Name: Sandesh Kawane Roll No: 15 Div: SY CS D

Assignment 7

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <stdbool.h>
#include <stddef.h>
struct AdjListNode
{
    int dest;
    int weight;
    struct AdjListNode* next;
};

struct AdjList
{

struct AdjListNode *head;
};
struct Graph
{
    int V;
    struct AdjList* array;
};

struct AdjListNode* newAdjListNode(
                int dest, int weight)
{
    struct AdjListNode* newNode =
            (struct AdjListNode*)
    malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*)
            malloc(sizeof(struct Graph));
    graph->V = V;

    graph->array = (struct AdjList*)
    malloc(V * sizeof(struct AdjList));

    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}
```

```
// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src,
                int dest, int weight)
{
    struct AdjListNode* newNode =
            newAdjListNode(dest, weight);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    newNode = newAdjListNode(src, weight);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

struct MinHeapNode
{
    int v;
    int dist;
};

struct MinHeap
{

    int size;

    int capacity;

    int *pos;
    struct MinHeapNode **array;
};

struct MinHeapNode* newMinHeapNode(int v,
                                int dist)
{
    struct MinHeapNode* minHeapNode =
        (struct MinHeapNode*)
    malloc(sizeof(struct MinHeapNode));
    minHeapNode->v = v;
    minHeapNode->dist = dist;
    return minHeapNode;
}

struct MinHeap* createMinHeap(int capacity)
{
    struct MinHeap* minHeap =
        (struct MinHeap*)
    malloc(sizeof(struct MinHeap));
    minHeap->pos = (int *)malloc(
            capacity * sizeof(int));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array =
        (struct MinHeapNode**)
```

```
                malloc(capacity *
    sizeof(struct MinHeapNode*));
    return minHeap;
}

void swapMinHeapNode(struct MinHeapNode** a,
                     struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
void minHeapify(struct MinHeap* minHeap,
                                int idx)
{
    int smallest, left, right;
    smallest = idx;
    left = 2 * idx + 1;
    right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->dist <
        minHeap->array[smallest]->dist )
    smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->dist <
        minHeap->array[smallest]->dist )
    smallest = right;

    if (smallest != idx)
    {
        // The nodes to be swapped in min heap
        struct MinHeapNode *smallestNode =minHeap->array[smallest];
        struct MinHeapNode *idxNode =
                minHeap->array[idx];

        // Swap positions
        minHeap->pos[smallestNode->v] = idx;
        minHeap->pos[idxNode->v] = smallest;

        // Swap nodes
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);

        minHeapify(minHeap, smallest);
    }
}

int isEmpty(struct MinHeap* minHeap)
{
    return minHeap->size == 0;
}
```

```
struct MinHeapNode* extractMin(struct MinHeap*
                               minHeap)
{
    if (isEmpty(minHeap))
        return NULL;

    // Store the root node
    struct MinHeapNode* root =
                minHeap->array[0];

    // Replace root node with last node
    struct MinHeapNode* lastNode =
        minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    // Update position of last node
    minHeap->pos[root->v] = minHeap->size-1;
    minHeap->pos[lastNode->v] = 0;

    // Reduce heap size and heapify root
    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
}

void decreaseKey(struct MinHeap* minHeap,
                        int v, int dist)
{
    // Get the index of v in heap array
    int i = minHeap->pos[v];

    // Get the node and update its dist value
    minHeap->array[i]->dist = dist;
    while (i && minHeap->array[i]->dist <
        minHeap->array[(i - 1) / 2]->dist)
    {
        // Swap this node with its parent
        minHeap->pos[minHeap->array[i]->v] =
                                    (i-1)/2;
        minHeap->pos[minHeap->array[
                            (i-1)/2]->v] = i;
        swapMinHeapNode(&minHeap->array[i],
                &minHeap->array[(i - 1) / 2]);

        // move to parent index
        i = (i - 1) / 2;
    }
}

bool isInMinHeap(struct MinHeap *minHeap, int v)
{
if (minHeap->pos[v] < minHeap->size)
    return true;
```

```
return false;
}

void printArr(int dist[], int n)
{
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(struct Graph* graph, int src)
{

    // Get the number of vertices in graph
    int V = graph->V;

    // dist values used to pick
    // minimum weight edge in cut
    int dist[V];

    // minHeap represents set E
    struct MinHeap* minHeap = createMinHeap(V);

    // Initialize min heap with all
    // vertices. dist value of all vertices
    for (int v = 0; v < V; ++v)
    {
        dist[v] = INT_MAX;
        minHeap->array[v] = newMinHeapNode(v,
                                    dist[v]);
        minHeap->pos[v] = v;
    }

    // Make dist value of src vertex
    // as 0 so that it is extracted first
    minHeap->array[src] =
        newMinHeapNode(src, dist[src]);
    minHeap->pos[src] = src;
    dist[src] = 0;
    decreaseKey(minHeap, src, dist[src]);

    // Initially size of min heap is equal to V
    minHeap->size = V;

    while (!isEmpty(minHeap))
    {
        // Extract the vertex with
        // minimum distance value
        struct MinHeapNode* minHeapNode =
                    extractMin(minHeap);

        // Store the extracted vertex number
        int u = minHeapNode->v;
```

```
        struct AdjListNode* pCrawl =
                    graph->array[u].head;
        while (pCrawl != NULL)
        {
            int v = pCrawl->dest;

            if (isInMinHeap(minHeap, v) &&
                    dist[u] != INT_MAX &&
            pCrawl->weight + dist[u] < dist[v])
            {
                dist[v] = dist[u] + pCrawl->weight;

                // update distance
                // value in min heap also
                decreaseKey(minHeap, v, dist[v]);
            }
            pCrawl = pCrawl->next;
        }
    }

    // print the calculated shortest distances
    printArr(dist, V);
}


// Driver program to test above functions
int main()
{
    // create the graph given in above figure
    int V = 9;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 4);
    addEdge(graph, 0, 7, 8);
    addEdge(graph, 1, 2, 8);
    addEdge(graph, 1, 7, 11);
    addEdge(graph, 2, 3, 7);
    addEdge(graph, 2, 8, 2);
    addEdge(graph, 2, 5, 4);
    addEdge(graph, 3, 4, 9);
    addEdge(graph, 3, 5, 14);
    addEdge(graph, 4, 5, 10);
    addEdge(graph, 5, 6, 2);
    addEdge(graph, 6, 7, 1);
    addEdge(graph, 6, 8, 6);
    addEdge(graph, 7, 8, 7);

    dijkstra(graph, 0);

}
```