CHAPTER 7
SOLVING SHORTEST PATH PROBLEM USING DIJKSTRA ALGORITHM

7.1 Learning Objectives

1. Students are able to represent graphs with weighted edges
2. Students are able to solve shortest path problem using Dijkstra algorithm

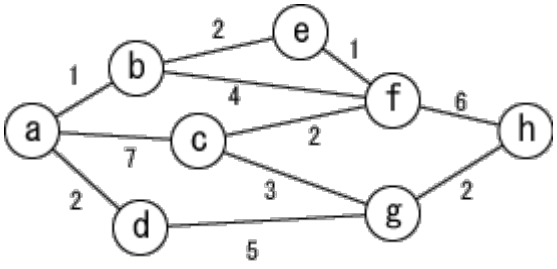
7.2 Material

7.2.1 Dijkstra Algorithm

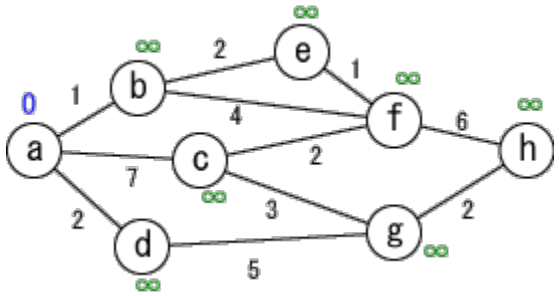
The Dijkstra algorithm is a method in which the shortest path to each node in a graph is determined one by one from the “directly connected” surrounding nodes of the starting node, and the range is gradually expanded. The algorithm works as follow.

1. The distance to each node is assumed to be “unknown” at the beginning and is firstly set to infinity.
2. Set the distance to the starting node to 0.
3. Select a node with the shortest distance among the ones that have not been previously selected and record that distance as the "shortest distance up until this point".
4. Calculate the distance of the nodes that are "directly connected" to the newly selected node in step 3 (passing through this node). If they are shorter than the distances recorded earlier, update the distance.
5. Finish when all nodes have been selected. Otherwise go to 3.

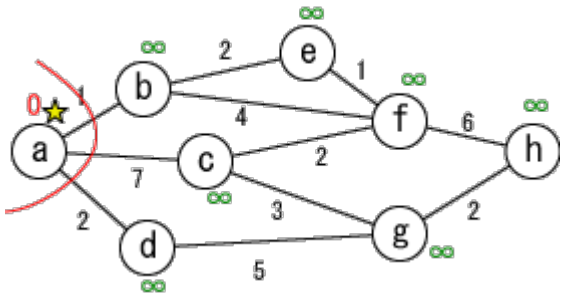
To better understand how this algorithm works in practice, assume the following graph.



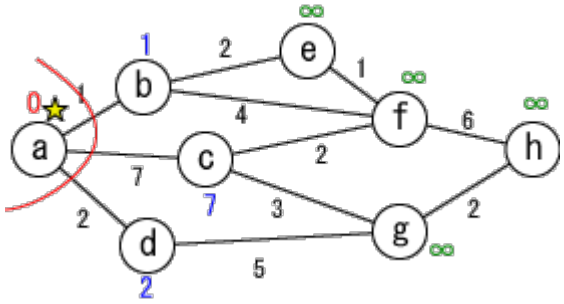
Let's find the shortest distance when the starting node is node-a and the destination node is node h.



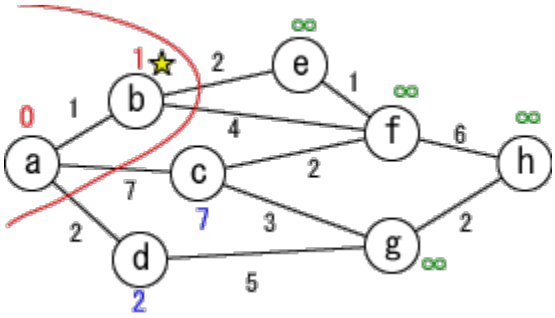
In the illustration figure, the distance to every node in the graph is written above/below each node. At first, these distances are unknown and initially set to infinity. Since node-a is the starting node, the distance to itself is updated to 0, and hence 0 is written above node-a.



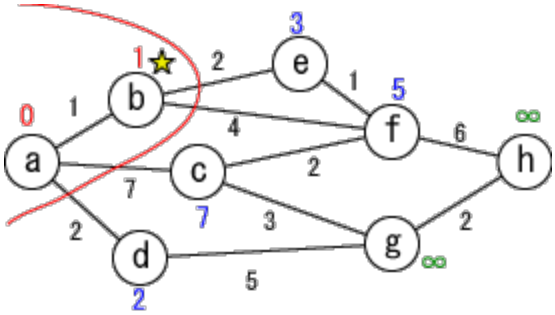
Now, we have to select a node with the lowest distance. Node-a will be selected because at the moment its distance is 0. This distance is then set as the shortest distance up until this point. A red curve is used to mark all the selected nodes (currently only node-a), nodes outside this curve is "not yet selected" (unselected). The newly selected node is marked by ☆.



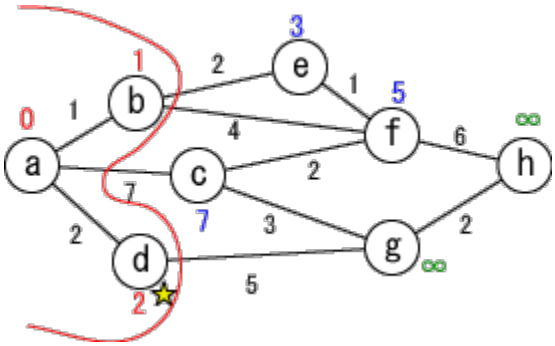
The distances from the starting node (node-a) to the "directly connected" but "unselected" nodes, i.e., nodes-(b, c, d), are calculated and updated if they are smaller than their previous distances. The distances from node-a to nodes-(b, c, d) are updated from ∞ to 1, 7, and 2, respectively.



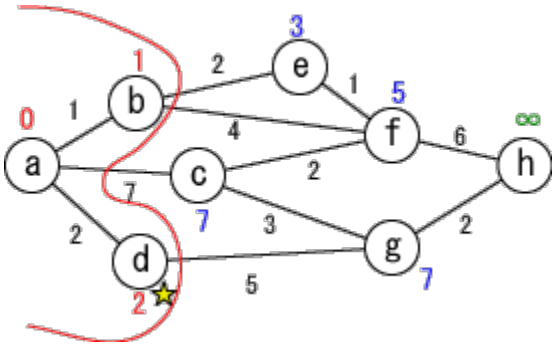
Now, we select node-b among the unselected nodes because it has the shortest distance (1), and set this distance as the minimum distance up until this point. If there are multiple nodes with the same shortest distances, you can select any of them. The newly selected node-b is marked with a star.



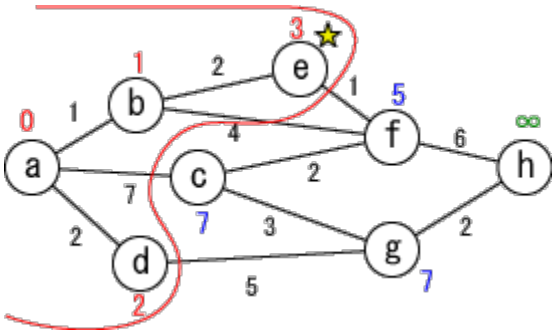
The distances from node-a (via the newly selected node-b) to the "directly connected" nodes to node-b but "unselected" ones, i.e., nodes-(e, f), are calculated and updated if they are smaller than their previous distances. The distances from node-a to nodes-(e, f) are updated from ∞ to 3 and 5, respectively.



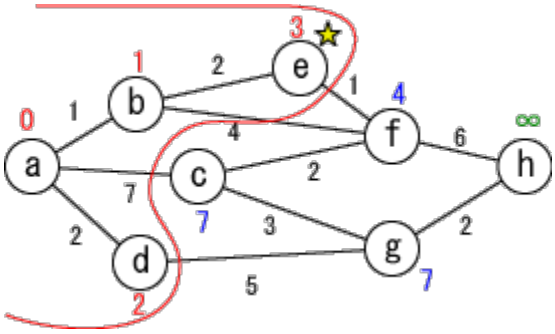
Next, node-d is selected among the unselected nodes because it has the shortest distance (2), and we set this distance as the minimum distance up until this point. The newly selected node-d is marked with a star.



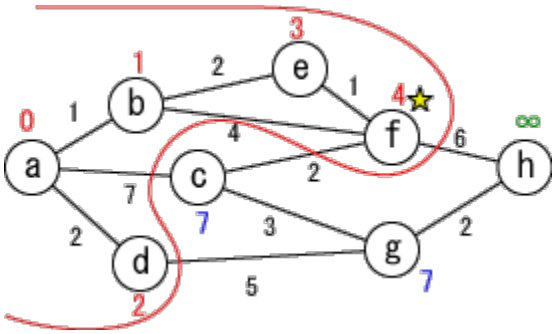
The distance from node-a (via the newly selected node-d) to the "directly connected" node to node-d but "unselected" one, i.e., node-g, is calculated and updated if it is smaller than its previous distance. The distance from node-a to nodes-g is updated from ∞ to 7.

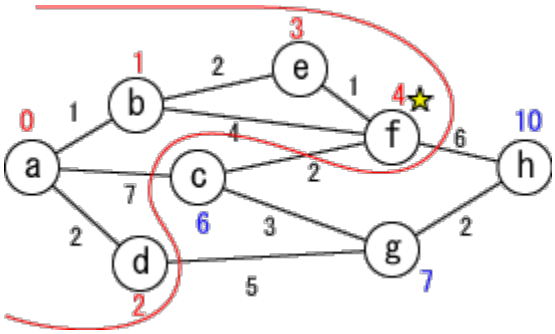


Next, node-e is selected among the unselected nodes because it has the shortest distance (3), and we set this distance as the minimum distance up until this point. The newly selected node-e is marked with a star.

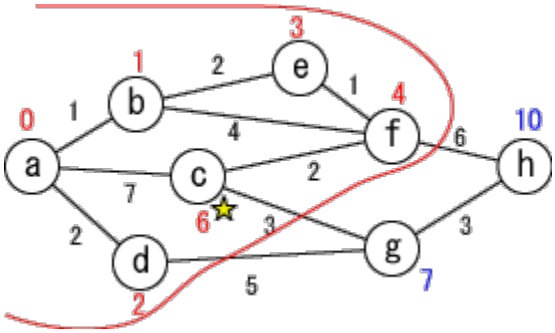


The distance from node-a (via the newly selected node-e) to the "directly connected" node to node-e but "unselected" one, i.e., node-f, is calculated and updated if it is smaller than its previous distance. The distance from node-a to nodes-f is updated from 5 to 4. The process is then continued until all nodes are selected. The graphical illustrations are as follows.

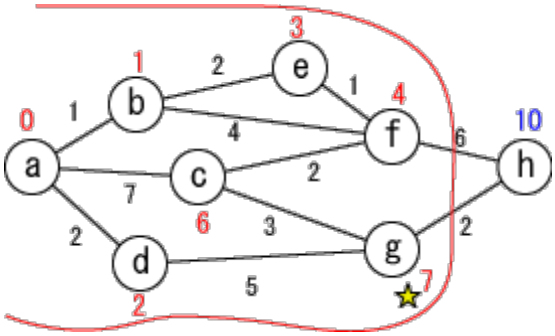


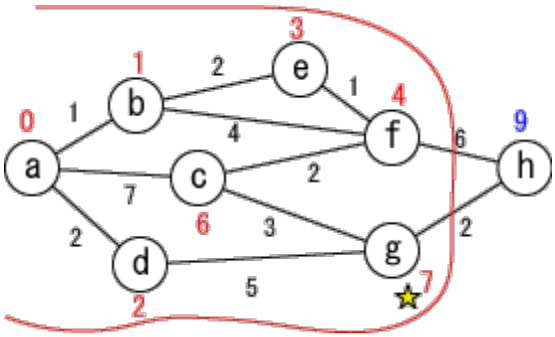


Node-f is selected and nodes-(c, h) are updated.

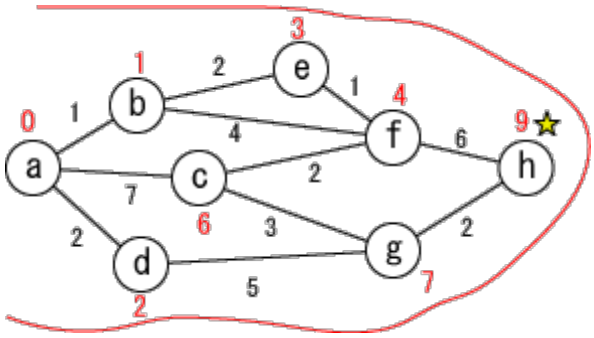


Node-c is selected but none is updated.





Node-g is selected and node-h is updated.



All nodes are selected. Finish! The distance from node-a to node-h is 9.

7.2.2 Java Implementation

The java implementation for calculating the shortest path using Dijkstra algorithm is as shown below. Here, there are two classes: Graph and GraphTest. As their names imply, one is use for graph representation and Dijkstra algorithm's implementation while the other is for testing.

Graph.java

Graph.java GraphTest.java

```
1  public class Graph {
2      private int numVertex;
3      private int[][] adjacencyMatrix;
4
5      public Graph(int numVertex) {
6          this.numVertex = numVertex;
7          adjacencyMatrix = new int[numVertex][numVertex];
8      }
9
10     public void addEdge(int from, int to, int len) {
11         adjacencyMatrix[from][to] = len;
12         adjacencyMatrix[to][from] = len;
13     }
14
15     public void displayGraph() {
16         for (int i=0 ; i<numVertex ; i++) {
17             for (int j=0 ; j<numVertex ; j++) {
18                 System.out.print(adjacencyMatrix[i][j] + " ");
19             }
20             System.out.println();
21         }
22     }
23
```

Graph.java (cont'd)

Graph.java | GraphTest.java

```
24      public void dijkstra(int src, int dst) {
25          int[] distance = new int[numVertex];
26          boolean[] fixed = new boolean[numVertex];
27          for (int i=0; i<numVertex; i++) {
28              distance[i] = Integer.MAX_VALUE;
29              fixed[i] = false;
30          }
31          distance[src] = 0;
32          while (true) {
33              int marked = minIndex(distance, fixed);
34              if (marked < 0) break;
35              if (distance[marked]==Integer.MAX_VALUE) break;
36              fixed[marked] = true;
37              for (int j=0; j<numVertex; j++) {
38                  if (adjacencyMatrix[marked][j]>0 && !fixed[j]) {
39
40                      int newDistance = distance[marked]+adjacencyMatrix[marked][j];
41
42                      if (newDistance < distance[j]) distance[j] = newDistance;
43                  }
44              }
45          }
46          if (distance[dst]==Integer.MAX_VALUE) {
47              System.out.println("no route");
48          } else {
49              System.out.println("distance="+distance[dst]);
50          }
51      }
52
```

Graph.java (cont'd)

Graph.java | GraphTest.java

```
53      public int minIndex(int[] distance, boolean[] fixed) {
54          int idx=0;
55          for (; idx<fixed.length; idx++)
56              if (!fixed[idx]) break;
57          if (idx == fixed.length) return -1;
58          for (int i=idx+1; i<fixed.length; i++)
59              if (!fixed[i] && distance[i]<distance[idx]) idx=i;
60          return idx;
61      }
62  }
```

GraphTest.java

```java
public class GraphTest {
    public static void main(String[] args) {
        int numVertex = 8;
        Graph map = new Graph(numVertex);

        //(A,B,C,D,E,F,G,H) = (0,1,2,3,4,5,6,7)

        /*Create the graph here*/
        map.addEdge(0,1,1);
        map.addEdge(0,2,7);
        map.addEdge(0,3,2);
        map.addEdge(1,4,2);
        map.addEdge(1,5,4);
        map.addEdge(2,5,2);
        map.addEdge(2,6,3);
        map.addEdge(3,6,5);
        map.addEdge(4,5,1);
        map.addEdge(5,7,6);
        map.addEdge(6,7,2);

        map.displayGraph();

        map.dijkstra(0,7);
    }
}
```

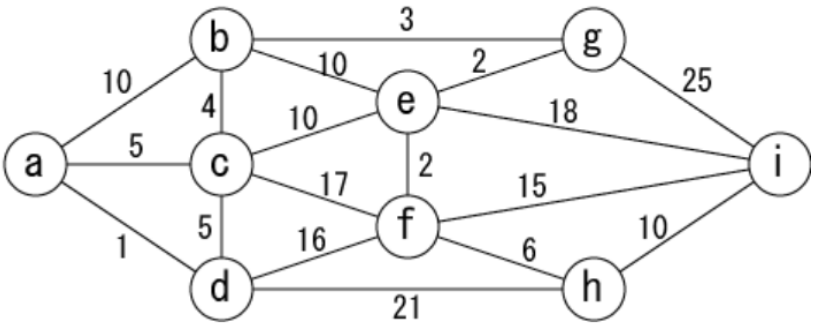
The result after running GraphTest on the graph shown in 7.2.1. is as follows.

```
PS C:¥Users¥alfam¥java> javac Graph.java
PS C:¥Users¥alfam¥java> javac GraphTest.java
PS C:¥Users¥alfam¥java> java GraphTest
0 1 7 2 0 0 0 0
1 0 0 0 2 4 0 0
7 0 0 0 0 2 3 0
2 0 0 0 0 0 5 0
0 2 0 0 0 1 0 0
0 4 2 0 1 0 0 6
0 0 3 5 0 0 0 2
0 0 0 0 0 6 2 0
distance=9
```

Note that adjacency matrix is used as the graph representation in this implementation.

7.3 Assignments

1. For the following graph, calculate the distance of the shortest path when the starting node is node-a and the destination node is node-i. Try calculating it manually (using pen and paper) and match the result with the code implementation. Do you get the same result?



2. Try to fully understand the code and modify it so that it also prints the actual shortest path. For example, the shortest path from A to H (graph at 7.2.1) is via A-D-G-H. Show which parts of the code (and how) did you modify it.