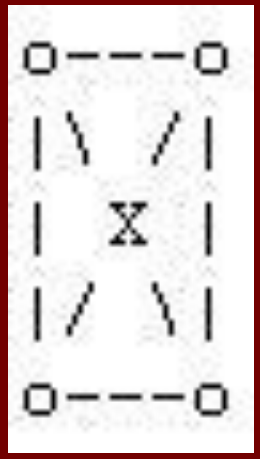
Lecture # 15

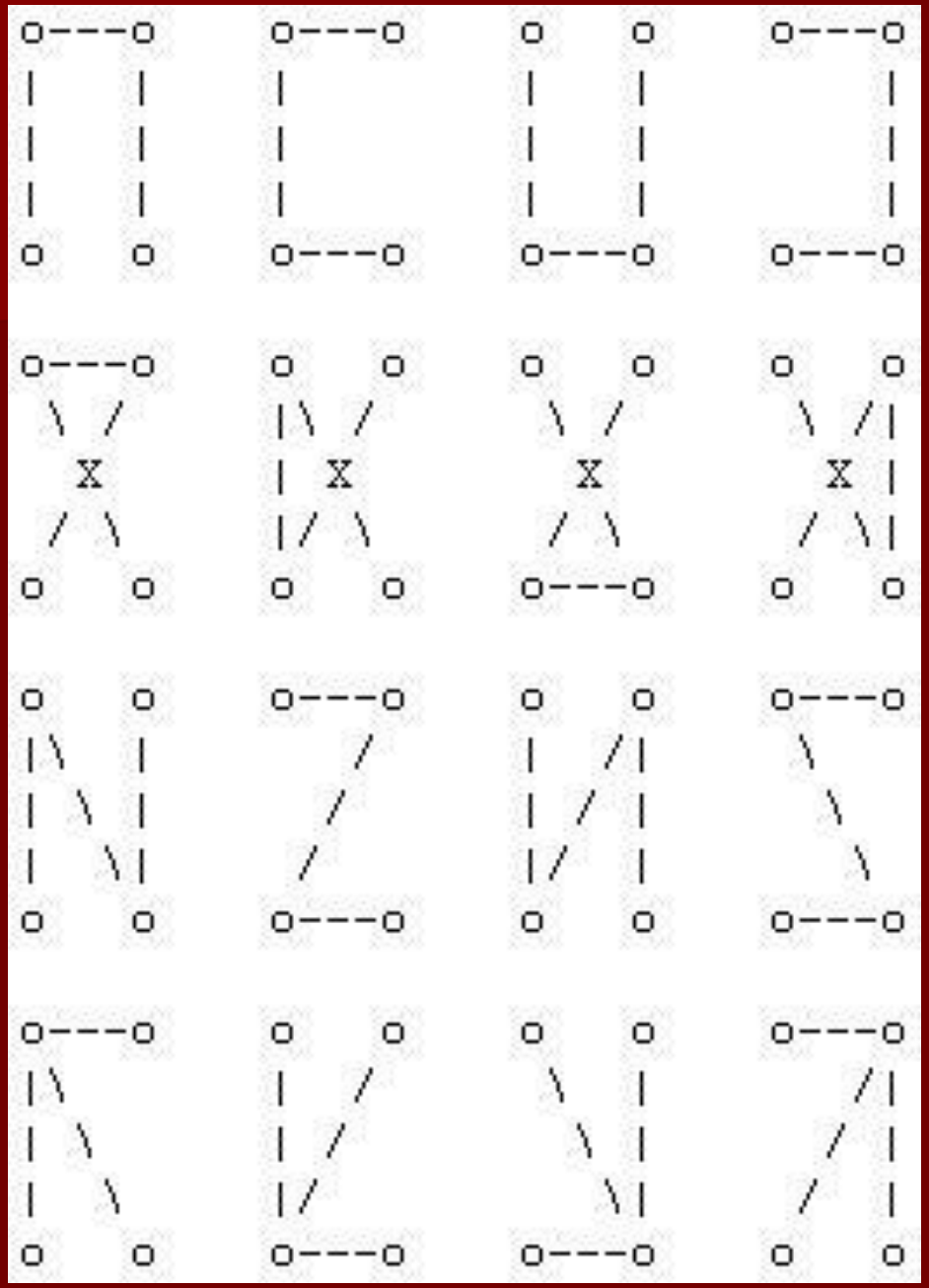
- Minimum Spanning Tree
- Kruskal’s Algorithm
- Prim's algorithm

Spanning Trees

- A spanning tree of a graph is a tree that contains all the vertices of the graph.
- A graph may have many spanning trees; for instance the following complete graph on four vertices has sixteen spanning trees as follows:



All possible 16 Spanning Trees



Minimum Spanning Tree (MST)

- Informally, the MST problem is to find a tree T of a given graph G that contains all the vertices of G and has the minimum total weight of the edges of G over all other such trees.

How to find minimum spanning tree?

- The stupid method is to list all spanning trees, and find minimum of list. We already know how to find minima.
- It's also not really an algorithm, because you'd still need to know how to list all the trees.
- A better idea is to find some key property of the MST that lets us be sure that some edge is part of it, and use this property to build up the MST one edge at a time.

- We will go through two algorithms which finds the MST:

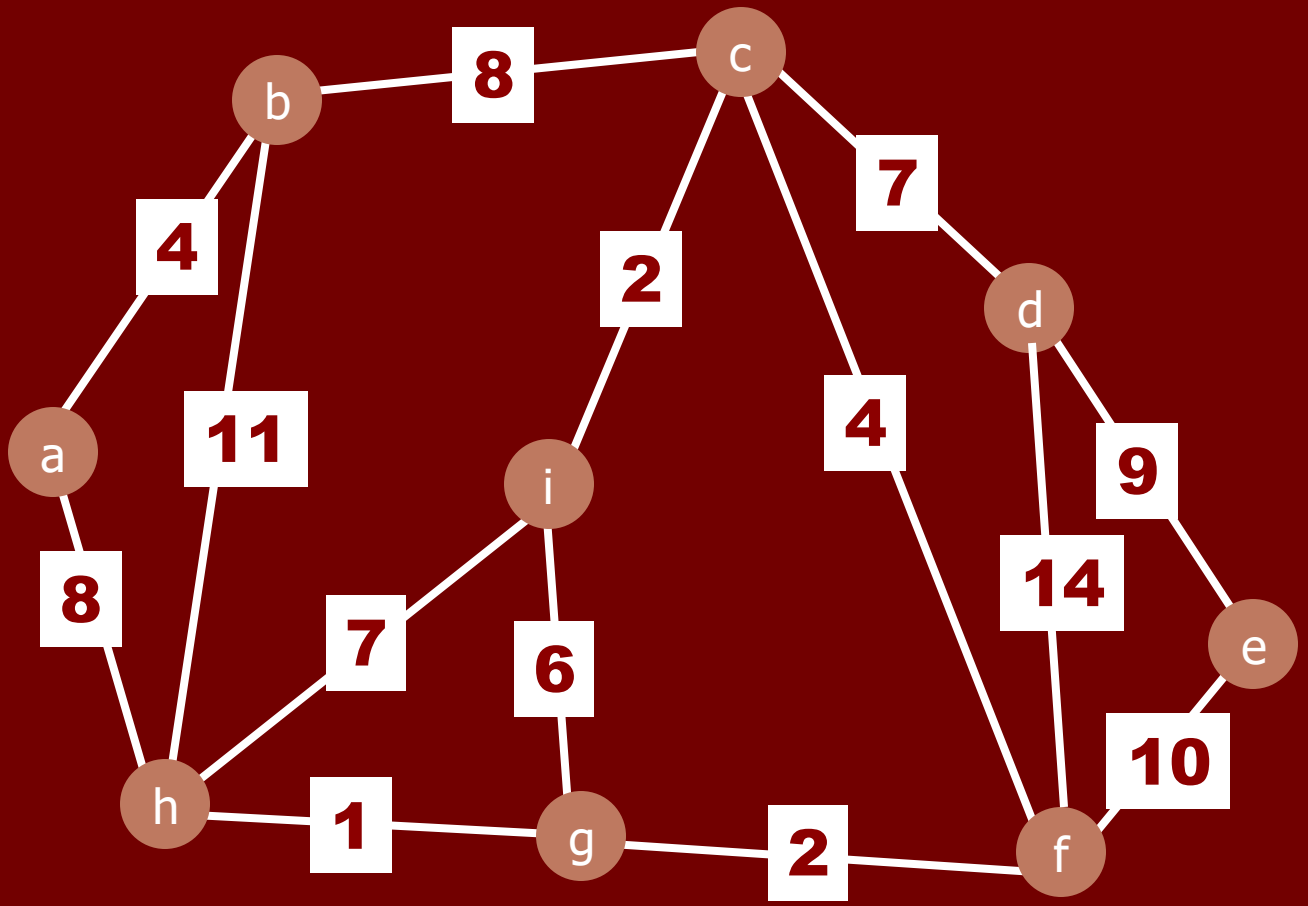
1. Kruskal's algorithm
2. Prim's algorithm

Kruskal's algorithm

1. sort the edges of G in increasing order by length
2. keep a sub graph S of G, initially empty
3. for each edge e in sorted order
4. if the endpoints of e are *disconnected* in S
   add e to S
5. return S

- Consider the following graph. Apply the previous algorithm on it to find the MST.

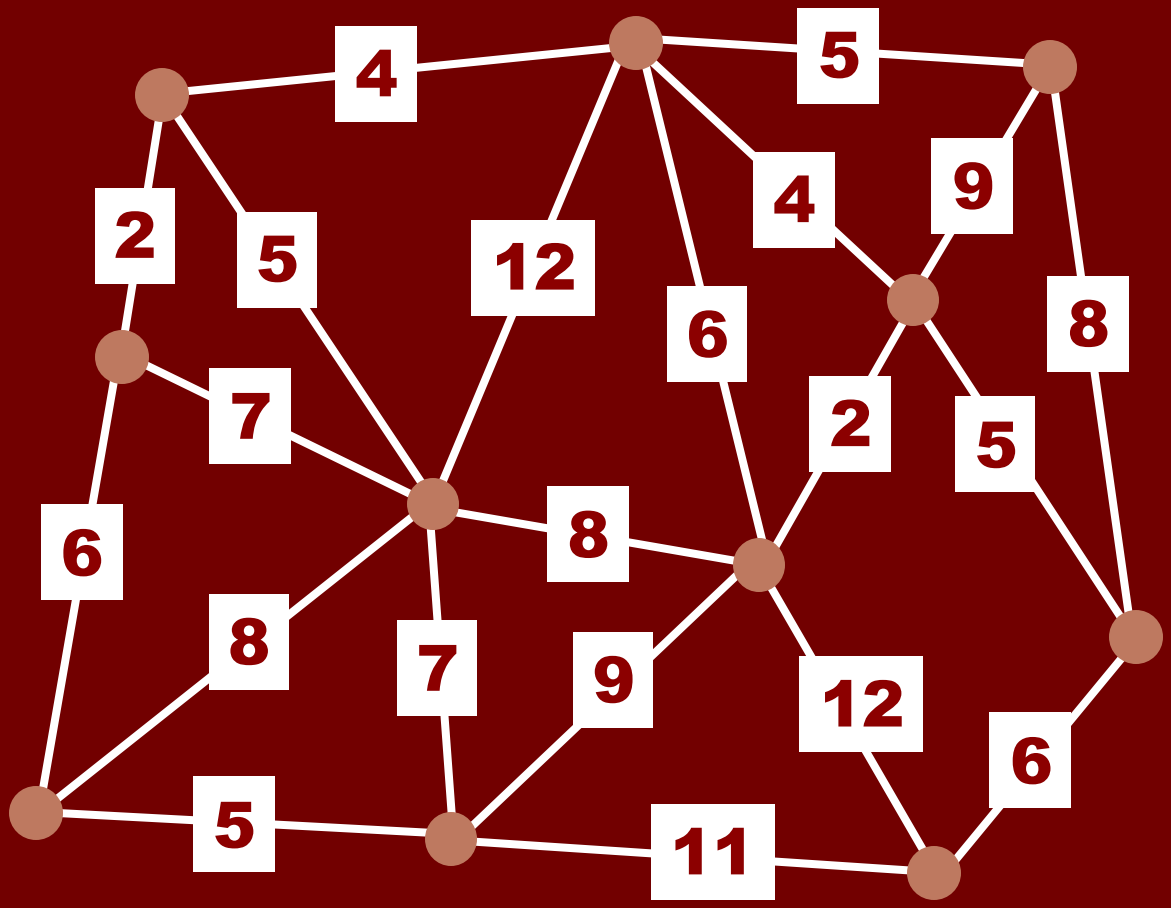
```
sort edges of G in increasing order by length
keep a sub graph S of G, initially empty
for each edge e in sorted order
    if the endpoints of e are disconnected in S
                add e to S
return S
```



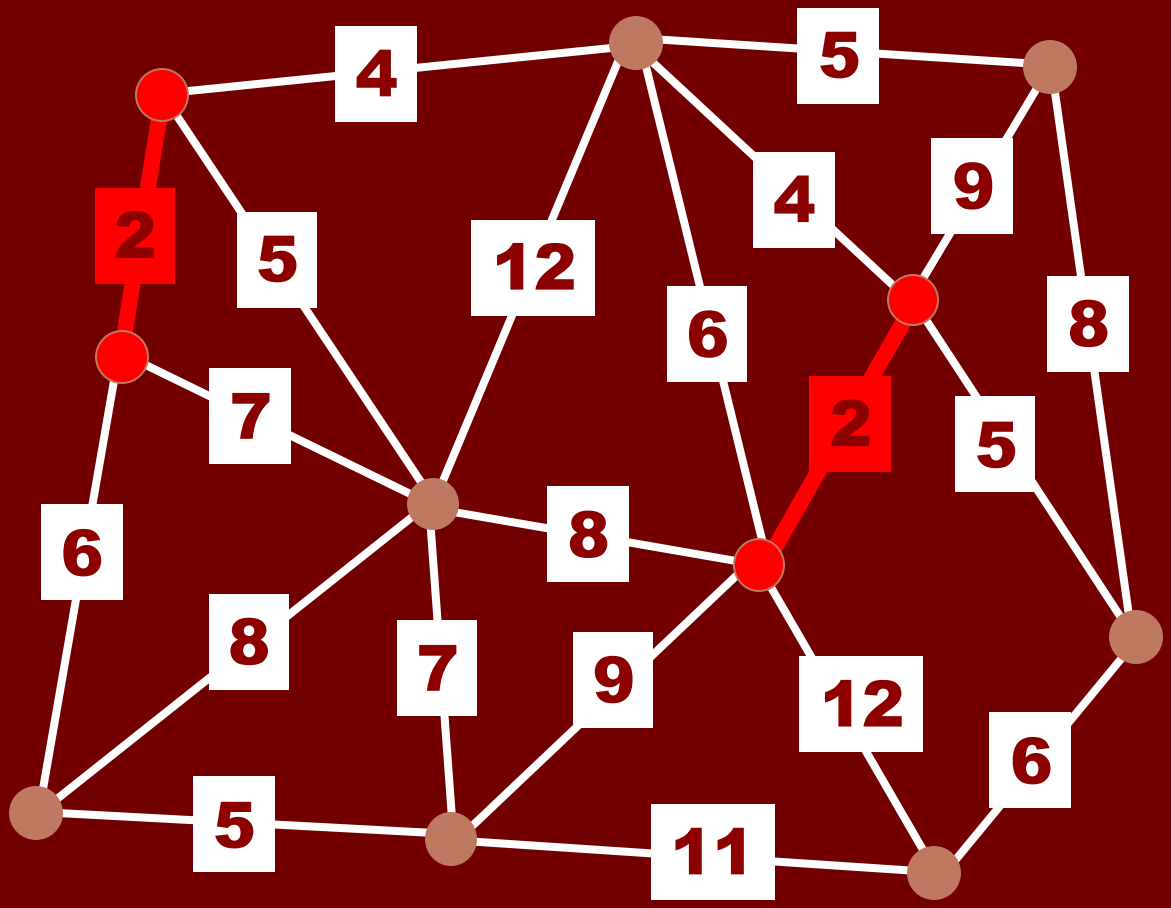
- This algorithm is known as a greedy algorithm, because it chooses at each step the cheapest edge to add to S. You should be very careful when trying to use greedy algorithms to solve other problems, since it usually doesn't work.

- E.g. if you want to find a shortest path from a to b, it might be a bad idea to keep taking the shortest edges. The greedy idea only works in Kruskal's algorithm.
- So we have to go through real life problems seriously and should opt the right choice of algorithms.

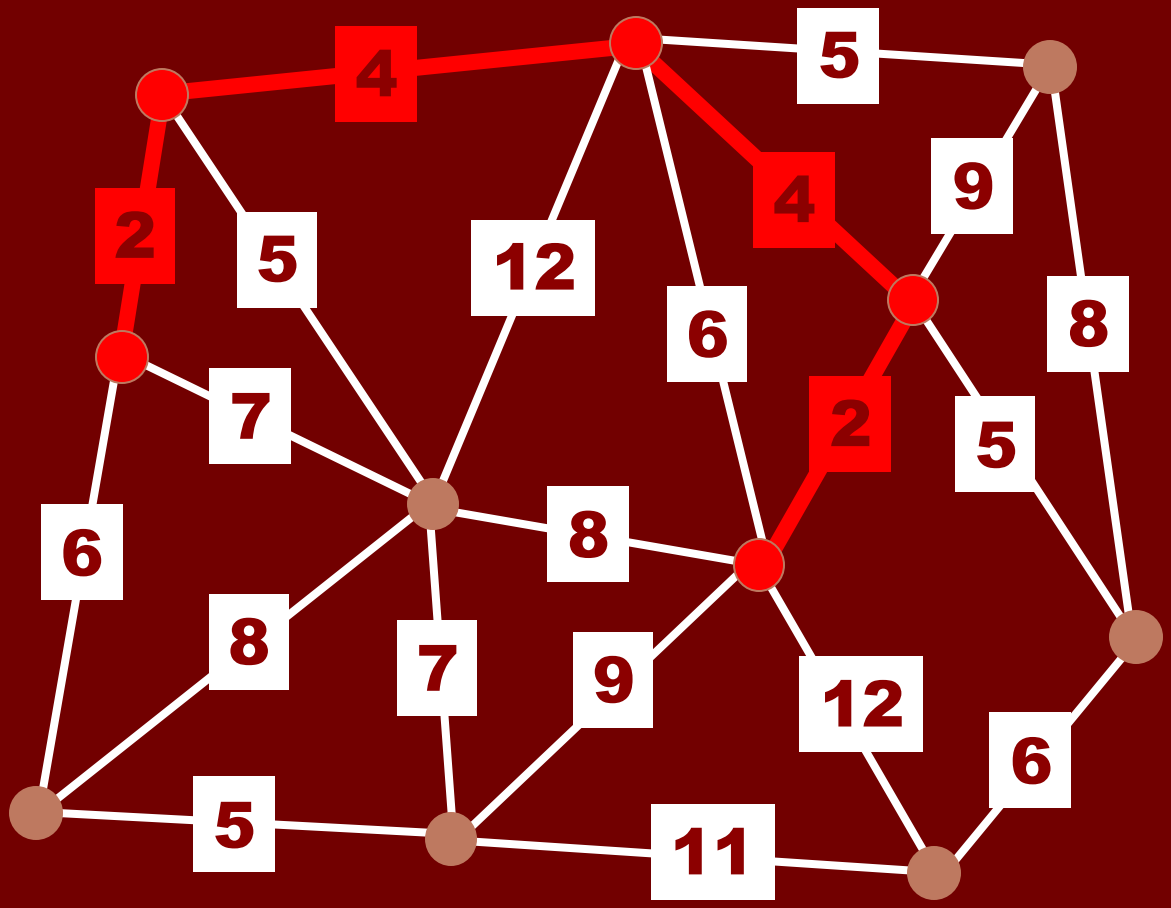
A Graph



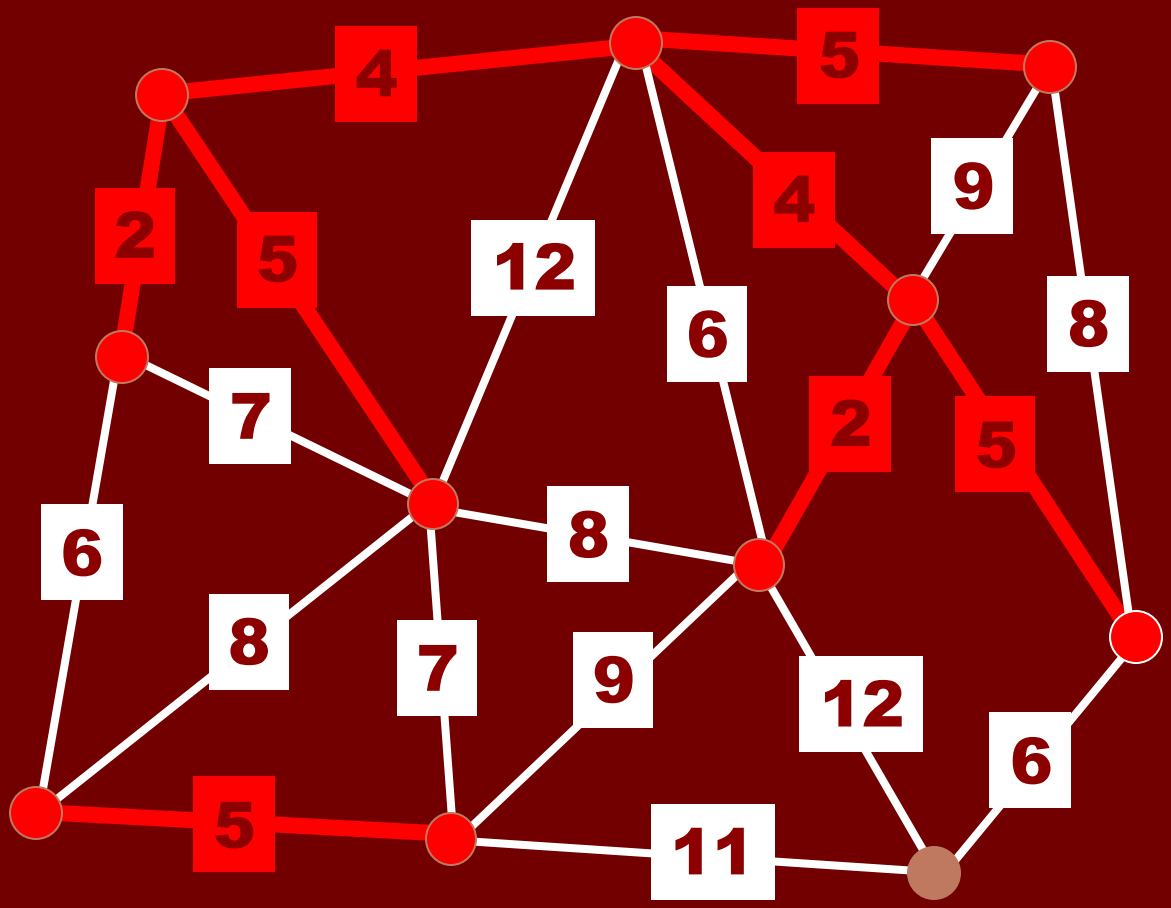
Kruskal's Algorithm



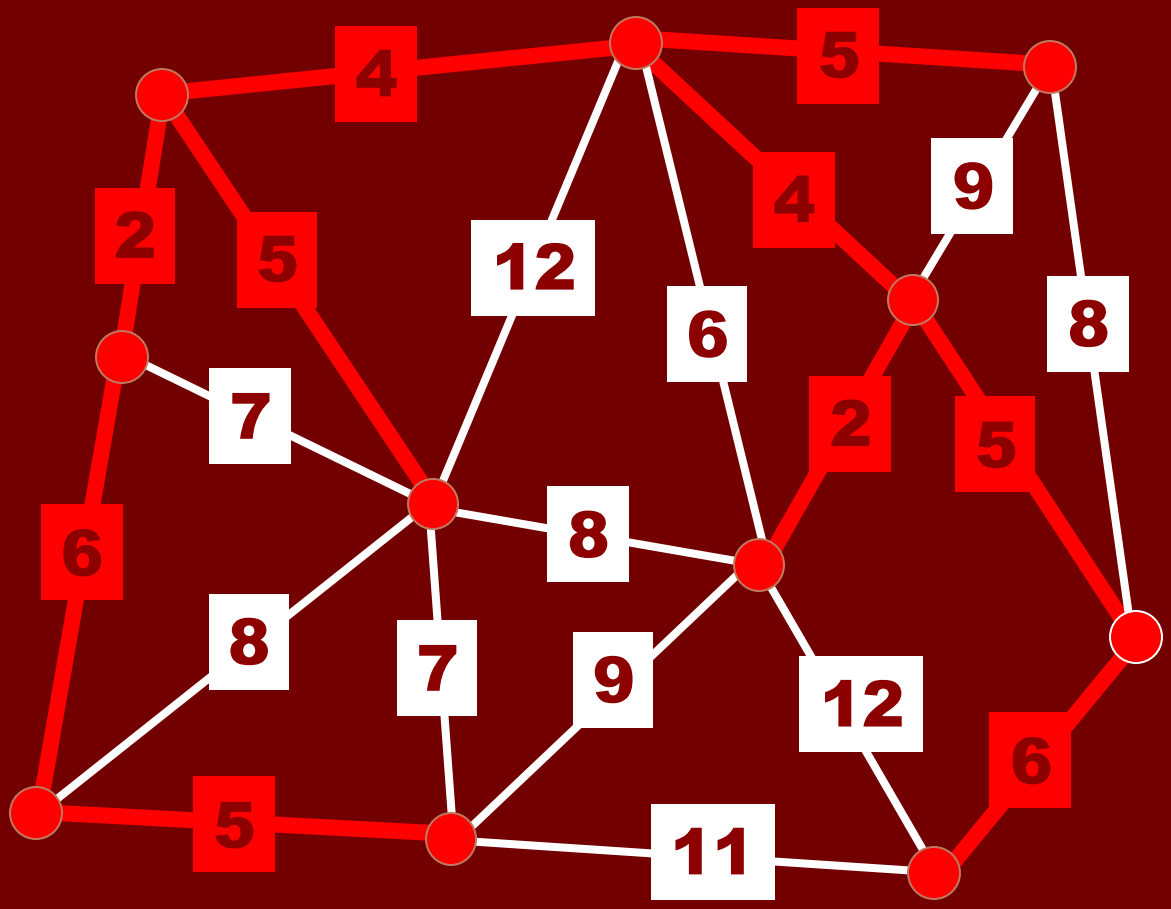
Kruskal's Algorithm



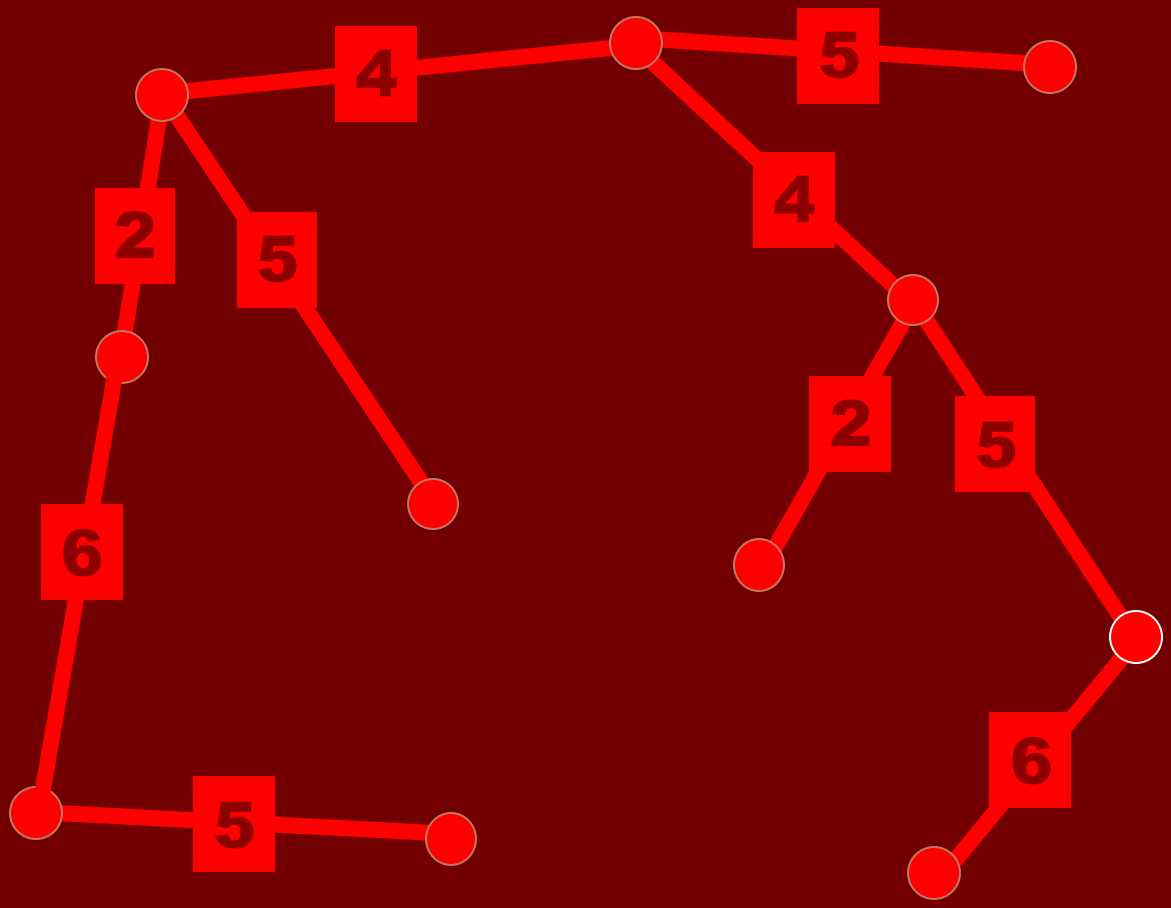
Kruskal's Algorithm



Kruskal's Algorithm



Kruskal's Algorithm



44

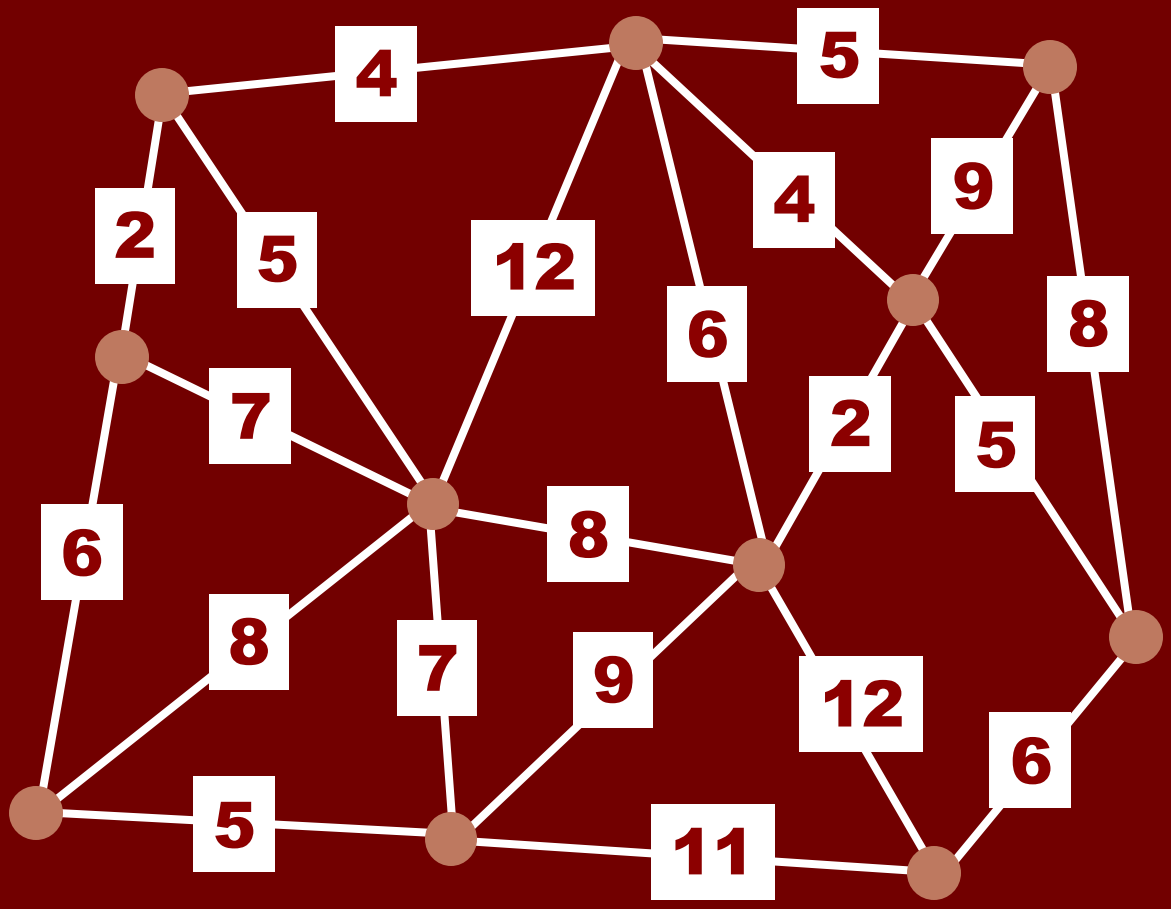
Analysis

- Since the graph is connected, we may assume that $E \geq V - 1$
- Sorting edges takes $\Theta(E \log E)$.
- The for loop performs **O(E) find** and **O(V) union** operations
- Thus the time is dominated by sorting. Overall time for Kruskal is $\Theta(E \log E) = \Theta(E \log V)$ if the graph is *sparse* (a graph with only a few edges, is a sparse graph).

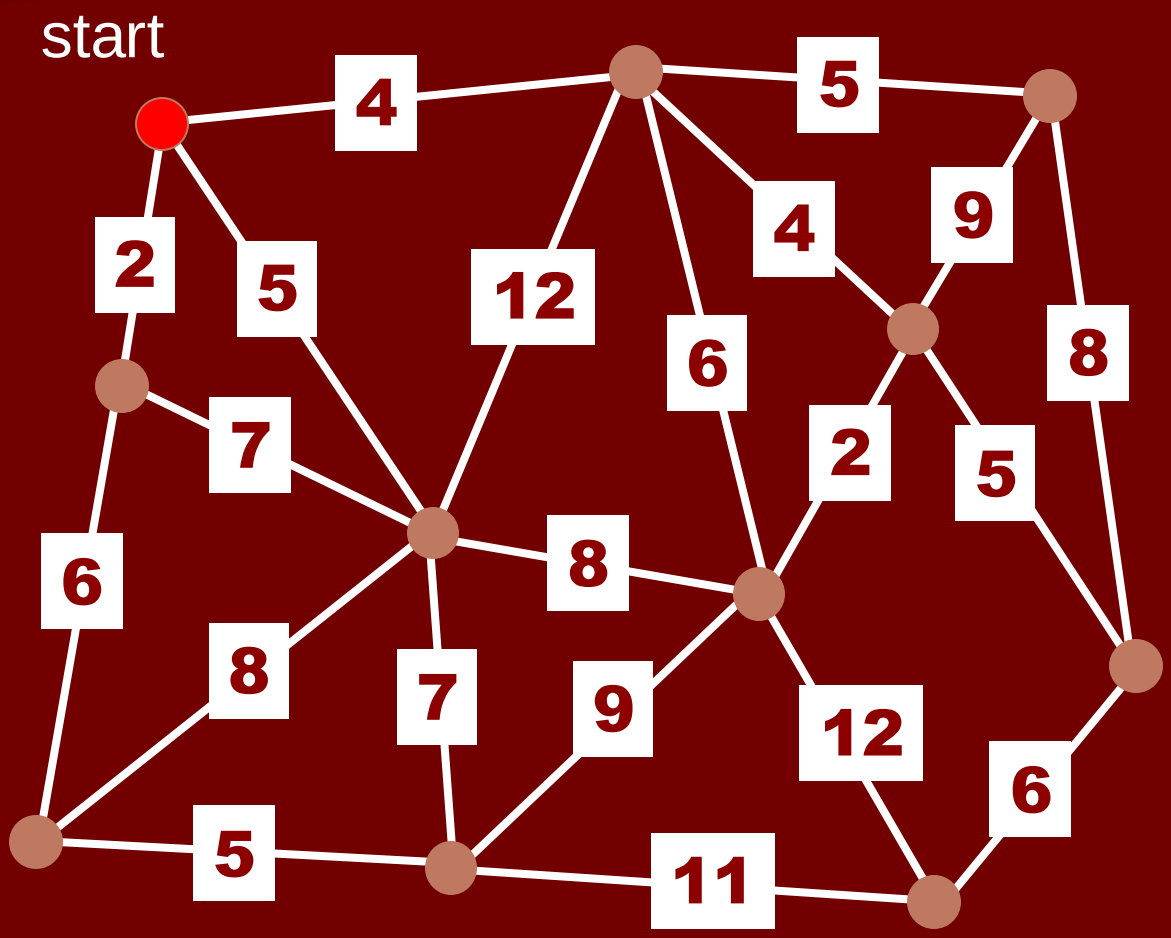
Prim's algorithm

- Kruskal's algorithm worked by ordering the edges, and inserting them one by one into the spanning tree, taking care never to introduce a cycle.
- Intuitively Kruskal's works by merging or splicing two trees together, until all the vertices are in the same tree.
- In contrast, Prim's algorithm builds the MST by adding leaves one at a time to the current tree. We start with a root vertex r; it can be any vertex. At any time, the subset of edges **A** forms a single tree (in Kruskal's, it formed a forest). We look to add a single vertex as a leaf to the tree.

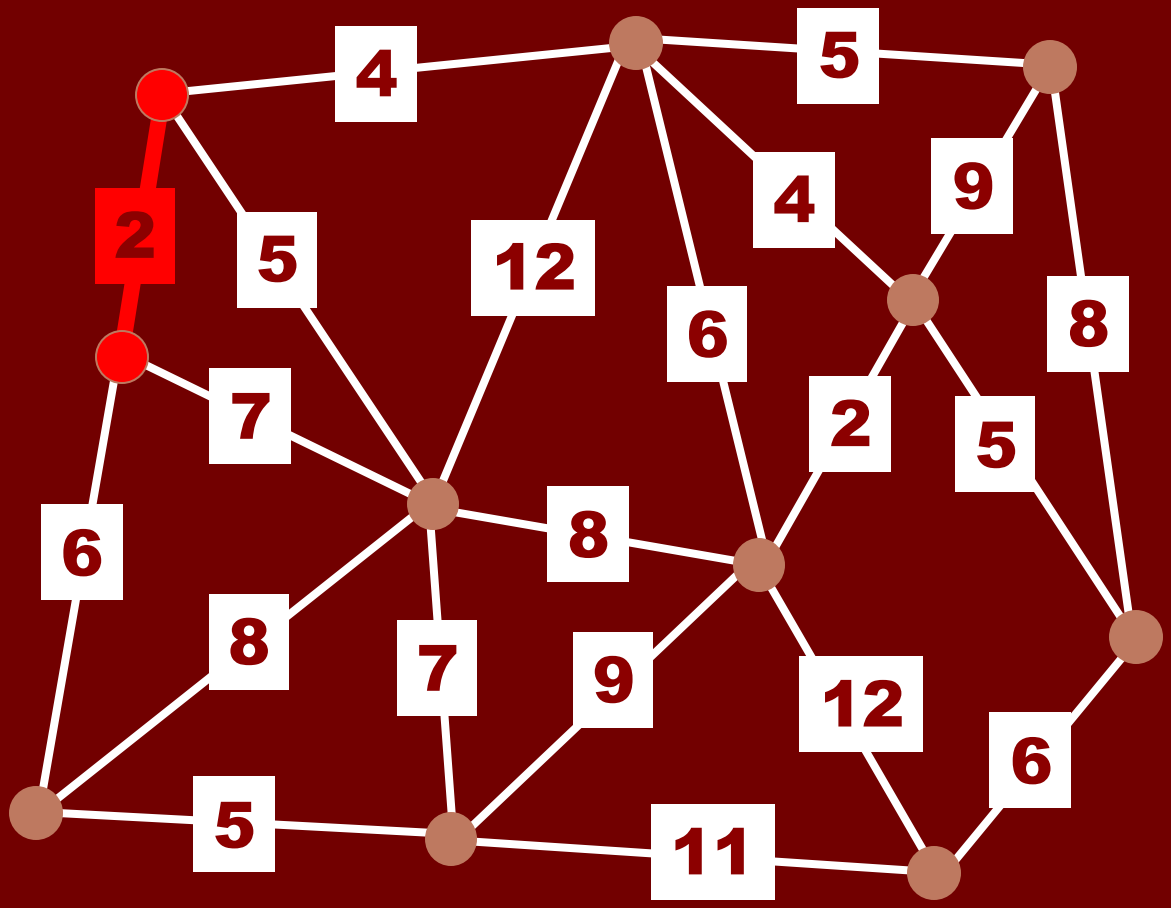
The Same Graph



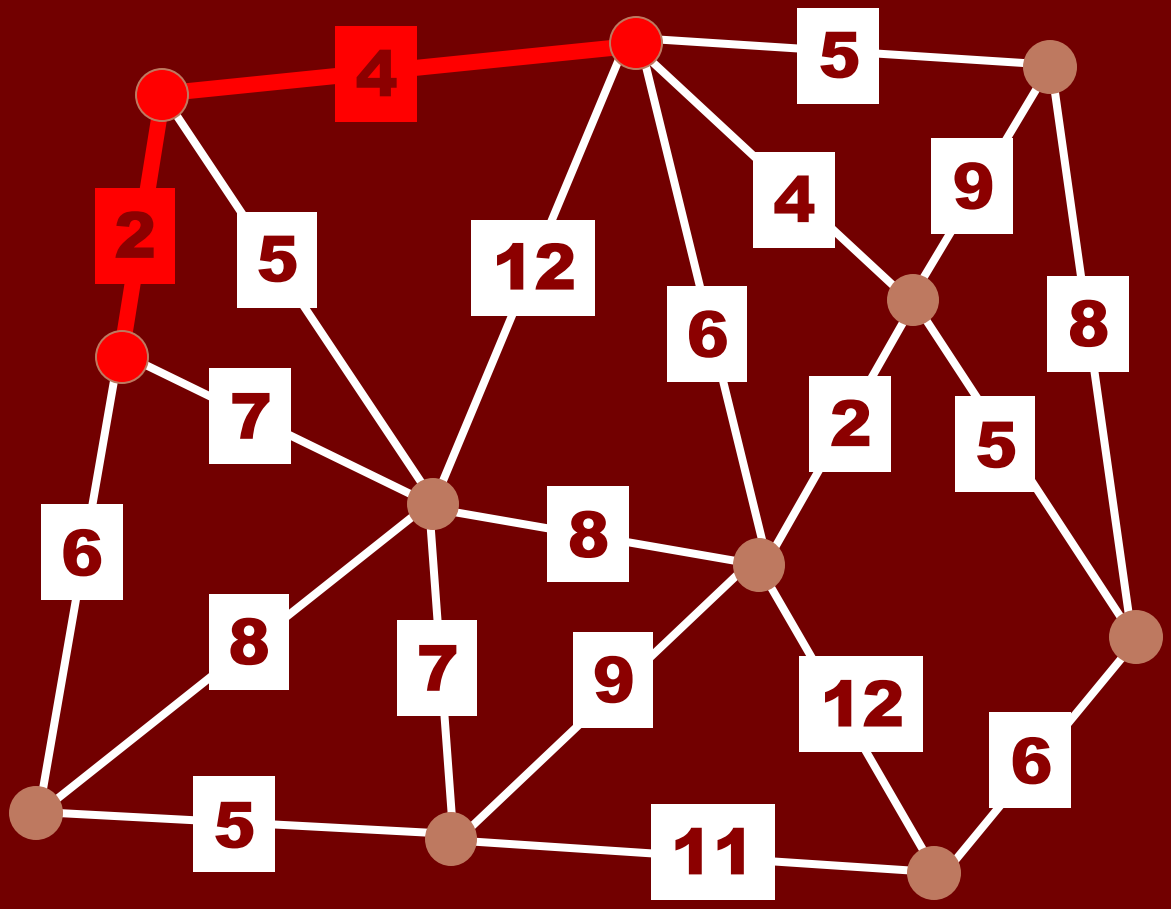
Prim's Algorithm



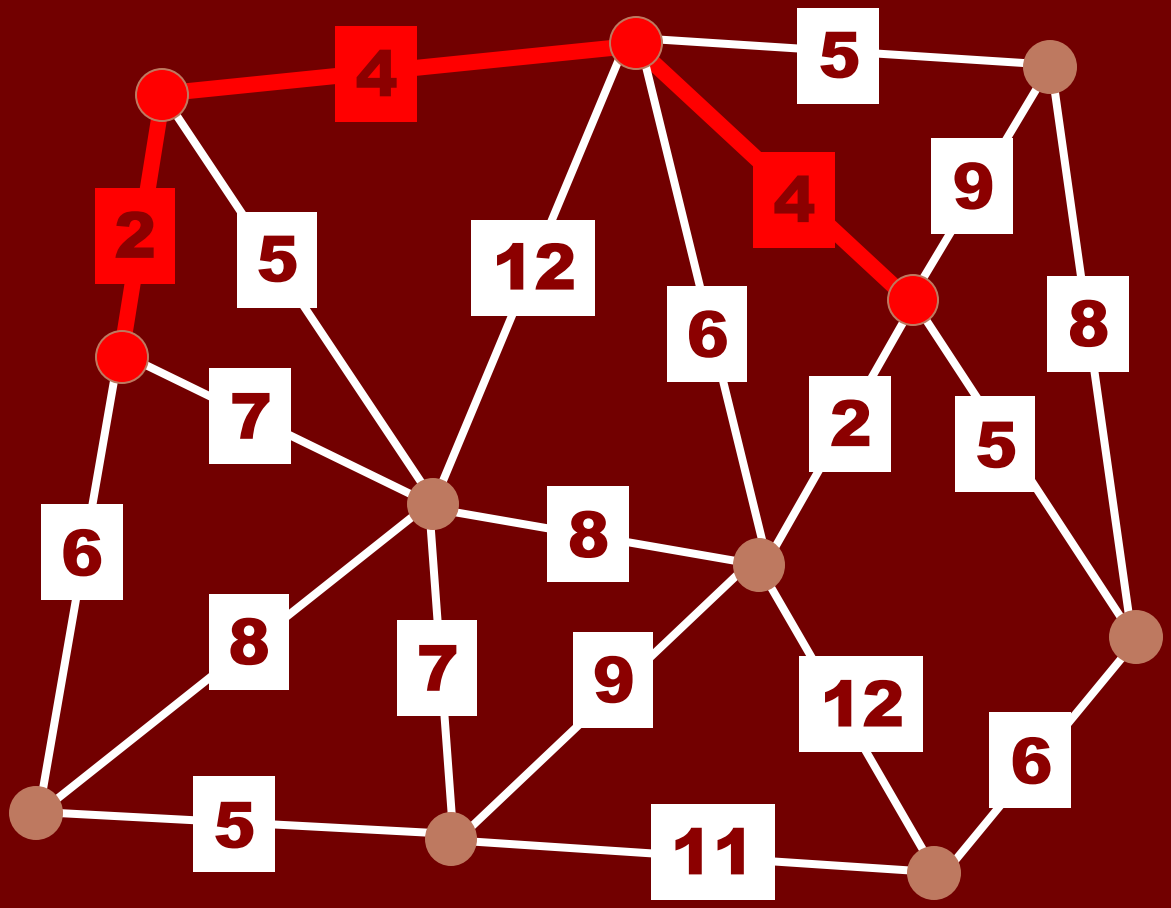
Prim's Algorithm



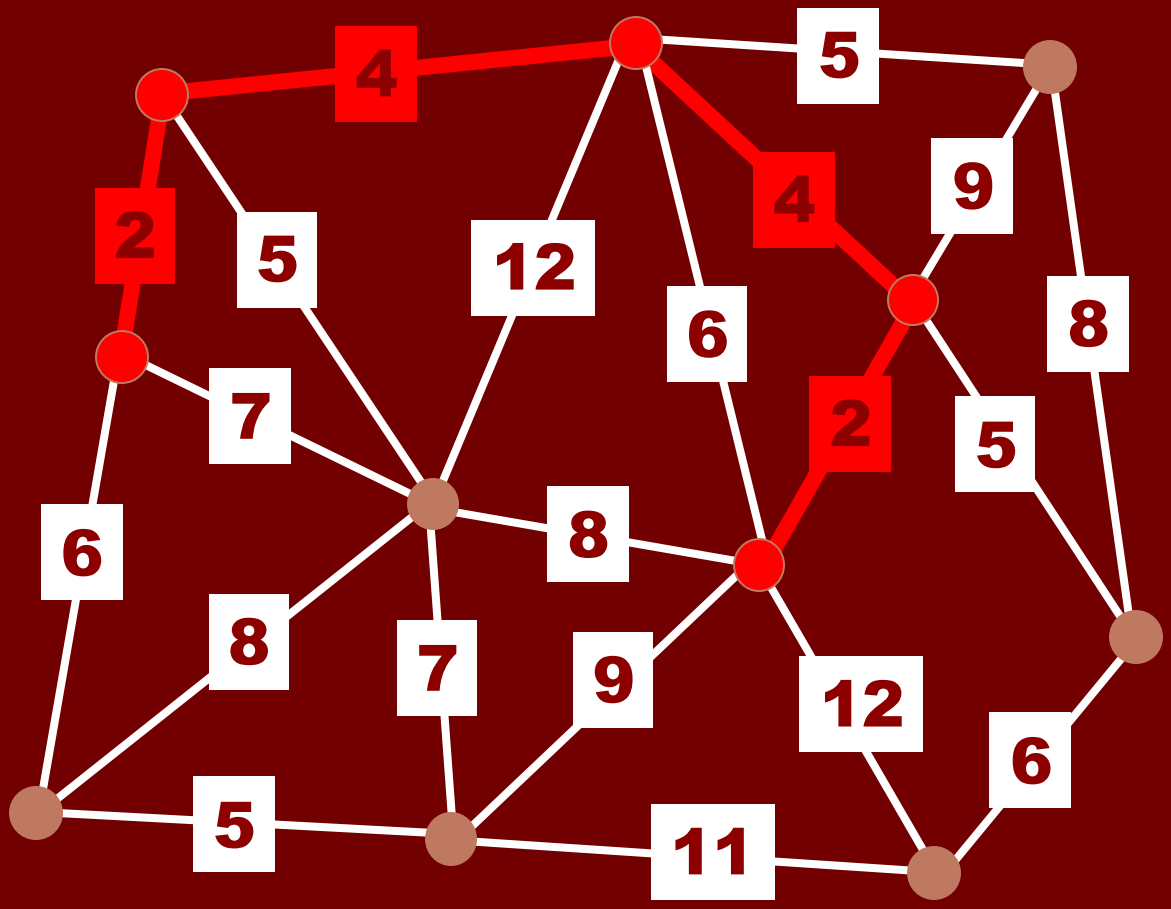
Prim's Algorithm



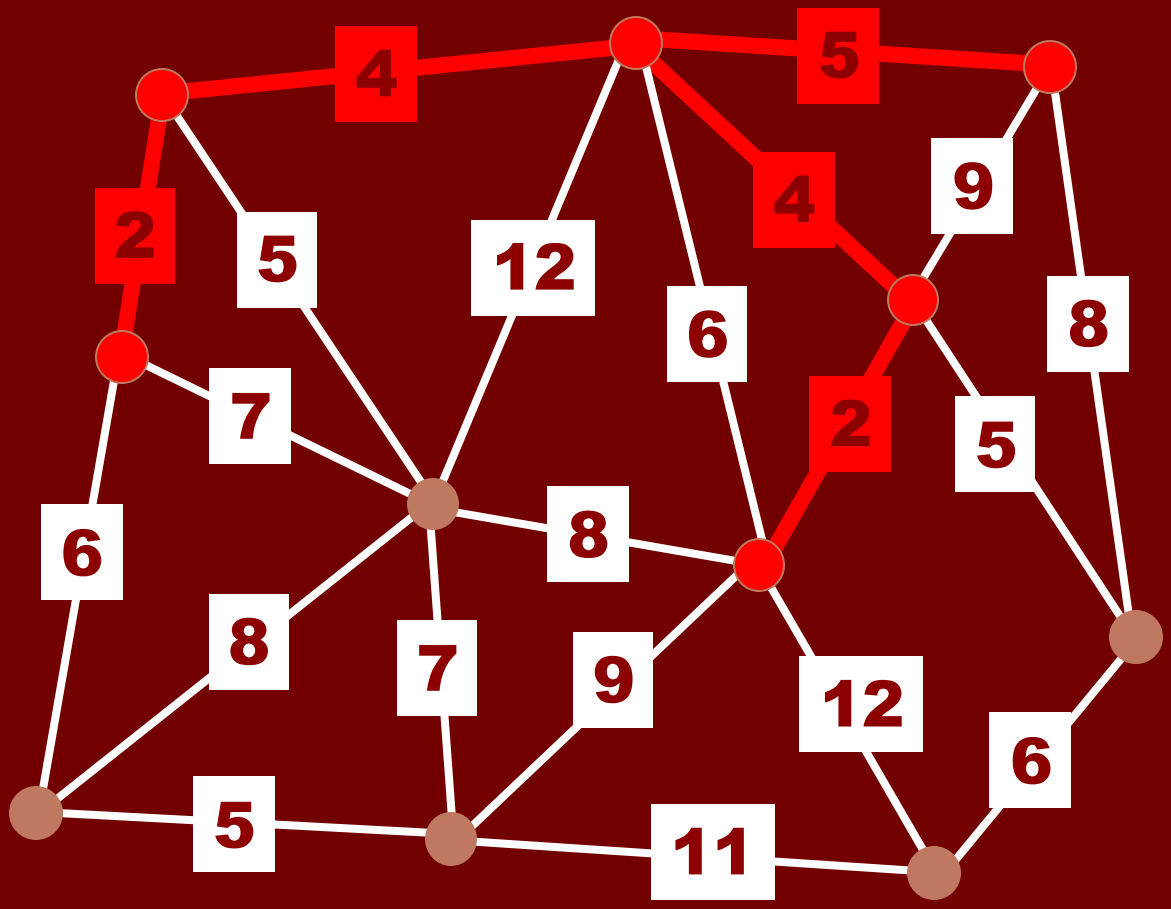
Prim's Algorithm



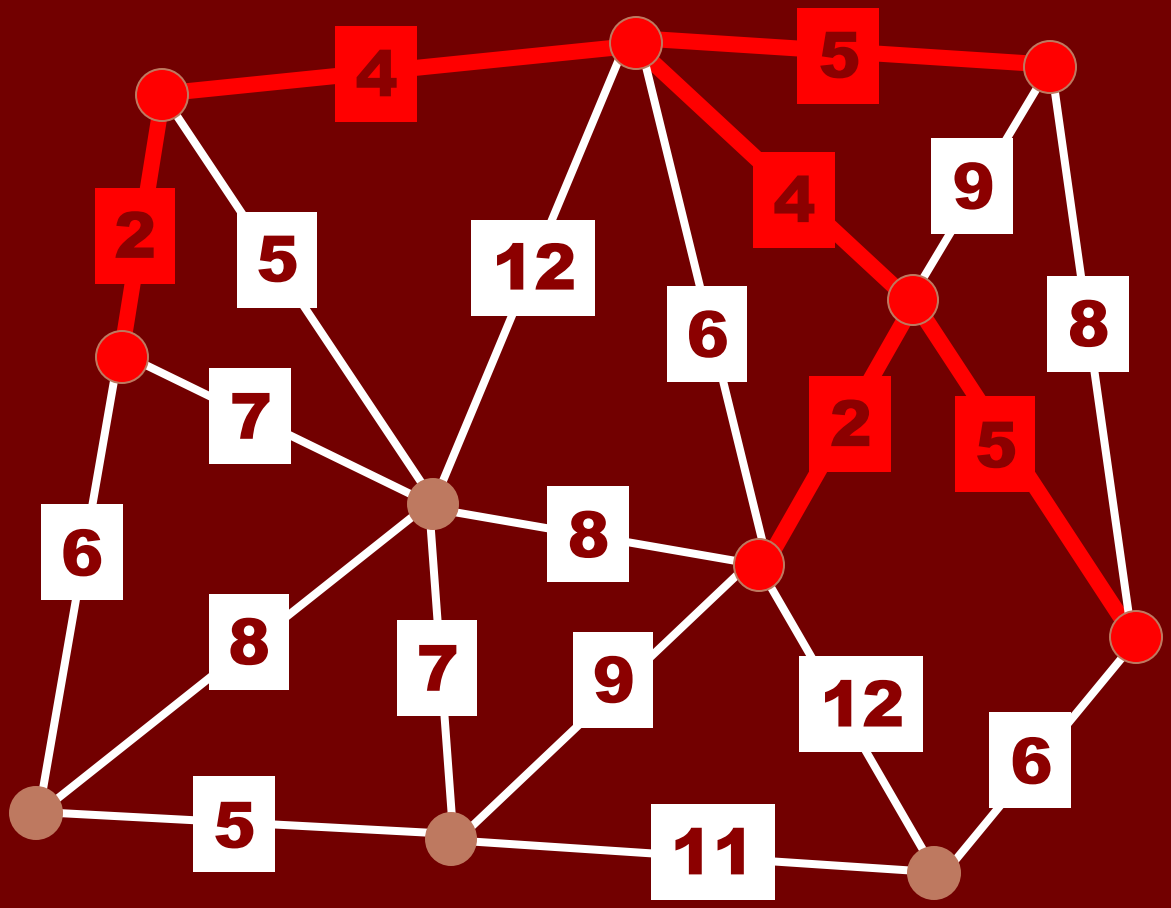
Prim's Algorithm



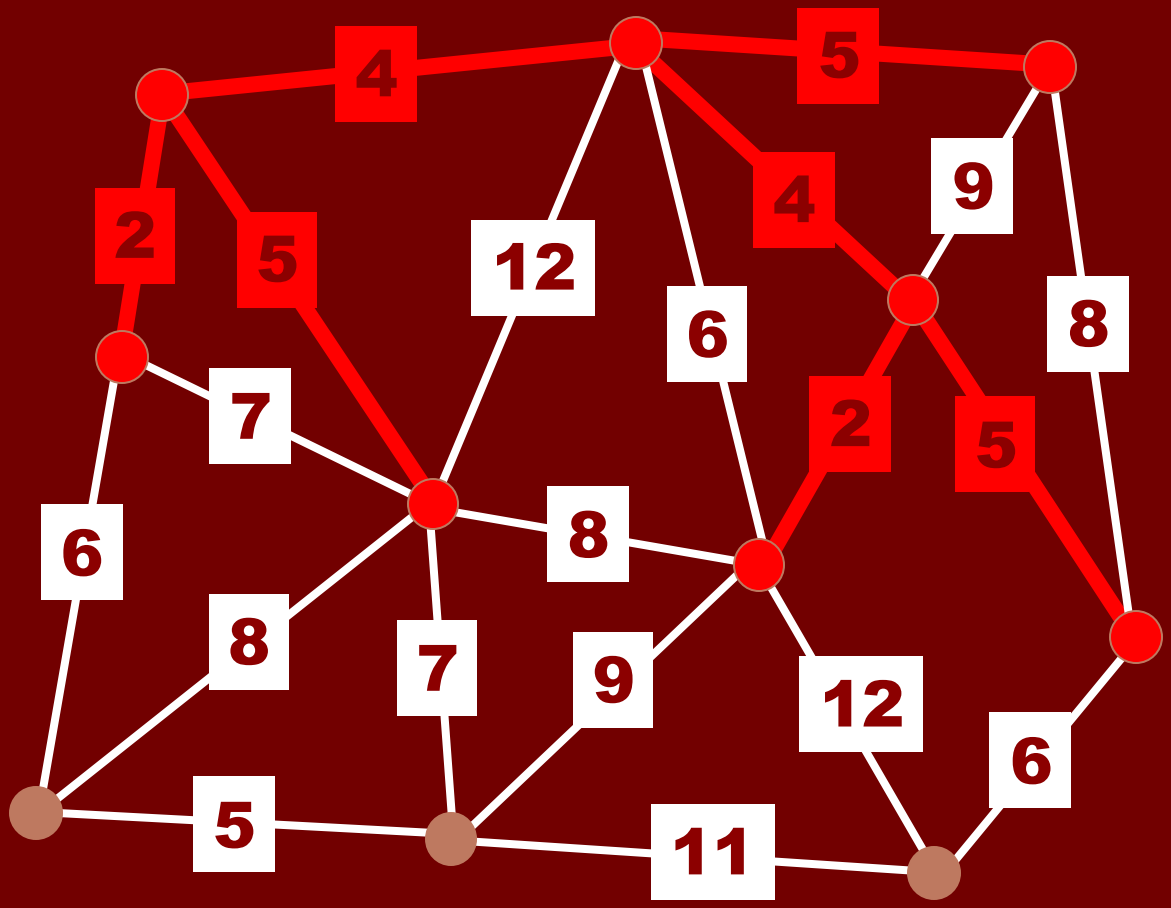
Prim's Algorithm



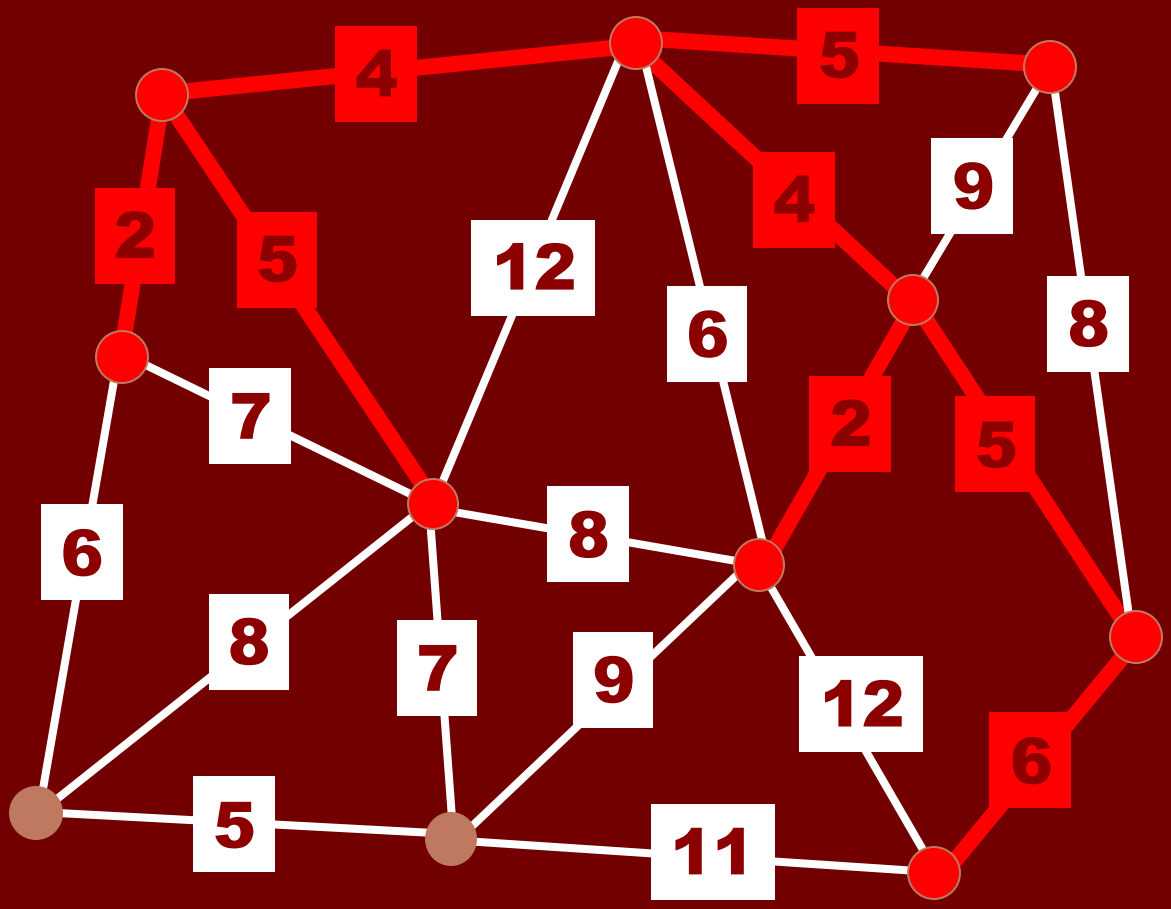
Prim's Algorithm



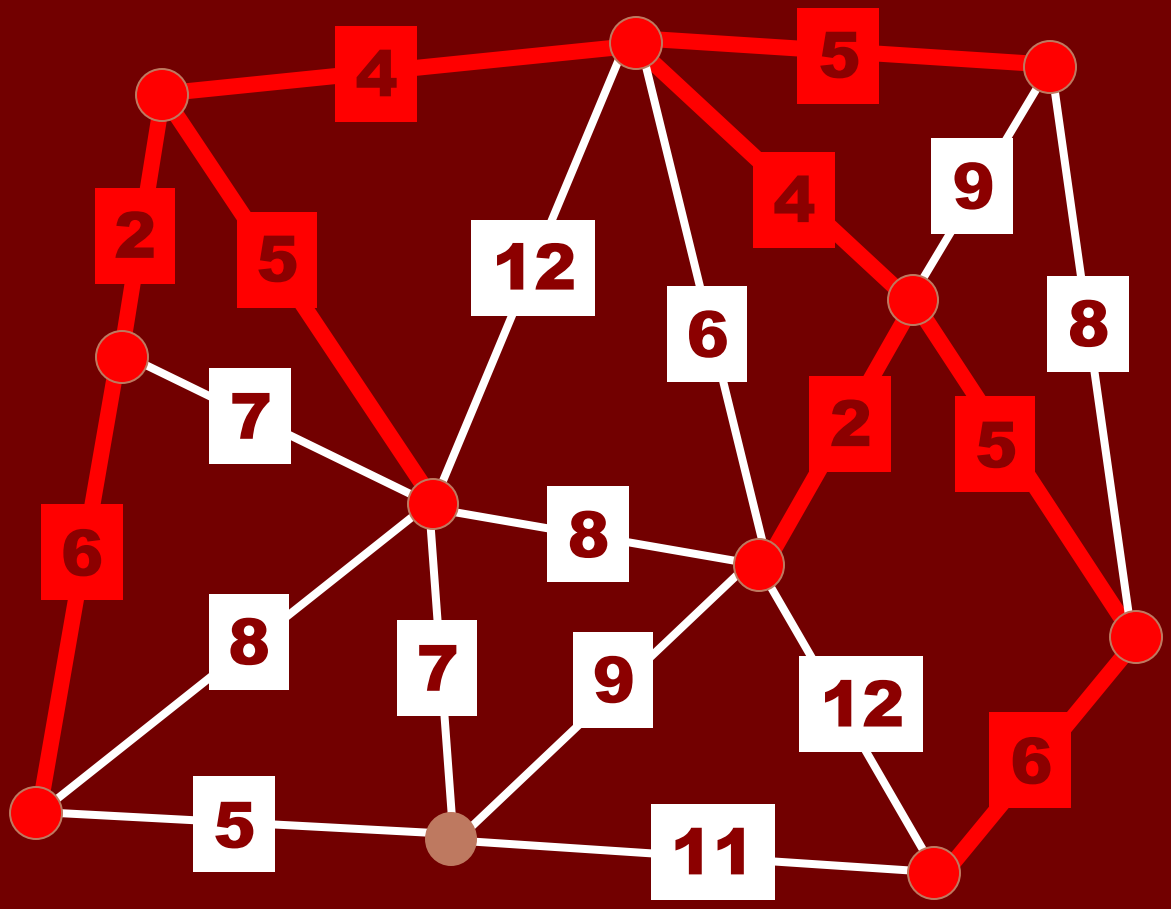
Prim's Algorithm



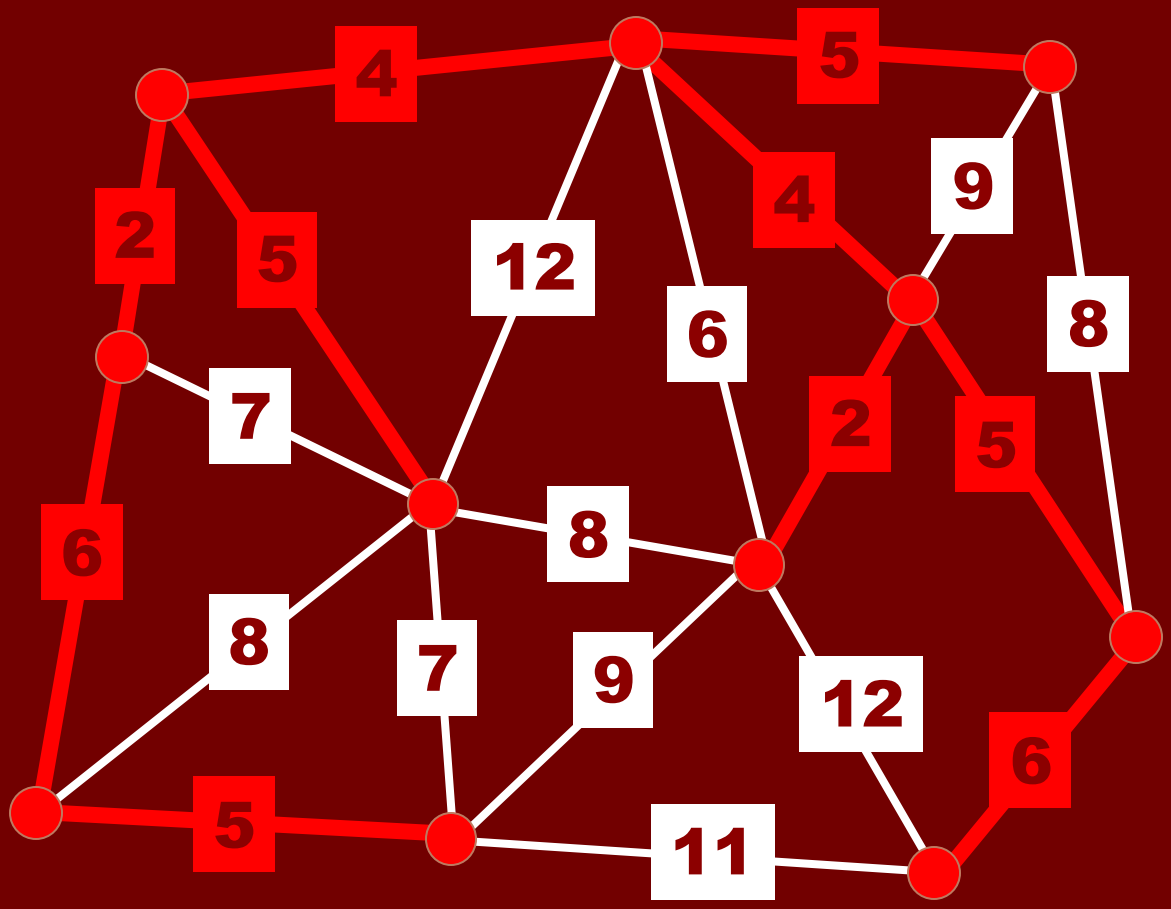
Prim's Algorithm



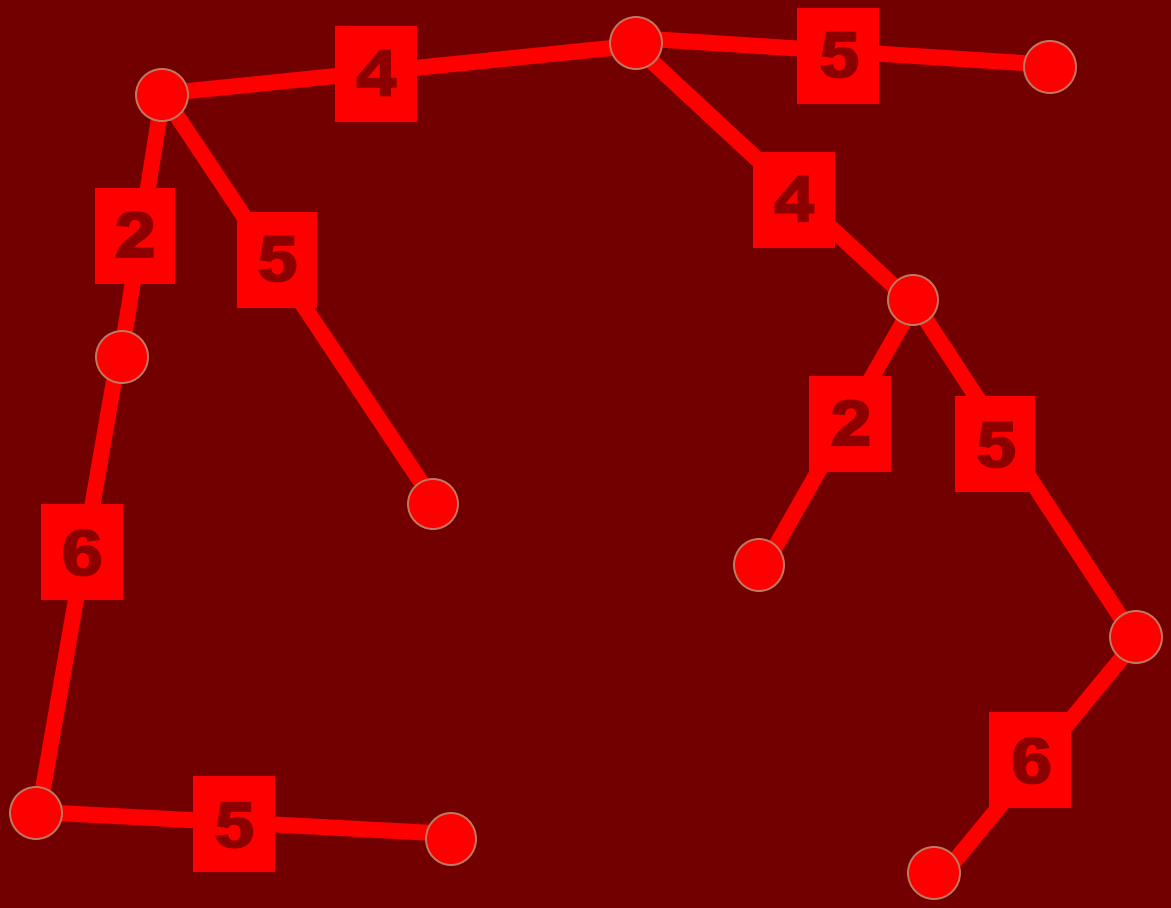
Prim's Algorithm



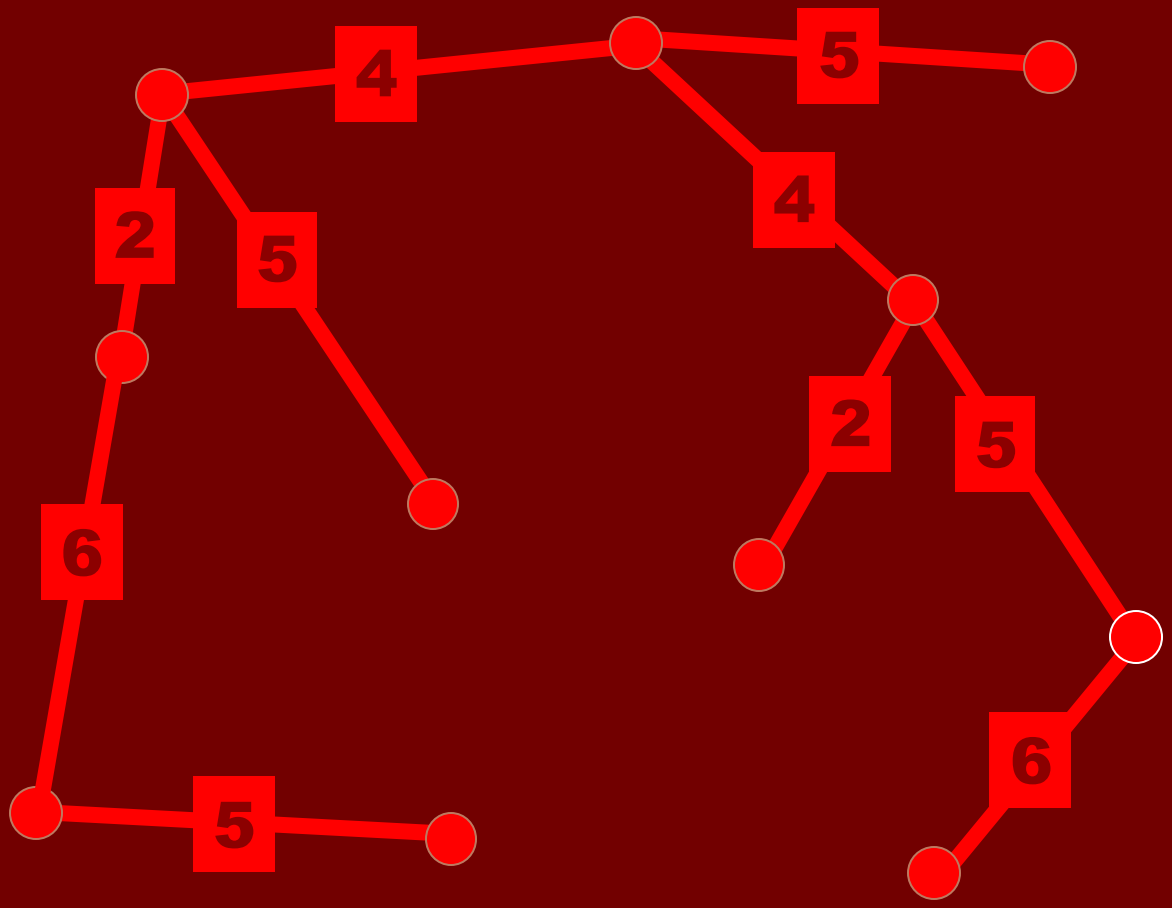
Prim's Algorithm



Prim's Algorithm



Kruskal's Algorithm



Why minimum spanning trees?

- The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money.

- A less obvious application is that the minimum spanning tree can be used to *approximately* solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.

- Connect all computers in a computer science building using least amount of cable.

- Note that if the weight in G are distinct, then the MST is unique.

Applications

- MST is central combinatorial problem with diverse applications.
  - Designing physical networks.
    - telephone, electrical, hydraulic, TV cable, computer, road
  - Cluster analysis.
    - delete long edges leaves connected components
    - finding clusters of quasars and Seyfert galaxies
    - analyzing fungal spore spatial patterns
  - Approximate solutions to NP-hard problems.
    - metric TSP, Steiner tree
  - Indirect applications.
    - describing arrangements of nuclei in skin cells for cancer research
    - learning salient features for real-time face verification
    - modeling locality of particle interactions in turbulent fluid flow
    - reducing data storage in sequencing amino acids in a protein

Thank You …