연습

- 크기가 NxN인 2차원 배열 A에 다음과 같은 순서로 접근해 숫자를 저장하시오.

A	1	2	3	4	5	j
1		1	5	8	10	
2			2	6	9	
3				3	7	
4					4	
5						
i						

N=5인 경우

- 접근 방법

인덱스로 표시한 접근 순서

1	2	3	4	→행과 열의 차이 m
1, 2 (1)	1, 3 (5)	1, 3 (8)	1, 5 (10)	
2, 3 (2)	2, 4 (6)	2, 4 (9)		
3, 4 (3)	3, 5 (7)			
4, 5 (4)				
4	3	2	1	마지막 행 번호 N-I

```
k ← 0
for m : 1 → N−1                    // i, j 차이
    for i : 1 → N−m                // i
        j ← i+m                    // j
            A[i][j] ← ++k
```

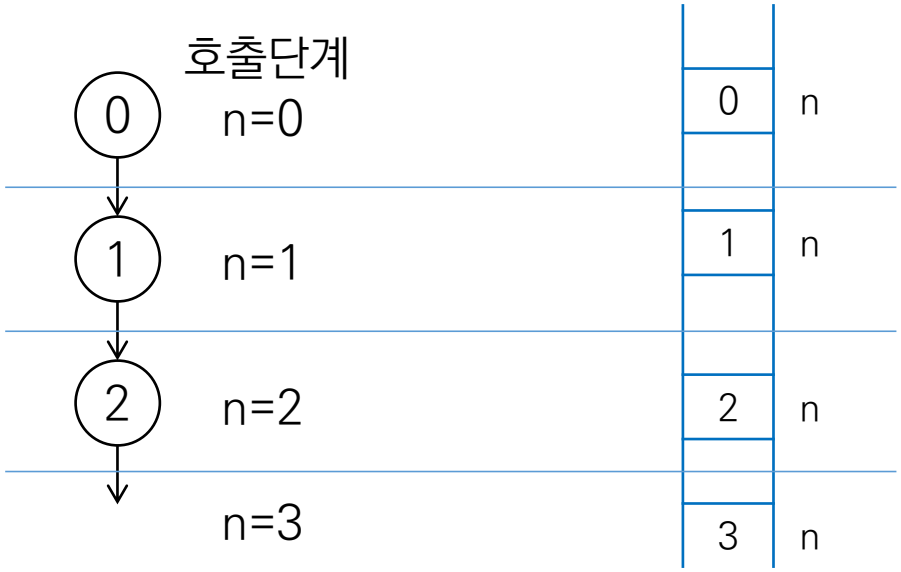
배열과 재귀호출

재귀호출의 기본

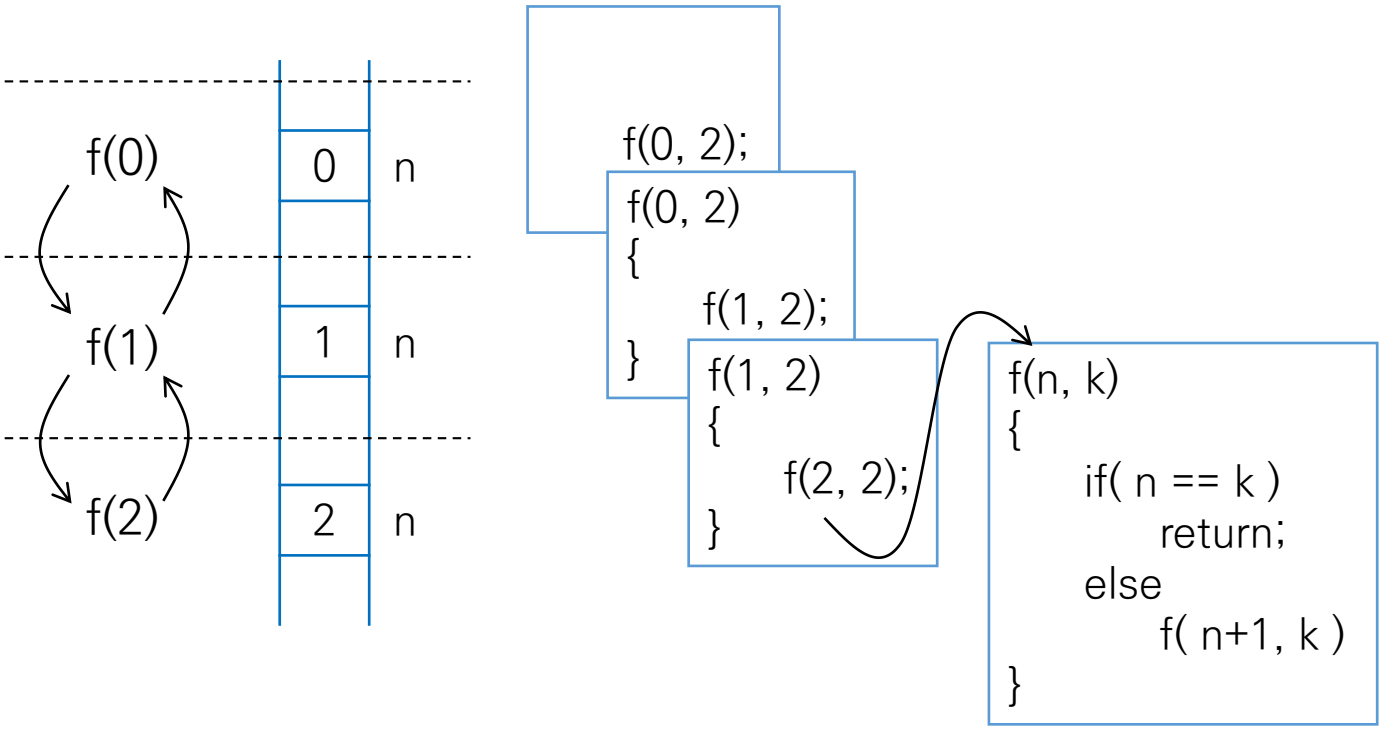
- 특징
  - 자기 자신을 호출하지만 사용하는 메모리 영역이 구분되므로 다른 함수를 호출하는 것과 같음.
  - 정해진 횟수만큼, 혹은 조건을 만족할 때 까지 호출을 반복함.

기본적인 재귀 호출

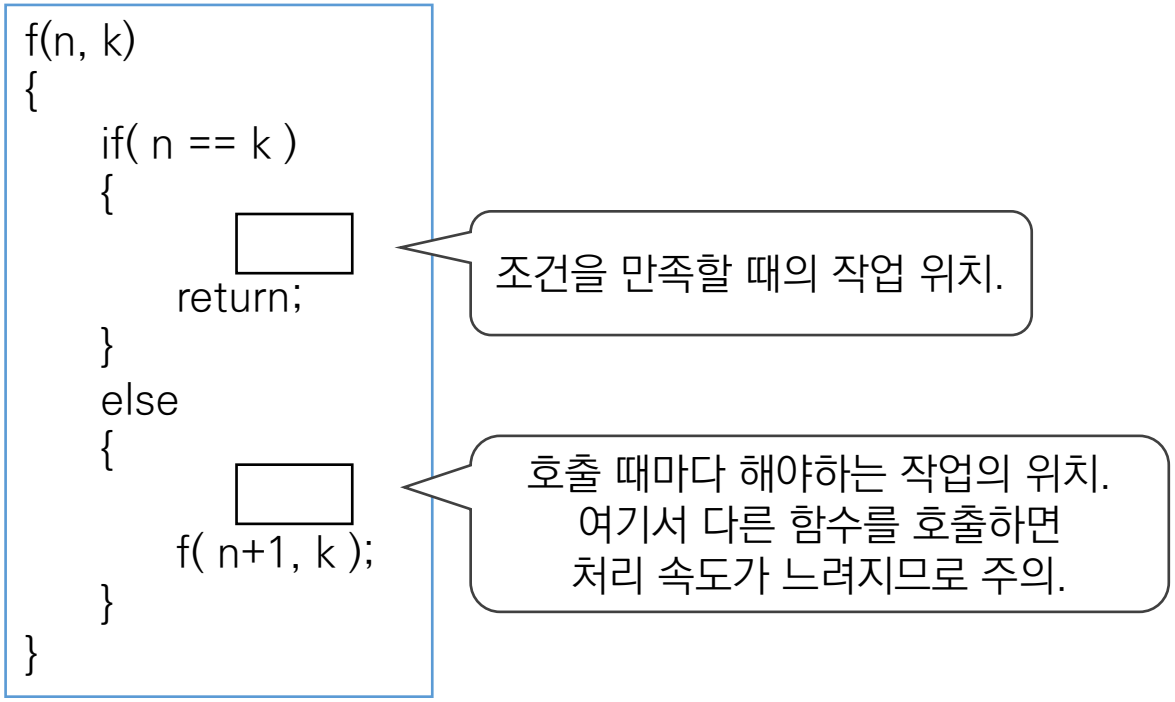
```
f(n, k)
{
      ...
      f(n+1, k);
}
```



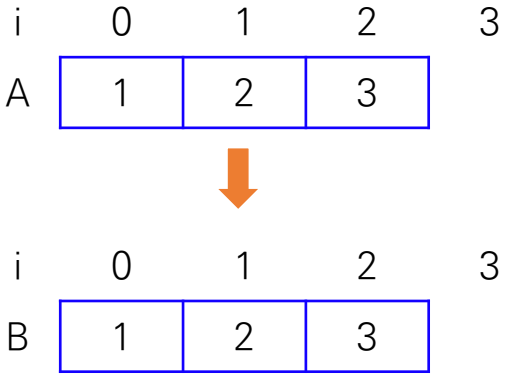
- 정해진 횟수만큼 호출하기
  - 호출 횟수에 대한 정보는 인자로 전달.
  - 정해진 횟수에 다다르면 호출 중단.



- 재귀함수의 구조
  - 재귀호출 단계마다 해야하는 작업 -> 재귀 호출 부분.
  - 재귀호출 완료 시 해야하는 작업 -> 재귀 호출 종료 부분.



- 재귀호출을 이용해 배열 복사하기
  - 호출 단계 n을 배열 인덱스로 활용.
  - 배열의 크기와 호출 단계가 같아지면(n==k) 재귀호출 중단, 배열 출력.
  - 배열의 크기가 재귀 호출의 횟수를 결정.



f(0, 3);

```
f( n, k )
{
    if( n == k )
    {
        printArray( );
        return;
    }
    else
    {
        B[n] = A[n];
        f( n+1, k );
    }
}
```

원하는 조건을 찾으면 중단하는 경우

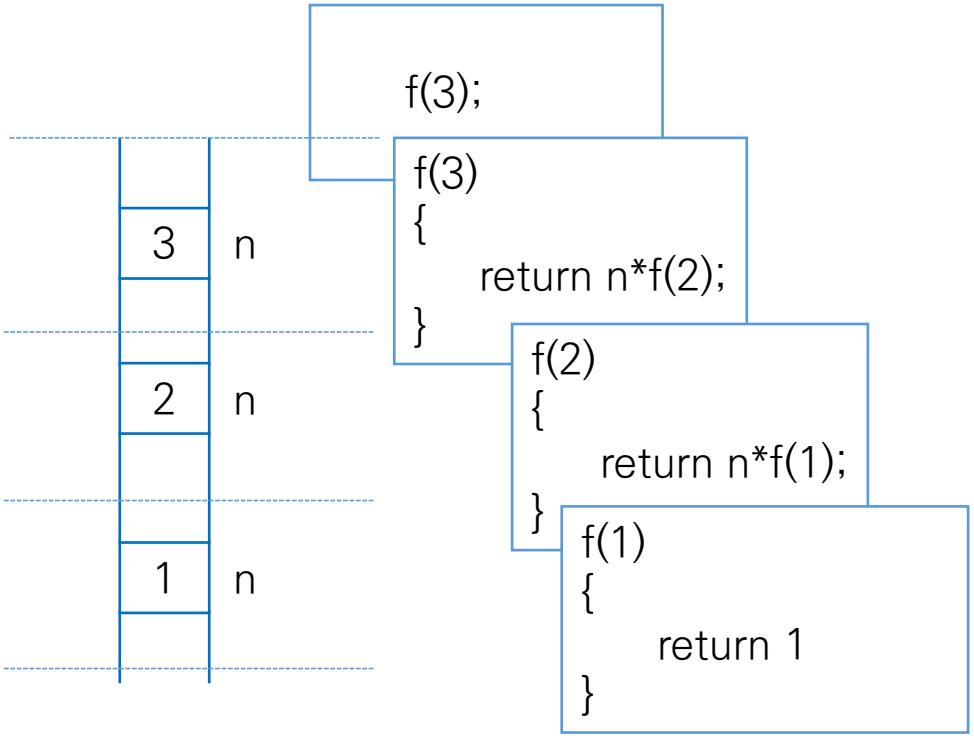
- 주어진 집합에 v가 들어있으면 1, 없으면 0을 리턴하는 재귀 호출.

a[ ] = { 3, 7, 6, 2, 1, 4, 8, 5}
v = 2

```
find ( n, k, v )
{
      if( n == k) // 배열을 벗어남
          return 0;
      else if ( a[n] == v)
          return 1;
      else
          r = f( n+1, k, v);
      }
```

- 리턴 값을 사용하는 재귀 호출
  - 팩토리얼 계산
    - 3! = 3*2*1 = 3 * 2!
    - f(n) = n * f(n-1); // n 〉 0;
    - 0! = 1;

```
long f(int n)
{
      if( n 〈 2 )
          return 1;
      else
          return n * f(n-1);
}
```



✓ 반복 구조의 팩토리얼 계산

```
// N!을 구하는 경우
long fact [N];

fact[0] = 1;
fact[1] = 1;
for i : 2 -> N
    fact[i] = i*fact[i-1];
```

✓ 활용 예

```
// N!을 1000000007로 나눈 나머지를 구하는 경우
long fact [N];

fact[0] = 1;
fact[1] = 1;
for i : 2 -> N
    fact[i] = i*fact[i-1] % 1000000007;
```

재귀 호출이 두 번인 경우

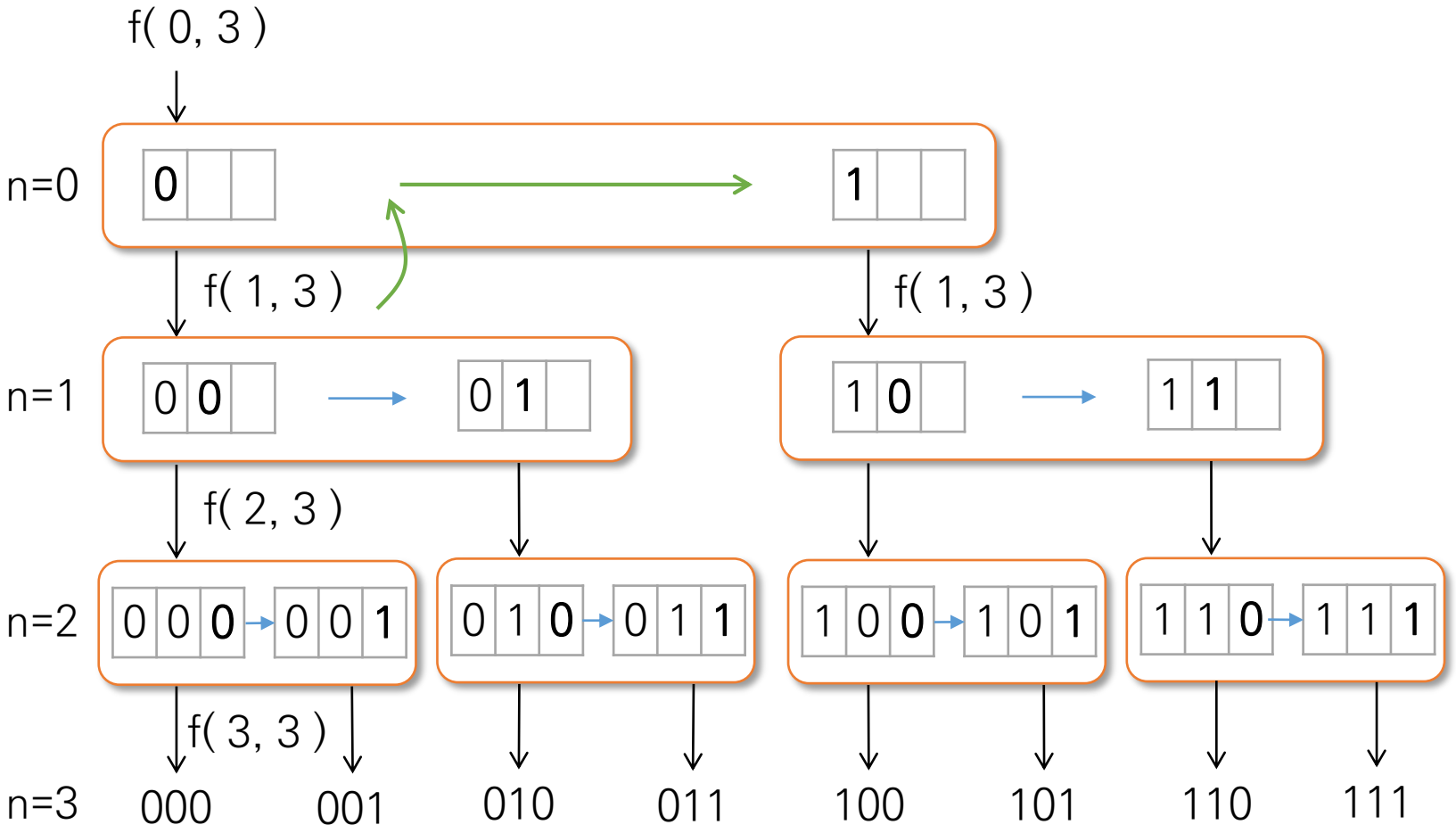
- 배열 L의 각 자리에 0/1이 오는 모든 경우 만들기.

호출단계 →	0	1	2
L	0/1	0/1	0/1
	0	0	0
	0	0	1
		…	
	1	1	0
	1	1	1

```
f( n , k )
{
    if( n == k )
    {
        …
    }
    else
    {
        L[n] = 0;
        f( n+1, k );
        L[n] = 1;
        f( n+1, k );
    }
}
```

L[n]의 두 가지 값에 따라 각각 호출

- f(n, k)호출에서 호출 단계 n과 배열크기 k의 활용



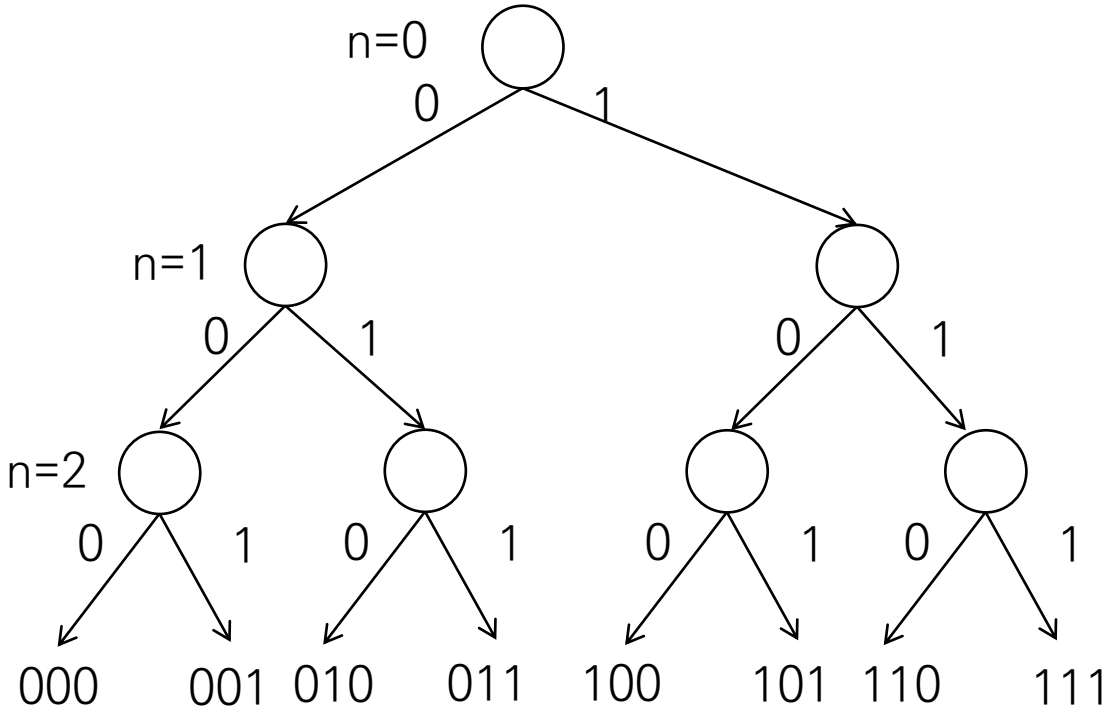
- 호출 깊이 n, 채울 배열의 크기 k, 배열 L
  - L[k]가 0, 1인 경우에 대해 각각 L[k+1] 결정 단계 호출.

```
int L[N];

f( n, k)
{
    if( n==k )
        //배열 L 출력
    else
        L[n] = 0;
        f( n+1, k );
        L[n] = 1;
        f( n+1, k );
}
```

```
int L[N];

f( n, k)
{
    if( n==k )
        //배열 L 출력
    else
        for( i : 0 -> 1)
            L[n] = i;
            f( n+1, k );
}
```

그림에서 수평으로 나타낸 화살표는 for문으로 처리한다.

- 트리 형태로 표현



연습

- {1, 2, 3}의 모든 부분 집합 출력하기.
  - { } {1} {2} {3} {1, 2} {1, 3} {2, 3} {1, 2, 3}
  - 각 원소의 포함 여부를 1/0으로 표시할 수 있음.

<table>
<tr><th></th><th>1</th><th>2</th><th>3</th><th colspan="3">포함 여부를 1과 0으로 표시</th></tr>
<tr><td>{1, 2, 3}</td><td rowspan="4">포함</td><td rowspan="2">포함</td><td>포함</td><td rowspan="4">1</td><td rowspan="2">1</td><td>1</td></tr>
<tr><td>{1, 2}</td><td>미포함</td><td>0</td></tr>
<tr><td>{1, 3}</td><td rowspan="2">미포함</td><td>포함</td><td rowspan="2">0</td><td>1</td></tr>
<tr><td>{1}</td><td>미포함</td><td>0</td></tr>
<tr><td>{2, 3}</td><td rowspan="4">미포함</td><td rowspan="2">포함</td><td>포함</td><td rowspan="4">0</td><td rowspan="2">1</td><td>1</td></tr>
<tr><td>{2}</td><td>미포함</td><td>0</td></tr>
<tr><td>{3}</td><td rowspan="2">미포함</td><td>포함</td><td rowspan="2">0</td><td>1</td></tr>
<tr><td>{ }</td><td>미포함</td><td>0</td></tr>
</table>

연습

- 전기 버스

충전지를 교환하는 방식의 전기버스를 운행하려고 합니다. 정류장에는 교체용 충전지가 있는 교환기가 있고, 충전지마다 최대로 운행할 수 있는 정류장 수가 표시되어 있습니다. 교체에는 시간이 걸리기 때문에 최소한의 교체 횟수로 목적지에 도착해야 합니다. 정류장과 충전지에 대한 정보가 주어질 때, 목적지에 도착하는데 필요한 최소한의 교환횟수를 구해보세요. (출발지에서는 항상 교체, 단 교환 횟수에서 제외.)

풀이 예시)

정류장	1	2	3	4	5
충전지	2	3	1	1	

출발지에서의 용량이 2이므로 2번이나 3번 정류장에서는 교체를 해야 합니다. 2번에서 교체하는 경우 마지막 정류장까지 갈 수 있으므로 교환횟수는 1번입니다. 3번에서 교체하는 경우 4번에서 또 교체해야 하므로, 5번에 도착하기 까지 2번의 교환이 필요합니다.

■ 입력

입력의 맨 첫 줄에는 전체 테스트케이스 개수가 주어지고, 각 줄의 첫 번째는 정류장의 개수 N, 이후에는 마지막 정류장을 제외한 정류장의 충전지 용량 B가 주어집니다. 충전지의 용량은 정류장 수보다 작습니다.
3 〈= N 〈= 10, 1〈=B〈N

```
3
5 2 3 1 1
10 2 1 3 2 2 5 4 2 1
10 1 1 2 1 2 2 1 2 1
```

■ 출력

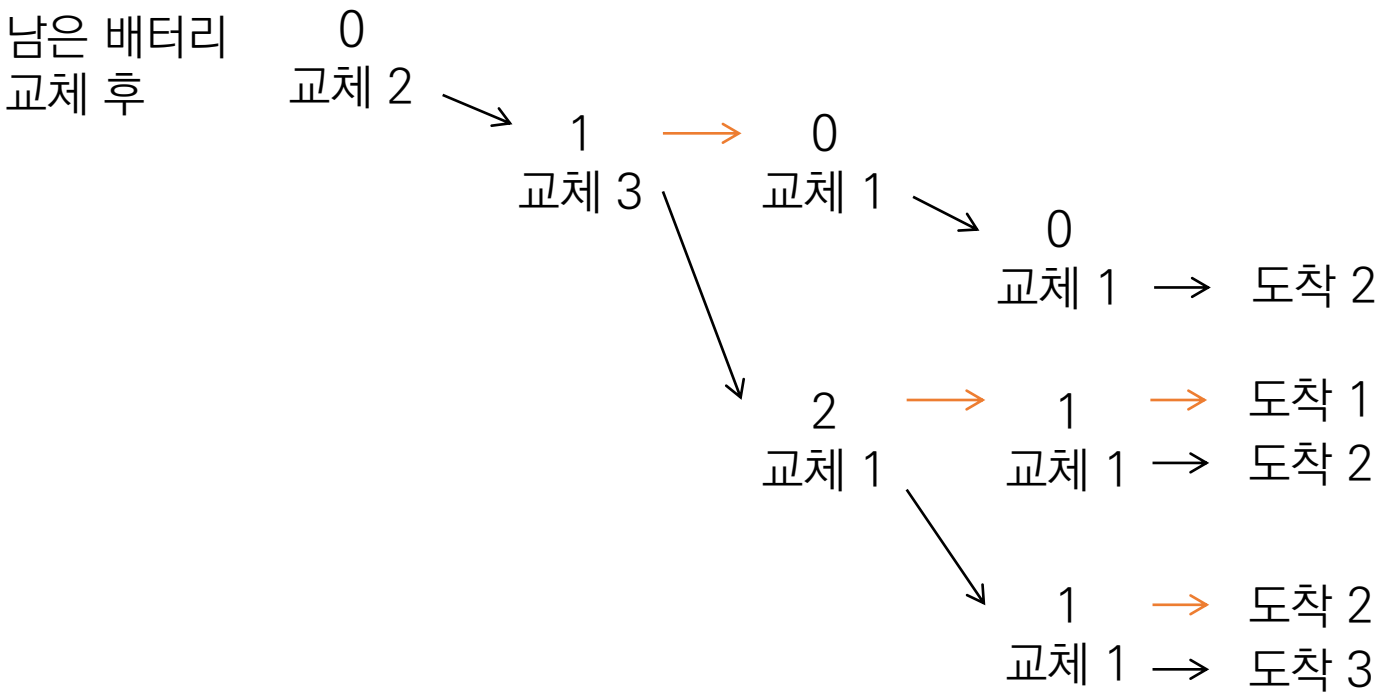
#테스트케이스 번호와 교환 횟수를 출력합니다.

```
#1 1
#2 2
#3 5
```

힌트

- 각 정류장에서는 교체와 통과 두 가지 선택이 가능.

정류장	1	2	3	4	5
충전지	2	3	1	1	



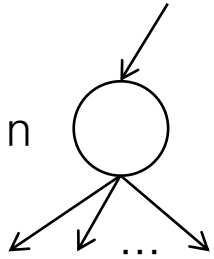
- 두 재귀호출의 리턴 값을 사용하는 경우.
  - 피보나치 수열
    - f(n) = f(n−1) + f(n−2);
    - f(0) = 0, f(1) = 1;

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

식을 그대로 옮기면 간단한 대신
연산횟수가 너무 많다.

* 위키의 피보나치 수 프로그램 참조.

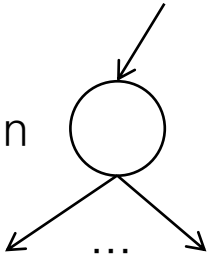
재귀 호출이 여러 번인 경우



항상 j번인 경우

```
        …
        else
            for i : 1 –〉 j
                L[n] = i;
                f( n+1, k);
}
```

예) 1, 2, 3을 중복 사용해 3자리 숫자 만들기



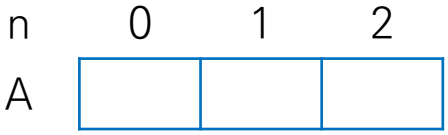
조건에 따라 달라지는 경우

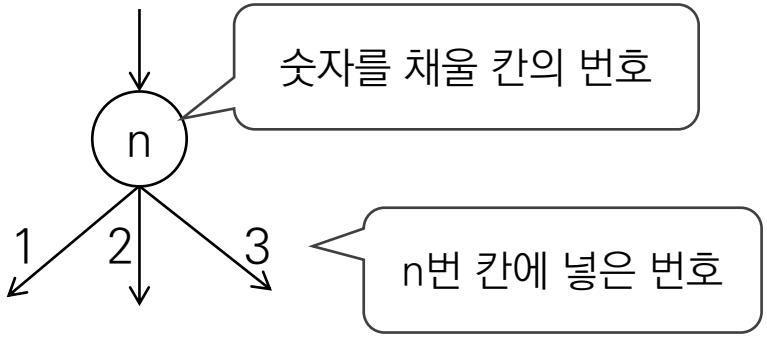
```
        …
        else
            for i : 1 –〉 j
                if ( i가 유효하면 )
                    L[n] = i;
                    f( n+1, k )
}
```

예) 그래프에서 인접 노드에 방문하기

연습

- 1, 2, 3을 중복 사용해 3자리 숫자 만들기.

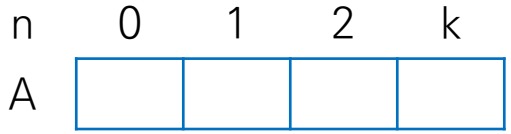


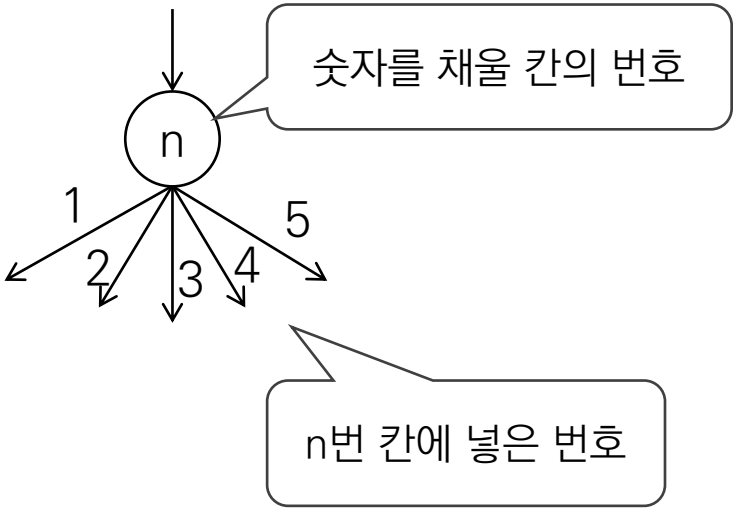



```
f(n, k)
{
      if( n == k)
            printA( );
            return;
       else
            for i : 1 -〉 3
                  A[n] = i;
                  f( n+1 );
}
```

연습

- 1, 2, 3, 4, 5중 3개의 숫자를 중복 사용해 3자리 숫자 만들기.



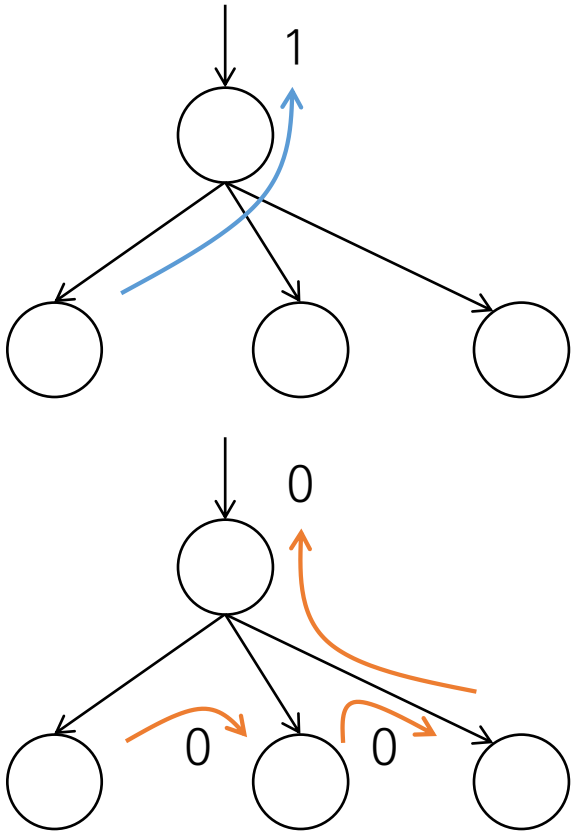



```
f(n, k)
{
    if( n == k)
        printA( );
        return;
     else
        for i : 1 -〉 5
            A[n] = i;
            f( n+1 );
}
```

원하는 조건을 찾으면 중단하는 경우

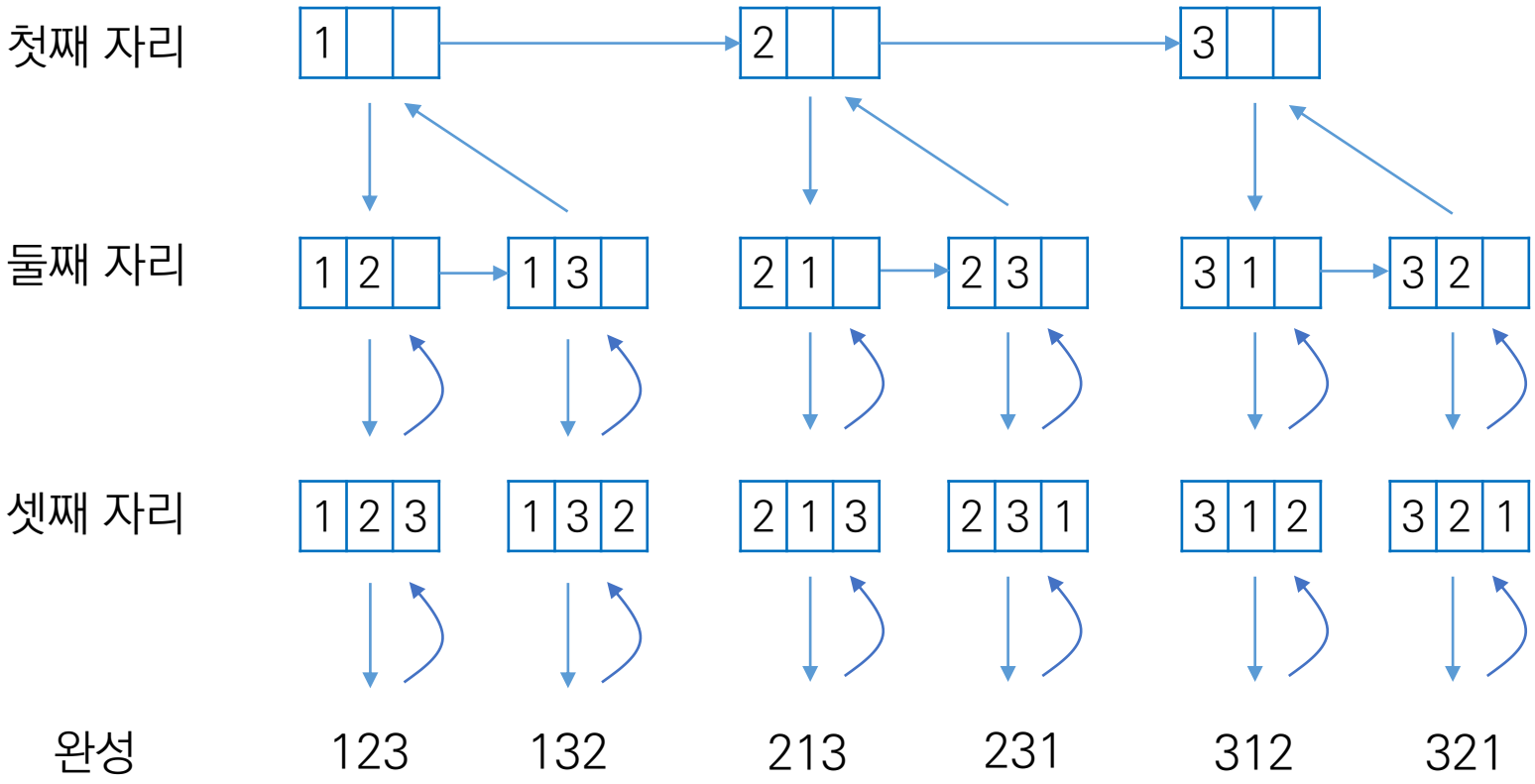
- 조건을 찾으면 1, 못 찾으면 0을 리턴.

```
if ( 답을 찾은 중단 조건 )
      …
      return 1;
else if( n == k)
      return 0;
else
      … // 현재 단계에서 처리할 일
      for i : 0 -〉 j
            r = f( n+1, k);
            if( r == 1)
                  return 1;
      return 0;
}
```

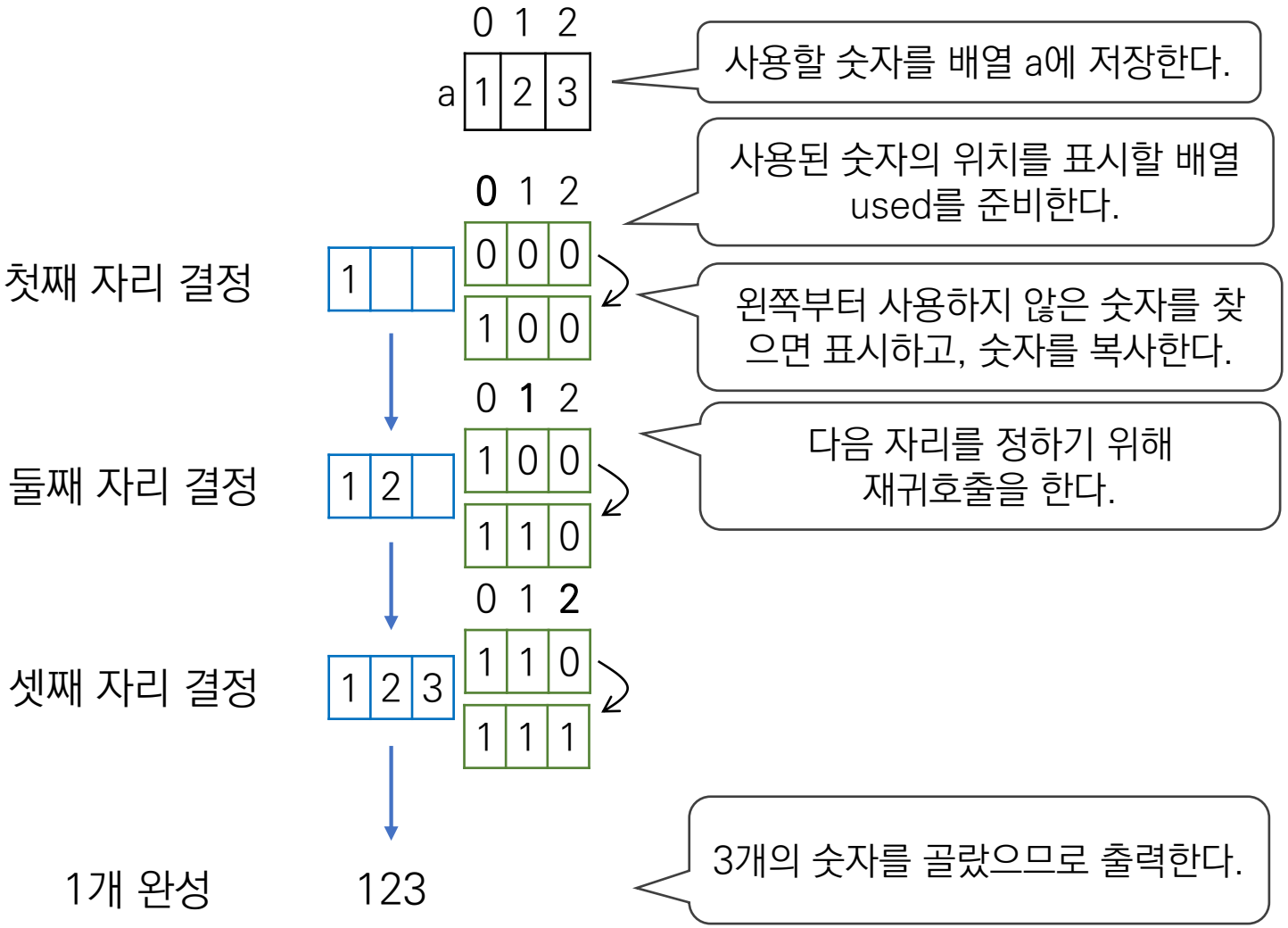


호출 횟수가 변하는 재귀 호출

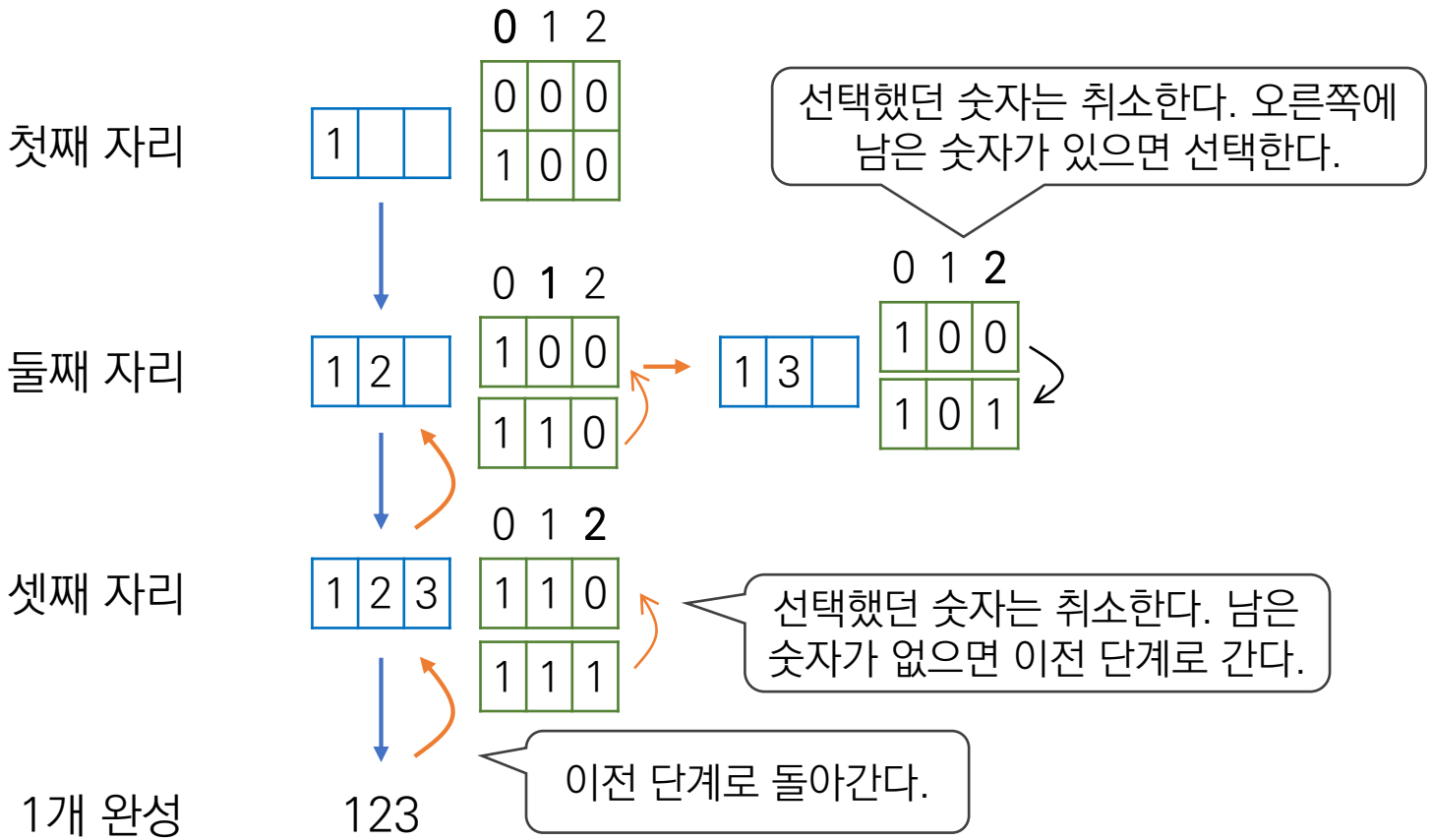
- 1, 2, 3으로 3자리 수 만들기



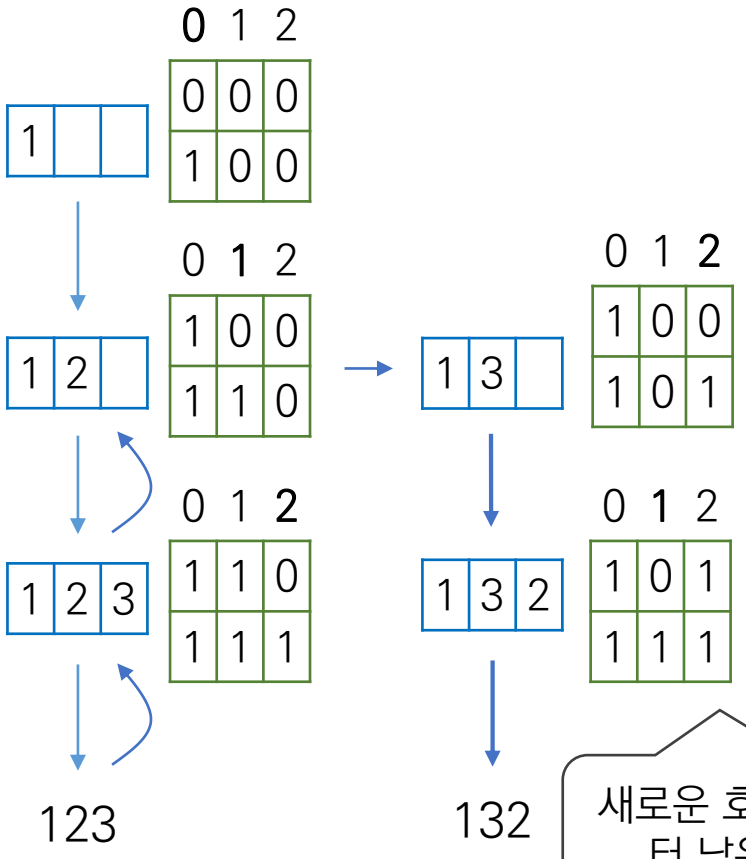
- 1, 2, 3으로 3자리 수 만들기



- 1, 2, 3으로 3자리 수 만들기 (계속)



- 1, 2, 3으로 3자리 수 만들기 (계속)



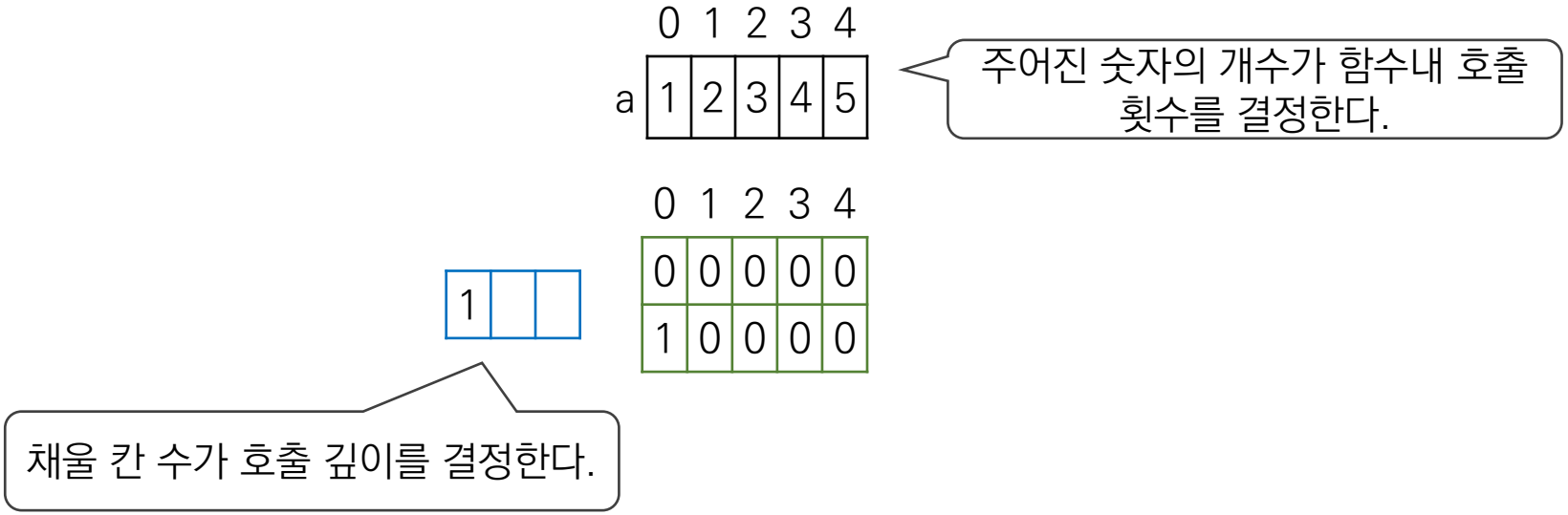
n : 선택한 숫자를 넣을 자리
k : 사용할 숫자 개수
i ; 사용하지 않은 숫자를 찾는 순서

```
for (int i = 0; i < k; i++)
{
    if (used[i] == 0)
    {
        used[i] = 1;
        p[n] = a[i];
        f(n+1, k);
        used[i] = 0;
    }
}
```

새로운 호출에서는 맨 왼쪽부터 남은 숫자를 찾는다.

연습

- {1,2,3,4,5}의 원소를 한번씩만 사용해 3자리 수 만들기.
  - n : 고른 숫자를 저장할 위치.
  - k : 골라야 할 숫자의 개수.
  - m : 고를 수 있는 숫자의 개수.



- n, k는 그대로 for에서 선택할 수 있는 개수만 늘어남

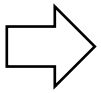
```
f(n, k, m)
{
    if( n == k )
        …
    else
    {
        for (int i = 0; i < m; i++)
        {
            if (used[i] == 0)
            {
                used[i] = 1;
                p[n] = a[i];
                f(n+1, k);
                used[i] = 0;
            }
        }
    }
}
```

연습

- A, B, C 사람이 3개의 일을 처리하는 시간이 각각 다르다고 한다. 각자 한가지 일을 한다고 할 때, 최소인 시간의 합을 구하라.

	1	2	3
A	13	8	10
B	7	10	12
C	12	8	10

개인별 소요시간



1	2	3	합계
A	B	C	33
A	C	B	33
B	A	C	25
B	C	A	25
C	B	A	32
C	A	B	32

배정에 순열을 활용

- 힌트

소요 시간

t	0	1	2	일
0	13	8	10	
1	7	10	12	
2	12	8	10	
사람				

순열 생성

	0	1	2	일
p				사람

예)

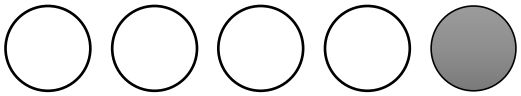
	0	1	2	일
p	1	0	2	사람

p[0] -〉 0번 일을 맡은 사람.

t[p[0]][0] -〉 0번 일을 맡은 사람이 일을 하는 데 걸리는 시간.

호출의 깊이가 변하는 재귀 호출

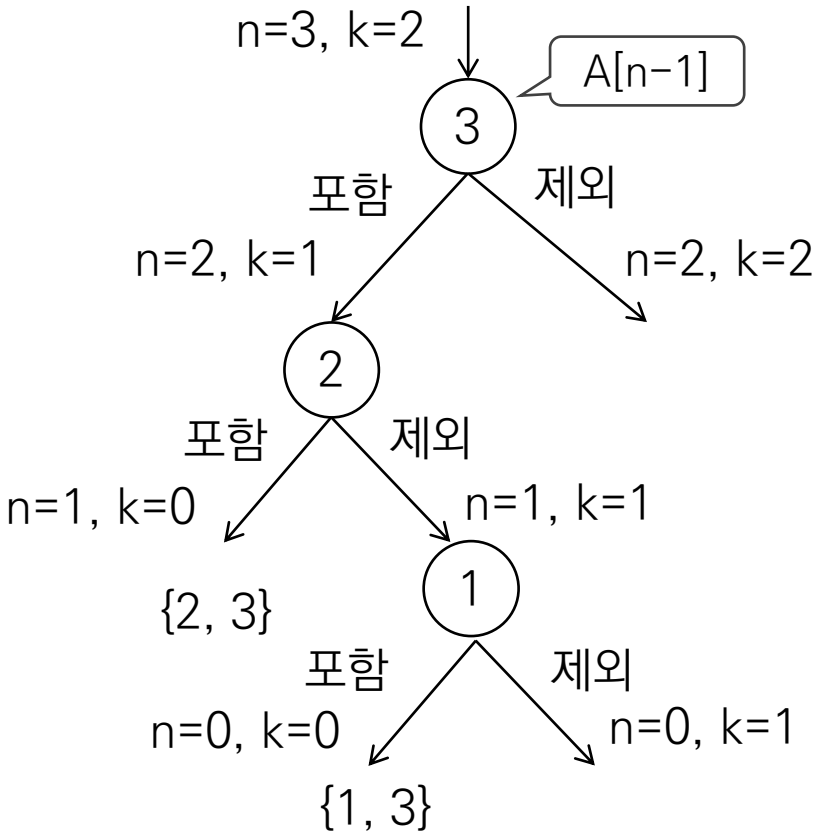
- 조합만들기
  - n개에서 k개를 고르는 경우의 수 : nCk
    - {1, 2, 3}에서 두 개의 숫자를 고르는 경우의 수 : ${}_3C_2$
      - {1, 2}, {1, 3}, {2, 3}
  - 두 경우로 나눠 생각할 수 있다.



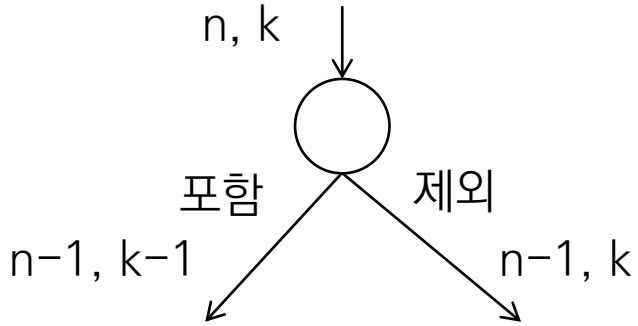
특정원소를 포함하는 경우와 포함하지 않는 경우로 나눠서 생각할 수 있음.

  - 특정 원소를 포함하면 남은 숫자가 하나 줄고(n-1), 골라야 하는 개수도 줄어든다 (k-1).
  - 특정 원소를 제외하면 남은 숫자가 하나 줄고(n-1), 골라야 하는 개수는 그대로 (k).
  - ${}_nC_k = {}_{n-1}C_{k-1} + {}_{n-1}C_k$

- n개에서 k개를 고르는 경우의 수 : nCk
  - A[] = {1, 2, 3}에서 두 개의 숫자를 고르는 경우의 수 : ${}_3C_2$



n : 선택가능한 개수
k : 골라야 하는 개수



n < k은 조건에 맞지 않으므로 리턴

연습

- {1, 2, 3}에서 2개를 고르는 조합 만들기.

```
nck( n, k)
{
        if (k == 0)
        {
            // 조합 출력
        }
        else if (n < k)
            return;
        else
        {
                c[k - 1] = a[n - 1];
                nck(n - 1, k - 1);

                nck(n - 1, k);
        }
}
```

n과 k가 줄어드는 특성을 인덱스에 활용

다음 호출에서 c[k-1]자리에 덮어쓰기 때문에 c[k-1]을 초기화할 필요가 없다.

연습

- {1, 2, 3, 4, 5}에서 3개를 고르는 조합 만들기.

345
245
145
235
135
125
234
134
124
123