Week 5 - Wednesday

COMP 4500

Last time

- What did we talk about last time?
- Exam 1!

- Before that:
  - Review

- Before that:
  - Scheduling to minimize lateness
  - Dijkstra's algorithm

Questions?

Assignment 3

Logical warmup

- An anthropologist studying on the Island of Knights and Knaves is told that an astrologer and a sorcerer are waiting in a tower
- When he goes up into the tower, he sees two men in conical hats
- One hat is blue and the other is green
- The anthropologist cannot determine which man is which by sight, but he needs to find the sorcerer
- He asks, "Is the sorcerer a Knight?"
- The man in the blue hat answers, and the anthropologist is able to deduce which one is which
- Which one is the sorcerer?

Minimum Spanning Trees

Minimum spanning tree

- We have a weighted, connected graph and we want to remove as many edges as possible such that:
  - The graph remains connected
  - The edges we keep have the smallest total weight
- This is the **minimum spanning tree** (MST) problem
- We can imagine pruning down a communication network so that it's still connected but only with the cheapest amount of wire total
- MST algorithms are also used as subroutines in other graph problems

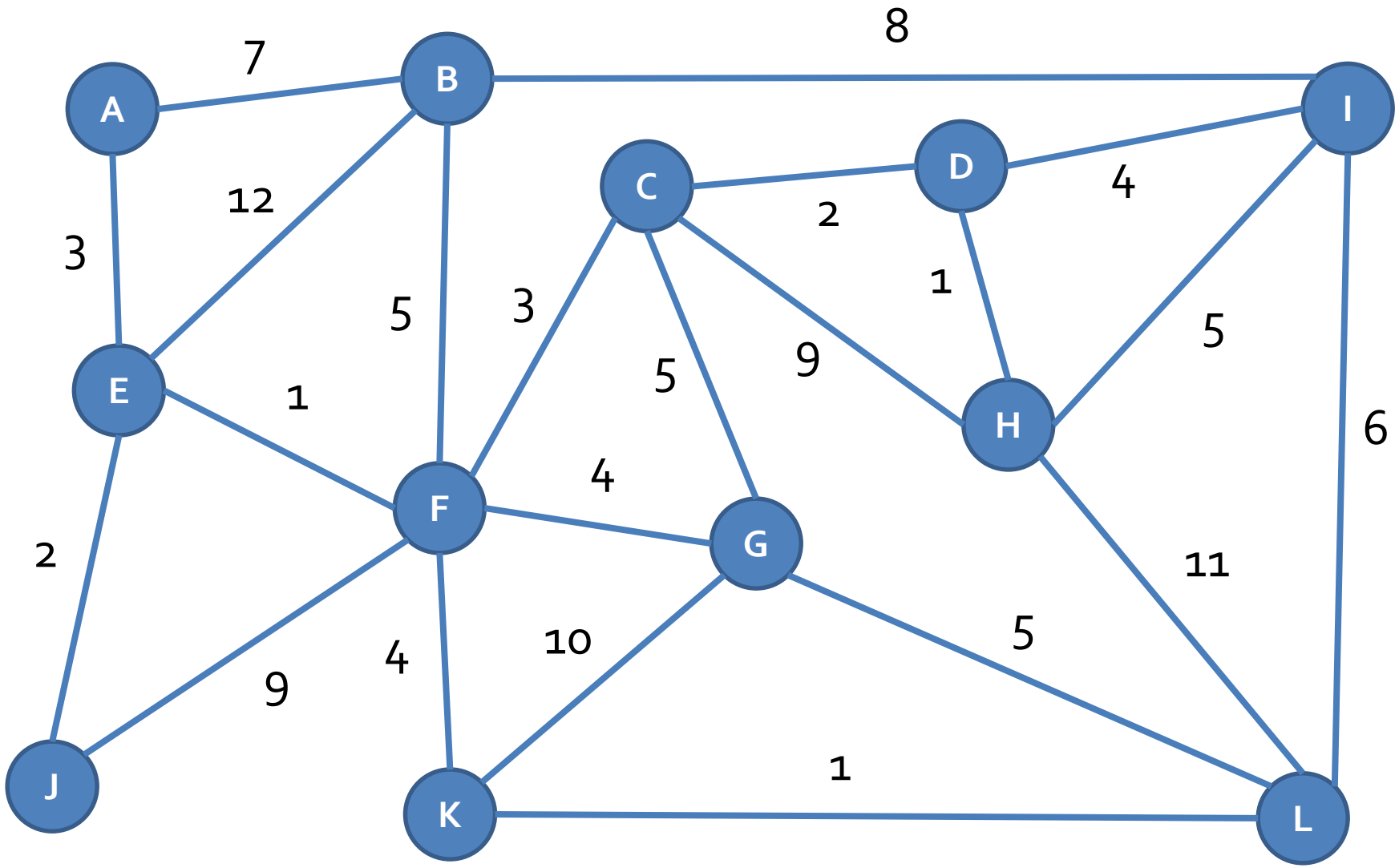
MST observations

- Assuming positive edge weights, the resulting graph is obviously a tree
  - If the graph wasn't connected, it wouldn't be a solution to our problem
  - If there was a cycle, we could remove an edge, make it cheaper, and still have connectivity

Approaches

- **Kruskal's algorithm:** Add edges to the MST in order of increasing cost unless it causes a cycle
- **Prim's algorithm:** Grow outward from a node, always adding the cheapest edge to a node that is not yet in the MST
- **Backwards Kruskal's algorithm:** Remove edges from the original graph in order of decreasing cost unless it disconnects the graph
- All three algorithms work!

MST example



Cut Property

- Assume all edge weights are distinct.
- Let ***S*** be a subset of nodes that is neither empty nor equal to ***V***.
- Let edge ***e*** = (***v***,***w***) be the minimum-cost edge with one end in ***S*** and the other in ***V*** – ***S***.
- Every minimum spanning tree contains ***e***.

Proof of Cut Property

- Let T be a spanning tree that does not contain e. We will try to find an edge e' in T that is more expensive than e that we can swap with e to make a cheaper spanning tree.
- The ends of e are v and w. Since T is a spanning tree, there must be a path P in T from v to w. Following P, we will eventually reach a node w' that is in $V - S$.
- Let $v' \in S$ be the node just before w' on P.
- Let $e' = (v', w')$ be the edge between v' and w'.

Proof continued

- If we exchange e for e', we get edges $T' = (T - \{e'\}) \cup \{e\}$.
- T' is connected since T was connected and any path that used to cross (v', w') can follow the part of P from v' to v, the edge e, and then the part of P from w to w'.
- T' is acyclic since the only cycle in $T' \cup \{e'\}$ is the one made up of e and path P, but it's gone since e' was deleted.
- Both e and e' have one end in S and the other in $V - S$, but e is the cheapest edge with this property, so its weight is lower.
- Thus, T' has lower cost than any spanning tree T that does not include e. ■

Kruskal's algorithm produces an MST

- **Proof:** Whenever we add an edge $e = (v, w)$, let S be the set of nodes that v has a path to before adding e. Node $v \in S$. But $w \notin S$, because e would otherwise create a cycle. Since e is the cheapest edge with one end in S and the other in $V - S$, the Cut Property says it must be part of every minimum spanning tree.
- Thus, Kruskal's algorithm adds exactly those edges that must be part of every minimum spanning tree. ■

Cycle Property

- Assume that all edge costs are distinct. Let ***C*** be any cycle in ***G***, and let edge ***e*** = (***v***,***w***) be the most expensive edge in ***C***. Then ***e*** does not belong to any minimum spanning tree of ***G***.

Proof of Cycle Property

- Let ***T*** be a spanning tree that contains ***e***. We can show that it doesn't have minimum cost.
- If we delete ***e*** from ***T***, it partitions nodes into two components, ***S***, containing ***v***, and ***V*** – ***S***, containing ***w***.
- The edges of cycle ***C***, with ***e*** removed, form a path ***P*** from ***v*** to ***w***. There must be some edge ***e'*** on ***P*** that crosses from ***S*** to ***V*** – ***S***.

Proof continued

- Consider the set of edges $T' = (T - \{e\}) \cup \{e'\}$.
- T' must be connected and have no cycles; thus, T' is a spanning tree.
- Since e is the most expensive edge in C, e' is cheaper, and T' is cheaper than T. ■

MST reflections

- Using the Cut Property, it's easy to show the correctness of Prim's algorithm
- Using the Cycle Property, it's easy to show the correctness of the Reverse Kruskal's algorithm
- It turns out that **any** algorithm that follows the Cut Property to add edges to a spanning tree or any algorithm that follows the Cycle Property to remove edges from a graph (or any combination of the two) will find an MST

What about when some edges have the same cost?

- In all MST algorithms, if there is a choice between edges with the same cost, either can be chosen
  - Provided that connectivity/cycle constraints are met
- A way to demonstrate this is to add tiny random amounts to the weights of all edges, much smaller than the difference between any non-equal cost edges
- These random changes serve as tie-breakers between edges of the same cost
  - However, they will not change the structure so that larger edges would have been chosen

Clustering

Clustering

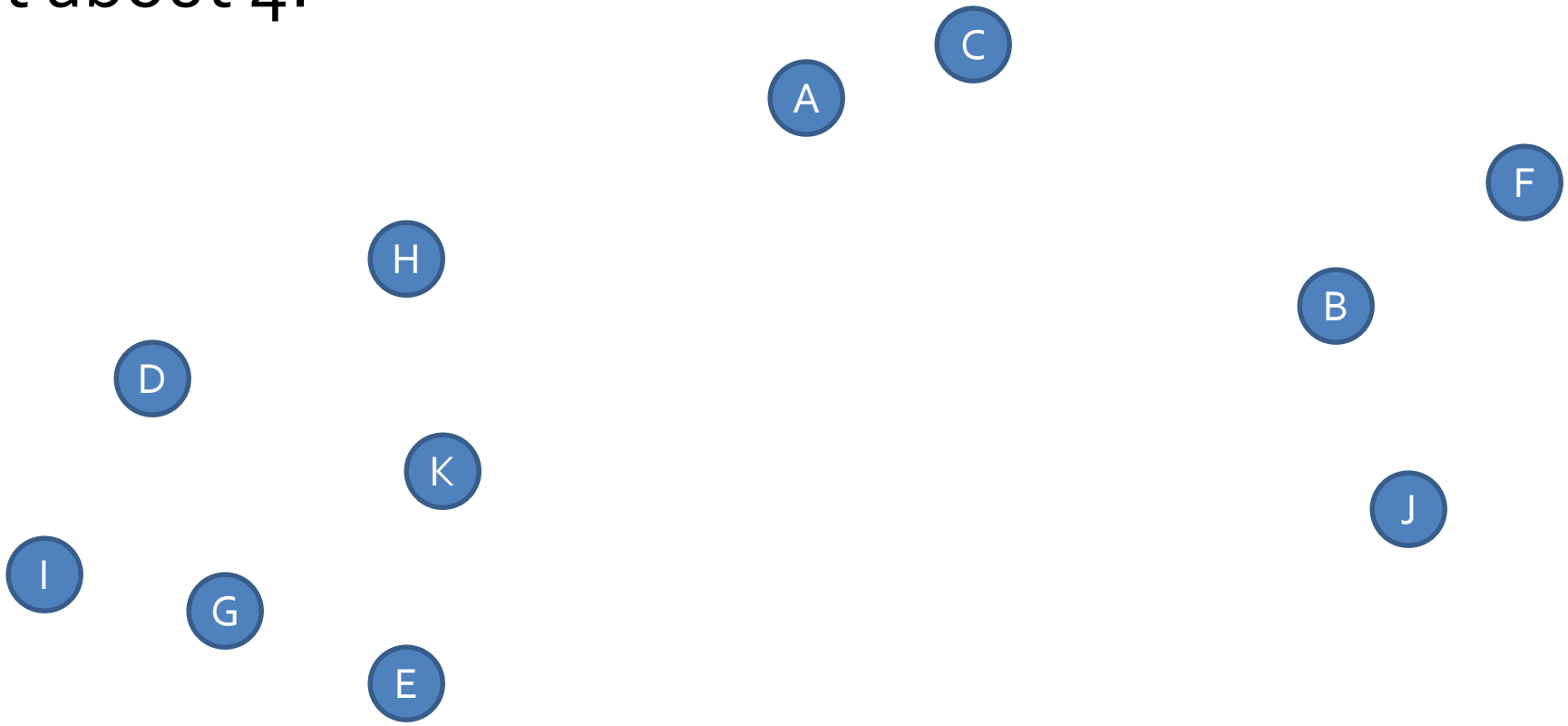
- Imagine you have a set of objects
  - Photographs
  - Documents
  - Microorganisms
- You want to classify them into related groups
- Usually, you have some **distance function** that says how far away any two objects are
- You want to group together objects so that all the objects in a group are close

Notes about distance

- The distance function is usually defined between all points
  - If the points are in the plane or another Euclidean space, the distance could simply be the distance between them
  - A more flexible way to define distance is as weights on graph edges in a complete graph
- The distance between a point and itself is o
- The distance between any two distinct points is greater than o
- The distance between two points is symmetrical

Clustering

- What if you wanted to cluster these points into three clusters?
- What about 4?



Clustering by maximum spacing

- What if we want to divide our objects into ***k*** non-empty sets:
  - $C_1, C_2, \ldots, C_k$
- The **spacing** of this ***k***-clustering is the minimum distance between any pair of points in different clusters
- We want to find clusters with maximum spacing
  - There are other metrics to optimize your clusters on

Algorithm

- We don't want to group together objects that are far apart
- We sort all of the edges by weight and begin adding them back to our graph in order
- If an edge connects nodes that are already in the same cluster, we skip it
  - Thus, we don't make cycles
- We stop when we have ***k*** connected components

MST saves the day

- This algorithm is exactly Kruskal's algorithm
  - Add edges by increasing size, skipping ones that make a cycle
- We simply stop when we have ***k*** connected components instead of connecting everything
  - Alternatively, you can make the MST and delete the ***k*** – 1 most expensive edges

We get a *k*-clustering of maximum spacing

- Proof:
  - Let our clustering be sets $C_1, C_2, \ldots, C_k$. The spacing of this clustering is d^*, the $(k-1)^{st}$ most expensive edge in the MST, the edge that Kruskal's algorithm would have added next.
  - Consider some other clustering $C'_1, C'_2, \ldots, C'_k$ that is not the same. One of our sets C_r must not be a subset of any set in $C'_1, C'_2, \ldots, C'_k$. Thus, there must be points p_i and p_j in C_r that are in different sets in the other clustering. Let $p_i \in C'_s$ and $p_j \in C'_t \neq C'_s$.

Proof continued

- Since p_i and p_j belong to the same component C_r, Kruskal's algorithm added all the edges in a p_i-p_j path P before it stopped.
- Thus, every edge on P is d^* or smaller.
- Let p' be the first node on P that does not belong to C'_s and let p be the node on P that comes just before p'. We know that $d(p,p') \leq d^*$. But p and p' are different sets in the clustering C'_1, C'_2,…, C'_k, so that clustering must have spacing at most d^*.
- Since any other clustering must have spacing at most d^*, the clustering C_1, C_2,…, C_k has maximum spacing. ■

Upcoming

Next time…

- Data compression

Reminders

- Start on Assignment 3
- Read section 4.8
- Extra credit opportunities (0.5% each):
  - Rublein research talk: 2/9 12:30-1:30 p.m. in Point 140
  - Rublein teaching demo: 2/9 3-4 p.m. in Point 140
  - Phadke research talk: 2/12 3-4 p.m. in Point 139
  - Phadke teaching demo: 2/13 10-10:55 a.m. in Towers 112
  - Hristov teaching demo: 2/19 11:30-12:25 a.m. in Point 113
  - Hristov research talk: 2/19 4:30-5:30 p.m. in Point 139