CMPT 280

Tutorial: Dijkstra's Algorithm

Mark Eramian

University of Saskatchewan

Dijkstra's Algorithm

```
Algoirthm dijkstra(G, s)
Solves the single-source shortest paths problem.
G is a weighted graph with non-negative weights.
s is the start vertex.

Let V be the set of vertices in G.

For each v in V
    v.tentativeDistance = infinity
    v.visited = false
    v.predecessorNode = null

s.tentativeDistance = 0

while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Implementing Dijkstra's Algorithm

- As we saw in class, while the algorithm as presented suggests that `tentativeDistance`, `visited` and `predecessorNode` are properties of nodes, it is not necessary to implement a specialized node object to contain this data.
- Since nodes are always mapped onto integer identifiers, we can store these values in a series of parallel arrays.

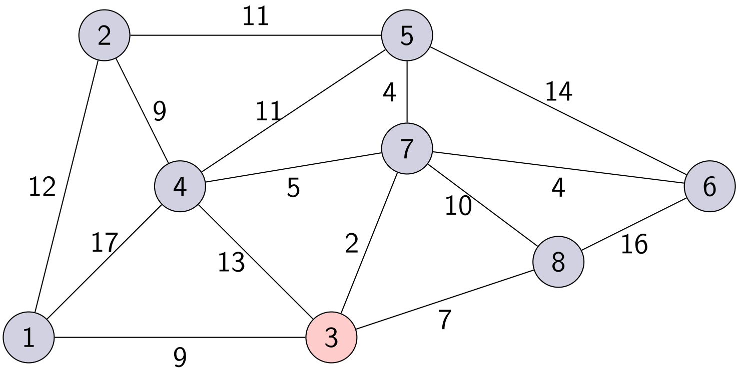
	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	F	F	F	F	F	F

	0	1	2	3	4	5	6	7	8
Predecessor:	-	-	-	-	-	-	-	-	-

	0	1	2	3	4	5	6	7	8
Tentative Distance:	∞	∞	∞	∞	∞	∞	∞	∞	∞

Tracing Dijkstra's Algorithm

Let's use Dijkstra's algorithm to find the shortest path from node 3 to every other node.



Dijkstra's Algorithm

Initialization for Start Node 3

Visited:

0	1	2	3	4	5	6	7	8
F	F	F	F	F	F	F	F	F

Predecessor:

0	1	2	3	4	5	6	7	8
-	-	-	-	-	-	-	-	-

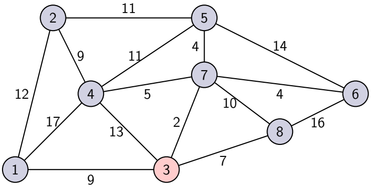
Tentative Distance:

0	1	2	3	4	5	6	7	8
∞	∞	∞	0	∞	∞	∞	∞	∞

Note: Offset 0 of these arrays are not used as we use node IDs as array offsets. Nodes are numbered 1 to n as in `lib280`.

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	F	F	F	F	F	F
Predecessor:	-	-	-	-	-	-	-	-	-
Tent. Dist.:	∞	∞	∞	0	∞	∞	∞	∞	∞



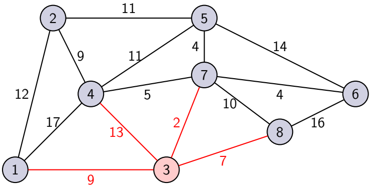
Unvisited node w/ Smallest tent. dist.: 3

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	F	F	F	F	F
Predecessor:	-	3	-	-	3	-	-	3	3
Tent. Dist.:	∞	9	∞	0	13	∞	∞	2	7



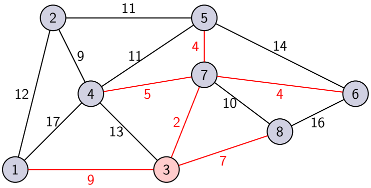
Processed Node 3, Next node to process: 7

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	F	F	F	T	F
Predecessor:	-	3	-	-	7	7	7	3	3
Tent. Dist.:	∞	9	∞	0	7	6	6	2	7



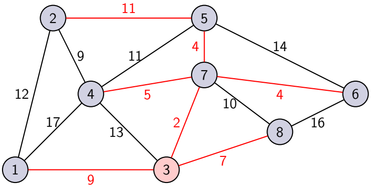
Processed Node 7, Next node to process: 5 or 6 (but we choose 5)

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	F	T	F	T	F
Predecessor:	-	3	5	-	7	7	7	3	3
Tent. Dist.:	∞	9	17	0	7	6	6	2	7



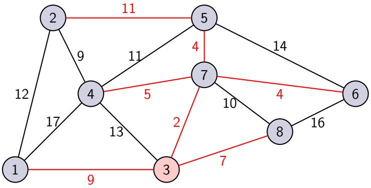
Processed Node 5, Next node to process: 6

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	F	T	T	T	F
Predecessor:	-	3	5	-	7	7	7	3	3
Tent. Dist.:	∞	9	17	0	7	6	6	2	7



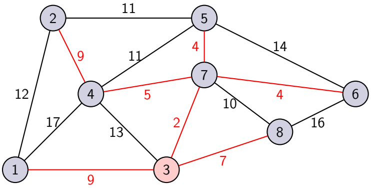
Processed Node 6, Next node to process: 4 or 8 (we choose 4)

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                   cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	T	T	T	T	F
Predecessor:	-	3	4	-	7	7	7	3	3
Tent. Dist.:	∞	9	16	0	7	6	6	2	7



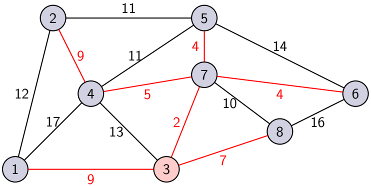
Processed Node 4, Next node to process: 8

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	F	F	T	T	T	T	T	T
Predecessor:	-	3	4	-	7	7	7	3	3
Tent. Dist.:	∞	9	16	0	7	6	6	2	7



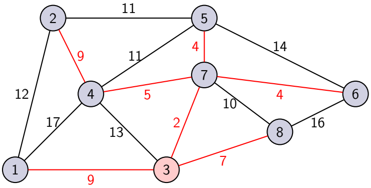
Processed Node 8, Next node to process: 1

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                 cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	T	F	T	T	T	T	T	T
Predecessor:	-	3	4	-	7	7	7	3	3
Tent. Dist.:	∞	9	16	0	7	6	6	2	7



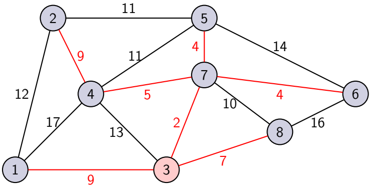
Processed Node 1, Next node to process: 2

```
while there is an unvisited vertex
    cur = the unvisited vertex with the smallest tentative distance.
    cur.visited = true

    // update tentative distances for adjacent vertices if needed
    // note that w(i,j) is the cost of the edge from i to j.
    For each z adjacent to cur
        if (z is unvisited and z.tentativeDistance >
                                cur.tentativeDistance + w(cur,z) )
            z.tentativeDistance = cur.tentativeDistance + w(cur,z)
            z.predecessorNode = cur
```

Dijkstra's Algorithm

	0	1	2	3	4	5	6	7	8
Visited:	F	T	T	T	T	T	T	T	T
Predecessor:	-	3	4	-	7	7	7	3	3
Tent. Dist.:	∞	9	16	0	7	6	6	2	7



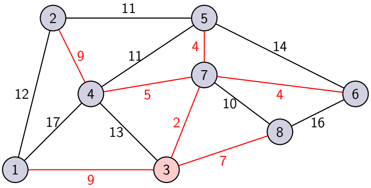
Processed Node 2, no more unvisited vertices, so we're done. Tentative distance array now contains the final lengths of the shortest paths from 3 to each node.

Dijkstra's Algorithm

Obtaining the Shortest Paths

To recover the shortest path from 3 to a node i, use the predecessor array starting at offset i, and follow the path back.

	0	1	2	3	4	5	6	7	8
Predecessor:	-	3	4	-	7	7	7	3	3



Recovering the shortest path from 3 to 2:

$2 \leftarrow 4 \leftarrow 7 \leftarrow 3$

Dijkstra's Algorithm

More Examples

For more practice, trace through Dijkstra's algorithm again, but from a different start node, or, for an entirely different graph... maybe a directed one?