```cpp
#include <iostream>
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
#include <fstream>

#define NON_HA_PARENT -1
using namespace std;

//funzioni generali
bool controllo_presenza_classe(string input);
bool controllo_presenza_vettore(string input, vector<string> vect_citta);
bool risp_main();
void dijkstra();

//specifiche
int prendi_n_nodi();
vector<string> crea_vect_citta(int n_nodi);
vector<vector<int>> costruisci_graph(int n_nodi, vector<string> vect_citta);
int prendi_source(int n_nodi, vector<string> vect_citta);

//algoritmo
int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool> sono_stati_visitati, int n_nodi);
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);



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"};

    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;
    }

    void aggiungi_citta() {
        string nuova_citta;
        cout << "Inserisci il nome della nuova citta: ";
        cin >> nuova_citta;
        for (int u = 0; u < nuova_citta.size(); u++)
        {
            nuova_citta[u] = tolower(nuova_citta[u]);
        }

        while (controllo_presenza_classe(nuova_citta) == true) //non c'è la città
        {
```

```
            cout << "Citta fa gia parte del database!" << endl;
            cin.clear();
            cin.ignore(256, '\n');
            cin >> nuova_citta;
        }
        nomi_citta.push_back(nuova_citta);

        cout << "Inserisci le distanze della nuova citta per " << endl;
        vector<int> distanze_nuova_citta;
        int d = 0;
        for (int i = 0; i < nomi_citta.size() - 1; i++)
        {
            cout << nomi_citta[i] << ": ";
            cin >> d;
            distanze_nuova_citta.push_back(d);
        }
        cout << "Ecco il vettore con le distanze della nuova citta che vuoi aggiungere! Copia e incolla nel
          programma per averlo permanentemente nel database."<<endl;
        for (int a = 0; a < distanze_nuova_citta.size(); a++)
        {
            cout << distanze_nuova_citta[a] << " ,";
        }

    }

    void display_nomi() {
        cout << "Ecco le citta disponibili" << endl;
        for (int i = 0; i < nomi_citta.size(); i++)
        {
            cout << i << ") " << nomi_citta[i] << endl;
        }
    }

    void display_database() {
        cout << "Ecco le distanze presenti nel database" << endl;
        for (int i = 0; i < distanze_citta.size(); i++)
        {
            cout << "Citta di "<< nomi_citta[i] << ":   \t";
            for (int j = 0; j < distanze_citta[i].size(); j++)
            {
                cout << distanze_citta[i][j] << " ";
            }cout << endl;
        }
    }

    int prendi_source_classe(int n_nodi) {
        string rad;
        vector<string>::iterator f;

        cout << "Inserisci il nome della citta da cui vuoi partire: ";
        cin >> rad;
        for (int u = 0; u < rad.size(); u++)
        {
            rad[u] = tolower(rad[u]);
        }

        while (controllo_presenza_classe(rad) == false) //non c'è la città
        {
            cout << "Citta non fa parte delle citta nel database!" << endl;
            cin.clear();
            cin.ignore(256, '\n');
            cin >> rad;
        }

        f = find(nomi_citta.begin(), nomi_citta.end(), rad);
        if (f != nomi_citta.end())
        {
            return (f - nomi_citta.begin());
        }

    }

    void dijkstrasudb() {
        int source = prendi_source_classe(nomi_citta.size());
        algoritmo(distanze_citta, source, nomi_citta.size(), nomi_citta);
    }
};

int main()
{
```

```
158     while(1){
159         db_le_citta db;
160         int x;
161         cout << "Ciao! Cosa vuoi fare?" << endl;
162         cout << "1. Aggiungere una nuova citta" << endl;
163         cout << "2. Vedere le citta disponibili" << endl;
164         cout << "3. Vedere il database" << endl;
165         cout << "4. Usare Dijkstra" << endl;
166         cout << "5. Dijkstra su tutto il database" << endl;
167         cin >> x;
168         switch (x)
169         {
170         case 1:
171             db.aggiungi_citta();
172             break;
173         case 2:
174             db.display_nomi();
175             break;
176         case 3:
177             db.display_database();
178             break;
179         case 4:
180             dijkstra();
181             break;
182         case 5:
183             db.dijkstrasudb();
184             break;
185         default:
186             break;
187         }
188         cout << endl;
189     }
190
191     return 0;
192 }
193
194 int prendi_n_nodi() {
195     int n_nodi;
196     cout << "Inserisci il numero di nodi: ";
197     cin >> n_nodi;
198     while (cin.fail())
199     {
200         cout << "Inserisci un numero!" << endl;
201         cin.clear();
202         cin.ignore(256, '\n');
203         cin >> n_nodi;
204     }
205     return n_nodi;
206
207 }
208
209 vector<string> crea_vect_citta(int n_nodi) {
210     string citta;
211     vector<string> vet;
212     db_le_citta db;
213
214     cout << "Inserisci i nomi delle citta / i nodi: ";
215     for (int o = 0; o < n_nodi; o++)
216     {
217         do
218         {
219             cin >> citta;
220             for (int u = 0; u < citta.size(); u++)
221             {
222                 citta[u] = tolower(citta[u]);
223             }
224             if (controllo_presenza_classe(citta) == false)
225             {
226                 cout << "Citta non dichiarata!" << endl;
227             }
228
229         } while (controllo_presenza_classe(citta) == false);
230         vet.push_back(citta);
231     }
232
233     return vet;
234 }
235
236 vector<vector<int>> costruisci_graph(int n_nodi, vector<string> vect_citta) {
237
```

```
238     int risp;
239     cout << "Vuoi costruire con distanze a tuo piacimento (seleziona 1) oppure gia determinate (seleziona 0) ?" << endl;
240     cin >> risp;
241
242     if (risp == 1)
243     {
244         vector<vector<int>> graph;
245         cout << "Costruiamo il graph con le citta e le loro distanze." << endl;
246         int x;
247         for (int i = 0; i < n_nodi; i++)
248         {
249             vector<int> v1;
250             for (int j = 0; j < n_nodi; j++) {
251                 cout << "Distanza da " << vect_citta[i] << " a " << vect_citta[j] << ": ";
252                 cin >> x;
253                 v1.push_back(x);
254             }
255             graph.push_back(v1);
256         }
257         return graph;
258     }
259     else {
260         db_le_citta db;
261         return db.costruzione_graph_classe(n_nodi, vect_citta);
262     }
263 }
264
265 int prendi_source(int n_nodi, vector<string> vect_citta) {
266     string rad;
267     vector<string>::iterator f;
268
269     cout << "Inserisci il nome della citta da cui vuoi partire: ";
270     cin >> rad;
271     for (int u = 0; u < rad.size(); u++)
272     {
273         rad[u] = tolower(rad[u]);
274     }
275
276     while (controllo_presenza_classe(rad) == false || controllo_presenza_vettore(rad, vect_citta) == false) //non c'è la città
277     {
278         cout << "Citta non fa parte delle città nel database o è estranea alle citta che hai selezionato!" << endl;
279         cin.clear();
280         cin.ignore(256, '\n');
281         cin >> rad;
282     }
283
284     f = find(vect_citta.begin(), vect_citta.end(), rad);
285     if (f != vect_citta.end())
286     {
287         return (f - vect_citta.begin());
288     }
289
290 }
291
292 bool controllo_presenza_classe(string input) {
293     db_le_citta db;
294     for (int i = 0; i < db.nomi_citta.size(); i++)
295     {
296         if (input == db.nomi_citta[i])
297         {
298             return true;
299         }
300     }
301     return false;
302 }
303
304 bool controllo_presenza_vettore(string input, vector<string> vect_citta) {
305     for (int i = 0; i <vect_citta.size(); i++)
306     {
307         if (input == vect_citta[i])
308         {
309             return true;
310         }
311     }
312     return false;
313 }
314
315 int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool> sono_stati_visitati, int n_nodi) {
```

```
316     int valoremin = INT_MAX;
317     int nodominimo = 0;
318     for (int i = 0; i < n_nodi; i++)
319     {
320         if (!sono_stati_visitati[i] && distanze[i] <= valoremin)
321             //viene preso e aggiunto il nodo con la
322             //minima distanza che non è nel vect bool dei visitati
323         {
324             valoremin = distanze[i];
325             nodominimo = i;
326         }
327     }
328     return nodominimo; //viene dato all'algoritmo
329 }
330
331 void algoritmo(vector<vector<int>> graph, int radice, int n_nodi, vector<string> vect_citta) {
332
333     vector<int> distanze(n_nodi), parent(n_nodi);
334     vector<bool> sono_stati_visitati(n_nodi); //flags per i visitati
335     //inizializzo
336     for (int i = 0; i < n_nodi; i++)
337     {
338         distanze[i] = INT_MAX;
339         sono_stati_visitati[i] = false;
340     }
341
342     distanze[radice] = 0;// distanza dalla radice = 0
343     parent[radice] = NON_HA_PARENT;
344
345     for (int i = 0; i < n_nodi - 1; i++)
346     {
347         int nodo_vicino = controllo_dei_nodi_vicini(distanze, sono_stati_visitati, n_nodi);
348         //nodi vicini vengono analizzati, int nodo vicino è il nodo preso in considerazione
349
350         sono_stati_visitati[nodo_vicino] = true; //è stato visitato il nodo vicino
351
352         for (int adiacente = 0; adiacente < n_nodi; adiacente++) //fase di update delle distanze dei n_nodi
              adiacenti
353         {
354             if (!sono_stati_visitati[adiacente]           //se non è nell'array dei visitati "sono_stati_visitati"
355                 && graph[nodo_vicino][adiacente]          //esiste la connessione tra il vicino e l'adiacente
356                 && distanze[nodo_vicino] != INT_MAX       //distanza del nodo vicino non è infinita
357                 && 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
358             {
359                 parent[adiacente] = nodo_vicino; //parent serve per printare il path
360                 distanze[adiacente] = distanze[nodo_vicino] + graph[nodo_vicino][adiacente];
361             }
362         }
363     }
364
365     //mostriamo
366     for (int i = 0; i < n_nodi; i++) {
367         if (i != radice)
368         {
369             cout << endl;
370             cout << "Nodo: " << vect_citta[radice] << " -> " << vect_citta[i] << "\t\tDistanza: " << distanze[i] <<
                  "\t\tPercorso: ";
371             mostra_path(i, parent, vect_citta);
372         }
373     }
374 }
375
376 void mostra_path(int vertice_corrente, vector<int> parent, vector<string>vect_citta) {
377     if (vertice_corrente == NON_HA_PARENT) {
378         return;
379     }
380     mostra_path(parent[vertice_corrente], parent, vect_citta);
381     cout << vect_citta[vertice_corrente] << " ";
382 }
383
384 bool risp_main() {
385     string risp;
386     cout << "Continuare? ";
387     cin >> risp;
388
389     for (int u = 0; u < risp.size(); u++)
390     {
391         risp[u] = tolower(risp[u]);
392     }
```

```
393
394     if (risp == "si")
395     {
396         return true;
397     }
398     else
399     {
400         return false;
401     }
402 }
403
404 void dijkstra() {
405     int n_nodi = prendi_n_nodi();
406     vector<string> vect_citta = crea_vect_citta(n_nodi);
407     vector<vector<int>> graph = costruisci_graph(n_nodi, vect_citta);
408     int source = prendi_source(n_nodi, vect_citta);
409     algoritmo(graph, source, n_nodi, vect_citta);
410 }
411
412
413
```