Chap 25 – All-Pairs Shortest Paths

25.1 Shortest paths and matrix multiplication

25.2 The Floyd-Warshall algorithm

25.3* Johnson's algorithm for sparse graphs

25.1 Shortest paths and matrix multiplication

- All-pairs shortest paths
  - A directed graph $G = (V, E)$ with weight function $w : E \to \mathbf{R}$.
  - Assume that we can number the vertices $1, 2, \ldots, n$.
  - So that G is given as an $n \times n$ adjacency matrix of weights: $W = (w_{ij}), 1 \le i, j \le n$, where

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

  - Goal

    Create an $n \times n$ matrix $D = (d_{ij})$ where $d_{ij} = \delta(i, j)$

25.1 Shortest paths and matrix multiplication

- All-pairs shortest paths
  - One way

    Run BELLMAN-FORD algorithm once for each vertex

    Time: $O(V)O(VE) = O(V^2E)$

    For dense graphs, it takes $O(V^4)$ time.
  - Another way: for non-negative weights

    Run DIJKSTRA'S algorithm once for each vertex
    - $O(VE \lg V)$ with binary heap – $O(V^3 \lg V)$ if dense
    - $O(VE + V^2 \lg V)$ with Fibonacci heap – $O(V^3)$ if dense
  - We'll see how to do in $O(V^3)$ in all cases, with no fancy data structure.

25.1 Shortest paths and matrix multiplication

- Dynamic programming solution
  - Optimal substructure

    Recall: Subpaths of shortest paths are shortest paths.
  - Let

    $l_{ij}^{(m)}$ = weight of shortest path $i \leadsto j$ that contains $\leq m$ edges
  - Boundary: $m = 0$

    There is a shortest path $i \leadsto j$ with ≤ 0 edge iff $i = j$

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

25.1 Shortest paths and matrix multiplication

- Dynamic programming solution
  - Recurrence: $m \geq 1$

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right)$$

(k is all possible predecessors of j)

$$= \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\} \quad \because w_{jj} = 0$$

  - Observation: When $m = 1$, $l_{ij}^{(1)} = w_{ij}$

$$l_{ij}^{(1)} = \min_{1\leq k\leq n}\left\{l_{ik}^{(0)} + w_{kj}\right\}$$

$$= l_{ii}^{(0)} + w_{ij} \quad \because l_{ii}^{(0)} \text{ is the only non-}\infty \text{ among } l_{ik}^{(0)}$$

$$= w_{ij}$$

25.1 Shortest paths and matrix multiplication

- Compute a solution bottom-up
  - Assume no negative-weight cycles
    $\Rightarrow$ all shortest paths are simple and contain $\le n-1$ edges
    $\Rightarrow \delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots$
  - Thus, let $L^{(m)} = \left(l_{ij}^{(m)}\right)$
    We may compute $L^{(1)}, L^{(2)}, \dots, L^{(n-1)}$, starting with
    $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$
  - Observation
    $L^{(m)} = L^{(n-1)}$ for all $m \ge n-1$

25.1 Shortest paths and matrix multiplication

- Compute a solution bottom-up
  - SLOW-APSP(W, n)

    $L^{(1)} = W$

    for $m = 2$ **to** $n - 1$

    let $L^{(m)}$ be a new $n \times n$ matrix

    $L^{(m)} =$ EXTEND$(L^{(m-1)}, W, n)$

    return $L^{(n-1)}$
  - Time
    - EXTEND $\Theta(n^3)$
    - SLOW-APSP $\Theta(n)\Theta(n^3) = \Theta(n^4)$

25.1 Shortest paths and matrix multiplication

- Compute a solution bottom-up
  - EXTEND(L, W, n)

    let L' 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'

25.1 Shortest paths and matrix multiplication

- EXTEND and matrix multiplication
  - EXTEND is like matrix multiplication.
  - // $L' = \text{EXTEND}(L, W)$

    for $i = 1$ **to** n

        for $j = 1$ **to** n

            $l'_{ij} = \infty$

            for $k = 1$ **to** n **do** $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
  - // $C = AB$

    for $i = 1$ **to** n

        for $j = 1$ **to** n

            $c_{ij} = 0$

            for $k = 1$ **to** n **do** $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

25.1 Shortest paths and matrix multiplication

- EXTEND and matrix multiplication
  - Thus, $L^{(m)}$ is analogous to A^m, where A is a square matrix.
  - We may compute $L^{(n-1)}$ in $O(\lg n)$ time by classic fast exponentiation algorithm:

$$a^n = \begin{cases} 1 & \text{if } n = 0 \\ a \cdot a^{n-1} & \text{if } n > 0 \text{ is odd} \\ (a^2)^{n/2} & \text{if } n > 0 \text{ is even} \end{cases}$$

  - However, since $L^{(m)} = L^{(n-1)}\ \forall m \geq n-1$, we can do better by repeated squaring:

$$L^{(1)} \quad L^{(2)} \quad L^{(4)} \quad L^{(8)} \ldots\ldots L^{(2^{\lceil \log(n-1) \rceil})}$$

$$L^{(n-1)} = L^{(2^{\log(n-1)})}$$

25.1 Shortest paths and matrix multiplication

- EXTEND and matrix multiplication
  - FASTER-APSP(W, n)

    $L^{(1)} = W$

    $m = 1$

    while $m < n - 1$

      let $L^{(2m)}$ be a new $n \times n$ matrix

      $L^{(2m)} = \text{EXTEND}(L^{(m)}, L^{(m)}, n)$

      $m = 2m$

    return $L^{(m)}$
  - Time $\Theta(\lg n)\Theta(n^3) = \Theta(n^3 \lg n)$
  - Space $\Theta(n^2 \lg n) \to \Theta(n^2)$, using 2 matrices (Ex. 25.1-8)

25.2 The Floyd-Warshall algorithm

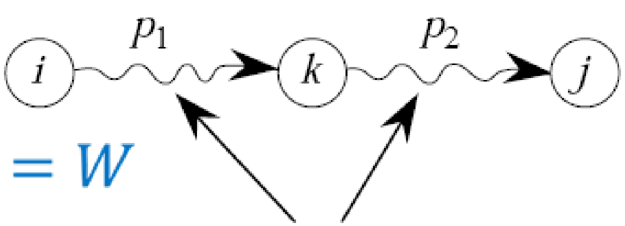
- Floyd-Warshall algorithm
  - Another dynamic programming solution
  - $d_{ij}^{(k)}$ = weight of shortest path $i \rightsquigarrow j$ with all intermediate vertices in $\{1,2,\ldots,k\}$

no intermediate vertex

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

k is an intermediate vertex

k isn't an intermediate vertex



  - Want $D^{(n)} = \left(d_{ij}^{(n)}\right)$, given $D^{(0)} = W$

25.2 The Floyd-Warshall algorithm

- Compute a solution bottom-up
  - FLOYD-WARSHALL(W, n)

    $D^{(0)} = W$

    for $k = 1$ **to** n

    let $D^{(k)}$ 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 $D^{(n)}$
  - Time $\Theta(n^3)$
  - Space $\Theta(n^3) \rightarrow \Theta(n^2)$, dropping superscripts (Ex. 25.2-4)

25.2 The Floyd-Warshall algorithm

- Transitive closure
  - Given a directed graph $G = (V, E)$

    Compute the transitive closure of G defined as

    $G^* = (V, E^*)$

    where $E^* = \{(i, j) : \text{there is a path } i \rightsquigarrow j \text{ in } G\}$
  - One way

    Assign weight of 1 to each edge in E

    Run FLOYD-WARSHALL

    - If $d_{ij}^{(n)} < n$, then there is a path $i \rightsquigarrow j$.
    - Otherwise, $d_{ij}^{(n)} = \infty$ and there is no path.

25.2 The Floyd-Warshall algorithm

- Transitive closure: A simpler way
  - $t_{ij}^{(k)} = 1$, if there is a path $i \rightsquigarrow j$ with all intermediate vertices in $\{1,2,\ldots,k\}$
    $= 0$, otherwise
  - $t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i = j \text{ or } (i,j) \in E \end{cases}$

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

    $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

25.2 The Floyd-Warshall algorithm

- Transitive closure: A simpler way
  - TRANSITIVE-CLOSURE(G)

    $n = |G.V|$

    for $i = 1$ **to** n

        for $j = 1$ **to** n

            if $i == j$ or $(i, j) \in G.E$ **then** $t_{ij}^{(0)} = 1$ **else** $t_{ij}^{(0)} = 0$

    for $k = 1$ **to** n

        let $T^{(k)}$ be a new $n \times n$ matrix

        for $i = 1$ **to** n

            for $j = 1$ **to** n

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