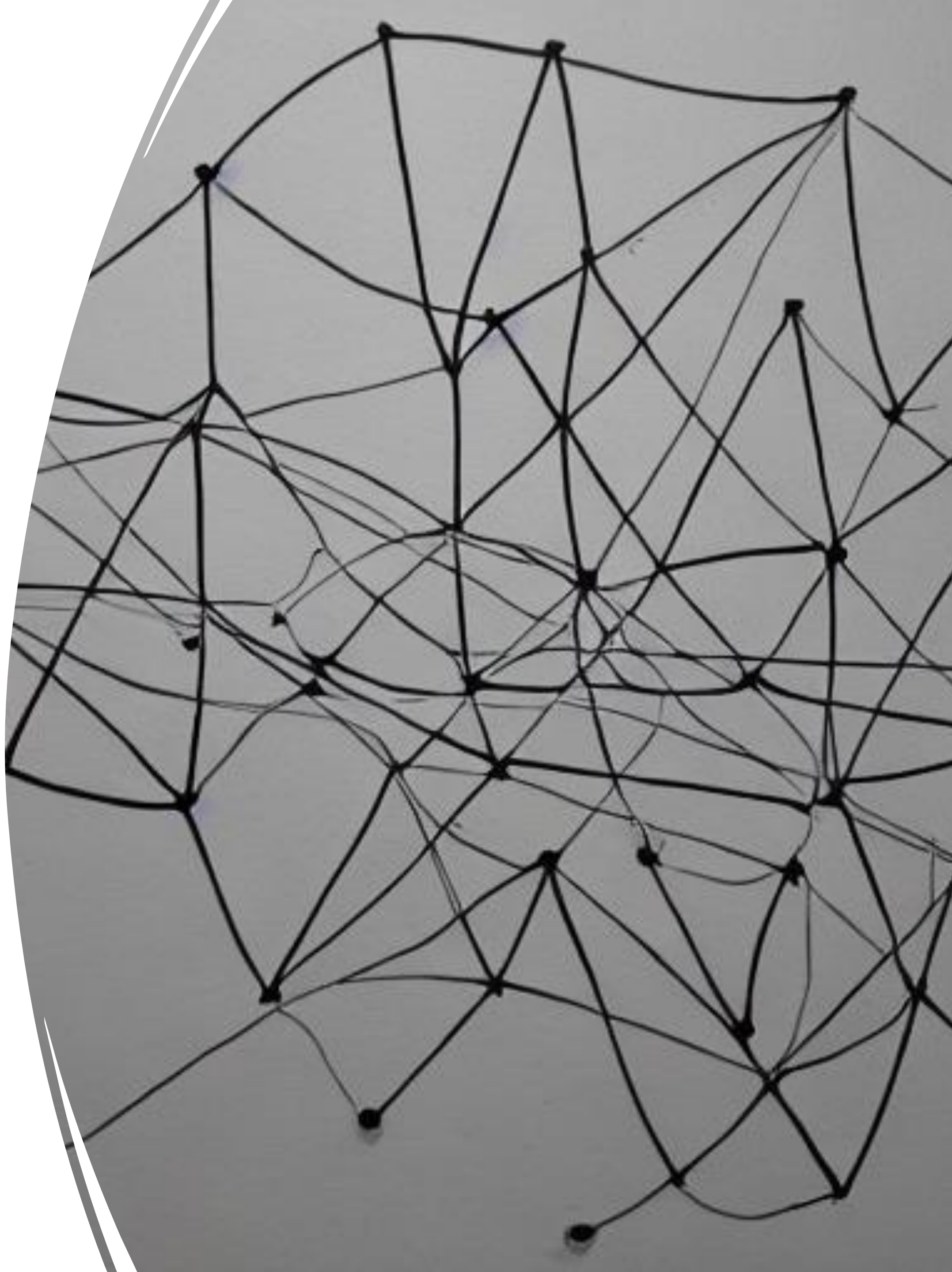
Please use the following QR code to check in and record your attendance.

00:01:59

CS 1037
Fundamentals of Computer Science II

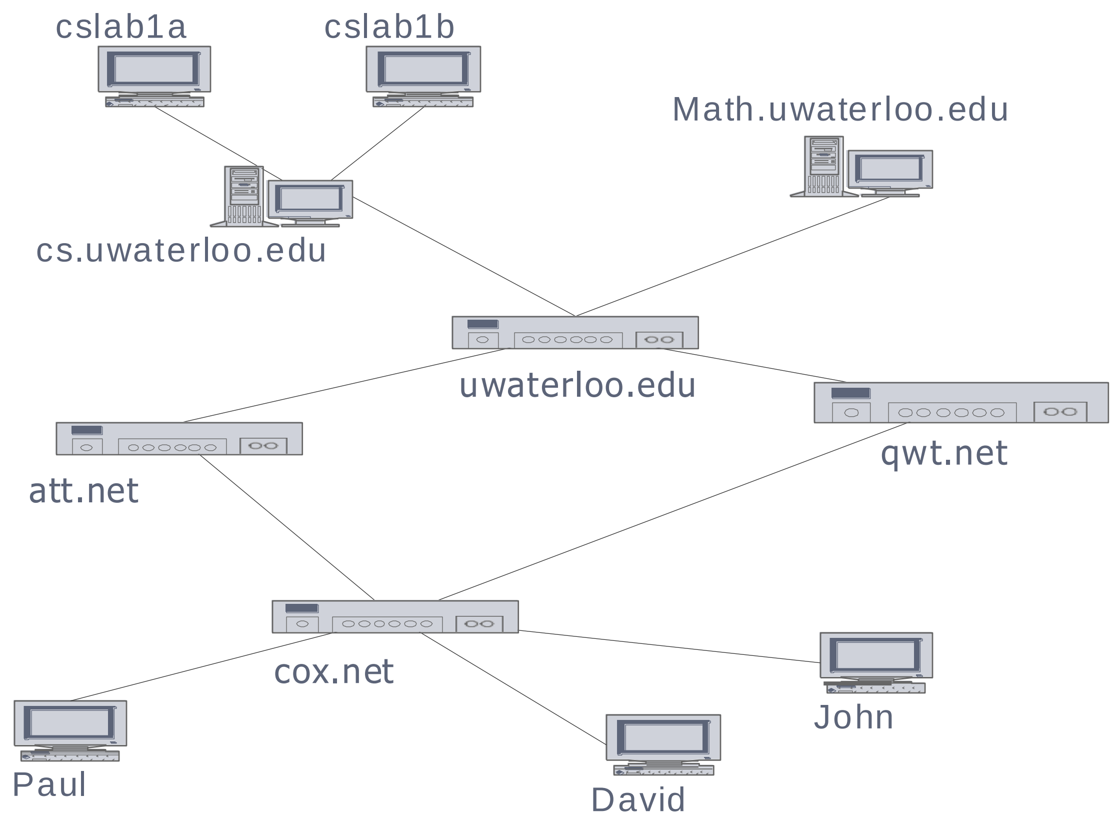
Graphs

Ahmed Ibrahim



Motivation

- A graph is a natural representation for a special type of data:
- Computer networks
    - Local area network
    - Internet
    - Web



II. FROM CIRCUIT TO GRAPH

A graph can be obtained from a circuit. We identify the graph G= (V, E) where V is the set of vertices and E is the set of edges. The edge between i^{th} and j^{th} vertices can be denoted by {i,j} ignoring the direction. Similarly the notation (i,j) can be used for oriented edges, where i is the start vertex and j is the end vertex. For example consider the circuit and its graph in the figure. There are five vertices and seven edges in the graph obtained from the given circuit. An edge is sometimes called branch and a vertex is called node in case of electrical circuits.

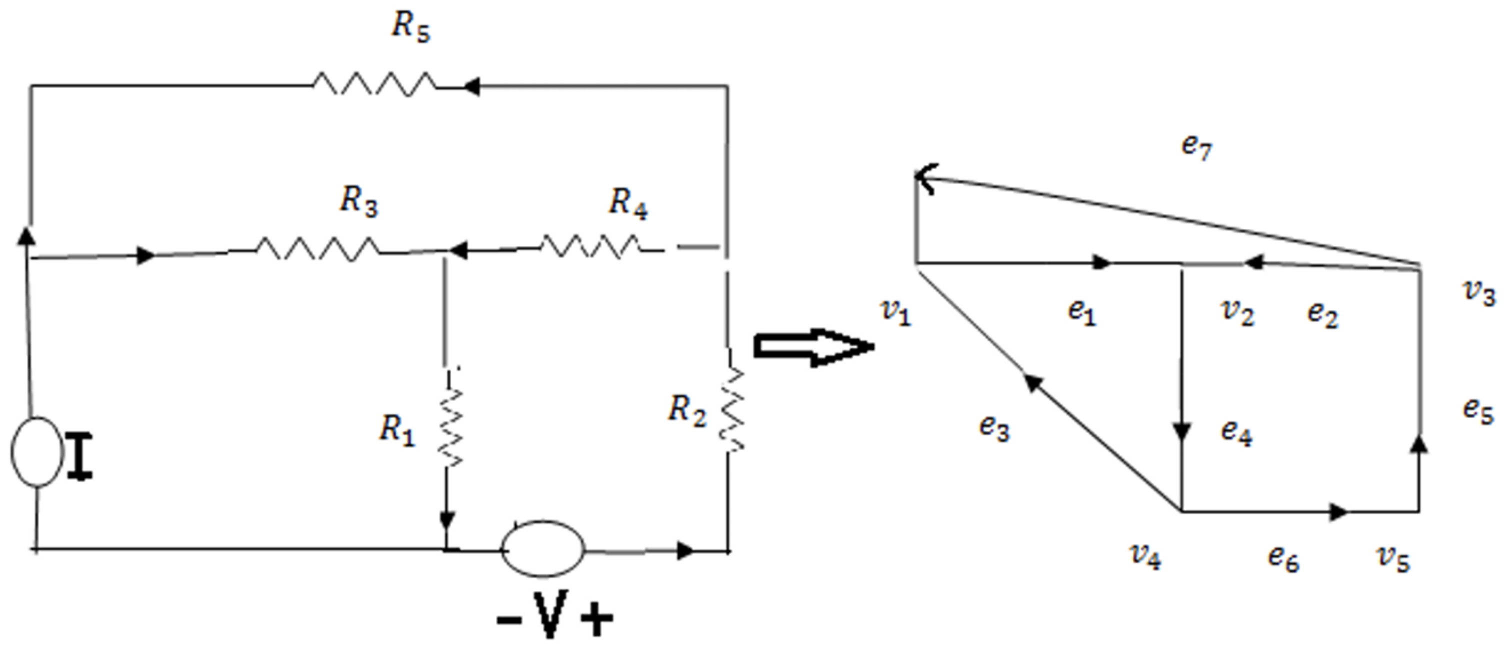
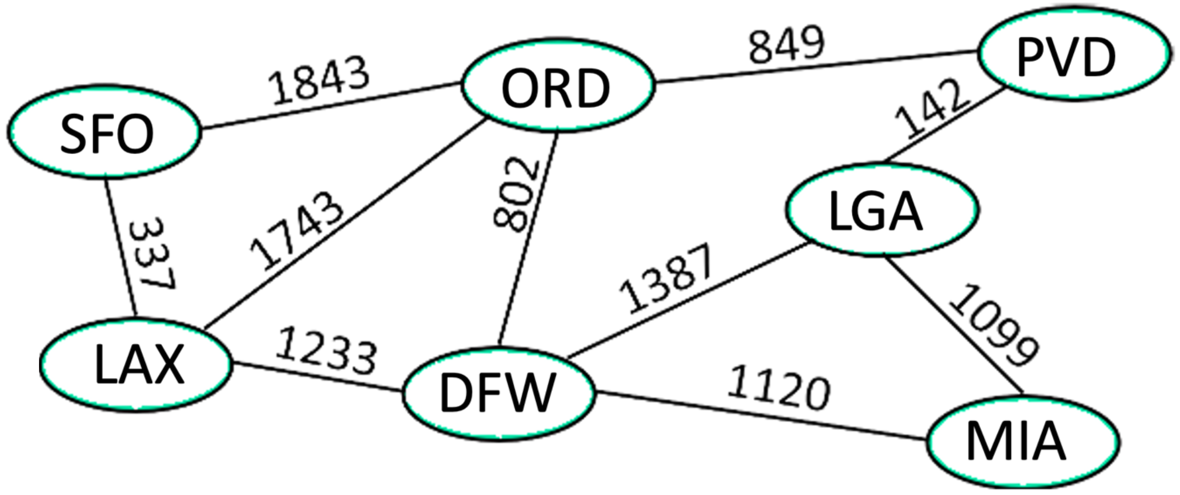


Fig: A circuit and its corresponding graph

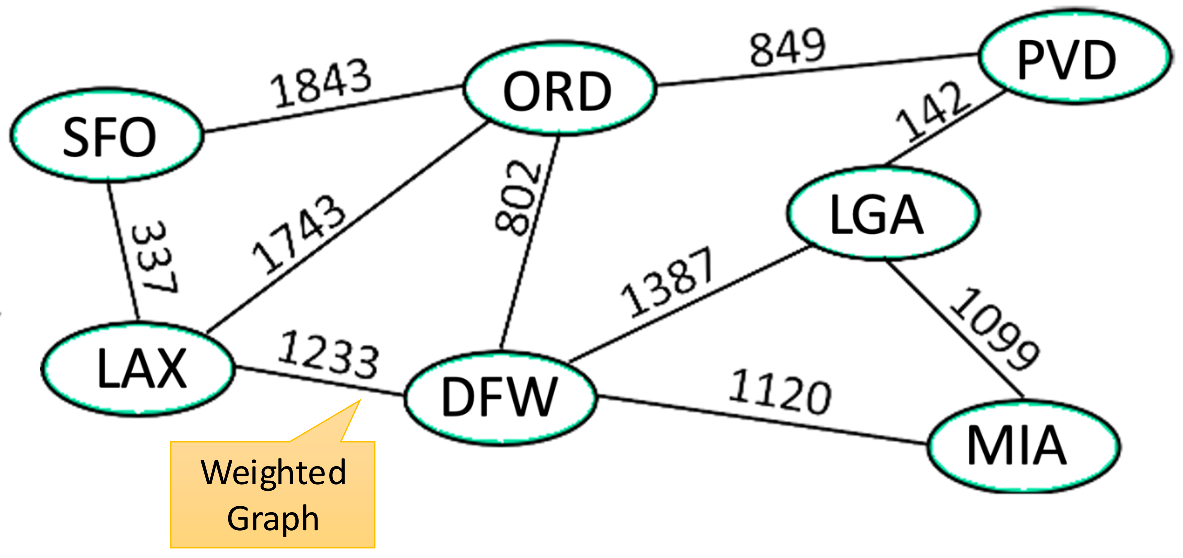
Motivation

- A graph is a natural representation for a special type of data:
- City map (Transportation Networks):
  - Each city is represented by a node
  - Can label each node with a three-letter airport code
  - Two cities with a direct flight between them are connected by an edge
  - You can label the edge with the mileage of the route, time to fly, etc.



Motivation

- You can answer many interesting questions using graphs
    - Can we reach one city from another city?
    - What is the route with a minimum number of connections between 2 cities?
    - What is the minimum mileage route between 2 cities?

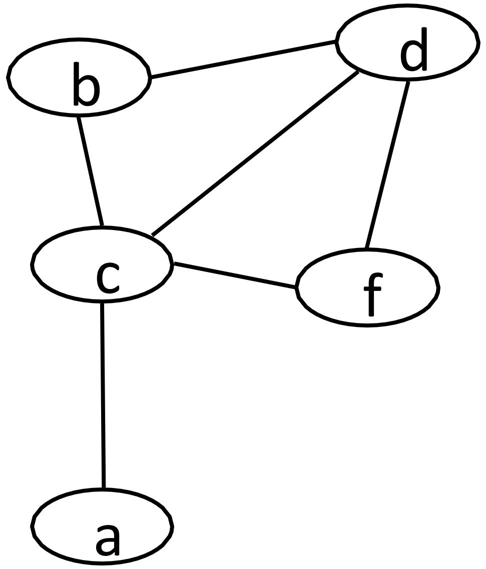


- Many interesting questions can be answered efficiently using graphs.

Graphs: Formal Definition

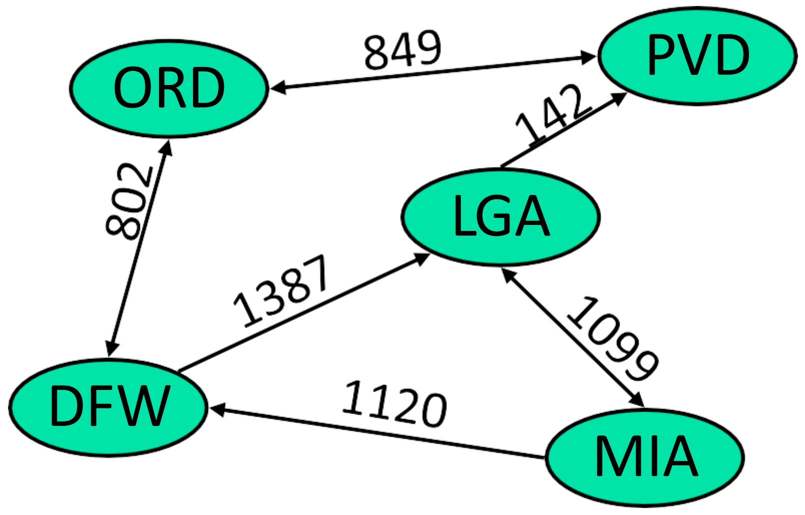
- A graph is a pair (***V, E***), where
  - V is a collection of **nodes** or **vertices**
  - E is a collection of pairs of vertices called **edges**
- In this example
  - ***V*** ={a,b,c,d,f}
  - ***E*** ={(a,c),(b,c),(c,f),(b,d),(d,f), (c,d)}

Pairs

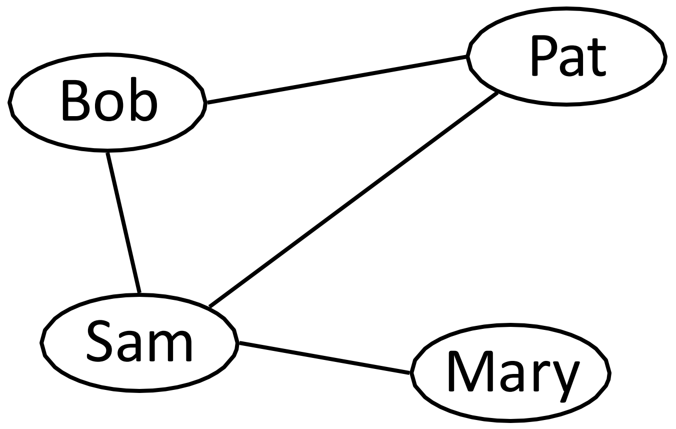


Graph Types

- Directed graph (Digraph)
  - All the edges are directed
  - e.g., flight route network (map)

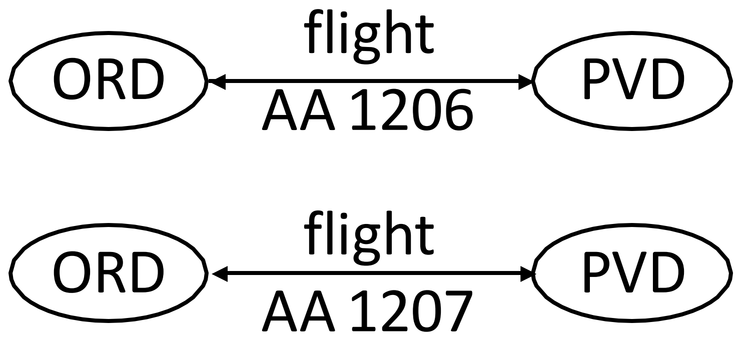


- Undirected graph
  - All edges are undirected
  - e.g.,“friends” network



Directed edge (cont.)

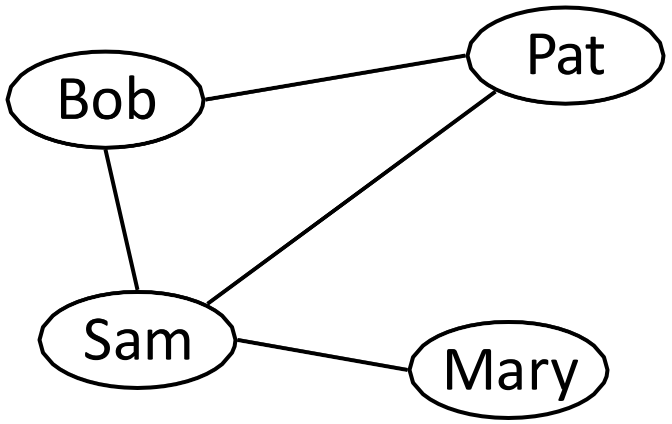
- Ordered pair of vertices (***u***,***v***)
  - First vertex ***u*** is the **origin**
  - Second vertex ***v*** is the **destination**
  - e.g., a flight
- (***v***,***u***) and (***u***,***v***) are two **different edges**



flight route network

Undirected Graph (cont.)

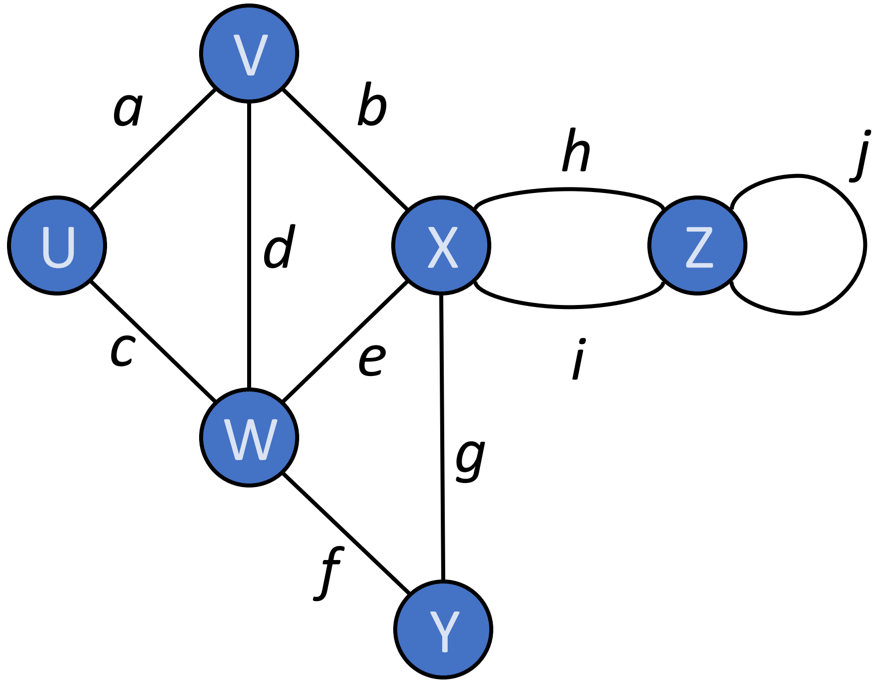
- Unordered pair of vertices (***u***,***v***)
  - e.g., a network of friends
- If ***Sam*** is a friend of ***Bob***, then ***Bob*** is also a friend of ***Sam***
- (***u***,***v***) and is (***v***,***u***) the **same edge**



A "friends" network

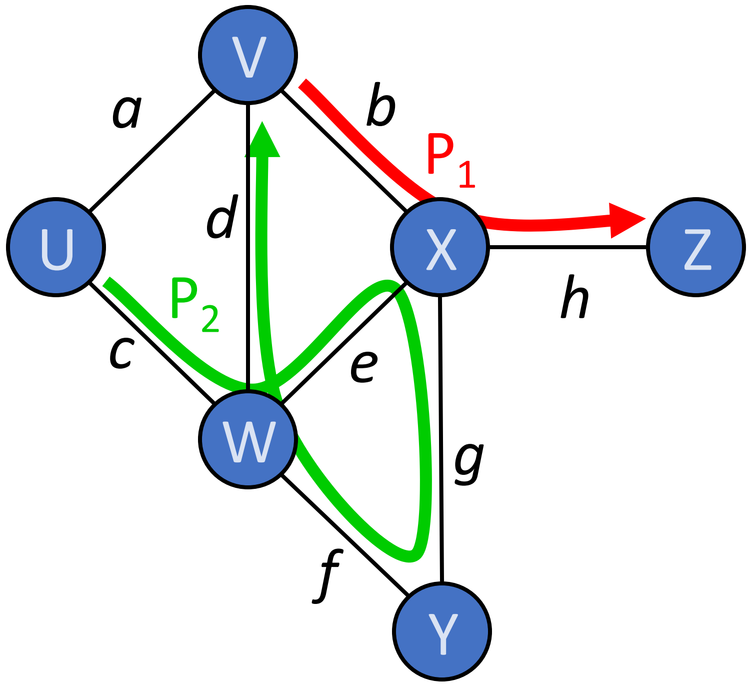
Graph Terminology

- **Endpoints** (or end vertices) of an edge
  - U and V are the *endpoints* of *a*
- **Edges** incident on a vertex
  - *a*, *d*, and *b* are *incident* on V
- **Adjacent** vertices
  - U and V are *adjacent*
- **Degree** of a vertex
  - X has *degree* 5
- **Parallel** (multiple) edges
  - *h* and *i* are *parallel edges*
- **Self-loop**
  - j is a *self-loop*



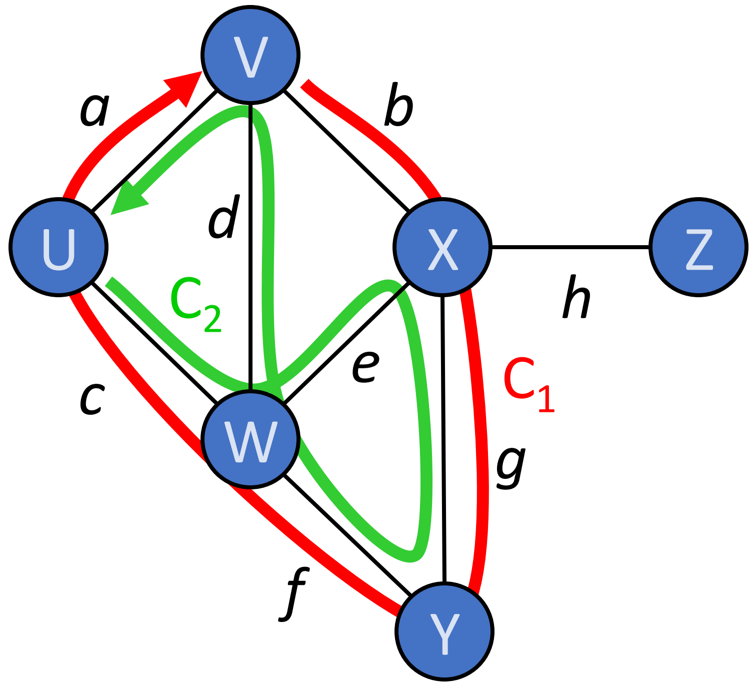
Graph Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
- Simple path
  - A path such that all its vertices and edges are **distinct**
- Examples
  - **P_1=(V,b,X,h,Z)** is a simple path
  - **P_2=(U,c,W,e,X,g,Y,f,W,d,V)** is a path that is not simple



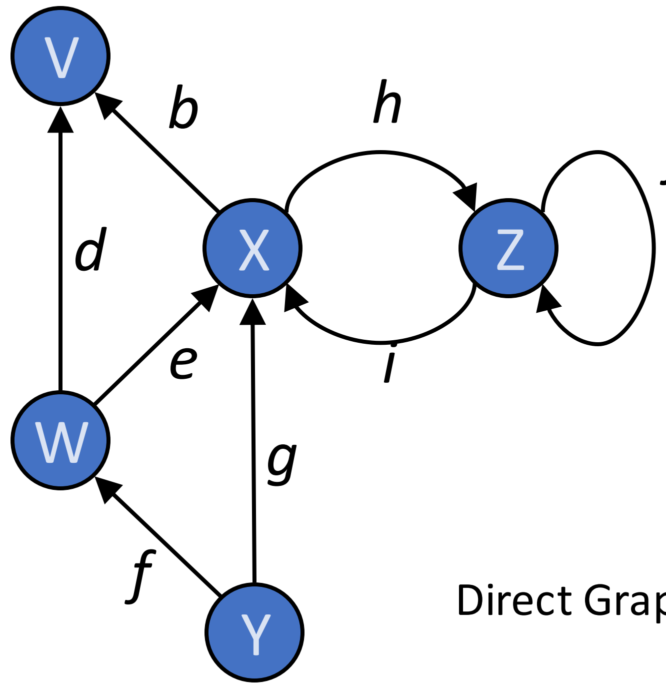
Graph Terminology (cont.)

- Cycle
  - circular sequence of alternating vertices and edges
- Simple cycle
  - cycle such that all its vertices and edges are **distinct**
- Examples
  - **C_1=(V,b,X,g,Y,f,W,c,U,a,V)** is a simple cycle
  - **C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,U)** is a cycle that is not simple



Graph Terminology (cont.)

- Outgoing edges of a vertex
  - *h* and *b* are the *outgoing edges* of X
- Incoming edges of a vertex
  - *e*, *g*, and *i* are *incoming edges* of X
- In-degree of a vertex
  - X has *in-degree* 3
- Out-degree of a vertex
  - X has *out-degree* 2



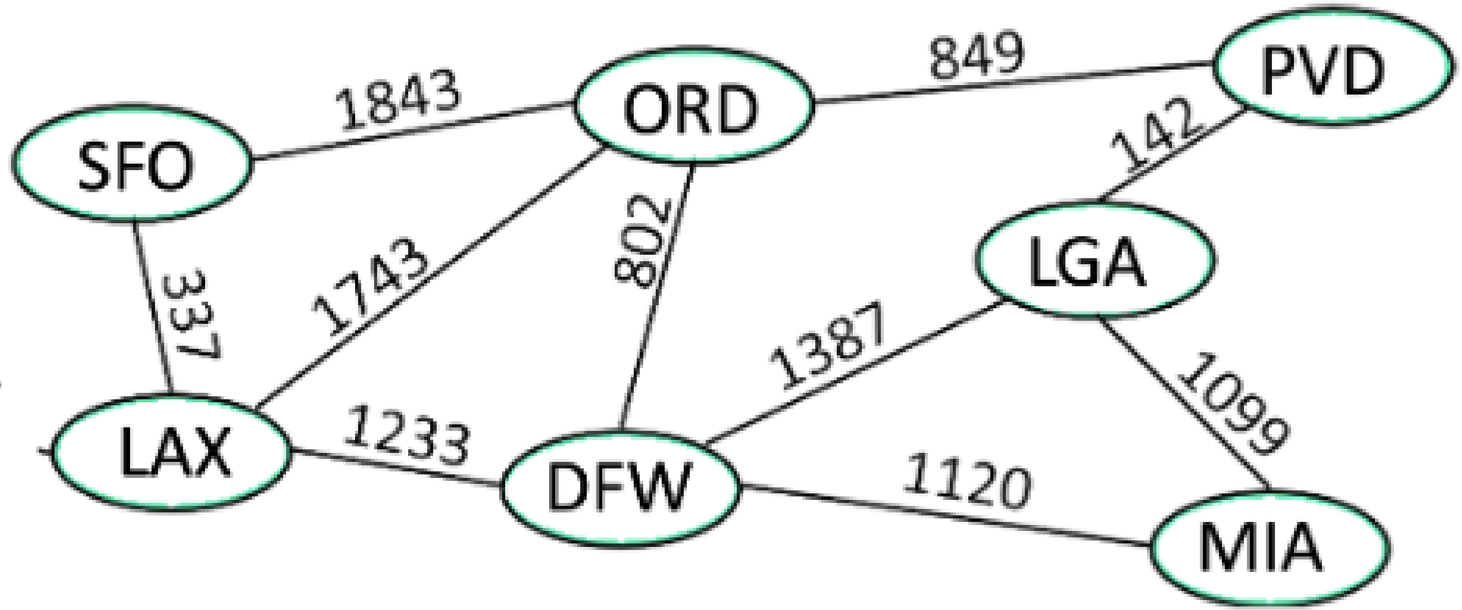
Direct Graph

Question!

- What is the degree of vertex x if it is adjacent to 4 vertices in a simple undirected graph?
  - a) 4
  - b) 3
  - c) 2
  - d) 1

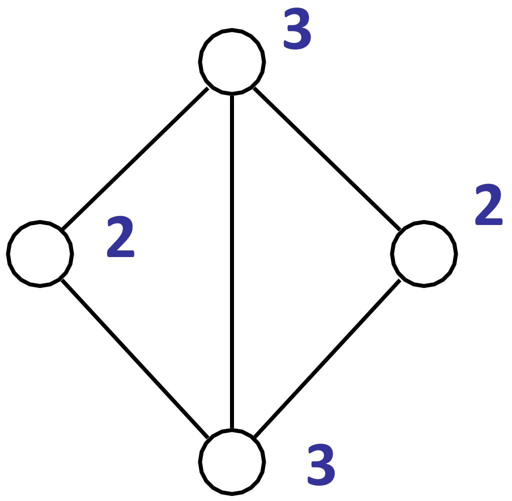
- The degree of a vertex is the number of adjacent vertices. Since x is adjacent to 4 vertices, its degree is 4.

Graph Properties



Undirected Graph Properties

- Let
  - ***n*** = number of vertices
  - ***m*** = number of edges
  - ***deg***(***v***) = degree of vertex ***v*** = number of adjacent vertices of ***v***
- Property #1 –> $\Sigma_v \deg(v) = 2m$
- Property #2 –> In an undirected graph with no self-loops and no parallel edges: $m \leq n(n-1)/2$



A graph with given number of vertices (4) and number of edges (5)

Directed Graph (Digraph) Properties

- Let
  - ***n*** = number of vertices
  - ***m*** = number of edges
  - ***deg*(*v*)** = degree of vertex ***v*** = number of adjacent vertices of ***v***

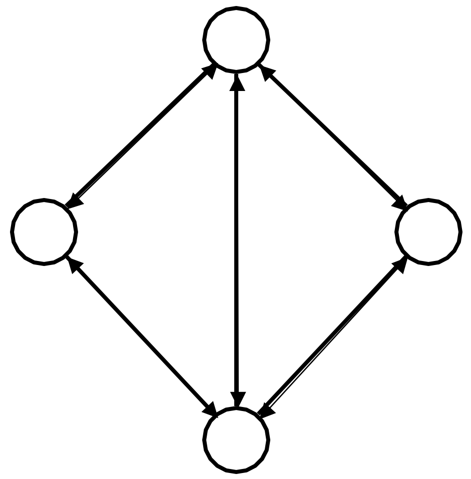
Property 1 – Total in-degree and out-degree

$\sum_v$ in-deg(***v***) = *m*

$\sum_v$ out-deg(***v***) = *m*

Property 2 – Total number of edges

In a directed graph with no self-loops and no parallel edges: $\boldsymbol{m} \leq \boldsymbol{n}\,(\boldsymbol{n} - 1)$



A digraph with given number of vertices (4) and maximum number of edges (12)

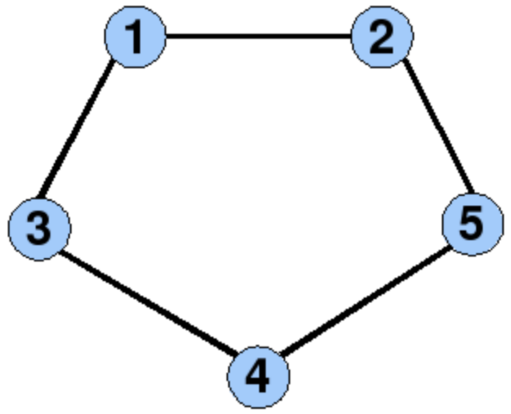
Representations of Graphs

In C programming, graphs can be represented in various ways, depending on the requirements and the graph's structure (e.g., directed, undirected, weighted, unweighted). The following slides explore some of the most common representations

Sparse vs. Dense Graphs

Sparse Graphs

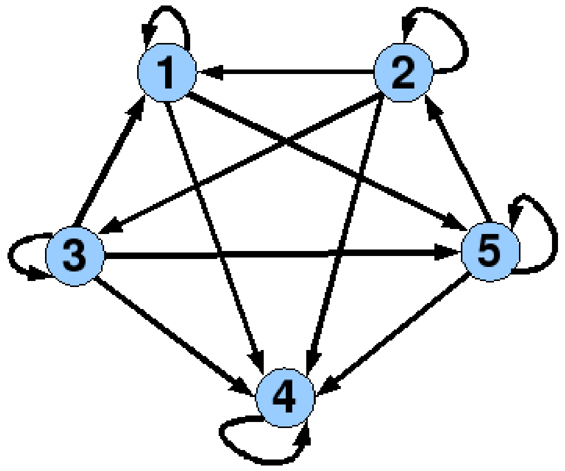
- A ***sparse*** graph is one for which: **$|E| \ll |V|^2$**, or the number of edges is significantly less than the square of the number of vertices
- Example:



5 « 25

Dense Graphs

- A ***dense*** graph is one for which:
- It's not the case that **$|E| \ll |V|^2$**
- Example:

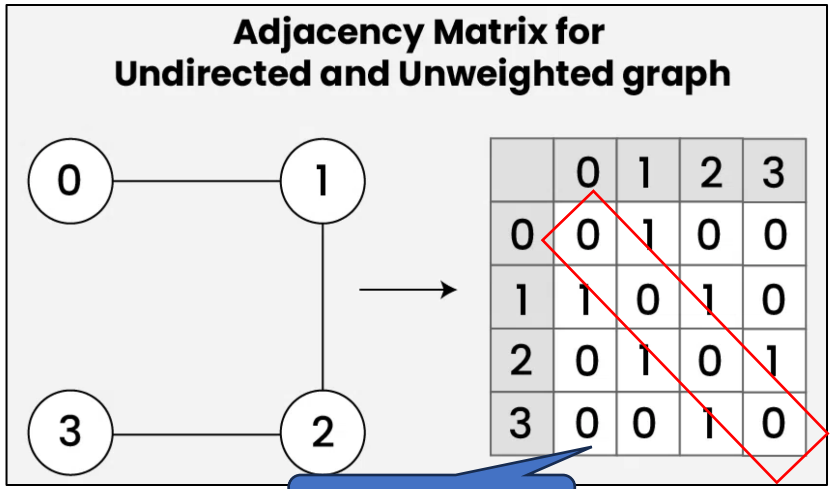


It's not the case that **15 « 25**

Source:https://www.d.umn.edu/~gshute/cs2511/slides/graphs

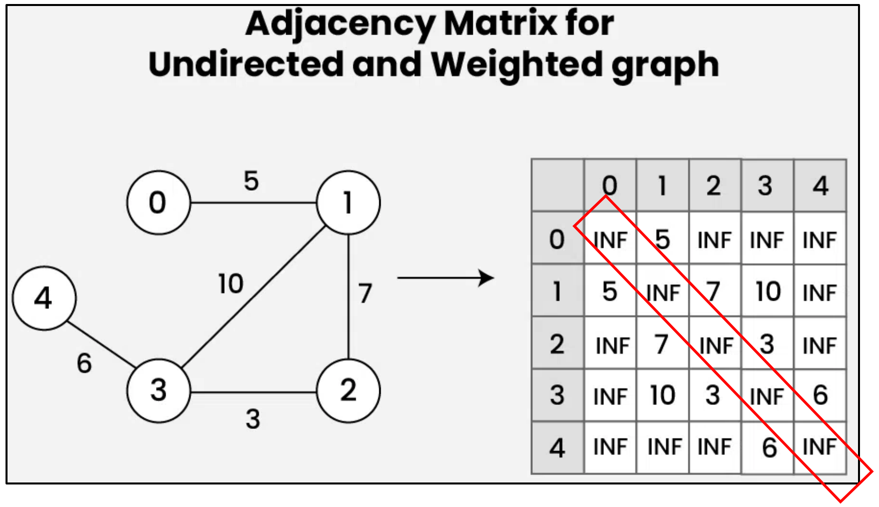
Adjacency Matrix for Undirected Graph

- Each vertex is associated with an integer index from ***0*** to ***n*** - 1
- Two-dimensional (*Boolean)* adjacency array M of size ***n*** by ***n***
- <u>The adjacency matrix is space-efficient only for **dense graphs**.</u>
- Easy to implement.



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

Symmetrical



	0	1	2	3	4
0	INF	5	INF	INF	INF
1	5	INF	7	10	INF
2	INF	7	INF	3	INF
3	INF	10	3	INF	6
4	INF	INF	INF	6	INF

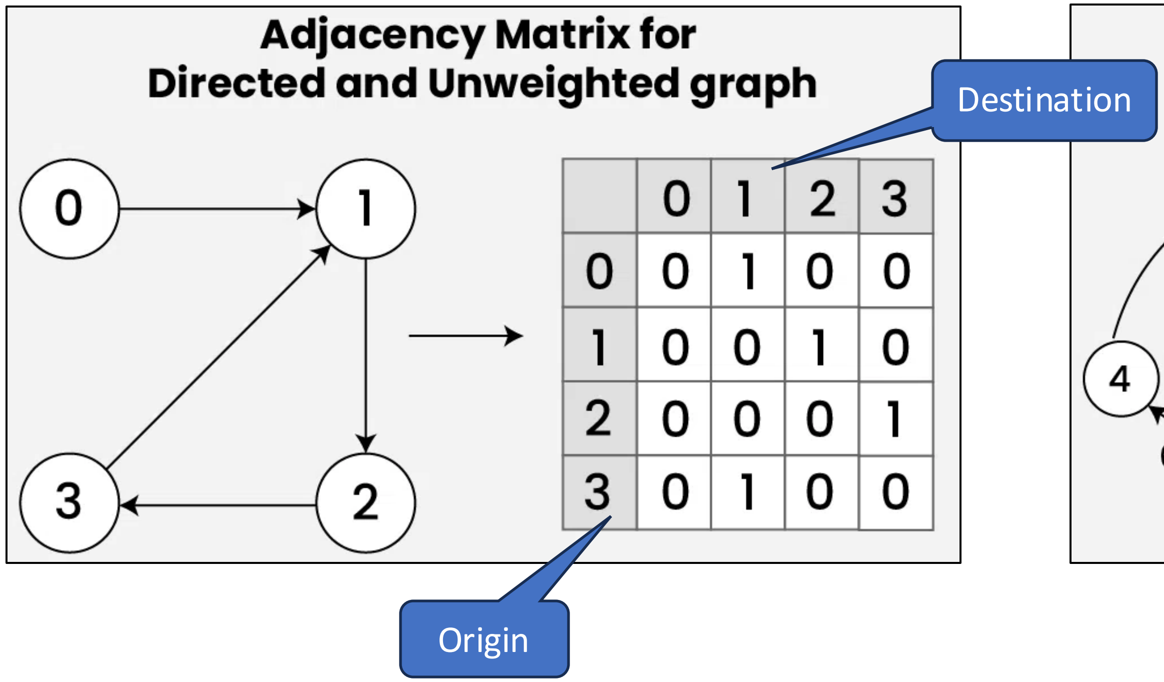
Source: https://www.geeksforgeeks.org/adjacency-matrix/

Adjacency Matrix Implementation for Undirected Graph

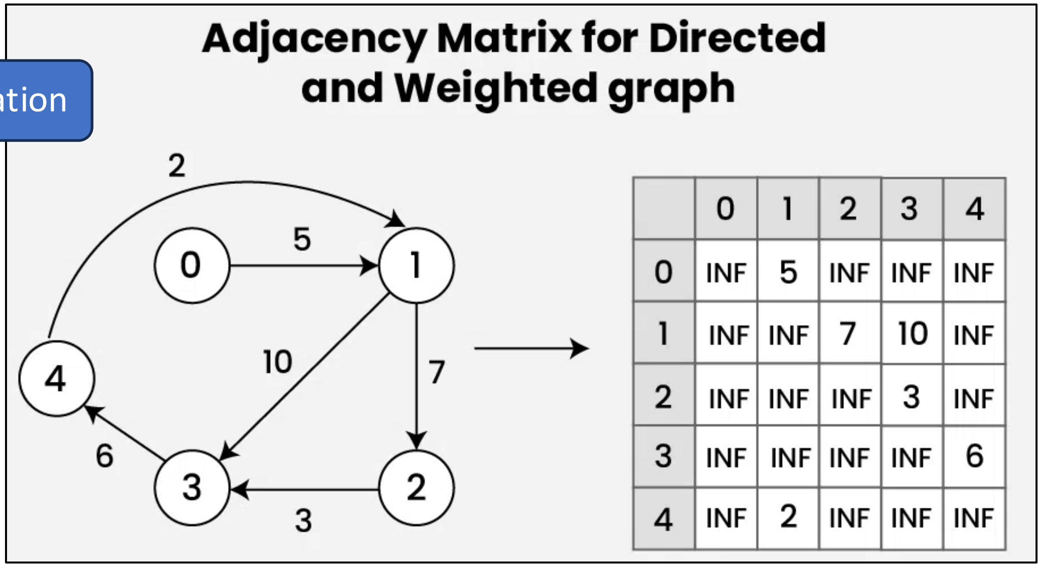
```
#include<stdio.h>
#define V 4
void addEdge(int mat[V][V], int i, int j) {
// Since the graph is undirected
    mat[i][j] = 1;
    mat[j][i] = 1;
}

void displayMatrix(int mat[V][V]) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
}
```

```
int main() {
    // Create a graph with 4 vertices and no edges
    // Note that all values are initialized as 0
    int mat[V][V] = {0};

    // Now add edges one by one
    addEdge(mat, 0, 1);
    addEdge(mat, 0, 2);
    addEdge(mat, 1, 2);
    addEdge(mat, 2, 3);

    printf("Adjacency Matrix Representation\n");
    displayMatrix(mat);

    return 0;
}
```

Adjacency Matrix for Directed Graph



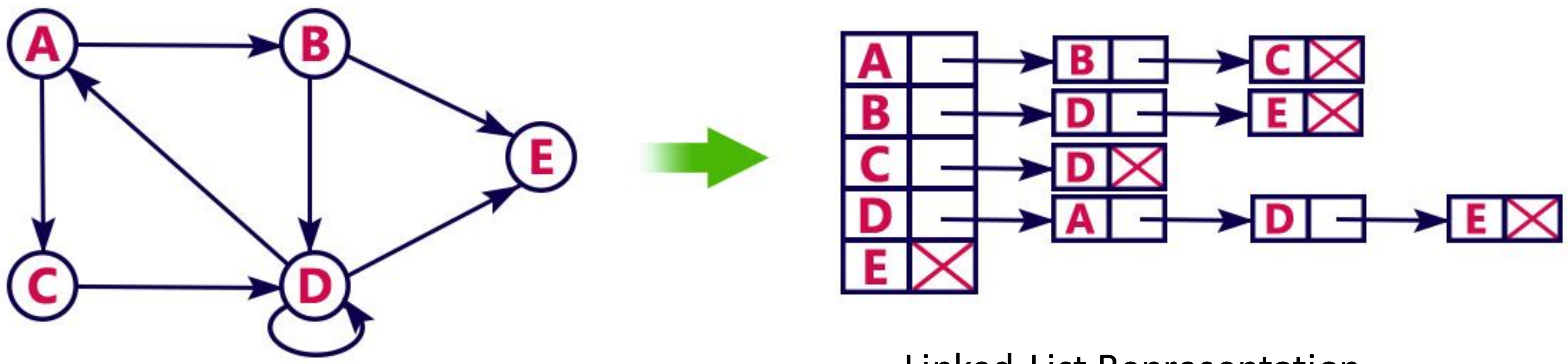
	0	1	2	3
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	1	0	0



	0	1	2	3	4
0	INF	5	INF	INF	INF
1	INF	INF	7	10	INF
2	INF	INF	INF	3	INF
3	INF	INF	INF	INF	6
4	INF	2	INF	INF	INF

Adjacency List for Directed Graph

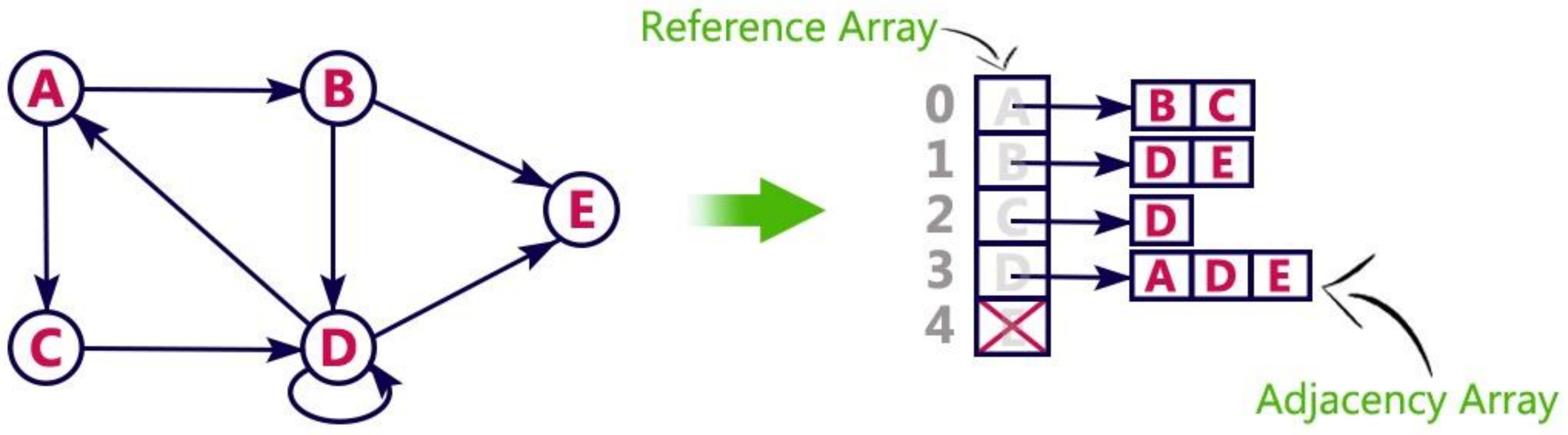
- An adjacency list is a collection of lists or arrays that represent a graph. In a directed graph, each vertex has a list of its neighbouring vertices.
- In the following representation, each vertex in the graph has a list of its adjacent vertices. For instance, take the following directed graph representation implemented with a linked list.



Linked-List Representation

Adjacency List for Directed Graph

- The same graph representation can also be implemented using an array as shown below.



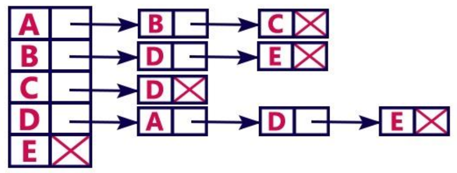
Array-Based Representation

The **adjacency array** is the flat list of all neighbours of each vertex, listed in order.

Source:http://www.btechsmartclass.com/data_structures/graph-representations.html

Adjacency List Implementation

```
// Function to create a graph
struct Graph* createGraph() {
struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

// Initialize the adjacency list
for (int i = 0; i < V; i++) {
   graph->adjList[i] = NULL;}
 return graph;
}
```

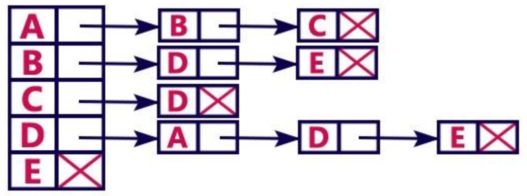


```
// Structure for adjacency list node
   struct Node {
   int dest;
   struct Node* next;};

// Structure for graph
  struct Graph {
  struct Node* adjList[V];};
```

Adjacency List Implementation (cont.)

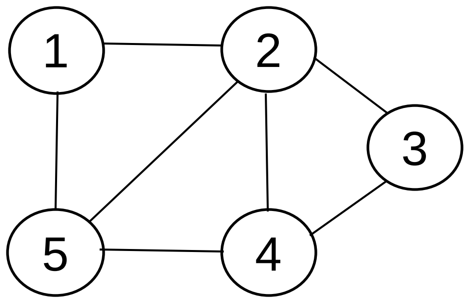
```
// Function to add an edge to the directed graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Create a new node
    struct Node* newNode = (struct
    Node*)malloc(sizeof(struct Node));
    newNode->dest = dest;
    newNode->next = graph->adjList[src];
    // Add the new node at the beginning of the list
    graph->adjList[src] = newNode;
}
```

```
// Function to display the adjacency list
void displayGraph(struct Graph* graph) {
  for (int i = 0; i < V; i++) {
    struct Node* temp = graph->adjList[i];
    printf("Vertex %d:", i);
      while (temp != NULL) {
      printf(" -> %d", temp->dest);
      temp = temp->next;
      }
  printf("\n");
  }
}
```

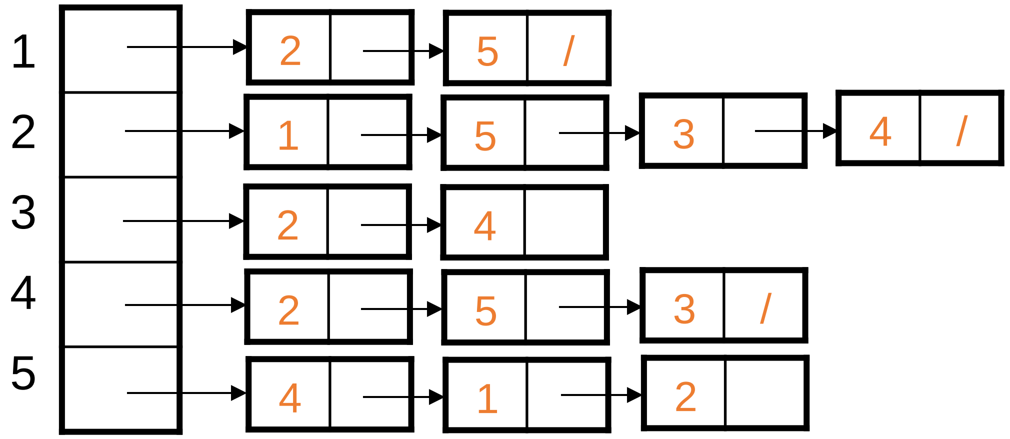


Adjacency List for Undirected Graph

- In an undirected graph, each vertex has a list of its neighbouring vertices.
- In the adjacency list representation for an **undirected graph**, for each edge between two vertices (say vertex u and vertex v), **we store v in the adjacency list of u, and we also store u in the adjacency list of v**. This is a key feature of undirected graphs, as it allows for bidirectional traversal of the edge.



Undirected graph



Advantages of Adjacency List Representation

- The adjacency list is **memory efficient**, especially for **sparse graphs** where the number of edges is much smaller than the number of vertices squared.
- It is easy to traverse the graph to find neighbours of a vertex.

Thank you