Specifically:

Find shortest paths from source $s$ to all other vertices in $G$, where $G$ is:

- directed
- weighted (positive weights): $w_{u,v} > 0$ for all $(u,v) \in E$.

Differences between unweighted and weighted?

1. **unweighted**

   ![Diagram](image)

   Once we update a distance, what is possible to have true about that node?

   $d_{15}$

   $d_{18}$

   $d_{12}$

   $\Rightarrow$ We will never re-update $u$'s dist is set to 5

   $v$'s dist is set to 6

   $x$'s dist is set to 7

   $\Rightarrow$ So we should update $u$!

   (Traversing an edge in the future cannot yield shorter path to $v, u, x$)
Difference #1

When to update \( u \text{ dist} \)?

For vertex \( x \), with neighbor \( u \):

\[
\begin{align*}
\text{If } u \text{ dist} &> x \text{ dist } + w_{x,u} \\
\text{then } u \text{ dist} &= x \text{ dist } + w_{x,u}
\end{align*}
\]

Another difference?

First, notice we process (i.e. set distance) nodes in ascending order of distance. Why?

\( \Rightarrow \) Want to process a node that is closer to \( s \) before one that is farther since the closer node may yield a shorter path to the farther one.

For unweighted, we visit nodes exactly in this order (1 hop away, 2 hops away, 3 hops away).

\[ S \rightarrow u \rightarrow x \rightarrow \bar{s} \]

We visit nodes in the same order they should be processed so can use a queue to get next node to process.

\[ S \rightarrow u \rightarrow x \rightarrow \bar{s} \]

Turns out (will prove later) when \( u \text{ dist} \) is minimum distance of all non-processed nodes, \( u \text{ dist} \) is shortest distance from \( s \) to \( u \).
For weighted:

We will visit \( u \) before \( y \) but we should process \( y \) first.

Can't use a queue anymore!

How to get node with minimum current distance?

Use Min Binary Heap!

Difference (2): Use Min Binary Heap to get the node with minimum

In general:

\[ \text{Do not want to process } u \text{ before } y \text{ (even though we visit } u \text{ before } y) \]
"Keep variable 'processed' to keep track of when a node's distance is final.
\nFor all \( v \), \( v.dist = \infty \) for BFS-SP."

/\* Finds shortest path distances from \( s \) to every other vertex in \( G \) *\/
Dijkstra's (\( G, s \)) // \( G = (V, E) \) \( w_{uv} \geq 0 \) for \( (uv) \in E \)
1. For all \( v \in V \), \( v.dist = \infty \), \( v.processed = false \)
2. \( s.dist = 0 \), \( s.processed = true \)
3. For all \( v \), \( v.pred = \) null

// Distances are updated throughout, so store all nodes
4. \( Vheap = BuildHeap (V) \) // min binary heap of nodes
   // ordered by dist
5. While (!\( Vheap.\text{empty}() \))
6. \( v = Vheap.\text{deleteMin}() \)
7. \( v.processed = true \)
8. For (each neighbor \( u \) of \( v \)):
9. \( \text{if} (\! u.processed \text{ and } v.dist + w_{uv} < u.dist) \)
10. \( \{ \text{relaxation} \} \)
11. \( u.dist = v.dist + w_{uv} \) // heap gets updated
12. \( u.pred = v \)
13. updateKey \((u, v.dist + w_{uv})\)
This just gives distances, How to keep track of paths?

\[
\begin{pmatrix}
10 & 90 & 80 \\
20 & 30 & \text{\textcopyright} \\
\end{pmatrix}
\]

Keep variable \( \text{pred} \).
Update \( u \cdot \text{pred} \) when \( u \cdot \text{dist} \) updated.

< Add to code >

\[\text{ex:}\]

\[
\begin{pmatrix}
0 & 2 & 4 & 11 \\
5 & 2 & 9 & \text{\textcopyright} \\
\end{pmatrix}
\]

\( v = s \ b \ d \ a \ c \)
\( \text{pred: null} \ s \ b \ s \ d \)

- getPath(c)
  \( x = e \ d \ b \ s \)
  \( p = c \ast d \ast b \ast s \)

getPath(\( x \)) //return shortest s - x path
\[
\begin{align*}
P &= x \ //\ path \\
\text{while} (x \cdot \text{pred} \neq \text{null}) \\
P &= P \ast \ x \cdot \text{pred} \\
\text{concatenate} \\
X &= x \cdot \text{pred} \\
\text{return concatenе}(P)
\end{align*}
\]
\[ \text{Recall Min Heap to store vertices.} \]

1-3: \( O(|V|) \)

4: \( O(|V|) \)

5-7: for each \( v \in V \), deleteMin: \( O(|V| \cdot \log |V|) \)

S-11: "" (\( uv \) \( \in \mathcal{E} \), update key \( (u, v, \text{dist} + w(u, v)) \): \( O(|E| \cdot \log |V|) \)

Notice Step 8 does not occur \( |V| \) times, only \( |E| \) times.

Total: \( O(|V| + |V| \cdot \log |V| + |E| \cdot \log |V|) = O(|V| \log |V| + |E| \log |V|) \)