CS 5800: Algorithms SEC 05 Summer Full 2024 (Seattle)
Homework 5

Payal Chavan

06/21/2024

Question 1: You are given a strongly connected directed graph G = (V, E) with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding shortest paths between all pairs of nodes, with the one restriction that these paths must all pass through v_0.

Solution:

1. We shall use the Floyd-Warshall algorithm with a slight modification to approach this problem. The algorithm is as follows:
2. Create a $2D$ matrix M and D of size |V| x |V|, where |V| represents the number of vertices.
3. Initialize each cell in matrix (M) to represent the distance from vertex (i) to vertex (j).
4. Initialize each cell in matrix (D) to represent shortest distance vertex through which we reach from vertex (i) to vertex (j).
5. When there's no direct path from vertex (i) to vertex (j), you can assign the value of infinity to (M[i][j]).
6. When there is no direct path from vertex (i) to vertex (j), you can assign the value of None to (D[i][j]). For a direct edge, (D[i][j] = j).
7. To compute the shortest path between vertices (i) and (j), you can iterate over intermediate vertex (k) (ranging from 1 to n).
   Update M[i][j] if M[i][k] + M[k][j] is shorter:
   M[i][j] = min(M[i][j], M[i][k] + M[k][j])
8. Along with shortest path, save the vertex from which we got the shortest distance.
   D[i][j] = k
9. To find the shortest distance between vertices (u) and (v) that passes through an intermediate vertex (v_0), begin at vertex (u).
   M[i][v_0] + M[v_0][j]
10. To find the shortest path between vertices (u) and (v) that passes through an intermediate vertex (v_0), begin at vertex (u).
    We will first find the path from u to v_0 using matrix (D). D[i][k] gives node from which we reached to node k. Repeat the process until we reach to v_0. Lets say this is Path1.

    Similarly, do the same process from vertex v_0 to v. Lets say this is Path2.

    Path1 + Path2 is our shortest path from vertex u to v through v_0.
11. The final result will be the shortest distance between all pairs of nodes passing through v_0, which is M[i][j].

Question 2: Suppose we want to find the minimum spanning tree of the following graph.

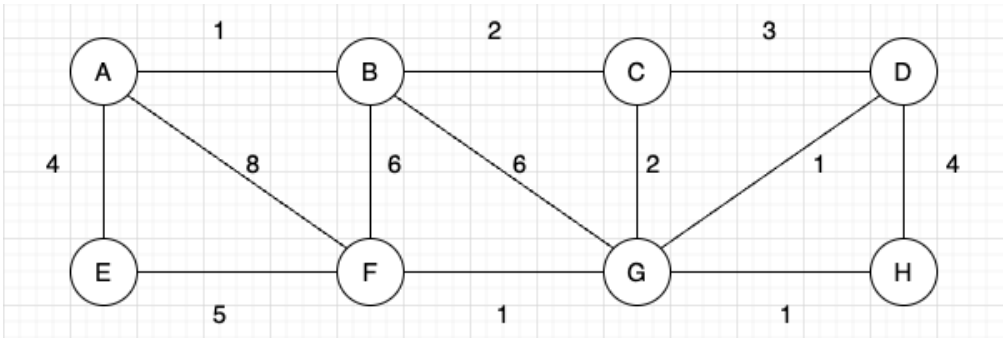

Figure 1: Q1. Graph

Part (a): Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node A). Draw a table showing the intermediate values of the cost array.

Solution (a):

In Prim's Algorithm, minimum edge weights are chosen to form a MST. This method always begins with a single node and proceeds through a number of neighboring nodes to investigate every related edge that it encounters.

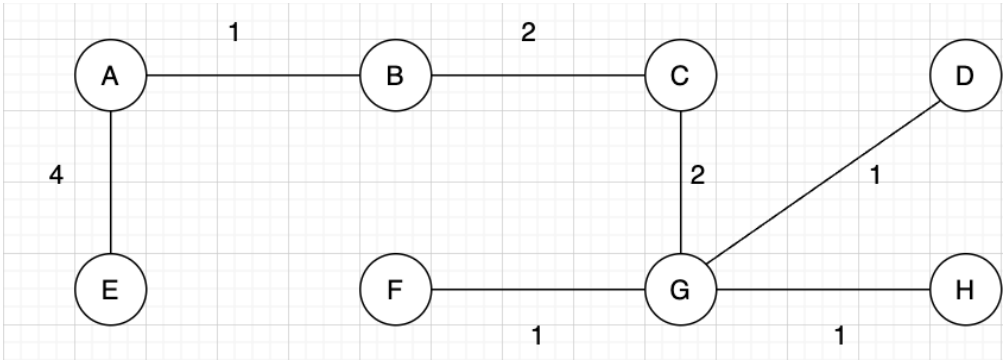

Figure 2: Prim's Algorithm MST

The order of MST is: $A \rightarrow B, B \rightarrow C, C \rightarrow G, G \rightarrow D, G \rightarrow H, G \rightarrow F, A \rightarrow E$.

The total cost for the above MST using Prim's Algorithm is 12.

The table for cost array for the Prim's Algorithm is shown below:

	A	B	C	D	E	F	G	H
Initial	0	inf	inf	inf	inf	inf	inf	inf
A	0	1/A	inf	inf	4/A	8/A	inf	inf
B	0	1/A	2/B	inf	4/A	6/B	6/B	inf
C	0	1/A	2/B	3/C	4/A	6/B	2/C	inf
G	0	1/A	2/B	1/G	4/A	1/G	2/C	1/G
D	0	1/A	2/B	1/G	4/A	1/G	2/C	1/G
H	0	1/A	2/B	1/G	4/A	1/G	2/C	1/G
F	0	1/A	2/B	1/G	4/A	1/G	2/C	1/G
E	0	1/A	2/B	1/G	4/A	1/G	2/C	1/G

Figure 3: Cost Array Table

Part (b): Run Kruskal's algorithm on the same graph. Show how the disjoint sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is used.

Solution (b):

In Kruskal's algorithm, all edges of the given graph are sorted in increasing order, without forming a cycle.
Order in which edges are added : A-B, D-G, F-G, G-H, B-C, C-G, A-E.

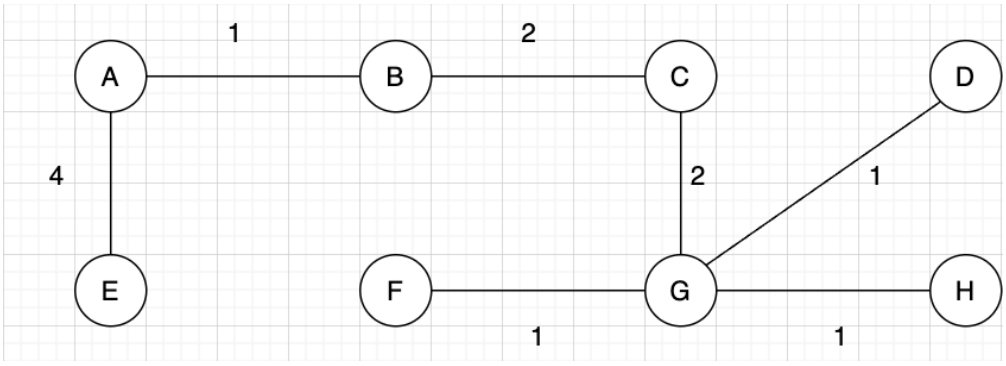

Figure 4: Kruskal's Algorithm MST

The total cost for the Kruskal's MST is also 12.

The disjoint-sets at every intermediate stage is shown below:

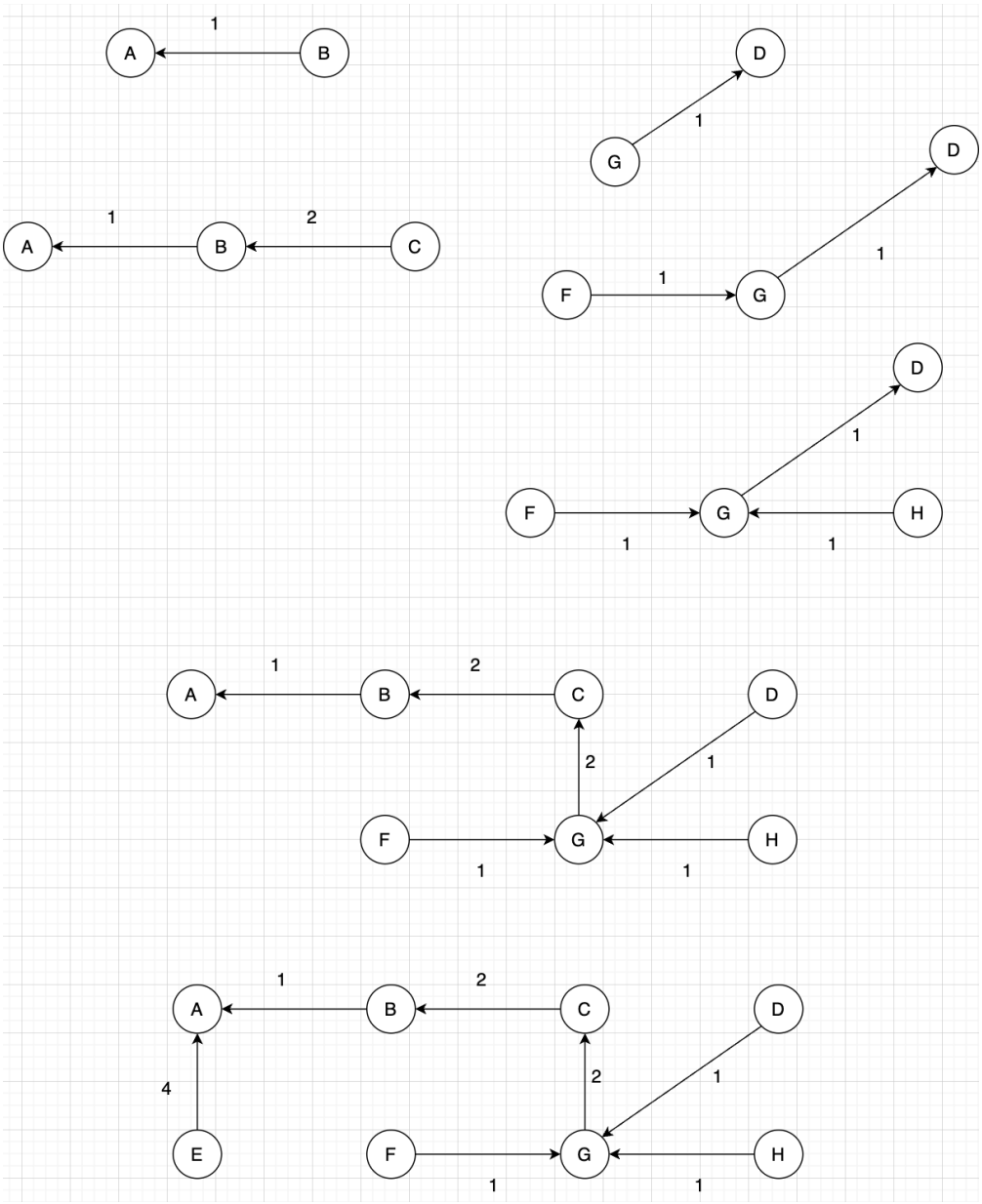

Figure 5: Sequence of disjoint-sets operations

Question 3: The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph G = (V, E) is undirected and connected. Do not assume that edge weights are distinct unless it is specifically stated.

Part (a): If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.

Solution (a):

The given statement is False.
In a graph, the minimum spanning tree (MST) includes all vertices without forming a cycle. Consequently, a single, unique heaviest edge can be part of the MST if it's the sole edge connecting to a vertex.

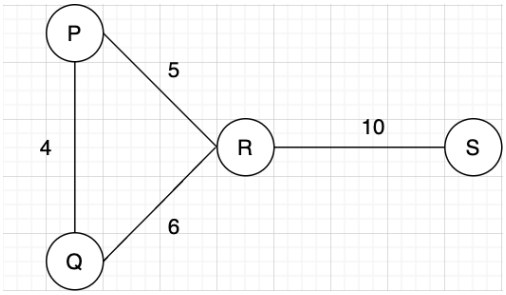

Figure 6: Example (a)

In the given graph, the edge RS has the heaviest weight, and it will be included in a minimum spanning tree because vertex S must be covered. Therefore, the statement is false.

Part (b): If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.

Solution (b):

The given statement is True.

By definition, an MST does not contain cycles. If there were a cycle, we could remove an edge from it to obtain a tree with the same vertices but a smaller total weight.
Suppose we have a graph (G) with a cycle that includes a single heaviest edge (e). When we consider a Minimum Spanning Tree (MST) denoted as (T), there must be at least one edge from this cycle that is not part of (T). We will refer to this edge as (e').
If $(e' \neq e)$, we can replace (e') with (e) in (T).
Since edge (e) is the only heaviest edge in the cycle, replacing edge (e') with (e) will not increase the total weight of the tree. If (e') is not equal to (e), we can always create a lighter spanning tree by replacing (e') with (e). This property guarantees that the unique heaviest edge (e) cannot be part of any Minimum Spanning Tree (MST).

Part (c): Let e be any edge of minimum weight in G. Then e must be part of some MST.

Solution (c):

The given statement is True.

If we have an edge (e) in graph (G) with the minimum weight among all edges, it becomes a strong candidate for inclusion in a Minimum Spanning Tree (MST). Since the sum of edge weights in an MST is minimized, (e) contributes to the lightest possible spanning tree. In other words, (e) must be part of some MST.

To prove this, let's assume that edge (e) is not part of any Minimum Spanning Tree (MST). We can construct a new spanning tree by adding (e) to any existing MST. This new tree will still connect all vertices and will have a total weight less than or equal to the original MST (since (e) has the minimum weight). However, this contradicts the definition of an MST, which must have the minimum possible total edge weight.

Part (d): If the lightest edge in a graph is unique, then it must be part of some MST.

Solution (d):

The given statement is True.

If a graph (G) has a unique lightest edge (e), and no other edge shares the same weight as (e), then (e) appears in every Minimum Spanning Tree (MST). Moreover, if the edge weights in (G) are all unique (meaning no two edges have the same weight), then (G) has a unique MST. The uniqueness of the lightest edge guarantees its inclusion in an MST, while the heaviest edge in any cycle is always excluded from any MST.

Part (e): If e is part of some MST of G, then it must be a lightest edge across some cut of G.

Solution (e):

The given statement is True.

An MST (Minimum Spanning Tree) is a tree that spans all the vertices of a graph while minimizing the sum of edge weights. It achieves this by including all the vertices and a subset of the edges, ensuring there are no cycles.
Now, consider a cut in the graph—a partition of its vertices into two disjoint sets. An edge that crosses this partition is known as a "cut edge." The key insight is that if an edge e belongs to some MST of graph G, then it must be the lightest (minimum weight) edge across some cut of G. In simpler terms, when e is part of an MST, there exists a cut in G such that e is the lightest edge that crosses that cut.

Part (f): If G has a cycle with a unique lightest edge e, then e must be part of every MST.

Solution (f):

The given statement is False.

Suppose we have a graph G with a cycle containing a unique lightest edge, denoted as e. This means that no other edge in that cycle has a smaller weight than e.
Now, consider removing e from the cycle. The remaining edges still form a connected subgraph, and this subgraph becomes a tree. Importantly, this tree is a Minimum Spanning Tree (MST) because it spans all the vertices and has the minimum total weight.
However, e need not be part of every MST of G. In other words, there can be other MSTs that do not include e. MSTs can have multiple valid solutions, and the presence of a unique lightest edge in a cycle doesn't guarantee its inclusion in every MST.

Part (g): The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST.

Solution (g):

The given statement is False.

When applying Dijkstra's Algorithm, the resulting tree consists of the shortest paths from the source vertex to other vertices. However, it's important to note that this tree is not necessarily a Minimum Spanning Tree (MST). Prim's algorithm on the other hand, constructs a Minimum Spanning Tree (MST) by greedily selecting edges. However, instead of focusing on distances, it prioritizes edges closest to any vertex currently in the working MST.

Consider the below graph,

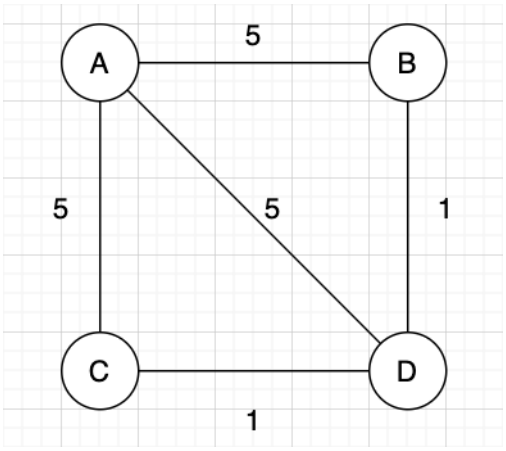

Figure 7: Example (g)

When applying Dijkstra's Algorithm, the resulting tree would include edges (AB), (AC), (AD). However, this tree does not necessarily form a Minimum Spanning Tree (MST).

Part (h): The shortest path between two nodes is necessarily part of some MST.

Solution (h):

The given statement is False.

In a Minimum Spanning Tree (MST), the goal is to connect all vertices while minimizing the total cost. This means reaching each vertex exactly once, creating a tree-like structure.
On the other hand, in Shortest Path problems, we focus only on finding the lowest-cost path from a source vertex to a destination vertex. We don't necessarily visit all other vertices; the emphasis is solely on the shortest weight between the source and destination.

To understand the difference, consider this example:

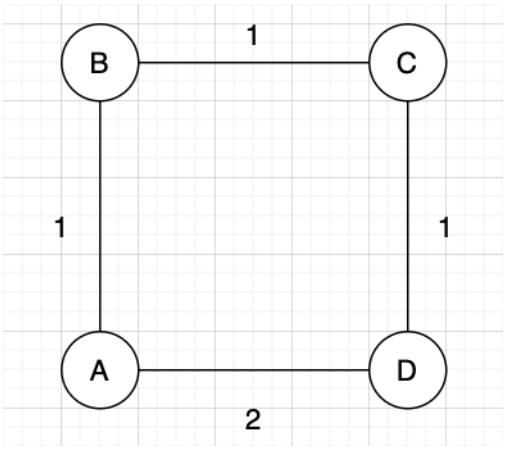

Figure 8: Example (h)

Suppose we have 4 vertices (A, B, C, D) having edges (AB, BC, CD, AD), and their respective weights are $(1, 1, 1, 2)$.
In MST case, edges (AB, BC, CD) will be on MST with total weight of 3. So cost of reaching A to D in MST is 3.
But in Shortest Path case, shortest path between A to D is AD which is 2. Edge (AD) was never on MST.

Question 4: A long string consists of the four characters A, C, G, T; they appear with frequency 31%, 20%, 9%, and 40%, respectively. What is the Huffman encoding of these four characters?

Solution:

Huffman encoding is a compression technique that reduces the size of data without losing information. It's particularly useful when dealing with data containing frequently occurring characters.
Given a string composed of four characters: A, C, G, and T. Their respective frequencies are 31%, 20%, 9%, and 40%.
For example, in a string of length 100, character A appears 31 times, C appears 20 times, G appears 9 times, and T appears 40 times.

Let us follow the below steps to solve this problem:

1. **Step 1:** Note down the characters and their frequencies.

Char	Frequency
A	0.31
C	0.2
G	0.09
T	0.4

Figure 9: Char/Freq Table

2. **Step 2:** Sort the characters by their frequencies in increasing order.

Char	Frequency
G	0.09
C	0.2
A	0.31
T	0.4

Figure 10: Sorted Char/Freq Table

3. **Step 3:** Create a binary tree by combining the two nodes with the lowest frequencies until there is only one node left.

   (a) The two characters with the smallest frequencies are:
   G: 0.09 and C: 0.2
   Combine G and C to create a new node with frequency $(G + C = 0.29)$.

(b) Update the existing table and again sort:
$G + C = 0.29$
$A = 0.31$
$T = 0.4$

(c) Once again combine the two nodes with the lowest frequencies:
$G + C + A = 0.6$
$T = 0.4$

(d) Update the table and sort them:
$T = 0.4$
$G + C + A = 0.6$

(e) Combine the remaining 2 nodes:
$G + C + A + T = 0.1$

Finally, we have the Huffman tree.

4. **Step 4:** Assign (0) to the left branch and (1) to the right branch, starting at the root. Navigate the tree to give each character a code.

Now let's label the codes and show the tree:

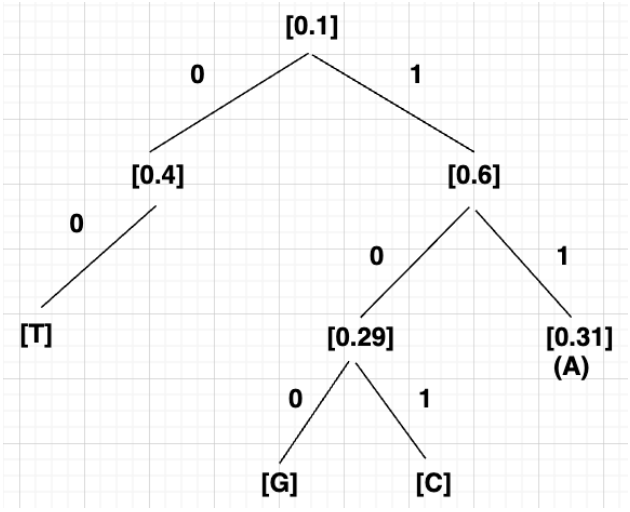

Figure 11: Huffman Encoding Tree

5. **Step 5:** Therefore, the final Huffman encoding of this we get as,
T: 00
G: 100
C: 101
A: 11

The same solution of this is verified through the Huffman Encoding code.

```
T 0
G 100
C 101
A 11
(base)
payalchavan@Payals-MacBook-Air /Users/payalchavan/Documents/Algorithms/Assignment5
```

Figure 12: Huffman Encoding Code Output

Reflection:

From this exercise, I learnt how to implement Prim's Algorithm and Kruskal's Algorithm on any given undirected connected graph. These algorithms provided valuable insights into drawing minimum spanning trees (MST). Additionally, I learnt Huffman Encoding concept and Floyd-Warshall's Algorithm.

Acknowledgements:

I would like to express my sincere gratitude to the following individuals and resources for their invaluable contributions to this assignment:
1) Prof. Bruce Maxwell: Thank you, Professor Bruce Maxwell for your guidance in this assignment. Your expertise in algorithms greatly influenced my work.
2) Classmates and TA's: I appreciate the discussions of my classmates, and TA's who clarified my doubts.
3) DPV Algorithms Textbook: This textbook was a useful resource for me to understand the basics of algorithms.
4) https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/: This website provided clear explanations for understanding Prim's Algorithm.
5) https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/?ref=lbp: This was a useful resource to understand Huffman Encoding.
6) https://www.programiz.com/dsa/kruskal-algorithm: This website provided clear explanations for understanding Kruskal's Algorithm.
7) https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/: This website provided clear explanations for understanding Floyd-Warshall Algorithm.

Time Travel Day 1