Data Structures

Single-Source Shortest Paths
Weight of a Path

- Let $G = (V, E)$ be a directed graph
- $w : E \rightarrow \mathbb{R}$ a weight function

The weight of a path $p = (v_0, v_1, \ldots, v_k)$ is

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$
Shortest Path

• Let \( u, v \in V \)

• The **shortest-path weight** \( u \) to \( v \) is

\[
\delta(u, v) = \begin{cases} 
\min \{ w(p) : u \leadsto v \} & \text{if there exists a path } u \leadsto v \\ 
\infty & \text{otherwise} 
\end{cases}
\]

• The **shortest path** \( u \) to \( v \) is any path \( p \) such that \( w(p) = \delta(u, v) \).
Shortest Path

- Representation
  - shortest-path weight
  - shortest path

- The shortest path is not unique
Variants

• **Single-source**
  – Find shortest paths from a given source vertex \( s \in V \) to every vertex \( v \in V \)

• Other variants:
  – **Single-destination**
    • Find shortest paths to a given destination vertex from every vertex \( v \in V \)
  – **Single-pair**
    • Find shortest path from \( u \) to \( v \)
    • No way known that’s better in worst case than solving single-source.
  – **All-pairs**
    • Find shortest path from \( u \) to \( v \) for all \( u, v \in V \)
Cycles

- **Shortest paths can’t contain cycles**
  - **Negative-weight** cycles
    - Negative weights are ok, as long as there are no negative-weight cycles
    - Otherwise, we can just keep going around it, and get \(-\infty\) for all \(v\) on the cycle
  - **Positive-weight** cycles - we can get a shorter path by omitting the cycle
  - **Zero-weight** cycles - no reason to use them
Generic Single-source shortest paths

genericSingleSourceShortestPath(V, s)

initSingleSource(V, s)

repeatedly relax edges

initSingleSource(V, s)

for each \( v \in V \) do

\( d[v] \leftarrow \infty \)

\( \pi[v] \leftarrow \text{null} \)

\( d[s] \leftarrow 0 \)

relax(u, v, w)

if \( d[v] > d[u] + w(u, v) \) then

\( d[v] \leftarrow d[u] + w(u, v) \)

\( \pi[v] \leftarrow u \)
Properties

• **Triangle inequality**
  – For all \((u, v) \in E\), we have \(\delta(s, v) \leq \delta(s, u) + w(u, v)\)

• **Upper-bound property**
  – Always have \(d[v] \geq \delta(s, v)\) for all \(v\)
  – Once \(d[v] = \delta(s, v)\), it never changes again

• **No-path property**
  – If \(\delta(s, v) = \infty\), then \(d[v] = \infty\) always

• **Convergence property**
  – If \(s \leadsto u \rightarrow v\) is a shortest path, \(d[u] = \delta(s, u)\), and we call \(\text{relax}(u, v, w)\),
    then \(d[v] = \delta(s, v)\) afterward

• **Path relaxation property**
  – Let \(p = v_0, v_1, \ldots, v_k\) be a shortest path from \(s = v_0\) to \(v_k\)
  – If we relax, in order, \((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\), even intermixed
    with other relaxations, then \(d[v_k] = \delta(s, v_k)\)
Single-source shortest paths in a DAG

```plaintext
dagShortestPaths(V, E, w, s)
topologically sort V
initSingleSource(V, s)
  for each vertex u, taken in topologically sorted order do
    for each vertex v ∈ Adj[u] do
      relax(u, v, w)

initSingleSource(V, s)
  for each v ∈ V do
    d[v] ← ∞
    π[v] ← null
  d[s] ← 0

relax(u, v, w)
  if d[v] > d[u] + w(u, v) then
    d[v] ← d[u] + w(u, v)
    π[v] ← u

Time: Θ(V + E)
```
Correctness

- Edges of any path must be relaxed in order of appearance in the path
- Hence, edges on any shortest path are relaxed in order
- Hence, by path-relaxation property, correct
Dijkstra’s algorithm

dijkstra(V, E, w, s) // No negative-weight edges
initSingleSource(V, s)
S ← ∅
Q ← V // priority queue
while (Q ≠ ∅) do
    u ← extractMin(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do
        relax(u, v, w)

initSingleSource(V, s)
for each v ∈ V do
    d[v] ← ∞
    π[v] ← null
    d[s] ← 0

relax(u, v, w)
if d[v] > d[u] + w(u, v) then
    d[v] ← d[u] + w(u, v)
    π[v] ← u
Correctness

• Claim
  – **Initially**: $S = \emptyset$
  – **Loop invariant**: $d[v] = \delta(s, v)$ for all $v \in S$
  – **At end**: $S = V$, hence $d[v] = \delta(s, v)$ for all $v \in V$

• Proof
  – Need to show that $d[u] = \delta(s, u)$ when $u$ is added to $S$
  – By contradiction
Analysis

• Binary Heap
  – Each operation takes $O(\lg V)$ time
  – Total $O(E \lg V)$.  
• Fibonacci Heap
  – There are $O(V)$ extractMin, taking $O(\lg V)$ amortized time each
  – There are $O(E)$ relax, taking $O(1)$ amortized time each
  – Total $O(V \lg V + E)$

```plaintext
dijkstra(V, E, w, s) // No negative-weight edges
initSingleSource(V, s)
S ← ∅
Q ← V // priority queue
while (Q ≠ ∅) do
  u ← extractMin(Q)
  S ← S ∪ {u}
  for each vertex v ∈ Adj[u] do
    relax(u, v, w)
```