Bellman-Ford Algorithm for shortest paths

Input: Directed graph $G = (V, E)$, edge weights $w : E \mapsto \mathbb{R}$, source $s \in V$.

Output: For each $u \in V$, $u.distance = \delta(s, u)$, shortest path distance from s to u, and, $u.parent =$ predecessor of u in such a path. If G has a negative cycle, algorithm returns $false$, otherwise $true$.

Idea: Dynamic program of the following recursive algorithm.

Define $d_k(u)$ to be the length of a shortest path from s to u that uses at most k edges. When $k = 0$, $d_0(u) = \infty$, if $u \neq s$, and, $d_0(s) = 0$. Recurrence for d_k:

$$d_k(u) = \min\{d_{k-1}(u), \min_{(p,u)\in E}\{d_{k-1}(p) + w(p, u)\}\}.$$

If G does not have a negative cycle, then $d_{|V|-1}(u) = \delta(s, u)$, because a simple shortest path has at most $|V| - 1$ edges. In addition, if $d_k(u) = d_{k-1}(u)$, for all $u \in V$, then the recursion can be stopped at k. If G has a negative cycle, then $d_{|V|}(u) \neq d_{|V|-1}(u)$, for some $u \in V$, and the algorithm returns $false$.

Dynamic program to compute d_k: Take 1

```
// Store d_k(u) in array d[ ] defined in Vertex class.
// Solve problems in increasing values of k to avoid recursive calls.
for u ∈ V do
    u.d[0] ← ∞;   u.parent ← null
s.d[0] ← 0
// Invariant: u.d[k − 1] = d_{k−1}(u), for all u ∈ V.
for k ← 1 to |V| do
    nochange ← true
    for u ∈ V do
        u.d[k] ← u.d[k − 1]
        for edge e = (p, u) ∈ E do
            if u.d[k] > p.d[k − 1] + w(e) then
                u.d[k] ← p.d[k − 1] + w(e)
                u.parent ← p
                nochange ← false
    if nochange then
        for u ∈ V do u.distance ← u.d[k]
        return true
return false // G has a negative cycle
```


Dynamic program to compute d_k: Take 2

Recurrence for d_k is guaranteed to be feasible, and therefore all elements of $u.d[\ ]$ can be overlaid on the same location, thus replacing the array by a scalar, $u.distance$. In addition, all edges are relaxed in each iteration of k. Edges of the graphs can be relaxed in any order, for a given k.

```
Bellman-Ford(Graph G = (V, E), Vertex s)
for u ∈ V do
    u.distance ← ∞
    u.parent ← null
s.distance ← 0
for k ← 1 to |V| do
    nochange ← true
    for edge e = (u, v) ∈ E do
        if v.distance > u.distance + w(e) then
            v.distance ← u.distance + w(e)
            v.parent ← u
            nochange ← false
    if nochange then
        return true
return false // G has a negative cycle
```


Faster algorithm: Take 3

Process edges out of u only when $u.distance$ changes. Keep track of how many times a node has been processed in field *count*. Worst-case RT is $O(|E||V|)$, but actual RT for many graphs is significantly less than the algorithm in Take 2.

```
Create a queue q to hold vertices waiting to be processed
for u ∈ V do
    u.distance ← ∞; u.parent ← null; u.count ← 0; u.seen ← false
s.distance ← 0; s.seen ← true; q.add(s)
while q is not empty do
    u ← q.remove(); u.seen ← false // no longer in q
    u.count ← u.count + 1
    if u.count ≥ |V| then return false // Negative cycle
    for Edge e = (u, v) ∈ u.Adj do
        if v.distance > u.distance + w(e) then
            v.distance ← u.distance + w(e)
            v.parent ← u
            if not v.seen then
                q.add(v); v.seen ← true
return true
```