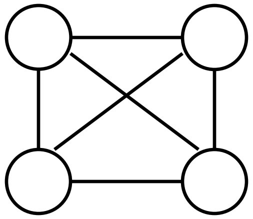
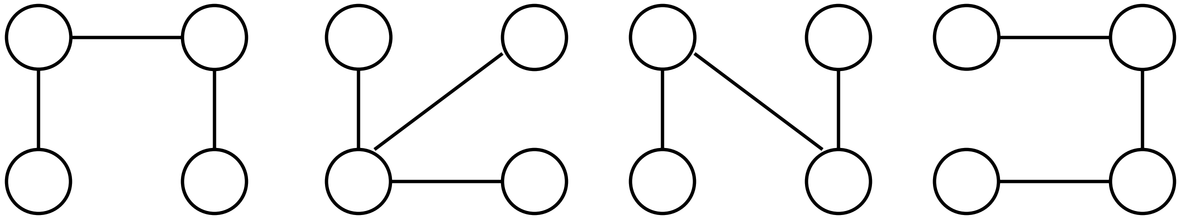
Spanning Trees

Spanning trees

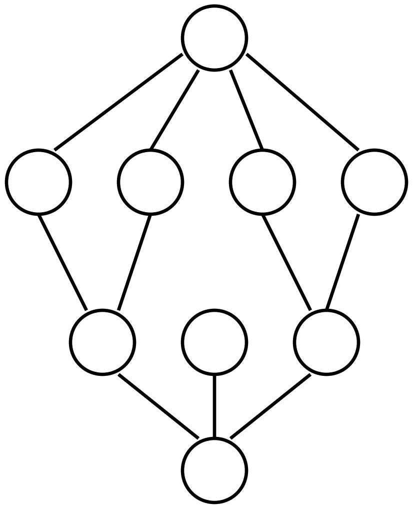
- Suppose you have a connected undirected graph
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction
- ...then a spanning tree of the graph is a connected subgraph in which there are no cycles

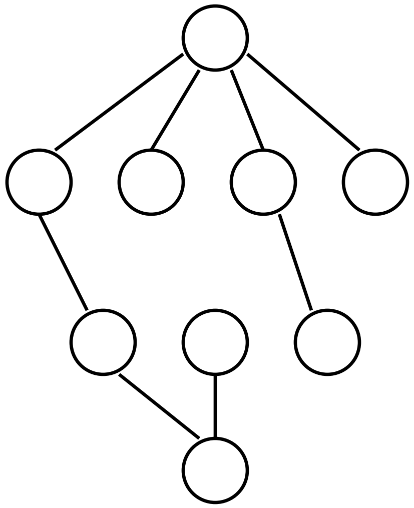

A connected, undirected graph

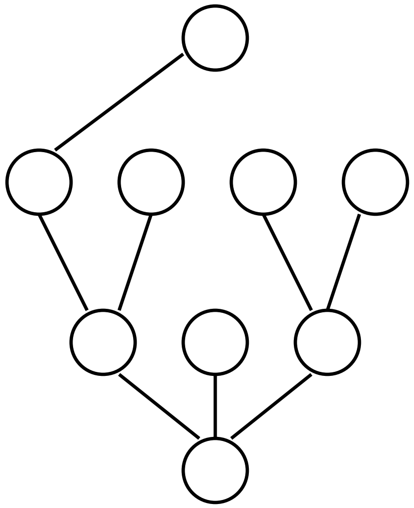

Four of the spanning trees of the graph

Finding a spanning tree

- To find a spanning tree of a graph,
  - pick an initial node and call it part of the spanning tree
  - do a search from the initial node:
    - each time you find a node that is not in the spanning tree, add to the spanning tree both the new node *and* the edge you followed to get to it


An undirected graph


One possible result of a BFS starting from top

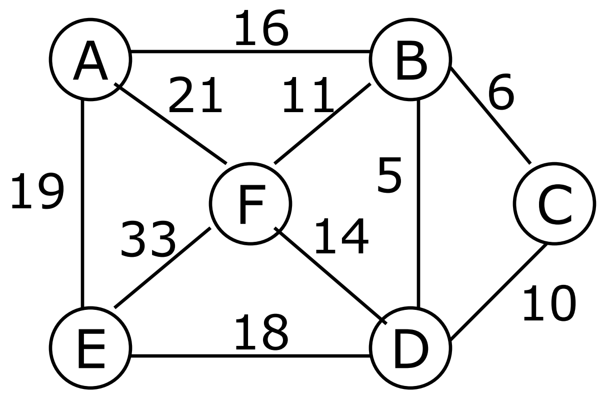

One possible result of a DFS starting from top

Minimizing costs

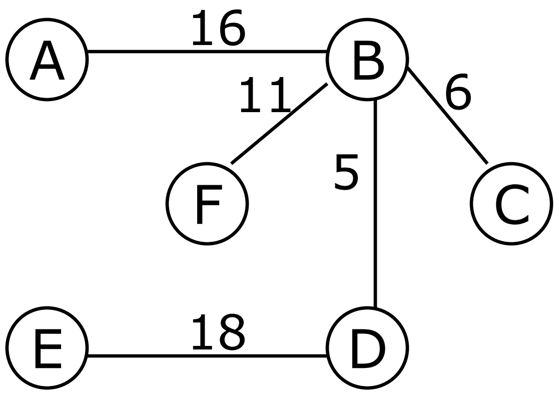
- Suppose you want to supply a set of houses (say, in a new subdivision) with:
  - electric power
  - water
  - sewage lines
  - telephone lines
- To keep costs down, you could connect these houses with a spanning tree (of, for example, power lines)
  - However, the houses are not all equal distances apart
- To reduce costs even further, you could connect the houses with a *minimum-cost* spanning tree

Minimum-cost spanning trees

- Suppose you have a connected undirected graph with a weight (or cost) associated with each edge
- The cost of a spanning tree would be the sum of the costs of its edges
- A minimum-cost spanning tree is a spanning tree that has the lowest cost



A connected, undirected graph



A minimum-cost spanning tree

Finding spanning trees

- There are two basic algorithms for finding minimum-cost spanning trees, and both are greedy algorithms

- **Kruskal's algorithm:** Start with *no* nodes or edges in the spanning tree, and repeatedly add the cheapest edge that does not create a cycle
  - Here, we consider the spanning tree to consist of edges only

- **Prim's algorithm:** Start with any *one node* in the spanning tree, and repeatedly add the cheapest edge, and the node it leads to, for which the node is not already in the spanning tree.
  - Here, we consider the spanning tree to consist of both nodes and edges

Kruskal's algorithm

```
T = empty spanning tree;
E = set of edges;
N = number of nodes in graph;
while T has fewer than N - 1 edges {
    remove an edge (v, w) of lowest cost from E
    if adding (v, w) to T would create a cycle
        then discard (v, w)
        else add (v, w) to T
}
```

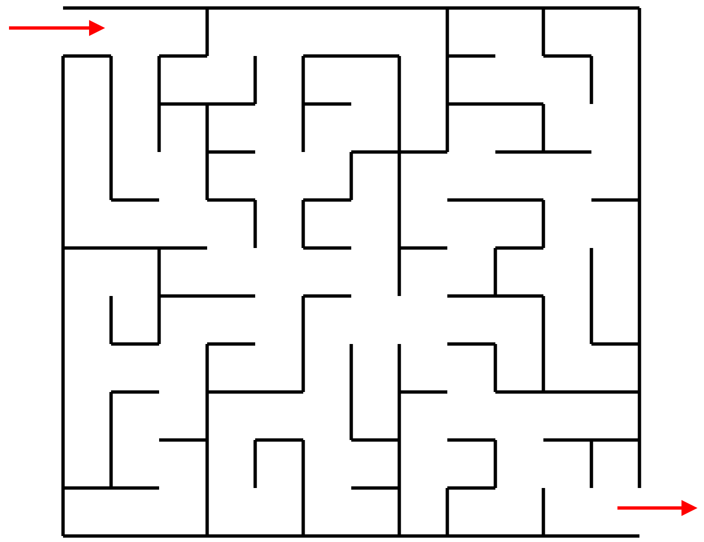
- Finding an edge of lowest cost can be done just by sorting the edges
- Efficient testing for a cycle requires a fairly complex algorithm (UNION-FIND) which we don't cover in this course

Prim’s algorithm

```
T = a spanning tree containing a single node s;
E = set of edges adjacent to s;
while T does not contain all the nodes {
    remove an edge (v, w) of lowest cost from E
    if w is already in T then discard edge (v, w)
    else {
        add edge (v, w) and node w to T
        add to E the edges adjacent to w
    }
}
```

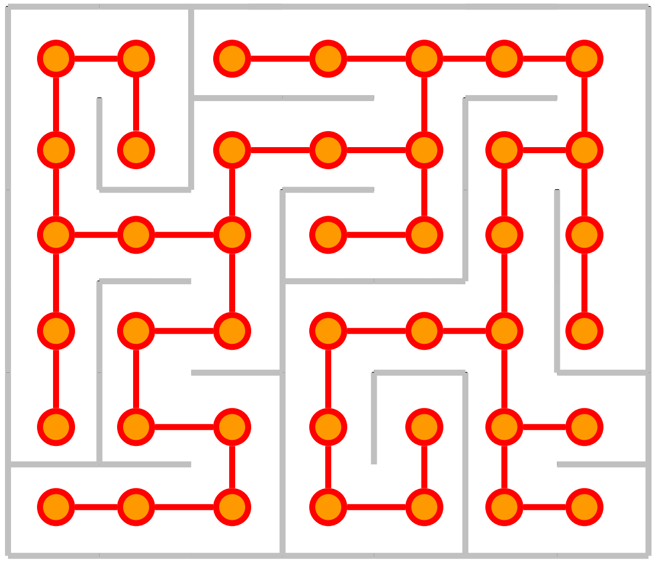
- An edge of lowest cost can be found with a priority queue
- Testing for a cycle is automatic
  - Hence, Prim’s algorithm is far simpler to implement than Kruskal’s algorithm

Mazes



- Typically,
  - Every location in a maze is reachable from the starting location
  - There is only one path from start to finish
- If the cells are "vertices" and the open doors between cells are "edges," this describes a spanning tree
- Since there is exactly one path between any pair of cells, *any* cells can be used as "start" and "finish"
- This describes a *spanning tree*

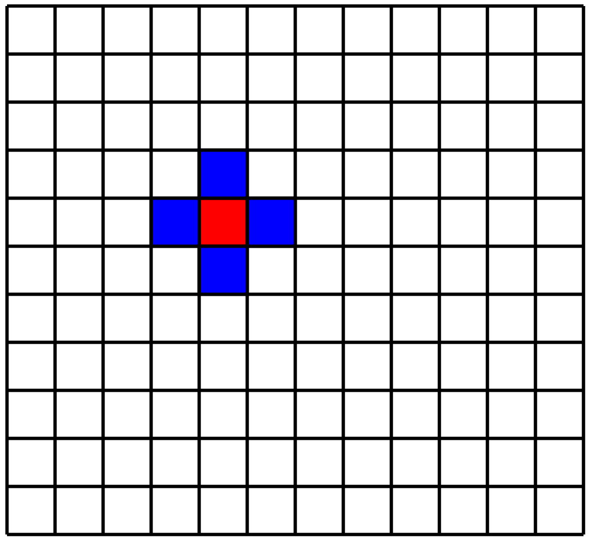
Mazes as spanning trees



- While not every maze is a spanning tree, most can be represented as such
- The nodes are “places” within the maze
- There is exactly one cycle-free path from any node to any other node

Building a maze I

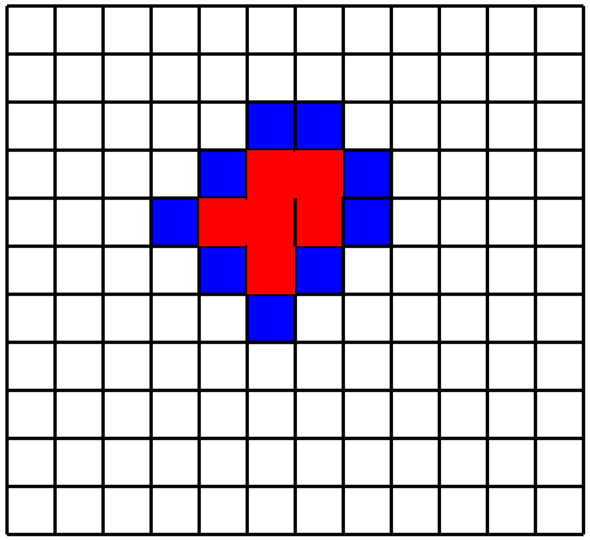
- This algorithm requires two *sets* of cells
  - the set of cells already in the spanning tree, IN
  - the set of cells adjacent to the cells in the spanning tree (but not in it themselves), called the FRONTIER
- Start with *all* walls present



- Pick any cell and put it into IN (red)
- Put all adjacent cells, that aren't in IN, into FRONTIER (blue)

Building a maze II

- Repeatedly do the following:
  - Remove any one cell C from FRONTIER and put it in IN
  - Erase the wall between C and some one adjacent cell in IN
  - Add to FRONTIER all the cells adjacent to C that aren't in IN (or in FRONTIER already)



- Continue until there are no more cells in FRONTIER
- When the maze is complete (or at any time), choose the start and finish cells

The End