Week 9 - Spanning Trees

AD 325 - 2021

Contents

Learning Outcomes

- Spanning trees
- Minimum spanning trees (MST)
- Applications of MST

Reading & Videos

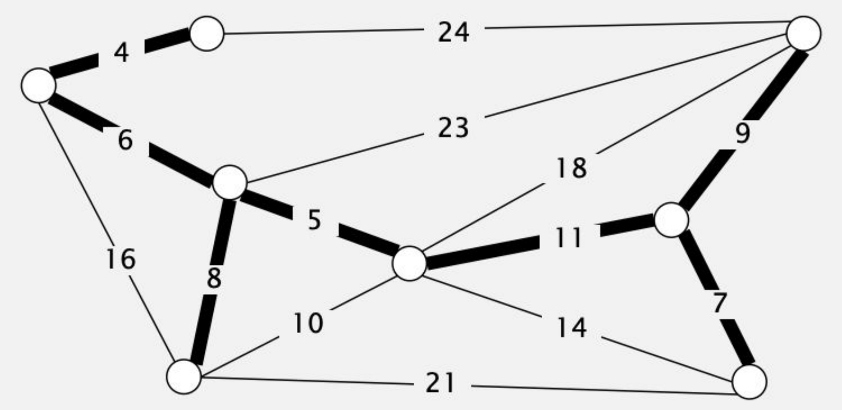
- https://www.coursera.org/learn/algorithms-part2/home/week/2 (MST)

Reference

- https://algs4.cs.princeton.edu/43mst/
- https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#minimumSpanningTree

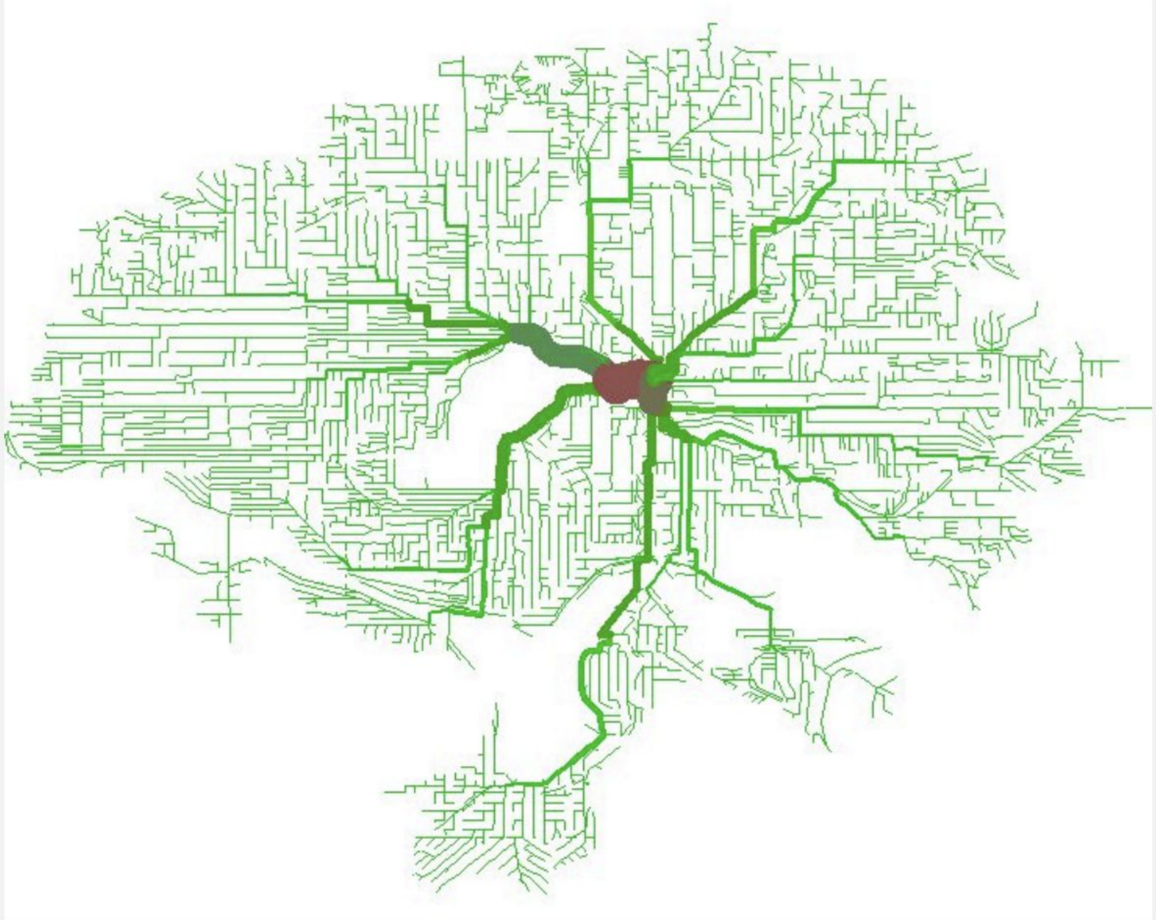
Overview

A **spanning tree** is a **connected**, **acyclical** subgraph that spans all vertices of a graph. Minimum Spanning Trees (MST) are fundamental to a wide range of applications - e.g. clustering, routing.



spanning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

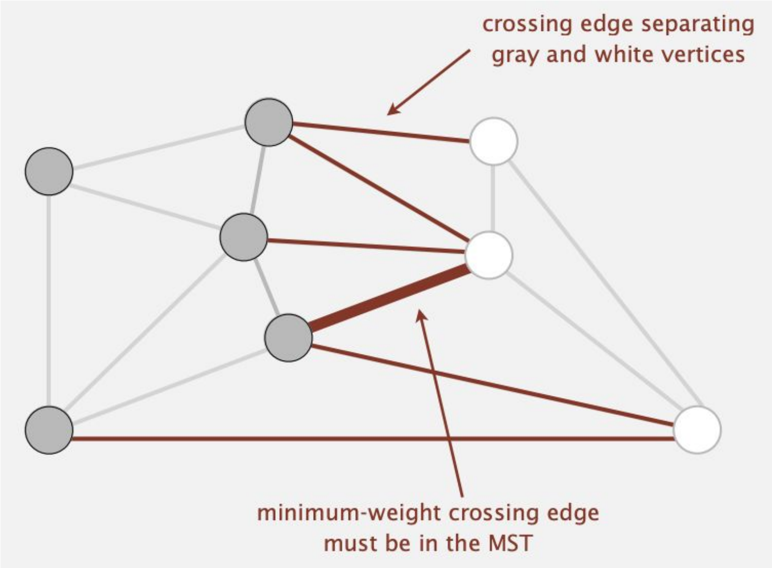
MST of bicycle routes in North Seattle



http://www.flickr.com/photos/ewedistrict/21980840

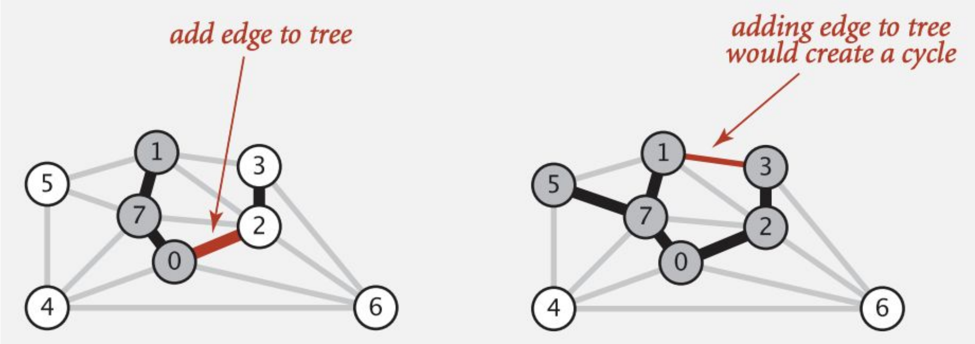
Finding a MST

- **Cut** (partition) the graph vertices into two sets
- Find **crossing edges** that connect any vertex in one set with a vertex in the other set
- Find the **crossing edge** with minimum weight using a **greedy** algorithm
- Connect the target vertex to the origin set
- Repeat until all vertices are connected (e.g. when number of edges is V-1)



Kruskal’s algorithm

- Arrange graph edges in ascending order of weight using a Min Priority Queue
- Maintain a **set** for each **connected component**
- Select next edge with lowest weight and ‘connect’ its vertices if doing so does not create a cycle (edges are not in same component)
- Can spawn multiple connected components that gradually merge
- Useful for identifying clusters



Prim's algorithm

- Start with vertex 0 and grow tree greedily
- Add the min weight edge with only one endpoint in the tree
- Keep track of visited and disallowed edges
- Repeat until tree has V-1 edges
- Array implementation is optimal for dense graphs
- Binary heap is much faster for sparse graphs