#### **Ques 2**

* Generate a random graph with n > 10 nodes.
* Generate random n^2/2 directly edges.
* Assign a weight in the range (1, 10).
* Apply Dijkstra’s algorithm.
* Find out the single source shortest path.

##### **Generate Random Graph**

Graph::Graph(**int** n) : number\_of\_vertices(n)

{

connectivity\_matrix = **new** **float** \*[number\_of\_vertices];

**int** i;

**for** (i = 0; i < number\_of\_vertices; i++)

{

connectivity\_matrix[i] = **new** **float**[number\_of\_vertices];

}

}

Graph::~Graph()

{

**int** i;

open\_set.clear();

closed\_set.clear();

**for** (i = 0; i < number\_of\_vertices; i++)

{

**delete** [] connectivity\_matrix[i];

}

**delete** [] connectivity\_matrix;

connectivity\_matrix = NULL;

}

**##### Initializing Graph**

**void** Graph::initiate\_graph(**float** d, **float** min, **float** max)

{

**if** (NULL == connectivity\_matrix){

cout << "Memory Allocate Failed!" <<endl;

**return** ;

}

**if** ((min <= 0.0) || (max <= 0.0) || (min >= max)){

cout << "Invalid Edge Cost Range!" <<endl;

**return** ;

}

graph\_density = d;

**int** i, j;

**float** random;

srand((**unsigned** **int**)time(NULL));

**for** (i = 0; i < number\_of\_vertices; i++)

{

connectivity\_matrix[i][i] = 0.0; *//the cost from i to i is 0*

**for** (j = i + 1; j < number\_of\_vertices; j++)

{

random = random\_generator(0.0, 1.0); *//get a decimal between 0 and 1*

**if** (random >= graph\_density) *//there is no path if random is less than density*

{

connectivity\_matrix[i][j] = 0.0;

connectivity\_matrix[j][i] = connectivity\_matrix[i][j]; *//undirected graph*

}

**else** *//else, there is a path*

{

connectivity\_matrix[i][j] = random\_generator(min, max); *//get a value between min to max, default is 1 to 10*

connectivity\_matrix[j][i] = connectivity\_matrix[i][j];

}

}

}

}

##### **Generate random cost edges**

**float** random\_generator(**float** lower, **float** upper)

{

*//Get random Cost*

**int** range = (**int**)upper \* 10 - (**int**)lower \* 10;

**int** temp = rand() % range + (**int**)lower \* 10;

**return** (**float**)temp / 10;

}

##### **Apply Dijkstra’s Algorithm**

void Graph::dijkstra\_algorithm(int s, int t){

**if** (closed\_set.empty() == true) {

Vertex V;

V.vertex\_no = s;

V.cost\_from\_start = 0;

V.path\_from\_start.push\_back(s);

closed\_set.push\_back(V);

}

int current = s;

**if** ((false == update\_open\_set(closed\_set.back())) && (open\_set.empty() == true)) {

**return**; //stop **when** open set **is** **not** updated **and** it **is** empty

}**else**{

current = update\_closed\_set();

}

**if** (current == t){

**return**; //stop **when** destination **is** included **in** closed set

} **else**{

dijkstra\_algorithm(current, t);

}

}

##### **Find Shortest Path**

**float** Graph::get\_shortest\_path(**int** s, **int** t){

**if** ((s < 1) || (s > number\_of\_vertices)) {

cout << "No Such Start Vertex " << s <<endl;

**return** 0.0;

}

**if** ((t < 1) || (t > number\_of\_vertices)) {

cout << "No Such End Vertex " << t <<endl;

**return** 0.0;

}

**if** (s == t) {

**return** 0.0;

}

dijkstra\_algorithm(s - 1, t - 1); *//perform dijkstra's algorithm*

Vertex T = closed\_set.back(); *//the last member of closed set*

**if** (T.vertex\_no != t - 1) *//the destination is not in*

{

cout << "No Path From " << s << " To " << t << endl;

open\_set.clear();

closed\_set.clear();

**return** -1;

}

cout <<" "<<s <<"\t" <<" "<<t << "\t";

cout <<T.cost\_from\_start <<"\t\t";

list<**int**>::iterator iter;

**for** (iter = T.path\_from\_start.begin(); iter != T.path\_from\_start.end(); iter++){

**if** (\*iter != T.path\_from\_start.back()){

cout << (\*iter) + 1 << "->";

}**else**{

cout << (\*iter) + 1 <<endl;

}

}

open\_set.clear();

closed\_set.clear();

**return** T.cost\_from\_start;

}

**SCREENSHOTS**



