SANGOLA COLLEGE, SANGOLA
Class-B.Sc(ECS)-II, SEM-IV 2024-25
Practical Assignments
Sub- Data Structure using C++-II

Assignment No- 4

1) Write a program to represent directed graph using adjacency matrix.

```
#include<iostream.h>
#include<conio.h>
int adj[50][50];

void main()
{
        int i, j, s, d, v, e;
        clrscr();

        cout<<"Enter number of vertices : ";
        cin>>v;
        cout<<"Enter number of edges : ";
        cin>>e;

        for(i = 1; i <= e; i++)
        {
                cout<<"\nEnter "<<i<<" edge : ";

                cout<<"\nEnter source vertex : ";
                cin>>s;
                cout<<"Enter destination vertex : ";
                cin>>d;

                if(s > 0 || d > 0 || s <= v || d <= v)
                {
                        adj[s][d] = 1;
                }
                else
                {
                        cout<<"\nInvalid edge. Please enter again...";
                        i--;
                }
        }
```

```
        cout<<"\nAdjacency Matrix of Directed Graph : \n";
        for(i = 1; i <= v; i++)
        {
                for(j = 1; j <= v; j++)
                {
                        cout<<"\t"<<adj[i][j];
                }
                cout<<"\n";
        }
        getch();
}
```

2) Write a program to represent undirected graph using adjacency matrix.

```
#include<iostream.h>
#include<conio.h>
int adj[50][50];

void main()
{
        int i, j, s, d, v, e;
        clrscr();

        cout<<"Enter number of vertices : ";
        cin>>v;
        cout<<"Enter number of edges : ";
        cin>>e;

        for(i = 1; i <= e; i++)
        {
                cout<<"\nEnter "<<i<<" edge : ";
                cout<<"\nEnter source vertex : ";
                cin>>s;
                cout<<"Enter destination vertex : ";
                cin>>d;

                if(s > 0 || d > 0 || s <= v || d <= v)
                {
                        adj[s][d] = 1;
                        adj[d][s] = 1;
                }
                else
```

```
		{
			cout<<"\nInvalid edge. Please enter again...";
			i--;
		}
	}
	cout<<"\nAdjacency Matrix of Un-Directed Graph : \n";
	for(i = 1; i <= v; i++)
	{
		for(j = 1; j <= v; j++)
		{
			cout<<"\t"<<adj[i][j];
		}
		cout<<"\n";
	}
	getch();
}
```

3) Write a program to represent weighted directed graph using adjacency matrix.

```
#include<iostream.h>
#include<conio.h>
int adj[50][50];


void main()
{
	int i, j, s, d, v, e, wt;
	clrscr();

	cout<<"Enter number of vertices : ";
	cin>>v;
	cout<<"Enter number of edges : ";
	cin>>e;

	for(i = 1; i <= e; i++)
	{
		cout<<"\nEnter "<<i<<" edge : ";
		cout<<"\nEnter source vertex : ";
		cin>>s;
		cout<<"Enter destination vertex : ";
		cin>>d;
		cout<<"Enter weight of edge : ";
```

```
                cin>>wt;

                if(s > 0 || d > 0 || s <= v || d <= v){
                        adj[s][d] = wt;
                }
                else{
                        cout<<"\nInvalid edge. Please enter again...";
                        i--;
                }
        }
        cout<<"\nAdjacency Matrix of Weighted Graph : \n";
        for(i = 1; i <= v; i++)
        {
                for(j = 1; j <= v; j++)
                {
                        cout<<"\t"<<adj[i][j];
                }
                cout<<"\n";
        }
        getch();
}
```

4) Write a program to implement graph using linked list with all basic operations.

```
#include<iostream.h>
#include<conio.h>

class edge
{
   public:
     int dest;
     edge *right;
};

class node
{
   node *next;
   int info;
   edge *adj;
   public:
     void ins_vert(int);
     void disp_ver();
```

```
        void search(int);
        void ins_edge(int, int);
        void find_adj(int);
        void disp_graph();
}*start;

void node :: ins_vert(int x)
{
   node *p = new node;

   p->next = NULL;
   p->info = x;
   p->adj = NULL;

   if(start == NULL)
   {
      start = p;
   }
   else
   {
      node *temp = start;

      while(temp->next != NULL)
      {
         temp =  temp->next;
      }
      temp->next = p;
   }
   cout << "\nVertex " << x << " inserted successfully.";
}

void node :: disp_ver()
{
   node *p =start;

   cout << "\nVertices in graph: ";
   while(p != NULL)
   {
      cout<<p->info<<"\t";
      p = p->next;
   }
}

void node :: search(int srch)
{
```

```
    node *p = start;
    int t = 0;

    while(p != NULL)
    {
        if(p->info == srch)
        {
            t = 1;
            break;
        }
        p = p->next;
    }
    if(t == 1)
    {
        cout << "\nVertex " << srch << " found.";
    }
    else
    {
        cout << "\nVertex " << srch << " NOT found.";
    }
}

void node :: ins_edge(int s, int d)
{
    node *p = start, *q = start;
    int a = 0, b = 0;

    while(p != NULL)
    {
        if(p->info == s)
        {
            a = 1;
            break;
        }
        p = p->next;
    }
    while(q != NULL)
    {
        if(q->info == d)
        {
            b = 1;
            break;
        }
        q = q->next;
    }
```

```
    if(a == 1 && b == 1)
    {
       edge *e = new edge;

       e->dest = d;
       e->right = NULL;
       if(p->adj == NULL)
       {
          p->adj = e;
       }
       else
       {
            edge *temp = p->adj;

          while(temp->right != NULL)
          {
             temp = temp->right;
          }
          temp->right = e;
       }
          cout << "\nEdge (" << s << " --> " << d << ") inserted successfully.";
    }
    else
    {
       cout << "\nInvalid edge!";
    }
}

void node :: find_adj(int x)
{
    node *p = start;
    int a = 0;

    while(p != NULL)
    {
       if(p->info == x)
       {
          a = 1;
          break;
       }
       p = p->next;
    }
    if(a == 1)
    {
       if(p->adj == NULL)
```

```
      {
         cout << "\nNo adjacent vertices for " << x;
      }
      else
      {
         edge *temp = p->adj;
         cout << "\nAdjacent vertices of " << x << " : ";
         while(temp != NULL)
         {
            cout<<temp->dest<<"\t";
            temp = temp->right;
         }
      }
   }
   else
   {
      cout << "\nInvalid edge!";
   }
}

void node :: disp_graph()
{
   node *p = start;
   cout << "\nGraph representation:\n";
   while(p != NULL)
   {
      cout<<p->info;
      edge *temp = p->adj;

      while(temp != NULL)
      {
         cout<<" --> "<<temp->dest;
         temp = temp->right;
      }
      cout<<"\n";
      p = p->next;
   }
}

void main()
{
   int v, s, d, ch;
   node obj;
   clrscr();
```

```
   do{
      cout<<"\n\n--- MENU ---";
      cout<<"\n1. Insert Vertex\n2. Display Vertices\n3. Search Vertex\n4. Insert
Edge\n5. Find Adjacent Vertices\n6. Display Graph\n7. Exit";
      cout<<"\nEnter your choice: ";
      cin>>ch;

      switch(ch)
      {
         case 1:
            cout<<"\nEnter vertex: ";
            cin>>v;
            obj.ins_vert(v);
            break;

         case 2:
            obj.disp_ver();
            break;

         case 3:
            cout<<"\nEnter vertex to search: ";
            cin>>v;
            obj.search(v);
            break;

         case 4:
            cout<<"\nEnter source vertex: ";
            cin>>s;
            cout<<"Enter destination vertex: ";
            cin>>d;
            obj.ins_edge(s, d);
            break;
         case 5:
            cout<<"\nEnter vertex to find adjacent vertices: ";
            cin>>v;
            obj.find_adj(v);
            break;

         case 6:
            obj.disp_graph();
            break;

         case 7:
            cout<<"\nExiting program...";
            break;
```

```
        default:
           cout<<"\nInvalid choice!";
      }
   }while(ch != 7);
   getch();
}
```

5) Write a program to implement Breadth first search (BFS) traversal of graph.

```
#include <iostream.h>
#include <conio.h>

#define true 1
#define false 0
int adj[20][20];
int visited[20];
int v, e;
void bfs(int);

void main()
{
   int i, s, d, n;
   clrscr();

   cout<<"\nEnter number of vertices: ";
   cin>>v;
   cout<<"Enter number of edges: ";
   cin>>e;

   for(i = 1; i <= e; i++)
   {
      cout<<"\nEnter "<< i <<" Edge ";
      cout<<"\nEnter Source vertex: ";
      cin>>s;
      cout<<"Enter Destination vertex: ";
      cin>>d;

      if(s > v || d > v || s <= 0 || d <= 0)
      {
         cout<<"\nIn-Valid Edge.";
         i--;
      }
```

```
      else
      {
         adj[s][d] = 1;
      }
   }
   cout<<"\nEnter traversing vertex: ";
   cin>>n;
   bfs(n);
   getch();
}

void bfs(int n)
{
   cout<<"\nBFS: ";
   int queue[20], front, rear, i;
   front = rear = -1;
   ++rear;
   queue[rear] = n;

   while (front != rear)
   {
      ++front;
      n = queue[front];
      if (visited[n] == false)
      {
         cout<<"\t"<<n;
         visited[n] = true;
         for (i = 1; i <= v; i++)
         {
            if (adj[n][i] == 1 && visited[i] == false)
            {
               ++rear;
               queue[rear] = i;
            }
         }
      }
   }
}
```

6) Write a program to implement Depth first search (DFS) traversal of graph.

```
#include <iostream.h>
#include <conio.h>

#define true 1
#define false 0
int adj[20][20];
int visited[20];
int v, e;
void dfs(int);

void main()
{
   int i, s, d, n;
   clrscr();

   cout<<"\nEnter number of vertices: ";
   cin>>v;
   cout<<"Enter number of edges: ";
   cin>>e;

   for(i = 1; i <= e; i++)
   {
      cout<<"\nEnter "<< i <<" Edge ";
      cout<<"\nEnter Source vertex: ";
      cin>>s;
      cout<<"Enter Destination vertex: ";
      cin>>d;
      if(s > v || d > v || s <= 0 || d <= 0)
      {
         cout<<"\nIn-Valid Edge.";
         i--;
      }
      else
      {
         adj[s][d] = 1;
      }
   }
   cout<<"\nEnter traversing vertex: ";
   cin>>n;
   dfs(n);
   getch();
}
```

```
void dfs(int n)
{
   cout<<"\nDFS: ";
   int stack[20], top, i;
   top = -1;
   ++top;
   stack[top] = n;

   while (top != -1)
   {
      n = stack[top--];
      if (visited[n] == false)
      {
         cout<<"\t"<<n;
         visited[n] = true;
         for (i = 1; i <= v; i++)
         {
            if (adj[n][i] == 1 && visited[i] == false)
            {
               ++top;
               stack[top] = i;
            }
         }
      }
   }
}
```
