

# CSCI 255: Lab #9 Graph Shortest Path

Darwin Jacob Groskleg

Tuesday, November 12th, 2019

## Contents

<a href="#">Contents</a>	<a href="#">i</a>
<a href="#">Console Output</a>	<a href="#">1</a>
<a href="#">main.cpp</a>	<a href="#">2</a>
<a href="#">graph.hpp</a>	<a href="#">6</a>
<a href="#">graph.cpp</a>	<a href="#">7</a>

See main.cpp for easier to read output.

1

**main.cpp**

```

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

```

```

55      79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
56      82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
57      85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
58      88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
59      91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
60      94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
61      97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
62      100(d:2147483647, p:-1)
63  Updating labels for node 5
64      distance: 2147483647 -> 5
65      parent: -1 -> 3
66      Node labels:
67      1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
68      4(d:3, p:3) 5(d:5, p:3) 6(d:2147483647, p:-1)
69      7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
70      10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
71      13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
72      16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
73      19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
74      22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
75      25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
76      28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
77      31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
78      34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
79      37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
80      40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
81      43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
82      46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
83      49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
84      52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
85      55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
86      58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
87      61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
88      64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
89      67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
90      70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
91      73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
92      76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
93      79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
94      82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
95      85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
96      88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
97      91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
98      94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
99      97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
100     100(d:2147483647, p:-1)
101  Adding node 4 to the solved set S={1 3 4 }
102  Updating labels for node 6
103     distance: 2147483647 -> 8
104     parent: -1 -> 4
105     Node labels:
106     1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
107     4(d:3, p:3) 5(d:5, p:3) 6(d:8, p:4)
108     7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
109     10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
110     13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)

```

```

111 16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
112 19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
113 22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
114 25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
115 28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
116 31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
117 34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
118 37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
119 40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
120 43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
121 46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
122 49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
123 52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
124 55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
125 58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
126 61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
127 64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)
128 67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
129 70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
130 73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
131 76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
132 79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
133 82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
134 85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
135 88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
136 91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
137 94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
138 97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
139 100(d:2147483647, p:-1)
140 Adding node 5 to the solved set S={1 3 4 5 }
141 Updating labels for node 6
142 distance: 8 -> 6
143 parent: 4 -> 5
144 Node labels:
145 1(d:0, p:-1) 2(d:2147483647, p:-1) 3(d:1, p:1)
146 4(d:3, p:3) 5(d:5, p:3) 6(d:6, p:5)
147 7(d:2147483647, p:-1) 8(d:2147483647, p:-1) 9(d:2147483647, p:-1)
148 10(d:2147483647, p:-1) 11(d:2147483647, p:-1) 12(d:2147483647, p:-1)
149 13(d:2147483647, p:-1) 14(d:2147483647, p:-1) 15(d:2147483647, p:-1)
150 16(d:2147483647, p:-1) 17(d:2147483647, p:-1) 18(d:2147483647, p:-1)
151 19(d:2147483647, p:-1) 20(d:2147483647, p:-1) 21(d:2147483647, p:-1)
152 22(d:2147483647, p:-1) 23(d:2147483647, p:-1) 24(d:2147483647, p:-1)
153 25(d:2147483647, p:-1) 26(d:2147483647, p:-1) 27(d:2147483647, p:-1)
154 28(d:2147483647, p:-1) 29(d:2147483647, p:-1) 30(d:2147483647, p:-1)
155 31(d:2147483647, p:-1) 32(d:2147483647, p:-1) 33(d:2147483647, p:-1)
156 34(d:2147483647, p:-1) 35(d:2147483647, p:-1) 36(d:2147483647, p:-1)
157 37(d:2147483647, p:-1) 38(d:2147483647, p:-1) 39(d:2147483647, p:-1)
158 40(d:2147483647, p:-1) 41(d:2147483647, p:-1) 42(d:2147483647, p:-1)
159 43(d:2147483647, p:-1) 44(d:2147483647, p:-1) 45(d:2147483647, p:-1)
160 46(d:2147483647, p:-1) 47(d:2147483647, p:-1) 48(d:2147483647, p:-1)
161 49(d:2147483647, p:-1) 50(d:2147483647, p:-1) 51(d:2147483647, p:-1)
162 52(d:2147483647, p:-1) 53(d:2147483647, p:-1) 54(d:2147483647, p:-1)
163 55(d:2147483647, p:-1) 56(d:2147483647, p:-1) 57(d:2147483647, p:-1)
164 58(d:2147483647, p:-1) 59(d:2147483647, p:-1) 60(d:2147483647, p:-1)
165 61(d:2147483647, p:-1) 62(d:2147483647, p:-1) 63(d:2147483647, p:-1)
166 64(d:2147483647, p:-1) 65(d:2147483647, p:-1) 66(d:2147483647, p:-1)

```

```

167         67(d:2147483647, p:-1) 68(d:2147483647, p:-1) 69(d:2147483647, p:-1)
168         70(d:2147483647, p:-1) 71(d:2147483647, p:-1) 72(d:2147483647, p:-1)
169         73(d:2147483647, p:-1) 74(d:2147483647, p:-1) 75(d:2147483647, p:-1)
170         76(d:2147483647, p:-1) 77(d:2147483647, p:-1) 78(d:2147483647, p:-1)
171         79(d:2147483647, p:-1) 80(d:2147483647, p:-1) 81(d:2147483647, p:-1)
172         82(d:2147483647, p:-1) 83(d:2147483647, p:-1) 84(d:2147483647, p:-1)
173         85(d:2147483647, p:-1) 86(d:2147483647, p:-1) 87(d:2147483647, p:-1)
174         88(d:2147483647, p:-1) 89(d:2147483647, p:-1) 90(d:2147483647, p:-1)
175         91(d:2147483647, p:-1) 92(d:2147483647, p:-1) 93(d:2147483647, p:-1)
176         94(d:2147483647, p:-1) 95(d:2147483647, p:-1) 96(d:2147483647, p:-1)
177         97(d:2147483647, p:-1) 98(d:2147483647, p:-1) 99(d:2147483647, p:-1)
178         100(d:2147483647, p:-1)
179 Adding node 6 to the solved set S={1 3 4 5 6 }
180 shortest distance from 1 To 6 is 6
181 Showing best path:
182 1
183 3
184 5
185 6
186
187 -----
188 * END OF CONSOLE SAMPLE
189 */
190 #include <iostream>
191 #include "graph.hpp"
192
193 using namespace std;
194
195 int main (void) {
196     Weighted_Graph graph(6); // SAY NO TO GLOBALS!
197
198     cerr << "This is the Weighted Graph Demo\n\n";
199
200     graph.Add_Edge(2, 1, 3);
201     graph.Add_Edge(2, 3, 5);
202     graph.Add_Edge(3, 4, 2);
203     graph.Add_Edge(4, 1, 5);
204     graph.Add_Edge(1, 3, 1);
205     graph.Add_Edge(3, 5, 4);
206     graph.Add_Edge(5, 6, 1);
207     graph.Add_Edge(4, 6, 5);
208
209     graph.Display();
210
211     int start;
212     int dest;
213
214     clog << "\n\n" << "Start: ";
215     cin >> start;
216     clog << "Destination: ";
217     cin >> dest;
218     cout << '\n';
219     graph.Show_Best_Path(start, dest);
220
221     return 0;
222 }

```

## graph.hpp

```

1  /* graph.hpp (Graph.h)
2  * -----
3  * Authors: Darwin Jacob Groskleg, Man Lin
4  */
5  #ifndef GRAPH_HPP_INCLUDED
6  #define GRAPH_HPP_INCLUDED
7
8  #include <deque>
9
10 #ifndef MAX_NODES
11 #define MAX_NODES 100
12 #endif
13
14 //modify from Turner's Weighted_Graph_demo
15 class Weighted_Graph
16 {
17     public:
18         Weighted_Graph(int node_count);
19
20         //add an edge to the graph
21         void Add_Edge(const int Node_1, const int Node_2, int edge_weight);
22
23         // Print the adjacency matrix to stdout
24         void Display() const;
25
26         //find the best path from start to dest
27         auto Best_Path(const int Start, const int Dest, bool verbose=false) const
28             -> std::deque<int>;
29
30         // Prints the shortest path between two nodes to stderr/console
31         void Show_Best_Path(int Start, int Dest) const;
32
33     private:
34         int number_of_nodes; //the number of vertexes in the graph
35         // The Node ID 0 is not used. The first real node has ID 1
36         int weight[MAX_NODES+1][MAX_NODES+1]; //store the weight of the edges
37
38         bool validVertex(int node) const;
39 };
40
41 #endif // GRAPH_HPP_INCLUDED

```

## graph.cpp

```

1  /* graph.cpp
2  * -----
3  * Authors: Darwin Jacob Groskleg
4  *
5  * IMPLEMENTATION of Weighted_Graph
6  */
7  #include "graph.hpp"
8
9  #include <iostream>
10 #include <iomanip>
11 #include <algorithm>
12
13 #include <climits> // INT_MAX
14
15 using namespace std;
16
17 // parameterized constructor
18 Weighted_Graph::
19 Weighted_Graph(int node_count) :
20     number_of_nodes(node_count)
21 {
22     if (node_count > MAX_NODES)
23         throw "Fail to Construct: node_count exceeds MAX_NODES!";
24 }
25
26 //add an edge to the graph, update the vertex and edge structure
27 void Weighted_Graph::
28 Add_Edge(const int Node_1, const int Node_2, int edge_weight)
29 {
30     if (!validVertex(Node_1) || !validVertex(Node_2)) throw "Invalid node!";
31     weight[Node_1][Node_2] = edge_weight;
32 }
33
34 //display the weight between the vertexes
35 void Weighted_Graph::
36 Display() const
37 {
38     cout << setw(4) << " ";
39     for (int i = 1; i <= number_of_nodes; ++i) {
40         cout << setw(4) << i << " ";
41     }
42     cout << '\n';
43
44     for (int i = 1; i <= number_of_nodes; ++i) {
45         cout << setw(4) << i;
46         for (int j = 1; j <= number_of_nodes; ++j) {
47             cout << setw(4) << weight[i][j] << " ";
48         }
49         cout << '\n';
50     }
51 }
52
53 // Find Shortest path between Start and Dest
54 // Approach: use Dijkstra's algorithm, greedy

```



```

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

```

```

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

```

```
167         << " is " << distance[Dest_ID] << '\n';
168     best_path.push_front(Dest_ID);
169     int next_node_id = Dest_ID;
170     while (next_node_id != Start_ID) {
171         next_node_id = parent[next_node_id];
172         best_path.push_front(next_node_id);
173     }
174     return best_path;
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";
183     }
184     else {
185         clog << "Showing best path:\n";
186         while (best_path.size() > 0) {
187             int next = best_path.front();
188             best_path.pop_front();
189             cout << next << '\n';
190         }
191     }
192     clog << '\n';
193 }
194
195
196
197 bool Weighted_Graph::
198 validVertex(int node) const
199 {
200     return node > 0 && node <= number_of_nodes;
201 }
202
```