Scientific Computing

Wed, Jan. 28

Announcements

* Homework 1 due Friday night, 11:59pm.

Office Hours:

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Idea #4: best = "earliest ending time"

This works on all our previous examples.
Can we break it?

Intuition: Picking the one that ends earliest gets you credit for a meeting that gets out of the room as quickly as possible.

Algorithm:

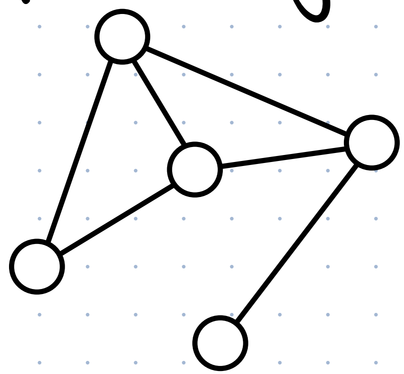
Let R be the set of requests.
Let A be the empty set.
While R is non-empty:
- Find the request with earliest end time.
- Add it to A.
- Remove it from R and remove all other requests that are not compatible.

A is the solution

* Coding the greedy algorithm!
    * Python lesson on functions and sort keys
    * Demo

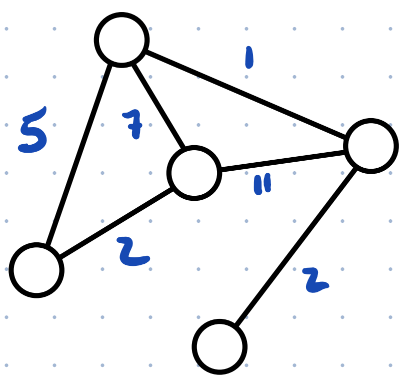
Problem #2: Minimum Spanning Tree

A graph is a set of vertices or nodes, connected in pairs by edges.

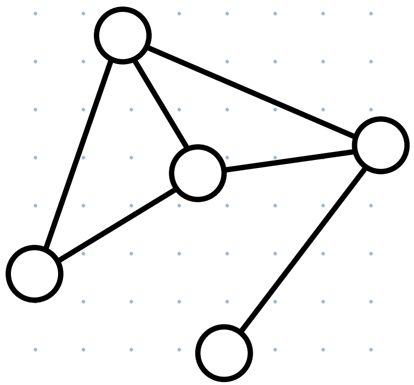


5 vertices
6 edges

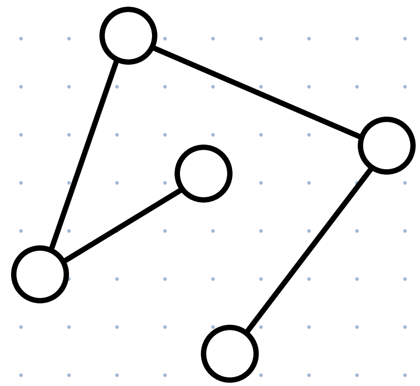
A weighted graph is a graph whose edges have real #s as "weights".



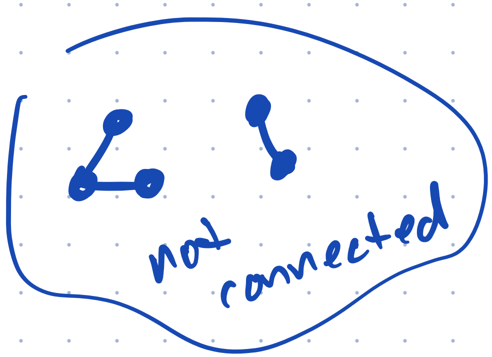
A **tree** is a graph that is connected (can reach every vertex from every vertex) and has no cycles.



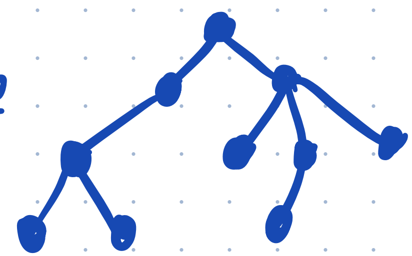
lots of cycles



no cycles = tree

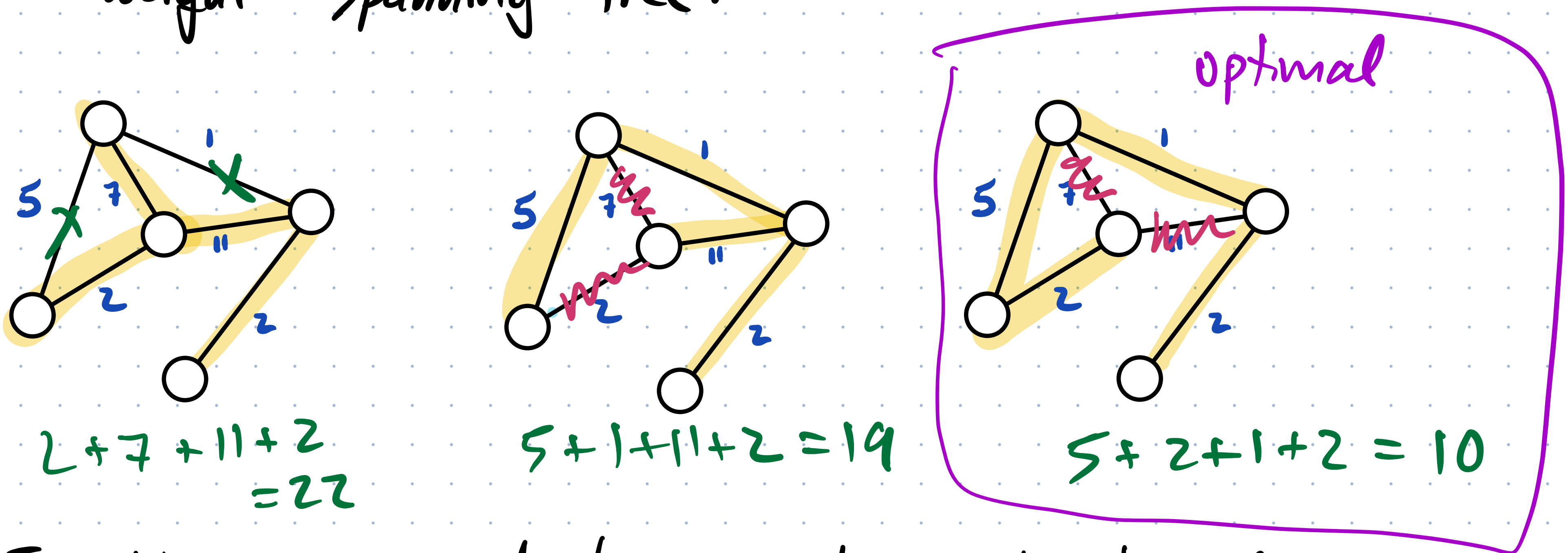


tree



Minimum Spanning Tree Problem:

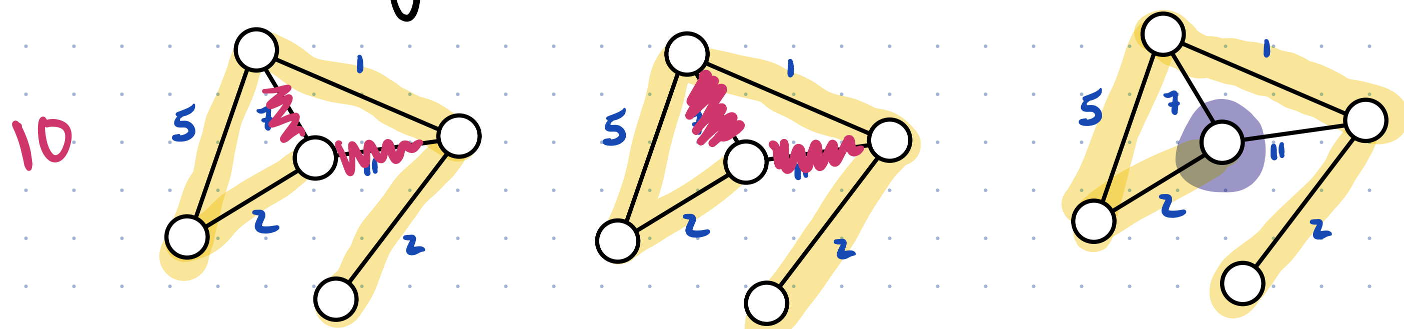
Given a weighted graph G, find the subset of edges that forms a minimum-weight spanning tree.



<u>Ex</u>: You might need to connect a bunch of buildings with cables, and the weight of the edges is the cost of the connection.

Possible Greedy Algorithms:

* pick the cheapest edge that doesn't make a cycle (starting with no edges, adding one at a time)
* start with all edges, and delete the most expensive one as long as it doesn't disconnect the graph
* pick one node as the start, and repeatedly choose the cheapest edge (that doesn't make a cycle) that connects to a node you have reached so far



Idea #1 Idea #2 Idea #3

In this example they all generated the same tree, but that doesn't always have to be true.

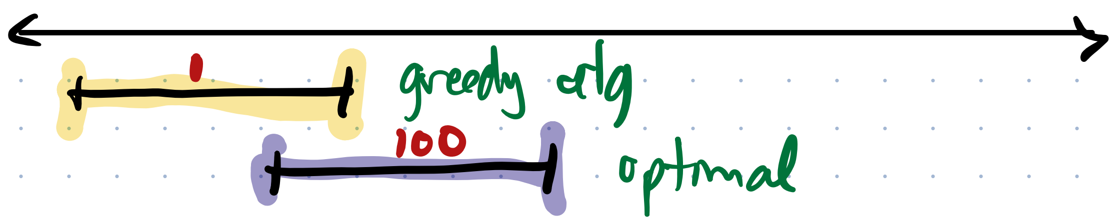
More importantly: are any of these guaranteed to give optimal solutions?

__Theorem__: They all do!
(We won't prove it in class.)

Problem #3: Weighted Interval Scheduling

This is like regular interval scheduling, except each request i comes with a value v_i and your goal is to maximize the total value of satisfied requests.

Our previous greedy algorithm is now pretty bad.



Possible Greedy Algos:

* best = "highest value"
* best = "shortest"
* best = "highest $\frac{\text{value}}{\text{duration}}$" (value density)

Are any of these optimal? No.

The is no known greedy algorithm that is optimal. There are 2^n subsets of a set of size n.

How long would brute force take? If there are n requests, you'd need to check all 2^n subsets of them. So, the run time would be exponential, something like $O(2^n)$ or $O(n \cdot 2^n)$.

This is "big-O" notation, and it tells you roughly how many steps an algorithm has to use.

For this particular problem, there is a technique to do it in $O(n \cdot \log(n))$ time — very fast!

"dynamic programming"