Slido: #ADA2020

CSIE 2136 Algorithm Design and Analysis, Fall 2020

National Taiwan University 國立臺灣大學

Graph Algorithms - IV

Hsu-Chun Hsiao

3.5-week Agenda

- Graph basics
  - Graph terminology [B.4, B.5]
  - Real-world applications
  - Graph representations [Ch. 22.1]
- Graph traversal
  - Breadth-first search (BFS) [Ch. 22.2]
  - Depth-first search (DFS) [Ch. 22.3]
- DFS applications
  - Topological sort [Ch. 22.4]
  - Strongly-connected components [Ch. 22.5]
- Minimum spanning trees [Ch. 23]
  - Kruskal's algorithm
  - Prim's algorithm
- Single-source shortest paths [Ch. 24]
  - Dijkstra algorithm
  - Bellman-Ford algorithm
  - SSSP in DAG
- All-pairs shortest paths [Ch. 25]
  - Floyd-Warshall algorithm
  - Johnson’s algorithm

Today's Agenda

- All-pairs shortest paths [Ch. 25]
  - Floyd-Warshall algorithm
  - Johnson's algorithm

Variants of shortest-path problems

- Single-source shortest-path problem: Given a graph $G = (V, E)$ and a source vertex s in V, find the minimum cost paths from s to every vertex in V
- Single-destination shortest-path problem: Given a graph $G = (V, E)$ and a destination vertex t in V, find the minimum cost paths to t from every vertex in V
- Single-pair shortest-path problem: Find a shortest path from s to t for given s and t
- All-pair shortest path problem: Find a shortest path from s to t for every pair of s and t

All-pairs shortest paths Algorithms

- Repeated squaring of matrices
- Floyd-Warshall algorithm
- Johnson’s algorithm

Recap: DP view of Bellman-Ford algorithm

- Let $\ell_{sv}^{(k)}$ be the shortest path value from s to v using at most k edges
  - Subproblems: given s, $\ell_{sv}^{(k)}$ for all v, k
  - Optimal substructure: by Lemma 24.1
- Base cases: $\ell_{ss}^{(0)} = 0$; $\ell_{sv}^{(0)} = \infty$ when $s \neq v$
- The recurrence relation can be formulated as

$$\ell_{sv}^{(k)} = \min\left\{\ell_{sv}^{(k-1)}, \min_{u \in V}\left\{\ell_{su}^{(k-1)} + w_{uv}\right\}\right\}$$

$$= \min_{u \in V}\left\{\ell_{su}^{(k-1)} + w_{uv}\right\}$$

$$w_{ij} = \begin{cases} 0, & i = j \\ w(i,j), & i \neq j \text{ and } (i,j) \in E \\ \infty, & i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- Optimal values: $\ell_{sv}^{(|V|-1)}$ for all $v \in V$

Generalization to all-pairs shortest paths

- Let $\ell_{ij}^{(k)}$ be the shortest path value from i to j using at most k edges
  - Subproblems: $\ell_{ij}^{(k)}$ for all i, j, k
  - Optimal substructure: by Lemma 24.1
- Base cases: $\ell_{ii}^{(0)} = 0$; $\ell_{ij}^{(0)} = \infty$ when $i \neq j$
- The recurrence relation can be formulated as

$$\ell_{ij}^{(k)} = \min_{x \in \mathrm{V}} \left\{ \ell_{ix}^{(k-1)} + w_{xj} \right\}$$

- Optimal values: $\ell_{ij}^{(|V|-1)}$ for all $i, j \in V$

```
//Extend shortest paths by one hop
EXTEND-SHORTEST-PATHS(L, W)
    n = W.rows
    let L' = (ℓ_ij') be a new nxn matrix
    for i = 1 to n
        for j = 1 to n
                ℓ'_ij = min_{x∈V} {ℓ_ix + w_xj}
    return L'
```

```
for x = 1 to n
    ℓ'_ij = min{l'_ij, ℓ_ix + w_xj}
```

- $L^{(k)} = (\ell_{ij}^{(k)})$, the matrix of $\ell_{ij}^{(k)}$s
- $W = (w_{ij})$, the matrix of w_{ij}s
- $L^{(1)} = W$
- Running time of Extend-Shortest-Paths: $\Theta(V^3)$

Similarity to matrix multiplication

- Think of `EXTEND-SHORTEST-PATHS(L, W)` as "multiplying" the two matrices, $L \cdot W$
  - $+$ is replaced by min, $\cdot$ is replaced by $+$
  - 0 (the identity for $+$) is replaced by ∞ (the identity for min)
- Then we have
  - $L^{(1)} = W$
  - $L^{(k)} = L^{(k-1)} \cdot W = W^k$
- Shortest path wights are: $L^{(n-1)} = W^{n-1}$
- The overall running time: $\Theta(V^4)$

Can we do better than $\Theta(V^4)$?

- Observation: $L^{(k)} = L^{(n-1)}$ for all $k \geq n-1$

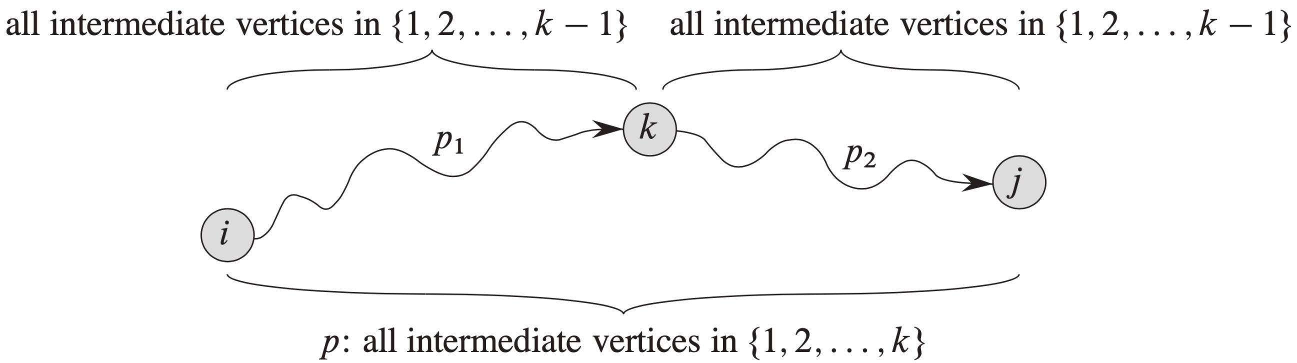
Q: Based on this observation, can we reduce it to $\Theta(V^3 \lg V)$?

Repeated squaring: keep squaring W for r times until $2^r > n-1$

Floyd-Warshall algorithm

Floyd-Warshall algorithm: intuition

- Consider a shortest path p_{ij} from i to j whose intermediate vertices are all in $\{1,2,\dots,k\}$
- Depending on whether k is an intermediate vertex of p_{ij}, there are two possible cases:
  - k is not an intermediate vertex of p_{ij} : all intermediate vertices are in $\{1,2,\dots,k-1\}$
  - k is an intermediate vertex of p_{ij}: p_{ij} can be decomposed into two sub-paths, $p_{ij} = i \rightsquigarrow k \rightsquigarrow j$, and the first (second) sub-path is a shortest path from i to k (k to j) with all intermediate vertices in $\{1,2,\dots,k-1\}$.



Floyd-Warshall algorithm: intuition

- Based on the observation, we can define a recurrence relation among shortest paths
- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to j whose intermediate vertices are all in $\{1,2,\ldots,k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & k \geq 1 \end{cases}$$

$$\boxed{w_{ij} = \begin{cases} 0, & i = j \\ w(i,j), & i \neq j \text{ and } (i,j) \in E \\ \infty, & i \neq j \text{ and } (i,j) \notin E \end{cases}}$$

- Claim: $d_{ij}^{(n)} = \delta(i,j)\ \forall i,j \in V$

Floyd-Warshall algorithm

```
FLOYD-WARSHALL(W)// W is the matrix of w_ij s
    n = W.rows
    D^(0) = W
    for k = 1 to n
        let D^(k) = (d_ij^(k)) be a new nxn matrix
        for i = 1 to n
            for j = 1 to n
                d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D^(n)
```

Q: What's the running time?

$\Theta(n^3)$

Q: How to construct the shortest paths?

Exercise 25.2-3, Exercise 25.2-7

Q: Can the following variant correctly compute all-pairs shortest path values?

```
FLOYD-WARSHALL-1(W)// W is the matrix of w_ij s
    n = W.rows
    D^(0) = W
    for k = 1 to n
        let D^(k) = (d_ij^(k)) be a new nxn matrix
    for i = 1 to n
        for j = 1 to n
            for k = 1 to n
                d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D^(n)
```

No

Johnson's algorithm for sparse graphs

Key idea: Reweighting

- Observation: If all edge weights are nonnegative, simply run Dijkstra's algorithm from each vertex
  - $O(V^2 \lg V + VE)$ using Fibonacci-heap min-priority queue

- Can we somehow reweight each edge such that all edge weights become nonnegative, while preserving the shortest paths?

Key idea: Reweighting

- Reweighing (using weight function $\hat{w}$ instead of w) should satisfy two important properties:
  1. Shortest-path preservation: $\forall\, u, v \in V$, a path p is a shortest path from u to v using weight function $w \Leftrightarrow \forall\, u, v \in V$, a path p is a shortest path from u to v using weight function $\hat{w}$
  2. Nonnegative weights: $\forall\, u, v \in V$, $\hat{w}(u, v)$ is nonnegative

Preserving shortest paths by reweighting

- Let $h: V \to \mathbb{R}$ be any function mapping vertices to real numbers
- Define a new weight function as

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

Q: Show that this reweighting preserve shortest paths
Q: Show that G has a negative-weight cycle using $w \Leftrightarrow G$ has a negative-weight cycle using $\hat{w}$

Producing nonnegative weights by reweighting

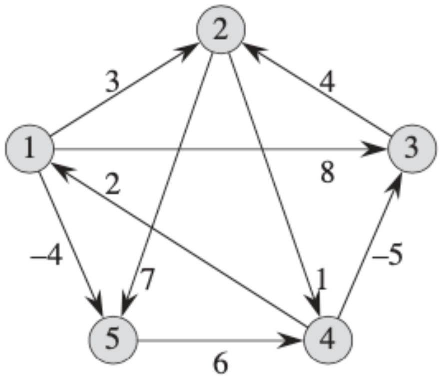
- <u>Goal:</u> Pick a function $h: V \to \mathbb{R}$ such that for all $u, v \in V$
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) \geq 0$$
- Johnson's algorithm takes advantage of the triangle inequality for shortest paths (Lemma 24.10)

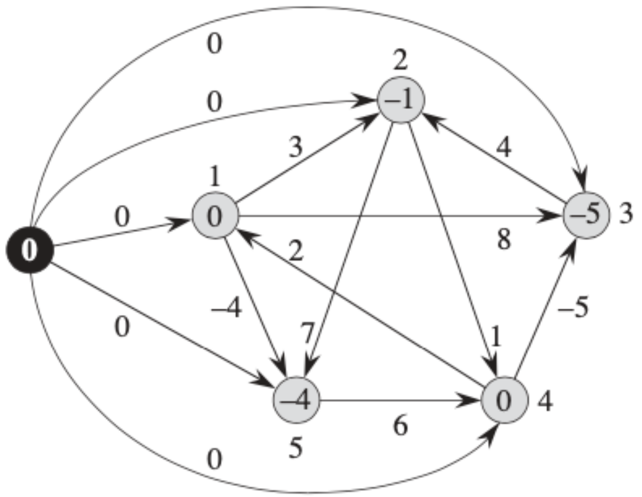
Triangle inequality (Lemma 24.10)

Given a source vertex s, for any edge $(u,v) \in E$, $\delta(s,v) \leq \delta(s,u) + w(u,v)$

Producing nonnegative weights by reweighting

- Pick a function $h: V \to \mathbb{R}$ such that for all $u, v \in V$
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v) \geq 0$$
  - Add an additional source vertex s
  - Add an edge from s to every vertex v in the original graph, $w(s,v) = 0$
  - Let $h(v) = \delta(s,v)$, which can be computed using Bellman-Ford algorithm





Johnson's Algorithm

JOHNSON(G, w)

compute G', where $G'.V = G.V \cup \{s\}$,
    $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
    $w(s, v) = 0$ for all $v \in G.V$
if BELLMAN-FORD(G', w, s) == FALSE
    print "the input graph contains a negative-weight cycle"
else for each vertex $v \in G'.V$
        set $h(v)$ to the value of $\delta(s, v)$
            computed by the Bellman-Ford algorithm
    for each edge $(u, v) \in G'.E$
        $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
    let $D = (d_{uv})$ be a new $n \times n$ matrix
    for each vertex $u \in G.V$
        run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v)$ for all $v \in G.V$
        for each vertex $v \in G.V$
            $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
    return D

1. Transform the graph and run Bellman-Ford algorithm from the added source vertex

2. Reweight edges

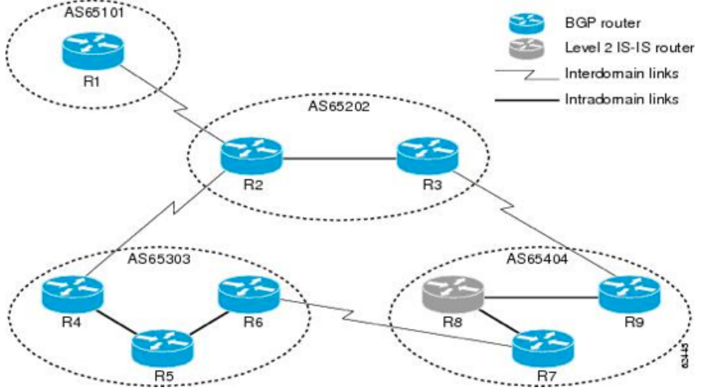
3. Run Dijkstra from each vertex and reconstruct the original distance

Time complexity

- Johnson's algorithm: $O(V^2 \lg V + VE)$
- C.f. Floyd-Warshall algorithm: $\Theta(V^3)$

Q: When will Johnson's algorithm run faster than Floyd-Warshall algorithm?
On sparse graphs, i.e., $|E| \sim |V|$

Application: Internet routing



Source: cisco.com

- Vertices = routers, ASes
- Edges = network links between routers
- Edge weight = delay, bandwidth, cost, hop count, etc.
- Link-state (commonly using Dijkstra's algorithm)
  - Nodes flood link state to whole network
  - E.g., Open Shortest Path First (OSPF)
- Distance-vector (commonly using Bellman-Ford's algorithm)
  - Nodes send vectors of destination and distance to neighbors
  - E.g., Routing Information Protocol (RIP)
- Path-vector (not necessarily shortest paths)
  - Nodes advertise the full paths to each destination
  - E.g., Border Gateway Routing Protocol (BGP)

Summary of graph algorithms

Graph search/traversal
Topological sort
Minimum spanning trees
Shortest paths
Negative cycle detection

BFS
DFS
Kruskal's
Prim's
Dijkstra's
Bellman-Ford