Shortest Paths

- Shortest Paths
- Properties of Shortest Paths
- Dijkstra’s Algorithm
- Shortest Paths on DAGs

Shortest paths. Given a directed, weighted graph $G$ and vertex $s$, find shortest path from $s$ to all vertices in $G$.

Shortest path tree. Represent shortest paths in a tree from $s$.
Applications

- Routing, scheduling, pipelining, ...

Properties of Shortest Paths

- Assume for simplicity:
  - All vertices are reachable from s.
  - \( \Rightarrow \) a (shortest) path to each vertex always exists.

Shortest Paths

- Shortest Paths
- Properties of Shortest Paths
- Dijkstra’s Algorithm
- Shortest Paths on DAGs

Properties of Shortest Paths

- **Subpath property.** Any subpath of a shortest path is a shortest path.
- **Proof.**
  - Consider shortest path from s to t consisting of \( p_1 \), \( p_2 \) and \( p_3 \).
  - Assume \( q_2 \) is shorter than \( p_2 \).
  - \( \Rightarrow \) Then \( p_1 \), \( q_2 \) and \( p_3 \) is shorter than \( p \).
Shortest Paths

- Shortest Paths
- Properties of Shortest Paths
- Dijkstra’s Algorithm
- Shortest Paths on DAGs

Dijkstra’s Algorithm

**Goal.** Given a directed, weighted graph with non-negative weights and a vertex \( s \), compute shortest paths from \( s \) to all vertices.

**Dijkstra’s algorithm.**
- Maintains distance estimate \( v.d \) for each vertex \( v \) = length of shortest known path from \( s \) to \( v \).
- Updates distance estimates by relaxing edges.

Dijkstra’s Algorithm

- Initialize \( s.d = 0 \) and \( v.d = \infty \) for all vertices \( v \in V\{s\} \).
- Grow tree \( T \) from \( s \).
- In each step, add vertex with smallest distance estimate to \( T \).
- Relax all outgoing edges of \( v \).

**Relax** \((u,v)\)

\[
\text{if } (v.d > u.d + w(u,v)) \\
v.d = u.d + w(u,v)
\]
Dijkstra's Algorithm

- **Lemma.** Dijkstra's algorithms computes shortest paths.
- **Proof.**
  - Consider some step after growing tree $T$ and assume distances in $T$ are correct.
  - Consider closest vertex $u$ of $s$ not in $T$.
  - Shortest path from $s$ to $u$ ends with an edge $(v,u)$.
    - $v$ is closer than $u$ to $s$ $\Rightarrow$ $v$ is in $T$. ($u$ was closest not in $T$)
    - $\Rightarrow$ shortest path to $u$ is in $T$ except last edge $(u,v)$.
    - Dijkstra adds $(u,v)$ to $T$ $\Rightarrow$ $T$ is shortest path tree after $n-1$ steps.

Dijkstra's Algorithm

- **Implementation.** How do we implement Dijkstra's algorithm?
- **Challenge.** Find vertex with smallest distance estimate.
Dijkstra’s Algorithm

Implementation. Maintain vertices outside T in priority queue.

- Key of vertex \( v = v.d \).
- In each step:
  - Find vertex \( u \) with smallest distance estimate = \text{EXTRACT-MIN}
  - Relax edges that \( u \) point to with \text{DECREASE-KEY}.

**Dijkstra’s Algorithm**

\[
\begin{align*}
\text{Dijkstra}(G, s) \\
\text{for all vertices } v \in V \\
v.d &= \infty \\
v.\pi &= \text{null} \\
\text{INSERT}(P, v) \\
\text{DECREASE-KEY}(P, s, 0) \\
\text{while } (P \neq \emptyset) \\
u &= \text{EXTRACT-MIN}(P) \\
\text{for all } v \text{ that } u \text{ point to} \\
\text{RELAX}(u, v) \\
\text{if } (v.d > u.d + w(u, v)) \\
v.d &= u.d + w(u, v) \\
v.\pi &= u
\end{align*}
\]

Time.
- \( n \) \text{EXTRACT-MIN}
- \( n \) \text{INSERT}
- \(< m \) \text{DECREASE-KEY}
- Total time with min-heap. \( O(n \log n + n \log n + m \log n) = O(m \log n) \)

Dijkstra’s Algorithm

- Priority queues and Dijkstra’s algorithm. Complexity of Dijkstra’s algorithm depend on priority queue.
  - \( n \) \text{INSERT}
  - \( n \) \text{EXTRACT-MIN}
  - \(< m \) \text{DECREASE-KEY}

<table>
<thead>
<tr>
<th>Priority queue</th>
<th>\text{INSERT}</th>
<th>\text{EXTRACT-MIN}</th>
<th>\text{DECREASE-KEY}</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(1) )</td>
<td>( O(n) )</td>
<td>( O(1) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
<td>( O(m \log n) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(1)^* )</td>
<td>( O(\log n)^* )</td>
<td>( O(1)^* )</td>
<td>( O(m + n \log n) )</td>
</tr>
</tbody>
</table>

\* = amortized

Greed. Dijkstra’s algorithm is a greedy algorithm.

Edsger W. Dijkstra

- Edsger Wybe Dijkstra (1930-2002)
- Contributions. Foundations for programming, distributed computation, program verifications, etc.
- Quotes. “Object-oriented programming is an exceptionally bad idea which could only have originated in California.”
- “The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.”
- “APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”
Shortest Paths on DAGs

- Challenge. Is it computationally easier to find shortest paths on DAGs?
- DAG shortest path algorithm.
  - Process vertices in topological order.
  - For each vertex \( v \), relax all edges from \( v \).
  - Also works for negative edge weights.

- Implementation.
  - Sort vertices in topological order.
  - Relax outgoing edges from each vertex.
  - Total time. \( O(m + n) \).

Lemma. Algorithm computes shortest paths in DAGs.

Proof.
- Consider some step after growing tree \( T \) and assume distances in \( T \) are correct.
- Consider next vertex \( u \) of \( s \) not in \( T \).
- Any path to \( u \) consists vertices in \( T \) + edge \( e \) to \( u \).
- Edge \( e \) is relaxed \( \implies \) distance to \( u \) is shortest.
Shortest Paths Variants

• Vertices
  • Single source.
  • Single source, single target.
  • All-pairs.

• Edge weights.
  • Non-negative.
  • Arbitrary.
  • Euclidian distances.

• Cycles.
  • No cycles
  • No negative cycles.

Shortest Paths

• Shortest Paths
• Properties of Shortest Paths
• Dijkstra’s Algorithm
• Shortest Paths on DAGs