DATA STRUCTURES AND ITS APPLICATIONS

Graph Representation

Sandesh B. J
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

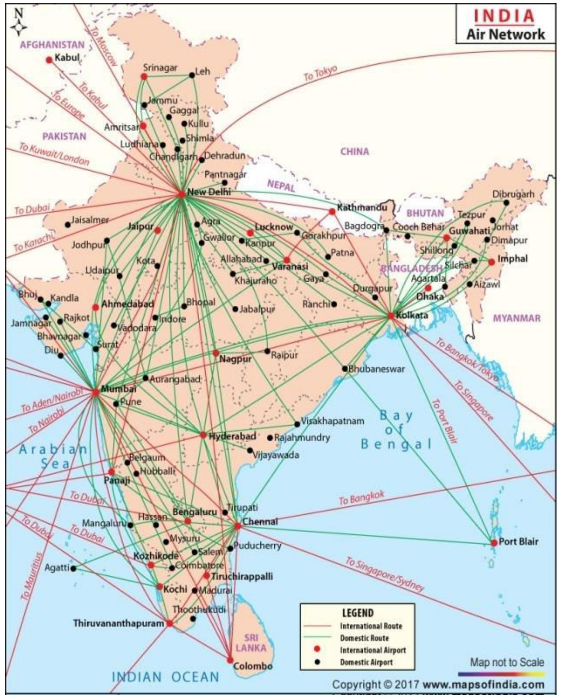
Graph Representation

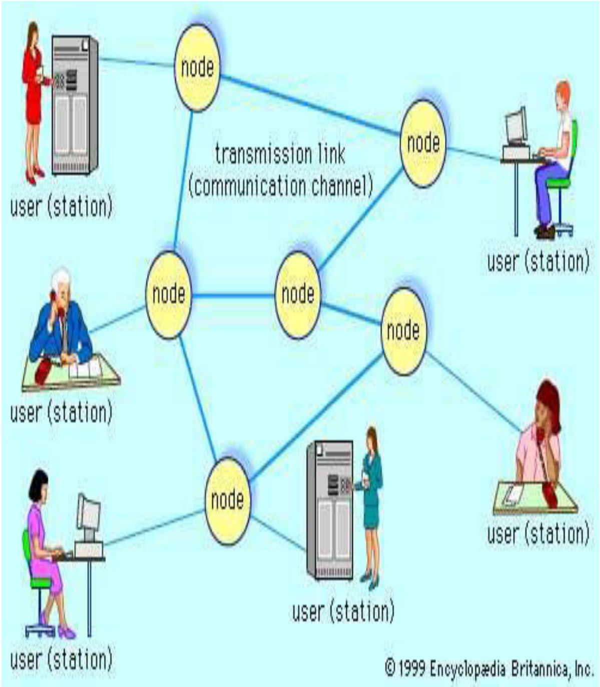
Sandesh B. J
Department of Computer Science & Engineering

Learning objectives:

- Why we need to represent the structure of the graph in computer memory?
- How do we represent the graph ?
- Data structure used for the representation:
  - ✓Adjacency Matrix and Adjacency List
- Representation : Directed, Undirected and Weighted graphs
- Implementation details of representation using ‘c’
- Multilinked Representation

Why Graph Representation?





Graph Representation

- How do we store mathematical structure of a graph in the computer memory?

- What types of data structures are used to represent the graph?

- Information Required to represent the graph
  - set of vertices of the graph
  - for each vertex neighbours of the vertex(edge information)

Graph Representation

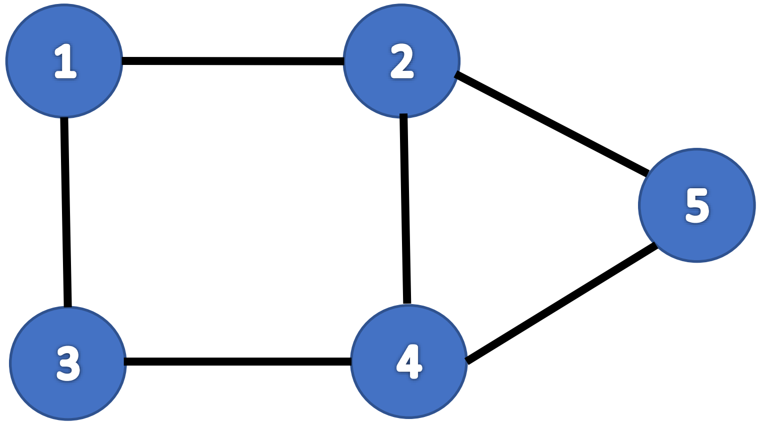
- Depending on the density of edges, ease of use and types of operation performed , Graphs can be represented by

  - ✓ Adjacency Matrix
    - Two Dimensional Array

  - ✓ Adjacency List
    - Linked List

Adjacency Matrix Representation – Undirected graph

- Adjacency matrix = n x n matrix M, graph of n vertices/nodes
- **M[i][j] = 1** if (i, j) is an edge
- **M[i][j] = 0 ,** no edge between the pair of vertices i and j



Undirected Graph

$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & \boxed{1} & 1 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 1 \\
3 & 1 & 0 & 0 & 1 & \boxed{0} \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
$$

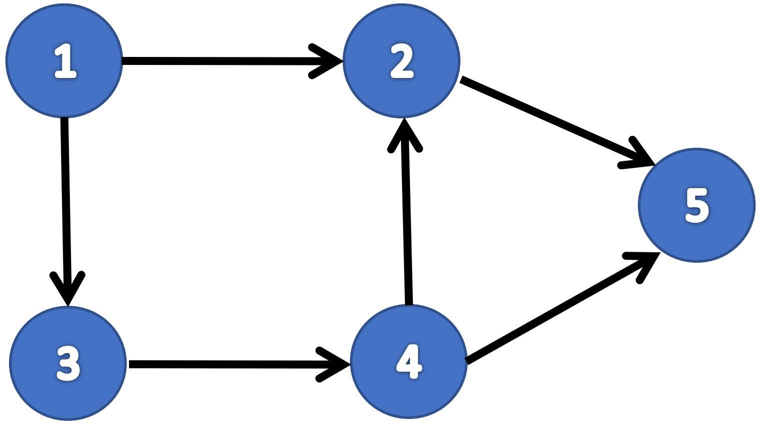
Main diagonal

Note:

- For undirected graph, M is symmetric i.e M[i][j] = M[j][i]
- Assume no edge from node to itself. So diagonal elements has value 0

Adjacency Matrix Representation – Directed graph

- In directed graph edge is directed
- In directed graph edge(i,j) is not equal to edge(j,i)
- Adjacency matrix is asymmetric



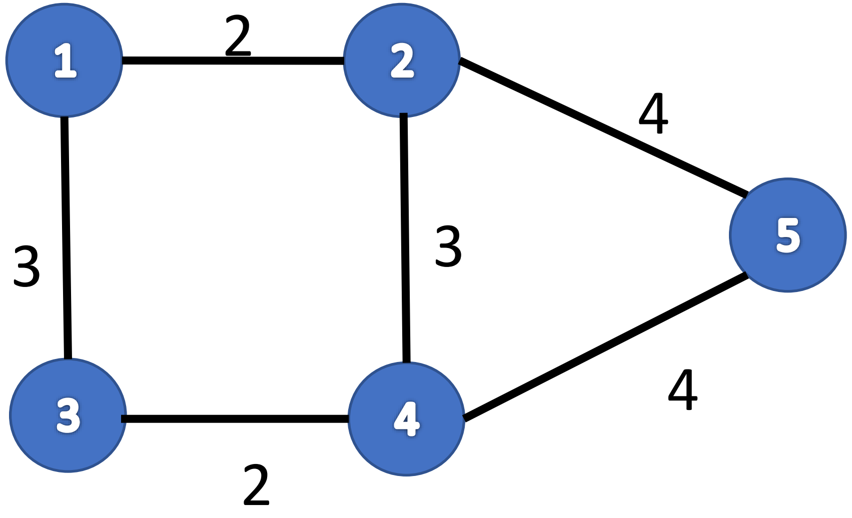
Directed Graph

$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

Main diagonal

Adjacency Matrix Representation – 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 stored in the adjacency matrix



Weighted Graph

$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
0 & 2 & 3 & 0 & 0 \\
2 & 0 & 0 & 3 & 4 \\
3 & 0 & 0 & 2 & 0 \\
0 & 3 & 2 & 0 & 4 \\
0 & 4 & 0 & 4 & 0
\end{bmatrix}
\end{array}
$$

Drawbacks of Adjacency Matrix Representation

- Number of nodes in the graph needs to be known in prior

- In case graph needs to be updated dynamically as the program proceeds new matrix must be created for each addition or deletion

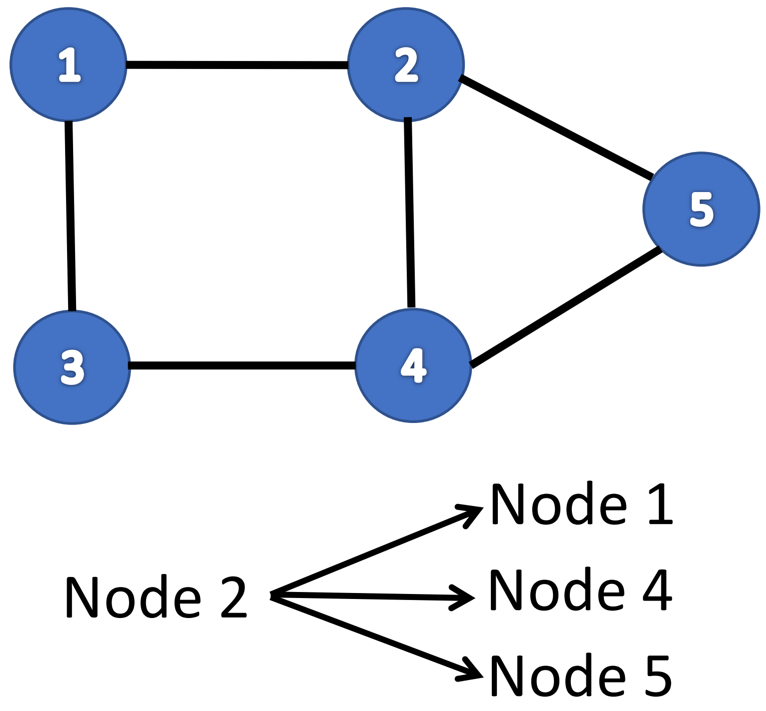
- To detect presence of edge between pair of nodes takes constant time O(1) but it takes $O(v^2)$ to visit all the neighbouring nodes of each node.

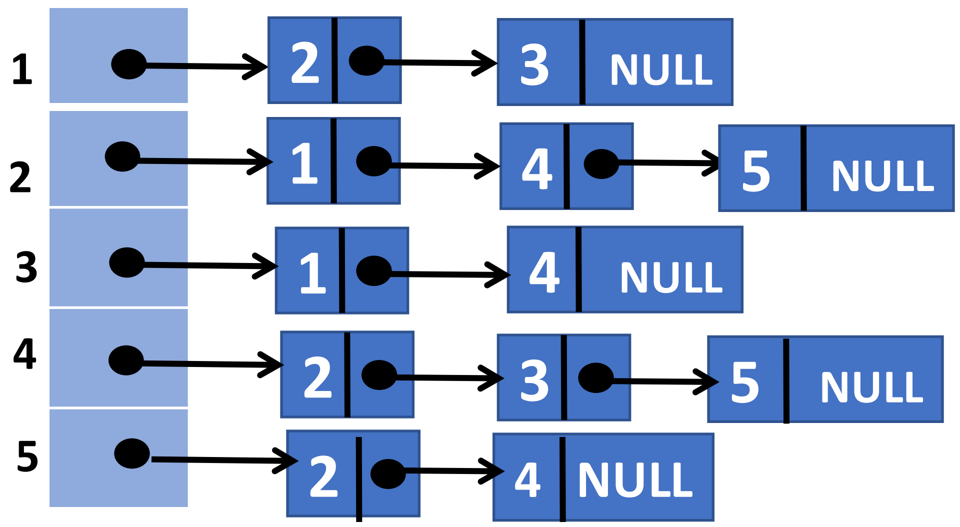
- Adjacency matrix becomes sparse in case graph has very few edges.

- space complexity is $O(v^2)$. v^2 locations are required for graph with v nodes

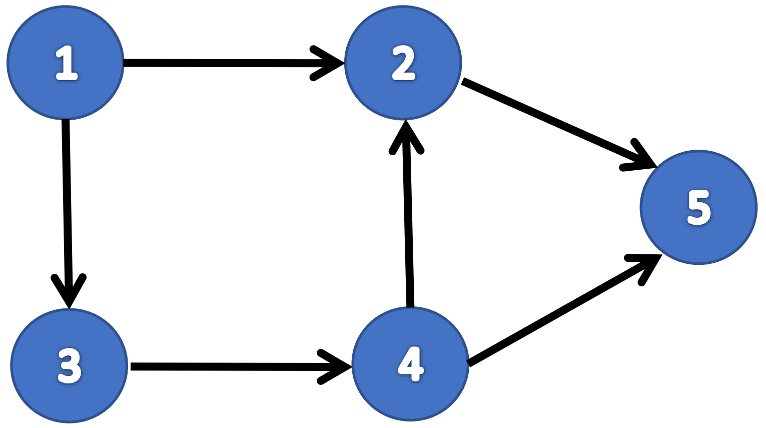
Adjacency list representation – Undirected graph

- Each node maintains
  - linked list of its neighbours

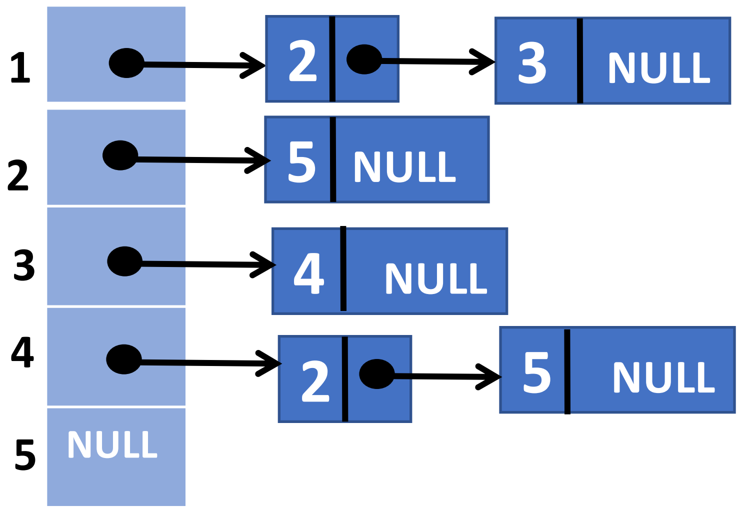




Adjacency list representation – Directed graph



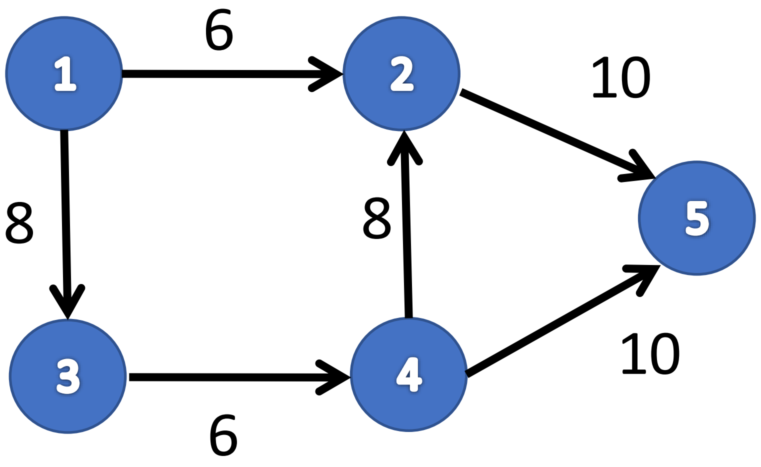
Directed Graph



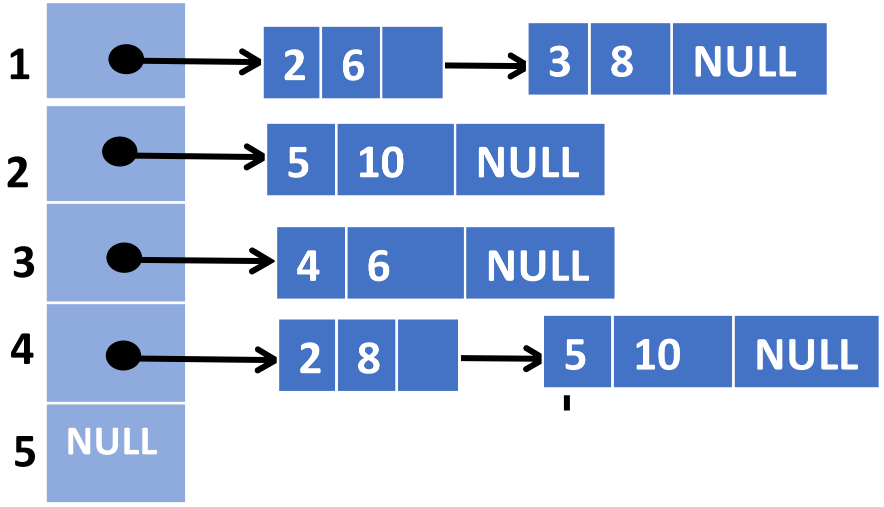
Node 2 → Node 5

Adjacency matrix representation – weighted graph

- In weighted graph the distance or cost between the nodes are represented on the edge
- Along with the information of adjacent node , cost/distance value is stored in each node of the linked list.



Weighted Graph



Pros and Cons of Adjacency List Representation

- Space Complexity of Adjacency list is O(V+E), because it stores the information of edges that actually exists in the graph
- In case of low density edges, the adjacency matrix becomes sparse using adjacency list is better for representation
- To detect the presence of edge between two nodes, we need to traverse the linked list of the node.

Implementation of graph using adjacency matrix

Data structure to represent the adjacency matrix:

- To represent only the existence of edge between nodes

```
#define maxnodes 50
adj[maxnodes][maxnodes];
```

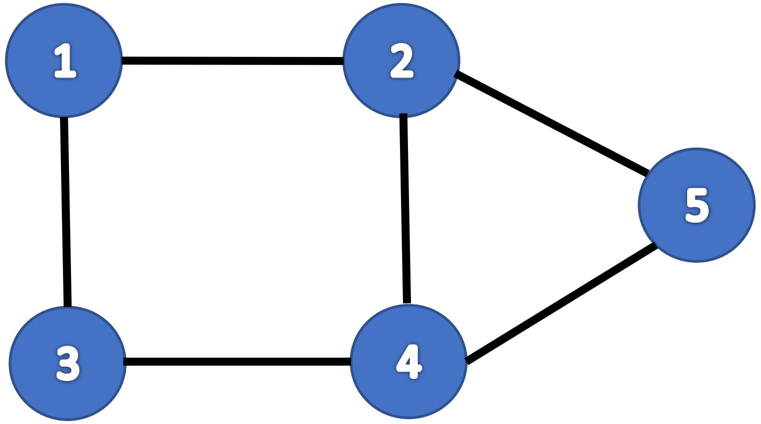
```
Typedef Boolean AdjecencyMatrix[maxnodes][maxnodes]

Typedef struct graph
{
    int n;     /* number of vertices in the graph */
    AdjacencyMatrix adj;
}Graph;
```

Implementation of Graph using Adjacency Matrix

To check the presence of edge between pair of nodes:

```
if ( adj[i][j] == 1)   */ i and j represents the vertex in a graph */
        then "edge exists between the pair of nodes"
else
           " there is  no edge between the pair of nodes"
```



$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Implementation of graph using adjacency matrix:

To add an edge from node1 to node2:

```
void join(int adj[][MAXNODES],int node1,int node2)
{
        adj[node1][node2]=TRUE;
}
```

To delete an edge from node1 to node2 if it exists:

```
void remv(int adj[][MAXNODES],int node1,int node2)
{
        adj[node1][node2]=FALSE;
}
```

Implementation of adjacency matrix:

To check whether arc exists between node1 and node2:

```
int adjacent(int adj[][MAX],int node1,int node2)
{
        return((if(adj[node1][node2])==TRUE)?TRUE:FALSE);
}
```

C-Representation of graphs- Adjacency Matrix

```
#define MAXNODES 50
struct node
{
        //information associated with each node
};
struct arc
{
        int adj;// information associated with each edge
};
struct graph
{
        struct node nodes[MAXNODES];
        struct arc arcs[MAXNODES][MAXNODES];
}
struct graph g;
```

C representation of weighted graph

- Weighted graph with fixed number of nodes is declared as

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

To add an edge from node1 to node2:

```
void  joinwt(struct arc g[][maxnodes],int node1,int node2, int wt)
{
  g[node1][node2].adj = TRUE;
  g[node1][node2].weight= wt;
}
```

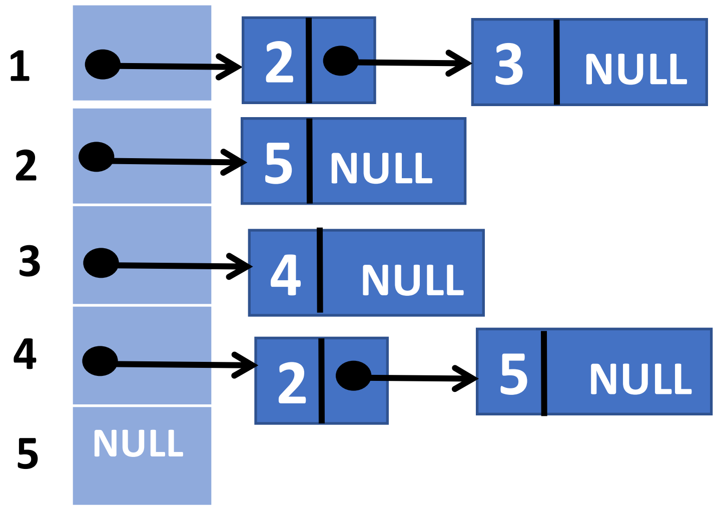
Implementation of adjacency List Representation:

```
struct node
{
   int node;
   struct node *nextedge;
};
```

NODE info	Nextedge

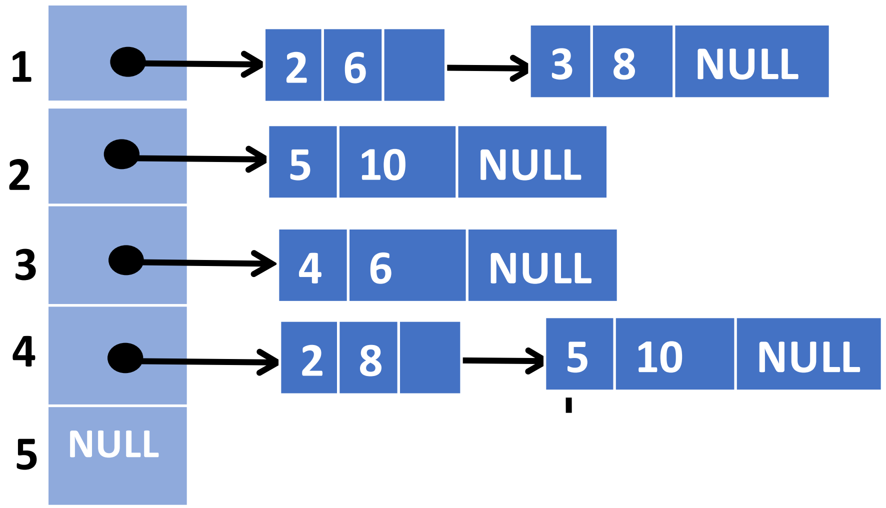
To create a node list:

```
 struct node  *nodelist;
 for(i=0; i < no_of_nodes; i++)
    nodelist[i]= NULL;
```

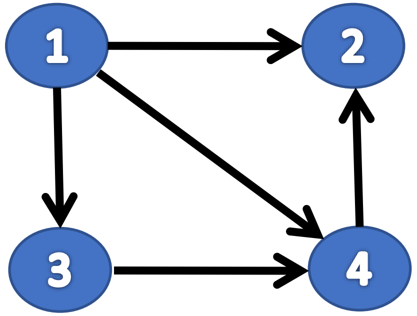



```
struct node
{
   int node;
   int weight;
   struct node *nextedge;
};
```

nodeinfo	weight	nextedge



Multilinked structure representation

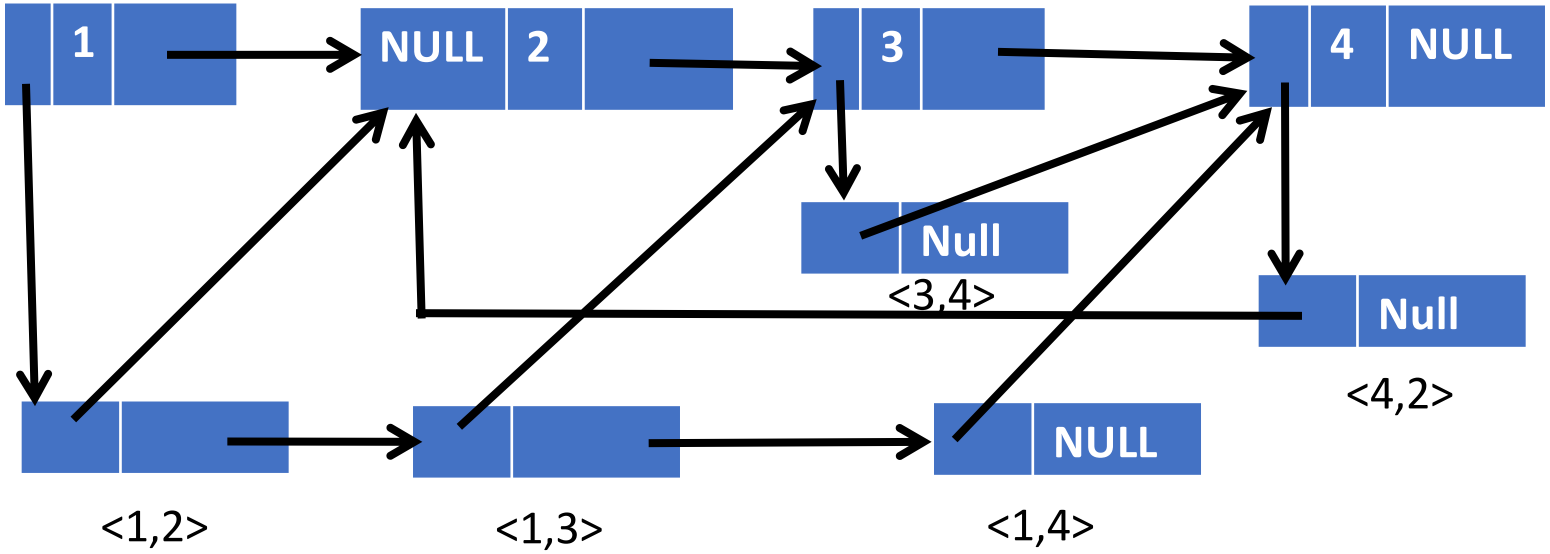


edgeptr	Info	nextnode

Sample header node representing the graph node

nodeptr	nextedge

A sample list node representing edge



Multilinked structure representation

Dynamic Implementation:

```
struct nodetype
{
  int info;
  struct nodetype  *ptr;
  struct node type *next;
};
struct nodetype *nodeptr;
```

edgeptr	Info	nextnode

Sample header node representing the graph node

nodeptr	nextedge

A sample list node representing edge

- Note that header node and list node have different formats and must be represented by different structures.
- In case of weighted graph in which each list node contains info field to store the weight of the edge the same dynamic implementation could be used for both types of nodes.

THANK YOU

Sandesh B. J
Department of Computer Science & Engineering

sandesh_bj@pes.edu

+91 80 6618 6623