DAA LAB TASK 10

Write a C/C++ Program to implement Dijkstra Algorithm **(Using Min-Heap Data Structure)** to find single source shortest path for a Graph G.

Instructions:

(i) Please ensure your code should contain build_heap (), extract_min () and decrease_key () functions to implement Dijkstra Algorithm.
(ii) You may store the Graph Weight Matrix in the main function itself.

CODE

```
#include <iostream>
#include <climits>
#include <vector>
using namespace std;

#define V 5

struct MinHeapNode {
    int v;
    int dist;
};

struct MinHeap {
    int size;
    int capacity;
    int *pos;
    MinHeapNode **array;
};


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


MinHeap* createMinHeap(int capacity) {
    MinHeap* minHeap = new MinHeap;
```

```
    minHeap->pos = new int[capacity];
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = new MinHeapNode*[capacity];
    return minHeap;
}



void swapMinHeapNode(MinHeapNode** a, MinHeapNode** b) {
    MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}



void minHeapify(MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int 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) {
        MinHeapNode* smallestNode = minHeap->array[smallest];
        MinHeapNode* idxNode = minHeap->array[idx];



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



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

        minHeapify(minHeap, smallest);
    }
}
```

```
void build_heap(MinHeap* minHeap) {
    int n = minHeap->size;
    for (int i = (n - 2) / 2; i >= 0; i--)
        minHeapify(minHeap, i);
}

MinHeapNode* extract_min(MinHeap* minHeap) {
    if (minHeap->size == 0)
        return nullptr;

    MinHeapNode* root = minHeap->array[0];
    MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
    minHeap->array[0] = lastNode;

    minHeap->pos[root->v] = minHeap->size - 1;
    minHeap->pos[lastNode->v] = 0;

    --minHeap->size;
    minHeapify(minHeap, 0);

    return root;
}


void decrease_key(MinHeap* minHeap, int v, int dist) {
    int i = minHeap->pos[v];
    minHeap->array[i]->dist = dist;

    while (i && minHeap->array[i]->dist < minHeap->array[(i - 1) / 2]->dist) {
        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]);
        i = (i - 1) / 2;
    }
}


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

```
void printArr(int dist[], int n) {
    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < n; ++i)
        cout << i << "\t\t" << dist[i] << "\n";
}


void dijkstra(int graph[V][V], int src) {
    int dist[V];



    MinHeap* minHeap = createMinHeap(V);

    for (int v = 0; v < V; ++v) {
        dist[v] = INT_MAX;
        minHeap->array[v] = newMinHeapNode(v, dist[v]);
        minHeap->pos[v] = v;
    }



    dist[src] = 0;
    minHeap->array[src] = newMinHeapNode(src, dist[src]);
    minHeap->pos[src] = src;
    minHeap->size = V;



    build_heap(minHeap);

    while (minHeap->size > 0) {
        MinHeapNode* minNode = extract_min(minHeap);
        int u = minNode->v;

        for (int v = 0; v < V; ++v) {
            if (graph[u][v] && isInMinHeap(minHeap, v) &&
                dist[u] != INT_MAX &&
                graph[u][v] + dist[u] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                decrease_key(minHeap, v, dist[v]);
            }
        }
    }
```

```
    printArr(dist, V);
}


int main() {

    int graph[V][V] = {
        {0, 9, 0, 0, 0},
        {9, 0, 10, 0, 0},
        {0, 10, 0, 15, 2},
        {0, 0, 15, 0, 6},
        {0, 0, 2, 6, 0}
    };

    int src = 0;
    dijkstra(graph, src);

    return 0;
}
```

OUTPUT

```
Vertex  Distance from Source
0               0
1               9
2               19
3               27
4               21

c:\Users\KIIT0001\OneDrive\Desktop>
```