Data Structure and Algorithms

Lecture

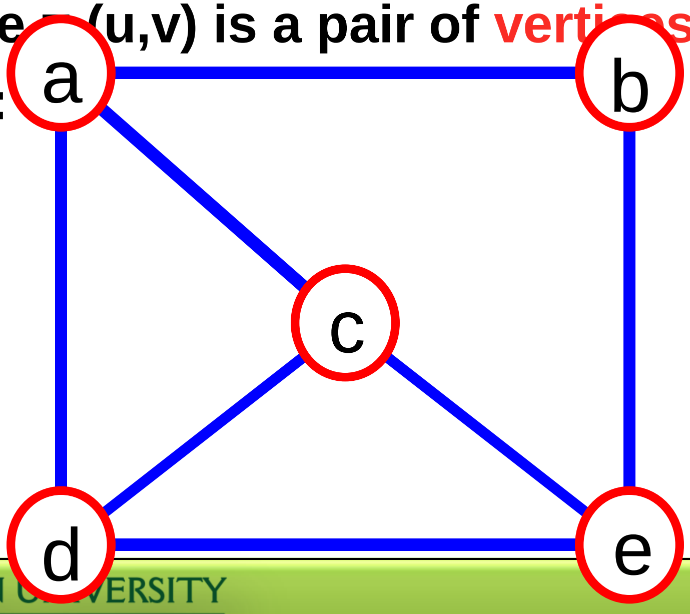
Graphs

What is a Graph?

- A graph consists of a number of data items, each of which is called a vertex. Any vertex may be connected to any other, these connections are called edges.
- A graph G = (V,E) is composed of:

  V: set of vertices

  E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:

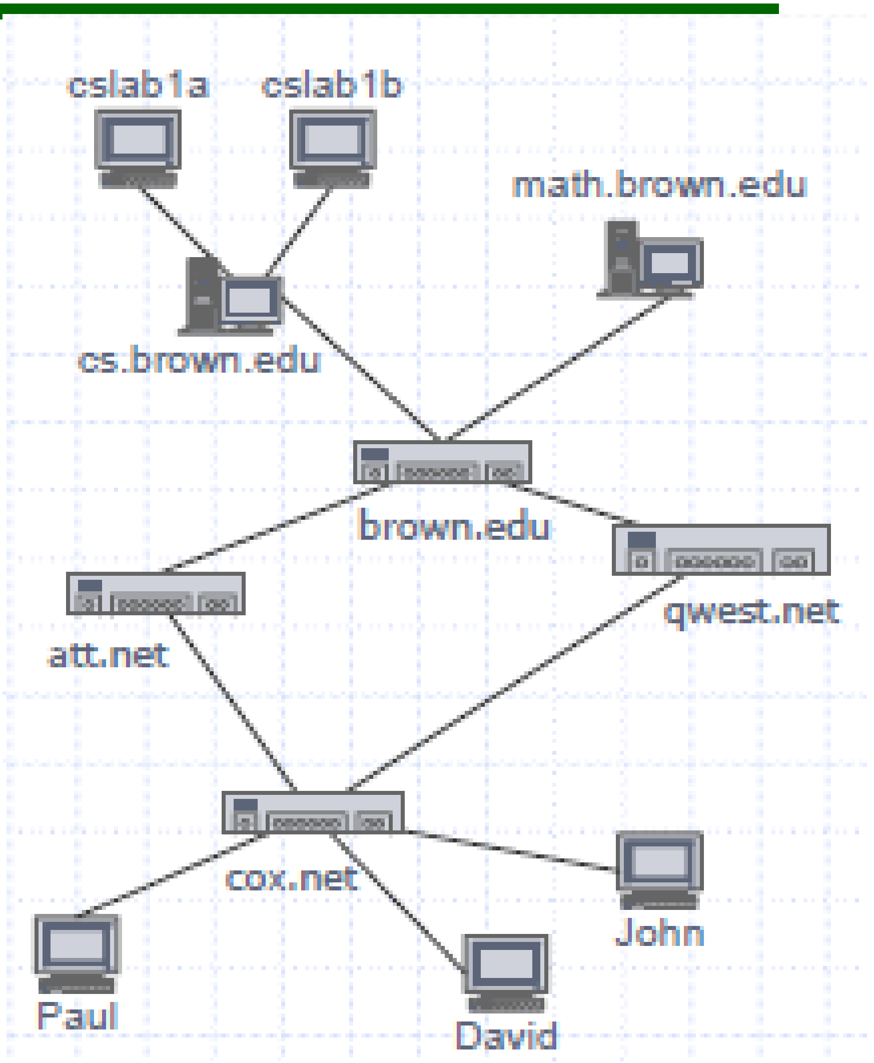


V= {a,b,c,d,e}

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

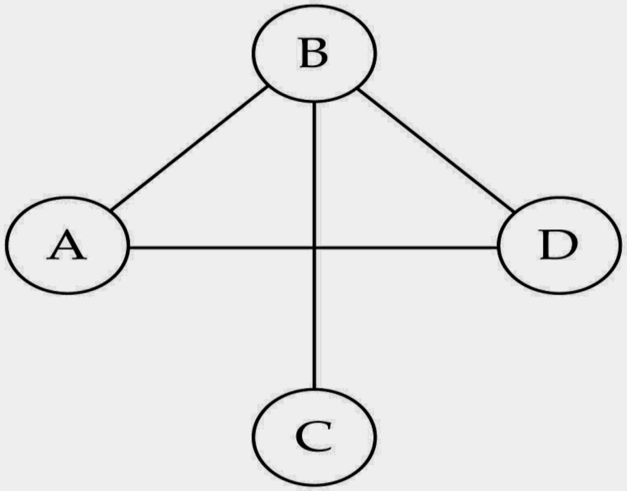
Applications

- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Computer networks
  - Local area network
  - Wide area network
  - Internet
- Databases
  - Entity-relationship diagram



Directed and Undirected Graph

- **Undirected graph**
  - **When the edges in a graph have no direction, the graph is called *undirected***

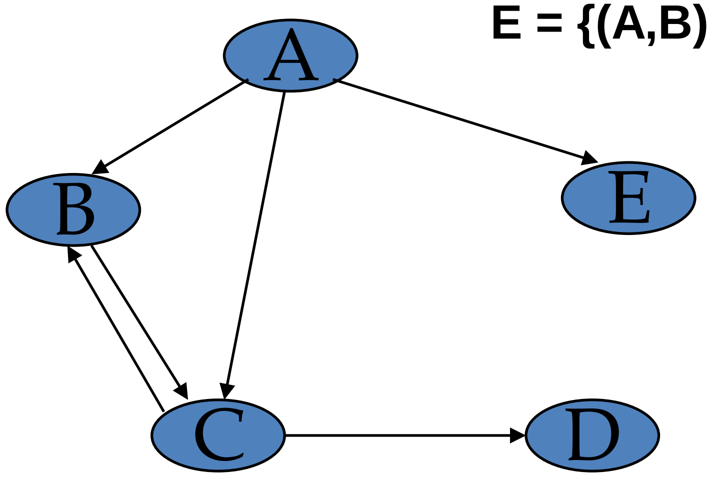


V(Graph1) = { A, B, C, D }
E(Graph1) = { (A, B), (A, D), (B, C), (B, D) }

Directed and Undirected Graph

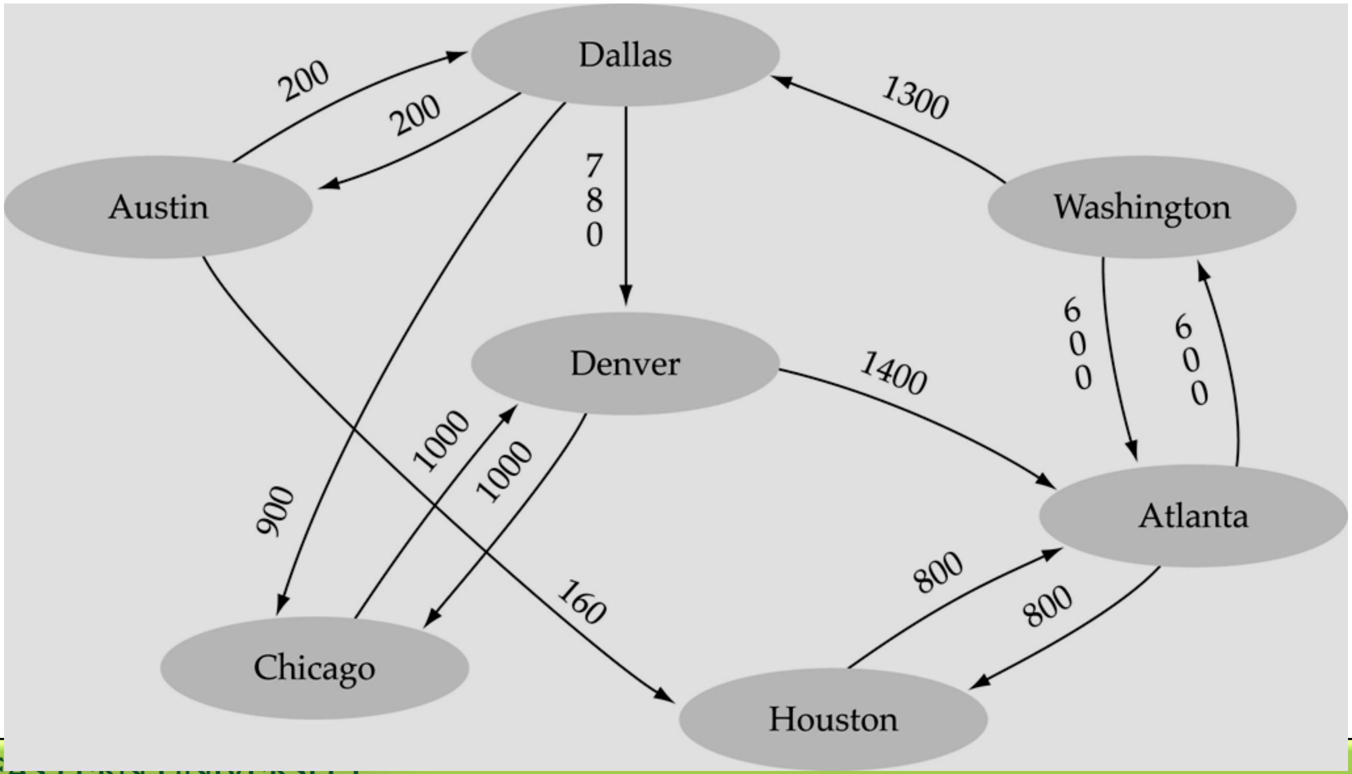
- When the edges in a graph have a direction, the graph is called *directed graph* .
- Also known as *digraph.*
- This kind of graph contains ordered pair of vertices i.e. if the graph is directed, the order of the vertices in each edge is important.

V = {A, B, C, D, E}
E = {(A,B), (B,C), (C,B), (A,C), (A,E), (C,D)}



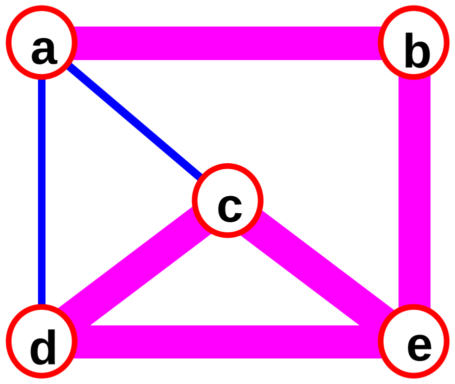
Graph Terminologies

- **Weighted Graph**
  - **A graph is suppose to be weighted if its every edge is assigned some value which is greater than or equal to zero.**

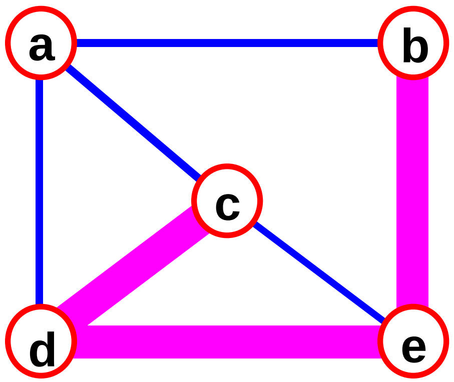


Graph Terminologies

- Adjacent Nodes
  - When there is an edge from one node to another then these nodes are called adjacent nodes.
- Path
  - sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.



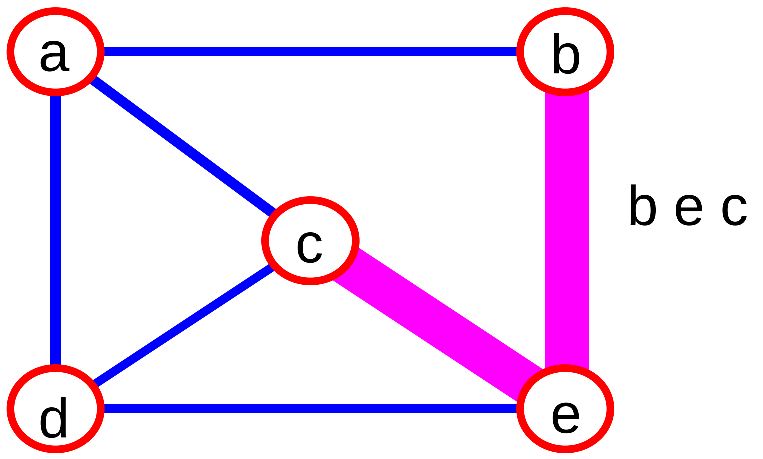
a b e d c



b e d c

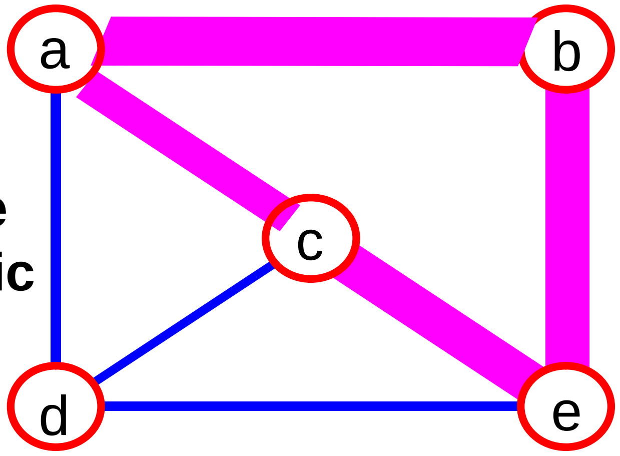
Graph Terminologies

- **Length of a Path**
  - **Length of a path is nothing but the total number of edges included in the path from source to destination node.**
- **Simple Path**
  - **path such that all its vertices and edges are distinct.**



Graph Terminologies

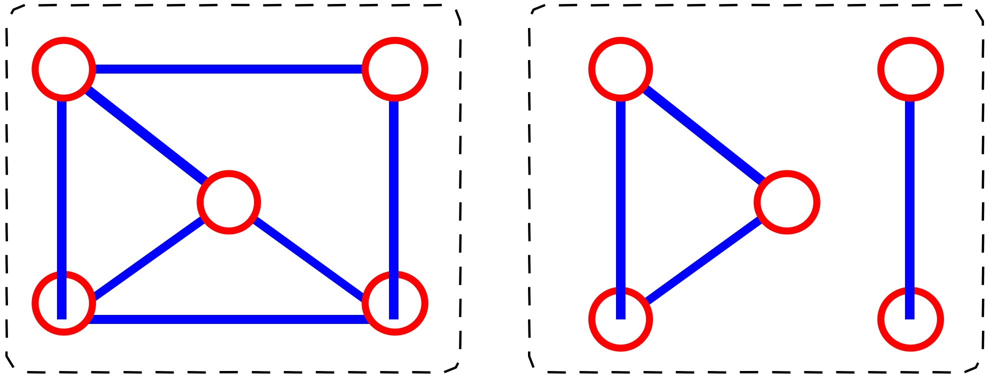
- **Cycle**
  - **simple path, except that the last vertex is the same as the first vertex**
- **Acyclic Graph**
  - **A graph without cycle is called acyclic Graph. A tree is a good example of acyclic graph.**



a b e c a

Graph Terminologies

- Connected Graph
  - An undirected graph is connected if, for any pair of vertices, there is a path between them.

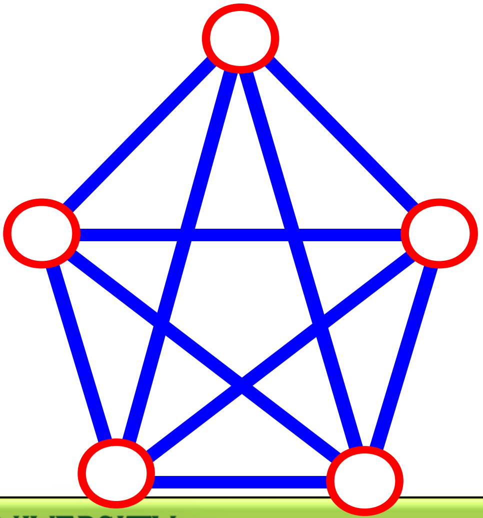


Connected Not Connected

  - A directed graph is connected if, for any pair of vertices, there is a path between them.

Graph Terminologies

- **Complete Graph**
  - **A graph in which all pairs of vertices are adjacent OR**
  - **A graph in which every vertex is directly connected to every other vertex**
  - Let **n** = Number of vertices, and
    m = Number of edges
  - **For a complete graph with *n* vertices, the number of edges is *n(n – 1)/2*. A graph with 6 vertices needs 15 edges to be complete.**

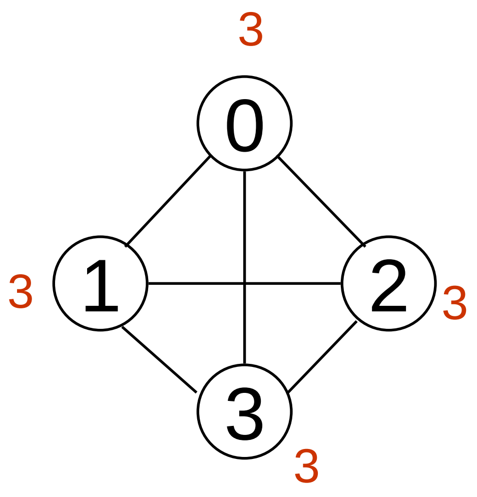


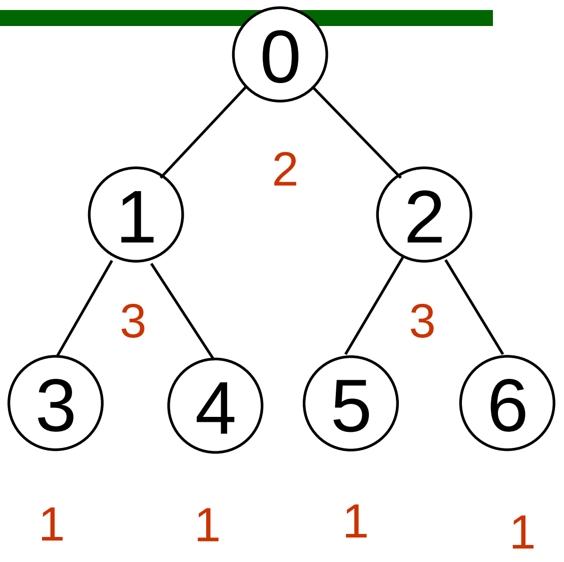
$\mathbf{n} = 5$
$\mathbf{m} = (5 * 4)/2 = 10$

Graph Terminologies

- Degree of a node
  - In an Undirected graph, the total number of edges linked to a node is called a degree of that node.
  - For directed graph there are two degree for every node
    - **Indegree**
      - The indegree of a node is the total number of edges coming to that node.
    - **Outdegree**
      - The outdegree of a node is the total number of edges going out from that node

Graph Terminologies

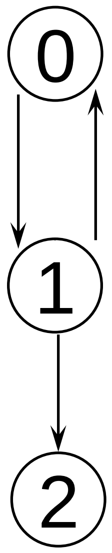




G2

Directed graph

in-degree
out-degree



in:1, out: 1

in: 1, out: 2

in: 1, out: 0

G3

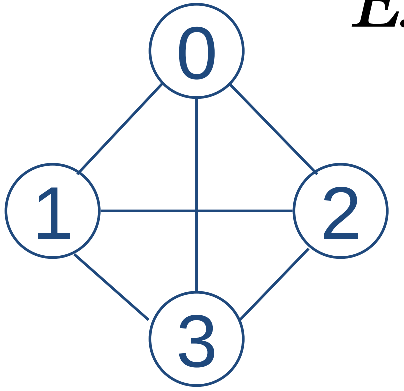
Graph Representation

- There are two methods to represent a Graph
  - Adjacency Metrix (Array Implementation)
  - Adjacency List (Linked List Implementation)

Adjacency Matrix

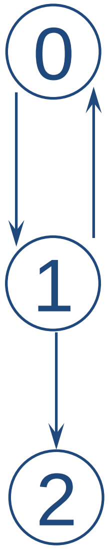
- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (vi, vj) is in E(G), adj_mat[i][j]=1
- If there is no such edge in E(G), adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; since adj_mat[i][j]=adj_mat[j][i]
- The adjacency matrix for a digraph may not be symmetric

Examples for Adjacency Matrix



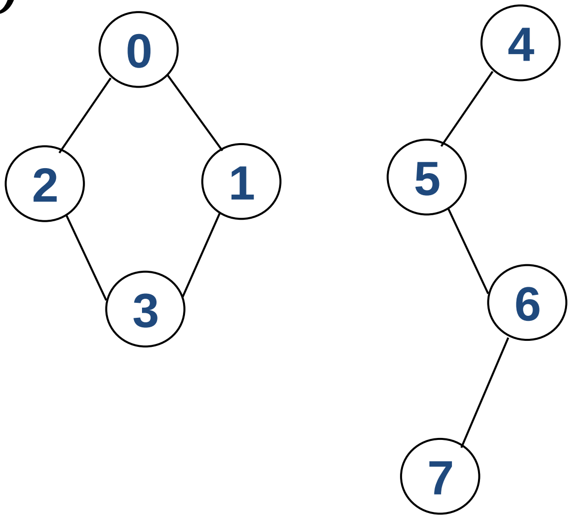
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

G_1



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

G_2



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

symmetric

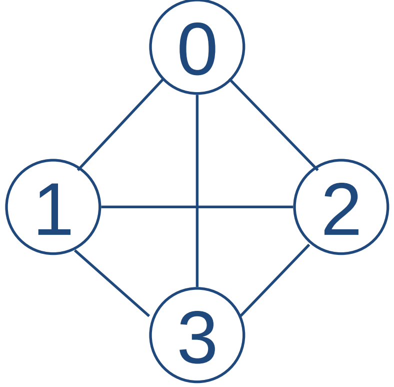
G_4

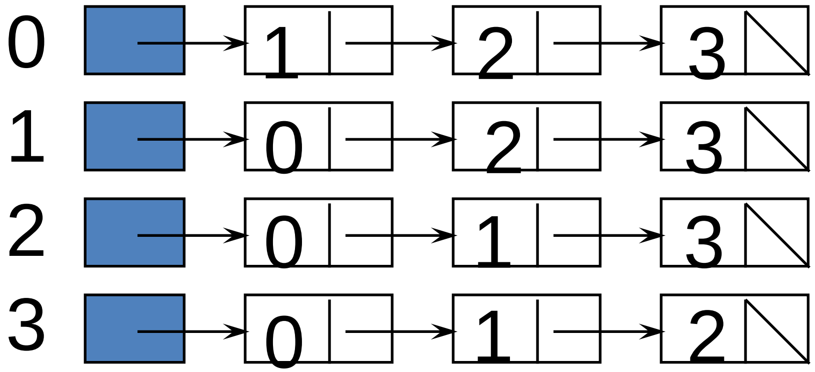
Adjacency Matrix

- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is $\sum_{j=0}^{n-1} adj\_mat[i][j]$
- We can also represent weighted graph with Adjacency Matrix. Now the contents of the martix will not be 0 and 1 but the value is substituted with the corresponding weight.

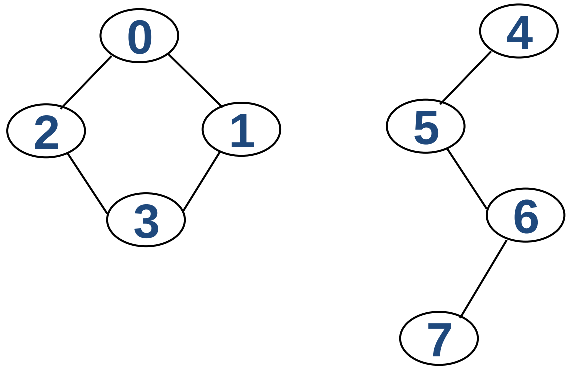
Adjacency List

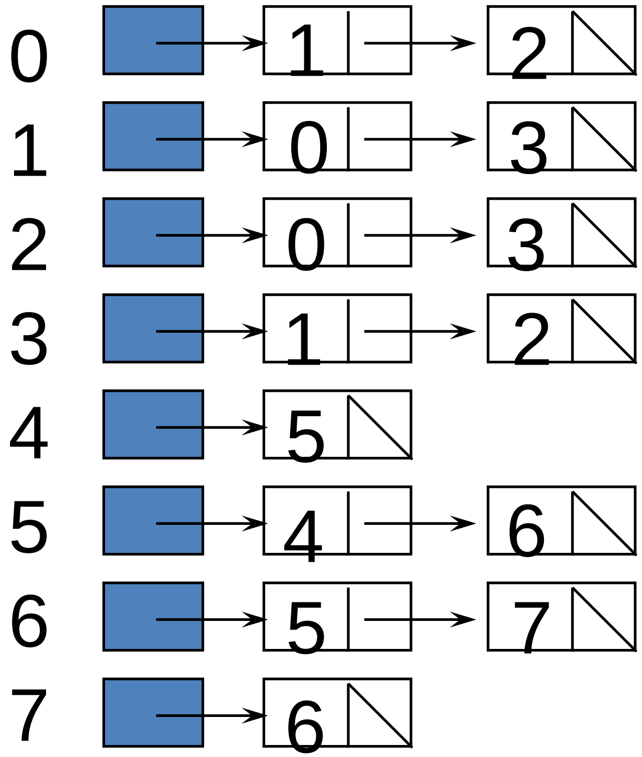
- A Single Dimension array of Structure is used to represent the vertices
- A Linked list is used for each vertex V which contains the vertices which are adjacent from V (adjacency list)


0
1
2
3

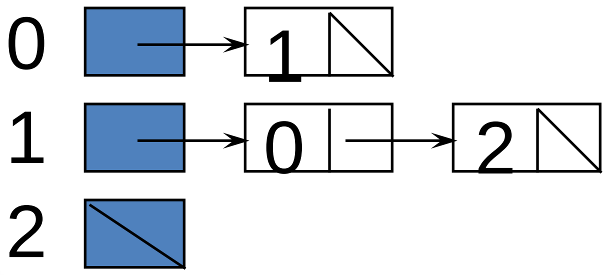

0 → 1 → 2 → 3
1 → 0 → 2 → 3
2 → 0 → 1 → 3
3 → 0 → 1 → 2

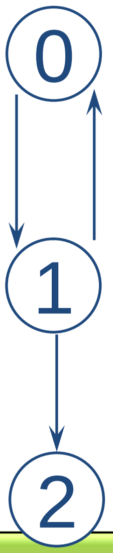
G_1


0
1
2
3
4
5
6
7


0 → 1 → 2
1 → 0 → 3
2 → 0 → 3
3 → 1 → 2
4 → 5
5 → 4 → 6
6 → 5 → 7
7 → 6

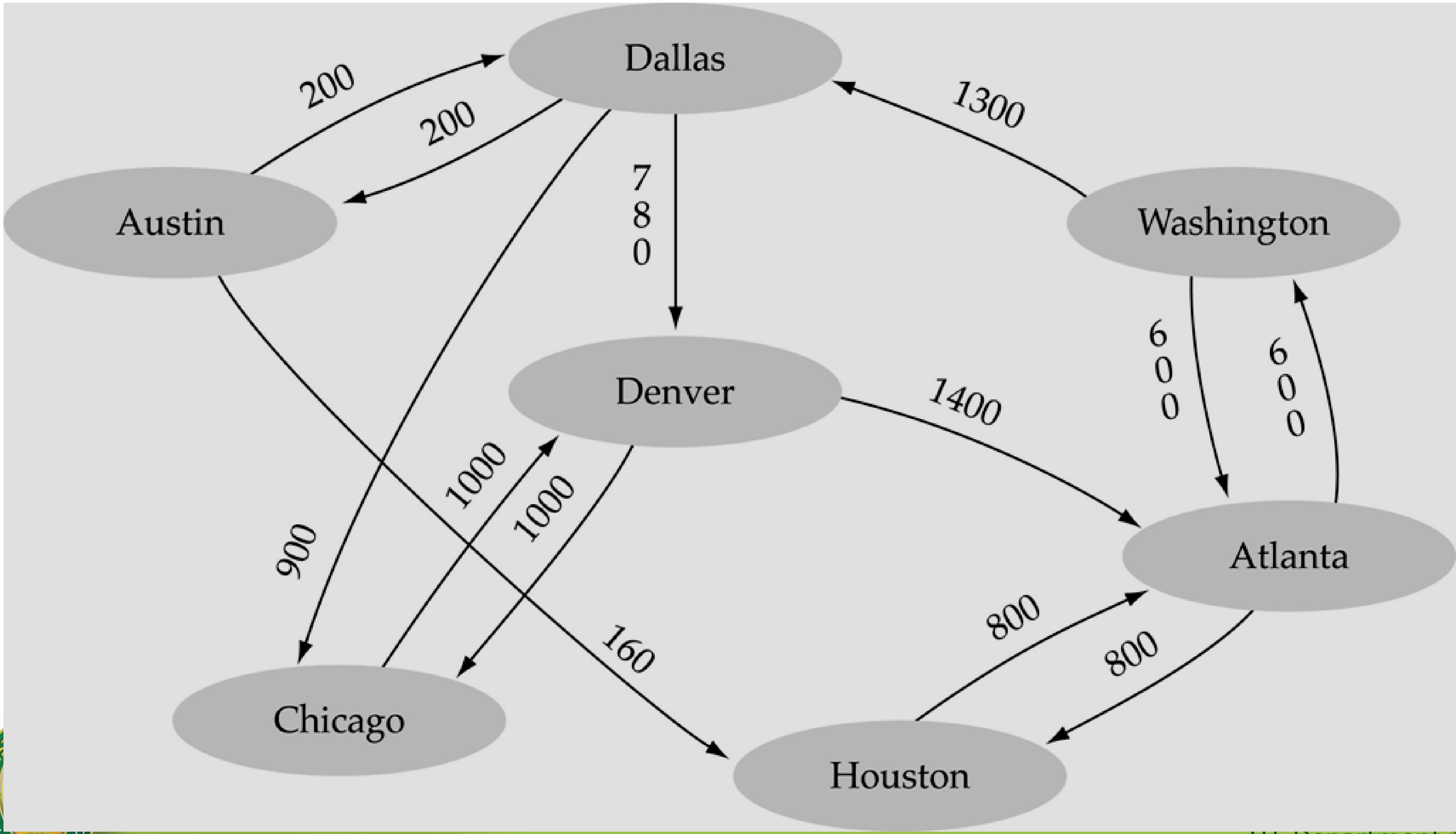
G_4


0 → 1
1 → 0 → 2
2

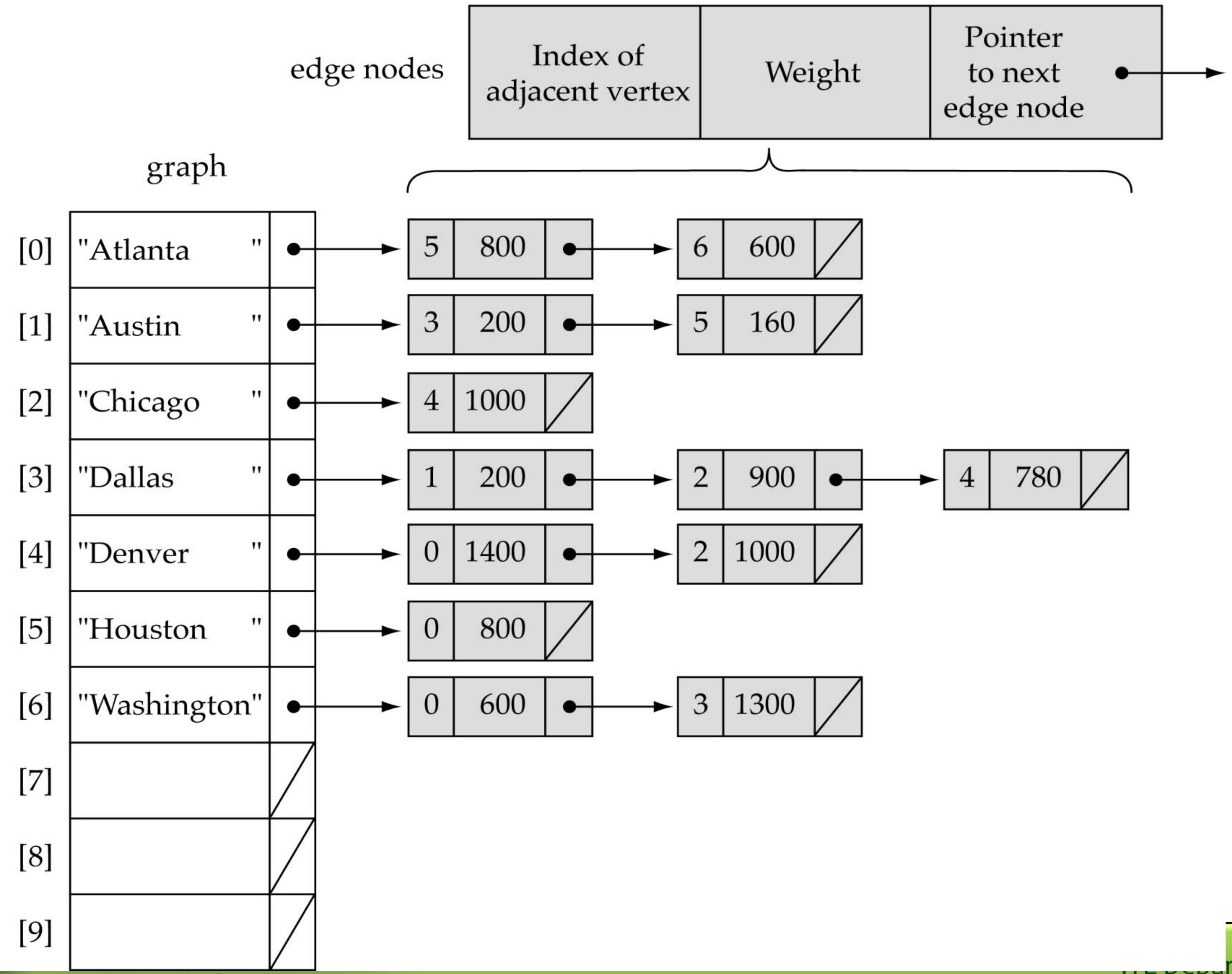

0
1
2

G_3

Another Example of Adjacency List



(a)


edge nodes
Index of adjacent vertex
Weight
Pointer to next edge node
graph
[0] "Atlanta "
[1] "Austin "
[2] "Chicago "
[3] "Dallas "
[4] "Denver "
[5] "Houston "
[6] "Washington"
[7]
[8]
[9]
5 800
6 600
3 200
5 160
4 1000
1 200
2 900
4 780
0 1400
2 1000
0 800
0 600
3 1300

Graph Traversal

- Traversal is the facility to move through a structure visiting each of the vertices once.

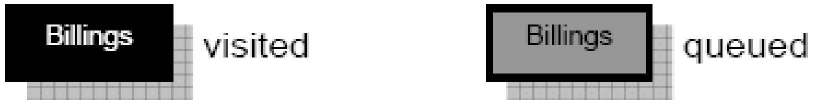
- We looked previously at the ways in which a binary tree can be traversed. Two of the traversal methods for a graph are breadth-first and depth-first.

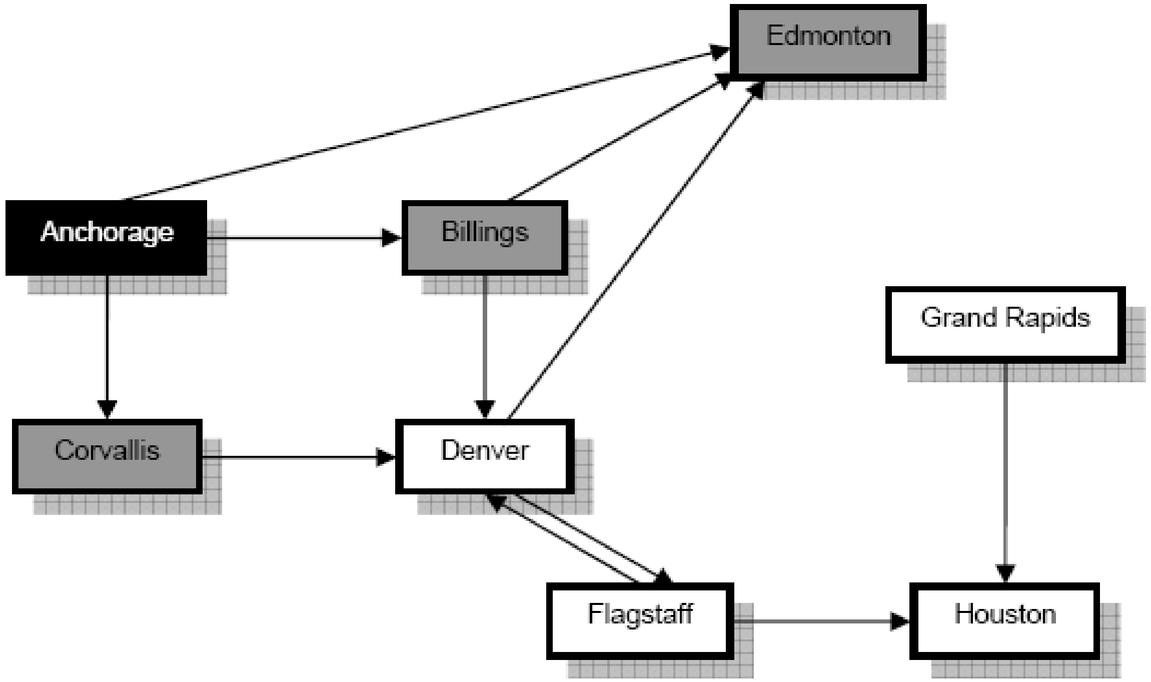
Breadth-First Graph Traversal

- This method visits all the vertices, beginning with a specified **start vertex**. It can be described roughly as “neighbours-first”.
- No vertex is visited more than once, and **vertices are visited only if they can be reached** – that is, if there is a path from the start vertex.
- Breadth-first traversal makes use of a **queue data structure**. The queue holds a list of vertices which have not been visited yet but which should be visited soon.
- Since a queue is a first-in first-out structure, vertices are visited in the order in which they are added to the queue.
- Visiting a vertex involves, for example, outputting the data stored in that vertex, and also **adding its neighbours to the queue**.
- Neighbours are not added to the queue if they are

Breadth-First Traversal Example

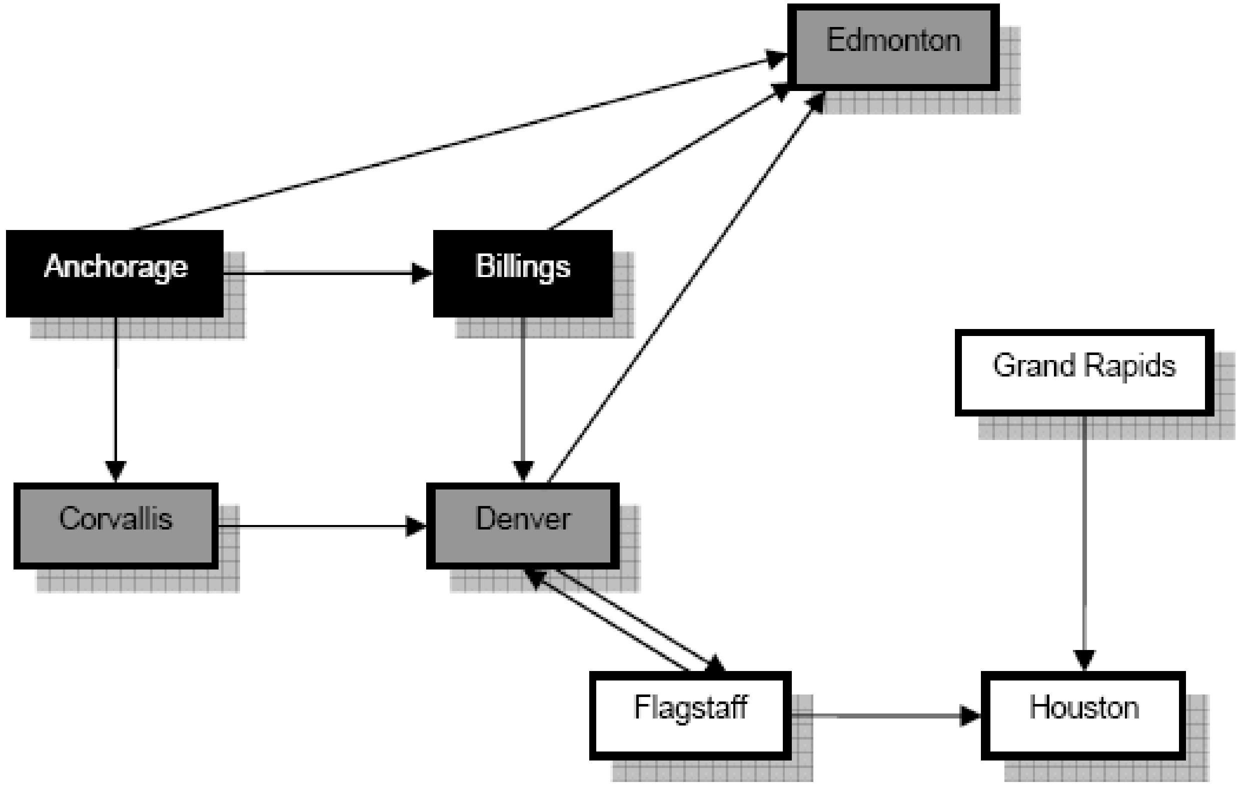
- Neighbours are added to the queue in alphabetical order. Visited and queued vertices are shown as follows:





Visited: *Anchorage*
Queue: *Billings, Corvallis, Edmonton* visit *Billings* next

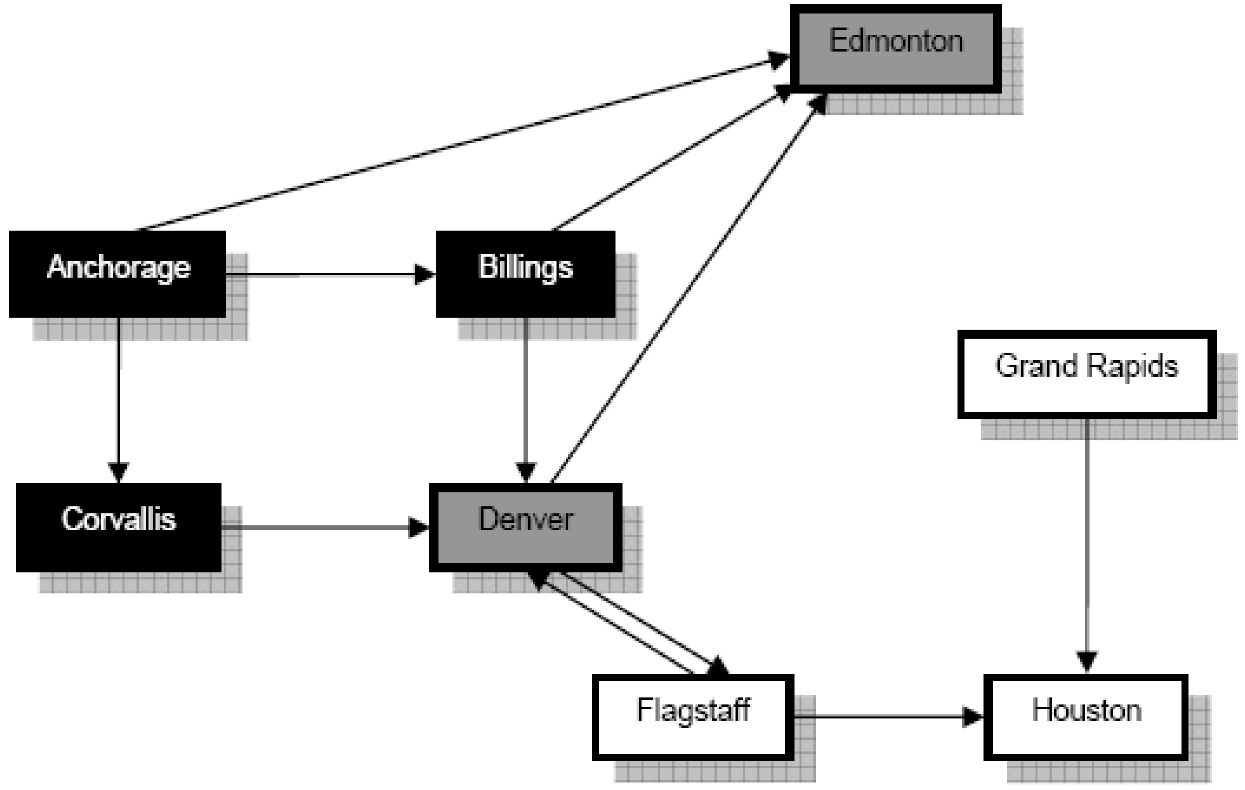
Breadth-First Graph Traversal



Visited: *Anchorage, Billings*
Queue: *Corvallis, Edmonton, Denver*   visit *Corvallis* next

- **Note** that we only add Denver to the queue as the other neighbours of Billings are already in the queue.

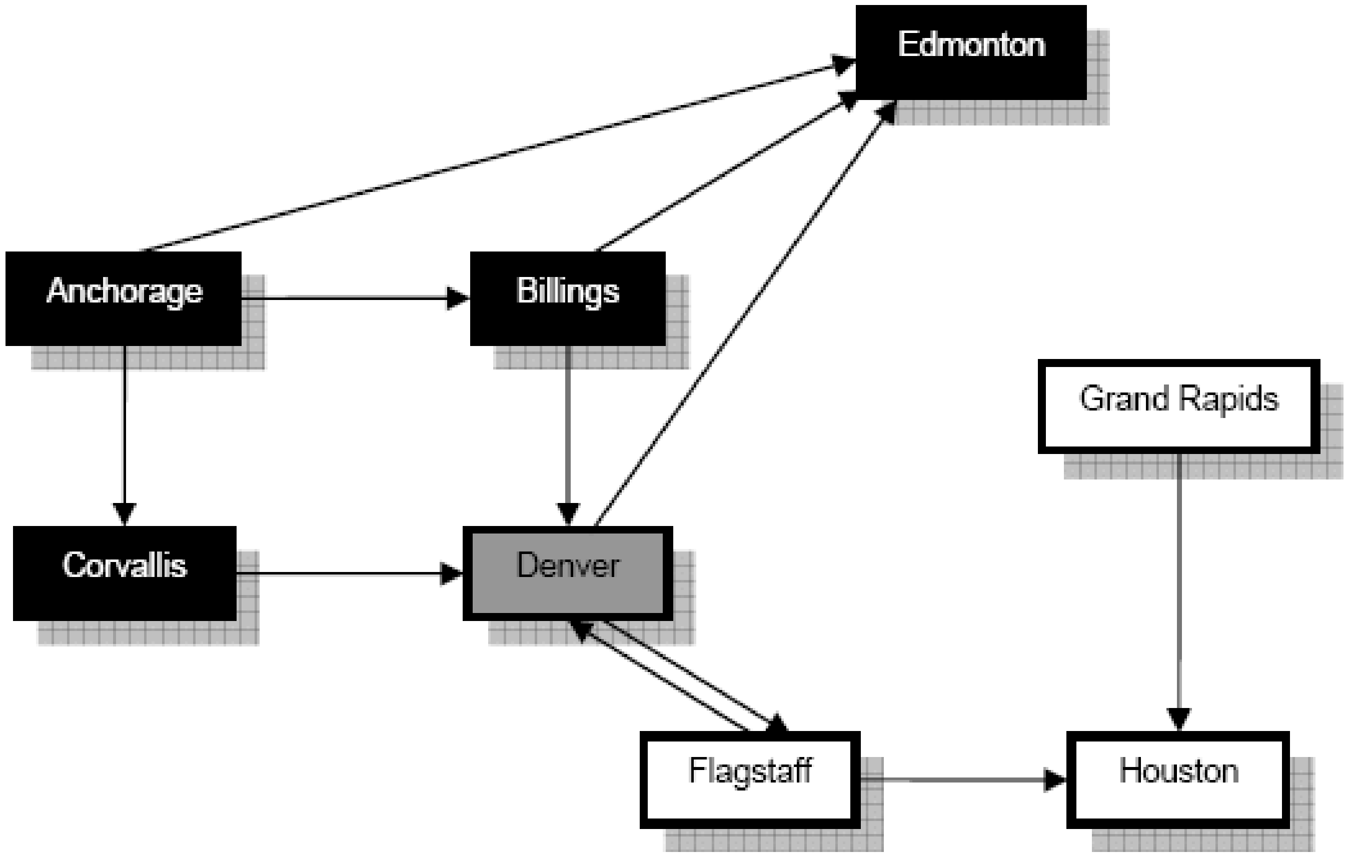
Breadth-First Graph Traversal



Visited: *Anchorage, Billings, Corvallis*
Queue: *Edmonton, Denver*   visit *Edmonton* next

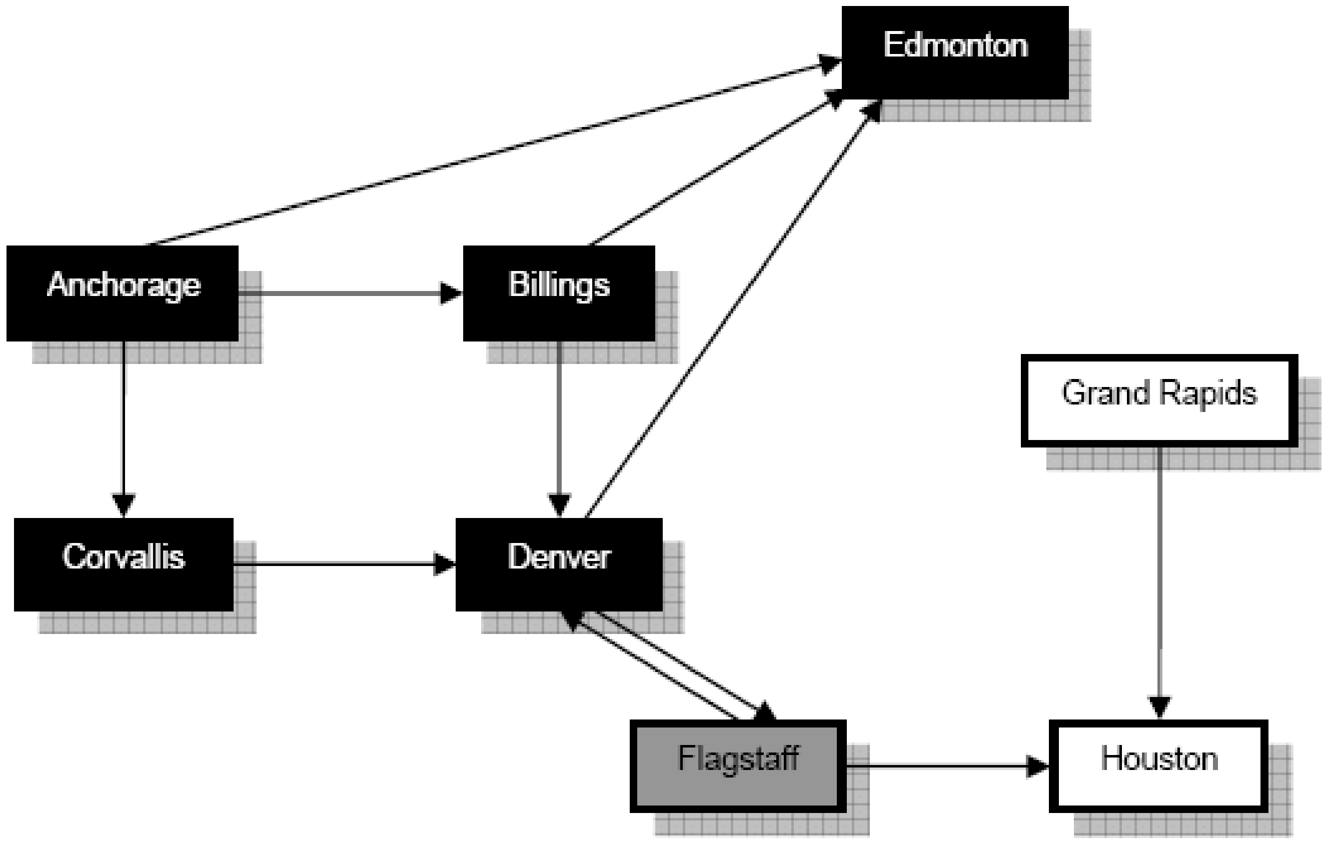
- **Note** that nothing is added to the queue as Denver, the only neighbour of Corvallis, is already in the queue.

Breadth-First Graph Traversal



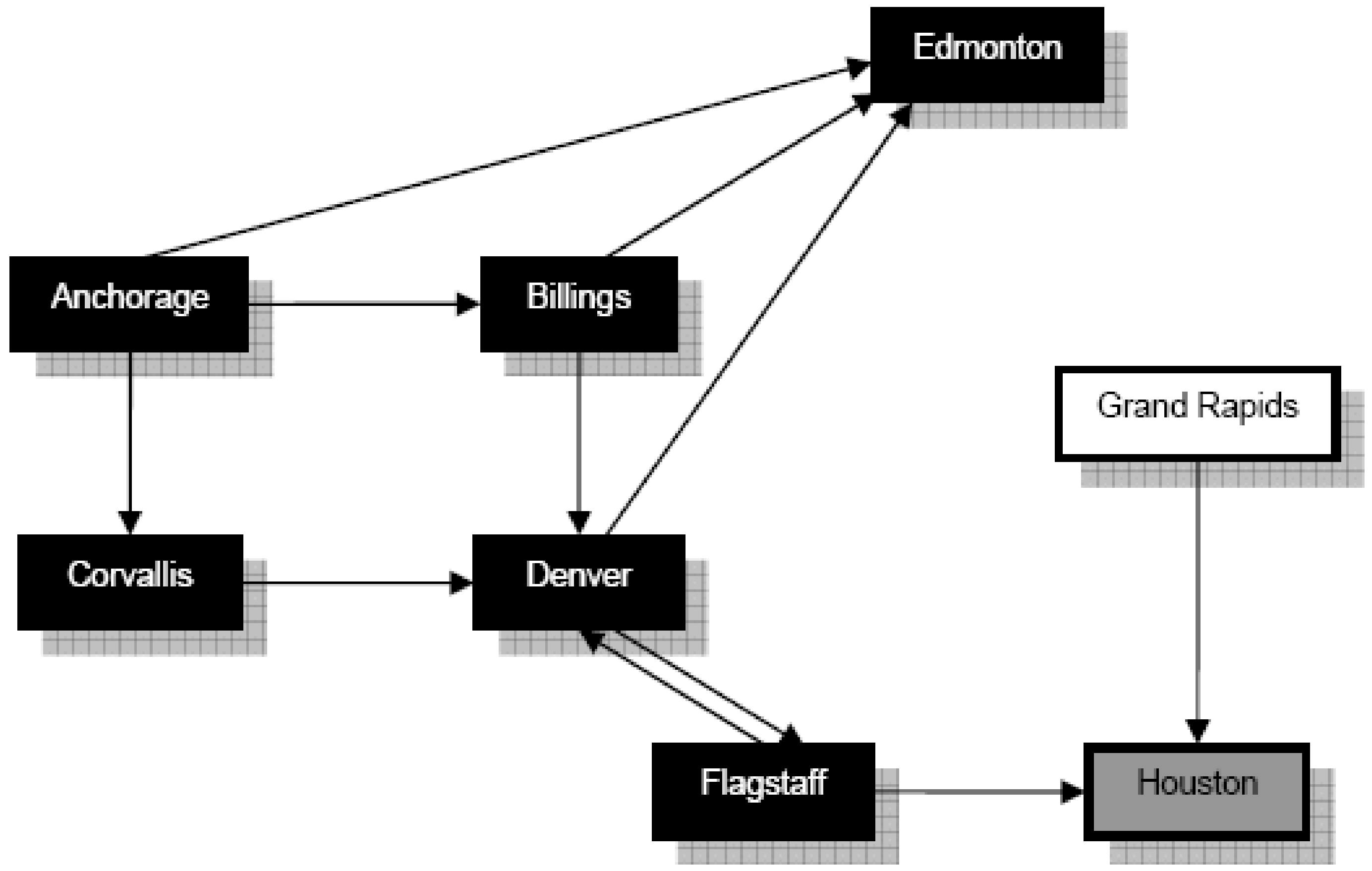
Visited: *Anchorage, Billings, Corvallis, Edmonton*
Queue: *Denver* visit *Denver* next

Breadth-First Graph Traversal



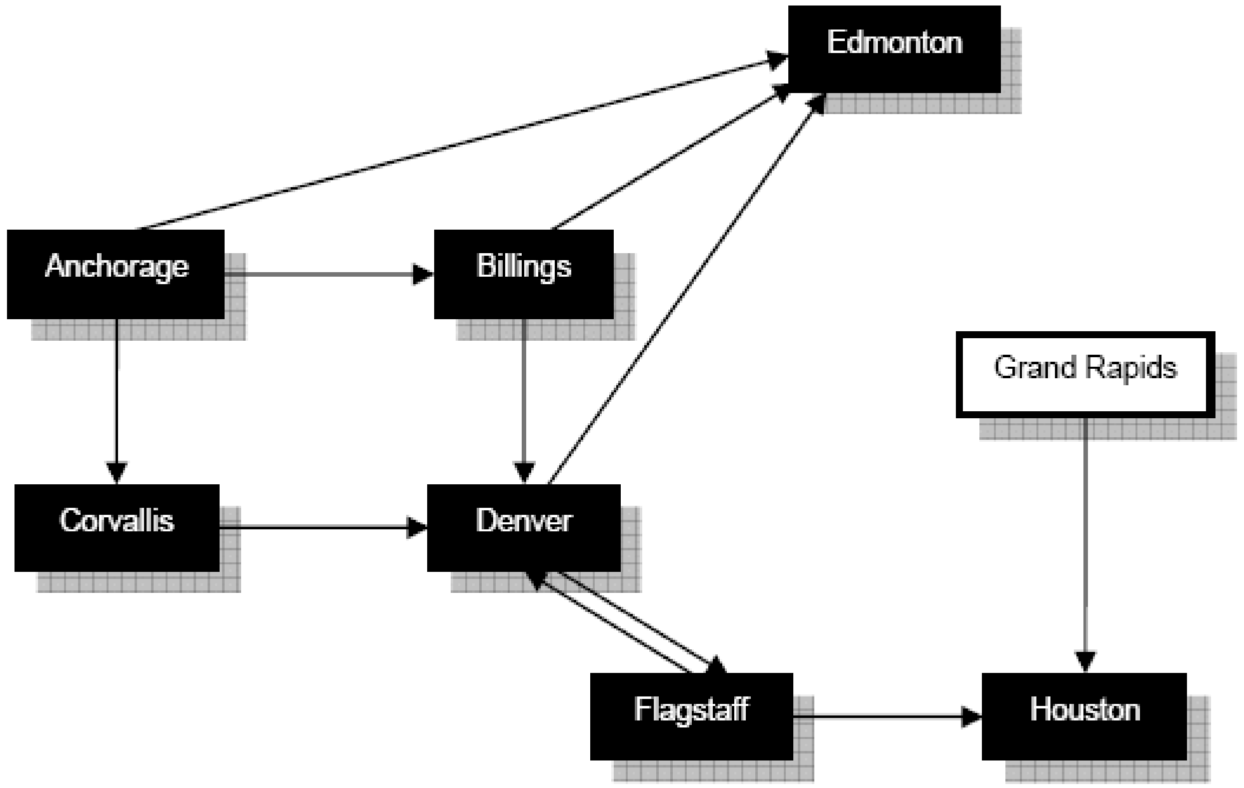
Visited: *Anchorage, Billings, Corvallis, Edmonton, Denver*
Queue: *Flagstaff* visit *Flagstaff* next

Breadth-First Graph Traversal



Visited: *Anchorage, Billings, Corvallis, Edmonton, Denver, Flagstaff*
Queue: *Houston* visit *Houston* next

Breadth-First Graph Traversal



Visited: *Anchorage, Billings, Corvallis, Edmonton, Denver, Flagstaff, Houston*
Queue: *empty*

- **Note** that Grand Rapids was not added to the queue as there is no path from Houston because of the edge direction.
- Since the queue is empty, we must stop, so the traversal is complete.
- The order of traversal was:*Anchorage, Billings, Corvallis, Edmonton, Denver, Flagstaff, Houston*

Depth-First Search (DFS)

Depth-First Search

1. From the given vertex, visit one of its adjacent vertices and leave others;
2. Then visit one of the adjacent vertices of the previous vertex;
3. Continue the process, visit the graph as deep as possible until:
   - A visited vertex is reached;
   - An end vertex is reached.

Depth-First Traversal

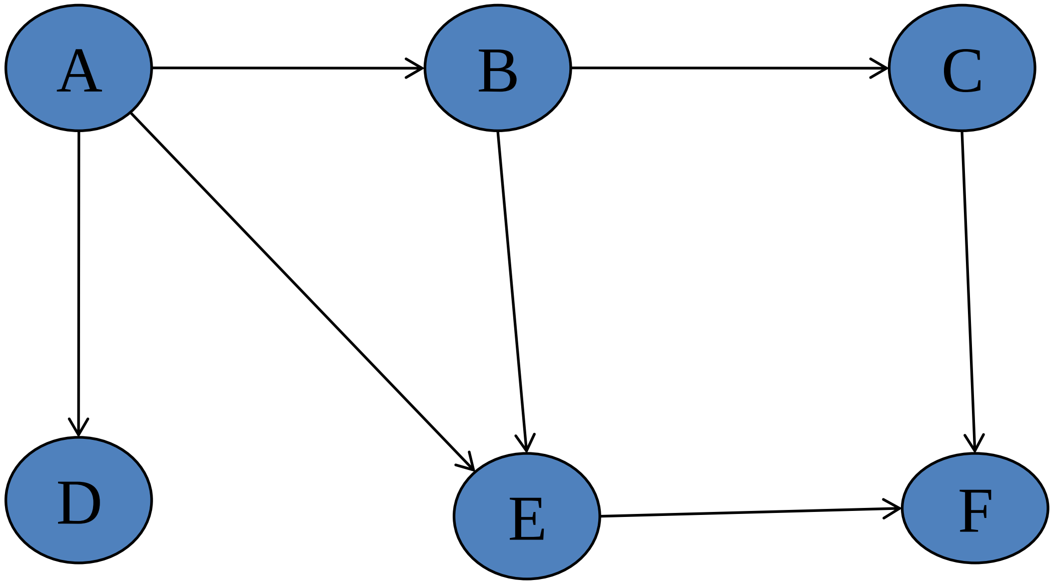
1. Depth-first traversal of a graph:
2. Start the traversal from an arbitrary vertex;
3. Apply depth-first search;
4. When the search terminates, backtrack to the previous vertex of the finishing point,
5. Repeat depth-first search on other adjacent vertices, then backtrack to one level up.
6. Continue the process until all the vertices that are reachable from the starting vertex are visited.
7. Repeat above processes until all vertices are visited.

Depth First Search Traversal

- This method visits all the vertices, beginning with a specified **start vertex**.
- This strategy proceeds along a path from vertex V as deeply into the graph as possible.
- This means that after visiting V, the algorithm tries to visit any unvisited vertex adjacent to V.
- When the traversal reaches a vertex which has no adjacent vertex, it back tracks and visits an unvisited adjacent vertex.

- Depth-first traversal makes use of a **Stack data structure**.

DFS Example



A B C F E D