We now develop the algorithm from the invariant (which consists of P1, P2, and P3) and the theorem. The algorithm will be basically a loop. Each iteration of the loop will place one more node in the settled set.

**Loopy question 1.** The first loopy question is: What should be done to truthify the invariant?

Stop the video and figure this out for yourself. Then continue.

Here’s what we do: Fix the frontier set to contain only start node v and set d[v] to 0. Now P2 is true. Make the settled set empty. That makes P1 true. P3 is also true since the settled set is empty.

**Loopy question 2.** We write the loop. When can the loop stop — i.e. when do all elements of array d have their final values?

Stop the video and figure this out. Then continue.

By P1, upon termination, all reachable nodes should be in the settled set. By P2 and P3, this will be the case when the frontier is empty. For if the frontier is empty, v is in the settled set (it cannot be in the far-off set). Since the frontier is empty, no edge leaves the settled set, so any node reachable from v is in the settled set.

Thus, the loop stops when the frontier is empty and continues when frontier not empty.

You might have chosen the loop condition the settled set doesn’t contain all nodes reachable from v. This condition may be much harder to implement than the one we chose because it may be difficult to determine the number of reachable nodes. But the condition the frontier is not empty is easy to check. Further, if the frontier set is empty, the number of reachable nodes is the size of the settled set!

**Loopy question 3.** How does the repetend make progress toward termination?

Stop the video and figure this out for yourself. Then continue the video.

The repetend should increase the size of the settled set, since the loop terminates when the settled set contains all nodes reachable from v.

The theorem shows us how to make the settled set bigger. Let f be a node in the frontier with minimum d-value. The theorem says that d[f] is the shortest-path distance from v to f. So move f from the frontier to the settled set. This keeps invariant P1 true. We show this happening. We also show that there may be edges leaving f and going to nodes in all three sets. We will have to deal with these to maintain invariants P2 and P3. And we show the shortest path from v to f.

**Loopy question 4: How is the invariant maintained?**

P2 may be false because there may be a path from v to f to a node like w2 with all but w2 in the settled set, and it may have a shorter distance than d[w]. P3 may be false because there may be an edge from the settled to the far-off set — like node w0. Stop the video and think out how to truthify P2 and P3. Don’t be discouraged if you don’t get it right; this is a lot harder to answer than the other three questions. Then continue the video.

We use a loop that processes all edges leaving f, that is, all edges (f, w) for some node w. There are two cases to consider, (1) w is in the far-off set and (2) w is in the settled or frontier sets. So we have an if-statement.

1. If w is a far-off set, like w0, set d[w] to d[f] + wgt(f, w) and put w in the frontier set. After this is done for all edges (f, w) with w in the far-off set, P3 is true. Let’s remove node w0 from the picture.
2. If \( w \) is in the settled or frontier sets, like nodes \( w_1 \) and \( w_2 \), a new path from \( v \) to \( f \) to \( w \) has been created with all settled nodes except perhaps for the last one. If its distance, \( d[f] + \text{wgt}(f, w) \), is less than \( d[w] \), a shorter path has been found, so change \( d[w] \) accordingly. Note that if \( w \) is in the settled set, like \( w_1 \), nothing will change.

After all edges leaving \( f \) are thus processed, invariants P1, P2, and P3 are true.

This completes the development of the algorithm. We give the algorithm below. Isn’t it neat? Notice how it is developed quite easily from the invariant. If you can remember the invariant, you can develop the algorithm whenever you have to. Practice it!

// Shortest path algorithm: For all nodes \( w \) reachable from node \( v \),
// set \( d[w] \) to the distance of the shortest path from \( v \) to \( w \).
F= \{v\}; \quad d[v]= 0; \quad S= \{\};
// invariant: P1, P2, and P3
while (F != \{\}) {
    f= node in F with minimum d value;
    Remove f from F and add it to S;
    for each \( w \) with (f, w) an edge {
        if (w not in S or F) {
            d[w]= d[f] + \text{wgt}(f, w);
            add w to F;
        }
        else if d[f] + \text{wgt}(f, w) < d[w] {
            d[w]= d[f] + \text{wgt}(f, w);
        }
    }
}