Eksamensnoter - Amortized Analysis

André Oskar Andersen (wpr684)

March 26, 2020

25 Single-Source Shortest Paths

- In a *shortest-paths problem*, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights. The *weight* $w(p)$ of path $p = \{v_0, v_1, ..., v_k\}$ is the sum of the weights of its constituent edges:

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

- We define the *shortest-path weight* $\delta(u, v)$ from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

  A *shortest path* from vetex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$

Variants

- In the *single-source shortest-paths problem* we are given a graph $G = (V, E)$ and we want to find a shortest path from a given *source* vertex $s \in V$ to each vertex $v \in V$

- In a *single-destination shortest-paths problem* the goal is to find a shortest path to a given *destination* vertex t from each vertex v.

- In a *single-pair shortest-path problem* the goal is to find a shortest path from u to v for given vertices u and v.

- In a *all-pair shortest-paths problem* the goal is to find a shortest path from u to v for every pair of vertices u and v. Although we can sole this problem by running a single-source algorithm once from each vertex, we usually can solve it faster

Optimal substructure of a shortest path

- Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it

- The following lemma states the optimal-substructure property of shortest paths more precisely:
  Lemma 24.1 (Subpaths of shortest paths are shortest paths):

  - Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \{v_0, v_1, ..., v_k\}$ be a shortest path from vertex v_0 vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \{v_i, v_{i+1}, ..., v_j\}$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Negative-weight edges

- Some isntances of the single-source shortest-paths problem may include edges whose weights are negative. If there is a negative-weight cycle on some path from s to v, we define $\delta(s, v) = -\infty$
- Dijkstra's algorithm assume that alle edge weights in the input graph are nonnegative. The Bellman-Ford algorithm allow negative-weight edges in the input graph and produce a correct answer as long as no negative-weight cycles are reachable from the source. Typically, if there is such a negative-weight cycle, the algorithm can detect and report its existence

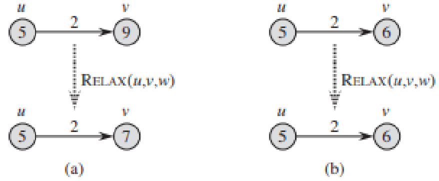
Cycles

- Without loss of generality we can assume that when we are finding shortest paths, they have no cycles, i.e., they are simple paths.

Representing shortest paths

- We often wish to compute not only shortest-path weights, but the vertices on shortest paths as well
- Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ as *predecessor* $v.\pi$ that is either another vertex or `NIL`
- In the mdst of executing a shortest-paths algorithm, howeever, the π values might not indicate shortest paths. We shall be interested in the *predecessor subgraph* $G_\pi = (V_\pi, E_\pi)$ induced by the π values. Here we define the vertex set V_π to be the set of vertices G with non-`NIL` predecessors, plus the source s:
  $$V_\pi = \{v \in V : v.\pi \neq \texttt{NIL}\} \cup \{s\}$$
  The directed edge set E_π is the set of edges induced by the π values for vertices in V_π:
  $$E_\pi = \{(v.\pi, v) \in E : v \in V_\pi - \{s\}\}$$
- A *shortest-paths tree* rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subset V$ and $E' \subset E$ such that
  1. V' is the set of vertices reachable from s in G
  2. G' forms a rooted tree with root s
  3. for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G

Relaxation



- For each vertex $v \in V$, we maintai nan attribute $v.d$ which is an upper bound on the weight of a shortest path from source s to v. We call $v.d$ a *shortest-path estimate*

- Afer initialization, we have $v.\pi = \texttt{NIL}$ for all $v \in V$, $s.d = 0$, and $v.d = \infty$ for $v \in V - \{s\}$

- 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 $v.d$ and $v.\pi$.

Properties of shortest paths and relaxation

- *Triangle inequality*: For any edge $(u, v) \in E$ we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$

- *Upper-bound property*: We always have $v.d \geq \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes

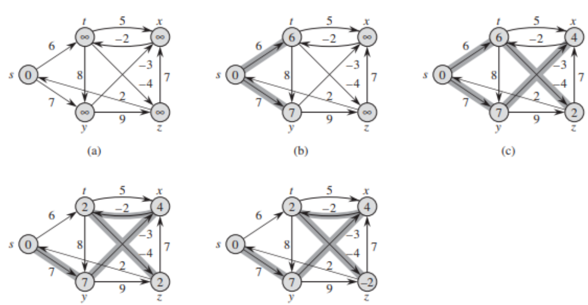
- *No-path property*: If there is not path from s to v, then we always have $v.d = \delta(s, v) = \infty$

- *Convergence property*: If $s \leadsto u \rightarrow$ is a shortest path in G for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all time afterward

- *Path-relaxation property*: If $p = \{v_0, v_1, ..., v_k\}$ is a shortest path from $s = v_0$ to v_k, aand we relax the edges of p in order $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$.

- *Predecessor-subgraph property*: Once $v.d = \delta(s, v)$ for all $v \in V$), the predecessor subgraph is a shortest-paths tree rooted at s

24.1 The Bellman-Ford algorithm

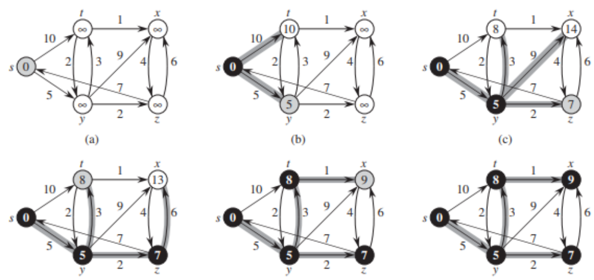


- The *Bellman-Ford algorithm* solves the single-source shortest-paths problem in general case in which edge weights may be negative.

- Given a weighted, directed graph with source s and weight function w, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortst paths and their weights

- The Bellman-Ford algorithm runs in time $O(VE)$

24.3 Dijkstra's algorithm



- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph for the case in which all edge weights are non-negative.

- Dijktra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u.

- The total running is $O((V + E)\lg V)$, which is $O(E \lg V)$ if all vertices are reachable from the source. We can achieve a running time of $O(V \lg V + E)$ by implementing the min-priority queue with a Fibonacci heap.

25 All-Pairs Shortest Paths

- In this chapter, we consider the problem of finding shortest paths between all pairs of vertices in a graph.

- We typically want the output in tabular form: the endtries in u's row and v's column should be the weight of a shortest path from u to v

- We assume that the vertices are numbered $1, 2, ..., |V|$, so that the input is an $n \times n$ matrix W representing the edge weights of an n-vertex directed graph. That is, $W = (w_{ij})$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{the weight of directed edge } (i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- We allow negative-weight edges, but we assume for the time being that the input graph contains no negative-weight cycles

- To solve the all-pair shortest-paths prolem on an input adjacency matrix, we need to compute not only the hosrtest-path weights but also a *predecessor matrix* $\prod = (\pi_{ij})$, where π_{ij} is `NIL` if either $i = j$ or there is not path from i to j, and otherweise π_{ij} is the predecessor of j on some shortest path from i. The subgraph induced by the ith row of the $\prod$ matrix should be a shortest-paths tree with root i. For each vertex $i \in V$, we define the *predecessor subgraph* of G for i as $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$, where

$$V_{\pi,i} = \{j \in V : \pi_{ij} \neq \texttt{NIL}\} \cup \{i\}$$

and

$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} - \{i\}\}$$