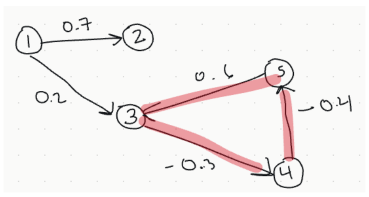


Figure 1: Counterexample for statement in **Exercise 1**: Negative cycle

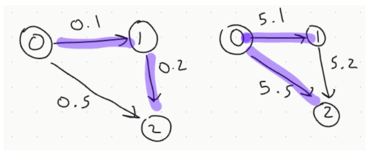


Figure 2: Counterexample for statement in **Exercise 1**: Paths with more edges

Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
January 19, 2025

4.4: Shortest Paths

Exercise 1. True or false: Adding a constant to every edge weight does not change the solution to the single-source shortest paths problem.

Solution. False. Consider the edge-weighted digraph on Figure 1. There is a negative cycle `3->4->5->3` reachable from `1`, so the single-source shortest paths problem has no solution when using `1` as the source, and any of vertices `3`, `4`, or `5` as the destination, according to **Proposition W**. However, if we add 0.2 to all edge weights in the graph, then the negative cycle is eliminated, and now we have a solution.

As another example, consider the graph in Figure 2. If the graph has vertices `0`, `1`, and `2`, with edge `0->1` with a weight of 0.1, edge `1->2` with a weight of 0.2, and edge `0->2` with a weight of 0.5, then the single-source shortest-path problem with vertex `0` as the source has the solution `0->1->2`. But if we add 5 to the weight of all edges, the solution is now `0->1` and `0->2`. In other words, adding a positive constant means that paths from the source to a destination that include more edges are affected more (they overall weight increment is more significant).

Exercise 2. Provide implementations of the constructor `EdgeWeightedDigraph(In in)` and the method `toString()` for `EdgeWeightedDigraph`.

Solution. See `com.segarciat.algs4.ch4.sec4.ex02`.

Exercise 3. Develop an implementation of `EdgeWeightedDigraph` for dense graphs that uses an adjacency-matrix (two-dimensional array of weights) representation (see **Exercise 4.4.10**). Ignore parallel edges.

Solution. See `com.segarciat.algs4.ch4.sec4.ex03`.

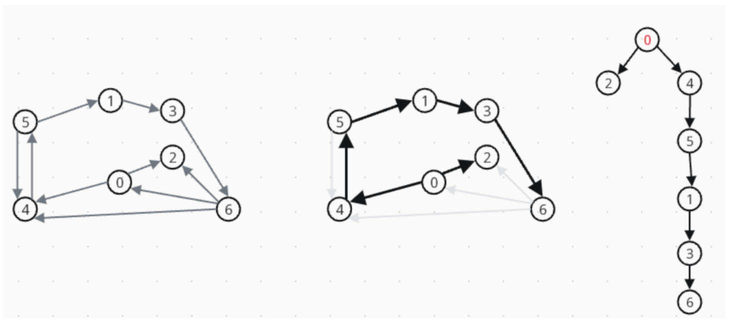


Figure 3: SPT when parent link representation for the graph implied by `tinyEWD.txt` with `0` as source and vertex `7` removed.

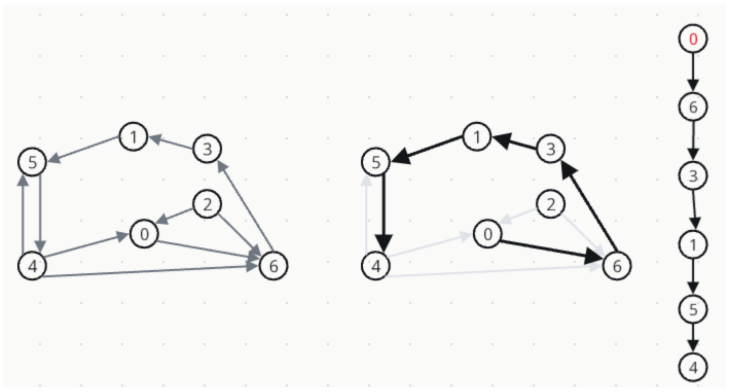


Figure 4: SPT when parent link representation for the graph implied by `tinyEWD.txt` with `0` as source, vertex `7` removed, and all edges reversed.

Exercise 4. Draw the (unique) SPT for source `0` of the edge-weighted digraph obtained by deleting vertex `7` from `tinyEWD.txt` (see page 644), and give the parent-link representation of the SPT. Answer the question for the same digraph with all edges reversed.

Solution. After deleting vertex `7` and its associated edges, the remaining of `tinyEWD.txt` is:

```
4 5  0.35
5 4  0.35
5 1  0.32
0 4  0.38
0 2  0.26
1 3  0.39
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```

See Figure 3 for the SPT with `7` removed, and see Figure 4 for the SPT with `7` removed and edges reversed, both with `0` as source.

Exercise 5. Change the direction of edge `0->2` in `tinyEWD.txt` (see page 644). Draw two different SPTs that are rooted at 2 for this modified edge-weighted digraph.

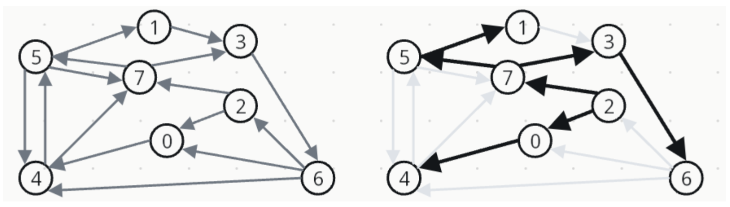


Figure 5: Modified edge-weighted digraph after reversing edge `0->2` in `tinyEWD.txt`, and SPT rooted at 2.

Solution. The full contents of `tinyEWD.txt` after reversal are:

```
8
15
4 5  0.35
5 4  0.35
4 7  0.37
5 7  0.28
7 5  0.28
5 1  0.32
0 4  0.38
2 0  0.26
7 3  0.39
1 3  0.39
2 7  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```

See Figure 5 for the resulting edge-weighted digraph and the shortest-path-tree. Note I was only able to find one SPT.

Exercise 6. Give a trace that shows the process of computing the SPT of the digraph defined in **Exercise 4.4.5** with the eager version of Dijkstra's algorithm.

Solution. See Figure 6 for the full trace.

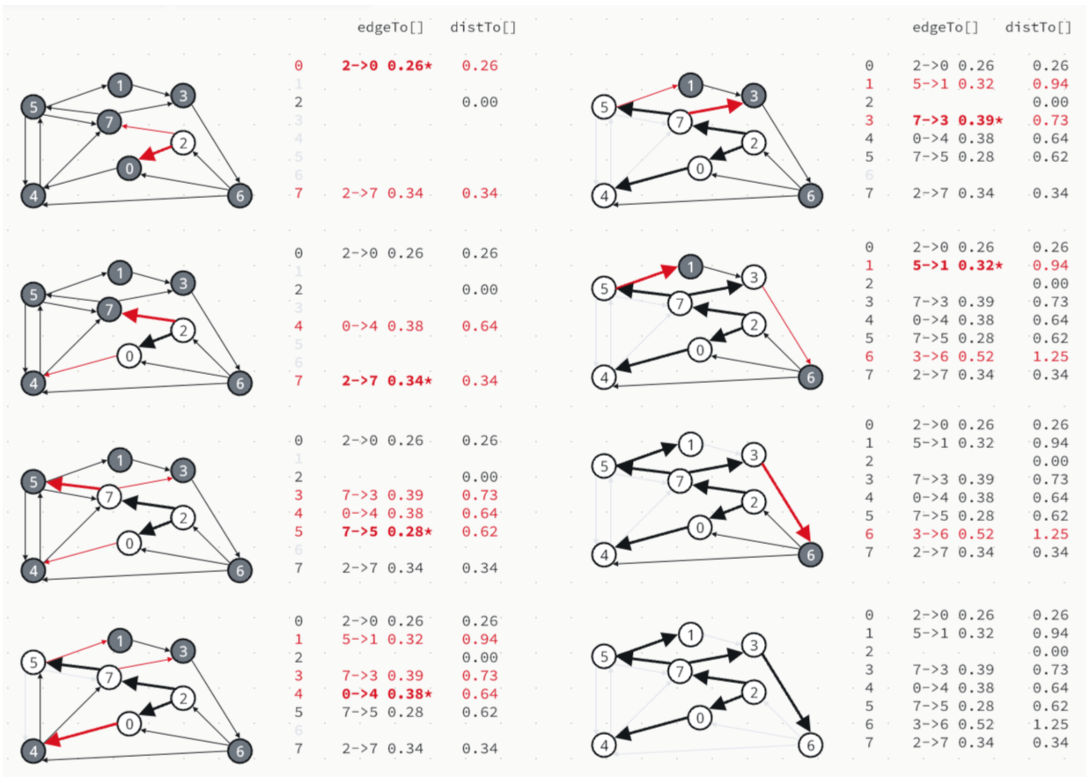
Exercise 7. Develop a version of `DijkstraSP` that supports a client method that returns a *second*-shortest path from `s` to `t` in an edge-weighted digraph (and returns `null` if there is only one shortest path from `s` to `t`).

Solution. See `com.segarciat.algs4.ch4.sec4.ex07`.

Exercise 8. The *diameter* of a digraph is the length of the maximum-length shortest path connecting two vertices. Write a `DijkstraSP` client that finds the diameter of a given `EdgeWeightedDigraph` that has nonnegative weights.

Solution. See `com.segarciat.algs4.ch4.sec4.ex08`.

Exercise 9. The table below, from an old published road map, purports to give the length of the shortest routes connecting the cities. It contains an error. Correct the table. Also, add a table that shows how to achieve the shortest routes.



```
     edgeTo[]     distTo[]
0    2->0 0.26*   0.26
1
2                 0.00
3
4
5
6
7    2->7 0.34    0.34
```

```
     edgeTo[]     distTo[]
0    2->0 0.26    0.26
1    5->1 0.32    0.94
2                 0.00
3    7->3 0.39*   0.73
4    0->4 0.38    0.64
5    7->5 0.28    0.62
6
7    2->7 0.34    0.34
```

```
0    2->0 0.26    0.26
1
2                 0.00
3
4    0->4 0.38    0.64
5
6
7    2->7 0.34*   0.34
```

```
0    2->0 0.26    0.26
1    5->1 0.32*   0.94
2                 0.00
3    7->3 0.39    0.73
4    0->4 0.38    0.64
5    7->5 0.28    0.62
6    3->6 0.52    1.25
7    2->7 0.34    0.34
```

```
0    2->0 0.26    0.26
1
2                 0.00
3    7->3 0.39    0.73
4    0->4 0.38    0.64
5    7->5 0.28*   0.62
6
7    2->7 0.34    0.34
```

```
0    2->0 0.26    0.26
1    5->1 0.32    0.94
2                 0.00
3    7->3 0.39    0.73
4    0->4 0.38    0.64
5    7->5 0.28    0.62
6    3->6 0.52    1.25
7    2->7 0.34    0.34
```

```
0    2->0 0.26    0.26
1    5->1 0.32    0.94
2                 0.00
3    7->3 0.39    0.73
4    0->4 0.38*   0.64
5    7->5 0.28    0.62
6
7    2->7 0.34    0.34
```

```
0    2->0 0.26    0.26
1    5->1 0.32    0.94
2                 0.00
3    7->3 0.39    0.73
4    0->4 0.38    0.64
5    7->5 0.28    0.62
6    3->6 0.52    1.25
7    2->7 0.34    0.34
```

Figure 6: Trace of the eager version of Dijsktra's algorithm to find the SPT rooted at 2 for the modified edge-weighted digraph after reversing edge `0->2` in `tinyEWD.txt`.

	Providence	Westerly	New London	Norwich
Providence	-	53	54	48
Westerly	53	-	18	101
New London	54	18	-	12
Norwich	48	101	12	-

Solution. The error is on the fourth row. It says that the shortest route from Norwich to Westerly is 101 units. But it also says that the shortest route from Norwich to New London is 12 units, and the shortest route from New London to Westerly is 18 units, which would imply that the shortest route from Norwich to Westerly is 30 units, not 101. To correct the table, I created a `DijkstraSP` client, from which I created the following table:

	Providence	Westerly	New London	Norwich
Providence	-	53	54	48
Westerly	53	-	18	30
New London	54	18	-	12
Norwich	48	30	12	-

See `com.segarciat.algs4.ch4.sec4.ex09`.

Exercise 10. Consider the edges in the digraph defined in **Exercise 4.4.4** to be undirected edges such that each edge corresponds to equal-weight edges in both directions in the edge-weighted digraph. Answer **Exercise 4.4.6** for this corresponding edge-weighted digraph.

Exercise 11. Use the memory-cost model of **Section 1.4** to determine the amount of memory used by `EdgeWeightedDigraph` to represent a graph with V vertices and E edges.

Solution. `EdgeWeightedDigraph` requires 16 bytes of object overhead, 4 bytes for its `int V` field, 4 bytes for its `int E` field, 8 bytes for its reference to the array field `Bag<DirectedEdge>[] adj`, and 24 bytes for the cost of the array itself, which makes for a flat cost of 56 bytes. The array `adj` has $8V$ references to `Bag<DirectedEdge>`, one for each adjacency list. Each `Bag<DirectedEdge>` requires 16 bytes of object overhead, 4 bytes for its `int size` field, 8 bytes for its `Node first` field, and 4 bytes of padding, for a cost of 32 bytes; since there are V of them, this amounts to $8V + 32V = 40V$ bytes. Now, for each `DirectedEdge`, we have a `Node` object that has a 40 byte cost. Meanwhile, the `DirectedEdge` itself requires 16 bytes of object overhead, 4 bytes for its `int v` field, 4 bytes for its `int w` field, and 8 bytes for its `double weight` field, amounting to 32 bytes. Since there are E of them, together with the cost of their corresponding `Node` wrapper nodes, this amounts to 72 bytes.

The overall cost is $56 + 40V + 72E$ bytes.

Exercise 12. Adapt the `DirectedCycle` and `Topological` classes from **Section 4.2** to use the `EdgeWeightedDigraph` and `DirectedEdge` APIs of this section, thus implementing `EdgeWeightedDirectedCycle` and `Topological` classes.

Solution. See `com.segarciat.algs4.ch4.sec4.ex12`.

Exercise 14. Show the paths that would be discovered by the two strawman approaches described on page 668 for the example `tinyEWDn.txt` shown on that page.

Solution. The file `tinyEWDn.txt` has the contents:

```
8
15
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
10->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2 -1.20
3->6  0.52
6->0 -1.40
6->4 -1.25
```

The actual SPT from `0` is depicted in Figure 7 Strawman I suggests adding the absolute value of the most negative weight to all edges. In this case, that value is $|-1.40| = 1.40$:

```
8
```

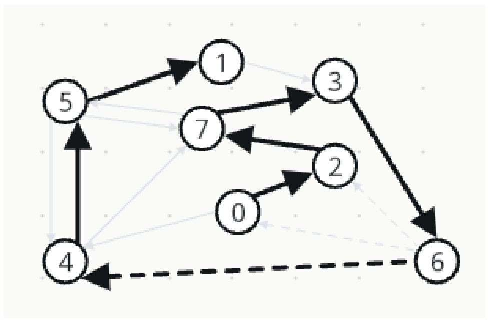


Figure 7: Actual SPT from `0` for `tinyEDWn.txt` (negative-weighted edges).

```
15
4->5  1.75
5->4  1.75
4->7  1.77
5->7  1.68
7->5  1.68
5->1  1.72
0->4  1.78
0->2  1.66
7->3  1.79
1->3  1.69
2->7  1.74
6->2  0.20
3->6  1.92
6->0  0.00
6->4  0.15
```

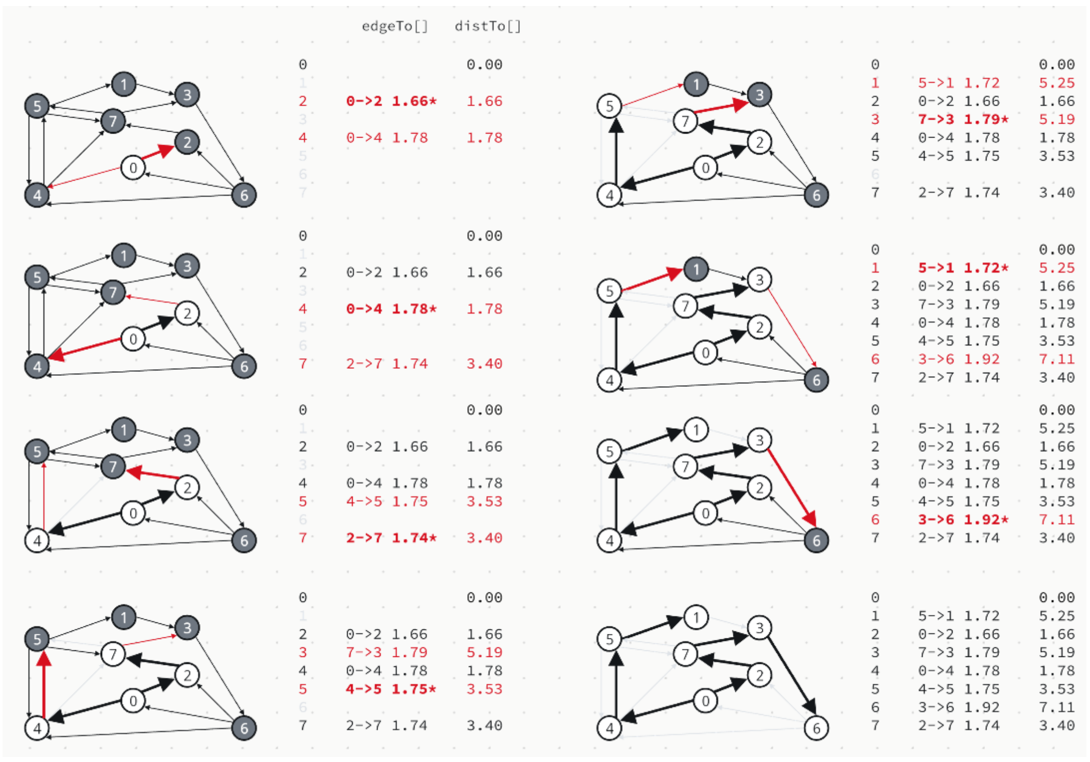
See Figure 8.

Exercise 15. What happens to Bellman-Ford if there is a negative cycle on the path from `s` to `v` and then you call `pathTo(v)`?

Solution. It would result in an out of memory error because the program would fall into an infinite loop. The problem is that when a negative cycle is encountered, the `edgeTo` entry for the first vertex in the cycle is overwritten by the last vertex by the last edge in the cycle. Therefore, when the parent-tree representation climbs back up, it will not go further than the first edge in the cycle, and will continue to enqueue the same cycle edges.

Exercise 16. Suppose that we convert an `EdgeWeightedGraph` into an `EdgeWeightedDigraph` by creating two `DirectedEdge` objects in the `EdgeWeightedDigraph` (one in each direction) for each `Edge` in the `EdgeWeightedGraph` (as described for Dijkstra's algorithm in the Q&A on page 684) and then use the Bellman-Ford algorithm. Explain why this approach fails spectacularly.

Solution. If a negative edge is present, then that edge is duplicated (once in each direction), creating a negative cycle. Since the digraph corresponds to the undirected graph, it is strongly connected, and hence the negative cycle is in the path to *every* vertex from *any* vertex.



```
   edgeTo[]     distTo[]
0               0.00
1
2  0->2 1.66*   1.66
3
4  0->4 1.78    1.78
5
6
7
```

```
0               0.00
1
2  0->2 1.66    1.66
3
4  0->4 1.78*   1.78
5
6
7  2->7 1.74    3.40
```

```
0               0.00
1
2  0->2 1.66    1.66
3
4  0->4 1.78    1.78
5  4->5 1.75    3.53
6
7  2->7 1.74*   3.40
```

```
0               0.00
1
2  0->2 1.66    1.66
3  7->3 1.79    5.19
4  0->4 1.78    1.78
5  4->5 1.75*   3.53
6
7  2->7 1.74    3.40
```

```
0               0.00
1  5->1 1.72    5.25
2  0->2 1.66    1.66
3  7->3 1.79*   5.19
4  0->4 1.78    1.78
5  4->5 1.75    3.53
6
7  2->7 1.74    3.40
```

```
0               0.00
1  5->1 1.72*   5.25
2  0->2 1.66    1.66
3  7->3 1.79    5.19
4  0->4 1.78    1.78
5  4->5 1.75    3.53
6  3->6 1.92    7.11
7  2->7 1.74    3.40
```

```
0               0.00
1  5->1 1.72    5.25
2  0->2 1.66    1.66
3  7->3 1.79    5.19
4  0->4 1.78    1.78
5  4->5 1.75    3.53
6  3->6 1.92*   7.11
7  2->7 1.74    3.40
```

```
0               0.00
1  5->1 1.72    5.25
2  0->2 1.66    1.66
3  7->3 1.79    5.19
4  0->4 1.78    1.78
5  4->5 1.75    3.53
6  3->6 1.92    7.11
7  2->7 1.74    3.40
```

Figure 8: Strawman 1 for `tinyEWDn.txt` for an SPT from 0, using Dijkstra's algorithm.
.

Exercise 17. What happens if you allow a vertex to be enqueued more than once in the same pass in the Bellman-Ford algorithm?

Solution. According to Sedgewick and Wayne, the running time can go exponential.

Exercise 18. Write a `CPM` client that prints all critical paths.

Solution. See `com.segarciat.algs4.ch4.sec4.ex18`.

Exercise 19. Find the lowest-weight cycle (best arbitrage opportunity) in the example shown in the text.

Exercise 20. Find a currency-conversion table online or in a newspaper. Use it to build an arbitrage table. *Note*: Avoid tables that are derived (calculated) from a few values and that therefore do not give sufficiently accurate conversion information to be interesting. *Extra credit*: Make a killing in the money-exchange market!

Exercise 21. Show, in the style of the trace in the text, the process of computing the SPT with the Bellman-Ford algorithm for the edge-weighted digraph of **Exercise 4.4.5**.

Solution. See Figure 9.

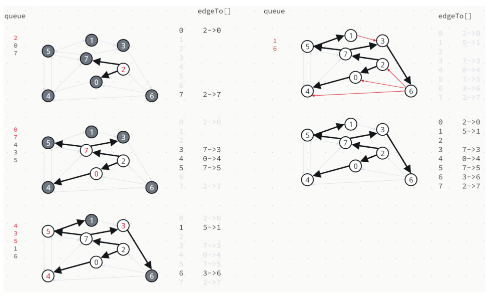


Figure 9: Trace of Bellman-Ford algorithm for **Exercise 21**.

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.