Dijkstra’s Algorithm for Shortest Paths
Section 24.3 (CLRS): Sections 4.4 (KT)
Shortest Path Problem

Shortest path network.
- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $\ell_e = \text{length of edge } e$.

Shortest path problem: find shortest directed path from $s$ to $t$.

cost of path = sum of edge costs in path

Cost of path $s$-2-3-5-$t$

$= 9 + 23 + 2 + 16$

$= 48$. 
Single Source Shortest Path Problem:
• Given a digraph $G = (V, E)$
• Numeric edge weights
• Source vertex, $s \in V$
• Determine the distance $\delta(s, v)$ from $s$ to every vertex $v$ in the graph

Are negative weight edges allowed? (could arise in financial transaction networks)

Dijkstra’s algorithm assumes no negative edge weights.
• Computing the distance from source to each vertex (not the actual path)
relax(u, v):
    if d[u] + w(u, v) < d[v]:  # is the path through u shorter?
        d[v] = d[u] + w(u, v)  # yes, then take it
        pred[v] = u  # record that we go through u

Fig. 15: Relaxation.
Dijkstra's Algorithm

Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$
\pi(v) = \min_{e = (u,v) : u \in S} \left( d(u) + \ell_e \right),
$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

![Diagram](image)
Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$.
Dijkstra’s Algorithm: Implementation

Build: Create a priority queue from a list of \( n \) elements, each with an associated key value.

Extract min: Remove (and return a reference to) the element with the smallest key value.

Decrease key: Given a reference to an element in the priority queue, decrease its key value to a specified value, and reorganize if needed.

---

```plaintext
dijkstra(G,w,s) {  
    for each (u in V) {  // initialization  
        d[u] = +infinity  
        mark[u] = undiscovered  
        pred[u] = null  
    }  
    d[s] = 0  // distance to source is 0  
    Q = a priority queue of all vertices u sorted by d[u]  
    while (Q is nonEmpty) {  // until all vertices processed  
        u = extract vertex with minimum d[u] from Q  
        for each (v in Adj[u]) {  // relax(u,v)  
            if (d[u] + w(u,v) < d[v]) {  // relax(u,v)  
                d[v] = d[u] + w(u,v)  
                decrease v’s key in Q to d[v]  
                pred[v] = u  
            }  
        }  
        mark[u] = finished  
    }  
    [The pred pointers define an ‘‘inverted’’ shortest path tree]  
}
```

---
Dijkstra’s Algorithm: Example

\[ T(n,m) = \sum_{u \in V} (\log n + \deg(u) \cdot \log n) = \sum_{u \in V} (1 + \deg(u)) \log n \]

\[ = \log n \sum_{u \in V} (1 + \deg(u)) = (\log n)(n + 2m) = \Theta((n + m) \log n). \]

Since \( G \) is connected, \( n \) is asymptotically no greater than \( m \), so this is \( O(m \log n) \).
Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.
Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ \}\nPQ = \{ s, 2, 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

- **S** = \{ s \}
- **PQ** = \{ 2, 3, 4, 5, 6, 7, t \}

**Graph Representation:**

- **Nodes:** s, 0, 2, 3, 4, 5, 6, 7, t
- **Edges:**
  - s to 0: 9
  - 0 to 2: 14
  - 2 to 3: 24
  - 3 to 4: 19
  - 4 to t: 6
  - 4 to 5: 11
  - 5 to 6: 30
  - 6 to 7: 14
  - 6 to 5: 6
  - 7 to t: 44

**Key Points:**

- **Distance Label:**
  - s: 0
  - 0: 9
  - 2: 9 + 14 = 23
  - 3: 24
  - 4: 19
  - 5: \infty
  - 6: \infty
  - 7: 20
  - t: \infty

- **Decrease Key:**
  - s: 9
  - 2: 14
  - 5: 30
  - 7: 20
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2 \}
PQ = \{ 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2 \}
PQ = \{ 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ \text{s, 2, 6} \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]

\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 6, 7 \}$
$PQ = \{ 4, 5, t \}$

delmin
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ \text{PQ} = \{ t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ PQ = \{ t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ \text{s, 2, 3, 4, 5, 6, 7, t} \}
PQ = \{ \}

[Graph of a network showing distances between nodes]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain \( \pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e \).

- Next node to explore = node with minimum \( \pi(v) \).
- When exploring \( v \), for each incident edge \( e = (v, w) \), update

  \[
  \pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}.
  \]

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by \( \pi(v) \).

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap ( ^\dagger )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>1</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>( m )</td>
<td>1</td>
<td>( \log n )</td>
<td>( \log_d n )</td>
<td>1</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>( n^2 )</td>
<td>( m \log n )</td>
<td>( m \log_{m/n} n )</td>
<td>( m + n \log n )</td>
<td></td>
</tr>
</tbody>
</table>

\( ^\dagger \) Individual ops are amortized bounds
Dijkstra's Algorithm: Proof of Correctness

Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest $s$-$u$ path.

Pf. (by induction on $|S|$)

Base case: $|S| = 1$ is trivial.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let $v$ be the next node added to $S$, and let $u$-$v$ be the chosen edge.
- The shortest $s$-$u$ path plus $(u, v)$ is an $s$-$v$ path of length $\pi(v)$.
- Consider any $s$-$v$ path $P$. We'll see that it's no shorter than $\pi(v)$.
- Let $x$-$y$ be the first edge in $P$ that leaves $S$, and let $P'$ be the subpath to $x$.
- $P$ is already too long as soon as it leaves $S$.

$$\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)$$

- $\ell(P)$: nonnegative weights
- $\ell(P')$: inductive hypothesis
- $\ell(x,y)$: defn of $\pi(y)$
- Dijkstra chose $v$ instead of $y$
Variants of Dijkstra’s

**Vertex weights:** There is a cost associated with each vertex. The overall cost is the sum of vertex and/or edge weights on the path.

**Single-Sink Shortest Path:** Find the shortest path from each vertex to a sink vertex t.

**Multi-Source/Multi-Sink:** You are given a collection of source vertices \{s_1, \ldots, s_k\}. For each vertex find the shortest path from its nearest source. (Analogous for multi-sink.)

**Multiplicative Cost:** Define the cost of a path to be the product of the edge weights (rather than the sum.) If all the edge weights are at least 1, find the single-source shortest path.
Give the final $d$ and $\pi$ values of the vertices obtained by running Dijkstra’s algorithm on the directed graph below with source $A$. 
Next Time

- Bellman-Ford Shortest Paths
- Section 24.1 (CLRS)
- Section 6.8(KT)