DATA STRUCTURES & ALGORITHMS

Graph

Instructor: Engr. Laraib Siddiqui

Introduction

A graph G is consist of two things:

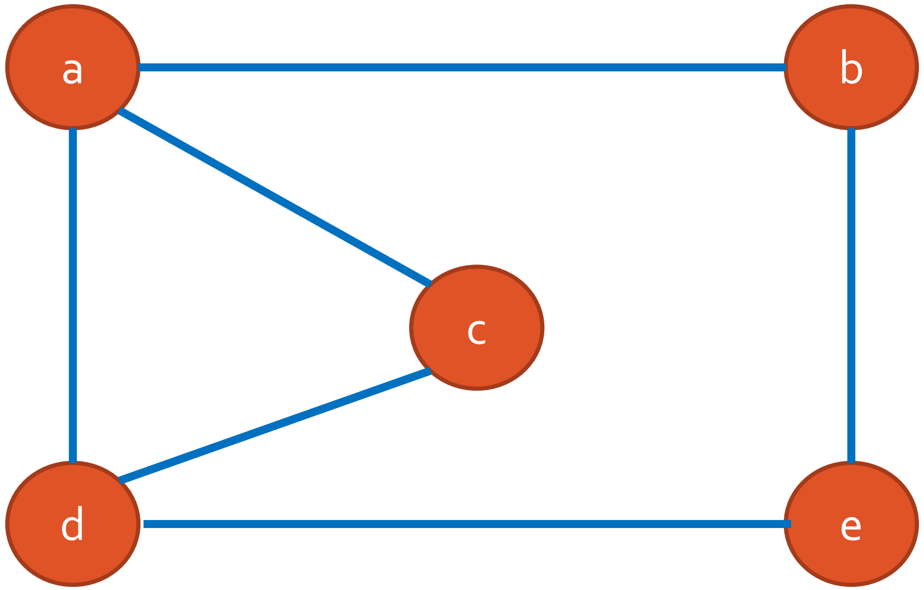
- A set V of elements called nodes/vertices.
- A set of E of edges such that each edge in E is identified with a unique pair {u, v} of nodes in V.

Graph can be indicated as G = (V, E)

Example

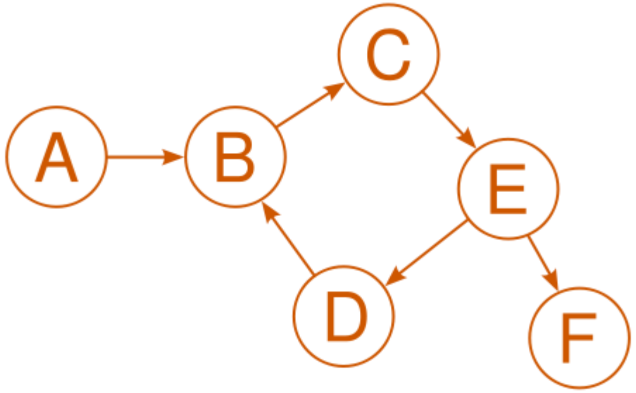
V = {a, b, c, d, e}

E = {(a,b), (a,c), (a,d), (b,e), (c,d), (c,e), (d,e)}

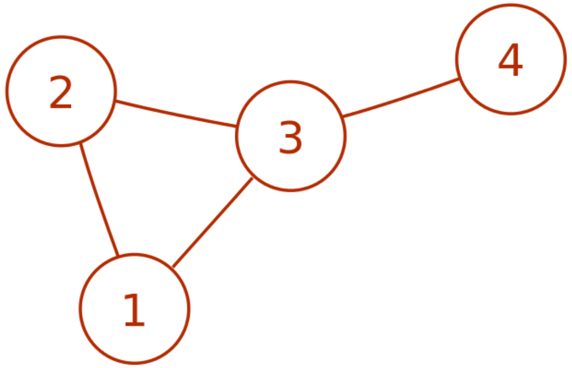


Types

- Directed
  - When edges have direction

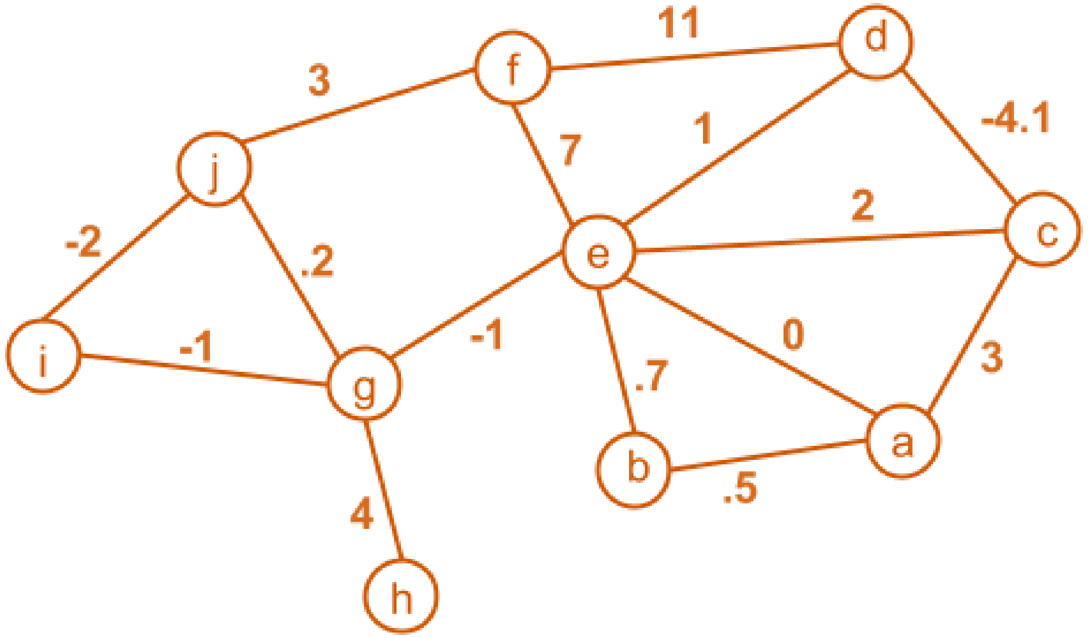


- Undirected
  - When edges have no direction



Weighted Graph

A graph in which each edge contains a value



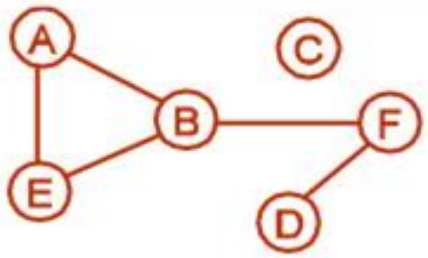
Key Terms

- **Adjacent nodes**: two nodes are adjacent if they are connected by an edge
- **Path**: sequence of vertices that connect two nodes in a graph
- **Complete graph**: in which every vertex is directly connected to every other vertex
- **Multi Graph:** in which numerous edges between a pair of vertices in a graph G= (V, E) occurs. There are no self-loops.
- **Cycle**: is a simple path with the same start and end vertex
- **Degree of a vertex**: number of edges that touch it
- **In degree**: no of incoming edges
- **Out degree**: no of outgoing edges

Graph Representation

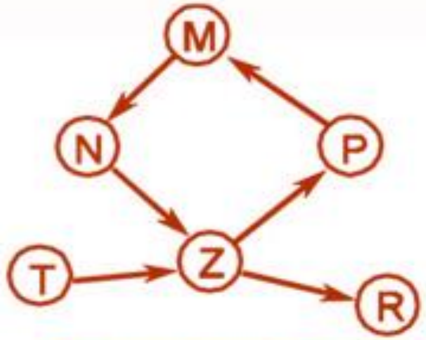
- Adjacency matrix representation
  - Square grid of Boolean values
  - If the graph contains N vertices then the grid contains N rows N columns
  - For two vertices numbered I and J the element at row I and column J is true if there is an edge from I to J, otherwise false
- Adjacency lists representation
  - A graph of n nodes is represented by a one dimensional array L of linked lists, where
    - L[i] is the linked list containing all the nodes adjacent from node i.
    - The node in the list L [i] are in no particular order

Adjacency Matrix Representation



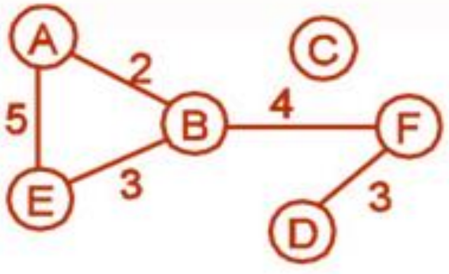
	A	B	C	D	E	F
A	0	1	0	0	1	0
B	1	0	0	0	1	1
C	0	0	0	0	0	0
D	0	0	0	0	0	1
E	1	1	0	0	0	0
F	0	1	0	1	0	0

Undirected Graph



	M	N	P	R	T	Z
M	0	1	0	0	0	0
N	0	0	0	0	0	1
P	1	0	0	0	0	0
R	0	0	0	0	0	0
T	0	0	0	0	0	1
Z	0	0	1	1	0	0

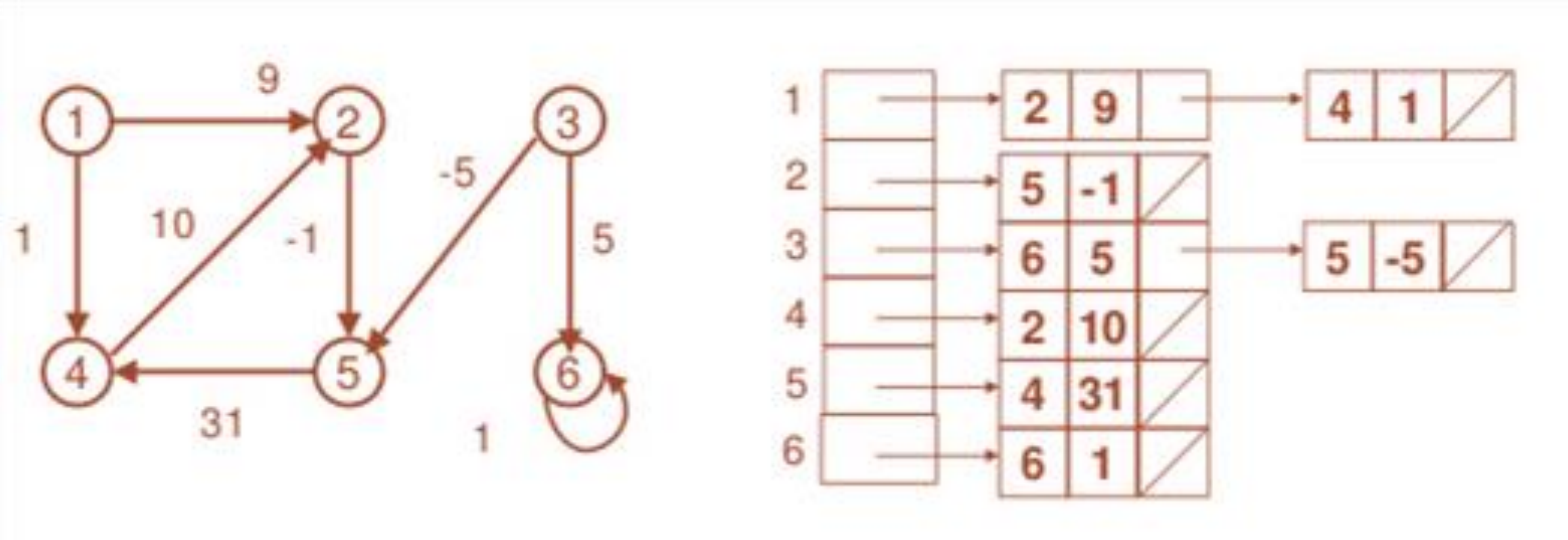
Directed Graph



	A	B	C	D	E	F
A	0	2	0	0	5	0
B	2	0	0	0	3	4
C	0	0	0	0	0	0
D	0	0	0	0	0	3
E	5	3	0	0	0	0
F	0	4	0	3	0	0

Weighted Graph

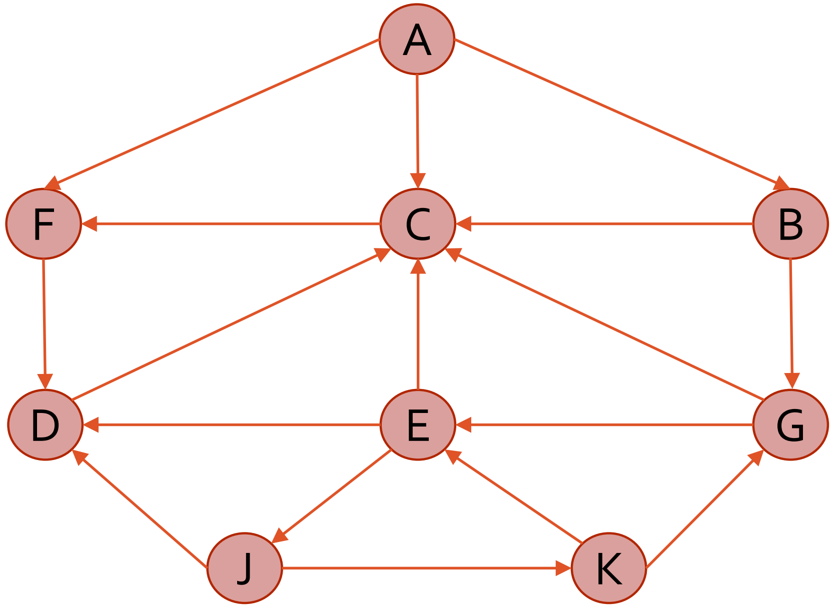
Adjacency Lists Representation



Breadth First Search (BFS) algorithm

Traversing a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Consider a given graph G. Suppose G represents the daily flights between cities of some airlines and suppose we want to fly from city A to city J with minimum no. of stops. In other words, we want the minimum path p from A to J.



Adjacency List	
A	F, C, B
B	G, C
C	F
D	C
E	D, C, J
F	D
G	C, E
J	D, K
K	E, G

Breadth First Search (BFS) algorithm

Adjacency List	
A	F, C, B
B	G, C
C	F
D	C
E	D, C, J
F	D
G	C, E
J	D, K
K	E, G

Step 1

F 1 Queue A

R 1 Origin Φ

Step 2

F 2 Queue A, F, C, B

R 4 Origin Φ, A, A, A

Step 3

F 3 Queue A, F, C, B, D

R 5 Origin Φ, A, A, A, F

Step 4

F 4 Queue A, F, C, B, D

R 5 Origin Φ, A, A, A, F

Step 5

F 5 Queue A, F, C, B, D, G

R 6 Origin Φ, A, A, A, F, B

Step 6

F 6 Queue A, F, C, B, D, G

R 6 Origin Φ, A, A, A, F, B

Step 7

F 7 Queue A, F, C, B, D, G, E

R 7 Origin Φ, A, A, A, F, B, G

Step 8

F 8 Queue A, F, C, B, D, G, E, J

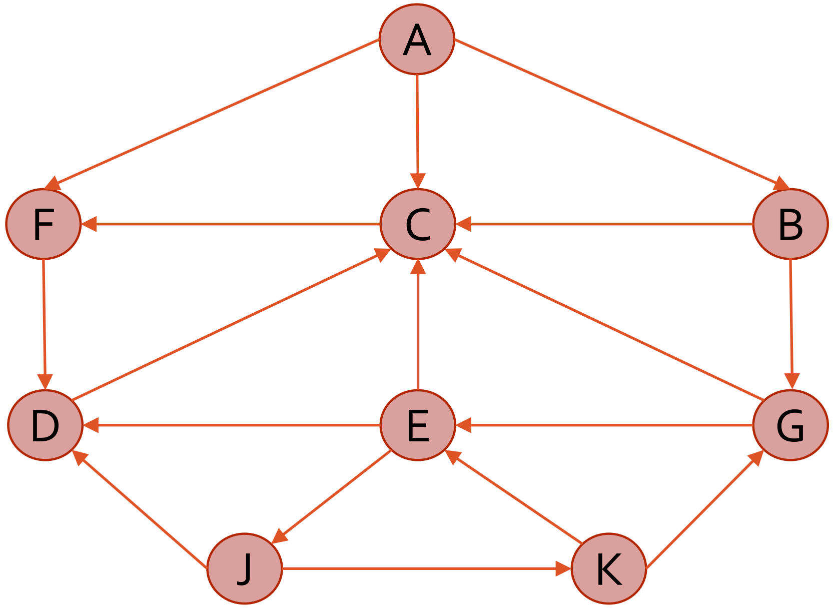
R 8 Origin Φ , A, A, A, F, B, G, E

J <— E <— G <— B <— A

Depth First Search (DFS) algorithm

Traversing a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Consider a given graph G. Suppose we want to find and print all nodes reachable from node J (including J itself).



Depth First Search (DFS) algorithm

Adjacency List	
A	F, C, B
B	G, C
C	F
D	C
E	D, C, J
F	D
G	C, E
J	D, K
K	E, G

Step 1

Stack J

Step 2

Print J

Stack D, K

Step 3

Print K

Stack D, E, G

Step 4

Print G

Stack D, E, C

Step 5

Print C

Stack D, E, F

Step 6

Print F

Stack D, E

Step 7

Print E

Stack D

Step 8

Print D

Stack

J, K, G, C, F, E, D