Experiment 18 Date: 15.12.2023

BFS and DFS

Aim:

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

Algorithm:

void bfs(int s, int n)

```
1. Start
2. int p, i;
3. enqueue(s);
4. vis[s] = 1;
5. p = dequeue();
6. if (p != 0)
        print “p”
7. 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)
                print “p”
8. for (i = 1; i <= n; i++)
        if (vis[i] == 0)
                bfs(i, n);
9. Stop
```

void enqueue(int item)

```
1. Start
2. if (rear == 19)
        print “QUEUE FULL”
3. else
        if (rear == -1)
                q[++rear] = item;
                front++;
```

```
         else
                 q[++rear] = item;
4.  Stop
```

int dequeue()

```
1.  Start
2.  int k;
3.  if ((front > rear) || (front == -1))
         return 0;
4.  else
         k = q[front++];
         return (k
5.  Stop
```

void dfs(int s, int n)

```
1.  Start
2.  int i, k;
3.  push(s);
4.  vis[s] = 1;
5.  k = pop();
6.  if (k != 0)
         print "k"
7.  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)
                 print "k"
8.  Stop
```

void push(int item)

```
1.  Start
2.  if (top == 19)
         print "Stack overflow"
3.  else
         stack[++top] = item;
```

```
4. Stop
5.
```

int pop()

```
1. Start
2. int k;
3.     if (top == -1)
           return (0);
4. else
           k = stack[top--];
           return (k);
5. Stop
```

void main()

```
1. Start
2. int n, i, s, ch, j;
3. char c;
4. print "ENTER THE NUMBER OF VERTICES”
5. for (i = 1; i <= n; i++)
           for (j = 1; j <= n; j++)
                   print “ENTER 1 IF %d HAS A NODE WITH %d ELSE 0”
6. print “THE ADJACENCY MATRIX IS”
7. for (i = 1; i <= n; i++)
           for (j = 1; j <= n; j++)
                   print “a[i][j]”
```

Program:

```
#include<stdio.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();

void main() {
    int n, i, s, ch, j;
    char c;
    printf("ENTER THE NUMBER VERTICES ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            printf("ENTER 1 IF %d HAS A NODE 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 U WANT TO CONTINUE(Y/N) ? ");
        scanf(" %c", &c);
    } while ((c == 'y') || (c == 'Y'));
}

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);
    }
}

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:

```
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ bfs&dfs.c
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ ./a.out
```

```
ENTER THE NUMBER VERTICES 4
ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0: 0
ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0: 1
ENTER 1 IF 1 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 1 HAS A NODE WITH 4 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0: 0
ENTER 1 IF 2 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 2 HAS A NODE WITH 4 ELSE 0: 1
ENTER 1 IF 3 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 3 HAS A NODE WITH 2 ELSE 0: 1
ENTER 1 IF 3 HAS A NODE WITH 3 ELSE 0: 0
ENTER 1 IF 3 HAS A NODE WITH 4 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 1 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 2 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 3 ELSE 0: 1
ENTER 1 IF 4 HAS A NODE WITH 4 ELSE 0: 0

THE ADJACENCY MATRIX IS
 0 1 1 1
 1 0 1 1
 1 1 0 1
 1 1 1 0

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE1
ENTER THE SOURCE VERTEX :1
1 2  3  4
DO U WANT TO CONTINUE(Y/N) ? y

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE2
ENTER THE SOURCE VERTEX :1
1    4  3  2
2    DO U WANT TO CONTINUE(Y/N) ?n
```

Experiment 19 Date: 20.12.2023

Prim’s Algorithm

Aim:

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

Algorithm:

int prims()

```
1.  Start
2.  int cost[MAX][MAX];
3.  int u, v, min_distance, distance[MAX], from[MAX];
4.  int visited[MAX], no_of_edges, i, min_cost, j;
5.  for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
            {
            if (graph[i][j] == 0)
                    cost[i][j] = infinity
            else
                    cost[i][j] = graph[i][j];
            spanning[i][j] = 0;
            }
6.  distance[0] = 0;
7.  visited[0] = 1;
8.  for (i = 1; i < n; i++)
            distance[i] = cost[0][i];
            from[i] = 0;
            visited[i] = 0;
9.  min_cost = 0;
10. no_of_edges = n - 1;
11. while (no_of_edges > 0)
            min_distance = infinity;
            for (i = 1; i < n; i++)
            {
            if (visited[i] == 0 && distance[i] < min_distance)
                    v = i;
                    min_distance = distance[i];
            }
```

```
          u = from[v];
          spanning[u][v] = distance[v];
          spanning[v][u] = distance[v];
          no_of_edges--;
          visited[v] = 1;
12. for (i = 1; i < n; i++)
          if (visited[i] == 0 && cost[i][v] < distance[i])
                distance[i] = cost[i][v];
                from[i] = v;
13. min_cost = min_cost + cost[u][v];
14. return (min_cost);
15. Stop
```

int main()

```
1.  Start
2.  int i, j, total_cost;
3.  print "Enter the number of vertices
4.  Print "Enter the adjacency matrix"
5.  for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
                scanf("%d", &graph[i][j]);
6.  print "Enter the cost of edges"
7.  for (i = 0; i < n; i++)
          for (j = i + 1; j < n; j++)
          {
          if (graph[i][j] != 0)
                print "Enter the cost of edge between vertex I and j"
                graph[j][i] = graph[i][j];
          }
8.  print "The cost adjacency matrix is"
9.  for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)

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

10. total_cost = prims();
11. print "Spanning tree matrix"
12. for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
                print "spanning[i][j]);"
13. print "Total cost of the spanning tree",total_cost
14. return 0;
```

15. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>

#define infinity 9999
#define MAX 20

int graph[MAX][MAX], spanning[MAX][MAX], n;
int prims();
int main() {
    int i, j, total_cost;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("\nEnter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    printf("\nEnter the cost of edges:\n");
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (graph[i][j] != 0) {
                printf("Enter the cost of edge between vertex %d and vertex %d: ", i,
j);
                scanf("%d", &graph[i][j]);
                graph[j][i] = graph[i][j];
            }
        }
    }

    printf("The cost adjacency matrix is:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%d\t", graph[i][j]);
        }
        printf("\n\n");
    }
    total_cost = prims();
    printf("\nSpanning tree matrix:\n");
    for (i = 0; i < n; i++) {
        printf("\n");
        for (j = 0; j < n; j++)
            printf("%d\t", spanning[i][j]);
    }
    printf("\n\nTotal cost of the spanning tree = %d", total_cost);

    return 0;
}

int prims() {
    int cost[MAX][MAX];
    int u, v, min_distance, distance[MAX], from[MAX];
    int visited[MAX], no_of_edges, i, min_cost, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            if (graph[i][j] == 0)
                cost[i][j] = infinity;
            else
                cost[i][j] = graph[i][j];
            spanning[i][j] = 0;
        }

    distance[0] = 0;
    visited[0] = 1;
    for (i = 1; i < n; i++) {
        distance[i] = cost[0][i];
        from[i] = 0;
        visited[i] = 0;
    }
    min_cost = 0;
    no_of_edges = n - 1;
    while (no_of_edges > 0) {
        min_distance = infinity;
        for (i = 1; i < n; i++)
            if (visited[i] == 0 && distance[i] < min_distance) {
                v = i;
                min_distance = distance[i];
            }
        u = from[v];

        spanning[u][v] = distance[v];
        spanning[v][u] = distance[v];
        no_of_edges--;
        visited[v] = 1;

        for (i = 1; i < n; i++)
            if (visited[i] == 0 && cost[i][v] < distance[i]) {
                distance[i] = cost[i][v];
                from[i] = v;
            }
        min_cost = min_cost + cost[u][v];
    }
    return (min_cost);
}
```

Output:

```
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ gcc prims.c
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ ./a.out

Enter the number of vertices: 4

Enter the adjacency matrix:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

Enter the cost of edges:
Enter the cost of edge between vertex 0 and vertex 1: 10
Enter the cost of edge between vertex 0 and vertex 2: 18
Enter the cost of edge between vertex 0 and vertex 3: 20
Enter the cost of edge between vertex 1 and vertex 2: 5
Enter the cost of edge between vertex 1 and vertex 3: 16
Enter the cost of edge between vertex 2 and vertex 3: 15
```

```
The cost adjacency matrix is:
0       10      18      20

10      0       5       16

18      5       0       15

20      16      15      0


Spanning tree matrix:

0       10      0       0
10      0       5       0
0       5       0       15
0       0       15      0

Total cost of the spanning tree = 30
```

Experiment 20 Date: 21.12.2023

Kruskal's Algorithm

Aim:

20. Program to implement Kruskal's algorithm.

Algorithm:

int find(int i)

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

int uni(int i, int j)

```
1. Start
2. if (i != j)
       parent[j] = i;
       return 1;
3. return 0;
4. Stop
```

void main()

```
1. Start
2. Print “Enter the number of vertices:”
3. Print “Enter the adjacency matrix.”
4. for (i = 1; i <= n; i++)
       for (j = 1; j <= n; j++)
           scanf("%d", &cost[i][j]);
           if (cost[i][j] == 0)
               cost[i][j] = 999;
5. print "Enter the cost of edges:”
6. for (i = 1; i <= n; i++)
       for (j = i + 1; j <= n; j++)
           if (cost[i][j] != 999)
               print “Enter the cost of edge between the vertex”
```

```
                    scanf("%d", &cost[i][j]);
                    cost[j][i] = cost[i][j];
7.  print “The edges of the minimum spanning tree are”
8.  while (ne < n)
        for (i = 1, min = 999; i <= n; i++)
            for (j = 1; j <= n; j++)
            {
            if (cost[i][j] < min)
                min = cost[i][j];
                a = u = i;
                b = v = j;
9.  u = find(u);
10. v = find(v);
11. if (uni(u, v))
        print "edge(%d,%d)=%d\n", ne++, a, b, min);
        mincost += min;
12. print “mincost”
```

Program:

```
#include <stdio.h>
#include <stdlib.h>

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[9][9], parent[9];

int find(int);
int uni(int, int);

void main()
{
   printf("\nEnter the number of vertices:");
   scanf("%d", &n);

   printf("\nEnter the adjacency matrix.\n");
   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n; j++)
      {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0)
```

```
            cost[i][j] = 999;
        }
    }

    printf("Enter the cost of edges:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = i + 1; j <= n; j++)
        {
            if (cost[i][j] != 999)
            {
                printf("Enter the cost of edge between vertex %d and vertex %d: ", i,
j);
                scanf("%d", &cost[i][j]);
                cost[j][i] = cost[i][j];
            }
        }
    }

    printf("The edges of the minimum spanning tree are\n");
    while (ne < n)
    {
        for (i = 1, min = 999; 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 edge(%d,%d)=%d\n", ne++, a, b, min);
            mincost += min;
        }
        cost[a][b] = cost[b][a] = 999;
    }
    printf("\n\tMinimum cost=%d\n", mincost);
}

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:

```
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ gcc Kruskal.c
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ ./a.out

Enter the number of vertices:4

Enter the adjacency matrix.
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

Enter the cost of edges:
Enter the cost of edge between vertex 1 and vertex 2: 10
Enter the cost of edge between vertex 1 and vertex 3: 18
Enter the cost of edge between vertex 1 and vertex 4: 20
Enter the cost of edge between vertex 2 and vertex 3: 5
Enter the cost of edge between vertex 2 and vertex 4: 16
Enter the cost of edge between vertex 3 and vertex 4: 15
```

```
The edges of the minimum spanning tree are
1 edge(2,3)=5
2 edge(1,2)=10
3 edge(3,4)=15

Minimum cost=30
```

Experiment 21 Date: 04.01.2024

Disjoint set operations

Aim:

21. Program to perform disjoint set operations create union.

Algorithm:

void initSets()

```
1. Start
2. int i;
3. for (i=0; i<numElements; i++)
        sets[i].parent=i;
        sets[i].rank=0;
```

int find(int element)

```
1. Start
2. if (sets[element].parent!=element)
        sets[element].parent=find(sets[element].parent);
3. return sets[element].parent;
4. Stop
```

void unionSets(int element1, int element2)

```
1. Start
2. int set1=find(element1);
3. int set2=find(element2);
4. if (set1 != set2)
        if (sets[set1].rank>sets[set2].rank)
                sets[set2].parent=set1;
5. else if(sets[set1].rank < sets[set2].rank
        sets[set1].parent =set2;
6. else
        sets[set2].parent =set1;
        sets[set1].rank++
7. Stop
```

void displaySets()

```
1. Start
2. int i;
3. print "Parent"
4. for (i=0; i<numElements; i++)
        print i;
5. print "Parent"
6. for (i=0; i<numElements; i++)
        print sets[i].parent)
7. print "rank"
8. for (i=0; i<numElements; i++)
        print sets[i].rank
9. Stop
```

int main()

```
1. Start
2. int i;
3. numElements = 6;
4. initSets();
5. displaySets();
6. unionSets(0, 1);
7. unionSets(1, 2);
8. unionSets (3, 4);
9. unionSets (4, 5);
10. unionSets (2, 4);
11. displaySets();
12. for (i=0; i<numElements; i++)
        print find(i);
13. return 0;
14. Stop
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ELEMENTS 1000
typedef struct Set
{
   int parent;
   int rank;
```

```
}Set;
Set sets[MAX_ELEMENTS];
int numElements;

void initSets() {
int i;
for (i=0; i<numElements; i++) {
   sets[i].parent=i;
   sets[i].rank=0;
}
}

int find(int element) {
   if (sets[element].parent!=element) {
      sets[element].parent=find(sets[element].parent);
}
return sets[element].parent;
}

void unionSets(int element1, int element2)
{
int set1=find(element1);
int set2=find(element2);
   if (set1 != set2)
   {
      if (sets[set1].rank>sets[set2].rank){
         sets[set2].parent=set1;
      }
      else if(sets[set1].rank < sets[set2].rank)
      {
         sets[set1].parent =set2;
      }
      else {
         sets[set2].parent =set1;
         sets[set1].rank++;
         }
   }
}

void displaySets()
{
int i;
```

```
printf("\nElement:\t");
for (i=0; i<numElements; i++)
   {
   printf("%d\t",i);
   }
   printf("\nParent:\t");
for (i=0; i<numElements; i++) {
   printf("%d\t", sets[i].parent);
   }
printf("\nRank:\t");
for (i=0; i<numElements; i++) {
   printf("%d\t", sets[i].rank);
   }
   printf("\n\n");
   }

int main(){
int i;
numElements = 6;
initSets();
displaySets();
unionSets(0, 1);
unionSets(1, 2);
unionSets (3, 4);
unionSets (4, 5);
unionSets (2, 4);
displaySets();
for (i=0; i<numElements; i++) {
   printf("%d",find(i));
}return 0;
}
```

Output:

```
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$gcc disjointset.c
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ ./a.out

Element:      0      1      2      3      4      5
Parent: 0     1      2      3      4      5
Rank:   0     0      0      0      0      0
```

```
Element:      0      1      2      3      4      5
Parent: 0     0      0      0      3      3
Rank:   2     0      0      1      0      0

The representative element of element 0 is 0
The representative element of element 1 is 0
The representative element of element 2 is 0
The representative element of element 3 is 0
The representative element of element 4 is 0
The representative element of element 5 is 0
```

Experiment 22 Date: 05.01.2024

Dijkstras algorithm

Aim:

22. 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. Declare dist[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, 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, 1,0,7), 0, 7),
{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
dijkstra(graph, 0);
return 0;
}
```

Output

```
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ gcc dijkstras.c
mits@mits-HP-280-Pro-G6-Microtower-PC:~/Desktop/S1MCA/ADS$ gcc ./a.out

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