T&R Team of Algorithm Design
College of Computer Science and Engineering, CQU

Algorithm Analysis & Design

Introduction to Algorithm

Chapter 25:

All-Pairs Shortest Paths

Outlines

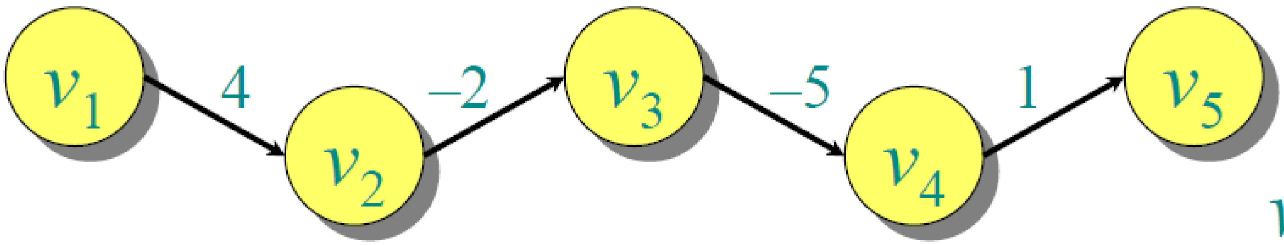
- **All-pairs shortest paths**
- **First solution: using Dijkstra's algorithm**
- **Second solution: dynamical programming**
  1. **matrix manipulation**
  2. **Floyd-Warshall algorithm**

Paths in Graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

Example:



$w(p) = -2$

Shortest Paths

A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

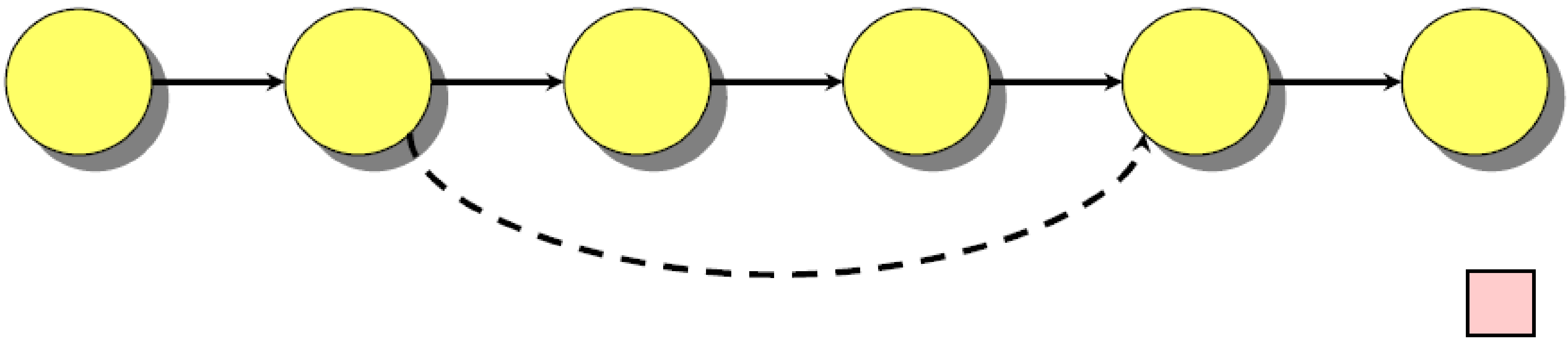
$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

Note: $\delta(u, v) = \infty$ if no path from u to v exists.

Optimal Sub-Structure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:

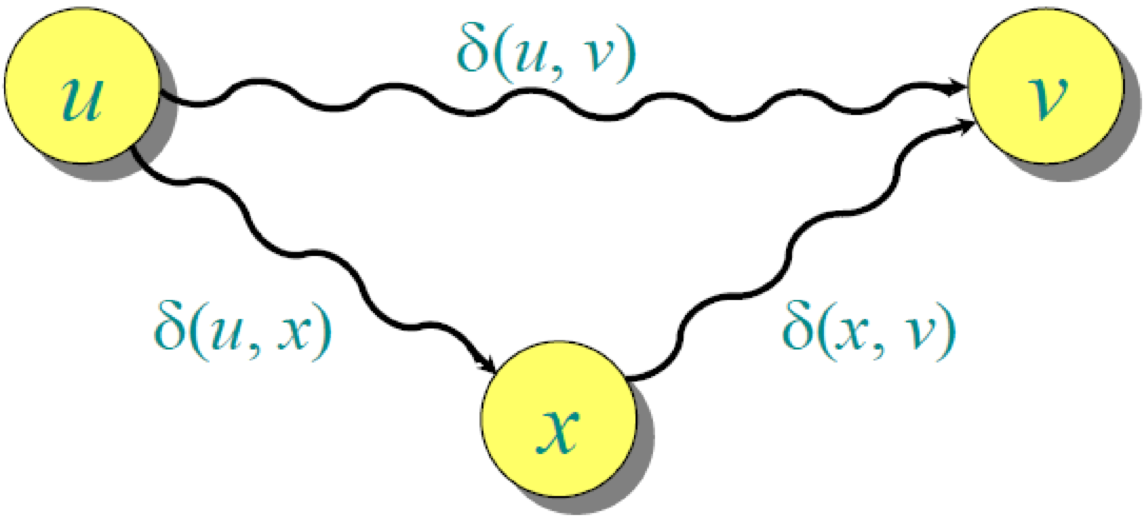


Triangle Inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \le \delta(u, x) + \delta(x, v).$$

Proof.

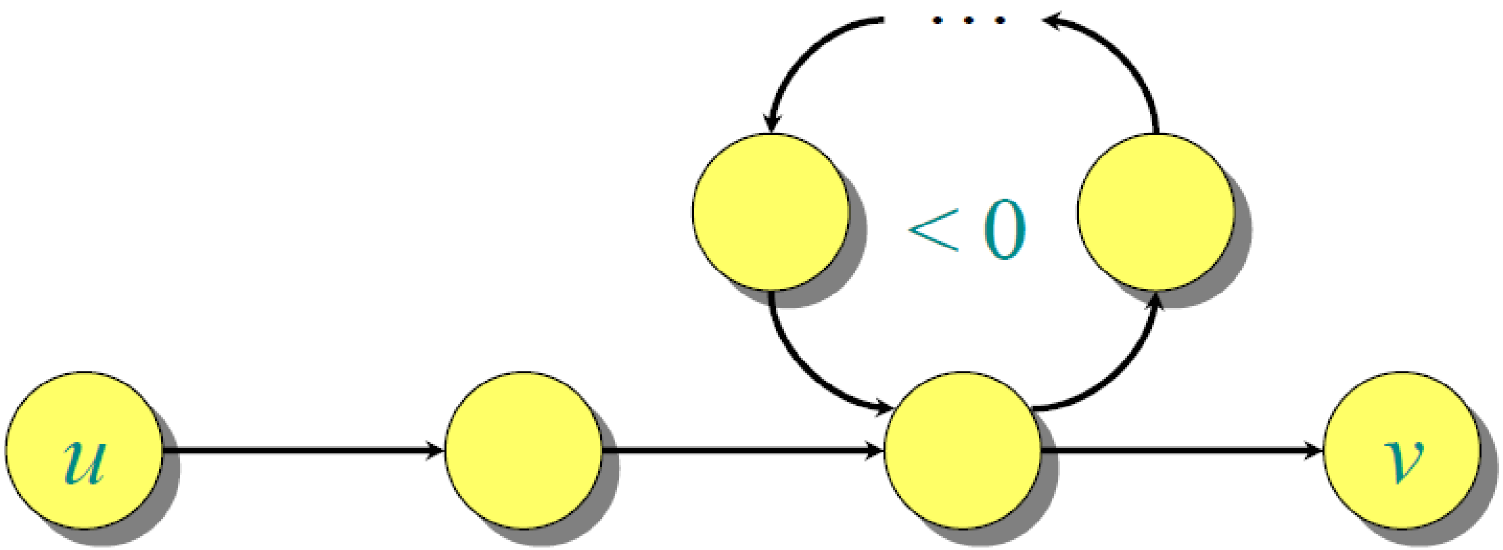


$\square$

Well-Defineness of SP

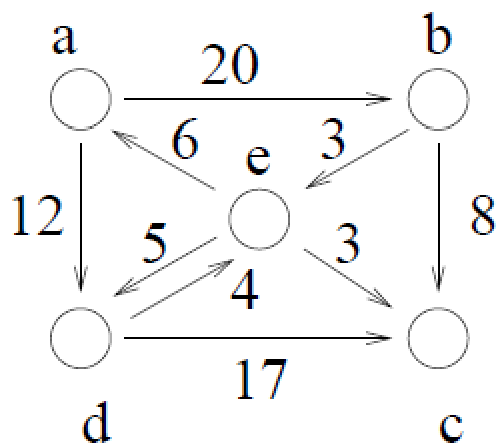
If a graph G contains a negative-weight cycle, then some shortest paths may not exist.

Example:

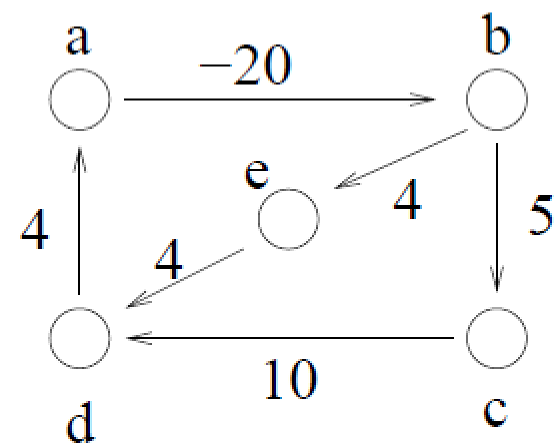


All-Pairs Shortest Paths

Given a weighted digraph $G = (V, E)$ with weight function $w : E \to \mathbf{R}$, (R is the set of real numbers), determine the length of the shortest path (i.e., distance) between all pairs of vertices in G.



without negative cost cycle



with negative cost cycle

Dijkstra’s Algorithm

Solution 1: Dijkstra's Algorithm

If there are no negative cost edges apply Dijkstra's algorithm to each vertex (as the source) of the digraph.

- Recall that D's algorithm runs in $\Theta((n+e)\log n)$. This gives a

$$\Theta(n(n+e)\log n) = \Theta(n^2\log n + ne\log n)$$

time algorithm, where $n = |V|$ and $e = |E|$.

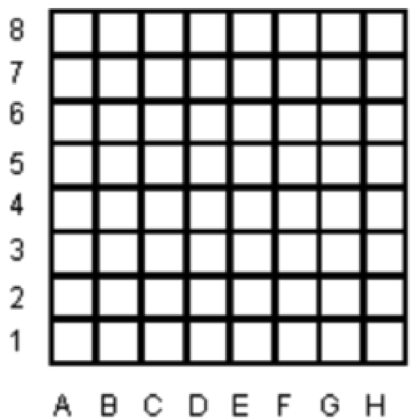
Solution 1: Dijkstra’s Algorithm

< 返回

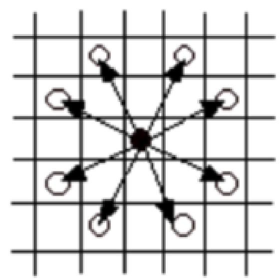
7-43 3.3.3 Camelot (190 分)

很久以前,亚瑟王和他的骑士习惯每年元旦去庆祝他们的友谊.在回忆中,我们把这些是看作是一个有一人玩的棋盘游戏. 有一个国王和若干个骑士被放置在一个由许多方格组成的棋盘上,没有两个骑士在同一个方格内.

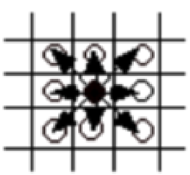
- 这个例子是标准的 8*8 棋盘



一个骑士可以从黑点移动到白点（如下图）， 但前提是他不掉出棋盘之外.



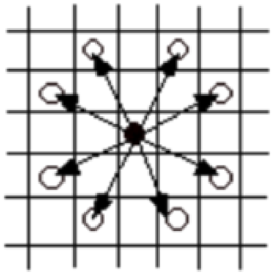
国王可以移动到任何一个相邻的方格,从黑点移动到白点（如下图）,但前提是他不掉出棋盘之外.



玩家的任务就是把所有的棋子移动到同一个方格里——用最小的步数. 为了完成这个任务,他必须按照上面所说的规则去移动棋子. 玩家必须选择一个骑士跟国王一起行动,其他的单独骑士则自己一直走到集中点. 骑士和国王一起走的时候,只算一个人走的步数.

Solution 1: Dijkstra's Algorithm

一个骑士可以从黑点移动到白点（如下图），但前提是他不掉出棋盘之外.



$Dist[x][y][s]$表示某个骑士走到棋盘位置(x,y)的最小步数，
$s\in\{0,1\}$, 0表示自己单独到达，1表示带着king一起到达。

$$Dist[x][y][0]=\min\begin{cases}\min(\{\,Dist[x+a][y+b][0]\mid a,b\in\{1,-1,2,-2\}\,\})+1\\ \\ Dist[x][y][0]\end{cases}$$

$$Dist[x][y][1]=min\begin{cases}\min(\{\,Dist[x+a][y+b][1]\mid a,b\in\{1,-1,2,-2\}\,\})+1\\ \\ Dist[x][y][0]+kingDist[x][y]\end{cases}$$

用DP计算，但bottom-up顺序不明确，直接迭代困难！

用Dijkstra Algorithm追踪bottom-up顺序

Dynamical Programming

To make DP work:

(1) How do we decompose the all-pairs shortest paths problem into subproblems?

(2) How do we express the optimal solution of a subproblem in terms of optimal solutions to some subsubproblems?

(3) How do we use the recursive relation from (2) to compute the optimal solution in a bottom-up fashion?

(4) How do we construct all the shortest paths?

Matrix multiplication

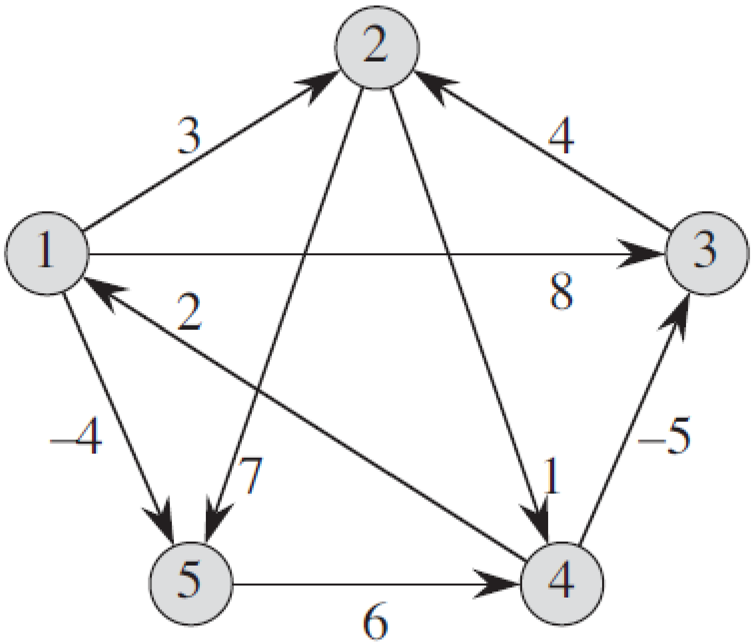
To simplify the notation, we assume that $V = \{1, 2, \ldots, n\}$.

Assume that the graph is represented by an $n \times n$ matrix with the weights of the edges:

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

Output Format: an $n \times n$ matrix $D = [d_{ij}]$ where d_{ij} is the length of the shortest path from vertex i to j.

Example



Without negative circle

Input
$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Output
$$\begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

How to decompose the problem

- Subproblems with smaller sizes should be easier to solve.

- An optimal solution to a subproblem should be expressed in terms of the optimal solutions to subproblems with smaller sizes.

These are guidelines ONLY.

Step 1: Decompose in a Natural Way

- Define $d_{ij}^{(m)}$ to be the length of the shortest path from i to j that contains at most m edges.
  Let $D^{(m)}$ be the $n \times n$ matrix $[d_{ij}^{(m)}]$.

- $d_{ij}^{(n-1)}$ is the true distance from i to j (see next page for a proof this conclusion).

- Subproblems: compute $D^{(m)}$ for $m = 1, \cdots, n-1$.

Question: Which $D^{(m)}$ is easiest to compute?

$d_{ij}^{(n-1)}$ = True Distance from i to j

Proof: We prove that any shortest path P from i to j contains at most $n-1$ edges.

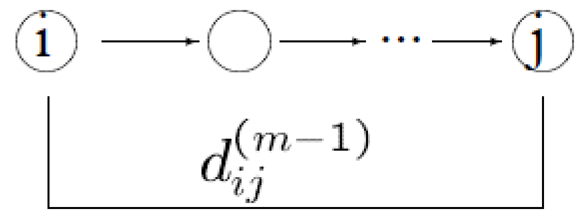
First note that since all cycles have positive weight, a shortest path can have no cycles (if there were a cycle, we could remove it and lower the length of the path).

A path without cycles can have length at most $n-1$ (since a longer path must contain some vertex twice, that is, contain a cycle).

Step 2： Recursive Formula

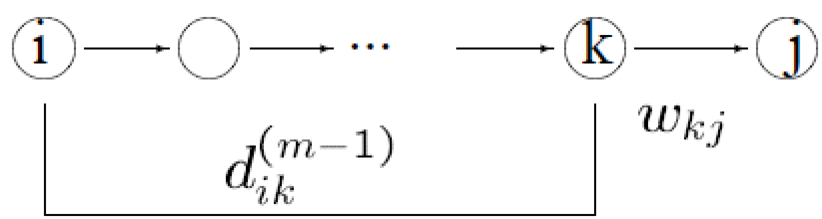
Consider a shortest path from i to j of length $d_{ij}^{(m)}$.

Case 1: It has at most $m-1$ edges.



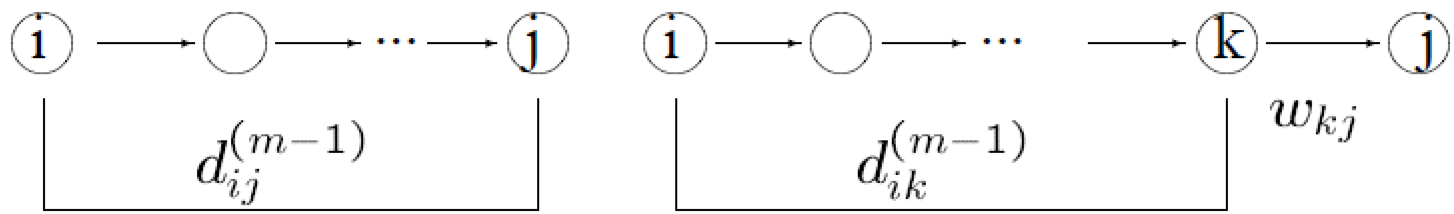
Then $d_{ij}^{(m)} = d_{ij}^{(m-1)} = d_{ij}^{(m-1)} + w_{jj}$.

Case 2: It has m edges. Let k be the vertex before j on a shortest path.



Then $d_{ij}^{(m)} = d_{ik}^{(m-1)} + w_{kj}$.

Step 2: Recursive Formula



Case 1: at most $m-1$ edges shortest path

Case 2: exactly m edges shortest path

Combining the two cases,

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

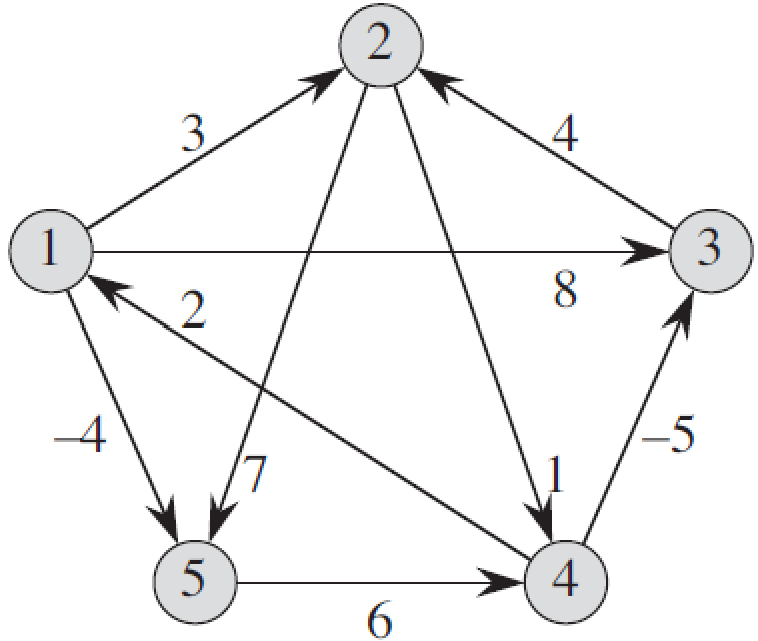
Step 3: Bottom-Up Computation

- Bottom: $D^{(1)} = [w_{ij}]$, the weight matrix.

- Compute $D^{(m)}$ from $D^{(m-1)}$, for $m = 2, ..., n-1$, using

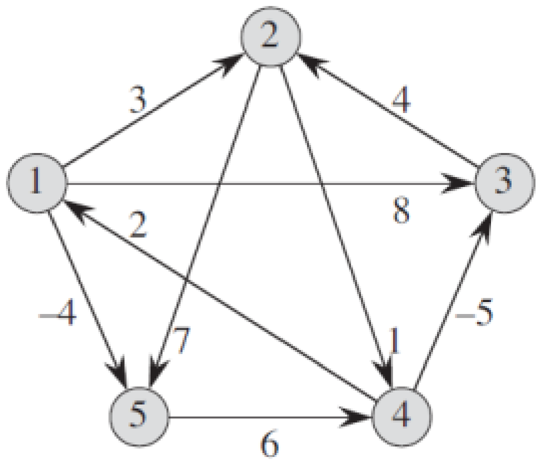
$$d_{ij}^{(m)} = \min_{1 \le k \le n} \left\{ d_{ik}^{(m-1)} + w_{kj} \right\}.$$

Example



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

Example



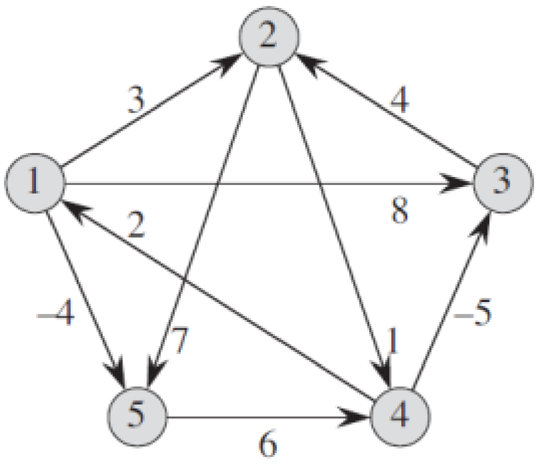
$$D^{(1)} \qquad\qquad\qquad\qquad D^{(1)}$$

$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \text{X} \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(2)} = \min_{1 \le k \le 5} \{ d_{ik}^{(1)} + d_{kj}^{(1)} \}$$

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

Example



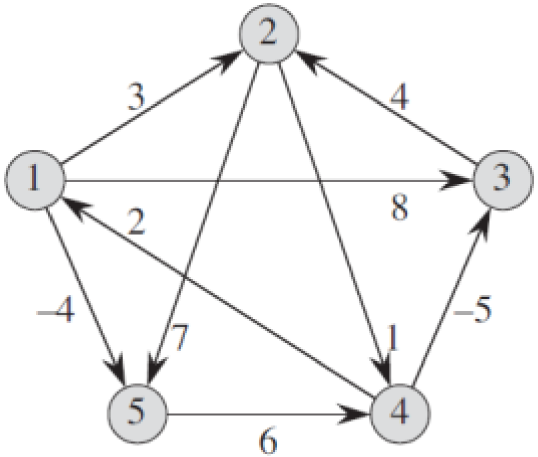
$$D^{(2)} \qquad\qquad D^{(1)}$$

$$\begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(3)} = \min_{1 \le k \le 5} \{ d_{ik}^{(2)} + d_{kj}^{(1)} \}$$

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

Example



$$D^{(3)} \qquad\qquad D^{(1)}$$

$$\begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \text{ x } \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(4)} = \min_{1 \le k \le 5} \{ d_{ik}^{(3)} + d_{kj}^{(1)} \}$$

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

The shortest distances between any pair of vertices

Algorithm

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

  for $i = 1$ to n

    for $j = 1$ to n

    $\{$

      $min = \infty;$

      for $k = 1$ to n

      $\{$

        $new = d_{ik}^{(m-1)} + w_{kj};$

        if $(new < min)$ $min = new;$

      $\}$

      $d_{ij}^{(m)} = min;$

    $\}$

Comments

- Algorithm uses $\Theta(n^3)$ space; how can this be reduced down to $\Theta(n^2)$?

- How can we extract the actual shortest paths from the solution?

- Running time $O(n^4)$, much worse than the solution using Dijkstra's algorithm. Can we improve this?

Improvement: Repeated Squaring

$$D^{(n-1)} = D^i, \text{ for all } i \geq n.$$

In particular, this implies that $D^{\left(2^{\lceil \log_2 n \rceil}\right)} = D^{(n-1)}$.

We can calculate $D^{\left(2^{\lceil \log_2 n \rceil}\right)}$ using "repeated squaring" to find

$$D^{(2)}, D^{(4)}, D^{(8)}, \ldots, D^{\left(2^{\lceil \log_2 n \rceil}\right)}$$

Improvement: Repeated Squaring

- Bottom: $D^{(1)} = [w_{ij}]$, the weight matrix.

- For $s \geq 1$ compute $D^{(2s)}$ using

$$d_{ij}^{(2s)} = \min_{1 \leq k \leq n} \left\{ d_{ik}^{(s)} + d_{kj}^{(s)} \right\}.$$

Given this relation we can calculate $D^{(2^i)}$ from $D^{(2^{i-1})}$ in $O(n^3)$ time. We can therefore calculate all of

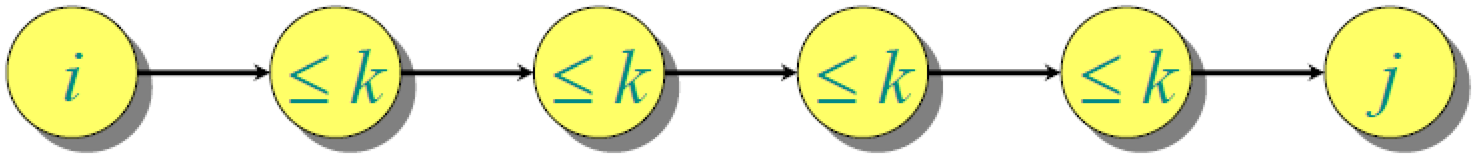
$$D^{(2)}, D^{(4)}, D^{(8)}, \ldots, D^{\left(2^{\lceil \log_2 n \rceil}\right)} = D^{(n)}$$

in $O(n^3 \log n)$ time, improving our running time.

Floyd-Warshell Algorithm

Definition: The vertices $v_2, v_3, \ldots, v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, \ldots, v_{l-1}, v_l \rangle$.

- Let $d_{ij}^{(k)}$ be the length of the shortest path from i to j such that *all* intermediate vertices on the path (if any) are in set $\{1, 2, \ldots, k\}$.



$d_{ij}^{(0)}$ is set to be w_{ij}, i.e., no intermediate vertex.

Let $D^{(k)}$ be the $n \times n$ matrix $[d_{ij}^{(k)}]$.

Floyd-Warshall Algorithm

Definition: The vertices $v_2, v_3, ..., v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, ..., v_{l-1}, v_l \rangle$.

- Claim: $d_{ij}^{(n)}$ is the distance from i to j. So our aim is to compute $D^{(n)}$.

- Subproblems: compute $D^{(k)}$ for $k = 0, 1, \cdots, n$.

Similar to a 0-1 knapsack problem!

The Structure of Shortest Paths

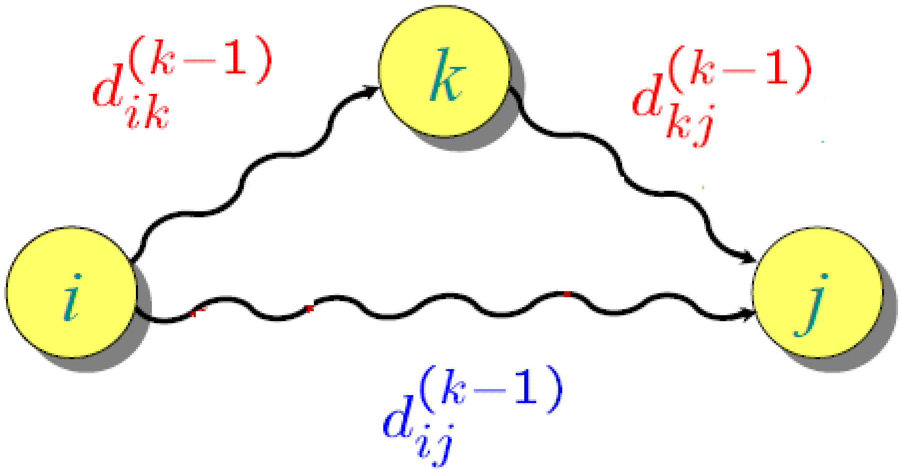
Observation 1: A shortest path does not contain the same vertex twice.

Non-negative circle!

Step 2: The Structure of Shortest Paths

Observation 2: For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:

k is a vertex on the path.



k is not a vertex on the path,

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

Step 3： Bottom-Up Computation

- Bottom: $D^{(0)} = [w_{ij}]$, the weight matrix.

- Compute $D^{(k)}$ from $D^{(k-1)}$ using

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

  for $k = 1, \ldots, n$.

Step 3： Bottom-Up Computation

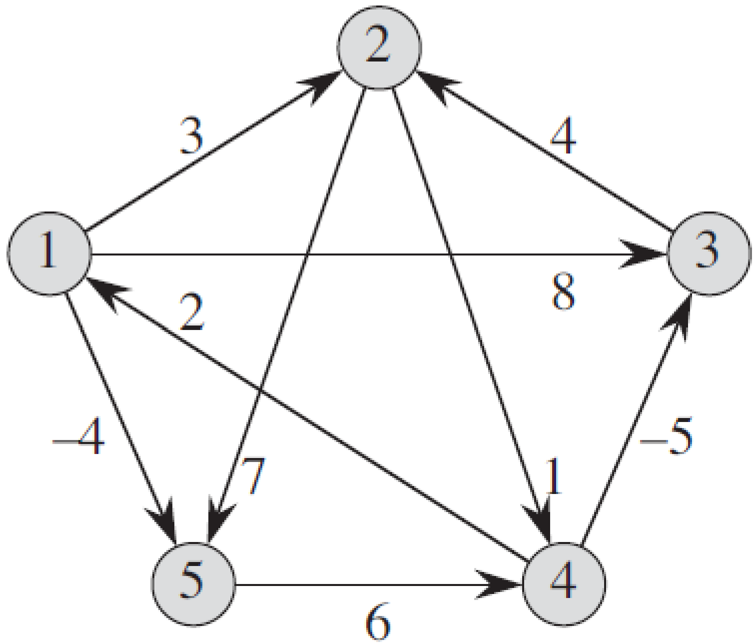
- Bottom: $D^{(0)} = [w_{ij}]$, the weight matrix.

- Compute $D^{(k)}$ from $D^{(k-1)}$ using

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

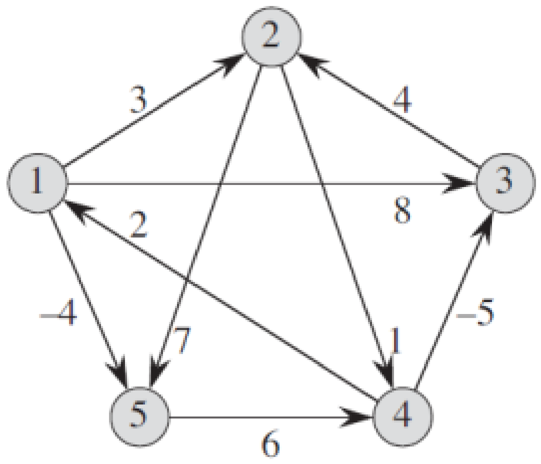
for $k = 1, \ldots, n$.

Example



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

Example



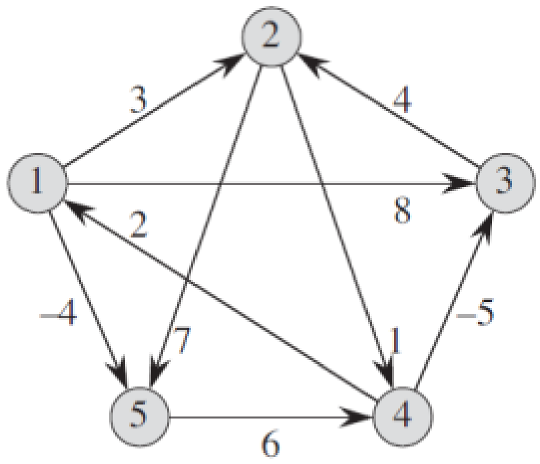
$$D^{(0)}$$

$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(1)} = min\{d_{ij}^{(0)},\ d_{i1}^{(0)} + d_{1j}^{(0)}\}$$

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

Example



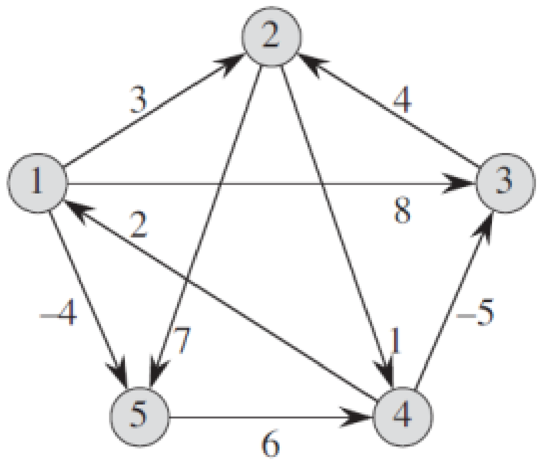
$$D^{(1)}$$

$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(2)} = min\{d_{ij}^{(1)},\ d_{i2}^{(1)} + d_{2j}^{(1)}\}$$

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

Example



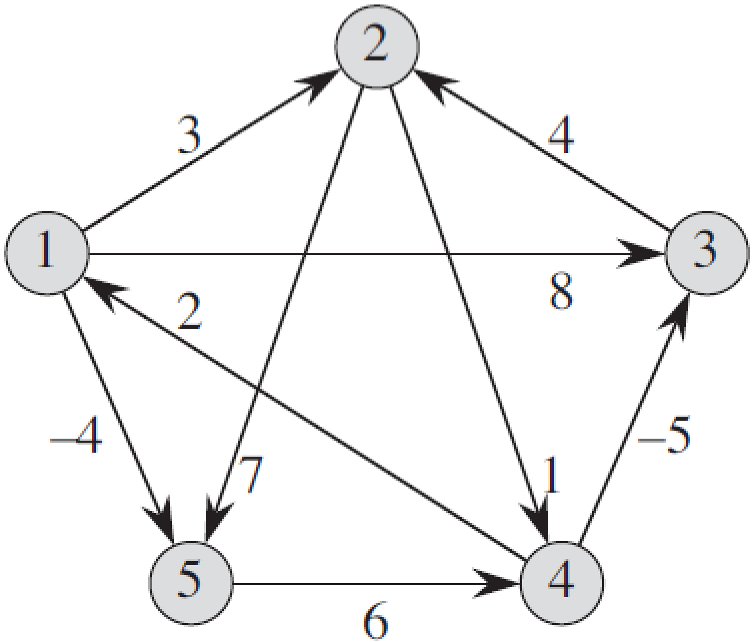
$$D^{(2)}$$

$$\begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{ij}^{(3)} = min\{d_{ij}^{(2)},\ d_{i3}^{(2)} + d_{3j}^{(2)}\}$$

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

Example



$$d_{ij}^{(5)} = min\{d_{ij}^{(4)},\ d_{i5}^{(4)} + d_{5j}^{(4)}\}$$

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

The shortest distances between any pair of vertices

Algorithm

Floyd-Warshall(w, n)

{ for $i = 1$ to n do   initialize

    for $j = 1$ to n do

    { $d[i, j] = w[i, j]$;

      $pred[i, j] = nil$;

    }

  for $k = 1$ to n do   dynamic programming

    for $i = 1$ to n do

      for $j = 1$ to n do

        if $(d[i, k] + d[k, j] < d[i, j])$

          $\{d[i, j] = d[i, k] + d[k, j]$;

          $pred[i, j] = k;\}$

  return $d[1..n, 1..n]$;

}

Comments

- The algorithm's running time is clearly $\Theta(n^3)$.

- The predecessor pointer `pred[i, j]` can be used to extract the final path (see later).

- Problem: the algorithm uses $\Theta(n^3)$ space.
  It is possible to reduce this down to $\Theta(n^2)$ space by keeping only one matrix instead of n.

Extracting The Shortest Paths

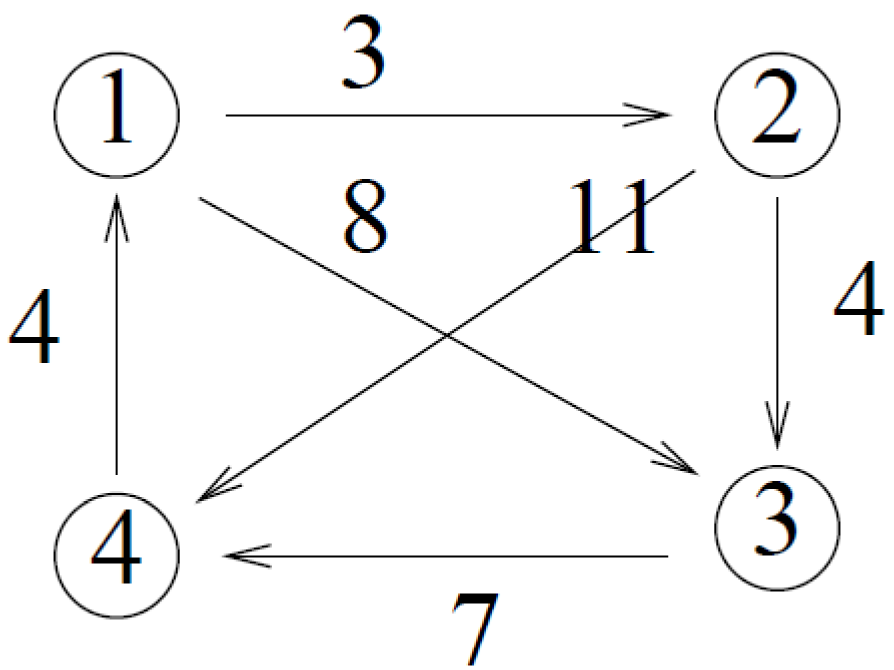
To find the shortest path from i to j, we consult $pred[i, j]$. If it is nil, then the shortest path is just the edge (i, j). Otherwise, we recursively compute the shortest path from i to $pred[i, j]$ and the shortest path from $pred[i, j]$ to j.

Exercises

- 25.1-1
- 24.1-7
- 25.2-1
- 25.2-4

Short Test in Class

Give $D^{(1)}$, $D^{(2)}$, $D^{(3)}$ with matrix multiplication algorithm, or $D^{(0)}$, $D^{(1)}$, $D^{(2)}$ by Floyd-Warshell algorithm.



算法分析课程组
重庆大学计算机学院

End of Section.