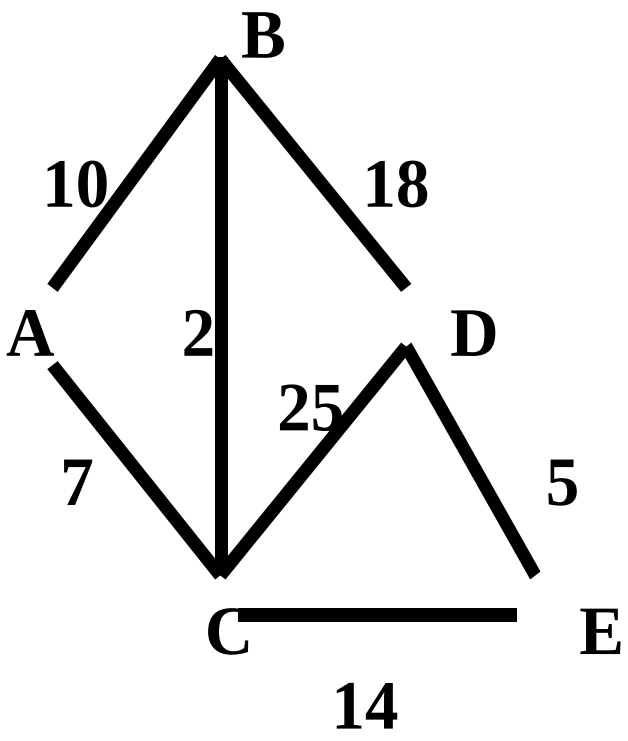
Dijkstra's algorithm:

Let pq be a priority queue of vertex-weight pairs, <w, wweight>, where wweight is the total weight of all edges on the shortest path so far from v1 to w.

Iterate, breadth-first, starting at v1 until a pair with v2 is removed from pq.

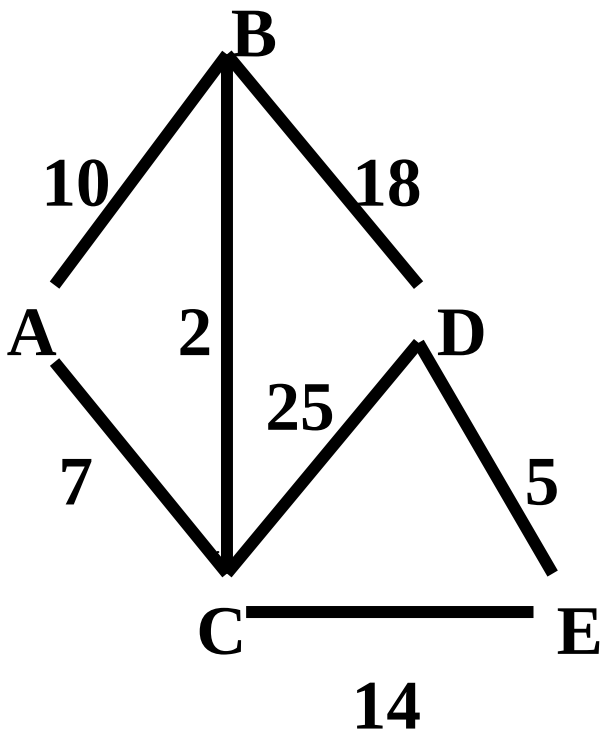
To keep track of total path weights, we have a Map object, weightSum, that associates with each vertex x the sum of the weights on the shortest path so far from v1 to x.

Finally, predecessor, another Map object, associates each vertex x with its predecessor on the shortest path so far from v1 to x. That map enables us to retrieve the shortest path when we are done.



Find the shortest path from A to D.

Initialize weightSum and predecessor, and add <A, 0> to pq**.**

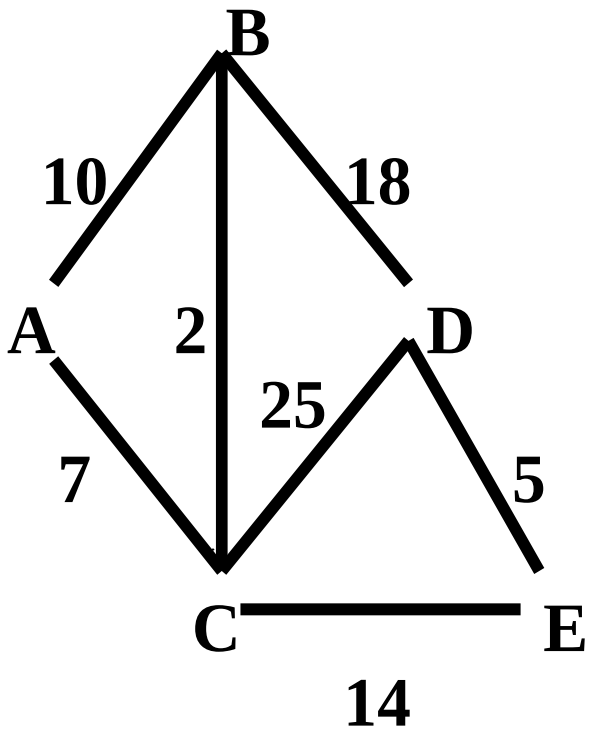


weightSum	predecessor	pq
A, 0	A	<A, 0>
B, 1000	null	
C, 1000	null	
D, 1000	null	
E, 1000	null	

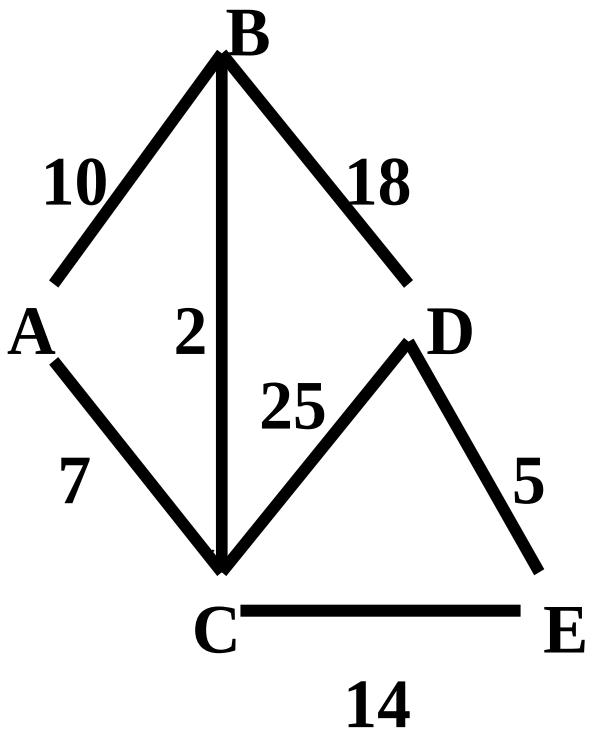
Now loop until D is removed from pq. For each pair <w, weight sum> removed from pq, iterate over the neighbors of w (provided that weight sum is <= w's weight sum).

For each neighbor x, if w's weight sum + weight of (w, x) is less than x's weight sum, then we have found a cheaper way to get from A to x!

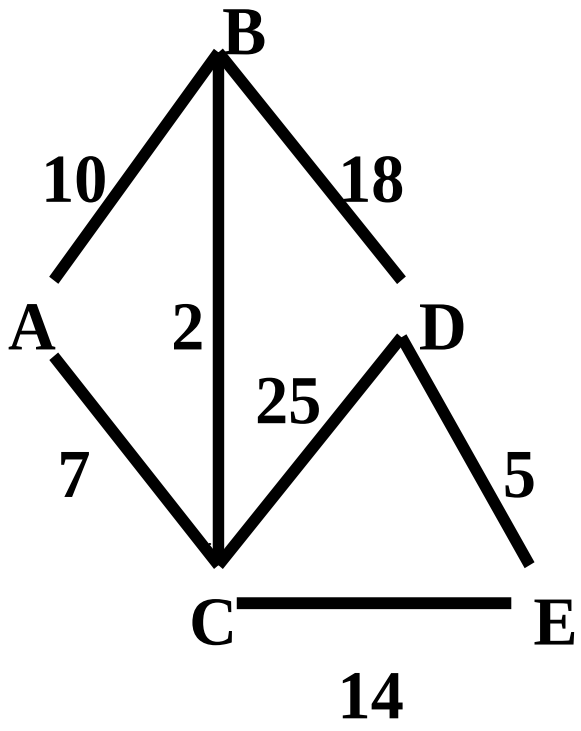
So replace x's weight sum with w's weight sum + weight of (w, x) and insert x and its new weight sum into pq. Also, make w the predecessor of x.



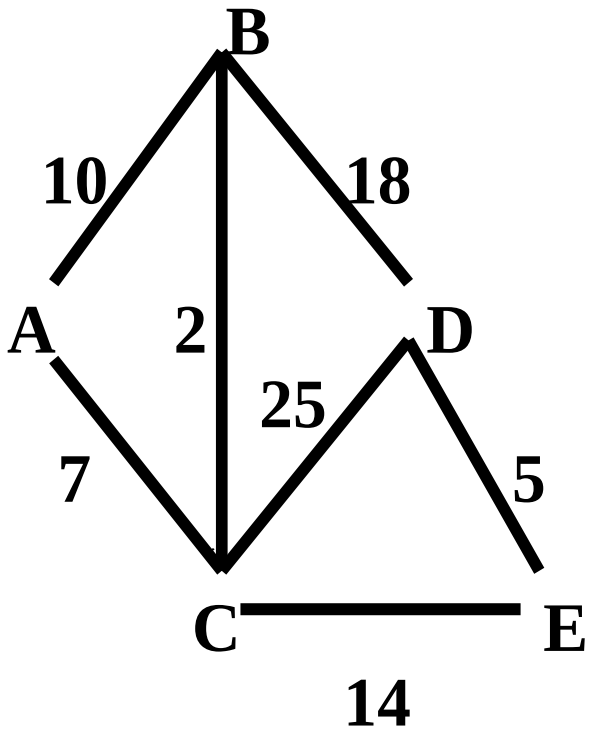
weightSum	predecessor	pq
A, 0	A	<C, 7>
B, 10	A	<B, 10>
C, 7	A	
D, 1000	null	
E, 1000	null	



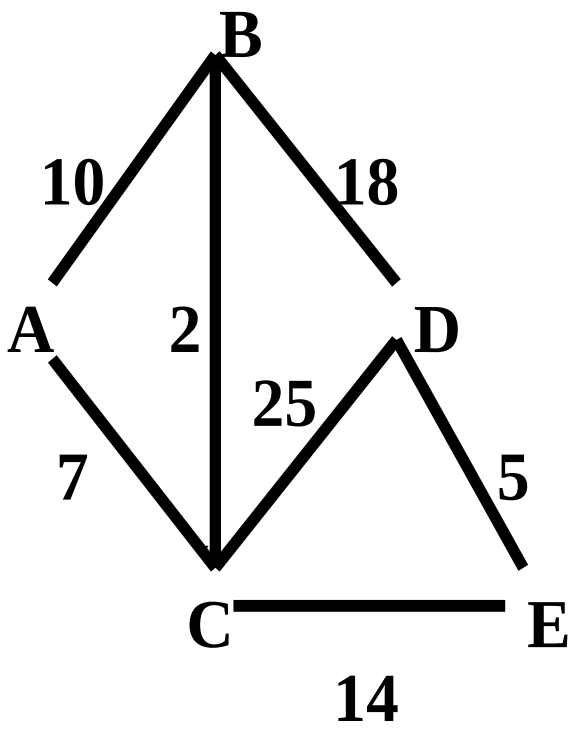
weightSum	predecessor	pq
A, 0	A	<B, 9>
B, 9	C	<B, 10>
C, 7	A	<E, 21>
D, 32	C	<D, 32>
E, 21	C	



weightSum	predecessor	pq
A, 0	A	
B, 9	C	<E, 21>
C, 7	A	<D, 27>
D, 27	B	<D, 32>
E, 21	C	



weightSum	predecessor	pq
A, 0	A	<E, 21>
B, 9	C	<D, 27>
C, 7	A	<D, 32>
D, 27	B	
E, 21	C	



weightSum	predecessor	pq
A, 0	A	<D, 26>
B, 9	C	<D, 27>
C, 7	A	<D, 32>
D, 26	E	
E, 21	C	

During the next loop iteration, when <D, 26> is removed from pq, we stop. The shortest path from A to D is, according to predecessor, A, C, E, D.