

청년 AI 아카데미 22기 알고리즘 실습

그리디기법 & 동적계획법

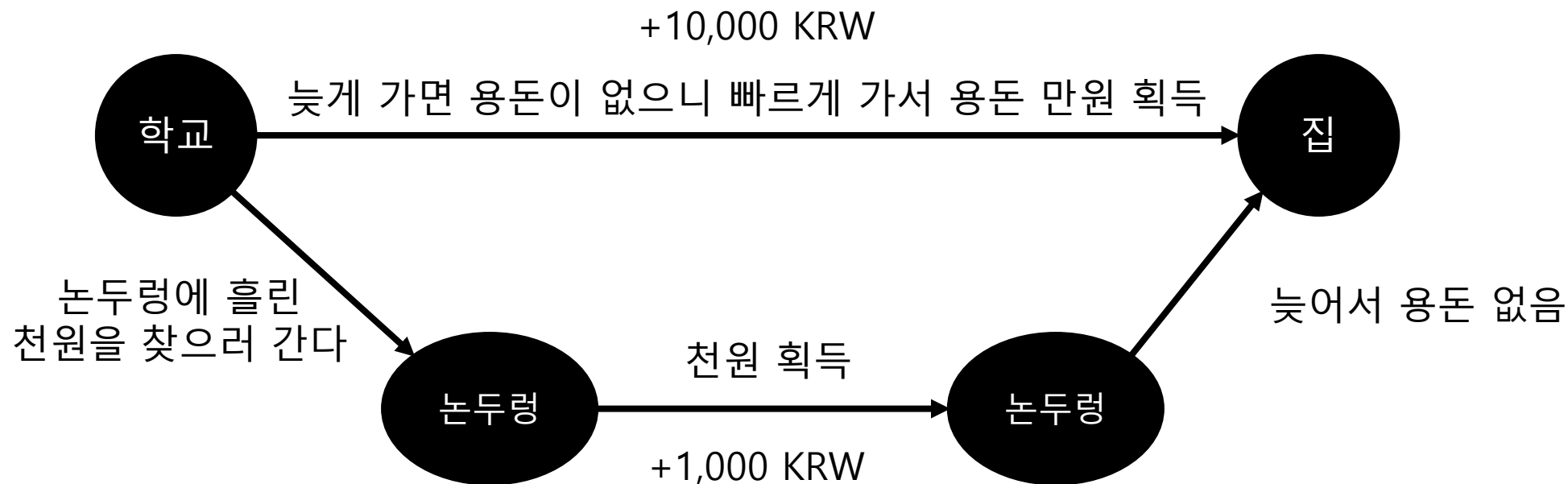
Today

- 그리디 기법
 - 세금 징수
 - 무거운 용액
- 다이나믹 프로그래밍
 - 피보나치 수열 2 (1차원 DP)
 - 세계암기대회 (2차원 DP)

Greedy Method?

Greedy의 기본: 현재 상황만 보고 가장 이득을 취할 수 있는 방향으로 움직인다!

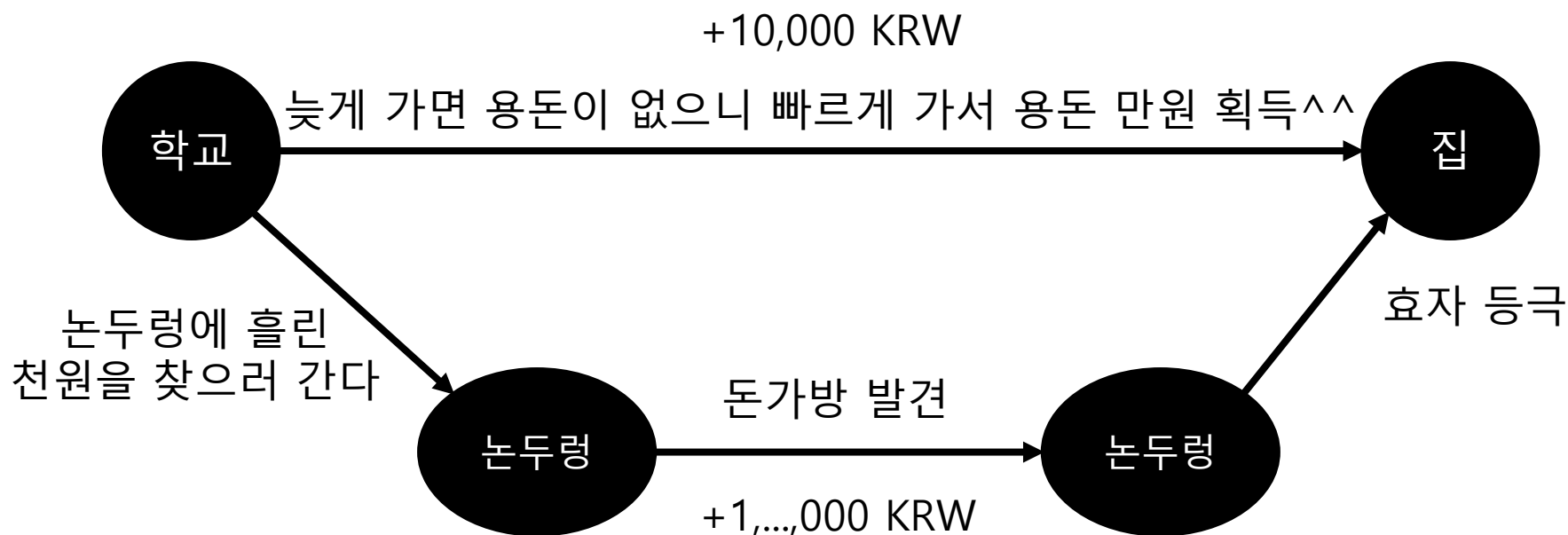
- 특정 상황에서 최적의 값을 찾는다.
- 알고리즘이 단순하고 빠르기 때문에 근사 알고리즘을 설계할 때 자주 채택한다.



Greedy Method?

Greedy의 기본: 현재 상황만 보고 가장 이득을 취할 수 있는 방향으로 움직인다!

- 특정 상황에서 최적의 값을 찾는다.
- 알고리즘이 단순하고 빠르기 때문에 근사 알고리즘을 설계할 때 자주 채택한다.



01. 세금 징수

특정 금액이 주어졌을 때, 해당 금액을 만드는 동전(지폐)의 최소 개수를 구합니다.

동전 단위 : 50000, 10000, 5000, 1000, 500, 100

예시 : 74100



VS



× 741

01. 세금 징수

특정 금액이 주어졌을 때, 해당 금액을 만드는 동전(지폐)의 최소 개수를 구합니다.

Greedy : 작은 단위에서부터? **큰 단위에서부터?**

*Hint: 하나씩 빼기보다는 나누기와 나머지를 활용!

$$9 = 5 + 1 + 1 + 1 + 1$$

$$9 // 5 = 1$$

$$9 \% 5 = 4$$

01. 세금 징수

```
t = int(input())  
for _ in range(t):  
    n = int(input())
```

가격이 높은 동전부터 최대한 많이 사용!

01. 세금 징수

주의!!

그리디 기법이 최적값을
항상 도출하는지 생각해야한다.

Ex1: 사용 가능한 동전 [100,500]

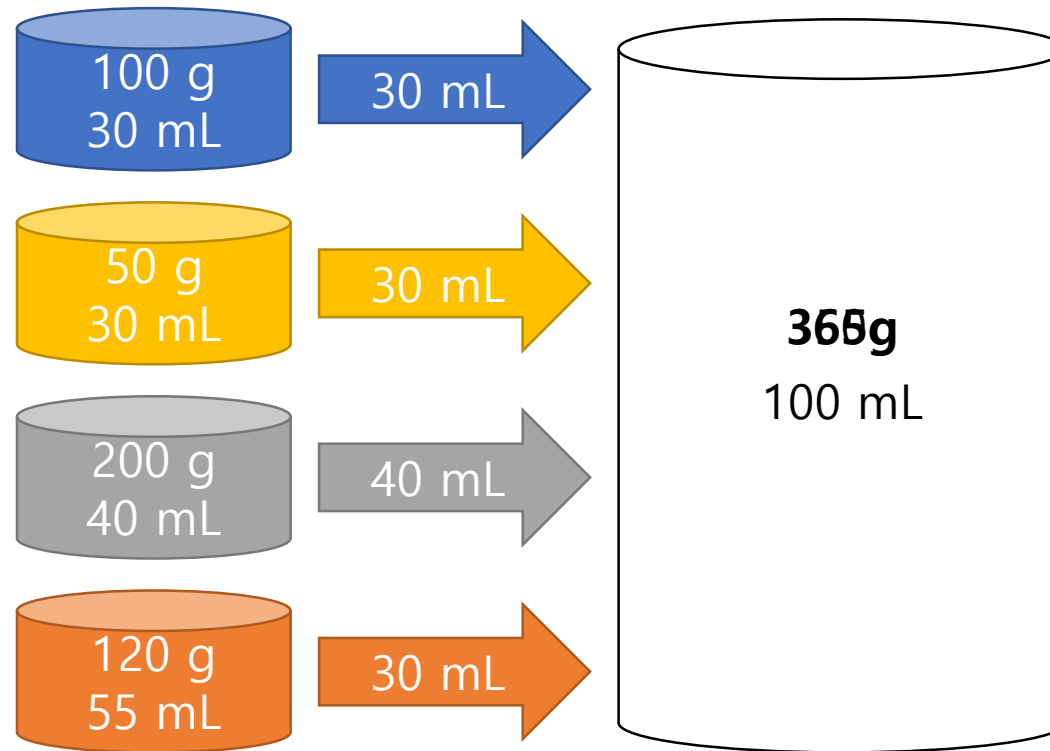
800원을 지불하는 방법 $\rightarrow 800 = 500 + 100 \times 3$ (4개)

Ex2: 사용 가능한 동전 [100,400,500]

800원을 지불하는 방법 $\rightarrow 800 = 400 \times 2$ (2개)

02. 무거운 용액

여러 용액의 총 부피와 총 무게가 주어졌을 때, 주어진 용액을 합성하여 특정 부피의 용액을 만들 때 합성 용액의 최대 무게를 계산합니다.



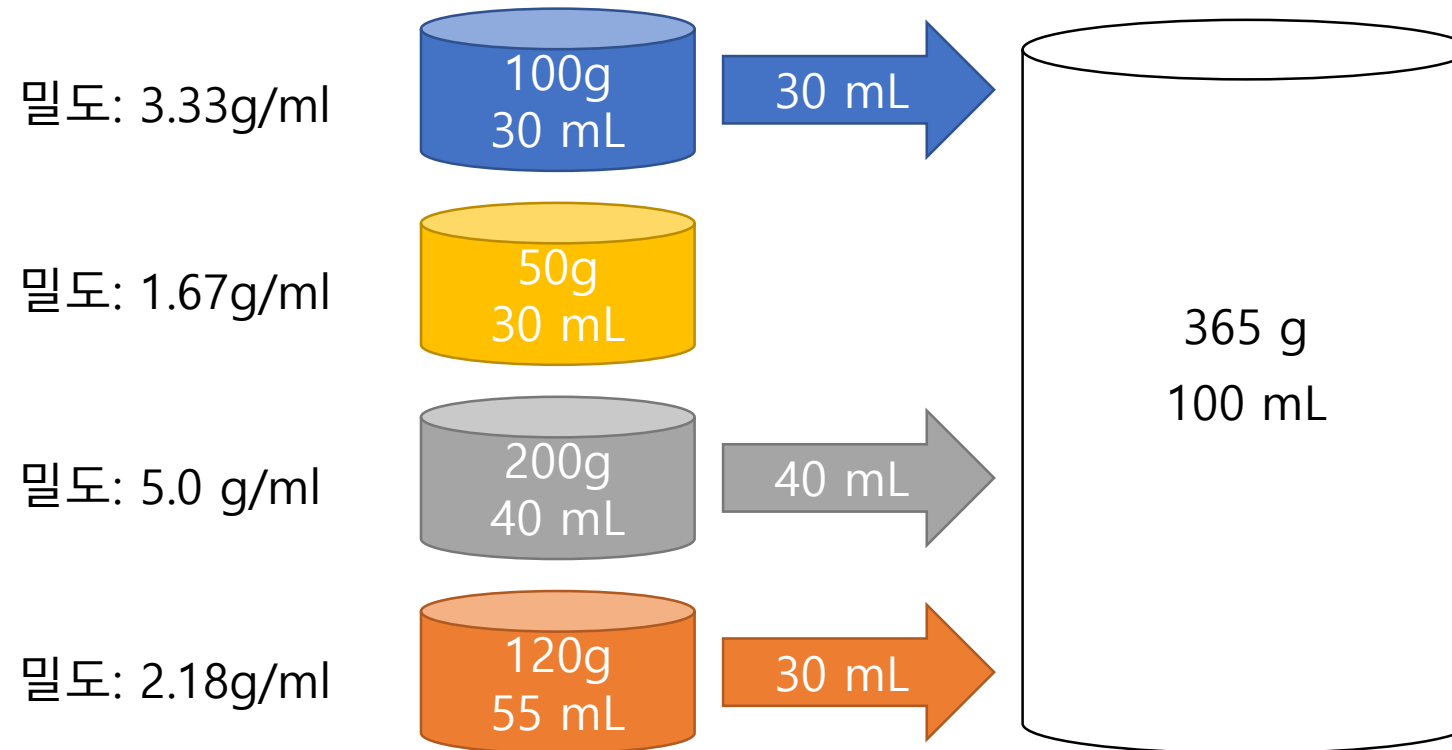
02. 무거운 용액

무거운 용액을 만들 때 가장 좋은 용액은 무엇인가요?

→ 보다 작은 공간을 차지하면서, 보다 더 무거운 용액

→ 같은 공간을 차지할 때 더 무거운 용액

→ **부피당 질량**이 무거운 용액



02. 무거운 용액

1. 질량 / 부피 로 내림차순 정렬
 2. 질량 / 부피가 큰 용액부터 차례대로 넣는다.
- 그 용액을 온전히 다 넣을 수 있다면?
→ 그 용액을 다 못 넣는다면?

밀도: 5.0 g/ml



40 mL (200 g)

밀도: 3.33g/ml



30 mL (100 g)

밀도: 2.18g/ml



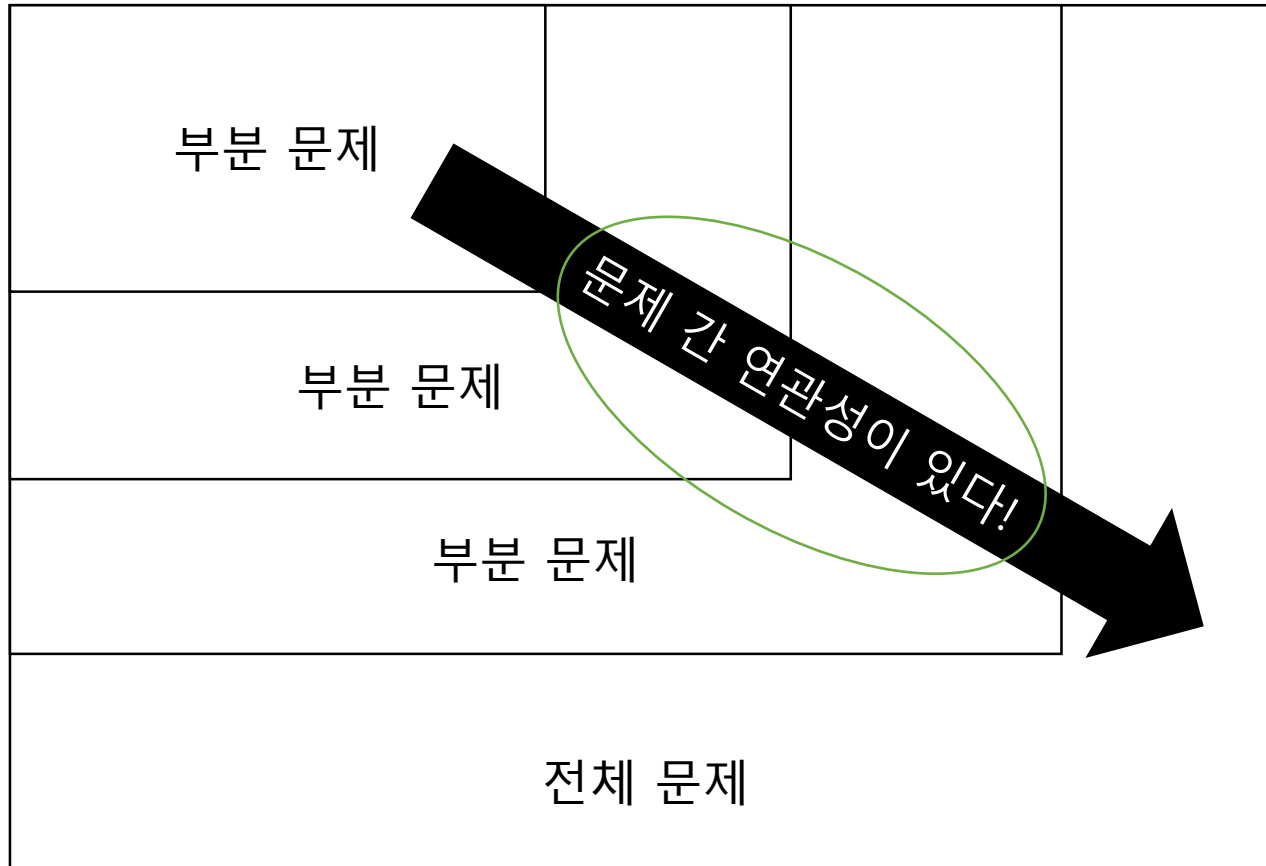
30 mL ($30 \times 120 / 55$ g)

밀도: 1.67g/ml



100 mL

Dynamic Programming?



작은 부분 문제를 풀고, 그것들을
이용해 큰 전체 문제를 해결!

- Bottom Up 방식
- 분할한 문제 간의 **연관성 O**

c.f. Divide & Conquer

- Top Down 방식
- 분할한 문제 간의 **연관성 X**



Dynamic Programming?

1. 부분 문제를 명확하게 정의합니다.

- 원래 문제에서 보통 데이터의 크기만 줄이는 경우로 생각
- 예외적인 케이스도 존재 → 기존 부분 문제 + 제한 조건을 추가

2. 원래 답의 위치를 파악합니다.

- 부분 문제가 곧 답의 힌트!
- 부분 문제의 해답을 모아둘 배열 등을 생성 가능
- 부분 문제가 틀림을 알아내는 데에도 주요한 포인트

사람이 해결해야 함!



3. 재귀 식(점화 식)을 부분 문제의 정의와 수학적 논리에 따라 잘 세웁니다.

- 부분 문제를 해결하기 위해선, 더 작은 부분 문제들의 답을 이용
- Backward Analysis: 이 문제의 답이 어디서 올 수 있었는가를 분석
- 식에 나오게 되는 변수들을 통해 반복 문을 어떻게 해야 할지 분석
- 식이 제대로 세워지지 않는다면, 1번으로 돌아감

4. 기저 조건(초기값)을 세웁니다.

- 문제 조건으로 주어진 초기 값
- 3번 식에서는 값을 구할 수 없는 예외적인 경우

Dynamic Programming?

자연수 N 이 주어지면, $1 \sim N$ 까지의 합을 더하는 프로그램

1. 부분 문제를 명확하게 정의
 $T[i]$: $1 \sim i$ 까지의 합
2. 원래 답의 위치
 $T[N]$ 일 것
→ `print(T[N])`
→ 답이 들어갈 T 배열을 생성
3. 재귀 식(점화 식)
 $T[i] = i + T[i-1]$
→ for문이 i 로 돈다. (N 까지)
4. 기저 조건(초기값)
 i 가 1일 때 $T[i] = 1$
→ for문 내에 조건이 들어간다.
→ 조건이 아닌 경우는 3번

```
N = int(input())
```

03. 피보나치 수열 2

n번째 피보나치 수를 계산합니다.

Dynamic Programming의 핵심은 대부분 펜과 노트에서 완성됩니다.

$$\text{FIB}[1] = 1$$

$$\text{FIB}[2] = 1$$

$$\text{FIB}[3] = 2$$

....

$$\text{FIB}[n] = \text{FIB}[n-1] + \text{FIB}[n-2]$$

03. 피보나치 수열 2

1. 부분 문제를 명확하게 정의
 - $FIB[i]$: i 번째 피보나치 수
2. 원래 답의 위치
 - $FIB[n]$
3. 재귀 식(점화 식)
 - $FIB[i] = FIB[i-1] + FIB[i-2]$
4. 기저 조건(초기값)
 - $i=1$ 일 때, $FIB[i] = 1$
 - $i=2$ 일 때, $FIB[i] = 1$

03. 피보나치 수열 2

```
1 t = int(input())
2
3 for _ in range(t):
4     n = int(input())
5
6
7
8
9
10
11
```

부분문제를 저장할 리스트를 선언하고, 부분 문제를 해결합니다

04. 세계 암기대회

지우가 오른쪽 아래로 가면서 잃을 수 있는 가장 적은 점수를 계산해주세요!

1	0	1	0	1
0	1	1	1	0
0	1	0	1	1

04. 세계 암기대회

각 칸으로 가면서 잃을 수 있는 최저 점수를 저장한다면?

1	0	1	0	1
0	1	1	1	0
0	1	0	1	1



1	1	2	2	3
1	2	2	?	?
1	2	?	?	??

04. 세계 암기대회

$mp(i, j)$: (i,j)까지 가면서 얻을 수 있는 최고 점수
 $p(i, j)$: (i,j)에 있는 점수 (0 또는 1)

$$mp(i, j) = \begin{cases} p(i, j) & \text{if } i = 0, j = 0 \\ mp(i, j - 1) + p(i, j) & \text{elif } i = 0, j > 0 \\ mp(i - 1, j) + p(i, j) & \text{elif } i > 0, j = 0 \\ \min(mp(i - 1, j), mp(i, j - 1), mp(i - 1, j - 1)) + p(i, j) & \text{elif } i > 0, j > 0 \end{cases}$$

ADD01. 부산에서 서울로

부산에서 서울까지 갈 때 최소 횟수로 주유소에서 기름 채우기

*Hint

- 이미 지난 주유소는 다시 갈 필요가 없다. (선형 탐색)
- 서울을 기름 안 넣는 주유소처럼 취급해보자!

ADD01. 부산에서 서울로

Idea: 갈 수 있는 가장 먼 주유소를 가자

유지하는 변수: 지금까지 주유한 횟수, 가장 마지막에 주유한 주유소

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0

7

14

19

29

36

51

71

81

서울: 100



주유 횟수: 0

마지막 주유소: 0

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 0
마지막 주유소: 0

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 0
마지막 주유소: 0

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 0

마지막 주유소: 0

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 1

마지막 주유소: 19

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 1

마지막 주유소: 19

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100

주유 횟수: 2
마지막 주유소: 36

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 3

마지막 주유소: 51

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100



주유 횟수: 4

마지막 주유소: 71

ADD01. 부산에서 서울로

기름 용량: 20



부산: 0 7 14 19 29 36 51 71 81 서울: 100

주유 횟수: 5

마지막 주유소: 81

ADD01. 부산에서 서울로

생각해봐야할 문제

- 부산에서 서울까지 주유소를 한 번도 거치지 않고 갈 수도 있다.
- 마지막 주유소까지 도달했으나, 서울까지 가지 못할 수 있다.

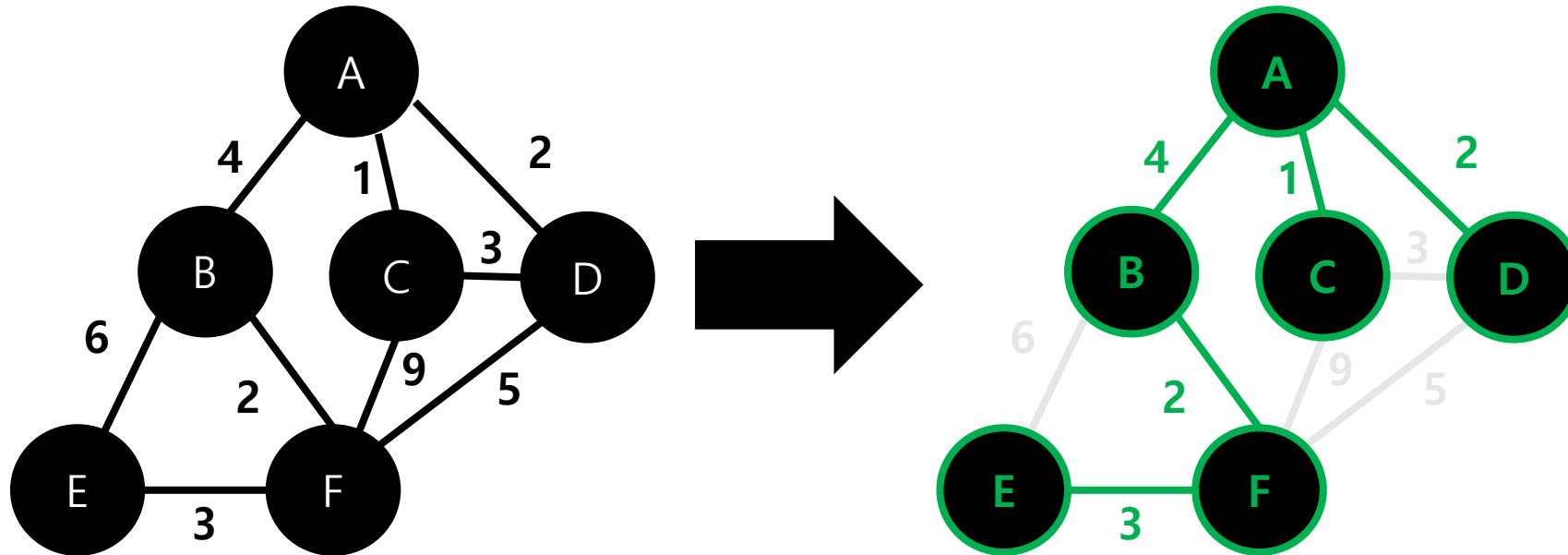
여러 가지 상황을 잘 고려하면서 해결해야 하는 문제!

ADD02. MST (Minimum Spanning Tree)

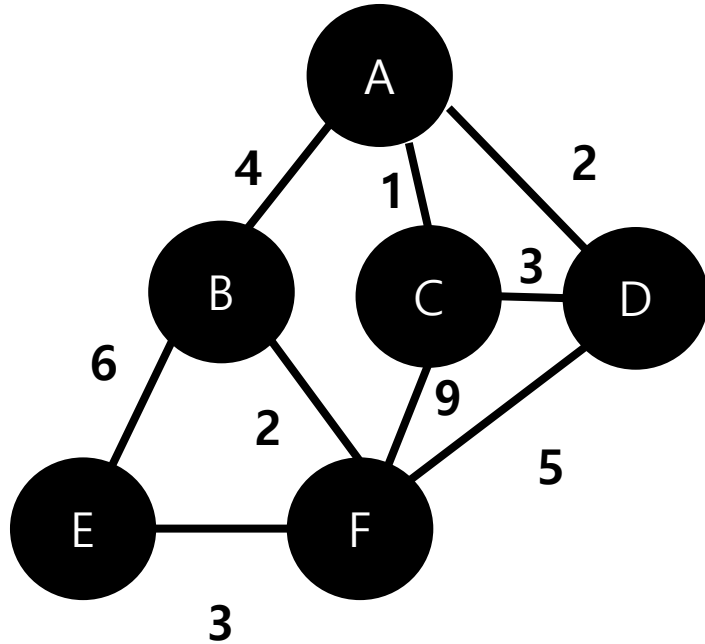
최소 신장 트리(MST)를 구하고, 그 가중치의 합을 출력하는 프로그램을 작성합니다.
Graph + Greedy!

두 가지 방법

- Prim 알고리즘 (Dijkstra 알고리즘과 유사)
- Kruskal 알고리즘 (Union-Find 자료구조)



ADD02. MST



< Prim Algorithm > 의 Idea: 확장

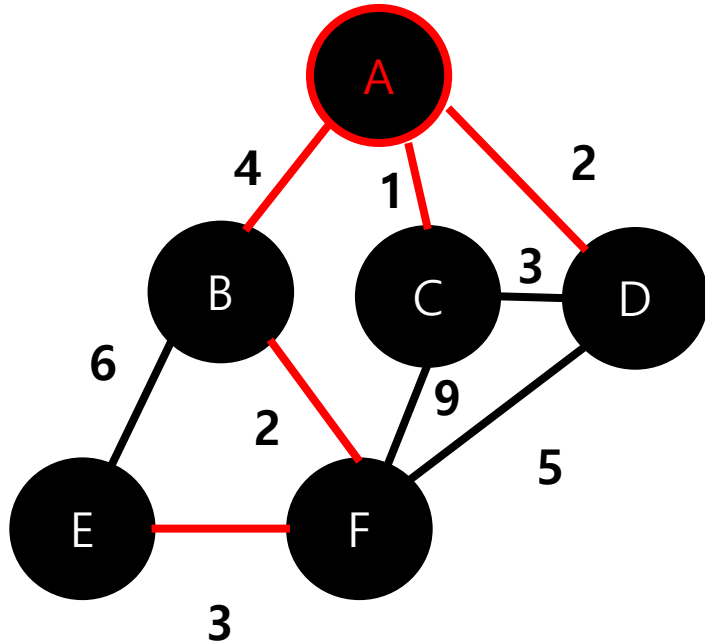
어느 한 정점에서 시작해서 그 정점과 연결된 노드 중에 가장 연결 비용이 작은 것을 연결!

→ 간선의 가중치로 우선순위 큐를 만든다

- 우선순위 큐에서 나온 간선의 필요성 확인
- 이미 연결된 두 정점이라면 이 간선도 필요 없다

가장 작은 걸 연결하다 보면 전체 합도 가장 작을 것!

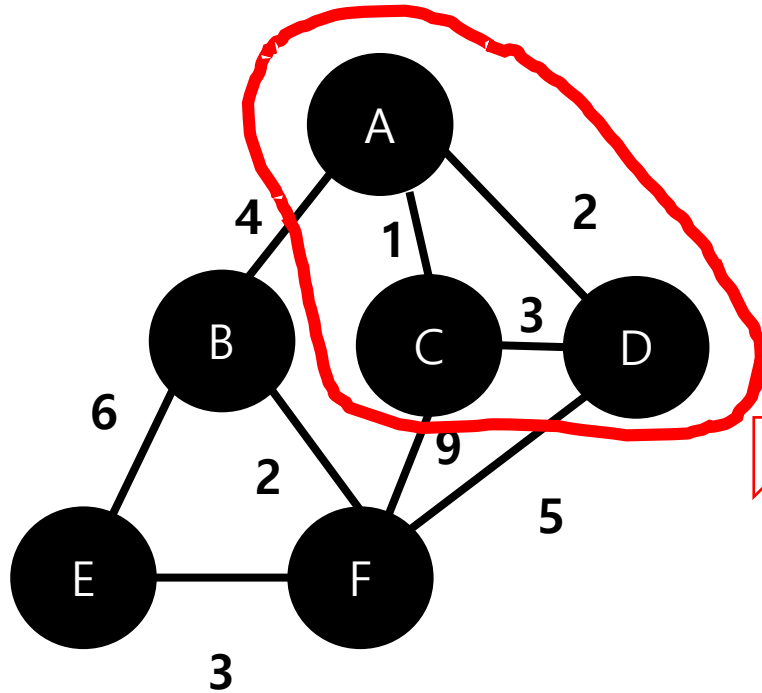
ADD02. MST



A에서 부터 시작!

간선목록	(4,B)	(1,C)	(2,D)				
연결완료	A						
간선목록	(4,B)	(2,D)	(3,D)	(9,F)			
연결완료	A	C					
간선목록	(4,B)	(3,D)	(9,F)	(5,F)			
연결완료	A	C	D				
간선목록	(4,B)	(9,F)	(5,F)				
연결완료	A	C	D				
간선목록	(9,F)	(5,F)	(6,E)	(2,F)			
연결완료	A	C	D	B			
간선목록	(9,F)	(5,F)	(6,E)	(3,E)			
연결완료	A	C	D	B	F		
간선목록	(9,F)	(5,F)	(6,E)				
연결완료	A	C	D	B	F	E	

ADD02. MST



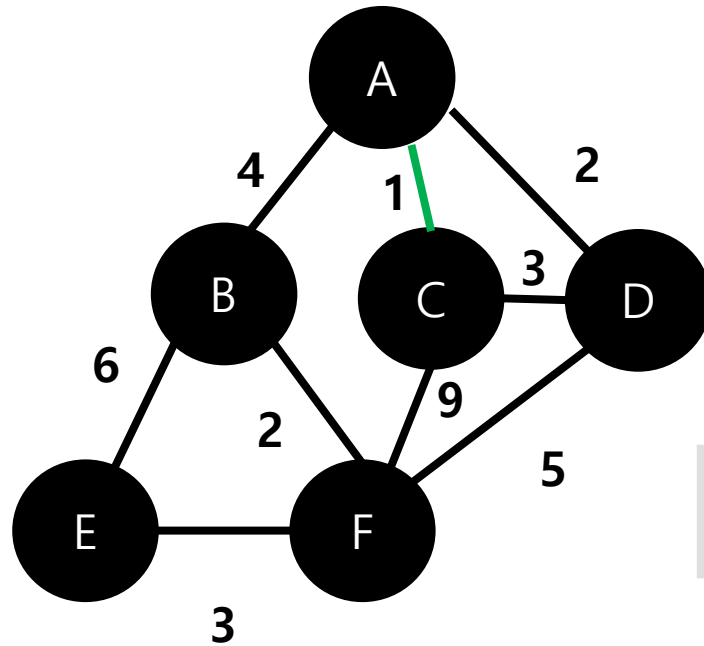
< Kruskal Algorithm > 의 Idea: 최소는 무조건

그래프 상에서 가장 가중치가 작은 간선들을 고른다

- 트리여야 하므로 사이클이 생기는지 확인
- 연결된 트리여야 하므로 모두가 연결되었는지 확인

사이클에서 가중치가 제일 큰 간선은 필요 없다!
알고리즘이 이를 걸러내므로 전체 합이 가장 작을 것!

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	C	D	E	F
rank[v]	0	0	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

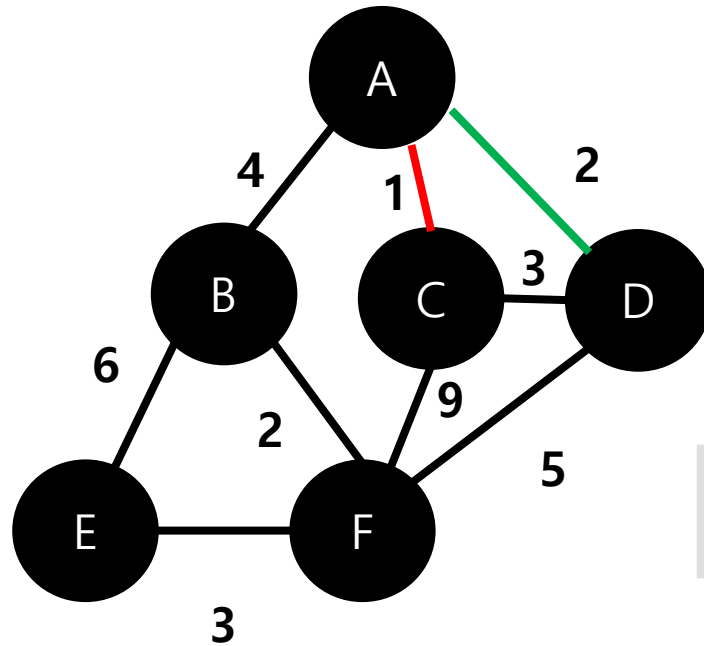
Algorithm *find*(x)

1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	A	D	E	F
rank[v]	1	0	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

Algorithm *find*(x)

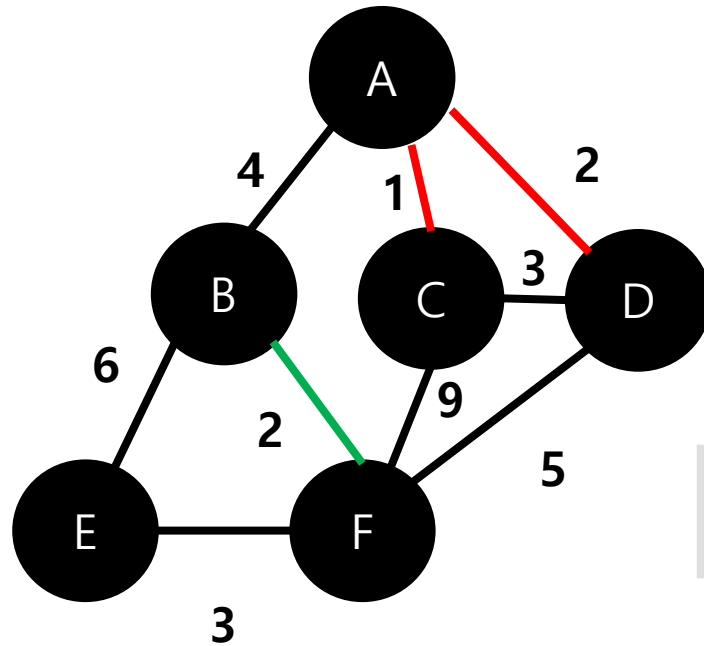
1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$



ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	A	A	E	F
rank[v]	1	0	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

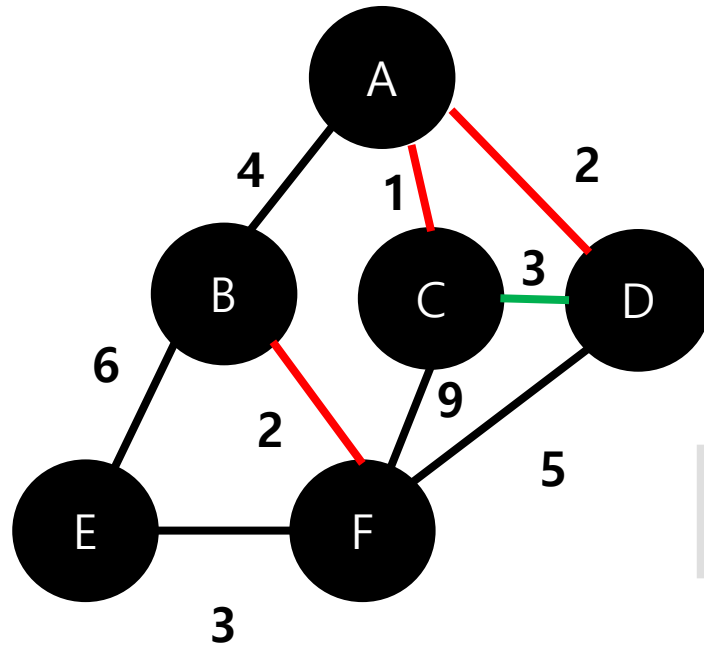
Algorithm *find*(x)

1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	A	A	E	B
rank[v]	1	1	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

Algorithm *find*(x)

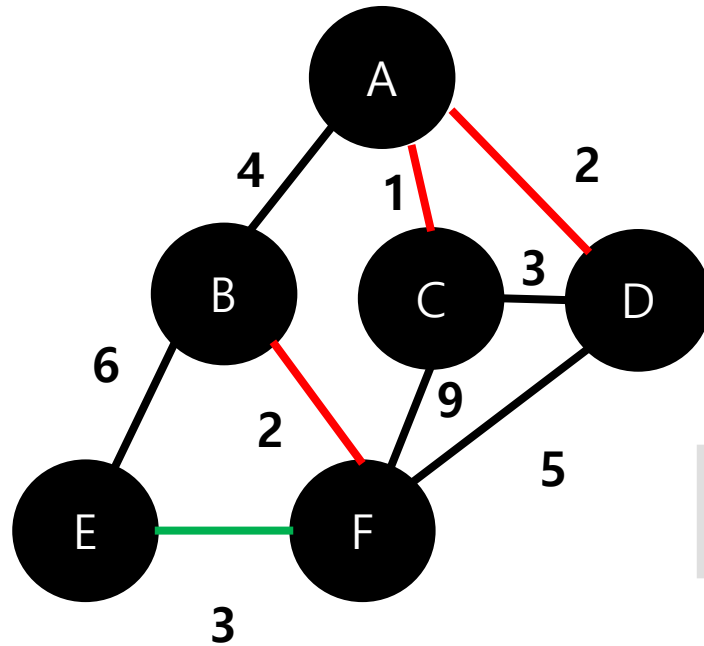
1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$



ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	A	A	E	B
rank[v]	1	1	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

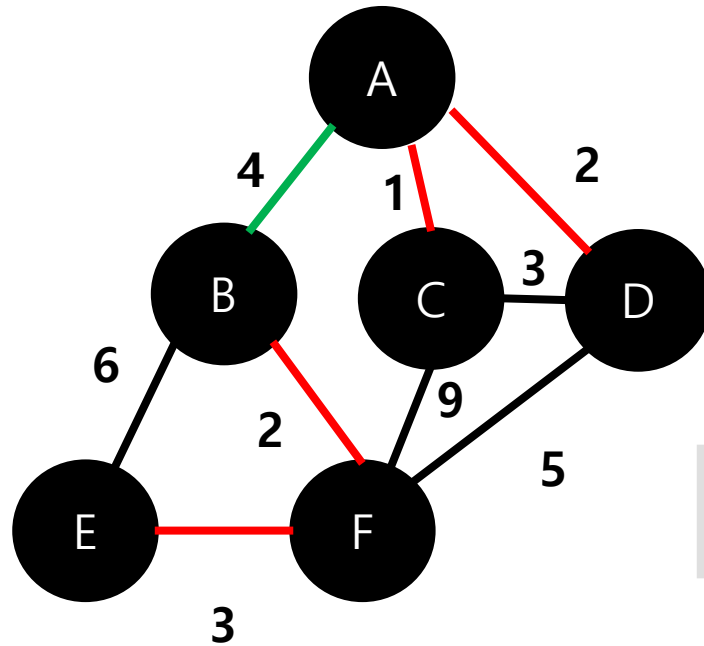
Algorithm *find*(x)

1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	B	A	A	B	B
rank[v]	1	1	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

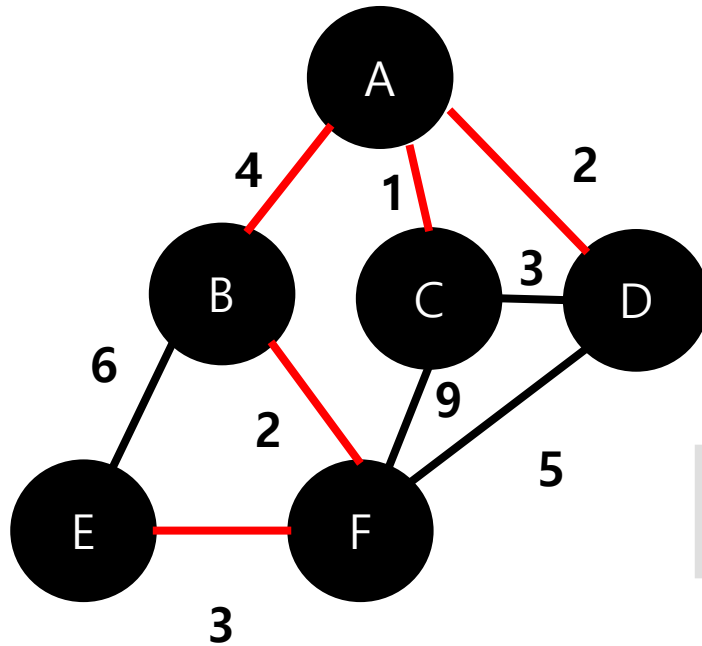
Algorithm *find*(x)

1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	A	A	A	B	B
rank[v]	2	1	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

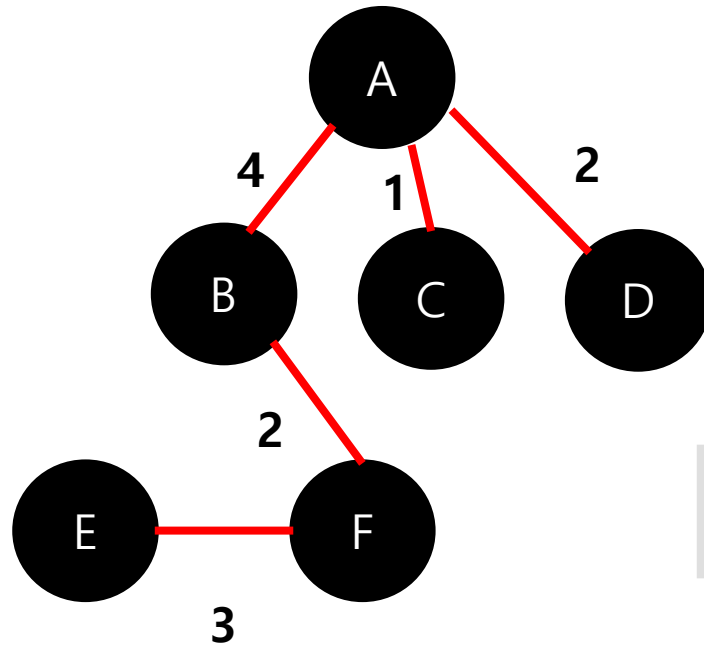
Algorithm *find*(x)

1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$

ADD02. MST



가장 작은 간선부터 연결해보자!

v	A	B	C	D	E	F
root[v]	A	A	A	A	B	B
rank[v]	2	1	0	0	0	0

Algorithm *makeset*(x)

1. $\pi(x) \leftarrow x$
2. $\text{rank}(x) \leftarrow 0$

Algorithm *find*(x)

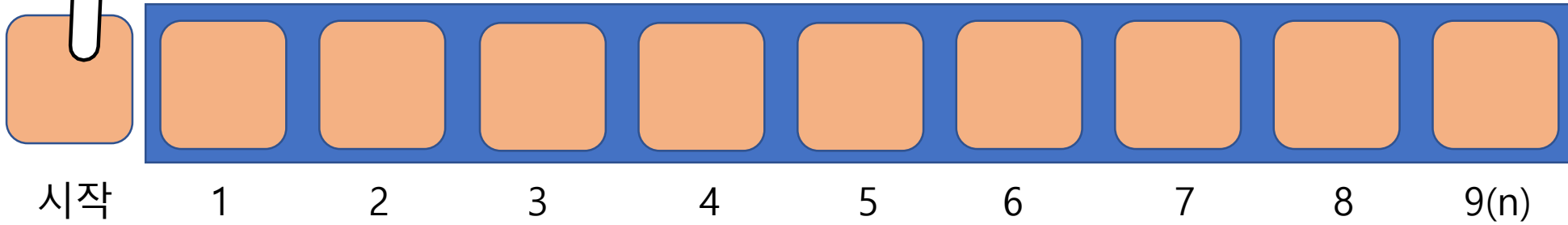
1. **while** $x \neq \pi(x)$
2. **do** $x \leftarrow \pi(x)$
3. **return** x

Algorithm *union*(x, y)

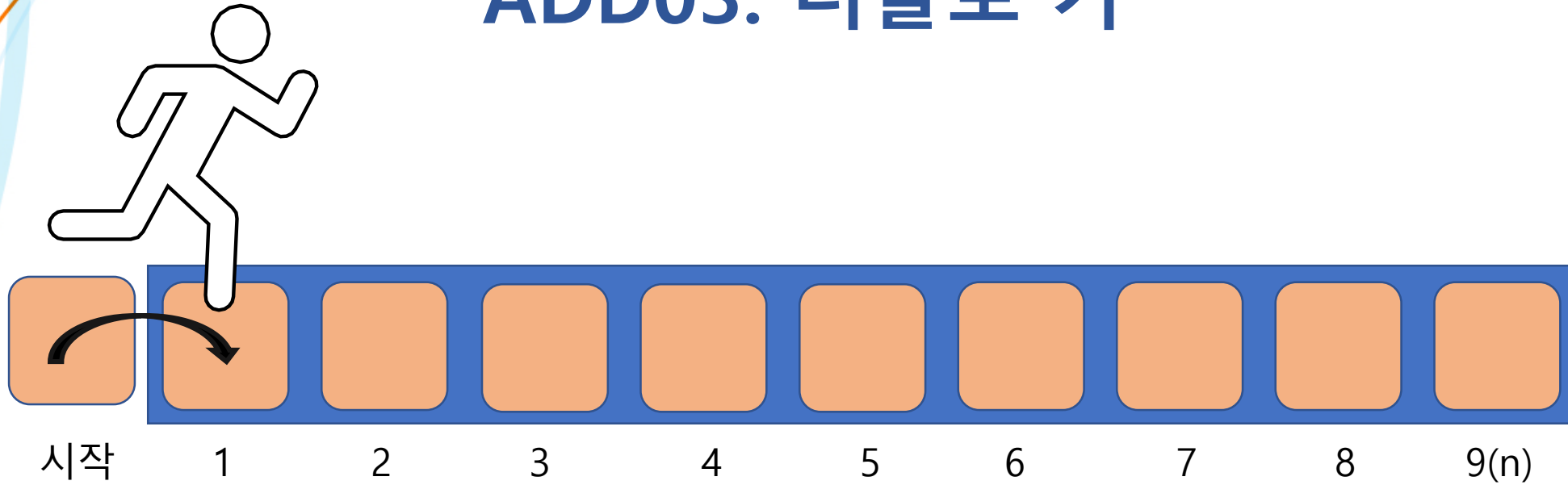
1. $r_x \leftarrow \text{find}(x)$
2. $r_y \leftarrow \text{find}(y)$
3. **if** $r_x = r_y$
4. **then return**
5. **if** $\text{rank}(r_x) > \text{rank}(r_y)$
6. **then** $\pi(r_y) \leftarrow r_x$
7. **else** $\pi(r_x) \leftarrow r_y$
8. **if** $\text{rank}(r_x) = \text{rank}(r_y)$
9. **then** $\text{rank}(r_y) \leftarrow \text{rank}(r_y) + 1$



ADD03. 더블로 가

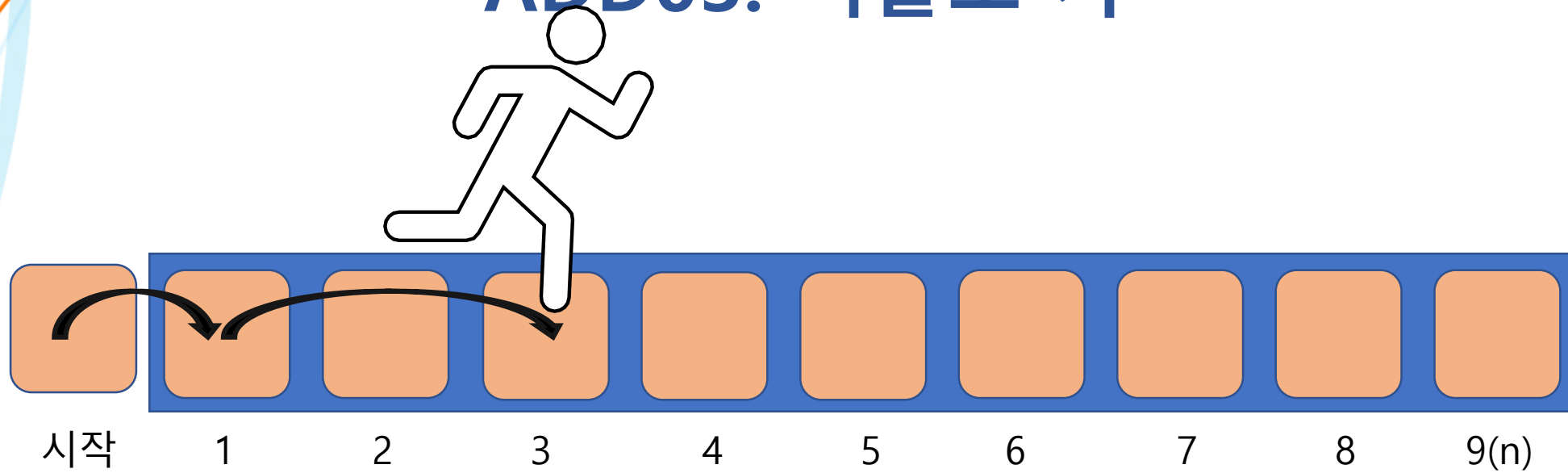


ADD03. 더블로 가



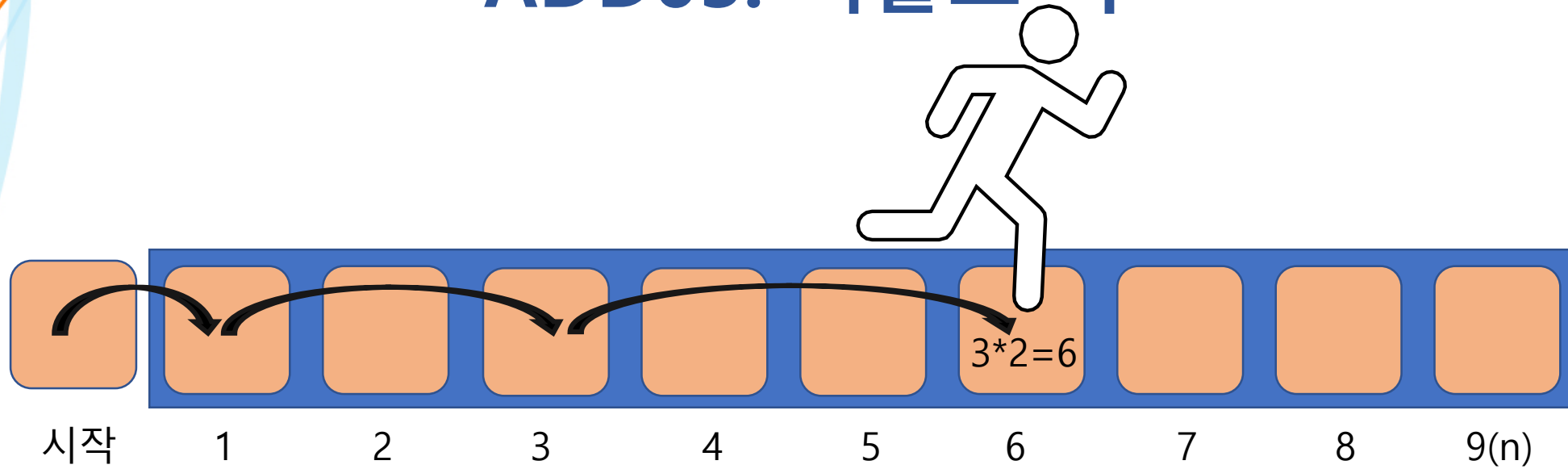
1. 한 칸만 이동하는 방법

ADD03. 더블로 가



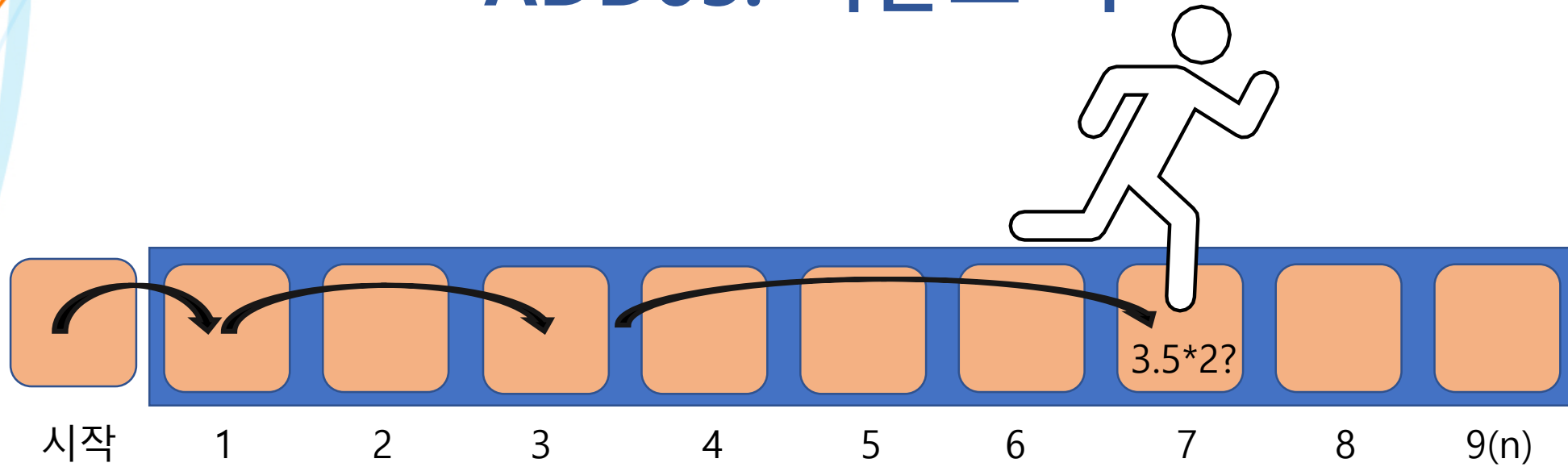
1. 한 칸만 이동하는 방법
2. 두 칸을 이동하는 방법

ADD03. 더블로 가



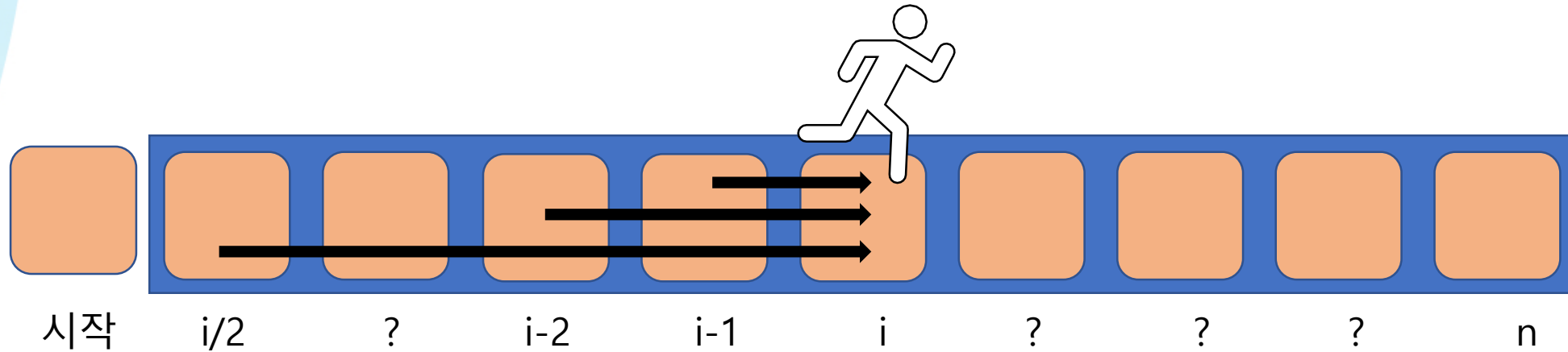
1. 한 칸만 이동하는 방법
2. 두 칸을 이동하는 방법
3. 더블로 이동하는 방법

ADD03. 더블로 가



1. 한 칸만 이동하는 방법
2. 두 칸을 이동하는 방법
3. ~~더블로 이동하는 방법~~

ADD03. 더블로 가



현재 어떤 발판 i 에 있을 때, **최대점수**

- 한칸 전 발판에서 온 경우
- 두칸 전 발판에서 온 경우
- 더블 이동해서 온 경우 (?)

→ {(각 경우) + (현재 발판의 점수)}의 최대점수

ADD03. 더블로 가

1. 부분 문제를 명확하게 정의

$T[i]$: i 번 발판을 밟았을 때의 최대 점수

2. 원래 답의 위치

$T[N]$ 일 것

→ `print(T[N])`

→ 답이 들어갈 T 배열을 생성

3. 재귀 식(점화 식)

(i 가 홀수면) $T[i] = \max(T[i-1], T[i-2]) + \text{data}[i]$

(i 가 짝수면) $T[i] = \max(T[i-1], T[i-2], T[i//2]) + \text{data}[i]$

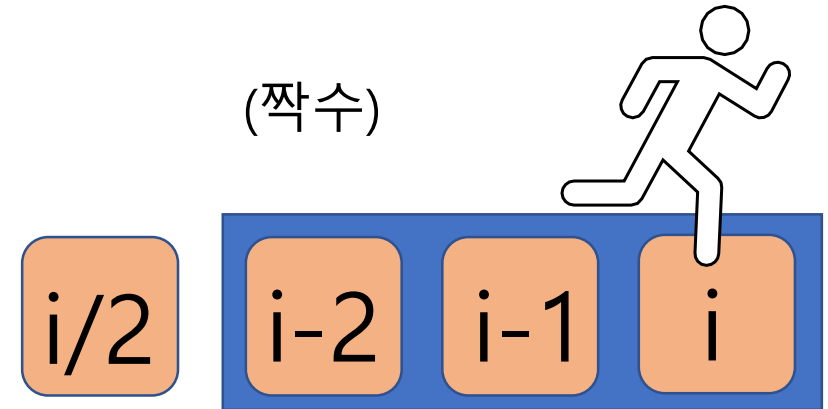
→ for문이 i 로 돈다. (N 까지)

4. 기저 조건(초기값)

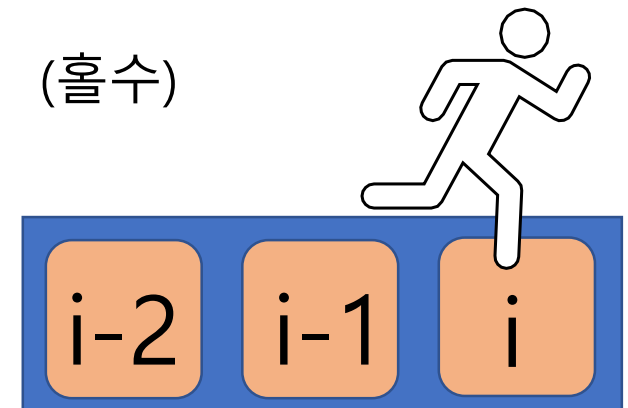
i 가 1일 때 $T[i] = \text{data}[i]$

→ for문 내에 조건이 들어간다.

(짝수)



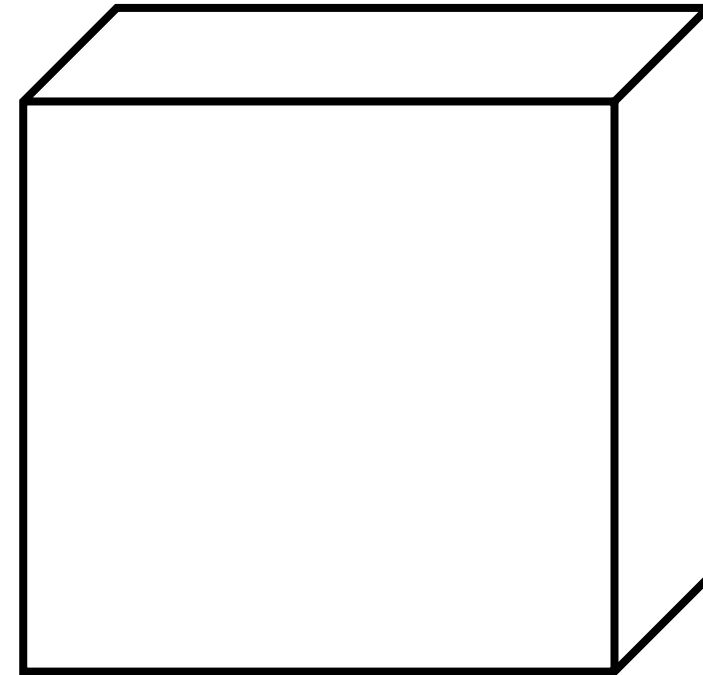
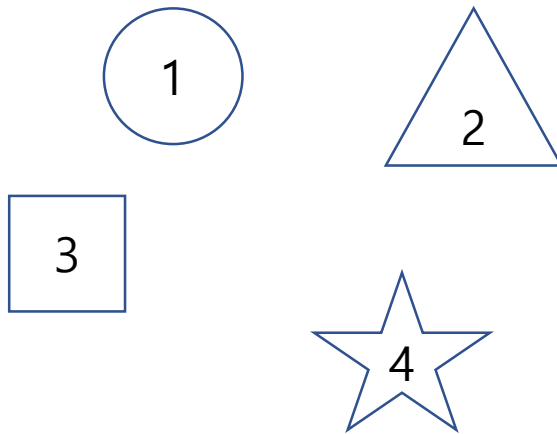
(홀수)



ADD04. 배송비 절약

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

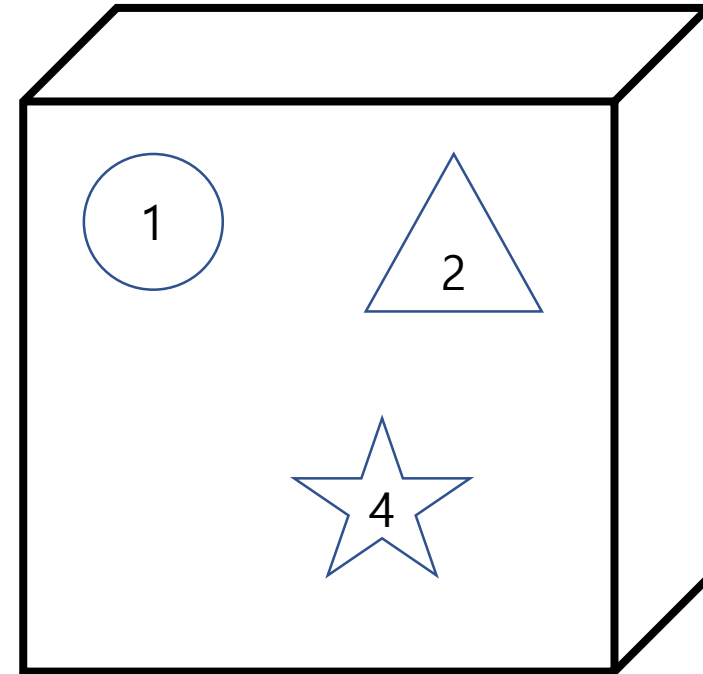
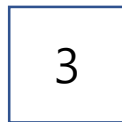
무게: 0 (MAX 10)
가치: 0



ADD04. 배송비 절약

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

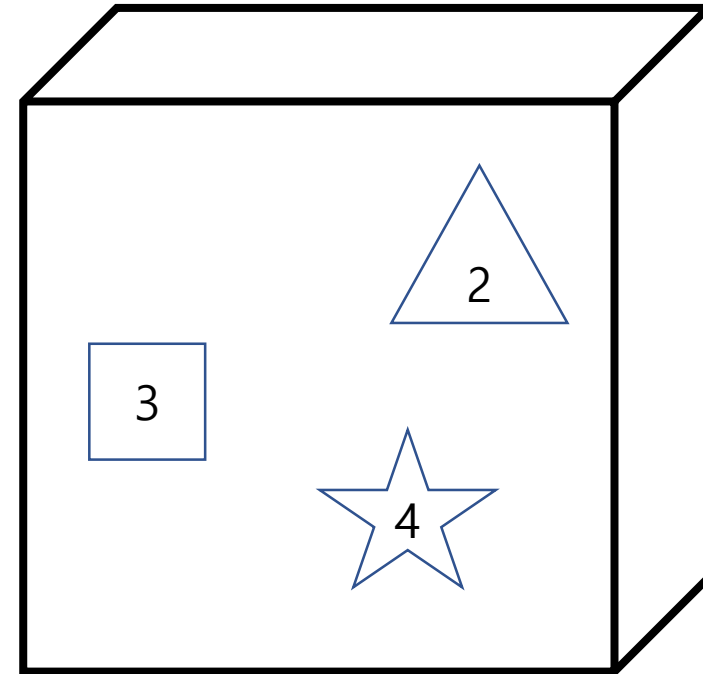
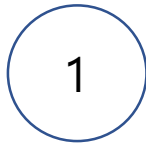
무게: $6+1+3=10$
가치: $10+30+200=240$



ADD04. 배송비 절약

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

무게: $1+5+3=9$ (MAX 10)
가치: $30+100+200=330$



ADD04. 배송비 절약

아이디어

$D[i][j]$ = i 번째까지의 물건을 가지고 무게 j 를 채울 때 가치 합의 최대값

$i \setminus j$	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0										
3	0										
4	0										??

ADD04. 배송비 절약

아이디어

$D[i][j]$ = i 번째까지의 물건을 가지고 무게를 j 이하로 채울 때 가치 합의 최대값

첫번째 물건을 가지고 무게 0부터 C 까지 만들 때 가치의 최대값을 저장.

$i-1$ 번째까지의 물건을 가지고 j 이하의 무게를 만드는 최대 가치에 i 번째 물건이 추가된 경우.

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30									
3	0										
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30								
3	0										
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200



ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30	30	30	30	30				
3	0										
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30	30	30	30	30	40			
3	0										
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200



ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30	30	30	30	30	40	40	40	40
3	0	30	30	30	30	100					
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200

ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30	30	30	30	30	40	40	40	40
3	0	30	30	30	30	100	130				
4	0										??

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200



ADD04. 배송비 절약

$$D[i][j] = \begin{cases} D[i-1][j] & \text{if } w_i > j \\ \max(D[i-1][j], D[i-1][j-w_i]+v_i) & \text{if } w_i \leq j \end{cases}$$

i \ j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	10	10	10	10	10
2	0	30	30	30	30	30	30	40	40	40	40
3	0	30	30	30	30	100	130	130	130	130	130
4	0	30	30	200	230	230	230	230	300	330	330

	1	2	3	4
무게	6	1	5	3
가치	10	30	100	200