dist(v): upper bound on distance from s to v

- length of some $s \rightsquigarrow v$ walk

pred(v): last edge on that walk

<u>InitSSSP(s):</u>

 $dist(s) \leftarrow 0$
 $pred(s) \leftarrow$ Null
 for all vertices $v \neq s$
  $dist(v) \leftarrow \infty$
  $pred(v) \leftarrow$ Null

edge $u \to v$ is <u>tense</u>

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

<u>Relax($u \to v$):</u>

 $dist(v) \leftarrow dist(u) + w(u \to v)$
 $pred(v) \leftarrow u$

<u>FordSSSP(s):</u>

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

Dijkstra:

Two observations

1) $u \to v$ becomes tense only after setting dist(u).

2) If edge weights ≥ 0 dist(v) is never set lower than dist(u) during Relax($u \to v$)

so if dist(u) is lowest of all tails of tense edges, dist(u) will never lower again priority queue ops

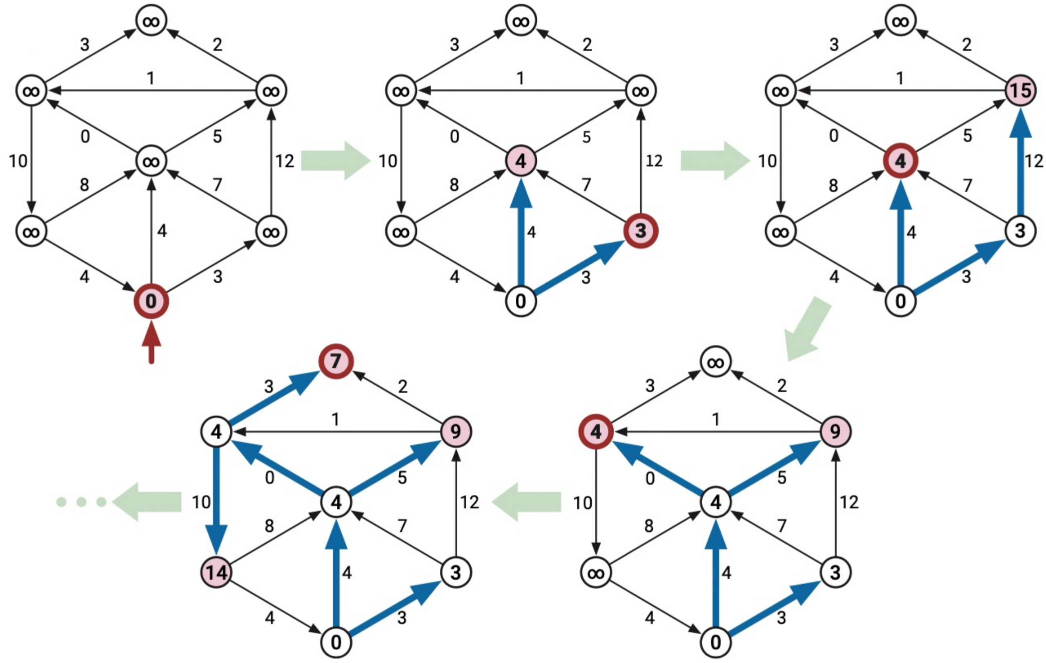
```
DIJKSTRA(s):
    INITSSSP(s)
    INSERT(s, 0)
    while the priority queue is not empty
        u ← EXTRACTMIN()
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                if v is in the priority queue
                    DECREASEKEY(v, dist(v))
            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

priority queue: holds pairs (element, key)

ExtractMin returns element with smallest key

e.x. binary heap

Will find shortest paths as long as no neg. weight cycles.



Analysis (assuming no negative weights)

u_i: ith vertex returned by ExtractMin ($u_1 = s$).

$d_i := \text{dist}(u_i)$ at moment of ith ExtractMin ($d_1 = 0$)

(For all we know right now, $u_i = u_j$ for some $i < j$)

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

Fix some i. Will show $d_{i+1} \geq d_i$.

If $u_i \to u_{i+1}$ is relaxed after ith ExtractMin,

$$
\begin{aligned}
d_{i+1} &= dist(u_{i+1}) \\
&= dist(u_i) + w(u_i \to u_{i+1}) \\
&\geq dist(u_i) \\
&= d_i
\end{aligned}
$$

o.w. u_{i+1} was already in p. queue

We extracted u_i, so

$$d_{i+1} = \mathrm{dist}(u_{i+1})$$
$$\geq \mathrm{dist}(u_i)$$
$$= d_i$$

Lemma: Each vertex is extracted at most once.

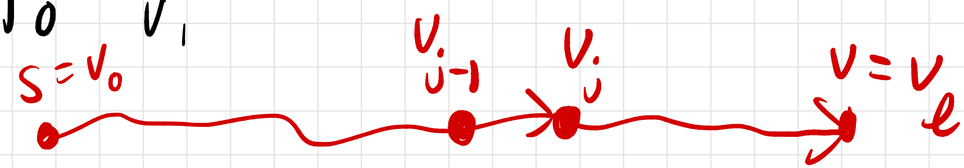
Suppose $v = u_i = u_j$ for some $j > i$.

To put v back in queue after first time, a relaxation decreased $\mathrm{dist}(v)$.

So $d_j < d_i$ $\perp$

Lemma: When Dijkstra ends, $\forall v \in V$, $dist(v)$ is distance from s to v.

Proof: Let $s = v_0 \to v_1 \to \dots \to v_\ell = v$ be the shortest path from s to v.



L_j: length of $v_0 \to v_1 \to \dots \to v_j$

Will prove by induction $dist(v_j) \le L_j$.

$dist(v_0) = dist(s) = 0 = L_0$ ✓

Consider $j > 0$.

By induction, we extracted v_{j-1}

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

or we set $dist(v_j) \leftarrow dist(v_{j-1}) + w(v_{j-1} \to v_j)$

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

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

$= L_j$

In particular, $dist(v) \le L_\ell$, the distance from s to v.

$dist(v) \geq distance$ also

$\Rightarrow dist(v) = distance$ ✓

Binary heap as the priority queue for $O(\log V)$ time per operation.

Non-neg. weights $\Rightarrow$ $|V|$ Inserts &

$|V|$ ExtractMins

$|E|$ DecreaseKeys

So, $O(E \log V)$ time.

If all edges have weight 1 (min # edges on path)

```
BFS(s):
    InitSSSP(s)
    Push(s)
    while the queue is not empty
        u ← Pull()
        for all edges u→v
            if dist(v) > dist(u) + 1          ⟨⟨if u→v is tense⟩⟩
                dist(v) ← dist(u) + 1         ⟨⟨relax u→v⟩⟩
                pred(v) ← u
                Push(v)
```

Runs in $O(V+E)$ time.

Directed Acyclic Graphs (dynamic programming)

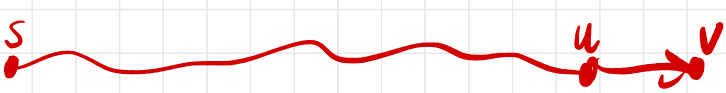
Easy even with arbitrary edge weights.

no cycles $\Rightarrow$ no negative cycles!

dist(v) := <u>actual</u> distance from s to v.

dist(s) = 0

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



```
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⟩⟩
```

- same algorithm -

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

$O(V+E)$ time. (any edge weights)

```
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)
```

- weights are 1: BFS $O(V+E)$
- no cycles: DAG SSSP $O(V+E)$
- non-negative weights: Dijkstra $O(E \log V)$
- o.w.: Bellman-Ford $O(VE)$
(can also detect if $\exists$ a negative cycle)

undirected graphs: Hope you don't have negative weights (min weight T-join)