CSCI 255: Lab #9 Graph Shortest Path

Darwin Jacob Groskleg

Tuesday, November 12th, 2019

Contents

Contents	i
Console Output	1
main.cpp	2
graph.hpp	6
graph.cpp	7

Console Output

Compiles and Runs with Input: 16

```
See main.cpp for easier to read output.
                                                                    Labelled Nodes:

1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)

4(d:3, p:3) 5(d:5, p:3) 6(d:2147483647, p:-1)

7(d:2147483647, p:-1) 8(d:2147483647, p:-1)

10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)

13(d:2147483647, p:-1) 11(d:2147483647, p:-1) 15(d:2147483647, p:-1)

16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)

19(d:2147483647, p:-1) 17(d:2147483647, p:-1) 12(d:2147483647, p:-1)

22(d:2147483647, p:-1) 26(d:2147483647, p:-1) 24(d:2147483647, p:-1)

25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)

28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 3(d:2147483647, p:-1)

28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 3(d:2147483647, p:-1)
                                                                                                              22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1) 25(d:2147483647, p:-1) 25(d:2147483647, p:-1) 30(d:2147483647, p:-1) 25(d:2147483647, p:-1) 30(d:2147483647, p:-1) 25(d:2147483647, p:-1) 30(d:2147483647, p:-1) 32(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1) 36(d:2147483647, p:-1) 36(d:2147483647, p:-1) 36(d:2147483647, p:-1) 36(d:2147483647, p:-1)
```

main.cpp

```
/* main.cpp (TestGraph.cpp)
1
2
    * Authors: Darwin Jacob Groskleg, Man Lin
3
4
    * Purpose: to rewrite the Best_Path method on Weighted_Graph to show the
5
                detailed steps of the shortest path algorithm.
6
7
    * CONSOLE SAMPLE
8
9
   This is the Weighted Graph Demo
10
11
                2
                      3
                                 5
           1
                            4
                                       6
12
      1
           0
                0
                      1
                            0
                                 0
                                       0
13
      2
           3
                0
                      5
                            0
                                 0
14
      3
           0
                0
                      0
                            2
                                 4
                                       a
15
           5
      4
                0
                      0
                            0
                                 0
                                       5
16
      5
           0
                            0
                0
                      0
                                 0
                                       1
17
      6
           0
                            0
                                       0
                0
                      0
                                 0
18
19
20
   Start: 1
21
   Destination: 6
22
   Adding node 1 to the solved set S={1 }
   Adding node 3 to the solved set S={1 3 }
   Updating labels for node 4
25
       distance: 2147483647 -> 3
26
       parent:
                 -1 -> 3
27
       Node labels:
28
            1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
29
            4(d:3, p:3) 5(d:2147483647, p:-1) 6(d:2147483647, p:-1)
30
            7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
31
            10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
32
            13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
33
            16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
34
            19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
35
            22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
36
            25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
37
            28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
38
            31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1) 34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
39
40
            37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
41
            40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
42
            43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
43
            46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
44
45
            49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
            52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
46
            55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
47
            58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
48
            61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
49
            64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
50
            67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
51
            70(d:2147483647,\ p:-1)\ 71(d:2147483647,\ p:-1)\ 72(d:2147483647,\ p:-1)\\ 73(d:2147483647,\ p:-1)\ 74(d:2147483647,\ p:-1)\ 75(d:2147483647,\ p:-1)
52
53
            76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
54
```

```
79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
55
           82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
56
           85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
57
           88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
58
           91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
           94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
           97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
61
           100(d:2147483647, p:-1)
62
   Updating labels for node 5
63
       distance: 2147483647 -> 5
64
                 -1 -> 3
       parent:
65
       Node labels:
66
           1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
67
            4(d:3, p:3) 5(d:5, p:3) 6(d:2147483647, p:-1)
68
            7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
69
            10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
70
            13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
71
            16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
72
            19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
            22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
74
            25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
75
           28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
76
           31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
77
           34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
78
           37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
79
           40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
80
           43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
81
           46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
82
           49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
83
           52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
84
           55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
85
           58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
           61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
87
            64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
88
            67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
89
            70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
90
            73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
91
            76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
92
            79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
93
           82(d:2147483647,\ p:-1)\ 83(d:2147483647,\ p:-1)\ 84(d:2147483647,\ p:-1)
94
           85(d:2147483647,\ p:-1)\ 86(d:2147483647,\ p:-1)\ 87(d:2147483647,\ p:-1)
95
           88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
96
           91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
97
           94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
98
           97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
99
           100(d:2147483647, p:-1)
100
101
   Adding node 4 to the solved set S=\{1 \ 3 \ 4 \ \}
   Updating labels for node 6
102
       distance: 2147483647 -> 8
103
       parent:
                 -1 \rightarrow 4
104
       Node labels:
105
            1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
106
            4(d:3, p:3) 5(d:5, p:3) 6(d:8, p:4)
107
            7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
108
           10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
109
           13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
```

```
16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
111
            19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
112
            22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
113
            25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
114
            28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
115
            31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
            34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
117
            37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
118
            40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
119
            43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
120
            46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
121
            49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
122
            52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
123
            55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
124
            58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
125
            61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
126
            64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
127
            67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
128
            70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
129
            73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
130
            76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
131
            79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
132
            82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
133
            85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
134
            88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
135
            91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
136
            94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
137
            97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
138
            100(d:2147483647, p:-1)
139
   Adding node 5 to the solved set S={1 3 4 5 }
140
   Updating labels for node 6
141
       distance: 8 -> 6
142
                  4 -> 5
       parent:
143
       Node labels:
144
            1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
145
            4(d:3, p:3) 5(d:5, p:3) 6(d:6, p:5)
146
            7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
147
            10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
148
            13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
149
            16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
150
            19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
151
            22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
152
            25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
153
            28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
154
            31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
155
            34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
156
            37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
157
            40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
158
            43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
159
            46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
160
            49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
161
            52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
162
            55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
163
            58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
164
            61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
165
            64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
166
```

```
67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
167
            70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
168
            73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
169
            76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
170
            79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
171
            82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
            85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
173
            88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
174
            91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
175
            94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
176
            97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
177
            100(d:2147483647, p:-1)
178
    Adding node 6 to the solved set S={1 3 4 5 6 }
179
    shortest distance from 1 To 6 is 6
180
    Showing best path:
181
182
   3
183
   5
184
   6
185
186
187
    * END OF CONSOLE SAMPLE
188
189
    #include <iostream>
190
    #include "graph.hpp"
191
192
    using namespace std;
193
194
    int main (void) {
195
        Weighted_Graph graph(6); // SAY NO TO GLOBALS!
196
197
        cerr << "This is the Weighted Graph Demo\n\n";</pre>
198
199
        graph.Add_Edge(2, 1, 3);
200
        graph.Add_Edge(2, 3, 5);
201
        graph.Add_Edge(3, 4, 2);
202
        graph.Add_Edge(4, 1, 5);
203
        graph.Add_Edge(1, 3, 1);
204
        graph.Add_Edge(3, 5, 4);
205
        graph.Add_Edge(5, 6, 1);
206
        graph.Add\_Edge(4, 6, 5);
207
208
        graph.Display();
209
210
        int start;
211
        int dest;
212
213
        clog << "\n\n" << "Start: ";
214
        cin >> start;
215
        clog << "Destination: ";</pre>
216
        cin >> dest;
217
        cout << '\n';
        graph.Show_Best_Path(start, dest);
220
        return 0;
221
_{222} \mid \}
```

graph.hpp

```
1 /* graph.hpp (Graph.h)
2
   * Authors: Darwin Jacob Groskleg, Man Lin
3
4
  #ifndef GRAPH_HPP_INCLUDED
   #define GRAPH_HPP_INCLUDED
   #include <deque>
8
   #ifndef MAX_NODES
10
   #define MAX_NODES 100
11
   #endif
12
13
   //modify from Turner's Weighted_Graph_demo
14
   class Weighted_Graph
15
   {
16
     public:
17
       Weighted_Graph(int node_count);
18
19
       //add an edge to the graph
20
       void Add_Edge(const int Node_1, const int Node_2, int edge_weight);
21
22
       // Print the ajacency matrix to stdout
^{23}
       void Display() const;
24
25
       //find the best path from start to dest
26
       auto Best_Path(const int Start, const int Dest, bool verbose=false) const
27
           -> std::deque<int>;
28
29
       // Prints the shortest path between two nodes to stderr/console
30
       void Show_Best_Path(int Start, int Dest) const;
31
32
     private:
33
       int number_of_nodes; //the number of vertexes in the graph
34
       // The Node ID 0 is not used. The first real node has ID 1
35
       int weight[MAX_NODES+1] [MAX_NODES+1]; //store the weight of the edges
36
37
       bool validVertex(int node) const;
38
   };
39
40
   #endif // GRAPH_HPP_INCLUDED
```

graph.cpp

```
1 /* graph.cpp
2
    * Authors: Darwin Jacob Groskleg
3
4
    * IMPLEMENTATION of Weighted_Graph
6
   #include "graph.hpp"
7
   #include <iostream>
9
   #include <iomanip>
10
   #include <algorithm>
11
12
   #include <climits> // INT_MAX
13
14
   using namespace std;
15
16
   // parameterized constructor
17
       Weighted_Graph::
18
   Weighted_Graph(int node_count) :
19
       number_of_nodes(node_count)
20
   {
21
       if (node_count > MAX_NODES)
22
            throw "Fail to Construct: node_count exceeds MAX_NODES!";
^{23}
   }
24
25
   //add an edge to the graph, update the vertex and edge structure
26
       void Weighted Graph::
27
   Add_Edge(const int Node_1, const int Node_2, int edge_weight)
28
29
       if (!validVertex(Node_1) || !validVertex(Node_2)) throw "Invalid node!";
30
       weight[Node_1] [Node_2] = edge_weight;
31
   }
32
33
   //display the weight between the vertexes
34
       void Weighted_Graph::
35
   Display() const
36
37
       cout << setw(4) << " ";
38
       for (int i = 1; i <= number_of_nodes; ++i) {</pre>
39
            cout << setw(4) << i << " ";
40
41
       cout << '\n';
42
43
       for (int i = 1; i <= number_of_nodes; ++i) {</pre>
44
            cout << setw(4) << i;
45
            for (int j = 1; j <= number_of_nodes; ++j) {</pre>
46
                cout << setw(4) << weight[i][j] << " ";</pre>
47
48
            cout << '\n';
49
       }
50
   }
51
52
   // Find Shortest path between Start and Dest
53
54 // Approach: use Dijkstra's algorithm, greedy
```

```
deque<int> Weighted_Graph::
55
   Best_Path(const int Start_ID, const int Dest_ID, bool verbose) const
56
57
        deque<int> best_path; //a queue to store the best path
58
        if (!(validVertex(Start_ID) && validVertex(Dest_ID))) {
59
            cout << "Invalid start or destination\n";</pre>
60
            return best_path; // empty deque
61
        }
62
63
        //an array storing the distance label for each vertex
64
        // distance from start node
65
        int distance[MAX_NODES+1];
66
        //an array storing the parent label for each vertex
67
        int parent[MAX_NODES+1];
68
        //a array indicating whether the vertex is already in the solved set
69
        bool is_solved[MAX_NODES+1];
70
71
        for (int i = 1; i <= MAX_NODES; ++i) {</pre>
72
            // if node i is connected to Start node
73
            if (weight[Start_ID][i] > 0) {
74
                distance[i] = weight[Start_ID][i];
75
                parent[i] = Start_ID;
76
            }
77
            else {
78
                distance[i] = INT_MAX;
79
                parent[i] = -1;
80
            }
81
            // want to initialize the array
82
            is_solved[i] = false;
83
84
85
        // HELPER LAMBDAS FOR LOGGED OPERATIONS
86
        auto addToSolved = [&] (int node) {
87
            // Operation 1
88
            is solved[node] = true;
89
            if (verbose) {
90
                clog << "Adding node " << node << " to the solved set S={";</pre>
91
                // Operation 2: show solved set S
92
                // nodes are false if not solved
93
                for (int x=1; x<=MAX_NODES; x++)</pre>
94
                     if (is_solved[x]) clog << x << ' ';</pre>
95
                clog << "}\n";
96
            }
97
        };
98
        auto updateDistanceLabel = [&] (int parent_node, int node) {
99
            // Operation 3: show nodes whose labels are updated and the
100
                             corresponding updated label (distance and parent).
101
            auto new distance = distance[parent node] + weight[parent node][node];
102
            if (verbose)
103
                clog << "Updating labels for node " << node</pre>
104
                     << "\n\tdistance: " << distance[node] << " -> " << new_distance</pre>
105
                     << "\n\tparent: " << parent[node] << " -> " << parent_node</pre>
106
                     << '\n';
107
            distance[node] = new_distance;
108
            parent[node] = parent_node;
109
            // Operation 4: show labels of all the nodes
110
```

```
if (verbose) {
111
                 clog << "\tLabelled Nodes: ";</pre>
112
                 for (int x=1; x<=MAX_NODES; x++) {</pre>
113
                     if (x%3 == 1) clog << "\n\t
114
                     clog << x << "(d:" << distance[x] << ", p:" << parent[x] << ") ";
115
116
                 clog << '\n';
117
            }
118
        };
119
120
        // THE ALGORITHM
121
        distance[Start_ID] = 0;
122
        addToSolved(Start_ID);
123
124
        while (!is_solved[Dest_ID]) {
125
            // Determine the node with least distance among
126
            // all nodes whose best distance is not yet known.
127
            int min_best_dist = INT_MAX;
128
            int best_node_id = -1;
129
            // find the node that is not in the solved set and has the minimal
130
            // distance
131
             for (int i = 1; i <= number_of_nodes; ++i) {</pre>
132
                 // best distance so far?
133
                 if (!is_solved[i] && distance[i] < min_best_dist) {</pre>
134
                     min_best_dist = distance[i];
135
                     best_node_id = i;
136
                 }
137
            }
138
139
            if (best_node_id == -1) {
140
                 // Destination is unreachable.
141
                 cerr << Dest_ID << " is unreachable from " << Start_ID << '\n';</pre>
142
                 return best_path; // empty deque
143
144
145
            // Best total distance so far for this node is the actual
146
            // best total distance.
147
            int v = best_node_id;
148
            addToSolved(v);
149
150
            // if applicable, update the label of the neighbours of the active node
151
            // (that is, the distance and parent)
152
             for (int i = 1; i <= number_of_nodes; ++i) {</pre>
153
                 if (!is_solved[i]
154
                     && weight[v][i] > 0
155
                     && (distance[v] + weight[v][i]) < distance[i]
156
                 ) {
157
                     // It does.
158
                     updateDistanceLabel(v, i);
159
                 }
160
            }
161
        }
162
163
        // At this point we know parent of each node on the
164
        // best path from Start to Dest
165
        clog << "shortest distance from " << Start_ID << " To " << Dest_ID</pre>
166
```

```
<< " is " << distance[Dest_ID] << '\n';
167
        best_path.push_front(Dest_ID);
168
        int next_node_id = Dest_ID;
169
        while (next_node_id != Start_ID) {
170
             next_node_id = parent[next_node_id];
171
             best_path.push_front(next_node_id);
172
173
        return best_path;
174
    }
175
176
177
    void Weighted_Graph::Show_Best_Path(int Start, int Dest) const
178
179
        deque<int> best_path = this->Best_Path(Start, Dest, true);
180
181
        if (best_path.size() == 0) {
182
             clog << "No path found\n";</pre>
183
        }
184
        else {
185
             clog << "Showing best path:\n";</pre>
186
             while (best_path.size() > 0) {
187
                 int next = best_path.front();
188
                 best_path.pop_front();
189
                 cout << next << '\n';</pre>
190
             }
191
192
        clog << '\n';</pre>
193
    }
194
195
196
197
        bool Weighted_Graph::
198
   validVertex(int node) const
199
200
        return node > 0 && node <= number_of_nodes;</pre>
201
202 | }
```