Spanning Tree in a Graph

- **Vertex** = router
- **Edge** = link between routers
- **Spanning tree** - Connects all the vertices
- **No cycles**

Undirected Graph

- $G = (V, E)$
  - $V$ is a set of vertices (or nodes)
  - $E$ is a set of unordered pairs of vertices

Example Graph:

- $V = \{1,2,3,4,5,6,7\}$
- $E = \{(1,2),(1,6),(1,5),(2,7),(2,3),(3,4),(4,7),(4,5),(5,6)\}$

- 2 and 3 are adjacent
- 2 is incident to edge $(2,3)$

Spanning Tree Problem

- **Input**: An undirected graph $G = (V, E)$. $G$ is connected.
- **Output**: $T$ contained in $E$ such that
  - $(V, T)$ is a connected graph
  - $(V, T)$ has no cycles

Spanning Tree Algorithm

```plaintext
ST(i: vertex)
mark i;
for each j adjacent to i do
  if j is unmarked then
    Add (i,j) to T;
    ST(j);
end(ST)
end ST
```

Main

- $T :=$ empty set;
- ST(1);
- end(Main)
Example of Depth First Search

Example Step 2

Example Step 3

Example Step 4

Example Step 5

Example Step 6
Example Step 7

Example Step 8

Example Step 9

Example Step 10

Example Step 11

Example Step 12
Example Step 13

Example Step 14

Example Step 15

Example Step 16

Best Spanning Tree

• Each edge has the probability that it won't fail
• Find the spanning tree that is least likely to fail

Example of a Spanning Tree

Probability of success = .85 x .95 x .89 x .95 x 1.0 x .84
= .5735
Minimum Spanning Trees

Given an undirected graph $G = (V,E)$, find a graph $G' = (V,E')$ such that:
- $E'$ is a subset of $E$
- $|E'| = |V| - 1$
- $G'$ is connected
- $\sum_{(u,v) \in E'} c_{uv}$ is minimal

Applications: wiring a house, power grids, Internet connections

Minimum Spanning Tree Problem

- Input: Undirected Graph $G = (V,E)$ and a cost function $C$ from $E$ to the reals. $C(e)$ is the cost of edge $e$.
- Output: A spanning tree $T$ with minimum total cost. That is: $T$ that minimizes

$$C(T) = \sum_{e \in T} C(e)$$

Find the MST

How many edges in a MST?

What is the total cost of each MST?

Two Different Approaches

Prim’s Algorithm
Almost identical to Dijkstra’s

Kruskals’s Algorithm
Completely different!

Prim’s algorithm

Idea: Grow a tree by adding an edge from the "known" vertices to the "unknown" vertices. Pick the edge with the smallest weight.

Prim’s Algorithm for MST

A node-based greedy algorithm
Builds MST by greedily adding nodes

1. Select a node to be the “root”
   - mark it as known
   - Update cost of all its neighbors
2. While there are unknown nodes left in the graph
   a. Select an unknown node $b$ with the smallest cost from some known node $a$
   b. Mark $b$ as known
   c. Add $(a, b)$ to MST
   d. Update cost of all nodes adjacent to $b$
Find MST using Prim’s

<table>
<thead>
<tr>
<th>V</th>
<th>Kwn</th>
<th>Distance</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Order Declared Known: v1

Prim’s Algorithm Analysis

Running time: Same as Dijkstra’s: \(O(|E| \log |V|)\)

Correctness:
Proof is similar to Dijkstra’s

Kruskal’s MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an \textit{edge with the smallest weight}.

\(G=(V,E)\)

Kruskal’s Algorithm for MST

\textit{An edge-based greedy algorithm}
Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge \((u, v)\) and mark it
   b. If \(u\) and \(v\) are not already connected, add \((u, v)\) to the MST and mark \(u\) and \(v\) as connected to each other

Example of Kruskal 1

Example of Kruskal 2
Example of Kruskal 2

Example of Kruskal 3

Example of Kruskal 4

Example of Kruskal 5

Example of Kruskal 6

Example of Kruskal 7
Kruskal’s Algorithm: Correctness
It clearly generates a spanning tree. Call it \( T_K \).

Suppose \( T_K \) is not minimum:

Pick another spanning tree \( T_{\text{min}} \) with lower cost than \( T_K \).

Pick the smallest edge \( e_1 = (u, v) \) in \( T_K \) that is not in \( T_{\text{min}} \).

\( T_{\text{min}} \) already has a path \( p \) in \( T_{\text{min}} \) from \( u \) to \( v \).

\( \Rightarrow \) Adding \( e_1 \) to \( T_{\text{min}} \) will create a cycle in \( T_{\text{min}} \).

Pick an edge \( e_2 \) in \( p \) that Kruskal’s algorithm considered after adding \( e_1 \) (must exist: \( u \) and \( v \) unconnected when \( e_1 \) considered).

\( \Rightarrow \) cost(\( e_2 \)) \( \geq \) cost(\( e_1 \))

\( \Rightarrow \) can replace \( e_2 \) with \( e_1 \) in \( T_{\text{min}} \) without increasing cost!

Keep doing this until \( T_{\text{min}} \) is identical to \( T_K \).

\( \Rightarrow \) \( T_K \) must also be minimal – contradiction!