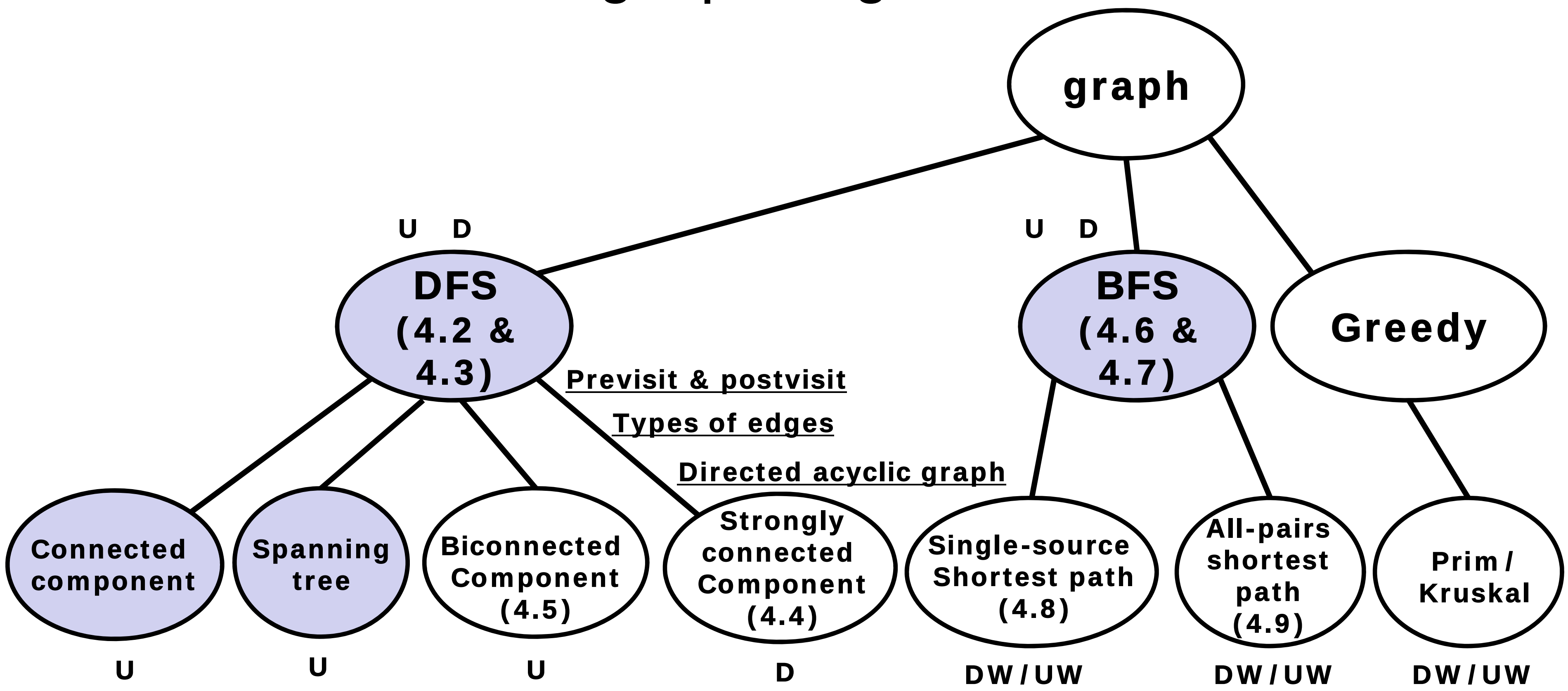
4.9 All pairs shortest path

Classification of graph algorithms



4.9 All pairs shortest path

(1) Basic concept (1)

- The problem of finding shortest paths for every two vertices *u* and *v*.

- Solving single-source shortest path for all vertices in G

- Floyd’s algorithm
  - A kind of dynamic programming

4.9 All pairs shortest path

(1) Basic concept (1)

- Dynamic programming
  - Finding an optimal solution for a sequence of decision
  - Decomposing a problem into a set of subproblems
  - Exploring all possible subproblems to find an optimal solution

4.9 All pairs shortest path

(2) Floyd's algorithm (1)

- Finding the all-pair's shortest path.
  - Input: adjacency matrix of a graph.
  - The weight of a path between two vertices is the sum of the weights of the edges along that path.
  - Negative weight is allowed.
  - Negative cycle is not allowed.

4.9 All pairs shortest path

(2) Floyd's algorithm (2)

- $A^k[i][j]$:
  - The cost of the shortest path from vertex i to j, using only those intermediate vertices with an index $\leq$ k.
- $A^{-1}[i][j]$
  - the weight of an edge connecting vertex i and vertex j

4.9 All pairs shortest path

(2) Floyd's algorithm (3)

– Basic idea:

- Starting from A^{-1}, successively generate the matrices to A^1, A^2, ..., A^n.

$$A^k[i][j] = \min \{ A^{k-1}[i][j], A^{k-1}[i][k] + A^{k-1}[k][j] \}$$

$$A^{-1}[i][j] = cost[i][j]$$

4.9 All pairs shortest path

(2) Floyd's algorithm (4)

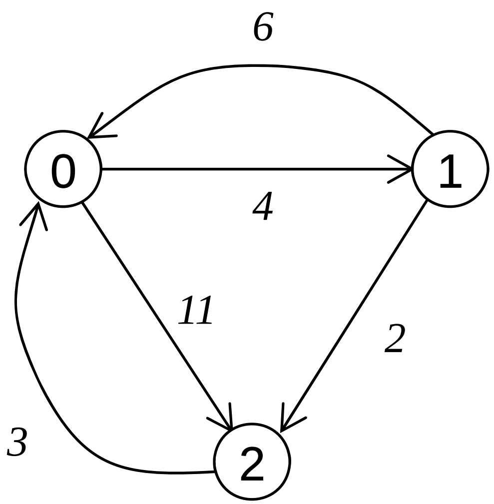
```
void Floyd ( int cost[][], int dist[][], int n )
{
    int i, j, k;
    for ( i = 0; i < n; i++ )
       for ( j = 0; j < n; j++ )
           dist[i][j] = cost[i][j];

    for ( k = 0; k < n; k++ ) {
       for ( i = 0; i < n; i++ )
           for ( j = 0; j < n; j++ )
              if ( dist[i][k] + dist[k][j] < dist[i][j] )
                  dist[i][j] = dist[i][k] + dist[k][j];
    }
}
```

4.9 All pairs shortest path

(2) Floyd's algorithm (5)

– Example:



A^{-1}	0	1	2
0			
1			
2			

A^{0}	0	1	2
0			
1			
2			

A^{1}	0	1	2
0			
1			
2			

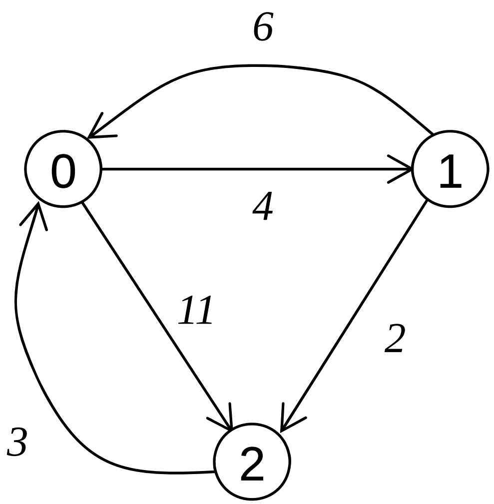
A^{2}	0	1	2
0			
1			
2			

```
if ( dist[i][k] + dist[k][j] < dist[i][j] )
    dist[i][j] = dist[i][k] + dist[k][j];
```

4.9 All pairs shortest path

(2) Floyd's algorithm (5)

– Example:



A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{0}	0	1	2
0			
1			
2			

A^{1}	0	1	2
0			
1			
2			

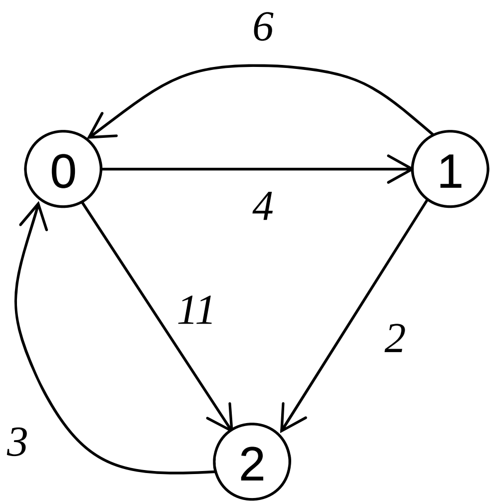
A^{2}	0	1	2
0			
1			
2			

```
if ( dist[i][k] + dist[k][j] < dist[i][j] )
    dist[i][j] = dist[i][k] + dist[k][j];
```

4.9 All pairs shortest path

(2) Floyd's algorithm (5)

– Example:



A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{0}	0	1	2
0	0	4	11
1	6	0	2
2	3	**7**	0

A^{1}	0	1	2
0			
1			
2			

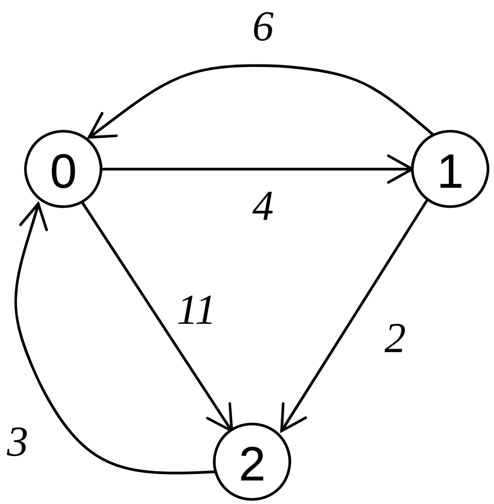
A^{2}	0	1	2
0			
1			
2			

```
if ( dist[i][k] + dist[k][j] < dist[i][j] )
    dist[i][j] = dist[i][k] + dist[k][j];
```

4.9 All pairs shortest path

(2) Floyd's algorithm (5)

– Example:



A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{0}	0	1	2
0	0	4	11
1	6	0	2
2	3	**7**	0

A^{1}	0	1	2
0	0	4	**6**
1	6	0	2
2	0	7	0

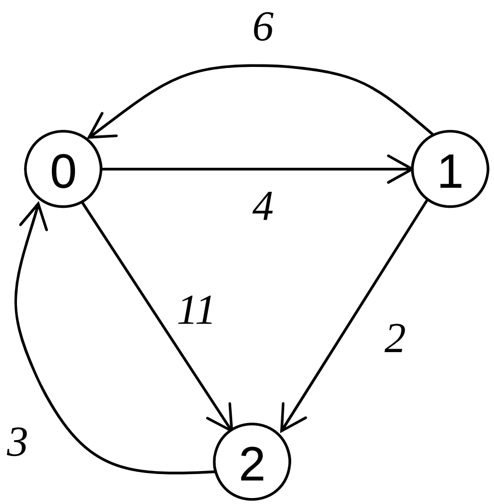
A^{2}	0	1	2
0			
1			
2			

```
if ( dist[i][k] + dist[k][j] < dist[i][j] )
    dist[i][j] = dist[i][k] + dist[k][j];
```

4.9 All pairs shortest path

(2) Floyd's algorithm (5)

– Example:



A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{0}	0	1	2
0	0	4	11
1	6	0	2
2	3	**7**	0

A^{1}	0	1	2
0	0	4	**6**
1	6	0	2
2	3	7	0

A^{2}	0	1	2
0	0	4	6
1	**5**	0	2
2	3	7	0

```
if ( dist[i][k] + dist[k][j] < dist[i][j] )
    dist[i][j] = dist[i][k] + dist[k][j];
```

All about graph

Type	Purpose	Operations	Performance
DFS	Traverse all vertices	Visiting all vertices & visiting all edges	O(n) + O(m)
SCC	Finding SCC	DFS on G^R and G	O(DFS)
BFS	Traverse all vertices	Visiting all vertices & visiting all edges	O(n) + O(m)
Dijkstra	Single source shortest path	Visiting all edges & managing queue	$O(n^2)$ (original) → O(m) + O(n log n)
Floyd	All pairs shortest path	Incrementing k	$O(n^3)$
Kruskal (Greedy)			
Prim (Greedy)			
MultiStage (Dynamic)			

4. Graph

4.0 Introduction

4.1 Why graph?

4.2 Depth-first search in undirected graph

4.3 Depth-first search in directed graphs

4.4 Strongly connected components

4.5 Biconnected component

4.6 Distances

4.7 Breadth-first search

4.8 Single source shortest path

4.9 All pairs shortest path

Contents

0. Prologue

1. Divide & conquer

2. Graph

3. Greedy algorithm

4. Dynamic programming

4.9 All pairs shortest path

- 다음은 Floyd algorithm에 대한 설명이다. 잘못된 것을 모두 고르시오.

(a) Floyd algorithm은 Dijkstra algorithm을 n번 수행한 것과 같은 시간 복잡도를 갖는다 (n: vertex의 수).

(b) Floyd algorithm은 dynamic programming의 일종이다.

(c) Floyd algorithm은 Dijkstra algorithm과 같이 negative edge가 있는 graph는 작동하지 않는다.

(d) Floyd algorithm은 Bellman-Ford algorithm과 같이 negative edge가 있는 graph에서도 작동한다.