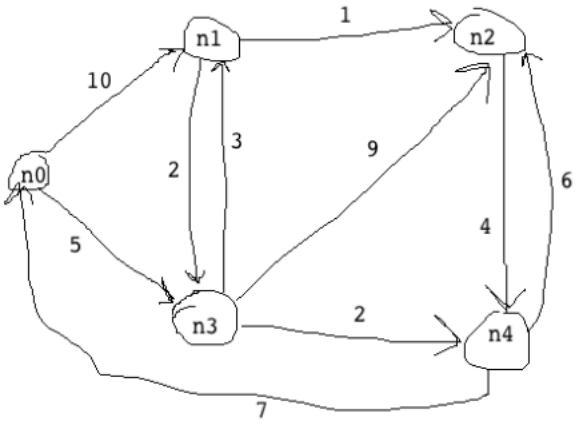
Name: Solutions January 17, 2018

HCS3 Block 6

Teach Yo Self: Shortest Path

The weights associated with each edge in the directed graph below represent distances. For example, the direct distance along the edge from n2 to n4 is 4, and the direct distance along the edge from n4 to n2 is 6.



1. On the graph above, highlight the shortest path from *n0* to *n2*.
What is the length of this path?

Highlight the path from *n0* to *n3* (length 5), *n3* to *n1* (length 3) and *n1* to *n2* (length 1), for a total length of 5 + 3 + 1 = 9.

We will now look at an algorithm for finding the shortest path from a given node to any other node.

Key Idea #1: Every node has a distance.

This is the length of the shortest path found *so far*, from the start node to this node. For example, let's choose *n0* to be our starting node. In the middle of running the algorithm, we find that *n2*'s distance is 14. That means we have already found a path of length 14 from *n0* to *n2*, but there could still be a shorter path from *n0* to *n2*. When the algorithm terminates, *n2*'s distance will be the length of *the* shortest path from *n0* to *n2*.

3. We wish to find the shortest path from *n0* to all other nodes.

   a. At the beginning of the algorithm, what distance should we use for *n0*?

      0

   b. At the beginning of the algorithm, what distance should we use for *n2*?

      infinity (same for every node except for the start node)

Key Idea #2: Every node has a predecessor.

n2's predecessor is the node immediately before it along the shortest path found so far from the start node to *n2*.

4. Suppose the shortest path found so far from *n0* to *n2* is:

   n0 ➔ *n3* ➔ *n4* ➔ *n2*

   Then *n2*'s predecessor is *n4*.

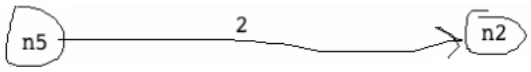
5. At the beginning of the algorithm, what is each node's predecessor value?

   null

Key Idea #3: Relaxing an edge

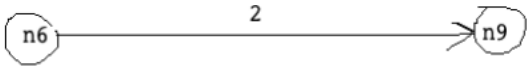
Relaxing an edge means determining if it can be helpful in finding a shorter path. If so, we must update its distance and predecessor values.

6. Show what happens when we relax this edge.



dist = 5 dist = ~~8~~ 7
pred = *n4* pred = ~~*n3*~~ *n5*

7. Show what happens when we relax this edge.



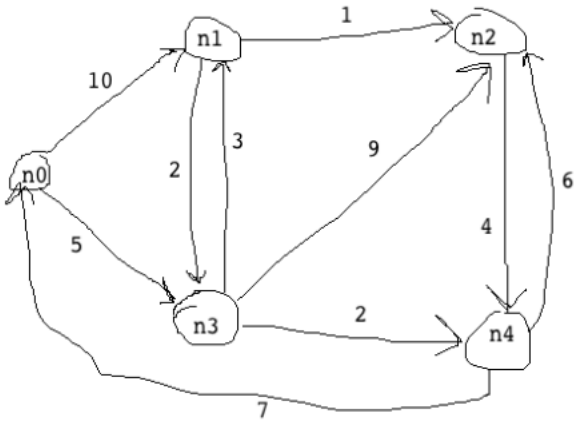
dist = 5 dist = 6 no change, because 5 + 2 > 6
pred = *n7* pred = *n8*

Dijkstra's Algorithm

(1) Find the non-visited node with the lowest distance.
(2) Relax it's outgoing edges.
(3) Repeat.

This algorithm will potentially relax each edge once.

8. We wish to find the shortest path from node *n0* to all other nodes in the graph below. Write each node's initial distance on the graph.



Now step through Dijkstra's algorithm. As you relax each edge, write it in the table below, along with the new distance and predecessor (if any).

Edge	**New Distance**	**New Predecessor**
n0 ➔ *n1*	10	*n0*
n0 ➔ *n3*	5	*n0*
n3 ➔ *n1*	8	*n3*
n3 ➔ *n2*	14	*n3*
n3 ➔ *n4*	7	*n3*
n4 ➔ *n0*	no change	no change
n4 ➔ *n2*	13	*n4*
n1 ➔ *n2*	9	*n1*
n1 ➔ *n3*	no change	no change
n2 ➔ *n4*	no change	no change

9. Suppose the variable *toVisit* refers to a data structure that keeps track of which nodes we have left to visit. Initially, all nodes are in *toVisit*.

```
while toVisit is not empty
{
    node = remove min from toVisit

    if _____ then stop

    relax each of node's outgoing edges
}
```

a. What is the missing stopping condition?

node is the one I'm trying to get to

b. What does it mean if the while loop terminates without ever reaching that stopping condition?

no way to get there. not connected

Let N stand for the number of nodes and E for the number of edges in a graph.

In a **sparse** graph, each node has O(1) edges. For example, maybe most nodes are directly connected to 3 other nodes.

10. In terms of N, in a sparse graph E = O(N)

In a **dense** graph, each node has O(N) edges. For example, maybe most nodes are directly connected to half of the other nodes.

11. In terms of N, in a dense graph E = O(N^2)

12. If a graph represents a network of roads and highways, is it likely to be sparse or dense? sparse

13. Suppose *toVisit* refers to an unordered `ArrayList` of nodes.

```
while toVisit is not empty
{
    node = remove min from toVisit

    relax each of node's outgoing edges
}
```

a. In terms of *N* and *E*, what is the expected total time spent removing all nodes from *toVisit*?

O(N^2)

b. In terms of *N* and *E*, what is the expected total time spent relaxing all edges?

O(E)

c. In terms of *N* and *E*, what is the expected time for this algorithm?

O($N^2 + E$)

d. In terms of *N*, what is the expected time for this algorithm on a sparse graph?

O(N^2)

e. In terms of *N*, what is the expected time for this algorithm on a dense graph?

O(N^2)

14. Given that we often need to remove the minimum value during Dijkstra's algorithm, what data structure should we use for *toVisit*?

    min-heap

15. Now suppose we use this better data structure for *toVisit*.

    a. In terms of N and E, what is the expected total time spent removing all nodes from *toVisit*?

       O($N \log N$)

    b. Why will relaxing an edge take more time for this data structure?

       have to re-heapify, which takes O(log N)

    c. In terms of N and E, what is the expected total time spent relaxing all edges?

       O($E \log N$)

    d. In terms of N and E, what is the expected time for this algorithm?

       O($E \log N$)

    e. In terms of N, what is the expected time for this algorithm on a sparse graph?

       O($N \log N$)

    f. In terms of N, what is the expected time for this algorithm on a dense graph?

       O($N^2 \log N$)