

# 순환

- memoization
- backpointer

# 배경

- 재귀적 해법
  - 큰 문제에 닭음꼴의 작은 문제가 깃든다
  - 잘쓰면 보약, 못쓰면 맹독
    - 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다
    - 재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다

# 재귀적 해법의 빛과 그림자

- 재귀적 해법이 바람직한 예
  - 퀵정렬, 병합정렬 등의 정렬 알고리즘
  - 계승(factorial) 구하기
  - ...
- 재귀적 해법이 치명적인 예
  - 피보나치수 구하기
  - LCS, Pebbles, Matrix Chain..
  - ...

# 도입문제: 피보나치수 구하기

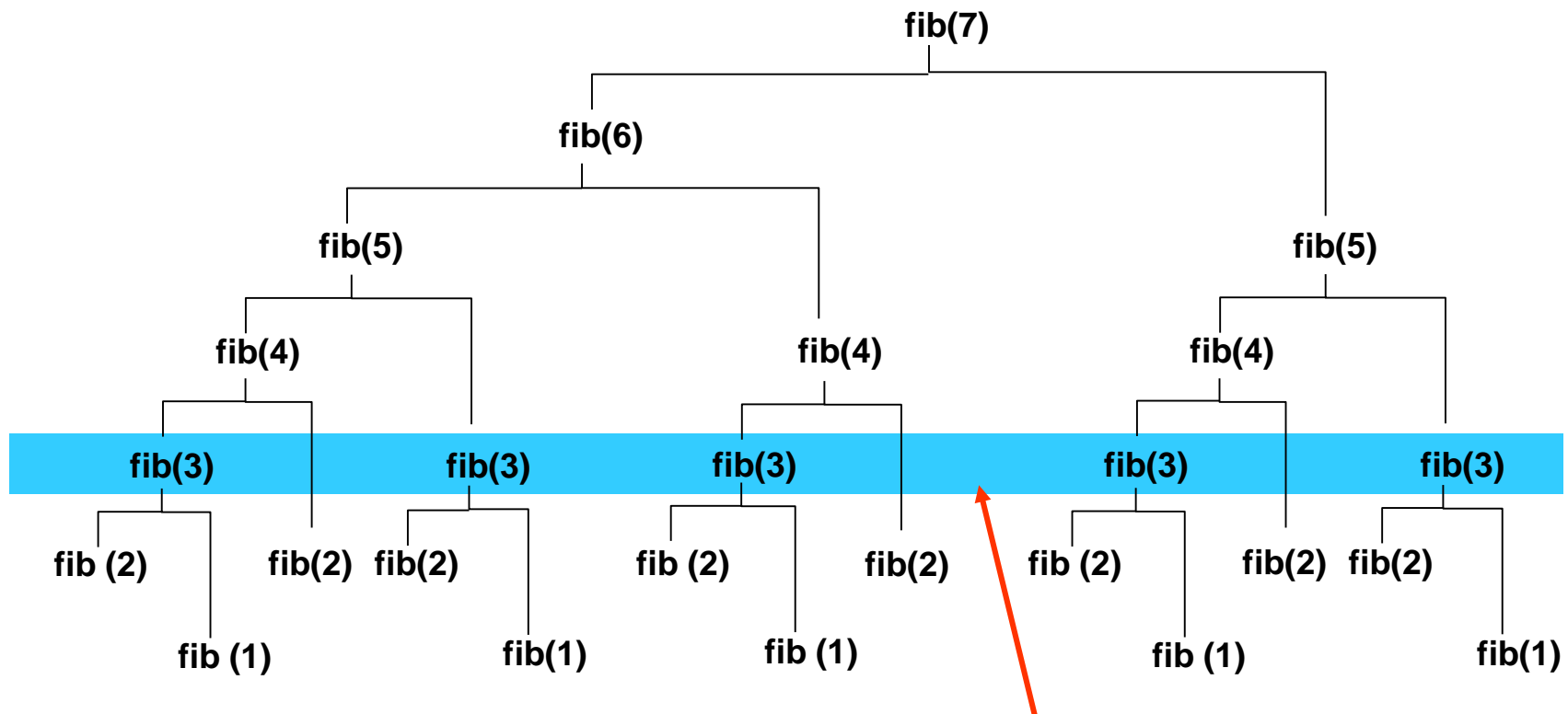
- $f_n = f_{n-1} + f_{n-2}$
- 아주 간단한 문제지만
  - Memoization의 동기와 구현이 다 포함되어 있다

# 피보나치수를 구하는 방법

```
fib(n)  
{  
    if (n = 1 or n = 2)  
        then return 1;  
    else return (fib(n-1) + fib(n-2));  
}
```

✓ 엄청난 중복 호출이 존재한다

# 피보나치 수열의 Call Tree



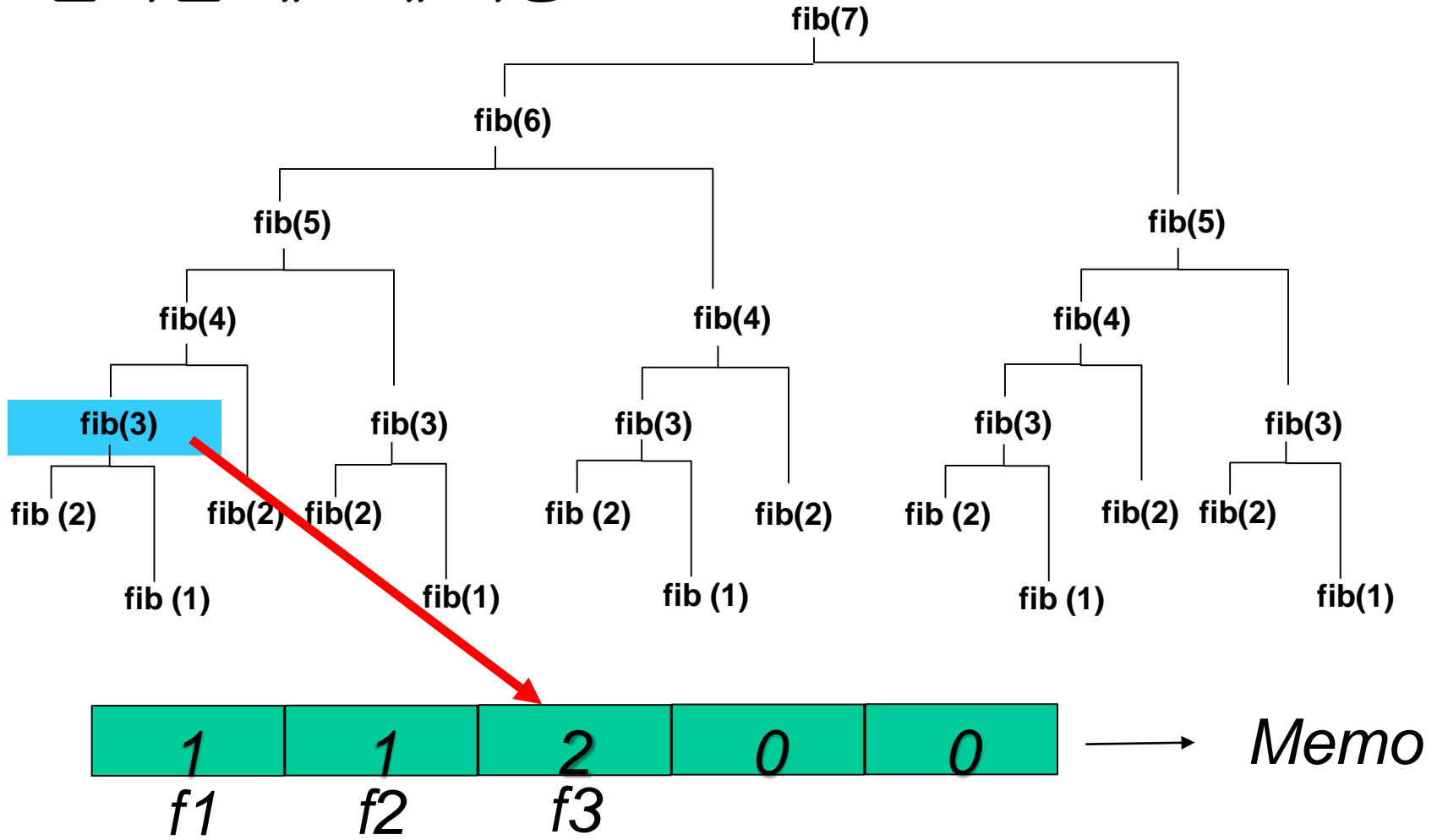
중복 호출의 예

# Memoization

중간에 호출의 반환값으로 계산된 결과를  
적당한 공간(메모)에 저장하고,  
재귀호출 전에 이미 계산된 결과가 있으면  
호출을 하지 않고 메모에 저장된 값을 이용한다.

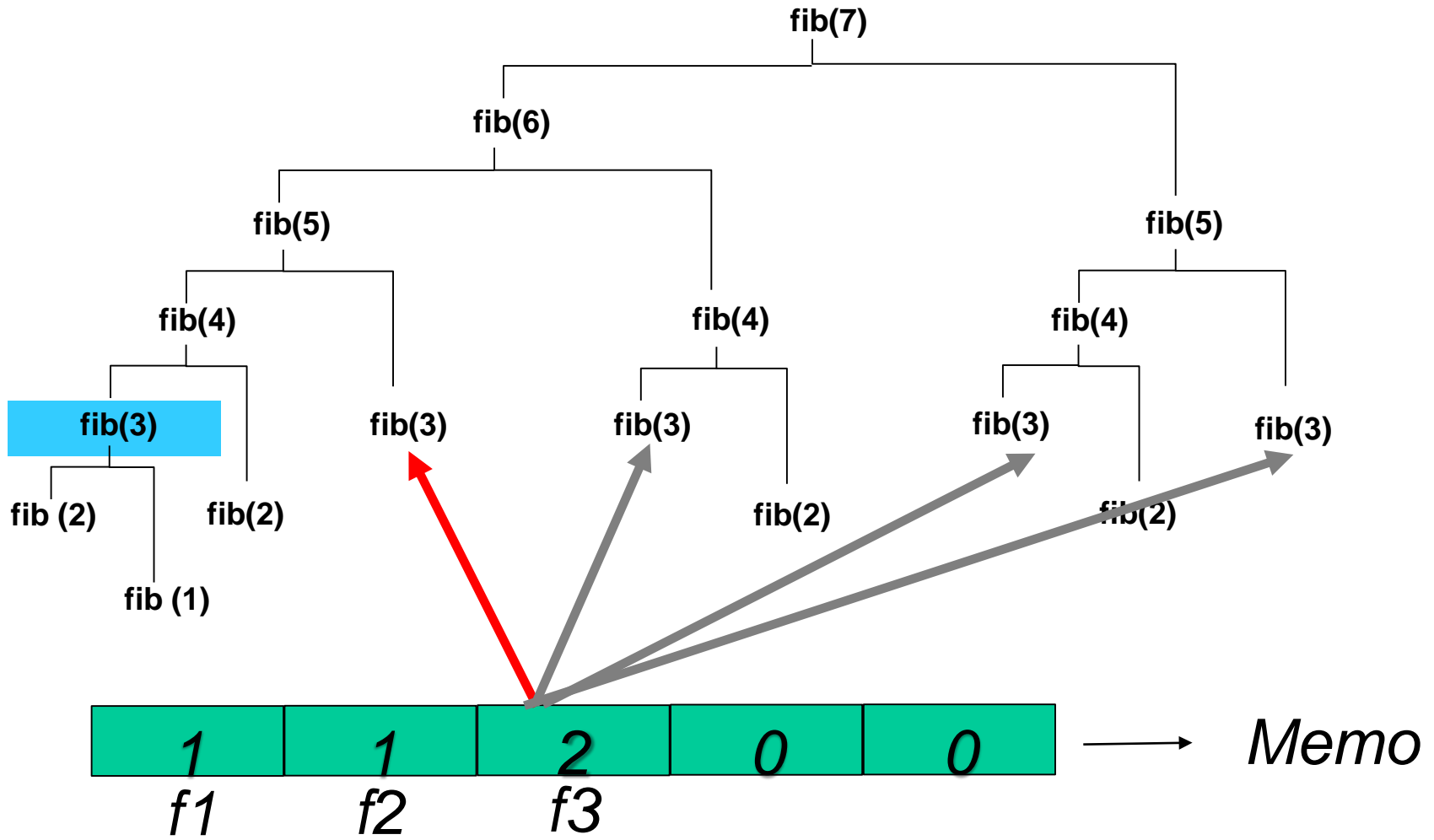
-중복 호출을 줄일 수 있다.

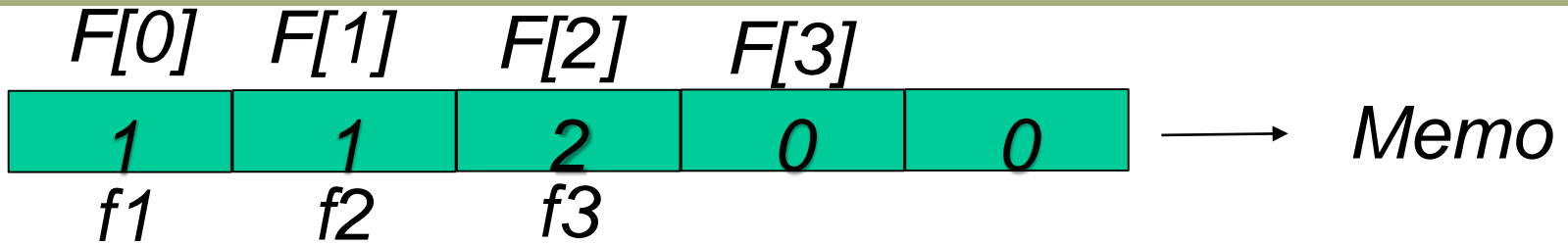
처음 계산은 재귀 호출을 통해 구하고  
그 결과를 메모에 저장





이후 호출은 메모에 기록된 값을 활용





$Memo \rightarrow \text{array } F \rightarrow \text{int} * F$

Memo space는 호출 전에 초기화

$F[0] = 1 ; // f_1$

$F[1] = 1 ; // f_2$

..

..

구하고자 하는 값은  $fn$

$fn = fn-1 + fn-2 \ (F[n-2] + F[n-3])$

```

int main(void)
{
    int n,i;
    int *F; //memo

    printf("입력:");
    scanf("%d",&n); //fn

    //memo 초기화
    F = (int*) malloc(sizeof(int) * (n + 1));
    for(i = 1; i <= n; i++)
        F[i] = 0; // F[1]부터 유의미한 값을 넣음, 0의 의미는

    printf("%d\n", fib(n, F)); //메모와 함께 재귀함수 호출
    free(F);
}

```

```

int fib(int n, int* F) // F[1]...F[n]까지 사용
{
    if( n == 1 || n == 2) return 1;
    if( F[n-1] == 0 ) // memo에서 fn-1 확인
        F[n-1] = fib(n-1,F); //memo에 없으면 호출

    if( F[n-2] == 0 ) // memo에서 fn-2 확인
        F[n-2] = fib(n-2,F); //memo에 없으면 호출

    return F[n-1] + F[n-2];
}

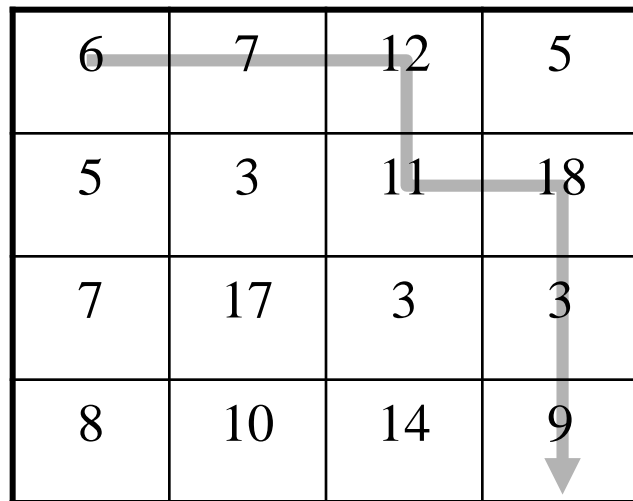
```

# 문제에 1: 행렬 경로 문제

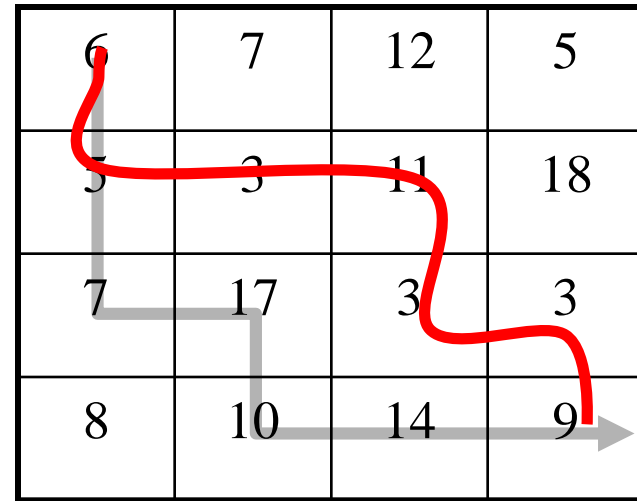
- 양 또는 음의 정수 원소들로 구성된  $n \times n$  행렬이 주어지고, 행렬의 좌상단에서 시작하여 우하단까지 이동한다. (이번 강의에서는 편의상 양수만)
- 이동 방법 (제약조건)
  - 오른쪽이나 아래쪽으로만 이동할 수 있다
  - 왼쪽, 위쪽, 대각선 이동은 허용하지 않는다
- 목표: 행렬의 좌상단에서 시작하여 우하단까지 이동하되, 방문한 칸에 있는 수들을 더한 값이 최소화되도록 한다.
  - 이번에는 그 **path**를 구한다.

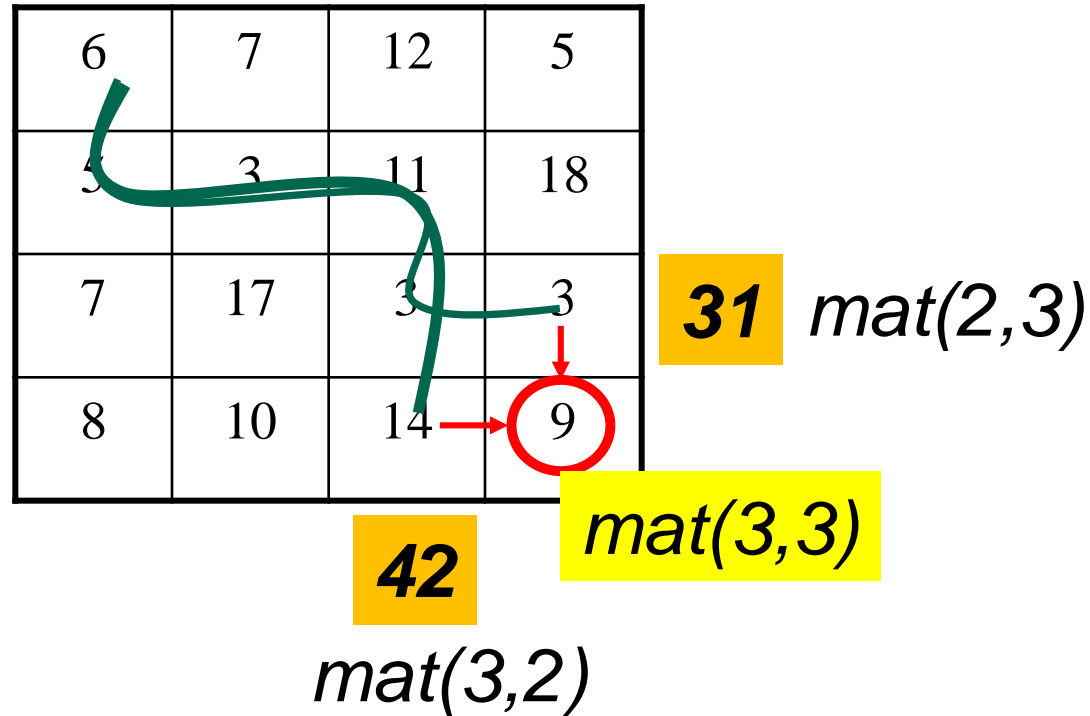
## 유효한 이동의 예

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9



6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9





$$mat(3,3) = \text{Min}(mat(2,3), mat(3,2)) + M[3][3] = 40$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

6	7	12	5	
5	3	11	18	
7	17	3	3	$mat(2,2)$
8	10	14	9	

$mat(3,1)$   $mat(3,2)$

$$mat(3,2) = \text{Min}(mat(3,1), mat(2,2)) + M[3][2]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수



6	7	12	5	
5	3	11	18	$mat(3,1)$
7	17	3	3	
8	10	14	9	

$mat(2,2)$   $mat(2,3)$

$$mat(2,3) = \text{Min}(mat(2,2), mat(3,1)) + M[2][3]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

$mat(0,1)$   $mat(0,2)$

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

$$mat(0,2) = mat(0,1) + M[0][2]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

*mat(1,0)*

*mat(2,0)*

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

$$mat(2,0) = mat(1,0) + M[2][0]$$

*int matrixPath(int i, int j)*

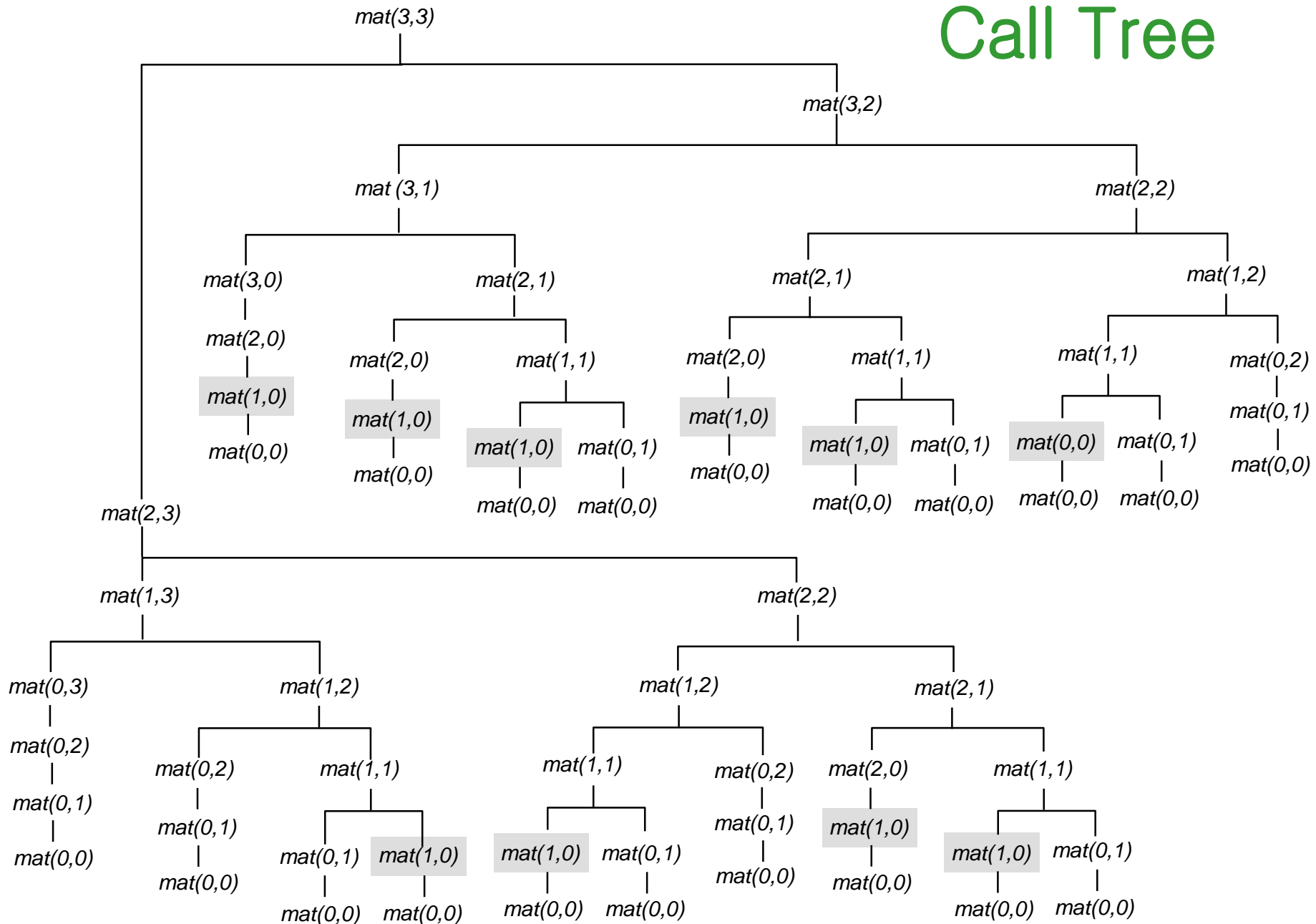
(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

# Recursive Algorithm

$\text{matrixPath}(M, i, j) \triangleright (i, j)$ 에 이르는 최저점수

```
{  
    if ( $i = 0$  and  $j = 0$ ) then return  $M[i][j]$ ; //  $M[0][0]$ 을 반환  
    else if ( $i = 0$ ) then return ( $\text{matrixPath}(0, j-1) + M[i][j]$ );  
    else if ( $j = 0$ ) then return ( $\text{matrixPath}(i-1, 0) + M[i][j]$ );  
    else return (( $\min(\text{matrixPath}(i-1, j), \text{matrixPath}(i, j-1)) + M[i][j]$ );  
}
```

# Call Tree



# Memoization

중간 계산 결과를 적당한 공간(메모)에 저장하고, 재귀호출 전에 이미 계산된 결과가 있으면 호출을 하지 않고 메모에 저장된 값을 이용한다.

- 중복 호출을 줄일 수 있다.

- **Memo도 2차원 !!!**

- **Mat(1,0), Mat(2,2) 등의 결과를 memo에 기록하고 접근할 수 있어야 함.**

# Memo 없는 코드

```
int main(void)
{
    int **m;
    int i, j, r, c;
    r = c = 4; // 4x4 matrix
    m = (int**) malloc( sizeof(int*) * r);
    for(i = 0; i < r; i++ )
        m[i] = (int*) malloc(sizeof(int) * c);
    for(i = 0; i < r; i++)
        for(j = 0; j < c; j++)
            scanf("%d", &m[i][j] );
    printf("%d\n", matrixPath(m, r, c, 3, 3) );
}
```

# Memo 없는 코드

```
int matrixPath(int **m, int r, int c, int i, int j )  
{  
    //앞의 알고리즘을 기반으로 작성한다.  
}
```



## Memo 있는 코드

```
int main(void)
{
    int **m, **M; // M은 메모
    int i,j,r,c;
    r=c=4; // 4x4 matrix
    m = (int**) malloc( sizeof(int*) * r );
    M = (int**) malloc( sizeof(int*) * r );
    for(i=0; i < r; i++ ) {
        m[i] = (int*) malloc(sizeof(int) * c );
        M[i] = (int*) malloc(sizeof(int) * c );
    }
    for(i=0; i < r; i++ )
        for(j=0; j < c; j++ ) {
            scanf("%d", &m[i][j] );
            M[i][j] = 0; //메모 초기화
        }
    printf("%d\n", matrixPath_memo(m,r,c,3,3,M) );
}
```

# Memo 있는 코드

```
int matrixPath_memo(int **m, int r, int c, int i, int j, int **M)
{
    // Lab에서 구현한다.
}
```

# Backpointer

계산 결과를 구할 때 이전의 중간 결과 중에서 어떤 결과를 선택하였는지를 기록.

-메모와 유사.

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

**31**  $mat(2,3)$

**42**  $mat(3,3)$

$mat(3,2)$

**BP**

			<b>UP</b>

**$BP[3][3] = UP$**

$$mat(3,3) = \text{Min}(mat(3,2), mat(2,3)) + M[3][3] = 40$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

$mat(2,2)$

$mat(3,1)$   $mat(3,2)$

**BP**

		UP	UP

**$BP[3][2] = UP$**

$$mat(3,2) = \text{Min}(mat(3,1), mat(2,2)) + M[3][2]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

$mat(3, 1)$

$mat(2, 2)$   $mat(2, 3)$

**BP**

			<b>LE</b>
		<b>UP</b>	<b>UP</b>

**$BP[2][3] = LE$**

$$mat(2, 3) = \text{Min}(mat(2, 2), mat(3, 1)) + M[2][3]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

$mat(0,1)$   $mat(0,2)$

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

**BP**

		<b>LE</b>	
			<b>LE</b>
		<b>UP</b>	<b>UP</b>

**$BP[0][2] = LE$**

$mat(0,2) = mat(0,1) + M[0][2]$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수

6	7	12	5
5	<i>mat(1,0)</i>		18
7	17	3	3
8	<i>mat(2,0)</i>		9

***BP***

		<b><i>LE</i></b>	
<b><i>UP</i></b>			<b><i>LE</i></b>
		<b><i>UP</i></b>	<b><i>UP</i></b>

$$BP[2][0] = UP$$

$$mat(2,0) = mat(1,0) + M[2][0]$$

*int matrixPath(int i, int j)*

(0,0)에서 (i,j)까지의 최저점수를 구하는 함수



6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9

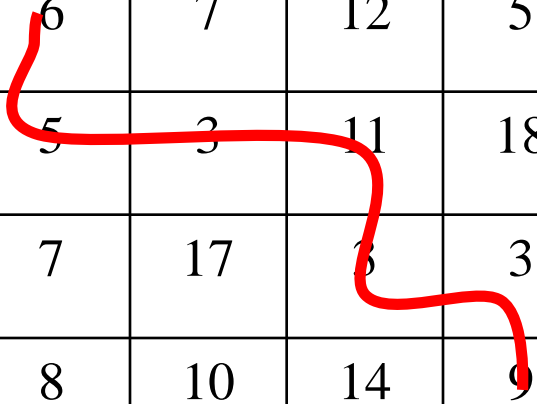
***BP***

<b><i>ST</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>

*void print\_path(int i, int j, BP)*

*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9



***BP***

<b><i>ST</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>

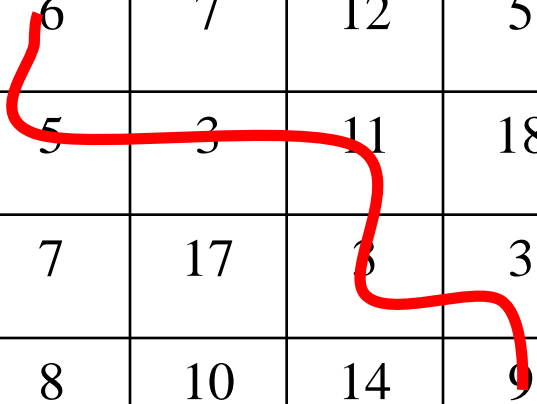


$$path(3,3) = path(2,3) + \text{"}\rightarrow <3,3>\text{"}$$

*void print\_path(int i, int j, BP)*

*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9



***BP***

<b><i>ST</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>

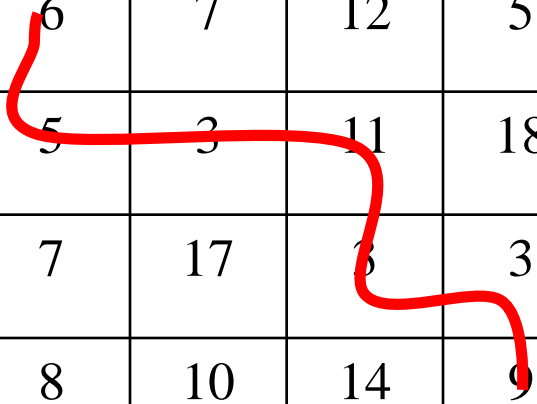


$$path(2,3) = path(2,2) + \text{"} \rightarrow \langle 2,3 \rangle \text{"}$$

*void print\_path(int i, int j, BP)*

*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9



***BP***

<b><i>ST</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>

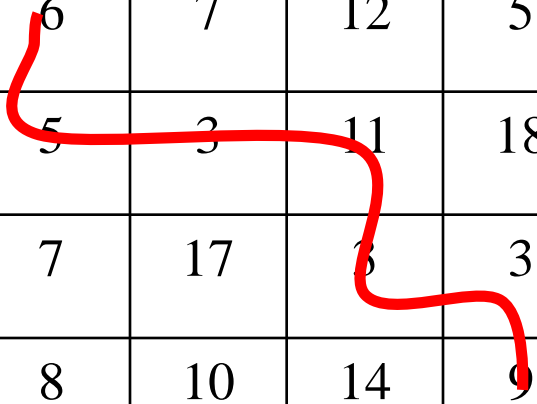


$$path(2,2) = path(1,2) + \text{"}\rightarrow \langle 2,2 \rangle\text{"}$$

*void print\_path(int i, int j, BP)*

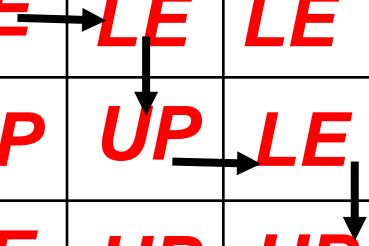
*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9



***BP***

<b><i>ST</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>	<b><i>LE</i></b>
<b><i>UP</i></b>	<b><i>LE</i></b>	<b><i>UP</i></b>	<b><i>UP</i></b>

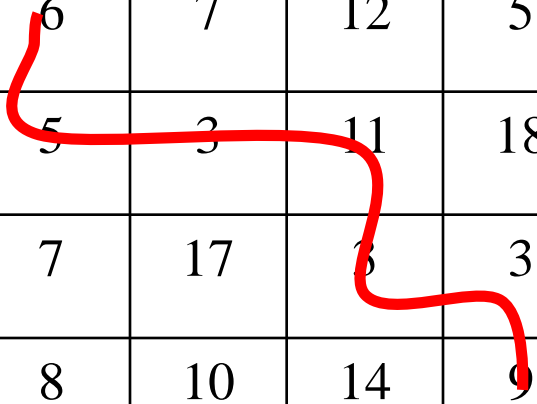


$$path(2,2) = path(1,2) + \text{"} \rightarrow \langle 2,2 \rangle \text{"}$$

*void print\_path(int i, int j, BP)*

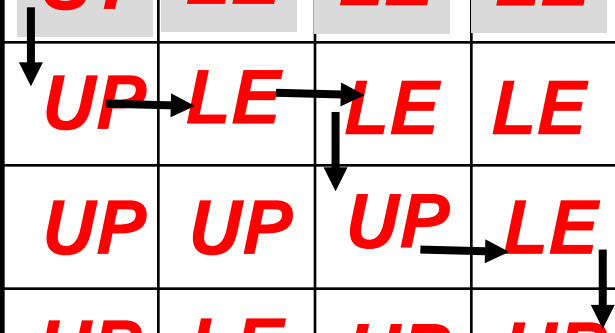
*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

6	7	12	5
5	3	11	18
7	17	8	3
8	10	14	9



**BP**

<b>ST</b>	<b>LE</b>	<b>LE</b>	<b>LE</b>
<b>UP</b>	<b>LE</b>	<b>LE</b>	<b>LE</b>
<b>UP</b>	<b>UP</b>	<b>UP</b>	<b>LE</b>
<b>UP</b>	<b>LE</b>	<b>UP</b>	<b>UP</b>



$$path(0, 1) = path(0, 0) + \text{"} \rightarrow \langle 0, 1 \rangle \text{"}$$

*void path(int i, int j, BP)*

*(0,0)에서 (i,j)까지의 최저점수 path를 출력하는 함수*

## Backpointer를 활용하여 path를 출력

```
void print_path(int i, int j , int **BP)
{
    if ( BP[i][j] == UP )
        print_path(i-1, j, BP);
    else if ( BP[i][j] == LE )
        print_path(i, j-1, BP);
    printf("<%d,%d> " , i, j);
}
```