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

▪ 점화식

- 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]
```

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

$$\boxed{ab \bmod n = [(a \bmod n)(b \bmod n)] \bmod n}$$

- f(n) = n * f(n−1) % 1000000007

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, …

✓ $f(0) = 0$, $f(1) = 1$인 경우

  ✓ 0, 1, 1, 2, 3, 5, 8, 13, 21, …

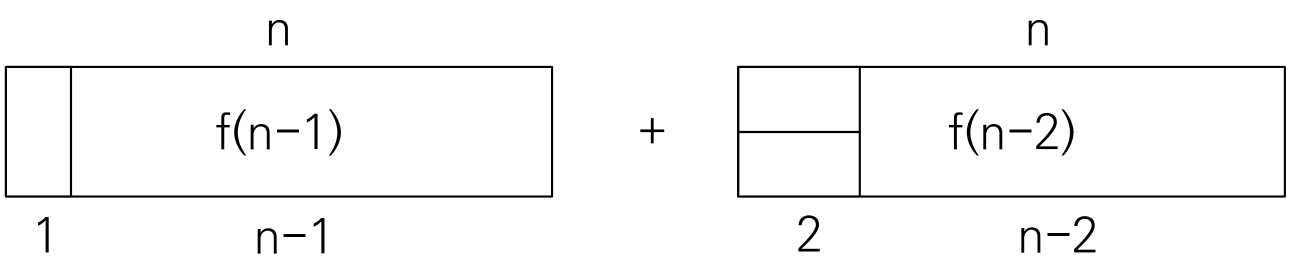
```
f(n)
    if(n<2)
        return 1
    else
        return f(n-1) + f(n-2)
```

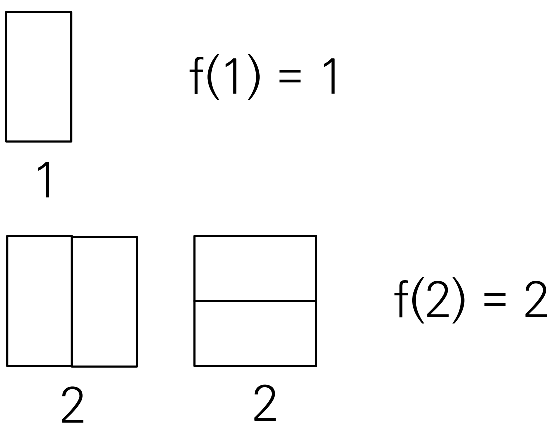
⇨

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

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

점화식을 구하기 위해 두 경우로 나눠서 생각한다.

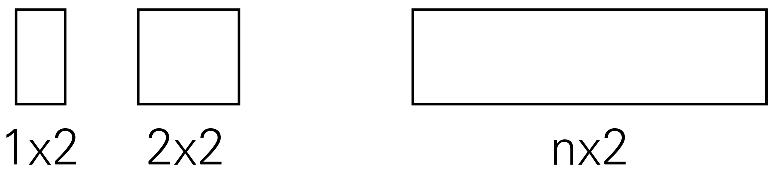


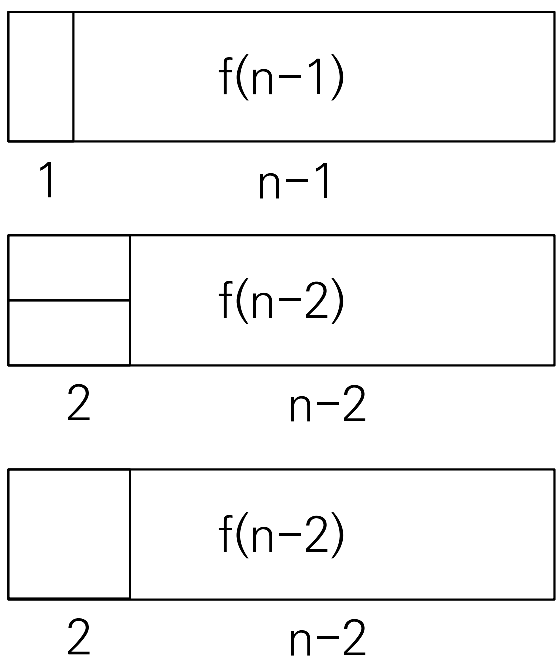


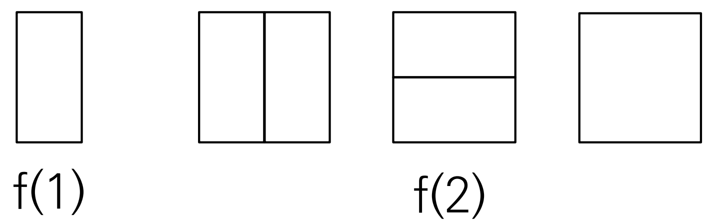
f(n) =

	1	2	3	4	5	6	7
d	1	2					

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







f(n) =

	1	2	3	4	5	6	7
d	1	3					

여러 개의 값 활용

- 1 2 3의 부분 집합으로 만든 중복 순열의 합이 N이 되는 경우의 수를 구하시오.

입력

4

4를 만들 수 있는 경우

1 + 1 + 1 + 1
2 + 1 + 1
1 + 2 + 1
3 + 1
1 + 1 + 2
2 + 2
1 + 3

4는 어떤 수에 1, 2, 3을 더해 만들 수 있으므로,

3 + 1 = 4
2 + 2 = 4
1 + 3 = 4

(4−1=3을 만드는 경우의 수)
+ (4−2=2를 만드는 경우의 수)
+ (4−3=1을 만드는 경우의 수)

▪ N이 10인 경우

n	1	2	3	4	5	6	7	8	9	10
d	1	2	4							

+

f(n) =

초기값

1	2	3
1	1+1 2	1+1+1 2+1 1+2 3

최장 증가 부분 수열

- 어떤 수열에서 숫자가 증가하는 순서로 숫자를 골라 만든 부분 수열 중, 숫자의 개수가 가장 많은 것을 최장 증가 부분 수열이라고 한다. 주어진 수열에서 최장 증가수열의 길이를 구하시오.

$\boxed{3\ 1\ 2\ 7\ 6\ 4\ 5} \longrightarrow \boxed{1\ 2\ 4\ 5}$

i	0	1	2	3	4	5	6
a	3	1	2	7	6	4	5
증가 수열의 길이 d	1	1	2	3	3	3	4

- a[i]까지의 증가 수열의 길이 d[i]
  - 최소값 1로 초기화 (이후 숫자가 계속 작아져도 최소한 1이 된다.)

i	0	1	2	3	4	5	6
a	3	1	2	7	6	4	5
증가 수열의 길이 d	1	1	1	1	1	1	1

0<=j<i인 j에 대해 a[j]<a[i]를 만족하는 경우 중, d[j]가 가장 큰 값을 골라 1을 더한다.

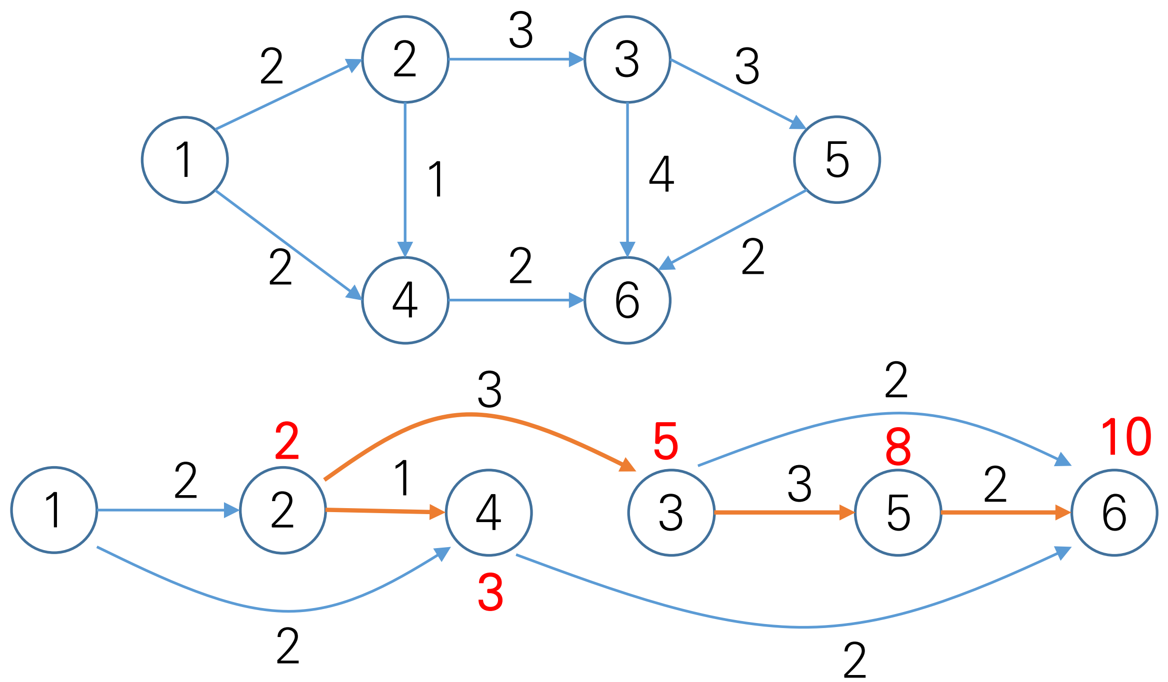
$$\boxed{\mathrm{d[i]} = \left(\max_{0 \leqq j < \mathrm{i}} \mathrm{d}[j] \right) + 1 ,\ \mathrm{a[j]} < \mathrm{a[i]}\text{인 경우}}$$

a	3	1	2	7	6	4	5
증가 수열의 길이 d	1						

- n개의 수에서 최장 증가 수열의 길이 : $\max_{0 \leqq \mathrm{i} < n} \mathrm{d}[i]$

최장거리 구하기

- 조건 : 사이클이 없는 방향성 그래프(DAG)
- 시작 노드 1, 도착 노드 6.
- 위상 정렬을 사용해 거리 계산 순서를 결정한다.

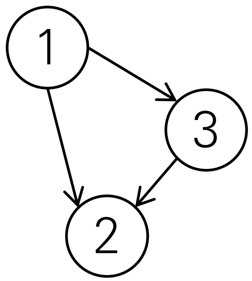


- 각 노드까지의 최대 거리 dis[], 인접행렬 adj[][].
  - n으로 진입하는 모든 노드 i에 대해, dis[i]+adj[i][n]이 최대인 값을 dis[n]으로 정함.

$$dis[n] = \max_{1 \le i \le V}(dis\,[i] + adj[i][n])\,, adj[i][n]! = 0 \text{인 노드 } i\text{에 대해}$$

```
max = 0
for i : 1->V
    if( adj[i][n] != 0 )
        if( max < dis[i] + adj[i][n] )
            max = dis[i] + adj[i][n]
dis[n] = max
```

✓위상 정렬




```
Sort( )
    for i : 0 -> N          // 진입차수가 0이면 enQ
        if( ind[i] == 0 )
            enQ(i)
    while(is_not_emptyQ())
        n = deQ( )
        visit(n)                    // 노드 n에서 해야할 일
        for i : 1 -> N
            if( adj[n][i] == 1 )
                ind[i]--      // n의 인접노드 진입차수 감소
                if( ind[i] == 0 )  // 진입차수가 0이면 enQ
                    enQ(i)
```

adj[] : 인접행렬
ind[] : 진입 차수

오른쪽과 아래로 이동

- 각 칸에서는 오른쪽이나 아래로만 이동할 수 있다.
- 출발은 맨 왼쪽 위, 도착은 맨 오른쪽 아래이다.
- 출발부터 도착까지 지나는 각 칸의 합계가 최대가 되도록 움직였을 때, 합계를 계산하라.
- 각 칸은 1에서 9사이의 숫자.

출발

	0	1	2	3	4
0	5	6	1	5	5
1	2	7	6	1	8
2	7	8	2	6	6
3	9	5	1	8	1
4	1	1	3	8	7

도착

- 각 칸까지의 최대 합계 d.
  - 각 칸에는 위와 왼쪽에서만 진입 가능.
  - 윗쪽 칸과 왼쪽 칸 중 합계가 큰 곳 + 현재 칸의 값.

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

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

1

1 1

1 2 1

1 3 3 1

1

1 1

1 2 1

1 3 3 1

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

▪ DP 배열

C[n][k] = 1, k = 0 or n == k
C[n][k] = C[n−1][k−1] + C[n−1][k]

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

계산하려는 값의 오른쪽 부분은 없어도 됨.

```
for i : 0 -> n
    for j : 0 -> i
```

```
for i : 0 -> n
    for j : 0 -> min(i, k)
```

중복 조합의 합

- 1 2 3으로 만든 중복 조합의 합이 N이 되는 경우의 수를 구하시오.

N 입력

4

4를 만들 수 있는 경우

1 + 1 + 1 + 1
1 + 1 + 2
1 + 3
2 + 2

1만 사용해 만들 수 있는 합 1, 2, 3, 4, …
2 : 2, 4, 6, 8…
3 : 3, 6, 9, 12, …
1, 2 : 3, 4, 5, …
1, 3 : 4, 5, 6, …
2, 3 : 5, 7, 8, …

- N이 10인 경우

	0	1	2	3	4	5	6	7	8	9	10	j
∅	0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	1	1	1	1	1	
2	1	1	2	2	3	3	4	4	5	5	6	
3	1	1	2	3	4	5	7	8	10	12	14	
i												

1만 사용, 1과 2 사용, 2만 사용

```
for i : 1 -> 3
    for j : 1 -> 10
        if j >= i
            d[i][j] = d[i-1][j] + d[i][j-i]
        else
            d[i][j] = d[i-1][j]
```

⇨

```
for i : 1 -> 3
    for j : i -> 10
        d[j] += d[j-i]
```

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

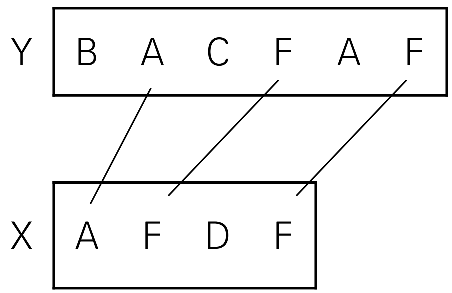
1 2 4 2 5 6 8 4

⇨ 2 5 4

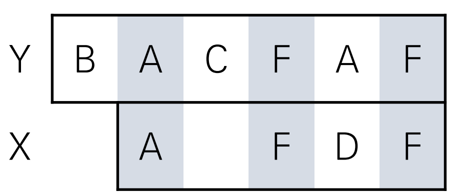
✓ 글자의 간격을 조절해 아래 위 글자를 일치시켜 본다.

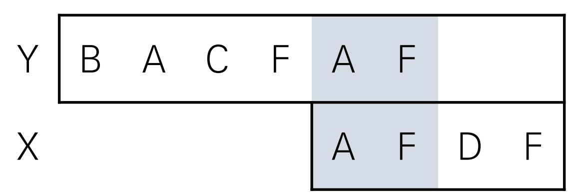
▪ LCS 적용

- 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.

	1	2	3	4	5	6
Y	B	A	C	F	A	F
X	A	F	D	F		

	j-1	j				
Y	B	A	C	F	A	F
X		A		F	D	F
	i-1	i				

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

			j-1	j			
Y	B	A	C	F	A	F
X	A	F	D	F		
		i-1	i			

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]

	Ø	B	A	C	F	A	F
Ø	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

- True/False를 사용.
  - { 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	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

$$m[i][j] = m[i-1][j] \mid\mid m[i-1][j-a_i],\ i>0,\ j>0$$
$$m[i][j] = 1,\ j=0$$
$$m[i][j] = 0,\ i=0,\ j>0$$

✓부분 집합의 합이 5가되는 경우의 수는?

- {a1, a3} {a2, a3} 모두 {1, 2}이다. 이 둘을 다른 경우로 보는 경우.

	0	1	2	3	4	5	j
$\emptyset$	1	0	0	0	0	0	
1	1	1	0	0	0	0	
1	1	2	1	0	0	0	
2	1	2	2	2	1	0	
2	1	2	3	4	3	2	
1	1	3	5	7	7	5	
a_i							

m[i][j] = 1, j=0
m[i][j] = 0, i=0, j〉0
m[i][j] = m[i−1][j] + m[i−1][j−a_i], i〉0, j〉0, ai〈=j
m[i][j] = m[i−1][j], i〉0, j〉0, ai〉j

✓부분 집합의 합이 5가 되는 경우의 수는?

- {a1, a3} {a2, a3} 모두 {1, 2}이다. 이 둘을 같은 경우로 보는 경우.
- 원소를 정렬한다.

	0	1	2	3	4	5
Φ	1	0	0	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0
1	1	1	1	1	0	0
2	1	1	2	2	1	1
2	1	1	2	2	2	2

j

a_i

$$m[i][j] = 1, \ j=0$$
$$m[i][j] = 0, \ i=0, \ j>0$$
$$m[i][j] = m[i-1][j], \ a_i > j$$
$$m[i][j] = m[i-1][j - a_i], \ a_i == a_{i-1}, \ ai<=j, \ i>0, \ j>0$$
$$m[i][j] = m[i-1][j] + m[i-1][j-a_i], \ a_i \ != a_{i-1}, \ i>0, \ j>0, \ ai<=j$$

연속한 1이 없는 이진수

- 1로 시작하는 n자리 이진수에서 연속한 1이 없는 경우의 수를 구하시오.
- n=4인 경우 1000, 1001, 1010이 가능.

i번째 자리	1	2	3	4	5	6
0인 경우	0	1	1	2		
1인 경우	1	0	1	1		
경우의 수	1	1	2	3		

- i번 째 자리가 0인 경우와 1인 경우를 나눠서 생각한다.
  - 0인 경우는 i-1이 0과 1인 경우 모두 가능하다.
  - 1인 경우는 i-1이 0인 경우만 가능하다.
  - 첫 번째 자리는 0인 경우는 불가 하므로 0, 1인 경우는 1가지 경우이다.

```
d[1][0] = 0
d[1][1] = 1
for i : 2 -> n
     d[i][0] = d[i-1][0] + d[i-1][1]
     d[i][1] = d[i-1][0]
dn = d[n][0] + d[n][1]
```

연속한 0이 없는 이진수

- 1로 시작하는 n자리 이진수에서 0이 3번 이상 연속하지 않는 경우의 수를 구하시오.
- n=4인 경우 1001, 1010, 1011, 1100, 1101, 1110, 1111.

i번째 자리	1	2	3	4	5	6
첫 번째 0인 경우	0	1	1	2		
두 번째 0인 경우	0	0	1	1		
1인 경우	1	1	2	4		
경우의 수	1	1	3	7		

- i번째 자리는 세 가지 경우가 가능 (i〉2인 경우)
  - 첫 번째 0인 경우 : d[i][0] (이전 자리가 1인 경우)
  - 두 번째 0인 경우: d[i][1] (이전 자리가 첫 번째 0인 경우)
  - 1인 경우 : d[i][2] (이전 자리가 0이거나 1인 경우)

```
d[1][0] = 0
d[1][1] = 0
d[1][2] = 1
for i : 2 -〉 n
        d[i][0] = d[i-1][2]
        d[i][1] = d[i-1][0]
        d[i][2] = d[i-1][0] + d[i-1][1] + d[i-1][2]
dn = d[n][0] + d[n][1] + d[n][2]
```

3가지 색으로 칠하기

- 축제 행렬이 움직이는 길가의 집을 3가지 색으로 칠하려고 한다.
  - 이웃한 집은 서로 다른 색을 칠한다.
  - 각 집마다 색상 별로 소요 비용이 다르다.
  - 첫번째 집부터 순서대로 칠해 나가야 한다.
  - 총 n개의 집을 칠하려고 할 때 전체를 칠하는 최소 비용은?
  - 3개의 집을 칠하는 경우
    - 첫번째 집부터 각 색으로 칠할 때의 비용이 주어진다.

m	1	2	3	색
1	5	7	6	
2	2	4	8	
3	5	2	3	
집				

▪ i번 집까지 칠하는 비용

- i번 집을 각 색으로 칠하는 비용을 계산한다.
- 단, i번 집을 칠할 색상은 i-1에서 사용하지 않은 색이어야 한다.
- 예를 들어 i번 집을 1번으로 칠하는 경우, i-1은 2 또는 3번으로 칠해져 있어야 한다.

m	1	2	3	색
1	5	7	6	
2	2	4	8	
3	5	2	3	
집				

d	1	2	3	색
1	5	7	6	
2				
3				
집				

```
d[1][1] = m[1][1]
d[1][2] = m[1][2]
d[1][3] = m[1][3]
for i : 2 -> n
      d[i][1] = min(d[i-1][2], d[i-1][3]) + m[i][1]
      d[i][2] = min(d[i-1][1], d[i-1][3]) + m[i][2]
      d[i][3] = min(d[i-1][1], d[i-1][2]) + m[i][3]
dn = min(d[n][1], d[n][2], d[n][3])
```

건너뛰기

- 2차원 배열에 표시된 숫자만큼 오른쪽이나 아래로 이동해 도착지에 도달하는 경로의 수를 찾는 문제.
  - 30이내의 자연수로 채워진 NxM 배열.
  - 맨 왼쪽 위 출발, 맨 오른쪽 아래 도착.

2	3	1	3	1
2	1	1	1	1
3	1	2	1	3
3	1	2	3	2
3	1	2	3	2

▪ i, j 칸에 도착하는 경우의 수 d[i][j]

- 왼쪽의 모든 칸 k에 대해 i, j로 올 수 있으면 d[i][j] += d[i][k]
- 위쪽의 모든 칸 k에 대해 i, j로 올 수 있는면 d[i][j] += d[k][j]

2	3	1	3	1
2	1	1	1	1
3	1	2	1	3
3	1	2	3	2
3	1	2	3	2

d j

	1				
i					

```
for i : 1 -> N
        for j : 1 -> M
                for k : 1 -> j-1
                {
                        if(A[i][k] == j-k)
                                D[i][j] += D[i][k];
                }
                for k : 1 -> i-1
                {
                        if(A[k][j] == i-k)
                                D[i][j] += D[k][j];
                }
        }
}
```

거스름 동전의 최소 개수

- 동전의 단위가 1, 3, 5인 나라가 있다. 9원의 거스름돈을 최소한의 동전을 사용해 거슬러 줄 때, 동전의 개수를 구하라.
  - d[i][j] : i 동전까지 고려해 j원을 만들 때 동전의 최소 개수
  - a[i] = {1, 3, 5} : 동전의 액면가
  - 같은 금액의 동전은 충분히 있다.
  - 예를 들어 4원을 만들려면, 1원만 사용하거나, 3원까지 사용해 4원을 만들 수 있다. 이 중 개수가 적은 쪽을 택한다.
    - d[(3)][4−3] : 3원까지 사용하고, 1원을 만들 때 동전의 최소 개수.
    - d[(3)][4] = min(d[(3)][4−3] + 1, d[(1)][4])

- i 동전까지 고려해 j원을 만들 때 동전의 최소 개수

	d	0	1	2	3	4	5	6	7	8	9	
1원	0	0	1	2	3	4	5	6	7	8	9	j
3원	1	0										
5원	2	0										
	i											

```
// 1원 짜리만 고려한 경우
d[0][j] = j

// 1원보다 큰 동전까지 고려한 경우
d[i][j] = d[i-1][j]   // j < a[i] (거스름이 i동전보다 작은 경우)
d[i][j] = min (d[i][j-a[i]] + 1, d[i-1][j])
```

0-1 배낭 짐싸기

- 크기가 정해진 박스가 가득 찰 때까지 물건을 담을 수 있는 해피 박스 이벤트가 열린다고 한다. 물건의 크기와 가치가 주어질 때 박스에 들어간 물건의 최대 가치는 얼마인가? 각 물건은 1개씩 있다.
  - 박스의 크기 10. 물건 크기의 합이 상자 크기를 넘지 않으면 담을 수 있다.
  - 준비된 물건의 크기와 가치

물건	크기	가치
1	6	12
2	5	10
3	5	15
4	4	6

- 박스의 크기를 늘려가며 담아 본다.
  - 1번 물건만 고려, 2번까지 고려, 3번까지 고려하는 식으로 생각해본다.
  - i번 물건을 넣을 수 있는 상자의 용량을 생각한다.

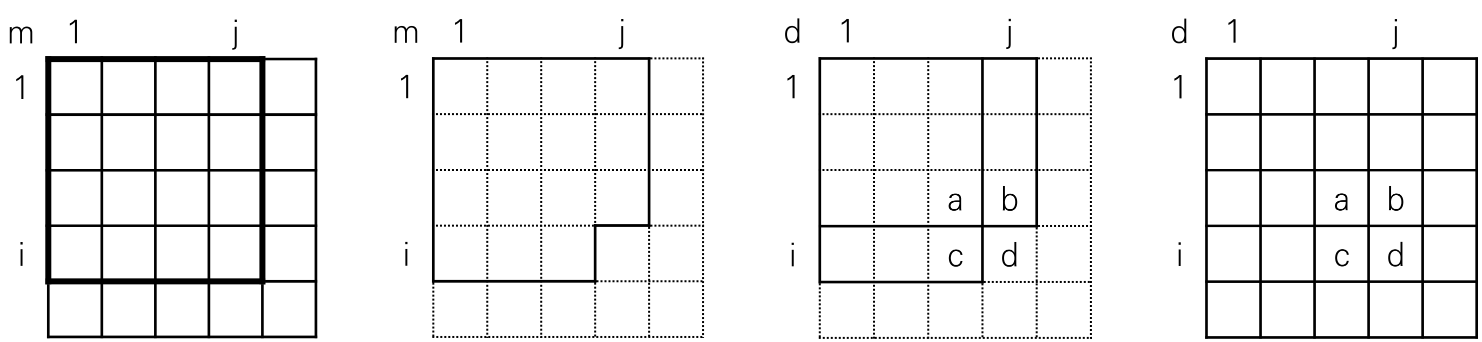
d[i][j] : 상자의 크기가 j이고 i번 물건까지 고려한 경우
d[i][j] = d[i-1][j] , j 〈 w[i] (상자보다 물건이 크면)
d[i][j] = max(d[i-1][j-w[i]] + v[i], d[i-1][j])

물건	크기 w	가치 v
1	6	12
2	5	10
3	5	15
4	4	6

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	12	12	12	12	12
2	0	0	0	0	0	10	12				
3	0	0	0	0	0						
4	0	0	0	0	6						

배열 원소의 합 구하기

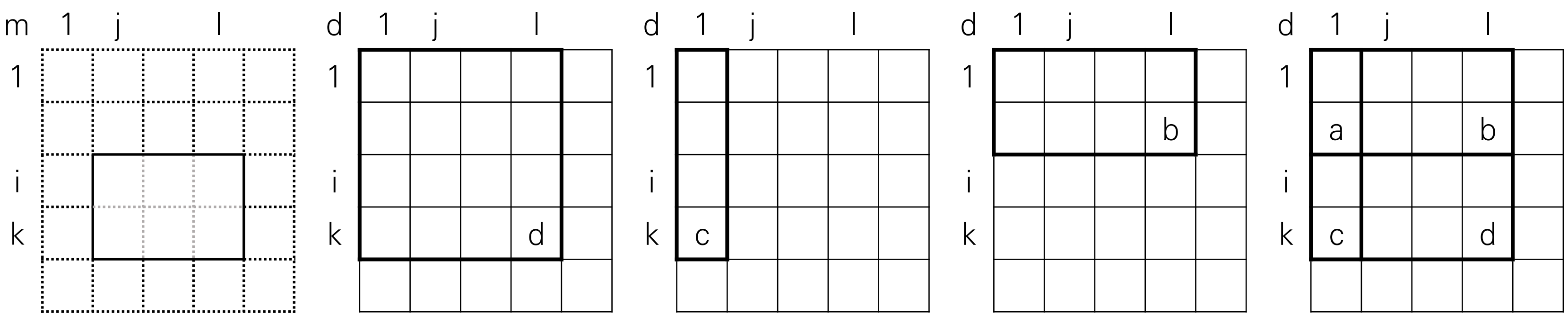
- 왼쪽 위 부터 i, j까지의 합 구하기.
  - 모든 i, j에 대해 구하는 경우.
  - d[i][j]에 i, j까지의 합을 저장한다.
  - d[i][j]는 i, j칸을 제외한 나머지 칸의 합에 i, j칸을 더한다.
    - $d[i][j] = d[i-1][j] + d[1][j-1] - d[i-1][j-1] + m[i][j]$



i, j를 제외한 부분의 합은 두 사각형 내부의 합을 더한 후 겹치는 부분을 뺀다.
b + c − a

▪2차원 배열 내부 사각 영역의 합 구하기.

- 왼쪽 위 i, j. 오른쪽 아래 k, l.
- 앞에서 구한 d[][]를 활용한다.
  - 1, 1 부터 k, l 까지 면적이 d.
  - 1, 1 부터 k, j−1 까지 면적이 c.
  - 1, 1 부터 i−1, l 까지 면적이 b.
  - 1, 1 부터 i−1, j−1 까지 면적이 a.
  - r = d − c − b + a



r = d[k][l] − d[k][j−1] − d[i−1][l] + d[i−1][j−1]

연속 행렬 곱셈

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

$$\underset{\mathbf{2}\times 2}{\begin{bmatrix} a1 & a2 \\ a3 & a4 \end{bmatrix}} \times \underset{2\times \mathbf{3}}{\begin{bmatrix} b1 & b2 & b3 \\ b4 & b5 & b6 \end{bmatrix}} = \underset{\mathbf{2}\times \mathbf{3}}{\begin{bmatrix} a1b1+a2b4 & a1b2+a2b5 & a1b3+a2b6 \\ a3b1+a4b4 & a3b2+a4b5 & a3b3+a4b6 \end{bmatrix}}$$

    - 곱셈 연산의 횟수 : 2 * 2 * 3 = 12
  - 곱하는 행렬 크기 저장
    - 중복은 제거
    - A1의 크기 p[0]xp[1], A2의 크기 p[1]*p[2]
    - Ai의 크기 p[i−1]xp[i]

	0	1	2
p	2	2	3

■ 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] x p[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

D	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

j (columns), i (rows)

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

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

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])
```

카드 게임

- 두 사람이 카드를 집는 게임.
  - 숫자 카드를 한 줄로 늘어 놓고 교대로 가져간다.
  - 순서마다 양끝의 카드 중 하나를 선택해 가질 수 있음.
  - 한 사람이 가져갈 수 있는 카드의 숫자 합계 최대값은?
  - 단, 두 사람은 같은 전략을 써서 카드를 가져간다.

5	3	2	1	8	4
i					j

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

TSP

- n개의 도시.
  - 1번 도시에서 출발해 나머지 도시를 1번씩 거쳐 1번으로 되돌아 올 때 최소 비용을 구하는 문제.
  - 도시 i에서 다른 도시 j로 갈 수 있고, 각각의 비용은 m[i][j].
  - 모든 도시를 원소로 갖는 집합을 S, 그 부분집합을 s라고 정의한다.
  - S의 원소 i에서 경유지 없이 1로 가는 최소 비용. 경유지는 공집합.
    - $d[i][\emptyset] = m[i][1]$, $1 < i <= n$
  - i에서 2를 거쳐 1로 가는 최소 비용.
    - $d[i][\{2\}] = m[i][2] + d[2][\emptyset]$, $1 < i <= n$ && $i != 2$
  - i에서 2와 3을 거쳐 1로 가는 최소 비용
    - $d[i][\{2, 3\}] = \min(m[i][2]+d[2][\{3\}], m[i][3]+d[3][\{2\}])$, $1 < i <= n$, $i \notin \{2,3\}$

- i에서 2,3,4를 거쳐 1로 가는 최소 비용. $i \notin \{2,3,4\}$
  - d[i][{2,3,4}] = min(m[i][2] + d[2][{3,4}],
    m[i][3] + d[3][{2,4}],
    m[i][4] + d[4][{2,3}])
- 1에서 모든 도시를 거쳐 1로 되돌아 오는 최소 비용
  - d[1][S−{1}]
- 경유지인 부분 집합의 원소를 1개 부터 n−1개 까지 늘림.