Lesson: Routing tables

Bookmark this page

By the end of this lesson, you will be able to create **routing tables** to navigate between any two vertices of a graph.

Video - Routing Tables

6:43 / 7:03 1.50x

As a second example, let’s also build the routing table from the spanning tree obtained

from a BFS.

Here's the spanning tree.

Again, we start from an empty table.v_1 has two children vertices, v_4 and v_2, so we

add v_1 in the corresponding columns.

The only child of v_4 is v_3, and the only child of v_2 is v_6.

Then, v_6 has two children, v_7 and v_5.

The routing table is complete!

If we read the path from v_1 to v_7 in the table we obtain v_7,v_6,v_2,v_1.

We can see that this path is shorter than the one obtained using a DFS, which was v_7,v_6,v_2,v_3,v_4,v_1.

This is an important difference between the DFS and the BFS.

BFS is a more cautious approach than

Transcripts

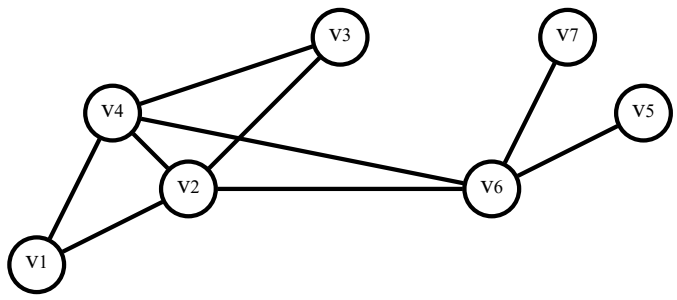
Download SubRip (.srt) file
Download Text (.txt) file

Hi everyone. In today's lesson, we will see how to build routing tables to navigate a graph, using the spanning trees obtained from graph traversal algorithms, such as BFS or DFS. These routing tables will help the artificial intelligence that you will develop in this course to move between two positions in the maze.

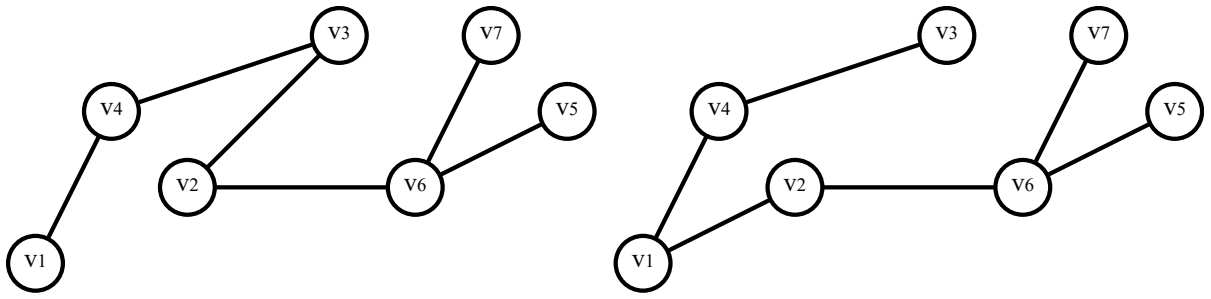
From spanning trees to routing

In the previous lesson you have learnt how to build spanning trees. Once we have such a spanning tree, we need a routing algorithm to be able to navigate between any two vertices in the graph.

Routing from a starting point to a destination is interpreted backwards. We already know what the starting point is, so we need to begin with the destination to find a path. So, a routing algorithm provides a path from the destination to the starting point. To better understand this, let's look at an example we've already seen in the previous lesson.



Let's consider the path between v_1 and v_7. We can see the spanning tree obtained using a DFS on the left, and the spanning tree obtained using a BFS on the right:

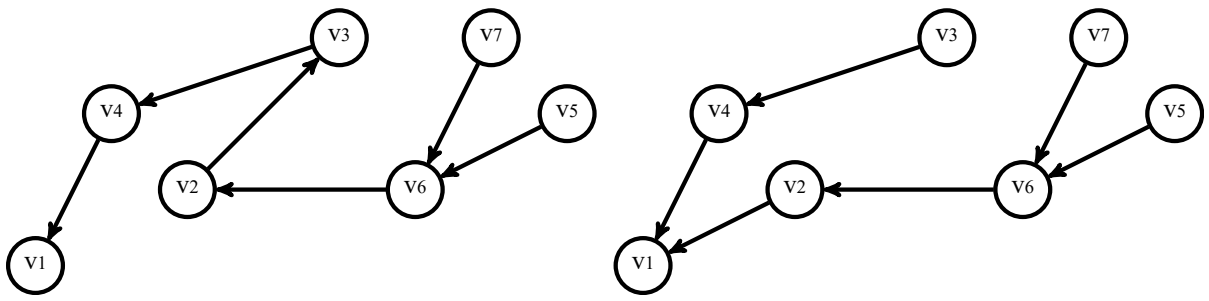


Routing from v_1 to any other vertex can be easy when we use trees. But to do this, we first need a few more definitions.

A rooted tree is a tree in which we designate a particular vertex as the root, which is also sometimes known as the origin. In this case, we will consider v_1 as the root, because it's our starting position. Therefore, the edges of this rooted tree have a natural orientation *towards the root*.

In a rooted tree, the parent of a vertex is the vertex connected to it on the path from the root. For example, the parent of v_2 is v_3 in the tree on the left. Every vertex except the root has a single parent. A child of a vertex v is a vertex of which v is the parent. So, v_2 is the child of v_3 in the tree on the left.

If we apply these concepts on the whole trees, we obtain the following rooted trees, where the orientation of the edges indicates the parents of the vertices. For example, in both graphs, v_1 has no parent, the parent of v_4 is v_1, and the children of v_6 are v_7 and v_5.



Routing tables

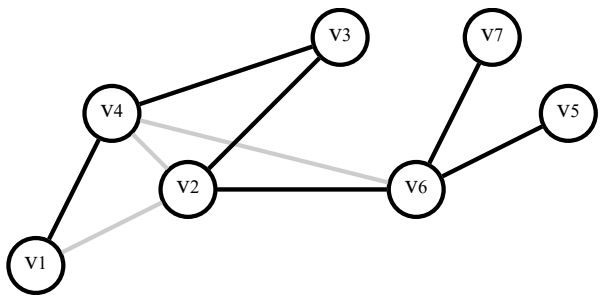
We can now build routing tables by considering the parents of each vertex in the path from v_1.

Routing tables are useful data structures that can be used to reconstruct a path. As we mentioned earlier, routing is considered backwards, from the destination back to the starting position.

A routing table has just one row, and as many columns as there are vertices in the graph. Each column corresponds to one vertex, and the associated value in the table is its parent in the spanning tree rooted in v_1.

Let us now see how this table is built step by step using a spanning tree.

Let's start with one obtained from a DFS.



We start from an empty table. v_1 is the root, so it doesn't have aparent. We simply note a "dot" $\cdot$ in the v_1 column.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\cdot$						

v_1 only has one child vertex, v_4. So we add v_1 in the column corresponding to v_4.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\cdot$			v_1			

v_4 only has one child, v_3, so we add v_4 in the column of v_3.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\cdot$		v_4	v_1			

Now let's simply apply the same principle to vertices v_2 and v_6, which are the only children of v_3 and v_2, respectively.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\cdot$	v_3	v_4	v_1		v_2	

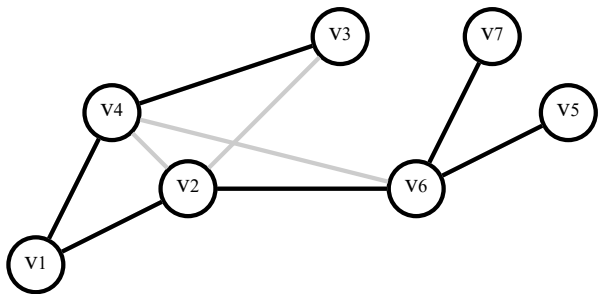
Finally, v_6 has two children, v_7 and v_5. So, let's add v_6 in the columns corresponding to v_5 and v_7.

So that's it, the routing table is complete.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
.	v_3	v_4	v_1	v_6	v_2	v_6

To read this table, you can simply start from the desired destination and read the corresponding entry, which refers to the parent vertex of the destination. For example, if we are looking for the path from v_1 to v_7, we first read the entry corresponding to v_7, which is v_6. By definition, the parent of the destination in turn corresponds to the previous vertex in the path from v_1 to the destination vertex. By reading the entries corresponding to each predecessor, you'll eventually reach the starting position v_1. So in our example, the next step is to identify the entry corresponding to v_6, which is v_2. By continuing like this, we obtain the path $v_7, v_6, v_2, v_3, v_4, v_1$.

As a second example, let's also build the routing table from the spanning tree obtained from a BFS.



Here's the spanning tree. Again, we start from an empty table.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
.						

v_1 has two children vertices, v_4 and v_2, so we add v_1 in the corresponding columns.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
.	v_1		v_1			

The only child of v_4 is v_3, and the only child of v_2 is v_6.

v_1	v_2	v_3	v_4	v_5	v_6	v_7
.	v_1	v_4	v_1		v_2	

Then, v_6 has two children, v_7 and v_5. The routing table is complete!

v_1	v_2	v_3	v_4	v_5	v_6	v_7
.	v_1	v_4	v_1	v_6	v_2	v_6

If we read the path from v_1 to v_7 in the table we obtain v_7, v_6, v_2, v_1. We can see that this path is shorter than the one obtained using a DFS, which was $v_7, v_6, v_2, v_3, v_4, v_1$. This is an important difference between DFS and BFS. BFS is a more cautious approach than DFS, as BFS gradually increases the distance from the starting position. Using BFS, we can be absolutely certain that we're going to find the shortest path between two vertices.

That's it for today. Thank you for your attention. I've really enjoyed talking about routing with you. You should remember that it is not sufficient to build the spanning tree to navigate between vertices in the graph, but that you also need a routing algorithm. See you soon for more algorithmic adventures.

‹ Previous Next ›

edX

About
Affiliates
edX for Business
Open edX
Careers
News

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap

Connect

Blog
Contact Us
Help Center
Media Kit
Donate