dist(\(v\)): upper bound on distance from \(s\) to \(v\)
- length of some \(s \rightarrow v\) walk
\(\text{pred}(u)\): last edge on that walk

\begin{align*}
\text{InitSSSP}(s): \\
dist(s) &\leftarrow 0 \\
\text{pred}(s) &\leftarrow \text{NULL} \\
\text{for all vertices } v &\neq s \\
dist(v) &\leftarrow \infty \\
\text{pred}(v) &\leftarrow \text{NULL}
\end{align*}

Edge \(u \rightarrow v\) is tense
if \(\text{dist}(u) + w(u \rightarrow v)\)
\(< \text{dist}(v)\)

\begin{align*}
\text{Relax}(u \rightarrow v): \\
\text{dist}(v) &\leftarrow \text{dist}(u) + w(u \rightarrow v) \\
\text{pred}(v) &\leftarrow u
\end{align*}

\begin{align*}
\text{FordSSSP}(s): \\
\text{InitSSSP}(s) \\
\text{while there is at least one tense edge} \\
\text{Relax any tense edge}
\end{align*}
Dijkstra:

Two observations

1) \( u \rightarrow v \) becomes tense only after setting \( \text{dist}(u) \).

2) If edge weights \( \leq 0 \)

\( \text{dist}(v) \) is never set lower than \( \text{dist}(u) \) during \( \text{Relax}(u \rightarrow v) \)

so if \( \text{dist}(u) \) is lowest of all tails of tense edges, \( \text{dist}(u) \) will never lower again
`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

ex. binary heap

Will find shortest paths as long as no neg weight cycles.
Analysis (assuming no negative weights)

$u_i$: $i$th vertex returned by $\text{Extract Min} (u_i = s)$.

d_i := \text{dist}(u_i) \text{ at moment of } i\text{th } \text{Extract Min} (d_i = 0)$

(For all we know right now, $u_i = u_j$ for some $i \neq j$.)
Lemma: For all \( \hat{u} \neq j \), we have
\[
d_j = d_{\hat{u}}.
\]

Fix some \( \hat{u} \). Will show \( d_{\hat{u}+1} \geq d_{\hat{u}} \).

If \( u_{\hat{u}} \rightarrow u_{\hat{u}+1} \) is relaxed after \( i \)th Extract Min,
\[
d_{\hat{u}+1} = \text{dist} (u_{\hat{u}+1}) \]
\[
= \text{dist} (u_{\hat{u}}) + w (u_{\hat{u}} \rightarrow u_{\hat{u}+1}) \]
\[
\geq \text{dist} (u_{\hat{u}}) \]
\[
= d_{\hat{u}}
\]

O.w. \( u_{\hat{u}+1} \) was already in p.
queue

We extracted \( u_{\hat{u}} \), so
\[
d_{\tilde{u}_{i+1}} = \text{dist}(u_{\tilde{u}_{i+1}}) \\
\geq \text{dist}(u_{\tilde{u}_i}) \\
= d_{\tilde{u}_i} 
\]

Lemma: Each vertex is extracted at most once.

Suppose \( v = u_{\tilde{u}_i} = u_{\tilde{u}_j} \) for some \( j > i \).

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

So \( d_{\tilde{u}_j} < d_{\tilde{u}_i} \).
Lemma: When Dijkstra ends, \( \forall v \in V, \text{dist}(v) \) is distance from \( s \) to \( v \).

Proof: Let \( s = v_0 \Rightarrow v_1 \Rightarrow \ldots \Rightarrow v_l = v \) be the shortest path from \( s \) to \( v \).

\[ S = v_0 \quad v_{i-1} \quad v_i \quad v = v \]

\( L_j \): length of \( v_0 \Rightarrow v_1 \Rightarrow \ldots \Rightarrow v_j \)

Will prove by induction \( \text{dist}(v_j) \leq L_j \).
\[ \text{dist}(v_j) = \text{dist}(s) = 0 = L_0 \checkmark \]

Consider \( j > 0 \).

By induction, we extracted \( v_{j-1} \), either \( \text{dist}(v_j) \leq \text{dist}(v_{j-1}) + w(v_{j-1} \rightarrow v_j) \) or we set \( \text{dist}(v_j) < \text{dist}(v_{j-1}) + w(v_{j-1} \rightarrow v_j) \).

So, \[ \text{dist}(v_j) \leq \text{dist}(v_{j-1}) + w(v_{j-1} \rightarrow v_j) \leq L_{j-1} + w(v_{j-1} \rightarrow v_j) = L_j \]

In particular, \( \text{dist}(v) \leq L_j \), the distance from \( s \) to \( v \).
\[ \text{dist}(v) = \text{distance also} \]
\[ \Rightarrow \text{dist}(v) = \text{distance} \checkmark \]

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

Non-neg. weights \( \Rightarrow \) IV) Inserts & IV) Extract Mins

1E1 Decrease Keys

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  \( \langle \text{if } u \rightarrow v \text{ is tense} \rangle \)
              dist(v) ← dist(u) + 1  \( \langle \text{relax } u \rightarrow v \rangle \)
              pred(v) ← u
              Push(v)

Runs in \(O(V+E)\) time.
Directed Acyclic Graphs (dynamic programming)

Easy even with arbitrary edge weights.

no cycles => no negative cycles!

dist(v) := actual distance from s to v.

dist(s) = 0

dist(v) = \begin{cases} 
0 & \text{if } v = s \\
\min_{u \rightarrow v} (\text{dist}(u) + w(u \rightarrow v)) & \text{otherwise}
\end{cases}
**DAGSSSP(s):**
for all vertices $v$ in topological order
  
  if $v = s$
    
    $dist(v) \leftarrow 0$
    
  else
    
    $dist(v) \leftarrow \infty$
    
  for all edges $u \rightarrow v$
    
    if $dist(v) > dist(u) + w(u \rightarrow v)$      \(\langle\text{if } u \rightarrow v \text{ is tense}\rangle\)
      
      $dist(v) \leftarrow dist(u) + w(u \rightarrow v)$    \(\langle\text{relax } u \rightarrow v\rangle\)

---

**Same algorithm**

---

**DAGSSSP(s):**

**INITSSSP(s)**

for all vertices $v$ in topological order

for all edges $u \rightarrow v$

if $u \rightarrow v$ is tense

RELAX($u \rightarrow v$)

---

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

---

**PUSHDAGSSSP(s):**

**INITSSSP(s)**

for all vertices $u$ in topological order

for all outgoing edges $u \rightarrow v$

if $u \rightarrow v$ is tense

RELAX($u \rightarrow v$)
- Weights are 1: BFS \( O(V + E) \)
- No cycles: DAG \( O(V + E) \)
- 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)