Minimum Spanning Tree

- What is a Minimum Spanning Tree.
- Constructing Minimum Spanning Trees.
- What is a Minimum-Cost Spanning Tree.
- Applications of Minimum Cost Spanning Trees.
- Prim’s Algorithm.
  - Example.
  - Implementation.
- Kruskal’s algorithm.
  - Example.
  - Implementation.
- Review Questions.
What is a Minimum Spanning Tree.

• Let \( G = (V, E) \) be a simple, connected, undirected graph that is not edge-weighted.

• A spanning tree of \( G \) is a free tree (i.e., a tree with no root) with \( |V| - 1 \) edges that connects all the vertices of the graph.

• Thus a minimum spanning tree for \( G \) is a graph, \( T = (V', E') \) with the following properties:
  - \( V' = V \)
  - \( T \) is connected
  - \( T \) is acyclic.

• A spanning tree is called a tree because every acyclic undirected graph can be viewed as a general, unordered tree. Because the edges are undirected, any vertex may be chosen to serve as the root of the tree.
Constructing Minimum Spanning Trees

- Any traversal of a connected, undirected graph visits all the vertices in that graph. The set of edges which are traversed during a traversal forms a spanning tree.

- For example, Fig:(b) shows the spanning tree obtained from a breadth-first traversal starting at vertex b.

- Similarly, Fig:(c) shows the spanning tree obtained from a depth-first traversal starting at vertex c.
What is a Minimum-Cost Spanning Tree

• For an edge-weighted, connected, undirected graph, G, the total cost of G is the sum of the weights on all its edges.
• A minimum-cost spanning tree for G is a minimum spanning tree of G that has the least total cost.
• Example: The graph

Has 16 spanning trees. Some are:

The graph has two minimum-cost spanning trees, each with a cost of 6:
Applications of Minimum-Cost Spanning Trees

Minimum-cost spanning trees have many applications. Some are:

• Building cable networks that join n locations with minimum cost.
• Building a road network that joins n cities with minimum cost.
• Obtaining an independent set of circuit equations for an electrical network.
• In pattern recognition minimal spanning trees can be used to find noisy pixels.
Prim’s Algorithm

• Prim’s algorithm finds a minimum cost spanning tree by selecting edges from the graph one-by-one as follows:
  • It starts with a tree, T, consisting of the starting vertex, x.
  • Then, it adds the shortest edge emanating from x that connects T to the rest of the graph.
  • It then moves to the added vertex and repeats the process.

Consider a graph G=(V, E);
Let T be a tree consisting of only the starting vertex x;
while (T has fewer than |V| vertices)
{
    find a smallest edge connecting T to G-T;
    add it to T;
}
Example

Trace Prim’s algorithm starting at vertex $a$:

<table>
<thead>
<tr>
<th>Pass:</th>
<th>initially</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>weight</th>
<th>$v_l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active vertex:</td>
<td>$a$</td>
<td>$d$</td>
<td>$e$</td>
<td>$f$</td>
<td>$c$</td>
<td>$b$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$\infty$</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>$a$</td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>$\infty$</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>$e$</td>
</tr>
<tr>
<td>$d$</td>
<td>$\infty$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>$a$</td>
</tr>
<tr>
<td>$e$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>$d$</td>
</tr>
<tr>
<td>$f$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>$e$</td>
</tr>
</tbody>
</table>

The resulting minimum-cost spanning tree is:
Implementation of Prim’s Algorithm.

• Prims algorithm can be implemented similar to the Dijskra’s algorithm as shown below:

```java
public static Graph primsAlgorithm(Graph g, Vertex start) {
    int n = g.getNumberOfVertices();
    Entry table[] = new Entry[n];
    for(int v = 0; v < n; v++)
        table[v] = new Entry();

    table[g.getIndex(start)].distance = 0;
    PriorityQueue queue = new BinaryHeap(g.getNumberOfEdges());
    queue.enqueue(new Association(new Integer(0), start));
    while(!queue.isEmpty()) {
        Association association = (Association)queue.dequeueMin();
        Vertex v1 = (Vertex) association.getValue();
        int n1 = g.getIndex(v1);
        if(!table[n1].known){
            table[n1].known = true;
            Iterator p = v1.getEmanatingEdges();
            while (p.hasNext()){
                Edge edge = (Edge) p.next();
                Vertex v2 = edge.getMate(v1);
                int n2 = g.getIndex(v2);
                Integer weight = (Integer) edge.getWeight();
                int d = weight.intValue();
```
if(!table[n2].known && table[n2].distance > d){
  table[n2].distance = d; table[n2].predecessor = v1;
  queue.enqueue(new Association(new Integer(d), v2));
}
}
}

GraphAsLists result = new GraphAsLists(false);
Iterator it = g.getVertices();
while (it.hasNext()){
  Vertex v = (Vertex) it.next();
  result.addVertex(v.getLabel());
}

it = g.getVertices();
while (it.hasNext()){
  Vertex v = (Vertex) it.next();
  if (v != start){
    int index = g.getIndex(v);
    String from = v.getLabel();
    String to = table[index].predecessor.getLabel();
    result.addEdge(from, to, new Integer(table[index].distance));
  }
}
return result;
Kruskal's Algorithm.

- Kruskal’s algorithm also finds the minimum cost spanning tree of a graph by adding edges one-by-one.

enqueue edges of G in a queue in increasing order of cost.
T = φ;
while (queue is not empty){
    dequeue an edge e;
    if (e does not create a cycle with edges in T)
        add e to T;
}
return T;
Example for Kruskal’s Algorithm.

Trace Kruskal's algorithm in finding a minimum-cost spanning tree for the undirected, weighted graph given below:

The minimum cost is: 24
```java
public static Graph kruskalsAlgorithm(Graph g) {
    Graph result = new GraphAsLists(false);
    Iterator it = g.getVertices();
    while (it.hasNext()) {
        Vertex v = (Vertex) it.next();
        result.addVertex(v.getLabel());
    }
    PriorityQueue queue = new BinaryHeap(g.getNumberOfEdges());
    it = g.getEdges();
    while (it.hasNext()) {
        Edge e = (Edge) it.next();
        if (e.getWeight() == null)
            throw new IllegalArgumentException("Graph is not weighted");
        queue.enqueue(e);
    }
    while (!queue.isEmpty()) {
        Edge e = (Edge) queue.dequeueMin();
        String from = e.getFromVertex().getLabel();
        String to = e.getToVertex().getLabel();
        if (!result.isReachable(from, to))
            throw new IllegalArgumentException("Graph is not weighted");
        queue.enqueue(e);
    }
    while (!queue.isEmpty()) {
        Edge e = (Edge) queue.dequeueMin();
        String from = e.getFromVertex().getLabel();
        String to = e.getToVertex().getLabel();
        if (!result.isReachable(from, to))
            result.addEdge(from, to, e.getWeight());
    }
    return result;
}
```
public abstract class AbstractGraph implements Graph {
    public boolean isReachable(String from, String to) {
        Vertex fromVertex = getVertex(from);
        Vertex toVertex = getVertex(to);
        if (fromVertex == null || toVertex == null)
            throw new IllegalArgumentException("Vertex not in the graph");
        PathVisitor visitor = new PathVisitor(toVertex);
        this.preorderDepthFirstTraversal(visitor, fromVertex);
        return visitor.isReached();
    }

    private class PathVisitor implements Visitor {
        boolean reached = false;
        Vertex target;
        PathVisitor(Vertex t) { target = t; }

        public void visit(Object obj) {
            Vertex v = (Vertex) obj;
            if (v.equals(target)) reached = true;
        }
        public boolean isDone() { return reached; }
        boolean isReached() { return reached; }
    }
}
Prim’s and Kruskal’s Algorithms

Note: It is not necessary that Prim's and Kruskal's algorithm generate the same minimum-cost spanning tree.

For example for the graph:

Kruskal's algorithm (that imposes an ordering on edges with equal weights) results in the following minimum cost spanning tree:

The same tree is generated by Prim's algorithm if the start vertex is any of: A, B, or D; however if the start vertex is C the minimum cost spanning tree is:
Review Questions

1. Find the breadth-first spanning tree and depth-first spanning tree of the graph $G_A$ shown above.
2. For the graph $G_B$ shown above, trace the execution of Prim's algorithm as it finds the minimum-cost spanning tree of the graph starting from vertex $a$.
3. Repeat question 2 above using Kruskal's algorithm.