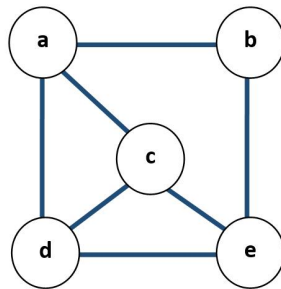


What is Graph?

- 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:



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

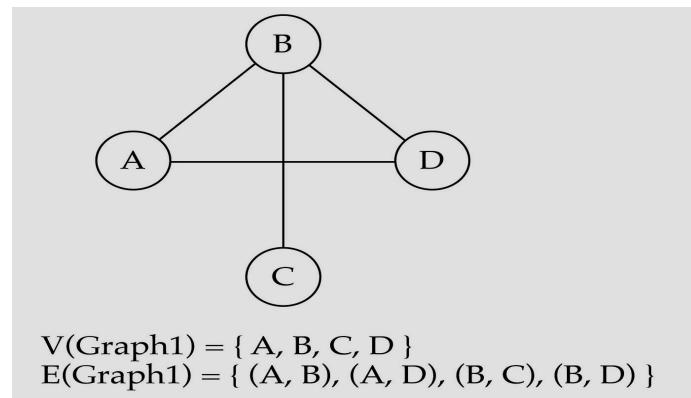
$E = \{(a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e)\}$

Main Application of Graphs

- 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

Directed And Undirected Graph

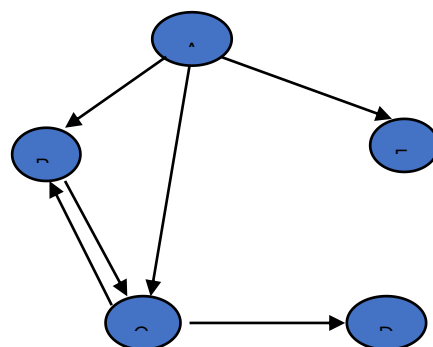
Undirected graph When the edges in a graph have no direction, the graph is called *undirected*.



- 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.

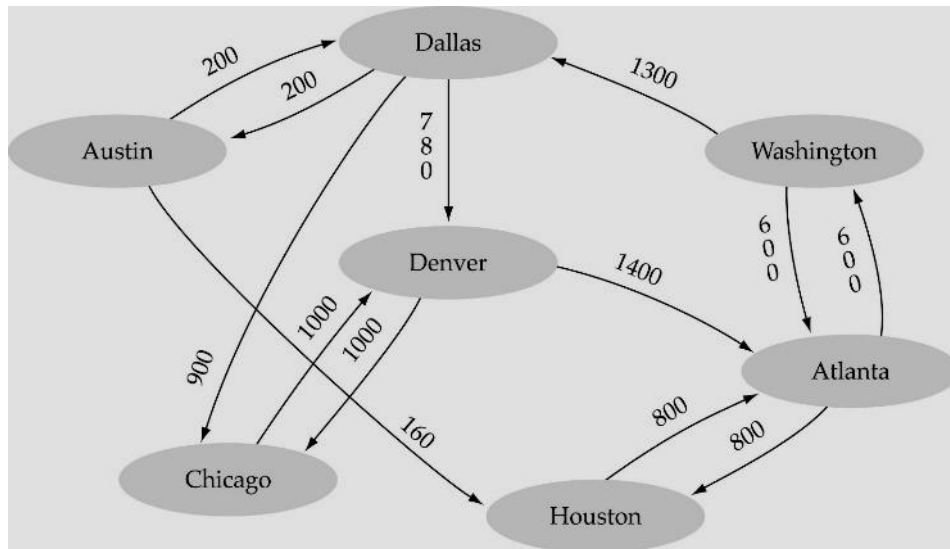
$V = \{A, B, C, D, E\}$

$E = \{(A,B), (B,C), (C,B), (A,C), (A,E), (C,D)\}$

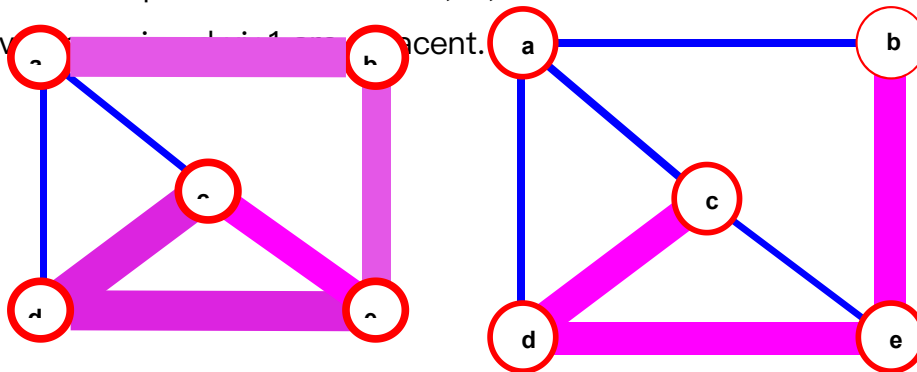


Basic Terminologies

- **Weighted Graph**– A graph in which every edge is assigned some value which is greater than or equal to zero.



- **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, \dots, v_k such that consecutive vertices v_i and v_{i+1} are adjacent.

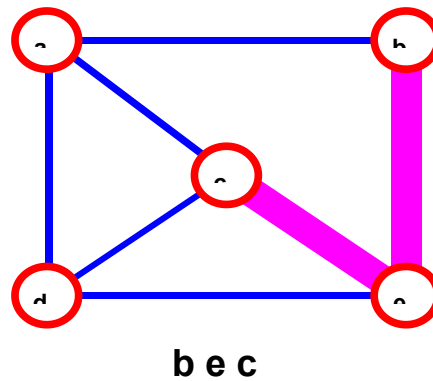


a b e d c

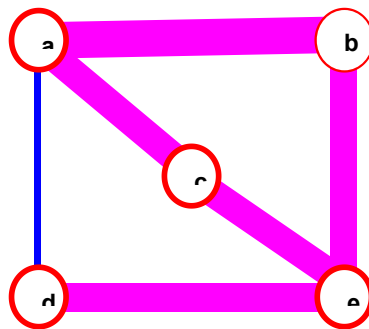
b e d c

- **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.

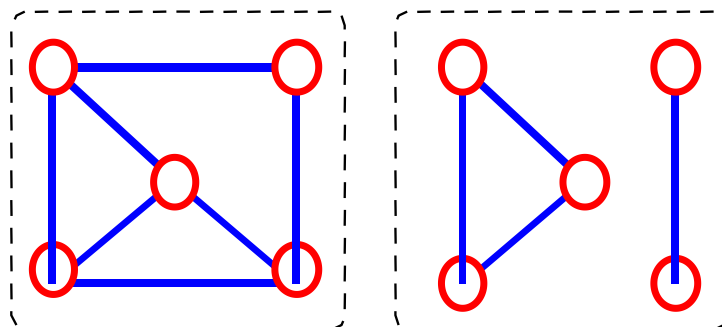
- **Simple Path**– path such that all its vertices and edges are distinct.



- **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.



- **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.

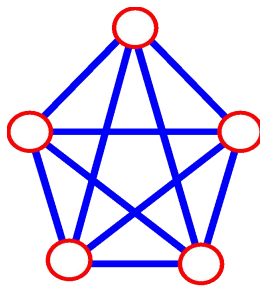
- **Complete Graph**– A graph in which all pairs of vertices are adjacent OR

A graph in which every vertex is directly connected to every other vertex

Let n = Number of vertices, and

m = Number of edges

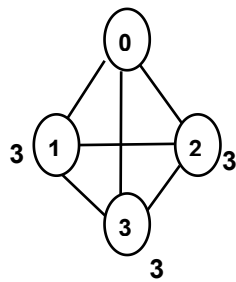
For a complete graph with n vertices, the number of edges is $n(n - 1)/2$. A graph with 6 vertices needs 15 edges to be complete.



$$n = 5$$

$$m = (5 * 4)/2 = 10$$

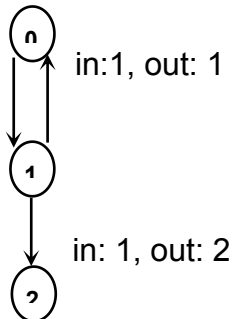
- **Degree of a node**– the total number of edges linked to a node in an Undirected graph, For directed graph there are two degree for every node:
- **Indegree**– The indegree of a node is the total number of edges coming to that node.
- **Outdegree**– The outdegree of a node is the total number of edges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



G3

Directed graph

**in-degree
out-degree**



G2

Graph Representation

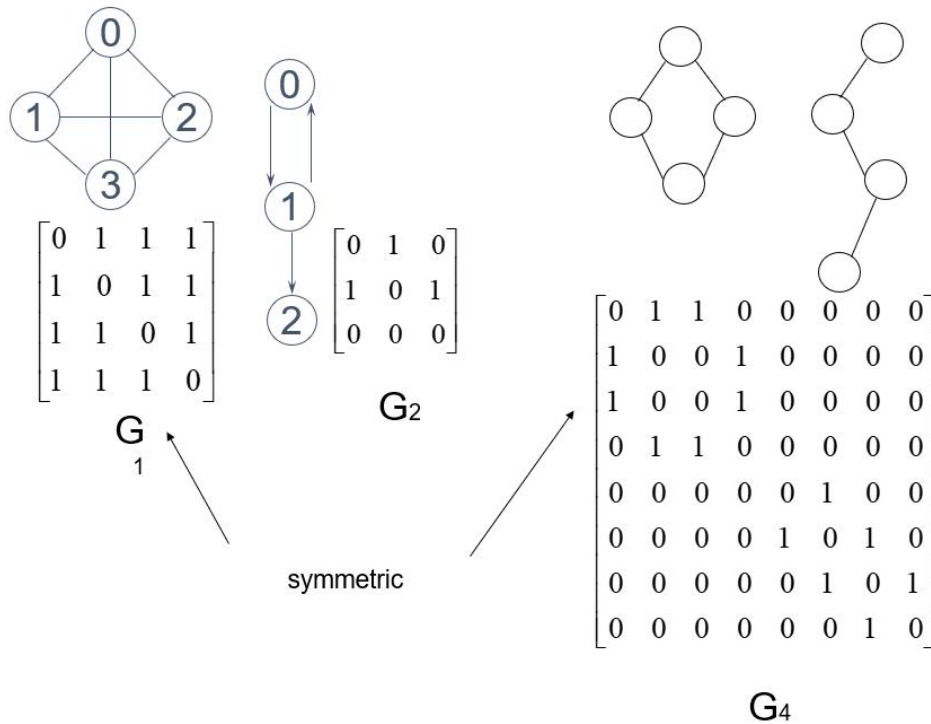
There are two methods to represent a Graph

- Adjacency Metrix (Array Implementation)
- Adjacency List (Linked List Implementation)

Adjacency Matrix

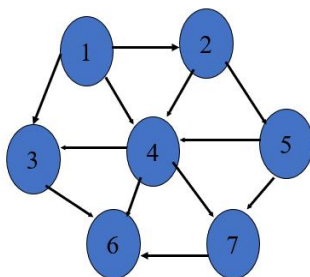
- Let $G=(V,E)$ be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (v_i, v_j) is in $E(G)$, $adj_mat[i][j]=1$
- If there is no such edge in $E(G)$, $adj_mat[i][j]=0$
- The adjacency matrix for an undirected graph is symmetric; since $adj_mat[i][j]=adj_mat[j][i]$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

Examples for Adjacency Matrix



Adjacency Matrix

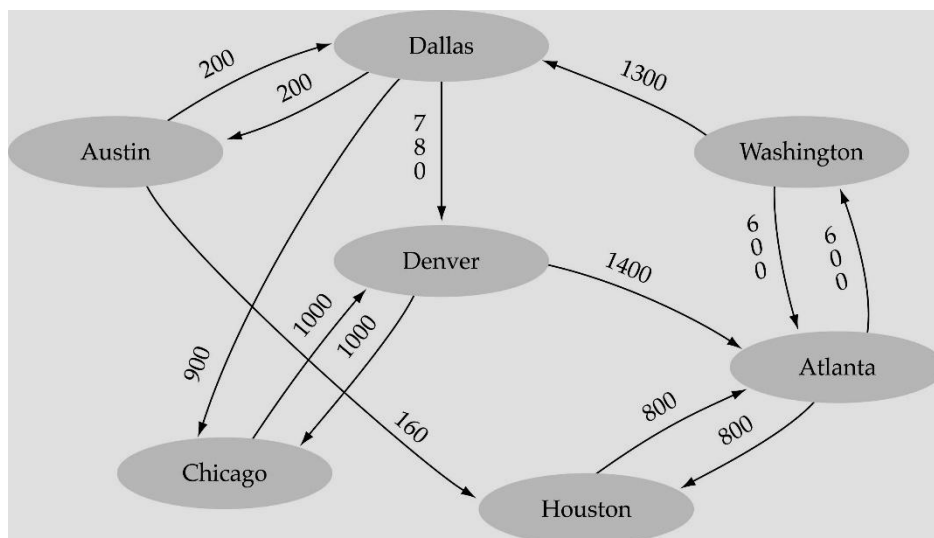
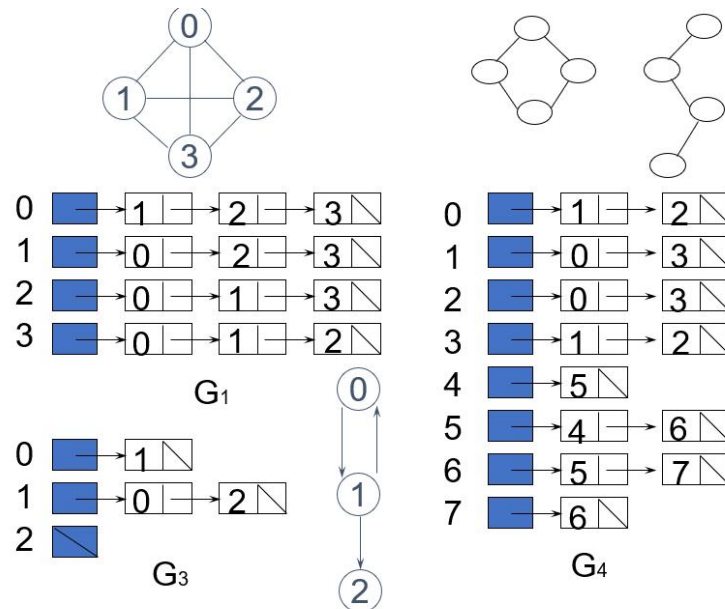
- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is
- We can also represent weighted graph with Adjacency Matrix. Now the contents of the matrix will not be 0 and 1 but the value is substituted with the corresponding weight.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Another Example Adjacency

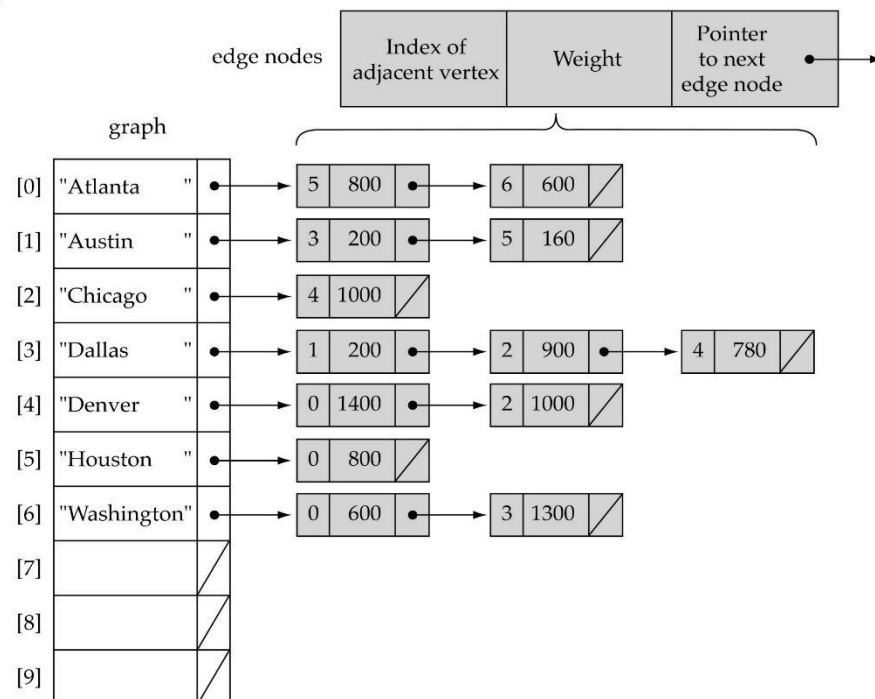
- **Adjacency List**– A Single Dimension array of Structure is used to represent the vertices. A Linked list is used for each vertex V which contains the vertices which are adjacent from V (adjacency list)



Another Example of Adjacency List

Linked-list Implementation

(a)



Graph Traversal

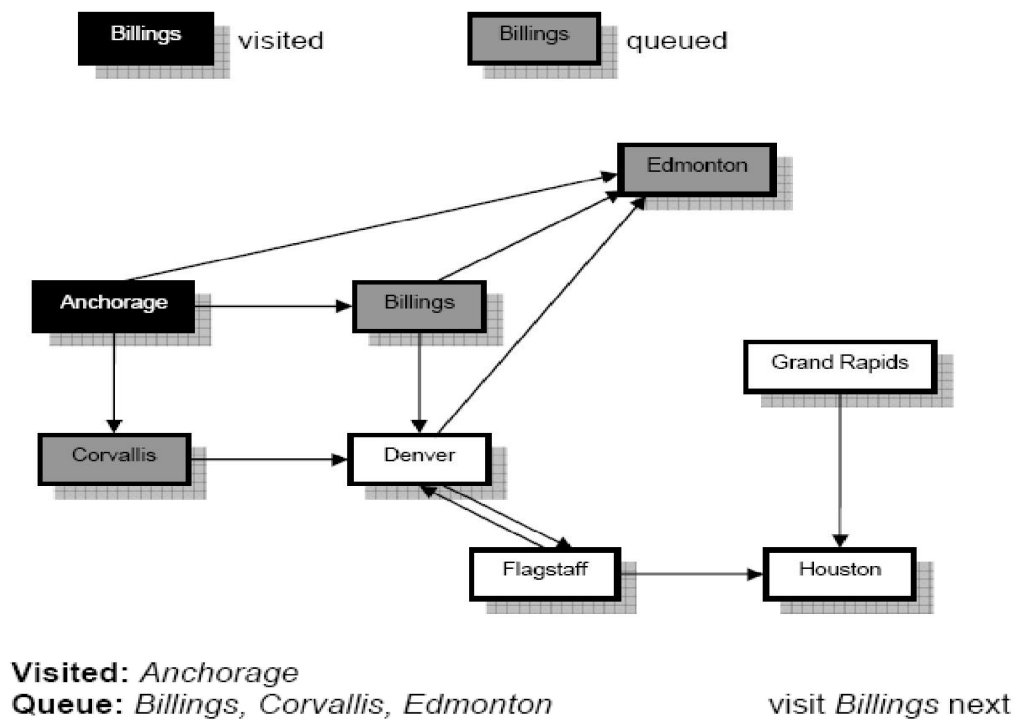
- **Traversal** is the facility to move through a structure visiting each of the vertices once.
- We looked previously at the ways in which a binary tree can be traversed. Two of the traversal methods for a graph are **breadth-first** and **depth-first**.

Breadth-First Graph Traversal

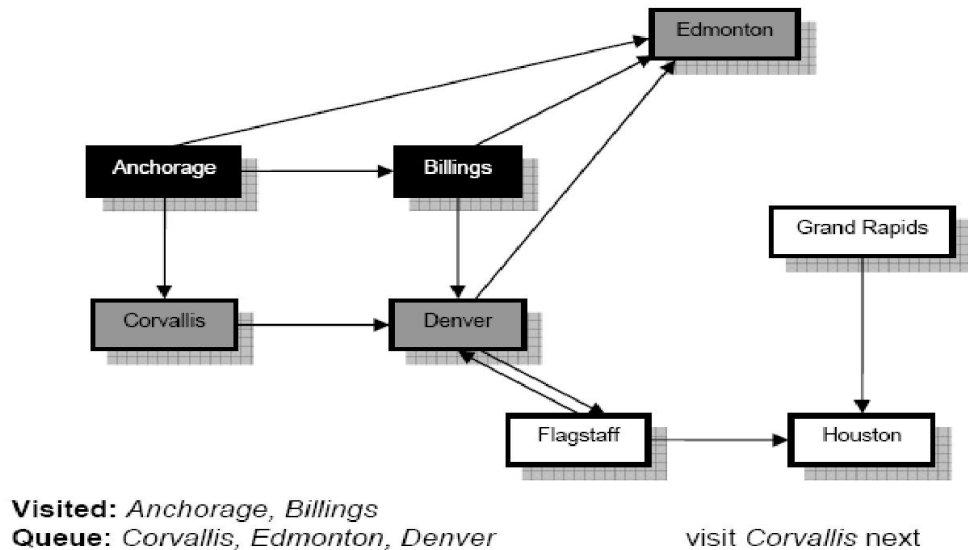
- This method visits all the vertices, beginning with a specified **start vertex**. It can be described roughly as “neighbours-first”.
- No vertex is visited more than once, and **vertices are visited only if they can be reached** — that is, if there is a path from the start vertex.

- 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 already in the queue, or have already been visited.

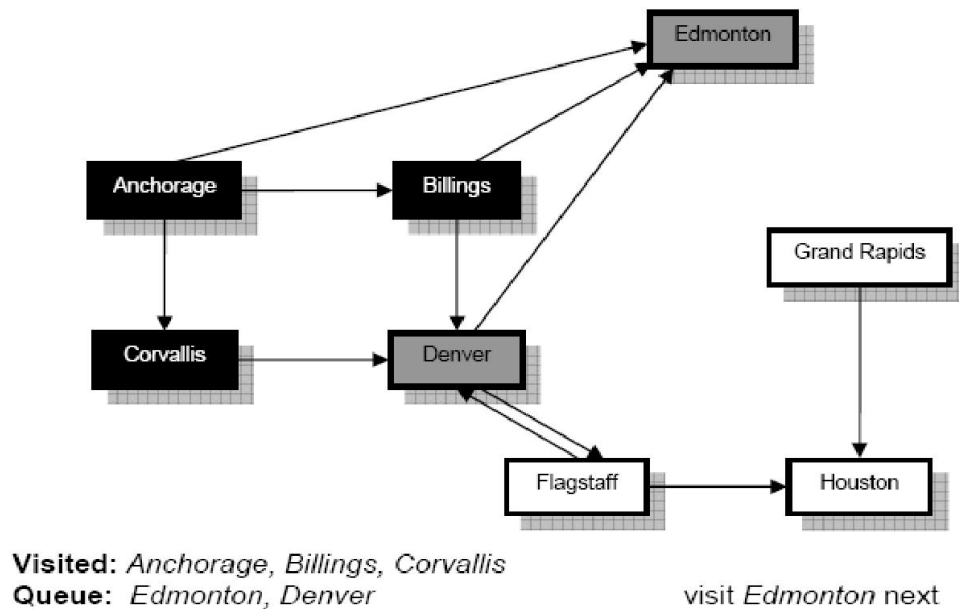
Breadth-First Traversal Example



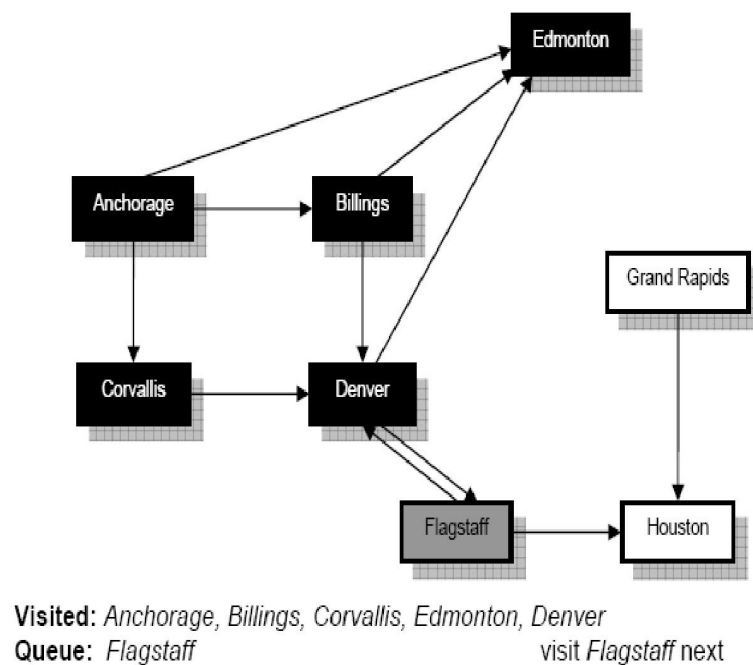
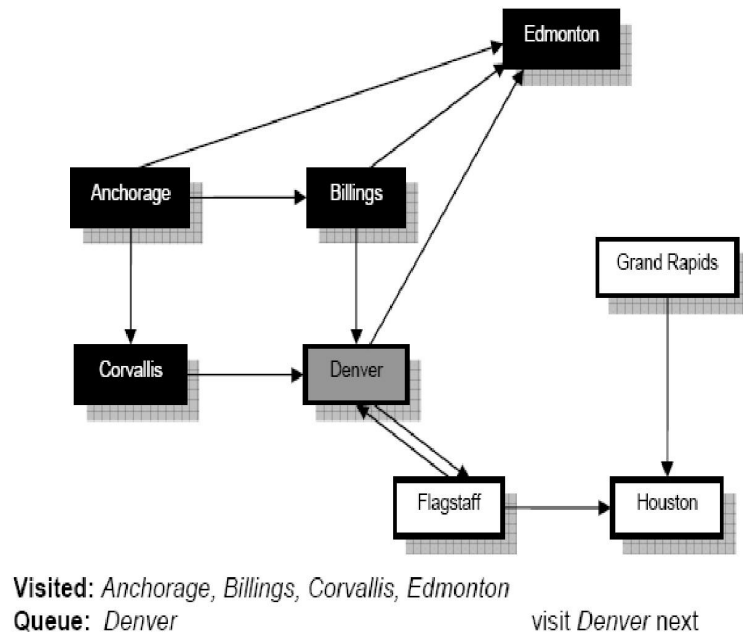
- Neighbors are added to the queue in alphabetical order. Visited and queued vertices are shown as follows:

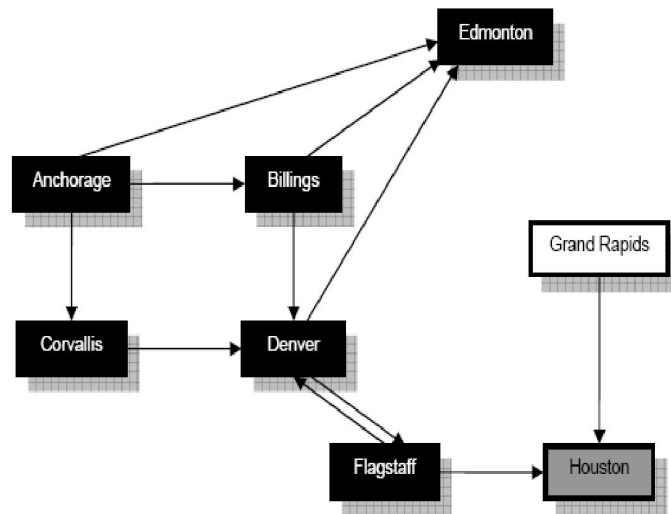


- Note** that we only add Denver to the queue as the other neighbours of Billings are already in the queue.

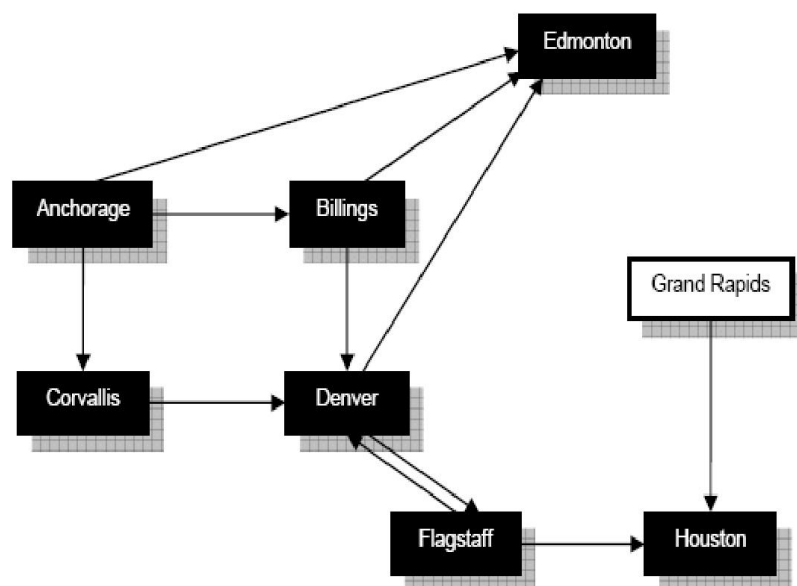


- **Note** that nothing is added to the queue as Denver, the only neighbour of Corvallis, is already in the queue.





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



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

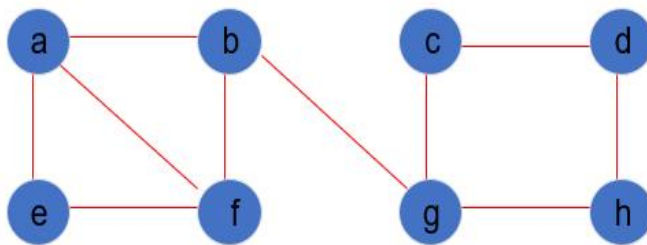
- From the given vertex, visit one of its adjacent vertices and leave others;
- Then visit one of the adjacent vertices of the previous vertex;
- Continue the process, visit the graph as deep as possible until:
 - A visited vertex is reached;
 - An end vertex is reached.

Depth–First Traversal

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



DFS traversal stack:

a
ab
abf
abfe
abf
ab
abg
abgc
abgcd
abgcdh
abgcd
...

DFS tree:

