アルゴリズムの設計と解析

教授：　黄　潤和　（W4022）
rhuang@hosei.ac.jp

SA：　広野　史明　（A4/A8）
fumiaki.hirono.5k@stu.hosei.ac.jp

Contents (L14 – All-pairs Shortest Path)

- All-pairs shortest paths
- Floyd-Warshall algorithm
- Matrix-multiplication algorithm

MIT video lecture

http://videolectures.net/mit6046jf05_demaine_lec19/

All-pairs shortest paths

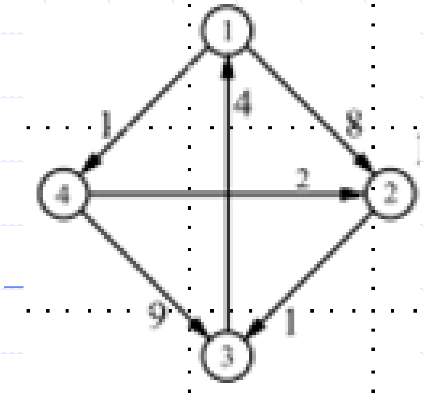
Run Dijkstra’s algorithm for each vertex

Dijkstra Algorithm is a single source shortest path (for nonnegative edge weights)

If we would like to make all-pairs shortest paths

Very simple idea is to use Dijkstra’s algorithm
(Q: how many times to run Dijkstra algorithm?)

Take an example,



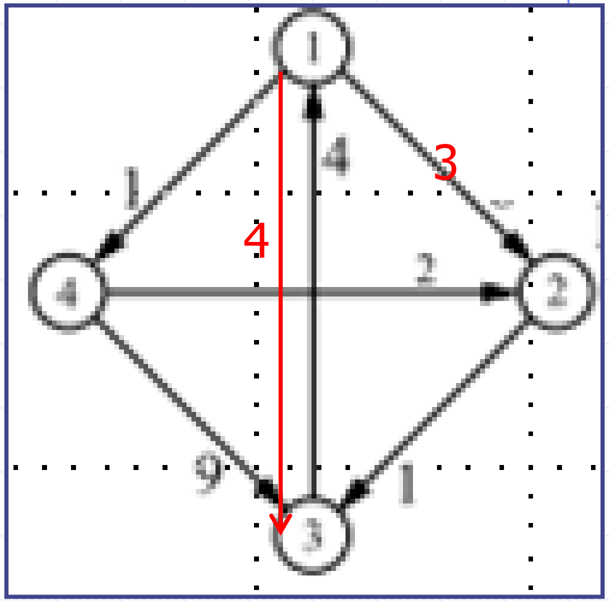
$$\begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

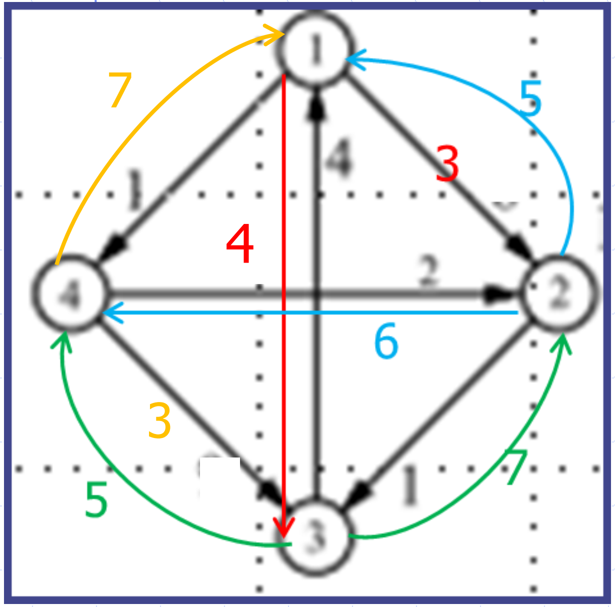
Dijkstra, from node 1
1->2, 1->4->2, (8→ 3)
1->3, 1→4->2->3, (-, 4)
1->4, (1)

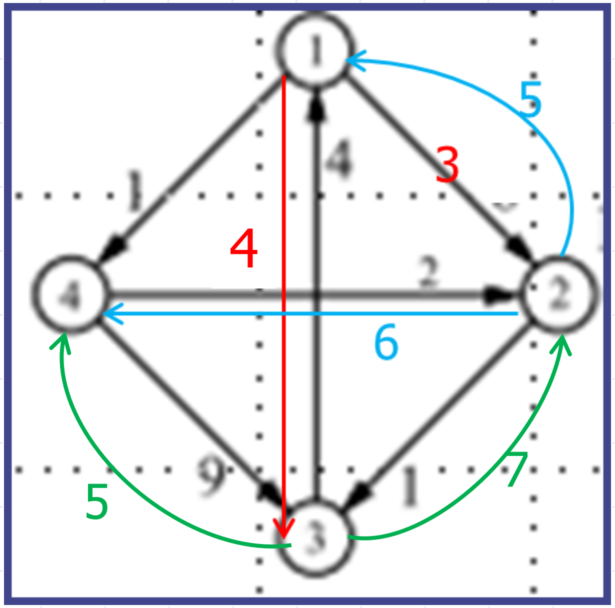
Dijkstra, from node 2
2->1, 2->3->1, (-, 5)
2->3, (1)
2->4, 2->1->4, (-, 6)

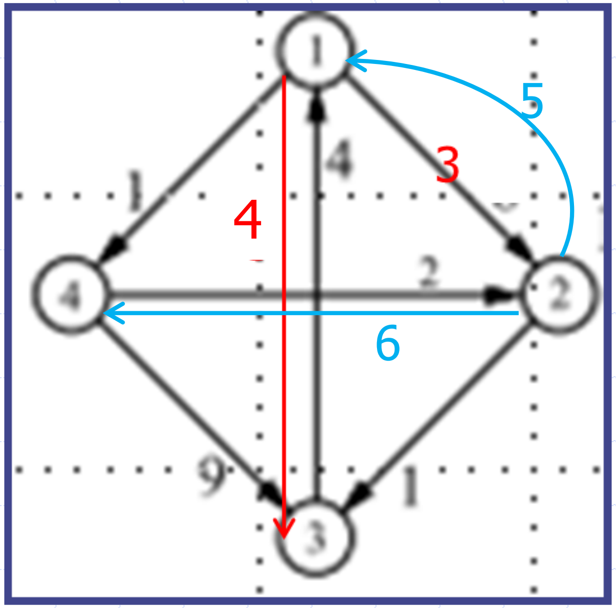
Dijkstra, from node 3
3->1, (4)
3->2, 3->1->2, (-, 7)
3->4, 3->1->4, (-, 5)

Dijkstra, from node 4
4->1, 4->2->1 (-, 7)
4->2, (2)
4->3, 4->2->3, (9→3)









Dijkstra algorithm’s time complexity

$$\text{Time} = \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$$

$$Q \qquad T_{\text{EXTRACT-MIN}} \quad T_{\text{DECREASE-KEY}}$$

Time Complexity:

-Dijkstra's original algorithm does not use a min-priority queue and runs in time $O(|V|^2)$.

-The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(|E| + |V| \log |V|)$

All-pairs shortest paths

using Dijkstra algorithm

Very simple idea is to use Dijkstra's algorithm (run Dijkstra algorithm for $|V|$ times)

Time complexity

If not using priority queue, it is $O(|V|^3)$

If using priority queue, it can become

$O(|E| + |V| \log(|V|))$ → Run $|V|$ times ➔ $O(|V||E| + |V|^2 \log(|V|))$

Floyd-Warshall Algorithm

- Floyd-Warshall Algorithm
  - runs in the same time complexity O($|V|^3$)

BUT, Dijkstra's doesn't work with negative-weight edges.

Introduction of Floyd-Warshall algorithm

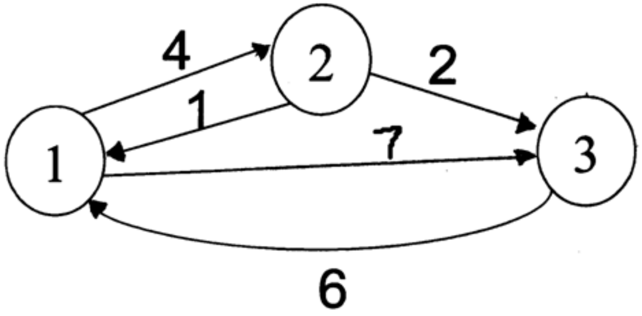
- The problem: find the shortest path between every pair of vertices of a graph
- The graph: may contain negative edges but
  no negative cycles
- A representation: a weight matrix where
  W(i,j)=0 if i=j.
  W(i,j)=∞ if there is no edge between i and j.
  W(i,j)="weight of edge"

How does it work?
Let us see some examples

Floyd Warshall Algorithm - Example

$$D^{(0)} = \begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \infty & 0 \end{bmatrix}$$

Original weights.



$$D^{(1)} = \begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \boxed{10} & 0 \end{bmatrix}$$

Consider Vertex 1 (the path goes through vertex 1):

D(3,2) = D(3,1) + D(1,2)

$$D^{(2)} = \begin{bmatrix} 0 & 4 & \boxed{6} \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

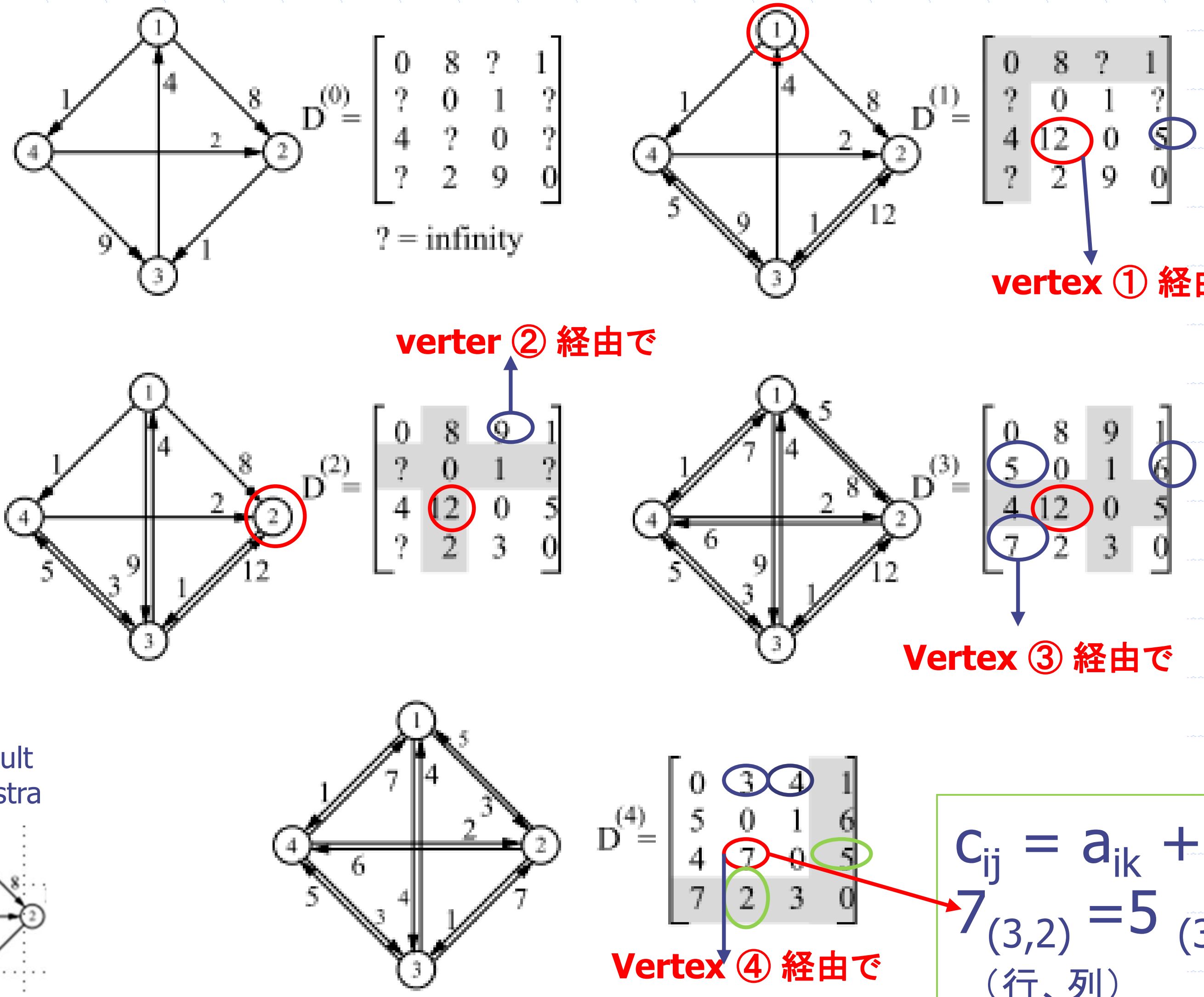
Consider Vertex 2 (the path goes through vertex 2):

D(1,3) = D(1,2) + D(2,3)

$$D^{(3)} = \begin{bmatrix} 0 & 4 & 6 \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

Consider Vertex 3 (the path goes through vertex 3):

Nothing changes.



$$D^{(0)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & ? & 0 & ? \\ ? & 2 & 9 & 0 \end{bmatrix}$$

? = infinity

$$D^{(1)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 9 & 0 \end{bmatrix}$$

vertex ① 経由で

verter ② 経由で

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 3 & 0 \end{bmatrix}$$

$$D^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

Vertex ③ 経由で

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

Vertex ④ 経由で

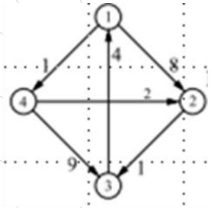
Figure 31: Floyd-Warshall Example.

$$c_{ij} = a_{ik} + b_{kj}$$

$$7_{(3,2)} = 5_{(3,4)} + 2_{(4,2)}$$

（行、列）

Below result from Dijkstra



$$\begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

Floyd Warshall Algorithm

- Looking at this example, we can come up with the following algorithm:
  - Let D store the matrix with the initial graph edge information initially, and update D with the calculated shortest paths.

```
For k=1 to n {
    For i=1 to n {
      For j=1 to n
          D[i,j] = min(D[i,j],D[i,k]+D[k,j])
   }
 }
```

  - The final D matrix will store all the shortest paths.
  - Looks like a matrix multiplication?

Compute matrix multiplication (1)

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

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

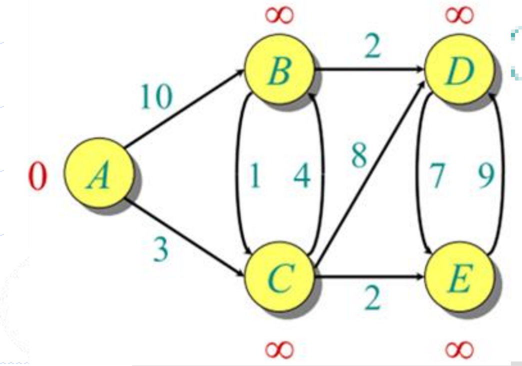
Time = $\Theta(n^3)$ using the standard algorithm.

What if we map “+” → “min” and “·” → “+”?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\} .$$

similar?

$$\boxed{d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}}$$



$^{(m-1)}$ “×” A.

$$= \mathrm{I} = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)}) .$$

Compute matrix multiplication (2)

Consequently, we can compute

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

yielding $D^{(n-1)} = (\delta(i, j))$.

- The final D matrix will store all the shortest paths.

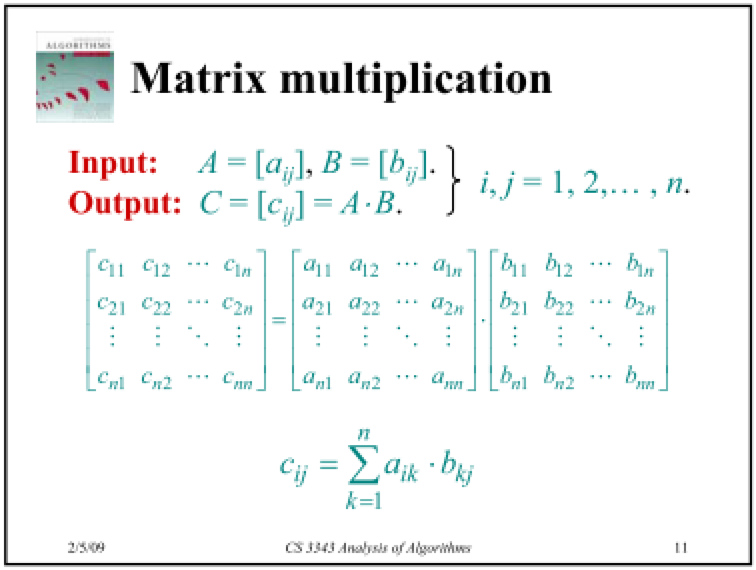
Time $= \Theta(n \cdot n^3) = \Theta(n^4)$.

Is it good?

?

n times of matrix multiplication, right?

Standard algorithm for multiplication

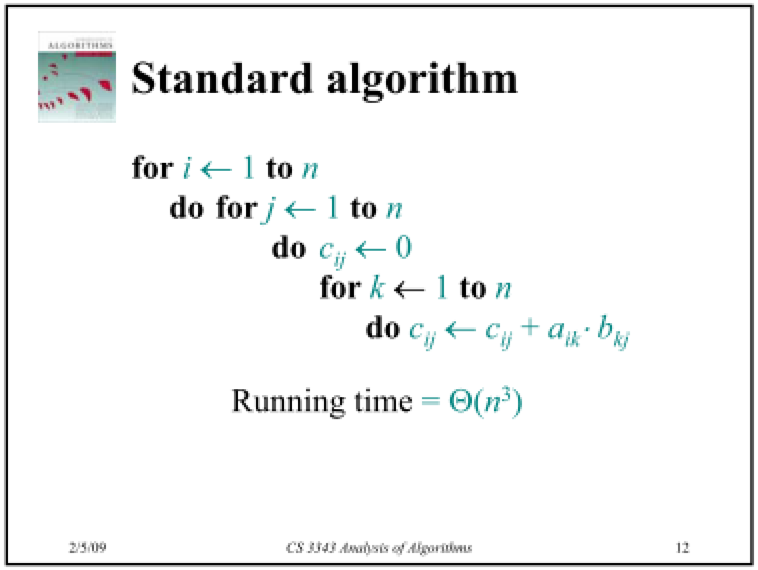


Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$ $\Big\}$ $i, j = 1, 2, \dots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

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

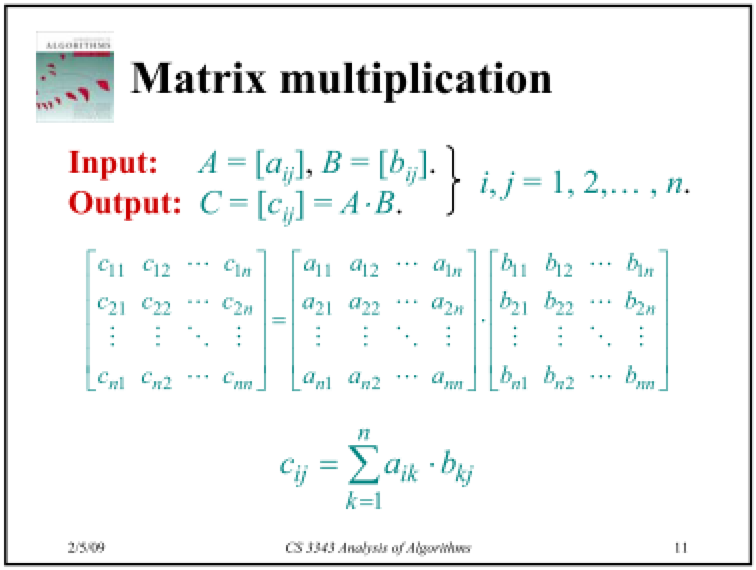


Standard algorithm

```
for $i \leftarrow 1$ to $n$
    do for $j \leftarrow 1$ to $n$
        do $c_{ij} \leftarrow 0$
            for $k \leftarrow 1$ to $n$
                do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
```

Running time $= \Theta(n^3)$

Improved algorithm for multiplication



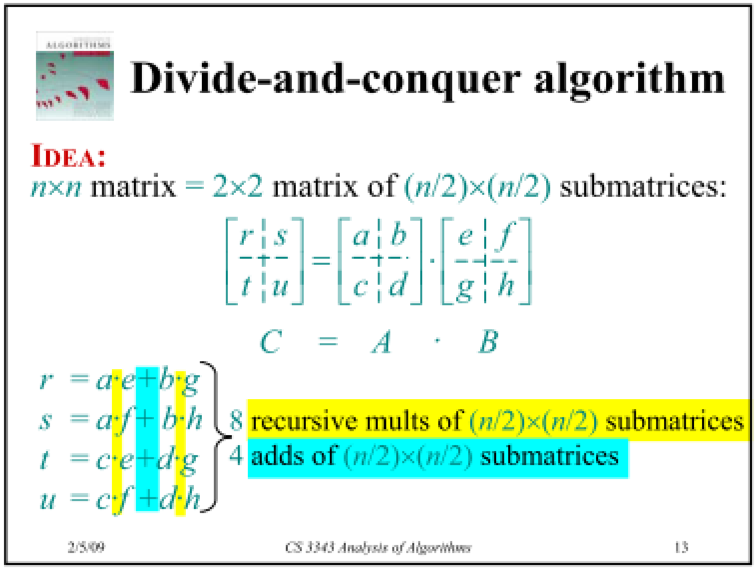
Matrix multiplication

Input: $A = [a_{ij}],\ B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$ $\Big\}\ i, j = 1, 2, \ldots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

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

2/5/09 CS 3343 Analysis of Algorithms 11



Divide-and-conquer algorithm

IDEA:
$n \times n$ matrix $= 2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= a \cdot e + b \cdot g \\ s &= a \cdot f + b \cdot h \\ t &= c \cdot e + d \cdot g \\ u &= c \cdot f + d \cdot h \end{aligned}\right\}$$

8 recursive mults of $(n/2) \times (n/2)$ submatrices
4 adds of $(n/2) \times (n/2)$ submatrices

2/5/09 CS 3343 Analysis of Algorithms 13

Repeated squaring: $A^{2k} = A^k \times A^k$.

Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n)\text{ squarings}}$.

Note: $A^{n-1} = A^n = A^{n+1} = \cdots$.

Time $= \Theta(n^3 \lg n)$.

All-pairs shortest path problem formal description

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

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

and for $m = 1, 2, \ldots, n-1$,

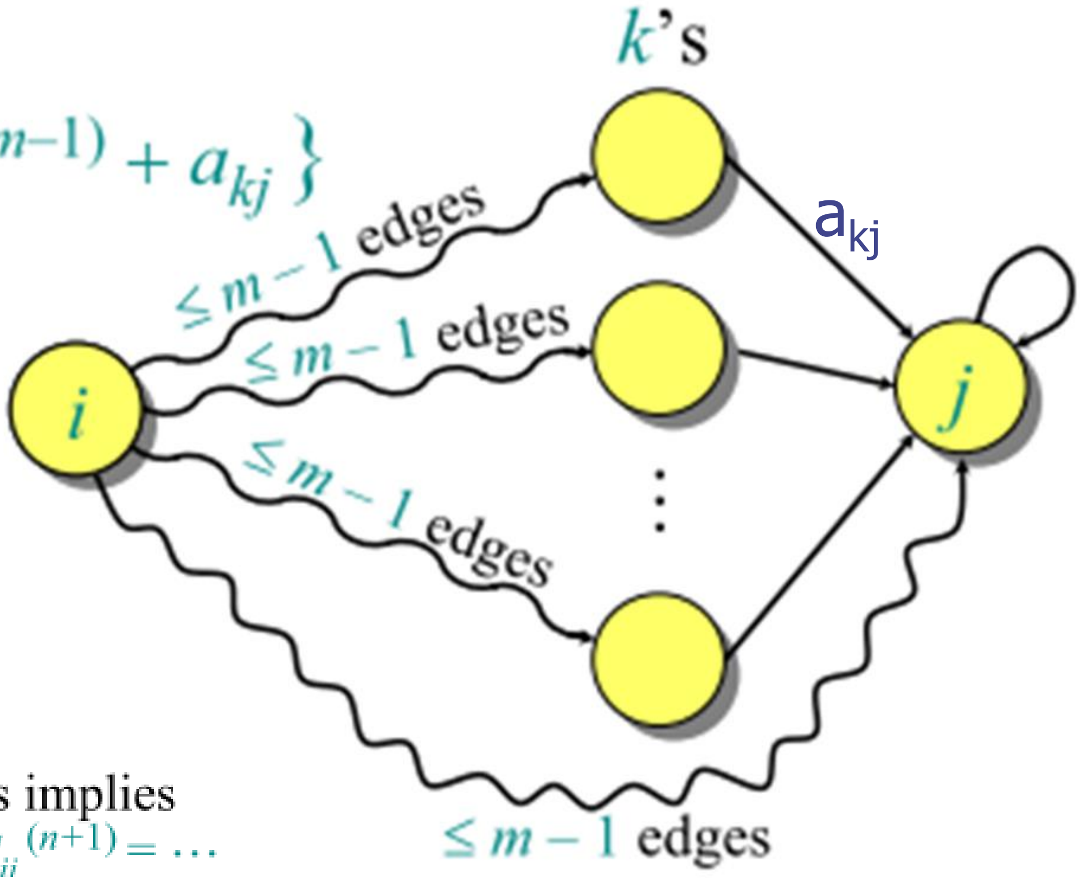
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \} .$$

Proof of Claim

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

Relaxation!

```
for $k \leftarrow 1$ to $n$
    do if $d_{ij} > d_{ik} + a_{kj}$
        then $d_{ij} \leftarrow d_{ik} + a_{kj}$
```

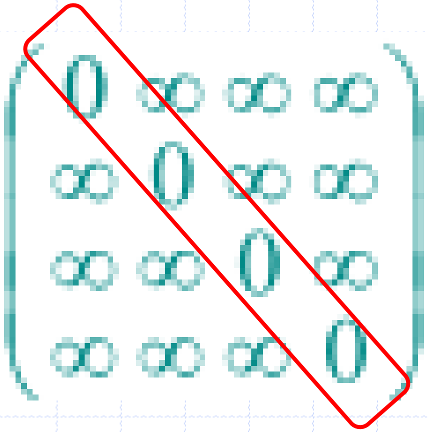


Notice that

No negative-weight cycles implies

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \ldots$$

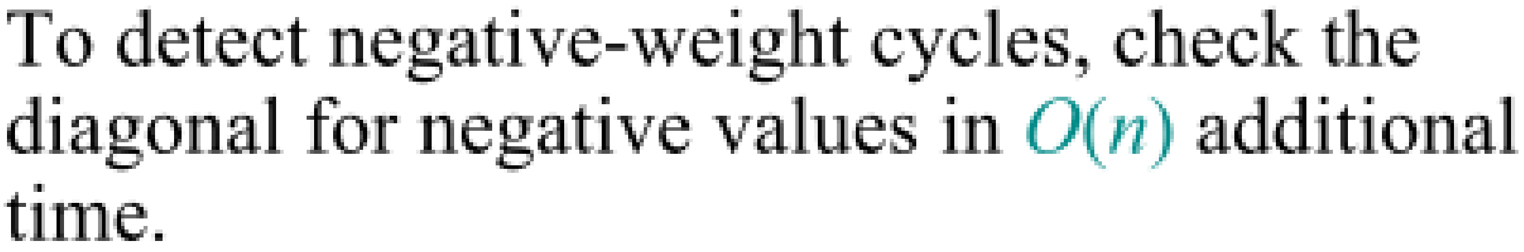
How to detect negative-weight cycle?



$$\begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

the diagonal: its value is 0?
it is negative value?

→ negative-weight cycle



To detect negative-weight cycles, check the diagonal for negative values in $O(n)$ additional time.

Input and output

Input reprentation:

We assume that we have a weight matrix

$W = (w_{ij})$ (i,j) in E

$w_{ij} = 0$ if $i=j$

$w_{ij} = w(i,j)$ if $i \neq j$ and (i,j) in E (has edge from i to j)

$w_{ij} = \infty$ if $i \neq j$ and (i,j) not in E (no edge from i to j)

Output representation:

If the graph has n vertices, we return a distance matrix (d_{ij}),

Where, d_{ij} the length of the path from i to j.

Intermediate Vertices

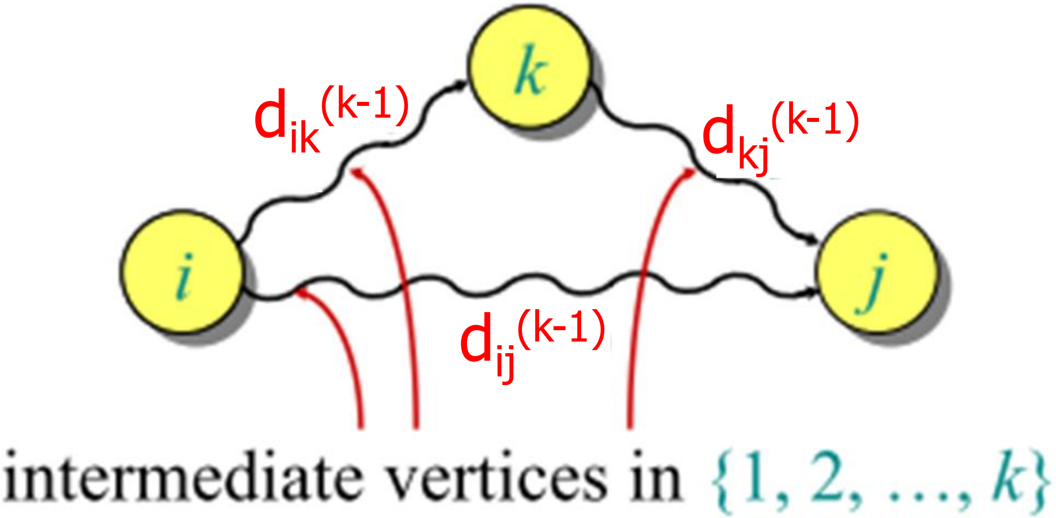
Without loss of generality, we will assume that V={1,2,…,n}, i.e., that the vertices of the graph are numbered from 1 to n.

Given a path p=(v_1, v_2,…, v_m) in the graph, we will call the vertices v_k with index k in {2,…,m-1} the intermediate vertices of p.

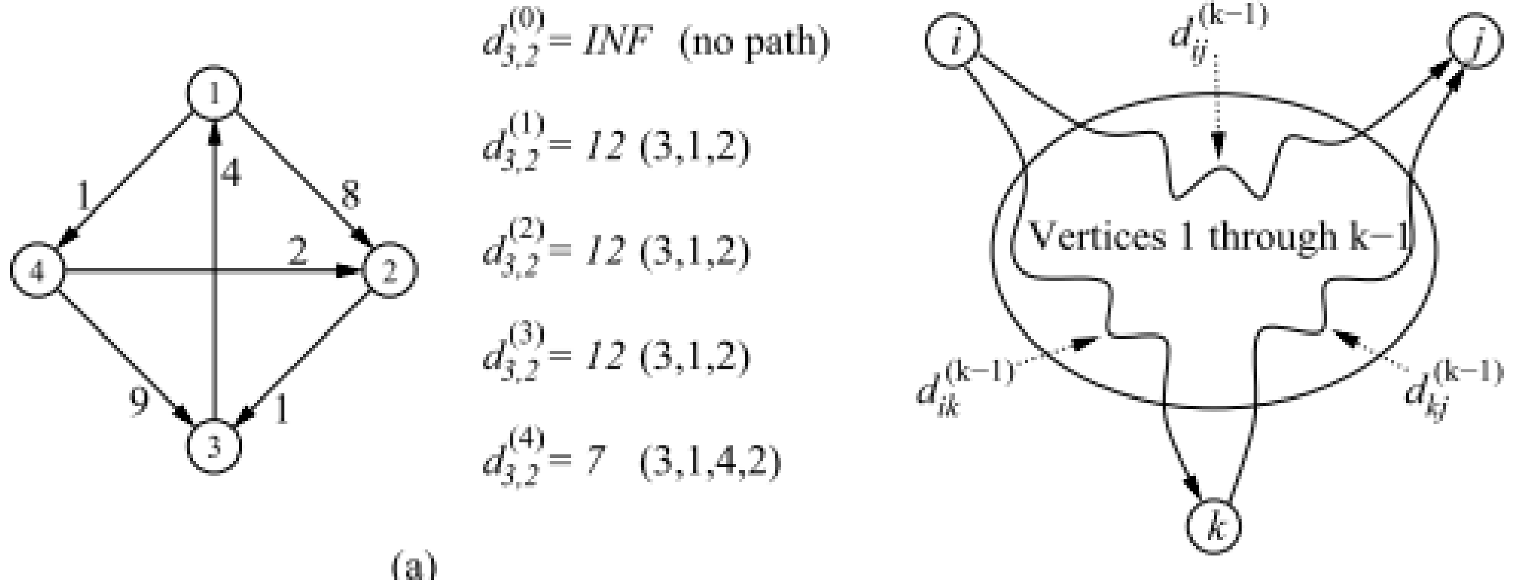
Conclusion

Therefore, we can conclude that

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



An example: take a look $d_{3,2}^{(k)}$



$d_{3,2}^{(0)} = INF$ (no path)

$d_{3,2}^{(1)} = 12$ (3,1,2)

$d_{3,2}^{(2)} = 12$ (3,1,2)

$d_{3,2}^{(3)} = 12$ (3,1,2)

$d_{3,2}^{(4)} = 7$ (3,1,4,2)

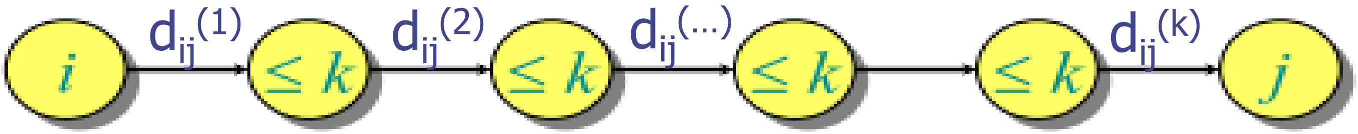
Figure 30: Floyd-Warshall Formulation.

Key Definition

The key to the Floyd-Warshall algorithm is the following definition:

Let $d_{ij}^{(k)}$ denote the length of the shortest path from i to j such that all intermediate vertices are contained in the set $\{1,\ldots,k\}$.

We have the following remark



Thus, the shortest path

$$\delta(i, j) = d_{ij}^{(n)}$$

Also, $d_{ij}^{(0)} = a_{ij}$

.

Recursive Formulation

If we do not use intermediate nodes, i.e., when k=0, then

$$d_{ij}^{(0)} = w_{ij}$$

If k>0, then

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

The Floyd-Warshall Algorithm

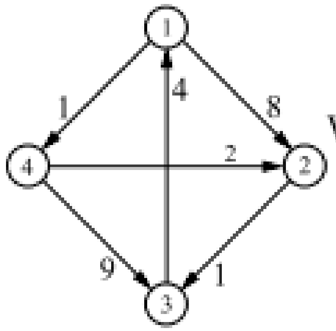
```
Floyd-Warshall(W)
n = # of rows of W;
$D^{(0)}$ = W;
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            $d_{ij}^{(k)} = \min_k\{d_{ij}^{(k-1)} , d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$;
        end-do;
    end-do;
end-do;
return $D^{(n)}$;
```

```
do if $d_{ij}^{(k-1)}>d_{ik}^{(k-1)}+d_{jk}^{(k-1)}$
    then $d_{ij}^{(k)} \leftarrow d_{ik}^{(k-1)}+d_{jk}^{(k-1)}$
```

relaxation

An example:

http://www.cs.umd.edu/~meesh/351/mount/lectures/lect24-floyd-warshall.pdf



$$W = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & ? & 0 & ? \\ ? & 2 & 9 & 0 \end{bmatrix} = D^{(1)}$$

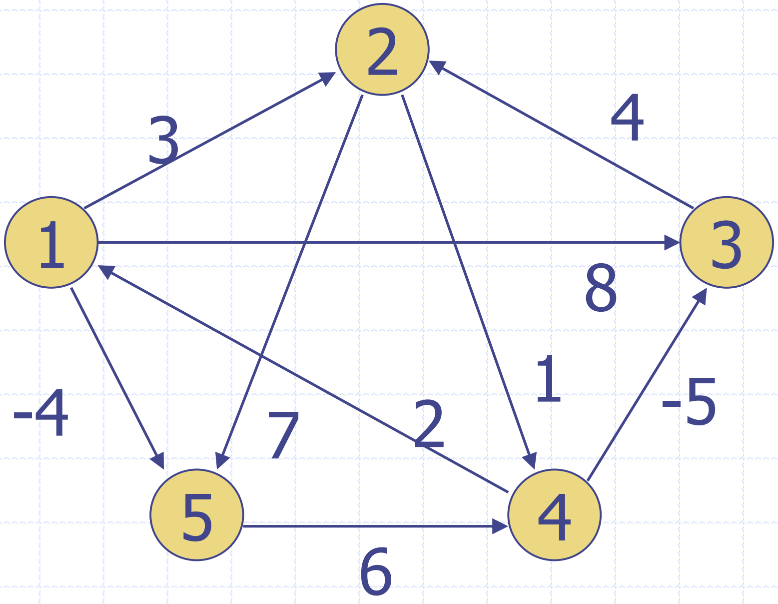
? = infinity

Work in class:
Please continue to get the final updated graph and matrix. (do not p10)

Figure 29: Shortest Path Example.

Work in class

Please write matrices: $D^{(0)}$, $D^{(1)}$, $D^{(2)}$, $D^{(3)}$, $D^{(4)}$, $D^{(5)}$,

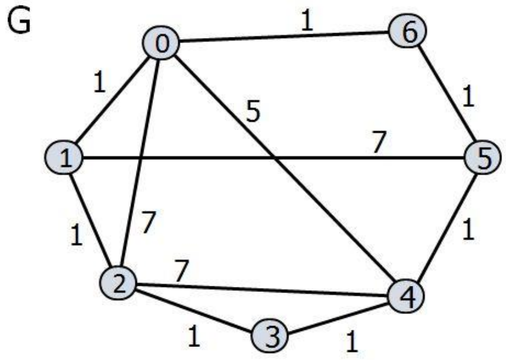


The Floyd-Warshall Algorithm – Pseudo-code

```
Floyd_Warshall(int n, int W[1..n, 1..n]) {
    array d[1..n, 1..n]
    for i = 1 to n do {                                 // initialize
        for j = 1 to n do {
            d[i,j] = W[i,j]
            pred[i,j] = null
        }
    }
    for k = 1 to n do                                   // use intermediates {1..k}
        for i = 1 to n do                               // ...from i
            for j = 1 to n do                           // ...to j
                if (d[i,k] + d[k,j]) < d[i,j]) {
                    d[i,j] = d[i,k] + d[k,j]            // new shorter path length
                    pred[i,j] = k                       // new path is through k
                }
    return d                                           // matrix of final distances
}
```

The Floyd-Warshall Algorithm in Java (1)

http://www.seas.gwu.edu/~simhaweb/cs151/lectures/module9/examples/

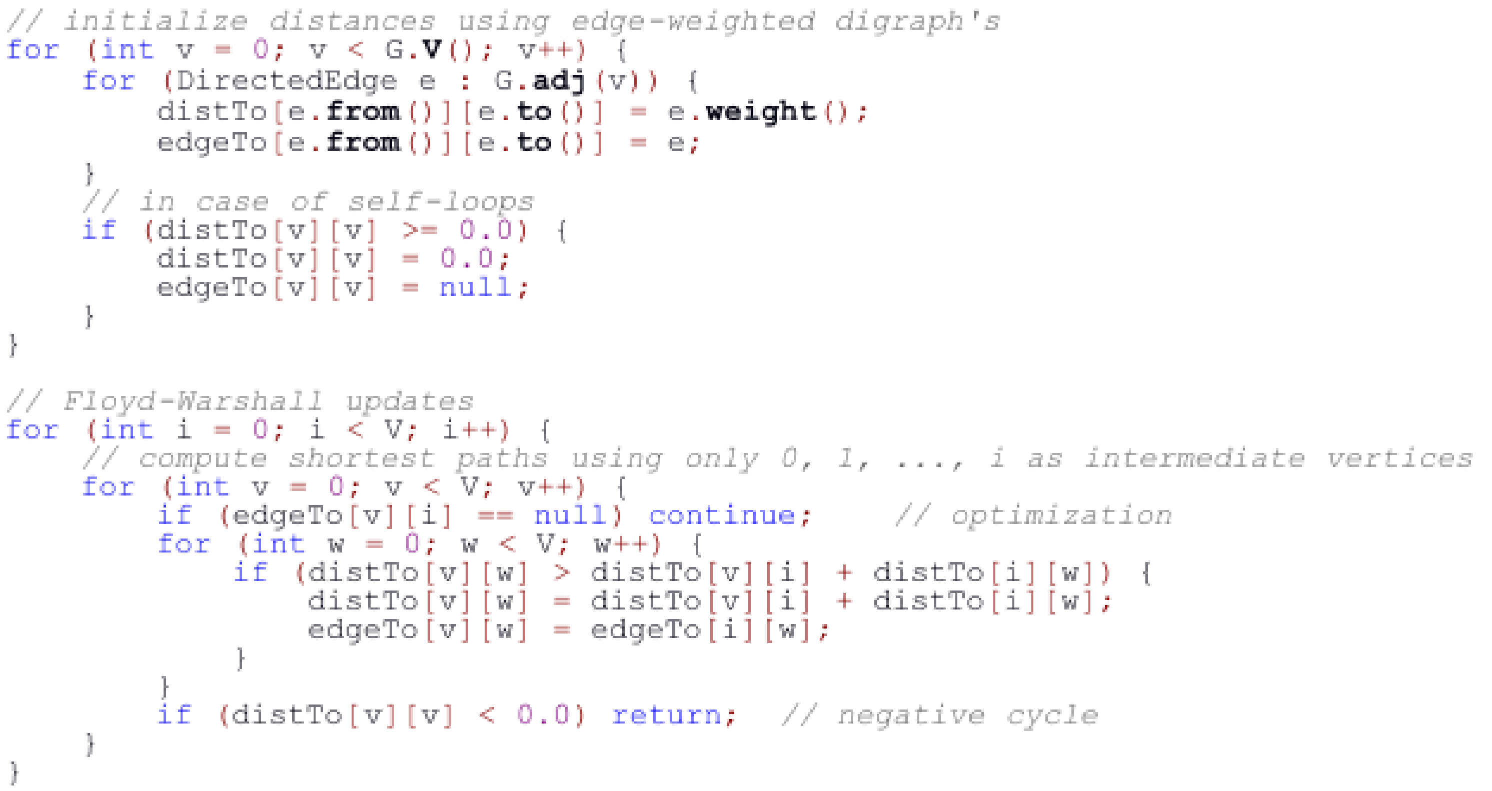


```
// Now iterate over k.

for (int k=0; k<numVertices; k++) {

  // Compute Dk[i][j], for each i,j

  for (int i=0; i<numVertices; i++) {
    for (int j=0; j<numVertices; j++) {
      if (i != j) {

        // D_k[i][j] = min ( D_k-1[i][j], D_k-1[i][k] + D_k-1[k][j].
        if (Dk_minus_one[i][j] < Dk_minus_one[i][k] + Dk_minus_one[k][j])
          Dk[i][j] = Dk_minus_one[i][j];
        else
          Dk[i][j] = Dk_minus_one[i][k] + Dk_minus_one[k][j];

      }
    }
  }

  // Now store current Dk into D_k-1
  for (int i=0; i<numVertices; i++) {
    for (int j=0; j<numVertices; j++) {
      Dk_minus_one[i][j] = Dk[i][j];
    }
  }

} // end-outermost-for
```

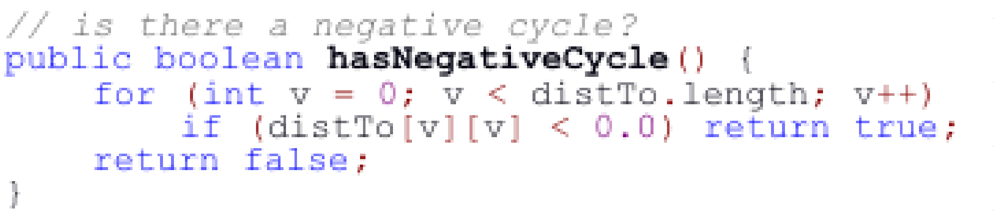
```
public static void main (String[] argv)
{
  // A test case.
    double[][] adjMatrix = {
      {0, 1, 7, 0, 5, 0, 1},
      {1, 0, 1, 0, 0, 7, 0},
      {7, 1, 0, 1, 7, 0, 0},
      {0, 0, 1, 0, 1, 0, 0},
      {5, 0, 7, 1, 0, 1, 0},
      {0, 7, 0, 0, 1, 0, 1},
      {1, 0, 0, 0, 0, 1, 0},
    };

    int n = adjMatrix.length;
    FloydWarshall fwAlg = new FloydWarshall ();
    fwAlg.initialize (n);
    fwAlg.allPairsShortestPaths (adjMatrix);

    // Print paths ... (not shown) ...

}
```

The Floyd-Warshall Algorithm in Java (2)

http://algs4.cs.princeton.edu/44sp/FloydWarshall.java.html



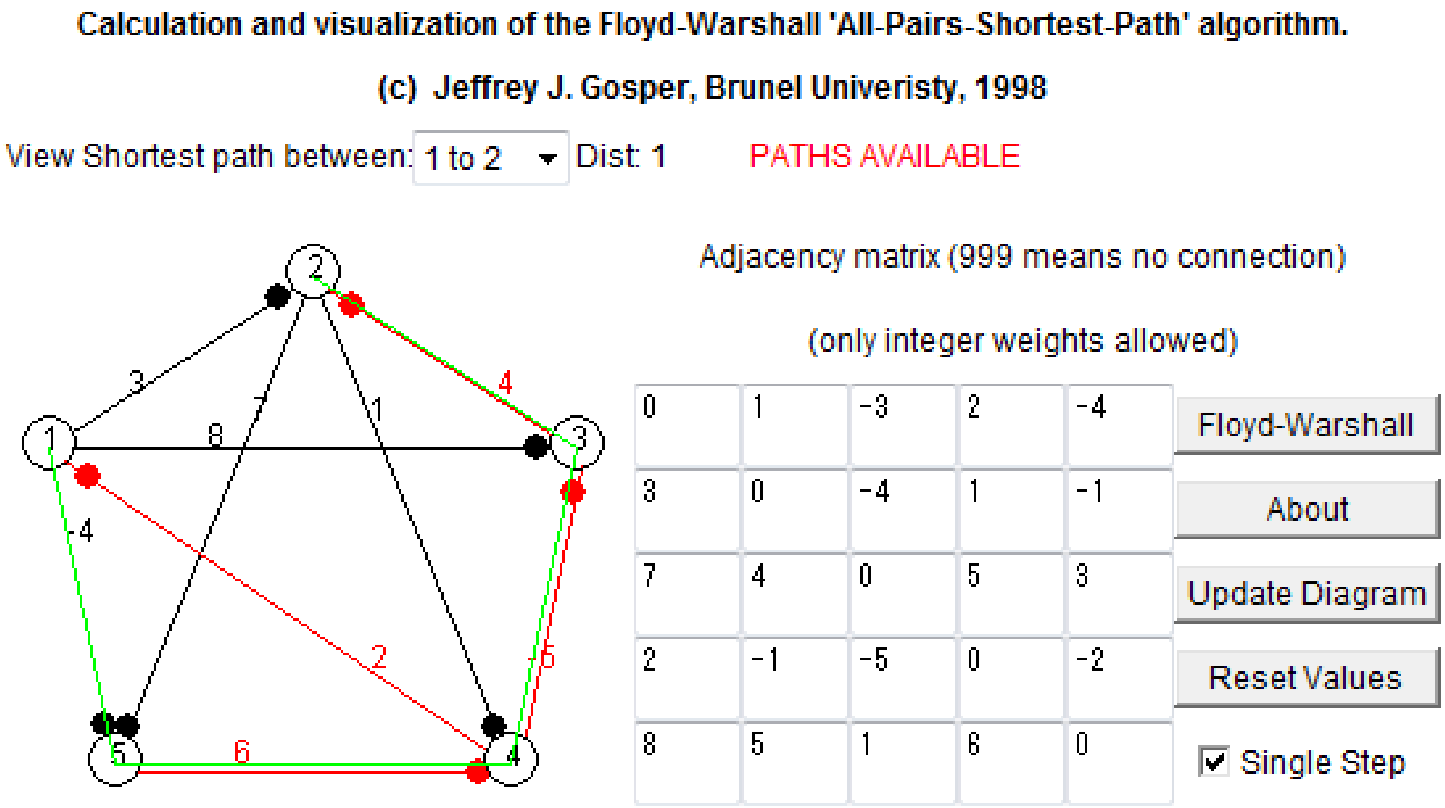
```
// initialize distances using edge-weighted digraph's
for (int v = 0; v < G.V(); v++) {
    for (DirectedEdge e : G.adj(v)) {
        distTo[e.from()][e.to()] = e.weight();
        edgeTo[e.from()][e.to()] = e;
    }
    // in case of self-loops
    if (distTo[v][v] >= 0.0) {
        distTo[v][v] = 0.0;
        edgeTo[v][v] = null;
    }
}

// Floyd-Warshall updates
for (int i = 0; i < V; i++) {
    // compute shortest paths using only 0, 1, ..., i as intermediate vertices
    for (int v = 0; v < V; v++) {
        if (edgeTo[v][i] == null) continue;     // optimization
        for (int w = 0; w < V; w++) {
            if (distTo[v][w] > distTo[v][i] + distTo[i][w]) {
                distTo[v][w] = distTo[v][i] + distTo[i][w];
                edgeTo[v][w] = edgeTo[i][w];
            }
        }
        if (distTo[v][v] < 0.0) return;  // negative cycle
    }
}
```



```
// is there a negative cycle?
public boolean hasNegativeCycle() {
    for (int v = 0; v < distTo.length; v++)
        if (distTo[v][v] < 0.0) return true;
    return false;
}
```

Visualization of the Floyd-Warshall Algorithm

http://www.pms.ifi.lmu.de/lehre/compgeometry/Gosper/shortest_path/shortest_path.html#visualization



Calculation and visualization of the Floyd-Warshall 'All-Pairs-Shortest-Path' algorithm.

(c) Jeffrey J. Gosper, Brunel Univeristy, 1998

View Shortest path between: 1 to 2 Dist: 1 PATHS AVAILABLE

Adjacency matrix (999 means no connection)

(only integer weights allowed)

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

http://homepage3.nifty.com/asagaya_avenue/trial/discussion/nishikawa_net.pdf

Exercise 14

Review for Final exam