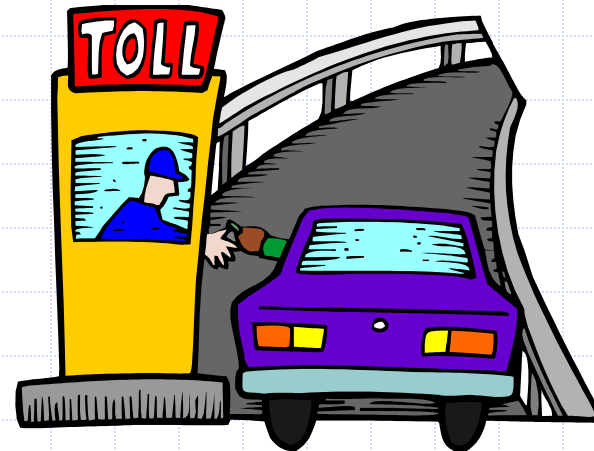


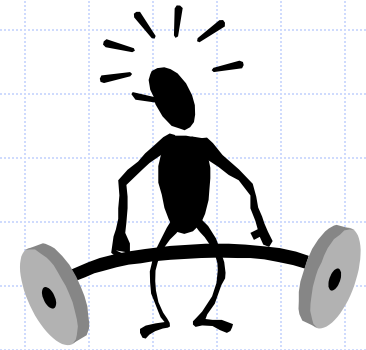
최소신장트리



Outline

- ◆ 16.1 가중그래프
- ◆ 16.2 최소신장트리
- ◆ 16.3 탐욕법
- ◆ 16.4 최소신장트리 알고리즘
- ◆ 16.5 응용문제

가중그래프

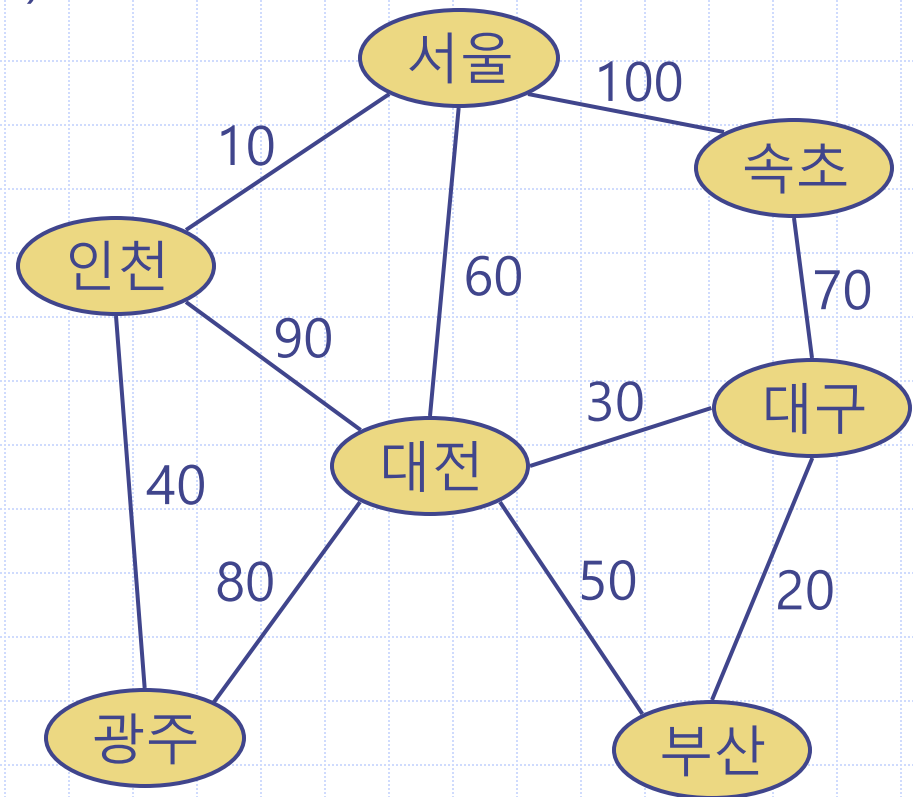


- ◆ **가중그래프**(weighted graph):
각 간선이 **무게**(weight)라 부르는 수치값을 가지는 그래프

- ◆ **무게**: 거리, 비용, 시간 등

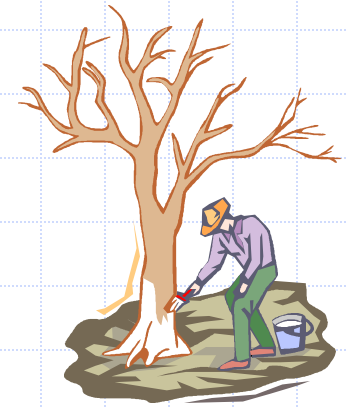
- ◆ **예**

- 오른쪽 철도 그래프에서,
간선의 무게는 양끝점
역간의 여행거리(km)를
표현



가중그래프 예

최소신장트리



신장 부그래프(spanning subgraph)

- 그래프 G 의 모든 정점들을 포함하는 부그래프

신장트리(spanning tree)

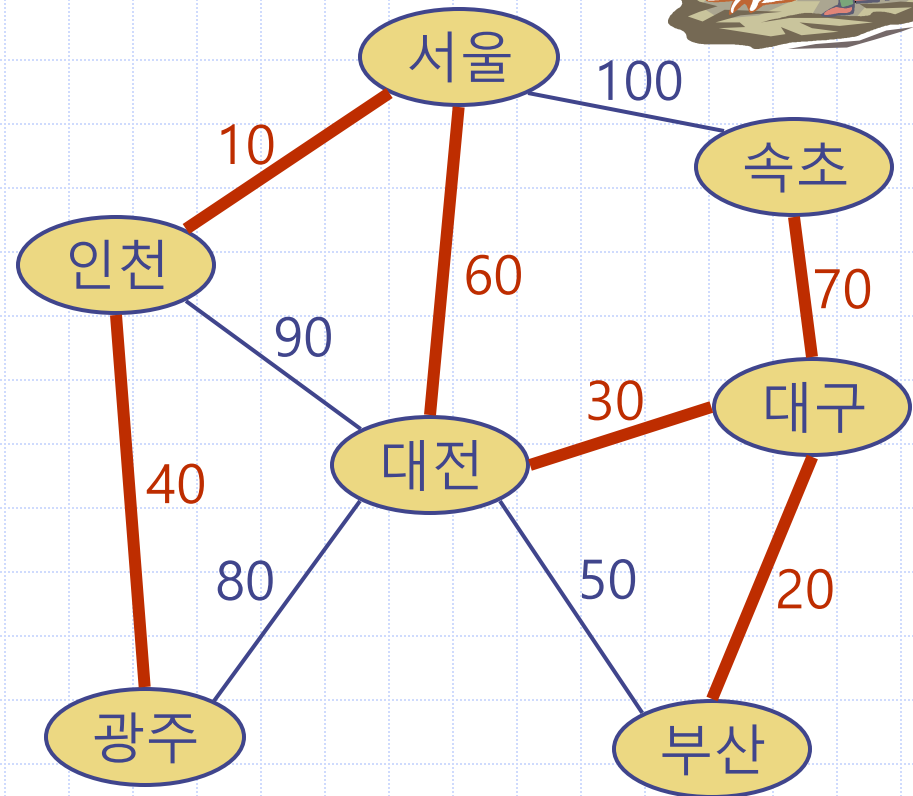
- (자유) 트리인 신장 부그래프

최소신장트리(minimum spanning tree, **MST**)

- 가중그래프의 총 간선 무게가 최소인 신장트리

◆ 응용

- 통신망
- 교통망



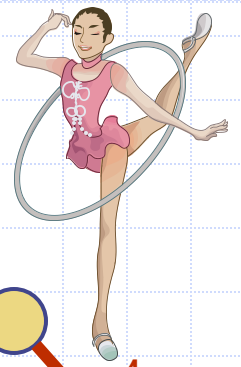
최소신장트리 예

최소신장트리 속성

- ◆ 싸이클 속성
- ◆ 분할 속성

최소신장트리 알고리즘의 정확성 검증에 이용

싸이클 속성

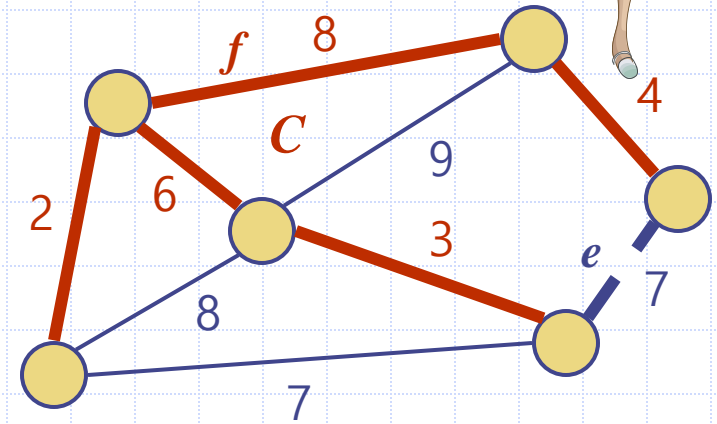


싸이클 속성

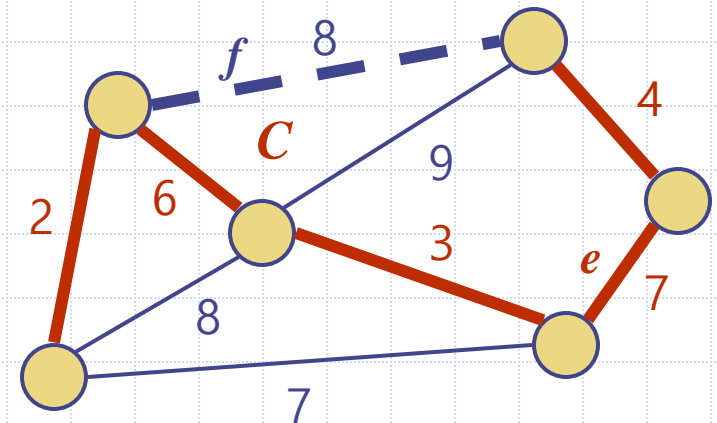
- T 를 가중그래프 G 의 최소신장트리라 하자
- e 를 T 에 존재하지 않는 G 의 간선으로, C 를 e 를 T 에 추가하여 형성된 싸이클로 가정
- 그러면 C 의 모든 간선 f 에 대해, $weight(f) \leq weight(e)$

증명

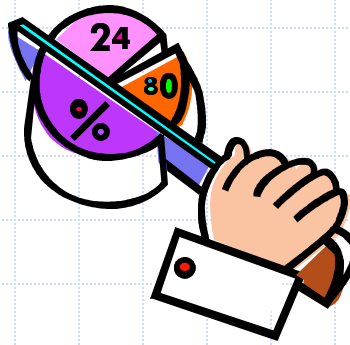
- 모순법
- 만약 $weight(f) > weight(e)$ 라면, f 를 e 로 대체함으로써 무게가 더 작은 신장트리를 얻을 수 있기 때문



↓ f 를 e 로 대체하면 더 좋은 신장트리를 얻는다



분할 속성

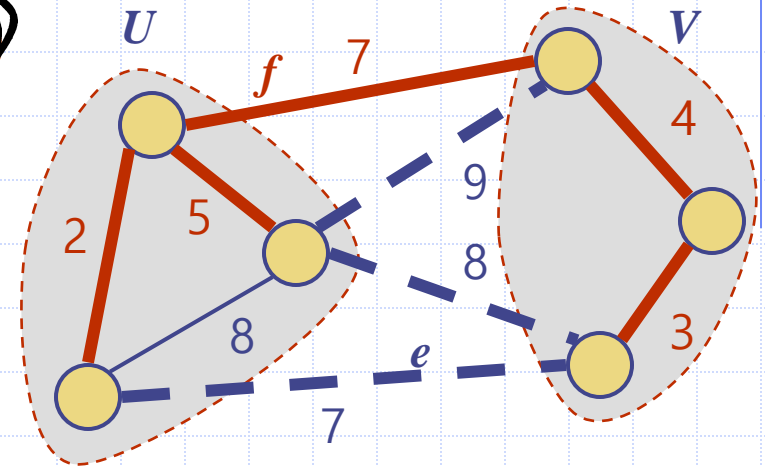


분할 속성

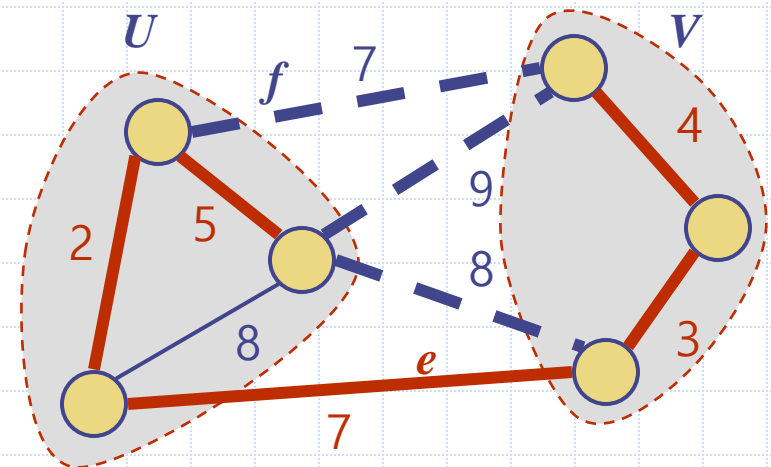
- G 의 정점들을 두 개의 부분집합 U 와 V 로 분할한다고 하자
- e 를 분할을 가로지르는 최소 무게의 간선이라고 하자
- 간선 e 를 포함하는 G 의 최소신장트리가 반드시 존재

증명

- T 를 G 의 MST라 하자
- 만약 T 가 e 를 포함하지 않는다면, e 를 T 에 추가하여 형성된 사이클 C 를 구성하는 간선들 가운데 분할을 가로지르는 간선 f 가 존재
- 사이클 속성에 의해,
 $weight(f) \leq weight(e)$
- 그러므로, $weight(f) = weight(e)$
- f 를 e 로 대체하면 또 하나의 MST를 얻을 수 있다



↓ f 를 e 로 대체하면 다른 MST를 얻는다





탐욕법

- ◆ 탐욕법(greedy method): 일반적인 알고리즘 설계 기법 가운데 하나 – 다음 요소에 기초하여 설계
 - 구성(configuration): 다양한 선택, 모음, 또는 찾아야 할 값들
 - 목표(objective): 구성에 할당된 점수가 존재하며, 이를 최대화 또는 최소화해야 하는 상황
- ◆ 탐욕적 선택 속성(greedy-choice property)을 가진 문제에 적용할 경우 가장 잘 맞는다
 - 출발 구성으로부터 시작하여 지속적인 지역적 향상을 통해 전체 최적해를 항상 찾을 수 있다
- ◆ 예
 - 잔돈 거스르기
 - 부분적 배낭 문제(the fractional knapsack problem)
 - 최소신장트리 문제(minimum spanning trees)

잔돈 거스르기 문제



◆ 구성

- 거슬러줘야 할 금액 정보
- 종류별로 여러 개의 동전들

◆ 목표

- 동전 수를 최소화

◆ 탐욕해

- 가능하면 항상 제일 큰 동전으로 거슬러준다

“만약 이 문제가 탐욕적 선택 속성을 가졌다면 탐욕해는 최적해가 된다”

◆ 예 1: 동전 종류가 32원, 8원, 1원인 경우

- 탐욕적 선택 속성 있음 - 왜냐면, 32원 이상의 금액은 32원을 빼고는 최소 수의 동전으로 만들 수가 없기 때문(8원을 넘지만 32원에 못 미치는 금액에 대해서도 마찬가지)

◆ 예 2: 동전 종류가 30원, 20원, 5원, 1원인 경우

- 탐욕적 선택 속성 없음 - 왜냐면, 40원은 두 개의 20원만으로 만들 수 있으나, 탐욕해는 세 개의 동전을 택하기 때문(어떤 세 개인가?)



부분적 배낭 문제

- ◆ 구성: n 항목의 집합 S – 각 항목 i 는 다음을 가진다
 - b_i : 양수의 이득
 - v_i : 양수의 부피
- ◆ 목표
 - 최대 부피 한도 V 내에서, 최대의 총이득을 주는 항목들을 선택
- ◆ 부분적 배낭 문제(the fractional knapsack problem):
각 항목의 일부만을 취할 수 있는 문제
 - x_i 가 항목 i 를 덜어 담는 양을 표시한다고 하면,
 - 목표: $\sum_{i \in S} b_i(x_i / v_i)$ 를 최대화
 - 제약: $\sum_{i \in S} x_i \leq V$



부분적 배낭 문제 예

- ◆ **구성:** n 항목의 집합 S – 각 항목 i 는 다음을 가진다
 - b_i : 양수의 이득
 - v_i : 양수의 부피
- ◆ **목표:** 최대 부피 한도 V 내에서, 최대의 총이득을 주는 항목들을 선택

항목:



A



B



C



D



E

이득:	32	15	27	5	4
부피:	8ml	3ml	9ml	5ml	2ml
가치:	4	5	3	1	2
(ml 당)					



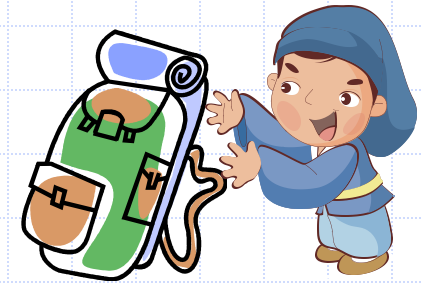
배낭 10ml

탐욕해:

- **B** (3ml)
- **A** (7ml)

총이득: 43

0-1 배낭 문제



- ◆ 0/1 배낭 문제(또는 all-or-nothing knapsack problem): 각 항목의 일부만을 취할 수 없는 문제
- ◆ 0/1 배낭 문제는 탐욕적 선택 속성을 만족하지 않는다
- ◆ 예: 최적해는 이득 = 36을 가져다 주는 {A, E}지만, 탐욕해는 이득 = 24를 가져다 주는 {B, E, D}를 고른다

항목:



A

B

C

D

E

이득:

32

15

27

5

4

무게:

8kg

3kg

9kg

5kg

2kg

가치:

4

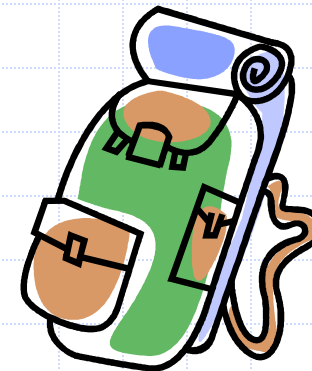
5

3

1

2

(kg 당)



배낭 10kg

탐욕해:

- B (3kg)
- E (2kg)
- D (5kg)

총이득: 24

최소신장트리 알고리즘

- ◆ Prim-Jarnik 알고리즘
- ◆ Kruskal 알고리즘
- ◆ Baruvka 알고리즘

모두 탐욕법으로 해결

Prim-Jarnik 알고리즘

◆ 탐욕 알고리즘

◆ 대상: 단순 연결 무방향 가중그래프

◆ 임의의 정점 s 를 택하여, s 로부터 시작하여 정점들을 (가상의) **배낭**에 넣어가며 배낭 안에서 MST T 를 키워 나간다 – s 는 T 의 루트가 된다

◆ 각 정점 v 에 라벨 $d(v)$ 를 정의 – $d(v)$ 는 배낭 안의 정점과 배낭 밖의 정점을 연결하는 간선의 무게

◆ 반복의 각 회전에서:

- 배낭 밖의 정점들 가운데 최소 $d(z)$ 라벨을 가진 정점 z 를 배낭에 넣는다
- z 에 인접한 정점들의 라벨을 갱신

Prim-Jarnik 알고리즘 (conti.)

- ◆ 배낭 밖의 정점들을 우선순위 큐에 저장
 - 키: 거리
 - 원소: 정점
- ◆ 보조 메소드
 - **Q.replaceKey**(e, k): 원소 e 의 키를 k 로 변경하고 우선순위 큐 Q 내의 e 의 위치를 갱신
- ◆ 각 정점 v 에 세 개의 라벨을 저장
 - 거리(distance): $d(v)$
 - 위치자(locator): 우선순위 큐에서의 v 의 위치
 - 부모(parent): $p(v)$, MST에서 v 의 부모를 향한 간선

Alg **PrimJarnikMST**(G)

input a simple connected weighted graph G with n vertices and m edges
output an MST T for G

1. **for each** $v \in G.vertices()$
 $d(v) \leftarrow \infty$
 $p(v) \leftarrow \emptyset$
2. $s \leftarrow$ a vertex of G
3. $d(s) \leftarrow 0$
4. $Q \leftarrow$ a priority queue containing all the vertices of G using d labels as keys
5. **while** ($!Q.isEmpty()$)
 {pull a vertex into the sack}
 $u \leftarrow Q.removeMin()$
 for each $e \in G.incidentEdges(u)$
 $z \leftarrow G.opposite(u, e)$
 if ($(z \in Q) \ \& \ (w(u, z) < d(z))$)
 $d(z) \leftarrow w(u, z)$
 $p(z) \leftarrow e$
 $Q.replaceKey(z, w(u, z))$

Prim-Jarnik 알고리즘이 탐욕법 알고리즘인 이유

◆ **구성:** 다양한 수치의 항목들 가운데 선택

- **직녀의 문제:** 다양한 이득을 주는 물건들 가운데 선택
- **MST 문제:** 다양한 무게의 간선들 가운데 선택

◆ **목표:** 총수치를 최대화 또는 최소화

- **직녀의 문제:** 배낭 안 물건들의 총이득을 최대화
- **MST 문제:** 배낭 안 간선들의 총무게를 최소화

◆ **탐욕적 해결:** 항상 최대/최소 수치 항목부터 배낭에 포함

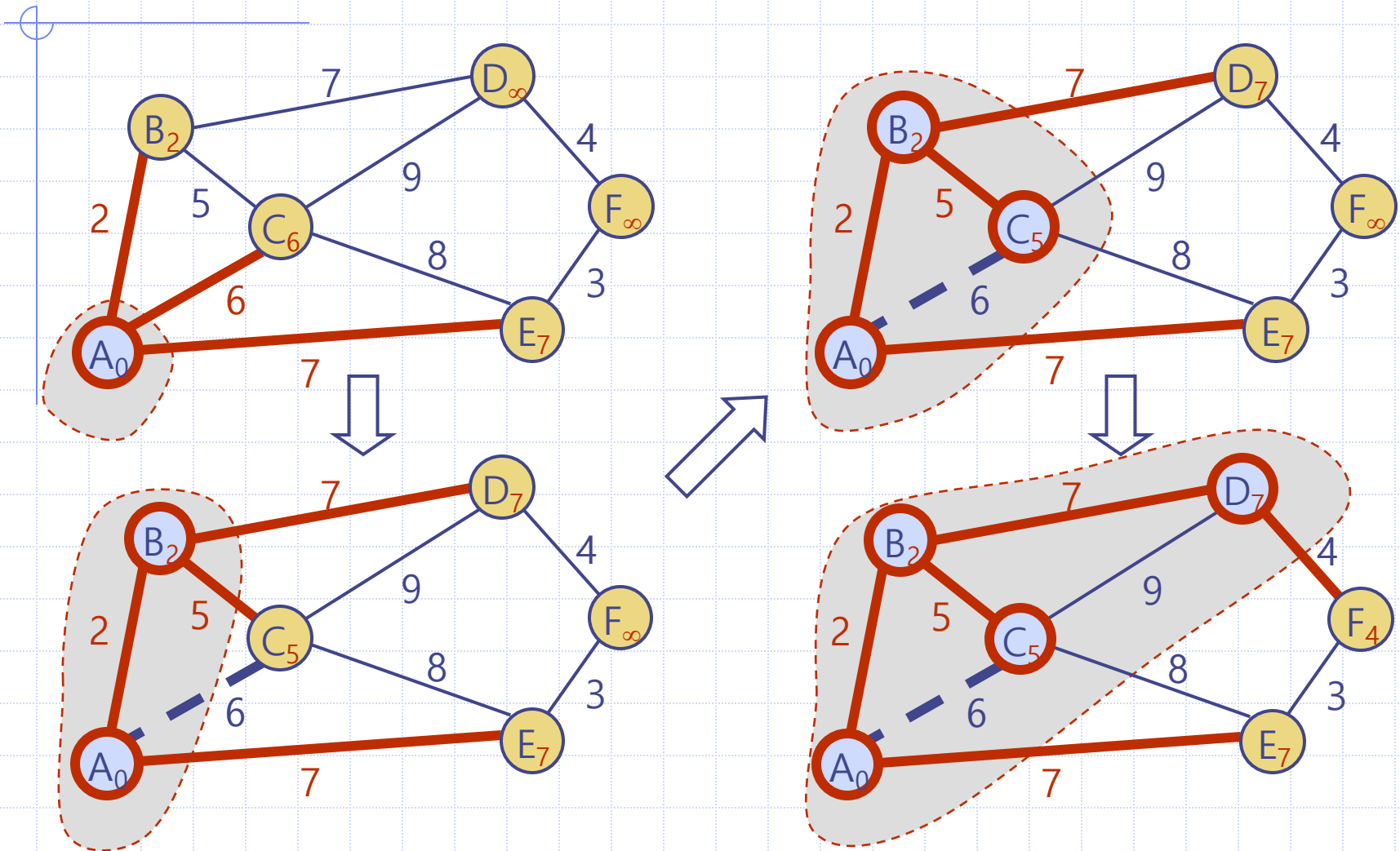
- **직녀의 문제:** 항상 최대 이득 물건부터 배낭에 포함
- **MST 문제:** 항상 최소 무게 간선부터 배낭에 포함

◆ 즉, 구성, 목표, 해결 절차 면에서 **탐욕법**의 일반 공식과 일치

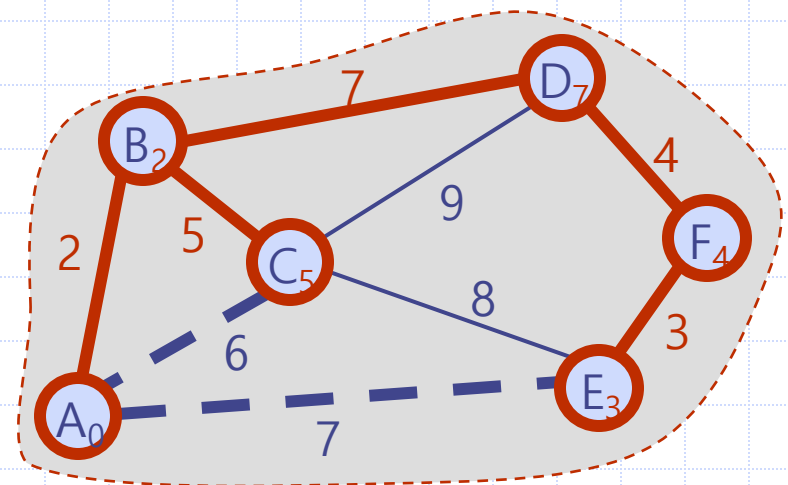
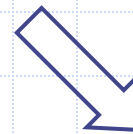
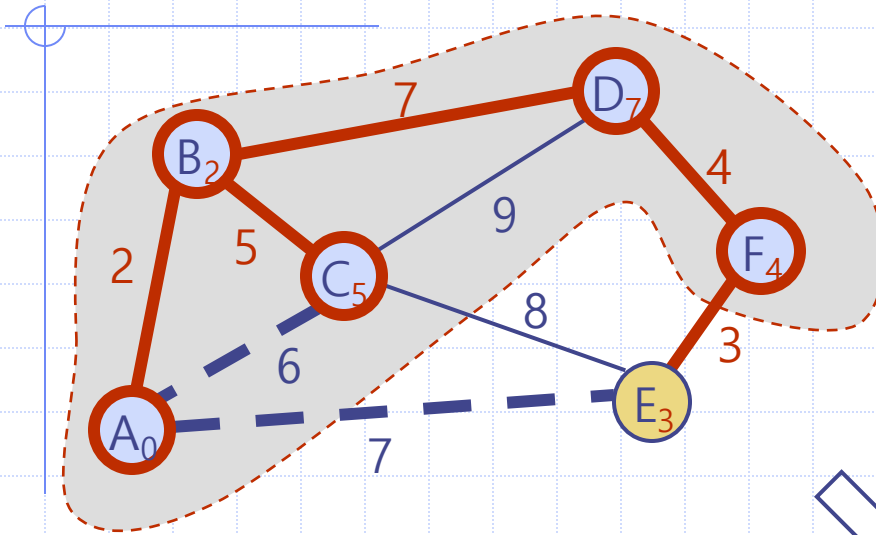
◆ **탐욕법을 사용하는 정당성**

- **탐욕적 선택 속성**
- 즉, 탐욕적 문제 구성과 목표 설정이 가능하더라도, **탐욕적 해결**로 목표를 성취할 수 있는 문제여야 함

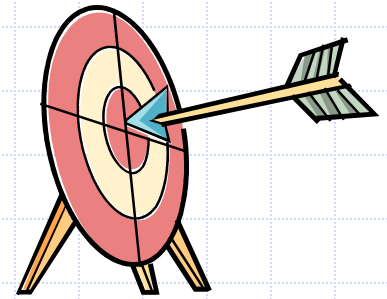
Prim-Jarnik 알고리즘 수행 예



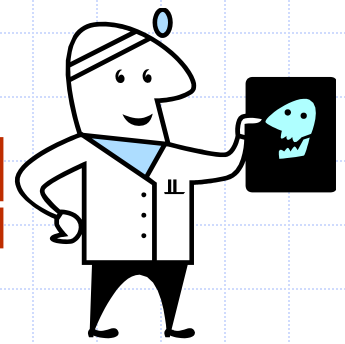
Prim-Jarnik 알고리즘 수행 예 (conti.)



Prim-Jarnik 알고리즘 정확성



- ◆ Prim-Jarnik 알고리즘은 반복의 각 회전에서 항상 배낭 C 안의 정점을 배낭 C 밖의 정점과 이어주는 **최소 무게**의 간선을 선택하므로, MST에 항상 타당한 간선을 추가한다
- ◆ 따라서 최소신장트리에 관한 **분할 속성**의 요건을 만족



Prim-Jarnik 알고리즘 분석

- ◆ 그래프 작업
 - 메소드 **incidentEdges**는 각 정점에 대해 한번 호출
- ◆ 라벨 작업
 - 정점 z 의 거리, 부모, 위치자 라벨을 $O(deg(z))$ 시간 읽고 쓴다
 - 라벨을 읽고 쓰는데 $O(1)$ 시간 소요
- ◆ 우선순위 큐 (힙으로 구현할 경우) 작업
 - 각 정점은 우선순위 큐에 한번 삽입되고 한번 삭제: 삽입과 삭제에 각각 $O(\log n)$ 시간 소요
 - 우선순위 큐 내의 정점 w 의 키는 최대 $deg(w)$ 번 변경: 각 키 변경에 $O(\log n)$ 시간 소요
- ◆ 그래프가 인접리스트 구조로 표현되어 있다면, **Prim-Jarnik** 알고리즘은 $O((n + m) \log n)$ 시간에 수행
 - 참고: $\sum_v deg(v) = 2m$
- ◆ 단순 연결그래프에서 $n = O(m)$ 이므로, 실행시간: $O(m \log n)$

Kruskal 알고리즘

◆ 탐욕 알고리즘

◆ 알고리즘을 위한 초기 작업

- 모든 정점을 각각의 독자적인 (실제의) 배낭에 넣는다
- 배낭 밖의 간선들을 우선순위 큐에 저장

- ◆ 키: 무게
- ◆ 원소: 간선

- 비어 있는 MST T 를 초기화

◆ 반복의 각 회전에서:

- 두 개의 다른 배낭 속에 양끝점을 가진 최소 무게의 간선을 MST T 에 포함하고 두 배낭을 하나로 합친다

◆ 반복이 완료되면:

- MST T 를 포함하는 한 개의 배낭만 남는다

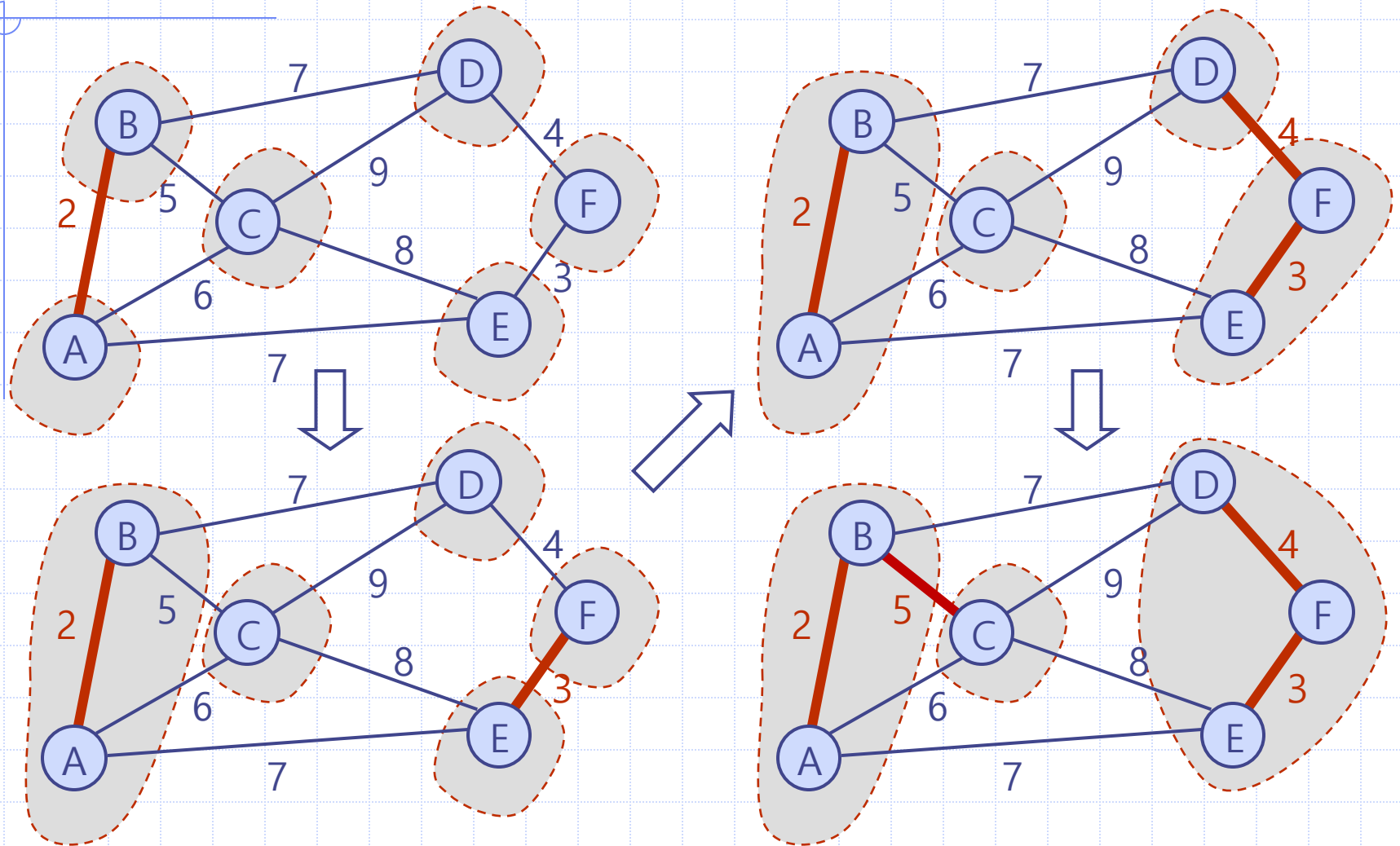
Alg *KruskalMST*(G)

input a simple connected weighted graph G with n vertices and m edges

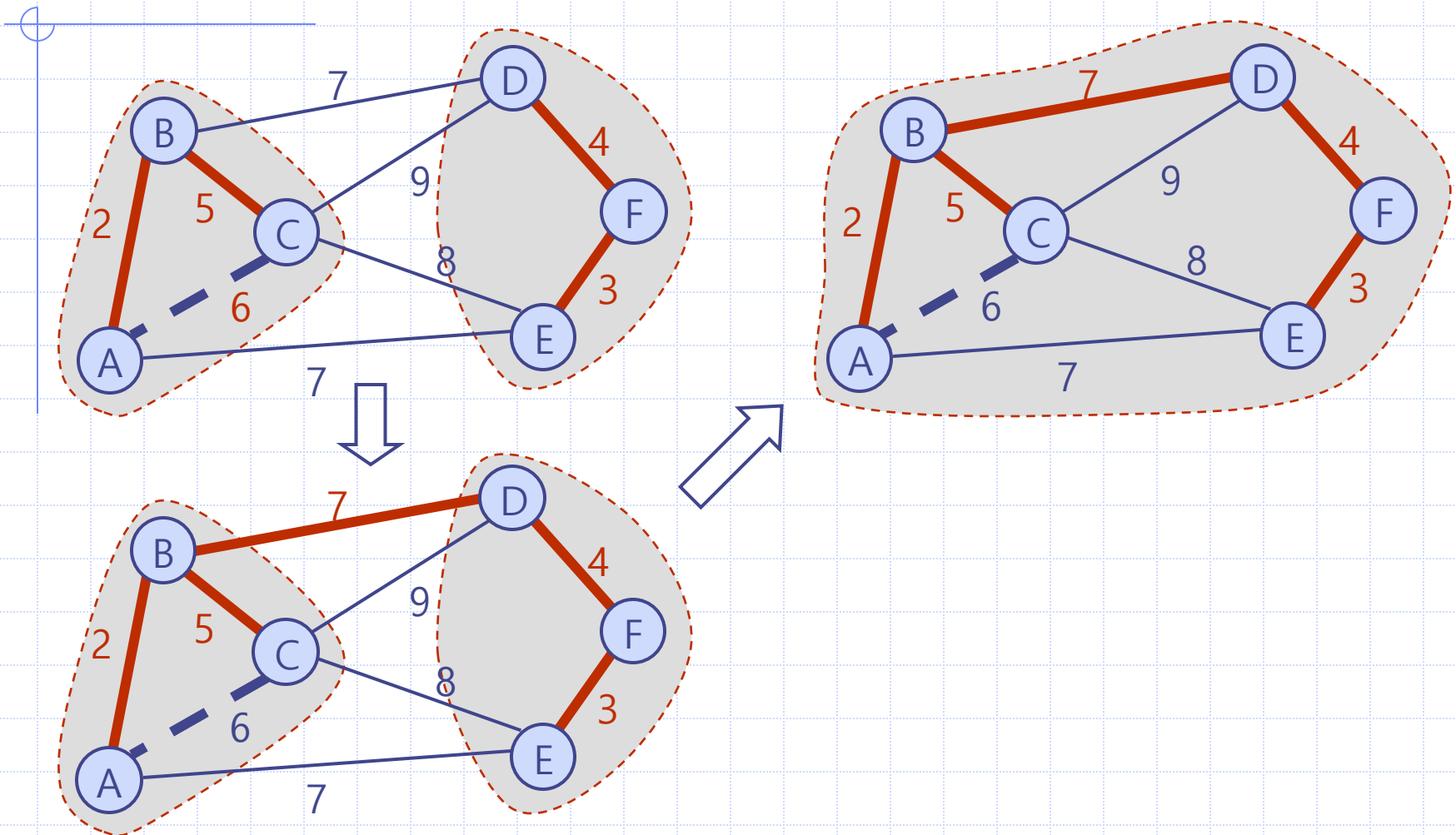
output an MST T for G

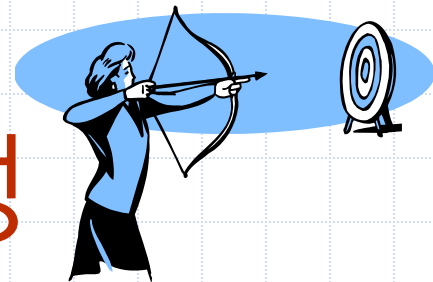
1. **for each** $v \in G.vertices()$
 define a $Sack(v) \leftarrow \{v\}$
2. $Q \leftarrow$ a priority queue containing all the edges of G using weights as keys
3. $T \leftarrow \emptyset$
4. **while** (T has fewer than $n - 1$ edges)
 $(u, v) \leftarrow Q.removeMin()$
 if ($Sack(u) \neq Sack(v)$)
 Add edge (u, v) to T
 Merge $Sack(u)$ and $Sack(v)$
5. **return** T

Kruskal 알고리즘 수행 예



Kruskal 알고리즘 수행 예 (conti.)

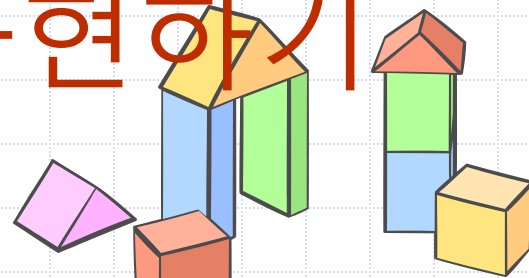




Kruskal 알고리즘 정확성

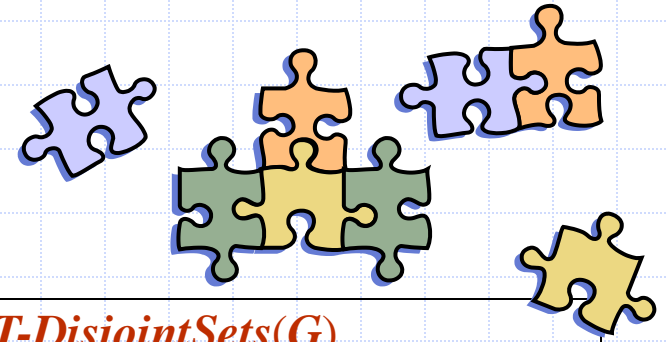
- ◆ **Kruskal** 알고리즘의 정확성은 **분할 속성**으로부터 유도
- ◆ **Kruskal** 알고리즘은 반복의 회전마다 간선 (u, v) 를 MST T 에 추가
- ◆ 이때 정점 집합 V 가 u 를 포함하는 배낭 V_1 과, V_1 에 속하지 않은 나머지 정점들을 모두 포함하는 배낭 V_2 로 **분할**되었다고 보면,
- ◆ 이는 모든 정점이 두 부분으로 분리된 **분할** 상태
- ◆ Q 로부터 간선들을 무게 순서로 추출하고 있으므로, (u, v) 는 양끝점이 각각 V_1 과 V_2 에 속한 **최소 무게의 간선**
- ◆ 따라서, **Kruskal** 알고리즘은 항상 **타당한** MST 간선을 추가한다

Kruskal 알고리즘을 구현하기 위한 데이터구조



- ◆ **Kruskal** 알고리즘은 **인접 정보**를 사용하지 않는다
 - 인접 정보(부착리스트 또는 인접행렬)가 생략된, 즉 **간선리스트** 구조로 표현된 그래프에서 수행 가능
- ◆ **Kruskal** 알고리즘은 **트리들의 숲**을 유지
 - 각 트리들을 **리스트**로 구현된 **분리집합**에 저장하고 각 원소에 소속 집합을 가리키는 참조를 저장
 - ◆ 작업 **find**(u)는 u 가 소속된 집합을 반환: $O(1)$ 시간 소요
 - ◆ 작업 **union**(u, v)은 크기가 작은 집합의 원소들을 큰 집합 리스트로 옮기며 원소들의 참조를 갱신: $O(\min(n_u, n_v))$ 시간 소요 – 여기서 n_u 와 n_v 는 각각 u 와 v 를 저장한 집합의 크기

분리집합에 기초한 Kruskal 알고리즘



◆ Kruskal 알고리즘의
분리집합에 기초한
버전은 **find**로 소속
집합 검사를,
union으로 배낭
합치기를 수행

Alg *KruskalMST-DisjointSets*(G)

input A simple connected weighted graph
 G with n vertices and m edges

output An MST T for G

1. Let D be a disjoint set of the vertices of G ,
where each vertex forms a separate set
2. Let Q be a priority queue storing the edges
of G , sorted by their weights
3. Let T be an initially-empty tree
4. **while** ($!Q.isEmpty()$)
 $(u, v) \leftarrow Q.removeMin()$
 if ($D.find(u) \neq D.find(v)$)
 Add (u, v) to T
 $D.union(u, v)$
5. **return** T



Kruskal 알고리즘 분석

- ◆ 우선순위 큐 초기화 작업
 - 힙을 사용하여 우선순위 큐 Q 를 구현하면, 반복적인 삽입 방식에 의해서는 $O(m \log m)$ 시간에, 상향식 힙 생성 방식에 의해서는 $O(m)$ 시간에 Q 생성 가능
- ◆ 반복의 각 회전에서
 - 최소 무게의 간선을 $O(\log m)$ 시간에 삭제 가능 – 여기서, G 가 단순그래프이므로, $O(\log m) = O(\log n)$
 - 각 배낭을 위한 분리집합을 리스트로 구현하면, $\text{find}(u) \neq \text{find}(v)$ 검사에 $O(1)$ 시간
 - 두 개의 배낭 $\text{Sack}(u)$, $\text{Sack}(v)$ 를 합치는데 $O(\min(|\text{Sack}(u)|, |\text{Sack}(v)|))$ 시간 – 원소가 새 배낭으로 합쳐질 때마다 최소 두 배 크기의 배낭으로 합쳐지므로, 각 원소는 최대 $\log n$ 번 이동 – 이를 모든 정점에 대해 누적하면 $O(n \log n)$
- ◆ 총 반복 회수: 최악의 경우 m 번
- ◆ 그러므로 **Kruskal** 알고리즘은 $O((n + m) \log n)$ 시간에 수행
- ◆ G 가 단순 연결그래프인 경우 $n = O(m)$ 이므로, 실행시간은 $O(m \log n)$

Baruvka 알고리즘

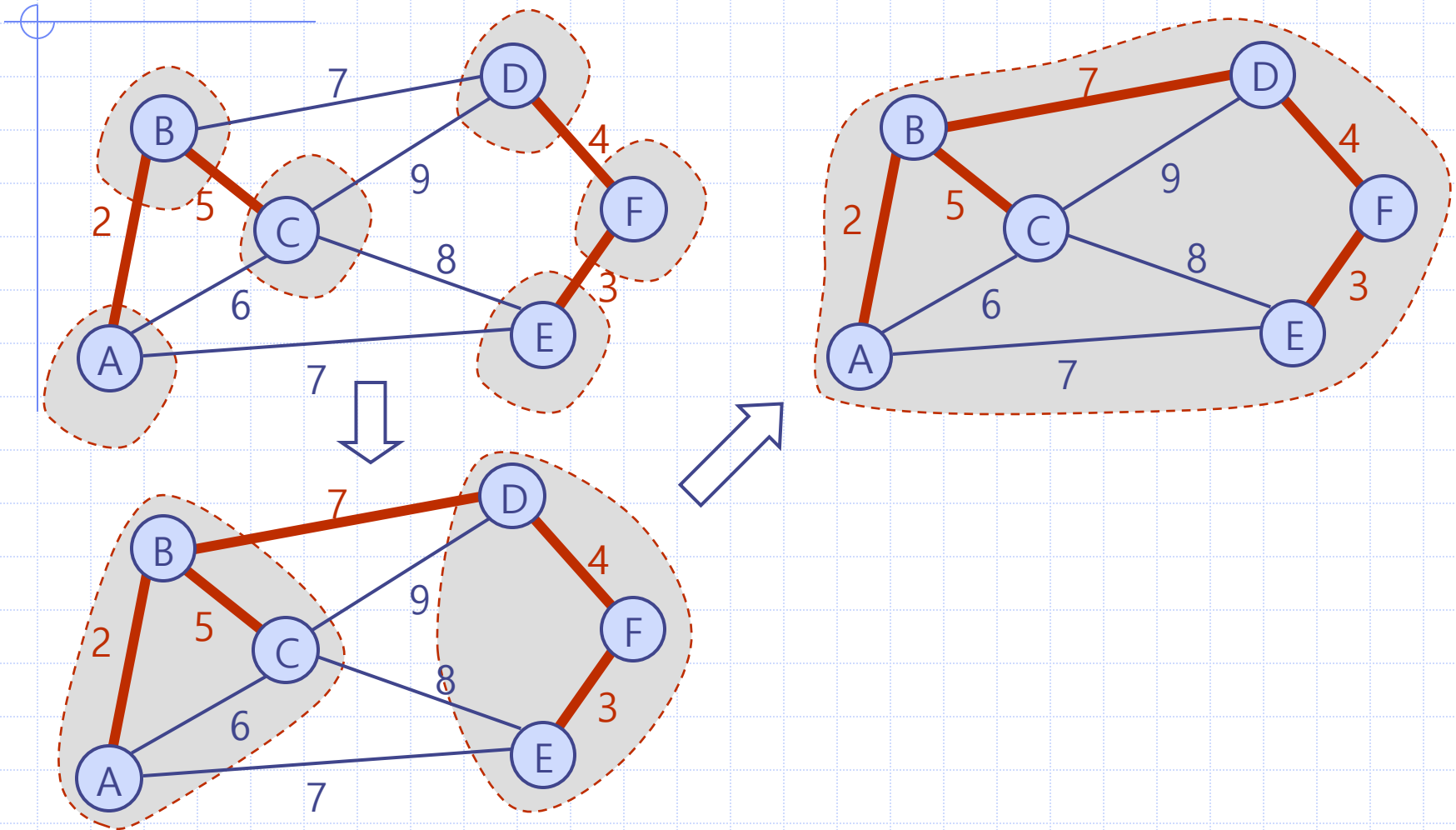
- ◆ a.k.a. Sollin 알고리즘
- ◆ Kruskal이나 Prim-Jarnik 알고리즘과 달리, 우선순위 큐를 사용하지 않는다
- ◆ Kruskal 알고리즘과 마찬가지로, Baruvka 알고리즘도 모든 정점을 각각의 독자적인 (실제의) 배낭에 넣고 시작
- ◆ Kruskal이나 Prim-Jarnik 알고리즘이 반복의 각 회전에서 한 개의 간선을 취함으로써 배낭을 키워 나가는 것과 달리, 한꺼번에 여러 개의 간선을 취하여 여러 수의 배낭을 동시에 키워 나감

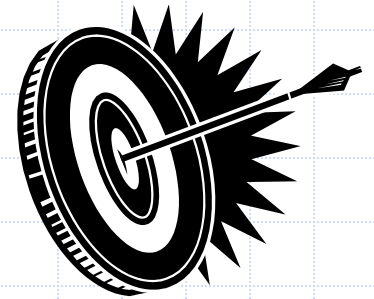
Alg *BaruvkaMST*(G)

input A simple connected weighted graph $G = (V, E)$ with n vertices and m edges
output An MST T for G

1. $T \leftarrow V$ {just the vertices of G }
2. **while** (T has fewer than $n - 1$ edges)
 for each connected component C_i in T
 Let edge e be the smallest-weight edge from C_i to another component in T
 if (e is not already in T)
 Add edge e to T
3. **return** T

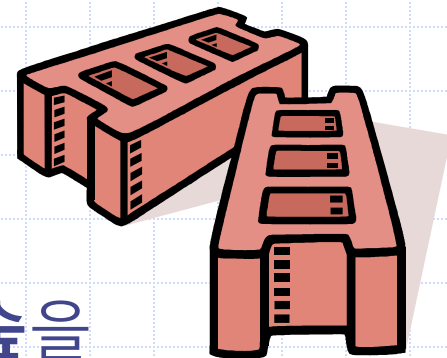
Baruvka 알고리즘 수행 예





Baruvka 알고리즘 정확성

- ◆ Baruvka 알고리즘 반복의 각 단계에서는, 현재의 MST T 의 각 연결요소 C_i 사이를 교차하는 최소 무게의 간선들을 취한다
- ◆ 여기서 v 가 C_i 에 속한 정점들과, C_i 바깥의 정점들로 **분할**되었다고 가정하면,
- ◆ C_i 를 위해 선택된 간선 e 는 e 가 MST에 반드시 포함되어야 한다는 MST에 관한 **분할 속성**의 요건을 만족
- ◆ 따라서 반복의 각 단계에서 취해지는 간선들은 모두 타당한 선택이 된다



Baruvka 알고리즘 구현

- ◆ 간선 삽입에 따른 연결요소(트리)들의 숲을 유지: 인접리스트를 사용하면 $O(1)$ 시간에 처리 가능
- ◆ 연결요소들을 찾기 위해 숲을 순회: DFS를 사용하면 $O(n)$ 시간에 수행 가능
- ◆ 정점이 소속된 연결요소 표시: 각 정점에 라벨 정의하여 표시
- ◆ 연결요소 C_i 에 인접한 최소 무게의 간선 찾기: 그래프 G 의 인접리스트에서 C_i 에 속한 정점들을 조사



Baruvka 알고리즘 분석

◆ 반복의 각 회전에서

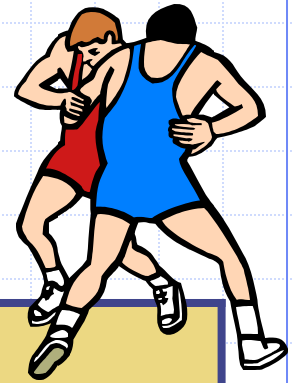
- 각 연결요소 C_i 사이를 교차하는 최소 무게의 간선들을 찾기 위해, 각 C_i 의 인접리스트를 전면적으로 탐색
 - ◆ 이때 그래프 G 의 모든 간선 (v, u) 를, (각 정점은 자신이 소속된 연결요소 라벨을 가지므로) 한 번은 v 에 대해 또 한 번은 u 에 대해, 합계 두 번 조사하므로 총 $O(m)$ 시간 소요
- 모든 정점을 재라벨: $O(n)$
- T 의 모든 간선을 순회: $O(n)$
- 따라서 반복 1회전의 수행 시간은 $O(m)$ ($\because n \leq m$)

◆ 반복회수

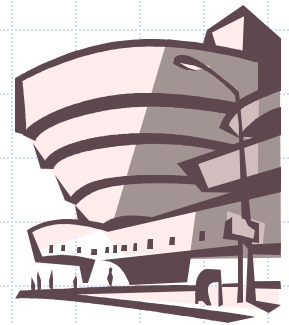
- 반복의 각 회전에서, 각 배낭으로부터 나오는 하나의 간선을 고르고, 각각의 새로운 연결요소들을 합쳐 새 배낭을 만든다
- 즉, 각각의 기존 배낭이 최소한 개의 다른 기존 배낭과 합쳐진다
- 따라서 반복의 회전마다 배낭의 총수는 최소 절반으로 줄어든다
- 따라서 총 반복회수: $O(\log n)$

◆ 그러므로 Baruvka 알고리즘의 실행시간: $O(m \log n)$

MST 알고리즘 비교

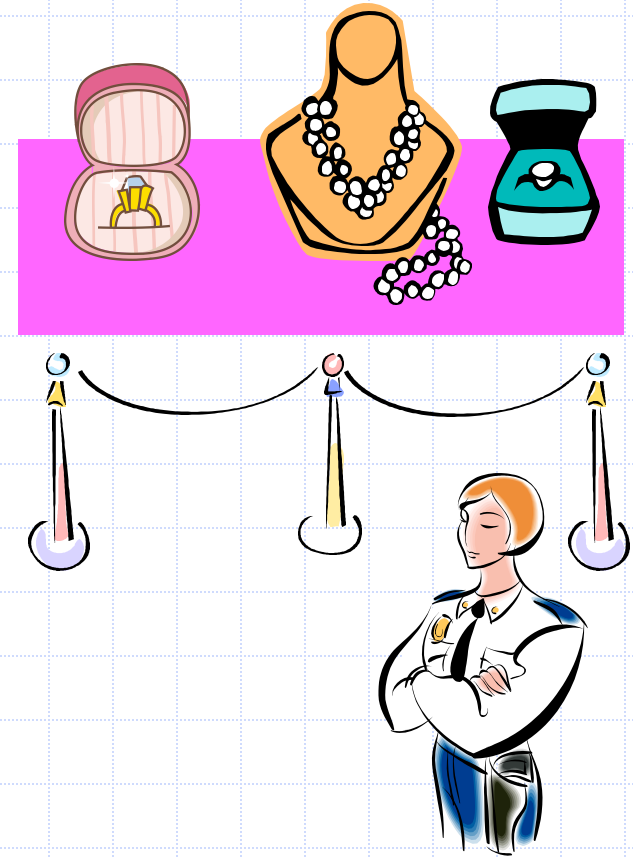


알고리즘	주요 전략	수행 시간	외부 데이터구조
Prim-Jarnik	탐욕	$O(m \log n)$	◆ 정점들을 저장하기 위한 우선순위 큐
Kruskal	탐욕	$O(m \log n)$	◆ 간선들을 저장하기 위한 우선순위 큐 ◆ 배낭들을 구현하기 위한 분리집합 (리스트로 구현 가능)
Baruvka	탐욕	$O(m \log n)$	◆ 연결요소를 표현하기 위한 데이터구조 필요

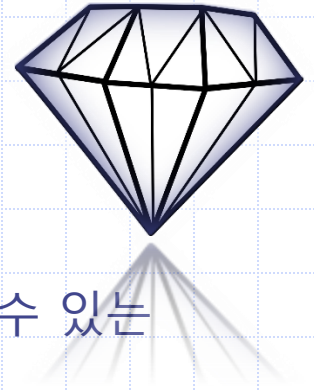


응용문제: 보석전시회

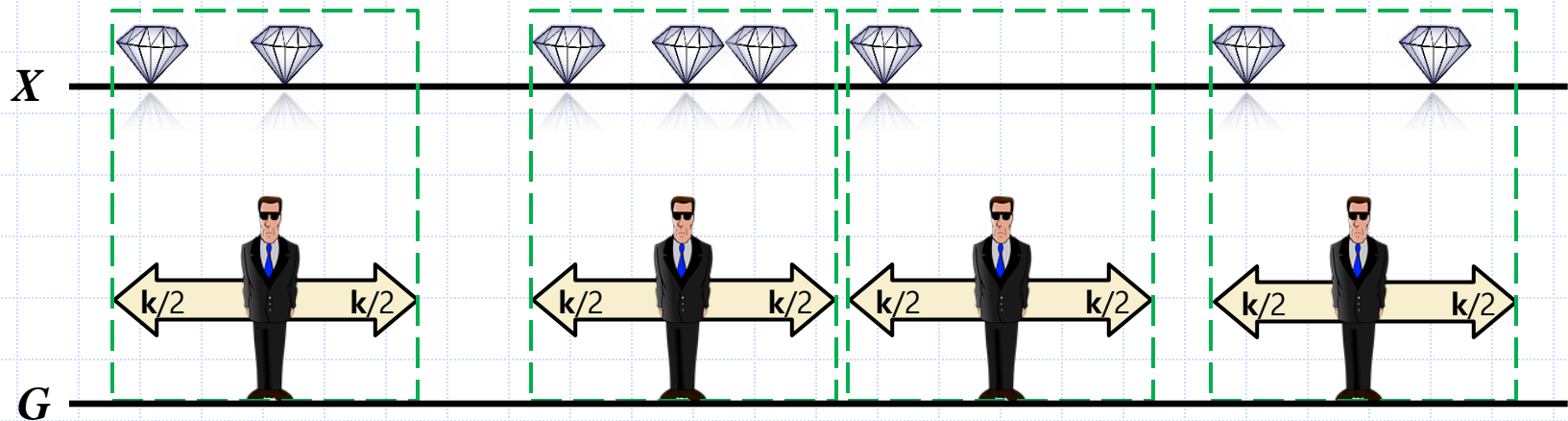
- ◆ 보석전시회가 열리고 있는 긴 복도를 표현하는 1차원 축 L 이 있다
- ◆ 복도축 L 을 따라 보석들이 놓인 위치를 나타내는 실수들의 집합 $X = \{x_0, x_1, \dots, x_{n-1}\}$ 가 주어졌다
- ◆ 한 명의 경비가 자신이 서있는 위치에서 좌우 양쪽으로 최대 k 거리까지 커버할 수 있다고 가정하자 – 따라서 좌우 한쪽으로는 $k/2$
- ◆ 최소 인원의 경비를 사용하여 X 의 위치들에 놓인 보석들을 모두 지킬 수 있도록 하는 경비들의 배치를 계산할 알고리즘 **gemGuard**(X, k)를 의사코드로 작성하라



해결



- ◆ L 상의 모든 지정된 위치를 최소 개수의 k 간격으로 커버할 수 있는 탐욕 알고리즘을 사용
- ◆ 우선 X 의 원소들(즉, 위치들)을 오름차순으로 정렬
- ◆ 다음, X 의 첫 원소 x_0 에 대해 x_0 로부터 $k/2$ 위치에 경비를 세운다 – 이 경비는 자신의 좌우 $k/2$ 거리 내에 있는 보석들을 모두 커버
- ◆ 커버되지 않은 다음 지점이 x_i 라면, x_i 로부터 위와 동일한 커버링 절차를 반복
- ◆ X 의 모든 원소들을 커버할 때까지 이 절차를 반복



해결 (conti.)

Alg *gemGuard*(X, k)

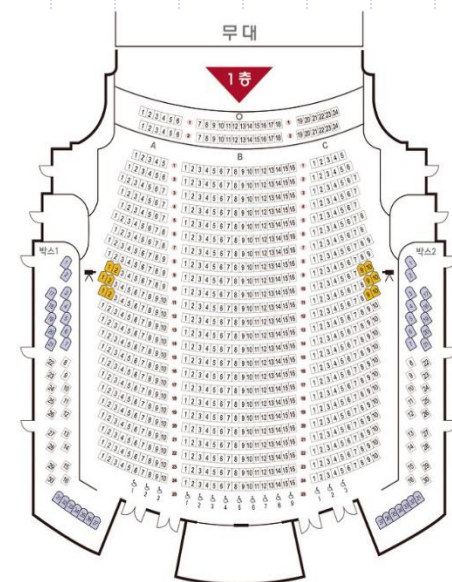
input list X of points on L , interval k

output list G of guard points

1. Sort the points in X in ascending order
2. $G \leftarrow$ empty list
3. $x \leftarrow X.removeFirst()$
4. $g \leftarrow x + k/2$
5. $G.addLast(g)$ { the first guard }
6. while ($!X.isEmpty()$)
 $x \leftarrow X.removeFirst()$
 if ($x - g > k/2$) { uncovered }
 $g \leftarrow x + k/2$
 $G.addLast(g)$ { next guard }
7. return $G.elements()$

- ◆ **목표:** 최대화해야 할 목표(최소의 경비로 최다의 보석을 커버)
- ◆ **구성:** 다양한 수치의 항목들(보석들의 위치)
- ◆ **탐욕적 해결:** 출발 이후 최소 거리에 있는 원소부터 반복적으로 처리
- ◆ **탐욕적 선택 속성:** 최적해 구함

응용문제: 공연홀 좌석배정



- ◆ 당신이 살고 있는 Sin City의 시립 공연홀의 관장은 오로지 돈, 즉 티켓 판매고에만 관심이 있다.
- ◆ 단체관람객이 공연을 보기 위해서는 그 단체의 모든 멤버가 좌석을 배정받아야만 하며 그렇지 않으면 관람을 포기한다
- ◆ 선착순에 의해 티켓을 판매하는 현재 시스템에서는 좌석이 남아 있더라도 남은 좌석 수보다 큰 단체가 관람을 원할 경우 이들을 수용할 수 없다
- ◆ 관장은 당신을 판매담당 고문으로 영입하여 티켓 판매고를 올리는 문제를 일임했다 - 여러가지 생각 끝에 당신은 새로운 티켓 판매전략을 제안했다
- ◆ 새 전략: “선착순으로 좌석을 배정하기 보다는 큰 단체에게 좌석을 우선 배정하고 그 후 크기가 작은 순서로 단체에게 배정하며 마지막에는 개인관람객(즉, 1인 단체)에게 배정”

응용문제: 공연홀 좌석배정 (conti.)

- ◆ 티켓을 사려는 단체들:

$G[1..m] = (g_1, g_2, \dots, g_m)$

- $g_i \geq 1$ 는 단체 i 의 크기

- ◆ 총 관람석 수: n

- ◆ 탐욕적 좌석배정 알고리즘 **seat**(G, n):

- **admit**(i): 단체 i 를 입장시킴
- **reject**(i)는 단체 i 를 좌석 부족으로 돌려보냄

Alg **seat**(G, n)

input array $G[1..m]$ of integers, integer n

output number of people admitted

1. $admitted \leftarrow 0$
2. $G \leftarrow \text{Sort } G \text{ largest to smallest}$
3. $remaining \leftarrow n$
4. **for** $i \leftarrow 1$ **to** m
 - if** ($G[i] \leq remaining$)
 - $admit(i)$
 - $remaining \leftarrow remaining - G[i]$
 - $admitted \leftarrow admitted + G[i]$
 - else**
 - $reject(i)$
5. **return** $admitted$

응용문제: 공연홀 좌석배정 (conti.)

- A. 탐욕 알고리즘 **seat**가 **최적** 알고리즘은 아니라는 것을 **예**를 들어 설명하라
- B. 만약 k 명의 사람을 입장시키는 것이 **최적**해라면, **seat** 알고리즘은 적어도 $k/2$ 명의 사람을 입장시킨다는 것을 설명하라

해결 A

- ◆ 예: $G = ((n + 2)/2, n/2, n/2)$
 - 구체적 예: $n = 100$ 에 대해 $G = (51, 50, 50)$
- ◆ G 에 대해 탐욕 알고리즘 seat는 $(n + 2)/2$ 크기의 단체를 우선 입장시킬 것이고 나머지 단체들은 모두 입장시키지 못할 것이다
- ◆ 반면, 최적 알고리즘이라면 $n/2$ 크기의 두 단체만을 입장시킴으로써 n 개의 좌석을 모두 채울 것이다

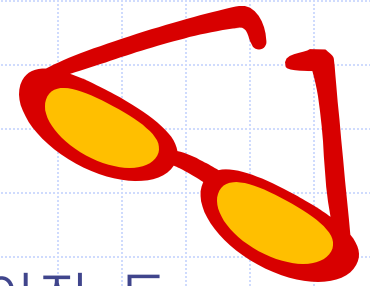
해결 B

- ◆ **seat** 알고리즘은 항상 적어도 $k/2$ 명의 사람을 입장시킨다는 명제에 대한 증명 – 여러 가지 경우로 나누어 고려하여 어떤 경우에도 명제가 성립함을 증명
- ◆ 아래 둘 중 하나다 – **seat** 알고리즘이 주어진 G 의 모든 단체를:
 - 입장시키든가 (전자)
 - 입장시키지 못하든가 (후자)
- ◆ (전자) **seat** 알고리즘이 모든 단체를 입장시킨다면:
 - k 명을 모두 입장시킨 것이니 적어도 $k/2$ 명을 입장시킨 것이 된다 – 따라서 증명됨

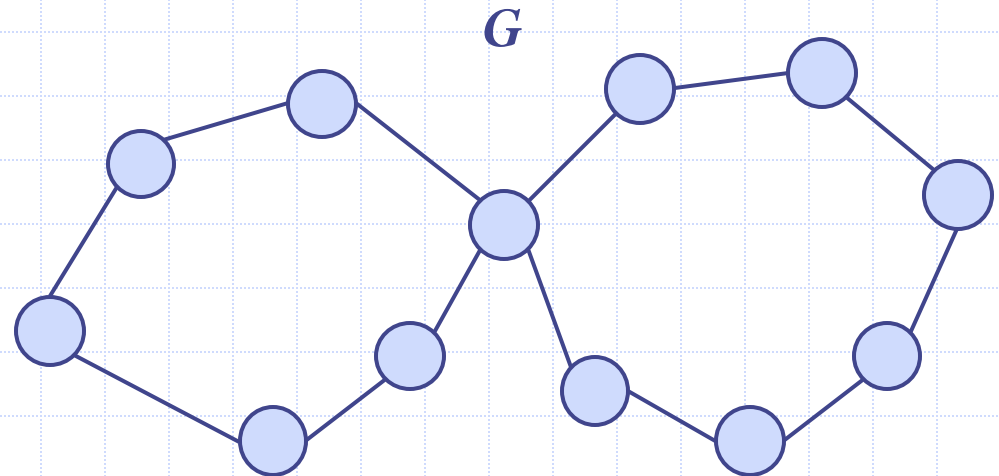
해결 B

- ◆ (후자) 모든 단체를 입장시키지는 못한다면:
 - 입장시키지 못한 단체 g_i 가 적어도 한 개 존재하며, 아래 둘 중 하나다
 - ◆ $g_i \geq k/2$
 - ◆ $g_i < k/2$
 - $g_i \geq k/2$ 경우
 - ◆ 탐욕 알고리즘 **seat**의 속성 상 g_i 보다 큰 단체를 먼저 입장시켰을 것이다 – 그런 단체가 없다면 g_i 를 먼저 입장시킬 것이 확실하기 때문
 - ◆ 즉, 그런 단체가 있든 없든 **seat** 알고리즘이 최소 $k/2$ 명의 사람을 입장시킨다 – 명제가 증명됨
 - $g_i < k/2$ 경우
 - ◆ 이는 **seat** 알고리즘 수행의 어떤 시점에서 $remaining < g_i < k/2$ 이 성립했다는 의미 – 즉, 그 시점에 남은 좌석 수가 $k/2$ 보다 작았음
 - ◆ $k \leq n$ 이므로, 이 말은 바로 그 시점에 적어도 $k/2$ 명의 관객이 이미 입장했다는 말과 같다 – 명제가 증명됨

응용문제: 8자-모양 그래프에서의 MST



- ◆ 아래 그림처럼, n 개의 정점과 m 개의 간선으로 이루어진, 두 개의 사이클이 한 정점에서 만난 형태의 단순 연결 무방향 가중그래프 G 가 있다
- ◆ **전제:** 두 개의 사이클은 반드시 동일한 개수의 정점으로 이루어진 것은 아니며, n 에 대해서도 아무런 가정이 없다
- ◆ G 의 **최소신장트리**를 구하는 **효율적인** 알고리즘을 작성하고 실행시간을 구하라



해결

◆ 8자-모양
그래프에 대한
최소신장트리는
기존의 MST
알고리즘을
사용하지 않고
지름길 메소드로
구할 수 있다

◆ 실행시간: $O(n)$

Alg *eight-ShapedMST*(G)

input a simple connected weighted graph G
consisting of two cycles with n vertices
and m edges

output an MST T for G

1. $e_1 \leftarrow$ the maximum-weight edge in the left
cycle of G $\{O(n)\}$
2. $e_2 \leftarrow$ the maximum-weight edge in the right
cycle of G $\{O(n)\}$
3. $T \leftarrow E - (\{e_1\} \cup \{e_2\})$
4. **return** T $\{Total\ O(n)\}$