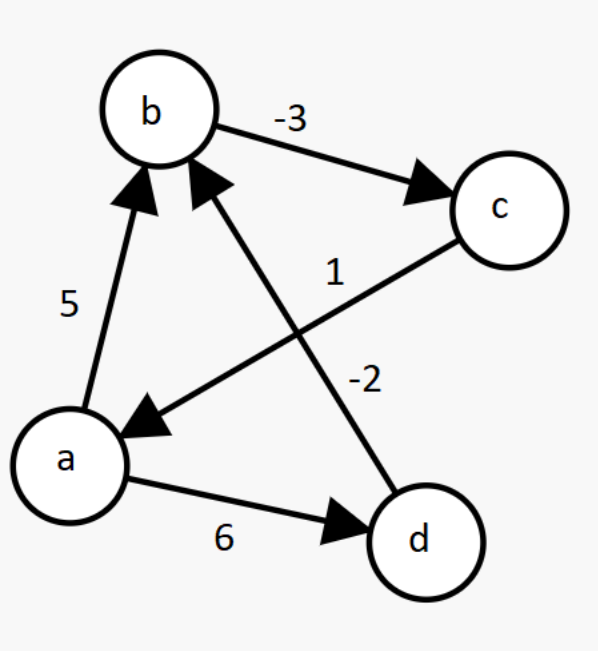
Distance Vector Routing (Bellman Ford) Algorithm Example

Example: Execute the Bellman Ford algorithm on the below network to provide the list of distances and previous nodes from the source. Assume node a is the source node.



In addition, so that I can see your work, when replacing a column's value, cross out the previous iteration's value, do not delete it.

Solution:

This is the Bellman Ford algorithm:

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0

RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

w represents the edge weights (aka distances between nodes) and s represents the source node.

π (pi) represents the previous vertex.

As you can see from the above algorithm, the number of times the doubly nested for loop (lines 2 and 3) executes is the number of vertices (aka nodes) minus 1, times the number of edges. In the above graph, there are 4 nodes and 5 edges, so there will be (4-1)*5 = 15 edge examinations done by the algorithm. In other words, all 5 edges will each be examined 3 times (5*3 = 15).

Note that unlike Dijkstra's algorithm, the Bellman Ford algorithm can handle negative edge weights (e.g. a node puts negative information on a route to ensure that route isn't used, textbook page 248).

With Dijkstra's algorithm, the order of vertices you analyze is dependent on the distance from the source (with smallest distance taking precedence). However, with the Bellman Ford algorithm, it doesn't matter the order of vertices that you analyze, because they're all going to be analyzed anyway in literally every iteration of the doubly nested for loop.

This below table is the final answer, work to get the values is shown below it.

Node	Distance from a	Previous Node
a	0 ~~Infinity~~	NIL
b	4 ~~5~~ ~~Infinity~~	d ~~a~~ ~~NIL~~
c	1 ~~2~~ ~~Infinity~~	b ~~b~~ ~~NIL~~
d	6 ~~Infinity~~	a ~~NIL~~

Note that the final answer of distance and previous node values is unique, but the order of edges being examined is not unique, so your work may look different than this document.

15 iterations total:
Line 2 for loop iterations:

Iteration 1:
a->b:
Is infinity > 0 + 5? Yes, so update b's distance entry to 0 + 5 = 5. Update previous node to a.

b->c:
Is infinity > 5 + (-3)? Yes, so update c's distance to 5 + (-3) = 2. Update previous node to b.

c->a:
Is 0 > 2 + 1? No, so don't update row a.

a->d:
Is infinity > 0 + 6? Yes, so update d's distance to 0 + 6 = 6. Update previous node to a.

d->b:
Is 5 > 6 + (-2)? Yes, so update b's distance to 6 + (-2) = 4. Update previous node to d.

Iteration 2:
a->b:
Is 4 > 0 + 5? No, so don't update row b.

b->c:
Is 2 > 4 + (-3)? Yes, so update c's distance to 4 + (-3) = 1. Update previous node to b (it's already b, but still gets updated anyway).

c->a:
Is 0 > 1 + 1? No, so don't update row a.

a->d:
Is 6 > 0 + 6? No, so don't update row d.

d->b:
Is 4 > 6 + (-2)? No, so don't update row b.

Iteration 3:
a->b:
Is 4 > 0 + 5? No, so don't update row b.

b->c:
Is 1 > 4 + (-3)? No, so don't update row c.

c->a:
Is 0 > 1 + 1? No, so don't update row a.

a->d:
Is 6 > 0 + 6? No, so don't update row d.

d->b:
Is 4 > 6 + (-2)? No, so don’t update row b.