Lecture 14: Single-Source Shortest Paths

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

October 31, 2023

Overview

- Dijkstra's Algorithm
  - Single-Source Shortest Paths
  - Weighted Shortest Paths as a BFS
  - Dijkstra's Algorithm
- Acknowledgements

Single-Source Shortest Paths

- **Input:** *Weighted directed* graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, vertex $s \in V$

  Adjacency list.

- **Output:** a shortest path from s to t for any $t \in V$

Single-Source Shortest Paths

- **Input:** *Weighted directed* graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, vertex $s \in V$

  Adjacency list.

- **Output:** a shortest path from s to t for any $t \in V$
- Think of graph as the network of roads in a province
- Edge weights account for how long it takes to drive through that edge (i.e. traffic)

Single-Source Shortest Paths

- **Input:** *Weighted directed* graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, vertex $s \in V$

  Adjacency list.

- **Output:** a shortest path from s to t for any $t \in V$
- How should we output all these paths?

  Just as in unweighted case (BFS), could output a "directed tree" where we can read off the shortest paths.

  Succinct representation of the output.

Single-Source Shortest Paths

- **Input:** *Weighted directed* graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, vertex $s \in V$

  Adjacency list.

- **Output:** a shortest path from s to t for any $t \in V$
- How should we output all these paths?

  Just as in unweighted case (BFS), could output a "directed tree" where we can read off the shortest paths.

  Succinct representation of the output.

- Why can't I just use BFS?
  Need to account for the *weights* of edges - shortest path *not necessarily* given by *least number* of edges.

Single-Source Shortest Paths

- **Input:** *Weighted directed* graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$, vertex $s \in V$

  Adjacency list.

- **Output:** a shortest path from s to t for any $t \in V$
- How should we output all these paths?

  Just as in unweighted case (BFS), could output a "directed tree" where we can read off the shortest paths.

  Succinct representation of the output.

- Why can't I just use BFS?
  Need to account for the *weights* of edges - shortest path *not necessarily* given by *least number* of edges.
- Can we modify our graph so that it becomes unweighted?

  OUI et NON! Let's look at that now...

Shortest Paths as BFS

- Can imagine our graph as a set of water pipes

Length of pipes given by edge weights.
Assume $w : E \to \mathbb{N}$.

Shortest Paths as BFS

- Can imagine our graph as a set of water pipes

  Length of pipes given by edge weights.

  Assume $w : E \to \mathbb{N}$.

- When we pump water through s, spreads at uniform speed

Shortest Paths as BFS

- Can imagine our graph as a set of water pipes

  Length of pipes given by edge weights.
  Assume $w : E \to \mathbb{N}$.
- When we pump water through s, spreads at uniform speed
- At each time τ, if water reaches another vertex t, we record the distance $s \to t$ to be τ

Shortest Paths as BFS

- Can imagine our graph as a set of water pipes

  Length of pipes given by edge weights.
  Assume $w : E \to \mathbb{N}$.
- When we pump water through s, spreads at uniform speed
- At each time τ, if water reaches another vertex t, we record the distance $s \to t$ to be τ
- Making above intuition algorithmic:
  - Make unweighted graph $H(U, F)$ from G as follows
    1. For each $e := (u, v) \in E$, create path of length $w(e)$ from $u \to v$ in H
       Add new vertices and edges appropriately.
    2. Run BFS on H, starting from s
    3. Return "compressed tree" only having vertices from V

Shortest Paths as BFS

- Can imagine our graph as a set of water pipes

  Length of pipes given by edge weights.
  Assume $w : E \to \mathbb{N}$.

- When we pump water through s, spreads at uniform speed
- At each time τ, if water reaches another vertex t, we record the distance $s \to t$ to be τ
- Making above intuition algorithmic:
  - Make unweighted graph $H(U, F)$ from G as follows
    1. For each $e := (u, v) \in E$, create path of length $w(e)$ from $u \to v$ in H
       Add new vertices and edges appropriately.
    2. Run BFS on H, starting from s
    3. Return "compressed tree" only having vertices from V
- *Problem*: running time of above algorithm will be linear in H, but $O(|U| + |F|) = O\left(n + \sum_{e \in E} w(e)\right)$

Dijkstra's Algorithm

- **Idea:** simulate physical process above directly in G

Dijkstra's Algorithm

- **Idea:** simulate physical process above directly in G
- Algorithm
  1. $T = \emptyset$, $R = \{s\}$, $D[u] = \infty$ for $u \neq s$, $D[s] = 0$, $Q = V$
  2. While $Q \neq \emptyset$:
     - let $u \in V \setminus R$ be closest vertex to R, $e = (v, u)$ be edge such that

       $D[v] + w(e)$ minimizes distance $s \to u$
     - extract u from Q
     - $T \leftarrow T + e$, $R \leftarrow R \cup \{u\}$
  3. return T

Dijkstra's Algorithm

- **Idea:** simulate physical process above directly in G
- Algorithm
  1. $T = \emptyset$, $R = \{s\}$, $D[u] = \infty$ for $u \neq s$, $D[s] = 0$, $Q = V$
  2. While $Q \neq \emptyset$:
     - let $u \in V \setminus R$ be closest vertex to R, $e = (v, u)$ be edge such that
       $$D[v] + w(e) \text{ minimizes distance } s \to u$$
     - extract u from Q
     - $T \leftarrow T + e$, $R \leftarrow R \cup \{u\}$
  3. return T
- **Correctness:** follows from our BFS process

Dijkstra's Algorithm

- **Idea:** simulate physical process above directly in G
- Algorithm
  1. $T = \emptyset$, $R = \{s\}$, $D[u] = \infty$ for $u \neq s$, $D[s] = 0$, $Q = V$
  2. While $Q \neq \emptyset$:
     - let $u \in V \setminus R$ be closest vertex to R, $e = (v, u)$ be edge such that
       $$D[v] + w(e) \text{ minimizes distance } s \to u$$
     - extract u from Q
     - $T \leftarrow T + e$, $R \leftarrow R \cup \{u\}$
  3. return T
- **Correctness:** follows from our BFS process
- **Runtime:** need to find closest vertex & update distances fast.

How can we do that?

Via priority-queue (min heap).

Using such a priority-queue, runtime is given by $O((n+m)\log n)$.

Dijkstra - Full Implementation

- Full Algorithm
  1. Initialization:
     - $T = \emptyset$, (edges of our shortest-path)
     - $R = \{s\}$, (set of "reached vertices")
     - $p[u] = NULL$ for all $u \in V$ (parents)
     - $D[u] = \infty$ for all $u \in V \setminus \{s\}$, $D[s] = 0$ (distance to s)
     - $Q = V$ priority-queue (min heap w/ values given by D)
  2. While $Q \neq \emptyset$:
     - $u = \text{EXTRACT-MIN}(Q)$
     - For $v \in N_{out}(u)$:

       if $D[u] + w((u, v)) < D[v]$, then:

       set $D[v] = D[u] + w((u, v))$,

       $p[v] = u$,

       $\text{DECREASE-KEY}(Q, v)$
     - $T \leftarrow T + (p[u], u)$, $R \leftarrow R \cup \{u\}$
  3. return T

Runtime of Dijkstra's Algorithm

- Each vertex is enqueued once and dequeued once
- When vertex is dequeued, check outgoing edges and update distances (if needed)
- All queue operations implemented in $O(\log n)$ time by min-heap
- **Total runtime**:

$$O\left(\left(n + \sum_{u \in V} \deg_{out}(u)\right) \log n\right) = O((n+m)\log n)$$

Correctness of Dijkstra's Algorithm

- Similar analysis than the one we did for MST
- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$
    1. **Base case:** true when $R = \{s\}$

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \rightarrow u$
  1. **Base case:** true when $R = \{s\}$
  2. **Induction step:** assume invariant holds for R, and the algorithm adds u to R. Want to show that $D[u]$ is the shortest path distance $s \rightarrow u$.

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$
  1. **Base case:** true when $R = \{s\}$
  2. **Induction step:** assume invariant holds for R, and the algorithm adds u to R. Want to show that $D[u]$ is the shortest path distance $s \to u$.
  3. Need to prove that

$$D[u] = \min_{w \in R} D[w] + w((w, u))$$

is the shortest path distance $s \to u$.

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$
  1. **Base case:** true when $R = \{s\}$
  2. **Induction step:** assume invariant holds for R, and the algorithm adds u to R. Want to show that $D[u]$ is the shortest path distance $s \to u$.
  3. Need to prove that
  $$D[u] = \min_{w \in R} D[w] + w((w, u))$$
  is the shortest path distance $s \to u$.
  4. By choice of u, we have that $D[u] \leq D[v]$ for any $v \notin R$.

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$
  1. **Base case:** true when $R = \{s\}$
  2. **Induction step:** assume invariant holds for R, and the algorithm adds u to R. Want to show that $D[u]$ is the shortest path distance $s \to u$.
  3. Need to prove that
  $$D[u] = \min_{w \in R} D[w] + w((w, u))$$
  is the shortest path distance $s \to u$.
  4. By choice of u, we have that $D[u] \leq D[v]$ for any $v \notin R$.
  5. Since $D[u]$ is the distance of *some* $s \to u$ path, we know it is at least the shortest path distance.

Correctness of Dijkstra's Algorithm

- Proof of correctness by induction.
- **Claim:** for any $u \in R$, $D[u]$ is the shortest path distance from $s \to u$
  1. **Base case:** true when $R = \{s\}$
  2. **Induction step:** assume invariant holds for R, and the algorithm adds u to R. Want to show that $D[u]$ is the shortest path distance $s \to u$.
  3. Need to prove that
  $$D[u] = \min_{w \in R} D[w] + w((w, u))$$
  is the shortest path distance $s \to u$.
  4. By choice of u, we have that $D[u] \leq D[v]$ for any $v \notin R$.
  5. Since $D[u]$ is the distance of *some* $s \to u$ path, we know it is at least the shortest path distance.
  6. For the converse, consider any $s \to u$ path P. Since $s \in R$ and $u \notin R$, there is edge $(x, v) \in P$ where $x \in R$ and $v \notin R$. (R is a cut)
  $$\Rightarrow \ell(P) \geq D[x] + w((x, v))$$
  as $D[x]$ is shortest $s \to x$ distance, since $x \in R$

Correctness of Dijkstra's (continued)

- We have

$$\ell(P) \geq D[x] + w((x, v))$$

OBS: here we used non-negative edge weights.

Correctness of Dijkstra's (continued)

- We have

$$\ell(P) \geq D[x] + w((x, v))$$

  OBS: here we used non-negative edge weights.

- But we know that

$$D[x] + w(x, v) \geq D[v] \geq D[u]$$

  as u was chosen by the algorithm.

Correctness of Dijkstra's (continued)

- We have

$$\ell(P) \geq D[x] + w((x, v))$$

OBS: here we used non-negative edge weights.

- But we know that

$$D[x] + w(x, v) \geq D[v] \geq D[u]$$

as u was chosen by the algorithm.

- Thus, we have

$$\ell(P) \geq D[u]$$

Since the above holds for *any* $s \to u$ path, $D[u]$ is the shortest path distance.

Shortest Path Tree

- Just like we did for BFS and DFS, the set of edges $(p[u], u)$ form a (directed) tree.

Shortest Path Tree

- Just like we did for BFS and DFS, the set of edges $(p[u], u)$ form a (directed) tree.
- Since such edges (by construction) satisfy

$$D[u] = D[p[u]] + w(p[u], u)$$

and we just proved that $D[u]$ is the shortest $s \to u$ path distance, this tree stores all shortest path distances from s!

Acknowledgement

- Based on Prof. Lau's Lecture 9

  `https://cs.uwaterloo.ca/~lapchi/cs341/notes/L09.pdf`
- Also based on [Kleinberg Tardos 2006, Chapter 4]
- For refresher on min heaps, see [CLRS 2009, Chapter 6.5]

References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford. (2009)

Introduction to Algorithms, third edition.

MIT Press

Kleinberg, Jon and Tardos, Eva (2006)

Algorithm Design.

Addison Wesley