Shortest Paths.

Input: A (directed) graph \( G = (V, E, \ell) \). An edge \( e \) has length \( \ell(e) \geq 0 \). A source node \( s \in V \).

(The length of a path \( P \) is defined as \( \ell(P) = \sum_{e \in P} \ell(e) \).)

Output: An array \( d[] \) where \( d[u] = \text{the length of the shortest path between } s \text{ and } u \).

Example:

\[
\begin{align*}
&d[x] = 2 \quad s \rightarrow u \rightarrow x \\
&d[y] = 3 \quad s \rightarrow u \rightarrow x \rightarrow y
\end{align*}
\]

Dijkstra's algorithm. [Dijkstra '59]

We maintain: \( S \subseteq V \), "explored" nodes of \( G \).

\( \forall u \in S \), we have found a shortest path from \( s \) to \( u \).

An array \( d'[\] \) such that:

\[
d'[v] = \begin{cases} 
\text{the length of the shortest path from } s \text{ to } v, \\
\infty & \text{if } v \text{ is within one hop from } s. \\
\min_{e=(u, v)} d(u) + \ell(e), & \text{otherwise.}
\end{cases}
\]

Example: \( S = \)

\[
\begin{align*}
&d'[y] = 4 \quad e = (u, y) \\
&d'[x] = 2 \quad e = (u, x)
\end{align*}
\]
Dijkstra's algorithm

\[ S = \{ s \} \quad d(s) = 0 \]

while \( S \neq V \)

- select a node \( v \in S \) with the smallest \( d'(v) \)

\[ d'(v) = \min_{e=(u,v)} \{ d(u) + l(e) : u \in S \} \]

Add \( v \) to \( S \) and set \( d(v) = d'(v) \)

End while.

1st loop:

\( S = \{ s, u \} \)

\[ d = (0, 1, 2, 0, 0) \]

\[ d' = (0, 1, 2, 2, 4, \infty, \infty) \]

\[ d'(x) = \min (d(u) + l(u,x), 2) \]

\[ d'(s) + l(s,x) = 4 \]

2nd loop:

\( S = \{ s, u, v \} \)

\[ d = (0, 1, 2, -1, -1) \]

\[ d' = (0, 1, 2, 2, 4, 5) \]

\( S \cup V \times Y \geq 2 \)

Eventually \( d(y) = 3 \), but at this point, because we can use at most one edge leaving \( S \), \( d'(y) = 1 \).

Correctness

At any point, \( \forall u \in S, \ d(u) = \text{length of shortest path from } s \text{ to } u \).

Base case: \( S = \{ s \} \).

Inductive step: we need to show that every time we add a new node \( v \) to \( S \), \( d(v) \) is the shortest distance from \( s \) to \( v \).

Suppose \( v \) minimizes \( d'(v) \) among all \( v \in S \).

\[ d(u) + l(u,v) \]

If there were a shorter path from \( s \) to \( v \),

\[ \ell(s \rightarrow u \rightarrow v) > \ell(s \rightarrow u \rightarrow x \rightarrow y) \rightarrow \ldots \neq \ell(s \rightarrow x \rightarrow y) \]

which means \( d'(y) < d'(v) \), a contradiction!
Priority Queues. (see Chapter 2.5 of the textbook) Recall arrays (and linked lists) and each element e has a priority value or key key(e). (in this lecture, smaller keys represent higher priorities.)

Operations: Let H be a priority queue. (that has ≤ n elements at any time)

Runtime = \( O(\log n) \) H. insert \((x, k)\) : insert an element \(x\) with key \(x = k\).

\( O(\log n) \) H. delete \((x)\) : delete \(x\).

\( O(1) \) H. min( ) : return an element \(x\) with the smallest key.

\( O(\log n) \) H. changekey \((x, k)\) : change the key of \(x\) to \(k\).

(We can implement a priority queue using a min-heap and a position array.)

Implementing Dijkstra:

Combine \(d\) and \(d'\) into one array \(d\).

\[ u \in S \quad u \notin S. \]

\[ S = \{s\}, \quad d(s) = 0. \]

For every \(u \in V\), \(u \notin S\):

If \((s, u) \in E\), \(d(u) = \delta(s, u)\) Else \(d(u) = +\infty\).

H. insert \((u, d(u))\); \(\leftarrow\) H. insert() is called \(O(n)\) times

End For.

While \(S \neq V:\) \(\leftarrow\) While loop runs \(O(n)\) times

\[ v = H. \text{min}() ; \quad H. \text{delete}(v) ; \quad \leftarrow\ \text{In total, } H. \text{min}() \text{ and } H. \text{delete}() \]

\[ S = S \cup \{v\} ; \quad \text{are called } O(n) \text{ times.} \]

For every edge \((v, w)\) where \(w \notin S\): \(\leftarrow\) Every edge appears at most once in this For loop.

If \(d(v) + \delta(v, w) < d(w)\):

\[ d(w) = d(v) + \delta(v, w) \]

H. changekey \((w, d(w))\)

End If

End For

End While

Return \(d[\cdot]\).

Overall runtime: other operations like changing \(S\) or \(d(w)\).

\[ O(n \cdot \log n + n + m \cdot \log n + m \cdot \log n + m + n) \]

\[ = O(m \cdot \log n). \]

Comments:

1. What if \( \delta(u, v) < 0 \) ? Dijkstra’s algorithm no longer works!

2. Dijkstra \(\approx\) continuous BFS.

3. If we use “for \(v=1..n\)” to find the \(v\) with the smallest \(d(v)\): Runtime \(O(n^2)\).