**SHORTEST PATH ALGORITHMS AND THEIR APPLICATION IN G.P.S**

**SLOT: G1**

**Name of faculty**: **Prof. KAVYA ALLURU**

**(SCHOOL OF COMPUTIND SCIENCE AND ENGINEERING)**

NOVEMBER, 2017

**By:**

**Pruthviraj.R.Patil (16BCE1096)**

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts with success. The acknowledgement transcends the reality of formality when we would like to express the deep gratitude and respect to all those people behind the screen who guided, inspired and helped me for the completion of my project work.

We express my sincere thanks to my guide **Prof. Kavya A (Professor for course CSE2003, VIT university, Chennai)**, for her generous help and constant encouragement during the project development.

**CERTIFICATE**

This is to certify that the Project work entitled “**SHORTEST PATH ALGORITHMS AND THEIR APPLICATION IN G.P.S** ” that is being submitted for CAL in**B.Tech DATA STRUCTURES AND ALGORITHMS(SCE2003)** is a record of bonafide work done under my supervision.The contents of this Project work have not been submitted for any other CAL course.

**ABSTRACT:**

The uses of the Data structures are vast in number and one of the major applications among them is the implementation of the Global Positioning System (GPS) that gives us the efficient pathway for travelling from given position to specified destination.

The graph Data structures are used to create the GPS and more than one techniques can be applied to achieve the working system. Some of the basic Algorithms that we can use are:

1. Bell-man ford Algorithm
2. Dijkstra’s Algorithm
3. Floyd-Warshall’s Algorithm
4. Johnson’s Algorithm

all the above mentioned algorithms follow the techniques of recursion and repeated squaring, the final output is achieved. Here we concentrate only on the Dijkstra’s and Floyd’s algorithm.

The GPS in the project shows the shortest and the efficient path for the traveller to move from his current position to the destinations as the above algorithms are applied using the weighted graph not only by using the factor of length but also by the factor of the traffic between the two positions i.e. source and the destination.

This program can be used for building the app to buy online bus tickets, having the customer decide the source and destination based on the distance and the traffic rate. It can also be used by many E-Commerce companies for the efficient home deliveries of the products to the customers at lesser time.

**DESCRIPTION AND ALGORITHMS**

1. **Dijkstra’s algorithm**

**Input**: Graph G=(V, E), directed or Undirected;

Positive edge lengths{Ie: e belongs to E}; Vertex s belongs to V

**Output**: For all the vertices u reachable from s, dist(u) to the distance from s to u.

**STEPS:**

For all u belongs to V:

Dist(u)=inf

Prev(u)=nill

Dist(s)=0

H=makequeue(V) (using the dist-values as keys)

While H is not empty:

U=deletemin(H)

For all edges (u, v) belongs to E:

If dist(v)> dist(u)+l(u, v):

dist(v)=dist(u)+l(u, v)

prev(v)=u

decreasekey(H, v)

**Floyd-Warshall’s Algorithm**

**STEPS:**

Create a |V| x |V| matrix, M, that will describe the distances between vertices

For each cell (i, j) in M:

if i == j:

M[i][j] = 0

if (i, j) is an edge in E:

M[i][j] = weight(i, j)

else:

M[i][j] = infinity

for i from 1 to |V|:

for j from 1 to |V|:

for k from 1 to |V|:

if M[i][j] > M[i][k] + M[k][j]:

M[i][j] = M[i][k] + M[k][j]

**ANALYSIS**

The time complexity of the Dijkstra’s algorithm can vary many times based on how it is being implemented. In case of using the Fibonacci heaps

**T(n) =O(|E| +|V|log|V|),**

If the minimum heap is used to implement the priority queue, then the time complexity becomes **O(VElogV)**

Whereas the time complexity of the Warshall’s algorithm is clearly:

**T(n)= O(V^3)**

**CODE IMPLEMENTATION WITH SCREEN SHOTS**

**Dijkstra’s algorithm is implemented using the minimum heaps as well as the adjacent list.**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <vector>

#include <algorithm>

#include <string>

struct AdjacentListNode

{

int destination;

int weight;

struct AdjacentListNode\* next;

};

struct AdjList

{

struct AdjacentListNode \*head;

};

struct Graph

{

int V;

struct AdjList\* array;

};

struct AdjacentListNode\* newAdjListNode(int destination, int weight)

{

struct AdjacentListNode\* newNode =

(struct AdjacentListNode\*) malloc(sizeof(struct AdjacentListNode));

newNode->destination = destination;

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;

}

void addEdge(struct Graph\* graph, int src, int destination, int weight)

{

struct AdjacentListNode\* newNode = newAdjListNode(destination, weight);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

newNode = newAdjListNode(src, weight);

newNode->next = graph->array[destination].head;

graph->array[destination].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)

{

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);

}

}

int isEmpty(struct MinHeap\* minHeap)

{

return minHeap->size == 0;

}

struct MinHeapNode\* extractMin(struct MinHeap\* minHeap)

{

if (isEmpty(minHeap))

return NULL;

struct MinHeapNode\* root = minHeap->array[0];

struct 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 decreaseKey(struct 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(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)

{

int V = graph->V;

int dist[V];

struct 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;

}

minHeap->array[src] = newMinHeapNode(src, dist[src]);

minHeap->pos[src] = src;

dist[src] = 0;

decreaseKey(minHeap, src, dist[src]);

minHeap->size = V;

while (!isEmpty(minHeap))

{

struct MinHeapNode\* minHeapNode = extractMin(minHeap);

int u = minHeapNode->v;

struct AdjacentListNode\* pCrawl = graph->array[u].head;

while (pCrawl != NULL)

{

int v = pCrawl->destination;

if (isInMinHeap(minHeap, v) && dist[u] != INT\_MAX &&

pCrawl->weight + dist[u] < dist[v])

{

dist[v] = dist[u] + pCrawl->weight;

decreaseKey(minHeap, v, dist[v]);

}

pCrawl = pCrawl->next;

}

}

printArr(dist, V);

}

int main()

{

int V = 15;

struct Graph\* graph = createGraph(V);

addEdge(graph, 0, 1, 6);

addEdge(graph, 0, 2, 9);

addEdge(graph, 1, 3, 5);

addEdge(graph, 1, 6, 6);

addEdge(graph, 1, 7, 8);

addEdge(graph, 2, 4, 7);

addEdge(graph, 3, 4, 7);

addEdge(graph, 3, 5, 9);

addEdge(graph, 3, 6, 7);

addEdge(graph, 4, 5, 5);

addEdge(graph, 5, 10, 14);

addEdge(graph, 5, 9, 6);

addEdge(graph, 6, 7, 8);

addEdge(graph, 6, 8, 6);

addEdge(graph, 6, 9, 3);

addEdge(graph, 7, 8, 6);

addEdge(graph, 7, 13, 2);

addEdge(graph, 8, 14, 17);

addEdge(graph, 9, 11, 8);

addEdge(graph, 9, 12, 8);

addEdge(graph, 10, 11, 8);

addEdge(graph, 11, 12, 12);

addEdge(graph, 12, 14, 11);

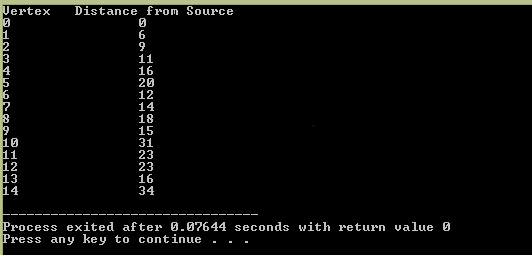
addEdge(graph, 13, 14, 20);

dijkstra(graph, 0);

return 0;

}

**OUTPUT SCREENSHOTS**



**FLOYD’s Algorithm:**

#include<stdio.h>

#define V 15

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshall (int graph[][V])

{

int dist[V][V], i, j, k;

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

for (k = 0; k < V; k++)

{

for (i = 0; i < V; i++)

{

for (j = 0; j < V; j++)

{

if(i==j)

{

continue;

}

else

{

if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

printSolution(dist);

}

void printSolution(int dist[][V])

{

printf ("Following matrix shows the shortest distances"

" between every pair of vertices \n");

for (int i = 0; i < V; i++)

{

for (int j = 0; j < V; j++)

{

if (dist[i][j] == INF)

printf("%7s", "INF");

else

printf ("%7d", dist[i][j]);

}

printf("\n");

}

}

int main()

{

int graph[V][V] =

{

{0, 6, 9, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF},

{6, 0, INF, 5, INF, 6, 8, INF, INF, INF, INF, INF, INF, INF},

{9, 5, 0, INF, 7, INF, INF, INF, INF, INF, INF, INF, INF, INF},

{INF, INF, INF, 0, 7, 9, 7, INF, INF, INF, INF, INF, INF, INF, INF},

{INF, INF, 7, 7, 0, 5, INF, INF, INF, INF, INF, INF, INF, INF, INF},

{INF, INF, INF, 9, 5, 6, INF, INF, INF, 6, 14, INF, INF, INF, INF},

{6, INF, INF, INF, INF, INF, 0, 8, 6, 3, INF, INF, INF, INF, INF},

{INF, 8, INF, 7, INF, INF, 8, 0, 6, INF, INF, INF, INF, 2, INF},

{INF, INF, INF, INF, INF, INF, 6, 6, 0, INF, INF, INF, INF, INF, 17},

{INF, INF, INF, INF, INF, 6, 3, INF, INF, 0, INF, 8, 8, INF, INF},

{INF, INF, INF, INF, INF, 14, INF, INF, INF, INF, 0, 8, INF, INF, INF},

{INF, INF, INF, INF, INF, INF, INF, INF, INF, 8, 8, 0, 12, INF, INF},

{INF, INF, INF, INF, INF, INF, INF, INF, INF, 8, INF, 12, 0, INF, 11},

{INF, INF, INF, INF, INF, INF, INF, 2, INF, INF, INF, INF, INF, 0, 20},

{INF, INF, INF, INF, INF, INF, INF, INF, 17, INF, INF, INF, 11, 20, 0},

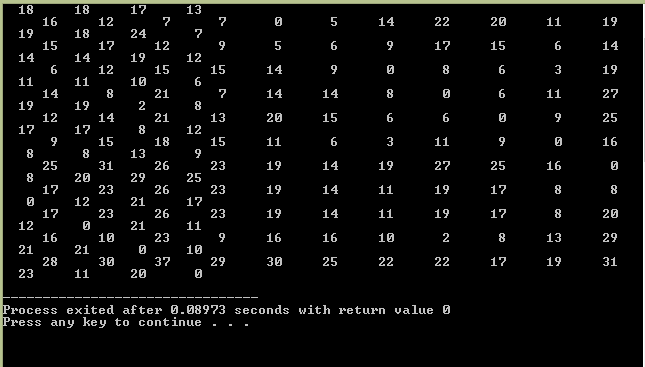
};

floydWarshall(graph);

return 0;

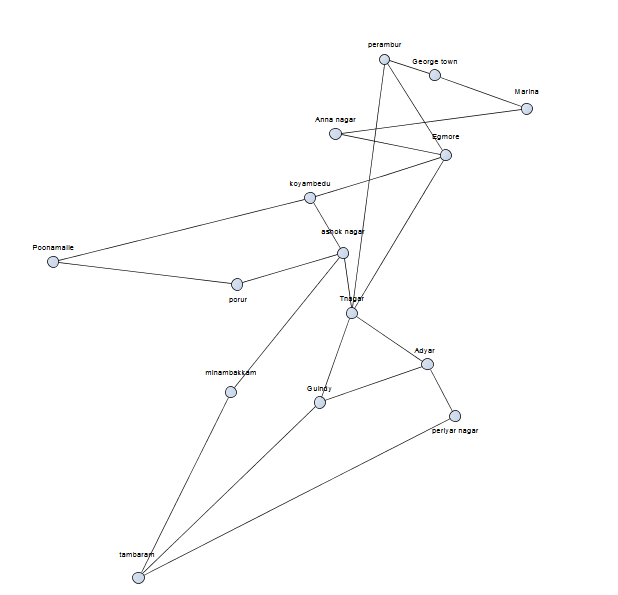
}

**OUTPUT SCREENSHOTS**



**(ADJACENCY MATRIX WITH LEAST DISTANCES BETWEEN ALL THE VERTICES)**

**(Here the input is given as per the abstract graph of the map of Chennai city is taken)**



The areas are mapped with following vertices:

1.George town

2. Marina

3. Perambur

4.Egmore

5.Annanagar

6.Koyambedu

7. Tnagar

8.Adyar

9.Guindy

10.Ashok Nagar

11. Poonamalle

12.Porur

13.Minambakkam

14.Periyar Nagar

15.Tambaram

**CONCLUSION**

The project shows the shortest and the efficient path for the traveller to move from his current position to the destinations as the above algorithms are applied using the weighted graph not only by using the factor of length but also by the factor of the traffic between the two positions i.e. source and the destination.

This program can be used for building the app to buy online bus tickets, having the customer decide the source and destination based on the distance and the traffic rate. It can also be used by many E-Commerce companies for the efficient home deliveries of the products to the customers at lesser time.