What about weighted graphs?

$G = (V, E, w)$ weighted graph $V = \{1,2,\ldots,n\}$, $w : E \to \mathbf{R}$.

We assume that there is no negative-cost cycle, but negative-cost edges may be present.

Weight matrix W defined by

$$W[i,j] = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$$

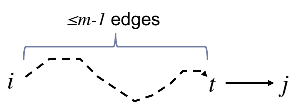
First method: matrix product

Let $d^{(m)}(i,j)$ be the minimum value of a path from i to j provided that this path contains **at most** m edges

We have to compute $d(i,j)=d^{(n-1)}(i,j)$

Idea : proceed by induction on m

$$d^{(0)}(i,j) = \begin{cases} 0 & \text{if } i=j \\ \infty & \text{otherwise} \end{cases}$$



For $m \geq 1$,

$$d^{(m)}(i,j) = \min\left(d^{(m-1)}(i,j), \min\{d^{(m-1)}(i,t)+W[t,j] \mid 1 \leq t \leq n\}\right) =$$

$$\min\{d^{(m-1)}(i,t)+ W[t,j] \mid 1 \leq t \leq n\}$$

In terms of matrices, we have $D^{(m)}=D^{(m-1)} \cdot W$ where

min plays the role of addition and

$+$ plays the role of multiplication

Computing $D=W^n$ by repeated squaring leads to the time complexity $O(n^3 \cdot \log n)$

Algorithm based on intermediate nodes: Floyd(-Warshall) algorithm

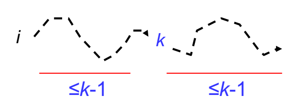
Notation

$D_k = (D_k[i,j] \mid 1 \leq i, j \leq n)$ with

$D_k[i,j] = \min\{ w(c) \mid c$ path from i to j with all intermediate nodes $\leq k \}$

$D_0 = W$

D_n = distance matrix of G $= D$



Lemma For all $k \geq 1$,

$D_k[i,j] = \min\{ D_{k-1}[i,j] , D_{k-1}[i,k] + D_{k-1}[k,j] \}$

Computation

of D_k from D_{k-1} in time $O(n^2)$

of $D = D_n$ in time $O(n^3)$

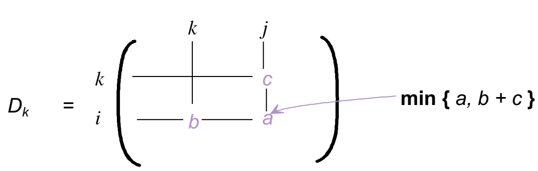
$D_0 = W$

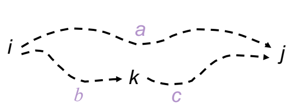
for $k = 1$ **to** n **do**

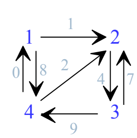
  for $i = 1$ **to** n **do**

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

      $D_k[i,j] =$ **min {** $D_{k-1}[i,j]$, $D_{k-1}[i,k] + D_{k-1}[k,j]$ **}**







$$D_0 = W = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 2 & \infty & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & \infty & 0 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & 13 \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix}$$

Representing shortest paths

Explicitely storing shortest paths from i to j, $1 \le i, j \le n$

n^2 paths of maximum length n-1: space $O(n^3)$

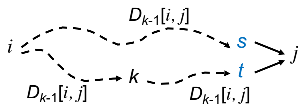
Predecessor matrix: space $\Theta(n^2)$

$\pi_k = (\pi_k[i, j] \mid 1 \le i, j \le n)$ where

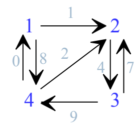
$\pi_k[i, j]$ = predecessor of j on some shortest path from i to j with all intermediate nodes $\le k$

Recurrence

$$\pi_0[i, j] = \begin{cases} i & \text{if } i \neq j \text{ and } (i, j) \in A \\ nil & \text{else} \end{cases}$$



$$\pi_k[i, j] = \begin{cases} \pi_{k-1}[i, j] & \text{if } D_{k-1}[i, j] \le D_{k-1}[i, k] + D_{k-1}[k, j] \\ \pi_{k-1}[k, j] & \text{else} \end{cases}$$



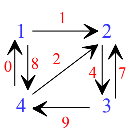
$$D_0 = W = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 2 & \infty & 0 \end{pmatrix} \qquad P_0 = \begin{pmatrix} - & 1 & - & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 4 & - & - \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & \infty & 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} - & 1 & - & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 1 & - & - \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} - & 1 & 2 & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & 13 \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_3 = \begin{pmatrix} - & 1 & 2 & 1 \\ - & - & 2 & 3 \\ - & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_4 = \begin{pmatrix} - & 1 & 2 & 1 \\ 4 & - & 2 & 3 \\ 4 & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$



$$D_4 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_4 = \begin{pmatrix} - & 1 & 2 & 1 \\ 4 & - & 2 & 3 \\ 4 & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

Example of a path

distance from 2 to 1 = $D_4[2,1]$ = 13

$P_4[2,1] = 4$; $P_4[2,4] = 3$; $P_4[2,3] = 2$;

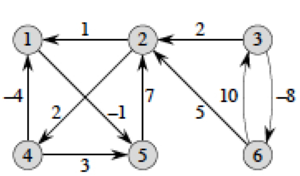
$$2 \xrightarrow{4} 3 \xrightarrow{9} 4 \xrightarrow{0} 1$$

Remarks

- For sparse graphs represented by adjacency lists there exists Johnson's algorithm that works in time $O(|V|^2 \cdot \log |V| + |V| \cdot |E|)$.
- Warshall's and Floyd-Warshall algorithms are examples of the ***dynamic programming*** technique that we will study later in more details

Exercise

- Run Floyd-Warshall on the following graph:



Shortest paths: summary

Unweighted single-source shortest paths

Breadth-first search $O(|V|+|E|)$

Weighted single-source shortest paths

depending on assumptions:

Dijkstra's algorithm $O(|V|^2)$

or $O(|V| + |E| \cdot \log |V|)$

Bellman-Ford algorithm $O(|E| \cdot |V|)$

All-pairs shortest paths

Floyd-Warshall algorithm $O(|V|^3)$