A **Spanning Tree** on a connected graph $G$ is a subgraph, $G'$, such that:

1. Every vertex is in $G'$ and
2. $G'$ is connected with the minimum number of edges

This construction will always create a new graph that is a ________ (connected, acyclic graph) that spans $G$.

---

A **Minimum Spanning Tree** is a spanning tree with the **minimal total edge weights** among all spanning trees.

- Every edge must have a weight
  - The weights are unconstrained, except they must be additive (e.g., can be negative, can be non-integers)
- Output of a MST algorithm produces $G'$:
  - $G'$ is a spanning graph of $G$
  - $G'$ is a tree

---

G' has a minimal total weight among all spanning trees. There may be multiple minimum spanning trees, but they will have the same total weight.

---

**Pseudocode for Kruskal’s MST Algorithm**

```
Pseudocode for Kruskal’s MST Algorithm
1   KruskalMST(G):
2     DisjointSets forest
3     foreach (Vertex v : G):
4         forest.makeSet(v)
5     PriorityQueue Q    // min edge weight
6     foreach (Edge e : G):
7         Q.insert(e)
8     Graph T = (V, {})
9     while |T.edges()| < n-1:
10        Vertex (u, v) = Q.removeMin()
11        if forest.find(u) == forest.find(v):
12           T.addEdge(u, v)
13           forest.union( forest.find(u),
14                           forest.find(v) )
15      return T
```

---

**Kruskal’s Algorithm**

```
KruskalMST(G):
DisjointSets forest
foreach (Vertex v : G):
    forest.makeSet(v)
PriorityQueue Q    // min edge weight
foreach (Edge e : G):
    Q.insert(e)
Graph T = (V, {})
while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
        T.addEdge(u, v)
        forest.union( forest.find(u),
                        forest.find(v) )
return T
```

---

"The Muddy City" by CS Unplugged, Creative Commons BY-NC-SA 4.0
**Kruskal’s Running Time Analysis**

We have multiple choices on which underlying data structure to use to build the Priority Queue used in Kruskal’s Algorithm:

<table>
<thead>
<tr>
<th>Priority Queue Implementation</th>
<th>Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>6-8</td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Based on our algorithm choice:

<table>
<thead>
<tr>
<th>Priority Queue Implementation</th>
<th>Total Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>

**Reflections**

Why would we prefer a Heap?

Why would be prefer a Sorted Array?

---

**Partition Property**

Consider an arbitrary partition of the vertices on $G$ into two subsets $U$ and $V$.

Let $e$ be an edge of minimum weight across the partition.

Then $e$ is part of some minimum spanning tree.

*Proof in CS 374!*

---

**Prim’s Minimum Spanning Tree Algorithm**

---

**CS 225 – Things To Be Doing:**

1. `mp_mazes` due Monday!
2. If your final project has not been approved get it revised.
3. Daily POTDs are ongoing for +1 point /problem but pausing over break
4. No lecture Friday