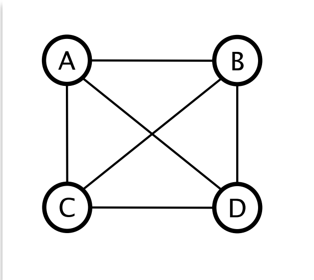
AUBURN UNIVERSITY

SAMUEL GINN COLLEGE OF ENGINEERING

Minimum Spanning Trees

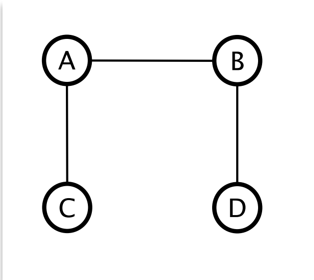
Spanning trees

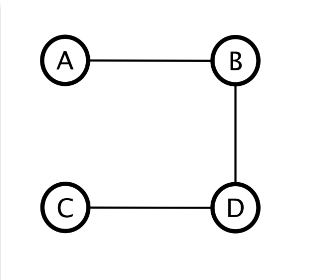


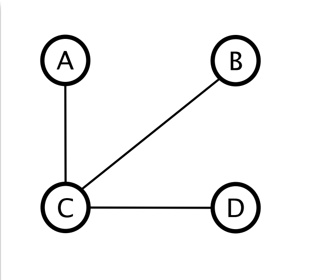
A **spanning tree** of a *connected, undirected* graph is a subgraph that contains

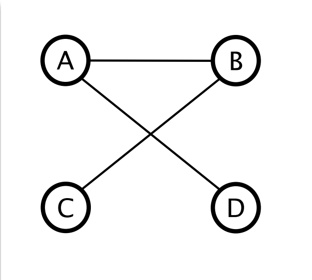
- all the vertices in the graph
- and the fewest number of edges such that the subgraph is connected

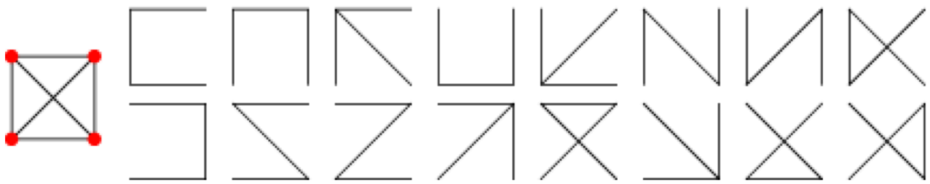
(contains no cycles)



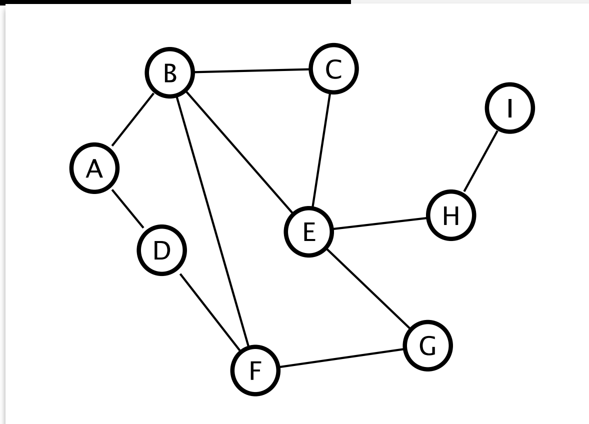








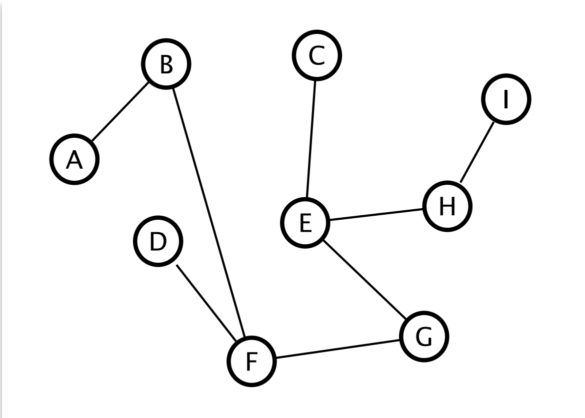
Constructing a spanning tree



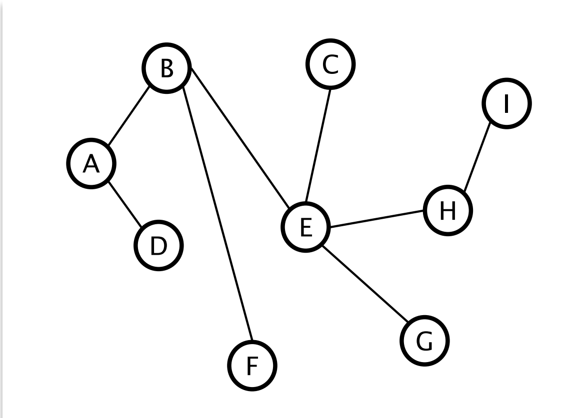
Use DFS (or BFS) and keep track of the edges crossed.

The set of crossed edges form a spanning tree.

DFS tree starting at A:



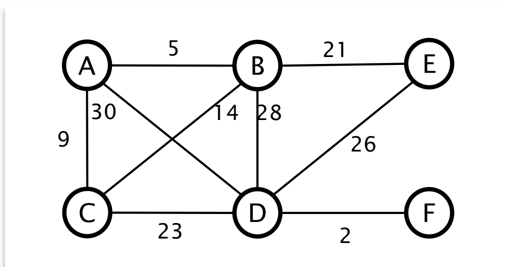
BFS tree starting at A:



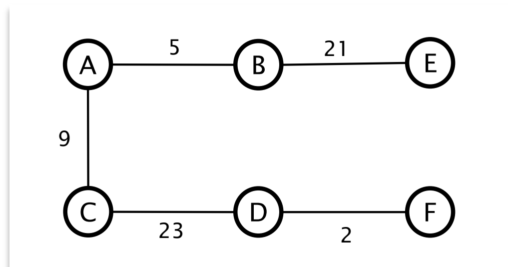
Minimum spanning trees

A **minimum spanning tree** of a connected, weighted, undirected graph is a spanning tree for the graph such that the sum of the edge weights is the minimum of all possible spanning trees for the graph.

Graph



MST

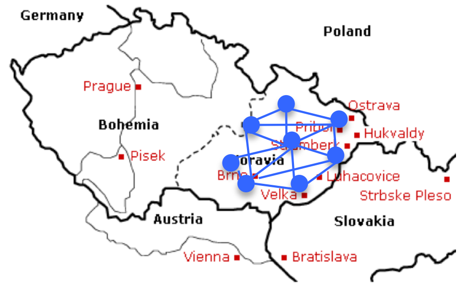


Minimum spanning trees

Otakar Borůvka

Czech mathematician

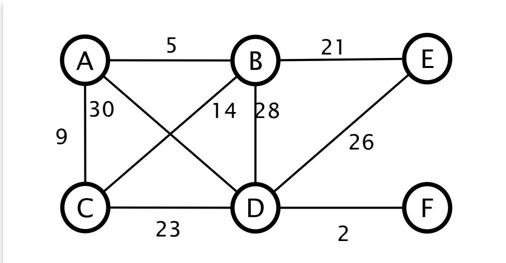
"On a Certain Minimal Problem" -1926



"Soon after the end of World War I, at the beginning of the 1920s, the Electric Power Company of Western Moravia, Brno, was engaged in rural electrification of Southern Moravia. In the framework of my friendly relations with some of their employees, I was asked to solve, from a mathematical standpoint, the question of the most economical construction of an electric power network. I succeeded in finding a construction – as it would be expressed today – of a maximal connected subgraph of minimum length, which I published in 1926."

Constructing MSTs and the greedy hueristic

Constructing a MST

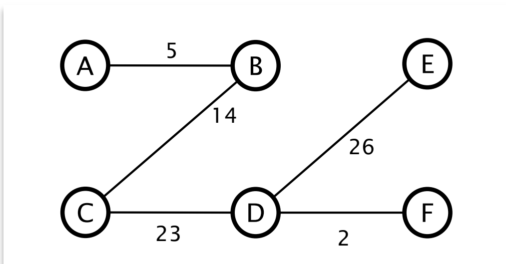


Use **DFS** and keep track of the edges crossed.

How to choose which neighbor to visit?

Choose the cheapest one that doesn't create a cycle. **The greedy choice**

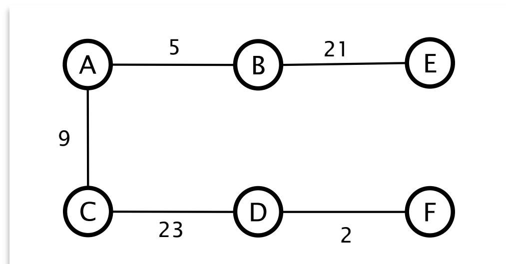
Resulting spanning tree:



Cost = 70

≠

MST (cost = 60)



A "greedy dfs" got us close, but not the optimal solution.

Greedy algorithms

Greedy algorithms are those that apply the following problem-solving heuristic when faced with choices:

Make locally optimal choices in hopes of arriving at a globally optimal solution

This is a *heuristic*, so it isn't guaranteed to always work. It can even produce the worst possible solution, or no solution at all.

Example: Arrive at a given monetary amount using the smallest possible number of coins.

US dollar, all coins

dollar, quarter, dime, nickel, penny

41¢

```
 41
-25
---
 16
-10
---
  6
 -5
---
  1
 -1
---
  0
```

Greedy heuristic: Choose the largest coin possible at each step.

Greedy algorithms

Greedy algorithms are those that apply the following problem-solving heuristic when faced with choices:

Make locally optimal choices in hopes of arriving at a globally optimal solution

This is a *heuristic*, so it isn't guaranteed to always work. It can even produce the worst possible solution, or no solution at all.

Example: Arrive at a given monetary amount using the smallest possible number of coins.

GBP, all coins

£5, £2, £1, 50p, 20p, 10p, 5p, 2p, 1p

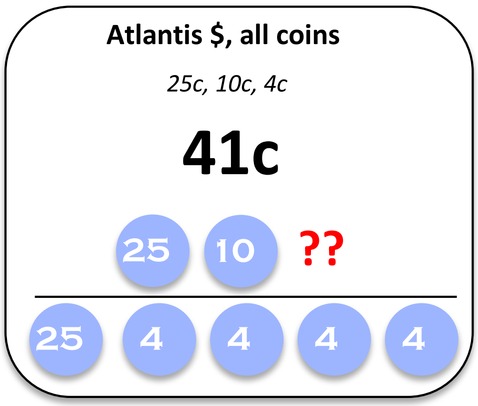
41p

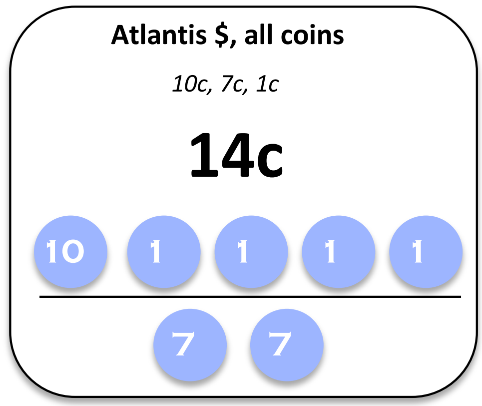
```
 41
-20
 21
-20
  1
 -1
  0
```

Greedy heuristic: Choose the largest coin possible at each step.

Greedy algorithms

The greedy heuristic isn't always applicable and, even when it is, it isn't guaranteed to find an optimal solution.



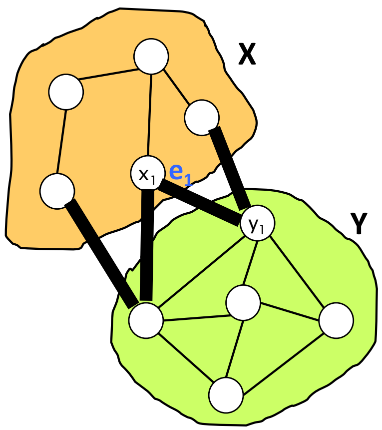


There is no general way of knowing whether or not the greedy heuristic is applicable or will lead to an optimal solution, but the problem should exhibit the **greedy choice property** and the **optimal substructure property**.

MST greedy property

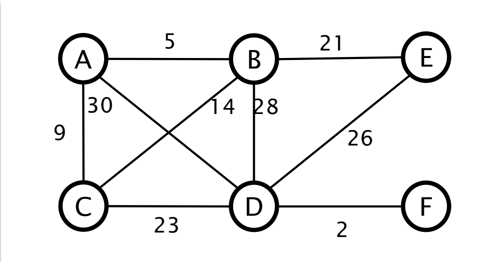
Important property of the MST problem:

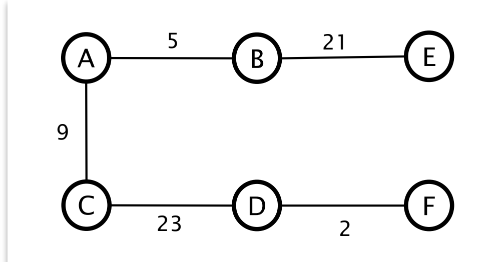
Given any division of the vertices of a graph into two sets, the minimum spanning tree contains the minimum cost edge that connects a vertex in one set to a vertex in the other set.

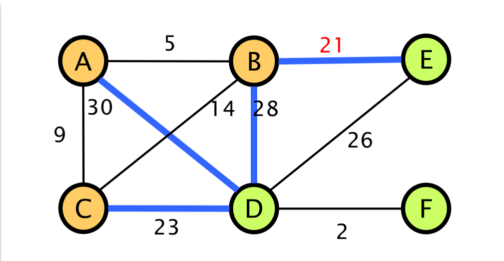


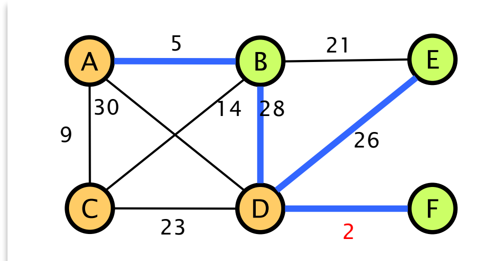
Proof: Call the minimum cost edge connecting the two sets X and Y e1, and assume that e1 is not in the MST. Then consider the graph formed by adding e1 to the purported MST. This graph has a cycle; in that cycle some other edge besides e1 must connect X and Y. Deleting this edge and adding e1 gives a lesser cost spanning tree, which contradicts the assumption that e1 is not in the MST.

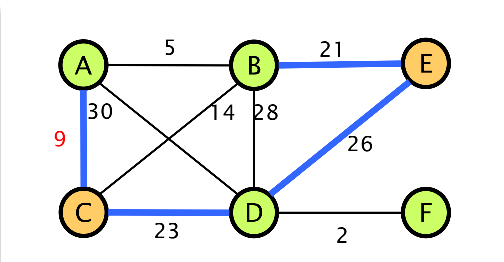
MST greedy property

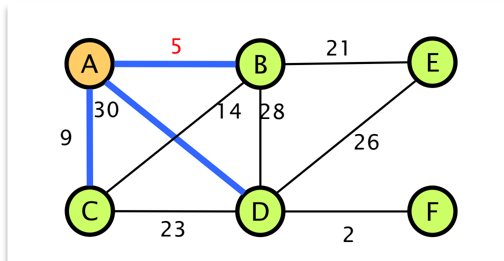










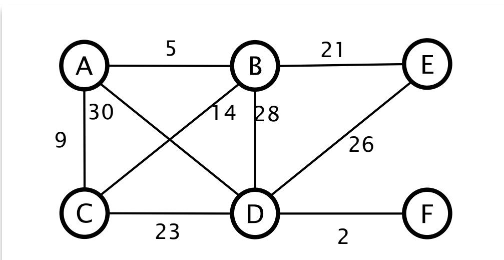


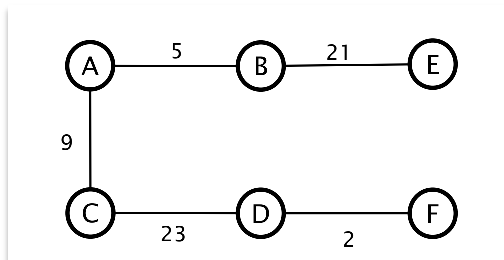
Prim’s algorithm

Prim’s algorithm

Starting from an arbitrary vertex, grow the MST by repeatedly choosing the cheapest edge that connects a vertex in the MST to a vertex that is not in the MST.

```
// pre-condition: The graph is connected.
Initialize the MST as an empty graph.
Arbitrarily select a vertex and add it to the MST.
Add all the edges in the graph to a collection E.
While (there are vertices not in the MST) {
    Select the edge of minimum weight that
        connects a vertex in the MST to a vertex
        that is not in the MST.
    Add this edge and new vertex to the MST.
}
```





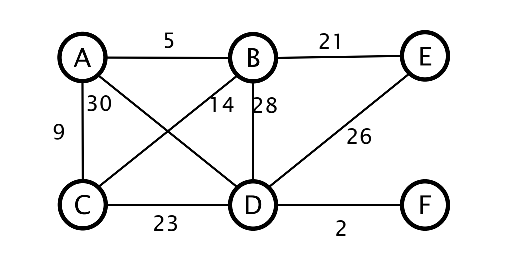
Kruskal’s algorithm

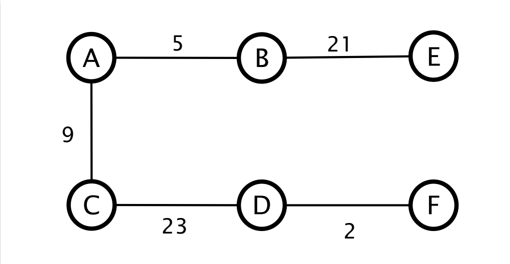
Kruskal's algorithm

Select the edges in ascending order of weight. Add each edge to the MST unless it would create a cycle. Stop when n-1 edges have been added.

```
// pre-condition: The graph is connected.
Initialize the MST with all vertices but no edges.
Add all the edges to a min heap.
Initialize count to 0.
Initialize N to the number of vertices in the graph.
While (count < N – 1) {
    Remove the next edge e from the min heap.
    If (adding e to the MST would not create a cycle) {
        Add e to the MST.
        count = count + 1
    }
}
```

How do we know if an edge (x, y) creates a cycle?





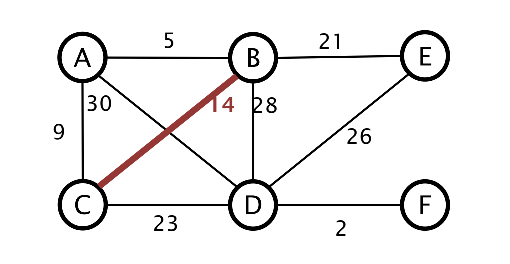
Kruskal's algorithm

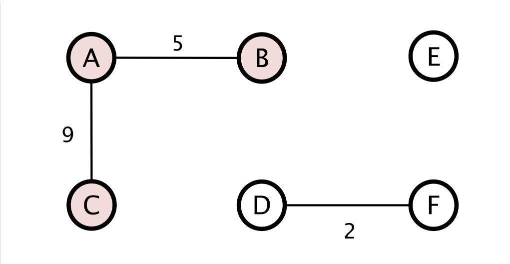
Select the edges in ascending order of weight. Add each edge to the MST unless it would create a cycle. Stop when n-1 edges have been added.

```
// pre-condition: The graph is connected.
Initialize the MST with all vertices but no edges.
Add all the edges to a min heap.
Initialize count to 0.
Initialize N to the number of vertices in the graph.
While (count < N – 1) {
    Remove the next edge e from the min heap.
    If (adding e to the MST would not create a cycle) {
        Add e to the MST.
        count = count + 1
    }
}
```

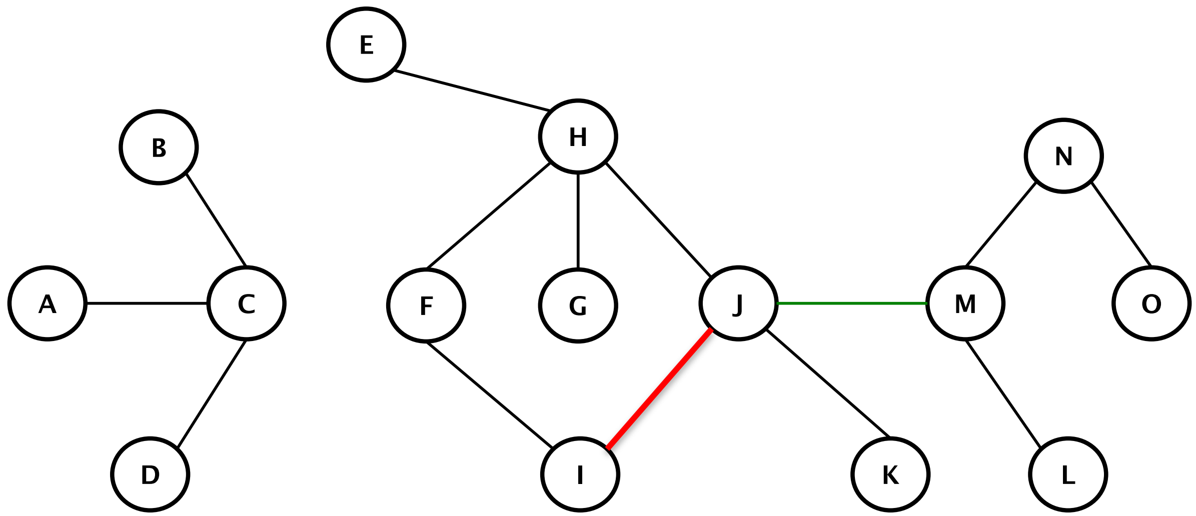
How do we know if an edge (x, y) creates a cycle?

x and y in same component





Union-find for detecting cycles




```
Add edge (x, y) if doesn't create a cycle
c1 = findComponent(x);
c2 = findComponent(y);
if (c1 != c2)
{
   add (x,y) to graph;
   union(c1, c2);
}
```

Add (I, J)? **No**

c1 = green
c2 = green

Add (M, J)? **Yes**

c1 = blue
c2 = green