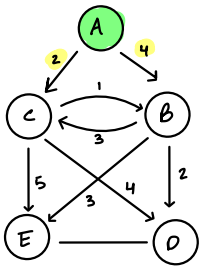
Dijkstra's algorithm

https://youtu.be/_lHSawdgXpI?si=vZEPLKjfbLzii0vR



Unvisited nodes = A, B, C, D, E

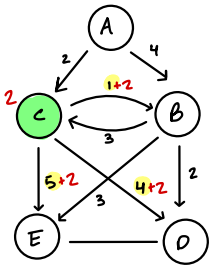
1.) Pick starting node: A

2.) give starting node a table of distances to edges

A) A: 0
B: 4
C: 2
D: ∞
E: ∞

3.) determind smallest value & cross off "A" from unvisited nodes

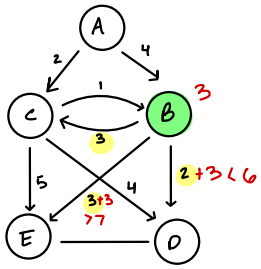
A) A: 0 Unvisited nodes = ~~A~~, B, C, D, E
B: 4
C: 2 ← smallest distance
D: null
E: null



4.) Make smallest node the current node & Repeat 2→3 adding the distances to the table until all nodes are visited

Unvisited nodes = ~~A~~, B, ~~C~~, D, E

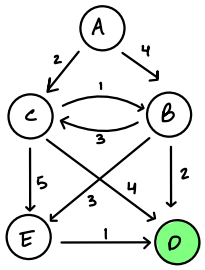
C) A: 0
B: 3 ← smallest unvisited
C: 2
D: 6
E: 7



Unvisited nodes = ~~A~~, ~~B~~, ~~C~~, D, E

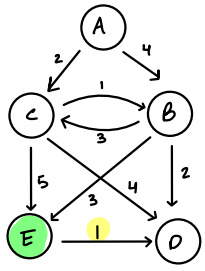
B) A: 0
B: 3 ← smallest unvisited
C: 2
D: 6
E: 7

B) A: 0
B: 3
C: 2
D: 5 ← smallest unvisited
E: 6



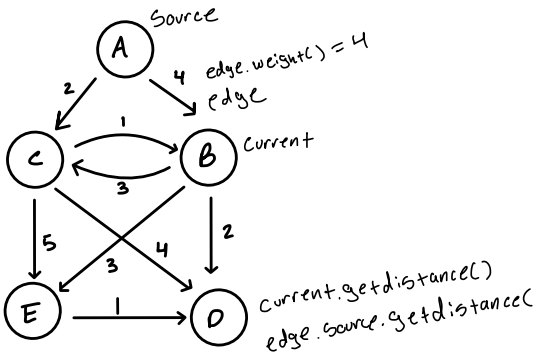
D) A: 0 no leaving path so table is unchanged
B: 3
C: 2
D: 5
E: 6 ← smallest unvisited

Unvisited nodes = ~~A~~, ~~B~~, ~~C~~, ~~D~~, E



E) A: 0
B: 3
C: 2
D: 5
E: 6

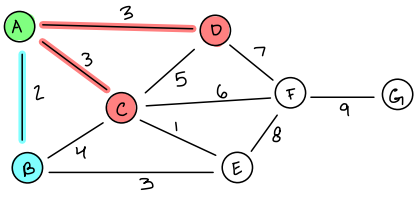
Unvisited nodes = ~~A~~, ~~B~~, ~~C~~, ~~D~~, ~~E~~



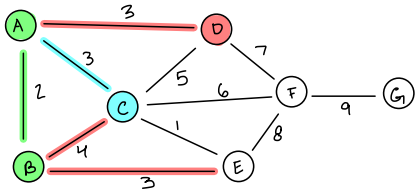
1.) Mark all Vertex as unvisited
2.) Mark Source Vertex as 0 and all other as inf
3.) Consider source Vertex as current Vertex
4.) Calculate Path length of neighors by adding weight of edge in current Vertex
5.) If new Path < Previos Path then Replace it
6.) Mark Current Vertex as visited after visiting neihbar vertex
7.) Select Vertex With Smallest Path as new current
8.) Repeat from Step 4 until unvisited list is empty

Minimum Spanning tree (Prims)

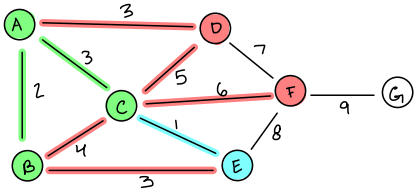
https://youtu.be/cplfcGZmX7I?si=9h6L0myq7IN3DA6Z



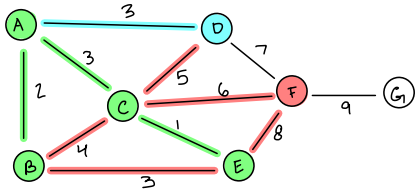
Visited (A)



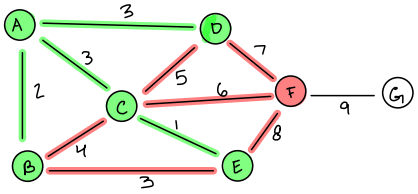
Visited (A,B)



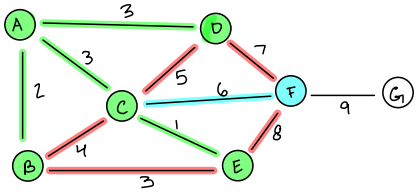
Visited (A,B,C)



Visited (A,B,C,E)

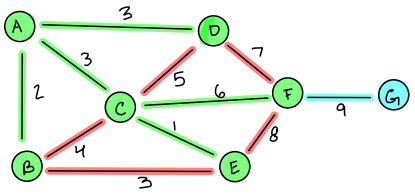


Visited (A,B,C,E,D)

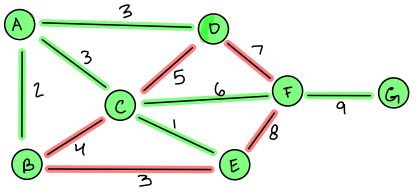


Visited (A,B,C,E,D)

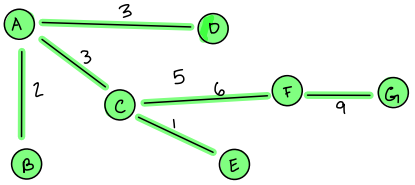
- Start by Making a Visited list to keep track of touched nodes
- Pick a node not in the Visited list
- add node to visited list
- examine all nodes Reachable from current node
- choose the smallest edge that connects to a unvisited node
- Make that edge → destination your current node and add it to visited list
- Repeat 2→6 until all nodes have been visited



Visited (A,B,C,E,D,F)

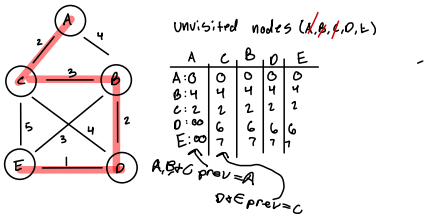


Visited (A,B,C,E,D,F,G)



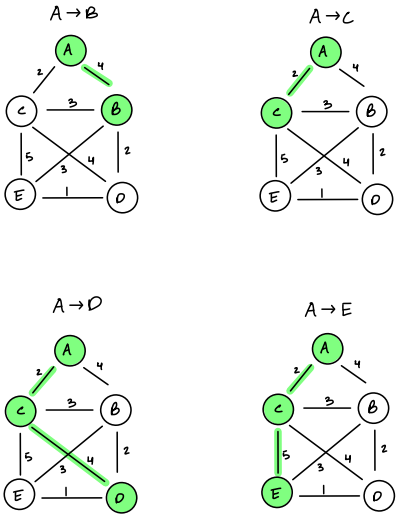
graph 1

Minimum Spanning tree test 1



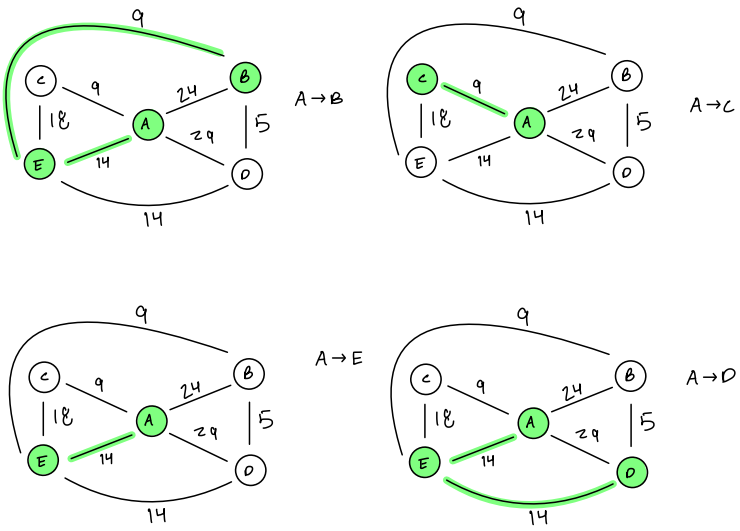

```
////////////////////////////
Graph number one

Added 5 nodes A,B,C,D,E
A
B
C
D
E

Distances and paths from A:
A: 0, Path: A
B: 4, Path: A -> B
C: 2, Path: A -> C
D: 6, Path: A -> C -> D
E: 7, Path: A -> C -> E

Minimum Spanning Tree:
A -> C Weight: 2
C -> B Weight: 3
B -> D Weight: 2
D -> E Weight: 1

////////////////////////////
```

Shortest Path Test 1



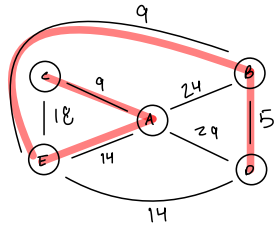
Graph 2

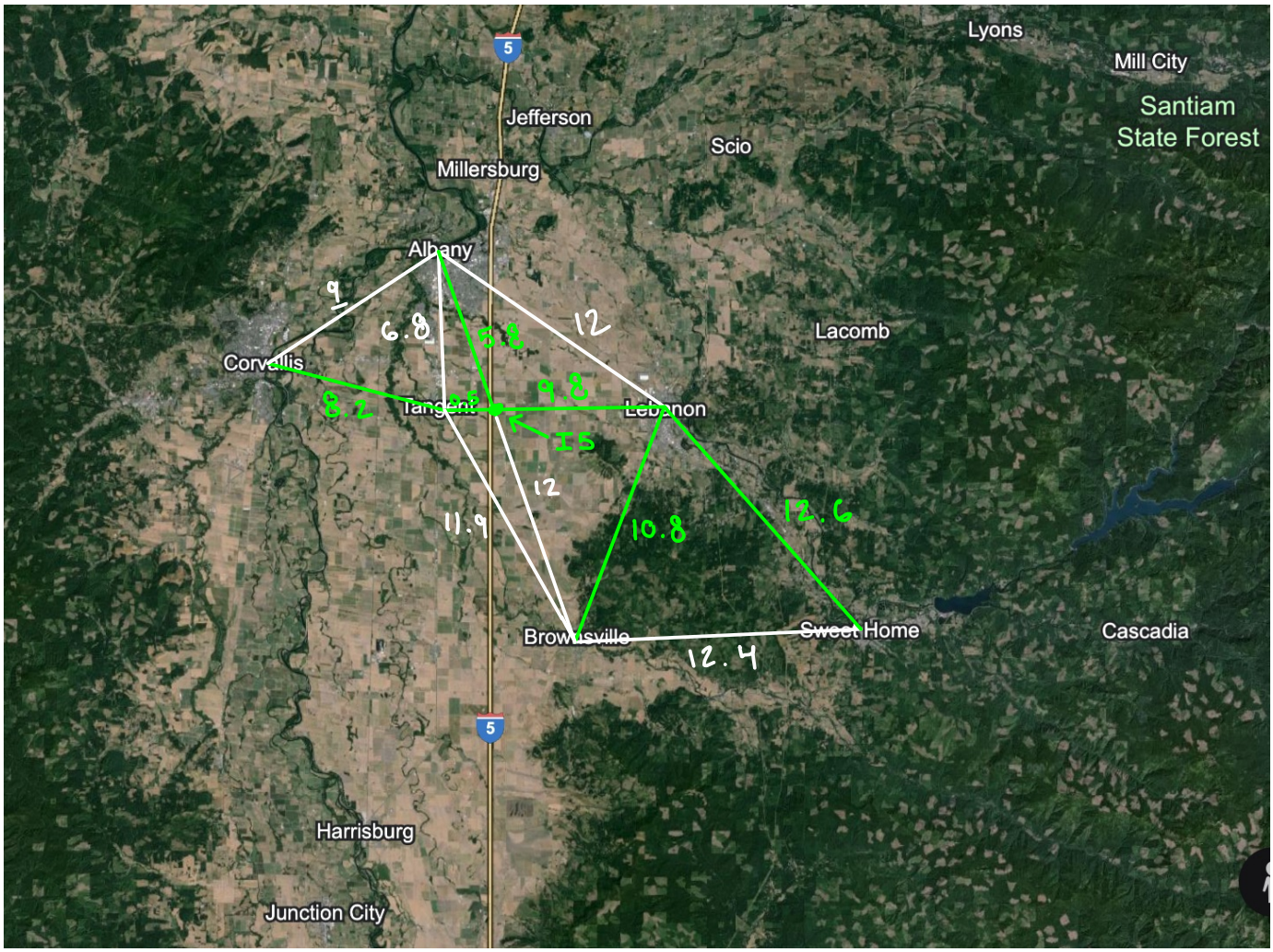
Shortest Path Test 2




```
////////////////////////////
Graph number 2

Added 5 nodes A,B,C,D,E
A
B
C
D
E

Distances and paths from A:
A: 0, Path: A
B: 23, Path: A -> E -> B
C: 9, Path: A -> C
D: 28, Path: A -> E -> D
E: 14, Path: A -> E

Minimum Spanning Tree:
A -> C Weight: 9
A -> E Weight: 14
E -> B Weight: 9
B -> D Weight: 5

////////////////////////////
```

Minimum Spanning tree test 2






```
//////////////////////////////////////////////////
Graph Oregon

Added 7 nodes Corvallis,Tangent,Brownsville,Sweet Home,Albany,Lebanon,I-5
Corvallis
Tangent
Brownsville
Sweet Home
Albany
Lebanon
I-5

Distances and paths from Sweet Home:
Corvallis: 29, Path: Sweet Home -> Lebanon -> I-5 -> Tangent -> Corvallis
Tangent: 21, Path: Sweet Home -> Lebanon -> I-5 -> Tangent
Brownsville: 12, Path: Sweet Home -> Brownsville
Sweet Home: 0, Path: Sweet Home
Albany: 24, Path: Sweet Home -> Lebanon -> Albany
Lebanon: 12, Path: Sweet Home -> Lebanon
I-5: 21, Path: Sweet Home -> Lebanon -> I-5

Minimum Spanning Tree:
Lebanon -> I-5 Weight: 9
I-5 -> Tangent Weight: 0
I-5 -> Albany Weight: 5
Tangent -> Corvallis Weight: 8
Lebanon -> Brownsville Weight: 10
Lebanon -> Sweet Home Weight: 12

//////////////////////////////////////////////////
```