Chapter. 02

알고리즘

동적 프로그래밍
Dynamic Programming

FAST CAMPUS
ONLINE
알고리즘 공채 대비반!

강사. 류호석

Life Changing Education

Chapter. 02

알고리즘
동적 프로그래밍(Dynamic Programming)

FAST CAMPUS
ONLINE
류호석 강사.

I 동적 프로그래밍(Dynamic Programming)이란?

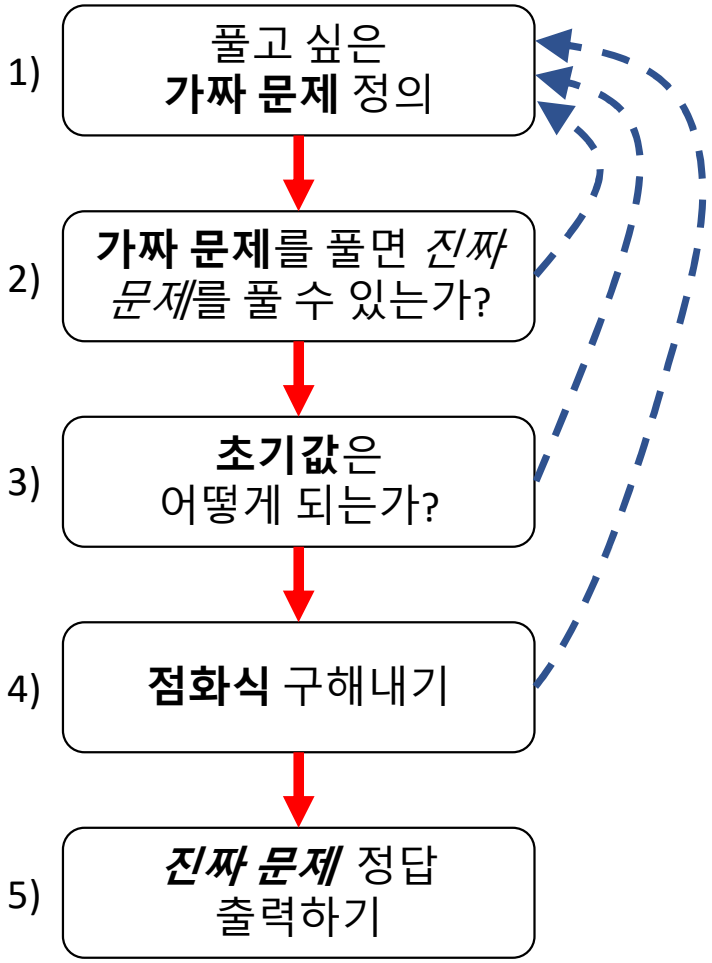
Dynamic := *동적인, 변화하는*

Programming := *문제를 해결하는*

문제의 크기를 변화하면서 정답을 계산하는데,

작은 문제의 결과를 *이용*해서 **큰 문제**의 정답을 빠르게 계산하는 알고리즘

I 동적 프로그래밍(Dynamic Programming)이란?



BOJ 2579 – 계단 오르기

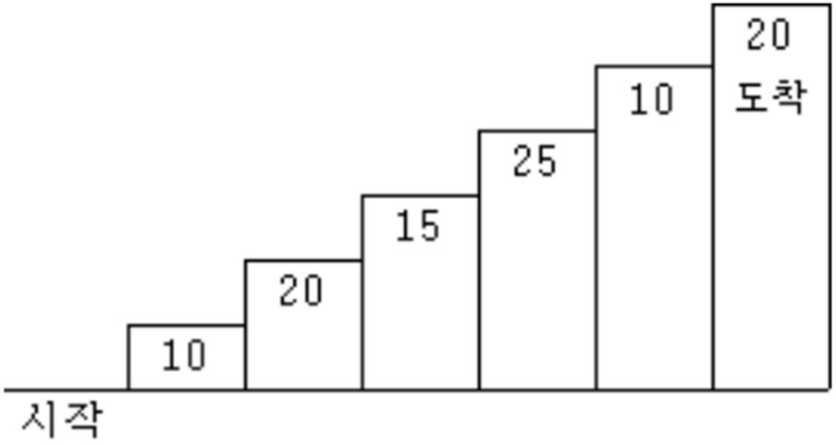
난이도: 2

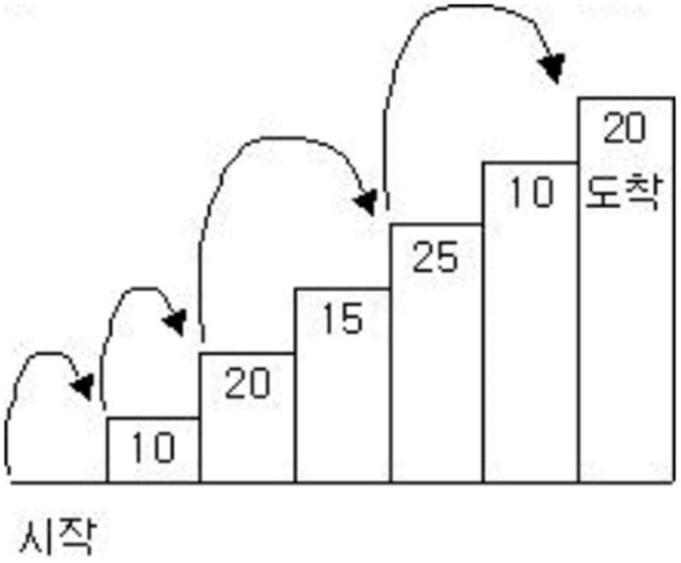
$1 \leq$ 계단의 개수, $N \leq 300$

1.계단은 한 번에 한 계단씩 또는 두 계단씩 오를 수 있다. 즉, 한 계단을 밟으면서 이어서 다음 계단이나, 다음 다음 계단으로 오를 수 있다.

2.연속된 세 개의 계단을 모두 밟아서는 안 된다. 단, 시작점은 계단에 포함되지 않는다.

3.마지막 도착 계단은 반드시 밟아야 한다.





I 정답의 최대치

N 개의 계단을 모두 밟는다고 치더라도

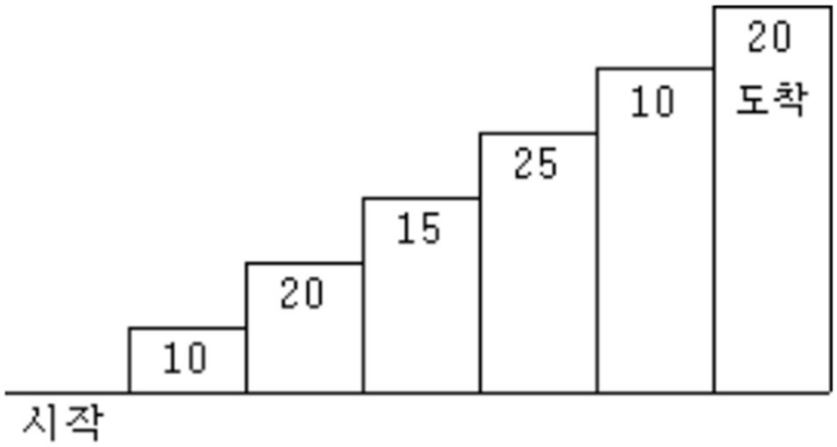
$$N * \text{최대 점수} = 300 * 10{,}000 = 300\text{만}$$

이기 때문에, 정답은 Integer 범위 내에 들어온다.

I 완전 탐색 접근

완전 탐색 접근을 통해서 모든 경우를 직접 하나하나 찾아내 보자.

본 문제에서 “경우”란, 조건을 만족하게 계단을 올라 도착까지 가는 방법을 의미한다.



I Dynamic Programming 접근

1) 풀고 싶은 **가짜 문제** 정의 – **일단 도전해보기**

진짜 문제 := N번째 계단에 도착하며 얻는 최대 점수

가짜 문제 := Dy[i] = i번째 계단에 도착하며 얻는 최대 점수

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i]							

(레벨 1: 진짜 문제랑 똑같은 가짜 문제인 경우)

I Dynamic Programming 접근

2) **가짜 문제**를 풀면 *진짜 문제*를 풀 수 있는가?

Dy 배열을 가득 채울 수만 있다면? *진짜 문제*에 대한 대답은 Dy[N] 이다.

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i]							

I Dynamic Programming 접근

3) **초기값**은 어떻게 되는가?

초기값: 쪼개지 않아도 풀 수 있는 “작은 문제”들에 대한 정답

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i]	0						

I Dynamic Programming 접근

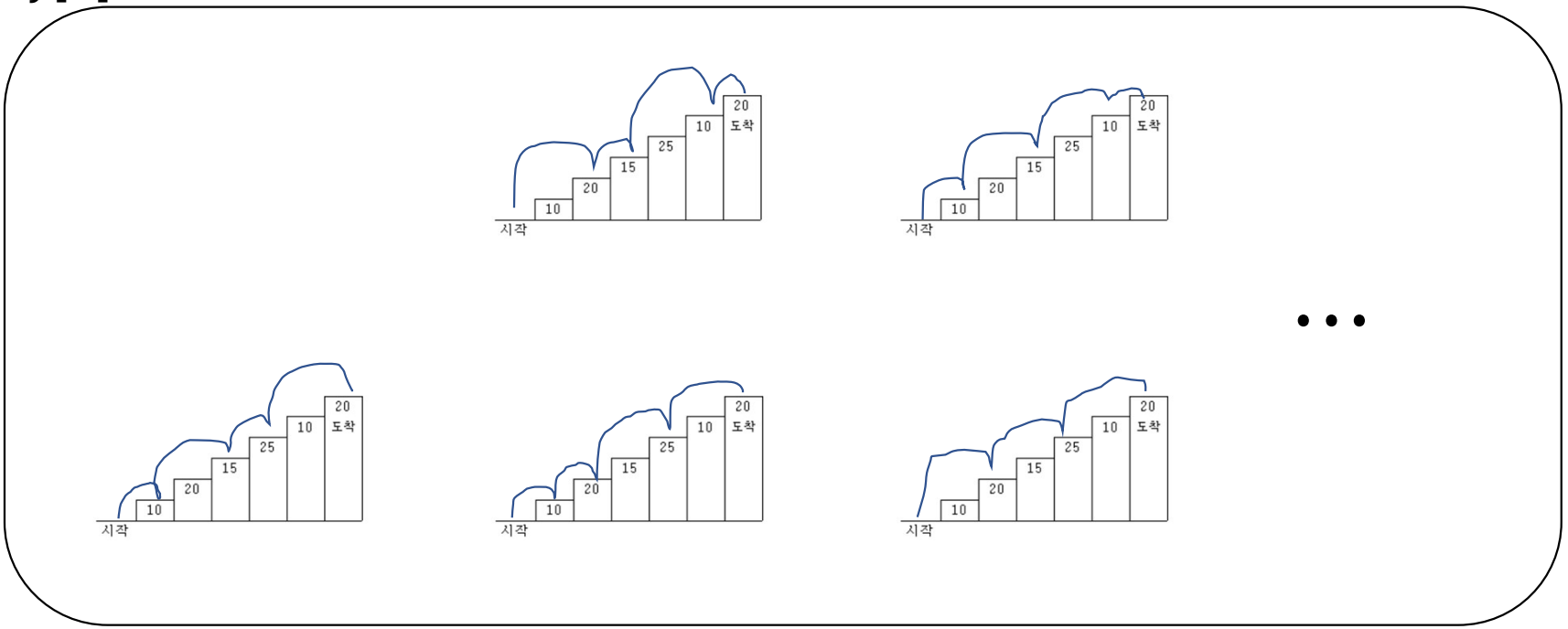
4) **점화식** 구해내기

1. Dy[i] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)
2. 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

I Dynamic Programming 접근

4-1) Dy[i] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)

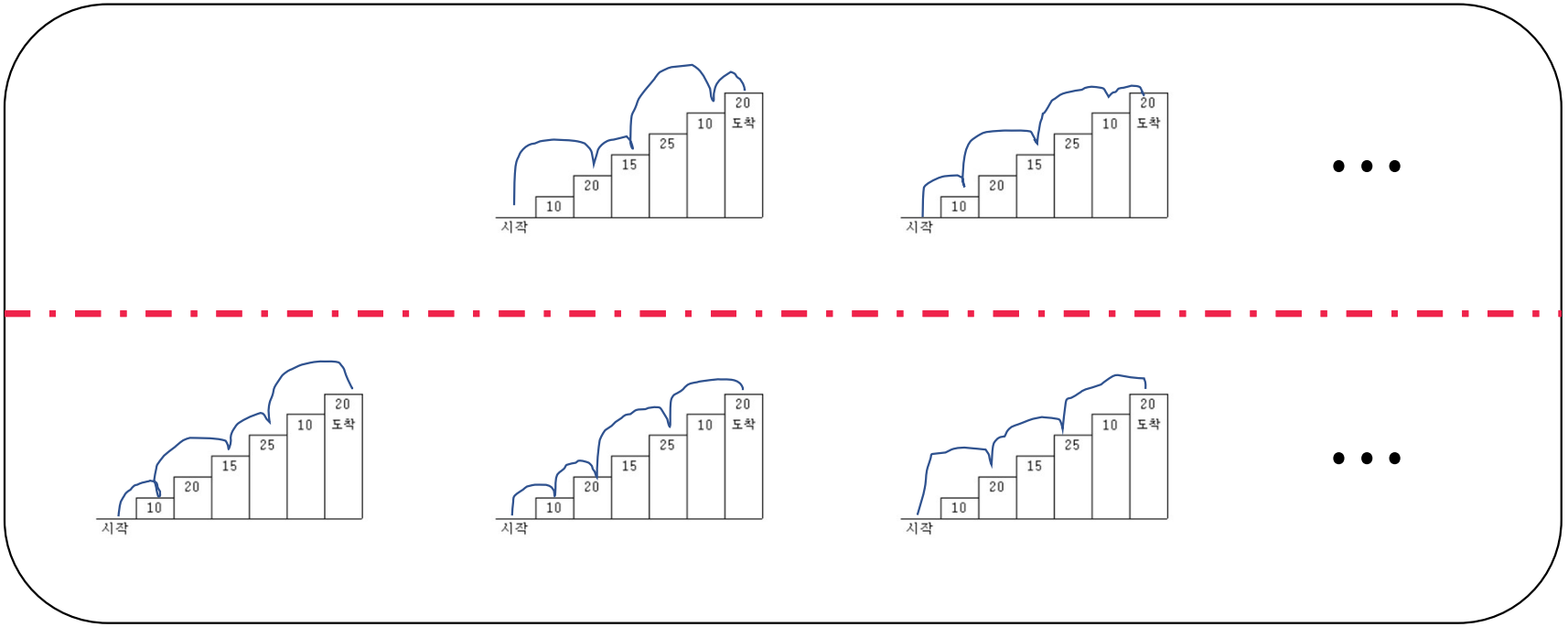
Dy[6] 계산에 필요한 탐색 경우들



I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

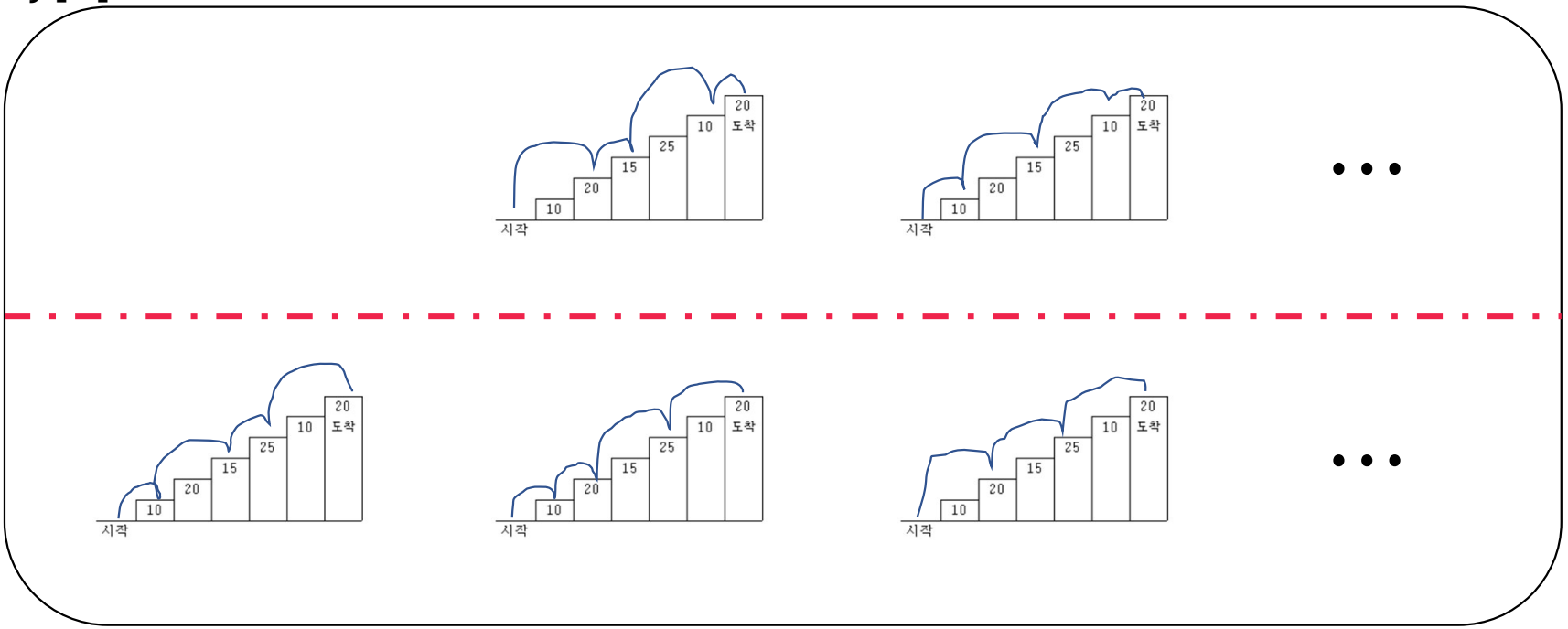
최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



I Dynamic Programming 접근

4-1) Dy[i] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)

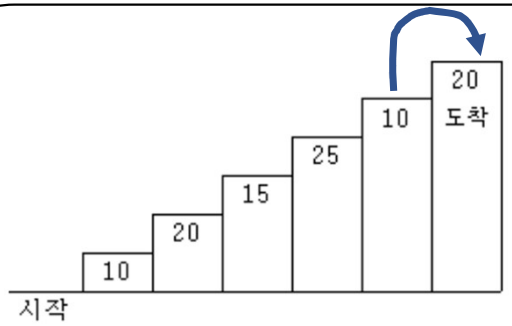
Dy[6] 계산에 필요한 탐색 경우들



I Dynamic Programming 접근

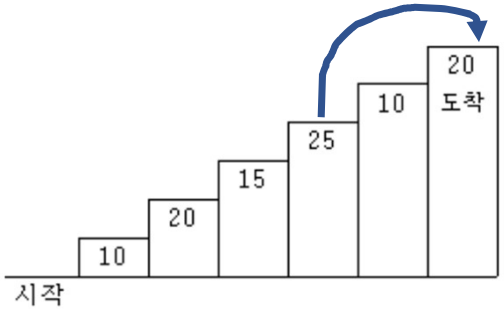
4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



Dy[i] = (i - 1) 번째 계단 최대 점수 + A[i]

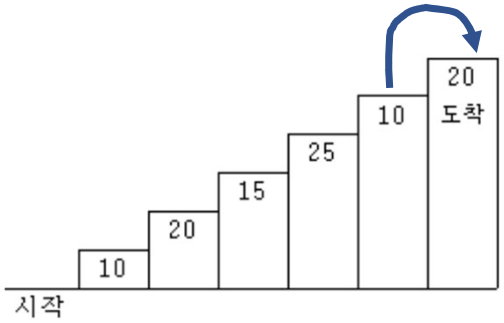
= ??????? + A[i]



Dy[i] = (i - 2) 번째 계단 최대 점수 + A[i]

I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기



(i - 1) 번째 계단 최대 점수 + A[i]

= ??????? + A[i]

모자란 정보 := (i - 1)번째 계단 직전에 (i - 2)번째 계단도 밟았었는가?

I Dynamic Programming 접근

1) 풀고 싶은 **가짜 문제** 정의 – **필요한 정보를 문제에 추가하기**

진짜 문제 := N번째 계단에 도착하며 얻는 최대 점수

가짜 문제 ➔

Dy[i][0] := i-1번째 계단은 **밟지 않고**, i번째 계단에 도착하며 얻는 최대 점수

Dy[i][1] := i-1번째 계단을 **밟고서**, i번째 계단에 도착하며 얻는 최대 점수

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]							
Dy[i][1]							

(레벨 2: 문제 조건에 맞게 상태를 추가한 경우)

I Dynamic Programming 접근

2) **가짜 문제**를 풀면 *진짜 문제*를 풀 수 있는가?

Dy 배열을 가득 채울 수만 있다면? *진짜 문제*에 대한 대답은 `max(Dy[N][0], Dy[N][1])` 이다.

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]							
Dy[i][1]							

I Dynamic Programming 접근

3) **초기값**은 어떻게 되는가?

초기값: 쪼개지 않아도 풀 수 있는 “작은 문제”들에 대한 정답

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20				
Dy[i][1]	-	-	30				

I Dynamic Programming 접근

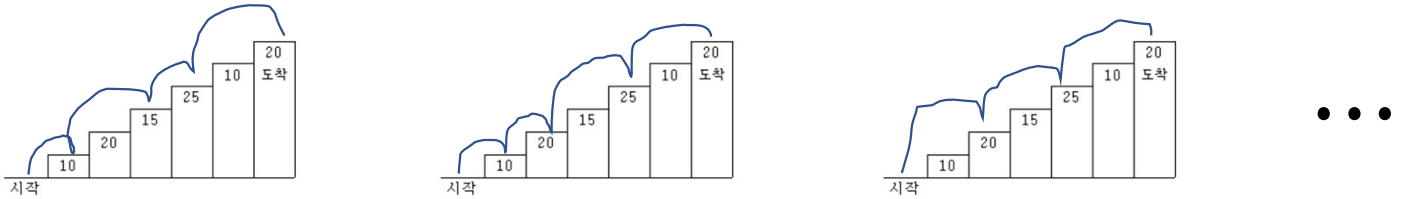
4) **점화식** 구해내기

1. Dy[i][0], Dy[i][1] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)
2. 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

I Dynamic Programming 접근

4-1) Dy[i][0] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기

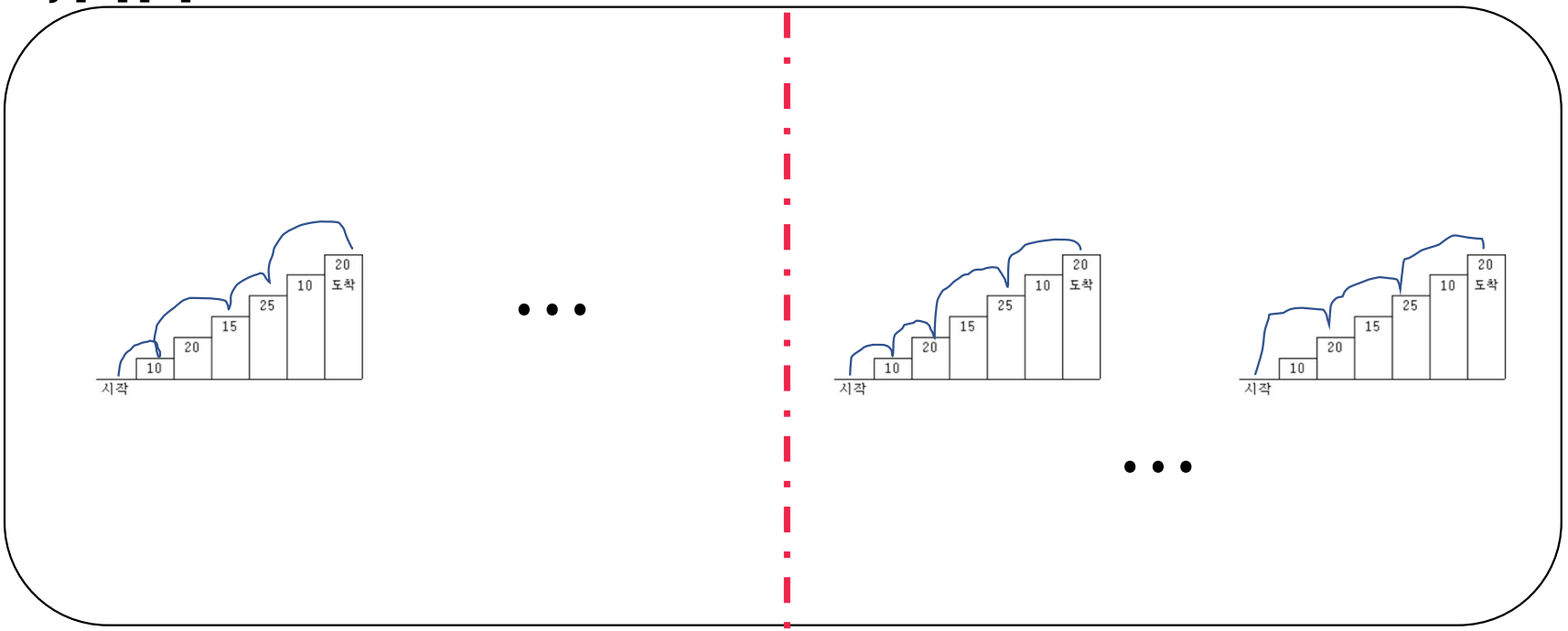
Dy[6][0] 계산에 필요한 탐색 경우들



I Dynamic Programming 접근

4-1) Dy[i][0] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기

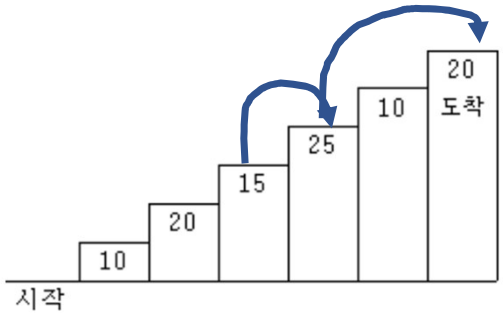
Dy[6][0] 계산에 필요한 탐색 경우들



I Dynamic Programming 접근

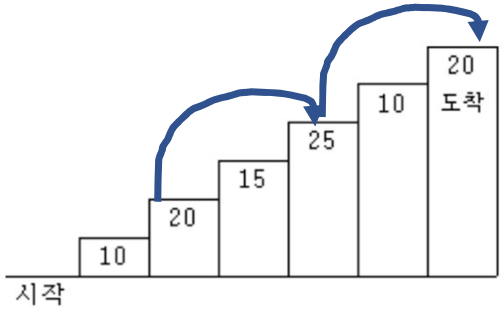
4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



i-2, i-3 두 계단 모두 밟은 경우

➔ Dy[i-2][1] + A[i]



i-2은 밟고 i-3 은 밟지 않은 경우

➔ Dy[i-2][0] + A[i]

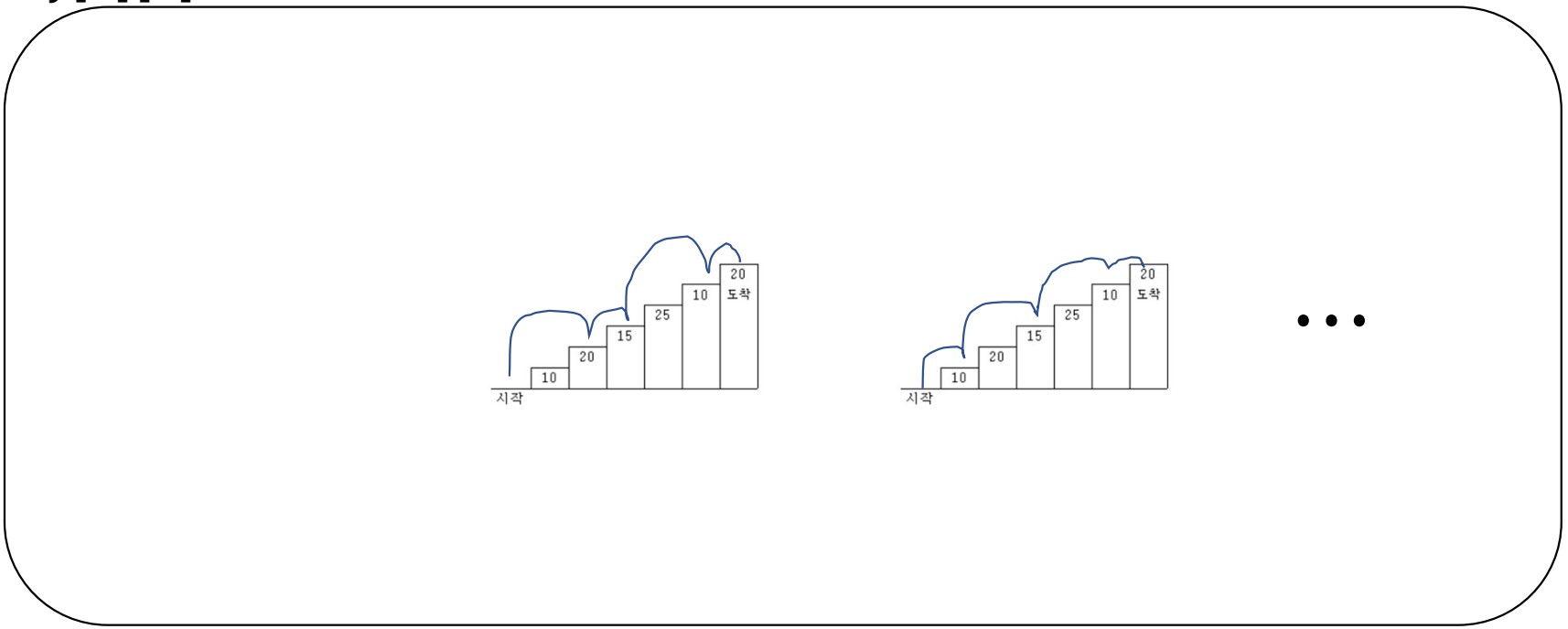
Dy[i][0] =

$$\max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}$$

I Dynamic Programming 접근

4-1) Dy[i][1] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기

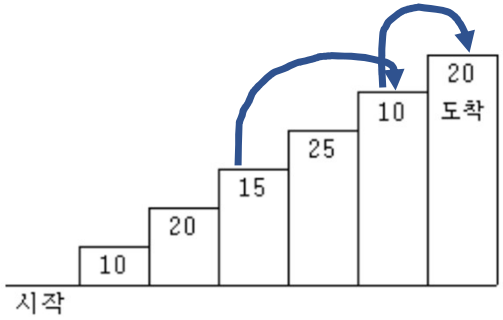
Dy[6][1] 계산에 필요한 탐색 경우들



I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



i-1은 밟고 i-2는 안 밟은 경우

➔ Dy[i-1][0] + A[i]

Dy[i][1] =

Dy[i-1][0] + A[i]

I Dynamic Programming 접근

4) **점화식** 구해내기

$$Dy[i][0] = \max \begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}$$

$$Dy[i][1] = Dy[i-1][0] + A[i]$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20				
Dy[i][1]	-	-	30				

I Dynamic Programming 접근

4) **점화식** 구해내기

$$Dy[i][0] = \max \begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}$$

```
Dy[i][1] = Dy[i-1][0] + A[i]
```

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	75
Dy[i][1]	-	-	30	35	50	65	65

I 시간, 공간 복잡도 계산하기

완전 탐색을 통해 모든 경우를 세면 정답의 개수만큼
의 시간이 걸리지만, Dy 배열을 1번지부터 N번지 까지
채우는 것은 O(N) 이라는 시간 복잡도면 충분하다!

I 구현

```
static void pro() {
    // 초기값 구하기
    /* TODO */

    // 점화식을 토대로 Dy 배열 채우기
    /* TODO */

    // Dy배열로 정답 계산하기
    int ans = /* TODO */;

    System.out.println(ans);
}
```

I 연습 문제

- BOJ 1149 – RGB 거리
- BOJ 2156 – 포도주 시식
- BOJ 2193 – 이친수
- BOJ 9465 – 스티커
- BOJ 1309 – 동물원
- BOJ 2688 – 줄어들지 않아

이외의 추천 문제가 추가되면 Github 자료에 코드 업로드

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

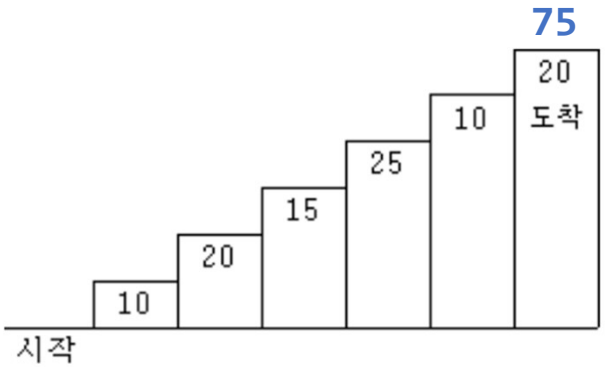
난이도: 3

FAST CAMPUS
ONLINE
류호석 강사.

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

난이도: 3

6) 최댓값이 75인건 알겠는데, **실제로 어떻게** 이동해야 할까?



i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	75
Dy[i][1]	-	-	30	35	50	65	65

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}, \quad \texttt{Come[i][0]} = \begin{cases} (i-2, 1) \\ (i-2, 0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][0]} = (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20				
Dy[i][1]	-	-	30				

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1]+A[i] \\ Dy[i-2][0]+A[i] \end{cases}, \quad \texttt{Come[i][0]}= \begin{cases} (i-2,1) \\ (i-2,0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]}= (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20				
Dy[i][1]	-	-	30				

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}, \quad \texttt{Come[i][0]} = \begin{cases} (i-2, 1) \\ (i-2, 0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]} = (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25			
Dy[i][1]	-	-	30	35			

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}, \quad \texttt{Come[i][0]} = \begin{cases} (i-2, 1) \\ (i-2, 0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]} = (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55		
Dy[i][1]	-	-	30	35	50		

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1]+A[i] \\ Dy[i-2][0]+A[i] \end{cases}, \quad \texttt{Come[i][0]}= \begin{cases} (i-2,1) \\ (i-2,0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]}= (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	
Dy[i][1]	-	-	30	35	50	65	

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1]+A[i] \\ Dy[i-2][0]+A[i] \end{cases}, \quad \texttt{Come[i][0]}= \begin{cases} (i-2,1) \\ (i-2,0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]}= (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	75
Dy[i][1]	-	-	30	35	50	65	65

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}, \quad \texttt{Come[i][0]} = \begin{cases} (i-2, 1) \\ (i-2, 0) \end{cases}$$

$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]} = (i-1,\ 0)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	75
Dy[i][1]	-	-	30	35	50	65	65

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

6) **어떤 파티션에서 왔는가?** 를 기록해주자

$$\texttt{Dy[i][0]} = \max\begin{cases} Dy[i-2][1] + A[i] \\ Dy[i-2][0] + A[i] \end{cases}, \quad \texttt{Come[i][0]} = \begin{cases} (i-2, 1) \\ (i-2, 0) \end{cases}$$

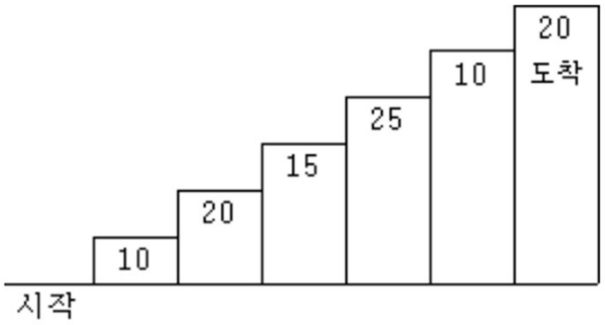
$$\texttt{Dy[i][1]} = \texttt{Dy[i-1][0]} + \texttt{A[i]}, \quad \texttt{Come[i][1]} = (i, -10)$$

i	0	1	2	3	4	5	6
A[i]	0	10	20	15	25	10	20
Dy[i][0]	-	10	20	25	55	45	75
Dy[i][1]	-	-	30	35	50	65	65

I 계단 오르기 – 심화 탐구(역추적, Backtrack)

난이도: 3

6) 최댓값이 75인건 알겠는데, **실제로 어떻게** 이동해야 할까?



Table을 채워 나갈 때에 기록을 함께 한다면 “실제 방법”도 찾을 수 있다.

이를 역추적, 혹은 Backtrack이라고 한다.

I BOJ 11057 – 오르막 수

난이도: 3

$1 \leq$ 수의 길이, $N \leq 1{,}000$

오르막 수란, 각 자리가 오름차순을 이루는 수를 말한다.

길이가 N인 수 중에서 오르막 수의 개수를 $10{,}007$로 나눈 나머지를 구하자.

단, 수는 0으로 시작할 수 있다.

예)

길이 1 := 0, 1, 2, 3, …, 9 ➔ 총 10개

길이 2 := 00, 01, …, 09, 11, 12, …, 99 ➔ 총 55개

길이 N := 몇 개?

I 정답의 최대치

정답을 10,007 로 나눈 나머지로 출력해야 한다.

즉, 문제를 푸는 과정에서 계속 나눈 나머지만 가지고 있다면 Integer 범위로도 충분할 것이다.

I 완전 탐색 접근

완전 탐색 접근을 통해서 모든 경우를 직접 하나하나 찾아내 보자.

본 문제에서 “경우”란, 길이에 맞는 **오르막 수를 전부 하나하나 찾는다**는 것이다.

I Dynamic Programming 접근

1) 풀고 싶은 **가짜 문제** 정의 – **일단 도전해보기**

진짜 문제 := 길이가 N인 오르막 수의 개수

가짜 문제 := `Dy[i]` = 길이가 `i`인 오르막 수의 개수

i	1	2	3	4	5	6	7
Dy[i]							

(레벨 1: 진짜 문제랑 똑같은 가짜 문제인 경우)

I Dynamic Programming 접근

1) 풀고 싶은 **가짜 문제** 정의

진짜 문제 := 길이가 N인 오르막 수의 개수

가짜 문제 :=

`Dy[i][last]` = 길이가 `i`이며 `last`로 끝나는 오르막 수 의 개수

Dy	0	1	2	3	…	8	9
i=1							
2							
…							
N							

(레벨 3: 새로운 정의가 필요한 경우)

I Dynamic Programming 접근

2) **가짜 문제**를 풀면 *진짜 문제*를 풀 수 있는가?

Dy 배열을 가득 채울 수만 있다면? *진짜 문제*에 대한 대답은

Dy[N][0] + Dy[N][1] + … + Dy[N][9] 이다.

Dy	0	1	2	3	…	8	9
i=1							
2							
…							
N							

I Dynamic Programming 접근

3) **초기값**은 어떻게 되는가?

초기값: 쪼개지 않아도 풀 수 있는 “작은 문제”들에 대한 정답

Dy	0	1	2	3	…	8	9
i=1	1	1	1	1	…	1	1
2							
…							
N							

I Dynamic Programming 접근

4) **점화식** 구해내기

1. Dy[i] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)
2. 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

I Dynamic Programming 접근

4-1) Dy[i][k] 계산에 필요한 탐색 경우를 공통점끼리 묶어 내기 (Partitioning)

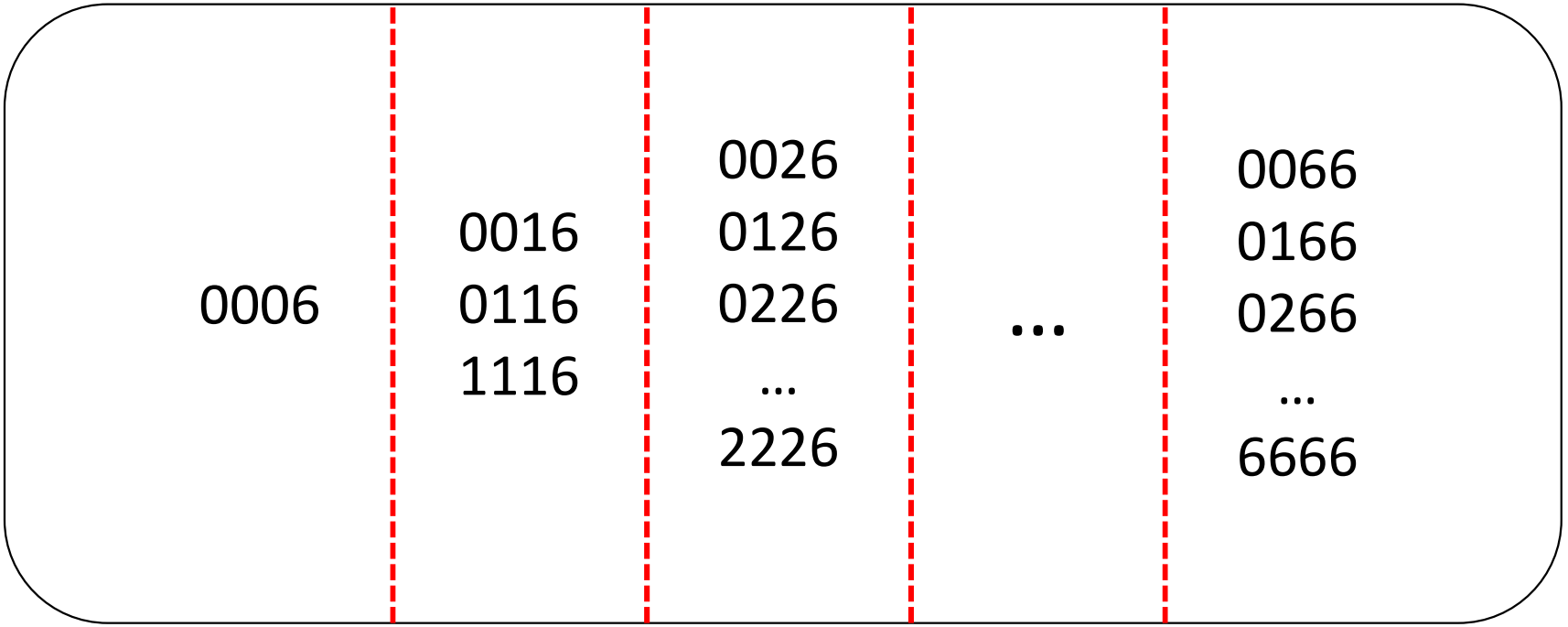
Dy[4][6] 계산에 필요한 탐색 경우들

0006	1116	2226		
0016	1126	2236		
0026	1136	2246	...	6666
...	...	...		
0666	1666	2666		

I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

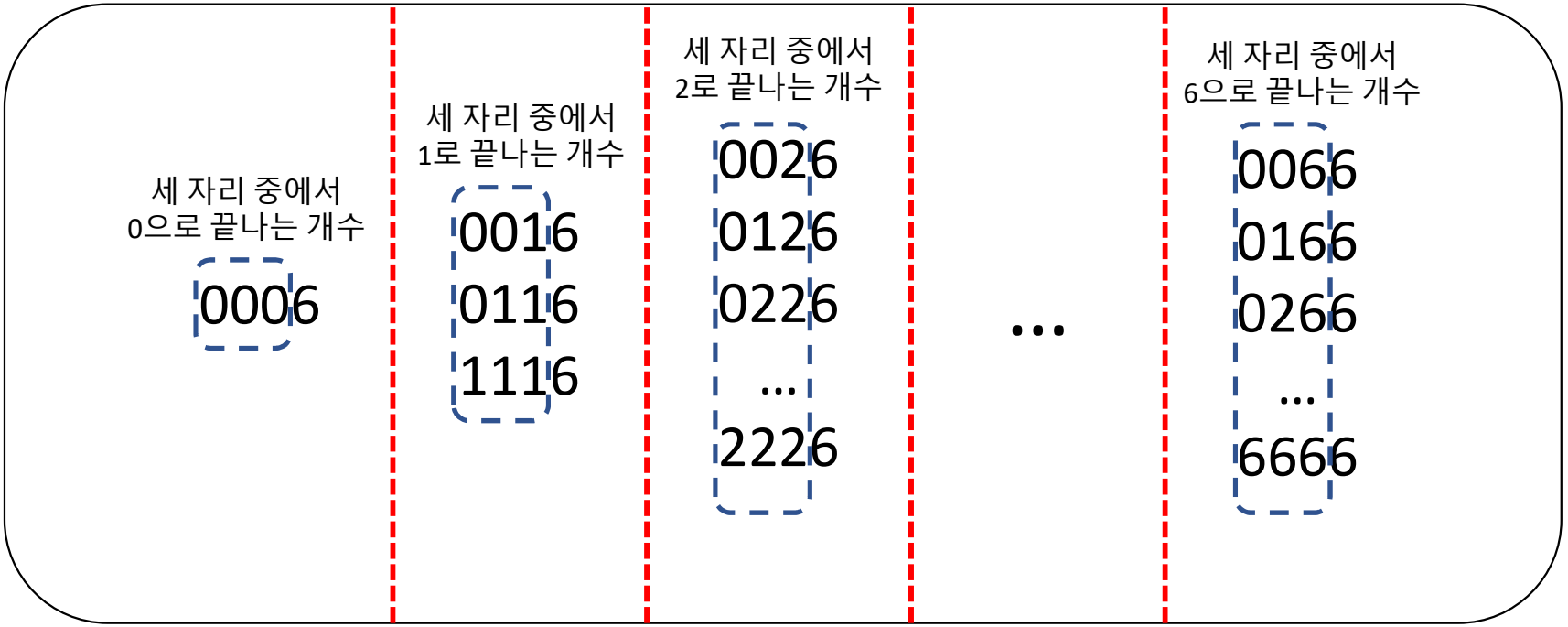
★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

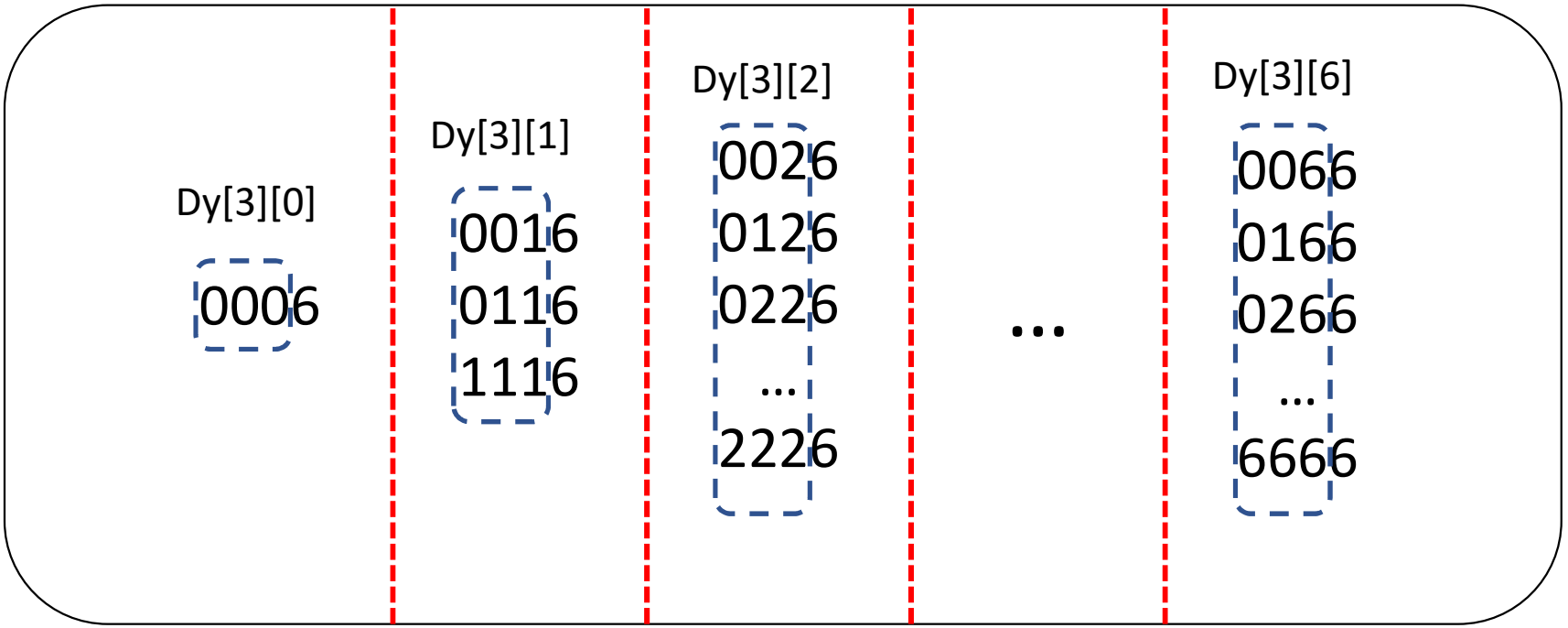
★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



I Dynamic Programming 접근

4-2) 묶어낸 부분의 정답을 Dy 배열을 이용해서 빠르게 계산해보기

★ 최대 점수 = (각 Partition의 최대 점수) 들 중의 최대 점수



I Dynamic Programming 접근

4) **점화식** 구해내기

$$Dy[i][j] = \sum_{k=0}^{k=j} Dy[i-1][k]$$

Dy	0	1	2	3	…	8	9
i=1	1	1	1	1	…	1	1
2	1	2	3	4	…	8	9
3	1	3	6	10	…	36	45
4	1	4	10	20	…	120	165

길이가 4인 오르막수 = 1+4+…+165 = 495

I 시간, 공간 복잡도 계산하기

완전 탐색을 통해 모든 경우를 세면 정답의 개수만큼

의 시간이 걸리지만, Dy 배열을 채우는 것은 O(N*10^2)

이라는 시간 복잡도면 충분하다!

I 구현

```
static void pro() {
    // 초기값 구하기
    /* TODO */

    // 점화식을 토대로 Dy 배열 채우기
    /* TODO */

    // Dy배열로 정답 계산하기
    int ans = /* TODO */;

    System.out.println(ans);
}
```

I 연습 문제

- BOJ 2688 – 줄어들지 않아
- BOJ 1562 – 계단 수
- BOJ 2096 – 내려가기
- BOJ 5557 – 1학년
- BOJ 1495 – 기타리스트
- BOJ 9095 – 1, 2, 3 더하기
- BOJ 15988 – 1, 2, 3 더하기 3
- BOJ 15990 – 1, 2, 3 더하기 5

이외의 추천 문제가 추가되면 Github 자료에 코드 업로드