Goal: Find shortest paths from source s to all other nodes in G where G is:
- directed
- weighted (positive weights): \( w_{uv} \geq 0 \) for all \((u,v)\in E\)

Let's start with unweighted version.

- Steps 1, 2 remain the same.

- Step 3: can we still use a queue?

- Why did we use a queue to store nodes in unweighted case?

```
  S -> X -> Y
```

- \(u, X\) visited before \(y\) so should process (set their distances) before \(y\)’s.

- Nodes should be processed (have their distances set) in the same order they are visited.

"Queue is FIFO so allows for first-node-visited to be first-node-processed."

True for weighted?

No!
For weighted:

\[ u \rightarrow x \rightarrow y \]

\[ \text{y (and x) visited before y, but we should process y before u. Why?} \]

\[ \text{An edge from y may lead to a better distance to u.} \]

"So can't process nodes in the order they are visited. Which node should get processed next?"

* \( \Rightarrow \) Process node with minimum distance.

ex: process s, x, y, u.

"Can't use a queue anymore, so how to get node with minimum distance?"

* Store nodes in a Min-Binary-Heap ordered by dist

(\( \Rightarrow \) update code. (4 changes))
Back to unweighted code.

Step 7: if \( u_{\text{dist}} = \infty \)

For unweighted:

When we update a node's distance, will that distance ever change? No.

A node u is processed if u-dist is the final (shortest) distance to u.

⇒ In unweighted, when u-dist is updated, u is processed (update just once).

What about weighted?

When we update u-dist, v-dist can that distance ever change? Yes!

\( u_{\text{dist}} \) can be improved to 7.
When should we update a node's distance?
How to modify Step 7 (and 8)?

⇒ if \( u \text{. dist} > v \text{. dist} + w_{v,u} \)
    \[ u \text{. dist} = v \text{. dist} + w_{v,u} \]

< Modify code >

Ques: For weighted, when is a node \( u \) processed
⇒ When \( u \) deleted from VHeap, \( u \) is processed
    (will prove later). Intuitively, no unprocessed
    node can improve \( u \text{. dist} \) since \( \text{dist} \) of all
    unprocessed nodes \( > u \text{. dist} \).

This will help to optimize (make more efficient)
    the code.

Another change: Problem with step 9?

" \( u \) may be updated multiple times and Step 9
    makes us reinser \( u \) into VHeap each time"

"So update \( u \)'s distance
    Then check if \( u \) in VHeap.
    If no, insert with new distance"
One more change...

Recall: when a node is deleted from VHeap, its distance is set.

How does this help to make algorithm more efficient?

Keep variable 'processed' for each node.
- Initialize to false
- When a node deleted, set processed to true
- Update dist only if not processed
/* Finds shortest paths from s to every other vertex in G */

Dijkstra's (G, s) // G = (V, E) where w_{u,v} > 0 for all (u,v) ∈ E

1. For all v ∈ V : v.dist = ∞, v.processed = false
2. s.dist = 0, s.processed = true
3. // For all v ∈ V, v.pred = null

4. VHeap.insert(s) // VHeap is min Binary Heap
   // ordered by dist

5. while (VHeap not empty):
6.   v = VHeap.deleteMin()
7.   v.processed = true
8.   // For each neighbor u of v:
9.   if (!u.processed and u.dist > v.dist + w_{v,u})
10.    u.dist = v.dist + w_{v,u} // VHeap updated
11.    u.pred = v
12.   if (VHeap does not contain u):
13.     VHeap.insert(u)

Note: In Step 6, VHeap is updated with
      VHeap.updateKey(u, v.dist + w_{v,u})
Code gives distances only. How to find paths?

\begin{align*}
\text{from } x & \quad \text{from } y \quad \text{from } z \\
& 110 \quad 20 \quad 55 \\
\end{align*}

Keep variable \text{pred} updated. Update \text{u.pred} when \text{u.dist} is updated.

\text{Add to code:}

\begin{itemize}
\item \text{ex'}
\end{itemize}

\begin{align*}
v & = s \ b \ d \ a \ c \\
\text{pred} & = \emptyset \ s \ b \ s \ d \\
\end{align*}

How to get shortest path from \( s \) to \( x \)?

Start from \( x \)

Check \text{pred} until \text{pred} = \text{null}.
void getPath(x) //returns shortest s-x path
    P = x    //path
    while (x, pred != null)
        P = P * x, pred
        x = x, pred

    return (reverse (P))

ex: get Path (c)

x = c, d, b, s
P = c * d * b * s
v = \{ a, c, b, e \} 
\text{pred = null} 
\text{shortest s-e path?} 
\text{shortest s-d path?} 
\text{runtime:} \quad \text{in terms of} \ n, \ e 
\Rightarrow \text{recall: min binary heap to store vertices.} 
\begin{align*} 
\text{steps 1-3: } & O(n) \\
\text{step 4: } & O(\log n) \\
\text{5: } & O(n) \times \text{times} \\
\text{6: } & O(\log n) \\
\text{7: } & O(1) \\
\text{8: } & O(e) \times \text{times} \\
\text{9: } & O(1) \\
10. & O(\log n) \quad \text{(decrease-key()) considered exactly once.} \\
11. & O(1) \\
12-13. & O(1) \\
\end{align*} 
\text{total:} 
\begin{align*} 
& O(n) + \\
& O(n) + \\
& O(1) + \\
& O(e) + \\
& O(1) + \\
& O(1) \\
& \text{not } O(n! \cdot e!)} \\
& \text{since each edge} \\
& \text{considered exactly once.}