Dijkstra's Algorithm using Min-Heap

```
#include <iostream>

#include <climits>

using namespace std;


#define V 5

struct MinHeapNode {

   int v;

   int dist;

};


struct MinHeap {

   int size;      // Number of heap nodes present currently

   int capacity;   // Capacity of the min-heap

   int *pos;      // Needed for decrease_key()

   MinHeapNode **array;

};


// Function to create a new MinHeapNode

MinHeapNode* newMinHeapNode(int v, int dist) {

   MinHeapNode* node = new MinHeapNode;

   node->v = v;

   node->dist = dist;

   return node;

}


// Function to create a MinHeap

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;
}

// Function to swap two nodes of min-heap
void swapMinHeapNode(MinHeapNode** a, MinHeapNode** b) {
    MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// Standard minHeapify function
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];

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

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

        minHeapify(minHeap, smallest);
    }
}

// Function to check if the given minHeap is empty
bool isEmpty(MinHeap* minHeap) {
    return minHeap->size == 0;
}

// ------------------ REQUIRED FUNCTIONS ------------------

// (i) Build Heap (initialization)
void build_heap(MinHeap* minHeap, int dist[]) {
    for (int v = 0; v < V; ++v) {
        minHeap->array[v] = newMinHeapNode(v, dist[v]);
        minHeap->pos[v] = v;
```

```
    }
    minHeap->size = V;

    // Build the heap (bottom-up heapify)
    for (int i = (minHeap->size - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

// (ii) Extract-Min function
MinHeapNode* extract_min(MinHeap* minHeap) {
    if (isEmpty(minHeap))
        return NULL;

    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;
}

// (iii) Decrease-Key function
```

```
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;
    }
}

// Utility to check if a vertex is in minHeap
bool isInMinHeap(MinHeap *minHeap, int v) {
    if (minHeap->pos[v] < minHeap->size)
        return true;
    return false;
}

// ------------------ Dijkstra Algorithm ------------------

void dijkstra(int graph[V][V], int src) {
    int dist[V]; // Output array: dist[i] will hold the shortest distance from src to i

    // Initialize distances
    for (int v = 0; v < V; ++v)
        dist[v] = INT_MAX;
    dist[src] = 0;
```

```
// Create a MinHeap
MinHeap* minHeap = createMinHeap(V);

// Build initial heap
build_heap(minHeap, dist);

while (!isEmpty(minHeap)) {
  // Extract vertex with minimum distance
  MinHeapNode* minNode = extract_min(minHeap);
  int u = minNode->v;

  // Update distance values of adjacent vertices
  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]);
    }
  }
}

// Print shortest distances
cout << "Vertex\tDistance from Source\n";
for (int i = 0; i < V; ++i)
  cout << i << "\t" << dist[i] << endl;
```

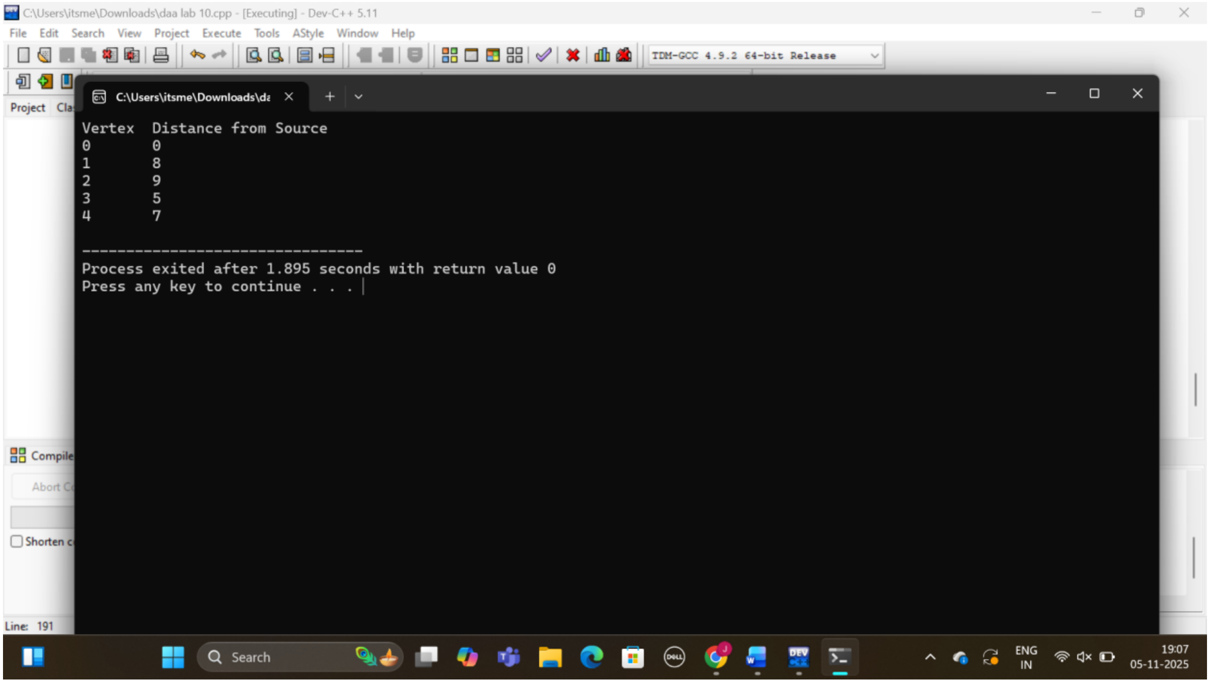
```
}

// ------------------ MAIN FUNCTION ------------------

int main() {
    // Graph represented as adjacency matrix
    // 0 means no edge
    int graph[V][V] = {
        {0, 10, 0, 5, 0},
        {0, 0, 1, 2, 0},
        {0, 0, 0, 0, 4},
        {0, 3, 9, 0, 2},
        {7, 0, 6, 0, 0}
    };

    int source = 0; // Starting vertex
    dijkstra(graph, source);

    return 0;
}
```


C:\Users\itsme\Downloads\daa lab 10.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
Vertex Distance from Source
0   0
1   8
2   9
3   5
4   7

Process exited after 1.895 seconds with return value 0
Press any key to continue . . .
Line: 191
Search
ENG IN
19:07
05-11-2025