

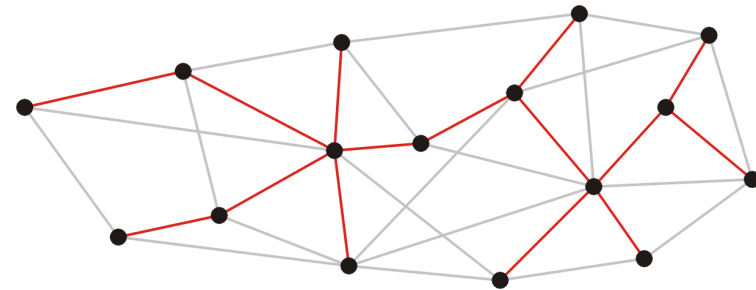
Minimum spanning trees



Carlos Moreno

cmoreno@uwaterloo.ca

EIT-4103



<https://ece.uwaterloo.ca/~cmoreno/ece250>

These slides, the course material, and course web site are based on work by Douglas W. Harder

Minimum spanning trees

Standard reminder to set phones to
silent/vibrate mode, please!



Minimum spanning trees

- During today's lesson:
 - Introduce the notion of spanning tree for a connected graph
 - Discuss the notion of minimum spanning trees
 - Look into two algorithms to find a minimum spanning tree:
 - Prim's algorithm
 - Kruskal's algorithm

Minimum spanning trees

- Given a connected graph with n vertices, a spanning tree is a collection of $n-1$ edges that connect all n vertices.

Minimum spanning trees

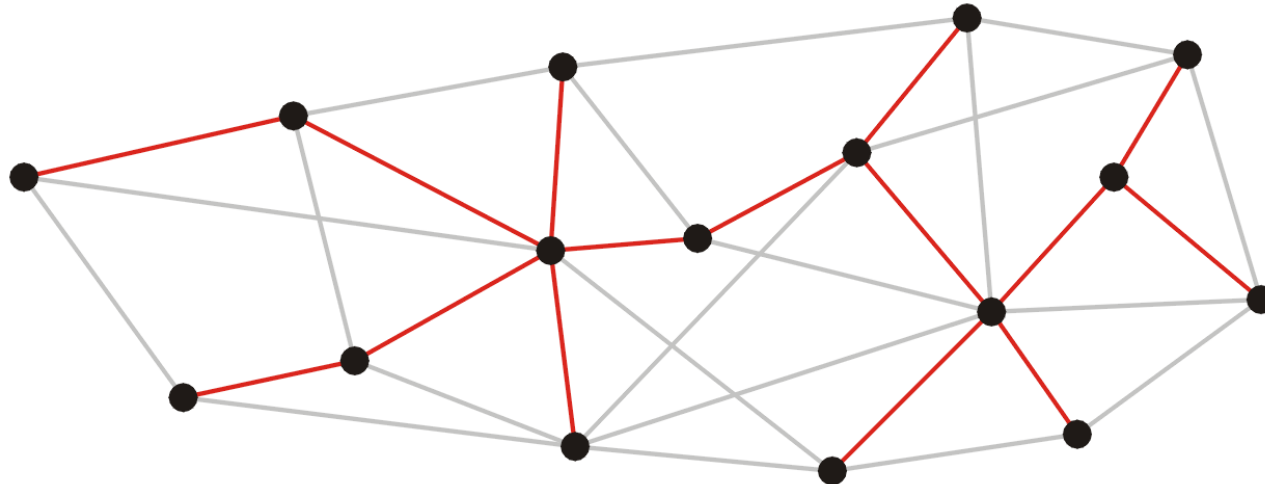
- Given a connected graph with n vertices, a spanning tree is a collection of $n-1$ edges that connect all n vertices.
 - $n-1$ is the minimum number of edges required to connect n vertices, resulting in a tree structure.
 - If we take any vertex to be the root, we form a tree by treating adjacent vertices as children.

Minimum spanning trees

- Given a connected graph with n vertices, a spanning tree is a collection of $n-1$ edges that connect all n vertices.
 - $n-1$ is the minimum number of edges required to connect n vertices, resulting in a tree structure.
 - If we take any vertex to be the root, we form a tree by treating adjacent vertices as children.
- We observe that a spanning tree is not necessarily unique.

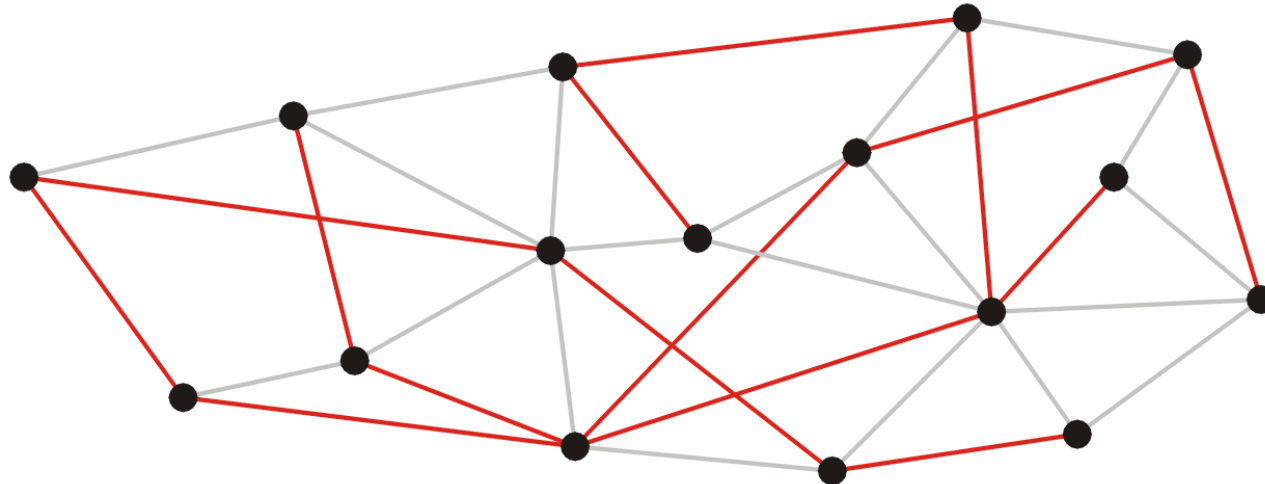
Minimum spanning trees

- This is an example of a spanning tree:



Minimum spanning trees

- For the same graph, this is also a spanning tree:



Minimum spanning trees

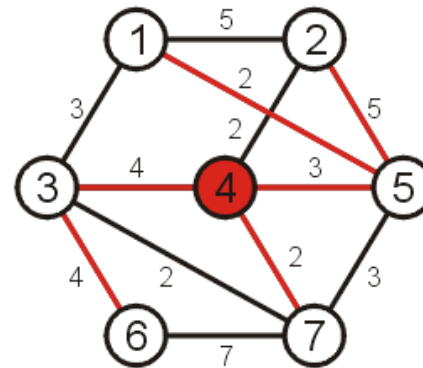
- As the name suggests, a spanning tree is a tree that spans the entire graph (i.e., that covers all the vertices, with connections restricted to be edges in the graph).

Minimum spanning trees

- If the graph is weighted, then a spanning tree has a weight, given by the sum of the edges that constitute the spanning tree.

Minimum spanning trees

- If the graph is weighted, then a spanning tree has a weight, given by the sum of the edges that constitute the spanning tree.
 - For example, the weight of this spanning tree is 20.



Minimum spanning trees

- A minimum spanning tree is a spanning tree with minimum weight.

Minimum spanning trees

- A minimum spanning tree is a spanning tree with minimum weight.
 - A minimum spanning tree is not necessarily unique!
 - That is, there may be several different spanning trees with the same weight — a weight such that no spanning tree has a weight lower than this.

Minimum spanning trees

- A (possibly obvious) observation:
 - The problem of finding a minimum spanning tree does not really make sense for unweighted graphs (which would be seen as weighted graphs with all edges having a weight of 1)

Minimum spanning trees

- A (possibly obvious) observation:
 - The problem of finding a minimum spanning tree does not really make sense for unweighted graphs (which would be seen as weighted graphs with all edges having a weight of 1)
 - *Why?*

Minimum spanning trees

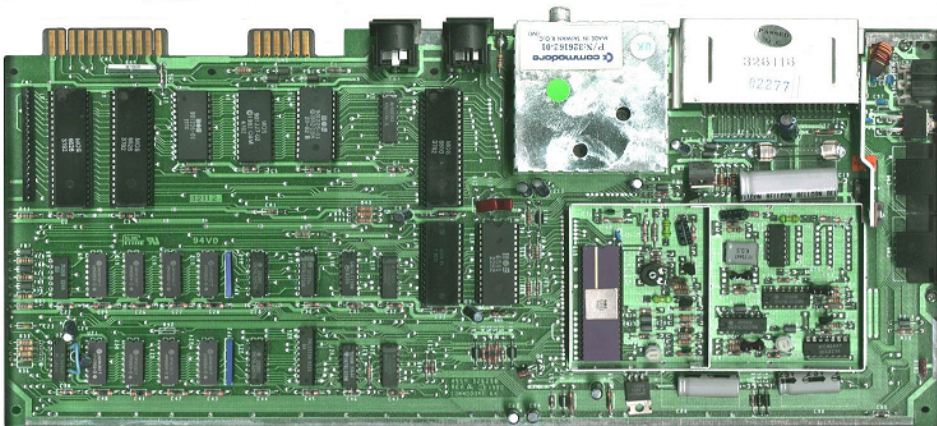
- A (possibly obvious) observation:
 - The problem of finding a minimum spanning tree does not really make sense for unweighted graphs (which would be seen as weighted graphs with all edges having a weight of 1)
 - *Why?*
 - We recall that a tree with n nodes has exactly $n-1$ edges; if all edges have equal weight, then every spanning tree is a minimum spanning tree, and finding one is trivial.

Minimum spanning trees

- Examples of applications are numerous. A couple of simple ones:

Minimum spanning trees

- Examples of applications are numerous. A couple of simple ones:
 - Supplying power to all circuit elements on a board
 - To a number of points (loads) in a building or local area.
 - A minimum spanning tree gives the lowest-cost option.

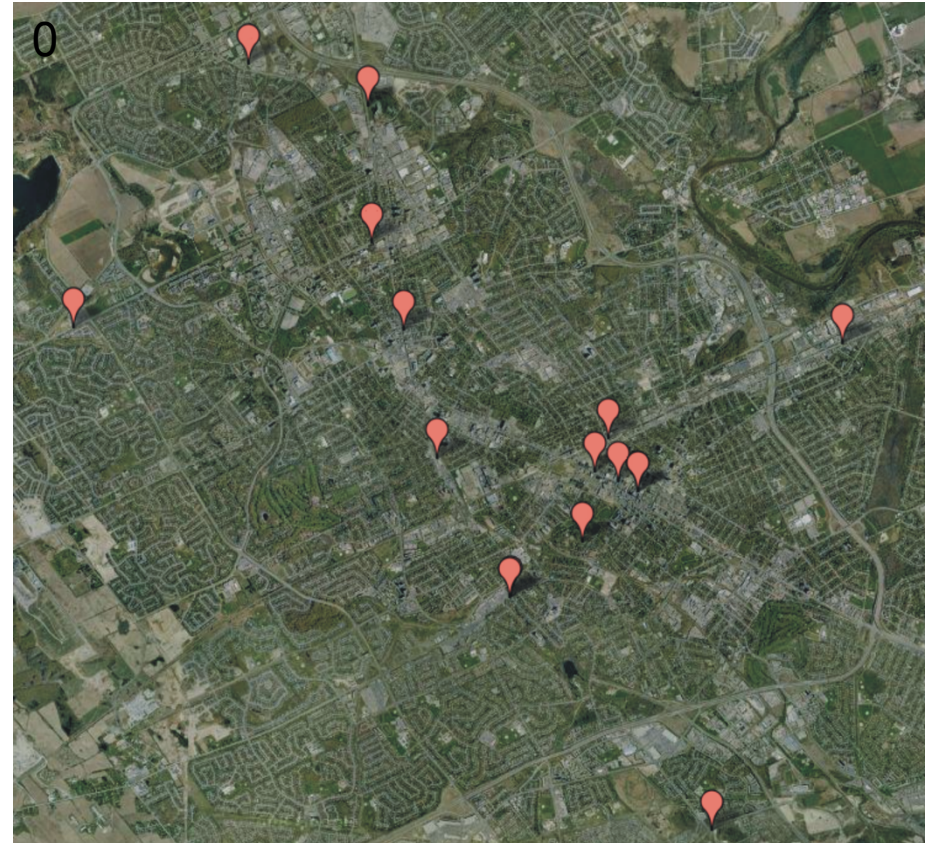


www.kpmb.com

www.commedore.ca

Minimum spanning trees

- Connecting a number of LANs (local area networks)
 - Weights could represent a monetary cost.



Minimum spanning trees

- Connecting a number of LANs (local area networks)
 - Weights could represent a monetary cost.
 - A minimum spanning tree would reveal the optimal (least expensive) solution.



Minimum spanning trees

- Let's look at one of the algorithms (Prim's algorithm).

Minimum spanning trees

- Let's look at one of the algorithms (Prim's algorithm).
 - Side note: as you may have noticed, you will implement (partially) Prim's algorithm for your last lab project.

Minimum spanning trees

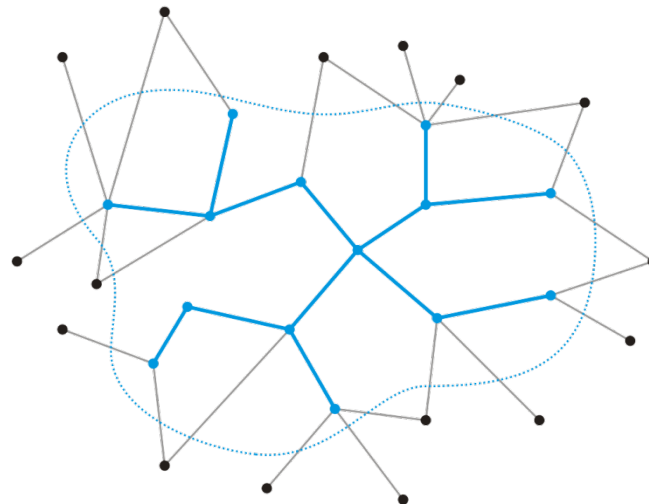
- Prim's algorithm has certain aspects in common with Dijkstra's algorithm.
 - At each iteration, the spanning tree is expanded by choosing the vertex with smallest distance to the “current” spanning tree.
 - Similar idea, and in fact, as we'll see, the reason why it works (and the argument to prove that this step works) is almost identical to Dijkstra's algorithm.

Minimum spanning trees

- Prim's algorithm has certain aspects in common with Dijkstra's algorithm.
 - At each iteration, the spanning tree is expanded by choosing the vertex with smallest distance to the “current” spanning tree.
 - Similar idea, and in fact, as we'll see, the reason why it works (and the argument to prove that this step works) is almost identical to Dijkstra's algorithm.
 - A key difference is that in Dijkstra's algorithm we select the vertex with lowest distance (the “total” distance from the starting vertex) — with Prim's algorithm, we select the lowest distance given by the edge that connects to the current spanning tree.

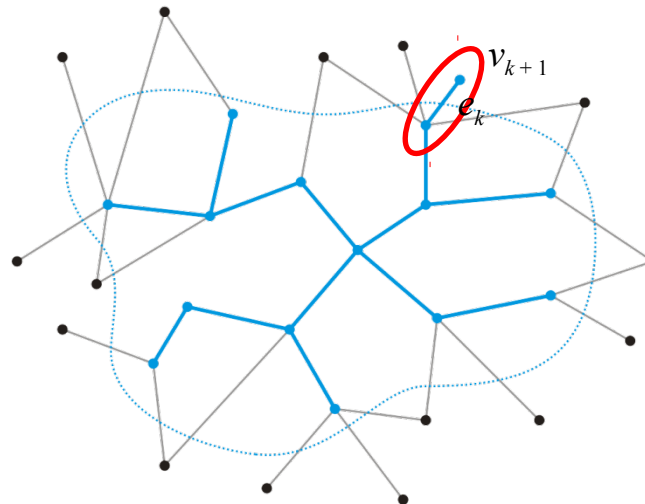
Minimum spanning trees

- Suppose we have determined a minimum spanning tree on k of the n vertices.
- We want to extend this to $k+1$ vertices.



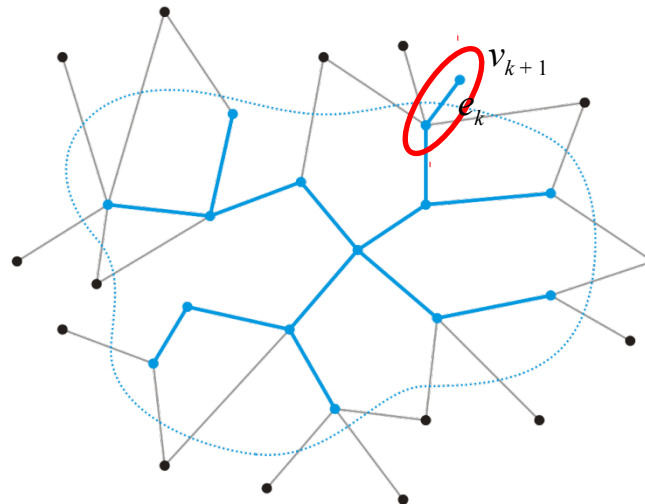
Minimum spanning trees

- We choose the vertex joined by the lowest weight edge, e_k .



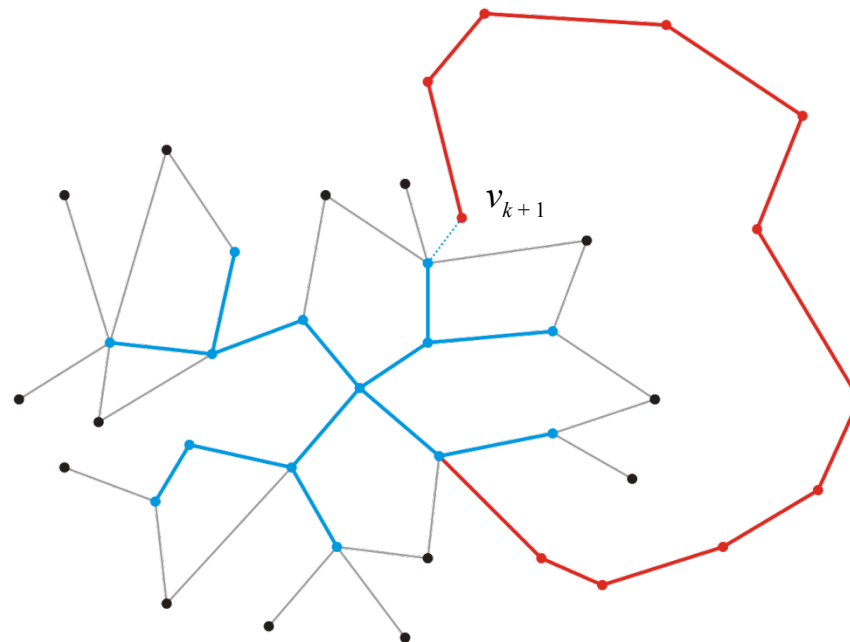
Minimum spanning trees

- How do we know for sure that this edge, e_k , is part of the minimum spanning tree for the n vertices?



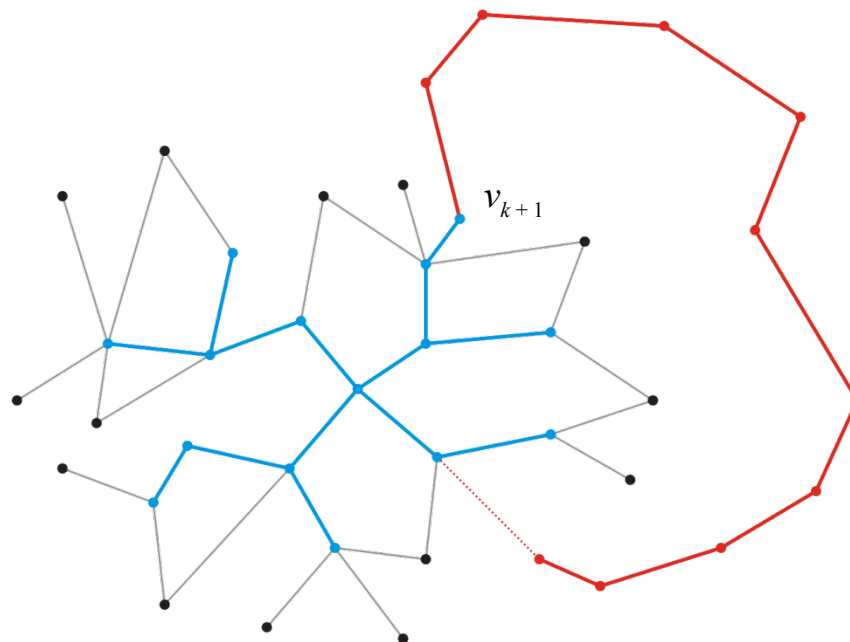
Minimum spanning trees

- Suppose it's not — that is, that v_{k+1} ends up being connected to the spanning tree through some other sequence of vertices.



Minimum spanning trees

- Since e_k was the smallest weight joining any vertex to the spanning tree of k nodes, we can find a spanning tree with lower weight, contradicting the assumption that this one was a minimum spanning tree.



Minimum spanning trees

- The algorithm is quite simple:
- 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 NULL

Minimum spanning trees

- Then, iterate while there are unvisited vertices with distance $< \infty$
 - Select the unvisited vertex with minimum distance
 - Mark that vertex as visited
 - For each adjacent vertex, if the weight of the connecting edge is less than the current distance associated to that vertex:
 - Update the distance to equal the weight of the edge
 - Set the current vertex as the parent of that adjacent vertex

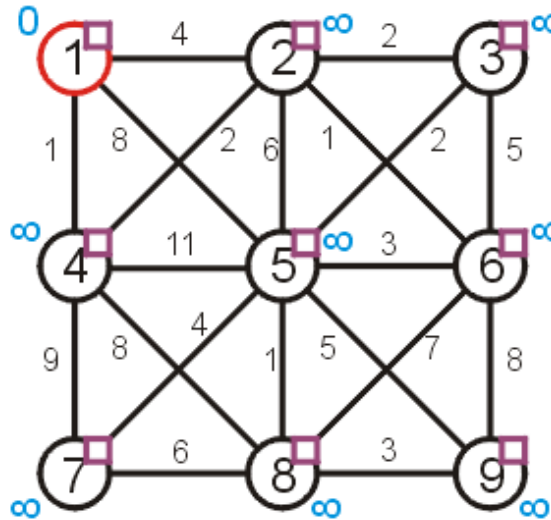
Minimum spanning trees

- Here's a demo/example of execution, directly from Professor Harder's slides:

Prim's Algorithm

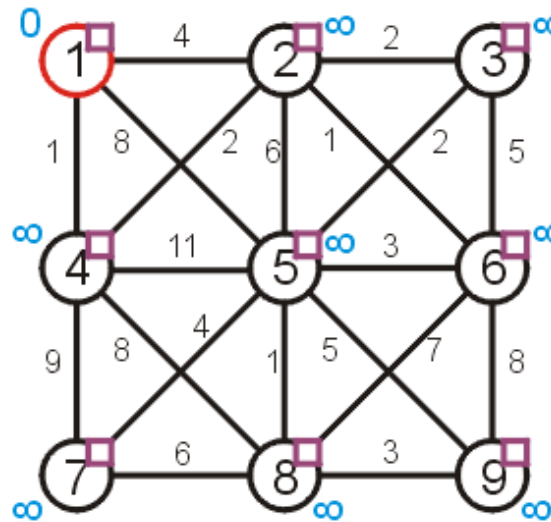
Prim's Algorithm

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

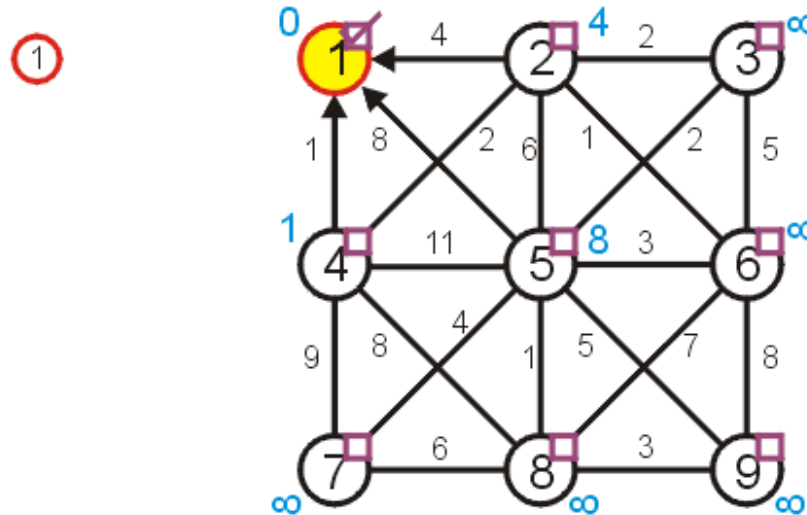


		Distance	Parent
1	F	0	0
2	F	∞	0
3	F	∞	0
4	F	∞	0
5	F	∞	0
6	F	∞	0
7	F	∞	0
8	F	∞	0
9	F	∞	34 0

Prim's Algorithm

Prim's Algorithm

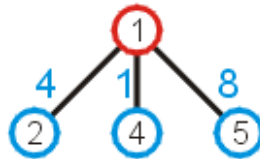
Visiting vertex 1, we update vertices 2, 4, and 5



		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	∞	35 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:



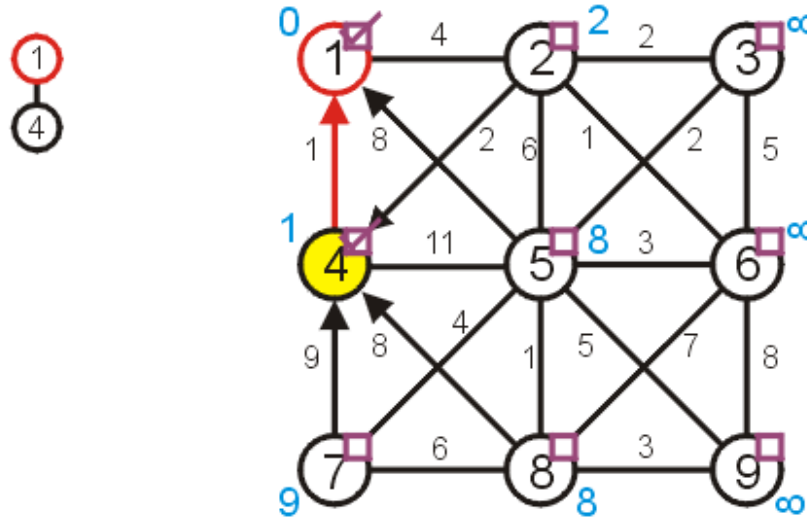
As we wish to find a *minimum* spanning tree, it makes sense we add that vertex with a connecting edge with least weight

Prim's Algorithm

Prim's Algorithm

The next unvisited vertex with minimum distance is vertex 4

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



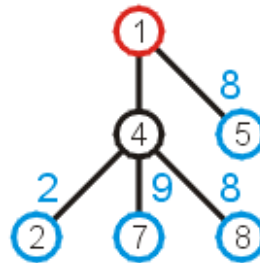
		Distance	Parent
1	T	0	0
2	F	2	4
3	F	∞	0
4	T	1	1
5	F	8	1
6	F	∞	0
7	F	9	4
8	F	8	4
9	F	∞	³⁷ 0

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)

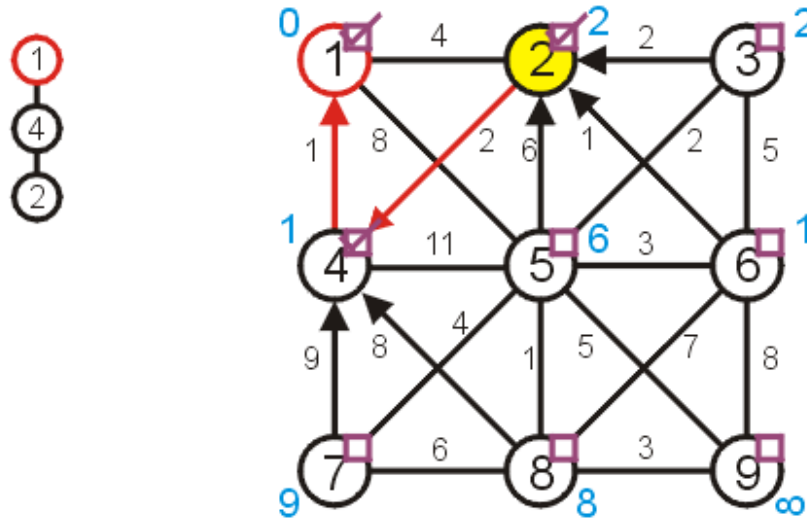


Prim's Algorithm

Prim's Algorithm

Next visit vertex 2

- Update 3, 5, and 6

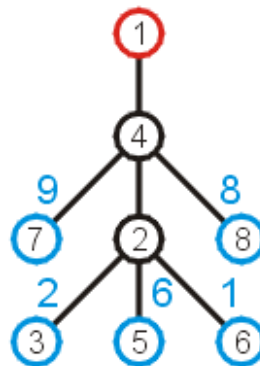


		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	6	2
6	F	1	2
7	F	9	4
8	F	8	4
9	F	∞	³⁹ 0

Prim's Algorithm

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

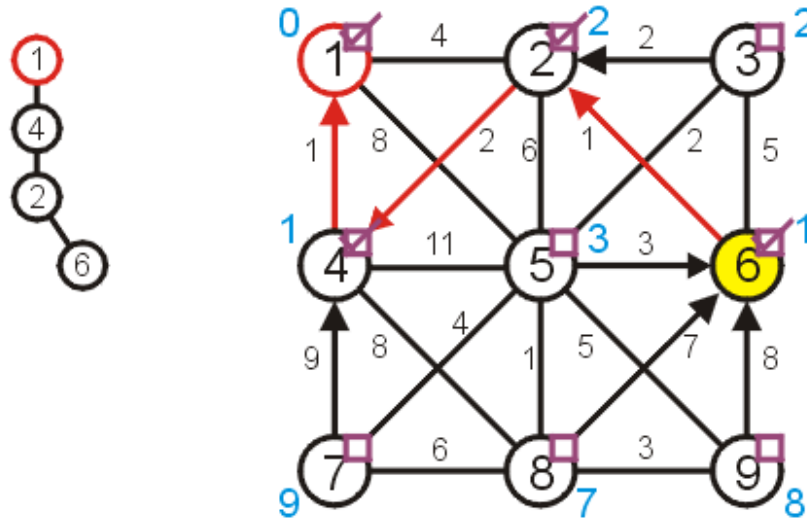


Prim's Algorithm

Prim's Algorithm

Next, we visit vertex 6:

- update vertices 5, 8, and 9



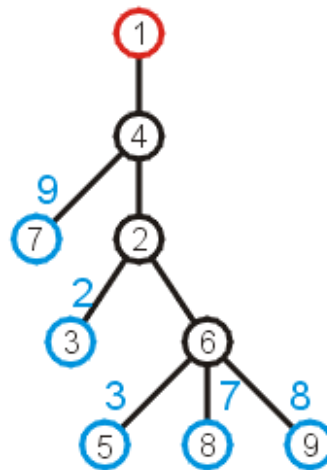
		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	3	6
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	4 ¹ 6

Prim's Algorithm

Prim's Algorithm

The edge with least weight is (2, 3)

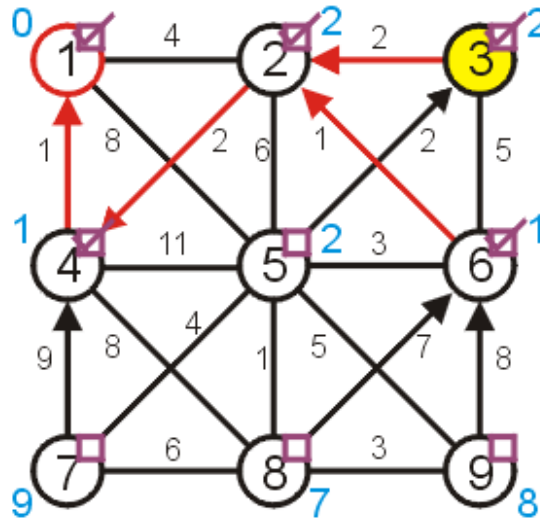
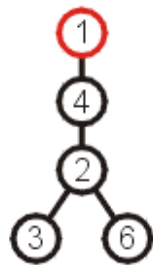
- This adds the weight of 2 to the weight minimum spanning tree



Prim's Algorithm

Prim's Algorithm

Next, we visit vertex 3 and update 5

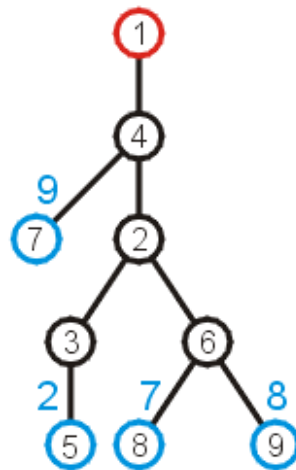


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	F	2	3
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	4 ³ 6

Prim's Algorithm

Prim's Algorithm

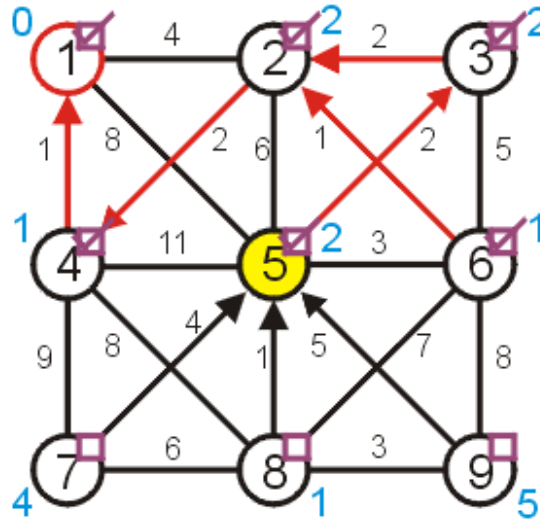
At this point, we can extend the tree by adding the edge (3, 5)



Prim's Algorithm

Prim's Algorithm

Visiting vertex 5, we update 7, 8, 9

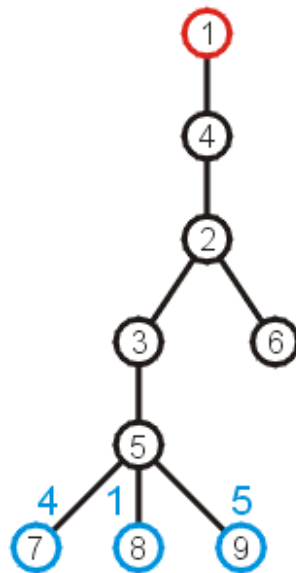


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	F	1	5
9	F	5	⁴⁵ 5

Prim's Algorithm

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

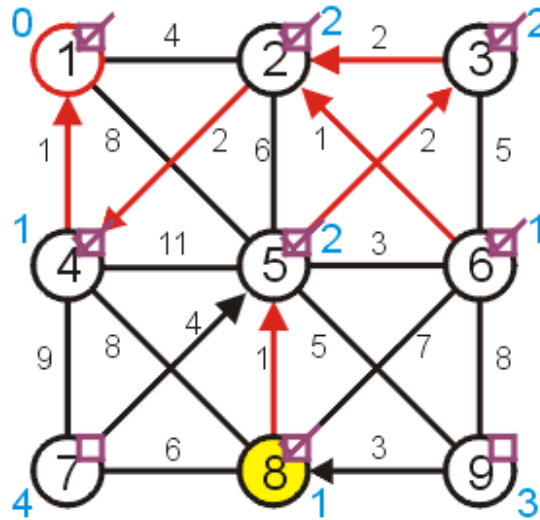
The edge to 8 has the least weight



Prim's Algorithm

Prim's Algorithm

Visiting vertex 8, we only update vertex 9

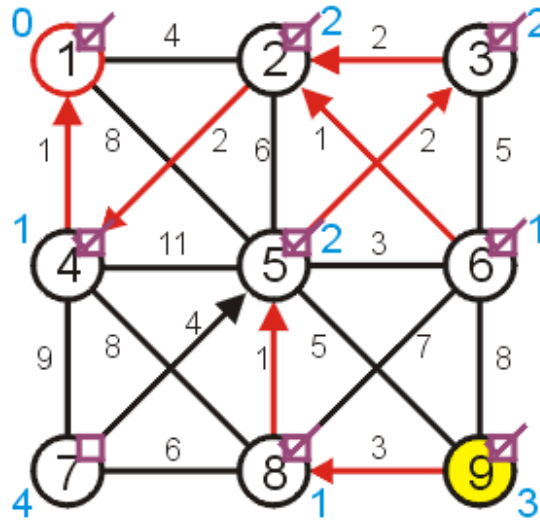


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	F	3	4 ⁷ 8

Prim's Algorithm

Prim's Algorithm

There are no other vertices to update while visiting vertex 9

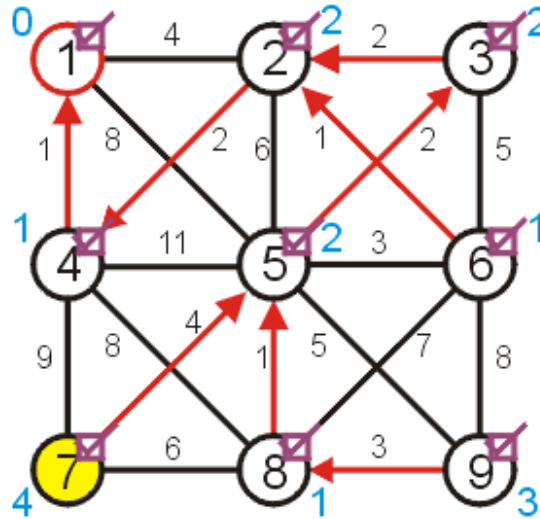
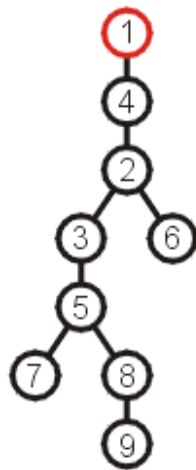


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	T	3	⁴⁸ 8

Prim's Algorithm

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

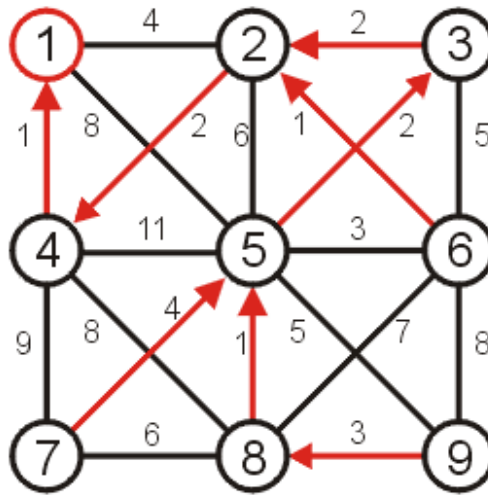
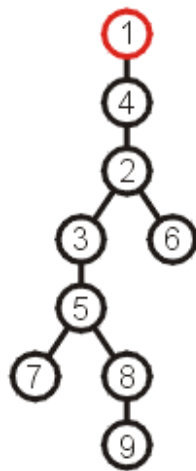
If at any point, all remaining vertices had a distance of ∞ , this would indicate that the graph is not connected

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

Prim's Algorithm

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	5 ¹ 8

Minimum spanning trees

- Next, we'll take a quick look at Kruskal's algorithm.

Minimum spanning trees

- Kruskal's algorithm takes a different — but also interesting — approach:
- Put the edges in order by weight, and add the lowest weight edge to the spanning tree if it does not create a cycle.

Minimum spanning trees

- Kruskal's algorithm takes a different — but also interesting — approach:
- Put the edges in order by weight, and add the lowest weight edge to the spanning tree if it does not create a cycle.
 - How do we (efficiently) determine whether adding an edge will create a cycle?

Minimum spanning trees

- How do we (efficiently) determine whether adding an edge will create a cycle?
 - The idea is to keep track of subsets of vertices that are connected. When considering an edge:
 - If the two vertices are in the same subset, discard the edge (it would produce a cycle)
 - Otherwise (i.e., if the two vertices are in two different subsets), then all the vertices in both subsets will now be part of a single subset (since the added edge is connecting them)
 - In other words, replace the two sets by the union of them.

Minimum spanning trees

- Notice that this is reasonably easy and efficient — we start with n sets, each containing one of the n vertices in the graph.
 - Normally, one could use a *disjoint sets* data structure.
 - But we could get away with an array where each vertex is associated with the set where it is — the set could have an arbitrary ID; the only thing that matters is that we can determine whether two vertices are in the same set; doesn't really matter what that set is called.

Summary

- Notice that this is reasonably easy and efficient — we start with n sets, each containing one of the n vertices in the graph.
 - Normally, one could use a *disjoint sets* data structure.
 - But we could get away with an array where each vertex is associated with the set where it is — the set could have an arbitrary ID; the only thing that matters is that we can determine whether two vertices are in the same set; doesn't really matter what that set is called.