Q. 1. Implement shortest path algorithm

**#include <limits.h>**

**#include <stdio.h>**

**// Number of vertices in the graph**

**#define V 9**

**// A utility function to find the vertex with minimum distance value, from**

**// the set of vertices not yet included in shortest path tree**

**int minDistance(int dist[], bool sptSet[])**

**{**

**// Initialize min value**

**int min = INT\_MAX, min\_index;**

**for (int v = 0; v < V; v++)**

**if (sptSet[v] == false && dist[v] <= min)**

**min = dist[v], min\_index = v;**

**return min\_index;**

**}**

**// A utility function to print the constructed distance array**

**int printSolution(int dist[], int n)**

**{**

**printf("Vertex Distance from Source\n");**

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

**printf("%d \t\t %d\n", i, dist[i]);**

**}**

**// Function that implements Dijkstra's single source shortest path algorithm**

**// for a graph represented using adjacency matrix representation**

**void dijkstra(int graph[V][V], int src)**

**{**

**int dist[V]; // The output array. dist[i] will hold the shortest**

**// distance from src to i**

**bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest**

**// path tree or shortest distance from src to i is finalized**

**// Initialize all distances as INFINITE and stpSet[] as false**

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

**dist[i] = INT\_MAX, sptSet[i] = false;**

**// Distance of source vertex from itself is always 0**

**dist[src] = 0;**

**// Find shortest path for all vertices**

**for (int count = 0; count < V - 1; count++)**

**{**

**// Pick the minimum distance vertex from the set of vertices not**

**// yet processed. u is always equal to src in the first iteration.**

**int u = minDistance(dist, sptSet);**

**// Mark the picked vertex as processed**

**sptSet[u] = true;**

**// Update dist value of the adjacent vertices of the picked vertex.**

**for (int v = 0; v < V; v++)**

**// Update dist[v] only if is not in sptSet, there is an edge from**

**// u to v, and total weight of path from src to v through u is**

**// smaller than current value of dist[v]**

**if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX**

**&& dist[u] + graph[u][v] < dist[v])**

**dist[v] = dist[u] + graph[u][v];**

**}**

**// print the constructed distance array**

**printSolution(dist, V);**

**}**

**// driver program to test above function**

**int main()**

**{**

**/\* Let us create the example graph discussed above \*/**

**int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },**

**{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },**

**{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },**

**{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },**

**{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },**

**{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },**

**{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },**

**{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },**

**{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };**

**dijkstra(graph, 0);**

**return 0;**

**}**

Q. 2. Write a menu driven program to perform following operations on singly linked list: Create,

Insert, Delete, and Display

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* head = NULL;

void insert(int x) {

Node\* temp = new Node();

temp->data = x;

temp->next = head;

head = temp;

}

void Delete(int n) {

Node\* temp1 = head;

if(n == 1) {

head = temp1->next;

delete temp1;

return;

}

for(int i=0; i<n-2; i++) {

temp1 = temp1->next;

}

Node\* temp2 = temp1->next;

temp1->next = temp2->next;

delete temp2;

}

void display() {

Node\* temp = head;

while(temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

int main() {

int choice, x, n;

while(1) {

cout << "1. Insert" << endl;

cout << "2. Delete" << endl;

cout << "3. Display" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch(choice) {

case 1: cout << "Enter the element: ";

cin >> x;

insert(x);

break;

case 2: cout << "Enter the element you want to delete: ";

cin >> n;

Delete(n);

break;

case 3: display();

break;

case 4: exit(0);

default: cout << "Invalid Input" << endl;

}

}

return 0;

}