Dijkstra’s

Dijkstra’s Shortest Path:

- Given: weighted graph, *G*, and source vertex, *v*

- Compute: shortest path to every other vertex in *G*

- Path length is sum of edge weights along path. Shortest path has smallest length among all possible paths

Algorithm:

Grow a collection of vertices for which shortest path is known

- paths contain only vertices in the set

- add as new vertex the one with the smallest distance to the source

- shortest path to an outside vertex must contain a current shortest path as a prefix Use a greedy algorithm

Edge Relaxation:

- Maintain value D[*u*] for each vertex
  - Each starts at infinity, and decreases as we find out about a shorter path from *v* to *u* (D[*v*] = 0)

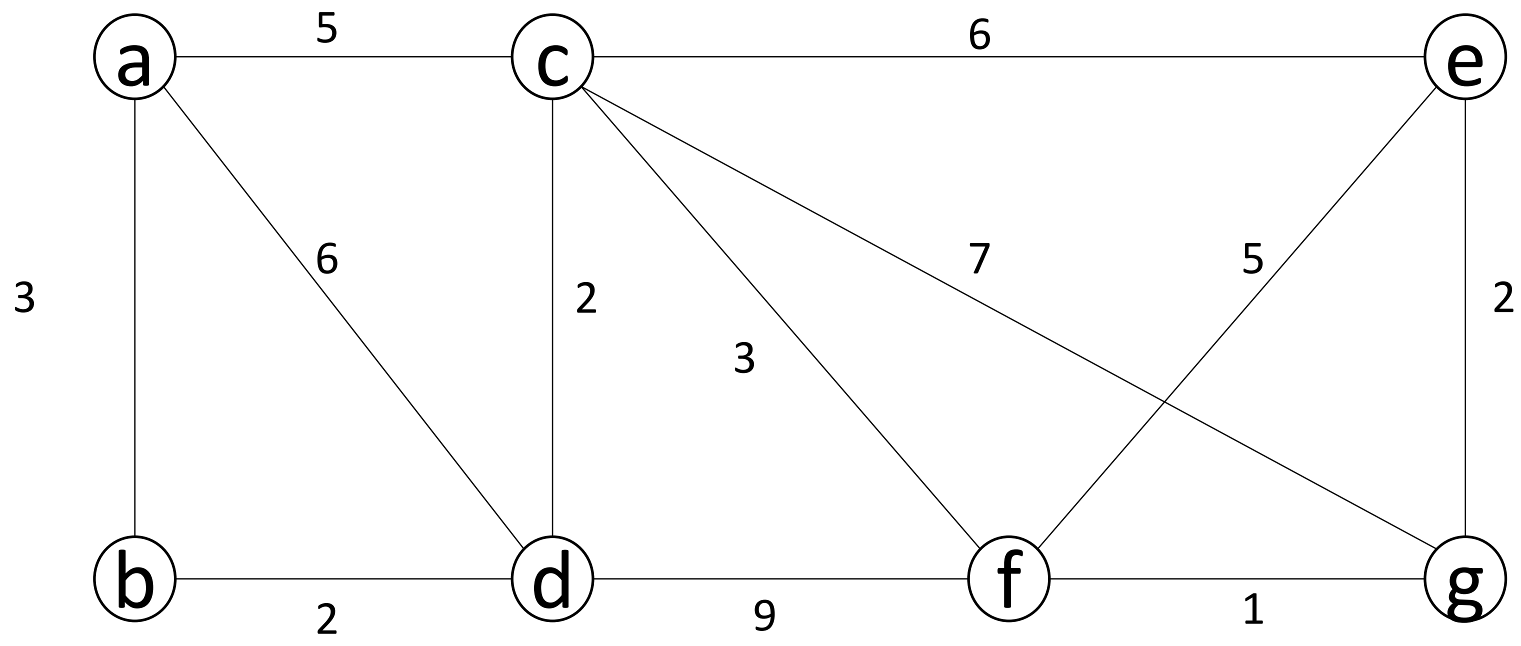
- Maintain priority queue, *Q*, of vertices to be relaxed
  - use D[*u*] as key for each vertex
  - remove min vertex from *Q*, and relax its neighbors

- Relaxation for each neighbor of u:
  - If D[*u*] + w(*u,z*) < D[z] then, D[z] = D[*u*] + w(*u,z*)

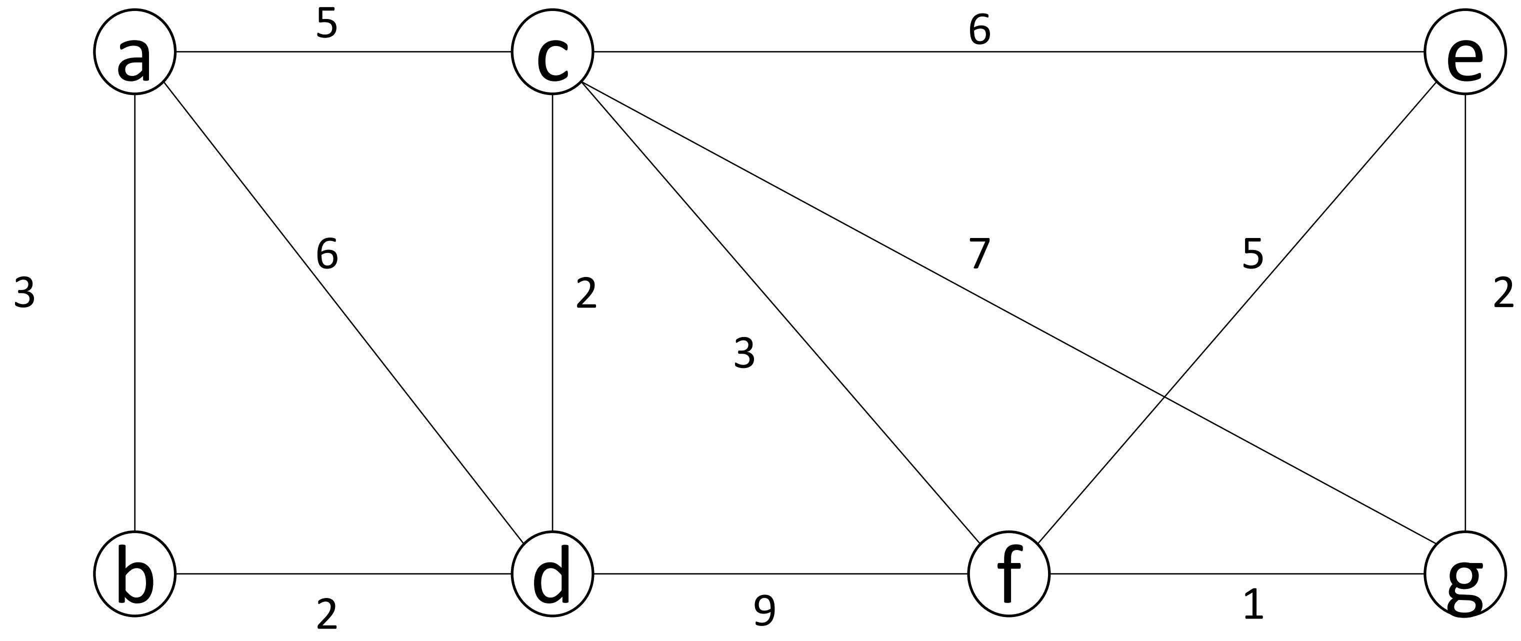
Pseudocode:

- ShortestPath(*G*, *v*)
  - init D array entries to infinity D[*v*]=0
  - add all vertices to priority queue *Q* while *Q* not empty do *u* = *Q*.removeMin()
  - for each neighbor, *z*, of *u* in *Q* do if D[*u*] + w(*u*,*z*) < D[*z*] then D[*z*] = D[*u*] + w(*u*,*z*)
    - Change key of *z* in *Q* to D[*z*]
    - return D as shortest path lengths

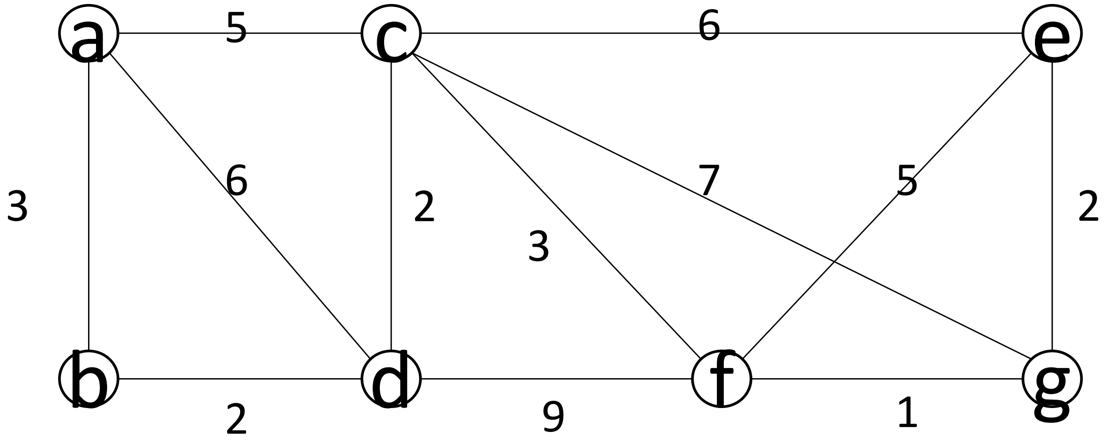
Worked Example:



Step 1: Draw a table with the set of vertices (v)

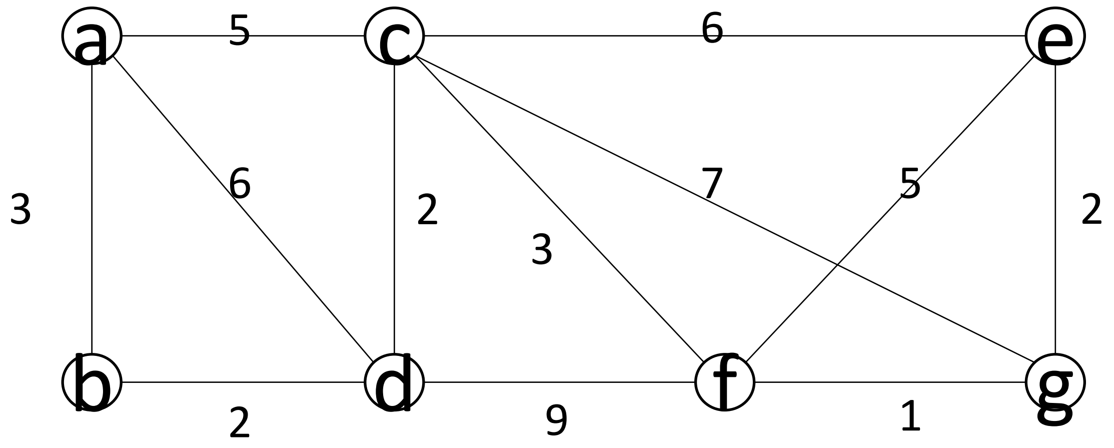


Step 1: Draw a table with the set of vertices (v)



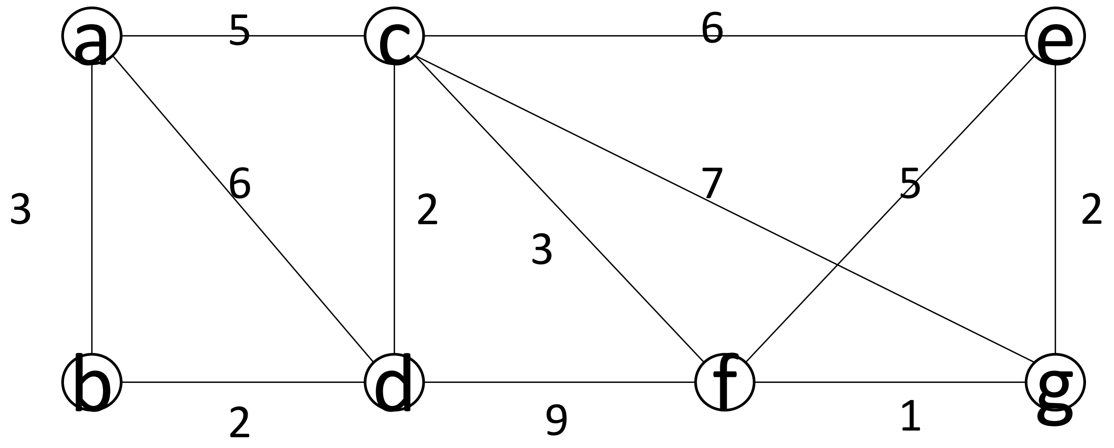
V	a	b	c	d	e	f	g
a							

Step 2: Mark all vertices starting from ‘a’. Subscript denotes the vertex we connect from.



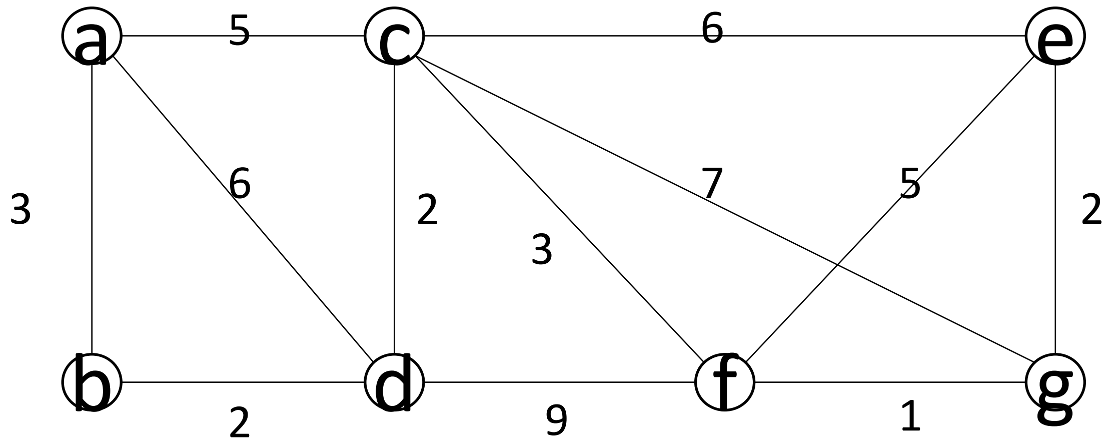
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a

Step 3: The lowest weight is marked red (shortest weight determined) and the next lowest weight in the row is looked for – "3a" in this case which is the weight from 'a' to 'b'. So, the next row in the table starts with 'b' and "3a" marked red.



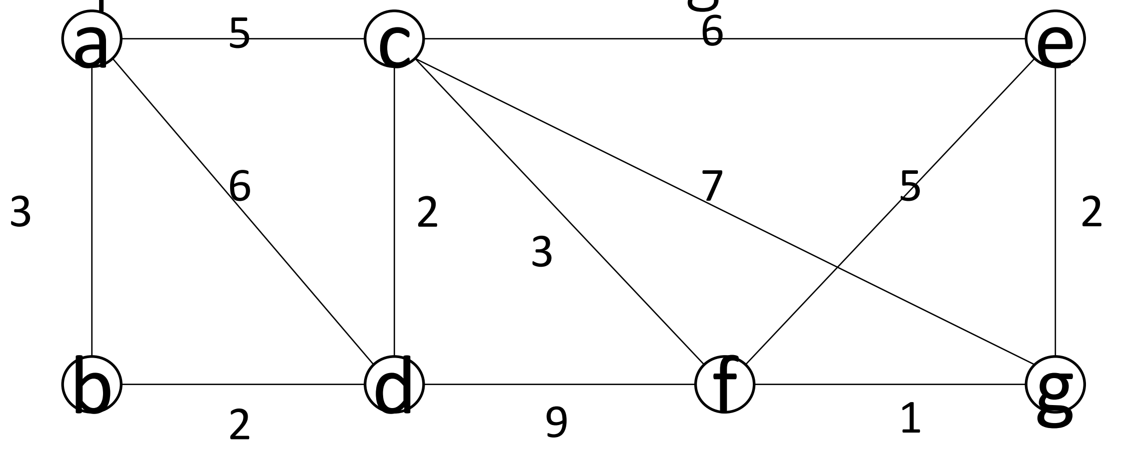
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a

Step 3: The lowest weight is marked red (shortest weight determined) and the next lowest weight in the row is looked for – "3a" in this case which is the weight from 'a' to 'b'. So, the next row in the table starts with 'b' and "3a" marked red.



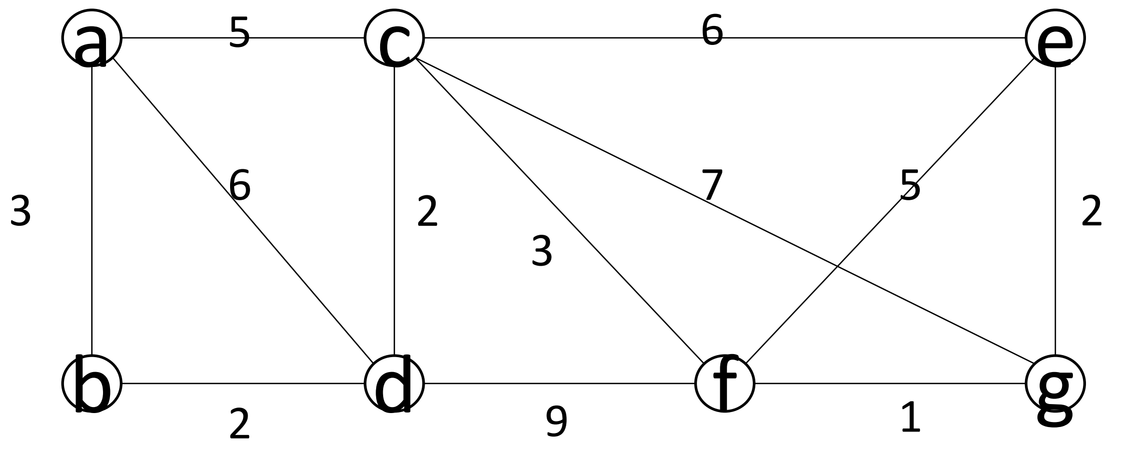
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a					

Step 3: The next adjacent vertex from b is looked for and compared with the weight when directly reached from a – "6a". When coming via vertex b, the total weight is a->b("3a")+b->d("2b") = "5b". This new weight will replace the older weight in the 'd' column.



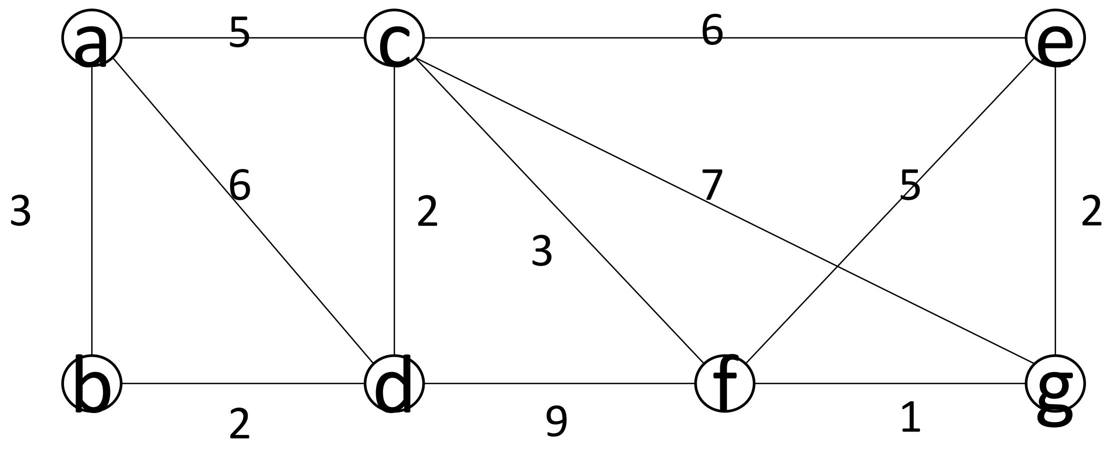
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a					

Step 3: The next adjacent vertex from b is looked for and compared with the weight when directly reached from a – “6a”. When coming via vertex b, the total weight is a->b(“3a”)+b->d(“2b”) = “5b”. This new weight will replace the older weight in the ‘d’ column.



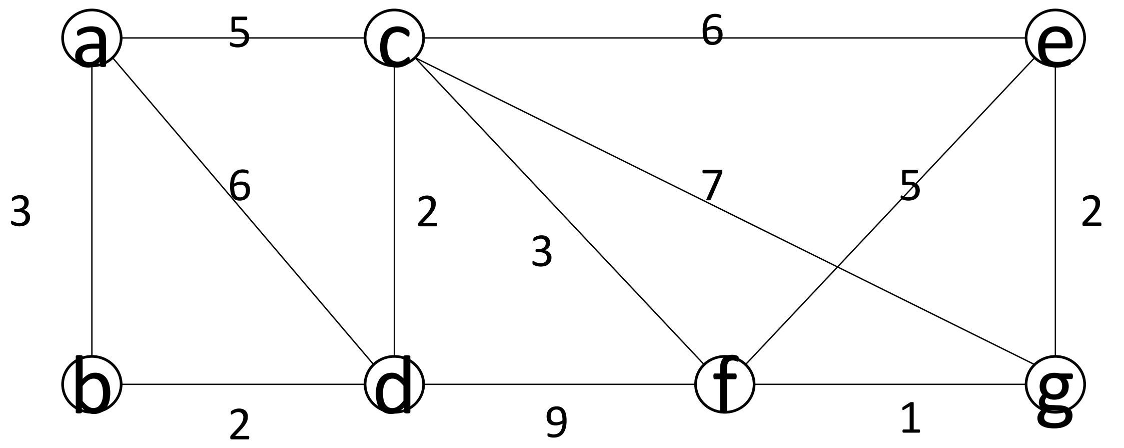
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a		5_b ⬅			

Step 3: Cannot reach anywhere else from ‘b’, so rest of the weights – ‘c’,’e’,’f’,’g’ are copied down from the first row as they were



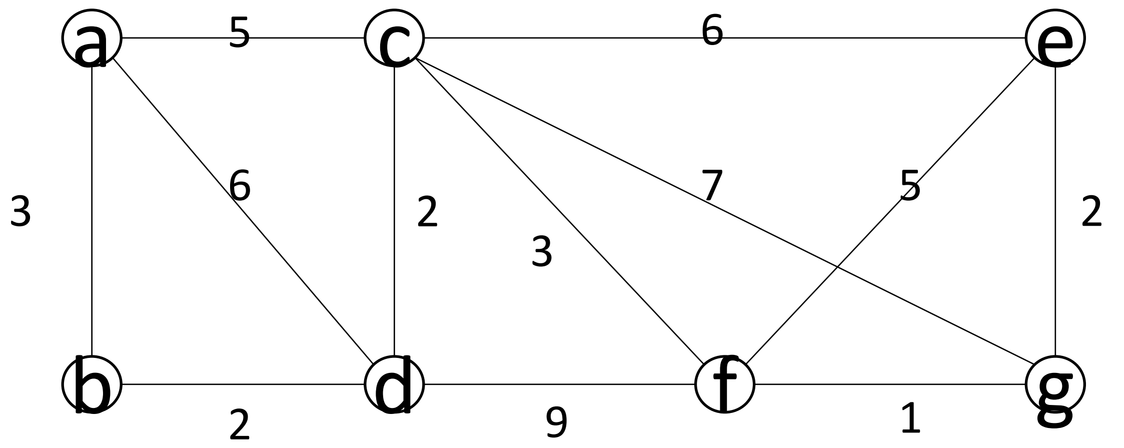
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a		5_b ⬅			

Step 3: Cannot reach anywhere else from ‘b’, so rest of the weights – ‘c’,’e’,’f’,’g’ are copied down from the first row as they were



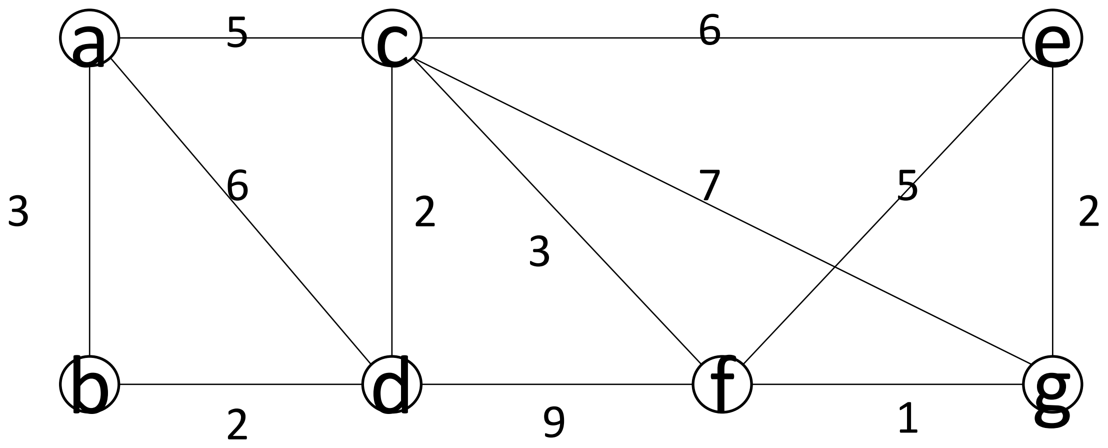
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b ⬅	∞_a	∞_a	∞_a

Step 4: Next lowest value is selected. So any of “5a” or “5b” will be valid.



V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b ⬅	∞_a	∞_a	∞_a

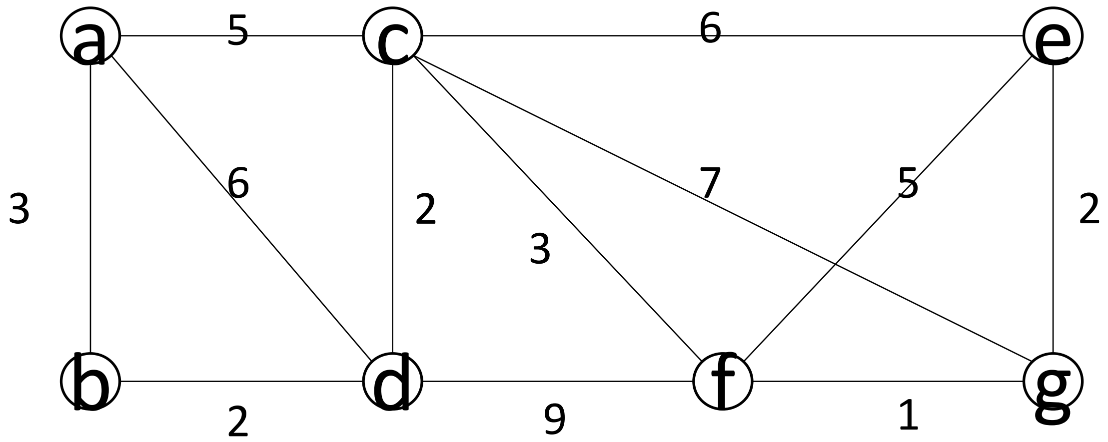
Step 4: Next lowest unmarked weight is selected. So any of "5a" or "5b" will be valid.



- c->d cost: 5a+2c = 7c which is !< 5b, so weight in column d remains 5b.
- c->e cost = 5a+6c = 11c which is < ∞, so new weight in column e becomes 11c
- Similarly for f and g, new weights are 8c and 12c respectively.
- Remember: Subscript is where I'm coming from!

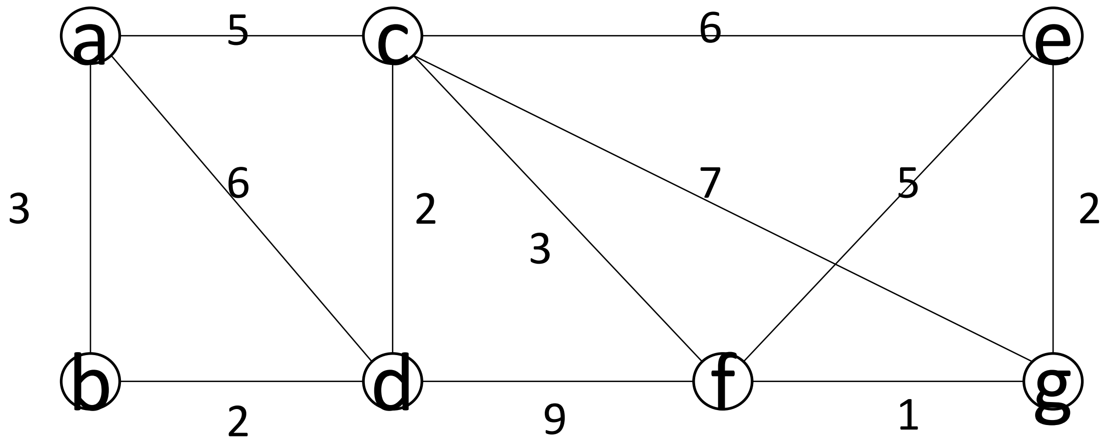
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c

Step 5: Next lowest "5b". d->f = 5b+9d = 14d !< 11c. 'f' and 'g' are unreachable so their weights remain as they are.



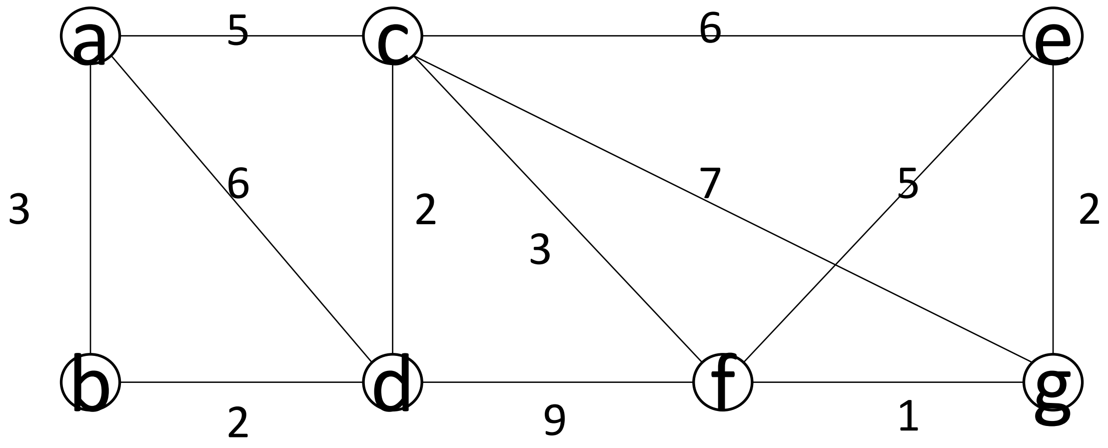
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c

Step 6: Next lowest "8c". f->e = 8c+5f = 13f !< 11c.
f->g = 8c+1f = 9f <12c



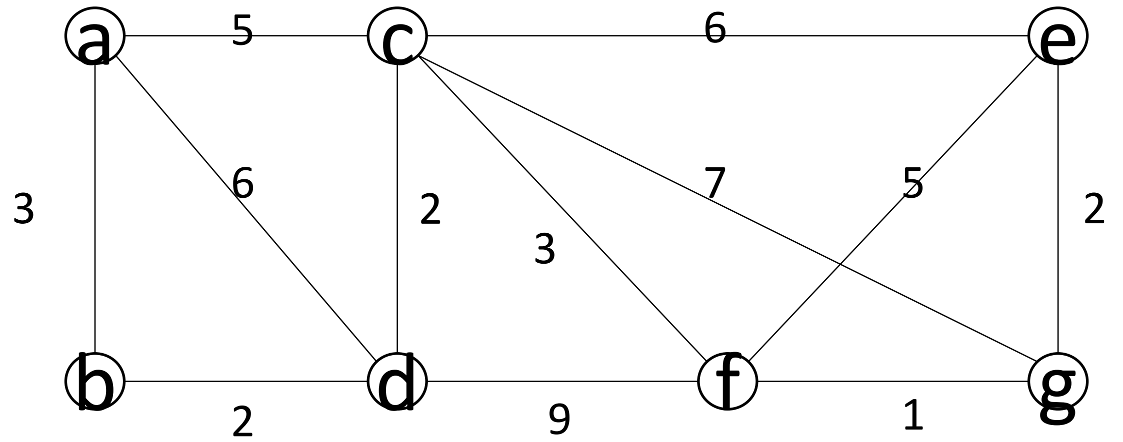
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f

Step 7: No more scope for improvement so the last row gives me the shortest path from 'a' to any of the other vertices.



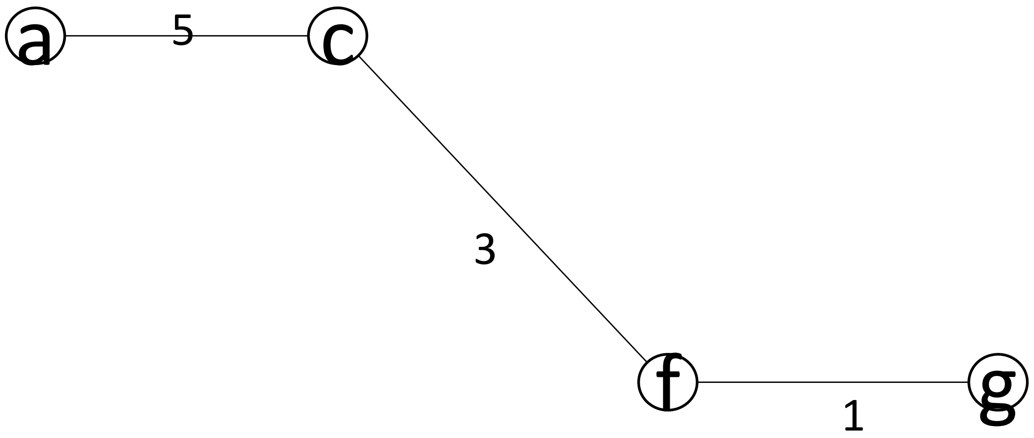
V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f

Shortest Path weight from ‘a’ to ‘g’ is = 9f, where f=8c, where c = 5a.



V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f

So, the shortest path from ‘a’ to ‘g’ would be:



V	a	b	c	d	e	f	g
a	0_a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b	0_a	3_a	5_a	5_b	∞_a	∞_a	∞_a
c	0_a	3_a	5_a	5_b	11_c	8_c	12_c
d	0_a	3_a	5_a	5_b	11_c	8_c	12_c
f	0_a	3_a	5_a	5_b	11_c	8_c	9_f

Djikstra Analysis:

- $O(n\log n)$ time to build priority queue
- $O(n\log n)$ time removing vertices from queue
- $O(m\log n)$ time relaxing edges
  - Changing key can be done in $O(\log n)$ time
- Total time: $O((n + m)\log n)$
  - which can be $O(n^2\log n)$ for dense graph