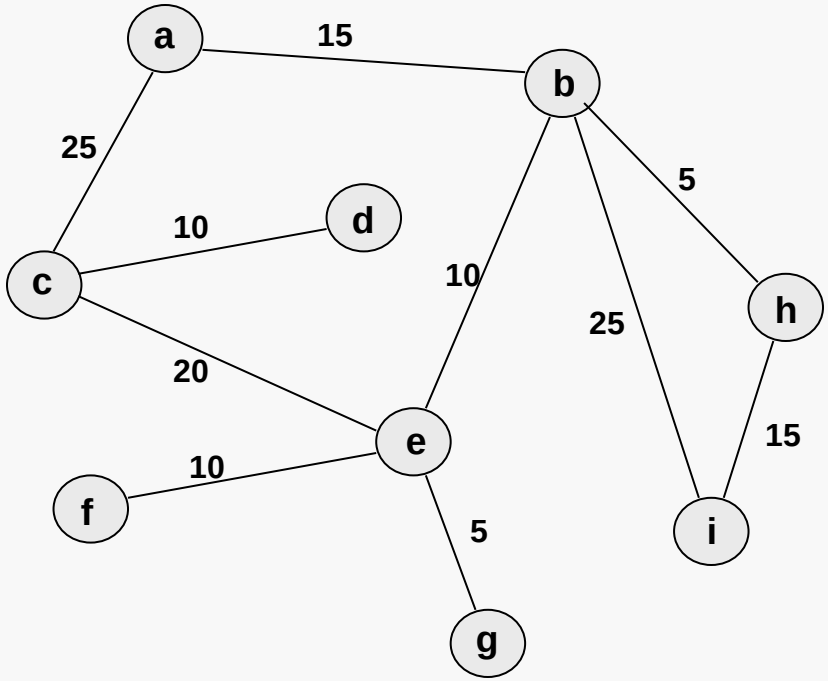
Weighted Graphs

In many applications, each edge of a graph has an associated numerical value, called a weight.

Usually, the edge weights are non-negative integers.

Weighted graphs may be either directed or undirected.

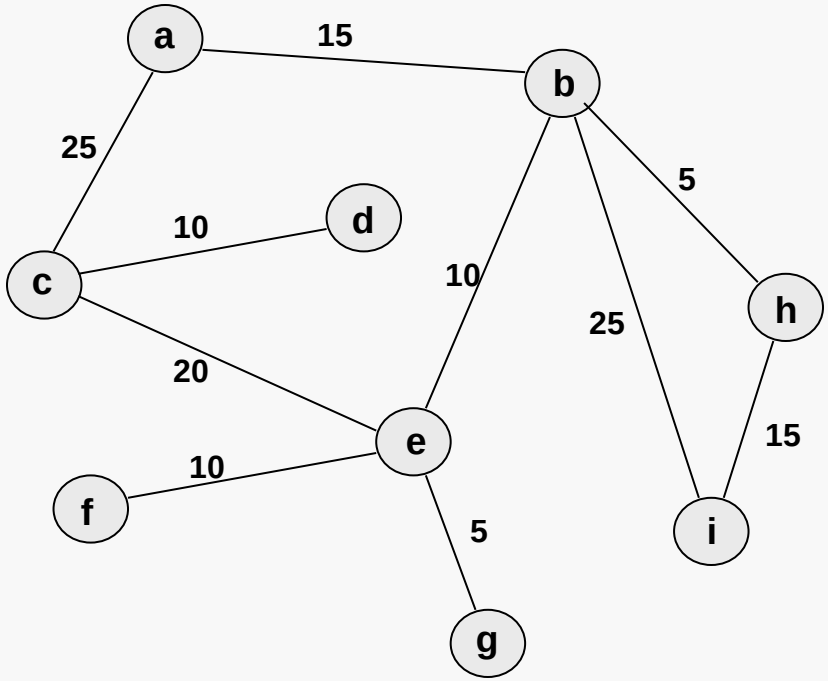


The weight of an edge is often referred to as the "cost" of the edge.

In applications, the weight may be a measure of the length of a route, the capacity of a line, the energy required to move between locations along a route, etc.

Shortest Paths (SSAD)

Given a weighted graph, and a designated node S, we would like to find a path of least total weight from S to each of the other vertices in the graph.



The total weight of a path is the sum of the weights of its edges.

We have seen that performing a DFS or BFS on the graph will produce a spanning tree, but neither of those algorithms takes edge weights into account.

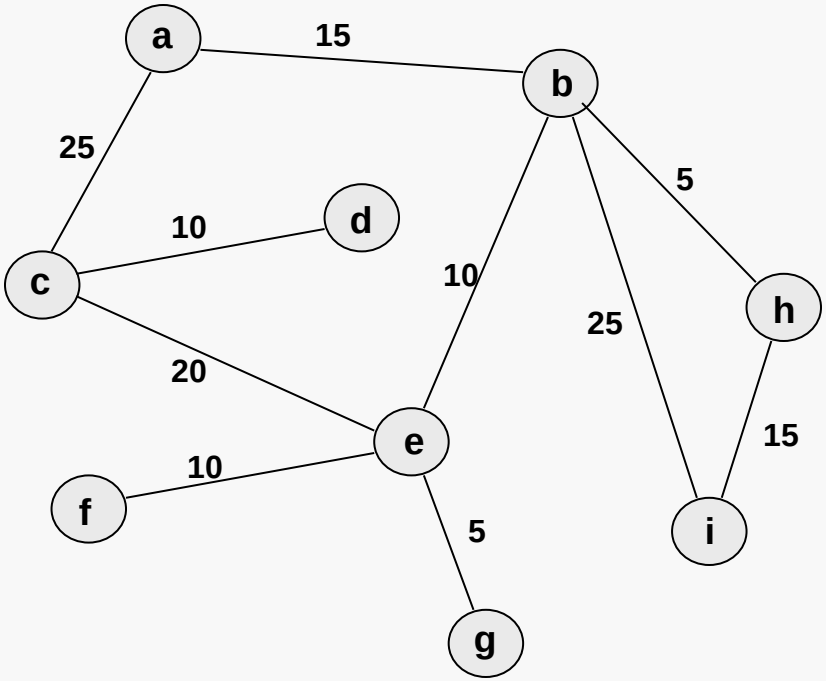
There is a simple, greedy algorithm that will solve this problem.

Dijkstra's SSAD Algorithm*

We assume that there is a path from the source vertex *s* to every other vertex in the graph.

Let S be the set of vertices whose minimum distance from the source vertex has been found. Initially S contains only the source vertex.

The algorithm is iterative, adding one vertex to S on each pass.



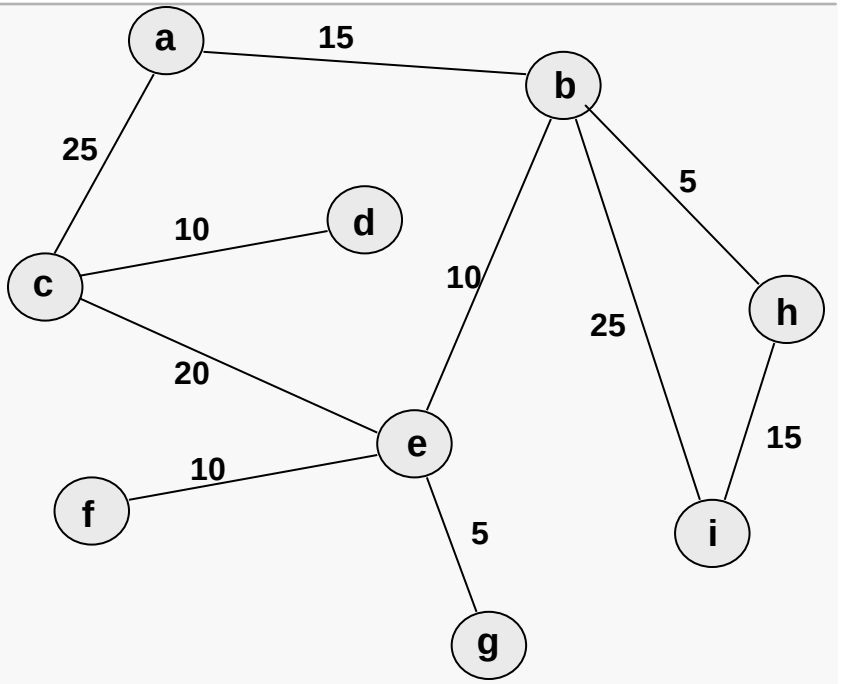
We maintain a table D such that for each vertex v, D(v) is the minimum distance from the source vertex to v via vertices that are already in S (aside possibly from v itself).

Greed: on each iteration, add to S the vertex v not already in S for which D(v) is minimal.

*1959

Dijkstra's Algorithm Trace

Let the source vertex be a.



S = {a}

D

a	b	c	d	e	f	g	h	i
0	15	25	∞	∞	∞	∞	∞	∞

S = {a,b}

D

a	**b**	c	d	e	f	g	h	i
0	15	25	∞	25	∞	∞	20	40

Dijkstra's Algorithm Trace

Continuing:

S = {a, b, h}

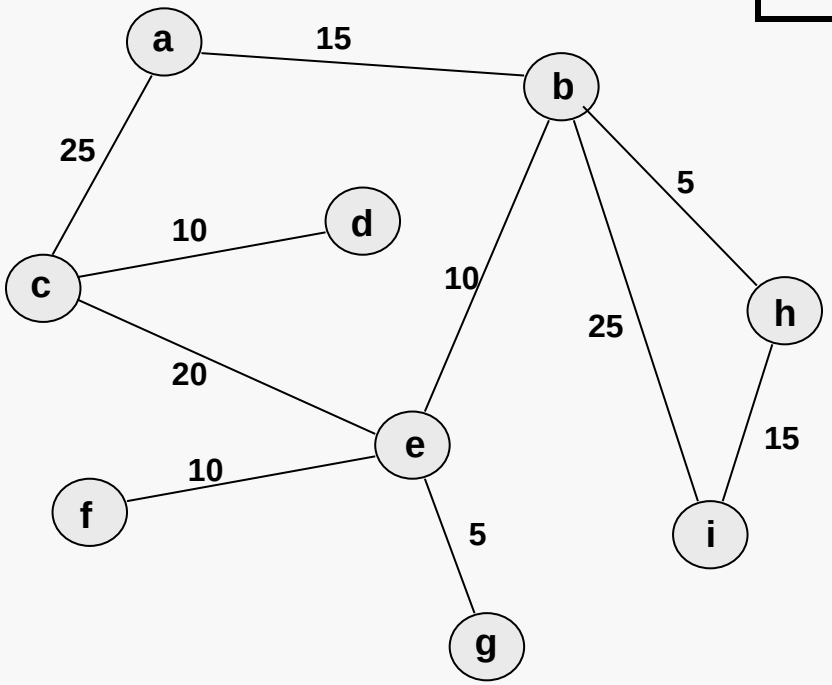
D

a	**b**	c	d	e	f	g	**h**	i
0	15	25	∞	25	∞	∞	20	35

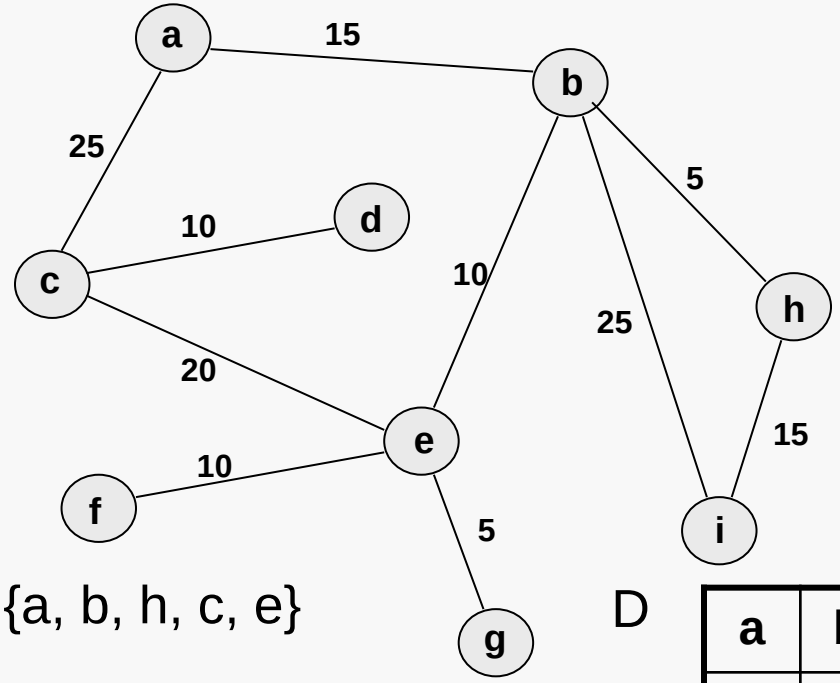
S = {a, b, h, c}

D

a	**b**	**c**	d	e	f	g	**h**	i
0	15	25	35	25	∞	∞	20	35



Dijkstra's Algorithm Trace



S = {a, b, h, c, e} D

a	**b**	**c**	d	**e**	f	g	**h**	i
0	15	25	35	25	35	30	20	35

S = {a, b, h, c, e, g} D

a	**b**	**c**	d	**e**	f	**g**	**h**	i
0	15	25	35	25	35	30	20	35

S = {a, b, h, c, e, g, f} D

a	**b**	**c**	d	**e**	**f**	**g**	**h**	i
0	15	25	35	25	35	30	20	35

Dijkstra's Algorithm Trace

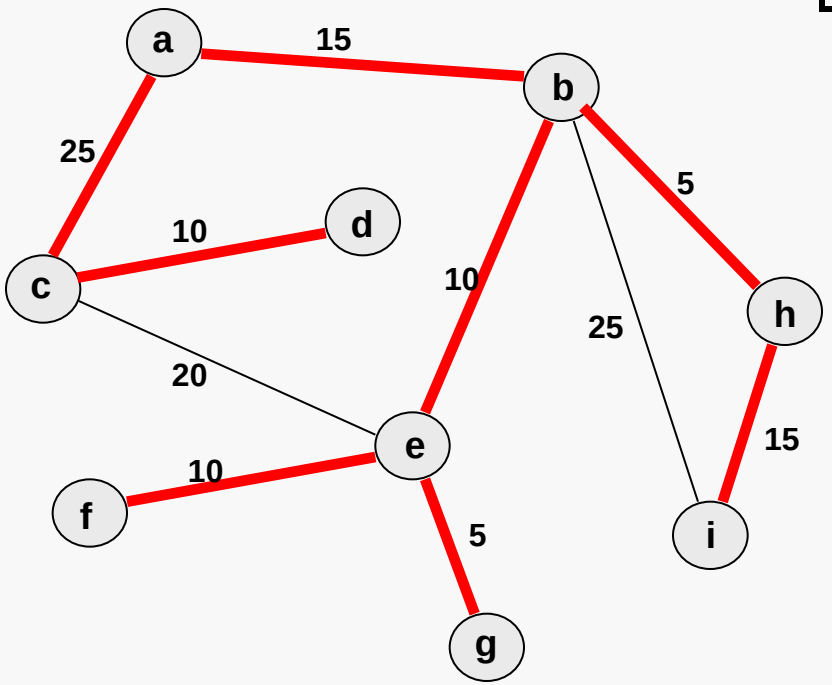
Continuing:

S = {a, b, h, c, e, g, f, d} D

a	b	c	d	e	f	g	h	i
0	15	25	35	25	35	30	20	35

S = {a, b, h, c, e, g, f, d, i} D

a	b	c	d	e	f	g	h	i
0	15	25	35	25	35	30	20	35



The corresponding tree is shown at left. As described, the algorithm does not maintain a record of the edges that were used, but that can easily be remedied.

Limitations

Dijkstra's SSAD Algorithm only works for graphs with non-negative weights.

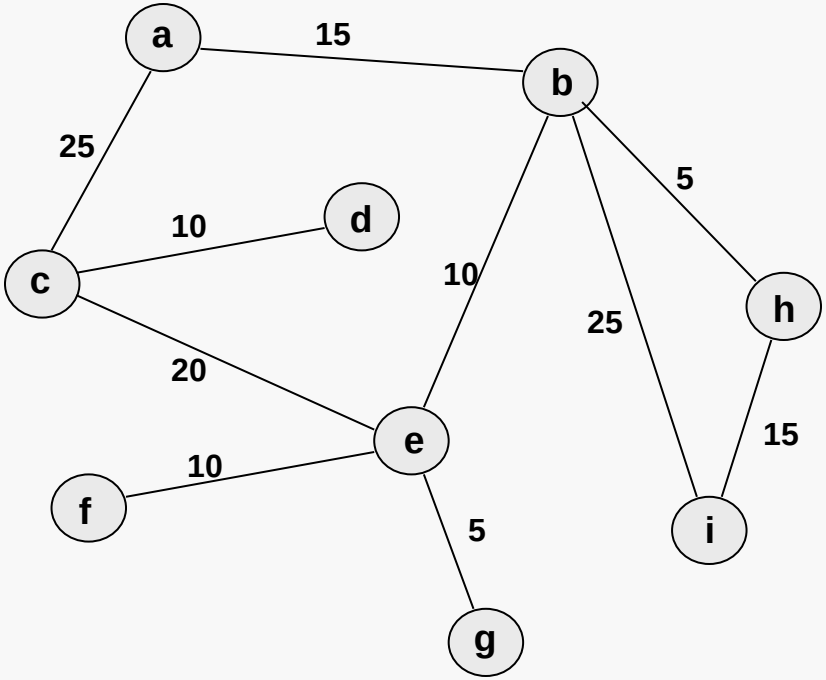
See the Bellman-Ford Algorithm, which works even if the weights are negative, provided there is no *negative cycle* (a cycle whose total weight is negative).

Minimal Spanning Tree

Given a weighted graph, we would like to find a spanning tree for the graph that has minimal total weight.

The total weight of a spanning tree is the sum of the weights of its edges.

We want to find a spanning tree T, such that if T' is any other spanning tree for the graph then the total weight of T is less than or equal to that of T'.



Jarnik-Prim MST Algorithm

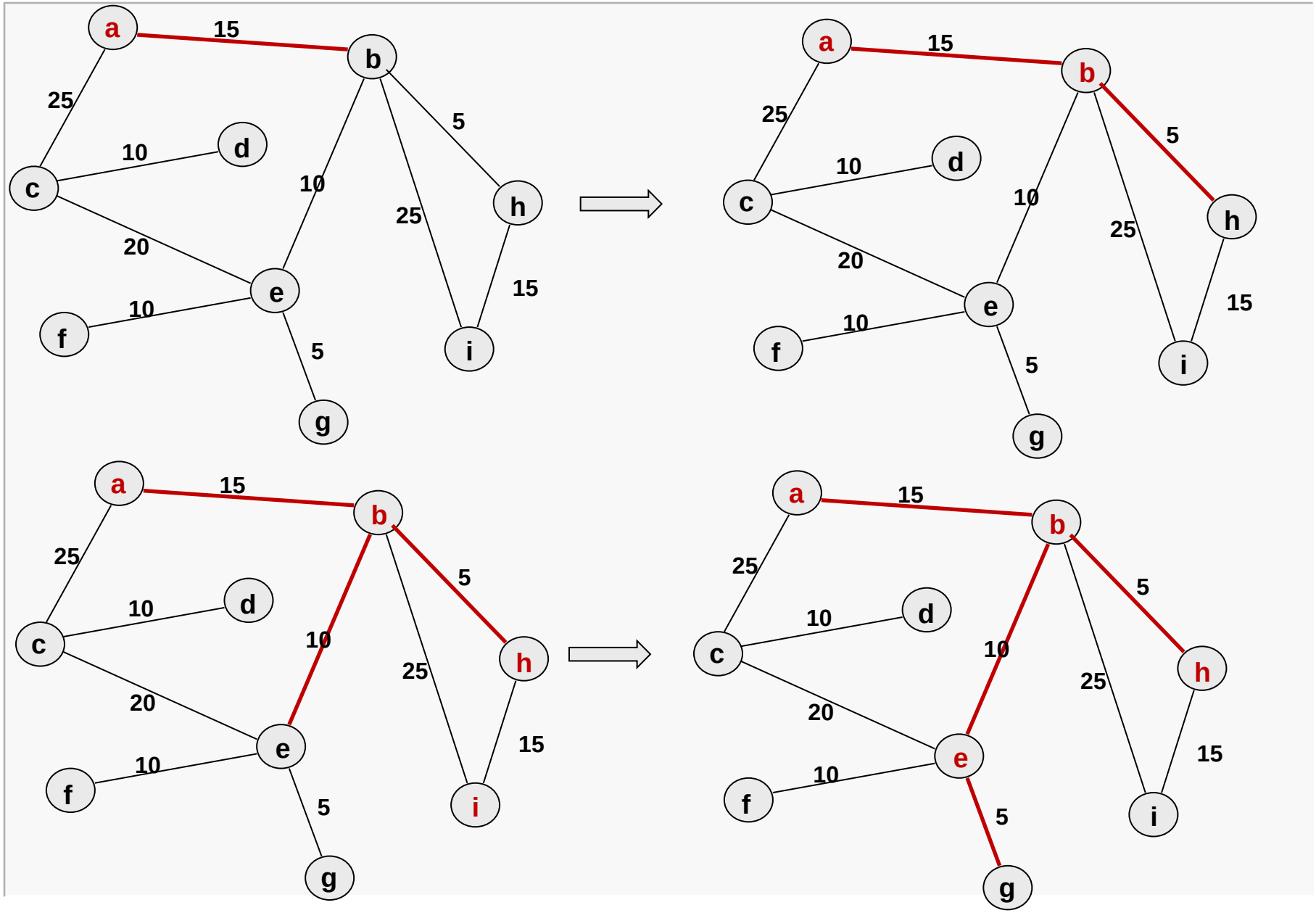
By modifying Dijkstra's SSAD Algorithm to build a list of the edges that are used as vertices are added, and storing the distance from nodes to the current tree (rather than from nodes to the source) we obtain Prim's Algorithm (V Jarnik, 1930 and R C Prim, 1957).

It turns out that this algorithm does, in fact, create a spanning tree of minimal weight if the graph to which it is applied is connected.

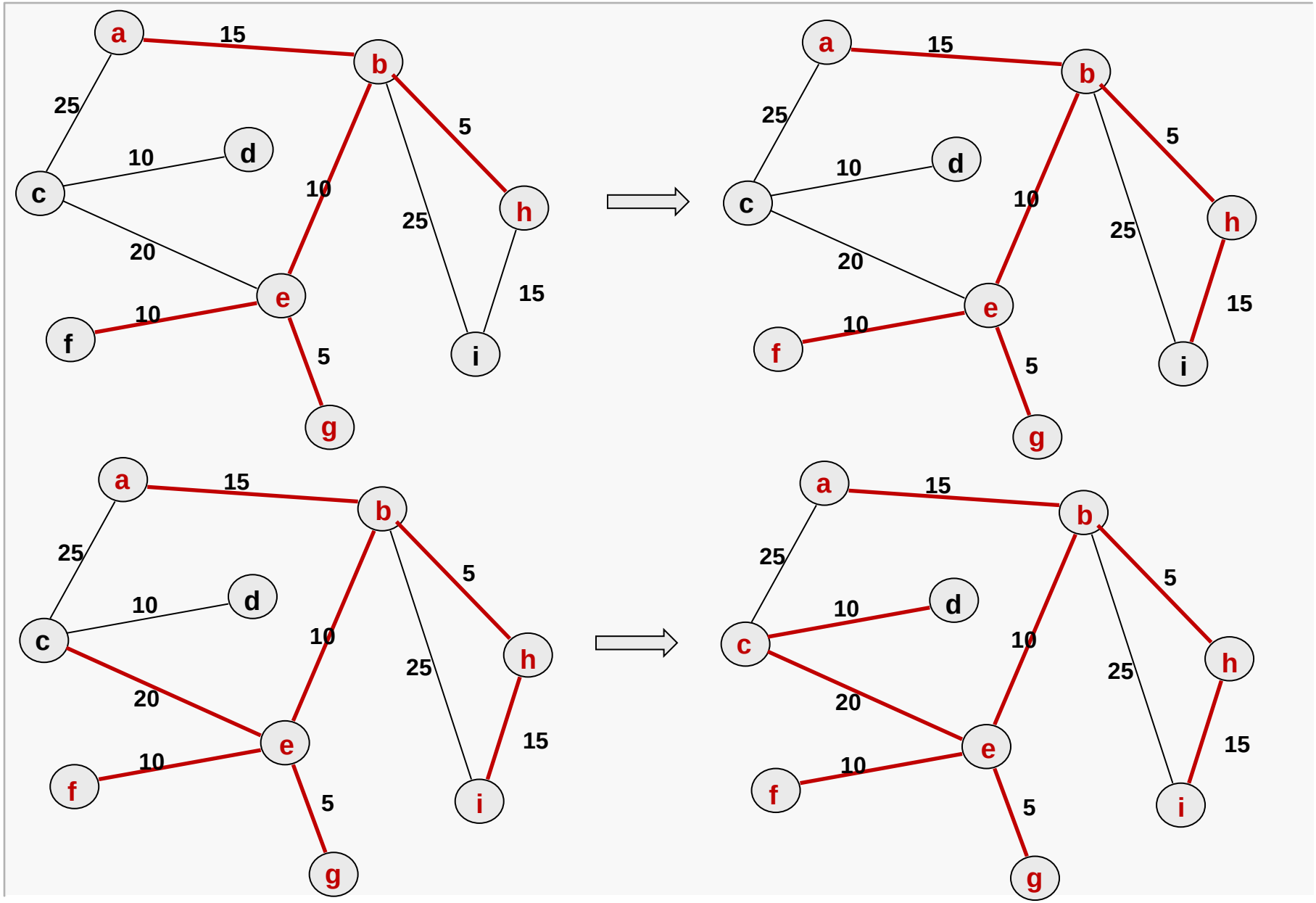
~~Since the complex steps in Prim's algorithm are the same as Dijkstra's, no detailed example is traced here.~~

QTP: why does Dijkstra's SSAD Algorithm not necessarily find a minimum-weight spanning tree?

Jarnik-Prim MST Algorithm Trace



Jarnik-Prim MST Algorithm Trace



Jarnik-Prim MST Algorithm Trace

A	B	C	D	E	F	G	H	I
0	15	25	inf	inf	inf	inf	inf	inf
	A	25	inf	10	inf	inf	5	25
		25	inf	10	inf	inf	B	15
		20	inf	B	10	5		15
		20	inf		10	E		15
		20	inf		E			15
		20	inf					H
		E	10					
			C