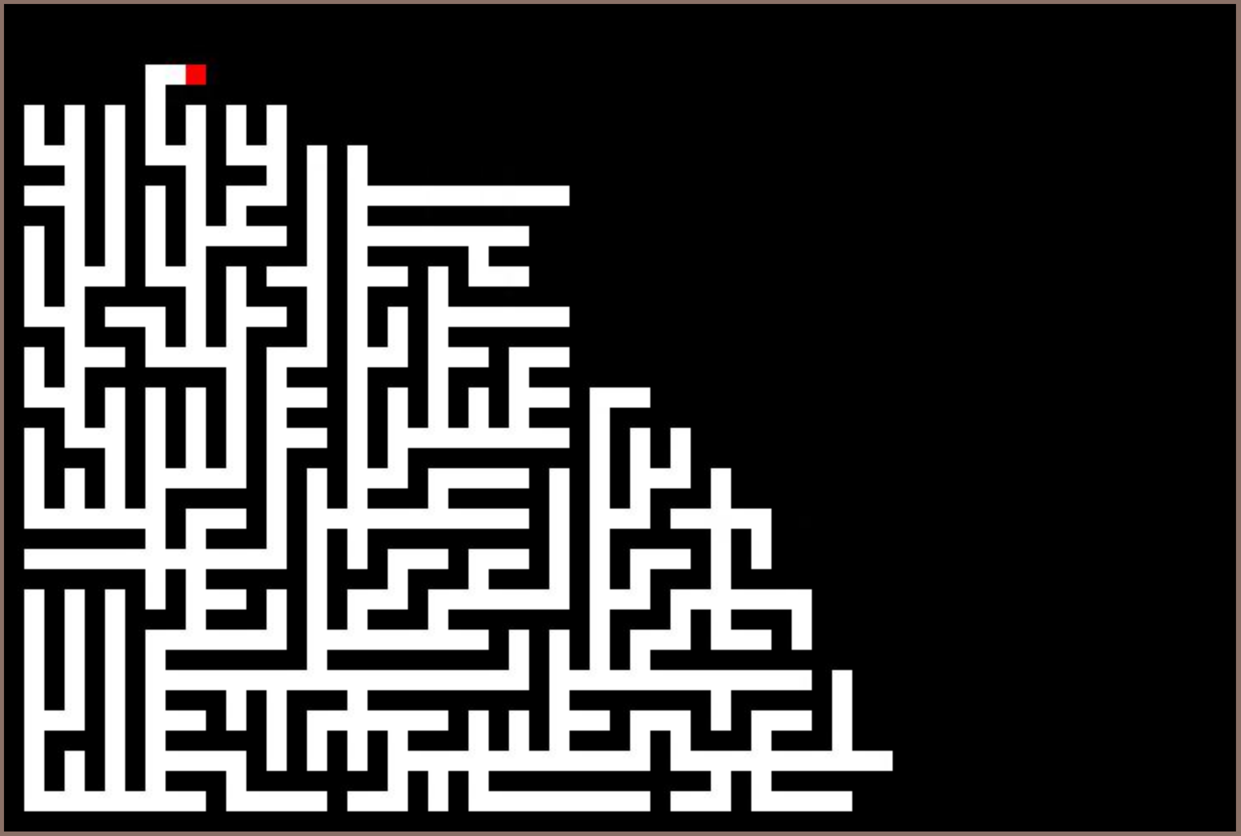


SPANNING TREES

Lecture 21

CS2110 – Fall 2015

Spanning trees

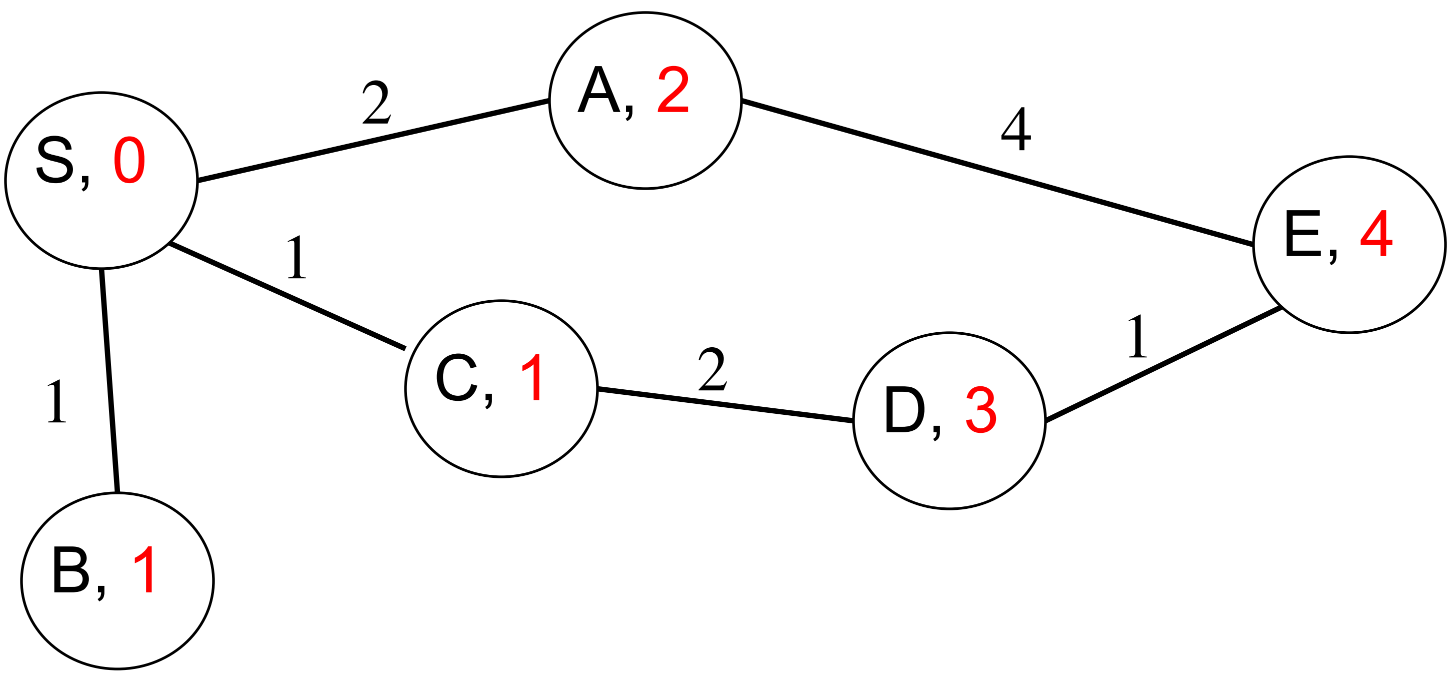
- Calculating the shortest path in Dijkstra's algorithm
- Definitions
- Minimum spanning trees
- 3 greedy algorithms (including Kruskal & Prim)
- Concluding comments:
  - Greedy algorithms
  - Travelling salesman problem

Dijkstra's algorithm using Nodes.

An object of class Node for each node of the graph.

Nodes have an identification, (S, A, E, etc).

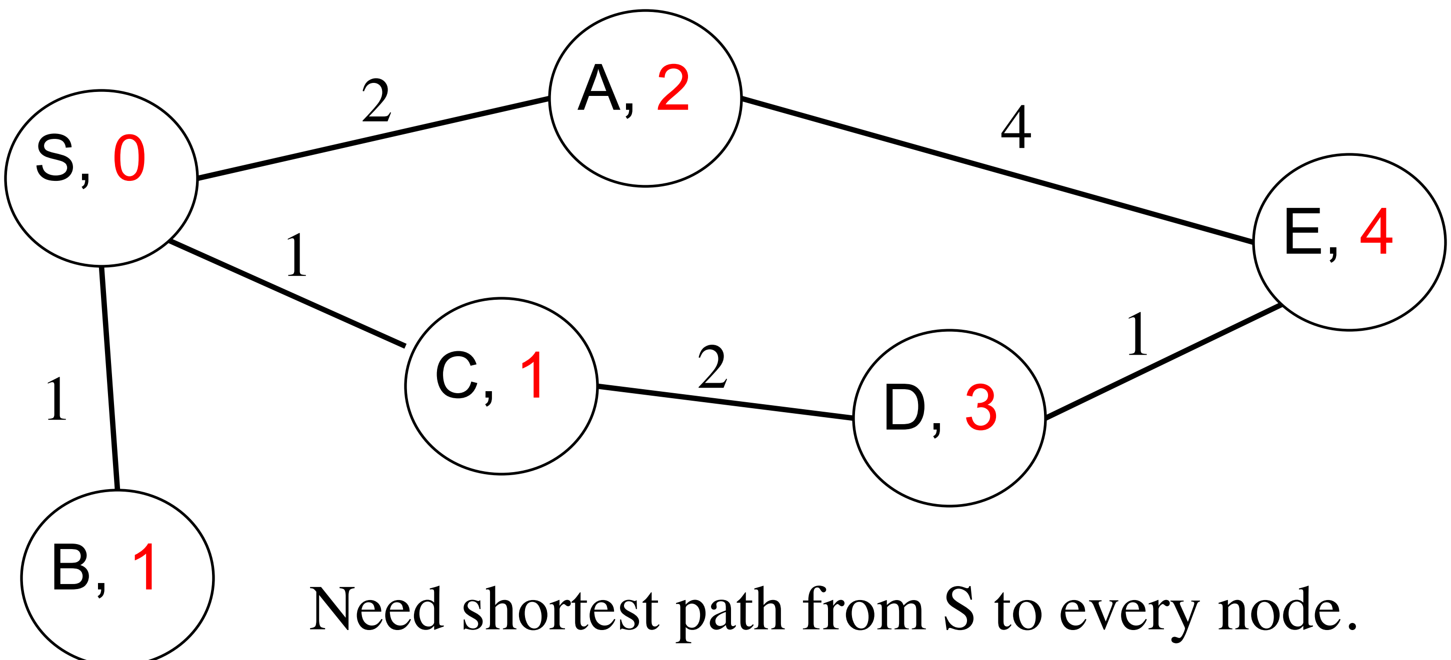
Nodes contain shortest distance from Start node (red).



Backpointers

Shortest path requires not only the distance from start to a node but the shortest path itself. How to do that?

In the graph, red numbers are shortest distance from S.

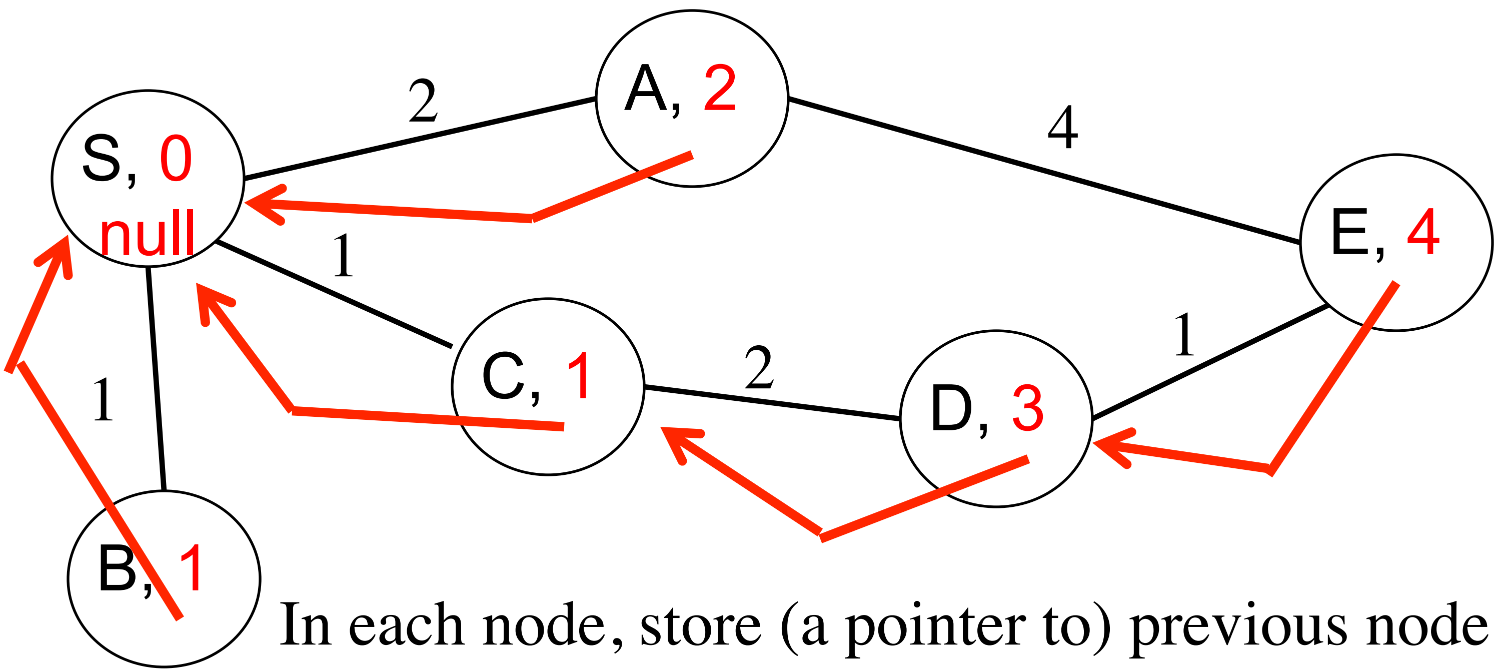


Need shortest path from S to every node.
Storing that info in node S wouldn't make sense.

Backpointers

Shortest path requires not only the distance from start to a node but the shortest path itself. How to do that?

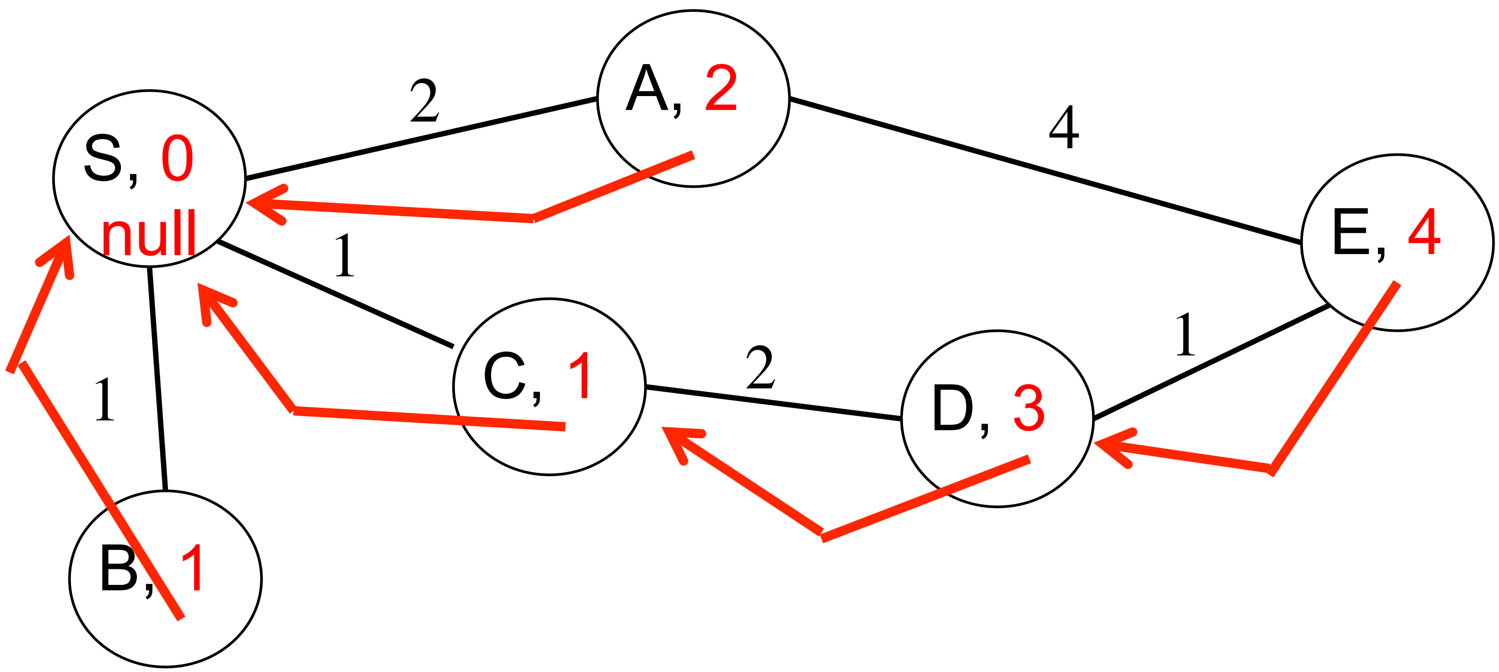
In the graph, red numbers are shortest distance from S.



In each node, store (a pointer to) previous node on the shortest path from S to that node. **Backpointer**

Backpointers

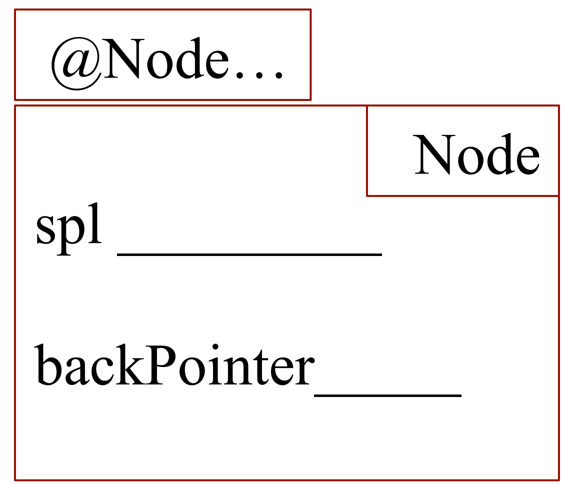
When to set a backpointer? In the algorithm, processing an edge (f, w): If the shortest distance to w changes, then set w's backpointer to f. It's that easy!



Each iteration of Dijkstra's algorithm

spl: shortest-path length calculated so far

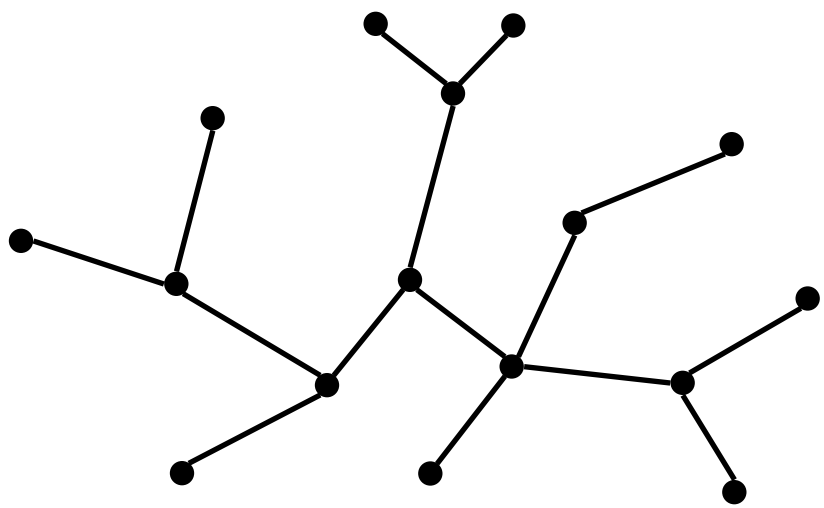

```
f= node in Frontier with min spl; Remove f from Frontier;
for each neighbor w of f:
    if w in far-off set
    then    w.spl=  f.spl + weight(f, w);
            Put w in the Frontier;
            w.backPointer=  f;
    else if f.spl + weight(f, w) < w.spl
         then   w.spl=  f.spl + weight(f, w)
                w.backPointer=  f;
```



Undirected trees

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

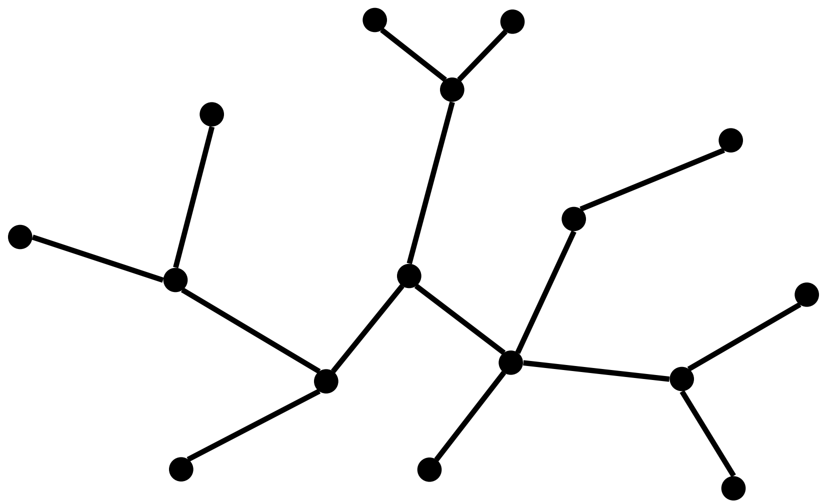
Root of tree?
It doesn't matter. Choose any vertex for the root



Facts about trees

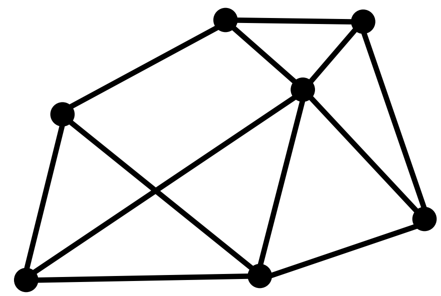
- $|E| = |V| - 1$
- connected
- no cycles

Any two of these properties imply the third, and imply that the graph is a tree

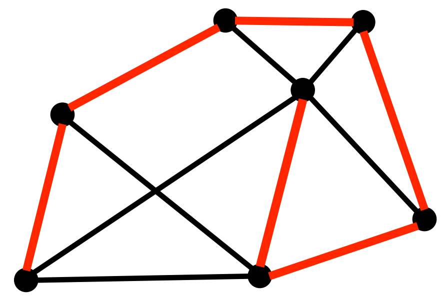


A ***spanning tree*** of a **connected undirected** graph (V, E) is a subgraph (V, E') that is a tree

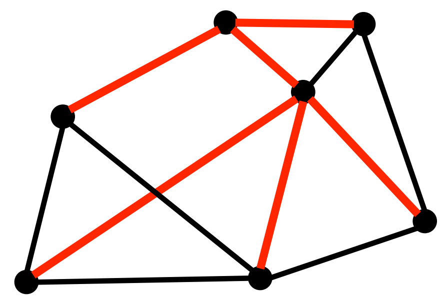
- Same set of vertices V
- E' ⊆ E
- (V, E') is a tree



- Same set of vertices V
- Maximal set of edges that contains no cycle

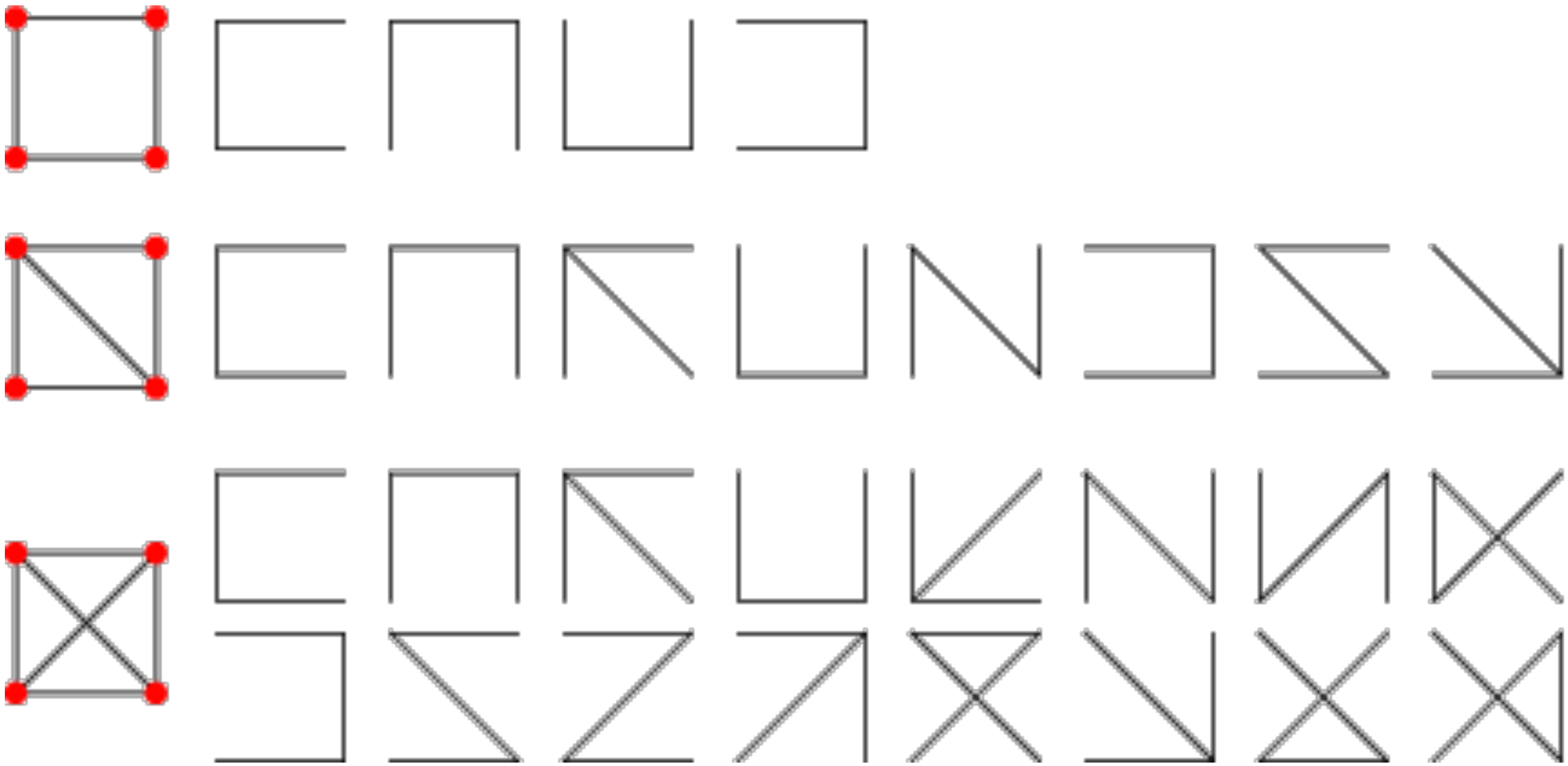


- Same set of vertices V
- Minimal set of edges that connect all vertices



Three equivalent definitions

Spanning trees: examples



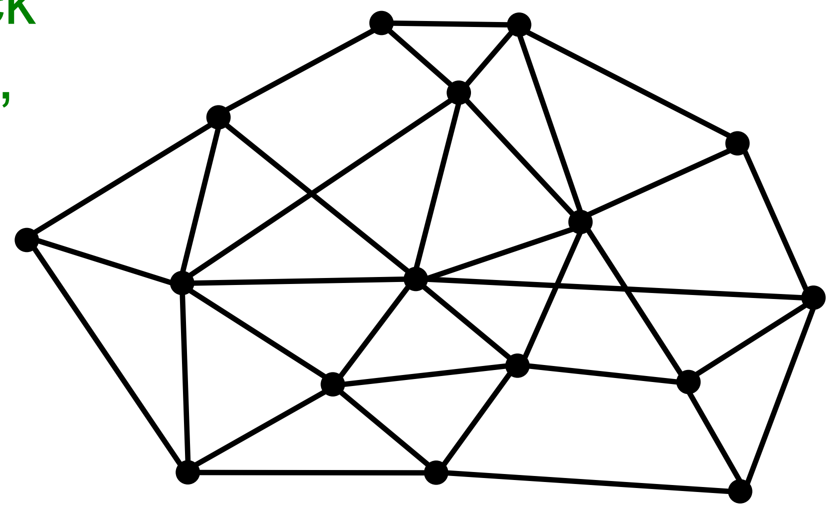
http://mathworld.wolfram.com/SpanningTree.html

Finding a spanning tree

Maximal set of edges that contains no cycle

A subtractive method

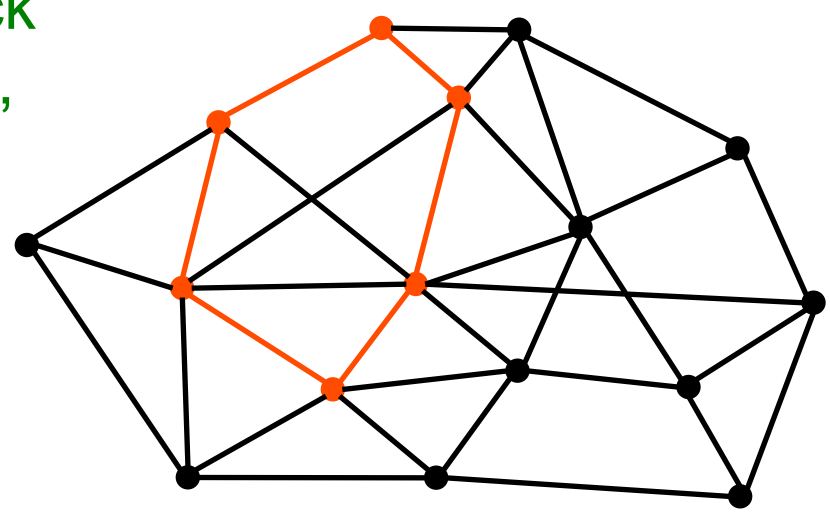
- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles



Finding a spanning tree

A subtractive method

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles

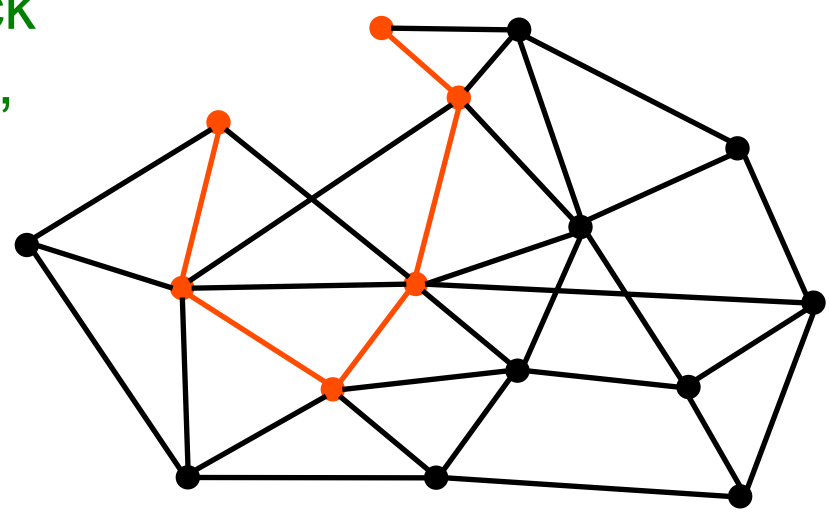


Finding a spanning tree

A subtractive method

nondeterministic algorithm

- Start with the whole graph – it is connected
- If there is a cycle, pick an edge on the cycle, throw it out – the graph is still connected (why?)
- Repeat until no more cycles



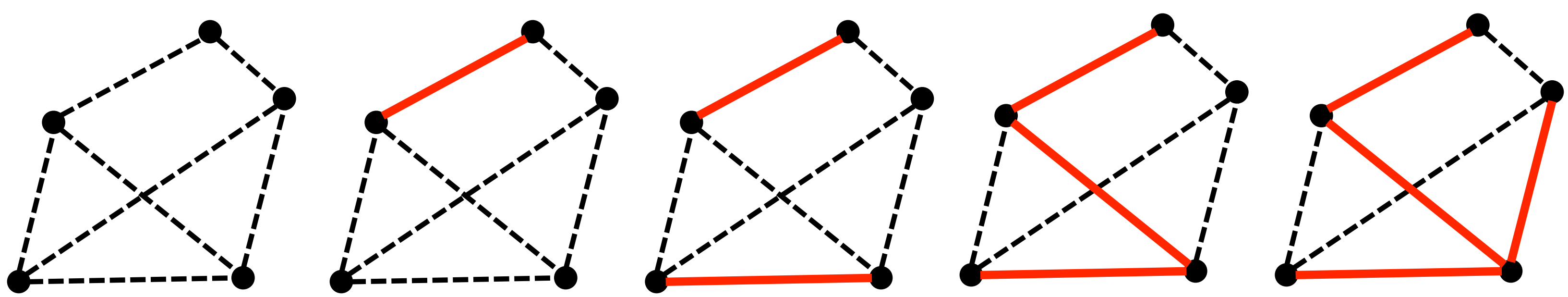
Finding a spanning tree: Additive method

- Start with no edges
- While the graph is not connected:
  Choose an edge that connects 2 connected components and add it – the graph still has no cycle (why?)

Minimal set of edges that connect all vertices

nondeterministic algorithm

Tree edges will be red.
Dashed lines show original edges.
Left tree consists of 5 connected components, each a node



Minimum spanning trees

- Suppose edges are weighted (> 0), and we want a spanning tree of *minimum cost* (sum of edge weights)

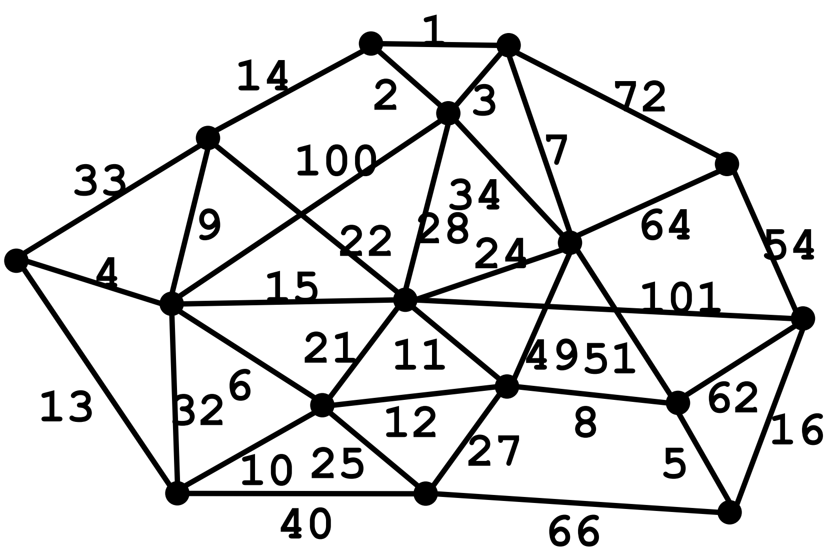
- Some graphs have exactly one minimum spanning tree. Others have several trees with the same cost, any of which is a minimum spanning tree

Minimum spanning trees

- Suppose edges are weighted (> 0), and we want a spanning tree of *minimum cost* (sum of edge weights)

- Useful in network routing & other applications

- For example, to stream a video



3 Greedy algorithm

A greedy algorithm: follow the heuristic of making a locally optimal choice at each stage, with the hope of finding a global optimum

Example. Make change using the fewest number of coins.
Make change for n cents, n < 100 (i.e. < $1)
Greedy: At each step, choose the largest possible coin

If n >= 50 choose a half dollar and reduce n by 50;
If n >= 25 choose a quarter and reduce n by 25;
As long as n >= 10, choose a dime and reduce n by 10;
If n >= 5, choose a nickel and reduce n by 5;
Choose n pennies.

3 Greedy algorithm

A greedy algorithm: follow the heuristic of making a locally optimal choice at each stage, with the hope of fining a global optimum. Doesn't always work

Example. Make change using the fewest number of coins.
Coins have these values: 7, 5, 1
Greedy: At each step, choose the largest possible coin

Consider making change for 10.
The greedy choice would choose: 7, 1, 1, 1.
But 5, 5 is only 2 coins.

3 Greedy algorithm

A greedy algorithm: follow the heuristic of making a locally optimal choice at each stage, with the hope of fining a global optimum. Doesn't always work

Example. Make change (if possible) using the fewest number of coins.

Coins have these values: 7, 5, 2

Greedy: At each step, choose the largest possible coin

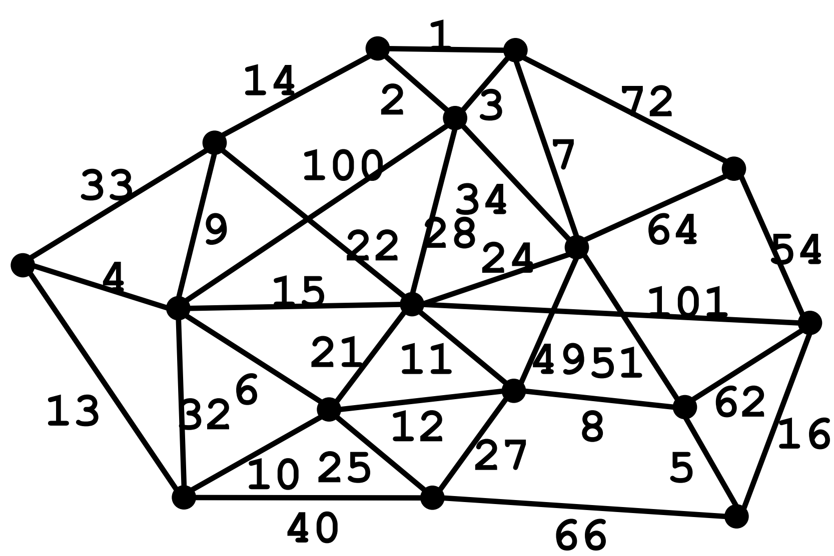
Consider making change for 10.

The greedy choice would choose: 7, 2 –and can't proceed!

But 5, 5 works

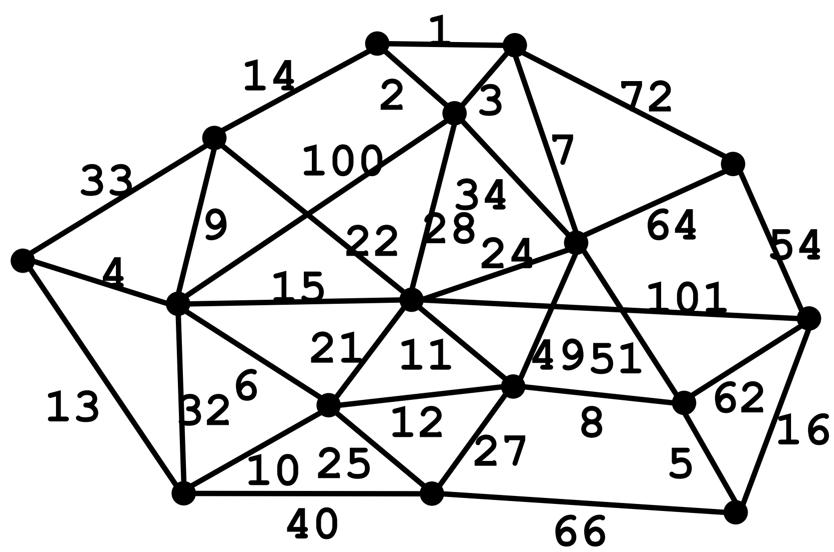
3 Greedy algorithms

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



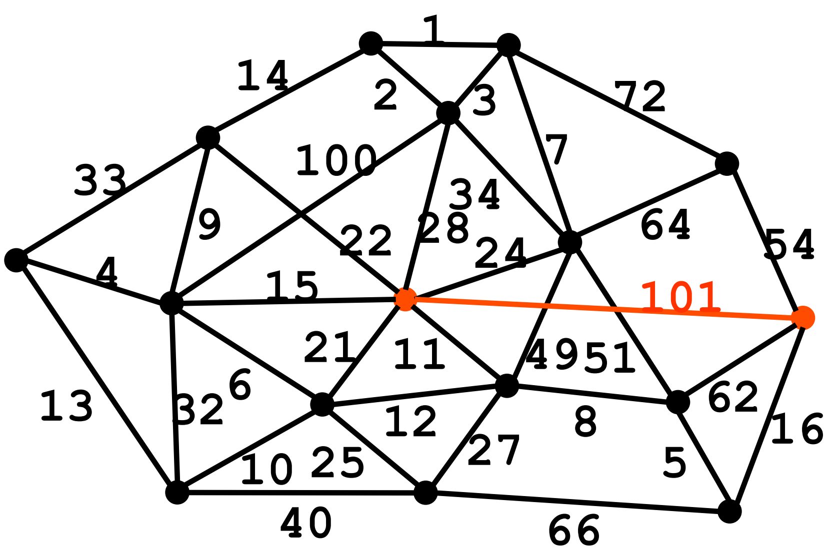
3 Greedy algorithms

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



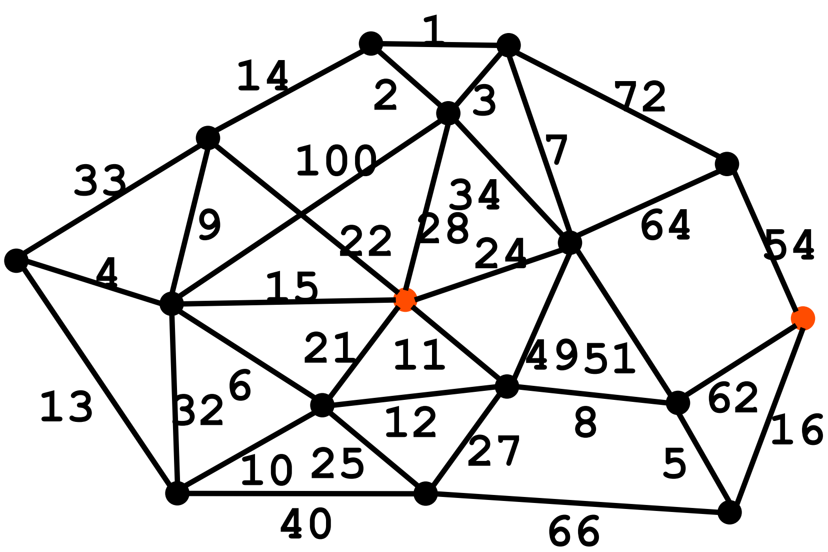
3 Greedy algorithms

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



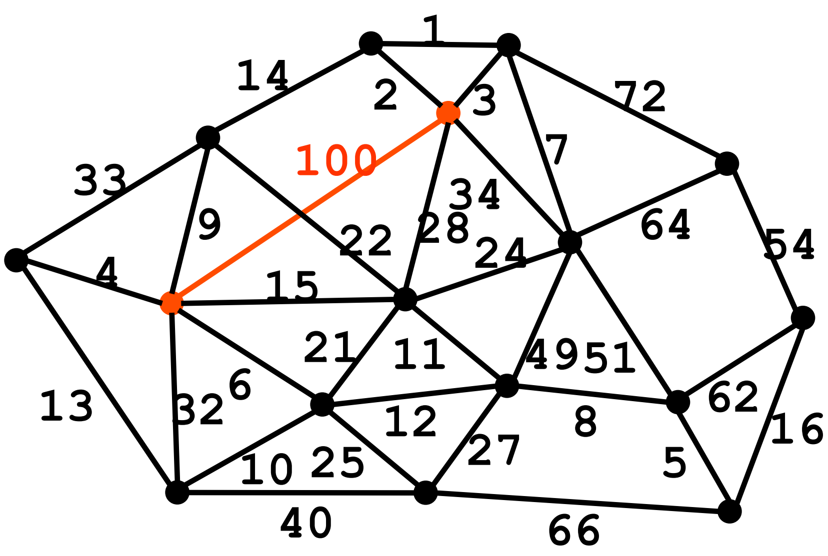
3 Greedy algorithms

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



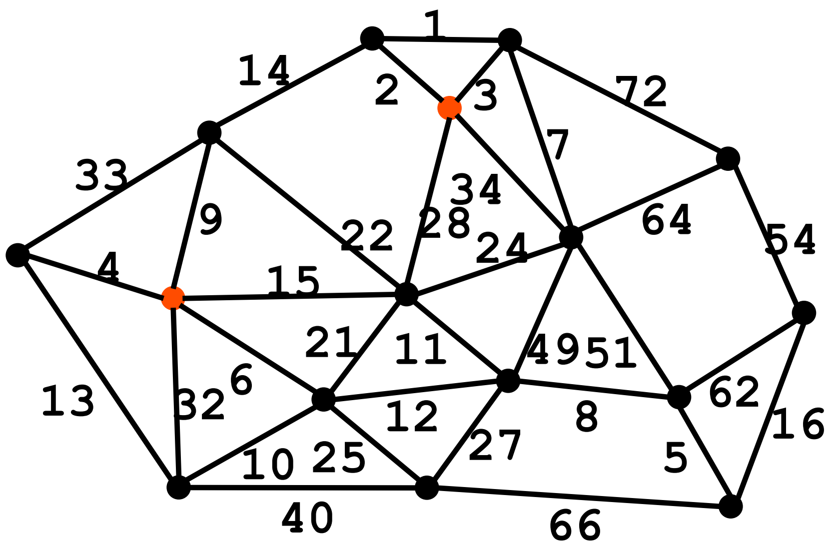
3 Greedy algorithms

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



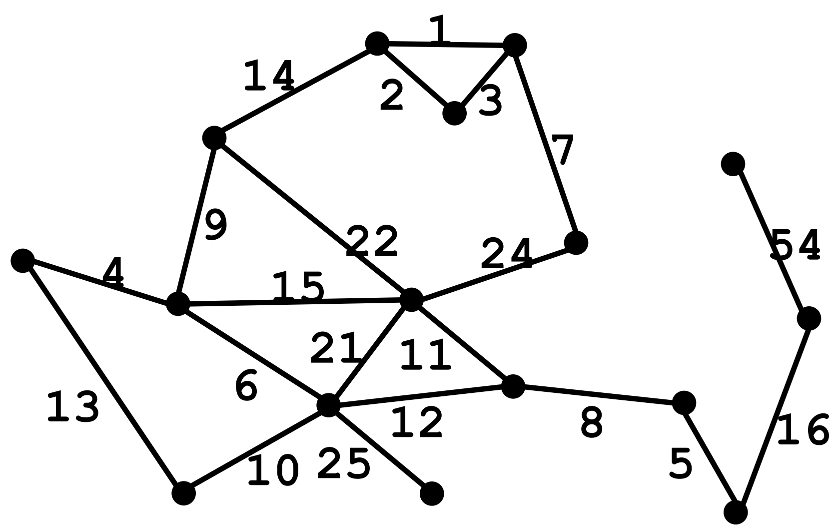
3 Greedy algorithms

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



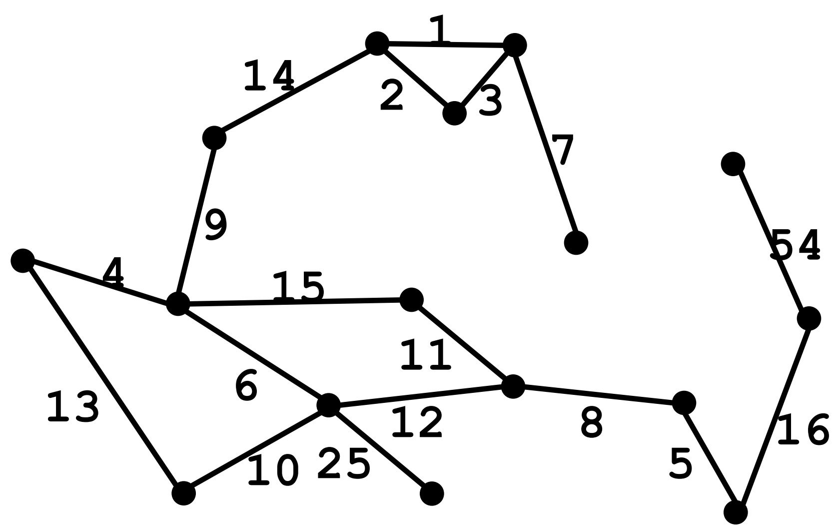
3 Greedy algorithms

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



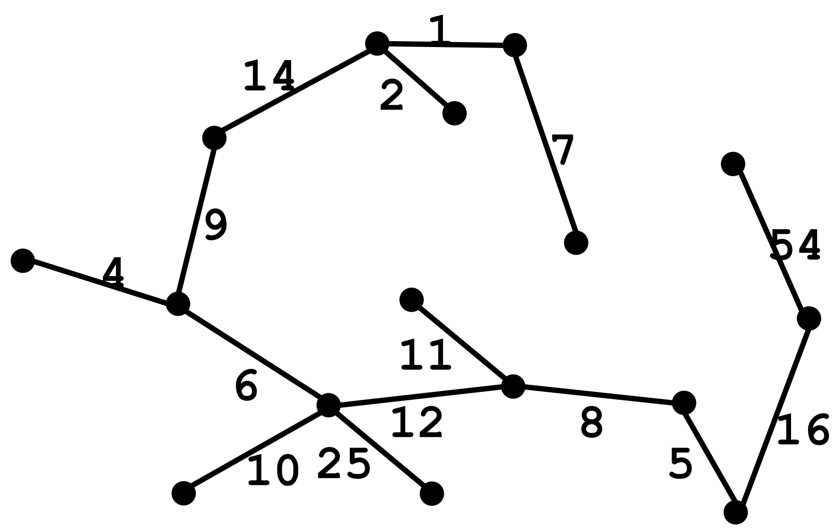
3 Greedy algorithms

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



3 Greedy algorithms

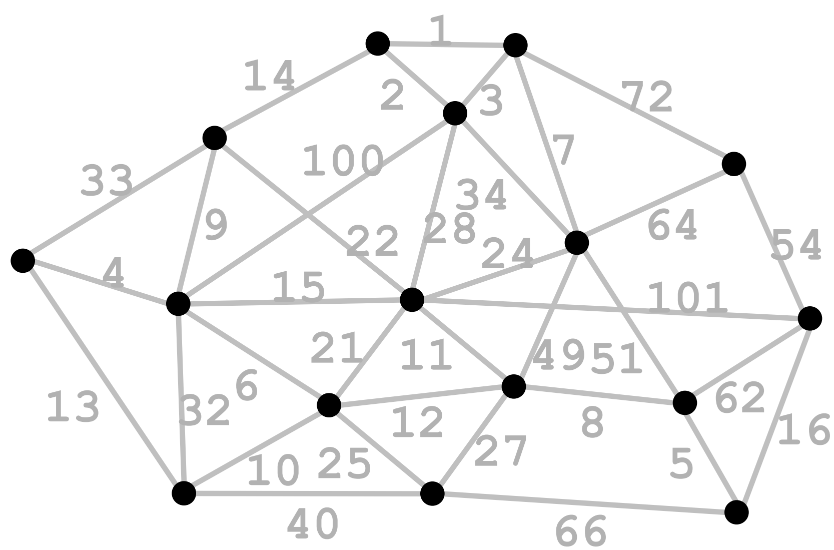
A. Find a max weight edge – if it is on a cycle, throw it out, otherwise keep it



3 Greedy algorithms: Kruskal

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

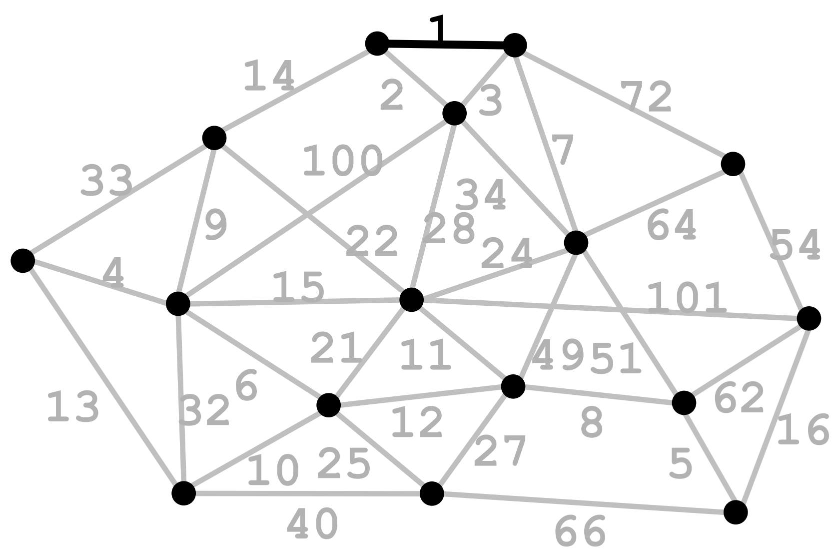
Kruskal's algorithm



3 Greedy algorithms: Kruskal

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

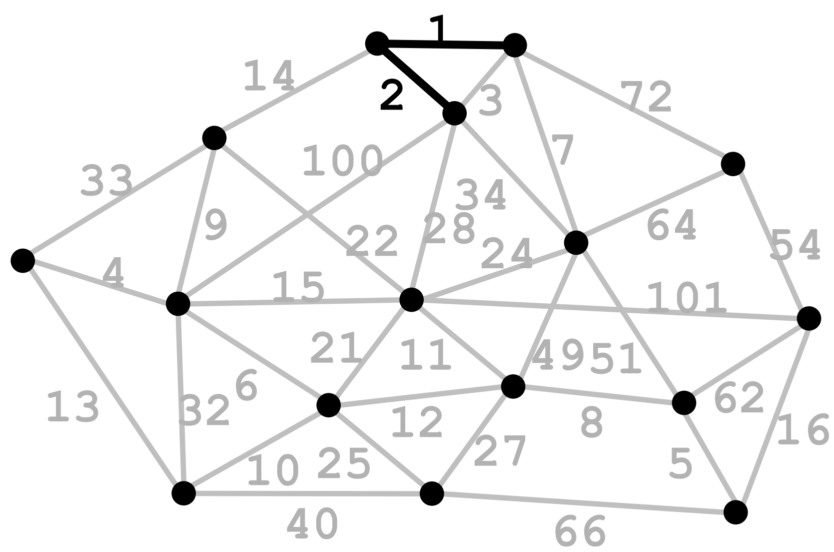
Kruskal's algorithm



3 Greedy algorithms: Kruskal

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

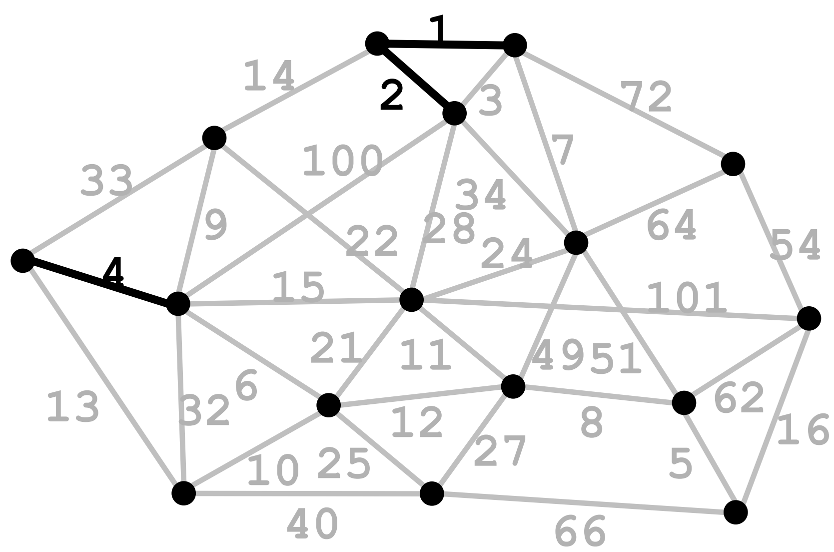
Kruskal's algorithm



3 Greedy algorithms: Kruskal

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

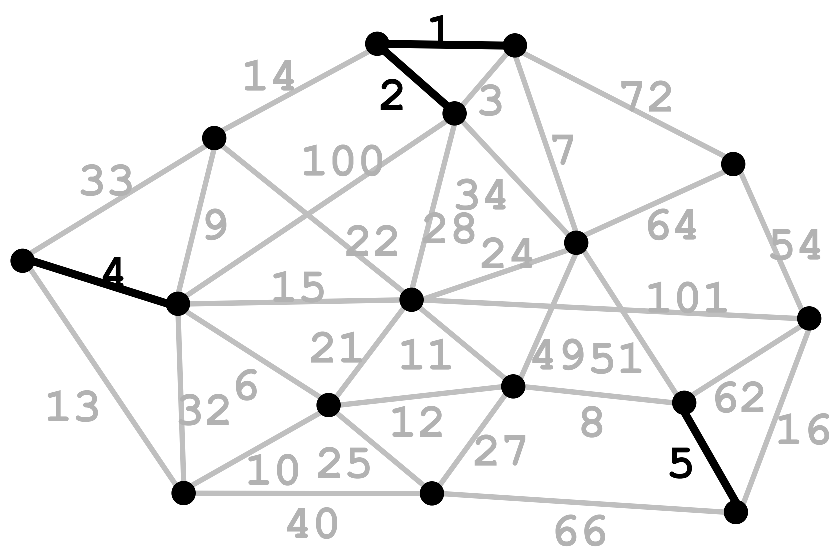
Kruskal's algorithm



3 Greedy algorithms: Kruskal

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

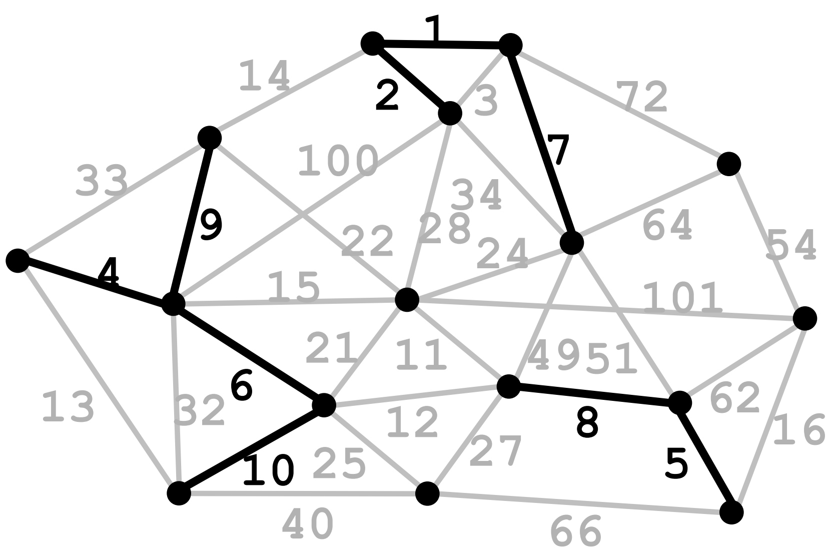
Kruskal's algorithm



3 Greedy Algorithms: Kruskal

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

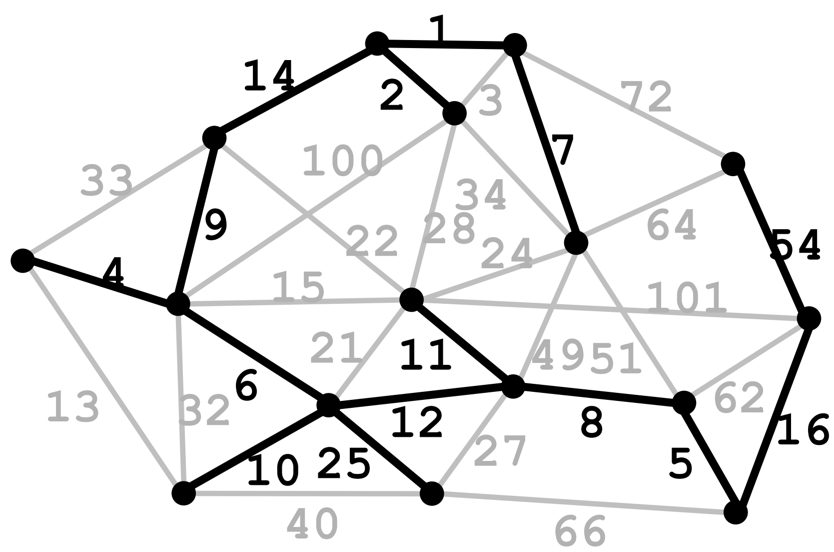
Kruskal's algorithm



3 Greedy Algorithms: Kruskal

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

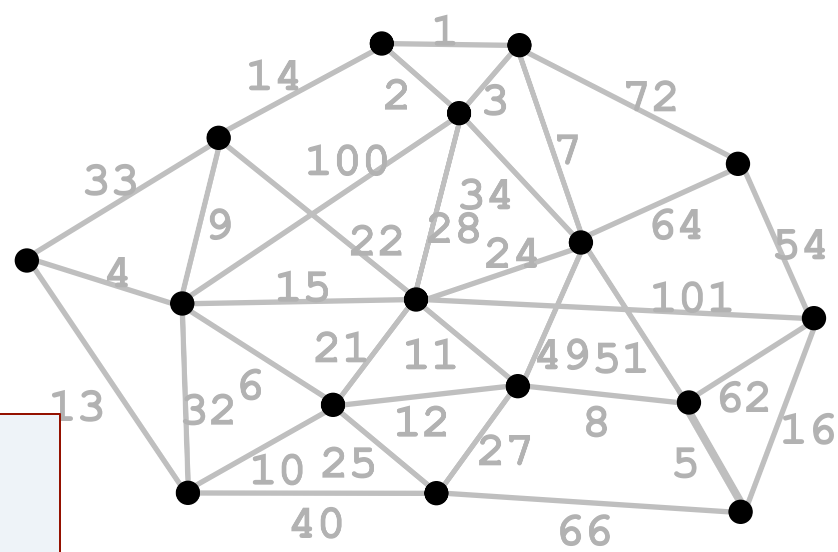
Kruskal's algorithm



3 Greedy algorithms: Prim

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

Prim's algorithm
(reminiscent of Dijkstra's algorithm)



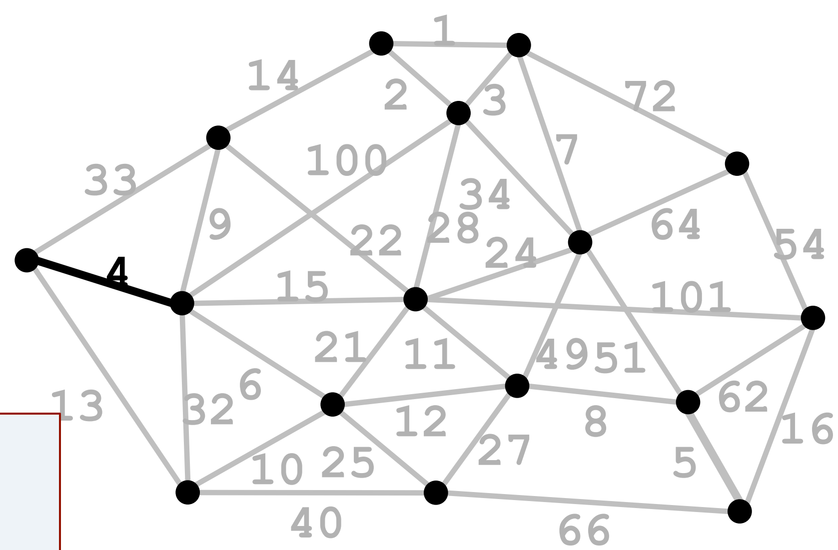
Invariant: the added edges must form a tree

3 Greedy algorithms: Prim

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

Prim's algorithm
(reminiscent of Dijkstra's algorithm)

Invariant: the added edges must form a tree

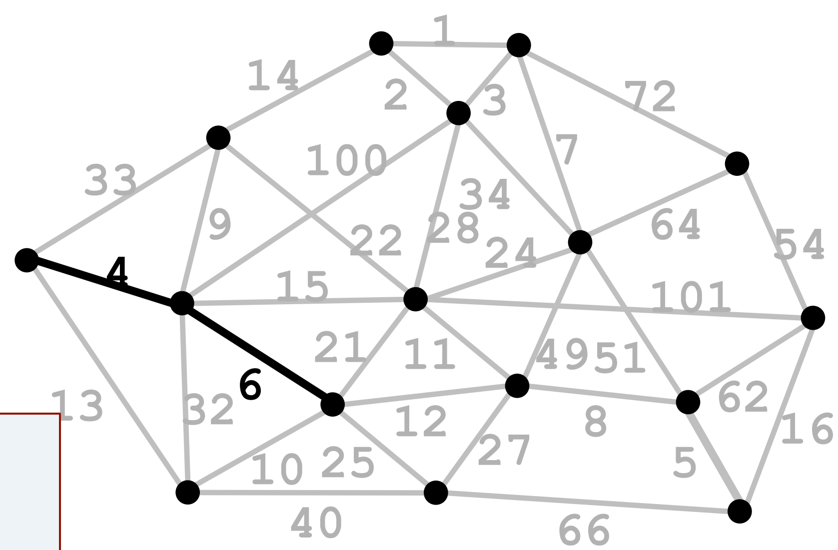


3 Greedy algorithms: Prim

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

Prim's algorithm
(reminiscent of Dijkstra's algorithm)

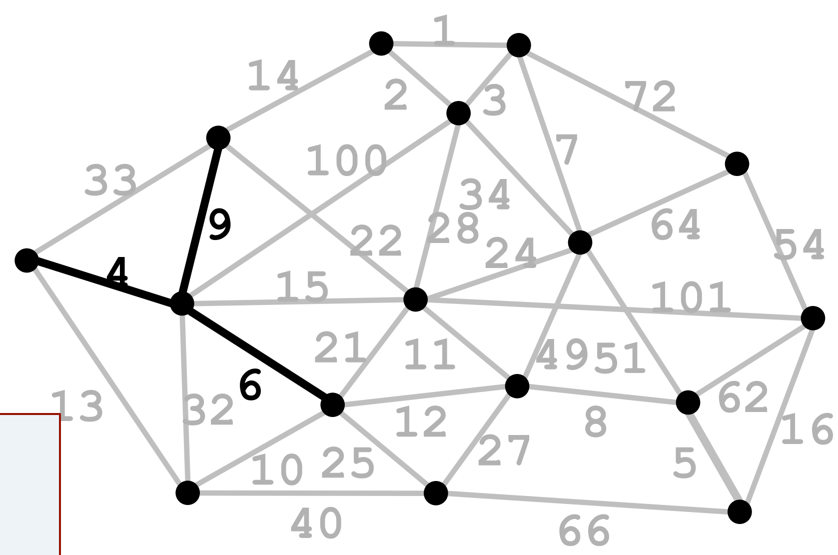
Invariant: the added edges must form a tree



3 Greedy algorithms: Prim

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

Prim's algorithm
(reminiscent of Dijkstra's algorithm)



Invariant: the added edges must form a tree

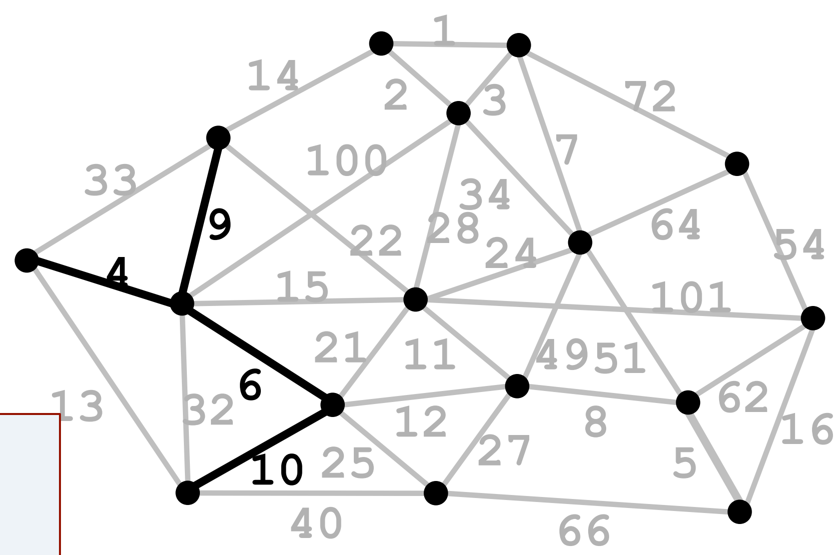
3 Greedy algorithms: Prim

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

Prim's algorithm

(reminiscent of Dijkstra's algorithm)

Invariant: the added edges must form a tree

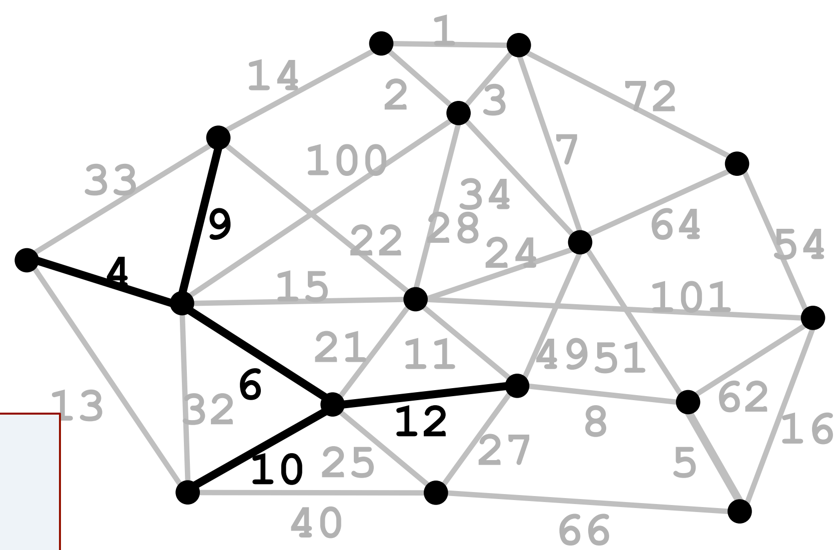


3 Greedy algorithms: Prim

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

Prim's algorithm
(reminiscent of Dijkstra's algorithm)

Invariant: the added edges must form a tree

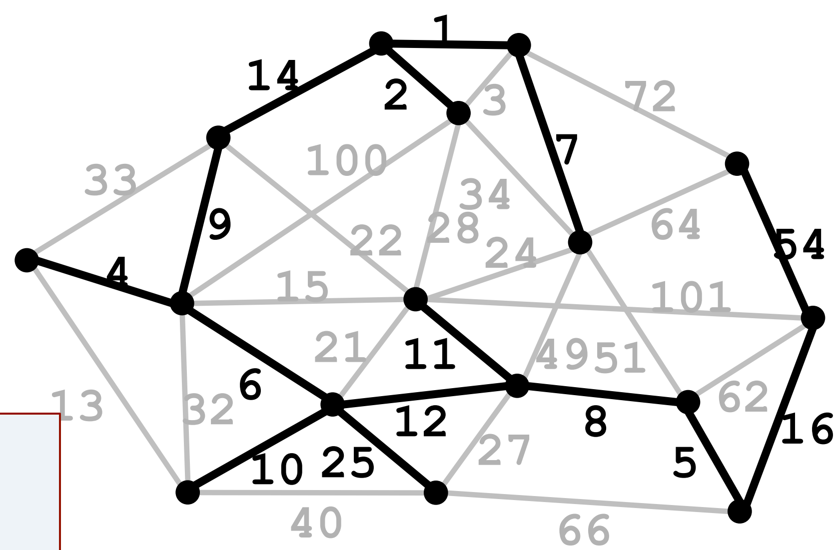


3 Greedy algorithms: Prim

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

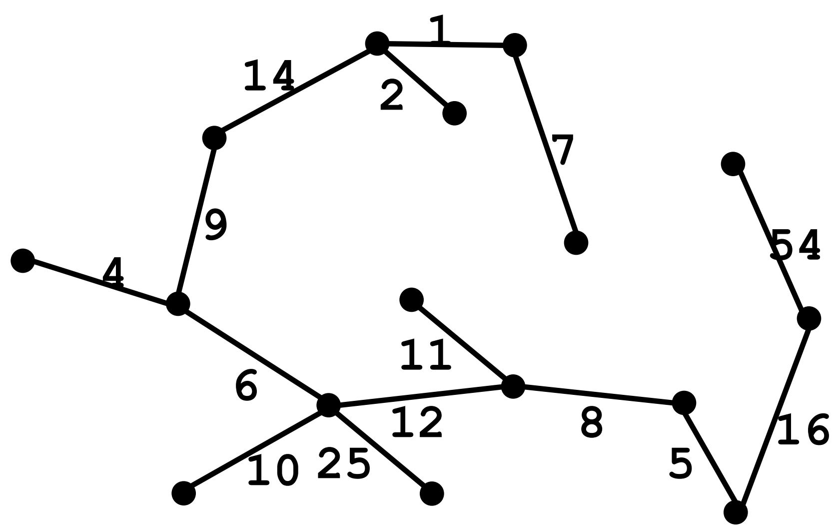
Prim's algorithm
(reminiscent of Dijkstra's algorithm)

Invariant: the added edges must form a tree



3 Greedy algorithms: Prim

When edge weights are all distinct, or if there is exactly one minimum spanning tree, the 3 algorithms all find the identical tree



Prim's algorithm

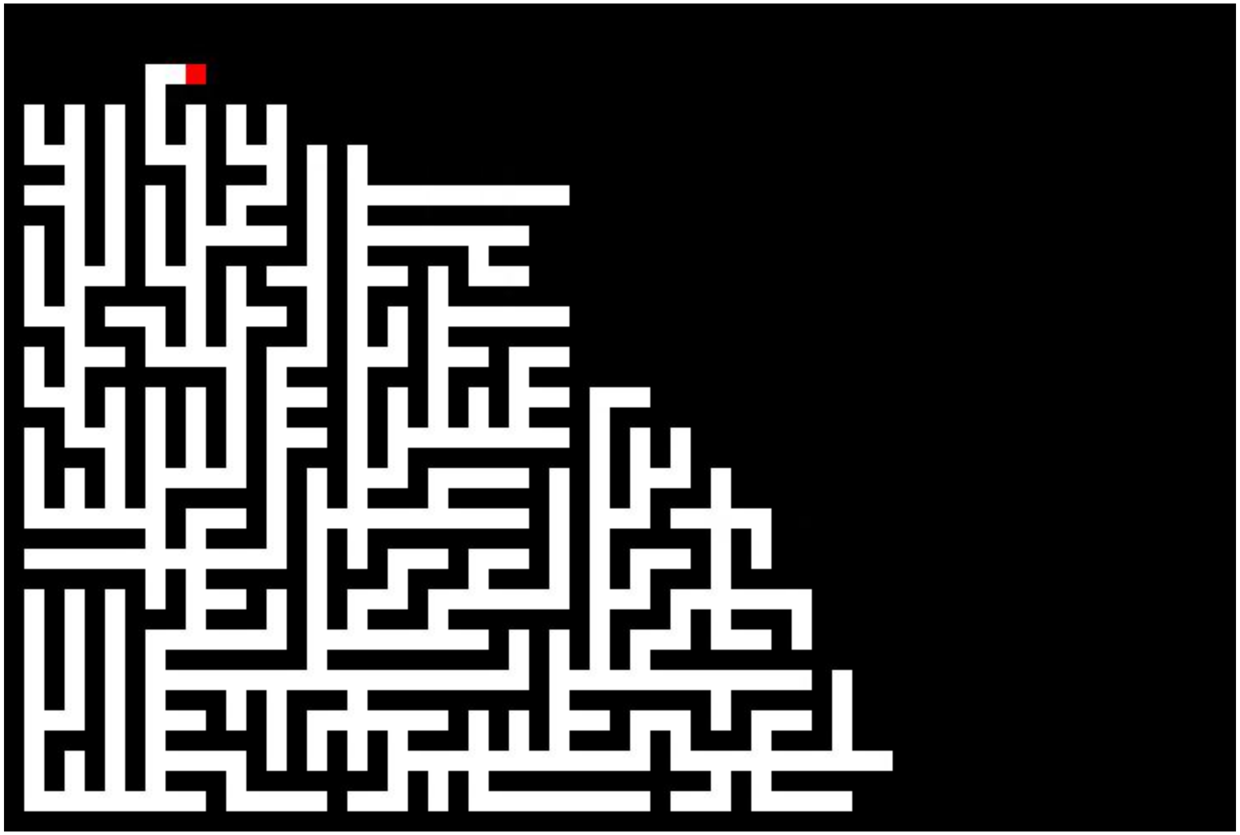

```
prim(s) {
  D[s]= 0; //start vertex
  D[i]= ∞ for all i ≠ s;
  while (a vertex is unmarked) {
    v= unmarked vertex
            with smallest D;
    mark v;
    for (each w adj to v)
      D[w]= min(D[w], c(v,w));
  }
}
```

- $O(m + n \log n)$ for adj list
  - Use a PQ
  - Regular PQ produces time $O(n + m \log m)$
  - Can improve to $O(m + n \log n)$ using a fancier heap

- $O(n^2)$ for adj matrix
  - while-loop iterates n times
  - for-loop takes $O(n)$ time

Application of MST

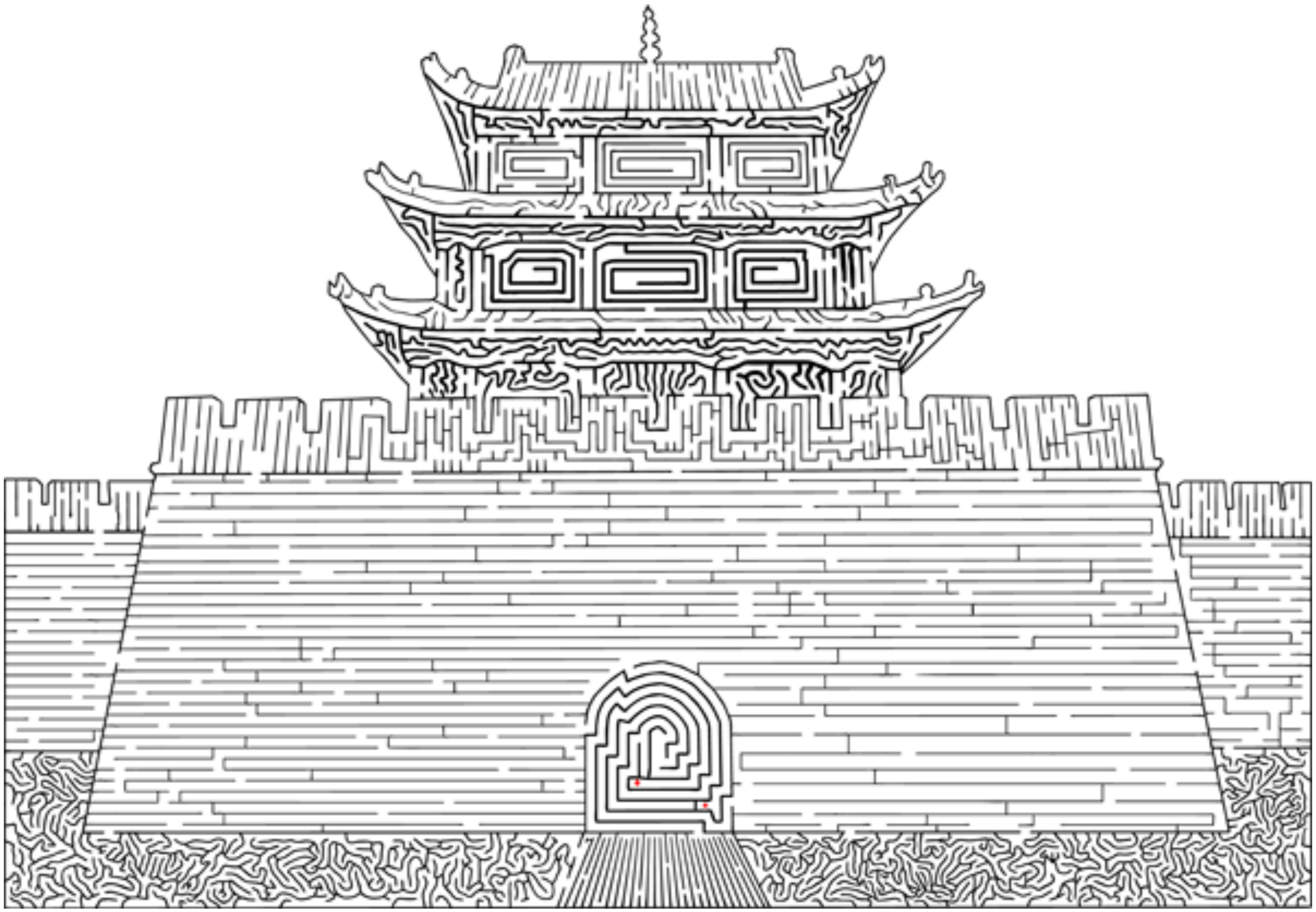
Maze generation using Prim's algorithm



The generation of a maze using Prim's algorithm on a randomly weighted grid graph that is 30x20 in size.

http://en.wikipedia.org/wiki/File:MAZE_30x20_Prim.ogv

More complicated maze generation



http://www.cgl.uwaterloo.ca/~csk/projects/mazes/

Greedy algorithms

- These are Greedy Algorithms
- Greedy Strategy: is an algorithm design technique
  - Like Divide & Conquer
- Greedy algorithms are used to solve optimization problems
  - Goal: 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: Making change

Given an amount of money, find smallest number of coins to make that amount

Solution: Use Greedy Algorithm:

Use as many large coins as you can.

Produces optimum number of coins for US coin system

May fail for old UK system

Similar code structures


```
while (a vertex is unmarked) {
    v= best unmarked vertex
    mark v;
    for (each w adj to v)
        update D[w];
}
```

- Breadth-first-search (bfs)
  - best: next in queue
  - update: D[w] = D[v]+1
- Dijkstra's algorithm
  - best: next in priority queue
  - update: D[w] = min(D[w], D[v]+c(v,w))
- Prim's algorithm
  - best: next in priority queue
  - update: D[w] = min(D[w], c(v,w))

c(v,w) is the v→w edge weight

Traveling salesman problem

Given a list of cities and the distances between each pair, what is the shortest route that visits each city exactly once and returns to the origin city?

- The true TSP is very hard (called NP complete)… for this we want the ***perfect*** answer in all cases.
- Most TSP algorithms start with a spanning tree, then "evolve" it into a TSP solution. Wikipedia has a lot of information about packages you can download…