제 4장 그리디 알고리즘

최적화 (optimization) 문제

- 정의
  - 가능한 해들 중에서 가장 좋은 (최대 또는 최소) 해를 찾는 문제
- 예 <> feasible
  - 선형계획법
    - A,B 두 상품을 생산
      - 상품 A는 개당 2원, B는 개당 5원의 이익
      - 상품 A를 생산하는 데 9개의 재료와 3시간 동안 기계를 사용
      - 상품 B를 생산하는 데 5개의 재료와 4시간 동안 기계를 사용
      - 재료는 최대 300개, 기계 가동 시간은 최대 200시간
      - 상품 A는 최소 5개 이상을 생산해야만 한다
    - 최대의 이익을 산출해 내는 상품 A와 B의 생산량을 결정하라.

결정 변수: 제품 A의 생산량 => x_1
제품 B의 생산량 => x_2

목적함수: $Max\ 2x_1 + 5x_2$ ← 이익의 최대화

subject to

제약식:

$s.t.\ 3x_1 + 4x_2 \leq 200$ ← 기계 가동시간 제약

$9x_1 + 5x_2 \leq 300$ ← 재료 사용량 제약

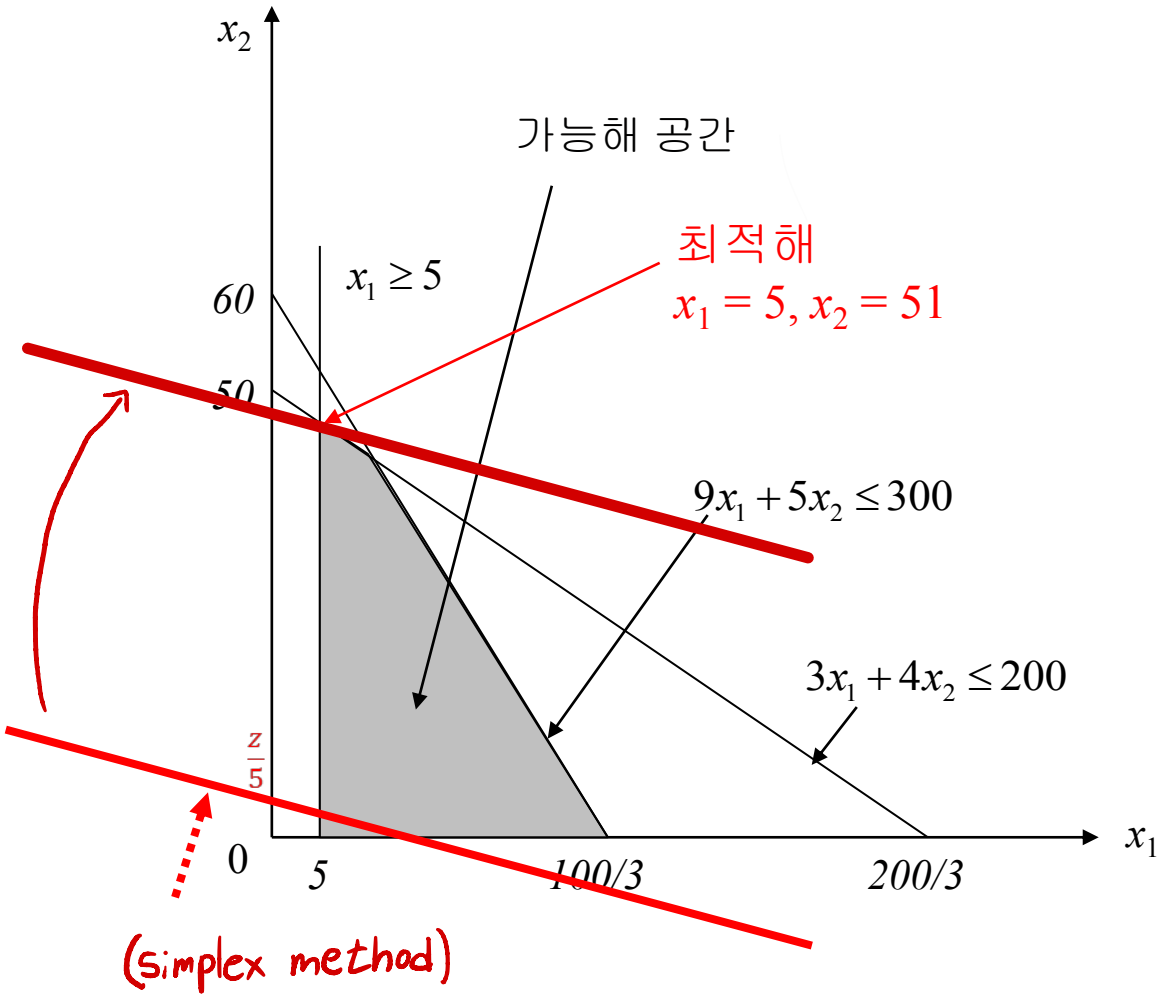
$x_1 \geq 5$ ← A의 최소 생산량 제약

$x_2 \geq 0$ ← 비음수인 해만을 구함

[예] 제품배합 문제의 기하학적 접근 – (계속)

$$Max\ \ 2x_1 + 5x_2$$
$$s.t.\ \ 3x_1 + 4x_2 \le 200$$
$$9x_1 + 5x_2 \le 300$$
$$x_1 \ge 5$$
$$x_2 \ge 0$$

$$Maximize\ z = 2x_1 + 5x_2$$
$$x_2 = -\frac{2}{5}x_1 + \frac{z}{5}$$



그리디 (Greedy) 알고리즘

- 그리디 알고리즘은 최적화 문제를 해결.
- 욕심쟁이 방법, 탐욕적 방법, 탐욕 알고리즘 등으로 불리기도 한다.
  - 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택
  - 이러한 선택을 '근시안적'인 선택이라고 말하기도 함
  - 근시안적인 선택으로 부분적인 최적해를 찾고, 이들을 모아서 문제의 최적해(Optimal Solution)를 얻음
- 그리디 알고리즘은 일단 한번 선택하면, 이를 절대로 번복하지 않음.
  - 선택한 데이터를 버리고 다른 것을 취하지 않는다.
  - 이러한 특성 때문에 대부분의 그리디 알고리즘들은 매우 단순
  - 또한 제한적인 문제들 만이 그리디 알고리즘으로 해결됨
  - 이러한 단점을 해결할 수 있는 방법이 동적계획법 (Dynamic Programming)

4.1 동전 거스름돈

- 동전 거스름돈 (Coin Change) 문제
  - 가장 간단하고 효율적인 방법은 남은 액수를 초과하지 않는 조건하에 '욕심내어' 가장 큰 액면의 동전을 취하는 것
- 최소 동전 수를 찾는 그리디 알고리즘
  - 단, 동전의 액면은 500원, 100원, 50원, 10원, 1원이다.

CoinChange

입력: 거스름돈 액수 W
출력: 거스름돈 액수에 대한 최소 동전 수

1. change=W, $n_{500}=n_{100}=n_{50}=n_{10}=n_1=0$ // n_{500}, n_{100}, n_{50}, n_{10}, n_1은 각각의 동전 카운트
2. while (change ≥ 500) change = change-500, n_{500}++ // 500원짜리 동전 수를 1 증가
3. while (change ≥ 100) change = change-100, n_{100}++ // 100원짜리 동전 수를 1 증가
4. while (change ≥ 50) change = change-50, n_{50}++ // 50원짜리 동전 수를 1 증가
5. while (change ≥ 10) change = change-10, n_{10}++ // 10원짜리 동전 수를 1 증가
6. while (change ≥ 1) change = change-1, n_1++ // 1원짜리 동전 수를 1 증가
7. return (n500 + n100 + n50 + n10 + n1) // 총 동전 수를 리턴한다.

- 그리디 알고리즘의 근시안적인 특성
  - 남아있는 거스름돈인 change에 대해 가장 높은 액면의 동전을 거스르며,
  - 동전을 처리하는 동안 자신보다 작은 액면의 동전은 전혀 고려하지 않음

Greedy Method의 예

- 동전들을 이용하여 거스름 돈 41 센트를 만든다. 이 때 사용되는 동전의 개수를 최소로 하여야 한다. 25센트, 10센트, 5센트, 1센트 짜리 동전이 주어졌다고 가정한다.
- 거슬러 주어야 하는 값을 넘지 않으면서 가장 큰 동전을 무조건 선택한다.
  - 가장 큰 25센트를 먼저 선택하면 16센트가 남고
  - 여기서 10센트를 선택하면 6센트가 남고
  - 5센트를 선택하면 1센트가 남고
  - 1센트를 선택하면 값이 맞아 떨어지므로 해가 구해짐.

Greedy Method의 함정(1)

- 41센트의 거스름 돈을 만들되 이번에는 25센트, 10센트, 4센트 짜리 동전들이 주어져 있다고 가정해보자.
  1) 41 - 25 = 16
  2) 16 - 10 = 6
  3) 6 - 4 = 2
- Greedy algorithm을 따라 25센트, 10센트, 4센트를 선택하면 남은 2센트를 거슬러 주는 것이 불가능하다.
- 실제 답은 다음과 같이 해야 함
  - 41 - 25 = 16
  - 16 - (4 * 4) = 0
- 선택을 한 후에 그 선택을 되짚어 보는 과정이 필요한데 greedy algorithm 은 그러한 과정을 전혀 거치지 않는다. 따라서 수학적 문제에 대해 항상 최적의 해를 찾는다고 할 수 없다. 많은 수학적 문제들은 greedy algorithm 으로 해를 찾을 수 없다.
- 이러한 문제점을 해결한 것이 동적 프로그래밍이다

Greedy Method의 함정(2)

- 만일 한국은행에서 160원짜리 동전을 추가로 발행한다고 가정
  - CoinChange 알고리즘이 항상 최소 동전 수를 계산할 수 있을까?

CoinChange 알고리즘의 결과

최소 동전의 거스름돈

- CoinChange 알고리즘은 항상 최적의 답을 주지는 못한다.
  - 어떤 경우에도 최적해를 찾는 동전 거스름돈 알고리즘
  - 동적 계획 알고리즘

Greedy Method의 함정(3)

- 배낭꾸리기 문제(Knapsack Problem)

부피가 V인 배낭과 n가지의 종류의 물건들($x_1, x_2, \ldots, x_n$)이 있다. 각각의 물건들은 그 무게가 w_j , 그 부피가 v_j 이다. 어떻게 하면 배낭의 부피를 넘지 않으면서 그 무게가 가장 무겁도록 채울 수 있을까? 또 그렇게 되려면 각 물건들을 각각 몇 개씩 넣어야 할까?.

N-P Complete

이 문제를 수식으로 표현해 보자. 물건 x_k 의 개수를 d_k 라 하면

$$\text{Maximize} \sum_{k=1}^{n} d_k w_k \quad \text{subject to} \sum_{k=1}^{n} d_k v_k \leq V$$

와 같이 나타낼 수 있다.

직관적으로 생각하면 최대의 가치를 가지는 물건들로 배낭을 먼저 채우거나 단위 무게당 가치가 가장 높은 것을 먼저 채우는 것이 현명할 것 같다. 그러나 실제로 그렇지 않은 경우도 많이 있다.

다음 예를 살펴보자.

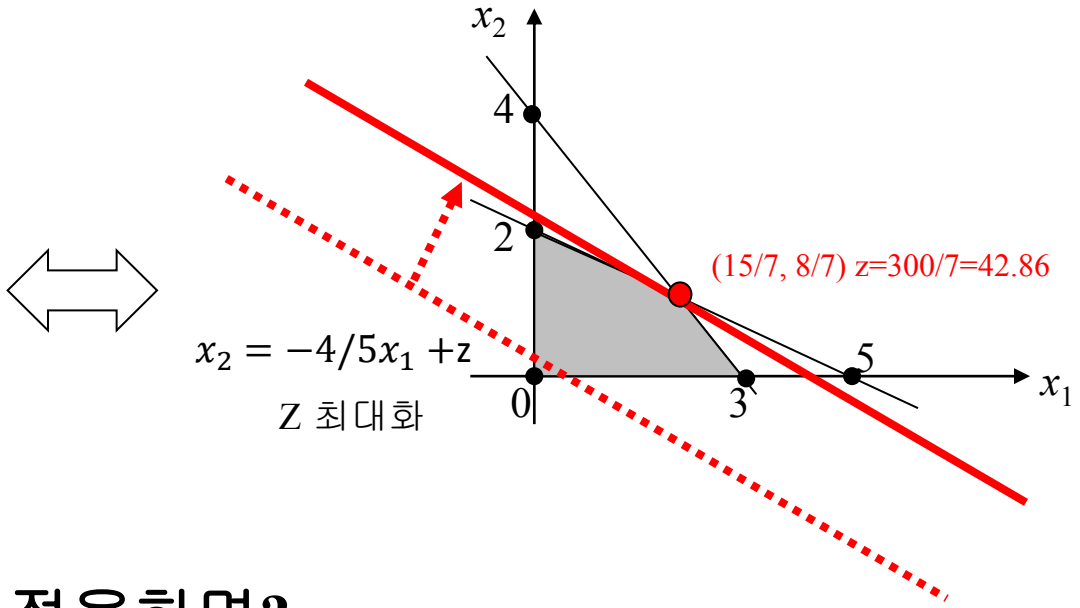
<예> 로빈 후드가 와의 보물창고에 들어가 보니 다음과 같은 보물들이 있었다.

품 목	무게(Kg)	가치
금병	6	480
작은 동상	2	158
은 컵	3	233
은전	1	2

그런데 로빈 후드의 배낭에는 13Kg 만 담을 수 있다. 먼저 금병 2개와 은전 1개를 택할 수 있다. 그렇다면 로빈 후드는 962 만큼의 가치를 담아가는 것이다. 그러나 다른 선택을 통해 로빈 후드는 1029 만큼의 가치를 담아갈 수도 있다. 어떻게 하면 될까?

Greedy Method의 함정(3)

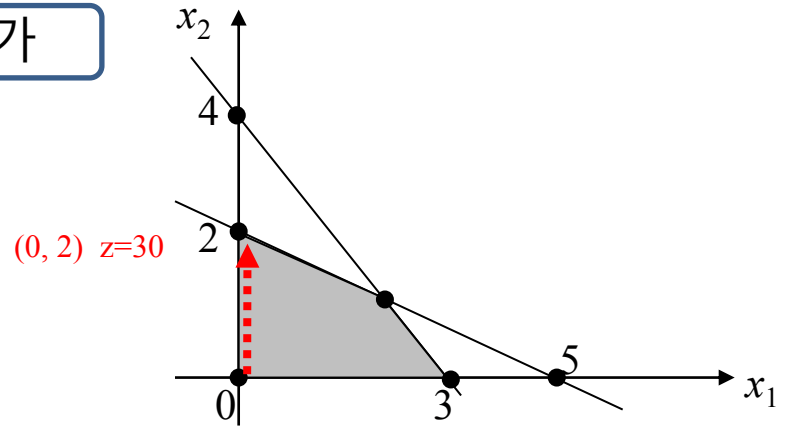
$$\begin{aligned}\text{Max}\quad & 12x_1 + 15x_2 \\ \text{s.t.}\quad & 4x_1 + 3x_2 \le 12 \\ & 2x_1 + 5x_2 \le 10 \\ & x_1, x_2 \ge 0\end{aligned}$$



그리디 알고리즘을 적용하면?

x_2를 증가

$$\begin{aligned}\text{Max}\quad & 12x_1 + 15x_2 \\ \text{s.t.}\quad & 4x_1 + 3x_2 \le 12 \\ & 2x_1 + 5x_2 \le 10 \\ & x_1, x_2 \ge 0\end{aligned}$$



Greedy Method 함정 탈출

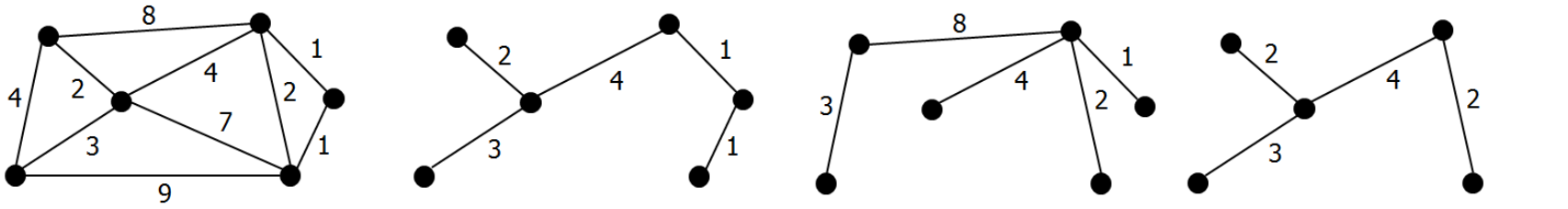
- 무작정 탐욕만 부리면 안됨
- Principle of Optimality
  - 다단계 결정과정에 대한 최적정책을 결정하는데 핵심적인 의미를 가지고 있음
  - 처음의 상태와 처음의 결정이 무엇이든 간에 목표에 도달하는 나머지 과정은 이때까지의 결정의 결과로서 나타난 상태에 관하여 최적정책을 구성하여야 함
  - "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first step" – R.E.Bellman(1957)
  - 현재의 최선의 선택이 전체의 최선이다
  - 동적프로그래밍(Dynamic Programming, DP)

- 동적프로그래밍 의 대표적인 사례
  - Minimum spanning tree 를 찾는 문제에서 Kruskal algorithm과 Prime algorithm
  - Shortest problem 에서의 Dijkstra's algorithm
  - Bellman의 동적프로그래밍

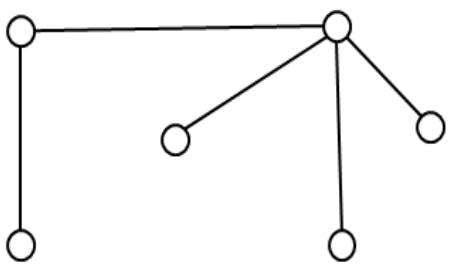
4.2 최소 신장 트리

- 최소 신장 트리 (Minimum Spanning Tree)
  - 정의
    - 주어진 가중치 그래프에서
      - 사이클이 없이
      - 모든 점들을 연결시킨 트리들 중
      - 선분들의 가중치 합이 최소인 트리
  - 예제
    - 가중치 그래프가 (a) 처럼 주어졌다면 최소 신장 트리 는?
    - (b) 최소 신장 트리. 그러나 (c),(d)는 최소 신장 트리 아님
      - (c)는 가중치의 합이 (b)보다 크고, (d)는 트리가 주어진 그래프의 모든 노드를 포함하지 않고 있다.

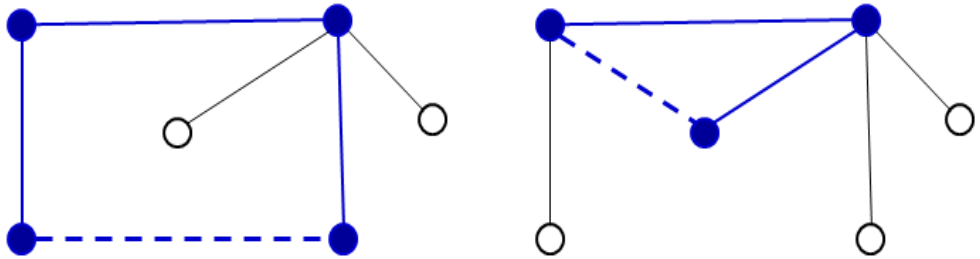
(a) (b) ←최소신장트리 (c) (d)



- 주어진 그래프의 신장 트리를 찾으려면
  - 사이클이 없도록 모든 점을 연결
  - 그래프의 점의 수가 n이면, 신장 트리에는 정확히 (n-1)개의 선분
- 트리에 선분을 하나 추가시키면, 반드시 사이클이 만들어진다.



트리



점선으로 된 선분을 추가하여 만들어진 사이클

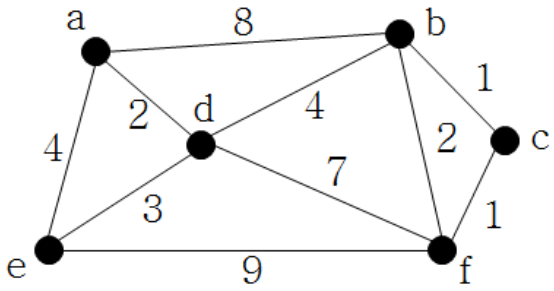
- 최소 신장 트리를 찾는 그리디 알고리즘
  - 크러스컬 (Kruskal)과 프림 (Prim) 알고리즘
- 크러스컬 알고리즘
  - 가중치가 가장 작은 선분이 사이클을 만들지 않을 때에만 '욕심내어' 그 선분을 추가함

KruskalMST(G)

입력: 가중치 그래프 G=(V,E), |V|=n , |E|=m
출력: 최소 신장 트리 T

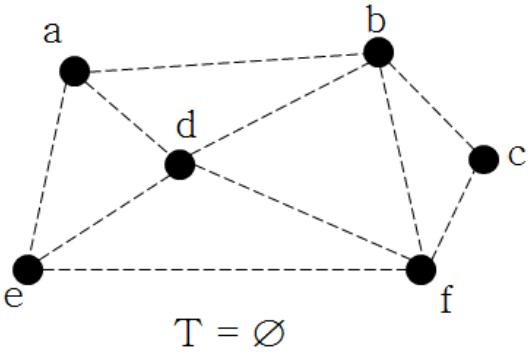
1. 가중치의 오름차순으로 선분들을 정렬. 정렬된 선분 리스트를 L
2. T=Ø // 초기화
3. while (T의 선분 수 < n-1) {
4. L에서 가장 작은 가중치를 가진 선분 e를 가져오고, e를 L에서 제거
5. if (선분 e가 T에 추가되어 사이클을 만들지 않으면)
6. e를 T에 추가
7. else
8. e를 버림.
   }
9. return 트리 T // T는 최소 신장 트리

- KruskalMST 알고리즘의예

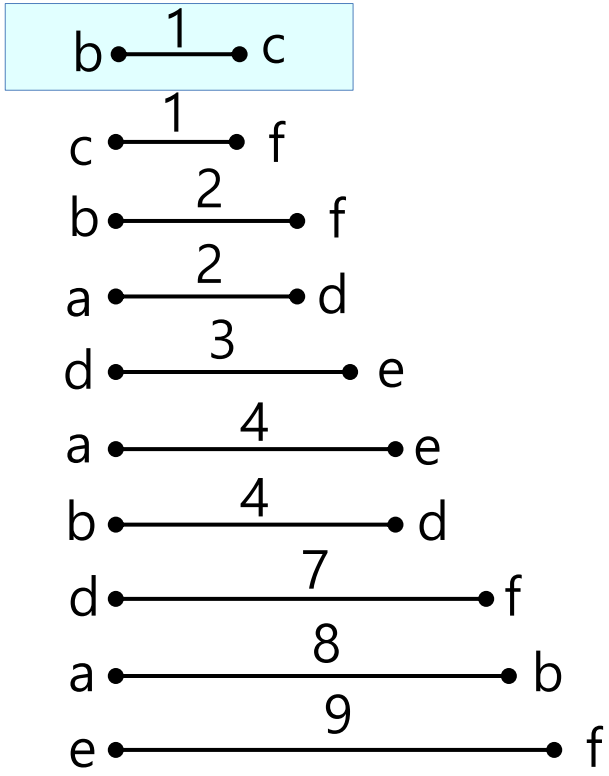


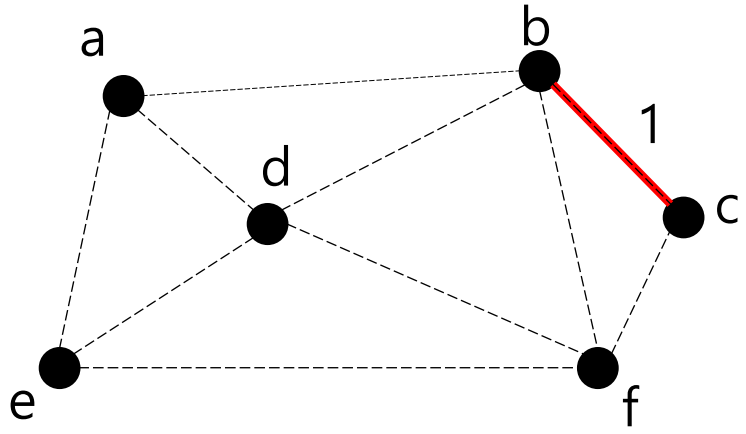
정렬된
리스트
L

b 1 c
c 1 f
b 2 f
a 2 d
d 3 e
a 4 e
b 4 d
d 7 f
a 8 b
e 9 f



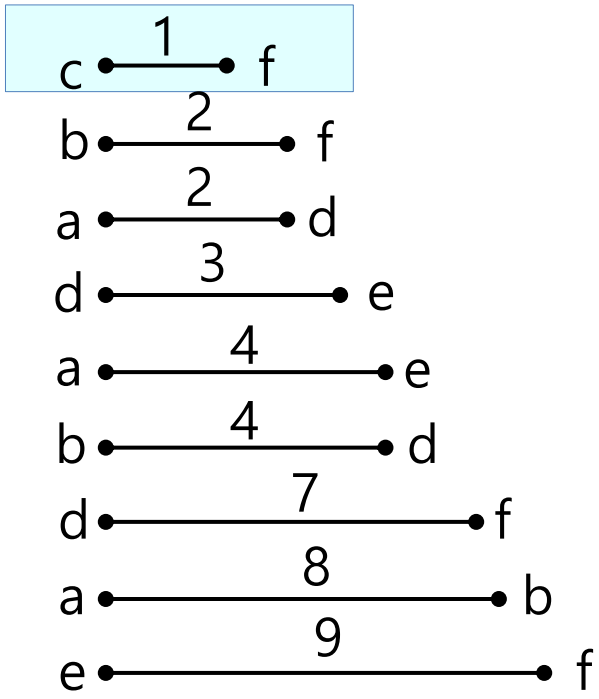
리스트
L

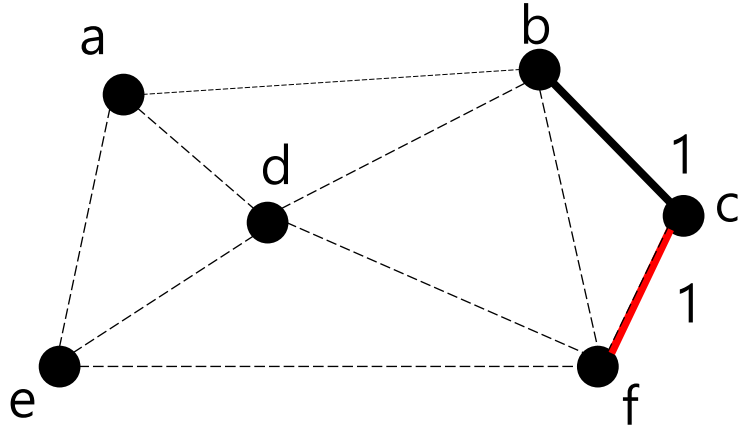




선분 (b,c) 추가

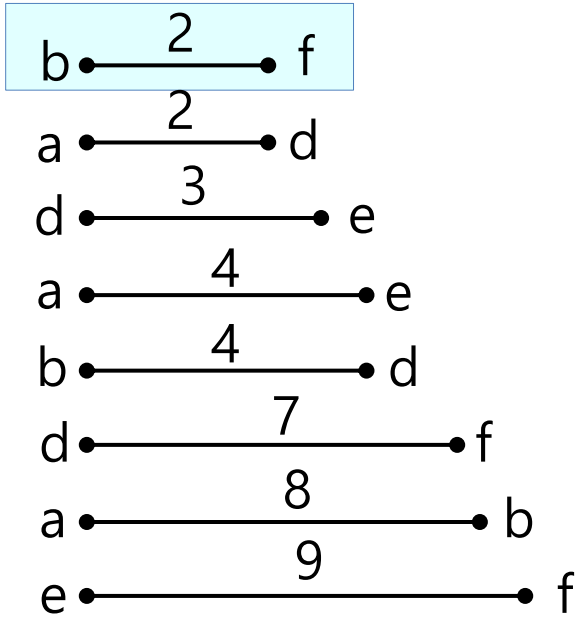
리스트
L

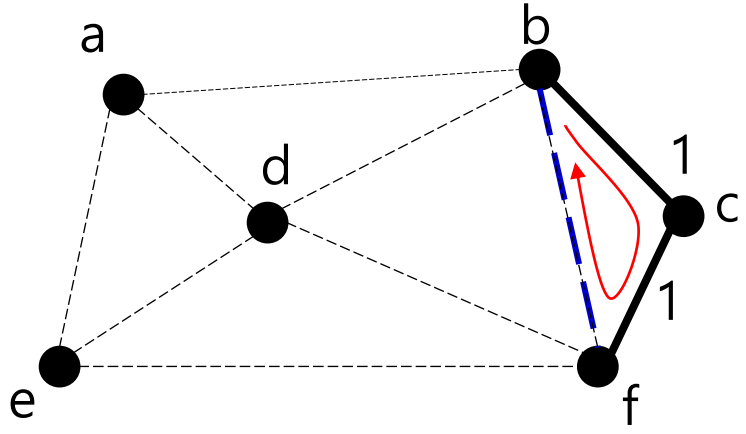




선분 (c,f) 추가

리스트
L

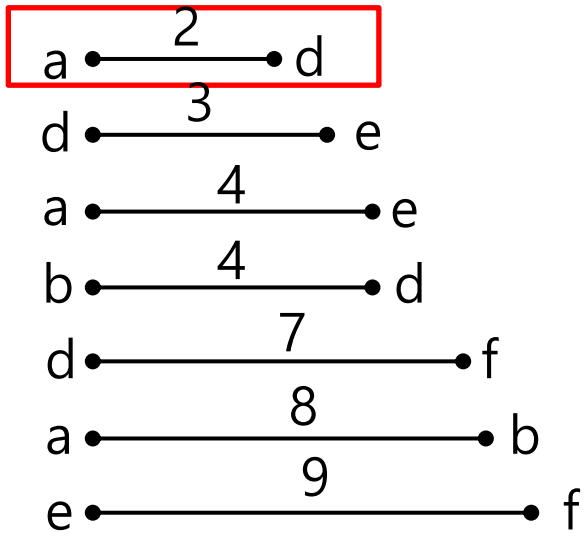


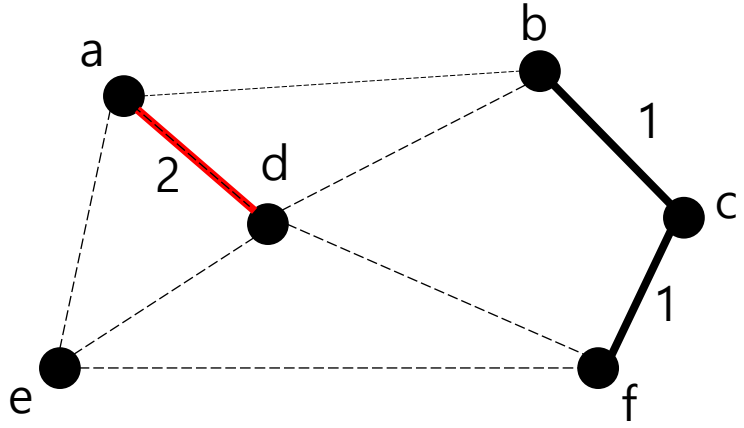


사이클 b-c-f-b

선분 (b,f) 버림

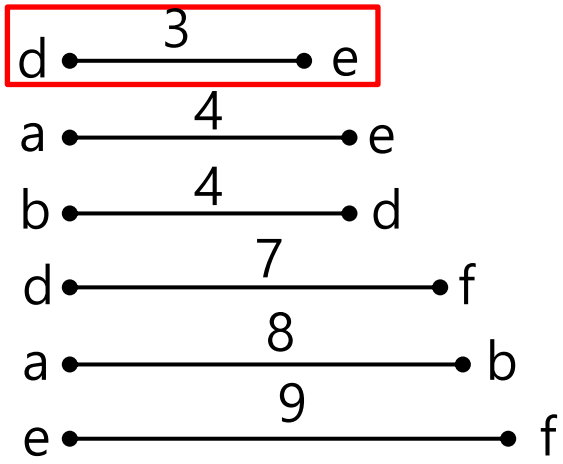
리스트
L

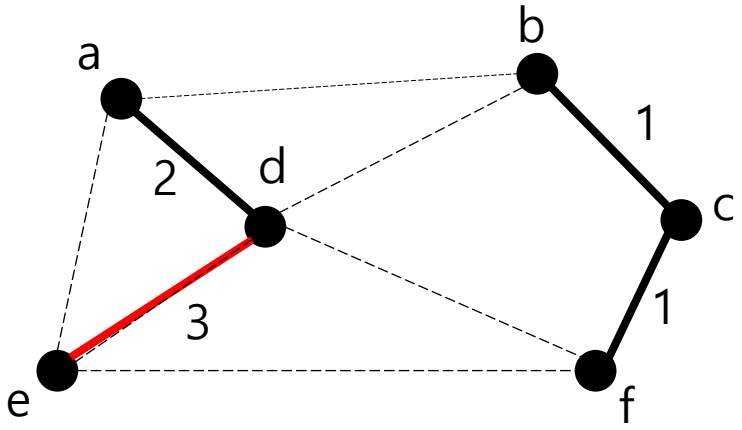




선분 (a,d) 추가

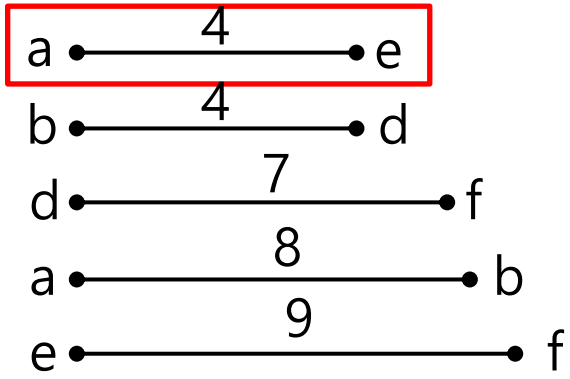
리스트
L

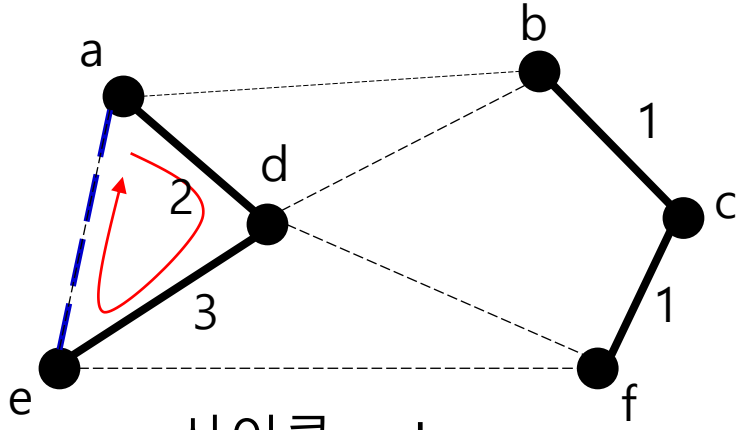




선분 (d,e) 추가

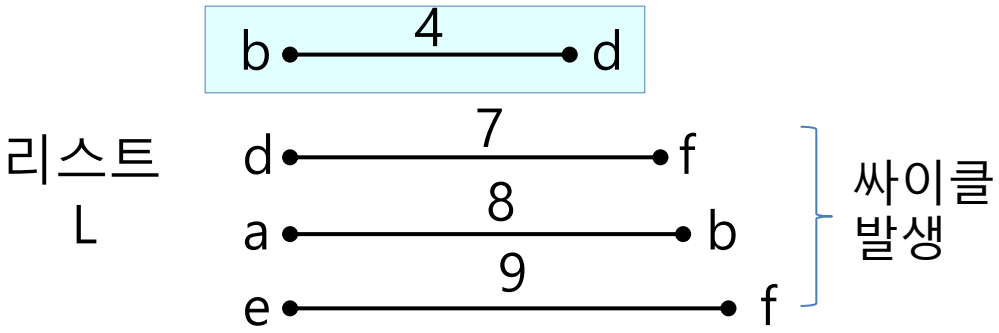
리스트
L

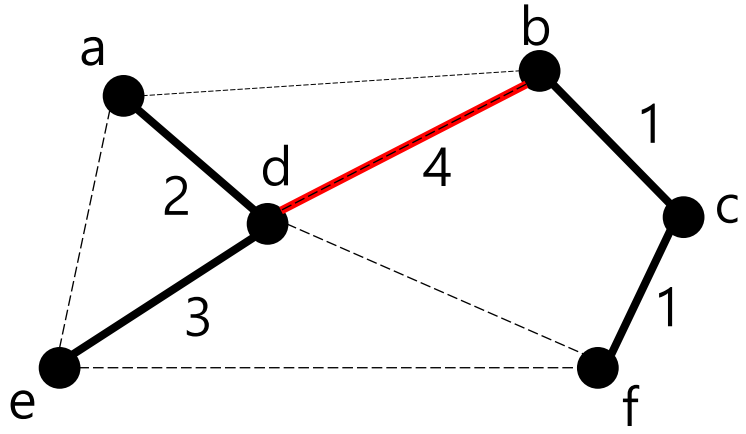




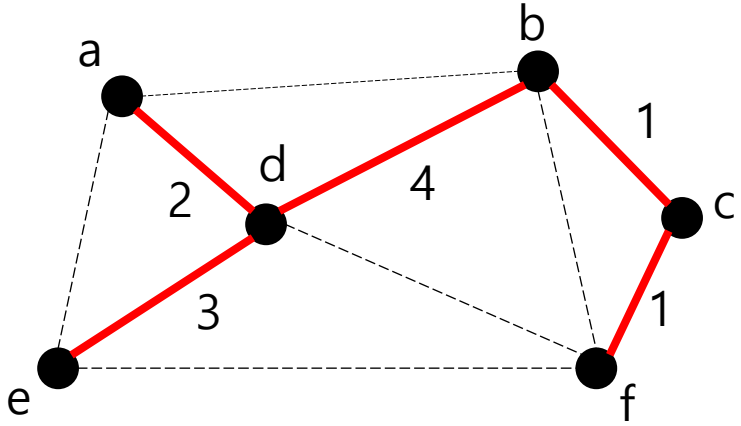
사이클 a-d-e-a

선분 (a,e) 버림





선분 (b,d) 추가



최종해

시간복잡도

Kruskal 알고리즘

```
1. 가중치의 오름차순으로 선분들을 정렬. 정렬된 선분 리스트를 L
2. T=Ø          // 초기화
3. while ( T의 선분 수 < n-1 ) {
4.    L에서 가장 작은 가중치를 가진 선분 e를 가져오고, e를 L에서 제거
5.    if (선분 e가 T에 추가되어 사이클을 만들지 않으면)
6.          e를 T에 추가
7.       else
8.             e를 버림.
   }
9. return 트리 T    // T는 최소 신장 트리
```

- Line 1: 정렬하는데 $O(m \log_2 m)$ 시간
  - 단, m은 입력 그래프에 있는 선분의 수이다.
- Line 2: T를 초기화하는 것이므로 O(1) 시간
- Line 3~8의 while-루프는 최악의 경우 m번 수행
  - 즉, 그래프의 모든 선분이 while-루프 내에서 처리되는 경우
  - L로부터 가져온 선분 e가 사이클을 만드는지를 검사하는데 $O(\log_2 m)$ 시간
  - 최악의 경우 m번 수행하므로 $O(m \log_2 m)$
- 따라서 크러스컬 알고리즘의 시간복잡도는
  - $O(m \log_2 m) + O(m \log_2 m) = O(m \log_2 m)$ 이다.

숙제
연습문제 4-5참고

그리디알고리즘의 활용 사례

- 최소신장트리
  - 프림 (Prim)의 최소 신장 트리 알고리즘
- 최단경로찾기
- 부분배낭문제
- 집합커버문제
- 작업스케쥴링
- 허프만코딩
- 동적계획법(5장)