Minimum Spanning Tree, Kruskal's and Prim's Algorithms, Applications in Networking

Submitted by:
Hardik Parikh
Soujanya Soni

OverView

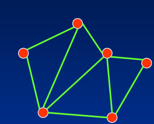
- Tree definition
- Spanning Trees
- Minimum Spanning Trees
- Kruskal's and Prim's algorithms for Minimum Spanning Trees
- Comparison & Analysis of Kruskal's and Prim's Algorithms
- Why MST ????
- Applications of MST in Sensor Networks
- Conclusion

What is Tree ?

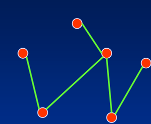
- **Definition:** A **tree** is a connected undirected graph with no simple circuits.
- Since a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops.
- Therefore, any tree must be a **simple graph**.
- **Theorem:** An undirected graph is a tree if and only if there is a **unique simple path** between any of its vertices.

Trees

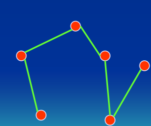
•**Example:** Are the following graphs trees?



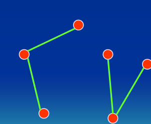
No.



Yes.



Yes.



No.

Spanning Trees

A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.

A graph may have many spanning trees

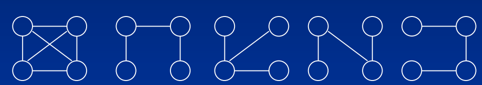
Spanning Tree properties:

On a connected graph G=(V, E), a spanning tree:

- is a connected subgraph
- acyclic
- is a tree (|E| = |V| - 1)
- contains all vertices of G (spanning)

Spanning trees

- Suppose you have a connected undirected graph
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction
- ...then a spanning tree of the graph is a connected subgraph in which there are no cycles
- Total of 16 different spanning trees for the graph below

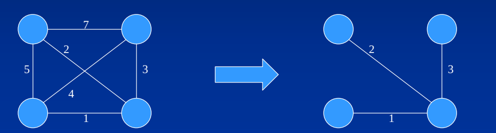


A connected, undirected graph   Four of the spanning trees of the graph

Minimum Spanning Trees

The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.

- Complete Graph   Minimum Spanning Tree



Minimum weight spanning tree (MST)

On a weighted graph, A MST:
- connects all vertices through edges with least weights.
- has w(T) ≤ w(T') for every other spanning tree T' in G
- Adding one edge not in MST will create a cycle

Finding Minimum Spanning Tree

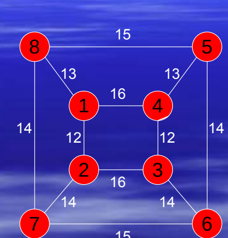
Algorithms :

- Kruskal's and
- Prim's

Kruskal's algorithm

→ Edge first
1. Arrange all edges in a list (L) in non-decreasing order
2. Select edges from L, and include that in set T, avoid cycle.
3. Repeat 3 until T becomes a tree that covers all vertices

Kruskal's Algorithm

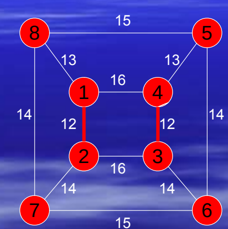


{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm

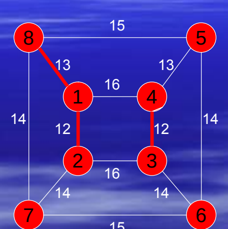
{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm



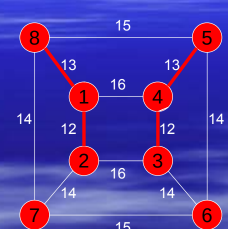
{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm



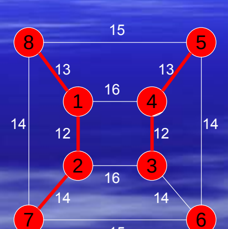
{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm



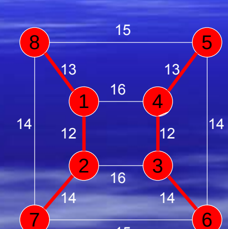
{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm



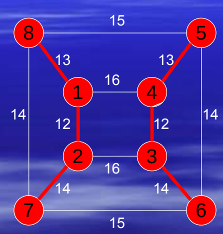
{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Kruskal's Algorithm



{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

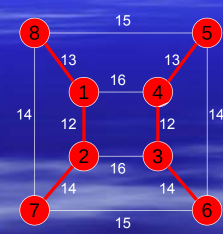
Kruskal's Algorithm



Skip {7,8} to avoid cycle

{1,2}	12	
{3,4}	12	
{1,8}	13	
{4,5}	13	
{2,7}	14	
{3,6}	14	
{7,8}	14	Skip
{5,6}	14	
{5,8}	15	
{6,7}	15	
{1,4}	16	
{2,3}	16	

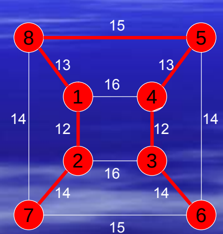
Kruskal's Algorithm



Skip {5,6} to avoid cycle

{1,2}	12	
{3,4}	12	
{1,8}	13	
{4,5}	13	
{2,7}	14	
{3,6}	14	
{7,8}	14	Skip
{5,6}	14	Skip
{5,8}	15	
{6,7}	15	
{1,4}	16	
{2,3}	16	

Kruskal's Algorithm



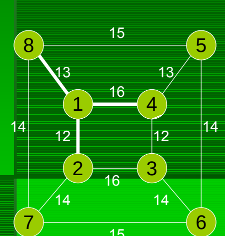
MST is formed

{1,2}	12
{3,4}	12
{1,8}	13
{4,5}	13
{2,7}	14
{3,6}	14
{7,8}	14
{5,6}	14
{5,8}	15
{6,7}	15
{1,4}	16
{2,3}	16

Prim's algorithm

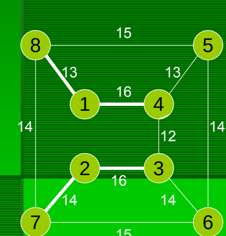
- Start form any arbitrary vertex
- Find the edge that has minimum weight form all known vertices
- Stop when the tree covers all vertices

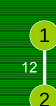
Prim's algorithm



Start from any arbitrary vertex

Prim's algorithm



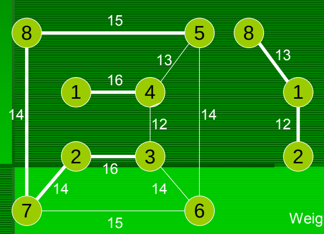


The best choice is {1,2}

What's next?

Weight:12

Prim’s algorithm



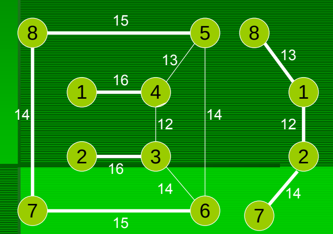
After {1,8} we may choose {2,7} or {7,8}

There are more than one MST

What’s next?

Weight:12+13

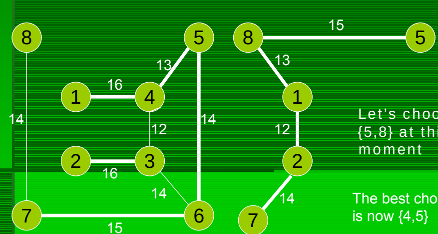
Prim’s algorithm



Let’s choose {2, 7} at this moment

We are free to choose {5,8} or {6,7} but not {7,8} because we need to avoid cycle

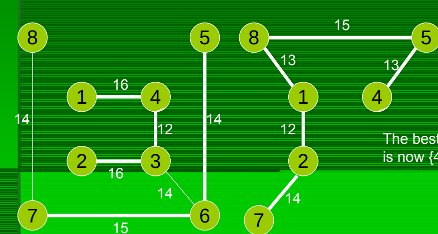
Prim’s algorithm



Let’s choose {5,8} at this moment

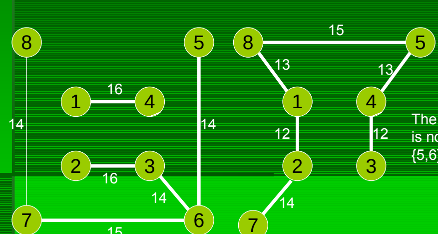
The best choice is now {4,5}

Prim’s algorithm



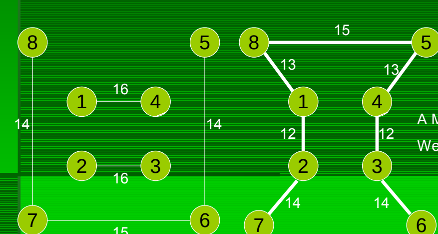
The best choice is now {4,3}

Prim’s algorithm



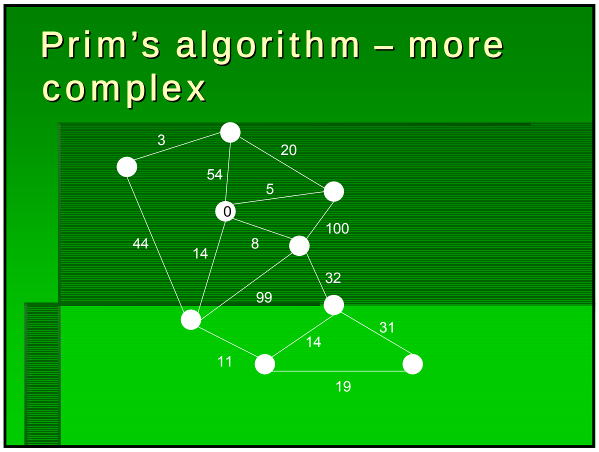
The best choice is now {3,6} or {5,6}

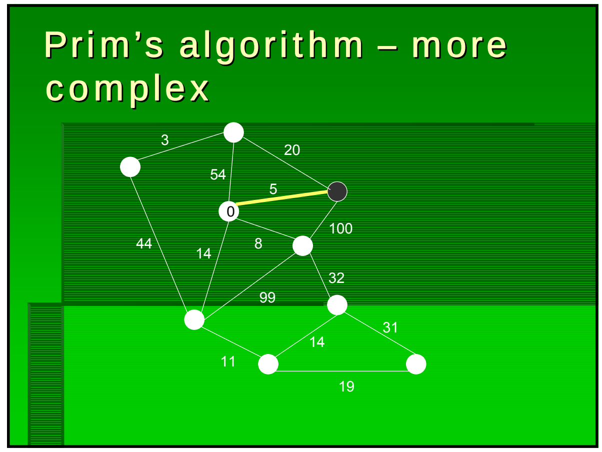
Prim’s algorithm

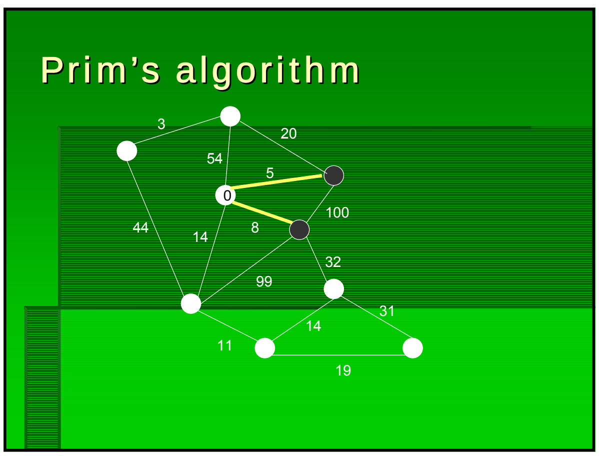


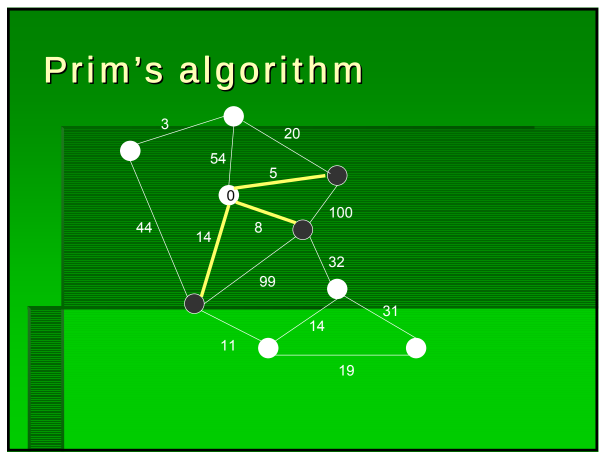
A MST is formed

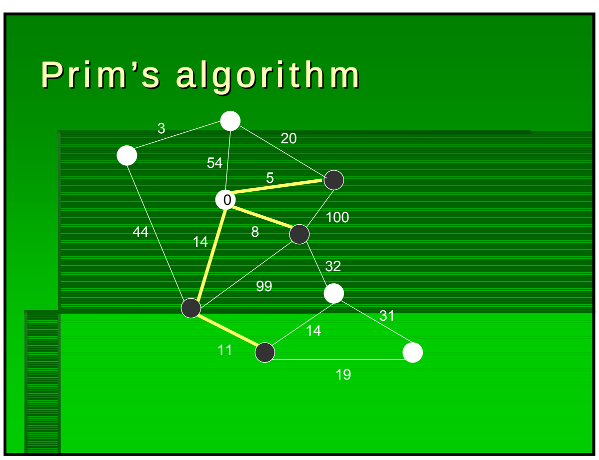
Weight = 93

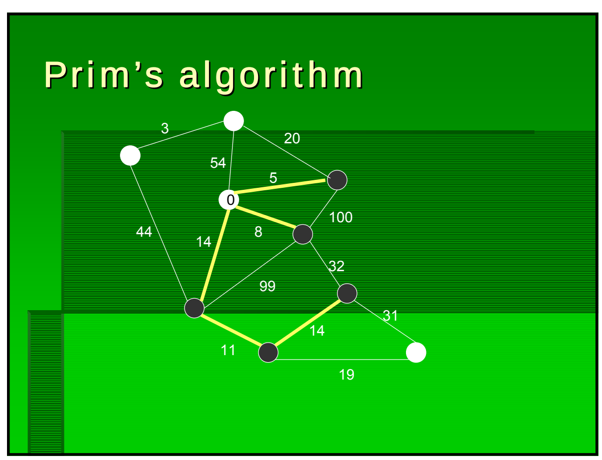

Prim's algorithm – more complex
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19


Prim's algorithm – more complex
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19

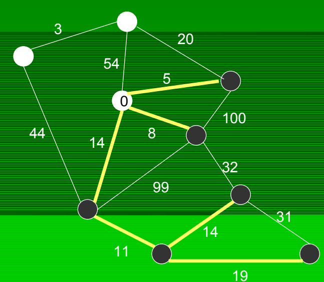

Prim's algorithm
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19


Prim's algorithm
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19

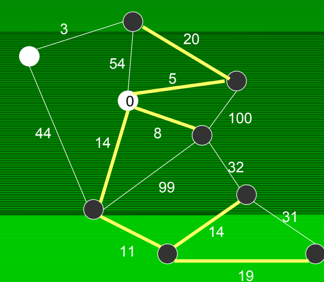

Prim's algorithm
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19


Prim's algorithm
3
20
54
5
0
100
44
14
8
32
99
31
14
11
19

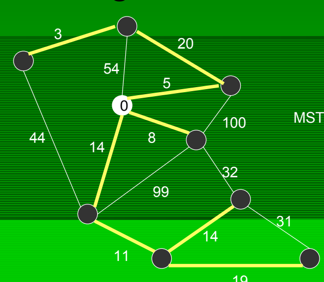
Prim's algorithm



Prim's algorithm



Prim's algorithm



MST is formed

Compare Prim and Kruskal

- Both have the same output → MST
- Kruskal's begins with forest and merge into a tree
- Prim's always stays as a tree
- If you don't know all the weight on edges → use Prim's algorithm
- If you only need partial solution on the graph → use Prim's algorithm

Compare Prim and Kruskal

Complexity

Kruskal: O(NlogN)

comparison sort for edges

Prim: O(NlogN)

search the least weight edge for every vertice

Analysis of Kruskal's Algorithm

Running Time = O(m log n) (m = edges, n = nodes)

Testing if an edge creates a cycle can be slow unless a complicated data structure called a "union-find" structure is used.

It usually only has to check a small fraction of the edges, but in some cases (like if there was a vertex connected to the graph by only one edge and it was the longest edge) it would have to check all the edges.

This algorithm works best, of course, if the number of edges is kept to a minimum.

Analysis of Prim's Algorithm

Running Time = O(m + n log n) (m = edges, n = nodes)

If a heap is not used, the run time will be O(n^2) instead of O(m + n log n). However, using a heap complicates the code since you're complicating the data structure. A Fibonacci heap is the best kind of heap to use, but again, it complicates the code.

Unlike Kruskal's, it doesn't need to see all of the graph at once. It can deal with it one piece at a time. It also doesn't need to worry if adding an edge will create a cycle since this algorithm deals primarily with the nodes, and not the edges.

For this algorithm the number of nodes needs to be kept to a minimum in addition to the number of edges. For small graphs, the edges matter more, while for large graphs the number of nodes matters more.

Why do we need MST?

- a reasonable way for clustering points in space into natural groups
- can be used to give approximate solutions to hard problems

Minimizing costs

- Suppose you want to provide solution to :
  - electric power, Water, telephone lines, network setup
- To minimize cost, you could connect locations using MST
- However, MST is not necessary the shortest path and it does not apply to cycle

MST don't solve TSP

- Travel salesman problem (TSP) can not be solved by MST :

→ salesman needs to go home
(what's the cost going home?)
→ TSP is a cycle
→ Use MST to approximate
→ solve TSP by exhaustive approach
try every permutation on cyclic graph

Applications of MST in Sensor Neworks

• Given a set of sensor nodes and data sink in a plane, the transmission power of each sensor node is adjusted in such a manner (using minimum spanning trees in terms of Euclidian distances), that a tree is formed from all sensor nodes to the sink and total transmission power of all sensor nodes is minimum.

Conclusion

Kruskal's has better running times if the number of edges is low, while Prim's has a better running time if both the number of edges and the number of nodes are low.

So, of course, the best algorithm depends on the graph and if you want to bear the cost of complex data structures.

Questions

?

The End

Thank you