

# 기본 알고리즘

## 제4장



2017-Fall

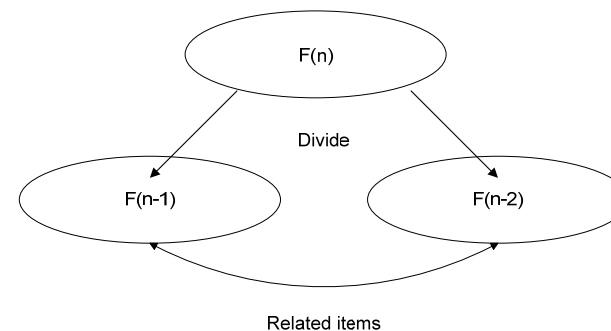
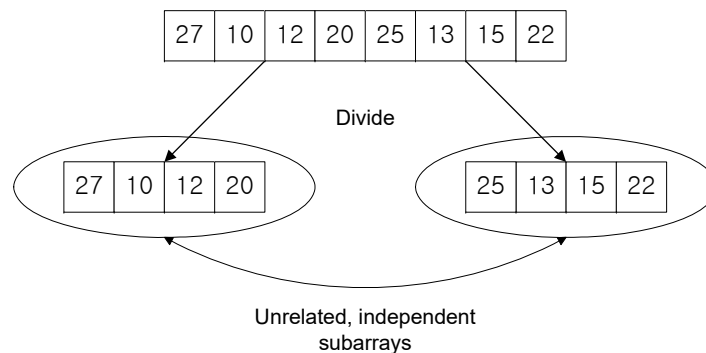
국민대학교 컴퓨터공학부 최준수

# 동적계획법

- 동적계획법

- 분할정복기법 (Divide-and-Conquer)과 유사

- 문제를 여러 개의 subproblem 으로 나누고, 각 subproblem을 해결한 후, 각 subproblem의 해답을 이용하여 원래 문제의 해답을 계산함
    - 그러나, 각 subproblem 이 독립적이지 않고, 서로 연관되어 있는 경우에는 매우 많은 반복연산이 이루어지고, 이로 하여금 많은 수행시간이 필요함.



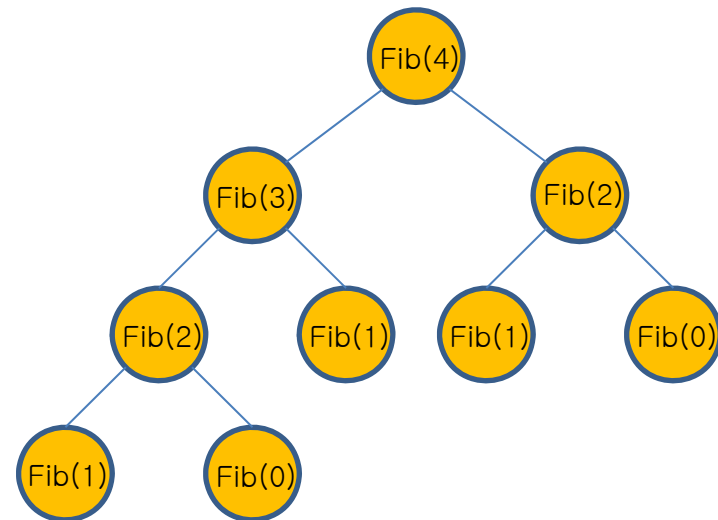
# 동적계획법 (2)

- 예
  - Fibonacci 수 계산

$$Fib(n) = \begin{cases} 0 & n = 0 \quad (\text{base case}) \\ 1 & n = 1 \quad (\text{base case}) \\ Fib(n-1) + Fib(n-2) & n > 1 \quad (\text{recursive step}) \end{cases}$$

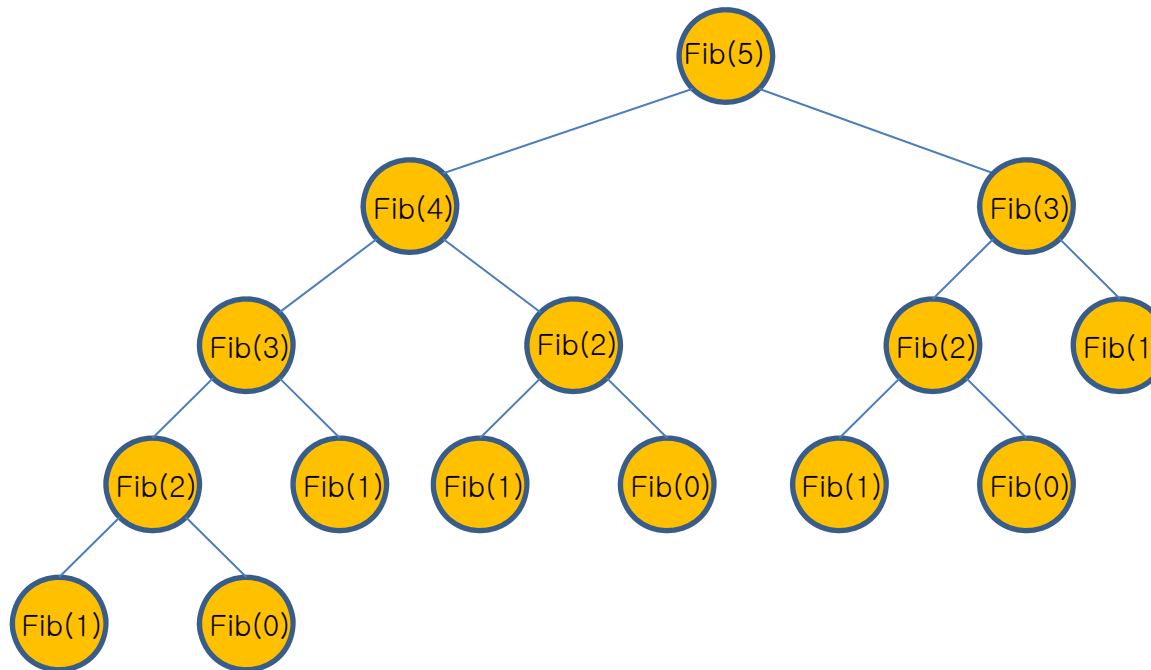
```
int fib(int n)
{
    if (n <= 1) /* base case */
        return n;
    else /* recursive step */
        return fib(n-1)+fib(n-2);
}

int main()
{
    fib(4);
}
```



# 동적계획법 (3)

- 예
  - Fibonacci 수 계산



# 동적계획법 (3)

- Fib(100) 계산시간은?

$$Fib(n) = \begin{cases} 0 & n = 0 \quad (\text{base case}) \\ 1 & n = 1 \quad (\text{base case}) \\ Fib(n-1) + Fib(n-2) & n > 1 \quad (\text{recursive step}) \end{cases}$$

- T(n) : Fib(n)을 계산할 때, 덧셈의 횟수

$$T(n) = \begin{cases} 0 & n = 0, 1 \quad (\text{base case}) \\ T(n-1) + T(n-2) + 1 & n > 1 \quad (\text{recursive step}) \end{cases}$$

- T(n) : Fib(n)을 계산할 때, 시간 (sec)

$$T(n) = \begin{cases} 10^{-9} & n = 0, 1 \quad (\text{base case}) \\ T(n-1) + T(n-2) + 10^{-9} & n > 1 \quad (\text{recursive step}) \end{cases}$$

# 동적계획법 (3)

- Fib(100) 계산시간은?

```
#include <stdio.h>
#include <time.h>

int fib(int n)
{
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);
}

void main(void)
{
    int i, value;
    time_t startTime, finishTime;
    clock_t startClock, finishClock;

    for(i=0; i<=100; i++)
    {
        time(&startTime); startClock = clock();
        value = fib(i);
        time(&finishTime); finishClock = clock();

        printf("Fib(%d): %d\n", i, value);
        printf("elapsed time: %.2lf sec\n",
            (double)(finishClock - startClock)/CLOCKS_PER_SEC);
    }
}
```

# 동적계획법 (3)

- Fib(100) 계산시간은?

```
C:\Windows\system32\cmd.exe
Fib(20): 6765
elapsed time: 0.00 sec
Fib(21): 10946
elapsed time: 0.00 sec
Fib(22): 17711
elapsed time: 0.00 sec
Fib(23): 28657
elapsed time: 0.00 sec
Fib(24): 46368
elapsed time: 0.01 sec
Fib(25): 75025
elapsed time: 0.01 sec
Fib(26): 121393
elapsed time: 0.01 sec
Fib(27): 196418
elapsed time: 0.02 sec
Fib(28): 317811
elapsed time: 0.03 sec
Fib(29): 514229
elapsed time: 0.05 sec
Fib(30): 832040
elapsed time: 0.09 sec
Fib(31): 1346269
```

```
C:\Windows\system32\cmd.exe
Fib(33): 3524578
elapsed time: 0.32 sec
Fib(34): 5702887
elapsed time: 0.52 sec
Fib(35): 9227465
elapsed time: 0.82 sec
Fib(36): 14930352
elapsed time: 1.37 sec
Fib(37): 24157817
elapsed time: 2.28 sec
Fib(38): 39088169
elapsed time: 3.54 sec
Fib(39): 63245986
elapsed time: 5.55 sec
Fib(40): 102334155
elapsed time: 9.49 sec
Fib(41): 165580141
elapsed time: 15.08 sec
Fib(42): 267914296
elapsed time: 24.33 sec
Fib(43): 433494437
elapsed time: 44.05 sec
```

# 동적계획법 (3)

- Fib(100) 계산시간은?

n	근사실행시간 (sec)	참고	n	근사실행시간 (year)	참고
35	1	Fib(1)	71	1	Fib(1)
36	1	Fib(2)	72	1	Fib(2)
37	2	Fib(3)	73	2	Fib(3)
38	3	Fib(4)	74	3	Fib(4)
39	5	Fib(5)	75	5	Fib(5)
72	39,088,169	Fib(38)	100	832,040	Fib(30)

기준

기준

1년 = 31,536,000초

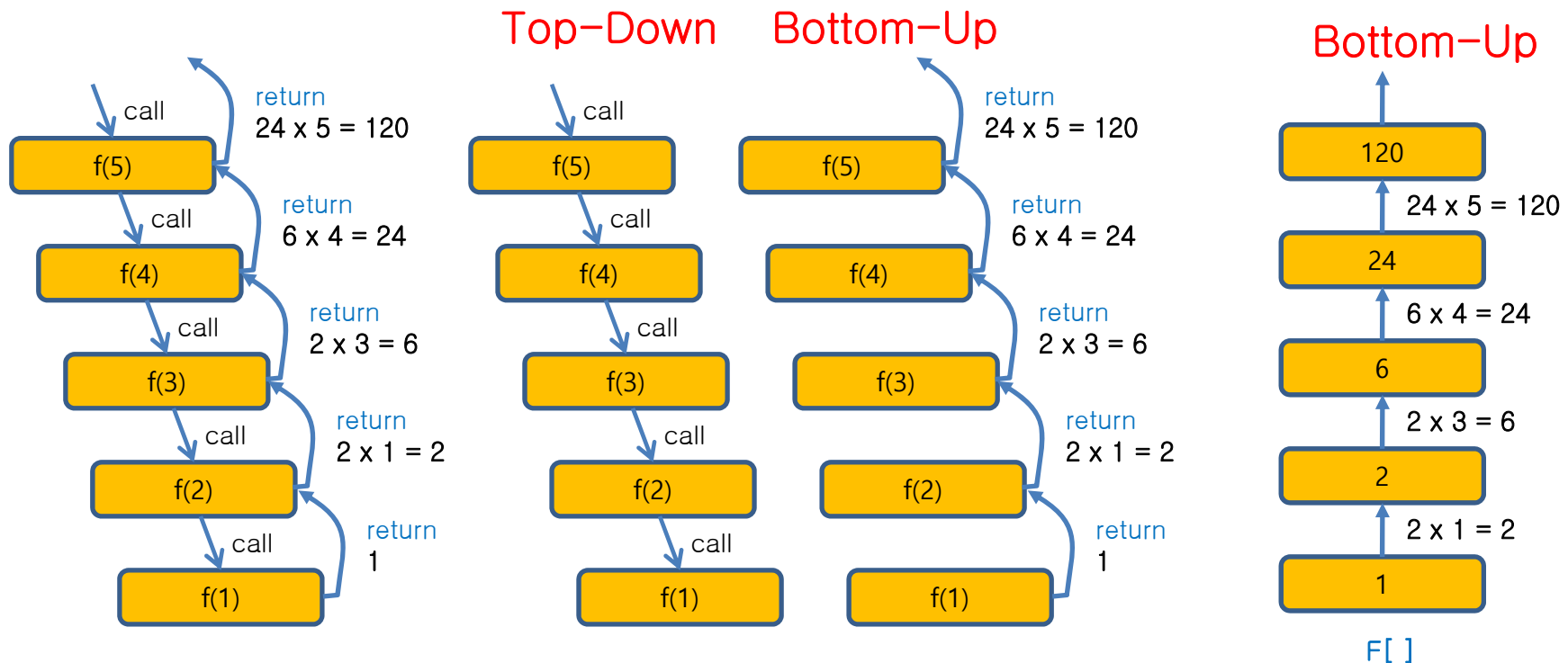


# 동적계획법 (4)

- 문제해결 방법
  - 반복연산의 제거 (Bottom-Up Approach)
    - 작은 문제부터 시작
    - 작은 문제를 해결한 후, 그 해답을 테이블에 저장
    - 큰 문제를 해결하면서 작은 문제를 해결할 필요가 있는 경우에는, 테이블에 저장된 작은 문제의 해답을 사용

# 동적계획법 (5)

- 문제해결 방법
  - Bottom-Up Approach



# 동적계획법 (6)

- 문제해결 방법
  - Bottom-Up Approach

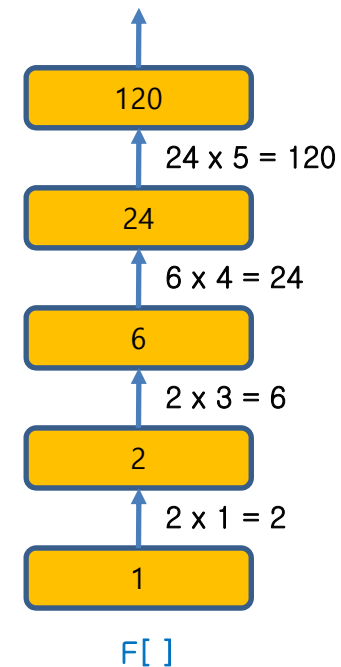
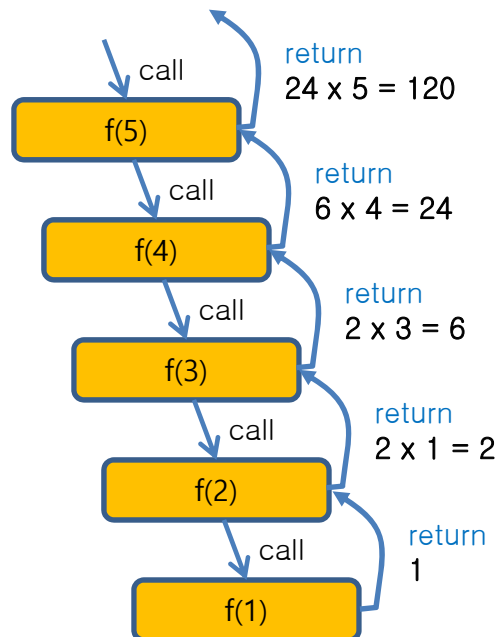
```
int f(int n)
{
    if (n <= 1) /* base case */
        return 1;
    else
        return n*f(n-1);
}
```

```
int f(int n)
{
    int i, F[100];

    F[0] = 1;          /* base case */

    for(i=1; i<=n; i++)
        F[i] = i * F[i-1];

    return F[n];
}
```



# 동적계획법 (7)

- 동적계획법으로 해결하는 문제의 형태
  - 최적화문제 (Optimization Problem)
    - 문제를 해결하는 해답이 여러 개 있지만, 그 중에서도 특정한 조건이 **최대**가 되는 혹은 **최소**가 되는 해답을 구하는 문제.
    - 예 : 동전교환문제
      - 동전의 종류에 1원, 5원, 10원, 25원, 50원이 있을 때, 거스름돈 137원을 동전으로 교환해 주고자 한다.
      - 거스름돈을 동전으로 교환하는 방법에는 여러 가지가 있다.
      - 그 방법 중에서, 동전의 개수가 **최소**가 되도록 교환하는 동전 그룹을 계산하시오.
      - 해답 : 동전 6개, 동전 그룹 : [50, 50, 25, 10, 1, 1]
    - 최적화 문제의 해답
      - 최적값 (위의 예에서는 동전의 최소개수, 6개)
      - 최적값을 가질 때의 해답 (위의 예에서는 동전의 그룹)
      - 어떤 경우에는 최적값만 필요한 경우도 있다.

# 동적계획법 (8)

- 동적계획법으로 문제를 해결하는 시나리오

(단계 1) 최적의 해답의 구조를 분석.

- 최적값 혹은 최적해답의 어떻게 구할 수 있는지를 분석함.
  - 이때, 많이 사용하는 방법이 “**working backward**” 기법이다.
  - “Working backward” 기법을 이용하면, 문제의 해답을 top-down 방법으로 분석할 수 있다.

(단계 2) 최적의 해답의 최적값을 **재귀식**으로 정의.

- 위 단계 1에서 “working backward” 기법으로 최적값을 top-down 방식으로 분석하면, 최적값을 재귀식으로 정의할 수 있다.

(단계 3) 상향식으로 최적값을 계산.

- 위 단계 2에서 재귀식으로 정의된 최적값을 상향식(bottom-up)으로 계산한다.
- 상향식으로 계산한다는 의미는 가장 작은 데이터에 대한 해답부터 계산하여 점차 큰 데이터의 해답을 계산해 나간다는 것을 나타낸다.
- 이때, 이미 계산한 작은 데이터에 대한 해답은 테이블에 저장해 두고, 필요한 경우에는 다시 계산하지 않고 테이블에 저장된 해답을 바로 사용하도록 한다.

# 동적계획법 (9)

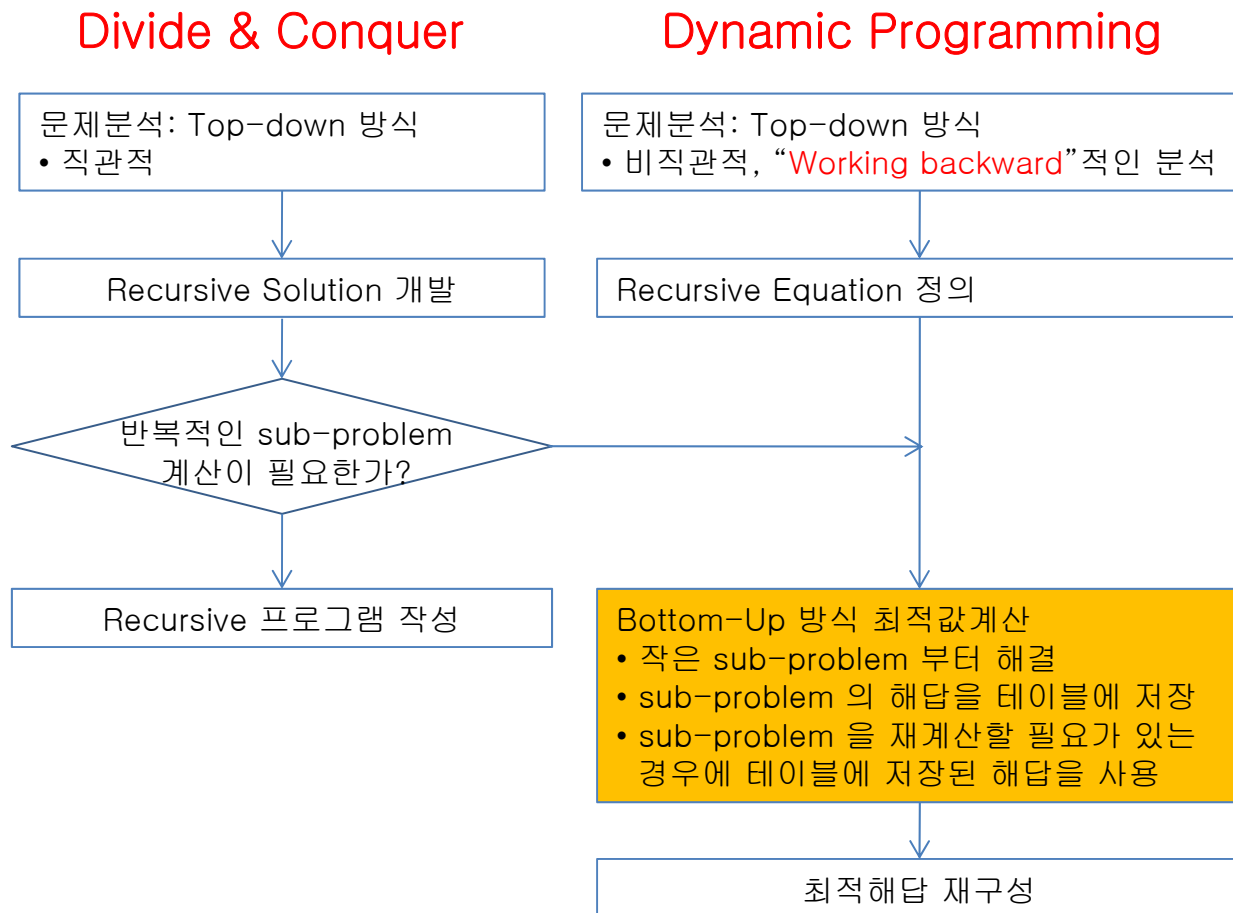
- 동적계획법으로 문제를 해결하는 방법

- (단계 4) 최적의 해답계산

- 위 단계 3에서는 최적값을 계산하는 과정이고, 단계4에서는 최적해답을 계산하는 단계이다.
- 예를 들어, 동전교환문제에서 단계3에서는 최소동전의 개수를 계산하고, 단계4에서 최소개수의 동전그룹을 계산한다.
- 단계 4에서 최적의 해답을 계산하기 위해서는 단계 3에서 최적값을 계산하는 과정에서 발생하는 정보를 이용한다. 이를 위해서는 단계 3에서 최적값을 계산하는 테이블 이외에 이 정보를 저장하는 테이블을 따로 만들고, 정보가 만들어질 때마다 테이블에 저장해 둔다. 이 테이블에 저장된 정보를 이용하여 단계 4에서 최적의 해답을 계산하게 된다.
- 단계 4에서 최적의 해답은 단계3에서 만들어 둔 정보를 이용하여 계산하기 때문에, 최적해답을 재구성(reconstruct), 혹은 구성(construct)한다라고도 한다.

# 동적계획법 (10)

- 분할정복기법과 동적계획법



# 1차 동적계획법

- 1차 동적계획법
  - 재귀식에서 1개의 변수가 필요한 문제
  - 1차원 배열로 동적계획법 구현
  - 예
    - Fibonacci 수 계산
    - 동전교환 (coin exchange) 문제
    - 기타



# Fibonacci 수 계산

## – Fibonacci 수 계산

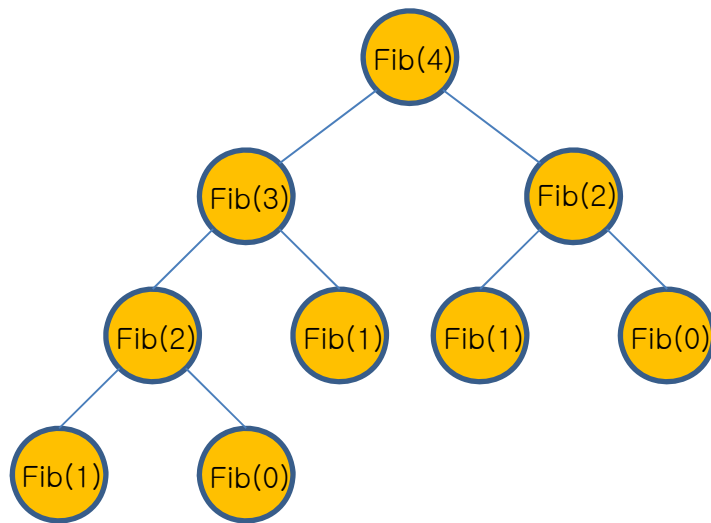
```
int fib(int n)
{
    if (n <= 1) /* base case */
        return n;
    else /* recursive step */
        return fib(n-1)+fib(n-2);
}
```

```
int fib(int n)
{
    int i, Fib[100];

    Fib[0] = 0;
    Fib[1] = 1; /* base case */

    for(i=2; i<=n; i++)
        Fib[i] = Fib[i-1] + Fib[i-2];

    return F[n];
}
```

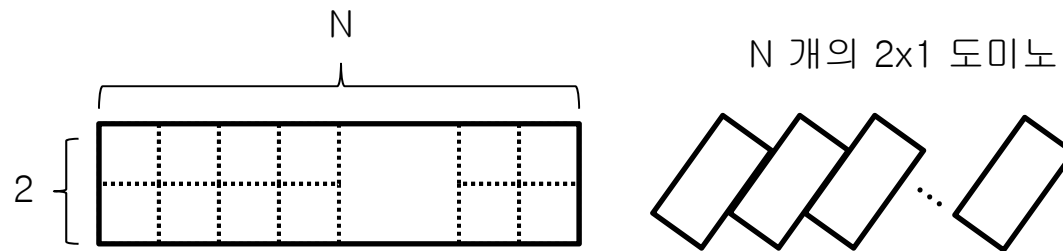


Bottom-Up

Fib[ ]

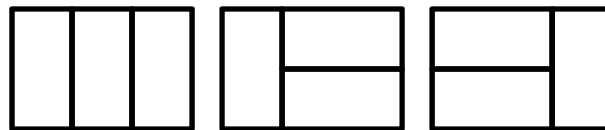
# 2xN 직사각형 타일 채우기 문제

- 2xN 직사각형 타일 채우기 문제
  - 크기가 2xN 인 직사각형이 주어졌을 때, 이 직사각형을 크기가 2x1 인 도미노 타일로 채우는 가지수를 계산하시오.



– 예

- 2x3 직사각형을 채우는 방법 : 3가지



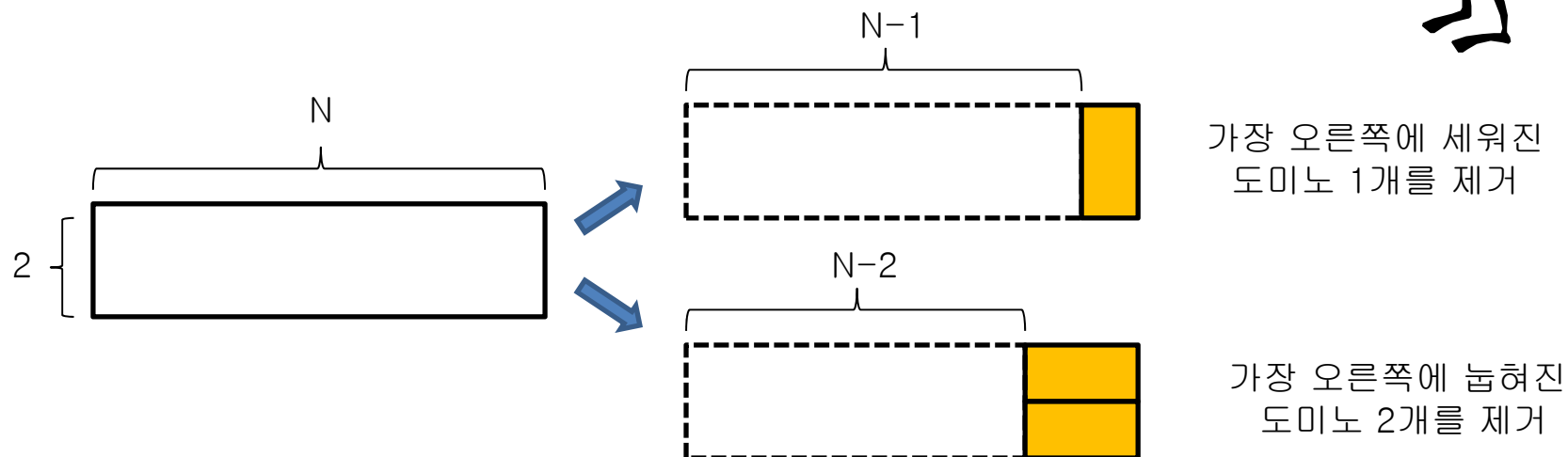
# 2xN 직사각형 타일 채우기 문제 (2)

- 단계 1:

- 도미노 타일로 채우는 방법의 구조 분석

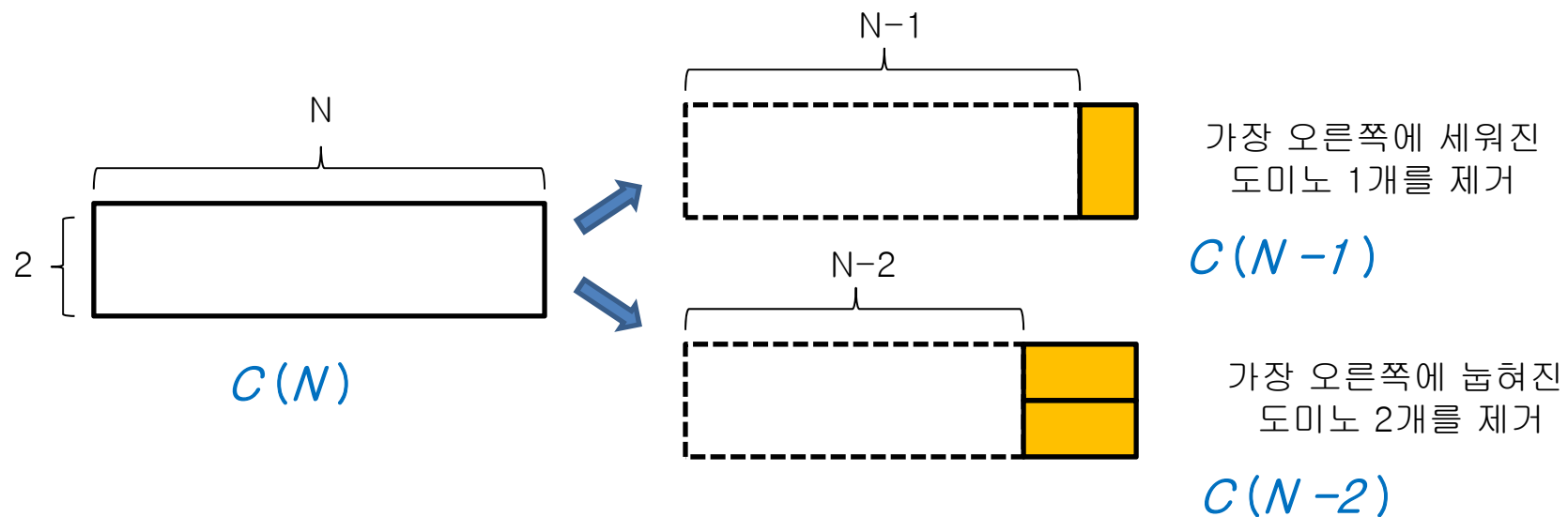
직사각형을 타일로 오른쪽에서 왼쪽으로 하나씩 다 채워 놓았다고 생각하고, 오른쪽에서 왼쪽으로 하나씩 타일을 빼 나갈 때, 어떤 형태가 나타나는지를 분석해보자.

Think  
"Working Backward"



# 2xN 직사각형 타일 채우기 문제 (3)

- 단계 2: 재귀식
  - $C(N)$  : 2xN 직사각형을 도미노 타일로 채우는 방법의 수



$$C(N) = C(N-1) + C(N-2)$$

# 2xN 직사각형 타일 채우기 문제 (4)

- 단계 2: 재귀식
  - $C(N)$  : 2xN 직사각형을 도미노 타일로 채우는 방법의 수

$$C(N) = \begin{cases} 1 & N = 1 \quad (\text{base case}) \\ 2 & N = 2 \quad (\text{recursive step}) \\ C(N-1) + C(N-2) & N > 2 \quad (\text{recursive step}) \end{cases}$$

Fibonacci 수와 같음

# 동전교환문제

- 동전교환문제

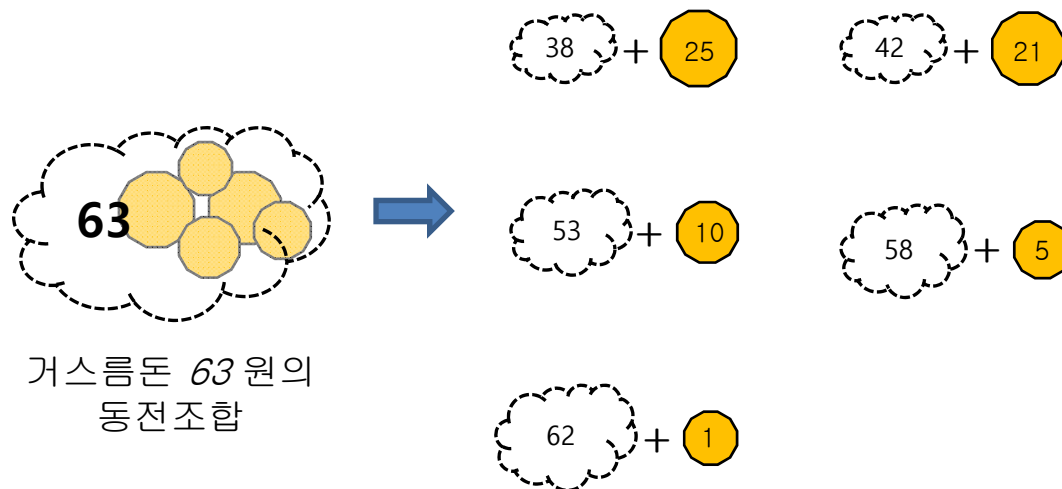
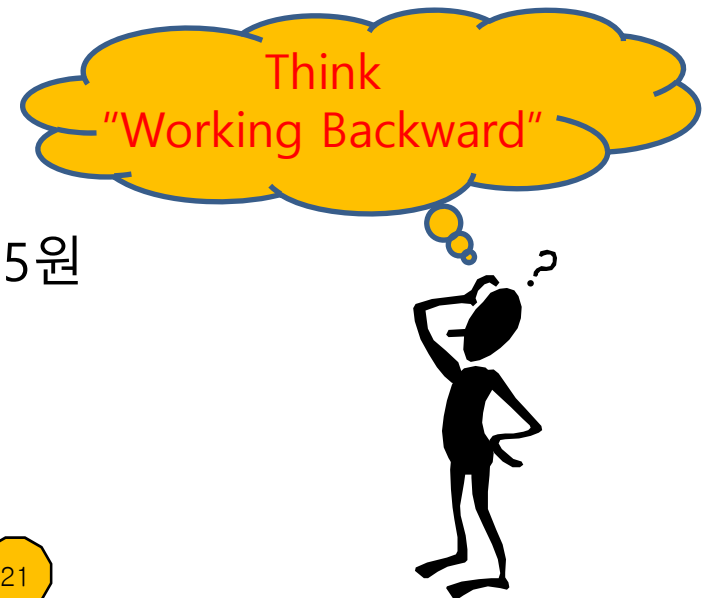
- 서로 다른 단위의 동전이 주어졌을 때, 거스름돈을 동전의 개수가 최소가 되도록 교환해 주려고 한다. 이때 교환해 주는 동전의 최소 개수와 교환해 주는 동전의 조합을 계산하시오. 단, 모든 단위의 동전은 무수히 많다고 가정한다.

- 예

- 동전의 종류 : 1원, 5원, 10원, 21원, 25원
    - 거스름 돈 63원
      - 최소동전개수 : 3개
        - » {21, 21, 21}

# 동전교환문제 (2)

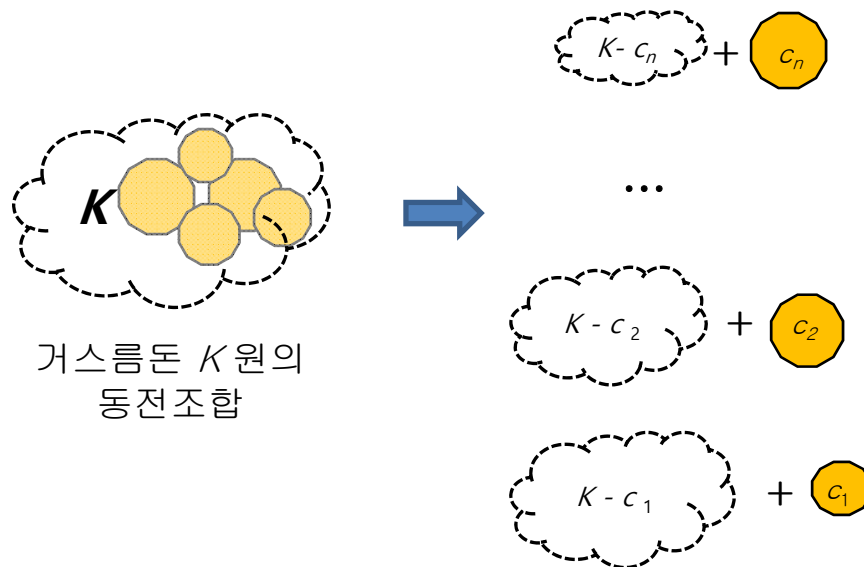
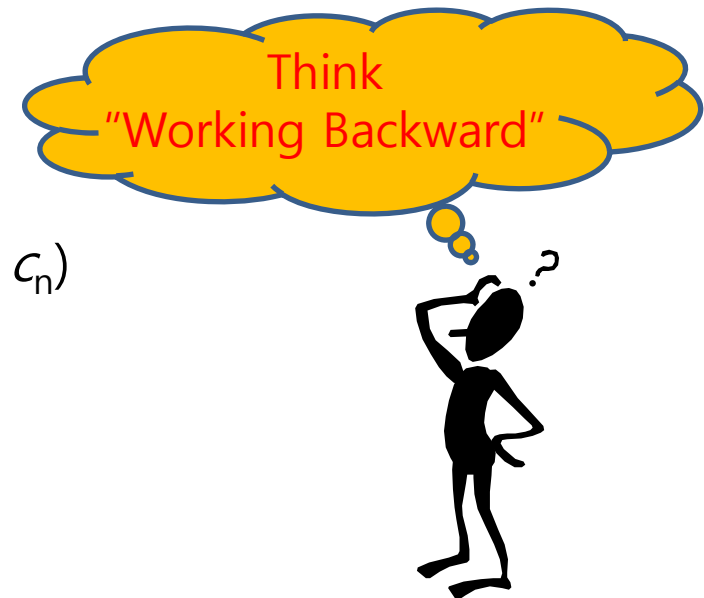
- 단계 1:
  - 동전조합의 구조분석
    - 동전의 종류 : 1원, 5원, 10원, 21원, 25원
    - 거스름 돈 : 63 원



거스름돈 63 원의  
동전조합

# 동전교환문제 (3)

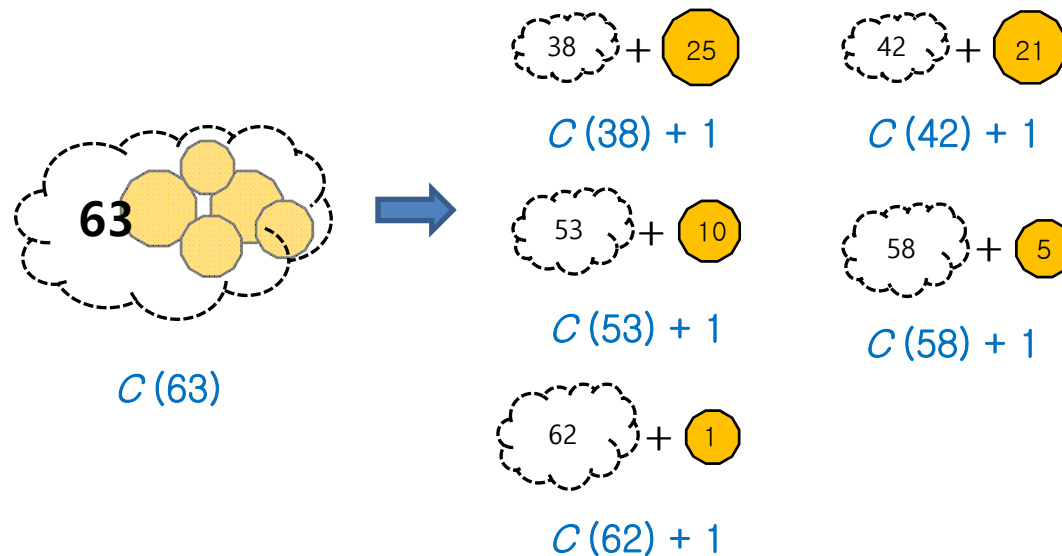
- 단계 1:
  - 동전조합의 구조분석
    - 동전의 종류 :  $n$  가지 ( $c_1 < c_2 < \dots < c_n$ )
    - 거스름 돈 :  $K$  원





# 동전교환문제 (4)

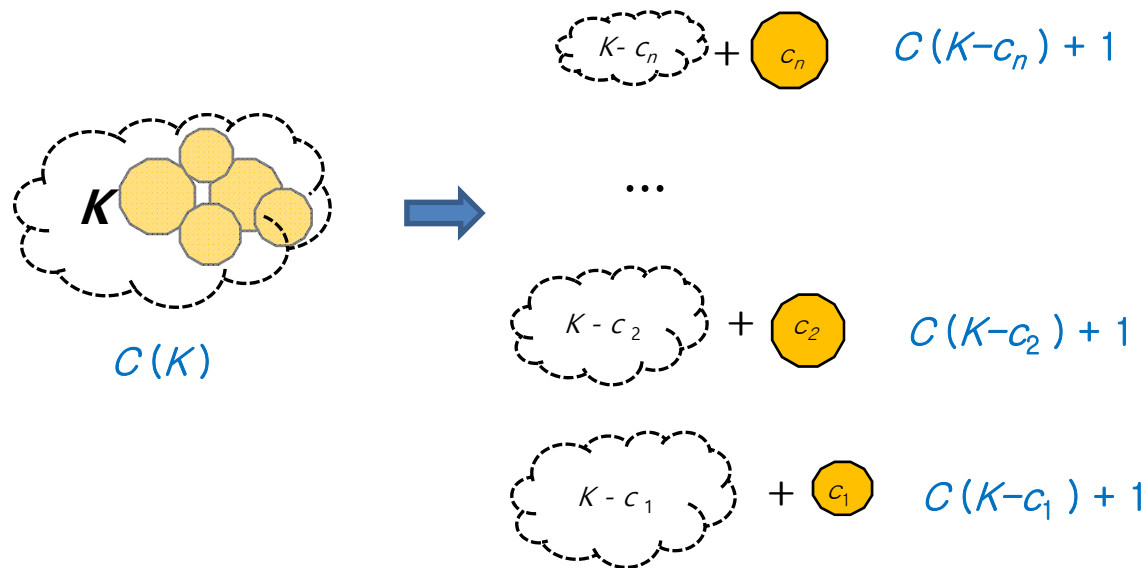
- 단계 2: 재귀식
  - $C(k)$  :  $k$  원을 바꿀 때, 최소 동전의 개수



$$\begin{aligned} C(63) &= \min \{ C(38)+1, C(42)+1, C(53)+1, C(58)+1, C(62)+1 \} \\ &= \min \{ C(38), C(42), C(53), C(58), C(62) \} + 1 \end{aligned}$$

# 동전교환문제 (5)

- 단계 2: 재귀식
  - $C(k)$  :  $k$  원을 바꿀 때, 최소 동전의 개수



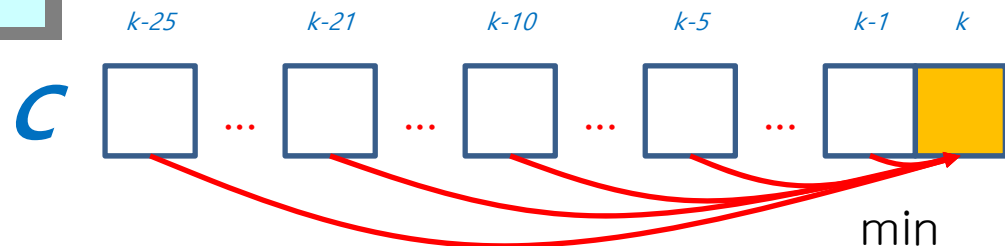
$$C(K) = \min_{1 \leq i \leq n} C(K - c_i) + 1$$

# 동전교환문제 (6)

- 단계 3: 최소동전의 개수 계산
  - $C[k]$  :  $k$  원을 바꿀 때, 최소 동전의 개수 저장

$$C[k] = \begin{cases} \infty & k < 0 \\ 0 & k = 0 \\ \min\{C[k-25], C[k-21], C[k-10], C[k-5], C[k-1]\} + 1 & k > 0 \end{cases}$$

$$C[k] = \begin{cases} \infty & k < 0 \\ 0 & k = 0 \\ \min_{1 \leq i \leq n} C[k - c_i] + 1 & k > 0 \end{cases}$$



$S[k]$  : 최소 개수의 거스름돈 동전을 계산하기 위하여  
위의  $C[k]$  를 계산할 때, 최소값으로 선택된 동전  $c_i$  를 저장

# 동전교환문제 (7)

- 단계 3: 최소동전의 개수 계산

- 동전의 종류 : 1원, 5원, 10원, 21원, 25원
- 거스름 돈 : 21 원

Compute  
"Bottom Up"



$k$	0	1	2	3	4	5	6	7	8	9	10
$C$	0	1	2	3	4	1	2	3	4	5	1
$S$	0	1	1	1	1	5	1	5	1	5	10

$k$	11	12	13	14	15	16	17	18	19	20	21
$C$	2	3	4	5	2	3	4	5	6	2	1
$S$	1	10	10	1	5	1	5	10	1	10	21

# 동전교환문제 (8)

- 단계 3: 최소동전의 개수 계산

Compute  
"Bottom Up"

```
void coinExchange(int coins[], int numDiffCoins, int change,  
                  int coinsUsed[], int lastCoin[])  
{  
    int cents, j;  
  
    /* coinsUsed = C, lastCoin = L */  
    coinsUsed[0]=lastCoin[0]=0;  
  
    for(cents = 1; cents <= change; cents++)  
    {  
        int minCoins, newCoin;  
  
        minCoins = cents;  
        newCoin = 1;  
        for(j=0; j<numDiffCoins; j++)  
        {  
            if (coins[j] > cents)  
                continue;  
            if (coinsUsed[cents-coins[j]] + 1 < minCoins)  
            {  
                minCoins = coinsUsed[cents-coins[j]] + 1;  
                newCoin = coins[j];  
            }  
        }  
  
        coinsUsed[cents] = minCoins;  
        lastCoin[cents] = newCoin;  
    }  
}
```



# 동전교환문제 (9)

- 단계 4: 최소동전의 집합 계산 (recursive)
  - 1차원 배열 L[ ] 이용

```
/* coinsUsed = C, lastCoin = L */  
void reconstruct(int change, int lastCoin[])  
{  
    if(change > 0)  
    {  
        reconstruct(change-lastCoin[change], lastCoin);  
        printf("%d ", lastCoin[change]);  
    }  
}
```

# 2차 동적계획법

- 2차 동적계획법
  - 재귀식에서 2개의 변수가 필요한 문제
  - 2차원 배열로 동적계획법 구현
  - 예
    - 이항계수 (binomial coefficient) 계산문제
    - 동전교환 (coin exchange) 문제
    - 최장공통부분수열 (longest common subsequence)
    - 대부분의 동적계획법 문제

# 이항계수 계산문제

- Binomial Coefficient (이항계수)

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{20}{15} = \frac{20!}{15!5!} = \frac{2432902008176640000}{1307674368000 \times 120} = 15504$$



# 이항계수 계산문제 (2)

- 단계 1, 단계 2:
  - 이항계수의 구조분석 및 재귀식
    - by Pascal

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \end{cases}$$

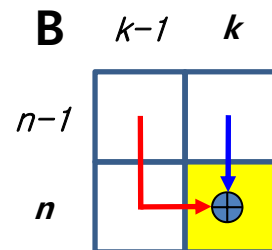
# 이항계수 계산문제 (3)

- 단계 3 : 이항계수 계산

- 2차원 배열  $B[n][k]$

- $B[n][k]$  : store  $\binom{n}{k}$

$$B[n][k] = \begin{cases} 1 & k = 0 \text{ or } k = n \text{ (base case)} \\ B[n-1][k-1] + B[n-1][k] & 0 < k < n \text{ (recursive step)} \end{cases}$$

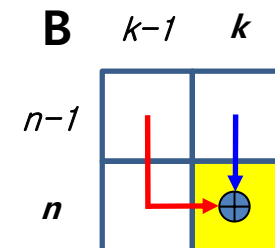


# 이항계수 계산문제 (4)

- 단계 3 : 이항계수 계산

		$k$							
$B$		0	1	2	3	4	5	6	7
$n$	0	1							
	1	1	1						
	2	1	2	1					
	3	1	3	3	1				
	4	1	4	6	4	1			
	5	1	5	10	10	5			
	6	1	6	15	20	15			

Compute  
"Bottom Up"



# 이항계수 계산문제 (5)

- 단계 3 : 이항계수 계산

```
#define MAX 30

#define MIN(a,b) ((a)<(b)?(a):(b))

int binCoeff(int n, int k)
{
    int i, j;
    int B[MAX][MAX];

    for(i=0; i<=n; i++)
        for(j=0; j<=MIN(i,k); j++)
            if(j==0 || j==i)
                B[i][j] = 1; /* base case */
            else
                B[i][j] = B[i-1][j-1]+B[i-1][j];

    return B[n][k];
}
```

Compute  
"Bottom Up"

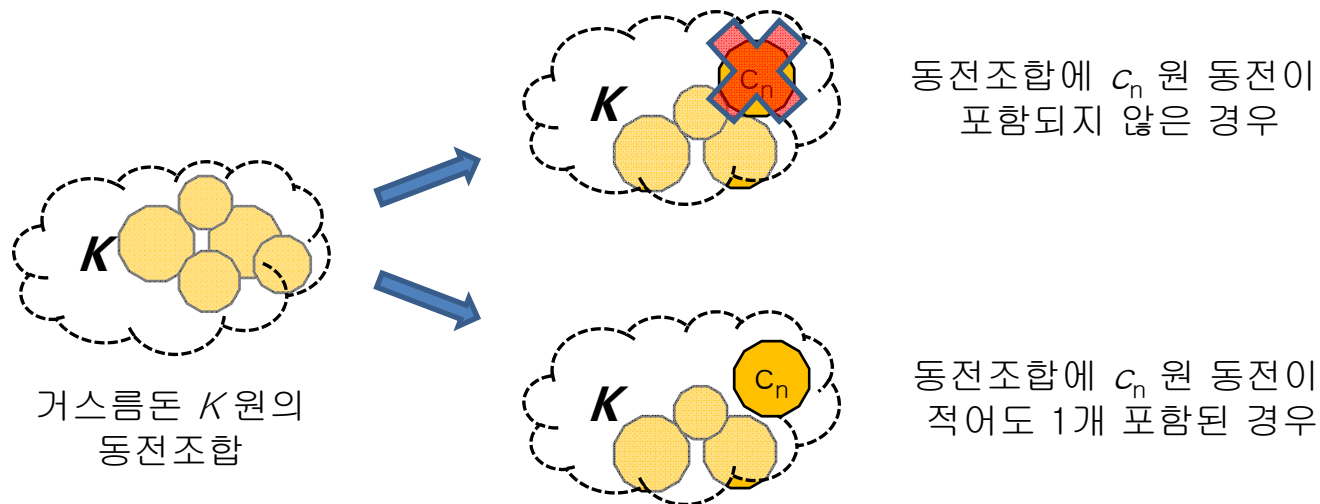
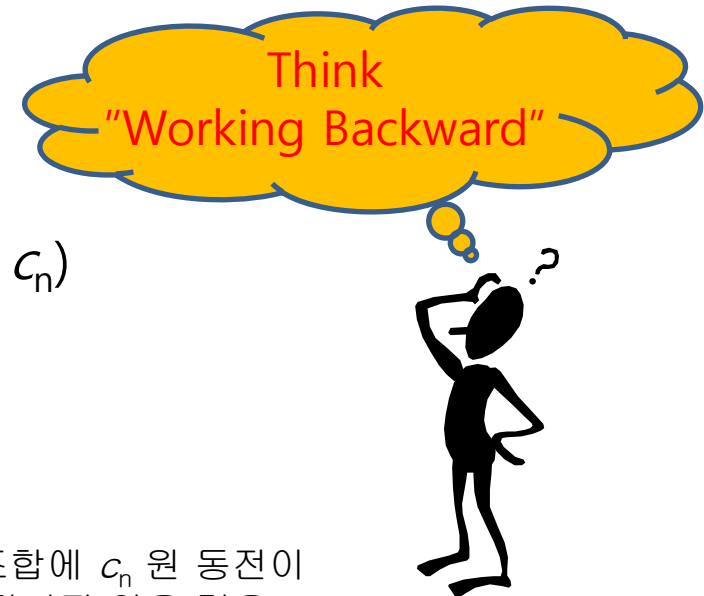


# 동전교환문제-2

- 동전교환문제-2
  - 서로 다른 단위의 동전이 주어졌을 때, 거스름돈을 교환해 주는 동전의 조합의 수를 계산하시오. 단, 모든 단위의 동전은 무수히 많다고 가정한다.
  - 예
    - 동전의 종류 : 1원, 2원, 3원
    - 거스름 돈 4원
      - 교환방법 : 4가지
        - » {1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}

# 동전교환문제-2 (2)

- 단계 1:
  - 동전조합의 구조분석
    - 동전의 종류 :  $n$  가지 ( $c_1 < c_2 < \dots < c_n$ )
    - 거스름 돈 :  $K$  원

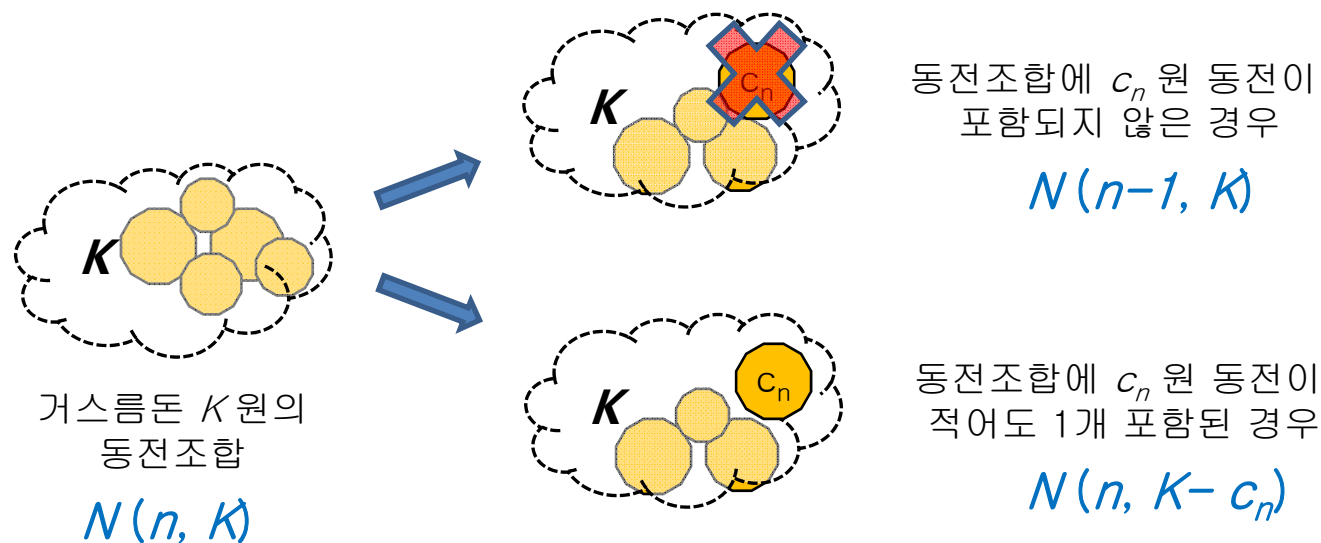


# 동전교환문제-2 (3)

- 단계 2:

- 재귀식

- $N(n, K)$  : 거스름돈  $K$  원을  $n$  개의 동전  $c_1, c_2, \dots, c_n$  으로 교환하는 동전조합의 수



$$N(n, K) = N(n-1, K) + N(n, K - c_n)$$

## 동전교환문제-2 (4)

- 단계 2:
  - 재귀식
    - $N(n, K)$  : 거스름돈  $K$  원을  $n$  개의 동전  $c_1, c_2, \dots, c_n$  으로 교환하는 동전조합의 수

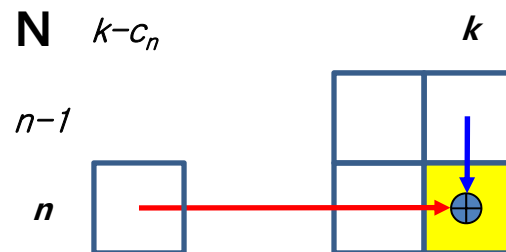
$$N(n, K) = \begin{cases} 0 & n = 0 \text{ and } K > 0 & \text{(base case)} \\ 1 & K = 0 & \text{(base case)} \\ 0 & K < 0 & \text{(base case)} \\ N(n-1, K) + N(n, K - c_n) & \text{otherwise} & \text{(recursive step)} \end{cases}$$



# 동전교환문제-2 (5)

- 단계 3: 동전조합의 수 계산
  - 2차원 배열 :  $N[n][k]$ 
    - $N[n][k]$  : 거스름돈  $k$  원을  $n$  개의 동전  $c_1, c_2, \dots, c_n$  으로 교환하는 동전조합의 수

$$N[n][k] = \begin{cases} 0 & n = 0 \text{ and } K > 0 & \text{(base case)} \\ 1 & K = 0 & \text{(base case)} \\ 0 & K < 0 & \text{(base case)} \\ N[n-1][k] + N[n][k - c_n] & \text{otherwise} & \text{(recursive step)} \end{cases}$$



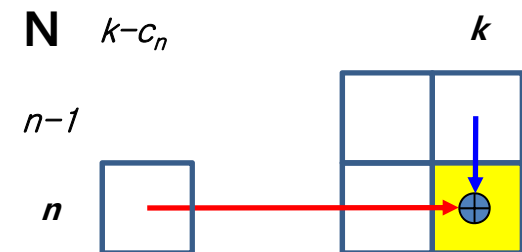
# 동전교환문제-2 (6)

- 단계 3: 동전조합의 수 계산
  - 동전의 종류 : 4가지 (1원, 2원, 3원, 5원)
  - 거스름돈 : 7원

Compute  
"Bottom Up"



		k							
		0	1	2	3	4	5	6	7
n	0	1	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1
	2	1	1	2	2	3	3	4	4
	3	1	1	2	3	4	5	7	8
	4	1	1	2	3	4	6	8	10



# 동전교환문제-2 (7)

- 단계 3 : 동전조합의 수 계산

```
#define MAX_COINS 101
#define MAX_CHANGE 1001

int countCoinExchange(int coins[], int numDiffCoins, int change)
{
    int i, j, numComb;
    int N[MAX_COINS][MAX_CHANGE] = {0};

    /* base cases */
    for(i = 1; i <= numDiffCoins; i++)
        N[i][0] = 1;
    for(i = 1; i <= change; i++)
        N[0][i] = 0;

    for(i = 1; i <= numDiffCoins; i++)
        for(j = 1; j <= change; j++)
        {
            if (j-coins[i] < 0) /* base case */
                numComb = 0;
            else
                numComb = N[i][j-coins[i]];
            N[i][j] = N[i-1][j] + numComb;
        }

    return N[numDiffCoins][change];
}
```

# Longest Common Subsequence

- Longest Common Subsequence (Substring)

(최장 공통 부분수열/부분스트링)

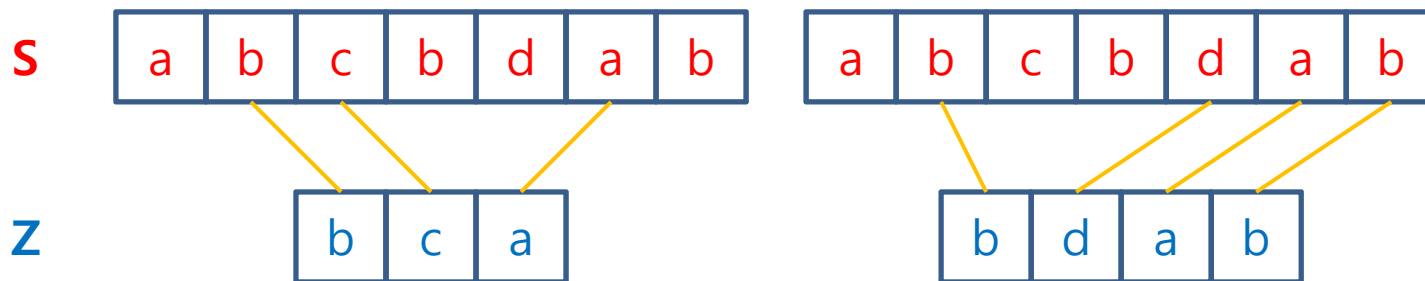
- Notations:

– Sequence  $S = S_m = \langle s_1, s_2, \dots, s_m \rangle$

– Subsequence of  $S$  :  $Z$

$Z = Z_k = \langle z_1, z_2, \dots, z_k \rangle$  such that

$z_1 = s_{i_1}, z_2 = s_{i_2}, \dots, z_k = s_{i_k} \ (i_1 < i_2 < \dots < i_k)$



# Longest Common Subsequence (2)

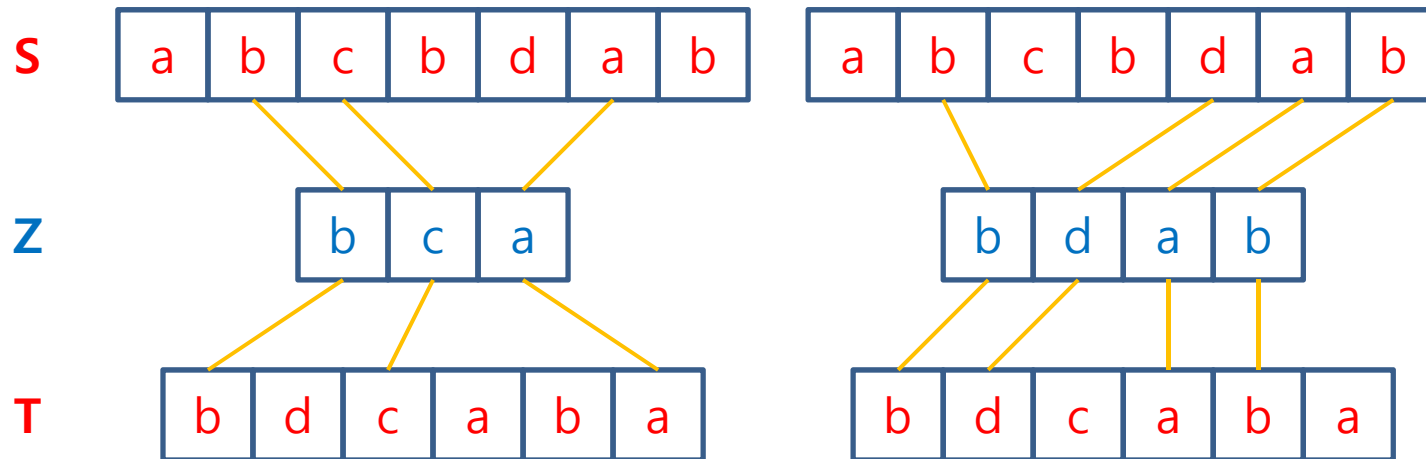
- Notations

- Sequences

- $S = S_m = \langle s_1, s_2, \dots, s_m \rangle$

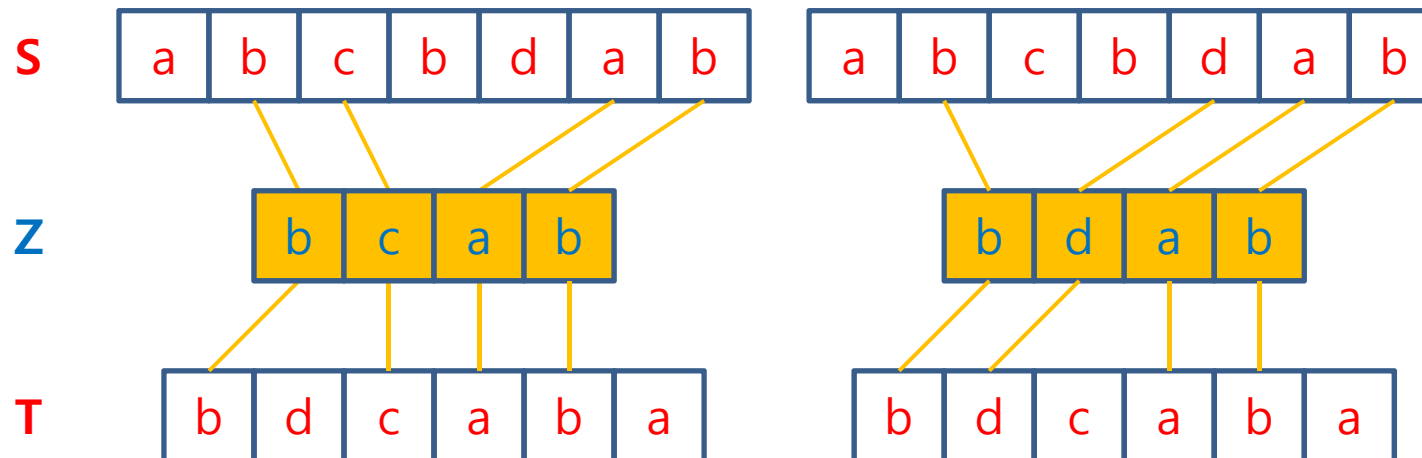
- $T = T_n = \langle t_1, t_2, \dots, t_n \rangle$

- Common Subsequence of  $S, T$ :  $Z$



# Longest Common Subsequence (3)

- Notations
  - Longest common subsequence of  $S$ ,  $T$ 
    - Not unique



# Longest Common Subsequence (4)

- Longest Common Subsequence Problem
    - Given two sequences  $S, T$ 
      - $S = S_m = \langle s_1, s_2, \dots, s_m \rangle$
      - $T = T_n = \langle t_1, t_2, \dots, t_n \rangle$
- find a longest common subsequence of  $S$  and  $T$ .

Optimization Problem  
(최적화문제)

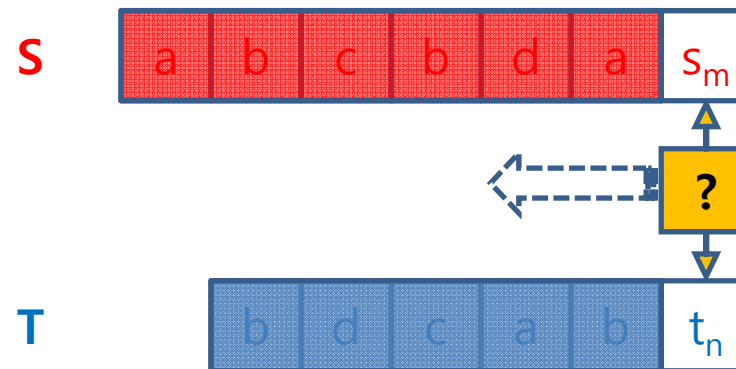
Dynamic Programming

- (1) Find the **length** of the longest common subsequence
- (2) Find a longest common subsequence

# Longest Common Subsequence (5)

- 단계 1:
  - 최장공통부분수열의 구조분석

Think  
"Working Backward"

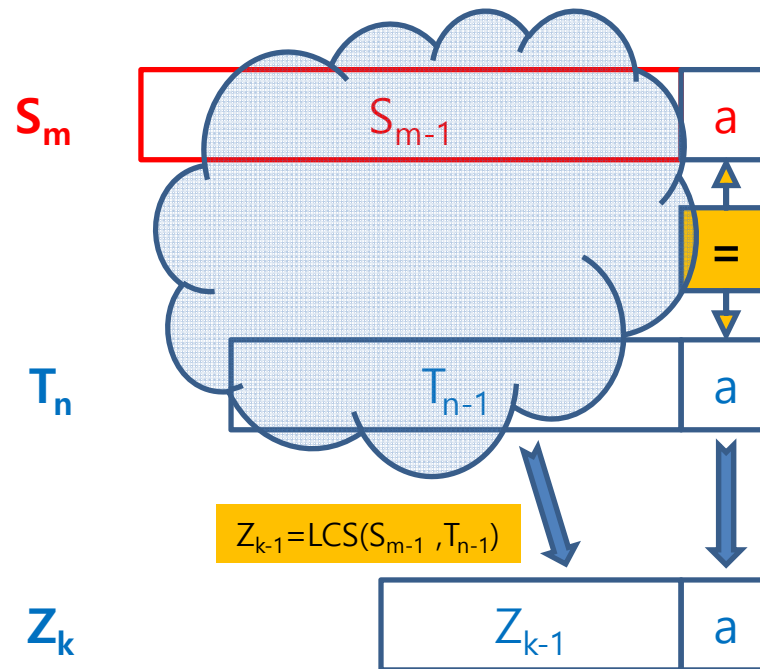




# Longest Common Subsequence (6)

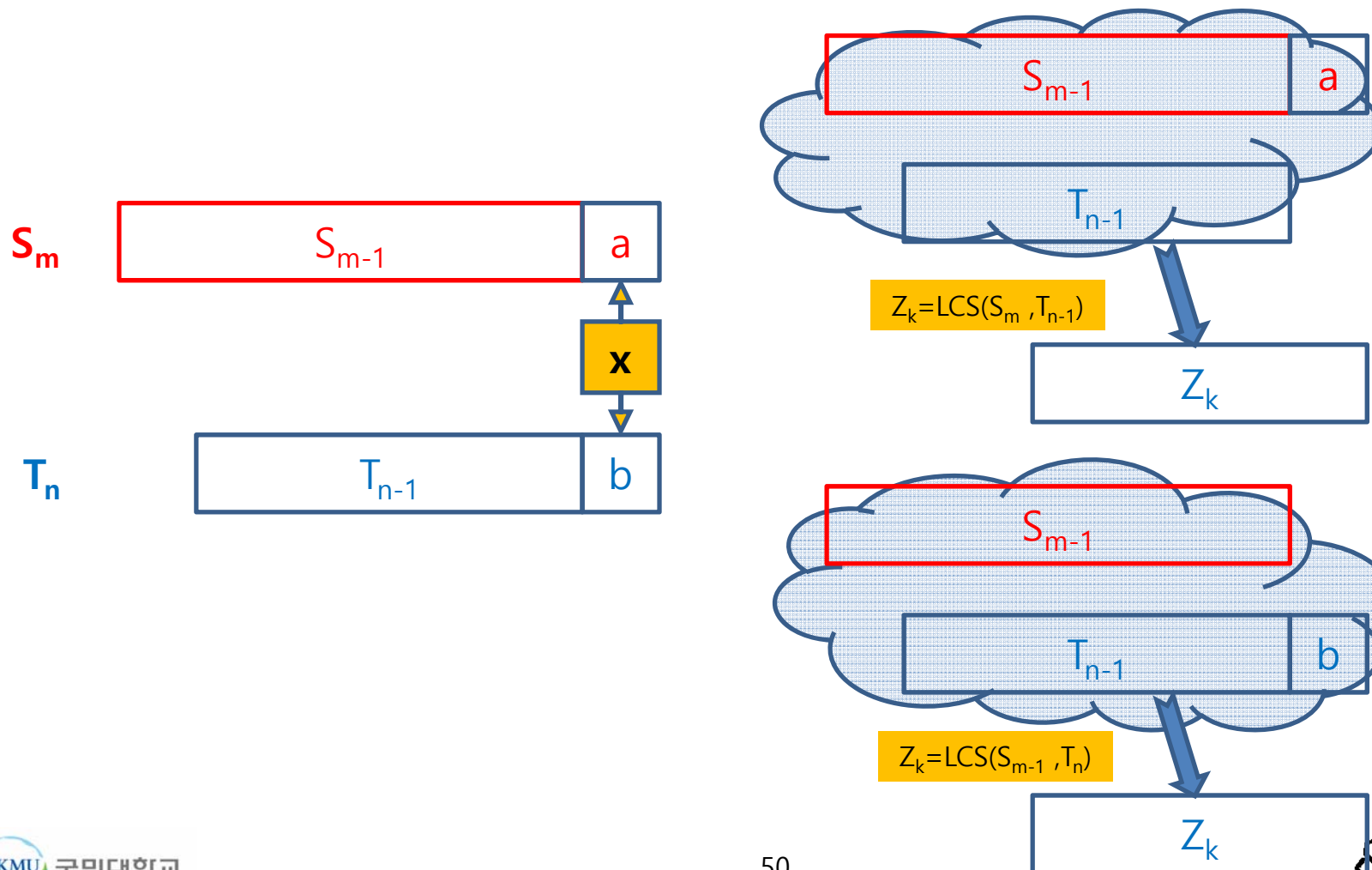
- 단계 1:
  - 최장공통부분수열의 구조분석

Think  
"Working Backward"



# Longest Common Subsequence (7)

- 단계 1: 최장공통부분수열의 구조분석



# Longest Common Subsequence (8)

- 단계 2: 재귀식

- $L(m, n)$  : Length of a LCS of  $S_m$  and  $T_n$

$$L(m, n) = \begin{cases} 0 & m = 0 \text{ or } n = 0 & \text{(base case)} \\ L(m-1, n-1) + 1 & m, n > 0 \text{ and } s_m = t_n & \text{(recursive step)} \\ \max\{L(m, n-1), L(m-1, n)\} & m, n > 0 \text{ and } s_m \neq t_n & \text{(recursive step)} \end{cases}$$

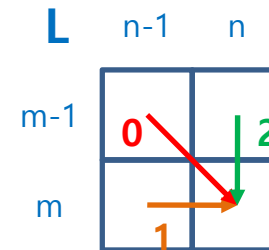
# Longest Common Subsequence (9)

- 단계 3: LCS 길이계산
  - 2차원 배열  $L[m][n]$ ,  $S[m][n]$
  - $L[m][n]$  : Store length of a LCS of  $S_m$  and  $T_n$

$$L[m][n] = \begin{cases} 0 & m = 0 \text{ or } n = 0 & (\text{base case}) \\ L[m-1][n-1] + 1 & m, n > 0 \text{ and } s_m = t_n & (\text{recursive step}) \\ \max\{L[m][n-1], L[m-1][n]\} & m, n > 0 \text{ and } s_m \neq t_n & (\text{recursive step}) \end{cases}$$

- $S[m][n]$  :  $S_m$  and  $T_n$  의 LCS 를 구하기 위한 정보 저장

$$S[m][n] = \begin{cases} 0 & \text{if } L[m][n] = L[m-1][n-1] \text{ } (s_m = t_n) \\ 1 & \text{if } L[m][n] = L[m][n-1] \\ 2 & \text{if } L[m][n] = L[m-1][n] \end{cases}$$



# Longest Common Subsequence (10)

- 단계 3: LCS 길이계산 구현

```
#define MAX_LENGTH 101
#define MAX(a,b) ((a)>(b)?(a):(b))

int L[MAX_LENGTH][MAX_LENGTH], S[MAX_LENGTH][MAX_LENGTH];

int lengthLCS(char s[], char t[], int m, int n)
{
    int i, j;

    /* base cases */
    for(i = 0; i <= m; i++)
        L[i][0] = 0;
    for(i = 0; i <= n; i++)
        L[0][i] = 0;

    for(i = 1; i <= m; i++)
        for(j = 1; j <= n; j++)
            if (s[i-1] == t[j-1]){
                L[i][j] = L[i-1][j-1]+1;
                S[i][j] = 0;
            }
            else{
                L[i][j] = MAX(L[i][j-1], L[i-1][j]);
                if (L[i][j] == L[i][j-1])
                    S[i][j] = 1;
                else
                    S[i][j] = 2;
            }

    return L[m][n];
}
```

Compute  
"Bottom Up"



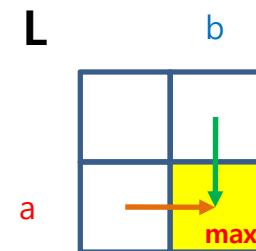
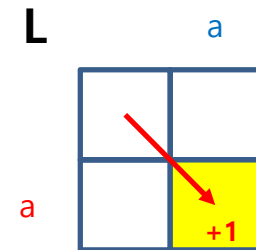
# Longest Common Subsequence (11)

- 단계 3: LCS 길이계산 구현

**T**

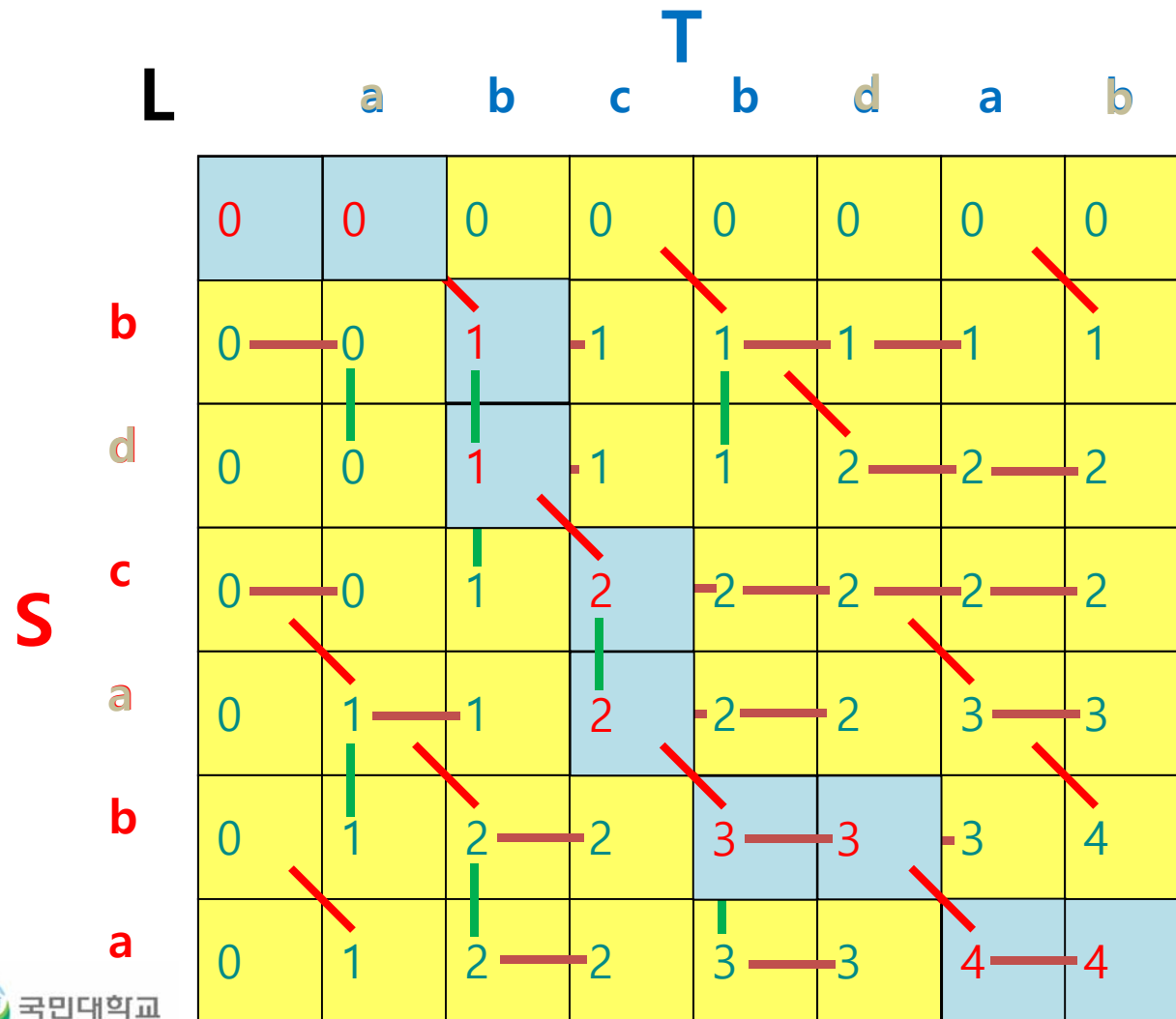
<b>L</b>		a	b	c	b	d	a	b
	0	0	0	0	0	0	0	0
<b>b</b>	0	0	1	1	1	1	1	1
<b>d</b>	0	0	1	1	1	2	2	2
<b>c</b>	0	0	1	2	2	2	2	2
<b>a</b>	0	1	1	2	2	2	3	3
<b>b</b>	0	1	2	2	3	3	3	4
<b>a</b>	0	1	2	2	3	3	4	4

Compute  
"Bottom Up"



# Longest Common Subsequence (12)

- 단계 4: LCS 계산



# Longest Common Subsequence (13)

- 단계 4: LCS 계산 (recursive)

```
void printLCS(char s[], char t[], int m, int n)
{
    if(m==0 || n==0)
        return;
    if(S[m][n] == 0)
    {
        printLCS(s, t, m-1, n-1);
        printf("%c", s[m-1]);
    }
    else if(S[m][n] == 1)
        printLCS(s, t, m, n-1);
    else if(S[m][n] == 2)
        printLCS(s, t, m-1, n);
}
```



# Chained Matrix Multiplication

- Matrix Multiplication

$A = [a_{ij}]$  : matrix of size  $p \times q$

$B = [b_{ij}]$  : matrix of size  $q \times r$

– Then  $AB = C([c_{ij}])$  is a matrix of size  $p \times r$  such that

$$c_{ij} = \sum_{k=1}^q a_{ik} b_{kj}$$

– To compute  $C=AB$ , it takes  
 $p \times q \times r$   
multiplications.

# Chained Matrix Multiplication

- Matrix Multiplication
  - Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 29 & 35 & 41 & 38 \\ 74 & 89 & 104 & 83 \end{bmatrix}$$

# Chained Matrix Multiplication

- Chained Matrix Multiplication
  - Note that matrix multiplication is **associative**

$$A \times B \times C = (A \times B) \times C = A \times (B \times C)$$

- How many multiplication of elements are needed to compute  $(AB)C$  and  $A(BC)$  ?

$A$  : matrix of size  $p \times q$

$B$  : matrix of size  $q \times r$

$C$  : matrix of size  $r \times s$

$(AB)C$  :  $pqr + prs$

$A(BC)$  :  $qrs + pqs$

❖ Even  $(AB)C = A(BC)$ , the number of multiplications of elements are different.

# Chained Matrix Multiplication

- Example:
  - Multiplication of the following 4 matrices

$A$	$\times$	$B$	$\times$	$C$	$\times$	$D$
$20 \times 2$		$2 \times 30$		$30 \times 12$		$12 \times 8$

- The number of elementary multiplication of the following order of matrix multiplications:

$$A(B(CD)) \quad 30 \times 12 \times 8 + 2 \times 30 \times 8 + 20 \times 2 \times 8 = 3,680$$

$$(AB)(CD) \quad 20 \times 2 \times 30 + 30 \times 12 \times 8 + 20 \times 30 \times 8 = 8,880$$

$$A((BC)D) \quad 2 \times 30 \times 12 + 2 \times 12 \times 8 + 20 \times 2 \times 8 = 1,232$$

$$((AB)C)D \quad 20 \times 2 \times 30 + 20 \times 30 \times 12 + 20 \times 12 \times 8 = 10,320$$

$$(A(BC))D \quad 2 \times 30 \times 12 + 20 \times 2 \times 12 + 20 \times 12 \times 8 = 3,120$$

- Therefore  $A((BC)D)$  is the optimal order for the multiplication  $ABCD$  of four matrices.

# Chained Matrix Multiplication

- Definition

- We are given matrices  $A_1 A_2 \dots A_n$ , where the dimensions of  $A_i$  are  $d_{i-1} \times d_i$ .

- How should we compute

$$A_1 \times A_2 \times \dots \times A_n$$

and what is the minimum number of elementary multiplications needed?

# Chained Matrix Multiplication

- 단계 1:
  - 연속행렬곱셈의 구조분석

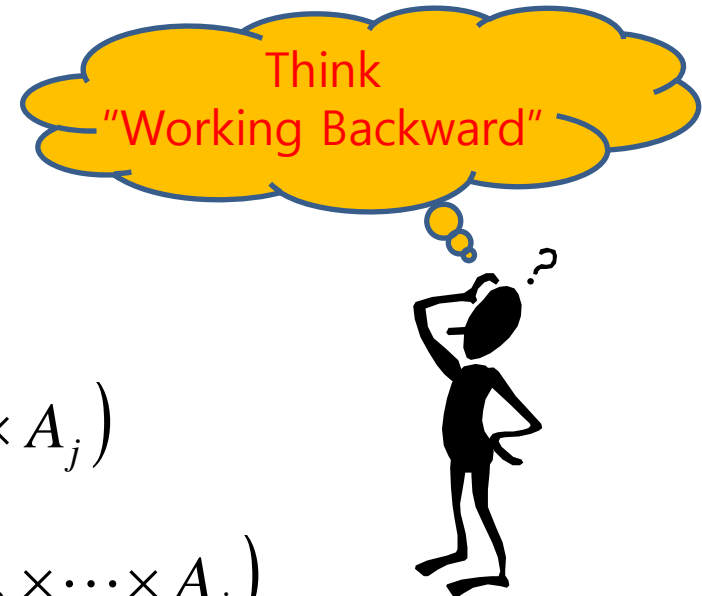
Think  
"Working Backward"



$$\begin{array}{l} A_1 \times A_2 \times \cdots \times A_n \quad \Rightarrow \quad \begin{array}{l} (A_1) \times (A_2 \times \cdots \times A_n) \\ (A_1 \times A_2) \times (A_3 \times \cdots \times A_n) \\ \dots \\ (A_1 \times \cdots \times A_k) \times (A_{k+1} \times \cdots \times A_n) \\ \dots \\ (A_1 \times A_2 \times \cdots \times A_{n-1}) \times (A_n) \end{array} \end{array}$$

# Chained Matrix Multiplication

- 단계 1:
  - 연속행렬곱셈의 구조분석



$$\begin{array}{ccc}
 & & (A_i) \times (A_{i+1} \times \dots \times A_j) \\
 & & (A_i \times A_{i+1}) \times (A_{i+2} \times \dots \times A_j) \\
 & & \dots \\
 & & (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j) \\
 & & \dots \\
 & & (A_i \times A_{i+1} \times \dots \times A_{j-1}) \times (A_j) \\
 A_i \times A_{i+1} \times \dots \times A_j & \Rightarrow & 
 \end{array}$$

# Chained Matrix Multiplication

- 단계 1:
  - Suppose the matrices are factored as follows:

$$(A_i \times \cdots \times A_k) \times (A_{k+1} \times \cdots \times A_j)$$

- The dimensions of the two factor matrices are  $d_{i-1} \times d_k$  and  $d_k \times d_j$ .
  - Recursively calls to find out how many multiplications are needed for each factor.
  - But what is the best choice for  $k$ ?
  - Since we do not know the best place to split the sequence of matrices into two factors, we minimize overall choices for  $k$ .



# Chained Matrix Multiplication

- 단계 2:
  - 재귀식
    - $M(i, j)$  : the minimum number of multiplications needed to compute  $A_i \times \dots \times A_j$ , for  $1 \leq i \leq j \leq n$ .
  - Then we have the following recurrence relation:

$$M(i, j) = \begin{cases} \min_{i \leq k \leq j-1} (M(i, k) + M(k+1, j) + d_{i-1}d_kd_j) & \text{if } 1 \leq i < j \leq n \\ 0 & \text{if } i = j \end{cases}$$

$$\underbrace{(A_i \times \dots \times A_k)}_{d_{i-1} \times d_k \text{ matrix}} \times \underbrace{(A_{k+1} \times \dots \times A_j)}_{d_k \times d_j \text{ matrix}}$$

$d_{i-1}d_kd_j$  multiplications

# Chained Matrix Multiplication

- 단계 2:
  - 재귀식

$$M(i, j) = \begin{cases} \min_{i \leq k \leq j-1} (M(i, k) + M(k+1, j) + d_{i-1}d_kd_j) & \text{if } 1 \leq i < j \leq n \\ 0 & \text{if } i = j \end{cases}$$

- Recursive (divide-and-conquer) algorithm ?

- Recurrence relation?
- Emmm, solve it with a recursive function.
- But wait for a moment!
- Some subproblems do overlap!
- Example:

$$[A_i \times \cdots \times A_k] \times [(A_{k+1} \times \cdots \times A_g) \times (A_{g+1} \times \cdots \times A_j)]$$

$$[(A_i \times \cdots \times A_k) \times (A_{k+1} \times \cdots \times A_g)] \times [A_{g+1} \times \cdots \times A_j]$$

# Chained Matrix Multiplication

- 단계 2:
  - 재귀식

$$M(i, j) = \begin{cases} \min_{i \leq k \leq j-1} (M(i, k) + M(k+1, j) + d_{i-1}d_kd_j) & \text{if } 1 \leq i < j \leq n \\ 0 & \text{if } i = j \end{cases}$$

- Table

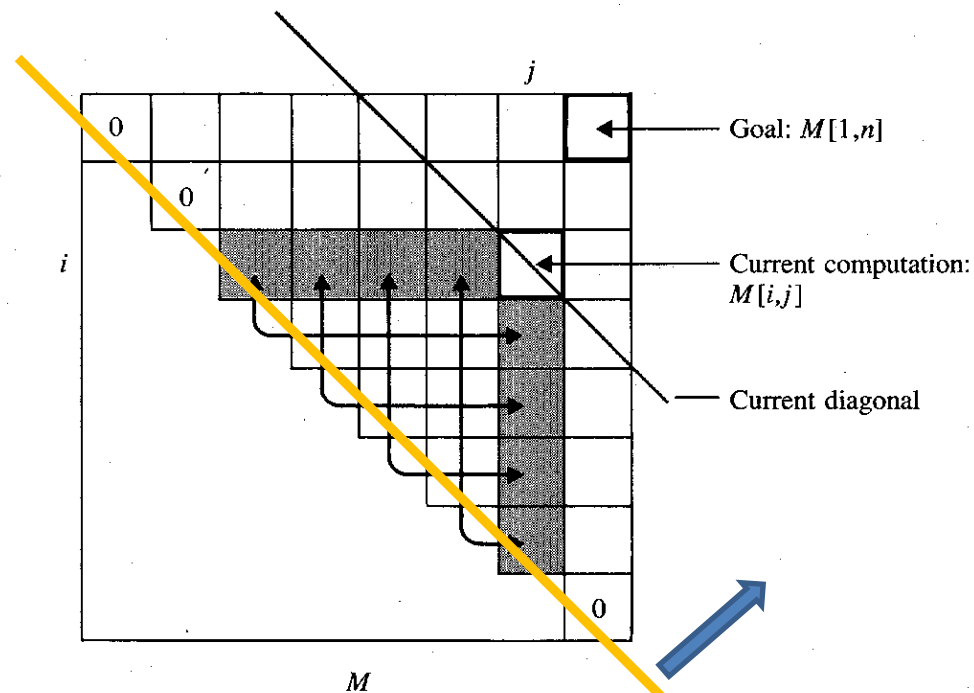
- Let  $M[i][j]$  be store  $M(i, j)$  which is the minimum number of multiplications needed to compute  $A_i \times \dots \times A_j$ , for  $1 \leq i \leq j \leq n$ .

$$M[i][j] = \begin{cases} \min_{i \leq k \leq j-1} (M[i][k] + M[k+1][j] + d_{i-1}d_kd_j) & \text{if } 1 \leq i < j \leq n \\ 0 & \text{if } i = j \end{cases}$$

# Chained Matrix Multiplication

- 단계 3:

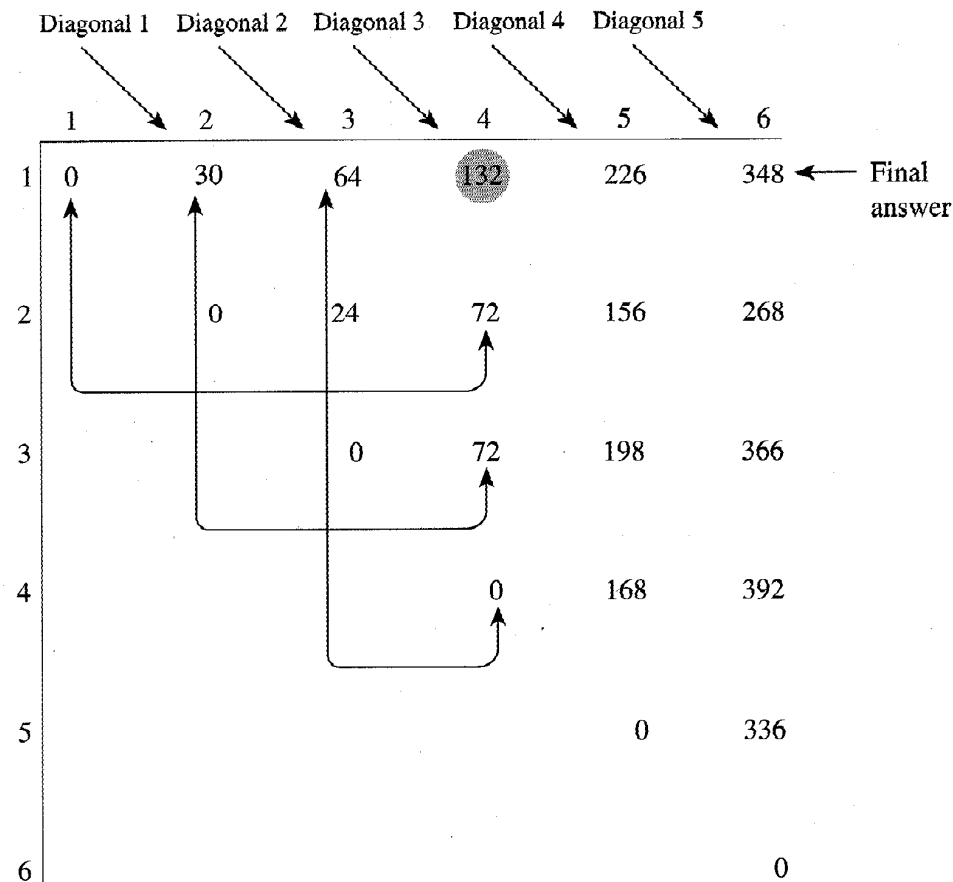
$$M[i][j] = \begin{cases} \min_{i \leq k \leq j-1} (M[i][k] + M[k+1][j] + d_{i-1}d_kd_j) & \text{if } 1 \leq i < j \leq n \\ 0 & \text{if } i = j \end{cases}$$



# Chained Matrix Multiplication

- 단계 3:  
– Example

$$\begin{array}{cccccc}
 A_1 & \times & A_2 & \times & A_3 & \times & A_4 & \times & A_5 & \times & A_6 \\
 5 \times 2 & & 2 \times 3 & & 3 \times 4 & & 4 \times 6 & & 6 \times 7 & & 7 \times 8 \\
 d_0 \ d_1 & & d_1 \ d_2 & & d_2 \ d_3 & & d_3 \ d_4 & & d_4 \ d_5 & & d_5 \ d_6
 \end{array}$$



# Chained Matrix Multiplication

- 단계 3:

- Example

- Diagonal 0

$$\begin{array}{cccccc}
 A_1 & \times & A_2 & \times & A_3 & \times & A_4 & \times & A_5 & \times & A_6 \\
 5 \times 2 & & 2 \times 3 & & 3 \times 4 & & 4 \times 6 & & 6 \times 7 & & 7 \times 8 \\
 d_0 & d_1 & d_1 & d_2 & d_2 & d_3 & d_3 & d_4 & d_4 & d_5 & d_5 & d_6
 \end{array}$$

$$M[i][i] = 0, \quad \text{for } 1 \leq i \leq 6$$

- Diagonal 1

$$\begin{aligned}
 M[1][2] &= \min_{1 \leq k \leq 1} (M[1][k] + M[k+1][2] + d_0 d_k d_2) \\
 &= M[1][1] + M[2][2] + d_0 d_1 d_2 \\
 &= 0 + 0 + 5 \times 2 \times 3 = 30
 \end{aligned}$$

$$M[2][3]$$

$$M[3][4]$$

$$M[4][5]$$

$$M[5][6]$$

# Chained Matrix Multiplication

- 단계 3:

- Example

- Diagonal 2

$$\begin{aligned}
 M[1][3] &= \min_{1 \leq k \leq 2} (M[1][k] + M[k+1][3] + d_0 d_k d_3) & M[2][4] \\
 &= \min(M[1][1] + M[2][3] + d_0 d_1 d_3, & M[3][5] \\
 &\quad M[1][2] + M[3][3] + d_0 d_2 d_3) & M[4][6] \\
 &= 64
 \end{aligned}$$

- Diagonal 1

$$\begin{aligned}
 M[1][4] &= \min_{1 \leq k \leq 3} (M[1][k] + M[k+1][4] + d_0 d_k d_4) & M[2][5] \\
 &= \min(M[1][1] + M[2][4] + d_0 d_1 d_4, & M[3][6] \\
 &\quad M[1][2] + M[3][4] + d_0 d_2 d_4, \\
 &\quad M[1][3] + M[4][4] + d_0 d_3 d_4) \\
 &= 132
 \end{aligned}$$

# Chained Matrix Multiplication

- 단계 3:

- Example

- Diagonal 4

$M[1][5]$

$M[2][6]$

- Diagonal 5

$M[1][6]$

$$\begin{array}{cccccc} A_1 & \times & A_2 & \times & A_3 & \times & A_4 & \times & A_5 & \times & A_6 \\ 5 \times 2 & & 2 \times 3 & & 3 \times 4 & & 4 \times 6 & & 6 \times 7 & & 7 \times 8 \\ d_0 & d_1 & d_1 & d_2 & d_2 & d_3 & d_3 & d_4 & d_4 & d_5 & d_5 & d_6 \end{array}$$



# Chained Matrix Multiplication

- 단계 3:
  - Algorithm:

```
int minmult (int n,
             const int d[],
             index P[][])
{
    index i, j, k, diagonal;
    int M[1..n][1..n];

    for (i = 1; i <= n; i++)
        M[i][i] = 0;
    for (diagonal = 1; diagonal <= n - 1; diagonal++) // Diagonal-1 is just
        for (i = 1; i <= n - diagonal; i++) {           // above the main
            j = i + diagonal;                             // diagonal.
            M[i][j] = minimum (M[i][k] + M[k + 1][j] + d[i - 1]*d[k]*d[j]);
                        i ≤ k ≤ j - 1
            P[i][j] = a value of k that gave the minimum;
        }
    return M[1][n];
}
```

# Chained Matrix Multiplication

- 단계 3:
  - Time Complexity Analysis
    - Basic operation:  $(n - \text{diagonal}) \times \text{diagonal}$
    - Input size :  $n$ , the number of matrices

$$T(n) = \sum_{\text{diagonal}=1}^{n-1} [(n - \text{diagonal}) \times \text{diagonal}]$$

$$= \frac{n(n-1)(n+1)}{6} \in \Theta(n^3)$$

```

for (i = 1; i <= n; i++)
    M[i][i] = 0;
for (diagonal = 1; diagonal <= n - 1; diagonal++) // Diagonal-1 is just
    for (i = 1; i <= n - diagonal; i++) {           // above the main
        j = i + diagonal;                             // diagonal.
        M[i][j] = minimum_{1 ≤ k ≤ j-1} (M[i][k] + M[k+1][j] + d[i-1]*d[k]*d[j]);
        P[i][j] = a value of k that gave the minimum;
    }
return M[1][n];
    
```

# Chained Matrix Multiplication

- 단계 4: (Reconstruction)
  - How an optimal order can be obtained?
    - In the above example, the optimal order is

$$(A_1((((A_2A_3)A_4)A_5)A_6))$$

- The matrix  $P[i][j]$  in the algorithm can be used to print the optimal order:
- $P[i][j]$  contains the value of  $k$  that gives the minimum, i.e., factorization is as follows:

$$A_i A_{i+1} \cdots A_{j-1} A_j = (A_i A_{i+1} \cdots A_k)(A_{k+1} \cdots A_{j-1} A_j)$$

# Chained Matrix Multiplication

- 단계 4: (Reconstruction)
  - Algorithm printing optimal order

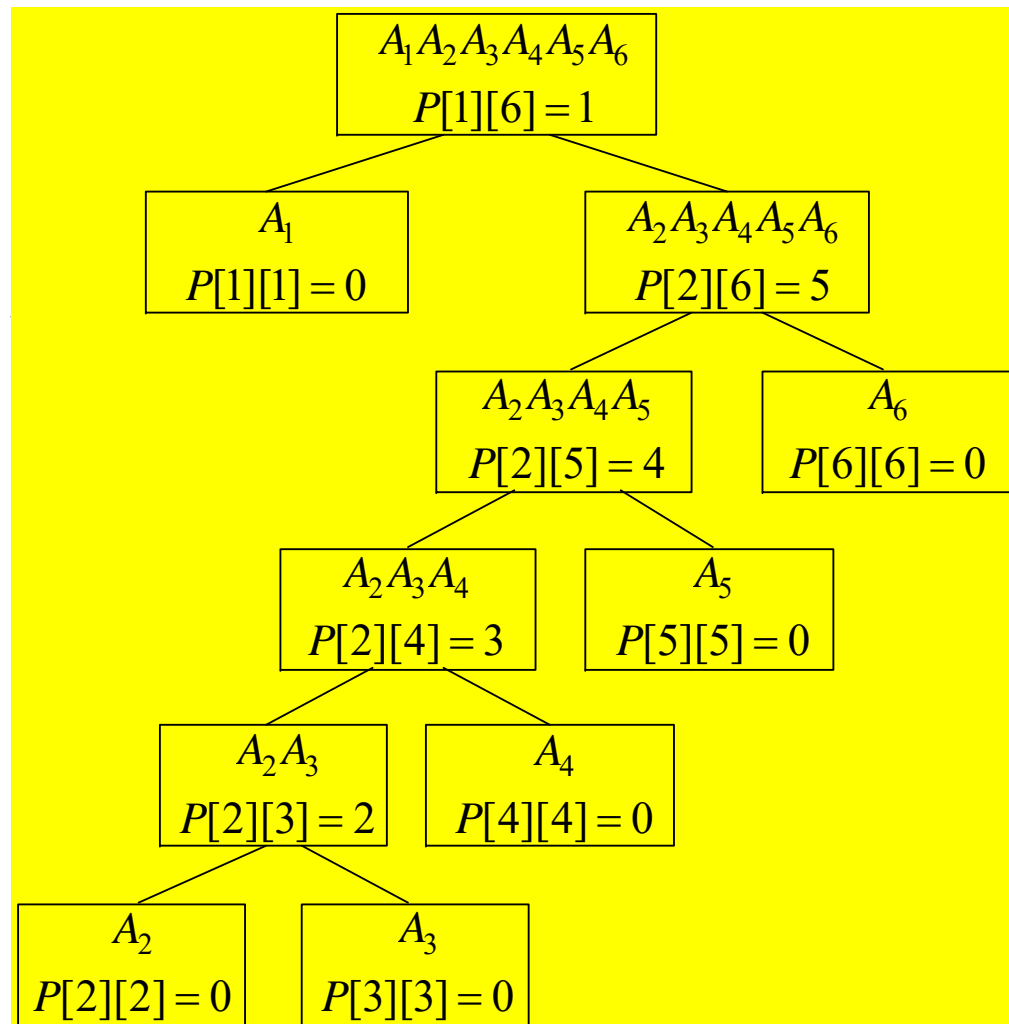
```
void order (index i, index j)
{
    if (i == j)
        cout << "A" << i;
    else {
        k = P[i][j];
        cout << "(";
        order(i, k);
        order(k + 1, j);
        cout << ")";
    }
}
```

Note that *order()* is a divide-and-conquer algorithm.

# Chained Matrix Multiplication

- 단계 4: (Reconstruction)
  - Example

	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5



# Homework

- Merging Files (파일합치기)
  - ICPC2015 (Korea, Internet Contest)

# Term Project

- Term Project
  - Sudoku
    - Solve with Dancing Link Data Structure
  - Crossword Puzzle
  - 진행방안
    - 조편성 : 6인 (경시대회 2팀을 합쳐)
    - 제안서 (Due Date : 11월 11일)
    - 최종보고서 (Due Date : 최종발표일 당일 제출)
    - 최종발표일 일정 및 장소
      - 최종발표 : 12월21일(월) 오후 6:00 – 오후 9:00
      - 장소 : 7호관 1층 114호실