Algorithms II, CS 502

All-Pairs Shortest Paths

Ugur Dogrusoz

Computer Eng Dept, Bilkent Univ

All-pairs shortest paths

- Given a weighted, directed graph $G=(V,E)$ with a weight function $w:E\rightarrow R$ that maps edges to real-valued weights. We wish to find, for every pair of vertices $u,v \in V$, a shortest (least-weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges.

All-pairs shortest paths (SP)

- Running single-source SPs for each vertex as the source
  - No negative weights: Use Dijkstra's algorithm with Fibonacci heap, resulting in $O(V^2 \lg V + V E)$ run time.
  - Negative weights: Use Bellman-Ford, resulting in $O(V^2 E)$ (=$O(V^4)$ for dense graphs).

All-pairs shortest paths (SP)

- Unlike single-source SP algorithms, we use adjacency matrix representation for all-pairs SPs. Why?
  - Assume vertices are numbered 1,2,…,|V|, so that the input is an nxn matrix $W=(w_{ij})$ representing the edge weights of an n-vertex directed graph $G=(V,E)$.

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

All-pairs shortest paths (SP)

- A predecessor matrix $\Pi=(\pi_{ij})$ maintains actual paths

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
if i == j
    print i
elseif π_ij == NIL
    print "no path from" i "to" j "exists"
else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
    print j
```

Shortest paths & matrix multiplication

- A dynamic programming (DP) solution to all-pairs SPs problem is similar to matrix multiplication
  - Each major loop of the dynamic program will invoke an operation that is very similar to matrix multiplication
  - The algorithm will look like repeated matrix multiplication

Shortest paths

- Steps for developing a DP solution:
  - Characterize the structure of an optimal solution.
  - Recursively define the value of an optimal solution.
  - Compute the value of an optimal solution in a bottom-up fashion.
  - Construct an optimal solution from computed information

Shortest paths

- Characterize the structure of an optimal solution
  - Already proved all subpaths of a shortest path are shortest paths
- Recursively define the value of an optimal solution
  - $l_{ij}^{(m)}$: the minimum weight of any path from vertex i to vertex j that contains at most m edges

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j , \\ \infty & \text{if } i \neq j . \end{cases}$$

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\}\right) \\ &= \min_{1\le k\le n}\{l_{ik}^{(m-1)} + w_{kj}\} . \end{aligned}$$

Shortest paths

- Compute the value of an optimal solution in a bottom-up fashion

EXTEND-SHORTEST-PATHS(L, W)

$n = L.rows$
let $L' = (l'_{ij})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
  for $j = 1$ **to** n
    $l'_{ij} = \infty$
    for $k = 1$ **to** n
      $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
return L'

- $\Theta(n^3)$ due to three nested loops

Shortest paths & matrix multiplication

- Very similar to computing the product C=AxB for nxn matrices A and B.

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \}$$

$$\begin{aligned} l^{(m-1)} &\to a\,, \\ w &\to b\,, \\ l^{(m)} &\to c\,, \\ \min &\to +\,, \\ + &\to \cdot \end{aligned}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Shortest paths

- Compute the shortest-path weights by extending shortest paths edge by edge in $\Theta(n^4)$ time

$$
\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W &= W\,, \\
L^{(2)} &= L^{(1)} \cdot W &= W^2\,, \\
L^{(3)} &= L^{(2)} \cdot W &= W^3\,, \\
&\vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W &= W^{n-1}
\end{aligned}
$$

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
n = W.rows
L^(1) = W
for m = 2 to n - 1
    let L^(m) be a new n × n matrix
    L^(m) = EXTEND-SHORTEST-PATHS(L^(m-1), W)
return L^(n-1)
```

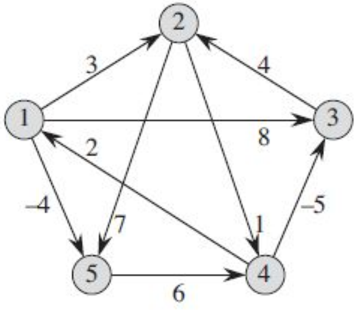
Shortest paths, improving running time

- $\Theta(n^3 \lg n)$ obtained by repeated squaring

$$
\begin{aligned}
L^{(1)} &= W\,, \\
L^{(2)} &= W^2 = W \cdot W\,, \\
L^{(4)} &= W^4 = W^2 \cdot W^2 \\
L^{(8)} &= W^8 = W^4 \cdot W^4\,, \\
&\vdots \\
L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}
\end{aligned}
$$

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
n = W.rows
L^(1) = W
m = 1
while m < n − 1
    let L^(2m) be a new n × n matrix
    L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
    m = 2m
return L^(m)
```

Shortest paths, improving running time



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd-Warshall algorithm

- Different DP formulation to solve all-pairs SPs on a directed graph $G=(V,E)$
  - Negative edges OK but no negative cycles
- The structure of a shortest path
  - For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1,2,\ldots,k\}$, and let p be a minimum-weight (simple) path from among them.

Floyd-Warshall algorithm

- Recursive solution
  - $d_{ij}^{(k)}$: weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1,2,…,k}

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

  - Final answer $D^{(n)} = (d_{ij}^{(n)})$
  - where $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$

Floyd-Warshall algorithm

- Computing SP weights bottom up

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
  let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

  - $\Theta(n^3)$ due to three nested loops

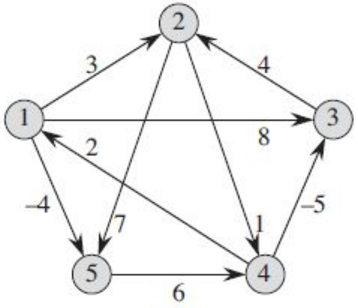
Floyd-Warshall algorithm

- Constructing a shortest path
  - Compute predecessor matrix π while the algorithm computes the matrices $D^{(k)}$.

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty , \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty . \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} , \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} . \end{cases}$$

Floyd-Warshall algorithm



Floyd-Warshall algorithm

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Floyd-Warshall algorithm

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Transitive closure

- Given a directed graph G=(V,E) with vertex set V={1,2,…,n}, we might wish to determine whether G contains a path from i to j for all vertex pairs (i,j) ϵ V. We define the transitive closure of G as the graph G*=(V,E*), where
  - E*={(i,j) : there is a path from vertex i to j in G}

Transitive closure

- Assign a weight of 1 to each edge and run Floyd-Warshall algorithm
  - There is a path from vertex i to j, then $d_{ij} < n$ (otherwise $d_{ij} = \infty$)
- Runs in $\Theta(n^3)$ time

Transitive closure

- Similar way (save time & space in practice)
  - Substitute OR for min and AND for + in Floyd-Warshall

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E , \\ 1 & \text{if } i = j \text{ or } (i,j) \in E , \end{cases}$$

and for $k \geq 1$,

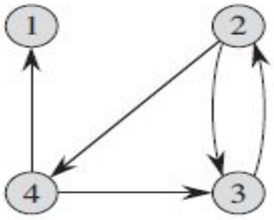
$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right) .$$

Transitive closure

TRANSITIVE-CLOSURE(G)

$n = |G.V|$
let $T^{(0)} = (t_{ij}^{(0)})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        if $i == j$ or $(i, j) \in G.E$
            $t_{ij}^{(0)} = 1$
        else $t_{ij}^{(0)} = 0$
for $k = 1$ **to** n
    let $T^{(k)} = (t_{ij}^{(k)})$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$
return $T^{(n)}$

Transitive closure



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson’s algorithm for sparse graphs

- Uses both Dijkstra’s algorithm and the Bellman-Ford algorithm as sub-routines
- Eliminates negative weights (assuming no negative cycles) by reweighting
- Runs Dijkstra’s algorithm once from each vertex

Johnson's algorithm for sparse graphs

- Assuming a Fibonacci heap min-priority queue implementation, running time is $O(V^2 \lg V + V E)$
- Asymptotically faster than either repeated squaring of matrices or the Floyd-Warshall algorithm for sparse graphs

Johnson's algorithm for sparse graphs

- New set of edge weights w' must satisfy:
  - For all pairs of vertices u,v ϵ V, a path p is a shortest path from u to v using weight function w if and only if p is also a shortest path from u to using weight function w'.
  - For all edges (u,v), the new weight w'(u,v) is nonnegative.
- We can preprocess G to determine the new weight function w' in O(V E) time.

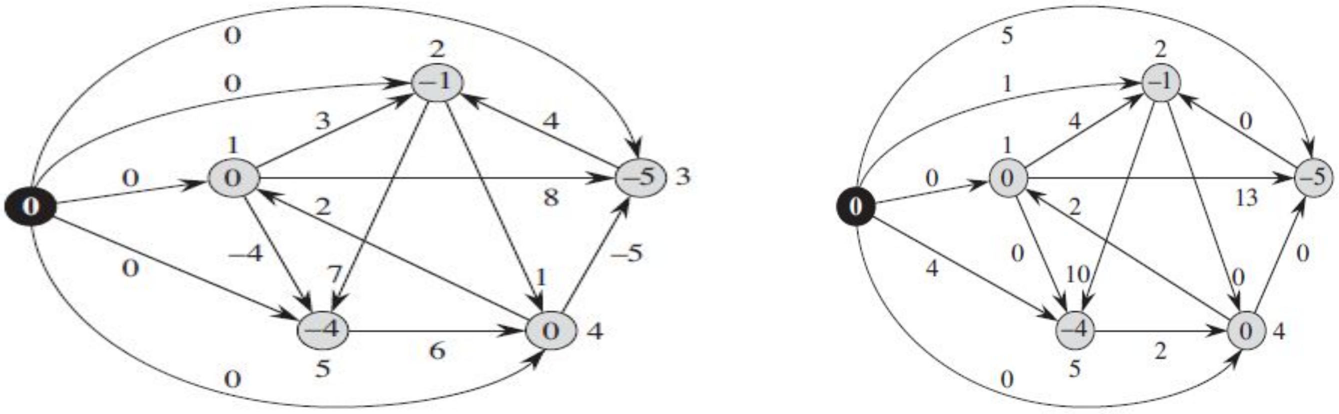
Johnson's algorithm for sparse graphs

- **Lemma 25.1 *(Reweighting does not change shortest paths)*** Given a weighted, directed graph $G=(V,E)$ with weight function $w:E\rightarrow R$, let $h:V\rightarrow R$ be any function mapping vertices to real numbers. For each edge $(u,v) \in E$, define $w'(u,v)=w(u,v)+h(u)-h(v)$.

- **Proof:** Let $p=\langle v_0,v_1,\ldots,v_k\rangle$ be any path from vertex v_0 to vertex v_k. Then p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function w'. That is, $w(p)=\delta(v_0,v_k)$ if and only if $w'(p)=\delta'(v_0,v_k)$.

  Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function w'.

Johnson’s algorithm for sparse graphs



Johnson's algorithm for sparse graphs