Assignment-6

Name – Ujjwal Lehri
Roll No. – 60216403224
Course – B.Tech CSE 4th sem

Q1. WAP to implement BFS and DFS

1) **Breadth First Search (BFS):** a graph traversal algorithm that explores all nodes at the current level before moving to the next level

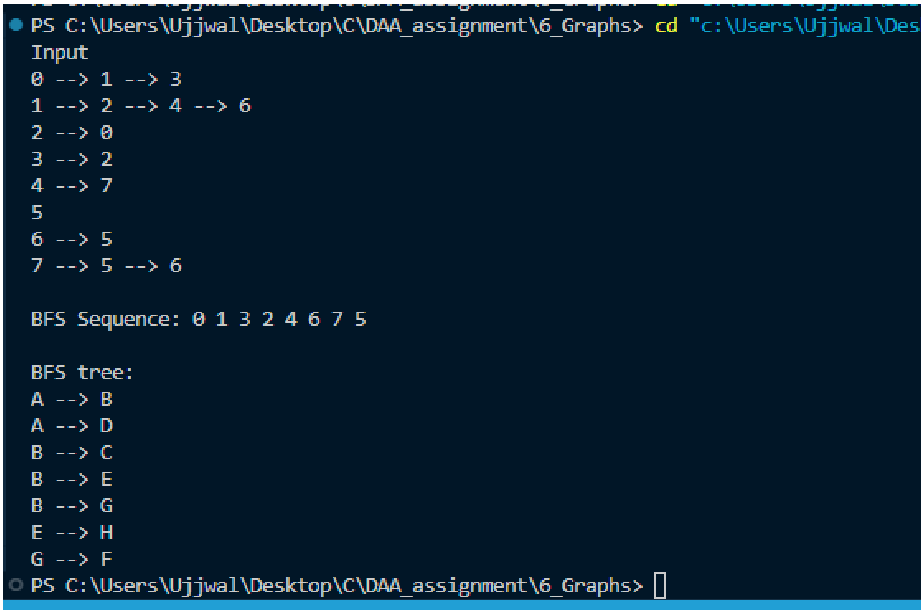
Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
typedef struct{
    int *items;
    int size;
    int front, end;
}Queue;

Queue *initQueue(int size){
    Queue *Q = (Queue *)malloc(sizeof(Queue));
    Q->size = size;
    Q->items = (int *)calloc(size, sizeof(int));
    Q->front = Q->end = -1;
    return Q;
}

void EnQ(Queue *Q, int val){
    if((Q->end+1)%Q->size == Q->front){
        printf("Queue is full");
        return;
    }
    if(Q->front == -1) Q->front = 0;
    Q->end = (Q->end+1)%Q->size;
    Q->items[Q->end] = val;
}

int DeQ(Queue *Q){
    if(Q->front == -1){
        printf("Queue is empty");
        return -1;
    }
    int val = Q->items[Q->front];
    if(Q->front == Q->end) Q->front = Q->end = -1;
    else Q->front = (Q->front+1)%Q->size;
    return val;
}
```

```c
int isEmpty(Queue *Q){
    if(Q->front == -1) return 1;
    return 0;
}

typedef struct{
    int V;
    int **adjMatrix;
    int *visited;
} Graph;

Graph *initGraph(int V){
    Graph *graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->visited = (int *)calloc(V, sizeof(int));
    graph->adjMatrix = (int **)malloc(V * sizeof(int*));
    for(int i = 0; i< V; i++)
        graph->adjMatrix[i] = (int *)calloc(V, sizeof(int*));

    return graph;
}

void addEdge(Graph *graph, int src, int dest){
    graph->adjMatrix[src][dest] = 1;
}

void printGraph(Graph* graph) {
    for (int i = 0; i < graph->V; i++) {
        printf("%d ", i);
        for (int j = 0; j < graph->V; j++) {
            if(graph->adjMatrix[i][j]){
                printf("--> %d ", j);
            }
        }
        printf("\n");
    }
    printf("\n");
}
```

```c
void BFS(Graph *graph, int startingVertex, int PiMatrix[8][8]){
    Queue *Q = initQueue(graph->V);

    EnQ(Q, startingVertex);

    while(!isEmpty(Q)){
        int v = DeQ(Q);
        printf("%d ", v);
        graph->visited[v] = 1;
        for(int i = 0; i<graph->V; i++){
            if(graph->adjMatrix[v][i] && !graph->visited[i]){
                graph->visited[i] = 1;
                PiMatrix[v][i] = v;
                EnQ(Q, i);
            }
        }
    }
}

void printPiMatrix(int PiMatrix[8][8], int V) {
    printf("\nBFS tree):\n");
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (PiMatrix[i][j] != -1) {
                printf("%c --> %c\n", 65+PiMatrix[i][j], 65+j);
            }
        }
    }
}

int main(){
    Graph* graph = initGraph(8);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 3);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 6);
    addEdge(graph, 2, 0);
    addEdge(graph, 3, 2);
    addEdge(graph, 4, 7);
    addEdge(graph, 6, 5);
    addEdge(graph, 7, 5);
    addEdge(graph, 7, 6);
    printGraph(graph);
    int PiMatrix[8][8];
    for(int i = 0; i<8; i++){
        for(int j = 0; j<8; j++){
            PiMatrix[i][j] = -1;
        }
    }
    BFS(graph, 0, PiMatrix);
    printPiMatrix(PiMatrix, 0);
    return 0;
}
```

Output:



```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:\Users\Ujjwal\Des
Input
0 --> 1 --> 3
1 --> 2 --> 4 --> 6
2 --> 0
3 --> 2
4 --> 7
5
6 --> 5
7 --> 5 --> 6

BFS Sequence: 0 1 3 2 4 6 7 5

BFS tree:
A --> B
A --> D
B --> C
B --> E
B --> G
E --> H
G --> F
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs>
```

2) Depth First Search (DFS): a graph traversal algorithm that explores as far as possible along each branch before backtracking.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100

typedef struct{
    int V;
    int **adjMatrix;
    int *visited;
} Graph;

Graph *initGraph(int V){
    Graph *graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->visited = (int *)calloc(V, sizeof(int*));
    graph->adjMatrix = (int **)malloc(V * sizeof(int*));
    for(int i = 0; i< V; i++)
        graph->adjMatrix[i] = (int *)calloc(V, sizeof(int*));
    return graph;
}

void addEdge(Graph *graph, int src, int dest){
    graph->adjMatrix[src][dest] = 1;
}
```

```
void printGraph(Graph* graph) {
    for (int i = 0; i < graph->V; i++) {
        printf("%d ", i);
        for (int j = 0; j < graph->V; j++) {
            if(graph->adjMatrix[i][j]){
                printf("--> %d ", j);
            }
        }
        printf("\n");
    }
    printf("\n");
}
int t = 0;
void DFS(Graph *graph, int startingVertex, int Stime[], int Ftime[]){
    int v = startingVertex;
    graph->visited[v] = 1;
    Stime[v] = ++t;
    printf("%c  ", 65+v);
    for(int i = 0; i<graph->V; i++){
        if(graph->adjMatrix[v][i] && !graph->visited[i]){
            DFS(graph, i, Stime, Ftime);
        }
    }
    Ftime[v] = ++t;
}
int main(){
    Graph* graph = initGraph(8);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 3);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 6);
    addEdge(graph, 2, 0);
    addEdge(graph, 3, 2);
    addEdge(graph, 4, 7);
    addEdge(graph, 6, 5);
    addEdge(graph, 7, 5);
    addEdge(graph, 7, 6);
    printf("Input");
    printGraph(graph);
    int Stime[8] = {0};
    int Ftime[8] = {0};
    printf("DFS Sequence: ");
    DFS(graph, 0, Stime, Ftime);
    printf("\n\n");
    for(int i = 0; i<8; i++){
        printf("%c (start = %02d, Finish = %02d)\n",65+i, Stime[i],
Ftime[i]);
    }
    return 0;}
```

Output:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:\Users\Ujjwal\D
Input0 --> 1 --> 3
1 --> 2 --> 4 --> 6
2 --> 0
3 --> 2
4 --> 7
5
6 --> 5
7 --> 5 --> 6

DFS Sequence: A  B  C  E  H  F  G  D

A (start = 01, Finish = 16)
B (start = 02, Finish = 13)
C (start = 03, Finish = 04)
D (start = 14, Finish = 15)
E (start = 05, Finish = 12)
F (start = 07, Finish = 08)
G (start = 09, Finish = 10)
H (start = 06, Finish = 11)
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs>
```

Q2. WAP to detect disjoint sets also calculates the number of disjoint sets there are.

Code:

```c
#include <stdio.h>
#define MAX 100

int parent[MAX];
int rank[MAX];

void makeSet(int n) {
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int find(int x) {
    if (parent[x] != x)
        parent[x] = find(parent[x]);
    return parent[x];
}
```

```c
void unionSets(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);

    if (rootX == rootY) return;

    if (rank[rootX] < rank[rootY])
        parent[rootX] = rootY;
    else if (rank[rootX] > rank[rootY])
        parent[rootY] = rootX;
    else {
        parent[rootY] = rootX;
        rank[rootX]++;
    }
}

int countDisjointSets(int n) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (parent[i] == i)
            count++;
    }
    return count;
}

void printDisjointSets(int n) {
    int sets[MAX][MAX];
    int size[MAX] = {0};
    for (int i = 0; i < n; i++) {
        int root = find(i);
        sets[root][size[root]++] = i;
    }

    printf("Disjoint Sets:\n");
    for (int i = 0; i < n; i++) {
        if (size[i] > 0) {
            printf("{ ");
            for (int j = 0; j < size[i]; j++) {
                printf("%d ", sets[i][j]);
            }
            printf("}\n");
        }
    }
}
```

```c
int main() {
    int n = 7;
    makeSet(n);

    unionSets(0, 1);
    unionSets(1, 2);
    unionSets(3, 4);
    unionSets(5, 6);

    printf("no. of disjoint sets: %d\n", countDisjointSets(n));
    printDisjointSets(n);

    return 0;
}
```

OUTPUT:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assig
no. of disjoint sets: 3
Disjoint Sets:
{ 0 1 2 }
{ 3 4 }
{ 5 6 }
PS C:\Users\Ujjwal\Desktop\C\DAA_assig
```

Q3. WAP to implement Dijkstra's and Bellman-Ford algo for single-source shortest path

1. **Dijkstra's Algorithm:**

   Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include <limits.h>
typedef struct{
    int V;
    int **adjMatrix;
} Graph;

typedef struct{
    int v, d, pi;
}Vertex;

typedef struct{
    Vertex *items;
    int capacity, size;
}PriorityQ;
```

```c
void swap(Vertex *a, Vertex *b, int *pos){
    int tempPos = pos[a->v];
    pos[a->v] = pos[b->v];
    pos[b->v] = tempPos;

    Vertex temp = *a;
    *a = *b;
    *b = temp;
}

void Heapify(Vertex *arr, int size, int i, int *pos){
    int L = 2*i + 1;
    int R = 2*i + 2;

    int min = i;
    if(L < size && arr[min].d > arr[L].d) min = L;
    if(R < size && arr[min].d > arr[R].d) min = R;

    if(min != i){
        swap(&arr[i], &arr[min], pos);
        Heapify(arr, size, min, pos);
    }

}


PriorityQ *initQueue(int capacity){
    PriorityQ *Queue = (PriorityQ*)malloc(sizeof(PriorityQ));
    Queue->items = (Vertex*)calloc(capacity , sizeof(Vertex));
    Queue->size = 0;
    Queue->capacity = capacity;
    return Queue;
}

void EnQ(PriorityQ *Q, Vertex val, int *pos){
    if(Q->size == Q->capacity){
        printf("Queue is full");
        return;
    }

    int i = Q->size;
    Q->items[i] = val;
    pos[val.v] = i;
    Q->size++;
```

```c
    while(i != 0 && Q->items[(i - 1) / 2].d > Q->items[i].d){
        swap(&Q->items[i], &Q->items[(i - 1) / 2], pos);
        i = (i - 1) / 2;
    }

}

Vertex ExtractMin(PriorityQ *Q, int *pos){
    if(Q->size == 0){
        printf("Queue is Empty");
        return (Vertex){-1,-1,-1};
    }

    Vertex val = Q->items[0];
    Q->items[0] = Q->items[Q->size - 1];
    pos[Q->items[0].v] = 0;
    Q->size--;
    Heapify(Q->items, Q->size, 0, pos);
    return val;
}


int isEmpty(PriorityQ *Q){
    return Q->size == 0;
}

void decreaseKey(PriorityQ *Q, int vertex, int newDist, int newPi, int
*pos) {
    int i = pos[vertex];
    Q->items[i].d = newDist;
    Q->items[i].pi = newPi;

    while (i != 0 && Q->items[(i - 1) / 2].d > Q->items[i].d) {
        swap(&Q->items[i], &Q->items[(i - 1) / 2], pos);
        i = (i - 1) / 2;
    }
}


void printQueue(PriorityQ *Q){
    printf("Queue (Min-Heap): ");
    for(int i = 0; i < Q->size; i++){
        printf("%d(%d)  ", Q->items[i].v, Q->items[i].d);
    }
    printf("\n");
}
```

```
Graph *initGraph(int V){
    Graph *graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->adjMatrix = (int **)malloc(V * sizeof(int*));
    for(int i = 0; i< V; i++)
        graph->adjMatrix[i] = (int *)calloc(V, sizeof(int*));

    return graph;
}

void addEdge(Graph *graph, int src, int dest, int weight){
    graph->adjMatrix[src][dest] = weight;
    // graph->adjMatrix[dest][src] = weight;
}

void Dijkstra(Graph *graph, int src, Vertex Sol[]){
    int visited[graph->V];
    int *pos = (int *)malloc(sizeof(int) * graph->V);
    for (int i = 0; i < graph->V; i++) visited[i] = 0;

    PriorityQ *Q = initQueue(graph->V);
    for(int i=0; i<graph->V; i++){
        if(i == src) EnQ(Q, (Vertex){i , 0, -1}, pos);
        else EnQ(Q, (Vertex){i , INT_MAX, -1}, pos);
    }

    while(!isEmpty(Q)){
        Vertex vertex = ExtractMin(Q, pos);
        visited[vertex.v] = 1;
        Sol[vertex.v] = vertex;
        for(int i = 0; i<graph->V; i++){
            if(graph->adjMatrix[vertex.v][i] != 0 && !visited[i]){
                int j = pos[i];
                int newDist = vertex.d + graph->adjMatrix[vertex.v][i];
                if( Q->items[j].d > newDist ){
                    decreaseKey(Q, i, newDist, vertex.v, pos);
                }
            }
        }
    }

    free(pos);
    free(Q->items);
    free(Q);

}
```

```c
void printGraph(Graph* graph) {
    for (int i = 0; i < graph->V; i++) {
        for (int j = 0; j < graph->V; j++) {
            printf("%02d ", graph->adjMatrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main(){
    int n= 6;
    Graph* graph = initGraph(n);
    addEdge(graph, 0, 1, 50);
    addEdge(graph, 0, 2, 45);
    addEdge(graph, 0, 3, 10);
    addEdge(graph, 1, 2, 10);
    addEdge(graph, 1, 3, 45);
    addEdge(graph, 2, 4, 30);
    addEdge(graph, 3, 0, 10);
    addEdge(graph, 3, 4, 15);
    addEdge(graph, 4, 1, 20);
    addEdge(graph, 4, 2, 35);
    addEdge(graph, 5, 5, 3);

    Vertex Sol[n];
    Dijkstra(graph, 0, Sol);

    for(int i = 1; i<n; i++){
        if(Sol[i].pi == -1) continue;
        printf("src(%d) ", Sol[0].v);
        printf("--> dest(%d) cost= %d \n", Sol[i].v, Sol[i].d);
    }
}
```

Output:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:
Input:
00 50 45 10 00 00
00 00 10 45 00 00
00 00 00 00 30 00
10 00 00 00 15 00
00 20 35 00 00 00
00 00 00 00 00 03

src(0) --> dest(1) cost= 45
src(0) --> dest(2) cost= 45
src(0) --> dest(3) cost= 10
src(0) --> dest(4) cost= 25
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs>
```

2. **BellmanFord Algorithm:**

   Code:

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

typedef struct {
    int V;
    int **adjMatrix;
} Graph;

typedef struct {
    int v;
    int d;
    int pi;
} Vertex;

Graph *initGraph(int V) {
    Graph *graph = (Graph *)malloc(sizeof(Graph));
    graph->V = V;
    graph->adjMatrix = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++)
        graph->adjMatrix[i] = (int *)calloc(V, sizeof(int));
    return graph;
}

void addEdge(Graph *graph, int src, int dest, int weight) {
    graph->adjMatrix[src][dest] = weight;
    // graph->adjMatrix[dest][src] = weight;
}
```

```c
int BellmanFord(Graph *graph, int src, Vertex Sol[]) {
    int V = graph->V;
    for (int i = 0; i < V; i++) {
        Sol[i].v = i;
        Sol[i].d = (i == src) ? 0 : INT_MAX;
        Sol[i].pi = -1;
    }

    for (int i = 1; i <= V - 1; i++) {
        for (int u = 0; u < V; u++) {
            for (int v = 0; v < V; v++) {
                if (graph->adjMatrix[u][v] != 0 && Sol[u].d != INT_MAX) {
                    int weight = graph->adjMatrix[u][v];
                    if (Sol[v].d > Sol[u].d + weight) {
                        Sol[v].d = Sol[u].d + weight;
                        Sol[v].pi = u;
                    }
                }
            }
        }
    }

    for (int u = 0; u < V; u++) {
        for (int v = 0; v < V; v++) {
            if (graph->adjMatrix[u][v] != 0 && Sol[u].d != INT_MAX) {
                int weight = graph->adjMatrix[u][v];
                if (Sol[u].d + weight < Sol[v].d) {
                    return 0;  // Negative cycle
                }
            }
        }
    }

    return 1; // No negative cycle
}

void printGraph(Graph *graph) {
    for (int i = 0; i < graph->V; i++) {
        printf("%d ", i);
        for (int j = 0; j < graph->V; j++) {
                printf("%02d ", graph->adjMatrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

```
int main() {
    int n = 5;
    Graph *graph = initGraph(n);

    addEdge(graph, 0, 1, -1);
    addEdge(graph, 0, 2, 4);
    addEdge(graph, 1, 2, 3);
    addEdge(graph, 1, 3, 2);
    addEdge(graph, 1, 4, 2);
    addEdge(graph, 3, 2, 5);
    addEdge(graph, 3, 1, 1);
    addEdge(graph, 4, 3, -3);

    printGraph(graph);

    Vertex Sol[n];
    int success = BellmanFord(graph, 0, Sol);

    if (success) {
        for (int i = 1; i < n; i++) {
            if (Sol[i].pi == -1) continue;
            printf("src(%d) --> dest(%d) cost= %d\n", Sol[0].v, Sol[i].v,
Sol[i].d);
        }
    } else {
        printf("Graph contains negative weight cycle.\n");
    }

    for (int i = 0; i < n; i++)
        free(graph->adjMatrix[i]);
    free(graph->adjMatrix);
    free(graph);



    return 0;
}
```

OutPut:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:\Use
o }
0 00 -1 04 00 00
1 00 00 03 02 02
2 00 00 00 00 00
3 00 01 05 00 00
4 00 00 00 -3 00

src(0) --> dest(1) cost= -1
src(0) --> dest(2) cost= 2
src(0) --> dest(3) cost= -2
src(0) --> dest(4) cost= 1
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs>
```

Q4. WAP to implement Floyd-Warshall's algorithm

Code:

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

#define INF INT_MAX

void printSolution(int **dist, int V) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            dist[i][j] == INF ? printf("%7s", "INF") : printf("%7d",
dist[i][j]);
        }
        printf("\n");
    }
}

void floydWarshall(int **adjMatrix, int V) {
    printf("\n---------------------------------------------");
    printf("\n\t\t Input \t\t\n");
    printf("---------------------------------------------\n");
    printSolution(adjMatrix, V);
    printf("---------------------------------------------\n\n");


    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (adjMatrix[i][k] != INF && adjMatrix[k][j] != INF &&
adjMatrix[i][j] > adjMatrix[i][k] + adjMatrix[k][j]) {
                    adjMatrix[i][j] = adjMatrix[i][k] + adjMatrix[k][j];
                }
            }
        }

        printf("\n----------------------------------------------------");
        printf("\n Shortest path considering %d intermediate vertices \n",
k+1);
        printf("----------------------------------------------------\n");
        printSolution(adjMatrix, V);
        printf("----------------------------------------------------\n\n");
    }
}
```

```c
int main() {
    int V = 5;
    int inputGraph[5][5] = {
        {0,   2,   10,  11,  INF},
        {INF, 0,   3,   INF, INF},
        {6,   INF, 0,   INF, 5},
        {INF, INF, 2,   0,   18},
        {7,   INF, INF, INF, 0},
    };

    int **adjMatrix = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; i++){
        adjMatrix[i] = (int *)malloc(V * sizeof(int));
        for (int j = 0; j < V; j++)
            adjMatrix[i][j] = inputGraph[i][j];
    }

    floydWarshall(adjMatrix, V);

    return 0;
}
```

Output:

```
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs"
PS C:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs> cd "c:\Users\Ujjwal\Desktop\C\DAA_assignment\6_Graphs\" ; i

 ---------------------------------------------
                  Input
 ---------------------------------------------
      0       2      10      11     INF
    INF       0       3     INF     INF
      6     INF       0     INF       5
    INF     INF       2       0      18
      7     INF     INF     INF       0
 ---------------------------------------------


 ----------------------------------------------------
  Shortest path considering 1 intermediate vertices
 ----------------------------------------------------
      0       2      10      11     INF
    INF       0       3     INF     INF
      6       8       0      17       5
    INF     INF       2       0      18
      7       9      17      18       0
 ----------------------------------------------------


 ----------------------------------------------------
  Shortest path considering 2 intermediate vertices
 ----------------------------------------------------
      0       2       5      11     INF
    INF       0       3     INF     INF
      6       8       0      17       5
    INF     INF       2       0      18
      7       9      12      18       0
 ----------------------------------------------------


 ----------------------------------------------------
  Shortest path considering 3 intermediate vertices
 ----------------------------------------------------
      0       2       5      11      10
      9       0       3      20       8
      6       8       0      17       5
      8      10       2       0       7
      7       9      12      18       0
 ----------------------------------------------------


 ----------------------------------------------------
  Shortest path considering 4 intermediate vertices
 ----------------------------------------------------
      0       2       5      11      10
      9       0       3      20       8
      6       8       0      17       5
      8      10       2       0       7
      7       9      12      18       0
 ----------------------------------------------------


 ----------------------------------------------------
  Shortest path considering 5 intermediate vertices
 ----------------------------------------------------
      0       2       5      11      10
      9       0       3      20       8
      6       8       0      17       5
      8      10       2       0       7
      7       9      12      18       0
 ----------------------------------------------------
```