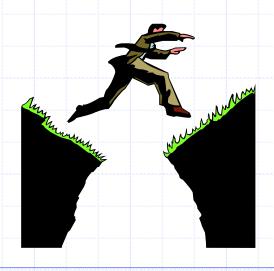
## 최단경로

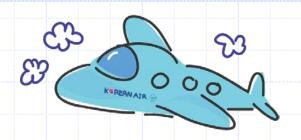


Algorithms 최단경로 1

#### Outline

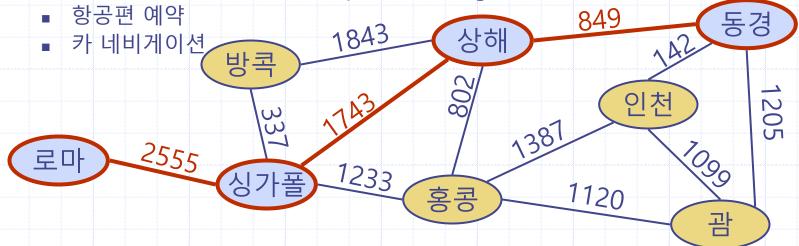
- ◆ 17.1 최단경로
  - ◈ 17.2 최단경로 알고리즘
  - ◈ 17.3 모든 쌍 최단경로
  - ◈ 17.4 응용문제

#### 최단경로 문제



- ◆ **최단경로 문제**(shortest path problem): 가중그래프와 두 개의 정점 u와 v가 주어졌을 때, u와 v 사이의 무게의 합이 최소인 경로를 구하는 문제
  - 최단경로길이: 간선들의 무게 합
- 예
  - 동경과 로마 사이의 최단경로
- ◈ 응용

■ 인터넷 패킷 라우팅(internet packet routing)



### 최단경로 속성

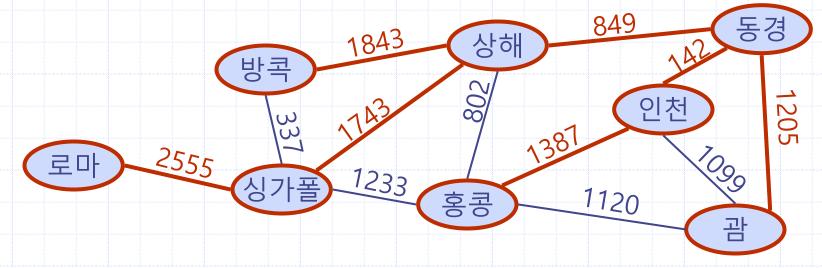
#### 속성 1

최단경로의 부분경로(subpath) 역시 최단경로다

#### 속성 2

출발정점으로부터 다른 모든 정점에 이르는 최단경로들의 트리가 존재 – **단일점 최단경로**(single-source shortest paths)

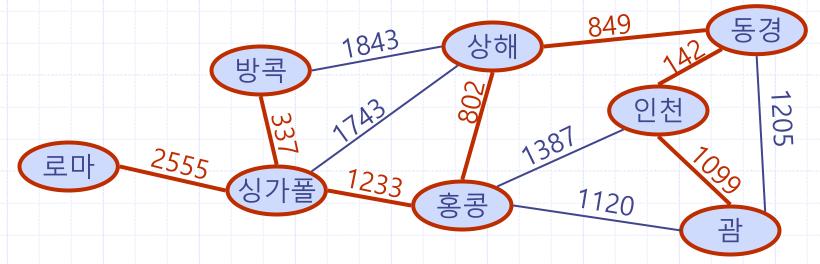
예: 동경으로부터의 최단경로



### 최소신장트리와 비교

- **◈ 최단경로**는 **최소신장트리**와 달리:
  - **무방향**뿐만 아니라 **방향그래프**에서도 정의
  - 그래프에 음의 무게를 가진 싸이클이 있거나, 무방향그래프에 음의 무게를 가진 간선이 있으면 부정
  - 최단경로트리(shortest path tree)는 루트트리

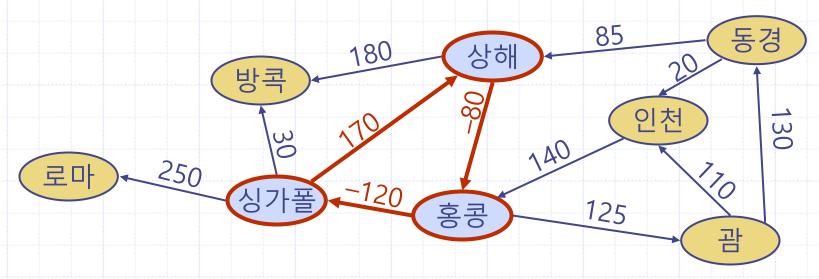
참고: 최소신장트리 예



### 음의 무게를 가지는 간선과 싸이클

- ◆ 가중 방향그래프에 음의 무게를 가진 싸이클이 존재한다면, 최단경로는 존재하지 않는다
- ◆ 가중 무방향그래프에 음의 무게를 가진 간선이 존재한다면, 최단경로는 존재하지 않는다

예: "항공요금 할인 전쟁이 도를 넘다!!"



# 최단경로 알고리즘

그래프	알고리즘	시간
음의 무게를 가진 간선이 없는 그래프	Dijkstra	$\mathbf{O}(m \log n) \text{ or } \mathbf{O}(n^2)$
음의 무게를 가진 간선이 있는 방향 그래프	BFS	$\mathbf{O}(nm)$
비가중그래프	BFS	$\mathbf{O}(n+m)$
DAG	Topological Ordering	$\mathbf{O}(n+m)$

Algorithms 최단경로 7

### Dijkstra 알고리즘

- ▼ 정점 s로부터 정점
   □ 가리: s와 v
   나이의 최단경로의 길이
- ▶ Dijkstra 알고리즘은 출발정점 s로부터 다른 모든 정점까지의 거리를 계산
- ◈ 전제
  - 그래프가 연결됨
  - 간선들은 무방향
  - 간선의 무게는 **음수가 아님**

- ★ s로부터 시작하여 정점들을 차례로 배낭에 넣다가 결국엔 모든 정점을 배낭에 넣는다
- ◈ 각 정점 v에 라벨 d(v)를 저장하여 배낭과 인접 정점들을 구성하는 부그래프에서 s로부터 v까지의 거리를 표현
- ◈ 반복의 각 회전에서:
  - 배낭 밖의 정점 가운데 거리 라벨
     d가 최소인 정점 u를 배낭에 넣는다
  - u에 인접한 정점들의 라벨을 3신

### Dijkstra 알고리즘을 위한 기본 구성

- - ◆ 우선순위 큐가 배낭 밖의 정점들을 보관
    - **키**: 거리
    - **원소**: 정점
  - - 거리(distance): *d*(*v*)
    - **위치자**(locator): 우선순위 큐에서 v의 위치

#### ◈ 보조 메쏘드

■ Q.replaceKey(e, k): 원소 e의 키를 k로 변경하고 e의 우선순위 큐 Q에서의 위치를 갱신

### Dijkstra 알고리즘

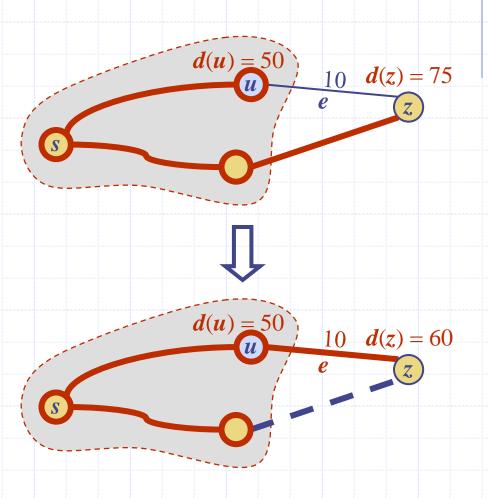
```
Alg DijkstraShortestPaths(G, s)
input a simple undirected
weighted graph G with
nonnegative edge weights,
a vertex s of G
output label d(u), for each
vertex u of G, s.t. d(u) is
the distance from s to u in
G
```

- 1. for each  $v \in G.vertices()$  $d(v) \leftarrow \infty$
- $2. d(s) \leftarrow 0$

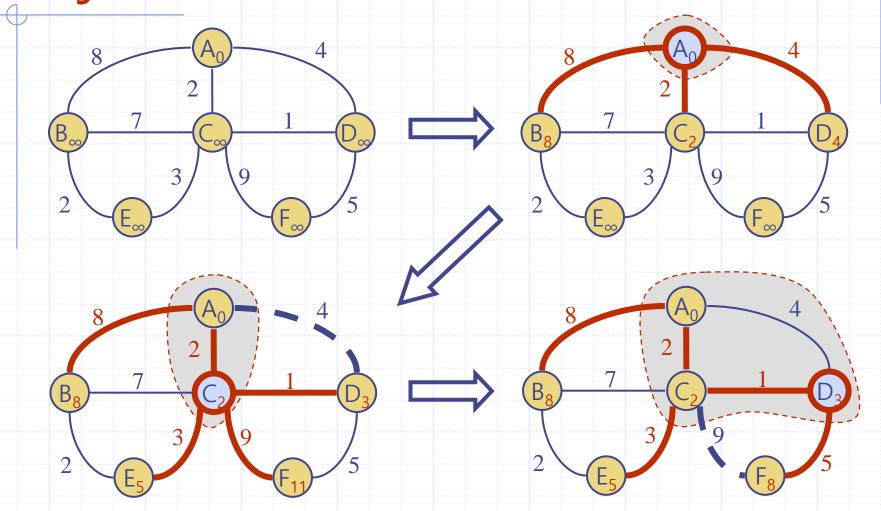
- Alg DijkstraShortestPaths(G, s) 3.  $Q \leftarrow$  a priority queue containing input a simple undirected weighted graph G with 3.  $Q \leftarrow$  a priority queue containing all the vertices of G using G labels as keys
  - 4. while (!Q.isEmpty())  $u \leftarrow Q.removeMin()$   $\{pull\ a\ vertex\ into\ the\ sack\}$ for each  $e \in G.incidentEdges(u)$   $z \leftarrow G.opposite(u, e)$ if  $(z \in Q.elements())$ if (d(u) + w(u, z) < d(z))  $d(z) \leftarrow d(u) + w(u, z)$   $\{relax\ edge\ e\}$  Q.replaceKey(z, d(z))

### 간선완화

- $\bullet$  간선 e = (u, z)에 대해고려하자 여기서:
  - *u*는 가장 최근에 배낭에 들어간 정점
  - ෭는 배낭 밖에 존재
  - 간선 e의 **완화**(relaxation)를 통해 거리 d(z)를 갱신 - 즉,  $d(z) \leftarrow min(d(z), d(u) + w(u, z))$

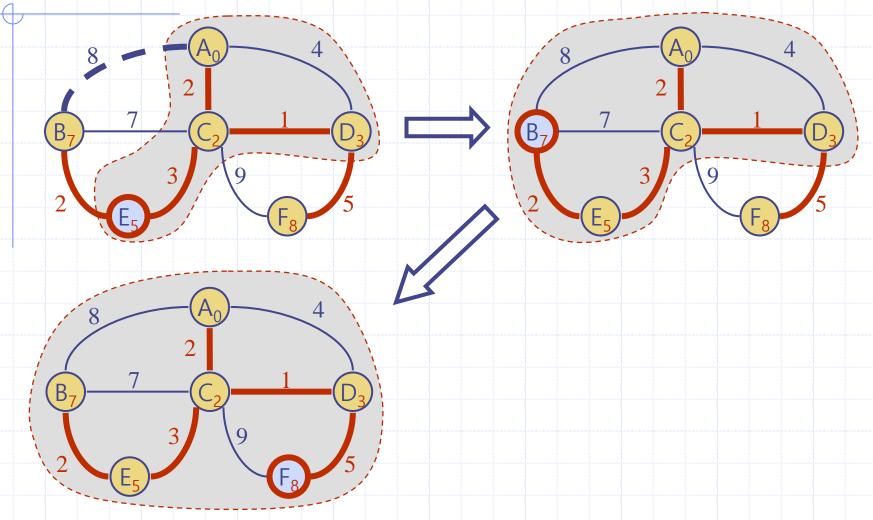


# Dijkstra 알고리즘 수행 예



Algorithms

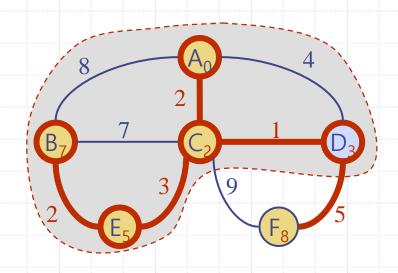
# Dijkstra 알고리즘 수행 예 (conti.)



Algorithms

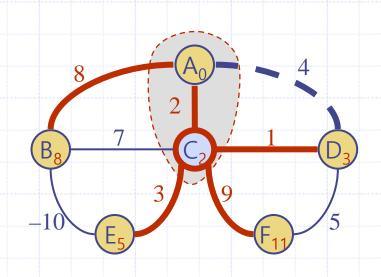
### Dijkstra 알고리즘이 정확한 이유

- ◆ Dijkstra 알고리즘은 탐욕법에 기초한다 따라서 거리가 늘어나는 순서로 정점을 배낭에 삽입
  - (모순법) 알고리즘이 모든 정점에 대한 최단경로를 찾지는 못했다고 가정하고, F를 알고리즘이 잘못 처리한 첫 번째 정점이라고 하자
  - 올바른 최단경로 상의 이전 노드 *D*가 고려될 때, *D*의 거리는 정확했으며 이 값에 기초하여 간선 (*D*, *F*)가 완화되었다
  - 그러므로,  $d(F) \ge d(D)$ 가 성립하는 한 F의 거리는 틀릴 수가 없다 – 즉, 거리가 잘못 계산된 정점은 있을 수 없다!



### 음의 무게를 가진 간선에 대해 정확하게 작동하지 않는 이유

- Dijkstra 알고리즘은 **탐욕법**에 기초한다 따라서 거리가 늘어나는 순서로 정점을 배낭에 삽입
- 음의 부착간선을 가진 노드를 배낭에 넣으면, 이미 배낭 속에 있는 정점들의 거리를 혼란케 한다
  - 그림 예: C의 올바른 거리는 1이지만, d(C) = 2를 가지고 이미 배낭속에 있다!
- ◈ 그럴싸한 해법: 이러면 될까?
  - 1. 모든 간선의 무게에 상수 k를 더하여 음의 무게를 없앤다
  - 2. 그렇게 해서 생성된 새로운 그래프에 대해 최단경로를 구한다
  - 3. 원래 그래프에 맞게 결과를 보정한다 - 즉, 모든 정점의 거리 라벨 값에서 해당 정점에 이르는 최단경로 상의 간선 수만큼 k를 뺀다



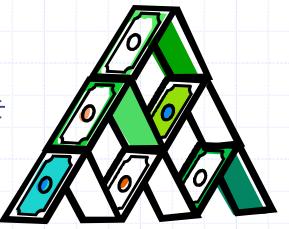
### Dijkstra 알고리즘 분석



- ◈ 그래프 작업
  - 메쏘드 incidentEdges는 각 정점에 대해 한번씩만 호출
  - ◈ 라벨 작업
    - 정점 z의 거리와 위치자 라벨을 O(deg(z))번 읽고 쓴다
    - 라벨을 읽고 쓰는데 **O**(1) 시간
  - ◈ 우선순위 큐의 구현에, 두 가지 선택 가능
    - **힙** 구현
    - 무순리스트 구현

# Dijkstra 알고리즘 분석 (conti.)

- ◆ **힙에 기초한 우선순위 큐**를 사용할 경우:
  - 각 정점은 우선순위 큐에 한번 삽입되고 한번 삭제되며, 각각의 삽입과 삭제에  $O(\log n)$  시간 소요
  - 우선순위 큐 내의 정점 z의 키는 최대 deg(z)번 갱신되며 각각의 키 갱신에  $O(\log n)$  시간 소요
- → 그래프가 **인접리스트** 구조로 표현된 경우, Dijkstra 알고리즘은 **O**((n + m)log n) 시간에 수행
  - 참고:  $\sum_{v} deg(v) = 2m$
- lacktriangle 그래프가 연결되었으므로, 실행시간은  $O(m \log n)$ 으로 표시 가능
- $\bullet$  그래프가 **단순**하다고 전제하므로, 이는 최악의 경우  $\mathbf{O}(n^2 \log n)$



### Dijkstra 알고리즘 분석 (conti.)

- ◆ 무순리스트에 기초한 우선순위 큐를 사용할 경우:
  - 각 정점은 우선순위 큐에 한번 삽입되고 한번 삭제되며, 삽입과 삭제에 각각  $\mathbf{O}(1)$  및  $\mathbf{O}(n)$  시간 소요
  - 우선순위 큐 내의 정점 z의 키는 최대 deg(z)번 갱신되며 각각의 키 갱신에  $\mathbf{O}(1)$  시간 소요
- lacktriangle 그래프가 **인접리스트** 구조로 표현된 경우, Dijkstra 알고리즘은  $\mathbf{O}(n^2+m)$  시간에 수행
  - 참고:  $\sum_{v} deg(v) = 2m$
- lacktriangle 그래프가 **단순**하다고 전제하므로, 이를  $\mathbf{O}(n^2)$ 로 단순화 가능



#### 두 가지 우선순위 큐 구현 비교

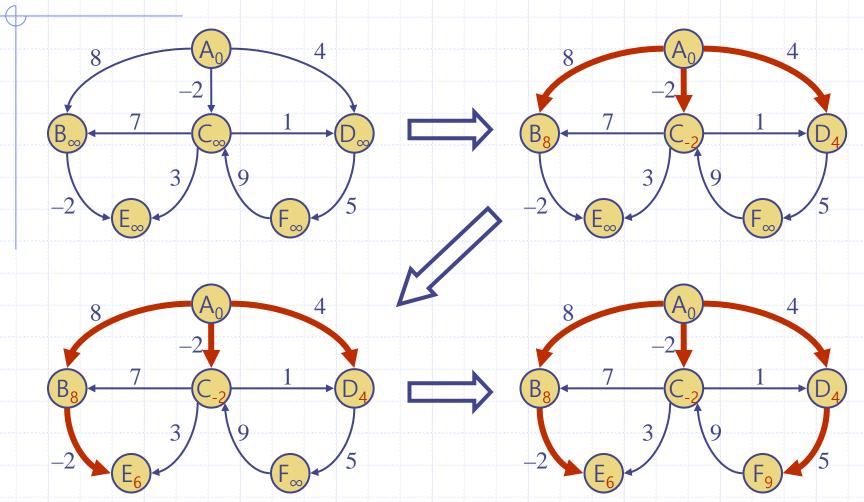
- ◆ Dijkstra 알고리즘의 **우선순위 큐**를 구현하는데 두 가지 선택 가능
  - **힙**으로 구현하면  $O(m \log n)$  시간
  - **무순리스트**로 구현하면  $O(n^2)$  시간
  - ◈ 두 구현은:
    - 모두 코딩하기 쉽다
    - 상수적 요소로 보면 비슷하다
  - ◈ 최악의 경우만을 생각하면 다음과 같이 선택이 달라진다
    - 희소그래프(즉, *m* < *n*<sup>2</sup>/log *n*)에 대해서는 **힙** 구현이 유리
    - 밀집그래프(즉, *m* > *n*<sup>2</sup>/log *n*)에 대해서는 **리스트** 구현이 유리

## Bellman-Ford 알고리즘

- ◆ 음의 무게를 가진 간선이 있더라도 작동
- ◆ 방향간선으로 전제 그렇지 않으면 음의 무게를 가진 싸이클이 있게 되기 때문
- *i*-번째 회전 시에 *i*개의 간선을 사용하는 최단경로를 찾는다
- **♦ 실행시간: O(nm)**
- ▶ BFS 탐색 전략에 기초 알고리즘을 다시 확장하면, 음의 무게를 가지는 싸이클이 있는 경우 이를 발견 가능
- ◆ 인접정보를 사용하지 않으므로 간선리스트 구조 그래프에서 수행 가능

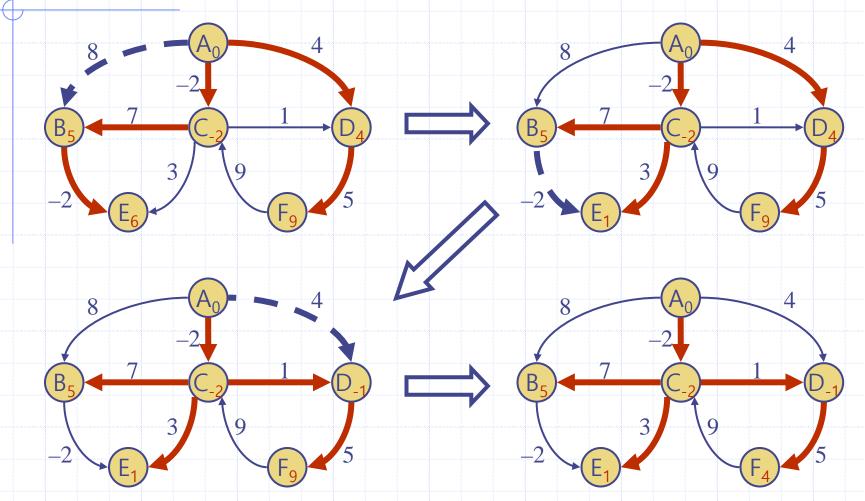
```
Alg Bellman-FordShortestPaths(G, s)
   input a weighted digraph G with n
                vertices, and a vertex s of G
   output label d(u), for each vertex u
                of G, s.t. d(u) is the distance
from s
                    to u in G
1. for each v \in G.vertices()
      d(v) \leftarrow \infty
2. d(s) \leftarrow 0
3. for i \leftarrow 1 to n-1
      for each e \in G.edges()
           {relax edge e}
           u \leftarrow G.origin(e)
           z \leftarrow G.opposite(u, e)
           d(z) \leftarrow min(d(z), d(u) + w(u, z))
```

# Bellman-Ford 알고리즘 수행예



Algorithms

# Bellman-Ford 알고리즘 수행 예 (conti.)



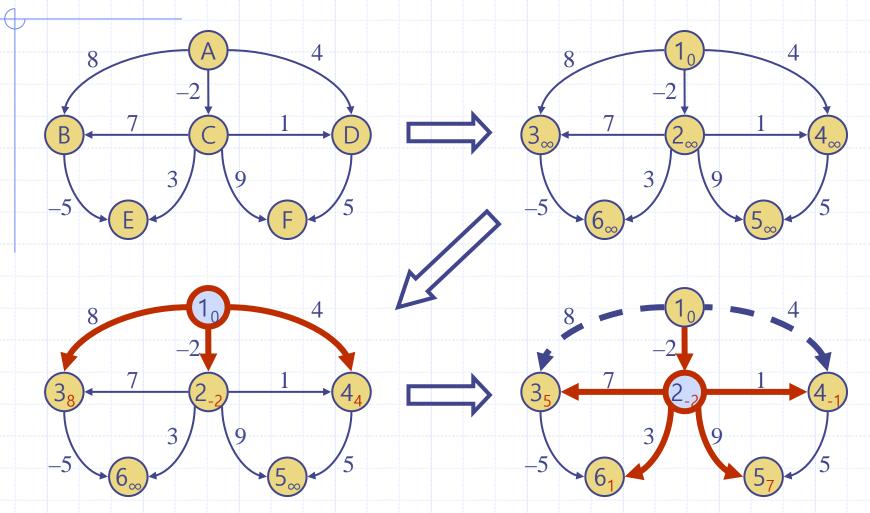
**Algorithms** 

### DAG에서의 최단경로

- ◈ 음의 무게를 가진 간선이 있더라도 작동
  - **◈ 위상순서**를 이용
  - 별다른 데이터구조를 사용하지 않음
  - ◆ Dijkstra 알고리즘 보다 훨씬 빠르다
  - 실행시간:O(n + m)

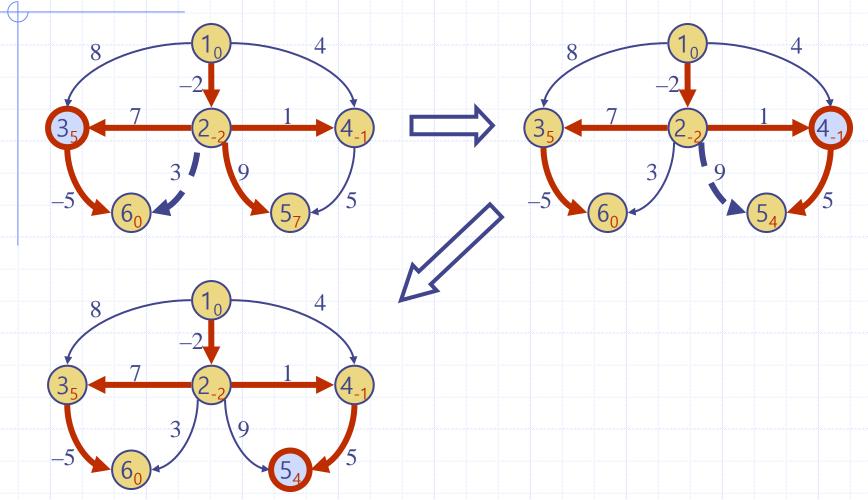
```
Alg DAGShortestPaths(G, s)
   input a weighted DAG G with n vertices and
           m edges, and a vertex s of G
   output label d(u), for each vertex u of G, s.t.
           d(u) is the distance from s to u in G
1. Compute a topological ordering (v_1, v_2, ..., v_n)
   of G
2. for each v \in G.vertices()
      d(v) \leftarrow \infty
3. d(s) \leftarrow 0
4. for i \leftarrow 1 to n-1
       for each e \in G.outIncidentEdges(v_i)
           \{\text{relax edge } e\}
           z \leftarrow G.opposite(v_i, e)
           d(z) \leftarrow min(d(z), d(v_i) + w(v_i, z))
```

# DAG에서의 최단경로 수행예



Algorithms

# DAG에서의 최단경로 수행예 (conti.)

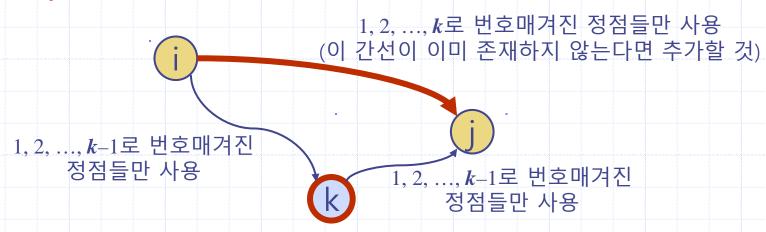


**Algorithms** 





- 모든 쌍 최단경로(all-pairs shortest paths) 문제: 가중 방향그래프 G의 모든 정점쌍 간의 거리를 찾는 문제
- ◆ 그래프에 음의 무게를 가진 간선이 없다면, Dijkstra 알고리즘을 n번 호출할 수도 있다 실행시간:  $O(nm \log n)$
- ◆ 그래프에 음의 무게를 가진 간선이 있다면, Bellman-Ford 알고리즘을 n번 호출할 수도 있다 실행시간:  $O(n^2m)$
- **대안: 동적프로그래밍**을 사용하면  $O(n^3)$  시간에 수행 가능 (Floyd-Warshall 알고리즘과 유사한 방식으로 수행)



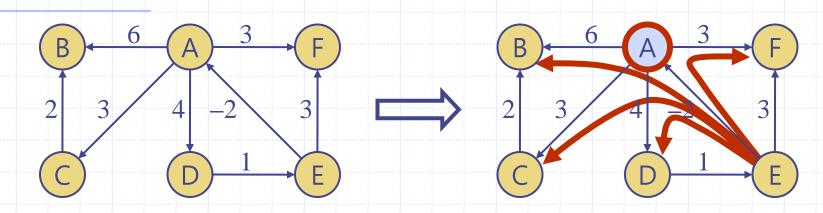
Algorithms

#### 모든 쌍 최단경로

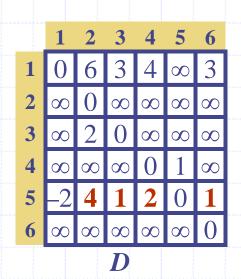
```
Alg<br/>allPairsShortestPaths(G)<br/>input a simple1. Let v_1, v_2, ..., v<br/>vertices of Ginput a simple2. for i \leftarrow 1 to nweighted digraph Gfor j \leftarrow 1 to nwithout negative-weight cyclesif (i = j)<br/>D[i, j]<br/>is the distance from v_i to v_j in Gelseif (i = j)<br/>D[i, j]<br/>D[i, j]<br/>is the distance from v_i to v_j in G
```

```
1. Let v_1, v_2, ..., v_n be an arbitrary numbering of the
   vertices of G
2. for i \leftarrow 1 to n
       for j \leftarrow 1 to n
             if (i = j)
                  D[i,j] \leftarrow 0
             elseif ((v_i, v_i) \in G.edges())
                  D[i,j] \leftarrow w(v_i,v_i)
             else
                  D[i,j] \leftarrow \infty
        for i \leftarrow 1 to n
             for j \leftarrow 1 to n
                  D[i,j] \leftarrow min(D[i,j], D[i,k] + D[k,j])
```

### 모든 쌍 최단경로 수행 예

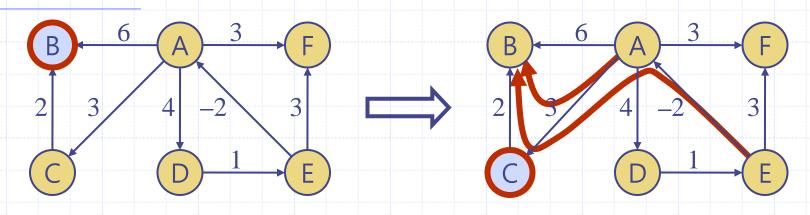


	1	2	3	4	5	6
1	0	6	3	4	$\infty$	3
2	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	2	0	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	0	1	$\infty$
5	-2	$\infty$	$\infty$	$\infty$	0	3
6	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	0
			D			

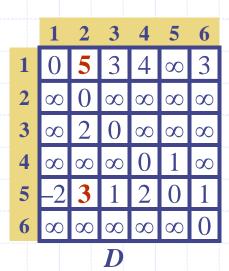


Algorithms 최단경로 28

# 모든 쌍 최단경로 수행예 (conti.)

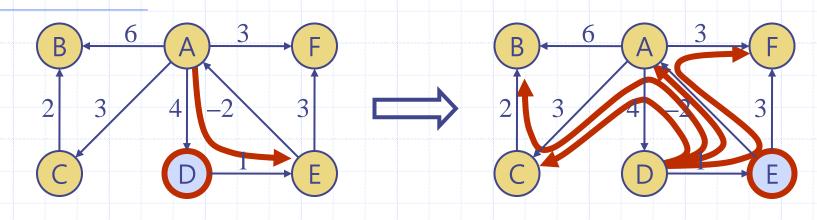


	1	2	3	4	5	6
1	0	6	3	4	$\infty$	3
2	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	2	0	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	0	1	$\infty$
5	-2	4	1	2	0	1
6	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	0
			D			

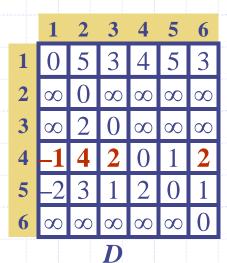


Algorithms

## 모든 쌍 최단경로 수행 예 (conti.)

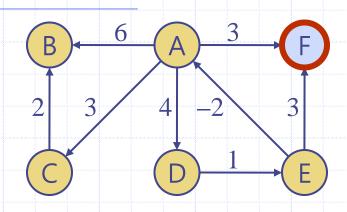


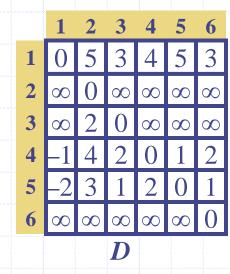
	1	2	3	4	5	6
1	0	5	3	4	5	3
2	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	2	0	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	0	1	$\infty$
5	-2	3	1	2	0	1
6	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	0
			D			



Algorithms 최단경로 30

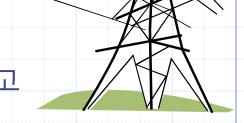
## 모든 쌍 최단경로 수행 예 (conti.)



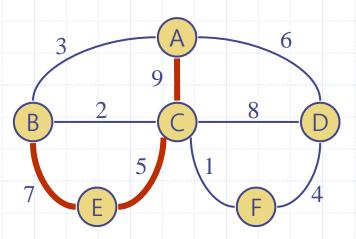


Algorithms

### 응용문제: 두 지국 사이의 최대 대역폭



- ◆ 전화망의 다이어그램이 그래프로 주어졌다고 가정하자
  - 정점: 지국
  - **간선:** 지국 간의 통신선로 (각 간선은 **대역폭**(bandwidth)과 함께 표시됨)
- ◆ 경로의 대역폭: 경로 내 최소의 대역폭을 가지는 간선(즉, 병목)의 대역폭
  - 예: A, B 간 최대대역폭 = 5
- ▶ 문제: 다이어그램과 두 개의 지국 x, y가 주어졌을 때, x, y 사이의 경로 가운데 최대대역폭을 구하는 알고리즘을 작성하라
- ◈ 힌트: Dijkstra 알고리즘의 확장



### 해결: Dijkstra 알고리즘 확장

- ◆ Dijkstra 알고리즘에서와 동일한 아이디어를 사용
- ◆ 각 정점 v에 새로운 라벨b를 유지
  - **bandwidth:** *b*(*v*), *x*로부터 *v*까지 경로의 대역폭
- ◆ 모든 정점에 대해 b 라벨 값을 0으로 초기화
  - 단, 출발점 *x*의 *b* 값은 무한대로 초기화
- 작업 removeMin은 우선순위 큐로부터 최대 b 값을 가지는 노드를 삭제
- ♦ 실행시간: (힙에 기초한 우선순위 큐를 사용할 경우) O((n + m)log n)

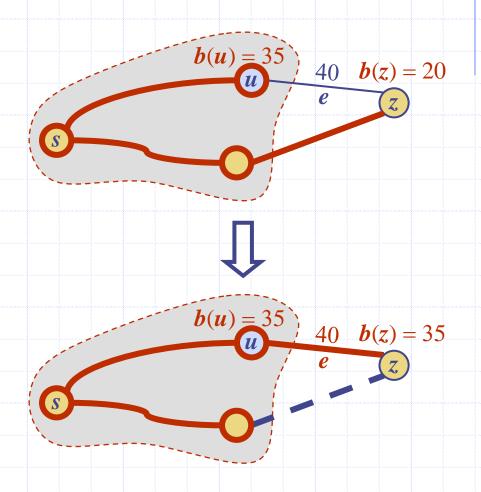
#### Alg DijkstraMaxBandwidth(G, x, y)

- 1. for each  $v \in G.vertices()$  $b(v) \leftarrow 0$
- $2. b(x) \leftarrow \infty$
- 3.  $Q \leftarrow$  a priority queue containing all the vertices of G using the b labels as keys
- 4. while (!Q.isEmpty())  $u \leftarrow Q.removeMin()$ for each  $e \in G.incidentEdges(u)$   $z \leftarrow G.opposite(u, e)$ if  $(z \in Q.elements())$ 
  - if (min(b(u), w(u, z)) > b(z))  $b(z) \leftarrow min(b(u), w(u, z))$ Q.replaceKey(z, b(z))
- 5. return b(y)

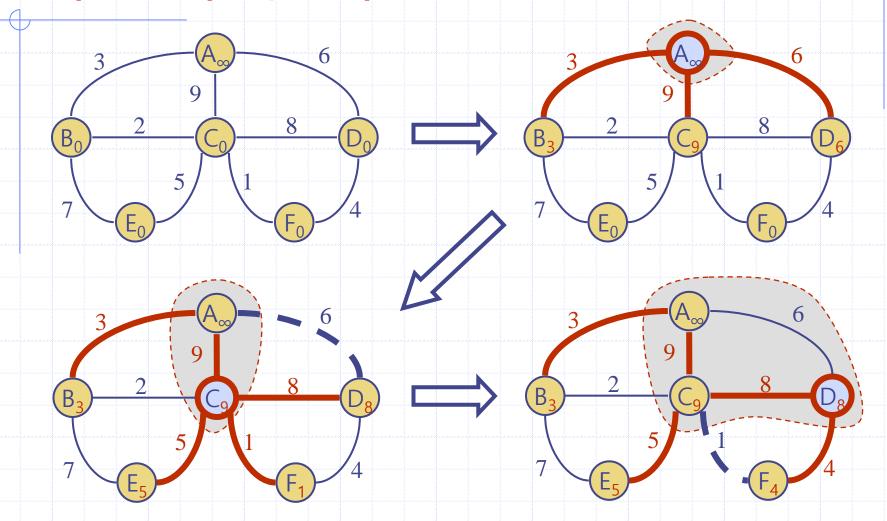
### 해결: 간선완화

- ◆ 간선완화 단계는 Dijkstra에서와 매우 유사
- ◆ 다음과 같은 간선 e = (u, z)을 고려해보자
  - *u*는 배낭에 최근에 추가된 정점
  - <sup>෭</sup>은 배낭에 존재하지 않는다
- ↑ 간선 e의 완화는 b(z), 즉,
   z에 대한 대역폭을
   다음과 같이 갱신

 $b(z) \leftarrow max(b(z), min(b(u), weight(e)))$ 

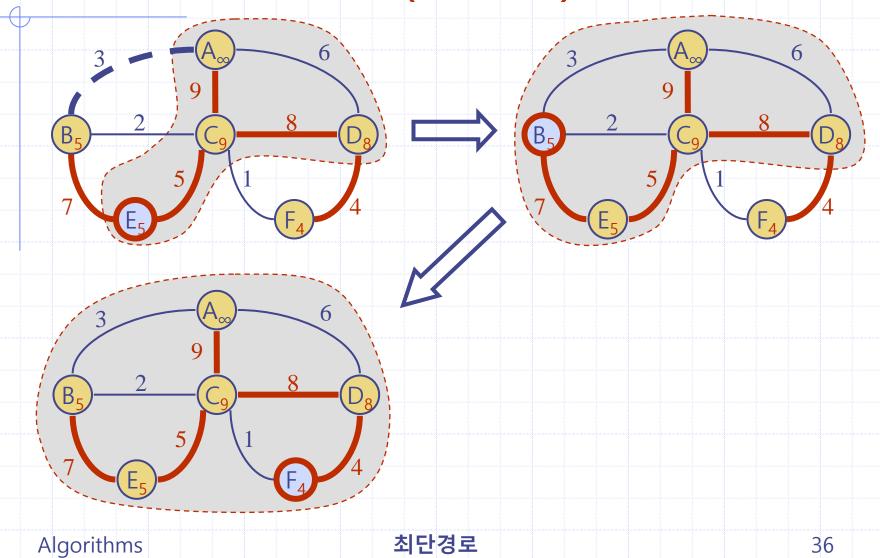


### 해결: 수행 예



Algorithms

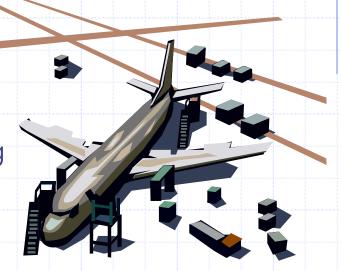
## 해결: 수행 예 (conti.)



36

# 응용문제: 항공편 스케줄링

- ◆ 다음 내용으로 구성된 항공편 시간표가 있다고 가정하자 ■
  - *n*개의 공항 집합: *A*
  - 각 공항  $a \subseteq A$  에 대한 **최소연결시간**(minimum connecting time): c(a)
  - m개의 항공편 집합: F
  - lacktriangle 각 항공편  $f \in F$  에 대한 정보
    - 출발공항: a₁(f) ∈ A
    - 도착공항:  $a_2(f) \subseteq A$
    - ◆ 출발시각: *t*₁(*f*)
    - ◆ 도착시각: *t*<sub>2</sub>(*f*)



# 응용문제: 항공편 스케쥴링 (conti.)

- ♥ 문제: 주어진 두 개의 공항 a, b와 시각 t에 대해 a에서 시각 t 정시 혹은 이후에 출발할 경우 가장 이른 시각에 b에 도착할 수 있도록 하는 연결항공편을 계산하고 알고리즘의 실행시간을 n과의 m의 함수로 구하라
  - ◆ 주의: 환승공항에서의 최소연결시간은 반드시 준수되어야 한다



#### 해결: 문제해결 개요

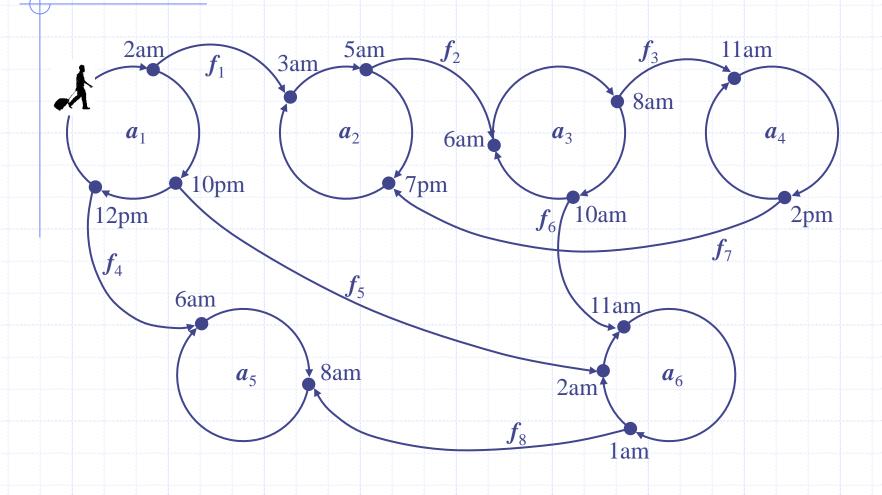
- ◈ 공항 a ∈ A 에서 출발하여 공항 b ∈ A 에 도착하는최소 여행시간 경로를 찾는 항공편 스케줄링 문제를최단경로 문제로 전환
- ◆ 즉, 최단경로 문제의 한 경우로 문제를 재구성하고 이를 해결함으로써 항공편 스케줄링 문제에 대한 해를 구한다

Algorithms 최단경로 39

# 해결: 최단경로 문제로 재구성

- ◆ 주어진 공항 집합 A와 항공편 집합 F에 대해 다음과 같이 가중 방향그래프 G를 구축
  - 각 공항  $a \in A$  의 24시간을 표현하는  $circle_i$ 를 그린다
  - lacktriangle 각 항공편  $f_i \in F$  의 출발공항  $a_0$ , 도착공항  $a_d$ , 출발시각  $t_d$ , 도착시각  $t_a$ 를 찾는다
  - $circle_o$ 의 시간  $t_d$  위치에 정점  $v_1$ 을 그려 넣고,  $circle_d$ 의 시간  $t_a+c(a_d)$  위치에 정점  $v_2$ 를 그려 넣는다
  - $lackbox{v}_1$ 에서  $lackbox{v}_2$ 로 향하며 무게  $m{t}_a$   $+ m{c}(m{a}_d) m{t}_d$ 를 가지는 방향간선을 그려 넣는다
    - ◆ **주의:** 자정을 지나가는 항공편도 정확한 무게를 계산
  - circlei에 놓인 간선들의 방향은 모두 시계방향이어야 한다.

# 해결: 완성된 가중 방향그래프



Algorithms

최단경로

## 해결: 최단경로 문제해결

- ◈ 항공편 스케줄링 문제해결: 그래프 G에서  $circle_a$ 의 시각 t 혹은 t 이후를 나타내는 **첫 정점**으로부터  $circle_b$ 의 **아무 정점**까지의 **최단경로**만 찾으면 된다
  - a는 최초 출발공항, b는 목표공항
  - **연결항공편**: 출발지로부터 도착지까지의 최단경로로부터 회수
- - **방향그래프**에 적용하도록 확장: 간선완화 과정에서 가장 최근에 배낭에 들어온 정점의 **진출 부착간선**들만 고려 (**참고:** 메쏘드 outIncidentEdges)
  - **최단경로**를 **회수**하도록 확장: 방문한 정점들에 대해 완화된 간선들의 부모-자식 관계를 포함

### 해결: 알고리즘 성능

- - ❖ 최단경로 찾기:Dijkstra 알고리즘의 실행시간과 동일
    - 우선순위 큐에 **힙**을 사용하면 총 수행시간: **O**((*n* + *m*)log *n*)
    - 일반적으로 항공스케줄링 문제는 단순그래프로 구축되므로 *n* = **O**(*m*)
    - 따라서 **실행시간: O**(*m* log *n*)

```
1. \mathbf{v} \leftarrow \text{first vertex on } \mathbf{circle}_{\mathbf{q}} \text{ representing time } \mathbf{t}
               or after t
2. for each u \in G.vertices()
         d(u) \leftarrow \infty
        p(v) \leftarrow \emptyset
3. d(v) \leftarrow 0
4. Q \leftarrow a priority queue containing all the
                vertices of G using d labels as keys
5. while (!Q.isEmpty())
         u \leftarrow O.removeMin()
         for each e \subseteq G.outIncidentEdges(u)
               z \leftarrow G.opposite(u, e)
               if (z \subseteq Q.elements())
                      if (d(u) + w(u, z) \leq d(z))
                            d(z) \leftarrow d(u) + w(u, z)
                           p(z) \leftarrow e
                            Q.replaceKey(z, d(z))
6. \mathbf{w} \leftarrow vertex \ on \ \mathbf{circle_h} \ with \ minimum \ \mathbf{d} \ label
7. return reversed path from w to v
```

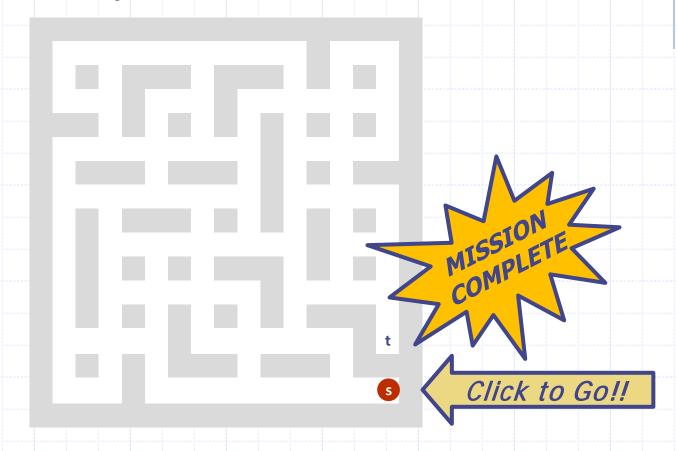
# 응용문제: 좌회전을 못하는 차



- 배경: 스포츠카를 훔치려는 온달의 계획
- ◈ 제약
  - 좌회전 불가능(따라서 유턴도 불가능)
  - 우회전을 최소화할 필요성(경보음 때문)
- ◆ 문제: 다음을 수행할 효율적인 메쏘드
  - 출발지 s에서 목적지 t까지 **우회전**을 최소화한 경로 찾기
  - 만약 두 개 이상의 경로가 우회전 수가 같다면 **총주행거리** 가 짧은 경로 찾기
- ◆ 도움: 디지털 도로지도
  - 전체 도시가  $m \times n$  격자형의 셀로 표시됨
  - 각 셀은 비어 있거나(운행 가능), 막혀 있거나(운행 불가) 둘 중 하나

# 응용문제: 좌회전을 못하는 차 (conti.)

◈ 도로지도 예: Sin City

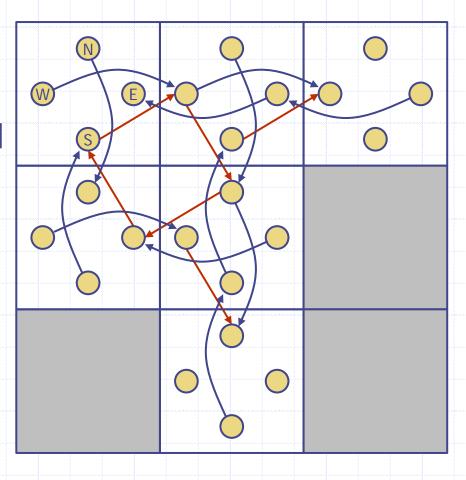


Algorithms

최단경로

### 해결: 최단경로 문제로 재구성

- ◆ 도로지도를 가중 방향그래프로 전환
- ◈ 정점
  - *mn*개의 셀 각각에 대해 E, W, S, N, 4개의 **방위** 정점을 생성
- ◈ 간선
  - **직진:** 셀의 *o* 정점에서 인접 셀의 *o* 정점으로 진행: **무게** = 1
  - **우회전:** 어떤 셀의 방위  $o_1$ 에서  $o_2$ 로 진행( $o_1 \neq o_2$ ): **무게** = 4mn + 1
  - **좌회전, U-턴**: 없음



### 해결: 최단경로 문제해결

- ◆ Dijkstra 알고리즘
  - 모든 간선의 무게가 **양수**
  - 최소의 우회전 경로, 그런 경로가 여럿이면 그 중 총주행거리가 최소인 경로를 구함
  - ◈ 성능
    - **그래프 구축: O**(*mn*) 시간
    - **최단경로 찾기:** 우선순위 큐에 **힙**을 사용하면 **O**(*mn* log *mn*) 시간

# 응용문제: 괴물성에 같힌 낙랑



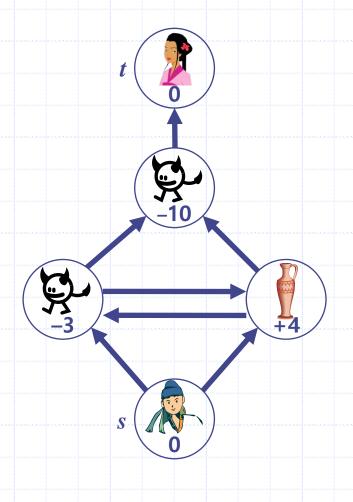
- ◈배경
  - 호동이 **괴물성**에 갇힌 낙랑을 구출
  - 괴물성의 구조: 방과 복도로 구성된 미로
- lacktriangle 미로: 방향그래프 G = (V, E)
  - **정점: 방**, 각 정점 V마다 라벨 *f*가 정의되어 있다
    - f(v) = 0 이면 v가 비어 있다
    - \* f(v) > 0 이면 v에 생명수가 있다 v에 들어가면 에너지 L이 f(v)
       만큼 올라간다.
    - f(v) < 0 이면 v에 **괴물**이 있다 -v에 들어가면 **에너지** L이 |f(v)| 만큼 떨어진다
  - **간선**: 방과 방을 연결하는 **일방통로**
- ◈ 제약: 에너지 레벨 L을 0 이상으로 유지  $-L \le 0$  이 되는 순간 호동이 죽는다

# 응용문제: 괴물성에 갇힌 낙랑

 \* 목표: 초기 에너지 L을 가지고 미로의 입구 방 s ∈ V에서 출발하여 여러 개의 방을 뒤져 낙랑이 있는 방 t ∈ V을 찾기 (s ≠ t) 

#### ◈ 전제

- s로부터 다른 모든 정점  $v \in V$  에 이르는 경로가 존재
- 모든 정점  $v \in V$  로부터 t에 이르는 경로가 존재
- 호동은 에너지  $L = L_0 > 0$  인 상태로 입구 s에서 출발
- 문제 단순화를 위해 입구 방은 비어 있다고 전제 – 즉, **f**(s) = 0



# 응용문제: 괴물성에 갇힌 낙랑

- 당신은 미로에 괴물과 생명수를 무작위로 배치하는 프로그램을 완성했다
- 프로그램에 의해 생성된 미로 가운데 어떤 것은 충분한 에너지  $L_0 > 0$ 로 미로에서 출발하지 않으면 s에서 t까지 안전한 여행이 불가능한 경우도 있다
- ★ s에서 t까지의 경로는 호동이 살아서 통과한다면, 즉 중간에 에너지가 0 이하로 떨어지는 일이 없다면 안전하다고 말한다
- lacktriangle 호동이 에너지  $m{L}_0 = m{r}$  로 시작할 때 안전한 경로가 존재한다면 그 미로를 **레벨** $-m{r}$ 이라고 정의하자

## 응용문제: 괴물성에 갇힌 낙랑

- ◈ 문제
- A. 현재처럼 **정점**이 아니라 **간선**에 무게가 실린, 그렇지만 동일한 문제로 재구성하라
- B. 에너지의 증가를 가져오는 **싸이클이 없다**는 전제 하에, 주어진 r에 대해 어떤 미로가 레벨-r인지 검사할 방법을 설명하라
- C. 이번엔 반대로, 에너지의 증가를 가져오는 **싸이클이 있을 수 있다**는 전제 하에, 주어진 **r**에 대해 어떤 미로가 레벨-**r**인지 검사할 방법을 설명하라
- D. 주어진 미로가 레벨-r이 되는 **최소**의 r을 찾아내거나 그런 r이 없으면 없다고 보고하는 효율적인 메쏘드를 설명하라

# 해결 (개요)

▶ 문제해결 개요: 그래프에 대한 통찰력과 Bellman-Ford 최단경로 알고리즘에 대한 이해를 바탕으로 해결 가능

#### 해결 A

- ◆ 간선에 무게가 실린 그래프로 변환
  - 입구 정점에 무게가 있다면, 무게 0의 새로운 입구 정점을 만들고 이 정점에서 원래 입구 정점으로 향하는 간선 추가
  - 다음과 같은 절차로 정점들의 무게를 간선으로 이동 **절차:** 정점 v가 무게 f(v)를 가진다면, 모든 간선  $(u, v) \subseteq E$  에 대해 w(u, v) = f(v)를 저장
  - 이제 간선에 무게가 가중되도록 변경된 그래프, 즉 가중그래프에 대해 모든 부분경로(subpath)의 무게가 양수가 되는 경로를 찾는, 동일한 문제로 재구성되었다

#### 해결 B

- ★ 레벨-r인지 검사할 방법: 에너지 증가 싸이클이 없는 경우, 다음과 같이 Bellman-Ford 알고리즘의 수정 버전을 사용
- ◈ 주어진 r에 대해 모든 정점 u에 대해 호동이 u에 도달하면 얻을 수 있는 최대 (양의) 에너지 e(u)를 구한다
  - 만약 e(u)가 양수면 그래프는 레벨-r임
  - 각 정점  $u \in V$  에 대해 e(u)의 하한 d(u)를 유지
  - 모든 d(u) 값을 입구에 대해서는 r로, 다른 모든 정점들에 대해서는  $-\infty$ 로 초기화

# 해결 B (conti.)

- ♦ Bellman-Ford 알고리즘을 수행하면서 간선을 완화하는 동안 (양의무게를 가진 **싸이클이 없다면**) d(u)는 e(u)에 수렴할 때까지 증가
  - 음의 에너지로 어떤 정점에 도달할 바엔 차라리 그 곳에 가지 않는 편이 낫다
  - 따라서 수정 결과가 **양수**일 때만 *d(u)*를 수정
  - 그렇지 않으면 d(u)를  $-\infty$ 인 채로 놔둔다
  - 이를 위해 **간선완화** 부분을 다음과 같이 수정

◆ 모든 간선이 n번 완화된 후 (양의 무게를 가진 싸이클이 없다면) 모든 d(u)는 해당 e(u)에 수렴한다(즉, 정점 u에 도달하면 얻을 수 있는 최대에너지) – 이 시점에 e(t)가 양수면 호동이 이곳에 양의 점수로 도달할수 있는 것이므로 그래프는 레벨-r이 된다.

#### 해결C

- ▶ 레벨-r인지 검사할 방법: 에너지 증가 싸이클이 있는 경우, 모든 간선을 n − 1 번 완화한 후에도 d(t)가 양수가 아니면, (Bellman-Ford 알고리즘에서 싸이클을 찾을 때처럼) 모든 간선을 한 번 더 완화한다
- $\bullet$  이때 어느 정점의 d(u)가 변화하면
  - *s*에서 에너지 *r*로 출발하여 도달 가능한 양의 무게를 가진 **싸이클**을 찾은 것이다
  - 따라서 호동은 t에 도달하기 위해 필요한 에너지를 모으기 위해 충분한 횟수만큼 이 싸이클을 돌면 된다
  - 그러므로 그래프는 레벨-r이 된다
- 만약 도달 가능한 양의 무게를 가진 **싸이클**을 찾지 못하고 d(t) =  $-\infty$  라면
  - 그래프는 레벨-r이 아니다
- lacktriangle 이 알고리즘의 정확성은 Bellman-Ford 알고리즘의 정확성에 기초하며 실행시간은 O(mn)이다

#### 해결 D

- ◈ 미로가 레벨-r이 되는 최소의 r 찾기: 앞서 수정된 간선완화 부분을 이용하여 최소 r을 찾을 수 있다
  - ◈ 먼저 그래프가 레벨-1인지 검사하여 맞다면 1을 답으로 반환
- ◈ 그렇지 않으면 그래프가 레벨-2인지 검사하고, 다시 그래프가 레벨-4인지, 다시 그래프가 레벨-8인지, ... 이런 식으로 계속 검사 즉, i번째 단계에서 그래프가 레벨- $2^{i-1}$ 인지 검사
  - 결국엔 그래프가 레벨- $2^{k-1}$ 이 아니지만 레벨- $2^k$ 인 k를 찾을 수 있다
  - lacktriangleright r의 최소값은 이 두 값 사이에 있다
  - r 값을 찾기 위해  $r = 2^{k-1}$ 과  $r = 2^k$  사이 구간에서 **이진탐색** 수행
  - 반복 횟수  $k = \lfloor \log r \rfloor$  이므로  $k + O(\log r) = O(\log r)$
  - 따라서 Bellman-Ford를  $O(\log r)$  번 수행해야 하므로 총 실행시간은  $O(mn \log r)$