Hope Foundation's
International Institute of Information Technology, Pune

Review of Faculty Orientation Workshop on Data Structures

Under the Aegis of BoS (E&TC), SPPU, Pune
SE E&TC/ Electronics) 2019 Course
(22nd to 26th June 2020)

Mrs. Vaishali Lele

Unit VI Graphs (06 Hrs)

- **Graph:** Basic Concepts & terminology.
- **Representation of graphs:** Adjacency matrix, Adjacency list.
- **Operations on graph:** Traversing a graph.
- **Spanning trees:** Minimum Spanning tree- Kruskal's Algorithm, Prim's Algorithm and Dijkstra's Shortest Path Algorithm.

Data Structures: CO-PO Mapping

Course Outcome	Blooms Taxonomy Level	After successful completion of the course students will be able to	Mapping with Syllabus Unit	PO MAPPING
CO204184.6	4	Apply the knowledge of graph for solving the problems of spanning tree and shortest path algorithm.	6	1, 2, 3, 4, 5, 9, 12

MAPPING	LEVEL	JUSTIFICATION
CO6 –PO1	2	Every Program is based on knowledge of mathematics, science and engineering fundamentals
CO6 –PO2	2	Design and debug the program using proper selection of data types and control structure to be carried out to obtain the specified solution with appropriate considerations.
CO6 –PO3	3	Selection of proper data structure is done based on given problem statement for formulating and analysing CEP.
CO6–PO4	2	Selection of proper data structures and algorithm is done based on given problem statement for formulating and analysing CEP.

Program Outcomes (POs)

1. Engineering knowledge
2. Problem analysis
3. Design/development of solutions
4. Conduct investigations of complex problems
5. Modern tool usage
6. The engineer and society
7. Environment and sustainability
8. Ethics
9. Individual and team work
10. Communication
11. Project management and finance
12. Life-long learning

Data Structures: CO-PO Mapping

Course Outcome	Blooms Taxonomy Level	After successful completion of the course students will be able to	Mapping with Syllabus Unit	PO MAPPING
CO204184.6	4	Apply the knowledge of graph for solving the problems of spanning tree and shortest path algorithm.	6	1, 2, 3, 4, 5, 9, 12

MAPPING	LEVEL	JUSTIFICATION
CO6 –PO5	3	Modern tools like turbo C, Codeblocks, GCC are used for development of programs.
CO6 –PO9	2	Development of algorithm using proper data structures may be divided into team and after the completion of entire code it could be integrated for the required final output.
CO6 –PO12	1	Integration and implementation of modular programs using proper algorithm and data structures will be useful throughout the life.

Program Outcomes (POs)

1. Engineering knowledge
2. Problem analysis
3. Design/development of solutions
4. Conduct investigations of complex problems
5. Modern tool usage
6. The engineer and society
7. Environment and sustainability
8. Ethics
9. Individual and team work
10. Communication
11. Project management and finance
12. Life-long learning

Data Structures: Books

TEXT BOOKS

T1: Ellis Horowitz, Sartaj Sahni, “Fundamentals of Data Structures”, Galgotia Books Source.
T2: Richard. F. Gilberg, Behrouz A. Forouzan, “Data Structures: A Pseudocode Approach with C,” Cengage Learning, second edition.

REFERENCE BOOKS

R1: Seymour Lipschutz, “Data Structure with C, Schaum’s Outlines”, Tata McGrawHill.
R2: E Balgurusamy, “Programming in ANSI C”, Tata McGraw-Hill, Third Edition.
R3: Yedidyah Langsam, Moshe J Augenstein, Aaron M Tenenbaum “Data structures using C and C++” PHI Publications, 2nd Edition.
R4: Reema Thareja, “Data Structures using C”, Second Edition, Oxford University Press, 2014

ADDITIONAL MATERIAL

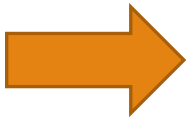
MOOC / NPTEL:

1. NPTEL Course “Programming & Data Structure” https://nptel.ac.in/courses/106/105/106105085/
2. NPTEL Course “Data Structure & Algorithms” https://nptel.ac.in/courses/106/102/106102064/

Data Structures: Topic – Book – Pages Mapping

Sr. No.	Topic	Reference / text book with page no.
UNIT VI: Graphs		
6.1	• Basic Concepts & terminology • Sequential representation of graphs : Adjacency matrix, Linked representation of a graph • Operations on graph • Traversing a graph	T2(481-490), R1(8.1-8.7,8.17-8.24)
6.2	Spanning trees; • Minimum Spanning tree • Kruskal's Algorithm • Prim's Algorithm • Dijkstra's Shortest Path Algorithm	R1(8.47-8.59)-Old Edition

Contents



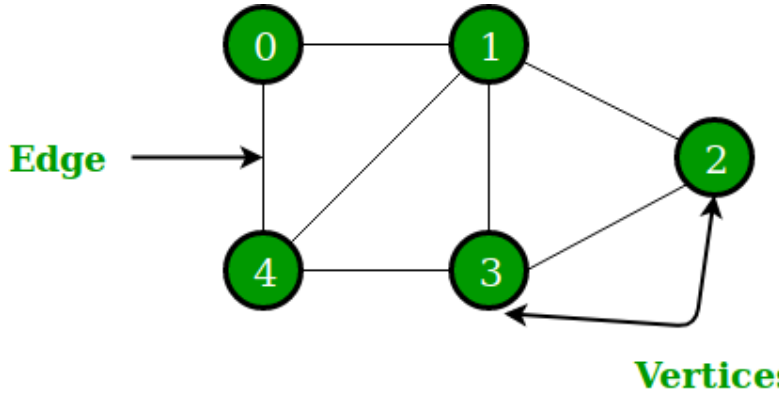
Basic concept

- A Graph is a non-linear data structure consisting of nodes and edges.

$$G = (V, E)$$

Where V = a set of **vertices** **and** E = a set of **edges**

- *A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of node*



Terminology

- **Vertex** – Each node of the graph is represented as a vertex.
- **Edge** – Edge represents a path between two vertices
- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge.
- **Path** – Path represents a sequence of edges between the two vertices.
- **Degree of a Node** is the number of edges the node is used to define

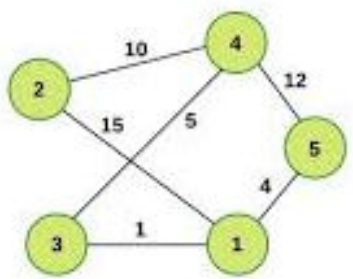
  Can also define **in-degree** and **out-degree**
  - In-degree: Number of edges pointing **to** a node
  - Out-degree: Number of edges pointing **from** a node

Terminology

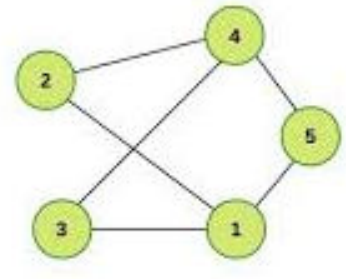
- **Cycle**: a path that starts and ends on the same vertex
- **Simple path**: a path that does not cross itself
- **Length** of a path: Number of edges in the path
- An *undirected* graph is **connected** if every pair of vertices has a path between it
- A *directed* graph is **strongly connected** if every pair of vertices has a path between them, in **both directions**

Types of Graphs:

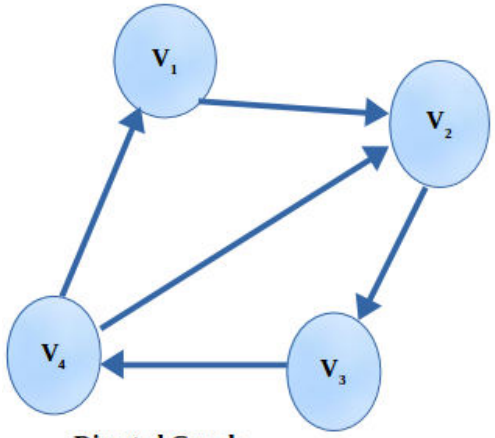
- Weighted or unweighted :
- Directed or undirected
- Cyclic or acyclic



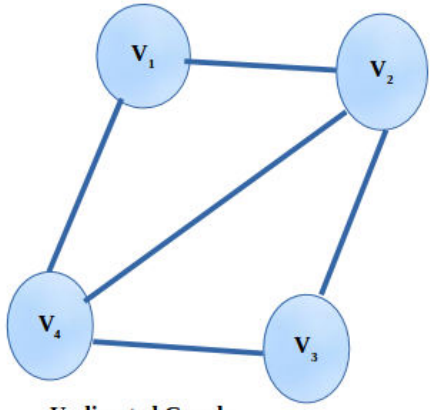
weighted graph



unweighted graph



Directed Graph



Undirected Graph

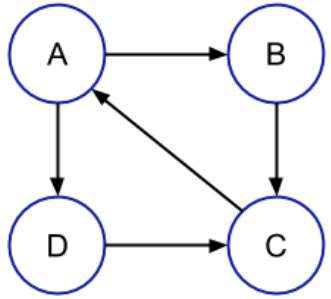


Figure: Cyclic Graph (also unweighted directed cyclic graph)

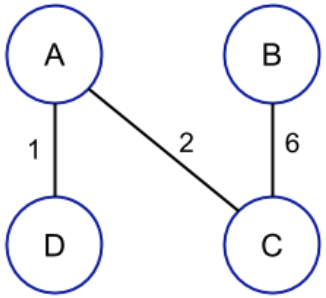


Figure: Acyclic Graph (also weighted undirected acyclic graph)

Types of Graphs

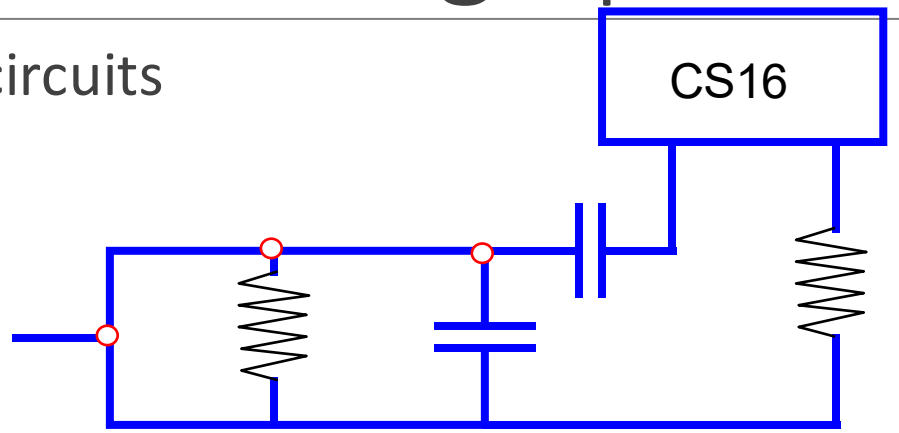
- **Weighted graph**: edges have a weight
  - Weight typically shows cost of traversing
- **Unweighted graph**: edges have no weight
  - Edges simply show connections
- **Undirected Graphs**: each edge can be traversed in **either direction**
- **Directed Graphs**: each edge can be traversed **only in a specified direction**
- A **Cyclic** graph contains cycles
- An **acyclic** graph contains no cycles

Applications of graphs

- To represent networks. The networks may include paths in a city or telephone network or circuit network.
- In social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, etc.
- **Electrical Engineering** – extensively used in designing circuit connections.
- **Computer Network** – The relationships among interconnected computers in the network follows the principles of graph theory.
- **Science** – The molecular structure and chemical structure of a substance, the DNA structure of an organism, etc., are represented by graphs.
- **Linguistics** – The parsing tree of a language and grammar of a language uses graphs.

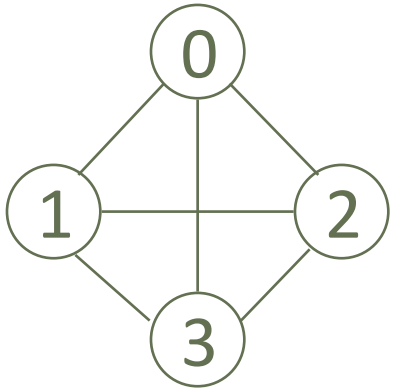
Applications of graphs

- Electronic circuits



- Networks (roads, flights, communications)

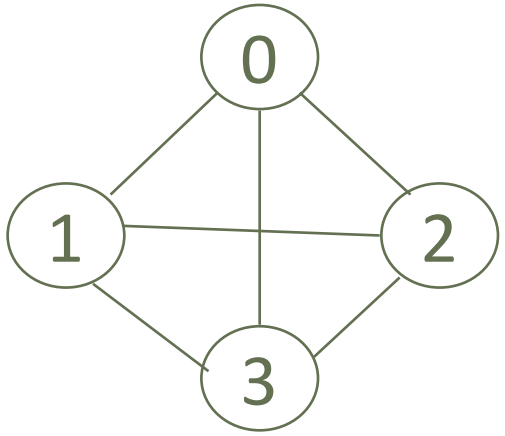
Adjacency Matrix

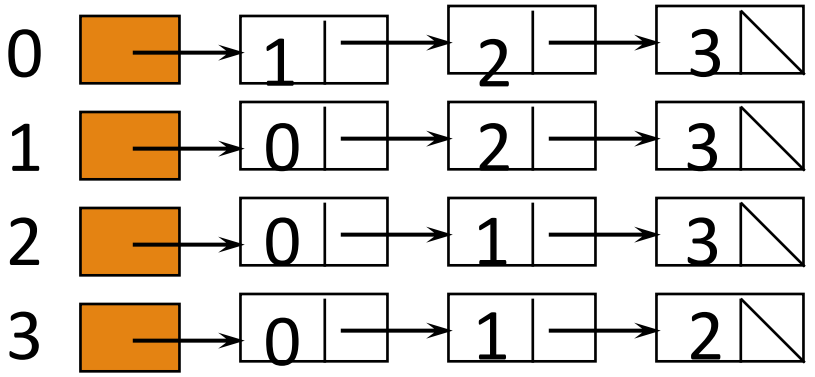


$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Adjacency Lists





Operations on Graphs:

- Add/Remove Vertex :
- Add/Remove Edge
- Traverse a graph

Insertion and deletion of nodes and edges in a graph is implemented by using adjacency matrix or adjacency list

Traversing a graph

- Visit the vertices of a graph in some specific order based on the graph's topology
- Graph traversal algorithms typically begin with a start vertex and attempt to visit the remaining vertices from there.
- Graph traversals must deal with a number of troublesome cases.

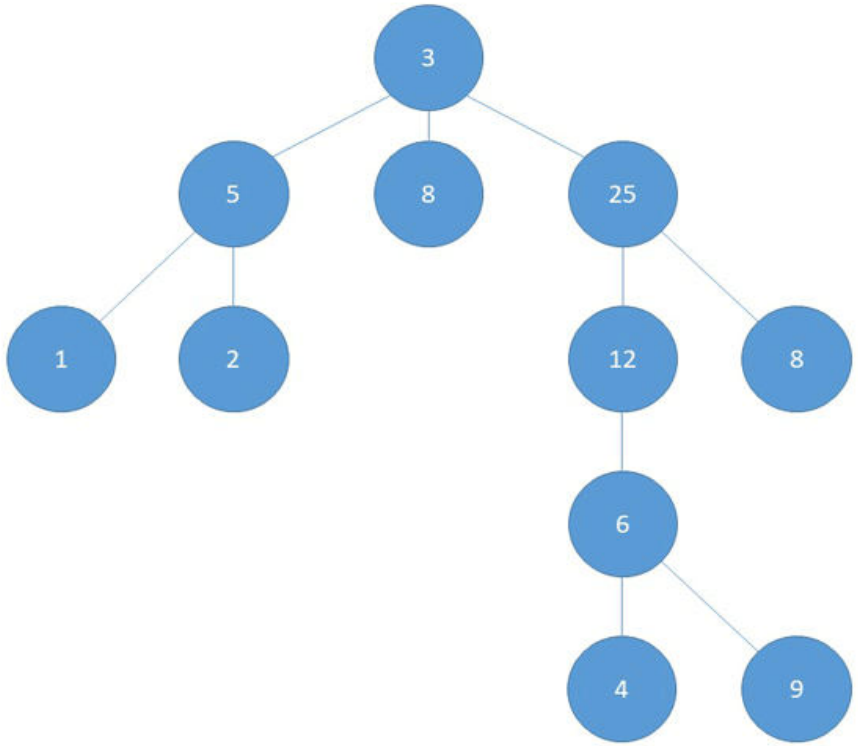
1. It might not be possible to reach all vertices from the start vertex. This occurs when the graph is not connected.
2. The graph might contain cycles, and we must make sure that cycles do not cause the algorithm to go into an infinite loop.

Traversal Methods

1) Breadth First Search(BFS)

2) Depth First Search(DFS)

Example of BFS




```
static void BFS(Graph G, int v)
{ LQueue Q = new LQueue(G.nodeCount());
Q.enqueue(v);
G.setValue(v, VISITED);
while (Q.length() > 0)
{ // Process each vertex on Q
 v = (Integer)Q.dequeue();
PreVisit(G, v);
int[] nList = G.neighbors(v);
for (int i=0; i< nList.length; i++)
if (G.getValue(nList[i]) != VISITED)
{ // Put neighbors on Q
G.setValue(nList[i], VISITED); Q.enqueue(nList[i]);
}
PostVisit(G, v);
} }
```

Properties of BFS

Notation ***Gs: connected component of s***

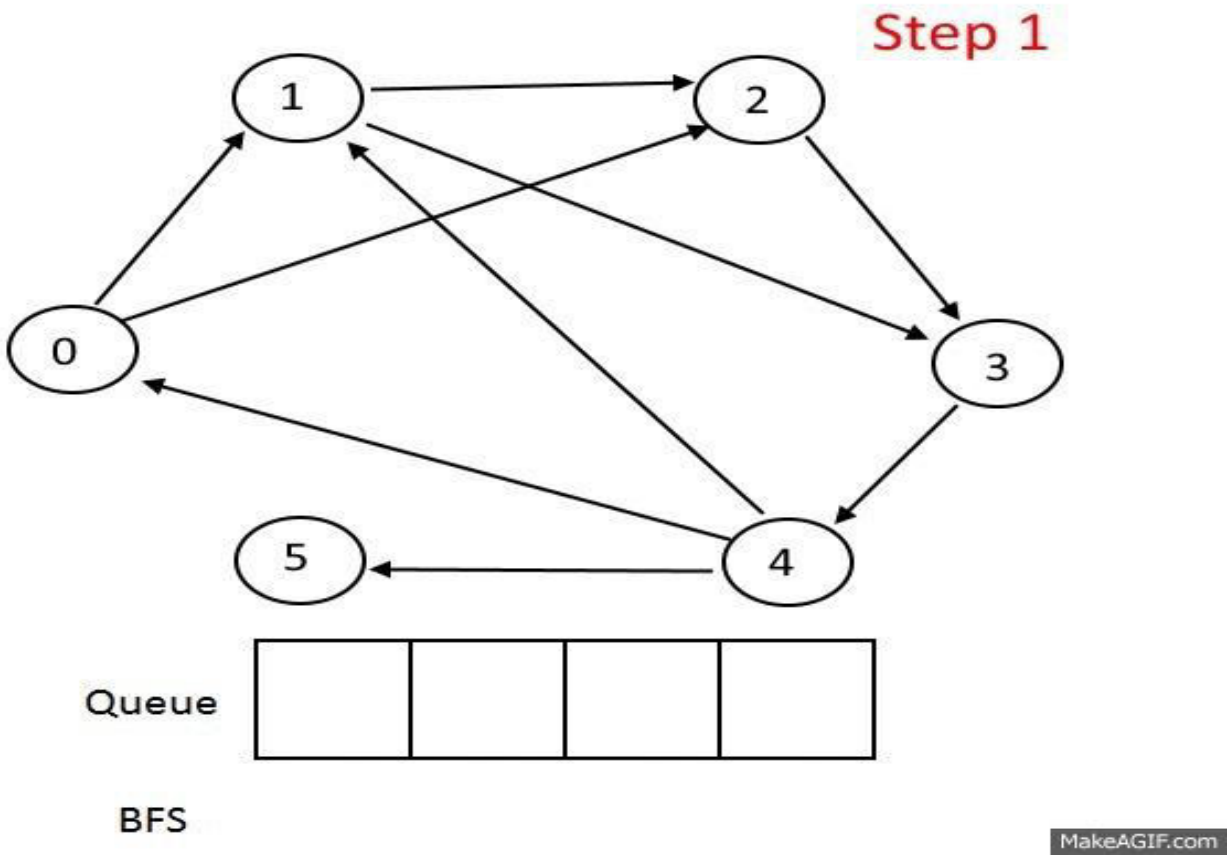
Property 1: ***BFS(G, s) visits all the vertices and*** edges of ***Gs***

Property 2 : The discovery edges labelled by ***BFS(G, s) form a spanning tree Ts*** of ***Gs***

Property 3 : For each vertex ***v in Li***

- The path of ***Ts from s to v has i*** edges
- Every path from ***s to v in Gs has at*** least ***i edges***

BFS



Analysis

Setting/getting a vertex/edge label takes ***O(1) time***

Each vertex is labelled twice

1. once as UNEXPLORED 2. once as VISITED

Each edge is labelled twice

1. once as UNEXPLORED 2. once as DISCOVERY or CROSS

Each vertex is inserted once into a sequence ***Li***

Method incidentEdges is called once for each vertex BFS runs in ***O(n + m) time provided the graph is*** represented by the adjacency list structure

Iterative BFS pseudocode

create a queue Q

mark v as visited and put v into Q

while Q is non-empty

remove the head u of Q

mark and enqueue all (unvisited) neighbours of u

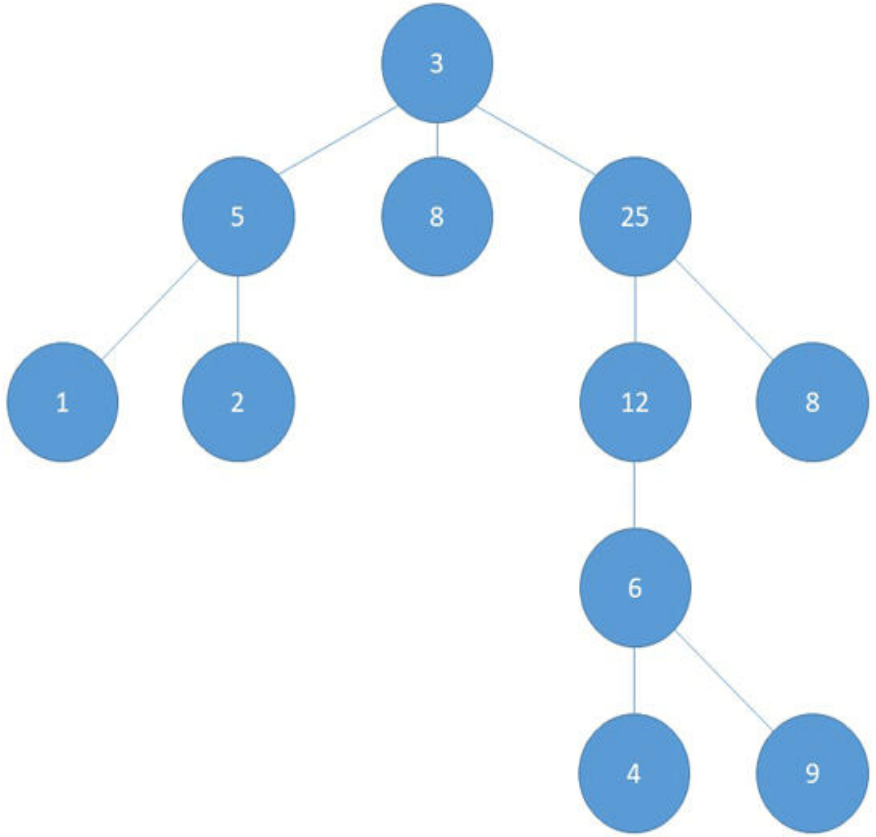
BFS Algorithm Complexity

- The time complexity of the BFS algorithm is O(V + E), where V is the number of nodes and E is the number of edges.
- The space complexity of the algorithm is O(V).

BFS Algorithm Applications

1. For GPS navigation
2. Path finding algorithms
3. In Ford-Fulkerson algorithm to find maximum flow in a network
4. Cycle detection in an undirected graph
5. In minimum spanning tree

Example of DFS




```
static void DFS(Graph G, int v)
{ PreVisit(G, v);
  G.setValue(v, VISITED);
   int[] nList = G.neighbors(v);
  for (int i=0; i< nList.length; i++)
      if (G.getValue(nList[i]) != VISITED)
DFS(G, nList[i]); PostVisit(G, v);
 }
```

Properties of DFS

Property 1 : ***DFS(G, v) visits all the*** vertices and edges in the connected component of ***v***

Property 2 : The discovery edges labelled by ***DFS(G, v)*** form a spanning tree of the connected component of ***v***

DFS pseudocode (recursive implementation)

```
DFS(G, u)
   u.visited = true
    for each v ∈ G.Adj[u]
      if v.visited == false
      DFS(G,v)
 init()
 { For each u ∈ G
      u.visited = false
      For each u ∈ G
DFS(G, u) }
```

DFS Algorithm (iterative implementation)

1. Create a stack of nodes and visited array.
2. Insert the root in the stack.
3. Run a loop till the stack is not empty.
4. Pop the element from the stack and print the element.
5. For every adjacent and unvisited node of current node, mark the node and insert it in the stack.

Analysis of DFS

Setting/getting a vertex/edge label takes ***O(1) time***

Each vertex is labelled twice

1. once as UNEXPLORED 2. once as VISITED

Each edge is labeled twice

1. once as UNEXPLORED 2. once as DISCOVERY or BACK

Method incidentEdges is called once for each vertex

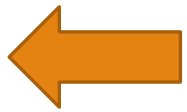
DFS runs in ***O(n + m) time provided the graph is*** represented by the adjacency list structure

DFS Algorithm Complexity

- The time complexity of the DFS algorithm is O(V + E), where V is the number of nodes and E is the number of edges.
- The space complexity of the algorithm is O(V).

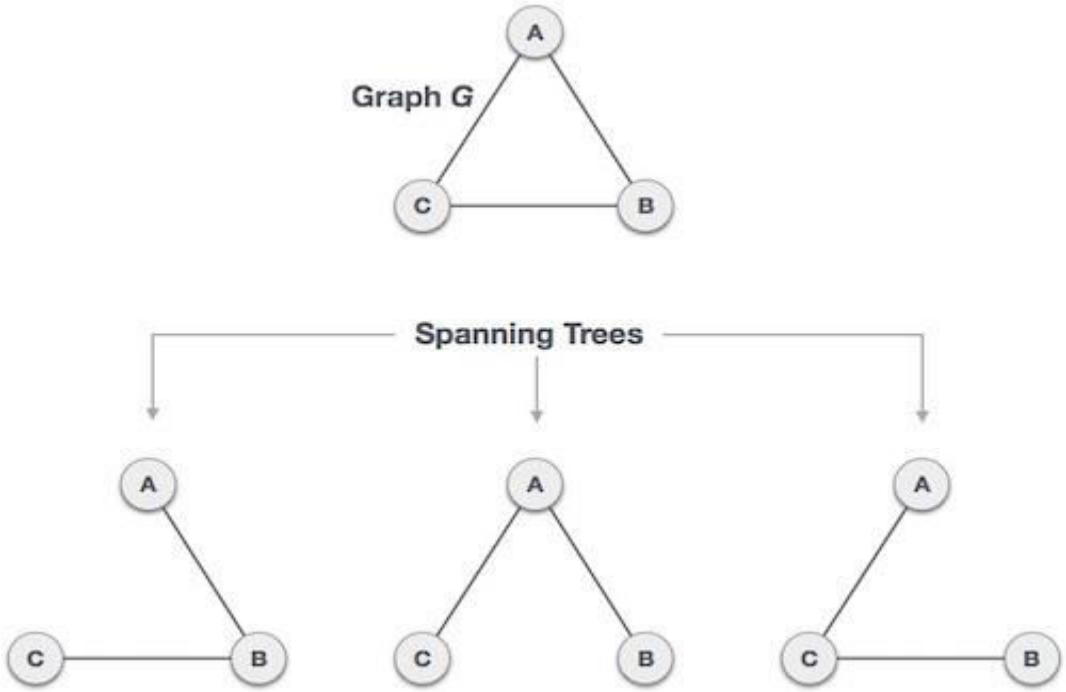
DFS Algorithm Applications

1. For finding the path
2. For finding the strongly connected components of a graph
3. For detecting cycles in a graph



Spanning Tree

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..



General Properties of Spanning Tree

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

Mathematical Properties of Spanning Tree

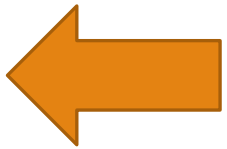
- Spanning tree has **n-1** edges, where **n** is the number of nodes (vertices).
- A complete graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees.

Application of Spanning Tree

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis

Minimum Spanning Tree (MST)

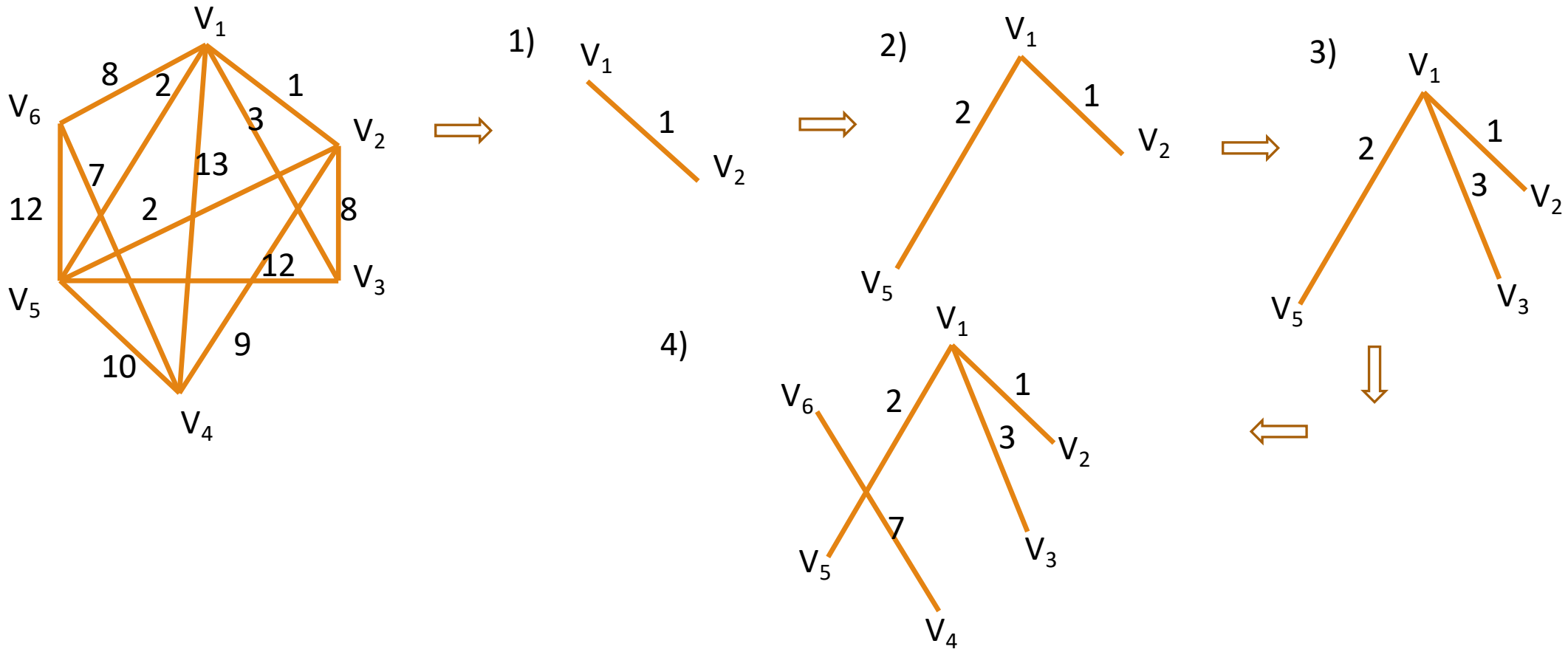
- In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.
- In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.



Kruskal's Algorithm

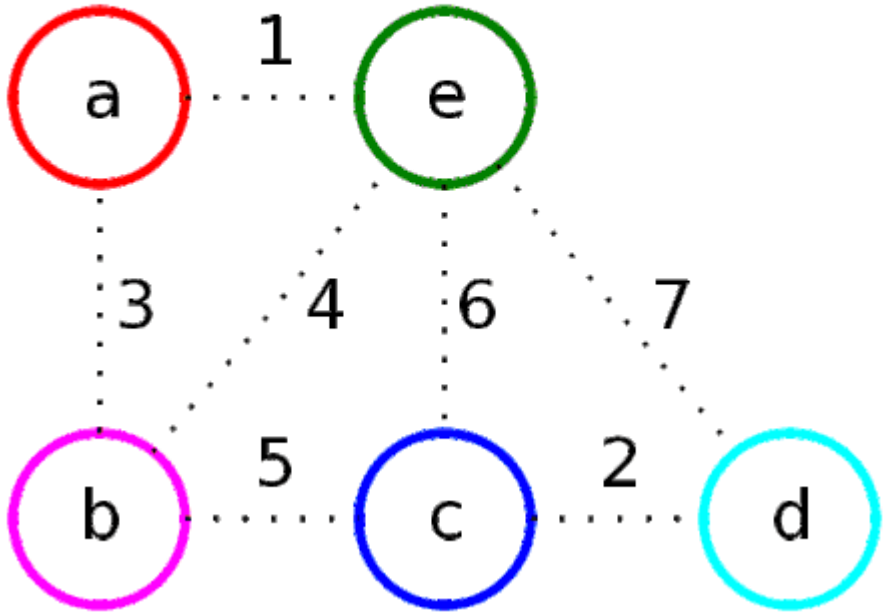
- Edge with minimum weight is selected.
- Cycle should not be formed.
- Each time the edge of minimum weight has to be selected.
- It is not necessary to have edges of minimum weights to be adjacent.

Example of Kruskal's Algorithm



Kruskal's Algorithm

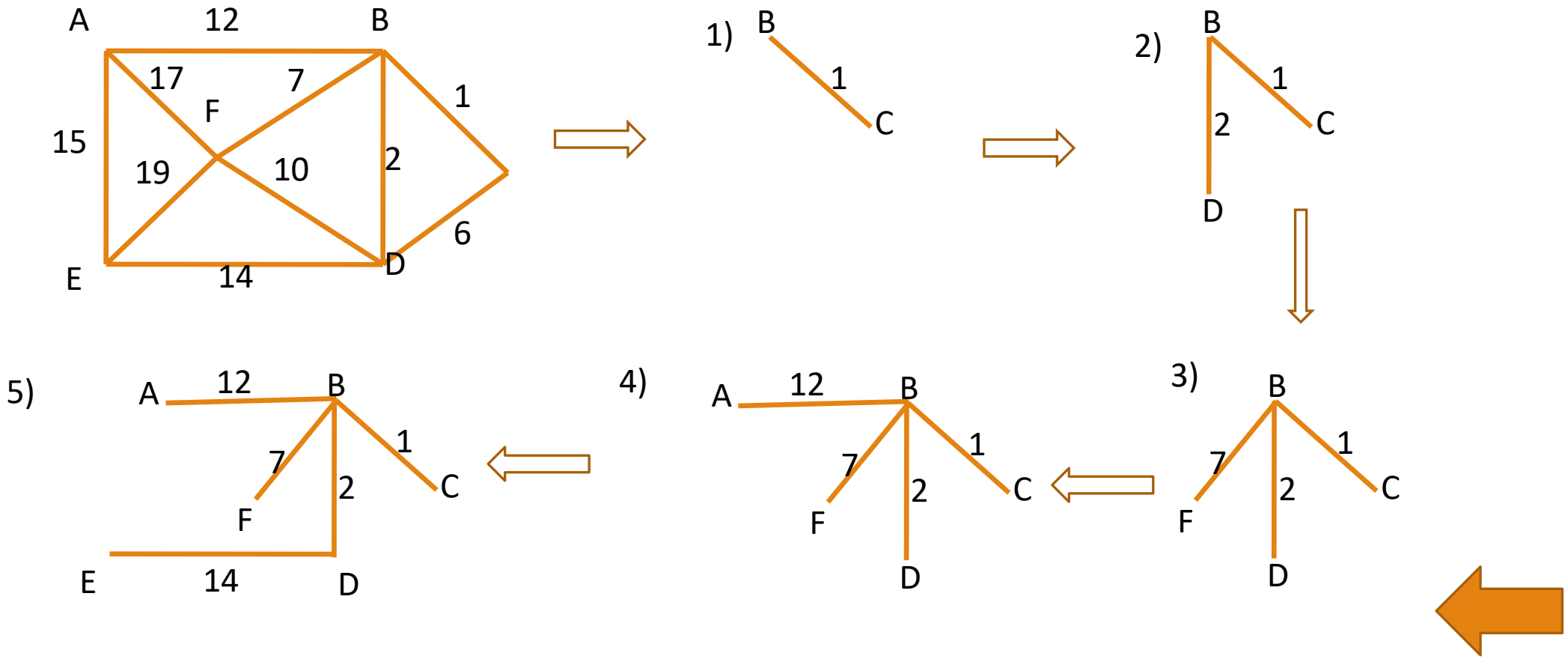
Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Prim's Algorithm

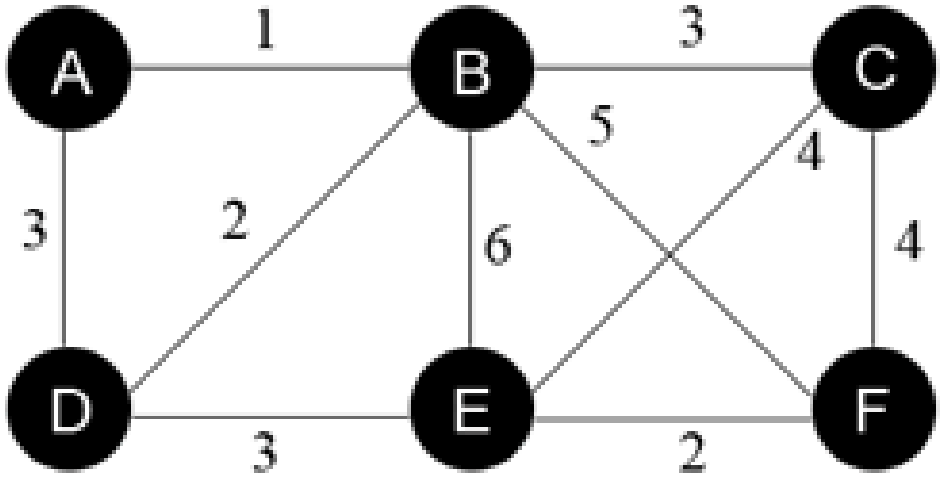
- Edge with minimum weight is selected
- Select edge with minimum weight adjacent to these vertices.
- Process will continue till all the vertices are included.
- Cycle should not be formed.

Example:


A
12
B
17
7
F
1
15
19
10
2
6
E
14
D
1)
B
1
C
2)
B
1
2
C
D
3)
B
7
1
F
2
C
D
4)
A
12
B
7
1
F
2
C
D
5)
A
12
B
7
1
F
2
C
E
14
D

Prim's Algorithm

SET: { }

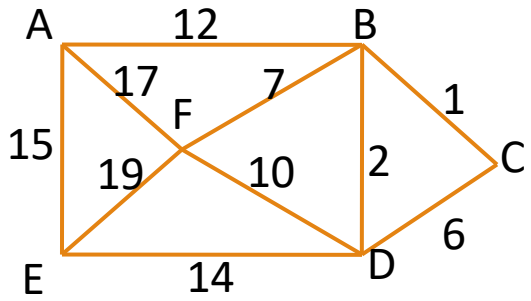


Dijkstra Algorithm

- Start finding the distances from source node and find all the paths from it to adjacent nodes.
- Select minimum distance from paths available.

Example:

Find out the shortest path from A to C using Dijkstra algorithm.



Source Node = A
Destination Node = C

dist(A,B) =12 minimum
dist(A,C) = ∞
dist(A,D)= ∞
dist(A,E) = 15
dist(A,F) = 17

dist(A,C) = min{∞, {d(A,B) + d(B,C)}}
= min{∞, (12+1)}
= min{∞,13}
= 13

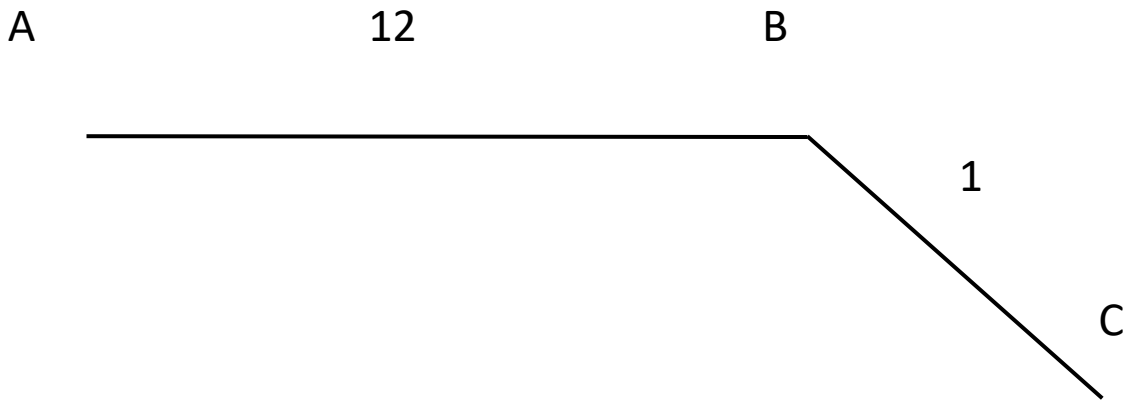
dist(A,C) = min{∞, {d(A,E)+d(E,D)+d(D,C)}}
= min{∞,{15 + 14 + 6}}
=min {∞,35}
= 35

dist(A,C) = min{∞, {d(A,F)+d(F,D)+d(D,C)}}
= min{∞, {17+10+6}}
= min{∞,33}
= 33

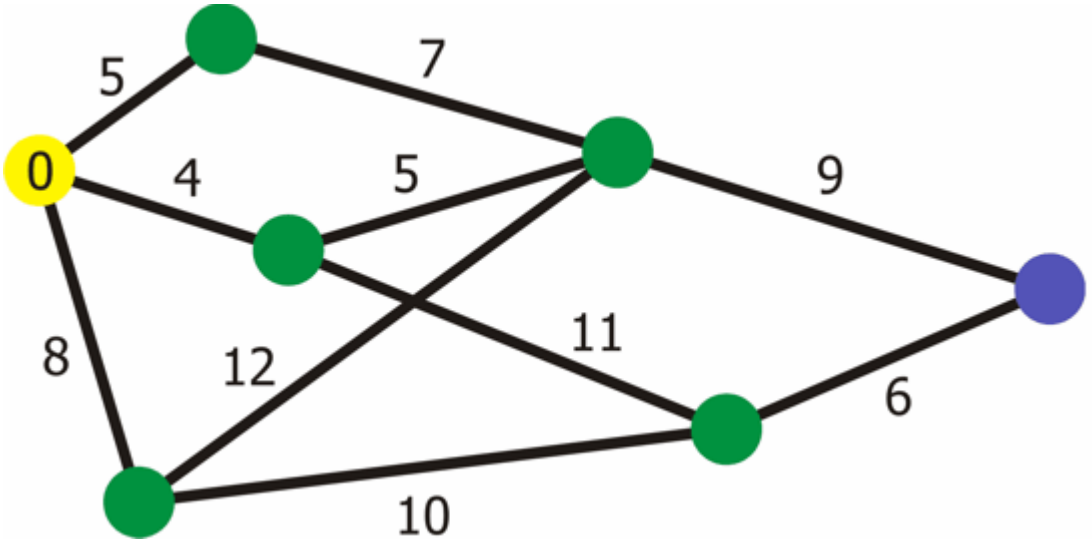
dist(A,C) =min{∞,{d(A,B)+d(B,D)+d(D,C)}}
= min{∞,{12+2+6}}
= min{∞,20}
=20

Example:

Among all these paths 13 is minimum distance. So shortest path from A to C is 13 which is shown as follows:



Dijkstra Algorithm



Questions :

Q. 1. In each of the following examples, please choose the best data structure(s). Options are: Array, Linked Lists, Stack, Queue, Tree, Graph. Justify your answer.

a) You have to store social network “feeds”. You do not know the size, and things may need to be added as and when it is required.

b) To store the customer order information in a drive-in burger place. (Customers keep on coming and they have to get their correct food at the payment/food collection window.)

c) To implement back functionality in the internet browser.

d) To store the possible moves in a chess game.

Q. 2 Can Prim's and Kruskal's algorithm yield different minimum spanning trees? Explain why or why not.

Questions:

Q.3 Consider a simple undirected weighted graph G(V, E) with 10 vertices and 45 edge, assume (u, v) are two vertices weight of a edge is =4lu- vl then the minimum cost of the spanning tree of G_ ?

Answer: The given undirected graph is a complete graph on 10 vertices, so no. of edges in Minimum spanning tree = 9

An edge is included in MST if it's cost is less than or equal to the weights of the remaining edges and doesn't form a cycle,

Now, the minimum possible weight is 4 because it is a simple graph, so loops are not allowed.

Now, if we consider edges,(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10),we obtain the required 9 edges each with a cost of 4

So, Total cost in MST = 9*4 = 36

Questions:

Q.4 Consider an undirected random graph of eight vertices. The probability that there is an edge between a pair of vertices is 1/2. What is the expected number of unordered cycles of length three?

(A) 1/8
(B) 1
(C) 7
(D) 8

Data Structures

Suggestions are Welcome!

Stay safe and healthy and happy

Enjoy the new version of teaching