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

void mostra_path(int vertice_corrente, vector<int> parent) {
    if (vertice_corrente == NON_HA_PARENT) {
        return;
    }
    mostra_path(parent[vertice_corrente], parent);
    cout << vertice_corrente << " ";
}

int controllo_dei_nodi_vicini(vector<int> distanze, vector<bool> sono_stati_visitati, int vertici) {

    int valoremin = INT_MAX;
    int nodominimo = 0;

    for (int i = 0; i < vertici; i++)
    {
        if (!sono_stati_visitati[i] && distanze[i] <= valoremin) // viene preso e aggiunto il nodo con la
                                                                 //  minima distanza che non è nel vettore bool dei visitati
        {
            valoremin = distanze[i];
            nodominimo = i;
        }
    }
    return nodominimo; //viene dato all'algoritmo
}

void mostra(vector<int> distanze, int vertici,int radice, vector<int> parent)
{
    cout << "Nodo:\t\t\tDistanza:\t\t\tPercorso più veloce:";
    for (int i = 0; i < vertici; i++) {
        if ( i != radice )
        {
            cout << endl;
            cout << radice << " -> " << i << "\t\t\t" << distanze[i] << "\t\t\t\t";
            mostra_path(i, parent);
        }
    }
}

void algoritmo(vector<vector<int>> graph, int radice, int vertici) {

    vector<int> distanze(vertici);
```

```
48      vector<bool> sono_stati_visitati(vertici); //flags per i visitati
49      vector<int> parent(vertici);
50
51      //inizializzo tutto
52      for (int i = 0; i < vertici; i++)
53      {
54          distanze[i] = INT_MAX;
55          sono_stati_visitati[i] = false;
56      }
57
58      distanze[radice] = 0;// distanza dalla radice = 0
59      parent[radice] = NON_HA_PARENT;
60
61      for (int i = 0; i < vertici - 1 ; i++)
62      {
63          int nodo_vicino = controllo_dei_nodi_vicini(distanze,
              sono_stati_visitati, vertici);
64          //nodi vicini vengono analizzati, int nodo vicino è il nodo
              preso in considerazione
65
66          sono_stati_visitati[nodo_vicino] = true; //è stato visitato il
              nodo vicino
67
68          for (int adiacente = 0; adiacente < vertici; adiacente++) //
              fase di update delle distanze dei vertici adiacenti
69          {
70              if (!sono_stati_visitati[adiacente]           //se non è
                  nell'array dei visitati "sono_stati_visitati"
71                  && graph[nodo_vicino][adiacente]          //esiste la
                      connessione tra il vicino e l'adiacente
72                  && distanze[nodo_vicino] != INT_MAX       //distanza
                      del nodo vicino non è infinita
73                  &&  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
74                  ) {
75                  parent[adiacente] = nodo_vicino;
76                  distanze[adiacente] = distanze[nodo_vicino] + graph
                      [nodo_vicino][adiacente];
77
78              }
79          }
80      }
81      mostra(distanze, vertici, radice, parent);
82  }
83
84  int main()
85  {
86      int vertici = 0;
87      int radice = 0;
88      int x = 0, y = 0;
89
```

```
90
91      cout << "Inserisci il numero di nodi: ";
92      cin >> vertici;
93
94
95      vector<vector<int>> graph;
96
97
98      cout << "Riempiamo il graph." << endl;
99
100     for (int i = 0; i < vertici; i++)
101     {
102         vector<int> v1;
103
104         cout << "Pesi dal nodo " << i << " agli altri nodi. " << endl;
105
106         for (int j = 0; j < vertici; j++) {
107             cout << "Nodo " << j << ": ";
108             cin >> x;
109             v1.push_back(x);
110         }
111
112         graph.push_back(v1);
113
114     }
115
116     cout << "La matrice associata al graph: " << endl;
117
118     for (int k = 0; k < vertici ; k++)
119     {
120         for (int l = 0; l < vertici; l++) {
121             cout << graph[k][l]<< "  ";
122         }
123         cout << endl;
124     }
125
126         cout << "Inserisci il nodo radice: ";
127         cin >> radice;
128         cout << endl << endl;
129
130     algoritmo(graph, radice , vertici);
131
132     return 0;
133 }
134
```