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

//void mostra_citta(vector<string> vect_citta, int n_nodi);
//void mostra(vector<int> distanze, int n_nodi, int radice, vector<int> parent, vector<string> vect_citta);

void algoritmo(vector<vector<int>> graph, int radice, int n_nodi, vector<string> vect_citta);
void mostra_path(int vertice_corrente, vector<int> parent, vector<string>vect_citta);
int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool> sono_stati_visitati, int n_nodi);
int prendi_source(int& n_nodi, vector<string>& vect_citta);
bool controllo_presenza(string& input);
vector<vector<int>> costruisci_graph(int n_nodi, vector<string> vect_citta);
vector<string> crea_vect_citta(int& n_nodi);

class db_le_citta
{
public:
    vector<vector<int>> distanze_citta ={//0   1   2   3   4   5   6   7   8   9   10  11  12
                                        {0  ,153,0  ,0  ,263,0  ,312,0  ,0  ,0  ,0  ,0  ,0  },          //bari
                                        {153,0  ,582,0  ,413,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  },          //lecce
                                        {0  ,582,0  ,211,592,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  },          //catania
                                        {0  ,0  ,211,0  ,716,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  },          //palermo
                                        {263,413,592,716,0  ,227,247,0  ,0  ,0  ,0  ,0  ,0  },          //napoli
                                        {0  ,0  ,0  ,0  ,227,0  ,210,275,0  ,0  ,0  ,513,0  },          //roma
                                        {312,0  ,0  ,0  ,247,210,0  ,399,365,0  ,0  ,0  ,0  },          //pescara
                                        {0  ,0  ,0  ,0  ,0  ,275,399,0  ,105,0  ,0  ,253,0  },          //firenze
                                        {0  ,0  ,0  ,0  ,0  ,0  ,365,105,0  ,154,213,294,0  },          //bologna
                                        {0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,154,0  ,278,0  ,0  },          //venezia
                                        {0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,213,278,0  ,346,143},          //milano
                                        {0  ,0  ,0  ,0  ,0  ,513,0  ,253,294,0  ,346,0  ,169},          //genova
                                        {0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,0  ,143,169,0  },          //torino
    };
    vector<string> nomi_citta =
      {"bari","lecce","catania","palermo","napoli","roma","pescara","firenze","bologna","venezia","milano","genova"
      ,"torino"};
    //aggiungere metodi per aggiunta di citta
    vector<vector<int>> costruzione_graph_classe(int& n_nodi, vector<string>& vect_citta)
    {
        vector<vector<int>> v1;
        for (int i = 0; i < n_nodi; i++) // loop per n nodi che ho messo
        {
            for (int j = 0; j < nomi_citta.size(); j++) //loop per n citta che esistono
            {
                if (vect_citta[i] == nomi_citta[j])
                {
                    v1.push_back(distanze_citta[j]);
                }
            }
        }
        return v1;
    }
};

int main()
{
    int n_nodi = 0;
    cout << "Inserisci il numero di nodi: ";
    cin >> n_nodi;

    vector<string> vect_citta = crea_vect_citta(n_nodi);
    vector<vector<int>> graph = costruisci_graph(n_nodi, vect_citta);
    int source = prendi_source(n_nodi, vect_citta);
    algoritmo(graph, source, n_nodi, vect_citta);
    return 0;
}

void algoritmo(vector<vector<int>> graph, int radice, int n_nodi, vector<string> vect_citta) {

    vector<int> distanze(n_nodi), parent(n_nodi);
    vector<bool> sono_stati_visitati(n_nodi); //flags per i visitati
    //inizializzo tutto
    for (int i = 0; i < n_nodi; i++)
    {
        distanze[i] = INT_MAX;
        sono_stati_visitati[i] = false;
    }

```

```
79      distanze[radice] = 0;// distanza dalla radice = 0
80      parent[radice] = NON_HA_PARENT;
81
82      for (int i = 0; i < n_nodi - 1; i++)
83      {
84          int nodo_vicino = controllo_dei_nodi_vicini(distanze, sono_stati_visitati, n_nodi);
85          //nodi vicini vengono analizzati, int nodo vicino è il nodo preso in considerazione
86
87          sono_stati_visitati[nodo_vicino] = true; //è stato visitato il nodo vicino
88
89          for (int adiacente = 0; adiacente < n_nodi; adiacente++) //fase di update delle distanze dei n_nodi
              adiacenti
90          {
91              if (!sono_stati_visitati[adiacente]            //se non è nell'array dei visitati "sono_stati_visitati"
92                  && graph[nodo_vicino][adiacente]           //esiste la connessione tra il vicino e l'adiacente
93                  && distanze[nodo_vicino] != INT_MAX        //distanza del nodo vicino non è infinita
94                  && distanze[nodo_vicino] + graph[nodo_vicino][adiacente] < distanze[adiacente]) //il peso del
                      viaggio dalla source al nodo adiacente è piccolo rispetto alla distanza corrente dell'adiacente
95              {
96                  parent[adiacente] = nodo_vicino; //parent serve per printare il path
97                  distanze[adiacente] = distanze[nodo_vicino] + graph[nodo_vicino][adiacente];
98              }
99          }
100     }
101
102     //mostriamo
103     for (int i = 0; i < n_nodi; i++) {
104         if (i != radice)
105         {
106             cout << endl;
107             cout << "Nodo: " << vect_citta[radice] << " -> " << vect_citta[i] << "\tDistanza: " << distanze[i] <<
                  "\tPercorso:\t";
108             mostra_path(i, parent, vect_citta);
109         }
110     }
111 }
112
113 void mostra_path(int vertice_corrente, vector<int> parent, vector<string>vect_citta) {
114     if (vertice_corrente == NON_HA_PARENT) {
115         return;
116     }
117     mostra_path(parent[vertice_corrente], parent, vect_citta);
118     cout << vect_citta[vertice_corrente] << " ";
119 }
120
121 int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool> sono_stati_visitati, int n_nodi) {
122     int valoremin = INT_MAX;
123     int nodominimo = 0;
124     for (int i = 0; i < n_nodi; i++)
125     {
126         if (!sono_stati_visitati[i] && distanze[i] <= valoremin) //viene preso e aggiunto il nodo con la
127             //minima distanza che non è nel vect bool dei visitati
128         {
129             valoremin = distanze[i];
130             nodominimo = i;
131         }
132     }
133     return nodominimo; //viene dato all'algoritmo
134 }
135
136 vector<vector<int>> costruisci_graph(int n_nodi, vector<string> vect_citta) {
137
138     int risp;
139     cout << "Vuoi costruire con distanze a tuo piacimento (seleziona 1) oppure gia determinate (seleziona 0) ?" <<
          endl;
140     cin >> risp;
141
142
143     if (risp == 1)
144     {
145         vector<vector<int>> graph;
146         cout << "Costruiamo il graph con le citta e le loro distanze." << endl;
147         int x;
148         for (int i = 0; i < n_nodi; i++)
149         {
150             vector<int> v1;
151             for (int j = 0; j < n_nodi; j++) {
152                 cout << "Distanza da " << vect_citta[i] << " a " << vect_citta[j] << ": ";
153                 cin >> x;
154                 v1.push_back(x);
```

```
            }
            graph.push_back(v1);
        }
        return graph;
    }
    else {
        db_le_citta db;
        //correggi quiwfghsnefhuweicnhfwefwiecoriwehufwefnweuchfnweoifnowefcoiwefnwefhunwecfhwe
        return db.costruzione_graph_classe(n_nodi, vect_citta); //sul database uso la funzione che mi controlla
    }
}

vector<string> crea_vect_citta(int& n_nodi) {
    string citta;
    vector<string> vet;
    db_le_citta db;

    cout << "Inserisci i nomi delle citta / i nodi: ";
    for (int o = 0; o < n_nodi; o++)
    {
        do
        {
            cin >> citta;
            for (int u = 0; u < citta.size(); u++)
            {
                citta[u] = tolower(citta[u]);
            }
            if (controllo_presenza(citta) == false)
            {
                cout << "Citta non dichiarata!" << endl;
            }
        } while (controllo_presenza(citta) == false);
        vet.push_back(citta);
    }

    return vet;
}

int prendi_source(int& n_nodi, vector<string>& vect_citta) {
    string rad;
    cout << "Inserisci il nome della citta da cui vuoi partire: ";

    do
    {
        cin >> rad;
        for (int u = 0; u < rad.size(); u++)
        {
            rad[u] = tolower(rad[u]);
        }
        if (controllo_presenza(rad) == false)
        {
            cout << "Citta non dichiarata!" << endl;
        }
    } while (controllo_presenza(rad) == false);

    for (int w = 0; w < n_nodi; w++)
    {
        if (rad == vect_citta[w]) {
            return w;
        }
    }
}

bool controllo_presenza(string& input) {
    db_le_citta db;
    for (int i = 0; i < db.nomi_citta.size(); i++)
    {
        if (input == db.nomi_citta[i])
        {
            return true;
        }
    }
    return false;
}
```