Dijkstra's Algorithm

This algorithm finds the shortest path from a source vertex to all other vertices in a weighted directed graph without negative edge weights.

Here is the algorithm for a graph G with vertices V = {v_1, ... v_n} and edge weights w_{ij} for an edge connecting vertex v_i with vertex v_j. Let the source be v_1.

Initialize a set S = ∅. This set will keep track of all vertices that we have already computed the shortest distance to from the source.

Initialize an array D of estimates of shortest distances. D[1] = 0, while D[i] = ∞, for all other i. (This says that our estimate from v_1 to v_1 is 0, and all of our other estimates from v_1 are infinity.)

While S != V do the following:

1) **Find the vertex (not is S) that corresponds to the minimal estimate of shortest distances in array D.**
2) **Add this vertex, v_i into S.**
3) **Recompute all estimates based on edges emanating from v. In particular, for each edge from v, compute D[i]+w_{ij}. If this quantity is less than D[j], then set D[j] = D[i]+w_{ij}.**

Essentially, what the algorithm is doing is this:

Imagine that you want to figure out the shortest route from the source to all other vertices. Since there are no negative edge weights, we know that the shortest edge from the source to another vertex must be a shortest path. (Any other path to the same vertex must go through another, but that edge would be more costly than the original edge based on how it was chosen.)

Now, for each iteration, we try to see if going through that new vertex can improve our distance estimates. We know that all shortest paths contain subpaths that are also shortest paths. (Try to convince yourself of this.) Thus, if a path is to be a shortest path, it must build off another shortest path. That's essentially what we are doing through each iteration, is building another shortest path. When we add in a vertex, we know the cost of the path from the source to that vertex. Adding that to an edge from that vertex to another, we get a new estimate for the weight of a path from the source to the new vertex.

This algorithm is greedy because we assume we have a shortest distance to a vertex before we ever examine all the edges that even lead into that vertex. In general, this works because we assume no negative edge weights. The formal proof is a bit drawn out, but the intuition behind it is as follows: If the shortest edge from the source to any vertex is weight w, then any other path to that vertex must go somewhere else, incurring a cost greater than w. But, from that point, there's no way to get a path from that point with a smaller cost, because any edges added to the path must be non-negative.

By the end, we will have determined all the shortest paths, since we have added a new vertex into our set for each iteration.

This algorithm is easiest to follow in a tabular format.

The adjacency matrix of an example graph is included below. Let a be the source vertex.

	a	b	c	d	e
a	0	10	inf	inf	3
b	inf	0	8	2	inf
c	2	3	0	4	inf
d	5	inf	4	0	inf
e	inf	12	16	13	0

Here is the algorithm:

	Estimates	b	c	d	e
Add to Set					
a		10	inf	inf	3
e		10	19	16	3
b		10	18	12	3
d		10	16	12	3

We changed the estimates to c and d to 19 and 16 respectively since these were improvements on prior estimates, using the edges from e to c and e to d. But, we did NOT change the 10 because 3+12, (the edge length from e to b) gives us a path length of 15, which is more than the current estimate of 10. Using edges bc and bd, we improve the estimates to both c and d again. Finally using edge dc we improve the estimate to c.

Now, we will prove why the algorithm works. We will use proof by contradiction. After each iteration of the algorithm, we "declare" that we have found one more shortest path. We will assume that one of these that we have found is NOT a shortest path.

Let t be the first vertex that gets incorrectly placed in the set S. This means that there is a shorter path to t than the estimate produced when t is added into S. Since we have considered all edges from the set S into vertex t, it follows that if a shorter path exists, its last edge must emanate from a vertex outside of S to t. But, all the estimates to the edges outside of S are greater than the estimate to t. None of these will be improved by any edge emanating from a vertex in S (except t), since these have already been tried. Thus, it's impossible for ANY of these estimates to ever become better than the estimate to t, since there are no negative edge weights. With that in mind, since each edge leading to t is non-negative, going through any vertex not in S to t would not decrease the estimate of its distance. Thus, we have contradicted the fact that a shorter path to t could be found. Thus, when the algorithm terminates with all vertices in the set S, all estimates are correct.

Try an example your self on the graph with the following adjacency matrix using a as the source.

	a	**b**	**c**	**d**	**e**
a	**0**	**3**	**4**	**inf**	**inf**
b	**inf**	**0**	**inf**	**6**	**8**
c	**inf**	**2**	**0**	**1**	**5**
d	**inf**	**inf**	**inf**	**0**	**2**
e	**inf**	**inf**	**inf**	**inf**	**0**

Path Reconstruction in Dijkstra's Algorithm

Given the history of the values in the distance estimate array, we can trace the shortest path. In the example below, consider determining the shortest distance from a to c:

Add to Set \ Estimates	b	c	d	e
a	**10**	**inf**	**inf**	**3**
e	**10**	**19**	**16**	**3**
b	**10**	**18**	**12**	**3**
d	**10**	**16**	**12**	**3**

When we updated the estimate to c to be 16, we added vertex d. This means that the last edge of the path was d -> c. Now, go to the column for d. We see that d's distance was updated when we added b to the set S. Thus, the last edge of the shortest path from a to d is the edge d->b. Finally, we go to the column for b and find that the shortest distance to b is obtained by the edge a->b. Thus, putting everything together, we have the path a->b->d->c.

Consider if we had filled out the chart as follows:

Add to Set \ Estimates	b	c	d	e
a	**10/a**	**inf**	**inf**	**3/a**
e	**10/a**	**19/e**	**16/e**	**3/a**
b	**10/a**	**18/b**	**12/b**	**3/a**
d	**10/a**	**16/d**	**12/b**	**3/a**

When we update a estimate, we ALSO update which vertex got us there. When we finish, we just need the information on the last row to reconstruct any shortest path from a.

For example, if we wanted to reconstruct the shortest path from a → c, we go to the entry in the array for vertex c and see that the shortest path from a to c visited d right before getting to c, so now we know that the shortest path is:

a .. d → c

Next, we go to the array entry for vertex d and see that its predecessor was vertex b, so now our path is:

a .. b → d → c

Finally, we go to the array entry for vertex b and see that its predecessor was vertex a, so the path above is complete:

a → b → d → c

A good exercise would be to take the posted Dijkstra's code and edit it to perform path reconstruction.