Graph theory: paths

Yulia Burkatovskaya
Department of Information Systems and Technologies
Associate professor

3. Paths

- Graph traversal
- Shortest paths

3.1. Graph traversal

Graph traversal is the problem of visiting all the vertices in a graph, updating and/or checking their values along the way.

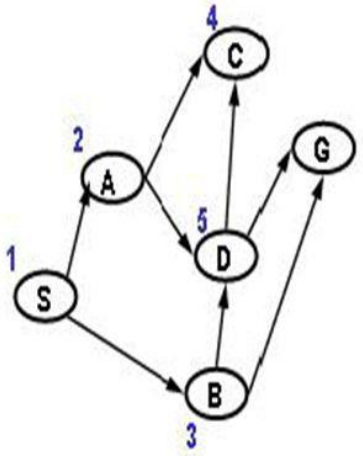
Breadth-first search (**BFS**) is a graph traversal algorithm that begins at a start vertex and explores all its neighbors (out-neighbors for a digraph). Then for each of those nearest vertices, it explores their unexplored neighbors, and so on, until all the vertices are visited.

Graph traversal

Example.

Breadth-First Search

	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S
5	(D A S) (G B S)	G,C,D,B,A,S
6		

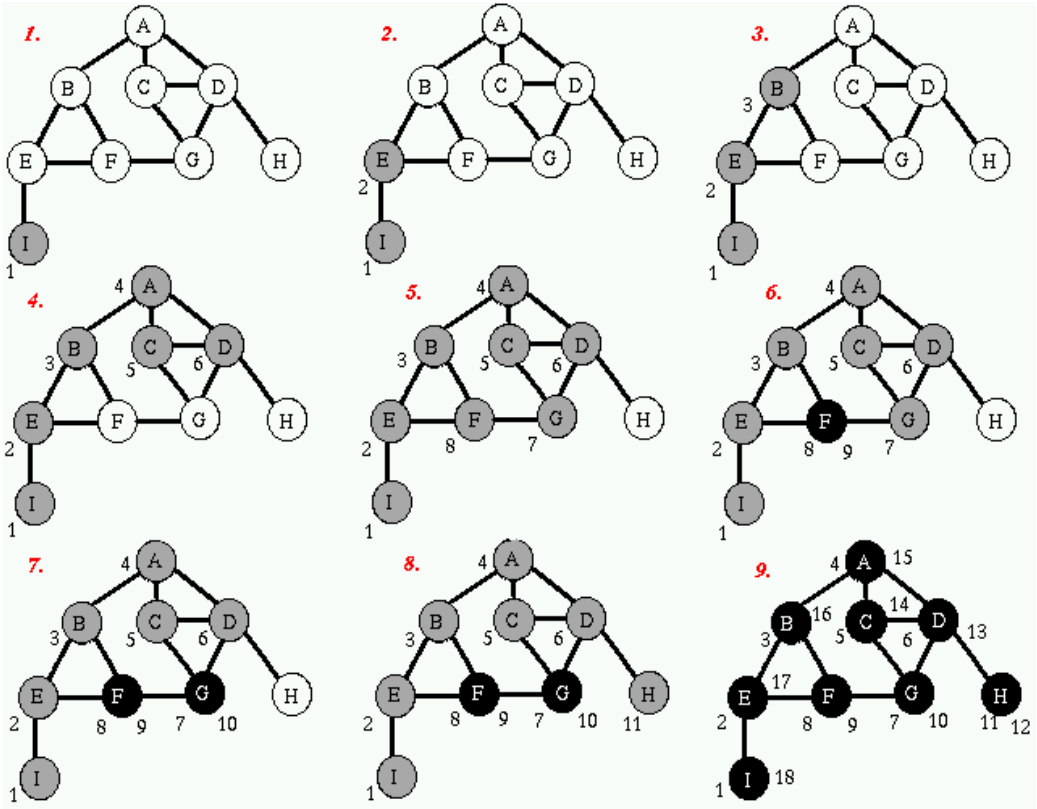


Graph traversal

Depth-first search (**DFS**) is a graph traversal algorithm that begins at a start vertex, explores its not visited neighbor and then considers that neighbor as a start vertex. If all the neighbors are visited then "**backtracking**" is used, i.e. the previous vertex is considered as a start vertex.

Graph traversal

Example. DFS.

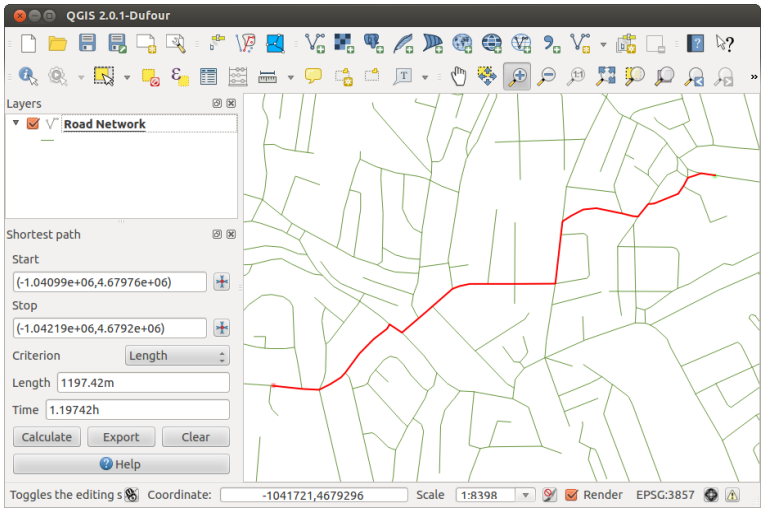


3.2. Shortest path

The shortest path problem is one of the classical problems in graph theory that is due to its practical importance.

Indeed, we constantly come across the search for optimal paths. For example, GPS-navigators search for a time-minimum path between two objects.

Example. Q-GIS.



Shortest path

Shortest paths problems

- The **single-pair shortest path problem**, in which we have to find shortest paths from a source vertex *v* to a single destination vertex *u*.
- The **single-source shortest path problem**, in which we have to find shortest paths from a source vertex *v* to all other vertices in the graph.
- The **single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex *v*.
- The **all-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices *v*, *u* in the graph.

3.2.1. Single-source shortest path problem

Here we consider the single-source shortest path problem for the following types of graphs:

- an unweighted graph (Lee algorithm);
- a weighted graph with positive weights (Dijkstra algorithm);
- a weighted graph with arbitrary weights (Dijkstra modified algorithm, Bellman-Ford algorithm).

Single-source shortest path problem in unweighted graphs

In an unweighted graph, the **shortest** path <*u*,*v*> is a path of minimum length | <*u*,*v*> |.

Lee algorithm (based on the BFS) is usually used to find the shortest path.

Lee algorithm was proposed by C.Y.Lee in 1961 as a connection routing method in electronic design automation, and it is still widely used for finding the shortest paths in graphs.

It uses a **wave propagation** style; a **wave *k*** are all vertices that can be reached in *k* steps from the start vertex. The wave stops when the target is reached, and the path is determined by backtracking through the vertices.

Lee algorithm

- *Start.* Given a graph $G(V,E)$ and start the vertex x, find the shortest path $<x,v>$ for every vertex v reachable from x.
- *Step 1.* Set the wave number $k=0$ and label the vertex x by 0. Other vertices are unlabeled.
- *Step 2.* For all the vertices labeled by k, label their unlabeled out-neighbors by $k+1$.
- *Step 3.* If there are vertices labeled by $k+1$, set $k=k+1$ and go to Step 2.
- *Step 4*. If $k=0$, go to the end. Else for all labeled vertices, determine the shortest paths.

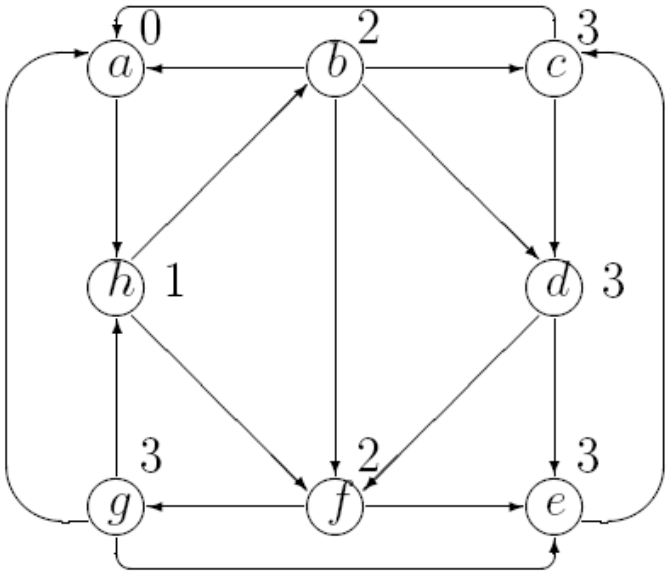
Lee algorithm

- *Step 5.* For the vertices labeled by 1, the shortest paths consist of one edge: $<x,v>=xv$. Set $m=1$.
- *Step 6.* If there are no vertices labeled by $m+1$ go to the end.
- *Step 7.* For every vertex y labeled by $m+1$, determine the preceding vertex z in the shortest path. The vertex z is labeled by m, and $(z,y)\in E$. The shortest path $<x,y>$ is constructed by adding the vertex y to the shortest path $<x,z>$.
- *Step 8.* Set $m=m+1$ and go to Step 6.
- *End.* All the vertices reachable from x are labeled; the label is the length of the shortest path from the vertex x. The shortest paths have been constructed.

Lee algorithm

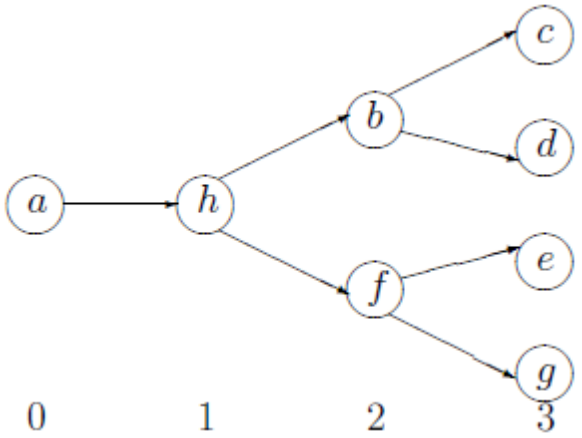
Example. Here numbers near the vertices are labels. For the vertices {*b*,*f*} labeled by 2, the preceding vertex in the shortest path should be labeled by 1. There is the only vertex *h* labeled by 1, and there are edges (*h*,*b*) and (*h*,*f*). We add *b* and *f* to path <*a*,*h*> and obtain the shortest paths

- <*a*,*b*>=*ahb*;
- <*a*,*f*>=*ahf*.



Lee algorithm

Example. We can demonstrate the execution of the algorithm at the diagram. Vertices with the same wave number are in the same vertical line; the wave number increases from left to right. The diagram also shows the shortest paths.



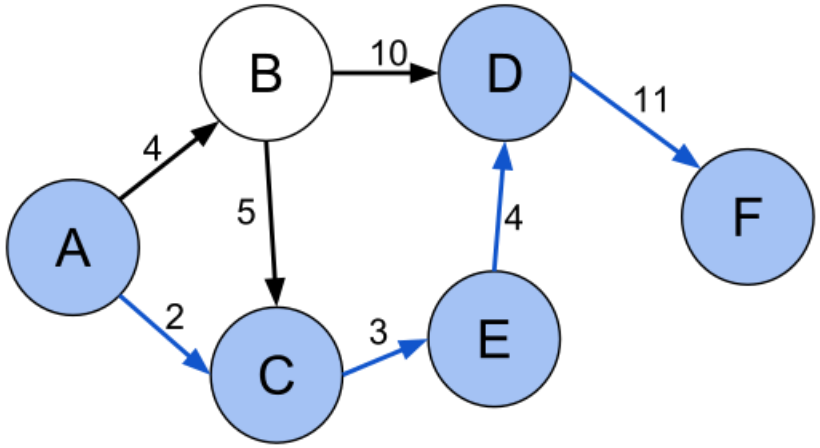
Single-source shortest path problem in weighted graphs

A **weighted graph** associates a label (**weight**) with every edge in the graph.

The **weight** of a path W(<u,v>) is the sum of weights of the edges included in the path.

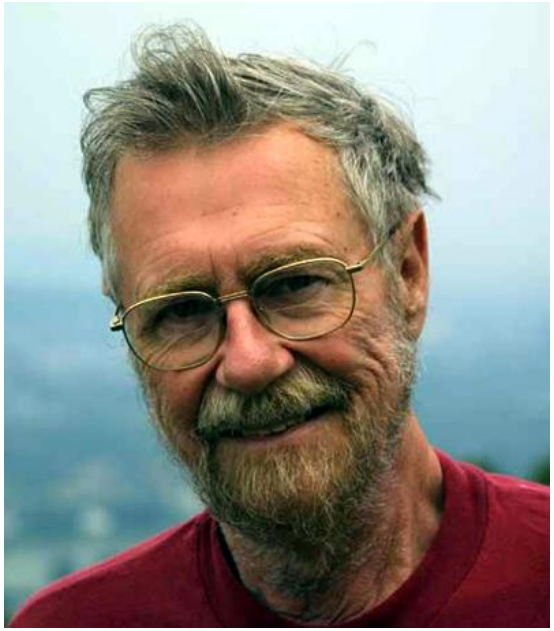
The **shortest** path <u,v> in a weighted graph is a path of minimal weight W(<u,v>) .

Example.



Dijkstra algorithm

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with **nonnegative edge path weights**. This algorithm is often used in routing and as a subroutine in other graph algorithms.



Dijkstra algorithm

- *Start.* Given a graph $G(V, E)$ with the weight matrix *W* and start the vertex x, find the shortest path $\langle x, v \rangle$ for every vertex v reachable from x.
- *Step 1.* Label the vertex x by 0 and other vertices by infinity. Put all the vertices into the priority queue *Q* and mark them as unvisited.
- *Step 2.* If the priority queue is empty go to Step 5.
- *Step 3.* Extract the vertex *t* with the minimum label from the queue *Q* (the vertex leaves the queue). Mark it as visited.
- *Step 4*. For all unvisited out-neighbors of the vertex *t*, update the labels:

$$\lambda(v) = min\{\lambda(v), \lambda(t) + w_{t,v}\}.$$

Then go to Step 2.

Dijkstra algorithm

- *Step 5.* For the vertices with finite labels, construct the shortest paths. Vertex y precedes vertex v in the shortest path if

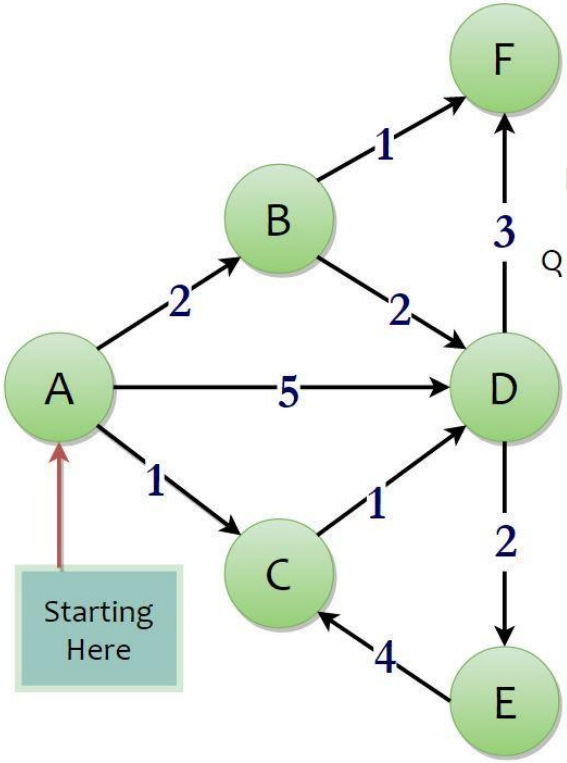
$$\lambda(y) + w_{y,v} = \lambda(v).$$

  The process is finished if $y = x$.

- *End.* All the vertices reachable from *x* are labeled by finite values; the label is the weight of the shortest path from the vertex *x*. The shortest paths have been constructed.

Dijkstra algorithm

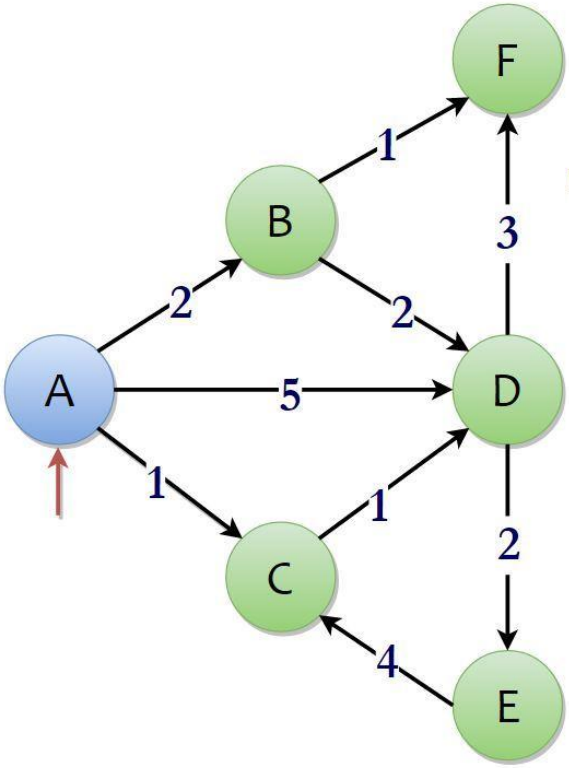
Example. http://theoryofprogramming.com/2015/01/11/dijkstras-algorithm/



Initially, D[A] = 0
D[B] = D[C] = D[D] = D[E] = D[F] = ∞
So, D = {0, ∞, ∞, ∞, ∞, ∞}
Q = {A(0), B(∞), C(∞), D(∞), E(∞), F(∞)}
The priorities in Q, are given in parenthesis

Dijkstra algorithm

Example.

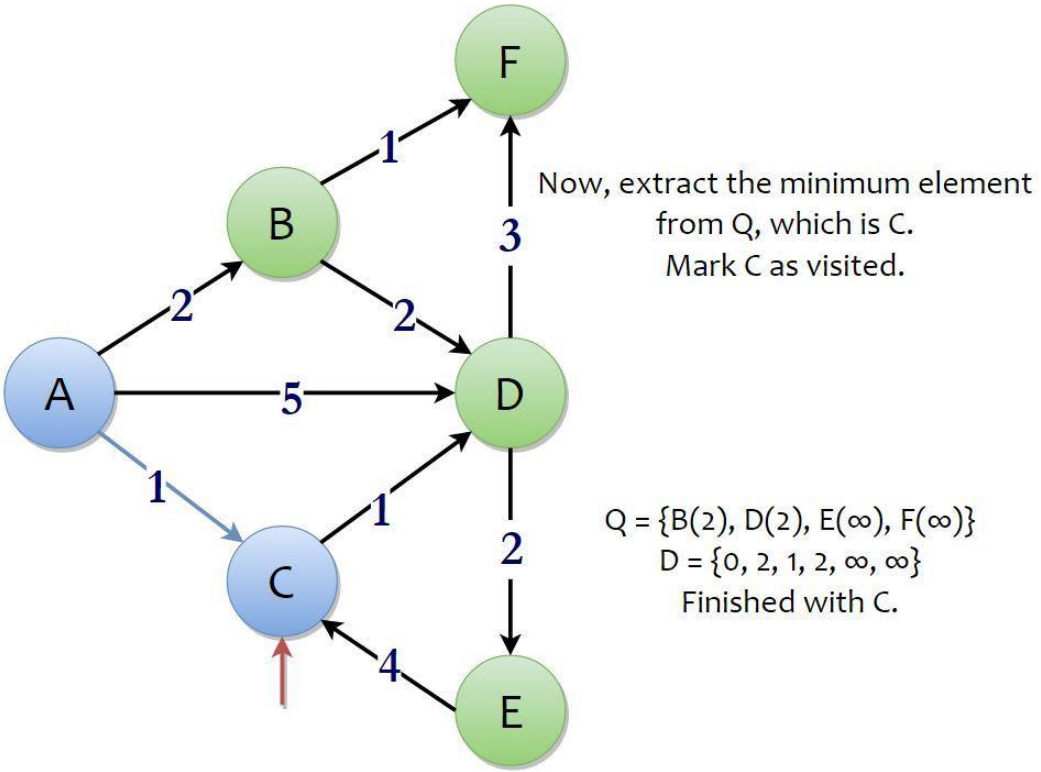


Now, extract the minimum element
from Q, which is A.
Mark A as visited.

Q = {B(2), C(1), D(5), E(∞), F(∞)}
D = {0, 2, 1, 5, ∞, ∞}
Finished with A

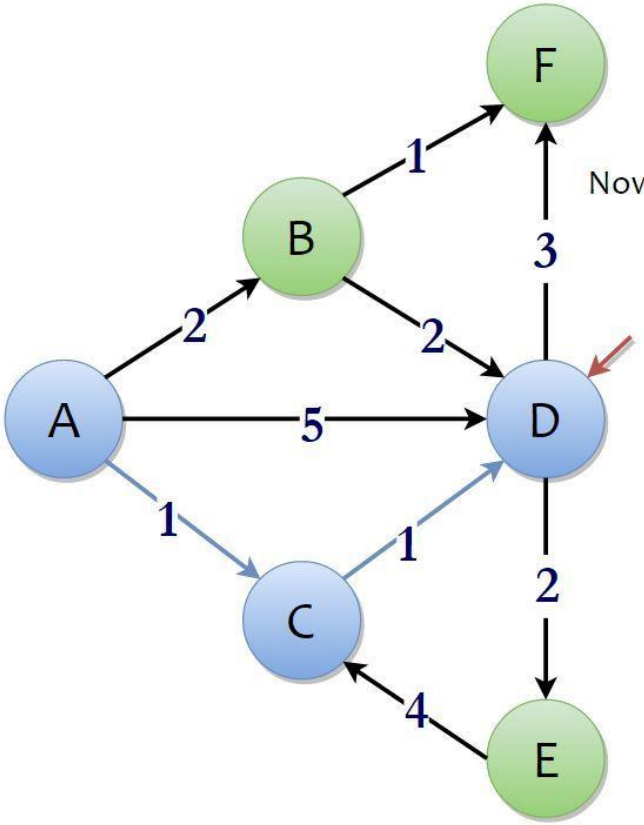
Dijkstra algorithm

Example.



Dijkstra algorithm

Example.

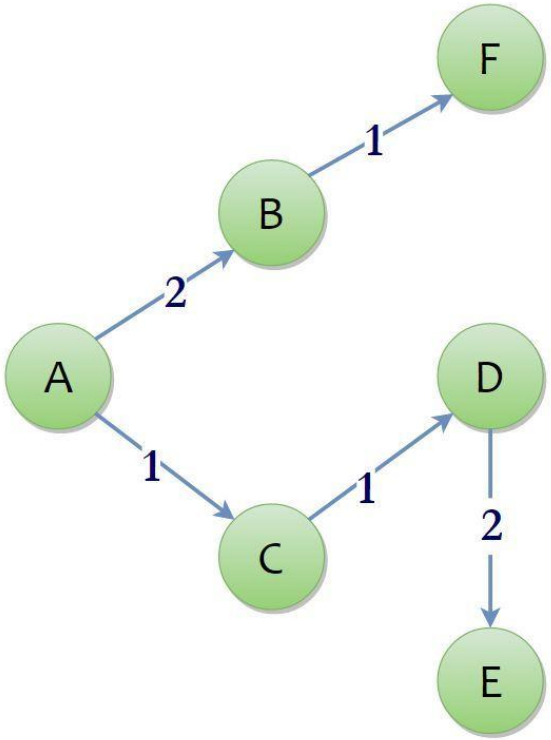


Now, extract the minimum element from Q, which are B and D.
Let's take D.
Mark D as visited.

Q = {B(2), E(4), F(5)}
D = {0, 2, 1, 2, 4, 5}
Finished with D

Dijkstra algorithm

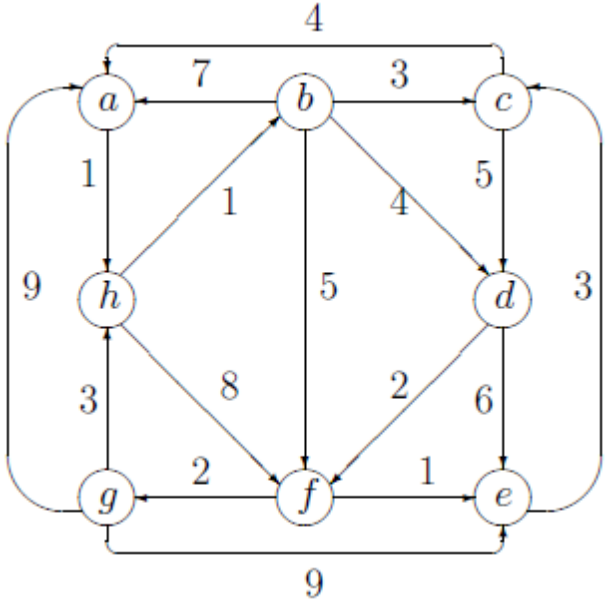
Example. Proceeding similarly…



If we keep only those edges that contribute to the shortest paths, we get the Shortest Path Tree

Table implementation

Example. Here v is the vertex extracted from the priority queue, $\lambda(v)$ is its label, columns $a - h$ contain current labels of the corresponding unvisited vertices.



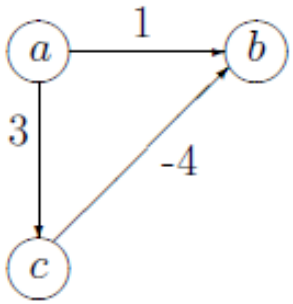
i	v	$\lambda(v)$	a	b	c	d	e	f	g	h
0			**0**	∞	∞	∞	∞	∞	∞	∞
1	a	0		∞	∞	∞	∞	∞	∞	**1**
2	h	1		**2**	∞	∞	∞	9	∞	
3	b	2			**5**	6	∞	7	∞	
4	c	5				**6**	∞	7	∞	
5	d	6					12	**7**	∞	
6	f	7					**8**		9	
7	e	8							**9**	
8	g	9								

Negative weights?

For graphs with negative weights of edges, Dijkstra algorithm does not work.

Example. Starting with vertex a, we mark it as visited. The labels $\lambda(b) = 1$ and $\lambda(c) = 3$. Then we extract the vertex b from the queue and mark it as visited. At the next iteration, we extract the vertex c. As we do not update labels for visited vertices, the label of the vertex b remains $\lambda(b) = 1$, in spite of

$$W(acf) = 3 - 4 = -1.$$



Modified Dijkstra algorithm

Step 4 is changed.

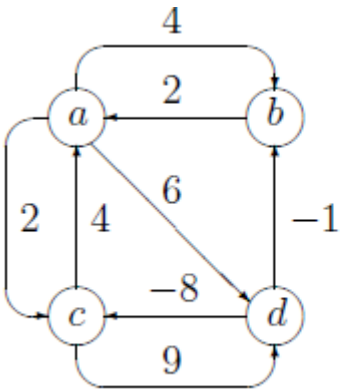
- *Step 4*. For **all** out-neighbors of the vertex t, update the labels:

$$\lambda(v) = min\{\lambda(v), \lambda(t) + w_{t,v}\}.$$

If for a visited vertex, the new label is less than the old one mark the vertex as unvisited and put it in the priority queue. Then go to Step 2.

Modified Dijkstra algorithm

Example. Here the visited vertices are written in *italic*, the vertex with the minimum weight from the queue is written **in bold**.



i	v	$\lambda(v)$	a	b	c	d
0			**0**	∞	∞	∞
1	a	0	*0*	4	**2**	6
2	c	2	*0*	**4**	*2*	6
3	b	4	*0*	*4*	*2*	**6**
4	d	6	*0*	*4*	**−2**	*6*
5	c	−2	*0*	*4*	*−2*	*6*

Bellman-Ford algorithm

Bellman-Ford algorithm is an example of the **dynamic programming method**. DP can be used:

- If the given problem can be broken up in to smaller subproblems and these smaller subproblems are in turn divided in to smaller ones, and in this process, if you observe some overlapping subproblems;
- If the optimal solutions to the subproblems contribute to the optimal solution of the given problem.

Bellman-Ford algorithm

In the shortest path problem:

- Any shortest path consists of shortest paths; i.e., if

$$< x, y > = x \ldots z \ldots y$$

then it contains the shortest paths $< x, z >$ and $< z, y >$.

- The weight of the shortest paths it the sum of the weightf of its subpaths; i.e.

$$W(x, y) = W(x, z) + W(z, y).$$

Bellman-Ford algorithm

- The algorithm first calculates the shortest distances which have at-most one edge in the path.
- Then, it calculates the weights of the shortest paths with at-most 2 edges, and so on. After the k-th iteration of outer loop, the weights of the shortest paths with at most k edges are calculated.
- Assuming that there is no negative weight cycle, if we have calculated shortest paths with at most k edges, then an iteration over all edges guarantees to give shortest path with at-most $k + 1$ edges.
- There can be maximum $p - 1$ edges in any simple path, that is if after the $(p - 1)$-th iteration the weight of the paths still change then the graph contains a negative-weight cycle.

Bellman-Ford algorithm

- *Start.* Given a graph $G(V, E)$ with the weight matrix W and start the vertex x, find the shortest path $\langle x, v \rangle$ for every vertex v reachable from x.
- *Step 1.* Label the vertex x by 0 and other vertices by infinity. Set the iteration number $k = 0$.
- *Step 2.* If $k = p$ then the graph contains a negative-weight cycle, go to the end. Else increase the iteration number $k = k + 1$.
- *Step 3.* For all edges (u, v) update the label $\lambda(v)$:

$$\lambda(v) = min\{\lambda(v), \lambda(u) + w_{u,v}\}.$$

Bellman-Ford algorithm

- *Step 4*. If some labels were changed then go to Step 2.
- *Step 5.* All the vertices reachable from x are labeled by finite values; the label is the weight of the shortest path from the vertex x. For these vertices, construct the shortest paths. Vertex y precedes vertex v in the shortest path if

$$\lambda(y) + w_{y,v} = \lambda(v).$$

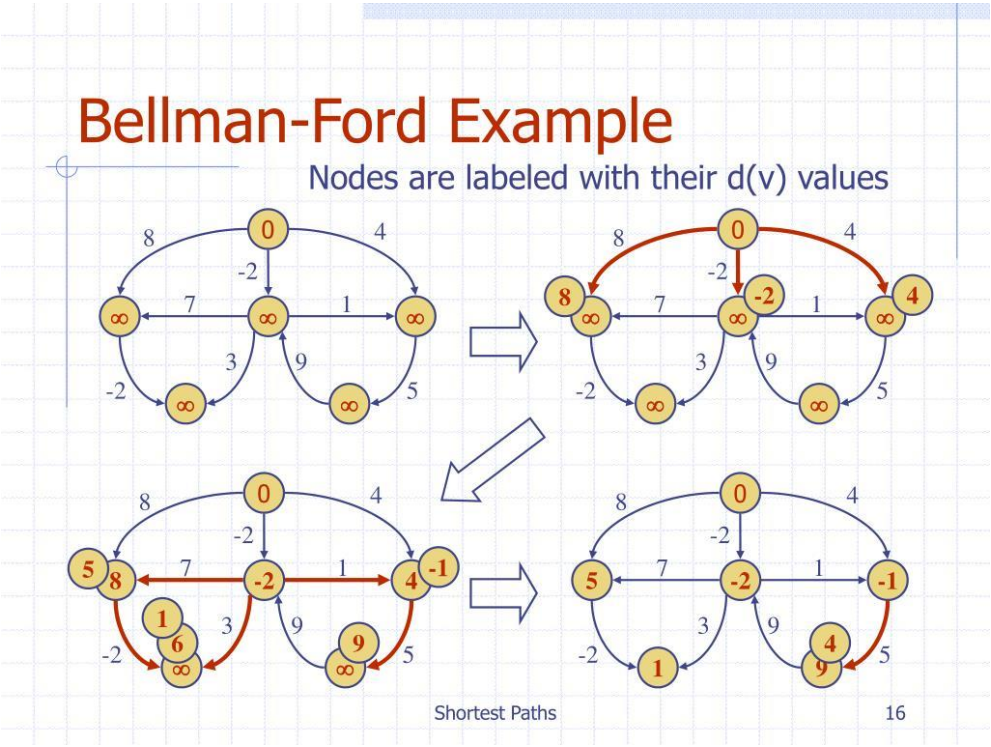
  The process is finished if $y = x$.
- *End.*

Bellman-Ford algorithm

Example.

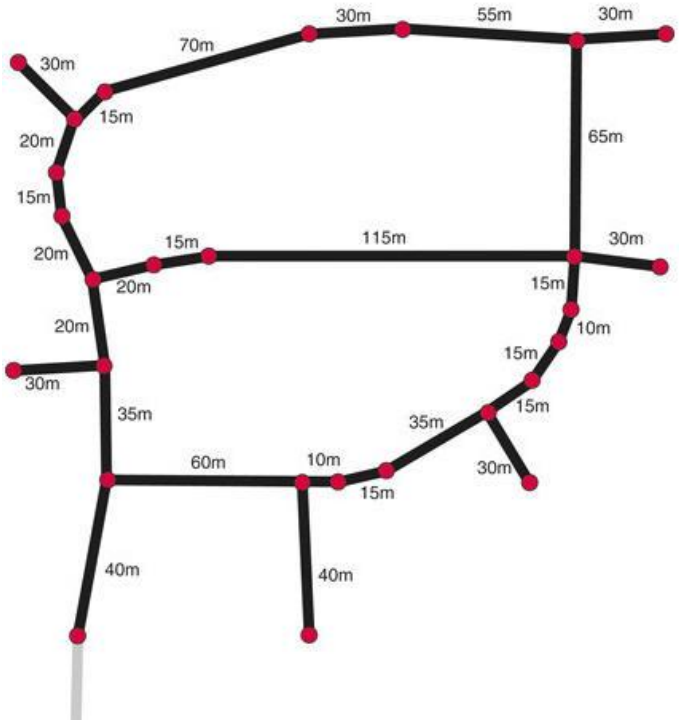
http://www.programming-algorithms.net/article/47389/Bellman-Ford-algorithm

https://image.slideserve.com/473818/bellman-ford-example-l.jpg



K shortest paths

- What if we need more than one path?
- **Yen's algorithm** computes single-source *K*-shortest loopless paths for a graph with non-negative edge cost.
- The algorithm was published by Jin Y. Yen in 1971 and employs any shortest path algorithm to find the best path, then proceeds to find *K* − 1 deviations of the best path.



Yen’s algorithm

- What if we need more than one path?
- **Yen's algorithm** computes single-source *K*-shortest loopless paths for a graph with non-negative edge cost.
- The algorithm was published by Jin Y. Yen in 1971 and employs any shortest path algorithm to find the best path, then proceeds to find $K - 1$ deviations of the best path.