We have looked at breadth first search and depth first search in terms of finding if a node is reachable in a graph. And now we are going to extend those algorithms by looking at graphs which have edges that have weight. And we want to find the shortest path to that node. This is commonly done through Dijkstra's algorithm. So first let's start our run again.

Let's start debugging.

Okay, I seem to be missing out on the graph part, so let's try and try it again.

Okay, so now we have our graph structure and the way we are defining our graph is we are defining it as a topple, character, character, integer, and it's a vector of tuples that's going to be our edge. And we say that A to B is going to be of length 4 and it's bidirectional A to E is 15, A to G is 6 and so on. So we want to put it into a more easy data structure. So we are going to use an unordered map of characters, mapping to the values of unordered map of characters integers. So it's going to be A to B is 4.

When we get A as an index, we'll get an unordered map of all the things A is connected to. B is one of them. And from B we can get the distance to B with V. So the first thing is we want to put our edges into a more easy forms where it's we have graph from two is weight and graph two from is going to be weight.

And once we have done that, we are going to use our Dijkstra algorithm. We are giving the graph which is the set of edges and we want to start from A and we want to find the distance to Z. So let's go into our DI algorithm. Along the way we also want to keep track of the path. And we are going to use an unordered map structure previous.

And this one is going to say, hey, what is the node that was previous to this node? So if we have taken a path of A, E, F, Z, then the unordered map will say, well, to get to Z, we were at F. To get the F, we were at E. To get the E, we were at A and A was our starting point. And we want to find the shortest possible path. So we are going to have a priority queue and we want to have this priority queue ordered from the smallest to the largest, smallest being the node that we can reach by the shortest distance. So we are going to record it as a pair.

The distance to get to that node and the character what that node is. We'll also keep track of the distances for convenience and to make sure that if we find a shorter distance and we'll make sure to keep track of where what nodes that have already been visited. So the first thing we do is for everything in the graph, we set the pairs. We set the node to have a maximum distance possible. So the distance from A to B right now is int max.

A to Z is int max. We are going to use that as the maximum value and we are hoping to find shorter values.

So distance to the start, well, that's going to be zero. And we start by putting into our priority queue the distance 0 and our starting node A. And then we are going to look at our current distance and current nodes. We'll take them out of the see, let's make sure. So our current distance 0, our current node 97 is A.

And if the current node is the target, well in that case we would return it, but it's not. So we skip that. And then we check have we visited this node before? If we have, we would skip it. If we haven't, then we put it into our visited set.

And then we are going to look at all of our neighbors that are from this node. Let's see if we can make it a bit bigger.

So we are going to look at the next nodes. Where can we go from A? Well, from A A we can go to B, E and G. And so our next node we are looking at G and our new distance to get to g would be 6. If this new distance 6 is less than distance to G, which is right now int max saying hey, we found a better way to get to G. So we update our distances. We update saying hey, to get to G we would go from A and then we push into our priority queue G with a distance of 6.

And we do the same for E and for B.

So these are all in our priority queue. So next our priority queue is going to pick the node that has the smallest distance. The one with the smallest distance is, is going to be current node. In this case B, it's got only a distance of 4. So from there we'll look at have we reached our target, if not, have we visited B before?

We haven't. We put B into visited and we look at all of B's neighbors from B. One of the things we can go to is E. The distance to e was 4. If we go A to B and then B to E, that new distance would be 16 so that's not really better. So we haven't found a better way to get to E. So we don't update it.

Next we look at our next next node. We can also get to C. The distance to get to C is A to B is 4, B to C is 10, so it's 14. That's better than int max. So we are going to put that into our priority queue. And then we look at A and A already has a distance of zero.

So if we go to A to B and then from B to A, that's not going to be better. That's going to be a distance of 8. So we wouldn't put that into our queue again. And now we look at our priority queue and which one is the smallest? So now in our priority queue the shortest distance is G. If we go from A to G, G is not our target.

We haven't visited G before. And now we are going to add these neighbors to our priority queue. We are going to add H with 19, we'll consider adding E. But E already has 15. A to G and G to E is also 15, so no need to add it. And that will be the only possibilities from G. And priority queue at each loop is giving us the smallest possible distance.

Now it gives us C because C has a total distance of 14. So we look at all possible ways that we can go from C. We haven't visited C before. We look at C's neighbors. C's neighbors are going to be B which has been visited and it would be longer to get there and E which would be A longer. But we also have D and F as possibility.

So we are going to update the distance for F and update the distance for D and we'll continue. We keep going through this process until we get to the our current node, it being the targets. Which is our current node now is Z. And at that time we can create the path that we have taken. So to create the path we start from Z and say hey, to get to Z, where did we have to go?

So we put all of it into our vector. Since we and the start goes to the end of the vector. And then we are going to reverse this vector. So we have starting from A and then we can print our path.

So our path, the best path is A to G to H to I to Z. So this is the shortest path, A to G, G to H, H to I, I to Z.

And now we can return and say hey, distance from A to Z is 28 and we have found the shortest distance using Dijkstra's algorithm. The important parts of Dijkstra's algorithm is that we are going to have to maintain a priority queue so that we are always looking at the shortest distance. We are keeping track of the distances that the shortest distance we have to get to each node. We are keeping track of which nodes have been visited so we don't visit them again, and we are only updating the distance and adding it to the queue if we have found a new and better way to get to that node, and that's Dijkstra's algorithm for finding the shortest path between two nodes.