Introduction to Algorithms

Mong-Jen Kao (高孟駿)

Tuesday 10:10 – 12:00

Thursday 15:30 – 16:20

Graph Algorithms

- Graph problem pervades computer science, and

  Algorithms for graphs are fundamental to the field.

The Single-Source Shortest Path Problem

To compute the shortest distance / paths between a given source vertex and every other vertices in the graph.

Single-Source Shortest Path Problem

- Given a directed / undirected graph $G = (V, E)$ with an edge weight / length function $w : E \mapsto \mathbb{Q}$ and a source vertex $s \in V$, compute the distance (shortest path) from s to every vertex $v \in V$.

- In this lecture, we will examine two algorithms for this problem.
  - Bellman-Ford Algorithm
    – less efficient but works when edges have negative weights.
  - Dijkstra's Algorithm – more efficient but only works when edges have non-negative weights.

Optimal Substructure of Shortest Paths

- Let $G = (V, E)$ be a graph with edge weight function $w : E \mapsto \mathbb{Q}$, $u, v \in V$ be two vertices, and

$$P = u, q_1, q_2, \ldots, q_k, v$$

be a shortest path between u and v.

  - Then, for any $1 \leq i \leq k$, both

$$Q_i = u, q_1, \ldots, q_k \quad \text{and} \quad Q_i' = q_k, q_{k+1}, \ldots, q_k, v$$

are shortest-paths between u and q_k, and q_k and v, respectively.

Common Information / Operations

- For the Bellman-Ford algorithm and the Dijkstra's algorithm, we will maintain the following common information.
  - For each $v \in V$,
    - $d[v]$ - the distance between s and v.
    - $\pi[v]$ - the predecessor of v in a shortest path to s.
  - Initially, $d[s] = 0$ and $d[v] \leftarrow \infty$ for all $v \in V \setminus \{s\}$.

Common Information / Operations

- For the Bellman-Ford algorithm and the Dijkstra's algorithm, we will use the following common operation to update $d[\cdot]$.

Relax(u, v, w) - to update $d[v]$ based on $d[u]$ and $w(u, v)$.

- If $d[v] > d[u] + w(u, v)$ // Found a better path for v
  then set $d[v] \leftarrow d[u] + w(u, v)$ and $\pi[v] \leftarrow u$.

The Bellman-Ford Algorithm

Any shortest path, if exists, uses at most $|V|$ edges.

- For any $i \geq 0$ and any $v \in V$,
  let $d^{(i)}[v]$ denote the weight of the shortest path between s and v that contains at most i edges.

- The Bellman-Ford algorithm uses the following recurrence formula to compute $d^{(i)}[v]$ for any $0 \leq i \leq |V|$.

$$\text{For any } v \in V, \qquad d^{(i)}[v] = \min_{p \in V} \left\{ d^{(i-1)}[p] + w(p, v) \right\}.$$

The Bellman-Ford Algorithm

Any shortest path, if exists, uses at most $|V|$ edges.

Bellman-Ford(G, w, s) - to compute shortest s-v path for all $v \in V$.

A. Initialize $d[\cdot]$ and $\pi[\cdot]$.

B. For $i \leftarrow 1$ to $|V| - 1$, do the following
   - For each $e = (u, v) \in E$, call Relax(u, v, w).

C. For any $e = (u, v) \in E$,
   - If $d[v] > d[u] + w(u, v)$, then report “negative cycle exists”.

If $d[v]$ can be improved after $|V|$ steps, there must exist a negative cycle.

The Bellman-Ford Algorithm

- The Bellman-Ford algorithm runs in $O(|E| \cdot |V|)$ time.
  - It detects the existence of negative cycles in the graph.
  - In this case, shortest paths / distances are undefined.

The Dijkstra's Algorithm

- Dijkstra's algorithm computes $d[\cdot]$ with a BFS-like approach.
  - In each iteration,
    it picks an unprocessed vertex $v \in V$ with the smallest $d[\cdot]$ value.
  - It follows that $d[v] = \delta(s, v)$.
  - The algorithm uses $d[v]$ to update $d[u]$ for all $u \in V$.
- To implement the BFS process, the algorithm uses a priority queue Q to store the unprocessed vertices with priority $d[\cdot]$.

The Dijkstra's Algorithm

Dijkstra's-Algorithm(G, w, s) - to compute shortest s-v path for all $v \in V$.

A. Initialize $d[\cdot]$ and $\pi[\cdot]$.

B. Add V to Q.

C. While $Q \neq \emptyset$, do the following
   - Let $u \leftarrow$ Extract-Min(Q).
   - For each $v \in N(u)$, call Relax(u, v, w).

Decrease-Key$(v, d[v])$ is called if $d[v]$ is improved upon this call.

The Dijkstra's Algorithm

- The time complexity of Dijkstra's algorithm depends on the underlying priority queue used to implement this algorithm.
  - If both Extract-Min and Decrease-Key operations take $O(\log|V|)$ time, then the algorithm takes $O\big((|V|+|E|)\log|V|\big)$ time.
  - If ***Fibonacci heap*** or ***Quake heap*** is used,
    then Decrease-Key can be done in amortized $O(1)$ time,
    and the Dijkstra's algorithm runs in $O(|V|\log|V|+|E|)$ time.

The All-Pair Shortest Path Problem

To compute the shortest distance / paths between every pair of vertices in the graph.

The All-Pair Shortest Path Problem

- Given a directed / undirected graph $G = (V, E)$ with an edge weight / length function $w : E \mapsto Q^{\geq 0}$,
  for every pair $u, v \in V$ of vertices,
  compute the distance (shortest path) between u and v in G.

- In this lecture,
  we will examine the Floyd-Warshall algorithm for this problem.

A Simple Alternative Algorithm

- For any $i \geq 0$ and $u, v \in V$, define $d^{(i)}[u, v]$ to be the length of any shortest u-v path that contains at most i edges.
  - We have the following recurrence formula for $d^{(i)}[u, v]$

$$d^{(i)}[u, v] = \min_{p \in V} \{ d^{(i-1)}[u, p] + w(p, v) \} .$$

  - Hence, we can compute $d^{(n)}[u, v]$ for all $u, v \in V$ in $O(n^4)$ time, where $n = |V|$.

- For any $i \geq 0$ and $u, v \in V$, define $d^{(i)}[u, v]$ to be the length of any shortest u-v path that contains at most i edges.
  - We have the following recurrence formula for $d^{(i)}[u, v]$

$$d^{(i)}[u, v] = \min_{p \in V} \{ d^{(i-1)}[u, p] + w(p, v) \} .$$

  - The running time of this algorithm can be improved to $O(n^3 \log n)$ by applying the fast exponentiation technique.
    - When i is even, we have

$$d^{(i)}[u, v] = \min_{p \in V} \{ d^{(i/2)}[u, p] + d^{(i/2)}(p, v) \} .$$

The Flyod-Warshall Algorithm

- The Floyd-Warshall algorithm uses a cleaver observation on the optimal substructure of any shortest path.
  - Let $v_1, v_2, \cdots, v_n$ be an arbitrary labeling of the vertices.
  - Let P be a shortest u-v path that uses v_k but not v_ℓ for all $\ell > k$.
    - Then P consists of shortest u-v_k path and shortest v_k-v path that only use vertices from $v_1, v_2, \ldots, v_{k-1}$.
  - This gives a new DP recurrence formula.

The Flyod-Warshall Algorithm

- Let $v_1, v_2, \cdots, v_n$ be an arbitrary labeling of the vertices.
- For any $1 \leq k \leq n$ and any $u, v \in V$,
  let $d^{(k)}[u, v]$ denote the length of any shortest u-v path that only uses vertices from $\{v_1, v_2, \ldots, v_k\}$.
  - Then we have

$$d^{(k)}[u, v] = \min_{p \in V} \left\{ d^{(k-1)}[u, p] + d^{(k-1)}[p, v] \right\}.$$

  - This leads to an $O(n^3)$ algorithm for this problem.

The Flyod-Warshall Algorithm

Floyd-Warshall(G, w) - to compute shortest s-v path for all $u, v \in V$.

A. Initialize $d[\cdot]$.

B. For $k = 1,2, \dots, n$, do the following.

   - For $u = 1,2, \dots, n$, do the following.
     - For $v = 1,2, \dots, n$, do the following.
       - $d[u, v] \leftarrow \min\{\ d[u, v],\ d[u, k] + d[k, v]\ \}$.