Minimum Spanning Trees

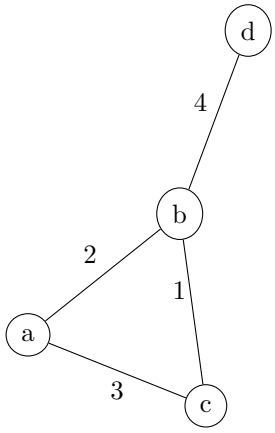
1. After calculating the minimum spanning tree, you decide to add a positive constant to all edge weights within the graph. Will this operation change the minimum spanning tree of the graph?

   Solution: No, this will not change the minimum spanning tree of the graph. Adding a positive constant to all edge weights increases all edge weights by the same constant, leaving the minimum edges still as the minimum edges in the graph.

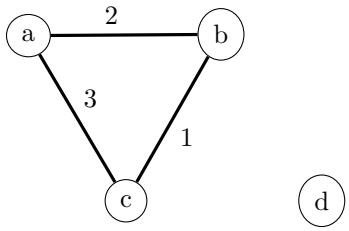
2. A colleague suggests the following algorithm to find the minimum spanning tree: Start with any vertex as a single-vertex minimum spanning tree, then add $V - 1$ edges to it, always taking next a min-weight edge connected to the vertex most recently added. Why does your colleague's algorithm not necessarily always produce a minimum spanning tree?

   Solution: Simply taking the minimum weighted edge connected to the most recently added vertex won't always result in a minimum spanning tree. For that matter, it may not even result in a spanning tree at all. Doing so assumes that that every such vertex is connected to a yet undiscovered vertex, which may not always be true. Furthermore, taking the minimum weighted edge may result in a cycle within the minimum spanning tree. Any minimum spanning tree that has a cycle within it cannot be a minimum spanning tree.

   Take, for example, the following graph.

   Were the minimum spanning tree generation to begin with vertex `a`, using the algorithm as described in the question would result in a minimum spanning tree like so:

   Vertex `d` has been left out of the solution. Therefore, the result is not a minimum spanning tree. Furthermore, the solution has a cycle so it could never be the minimum spanning tree.

3. A colleague argues that every graph has only one possible minimum spanning tree? Is your colleague correct? Under what cases can a graph have multiple minimum spanning trees?

Solution: No, all graphs may not have only one possible minimum spanning tree. For example, graphs with non-distinct edge weights may have multiple minimum spanning trees.

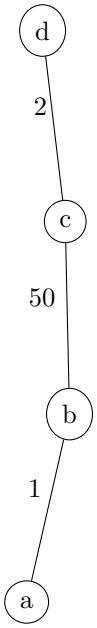
4. Describe an algorithm on how to find a *maximum* spanning tree?

Solution: Multiply all edge weights by -1 and apply any minimum spanning tree algorithm (e.g. Prim's, Kruskal's). Negate the edge weights once more after the minimum spanning tree algorithm completes to restore the edge weights to their original values.

5. Suppose that a graph has distinct edge weights. Does the edge with the smallest weight have to belong to the minimum spanning tree? Can the edge with the largest weight belong to the minimum spanning tree? Why or why not?

Solution: The minimum edge must belong to the minimum spanning tree. The cut property states that given a partitioning of the graph into two distinct sets, the minimum weighted edge in the crossing edges between the two partitions must be in the minimum spanning tree. In order to inspect every vertex for addition to the minimum spanning tree, the minimum weighted edge must appear in some cut of the graph.

There is no exclusion on the edge with the largest weight belonging to the minimum spanning tree. If the edge with maximum weight is the only edge connecting a particular vertex to the minimum spanning tree, then the edge with maximum weight must be on the graph. This is exemplified in the following graph.



In this case, the graph is equal to its own minimum spanning tree.

Shortest Paths

6. After calculating the shortest paths from a single source to all other vertices in a graph, you decide to add a positive constant to all edge weights within the graph. Will this operation change the shortest paths of the graph?

Solution: Yes, it is possible to change the shortest paths within a graph by adding positive edge weights. For example, consider a graph that has two paths from vertex a to vertex d: $a \xrightarrow{1} b \xrightarrow{1} c \xrightarrow{1} d = 3$ and $a \xrightarrow{4} d = 4$. In such a graph, the shortest path from vertex a to vertex d would be $a \xrightarrow{1} b \xrightarrow{1} c \xrightarrow{1} d = 3$. Adding a scalar value of 1 to each edge would result in paths from vertex a to vertex d updated to $a \xrightarrow{2} b \xrightarrow{2} c \xrightarrow{2} d = 6$ and $a \xrightarrow{5} d = 5$. With the new edge weights, the shortest path from vertex a to vertex d is $a \xrightarrow{5} d = 5$.

7. Why doesn't Dijkstra's algorithm work for a graph with negative edge weights?

Solution: Dijkstra's algorithm won't work for a graph with negative edge weights. This is because it selects an edge to be on the shortest path tree based on the the vertex that is currently closest to the source. Doing so may ignore a currently undiscovered vertex (and edge) that has a negative weight. Furthermore, Dijkstra's algorithm is a greedy algorithm. Once it makes a decision, it never undoes the decision. Therefore, when it later discovers the negative edge, it won't have made correct decisions up to that point.

8. A colleague thinks that a minimum spanning tree algorithm can also be used to determine the shortest paths between vertices in a graph. Is your colleague correct? Why or why not?

Solution: The colleague is not correct in thinking that a minimum spanning tree algorithm can also be used to find shortest paths in a graph. The goals of the two problems and algorithms are different from each other. A minimum spanning tree algorithm selects the set of edges that have the minimum weight. A shortest path algorithm is concerned with a path between two vertices, disregarding the relationship that other vertices play in the graph.

9. How can you modify Dijkstra's algorithm to efficiently find the longest paths between a single source vertex and all other vertices? Assume there are no cycles within the graph.

Solution: Dijkstra's algorithm can be modified to find longest paths efficiently by modifying the `relax` method to use the opposite relational operator (i.e. select a new edge if it creates a longer path than previously known).

10. You can use Dijkstra's algorithm to find the shortest path from a source vertex to all other vertices in the graph in $O(E + V \log_2 V)$ time. However, suppose you have a directed graph where each edge has a weight of either 1 or 2. For such a graph, how can you find the shortest path from a source vertex to all other vertices in the graph in $O(V + E)$ time?

Solution: Use breadth first search instead. For all edges that have an edge weight of 2, split them to two edges with an edge weight of 1 and a new node in between. The standard breadth first search algorithm to determine shortest paths now applies. Note that the directed graph may have cycles and isn't strictly a directed acyclic graph (DAG). This disqualifies the use of topological sort as a potential solution.

11. Suppose you want to calculate the shortest path from some n vertices to all other vertices in the graph. It might be tempting to run Dijkstra's algorithm n times. However, there is a more efficient way. How can you modify either the graph or Dijkstra's algorithm to support multiple source vertices?

Solution: Create a new source vertex which has zero weight edges to all desired source vertices. Run Dijkstra's algorithm on the new source vertex.