CSE 2320 Notes 16: Shortest Paths

(CLRS 24.3, 25.2)

16.A. CONCEPTS

(Aside: [http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2597757.2530531](http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2597757.2530531))

Input:

Directed graph with non-negative edge weights (stored as adj. matrix for Floyd-Warshall)
Dijkstra – source vertex

Output:

Dijkstra – tree that gives a shortest path from source to each vertex
Floyd-Warshall – shortest path between each pair of vertices (“all-pairs”) as matrix

16.B. DIJKSTRA’S ALGORITHM – three versions

```plaintext
0 1 2 3 4 5 6 7
0(-) ∞ ∞ ∞ ∞ ∞ ∞ ∞
* 10(0) 20(0) 15(0)
* 13(4) 11(1) 20(1) * 18(4)
* 14(2) 16(2) 20(2) * 15(3)
* 18(7) * * *
```
Similar to Prim’s MST:

\[
\begin{align*}
S &= \text{vertices whose shortest path is known (initially just the source)} \\
\text{Length of path} \\
\text{Predecessor (vertex) on path (AKA shortest path tree)} \\
T &= \text{vertices whose shortest path is not known}
\end{align*}
\]

Each phase moves a T vertex to S by virtue of that vertex having the shortest path among all T vertices.

Third version may be viewed as being BFS with the FIFO queue replaced by a priority queue.

1. “Memoryless” – Only saves shortest path tree and current partition.
   \( \text{http://ranger.uta.edu/~weems/NOTES2320/dijkstraMemoryless.c} \)

Place desired source vertex \( x \in V \) in S

\[
\begin{align*}
T &= V - \{x\} \\
x.\text{distance} &= 0 \\
x.\text{pred} &= (-1) \\
\text{while } T \neq \emptyset \\
\quad \text{Find the edge } (s, t) \text{ over all } t \in T \text{ and all } s \in S \text{ with minimum value for } s.\text{distance} + \text{weight}(s, t) \\
\quad \text{(i.e. scan adj. list for each } s) \\
\quad t.\text{distance} &= s.\text{distance} + \text{weight}(s, t) \\
\quad t.\text{pred} &= s \\
\quad T &= T - \{t\} \\
\quad S &= S \cup \{t\}
\end{align*}
\]

Since no substantial data structures are used, this takes \( \Theta(VE) \) time.

2. Maintains T-table that provides the predecessor vertex in S for each vertex \( t \in T \) to give the shortest possible path through S to t. \( \text{http://ranger.uta.edu/~weems/NOTES2320/dijkstraTable.c} \)

Eliminates scanning all S adjacency lists in every phase, but still scans the list of the last vertex moved from T to S.

Place desired source vertex \( x \in V \) in S

\[
\begin{align*}
T &= V - \{x\} \\
x.\text{distance} &= 0 \\
x.\text{pred} &= (-1) \\
\text{for each } t \in T \\
\quad \text{Initialize } t.\text{distance} \text{ with weight of } (x, t) \text{ (or } \infty \text{ if non-existent) and } t.\text{pred} = x
\end{align*}
\]
while \( T \neq \emptyset \)

Scan \( T \) entries to find vertex \( t \) with minimum value for \( t.\text{distance} \)

\[
T = T - \{t\}
\]

\[
S = S \cup \{t\}
\]

for each vertex \( x \) in adjacency list of \( t \) (i.e. \( (t, x) \))

if \( x \in T \) and \( t.\text{distance} + \text{weight}(t, x) < x.\text{distance} \)

\[
x.\text{distance} = t.\text{distance} + \text{weight}(t, x)
\]

\[
x.\text{pred} = t
\]

Analysis:

Initializing the \( T \)-table takes \( \Theta(V) \).

Scans of \( T \)-table entries contribute \( \Theta(V^2) \).

Traversals of adjacency lists contribute \( \Theta(E) \).

\( \Theta(V^2 + E) \) overall worst-case.

3. Replace \( T \)-table by a min-heap.

( http://ranger.uta.edu/~weems/NOTES2320/dijkstraHeap.cpp )

The time for updating distances and predecessors increases, but the time for selection of the next vertex to move from \( T \) to \( S \) improves.

Place desired source vertex \( x \in V \) in \( S \)

\[
T = V - \{x\}
\]

\[
x.\text{distance} = 0
\]

\[
x.\text{pred} = (-1)
\]

for each \( t \in T \)

Initialize \( T \)-heap with weight (as the priority) of \( (x, t) \) (or \( \infty \) if non-existent) and \( t.\text{pred} = x \)

\text{minHeapInit}(T\text{-heap}) // a \text{fixDown} at each parent node in heap

while \( T \neq \emptyset \)

Use \text{heapExtractMin} /* \text{fixDown} */ to obtain \( T \)-heap entry with minimum \( t.\text{distance} \)

\[
T = T - \{t\}
\]

\[
S = S \cup \{t\}
\]

for each vertex \( x \) in adjacency list of \( t \) (i.e. \( (t, x) \))

if \( x \in T \) and \( t.\text{distance} + \text{weight}(t, x) < x.\text{distance} \)

\[
x.\text{distance} = t.\text{distance} + \text{weight}(t, x)
\]

\[
x.\text{pred} = t
\]

\text{minHeapChange}(T\text{-heap}) // \text{fixUp}

Analysis:

Initializing the \( T \)-heap takes \( \Theta(V) \).

Total cost for \text{heapExtractMins} is \( \Theta(V \log V) \).

Traversals of adjacency lists and \text{minHeapChanges} contribute \( \Theta(E \log V) \).

\( \Theta(E \log V) \) overall worst-case, since \( E > V \).
Which version is the fastest?

<table>
<thead>
<tr>
<th>Theory</th>
<th>Sparse ( E = O(V) )</th>
<th>Dense ( E = \Omega(V^2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( \Theta(EV) )</td>
<td>( \Theta(V^2) )</td>
<td>( \Theta(V^3) )</td>
</tr>
<tr>
<td>2. ( \Theta(V^2 + E) )</td>
<td>( \Theta(V^2) )</td>
<td>( \Theta(V^2) )</td>
</tr>
<tr>
<td>3. ( \Theta(E \log V) )</td>
<td>( \Theta(V \log V) )</td>
<td>( \Theta(V^2 \log V) )</td>
</tr>
</tbody>
</table>

16.C. FLOYD-WARSHALL ALGORITHM

Based on adjacency matrices. Will examine three versions:

Warshall’s Algorithm – After \( \Theta(V^3) \) preprocessing, processes each path existence query in \( \Theta(1) \) time.

Warshall’s Algorithm with Successors (or predecessors or transitive vertices) - After \( \Theta(V^3) \) preprocessing, provides a path in response to a path existence query in \( O(V) \) time (similar to dynamic programming backtrace).

Floyd-Warshall Algorithm (with Successors) - After \( \Theta(V^3) \) preprocessing, provides each shortest path in \( O(V) \) time.

Warshall’s Algorithm:

\[
\begin{align*}
\text{for } (j=0; \ j<V; \ j++) \\
\quad \text{for } (i=0; \ i<V; \ i++) \\
\quad \quad \text{if } (A[i][j]) \\
\quad \quad \quad \text{for } (k=0; \ k<V; \ k++) \\
\quad \quad \quad \quad \text{if } (A[j][k]) \\
\quad \quad \quad \quad \quad \ A[i][k]=1;
\end{align*}
\]

\hspace{1cm}

\[
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 \\
0 & 1 & \quad & \quad & \\
1 & 1 & 1 & \quad & \\
2 & 1 & \quad & \quad & \\
3 & 1 & \quad & \quad & \\
4 & \quad & \quad & \quad & \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 \\
0 & 1 & \quad & \quad & \\
1 & 1 & 1 & \quad & \\
2 & 1 & \quad & \quad & \\
3 & 1 & \quad & \quad & \\
4 & \quad & \quad & \quad & \\
\end{bmatrix}
\]
If zero-edge paths are useful for an application (i.e. reflexive, self-loops), the diagonal may be all ones.

Why does it work?

a. *Correct* in use of transitivity.

b. Is it *complete*?

<table>
<thead>
<tr>
<th>When</th>
<th>Paths That Can Be Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before $j=0$</td>
<td>$x \rightarrow y$</td>
</tr>
<tr>
<td>After $j=0$</td>
<td>$x \rightarrow 0 \rightarrow y$</td>
</tr>
<tr>
<td>After $j=1$</td>
<td>$x \rightarrow 1 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 0 \rightarrow 1 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 1 \rightarrow 0 \rightarrow y$</td>
</tr>
<tr>
<td>After $j=2$</td>
<td>$x \rightarrow 2 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 0 \rightarrow 2 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 1 \rightarrow 2 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 2 \rightarrow 0 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 2 \rightarrow 1 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 0 \rightarrow 2 \rightarrow 1 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 1 \rightarrow 2 \rightarrow 0 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow y$</td>
</tr>
<tr>
<td></td>
<td>$x \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow y$</td>
</tr>
</tbody>
</table>

...  

After $j=p$  
$x \rightarrow$ Permutation of *subset* of $0 \ldots p \rightarrow y$

After $j=V-1$  
ALL PATHS

Math. Induction:
Warshall’s Algorithm with Successors

Successor Matrix

Buc-ee’s directions:

\[
\begin{array}{ccc}
100 & 20 & 37 \\
s[100][200]=20 & s[20][200]=37 & s[37][200]=200 \\
\end{array}
\]

Initialize:

\[
\begin{array}{c}
\begin{array}{c}
\text{s[x][y]=y} \\
\text{(-1 otherwise)}
\end{array}
\end{array}
\]

Warshall Matrix Update:

\[
\begin{array}{c}
\begin{array}{c}
succ[i][j] = A \\
succ[j][k] = B \\
succ[i][k] = ?
\end{array}
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 3 & & \\
1 & 3 & 4 & \\
2 & 1 & & \\
3 & 2 & & \\
4 & & & \\
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{for (j=0; j<V; j++)} \\
\text{for (i=0; i<V; i++)} \\
\text{if (s[i][j] != (-1))} \\
\text{for (k=0; k<V; k++)} \\
\text{if (succ[i][k]==(-1) & succ[j][k]!=(-1))} \\
\text{succ[i][k] = succ[i][j];}
\end{array}
\end{array}
\]

Suppose code [in box] is removed for this graph:

\[
\begin{array}{c}
\begin{array}{c}
\text{0} \\
\text{1} \\
\text{2} \\
\text{3} \\
\text{4}
\end{array}
\end{array}
\]
Other ways to save path information:

Predecessors (warshallPred.c)

Transitive/Intermediate/Column (warshallCol.c)
Floyd-Warshall Algorithm with Successors (http://ranger.uta.edu/~weems/NOTES2320/floydWarshall.c)

After j = p has been processed, the shortest path from each x to each y that uses only vertices in 0 ... p as intermediate vertices is recorded in matrix.

for (j=0; j<n; j++)
{
    for (i=0; i<n; i++)
        if (dist[i][j]<oo)
            for (k=0; k<n; k++)
                if (dist[j][k]<oo)
                {
                    newDist=dist[i][j]+dist[j][k];
                    if (newDist<dist[i][k])
                    {
                        dist[i][k]=newDist;
                        succ[i][k]=succ[i][j];
                    }
                }
}
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>oo</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>oo</td>
<td>oo</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
</tr>
<tr>
<td>4</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>oo</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>oo</td>
<td>oo</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
</tr>
<tr>
<td>4</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
<td>oo</td>
</tr>
</tbody>
</table>

Note: In this example, zero-edge paths are not considered.