Department of Computer science and Engineering

PES UNIVERSITY

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

GRAPHS

Abstract

Implementation of Graph using Adjacency List.

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

Implementation of adjacency List:

C representation of adjacency list

1.using array implementation

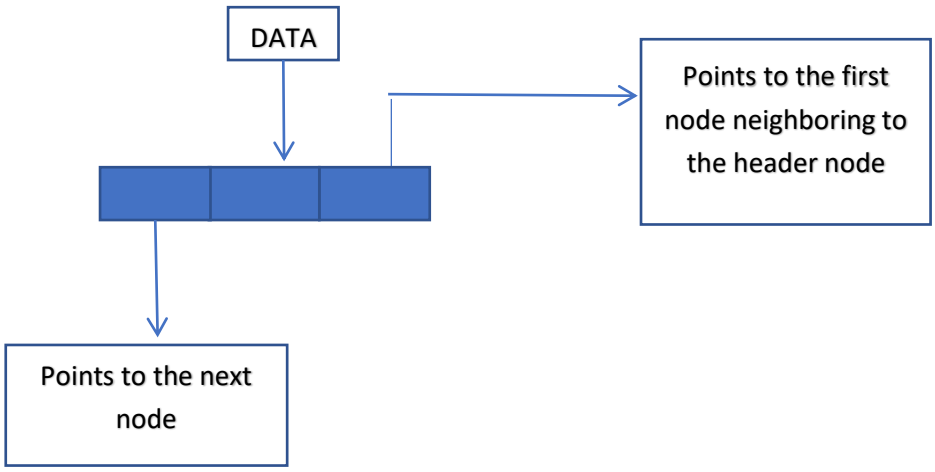
```
#define MAX 50
struct node
{
  int info;
  int point;
  int next;
};
struct node NODE[MAX];
```

2.Using dynamic implementation

```
struct node
{
   int info;
  strut node *pointer;
   struct node *next;
};
struct node *nodeptr;
```

Two lists are maintained for adjacency of list. The first list is used to store information of all the nodes. The second list is used to store the information of adjacent nodes for each node of a graph.

Header node is used to represent each list, which is assumed to be the first node of a list as shown below.



The different operations performed on adjacency list are

1. To create an arc between two nodes

```
void jointwt(int p,int q,int wt)
    {
        int r,r2;
        r2=-1;
       r=node[p].point;
       While(r>=0 && node[r].point!=q)
        {
            r2=r;
            r=node[r].next;
         }
       If(r>=0)
        {
            node[r].info=wt;
```

```
                    return;
                }
            R=getnode();
            node[r].point=q;
            node[r].next=-1;
             node[r].info=wt;
             (R2<0)?(node[p].point=r):(node[r2].next=r);
}
```

2. To remove an arc between two nodes if it exists:

```
Void remv(int p,int q)
{
    int r,r2;
   r2=-1;
   r=node[p].point;
  while(r>=0 ** node[r].point !=q)
 {
     r2=r;
    r=node[r].next;
}
If(r>=0)
{
   (r2<0)?(node[p].point=node[r].next):(node[r2].next=node[r].next);
   freenode(r);
   return;
}
```

```
}
```

3. Checks whether a node is adjacent to another node

```
int adjacent(int p,int q)
{
   int r;
    r=node[p].point;
   while(r>=0)
        If(node[r].point==q)
            return(TRUE)
         else
            r=node[r].next;
              return(FALSE);
}
```

4. Find a node with a specific information in a graph:

```
int findnode(int graph,int x)
{
        int p;
        p=graph;
        while(p>=0)
                if (node[p].info==x)
                        return(p);
                else
                        p=node[p].next;
```

```
        return(-1);
}
```

5. Add a node with specific information to a graph.

```
int addnode(int *graph,int x)
{
    int p;
    p=getnode();
    node[p].info=x;
    node[p].point=-1;
    node[p].next=*graph;
    *graph=p;
    return(p);
}
```