Lesson: Dijkstra's Algorithm

Bookmark this page

By the end of this lesson, you will be able to **describe the Dijkstra's algorithm and apply it to find shortest paths in nonnegatively weighted graphs**.

Video - Dijkstra's Algorithm

0:05 / 6:58 1.50x

Start of transcript. Skip to the end.

Hi and welcome to this lesson on the Dijkstra algorithm.

This week, we're going to talk about shortest

paths in weighted graphs.

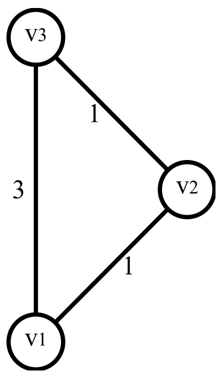
The graph traversal algorithms that we've already looked at in the previous lesson can't

be directly applied to find shortest paths in weighted graphs.

Weighted Graphs

Weighted graphs

This week, we're going to talk about shortest paths in weighted graphs. The graph traversal algorithms that we've already looked at in the previous lesson can't be directly applied to find shortest paths in weighted graphs. To illustrate why the graph traversal algorithms already presented do not necessarily work for weighted graphs, consider the following toy graph:



Here, a breadth-first search, BFS, starting from vertex v_1, will not produce a spanning tree of shortest paths. For example, the actual shortest path from v_1 to v_3 is $\{v_1, v_2\}, \{v_2, v_3\}$, where the path found using a BFS is $\{v_1, v_3\}$. This is because a BFS favors a smaller number of hops from the initial vertex, regardless of the weights.

Dijkstra's algorithm

Dijkstra's algorithm is a modification of the BFS that allows us to find shortest paths, even in the presence of weights, if those are non-negative.

A quick note about weights. So far in this MOOC, we've only considered weights that are non-negative. In some cases it might be meaningful to add negative weights. For example, imagine a taxi driver that can travel between cities. Sometimes he earns money because he has a client and sometimes he loses money because he travels alone. We could thus consider a graph whose vertices represent cities and edges are weighted by the cost of the corresponding travel. Some weights would then be positive and others negative.

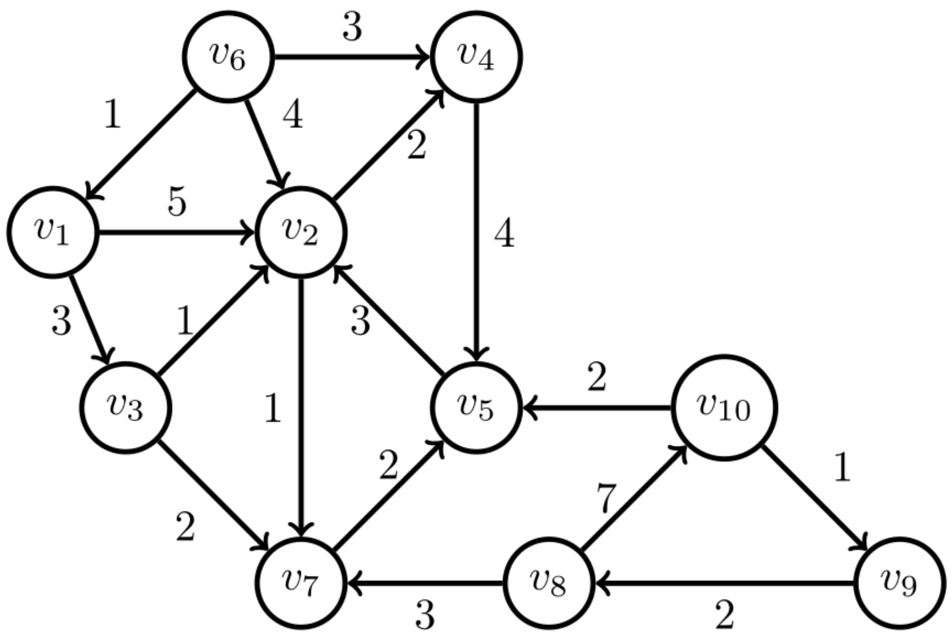
The simplest way to describe Dijkstra's algorithm is to imagine the starting vertex v_1 as a tap from which water is pouring out. The water will progress along the vertices and traverse the graph with a speed inversely proportional to the edge weights, so it's going to reach the closest vertices first. In other words, the Dijkstra's algorithm is a traversal algorithm in which vertices are visited by increasing distance (not necessarily increasing number of hops) from the initial vertex.

At each step, the Dijkstra algorithm stores two kinds of information: 1) which vertices have already been explored, and 2) an approximation of distances from v_1 to all other vertices in the graph.

The Dijkstra algorithm repeats two steps. In step 1, it selects an unexplored vertex v that is at a minimum distance from v_1. In step 2, it updates the distances from v_1 to the other vertices in the graph, using information about v and its neighbors.

Note that the algorithm always yields a spanning tree of minimum paths in the graph, provided weights are non-negative. We're not going to prove this result here, but it's not so hard to do a very nice exercise if you'd like to go a little deeper into the concepts described in this lesson.

Let's describe more precisely how the Dijkstra algorithm works using the following example:



We're going to build a minimum spanning tree starting from vertex v_1 in this graph.

So. We initialize two data structures: first, the set of explored vertices, which are initially empty, and second, the array of distances from v_1 to the other vertices in the graph, initialized to $+\infty$ everywhere but v_1, where we insert a 0.

vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
explored	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*
distance	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

The first step is to select a nonexplored vertex that is at minimum distance from the starting vertex. Here, we select v_1 as our starting vertex and add it to the set of explored vertices.

The second step is to update the distances taking into account the selected vertex v_1: here v_1 has two neighbors, v_2 and v_3. Reaching them in one hop from v_1 costs 5 and 3, respectively. So we update our distances from v_1 by saying that v_2 is at distance 5 and v_3 is at distance 3.

vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
explored	*Yes*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*
distance	0	5	3	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Note that the length of a shortest path from v_1 to v_2 is actually 4, but that we can't tell that yet given the results from the algorithm alone.

Let's now start a new iteration of the algorithm:

In the first step, we select an unexplored vertex at a minimum distance from v_1. Here, we choose v_3, which is at distance 3. We add it to the set of explored vertices.

In the second step, we look at the neighbors of v_3, which are v_2 and v_7, with corresponding weights of 1 and 2. So if we want to reach v_2 by going through v_3 one hop before, we have a total cost of 3, which is the cost to go to v_3, plus 1, which is the cost of doing one hop from v_3 to v_2. This is better than the previous distance we estimated for v_2, so we modify the distances accordingly. As far as v_7 is concerned, we now have an estimated distance of 3, the cost of going to v_3 plus 2 which is the cost of one hop from v_3 to v_7.

$vertex$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$explored$	Yes	No	Yes	No	No	No	No	No	No	No
$distance$	0	4	3	$+\infty$	$+\infty$	$+\infty$	5	$+\infty$	$+\infty$	$+\infty$

A new iteration begins. Now we select the next unexplored vertex with the smallest distance, which is v_2. We add it to the set of explored vertices.

The vertex v_2 has two neighbors, v_4 and v_7. The vertex v_4 is estimated to be at distance $4+2=6$, and v_7 at distance $4+1=5$. So we update the distance to v_4. As far as v_7 is concerned, the newfound distance is not better than the old one, so it has no effect.

$vertex$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$explored$	Yes	Yes	Yes	No	No	No	No	No	No	No
$distance$	0	4	3	6	$+\infty$	$+\infty$	5	$+\infty$	$+\infty$	$+\infty$

A new iteration begins. We select v_7 at distance 5 and add it to the list of explored vertices.
And so on... Finally we obtain the following tables:

$vertex$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$explored$	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No
$distance$	0	4	3	6	7	$+\infty$	5	$+\infty$	$+\infty$	$+\infty$

It is easy to verify that the distances found using Dijkstra's algorithm are indeed those of shortest paths from v_1 to accessible vertices in the graph.

That's all for this lesson! Next we will see how to implement the Dijkstra algorithm by using min-heaps.

Previous

Next

Connect

Blog
Contact Us
Help Center
Media Kit
Donate

GET IT ON Google play
Download on the App Store