Exploration: Routing and Dijkstra's Algorithm

Introduction

We begin our second week on the Network Layer with a discussion of routing.

In order to create routes through networks, we need a way to model them. Networks can be modeled as a collection of Nodes connected to each other by lines (edges). These Edges have weights. The weights represent the "cost" of sending a packet from one node to another. These costs are determined by speed, distance, additional hardware, traffic, bottlenecks, etc.

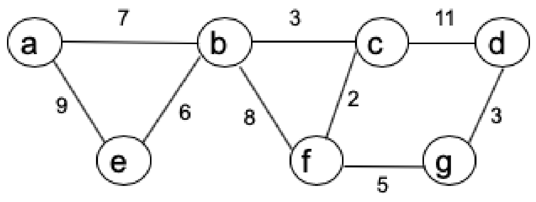
To determine a route or path, an algorithm is employed that can take into account the structure of nodes, weights, and edges. Many algorithms exist, with various emphasis and tradeoffs. One class of algorithms seek the "shortest-path" through the internet. Shortest-path refers to the path with lowest total weight (sum of weights of all edges in the path). Note that this may not be the path with the shortest physical distance or the fewest hops.

Routers store routing information as the "next-hop" in the path. This means that no router will contain full path information, only the next hop in the chain. This is done for simplicity (imagine having to recalculate the full path for every route every time something changes), and to save on storage.

Dijkstra's Algorithm

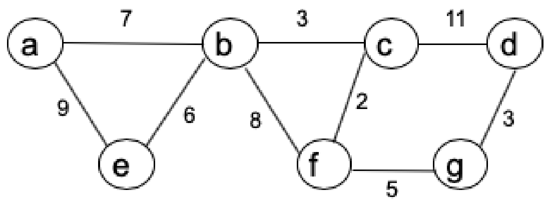
We are going to examine one of the better-known shortest path algorithms, known as "Dijkstra's Algorithm". We will be using an iterative approach, storing information in tables. This might differ a bit from your textbook, but is the same algorithm.

Here is a very simple network model (see below). Our goal is to find the shortest-path between our starting node d, and node a. Of course it would be easy to do this by visual inspection, but our purpose here is to illustrate the algorithm.



Initialization

Dest	D	R	P
a	∞	0	0
b	∞	0	0
c	11	c	d
d	*	*	*
e	∞	0	0
f	∞	0	0
g	3	g	d



The Destination (Dest) column refers to the destination node.

D represents the cost to reach that node, so, for example, D(g) will be the cost from our starting node d to node g. Node d is connected directly to c and g, so we note the weights required to reach c and g as 11 and 3 respectively. All other weights are initialized to infinity.

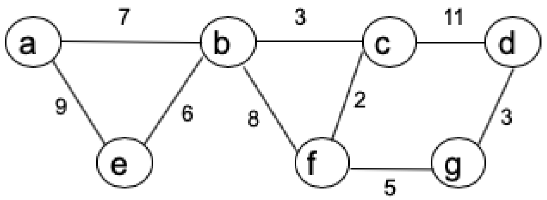
R is the next-hop in the path from our starting node to the Dest node. Our next hop to reach node c, is in fact our only hop to reach c, so we have R(c) = c. Similarly, we put R(g) = g.

Finally, we have column P, which represents the previous node. P is unnecessary for actual routing tables, but it conveniently lets us build and display the full path. For both g and c, we previously started at node d.

In addition to the table, we keep track of a few variables: The "current node" is held in u, and the set of all unvisited nodes are held in S. Currently u=d, and S={a,b,c,e,f,g}.

Iteration 1

Dest	**D**	**R**	**P**
a	∞	0	0
b	∞	0	0
c	11	c	d
d	*	*	*
e	∞	0	0
f	**8**	**g**	**g**
g	3	g	d



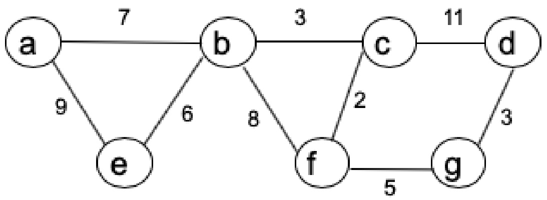
In our first iteration, we scan column D to find the lowest cost node in S (that is to say it has not been visited yet). It is D(g), so now g becomes our current node (u=g). We also remove g from S, so S={a,b,c,e,f}.

Next we examine the nodes connected directly to g that have not been visited yet. There is only one node, which is f. The cumulative cost to reach f is the cost of our current node D(g), plus the edge weight of the connection to f. Thus D(f) = D(g) + 5 = 3 + 5 = 8. If the result, 8, is lower than the current value of D(f), then we update row f with the new cost, next hop from the starting node, and previous hop information. 8 is less than infinity, so we update the table (as shown above).

Iteration 2

In our second iteration, we scan column D to find the lowest cost node in S. It is D(f), so now f becomes our current node (u=f). We also remove f from S, so S={a,b,c,e}.

Dest	**D**	**R**	**P**
a	∞	0	0
b	**16**	**g**	**f**
c	**10**	**g**	**f**
d	*	*	*
e	∞	0	0



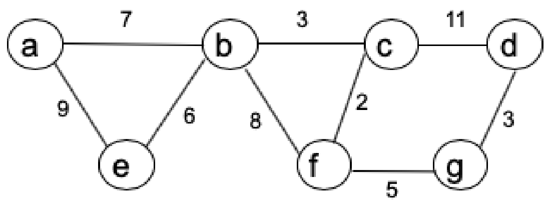
Dest	D	R	P
f	**8**	**g**	**g**
g	3	g	d

Next we examine the nodes connected directly to f that have not yet been visited. There are two, c and b. The cumulative cost to reach c is D(c) = D(f) + 2 = 8 + 2 = 10. If the result, 10, is lower than the current value of D(c), and it is, then we update all the values in row c. Similarly, row b is updated.

...Final Iteration

We continue in exactly the same way for several more iterations, until the set S is empty. The final result is shown below:

Dest	D	R	P
a	**20**	**g**	**b**
b	**13**	**g**	**c**
c	**10**	**g**	**f**
d	*	*	*
e	**19**	**g**	**b**
f	**8**	**g**	**g**
g	3	g	d



Here is an example showing this problem with all the steps included: dijkstra.pdf (https://oregonstate.instructure.com/courses/1798856/files/84020686?wrap=1) (https://oregonstate.instructure.com/courses/1798856/files/84020686?wrap=1)

We can now construct the full path from column P, working backwards from our destination, a. The calculated shortest path is therefore d-g-f-c-b-a. As mentioned earlier, a router will not store column P. The router only cares about the next hop in the path, which is given in column R. In our simplistic network, the next hop for our starting node d is always g, no matter the destination node.

It should be noted that routing algorithms are an area of active research. There are many types and derivations, and the complexity may be much greater than we have seen with Dijkstra's. Routing calculations are often performed by specialized computers instead of the routers, taking advantage of more memory and computing power. The results of these routing calculations are then distributed directly to the routers,. These systems are collectively known Software Defined Networking (SDN).

For more on routing and for a complete run-through of Dijkstra's algorithm, be sure to view the video lecture, then test your knowledge with the Self-Check exercises.

Video Lecture

Routing Algorithms

CS 372 Lecture

Routing

- Modeling the network core
- Djikstra's Algorithm

Note: Many of the lecture slides are based on presentations that accompany *Computer Networking: A Top Down Approach*, 6th edition, by Jim Kurose & Keith Ross, Addison-Wesley, 2013.

Oregon State University

0:00 / 12:12 1x

(PDF (https://oregonstate.instructure.com/courses/1798856/files/83165058/download?wrap=1) (https://oregonstate.instructure.com/courses/1798856/files/83165058/download?wrap=1) |PPT (https://oregonstate.instructure.com/courses/1798856/files/83165040/download?wrap=1) (https://oregonstate.instructure.com/courses/1798856/files/83165040/download?wrap=1))

Self-Check Exercises

A routing algorithm is used to find a datagram's path through a network.

○ True ○ False

Reuse <> Embed H5P

Resources

- Dijkstra's Algorithm Example with Solution (pdf) (https://oregonstate.instructure.com/courses/1798856/files/83165125/download)
- Dijkstra's Algorithm Solver (https://mdahshan.github.io/dijkstra/)
  "Dijkstra's Algorithm Solver." In https://mdahshan.github.io/dijkstra/ (https://mdahshan.github.io/dijkstra/) by Mostafa Dahshan