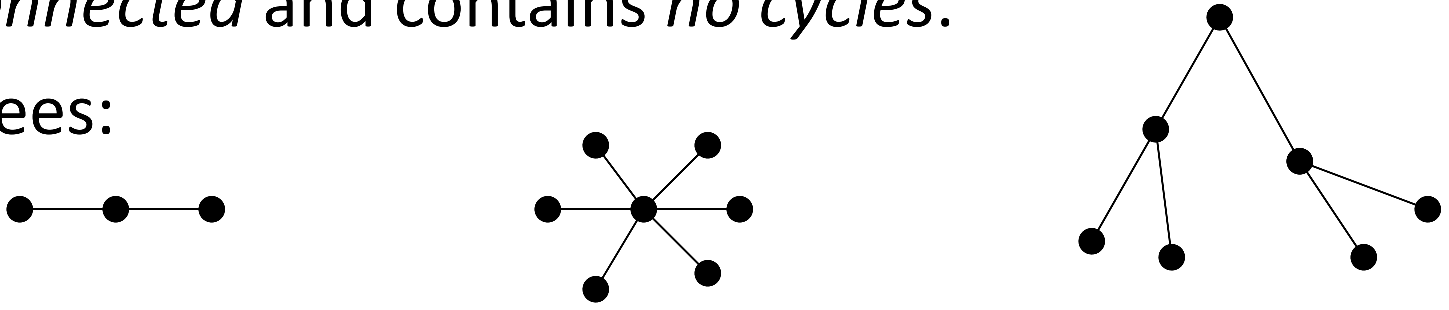
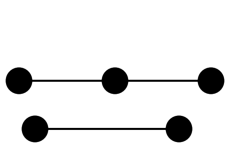
Spanning Trees

Tree

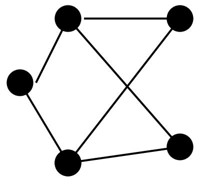
- We call an undirected graph a **tree** if the graph is *connected* and contains *no cycles*.
- Trees:



- Not Trees:



Not
connected



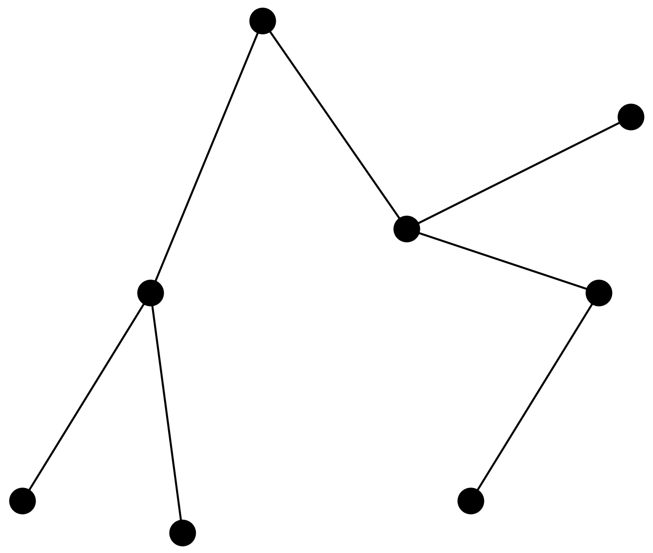
Has a cycle

Number of Vertices

- If a graph is a tree, then the number of edges in the graph is one less than the number of vertices.
- **A tree with *n* vertices has *n* – 1 edges.**
  - Each node has one parent except for the root.
    - Note: Any node can be the root here, as we are not dealing with rooted trees.

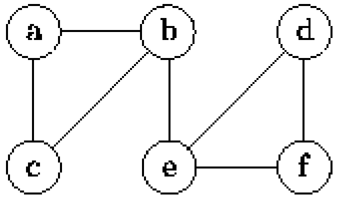
Connected Graph

- A **connected graph** is one in which there is *at least one path* between each pair of vertices.

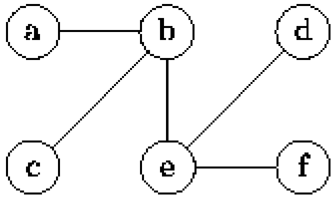


Spanning Tree

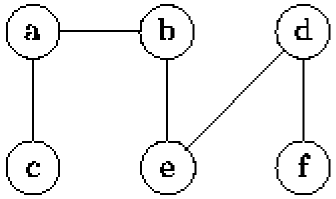
- In a tree there is always **exactly one path** from each vertex in the graph to any other vertex in the graph.
- A **spanning tree** for a graph is a subgraph that includes every vertex of the original, and is a tree.



(a) Graph G

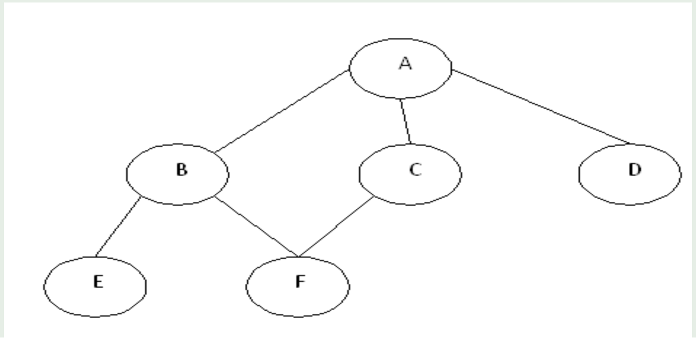
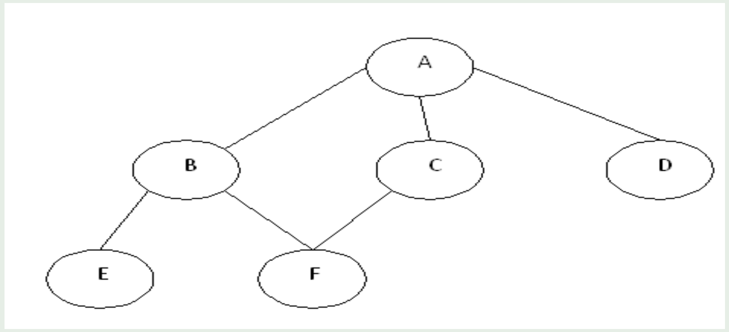


(b) Breadth-first spanning tree of G rooted at b



(c) Depth-first spanning tree of G rooted at c

Breadth First Search & Depth First Search

BFS	DFS
In this we visit the nodes level by level, so it will start with level 0, which is the root node then next, then the last level.	In this we visit the root node first then its children until it reaches the end node.
Queue is used to implement BFS. E.G.	Stack is used to implement DFS. E.G.
	

Its BFS will be A,B,C,D,E,F.

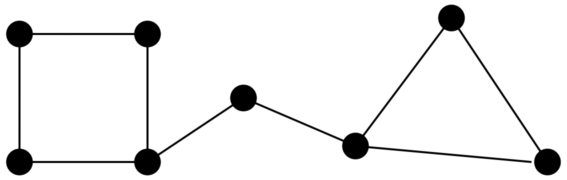
Its DFS will be A,B,E,F,C,D

Non-Connected Graphs

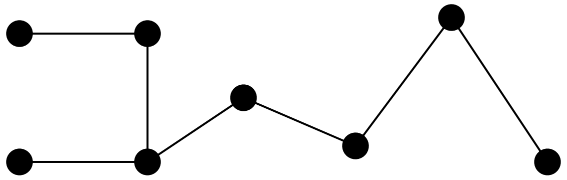
- If the graph is not connected, we get a spanning tree for each **connected component** of the graph.
  - That is we get a forest.

Finding a Spanning Tree

Find a spanning tree for the graph below.



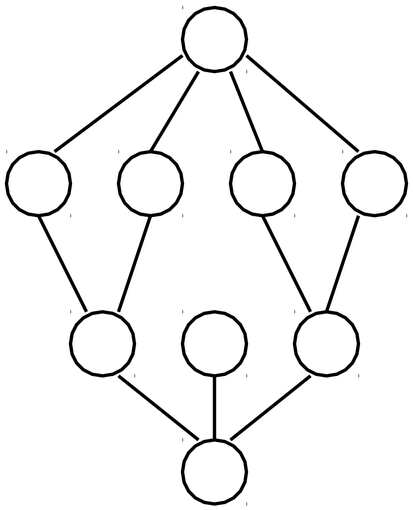
We could break the two cycles by removing a single edge from each. One of several possible ways to do this is shown below.

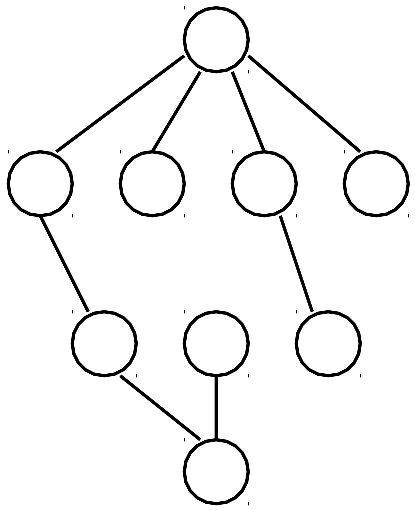


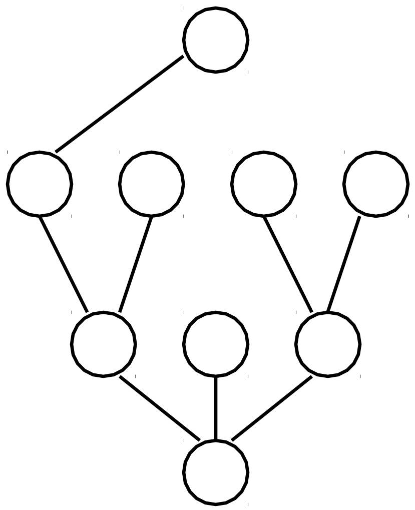
Was breadth-first or depth-first search (or neither) used to create this?

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

Minimum Spanning Tree

- A spanning tree that has minimum total weight is called a **minimum spanning tree** for the graph.
  - Technically it is a minimum-weight spanning tree.
- If all edges have the same weight, breadth-first search or depth-first search will yield minimum spanning trees.
  - For the rest of this discussion, we assume the edges have weights associated with them.

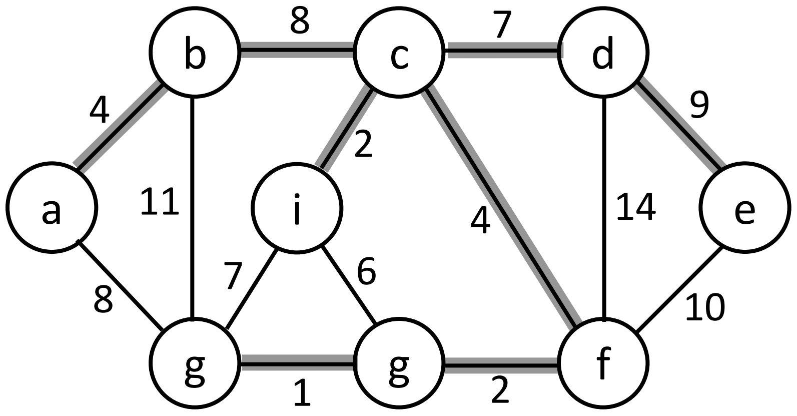
Note, we are strictly dealing with undirected graphs here, for directed graphs we would want to find the optimum branching or arborescence of the directed graph.

Minimum Spanning Tree

- Minimum-cost spanning trees have many applications.
  - Building cable networks that join *n* locations with minimum cost.
  - Building a road network that joins *n* cities with minimum cost.
  - Obtaining an independent set of circuit equations for an electrical network.
  - In pattern recognition minimal spanning trees can be used to find noisy pixels.

Minimum Spanning Trees

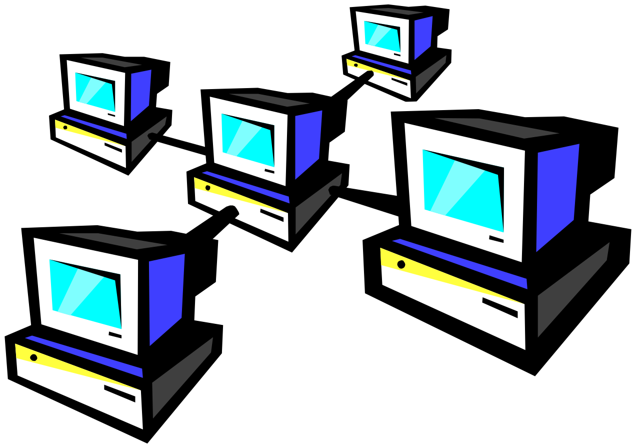
- Spanning Tree
  - A tree (i.e., connected, acyclic graph) which contains all the vertices of the graph
- Minimum Spanning Tree
  - Spanning tree with the **minimum sum of weights**

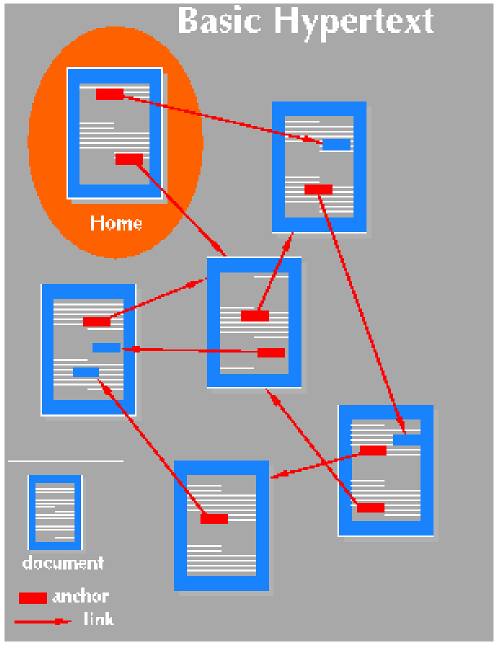


- Spanning forest
  - If a graph is not connected, then there is a spanning tree for each connected component of the graph

Applications of MST

– Find the least expensive way to connect a set of cities, terminals, computers, etc.

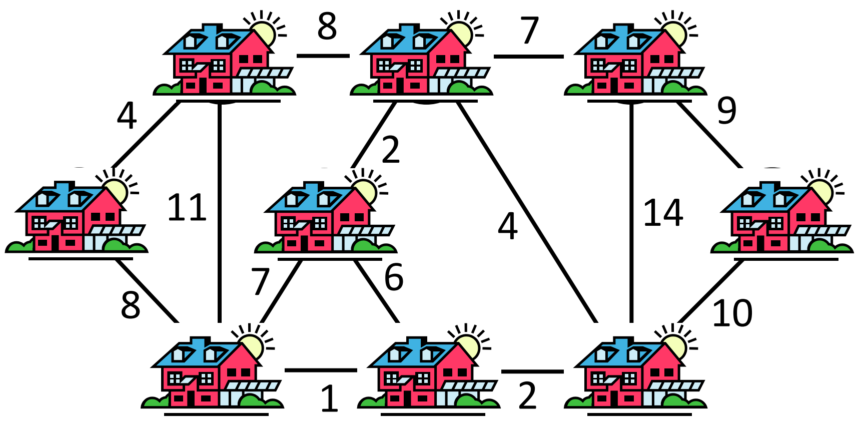




Example

Problem

- A town has a set of houses and a set of roads
- A road connects 2 and only 2 houses
- A road connecting houses u and v has a repair cost w(u, v)



Goal: Repair enough (and no more) roads such that:

1. Everyone stays connected

   i.e., can reach every house from all other houses

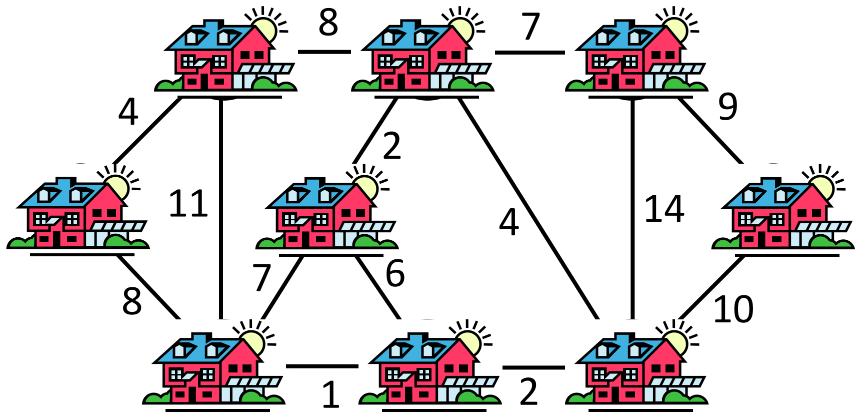
2. Total repair cost is minimum

Minimum Spanning Trees

- A connected, undirected graph:
  - Vertices = houses, Edges = roads
- A **weight** $w(u, v)$ on each edge $(u, v) \in E$

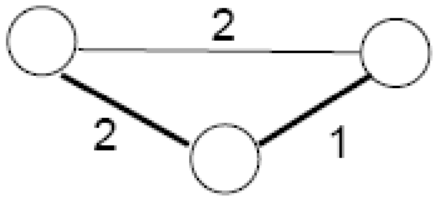
Find $T \subseteq E$ such that:

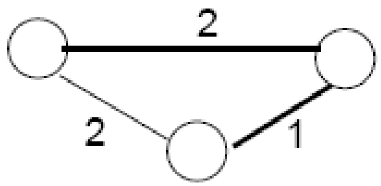
1. T connects all vertices
2. $w(T) = \Sigma_{(u,v)\in T}\, w(u, v)$ is minimized



Properties of Minimum Spanning Trees

- Minimum spanning tree is **not** unique

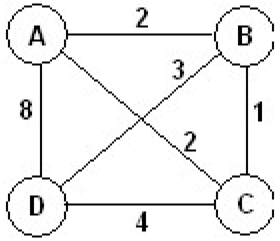




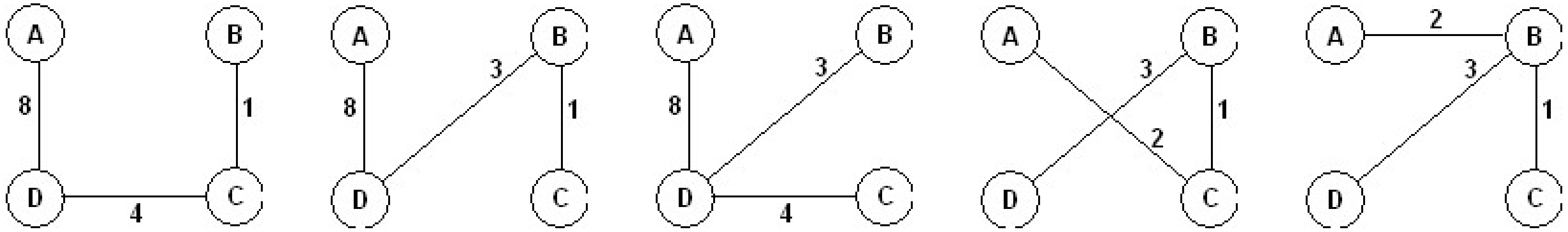
- MST has no cycles – see why:
  - We can take out an edge of a cycle, and still have the vertices connected while reducing the cost
- # of edges in a MST:
  - |V| - 1

Minimum Spanning Tree

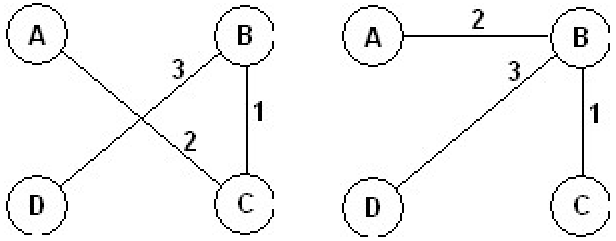
- Consider this graph.



- It has 20 spanning trees. Some are:



- There are two minimum-cost spanning trees, each with a cost of 6:



Minimum Spanning Tree

- Brute Force option:
  1. For all possible spanning trees
     1. Calculate the sum of the edge weights
     2. Keep track of the tree with the minimum weight.
- Step i) requires N-1 time, since each tree will have exactly N-1 edges.
- If there are M spanning trees, then the total cost will O(MN).
- Consider a complete graph, with N(N-1) edges. How big can M be?

Brute Force MST

- For a complete graph, it has been shown that there are N^{N-2} possible spanning trees!
- Alternatively, given N items, you can build N^{N-2} distinct trees to connect these items.
- Note, for a lattice (like your grid implementation), the number of spanning trees is $\boldsymbol{O}(e^{1.167N})$.

Minimum Spanning Tree

- There are many approaches to computing a minimum spanning tree. We could try to detect cycles and remove edges, but the two algorithms we will study build them from the bottom-up in a *greedy* fashion.
- **Kruskal's Algorithm – *starts with a forest of single node trees*** and then adds the edge with the minimum weight to connect two components.
- **Prim's Algorithm – *starts with a single vertex*** and then adds the minimum edge to extend the spanning tree.

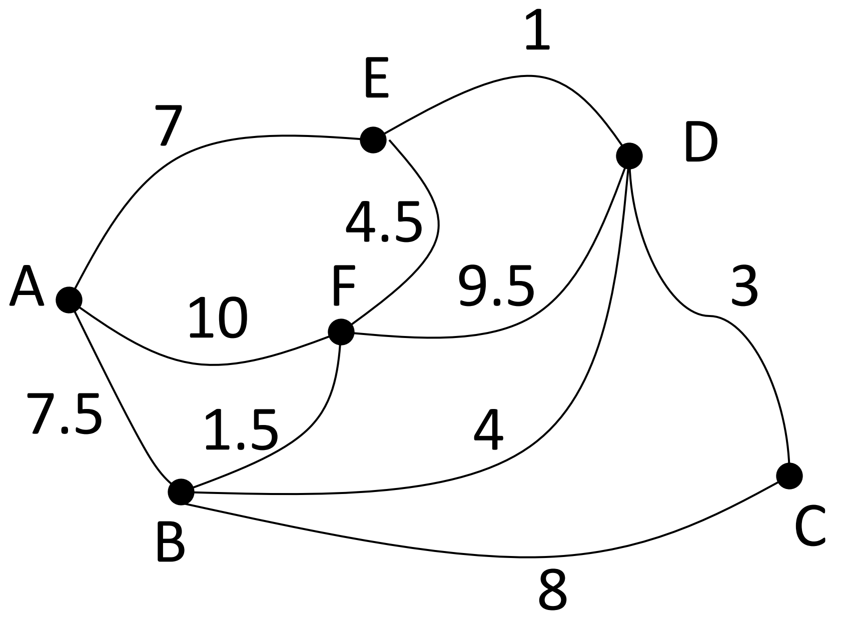
Kruskal’s Algorithm

- Greedy algorithm to choose the edges as follows.

Step 1	**First edge: choose any edge with the minimum weight.**
Step 2	**Next edge: choose any edge with minimum weight from *those not yet selected*. (The subgraph can look disconnected at this stage.)**
Step 3	**Continue to choose edges of minimum weight from those not yet selected, except *do not select any edge that creates a cycle* in the subgraph.**
Step 4	**Repeat step 3 until the subgraph connects all vertices of the original graph.**

Kruskal’s Algorithm

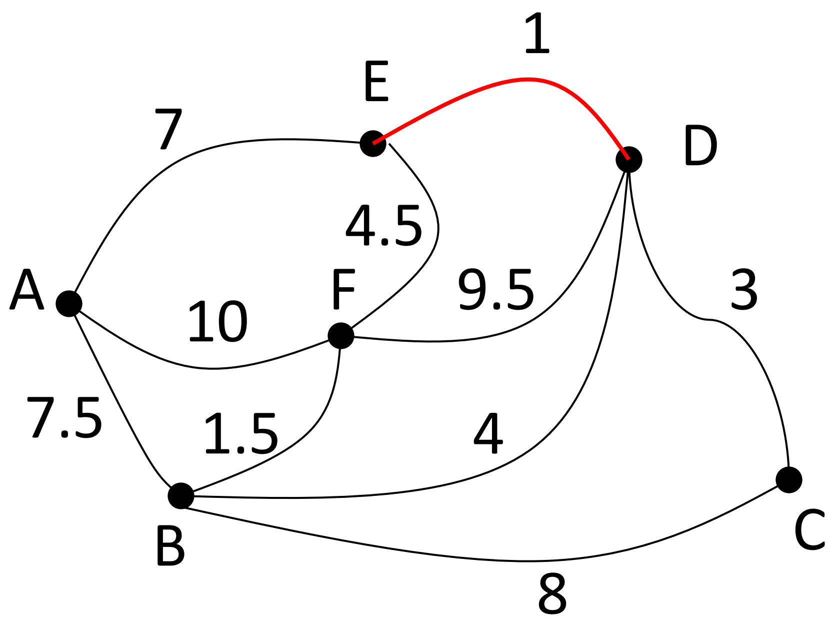
Use Kruskal’s algorithm to find a minimum spanning tree for the graph.



Kruskal’s Algorithm

Solution

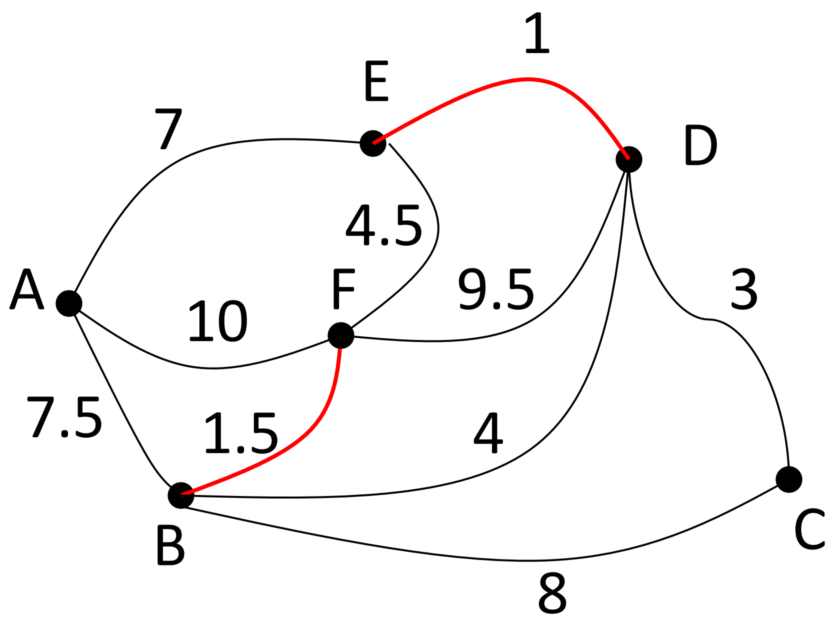
First, choose ED (the smallest weight).



Kruskal’s Algorithm

Solution

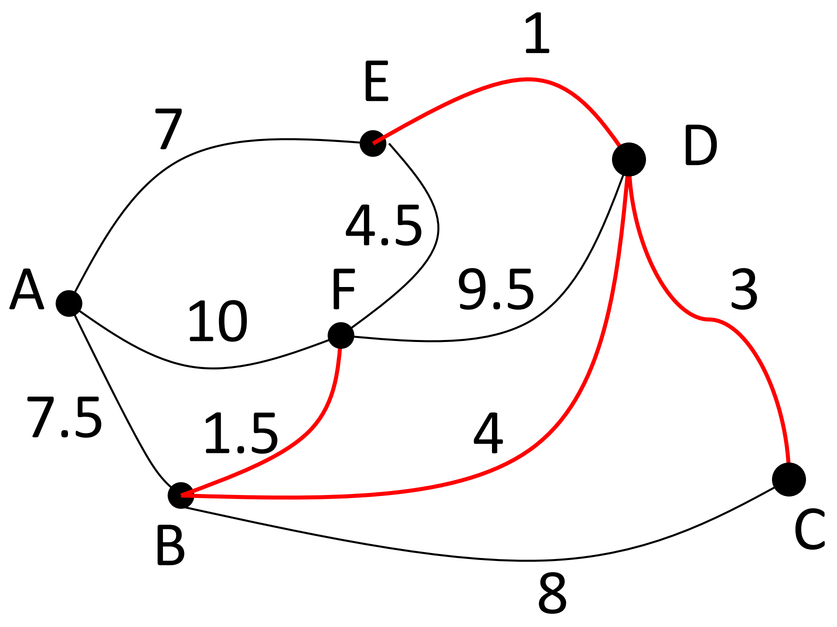
Now choose BF (the smallest remaining weight).



Kruskal’s Algorithm

Solution

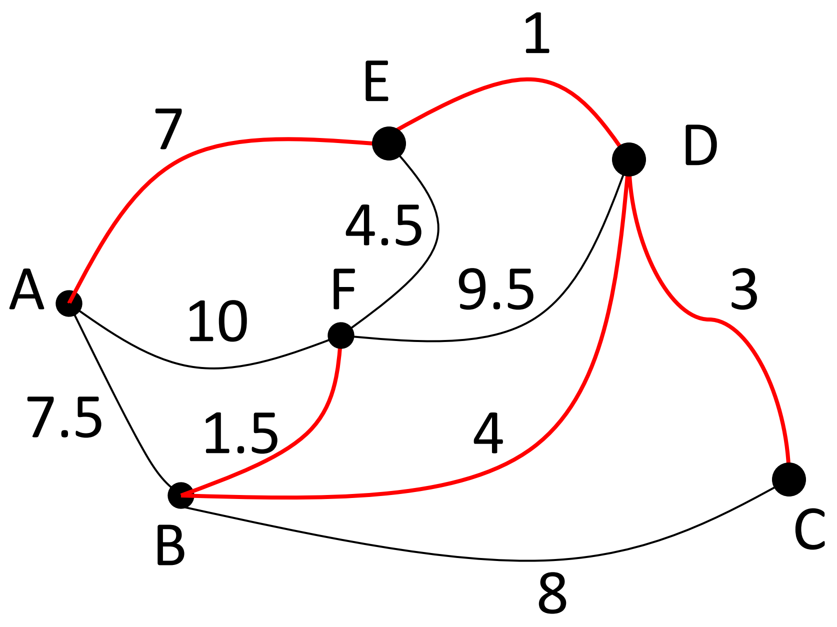
Now CD and then BD.



Kruskal’s Algorithm

Solution

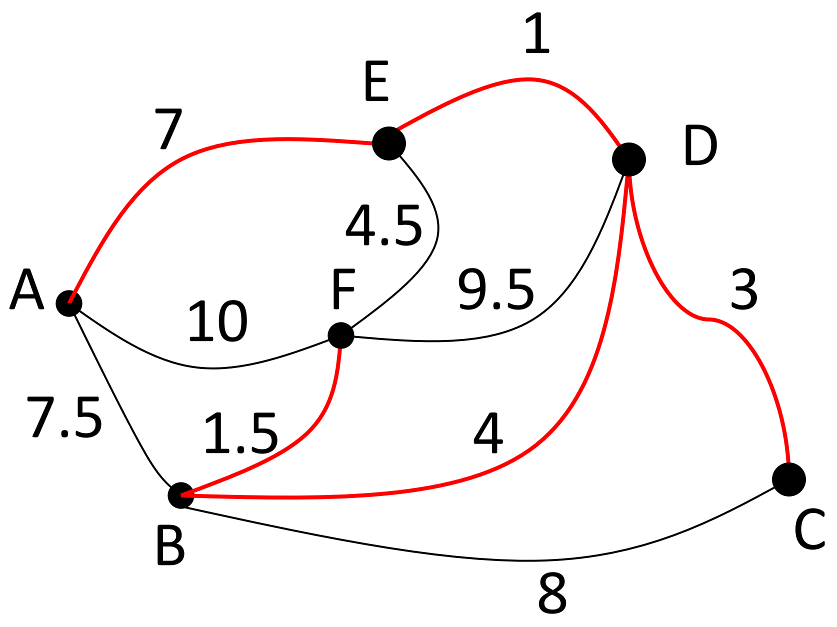
Note EF is the smallest remaining, but that would create a cycle. Choose AE and we are done.



Kruskal's Algorithm

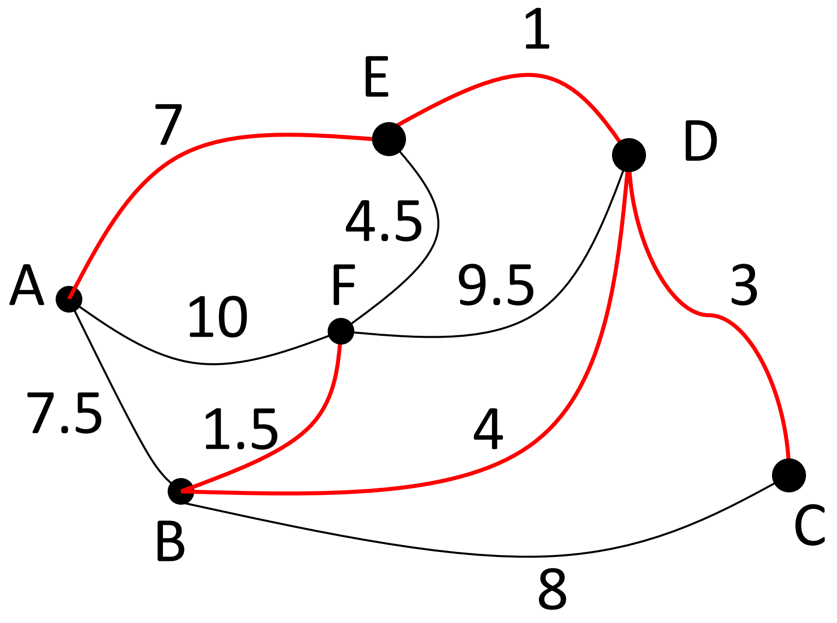
Solution

The total weight of the tree is 16.5.



Kruskal’s Algorithm

- Some questions:
  1. How do we know we are finished?
  2. How do we check for cycles?



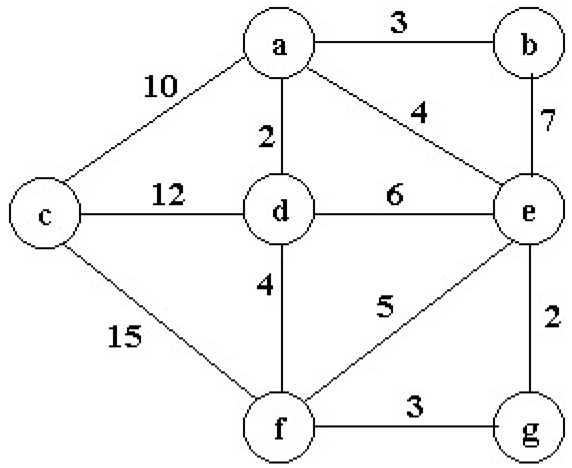
Kruskal’s Algorithm

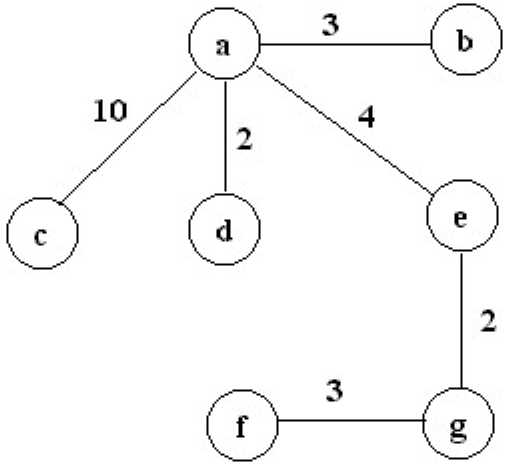
```
Build a priority queue (min-based) with all of the edges of G.
T = $\phi$ ;
while(queue is not empty){
   get minimum edge e from priorityQueue;
   if(e does not create a cycle with edges in T)
       add e to T;
}
return T;
```

Kruskal’s Algorithm

edge	ad	eg	ab	fg	ae	df	ef	de	be	ac	cd	cf
weight	2	2	3	3	4	4	5	6	7	10	12	15
insertion status	√	√	√	√	√	x	x	x	x	√	x	x
insertion order	1	2	3	4	5					6		

- Trace of Kruskal's algorithm for the undirected, weighted graph:





The minimum cost is: 24

Kruskal’s Algorithm – Time complexity

- Steps
  - Initialize forest $O(\ |V|\ )$
  - Sort edges $O(\ |E|\log|E|\ )$
    - Check edge for cycles $O(\ |V|\ )$ x
    - Number of edges $O(\ |V|\ )$ $O(\ |V|^2\ )$
  - Total $O(\ |V|+|E|\log|E|+|V|^2\ )$
  - Since $|E| = O(\ |V|^2\ )$ $O(\ |V|^2\log|V|\ )$

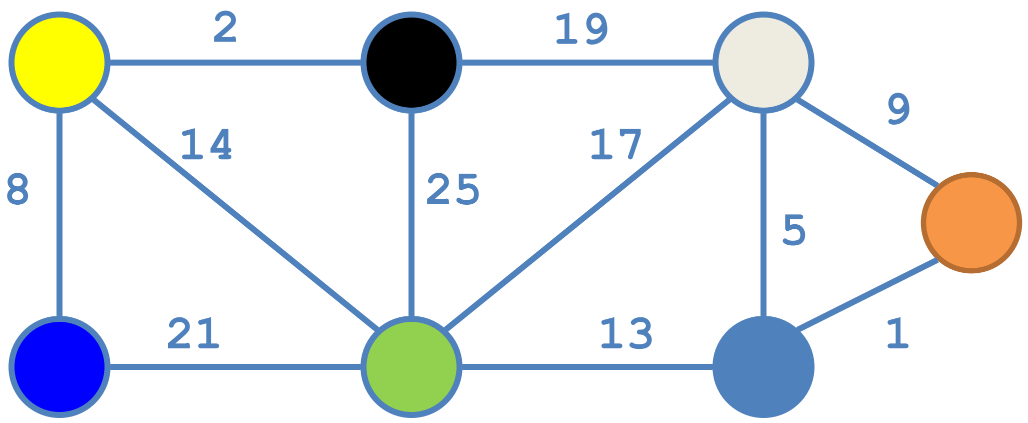
  - Thus we would class MST as $O(\ n^2\log n\ )$ for a graph with n vertices
  - This is an ***upper bound***, some improvements on this are known.

Kruskal's Algorithm

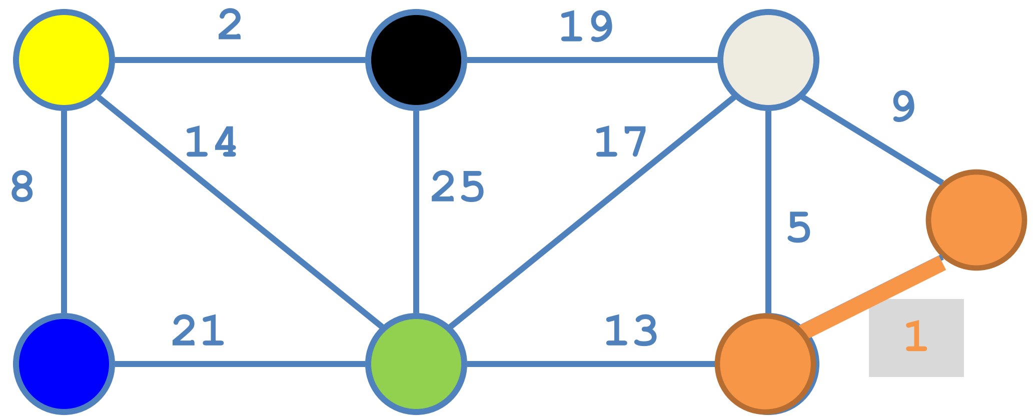
- Another implementation is based on sets.

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E by increasing edge weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T U {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

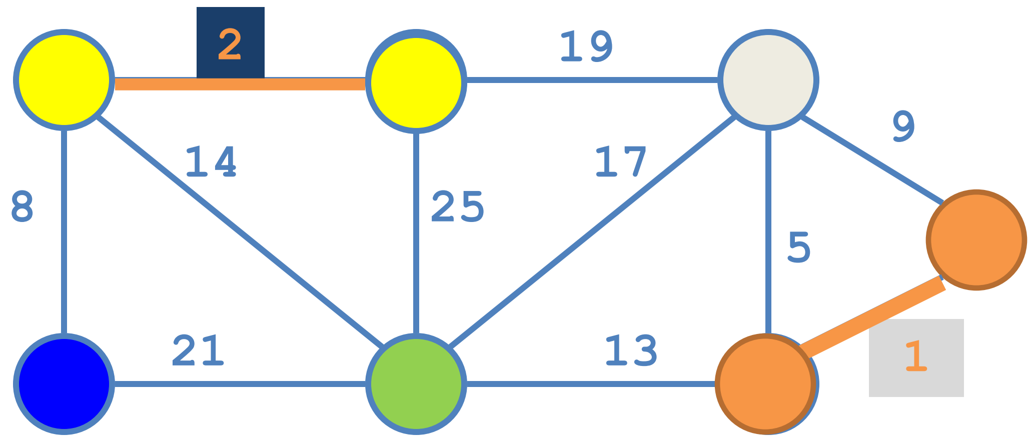
Kruskal's Algorithm



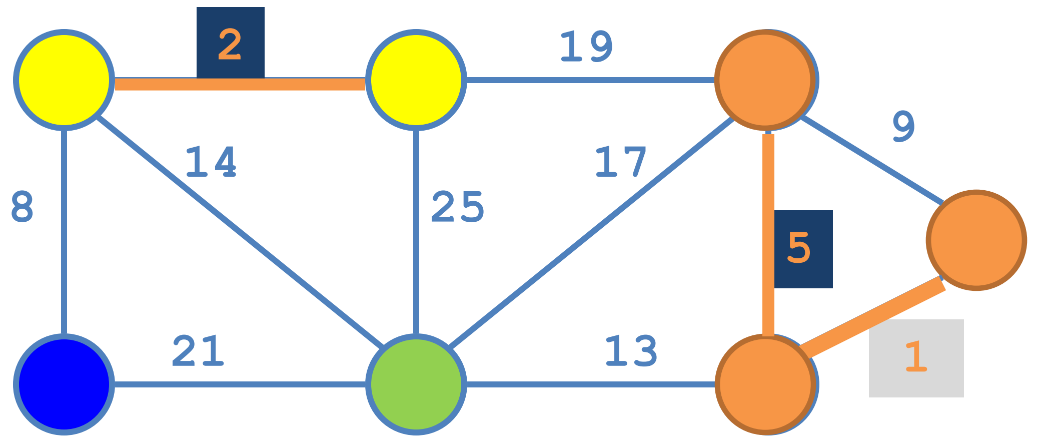
Kruskal’s Algorithm



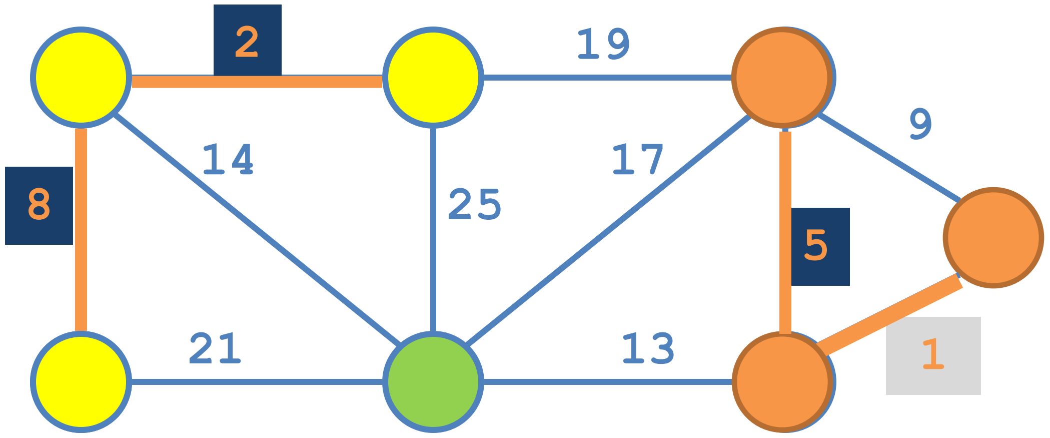
Kruskal's Algorithm



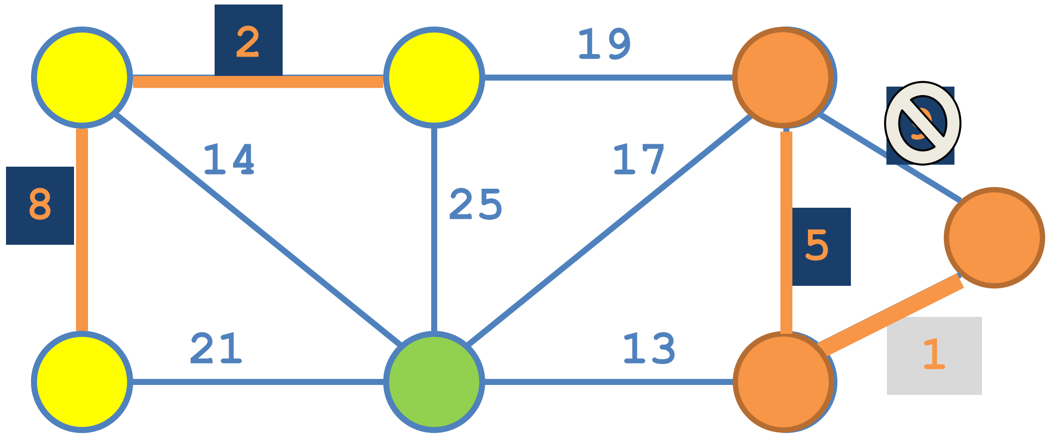
Kruskal’s Algorithm



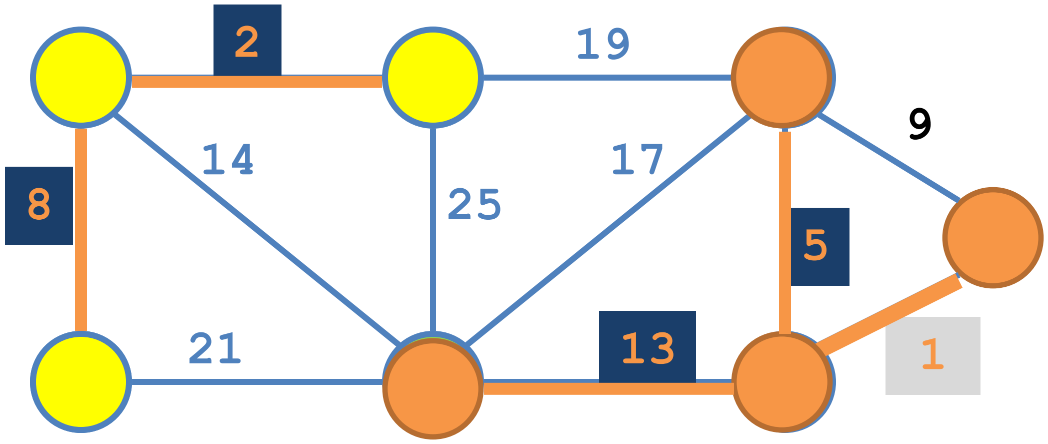
Kruskal’s Algorithm



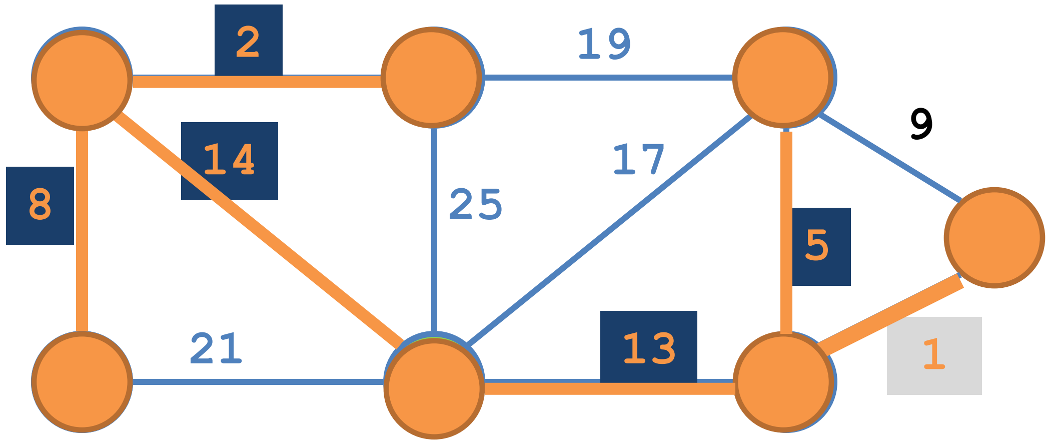
Kruskal's Algorithm



Kruskal’s Algorithm



Kruskal's Algorithm



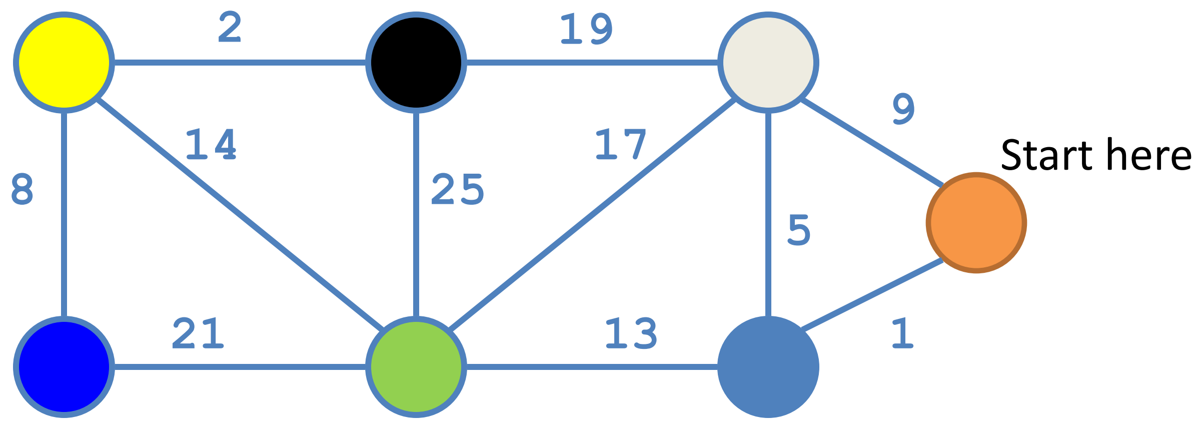
Prim’s Algorithm

- Prim’s algorithm finds a minimum cost spanning tree by selecting edges from the graph one-by-one as follows:
- It starts with a tree, T, consisting of a single starting vertex, x.
- Then, it finds the shortest edge emanating from x that connects T to the rest of the graph (i.e., a vertex not in the tree T).
- It adds this edge and the new vertex to the tree T.
- It then picks the shortest edge emanating from the revised tree T that also connects T to the rest of the graph and repeats the process.

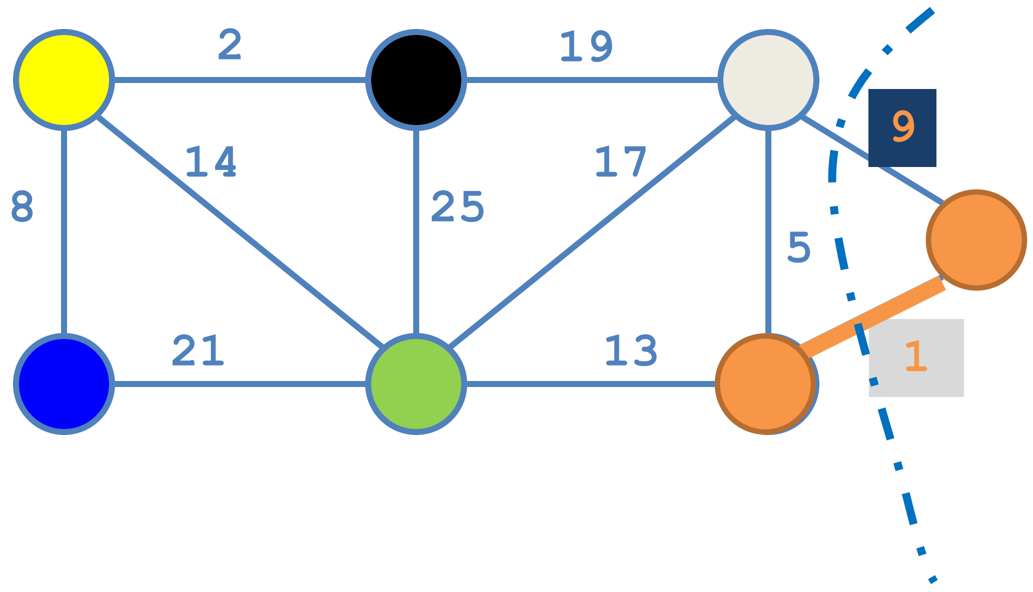
Prim’s Algorithm Abstract

```
Consider a graph G=(V, E);
Let T be a tree consisting of only the starting
   vertex x;
while (T has fewer than I V I vertices)
{
      find a smallest edge connecting T to G-T;
      add it to T;
}
```

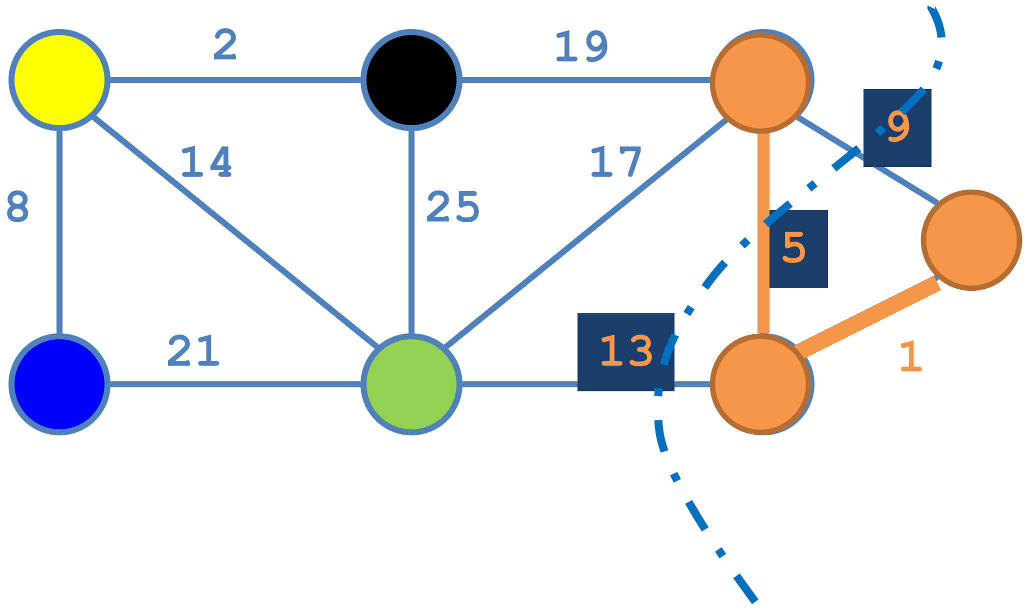
Prim’s Algorithm



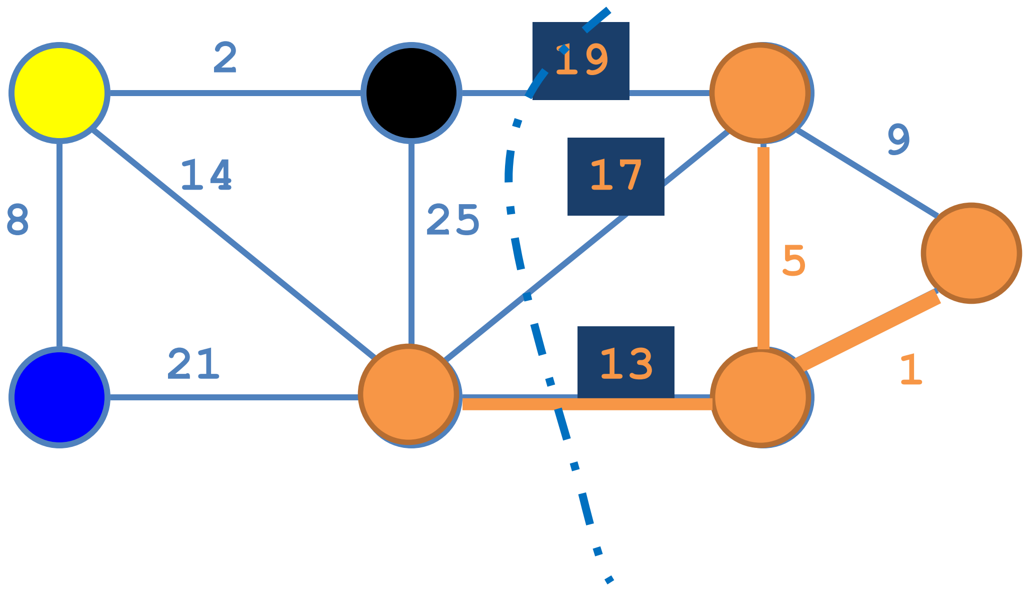
Prim’s Algorithm



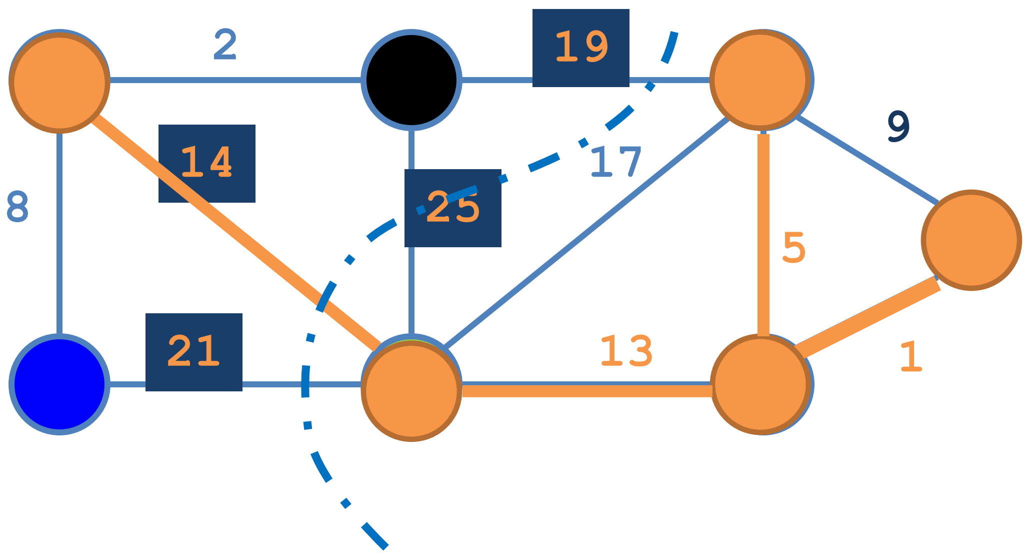
Prim’s Algorithm



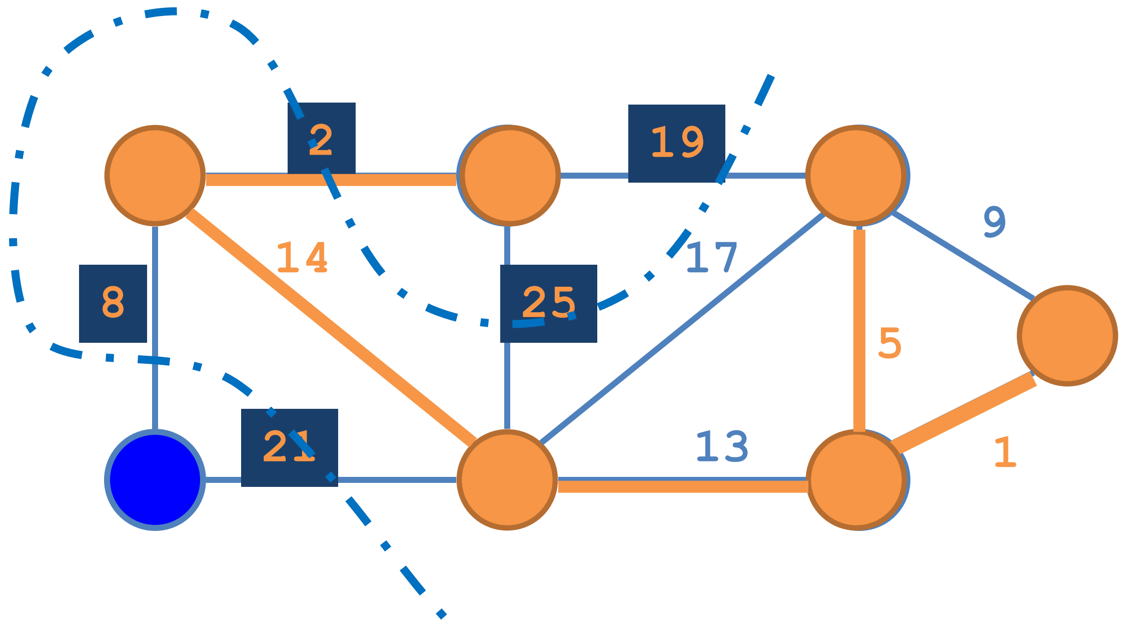
Prim’s Algorithm



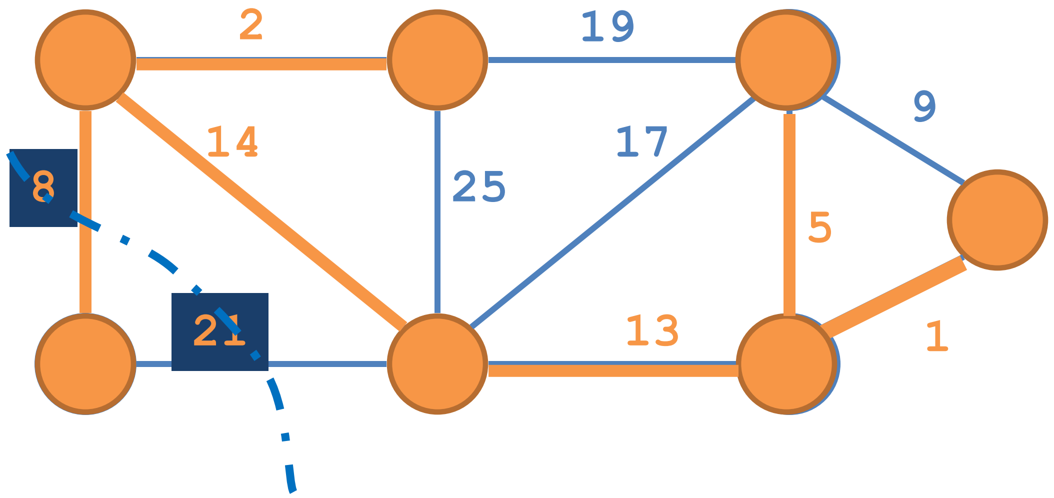
Prim's Algorithm



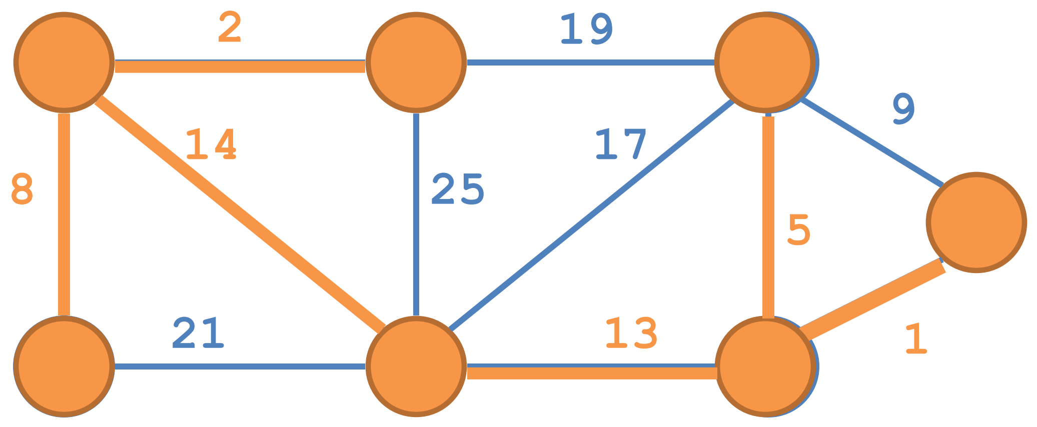
Prim's Algorithm



Prim’s Algorithm

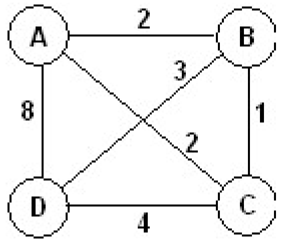


Prim’s Algorithm

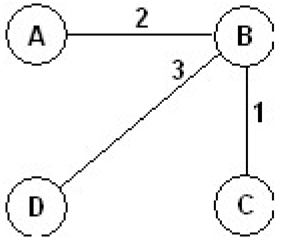


Prim’s and Kruskal’s Algorithms

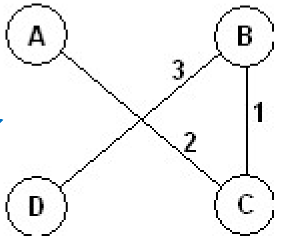
- It is not necessary that Prim's and Kruskal's algorithm generate the same minimum-cost spanning tree.



- For example for the graph shown on the right:



- Kruskal's algorithm results in the following minimum cost spanning tree:
  - The same tree is generated by Prim's algorithm if the start vertex is any of: A, B, or D.



- However if the start vertex is C the minimum cost spanning tree generated by Prim’s algorithm is:

Implementation Details

- Prim’s Algorithm from your book is horrible from a SE stand-point!

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```

Q is a priority queue
key is the internal priorities in Q!
changing key, changes the queue.

Implementation Details

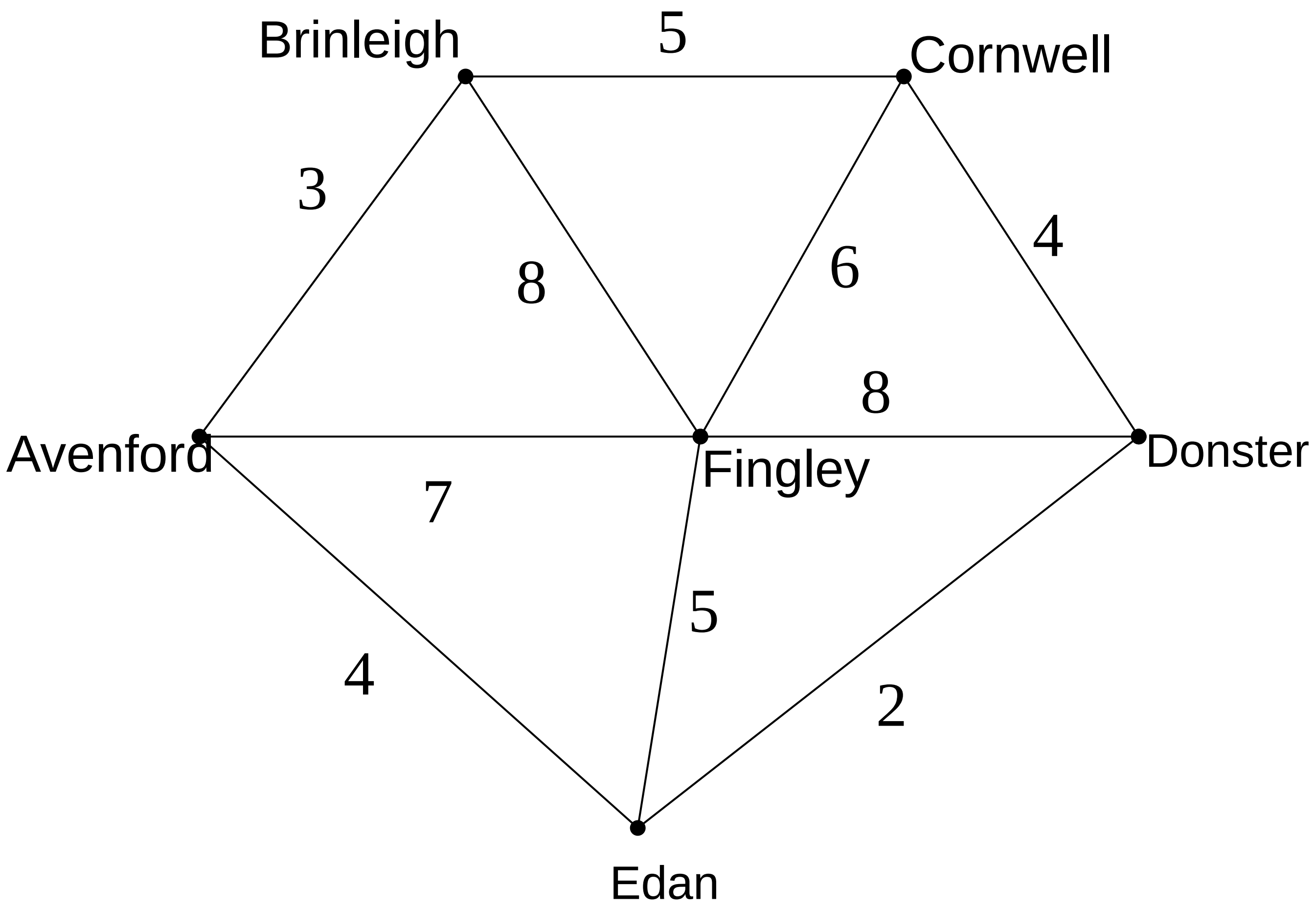
- ***Why queue the vertices,*** rather than the newly discovered edges?

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    DecreaseKey(r, 0);
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                DecreaseKey(v, w(u,v));
```

A Fibonacci Heap allows this to be done in O(1) time.

Prim’s algorithm with an Adjacency Matrix

A cable company want to connect five villages to their network which currently extends to the market town of Avenford. What is the minimum length of cable needed?

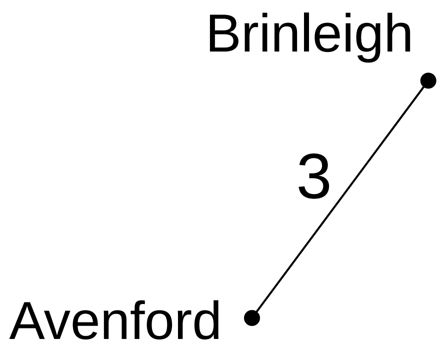


Prim’s algorithm with an Adjacency Matrix

	A	B	C	D	E	F
A	-	3	-	-	4	7
B	3	-	5	-	-	8
C	-	5	-	4	-	6
D	-	-	4	-	2	8
E	4	-	-	2	-	5
F	7	8	6	8	5	-

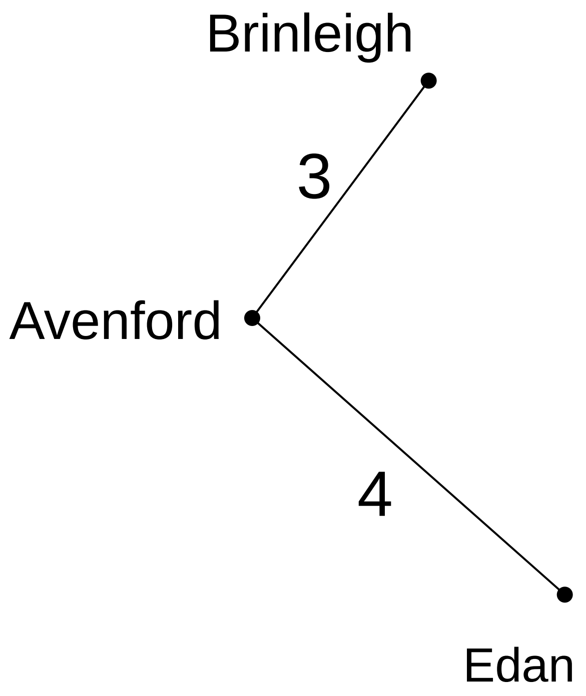
- Start at vertex A. Label column A “1” .
- Delete row A
- Select the smallest entry in column A (AB, length 3)

	1 A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
B	3	-	5	-	-	8
C	-	5	-	4	-	6
D	-	-	4	-	2	8
E	4	-	-	2	-	5
F	7	8	6	8	5	-



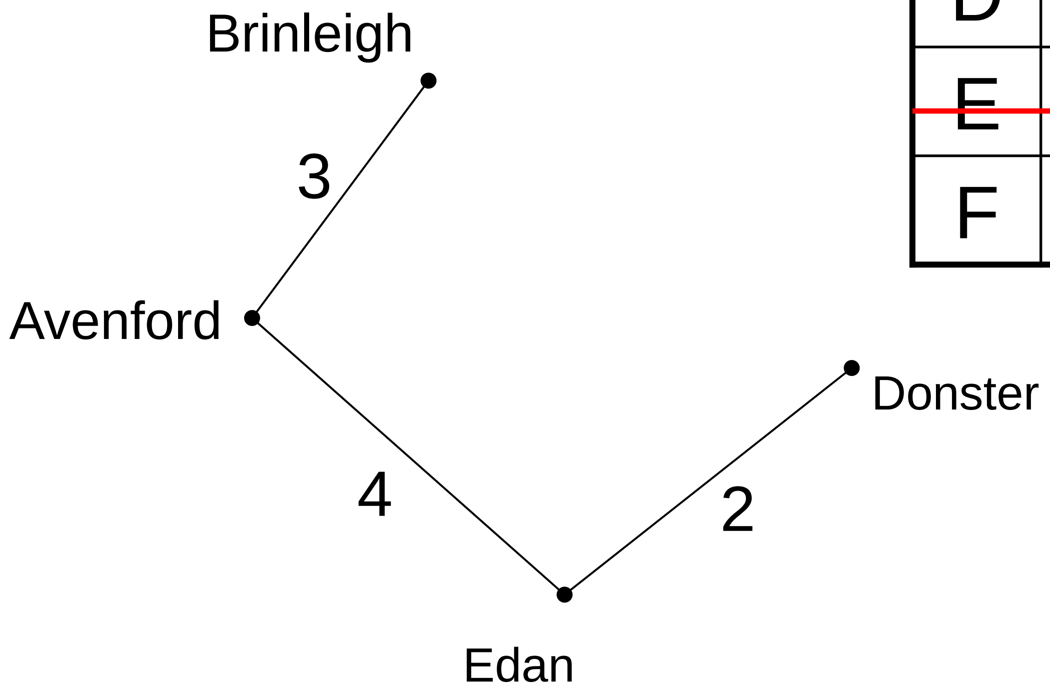
- Label column B “2”
- Delete row B
- Select the smallest uncovered entry in either column A or column B (AE, length 4)

	1	2				
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	~~3~~	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
C	-	5	-	4	-	6
D	-	-	4	-	2	8
E	4	-	-	2	-	5
F	7	8	6	8	5	-



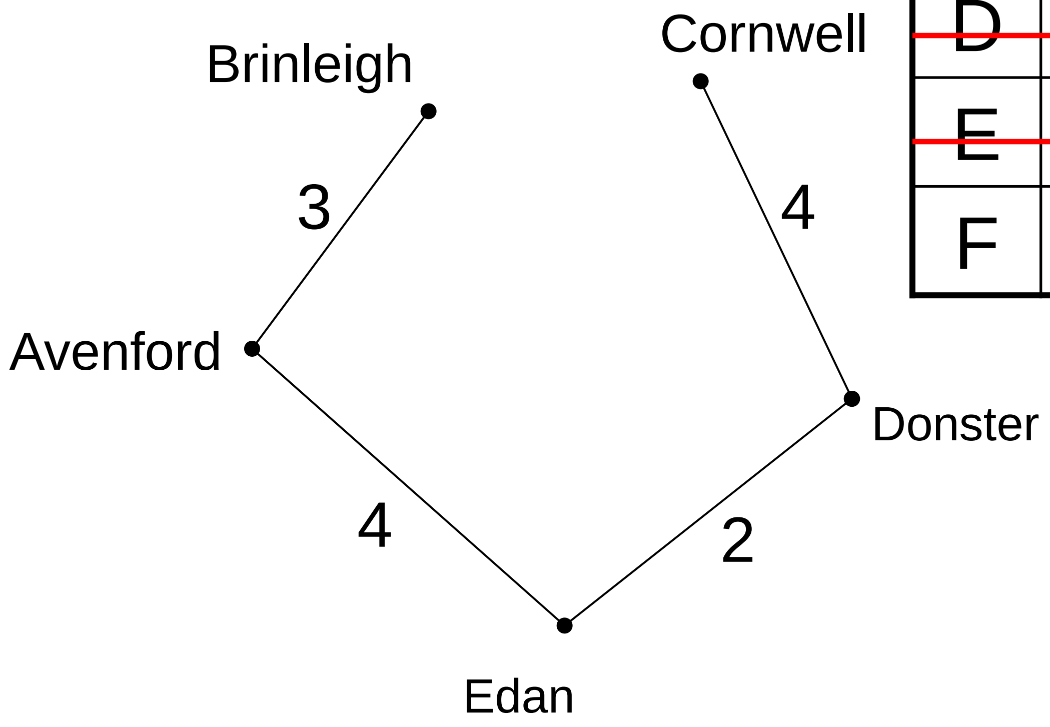
- Label column E "3"
- Delete row E
- Select the smallest uncovered entry in either column A, B or E (ED, length 2)

	1	2			3	
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	~~3~~	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
C	-	5	-	4	-	6
D	-	-	4	-	2	8
~~E~~	~~4~~	~~-~~	~~-~~	~~2~~	~~-~~	~~5~~
F	7	8	6	8	5	-



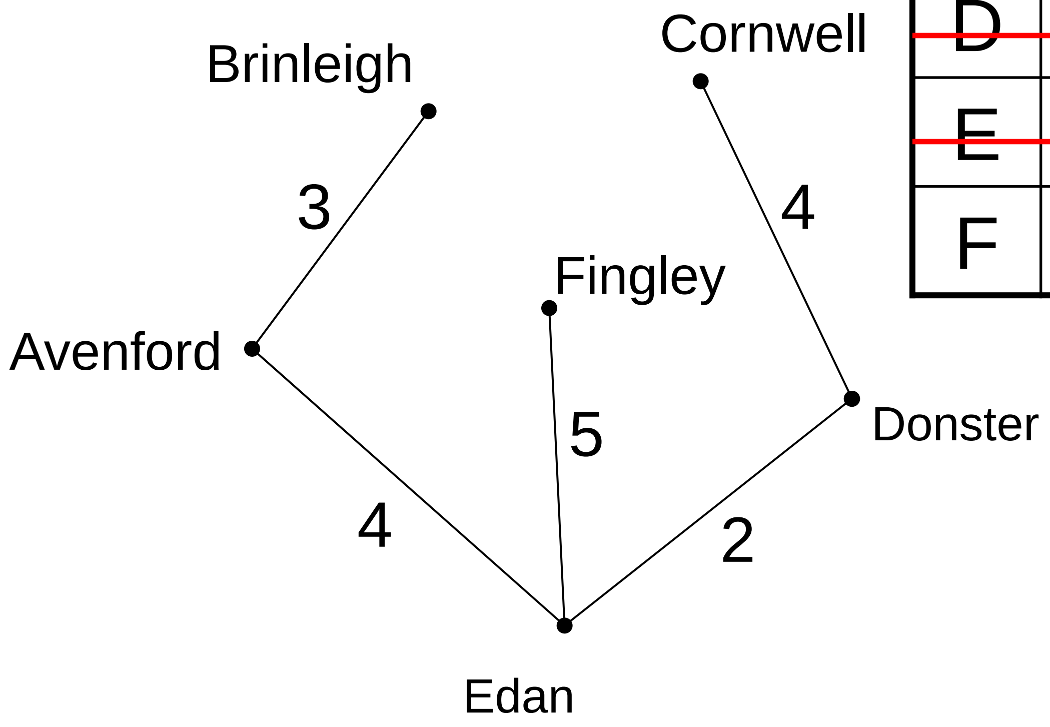
- Label column D “4”
- Delete row D
- Select the smallest uncovered entry in either column A, B, D or E (DC, length 4)

	1	2		4	3	
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	3	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
C	-	5	-	4	-	6
~~D~~	~~-~~	~~-~~	~~4~~	~~-~~	2	~~8~~
~~E~~	4	~~-~~	~~-~~	~~2~~	~~-~~	~~5~~
F	7	8	6	8	5	-



- Label column C “5”
- Delete row C
- Select the smallest uncovered entry in either column A, B, D, E or C (EF, length 5)

	1	2	5	4	3	
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	~~3~~	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
~~C~~	~~-~~	~~5~~	~~-~~	~~4~~	~~-~~	~~6~~
~~D~~	~~-~~	~~-~~	~~4~~	~~-~~	2	~~8~~
~~E~~	~~4~~	~~-~~	~~-~~	~~2~~	~~-~~	~~5~~
F	7	8	6	8	5	-

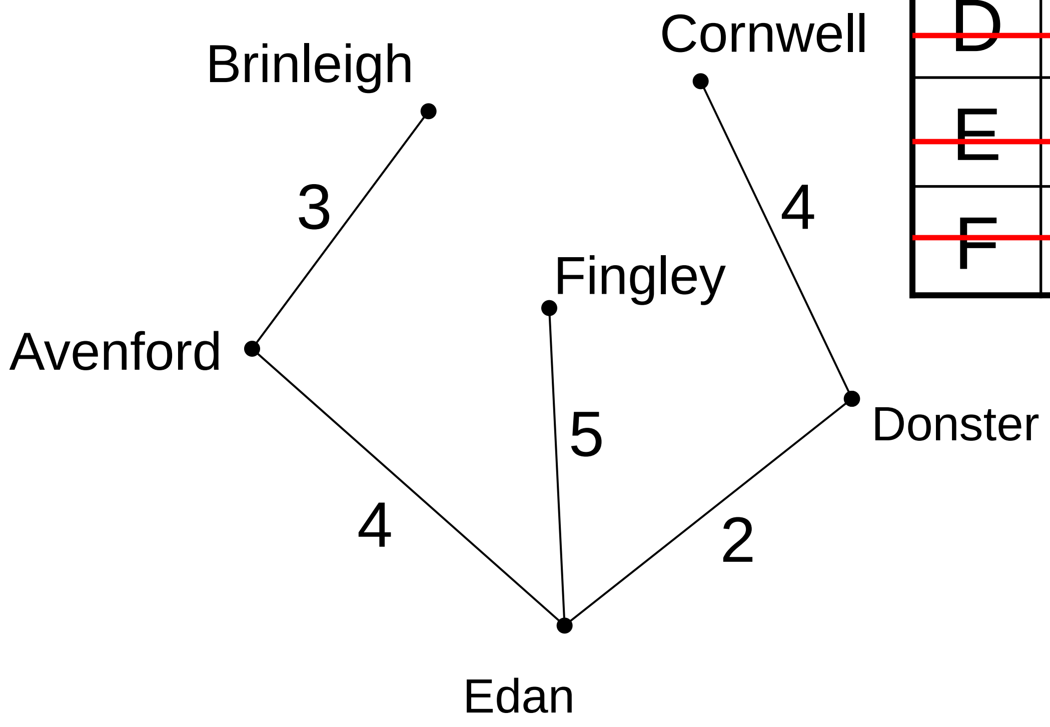


FINALLY

•Label column F "6"

•Delete row F

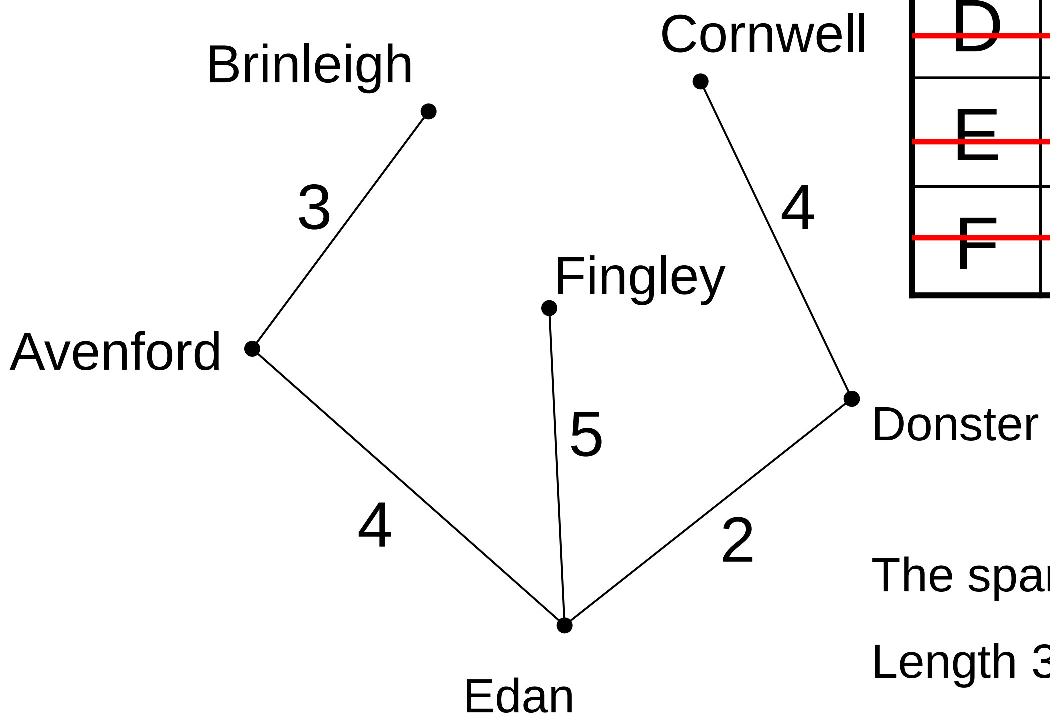
	1	2	5	4	3	6
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	3	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
~~C~~	~~-~~	~~5~~	~~-~~	4	~~-~~	~~6~~
~~D~~	~~-~~	~~-~~	~~4~~	~~-~~	2	~~8~~
~~E~~	4	~~-~~	~~-~~	~~2~~	~~-~~	~~5~~
~~F~~	~~7~~	~~8~~	~~6~~	~~8~~	5	~~-~~



FINALLY

- Label column F "6"
- Delete row F

	1	2	5	4	3	6
	A	B	C	D	E	F
~~A~~	~~-~~	~~3~~	~~-~~	~~-~~	~~4~~	~~7~~
~~B~~	3	~~-~~	~~5~~	~~-~~	~~-~~	~~8~~
~~C~~	~~-~~	~~5~~	~~-~~	4	~~-~~	~~6~~
~~D~~	~~-~~	~~-~~	~~4~~	~~-~~	2	~~8~~
~~E~~	4	~~-~~	~~-~~	~~2~~	~~-~~	~~5~~
~~F~~	~~7~~	~~8~~	~~6~~	~~8~~	5	~~-~~



The spanning tree is shown in the diagram

Length 3 + 4 + 4 + 2 + 5 = 18Km

Kruskal vs. Prim

- Both are Greedy algorithms
  - Both take the next minimum edge
  - Both are optimal (find the global min)
- Different sets of edges considered
  - Kruskal – all edges
  - Prim – Edges from Tree nodes to rest of G.
- Both need to check for cycles
  - Kruskal – set containment and union.
  - Prim – Simple boolean.
- Both can terminate early
  - Kruskal – when |V|-1 edges are added.
  - Prim – when |V| nodes are added (or |V|-1 edges).
- Both are $O(|E| \log |V|)$
  - Prim can be $O(|E| + |V| \log |V|)$ w/ Fibonacci Heaps
  - Prim with an adjacency matrix is $O(|V|^2)$.

DIFFERENCE

PRIM'S ALGORITHM

- In this, at every step edges of minimum weight that are incident to a vertex already in the tree and not forming a cycle is choosen .

KRUSAL'S ALGOTITHM

- In this, at every step edges of minimum weight that are not necessarily incident to a vertex already in the tree and not forming a cycle is choosen .