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 have equal total weight!

- We covered the first classical algorithm (Kruskal) already!

**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 MST Algorithm**

**Pseudocode for Prim’s MST Algorithm**

```
1 PrimMST(G, s):
2     Input: G, Graph;
3     s, vertex in G, starting vertex of algorithm
4     Output: T, a minimum spanning tree (MST) of G
5     foreach (Vertex v : G):
6         d[v] = +inf
7         p[v] = NULL
8     d[s] = 0
9     PriorityQueue Q   // min distance, defined by d[v]
10     Q.buildHeap(G.vertices())
11     Graph T           // "labeled set"
12     repeat n times:
13         Vertex m = Q.removeMin()
14         T.add(m)
15         foreach (Vertex v : neighbors of m not in T):
16             if cost(v, m) < d[v]:
17                 d[v] = cost(v, m)
18                 p[v] = m
19     return T
```

---

<table>
<thead>
<tr>
<th>Adj. Matrix</th>
<th>Adj. List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Unsorted Array</td>
</tr>
</tbody>
</table>
Running Time of MST Algorithms

<table>
<thead>
<tr>
<th>Kruskal’s MST</th>
<th>Prim’s MST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q: What must be true about the connectivity of a graph when running an MST algorithm?

...what does this imply about the relationship between \( n \) and \( m \)?

<table>
<thead>
<tr>
<th>Kruskal’s MST</th>
<th>Prim’s MST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q: Suppose we built a new heap that optimized the decrease-key operation, where decreasing the value of a key in a heap updates the heap in amortized constant time, or \( O(1)^* \). How does that change Prim’s Algorithm runtime?

Final big-O Running Times of classical MST algorithms:

<table>
<thead>
<tr>
<th>Kruskal’s MST</th>
<th>Prim’s MST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shortest Path Home:

Dijkstra’s Algorithm (Single Source Shortest Path)

Dijkstra’s Algorithm Overview:
- The overall logic is the same as Prim’s Algorithm
- We will modify the code in only two places – both involving the update to the distance metric.
- The result is a directed acyclic graph or DAG

CS 225 – Things To Be Doing:

1. Programming Exam C is on-going
   Exam: Sunday, Dec 2 – Tuesday, Dec 4
2. MP7 Released – Slightly different structure: Hard Deadline on Monday, Dec. 3 (TONIGHT) for Part 1
   EC Due on Wednesday, Dec. 5 (combing story)
3. lab_finale in lab this week!
4. Daily POTDs are ongoing for +1 point /problem