CE191: Civil and Environmental Engineering Systems Analysis

Lab 4: Shortest path routing on road networks

Due: Wednesday 11/18/09 1:00pm

Professor A. Bayen *Fall 09*

Correction

1. The Pseudo code in Algorithm 1 has been corrected. Question 2.2 has also been modified to reflect the change. The change means that the check "s.t. d has not been explored only applies to adding d into the queue.
2. For iteration 3 in Table 1 *cost* reads $[0, 5, 3, 7, \infty, \infty, \infty]$.

1 Lab overview

In this lab, you will implement Dijkstra's shortest path problem to solve routing problems on the freeways in the San Francisco Bay Area. Section 2 reviews the Dijkstra algorithm and its features. In Section 3, you are asked to implement the Dijkstra algorithm in MATLAB. Finally, in Section 4, you will run your algorithm on the San Francisco Bay Area, and compare the shortest path based on traveling at the speed limit, and the shortest path under congested traffic conditions. **Please remember to submit your MATLAB code (.m files in one ZIP file), and explain in the report how to run the code.**

2 Dijkstra algorithm

Dijkstra's algorithm is a method to solve the shortest path from a single source to all nodes in the graph, where the graph has nonnegative costs on the links. In this section,

will review a variation of the Dijkstra algorithm presented in class, which will be the basis of the algorithm you are asked to implement in this lab. The pseudo code for Dijkstra's algorithm is in Algorithm 1.

Algorithm 1 Dijkstra's algorithm

```
1 function Dijkstra(Graph, source)
2   Q = source %Put the source in the queue to explore.
3   for each node i in Graph
4      cost(i) = +∞ %Unknown cost from source to i
5      pred(i) = +∞ %Unknown predecessor node
6   end
7   cost(source) := 0
8   while Q is not empty %The main loop
9      minNode = node in Q for which
               cost(minNode) =min{cost(i), for all nodes i in Q}
10      remove minNode from Q
11*     for each downstream node d of minNode
12*       if d has not been explored
13*           Q = Q ∪ {d} % add d to the queue
14*       end
15        if cost(minNode) + cost_between(minNode, d) < cost(d)
16           cost(d) = cost(minNode) + cost_between(minNode, d)
17           pred(d) = minNode
18        end
19     end
20   end
21 return cost[]
```

Consider the graph in Figure 1. Following Algorithm 1, the solution is constructed in Table 1.

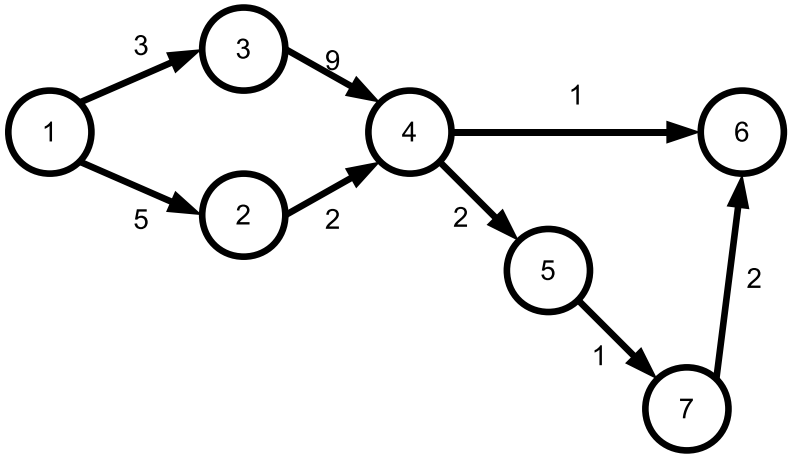

Figure 1: Example 1 Graph

iteration	minNode	Q	cost	pred
0	–	[1]	$[0, \infty, \infty, \infty, \infty, \infty, \infty]$	$[\infty, \infty, \infty, \infty, \infty, \infty, \infty]$
1	1	[2, 3]	$[0, 5, 3, \infty, \infty, \infty, \infty]$	$[\infty, 1, 1, \infty, \infty, \infty, \infty]$
2	3	[2, 4]	$[0, 5, 3, 12, \infty, \infty, \infty]$	$[\infty, 1, 1, 3, \infty, \infty, \infty]$
3	2	[4]	$[0, 5, 3, 7, \infty, \infty, \infty]$	$[\infty, 1, 1, 2, \infty, \infty, \infty]$
4				
5				
6				
7				

Table 1: Solution following Dijkstra's algorithm. *dist* and *pred* are indexed in the node order [1, 2, 3, 4, 5, 6, 7].

Question 2.1 Fill out the remaining rows of Table 1 following Algorithm 1.

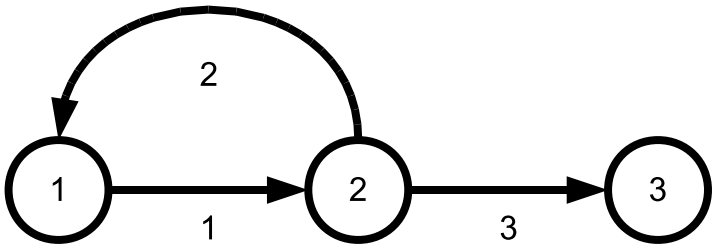

Figure 2: Example 2 Graph

Question 2.2 On line 11*-14* of the pseudo code in Algorithm 1 we must make sure the downstream node d has not already been explored when adding it back into the Queue. Explain why this is necessary. ***Hint:*** *Consider the graph in Figure 2. Build a table similar to Table 1, but assume line 11*-14* now reads:*

```
11*     for each downstream node d of minNode
12*
13*           Q = Q ∪ {d} % add d to the queue
14*
```

Will the algorithm terminate?

Question 2.3 In order to find the shortest path to a destination node *dest*, a colleague suggests Algorithm 1 can be modified to stop before all nodes in the graph are explored. The proposed modification to the pseudo code appears between lines 9 and 10 as follows:

```
9       minNode = node in Q for which
              cost(minNode) =min{cost(i), for all nodes iinQ}
9.5     if cost(minNode) ≥ cost(dest), break %new stopping criterion
10      remove minNode from Q
```

Will the modification still allow the algorithm to compute the shortest path to *dest*? If yes, explain why this stopping criterion is useful.

Question 2.4 For practical routing applications, it is meaningful to know the optimal path in addition to the minimum cost. Using the *pred* variable, explain in pseudo code how to reconstruct the optimal path from the source node *source* to the destination node *dest*. The result should be the list of nodes visited on the optimal path from *source* to *dest*.

3 Matlab Implementation

In this section you will implement Dijkstra's algorithm to run on the graph in Figure 1. The graph data is provided to you in the files example1Graph.mat and example1Cost.mat.

The file example1Graph.mat contains a scalar $numNodes$, and a matrix called $Graph$. The scalar $numNodes$ contains the number of nodes in the graph (numbered $1, 2, \cdots, numNodes$). The matrix $Graph$ contains information about the graph connectivity, summarized in Table 2.

linkID	upstream node	downstream node
12	1	2
13	1	3
...	...	...

Table 2: *Graph* matrix for the graph in Figure 1, contained in the file example1Graph.mat.

The fist column gives the link name, which connects the upstream node given in the second column with the downstream node in the third column. Thus, link 12 connects upstream node 1 to downstream node 2, while link 13 connects upstream node 1 to downstream node 3.

The file example1Cost.mat contains a matrix called $Cost$, which is summarized in Table 3. The first column of the matrix gives the link name, and the second column gives the cost of traversing the link.

linkID	free flow travel time (seconds)
12	5
13	3
...	...

Table 3: *Cost* matrix for the graph in Figure 1, contained in the file example1Cost.mat.

Using the data from example1Cost.mat and example1Graph.mat, you will create the file dijkstra.m, with the following signature:

```
function [optimalPath,pathCost] =...

    dijkstra(Graph,numNodes, Cost,source,dest)

%Graph: the Graph matrix in Table 2
%Cost: the Cost matrix in Table 3
%source: the source node
%dest: the destination node
%RETURNS
%optimalPath: vector of links on the optimal path
%pathCost: total cost of the optimal path
```

Figure 3: dijkstra.m signature

In order to simplify the implementation of dijkstra.m, we will first create four additional functions, which will be used in dijkstra.m.

Question 3.1 Create the following function:

```
function[downstreamNodes]= getDownstreamNodes(Graph, upstreamNode)
```

The function should take the *Graph* matrix and a node *upstreamNode* as inputs, and return a vector *downstreamNodes* containing all downstream nodes of *upstreamNode*. For example, using the *Graph* matrix from example1Graph.mat (also shown in Figure 1) $getDownstreamNodes(Graph, 1)$ should return the vector $downstreamNodes = [2, 3]$.

Question 3.2 Create the following function:

```
function[linkCost]= getCostBetween(Graph,Cost,upstreamNode,downstreamNode)
```

The function should take the *Graph* matrix, the *Cost* matrix, and the nodes *upstreamNode* and *downstreamNode* as inputs, and return a scalar value *linkCost*, which is the cost of traversing the link connecting *upstreamNode* and *downstreamNode*. For example, using the *Graph* matrix from example1Graph.mat (also shown in Figure 1) $cost\_between(Graph, Cost, 1, 2)$ should return the scalar $linkCost = 5$.

Question 3.3 Create the following function:

```
function[pathLinks]= getPathLinks(Graph,pred,source,dest)
```

The function should take the *Graph* matrix, the nodes *source* and *dest*, and the *pred* vector as inputs, and return a vector *PathLinks* containing the links traversed on the path stored in *pred*. You will need to modify your pseudo code from Question 2.4, so that the the links are returned. For example, using the *Graph* matrix from example1Graph.mat (also shown in Figure 1), $getPathLinks(Graph, [\infty, 1, 1, 2, \infty, \infty, \infty], 1, 4)$ should return the vector $pathLinks = [12, 24]$.

Note that some additional steps are needed in order to implement the Dijkstra pseudo code in Algorithm 1, because MATLAB does not support Queues. Instead, you will create a vector called *queueStatus*, with an entry for each node in the graph. Let $queueStatus(i) = -1$ if node i has not been explored, $queueStatus(i) = 1$ if node i is in the queue, and $queueStatus(i) = 0$ if node i has been explored.

Question 3.4 Create the following function:

```
function[minNode]= getMinNodeInQueue(queueStatus,cost)
```

The function should take the vectors *queueStatus* and *cost*, and return a scalar *minNode* equal to the node with minimum *cost* which has a *queueStatus* equal to 1. For example, using the *Graph* matrix from example1Graph.mat (also shown in Figure 1), consider the start of iteration 2 (Table 1) of the algorithm. We have $queueStatus = [0, 1, 1, -1, -1, -1, -1]$ and $cost = [0, 5, 3, \infty, \infty, \infty, \infty]$. Then $getMinNodeInQueue(\ queueStatus, cost)$ should return the scalar $minNode = 3$.

Question 3.5 Modify the pseudo code in Algorithm 1 to fit the function signature of dijkstra.m in Figure 3. The revised pseudo code should include the use of the functions $getDownstreamNodes()$, $getCostBetween()$, $getPathLinks()$, and $getMinNodeInQueue()$. It should also use the vector $queueStatus$ in place of the Queue, and incorporate the stopping criterion proposed in Question 2.3.

Question 3.6 Implement the pseudo code from Question 3.5 in MATLAB. Run Dijkstra.m on the example 1 Graph using the data provided in example1Cost.mat and example1Graph.mat, using a source node = 1, and a destination node = 6. Verify your results with the results computed by hand in Section 2.

4 Real world network

In this section you will use your Dijkstra's algorithm to compute optimal paths on the Bay Area highway road network, shown in Figure 4. The graph is summarized in the file BayAreaGraph.mat, which contains a matrix called *Graph*, and a scalar called $numNodes$ (as in the previous section). The *Graph* matrix is summarized in Table 4.

linkID	upstream node	downstream node
61699	2140	2123
61698	865	874
...	...	...

Table 4: Bay Area Highways Graph

Thus, link 61218 connects node 610 to node 653.

There are two sets of link costs provided for the Bay Area graph. The file BayAreaSpeedLimitCost.mat (Table 5) contains a *Cost* matrix, where the cost is the travel time required to traverse the link when traveling at the speed limit. The file BayAreaCongestedCost.mat (Table 6) contains a *Cost* matrix, where the cost is the travel time required to traverse the link during heavy morning traffic. The data is extracted from a real time traffic feed obtained from http://traffic.berkeley.edu in October 2009.

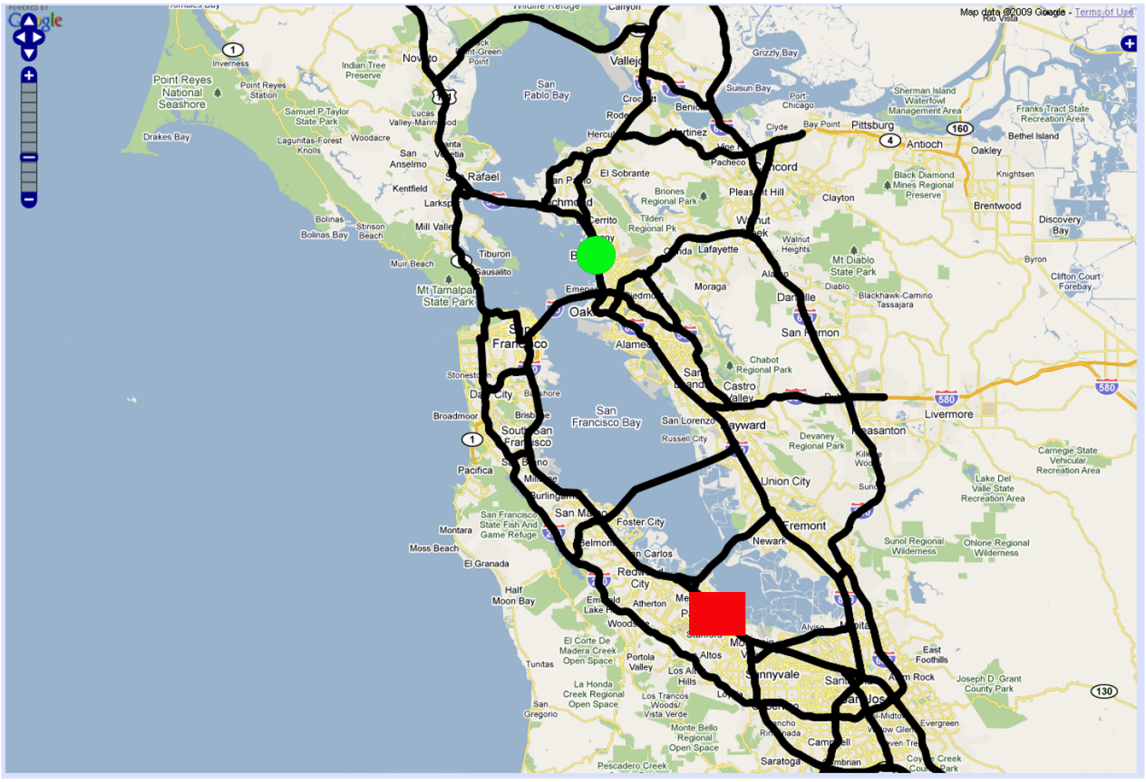

Figure 4: BayAreaGraph.mat includes major highways in the Bay Area. The origin node is 1741 in Berkeley (green circle), and the destination node is 1003 in Palo Alto (red square).

linkID	speed limit travel time (seconds)
61699	11.5
61698	12.77
...	...

Table 5: *Cost* matrix for the graph in Figure 4, stored in BayAreaSpeedLimitCost.mat

linkID	congested travel time (seconds)
6128	7.58
6129	33.56
...	...

Table 6: *Cost* matrix for the graph in Figure 4, stored in BayAreaCongestedCost.mat

Question 4.1 Using dijkstra.m, compute the shortest path from node 1741 in Berkeley to node 1003 in Palo Alto using the speed limit cost data in BayAreaSpeedLimitCost.mat. Use the function $Route2file(optimalPath, Cost)$ to save the optimal path returned by Dijkstra.m to the file OptimalRoute.txt.
Display your result in the visualizer at http://mmlive.ccit.berkeley.edu:9090/visualizer, and include a screenshot of your result. What is the travel time (in minutes)?

Question 4.2 Using dijkstra.m, compute the shortest path from node 1741 in Berkeley to node 1003 in Palo Alto using the speed limit cost data in BayAreaCongested.mat. Use the function $Route2file(optimalPath, Cost)$ to save the optimal path returned by Dijkstra.m to the file OptimalRoute.txt.
Display your result in the visualizer at http://mmlive.ccit.berkeley.edu:9090/visualizer, and include a screenshot of your result. What is the travel time (in minutes)? How long would the route which was optimal when traveling at the speed limit take if the same route was traversed during congestion?