```
InitSSSP(s):
    dist(s) ← 0
    pred(s) ← Null
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← Null
```

Edge $u \to v$ is tense if

$$dist(u) + w(u \to v) < dist(v)$$

```
Relax(u→v):
    dist(v) ← dist(u) + w(u→v)
    pred(v) ← u
```

```
FordSSSP(s):
    InitSSSP(s)
    while there is at least one tense edge
        Relax any tense edge
```

```
BellmanFord(s)
    InitSSSP(s)
    repeat V − 1 times
        for every edge u→v
            if u→v is tense
                Relax(u→v)
    for every edge u→v
        if u→v is tense
            return "Negative cycle!"
```

every edge in G |V|-1 times

Finds shortest paths if no cycle has negative weight.

O(VE) time

No Negative Weight Edges:

Dijkstra's

Observations

1) $u \to v$ can only become tense if $dist(u)$ decreases

2) If you relax $u \to v$, you'll $dist(v) \geq dist(u)$ (if $w(u \to v) \geq 0$)

Keep a priority queue of tail vertices with key $= dist(u)$

```
DIJKSTRA(s):
    INITSSSP(s)
    INSERT(s, 0)
    while the priority queue is not empty
        u ← EXTRACTMIN()
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                if v is in the priority queue
                    DECREASEKEY(v, dist(v))
                else
                    INSERT(v, dist(v))
```

This is FordSSSP with Observation 1, so it's correct!

Analysis (assuming no neg. weights)

u_i : vertex returned by ith call to ExtractMin (so $u_1 = s$)

$d_i := \text{dist}(u_i)$ at the moment we do the ith ExtractMin (so $d_1 = 0$)

For all we know so far, $u_i = u_j$ for some $i < j$.

Lemma: For all $i < j$, we have $d_j \geq d_i$.

Proof:

Fix i. We'll show $d_{i+1} \geq d_i$.

Suppose we relax $u_i \to u_{i+1}$ during ith round.

Immediately after

$$\mathrm{dist}(u_{i+1}) = \mathrm{dist}(u_i) + w(u_i \to u_{i+1})$$
$$\geq \mathrm{dist}(u_i).$$

Otherwise u_{i+1} was already in queue. But we didn't Extract it, so $\text{dist}(u_i) \le \text{dist}(u_{i+1})$.

Lemma: Each vertex is extracted at most once.

Proof: Suppose $v = u_i = u_j$ for some $j > i$.

We pulled it out, but put it back, so $d_j < d_i$.

But we just argued that never happens! $\perp$

Lemma: When Dijkstra ends, for all v, $dist(v)$ is the distance to v.

Let $s = v_0 \to v_1 \to \ldots \to v_\ell = v$ be the shortest path to v.

Let L_j be the length of $v_0 \to \ldots \to v_j$. We'll prove by induction on j that $dist(v_j) \le L_j$.

$dist(v_0) = dist(s) = 0 = L_0$.

Suppose $j > 0$. By induction, we can assume we Extract v_{j-1} & $dist(v_{j-1}) \leq L_{j-1}$ at that time.

At that moment, either

$$dist(v_j) \leq dist(v_{j-1}) + w(v_{j-1} \to v_j)$$

or we set $dist(v_j) =$

$$dist(v_{j-1}) + w(v_{j-1} \to v_j)$$

when we look at $v_{j-1} \to v_j$.

Either way, $dist(v_j) \leq dist(v_{j-1}) + w(v_{j-1} \to v_j)$

$$\leq L_{j-1} + w(v_{j-1} \to v_j)$$

$$= L_j$$

In particular $dist(v) =$

$dist(v_\ell) =$

$L_\ell =$

distance to v

So, we do at most one Insert & ExtractMin per vertex

at most one Decreasekey per edge

With a binary heap, $O(\log V)$ per op.

$O(E \log V)$ time total

Still correct with negative weight edges.

Still fast with very few negative weight edges...

But with many negative edges it may take exponential time.

$O(V)$ Insert & ExtractMins

$O(E)$ DecreaseKeys

Binary Heap: $O(\log V)$ time per op

Fibonacci Heap:

$O(1)$ time (on average)

Insert & DecreaseKey

$O(\log V)$ time (on average)

ExtractMin

Dikstra with Fib. Heaps: $O(E + V\log V)$ total

```
JARNÍK(V, E, s):
    JARNÍKINIT(V, E, s)
    JARNÍKLOOP(V, E, s)
```

```
JARNÍKINIT(V, E, s):
    for each vertex v ∈ V \ {s}
        if vs ∈ E
            edge(v) ← vs
            priority(v) ← w(vs)
        else
            edge(v) ← NULL
            priority(v) ← ∞
        INSERT(v)
```

```
JARNÍKLOOP(V, E, s):
    T ← ({s}, ∅)
    for i ← 1 to |V| − 1
        v ← EXTRACTMIN
        add v and edge(v) to T
        for each neighbor u of v
            if u ∉ T and priority(u) > w(uv)
                edge(u) ← uv
                DECREASEKEY(u, w(uv))
```

$O(E \log V)$ with binary heaps

$O(E + V \log V)$ with Fib heaps

(Don't do this in practice. Stick to binary heaps.)

Edge Weights = 1 (want to minimize # edges on a path)

Use breadth-first search.

<u>BFS(s):</u>
    InitSSSP(s)
    Push(s)
    while the queue is not empty
        $u \leftarrow$ Pull()
        for all edges $u \to v$
            if $dist(v) > dist(u) + 1$ ⟨⟨*if $u \to v$ is tense*⟩⟩
                $dist(v) \leftarrow dist(u) + 1$ ⟨⟨*relax $u \to v$*⟩⟩
                $pred(v) \leftarrow u$
                Push(v)

$O(V+E)$ time

Directed-Acyclic Graphs

$$dist(v) = \begin{cases} 0 & \text{if } v = s \\ \min\limits_{u \to v} \left(dist(u) + w(u \to v) \right) & \text{otherwise} \end{cases}$$

No (negative) cycles.

```
DagSSSP(s):
  for all vertices v in topological order
      if v = s
          dist(v) ← 0
      else
          dist(v) ← ∞
          for all edges u→v
              if dist(v) > dist(u) + w(u→v)        ⟨⟨if u→v is tense⟩⟩
                  dist(v) ← dist(u) + w(u→v)       ⟨⟨relax u→v⟩⟩
```

```
PushDagSSSP(s):
  InitSSSP(s)
  for all vertices u in topological order
      for all outgoing edges u→v
          if u→v is tense
              Relax(u→v)
```

$O(V+E)$ time

All-Pairs Shortest Paths

Want to compute $dist(u,v)$ for all $u,v \in V$: distance from u to v.

$\Theta(V^2)$ values to compute

```
ObviousAPSP(V, E, w):
  for every vertex s
      dist[s, ·] ← SSSP(V, E, w, s)
```

Unweighted or DAG:

$V \cdot O(E) = O(VE) = O(V^3)$ time

Non-Negative Weights:

$V \cdot O(E + V\log V) = O(VE + V^2 \log V)$

$= O(V^3)$

($O(V^3 \log V)$ with binary heaps)

Otherwise:

$V \cdot O(VE) = O(V^2 E) = O(V^4)$

Can we get $O(V^3)$ even with negative length edges?