

자료구조 & 알고리즘

for(A;B;C)
D;

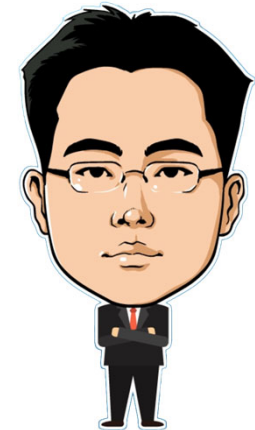


알고리즘
(Algorithms)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



백문이불여일타(百聞而不如一打)



목 차



- **그래프 알고리즘**
- **탐욕 알고리즘**
- **동적 프로그래밍**
- **고급 알고리즘 설계와 분석**



그래프 알고리즘



- 그래프 알고리즘

백문이불여일타(百聞而不如一打)

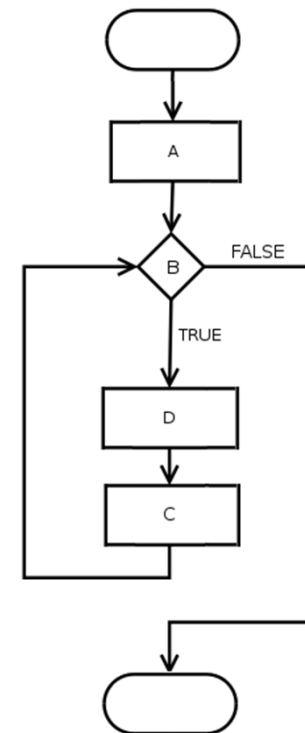
- 그래프의 이해: 그래프 표현과 순회
- 최소 신장 트리
- 최단 경로

- 탐욕 알고리즘

- 동적 프로그래밍

- 고급 알고리즘 설계와 분석

for(A;B;C)
D;

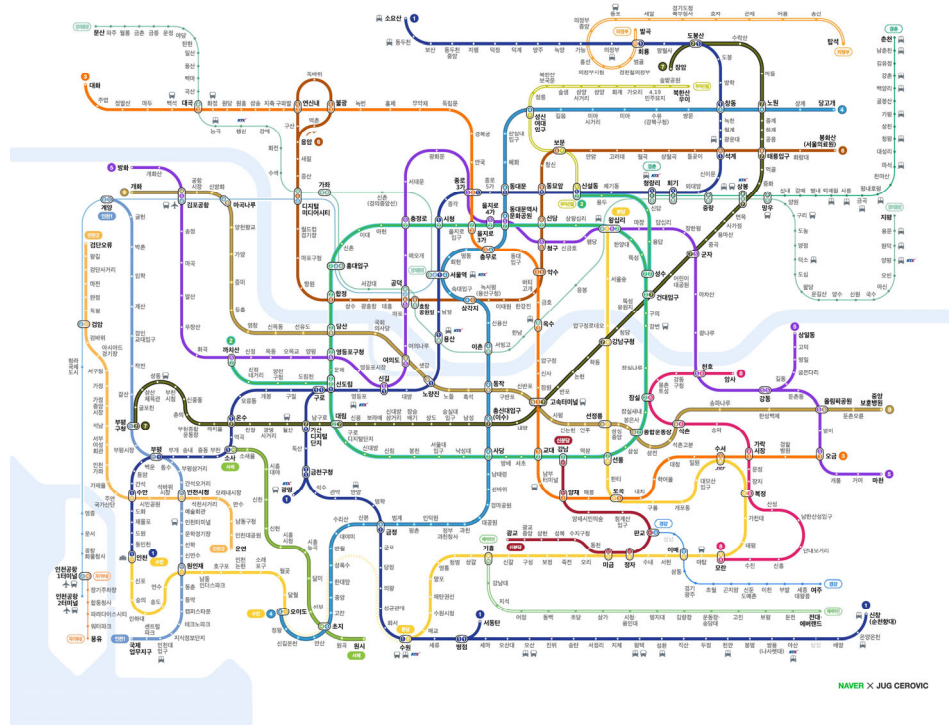


그래프 이해 (1/3)

- **그래프(Graph)**

- 연결되어 있는 원소 간의 관계를 표현하는 자료구조

- 그래프의 예: 인맥 지도, 수도 배관 배수 시스템, 물질의 분자 구조

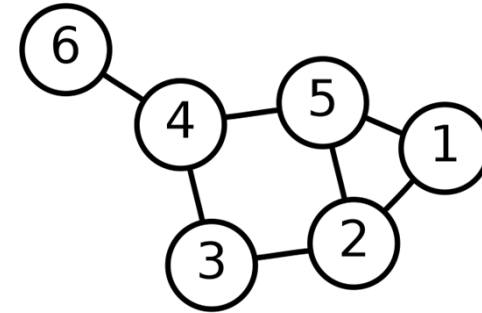


그래프 이해 (2/3)

● 그래프 정의

○ 그래프 **G**는 집합(Set) 두 개로 구성

- 정점(Vertex 또는 Node)
- 간선(Edge)



$$G = (V, E)$$

- **V** 는 그래프에 있는 정점들의 집합을 의미한다(대상: 대상물, 개념 등).
- **E** 는 정점을 연결하는 간선들의 집합을 의미한다(대상들 간의 관계).

그래프의 이해 (3/3)

● 그래프 종류

○ 무향 그래프(Undirected Graph)

- 두 정점을 연결하는 간선에 방향성이 없는 그래프

○ 유향 그래프(Directed Graph)

- 두 정점을 연결하는 간선에 방향성이 있는 그래프

○ 가중치 그래프(Weight Graph)

- 두 정점을 연결하는 간선에 가중치가 할당된 그래프
 - 가중치는 두 정점 사이의 거리 또는 지나는 시간이 될 수도 있다.
 - 또한 음수인 경우도 존재한다.

○ 완전 그래프(Complete Graph)

- 모든 정점들 사이에 1:1로 직접 연결된 간선을 지닌 그래프

○ 부분 그래프(Subgraph)

- 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
 - 부분 그래프는 원래의 그래프에 없는 정점이나 간선을 포함하지 않는다.

○ 트리(Tree): 순환이 없는 연결된 그래프

그래프 표현: 인접 행렬 (1/3)

- **인접 행렬**(Adjacent Matrix)

- 순차 자료구조를 이용하는 2차원 배열의 방법

- 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
 - N 개의 정점을 가진 그래프: **$N \times N$ 정방 행렬**
 - 행렬의 행과 열: 그래프의 정점
 - 행렬 값: 두 정점이 인접되어 있으면 1, 인접되어 있지 않으면 0

- 무향 그래프의 인접 행렬

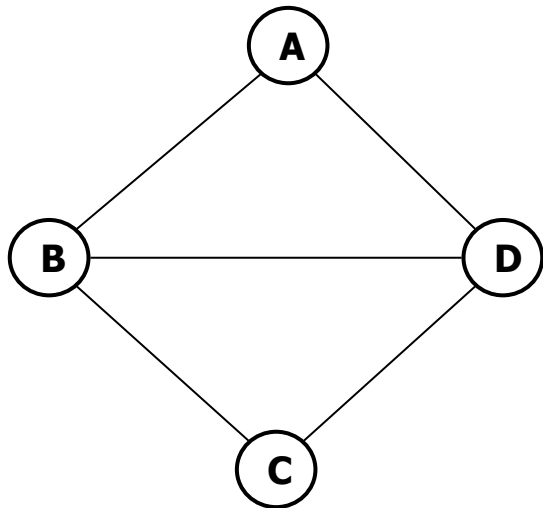
- 행 i 의 합 = 열 i 의 합 = 정점 i 의 차수

- 유향 그래프의 인접 행렬

- 행 i 의 합 = 정점 i 의 진출 차수
- 열 i 의 합 = 정점 i 의 진입 차수

그래프 표현: 인접 행렬 (2/3)

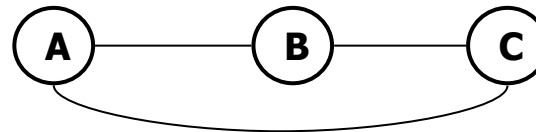
- 인접 행렬: 2차원 배열



[그래프 G1]

	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

정점 B의 차수: $3 = 1+0+1+1$

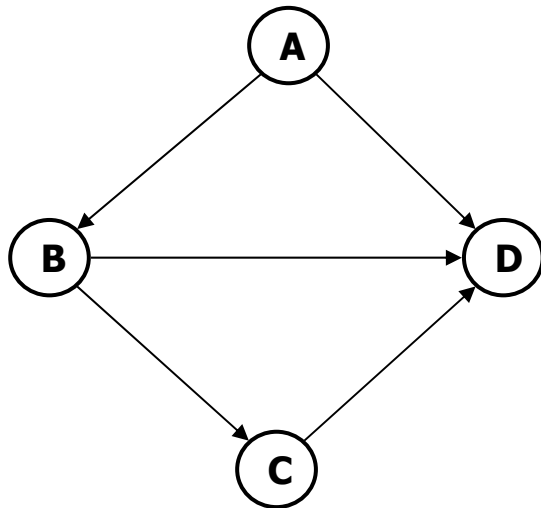


[그래프 G2]

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

그래프 표현: 인접 행렬 (3/3)

- 인접 행렬: 2차원 배열

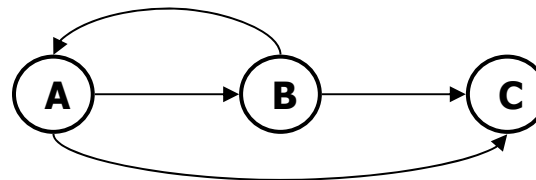


[그래프 G3]

	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

정점 B의 진출 차수: $2 = 0 + 0 + 1 + 1$

정점 B의 진입 차수: $1 = 1 + 0 + 0 + 0$



[그래프 G4]

	A	B	C
A	0	1	1
B	1	0	1
C	0	0	0

그래프 표현: 인접 리스트 (1/3)

- **인접 리스트**(Adjacent List)
 - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트
 - 정점의 헤드 노드
 - 정점에 대한 리스트의 시작을 표현
 - 인접 리스트의 각 노드
 - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
 - 각 정점의 차수만큼 노드를 연결
 - 리스트 내의 노드들은 인접 정점에 대해서 오름 차순으로 연결

그래프 표현: 인접 리스트 (2/3)

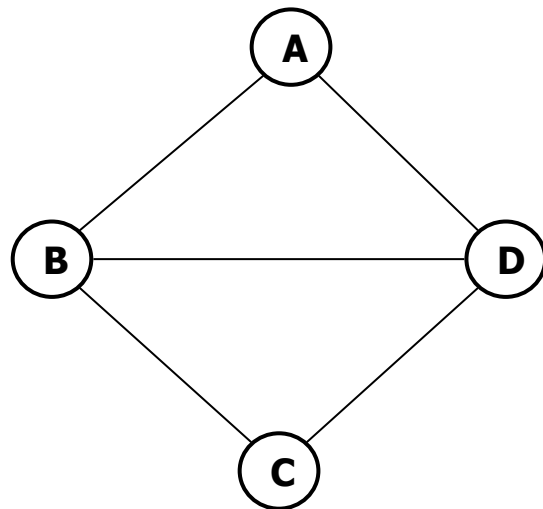
● 인접 리스트: 무향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 무향 그래프

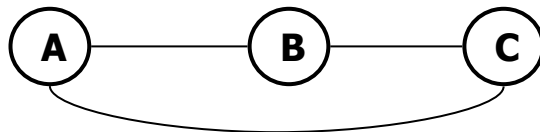
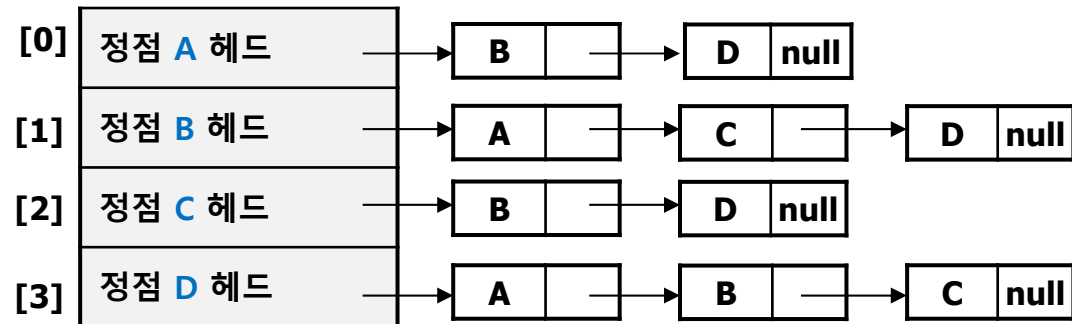
헤드 노드의 배열 크기: n

연결하는 노드의 총 수: $2e$

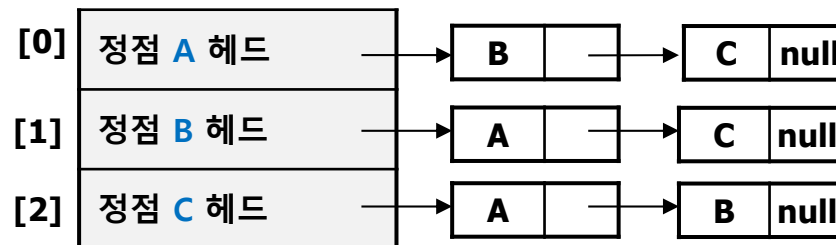
각 정점의 헤드에 연결된 노드의 수: 정점의 차수



[그래프 G1]



[그래프 G2]



그래프 표현: 인접 리스트 (3/3)

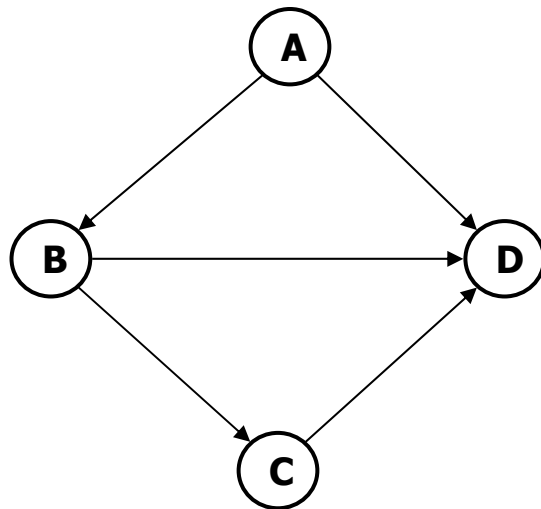
● 인접 리스트: 유향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 유향 그래프

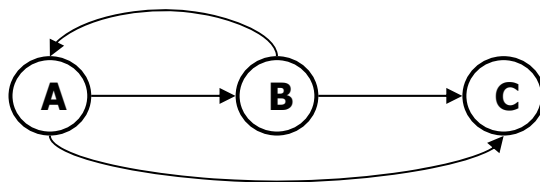
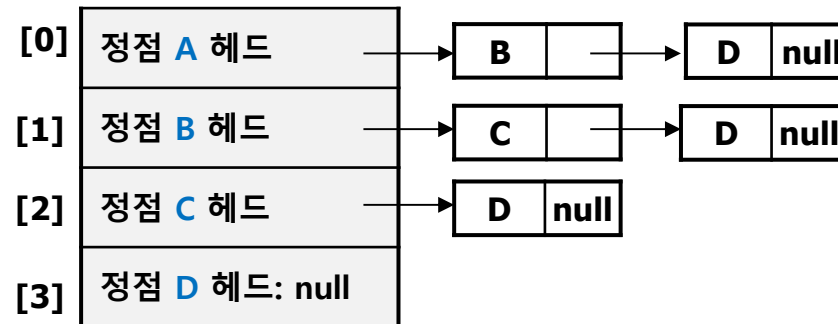
헤드 노드의 배열 크기: n

연결하는 노드의 총 수: e

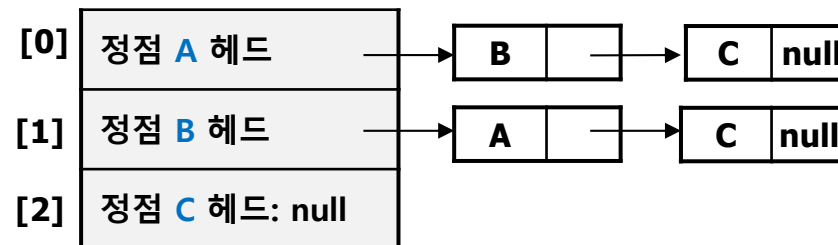
각 정점의 헤드에 연결된 노드의 수: 정점의 진출 차수



[그래프 G3]



[그래프 G4]



그래프 순회 (1/2)

- **그래프 순회**(Graph Traversal)

- 그래프 탐색(Graph Search)

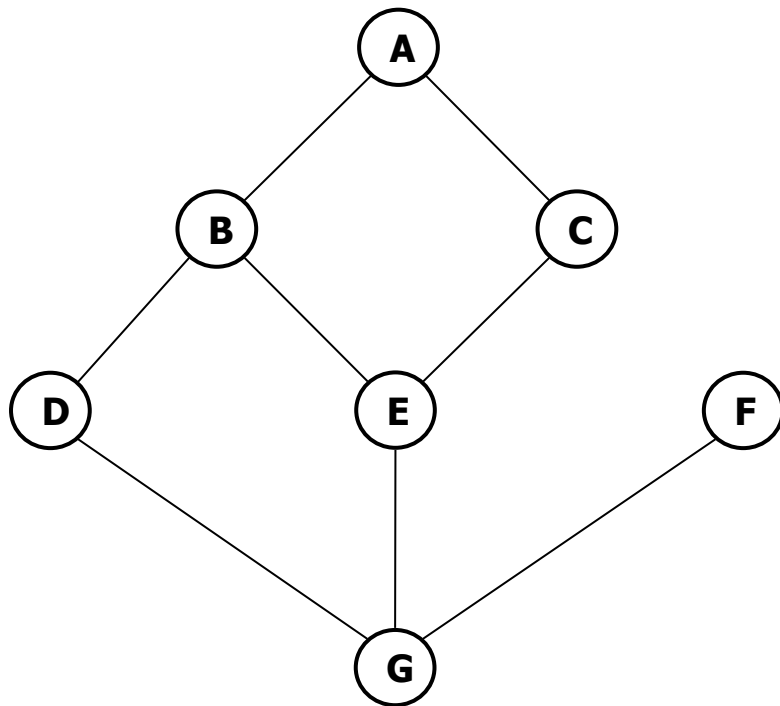
- 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문한다.

- 그래프 순회의 두 가지 방법

- 깊이 우선 탐색(DFS, Depth First Search)
 - 너비 우선 탐색(BFS, Breadth First Search)

그래프 순회 (2/2)

- 그래프 순회: 구현 및 동작과정



```
Microsoft Visual Studio 디버그 x + v

##### 그래프(G9): 인접 리스트 #####

정점 A의 인접 리스트 B ->> C ->> NULL
정점 B의 인접 리스트 A ->> D ->> E ->> NULL
정점 C의 인접 리스트 A ->> E ->> NULL
정점 D의 인접 리스트 B ->> G ->> NULL
정점 E의 인접 리스트 B ->> C ->> G ->> NULL
정점 F의 인접 리스트 G ->> NULL
정점 G의 인접 리스트 D ->> E ->> F ->> NULL

##### 그래프(G9): 깊이 우선 탐색(DFS) #####

A B D G E C F

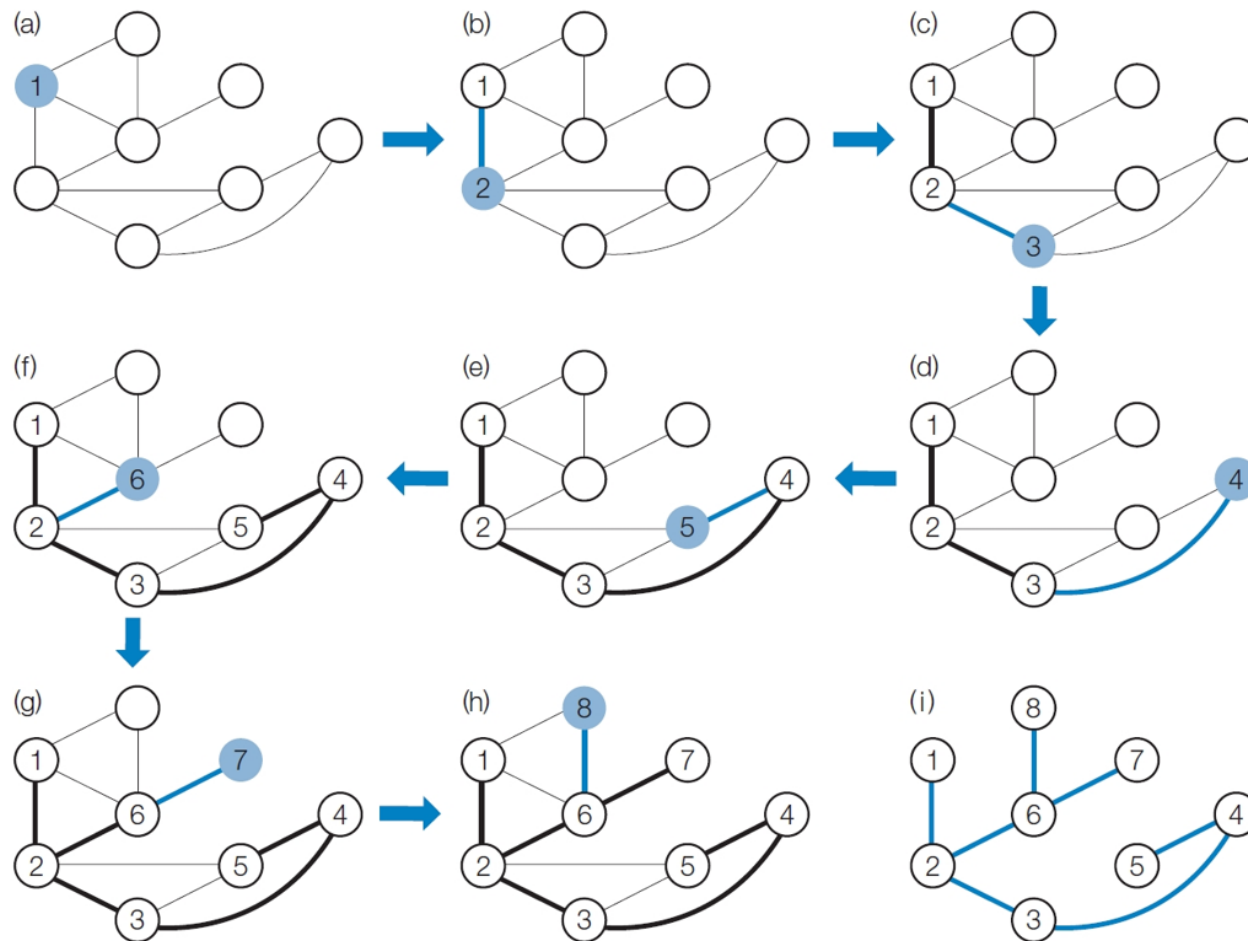
##### 그래프(G9): 너비 우선 탐색(BFS) #####

A B C D E G F
```

[그래프 G9 와 그래프 순회(DFS, BFS)]

그래프 순회: 깊이 우선 탐색

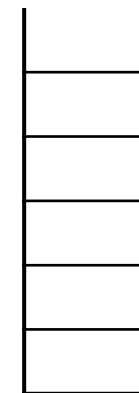
● 깊이 우선 탐색: 동작 과정



정점 방문 여부: T/F

A	B	C	D	E	F	G	H
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
F	F	F	F	F	F	F	F

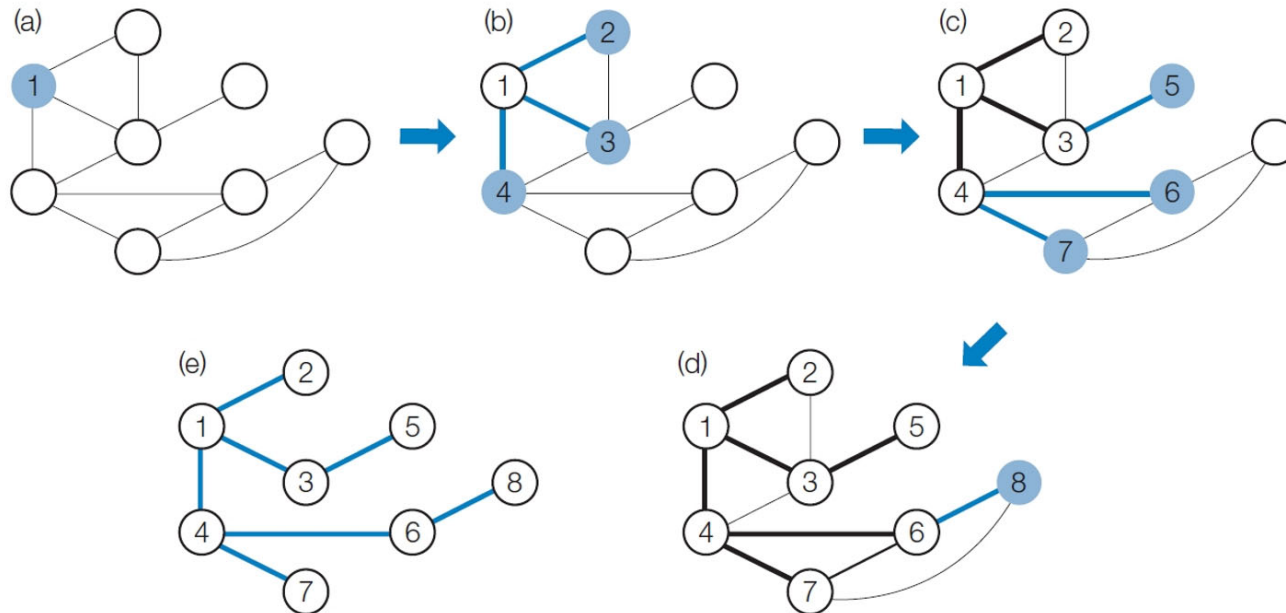
Visited



Stack

그래프 순회: 너비 우선 탐색

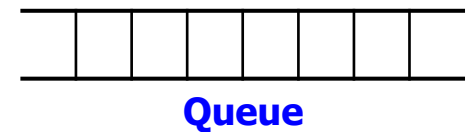
- 너비 우선 탐색: 동작 과정



정점 방문 여부: T/F

A	B	C	D	E	F	G	H
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
F	F	F	F	F	F	F	F

Visited



그래프 알고리즘

최소 신장 트리

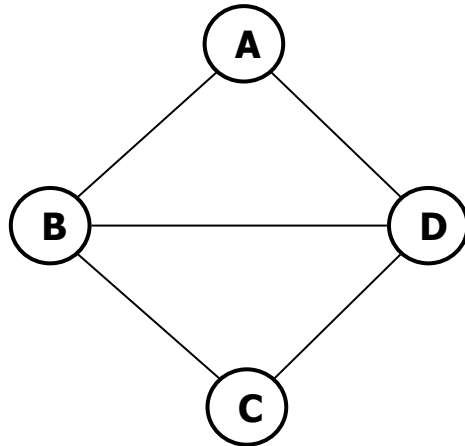


신장 트리 (1/3)

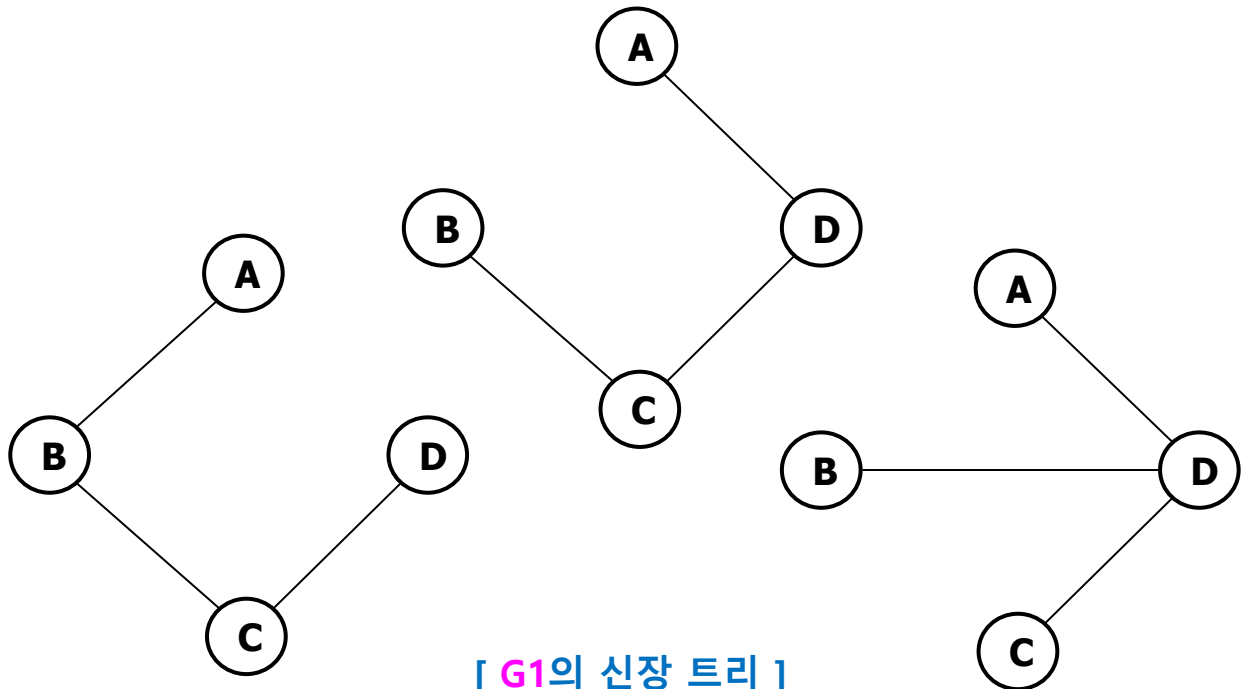
- **신장 트리**(Spanning Tree)

- N 개의 정점으로 이루어진 무향 그래프 G 에서 N 개의 모든 정점과 $N-1$ 개의 간선으로 만들어진 트리

- 그래프의 관점에서 트리는 사이클이 없는 단순 연결 그래프



[그래프 **G1**]



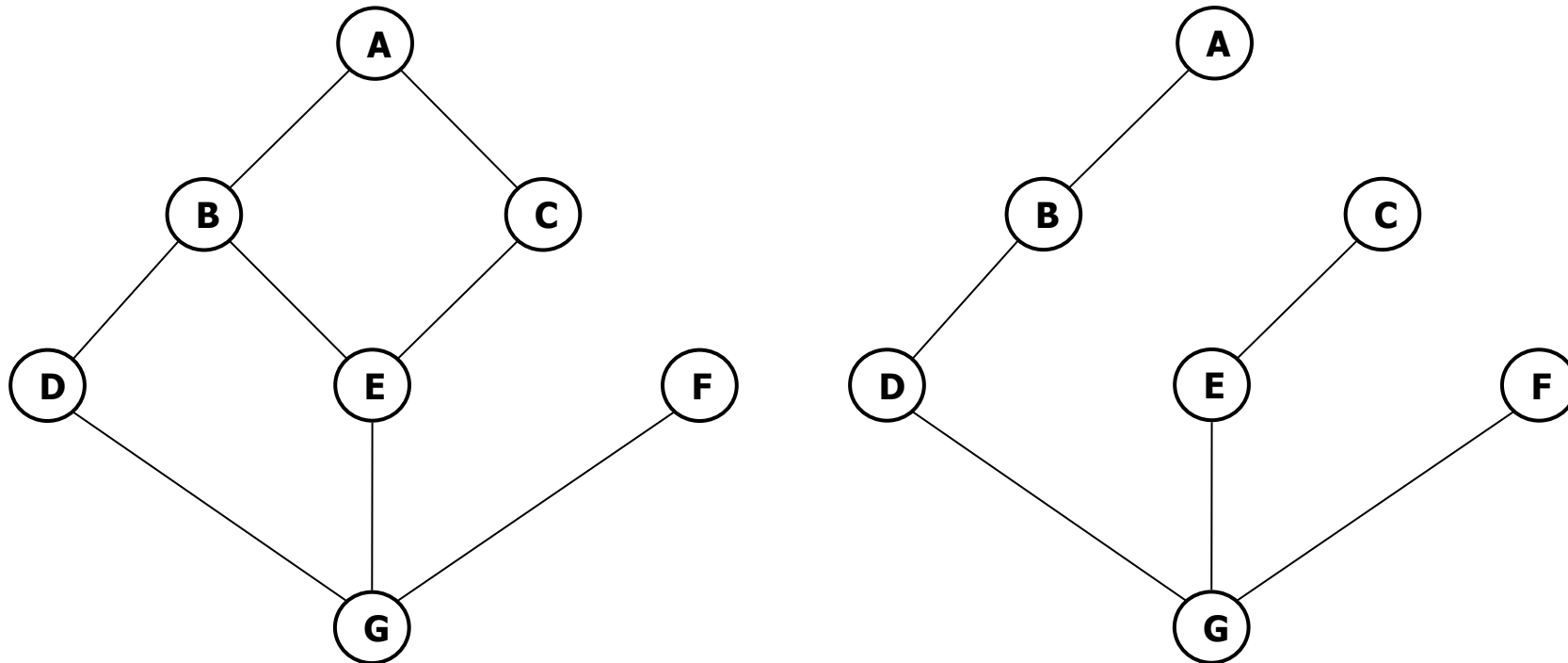
[**G1**의 신장 트리]

신장 트리 (2/3)

- 신장 트리: 깊이 우선 신장 트리

- 깊이 우선 신장 트리 (Depth First Spanning Tree)

- 깊이 우선 탐색을 이용하여 생성된 신장 트리



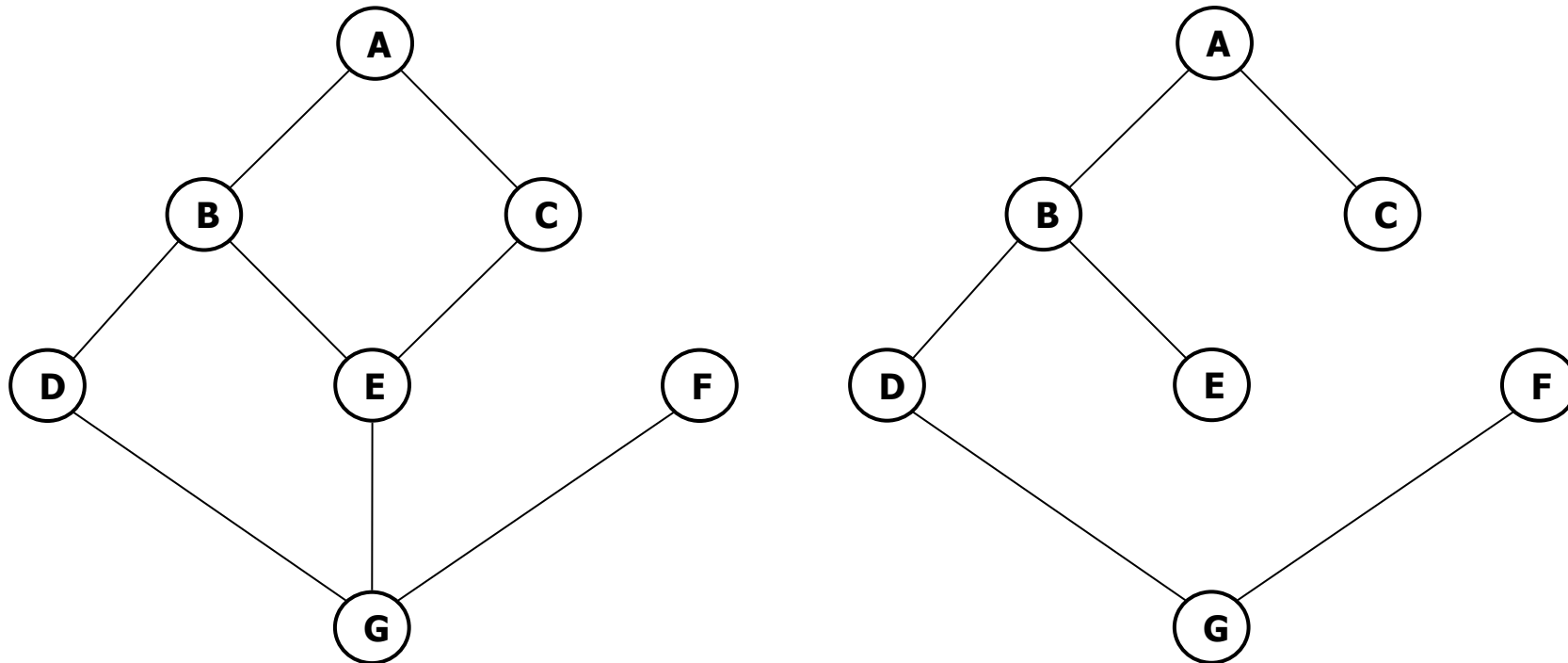
[그래프 G9 와 깊이 우선 신장 트리]

신장 트리 (3/3)

- 신장 트리: 너비 우선 신장 트리

- 너비 우선 신장 트리 (Breadth First Spanning Tree)

- 너비 우선 탐색을 이용하여 생성된 신장 트리



[그래프 G와 너비 우선 신장 트리]

최소 신장 트리

- **최소 신장 트리**(Minimum Spanning Tree)

- 무향 가중치 그래프에서 신장 트리를 구성하는 간선들의 가중치 합이 최소인 신장 트리

- 신장 트리의 비용(Cost of a Spanning Tree)
 - 신장 트리를 구성하는 간선 가중치의 합
 - 최소 신장 트리: 비용을 최소로 하는 신장 트리
- 가중치 그래프의 간선에 주어진 가중치
 - 비용이나 거리, 시간을 의미하는 값

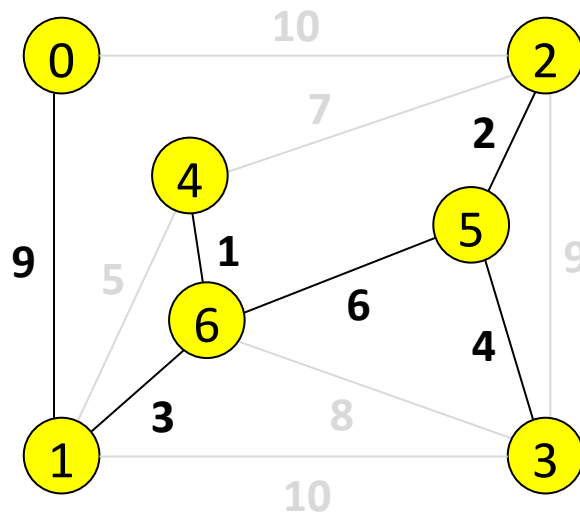
- **최소 신장 트리 생성 알고리즘**

- Kruskal 알고리즘
- Prim 알고리즘

최소 신장 트리: Kruskal 알고리즘 (1/2)

● Kruskal 알고리즘: 동작 과정

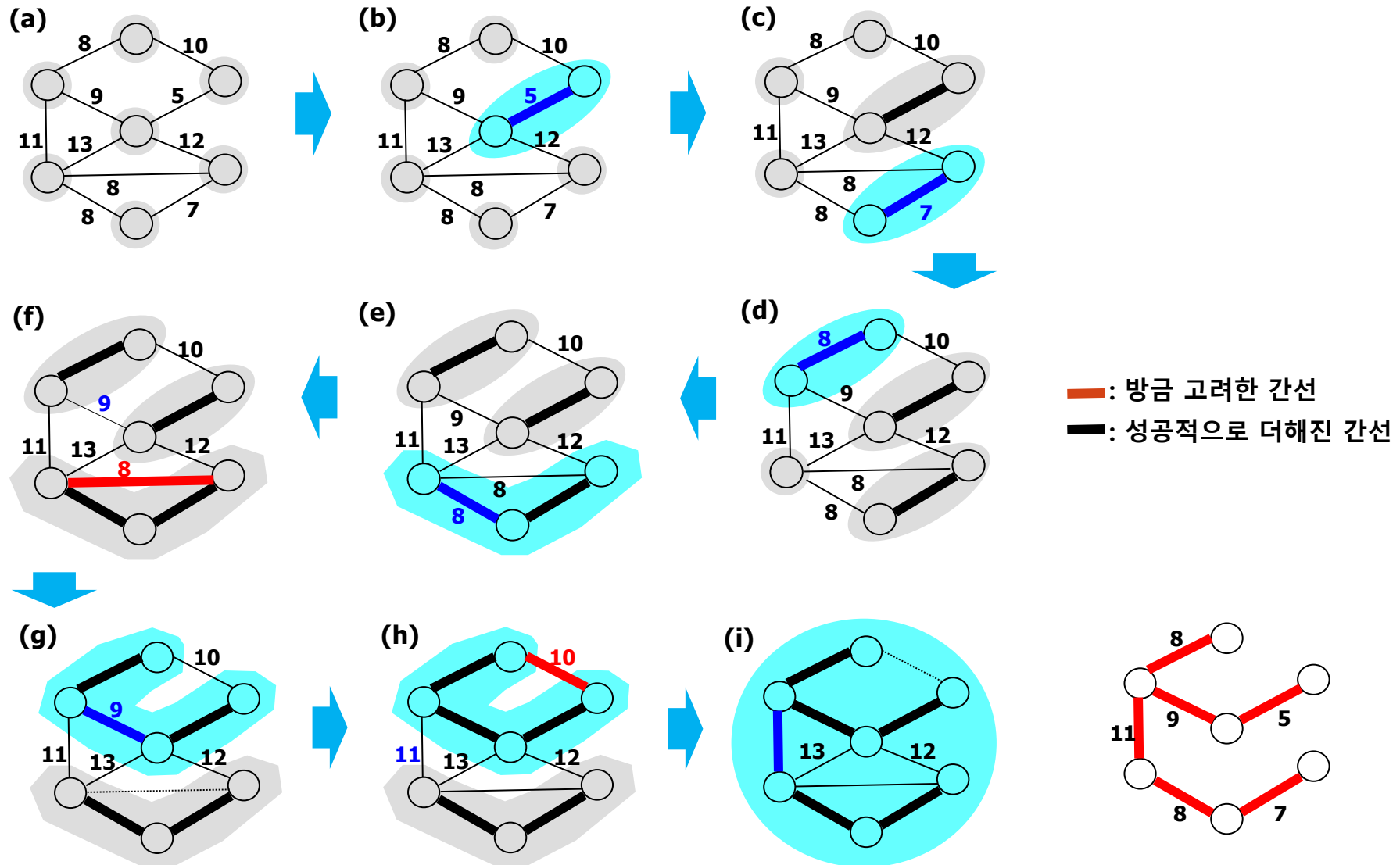
정렬된 L



(0, 1) 9
(0, 2) 10
(1, 3) 10
(1, 4) 5
(1, 6) 3
(2, 3) 9
(2, 4) 7
(2, 5) 2
(3, 5) 4
(3, 6) 8
(4, 6) 1
(5, 6) 6

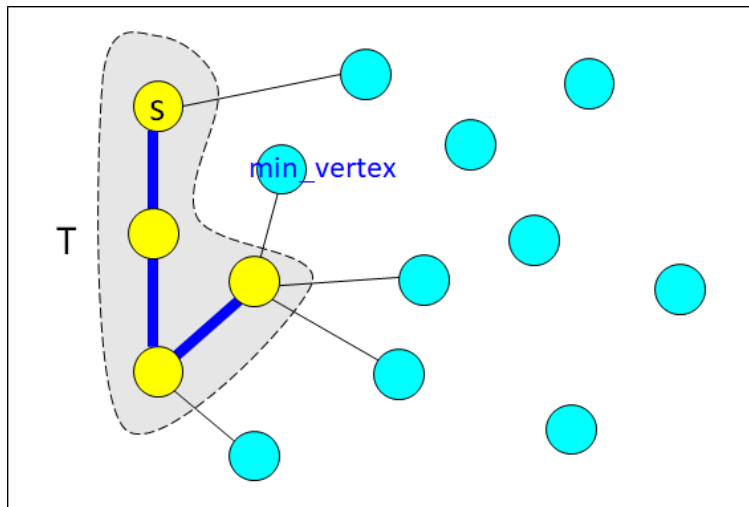
(4, 6) 1
(2, 5) 2
(1, 6) 3
(3, 5) 4
~~(1, 4) 5~~
(5, 6) 6
~~(2, 4) 7~~
~~(3, 6) 8~~
(0, 1) 9
~~(2, 3) 9~~
~~(0, 2) 10~~
(1, 3) 10

최소 신장 트리: Kruskal 알고리즘 (2/2)

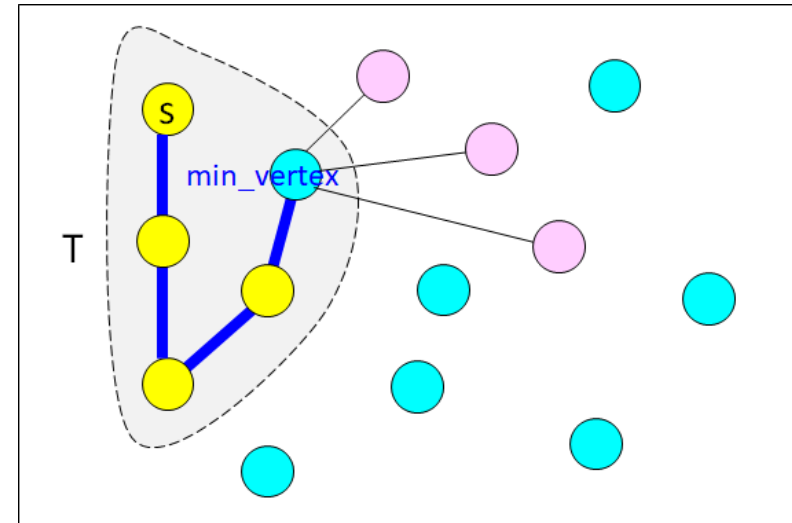


최소 신장 트리: Prim 알고리즘 (1/3)

● Prim 알고리즘: 동작 과정



(a)



(b)

- (a) 트리에 가장 가까운 정점 **minVertex** 를 찾아...
 - 트리 밖에 있는 정점들의 **D** 의 원소들 중에서 최솟값을 찾아...
- (b) 트리에 추가한 후, 정점 **minVertex**에 인접하면서 트리에 속하지 않은 각 정점의 **D** 원소가 이전 값보다 작으면 갱신

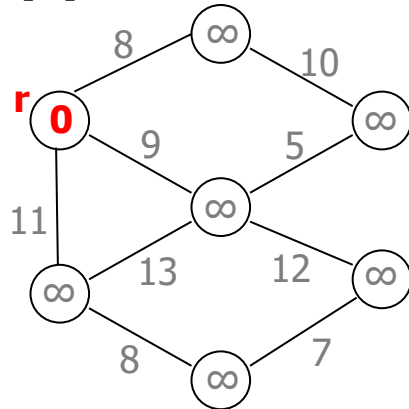
● : 방금 S 에 포함된 정점

최소 신장 트리: Prim 알고리즘 (2/3)

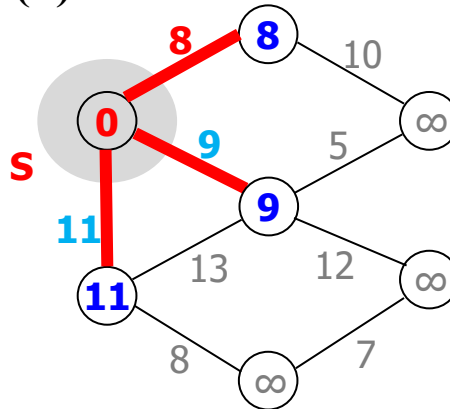
● : 방금 이완이 일어난 정점

● Prim 알고리즘: 동작 과정

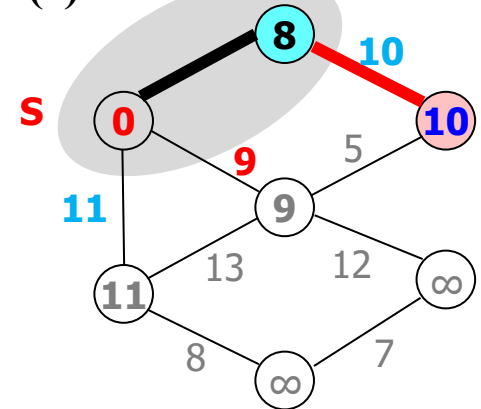
(a)



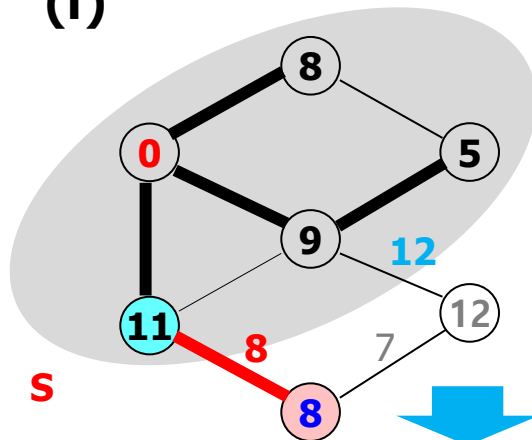
(b)



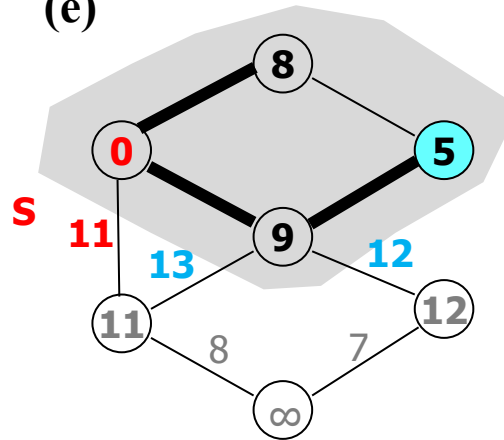
(c)



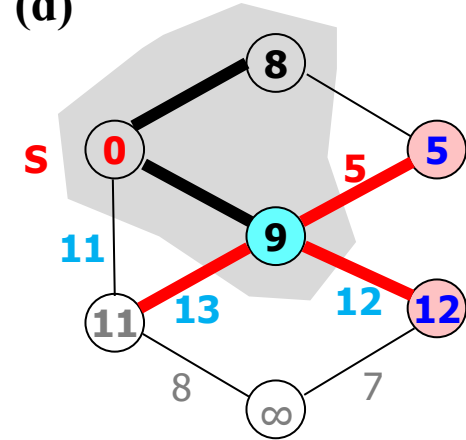
(f)



(e)



(d)

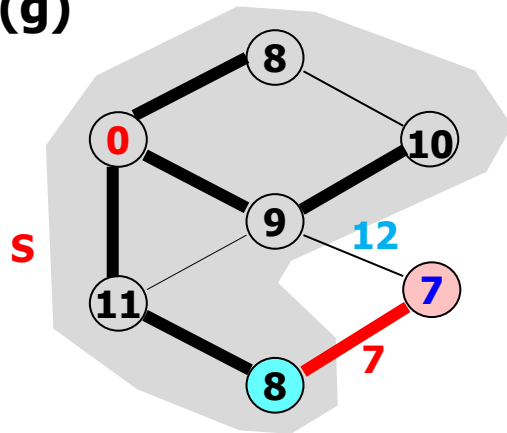


최소 신장 트리: Prim 알고리즘 (3/3)

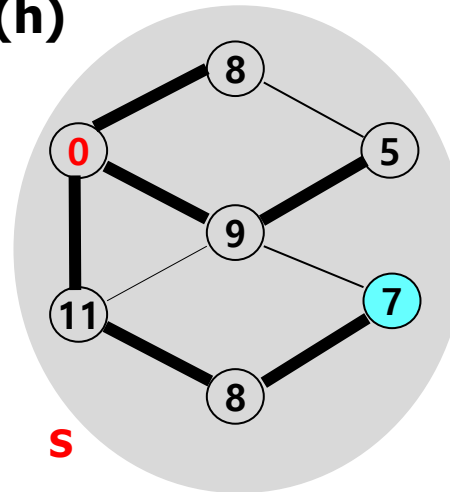
- Prim 알고리즘: 동작 과정



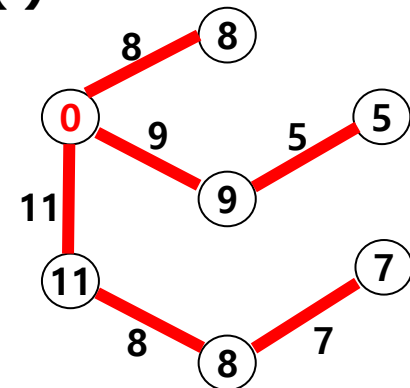
(g)



(h)



(i)



● : 방금 S 에 포함된 정점

● : 방금 이완이 일어난 정점



탐욕 알고리즘



백문이불여일타(百聞而不如一打)

- 그래프 알고리즘

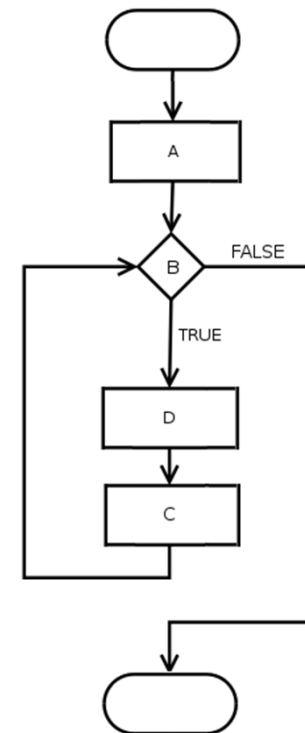
- 탐욕 알고리즘

 - 최적해

- 동적 프로그래밍

- 고급 알고리즘 설계와 분석

for(A;B;C)
D;



탐욕 알고리즘

● 탐욕 알고리즘(Greedy Algorithms)

○ 눈앞의 이익만 취하고 보는 알고리즘

- 현재 시점에 가장 이익이 되어 보이는 해를 선택하는 행위를 반복한다.
- 대부분 최적해와의 거리가 멀다(드물게 최적해가 보장되는 경우도 있다).

```
do {  
    우선 가장 좋아 보이는 선택을 한다.  
} until (해 구성 완료)
```

```
// 탐욕 알고리즘의 전형적인 구조  
Greedy(C)           // C : 원소들의 총집합  
{  
    S ← ∅;  
    while (C ≠ ∅ and S 는 아직 온전한 해가 아님) {  
        x ← C 에서 가장 좋아 보이는 원소;  
        집합 C 에서 x 제거;    // C ← C - {x}  
        if (S 에 x 를 더해도 됨) then S ← S ∪ {x};  
    }  
    if (S가 온전한 해임) then return S;  
    else return "no solution!";  
}
```

탐욕 알고리즘: 최적해 (1/4)

● 최적해가 보장되는 예: 회의실 배정 문제

○ 회의실 배정 문제

- 회사에 회의실이 1개 있다.
- 여러 부서에서 회의실을 사용하므로 미리 신청을 받아 스케줄링을 한다.
 - 회의 시작 시간과 종료 시간을 명시하여 신청한다.
- 목표: 겹치는 회의가 없게 하면서 가장 많은 수의 회의를 소화할 수 있도록 한다.

• 탐욕스러운(Greedy) 아이디어들

- 소요 시간이 가장 짧은 회의순으로 배정
- 시작 시간이 가장 이른 회의순으로 배정
- 종료 시간이 가장 이른 회의순으로 배정

이것 만이 최적해를 보장한다.

• 예: 8개의 회의가 신청된 상황

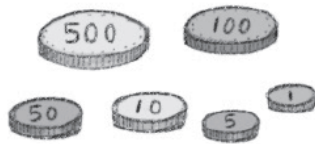
- (3, 5), (1, 6), (6, 7), (5, 9), (8, 13), (7, 14), (12, 18), (16, 20)

탐욕 알고리즘: 최적해 (2/4)

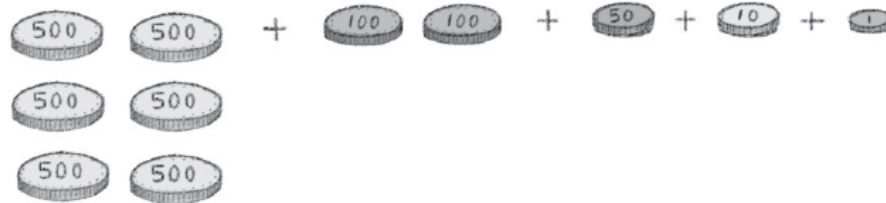
- **최적해가 보장되지 않는 예: 동전 바꾸기**

- 동전 바꾸기: 최적해를 보장하는 예

동전의 액면



3,256원 만들기



동전 바꾸기

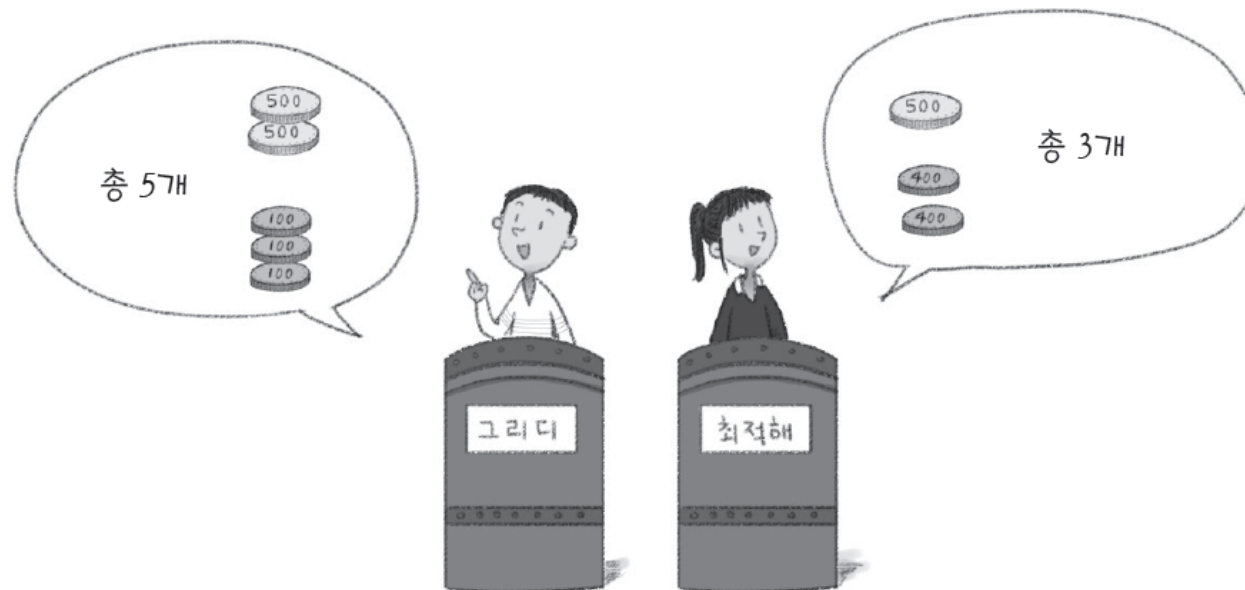
총 11개

이렇게 동전의 액면이 모두 바로 아래 액면의 배수가 되면
그리디 알고리즘으로 최적해가 보장된다

탐욕 알고리즘: 최적해 (3/4)

- **최적해가 보장되지 않는 예: 동전 바꾸기**

- 동전 바꾸기: 최적해를 보장하지 않는 예



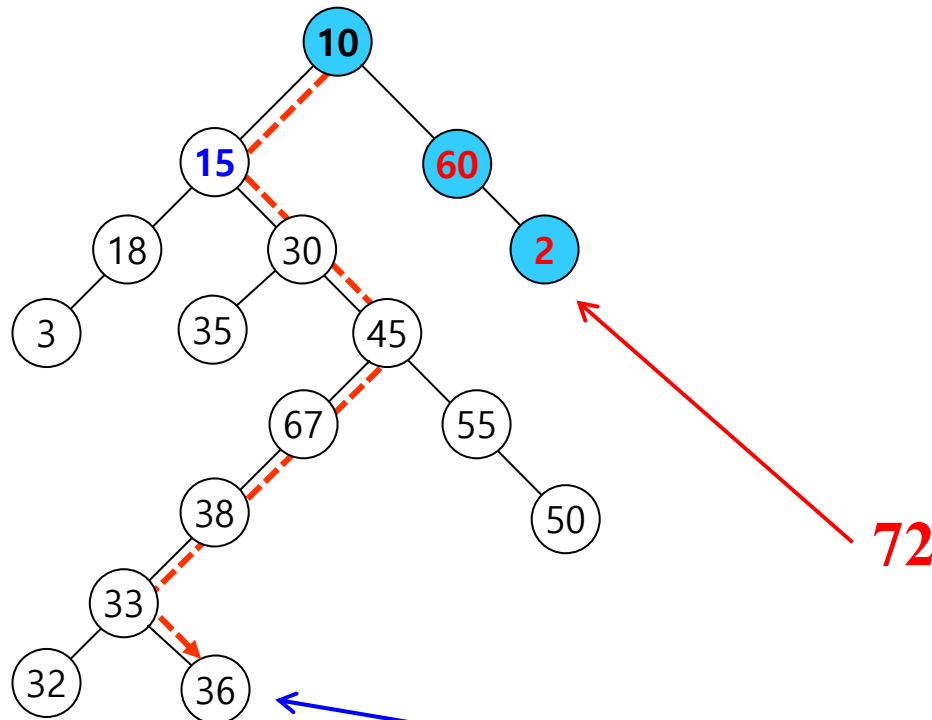
동전 액면이 바로 아래 액면의 배수가 되지 않으면,
탐욕 알고리즘으로 최적해가 보장되지 않는다.

탐욕 알고리즘: 최적해 (4/4)

● 최적해가 보장되지 않는 예: 이진 트리의 최적 합 경로 찾기

○ 이진 트리의 최적 합 경로 찾기

- 루트부터 시작해 왼쪽으로 분기할지 오른쪽으로 분기할지를 매 단계마다 결정한다.
 - 임의의 노드에 이르면 그제서야 그 노드의 자식에 할당된 가중치가 공개된다.
 - 이렇게 내려가다 단말 노드를 만나면 끝난다.



탐욕 알고리즘이 실패한 예

동적 프로그래밍



- 그래프 알고리즘

백문이불여일타(百聞而不如一打)

- 탐욕 알고리즘

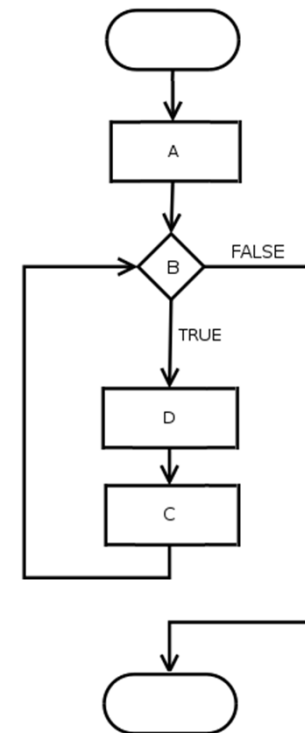
- 동적 프로그래밍

 - 행렬 경로

 - 최단 경로

- 고급 알고리즘 설계와 분석

for(A;B;C)
D;



동적 프로그래밍 (1/5)

● 재귀적 해법

- 큰 문제에 닮은 꼴의 작은 문제가 깃든다.
- 잘 쓰면 보약, 잘못 쓰면 맹독
 - 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다.
 - 재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다.
- 재귀적 해법이 바람직한 예
 - 계승(factorial) 구하기
 - 퀵 정렬, 병합 정렬 등의 정렬 알고리즘
 - 그래프의 깊이 우선 탐색(DFS, Depth First Search)
- 재귀적 해법이 치명적인 예
 - 피보나치 수 구하기
 - 행렬 곱셈 최적순서 구하기

동적 프로그래밍 (2/5)

- 동적 프로그래밍의 적용 조건

- 최적 부분 구조(Optimal Substructure)

- 큰 문제의 해답에 그보다 작은 문제의 해답이 포함되어 있다.
 - 최적 부분 구조를 가진 문제의 경우에는 재귀 호출을 이용하여 문제를 풀 수 있다.

- 재귀 호출 시 중복(overlapping recursive calls)

- 재귀적으로 구현했을 때 중복 호출로 심각한 비효율이 발생한다.

동적 프로그래밍이 그 해결책 !!!

- 위의 두 성질이 있는 문제에 대해 적절한 저장 방법으로 중복 호출의 비효율을 제거한 것을 동적 프로그래밍이라고 한다.

동적 프로그래밍 (3/5)

● 피보나치 수열

○ 피보나치(Fibonacci)

- 1,200년 경에 활동한 이탈리아 수학자

“아주 간단한 문제지만...

동적 프로그래밍의 동기와 구현이 다 포함되어 있다.”

“토끼 한 마리가 매년 새끼 한 마리를 낳는다.
새끼는 한 달 후부터 새끼를 낳기 시작한다.
최초 토끼 한 마리가 있다고 하면...
한 달 후에 토끼는 두 마리가 되고 두 달 후에는 세 마리가 되고...”

// 피보나치 수열: 재귀적 용법

Fibonacci(num)

{

if (num = 1 or num = 2)
then return 1;

else

return (**Fibonacci**(num - 1) + **Fibonacci**(num - 2));

}

$$f_n = f_{n-1} + f_{n-2} (n \geq 3)$$

$$f_1 = f_2 = 1 (n = 1, 2)$$

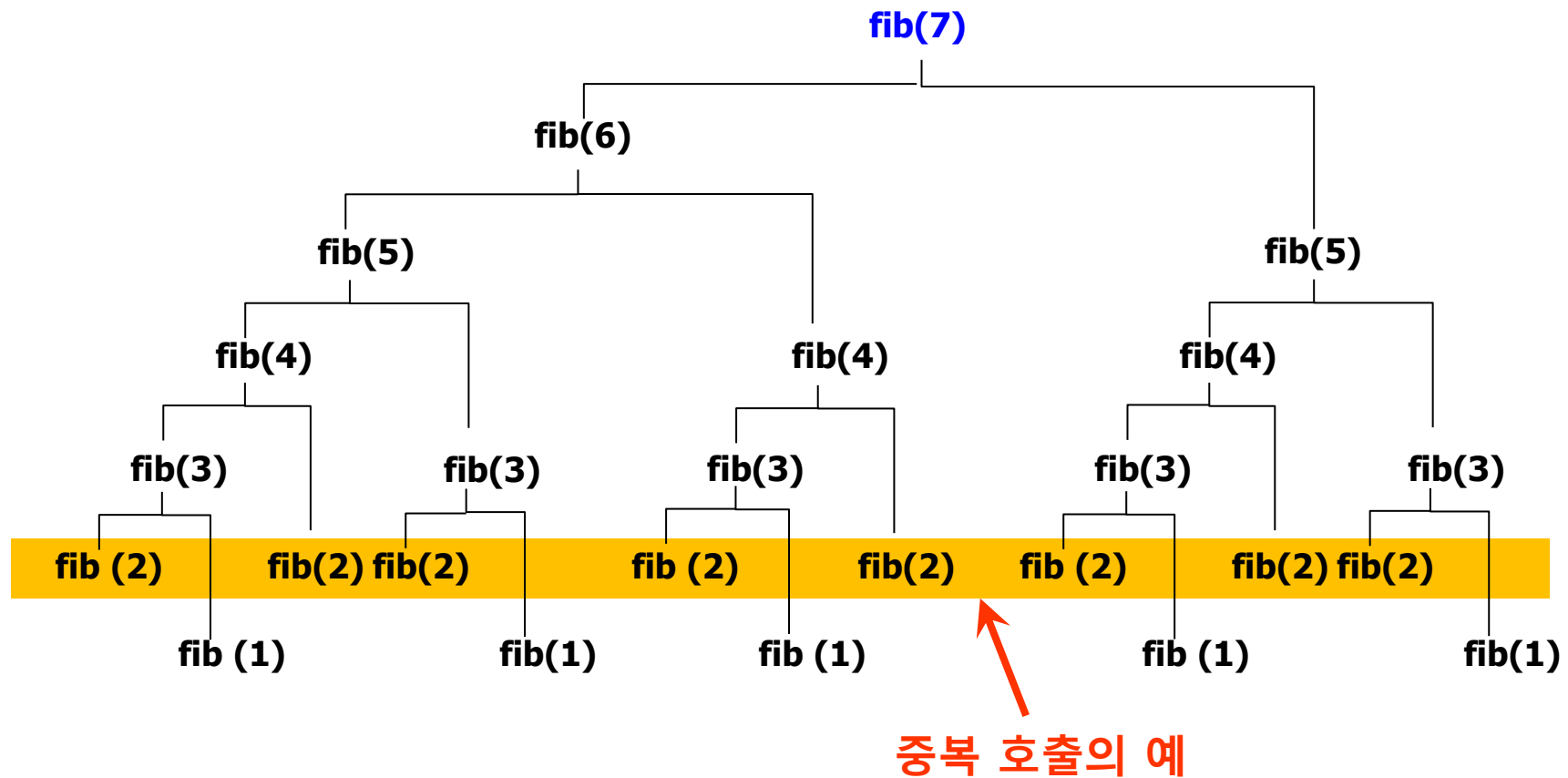
“엄청난 중복 호출이 존재한다.”

--> 재귀적 알고리즘은 **지수함수에 비례**하는 시간이 든다.

동적 프로그래밍 (4/5)

- 피보나치 수열: 재귀적 용법

- 문제점: 중복 호출!!!



동적 프로그래밍 (5/5)

- 피보나치 수열: 동적 프로그래밍 알고리즘

Fibonacci(n)

{

$f[1] \leftarrow f[2] \leftarrow 1;$

for $i \leftarrow 3$ **to** n

$f[i] \leftarrow f[i - 1] + f[i - 2];$

return $f[n];$

}

fibonacci

1	1	?	?	?	?	?
---	---	---	---	---	---	---

$\Theta(n)$

Fibonacci(n)

{

$first \leftarrow second \leftarrow 1;$

for $i \leftarrow 3$ **to** n

$res \leftarrow first + second;$

return $res;$

}

동적 프로그래밍

행렬 경로



동적 프로그래밍: 행렬 경로 (1/5)

● 행렬 경로

○ 양수 원소들로 구성된 $N * N$ 행렬

- 행렬의 좌 상단에서 시작하여 우 하단까지 이동한다.
- 방문한 칸에 있는 수들을 더한 값이 최소화(최대화) 되도록 한다.
- 이동 방법(제약 조건)
 - 오른쪽이나 아래쪽으로만 이동할 수 있다.
 - 왼쪽, 위쪽, 대각선 이동은 허용하지 않는다.

○ 유효한 이동

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

동적 프로그래밍: 행렬 경로 (2/5)

- 행렬 경로

- 불법 이동의 예

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

불법 이동 (상향)

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

불법 이동 (좌향)

동적 프로그래밍: 행렬 경로 (3/5)

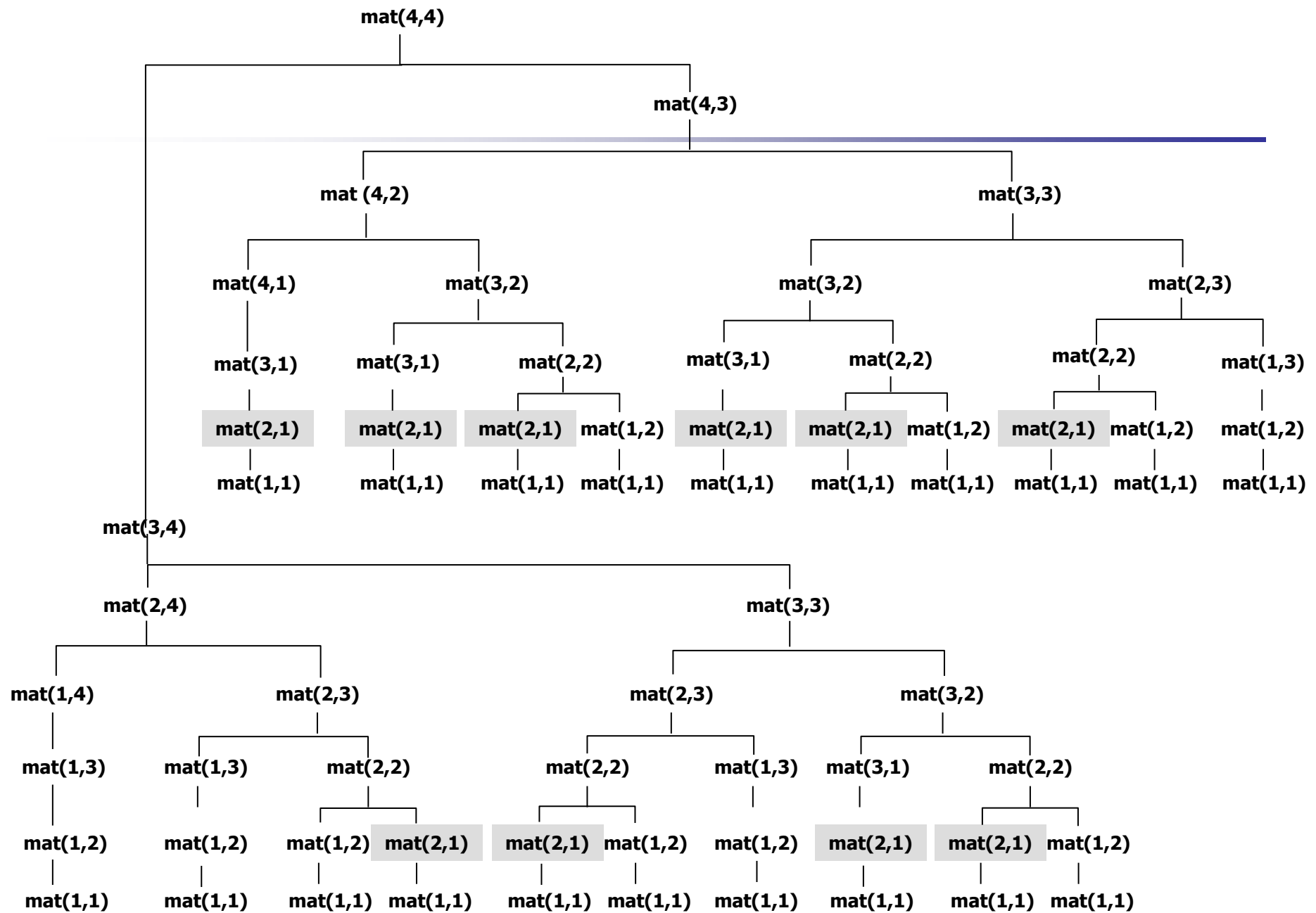
- 행렬 경로: 재귀적 용법

- 시간 복잡도: $O(n * n) = O(n^2)$

- 행렬의 크기가 커질수록 중복 호출이 매우 큰 횟수로 발생

```
matrixPath(i, j)
// (i, j)에 이르는 최고 점수
{
    if (i = 0 or j = 0) then
        return 0;
    else
        return ( mij + ( max(matrixPath(i-1, j), matrixPath(i, j-1)) ) );
}
```

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9



동적 프로그래밍: 행렬 경로 (5/5)

- 행렬 경로: 동적 프로그래밍

matrixPath(n)

// (n, n)에 이르는 최고 점수

// C_{ij} -- 원소(1,1)에서(i, j)에 이르는 최댓값

{

 for i ← 0 to n

$c[i, 0] \leftarrow 0;$

 for j ← 1 to n

$c[0, j] \leftarrow 0;$

 for i ← 1 to n

 for j ← 1 to n

$c[i, j] \leftarrow m_{ij} + \max(c[i-1, j], c[i, j-1]);$

 return $c[n, n];$

}

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

$\Theta(n^2)$

동적 프로그래밍

최단 경로



최단 경로

- **최단 경로**(Shortest Paths)

- **조건: 간선의 가중치가 있는 유향 그래프**

- 무향 그래프는 각 간선에 대해 양쪽으로 유향 간선이 있는 그래프로 생각.
 - 즉, 무향 간선 (u, v) 는 유향 간선 (u, v) 와 (v, u) 를 의미한다고 가정하면 된다.

- **두 정점 사이의 최단경로**

- 두 정점 사이의 경로들 중 간선의 가중치 합이 최소인 경로
 - 간선 가중치의 합이 음인 사이클이 있으면 문제가 정의되지 않는다.

- **단일 시작점 최단경로**

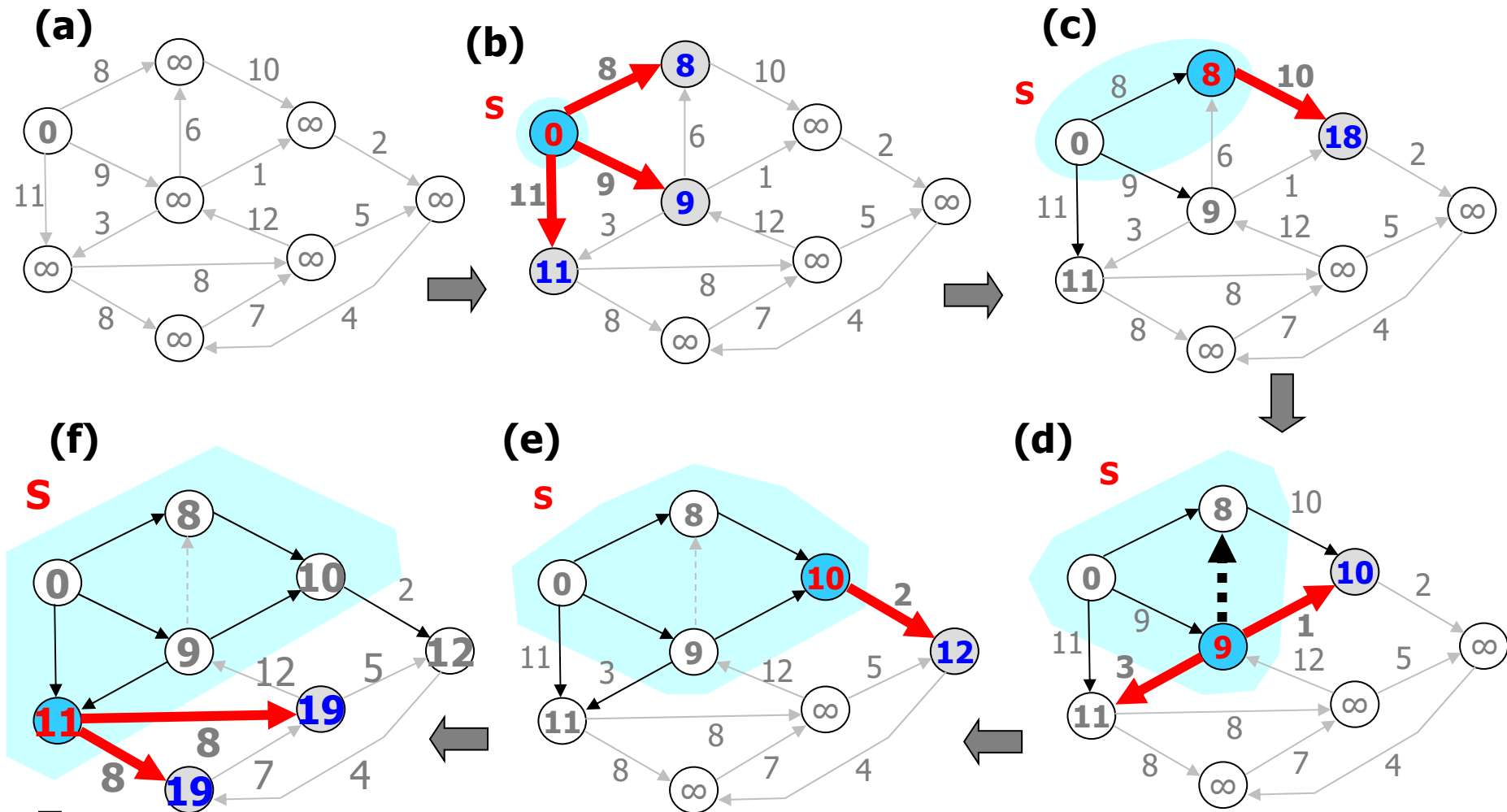
- 단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다.
 - **Dijkstra 알고리즘**: 음의 가중치를 허용하지 않는 최단 경로
 - **Bellman-Ford 알고리즘**: 음의 가중치를 허용하는 최단 경로
 - 순환이 없는 유향 그래프(DAG)의 최단 경로

- **모든 쌍 최단경로**

- 모든 정점 쌍 사이의 최단경로를 모두 구한다.
 - **Floyd-Warshall 알고리즘**

최단 경로: Dijkstra 알고리즘 (1/3)

● Dijkstra 알고리즘: 동작 과정 #1

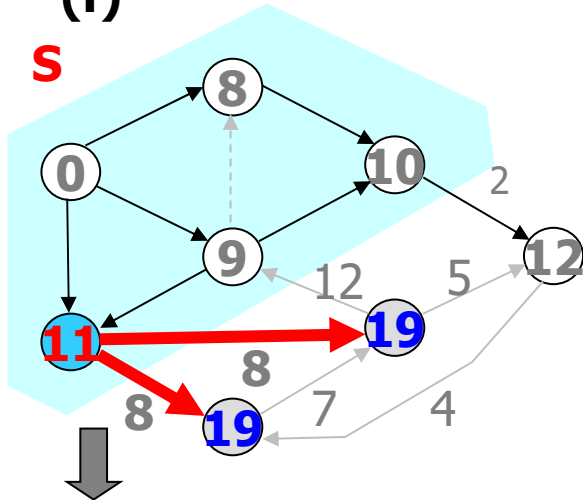


최단 경로: Dijkstra 알고리즘 (2/3)

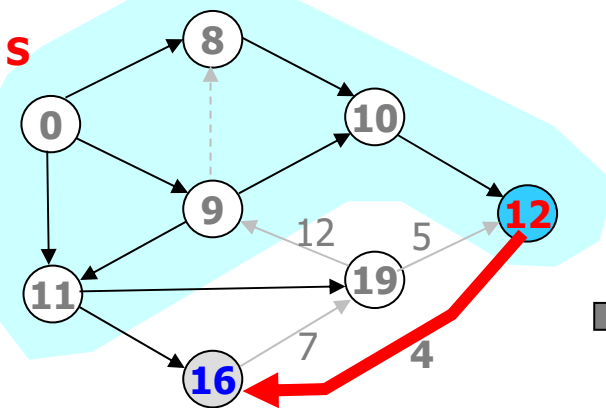


(f)

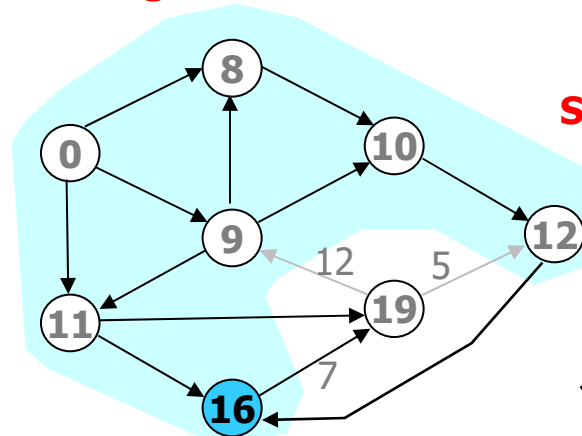
● Dijkstra 알고리즘: 동작 과정 #2



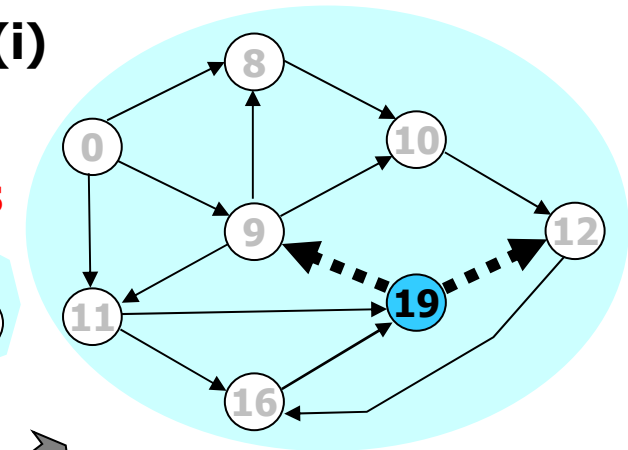
(g)



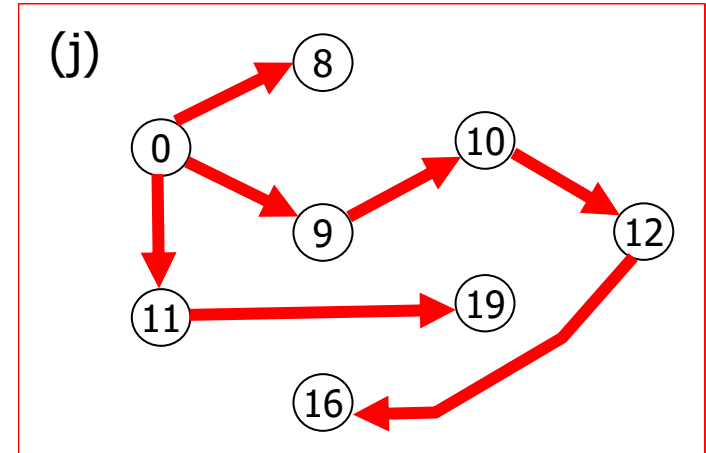
(h)



(i)

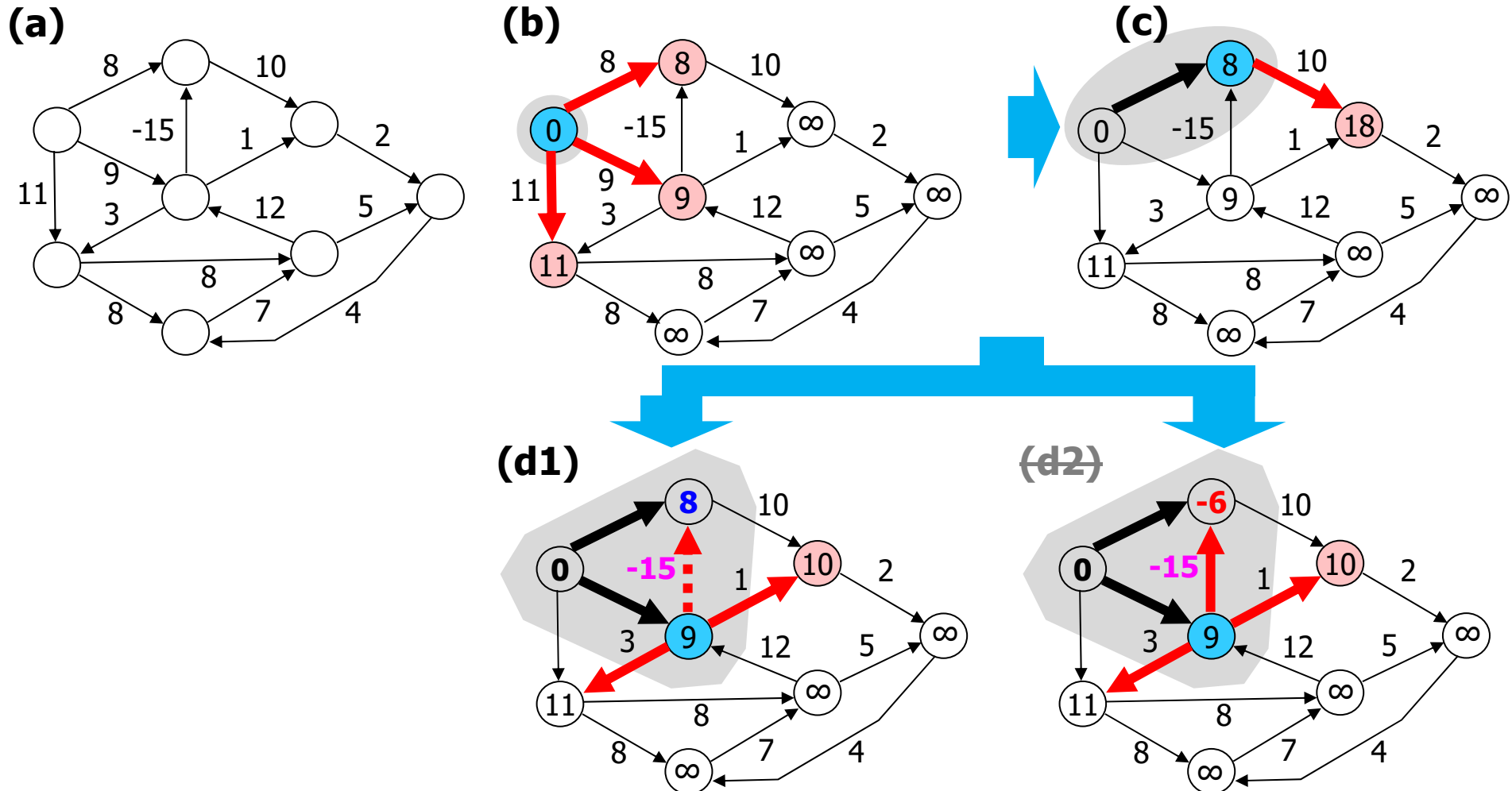


(j)



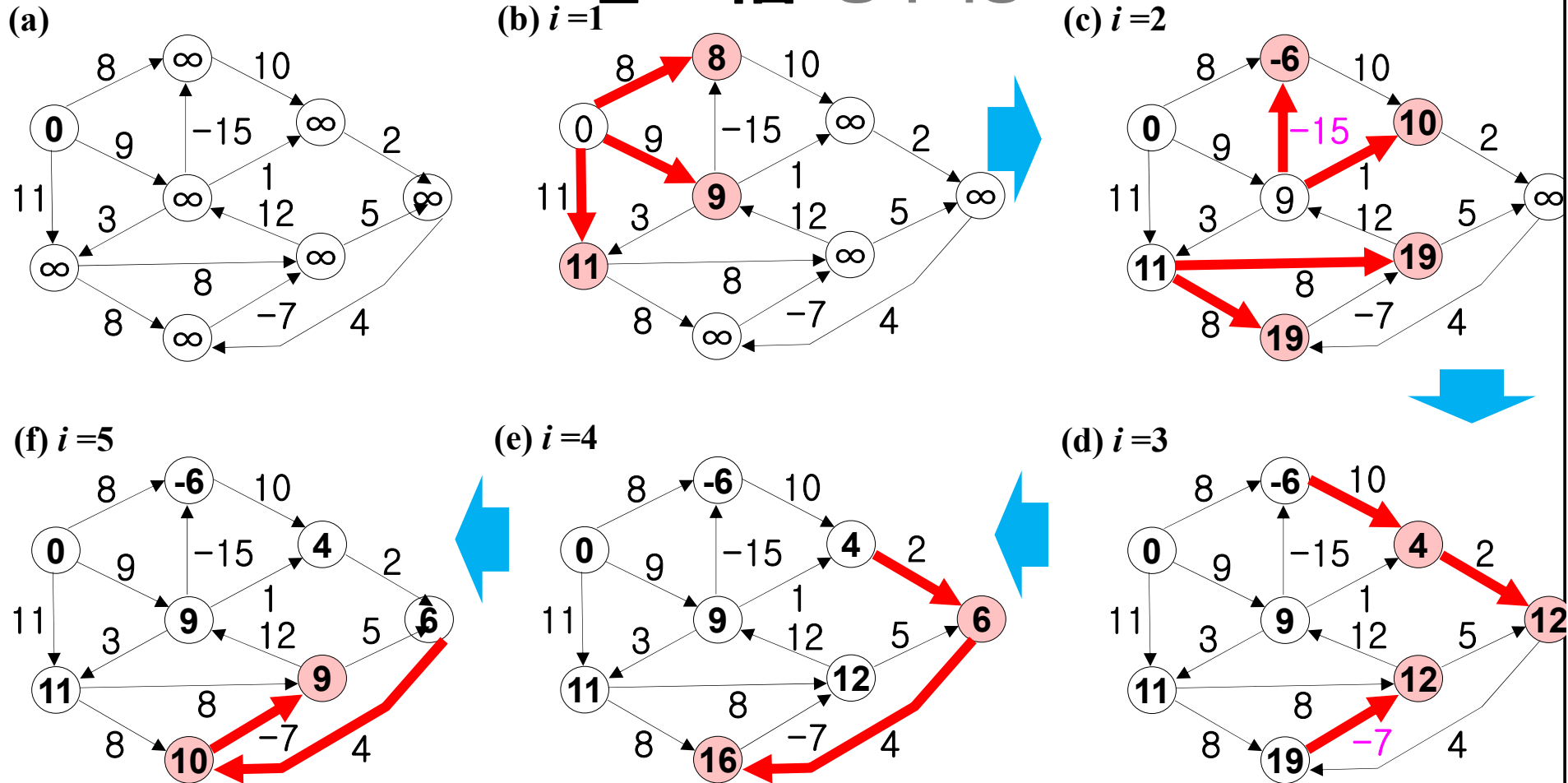
최단 경로: Dijkstra 알고리즘 (3/3)

- Dijkstra 알고리즘: 알고리즘이 작동하지 않는 예



최단 경로: Bellman-Ford 알고리즘 (1/4)

● Bellman-Ford 알고리즘: 동작 과정 #1

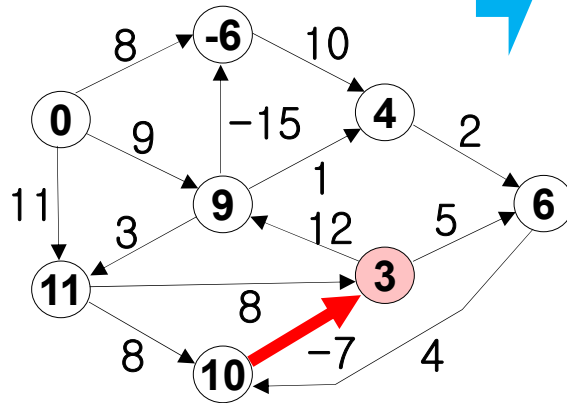


최단 경로: Bellman-Ford 알고리즘 (2/4)

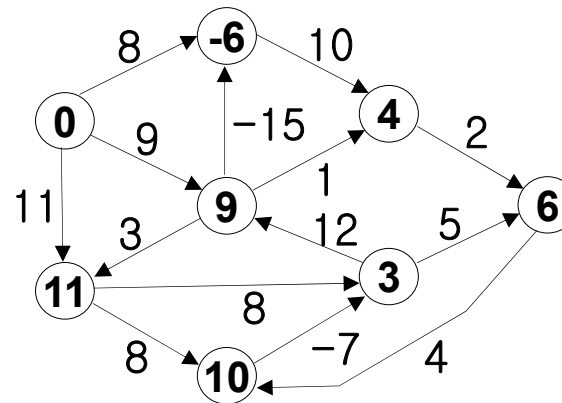
- Bellman-Ford 알고리즘: 동작 과정 #2



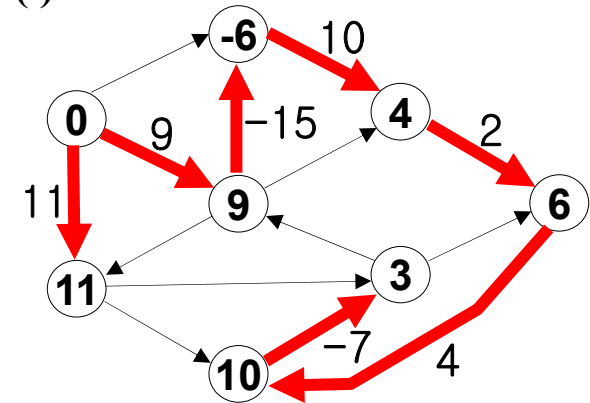
(g) $i=6$



(h) $i=7$

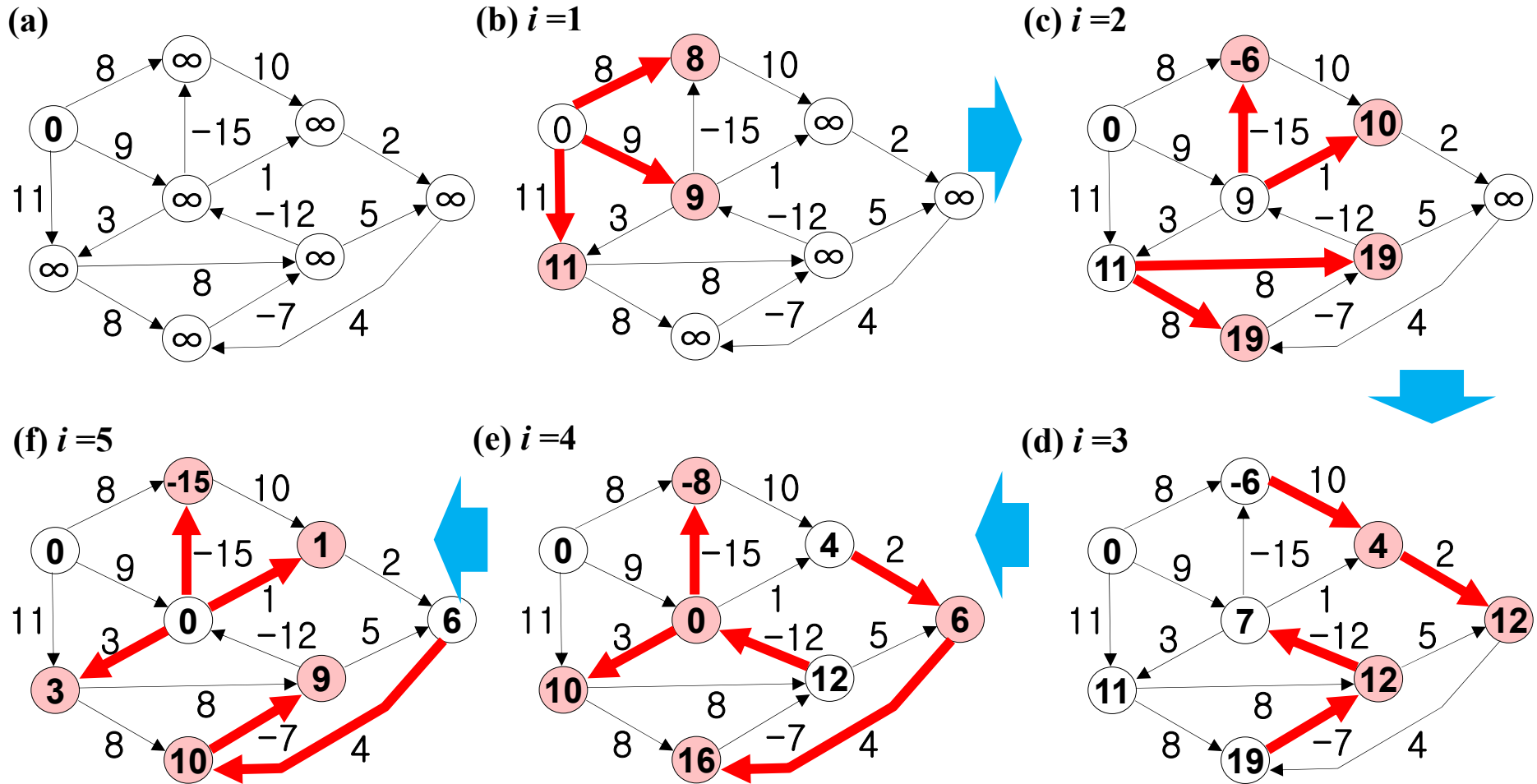


(i)



최단 경로: Bellman-Ford 알고리즘 (3/4)

● Bellman-Ford 알고리즘: 음의 사이클이 있는 경우 #1

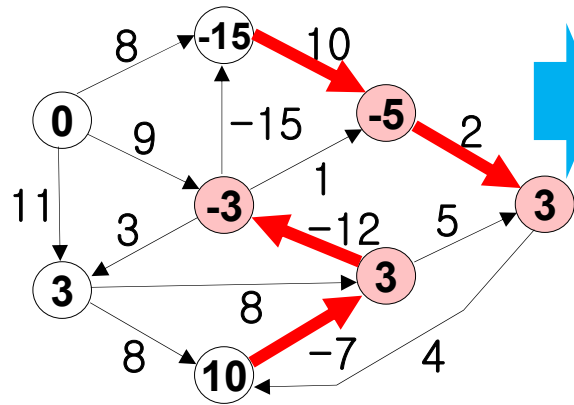


최단 경로: Bellman-Ford 알고리즘 (4/4)

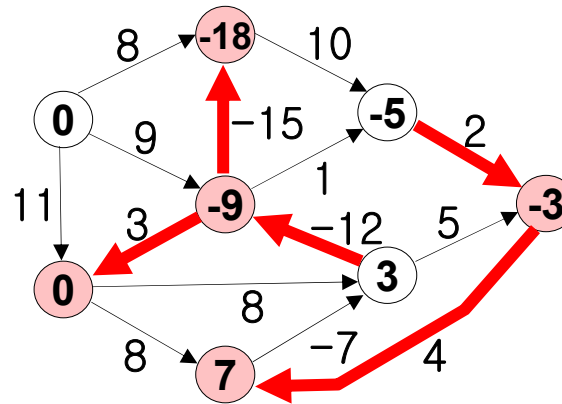
- Bellman-Ford 알고리즘: 음의 사이클이 있는 경우 #2



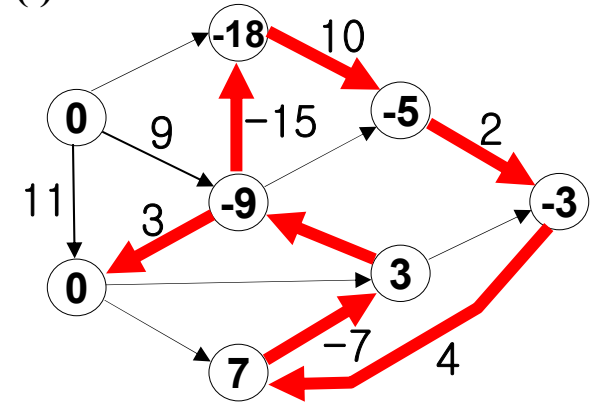
(g) $i=6$



(h) $i=7$

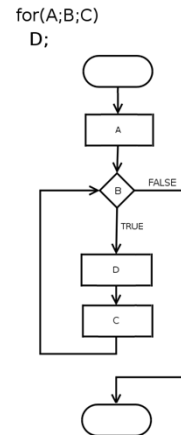


(i)



참고문헌

- [1] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [2] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛아카데미, 2022.
- [3] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.
- [5] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [6] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2024 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2024 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2024 of viewing the site, <https://programmers.co.kr/>.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

