Lecture 11: Weighted Shortest Paths

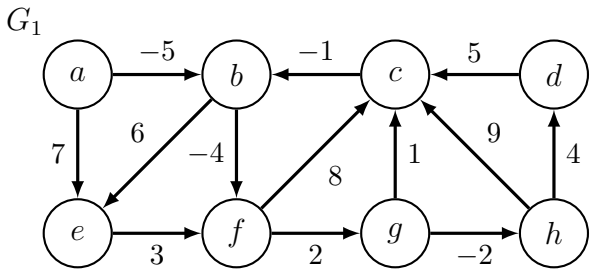
Review

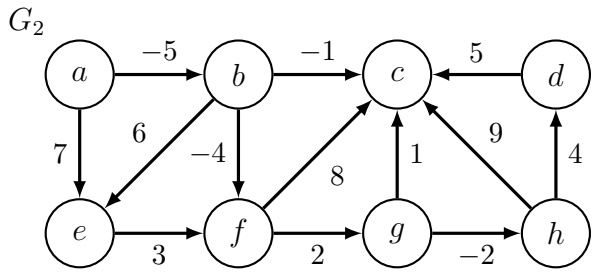
- Single-Source Shortest Paths with BFS in $O(|V| + |E|)$ time (return distance per vertex)
- Single-Source Reachability with BFS or DFS in $O(|E|)$ time (return only reachable vertices)
- Connected components with Full-BFS or Full-DFS in $O(|V| + |E|)$ time
- Topological Sort of a DAG with Full-DFS in $O(|V| + |E|)$ time
- **Previously:** distance = number of edges in path **Today:** generalize meaning of distance

Weighted Graphs

- A **weighted graph** is a graph $G = (V, E)$ together with a weight function $w : E \to \mathbb{Z}$
- i.e., assigns each edge $e = (u, v) \in E$ an integer weight: $w(e) = w(u, v)$
- Many applications for edge weights in a graph:
  - distances in road network
  - latency in network connections
  - strength of a relationship in a social network
- Two common ways to represent weights computationally:
  - Inside graph representation: store edge weight with each vertex in adjacency lists
  - Store separate Set data structure mapping each edge to its weight
- We assume a representation that allows querying the weight of an edge in $O(1)$ time

Examples





Weighted Paths

- The **weight** $w(\pi)$ of a path π in a weighted graph is the sum of weights of edges in the path
- The **(weighted) shortest path** from $s \in V$ to $t \in V$ is path of minimum weight from s to t
- $\delta(s,t) = \inf\{w(\pi) \mid \text{path } \pi \text{ from } s \text{ to } t\}$ is the **shortest-path weight** from s to t
- (Often use "distance" for shortest-path weight in weighted graphs, not number of edges)
- As with unweighted graphs:
  - $\delta(s,t) = \infty$ if no path from s to t
  - Subpaths of shortest paths are shortest paths (or else could splice in a shorter path)
- Why infimum not minimum? Possible that no finite-length minimum-weight path exists
- When? Can occur if there is a negative-weight cycle in the graph, Ex: (b, f, g, c, b) in G_1
- A **negative-weight cycle** is a path π starting and ending at same vertex with $w(\pi) < 0$
- $\delta(s,t) = -\infty$ if there is a path from s to t through a vertex on a negative-weight cycle
- If this occurs, don't want a shortest path, but may want the negative-weight cycle

Weighted Shortest Paths Algorithms

- Next four lectures: algorithms to find shortest-path weights in weighted graphs
- (No parent pointers: can reconstruct shortest paths tree in linear time after. Next page!)
- Already know one algorithm: Breadth-First Search! Runs in $O(|V| + |E|)$ time when, e.g.:
  - graph has positive weights, and all weights are the same
  - graph has positive weights, and sum of all weights at most $O(|V| + |E|)$
- For general weighted graphs, we don't know how to solve SSSP in $O(|V| + |E|)$ time
- But if your graph is a **Directed Acyclic Graph** you can!

Restrictions		SSSP Algorithm		
Graph	Weights	Name	Running Time $O(\cdot)$	Lecture
General	Unweighted	BFS	$\|V\| + \|E\|$	L09
DAG	Any	DAG Relaxation	$\|V\| + \|E\|$	L11 (Today!)
General	Any	Bellman-Ford	$\|V\| \cdot \|E\|$	L12
General	Non-negative	Dijkstra	$\|V\| \log \|V\| + \|E\|$	L13

Shortest-Paths Tree

- For BFS, we kept track of parent pointers during search. Alternatively, compute them after!
- If know $\delta(s, v)$ for all vertices $v \in V$, can construct shortest-path tree in $O(|V| + |E|)$ time
- For weighted shortest paths from s, only need parent pointers for vertices v with finite $\delta(s, v)$

- Initialize empty P and set $P(s) =$ `None`
- For each vertex $u \in V$ where $\delta(s, v)$ is finite:
  - For each outgoing neighbor $v \in \text{Adj}^+(u)$:
    - If $P(v)$ not assigned and $\delta(s, v) = \delta(s, u) + w(u, v)$:
      - There exists a shortest path through edge (u, v), so set $P(v) = u$
- Parent pointers may traverse cycles of zero weight. Mark each vertex in such a cycle.
- For each unmarked vertex $u \in V$ (including vertices later unmarked):
  - For each $v \in \text{Adj}^+(u)$ where v is marked and $\delta(s, v) = \delta(s, u) + w(u, v)$:
    - Unmark vertices in cycle containing v by traversing parent pointers from v
    - Set $P(v) = u$, breaking the cycle

- **Exercise:** Prove this algorithm correctly computes parent pointers in linear time
- Because we can compute parent pointers afterward, we focus on computing distances

DAG Relaxation

- **Idea!** Maintain a distance estimate $d(s, v)$ (initially ∞) for each vertex $v \in V$, that always upper bounds true distance $\delta(s, v)$, then gradually lowers until $d(s, v) = \delta(s, v)$
- When do we lower? When an edge violates the triangle inequality!
- **Triangle Inequality:** the shortest-path weight from u to v cannot be greater than the shortest path from u to v through another vertex x, i.e., $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$ for all $u, v, x \in V$
- If $d(s, v) > d(s, u) + w(u, v)$ for some edge (u, v), then triangle inequality is violated :(
- Fix by lowering $d(s, v)$ to $d(s, u) + w(u, v)$, i.e., **relax** (u, v) to satisfy violated constraint
- **Claim:** Relaxation is **safe**: maintains that each $d(s, v)$ is weight of a path to v (or ∞) $\forall v \in V$
- **Proof:** Assume $d(s, v')$ is weight of a path (or ∞) for all $v' \in V$. Relaxing some edge (u, v) sets $d(s, v)$ to $d(s, u) + w(u, v)$, which is the weight of a path from s to v through u. □

- Set $d(s, v) = \infty$ for all $v \in V$, then set $d(s, s) = 0$
- Process each vertex u in a topological sort order of G:
  - For each outgoing neighbor $v \in \text{Adj}^+(u)$:
    - If $d(s, v) > d(s, u) + w(u, v)$:
      - relax edge (u, v), i.e., set $d(s, v) = d(s, u) + w(u, v)$

- **Example:** Run DAG Relaxation from vertex a in G_2

Correctness

- **Claim:** At end of DAG Relaxation: $d(s, v) = \delta(s, v)$ for all $v \in V$
- **Proof:** Induct on k: $d(s, v) = \delta(s, v)$ for all v in first k vertices in topological order
  - Base case: Vertex s and every vertex before s in topological order satisfies claim at start
  - Inductive step: Assume claim holds for first k' vertices, let v be the $(k' + 1)^{\text{th}}$
  - Consider a shortest path from s to v, and let u be the vertex preceding v on path
  - u occurs before v in topological order, so $d(s, u) = \delta(s, u)$ by induction
  - When processing u, $d(s, v)$ is set to be no larger ($\leq$) than $\delta(s, u) + w(u, v) = \delta(s, v)$
  - But $d(s, v) \geq \delta(s, v)$, since relaxation is safe, so $d(s, v) = \delta(s, v)$ □
- Alternatively:
  - For any vertex v, DAG relaxation sets $d(s, v) = \min\{d(s, u) + w(u, v) \mid u \in \text{Adj}^-(v)\}$
  - Shortest path to v must pass through some incoming neighbor u of v
  - So if $d(s, u) = \delta(s, u)$ for all $u \in \text{Adj}^-(v)$ by induction, then $d(s, v) = \delta(s, v)$ □

Running Time

- Initialization takes $O(|V|)$ time, and Topological Sort takes $O(|V| + |E|)$ time
- Additional work upper bounded by $O(1) \times \sum_{u \in V} \deg^+(u) = O(|E|)$
- Total running time is linear, $O(|V| + |E|)$

6.006 Introduction to Algorithms

Spring 2020