Minimum Spanning Trees

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston	Ethan Knutson	Nathan Lipiarski
Amanda Park	Farrell Fileas	Sam Long
Anish Velagapudi	Howard Xiao	Yifan Bai
Brian Chan	Jade Watkins	Yuma Tou
Elena Spasova	Lea Quan	

Announcements

- HW7 coming soon
  - HW6 and HW7 will be out concurrently; please prefix Piazza posts with "HW6: …" or "HW7: …"
  - HW7 exercises a different set of skills: reading/understanding a large codebase and figuring out where to plug in
  - **Read the spec carefully**; HW7 substantially longer than last quarter's because we added **a ton of hints**

- 20sp instructors want current students to TA next quarter!
  - Check Piazza or course webpage for more details

- Did you find the midterm review session useful? Come to a workshop!
  - Wed 2:30-3:20 and Fri 11:30-12:20 @ CSE 203

- **Cite your sources**. PLEASE. It's a crucial habit to get into.

Feedback from Reading Quiz

- I don’t understand the cut property and/or how it relates to MSTs
- Will we be studying non-greedy algorithms later?
- How is a MST different from a weighted tree?

Poll Everywhere

pollev.com/uwcse373

How’d the midterm go?

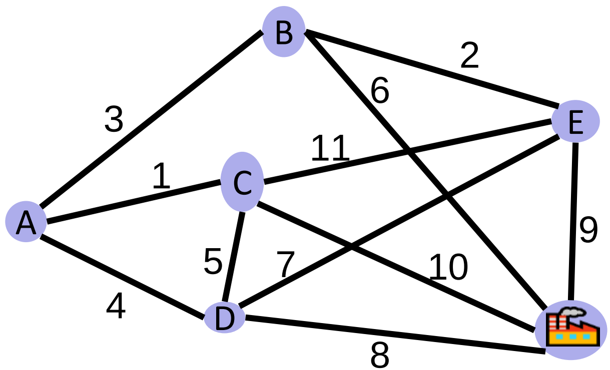
A. Crushed it
B. Tough, but I think it went ok
C. Coulda done better
D. Ugh
E. 😭
F. I’m not sure …

Lecture Outline

- **Introduction to Minimum Spanning Trees**
- Prim's Algorithm
- Kruskal's Algorithm
- Applications of MSTs

Problem Statement

- Your friend at the electric company needs to connect all these cities to the power plant
- She knows the cost to lay wires between any pair of cities and wants the cheapest way to ensure electricity gets to every city

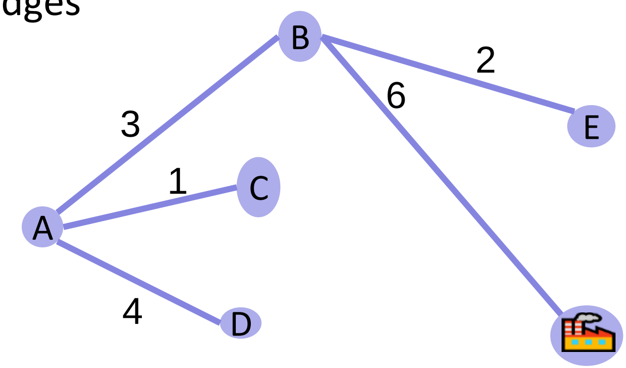


- Assume:
  - All edge weights are positive
  - The graph is undirected

Solution Statement

- We need a set of edges such that:
  - Every vertex touches at least one edges ("the edges **span** the graph")
  - The graph using just those edges is **connected**
  - The total weight of these edges is **minimized**

- Claim: The set of edges we pick never forms a cycle. Why?
  - V-1 edges is the exact number of edges to connect all vertices
  - Taking away 1 edge breaks connectiveness
  - Adding 1 edge makes a cycle



Poll Everywhere

pollev.com/uwcse373

- Which of these are trees?

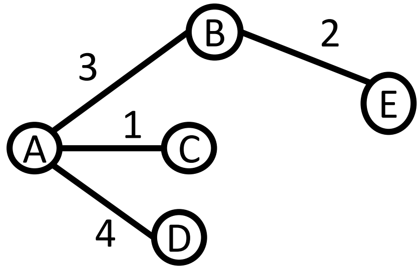
A. Tree / Not-Tree / Not-Tree

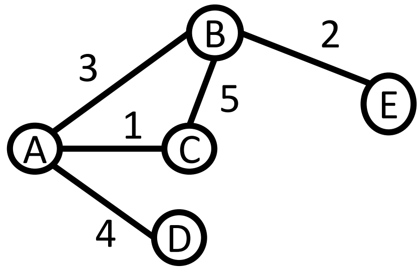
B. Tree / Tree / Not-Tree

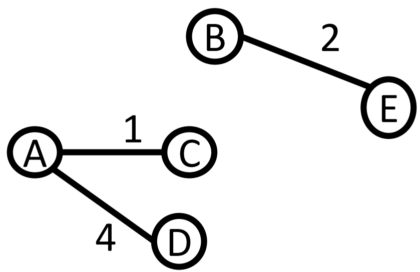
C. Tree / Not-Tree / Tree

D. Tree / Tree / Tree

E. I’m not sure …

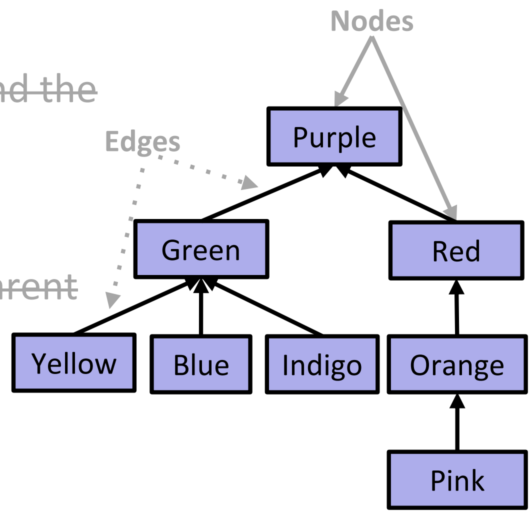


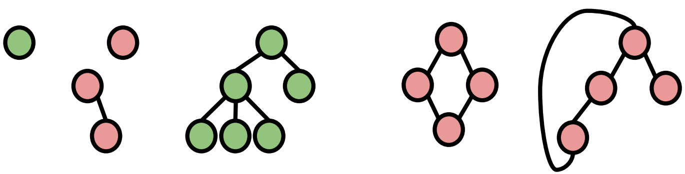




Review (AGAIN?!?!): The Tree Data Structure

- A **Tree** is a collection of nodes~~; each node has <= 1 parent and >= 0 children~~
  - ~~**Root node**: the "top" of the tree and the only node with no parent~~
  - ~~**Leaf node**: a node with no children~~
  - **Edge**: the connection between ~~a parent and child~~ two nodes
  - There is exactly one path between any pair of nodes

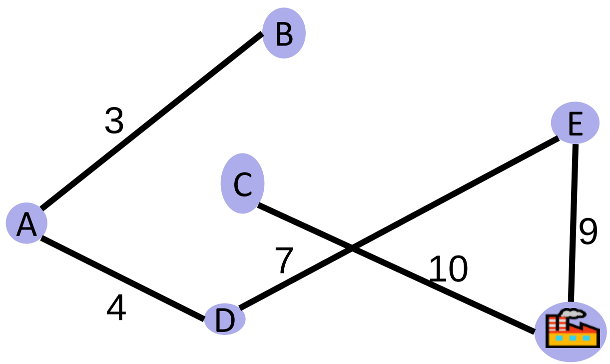


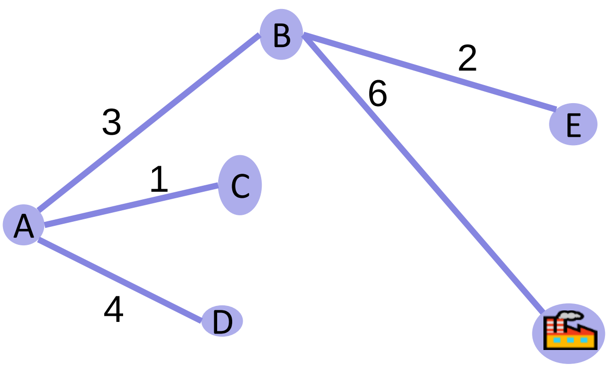


A tree is a connected acyclic graph!

Solution Statement (v2)

- We need a ~~set of edges such that~~ Minimum Spanning Tree:
  - Every vertex touches at least one edges (“the edges **span** the graph”)
  - The graph using just those edges is **connected**
  - The total weight of these edges is **minimized**



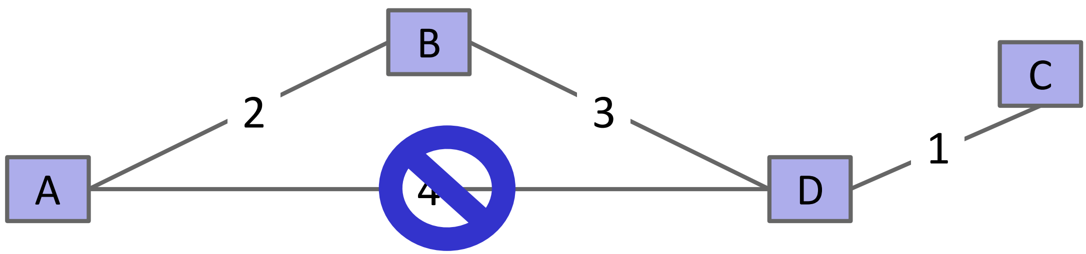


Lecture Outline

- Minimum Spanning Trees
- **Prim's Algorithm**
- Kruskal's Algorithm
- Applications of MSTs

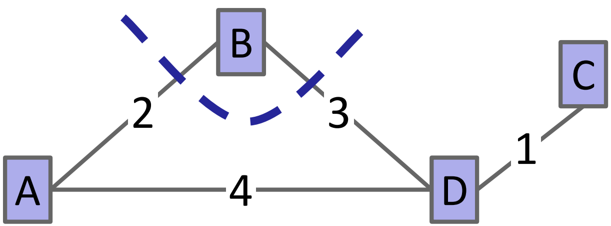
Cycle Property

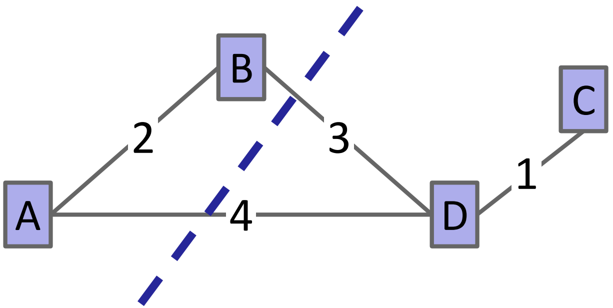
- Given any cycle, the heaviest edge along it must NOT be in the MST



Cut Property

- Given any cut, the minimum-weight crossing edge must be IN the MST
  - A cut is a partitioning of the vertices into two sets
  - (other crossing edges can also be in the MST)





If only we knew of an algorithm that repeatedly divided the vertices into two sets and chose the minimum edge between the two sets …

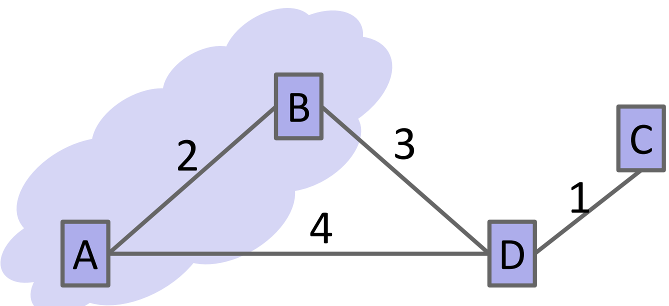
Graph Algorithms We Know

- DFS
  - Point-to-point connectivity verification
- BFS
  - All-pairs shortest paths in an unweighted graph
- Dijkstra's
  - All-pairs shortest paths in a weighted graph
- A* Search
  - Point-to-point shortest path in a weighted graph

Demos: https://qiao.github.io/PathFinding.js/visual/

Dijkstra's Review

- Dijkstra's grows the set of "vertices for which we know the shortest path from s"
- Dijkstra's visits vertices in order of distance from the source
- Dijkstra's relaxes an edge based on its distance from the source




```
dijkstras(Node s, Graph g) {
  PriorityQueue unvisited;
  unvisited.addAll(g.allNodes(), ∞)
  unvisited.changePriority(s, 0);

  Map<Node, Integer> distances;
  Map<Node, Node> previousNode

  while (! unvisited.isEmpty()) {
    Node n = unvisited.removeMin();
    for (Node i : n.neighbors) {
      if (distances[i]
        < distances[n]
          + g.edgeWeight(n, i)) {
        continue;
      } else {
        distances[i] = distances[n]
          + g.edgeWeight(n, i);
        unvisited.changePriority(i,
          distances[i]);
        previousNode[i] = n;
      }}}}
```

Adapting Dijkstra's Algorithm

- MSTs don't have a "source vertex"
  - Replace "vertices for which we know the shortest path from s" with "vertices in the MST-under-construction"
  - Visit vertices in order of *distance from MST-under-construction*
  - Relax an edge based on its *distance from source*

- Note:
  - Prim's algorithm was developed in 1930 by Votěch Jarník, then independently rediscovered by Robert Prim in 1957 and Dijkstra in 1959. It's sometimes called Jarník's, Prim-Jarník, or DJP

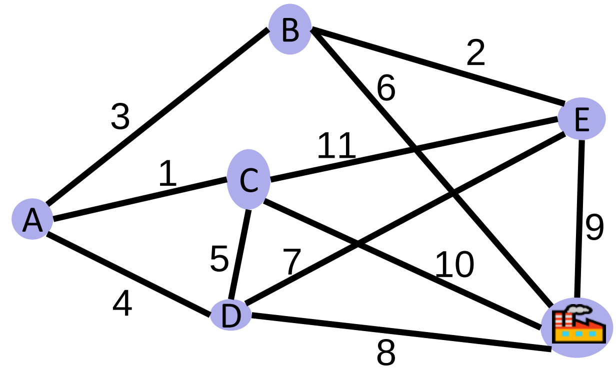
Prim's Algorithm

```
prims(Node s, Graph g) {
  PriorityQueue unvisited;
  unvisited.addAll(g.allNodes(), ∞)
  unvisited.changePriority(s, 0);

  Map<Node, Integer> distances;
  Map<Node, Node> previousNode

  while (! unvisited.isEmpty()) {
    Node n = unvisited.removeMin();
    for (Node i : n.neighbors) {
      if (distances[i]
        < g.edgeWeight(n, i)) {

        continue;
      } else {
        distances[i] =
          g.edgeWeight(n, i);
        unvisited.changePriority(i,
          g.edgeWeight(n, i));
        previousNode[i] = n;
      }}}}
```

```
dijkstras(Node s, Graph g) {
  PriorityQueue unvisited;
  unvisited.addAll(g.allNodes(), ∞)
  unvisited.changePriority(s, 0);

  Map<Node, Integer> distances;
  Map<Node, Node> previousNode

  while (! unvisited.isEmpty()) {
    Node n = unvisited.removeMin();
    for (Node i : n.neighbors) {
      if (distances[i]
        < distances[n]
          + g.edgeWeight(n, i)) {
        continue;
      } else {
        distances[i] = distances[n]
          + g.edgeWeight(n, i);
        unvisited.changePriority(i,
          distances[i]);
        previousNode[i] = n;
      }}}}
```

Your Turn

```
prims(Node s, Graph g) {
  PriorityQueue unvisited;
  unvisited.addAll(g.allNodes(), ∞)
  unvisited.changePriority(s, 0);

  Map<Node, Integer> distances;
  Map<Node, Node> previousNode

  while (! unvisited.isEmpty()) {
    Node n = unvisited.removeMin();
    for (Node i : n.neighbors) {
      if (distances[i]
        < g.edgeWeight(n, i)) {
        continue;
      } else {
        distances[i] =
          g.edgeWeight(n, i);
        unvisited.changePriority(i,
          distances[i]);
        previousNode[i] = n;
      }}}}
```



Node	distances	previous Node
A		
B		
C		
D		
E		
F		

Prim's Demos and Visualizations

- Dijkstra's Visualization
  - https://www.youtube.com/watch?v=1oiQ0hrVwJk
  - Dijkstra's proceeds radially from its source, because it chooses edges by *path length from source*
- Prim's Visualization
  - https://www.youtube.com/watch?v=6uq0cQZOyoY
  - Prim's jumps around the graph (the fringe), because it chooses edges by *edge weight* (there's no source)
- Demo: https://docs.google.com/presentation/d/1GPizbySYMsUhnXSXKvbqV4UhPCvrt750MiqPPgU-eCY/present?ueb=true&slide=id.g9a60b2f52_0_205

Prim's Algorithm: Runtime

- Assuming a binary heap implementation

	# Operations	Cost per operation	Total Cost
PQ add	V	O(log V)	O(V log V)
PQ removeMin	V	O(log V)	O(V log V)
PQ changePriority	E	O(log V)	O(E log V)

- Runtime: O(V log V + V log V + E log V) = O(V log V + E log V)

Lecture Outline

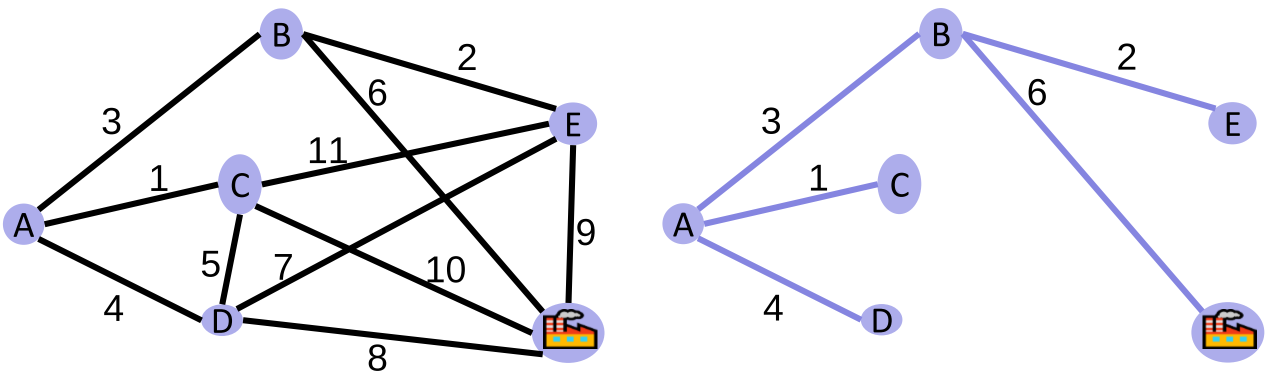
- Minimum Spanning Trees
- Prim’s Algorithm
- **Kruskal’s Algorithm**
- Applications of MSTs

A Different Approach

- Prim's thinks vertex by vertex
  - Eg, add the closest vertex to the currently reachable set
- What if you think edge by edge instead?
  - Eg, start from the lightest edge; add it if it connects new things to each other (don't add it if it would create a cycle)

Your Turn

- Can you find an MST in this graph by considering edges in sorted order?



Kruskal's Algorithm

- Visualization:
  https://www.youtube.com/watch?v=ggLyKfBTABo

- Conceptual demo:
  https://docs.google.com/presentation/d/1RhRSYs9Jbc335P24p7vR-6PLXZUl-1EmeDtqieL9ad8/present?ueb=true&slide=id.g375bbf9ace_0_645

```
kruskals(Graph g) {
  msts = {}
  for (n in g.allNodes()) {
    msts.add(makeMST(n));
  }

  finalMST = {};
  for ((u, v) in sort(g.allEdges()) {
    uMST = msts.find(u);
    vMST = msts.find(v);
    if (uMST != vMST) {
      finalMST.add(u, v);
      msts.union(uMST, vMST);
}}}
```

Kruskal’s Algorithm: Runtime

- Assuming an unknown data structure for “msts”, Kruskal’s runtime looks like:

	# Operations	Cost per operation	Total Cost
add	V	?	
find	2E	?	
union	V-1	?	

- Runtime: O(V * union + E * find)

Lecture Outline

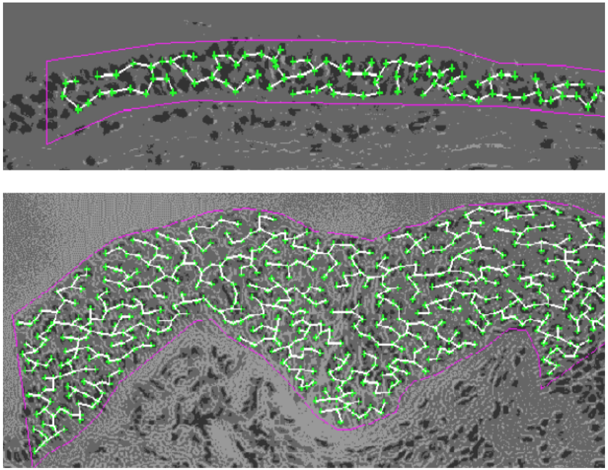
- Minimum Spanning Trees
- Prim's Algorithm
- Kruskal's Algorithm
- **Applications of MSTs**

Applications of MSTs

- Handwriting recognition
  - http://dspace.mit.edu/bitstream/handle/1721.1/16727/43551593-MIT.pdf;sequence=2

Figure 4-3: A typical minimum spanning tree

- Medical imaging
  - e.g. arrangement of nuclei in cancer cells



For more, see: http://www.ics.uci.edu/~eppstein/gina/mst.html

tl;dr

- Minimum Spanning Trees are a subgraph that “covers” all the vertices but not all the edges
  - Lots of cool applications!
- Two algorithms for finding MSTs:
  - Prim’s and Kruskal’s
  - Prim’s is reasonably fast greedy algorithm that looks like Dijkstra’s
  - Same with Kruskal’s, but we need another data structure before we can complete it