```cpp
#include <iostream>
#include <stdio.h>
#include <vector>
#define NON_HA_PARENT -1
using namespace std;

void costruisci_vettore_citta(int vertici, vector<string> vettore_citta, string citta) {
    cout << "Inserisci il numero di nodi: ";
    cin >> vertici;
    cout << "Inserisci i nomi delle citta: ";
    for (int o = 0; o < vertici; o++)
    {
        cin >> citta;
        vettore_citta.push_back(citta);
    }
}

void prendi_source(int vertici, vector<string> vettore_citta, string radice_stringa, int radice) {
    cout << "Inserisci il nome della citta da cui vuoi partire: ";
    cin >> radice_stringa;

    for (int u = 0; u < radice_stringa.size(); u++)
    {
        radice_stringa[u] = tolower(radice_stringa[u]);
    }

    for (int w = 0; w < vertici; w++)
    {
        if (radice_stringa == vettore_citta[w]) {
            radice = w;
        }

    }
    cout << endl;
}

void costruisci_graph(vector<vector<int>> graph, int vertici, vector<string> vettore_citta) {

    cout << "Costruiamo il graph con le citta." << endl << endl;
    int x;
    for (int i = 0; i < vertici; i++)
    {
        vector<int> v1;

        for (int j = 0; j < vertici; j++) {
            cout << "Distanza da " << vettore_citta[i] << " a " << vettore_citta[j] << ": ";
            cin >> x;
            v1.push_back(x);
        }
```

```
50          graph.push_back(v1);
51      }
52      /*
53      cout << "La matrice associata e: " << endl;
54
55      for (int k = 0; k < vertici ; k++)
56      {
57          for (int l = 0; l < vertici; l++) {
58              cout << graph[k][l]<< "  ";
59          }
60          cout << endl;
61      }
62  */
63  }
64
65  void mostra_citta(vector<string> vettore_citta, int vertici) {
66      cout << "Ecco le citta ordinate per numero: " << endl;
67      for (int k = 0; k < vertici; k++)
68      {
69          cout << k << ") " << vettore_citta[k] << endl;
70      }
71  }
72
73  void mostra_path(int vertice_corrente, vector<int> parent,
      vector<string>vettore_citta) {
74      if (vertice_corrente == NON_HA_PARENT) {
75          return;
76      }
77      mostra_path(parent[vertice_corrente], parent, vettore_citta);
78      cout << vettore_citta[vertice_corrente] << " ";
79  }
80
81  int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool>
      sono_stati_visitati, int vertici) {
82      int valoremin = INT_MAX;
83      int nodominimo = 0;
84      for (int i = 0; i < vertici; i++)
85      {
86          if (!sono_stati_visitati[i] && distanze[i] <= valoremin) //
              viene preso e aggiunto il nodo con la
87                                                                    //
                      minima distanza che non è nel vettore bool dei
                      visitati
88          {
89              valoremin = distanze[i];
90              nodominimo = i;
91          }
92      }
93      return nodominimo; //viene dato all'algoritmo
94  }
95
96  void mostra(vector<int> distanze, int vertici,int radice, vector<int>
      parent,vector<string> vettore_citta)
```

```
 97 {
 98
 99     cout << "Citta:\t\t\t\tDistanza:\t\t\t\tPercorso piu veloce:";
100     for (int i = 0; i < vertici; i++) {
101         if ( i != radice )
102         {
103             cout << endl;
104             cout << vettore_citta[radice] << " -> " <<vettore_citta[i]
                  << "\t\t\t" << distanze[i] << "\t\t\t"; mostra_path(i,
                  parent, vettore_citta);
105         }
106     }
107 }
108
109 void algoritmo(vector<vector<int>> graph, int radice, int vertici,
      vector<string> vettore_citta) {
110
111     vector<int> distanze(vertici), parent(vertici);
112     vector<bool> sono_stati_visitati(vertici); //flags per i visitati
113     distanze[radice] = 0;// distanza dalla radice = 0
114     parent[radice] = NON_HA_PARENT;
115
116     //inizializzo tutto
117     for (int i = 0; i < vertici; i++)
118     {
119         distanze[i] = INT_MAX;
120         sono_stati_visitati[i] = false;
121     }
122
123     for (int i = 0; i < vertici - 1 ; i++)
124     {
125         int nodo_vicino = controllo_dei_nodi_vicini(distanze,
              sono_stati_visitati, vertici);
126         //nodi vicini vengono analizzati, int nodo vicino è il nodo
              preso in considerazione
127
128         sono_stati_visitati[nodo_vicino] = true; //è stato visitato il
              nodo vicino
129
130         for (int adiacente = 0; adiacente < vertici; adiacente++) //
              fase di update delle distanze dei vertici adiacenti
131         {
132             if (!sono_stati_visitati[adiacente]           //se non è
                  nell'array dei visitati "sono_stati_visitati"
133                 && graph[nodo_vicino][adiacente]          //esiste la
                      connessione tra il vicino e l'adiacente
134                 && distanze[nodo_vicino] != INT_MAX       //distanza
                      del nodo vicino non è infinita
135                 &&  distanze[nodo_vicino] + graph[nodo_vicino]
                      [adiacente] < distanze[adiacente]) //il peso del
                      viaggio dalla source al nodo adiacente è piccolo
                      rispoetto alla distanza con v, cioè
136             {
```

```
137                 distanze[adiacente] = distanze[nodo_vicino] + graph
                      [nodo_vicino][adiacente];
138                 parent[adiacente] = nodo_vicino; //parent serve per
                      printare il path
139             }
140         }
141     }
142     mostra(distanze, vertici, radice, parent, vettore_citta);
143 }
144
145 int main()
146 {
147     int vertici = 0, radice = 0;
148     string citta, radice_stringa;
149     vector<string> vettore_citta;
150     vector<vector<int>> graph;
151
152     costruisci_vettore_citta(vertici, vettore_citta, citta);
153     mostra_citta(vettore_citta, vertici);
154     costruisci_graph(graph, vertici, vettore_citta);
155     prendi_source(vertici, vettore_citta, radice_stringa, radice);
156     algoritmo(graph, radice , vertici, vettore_citta);
157
158     return 0;
159 }
160
```