동적 계획법

동적 계획법 (DP, Dynamic Programming)

- 작은 부분부터 문제를 풀어 전체 문제를 해결하는 방법.
- 점화식을 찾은 후 반복 구조로 구현.
- 빠른 속도로 해를 구할 수 있다.
- 매우 다양한 문제에 적용되므로 쉬운 것부터 많은 연습이 필요.

■ 팩토리얼

- f(n) = n * f(n−1)
- f(1) = 1, f(0) = 1

```
f(n)
    if(n<2)
        return n
    else
        return n*f(n-1)
```

⇨

```
f[0] = 1
f[1] = 1
for i : 2 -> n
    f[i] = i * f[i-1]
```

연습

n!을 1,000,000,007로 나눈 나머지를 출력하는 프로그램을 작성하시오.

▪ 피보나치 수(Fibonacci number)

- f(n) = f(n−1) + f(n−2)
- f(0) = 1, f(1) = 1
  - 1, 1, 2, 3, 5, 8, 13, 21, …

✓f(0) = 0, f(1) = 1인 경우

✓0, 1, 1, 2, 3, 5, 8, 13, 21, …

```
f(n)
    if(n<2)
        return 1
    else
        return f(n−1) + f(n−2)
```

⇨

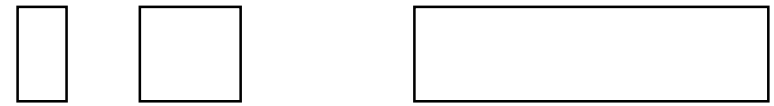
```
f[0] = 1
f[1] = 1
for i : 2 -> n
    f[i] = f[i−1] + f[i−2]
```

연습

크기가 2x1인 타일을 2xn인 공간에 붙이는 경우의 수를 구하시오.

연습

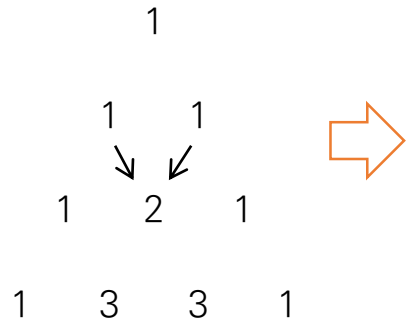
- 1x2, 2x2 크기의 타일을 nx2 크기의 공간에 붙이는 경우의 수를 구하시오. 1x2 타일은 가로, 세로로 모두 붙일 수 있음.

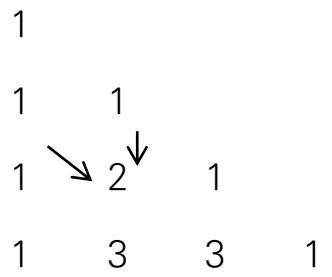


1x2 2x2 nx2

▪ 이항계수

- $(a+b)^n$에서 $a^{n-k}b^k$의 계수를 구하는 문제. $\binom{n}{k}$
- (a+b)(a+b)...(a+b)처럼 n개의 (a+b)가 있을 때, k개의 b와 n-k개의 a를 고르는 경우의 수. 또는 k개의 a와 n-k개의 b를 고르는 경우의 수.
- 파스칼의 삼각형





	0	1	2	3	k
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
n					

```
C[n][k] = 1, k = 0 or n == k
C[n][k] = C[n-1][k-1] + C[n-1][k]
```

▪ 오른쪽과 아래로 움직이기

- 숫자판의 맨 왼쪽 위에서 출발, 오른 쪽 아래 도착.
- 각 칸에서는 오른쪽과 아래로만 이동 가능.
- 지나는 칸의 숫자 합계가 최대가 되도록 움직였을 때 합계 구하기.

m	1	2	3	4	5	j
1	5	2	8	8	8	
2	4	4	8	5	8	
3	10	9	6	3	5	
i						

d	1	2	3	4	5	j
1						
2						
3						
i						

(i, j)칸에 진입은 위(i−1,j)나 왼쪽(i, j−1)에서만 가능.
각 칸까지의 최대 합계를 d[i][[j]라고 하면,
d[i][j] = max(d[i−1][j], d[i][j−1]) + m[i][j], i〉1, j〉1
d[i][j] = d[i][j−1] + m[i][j], i = 1
d[i][j] = d[i−1][j] + m[i][j], j = 1

d	0	1	2	3	4	5	j
0	0	0	0	0	0	0	
1	0						
2	0						
3	0						
i							

d[i][j] = 0, i==0 또는 j==0
d[i][j] = max(d[i−1][j], d[i][j−1]) + m[i][j], i〉0, j〉0

▪ 최장 공통 부분 수열

(LCS, Longest Common Subsequence)

- 부분 수열
  - 주어진 수열에서 순서가 바뀌지 않게 일부 숫자를 고른 것.
- 최장 공통 부분 수열
  - 두 개의 수열에서 고른 부분 수열이 일치할 때, 가장 긴 수열의 길이 구하기.

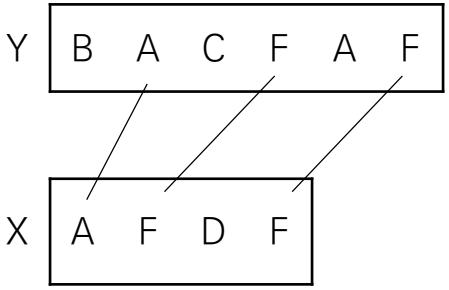
3 2 5 7 4

⇨ 2 5 4

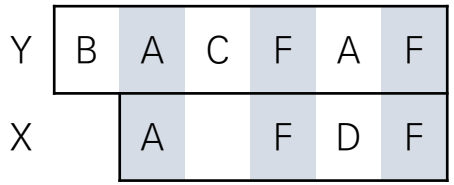
1 2 4 2 5 6 8 4

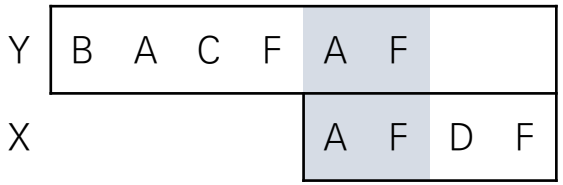
✓ 글자의 간격을 조절해 아래 위 글자를 일치시켜 본다.

- LCS 적용
  - 양쪽에서 같은 글자끼리 선으로 연결 할 때, 교차되지 않는 선분의 최대 개수는?

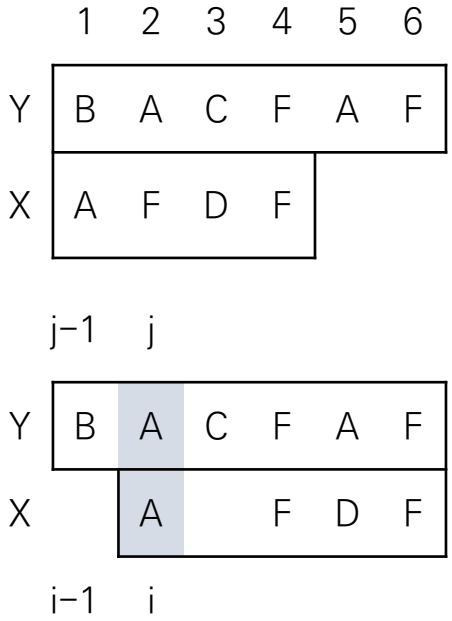


글자를 맞추는 여러가지 방법





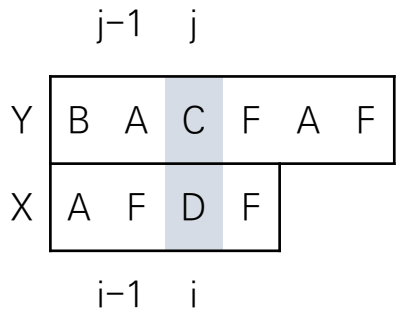
- LCS 테이블
  - 각각에 속한 글자를 Xi, Yj라고 하면, 이 경우 1<=i<=4, 1<=j<=6.
  - i 또는 j가 0인 경우는 글자가 없는 경우를 나타냄.
  - M[i][j]는 Xi와 Yj까지 고려했을 때의 LCS 길이를 저장.



Xi == Yj인 경우
M[i][j] = M[i-1][j-1]+1

글자가 일치하는 경우, i-1과 j-1까지
고려한 수열의 길이 +1

- LCS 테이블



Xi != Yj인 경우
M[i][j] = max(M[i][j−1], M[i−1][j])

글자가 일치하지 않는 경우, i와 j−1까지
또는 i−1과 j까지 고려한 길이 중 큰 값

i == 0 또는 j == 0인 경우
M[i][j] = 0

M[i][j]

	∅	B	A	C	F	A	F
∅	0	0	0	0	0	0	0
A	0	0	1	1	1	1	1
F	0						
D	0						
F	0						

j

i

▪ 부분 집합의 합

- { 1, 1, 2, 2, 1 }을 원소로 갖는 집합 a의 부분 집합에서, 합이 5가 되는 경우가 있는가?
- 부분집합으로 고려한 원소 a_i, 0 <= i <= 5, a_0는 공집합.
- 부분집합의 합 j, 0 <= j <= 5
- m[i][j]는 i원소까지 고려했을 때, 부분 합 j를 만들 수 있으면 1 아니면 0.
- j가 존재하려면 m[i-1][j]가 존재하거나 m[i-1][j-a_i]가 존재해야 함.
  - m[i-1][j]==1 : a_i를 고려하기 전에 이미 합이 j인 경우가 존재함.
  - m[i-1][j-a_i]==1 : a_{i-1}까지 고려한 합 중에 j-a_i가 있으면, a_i를 더해서 j를 만들 수 있음.

- 부분 집합의 합 DP

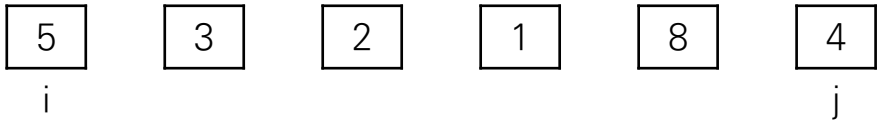
		0	1	2	3	4	5	s_j
0	ϕ	1	0	0	0	0	0	
1	1	1	1	0	0	0	0	
2	1	1	1	1				
3	2	1						
4	2	1						
5	1	1						
i	a_i							

어떤 원소든 고려해도
포함하지 않으면 합이 0

m[i][j] = m[i−1][j] || m[i−1][j−a_i], i〉0, j〉0
m[i][j] = 1, j=0
m[i][j] = 0, i=0, j〉0

▪ 카드 게임

- 숫자 카드를 한 줄로 늘어 놓고 교대로 가져가는 게임.
- 순서마다 양쪽 끝의 카드 중 하나를 선택해 가질 수 있음.
- 게임이 끝났을 때 한사람이 가져온 카드의 합이 최대인 경우 그 합은?
- 두 사람은 같은 전략을 써서 카드를 가져감.



남은 카드의 왼쪽 인덱스 i, 오른쪽이 j라고 했을 때,
이번 순서에 가져갈 수 있는 카드는 i 또는 j 이다.
한 사람이 가져갈 수 있는 카드의 최대 값이 m[i][j]라고 한다.

▪ 카드 게임

✓ m[i][j]는 i부터 j까지 남은 카드에서 가질 수 있는 카드의 최대 합계.

5	3	2	1	8	4	
i					j	이번 차례에 남은 구간
	i+1				j	i를 택하면 다음 사람에 남는 구간
		i+2			j	다음 사람이 i+1을 택하면 남는 구간
	i+1			j-1		다음 사람이 j를 택하면 남는 구간

다음 사람은 m[i+2][j]와 m[i+1][j-1] 중 작은 쪽을 남길 것임.

i			j-1		j를 택하면 다음 사람에 남는 구간
	i+1		j-1		다음 사람이 i를 택하면 남는 구간
i		j-2			다음 사람이 j-1을 택하면 남는 구간

다음 사람은 m[i+1][j-1]과 m[i][j-2] 중 작은 쪽을 남길 것임.

a[]는 카드 숫자가 저장된 배열.

```
min[i][j] = max(a[i] + min(m[i+2][j], m[i+1][j-1]),   // i를 택한 경우
                a[j]+ min(m[i+1][j-1], m[i][j-2])  // j를 택한 경우
```

■ 연속 행렬 곱셈

- 행렬 곱셈에서 결합법칙을 적절히 사용하면 연산 횟수를 줄일 수 있음.
- 행렬의 곱셈 A1 x A2

$$\begin{bmatrix} a1 & a2 \\ a3 & a4 \end{bmatrix} \times \begin{bmatrix} b1 & b2 & b3 \\ b4 & b5 & b6 \end{bmatrix} = \begin{bmatrix} a1b1+a2b4 & a1b2+a2b5 & a1b3+a2b6 \\ a3b1+a4b4 & a3b2+a4b5 & a3b3+a4b6 \end{bmatrix}$$

2 x 2 　　2 x **3** 　　**2** x **3**

  - 곱셈 연산의 횟수 : 2 * 2 * 3 = 12

- 곱하는 행렬 크기 저장
  - 중복은 제거

	0	1	2
p	2	2	3

  - A1의 크기 p[0]xp[1], A2의 크기 p[1]xp[2]
  - Ai의 크기 p[i−1]xp[i]

- 연속 행렬 곱셈
  - A1(2x2), A2 (2x3), A3 (3x4) 행렬을 곱하는 경우
  - A1A2의 크기 2x3
  - A1A2A3의 크기 2x4
  - Ai부터 Aj까지 곱하는 경우의 크기 p[i−1]xp[j]

	Ai				Aj	
	A1		A2		A3	
	2	2	2	3	3	4
	p[0]	p[1]	p[1]	p[2]	p[2]	p[3]

A1A2	p[0]*p[2]
A1A2A3	p[0]*p[3]
Ai…Aj	p[i−1]*p[j]

■ 연속 행렬 곱셈

- A1A2A3의 곱셈 횟수
  - (A1)(A2A3)와 (A1A2)(A3)의 곱셈 횟수 중 작은 쪽을 택함.
  - (왼쪽)(오른쪽)과 같이 결합 법칙이 적용 된 경우의 전체 곱셈 횟수.
    - (왼쪽) 내부 곱셈횟수 + (오른쪽) 내부 곱셈 횟수 + (왼쪽)(오른쪽) 사이 곱셈 횟수
  - (Ai…Ak)(A_{k+1}…Aj)로 표현. (i 〈= k 〈 j)

	Ai		Ak		Aj	
	A1		A2		A3	
	2	2	2	3	3	4
	p[0]	p[1]	p[1]	p[2]	p[2]	p[3]
A1A2	p[0]*p[1]*p[2]					
(A1A2)(A3)	(p[0]*p[1]*p[2]) + (0) + p[0]*p[2]*p[3]					

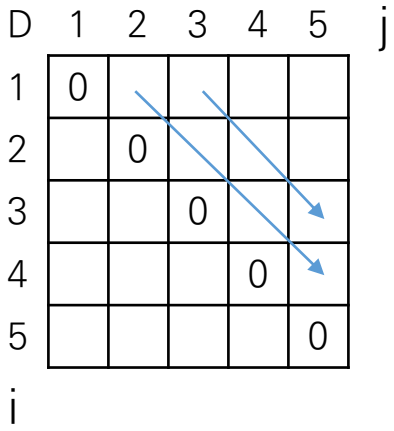
(Ai…Ak)곱셈횟수 + (A_{k+1}…Aj) 곱셈횟수 +
(곱한 결과)(곱한 결과) 사이의 곱셈 횟수

■ 연속 행렬 곱셈

- D[i][j]는 Ai부터 Aj까지 최소 곱셈의 횟수.
  - (Ai…Ak)(A_{k+1}…Aj)인 경우
    - D[i][j] = D[i][k] + D[k+1][j] + p[i−1]p[k]p[j]
    - i<=k<j이므로 모든 k에 대해 계산해서 최소값을 택한다.

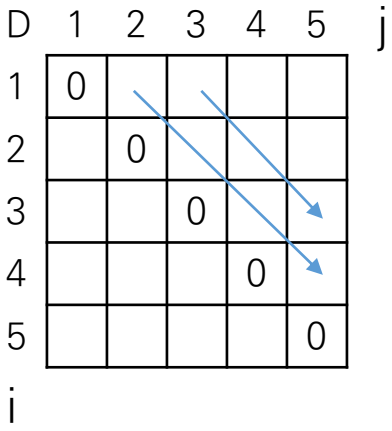
D[i][j] = min(D[i][k] + D[k+1][j] + p[i−1]p[k]p[j]), i<=k<j인 모든 k에 대해
D[i][j] = 0, i==j 인 경우

행렬 A1…A5에 대한 곱셈



A1A2A3은 A1A2와 A2A3가 필요.
A1A2A3A4는 A1A2A3, A2A3A4, A1A2, A2A3가 필요.
2개, 3개… 식으로 곱해지는 행렬의 개수를 늘려감

- $A1 \cdots A_N$에 대한 연속 행렬 곱셈



l	행렬 수	계산 순서	i	j
1	2	(1, 2) (2, 3) (3, 4) (4, 5)	1~4	i+1
2	3	(1, 3) (2, 4) (3, 5)	1~3	i+2
3	4	(1, 4) (2, 5)	1~2	i+3
4	5	(1, 5)	1~1	i+4
			1~(N−l)	i+l

```
for l : 1 -> N-1
    for i : 1 -> N-l
        j = i + l
        for k : i -> j-1
            D[i][j] = min(D[i][k] + D[k+1][j] + p[i-1]p[k]p[j])
```