Graphs 3

Anders Kalhauge and Marjahan Begum

Further Graph Algorithms

- Spanning Tree
- Minimum Spanning Tree
- Shrotest Paths
  - Dijkstra’s algorithm

Spanning Tree

- An acyclic graph is a graph with no cycles
- A tree is an acyclic connected graph
- A spanning tree of a connected graph is a subgraph that contains all of that graph's vertices and is a single tree

Spanning Tree

- Spanning Trees: A subgraph of a undirected graph G = (V,E) is a spanning tree of G if it is a tree and contains every vertex of G

Minimum Spanning Tree

- A Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

- There also can be many minimum spanning trees.

Direct Application

- Consider supplying power to
  - All circuit elements on a board
  - A number of loads within a building

- Cluster Analysis

- Handwriting recognition

- Image segmentation

- Bottom line - A minimum spanning tree will give the lowest-cost solution

Application

Consider an *ad hoc* wireless network

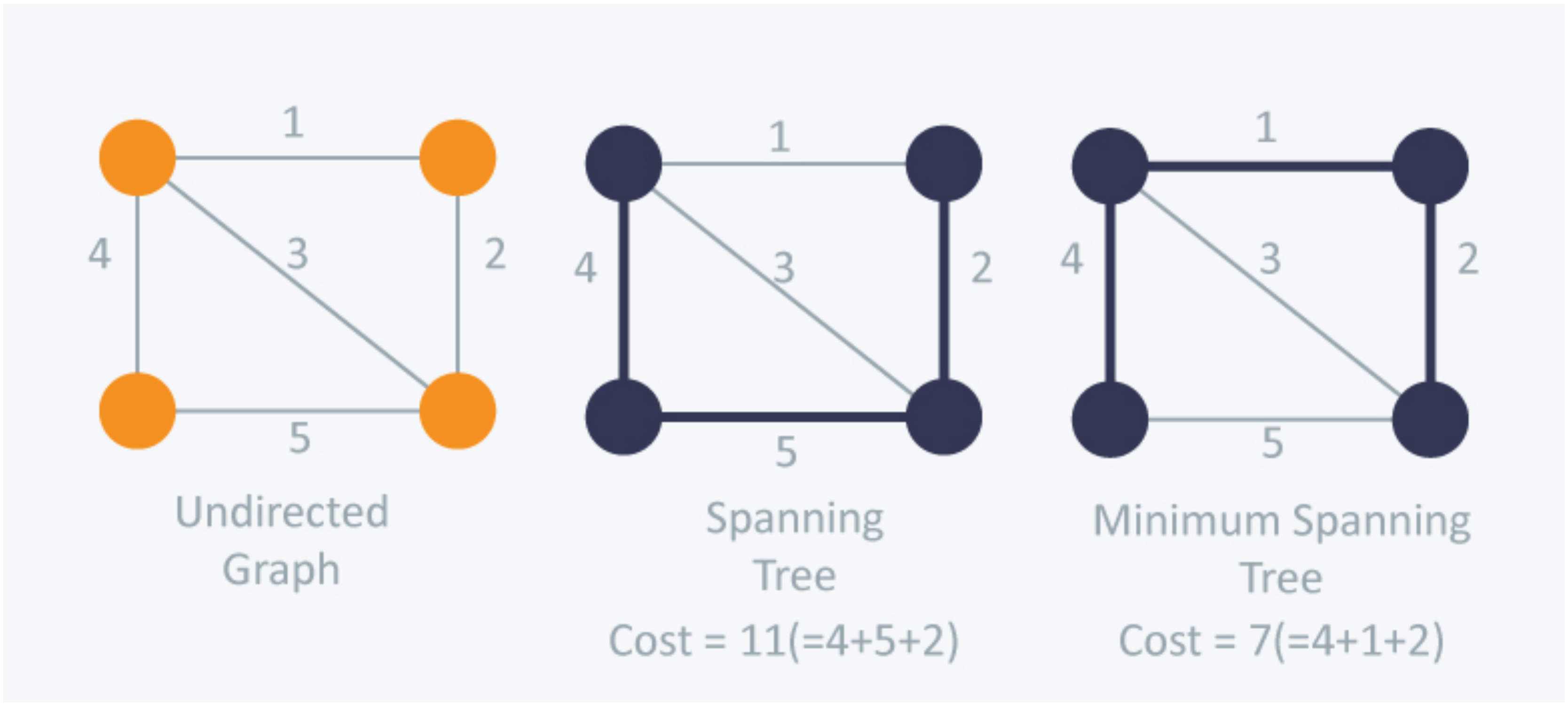
– Any two terminals can connect with any others

Problem:

– Errors in transmission increase with transmission length

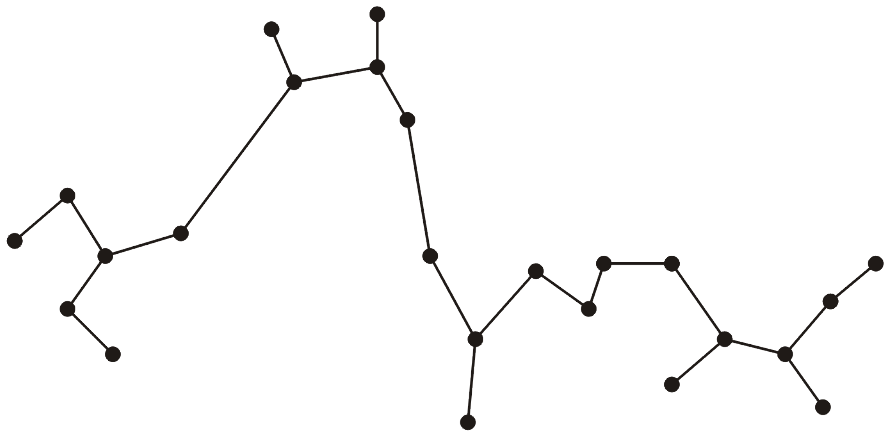
– Can we find clusters of terminals which can communicate safely?

Simple example



Application

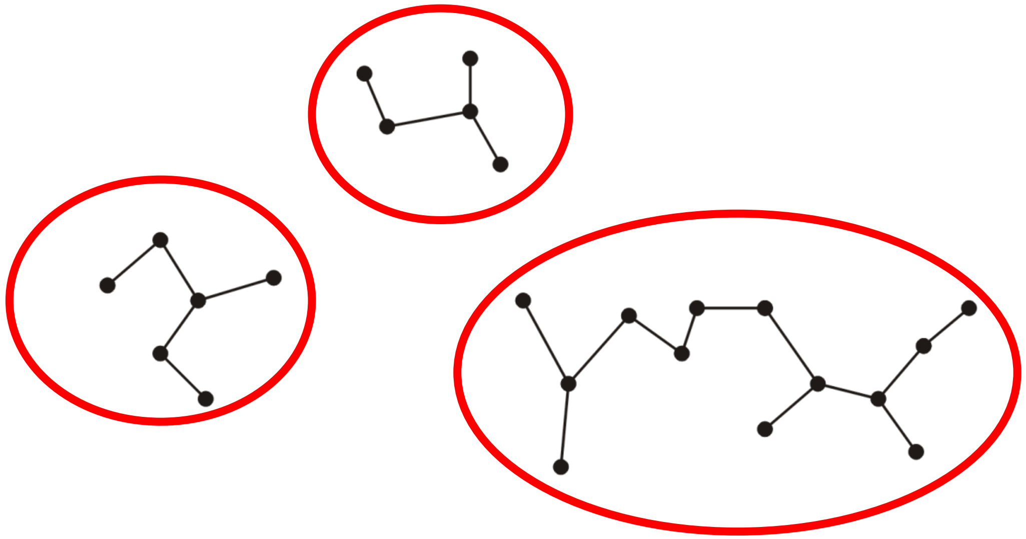
Find a minimum spanning tree



Application

Remove connections which are too long

This *clusters* terminals into smaller and more manageable sub-networks



Algorithms for MST

- Prim’s algorithm
- Kruskal’s algorithm

Prim’s Algorithm

Iterate while there exists an unvisited vertex with distance $< \infty$

- 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

Prim’s Algorithm

Halting Conditions:

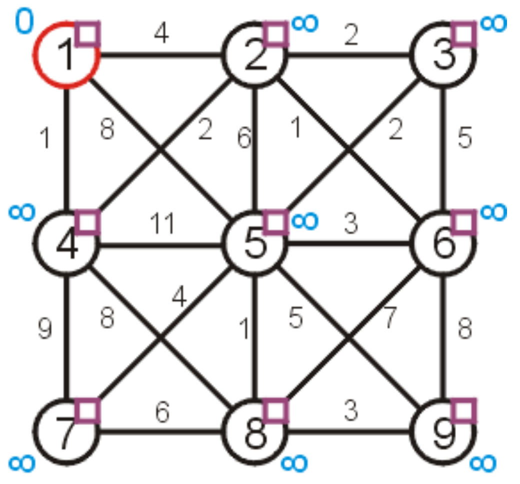
- There are no unvisited vertices which have a distance < ∞

If all vertices have been visited, we have a spanning tree of the entire graph

If there are vertices with distance ∞, then the graph is not connected and we only have a minimum spanning tree of the connected sub-graph containing the root

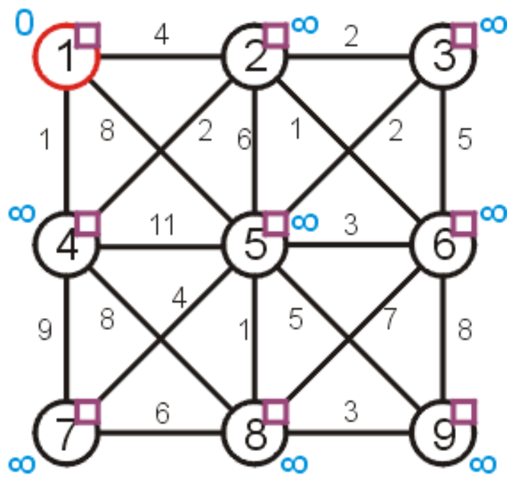
Prim's Algorithm

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



Prim’s Algorithm

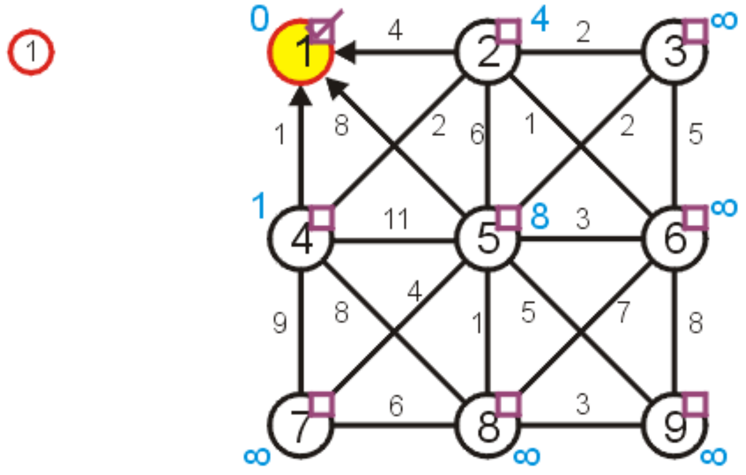
First we set up the appropriate table and initialize it



		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

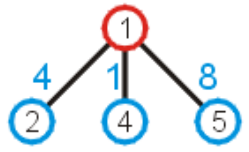
Visiting vertex 1, we update vertices 2, 4, and 5



		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:



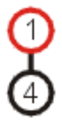
As we wish to find a *minimum* spanning tree, it makes sense we add that vertex with a connecting edge with least weight

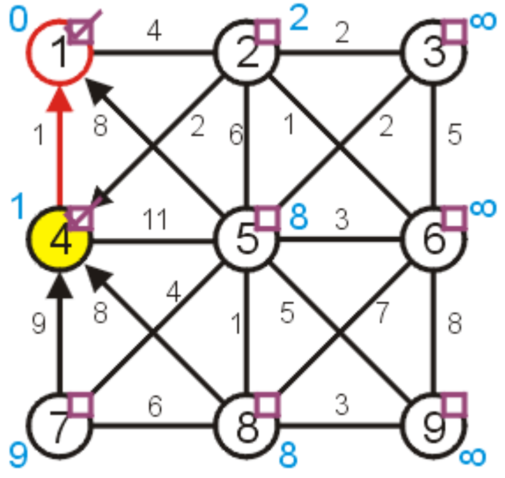
Prim's Algorithm

The next unvisited vertex with minimum distance is vertex 4

– Update vertices 2, 7, 8

– Don't update vertex 5



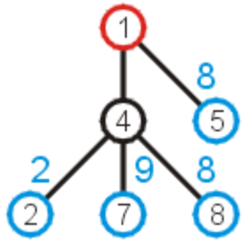


		Distance	Parent
1	T	0	0
2	F	2	4
3	F	∞	0
4	T	1	1
5	F	8	1
6	F	∞	0
7	F	9	4
8	F	8	4
9	F	∞	0

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)

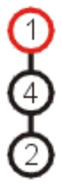


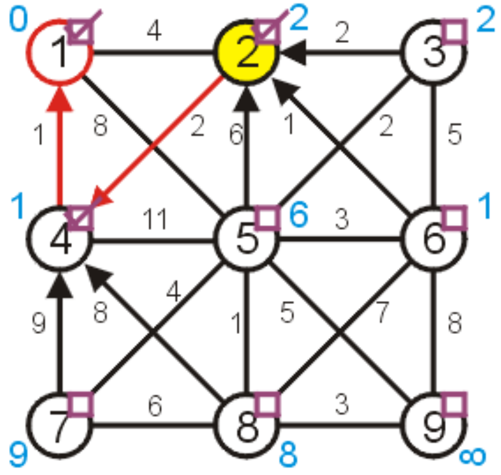
We add that edge with the least weight: (4, 2)

Prim's Algorithm

Next visit vertex 2

– Update 3, 5, and 6

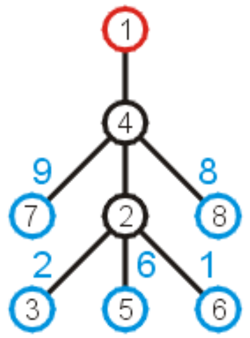




		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	6	2
6	F	1	2
7	F	9	4
8	F	8	4
9	F	∞	0

Prim’s Algorithm

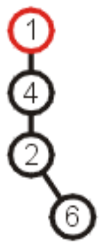
Again looking at the shortest edges to each of the vertices adjacent to the current tree, we note that we can add (2, 6) with the least increase in weight

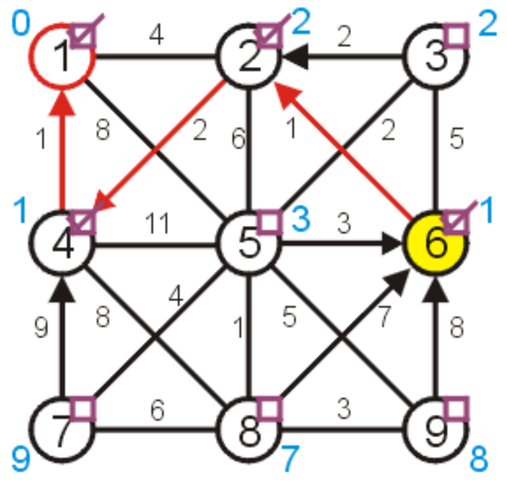


Prim's Algorithm

Next, we visit vertex 6:

– update vertices 5, 8, and 9



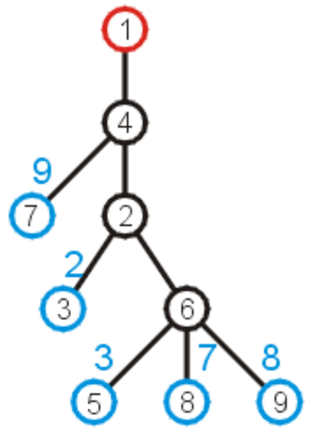


		Distance	Parent
1	T	0	0
2	T	2	4
3	F	2	2
4	T	1	1
5	F	3	6
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	6

Prim’s Algorithm

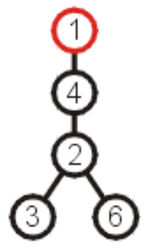
The edge with least weight is (2, 3)

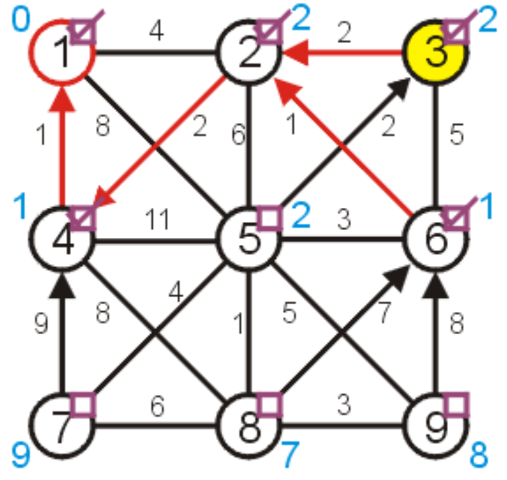
– This adds the weight of 2 to the weight minimum spanning tree



Prim's Algorithm

Next, we visit vertex 3 and update 5

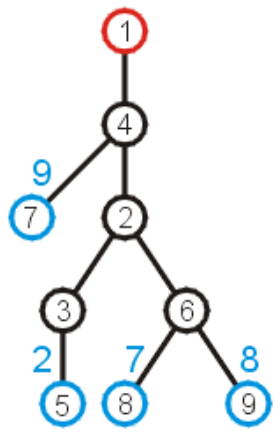




		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	F	2	3
6	T	1	2
7	F	9	4
8	F	7	6
9	F	8	6

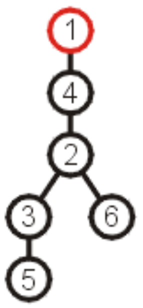
Prim’s Algorithm

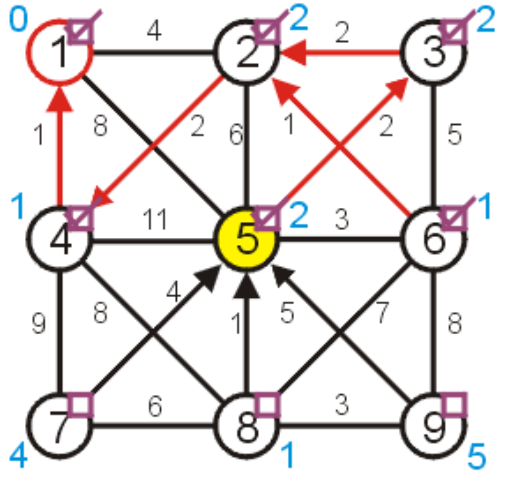
At this point, we can extend the tree by adding the edge (3, 5)



Prim's Algorithm

Visiting vertex 5, we update 7, 8, 9



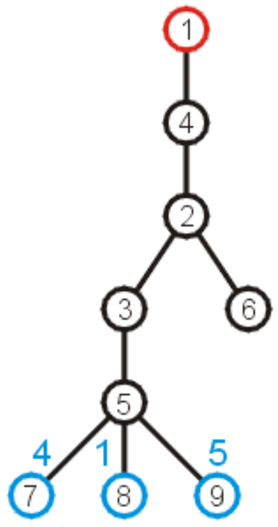


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	F	1	5
9	F	5	5

Prim’s Algorithm

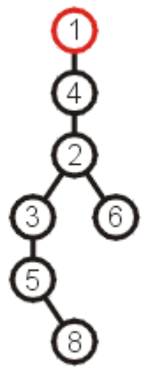
At this point, there are three possible edges which we could include which will extend the tree

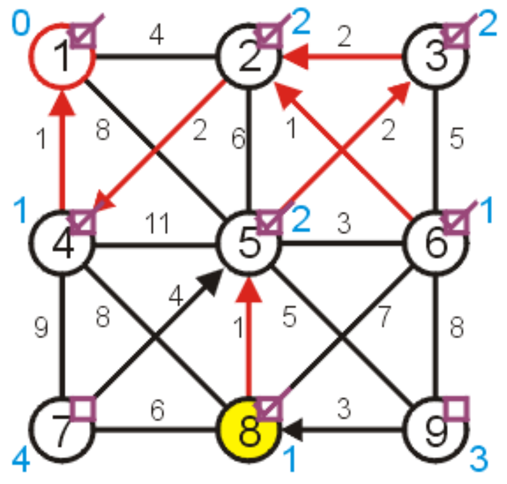
The edge to 8 has the least weight



Prim’s Algorithm

Visiting vertex 8, we only update vertex 9

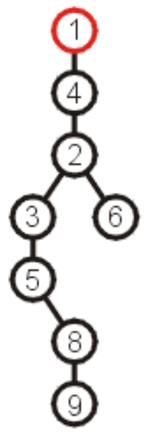


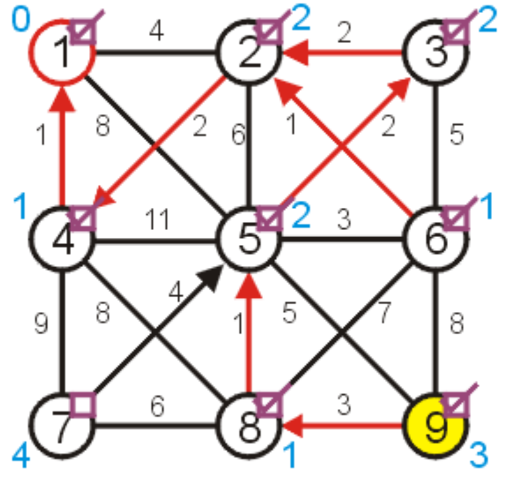


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	F	3	8

Prim's Algorithm

There are no other vertices to update while visiting vertex 9

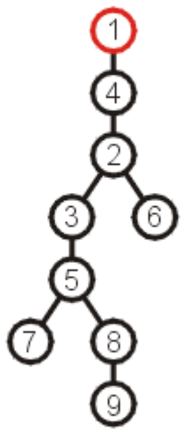


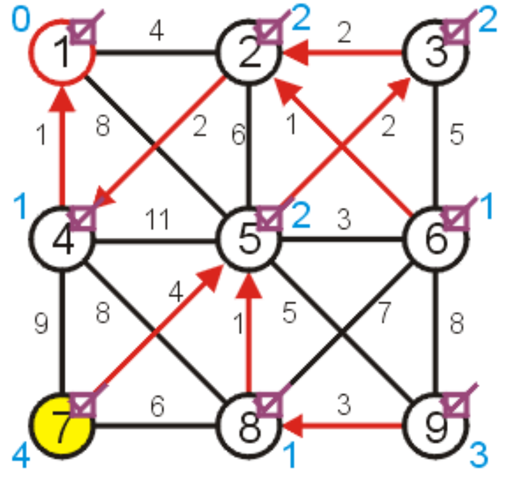


		Distance	Parent
1	T	0	0
2	T	2	4
3	T	2	2
4	T	1	1
5	T	2	3
6	T	1	2
7	F	4	5
8	T	1	5
9	T	3	8

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

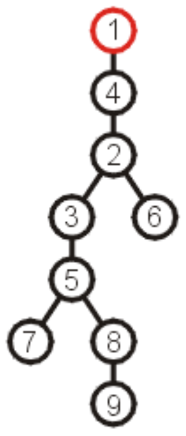
At this point, there are no more unvisited vertices, and therefore we are done

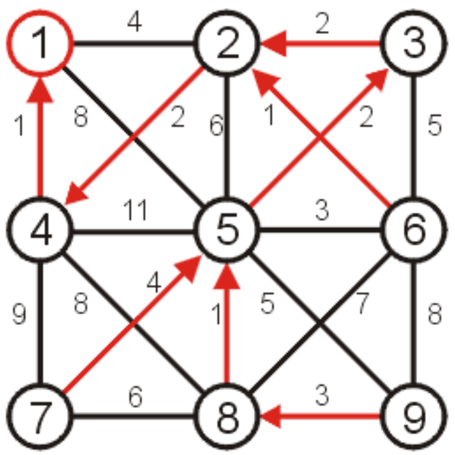
If at any point, all remaining vertices had a distance of ∞, this would indicate that the graph is not connected

- 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

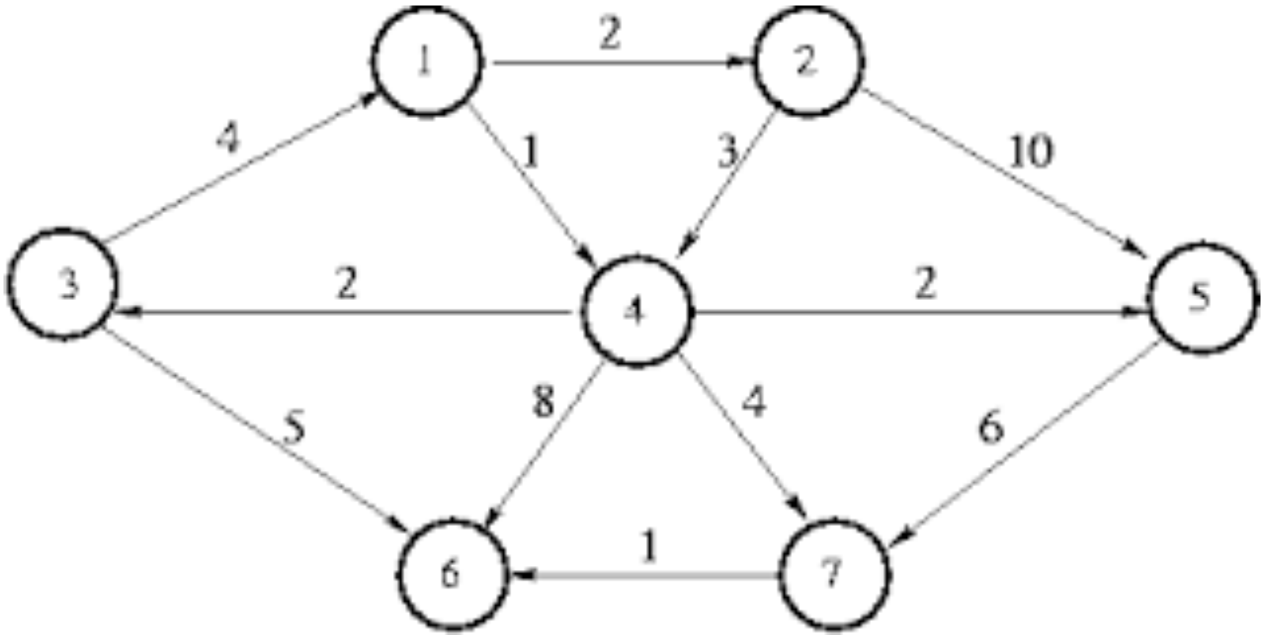
Prim’s Algorithm

To summarize:

- we begin with a vertex which represents the root
- starting with this trivial tree and iteration, we find the shortest edge which we can add to this already existing tree to expand it

This is a reasonably efficient algorithm: the number of visits to vertices is kept to a minimum

Exercise – identify MST



Shortest Path

- Finding a path between X vertices in a graph such that the total sum of the edges weights is minimum.
- This problem could be solved easily using **(BFS)** if all edge weights were (1), but here weights can take any value.

Dijkstra’s algorithm

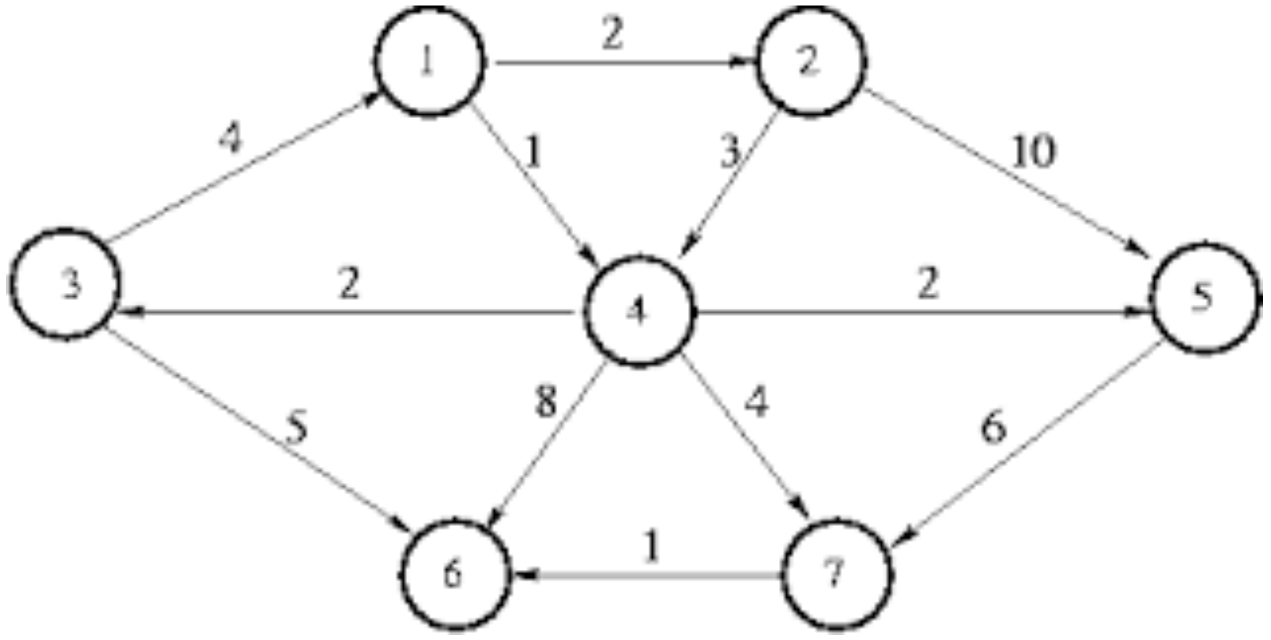
- Set all vertices distances = infinity except for the source vertex, set the source distance = 00.
- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.
- Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without using it.

Apply the same algorithm again until the priority queue is empty.

Dijkstra’s algorithm

- Find that unvisited vertex ***v*** that has a minimum distance to it
- Mark it as having been visited
- Consider every adjacent vertex ***w*** that is unvisited:
    - Is the distance to ***v*** plus the weight of the edge (***v***, ***w***) less than our currently known shortest distance to ***w***
    - If so, update the shortest distance to ***w*** and record ***v*** as the previous pointer
- Continue iterating until all vertices are visited or all remaining vertices have a distance to them of infinity

Exercise – identify shortest paths from node 1 using Dijkstra's algorithm



Exercise

Using your easyJet routes below implement the MST and Dijkstra algorithms

- London to Paris 1
- Paris to London 1
- London to Amsterdam 1
- Amsterdam to London 1
- London to Barcelona 3
- Barcelona to London 3
- London to Edinburgh 2
- Edinburgh to London 2
- London to Rome 4
- Rome to London 4
- Athens to London 4
- Paris to Barcelona 2
- Barcelona to Paris 2
- Paris to Nice 1
- Paris to Venice 3
- Venice to Paris 3
- Rome to Athens 2
- Venice to Rome 1
- Naples to Milan 1
- Athens to Paris 4

Acknowledgement

- I would like to acknowledge the site https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/#graph-algorithms for the slides