

CSE 3318 Notes 15: Shortest Paths

(Last updated 1/5/24 11:43 AM)

CLRS 22.3, 23.2

15.A. CONCEPTS

(Aside: <https://dl-acm-org.ezproxy.uta.edu/doi/10.1145/2530531>)

Input:

Directed graph with *non-negative* edge weights (stored as adj. matrix for Floyd-Warshall)

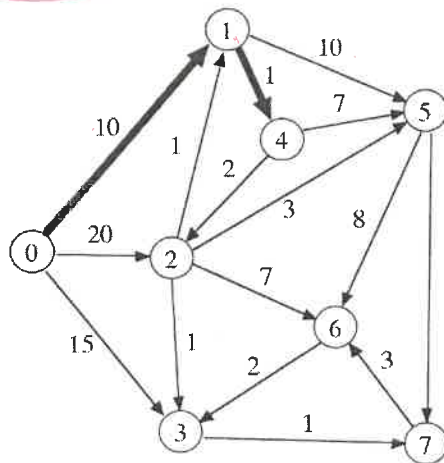
Dijkstra – source vertex

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

15.B. DIJKSTRA’S ALGORITHM – three versions



0	1	2	3	4	5	6	7
0(-) *	∞	∞	∞	∞	∞	∞	∞
	10(0) *	20(0)	15(0)				
		13(4)		11(1) *	20(1) 18(4)		
<hr/>							
		*	14(2) *		16(2)	20(2)	15(3) *
					*	18(7)	
						*	

Similar to Prim's MST:

S = vertices whose shortest path is known (initially just the source)

Length of path

Predecessor (vertex) on path (AKA shortest path tree)

T = vertices whose shortest path is not known

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.

(<https://ranger.uta.edu/~weems/NOTES3318/dijkstraMemoryless.c>)

Place desired source vertex $x \in V$ in S

$T = V - \{x\}$

$x.\text{distance} = 0$

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

while $T \neq \emptyset$

Find the edge (s, t) over all $t \in T$ and all $s \in S$ with minimum value for $s.\text{distance} + \text{weight}(s, t)$
(i.e. scan adj. list for each s)

$t.\text{distance} = s.\text{distance} + \text{weight}(s, t)$

$t.\text{pred} = s$

$T = T - \{t\}$

$S = S \cup \{t\}$

Since no substantial data structures are used, this takes $\Theta(EV)$ 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 . (<https://ranger.uta.edu/~weems/NOTES3318/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

$T = V - \{x\}$

$x.\text{distance} = 0$

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

for each $t \in T$

Initialize $t.\text{distance}$ with weight of (x, t) (or ∞ if non-existent) and $t.\text{pred} = x$

```

while T ≠ ∅
    Scan T entries to find vertex t with minimum value for t.distance
    T = T - {t}
    S = S ∪ {t}
    for each vertex x in adjacency list of t (i.e. (t, x))
        if x ∈ T and t.distance + weight(t, x) < x.distance
            x.distance = t.distance + weight(t, x)
            x.pred = t

```

$\Theta(E)$

Analysis:

Initializing the T-table takes $\Theta(V)$.
 Scans of T-table entries contribute $\Theta(V^2)$.
 Traversals of adjacency lists contribute $\Theta(E)$.
 $\Theta(V^2 + E) = \Theta(V^2)$ overall worst-case.

3. Replace T-table by a min-heap.

(<https://ranger.uta.edu/~weems/NOTES3318/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.distance = 0
x.pred = (-1)
for each t ∈ T

```

```

    Initialize T-heap with weight (as the priority) of (x, t) (or ∞ if non-existent) and t.pred = x
    minHeapInit(T-heap) // a fixDown at each parent node in heap
while T ≠ ∅
    Use heapExtractMin /* fixDown */ to obtain T-heap entry with minimum t.distance
    T = T - {t}
    S = S ∪ {t}
    for each vertex x in adjacency list of t (i.e. (t, x))
        if x ∈ T and t.distance + weight(t, x) < x.distance
            x.distance = t.distance + weight(t, x)
            x.pred = t
            minHeapChange(T-heap) // fixUp

```

$V \log V$

$E \log V$

Analysis:

Initializing the T-heap takes $\Theta(V)$.
 Total cost for heapExtractMins is $\Theta(V \log V)$.
 Traversals of adjacency lists and minHeapChanges contribute $\Theta(E \log V)$.
 $\Theta(E \log V)$ overall worst-case, since $E > V$.

Which version is the fastest?

	Theory	Sparse ($E = O(V)$)	Dense ($E = \Omega(V^2)$)
1.	$\Theta(EV)$	$\Theta(V^2)$	$\Theta(V^3)$
2.	$\Theta(V^2)$	$\Theta(V^2)$	$\Theta(V^2)$
3.	$\Theta(E \log V)$	$\Theta(V \log V)$	$\Theta(V^2 \log V)$

15.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).

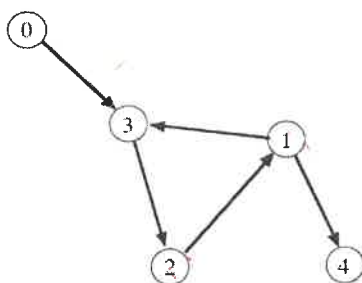
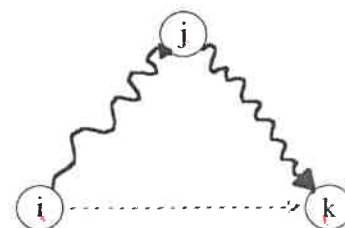
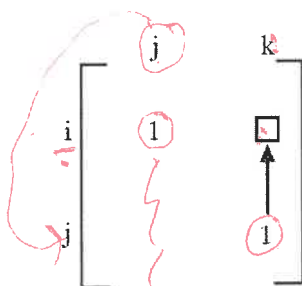
Floyd-Warshall Algorithm (with Successors) - After $\Theta(V^3)$ preprocessing, provides each *shortest* path in $O(V)$ time.

Warshall's Algorithm:

```

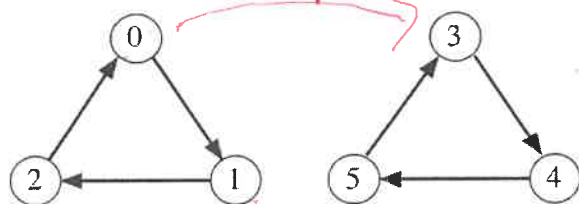
for (j=0; j<V; j++)
  for (i=0; i<V; i++)
    if (A[i][j])
      for (k=0; k<V; k++)
        if (A[j][k])
          A[i][k]=1;

```



	0	1	2	3	4
0	—	1	1	1	1
1	—	1	1	1	1
2	—	1	1	1	1
3	—	1	1	1	1
4	—	—	—	—	—

If zero-edge paths are useful for an application (i.e. reflexive, self-loops), the diagonal may be all ones.



	0	1	2	3	4	5
0	1	1	1			
1	1	1	1			
2	1		1			
3				1	1	1
4				1	1	1
5				1	1	1

Why does it work?

- Correct in use of transitivity.
- Is it complete?

When

Before $j=0$

After $j=0$

After $j=1$

After $j=2$

After $j=p$

After $j=V-1$

Paths That Can Be Detected

$x \rightarrow y$

$x \rightarrow 0 \rightarrow y$

$x \rightarrow 1 \rightarrow y$

$x \rightarrow 0 \rightarrow 1 \rightarrow y$

$x \rightarrow 1 \rightarrow 0 \rightarrow y$

$x \rightarrow 2 \rightarrow y$

$x \rightarrow 0 \rightarrow 2 \rightarrow y$

$x \rightarrow 1 \rightarrow 2 \rightarrow y$

$x \rightarrow 2 \rightarrow 0 \rightarrow y$

$x \rightarrow 2 \rightarrow 1 \rightarrow y$

$x \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow y$

$x \rightarrow 0 \rightarrow 2 \rightarrow 1 \rightarrow y$

$x \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow y$

$x \rightarrow 1 \rightarrow 2 \rightarrow 0 \rightarrow y$

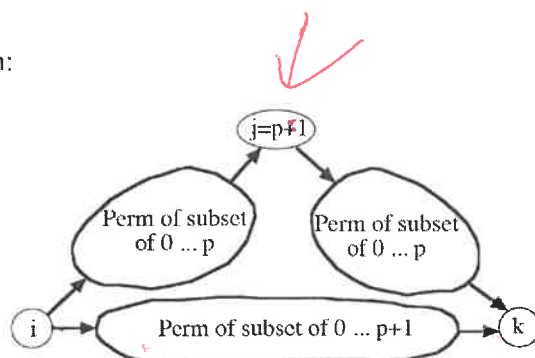
$x \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow y$

$x \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow y$

$x \rightarrow \text{Permutation of subset of } 0 \dots p \rightarrow y$

ALL PATHS

Math. Induction:

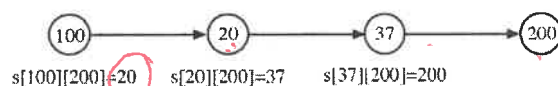


Suppose that there is only one path from vertex 5 to vertex 10 in a directed graph:
 $5 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 10$. During the scan of which column will Warshall's algorithm record the presence of this path?

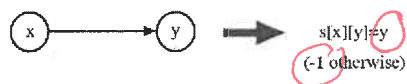
Warshall's Algorithm with Successors

Successor Matrix

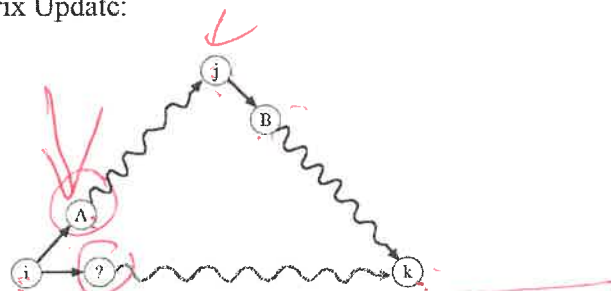
Buc-ee's directions:



Initialize:



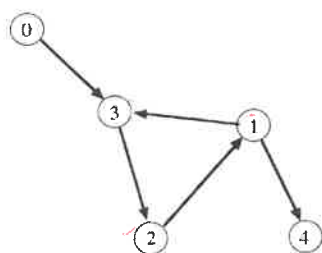
Warshall Matrix Update:



$\text{succ}[i][j] = A$

$\text{succ}[j][k] = B$

$\text{succ}[i][k] = ?$



	0	1	2	3	4
0	-	3	3	3	3
1	-	3	3	3	4
2	-	1	1	1	1
3	-	3	2	2	2
4	-	-	-	-	-

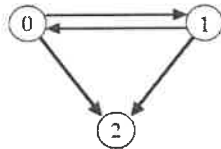
0 3 2 1 4
 2 1 3

```

for (j=0; j<V; j++)
  for (i=0; i<V; i++)
    if (s[i][j] != (-1))
      for (k=0; k<V; k++)
        if (succ[i][k]==(-1) && succ[j][k]!=(-1))
          succ[i][k] = succ[j][k];

```

Suppose code in box is removed for this graph:



Complete Example (<https://ranger.uta.edu/~weems/NOTES3318/warshall.c>) saving paths using successors:

	0	1	2	3	4
0	-1	-1	-1	3	-1
1	-1	-1	-1	3	4
2	-1	1	-1	-1	-1
3	-1	-1	2	-1	-1
4	-1	-1	-1	-1	-1

	0	1	2	3	4
0	-1	-1	-1	3	-1
1	-1	-1	-1	3	4
2	-1	1	-1	-1	-1
3	-1	-1	2	-1	-1
4	-1	-1	-1	-1	-1

	0	1	2	3	4
0	-1	-1	-1	3	-1
1	-1	-1	-1	3	4
2	-1	1	-1	1	1
3	-1	-1	2	-1	-1
4	-1	-1	-1	-1	-1

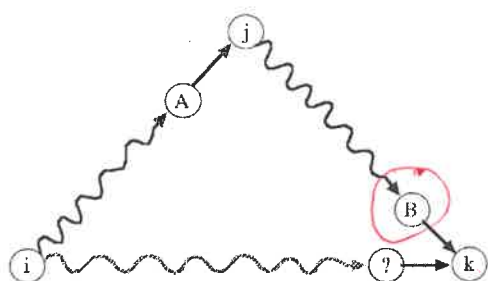
	0	1	2	3	4
0	-1	-1	-1	3	-1
1	-1	-1	-1	3	4
2	-1	1	-1	1	1
3	-1	2	2	2	2
4	-1	-1	-1	-1	-1

	0	1	2	3	4
0	-1	3	3	3	3
1	-1	3	3	3	4
2	-1	1	1	1	1
3	-1	2	2	2	2
4	-1	-1	-1	-1	-1

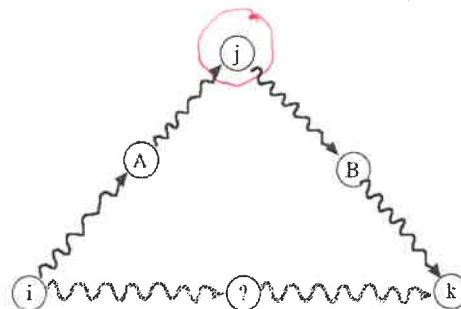
	0	1	2	3	4
0	-1	3	3	3	3
1	-1	3	3	3	4
2	-1	1	1	1	1
3	-1	2	2	2	2
4	-1	-1	-1	-1	-1

Other ways to save path information:

Predecessors (warshallPred.c)



Transitive/Intermediate/Column (warshallCol.c)

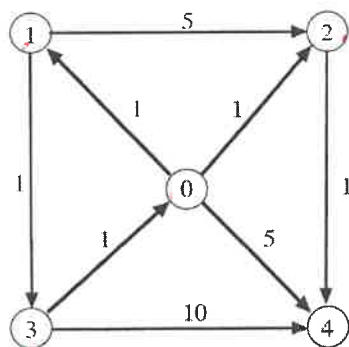


```
for (j=0; j<n; j++)
  for (i=0; i<n; i++)
    if (pred[i][j] != (-1))
      for (k=0; k<n; k++)
        if (pred[i][k] == (-1) && pred[j][k] != (-1))
          pred[i][k] = pred[j][k];
```

Floyd-Warshall Algorithm with Successors (<https://ranger.uta.edu/~weems/NOTES3318/floydWarshall.c>)

After $j = p$ has been processed, the *shortest path* from each x to each y that uses *only* vertices in $0 \dots p$ 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];
          }
        }
}
```



	0	1	2	3	4
0		1	1		5
1			5	1	
2					1
3	1				10
4					

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>10 4</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

	0	1	2	3	4
0	<u>00</u>	<u>1 1</u>	<u>1 2</u>	<u>00</u>	<u>5 4</u>
1	<u>00</u>	<u>00</u>	<u>5 2</u>	<u>1 3</u>	<u>00</u>
2	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>1 4</u>
3	<u>1 0</u>	<u>2 0</u>	<u>2 0</u>	<u>00</u>	<u>6 0</u>
4	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>00</u>

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

$3 \xrightarrow{1} 0 \xrightarrow{1} 2 \xrightarrow{1} 4 \checkmark$
 $3 \xrightarrow{1} 0 \xrightarrow{1} 1 \xrightarrow{1} 3 \checkmark$