Data Structures and Algorithms

UBCO Master of Data Science – DATA 532

What are we going to learn today ?

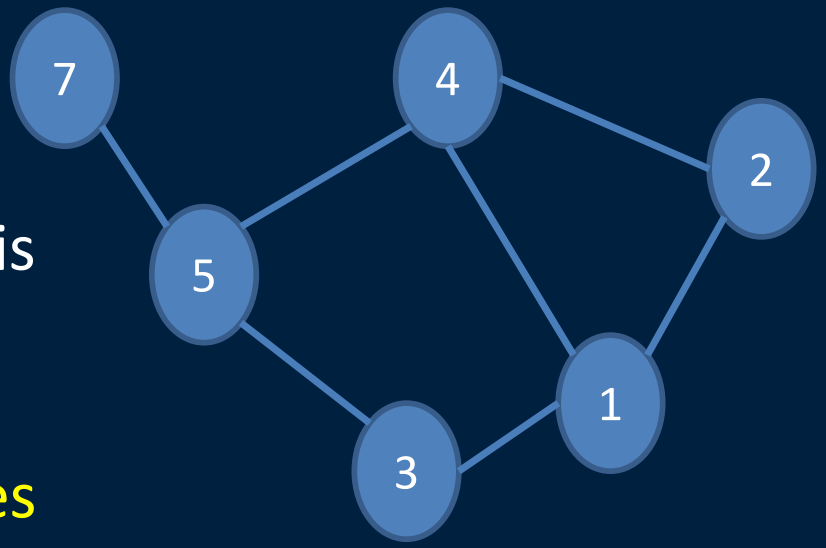
- New data structure called Graphs
- Terminology used in Graphs
- Different types of Graphs
- Representation of Graphs for implementation purposes
- Graphs traversal:
  - BFS
  - DFS

Graphs

Trees are limited in that they can only represent hierarchically structured information (e.g. no arcs between siblings).

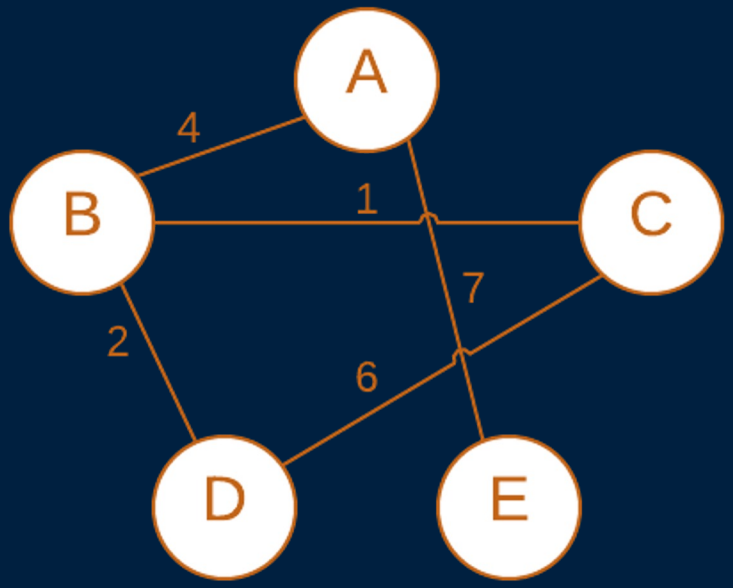
A graph is a generalization of a tree that is much less constrained in the type of relationships it can represent.

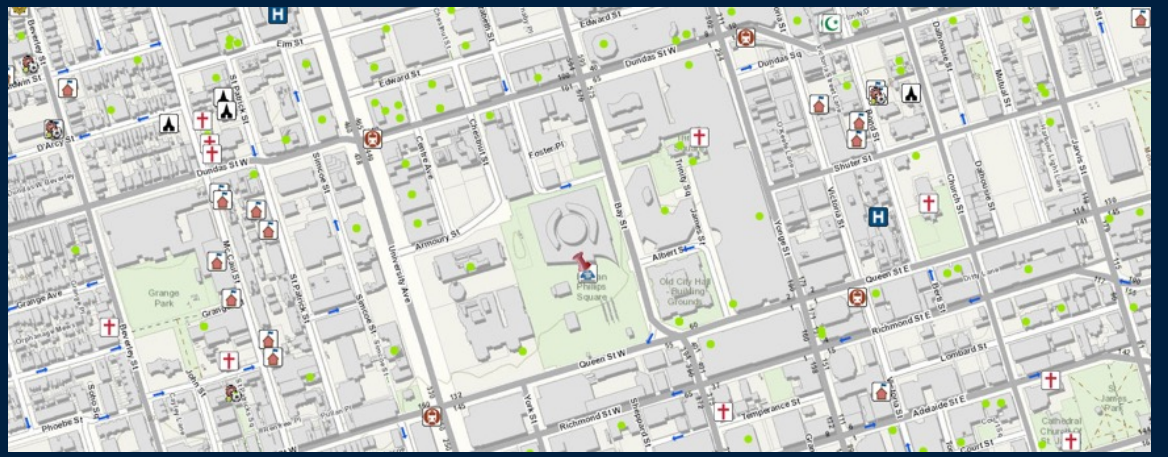
Informally, a graph is a collection of nodes and edges (the connections between them)



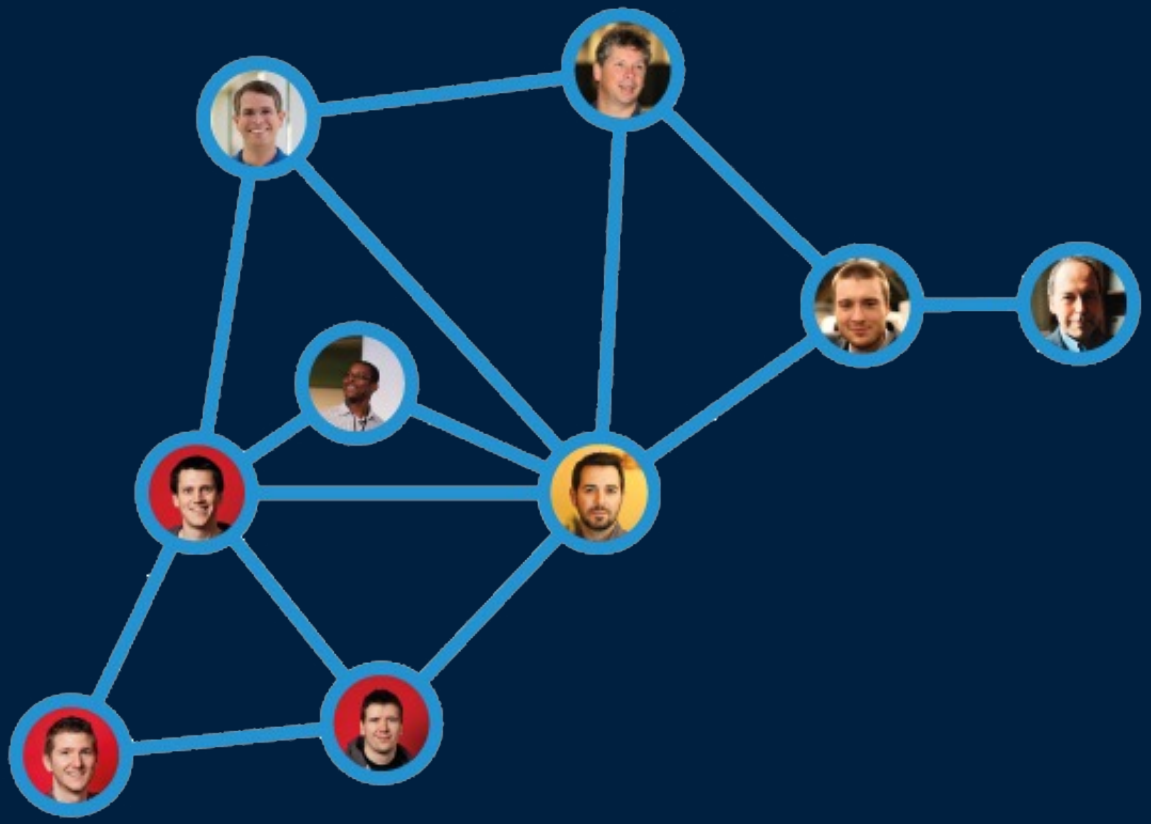
Practical Example of Graphs

A salesman spends his time visiting *n* cities (nodes) cyclically. In one tour he visits each city just once , and finishes where he started. In what order should he visit them to minimize the distance travelled?





Another Example: Social Networks (Facebook, LinkedIn…)



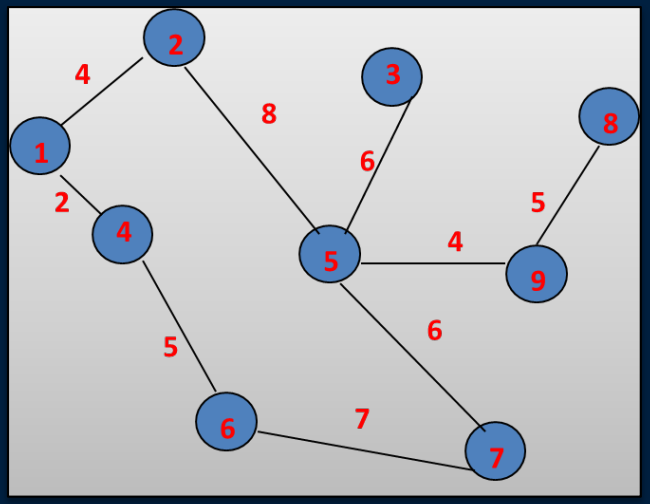
Nodes and Edges in Graphs

Consider the following applications:

1) Course planning:
   - Nodes = courses
   - edges = prerequisites (e.g. order).

2) Circuit analysis:
   - Nodes = components
   - edges = wire connections.

3) Game playing:
   - Nodes = board state
   - edges = moves.



4) Route finding: (path between node 1 and node 8. What is the shortest distance?)
   - Nodes = towns
   - edges = roads

Graphs – Basic Concepts

- **Basic definitions: vertices and edges**
- **More definitions: paths, simple paths, cycles, loops**
- **Connected and disconnected graphs**
- **Spanning trees**
- **Complete graphs**
- **Weighted graphs and networks**
- **Graph representation**
  - Adjacency matrix

Vertices and Edges

Definition: A graph is a collection of vertices (nodes) and edges

Vertices: can have names and properties

Edges: connect two vertices,

can be labeled,

can be directed

Adjacent vertices: there is an edge between the vertices

More Definitions

Order of graph : The number of vertices, or the ***cardinality* of *V*,** is called the ***order of graph*** and devoted by $|V|$

Size of graph: The number of edges, ***The cardinality of E***, called the ***size of graph***, is denoted by $|E|$

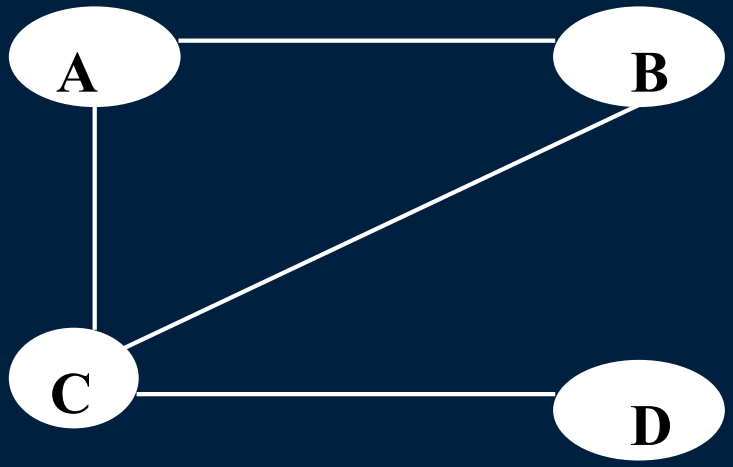
Degree of a vertex: The number of edges incident to a vertex is known as the degree of the vertex.

Example

Graph1

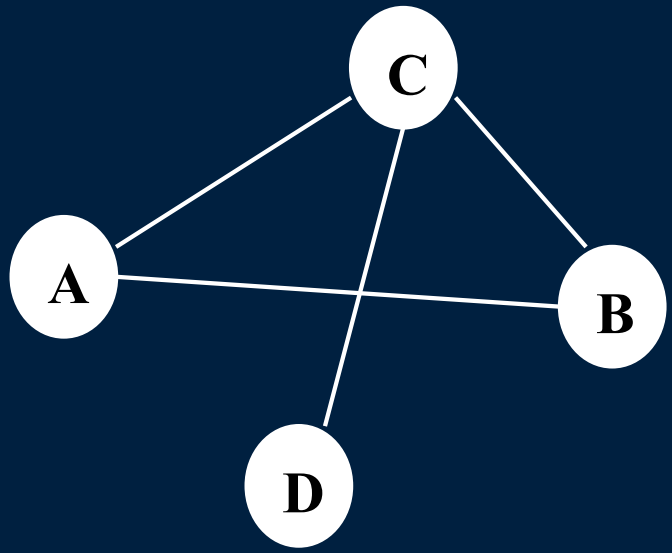
Vertices: A,B,C,D

Edges: AB, AC, BC, CD

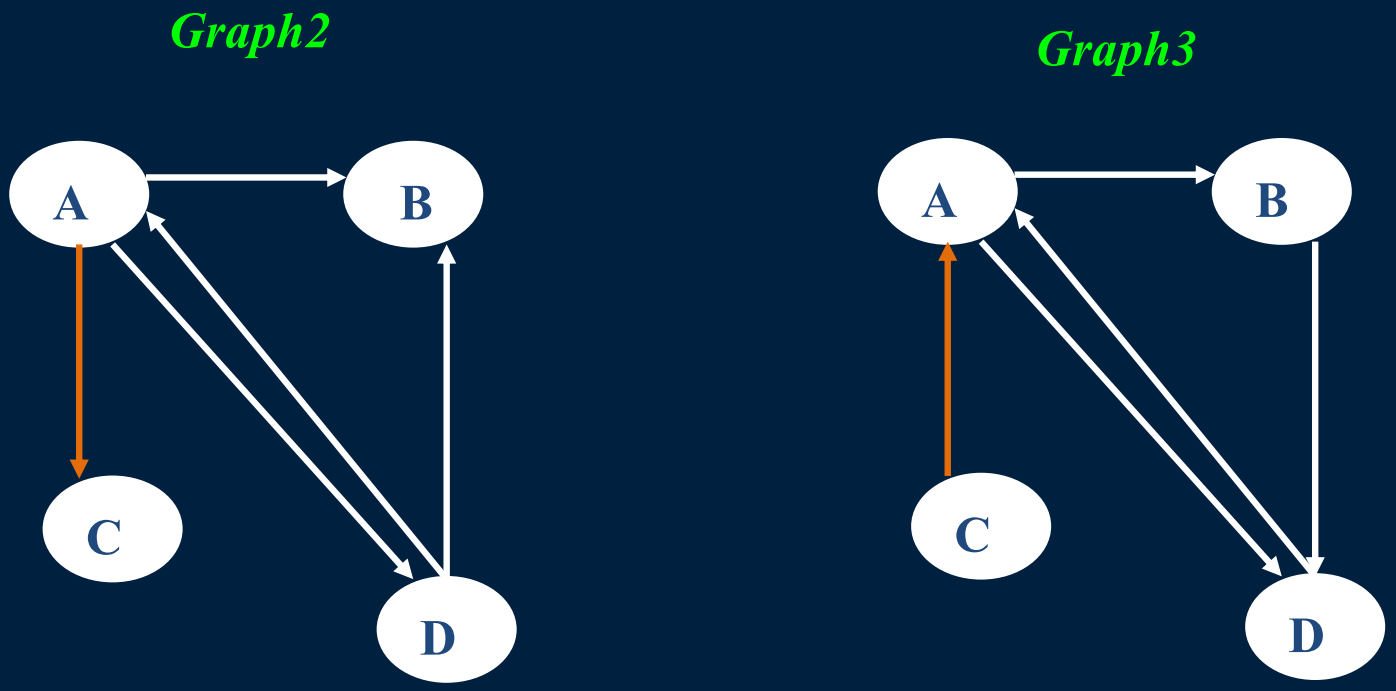


What do you notice in these two graphs:

Two ways to draw the same graph

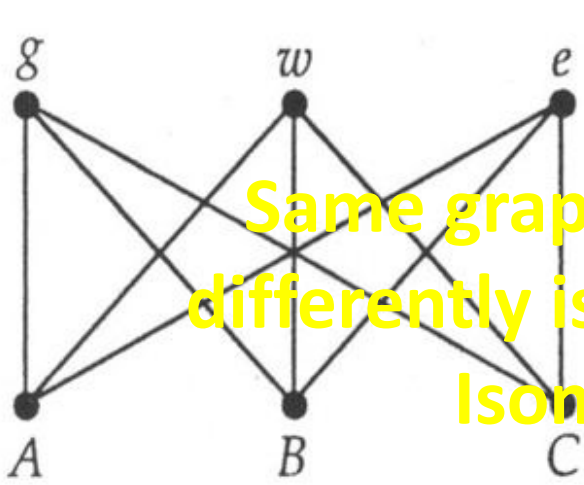


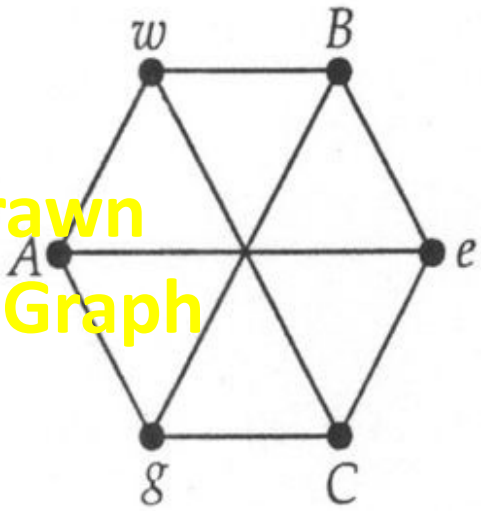
Directed and undirected graphs



These are two different directed graphs

Are these two graphs the same ?





Same graphs when drawn differently is known as Graph Isomorphism

A) Yes

B) No

C) Can't say

More definitions : Path

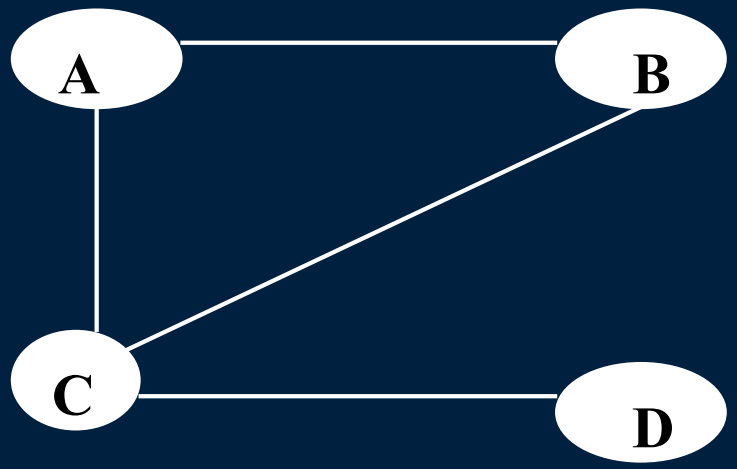
A list of vertices in which successive vertices are connected by edges

A B C

B A C D

A B C A B C A B C D

B A B A C



More definitions : Simple Path

No vertex is repeated.

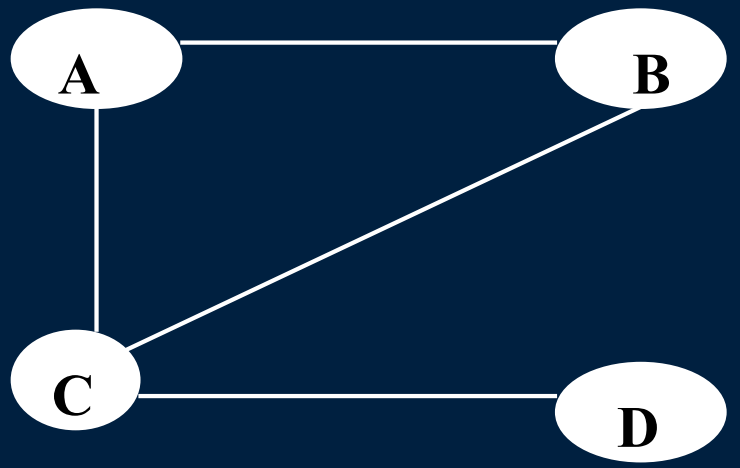
A B C D

D C A

D C B

A B

A B C



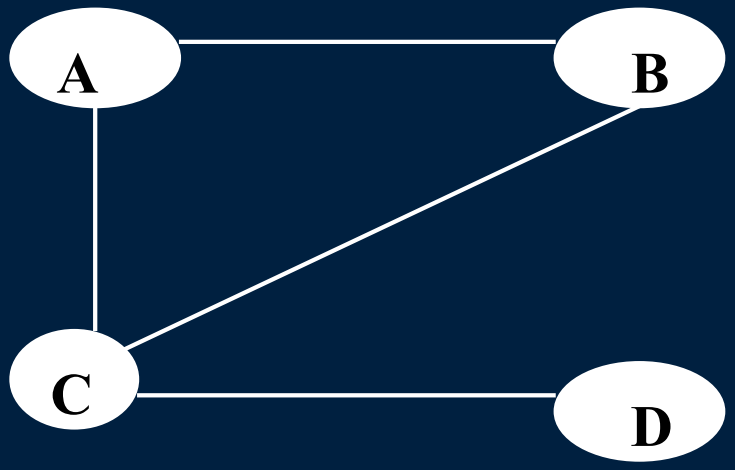
More definitions : Cycle

Simple path with distinct edges, except that the first vertex is equal to the last (one repetition exception)

A B C A

B A C B

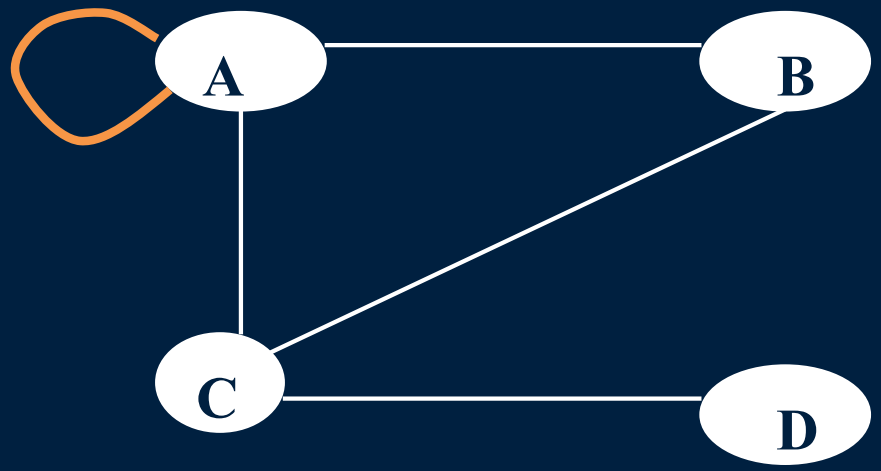
C B A C



A graph without cycles is called **acyclic graph.**

More definitions : Loop

An edge that connects the vertex with itself

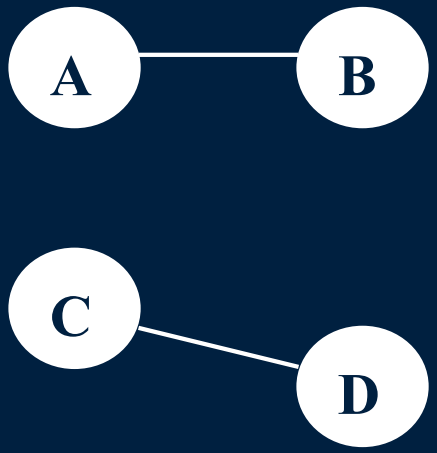


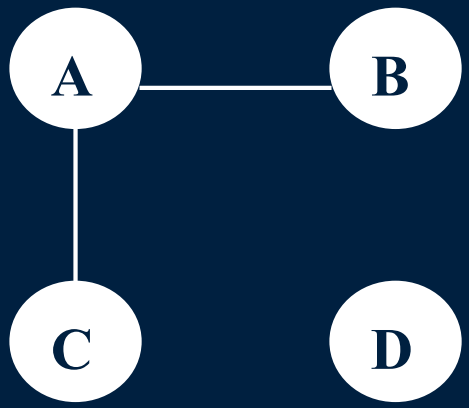
Connected and Disconnected graphs

Connected graph: There is a path between each two vertices

Disconnected graph: There are at least two vertices not connected by a path.

Examples of disconnected graphs:

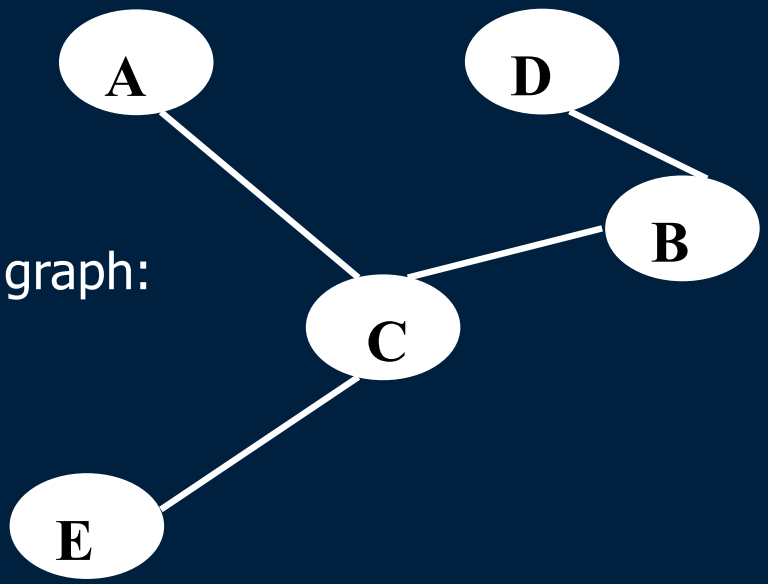




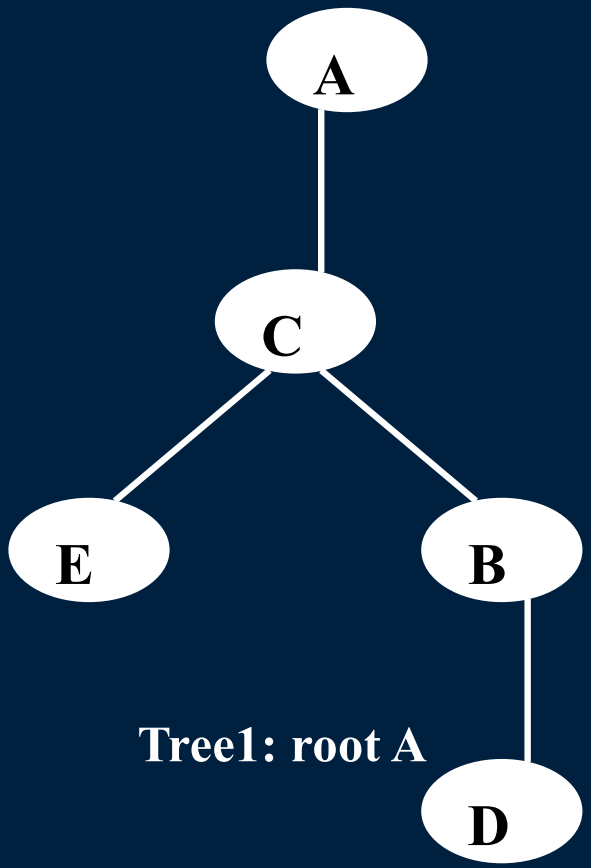
Graphs and Trees

Tree: an undirected graph with no cycles, and a node chosen to be the root

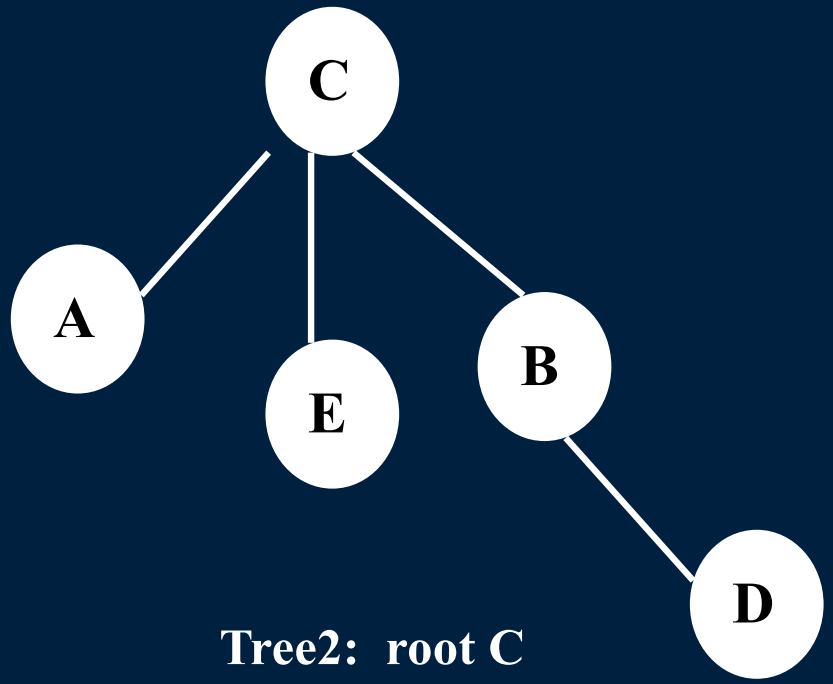
Source graph:



Graphs and Trees



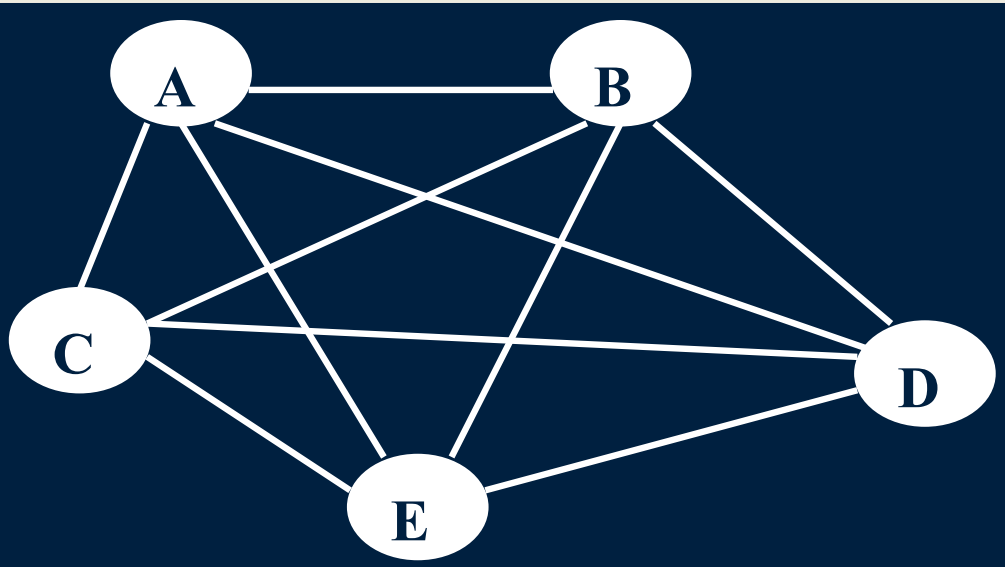
Tree1: root A



Tree2: root C

Complete graphs

Graphs with all edges present – each vertex is connected to all other vertices



A complete graph

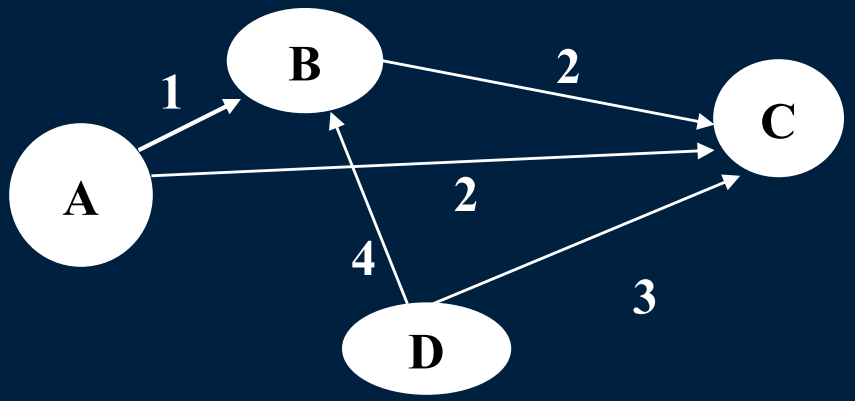
Dense graphs: relatively few of the possible edges are missing

Sparse graphs: relatively few of the possible edges are present

Weighted graphs and Networks

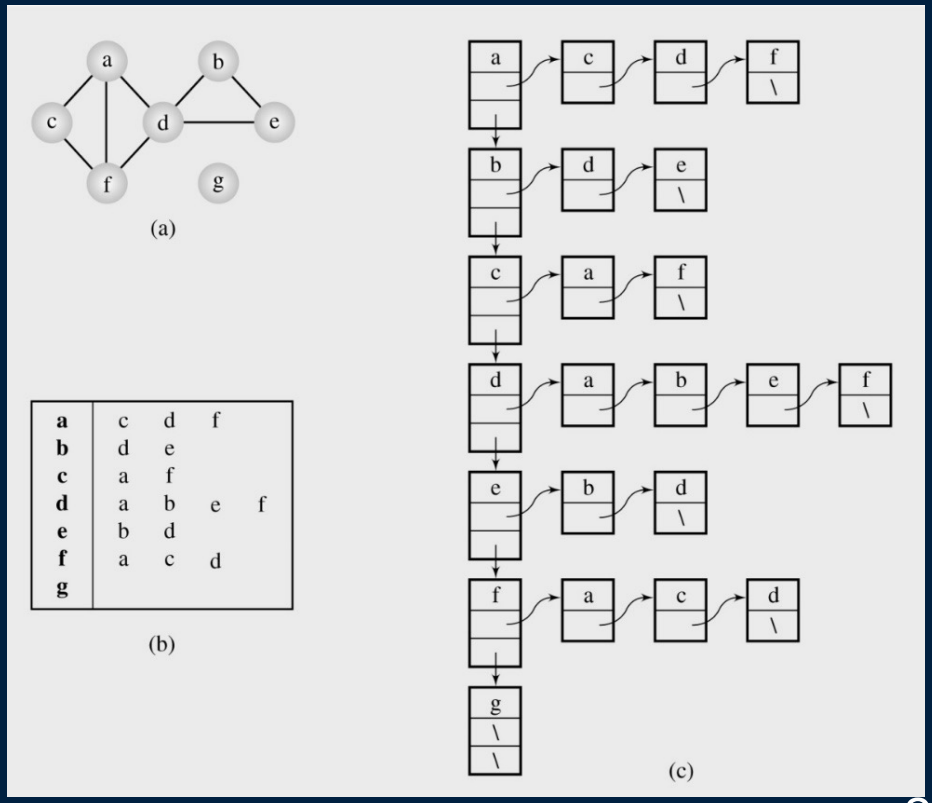
Weighted graphs – weights are assigned to each edge (e.g. road map)

Networks: directed weighted graphs (some theories allow networks to be undirected)



Graph Representation (1) : Adjacency List

- Graphs can be represented in a number of ways
- One of the simplest is an ***adjacency list***, where each vertex adjacent to a given vertex is listed
- This can be implemented as a table (known as a ***star representation***) or a linked list



Graph Representation (2) : Adjacency matrix – undirected graphs

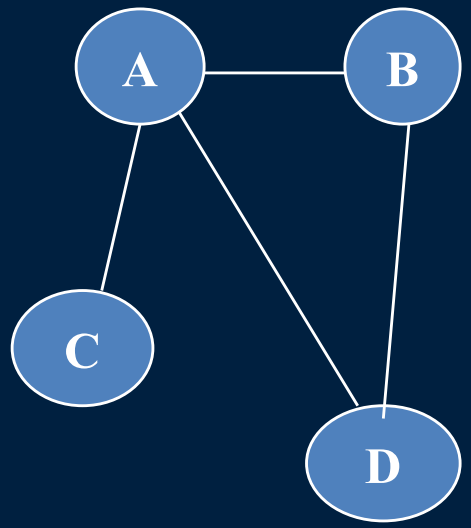
1) Diagonal entries are zero.

2) Adjacency matrix of an undirected graph is symmetric.
A(i,j) = A(j,i) for all i and j.

Vertices: A,B,C,D

Edges: AC, AB, AD, BD

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0



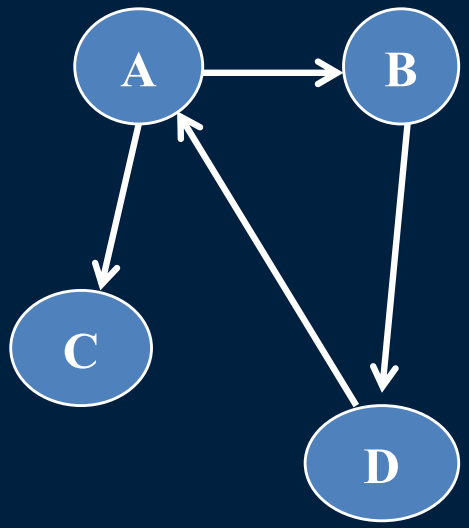
Adjacency matrix – directed graphs

Vertices: A,B,C,D

Edges: AC, AB, BD, DA

	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	0	0	0	0
D	1	0	0	0

- Diagonal entries are zero.



Traversal and Search

Like tree traversals, graph traversals visit each node only once

Similar to trees, there are depth-first and breadth-first traversals.

We can also use traversal to perform a search. The only difference between search and traversal is that search terminates once the goal is found.

However, we cannot apply the same tree traversal algorithms to graphs because of cycles and isolated vertices

Depth First Search (DFS)

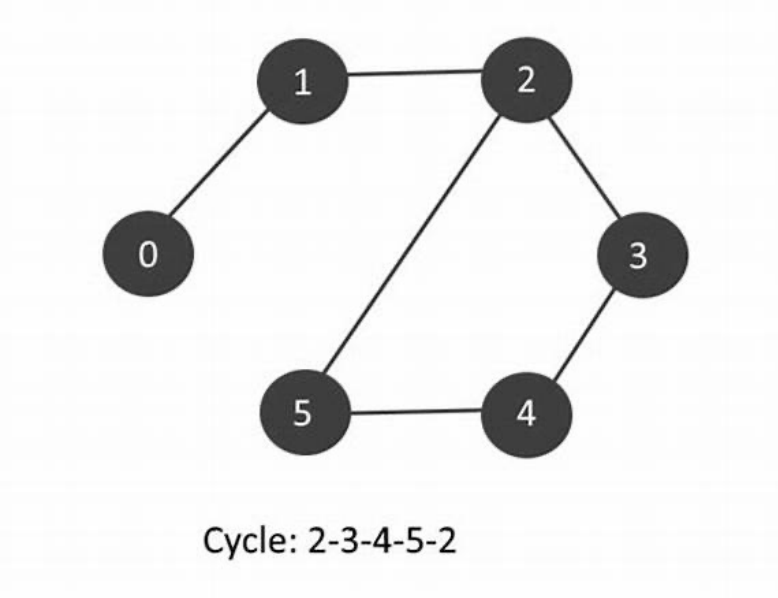
Application: Path Finding

We can specialize the DFS algorithm to find a path between two given vertices u and z.
i) Call DFS(G, u) with u as the start vertex.
ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
iii) As soon as destination vertex z is encountered, return the path as the contents of the stack

Depth First Search (DFS) - Application

Detecting cycle in a graph

A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.



Depth-First Traversal

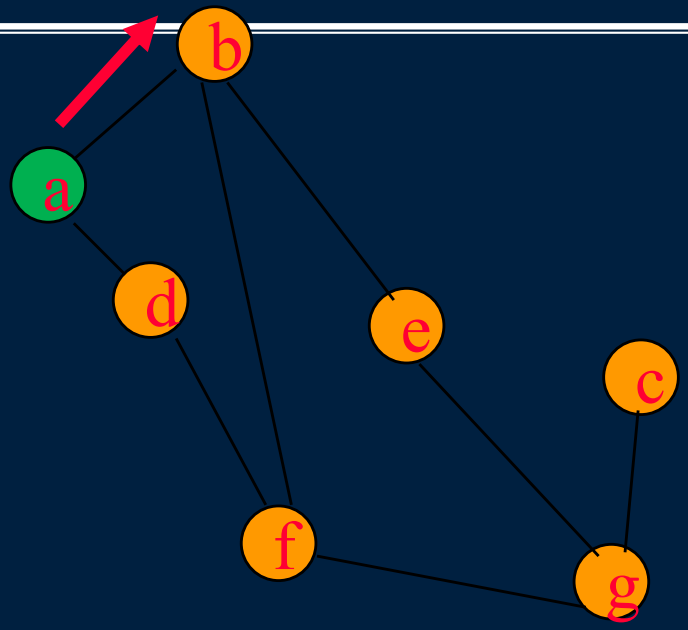
The basic strategy is to visit each node, and then visit all of its unvisited neighbours.

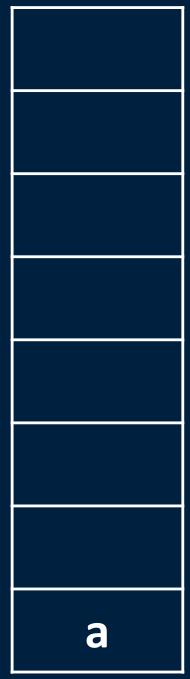
The following is a Pseudocode for depth-first traversal

```
DepthFirstTraversal:
        while(there exists unvisited node v in the graph)
                DFS(v)

DFS (v) :
   Label vertex v as reached.
   for each unreached vertex u adjacent to v
          DFS(u)    //recursive call
```

Depth-First Search Example



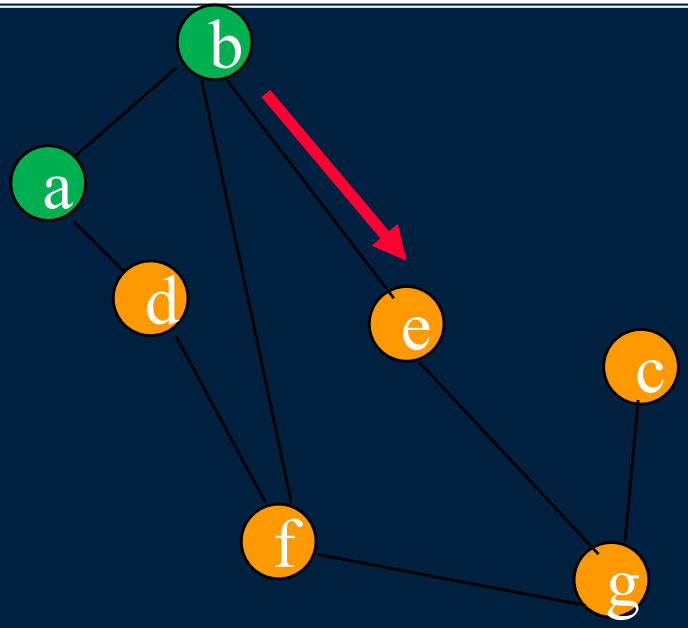


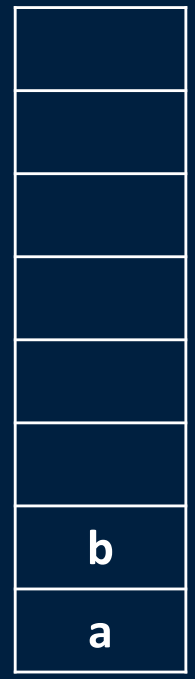
Main calling function called DFS with vertex a

Label vertex a as visited and put it in the stack

We do a depth first search from adjacent vertices either b or d

Suppose that vertex b is selected.

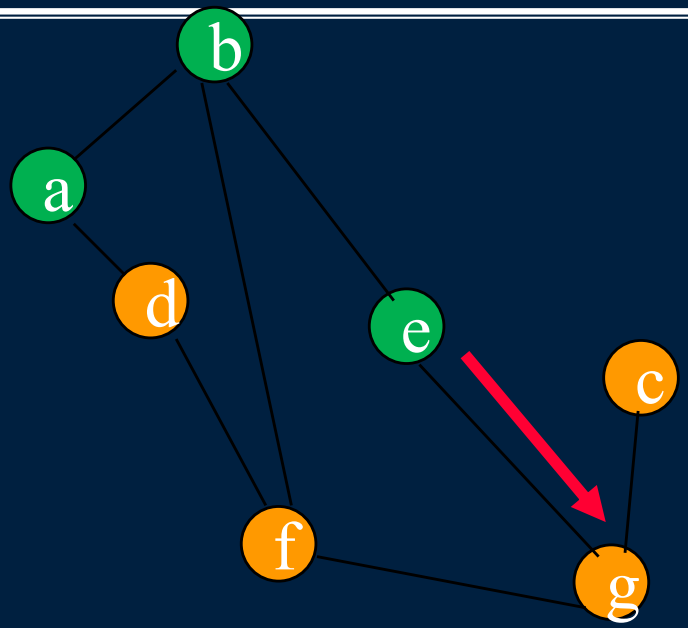


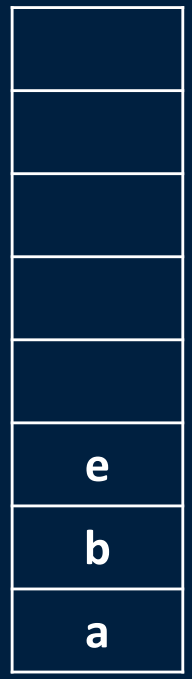


Label vertex b as visited and put it in the stack

We do a depth first search from e or f

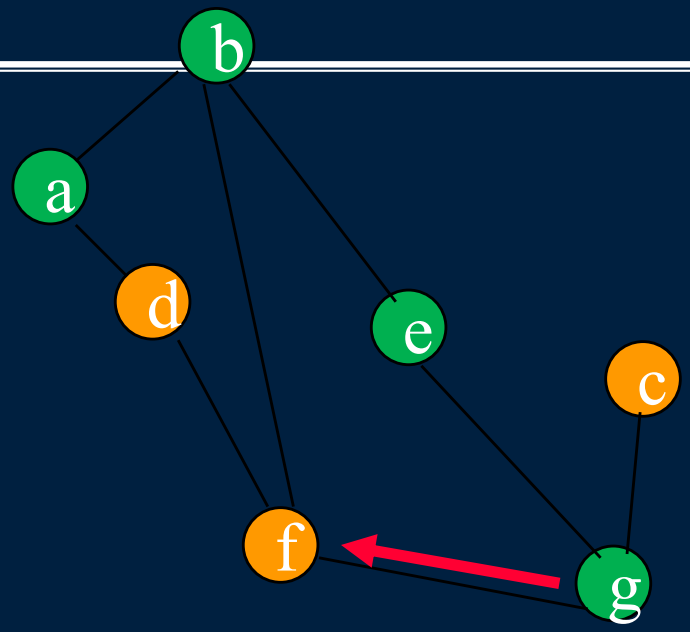
Suppose that vertex e is selected.





Label vertex e as visited and put it in the stack

We do a depth first search from g.

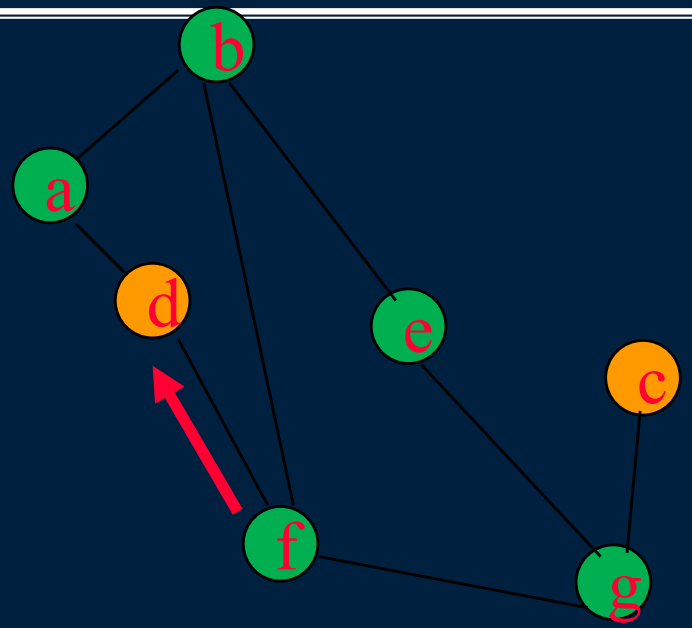


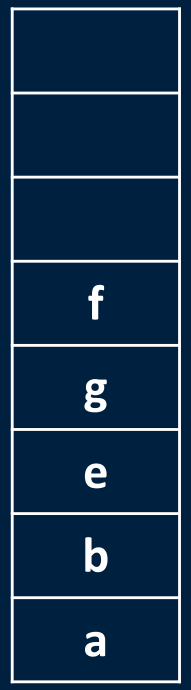
g
e
b
a

Label vertex g as visited and put it in the stack

and do a depth first search from f or c.

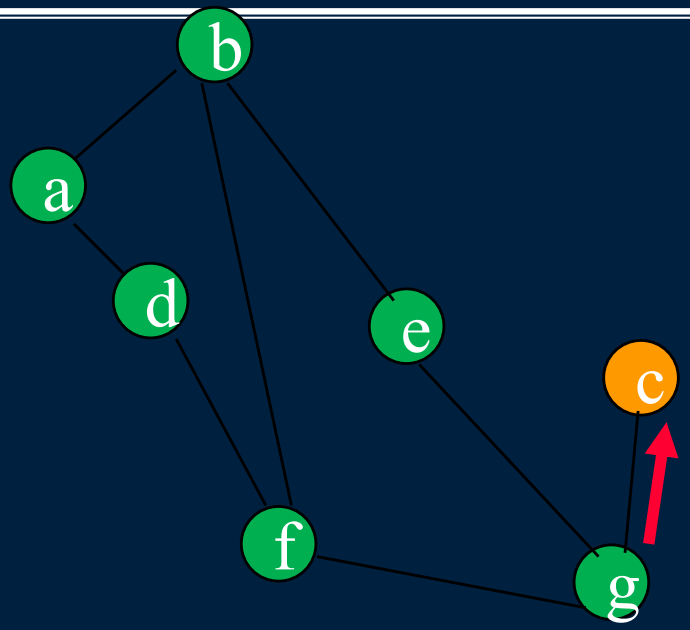
Suppose that vertex f is selected.

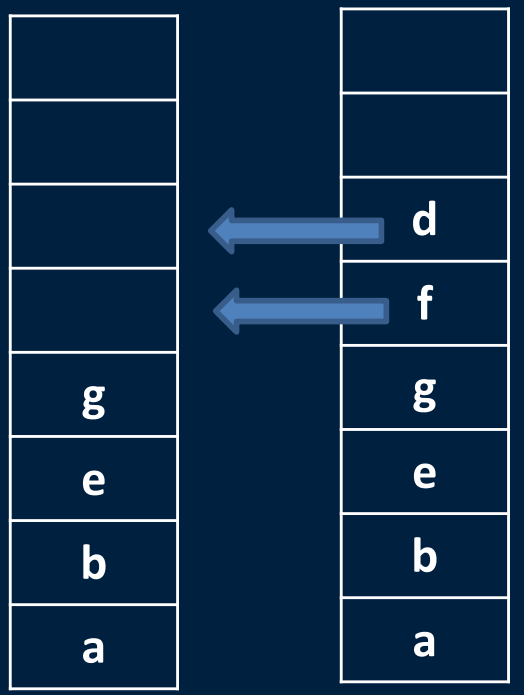




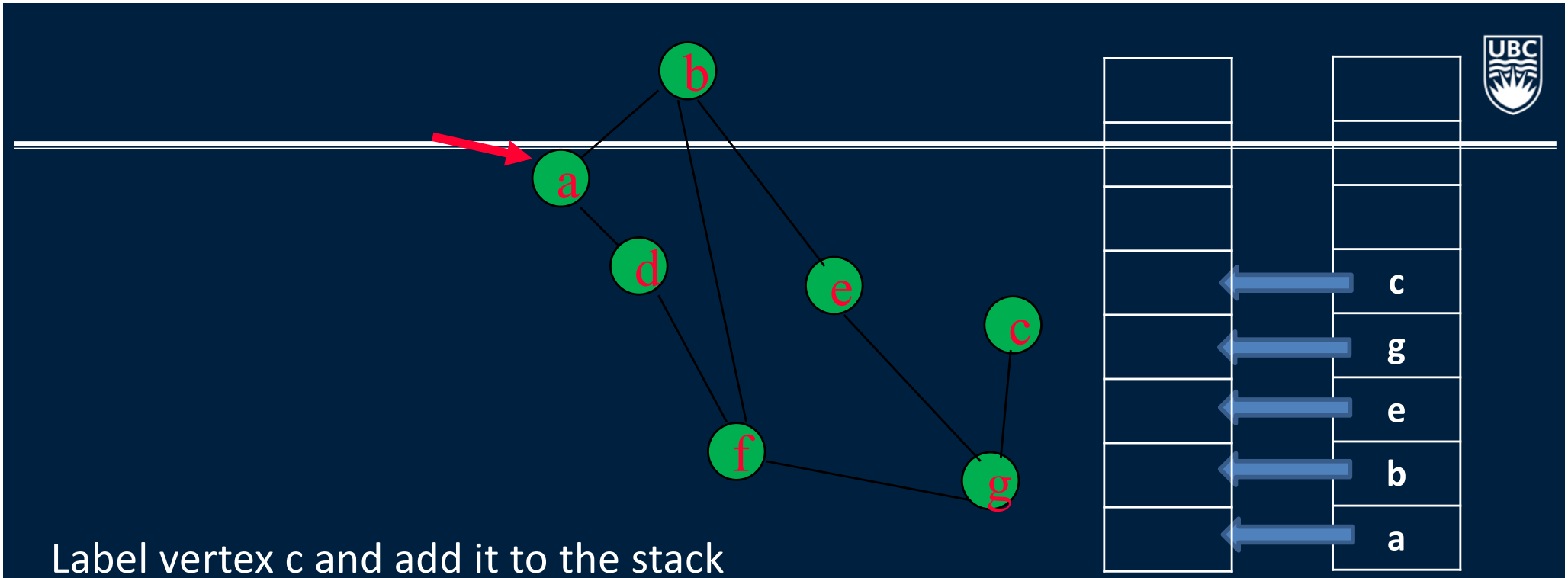
Label vertex f as visited and put it in the stack

and do search from vertex d





Label vertex d as visited and put it in the stack. Then the recursive function backtracks all the way to g until it finds a vertex that has an adjacent node unvisited. This is done by pop the stack



Label vertex c and add it to the stack

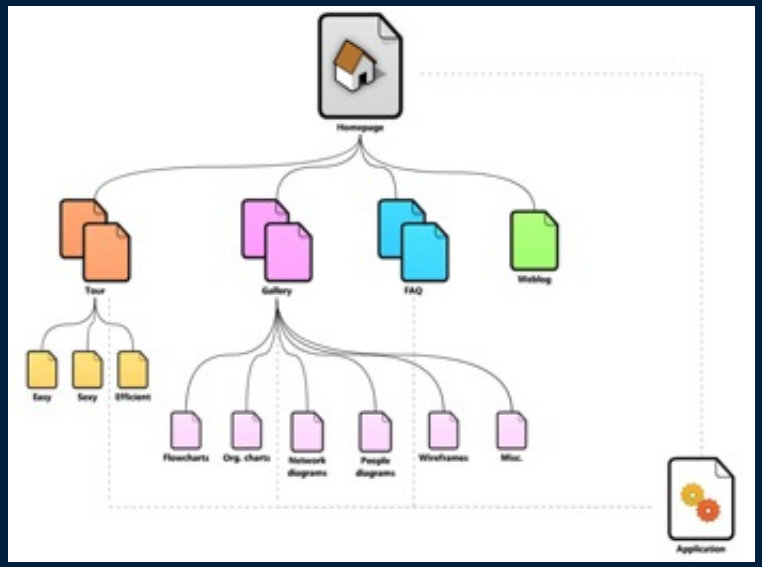
The recursive function then backtracks all the way to vertex a. There are no more nodes to visit, so it returns to the calling function DepthFirstTraversal( ).

The bottom line is that the search stops when the stack is empty

Breadth First Search (BFS)

- Application: **Crawlers in Search Engines**

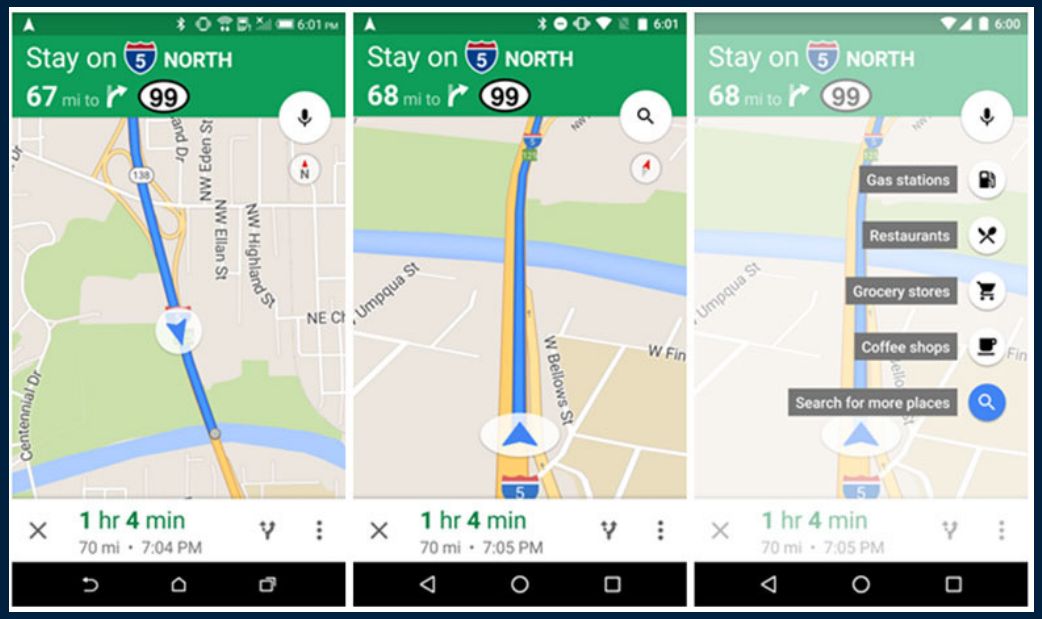
  Crawlers build index using Breadth First. The idea is to start from source page and follow all links from source and keep doing same. Depth First Traversal can also be used for crawlers, but the advantage with Breadth First Traversal is, depth or levels of the built tree can be limited.



BFS - Application

GPS Navigation systems

Breadth First Search is used to find all neighboring locations.



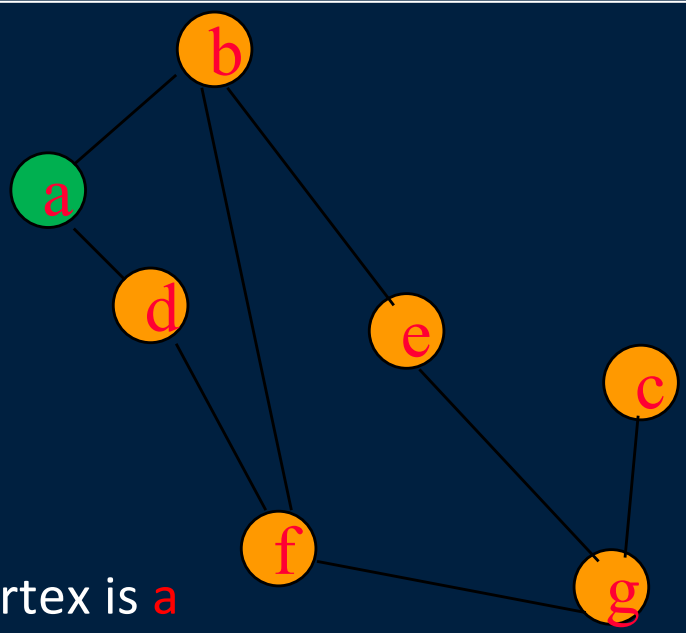
Breadth-First Traversal

Visit start vertex and put into a FIFO queue.

Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.

```
BreadthFirstTraversal:
    create queue Q
    enqueue start node in Q
    mark start node as visited
    while Q is not empty
            dequeue node v from Q
            mark v as visited
            for each adjacent node w to node v
                    if w is not visited
                        enqueue w in Q
                        mark w as visited
```

Breadth-First Example



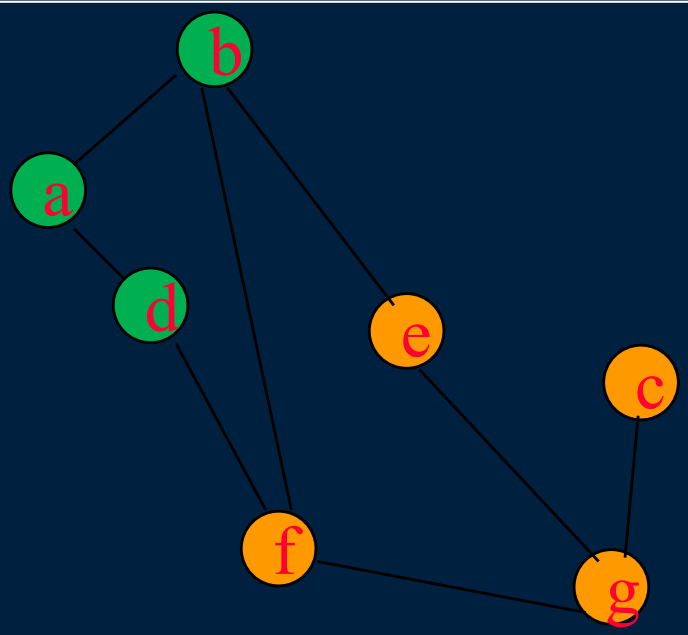
queue Q

a

start vertex is a

Put it in Q and mark it as visited

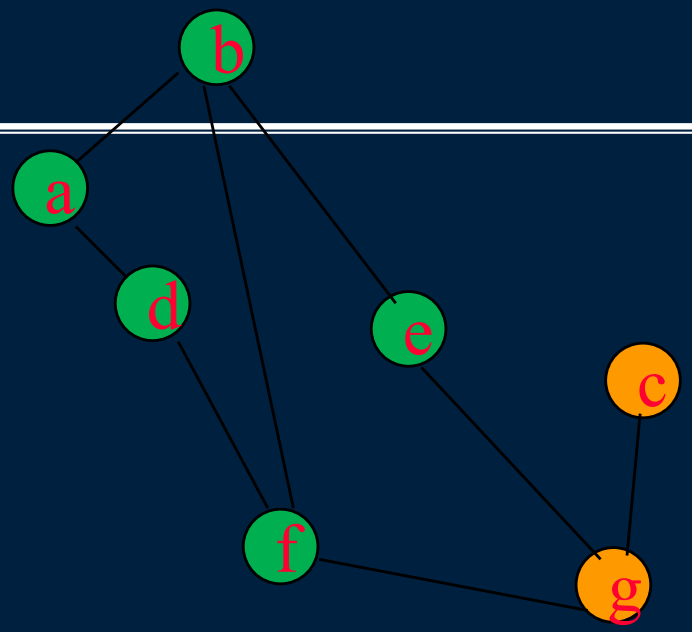
Dequeue a from Q and visit all adjacent vertices of a and put them in Q



queue Q

b d

Dequeue b from Q and visit all adjacent vertices of b and put them in Q

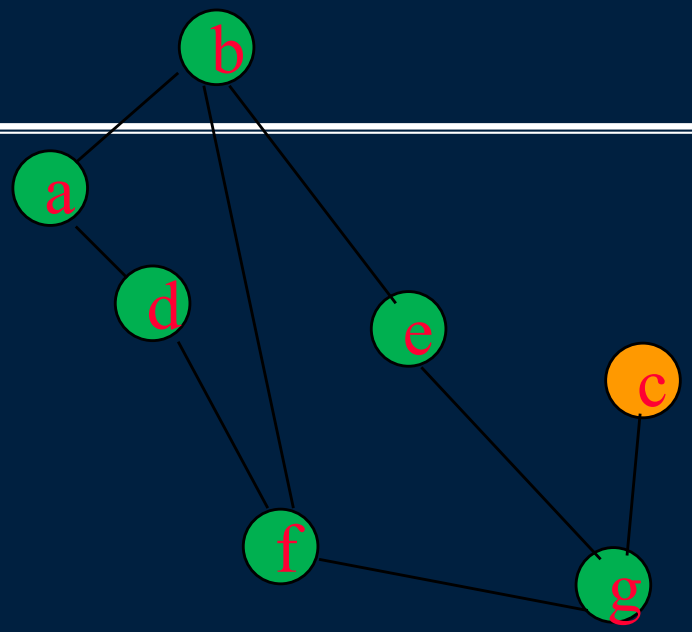


queue Q

d e f

Dequeue d from Q and visit all adjacent vertices of d – no nodes to visit

Dequeue e from Q and visit all unvisited adjacent vertices and put them in Q

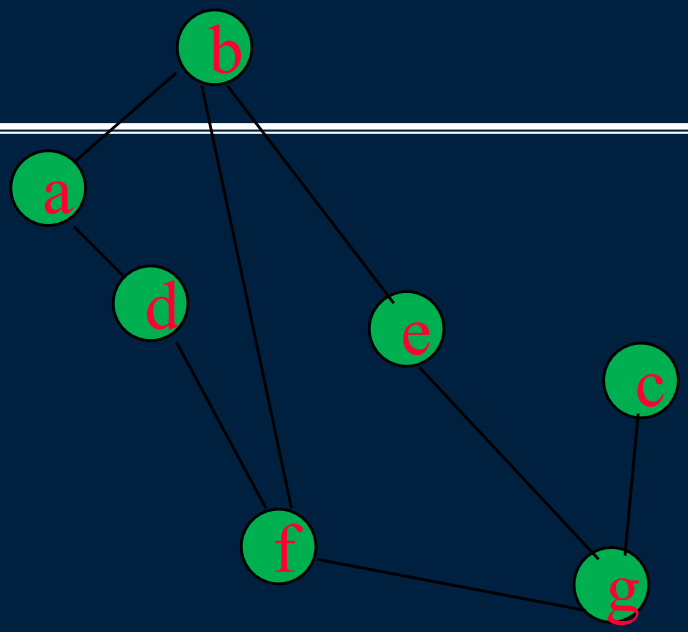


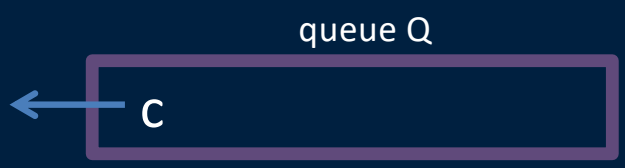
queue Q

f g

Dequeue f from Q and visit all adjacent vertices of d – no nodes to visit

Dequeue g from Q and visit all unvisited adjacent vertices and put them in Q





Dequeue c from Q and visit all adjacent vertices of d – no nodes to visit

Now queue Q is empty, the traversal terminates

Summary

We learnt a new data structure - Graphs

- The notion of Directed and Undirected Graphs
- Various variants of Graphs, Spanning Trees
- Learnt terminology associated with Graphs
- We learned the Adjacency Matrix representation of Graphs
- We learned how to traverse a graph


THE UNIVERSITY OF BRITISH COLUMBIA