**Jadavpur University**

**Department of Computer Science and Engineering**

**M.Tech in Computer Technology**

**1st Year, 1st Semester**

**Programming Lab**

**Assignment-6**

For this assignment consider graphs are stored as an adjacency matrix

1. Write a function void addVertex(int n) that adds a vertex with name n to the graph. If there is already a vertex with name n, then the function should do nothing. Otherwise the new vertex should be made the last vertex in the vertex list of the graph.

2. Write a function void addEdge (int u int v) that does the following. The function should add a new edge from the vertex with name u to vertex with name v to the graph. If there is no vertex named u or no vertex named v, then the function should do nothing. If there is already an edge between u and v, the function should not do anything

3. Write a function void delEdge(int u, int v) that does the following. The function should remove the edge from vertex with name u to vertex with name v from the graph. If there is no such edge in the graph, then the function should do nothing.

4. Write a function void delVertex(int u) that does the following. The function should remove the vertex named u and all edges that either come into u or go out of u. If there is no vertex with name u, then the function should do nothing.

5. Write a program to find approachable nodes from a given source of a given graph using queue as an intermediate data structure (BFS).

6. Write a program to traverse various nodes of a given graph using stack as an intermediate data structure (DFS).

7. Write a program to find shortest path from a given source to all the approachable nodes (Single source shortest path Dijkstra's algorithm).

8. Write a program to find shortest path between all the source destination pairs (All pairs shortest path Floyd's algorithm.

9. Write a program to arrange all the nodes of a given graph (Topological sort).

10. Write a program to find Minimal spanning tree of a graph using Kruskal's algorithm.

11. Write a program to find Minimal spanning tree of a graph using Prim's algorithm.

**Program 1 & Program 2 & Program 3 & Program 4:-**

#include<stdio.h>

void addVertex(int nodeNumber, int n, int graph[][n]){

    if(nodeNumber == 0){

        graph[0][0] = 0;

    }

    else {

        int i;

        for(i = 0; i < nodeNumber; i++){

            graph[i][nodeNumber] = 0;

        }

        for(i = 0; i <= nodeNumber; i++){

            graph[nodeNumber][i] = 0;

        }

    }

}

void addEdge(int n, int graph[][n], int u, int v){

    if(u < n && v < n){

        graph[u][v] = 1;

        graph[v][u] = 1;

    }

}

void delEdge(int n, int graph[][n], int u, int v){

    if(u < n && v < n){

        graph[u][v] = 0;

        graph[v][u] = 0;

    }

}

void delVertex(int n, int graph[][n], int v, int \*n\_ptr){

    if(v < n){

        int i;

        for(i = 0; i < n; i++){

            graph[v][i] = 0;

            graph[i][v] = 0;

        }

        \*n\_ptr = \*n\_ptr - 1;

    }

}

int main(){

    int n, i, j;

    printf("Enter the number of nodes in the graph: ");

    scanf("%d",&n);

    int graph[100][100];

    for(i = 0; i < n; i++){

        addVertex(i, n, graph);

    }

    addEdge(n, graph, 0, 1);

    printf("Edge betwen %dth verted and %dth vertex has been added...\n",0,1);

    delEdge(n, graph, 0, 1);

    printf("Edge betwen %dth verted and %dth vertex has been deleted...\n",0,1);

    delVertex(n, graph, n - 1, &n);

    printf("vertex %d has been deleted...\n",n);

    for(i = 0; i < n; i++){

        for(j = 0; j < n; j++){

            printf("%d ",graph[i][j]);

        }

        printf("\n");

    }

    return 0;

}

**Program 5**:-

#include<stdio.h>

#include<stdlib.h>

struct queue{

    int front, rear;

    int \*list;

};

void bfs\_algo(int n, int graph[][n], int bfs[], struct queue queue){

    int i, ind = 0, node = 0, visited[n];

    queue.list[queue.rear] = node;

    queue.rear++;

    for(i = 0; i < n; i++){

        visited[i] = 0;

    }

    while(queue.front < queue.rear){

        node = queue.list[queue.front];

        queue.front++;

        visited[node] = 1;

        bfs[ind] = node;

        ind++;

        for(i = 0; i < n; i++){

            if(visited[i] == 0 && graph[node][i] == 1){

                queue.list[queue.rear] = i;

                queue.rear++;

            }

        }

    }

}

int main(){

    int n, i, j;

    printf("Enter the number of nodes: ");

    scanf("%d",&n);

    int val, graph[n][n], bfs[n];

    struct queue queue;

    queue.list = (int \*)malloc(sizeof(int) \* n \* n);

    queue.front = 0;

    queue.rear = 0;

    for(i = 0; i < n; i++){

        for(j = i; j < n; j++){

            printf("Enter the edge for %d<-->%d: ",i,j);

            scanf("%d",&val);

            graph[i][j] = val;

            graph[j][i] = val;

        }

    }

    bfs\_algo(n, graph, bfs, queue);

    printf("bfs of the graph is: \n");

    for(i = 0; i < n; i++){

        printf("%d ",bfs[i]);

    }

    return 0;

}

**Program 6**:-

#include<stdio.h>

void dfs\_stack(int n, int graph[][n], int dfs[]){

    int node, dfs\_ind = 0, i, top = 0, stack[2\*n], visited[n];

    for(i = 0; i < n; i++){

        visited[i] = 0;

    }

    stack[0] = 0;

    while(top >= 0){

        node = stack[top];

        top--;

        dfs[dfs\_ind] = node;

        dfs\_ind++;

        for(i = 0; i < n; i++){

            if(visited[i] == 0 && graph[node][i] == 1)

                stack[++top] = i;

        }

        visited[node] = 1;

    }

}

int main(){

    int n, i, j;

    printf("Enter the number of nodes: ");

    scanf("%d",&n);

    int val, graph[n][n], dfs[n];

    for(i = 0; i < n; i++){

        for(j = i; j < n; j++){

            printf("Enter the edge for %d<-->%d: ",i,j);

            scanf("%d",&val);

            graph[i][j] = val;

            graph[j][i] = val;

        }

    }

    dfs\_stack(n,graph,dfs);

    printf("dfs of the graph is: \n");

    for(i = 0; i < n; i++)

        printf("%d ",dfs[i]);

}

**Program 7**:-

#include<stdio.h>

#define INT\_MAX 1000009

void dijkstras\_algo(int n, int graph[][n], int visited[], int distance[]){

    int loop;

    for(loop = 0; loop < n; loop++){

        int i,ind = -1, min\_val = INT\_MAX;

        for(i = 0; i < n; i++){

            if(visited[i] == 0 && distance[i] < min\_val){

                min\_val = distance[i];

                ind = i;

            }

        }

        visited[ind] = 1;

        for(i = 0; i < n; i++){

            if(visited[i] == 0 && graph[ind][i] > 0){

                if(distance[ind] + graph[ind][i] < distance[i])

                    distance[i] = distance[ind] + graph[ind][i];

            }

        }

    }

}

int main(){

    int n,i,j;

    printf("Enter the number of nodes: ");

    scanf("%d",&n);

    int graph[n][n], visited[n],distance[n];

    for(i = 0; i < n; i++){

        for(j = 0; j < n; j++){

            printf("Enter the weight of the edge %d --> %d: ",i,j);

            scanf("%d",&graph[i][j]);

        }

    }

    for(i = 0; i < n; i++){

        visited[i] = 0;

        distance[i] = INT\_MAX;

    }

    distance[0] = 0;

    dijkstras\_algo(n, graph, visited, distance);

    printf("The shortest distance to each node is(starting from 0): ");

    for(i = 0; i < n; i++){

        printf("%d ",distance[i]);

    }

    return 0;

}

**Program 8**:-

#include<stdio.h>

#define INT\_MAX 1000009

void floyd\_algo(int n, int srt\_paths[][n]){

    int i, j, k;

    for(i = 0; i < n; i++){

        for(j = 0; j < n; j++){

            for(k = 0; k < n; k++){

                if(j == k || j == i || k == i)

                    continue;

                if(srt\_paths[j][k] > srt\_paths[j][i] + srt\_paths[i][k])

                    srt\_paths[j][k] = srt\_paths[j][i] + srt\_paths[i][k];

            }

        }

    }

}

int main(){

    int n, i, j;

    printf("Enter the numebr of nodes: ");

    scanf("%d",&n);

    int val, graph[n][n], srt\_paths[n][n];

    for(i = 0; i < n; i++){

        for(j = 0; j < n; j++){

            printf("Enter the edge for %d --> %d : ",i,j);

            scanf("%d",&val);

            if(val == -1){

                graph[i][j] = INT\_MAX;

                srt\_paths[i][j] = INT\_MAX;

            }

            else{

                graph[i][j] = val;

                srt\_paths[i][j] = val;

            }

        }

    }

    floyd\_algo(n, srt\_paths);

    printf("The paths are: \n");

    for(i = 0; i < n; i++){

        for(j = 0; j < n; j++){

            printf("%d ",srt\_paths[i][j]);

        }

        printf("\n");

    }

}

**Program 9**:-

#include<stdio.h>

void topological\_sort(int n,int graph[][n],int visited[],int stack[],int node,int \*top){

    visited[node] = 1;

    for(int i=0;i < n;i++){

        if(graph[node][i] == 1 && visited[i] == 0)

            topological\_sort(n,graph,visited,stack,i,top);

    }

    stack[\*top] = node;

    \*top = \*top + 1;

}

int main(){

    int n,i,j,val;

    printf("Enter the number of node: ");

    scanf("%d",&n);

    int graph[n][n],stack[n],visited[n],top=0;

    for(i = 0;i < n;i++){

        for(j = 0;j< n;j++){

            printf("enter the edge for %d and %d: ",i,j);

            scanf("%d",&val);

            graph[i][j] = val;

            // graph[j][i] = val;

        }

    }

    for(i = 0;i < n;i++){

        visited[i] = 0;

    }

    topological\_sort(n,graph,visited,stack,0,&top);

    for(i=n-1;i>=0;i--){

        printf("%d ",stack[i]);

    }

}

**Program 10**:-

#include<stdio.h>

void insert\_edge(int arr1[],int arr2[]){

    for(int i=0;i<3;i++){

        arr1[i] = arr2[i];

    }

}

void right\_shift(int edges[][3],int start,int end){

    int i;

    for(i=start;i > end; i--){

        insert\_edge(edges[i],edges[i-1]);

    }

}

int getParent(int parent[], int node){

    if(parent[node] == node)

        return node;

    parent[node] = getParent(parent,parent[node]);

    return parent[node];

}

void union\_node(int parent[],int rank[],int node1,int node2){

    int parent\_node1 = getParent(parent,node1);

    int parent\_node2 = getParent(parent,node2);

    if(rank[parent\_node1] > rank[parent\_node2])

        parent[parent\_node2] = parent\_node1;

    else if(rank[parent\_node2] > rank[parent\_node1])

        parent[parent\_node1] = parent\_node2;

    else {

        parent[parent\_node2] = parent\_node1;

        rank[parent\_node1]++;

    }

}

int mst(int n, int graph[][n], int parent[], int rank[]){

    int i,j,k,edges[n\*n][3],last\_index = 0;

    for(i=0;i<n;i++){

        for(j=i + 1;j<n;j++){

            if(graph[i][j] != 0){

                int arr[3] = {i,j,graph[i][j]};

                if(last\_index == 0){

                    insert\_edge(edges[last\_index],arr);

                    last\_index++;

                } else {

                    if(edges[last\_index-1][2] < graph[i][j]){

                        insert\_edge(edges[last\_index],arr);

                        last\_index++;

                    } else {

                        for(k=0;k<last\_index;k++){

                            if(graph[i][j] < edges[k][2]){

                                right\_shift(edges,last\_index,k);

                                last\_index++;

                                insert\_edge(edges[k],arr);

                                break;

                            }

                        }

                    }

                }

            }

        }

    }

    int weight = 0;

    for(i = 0; i < last\_index; i++){

        int parent1 = getParent(parent,edges[i][0]);

        int parent2 = getParent(parent,edges[i][1]);

        if(parent1 != parent2){

            union\_node(parent, rank, edges[i][0], edges[i][1]);

            weight += edges[i][2];

        }

    }

    return weight;

}

int main(){

    int n;

    printf("Enter the number of nodes in the graph: ");

    scanf("%d",&n);

    int i,j,graph[n][n],val;

    for(i=0;i<n;i++){

        for(j=i;j<n;j++){

            printf("Enter the weight for the edge %d<-->%d: ",i,j);

            scanf("%d",&val);

            graph[i][j] = val;

            graph[j][i] = val;

        }

    }

    int parent[n],rank[n];

    for(i=0;i<n;i++){

        parent[i] = i;

        rank[i] = 0;

    }

    int mst\_weight = mst(n,graph, parent, rank);

    printf("The weight of the minimum spanning tree of this graph is: %d",mst\_weight);

    return 0;

}

**Program 11**:-

#include<stdio.h>

#define INT\_MAX 1000009

int prims\_mst(int n, int graph[][n], int key[], int mst[], int parent[]){

    int j, sum = 0;

    for(j = 0; j < n; j++){

        int i, u = -1, min\_val = INT\_MAX;

        for(i = 0; i < n; i++){

            if(mst[i] == 0 && key[i] < min\_val){

                min\_val = key[i];

                u = i;

            }

        }

        mst[u] = 1;

        for(i = 0; i < n; i++){

            if(mst[i] == 0 && graph[u][i] != 0 && key[i] > graph[u][i]){

                key[i] = graph[u][i];

                parent[i] = u;

            }

        }

    }

    for(j = 0; j < n; j++)

        sum += key[j];

    return sum;

}

int main(){

    int n;

    printf("Enter the number of nodes in the graph: ");

    scanf("%d",&n);

    int i,j,graph[n][n],val;

    for(i=0;i<n;i++){

        for(j=i;j<n;j++){

            printf("Enter the weight for the edge %d<-->%d: ",i,j);

            scanf("%d",&val);

            graph[i][j] = val;

            graph[j][i] = val;

        }

    }

    int key[n],mst[n],parent[n];

    for(i = 0; i < n; i++){

        key[i] = INT\_MAX;

        mst[i] = 0;

        parent[i] = -1;

    }

    key[0] = 0;

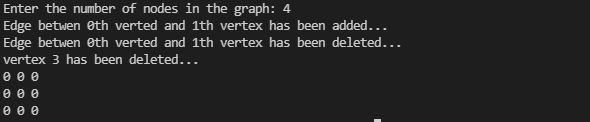
    int min\_weight = prims\_mst(n,graph,key,mst,parent);

    printf("The weight of the minimum spanning tree of this graph is: %d",min\_weight);

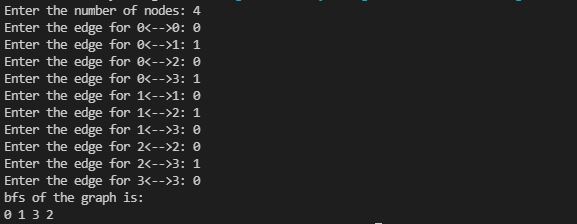
    return 0;

}

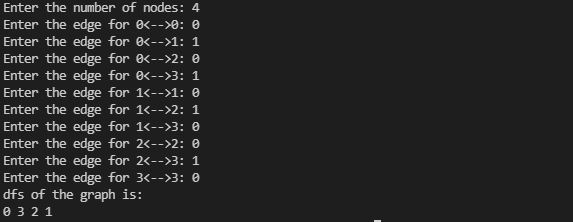
**Output 1 & Output 2 & Output 3 & Output 4**:-



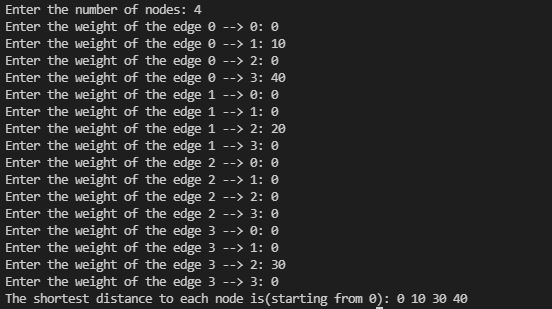
**Output 5**:-



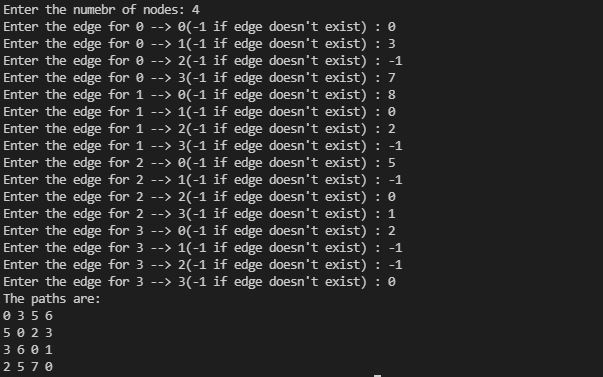
**Output 6**:-



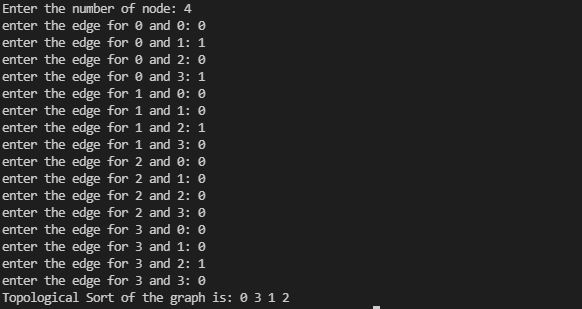
**Output 7**:-



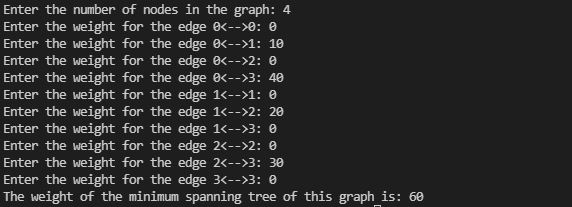
**Output 8**:-



**Output 9**:-



**Output 10**:-



**Output 11**:-

