Math 4707: Minimum Spanning Tree + Traveling Salesman Problem

3/8

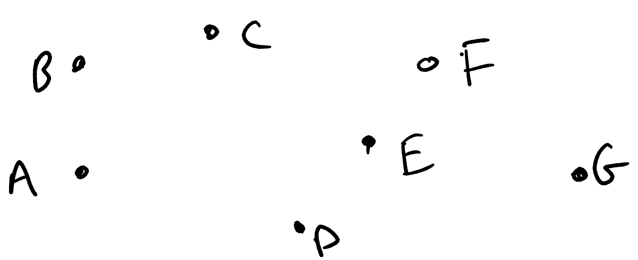
Ch. 9 of LPV

Reminders: • HW #2 has been graded.
• HW #3 is due this Wed., 3/10

Today we will start discussing optimization problems, which are about finding "the best" of a certain discrete structure. As we'll see, many of these optimization problems take as input a graph, maybe w/ some extra data.

Minimum Spanning Tree

The 1st optimization problem we'll consider is the Minimum Spanning Tree problem. Imagine you are planning a railroad network. You have several cities you'd like to link up:

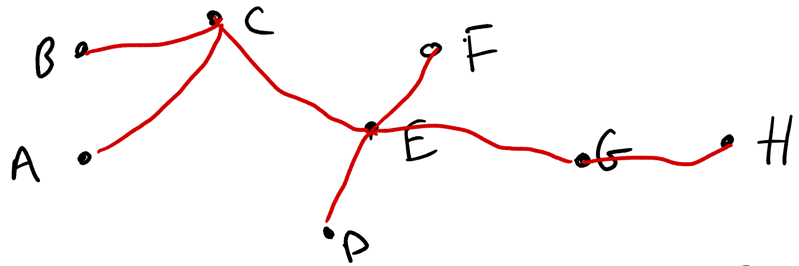


You know the cost to join any pair of cities:

e.g. A — B = \$1 million

C — D = \$3 million . . .

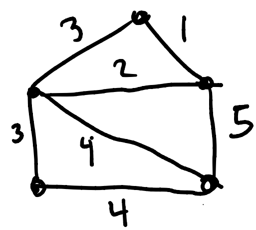
You want to figure out what's the cheapest network you can build, subject to the requirement that the network connects all the cities. We know what minimally connected graphs are: trees!



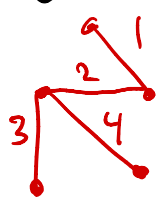
But how to find the cheapest tree? We could just check all trees. But remember there are n^{n-2} trees for n cities . . . would take a long time to consider all.

We can do something smarter . . . .

Formally, the input to the Minimum Spanning Tree (MST) problem is a connected graph $G=(V,E)$, together with a cost function $c: E \to \mathbb{R}_{\geq 0}$ that assigns to each edge a nonnegative real "cost":



And the output should be a spanning tree T with the lowest total cost (i.e., sum of costs of edges) among all spanning trees:



How can we quickly find such a MST?

Idea: Greedy Algorithm! Let's try to be "optimistic" and just keep adding the cheapest edges we can until we've formed a spanning tree of G.

Of course, we don't want to add any old edge at each step of the algorithm, because then we might form a cycle, which our tree cannot have.

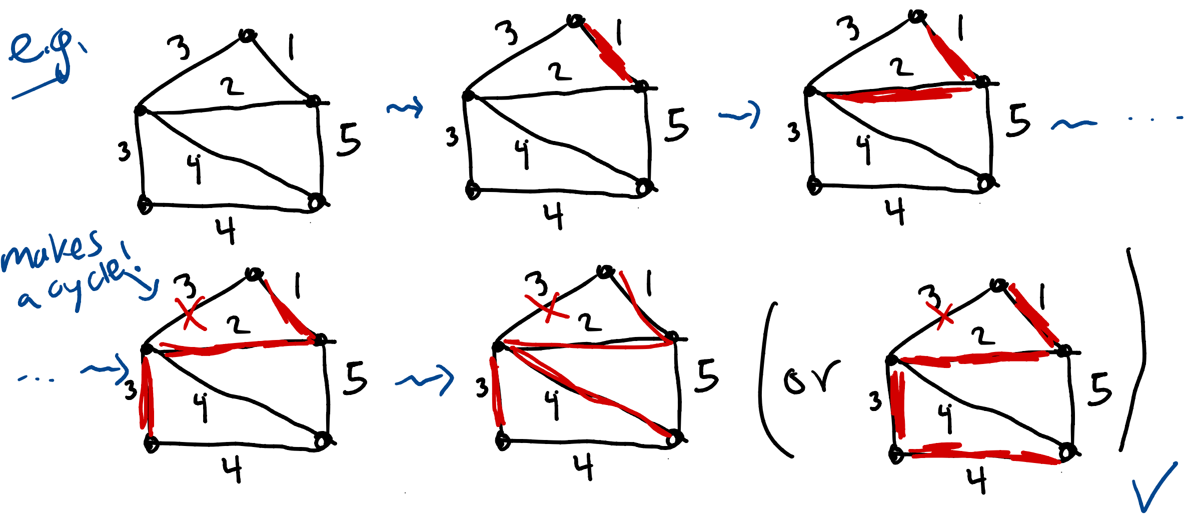
So the algorithm is:

1. Among all edges we have not yet selected, and which don't form a cycle w/ edges we've selected, select the/a cheapest one.

2. If the edges we've selected don't yet form a spanning tree, repeat step # 1.

3. Otherwise, return that spanning tree.

This greedy alg. for MST is called Kruskal's alg.

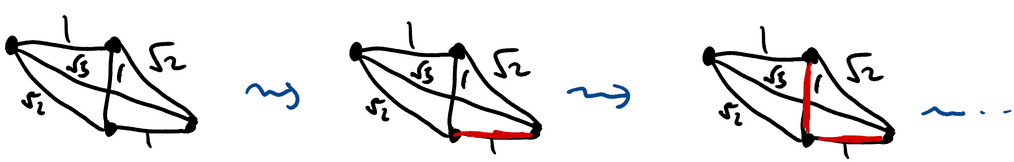


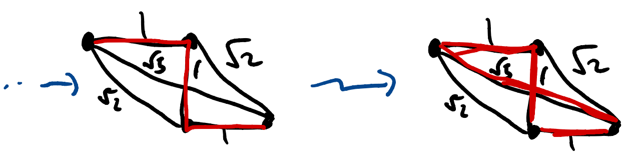
It's pretty clear that because we never form a cycle, but keep adding edges, eventually we will terminate w/ a spanning tree.

But why does Kruskal produce a MST?

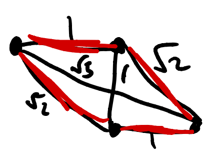
First let's show that a greedy algorithm doesn't always work to solve any problem. Consider the problem of finding a minimum cost Hamiltonian cycle in a graph. And suppose we tried a greedy alg. which repeatedly adds the cheapest edge that could extend our set towards a Ham. cycle.

e.g.





BUT



$1+1+1+\sqrt{5} > 1+\sqrt{2}+1+\sqrt{2}$

our greediness led to bad choices!

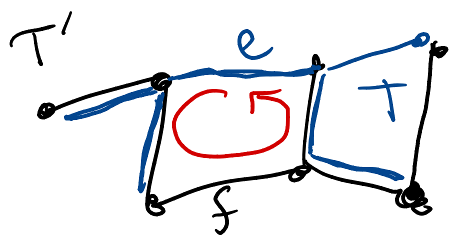
So for Kruskal's alg. succeed at finding a MST, have to use something about trees.

pf that Kruskal gives a MST:

Let T be the spanning tree of G produced by Kruskal. Let T' be any other spanning tree of G. We want to show the cost of T' is at least as much as of T.

Let e be the first edge of T we add during Kruskal's alg. that is not in T'. If we add e to T', we must get a cycle:

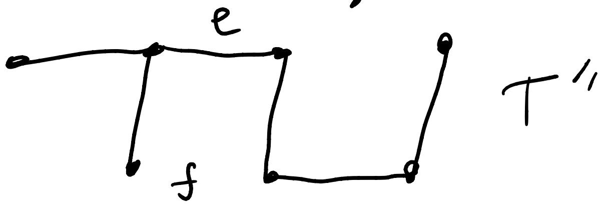
why?



Since T has no cycles, there must be some edge f in that cycle that's in T' but not in T. We claim f costs at least as much as edge e. Why?

If f were cheaper than e, we would've added it at step of Kruskal we added e.
(Why does f not form a cycle w/ any of the earlier edges in T?)

So by replacing f w/ e in T', we get a new spanning tree T'':



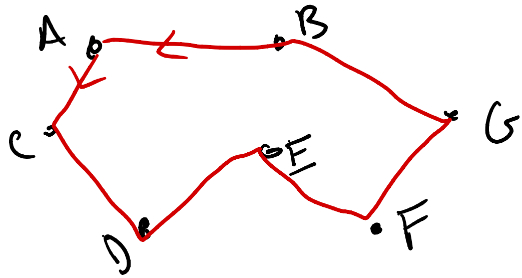
T'' is at least as cheap as T', and has strictly more edges in common w/ T.
So repeating this process will show that T is at least as cheap as T'.

Thus, T is indeed a MST. ▨

Rmk: There are many variants of Kruskal's alg. See the book...

Traveling Salesman Problem

Imagine you are a salesman and you want to visit all the towns in your region, and come back to your hometown at the end, in order to make your sales pitch:



To save money, you want to find the shortest cycle connecting all the towns. This is the Traveling Salesman Problem (TSP), and we can see that it's the same as finding the minimum Hamiltonian cycle in a complete graph K_n w/ edge costs $c: E \to \mathbb{R}_{\geq 0}$.

We already saw that greedy alg. doesn't work here, and in fact it is believed that no TSP alg. is substantially faster than brute force checking all $(n-1)!$ Ham. cycles.

But let's suppose our costs satisfy the triangle inequality

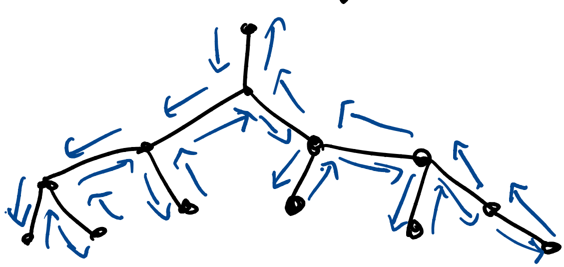
$$c(A-B) + c(B-C) \geq c(A-C).$$

For example, this is true if cost = distance:

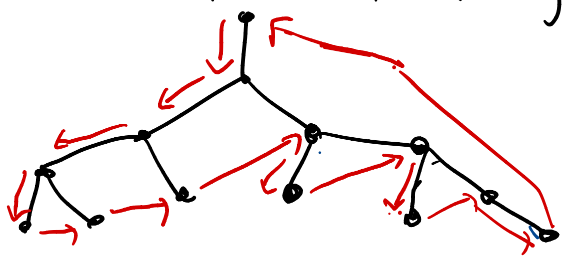
Then we can use MST to quickly find an approximate solution to TSP that's at most double cost of optimal.

The idea is to use MST traversal w/ shortcuts. We start by finding the MST of our graph, using Kruskal:

T (MST)



Then we <u>traverse</u> this MST as depicted, crossing each edge twice. This is a circuit that visits each vertex at least once, but it will visit some vertices many times. So modify it using shortcuts where we skip to the next vertex we haven't visited yet:



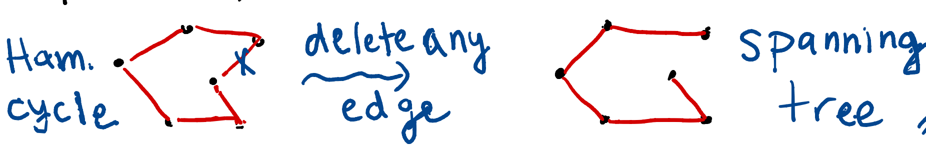
This will be a Ham. cycle, and the triangle inequality guarantees its cost is $\leq$ cost of traversal w/out shorcuts.

So

$$\begin{aligned}\text{cost}\left(\begin{matrix}\text{MST traversal}\\ \text{w/ shortcuts}\end{matrix}\right) &\leq \text{cost}(\text{MST traversal})\\ &= 2\cdot\text{cost}(\text{MST})\\ &\leq 2\cdot\text{cost}(\text{min. Ham. cycle}),\end{aligned}$$

where cost(MST) $\leq$ cost (min Ham. cycle)

since



//

Now let's take a 5 min break

and when we come back

do a worksheet

on MST + TSP

in breakout groups.