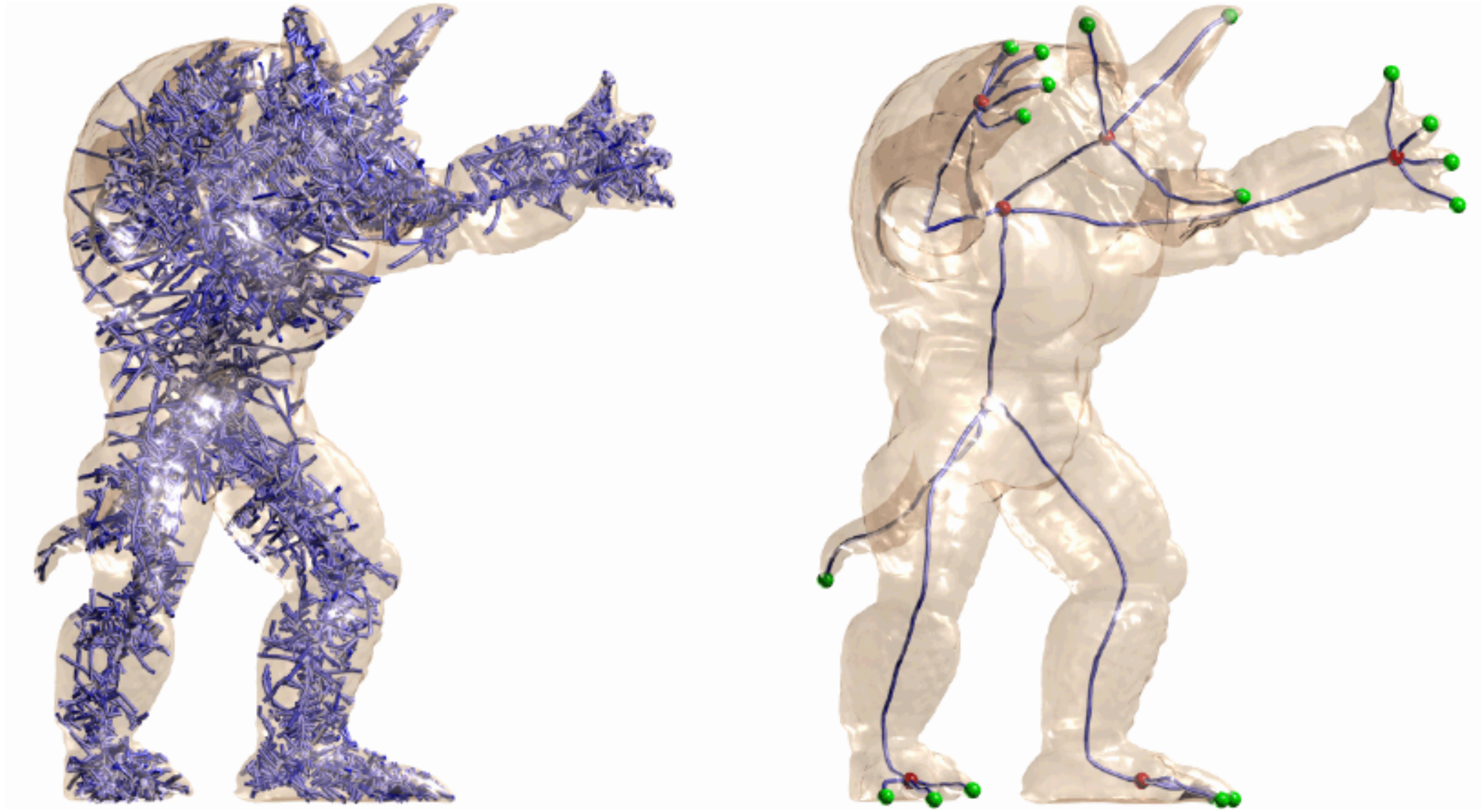


CSCI 241

Lecture 24
Minimum Spanning Trees; Prim’s Algorithm

Announcements

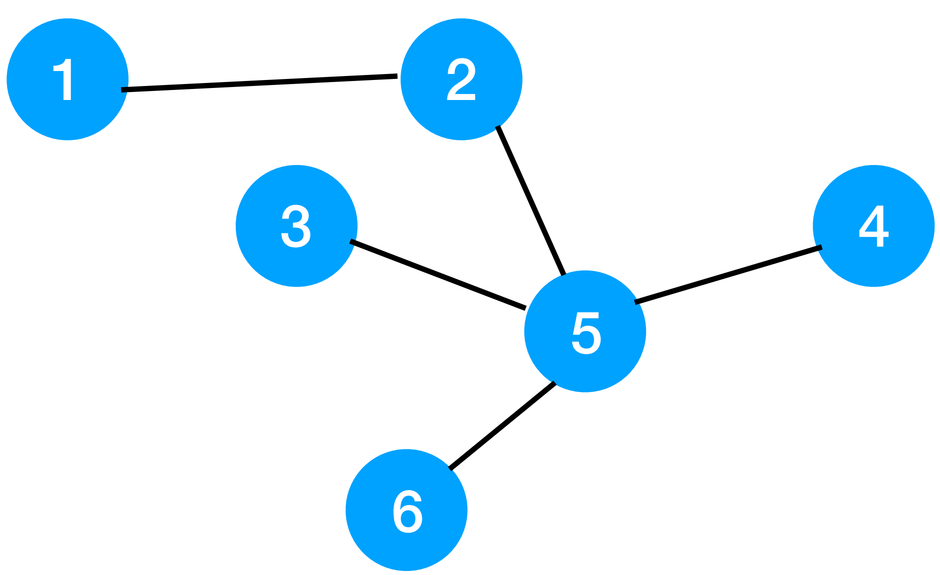
- A4: if you get "can't find JUnit" errors when running make buildtest, copy the lib/ directory from A3 into A4. The skeleton didn't have the jarfiles needed to run JUnit tests until last night.
- If your A3 didn't pass all tests, use the .class files included in the build folder.
  - If you're having trouble getting this to work, email me or come see me at office hours. Don't let A3 get in your way of completing A4.
- You may change `ShortestPaths.parseGraph` from `private` to `protected` and use it in ShortestPathsTest.java.
- Friday's quiz: DFS, BFS, Dijkstra, Minimum Spanning Trees

Goals

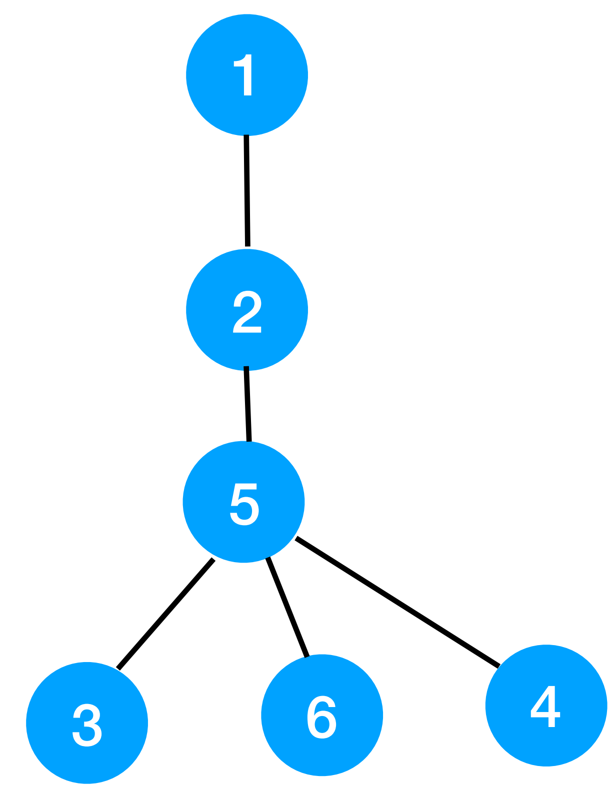
- Understand the definition of spanning tree
- Understand the additive and subtractive approaches to finding spanning trees.
- Understand the definition of a minimum spanning tree.
- Be able to run Kruskal’s algorithm and Prim’s algorithm on a graph on paper.

Trees vs Graphs

- Trees are graphs!
- A tree is an undirected graph with exactly 1 path between all pairs of nodes.
- Implication: no cycles!



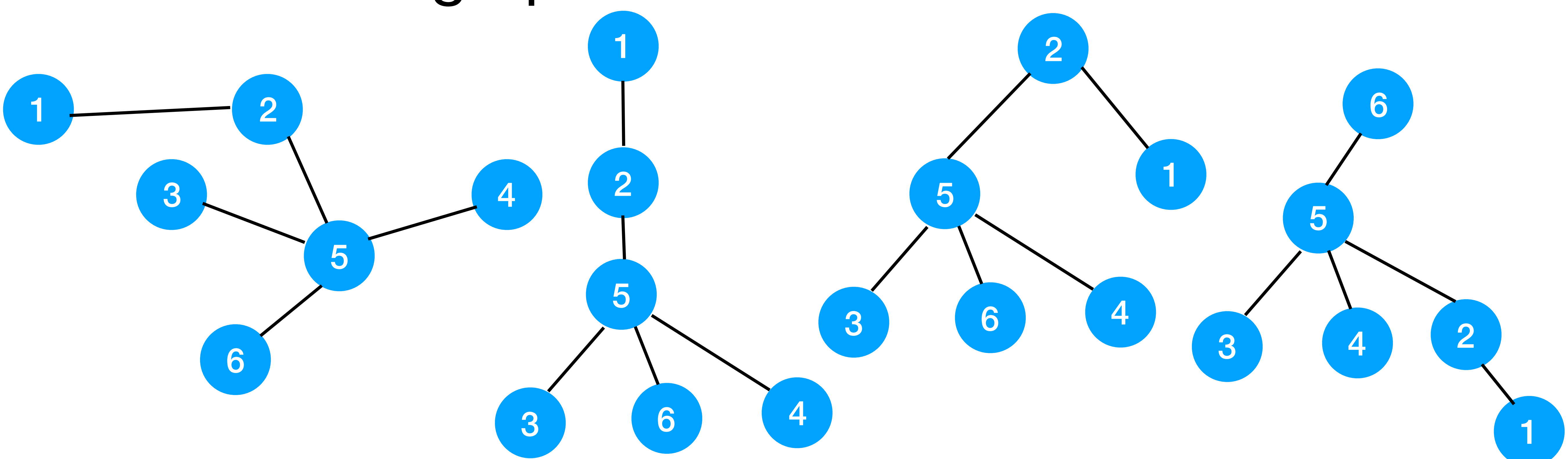
$$V = \{1,2,3,4,5,6\}$$
$$E = \{(1,2), (2,5), (3,5) \\ (4,5), (5,6)\}$$



Many problems are easy in trees and harder in graphs.

Trees vs Graphs

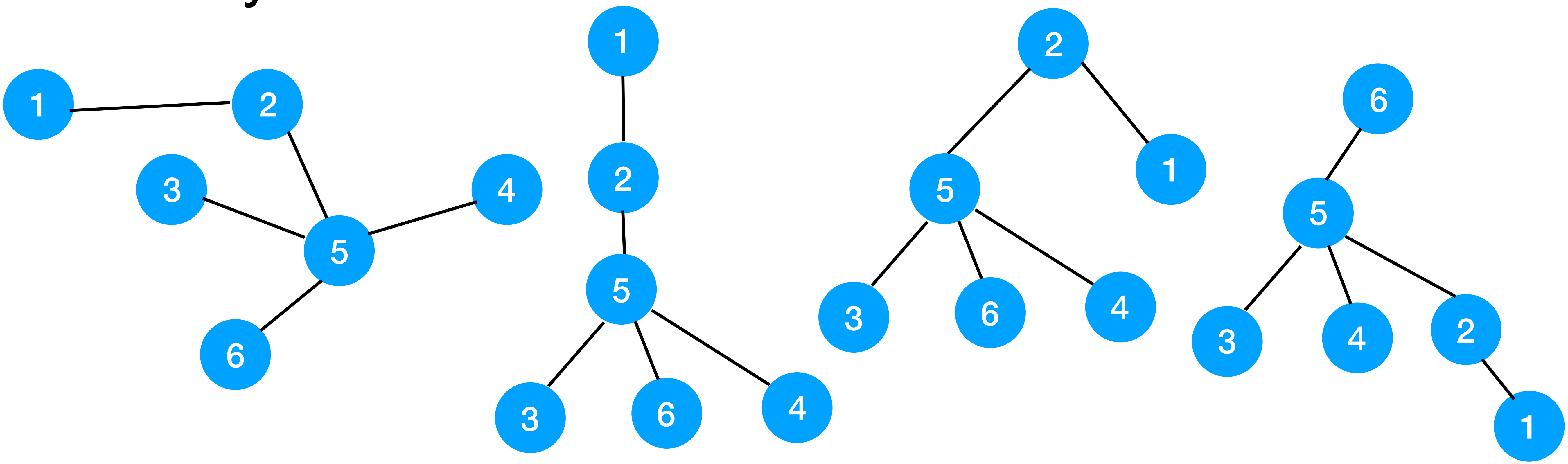
- A tree is an undirected graph with exactly 1 path between all pairs of nodes.
- Undirected tree: a tree with no root specified. All these graphs are the same tree:



Many problems are easy in trees and harder in graphs.

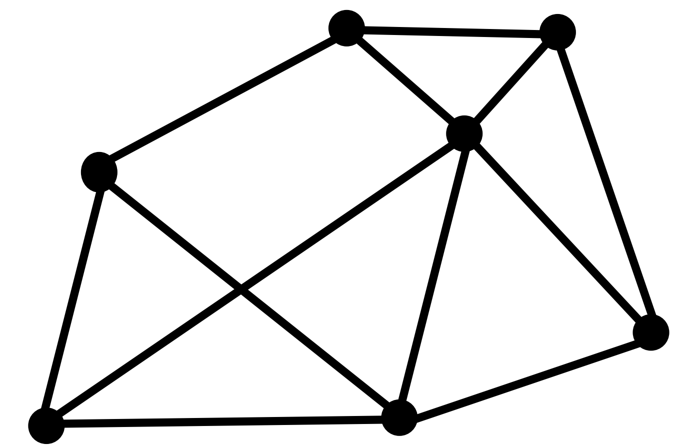
Tree Facts

- $|E| = |V| - 1$
- Connected
- Acyclic

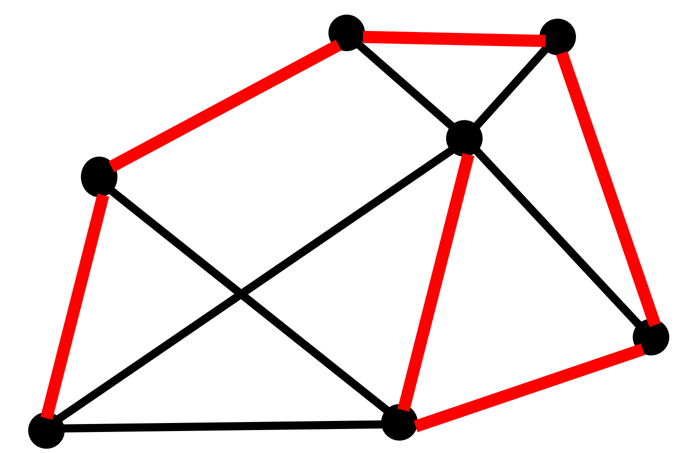


Spanning Trees

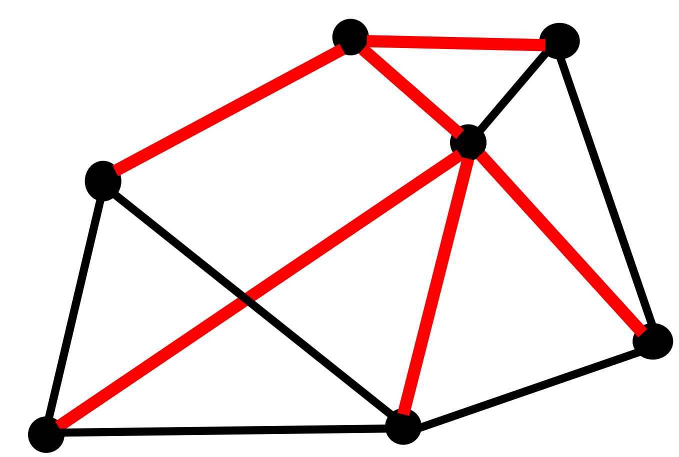
- A spanning tree of a connected undirected graph (V, E) is a subgraph (V, E') that is a tree.



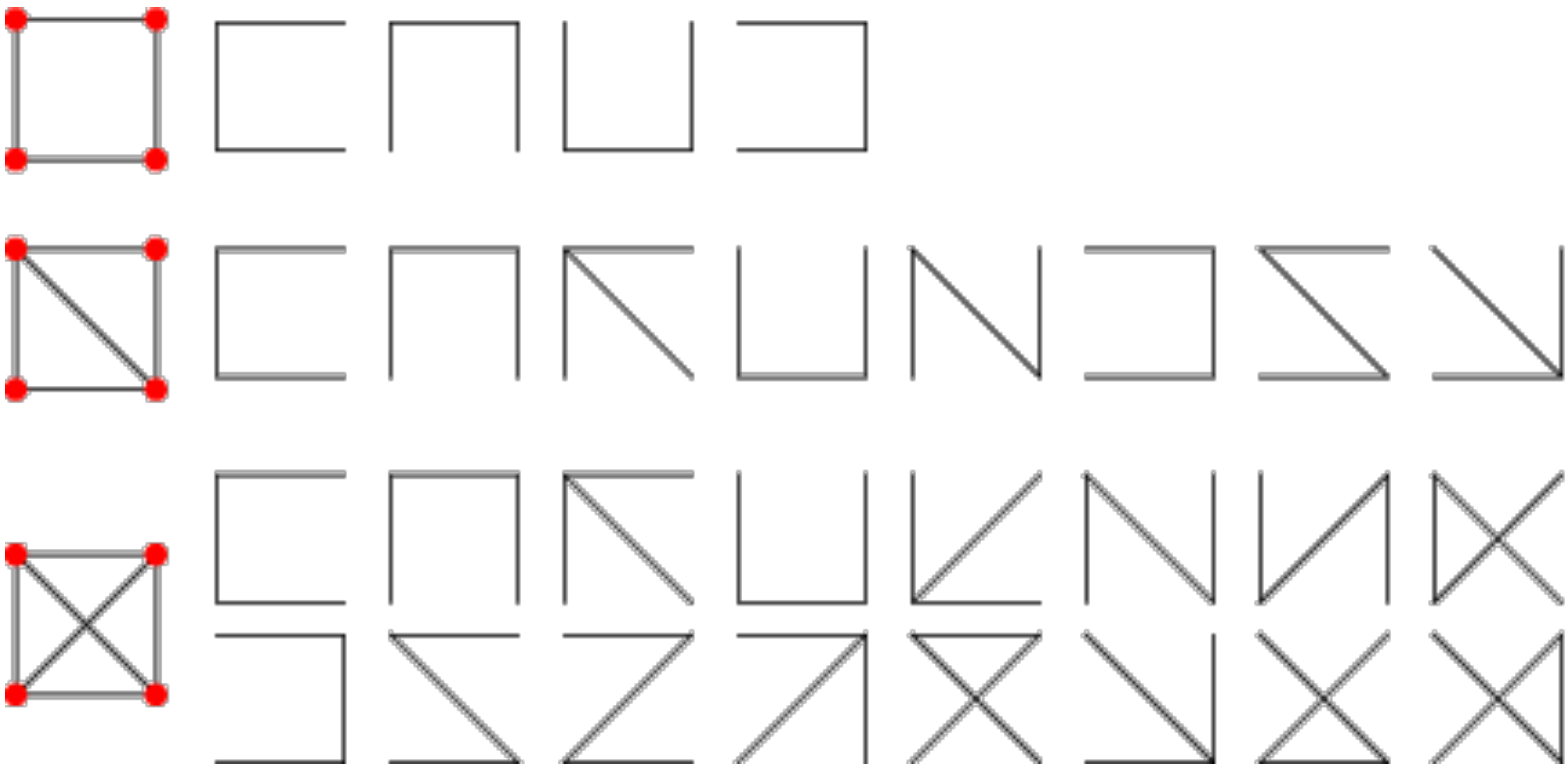
  - V is the same - tree has **all** the nodes
  - E' ⊆ E: tree has a **subset** of the edges



- Equivalent definitions:
  - Maximal set of edges containing no cycles.
  - Minimal set of edges connecting all nodes.



Examples



Unless a graph is itself a tree, it has multiple possible spanning trees.

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

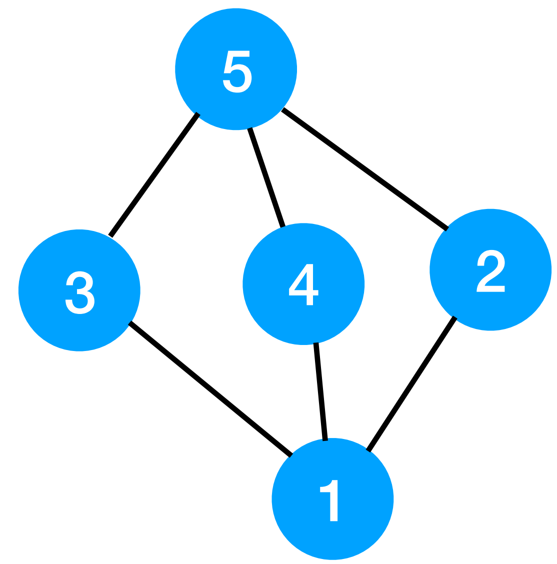
How many spanning trees does this graph have?

A. 4

B. 8

C. 12

D. 16



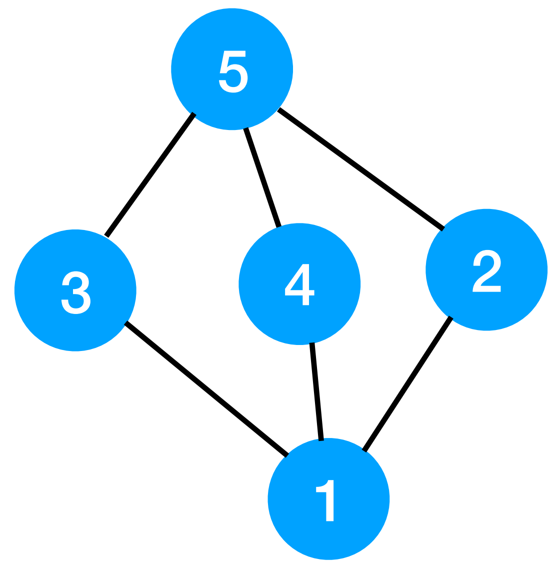
How many spanning trees does this graph have?

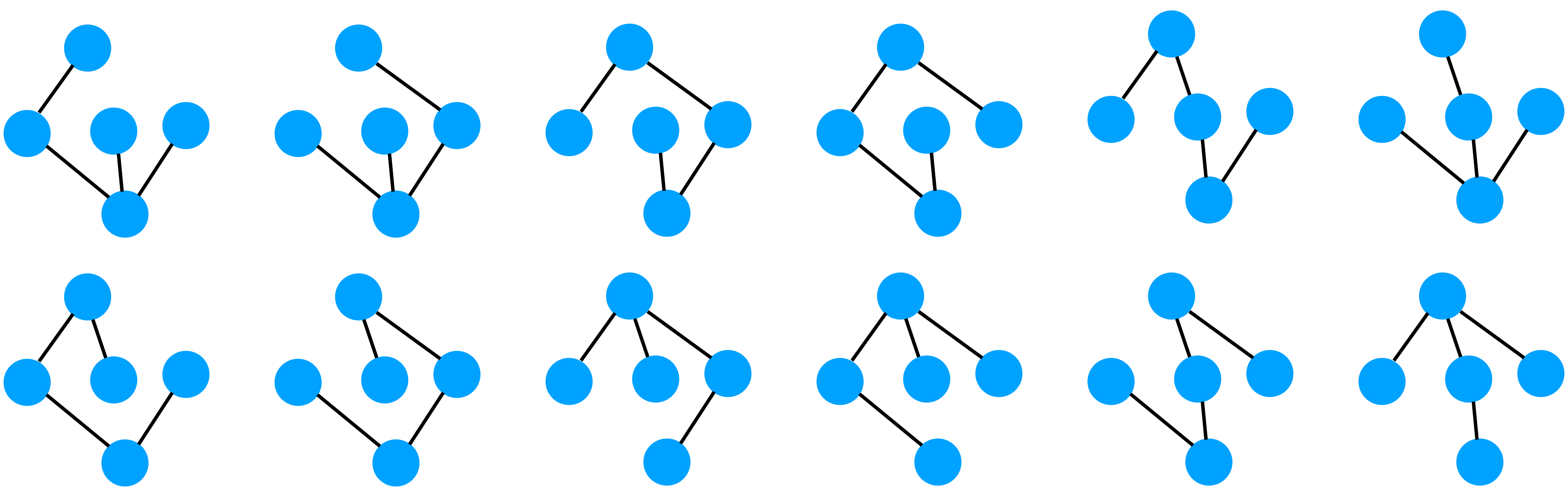
A. 4

B. 8

C. 12

D. 16

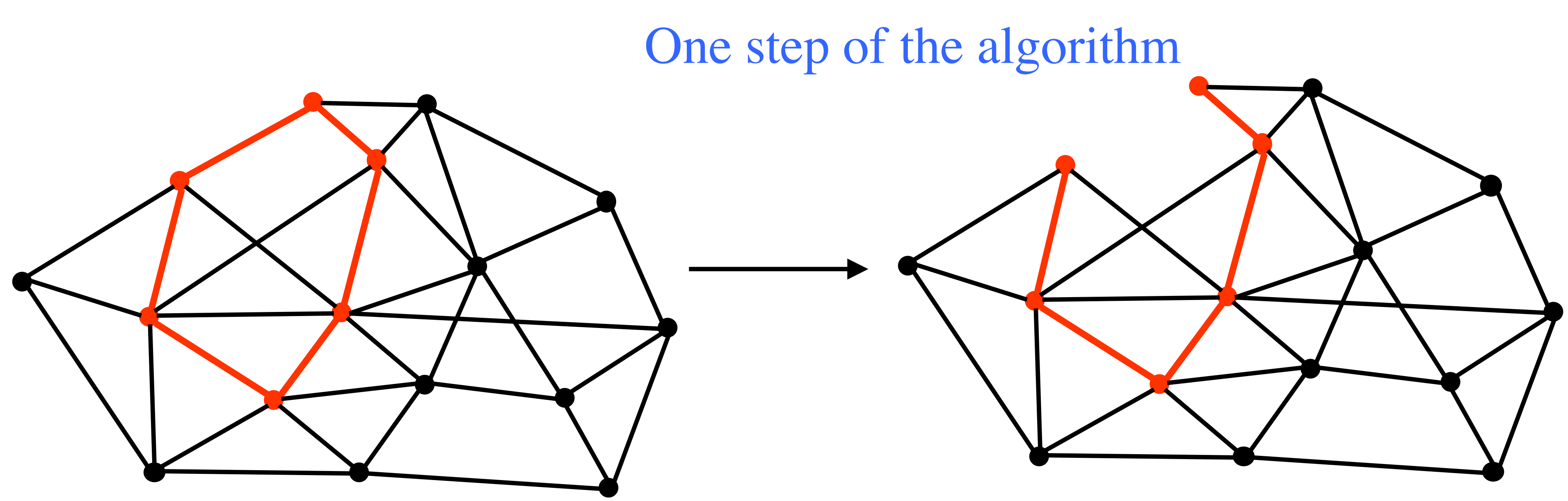




Finding a Spanning Tree

Subtractive method:

- Start with the whole graph
- While there exists a cycle:
  - remove an edge from that cycle.



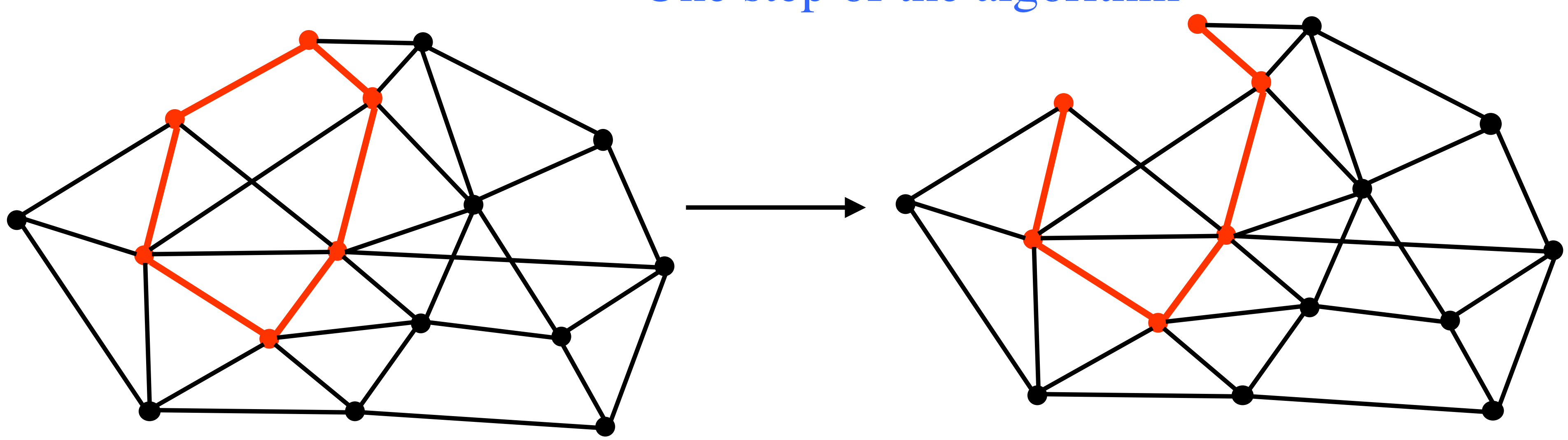
Finding a Spanning Tree

Subtractive method:

- Start with the whole graph
- While **there exists a cycle**:
  - remove an edge from that cycle.

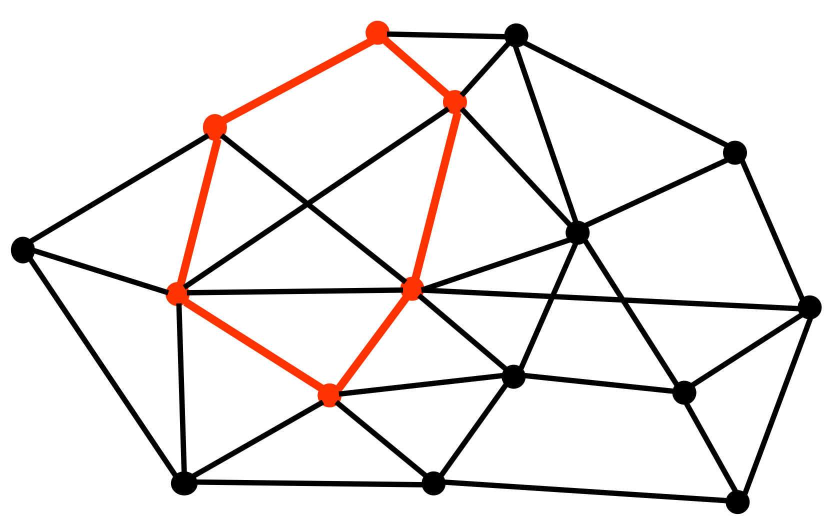
How do we know?

One step of the algorithm



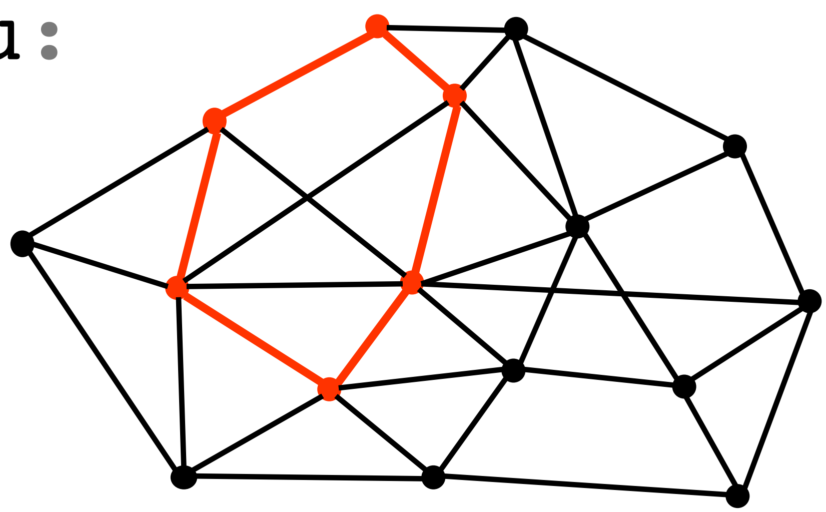
Finding Cycles: Use DFS!

```
/** Visit all nodes explorable from u.
  * Pre: u is unvisited. */
public static boolean dfs(Node u) {
  Stack s = (u);
  while (s is not empty) {
    u = s.pop();
    if (u has not been visited) {
      visit u;
      for each edge (u, v) leaving u:
        s.push(v);
    }
  }
}
```



Finding Cycles: Use DFS!

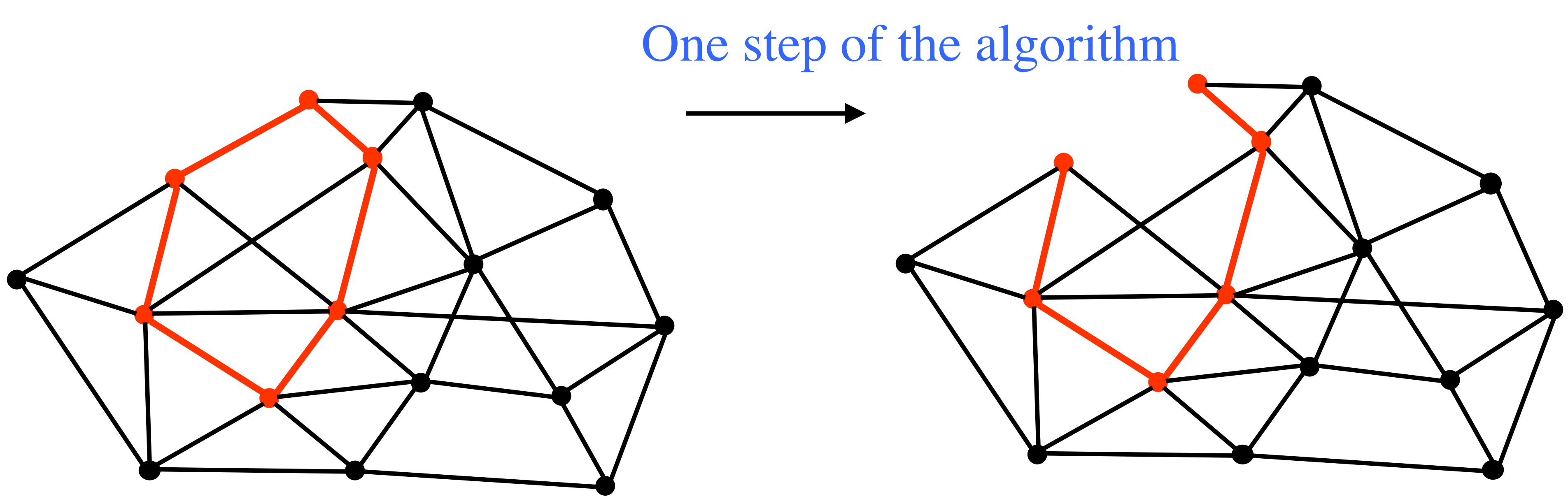
```
/** returns whether there is a cycle in
  * the nodes explorable from u */
public static boolean hasCycle(Node u) {
  Stack s = (u);
  while (s is not empty) {
    u = s.pop();
    if (u has been visited) {
      return true
    } else { // u has not been visited)
      visit u;
      for each edge (u, v) leaving u:
        s.push(v);
    }
  }
}
```



Finding a Spanning Tree

Subtractive method:

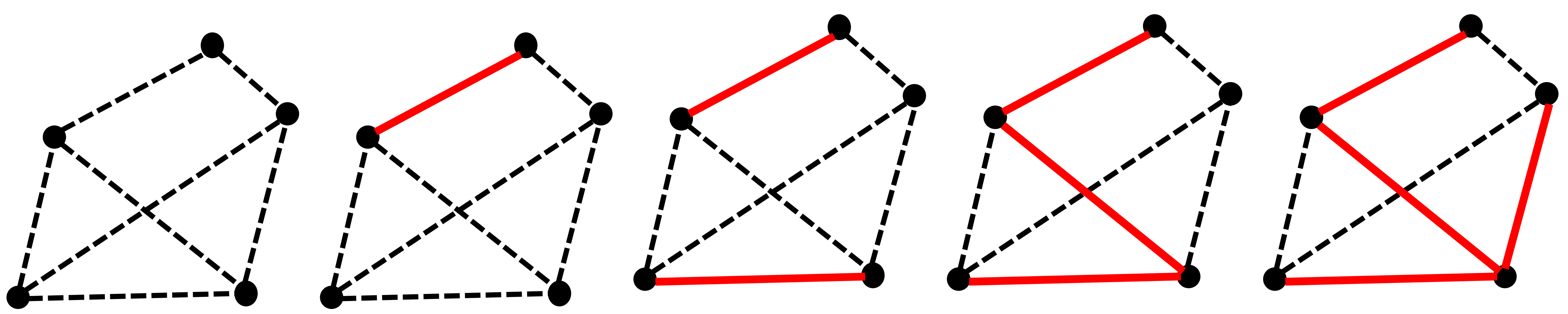
- Start with the whole graph
- n = arbitrary start node
- While **hasCycle(n)**:
  - remove an edge from that cycle.



Finding a Spanning Tree

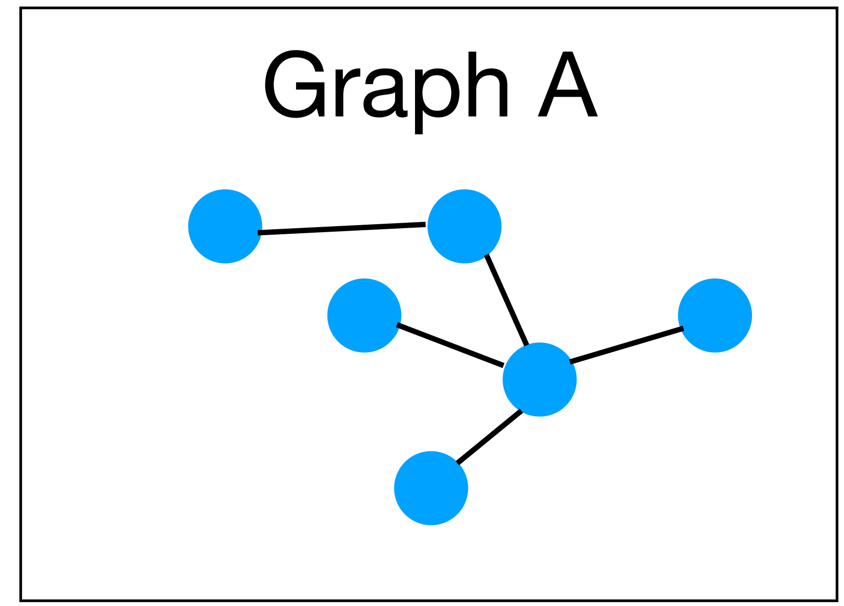
Additive method:

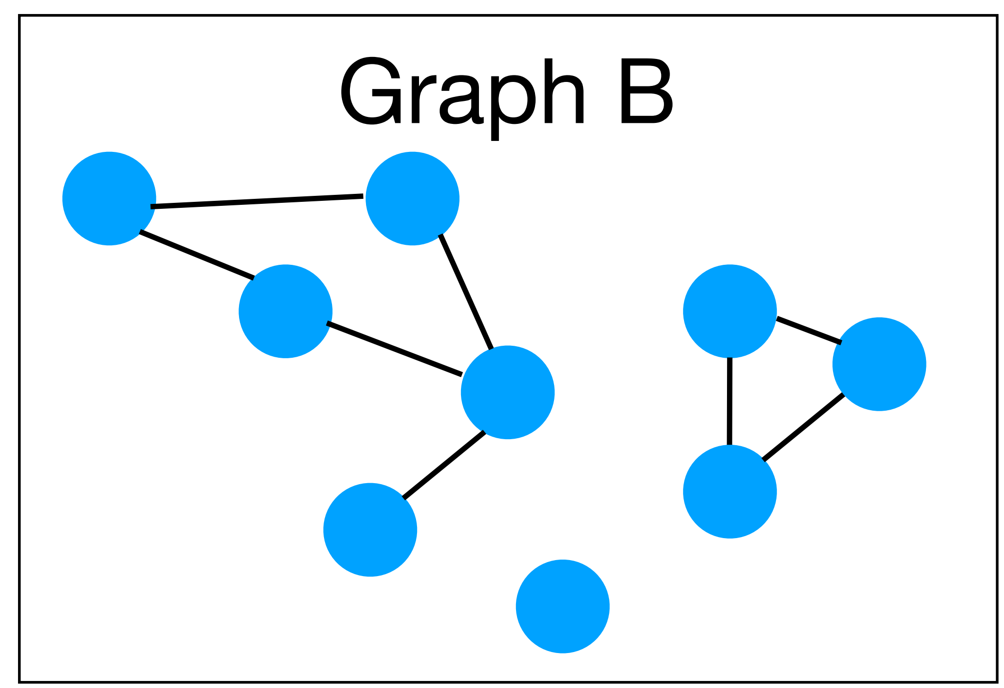
- Start with a graph containing **all** nodes and **no** edges
- While the graph is not connected:
  - add an edge that connects two connected components



Connected Components

- A connected component of G is a subgraph that is connected.
- Graph A has one connected component.
- How many does Graph B have?
  A. 1
  B. 2
  C. 3
  D. 4

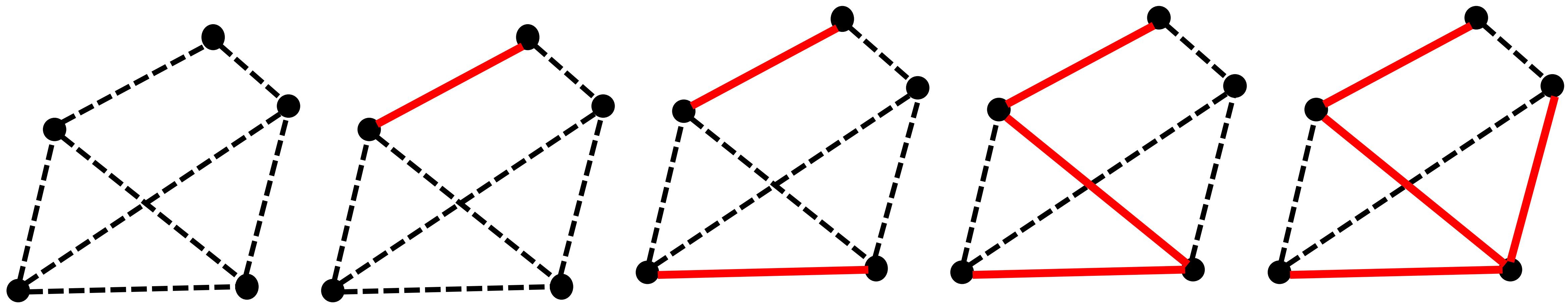




Finding a Spanning Tree

Additive method:

- Start with a graph containing **all** nodes and **no** edges
- While the graph is not connected:
  - add an edge that connects two connected components

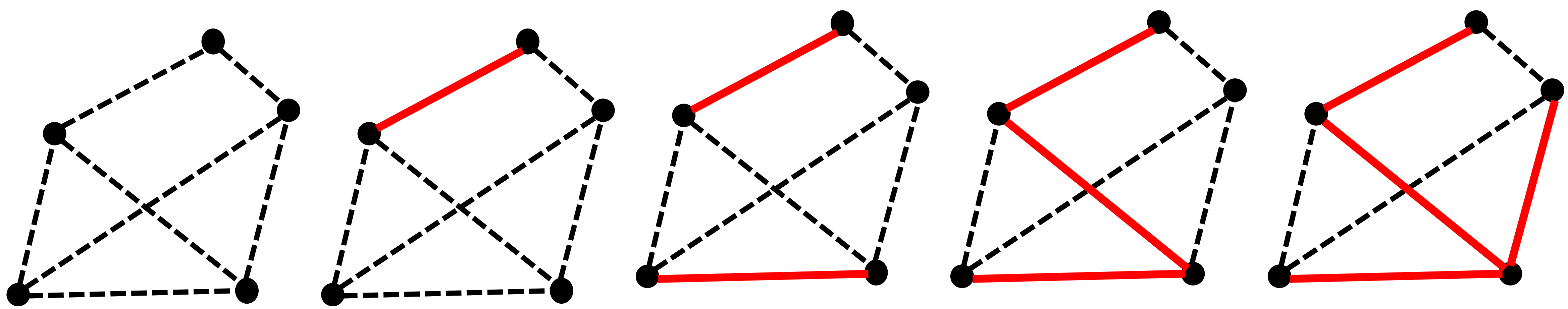


Finding a Spanning Tree

Additive method:

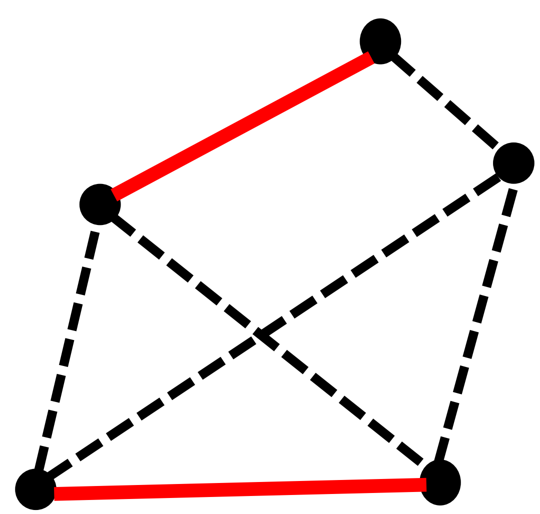
How do we know?

- Start with a graph containing **all** nodes and **no** edges
- While the **graph is not connected**:
  - add an edge that connects two connected components



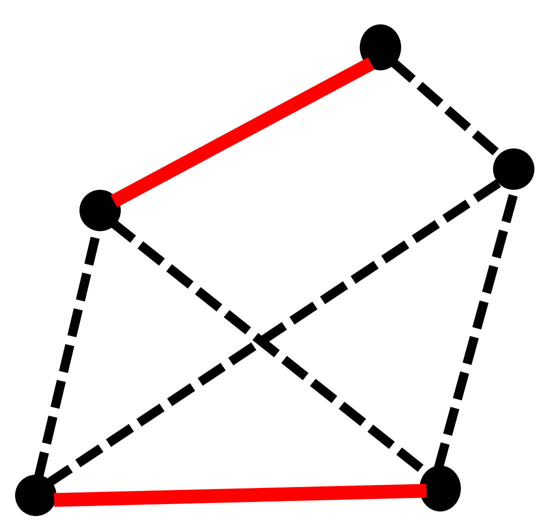
Finding Connected Components: Use DFS!

```
/** Visit all nodes explorable from u.
  * Pre: u is unvisited. */
public static boolean dfs(Node u) {
  Stack s = (u);
  while (s is not empty) {
    u = s.pop();
    if (u has not been visited) {
      visit u;
      for each edge (u, v) leaving u:
        s.push(v);
    }
  }
}
```



Finding Connected Components: Use DFS!

```
/** the set of vertices connected to u */
public static Set<Node> component(Node u) {
   Stack s = (u);
   Set<Node> comp = ();
   while (s is not empty) {
      u = s.pop();
      if (u has not been visited) {
         comp.add(u)
         visit u;
         for each edge (u, v) leaving u:
            s.push(v);
      }
   }
   return comp;
}
```

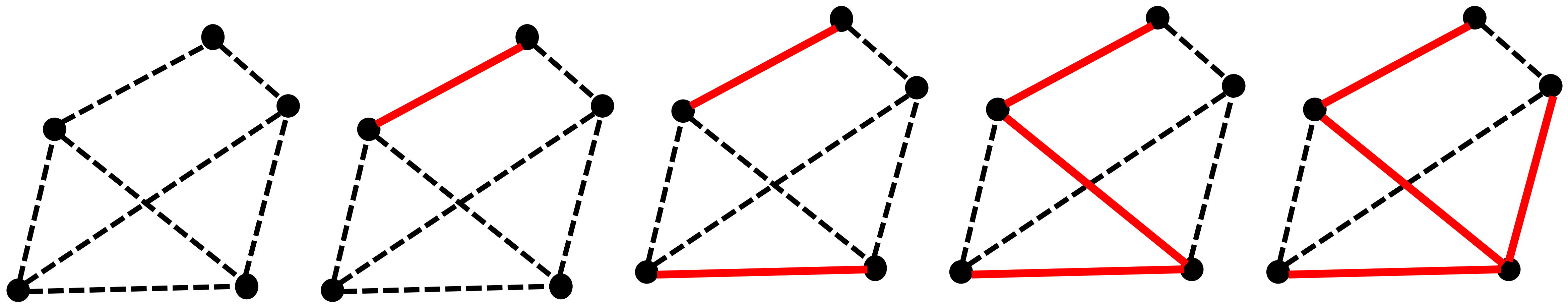


Finding a Spanning Tree

Additive method:

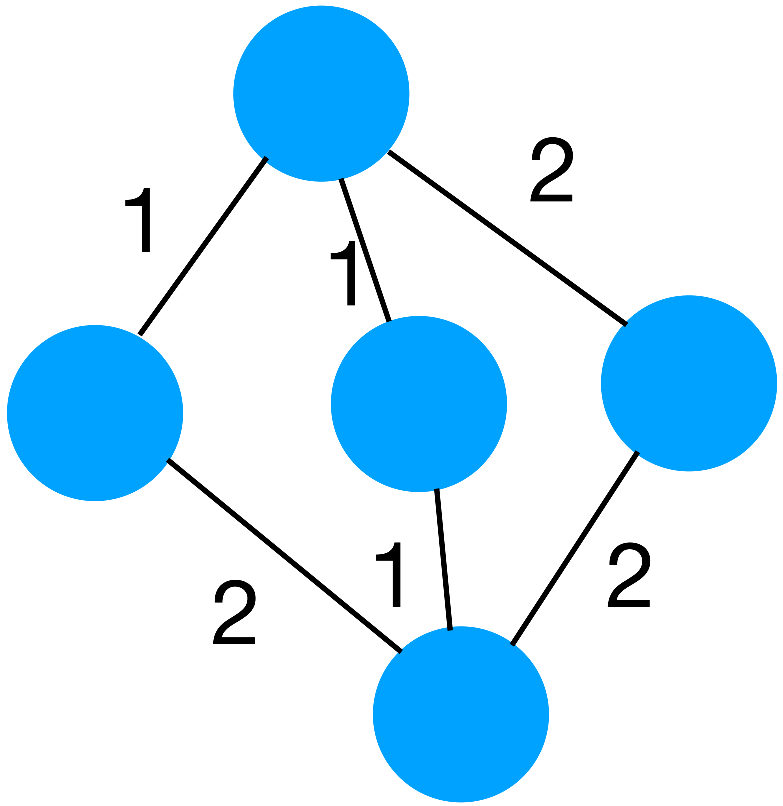
How do we know?

- Start with a graph containing **all** nodes and **no** edges
- While the **component(u) != V**:
  - add an edge that connects two connected components



Minimum Spanning Trees

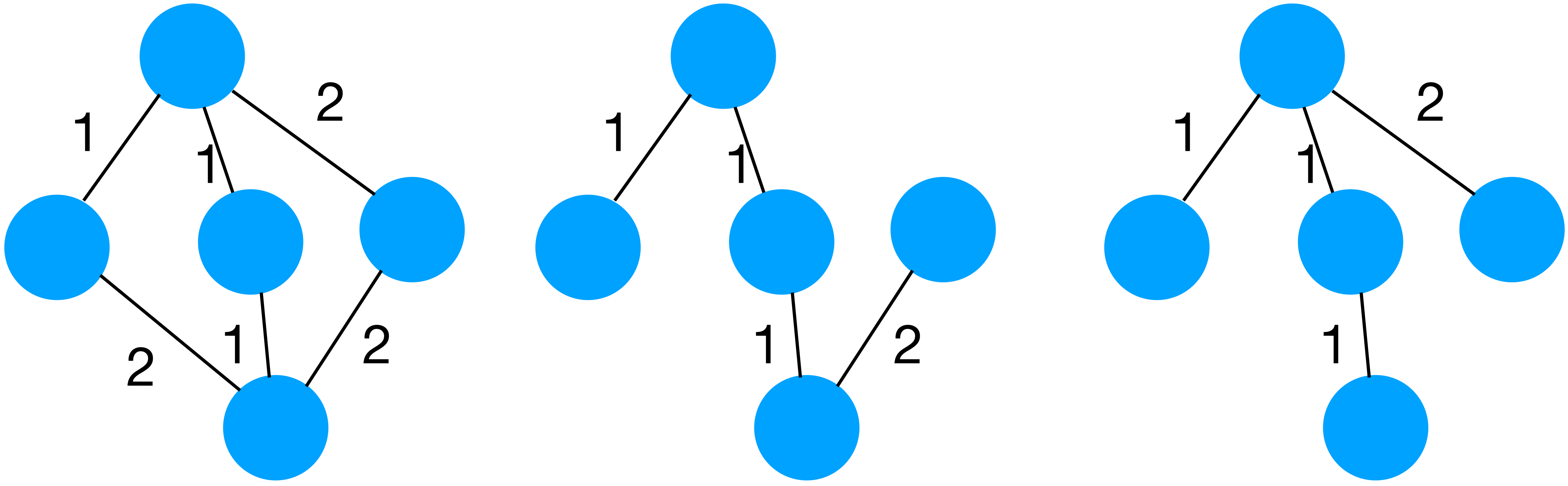
- If edges are weighted, we might want to find the spanning tree with minimum total edge weight.



Minimum Spanning Trees

- If edges are weighted, we might want to find the spanning tree with minimum total edge weight.

- MSTs are not necessarily unique:

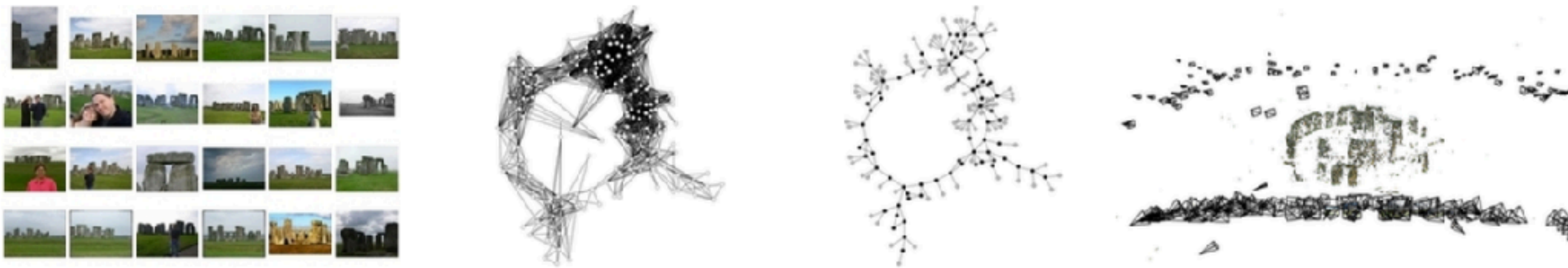


Applications of MSTs

- Transport networks
- Network routing
- Social marketing
- …

Applications of MSTs

- Computer vision???

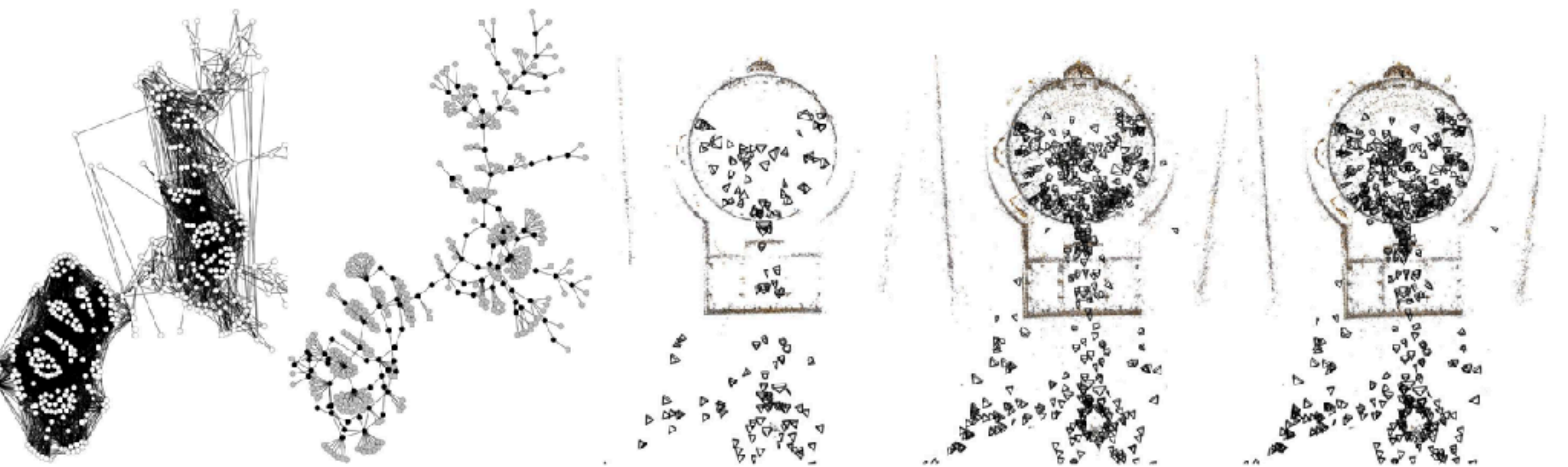


A few sample photos from a collection of Flickr images of Stonehenge.

An *image graph* for this photo collection.

Our computed skeletal graph.

A view of the complete reconstruction.



Finding a MST

- Recall: A MST has V-1 Edges
- Subtractive approach - remove all but V-1 edges
    - O(|V|^2 * time to decide which edge).
- Additive approach - add V-1 edges:
    - O(|V| * time to decide which edge)

Finding a MST: Two Additive algorithms

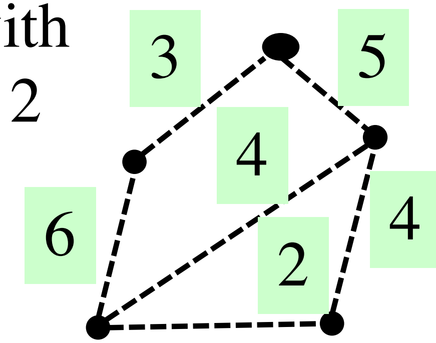
- Kruskal's Algorithm:
  - Add the minimum-weight edge that does not form a cycle.
- Prim's Algorithm:
  - Add the minimum-weight edge from the current spanning tree that does not form a cycle.

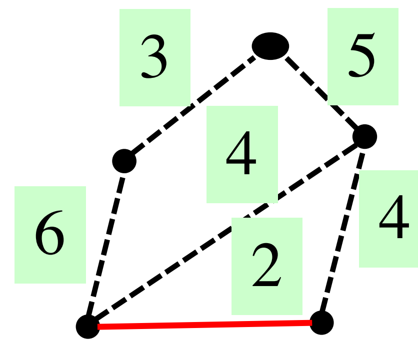
MST using Kruskal's algorithm

Minimal set of edges that connect all vertices

At each step, add an edge (that does not form a cycle) with minimum weight

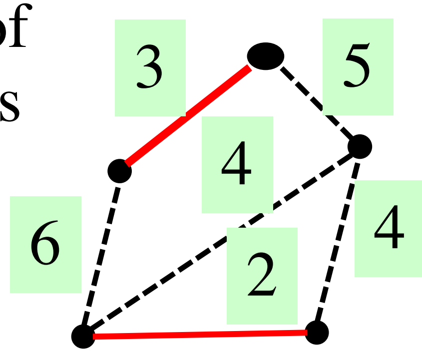
edge with weight 2



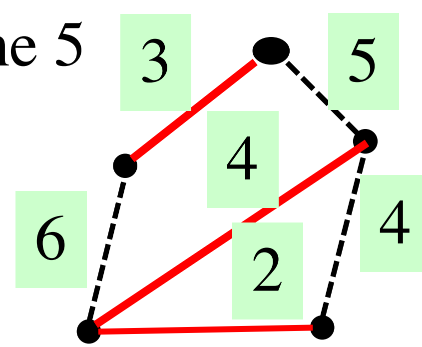


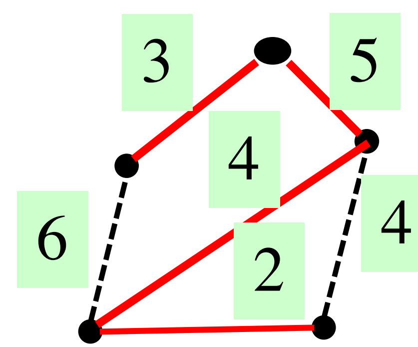
edge with weight 3

One of the 4's



The 5





Red edges need not form tree (until end)

Kruskal

Minimal set of edges that connect all vertices

Start with the all the nodes and no edges, so there is a forest of trees, each of which is a single node (a leaf).

At each step, add an edge (that does not form a cycle) with minimum weight

We do not look more closely at how best to implement Kruskal's algorithm —which data structures can be used to get a really efficient algorithm.

Leave that for later courses, or you can look them up online yourself.

We now investigate Prim's algorithm

MST using "Prim's algorithm" (should be called "JPD algorithm")

Developed in 1930 by Czech mathematician **Vojtěch Jarník**.
Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)

Developed in 1957 by computer scientist **Robert C. Prim**.
Bell System Technical Journal, 36 (1957), pp. 1389–1401

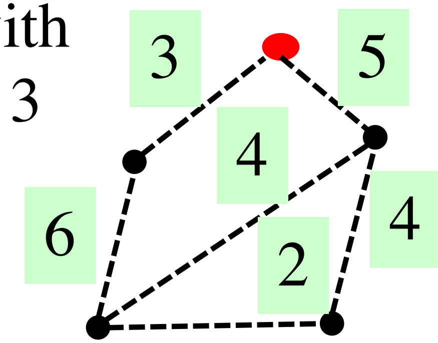
Developed about 1956 by **Edsger Dijkstra** and published in in 1959. *Numerische Mathematik* 1, 269–271 (1959)

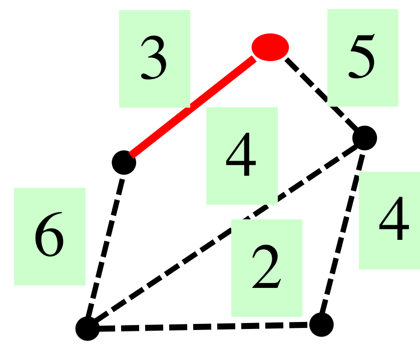
Prim's algorithm

Minimal set of edges that connect all vertices

At each step, add an edge (that does not form a cycle) with minimum weight, but keep added edge connected to the start (red) node

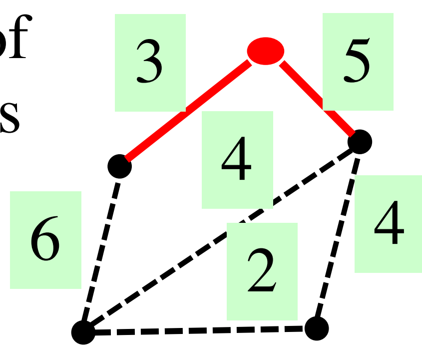
edge with weight 3



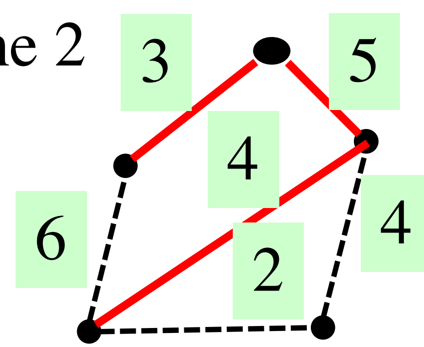


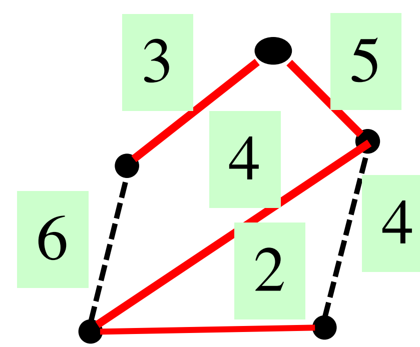
edge with weight 5

One of the 4's



The 2





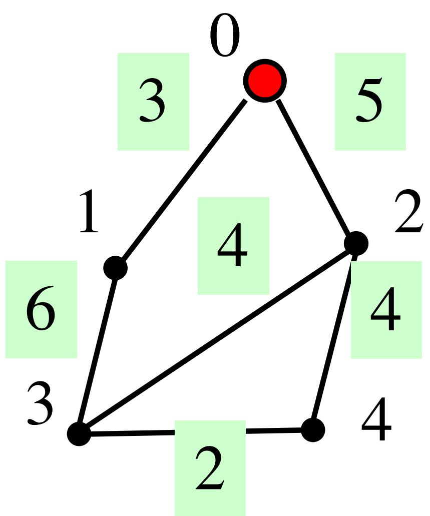
Difference between Prim and Kruskal

Minimal set of edges that connect all vertices

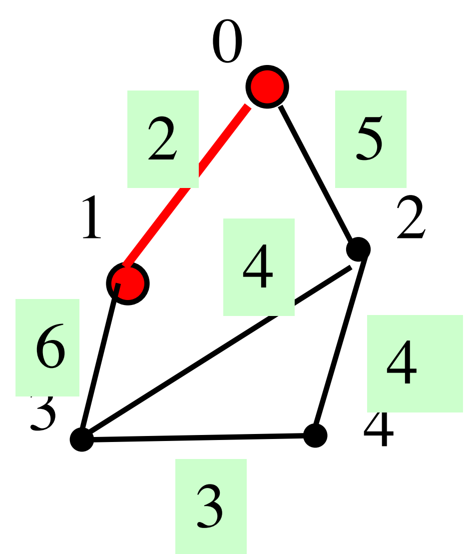
Prim requires that the constructed red tree always be connected.
Kruskal doesn't

But: Both algorithms find a minimal spanning tree

Here, Prim chooses (0, 1)
Kruskal chooses (3, 4)



Here, Prim chooses (0, 2)
Kruskal chooses (3, 4)



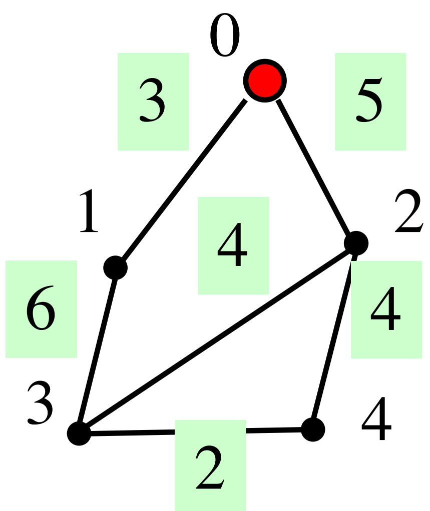
Difference between Prim and Kruskal

Prim requires that the constructed red tree always be connected.
Kruskal doesn’t

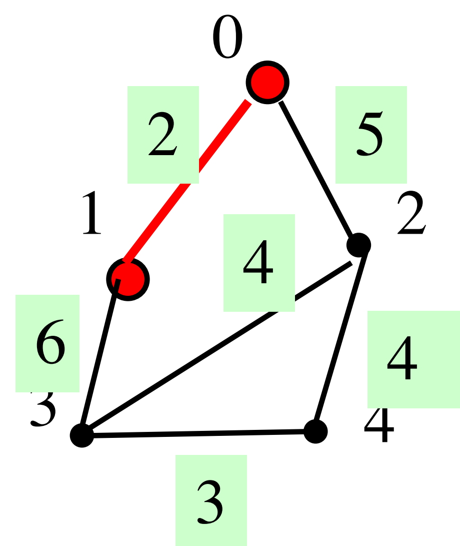
But: Both algorithms find a minimal spanning tree

Minimal set of edges that connect all vertices

Here, Prim chooses (0, 1)
Kruskal chooses (3, 4)



Here, Prim chooses (0, 2)
Kruskal chooses (3, 4)



Difference between Prim and Kruskal

Minimal set of edges that connect all vertices

Prim requires that the constructed red tree always be connected.
Kruskal doesn’t

But: Both algorithms find a minimal spanning tree

If the edge weights are all different, the Prim and Kruskal algorithms construct the same tree.