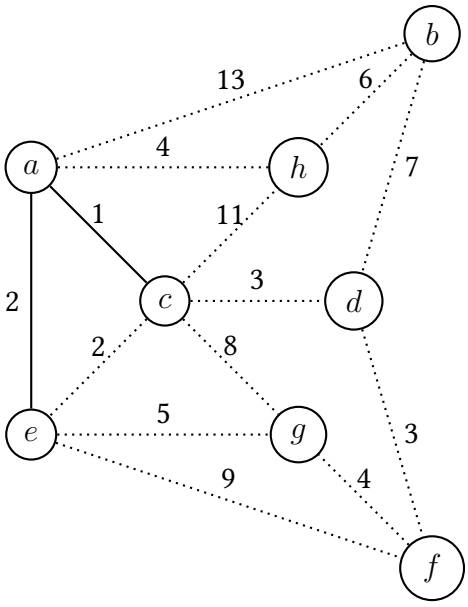


Steps:

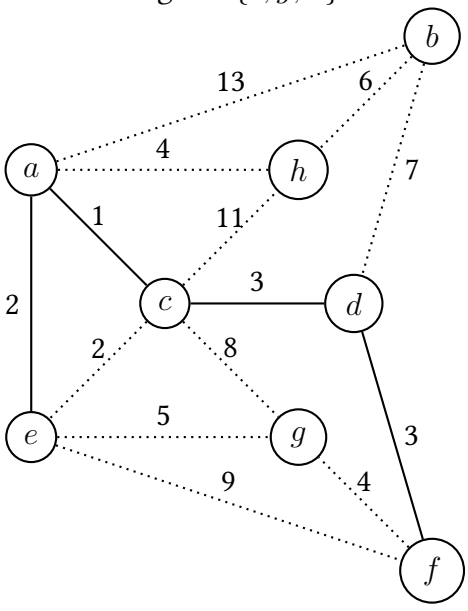
i. We will start with the lowest weighted edge first i.e., the edges with weight 1 (ac).
$MST = 0 + 1 = 1$
Unvisited edges = $\{b, d, e, f, g, h\}$



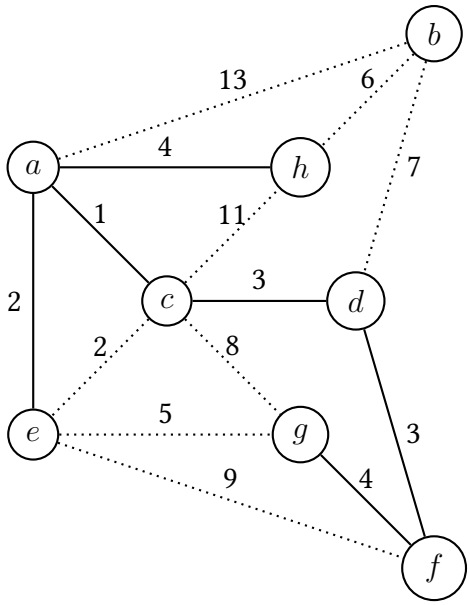
ii. After that we will select the second lowest weighted edge i.e., edges with weight 2 (ae or ce). We cannot select both the edges and they are not disjoint and forms a cycle. So, we need to select only one of the two edges. Let's go with the edge ae.
$MST = 1 + 2 = 3$
Unvisited edges = $\{b, d, f, g, h\}$



iii. Next, the least weighted edge is 3, the edges are (cd or df). Since both are disjoint edges, we can select both.
$MST = 3 + 3 + 3 = 9$
Unvisited edges = $\{b, g, h\}$

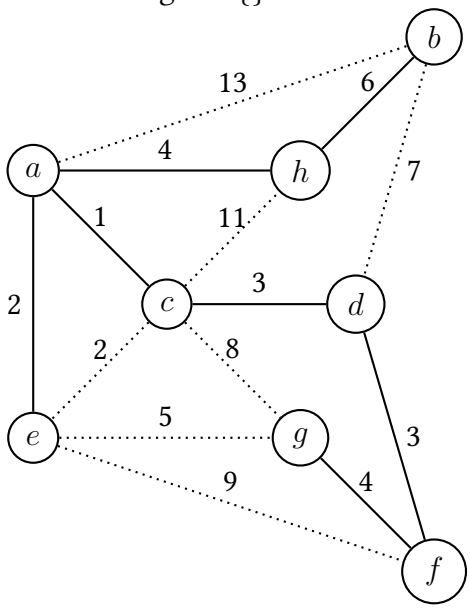


iv. Next, the least weighted edge is 4, the edges are (ah or gf). Since both are disjoint edges, we can select both.
$MST = 9 + 4 + 4 = 17$
Unvisited edges = $\{b\}$



v. Next, the least weighted edge is 5 and the edge is eg. If we consider the edge, there would be a cycle formed $acdfgea$. So, we should not consider this edge.

vi. Next, the least weighted edge is 6, the edge which has the weight is bh. No cycle would be formed, so we can consider this edge.
$MST = 17 + 6 = 23$
Unvisited edges = $\{\}$



Since all the nodes are visited, we can now stop the process.
The cost of the minimum spanning tree is $MST = 23$ and the order of the edges added to the MST are ac, ae, cd, df, ah, gf, bh.

2. (5 points) Prove or disprove: If a graph has a cycle, the heaviest edge in the cycle will not be part of any MST.

   Ans:

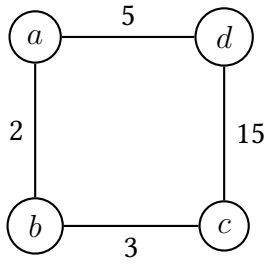
   The above statement "If a graph has a cycle, the heaviest edge in the cycle will not be part of any MST" is **True**.

   Proof (by contradiction):

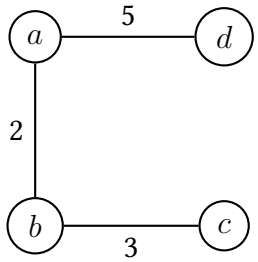
   Suppose G is the given cyclic graph. Assume T is the spanning tree with the heaviest edge p included. Let the cost of the T be x. If you remove the heaviest edge p from the spanning tree, we would have two nonempty connected components $G1$ and $G2$. Since G is a cyclic graph, there would be another edge which connects $C1$ and $C2$. Let's assume this edge be q. Let's consider the new spanning tree be T' by removing the edge p and adding the edge q. Consider the cost of the T' be y, we can infer that $y < x$ since the weight of $q <$ weight of p (heaviest edge). So, T would not be the MST.

   Example

   Given graph

   Minimum Spanning Tree

   The heaviest edge cd (weight 15) is not a part of the MST.

3. (5 points) Prove or disprove: Adding a constant to every edge doesn't change the MST.

   Ans:

   The above statement "Adding a constant to every edge doesn't change the MST" is **True**.

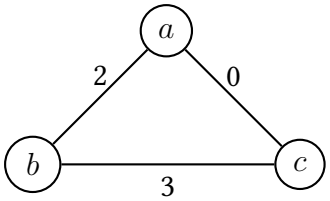
   Proof:

   Let's assume the given graph G has n nodes/vertices. Any spanning tree for G would have $(n-1)$ edges. Let the MST be T and the minimum cost be c. Let's add x to all the weights of the edges. Let's say the new graph is G' and the spanning tree be T'. This increment would increase the cost of every spanning tree by $(n-1)*x$ since it picks the same number

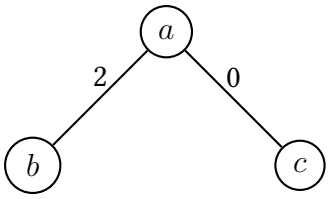
of edges of graph G to obtain minimum spanning tree T' for the graph G'. Ans so the new minimum cost be $c' = c + (n-1) * x$. So, any spanning tree with the minimum cost c for the graph G will also have the minimum cost c' in the new graph G'.

Example

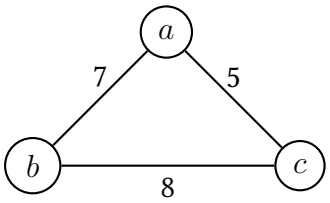
Given graph G



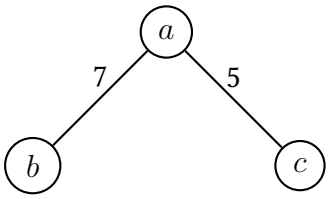
Minimum Spanning Tree for G is T



Adding 5 on all the weights of the given graph G, let's say the new graph is G'



Minimum Spanning Tree for G' is T'



The Minimum Spanning Trees T and T' are the same even after all the weights are incremented with a constant.

4. (15 points) You are the newly appointed mayor of NYC. You have been given a good budget to renew n streets. You would like to renew the streets so that the maximum number of commuters will use the new streets. You have a graph of the existing streets and information about, on average how many times per day they are used.

   You want to renew the most used streets. You are given a graph G with all the streets as edges, intersections as nodes and average usage as weights.

   (a) (12 points) Design an algorithm to determine which of the n streets need to be renewed.

Ans: Let us formulate this problem into a graphs where cities are considered as vertices and streets (path between the cities) as edges.

We are given maximum number of times a street is used by commuters, so let us consider the number as weight of the edge. Now we need to determine which edges we need to pick such that it is used by most number of commuters.

To pick the edges used by most number of commuters, we will follow modified kruskal algorithm to build up on the edges we select. The process we follow is sort all the edges in descending order. Pick the edge and see if it forms cycle with already selected edges. If it does not form cycle, add it to the selected set of edges. We will stop the process when we have n edges in our selected edges list.

```
class Graph(object):
    def __init__(self, num_vertices):
        self.V = num_vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, root, i):
        if root[i] == i:
            return i
        print(i, root[i])
        root[i] = self.find(root, root[i])
        return root[i]

    def union(self, root, rank, x, y):
        print(f"root: {root}, rank: {rank}")
        xroot = self.find(root, x)
        yroot = self.find(root, y)
        if rank[xroot] < rank[yroot]:
            root[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            root[yroot] = xroot
        else:
            root[yroot] = xroot
            rank[xroot] += 1
        print(f"root: {root}, rank: {rank}")


    def kruskals(self):
        n = input("No. of streets to be renewed: ")
        result = []
        i, e = 0, 0
```

```
# sort the graph based on edge weights
# in descending order
self.graph = sorted(
                    self.graph,
                    key = lambda item: item[2],
                    reverse=True
                )

# initialize root, which keeps track of the MST
# and the rank, which keeps track of where each
# node belongs
root = []
rank = []
for node in range(self.V):
    root.append(node)
    rank.append(0)

while e < n:
    u, v, w = self.graph[i]
    i = i + 1
    x = self.find(root, u)
    y = self.find(root, v)
    print(f"x, y: {x}, {y}")
    if x != y:
        e = e + 1
        result.append([u, v, w])
        self.union(root, rank, x, y)

for u, v, w in result:
    print(f'{u} - {v}: {w}')
```

(b) (3 points) State your running time.
Ans: $\mathcal{O}(E \log E)$

5. (15 points) After providing the solution to question 4, suddenly a new restaurant opened in town selling the world's best pizzas. This changed the traffic on that street. Eg: It now was 50 commuters/hour instead of just 5 commuters/hour.

   You want to renew the most used streets. You are given a graph G with all the streets as edges, intersections as nodes and average usage as weights.

   (a) (12 points) Design an algorithm to update the list of n streets you received from question 4 that were supposed to be renewed, if the traffic at exactly one street (n_i) was changed from w_i to w'_i.
   Write a generic algorithm.

Note: The traffic may either increase or decrease for a particular street after opening of a new shop. Your algorithm should cover all possible cases.

Hint: Use the output from $Q4$ and your original graph (G), as comparison with the changed edge (n_i) to determine all possible cases your algorithm should cover.

Ans:

Let's say (u, v) is the street (edge) where new restaurant has been opened.

T is the spanning tree obtained from above question.

Scenario1:

Assume (u, v) is in the spanning tree obtained from the question $Q4$.

```
if (weight(u, v) got increased by x):
    spanning tree T remains same
else if (weight(u, v) got decreased by x):
    let (a, b) be the min weight in T
    if ((u, v) - x > (a, b)):
        spanning tree T remains same
    else:
        Remove the edge (u, v) from T
        Add (u, v) to the remaining sorted edges of the
            graph G in their sorted positions
        Pick the max edge from sorted edges such that it
            doesn't form a cycle in spanning(T)
```

Scenario2:

Assuming that (u, v) is not in the maximum spanning tree T obtained from question $Q4$.

```
if (weight(u, v) got decremented by x):
    spanning tree T remains same
else if (weight(u, v) got incremented by x):
    let (a, b) be the min weight in T
    if ((u, v) + x < (a, b)):
        spanning tree T remains same
    else if ((u, v) + x >= (a, b)):
        Remove the edge (a, b) from T
        Add (u, v), (a, b) to the remaining sorted edges of
            the graph G in their sorted positions
        Pick the max edge from sorted edges such that it
            doesn't form a cycle in spanning(T)
```

(b) (3 points) State your running time.

Ans: As we are using path compression and rank, they would complement each other, the amortized time complexity for cycle detection is constant. In worst case we need to process all the edges, Therefore, time complexity is $\mathcal{O}(E)$.