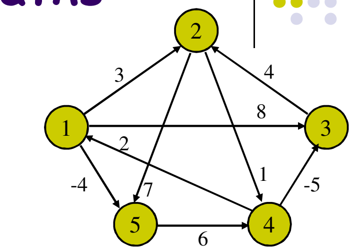
Algorithms

Dana Shapira

Lesson #8:

All-Pairs Shortest Paths

All-Pairs Shortest Paths

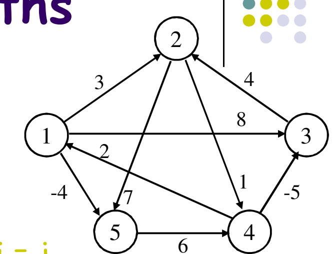


- ***Given:***
  - Directed graph $G = (V, E)$
  - Weight function $w : E \rightarrow \mathbf{R}$
- ***Compute:***
  - The shortest paths between all pairs of vertices in a graph
  - Representation of the result: an $n \times n$ matrix of shortest-path distances $\delta(u, v)$

All-Pairs Shortest Paths - Solutions

- Run **BELLMAN-FORD** once from each vertex:
  - $O(|V|^2|E|)$, which is $O(|V|^4)$ if the graph is dense
- If no negative-weight edges, could run **Dijkstra's** algorithm once from each vertex:
  - $O(|V|^3)$ if the graph is dense.
  - $O(|V||E|\lg|V|)$ if the graph is sparse.

All-Pairs Shortest Paths



- Assume the graph (G) is given as adjacency matrix of weights
  - $W = (w_{ij})$, $n \times n$ matrix, $|V| = n$
  - Vertices numbered 1 to n

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

- Output the result in an $n \times n$ matrix
  $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$
- Note: negative weights are allowed but for now no negative cycles.

Dynamic Programming

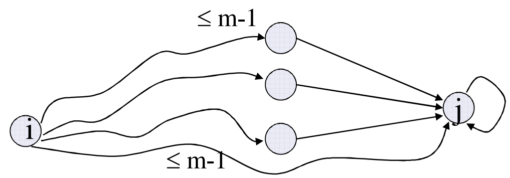
- $d_{ij}^{(m)}$ = weight from i to j with $\le m$ edges

  $d_{ij}^{(0)} = 0$ if $i = j$ and $d_{ij}^{(0)} = \infty$ if $i \neq j$

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

  $= \min_k\{d_{ik}^{(m-1)} + w_{kj}\}$ (Why?)

- Runtime = $O(n^4)$



Matrix Multiplication

- Similar: $C = A \cdot B$, two $n \times n$ matrices

  $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$ $O(n^3)$ operations
  - $c_{ij}= \min_k \{a_{ik} + b_{kj}\}$
  - $D^{(1)}=W$
  - $D^{(m)} = D^{(m-1)} \cdot W=W^m$

- Time is $O(n \cdot n^3) = O(n^4)$
- Repeated squaring: $W^{2n} = W^n \times W^n$

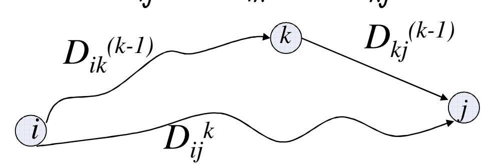
  Compute $W, W^2, W^4, \ldots, W^{2k}$, $k= \log n$, $\mathbf{O(n^3 \log n)}$

Floyd-Warshall Algorithm

- Vertices in G are given by $V = \{1, 2, \ldots, n\}$
- Consider a path $p = \langle v_1, v_2, \ldots, v_k\rangle$
  - An **intermediate** vertex of p is any vertex in the set $\{v_2, v_3, \ldots, v_{k-1}\}$

Floyd-Warshall Algorithm

- $d_{ij}^{(k)}$ = length of shortest path from i to j with intermediate vertices from $\{1, 2, \ldots, k\}$:
- $\delta(i, j)= d_{ij}^{(n)}$
- Dynamic Programming: recurrence
  - $d_{ij}^{(0)} = w_{ij}$
  - $d_{ij}^{(k)} = \min \{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$

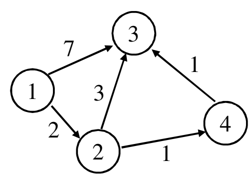


intermediate nodes in *{1, 2, ..., k}*

Example

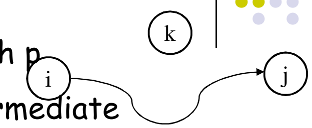
$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from {1, 2, …, k}

- $d_{13}^{(0)}$ =
- $d_{13}^{(1)}$ =
- $d_{13}^{(2)}$ =
- $d_{13}^{(3)}$ =
- $d_{13}^{(4)}$ =

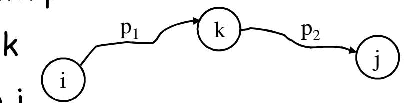


The Structure of a Shortest Path

- k is not an intermediate vertex of path p
  - Shortest path from i to j with intermediate vertices from {1, 2, …, k} is a shortest path from i to j with intermediate vertices from {1, 2, …, k - 1}



- k is an intermediate vertex of path p
  - p_1 is a shortest path from i to k
  - p_2 is a shortest path from k to j
  - k is not intermediary vertex of p_1, p_2
  - p_1 and p_2 are shortest paths from i to k with vertices from {1,2,…,k-1}



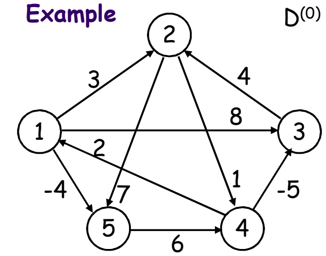
FLOYD-WARSHALL(W)

```
D(0) ← W
for k ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            do dij(k) ← min (dij(k-1), dik(k-1) + dkj(k-1))
return D(n)
```

- Running time: $\Theta(n^3)$

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

Example



$D^{(0)} = W$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^{(1)}$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(2)}$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(3)}$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

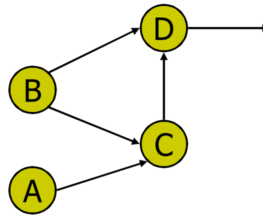
$D^{(4)}$

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

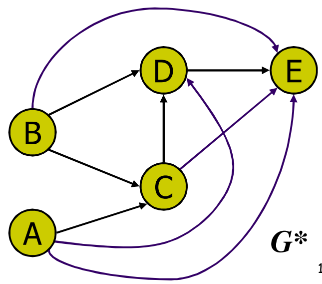
Question

- A *Transitive Closure* of a graph
  G =(V,E) is the graph
  G=(V,E*)* where
  *E**={(*i,j*)/ there exist a path from *i* to *j* in *G*.}

  Given a graph *G*, use Floyd-Warshall's algorithm to compute the closure of *G*



G



*G**

Summary- Single Source Shortest Path

Type of Graph	Algorithm	Time
No weights	BFS	O(|V|+|E|)
Non negative weights	Dijkstra	O(|V|²)/O(|E|log|V|)
Weighted graph	Belman-Ford	O(|V||E|)

Summary- All-Pairs Shortest Paths

Type of Graph	Algorithm	Time
No weights	|V| times BFS	O(|V|(|E|+|V|))
Non negative weights	|V| times Dijkstra	O(|V|³)/O(|V||E|log|V|)
Weighted graph	|V| times Belman-Ford	O(|V|²|E|)
Weighted graphs No negative cycles	Floyd-Warshall	O(|V|³)

Sparse graphs

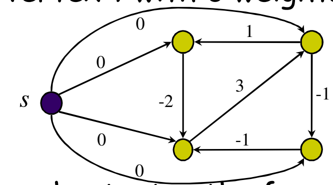
- *What if the graph is sparse?*
  - If no negative edges – run repeated Dijkstra's
  - If negative edges – let us somehow change the weights of all edges (to *w´*) and then run repeated Dijkstra's
- Requirements for ***reweighting***:
  - *Non-negativity*: for each (*u,v*), *w´*(*u,v*) ≥ 0
  - *Shortest-path equivalence*: for all pairs of vertices *u* and *v*, a path *p* is a shortest path from *u* to *v* using weights *w if and only if p* is a shortest path from *u* to *v* using weights *w´*.

Reweighting theorem

- *Rweighting does not change shortest paths*
  - Let h: V $\rightarrow$ **R** be any function
  - For each $(u,v) \in E$, define
    $w'(u,v) = w(u,v) + h(u) - h(v)$.
  - Let $p = (v_1, v_2, \ldots, v_k)$ be any path from v_1 to v_k
  - *Then*: $w(p) = \delta(v_1, v_k) \Leftrightarrow w'(p) = \delta'(v_1, v_k)$

Choosing reweighting function

- *How to choose function h?*
- The idea of Johnson:
  - 1. Augment the graph by adding vertex s and edges (s,v) for each vertex v with 0 weights.



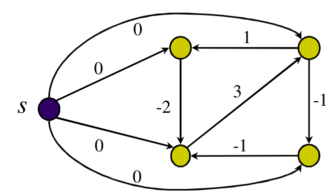
  - 2. Compute the shortest paths from s in the augmented graph (using Belman-Ford).
  - 3. Make $h(v) = \delta(s, v)$

Johnson's algorithm

- *Why does it work?*
  - By definition of the shortest path:
    for all edges (u,v), $h(v) \leq h(u) + w(u,v)$
  - Thus, $w(u,v) + h(u) - h(v) \geq 0$
- Johnson's algorithm:
  1. Construct augmented graph
  2. Run Bellman-Ford (possibly report a negative cycle), to find $h(v) = \delta(s, v)$ for each vertex v
  3. Reweight all edges:
     - $w'(u,v) \leftarrow w(u,v) + h(u) - h(v)$.
  4. For each vertex u:
     - Run Dijkstra's from u, to find $\delta'(u, v)$ for each v
     - For each vertex v: $D[u][v] \leftarrow \delta'(u, v) + h(v) - h(u)$

Example, Analysis

- *Do the reweighting on this example:*



- *What is the running time of Johnson's?*