Experiment 24 **Date: 01/01/2025**

Implement BFS and DFS on a connected graph

Aim:

Write a program to implement BFS and DFS on a connected undirected graph.

Algorithm:

Declare stack, queue and global variables

1 Declare q[20], stack[20], a[20][20], vis[20]
2 Declare top = -1, front = -1, rear = -1

main()

1 Start
2 Declare n, i, s, ch, j, c, dummy.
3 Read the number of vertices.
4 Read adjacency matrix.
5 For i=1 to n
Set vis[i]=0
6 Display choices
7 Read the source vertex
8 Read option choice
If ch==1 call bfs(s, n)
If ch==2 dfs(s, n)
9 Read whether the user want to continue or not
10 Repeat steps 5 to 9 while (((c == 'y') || (c == 'Y')))
11 Stop

void bfs(int s, int n)

1 Start.
2 Declare p, i
3 Call enqueue(s)
4 Set vis[s] = 1
5 Call p = dequeue()
6 If (p != 0)
Display p
7 While (p != 0)
For i=1 to n
if ((a[p][i] != 0) && (vis[i] == 0))
call enqueue(i)
set vis[i] = 1
Call p = dequeue()
If (p != 0)
Display p
8 For i=1 to n
If (vis[i] == 0)
Call bfs(i, n)
9 Exit.

void enqueue(int item)

```
1  Start.
2  If ((rear == 19))
       Display “Queue is full”
3  Else
       If (rear == -1)
          Set q[++rear] = item
          Set front=front+1
       Else
          Set q[++rear] = item
4  Exit.
```

int dequeue()

```
1  Start.
2  Declare k
3  If ((front > rear) || (front == -1))
       Return k
4  Else
       Set k = q[front++]
       Return k
5  Exit.
```

void dfs(int s, int n)

```
1  Start.
2  Declare k, i
3  Call push(s)
4  Set vis[s] = 1
5  Call k = pop()
6  If (k != 0)
       Display k
7  While (k != 0)
       For i=1 to n
         if ((a[k][i] != 0) && (vis[i] == 0))
           call push(i)
           set vis[i] = 1
       Call k = pop()
       If (k != 0)
       Display k
8  For i=1 to n
       If (vis[i] == 0)
          Call dfs(i, n)
9  Exit.
```

void push(int item)

```
1  Start.
2  If ((top == 19))
       Display “stack is full”
3  Else
```

Set stack[++top] = item
4 Exit.

int pop()

1 Start.
2 Declare k
3 If (top == -1)
Go to step 5
4 Else
Set k = stack[top--]
Return k
5 Exit.

Program

```
#include <stdio.h>
#include <stdlib.h>
int q[20], top = -1, front = -1, rear = -1, a[20][20], vis[20], stack[20];
int dequeue();
void enqueue(int item);
void bfs(int s, int n);
void dfs(int s, int n);
void push(int item);
int pop();
int main() {
   int n, i, s, ch, j;
   char c, dummy;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         printf("Enter 1 if %d has an edge with %d else 0: ", i, j);
         scanf("%d", &a[i][j]);
      }}
   printf("The Adjacency Matrix is\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         printf(" %d", a[i][j]);
      }
      printf("\n");
   }
   do {
      for (i = 1; i <= n; i++)
         vis[i] = 0;
      printf("\nMENU");
      printf("\n1. B.F.S");
      printf("\n2. D.F.S");
      printf("\nEnter your choice: ");
      scanf("%d", &ch);
```

```
            printf("Enter the source vertex: ");
            scanf("%d", &s);
            switch (ch) {
               case 1:
                  bfs(s, n);
                  break;
               case 2:
                  dfs(s, n);
                  break;
            }
            printf("Do you want to continue (Y/N)? ");
            scanf(" %c", &c);
         } while ((c == 'y') || (c == 'Y'));
         return 0;
      }
      void bfs(int s, int n) {
         int p, i;
         enqueue(s);
         vis[s] = 1;
         p = dequeue();
         if (p != 0)
            printf(" %d", p);
         while (p != 0) {
            for (i = 1; i <= n; i++)
               if ((a[p][i] != 0) && (vis[i] == 0)) {
                  enqueue(i);
                  vis[i] = 1;
               }
            p = dequeue();
            if (p != 0)
               printf(" %d ", p);
         }
         for (i = 1; i <= n; i++)
            if (vis[i] == 0)
               bfs(i, n);
      }
      void enqueue(int item) {
         if (rear == 19)
            printf("QUEUE FULL");
         else {
            if (rear == -1) {
               q[++rear] = item;
               front++;
            } else
               q[++rear] = item;
         }}
      int dequeue() {
         int k;
         if ((front > rear) || (front == -1))
```

```
        return 0;
    else {
        k = q[front++];
        return k;
    }}
void dfs(int s, int n) {
    int i, k;
    push(s);
    vis[s] = 1;
    k = pop();
    if (k != 0)
        printf(" %d ", k);
    while (k != 0) {
        for (i = 1; i <= n; i++)
            if ((a[k][i] != 0) && (vis[i] == 0)) {
                push(i);
                vis[i] = 1;
            }
        k = pop();
        if (k != 0)
            printf(" %d ", k);
    }
    for (i = 1; i <= n; i++)
        if (vis[i] == 0)
            dfs(i, n);
}
void push(int item) {
    if (top == 19)
        printf("Stack overflow ");
    else
        stack[++top] = item;
}
int pop() {
    int k;
    if (top == -1)
        return 0;
    else {
        k = stack[top--];
        return k;
    }}
```

Output

```
Enter the number of vertices: 4
Enter 1 if 1 has an edge with 1 else 0: 1
Enter 1 if 1 has an edge with 2 else 0: 1
Enter 1 if 1 has an edge with 3 else 0: 1
Enter 1 if 1 has an edge with 4 else 0: 1
Enter 1 if 2 has an edge with 1 else 0: 0
```

```
Enter 1 if 2 has an edge with 2 else 0: 0
Enter 1 if 2 has an edge with 3 else 0: 1
Enter 1 if 2 has an edge with 4 else 0: 1
Enter 1 if 3 has an edge with 1 else 0: 0
Enter 1 if 3 has an edge with 2 else 0: 0
Enter 1 if 3 has an edge with 3 else 0: 1
Enter 1 if 3 has an edge with 4 else 0: 0
Enter 1 if 4 has an edge with 1 else 0: 0
Enter 1 if 4 has an edge with 2 else 0: 1
Enter 1 if 4 has an edge with 3 else 0: 1
Enter 1 if 4 has an edge with 4 else 0: 0

The Adjacency Matrix is
 1 1 1 1
 0 0 1 1
 0 0 1 0
 0 1 1 0

MENU
1. B.F.S
2. D.F.S
Enter your choice: 1
Enter the source vertex: 3
3 1 2 4
Do you want to continue (Y/N)? y

MENU
1. B.F.S
2. D.F.S
Enter your choice: 2
Enter the source vertex: 1
1 4 3 2
Do you want to continue (Y/N)? n
```

Experiment 25 **Date: 06/01/2025**

Implement Prim's Algorithm for finding the MCST

Aim:

Program to implement Prim's Algorithm for finding the minimum cost spanning tree.

Algorithm:

main()

1. b1.Read n
2. int i=1,i<=n
3. for(int j=1;j<=n;j++)
4. read cost[i][j]
5. cost[i][j]==0
6. cost[i][j]=INF
7. visited[1]=1
8. while (no_edges<n-1)
9. min=INF
10. a=0
11. b=0
12. i=1,i<=n
13. if(visited[i]==1)
14. j=1,j<=n
15. if (visited[j]==0 && cost[i][j]!=INF)
16. if (cost[i][j]<min)
17. min=cost[i][j];
18. a=i
19. b=j
20. no_edges
21. visited[b]=1
22. total_cost=total_cost+min
23. Print total cost
24. exit

Program

```
#include<stdio.h>
#define INF 999
int cost[10][10],visited[10]={0,0,0,0,0,0,0,0,0,0};
int n,i,j,no_edges=0,total_cost=0,min,a,b;
int main()
{
printf("Enter the number of vertices : ");
scanf("%d",&n);
printf("Enter the cost adjacency matrix : \n");
for(int i=1;i<=n;i++)
   {
```

```
for(int j=1;j<=n;j++)
      {
scanf("%d",&cost[i][j]);
         if (cost[i][j]==0)
         {
            cost[i][j]=INF;
         }}}
printf("\nThe minimum cost spanning tree edges are:\n");
visited[1]=1;
   while (no_edges<n-1)
   {
      min=INF;
      a=0;
      b=0;
      for(i=1;i<=n;i++)
      {
         if(visited[i]==1)
         {
            for (j=1;j<=n;j++)
            {
               if (visited[j]==0 && cost[i][j]!=INF)
               {
                  if (cost[i][j]<min)
                  {
                     min=cost[i][j];
                     a=i;
                     b=j;
                  }}}}}
no_edges++;
      visited[b]=1;
printf("%d-%d:%d\n",a,b,min);
total_cost=total_cost+min;
   }
printf("Total cost : %d\n",total_cost);
}
```

Output

```
Enter the number of vertices : 3
Enter the cost adjacency matrix :
1
0
0
0
1
1
1
1
1
```

The minimum cost spanning tree edges are:
0-0:999
0-0:999
Total cost : 1998

Experiment 26 **Date: 06/01/2025**

Implement Kruskal's Algorithm using Disjoint sets for finding MCST

Aim:

Program to implement Kruskal's algorithm using Disjoint sets.

Algorithm:

main()

1. Start
2. Declare cost[10][10], visited[10]={0,0,0,0,0,0,0,0,0,0}, parent[10]
3. Declare n,i,j,no_edges=0,total_cost=0,min,a,b,u,v
4. Read n
5. Set i=1, i<=n
6. Set j=1,j<=n
7. Read cost[i][j]
8. if (cost[i][j] ==0)
9. cost[i][j] =INF
10. visited [1]=1
11. while (no_edges<n-1)
12. min=INF
13. a=0
14. b=0
15. Set i=1, i<=n
16. Set j=1, j<=n
17. if (cost[i][j]<min)
18. min=cost[i][j]
19. a=u=i
20. b=v=j
21. u=find(u)
22. v=find(v)
23. if(uni(u,v))
24. Print a,b,min
25. no_edges++
26. total_cost=total_cost+min
27. cost[a][b]=cost[b][a]=INF
28. Printtotal_cost
29. Stop

find(int)

1. Start
2. while(parent[i])
3. i=parent[i]
4. return i
5. Stop

unit(int ,int)

1. Start
2. if(i!=j)

3. parent[j]=i
4. return 1
5. return
6. Stop

Program

```
#include<stdio.h>
#define INF 999
int cost[10][10],visited[10]={0,0,0,0,0,0,0,0,0,0},parent[10];
int n,i,j,no_edges=0,total_cost=0,min,a,b,u,v;
int find(int);
int uni(int,int);
int main()
{
printf("Enter the no. of vertices : ");
scanf("%d",&n);
printf("Enter the adjacency matrix : \n");
for(int i=1;i<=n;i++)
   {
for(int j=1;j<=n;j++)
      {
scanf("%d",&cost[i][j]);
         if (cost[i][j]==0)
         {
            cost[i][j]=INF;
         }}}
printf("\nCost of edges\n");
visited[1]=1;
   while (no_edges<n-1)
   {
      min=INF;
      a=0;
      b=0;
      for(i=1;i<=n;i++)
      {
         for (j=1;j<=n;j++)
         {
            if (cost[i][j]<min)
            {
               min=cost[i][j];
               a=u=i;
               b=v=j;
            }}}
      u=find(u);
      v=find(v);
      if(uni(u,v))
      {
printf("%d-%d:%d\n",a,b,min);
```

```
no_edges++;
total_cost=total_cost+min;
        }
        cost[a][b]=cost[b][a]=INF;
    }
printf("Total cost : %d\n",total_cost);
}
int find(int i)
{
    while(parent[i])
i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
    }
    return 0;
}
```

Output

```
Enter the no. of vertices: 3
Enter the adjacency matrix:
1
0
0
1
1
1
0
0
1
Cost of edges
2-1:1
2-3:1
Total cost: 2
```

Experiment 27 **Date: 08/01/2025**

Implement Dijkstras Algorithm

Aim:

Program for single source shortest path algorithm using Dijkstras algorithm.

Algorithm:

minDistance(int , bool)

1. Start
2. Set v = 0, v < V
3. if (sptSet[v] == false &&dist[v] <= min)
4. min = dist[v], min_index = v
5. return min_index
6. Stop

printSolution(dist[])

1. Start
2. Set i = 0, i< V
3. Print i, dist[i]
4. Stop

dijkstra(graph[V][V],src)

1. Start
2. declaredist[V]
3. declare sptSet[V]
4. declare i = 0; i< V; i++)
5. Set dist[i] = INT_MAX, sptSet[i] = false
6. Set dist[src] = 0
7. Set count = 0, count < V - 1
8. Set u = minDistance(dist, sptSet)
9. Set sptSet[u] = true
10. Set v = 0, v< V
11. if!sptSet[v] && graph[u][v]
12. Set dist[u] != INT_MAX
13. Set dist[u] + graph[u][v] <dist[v])
14. Set dist[v] = dist[u] + graph[u][v]
15. Print Solution(dist)
16. Stop

Program

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
```

```
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false &&dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}
void printSolution(int dist[])
{   printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i< V; i++)
        printf("%d \t\t\t\t %d\n", i, dist[i]);
}


void dijkstra(int graph[V][V], int src)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i< V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v]
                &&dist[u] != INT_MAX
                &&dist[u] + graph[u][v] <dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist);
}


int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    dijkstra(graph, 0);
    return 0;
}
```

Output

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14