

```

#include <iostream>
#include <vector>
#include <map>
#include <list>
#include <algorithm>
#include <deque>
#include <fstream>

using namespace std;
class Edge {
public:
    class Node *to;
    class Node *from;
    double weight;
};

//node class
class Node {
public:
    Node();
    char id; //name of node
    multimap <double, Edge*> edges; //edge list
    list <Edge*> ledges;
    Edge *backlink; //sets all backlinks to null
    double distance; //sets all distances to -1
};

Node::Node() {
    backlink = NULL;
    distance = -1;
}

//graph class
class Graph {
public:
    map <char, Node *> graph; //graph vector
    vector <Node *> v;
    //~Graph(); //destructor to delete all nodes
    multimap <double, Node*> Dmap;
    double Dijkstra(Node *start, Node *end);
    deque <Node*> path;
};

double Graph::Dijkstra(Node *start, Node *end) {
    list <Edge*>::const_iterator lit;
    Dmap.insert(make_pair(0, start));
    start->distance = 0;
    double d;
    Node *n;
    Edge *e;
    double dist;

    while (!Dmap.empty()) {
        n = Dmap.begin()->second;
        dist = Dmap.begin()->first;
        Dmap.erase(Dmap.begin()->first);

        for (lit = n->ledges.begin(); lit != n->ledges.end(); lit++) {
            e = *lit;
            d = e->weight + dist;

            if (e->to->distance == -1 || d < e->to->distance) {
                if (Dmap.find(e->to->distance) != Dmap.end())
                    Dmap.erase(e->to->id);

                e->to->distance = d;
                e->to->backlink = e;
                Dmap.insert(make_pair(e->to->distance, e->to));
            }
        }
    }
}

```

```

    n = end;
    while (n != start) {
        path.push_front(n);
        n = n->backlink->from;
    }

    return 0;
}

int main(int argc, char *argv[]) {

    Graph g;
    map<char, Node*>::const_iterator mit;
    map<double, Edge*>::const_iterator eit;
    int total = 0;
    char from, to, start, end;
    double weight;
    int i;
    Node *s;
    Node *e;
    ifstream fin;

    if (argc != 2) return -1;

    fin.open(argv[1]);

    //get total number of nodes
    fin >> total;

    for (i = 0; i < total; i++) {
        Node *n = new Node;
        n->id = 'A' + i;
        g.graph.insert(make_pair(n->id, n));
        g.v.push_back(n);
    }

    while (fin >> from >> to >> weight) {
        Node *n1 = g.graph.find(from)->second;
        Node *n2 = g.graph.find(to)->second;
        Edge *e = new Edge;
        e->weight = weight;
        e->from = n1;
        e->to = n2;
        n1->edges.insert(make_pair(weight, e));
        n1->ledges.push_back(e);
    }

    for (mit = g.graph.begin(); mit != g.graph.end(); mit++) {
        Node *n = mit->second;

        for (eit = n->edges.begin(); eit != n->edges.end(); eit++) {
        }
    }

    printf("Starting node: ");
    cin >> start;
    printf("Ending node: ");
    cin >> end;

    s = g.graph.find(start)->second;
    e = g.graph.find(end)->second;

    g.Dijkstra(s, e);

    printf("Path: ");

    printf("%c ", g.graph.find(start)->first);

    for (i = 0; (size_t) i < g.path.size(); i++) {
        printf("%c ", g.path[i]->id);
    }
}

```

```
    }  
    printf("\nDistance: %.2lf\n", g.path[g.path.size()-1]->distance);  
  
    return 0;  
}
```