All-Pairs Shortest Path Problem

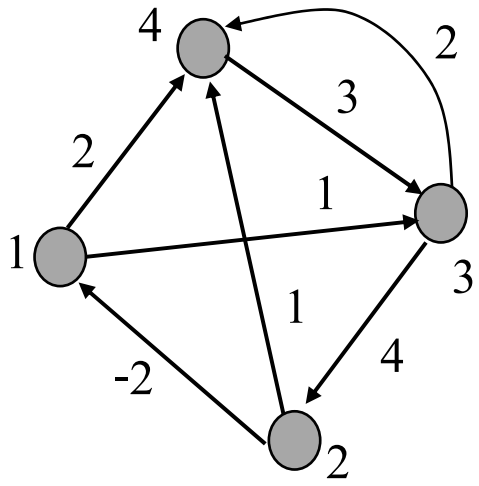
สมชาย ประสิทธิ์จูตระกูล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

Outline

- All-pairs shortest paths problem
- A dynamic-programming algorithm
- Floyd-Warshall algorithm

All-Pairs Shortest Paths

- Input : A weighted, directed graph $G = (V, E)$
  with a weight function $w : E \to \mathbf{R}$
- Output : For every pair of vertices i and $j \in V$,
  find a least-weight (shortest) path from i to j



no negative-weight cycles

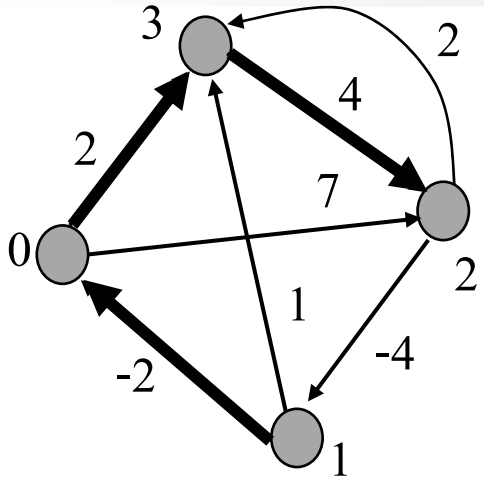
All-Pairs Shortest Paths

- Running a single-source shortest-paths algorithm once for each vertex
  - Dijkstra (all edge weights are nonnegative)
    - linear array : $O(v \cdot v^2) = O(v^3)$
    - binary heap : $O(v \cdot e \log v)$
    - Fibonacci heap : $O(v \cdot (e + v \log v)) = O(ve + v^2 \log v)$
  - Bellman-Ford (negative-weight edges are allowed)
    - $O(v^2e)$ i.e., $O(v^4)$ for dense graph

All-Pairs Shortest Paths

- We will present three dynamic-programming algorithms
  - $\Theta(v^4)$ slow algorithm
  - $\Theta(v^3 \log v)$ algorithm using repeated squaring
  - $\Theta(v^3)$ Floyd-Warshall algorithm

Input and Output



	0	1	2	3
0	0	-	7	2
1	-2	0	-	1
2	-	-4	0	2
3	-	-	4	0

W

Input

D

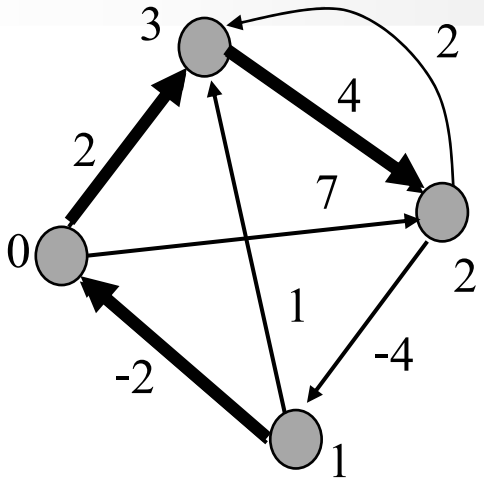
	0	1	2	3
0	0	2	6	2
1	-2	0	4	0
2	-6	-4	0	-4
3	-2	0	4	0

	0	1	2	3
0	-	2	3	0
1	1	-	3	0
2	1	2	-	0
3	1	2	3	-

P

Output

Shortest Paths Matrix



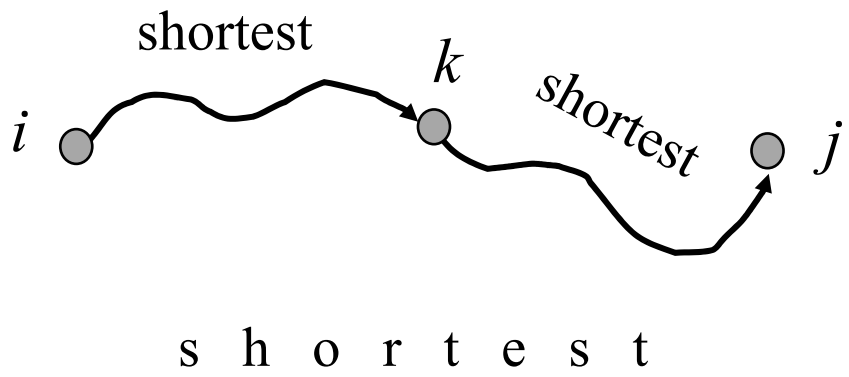
	0	1	2	3
0	-	2	3	0
1	1	-	3	0
2	1	2	-	0
3	1	2	3	-

P

```
01: PrintShortestPath( P, i, j ) {
02:   if ( i == j ) print i
04:   else if ( P[i][j] == null )
05:           print "No path from " + i + " to " + j
06:         else {
07:           PrintShortestPath( P, i, P[i][j] )
08:           print j
09:         }
10: }
```

A Dynamic-Programming Algorithm

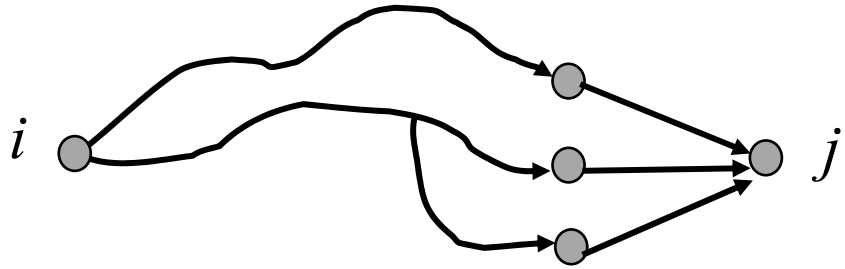
- Shortest path problem has optimal substructure



Recurrence for an Optimal Solution

- Let $d_{ij}(m)$ be the minimum weight of any path from i to j that contains at most m edges

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



$$d_{ij}(m) = \min\left(d_{ij}(m\text{-}1), \min_{1 \le k \le v} \{ d_{ik}(m\text{-}1) + w_{kj} \} \right)$$

Computing the Shortest Path Weights

- Let $d_{ij}(m)$ be the minimum weight of any path from i to j that contains at most m edges
- Any simple path of a graph with v vertices has at most v-1 edges, therefore we want $d_{ij}(v\text{-}1)$.
- Given a graph in form of an adjacency matrix W we start with $D(1)$ which equals to W, then compute $D(2)$, $D(3)$, ..., $D(v\text{-}1)$

$$d_{ij}(m) = \min\left(d_{ij}(m\text{-}1), \min_{1 \le k \le v} \{ d_{ik}(m\text{-}1) + w_{kj} \} \right)$$

Computing the Shortest Path Weights

```
01: AllPairsShortestPathSlow( W ) {
02:   D(1) = W
03:   for(int m = 2; m < W.length-1; m++)
04:     D(m) = ExtendShortestPath( D(m-1), W );
05:   return D(W.length - 1);
06: }
```

$\Theta(v^4)$

```
01: ExtendShortestPath( D, W ) {
02:   for (int i = 0; i < W.length; i++ ) {
03:     for (int j = 0; j < W.length; j++) {
04:       DD[i][j] = D[i][j];
05:       for (int k = 0; k < W.length; k++) {
06:         DD[i][j] = min( DD[i][j], D[i][k] + W[k][j]);
07:       }
08:     }
09:   }
10:   return DD;
11: }
```

Constructing an Optimal Solution

```
01: ExtendShortestPath( D, W, P ) {
02:   for (int i = 0; i < W.length; i++ ) {
03:     for (int j = 0; j < W.length; j++) {
04:       DD[i][j] = D[i][j];
05:       for (int k = 0; k < W.length; k++) {
06:         if (DD[i][j] > D[i][k] + W[k][j]) {
07:           P[i][j] = k;
08:           DD[i][j] = D[i][k] + W[k][j]);
09:         }
10:       }
11:     }
12:   }
13:   return DD;
14: }
```

Repeated Squaring

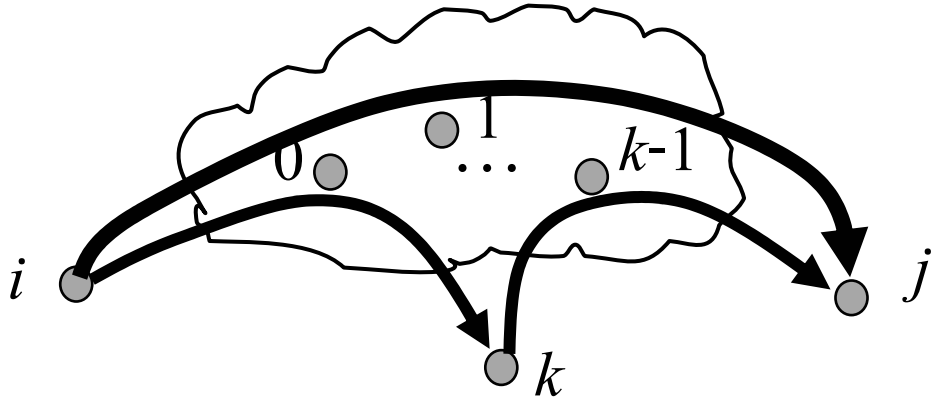
- Let $d_{ij}(m)$ be the minimum weight of any path from i to j that contains at most m edges
- We start with $D(1)$ then compute $D(2)$, ..., $D(v\text{-}1)$
- Why not compute $D(2)$, $D(4)$, ..., $D(2^k)$ where 2^k is the smallest value not less than v-1 ?

$\Theta(v^3 \log v)$

```
01: AllPairShortestPathRepeatedSquare( W ) {
02:   D(1) = W; m = 1;
03:   for(int m = 1; m < W.length-1; m = 2*m )
04:     D(2*m) = ExtendShortestPath( D(m), D(m) );
05:   return D(2*m);
06: }
```

Floyd-Warshall Algorithm

- Let $d_{ij}(k)$ be the weight of a shortest path from i to j with all intermediate vertices in the set $\{0, 1, ..., k\}$



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

$$d_{ij}(-1) = w_{ij}$$

Computing the Shortest Path Weights

- Let $d_{ij}(k)$ be the weight of a shortest path from i to j with all intermediate vertices in the set $\{0, 1, ..., k\}$
- Any simple path of a graph with v vertices in the set $\{0, 1, 2, ..., v\text{-}1\}$ can has all the vertices be its intermediate, therefore we want $d_{ij}(v\text{-}1)$.
- Given a graph in form of an adjacency matrix W we start with $D(\text{-}1)$ which equals to W, then compute $D(0)$, $D(1)$, ..., $D(v\text{-}1)$

Computing the Shortest Path Weights

```
01: AllPairShortestPathFloydWarShall( W ) {
02:   D(-1) = W
03:   for (int k = 0; k < W.length; k++) {
04:     for (int i = 0; i < W.length; i++) {
05:       for (int j = 0; j < W.length; j++) {
06:         D(k)[i][j] = min( D(k-1)[i][j],
07:                           D(k-1)[i][k] + D(k-1)[k][j] );
08:       }
09:     }
10:   }
11:   retuen D(W.length-1);
12:
```

$\Theta(v^3)$

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

$$d_{ij}(-1) = w_{ij}$$