Continue SSSP:

Vertex attributes:

P[\(v\)] : Parent, encodes shortest path tree (like BFS).
\(d[\(v\)]\) : Estimate at \(8(5, x)\)

Predecessor subgraph: \(G_p = (V_p, E_p)\):

\(V_p = \{ x \in V \mid P[\(x\)] \neq \text{null} \text{ or } x = s \}\)

\(E_p = \{ (P[\(x\)], x) \mid P[\(x\)] \neq \text{null} \}\)

To print shortest paths: use
\(\text{printPath}(G, s, x)\) on p. 101.
helper functions:

\textbf{Initialize} \((C, S)\)

1.) for all \(x \in V\)
2.) \(d[x] = \infty\)
3.) \(p[x] = \text{nil}\)
4.) \(d[\text{nil}] = 0\)

\textbf{Relax} \((x, y)\) \hspace{0.5cm} \text{pre: } y \in \text{adj} \{x\}

1.) if \(d[y] > d[x] + w(x, y)\)
2.) \(d[y] = d[x] + w(x, y)\)
3.) \(p[y] = x\)
Note:

- \( \text{Relax}(x,y) \) changes at most the attribute of \( y \).
- After \( \text{Relax}(x,y) \) then \( d[y] \leq d[x] + w[x,y] \) must be true.

Both algorithms begin by calling \( \text{Initialize}(G,e) \), then execute a seq. of calls to \( \text{Relax}(i,j) \).
Lemma 1

Let \( x \in V(G) \) and suppose that after Initialize \((G, s)\), some sequence of calls to Relax (. , .) results in \( d_l[x] \) being finite. Then \( G \) must contain an \( s-x \) path of weight \( d_l[x] \).

hw7 #4: Prove this.

Hint: Use strong induction on the length of the relaxation seq.
Let \( n = \# \) calls to Relax (. , .) start at \( n = 0 \).
Lemma 2:
After Initialize \((G, s)\), the inequality

\[
8/s(s, x) \leq d[x] \quad (\forall x \in V(G))
\]
is maintained over any sequence of calls to Relax(\(\cdot\), \(\cdot\)).

Proof: (contradiction)
If \(d[x] < 8(s, x)\) were to become true after some relaxation seq.,
then \(d[x]\) would be finite. So by Lemma 1, \(G\) would contain
an \(s-x\) path of weight \(d[x]\).
This contradicts the very definition of \(8(s, x)\). \(\therefore\) no such relaxation seq. exists. \(\Box\)
Lemma 3 (Path relaxation property)

If

\[ P : s = x_0, x_1, x_2, \ldots, x_k \]

is a min-weight \( S - x_k \) Path, and if the edges of \( P \) are relaxed in order

\[ (x_0, x_1), (x_1, x_2), (x_2, x_3), \ldots, (x_{k-1}, x_k) \]

then \( d(x_k) = \ell(S, x_k) \). This is true regardless of any other relaxations that may occur, even if interleaved with the above sequence.


Induction on \( k = \# \text{edges in a shortest path} \).
BellmanFord \((G, s)\)

1. Initialize \((G, s)\)
2. for \(i = 1\) to \(|V| - 1\)
3. for each \((x, y) \in E\)
4. Relax \((x, y)\)
5. for each \((x, y) \in E\)
6. if \(d[y] > d[x] + w(x, y)\)
7. return false
8. return true

**Runtime:** \(n = |V|, m = |E|\)

- **Initialize:** \(\Theta(n)\)
- **Relax \((x, y)\)** costs \(\Theta(1)\)
  - each edge is relaxed \(n-1\) times
  - \(2\) relabels \(2\) rounds \(= 2 \times 2 = 4\)
  - \(\Theta(m(n-1)) = \Theta(mn)\)
- **Load \(s\) to \(v\)** costs \(\Theta(m)\)

**Total cost:** \(\Theta(mn)\)
Dijkstra's:

- requires non-negative edge weights
- maintains a set \( S \subseteq V \) whose shortest path weights are known:
  \[ x \in S \implies \delta(x) = \delta(S, x) \]
- maintains a min-priority queue \( Q \) containing vertices not in \( S \), i.e. \( Q = V - S \). Keys in this priority queue are \( d \)-values.
Dijkstra(G, s)

1. Initialize(G, s)
2. \( S = \emptyset \)
3. \( Q = V \) // keys are d-values
4. while \( Q \neq \emptyset \)
5. \( x = \text{ExtractMin}(Q) \)
6. \( S = S \cup \{x\} \)
7. for all \( y \in \text{adj}(x) \)
8. \( \text{Relax}(x, y) \)

observe: the call to \( \text{Relax}(\cdot, \cdot) \) contains an implicit call to \( \text{DecreaseKey}(\cdot) \)

\( \text{Relax}(x, y) \) \( \text{Pre} \: y \in \text{adj}(x) \)

1. if \( d[x] > d[x] + w(x, y) \)
2. \( \text{DecreaseKey}(y, d[x] + w(x, y)) \)
3. \( P[y] = x \)

Runtime: \( n = |V|, m = |E| \)

assumne \( P, Q \) implemented as a min-heap.
* Line 3, BuildHeap : $\Theta(n)$
* Line 5, ExtractMin : $\Theta(\log n)$
* All executions of line 5:
  $\Theta(n \log n)$
* Line 8, DecreaseKey : $\Theta(\log n)$
* All executions of line 8:
  $\Theta(m \log n)$
* Total cost : $\Theta((n+m) \log n)$. 