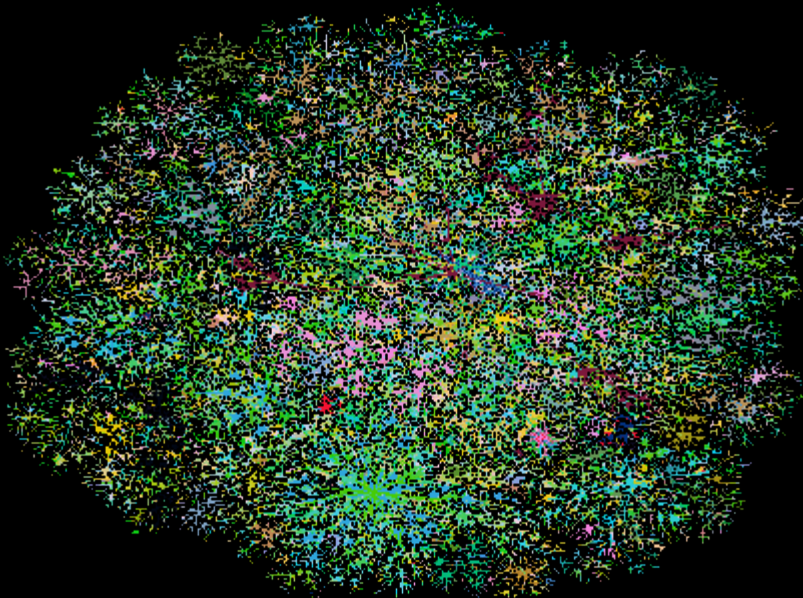


CS/ENGRD 2110

Object-Oriented Programming and Data Structures

Fall 2012

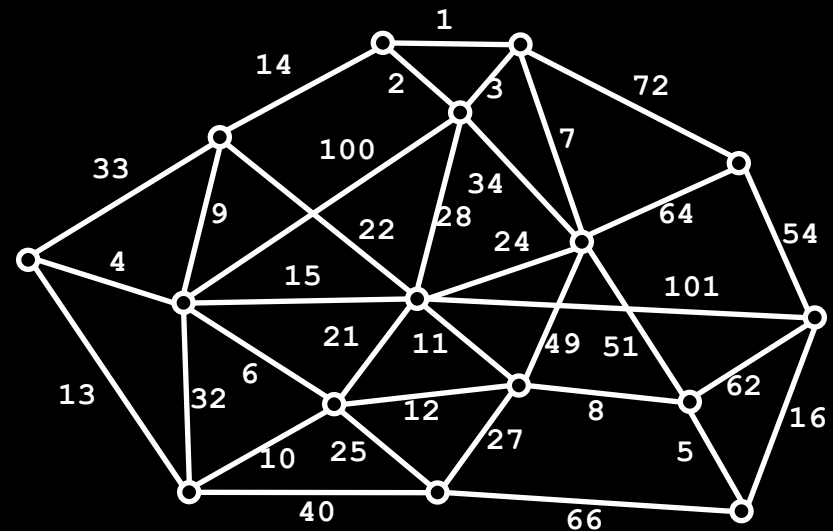
Doug James



Lecture 20: Other Algorithms on Graphs

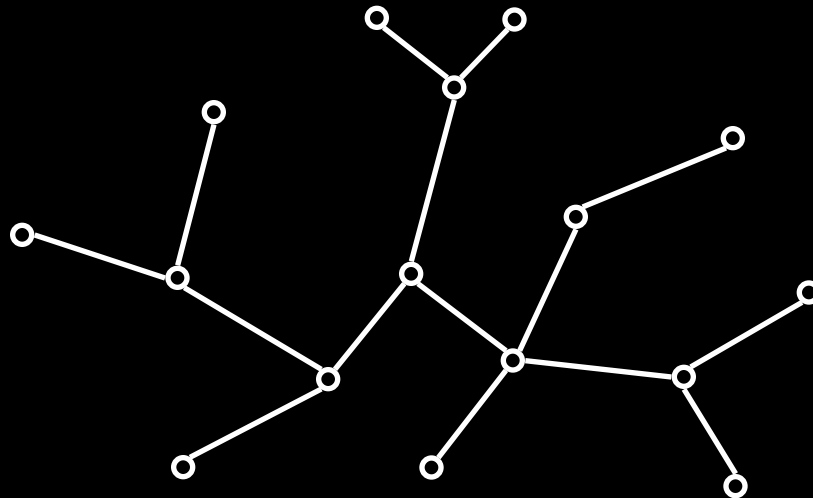
Minimum Spanning Trees

- Example Problem:
 - Nodes = neighborhoods
 - Edges = possible cable routes
 - Goal: Find lowest cost network that connects all neighborhoods
- Analogously:
 - Router network
 - Clustering
 - Component in many approximation algorithms



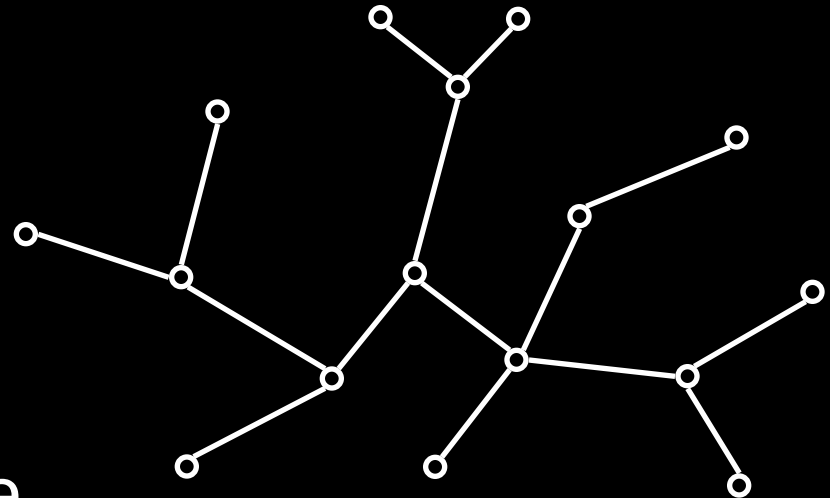
Undirected Trees

- An undirected graph is a *tree* if there is exactly one (simple) path between any pair of vertices



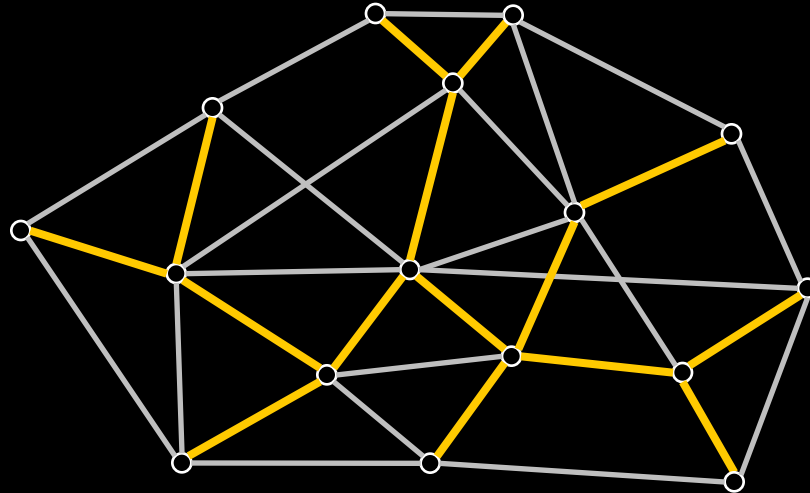
Facts About Trees

- Properties of (undirected) trees
 - $|E| = |V| - 1$
 - Connected
 - no cycles
- In fact, any two of these properties imply the third, and imply that the graph is a tree



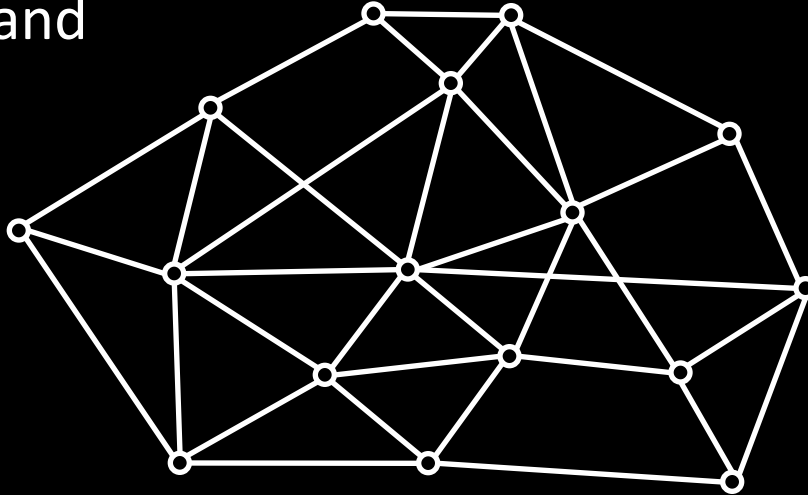
Spanning Trees

- A spanning tree of a connected undirected graph (V, E) is a subgraph (V, E') that is a tree
 - Same set of vertices V
 - $E' \subseteq E$
 - (V, E') is a tree



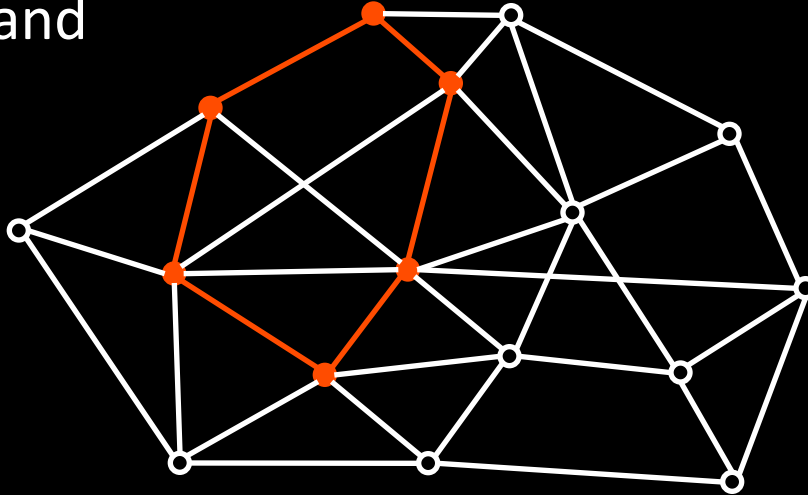
Finding a Spanning Tree

- A subtractive method
 - Start with the whole graph – it is connected
 - Find a cycle (how?), pick an edge on the cycle and throw it out
 - the graph is still connected (why?)
 - Repeat until no more cycles



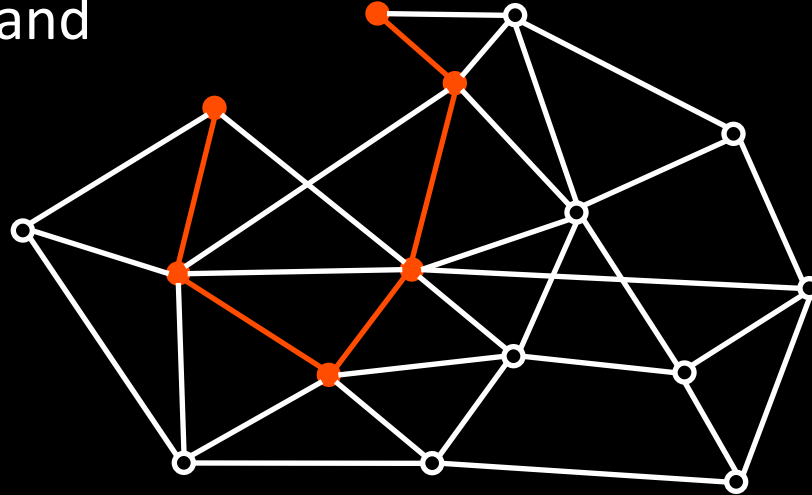
Finding a Spanning Tree

- A subtractive method
 - Start with the whole graph – it is connected
 - Find a cycle (how?), pick an edge on the cycle and throw it out
 - the graph is still connected (why?)
 - Repeat until no more cycles



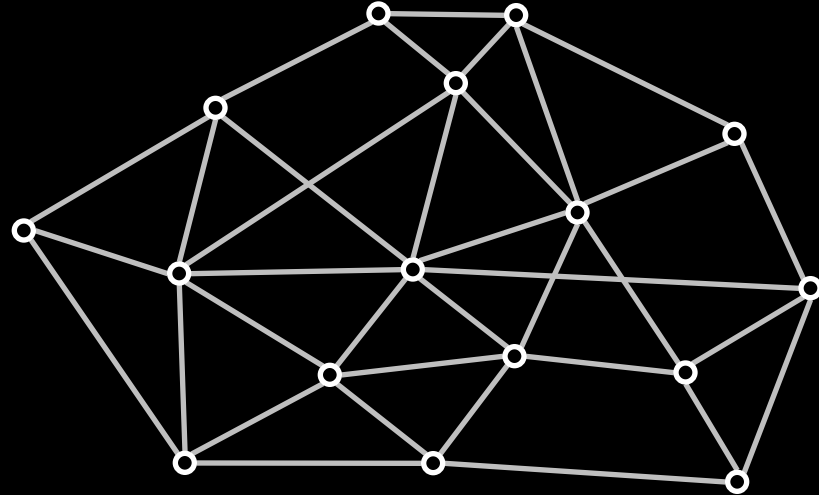
Finding a Spanning Tree

- A subtractive method
 - Start with the whole graph – it is connected
 - Find a cycle (how?), pick an edge on the cycle and throw it out
 - the graph is still connected (why?)
 - Repeat until no more cycles



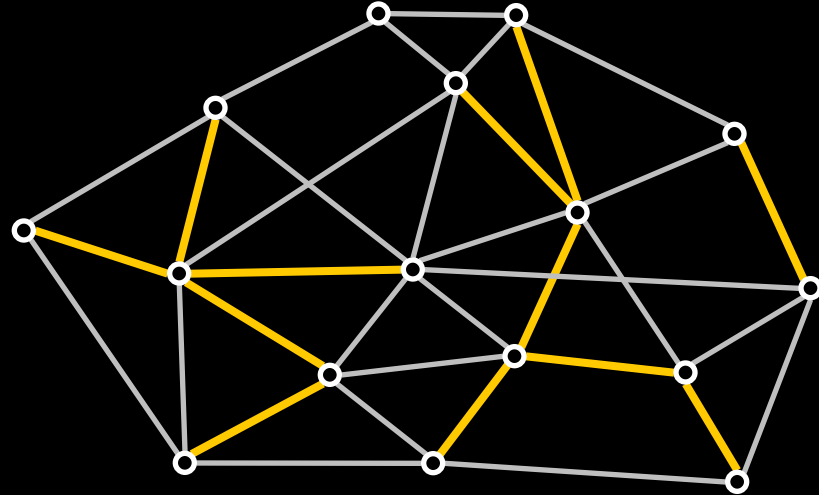
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



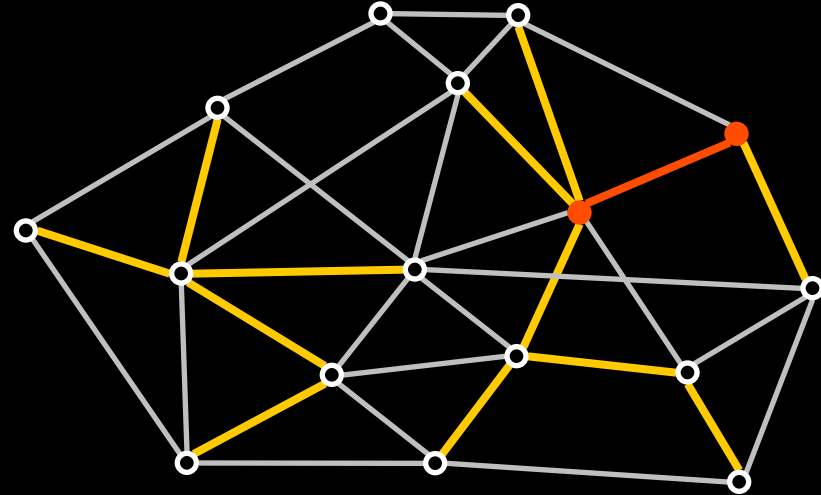
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



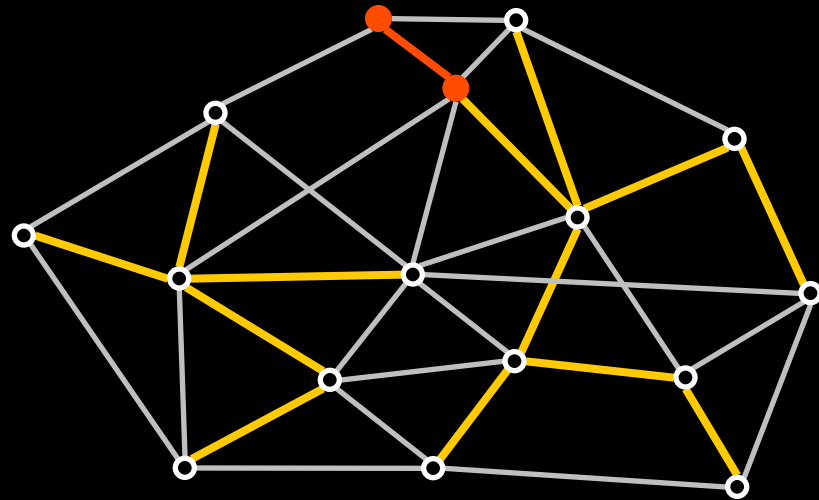
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



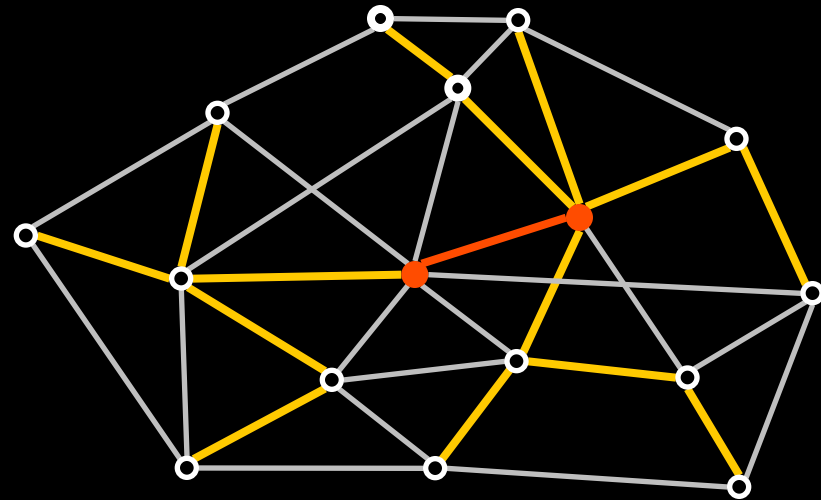
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



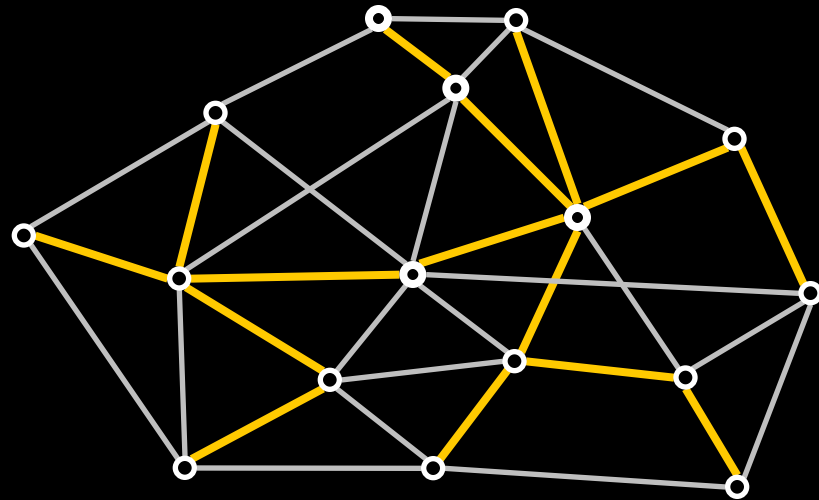
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



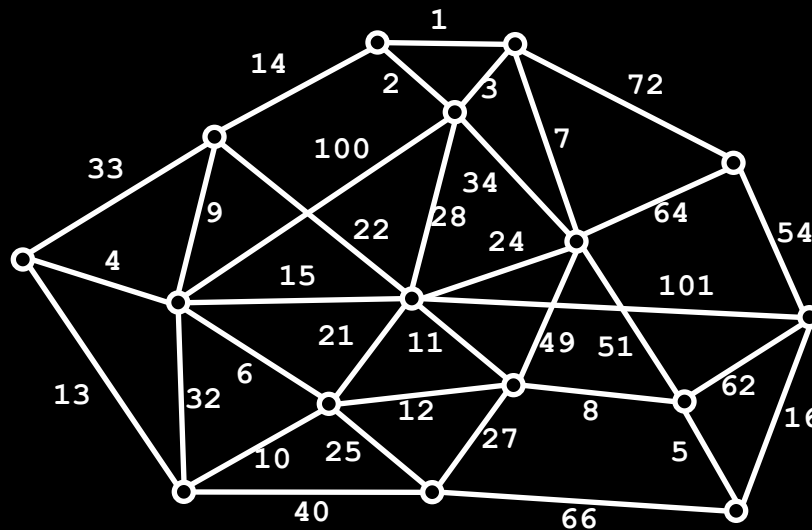
Finding a Spanning Tree

- An additive method
 - Start with no edges – there are no cycles
 - Find connected components (how?).
 - If more than one connected component, insert an edge between them
→ still no cycles (why?)
 - Repeat until only one component



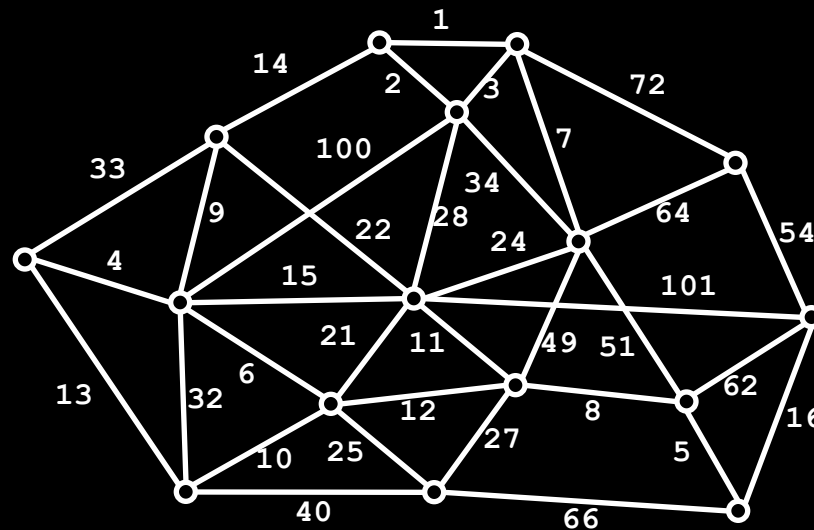
Minimum Spanning Trees

- Suppose edges are weighted, and we want a spanning tree of **minimum cost** (sum of edge weights)



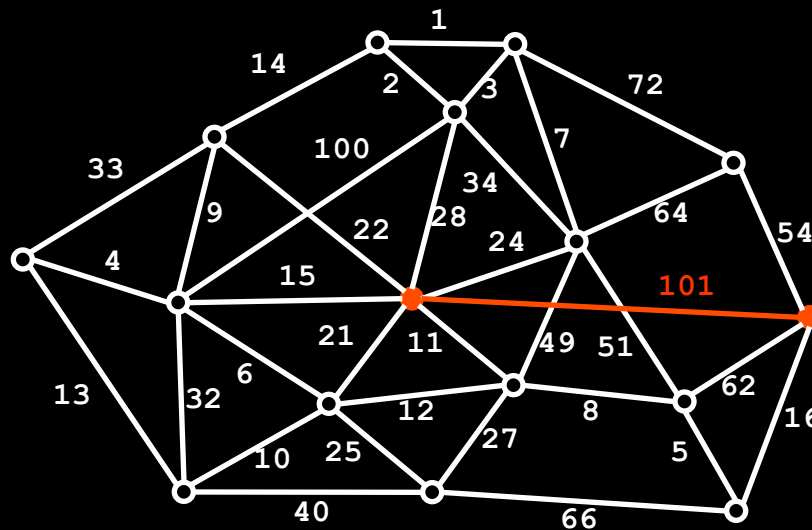
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



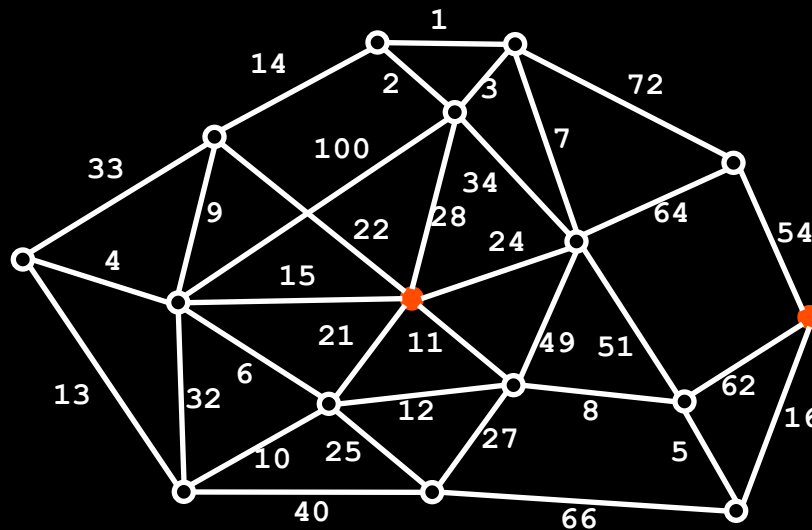
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



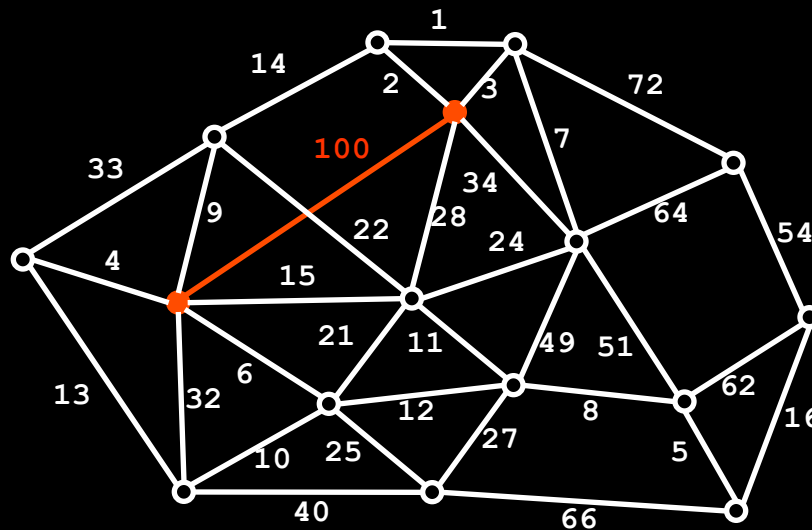
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



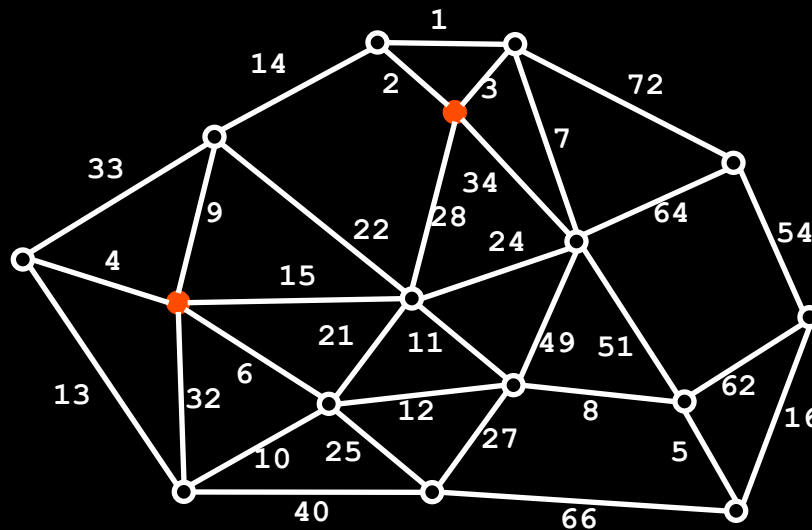
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



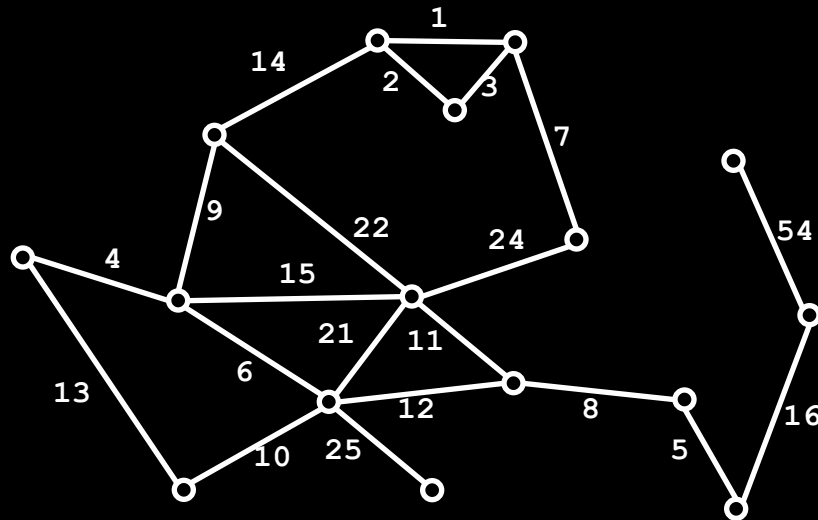
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



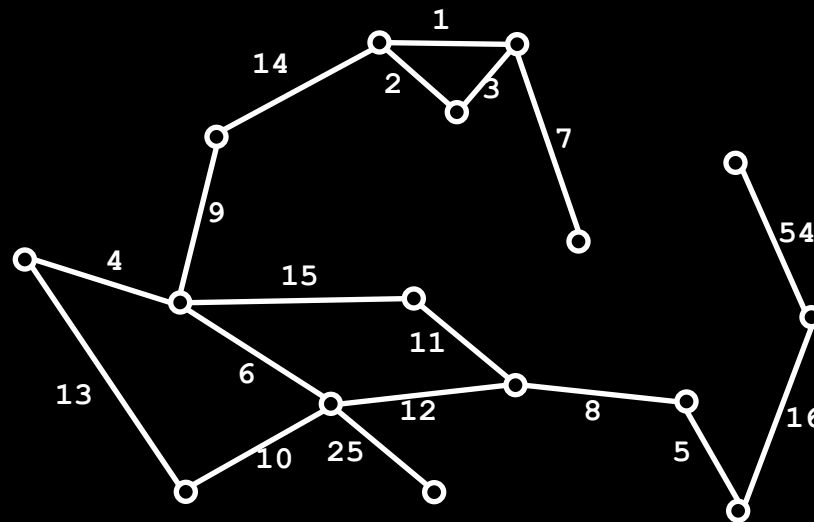
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



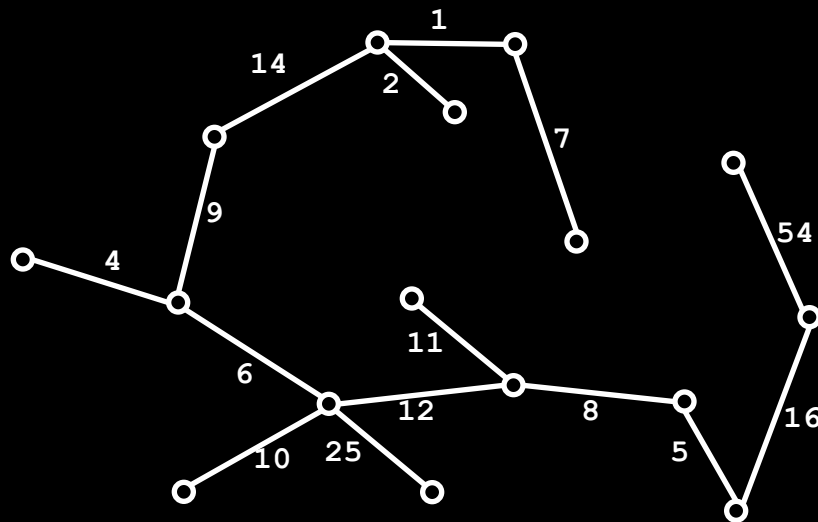
3 Greedy Algorithms

- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



3 Greedy Algorithms

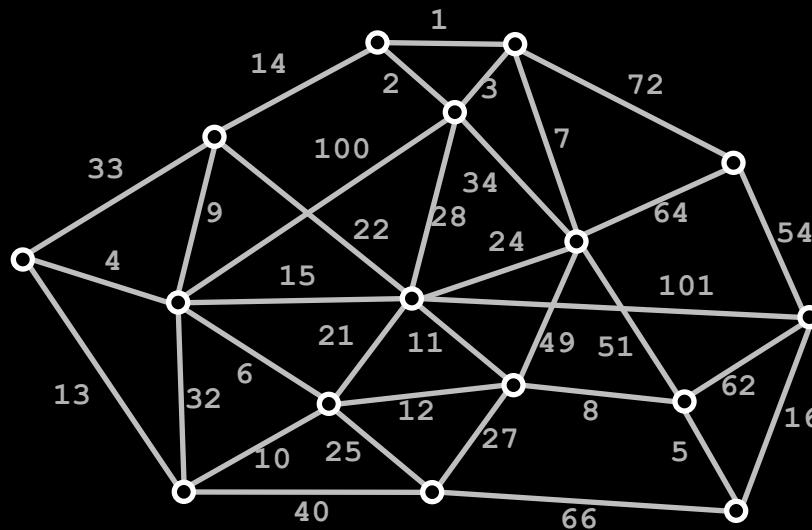
- Algorithm A: Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

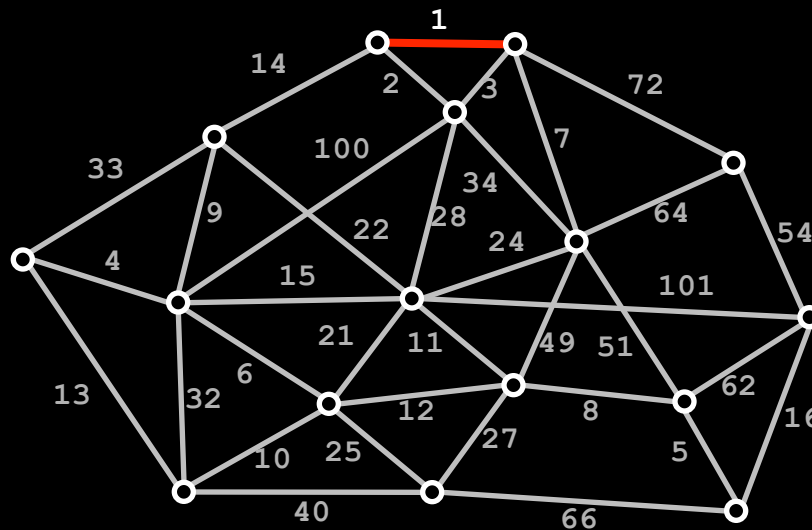
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

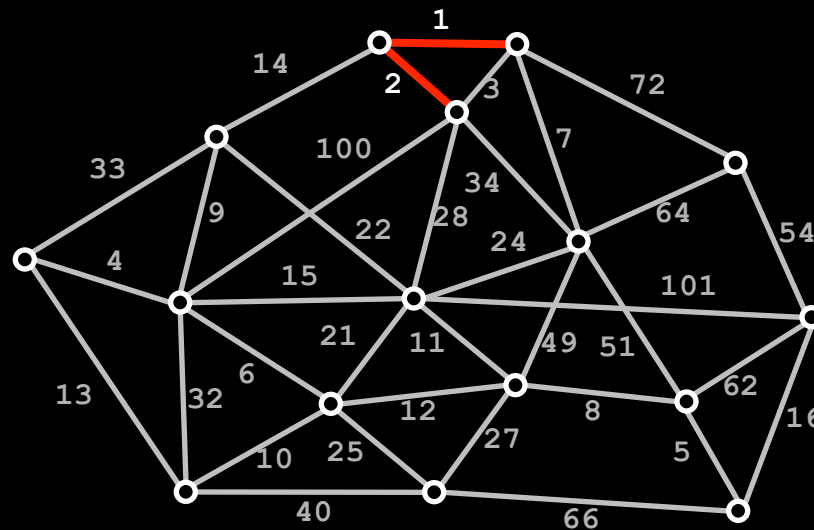
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

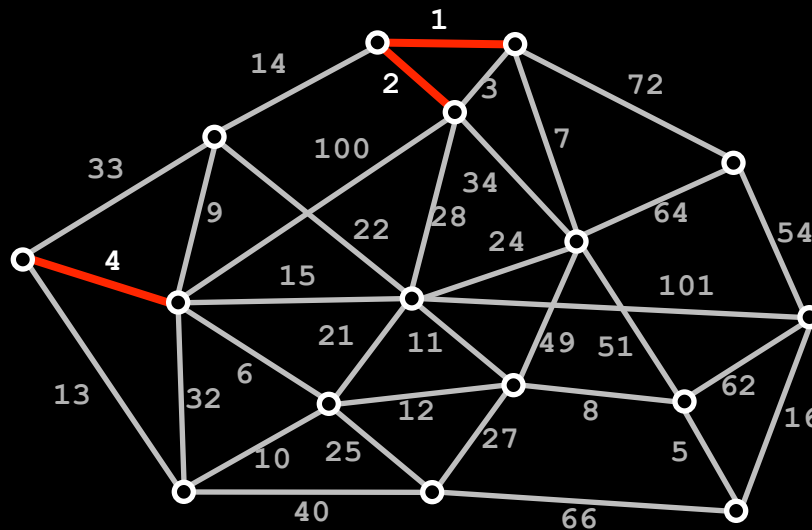
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

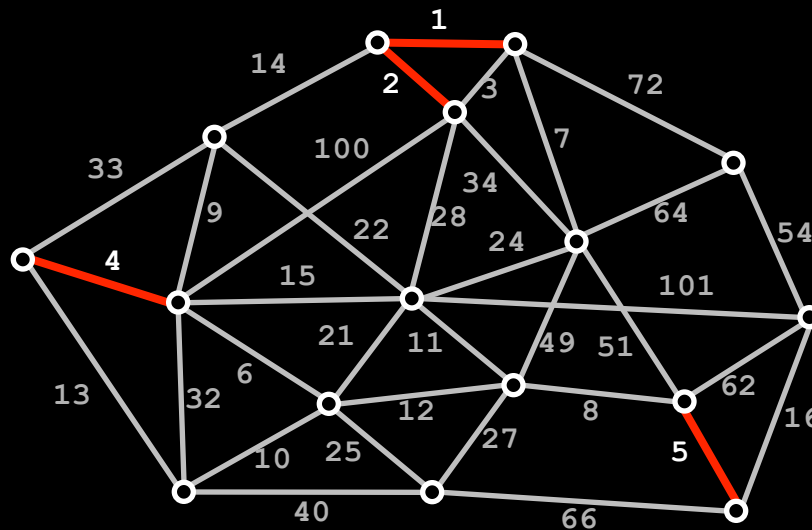
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

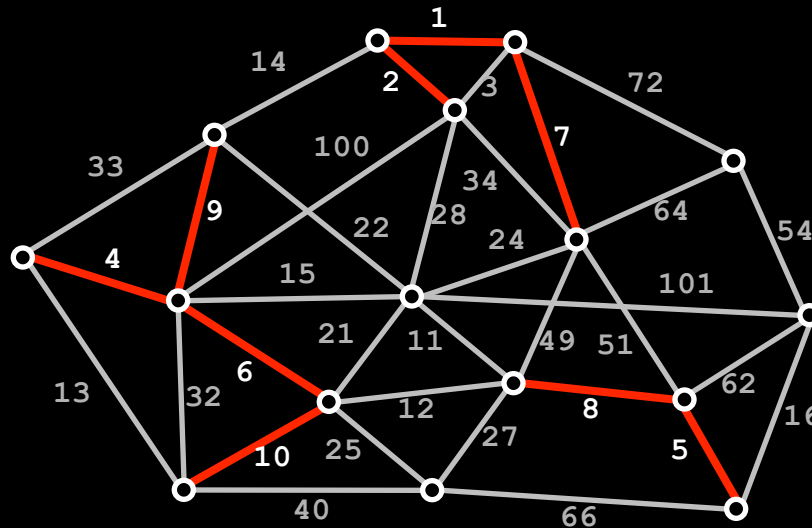
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

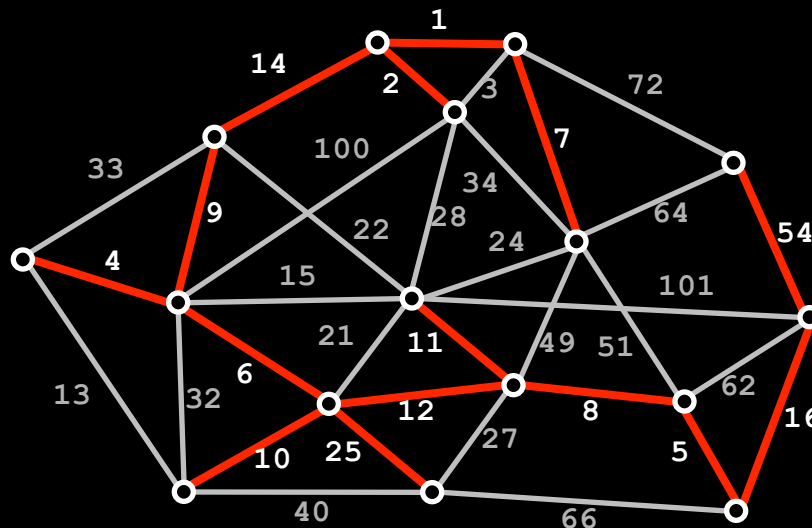
Kruskal's algorithm



3 Greedy Algorithms

- Algorithm B: Find a min weight edge – if it forms a cycle with edges already taken, throw it out, otherwise keep it

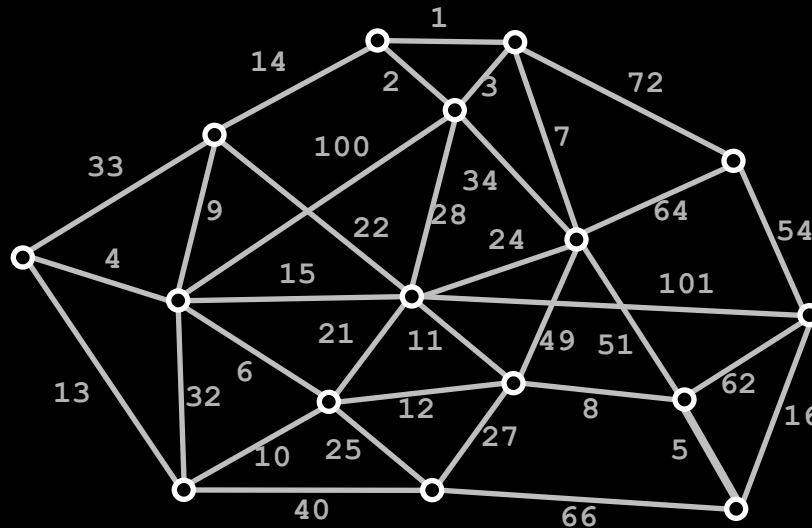
Kruskal's
algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

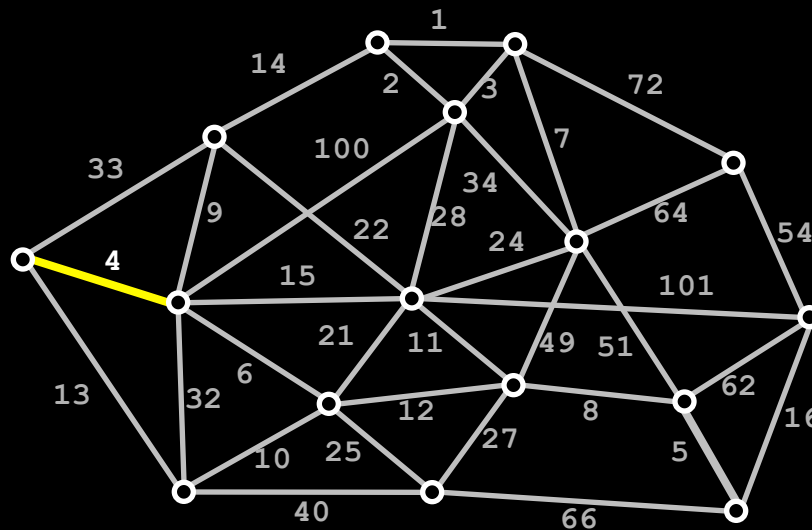
Prim's algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

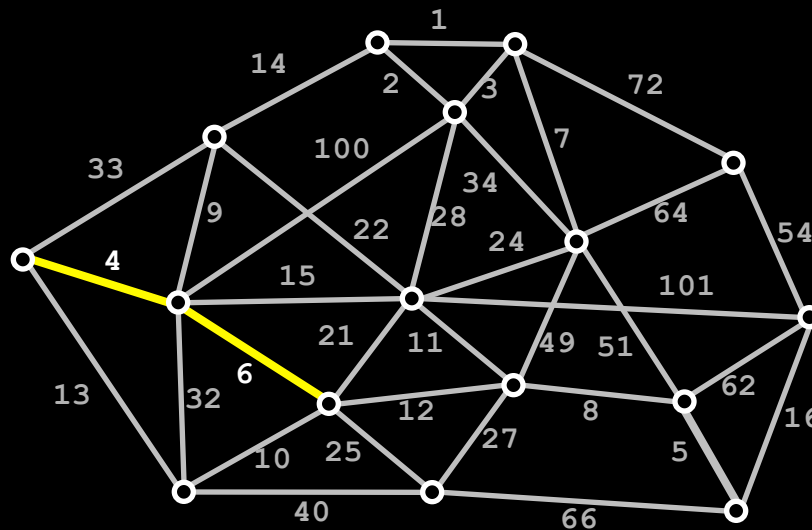
Prim's algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

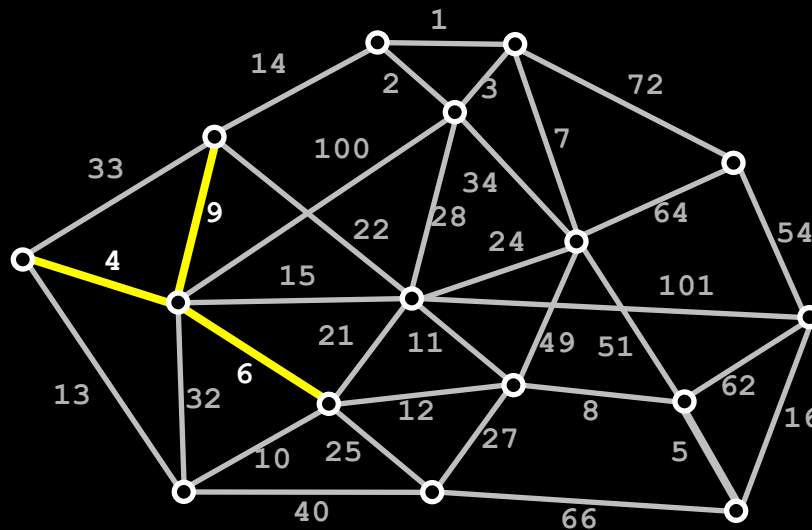
Prim's algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

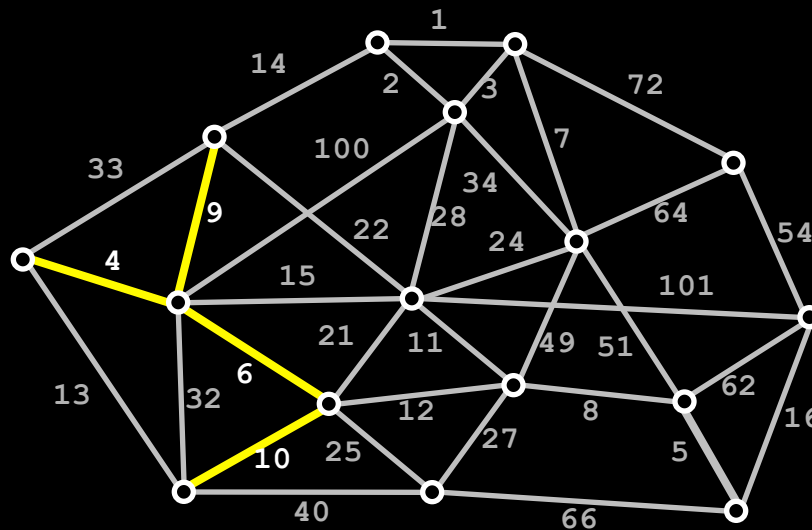
Prim's algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

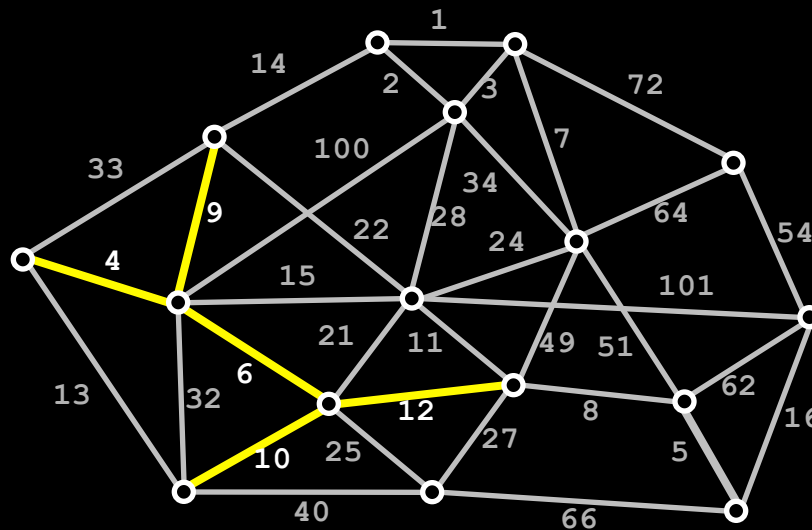
Prim's algorithm



3 Greedy Algorithms

- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

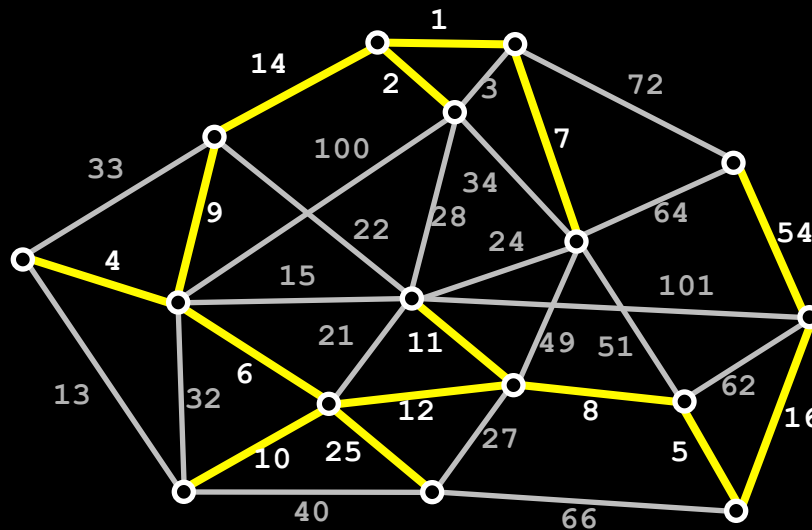
Prim's algorithm



3 Greedy Algorithms

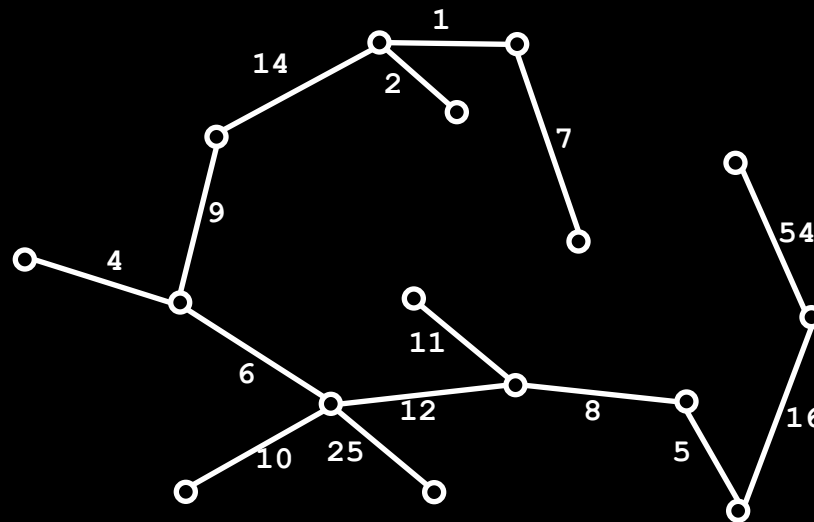
- Algorithm C: Start with any vertex, add min weight edge extending that connected component that does not form a cycle

Prim's algorithm

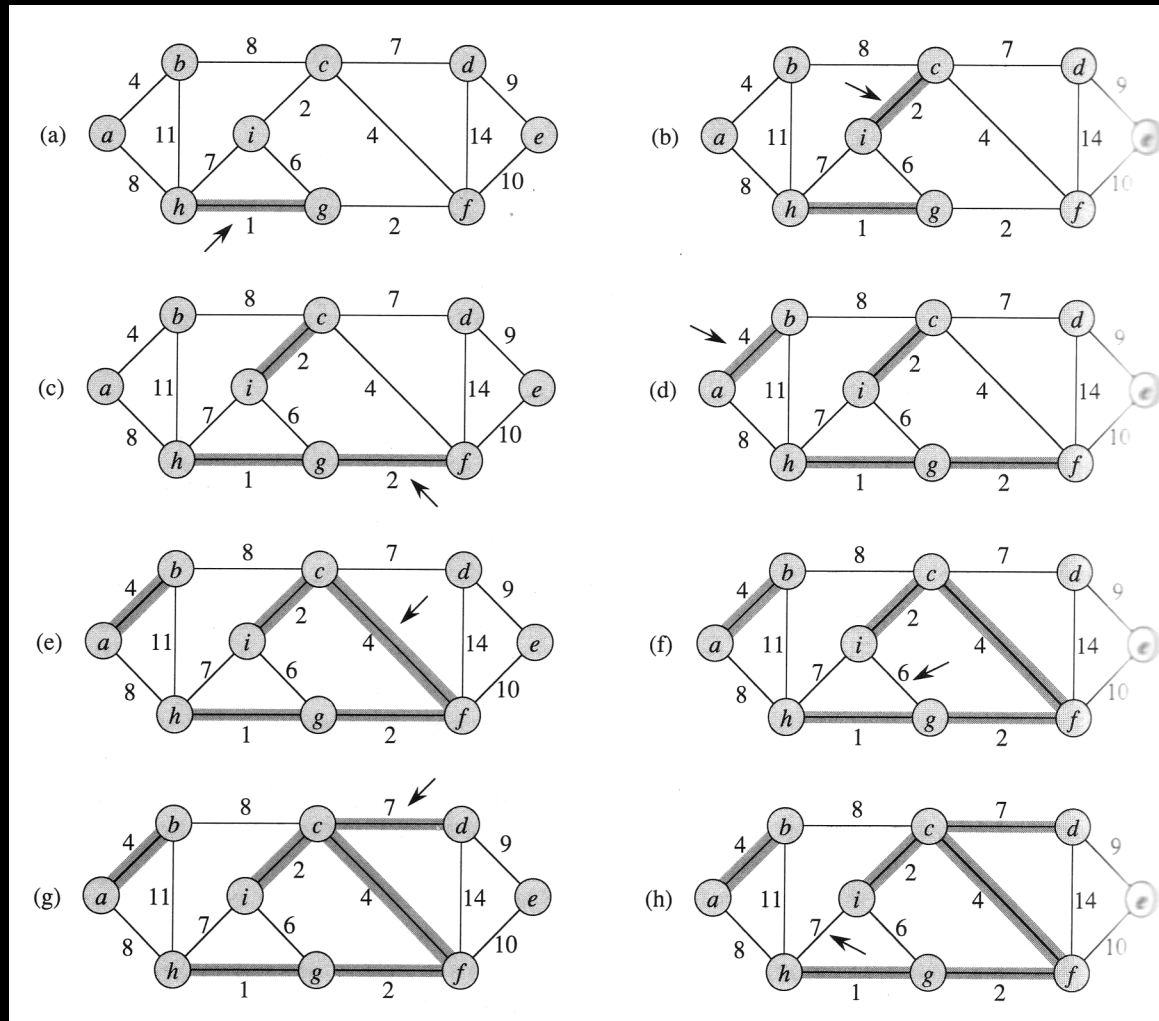


3 Greedy Algorithms

- All 3 greedy algorithms give the same minimum spanning tree (assuming distinct edge weights)

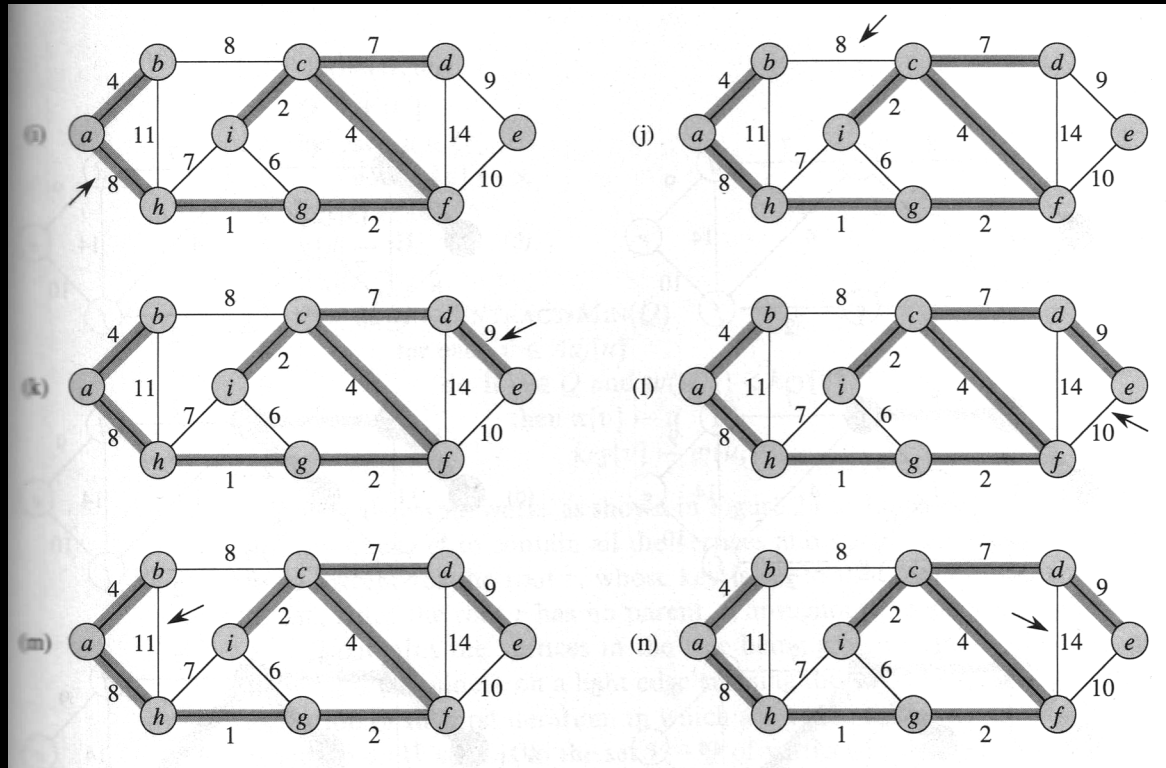


Example 2: Kruskal's algorithm



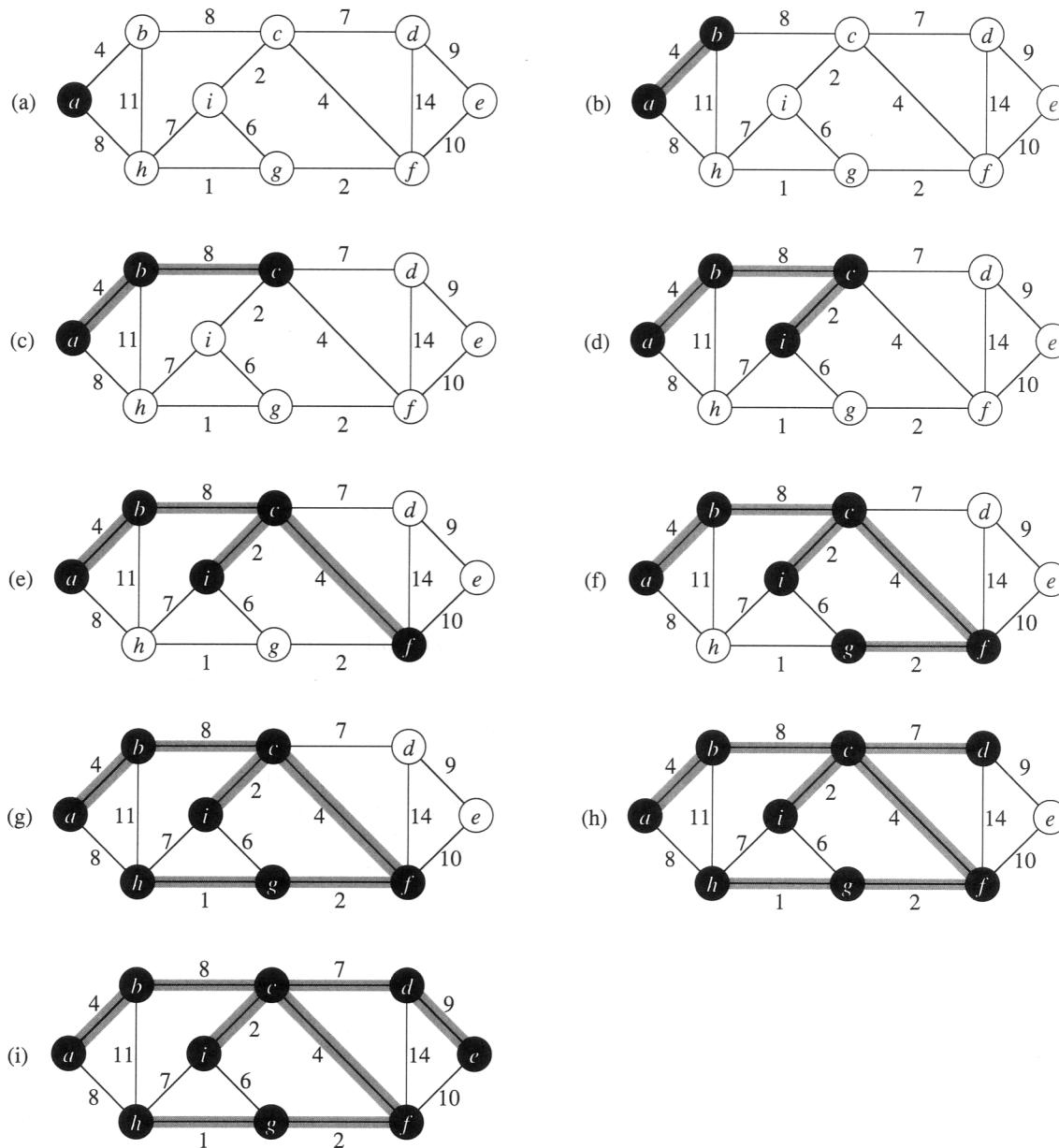
From "Introduction to Algorithms" by Cormen et al.

Example 2: Kruskal's algorithm



From "Introduction to Algorithms" by Cormen et al.

Example 2: Prim's algorithm



From "Introduction to Algorithms" by Cormen et al.

Prim's Algorithm

```
prim(s) {  
    D[t] = infty for all vertices t  
    D[s] = 0; //s is start vertex  
    while (some vertices are unmarked) {  
        x = unmarked vertex with smallest D;  
        mark x;  
        for (each y adj to x) {  
            D[y] = min(D[y], w(x,y));  
            // record pred[y] = x if changed  
        }  
    }  
}
```


Min "distance" to
connected component

- $O(n^2)$ for adj matrix
 - While-loop is executed n times
 - For-loop takes $O(n)$ time
- $O(m \log n)$ for adj list
 - Use a PQ. Can initialize in $O(n)$ time.
 - Regular PQ produces time $O(m \log n)$
 - Can improve to $O(m + n \log n)$ using a fancier heap
 - Still $m=O(n^2)$ if graph is not sparse

Prim's Algorithm

```
prim(s) {  
    D[t] = infty for all vertices t  
    D[s] = 0; //s is start vertex  
    pred[s] = null;  
    while (some vertices are unmarked) {  
        x = unmarked vertex with smallest D;  
        mark x;  
        for (each y adj to x) {  
            if( w(x,y) < D[y] ) {  
                D[y] = w(x,y);  
                pred[y] = x;  
            }  
        }  
    }  
}
```

Min "distance" to
connected component



Greedy Algorithms

- These are examples of **Greedy Algorithms**
- The Greedy Strategy is an algorithm design technique
 - Like Divide & Conquer
- Greedy algorithms are used to solve optimization problems
 - The goal is to find the best solution
- Works when the problem has the greedy-choice property
 - A global optimum can be reached by making locally optimum choices
- Example “Change Making”:
 - Given an amount of money, find the smallest number of coins to make that amount
- Solution: Greedy Algorithm
 - Give as many large coins as you can
 - This greedy strategy produces the optimum number of coins for the US coin system
- Different money system \Rightarrow greedy strategy may fail

Similar Code Structures

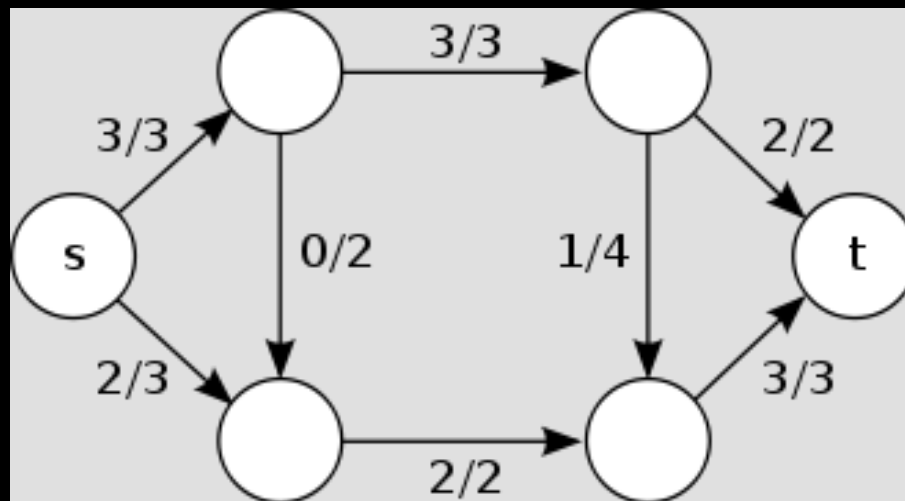
```
while (some vertices are
      unmarked) {
    x = best of unmarked
      vertices;
    mark x;
    for (each y adj to x)
      update y;
}
```

- BFS (unweighted)
 - best: next in queue
 - update: $D[y] = D[x] + 1$
- BFS (weighted) → Dijkstra
 - best: next in PQ
 - update: $D[y] = \min\{ D[y], D[x] + w(x,y) \}$
- Prim
 - best: next in PQ
 - update: $D[y] = \min\{ D[y], w(x,y) \}$

Other Graph Problems

Network Flow

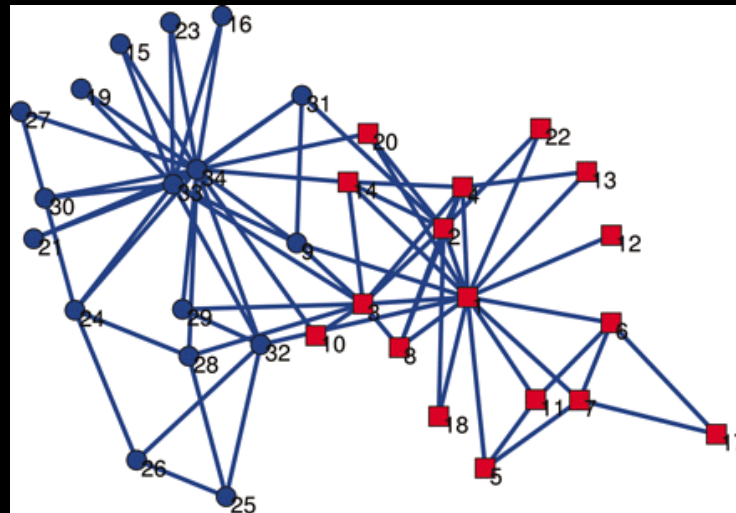
- How many “units” can flow from s to t ?
 - Flow in water network
 - Traffic flow



→ Ford-Fulkerson Algorithm

Minimum Cut

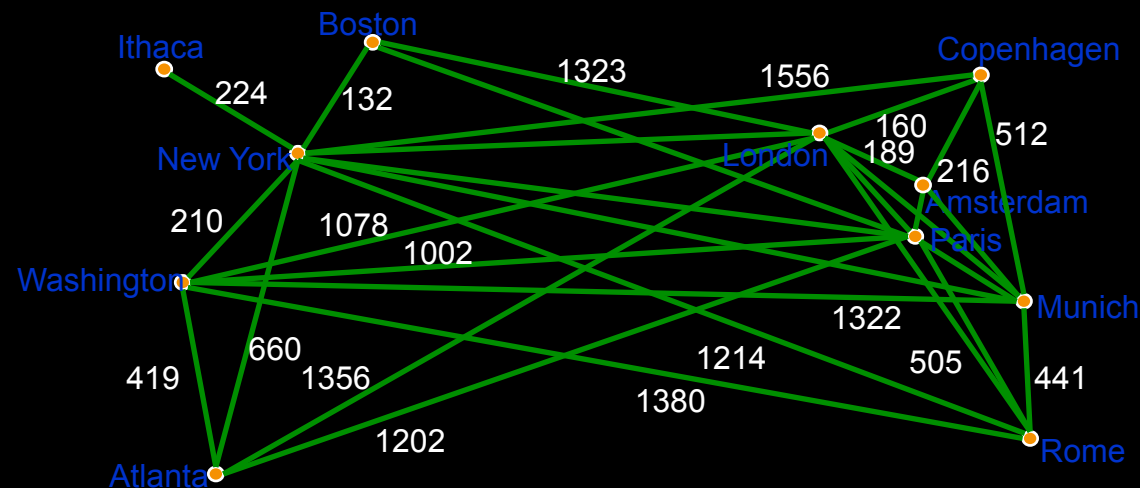
- Cut graph so that Source and Sink are separated, and the sum of the edges that are cut is minimized.
 - Traffic bottlenecks
 - Clustering in social networks



→ Duality with Maximum Flow

Traveling Salesperson

- Find a path of minimum distance that visits every city.
 - Planning and logistics
 - Microchip design



- NP-Hard \rightarrow there is probably no $O(n^k)$ algorithms