Department of Computer science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

GRAPHS

Abstract

Introduction, Properties, Representation of graphs: Adjacency Matrix, Adjacency List

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

Overview:

Graph G is a pair (V, E), where V is a finite set of vertices and E is a finite set of edges. We will often denote n = |V|, e = |E|. It consists of set of nodes and arcs. Each arch in a graph is specified by a pair of nodes.

There are three types of graph

1. Directed graph

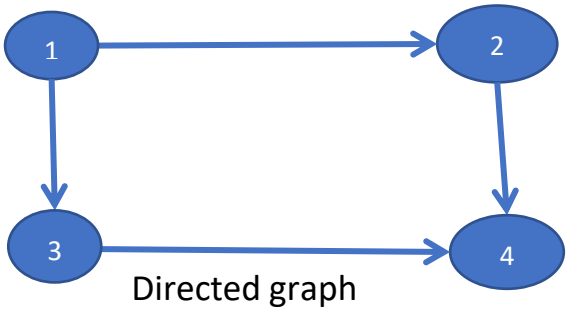
2. Undirected graph

3. Weighted graph

1. Directed graph: A graph G= (V, E) in which every edge is directed is called directed graph. In directed graph pair of vertices representing any edge is ordered. For example in the below fig < v_1, v_2 > and < v_2, v_1> represents the different edge. Directed graph is also known as digraph.

Examples: 1

2.



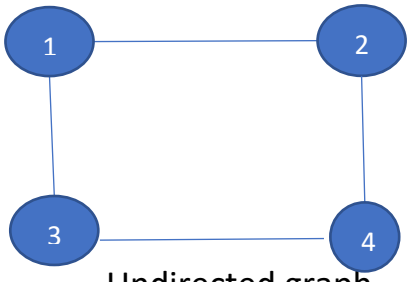
Directed graph

In the above Example.2 V={1,2,3,4} is set of vertices and E={<1,2>,<2,4>,<3,4>,<1,3>} is set of edges. Since all the edges are directed it is a directed graph

2. Undirected graph: The graph G= (V, E) in which every edge is undirected .In undirected graph the pair of vertices representing any edge is unordered.
Example1: The < v_1, v_2 > and < v_2, v_1> represents the same edge.

Example.1

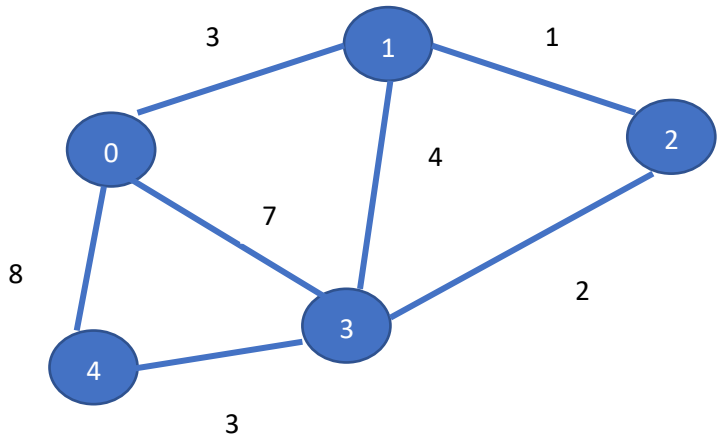
Example.2



Undirected graph

In the above example.2 V={1,2,3,4} is the set of vertices and E={(1,2),(1,3),(2,1),(2,4),(4,2),(3,4),(4,3),(3,1)} is the set of edges. Since all the edges are undirected the above graph is undirected graph

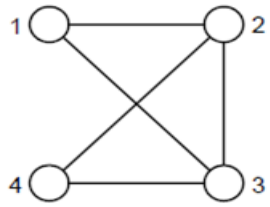
3. Weighted graph or Network: A weighted graph is a graph where each edge has a numerical value called weight. Given below is an example of a weighted graph. Weighted graph can be directed and undirected.



Weighted graph

Graph terminologies:

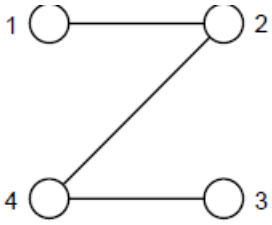
Adjacent Nodes: When there is an edge from one node to another then these nodes are called adjacent nodes.



Adjacent Node

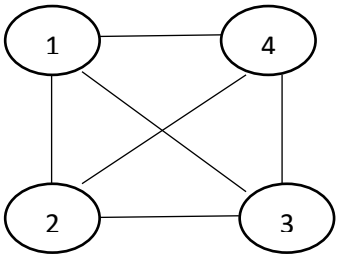
In the above graph 2 is adjacent to 3 and 4 is adjacent to 3 and so on

Path: A path from edges u_0 to a node u_n is a sequence of nodes u_0, u_1, u_2.....u_{n-1},u_n. Here u_0 is adjacent to u_1, u1 is adjacent to u_2 and u_{n-1} is adjacent to u_{n-1}. In the below graph, the path from vertex 1 to 3 is denoted by (1,2,4,3) which can also be written as (1, 2), (2, 4), (4, 3).



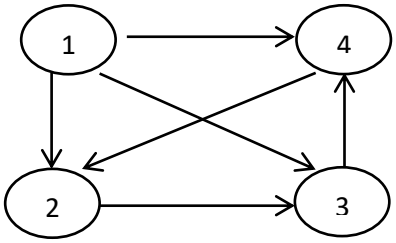
Path

Length of the path: Length of the path is the total number of edges included in the path from the source node to destination node.



Undirected graph

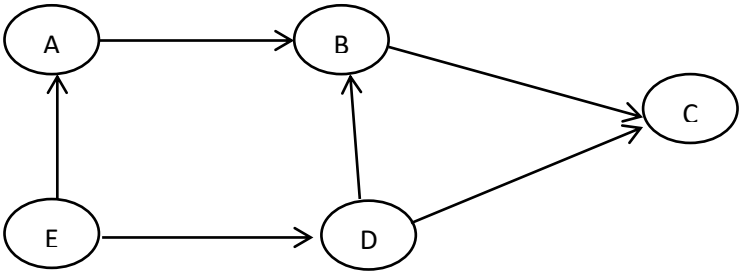
For example in the above undirected graph ,the path(1,2,3,4) has length 3 since there are three edges(1,2),(2,3),(3,4). The path 1, 2, 3 has length 2 since there are 2 edges (1, 2), (2, 3).



Directed graph

Similarly in the directed graph, the path <1,2,3,4> has length 3 since there are 3 edges <1,2>,<2,3>,<3,4> and the path <1,2,3> has length 2 since there are 2 edges <1,2>,<2,3>.

Degree: In an undirected graph, the total number of edges linked to a node is called degree of the node. In digraph there are 2 degrees for every node called indegree and outdegree.

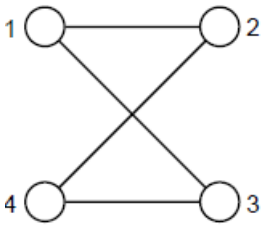


Degree

In-degree: The in-degree of a node n is the total number of arcs that have n as the head. For example C is receiving 2 edges so indegree of C is 2.

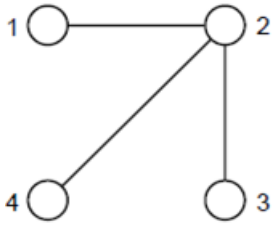
Outdegree: The outdegree of a node n is the total number of arcs that have n as the tail. For example outdegree of D is 1.

Cycle graph: A path from node to itself be called a cycle or cycle is path in which first and last vertices are same. A graph with at-least one cycle is called cyclic graph. A directed acyclic graph is called dag. For example the path<1, 2, 4, 3, and 1> is cycle since the first node and the last node are same. It can be represented as <1,2>,<2,4>,<4,3>,<3,1>.



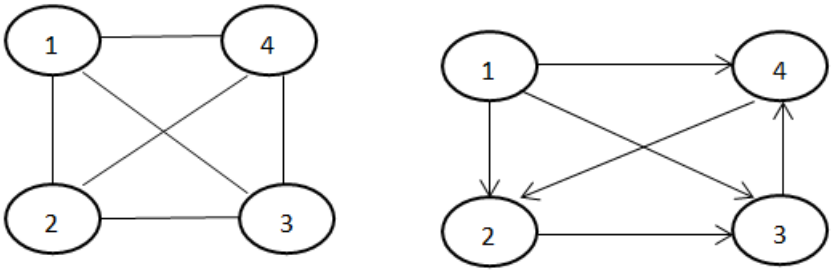
Cycle

Acyclic graph: A graph with no cycle is called acyclic graph. Tree is an acyclic graph.



Acyclic graph

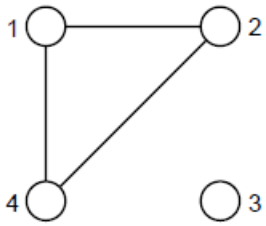
Connected: A graph is called connected if there is a path from any vertex to any other vertex. A tree is defined as connected undirected graph with no cycles. Example



Connected graph

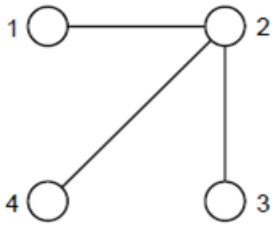
Disconnected graph: if there exist atleast one vertex in a graph that cannot be reached from the other vertices in the graph,then such graph is aclled

disconnected graph.In the below example vertex 3 cannot be reached by any other vertices in the graph.



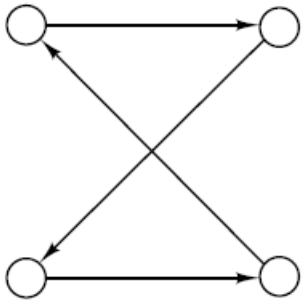
Disconnected graph

Tree: A tree is defined as connected undirected graph with no cycles.



Tree

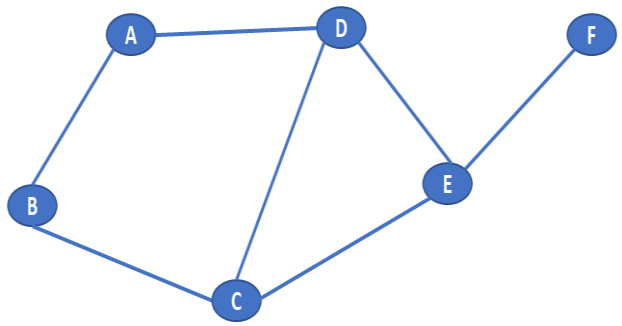
Directed cycle: In directed graph all the edges in a path or cycle have same direction



Directed cycle

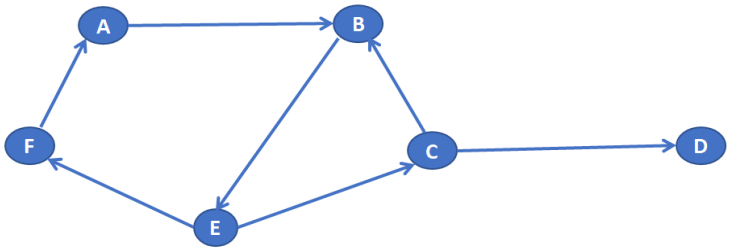
Properties of graph (referred from geeks for geeks):

1. **Undirected graph**
   a. The number of possible pairs in an m vertex graph is m*(m-1).
   b. The number of edges in an undirected graph is m*(m-1)/2 since the edge (u, v) is same as the edge(v, u).



Undirected graph

2. **Directed graph**
   a. The number of possible pairs in an m vertex graph is m*(m-1)
   b. The number of edges in an directed graph is m*(m-1) since the edge(u, v) is not the same as the edge(v, u)
   c. The number of edges in an directed graph is<=m*(m-1)



Directed graph

Representation of graph

Graph representation is the method to store the graphs in computer memory. Graph is represented using a set of vertices, and for each vertex, the vertices are

directly connected to it by an edge. For a weighted graph weight will be associated with each edge. The choice of graph representation depends on the application and the types of operation performed and ease of use.

There are 2 ways of representing the graph

1. Adjacency matrix.
2. Adjacency List.

1. Adjacency Matrix:

Let G= (V, E) be a graph where V is the set of vertices and E is the set of edges. Let N be the number of vertices in the graph G. The adjacency matrix A of a graph G is defines as

A[i][j] = 1 if there is an edge from vertex i to j

0 if there is no edge from vertex i to j

The adjacency matrix of a graph is a Boolean square matrix or bit matrix with n rows and n columns with entries zeros and ones.

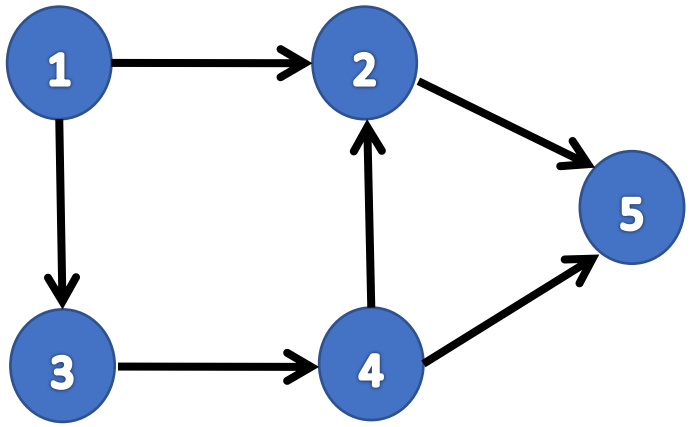
In an undirected graph, if there exist an edge (i, j),then A[i][j] and A[j][i] is made one since (i, j) is similar to (j, i).

In the directed graph if there exist an edge <i,j>, then A[i][j]=1

If there exist no edge between vertex i and j then A[i][j]=0

The matrix is symmetric in case of undirected graph, while it may be asymmetric if the graph is directed.

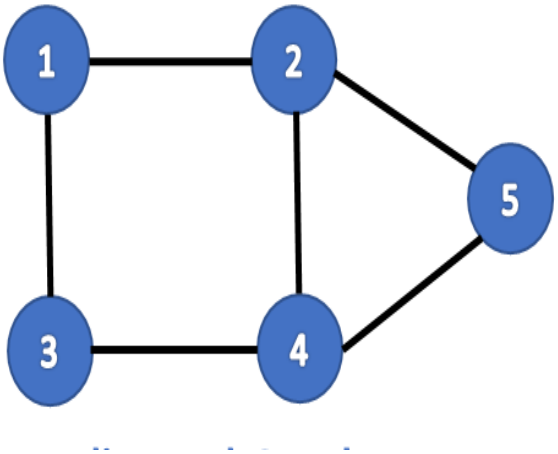
1. Directed graph



Directed graph

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	0	1
3	0	0	0	1	0
4	0	1	0	0	1
5	0	0	0	0	0

2. Undirected graph

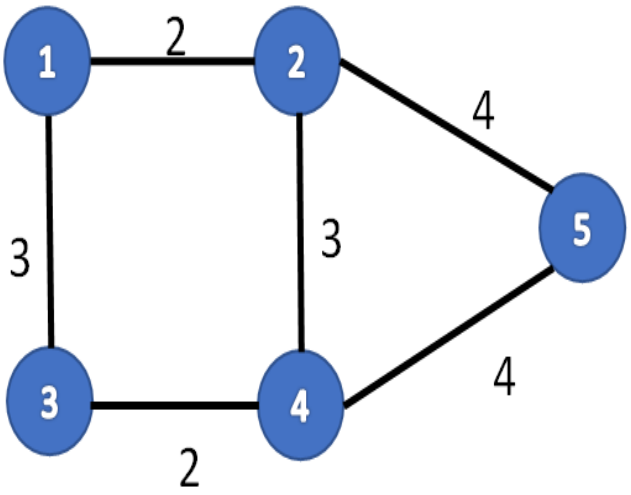


Undirected graph

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

3. Weighted graph:

In the weighted graph, distance or cost between the nodes are represented on the edge cost/distance value specified on the edge between adjacent nodes are used for representation in the adjacency matrix.



Weighted graph

	1	**2**	**3**	**4**	**5**
1	0	2	3	0	0
2	2	0	0	3	4
3	3	0	0	2	0
4	0	3	2	0	4
5	0	4	0	4	0

C representation of graph:

The number of nodes in the graph is constant, that is arcs may be added or deleted but the nodes may not. A graph with 25 nodes could be declared as

A graph with 25 nodes can be declared as

```
#define MAX 25

struct node

{

                    //information associated with each node

};

struct arc

{

                    int adj;//information associated with each arc

};

struct graph

{

                    struct node nodes[MAX];

                    struct arc arcs[MAX][MAX];

};

struct graph g;
```

Each node in a graph is represented by an integer number from zero to MAX-1, and the array field nodes represent the appropriate information assigned to each node. The array field an arcs is a two dimensional array representing every possible ordered pair of nodes. The value of g.arcs[i][j].adj is either one or zero depending on whether a node i is adjacent to j.

A weighted graph with fixed number of nodes can be declared as

```
struct arc
{
                    int adj;
                    int weight;
};
struct arc g[MAX][MAX];
```

Drawback of adjacency matrix representation of graph is it requires advance knowledge of the number of nodes. A new matrix must be created for each addition and deletion of a node. Adjacency matrix representation is inefficient in real world situation where a graph may have hundreds of nodes. An alternate solution to this is to use adjacency list.

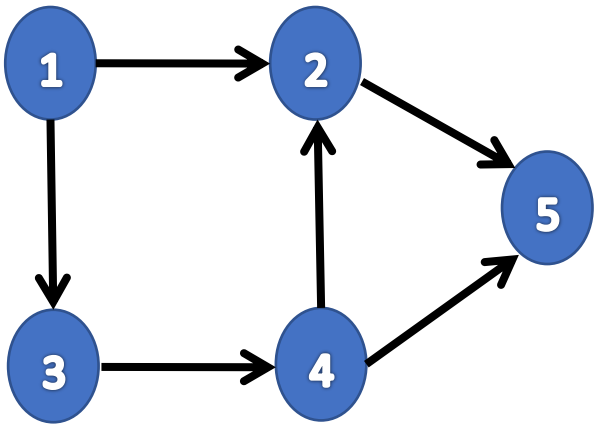
Function to read adjacency matrix

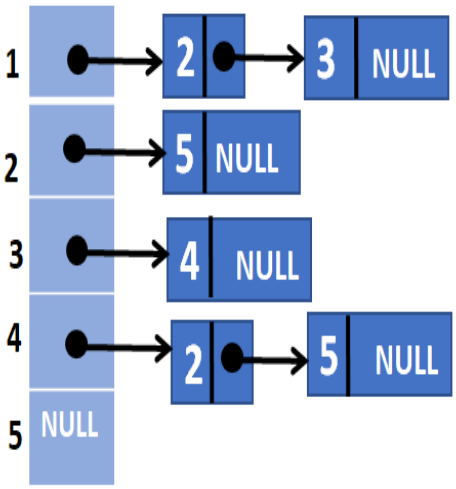
```
void read_admat(int A[10][10], int n)
{
    int I,j;
     for(i=0; i<n; i++)
    {
          for(j=0; j<n; j++)
           {
              scanf("%d", &A[i][j]);
            }
     }
}
```

2. Adjacency Linked List:

Adjacency linked list is an array of n linked list in which every vertex of a graph contains the list of adjacent vertices

1. Directed graph





Nodes adjacent to 1 are 2 and 3

Nodes adjacent to 2 is 5

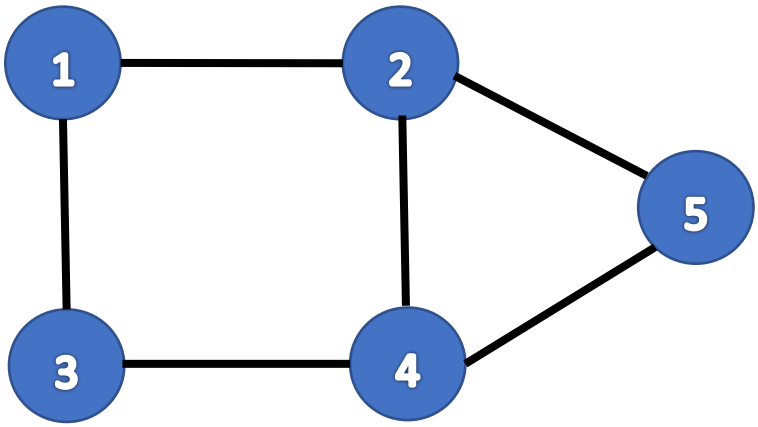
Nodes adjacent to 3 is 4

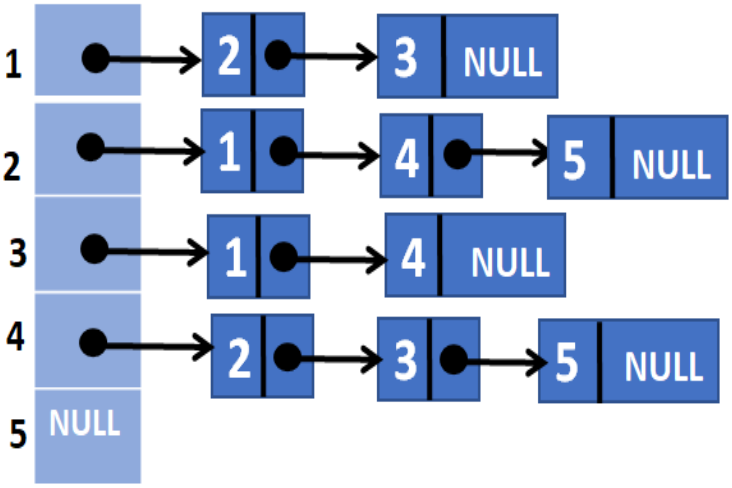
Nodes adjacent to 4 is 2 and 5

No Nodes adjacent to 5

Directed graph

2. Undirected graph





Nodes adjacent to 1 are 2 and 3

Nodes adjacent to 2 are 1, 4 and 5

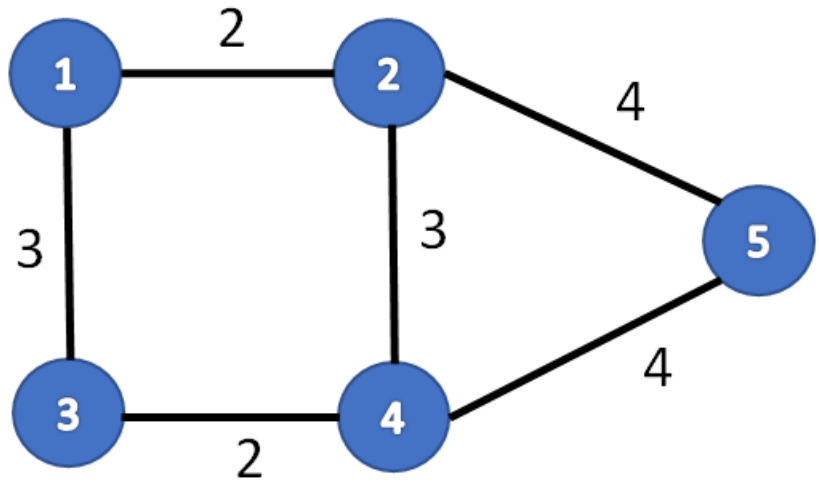
Nodes adjacent to 3 are 1 and 4

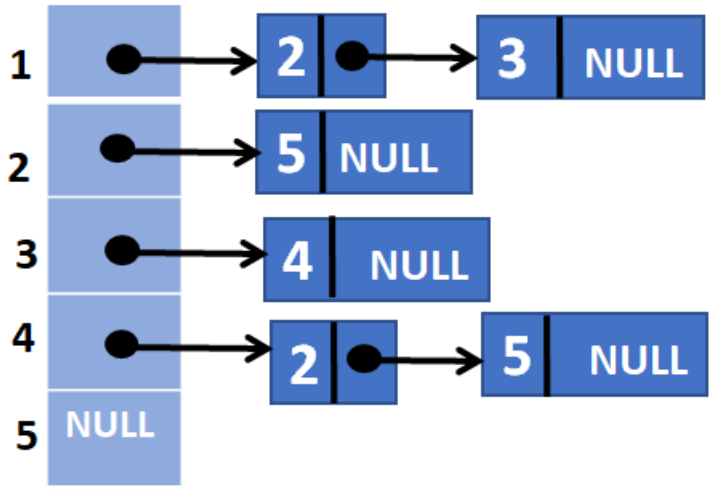
Nodes adjacent to 4 are 2, 3 and 5

No Nodes adjacent 5

Undirected graph

3. Weighted graph





Weighted graph

C representation of adjacency list

Two types of allocated nodes are needed since each graph carries some information.(arcs does not carry any info).

1. header nodes

arcptr	info	nextnode

A Sample header node

Each header node contains an info field and two pointers. The first of these to is to adjacency list of arcs emanating from the graph node, and the second is to next header node in the graph.

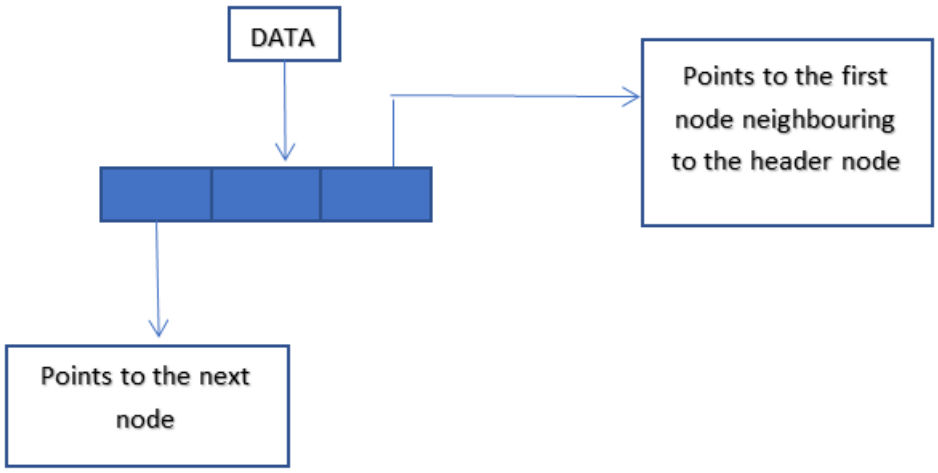
2. Adjacency list nodes

Ndptr	nextarc

A sample list node representing an arc

Each arc node contains two pointers, one for nextarc node in adjacency list and the other to the header node representing the graph that terminates the arc. Refer to the diagram from textbook T1 8.3.1 page no 543.

Header nodes and list nodes have different formats and must be represented by different structures. However for simplicity we make assumption that both header and the list nodes have same format and contain two pointers and a single integer information field.



1. Using array implementation the nodes are declared as

```
#define MAX 50

struct node

{

    int info;

     int point;

      int next;

};

struct node NODE[MAX];
```

In the case of header node, node[p] represents a graph node A, node[p].info represents the information associated with the graph node, node[p].next points to the next graph node and node[p].point points to the first list node representing an arc emanating from A. In the case of a list node, node[p] represents an arc <A, B>, node[p].info represents the weight of arc, node[p].point points to the header node representing the graph node B.

2. Using dynamic implementation

```
struct node

{

    int info;

    strut node *pointer;

    struct node *next;

};

struct node *nodeptr;
```

Function to read the adjacency list

```
void read_adlist(nodeptr a[],int n)
{
    int I,j,m ele;
    for(i=0;i<n;i++)
     {
        printf("enter the number of nodes adjacent to %d:",i);
         scanf("%d",&m);
         if(m==0) continue;
         printf("Enter the nodes adjacent to %d",i);
        for(j=0;j<m;j++)
        {
         scanf("%d",&ele);
        a[i]=insert_rear(ele,a[i]); //Function to insert an element at the rear of the list
         }
    }
}
```

Advantages of using Adjacency list:

1. Adding a new vertex is easy

2. Graphs can be stored in more compact form