Graphs

PART II
Weighted Graphs

Outline of this lecture

- ***The (All-Pairs) Shortest Path Problem***

  Dijkstra's Algorithm

  The A* Algorithm

- ***Minimum –Cost Spanning Tree Problem***

  Kruskal, Prim, Sollin, et al, Methods

Greedy Algorithms

The essence of the greedy algorithm approach is that the choice made should be locally optimal, i.e. it has to be the best immediate choice among all feasible choices at that step.

On each step *greedy* grab is made of the best alternative available in the hope that a sequence of locally optimal choices will lead to a (*globally*) optimal solution to the entire problem.

Example ….

Go to a shop. Buy something. Say you have to pay 71 dollars for it. You give a cashier a 100. You want your change back with minimum number of notes. You get your change one note at a time, but never exceeding the change, i.e., 29 dollars.

We get the following:

Step1: Well you takes the biggest note that is at most 29, so you take 20 dollar note.

Step2: You need 9 more dollars. You take the biggest note that is not more than 9, so you take 5 dollar note.

Step3: You take the biggest note less than 4. So you take 2 dollar note.

Step4: You take the biggest note that is not more than 2. So you take 2 dollar note.

See what we did. At every step we took the best possible choice that does not violate the solution. This is a greedy algorithm. Greedy since at every step you take the best at the current moment; without thinking what will happen as a consequence.

Will this always give you back your change with the minimum number of notes?

Well yes, but only because we made it that way, that is since we use 1,2,5,20 system for notes.

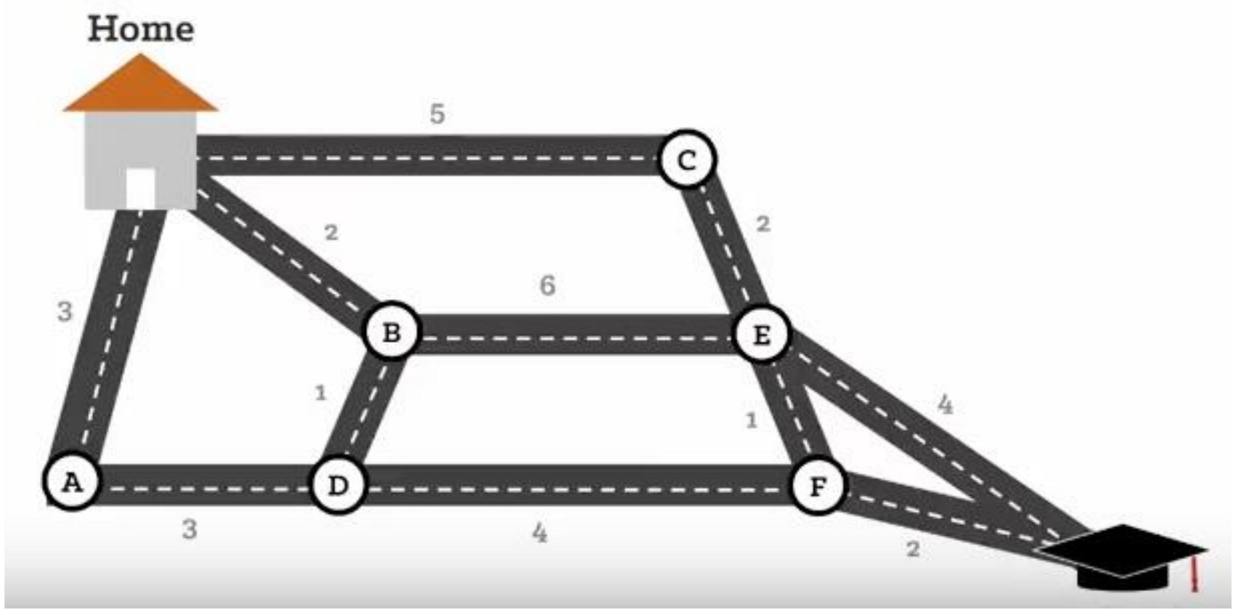
But suppose that all possible notes are 25, 14 and 1. Then if we take our change greedily we take 25, 1, 1, 1, 1. In total 5 notes. We could use less notes if we would take 14,14,1. We were greedy and optimized only depending on the current status. Hence greedy is not always optimal.

Weighted Graphs

SHORTEST PATH PROBLEMS

Shortest Path Problems

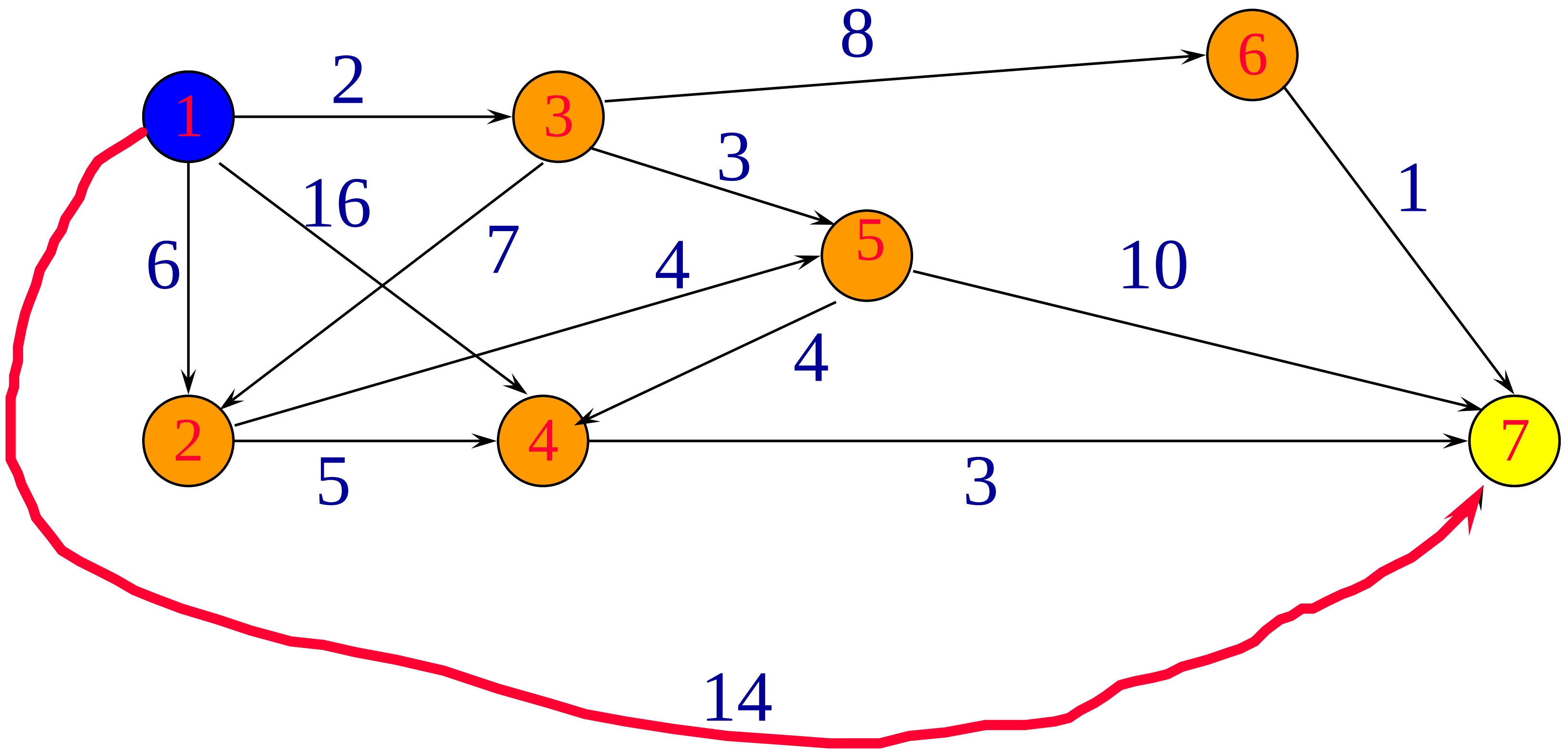
Suppose a student wants to go from home to school in the shortest possible way. She knows some roads are heavily congested and difficult to use.



Shortest Path Problems

- Directed weighted graph.
- Path length is sum of weights of edges on path.
- The vertex at which the path begins is the source vertex.
- The vertex at which the path ends is the destination vertex.

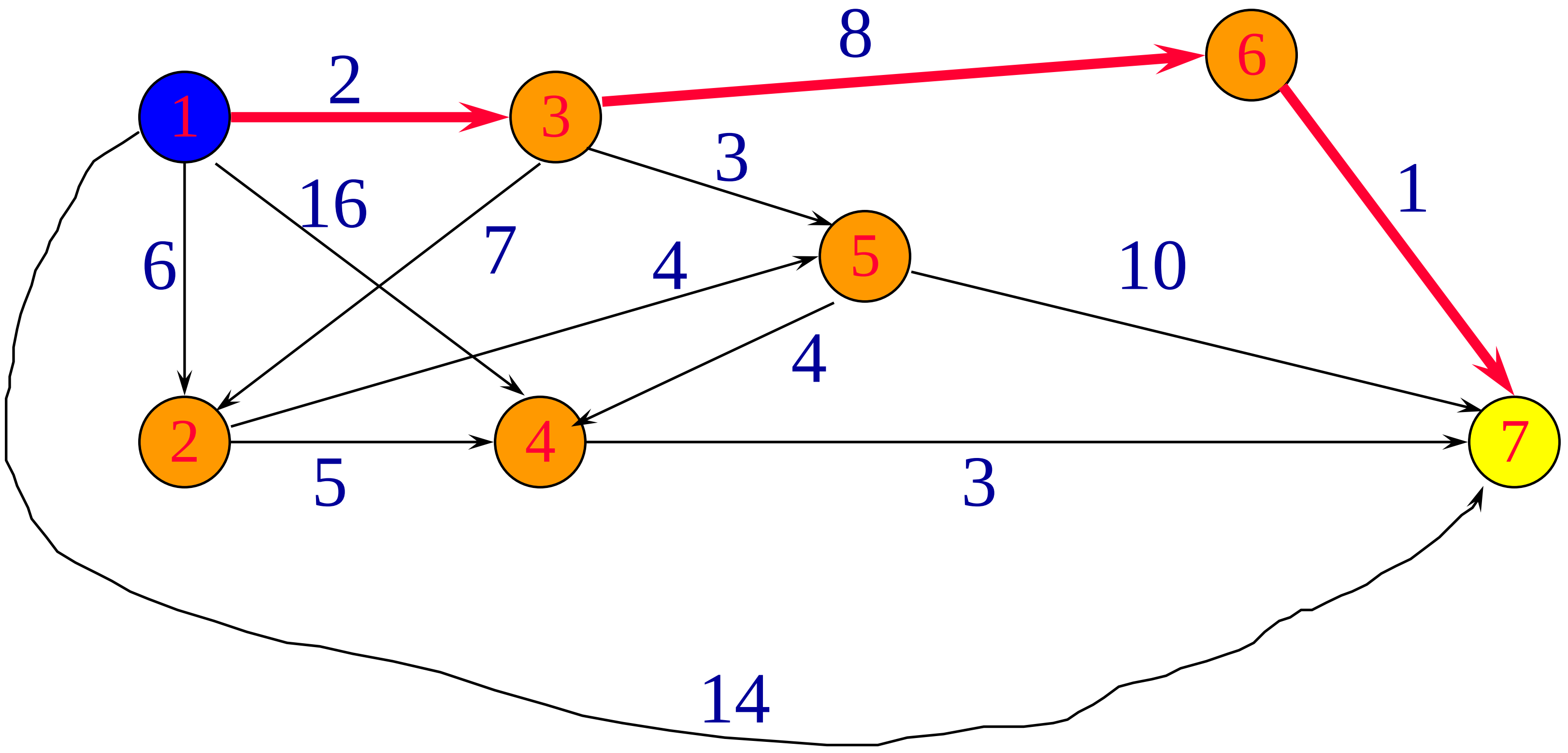
Example



A path from 1 to 7. Path length is 14.

@Sudharshan Welihinda - 2021

Example



Another path from 1 to 7. Path length is 11.

Shortest-Path Variants

No need to consider different solution or algorithm for each variant (we reduce it into two types)

- Single-source single-destination (1-1): Find the shortest path from source *s* to destination *v*.
- Single-source all-destination(1-Many): Find the shortest path from *s* to each vertex *v*.
- Single-destination shortest-paths (Many-1): Find a shortest path to a given *destination* vertex *t* from each vertex *v*.
- All-pairs shortest-paths problem (Many-Many): Find a shortest path from *u* to *v* for every pair of vertices *u* and *v*.

Shortest-Path Variants

Single-Source Single-Destination (1-1) - No good solution that beats 1-M variant - Thus, this problem is mapped to the 1-M variant	**Single-Source All-Destination (1-M)** - Need to be solved (several algorithms) - We will study this one
All-Sources Single-Destination (M-1) - Reverse all edges in the graph - Thus, it is mapped to the (1-M) variant	**All-Sources All-Destinations (M-M)** - Need to be solved (several algorithms) - We will study it (if time permits)

Shortest-Path Variants

Single-Source Single-Destination (1-1)

- No good solution that beats 1-M variant
- Thus, this problem is mapped to the 1-M variant

Single-Source All-Destination (1-M)

- Need to be solved (several algorithms)
- We will study this one

All-Sources Single-Destination (M-1)

- Reverse all edges in the graph
- Thus, it is mapped to the (1-M) variant

All-Sources All-Destinations (M-M)

- Need to be solved (several algorithms)
- We will study it (if time permits)

Shortest Path Problems

One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra's algorithm – aka The greedy single source all destinations algorithm

The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.

Applications of Dijkstra’s shortest path algorithm

- **Digital Mapping Services in Google Maps:**

  Dijkstra’s Algorithm is used to find the minimum distance between two locations along the path. Consider Sri Lanka as a graph and represent a city/place with a vertex and the route between two cities/places as an edge, then by using this algorithm, the shortest routes between any two cities/places or from one city/place to another city/place can be calculated.

- **Flighting Agenda**

  If an agent wants to determine the earliest arrival time for the destination given an origin airport and start time. There this algorithm comes into use.

- **Robotoc Path**

  The robot/drone moves in the ordered direction by following the shortest path to keep delivering the package in a minimum amount of time.

Dijktra's Algorithm

- Assign to every node a tentative distance value : set it zero to our initial node and infinity to all the other nodes.

- Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.

- For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

- When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

Dijktra’s Algorithm ….

- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

- Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

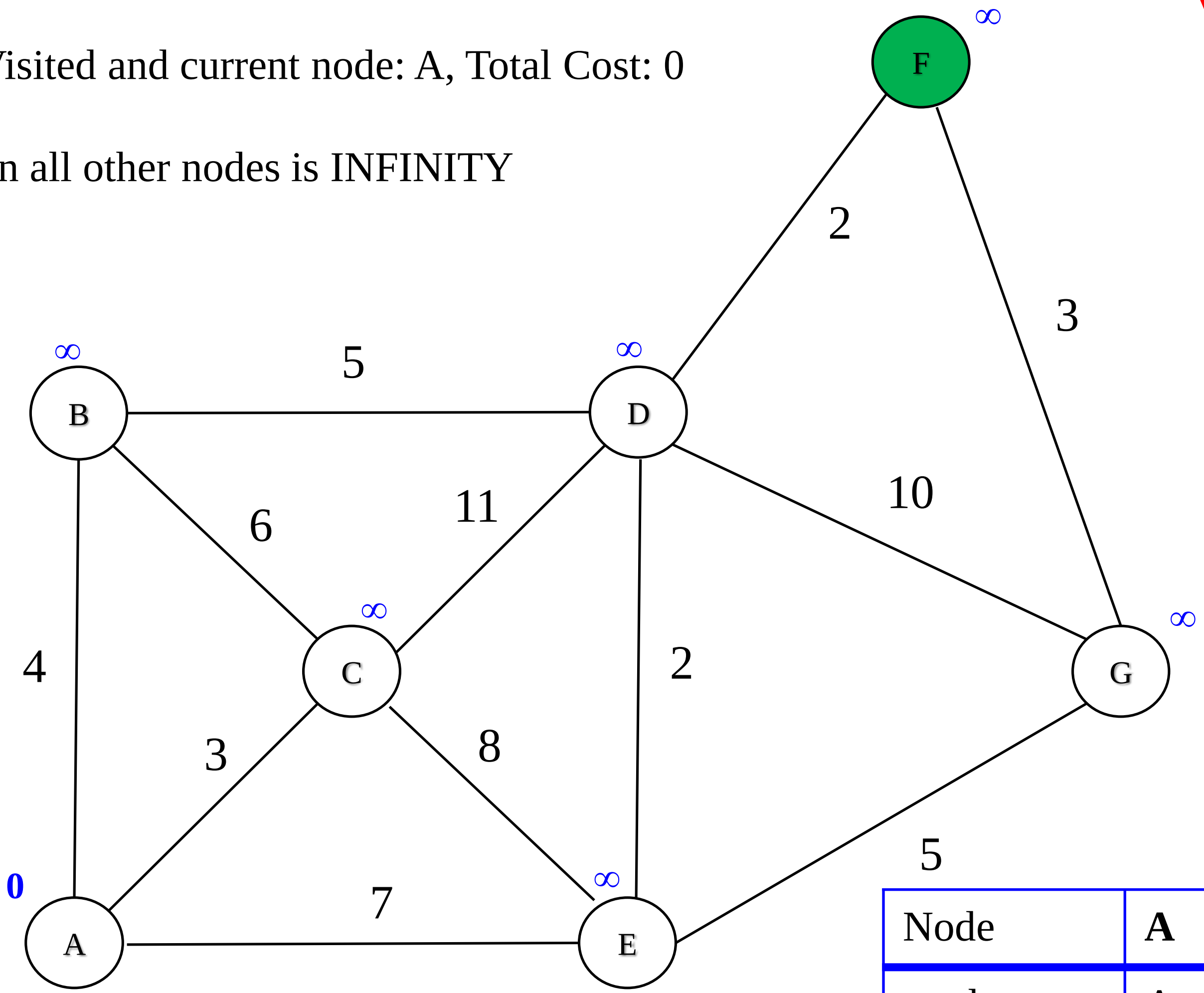
Dijktra's Algorithm

Start condition:

Visited and current node: A, Total Cost: 0

on all other nodes is INFINITY

Visited :{}



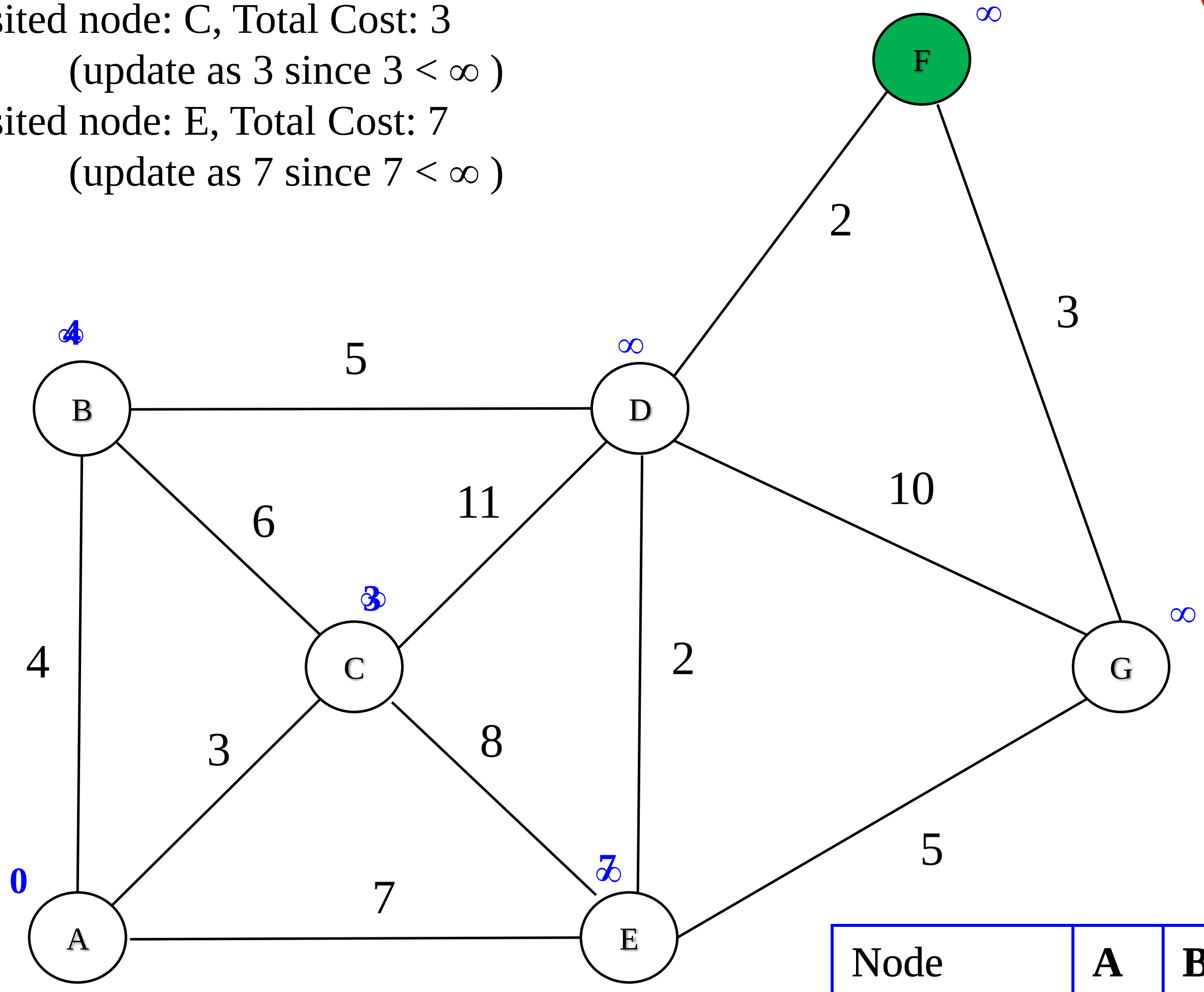
Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
predecessor	A						

@Sudharshan Welihinda - 2021

Dijktra's Algorithm

Visited node: B, Total Cost: 4
(update as 4 since 4 < ∞)
Visited node: C, Total Cost: 3
(update as 3 since 3 < ∞)
Visited node: E, Total Cost: 7
(update as 7 since 7 < ∞)

Visited :{A}



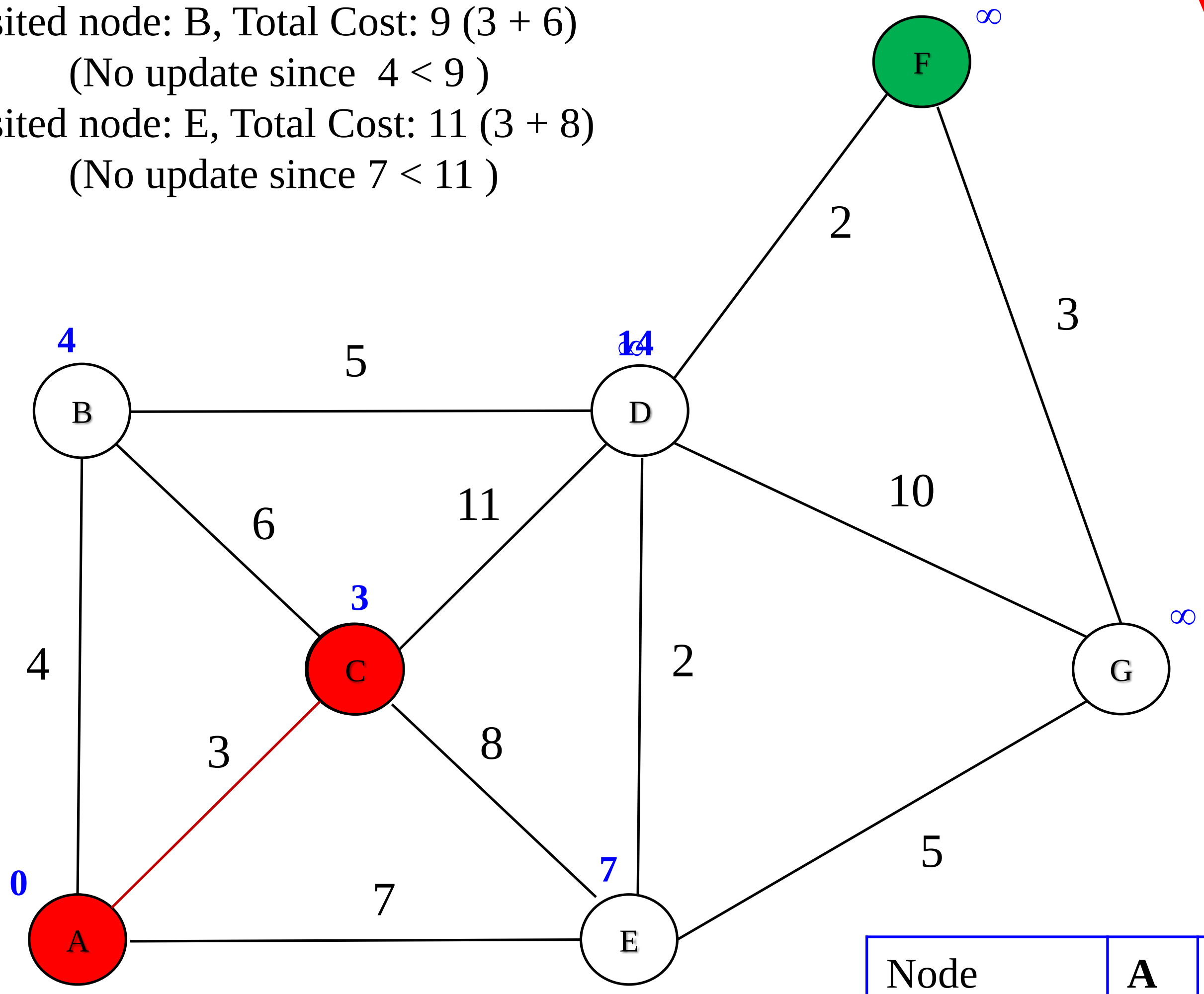
Node	A	B	C	D	E	F	G
predecessor	A						

@Sudharshan Welihinda - 2021

Dijktra’s Algorithm

Visited node: D, Total Cost: 14 (3 + 11)
(update as 14 since 14 < ∞)
Visited node: B, Total Cost: 9 (3 + 6)
(No update since 4 < 9)
Visited node: E, Total Cost: 11 (3 + 8)
(No update since 7 < 11)

Visited :{A,C}



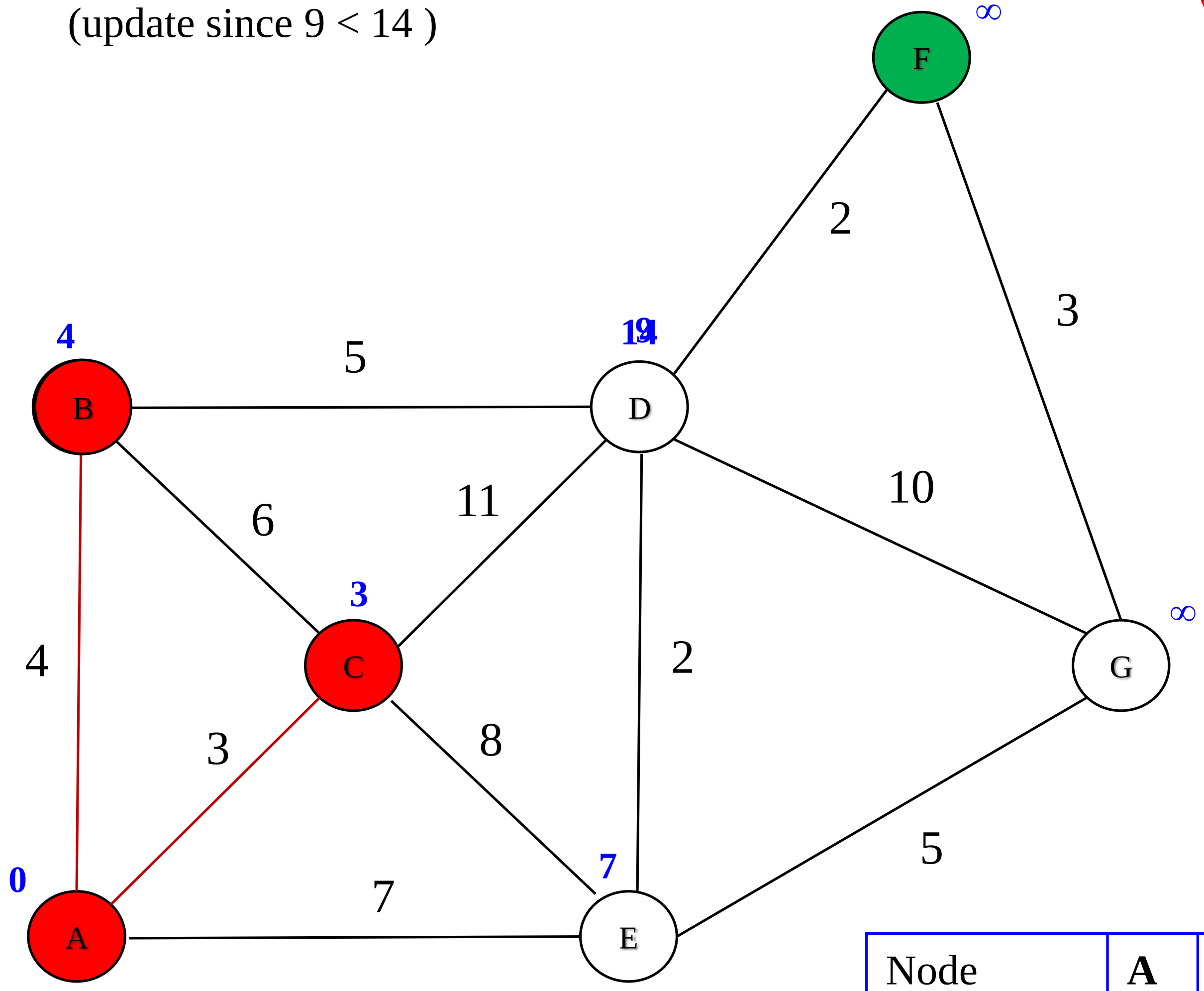
Node	A	B	C	D	E	F	G
predecessor	A	A	A		A		

@Sudharshan Welihinda - 2021

Dijktra's Algorithm

Visited node: D, Total Cost: 9 (4 + 5)
(update since 9 < 14)

Visited :{A,C,B}



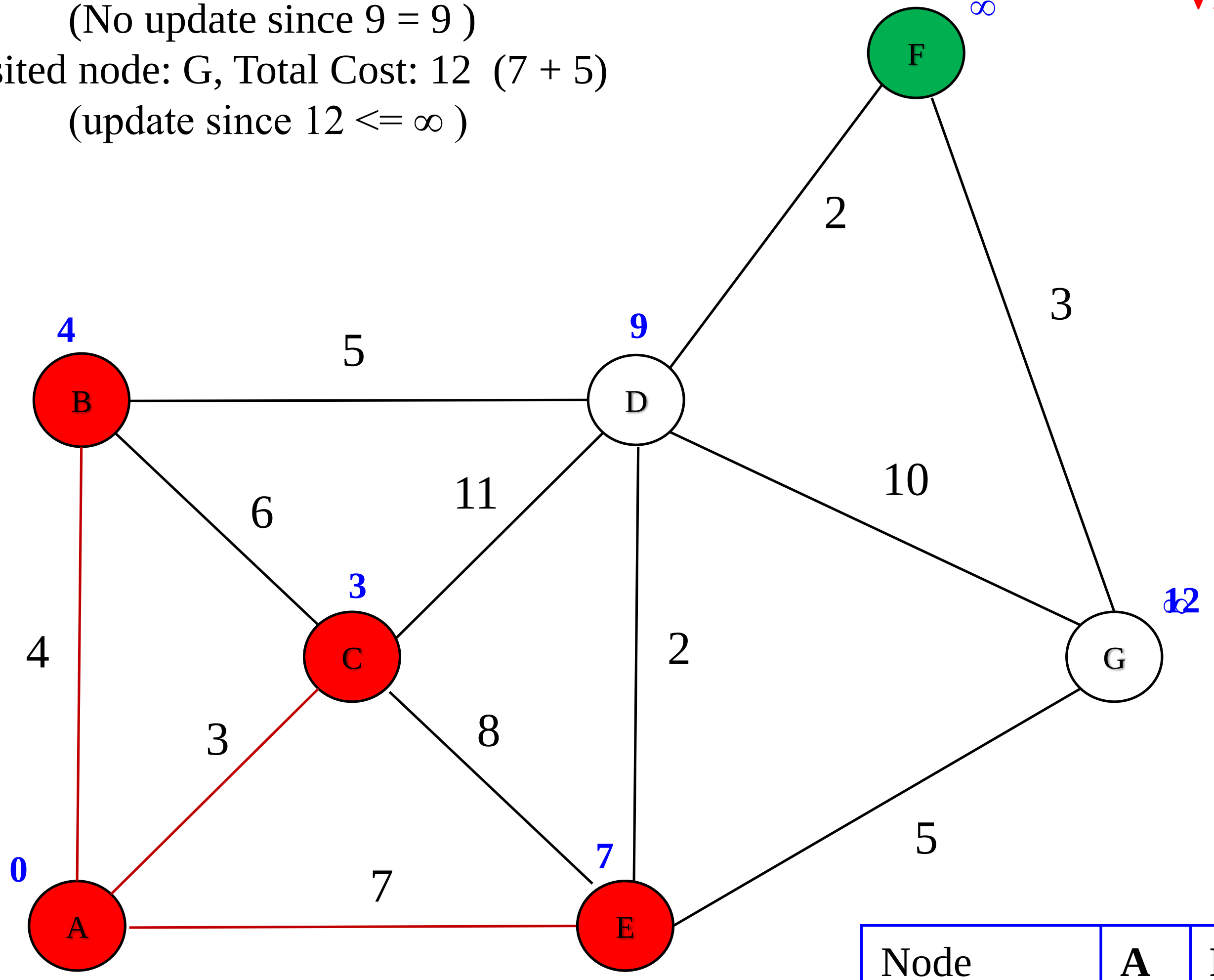
Node	A	B	C	D	E	F	G
predecessor	A	A	A		A		

@Sudharshan Welihinda - 2021

Dijktra's Algorithm

Visited node: D, Total Cost: 9 (7 + 2)
(No update since 9 = 9)
Visited node: G, Total Cost: 12 (7 + 5)
(update since 12 <= ∞)

Visited :{A,C,B,E}



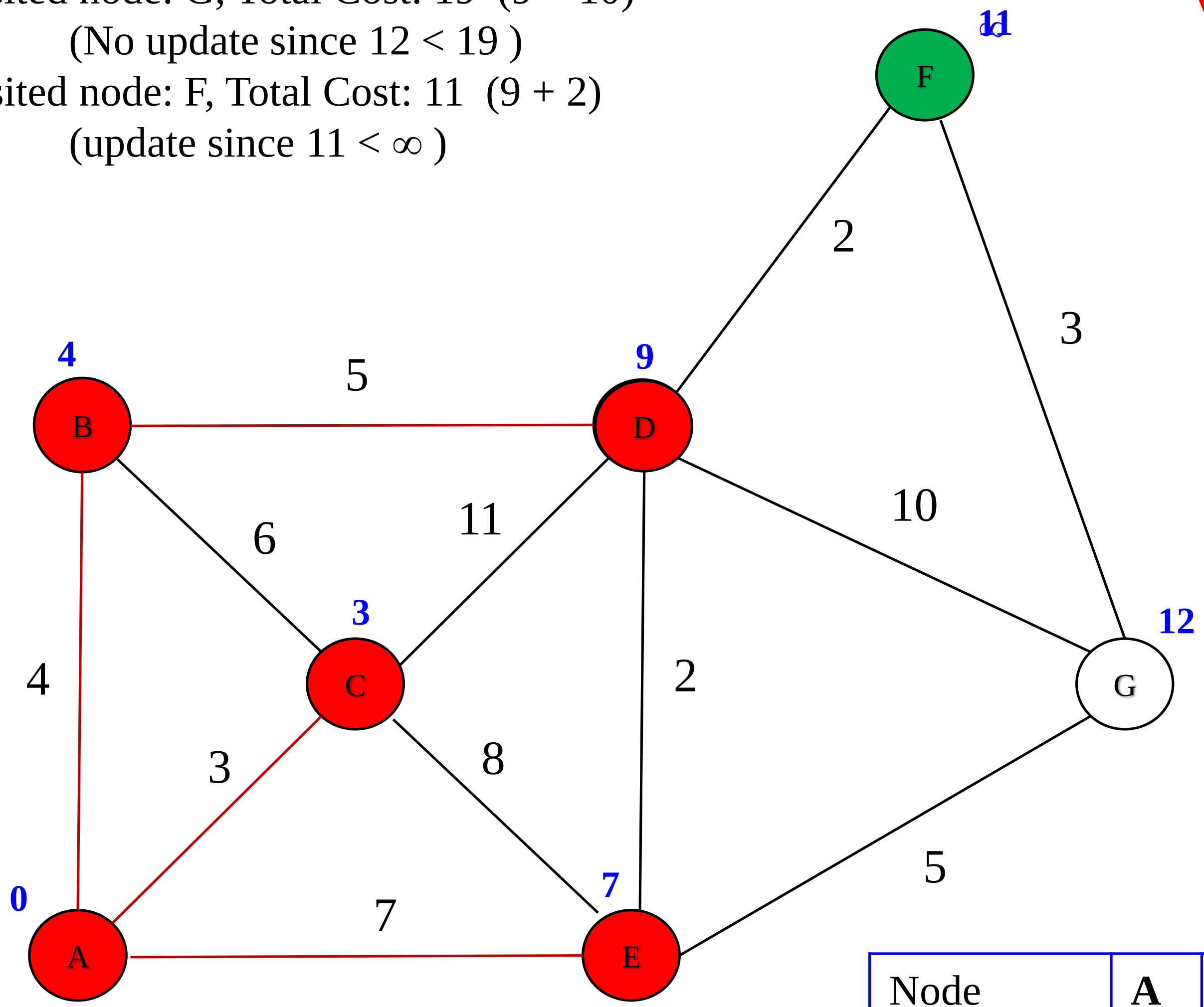
Node	A	B	C	D	E	F	G
predecessor	A	A	A		A		

@Sudharshan Welihinda - 2021

Dijktra’s Algorithm

Visited node: G, Total Cost: 19 (9 + 10)
(No update since 12 < 19)
Visited node: F, Total Cost: 11 (9 + 2)
(update since 11 < ∞)

Visited :{A,C,B,E,D}

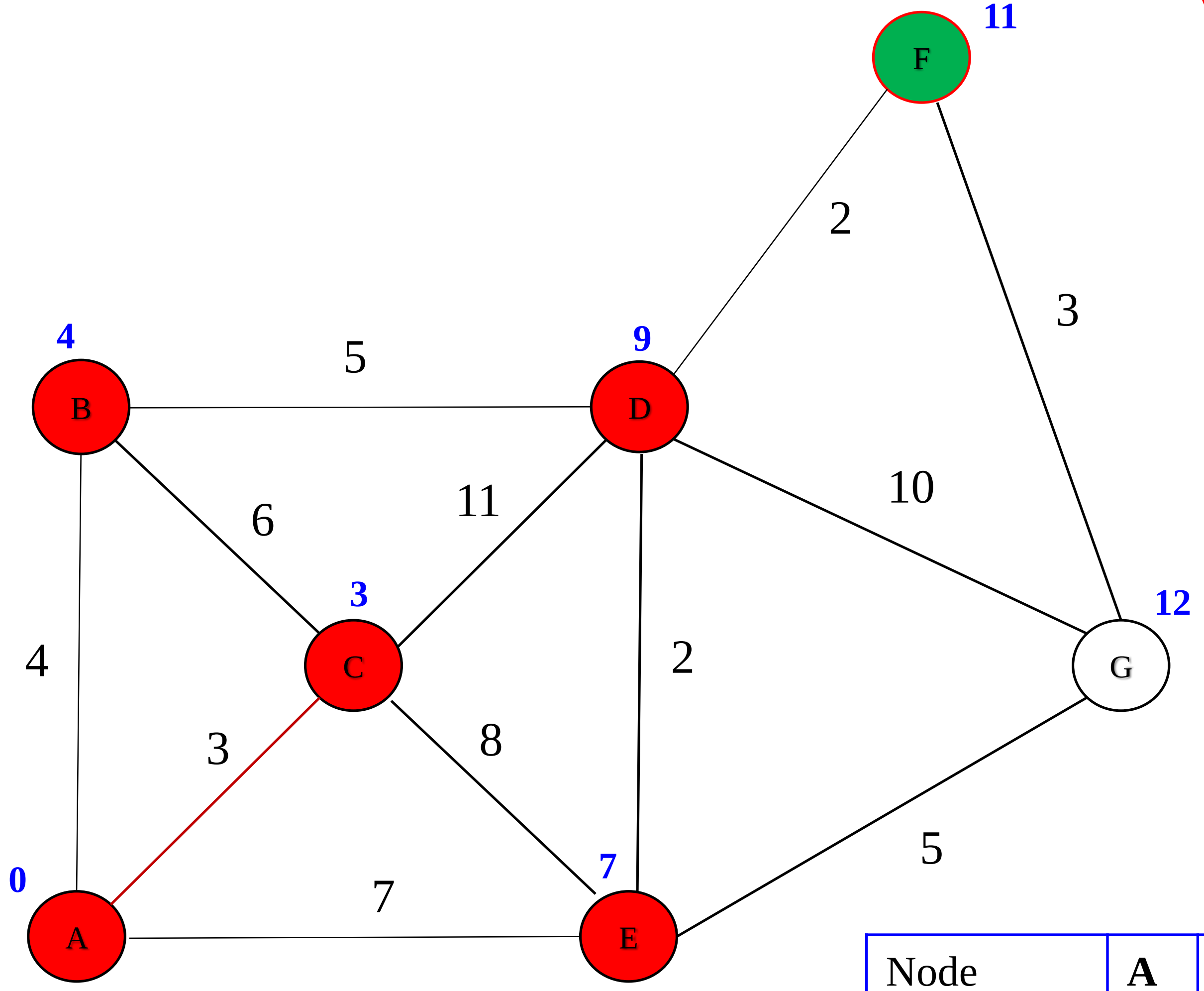


Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
predecessor	A	A	A	B	A		

@Sudharshan Welihinda - 2021

Dijktra’s Algorithm

Visited :{A,C,B,E,D,F}



Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
predecessor	A	A	A	B	A	D	E

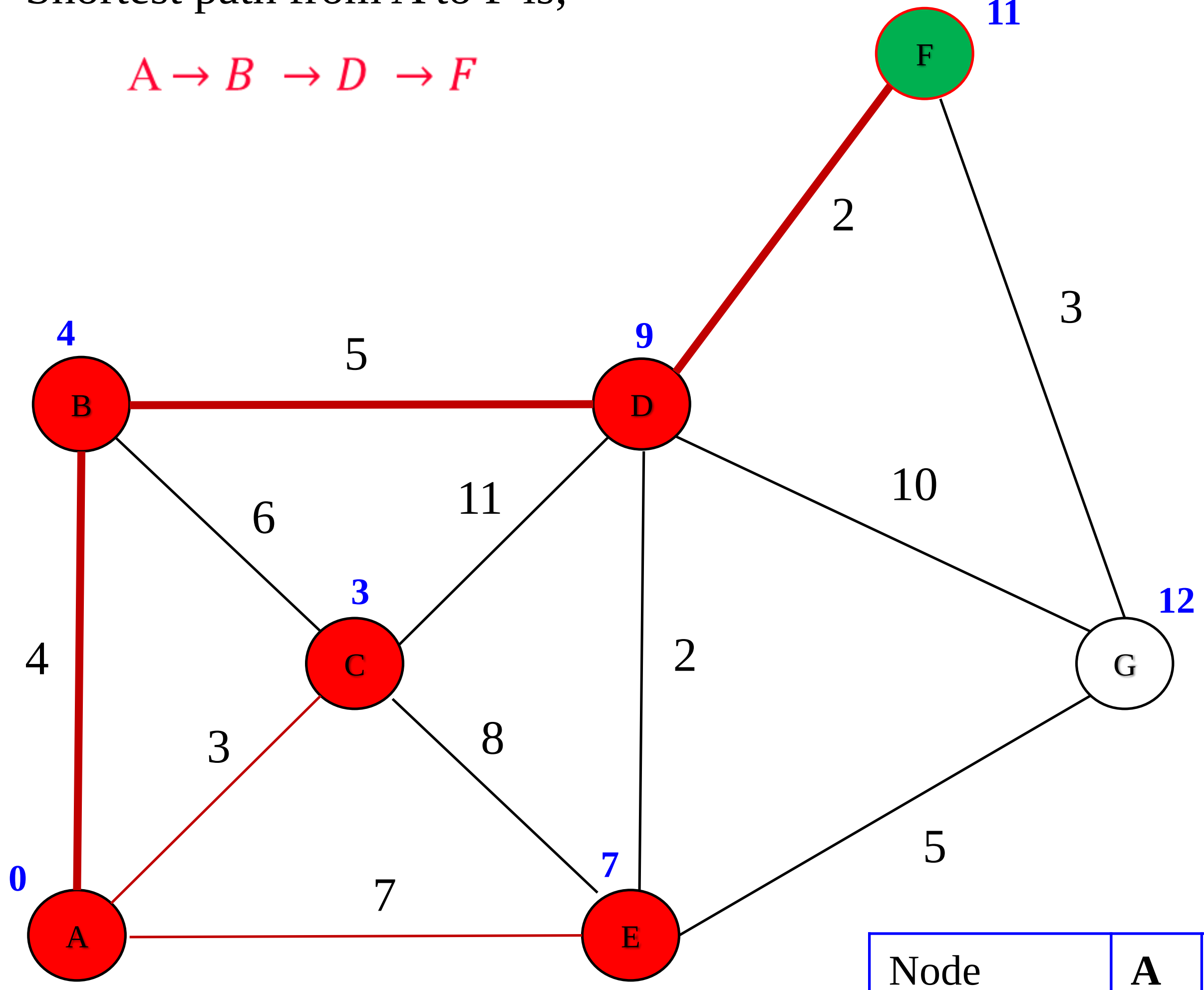
@Sudharshan Welihinda - 2021

Dijktra’s Algorithm

Shortest path from A to F is;

$A \rightarrow B \rightarrow D \rightarrow F$

Visited :{A,C,B,E,D,F}

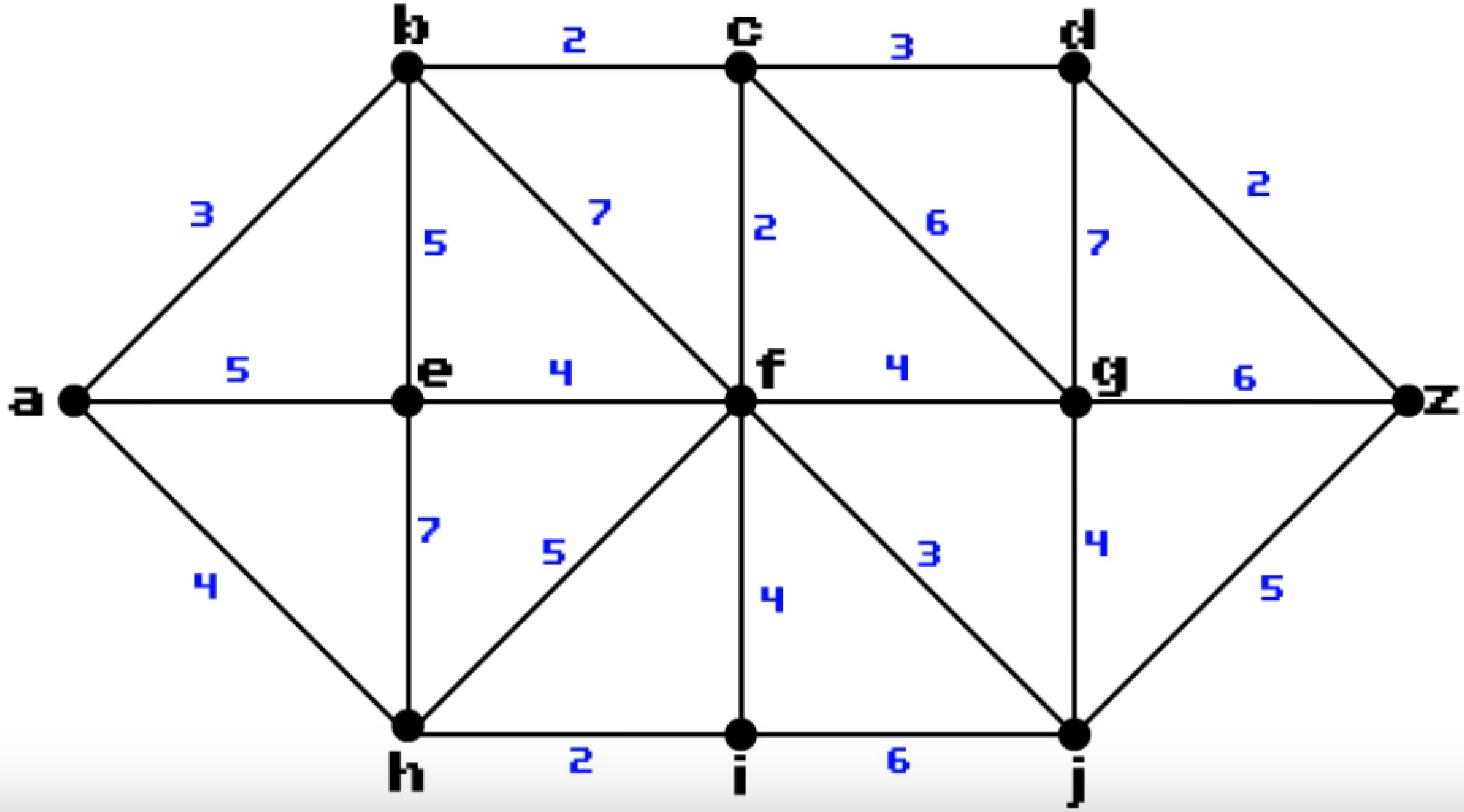


Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**
predecessor	A	A	A	B	A	D	E

@Sudharshan Welihinda - 2021

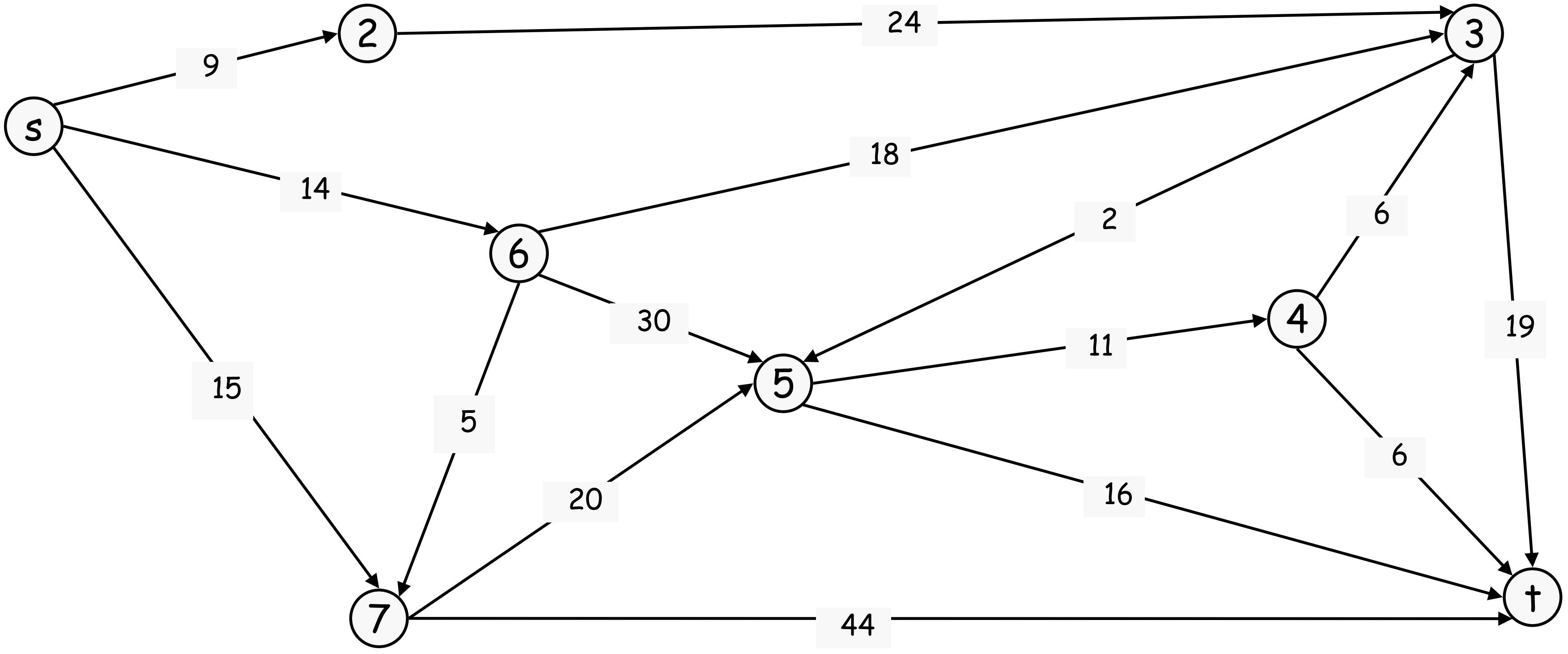
Algorithm example

Set distance zero for the starting vertex.



4:00

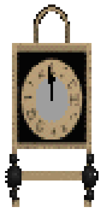
- Find shortest path from s to t.



Data Structures For Previous Algorithm

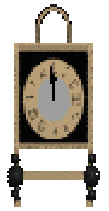
- The greedy single source all destinations algorithm – aka *Dijkstra's algorithm*.
- Implement d() and p() as 1D arrays.
- Keep a linear list L of reachable vertices to which shortest path is yet to be generated.
- Select and remove vertex v in L that has smallest d() value.
- Update d() and p() values of vertices adjacent to v.

Complexity



- O(n) to select next destination vertex.
- O(out-degree) to update d() and p() values when adjacency lists are used.
- O(n) to update d() and p() values when adjacency matrix is used.
- Selection and update done once for each vertex to which a shortest path is found.
- Total time is $O(n^2 + e) = O(n^2)$.

Complexity



- When a min heap of d() values is used in place of the linear list L of reachable vertices, total time is $O((n+e) \log n)$, because $O(n)$ remove min operations and $O(e)$ change key (d() value) operations are done.
- When e is $O(n^2)$, using a min heap is worse than using a linear list.
- When a Fibonacci heap is used, the total time is $O(n \log n + e)$.

Minimum Cost Spanning Tree Algorithms

Tree

- Connected graph that has no cycles.
- n vertex connected graph with n-1 edges.

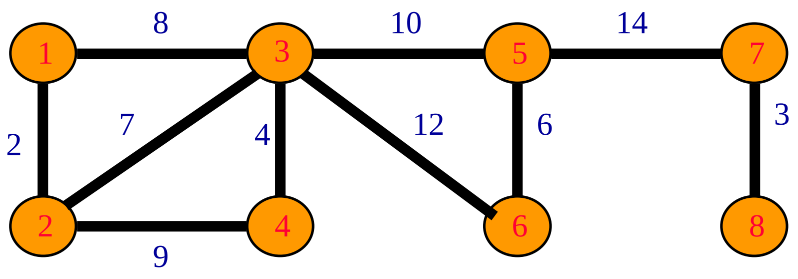
What is a Cycle Graph?

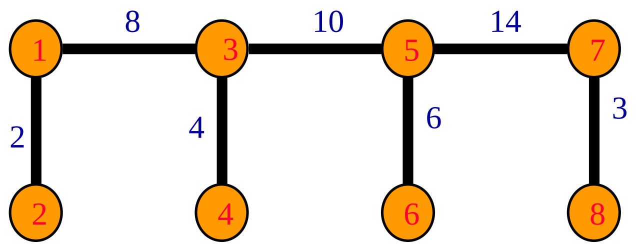
a **cycle graph** or **circular graph** is a graph that consists of a single cycle, or in other words, some number of vertices connected in a closed chain.

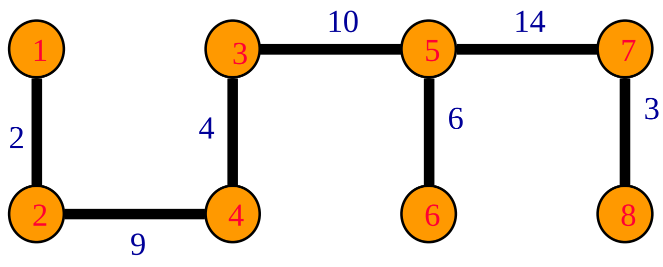
Spanning Tree

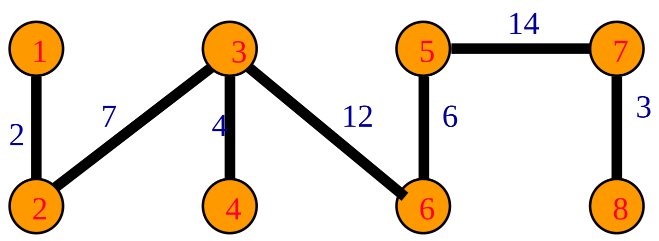
- Subgraph that includes all vertices of the original graph.
- Subgraph is a tree.
    - If original graph has n vertices, the spanning tree has n vertices and n-1 edges.

Spanning Tree









Minimum Cost Spanning Tree

- Consider a ***weighted connected*** graph G.

- A ***minimum weight(cost) spanning tree*** is a ***sub graph*** (a tree) *G'* of *G* such that

    1. a **single path** exists between any two distinct nodes in G' and *V(G') = V(G).*
    2. the **cost** of *G'* (i.e. sum of weights of edges in *G'*) is a ***minimum.***

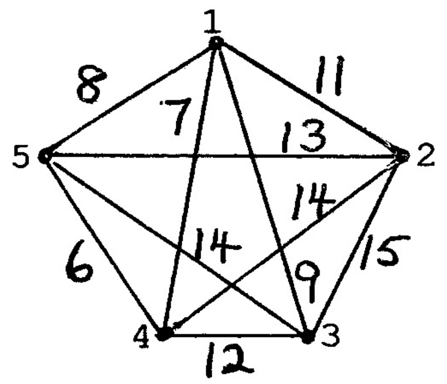
- Condition (1) implies that *G'* is a tree (i.e connected graph that has no cycles).

Problem

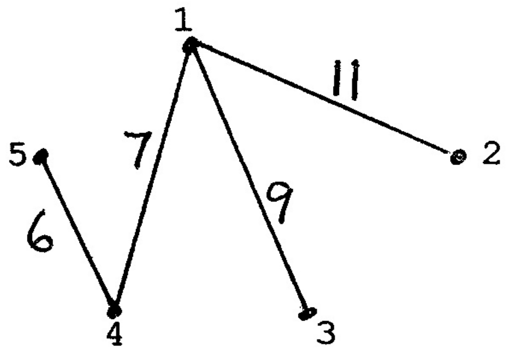
- A company wishes to establish data links between its five main offices, so that each office can communicate (directly or indirectly) with any other office. The company has obtained estimates of the cost of establishing a link between any two offices, as shown below, and would like to implement the network at minimum cost.

- The solution is the ***minimum weight spanning tree***, as shown below

Minimum Cost Spanning Tree



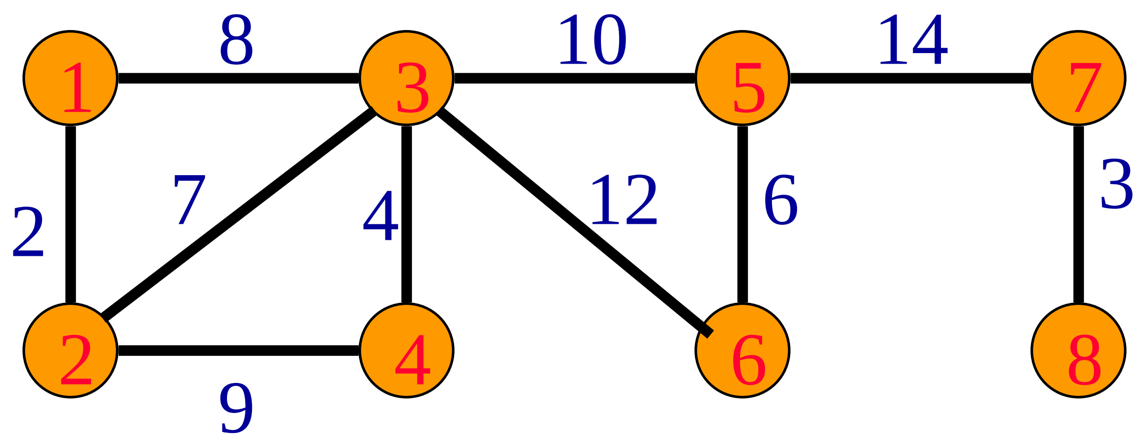
Complete weighted graph. (G_w)



Minimum-weight spanning tree.

@Sudharshan Welihinda - 2021

Example



- Network has 10 edges.
- Spanning tree has only n - 1 = 7 edges.
- Need to either select 7 edges or discard 3.

Edge Selection Greedy Strategies
Kruskal's Method

- Start with an n-vertex 0-edge forest. Consider edges in ascending order of cost. Select edge if it does not form a cycle together with already selected edges.

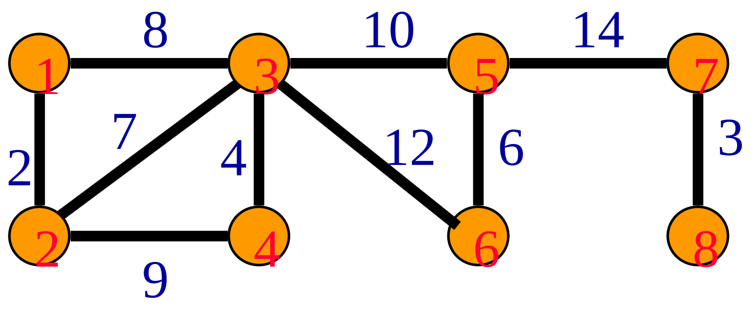
Edge Selection Greedy Strategies
Kruskal's Method

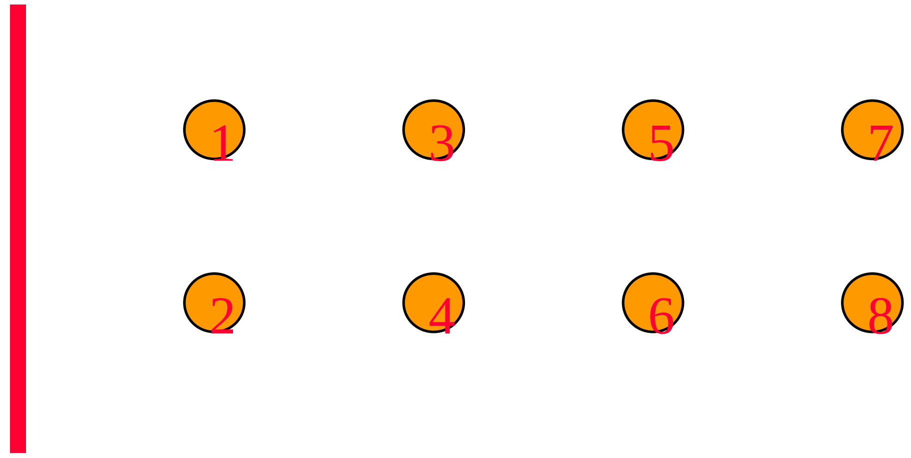
The steps are:

1. The forest is constructed - with each node in a separate tree.
2. The edges are placed in a priority queue.
3. Until we've added n-1 edges,
    1. Extract the cheapest edge from the queue,
    2. If it forms a cycle, reject it,
    3. Else add it to the forest. Adding it to the forest will join two trees together.

Every step will have joined two trees in the forest together, so that at the end, there will only be one tree in T.

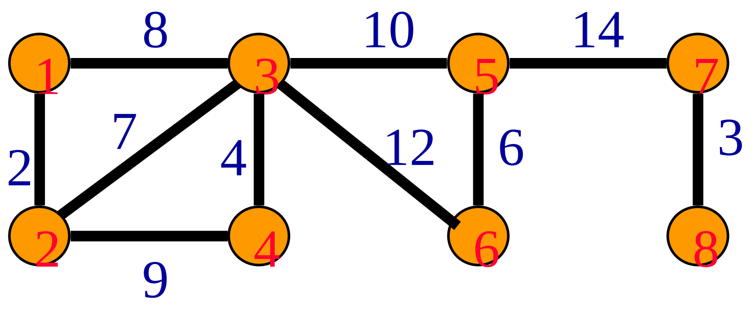
Kruskal's Method

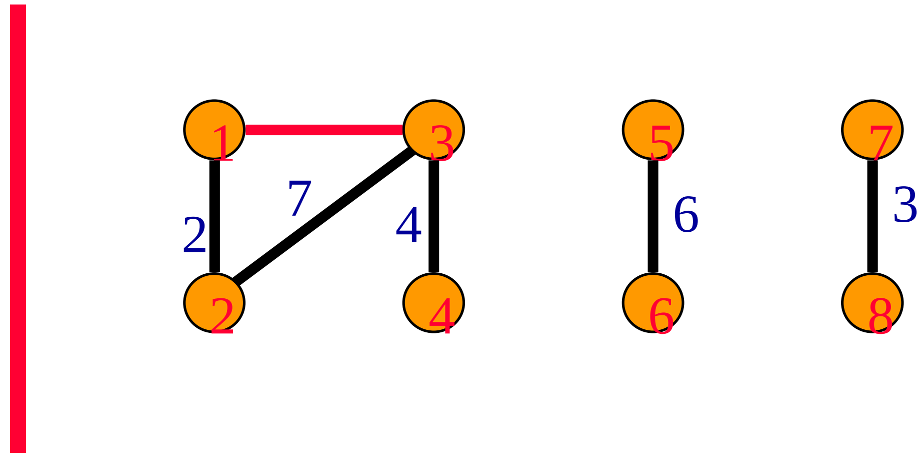




- Start with a forest that has no edges.
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.

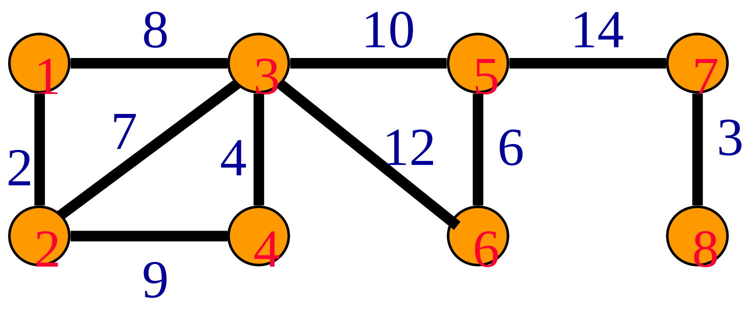
Kruskal’s Method

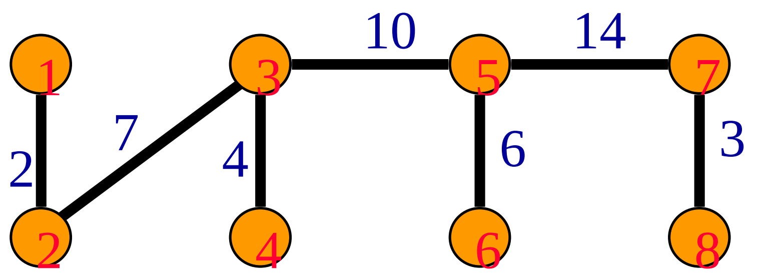




- Edge (7,8) is considered next and added.
- Edge (3,4) is considered next and added.
- Edge (5,6) is considered next and added.
- Edge (2,3) is considered next and added.
- Edge (1,3) is considered next and rejected because it creates a cycle.

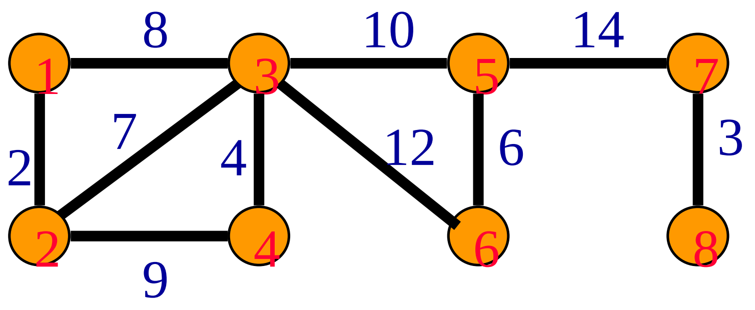
Kruskal's Method

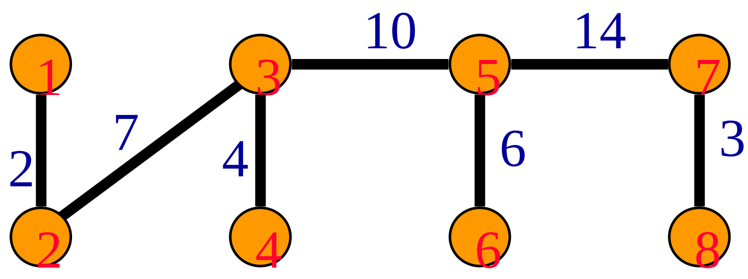




- Edge (2,4) is considered next and rejected because it creates a cycle.
- Edge (3,5) is considered next and added.
- Edge (3,6) is considered next and rejected.
- Edge (5,7) is considered next and added.

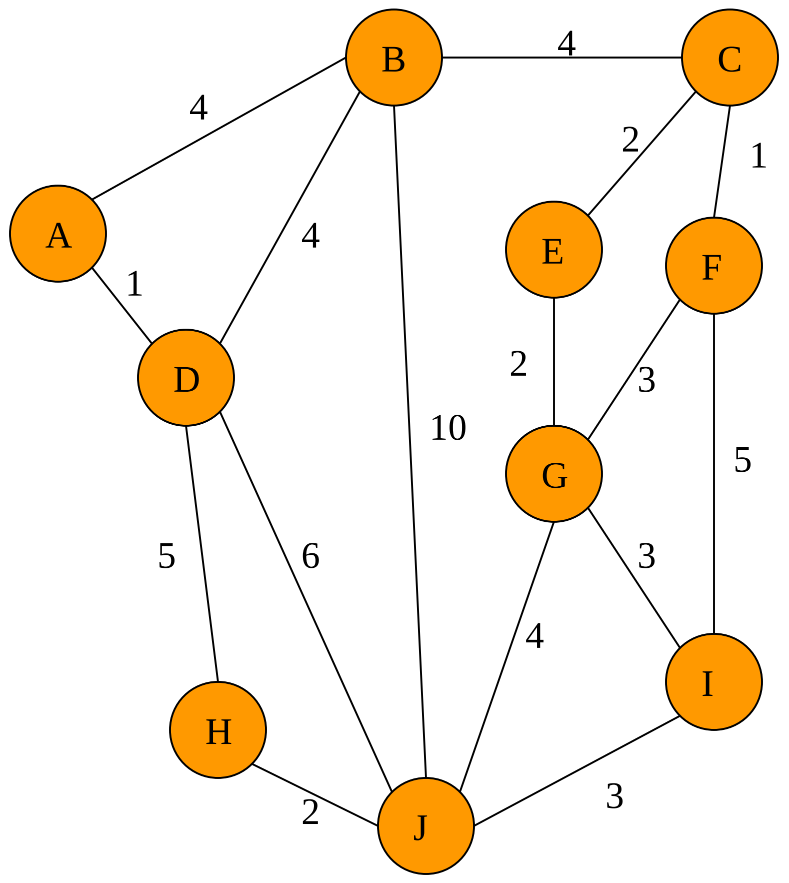
Kruskal’s Method



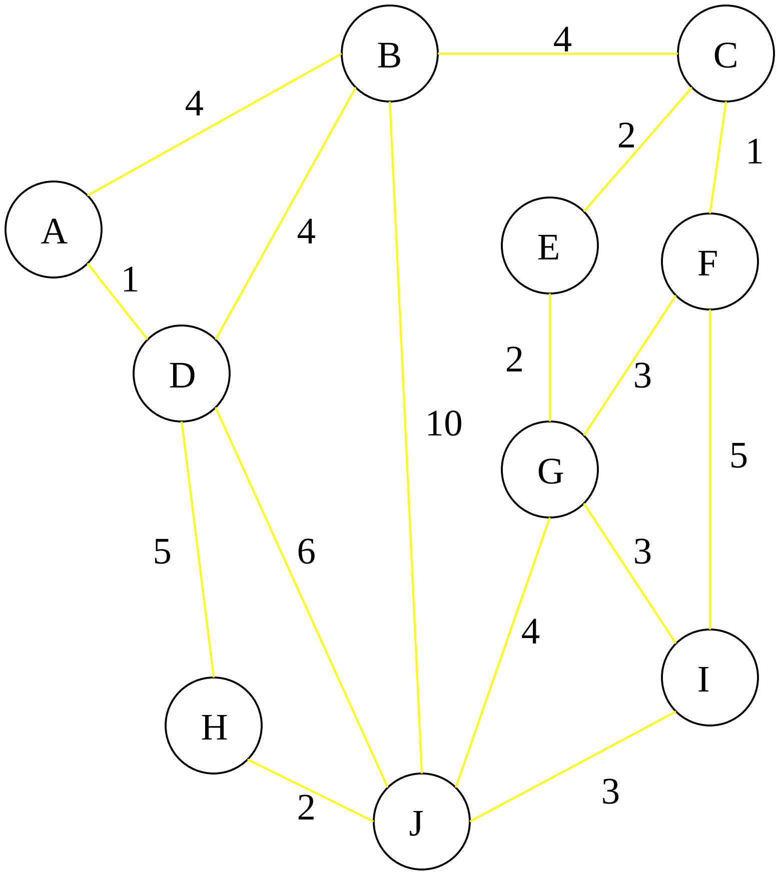


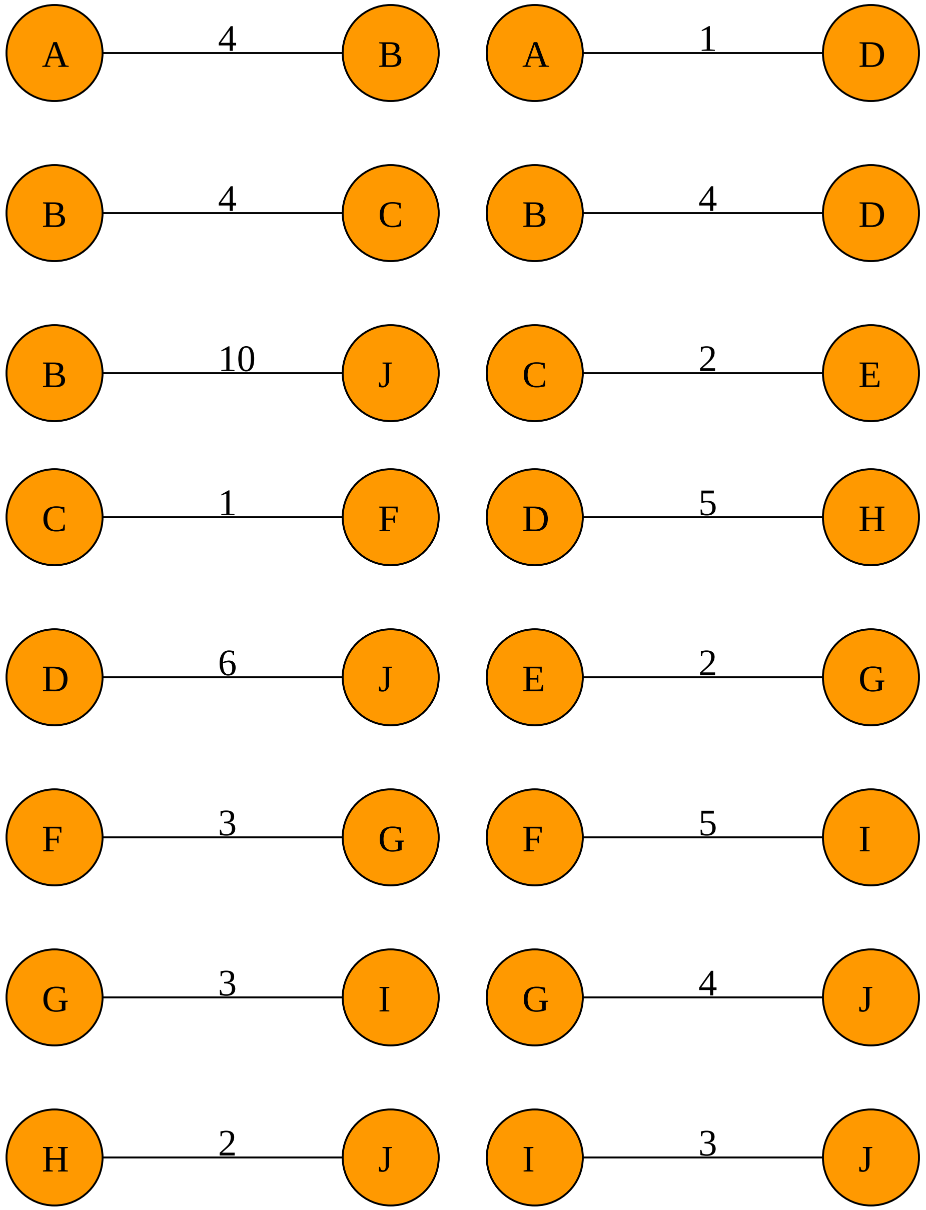
- n - 1 edges have been selected and no cycle formed.
- So we must have a spanning tree.
- Cost is 46.
- Min-cost spanning tree is unique when all edge costs are different.

Complete Graph



@Sudharshan Welihinda - 2021

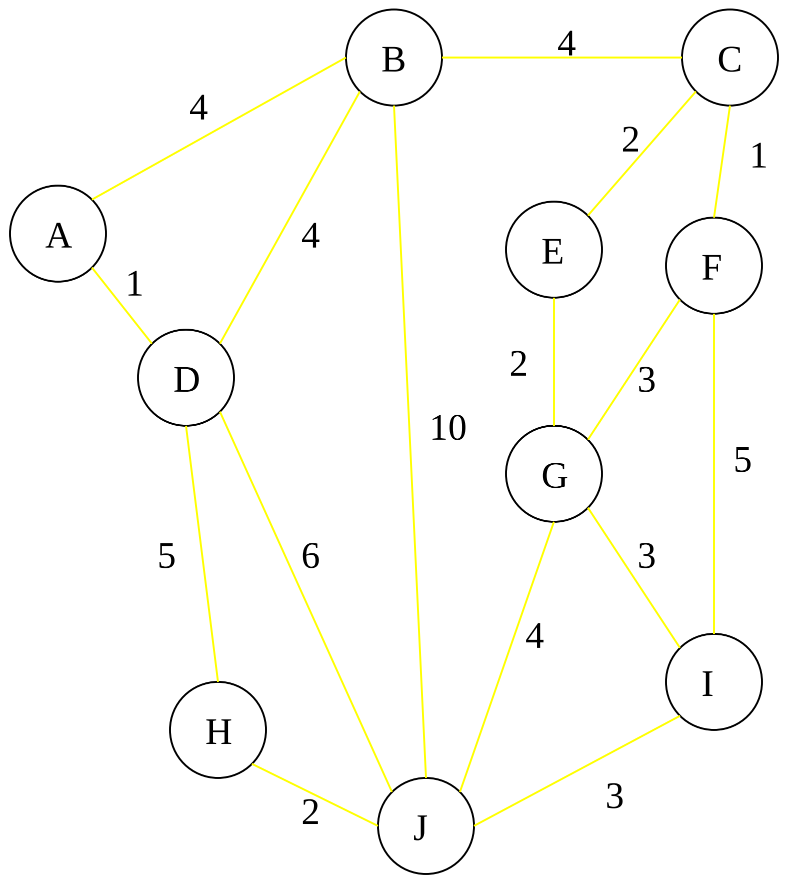


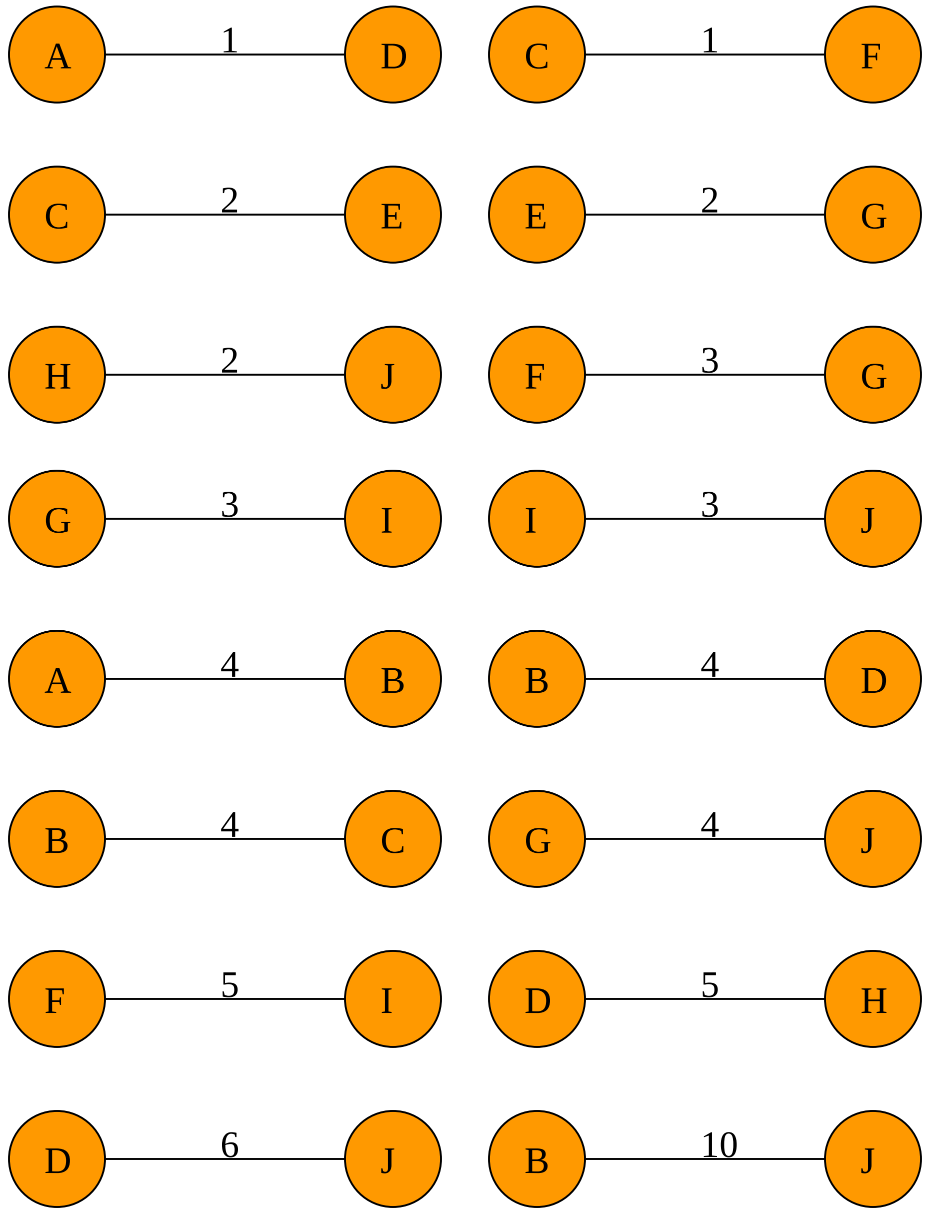


@Sudhars an Welihinda - 2021

Sort Edges

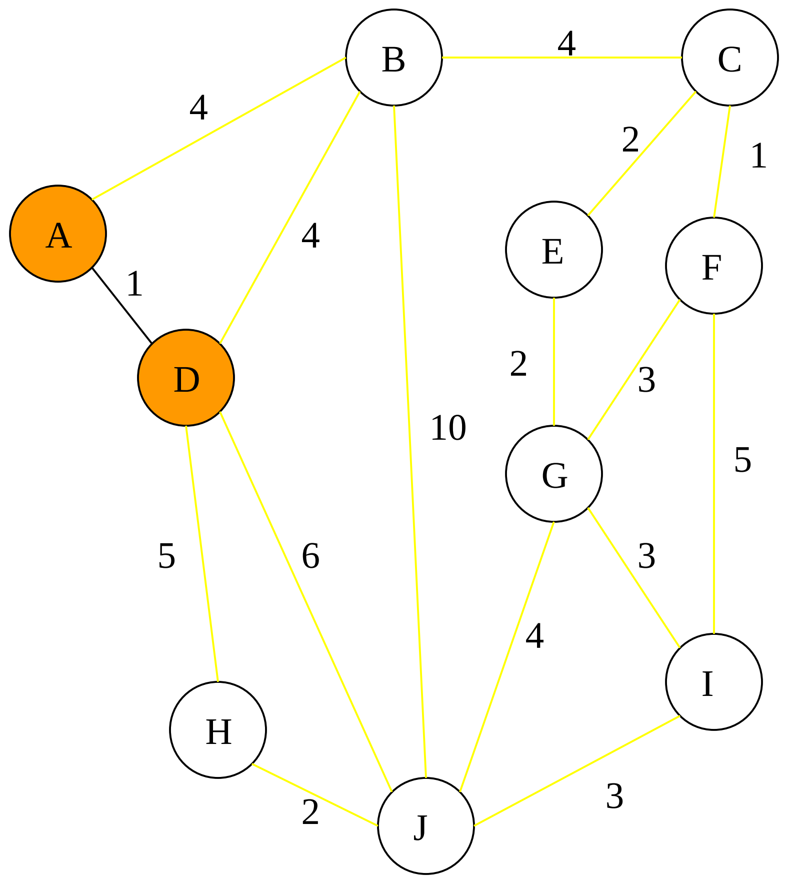
(in reality they are placed in a priority queue - not sorted - but sorting them makes the algorithm easier to visualize)

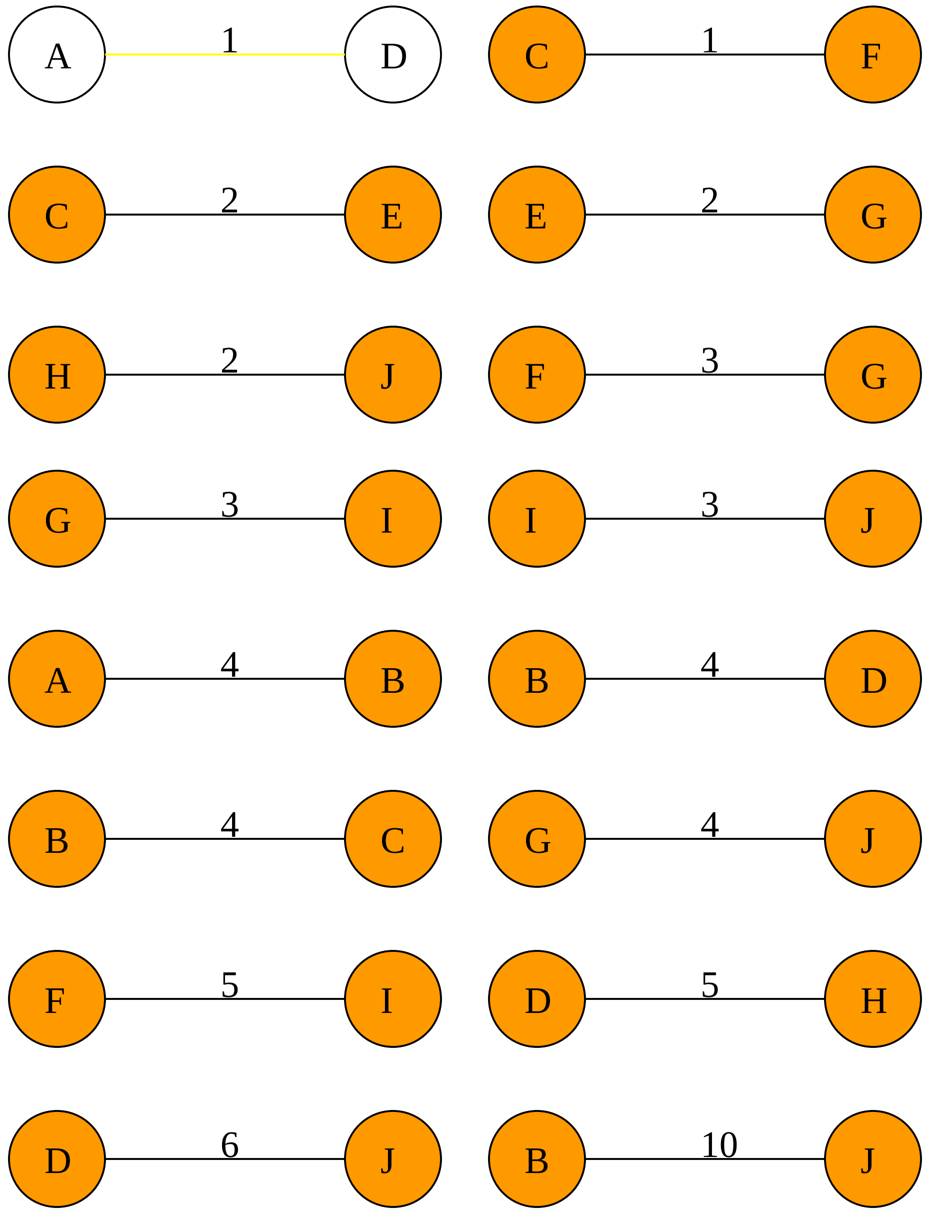




@Sudharshan Welihinda - 2021

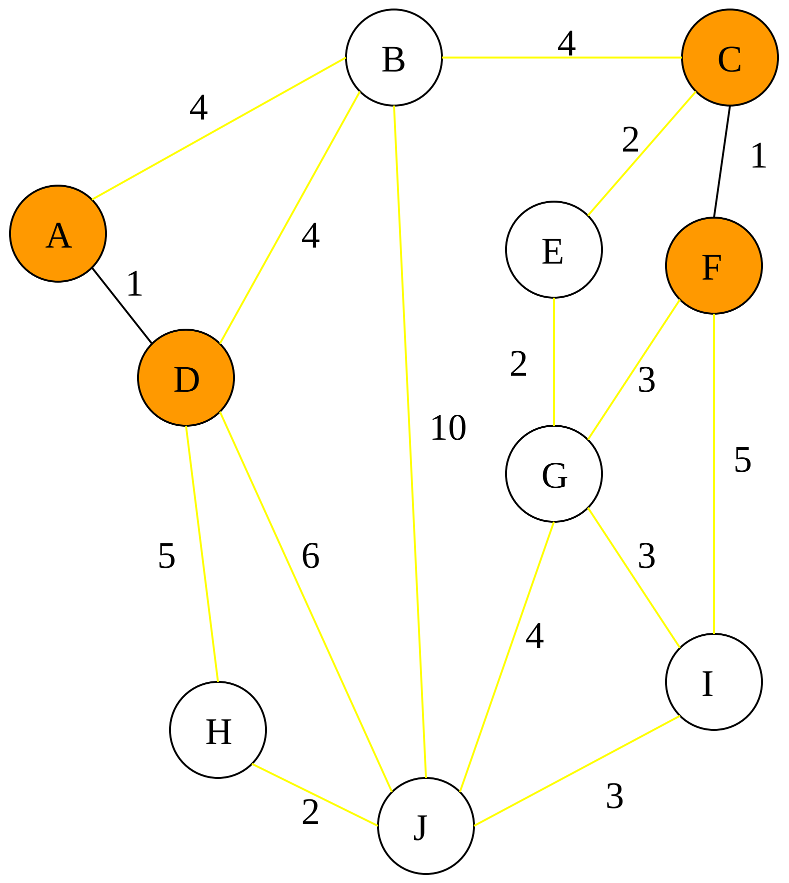
Add Edge

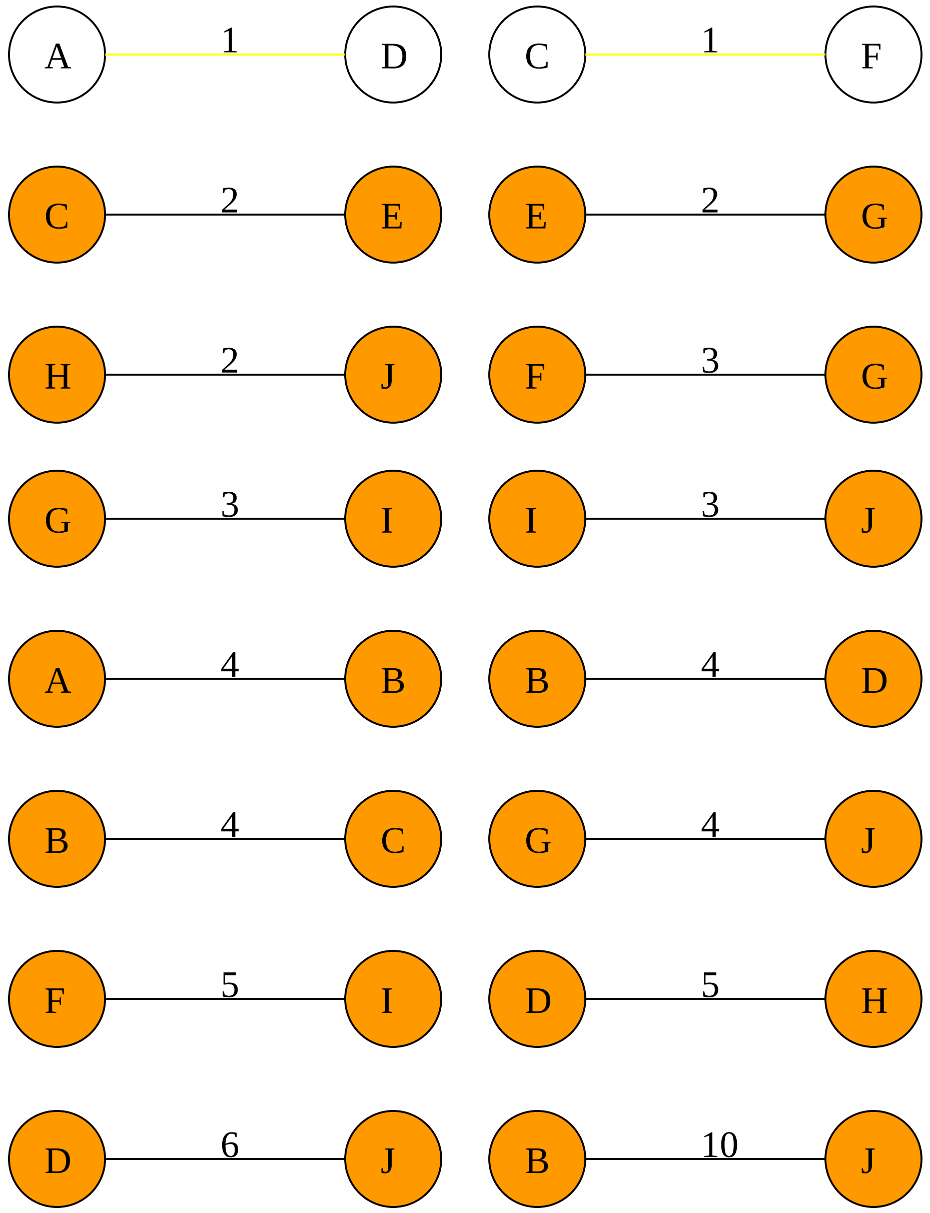




@Sudharshan Welihinda - 2021

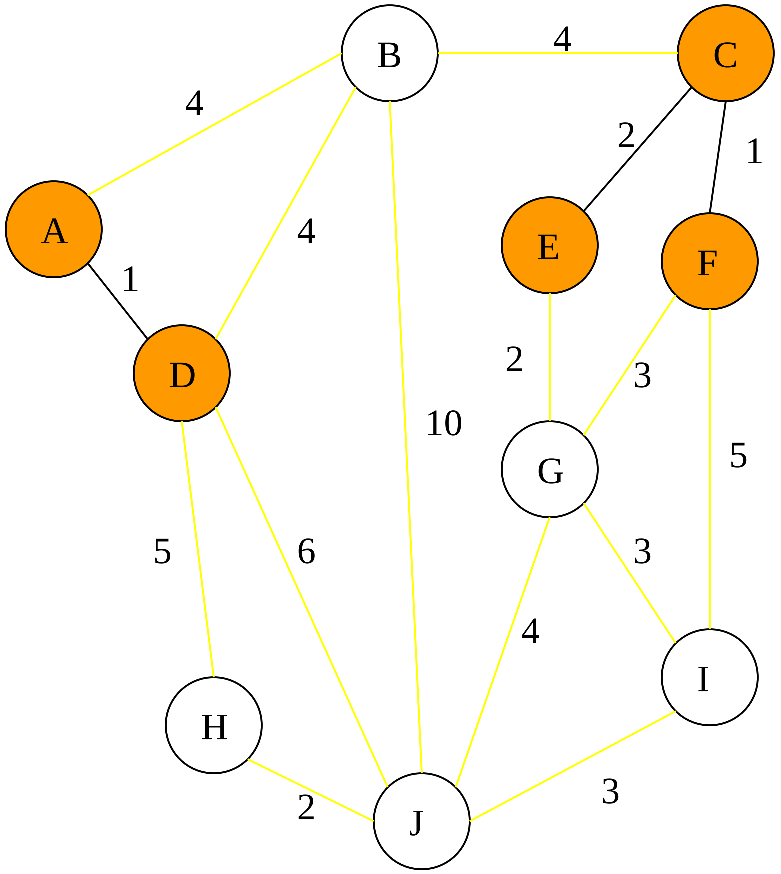
Add Edge

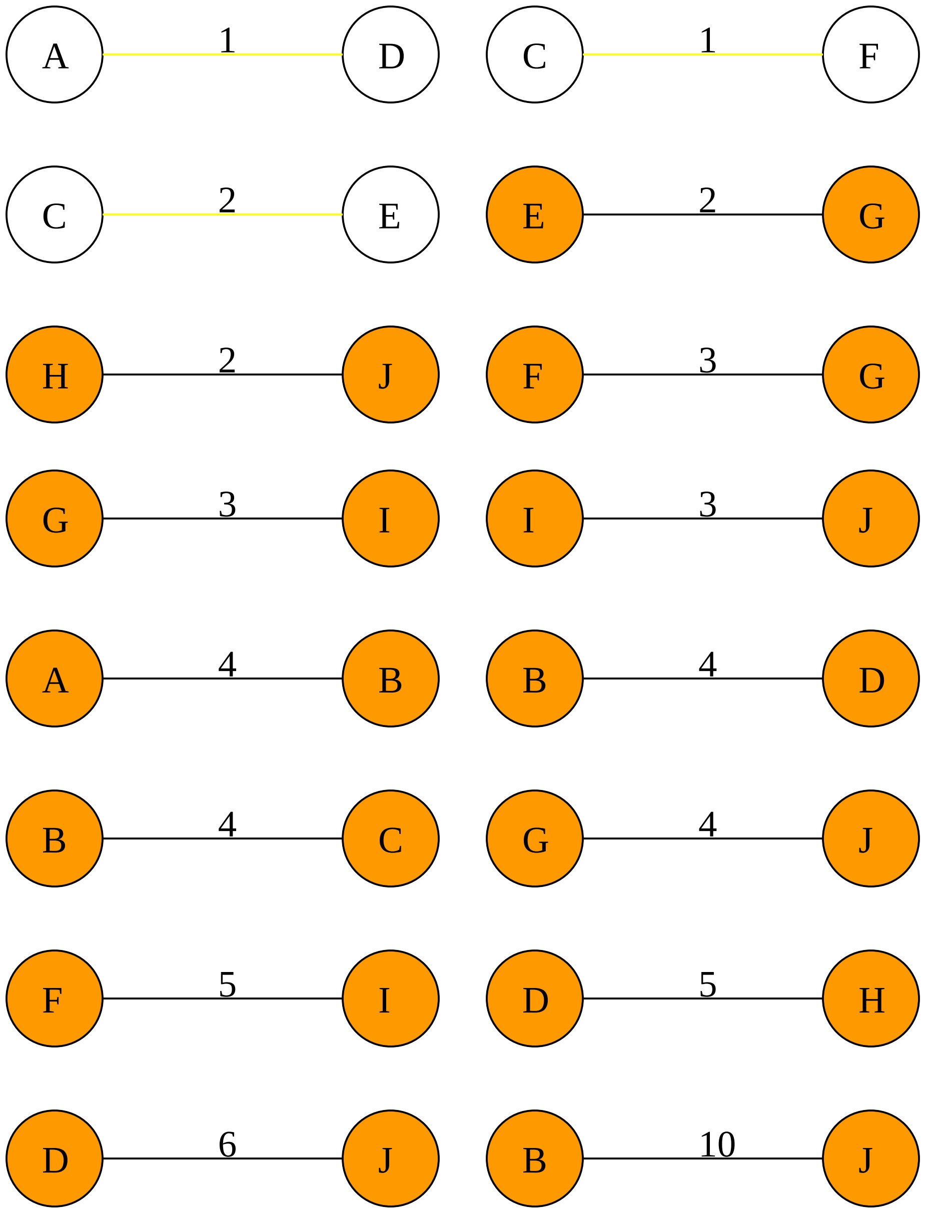




@Sudharshan Welihinda - 2021

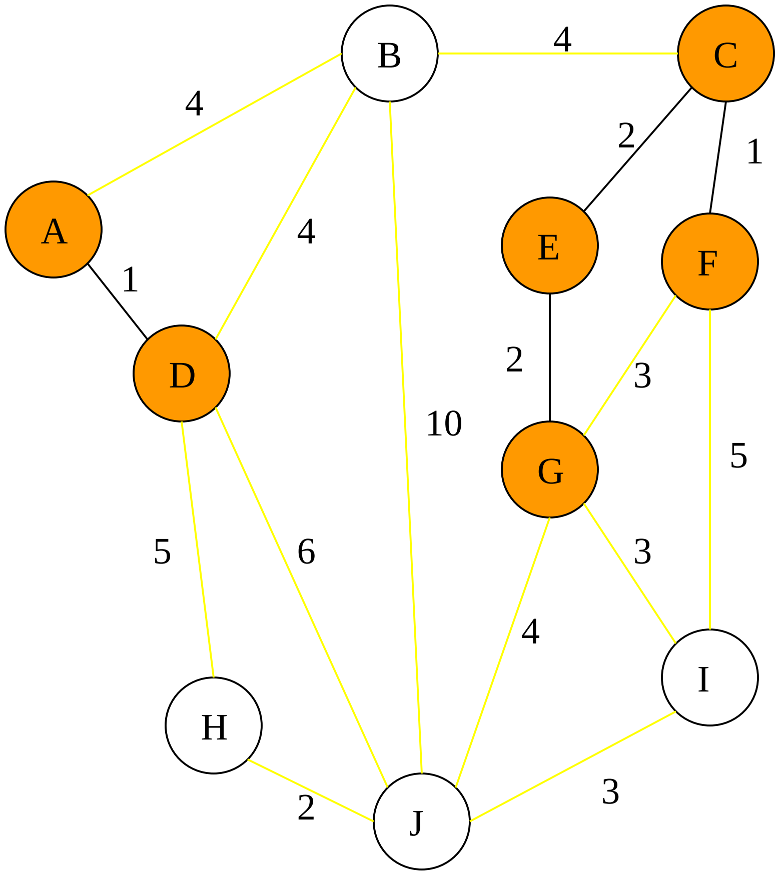
Add Edge

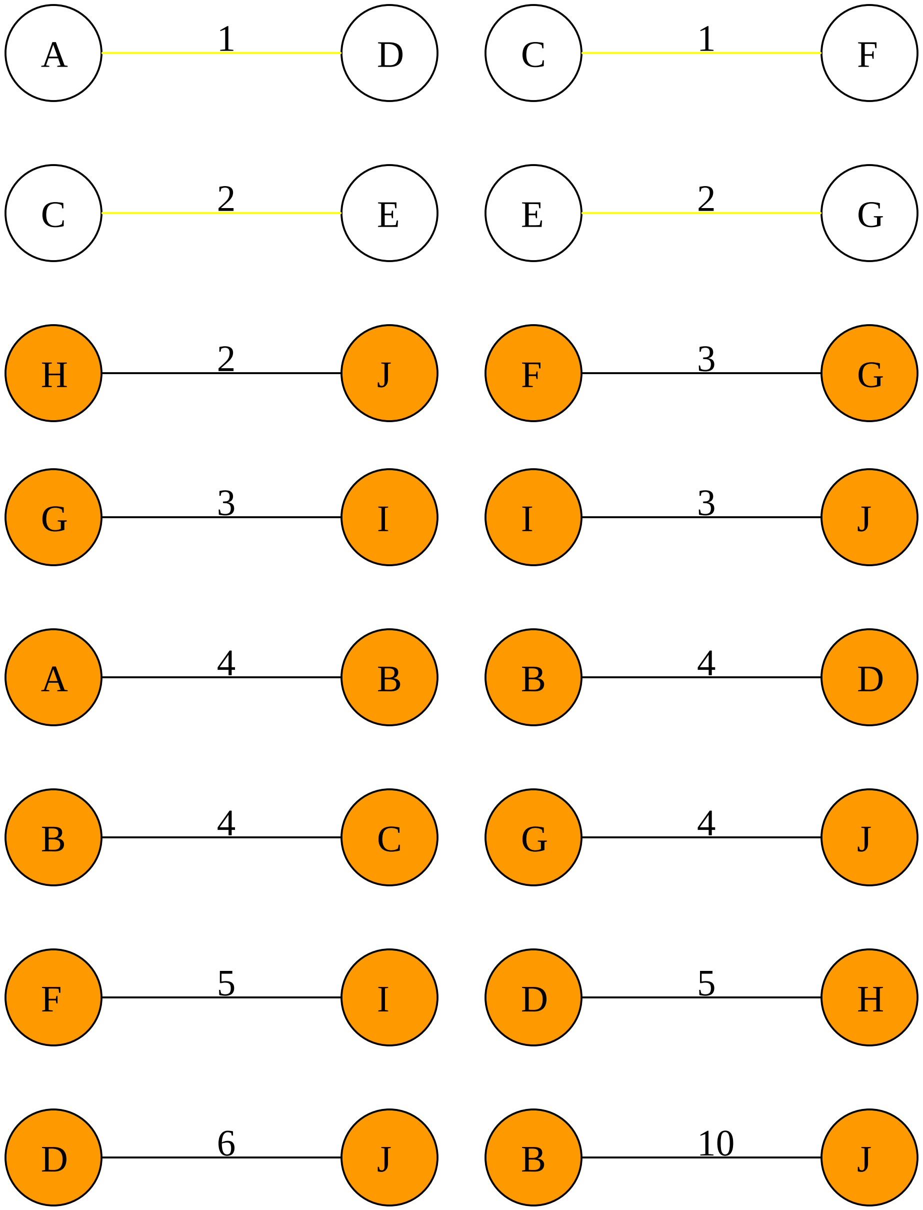




@Sudharshan Welihinda - 2021

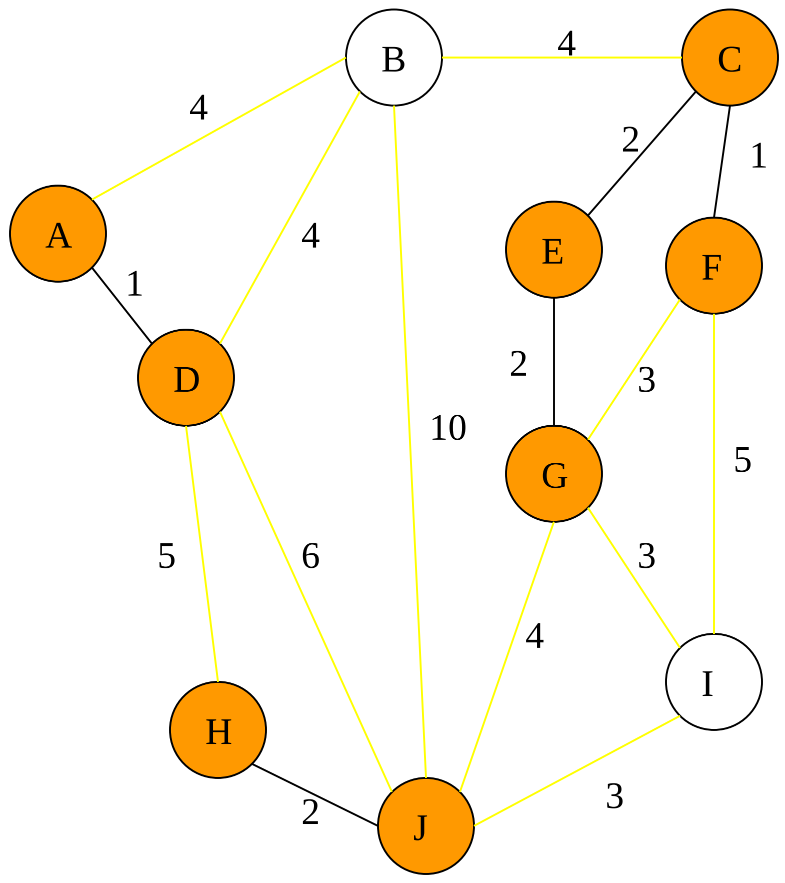
Add Edge

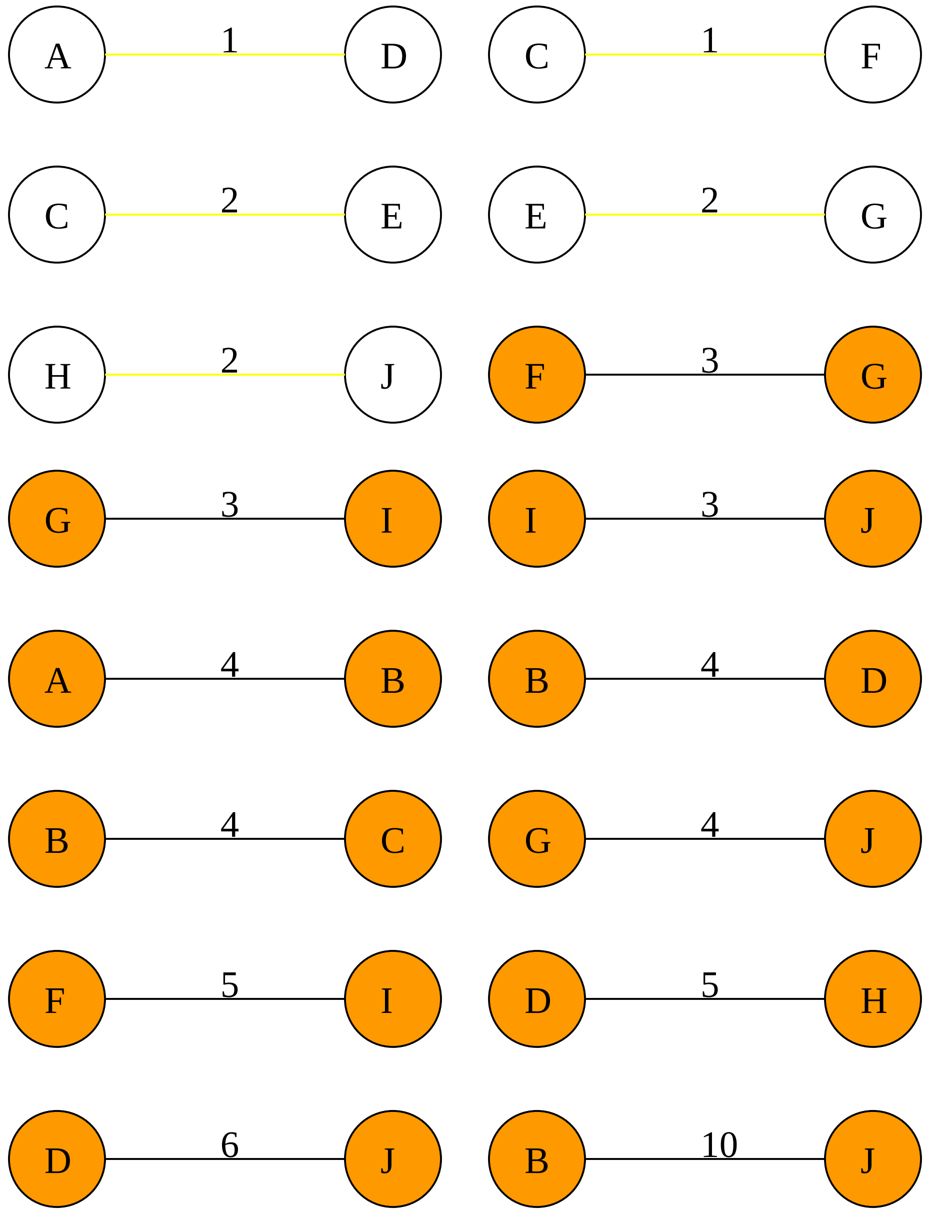




@Sudharshan Welihinda - 2021

Add Edge

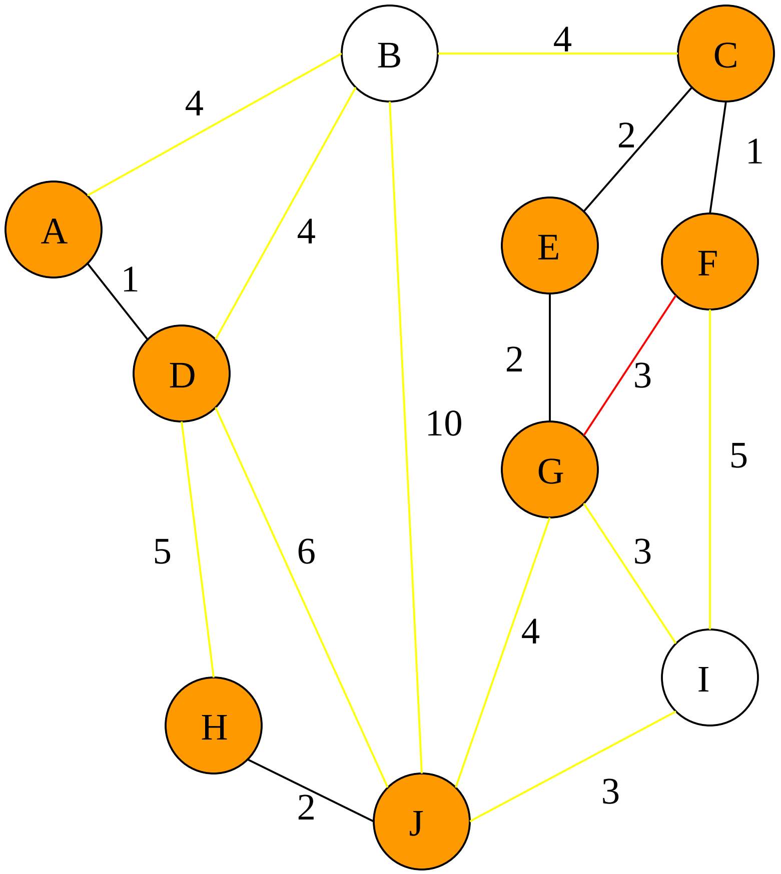


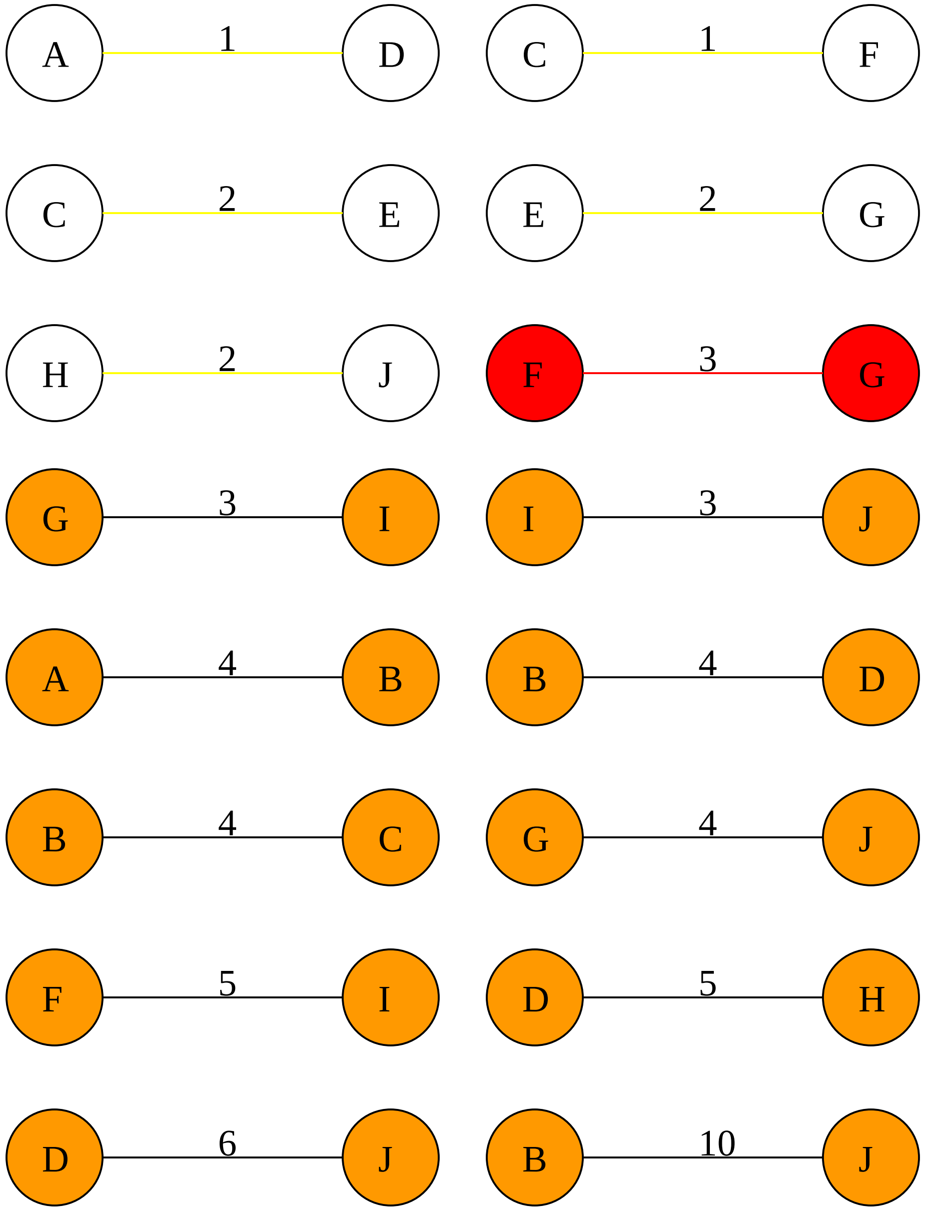


@Sudharshan Welihinda - 2021

Cycle

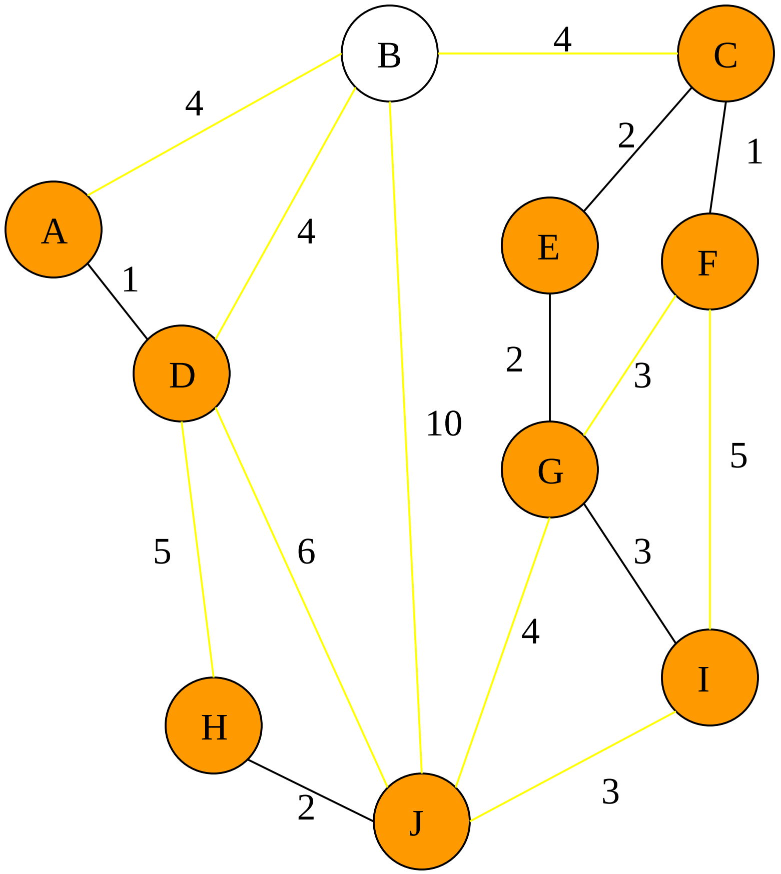
Don’t Add Edge

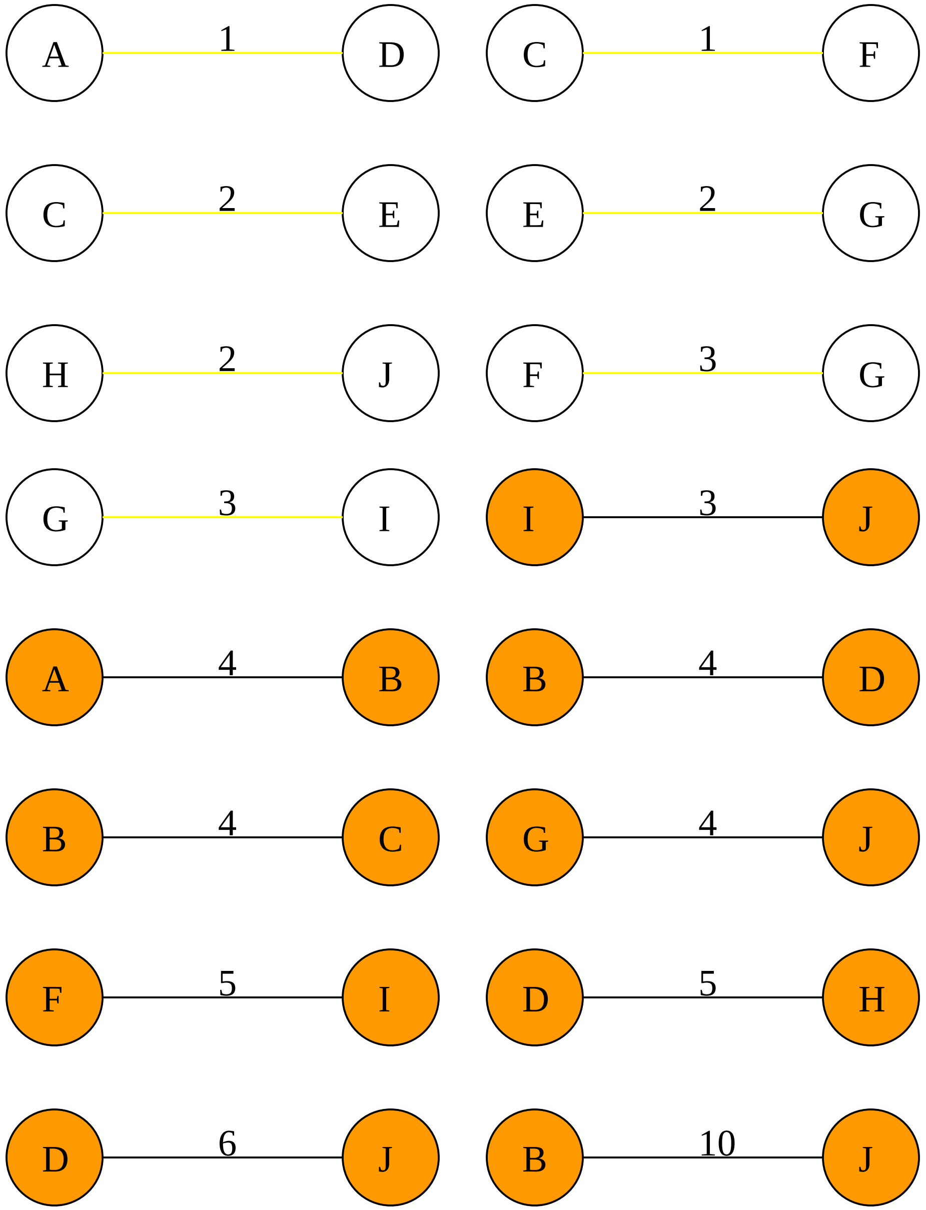




@Sudharshan Welihinda - 2021

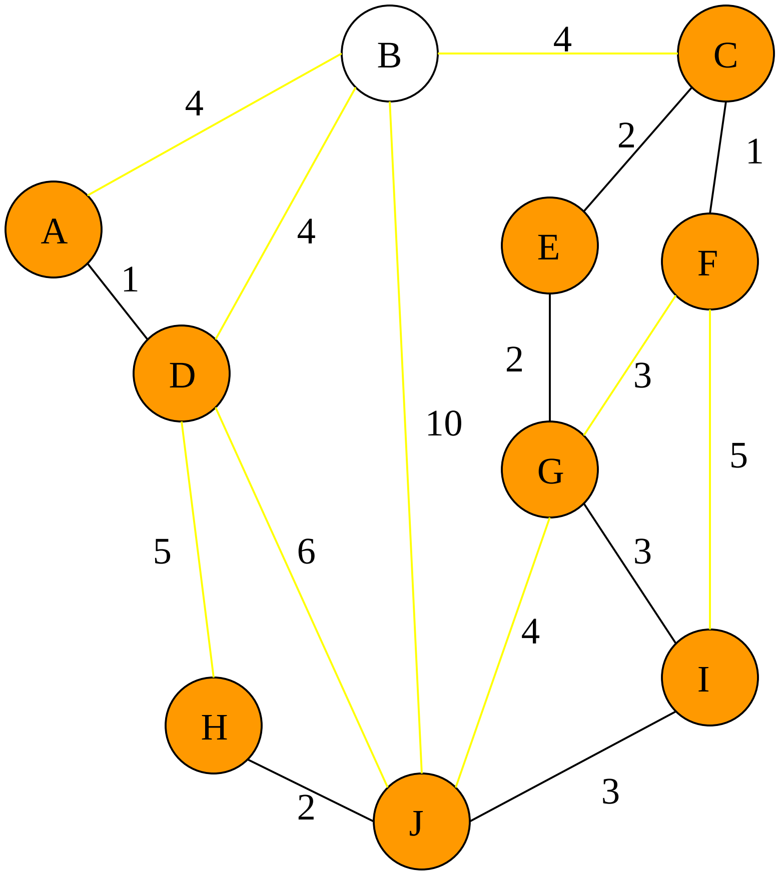
Add Edge

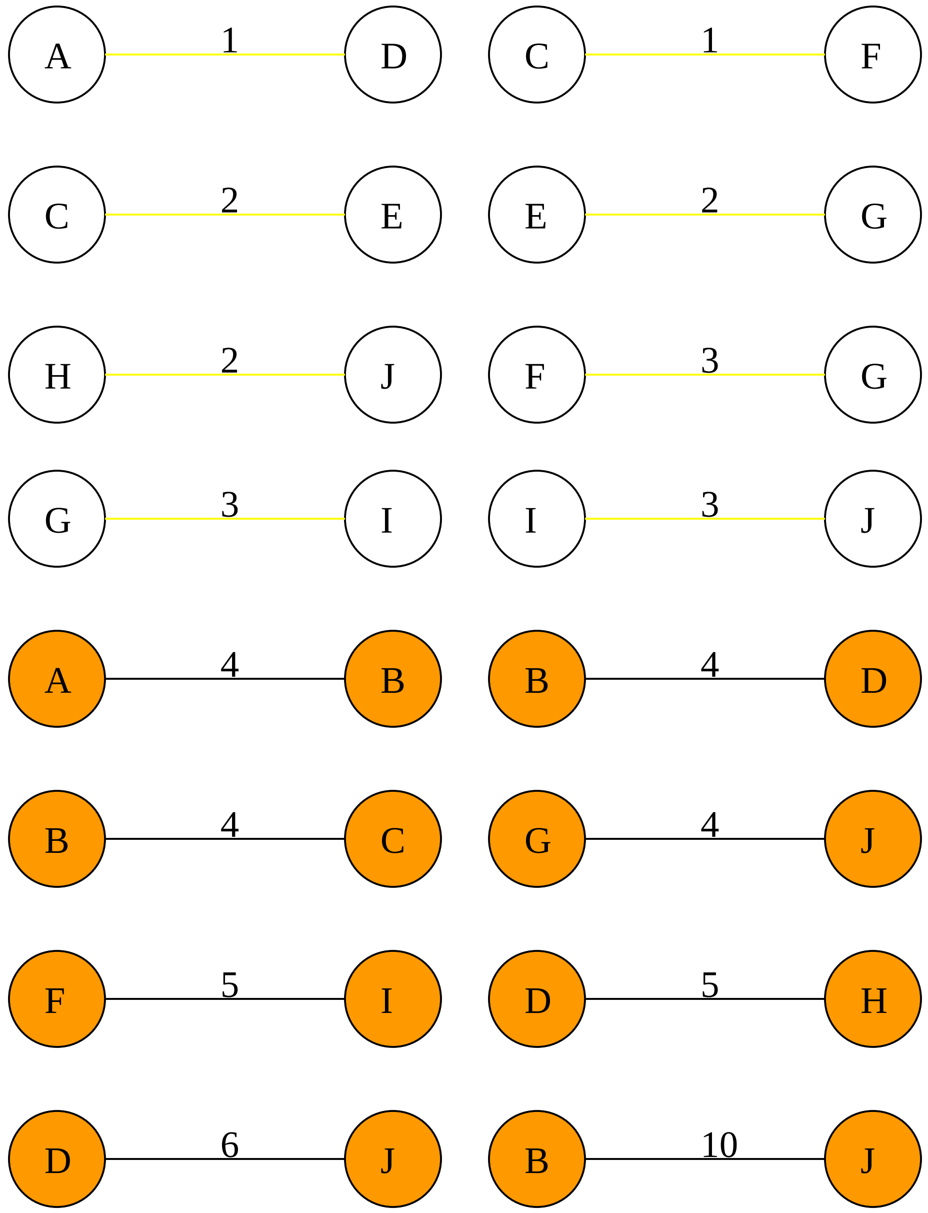




@Sudharshan Welihinda - 2021

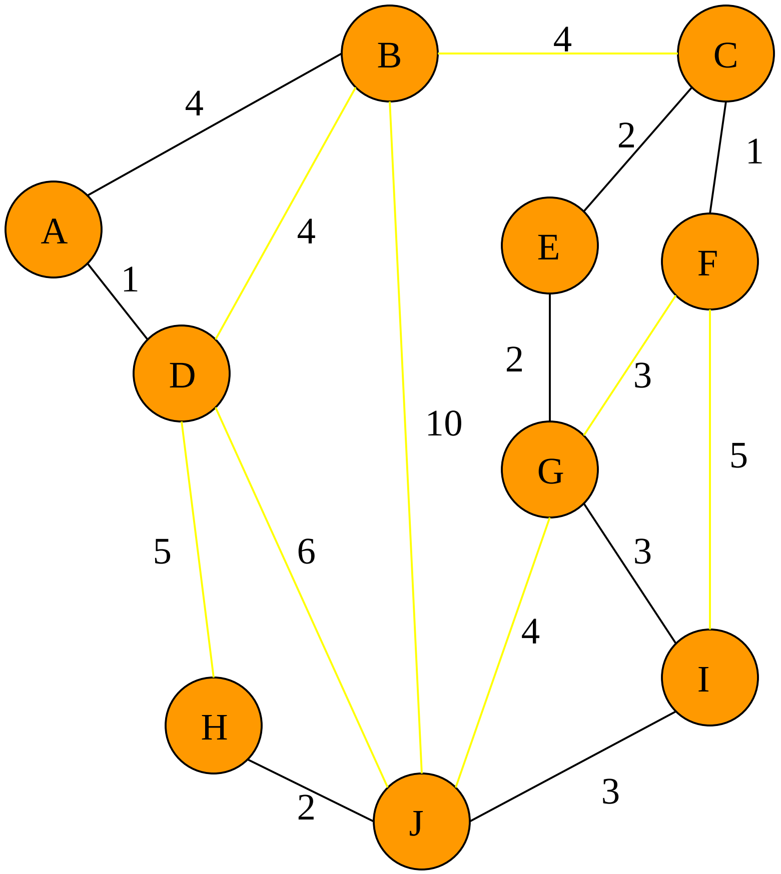
Add Edge

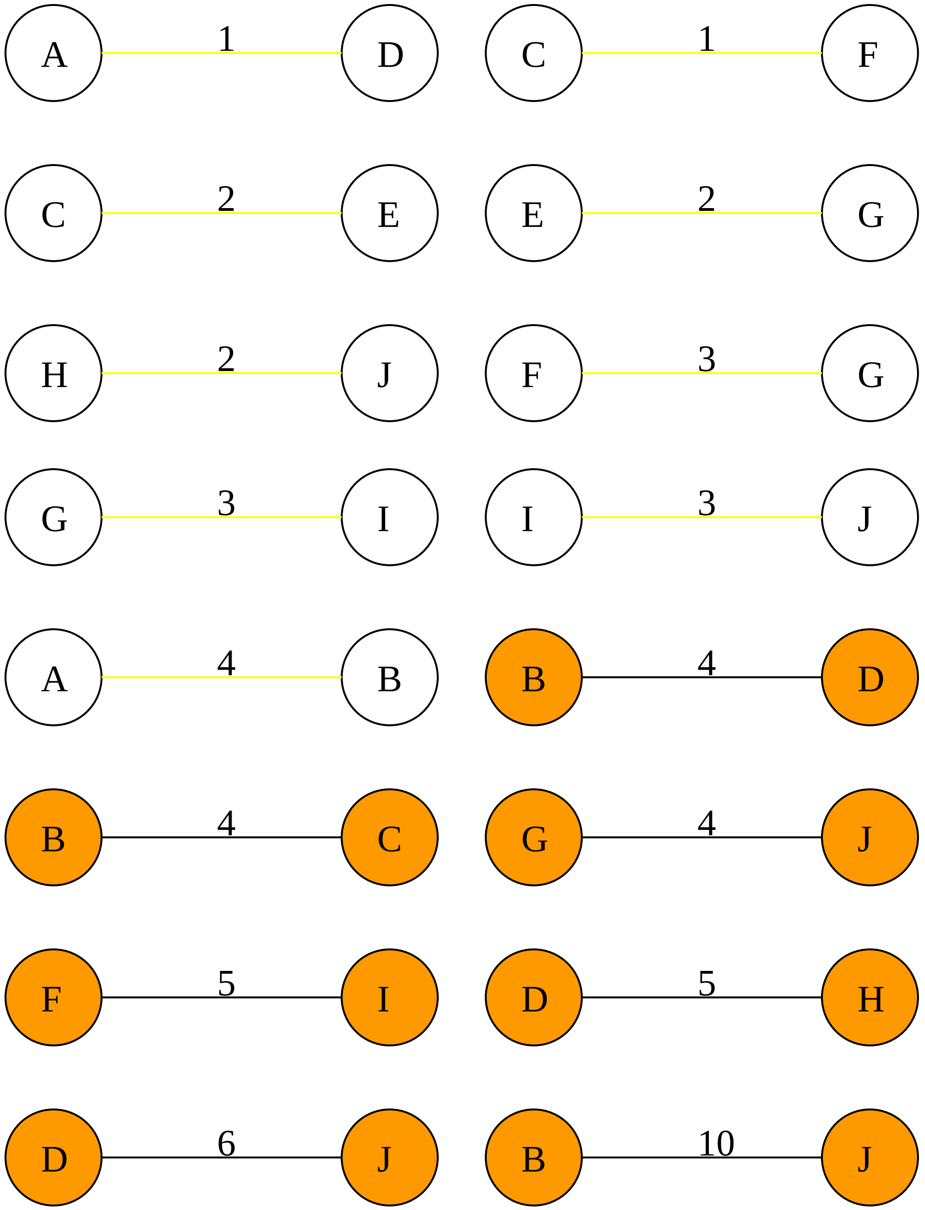




@Sudharshan Welihinda - 2021

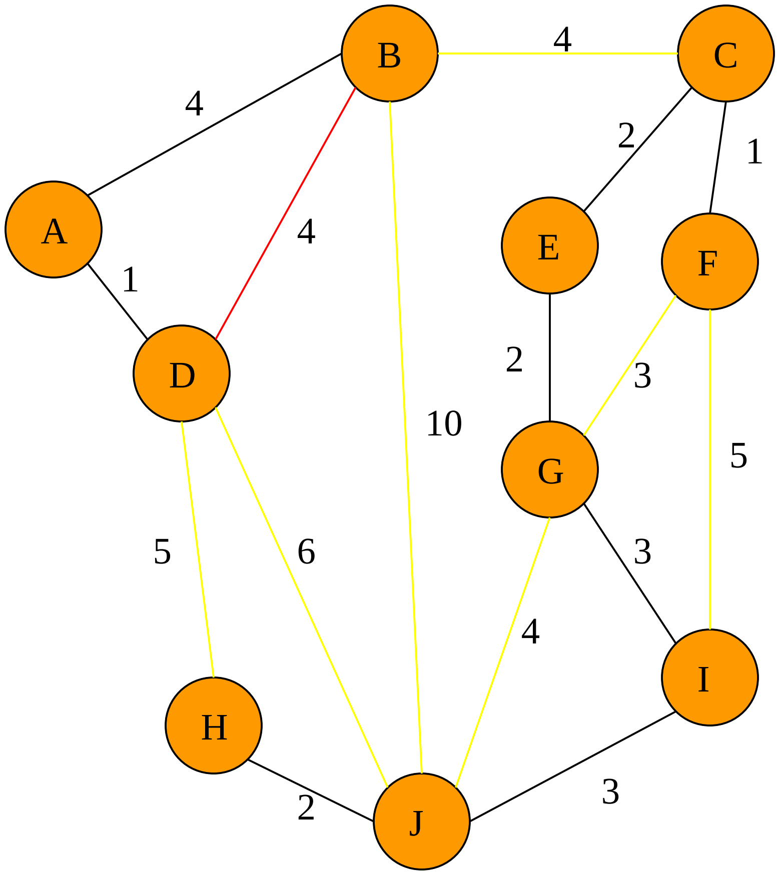
Add Edge

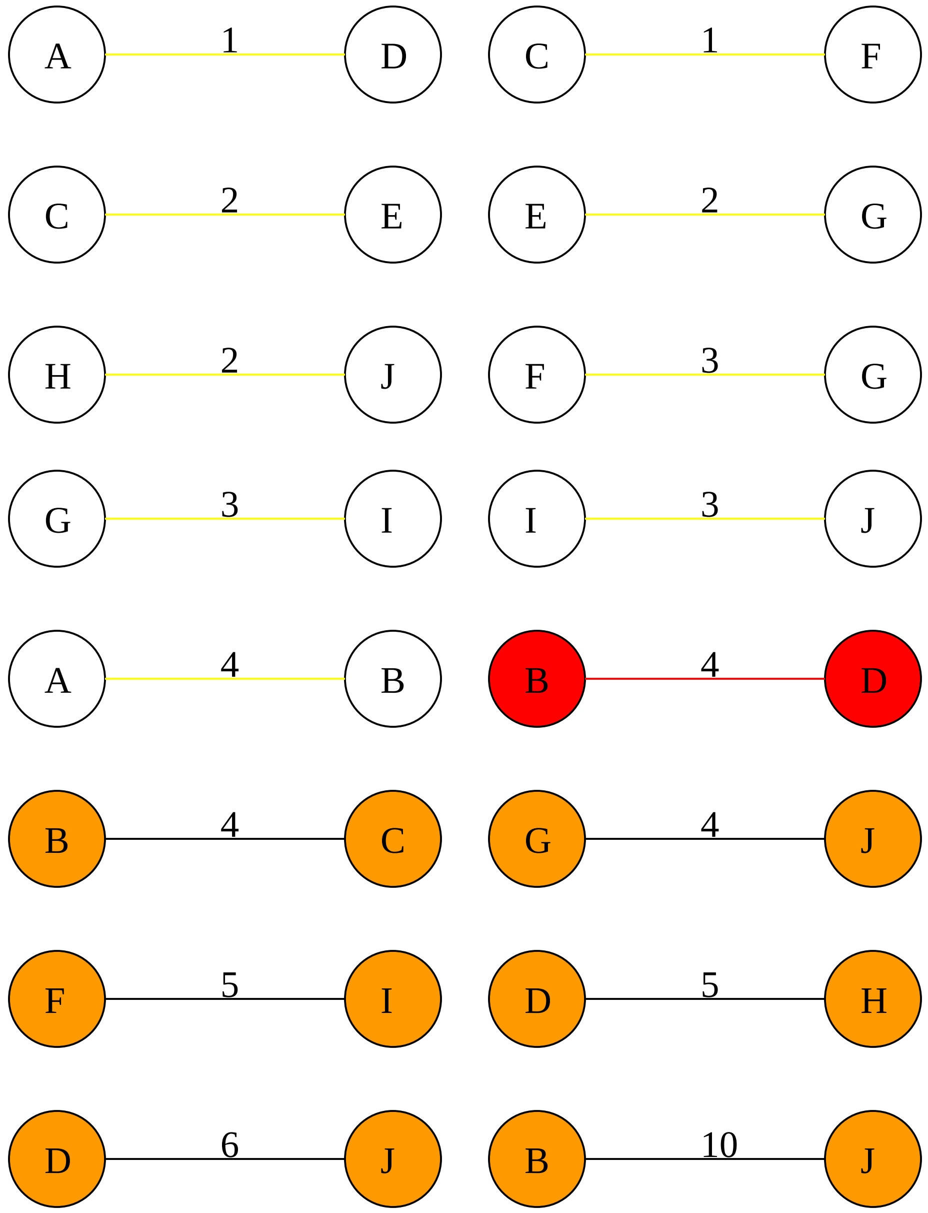

A
B
C
D
E
F
G
H
I
J
4
4
1
4
2
1
2
3
10
5
5
6
3
4
2
3


A 1 D
C 1 F
C 2 E
E 2 G
H 2 J
F 3 G
G 3 I
I 3 J
A 4 B
B 4 D
B 4 C
G 4 J
F 5 I
D 5 H
D 6 J
B 10 J

Cycle

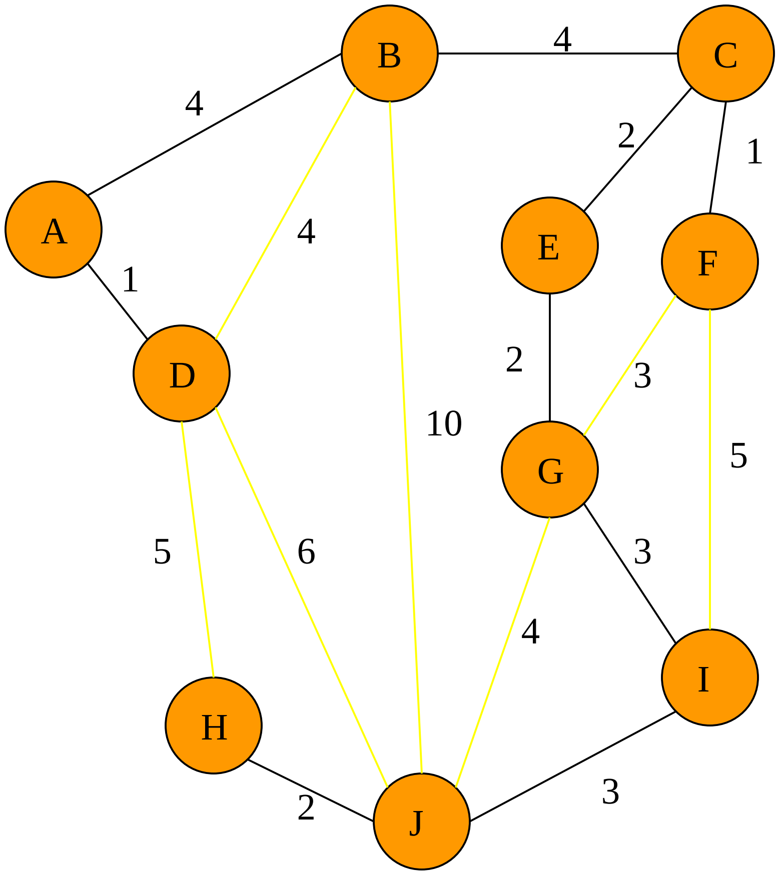
Don’t Add Edge

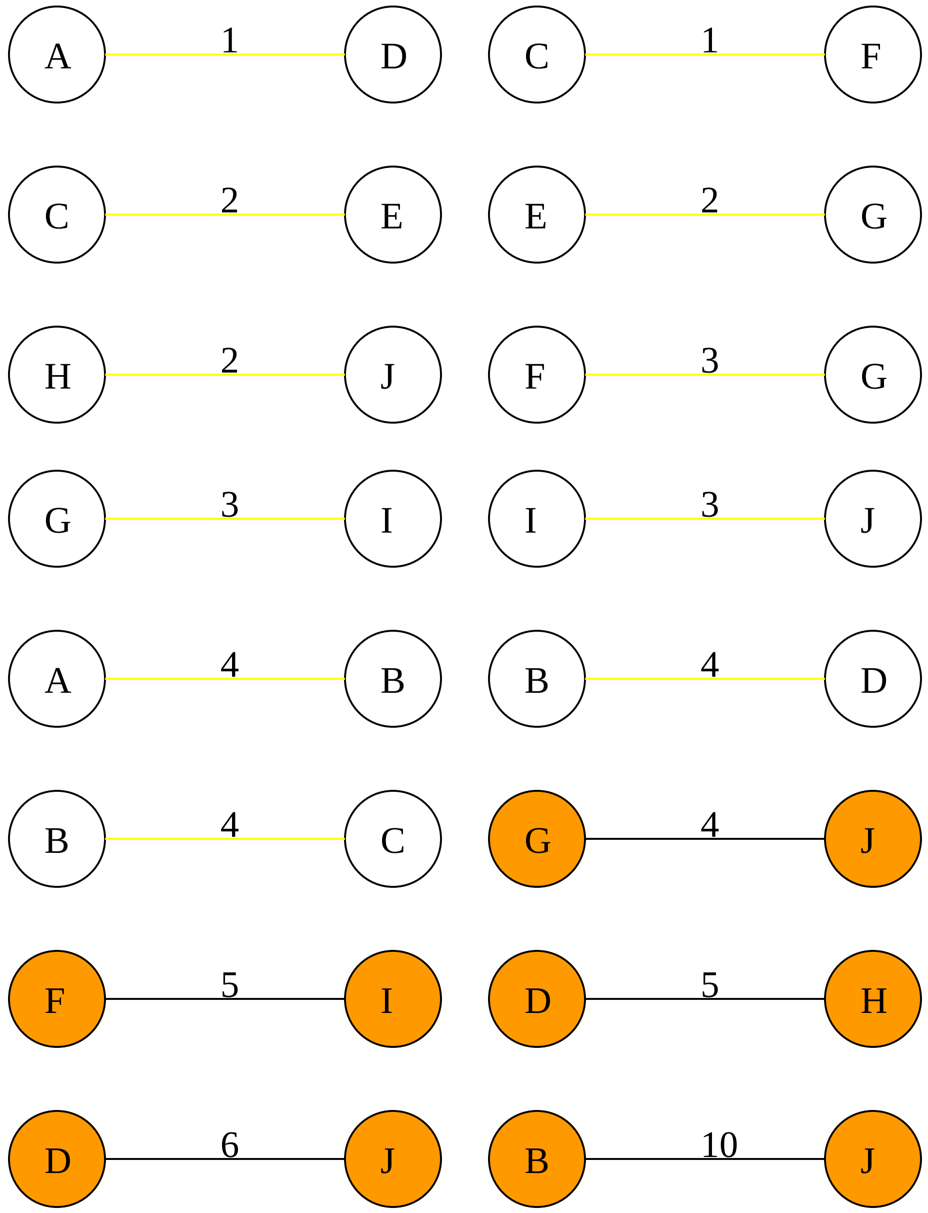




@Sudharshan Welihinda - 2021

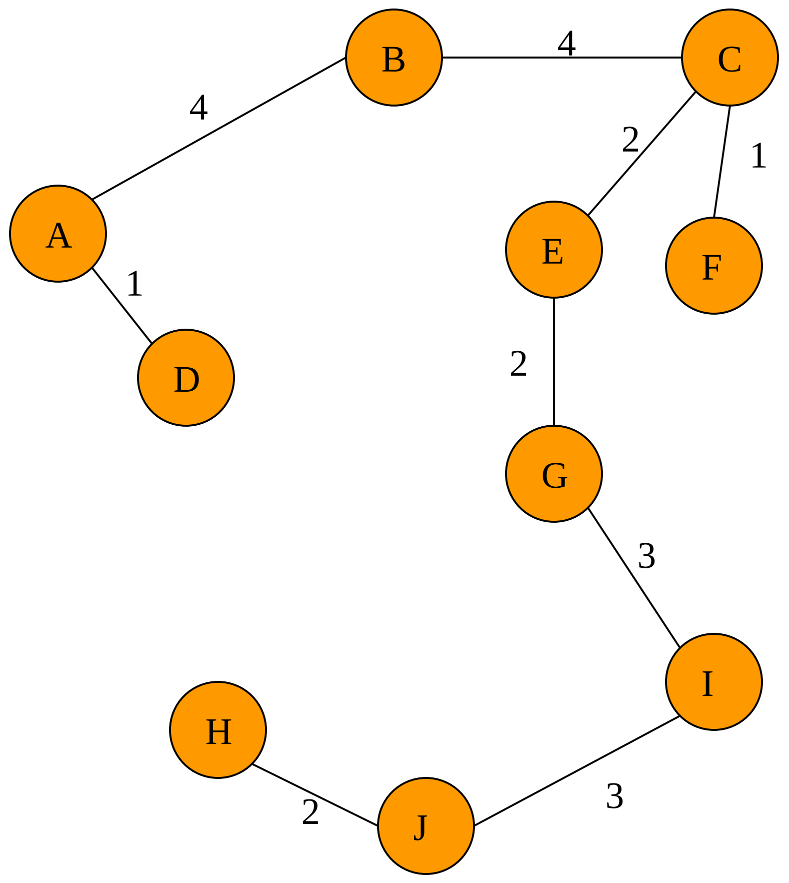
Add Edge



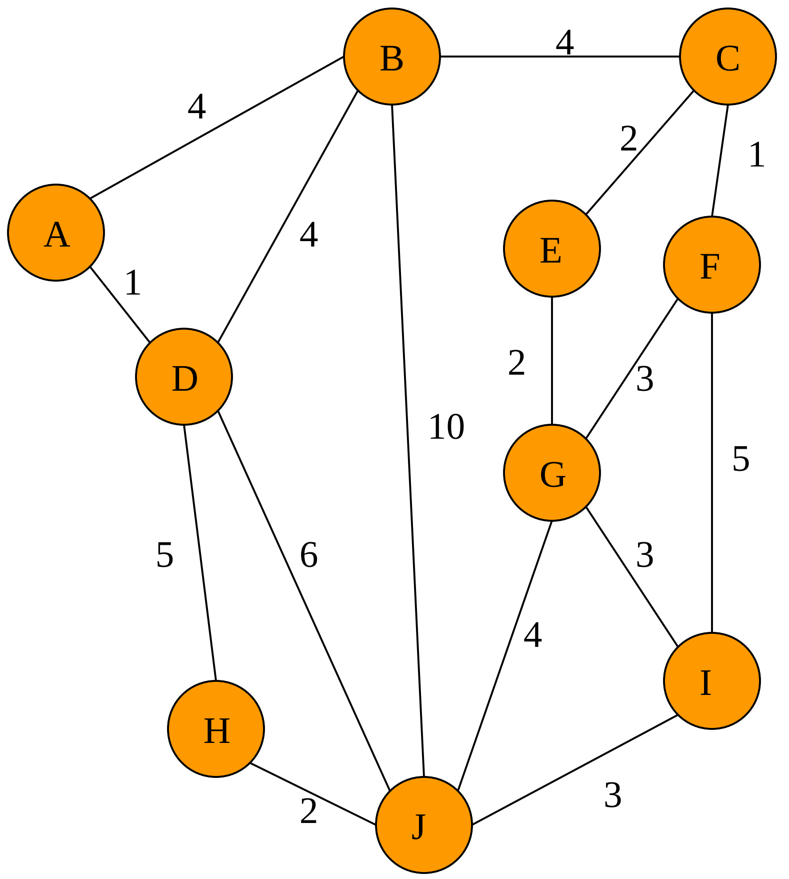


@Sudharshan Welihinda - 2021

Minimum Spanning Tree



Complete Graph



@Sudharshan Welihinda - 2021

Edge Selection Greedy Strategies
Prim’s Method

- Start with a 1-vertex tree and grow it into an n-vertex tree by repeatedly adding a vertex and an edge. When there is a choice, add a least cost edge.

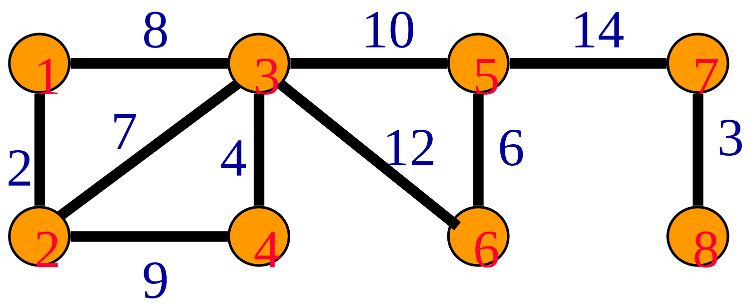
Edge Selection Greedy Strategies
Prim's Method

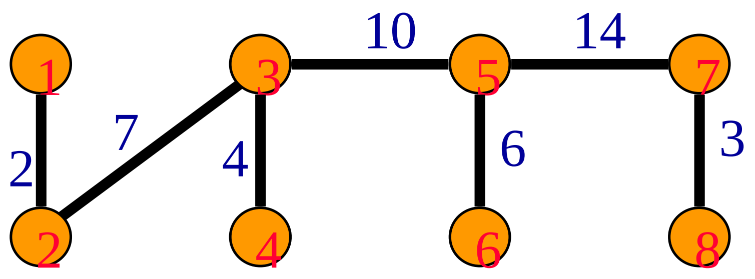
The steps are:

1. The new graph is constructed - with one node from the old graph.
2. While new graph has fewer than n nodes,
    1. Find the node from the old graph with the smallest connecting edge to the new graph,
    2. Add it to the new graph

Every step will have joined one node, so that at the end we will have one graph with all the nodes and it will be a minimum spanning tree of the original graph.

Prim's Method

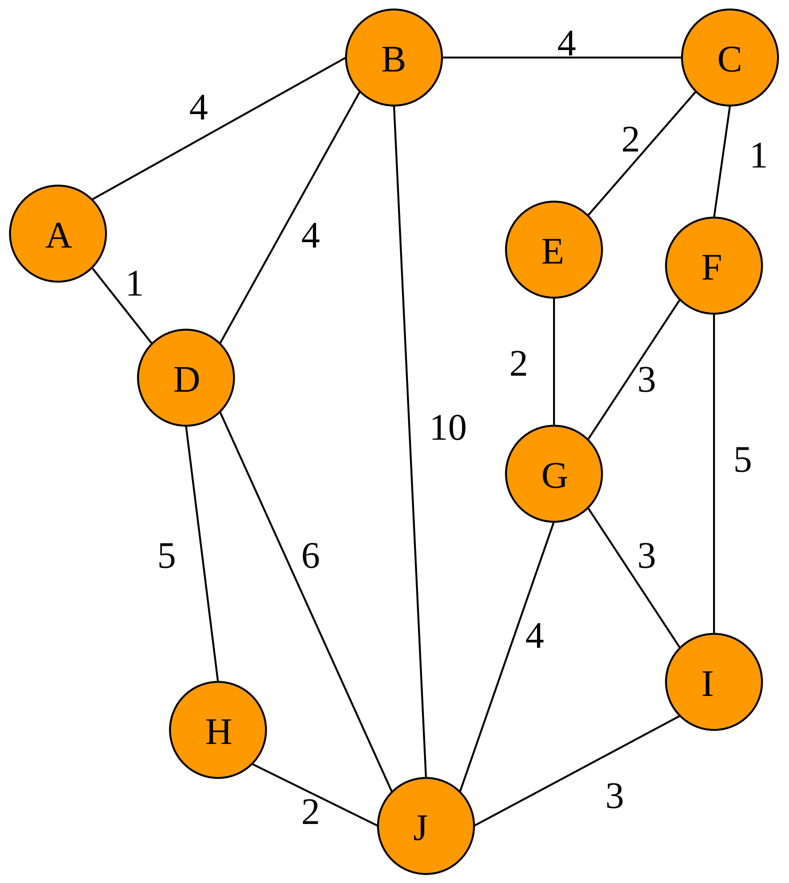




- Start with any single vertex tree.
- Get a 2-vertex tree by adding a cheapest edge.
- Get a 3-vertex tree by adding a cheapest edge.
- Grow the tree one edge at a time until the tree has n - 1 edges (and hence has all n vertices).

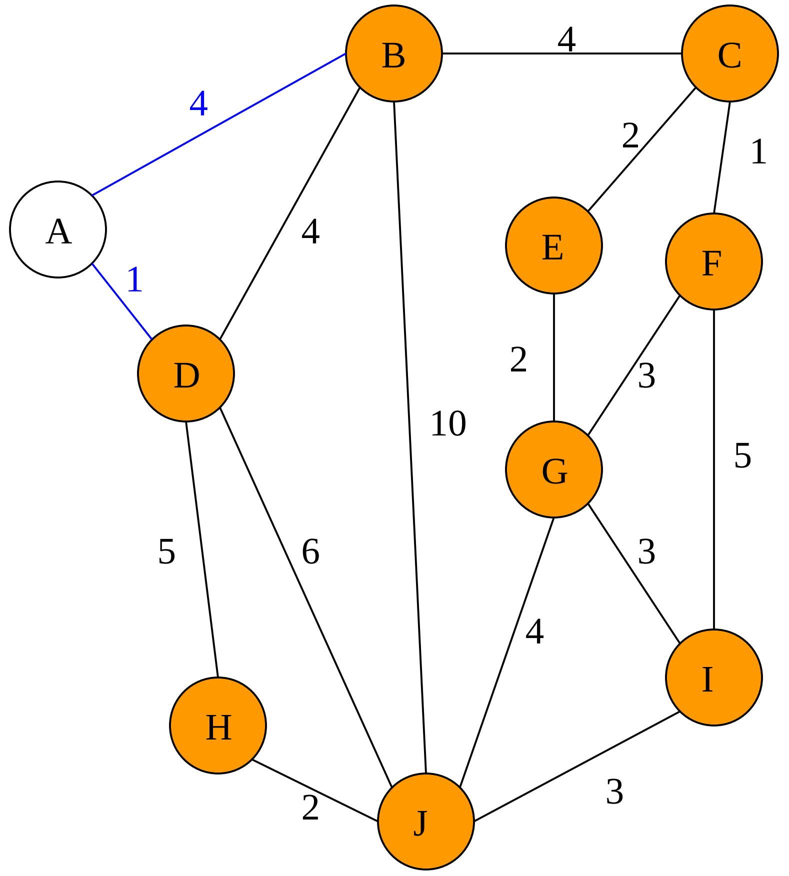
@Sudharshan Welihinda - 2021

Complete Graph

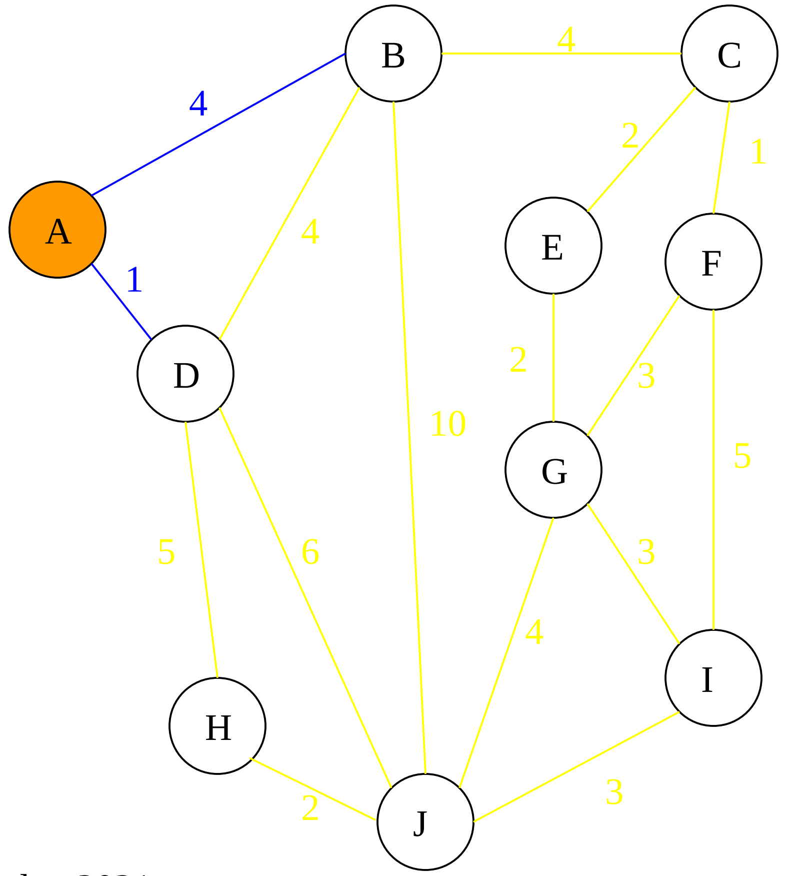


@Sudharshan Welihinda - 2021

Old Graph

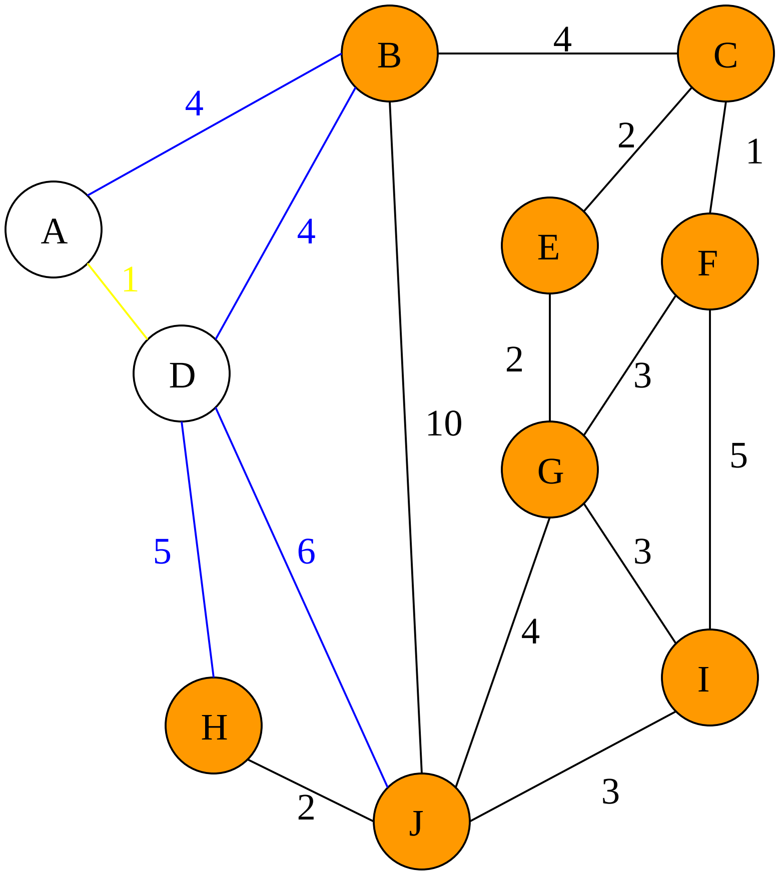


New Graph

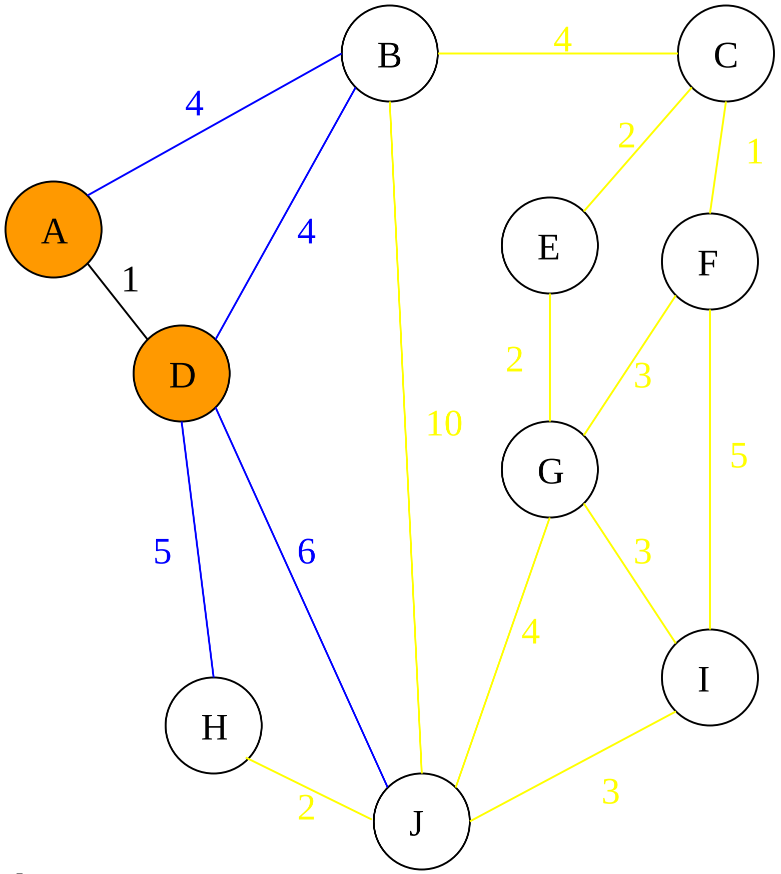


@Sudharshan Welihinda - 2021

Old Graph

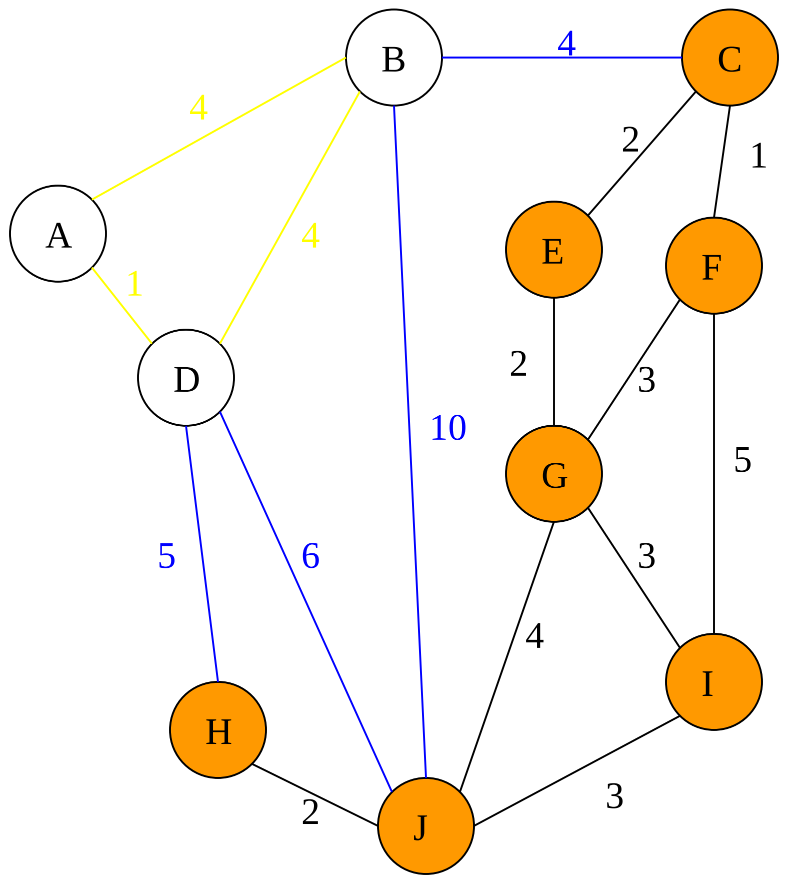


New Graph

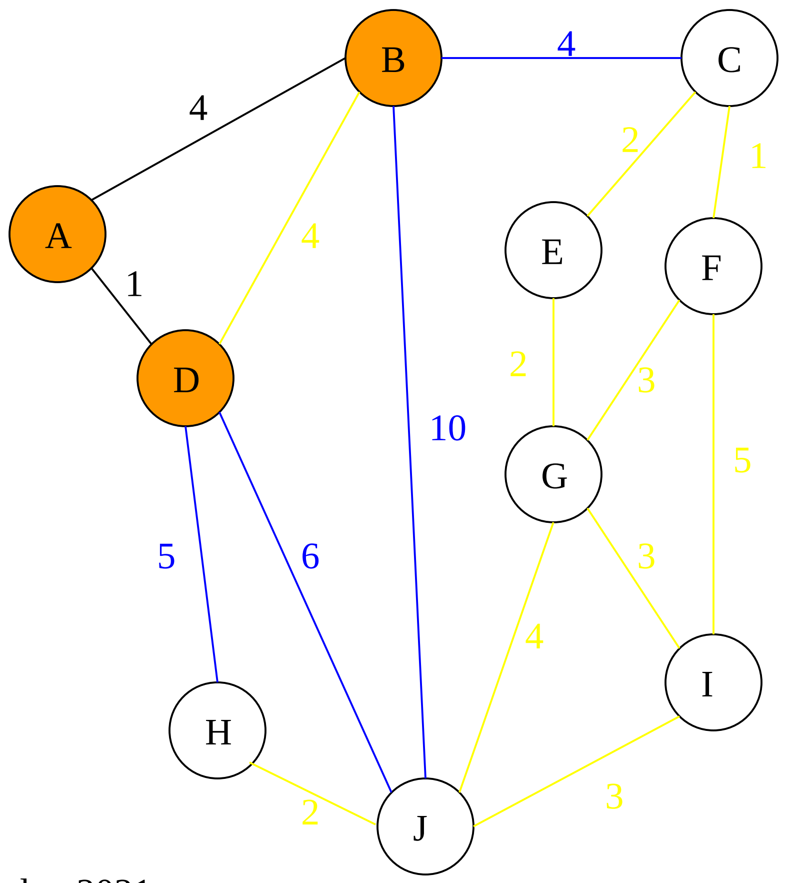


@Sudharshan Welihinda - 2021

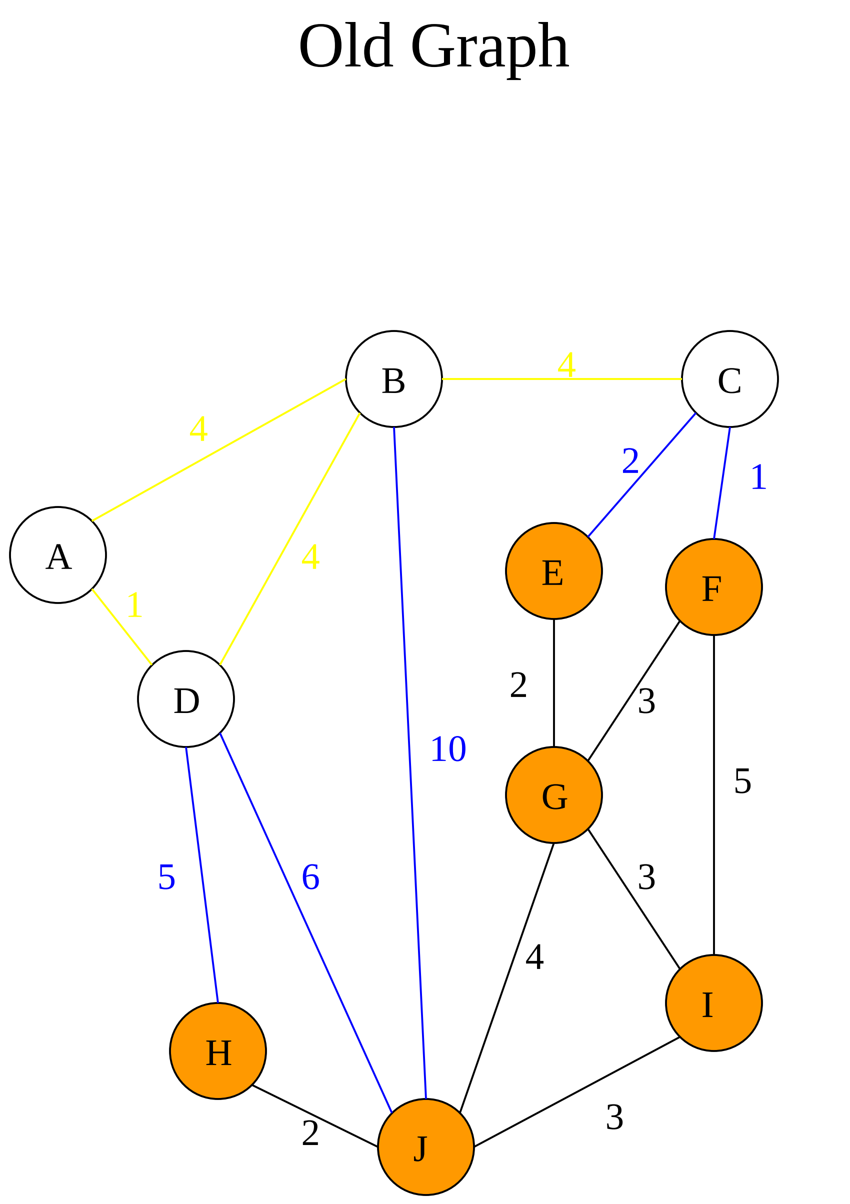
Old Graph

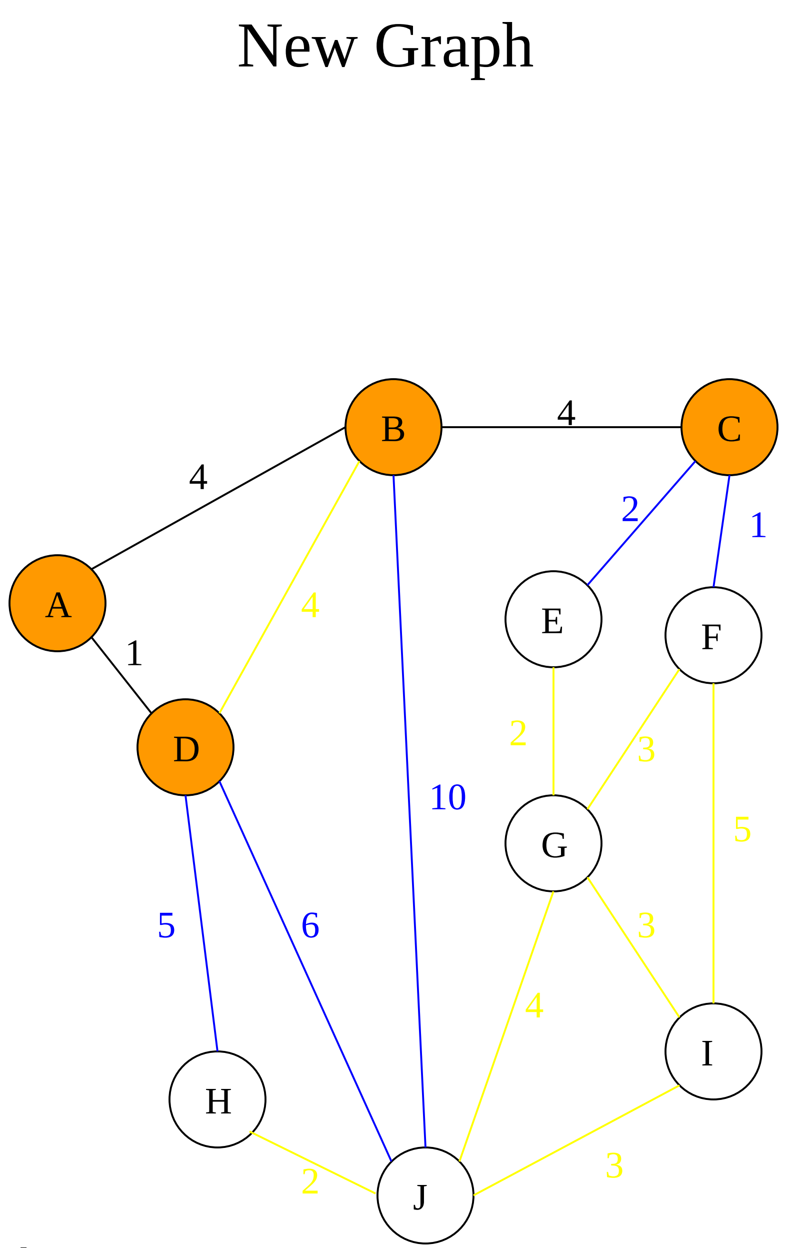


New Graph



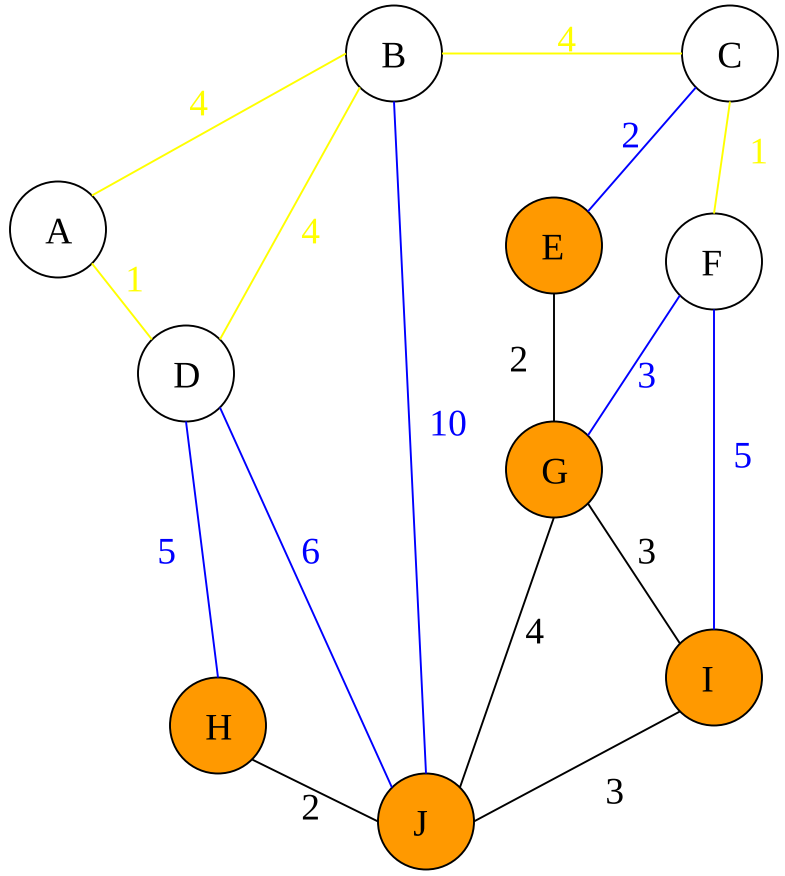
@Sudharshan Welihinda - 2021



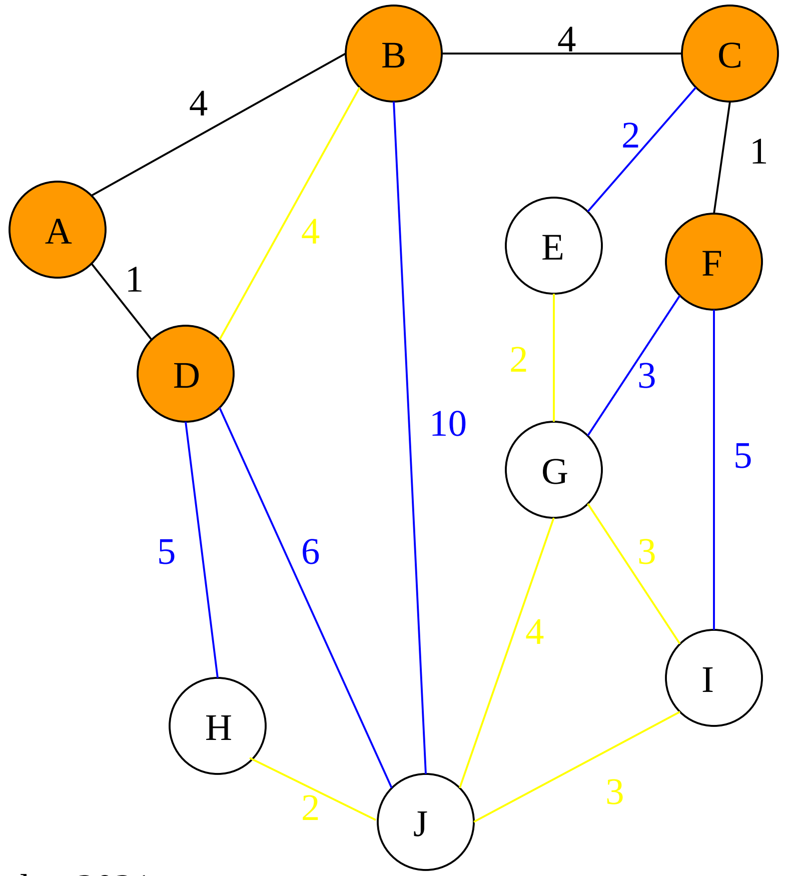


@Sudharshan Welihinda - 2021

Old Graph

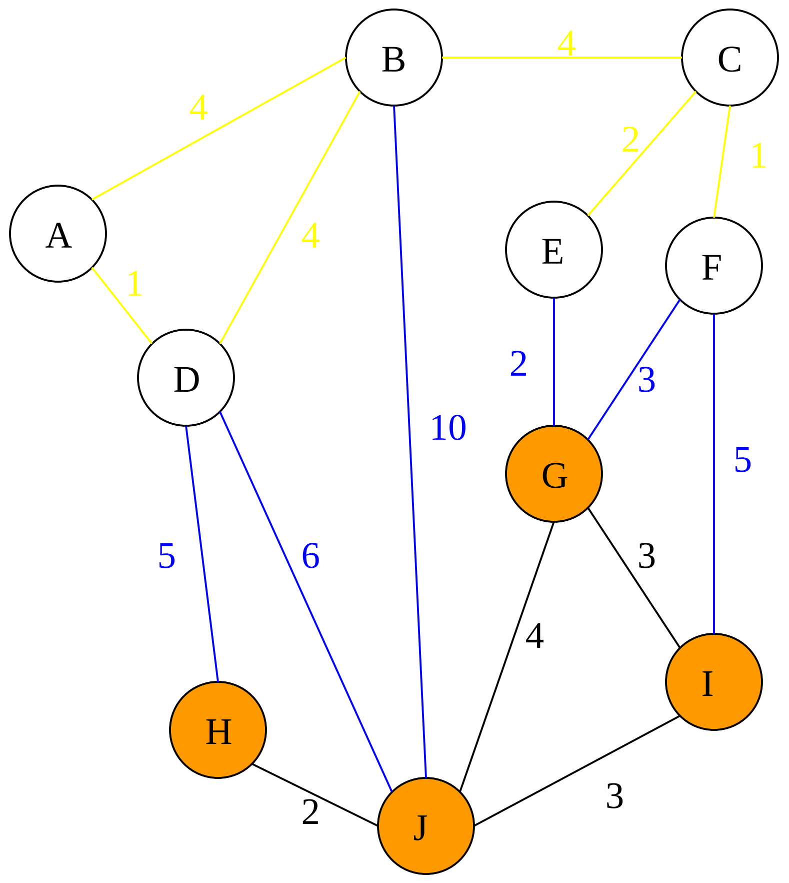


New Graph

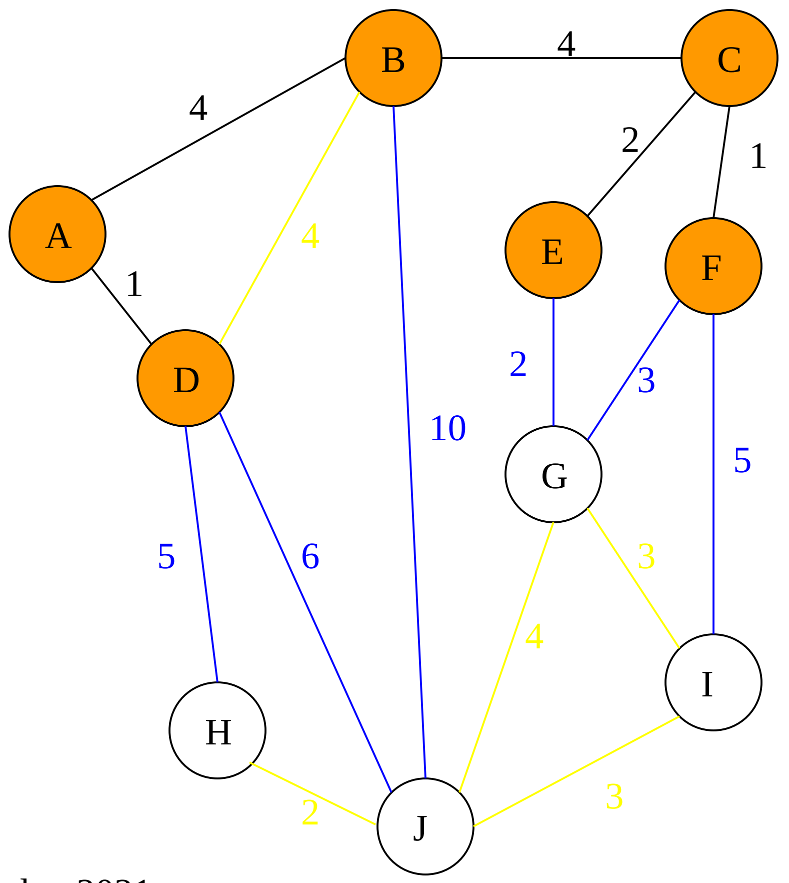


@Sudharshan Welihinda - 2021

Old Graph

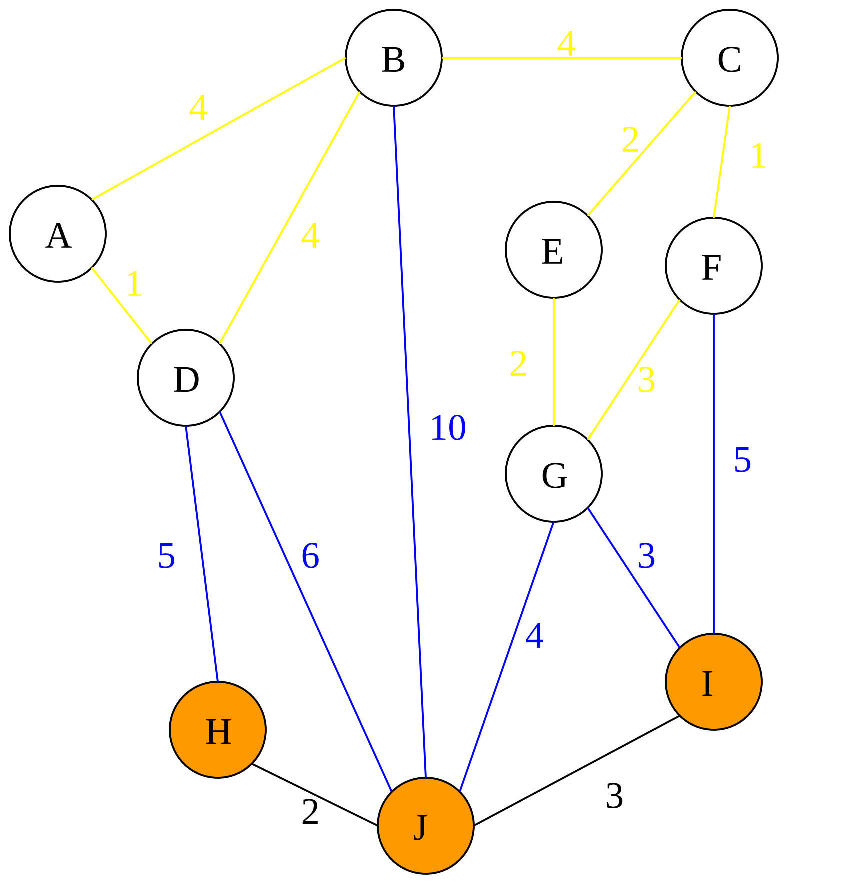


New Graph

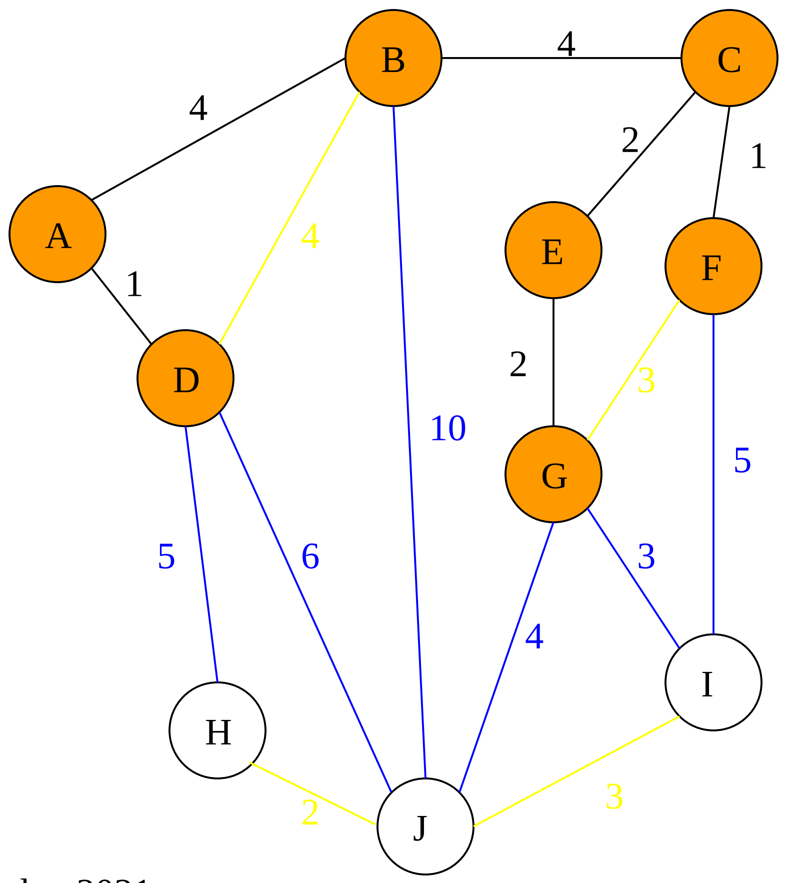


@Sudharshan Welihinda - 2021

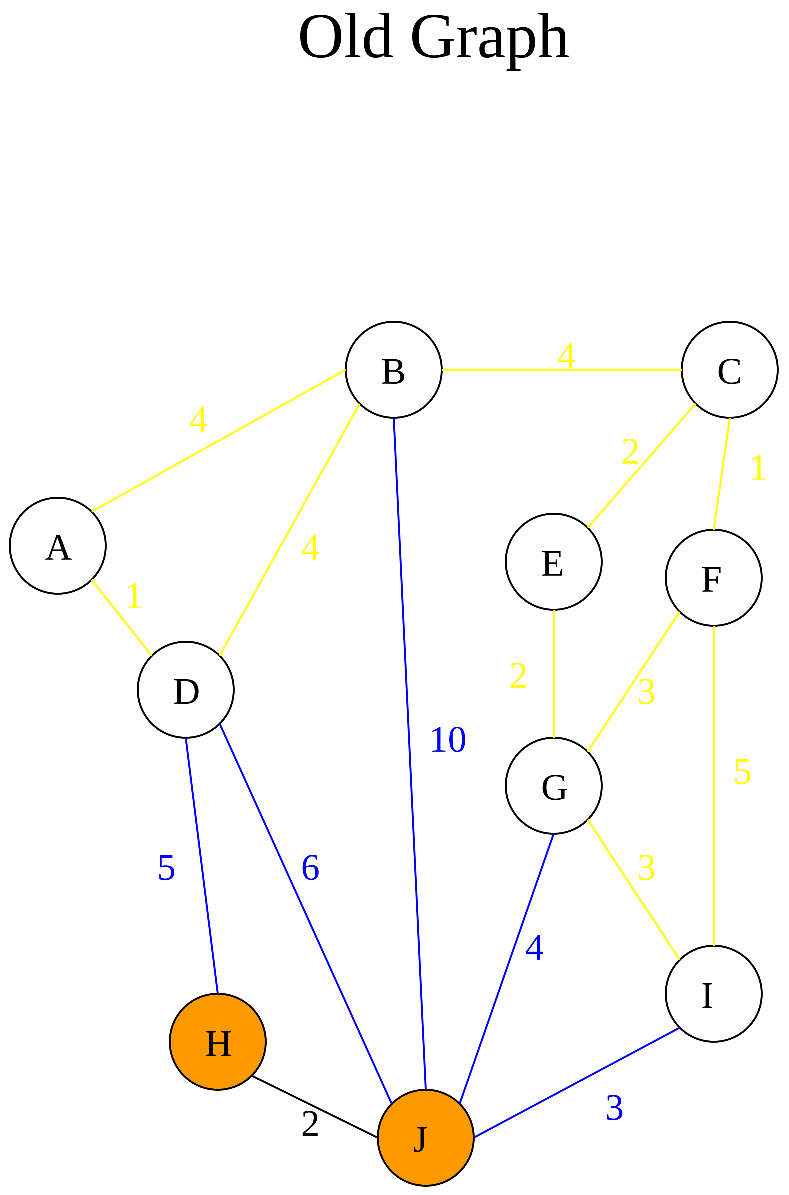
Old Graph

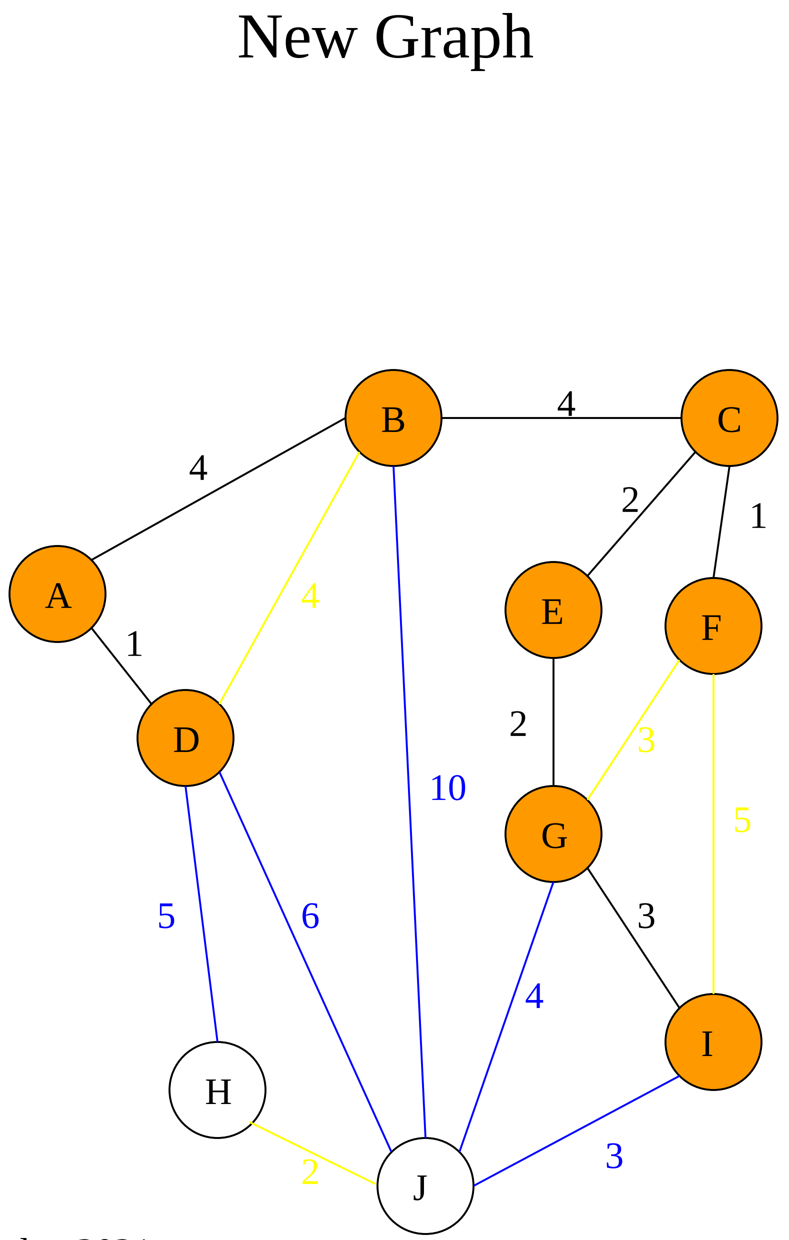


New Graph



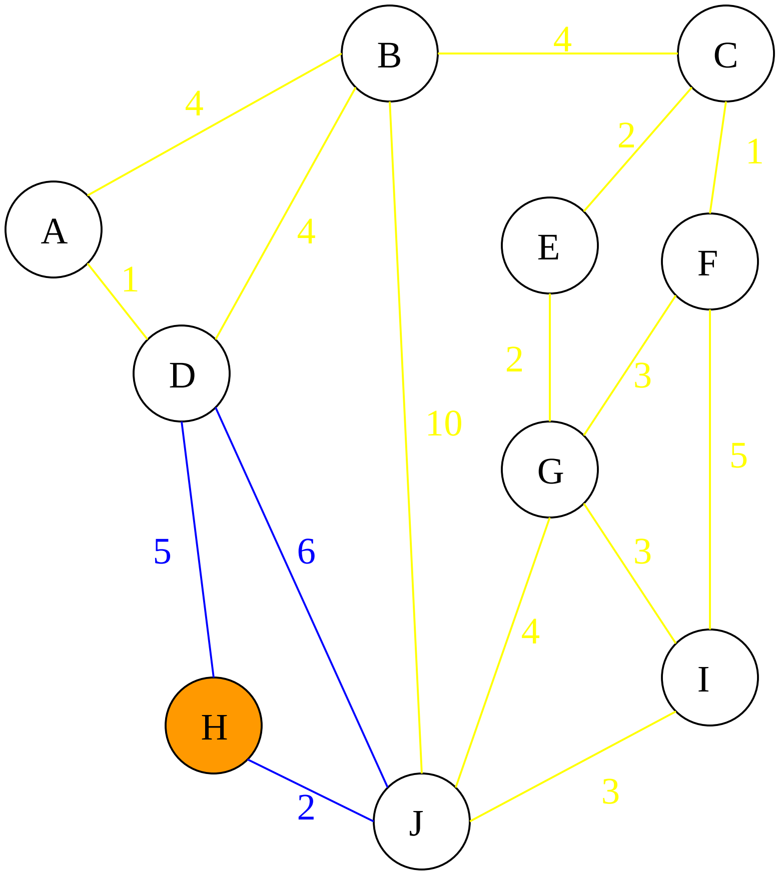
@Sudharshan Welihinda - 2021



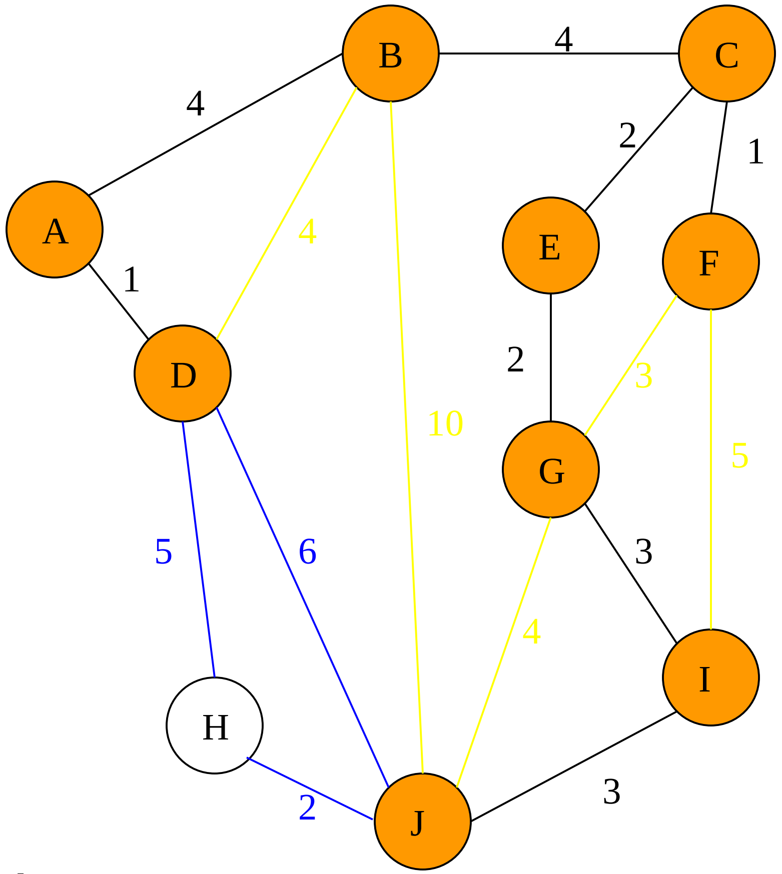


@Sudharshan Welihinda - 2021

Old Graph

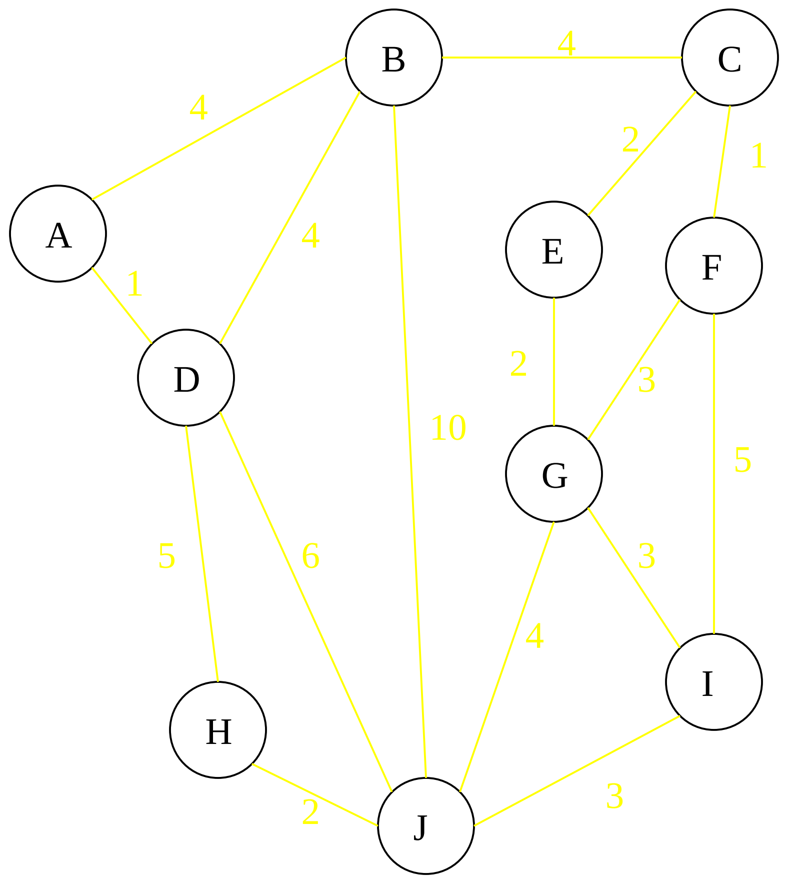


New Graph

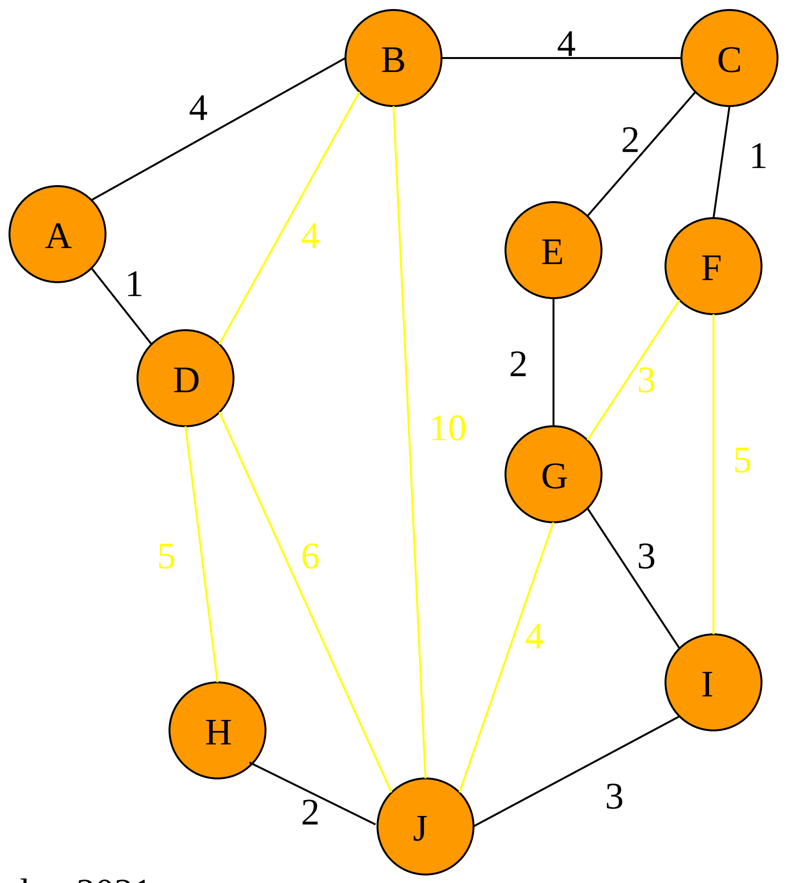


@Sudharshan Welihinda - 2021

Old Graph

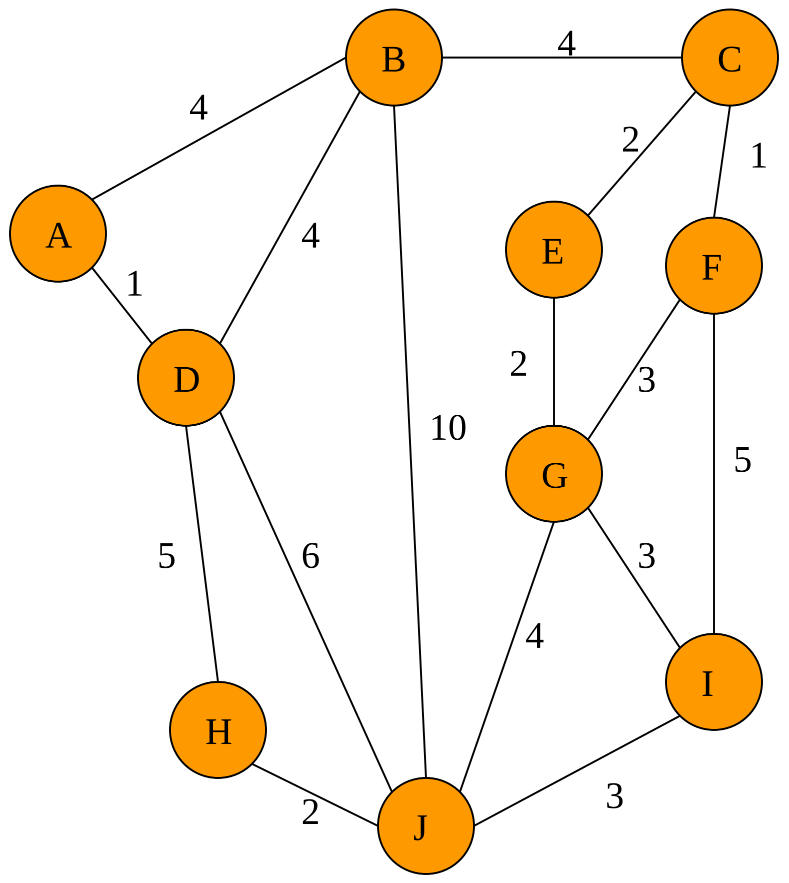


New Graph

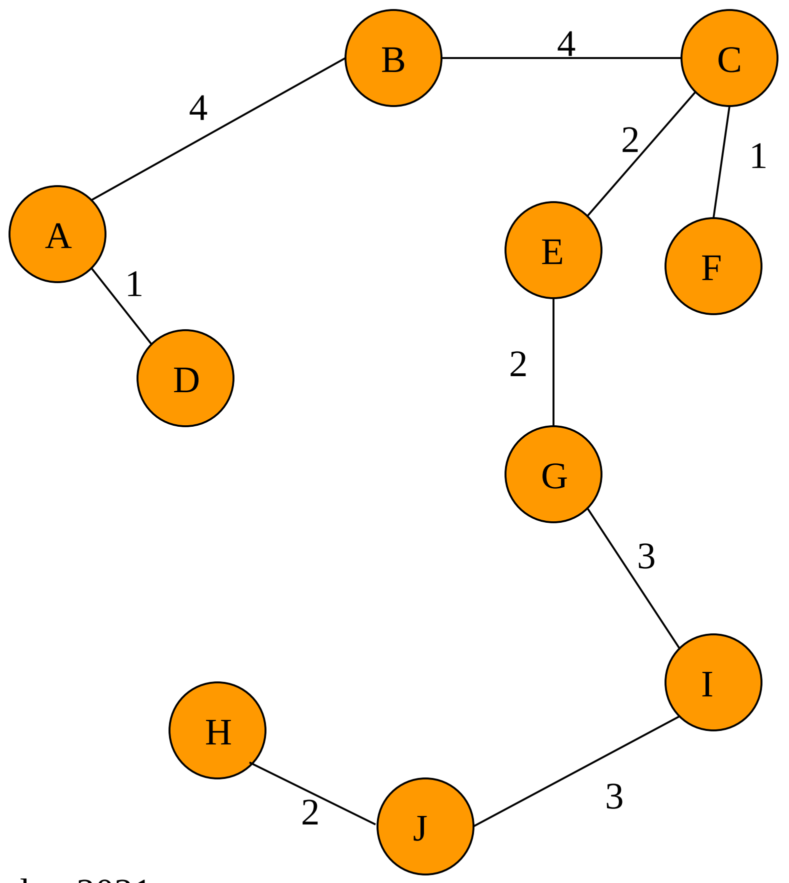


@Sudharshan Welihinda - 2021

Complete Graph



Minimum Spanning Tree

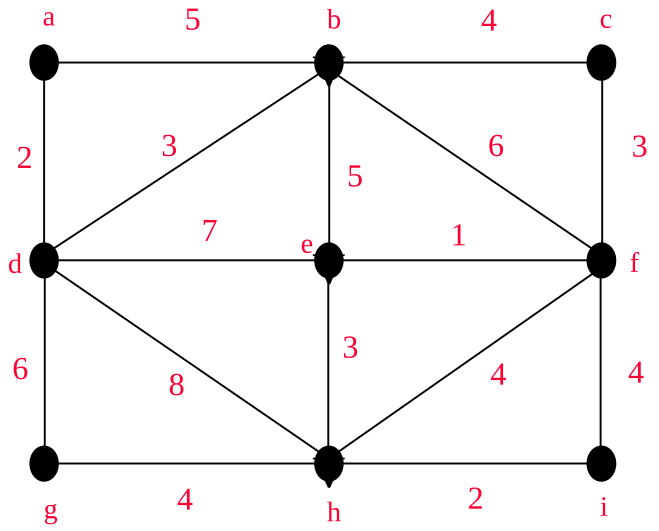


@Sudharshan Welihinda - 2021

Question

Given a Minimum spanning tree (MST) for an edge-weighted graph G, suppose that an edge in G, which does not disconnect G, is deleted. Describe how to find the MST of the new graph.

Question



Given the above graph, describe and apply Kruskal’s method in order to derive the minimum spanning tree (MST) from this graph? What will be the MST derived from the graph?

Exercise :

Let us assume that the following notation stands for times spent in minutes and connections among cities, represented by the letters A,B,C,D,E.

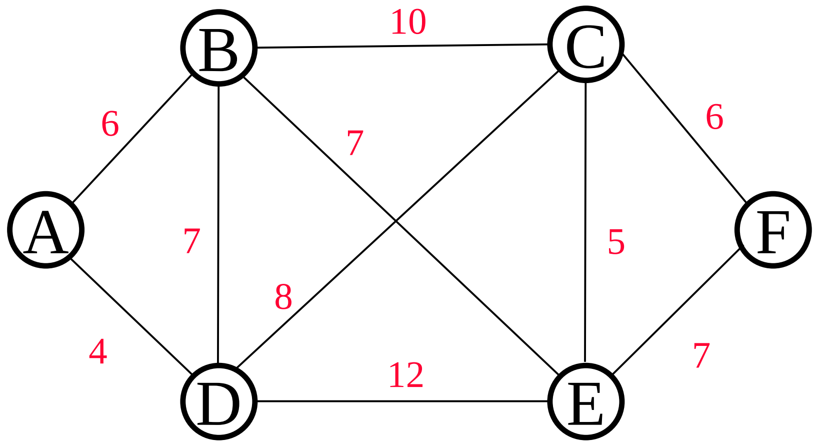
A->B: 50, A->D: 80, B->C: 60, B->D: 90, C->E: 40, D->C: 20, D->E: 70, E->B: 50

For instance, you can travel from city A to city B in 50 mnts. Given this knowledge, we need to figure out what is the travel path from city A to city E with the least time spent. Subsequently,

(a) Name the algorithm and data structure you should apply to resolve the problem
(b) Specify the path to follow and least time spent, in order to move from city A to city E.

(2014 May)

Exercise :



Let us assume that you are given the above graph representing the potential costs (weights on the edges) associated with laying out a cable TV network among different cities (nodes on the graph).

Suggest a network solution for connecting all cities at the minimum cost possible.

Justify your approach by briefly explaining the method (algorithmic approach) you followed.