Greedy Spanning Trees

CSE 180
Algorithmic Thinking

But First! A Change Making Caveat

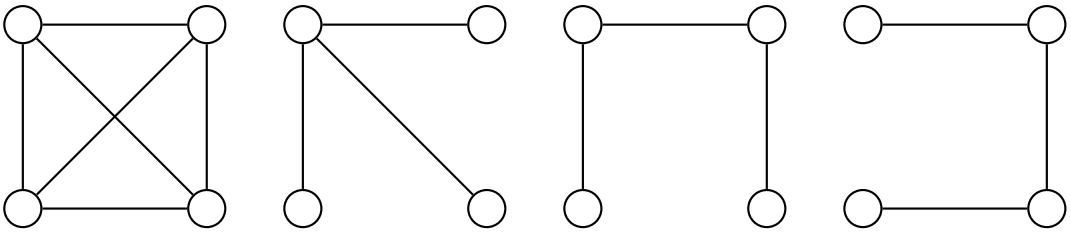
With certain values for the coins, and provided an adequate supply of each denomination, the straightforward greedy algorithm always produces an optimal solution. But this is not always true for some coinage systems, e.g., Old English coinage:

- half-crowns (30 pence)
- florins (24 pence)
- shillings (12 pence)
- sixpence (6 pence)
- threepence (3 pence)
- pennies (1 pence)

Find a small amount to make change (between 10 and 99 cents) where the greedy algorithm for this coinage system does **not** produce an optimal solution. Hint: The optimal solution will need only two coins, whereas three will be the result of the greedy algorithm.

Now, what is a Spanning Tree?

A spanning tree is a tree containing all the nodes of a given connected graph, but only some of the links. The graph below on the left has several spanning trees, three of which are shown to its right:



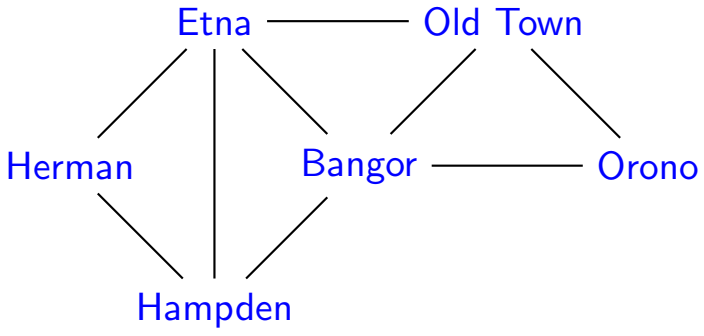
Where does greed come into play? Finding a spanning tree of minimum total weight, known as a **minimum spanning tree**.

Practical Applications

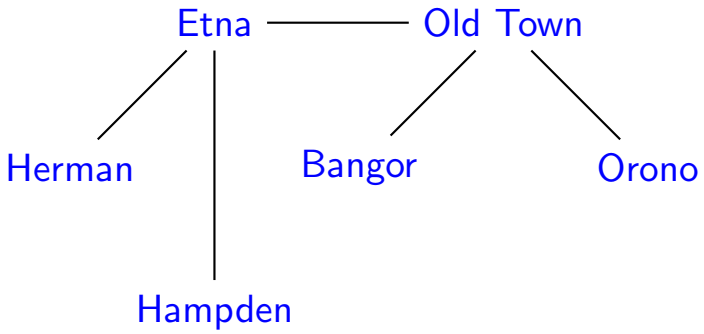
Minimum spanning trees find application in all kinds of "minimal path routing" settings:

- circuits
- networks
- roads
- sewers
- ...

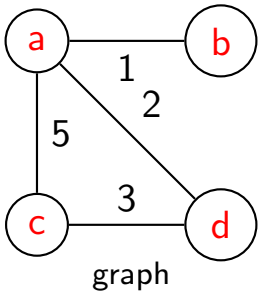
For example, plowing snowy roads in Maine — how do we plow the fewest roads so there will always be cleared roads connecting any two towns?



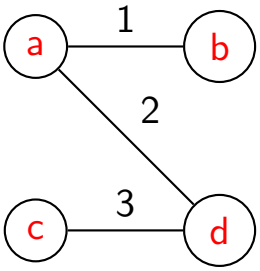
How to Plow



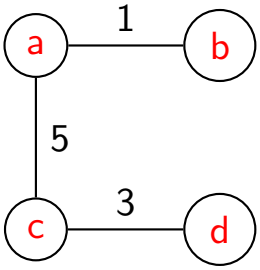
MST Example



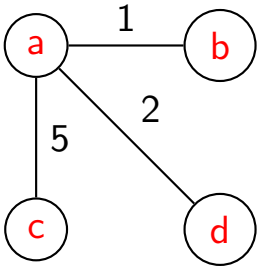
graph



$w(T_1) = 6$



$w(T_2) = 9$



$w(T_3) = 8$

A graph and (all of) its spanning trees; T_1 is the **MST**.

Kruskal's MST Algorithm

Invented by Joseph Kruskal in 1956.

1. Start with an empty tree.
2. Pick the link with the smallest weight that doesn't create a cycle in the tree.
3. Add the link to the tree and remove it from the graph.
4. Continue until all the graph's nodes are part of the tree.

The trick is to find an efficient way of detecting/avoiding cycles!

Practice with Kruskal's

Run Kruskal's algorithm on this graph. Your solution should be a list of links, written as two-letter combinations, where the earlier letter in the alphabet comes first, e.g., **ad**, not **da**. The list should be in the order the algorithm picks them, breaking ties alphabetically. E.g., **ef**, **ad**, ..., but not **ef**, **hi**, ..., because, while **ad** and **hi** both have weight 2, **ad** comes first alphabetically.