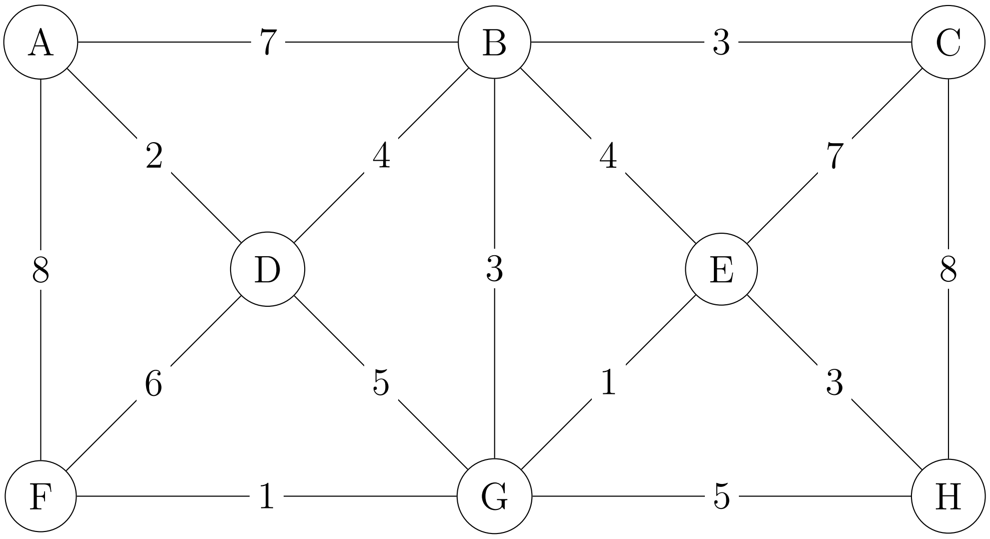
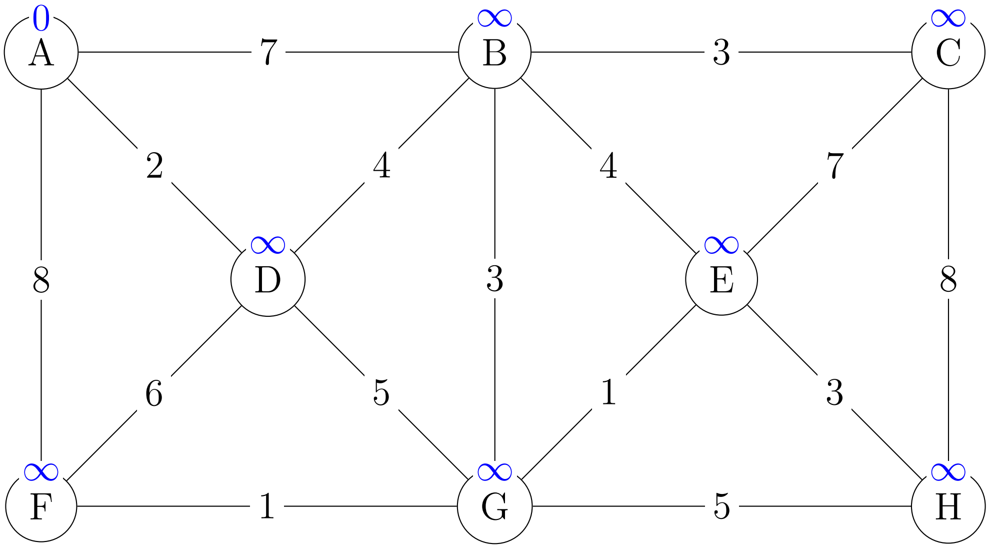
Dijsktra's Algorithm

1. Create two sets of nodes, the first is the set of visited nodes and it will start empty, and the second is the set of unvisited nodes which will initially contain all the nodes.
2. Assign a distance to each node, for now the distance to the start node is 0 and every other node is infinite.
3. Select the node from the unvisited set with the lowest distance to be the current node.
4. For each edge that travels from the current node to an unvisited node, calculate the distance of travelling from the current node to the new node via this edge. If this new distance is less than the current distance to the new node, then replace the distance to the new node.
5. Once all the edges leaving the current node to unvisited nodes have been considered, remove the current node from the unvisited set and add it to the visited set.
6. Repeat steps 3 to 5 until all nodes have been visited. Once this is done you can re-trace the path either by inspection or by recording the previous node alongside the distance to each node.

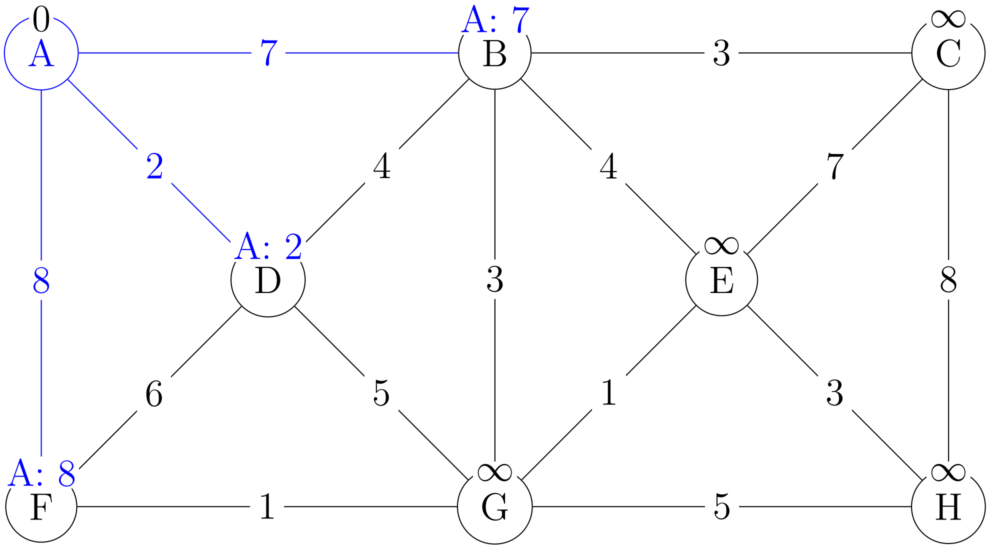
This is quite a lot to take in, so lets work through an example of the algorithm. First consider the graph,



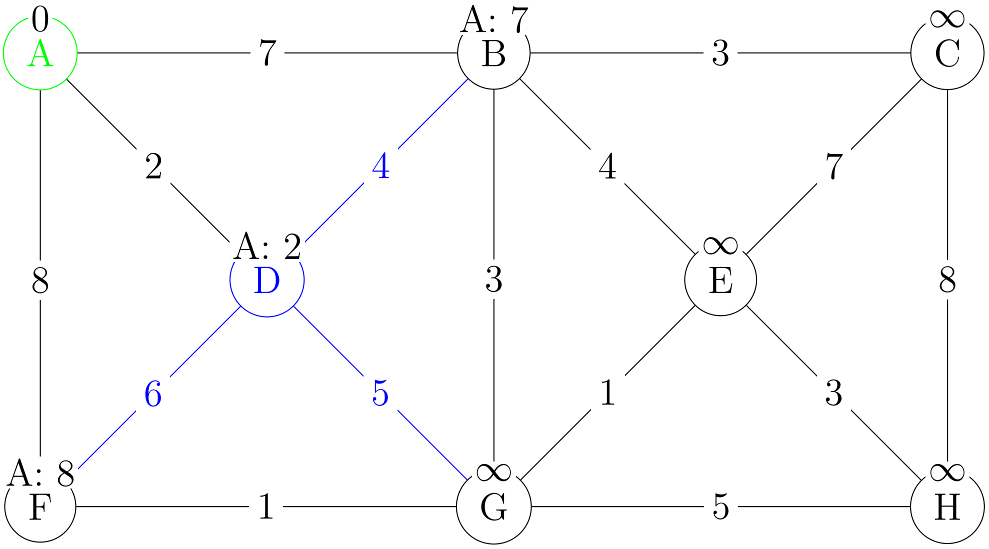
Let's say we want to find the shortest path from A to H. For the first part of the algorithm we will give A a distance of 0—since it's the start node—and every other node a distance of infinity.



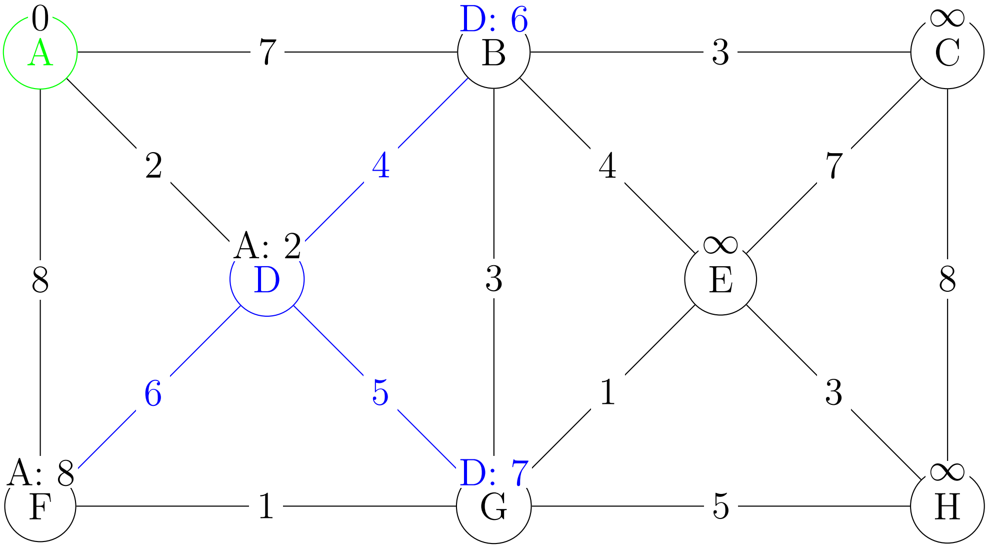
Since all our nodes are unvisited we will choose the node with the lowest distance to be our current node, in this case node A with distance $d_A = 0$. For each edge leaving from A with weight w, we will calculate the sum of $d_A + w$ and if this is smaller than the weight of the destination node we will update that node's distance and previous node to be $d_A + w$ and A. Since currently all the nodes connected to A have a distance of infinity, they will all be updated to this new weight.



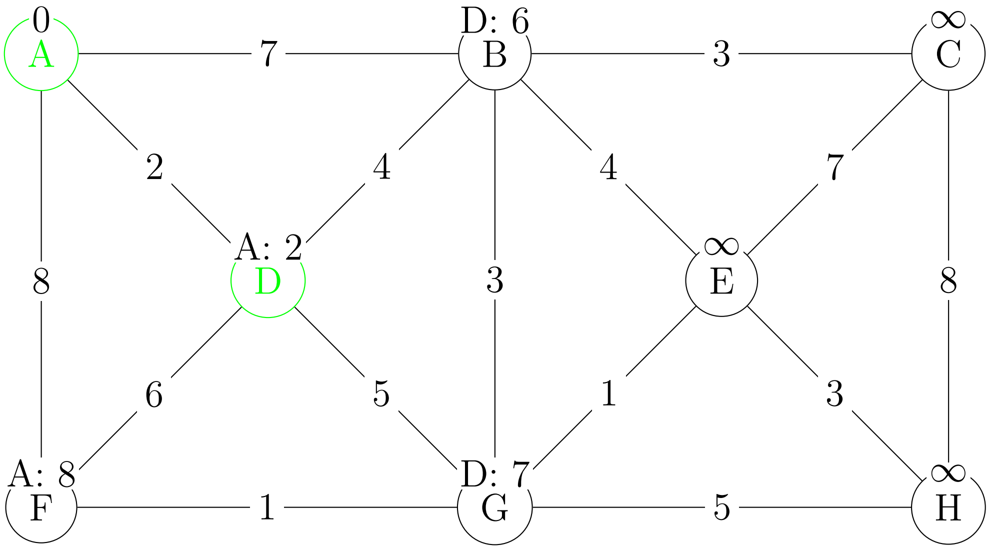
Now we will add A to our list of visited nodes, which we will mark as green. And we will select the unvisited node with the **lowest distance** to be the next current node. In this case the next lowest node is D with $d_D = 2$. Similar to before we will check each edge leaving D to an unvisited node and calculate



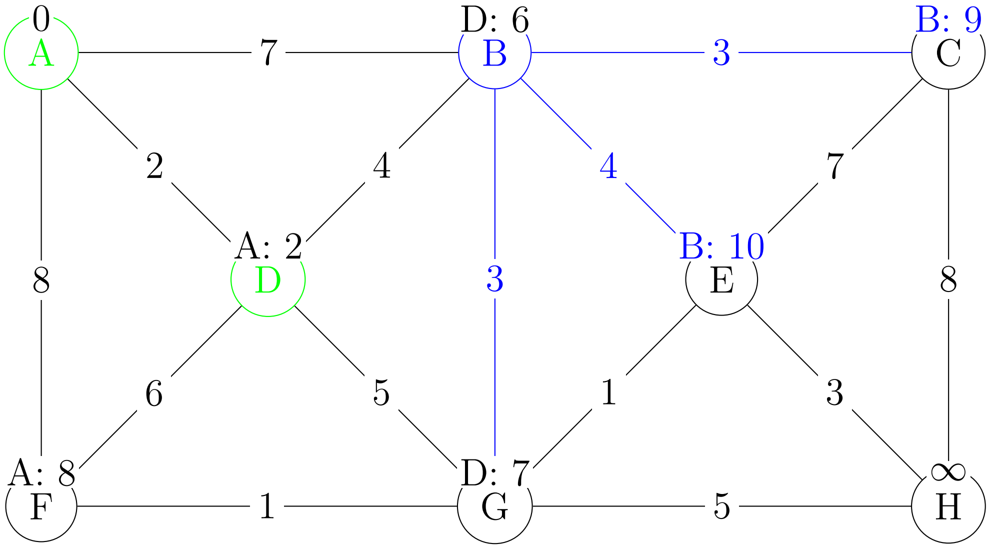
Notice that for the edge from D to B, $d_D + 4$ is 6, which is less than the current distance to B from A, this means that will will update the distance to B with our new best path via D.

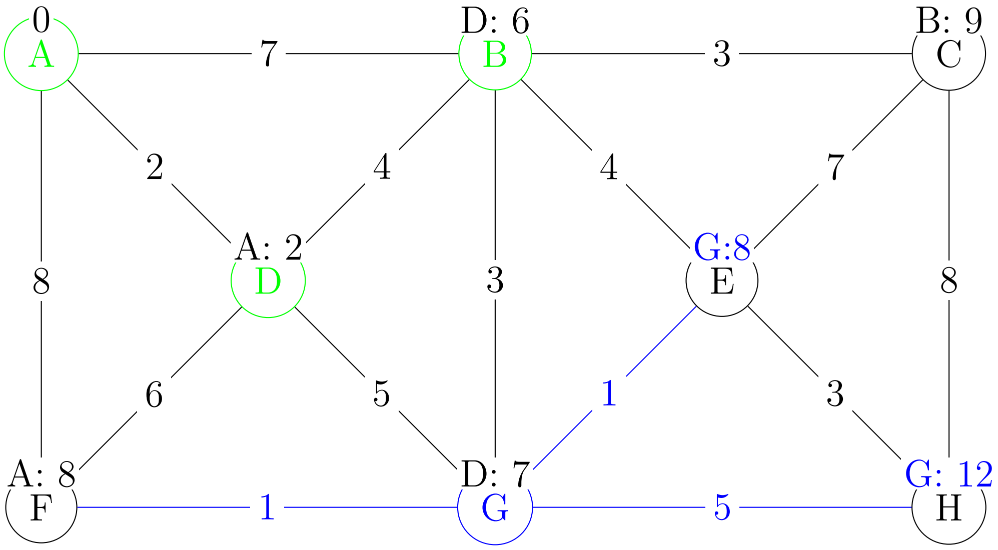


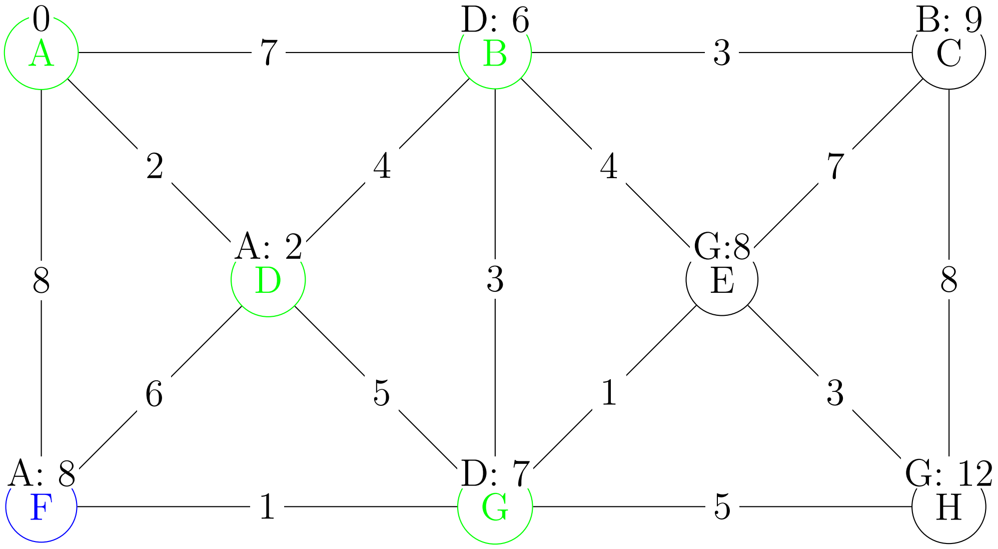
Then we will add node D to the visited set.

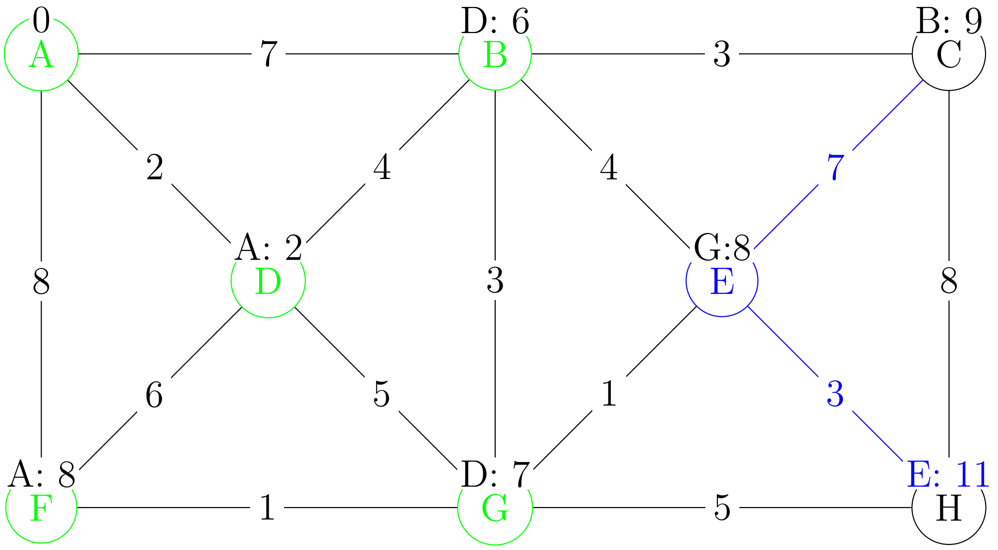


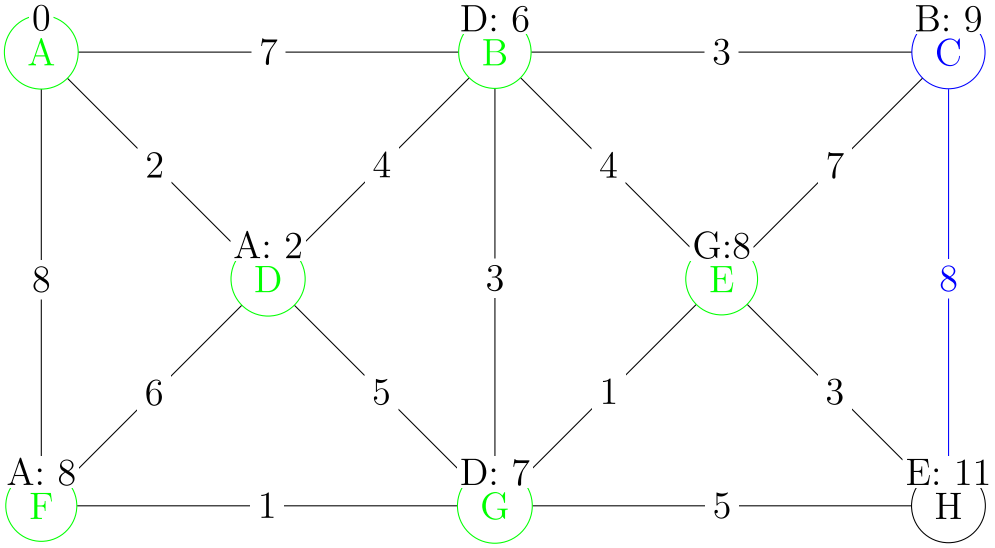
Then we will choose the unvisited node with the lowest distance to be our current node, this time B. I'm not going to narrate the visiting of the remaining nodes, but hopefully you will be able to see the process through the illustrations.

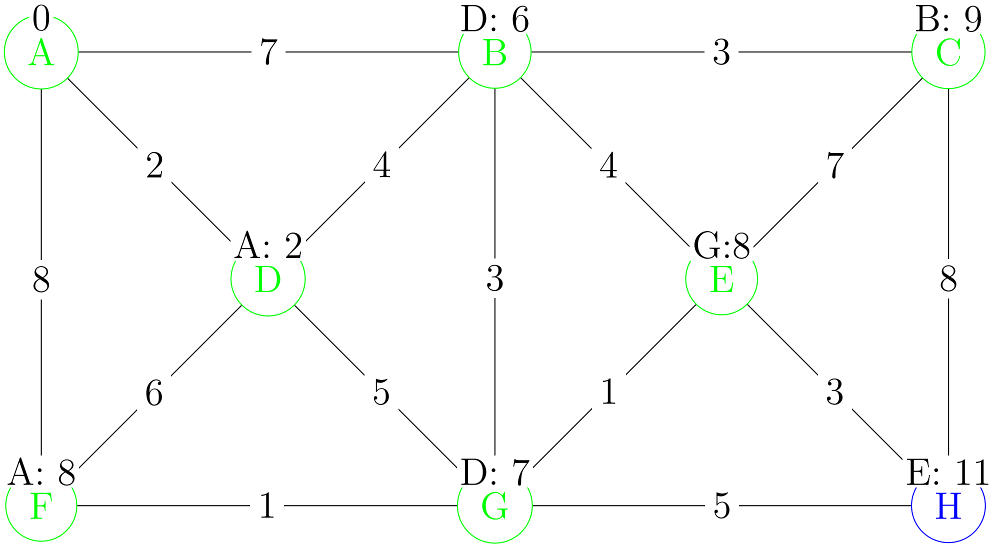


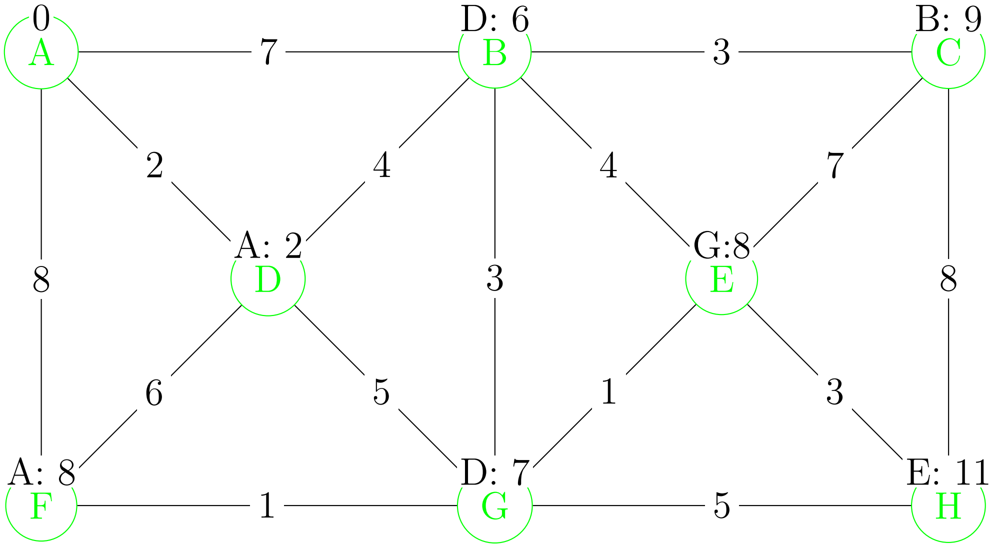

0
A
7
D: 6
B
3
B: 9
C
2
4
4
7
8
A: 2
D
3
G:8
E
8
6
5
1
3
A: 8
F
1
D: 7
G
5
G: 12
H


0
A
7
D: 6
B
3
B: 9
C
2
4
4
7
8
A: 2
D
3
G:8
E
8
6
5
1
3
A: 8
F
1
D: 7
G
5
G: 12
H


0
A
7
D: 6
B
3
B: 9
C
2
4
4
7
8
A: 2
D
3
G:8
E
8
6
5
1
3
A: 8
F
1
D: 7
G
5
E: 11
H







Now that all nodes have been visited, we know that the distance or total weight to get from A to H is 11, and we can trace back the path by following the markers of the previous node. The path in reverse will be H → E → G → D → A, which means that the shortest path from A to H is A → D → G → E → H.