CS 380
ALGORITHM DESIGN AND ANALYSIS

Lecture 20: Single Source Shortest Path
Text Reference: Chapter 24

Shortest Paths

- Finding the shortest path between two nodes comes up in many applications
  - Transportation problems
  - Motion planning
  - Communication problems
  - Six degrees of separation!

https://visualgo.net/en
(although some algorithms are done differently than in our text)
Shortest Paths

• In an unweighted graph, the cost of a path is just the number of edges on the shortest paths
• What algorithm have we already covered that can do this?

Shortest Paths Problems

Input: a directed graph $G = (V, E)$ and a weight function $w: E \rightarrow R$

• The **weight of a path** $p = v_0, v_1, v_2, \ldots v_k$ is
  \[
  w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)
  \]
• The **shortest-path weight** $\delta(u, v)$ is the minimum of $w(p)$ over all paths from $u$ to $v$.
• The **shortest-path from $u$ to $v$** is any path whose weight is $\delta(u, v)$. 
Example

Variants

- Single Source Shortest Paths
- Single Destination Shortest Paths
- Single Pair Shortest Path
- All Pairs Shortest Paths
Subpaths

- Subpaths of shortest paths are shortest paths
- Lemma: If \( p = v_0, v_1, v_2, ..., v_j, ..., v_k \) is a shortest path from \( v_0 \) to \( v_k \), then \( p' = v_0, v_1, v_2, ..., v_j \) is a shortest path from \( v_0 \) to \( v_j \) for any \( 1 \leq j \leq k \)

Negative Weight Edges and Cycles

- Fine, as long as no negative-weight cycles are reachable from the source (Why an issue?)

- Note that some algorithms will only work if there are NO negative weight edges in the graph
Cycles

• Shortest paths can’t contain cycles:
  • Already ruled out negative-weight cycles
  • Positive-weight → we can get a shorter weight by omitting the cycle
  • Zero-weight: no reason to use them → assume that our solutions will not use them

Output

• For each vertex $v$ in $V$:
  • $d[v] = \delta(s,v)$ (or $v.d$)

• $\pi[v] = \text{predecessor of } v \text{ on a shortest path from } s$ (or $v.\pi$)
Initialization

- All the shortest-paths algorithms start with

\[
\text{INIT-SINGLE-SOURCE}(G, s)
\]

\[
\text{for each } v \in G. V \\
\quad v.d = \infty \\
\quad v.\pi = \text{NIL} \\
\quad s.d = 0
\]

Relaxation

- The process of relaxing an edge \((u, v)\) consists of testing whether we can improve the shortest path to \(v\) found so far by going through \(u\) and, if so, updating \(d[v]\) (i.e. \(v.d\)) and \(\pi[v]\) (i.e. \(v.\pi\))

\[
\text{RELAX}(u, v, w)
\]

\[
\text{if } v.d > u.d + w(u, v) \\
\quad v.d = u.d + w(u, v) \\
\quad v.\pi = u
\]
Single-Source Shortest-Paths

- For all single-source shortest-paths algorithms we’ll look at:
  - Start by calling INIT-SINGLE-SOURCE
  - Then relax edges
- The algorithms differ in the order and how many times they relax each edge
Bellman-Ford Algorithm

- Allows negative-weight edges
- Computes $d[v]$ and $\pi[v]$ for all $v$ in $V$
- Returns true if no negative-weight cycles are reachable from $s$, false otherwise
Example

Dijkstra’s Algorithm

- No negative-weight edges
- Essentially a weighted version of BFS
  - Instead of a FIFO Queue, use a min priority queue
  - Keys are shortest-path weights (d[v])
- Have two sets of vertices
  - S = vertices whose final shortest-path weights are determined
  - Q = priority queue = V - S
DIJKKTRA

DIJKKTRA(G, w, s)
INIT-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V  // i.e., insert all vertices into Q
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)

Example

```
  s
  +---------+  +---------+  +---------+  +---------+
  |          |  |          |  |          |  |          |
  +---------+  +---------+  +---------+  +---------+
     10     3     5     4     2     1
     x      y      z
```
Your Turn

- What is the single-source shortest-path tree starting at a?