Last lecture for the semester and we are ready to finish our unit on graphs!
Today, we will talk about minimum spanning trees and will see two famous algorithms to calculate them.
Given an edge weighted graph \( G \) (not a digraph), a spanning tree of \( G \) is a subgraph \( T \) that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices of \( G \).

Given an edge weighted graph (not a digraph), a spanning tree will be a subgraph that is a tree (that is connected and acyclic), and spanning (that is it includes all the vertices of the original graph). For example, on the graph above, you can see three possible spanning trees. They all are subgraphs that contain all three vertices, and form a connected and acyclic graph.
Let’s talk about some graph properties in relation to spanning trees. As we already saw, a graph can have more than one spanning tree and all spanning trees of a graph have the same number of vertices and edges. A spanning tree has \( n-1 \) edges, where \( n \) is the number of nodes (vertices). By definition, it cannot have a cycle. And adding an edge to it would create a cycle (that is known as maximally acyclic). Also, removing one edge from the spanning tree would disconnect it (that is known as minimally connected).
Minimum spanning tree (MST) problem

- Given a connected edge-weighted undirected graph find a spanning tree of minimum weight.

The minimum spanning tree problem states that given a connected edge-weighted undirected graph, we have to find a spanning tree of minimum weight, e.g., as in the example above.
There are a lot of scenarios that finding a minimum spanning tree is useful and I have a couple of links here but in general it makes sense when you want fully connected system at the lowest possible cost.
Let's see how to calculate MSTs with our first such algorithm, Kruskal's.
Kruskal’s algorithm

- Sort edges in ascending order of weight.
- Starting from the one with the smallest weight, add it to the MST unless doing so would create a cycle.
- Running time of $|E| \log |V|$ in worst case.
- Uses union-find, a data structure we haven’t covered.

It’s a very simple algorithm. You sort edges in ascending order of weight. You start from the one with the smallest weight, add it to the MST unless doing so would create a cycle. That can be done in $E \log V$ time in worst case. Kruskal’s algorithm uses union-find, a data structure we haven’t covered.
Let's assume we have this edge-weighted graph and the weights on the right that are in ascending order of weight.
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.

We start at 0-7 and add the edge to the MST.
Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.

next comes 2-3 which does not create a cycle so it gets added to the MST.
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

Same for 1-7
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

and 5-7
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

But not 1-3 since it would create a cycle.
Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

Nor 2-7
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

But 4-5 would be OK
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree \(T\) unless doing so would create a cycle.

But not 1-2
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.

nor 4-7
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.

Nor 0-4
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.

We can add 6-2
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

The rest all would create cycles
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle.
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree \( T \) unless doing so would create a cycle.
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

Tada! we have a minimum spanning tree!
Try to run Kruskal’s algorithm on this graph.
This is the MST you should have gotten.
Lecture 24: Minimum Spanning Trees

- Introduction
- Kruskal’s Algorithm
- Prim’s Algorithm

Let’s see how Prim’s algorithm calculates the MST.
Prim’s algorithm

- Start with a random vertex (here, 0) and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $|V| - 1$ edges.

- Two versions, lazy and eager. We will see lazy, here...
- Uses min-priority queue.
- Running time of $|E| \log |V|$ in worst case, as well.

We will start with a random vertex say 0, and greedily grow the MST by adding the min weight edge with exactly one endpoint in the MST and repeat it n-1 times. There are two versions, a lazy and eager and we will see the lazy approach here. The algorithm uses a min-priority queue and its running time is also $E \log V$. 
Let's see the algorithm on the graph above.

**Prim's algorithm demo**

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V-1$ edges.

We'll start with vertex 0
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

and consider all the edges that have an endpoint to our MST which right now only contains 0. That would be edges 0-2, 0-4, 6-0, and 0-7. The one with minimum weight is edge 0-7 so we will add it to the MST which now includes edge 0-7.
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

The edges with one endpoint in $T$ would be 1-7, 0-2, 5-7, 2-7, 4-7, 0-4, and 6-0. The minimum is 1-7
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

Now MST contains the edges 0-7 and 1-7
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V-1$ edges.

We proceed similarly with the rest which adds the 0-2 edge.
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

so now the MST contains edges 0-7, 1-7, and 0-2.
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

Net one would be 2-3
<table>
<thead>
<tr>
<th>Prim's algorithm demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Start with vertex 0 and greedily grow tree $T$.</td>
</tr>
<tr>
<td>• Add to $T$ the min weight edge with exactly one endpoint in $T$.</td>
</tr>
<tr>
<td>• Repeat until $V - 1$ edges.</td>
</tr>
</tbody>
</table>

MST edges
0-7 1-7 0-2 2-3
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree \( T \).
- Add to \( T \) the min weight edge with exactly one endpoint in \( T \).
- Repeat until \( V - 1 \) edges.

followed by 5-7
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

MST edges

0-7  1-7  0-2  2-3  5-7
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $|V| - 1$ edges.

MST edges
0-7 1-7 0-2 2-3 5-7

Edges with exactly one endpoint in $T$ (sorted by weight)

4-5 0.35
4-7 0.37
0-4 0.38
6-2 0.40
3-6 0.52
6-0 0.58

Min weight edge with exactly one endpoint in $T$
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

**MST edges**

0-7  1-7  0-2  2-3  5-7  4-5
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.

**MST edges**
- 0-7
- 1-7
- 0-2
- 2-3
- 5-7
- 4-5

and 6-2
Prim's algorithm demo

- Start with vertex 0 and greedily grow tree $T$.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V-1$ edges.

We have repeated this process $V-1$ times and this is the MST we ended up with.
Let's apply Prim's algorithm starting at index 0.
Answer

and here's the answer!
Lecture 24: Minimum Spanning Trees

- Introduction
- Kruskal’s Algorithm
- Prim’s Algorithm
Readings:

- Recommended Textbook: Chapter 4.3 (Pages 604-629)
- Website:
  - https://algs4.cs.princeton.edu/43mst/
- Visualization:
  - https://visualgo.net/en/mst
Problem

- Run Kruskal’s and Prim’s algorithm (starting at index 0) on the following graph:
Problem

- Run Kruskal’s and Prim’s algorithm (starting at index 0) on the following graph.
- Both will provide the same MST: