```
-----------------------------------------------------------------
graph_wgt.cpp: basic code for reading and storing weighted graph
-----------------------------------------------------------------

#include <...>
using namespace std;

template <typename Tkey, typename Twgt>
class graph {
  public:
    graph() {}
    void read(const char *);
    void print();

  private:
    enum { WGT_UNDIRECTED, WGT_DIRECTED } graph_type;

    vector< Tkey > V;          // vertex list
    vector< vector<int> > E;   // edge matrix
    vector< vector<Twgt> > W;  // weight matrix
    map<Tkey,int> key_map;     // key-to-index map
};

template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::read(const char *fname) {
  ifstream in(fname);

  // Determine graph type: WGT_UNDIRECTED or WGT_DIRECTED
  see graph1_handout for similar code for unweighted graphs

  // Create mapping from key to index

  Tkey key1, key2;
  Twgt wgt;

  vector< pair<int,int> > Eij;
  vector< Twgt > Wij;

  while (in >> key1 >> key2 >> wgt) {
    key_map.insert(make_pair(key1, key_map.size()));
    key_map.insert(make_pair(key2, key_map.size()));

    Eij.push_back(make_pair(key_map[key1], key_map[key2]));
    Wij.push_back(wgt);
  }

  in.close();

  // Create vertex list and edge matrix

  V.resize(key_map.size());
  E.resize(key_map.size());
  W.resize(key_map.size());

  typename map<Tkey,int>::iterator kmp;
  for (kmp=key_map.begin(); kmp!=key_map.end(); ++kmp)
    V[kmp->second] = kmp->first;

  vector< map<int,Twgt> > EW;
  EW.resize(key_map.size());

  for (int k=0; k<(int)Eij.size(); k++) {
    int i = Eij[k].first;
    int j = Eij[k].second;
    Twgt w = Wij[k];
    EW[i].insert(make_pair(j,w));
    if (graph_type == WGT_UNDIRECTED)
      EW[j].insert(make_pair(i,w));
  }

  typename map<int,Twgt>::iterator p;
  for (int i=0; i<(int)EW.size(); i++) {
    for (p=EW[i].begin(); p!=EW[i].end(); ++p) {
      E[i].push_back(p->first);
      W[i].push_back(p->second);
    }
  }
}

template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::print() { ... }

int main(int argc, char *argv[]) {
  if (argc != 2)
    return 0;

  graph<string,int> G;

  G.read(argv[1]);
  G.print();
}

-----------------------------------------------------------------
Hint: Note use of map to ensure unique (index,Twgt) EW listings
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
graph_wgtroute.cpp: compute dijkstra-route from source to sink
-----------------------------------------------------------------

#include <...>
using namespace std;

template <typename Tkey, typename Twgt>
class graph {

  see previous page for basic weigthed graph definitions

  public:
    void dijkstra_route(Tkey &, Tkey &);

  private:
    void dijkstra_route(int, int);
    void show_route(int, int);

    typedef enum { WHITE, BLACK } vcolor_t;
    vector<vcolor_t> vcolor;
    vector<Twgt> vdist;
    vector<int> vlink;
};

template <typename Tkey, typename Twgt>
void
graph<Tkey,Twgt>::dijkstra_route(Tkey &source_key, Tkey &sink_key) {
  modified version of similar bfs_distance() function
}

template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::show_route(int source, int sink) { ... }

int main(int argc, char *argv[]) {
  if (argc != 4) {
    cerr << "usage: " << argv[0]
         << " source sink graph.txt\n";
    return 0;
  }

  string source = argv[1];
  string sink = argv[2];

  graph<string,int> G;

  G.read(argv[3]);
  G.dijkstra_route(source, sink);
}
```

```
template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::dijkstra_route(int source, int sink) {
  vcolor.assign(V.size(), WHITE);

  vdist.assign(V.size(), numeric_limits<Twgt>::max());
  vdist[source] = 0;

  vlink.assign(V.size(), -1);
  vlink[source] = source;

  while (1) {
    int next_i = -1;
    Twgt mindist = numeric_limits<Twgt>::max();

    for (int i=0; i<(int)vcolor.size(); i++) {
      if (vcolor[i] == WHITE && mindist > vdist[i]) {
        next_i = i;
        mindist = vdist[i];
      }
    }

    int i = next_i;
    if (i == -1)
      return;

    vcolor[i] = BLACK;

    if (i == sink)
      break;

    for (int k=0; k<(int)E[i].size(); k++) {
      int j = E[i][k];
      Twgt wij = W[i][k];
      if (vcolor[j] == WHITE) {
        if (vdist[j] > vdist[i] + wij) {
          vdist[j] = vdist[i] + wij;
          vlink[j] = i;
        }
      }
    }
  }
}

-----------------------------------------------------------------
Hint: Note how Dijkstra's algorithm can undo earlier decisions
-----------------------------------------------------------------
```