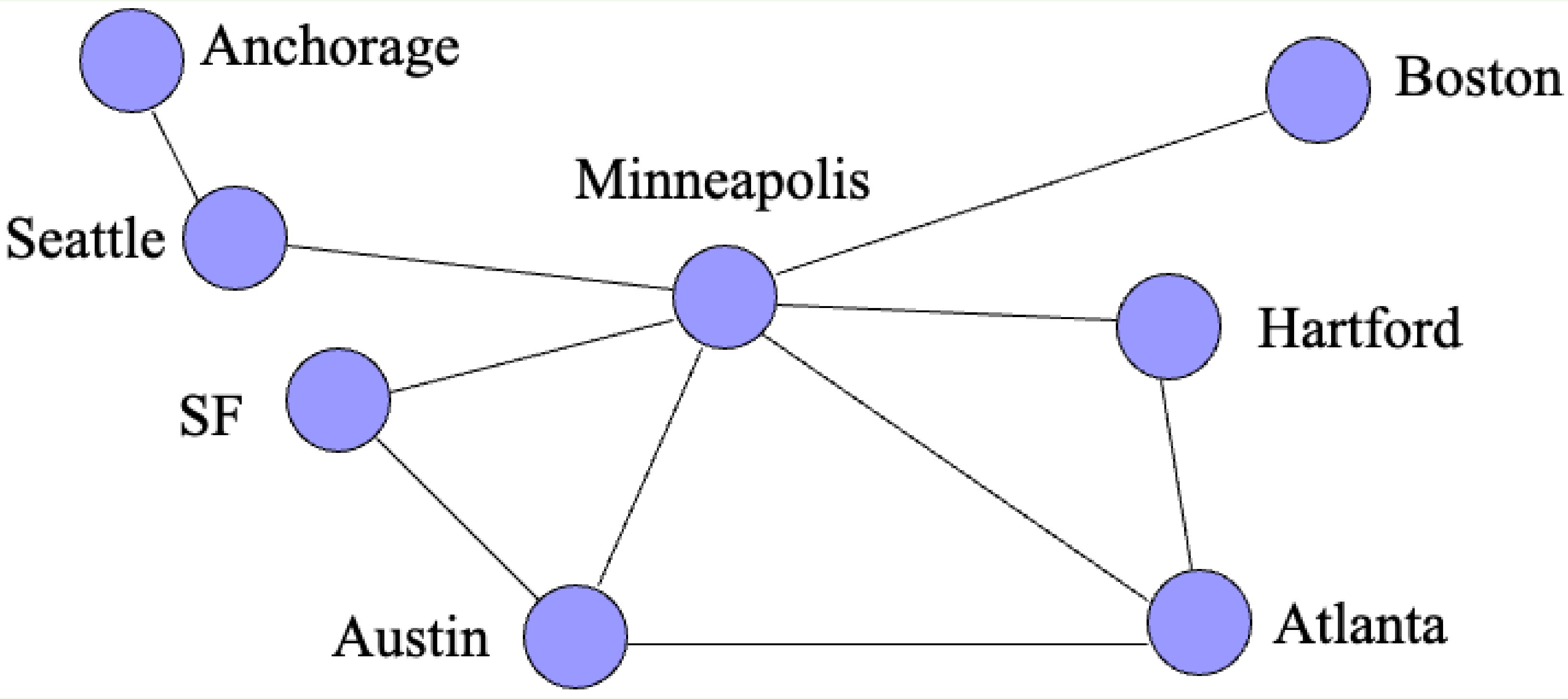
CSCE 2110
Foundations of Data Structures

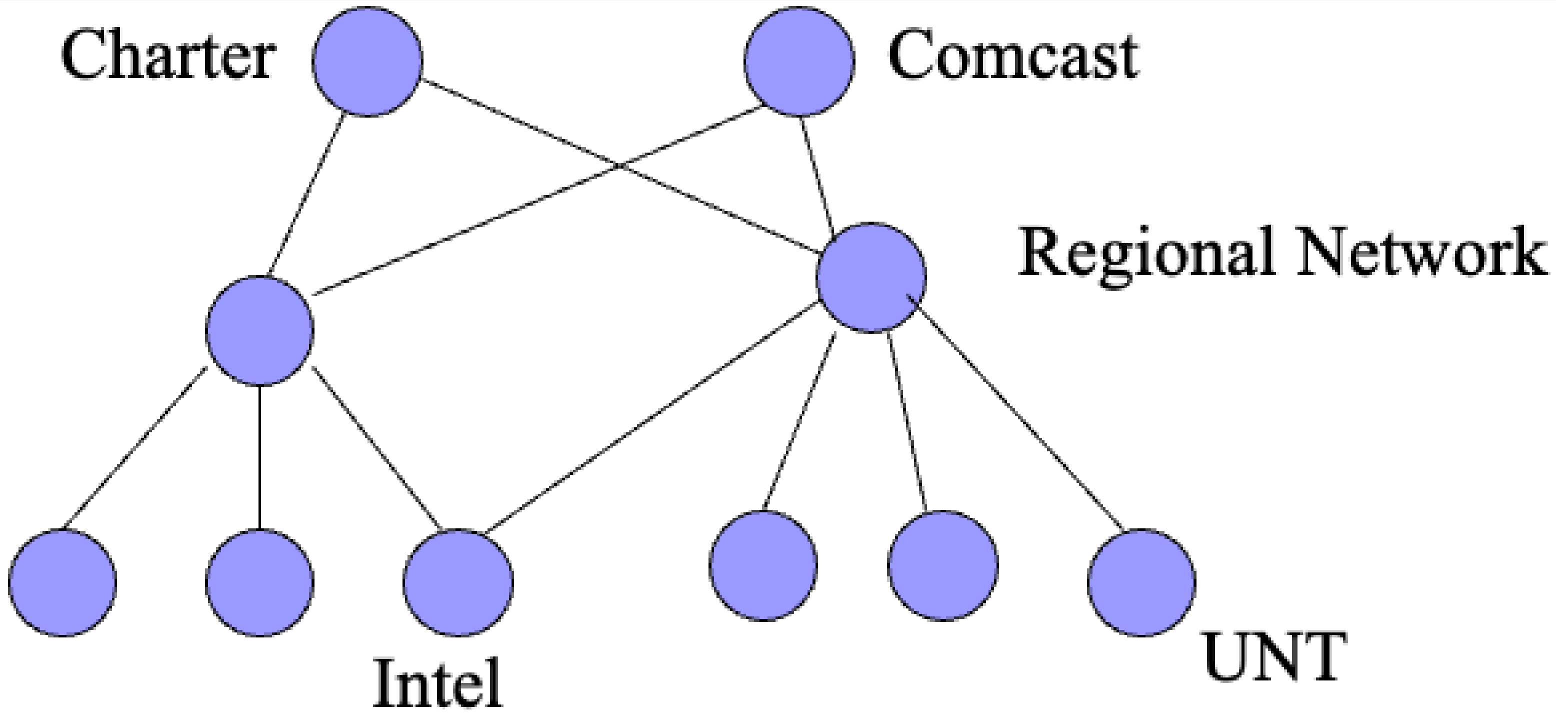
Graph I

Slides borrowed/adapted from Prof. Yan Huang from UNT

Northwest Airline Flight

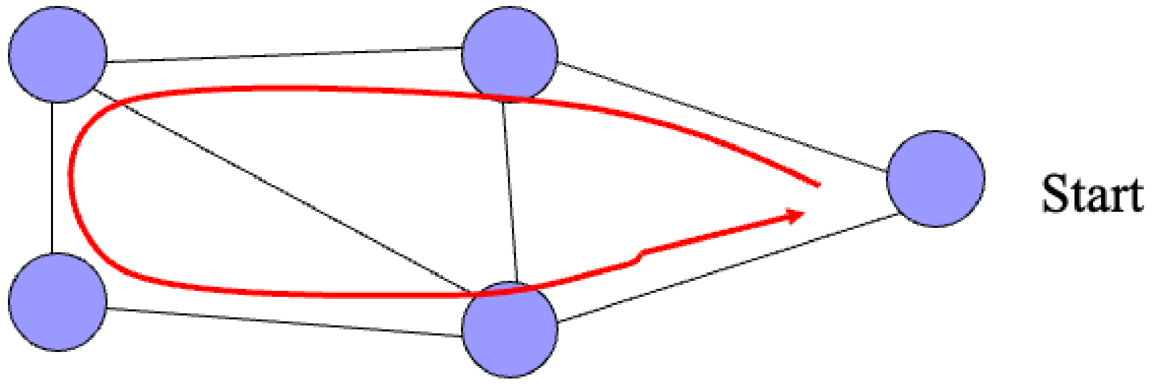


Computer Network Or Internet



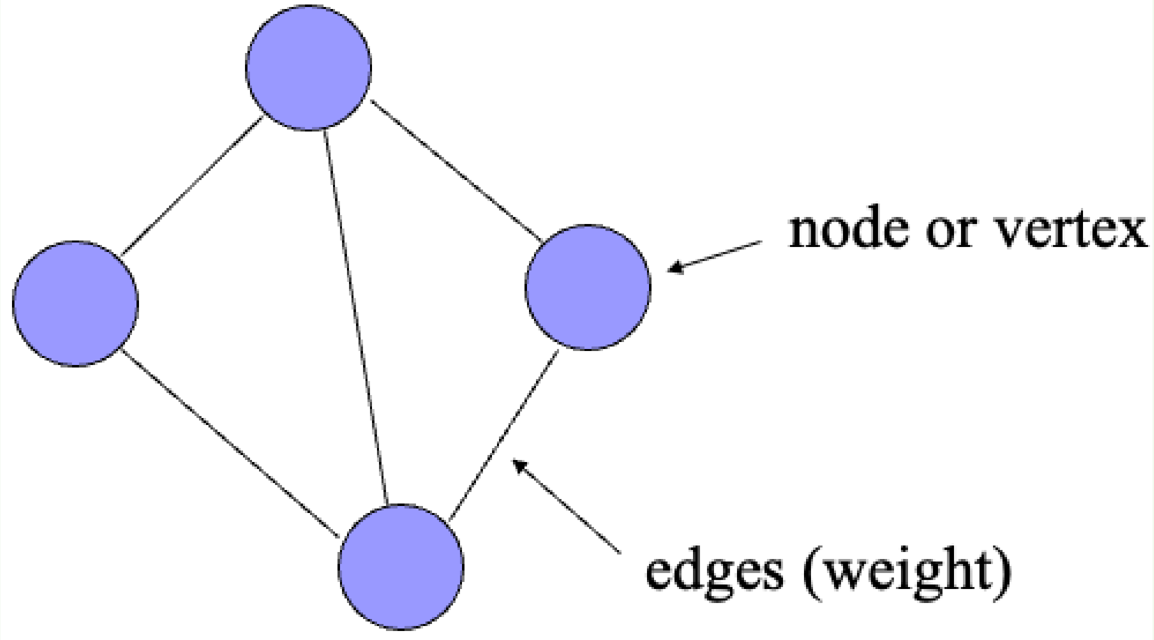
Application

- Traveling Salesman



  - Find the shortest path that connects all cities without a loop.

Concepts of Graphs

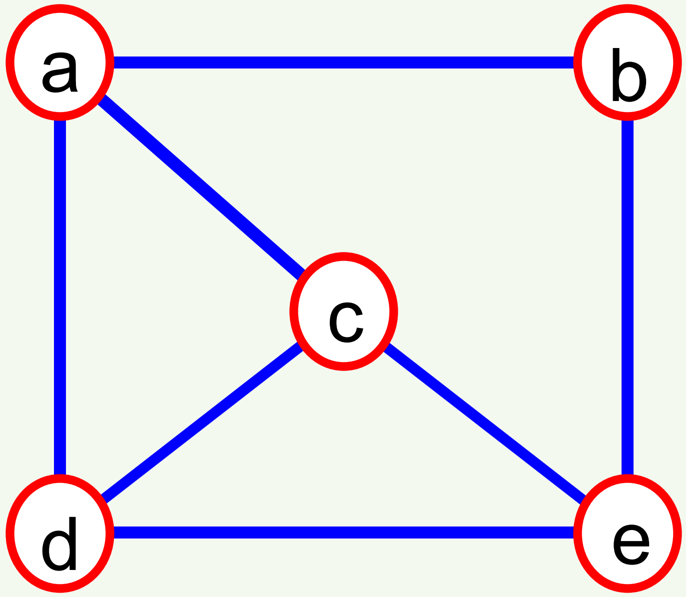


Graph Definition

- A graph G = (V,E) is composed of:

  V: set of vertices (nodes)

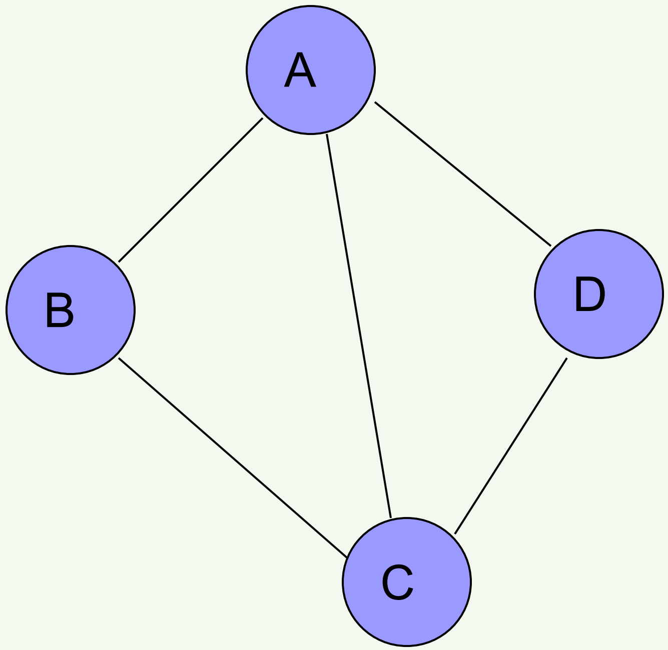
  E: set of edges (arcs) connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:

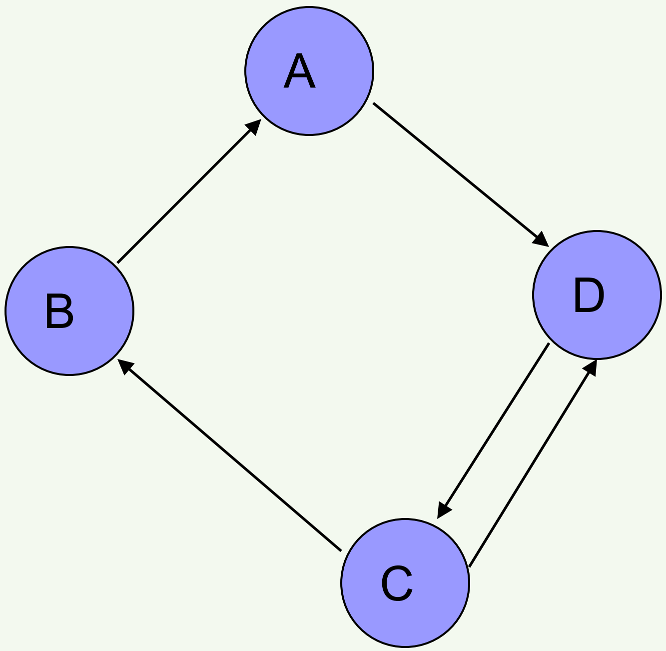


V= {a,b,c,d,e}

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

Undirected vs. Directed Graph





Undirected Graph

- edges have no direction

Directed Graph

- edges have a specific direction from one vertex to another.

Degree of a Vertex

- The degree of a vertex is the number of edges to that vertex

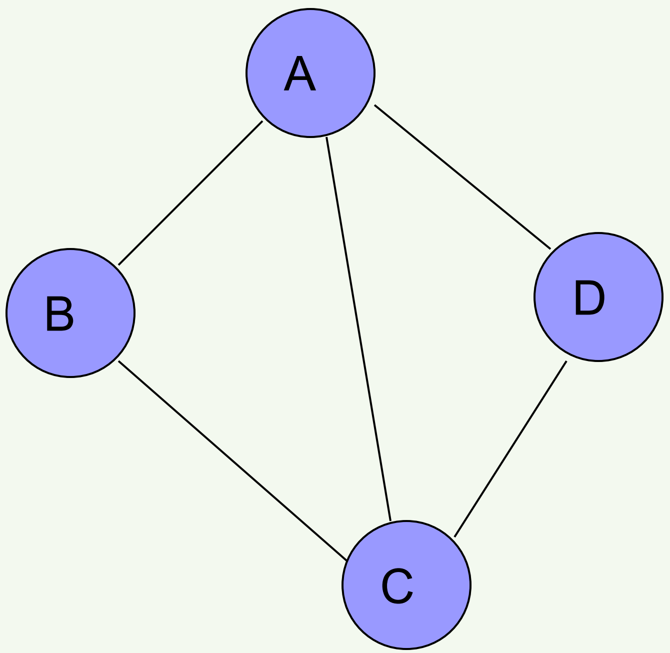
- For directed graph,
  - the in-degree of a vertex *v* is the number of edges that have *v* as the head
  - the out-degree of a vertex *v* is the number of edges that have *v* as the tail

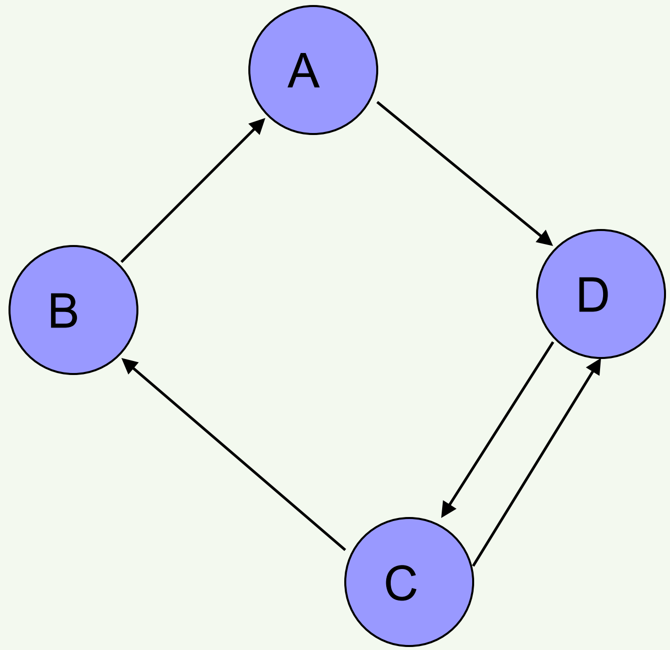
if *di* is the degree of a vertex *i* in a graph *G* with *n* vertices and *e* edges, the number of edges is

$$e = \left(\sum_{0}^{n-1} d_i\right) / 2$$

Hint: Adjacent vertices are counted twice.

Degree of a Vertex





Degree(A)=?

Degree(B)=?

Degree(C)=?

Degree(D)=?

In-degree(A)=? Out-degree(A)=?

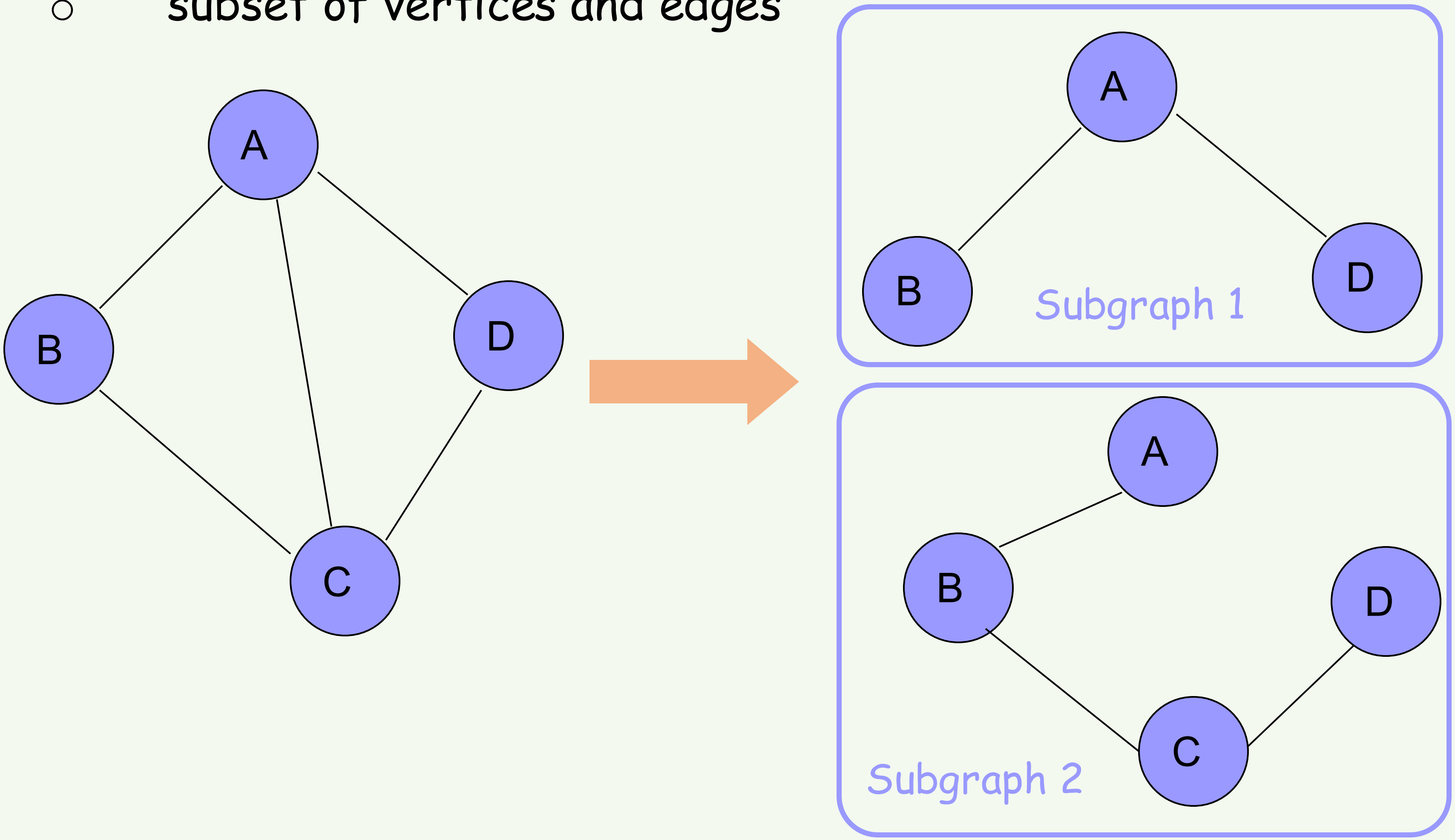
In-degree(B)=? Out-degree(B)=?

In-degree(C)=? Out-degree(C)=?

In-degree(D)=? Out-degree(D)=?

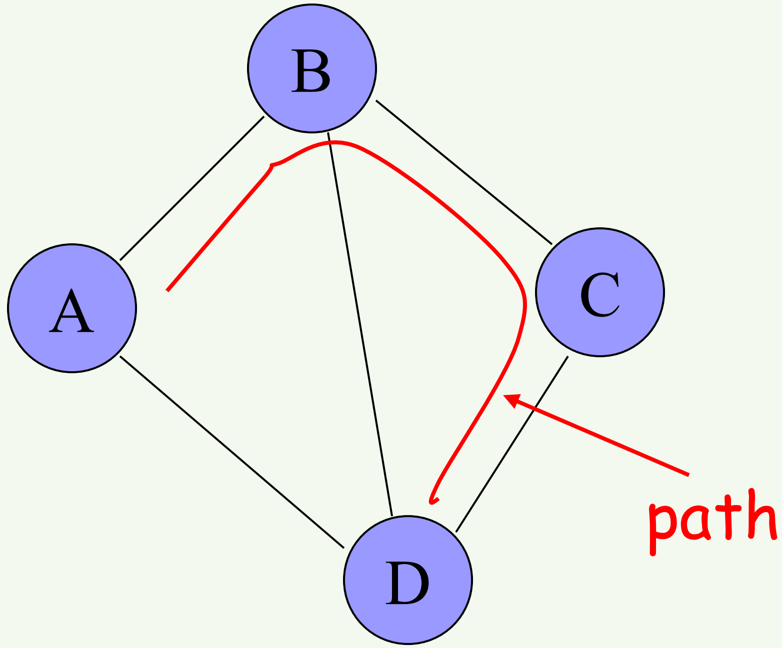
Subgraph

- Subgraph:
  - subset of vertices and edges



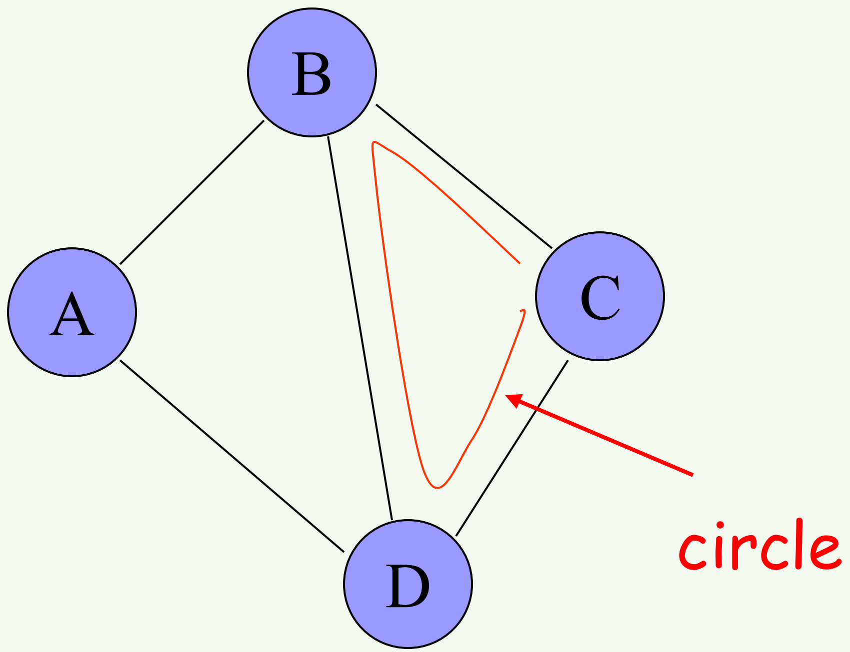
Simple Path

- A simple path is a path such that all vertices are distinct, except that the first and the last could be the same.
  - *ABCD* is a simple path

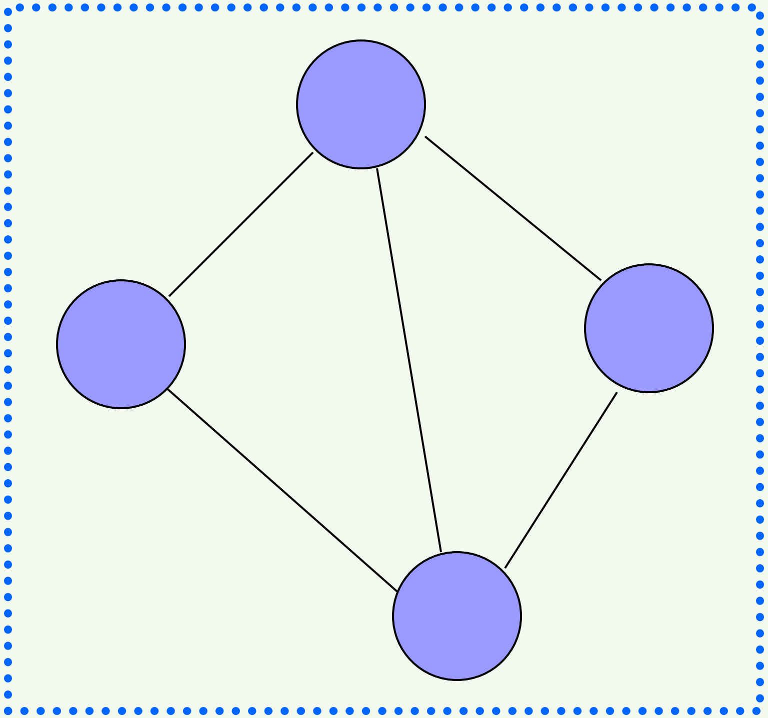


Cycle

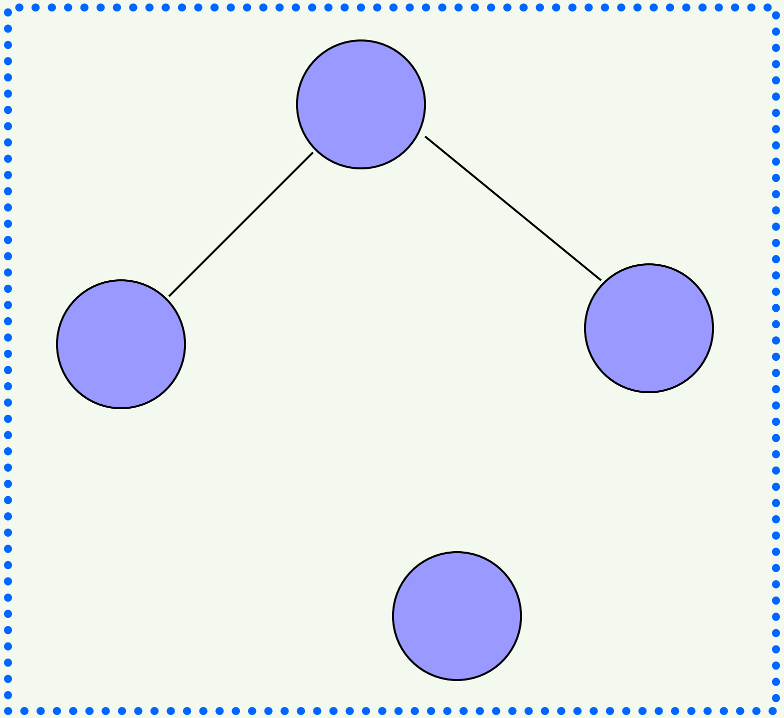
- A cycle is a path that starts and ends at the same point. For undirected graph, the edges are distinct.
  - *CBDC* is a cycle



Connected vs. Unconnected Graph



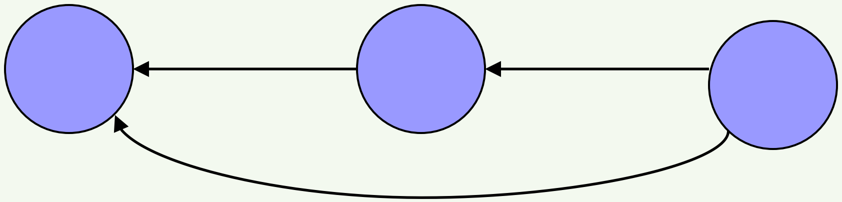
Connected Graph



Unconnected Graph

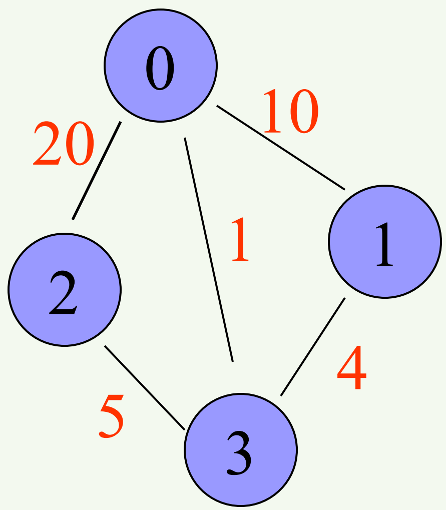
Directed Acyclic Graph

- Directed Acyclic Graph (DAG) : directed graph without cycle



Weighted Graph

- Weighted graph: a graph with numbers assigned to its edges
- Weight: cost, distance, travel time, hop, etc.



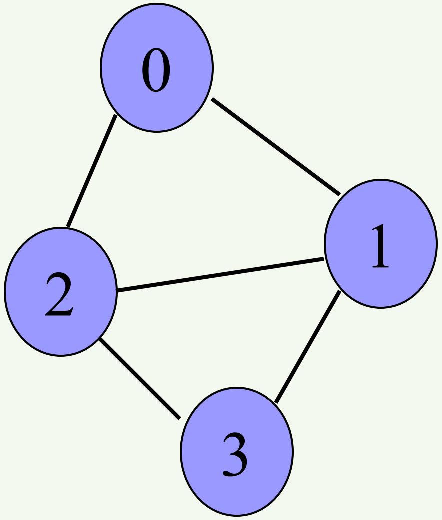
Representation Of Graph

- Two representations
  - Adjacency Matrix
  - Adjacency List

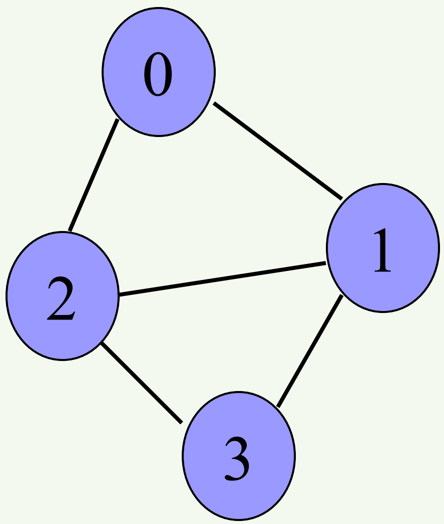
Adjacency Matrix

- Assume N nodes in graph
- Use 2D Matrix A[0…N-1][0…N-1]
  - if vertex i and vertex j are adjacent in graph, A[i][j] = 1,
  - otherwise A[i][j] = 0
  - if vertex i has a loop, A[i][i] = 1
  - if vertex i has no loop, A[i][i] = 0

Example of Adjacency Matrix



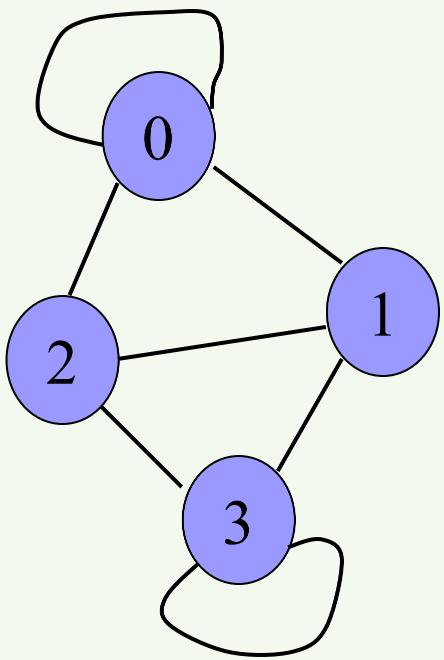
Example of Adjacency Matrix



A[i][j]	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

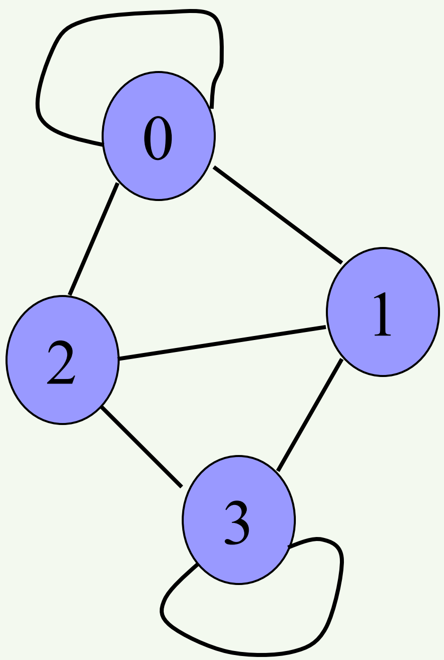
So, Matrix A = $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & \end{pmatrix}$

Example of Adjacency Matrix



So, Matrix A =?

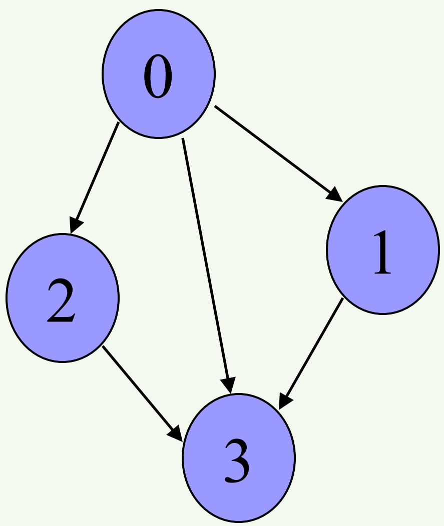
Example of Adjacency Matrix



A[i][j]	0	1	2	3
0	1	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	1

So, Matrix A = $\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & \end{pmatrix}$

Example of Adjacency Matrix



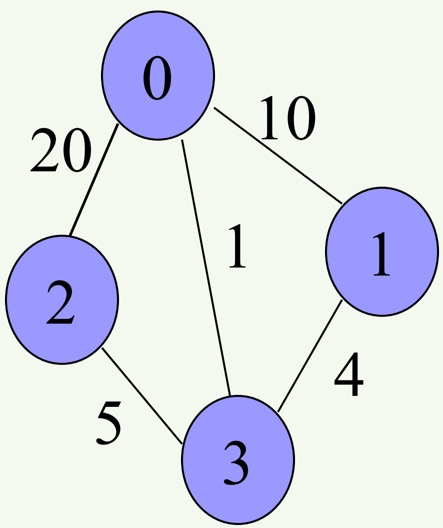
A[i][j]	0	1	2	3
0	0	1	1	1
1	0	0	0	1
2	0	0	0	1
3	0	0	0	0

So, Matrix $A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \end{pmatrix}$

Undirected vs. Directed

- Undirected graph
  - adjacency matrix is symmetric
  - A[i][j]=A[j][i]
- Directed graph
  - adjacency matrix may not be symmetric
  - A[i][j] $\neq$ A[j][i]

Weighted Graph



A[i][j]	0	1	2	3
0	0	20	10	1
1	20	0	0	5
2	10	0	0	4
3	1	5	4	0

So, Matrix A =

0 20 10
1

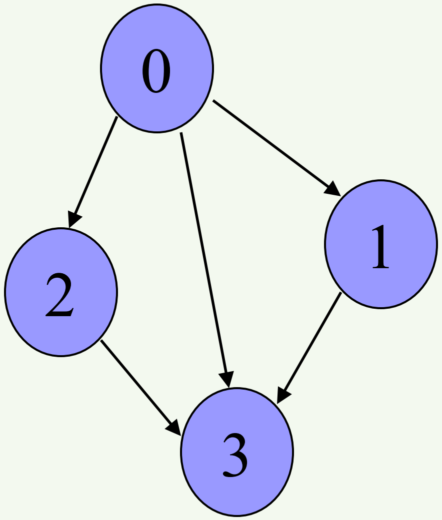
20 0 0
5

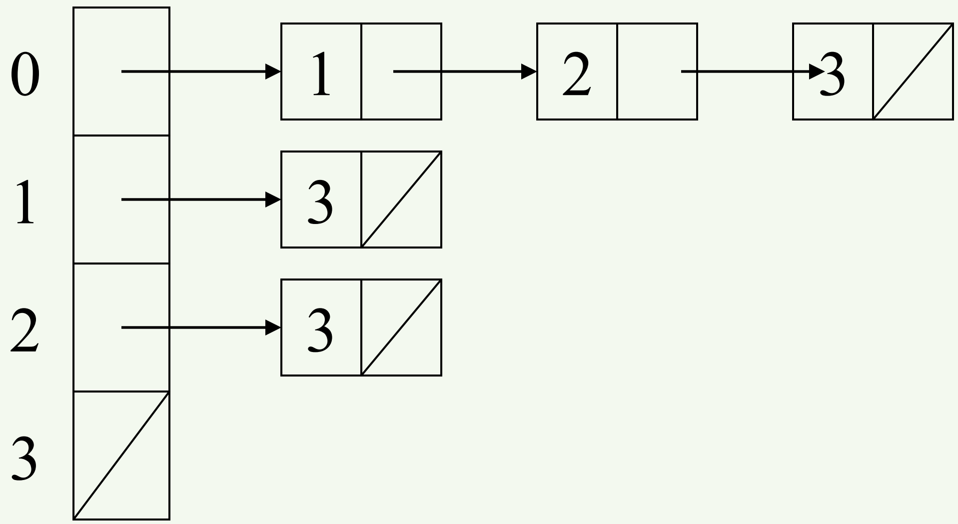
10 0 0
4

1 5 4

Adjacency List

- An array of list
- the *i*th element of the array is a list of vertices that connect to vertex i

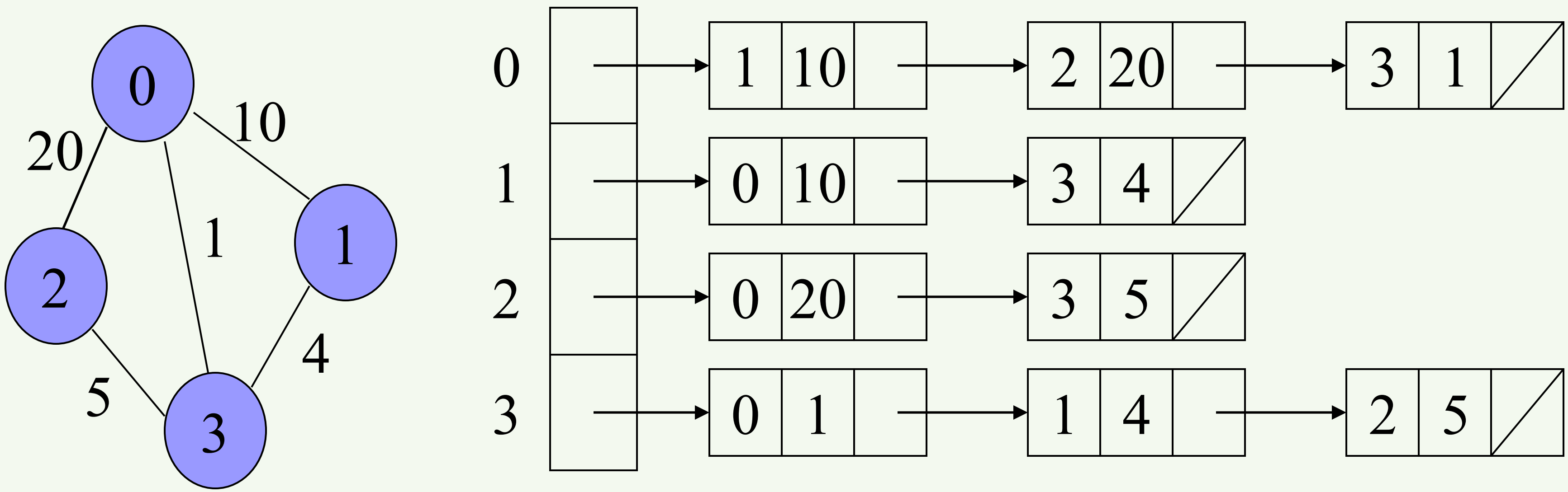




vertex 0 connect to vertex 1, 2 and 3
vertex 1 connects to 3
vertex 2 connects to 3

Weighted Graph

- Weighted graph: extend each node with an addition field: weight

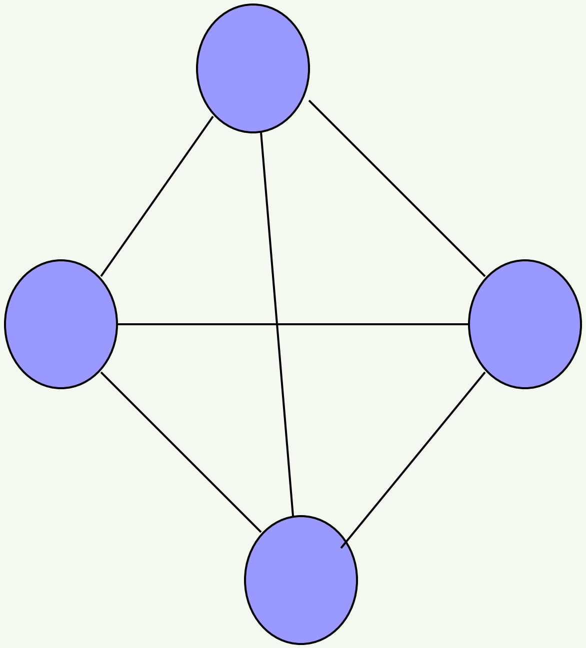


Comparison Of Representations

Cost	Adjacency Matrix	Adjacency List
Given two vertices u and v: **find out whether u and v are adjacent**	O(1)	degree of node O(N)
Given a vertex u: **enumerate all neighbors of u**	O(N)	degree of node O(N)
For all vertices: **enumerate all neighbors of each vertex**	$O(N^2)$	Summations of all node degree O(E)

Complete Graph

- There is an edge between any two vertices

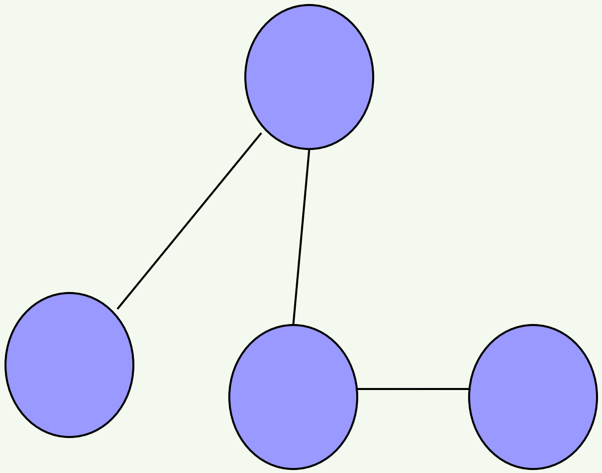


Total number of edges in graph:

$E = N(N-1)/2 = O(N^2)$

Sparse Graph

- There is a very small number of edges in the graph



For example:

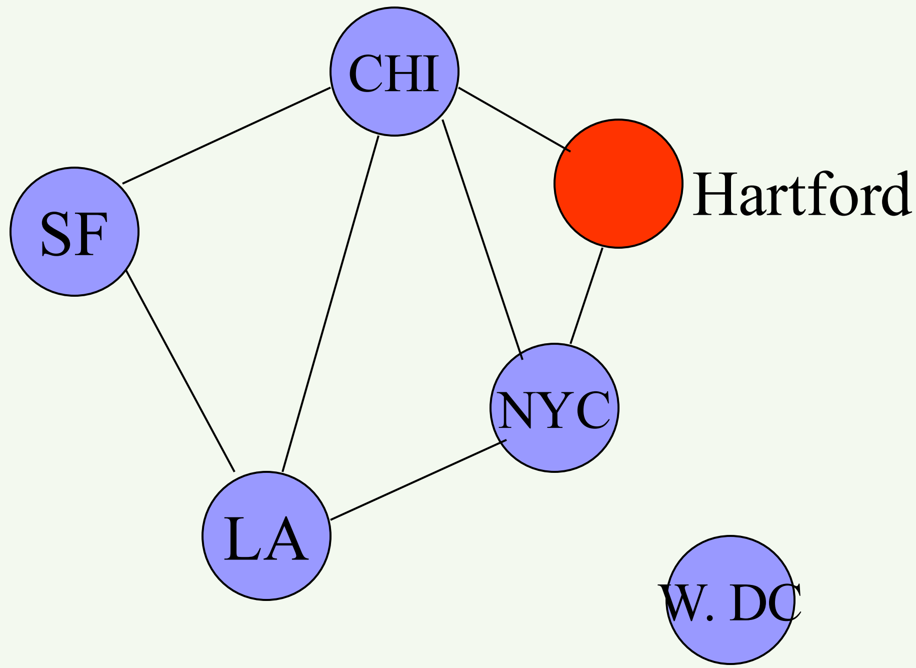
$$E = N-1 = O(N)$$

Space Requirements

- Memory space:
  - adjacency matrix $O(N^2)$
  - adjacency list $O(E)$
- Sparse graph
  - adjacency list is better
- Dense graph
  - same running time

Graph Traversal

- List out all cities that United Airline can reach from Hartford Airport

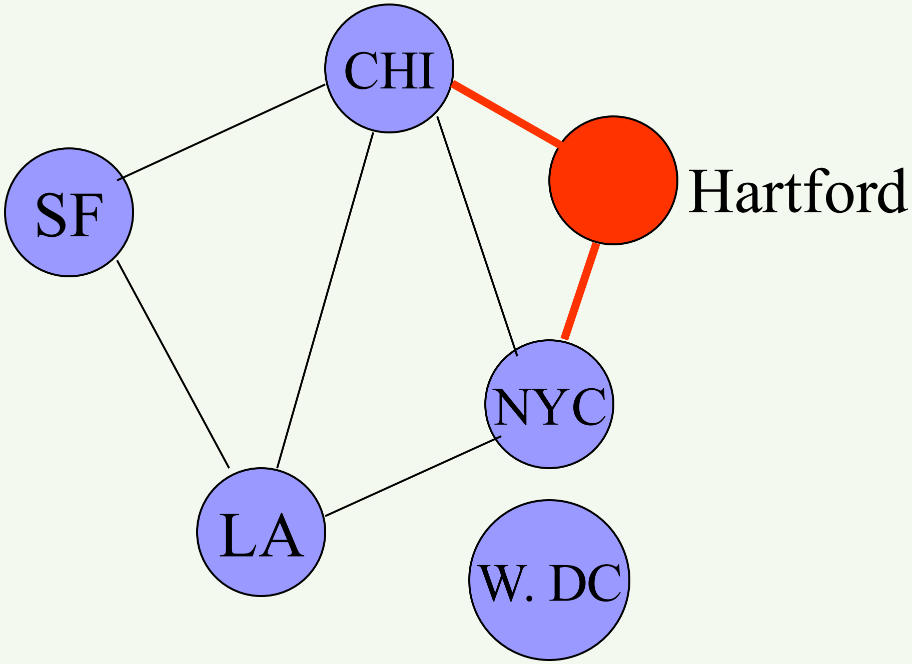


Graph Traversal

- From vertex u, list out all vertices that can be reached in graph G
- Set of nodes to expand
- Each node has a flag to indicate visited or not

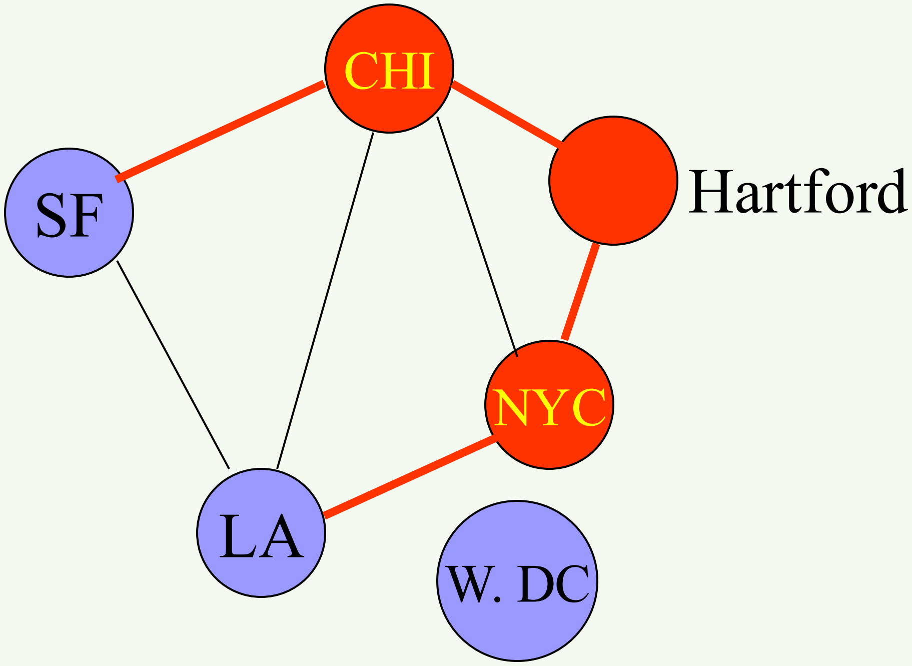
Traversal Algorithm

- Step 1: { Hartford }
  - find unvisited neighbors of Hartford
  - { Hartford, NYC, CHI }



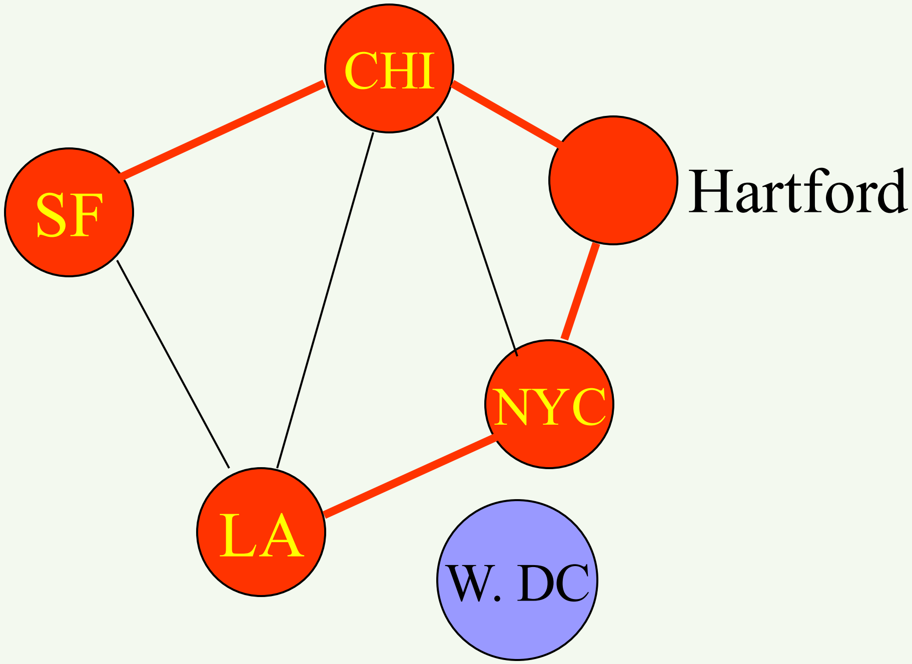
Traversal Algorithm

- Step 2: { Hartford, NYC, CHI }
  - find unvisited neighbors of NYC, CHI
  - { Hartford, NYC, CHI, LA, SF }



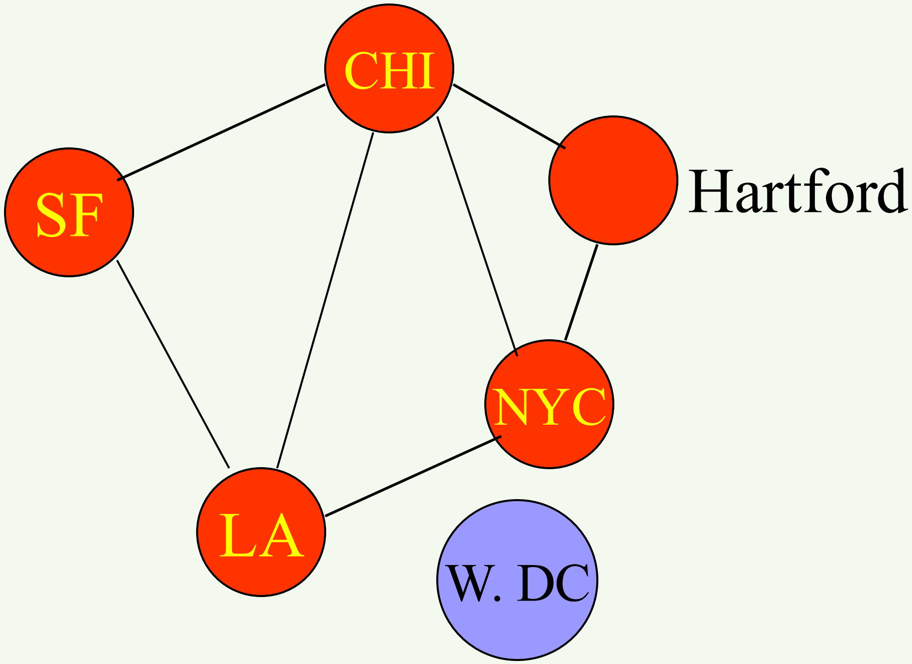
Traversal Algorithm

- Step 3: {Hartford, NYC, CHI, LA, SF }
  - find unvisited neighbors of LA, SF
  - no other new neighbors



Traversal Algorithm

- Finally, we get all cities that United Airline can reach from Hartford Airport
  - {Hartford, NYC, CHI, LA, SF }



Algorithm of Graph Traversal

1. Mark all nodes as unvisited
2. Pick a starting vertex u, add u to probing list
3. While (probing list is not empty)

   {

   Remove a node v from probing list

   Mark node v as visited

   For each neighbor w of v, if w is unvisited,
   add w to the probing list

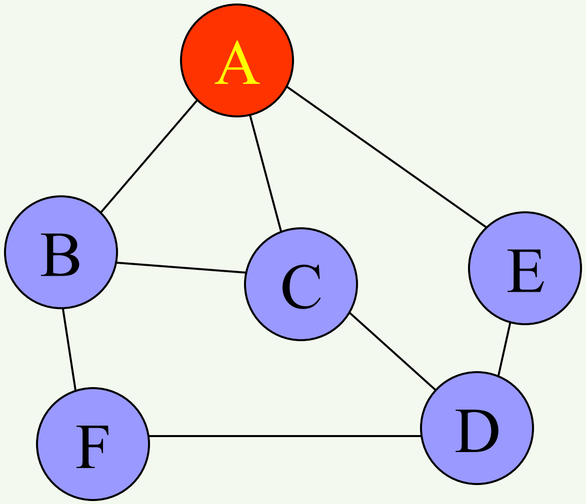
   }

Graph Traversal Algorithms

- Two algorithms
  - Depth First Traversal
  - Breadth First Traversal

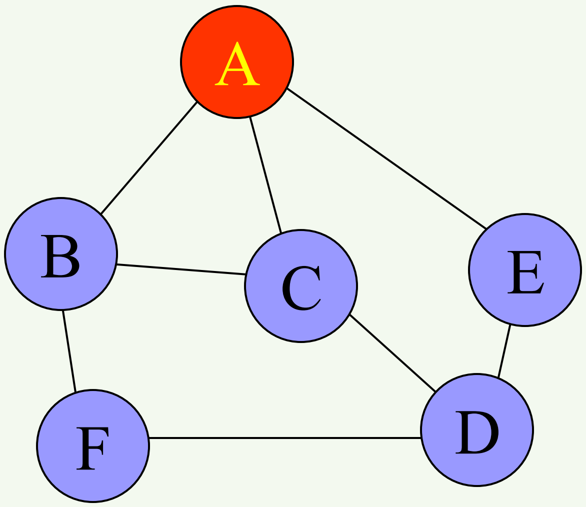
Depth First Traversal

- Probing List is implemented as stack (LIFO)
- Example
  - A's neighbor: B, C, E
  - B's neighbor: A, C, F
  - C's neighbor: A, B, D
  - D's neighbor: E, C, F
  - E's neighbor: A, D
  - F's neighbor: B, D
  - start from vertex A

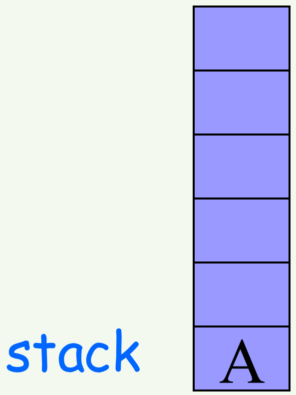


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

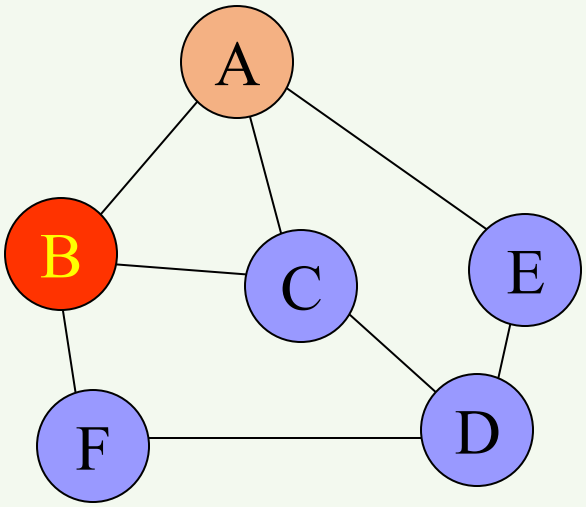


- Initial State
  - Visited Vertices { }
  - Probing Vertices { A }
  - Unvisited Vertices { A, B, C, D, E, F }

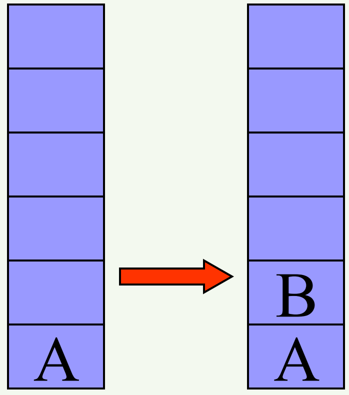


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

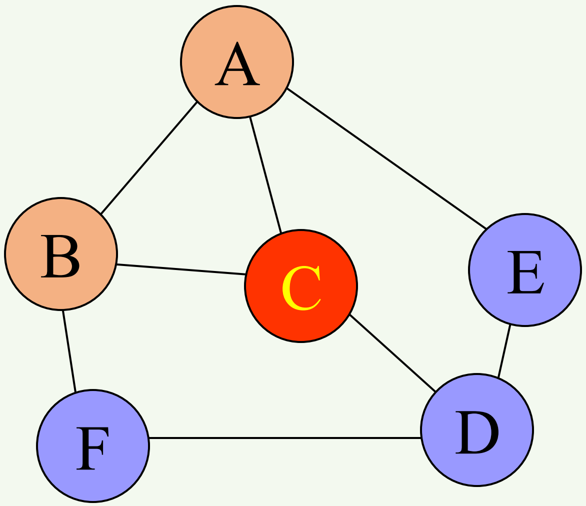


- Peek a vertex from stack, it is A, mark it as visited
- Find A's first unvisited neighbor, push it into stack
  - Visited Vertices { A }
  - Probing vertices { A, B }
  - Unvisited Vertices { B, C, D, E, F }

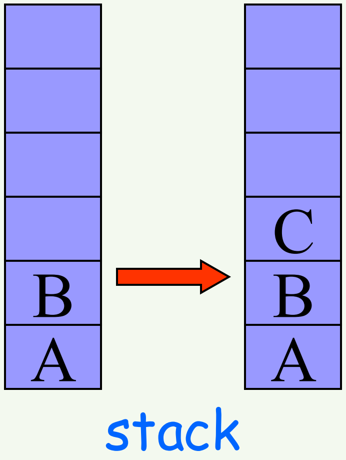


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

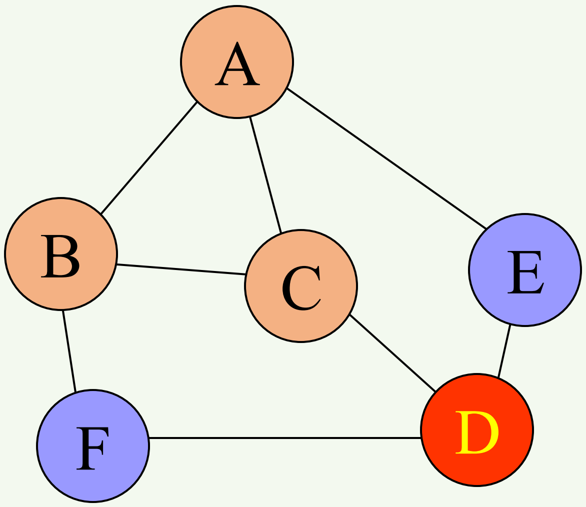


- Peek a vertex from stack, it is B, mark it as visited
- Find B's first unvisited neighbor, push it in stack
  - Visited Vertices { A, B }
  - Probing Vertices { A, B, C }
  - Unvisited Vertices { C, D, E, F }

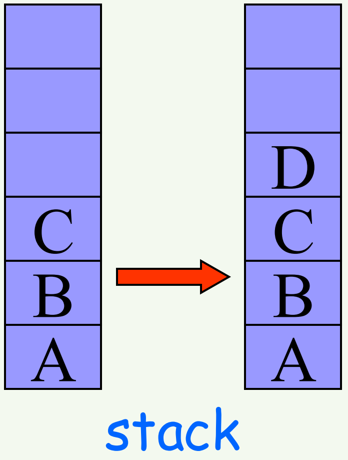


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

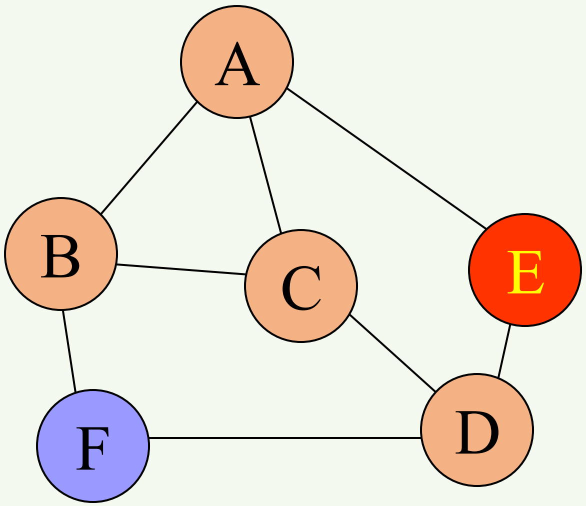


- Peek a vertex from stack, it is C, mark it as visited
- Find C's first unvisited neighbor, push it in stack
  - Visited Vertices { A, B, C }
  - Probing Vertices { A, B, C, D }
  - Unvisited Vertices { D, E, F }

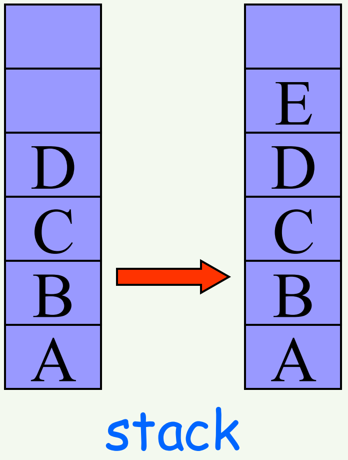


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

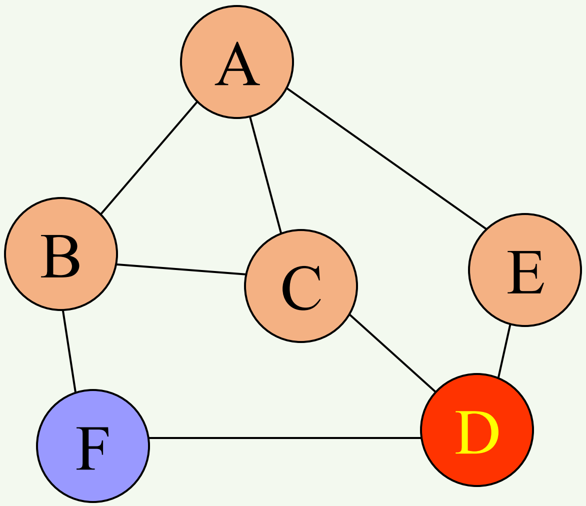


- Peek a vertex from stack, it is D, mark it as visited
- Find D's first unvisited neighbor, push it in stack
  - Visited Vertices { A, B, C, D }
  - Probing Vertices { A, B, C, D, E }
  - Unvisited Vertices { E, F }

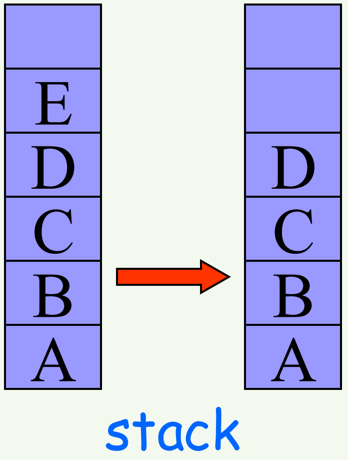


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

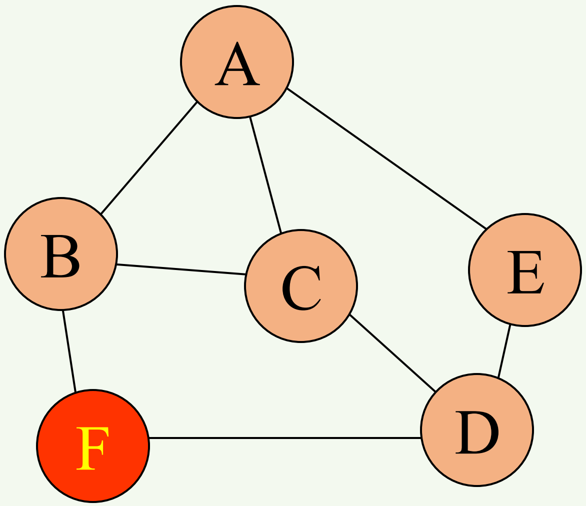


- Peek a vertex from stack, it is E, mark it as visited
- Find E's first unvisited neighbor, no vertex found, Pop E
  - Visited Vertices { A, B, C, D, E }
  - Probing Vertices { A, B, C, D }
  - Unvisited Vertices { F }

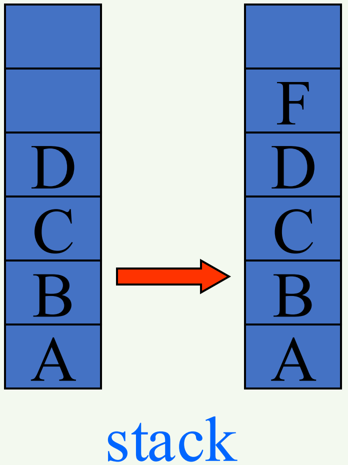


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

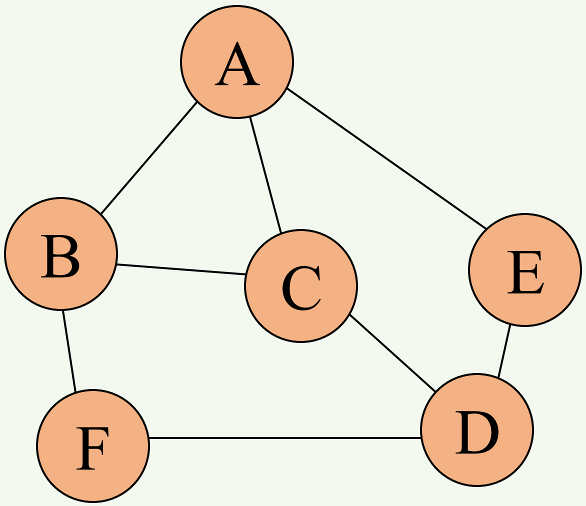


- Peek a vertex from stack, it is D, mark it as visited
- Find D's first unvisited neighbor, push it in stack
  - Visited Vertices { A, B, C, D, E }
  - Probing Vertices { A, B, C, D, F}
  - Unvisited Vertices { F }

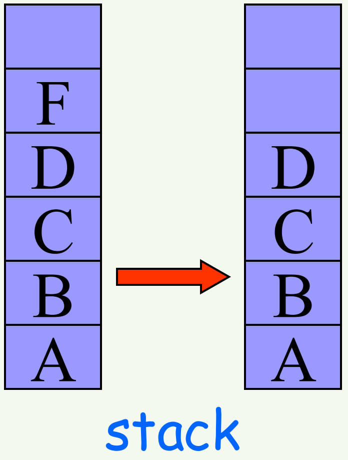


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

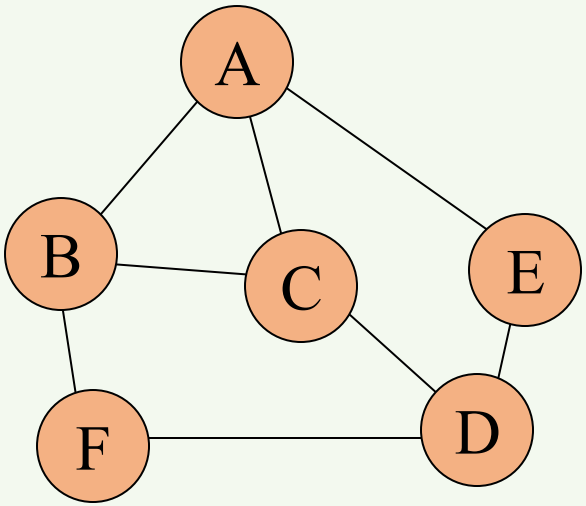


- Peek a vertex from stack, it is F, mark it as visited
- Find F's first unvisited neighbor, no vertex found, Pop F
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { A, B, C, D}
  - Unvisited Vertices { }

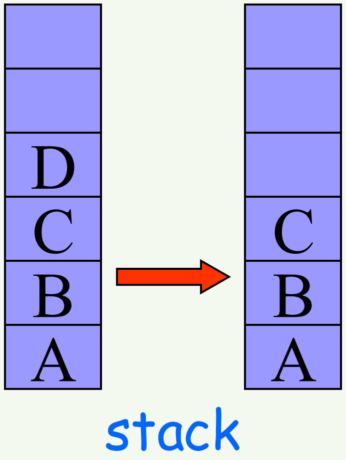


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

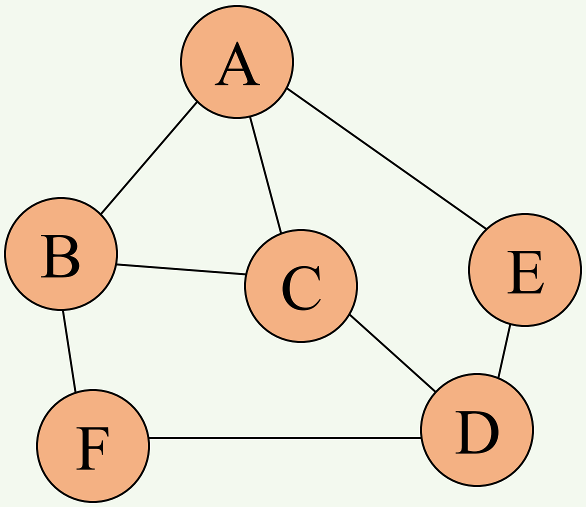


- Peek a vertex from stack, it is D, mark it as visited
- Find D's first unvisited neighbor, no vertex found, Pop D
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { A, B, C }
  - Unvisited Vertices { }

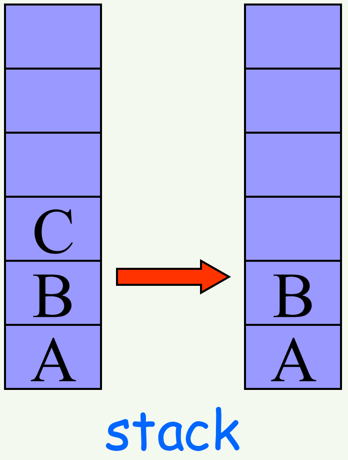


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

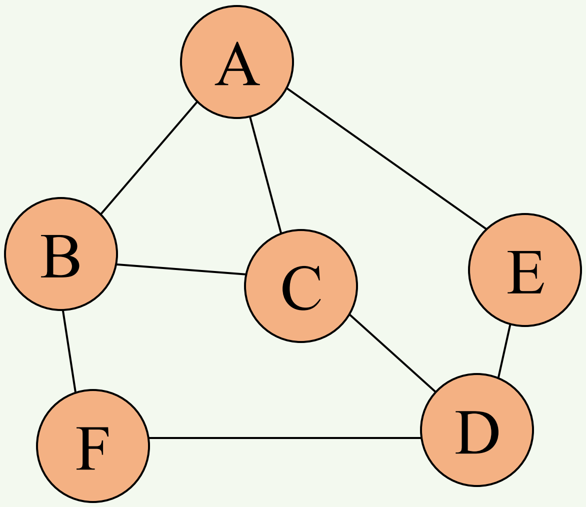


- Peek a vertex from stack, it is C, mark it as visited
- Find C's first unvisited neighbor, no vertex found, Pop C
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { A, B }
  - Unvisited Vertices { }

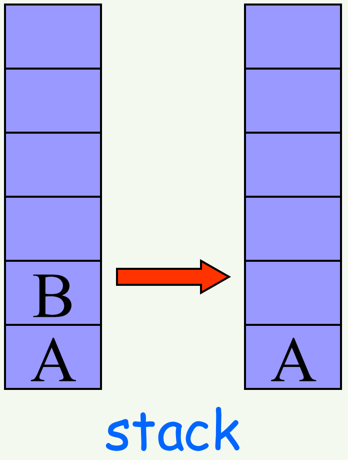


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

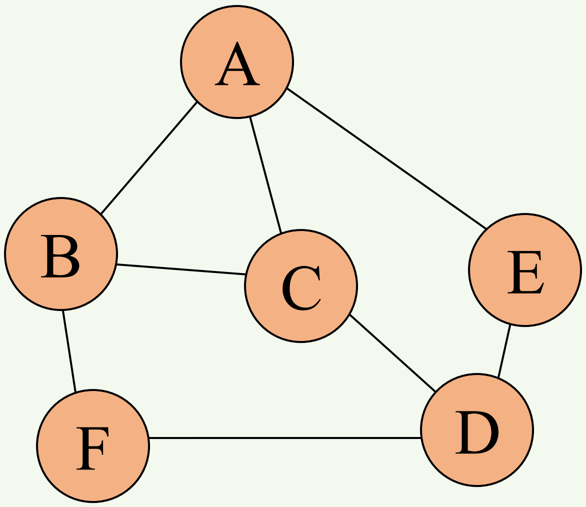


- Peek a vertex from stack, it is B, mark it as visited
- Find B's first unvisited neighbor, no vertex found, Pop B
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { A }
  - Unvisited Vertices { }

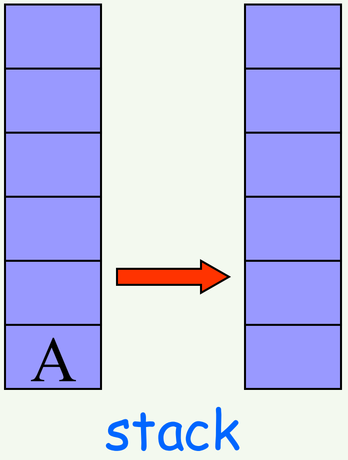


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

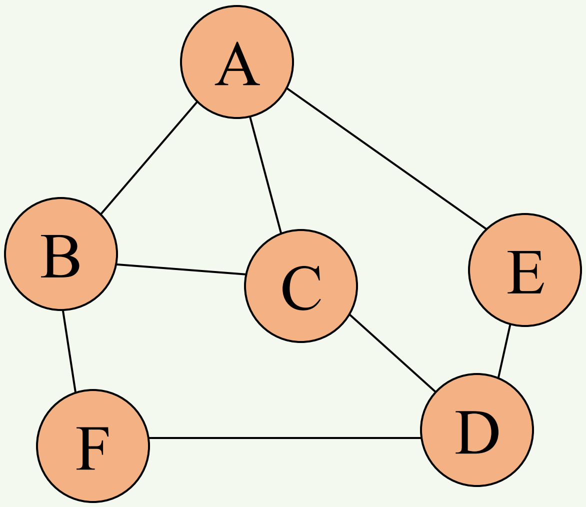


- Peek a vertex from stack, it is A, mark it as visited
- Find A's first unvisited neighbor, no vertex found, Pop A
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { }
  - Unvisited Vertices { }

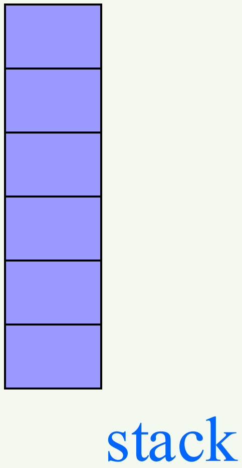


Depth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

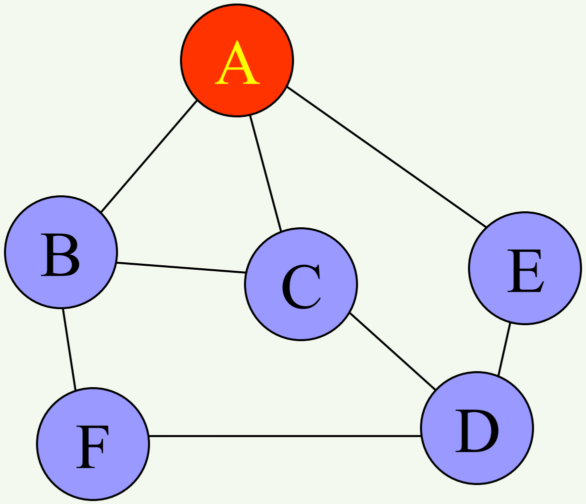


- Now probing list is empty
- End of Depth First Traversal
  - Visited Vertices { A, B, C, D, E, F }
  - Probing Vertices { }
  - Unvisited Vertices { }



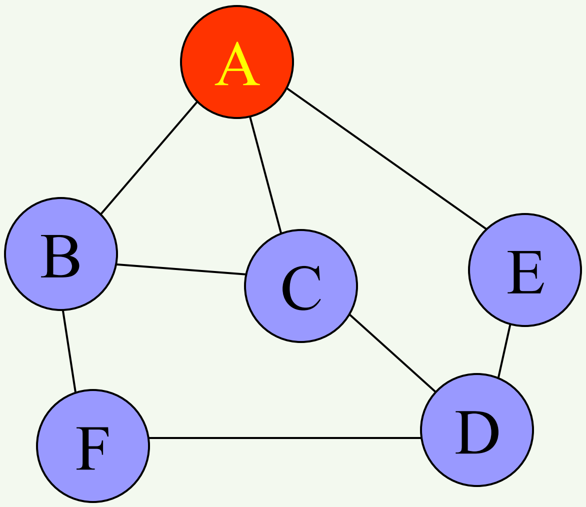
Breadth First Traversal

- Probing List is implemented as queue (FIFO)
- Example
  - A's neighbor: B C E
  - B's neighbor: A C F
  - C's neighbor: A B D
  - D's neighbor: E C F
  - E's neighbor: A D
  - F's neighbor: B D
  - start from vertex A

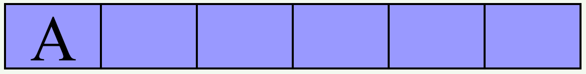


Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D



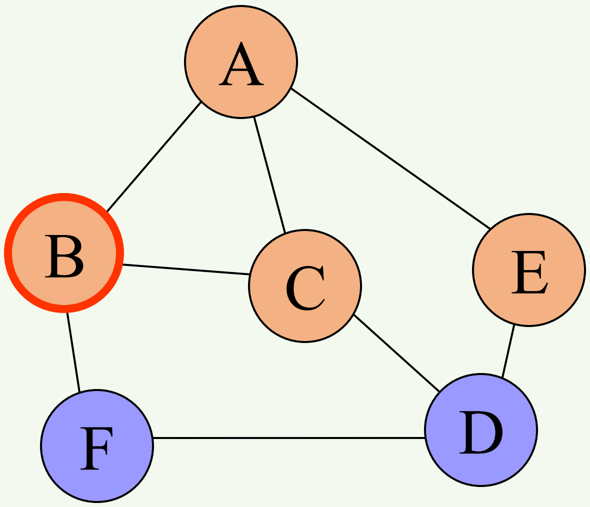
- Initial State
  - Visited Vertices { }
  - Probing Vertices { A }
  - Unvisited Vertices { A, B, C, D, E, F }



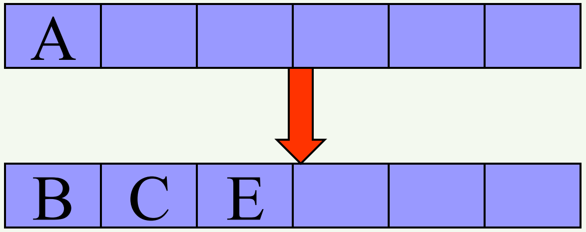
queue

Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

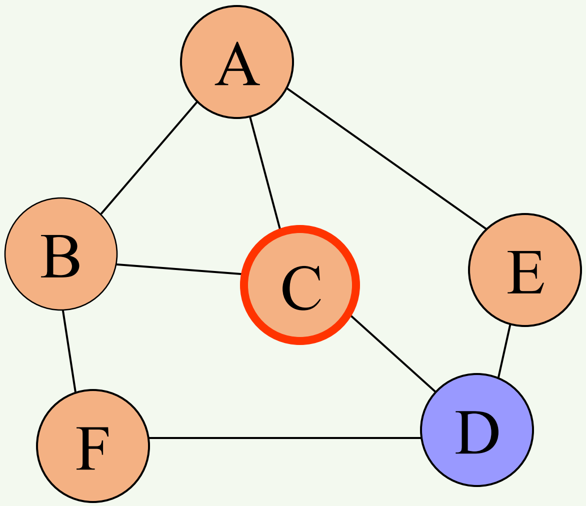


- Delete first vertex from queue, it is A, mark it as visited
- Find A's all unvisited neighbors, mark them as visited, put them into queue
  - Visited Vertices { A, B, C, E }
  - Probing Vertices { B, C, E }
  - Unvisited Vertices { D, F }

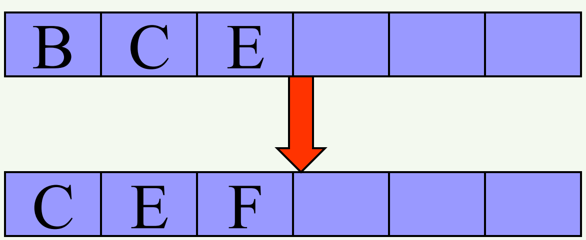


Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D



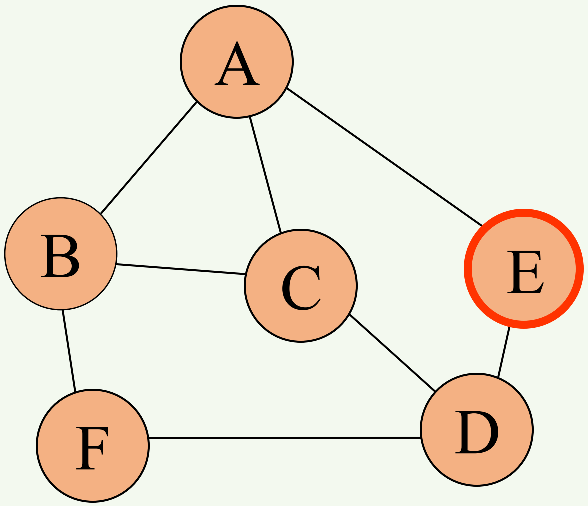
- Delete first vertex from queue, it is B, mark it as visited
- Find B's all unvisited neighbors, mark them as visited, put them into queue
    - Visited Vertices { A, B, C, E, F }
    - Probing Vertices { C, E, F }
    - Unvisited Vertices { D }



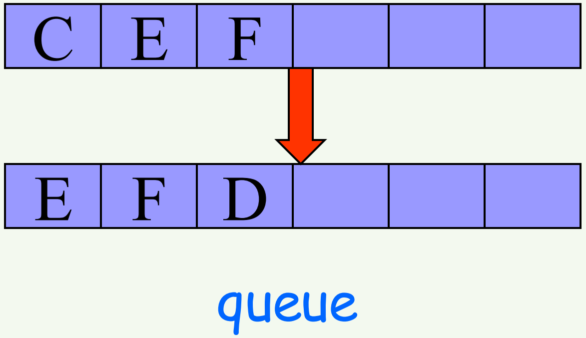
queue

Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

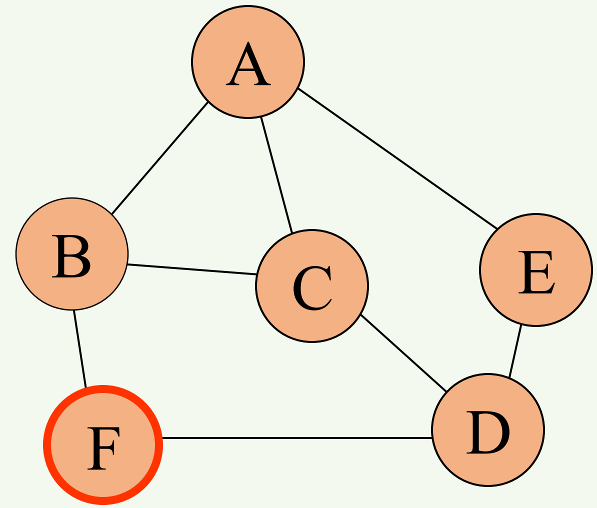


- Delete first vertex from queue, it is C, mark it as visited
- Find C's all unvisited neighbors, mark them as visited, put them into queue
  - Visited Vertices { A, B, C, E, F, D }
  - Probing Vertices { E, F, D }
  - Unvisited Vertices { }

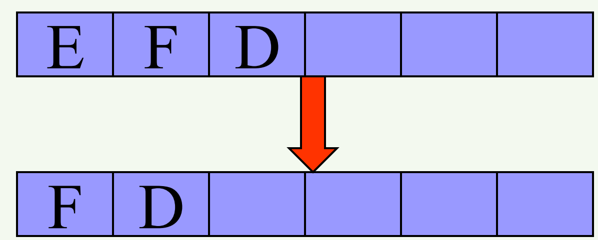


Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D



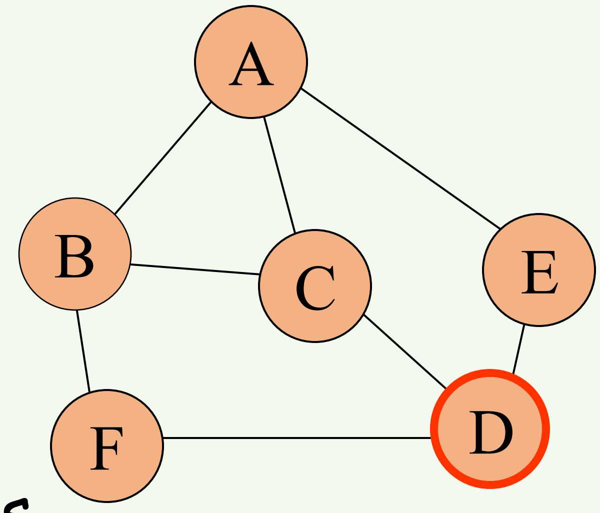
- Delete first vertex from queue, it is E, mark it as visited
- Find E's all unvisited neighbors, no vertex found
  - Visited Vertices { A, B, C, E, F, D }
  - Probing Vertices { F, D }
  - Unvisited Vertices { }



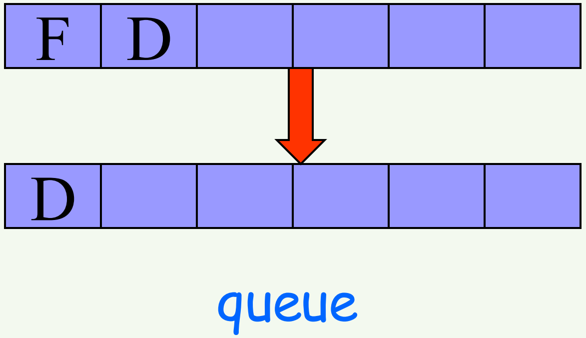
queue

Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D

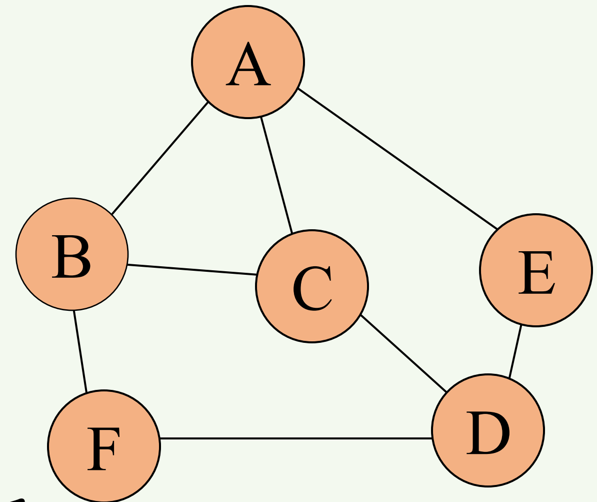


- Delete first vertex from queue, it is F, mark it as visited
- Find F's all unvisited neighbors, no vertex found
  - Visited Vertices { A, B, C, E, F, D }
  - Probing Vertices { D }
  - Unvisited Vertices { }

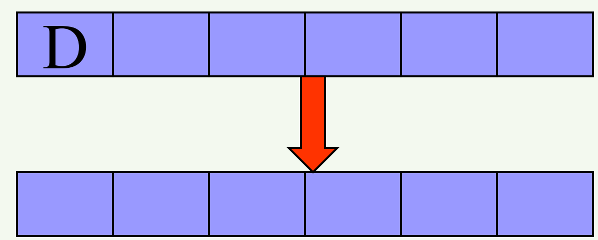


Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D



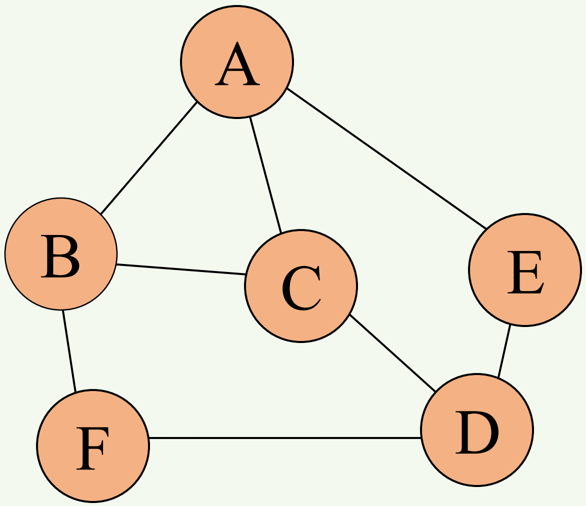
- Delete first vertex from queue, it is D, mark it as visited
- Find D's all unvisited neighbors, no vertex found
  - Visited Vertices { A, B, C, E, F, D }
  - Probing Vertices { }
  - Unvisited Vertices { }



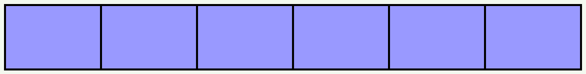
queue

Breadth First Traversal (Cont)

- A's neighbor: B C E
- B's neighbor: A C F
- C's neighbor: A B D
- D's neighbor: E C F
- E's neighbor: A D
- F's neighbor: B D



- Now the queue is empty
- End of Breadth First Traversal
  - Visited Vertices { A, B, C, E, F, D }
  - Probing Vertices { }
  - Unvisited Vertices { }



queue

Difference Between DFT & BFT

- Depth First Traversal (DFT)
  - order of visited: A, B, C, D, E, F
- Breadth First Traversal (BFT)
  - order of visited: A, B, C, E, F, D