Minimum Spanning Tree

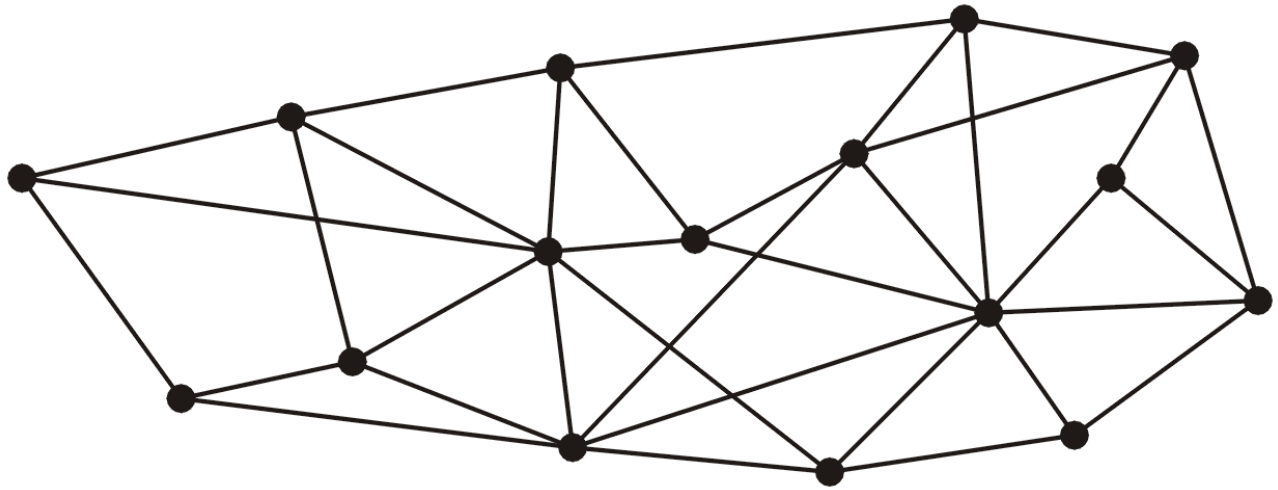
Spanning trees

Given a connected graph with $|V| = n$ vertices, a spanning tree is defined a collection of $n - 1$ edges which connect all n vertices

- The n vertices and $n - 1$ edges define a connected sub-graph

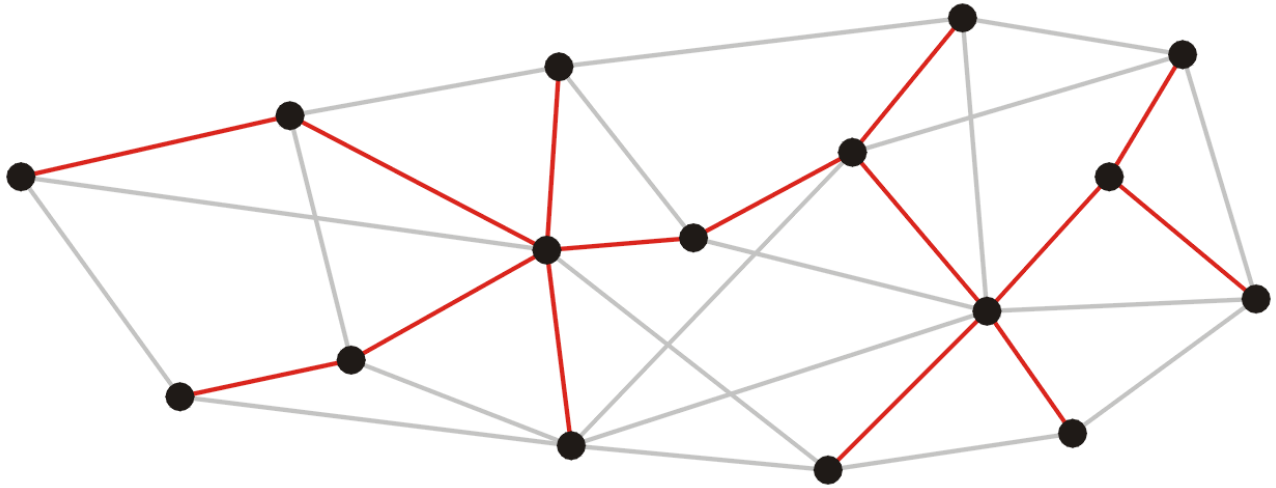
A spanning tree is not necessarily unique

This graph has 16 vertices and 35 edges



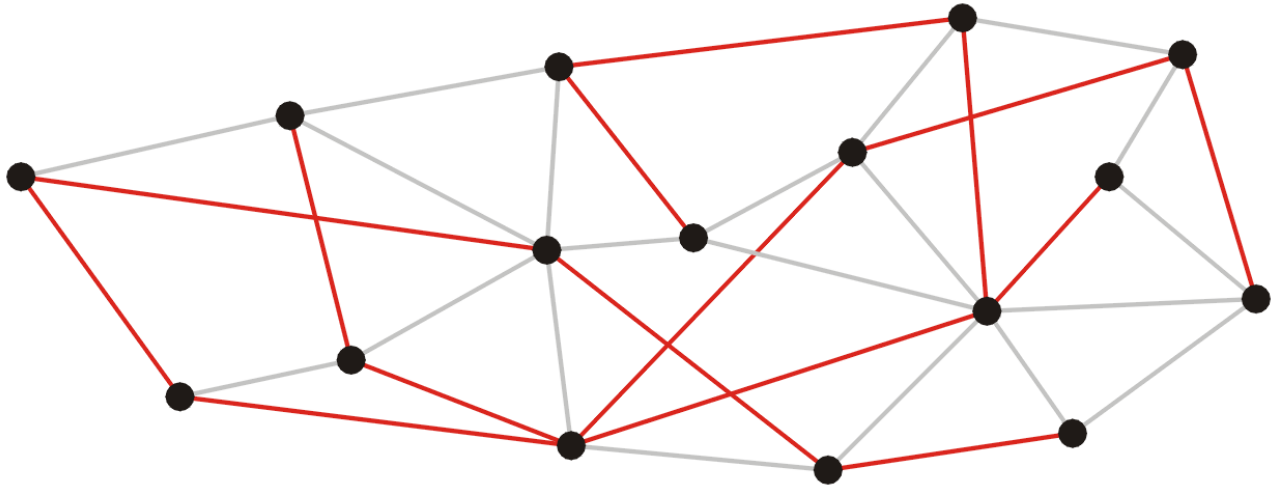
Spanning trees

These 15 edges form a minimum spanning tree



Spanning trees

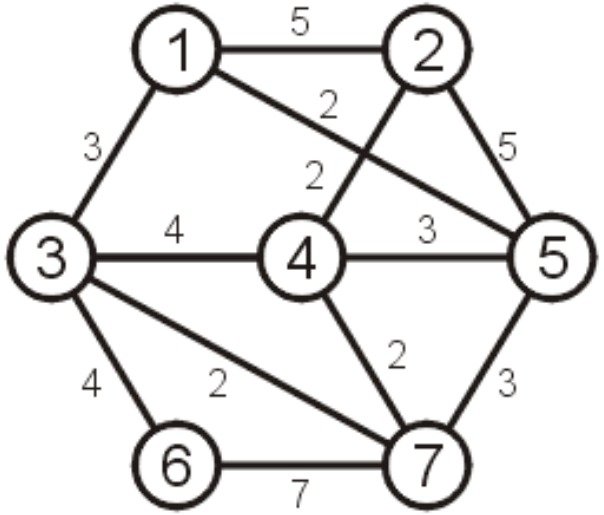
As do these 15 edges:

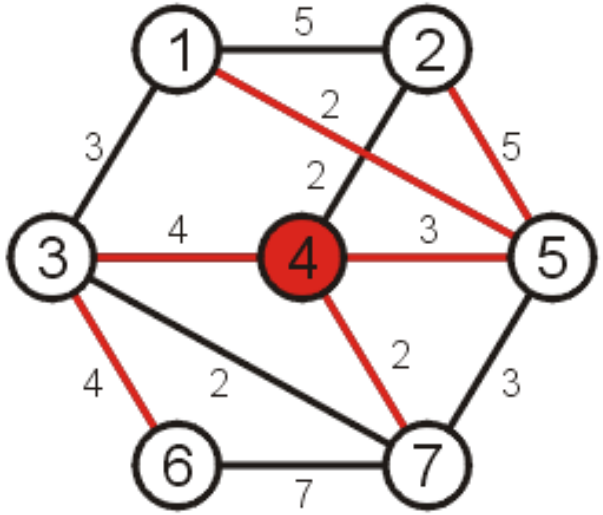


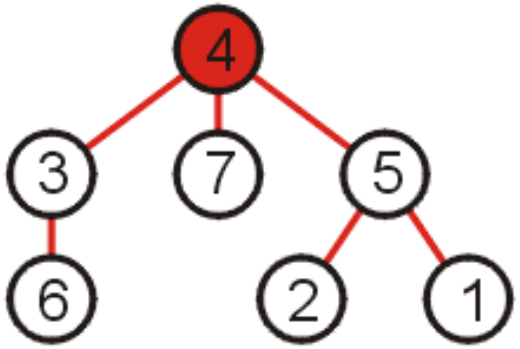
Spanning trees on weighted graphs

The weight of a spanning tree is the sum of the weights on all the edges which comprise the spanning tree

- The weight of this spanning tree is 20





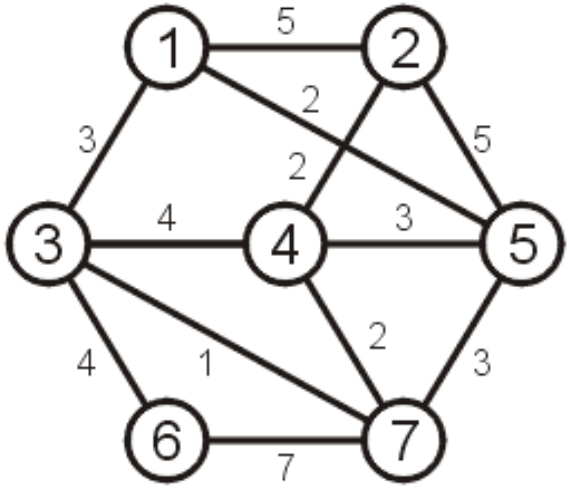


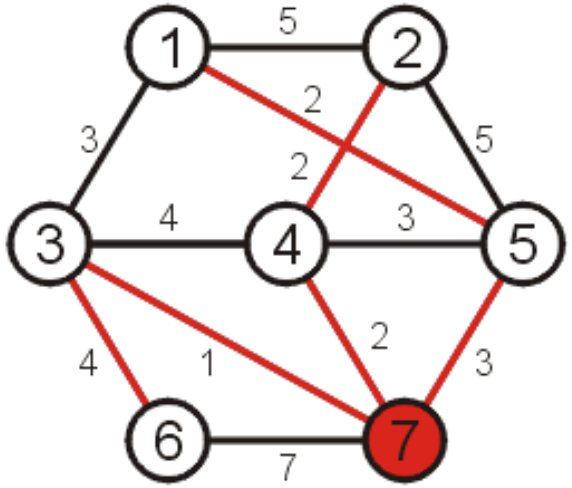
Minimum Spanning Trees

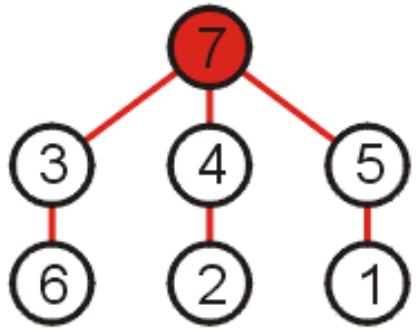
Which spanning tree minimizes the weight?

- Such a tree is termed a ***minimum spanning tree***

The weight of this spanning tree is 14







Unweighted graphs

Observation

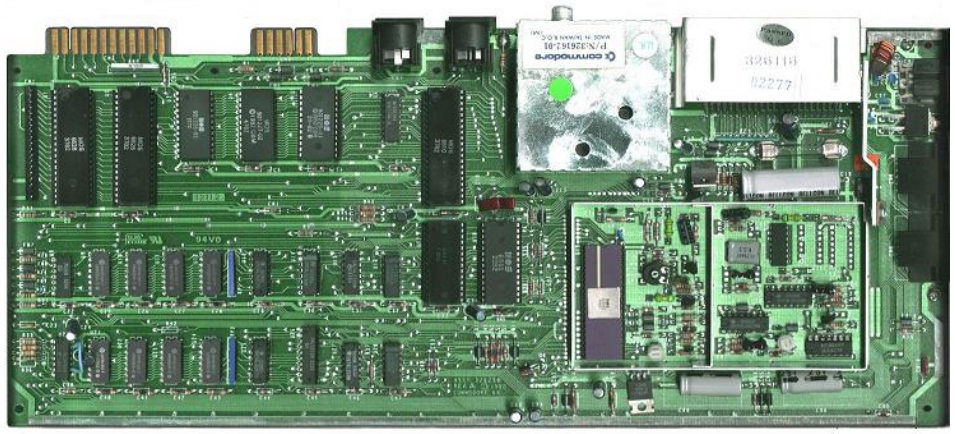
- In an unweighted graph, we nominally give each edge a weight of 1
- Consequently, all minimum spanning trees have weight $|V| - 1$

Application

Consider supplying power to

- All circuit elements on a board
- A number of loads within a building

A minimum spanning tree will give the lowest-cost solution



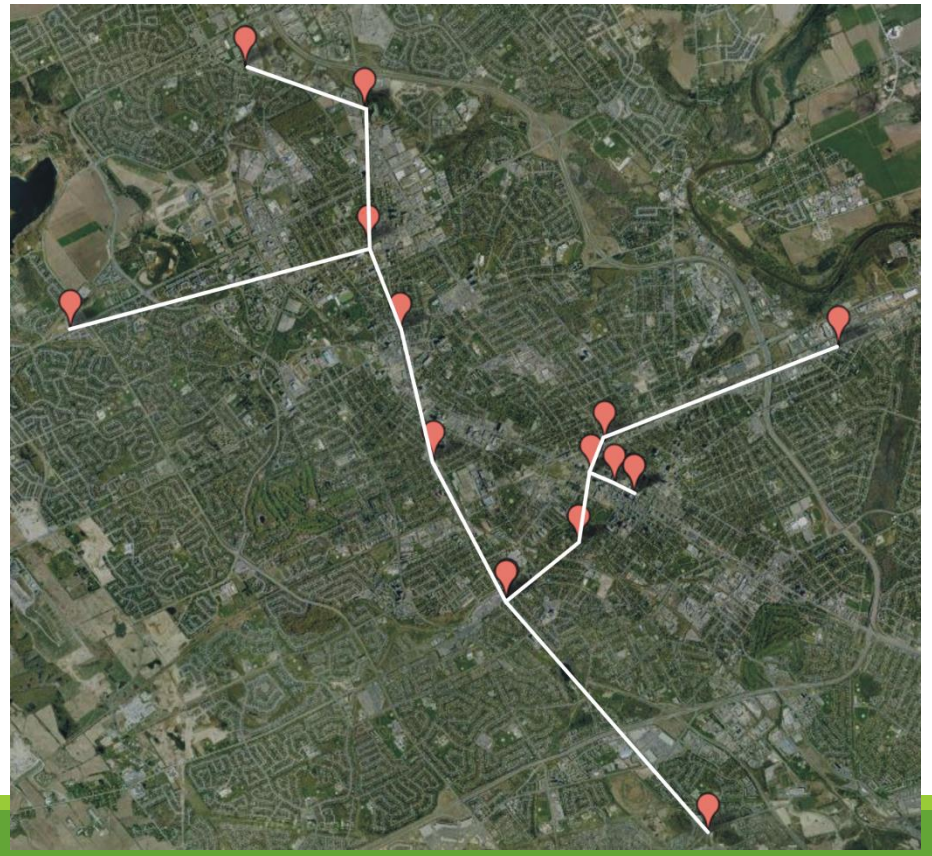
Application

Consider attempting to find the best means of connecting a number of LANs

- Minimize the number of bridges
- Costs not strictly dependent on distances

Application

A minimum spanning tree will provide the optimal solution



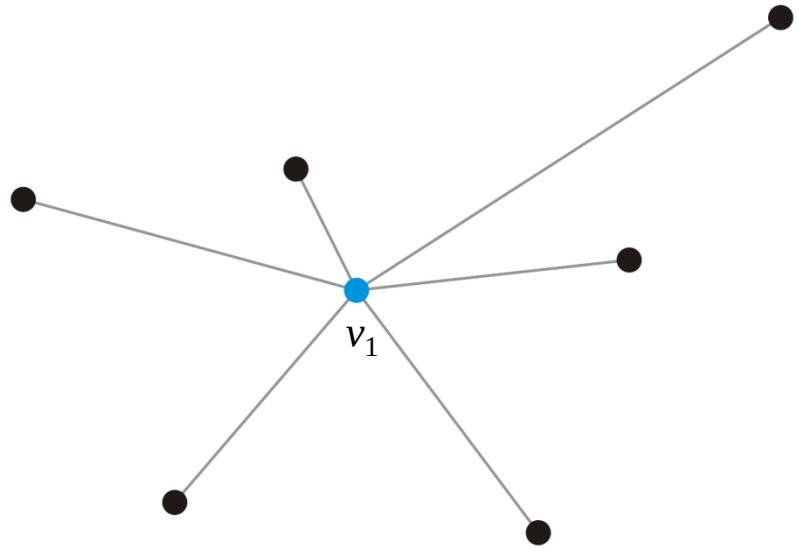
Algorithms

Two common algorithms for finding minimum spanning trees are:

- Prim's algorithm
- Kruskal's algorithm

Prim's algorithm

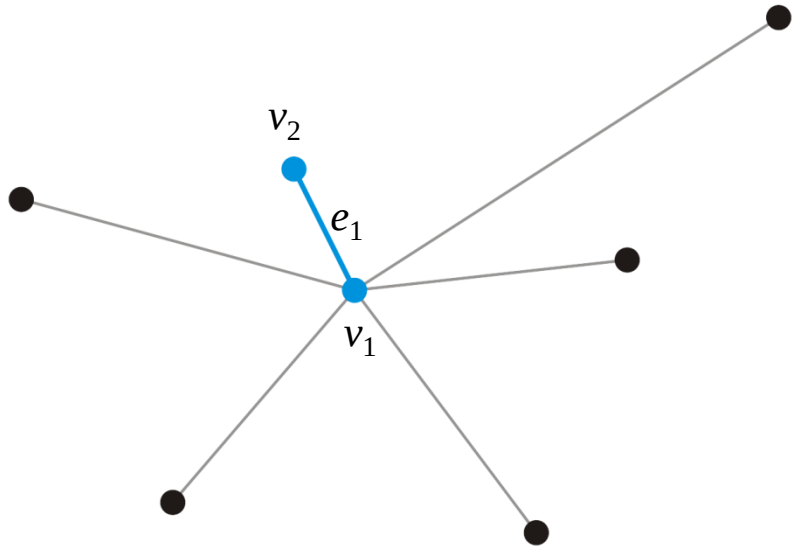
Given a single vertex v_1, it forms a minimum spanning tree on one vertex



Strategy

Add that adjacent vertex v_2 that has a connecting edge e_1 of minimum weight

- This forms a minimum spanning tree on our two vertices and e_1 must be in any minimum spanning tree containing the vertices v_1 and v_2



Minimum Spanning Trees

Prim's algorithm for finding the minimum spanning tree states:

- Start with an arbitrary vertex to form a minimum spanning tree on one vertex
- At each step, add that vertex v not yet in the minimum spanning tree that has an edge with least weight that connects v to the existing minimum spanning sub-tree
- Continue until we have $n - 1$ edges and n vertices

Initialization:

- Select a root node and set its distance as 0
- Set the distance to all other vertices as ∞
- Set all vertices to being unvisited
- Set the parent pointer of all vertices to 0

Prim's Algorithm

Iterate while there exists an unvisited vertex with distance < ∞

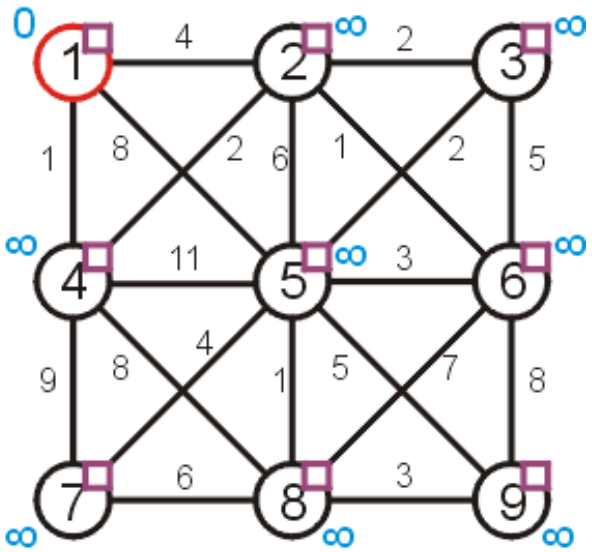
- Select that unvisited vertex with minimum distance
- Mark that vertex as having been visited
- For each adjacent vertex, if the weight of the connecting edge is less than the current distance to that vertex:
  - Update the distance to equal the weight of the edge
  - Set the current vertex as the parent of the adjacent vertex

Halting Conditions:

- There are no unvisited vertices which have a distance < ∞

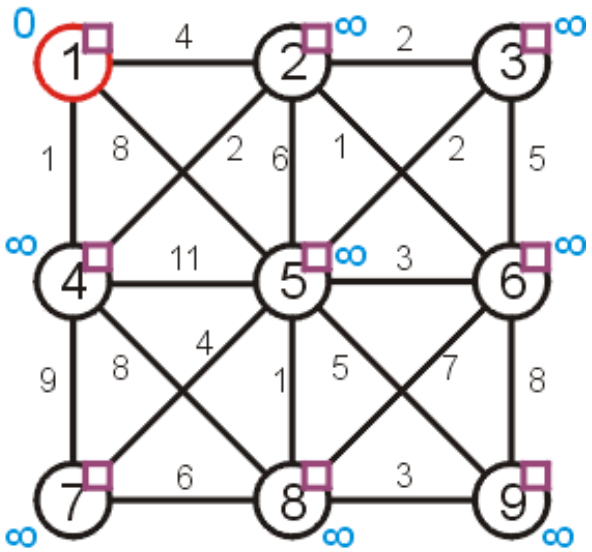
Prim's Algorithm

Let us find the minimum spanning tree for the following undirected weighted graph



Prim's Algorithm

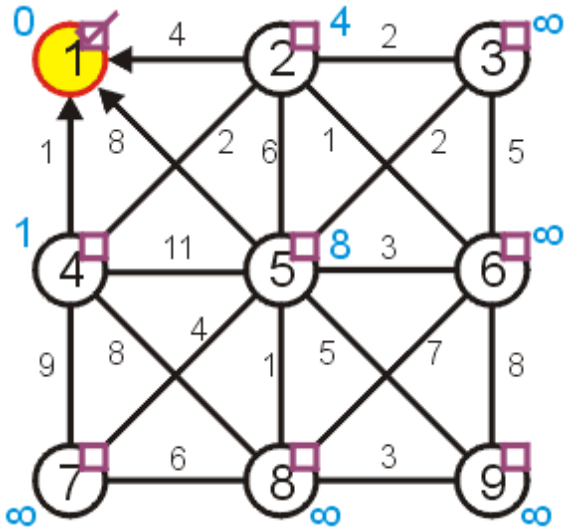
First we set up the appropriate table and initialize it



		Distance	Parent
1	F	0	0
2	F	∞	0
3	F	∞	0
4	F	∞	0
5	F	∞	0
6	F	∞	0
7	F	∞	0
8	F	∞	0
9	F	∞	0

Prim's Algorithm

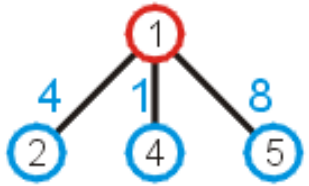
Visiting vertex 1, we update vertices 2, 4, and 5



		Distance	Parent
1	T	0	0
2	F	4	1
3	F	∞	0
4	F	1	1
5	F	8	1
6	F	∞	0
7	F	∞	0
8	F	∞	0
9	F	∞	0

Prim's Algorithm

What these numbers really mean is that at this point, we could extend the trivial tree containing just the root node by one of the three possible children:

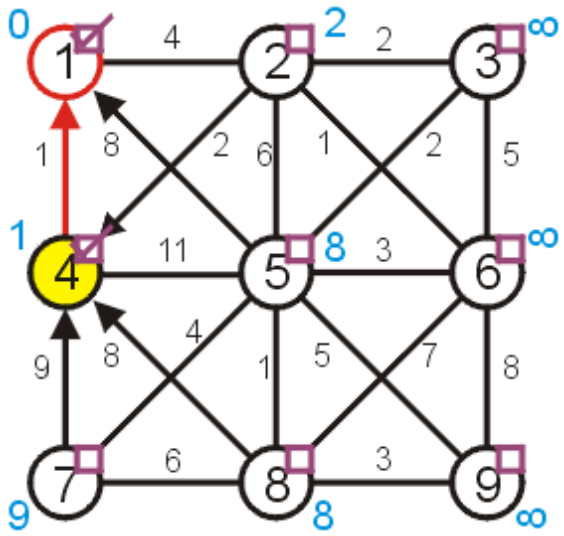


As we wish to find a *minimum* spanning tree, it makes sense we add that vertex with a connecting edge with least weight

Prim's Algorithm

The next unvisited vertex with minimum distance is vertex 4

- Update vertices 2, 7, 8
- Don't update vertex 5



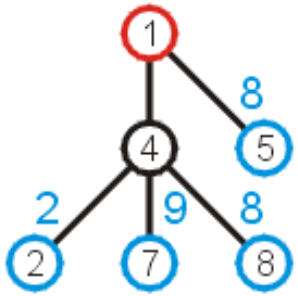
		Distance	Parent
1	T	0	0
2	F	2	4
3	F	∞	0
4	T	1	1
5	F	8	1
6	F	∞	0
7	F	9	4
8	F	8	4
9	F	∞	0

Prim's Algorithm

Now that we have updated all vertices adjacent to vertex 4, we can extend the tree by adding one of the edges

(1, 5), (4, 2), (4, 7), or (4, 8)

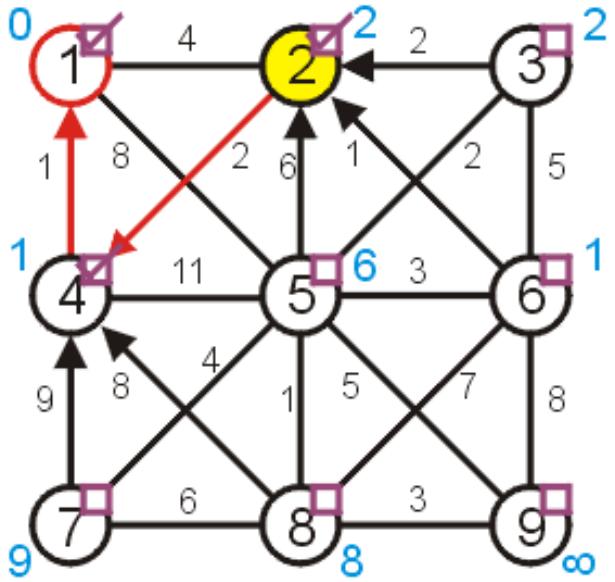
We add that edge with the least weight: (4, 2)



Prim's Algorithm

Next visit vertex 2

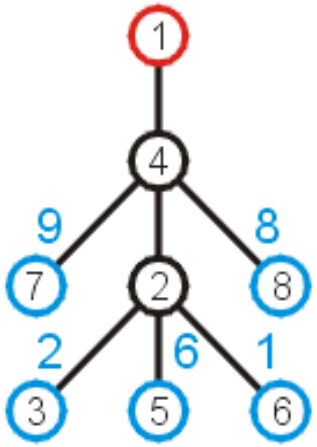
- Update 3, 5, and 6



		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	6	2
6	F	1	2
7	F	9	4
8	F	8	4
9	F	∞	0

Prim's Algorithm

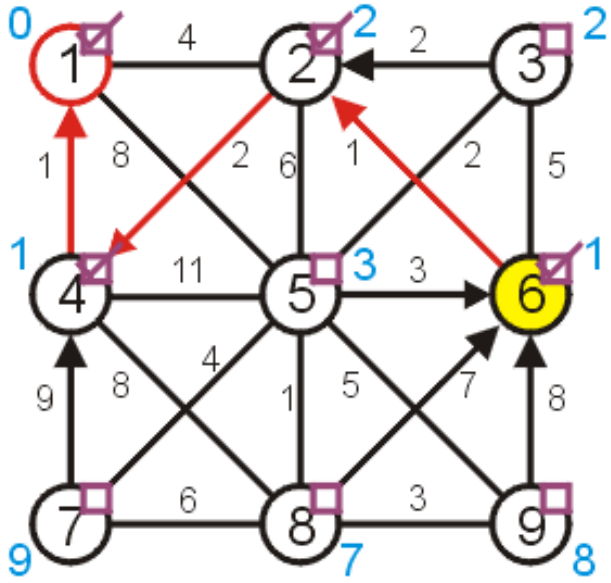
Again looking at the shortest edges to each of the vertices adjacent to the current tree, we note that we can add (2, 6) with the least increase in weight



Prim's Algorithm

Next, we visit vertex 6:

- update vertices 5, 8, and 9

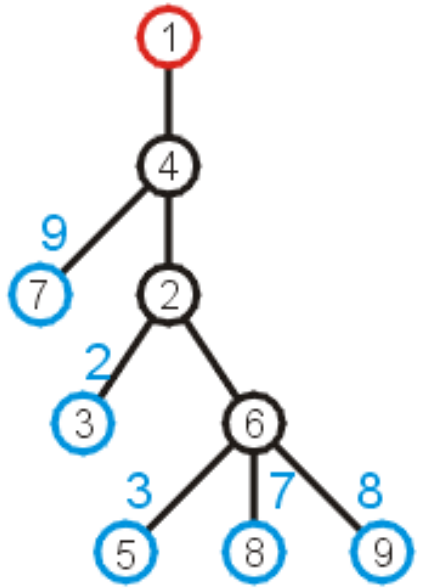


		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	3	6
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	6

Prim's Algorithm

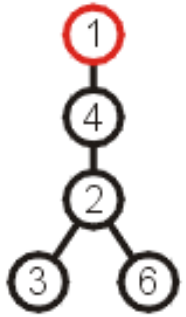
The edge with least weight is (2, 3)

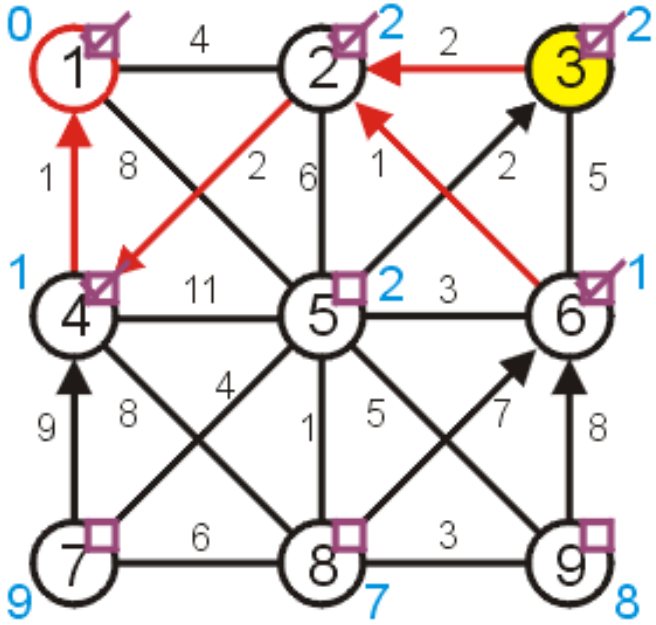
- This adds the weight of 2 to the weight minimum spanning tree



Prim's Algorithm

Next, we visit vertex 3 and update 5

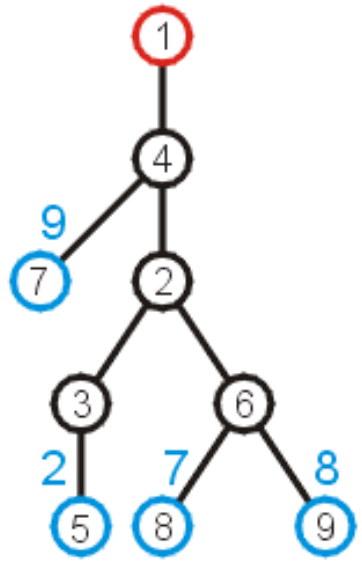




		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	F	2	3
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	6

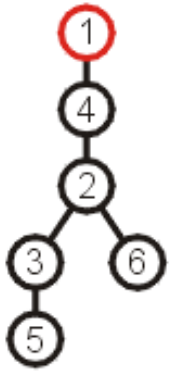
Prim's Algorithm

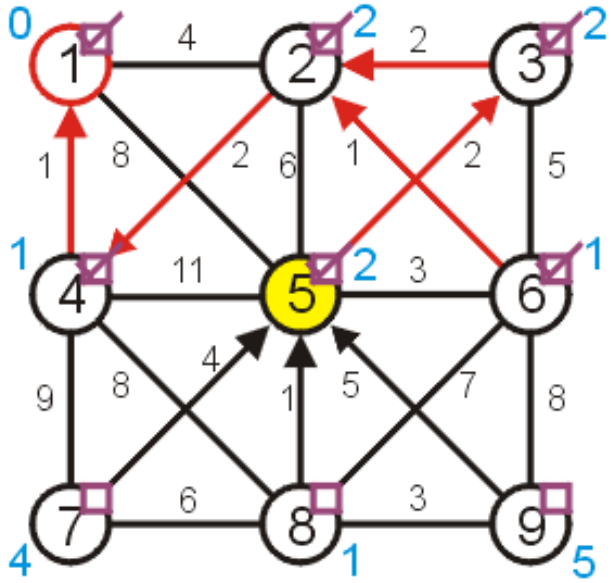
At this point, we can extend the tree by adding the edge (3, 5)



Prim's Algorithm

Visiting vertex 5, we update 7, 8, 9



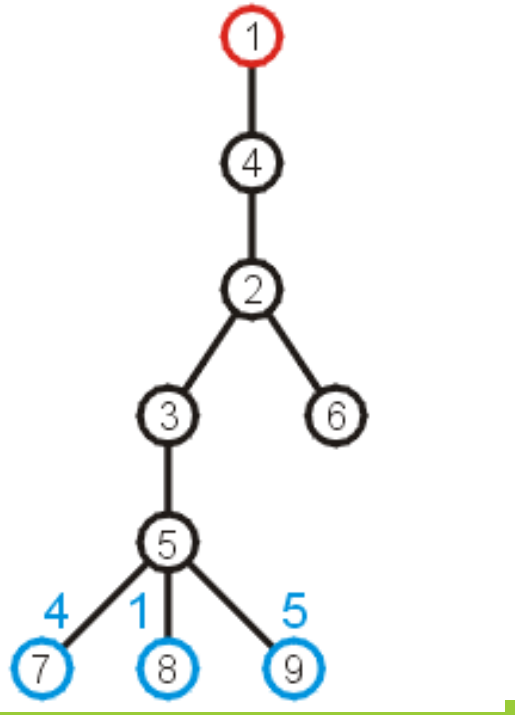


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	F	1	5
9	F	5	5

Prim's Algorithm

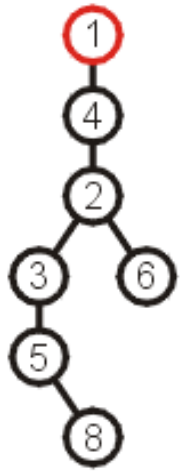
At this point, there are three possible edges which we could include which will extend the tree

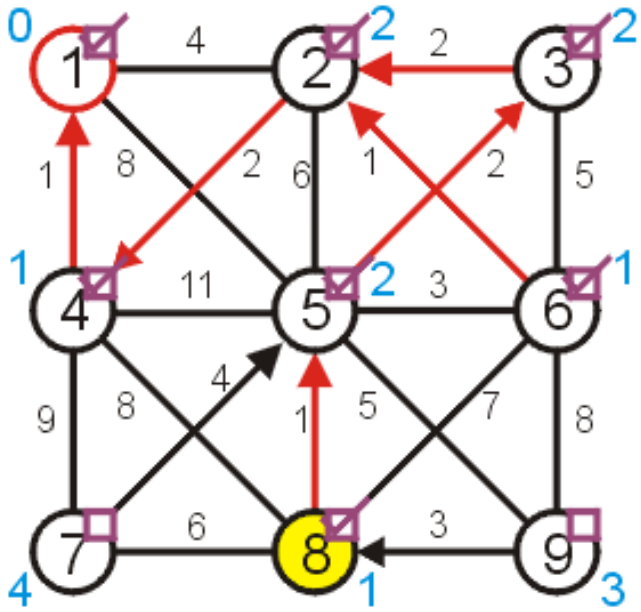
The edge to 8 has the least weight



Prim's Algorithm

Visiting vertex 8, we only update vertex 9

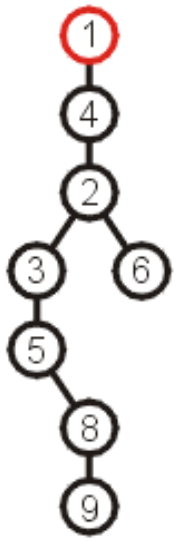


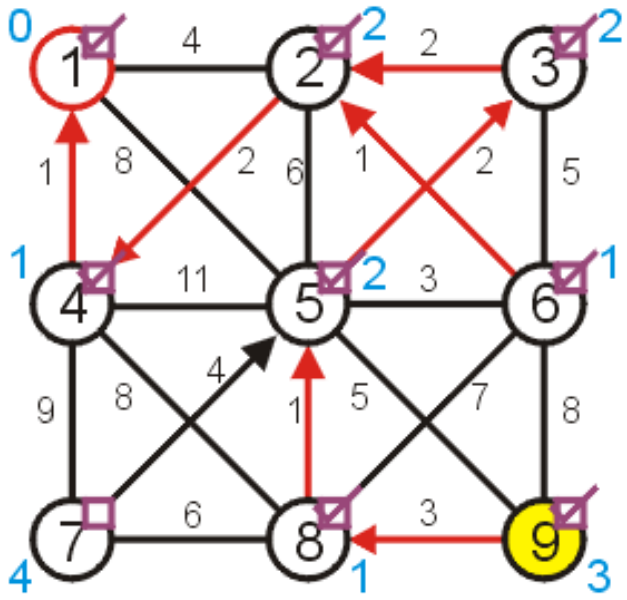


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	F	3	8

Prim's Algorithm

There are no other vertices to update while visiting vertex 9

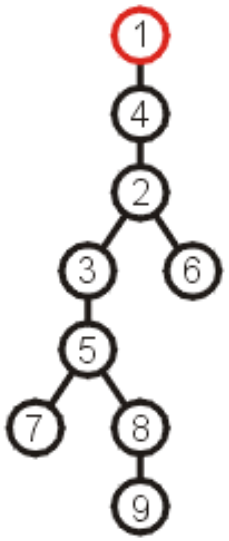


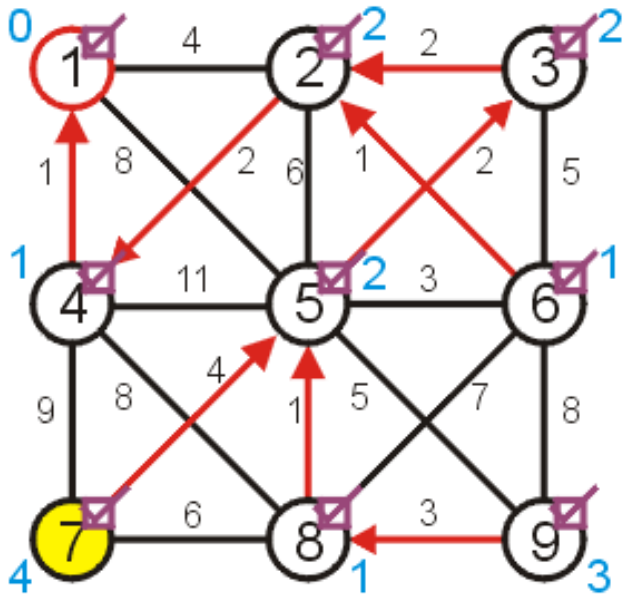


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	T	3	8

Prim's Algorithm

And neither are there any vertices to update when visiting vertex 7





		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	T	4	5
8	T	1	5
9	T	3	8

Prim's Algorithm

At this point, there are no more unvisited vertices, and therefore we are done

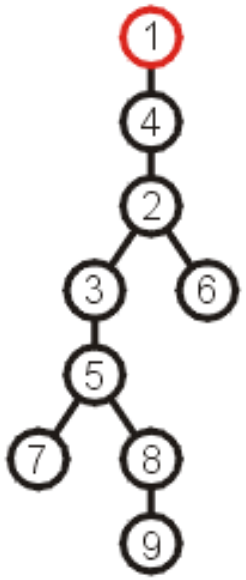
If at any point, all remaining vertices had a distance of ∞,

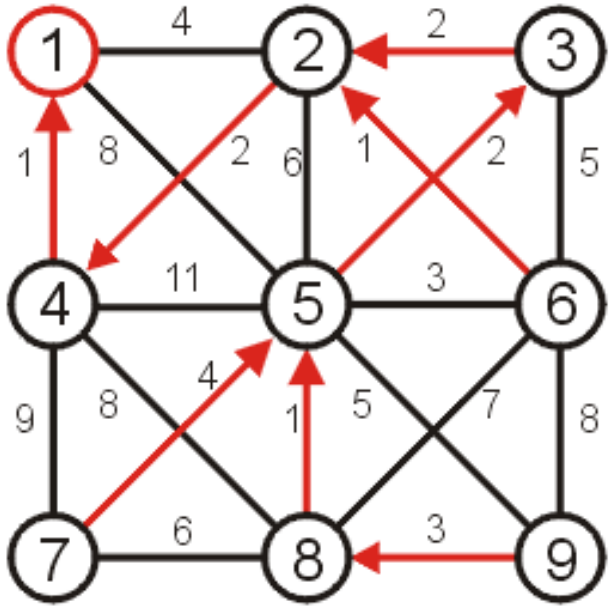
this would indicate that the graph is not connected

- in this case, the minimum spanning tree would only span one connected sub-graph

Prim's Algorithm

Using the parent pointers, we can now construct the minimum spanning tree





		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	T	4	5
8	T	1	5
9	T	3	8

Implementation and analysis

The initialization requires $\Theta(|V|)$ memory and run time

We iterate $|V| - 1$ times, each time finding the *closest* vertex

- Iterating through the table requires is $\Theta(|V|)$ time
- Each time we find a vertex, we must check all of its neighbors
- With an adjacency matrix, the run time is $\Theta(|V|(|V| + |V|)) = \Theta(|V|^2)$
- With an adjacency list, the run time is $\Theta(|V|^2 + |E|) = \Theta(|V|^2)$

Can we do better?

Kruskal's Algorithm

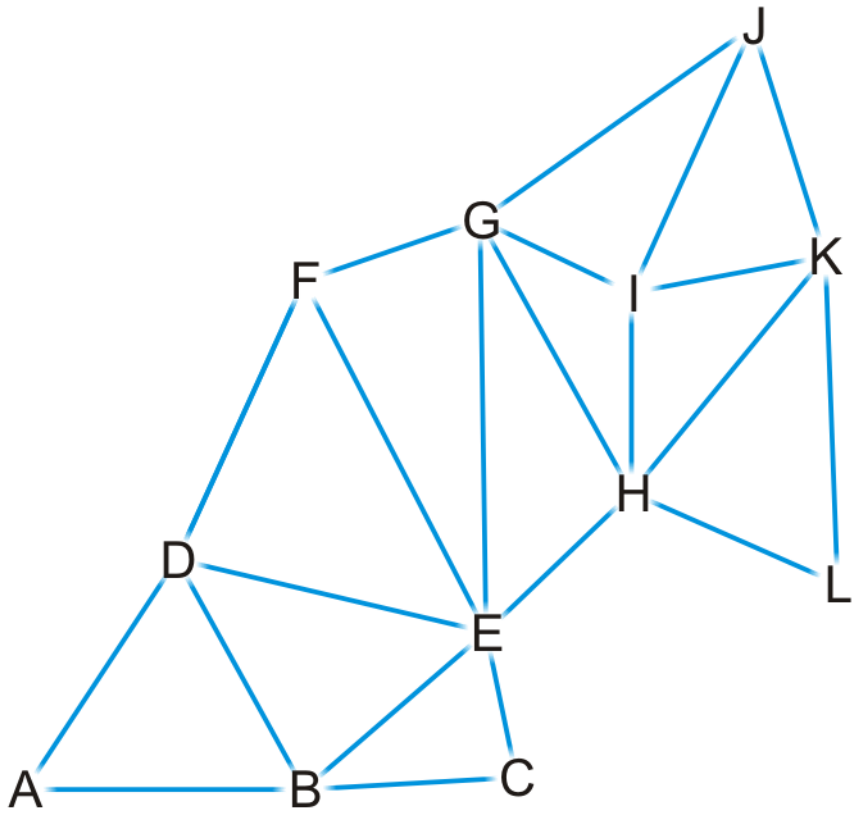
Kruskal's algorithm sorts the edges by weight and goes through the edges from least weight to greatest weight adding the edges to an empty graph so long as the addition does not create a cycle

The halting point is:

- When $|V| - 1$ edges have been added
  - In this case we have a minimum spanning tree

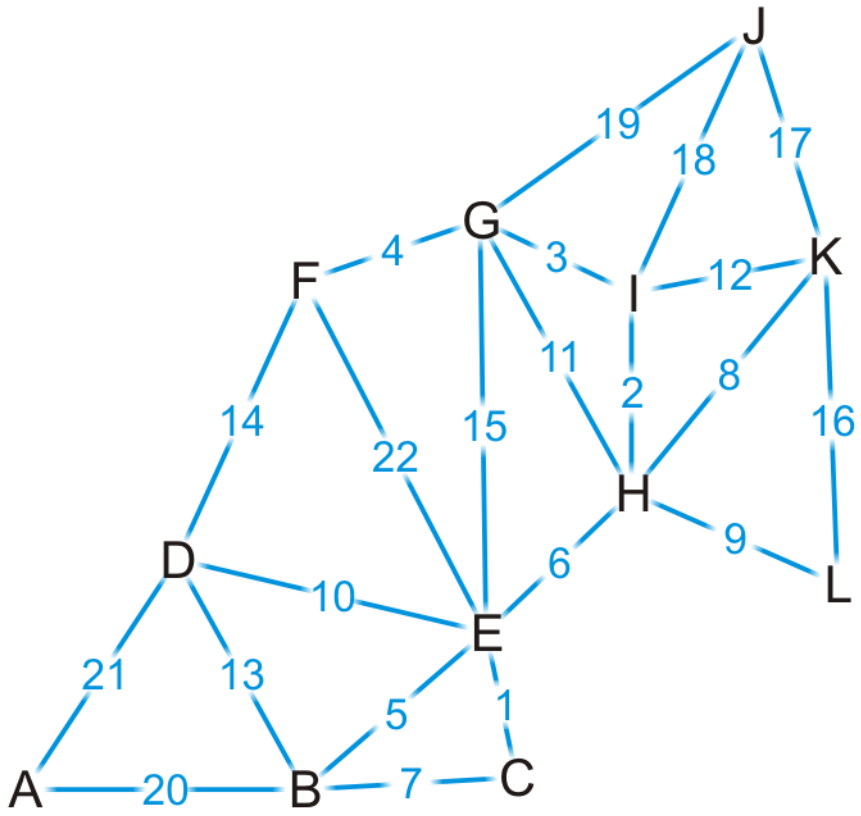
Example

Here is our abstract representation



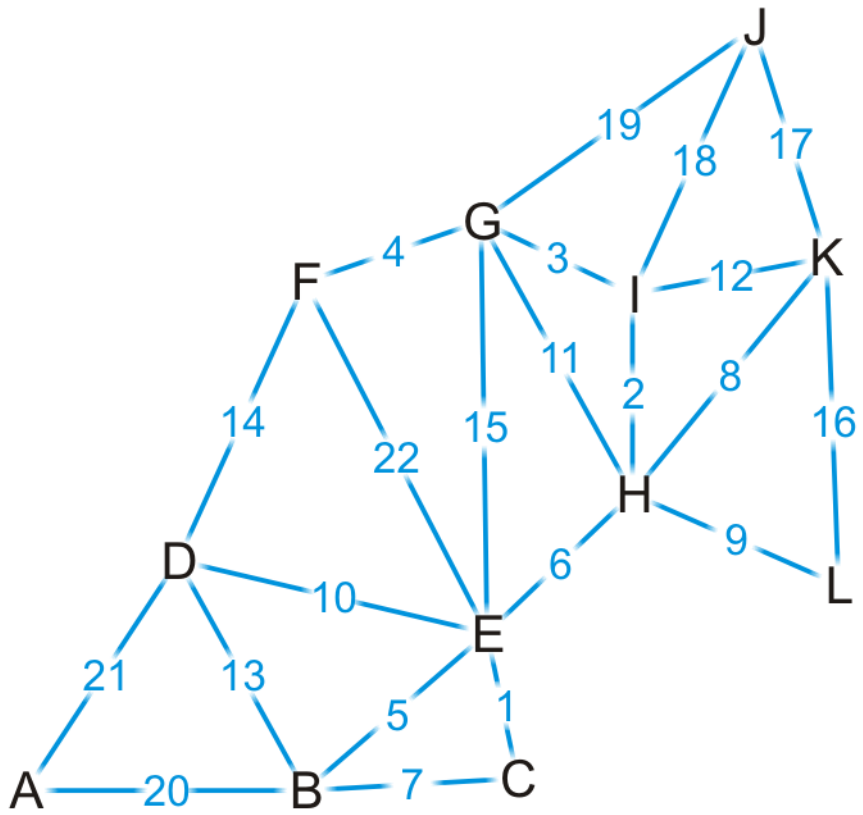
Example

Let us give a weight to each of the edges



Example

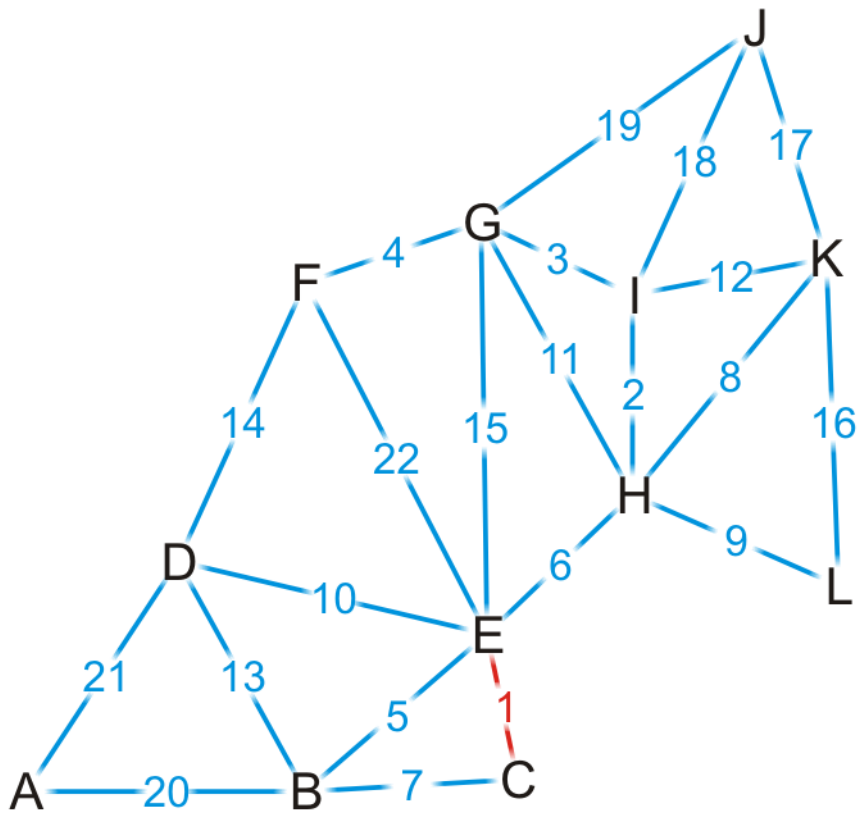
First, we sort the edges based on weight



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

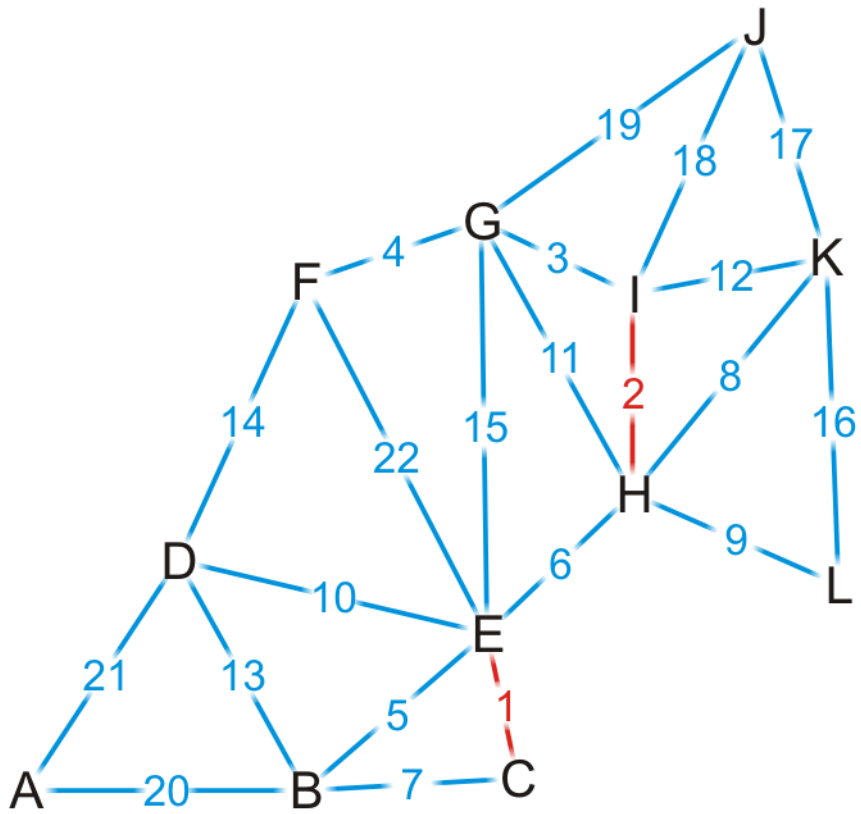
We start by adding edge {C, E}



→ {C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

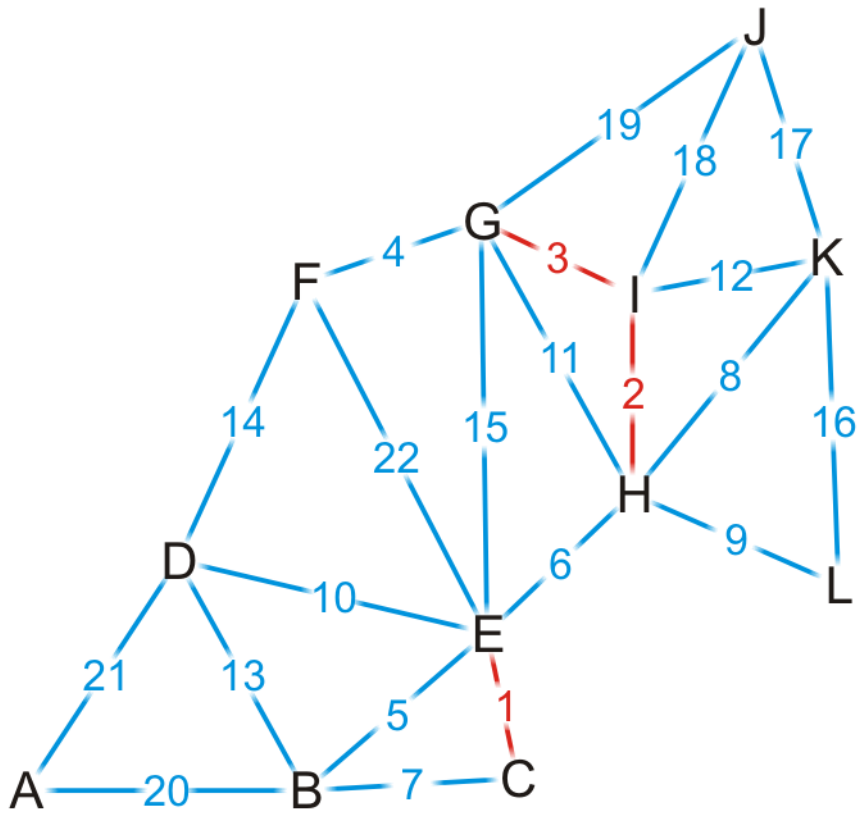
We add edge {H, I}



{C, E}
→ {H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

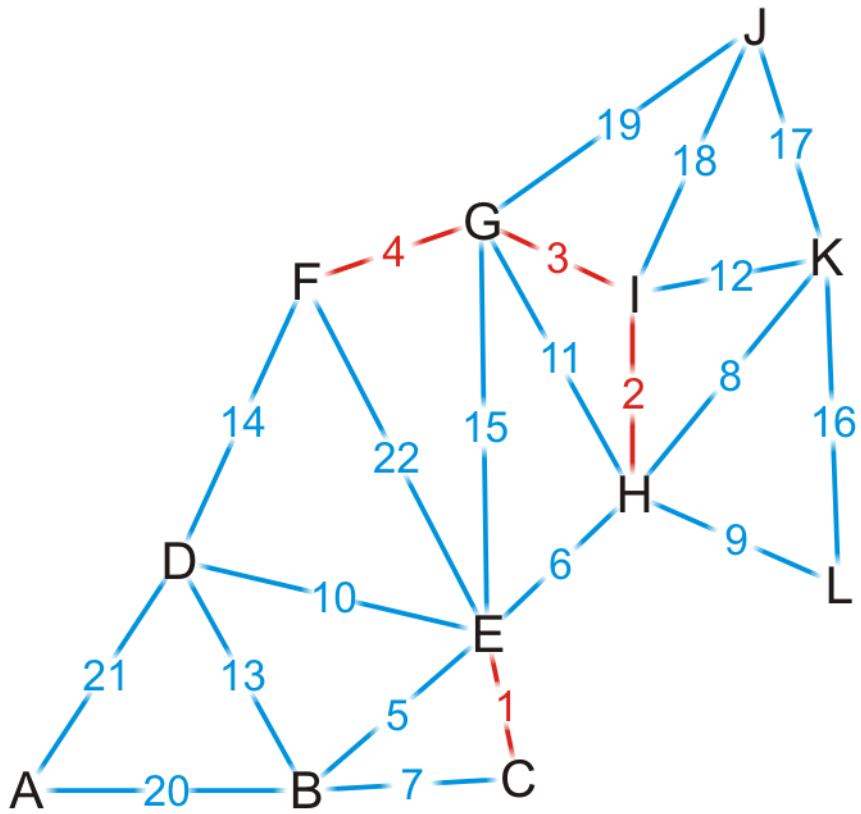
We add edge {G, I}



{C, E}
{H, I}
→ {G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

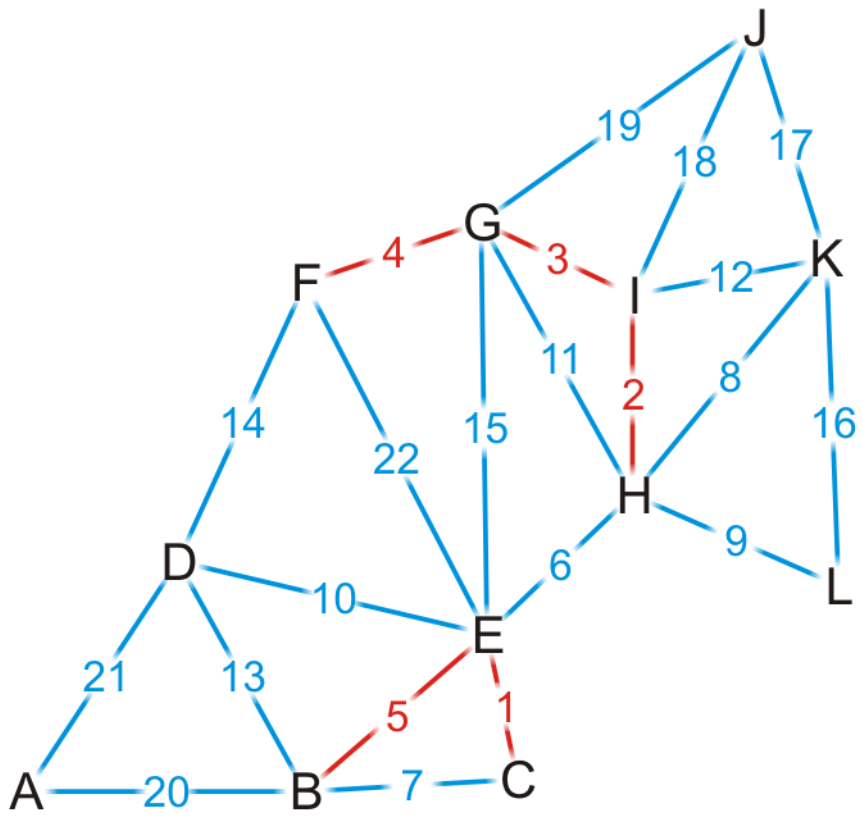
We add edge {F, G}



{C, E}
{H, I}
{G, I}
→ {F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We add edge {B, E}

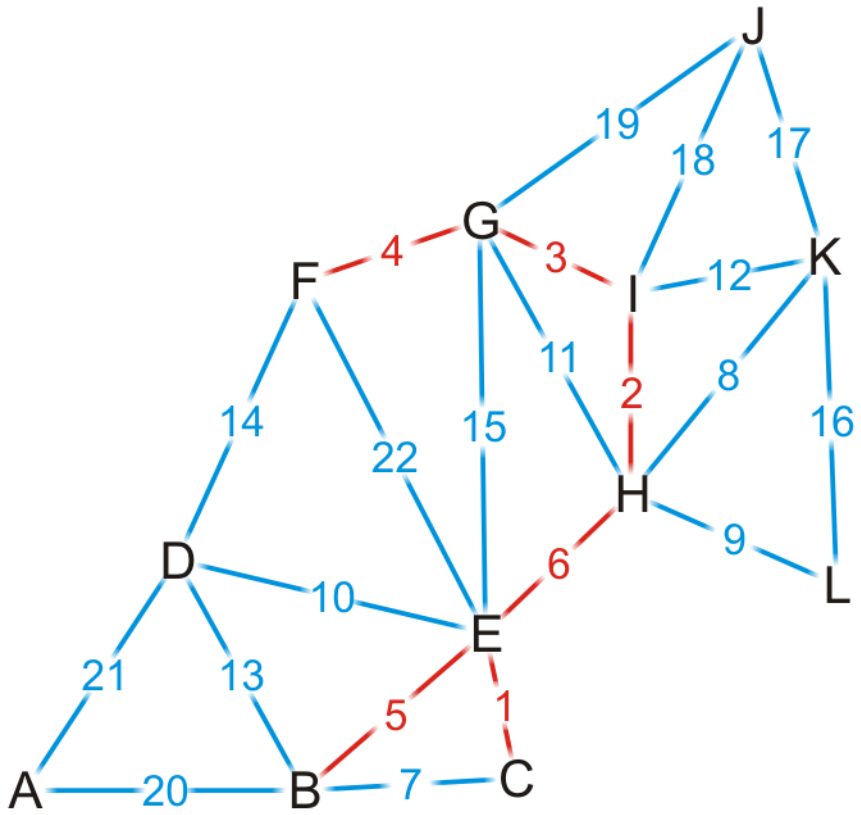


{C, E}
{H, I}
{G, I}
{F, G}
→ {B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We add edge {E, H}

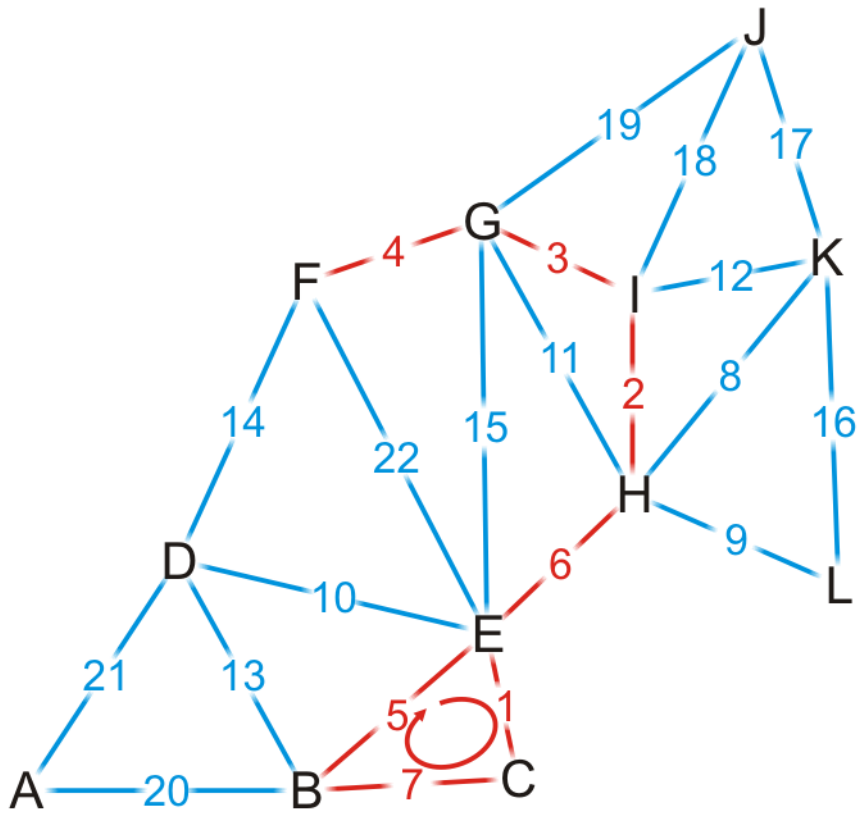
- This coalesces the two spanning sub-trees into one



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
→ {E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

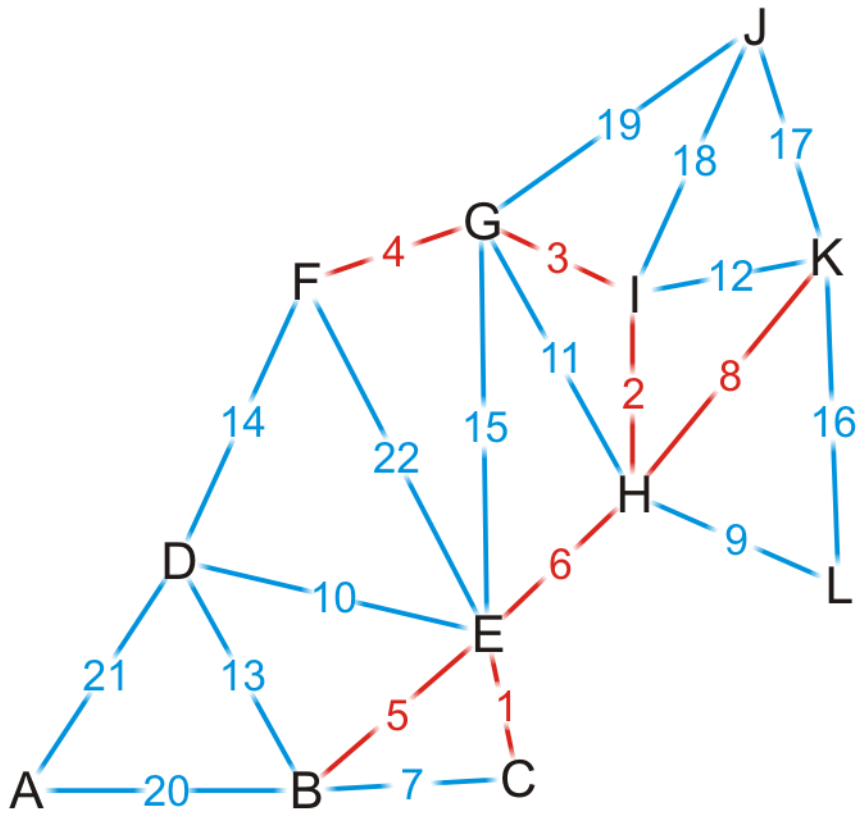
We try adding {B, C}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
→ {B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

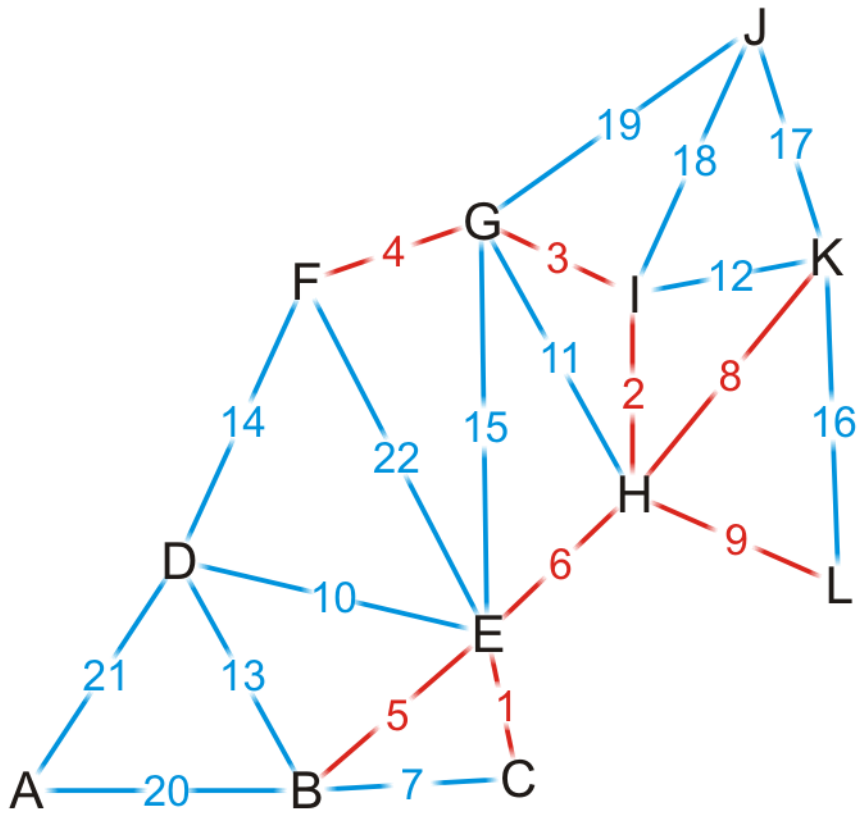
We add edge {H, K}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
→ {H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

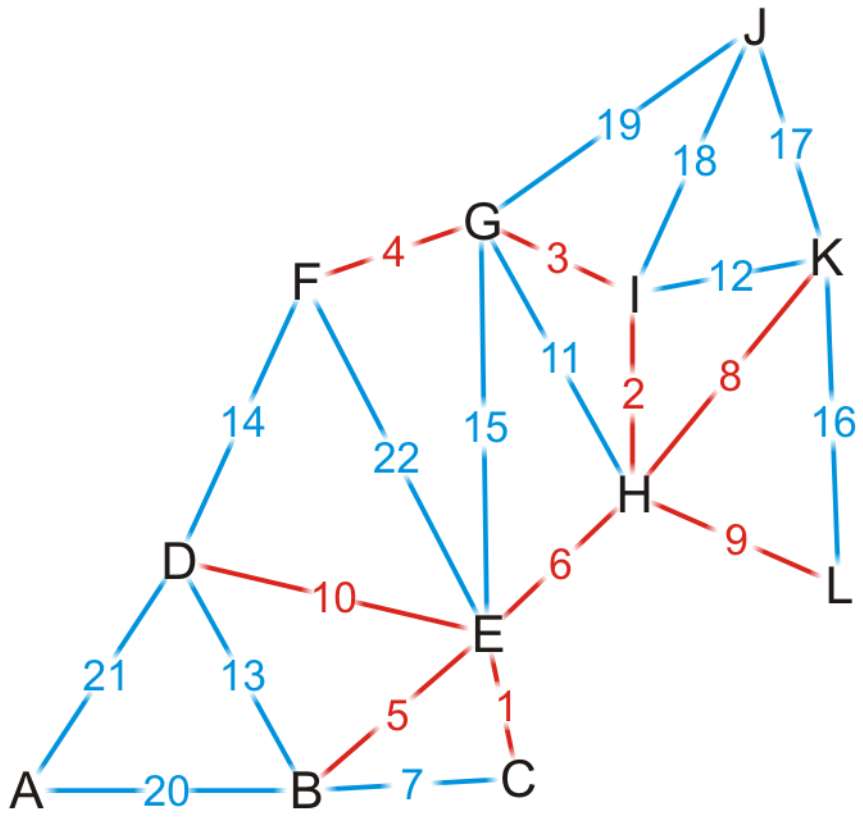
We add edge {H, L}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
→ {H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

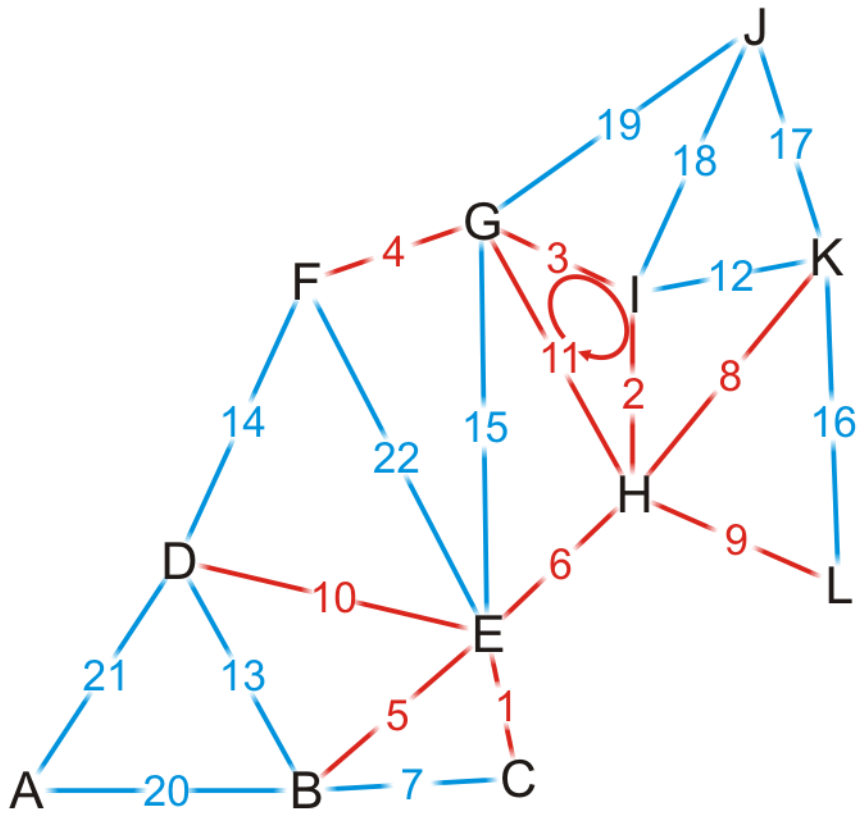
We add edge {D, E}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
→ {D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

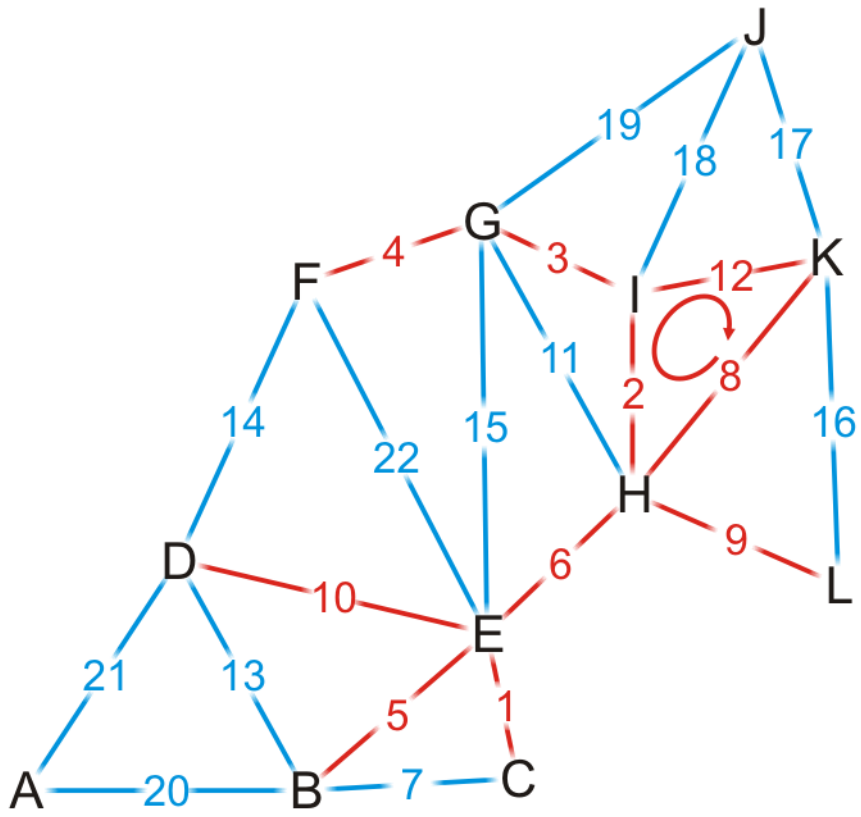
We try adding {G, H}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
→ {G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

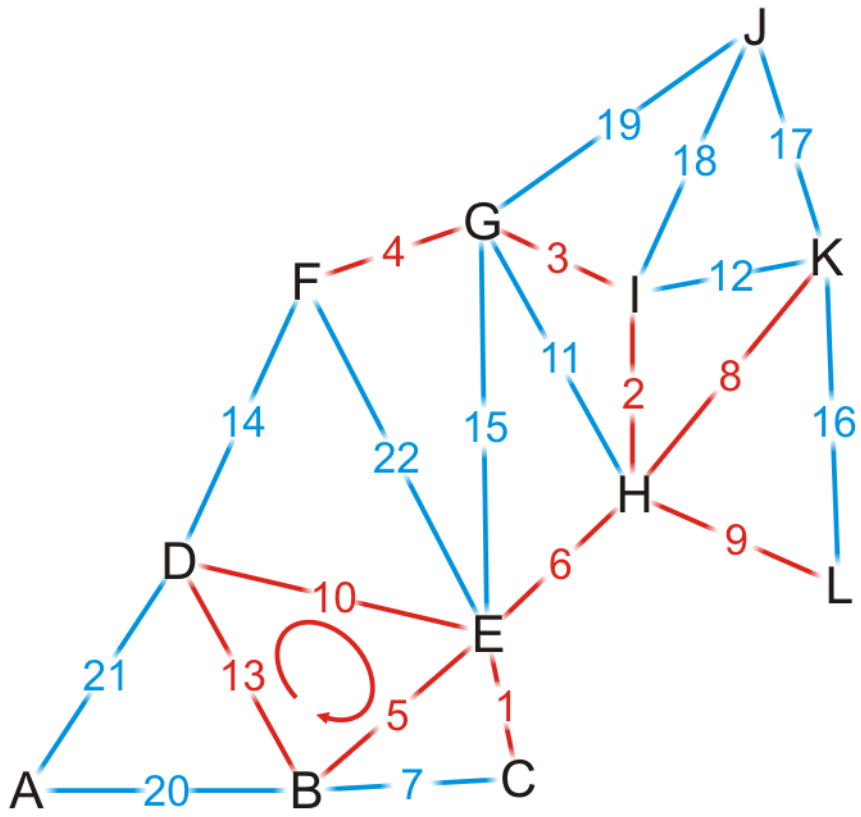
We try adding {I, K}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
→ {I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

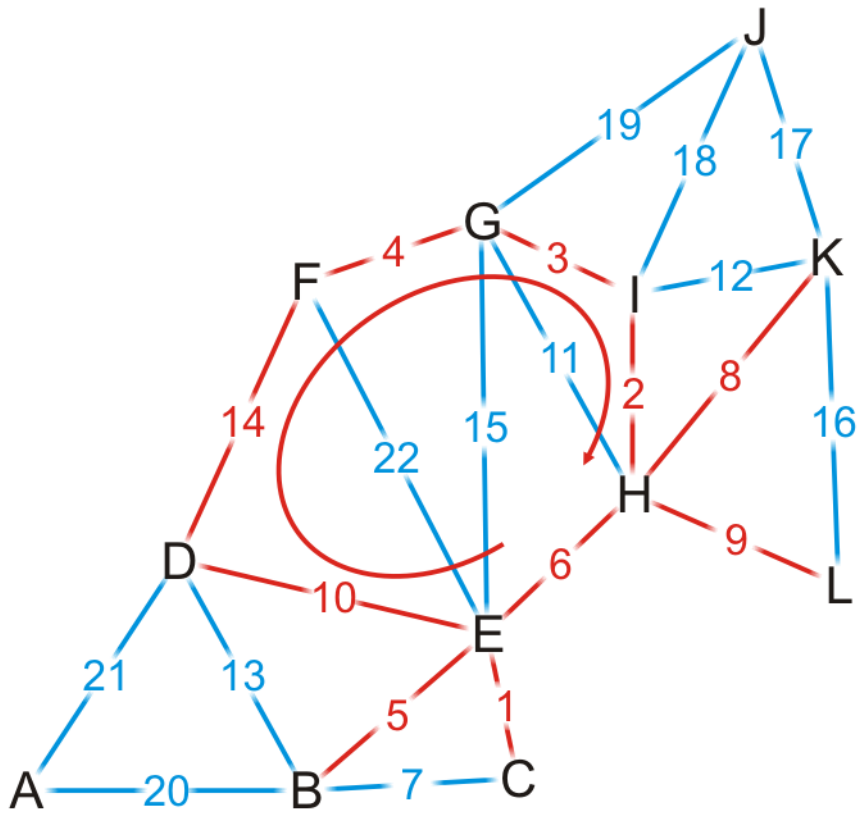
We try adding {B, D}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
→ {B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

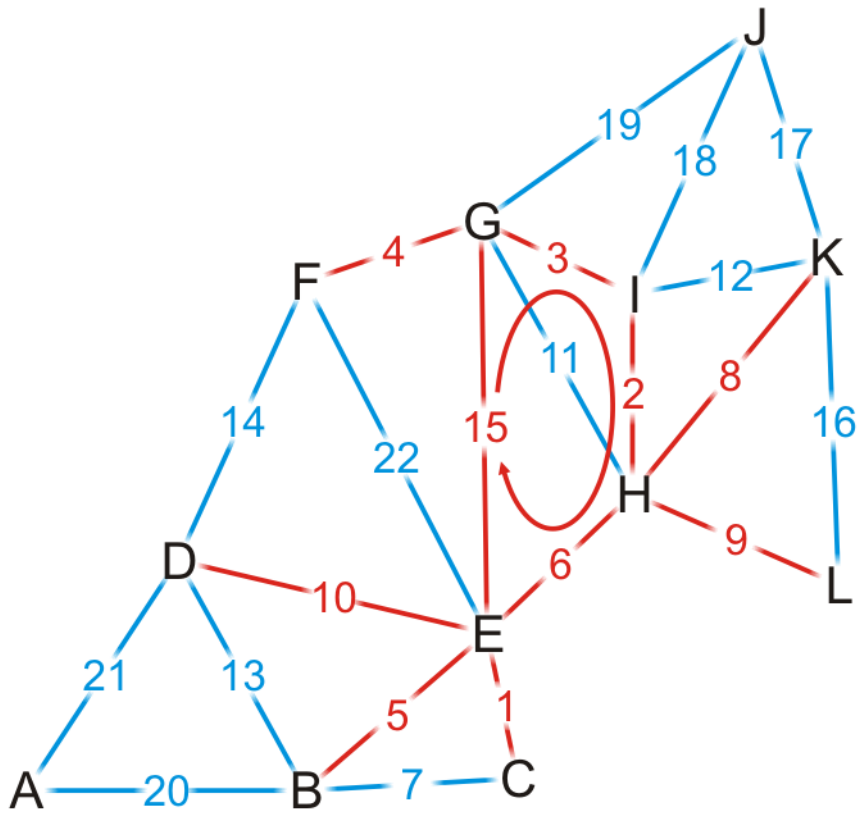
We try adding {D, F}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
→ {D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

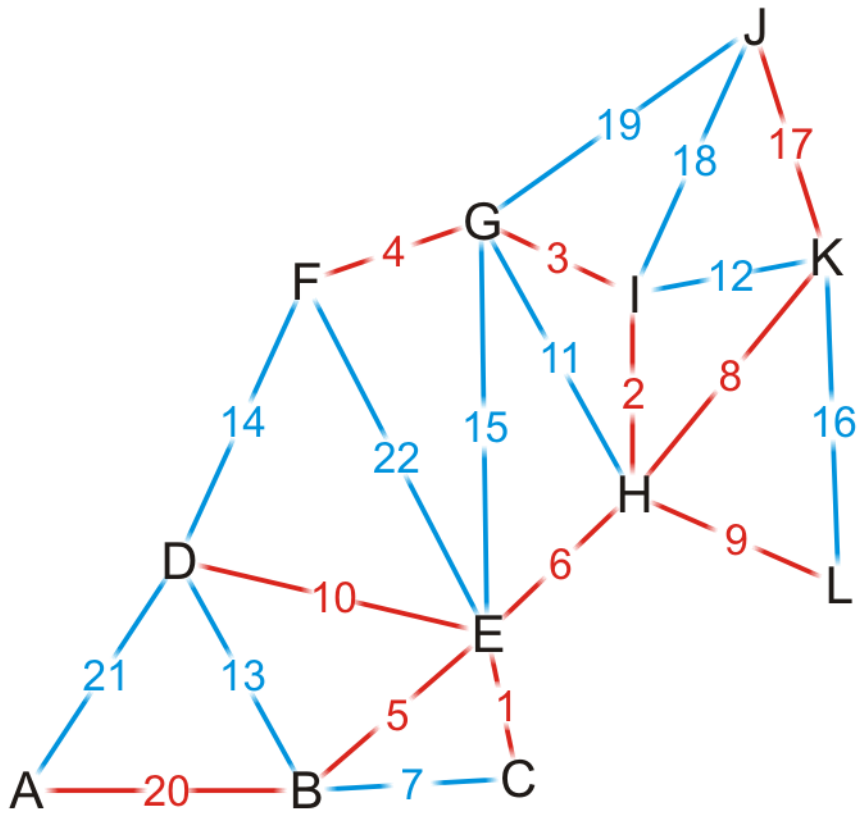
We try adding {E, G}, but it creates a cycle



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
→ {E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

By observation, we can still add edges {J, K} and {A, B}

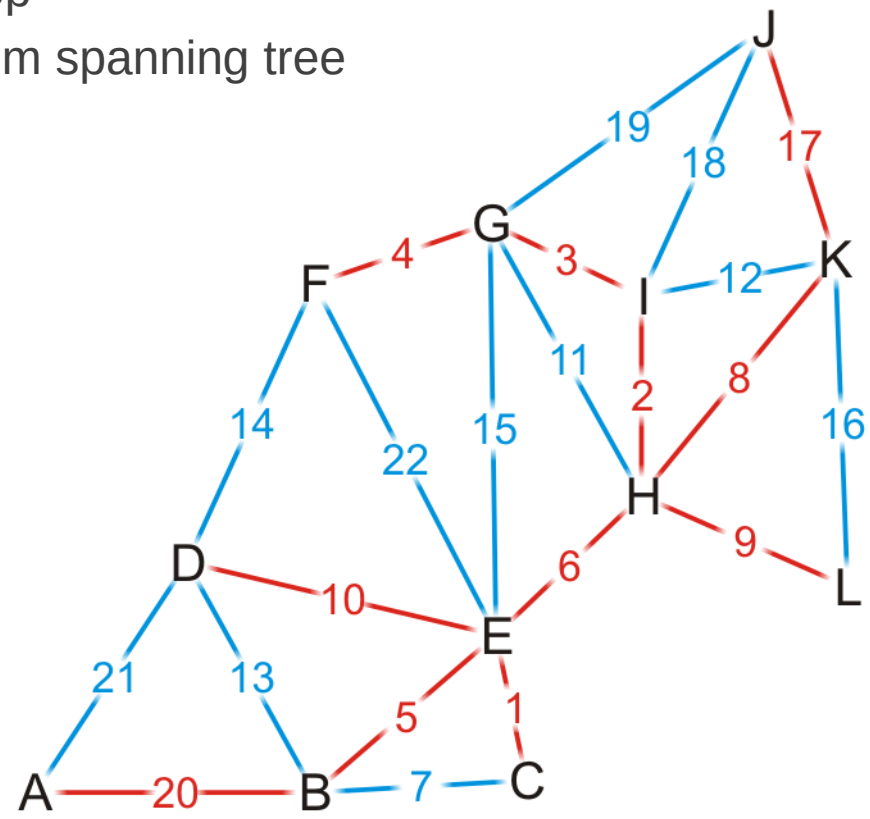


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
→ {K, L}
→ {J, K}
→ {J, I}
→ {J, G}
→ {A, B}
{A, D}
{E, F}

Example

Having added {A, B}, we now have 11 edges

- We terminate the loop
- We have our minimum spanning tree



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Analysis

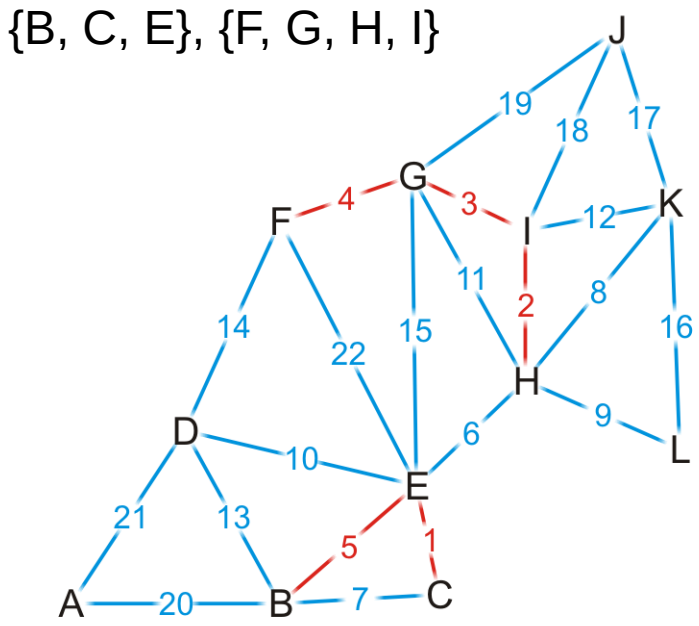
Implementation

- We would store the edges and their weights in an array
- We would sort the edges using either quicksort or some distribution sort
- To determine if a cycle is created, we could perform a traversal
  - A run-time of $O(|V|)$
- the run-time would be $O(|E| \ln(|E|) + |E| \cdot |V|)$
- Consequently, the run-time would be $O(|E| \ln(|V|) + |E||V|) = O(|E| \cdot |V|)$

Analysis

Instead, we could use disjoint sets

- Consider edges in the same connected sub-graph as forming a set



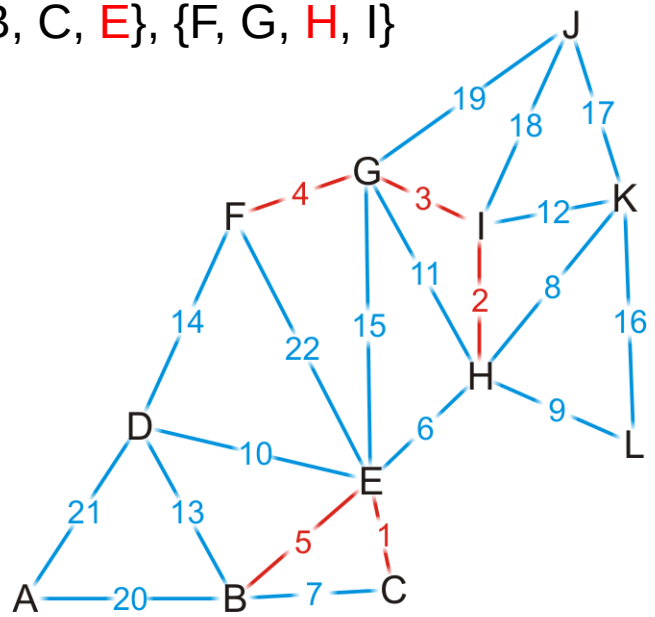
Analysis

Instead, we could use disjoint sets

- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets, take the union of the two sets

Add edge (E, H)?

{B, C, E}, {F, G, H, I}



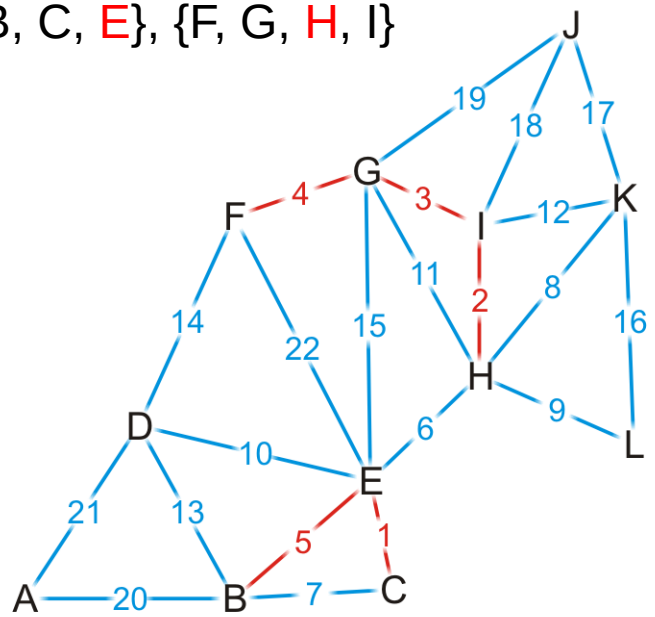
Analysis

Instead, we could use disjoint sets

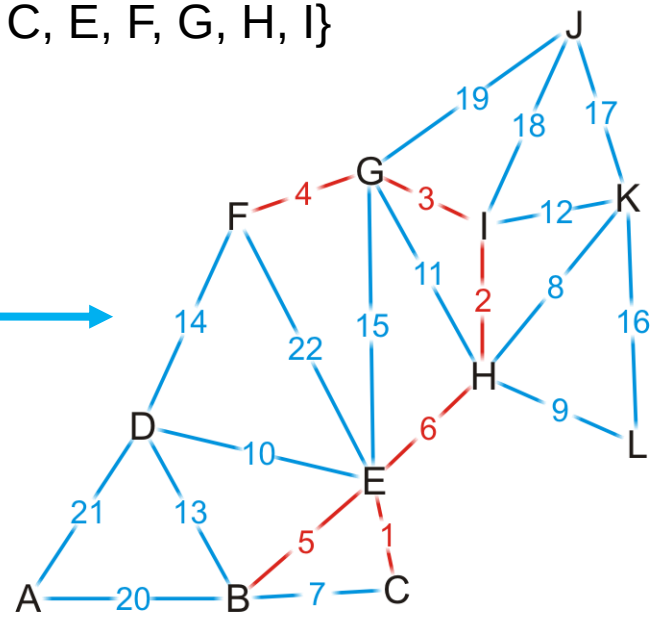
- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets, take the union of the two sets

Add edge (E, H)?

{B, C, E}, {F, G, H, I}



{B, C, E, F, G, H, I}



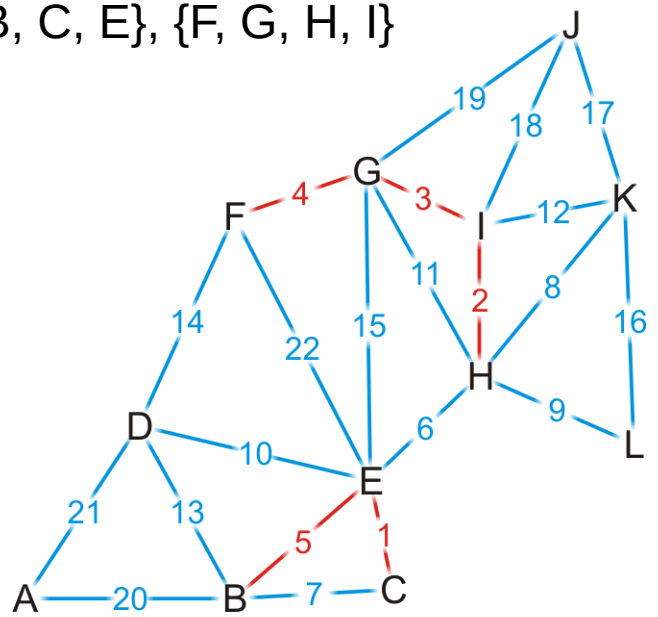
Analysis

Instead, we could use disjoint sets

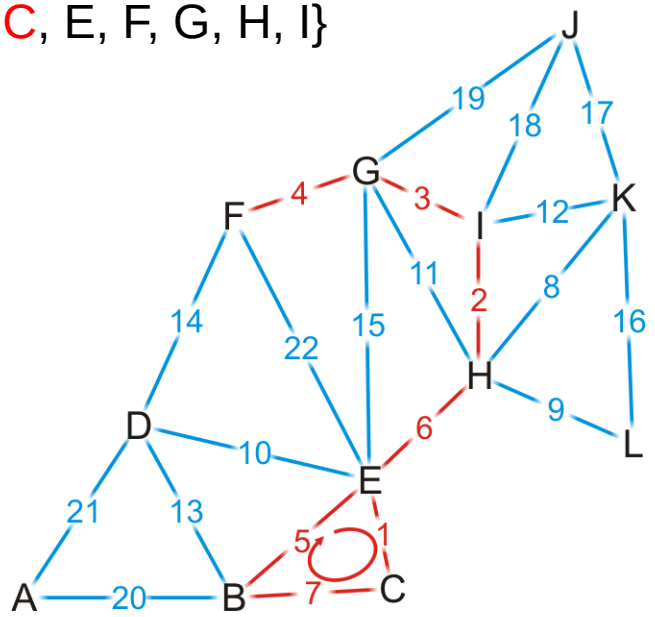
- Consider edges in the same connected sub-graph as forming a set
- If the vertices of the next edge are in different sets, take the union of the two sets
- Do not add an edge if both vertices are in the same set

Add edge (E, H)?

{B, C, E}, {F, G, H, I}



{B, C, E, F, G, H, I}



Analysis

checking and building the minimum spanning tree is now $O(|E|)$

The dominant time is now the time required to sort the edges:

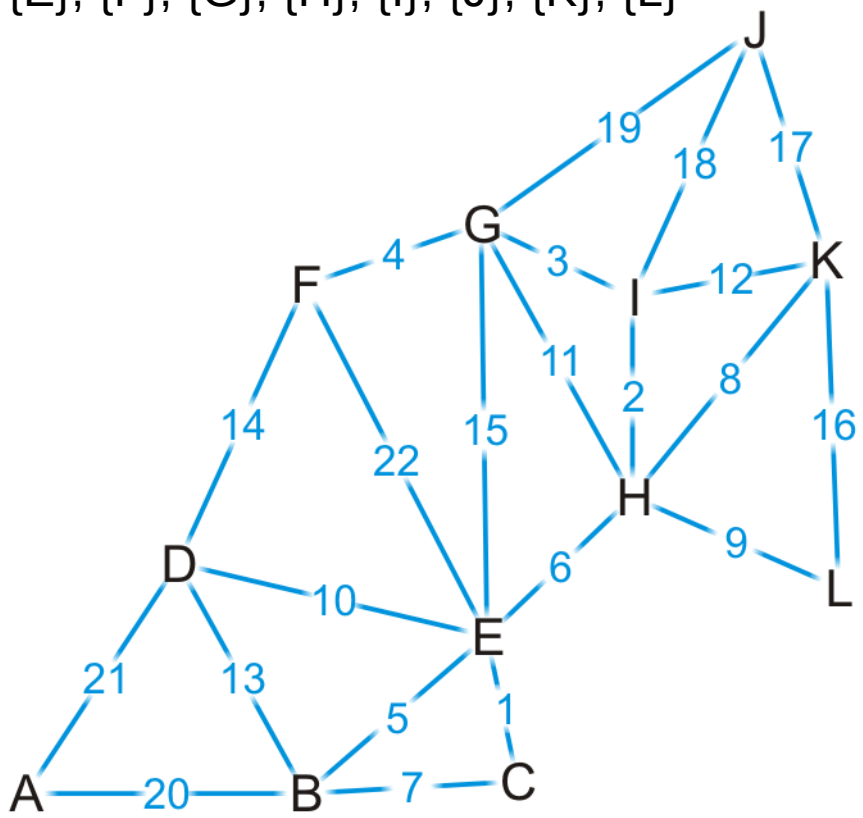
- Using quicksort, the run-time is $O(|E| \ln(|E|)) = O(|E| \ln(|V|))$
- If there is an efficient $\Theta(|E|)$ sorting algorithm, the run-time is then $\Theta(|E|)$

Going through the example again with disjoint sets

Example

We start with twelve singletons

{A}, {B}, {C}, {D}, {E}, {F}, {G}, {H}, {I}, {J}, {K}, {L}

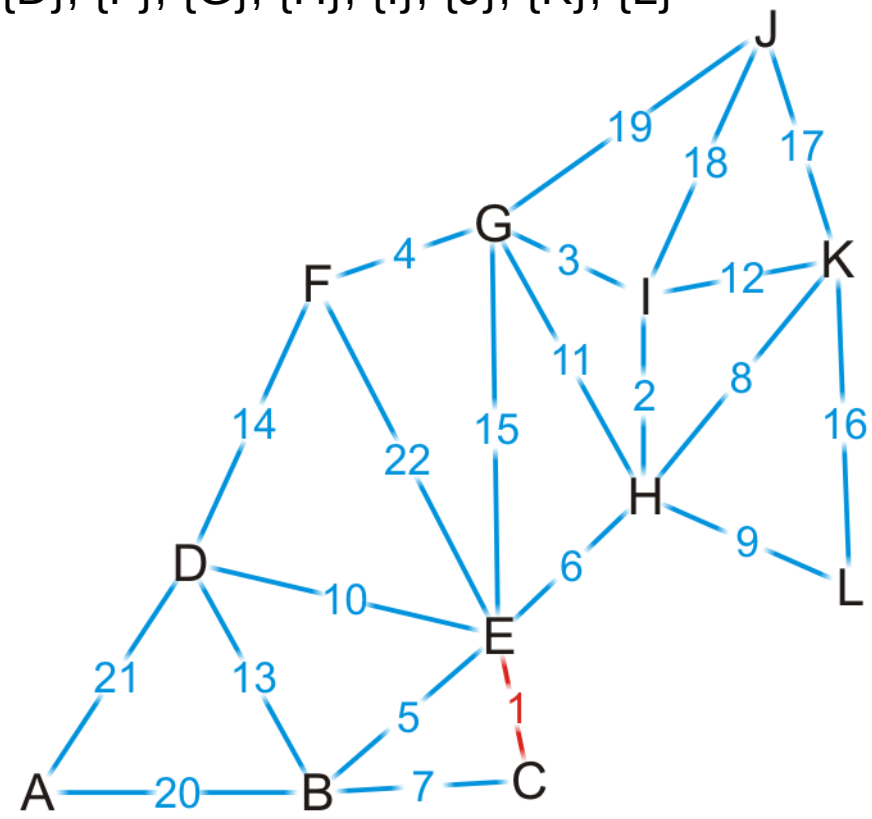


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We start by adding edge {C, E}

{A}, {B}, {C, E}, {D}, {F}, {G}, {H}, {I}, {J}, {K}, {L}

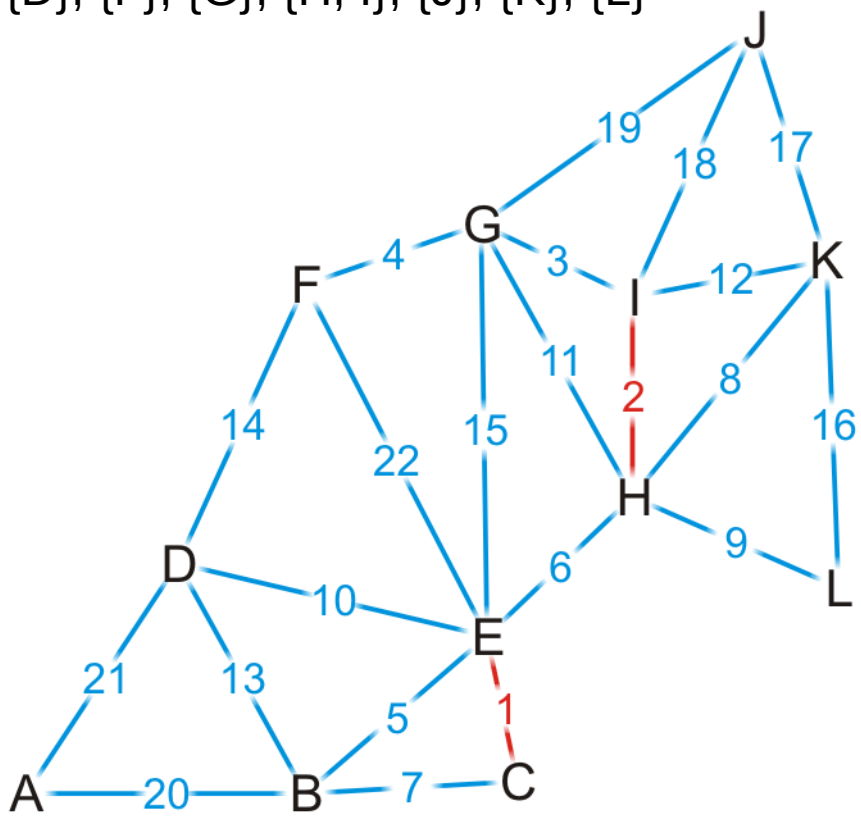


→ {C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We add edge {H, I}

{A}, {B}, {C, E}, {D}, {F}, {G}, {H, I}, {J}, {K}, {L}

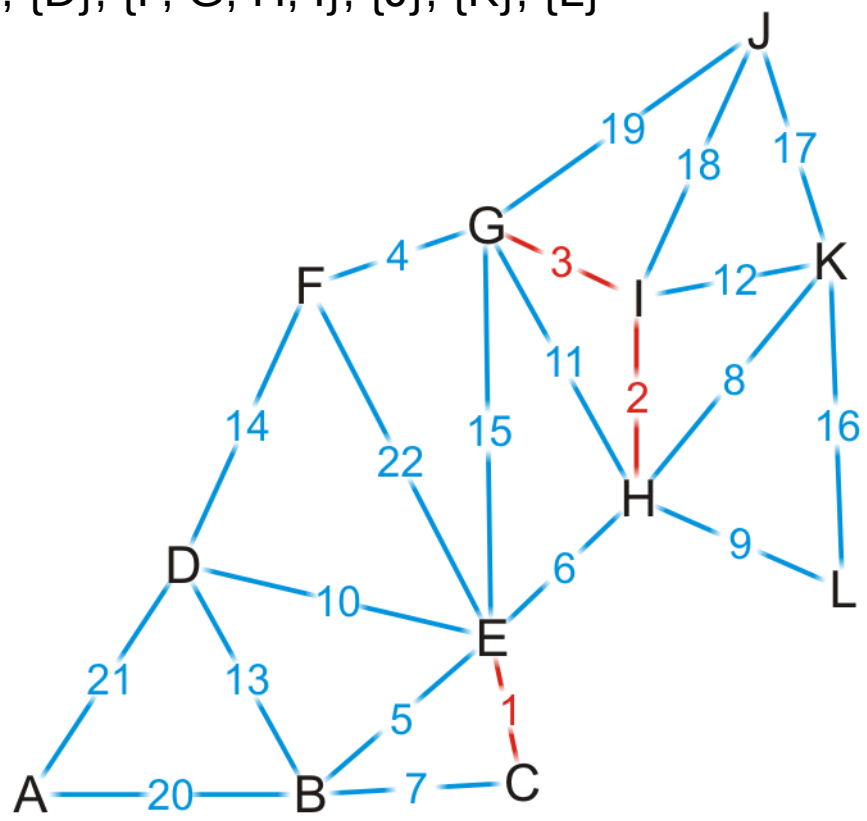


{C, E}
→ {H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Similarly, we add {G, I}, {F, G}, {B, E}

{A}, {B, C, E}, {D}, {F, G, H, I}, {J}, {K}, {L}

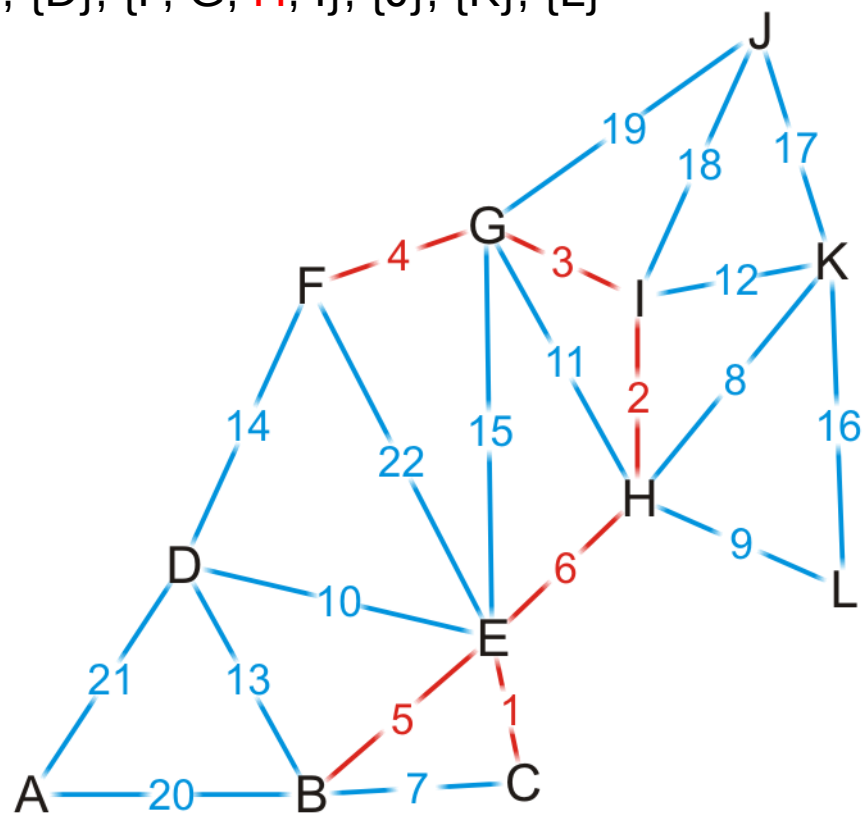


{C, E}
{H, I}
→ {G, I}
→ {F, G}
→ {B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

The vertices of {E, H} are in different sets

{A}, {B, C, E}, {D}, {F, G, H, I}, {J}, {K}, {L}

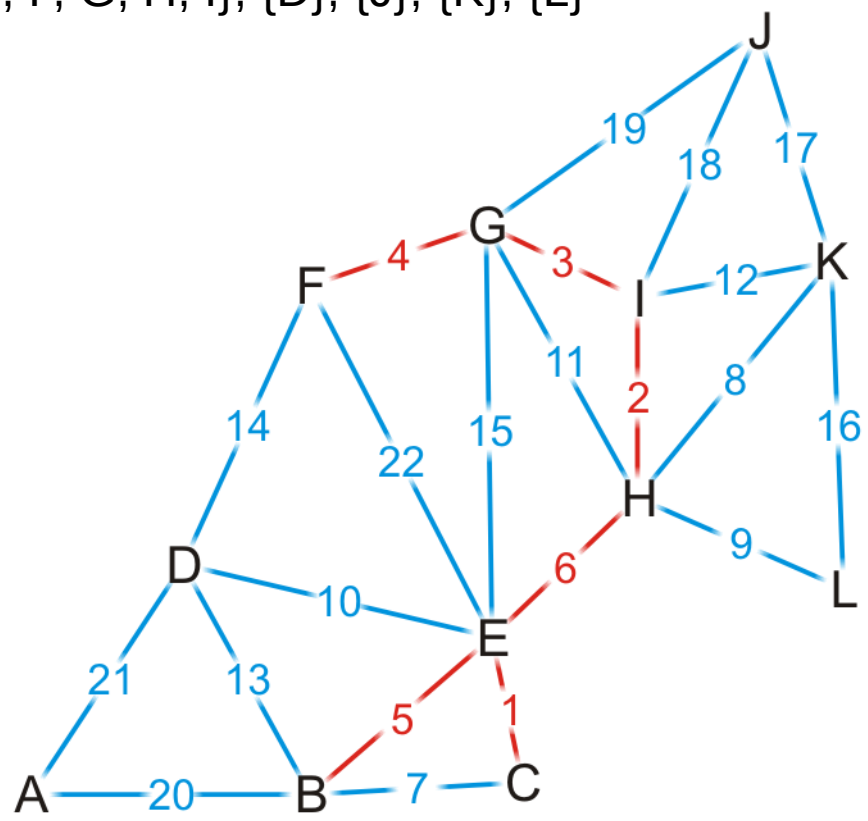


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
→ {E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Adding edge {E, H} creates a larger union

{A}, {B, C, E, F, G, H, I}, {D}, {J}, {K}, {L}

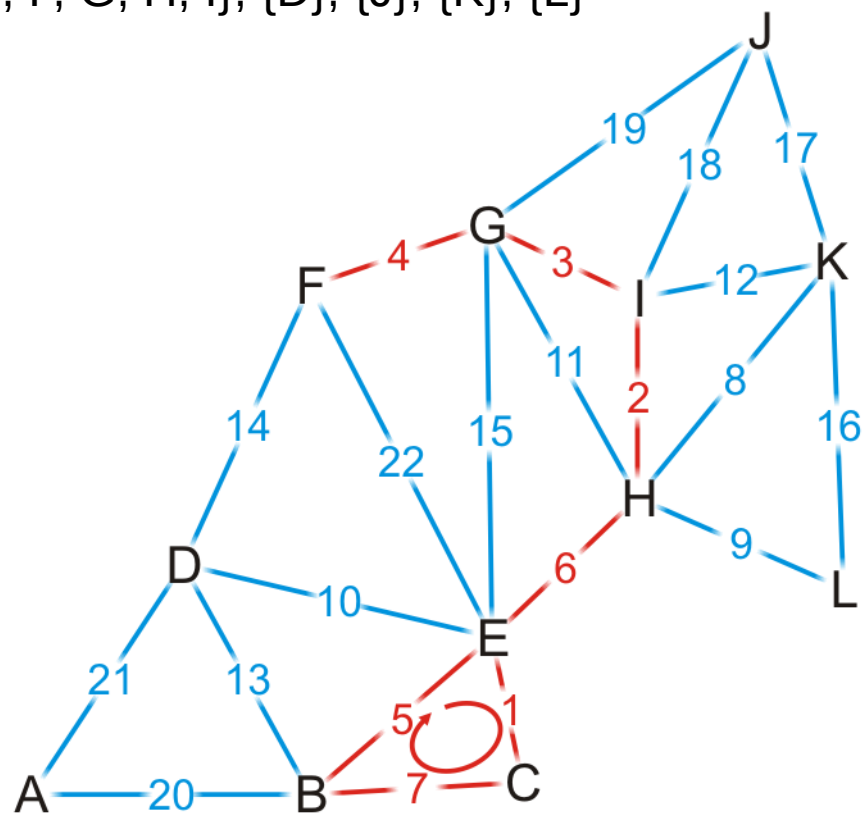


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
→ {E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We try adding {B, C}, but it creates a cycle

{A}, {B, C, E, F, G, H, I}, {D}, {J}, {K}, {L}

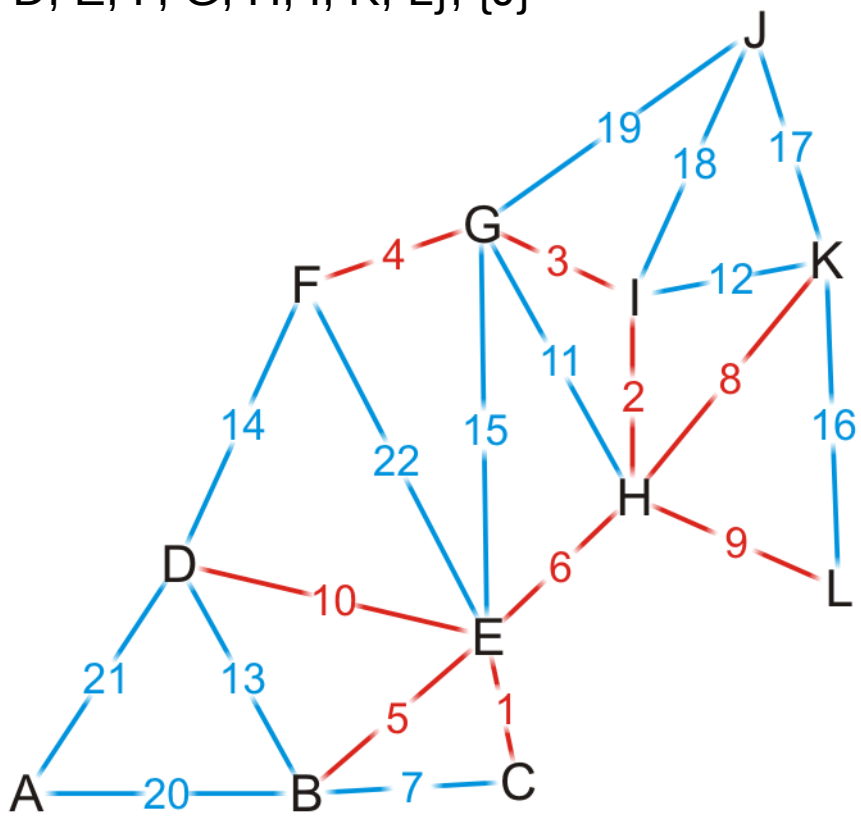


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
→ {B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We add edge {H, K}, {H, L} and {D, E}

{A}, {B, C, D, E, F, G, H, I, K, L}, {J}

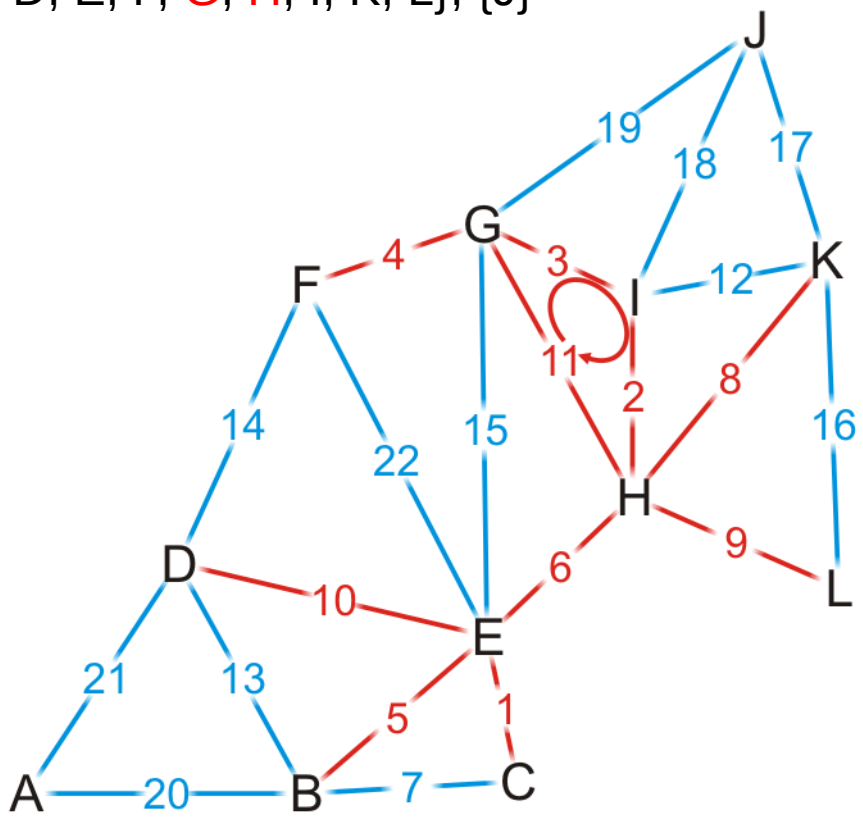


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
→ {H, K}
→ {H, L}
→ {D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Both G and H are in the same set

{A}, {B, C, D, E, F, G, H, I, K, L}, {J}

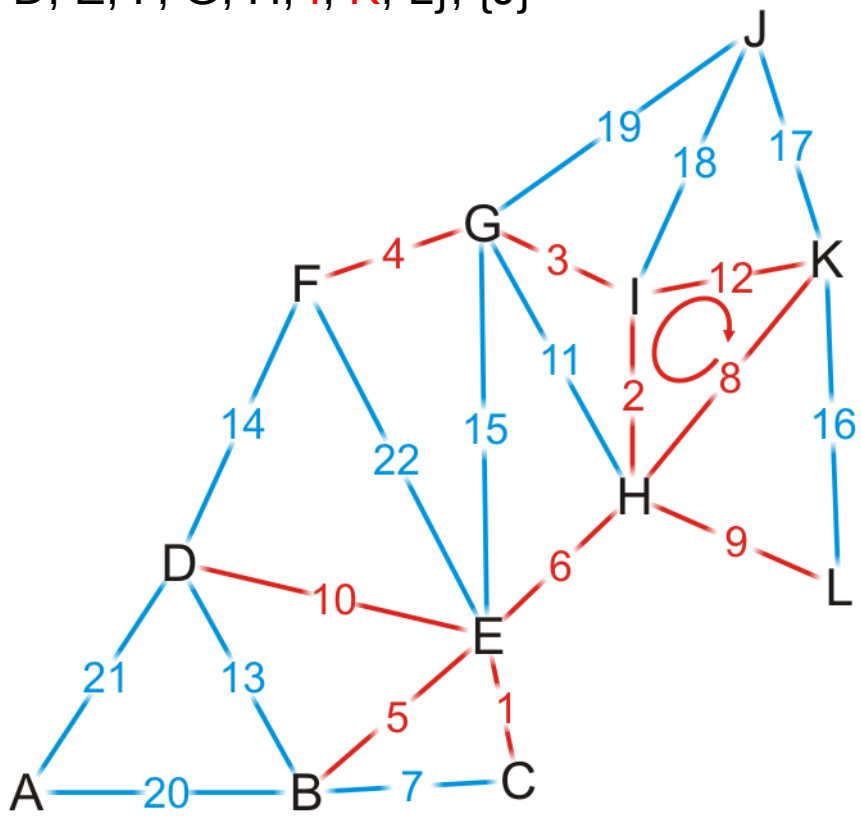


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
→ {G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Both {I, K} are in the same set

{A}, {B, C, D, E, F, G, H, I, K, L}, {J}

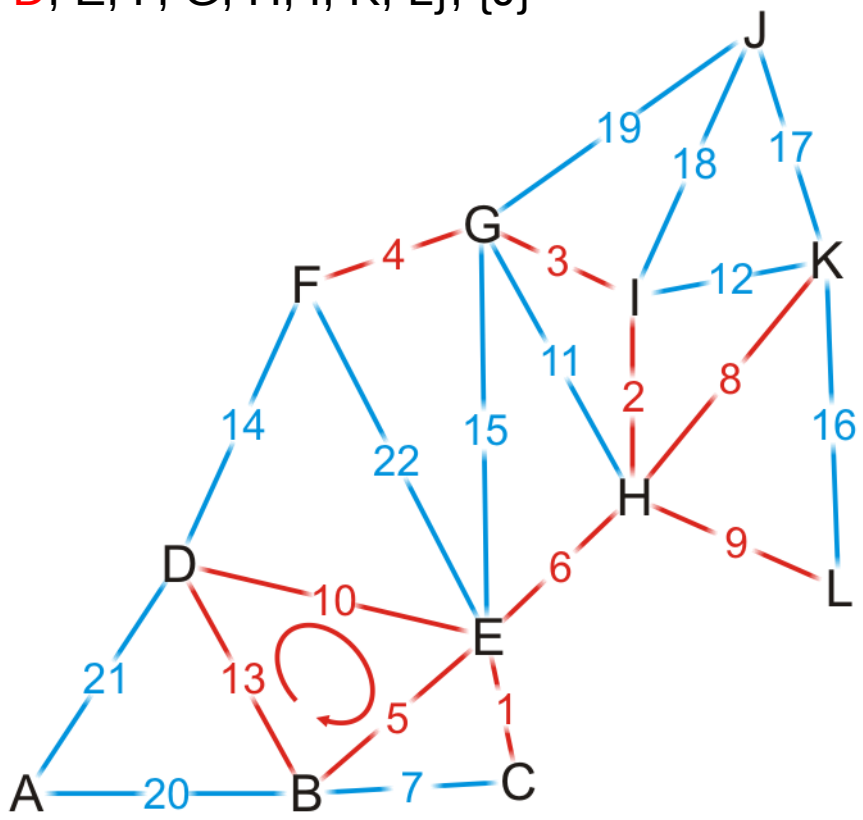


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
→ {I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Both {B, D} are in the same set

{A}, {B, C, D, E, F, G, H, I, K, L}, {J}

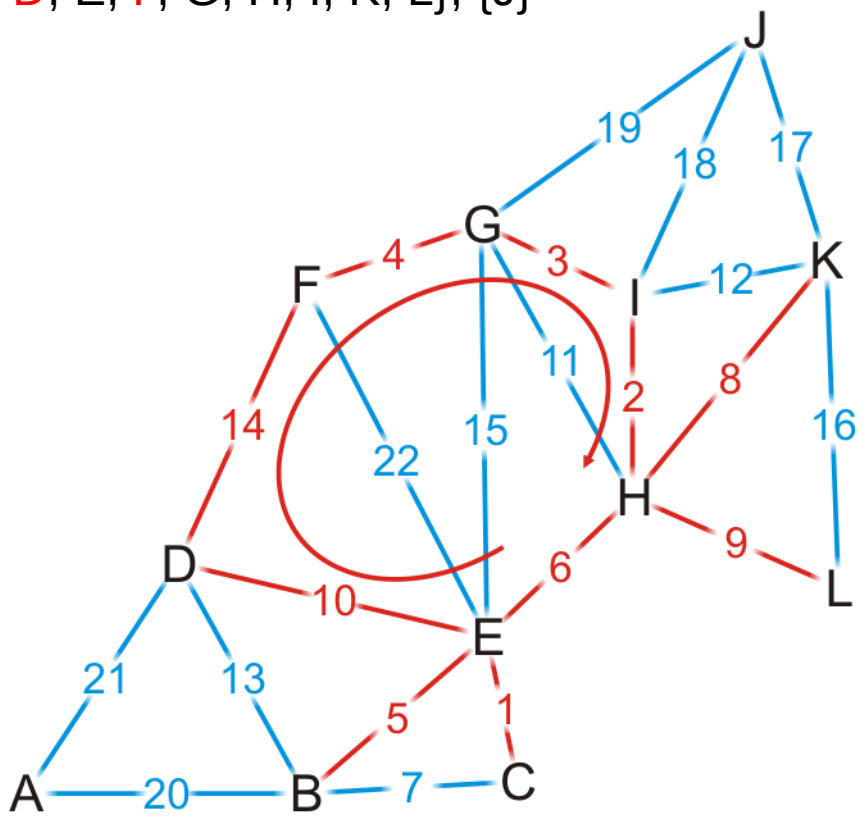


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
→ {B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

Both {D, F} are in the same set

{A}, {B, C, D, E, F, G, H, I, K, L}, {J}

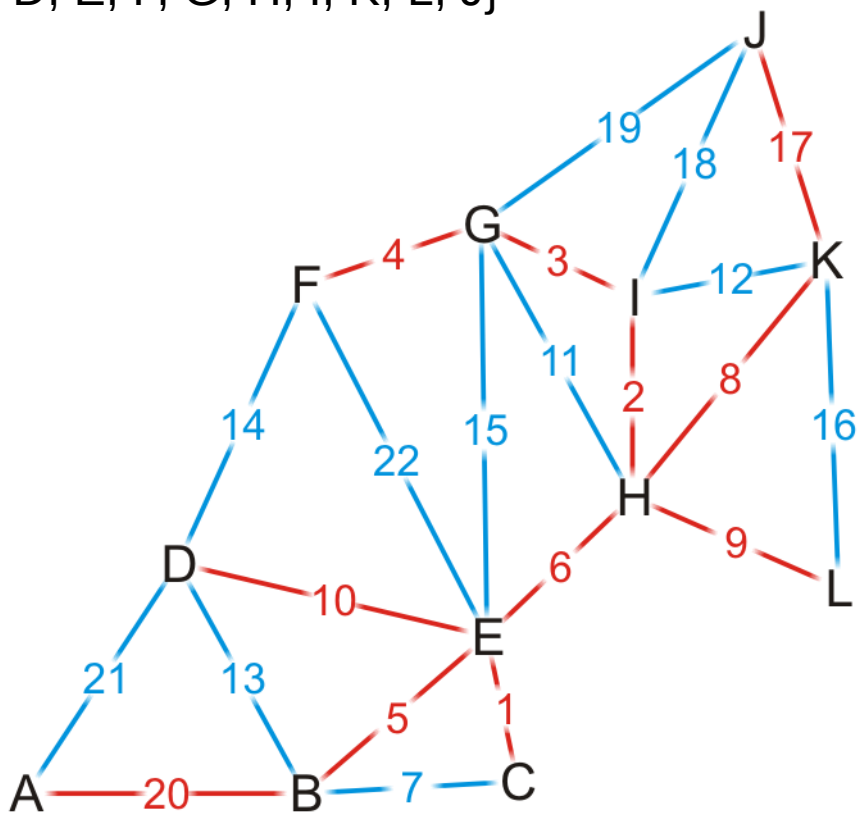


{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
→ {D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
{A, B}
{A, D}
{E, F}

Example

We end when there is only one set, having added (A, B)

{A, B, C, D, E, F, G, H, I, K, L, J}



{C, E}
{H, I}
{G, I}
{F, G}
{B, E}
{E, H}
{B, C}
{H, K}
{H, L}
{D, E}
{G, H}
{I, K}
{B, D}
{D, F}
{E, G}
{K, L}
{J, K}
{J, I}
{J, G}
→ {A, B}
{A, D}
{E, F}