Shortest Paths

The problem:

• Let $G = (V, E)$ be a directed graph
• Edge weights $\text{wt} : E \to \mathbb{R}$
• Convention: $\text{wt}(u, v) := \infty$ if $(u, v) \notin E$
• The weight of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$:

$$\text{wt}(p) := \sum_{i=1}^{k} \text{wt}(v_{i-1}, v_i)$$

• The shortest distance from $u$ to $v$:

$$\delta(u, v) := \min \{ \text{wt}(p) : p \text{ is a path from } u \text{ to } v \}$$
Some extremes:

- If there is no path from $u$ to $v$, then $\delta(u, v) := \infty$
- If there is a path from $u$ to $v$ that contains a negative weight cycle, then $\delta(u, v) := -\infty$

Cycles:

- A shortest path cannot contain either:
  - a negative weight cycle, or
  - a positive weight cycle

  but may contain a zero-weight cycle

- If there is a shortest path:
  - there is always a shortest path with no cycles
  - there is always a shortest path with $\leq |V| - 1$ edges
Shortest Path Variations:

- Single source
- Single destination
- Single pair
- All pairs
Single source shortest paths

**Goal:** compute shortest paths from a given node $s$ to all other nodes

We will calculate $d[ν] = δ(s, ν)$ for all $ν ∈ V$

We will also calculate an implicit “shortest path tree”:

- $π[ν] =$ predecessor of $ν$ on the tree path from $s$ to $ν$

Code to print a shortest path to $ν$, in reverse:

```plaintext
while $ν ≠ s$ do: print $ν$, $ν ← π[ν]$
```
A shortest path tree:
Dijkstra’s Algorithm

Assumption: No negative edge weights

Idea:

Beginning at s, we grow a shortest path tree, edge by edge

We will use a “greedy” strategy, choosing the edge that yields a new path of least weight

We will maintain:

- a set $R$ of nodes in current shortest path tree
- a set $Q$ of “fringe nodes” — successors of some tree node — we will move nodes from $Q$ to $R$ one at a time
- for each $v \in V$:
  - $d[v]$: weight of current best path to $v$
  - $\pi[v]$: predecessor on current best path to $v$
Algorithm:

// Initialization
for each \( v \in V \):
    \( d[v] \leftarrow \infty \) // weight of current best path to \( v \)
    \( \pi[v] \leftarrow Nil \) // predecessor on current best path to \( v \)

\( d[s] \leftarrow 0 \)
\( R \leftarrow \emptyset \) // tree nodes
\( Q \leftarrow \{s\} \) // fringe nodes

// Main loop
while \( Q \) not empty
    remove \( u \) from \( Q \) with minimal \( d[u] \)
    add \( u \) to \( R \)
    for each \( v \in \text{Successor}(u) \) do // explore edge \( u \rightarrow v \)
        if \( d[u] + \text{wt}(u, v) < d[v] \) then
            if \( d[v] = \infty \) then add \( v \) to \( Q \)
            \( d[v] \leftarrow d[u] + \text{wt}(u, v) \)
            \( \pi[v] \leftarrow u \)

See Dijkstra Demo
Correctness

*Observation 1:* Every node gets added to $Q$ at most once, and every edge is explored at most once; a node is added to $Q$ $\iff$ it is reachable from $s$

*Observation 2:* for every $v \in V$, the value $d[v]$ only decreases as the algorithm proceeds, but never falls below $\delta(s, v)$ — this is because $d[v]$ is either $\infty$ or measures the weight of some path from $s$ to $v$

**Claim:** whenever we remove a node $u$ from $Q$, we have

$$d[u] = \delta(s, u)$$

We prove this by induction on the number of times we remove a node

**Base case (first loop iteration):** we remove $s$, and $d[s] = 0 = \delta(s, s)$ ✓

**Induction step:**

- Consider a point in time at the beginning of a subsequent loop iteration
- $R$ contains all nodes removed from $Q$ in previous loop iterations (in particular, $R$ contains $s$)
- assume $d[x] = \delta(s, x)$ for all $x \in R$ (induction hypothesis)
- want to show that $d[u] = \delta(s, u)$

By Observation 2, we know $d[u] \geq \delta(s, u)$

So we want to show that $d[u] \leq \delta(s, u)$
Correctness (cont’d)

We want to show that $d[u] \leq \delta(s, u)$

Consider a shortest path $q$ from $s$ to $u$, let $y$ be the first node along the path outside $R$, and let $x$ be its predecessor (which is in $R$):

\[
\begin{align*}
\text{s} & \xrightarrow{p} \text{x} \rightarrow \text{y} \xrightarrow{q} \text{u} \\
\end{align*}
\]

$p$ is the subpath from $s$ to $x$)

We have:

\[
\begin{align*}
\delta(s, u) &= \text{wt}(q) \quad [q \text{ is a shortest path from } s \text{ to } u] \\
&\geq \text{wt}(p) + \text{wt}(x, y) \quad [\text{edge weights are nonnegative}] \\
&\geq \delta(s, x) + \text{wt}(x, y) \\
&= d[x] + \text{wt}(x, y) \quad [\text{induction hypothesis and } x \in R] \\
&\geq d[y] \quad [x \rightarrow y \text{ explored when } x \text{ was added to } R] \\
&\geq d[u] \quad [u \text{ was chosen with minimal } d\text{-value}]
\end{align*}
\]

Finally: when the algorithm finishes, for every node $v$ that never entered $Q$, we have $d[v] = \infty = \delta(s, v)$, as there is no path from $s$ to $v$
An additional property

Using a very similar induction proof, one can easily show that at any point in time, we have

$$\delta(s, x) \leq \delta(s, y)$$

for all $x \in R$ and $y \in V \setminus R$

This implies the following:

*Nodes are removed from $Q$ in order of increasing distance from $s*$
Implementation: priority queue

- \( n := |V| \) ExtractMin’s / Insert’s
- \( m := |E| \) Decrease’s

Running Time:

- heap: \( O((n + m) \log n) \)
  - all operations: \( O(\log n) \)
- unsorted list: \( O(n^2) \)
  - \( \text{ExtractMin: } O(n), \text{ Decrease / Insert: } O(1) \)
- “Fibonacci heap”: \( O(n \log n + m) \) (an advanced data structure)
A linear time special case:

- assume all edges weights are integers in the range $[0..B]$ for some small bound $B$
- we can implement Dijkstra in time $O(nB + m)$
- so for constant $B$, this is linear time

Some observations:

- recall: nodes are removed from $Q$ in order of increasing distance from $s$
- recall: the $d$-value of any node only decreases over time
- the maximum distance from $s$ of any node is $\leq (n - 1)B$ — why?
- the maximum $d$-value of any node in $Q$ is $\leq nB$ — why?
An implementation (first attempt):

• Use an array $A[0..nB]$

• Entry $A[i]$ is a “bucket” of nodes in $Q$ whose current $d$-value is $i$

• Initialize $next \leftarrow 0$

• $ExtractMin$:
  - while $A[next]$ empty do: increment $next$
  - remove and return any node from bucket $A[next]$
  - $Total cost$: $O(nB)$

• $Insert / Decrease$:
  - add/move node to appropriate bucket
  - $Key fact$: the node will never land in a bucket of index smaller than $next$ — why?
  - $Total cost$: $O(n + m)$
More observations:

- Let $\min(Q) :=$ the smallest $d$-value for any node in $Q$
- when a node is added to $Q$, it’s $d$-value is at most $\min(Q) + B$ — why?
- at any point in time, the $d$-value of any node in $Q$ lies in $[\min(Q) .. \min(Q) + B]$ — why?
- In the above implementation, at any point in time, most buckets are empty:
  - only entries $\min(Q) .. \min(Q) + B$ of $A$ are non-empty

- **A more space efficient representation:**
  - use a “circular array” $A'[0 .. B]$ to represent the non-empty part of $A$
  - $A[i]$ is stored at $A'[i \mod (B + 1)]$
Problem solving by reduction

You are given a directed graph $G = (V, E)$ along with nodes $s, t \in V$

Edges are colored $\text{red}$ and $\text{green}$

A path is called $\text{admissible}$ if it contains at most 3 $\text{red}$ edges

Determine if there is an admissible path from $s$ to $t$, and if so, find one with the minimal number of green edges

Solve this problem by $\text{recasting}$ it as a standard shortest path problem: running time $O(|V| + |E|)$. 
Idea:

Make 4 copies of graph: $G^{(0)}, G^{(1)}, G^{(2)}, G^{(3)}$

Green edge $u \rightarrow v$ in $G$ maps to $u^{(i)} \rightarrow v^{(i)}$ with edge weight 1

Red edge $u \rightarrow v$ in $G$ maps to $u^{(i)} \rightarrow v^{(i+1)}$ with edge weight 0

Add 0-weight edges from each $t^{(i)}$ to new node $t'$

Find shortest path from $s^{(0)}$ to $t'$