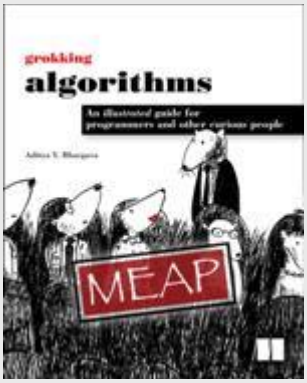


Grokking Algorithms: Dijkstras Algorithm

By Aditya Y. Bhargava

In this article, based on the book Grokking Algorithms, I'll discuss how to find the fastest path betwewen two points using an algorithm called *dijkstra's algorithm.*

Let's say in a previous exercise, we figured out the fastest way to get from Point A to Point B:

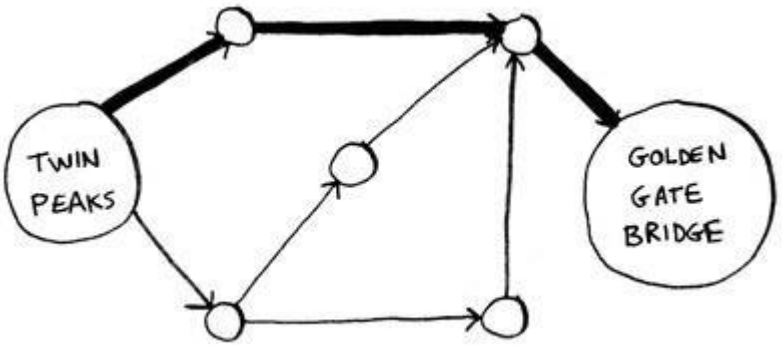


Figure 1

It's not really the "fastest" path. It is the shortest path, because it has the least number of segments (3 segments). But suppose we add travel times to those segments. Now we see that there is a faster path:

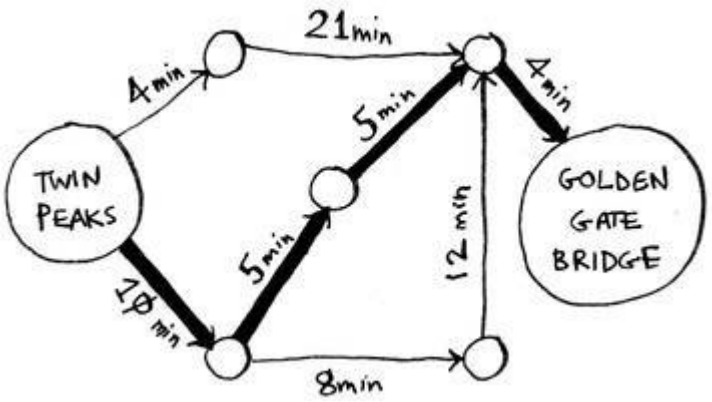


Figure 2

Breadth-first search will find you the path with the least number of segments (the first graph above). What if you want the *fastest* path instead (the second graph above)? We can do that with a different algorithm called *dijkstra's algorithm*. Let's see how it works with this graph:

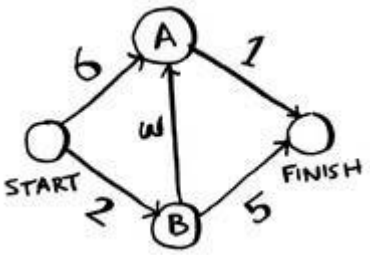


Figure 3

Each segment has a travel time in minutes. We will use Dijkstra's algorithm to go from start to finish in the shortest possible time. If we ran BFS on this graph, we would get this shortest path:

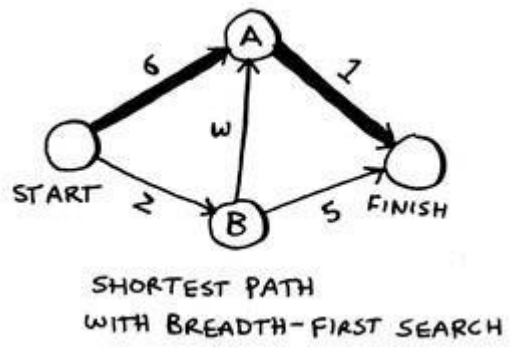


Figure 4

But that path takes 7 minutes. Let's see if we can find a path that takes less time!

There are four steps to Dijkstra's algorithm:

1. Find the "cheapest" node. This is the node you can get to in the least amount of time.
2. Update the costs of the neighbors of this node. I'll explain what I mean by this shortly.
3. Repeat until you have done this for every node in the graph.
4. Calculate the final path.

Step one: find the cheapest node. You are standing at the start, wondering if you should go to node A or node B. How long does it take to get to each node?

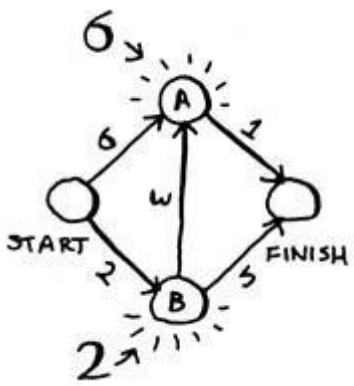


Figure 5

It takes 6 minutes to get to node A, and 2 minutes to get to node B. The rest of the nodes, we don't know yet:

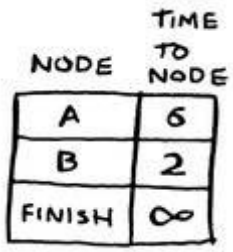
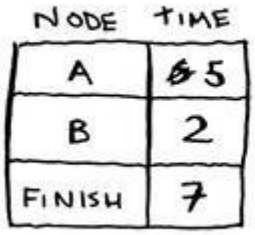


Figure 6

Since we don't know how long it takes to get to the finish yet, we'll put down infinity (you'll see why soon).

Node B is the closest node…it is 2 minutes away.

Step 2: calculate how long it takes to get to all of node B's neighbors, *by following the edge from B:*



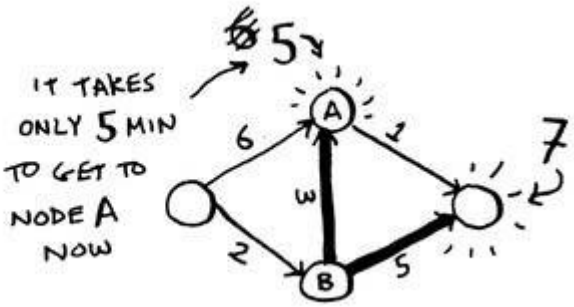


Figure 7

Hey, we just found a shorter path to node A! It used to take 6 minutes to get to node A:

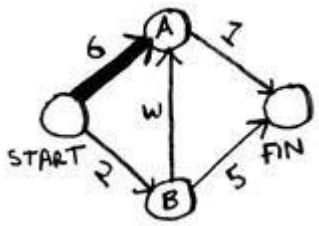


Figure 8

But if we go through node B, there's a path that only takes 5 minutes!

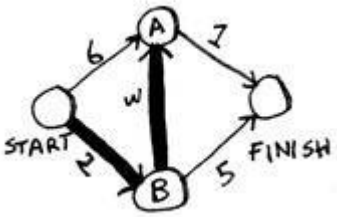


Figure 9

When you find a shorter path for a neighbor of B, update its cost. In this case, we found:

- a shorter path to A (down from 6 minutes to 5 minutes)
- a shorter path to the finish (down from infinity to 7 minutes)

Step 3: repeat!

Step 1 again: find the node that takes the least amount of time to get to. We are done with node B, so node A has the next smallest time estimate:

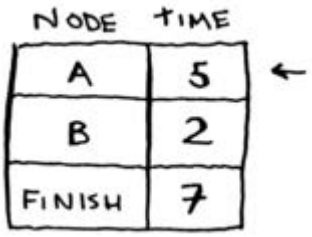


Figure 10

Step 2 again: update the costs for node A's neighbors:

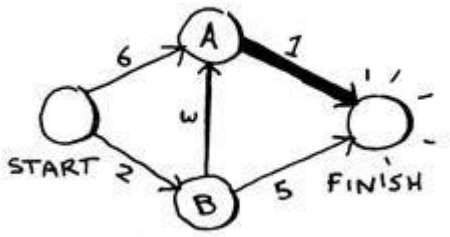


Figure 11

Woo, it takes 6 minutes to get to the finish now!

We have run Dijkstra's algorithm for every node (we don't need to run it for the finish node). At this point, we know:

- It takes 2 minutes to get to node B
- It takes 5 minutes to get to node A
- It takes 6 minutes to get to the finish

Figure 7

I will save the

NODE	TIME
A	5
B	2
FINISH	6

Figure 12

The final path is:

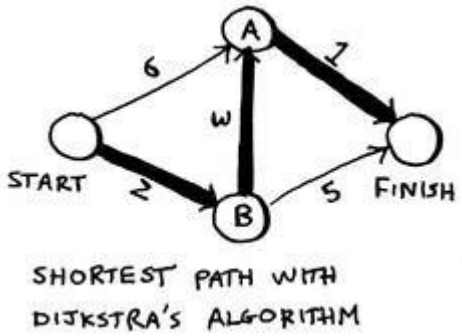


Figure 13

Breadth-first search would not have found this as the shortest path, because it has three segments. And there's a way to get from the start to the finish in two segments:

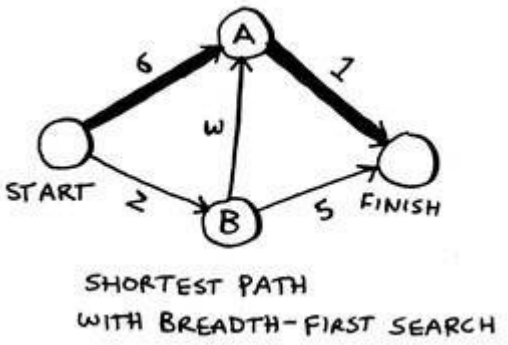


Figure 14

Using a breadth-first search to find the shortest path between two points just means the path with the least number of segments. But in Dijkstra's algorithm, you assign a number or *weight* to each segment. Then Dijkstra's algorithm finds the path with the smallest total weight.

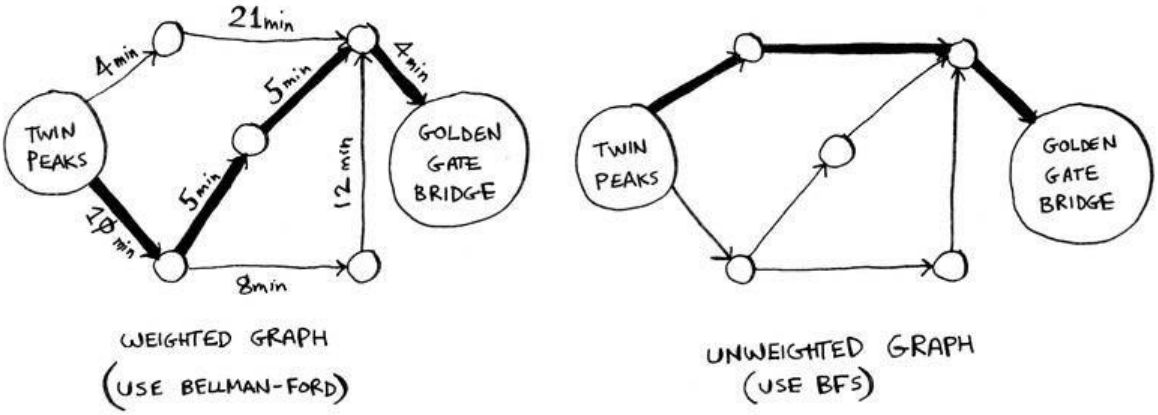


Figure 15

To recap, Dijkstra's algorithm has four steps:

1. Find the "cheapest" node. This is the node you can get to in the least amount of time.
2. Check if there's a cheaper path to the neighbors of this node. If so, update their costs.
3. Repeat until you have done this for every done in the graph.
4. Calculate the final path.