Combinatorial Optimization

Paths and Trees

Dias, Schiewea, Pattanaki *Instructor: Pattanaki*

§ Week II §

Problem 1: Shortest Distance

Compute the shortest-paths tree from s using Dijkstra's algorithm. Sketch each iteration.

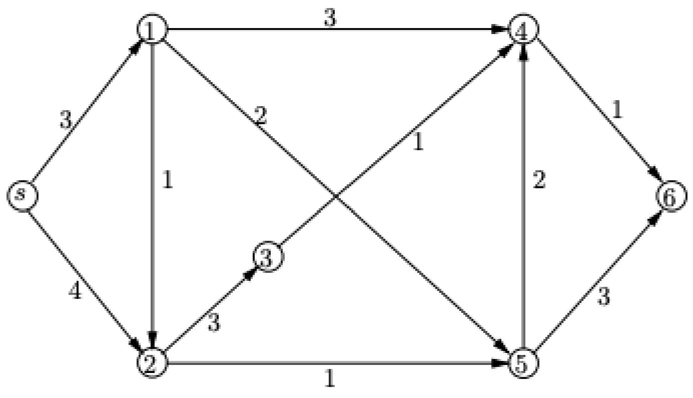


Figure 1: Example of a flow network

Solution:

Using Dijkstra's algorithm:

- We set the distance to all nodes as infinite and the predecessor to all of them as "unknown";
- We start from the source s, where we can update the predecessor for nodes 1 and 2 such that $p_1 = s$ and $p_2 = 1$; their distance is also updated from $d_1 = 3$ and $d_2 = 4$;
- We choose the minimal distance, which is to node 1. Onto the next iteration:
- From node 1, we can reach node 2, node 4 and node 5. With this in mind, we are updating the distances and predecessors: p_2 could be s or node 1; $p_4 = 1$ and $p_5 = 1$; the distance will be: for d_2 is either 4 (coming from node 1) or 4 (coming from source s), $d_5 = 5$ and $d_4 = 6$;
- Then, we choose the minimal distance once again: reaching node 2 from the source s ($p_2 = s$ with cost equal to $d_2 = 4$, setting node 2 for the next iteration;
- From node 2, we can reach node 3 and node 5; we update the predecessor and distances. For node 3, $d_3 = 7$ and $p_2 = 2$. For node 5, for d_5 we have two options: 5 (coming from node 1 $p_5 = 1$) or 5 (coming from node 2 $p_5 = 2$);
- We can choose whichever we want. Let's pick $p_5 = 1$, which set up node 5 for the next iteration;
- From node 5, we can reach node 4 and node 6. For node 4, we can update the possible predecessor and distance: p_4 can be 5 costing 7 or keep $p_4 = 1$ with a distance equal to 6. In this case, we have to choose the minimal. We can also set $d_6 = 8$ and $p_6 = 5$

- Then $p_4 = 1$ and $d_4 = 6$ which set up node 4 for the next iteration;
- From node 6, we can reach only node 6, so we update d_6 and p_6. d_6 can be either 8 (coming from 5 $p_6 = 5$) or 7 (coming from $p_6 = 4$). Once again, we choose the minimal;
- Then, $d_6 = 7$ and $p_6 = 4$;
- The only node not confirmed is not 3, which does not reach any other node that we haven't visited, which set up $d_3 = 7$ and $p_3 = 2$.

The updated table:

$$\begin{pmatrix} & d_v & p_v \\ 1 & s & 3 \\ 2 & s/\xcancel{1} & 4/\xcancel{4} \\ 3 & 2 & 7 \\ 4 & 1/\xcancel{5} & 6/\xcancel{7} \\ 5 & 1/\xcancel{2} & 5/\xcancel{5} \\ 6 & \xcancel{5}/4 & \xcancel{8}/7 \end{pmatrix}$$

The resulting structure is:

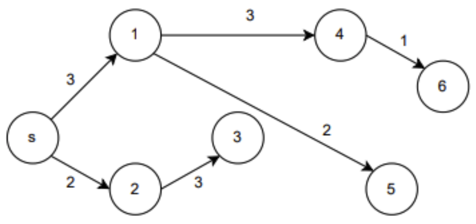


Figure 2: Solution for shortest path using Dijkstra's algorithm (using the description above)

Problem 2: Prim's vs Kruskal's

Suppose we have an undirected graph with weights that can be either positive or negative. Do Prim's and Kruskal's algorithim produce a MST for such a graph?

Solution:

Yes. Add some sufficient large positive constant (the absolute value of the smallest cost edge) to all edge costs so that all edge costs become non-negative. Then, apply either MST algorithm. It should be clear that Kruskal's algorithm will pick the same set of edges, regardless of whether we add a constant since the selection of edges only depends on ordering costs. For Prim, the same argument can be made. At each stage of Prim, a crossing edge of the least cost is chosen. The choice won't change if a constant is added to all edges.

Problem 3: Prim's variation

In some problems we want that certain pairs of vertices are directly connected with each other. Modify the Prim's algorithm in order to solve the minimum spanning tree problem with this additional constraint.

Solution:
There are several ways to modify Prim's algorithm in order to take into account this additional constraint.

One of them is the following. Given a graph G = (V, E), let S be the set of edges representing our additional constraint, i.e., S is the set of edges that should be included in the spanning tree. Now, set the weight of all the edges in S to some value less than the minimum weight of edges in E S. Then we run Prim's algorithm on the graph with modified weights. The cost of the spanning tree found is calculated using the original weights.

Problem 4: Reliability

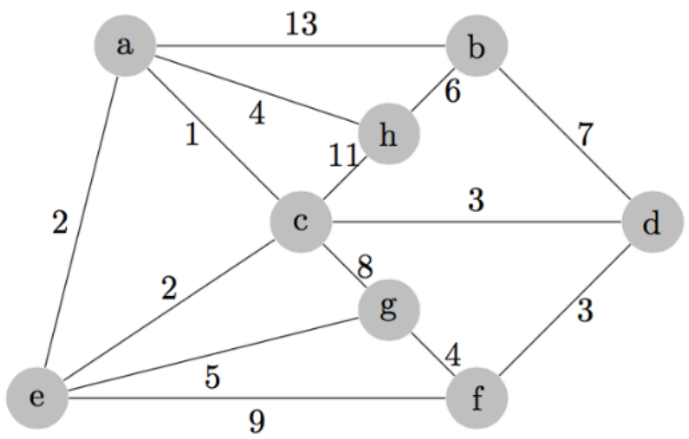
Let $G = (V, A)$ be a digraph with edge-weights $0 \geq p_a \geq 1$ for each edge $a \in A$. Interpret p_a as the probability that the arc a does fail. For an elementary path from a node s to a node t the reliability of that path can be understood as the product of all p_a of edges on that path; namely, it is the probability, that the path does not fail. Show, how a shortest path algorithm can help computing a path from s to t with maximal reliability.

Solution:

Using the probability values, we can create a graph where each edge represents the chance of failure between two nodes. The shortest path algorithm finds the shortest path possible between any node and the source. By using Dijkstra's algorithm, we ensure that the path from the source to that particular node is continuously formed by the combined edges, creating the most reliable path from the source to a node. By choosing the edges with minimal distance, we choose the smallest probability value and, therefore, the highest reliability.

Problem 5: Spanning Tree

For the graph below:



1. What is the cost of a minimum spanning tree?

   Solution:

   The MST for this graph has a total weight equal to **23**.

   Let us solve this problem via Kruskal:

   - First, we order all the edges in increasing weight value:
     $E = \{(a, c), (a, e), (c, e), (c, d), (d, f), (a, h), (g, f), (e, g), (b, h), (b, d), (g, c), (e, f), (h, c), (a, b)\}$
   - Now, let's build the tree using them. Remember that a cycle should be avoided at all costs:
     (a) Add edge (a, c), then we have two choices (a, e) and (c, e). Let's pick (a, e);
     (b) With the previous choice in mind, we cannot pick (c, e); otherwise, the cycle (a, c, e) would be created;

(c) Now, we can take both (c, d) and (d, f);
(d) Next, we add both (a, h) and (g, f);
(e) (e, g) cannot be added otherwise a cycle (a, c, d, f, g, e) would be created;
(f) Next, we add b, h;
(g) (b, d) cannot be added because of a cycle (a, h, b, d, c);
(h) (g, c) cannot be added because of a cycle (c, d, f, g);
(i) (e, f) cannot be added because of a cycle (a, c, d, f, e);
(j) (h, c) cannot be added because of a cycle (a, h, c);
(k) (a, b) cannot be added because of a cycle (a, h, b);

The resulting tree is as follows:

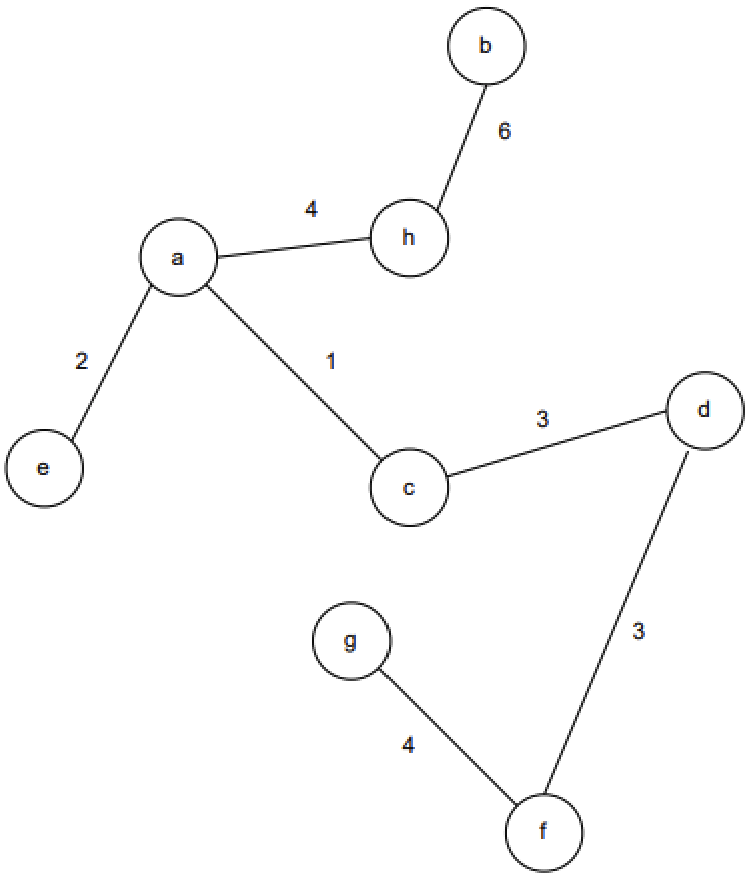


Figure 3: Solution from Kruskal's algorithm (as described above)

Now with Prim's algorithm:

- We can start from any node. Let's pick c;
  (a) From c, we choose the edge with smallest weight, which is (a, c);
  (b) From the "region" formed by nodes a and c, the smallest outgoing edges are (c, e) and (a, e). Let's pick (c, e). This automatically forbids (a, e) to be added because the edge would connect to a node already in the "region";
  (c) Now, from this updated "region", we can add both edges (c, d) and (d, f) sequentially;
  (d) Same can be done with edges (a, h) and (g, f);
  (e) Edge (e, g) cannot be used because it is an edge connecting to a node already in the "region";

(f) Edge (h, b) is now added, resulting in an update in the "region";
(g) All the remaining edges would point towards the "region", invalidating them as options.
(h)

The resulting tree is:

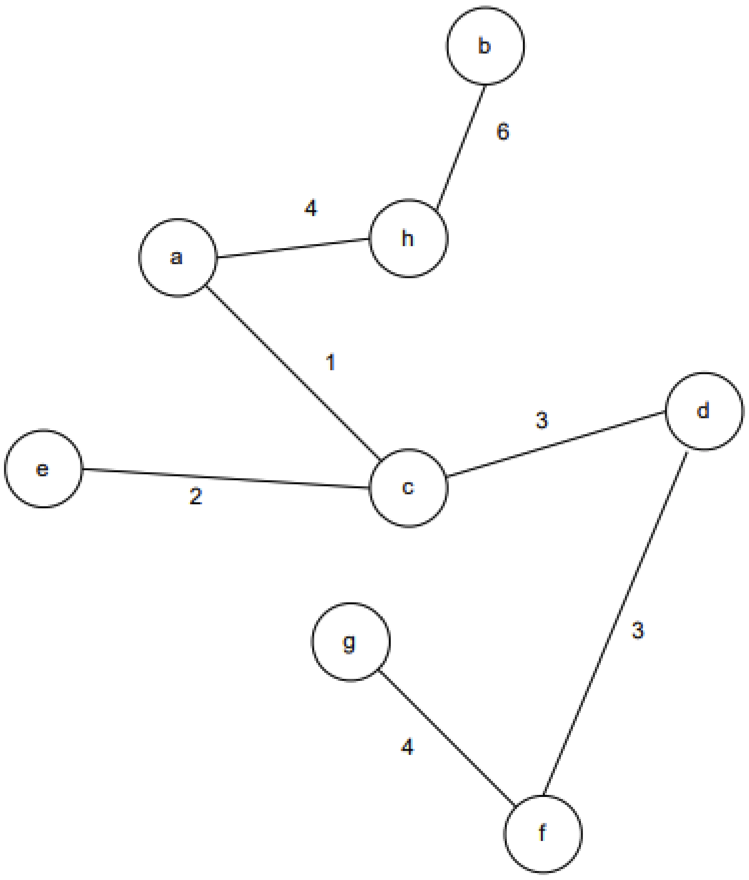


Figure 4: Solution from Prim's algorithm (as described above)

2. How many minimum spanning trees does it have?

Solution:

Based on the previous question, at least **two**.