Data Structures

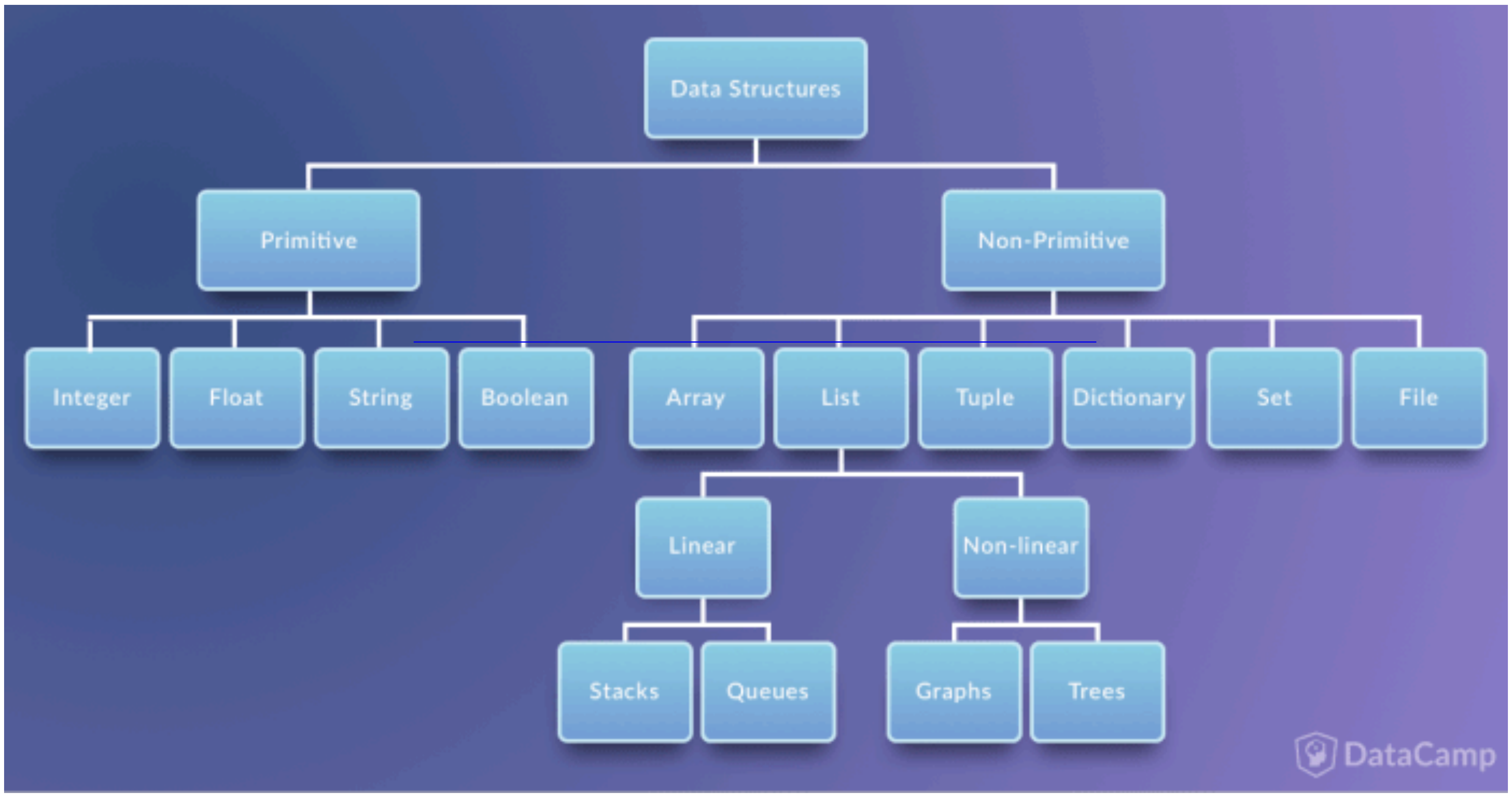
Lecture 11: Graphs

Nopadon Juneam
Department of Computer Science
Kasetsart university

Outlines

- Graph and its basic notions
  - Directed/undirected graph
  - Basic graph terminology
- Two standard graph representations
  - Adjacency list
  - Adjacency matrix
- Basic operations on graphs

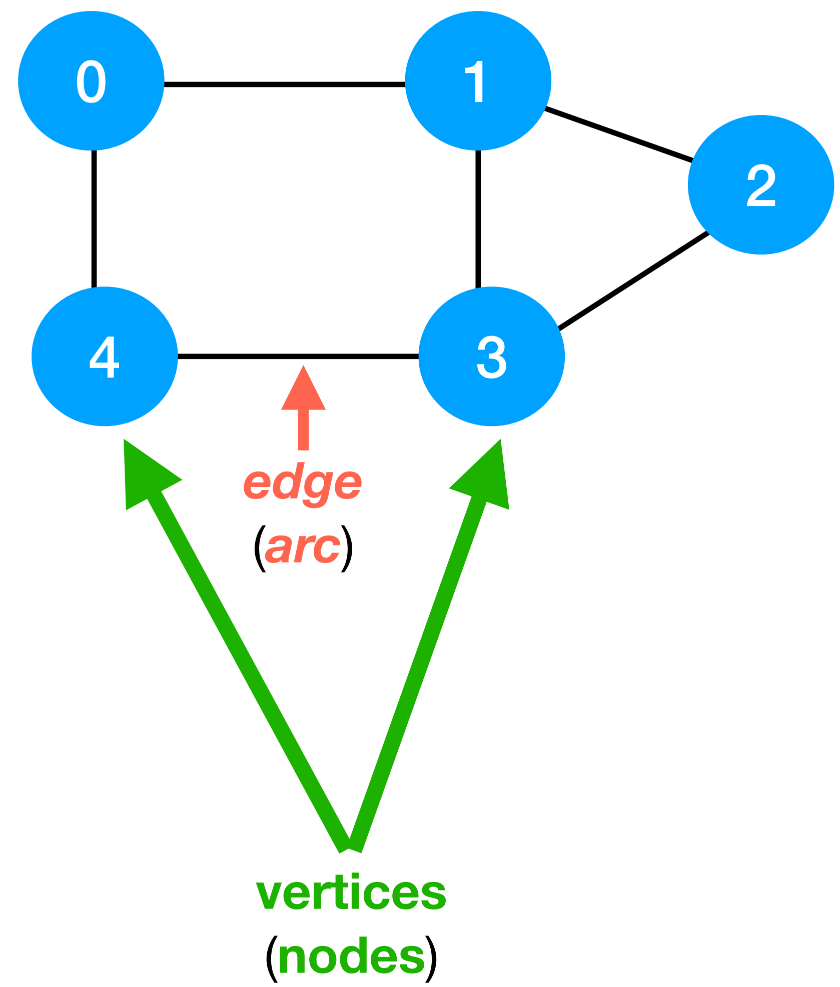
Classification of Data Structures



Source: https://www.datacamp.com/community/tutorials/data-structures-python

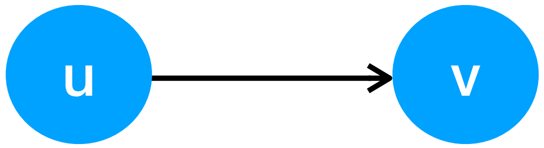
Graphs

- A **graph** is a *non-linear* data structure.
- Informally, a graph consists of a finite set of ***vertices*** (or ***nodes***) and a set of ***edges*** (or **arcs**) which connect a pair of nodes.
- In the example, a graph is given by
  - The set of vertices $V = \{0, 1, 2, 3, 4\}$.
  - The set of edges $E = \{\{0,1\}, \{0,4\}, \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\}\}$.

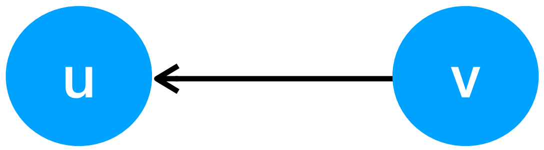


Directed/undirected Edge

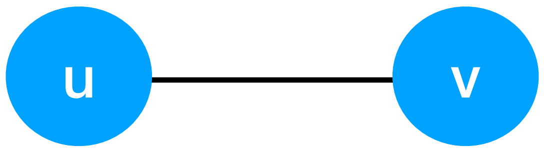
- Edges in a graph are either ***directed*** or ***undirected***:
  - An edge (u, v) is said to be *directed* from vertex u to vertex v if the pair (u, v) is *ordered*, with *u preceding v.*
  - An edge (u, v) is said to be *undirected* if the pair (u, v) is *unordered.*
    - Note that sometimes we denote undirected edges with set notation, as {u, v}.



a directed edge (u, v)



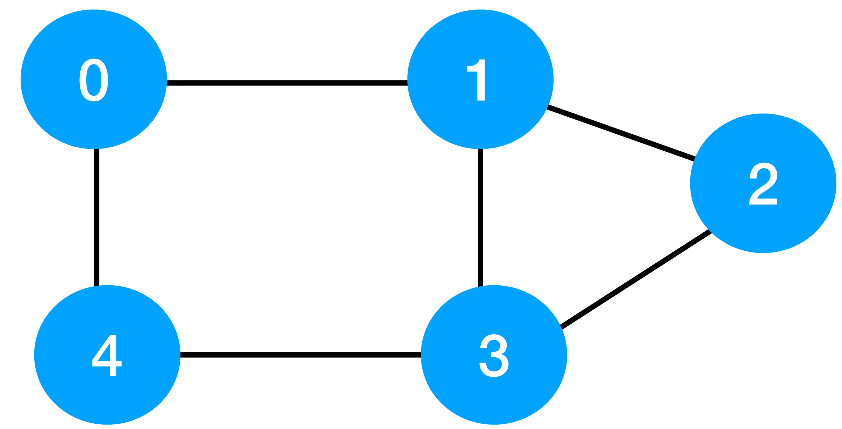
a directed edge (v, u)



an undirected edge {u, v}

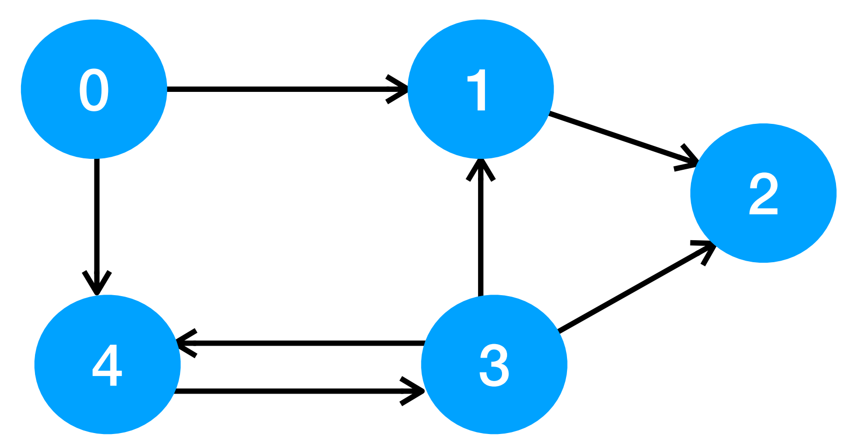
Directed/undirected Graph

- If all the edges in a graph are undirected, then we say the graph is an ***undirected graph***.



Undirected graph

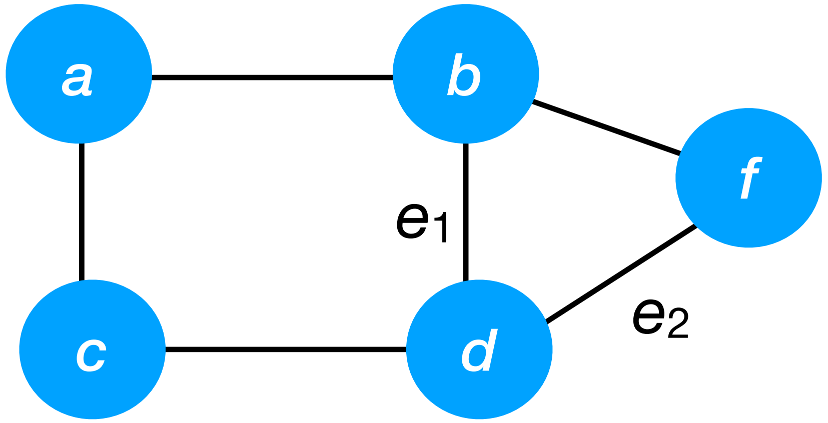
- Likewise, a **directed graph**, a graph whose edges are all directed.



Directed graph

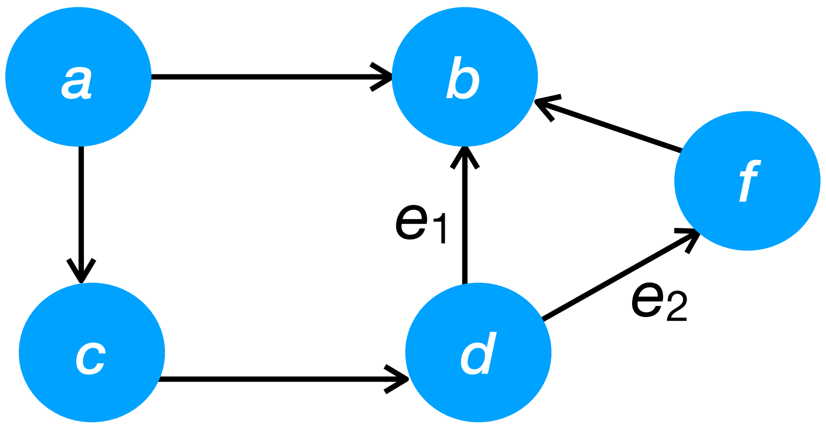
Basic Graph Terminology (1)

- The two vertices connected by an edge are called the ***end-vertices*** (or ***endpoints***) of that edge.
  - For example, *b* and *d* are the end-vertices of edge e_1.
- Two vertices are said to be ***adjacent*** if they both are the end-vertices of the same edge.
  - For instance, *d* and f are adjacent.
- An edge is said to be ***incident*** on a vertex if the vertex is one of the edge's end-vertices.
  - For instance, e_1 and e_2 are incident edges of *d.*
- The ***degree*** of a vertex *v*, denoted by *deg*(*v*), is the number of incident edges of *v*.
  - For instance, *deg*(*f*) = 2.



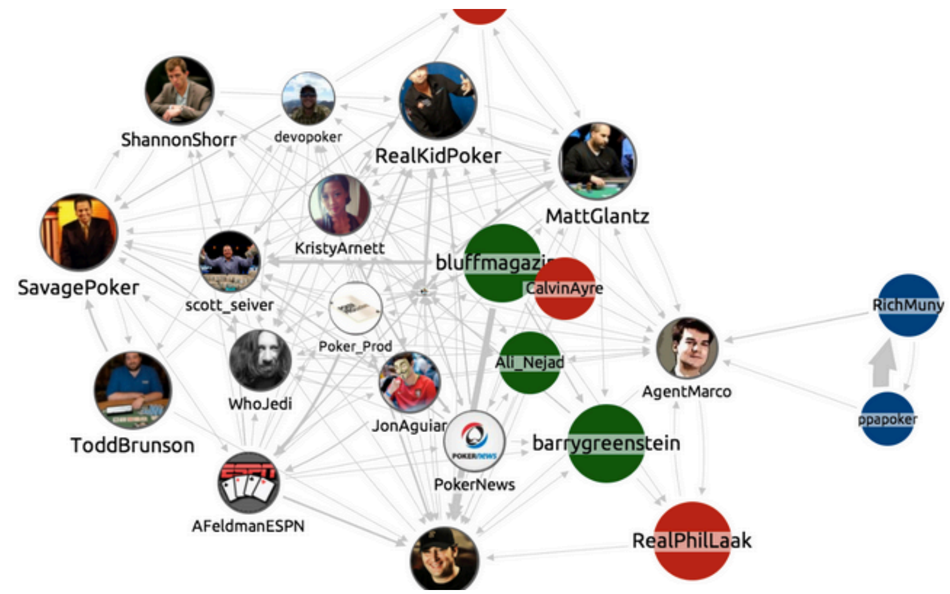
Basic Graph Terminology (2)

- If an edge is directed, its first endpoint is its ***origin*** and the other is the ***destination*** of the edge
    - The ***outgoing edges*** of a vertex are the directed edges whose origin is that vertex.
        - For instance, e_1 and e_2 are the outgoing edges of *d.*
    - The ***incoming edges*** of a vertex are the directed edges whose destination is that vertex.
        - For instance, e_2 is the incoming edge of *f.*
    - The ***in-degree*** and ***out-degree*** of a vertex *v* are the number of the incoming and outgoing edges of *v,* denoted by *indeg*(*v*) and *outdeg*(*v*), respectively.
        - For instance, *indeg*(*d*) = 1; out*deg*(*d*) = 2.

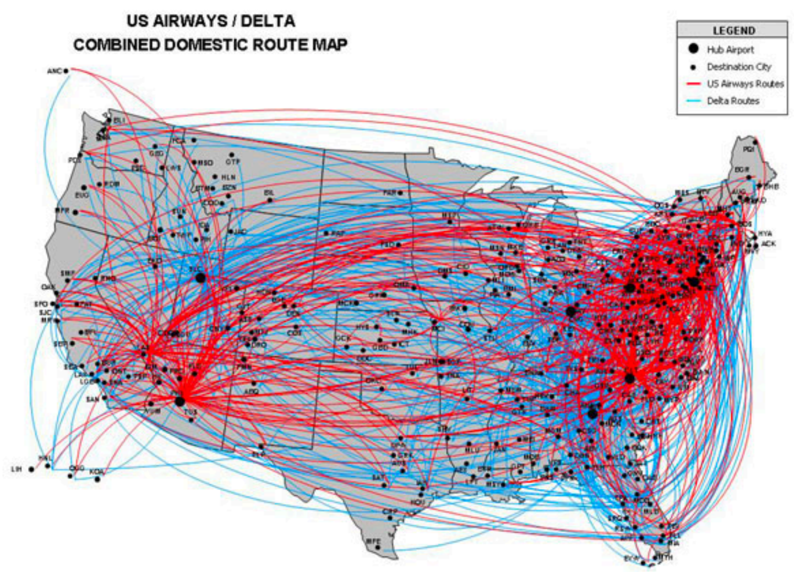


Graph Applications

Study of road networks [1]



Study of social networks [2]



Study of flight networks [3]

[1] source: https://notes.zouhairj.com/google-maps-algorithm-work-find-efficient-route/
[2] source: https://cambridge-intelligence.com/
[3] source: http://passyworldofmathematics.com/traversable-and-hub-networks/

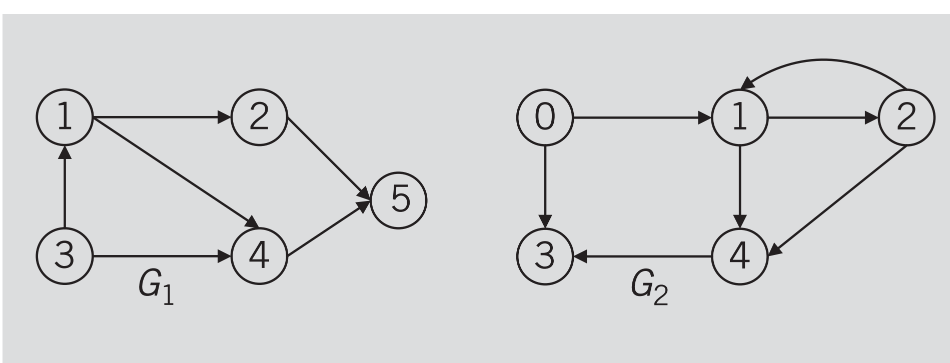
Graph Representations

- The most commonly used representations for graphs:
  - **Adjacency matrix**
  - **Adjacency list**
- There are other representations e.g. i*ncidence matrix* and *incidence list*. However, the choice of the graph representations is situation specific. It depends on the type of operations to be performed and ease of use.

Adjacency Matrix

- The ***adjacency-matrix representation*** is a 2D array of size *n x n*, where *n* is the number of vertices in a graph.

- The (*i*, *j*)-th entry of the array is 1 if there is an edge from vertex *i* to vertex *j*; otherwise, the (*i*, *j*)-th entry is 0.

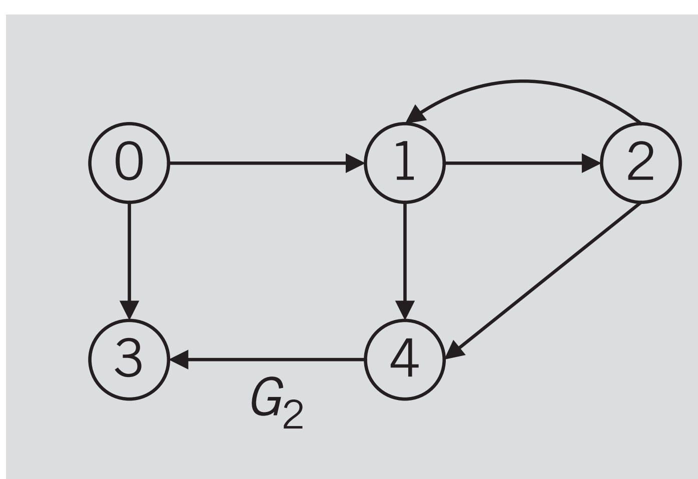


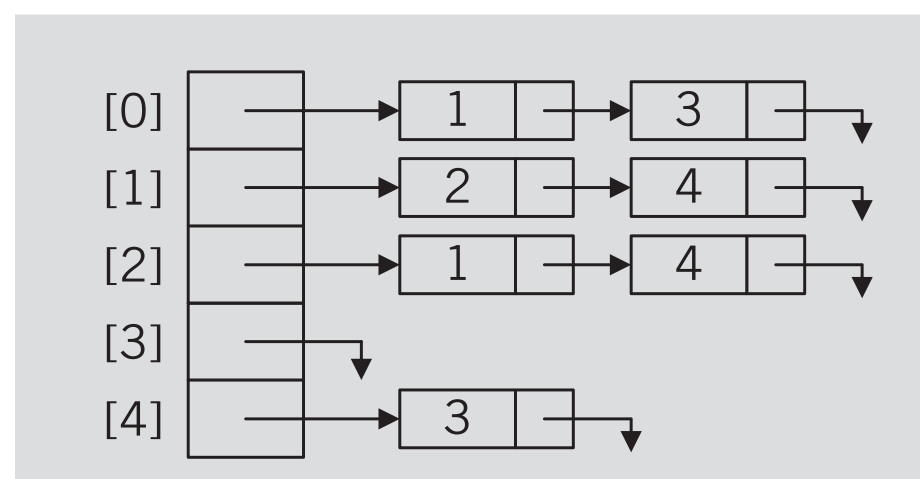
$$A_{G_1} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad A_{G_2} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency List

- The ***adjacency-list representation*** of a graph *G* consists of an array *Adj* of *n* lists, where *n is* the number of vertices in a graph; one list for each vertex in V.

- For each *u* in *V*, the adjacency list *Adj*[*u*] contains all the vertices *v* such that there is an edge (*u*, *v*) in *E.* In other words, *Adj*[*u*] consists of all the vertices adjacent to *u.*





Basic Operations on Graphs

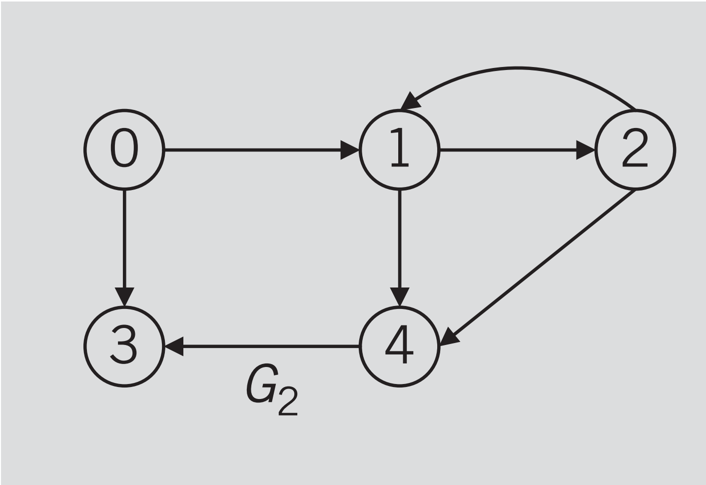
- Basic operations commonly performed on a graph:
  - Create the graph
  - Add an edge to the graph
  - Print the graph

Basic Graph Operations Using Adjacency-Matrix Representation (1)

```
 // A simple adjacency-matrix representation of graph using 2D array
#include<stdio.h>
#include<stdlib.h>
// Function to create a graph with n vertices
int** createGraph(const int n) {
    // Return 2D array of size n*n
    int** adjMatrix = malloc(sizeof(int*)*n);
    for (int i=0; i<n; i++) {
        adjMatrix[i] = malloc(sizeof(int)*n);
        for (int j=0; j<n; j++)
            adjMatrix[i][j] = 0;
    }
    return adjMatrix;
}
//Function to add a directed edge into the graph
void addEdge(int** adjMatrix, int u, int v) {
    adjMatrix[u][v] = 1;
}
// Function to print the adjacency matrix of the graph
void printGraph(int** adjMatrix, int n)
{
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }
}
```

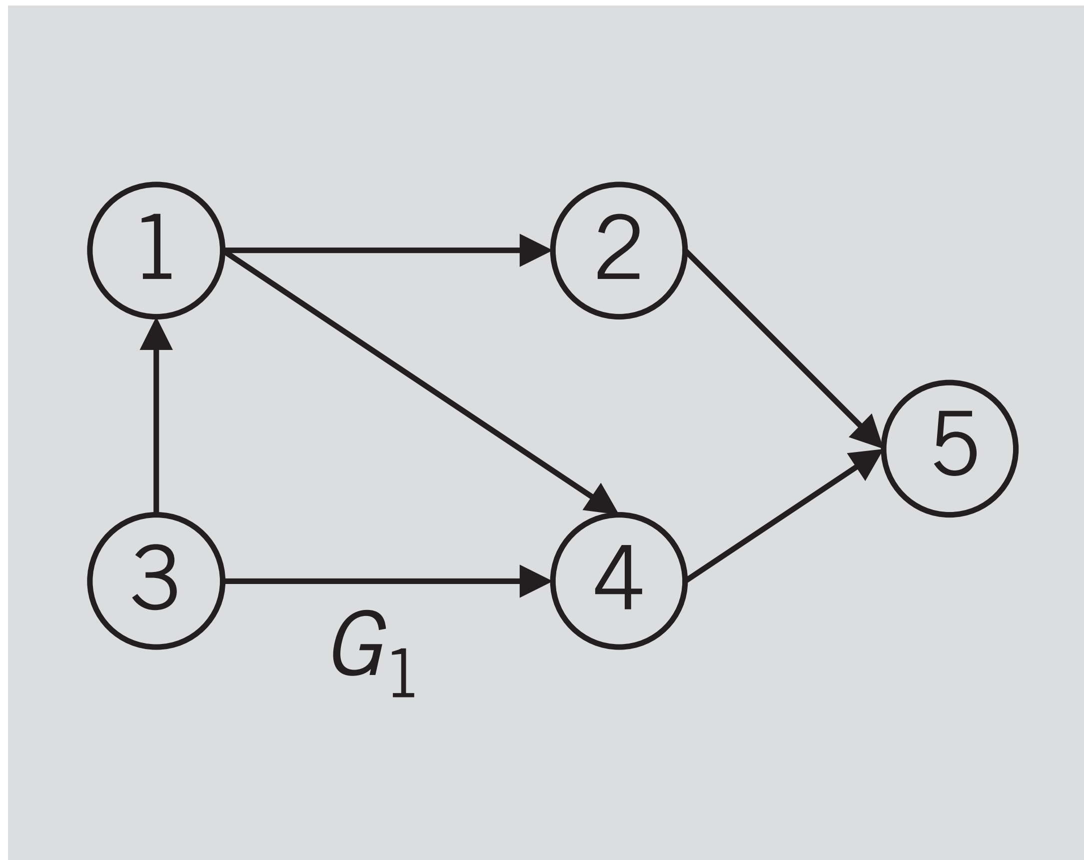
Basic Graph Operations Using Adjacency-Matrix Representation (2)

```
// Driver code
int main()
{
    int n = 5;

    int** adjMatrix = createGraph(n);

    //Vertex numbers should be from 0 to 4
    addEdge(adjMatrix, 0, 1);
    addEdge(adjMatrix, 0, 3);
    addEdge(adjMatrix, 1, 2);
    addEdge(adjMatrix, 1, 4);
    addEdge(adjMatrix, 2, 1);
    addEdge(adjMatrix, 2, 4);
    addEdge(adjMatrix, 4, 3);

    printGraph(adjMatrix, n);
    return 0;
}
```




```
Lasalle:codes dmodify$ ./a.out
0 1 0 1 0
0 0 1 0 1
0 1 0 0 1
0 0 0 0 0
0 0 0 1 0
```

Programming Exercise



- Let’s try to create the above graph using C code