5장

그리디 알고리즘

GREEDY ALGORITHM

지금 이 순간 가장 좋아 보이는 것을 고른다

이번 장에서 배울 내용

5.1 그리디의 기본

5.2 거스름돈 문제

5.3 활동 선택 문제

5.4 배낭 문제

5.5 그리디가 통하는 조건

5.6 크루스칼 알고리즘

보충 상세 구현

5.8 다익스트라 알고리즘

그리디: 지금 가장 좋아 보이는 것을 고른다

그리디(greedy)는 '탐욕스러운'이라는 뜻입니다.
매 단계 지금 기준으로
가장 좋아 보이는 것을 고릅니다.

후보를 다 알아도 지금 기준 하나로

앞을 못 봐서 틀리는 것이 아닙니다.
정해 둔 기준으로만 고르기 때문입니다.

고른 것은 확정, 되돌리지 않음

앞의 선택을 다시 검토하는 단계가 없습니다.

완성 해를 모두 비교하지 않음

모든 해를 만들어 비교하는 것은 완전탐색입니다.

**되돌아보지 않아서 빠르고,
그래서 틀릴 수도 있습니다.**



손님은 '지금 맛있어 보이는 것' 기준 하나로 접시를 채웠고, 내려놓지 않아 스테이크 자리가 없습니다.

지역 최적해와 전역 최적해

안개 낀 산에서 가장 높은 봉우리를 찾습니다.
가장 단순한 전략은 지금보다 높은 쪽으로
한 걸음씩 오르는 것입니다.

지역 최적해 (local optimum)

주변 어디보다 높은 곳.
작은 언덕 꼭대기일 수 있습니다.

전역 최적해 (global optimum)

산 전체에서 가장 높은 곳.
우리가 원하는 답입니다.

매 걸음의 최선이 모여도
전체의 최선이라는 보장은 없습니다.



마시멜로 하나를 지금 먹는 것도 지금만 보면 최선입니다.
기다려서 두 개를 받는 쪽이 전체로는 더 좋습니다.

그림의 '국소적 최적해(Local Maximum)'는 지역 최적해(local optimum)와 같은 말입니다.

그리디 알고리즘의 세 단계

해답이 완성될 때까지 세 단계를 되풀이합니다.

① 선택

selection
남은 후보 중 지금 기준으로
가장 좋은 것 하나를 고릅니다.

② 적정성 검사

feasibility
그 선택을 넣어도 제약을
어기지 않는지 확인합니다.

③ 해답 검사

solution
모은 것이 완전한 해답인지
확인합니다. 아니면 ①로.

greedy.pseudo

```
GREEDY(C):                                          // C: 후보 집합
    S <- 빈 집합                                     // 지금까지 확정한 선택
    while C가 비어 있지 않고 S가 해답이 아니면:        // 해답 검사
        C에서 지금 가장 좋은 후보 x를 꺼낸다          // 선택
        if S + {x}가 제약을 지키면:                   // 적정성 검사
            S <- S + {x}                            // 확정, 되돌리지 않음
    if S가 해답이면 return S else return "해 없음"
```

한 번 확정한 선택은 다시 검토하지 않습니다.

이 장의 지도: 통하는 문제와 통하지 않는 문제

모두 '지금 가장 좋은 것 고르기'이지만,
최적해를 보장하는 문제와 그렇지 않은 문제가 있습니다.

절	문제	매 단계의 선택	최적해
5.2	거스름돈 (500·100·50·10원)	가장 큰 동전	**보장**
5.2	거스름돈 (1·3·4원)	가장 큰 동전	**보장 못 함**
5.3	활동 선택 (회의 배정)	가장 먼저 끝나는 회의	**보장**
5.4	분할 배낭 (쪼갤 수 있음)	무게당 가치가 가장 높은 보물	**보장**
5.4	0/1 배낭 (쪼갤 수 없음)	무게당 가치가 가장 높은 보물	**보장 못 함**
5.6	최소 신장 트리 (크루스칼)	사이클을 만들지 않는 가장 싼 간선	**보장**
5.8	최단 경로 (다익스트라)	시작점에서 가장 가까운 미확정 정점	**보장 (간선 비용 ≥ 0)**

왜 어떤 문제에서는 통하고 어떤 문제에서는 통하지 않는지는 5.5절에서 두 조건으로 정리합니다.

손님에게 1,260원을 거슬러 줍니다

동전은 500원, 100원, 50원, 10원
네 종류가 충분히 있습니다.

목표: 동전 개수를 최소로

그리디의 기준

남은 돈 이하의 가장 큰 동전부터 고릅니다.

한 번에 몫만큼

그 동전으로 낼 수 있는 만큼 한꺼번에 냅니다.

나머지로 반복

남은 돈으로 다음 큰 동전을 봅니다.



만화의 점원도 가장 큰 500원부터 집습니다.

몫과 나머지로 한 번에 셉니다

500원을 하나씩 빼지 않습니다.
나눗셈 한 번으로 쓸 개수와 남은 돈을 함께 구합니다.

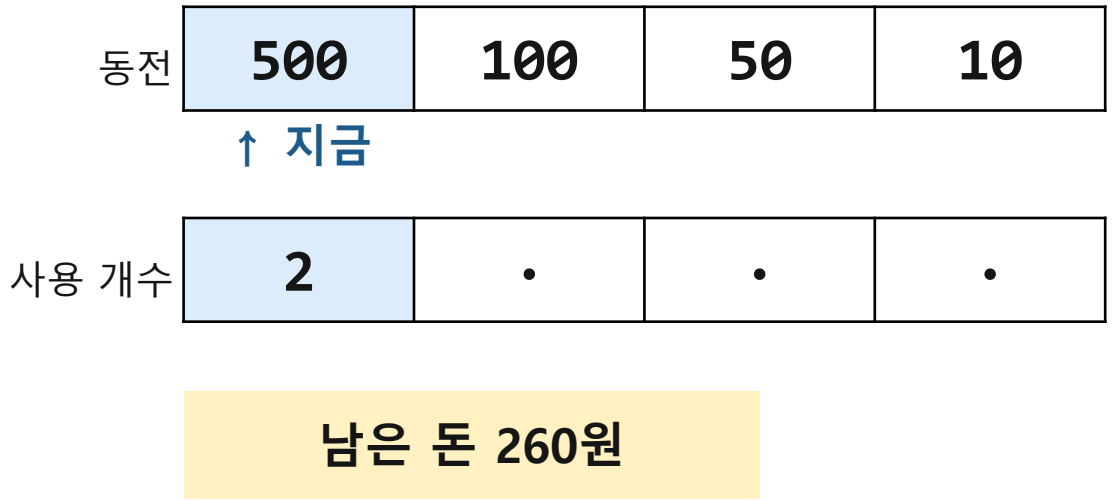
식 (Python)	뜻	1,260원에서
`1260 // 500`	몫: 500원을 몇 개 쓰는가	**2**
`1260 % 500`	나머지: 다음 동전으로 넘기는 돈	**260**

1260 ÷ 500 = 2.52처럼 소수로 보지 않습니다. 500원 2개를 쓰고 260원이 남습니다.

그리디의 세 단계	거스름돈에서
① 선택	남은 돈 이하의 가장 큰 동전
② 적정성 검사	몫만큼 써도 남은 돈을 넘지 않는다
③ 해답 검사	남은 돈이 0원이면 끝

① 500원 · 1,260원에서 2개

1,260원 이하의 가장 큰 동전은 500원입니다.
1260 // 500 = 2, 1260 % 500 = 260



단계	이번 판단
① 선택	500원
② 적정성	2개 = 1,000원 ≤ 1,260원
③ 해답 검사	260원 남음 → 계속

지금까지 확정한 동전

500원 2개

파랑: 지금 보는 동전　　회색: 이미 처리한 동전

계산 결과: 남은 돈 0원, 모두 6개

마지막 10원입니다. 10 // 10 = 1, 10 % 10 = 0
남은 돈이 0원이 되어 해답이 완성됩니다.

동전	500	100	50	10
				↑ 지금
사용 개수	2	2	1	1

남은 돈 0원

단계	이번 판단
① 선택	10원
② 적정성	1개 = 10원 ≤ 10원
③ 해답 검사	0원 → 완성

지금까지 확정한 동전

2 + 2 + 1 + 1 = 6개

이 6개가 최소인 이유는
뒤의 배수 조건에서 봅니다.

파랑: 지금 보는 동전 회색: 이미 처리한 동전

복잡도: 금액이 아니라 동전 종류 수

K = 동전 종류의 수. 지금은 500, 100, 50, 10원이라 K = 4입니다.

거슬러 줄 돈	몫과 나머지: 반복 횟수	하나씩 빼기: 빼는 횟수
1,260원	4번	6번
1,260,000원	**4번**	**2,520번**

반복 한 번: 몫과 나머지 한 번씩 → O(1)

금액이 커도 나눗셈 한 번이면 개수가 나옵니다.

전체: K × O(1) = O(K)

목록이 이미 큰 단위부터 적혀 있을 때입니다. 정렬부터 해야 하면 O(K log K)가 더해집니다.

하나씩 빼는 방법은 반복 횟수가 금액에 비례해서 늘어납니다.

동전 종류가 고정이면 K는 상수이므로 O(1)이라고도 씁니다.

왜 큰 동전부터가 최적인가: 배수 조건

한국 동전은 큰 단위가 바로 아래 단위의 배수입니다.

큰 동전 1개	같은 금액의 작은 동전	개수
500원	100원 5개	1개 ↔ 5개
100원	50원 2개	1개 ↔ 2개
50원	10원 5개	1개 ↔ 5개

작은 동전이 쌓이면 큰 동전 하나로 바꿀 수 있습니다

100원 5개를 500원 1개로 바꾸면 금액은 같고 개수는 5개에서 1개로 줄어듭니다.

그래서 큰 동전을 덜 쓰면 개수만 늘어납니다

큰 동전을 낼 수 있는 만큼 쓰는 그리디의 선택이 손해가 되지 않습니다.

배수 조건이면 그리디가 최적입니다(충분조건).

배수 조건이 깨지면 이 보장이 사라집니다.
늘 틀린다는 뜻이 아니라 틀리는 입력이 생길 수 있다는 뜻입니다.

연습: 동전 1, 3, 4원으로 6원 · 시작

주장: "남은 돈 이하의 가장 큰 동전을 하나씩 고르면 어떤 화폐 체계에서도 개수가 최소다."

동전	4	3	1
사용 개수	•	•	•

남은 돈 6원

문제 조건

동전: 1원, 3원, 4원
(각각 무제한)

목표: 정확히 6원을
가장 적은 개수로

그리디: 남은 돈 이하의
가장 큰 동전 하나를 고르고
0원이 될 때까지 반복

이 주장을 반박해 봅시다.

파랑: 지금 고른 동전　　회색: 남은 돈보다 커서 고를 수 없는 동전

① 4원 · 6원에서 4원 하나

6원 이하의 가장 큰 동전은 4원입니다.
4원 하나를 확정합니다. 남은 돈 6 − 4 = 2원.



단계	이번 판단
① 선택	4원
② 적정성	4원 ≤ 6원
③ 해답 검사	2원 남음 → 계속

지금까지 고른 동전

4원 (1개)

파랑: 지금 고른 동전　　회색: 남은 돈보다 커서 고를 수 없는 동전

최적: 3 + 3 = 2개 · 반례 완성

4원을 쓰지 않으면 3원 두 개로 6원이 됩니다.
2개 < 3개, 그리디의 답은 최소가 아닙니다.



그리디: 4 + 1 + 1 = 6원 → 3개

최적: 3 + 3 = 6원 → 2개

**이 입력 하나로
주장이 깨집니다.**

반박에 쓸 것:
그리디의 결과, 더 좋은 해,
두 개수의 비교

파랑: 그리디가 쓴 동전 초록: 최적해가 쓴 동전

'항상 최적'은 반례 하나로 반박합니다

'항상 최적이다'라는 주장은 맞지 않는 입력이 하나만 있어도 거짓입니다.

① 문제의 조건을 지키는 입력

동전 1, 3, 4원과 금액 6원은 문제가 허용하는 입력입니다.
조건을 어긴 입력에서 틀리는 것은 반박이 아닙니다.

② 그리디의 결과보다 더 좋은 해

그리디 3개보다 적은 2개를 보였습니다. 그리디와 개수가 같은 다른 해는 반례가 아닙니다.

③ 하나면 충분

반례를 여러 개 모을 필요가 없습니다. 실행이 느린 입력은 최적성의 반례가 아닙니다.

반대로, 예제 몇 개에서 맞았다고 '항상 최적'이 증명되지는 않습니다.

맞다는 것을 보이는 방법은 5.3절 끝과 5.5절에서 다룹니다.

연습: coin_count · 1원까지 다섯 종류

coin_count(amount): amount원을 거슬러 줄
최소 동전 개수를 돌려줍니다. 큰 동전부터
몫만큼 쓰고 나머지를 다음 동전으로 넘깁니다.

동전	500	100	50	10	1
몫 (개수)	2	2	1	1	3
남은 돈	263	63	13	3	0

coin_count(1263) = 2 + 2 + 1 + 1 + 3 = 9

coin_count.pseudo

```
COIN_COUNT(amount):
  count <- 0
  for coin in [500, 100, 50, 10, 1]:
    count <- count + 몫(amount, coin)
    amount <- 나머지(amount, coin)
  return count
```

몫: Python //, C / 나머지: %

금액과 관계없이 5번 → O(1)

K = 동전 종류 수로 쓰면 O(K)
하나씩 빼면 O(amount)
amount = 0이면 0개

함수 Python 구현

정수배 화폐 거스름돈

```python
def coin_count(amount):
    count = 0
    for coin in [500, 100, 50, 10, 1]:
        count += amount // coin
        amount %= coin
    return count
```

고정된 5종류이므로 시간 O(1). 일반 m종류에서는 시간 O(m).

함수 C 구현

정수배 화폐 거스름돈

```c
int coin_count(int amount) {
    int coins[]={500,100,50,10,1}, count=0;
    for (int i=0; i<5; ++i) {
        count+=amount/coins[i];
        amount%=coins[i];
    }
    return count;
}
```

고정된 5종류이므로 시간 O(1). 일반 m종류에서는 시간 O(m).

회의실 하나, 회의는 최대한 많이

회의실은 하나, 요청은 여러 건입니다.
회의마다 시작 시각과 종료 시각이 있습니다.

회의 하나 = 활동(activity)

그래서 활동 선택 문제라고 부릅니다.

한 번에 한 회의만

시간이 겹치는 두 회의는 함께 넣을 수 없고,
회의를 도중에 끊을 수도 없습니다.

목표: 회의의 개수를 최대로

사용 시간의 합이 아니라 넣은 회의의 수입니다.

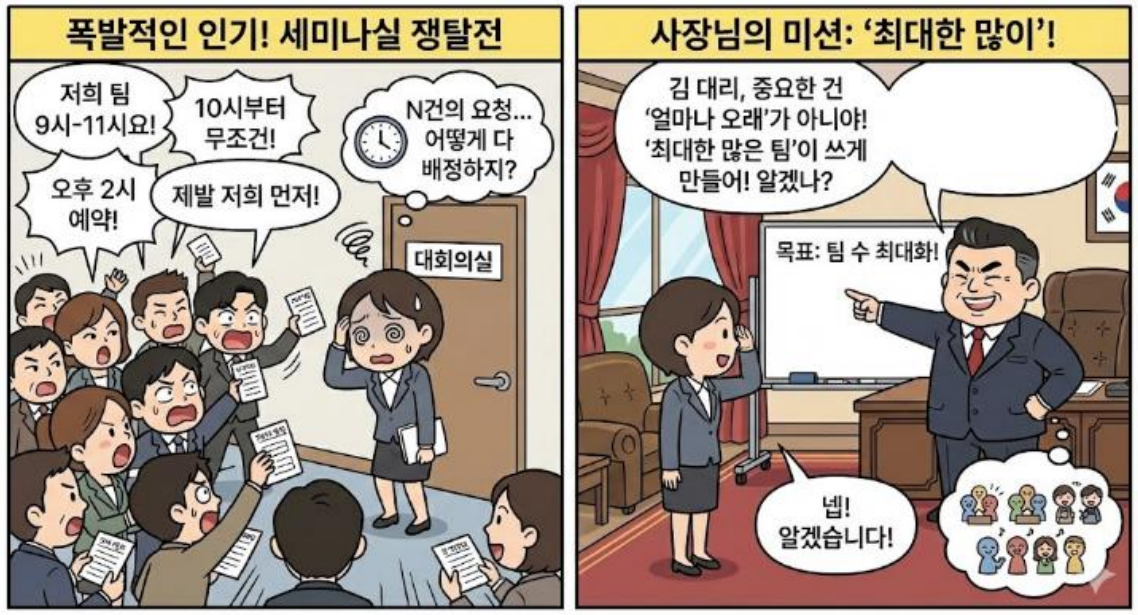
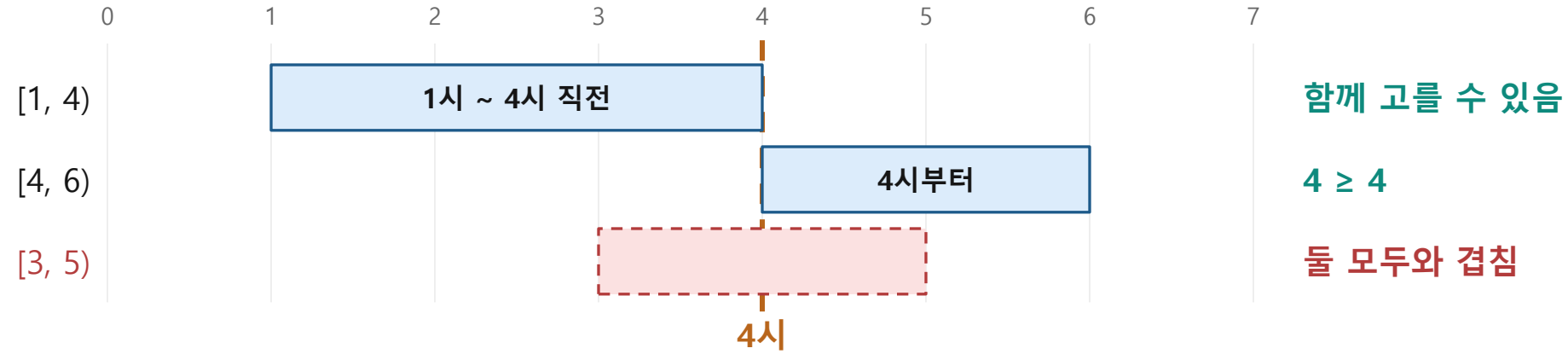


그림 속 '세미나실'은 이 강의의 회의실입니다. 목표는 '오래'가 아니라 '많이'입니다.

[시작, 종료) 표기: 끝나는 시각에 바로 시작

[1, 4)는 1시에 시작해 4시에 끝나는 회의입니다.
[는 1시를 포함하고,)는 4시를 포함하지 않습니다.



규칙: 다음 회의의 시작 ≥ 앞 회의의 종료이면 겹치지 않습니다.

[1, 4)와 [4, 6)은 함께 고를 수 있습니다. 4시에 하나가 끝나고 다른 하나가 시작합니다.
[3, 5)는 4시 전에 시작해 4시 뒤에 끝나므로 [1, 4), [4, 6) 둘 모두와 겹칩니다.

실패 ① 일찍 시작하는 회의부터? · 1개

시작이 빠른 순으로 보며 겹치지 않으면 고릅니다.
[1, 10)을 먼저 고르자 나머지 둘이 모두 겹칩니다.



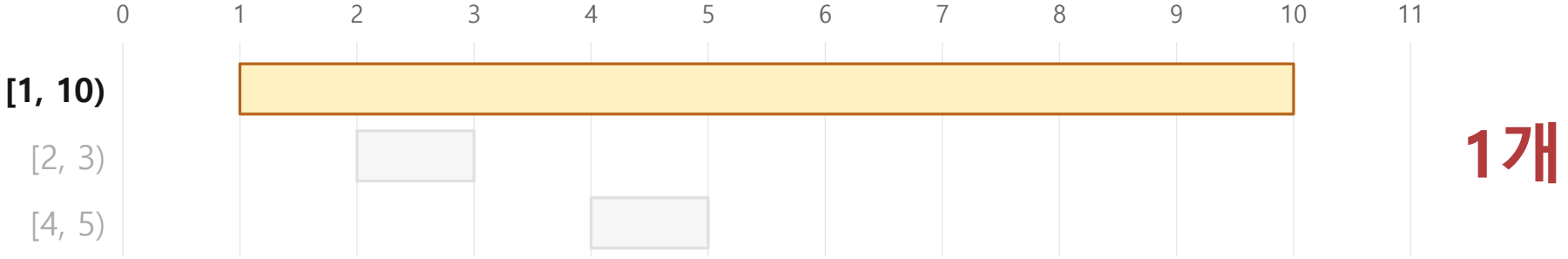
시작 순 그리디: 1개

노랑: 고른 회의 빨강 점선: 겹쳐서 버린 회의 주황 점선: 회의실이 비는 시각

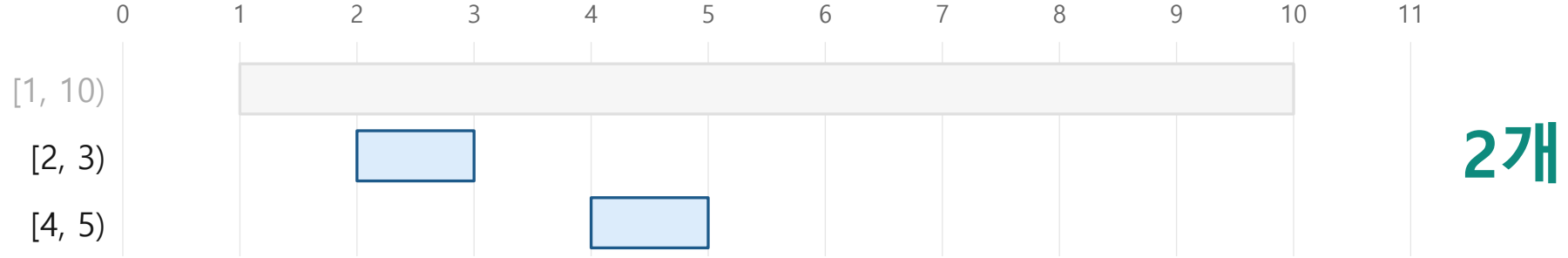
반례 완성: 시작 순 1개 < 최적 2개

[1, 10)을 버리면 [2, 3)과 [4, 5)를 함께 넣을 수 있습니다.
그리디 1개 < 최적 2개, 이 기준은 최적을 보장하지 않습니다.

시작 순 그리디: [1, 10) 하나



최적: [2, 3)과 [4, 5) 두 개



노랑: 그리디가 고른 회의 파랑: 최적해의 회의

성공 전략: 가장 먼저 끝나는 회의부터

회의실을 가장 빨리 비워 주는 회의부터 고릅니다.
last_end는 마지막으로 고른 회의의 종료 시각입니다.

① 정렬

회의를 종료 시각이
빠른 순으로 늘어놓습니다.

② 고르기

시작 ≥ last_end이면 고르고
last_end를 그 회의의
종료 시각으로 옮깁니다.

③ 반복

끝까지 봅니다.
조건을 처음 만족하는 회의가
곧 겹치지 않는 회의 중
가장 먼저 끝나는 회의입니다.

기준별 비교	시작 순	짧은 순	종료 순	최적
[1, 10), [2, 3), [4, 5)	**1개**	2개	**2개**	2개
[1, 5), [4, 6), [5, 9)	2개	**1개**	**2개**	2개

두 예에서 맞았다고 해서 증명된 것은 아닙니다. 왜 맞는지는 이 절 끝에서 봅니다.
'안전한 선택'(그렇게 골라도 최적해를 놓치지 않는 선택)의 정확한 뜻은 5.5절에서 정의합니다.

예제: 회의 여섯 건을 종료 시각 순으로

회의 여섯 건을 종료 시각이 빠른 순으로 정렬합니다.
칸 아래의 작은 글자는 입력 순서로 붙인 회의 번호입니다.

입력 순서	[1, 3)	[2, 5)	[3, 9)	[6, 8)	[5, 7)	[8, 11)
	1번	2번	3번	4번	5번	6번

종료 순	[1, 3)	[2, 5)	[5, 7)	[6, 8)	[3, 9)	[8, 11)
	1번	2번	5번	4번	3번	6번

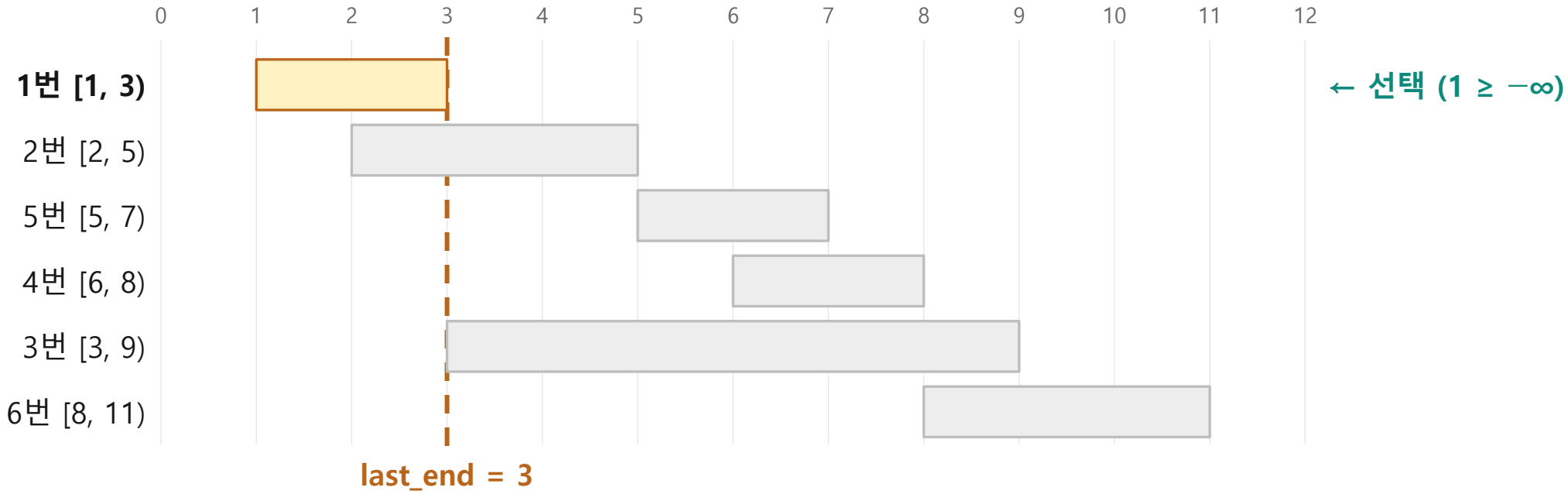
종료 시각: 3 < 5 < 7 < 8 < 9 < 11

last_end = −∞ (아직 고른 회의 없음)

노랑 칸: 입력 순서와 자리가 바뀐 회의

① 1번 [1, 3) · 선택

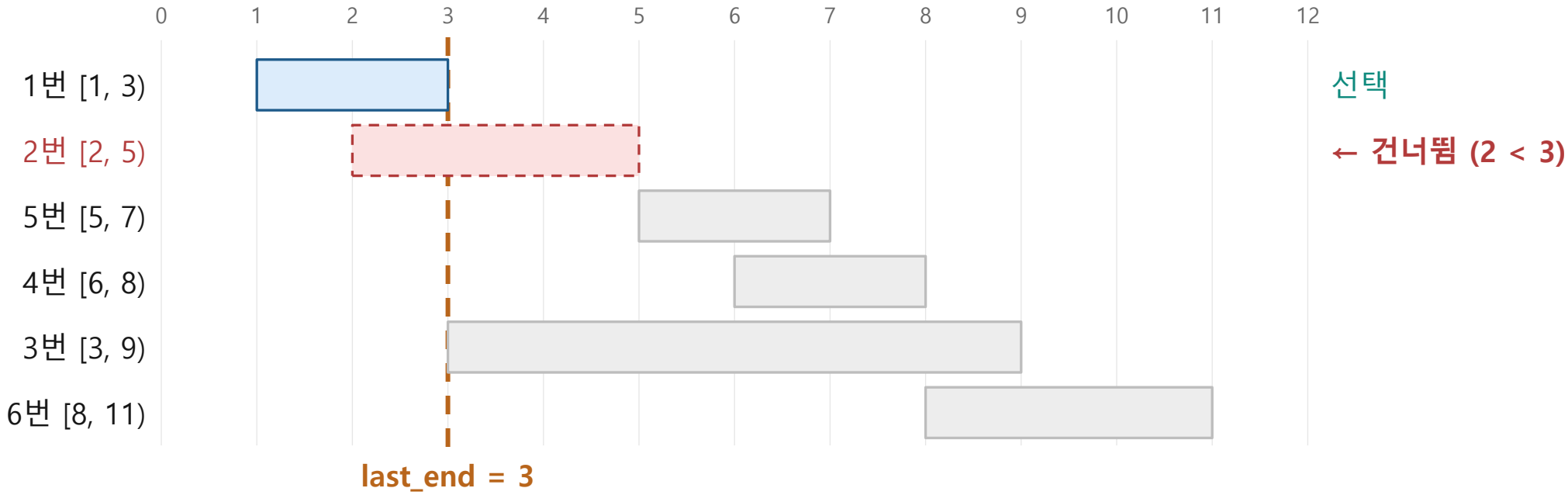
last_end = −∞이므로 1 ≥ −∞가 참입니다. 겹칠 회의가 없습니다.
1번을 고르고 last_end를 3으로 옮깁니다.



파랑: 고른 회의 노랑: 지금 고른 회의 빨강 점선: 건너뜀 회색: 아직 보지 않은 회의 주황 점선: last_end

② 2번 [2, 5) · 건너뜀

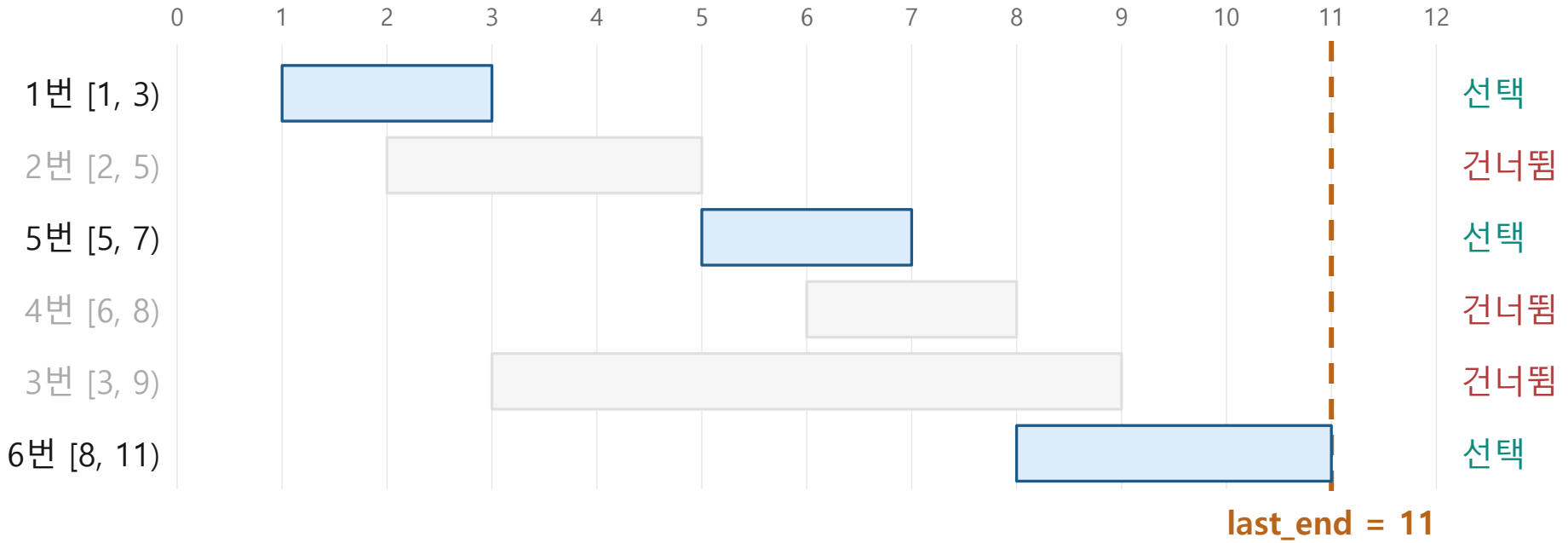
2번은 2시에 시작하는데, 직전에 고른 1번은 3시에 끝납니다.
2 ≥ 3을 만족하지 못해 건너뜁니다. last_end는 3 그대로입니다.



파랑: 고른 회의 노랑: 지금 고른 회의 빨강 점선: 건너뜀 회색: 아직 보지 않은 회의 주황 점선: last_end

완료: 1번, 5번, 6번 · 3개

고른 회의는 1번 [1, 3), 5번 [5, 7), 6번 [8, 11), 모두 3개입니다.
앞의 판정을 되돌린 적은 한 번도 없습니다.



이 3개가 최대인지는 뒤의 '왜 옳은가'에서 확인합니다.

파랑: 고른 회의　　회색: 건너뛴 회의　　주황 점선: last_end

복잡도: 정렬 O(n log n) + 한 번 훑기 O(n)

n = 회의의 수

단계	하는 일	시간
정렬	종료 시각 순으로 정렬 (sort, qsort)	O(n log n)
훑기	앞에서부터 한 번, 회의마다 비교 한 번	O(n)
공간	고른 회의 목록	O(n)

O(n log n) + O(n) = O(n log n)

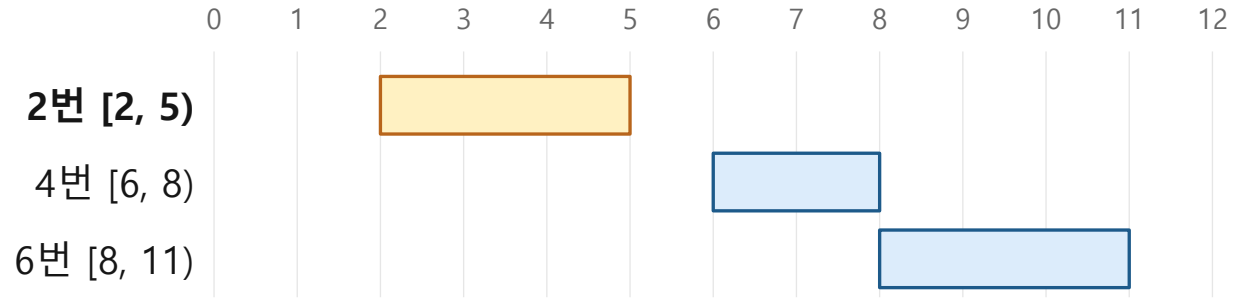
Python의 sort는 최악에도 O(n log n), C의 qsort는 보통 O(n log n)
전체 비용은 정렬이 정합니다.

입력이 이미 종료 시각 순이면 O(n)
정렬을 빼고 한 번 훑기만 하면 됩니다.

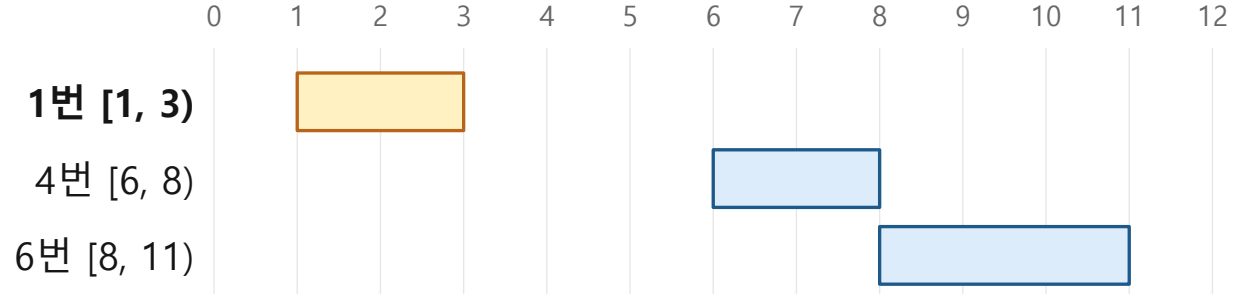
왜 옳은가: 첫 회의를 바꿔 넣어도 최적

최적해 O를 하나 가져옵니다.
예: 2번, 4번, 6번(3개). O의 첫 회의 2번을
그리디의 첫 회의 1번으로 바꿔 봅니다.

최적해 O의 예: 2번, 4번, 6번 (3개)



O의 2번을 1번으로 바꾼 O′: 3개 그대로



노랑: 바꾸는 회의 파랑: 그대로 두는 회의

① g = 그리디의 첫 회의 (1번)
= 전체에서 가장 먼저 끝나는 회의
x = O에서 가장 먼저 끝나는 회의 (2번)
→ g의 종료 ≤ x의 종료 (3 ≤ 5)

② O의 나머지는 x가 끝난 뒤 시작
→ g와도 겹치지 않습니다.

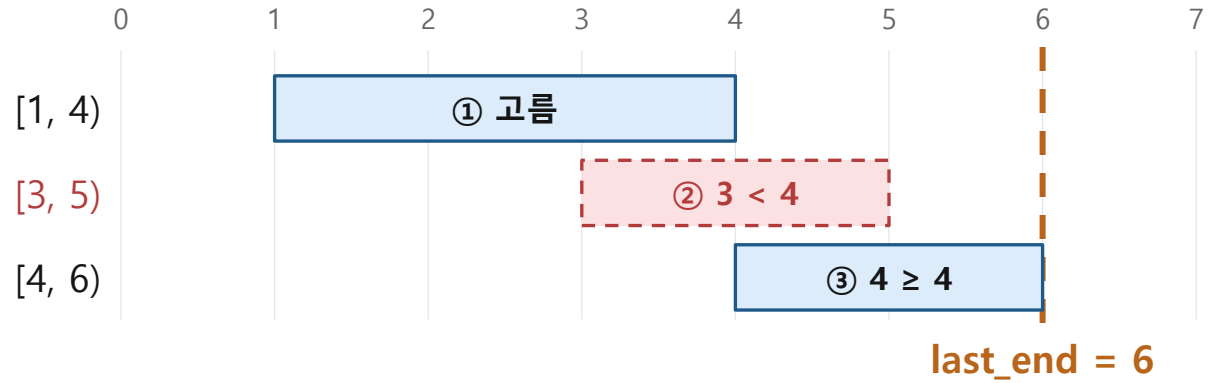
③ 개수가 같으므로 O′도 최적해
→ 남은 회의에도 되풀이합니다.

첫 회의를 포함하는 최적해가 적어도 하나 있습니다.

이 방법이 교환 논증입니다:
최적해의 일부를 탐욕 선택으로
바꿔도 여전히 최적해임을 보입니다.

연습: activity_count · [1, 4), [3, 5), [4, 6)

종료 시각 순으로 정렬되어 들어온 활동의 최대 개수를 셉니다.
시작 ≥ last_end이면 고르고 count를 1 늘립니다.



activity_count = 2

[1, 4)와 [4, 6)

차례	활동	시작 ≥ last_end ?	count	last_end
처음			0	−∞
1	[1, 4)	1 ≥ −∞ → 고름	1	4
2	[3, 5)	3 ≥ 4 아님 → 건너뜀	1	4
3	[4, 6)	4 ≥ 4 → 고름	**2**	6

activity_count: 빈 목록은 0, 시간 O(n)

activity_count.pseudo

```
ACTIVITY_COUNT(activities):     // 종료 시각 순으로 정렬되어 있다
    count <- 0
    last_end <- -∞
    for (start, end) in activities:
        if start >= last_end:   // 끝나는 시각에 바로 시작해도 된다
            count <- count + 1
            last_end <- end
    return count                // 빈 목록이면 0
```

빈 목록. for 문이 한 번도 돌지 않아 count = 0을 그대로 돌려줍니다.
목록[0]을 먼저 넣는 방식은 빈 목록에서 오류가 납니다.

시각이 0 이상으로 주어지면 last_end를 0으로 시작해도 됩니다. 첫 활동이 그대로 통과합니다.

시간 복잡도. 정렬된 입력을 한 번 훑으므로 O(n), 직접 정렬부터 하면 O(n log n)입니다.

함수 Python 구현

활동 선택

```python
def activity_count(activities):
    last = 0
    count = 0
    for start, end in activities:
        if start >= last:
            count += 1
            last = end
    return count
```

정렬된 입력에서 시간 O(n). 정렬까지 하면 O(n log n).

함수 C 구현

활동 선택

```c
int activity_count(const int start[], const int end[], int n) {
    int last=0, count=0;
    for (int i=0; i<n; ++i) {
        if (start[i]>=last) {
            ++count;
            last=end[i];
        }
    }
    return count;
}
```

정렬된 입력에서 시간 O(n). 정렬까지 하면 O(n log n).

배낭 문제: 용량 안에서 가치를 최대로

배낭에 넣을 물건 A·B·C가 각각 1개씩 있습니다. 총무게가 용량을 넘지 않게 골라,
담은 물건의 가치 합을 가장 크게 만드는 문제입니다.

물건	무게	가치
A	20kg	14,000 G
B	15kg	9,000 G
C	15kg	9,000 G

배낭 용량은 **30kg**입니다. G(골드)는 가치의 단위입니다.

분할 배낭: 물건의 일부만 덜어 담을 수 있음

예: 15kg 물건에서 10kg만 담으면, 가치도 전체의 10/15만 얻습니다.

0/1 배낭: 안 넣기(0) 또는 통째로 넣기(1)

예: 30kg 조각상은 반으로 자르거나 20kg만 담을 수 없습니다.

먼저 분할 배낭(30kg)을 계산하고, 뒤에서 0/1 배낭(50kg)과 비교합니다.

① 무게당 가치(가성비) 순으로 정렬

1kg에 얼마인지, 즉 무게당 가치를 구해
높은 순서로 세웁니다. 코드에서는 ratio라고 부릅니다.

보물	무게	가치	무게당 가치	담은 무게	얻은 가치
A	20kg	14,000 G	**700 G/kg**	·	·
B	15kg	9,000 G	**600 G/kg**	·	·
C	15kg	9,000 G	**600 G/kg**	·	·

배낭 (용량 30kg)

빈 공간 30kg

무게당 가치 = 가치 ÷ 무게

A: 14,000 ÷ 20 = 700

B: 9,000 ÷ 15 = 600

C: 9,000 ÷ 15 = 600

순서: A → B → C

B와 C는 같으므로
입력 순서대로 B를 먼저 봅니다.

노랑: 정렬 기준인 무게당 가치　　회색 막대: 비어 있는 배낭

A를 모두 담고, B는 남은 10kg만

B는 15kg인데 남은 용량은 10kg입니다.
통째로는 들어가지 않으니 10kg만 덜어 담습니다.

보물	무게	가치	무게당 가치	담은 무게	얻은 가치
A	20kg	14,000 G	700 G/kg	**20kg**	**14,000 G**
B	15kg	9,000 G	600 G/kg	**10kg (2/3)**	**6,000 G**
C	15kg	9,000 G	600 G/kg	·	·

배낭 (용량 30kg)

A 20kg	B 10kg

적정성 검사: 15kg > 남은 10kg
→ 10kg만 덜어 담습니다.

덜어 담는 비율

10 ÷ 15 = 2/3

얻는 가치

9,000 × 2/3 = 6,000 G

= 10kg × 600 G/kg

파랑: 통째로 담은 보물　초록: 일부만 덜어 담은 보물

의사코드: 무게당 가치 순으로 채우기

fractional_knapsack.pseudo

```
FRACTIONAL_KNAPSACK(items, capacity):
    items를 무게당 가치가 큰 순서로 정렬
    total <- 0
    for item in items:
        take <- min(capacity, item.weight)
        total <- total + take * item.ratio
        capacity <- capacity - take
        if capacity == 0: break
    return total
```

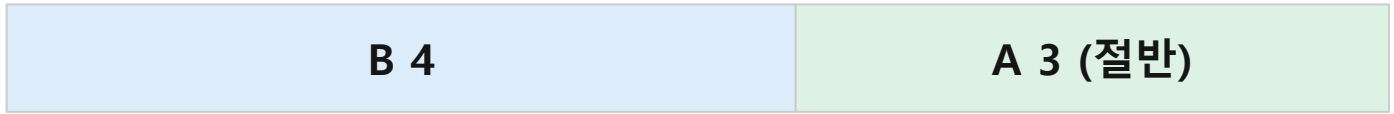
단계	의사코드에서
① 선택	정렬된 순서대로, 남은 보물 중 무게당 가치가 가장 높은 것을 봅니다.
② 적정성 검사	남은 용량과 물건 무게 중 작은 값(take)만큼 담습니다.
③ 해답 검사	남은 용량이 0이면 break. 아니면 다음 물건을 봅니다.

연습: 용량 7 · A(6, 18) · B(4, 20)

쪼갤 수 있는 배낭, 보물은 (무게, 가치)입니다.
무게당 가치 A 18 ÷ 6 = 3, B 20 ÷ 4 = 5 → B부터 봅니다.

보물	무게	가치	무게당 가치	담은 무게	얻은 가치
B	4	20	**5**	**4 (전부)**	**20**
A	6	18	**3**	**3 (3/6)**	**9**

배낭 (용량 7)



① B: 4 ≤ 7 → 전부, 가치 20

② 남은 용량 7 − 4 = 3

③ A: 6 > 3 → 3만 덜어 담기

3 ÷ 6 = 1/2 → 18 × 1/2 = 9

합계 20 + 9 = 29

파랑: 통째로 담은 보물 초록: 일부만 덜어 담은 보물 노랑: 무게당 가치

연습: 용량 5 · A(2, 8) · B(4, 16)

무게당 가치는 A 8 ÷ 2 = 4, B 16 ÷ 4 = 4로 같습니다.
같으면 앞에 있는 A부터 봅니다.

보물	무게	가치	무게당 가치	담은 무게	얻은 가치
A	2	8	**4**	**2 (전부)**	**8**
B	4	16	**4**	**3 (3/4)**	**12**

배낭 (용량 5)

A 2	B 3 (3/4)

① A: 2 ≤ 5 → 전부, 가치 8

② 남은 용량 5 − 2 = 3

③ B: 4 > 3 → 3만 덜어 담기

3 ÷ 4 = 3/4 → 16 × 3/4 = 12

합계 8 + 12 = 20

B부터 담아도 16 + 4 = 20.
무게당 가치가 모두 4라서 5 × 4 = 20입니다.

파랑: 통째로 담은 보물　초록: 일부만 덜어 담은 보물　노랑: 무게당 가치

0/1 배낭: 물건은 통째로만 담기

배낭 용량은 50kg이고, 아래 물건 A·B·C가 각각 1개씩 있습니다.
물건은 통째로 넣거나 빼야 합니다. 잘라서 일부만 넣을 수 없습니다.
50kg 안에서 총가치를 최대로 만드세요. 먼저 가성비 순으로 넣어 봅니다.

물건	무게(kg)	가치	무게당 가치	결정	얻은 가치
A	10	60	**6**	**담음**	**60**
B	20	100	**5**	**담음**	**100**
C	30	120	**4**	**건너뜀**	0

배낭 (용량 50kg)

A 10kg	B 20kg	빈 공간 20kg

C는 30kg짜리 물건 하나입니다. 잘라서 20kg만 담는 것은 규칙 위반입니다.
0 = 넣지 않음, 1 = 통째로 넣음. 반 개를 넣는 선택은 없습니다.

① A 담음: 남은 용량 40kg

② B 담음: 남은 용량 20kg

③ C 30kg: 남은 20kg에 못 넣음

그리디의 답

60 + 100 = 160

연습: 조합을 다 보면 B + C = 220

담은 물건	무게 합(kg)	가치 합	판정
없음	0	0	가능
A	10	60	가능
B	20	100	가능
C	30	120	가능
A + B	30	160	**가능 (그리디의 답)**
A + C	40	180	가능
B + C	**50**	**220**	**가능 · 최대**
A + B + C	60	—	50kg 초과

최적해는 B + C = 220입니다. 그리디의 답 160보다 60 큽니다.

가성비 1등 A를 먼저 넣으면 B와 C를 함께 넣을 수 없습니다.
A를 빼고 B+C를 담으면 무게 20+30=50kg, 가치 100+120=220입니다.

0/1 배낭에서 그리디가 실패하는 이유

① 첫 선택이 최적해에 없다

A(10kg)를 넣으면 40kg이 남아 B(20kg)와 C(30kg)를 함께 넣지 못합니다.

② 분할 배낭과의 차이

분할 배낭이면 A 10kg, B 20kg, C 일부 20kg로 가치 240을 얻습니다.
0/1에서는 C 일부를 넣을 수 없습니다. B+C를 통째로 담는 220이 최적입니다.

③ 모든 조합을 보면?

보물 n개를 담거나 말거나: 2^n가지. 보물이 30개면 10억 가지가 넘습니다.

④ 6장 동적 계획법

같은 부분 문제의 답을 표에 저장해 다시 계산하지 않습니다. 0/1 배낭의 정확한 답은 6장에서 구합니다.

0/1 배낭에서 무게당 가치 그리디는 최적해를 보장하지 못합니다.

그리디가 최적해를 보장하는 두 조건

그리디는 매 단계 지금 기준의 최선을 확정하고 되돌리지 않습니다.
이 방식이 최적해를 낸다고 보장하려면 두 성질을 보여야 합니다.

① 탐욕적 선택 속성

greedy choice property
지금 가장 좋아 보이는 선택을
포함하는 최적해가 있다

② 최적 부분 구조

optimal substructure
그 선택을 하고 남은 문제의 최적해가
전체 최적해의 나머지가 된다

①로 첫 선택이 안전하고, ②로 남은 문제를 같은 방법으로 풀 수 있습니다.

이 둘을 단계마다 이어 붙이면 그리디의 답이 전체 최적해가 됩니다. 둘 중 하나라도 보이지 못하면 보장이 사라집니다.

① 탐욕적 선택 속성 (greedy choice property)

지금 기준으로 가장 좋아 보이는 선택을 해도, 그 선택을 포함하는 최적해가 적어도 하나 있다는 성질입니다.

판별 질문: 이 선택을 포함하는 최적해가 있는가?

예

그 선택을 확정하고 남은 문제로 넘어가도 최적해를 놓치지 않습니다.

아니오

그 선택을 하는 순간 최적해로 가는 길이 모두 막힙니다.

"안전하다"는 다른 선택보다 더 좋다는 뜻이 아니라, 이 선택을 해도 최적해로 가는 길이 남아 있다는 뜻입니다. 최적해가 여러 개라면 그중 하나만 이 선택을 포함해도 됩니다.

교환 논증의 핵심

최적해의 일부를 탐욕 선택으로 바꿔도 여전히 최적해임을 보입니다.

① 어떤 최적해 O에서 시작합니다

O에 탐욕 선택 g가 없다고 합시다.

② O의 일부를 g로 바꿉니다

바꾼 O′도 문제의 조건(겹치지 않음, 용량, 금액)을 지킵니다.

③ 값이 나빠지지 않음을 보입니다

O′의 값이 O보다 나쁘지 않으면 O′도 최적해이고, O′에는 g가 들어 있습니다.

예	바꾸는 것	나빠지지 않는 이유
활동 선택 (5.3)	O의 첫 회의 → 1번 [1, 3)	1번이 더 일찍 끝나 겹치지 않고, 개수 그대로
분할 배낭 (5.4)	안의 B·C 1kg → 밖의 A 1kg	1kg당 600 G → 700 G, 가치가 늘어남
표준 동전 (5.2)	100원 5개 → 500원 1개	같은 금액을 동전 4개 적게

결론은 늘 같습니다. 탐욕 선택을 포함하는 최적해가 있다 → 탐욕적 선택 속성이 성립합니다.

② 최적 부분 구조 (optimal substructure)

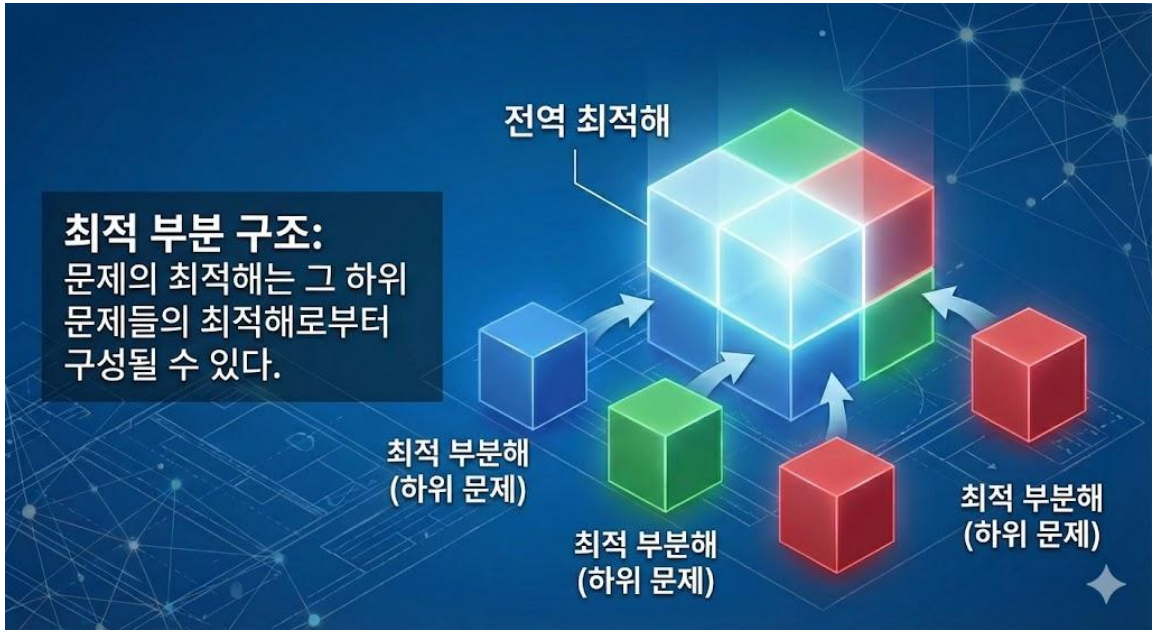
선택하고 남은 부분 문제(하위 문제)의 최적해에 그 선택을 더하면 전체 최적해가 된다는 성질입니다. 거꾸로 말하면 전체 최적해 안에 부분 문제의 최적해가 들어 있습니다.

예: 활동 선택

1번 [1, 3)을 고르면 남은 문제는 "3시 이후에 시작하는 회의 중 최대 개수 고르기"입니다. 그 최적해에 1번을 더하면 전체 최적해가 됩니다.

그리디는 선택을 확정한 뒤 남은 문제를 같은 방법으로 풉니다. 그래서 이 성질이 필요합니다.

4장의 최대 구간 합처럼 분할 정복으로 푸는 최적화 문제, 6장의 동적 계획법도 이 성질을 씁니다.



작은 문제의 최적해(그림의 최적 부분해)를 모아 전체 최적해(그림의 전역 최적해)를 만듭니다.

주의: ②만으로는 그리디를 보장할 수 없다

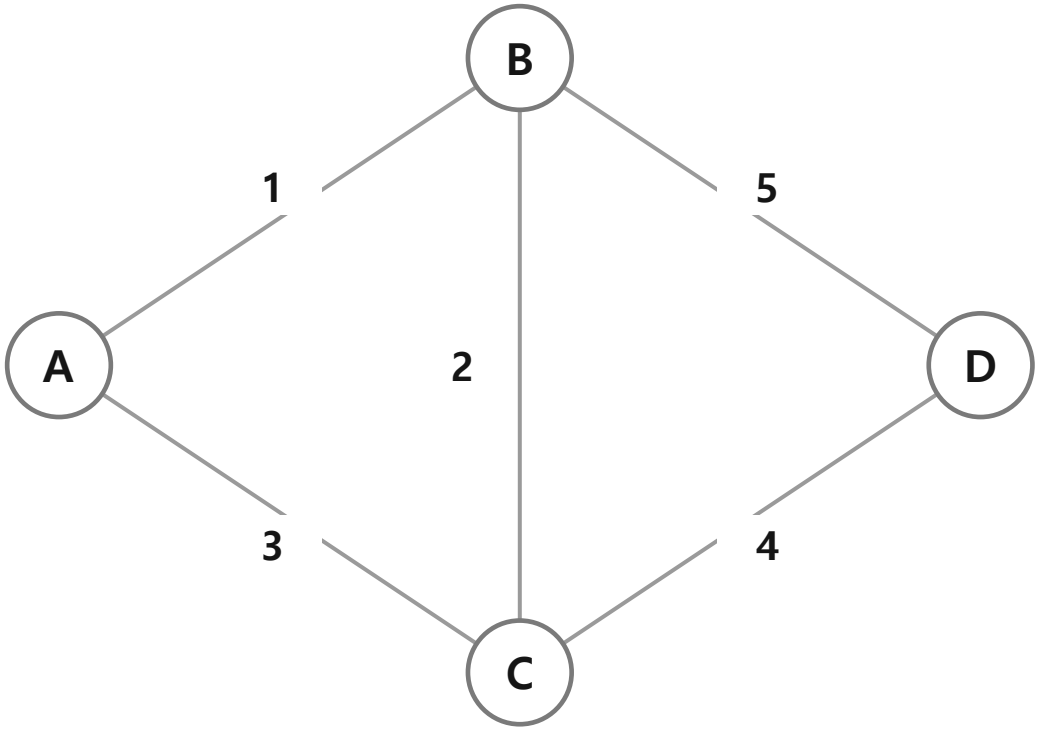
문제 (그리디의 기준)	① 탐욕적 선택	② 최적 부분 구조	그리디의 결과
활동 선택 (종료가 빠른 순)	성립	성립	**최적**
분할 배낭 (무게당 가치 순)	성립	성립	**최적**
1·3·4원 동전 (큰 동전부터)	**깨짐**	성립	보장 못 함 → 6장 DP
0/1 배낭 (무게당 가치 순)	**깨짐**	성립	보장 못 함 → 6장 DP
최단 경로, 간선 비용 ≥ 0 (다익스트라)	성립	성립	**최적 (5.8)**
최단 경로, 음수 간선 (다익스트라)	**깨짐**	성립	틀릴 수 있음 (5.8)

1·3·4원 동전과 0/1 배낭은 ①이 깨진 예입니다. ②는 성립하므로 6장 동적 계획법으로 풀립니다.

최단 경로는 음수 간선이 있어도(한 바퀴 돌수록 짧아지는 순환이 없다면) ②가 성립합니다. 다익스트라가 옳으려면 "간선 비용이 모두 0 이상이면, 아직 확정하지 않은 곳 중 지금 가장 가까운 곳은 더 가까워지지 않는다"는 ①의 근거가 더 필요합니다.

그래프 용어: 정점, 간선, 가중치

도시와 도로를 그래프로 나타냅니다. 왼쪽은 정점 4개, 간선 5개짜리 그래프입니다.



원: 정점 선: 간선 선 위의 수: 가중치

용어	뜻	이 그림에서
정점 (vertex)	점. 예: 도시	V = 4 (A~D)
간선 (edge)	두 정점을 잇는 선. 예: 도로	E = 5
가중치 (weight)	간선에 붙은 비용	A – B 1, C – D 4
경로 (path)	간선을 따라 이어진 길	A – B – D
사이클 (cycle)	출발점으로 돌아오는 닫힌 길	A – B – C – A
신장 (spanning)	모든 정점을 포함한다	A~D 모두

V는 정점의 수, E는 간선의 수를 나타내는 기호입니다.

최소 신장 트리 (MST)

신장 트리: 모든 정점을 잇고, 사이클이 없는 간선 모음

최소 신장 트리(Minimum Spanning Tree, MST): 신장 트리 중 가중치 합이 가장 작은 것

모두 연결

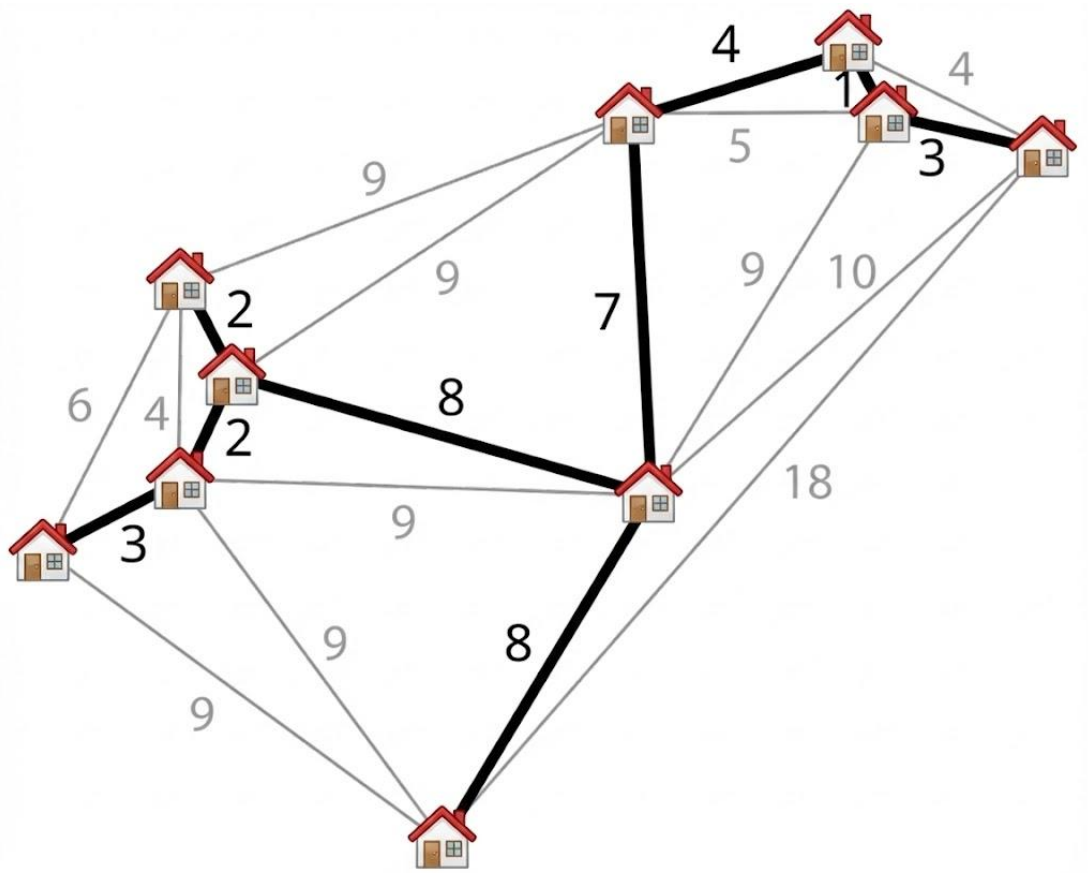
어느 정점도 떨어져 있으면 안 됩니다.

사이클 없음

사이클이 있으면 그중 간선 하나를 빼도 모두 연결된 채입니다. 모두 잇는 데 사이클은 필요 없습니다.

간선은 정확히 V − 1개

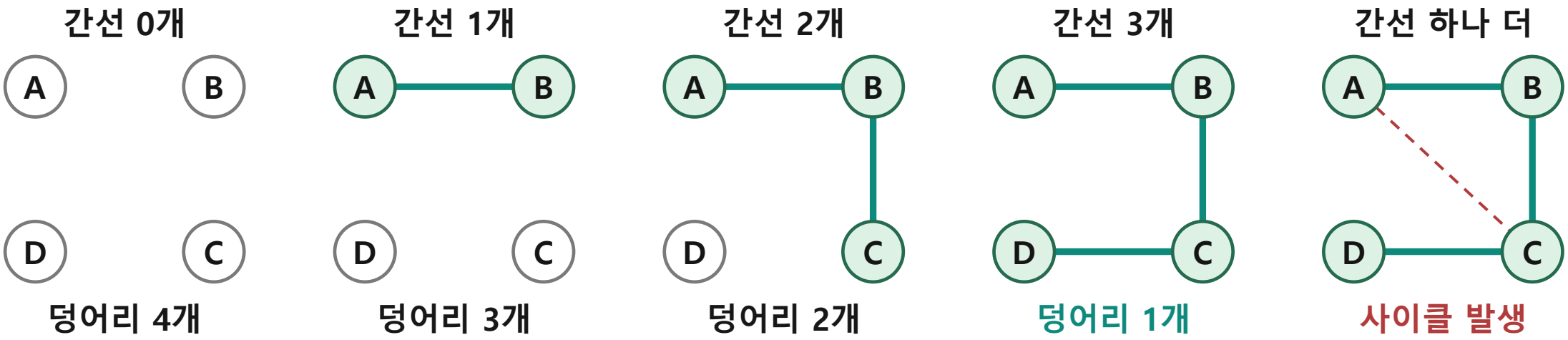
정점이 V개면 간선은 V − 1개입니다(다음 슬라이드).



굵은 선 9개 = 최소 신장 트리(정점 10개, 합 38). 집 = 정점, 선 = 간선, 수 = 가중치 (맨 위 짧은 굵은 선 = 1)

신장 트리의 간선은 V − 1개

덩어리 = 서로 경로로 이어진 정점들의 묶음입니다. 서로 다른 두 덩어리를 잇는 간선을 하나 더할 때마다 덩어리가 하나씩 줄어듭니다.



정점 V개를 한 덩어리로 이으려면 간선 V − 1개가 필요합니다.

간선을 하나 더 넣으면 이미 이어진 두 정점을 또 잇게 되어 덩어리는 줄지 않고 사이클만 생깁니다. 그래서 신장 트리의 간선은 정확히 V − 1개입니다.

크루스칼: 가장 싼 간선부터 잇는다

① 정렬
모든 간선을 가중치 오름차순으로 줄 세웁니다.

② 싼 간선부터 검사
사이클을 만들지 않으면 채택하고, 만들면 버립니다.

③ 멈춤
채택한 간선이 V − 1개가 되면 끝납니다.

사이클 판정: 간선의 두 끝이 이미 같은 덩어리이면 사이클

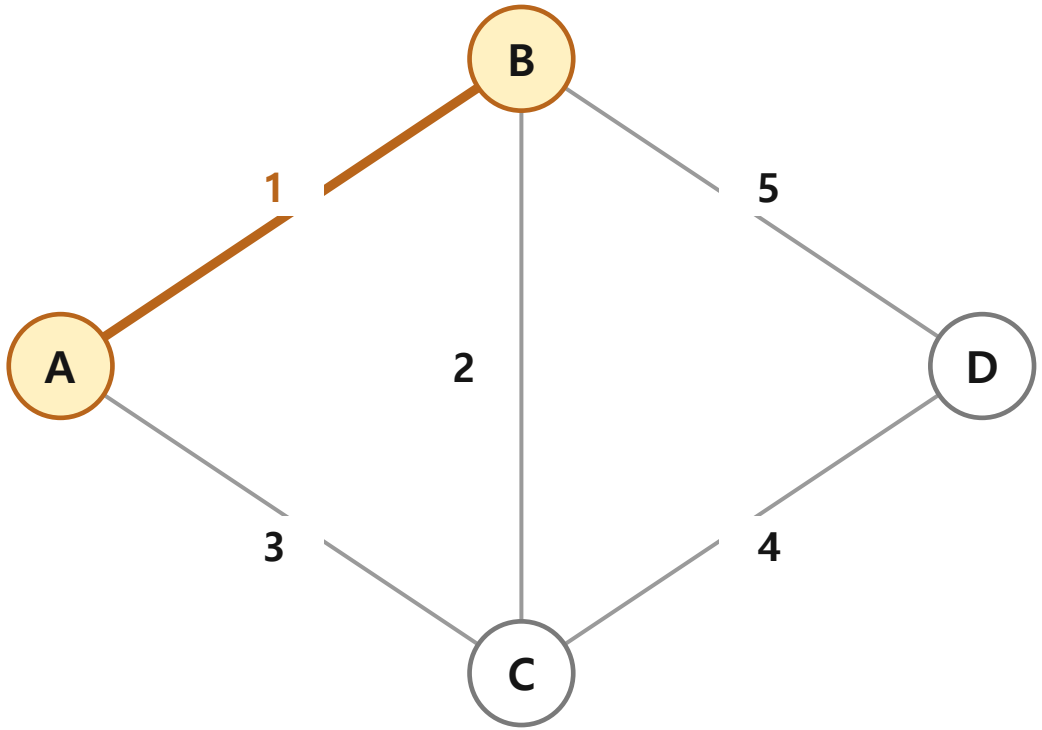
선택 = 남은 간선 중 가장 싼 것, 적정성 검사 = 사이클이 생기지 않는가, 해답 검사 = 간선이 V − 1개인가



왼쪽 그림: 가장 싼 간선부터 잇는다 오른쪽 그림: 이미 이어진 두 곳을 또 이으면 사이클이 되므로 버린다

크루스칼 ① AB 1 채택

간선을 정렬하면 AB 1, BC 2, AC 3, CD 4, BD 5입니다.
정점이 4개이므로 간선 3개(V − 1)에서 멈춥니다.



간선	가중치	결과
AB	**1**	**채택**
BC	2	
AC	3	
CD	4	
BD	5	

A와 B는 서로 다른 덩어리 → 채택

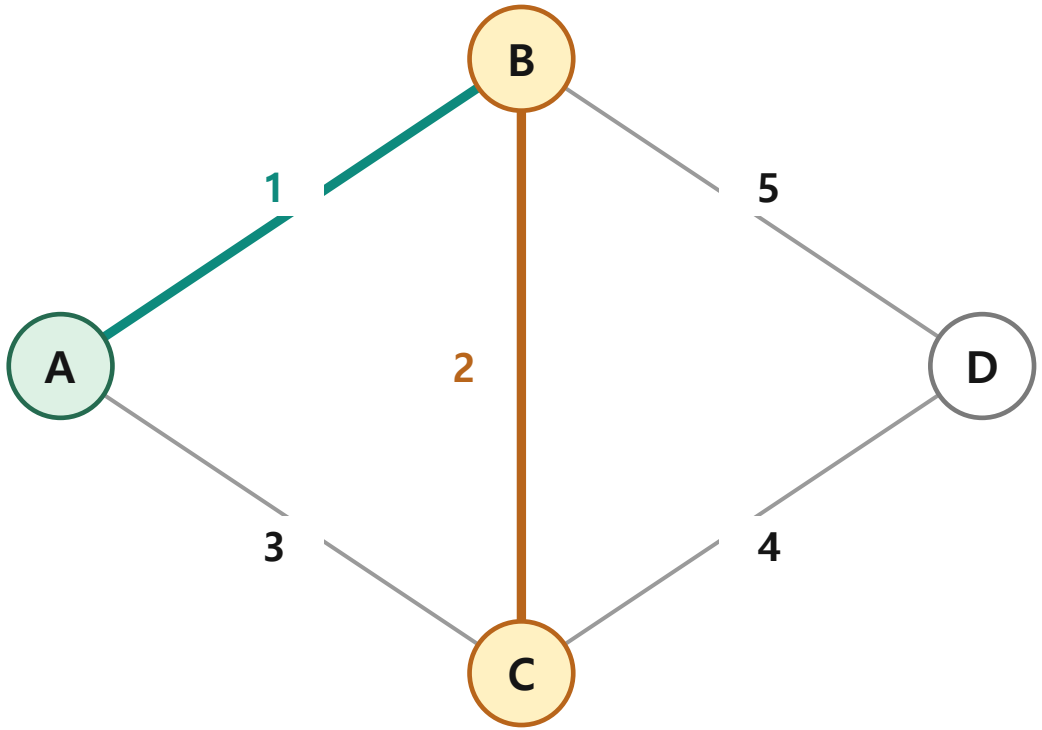
덩어리: {A, B} {C} {D}

합 1

청록: 채택한 간선 주황: 지금 보는 간선(노랑 정점: 그 두 끝) 빨강 점선: 버린 간선 초록 정점: 채택한 간선에 닿은 정점

연습 ② BC 2 채택

B({A, B})와 C(혼자)는 서로 다른 덩어리입니다 → 채택.
덩어리: {A, B, C} {D}



간선	가중치	결과
AB	1	채택
BC	**2**	채택
AC	3	
CD	4	
BD	5	

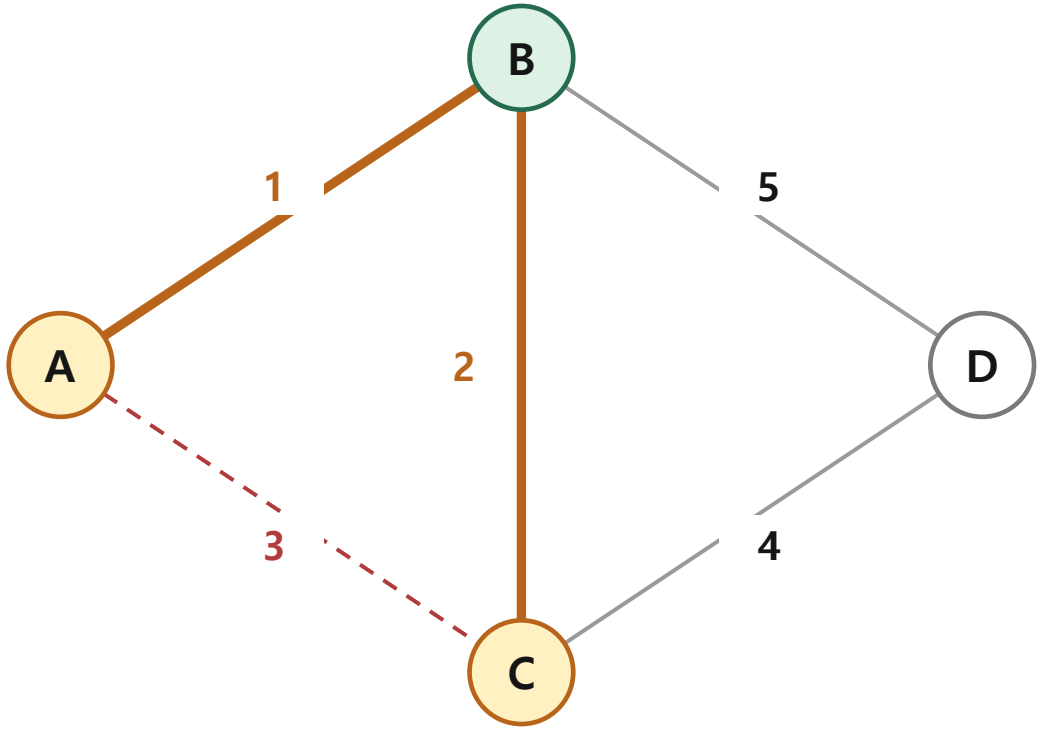
채택한 간선 2개

합 1 + 2 = 3

청록: 채택한 간선 주황: 지금 보는 간선(노랑 정점: 그 두 끝) 빨강 점선: 버린 간선 초록 정점: 채택한 간선에 닿은 정점

연습 ③ AC 3 사이클 → 버림

A와 C는 이미 A – B – C로 이어져 있습니다.
AC를 더하면 사이클 A – B – C – A가 생기므로 버립니다.



간선	가중치	결과
AB	1	채택
BC	2	채택
AC	**3**	사이클
CD	4	
BD	5	

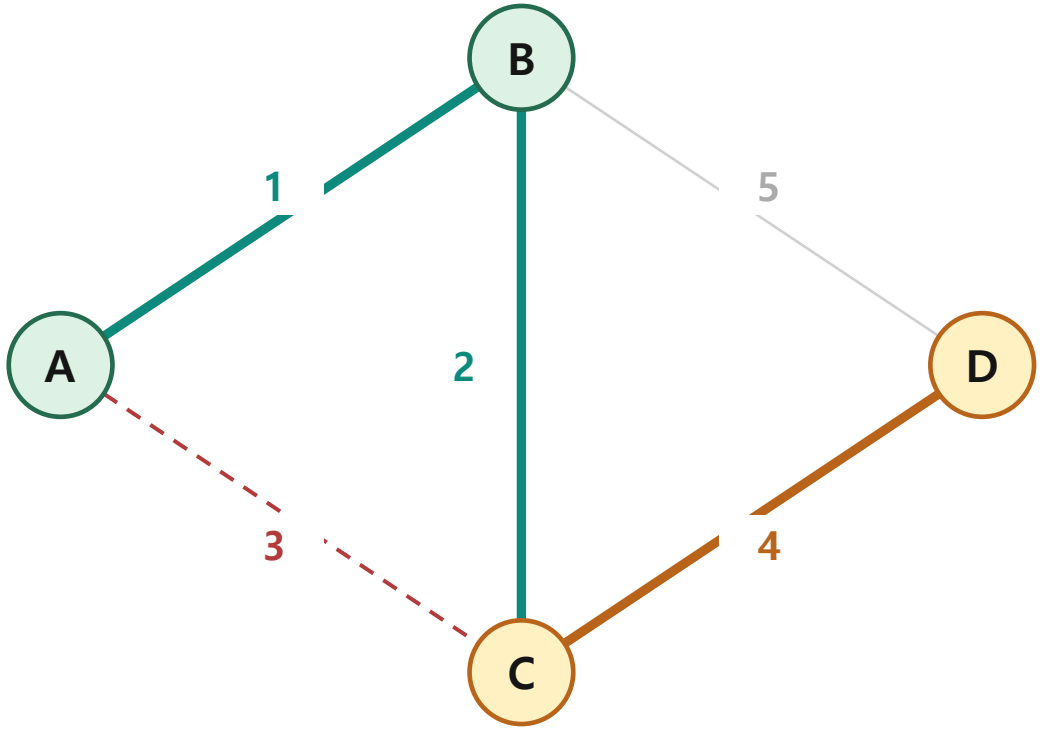
채택한 간선 그대로 2개

합 3

주황: A와 C를 이미 잇는 길 빨강 점선: 버린 AC(노랑 정점: 그 두 끝) 초록 정점: 채택한 간선에 닿은 정점

연습 ④ CD 4 채택 · 합 7

D가 연결되어 간선 3개(= V − 1)가 되었으므로 멈춥니다.
BD(5)는 확인하지 않습니다.



간선	가중치	결과
AB	1	채택
BC	2	채택
AC	3	사이클
CD	**4**	채택
BD	5	확인하지 않음

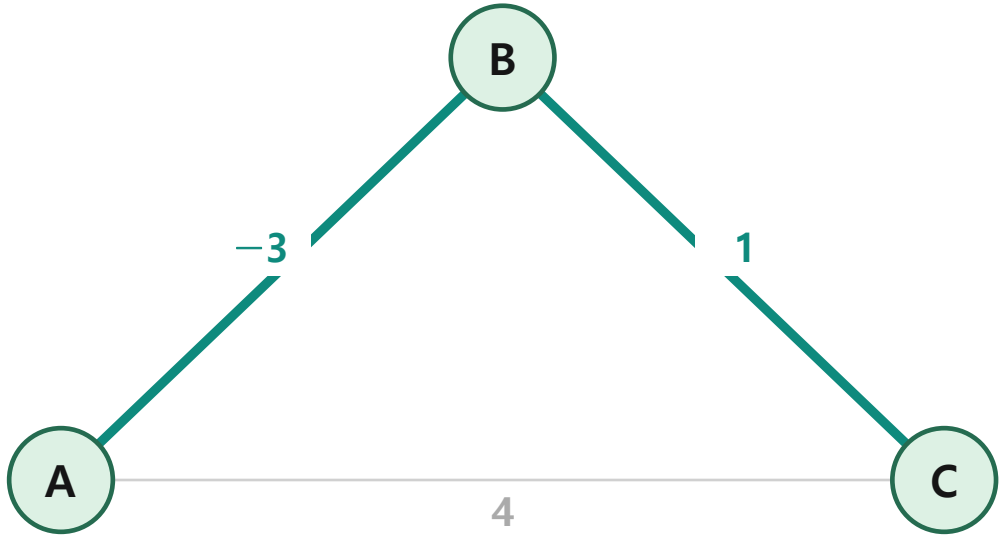
AB → BC → (AC 사이클) → CD → 멈춤

합 1 + 2 + 4 = 7

청록: 채택 주황: 지금 보는 간선(노랑 정점: 그 두 끝) 빨강 점선: 버림 회색: 확인하지 않음 초록 정점: 채택한 간선에 닿은 정점

연습: 음수 가중치 AB = −3

정점 A, B, C와 AB = −3, BC = 1, AC = 4.
정렬하면 AB(−3)가 맨 앞입니다.



간선	가중치	결과
AB	−3	① 채택
BC	1	② 채택
AC	4	확인하지 않음

간선 2개 = V − 1 → 멈춤

합 −3 + 1 = −2

크루스칼은 가중치를 크기 비교에만 씁니다. 음수여도 그대로 성립합니다. 음수 간선에서 틀릴 수 있는 다익스트라(5.8)와 다릅니다.

청록: 채택한 간선 회색: 확인하지 않은 간선

한 시작점에서 모든 정점까지의 최단 거리

시작점 하나에서 **다른 모든 정점까지** 가장 짧은 경로의 길이를 구합니다.

dist[v]

시작점에서 v까지 지금까지 찾은 가장 짧은 거리.
처음엔 시작점 0, 나머지 ∞

확정

dist[v]가 더 줄어들지 않는다고 결론 내리는 것

다익스트라의 선택

아직 확정하지 않은 정점 중 dist가 가장 작은 곳부터 확정합니다

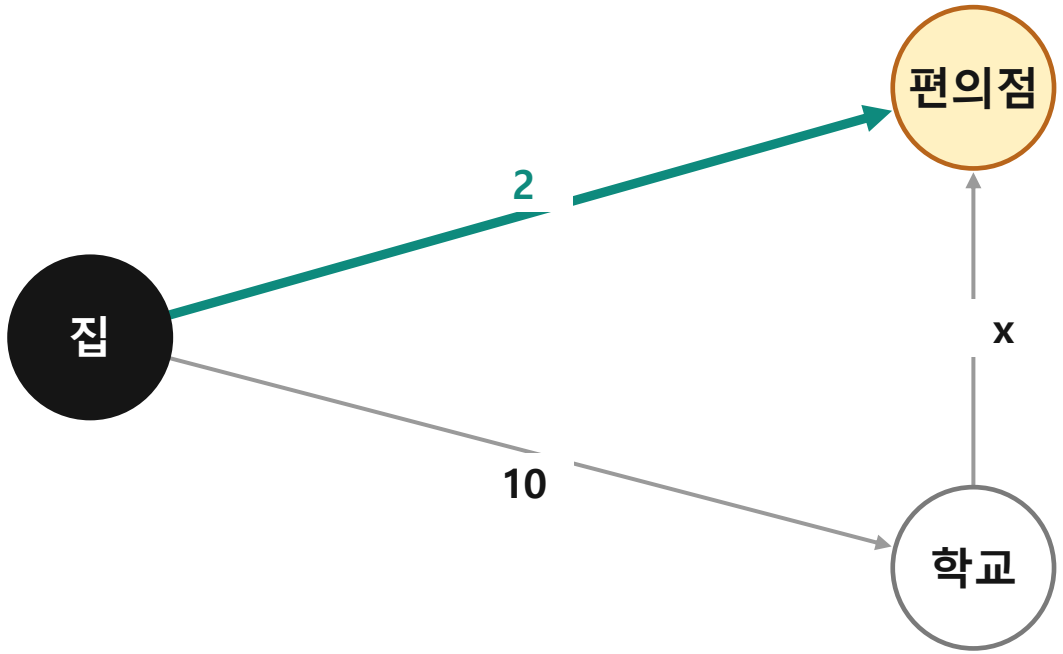
내비게이션 길 찾기, 네트워크 라우팅(예: OSPF)에 다익스트라나 그 변형이 쓰입니다.



그림: 집에서 편의점 2분, 학교 10분, 병원 15분. 아직 확정하지 않은 곳 중 가장 가까운 편의점부터 확정합니다.

왜 가장 가까운 곳을 확정해도 안전한가

편의점 2분, 학교 10분입니다. 학교를 거쳐 편의점에 가면
10 + x분(x ≥ 0)이라 2분보다 짧아질 수 없습니다.



학교를 거쳐 편의점에 가는 길

10 + x ≥ 10 > 2

학교까지 이미 10분입니다. 간선 비용이 0 이상이면 길을 더 갈수록 거리가 줄지 않으므로 편의점 2분은 더 줄어들 수 없습니다.

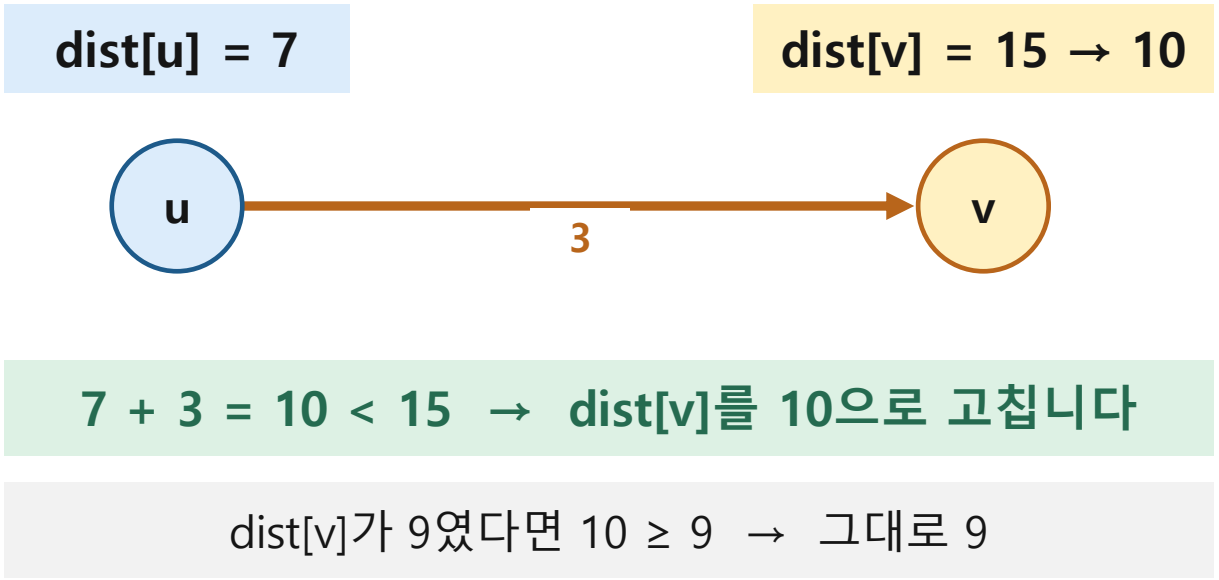
x = −9라면?

10 − 9 = 1 < 2. 음수 간선이 있으면 이 논리가 깨집니다.

검정: 시작점 노랑: 지금 가장 가까운 미확정 정점 x: 학교 → 편의점 비용

완화 · dist[v] 15 → 10

u를 거쳐 v로 가는 길이 더 짧으면 dist[v]를 고칩니다.
연습문제 예: dist[u] = 7, 가중치 3, dist[v] = 15



완화(relaxation)

dist[v] = min(dist[v], dist[u] + w)

relax.pseudo

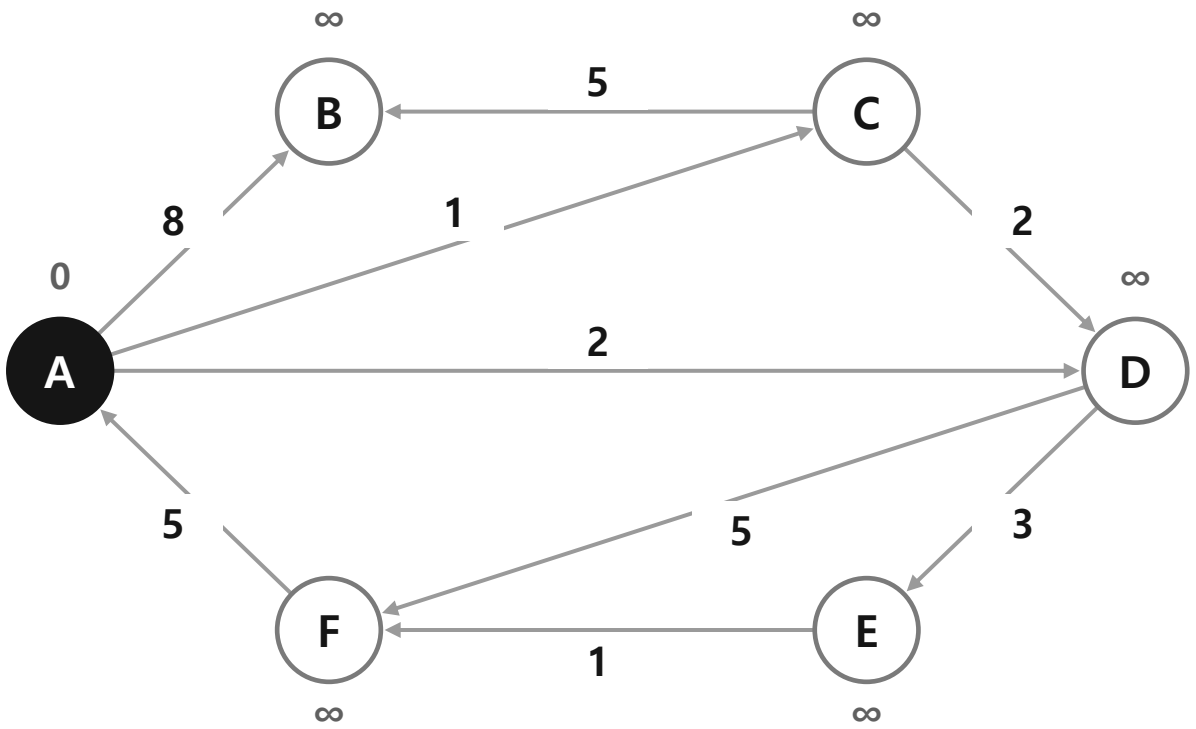
```
if dist[u] + w < dist[v] then
    dist[v] <- dist[u] + w
```

w: u → v 간선의 비용. 더 짧을 때만 고치고, 같거나 길면 그대로 둡니다.

파랑: dist를 아는 정점 u 노랑: 고치려는 정점 v 주황: 완화하는 간선 u → v

예제: A에서 출발 · 초기화

방향 그래프 A~F, 시작점은 A입니다. dist[A] = 0, 나머지는 아직 길을 모르니 ∞로 둡니다.



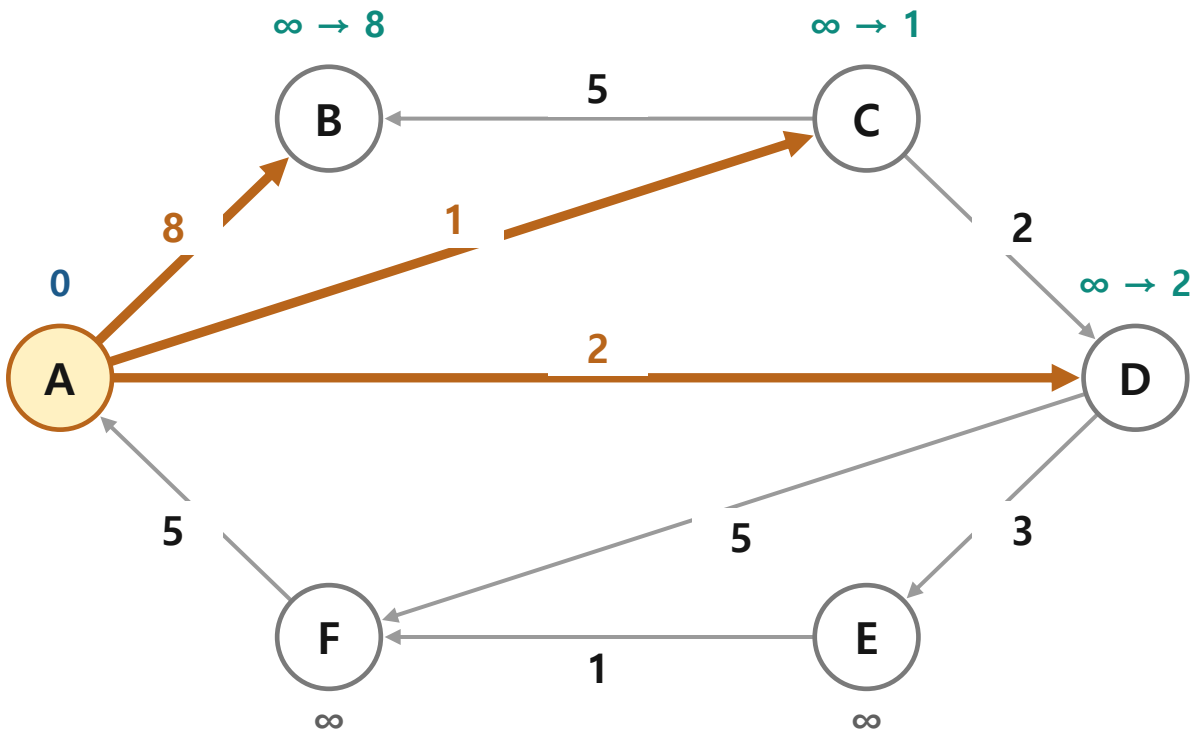
정점	dist	상태
A	0	미확정
B	∞	미확정
C	∞	미확정
D	∞	미확정
E	∞	미확정
F	∞	미확정

다음: 미확정 중 dist가 가장 작은 A(0)를 확정

검정: 시작점 정점 옆 숫자 = 지금의 dist ∞: 아직 길을 모름

① A 확정 · B 8, C 1, D 2

미확정 중 가장 작은 A(0)를 확정합니다. A에서 나가는 간선으로 B, C, D를 고칩니다.



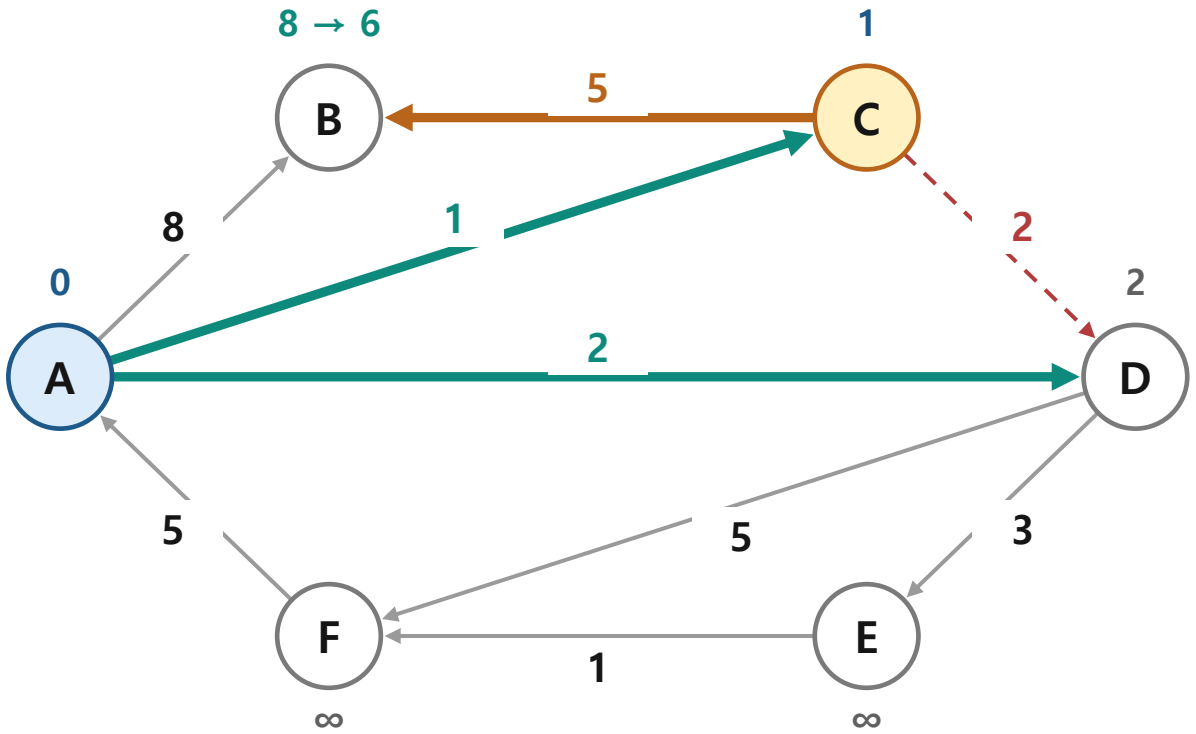
정점	dist	상태
A	0	이번 확정
B	∞ → 8	미확정
C	∞ → 1	미확정
D	∞ → 2	미확정
E	∞	미확정
F	∞	미확정

0 + 8 = 8, 0 + 1 = 1, 0 + 2 = 2. 다음: 가장 작은 C(1)

노랑: 이번에 확정 파랑: 확정 주황: 이번에 완화한 간선 청록: 지금 dist를 만든 간선 빨강 점선: 줄지 않음

② C 확정 · B 8 → 6

가장 작은 C(1)를 확정합니다. C → B로 1 + 5 = 6 < 8이라 B를 6으로 고칩니다.



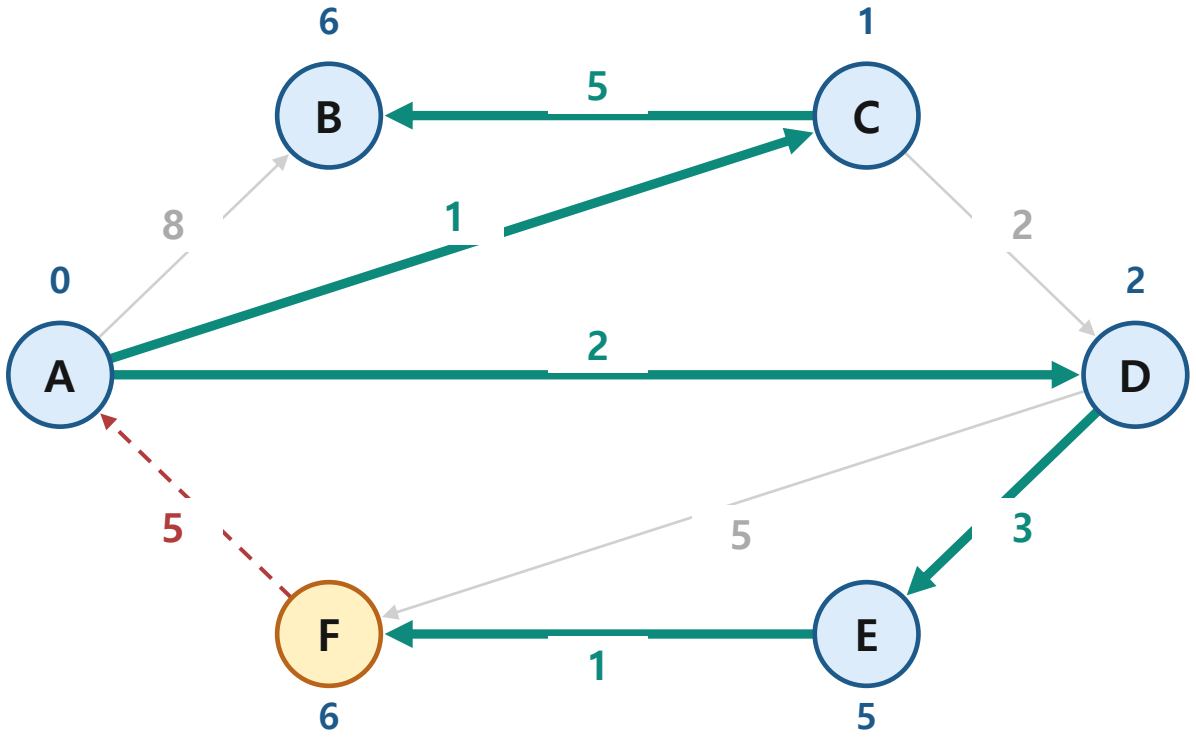
정점	dist	상태
A	0	확정
B	8 → 6	미확정
C	1	이번 확정
D	2	미확정
E	∞	미확정
F	∞	미확정

C → D는 1 + 2 = 3 ≥ 2라 그대로입니다. 다음: D(2)

노랑: 이번에 확정 파랑: 확정 주황: 이번에 완화한 간선 청록: 지금 dist를 만든 간선 빨강 점선: 줄지 않음

계산 결과: 모든 정점의 최단 거리

F(6)를 확정합니다. F → A는 A가 이미 확정되어 고치지 않습니다(6 + 5 = 11 ≥ 0이기도 합니다).



정점	dist	상태
A	0	확정
B	6	확정
C	1	확정
D	2	확정
E	5	확정
F	6	이번 확정

모든 정점 확정. 청록 간선 5개 = 최단 경로 트리. A → D → E → F = 2 + 3 + 1 = 6

청록: 최단 경로 트리 빨강 점선: 줄지 않은 간선 연회색: 쓰지 않은 간선

다익스트라 의사코드 (배열판)

dijkstra.pseudo

```
DIJKSTRA(G, start):
    dist[모든 v] <- ∞;  visited[모든 v] <- False;  dist[start] <- 0
    V번 반복:
        u <- 미확정 중 dist가 가장 작은 정점 (모두 ∞면 break);  visited[u] <- True   // 확정
        for u에서 나가는 간선 (u, v, w):
            if not visited[v] and dist[u] + w < dist[v] then
                dist[v] <- dist[u] + w          // 완화
    return dist
```

선택: 확정하지 않은 정점 중 dist가 가장 작은 정점

배열을 처음부터 끝까지 훑으면 V번 비교, 이것을 V번 반복 → $O(V^2)$

확정: visited[u] <- True

확정한 정점은 다시 고치지 않습니다

완화: u에서 나가는 간선마다

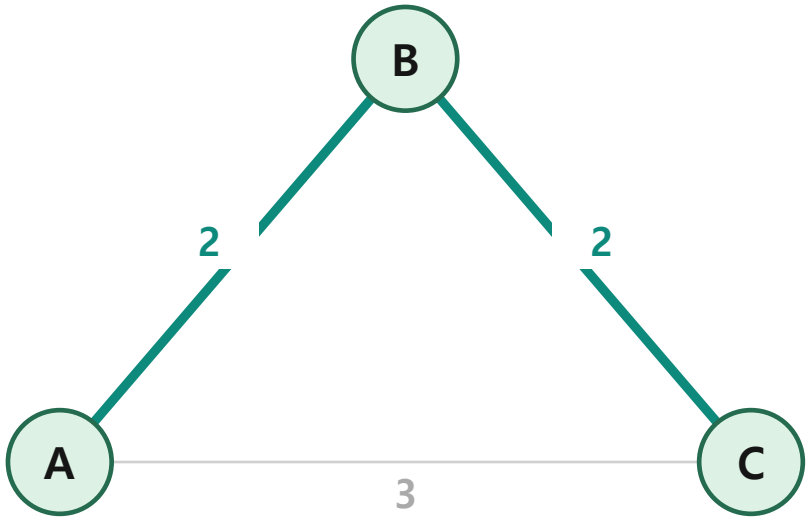
인접 행렬이면 한 행 V칸을 훑으므로 반복마다 O(V)

배열판 전체 $O(V^2)$: 전체 구현은 보충자료에 있습니다

최소 신장 트리와 최단 경로 트리는 다를 수 있다

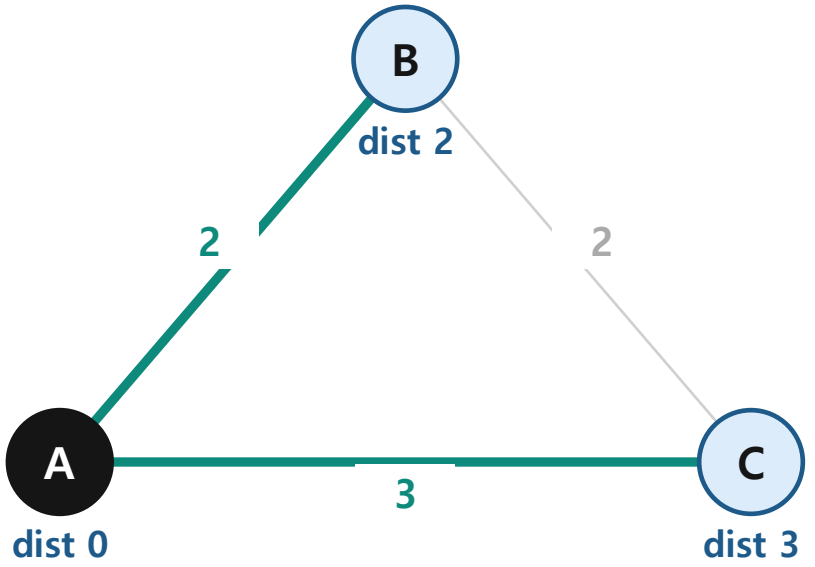
MST는 간선 합을, 최단 경로 트리는 시작점에서 각 정점까지의 거리를 최소로 합니다. 같은 그래프에서도 둘은 다를 수 있습니다.

최소 신장 트리(크루스칼)



간선 합 2 + 2 = 4 (최소) · A → C 거리 4

최단 경로 트리(다익스트라, 시작 A)

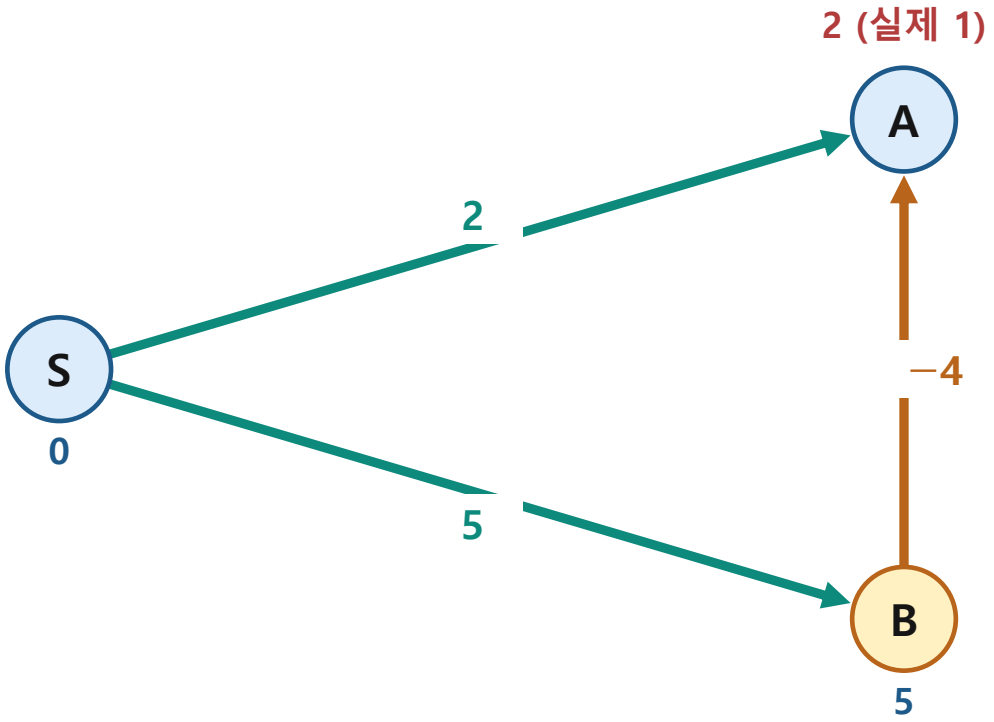


간선 합 2 + 3 = 5 · A → C 거리 3 (최단)

청록: 트리 간선 연회색: 트리에 없는 간선 방향 없는 간선은 양방향 간선 2개로 보고 다익스트라를 돌립니다

음수 간선 반례: 실제 최단 1을 놓친다

B(5)를 확정하고 B → A를 보면 5 + (−4) = 1 < 2입니다.
하지만 A는 이미 확정되어 고치지 않습니다.



정점	dist	상태
S	0	확정
A	2	확정
B	5	이번 확정

확정 방식의 답: A = 2

실제 최단은 S → B → A = 1

A를 2로 확정한 뒤 다시 처리하지 않으므로, A에서 나가는 간선이 있었다면 그 뒤 정점도 틀립니다.

주황: 1을 만드는 간선 B → A 음수 간선이 있으면 벨만-포드 같은 다른 알고리즘을 씁니다

5장 정리

01 그리디는 되돌아보지 않는다

매 단계 지금 기준의 최선을 확정하고 번복하지 않습니다. 선택 → 적정성 검사 → 해답 검사를 해답이 될 때까지 반복합니다.

02 최적을 보장하는 두 조건

탐욕적 선택 속성(이 선택을 포함하는 최적해가 있다)과 최적 부분 구조. "항상 최적"은 반례 하나로 반박합니다.

03 정렬 기준이 곧 전략

활동 선택은 종료가 빠른 순, 쪼갤 수 있는 배낭은 가성비가 큰 순, 크루스칼은 가중치가 작은 순.

04 최적을 보장하는 경우

배수 조건의 거스름돈, 활동 선택, 쪼갤 수 있는 배낭(가성비 순), 크루스칼 MST, 다익스트라(간선 비용이 모두 0 이상일 때).

05 보장하지 못하는 경우

{1, 3, 4} 동전, 0/1 배낭, 음수 간선이 있는 다익스트라.

06 다음은 동적 계획법

0/1 배낭과 일반 동전 문제는 남은 상태별 답을 저장하는 동적 계획법으로 이어집니다.

Python · C 대응표

같은 알고리즘을 두 언어로 쓸 때 자주 바뀌는 부분입니다.

하는 일	Python	C
정렬	list.sort(key=..., reverse=True)	qsort(배열, 개수, 크기, 비교 함수)
큰 것부터 보기	coins.reverse()	for (i = N - 1; i >= 0; i--)
무한대	float("inf"), float("-inf")	long long, INF = 10^{18}
최솟값 꺼내기	heapq.heappush / heappop	배열을 훑어 최소 찾기 O(V)
묶음 자료	class Item, 튜플 (w, u, v)	typedef struct { ... } Item;
그래프(다익스트라)	딕셔너리 {정점: {이웃: 비용}}	graph[u][v], -1 = 없음, 0 = 유효
입력	sys.stdin.readline().split()	scanf("%d %d", &N, &K)
몫 · 나머지	K // coin, K % coin	K / coins[i], K % coins[i]

비교 함수는 a가 b보다 앞에 와야 하면 음수, 같으면 0, 뒤에 와야 하면 양수를 돌려줍니다. 오름차순이면 앞이 작을 때 음수입니다(배낭의 by_ratio는 반대).

INF는 큰 정수입니다. 더하기 전에 도달 여부와 자료형 범위를 확인합니다.