

# 2017 SCSC Workshop

Sogang univ.  
System modeling & Optimization Lab  
Sangdurk Han

# Dynamic Programming

# Dynamic Programming

- 큰 문제의 해답에 작은 문제의 해답이 포함되어 있고, 이를 재귀 호출 알고리즘으로 구현하면 지나친 중복이 발생하는 경우에 이 재귀적 중복을 해결하는 방법
- 일반적으로 최적화 문제에 적용
- 잦은 출제, 많이 풀어보자

# Dynamic Programming

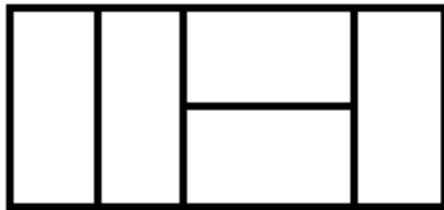
- 1. 정의
- 2. 초기화
- 3. 점화식 (dp식)

# Dynamic Programming

- 1. Iterative
- 2. Recursive
- 구현하기 편한 걸 사용하면 된다.

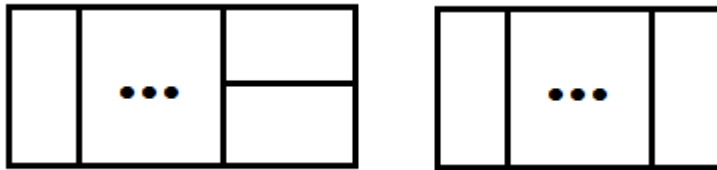
# 2 x n 타일링

- $2 \times N$  크기의 직사각형을  $1 \times 2$ ,  $2 \times 1$  타일로 채우는 방법의 수를 구하는 프로그램을 작성하시오.
- 아래 그림은  $2 \times 5$  크기의 직사각형을 채운 한 가지 방법의 예시이다.



# Solution

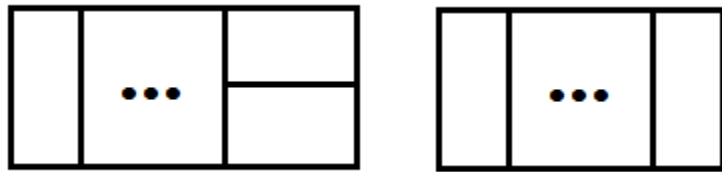
- 1. 정의
- $D[i]$  = 왼쪽에서부터  $i$ 칸까지 채웠을 때,  
경우의 수
- 2. 초기화  $D[0] = 0$
- 3. 점화식  
타일이 추가 되는 경우



# Solution

- 3. 점화식

타일이 추가 되는 경우



$$D[i] = D[i-2] + D[i-1]$$



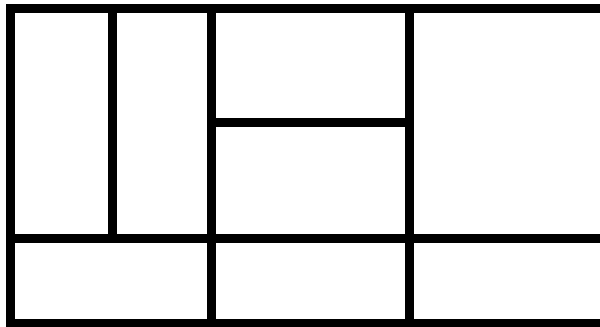
# Solution (code)

```
int d[N];
int main(){
    int n;
    scanf("%d", &n);
    d[0] = 1;

    for(int i=0; i<n; i++){
        d[i+1] = (d[i+1]+d[i]) % M;
        d[i+2] = (d[i+2]+d[i]) % M;
    }
    printf("%d\n", d[n]);
}
```

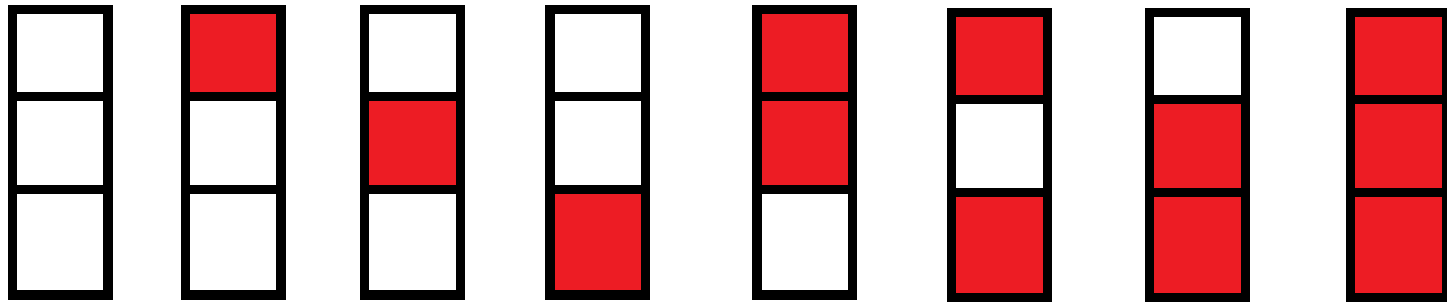
# 3 x n 타일링

- $3 \times N$  크기의 직사각형을  $1 \times 2$ ,  $2 \times 1$  타일로 채우는 방법의 수를 구하는 프로그램을 작성하시오.
- 아래 그림은  $3 \times 5$  크기의 직사각형을 채운 한 가지 방법의 예시이다.



# Solution

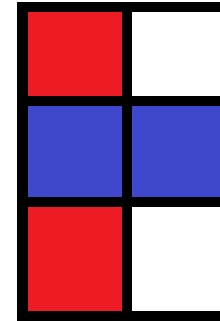
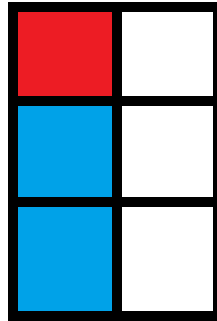
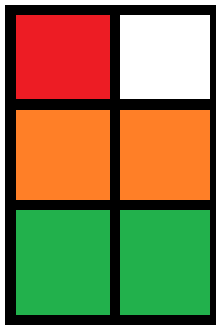
- 1. 정의
- $D[i][p]$  =  $i-1$  번째 칸까지 채웠고,  
     $i$  번째 칸이 패턴  $p$ 로 채워졌을 때
- 패턴



# Solution

- 2. 초기화
- $D[0][7] = 1$  // 0번째 칸은 모두 채워져 있다고 생각
- 3. 점화식  
다음에 올 수 있는 패턴 생각

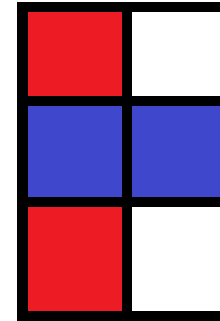
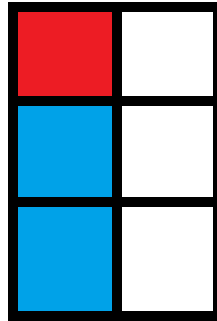
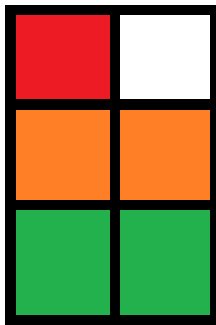
Ex)



# Solution

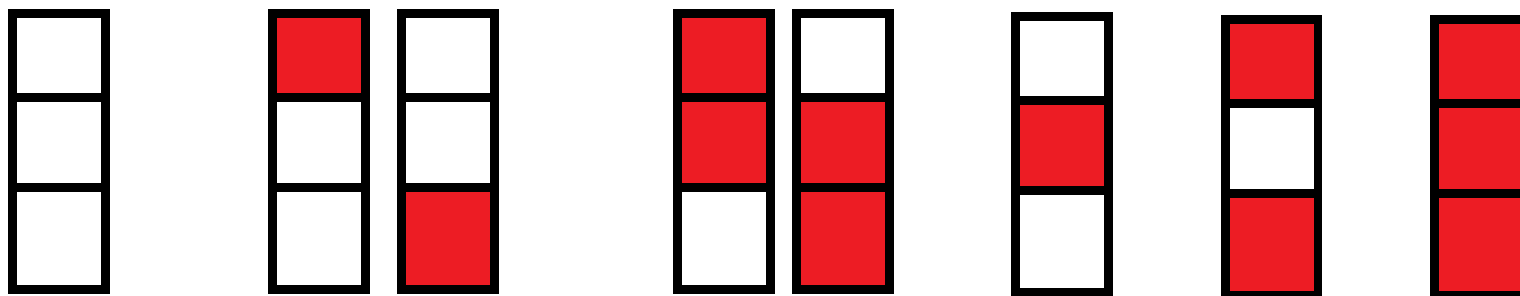
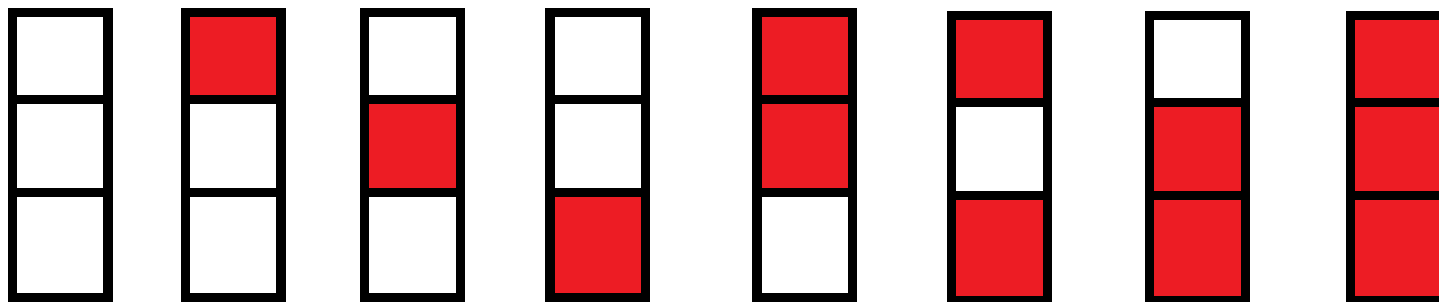
- 2. 초기화
- $D[0][7] = 1$  // 0번째 칸은 모두 채워져 있다고 생각
- 3. 점화식  
다음에 올 수 있는 패턴 생각

Ex)



# Solution

- 패턴을 줄여보자



# Solution

- 다른 방법도 많고, 풀이 방법도 무수히 많음
- 더 좋은 방법을 찾아보자 !



이런 경우가 존재할까?

# 조약돌 놓기 문제









- $3 \times N$  테이블의 각 칸에 숫자가 쓰여 있다. 테이블의 칸들 중에 일부에 제한 조건을 만족하는 방법으로 조약돌을 놓아 조약돌이 놓인 곳에 있는 수의 합을 최대로 하자.
- 1. 가로나 세로로 인접한 두 칸에 동시에 조약돌이 놓일 수 없다.
- 2. 각 열에는 하나 이상의 조약돌을 놓는다.



# 문제의 예

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

문제의 예

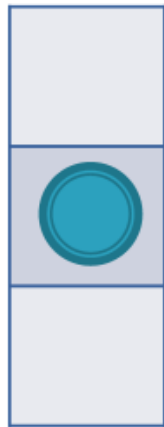
	7	12	-5	5		11	
-8		14		7	13		5
	12		4		-2	9	

올바른 예

# 4가지 패턴



1



2



3

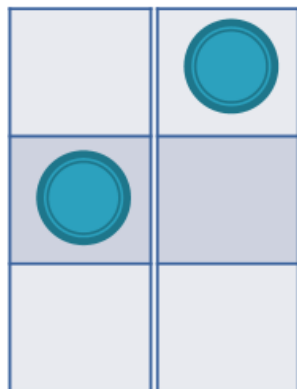


4

# 서로 양립할 수 있는 패턴

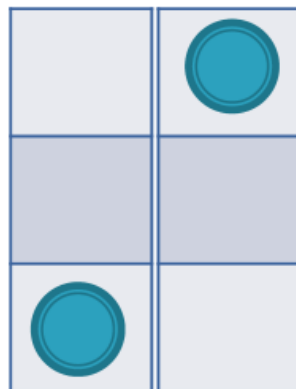


1



2

1

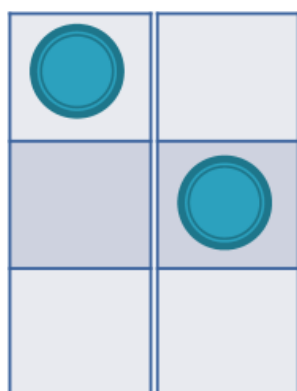


3

1

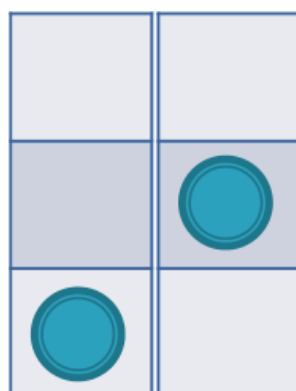


2



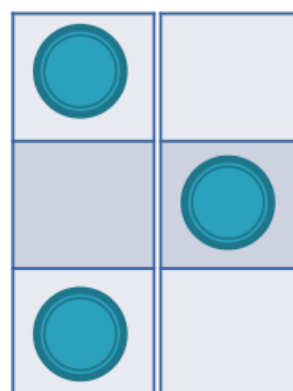
1

2



3

2



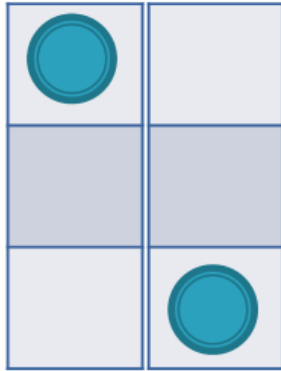
4

2

# 서로 양립할 수 있는 패턴

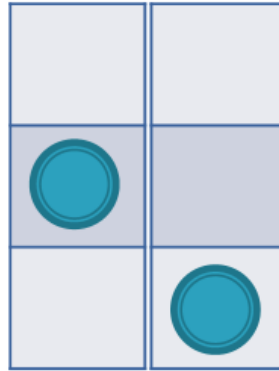


3



1

3

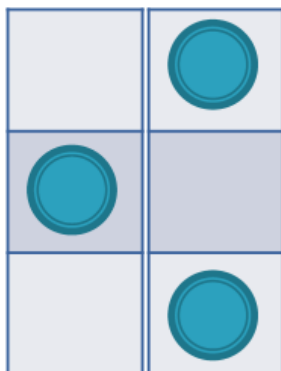


2

3



4



2

4

# 정의

- $N$ 열 중 1열부터  $i$ 열까지 돌을 놓는 경우에 1열부터  $i$ 열까지의 합 의 최고치를 생각해보자.
- $i$ 열에는 4가지 패턴 중 하나로 돌이 놓일 것이므로 아래 4가지를 구해보.
- $i$ 열이 패턴 1 ~ 4로 놓여 있을 경우의 최고점



1



2



3



4

# 최적 부분 구조의 존재

- $i$ 열까지의 최적해는  $i-1$ 열까지의 최적해를 포함하고 있다.
- 즉, 자신보다 크기가 하나 작은 문제의 최적해를 자신의 최적해를 구성하는 데에 사용한다.
- 큰 문제의 해답이 작은 문제의 해답을 포함한다.

# 점화식 재귀적 관계

- $D[i][p] = i$ 열이 패턴  $p$ 로 놓일 때의 최고 점수
- $W[i][p] = i$ 열이 패턴  $p$ 로 놓일 때  $i$ 열에 돌이 놓인 곳의 점수 합
- $D[i][p] = W[1][p] \ (i = 1)$   
 $\text{Max}(D[i - 1][q]) + W[i][p] \ (i > 1)$   
이때  $q$ 는  $p$ 와 양립하는 패턴

# 조약돌 놓기 문제 (재귀 호출)

```
int Pebble(int i, int p){
    if( i == 1 ) return W[i][p];
    else{
        Max = -INF;
        for(int q=1; q<=4; q++){
            if( q와 p가 양립 ){
                temp = pebble(i-1, q);
                if( temp > Max)
                    Max = temp;
            }
        }
        return Max + W[i][p];
    }
}
```



# 중복 호출의 횟수

문제의 크기 (N)	부분 문제의 총 수	함수 Pebble()의 총 호출 횟수
1	4	4
2	8	12
3	12	30
4	16	68
5	20	152
6	24	332
7	28	726

# 조약돌 놓기 문제 (DP)

```
int pebble(int n){  
    for(int p=1; p<=4; p++)  
        D[1][p] = W[1][p];  
    for(int i=2; i<=n; i++)  
        for(int p=1; p<=4; p++)  
            D[i][p] = Max(D[i-1][q]) + W[i][p];  
    return Max( D[n][p] );  
}
```

# 점화식은 하나가 아니다!

- 어떤 문제에 대한 점화식은 단 하나만 존재하지 않을 수 있으며 **시간**, **메모리** 소모가 달라질 수 있다.
- 점화식의 정의를 어떻게 세우느냐에 따라, 해를 구하는 과정은 다를지 몰라도, 답은 항상 같다.
- 처음에는 비효율(**생각나는 대로**)적으로도 정의를 내리고 필요 없는 정보들을 제거한다.

# New FA

- 서강대학교에는 FA제도가 존재한다. 하지만, FA제도를 반대하는 학생들이 많아지자, 학교에서는 새로운 FA제도를 만들었다.
- 출결은 다음과 같이 3가지가 존재한다.
  1. 출석, 2. 결석, 3. 지각
- FA를 받는 사람은 **지각을 두 번 이상** 했거나, **결석을 세 번 연속**으로 한 사람이다.
- 어떤 학기의 수업이 수  $N$ 이 주어졌을 때, FA를 받지 않는 출결 정보의 경우의 수를 구해보자.

# New FA

- 서강대학교에는 FA제도가 존재한다. 하지만, FA제도를 반대하는 학생들이 많아지자, 학교에서는 새로운 FA제도를 만들었다.
- 출결은 다음과 같이 3가지가 존재한다.
  1. 출석, 2. 결석, 3. 지각
- FA를 받는 사람은 **지각을 두 번 이상** 했거나, **결석을 세 번 연속**으로 한 사람이다.
- 어떤 학기의 수업이 수  $N$ 이 주어졌을 때, FA를 받지 않는 출결 정보의 경우의 수를 구해보자.

# New FA

- 예를 들어, 어떤 학기의 수업의 개수가 4개이고, 출석은 O, 지각은 L, 결석은 A라고 표시하면, FA를 받지 않는 출결 정보는 다음과 같이 43가지가 존재한다.

0000	000A	000L	00A0	00AA	00AL	00LO	00LA	0A00	0A0A	0A0L	0AA0
0AAL	0ALO	0ALA	0LO0	0LOA	0LA0	0LAA	A000	A00A	A00L	A0A0	A0AA
A0AL	A0LO	A0LA	AA00	AA0A	AA0L	AAL0	AALA	AL00	ALOA	ALAO	ALAA
L000	L00A	L0A0	L0AA	LA00	L0A0	LAA0					

# Solution 1

- 1. 정의
- $D[Day][Now][Prev][Prev2][Late]$   
= Day일, 지금 상태가 Now이고,  
전날 상태가 Prev, 전전날 상태가 Prev2이고,  
지각이 Late번 했을 때
- (상태 : 출석(0), 결석(1), 지각(2))

# Solution 1

- 2. 초기화

- if( Late  $\geq$  2 ) continue;

- if( Now == 2 || Prev == 2 || prev2 == 2 )

- D[3][Now][Prev][Prev2][1] = 1;

- else

- D[3][Now][Prev][Prev2][0] = 1;



# Solution 1

- 3. 점화식 (엄청 복잡..)

오늘 출석을 했고, ( $\text{Now} = 0$ ), 전날  $\text{Prev}$ , 전전날  $\text{Prev2}$ 이다.

이 것은  $D[\text{Day}][0][\text{Prev}][\text{Prev2}]$ 와 같다.

그럼  $\text{Day}-1$ 날에는

$D[\text{Day}-1][\text{Prev}][\text{Prev2}][\text{Prev3}]$ 와 같을 것이다.

따라서, 오늘 출석을 했고, 지각을 0번 했다면

$D[\text{Day}][0][\text{Prev}][\text{Prev2}][0]$

$\quad \quad \quad += D[\text{Day}-1][\text{Prev}][\text{Prev2}][\text{Prev3}][0]$

# Solution 1

- 3. 점화식 (엄청 복잡..)

오늘 출석을 했고, 지각을 1번 했다면,

$$\begin{aligned} D[Day][0][Prev][Prev2][1] \\ += D[Day-1][Prev][Prev2][Prev3][1] \end{aligned}$$

마찬가지로, 오늘 결석을 했고 지각을 0번, 1번 했을 때도 밑과 같다.

$$\begin{aligned} D[Day][1][Prev][Prev2][0] \\ += D[Day-1][Prev][Prev2][Prev3][0] \end{aligned}$$
$$\begin{aligned} D[Day][1][Prev][Prev2][1] \\ += D[Day-1][Prev][Prev2][Prev3][1] \end{aligned}$$

# Solution 1

- 3. 점화식 (엄청 복잡..)

마지막으로 ,오늘 지각을 했으면, 지각을 0번 했을 경우는 없다.

따라서 다음과 같은 식이 나온다.

```
D[Day][2][Prev][Prev2][1]
    += D[Day-1][Prev][Prev2][Prev3][0]
```

# Solution 1

- 정답은  $D[\text{Day}][i][j][k][l]$ 에 들어있는 모든 값을 더하면 된다.
- $\text{Day} = N$   
 $0 \leq i \leq 2, 0 \leq j \leq 2, 0 \leq k \leq 2, 0 \leq l \leq 1$

# Solution 1 (code)

```
for(int i=4; i<=N; i++){
    for(int Prev=0; Prev<3; Prev++){
        for(int Prev2=0; Prev2<3; Prev2++){
            for(int Prev3=0; Prev3<3; Prev3++){
                if(!D[i-1][Prev][Prev2][Prev3]) continue;
                D[i][0][Prev][Prev2][0] += D[i-1][Prev][Prev2][Prev3][0];
                D[i][0][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][1];
                D[i][1][Prev][Prev2][0] += D[i-1][Prev][Prev2][Prev3][0];
                D[i][1][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][1];
                D[i][2][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][0];
            }
        }
    }
}
```

# Solution 1 (code)

```
for(int i=4; i<=N; i++){
    for(int Prev=0; Prev<3; Prev++){
        for(int Prev2=0; Prev2<3; Prev2++){
            for(int Prev3=0; Prev3<3; Prev3++){
                if(!D[i-1][Prev][Prev2][Prev3]) continue;
                D[i][0][Prev][Prev2][0] += D[i-1][Prev][Prev2][Prev3][0];
                D[i][0][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][1];
                D[i][1][Prev][Prev2][0] += D[i-1][Prev][Prev2][Prev3][0];
                D[i][1][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][1];
                D[i][2][Prev][Prev2][1] += D[i-1][Prev][Prev2][Prev3][0];
            }
        }
    }
}
```

# Solution 2

- 1. 정의
- $D[Day][i][j][k]$   
= 수업이 총 Day개이고, 결석을 연속으로 i번,  
지각을 j번, Day날의 출결 정보가 k일 때
- (상태 : 출석(0), 결석(1), 지각(2))

# Solution 2

- 2. 초기화
- 첫날 가능한 경우는 출석, 결석, 지각 세 가지
- 1) 출석을 했을 때  $D[1][0][0][0] = 1$
- 2) 결석을 했을 때  $D[1][1][0][1] = 1$
- 3) 지각을 했을 때  $D[1][0][1][2] = 1$



# Solution 2

- 3. 점화식

오늘 출석을 했을 때는, 오늘을 포함해서 결석을 연속으로 할 수 없으므로,  $i=0$ 이다.

이 때에는, 지각을 0번 했을 때와, 지각을 1번 했을 때로 경우의 수를 나눌 수 있다.

# Solution 2

- 3. 점화식

지각을 0번 했을 때는 전날 할 수 있는 출결은 결석과 출석 밖에 없으므로 다음과 같은 식을 세울 수 있다.

$$\begin{aligned} D[i][0][0][0] &= D[i-1][0][0][0] \\ &\quad + D[i-1][1][0][1] \\ &\quad + D[i-1][2][0][1] \end{aligned}$$

# Solution 2

- 3. 점화식

지각을 1번 했을 때는 위의 경우에 지각을 하는 경우도 있으므로 다음과 같이 세울 수 있다.

$$\begin{aligned} D[i][0][1][0] &= D[i-1][0][1][0] \\ &\quad + D[i-1][1][1][1] \\ &\quad + D[i-1][2][1][1] \\ &\quad + D[i-1][0][1][2] \end{aligned}$$

# Solution 2

- 3. 점화식

오늘 결석을 했을 때는, 다음과 같이 2가지 경우로 나눌 수 있다.

1) 오늘이 첫 번째 결석일 때

2) 오늘이 두 번째 연속 결석일 때

또, 각각의 경우에 지각을 0번 했을 때, 1번 했을 때와 같이 4가지 경우로 나눌 수 있다.

# Solution 2

- 3. 점화식

오늘이 첫 번째 결석이고, 지각을 한 번도 한 적이 없다면, 전날은 반드시 출석이어야 한다.

$$D[i][1][0][1] = D[i-1][0][0][0]$$

오늘이 두 번째 결석이고, 지각을 한 번도 한 적이 없다면, 전날은 반드시 결석이어야 한다.

$$D[i][2][0][1] = D[i-1][1][0][1]$$

# Solution 2

- 3. 점화식

오늘이 첫 번째 결석이고, 지각을 한 번 한 적이, 전날은 출석이거나, 지각이어야 한다.

$$D[i][1][1][1] = D[i-1][0][1][0] + D[i-1][0][1][2]$$

오늘이 두 번째 결석이고, 지각을 한 번 한 적이 있다면, 전날은 반드시 결석이다.

$$D[i][2][1][1] = D[i-1][1][1][1]$$

# Solution 2

- 3. 점화식

오늘 지각을 했을 때는, 전날 까지는 지각이 있으면 안되고, 전날은 출석이나 결석이어야 한다.

$$\begin{aligned} D[i][0][1][2] &= D[i-1][0][0][0] \\ &\quad + D[i-1][1][0][1] \\ &\quad + D[i-1][2][0][1] \end{aligned}$$

# Solution 2

- 3. 점화식

오늘 결석을 했을 때는, 다음과 같이 2가지 경우로 나눌 수 있다.

1) 오늘이 첫 번째 결석일 때

2) 오늘이 두 번째 연속 결석일 때

또, 각각의 경우에 지각을 0번 했을 때, 1번 했을 때와 같이 4가지 경우로 나눌 수 있다.



# Solution 2

- 정답은  $D[\text{Day}][i][j][k]$ 에 들어있는 모든 값을 더하면 된다.
- $\text{Day} = N$   
 $0 \leq i \leq 2, 0 \leq j \leq 2, 0 \leq k \leq 2$

## Solution 2 (code)

```
for(int i=2; i<=N; i++){  
    D[i][0][0][0] = D[i-1][0][0][0] + D[i-1][1][0][1]  
                  + D[i-1][2][0][1];  
    D[i][0][1][0] = D[i-1][0][1][0] + D[i-1][1][1][1]  
                  + D[i-1][2][1][1]  
                  + D[i-1][0][1][2];  
    D[i][1][0][1] = D[i-1][0][0][0];  
    D[i][2][0][1] = D[i-1][1][0][1];  
    D[i][1][1][1] = D[i-1][0][1][0]  
                  + D[i-1][0][1][2];  
    D[i][2][1][1] = D[i-1][1][1][1];  
    D[i][0][1][2] = D[i-1][0][0][0] + D[i-1][1][0][1]  
                  + D[i-1][2][0][1];  
}
```

# Solution 3

- 1. 정의
- $D[Day][Late][Prev][Now]$   
= Day일, 지각 Late번,  
전날의 상태 Prev, 오늘 상태 Now
- (상태 : 출석(0), 결석(1), 지각(2))

# Solution 3

- 1. 정의

이 점화식에서는 경우의 수를 생각할 필요가 없다.

문제에 써 있는 “결석은 연속 두 번 까지 허용된다”  
와 “지각은 단 한 번만 할 수 있다.”를

출석 : 0

지각 : 1

결석 : 2

라는 숫자로 표현해서 간단하게 나타낼 수 있다.

# Solution 3

- 2. 초기화

```
for(int i=0; i<=2; i++){  
    for(int j=0; j<=2; j++){  
        if(i%2+j%2==2) continue;  
        D[2][i%2+j%2][i][j]=1;  
    }  
}
```

- (  $i\%2 + j\%2$ )가 의미하는 것은 ?  
 지각의 횟수

# Solution 3

- 3. 점화식

```
D[Day][Late+Now%2][Prev][Now]  
+= D[Day-1][Late][Prev2][Prev]
```

이 점화식에서는 변수의 범위를 정하는 것이 중요하다.

$$0 \leq \text{Prev} \leq 2$$
$$0 \leq \text{Now} \leq 2$$
$$0 \leq \text{Late} + \text{Now} \% 2 \leq 1$$
$$0 \leq \text{Prev2} \leq 2$$

# Solution 3

- 3. 점화식

결석을 연속으로 3번 했을 때의 처리는 어떻게?

$\text{Prev} + \text{Now} + \text{Prev2} == 60$ 이면 결석이 연속 3번

수 0, 1, 2 중 2로 나누어서 나머지가 1인 것은 1  
뿐이므로 지각을 1로 표현해서 간단하게 나타낼  
수 있다.

# Solution 3

- 정답은  $D[\text{Day}][\text{Late}][\text{Prev}][\text{Now}]$ 에 들어있는 모든 값을 더하면 된다.
- $\text{Day} = N$   
 $0 \leq \text{Late} \leq 1, 0 \leq \text{Prev} \leq 2, 0 \leq \text{Now} \leq 2$



# Solution 3 (code)

```
for(int i=3; i<=N; i++){
    for(int Prev=0; Prev<=2; Prev++){
        for(int Now=0; Now<=2; Now++){
            for(int Late=0; Late+Now%2<=1; Late++){
                for(int Prev2=0; Prev2<=2; Prev2++){
                    if(Prev+Now+Prev2==6) continue;
                    D[i][Late+Now%2][Prev][Now]
                        += D[i-1][Late][Prev][Prev2];
                }
            }
        }
    }
}
```

# Solution 4

- 1. 정의
- $D[i][j][k]$   
= i일, 지각 j번, 연속 결석 k번
- (상태 : 출석(0), 결석(1), 지각(2))  
  
k = 0 (출석 or 지각)  
K = 1, 2 (연속 결석이 1번 or 2번)

# Solution 4 (code)

```
D[0][0][0] = 1;
for(int i=1; i<=N; i++){
    for(int j=0; j<2; j++){
        for(int k=0; k<3; k++){
            D[i][j][0] += d[i-1][j][k];
            if(j==0)
                D[i][1][0] += d[i-1][j][k];
            if(k<2)
                D[i][j][k+1] += D[i-1][j][k];
        }
    }
}
```

# Solution 5

- 1. 정의
- $D[i][j]$   
=  $i$ 일, 연속 결석  $j$ 번
- (상태 : 출석(0), 결석(1), 지각(2))
- 이 점화식에는 지각이 존재하지 않는다.  
일단 지각이 없다고 생각하고 문제를 풀어보자.

# Solution 5 (Best)

- 1. 정의
- 결석 연속 0번 : 출석
- 결속 1번, 2번 : 결석
- 위와 같이 생각해서 문제를 풀면 된다.

# Solution 5 (Best)

- 3. 점화식

따라서, 오늘 출석을 했을 때는, 전날 아무거나 해도 상관없다.

$$D[i][0] = D[i-1][0] + D[i-1][1] + D[i-1][2]$$

오늘 한 결석이 연속 한 번이라면, 전날은 반드시 출석이어야 한다. (지각이 없으므로)

$$D[i][1] = D[i-1][0]$$

오늘 한 결석이 연속 두 번이라면, 전날은 반드시 연속 한 번한 결석이어야 한다.

$$D[i][2] = D[i-1][1]$$

# Solution 5 (Best) (code)

```
for(int i=2;i<=N; i++){
    D[i][0]=D[i-1][0]+D[i-1][1]+D[i-1][2];
    D[i][1]=D[i-1][0];
    D[i][2]=D[i-1][1];
}
sum = D[N][0]+D[N][1]+D[N][2];
for(int i=0;i<N;i++)
    for(int j=0;j<=2;j++)
        for(int k=0;k<=2;k++)
            sum+=D[i][j]*D[N-i-1][k];
```

Q&A