

알고리즘 학습

- 1단계 : 배열 연습.
 - 2단계 : 재귀, 백트래킹. 그래프 기본
 - 재귀호출, 조합, 순열, DFS, BFS, 위상정렬.
 - 미로 찾기 등
 - 3단계 : 그래프 최적화, 동적계획법(DP)
 - 프림, 크루스칼, 다익스트라 알고리즘 등.
 - 최단거리, MST, 연속행렬 곱셈 등.
 - 일반적인 알고리즘 교재는 문제풀이 중심.
 - 수식과 관련된 내용은 이산수학에서 다룸.
-

복습

■ 정수형

- 연산결과는 64-bit 정수형이 필요한 경우가 있음.
 - long long (C언어), long (Java)
 - 64-bit를 넘어가는 정수형을 다루는 경우 수식으로 처리해야 함.
 - C의 경우 scanf()의 서식문자로 입력을 한 자리 씩 읽을 수 있음.
 - 예) %1d, %1x
 - for문에는 int형 사용.
 - 대략 10자리 숫자.
-

■ 자료형

- 실수형

- double형 사용 (float은 표현할 수 있는 범위가 작음).
- 고정형 sign bit를 사용하므로 unsigned를 붙여도 자리수가 늘어나지 않음.
- 출력에 서식문자를 사용하면 마지막 자리 뒤에서 반올림 됨.
 - 예) %.3f

■ 배열

- 배열의 크기에 주의해야 함.
 - 인덱스를 0번부터 사용하는 경우와 1번부터 사용하는 경우.
 - 인덱스 연산은 범위를 미리 정의하고 코딩을 시작해야 함.
 - 배열의 끝 부분을 읽지 않는 경우가 많으므로 주의.
 - C언어의 경우 큰 배열은 전역으로 선언한다.
 - 스택 영역의 크기가 제한되어 있기 때문
 - 배열의 최대 크기.
 - 정수형 백만개를 저장하는 1차원 배열(혹은 1000x1000 2차원 배열).
 - 배열에 저장할 수 없는 경우 수식을 알아내야 함.
-

프로그래밍 도구

■ C/C++

- Visual Studio 2013 Express 이상 버전
- Dev-C++ 최신버전
- ✓ Linux 기반의 채점사이트 서버에서는 gcc 사용.
- ✓ C 문법으로 작성했더라도 .cpp로 작성한 경우, 서버 제출시 사용언어를 C++로 선택해야 함.

■ Java

- Eclipse + JDK
-

연습

- 크기가 $N \times N$ 인 2차원 배열에 다음과 같은 모양으로 숫자가 저장된다.
- 크기 N 과 행, 열이 주어지면 해당 위치에 저장된 숫자를 출력하십시오.

	0	1	2	3	4	5	열
0		1	6	10	13	15	
1			2	7	11	14	
2				3	8	12	
3					4	9	
4						5	
5							
행							

입력
1 // 테스트케이스 개수 6 2 4 // N 행 열
출력
#1 8

■ 접근 방법

인덱스로 표시한 접근 순서

1	2	3	4	5	→행과 열의 차이 i
0, 1 (1)	0, 2 (6)	0, 3 (10)	0, 4 (13)	0, 5 (15)	
1, 2 (2)	1, 3 (7)	1, 4 (11)	1, 5 (14)		
2, 3 (3)	2, 4 (8)	2, 5 (12)			
3, 4 (4)	3, 5 (9)				
4, 5 (5)					
4	3	2	1	0	마지막 행 번호 N-i

```
k = 1;
for (i = 1; i <= N; i++) // 행과 열의 차이
{
    for (j = 0; j <= N - i; j++) // 행 번호
    {
        arr[j][j + i] = k++; // 열 번호 j+i
    }
}
```

재귀호출

- 여러 형태의 재귀호출 연습 -

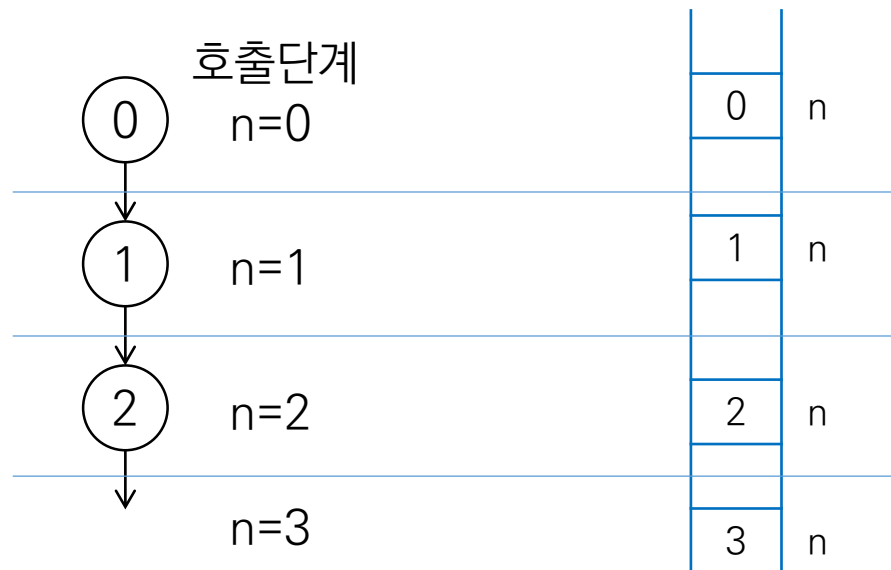
재귀호출의 기본

■ 특징

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

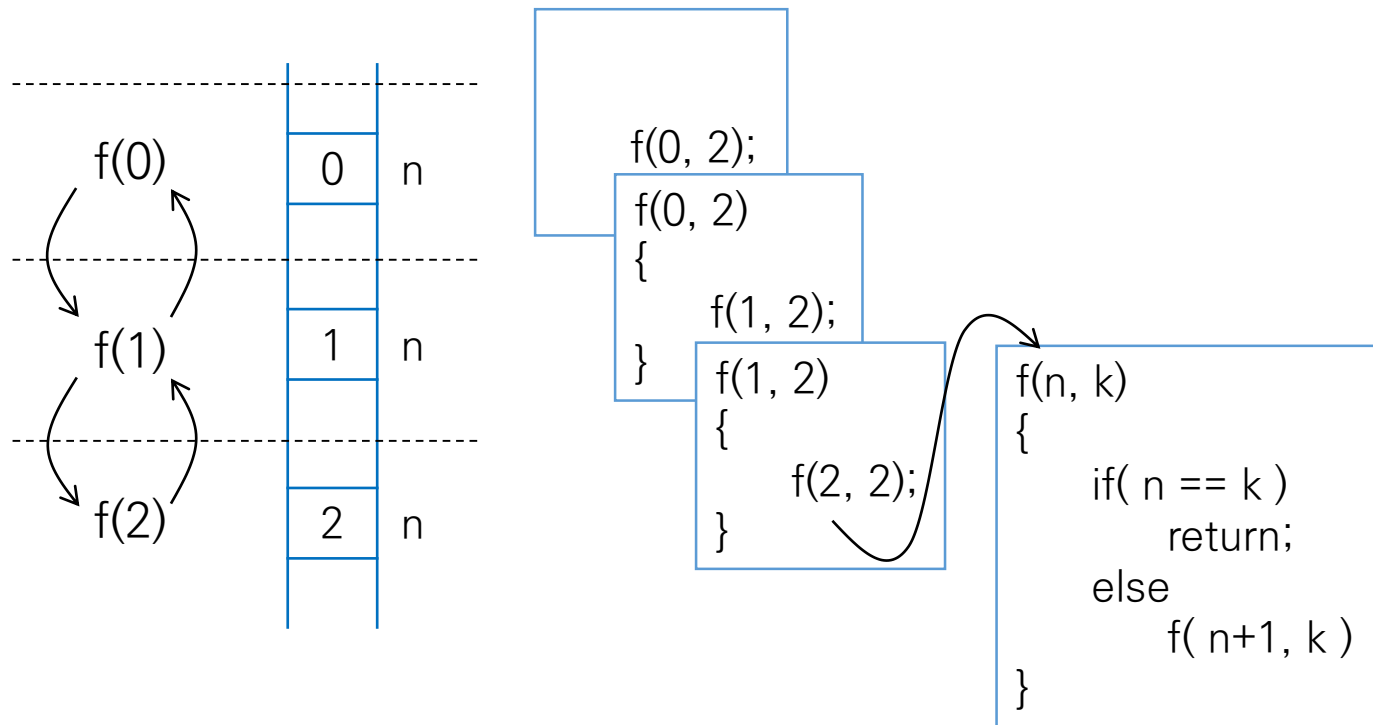
기본적인 재귀 호출

```
f(int n)
{
    ...
    f(n+1);
}
```



■ 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.

```
f(n, k)
{
    if( n == k )
    {
        
        return;
    }
    else
    {
        
        f( n+1, k );
    }
}
```

조건을 만족할 때의 작업 위치.

호출 때마다 해야하는 작업의 위치.
여기서 다른 함수를 호출하면
처리 속도가 느려지므로 주의.

■ 재귀호출을 이용해 배열 복사하기

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

i	0	1	2	3
A	1	2	3	



i	0	1	2	3
B	1	2	3	

$f(0, 3);$

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

원하는 조건을 찾으면 중단하는 경우

- 주어진 집합에 v 가 들어있으면 1, 없으면 0을 리턴하는 재귀 호출을 만드시오.

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

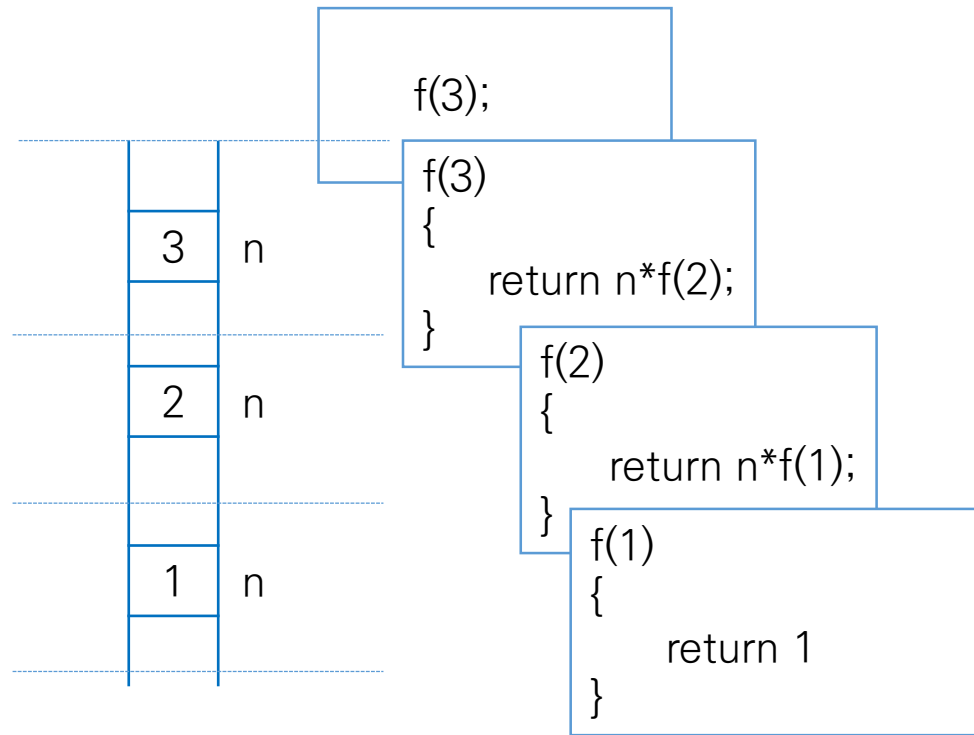
```
find ( n, k, v )  
{  
    if( n == k) // 배열을 벗어남  
        return 0;  
    else if ( a[n] == v)  
        return 1;  
    else  
        r = f( n+1, k, v);  
}
```

■ 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

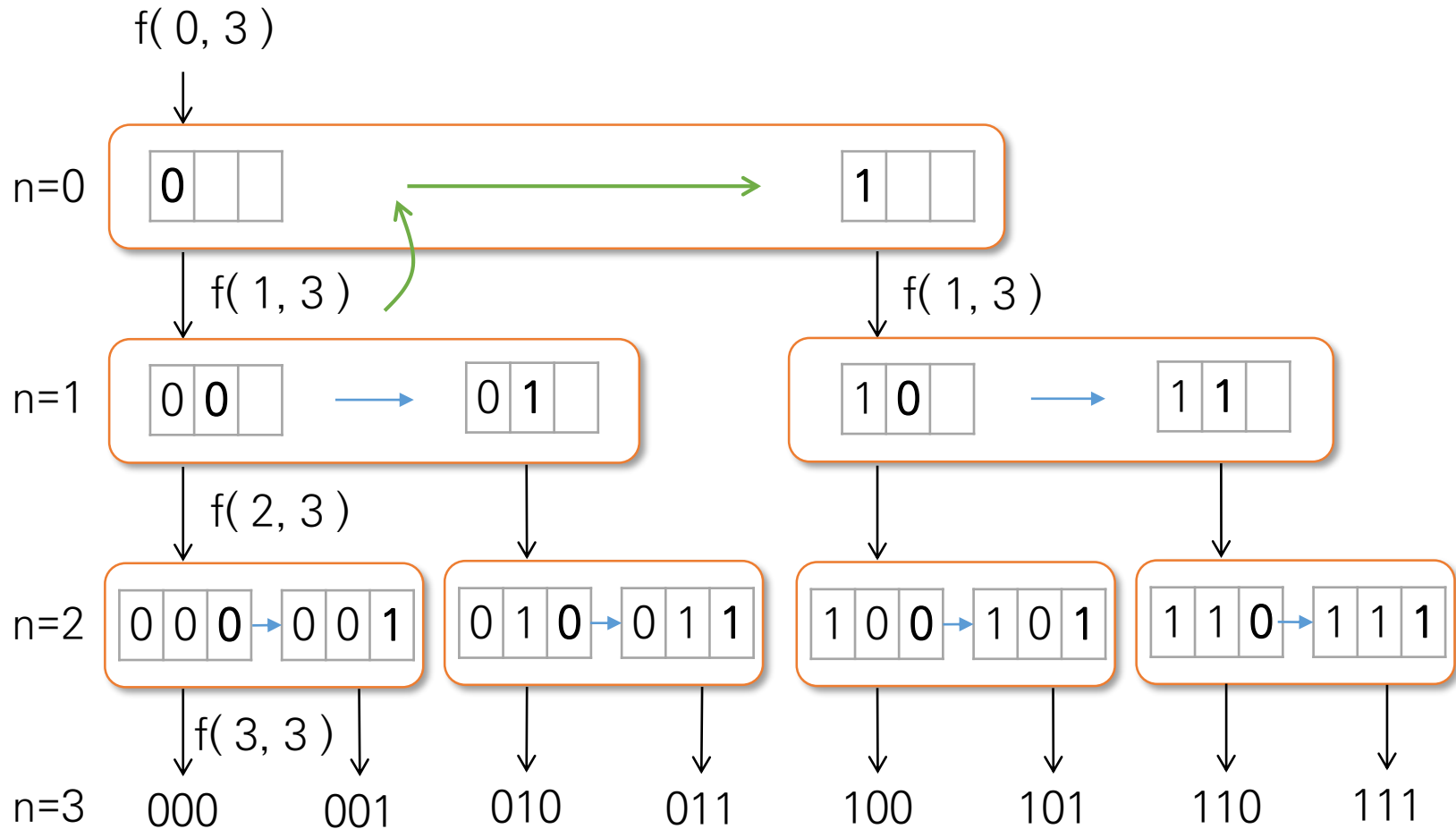
- 배열 L의 각 자리에 0/1이 오는 모든 경우 만들기.

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 n 과 배열크기 k 의 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];

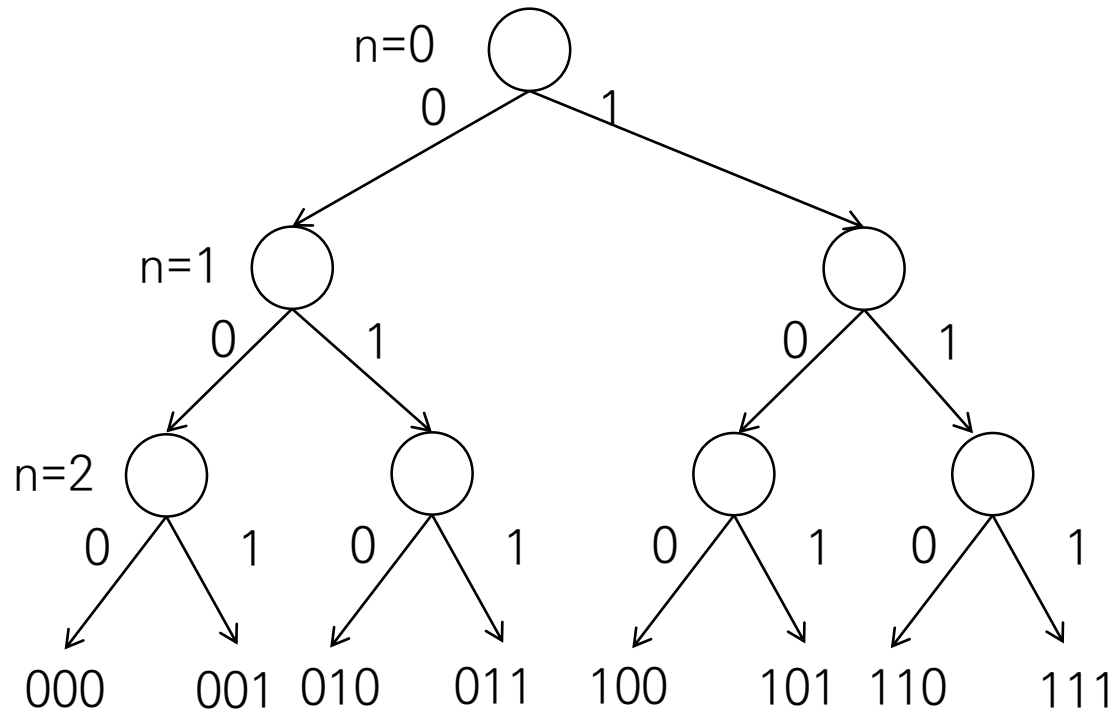
f( n, k)
{
    if( n==k )
        //배열 L 출력
    else
        L[n] = 0;
        f( n+1, k );
        L[n] = 1;
        f( n+1, k );
}
```

```
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.

	1	2	3	포함 여부를 1과 0으로 표시		
{1, 2, 3}	포함	포함	포함	1	1	1
{1, 2}			미포함			0
{1, 3}		미포함	포함		0	1
{1}			미포함			0
{2, 3}	미포함	포함	포함	0	1	1
{2}			미포함			0
{3}		미포함	포함		0	1
{}			미포함			0

- 두 재귀호출의 리턴 값을 사용하는 경우.

- 피보나치 수열

- $f(n) = f(n-1) + f(n-2);$

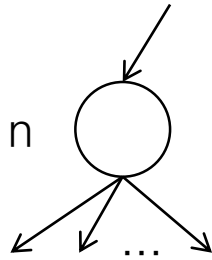
- $f(0) = 0, f(1) = 1;$

```
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.

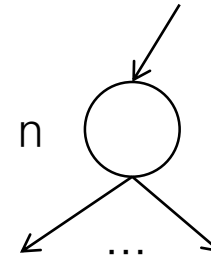
재귀 호출이 여러 번인 경우



항상 j번인 경우

```
...  
else  
  for i : 1 -> j  
    L[n] = i;  
    f( n+1, k);  
}
```

예) 1, 2, 3을 중복 사용해 3자리 숫자 만들기



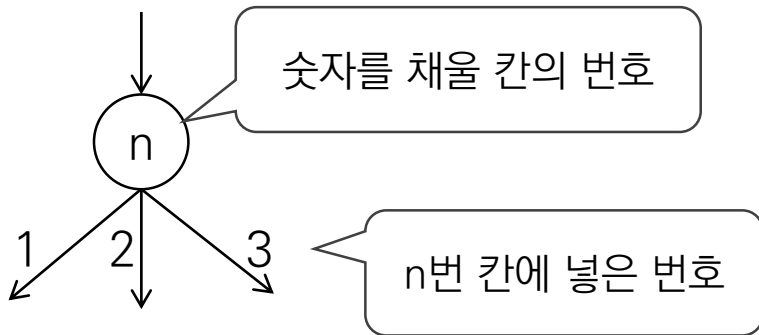
조건에 따라 달라지는 경우

```
...  
else  
  for i : 1 -> j  
    if ( i가 유효하면 )  
      L[n] = i;  
      f( n+1, k )  
}
```

예) 그래프에서 인접 노드에 방문하기

연습

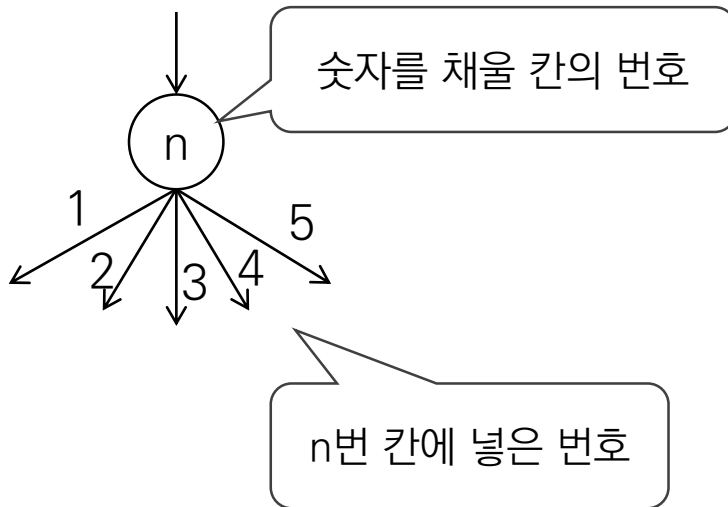
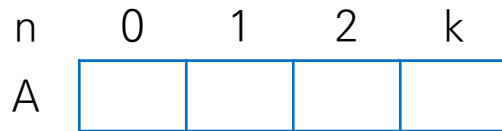
- 1, 2, 3을 중복 사용해 3자리 숫자 만들기.



```
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.

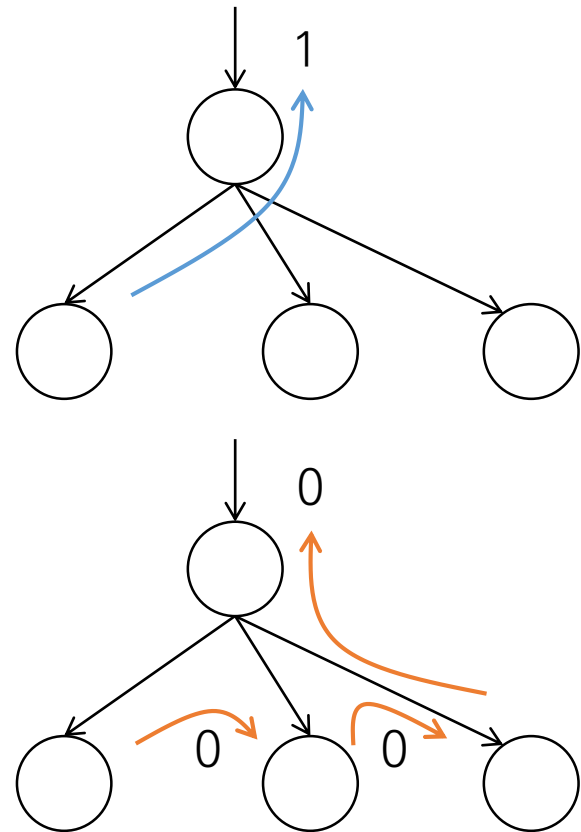


```
f(n, k)
{
    if( n == k)
        printA( );
        return;
    else
        for i : 1 -> 5
            A[n] = i;
            f( n+1 );
}
```


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;  
}
```

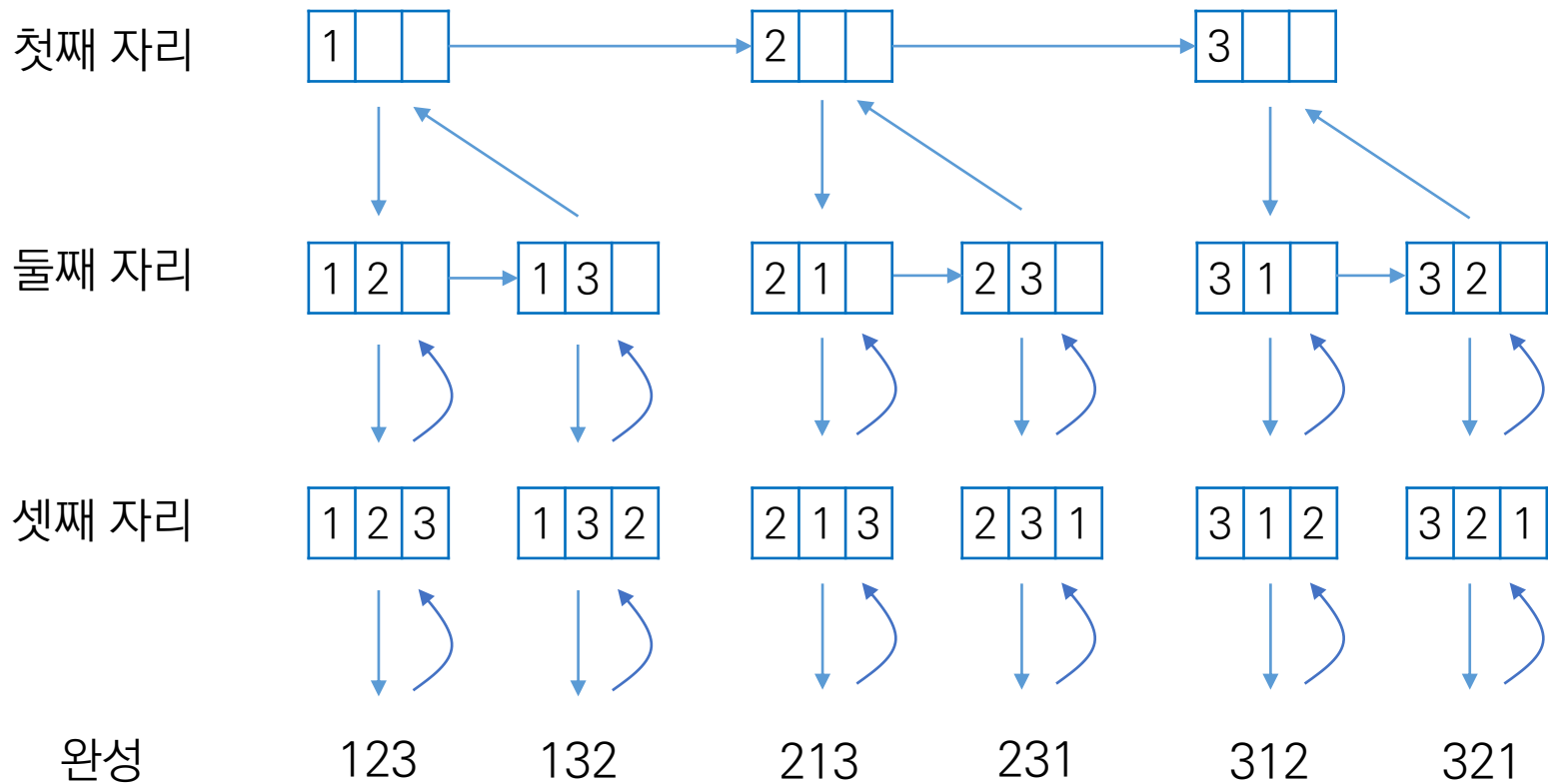


연습

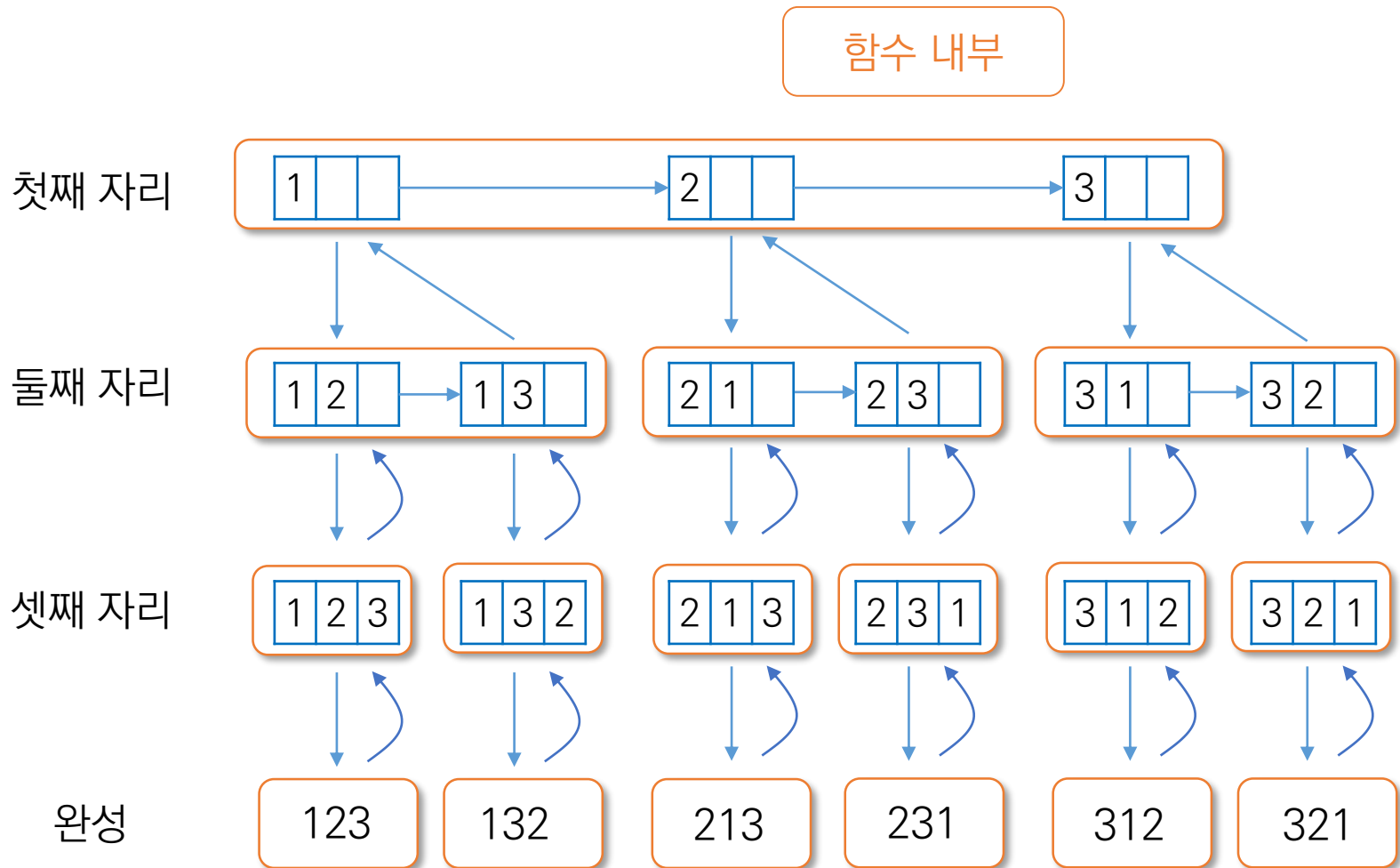
- 배열안에 찾는 숫자가 있으면 1, 없으면 0을 리턴하는 재귀함수 만들기.
 - `int arr[] = { 3, 7, 6, 2, 5, 4, 8, 9};`
 - 찾고자 하는 숫자 5.

호출 횟수가 변하는 재귀 호출

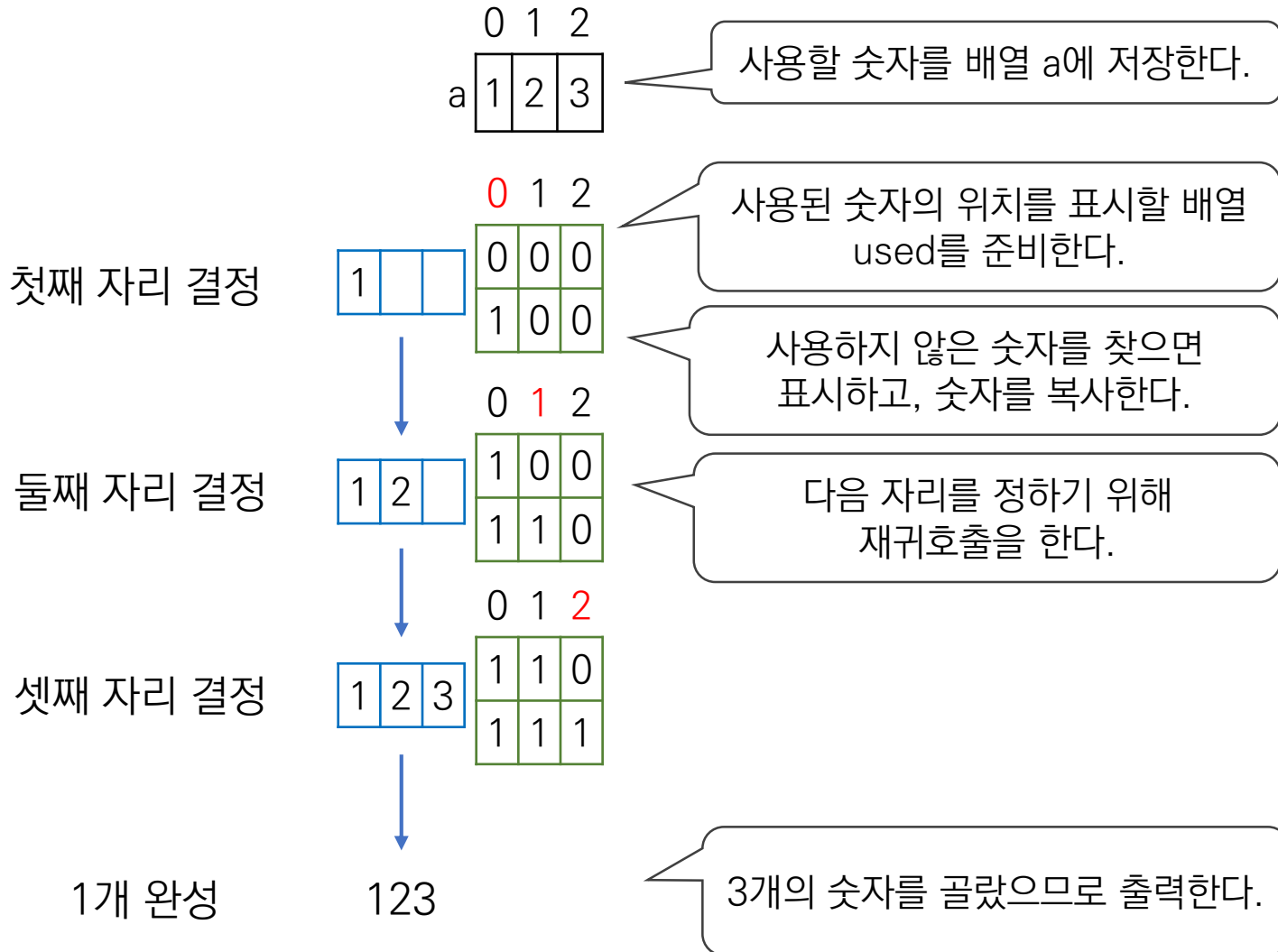
- 1, 2, 3으로 3자리 수 만들기



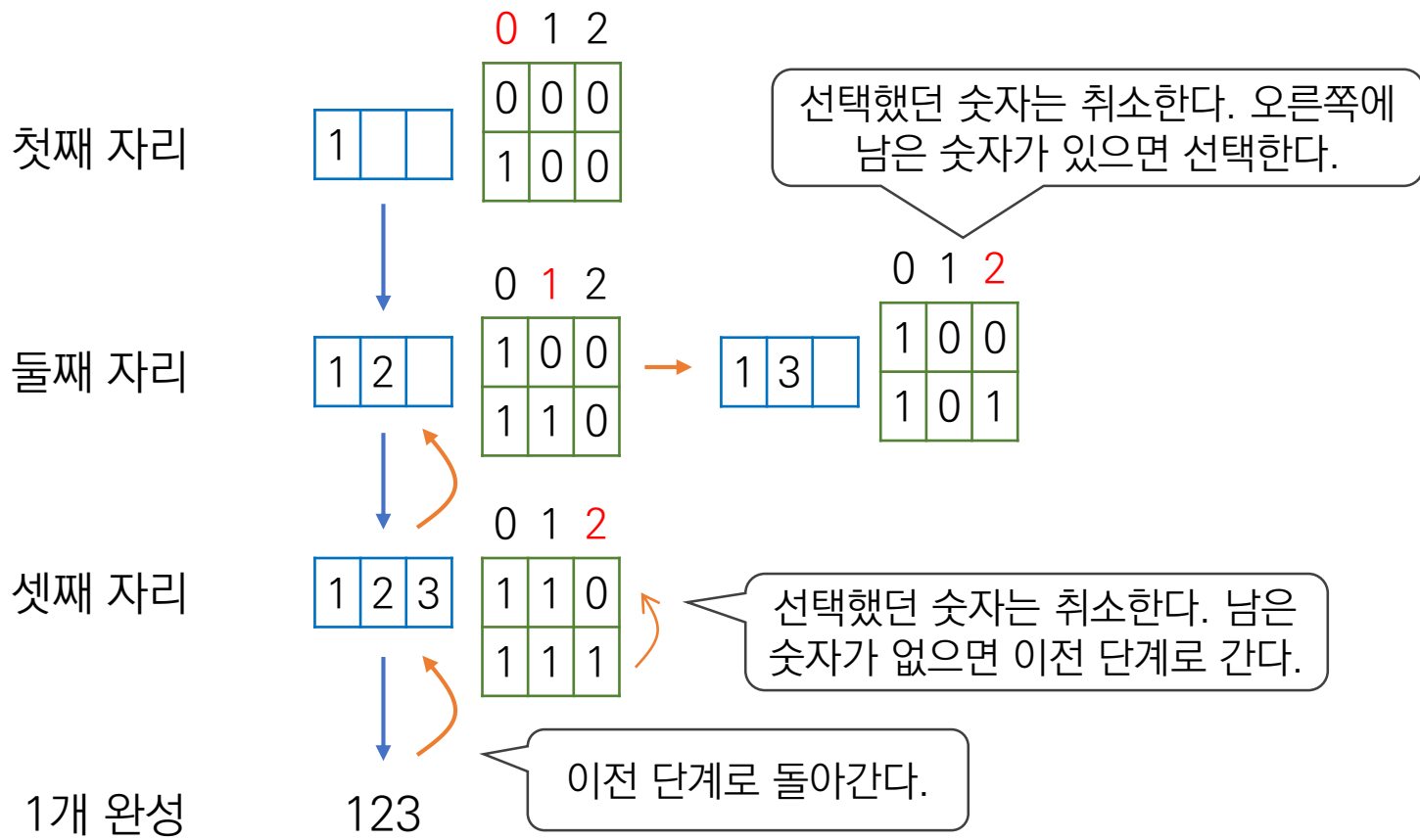
■ 1, 2, 3으로 3자리 수 만들기



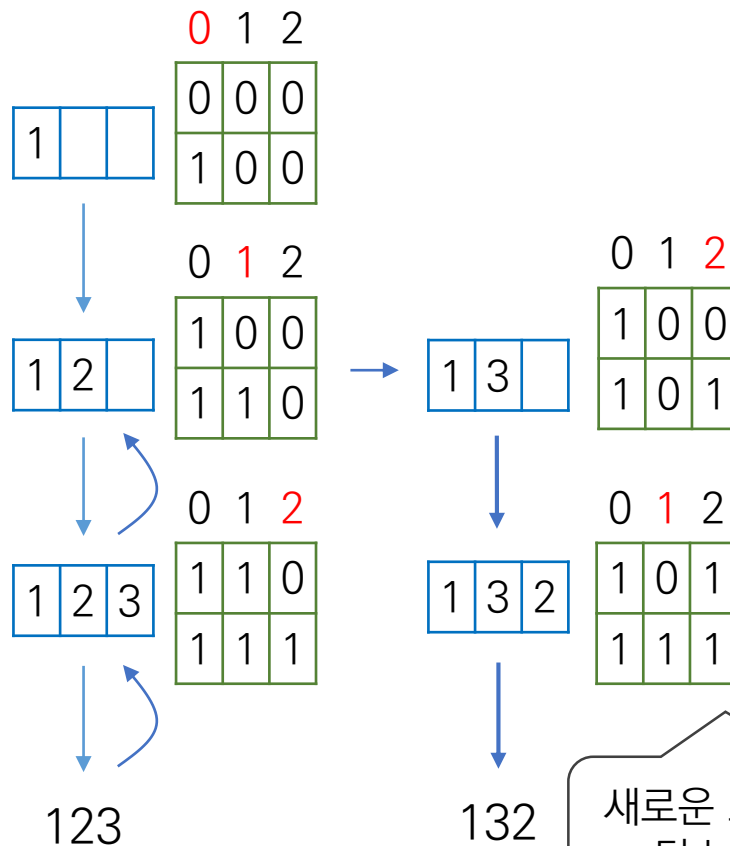
■ 1, 2, 3으로 3자리 수 만들기



■ 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];
        perm(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.

	0	1	2	3	4
a	1	2	3	4	5

주어진 숫자의 개수가 함수내 호출 횟수를 결정한다.

1		
---	--	--

0	1	2	3	4
0	0	0	0	0
1	0	0	0	0

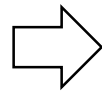
채울 칸 수가 호출 깊이를 결정한다.

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



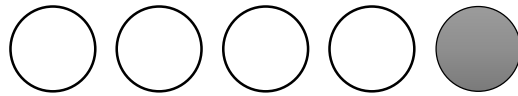
1	2	3	합계
A	B	C	33
A	C	B	33
B	A	C	25
B	C	A	25
C	B	A	32
C	A	B	32

배정에 순열을 활용

호출의 깊이가 변하는 재귀 호출

■ 조합만들기

- n 개에서 k 개를 고르는 경우의 수 : nC_k
 - $\{1, 2, 3\}$ 에서 두 개의 숫자를 고르는 경우의 수 : ${}_3C_2$
 - $\{1, 2\}, \{1, 3\}, \{2, 3\}$
- 두 경우로 나눠 생각할 수 있다.

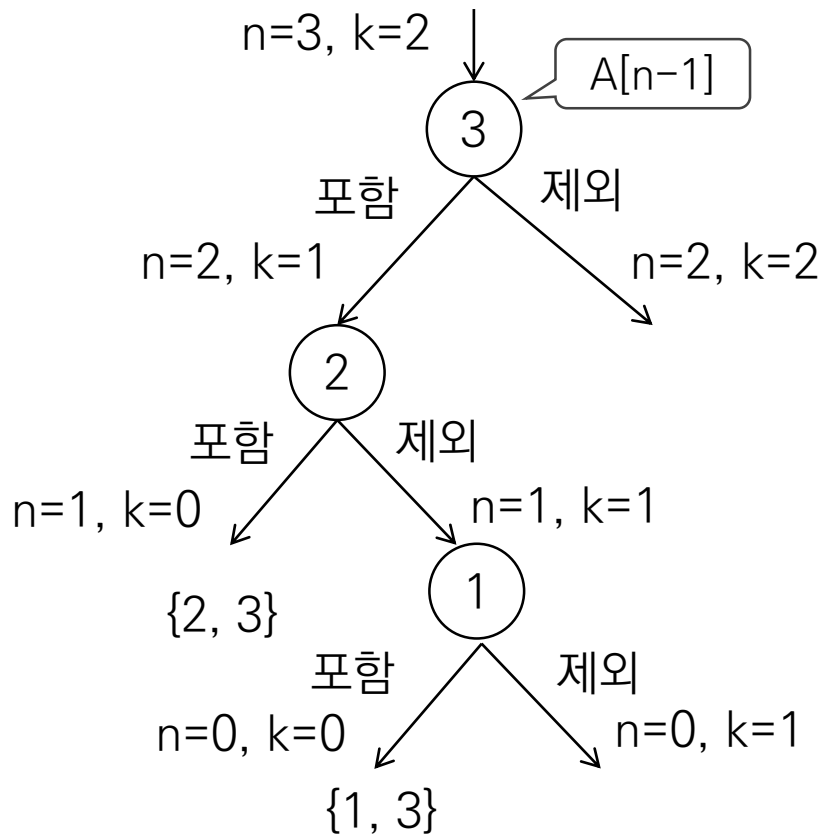


특정원소를 포함하는 경우와
포함하지 않는 경우로 나눠서
생각할 수 있음.

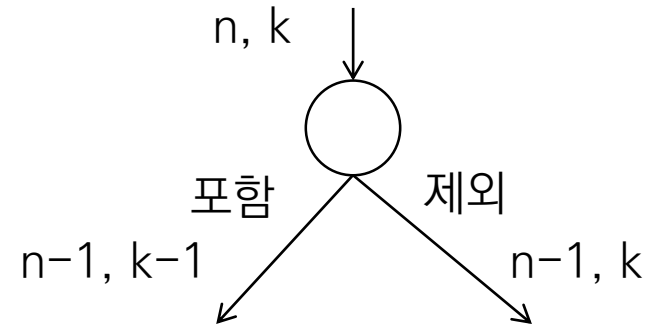
- 특정 원소를 포함하면 남은 숫자가 하나 줄고($n-1$), 골라야 하는 개수도 줄어든다 ($k-1$).
- 특정 원소를 제외하면 남은 숫자가 하나 줄고($n-1$), 골라야 하는 개수는 그대로 (k).
- ${}_nC_k = {}_{n-1}C_{k-1} + {}_{n-1}C_k$

■ n 개에서 k 개를 고르는 경우의 수 : nCk

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



n : 선택가능한 개수
 k : 골라야 하는 개수



$n < k$ 은 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

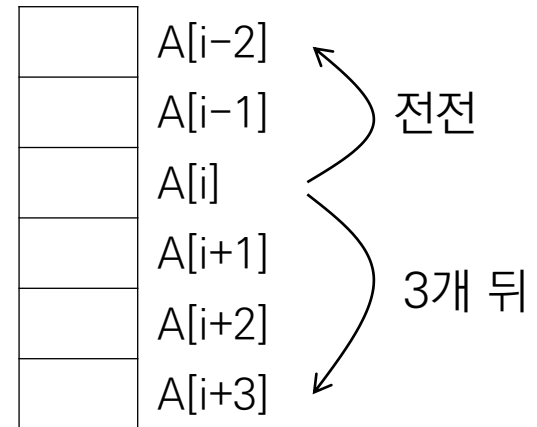
이진트리

- 이진트리의 저장 방법과 순회-

자료구조

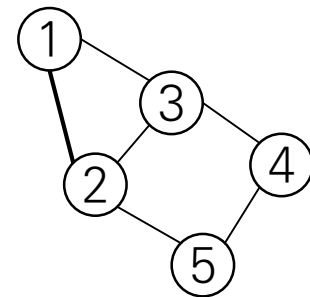
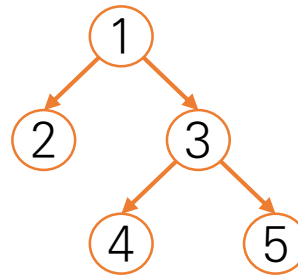
■ 선형 자료구조

- 저장된 자료의 순서만 구분할 수 있음.
- 이전, 이후 자료는 하나씩만 존재.



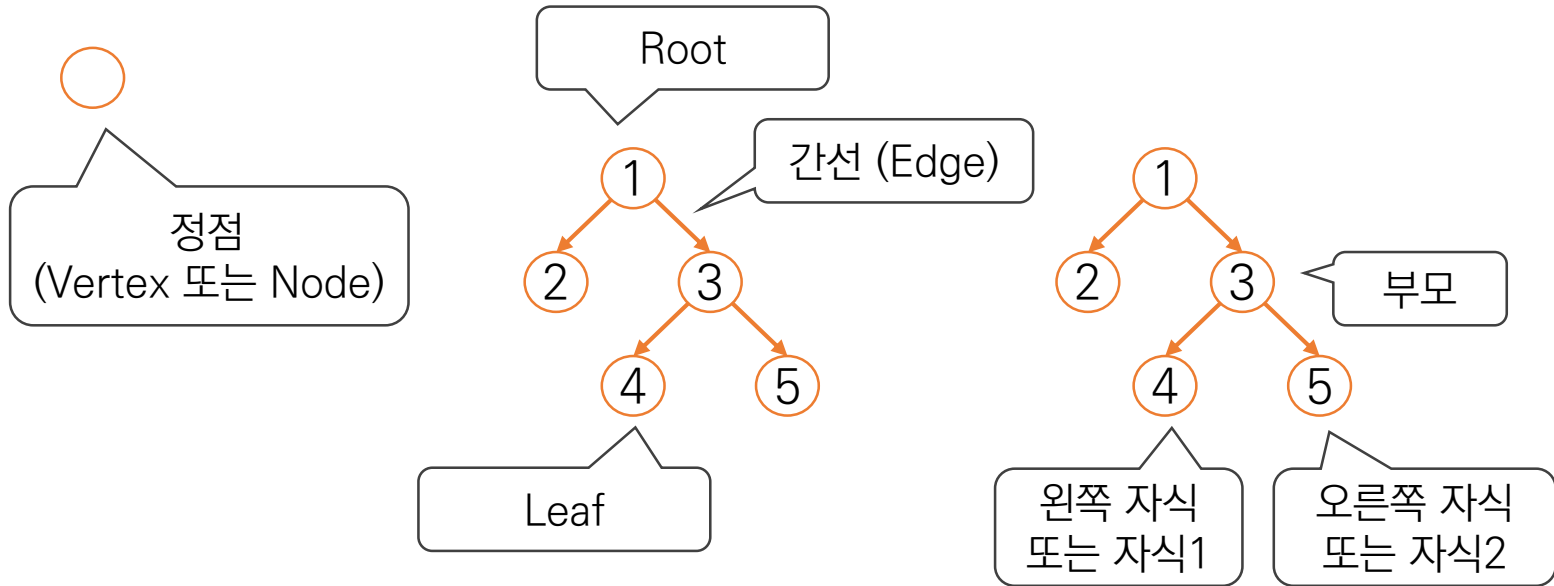
■ 비선형 자료구조

- 1:N 관계를 표시하는 경우.
 - 이진트리, 힙.
- N:N 관계를 표시하는 경우.
 - 그래프.



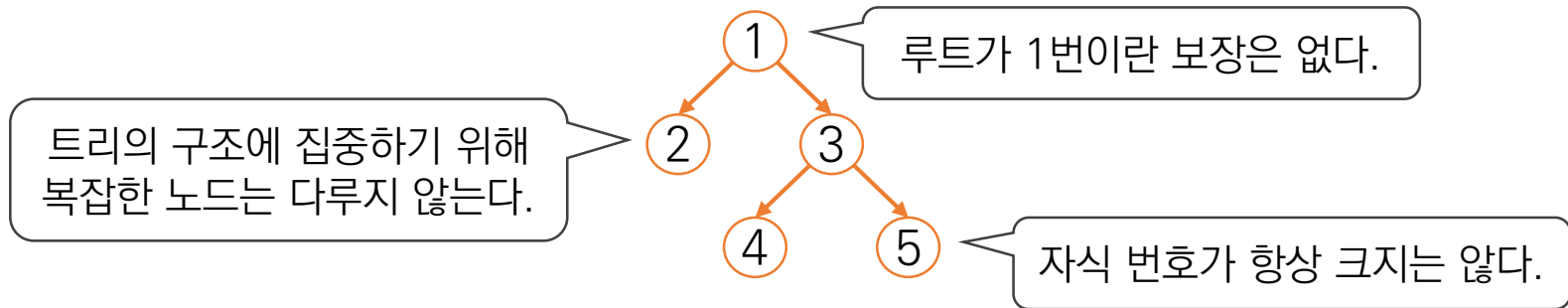
이진트리

- 1:2 관계로 나타낸 트리 구조.



■ 이진트리 (계속)

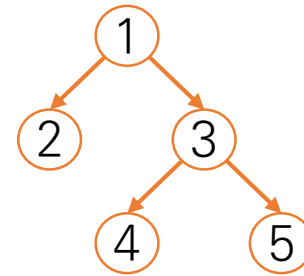
- 조상
 - 어떤 정점에서 루트까지의 경로에 있는 정점들.
 - 4의 조상 : 3, 1
- 자손
 - 어떤 정점 아래에 있는 정점들.
 - 3의 자손 : 4, 5



■ 트리 정보

- 입력

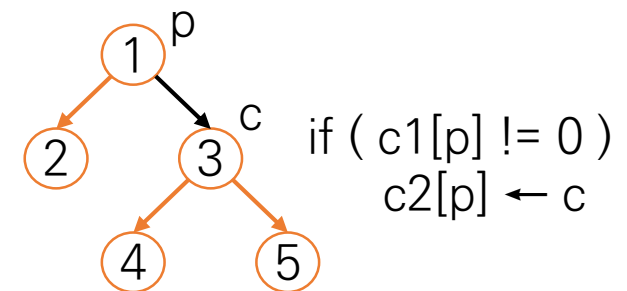
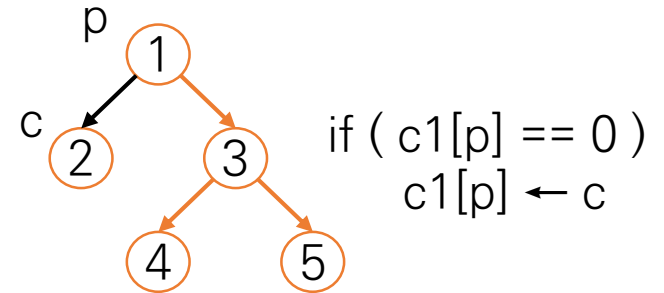
4 \leftarrow 간선의 개수 N
1 2 1 3 3 4 3 5 \leftarrow 부모 자식 순



■ 부모 번호를 인덱스로 자식 번호를 저장.

부모	p	0	1	2	3	4	5
자식1	c1	0	2	0	0	0	0
자식2	c2	0	0	0	0	0	0

부모	p	0	1	2	3	4	5
자식1	c1	0	2	0	0	0	0
자식2	c2	0	3	0	0	0	0



```

for i : 1 → N
  read p, c;
  if( ch1[p] == 0)
    ch1[p] = c;
  else
    ch2[p] = c;
  
```

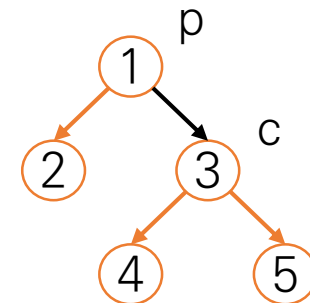
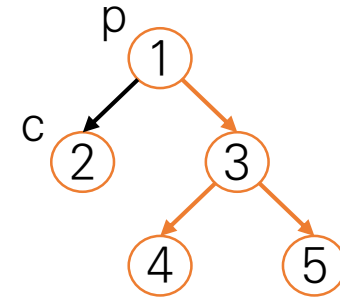
- 자식 번호를 인덱스로 부모 번호를 저장.

자식	c	0	1	2	3	4	5
부모	a	0	0	1	0	0	0

$a[c] \leftarrow p$

자식	c	0	1	2	3	4	5
부모	a	0	0	1	1	0	0

$a[c] \leftarrow p$



```
for i : 1 -> N
  read p, c;
  pa[c] = p;
```

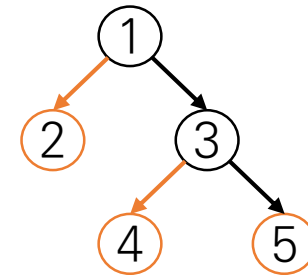
■ 루트 찾기

- 부모가 없는 노드를 찾으면 됨.

■ 조상 노드 찾기

자식	c	0	1	2	3	4	5
부모	a	0	0	1	1	3	3

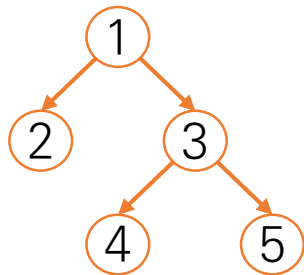
5번 노드의 조상 찾기



```
c = 5;
while( a[c] != 0 ) // 루트인지 확인
{
    c = a[c];
    print(c);
}
```

■ 이진 트리 순회 1

- 모든 노드를 빠짐없이, 중복도 없이 방문하는 방법.



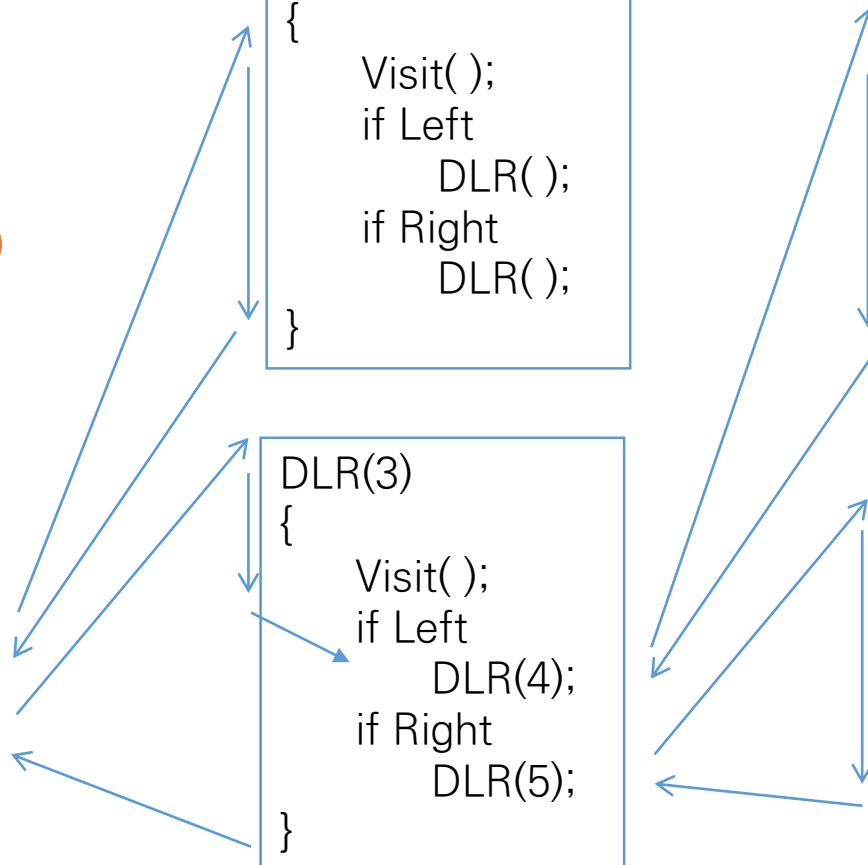
```
DLR(1)
{
  Visit( );
  if Left
    DLR(2);
  if Right
    DLR(3);
}
```

```
DLR(2)
{
  Visit( );
  if Left
    DLR( );
  if Right
    DLR( );
}
```

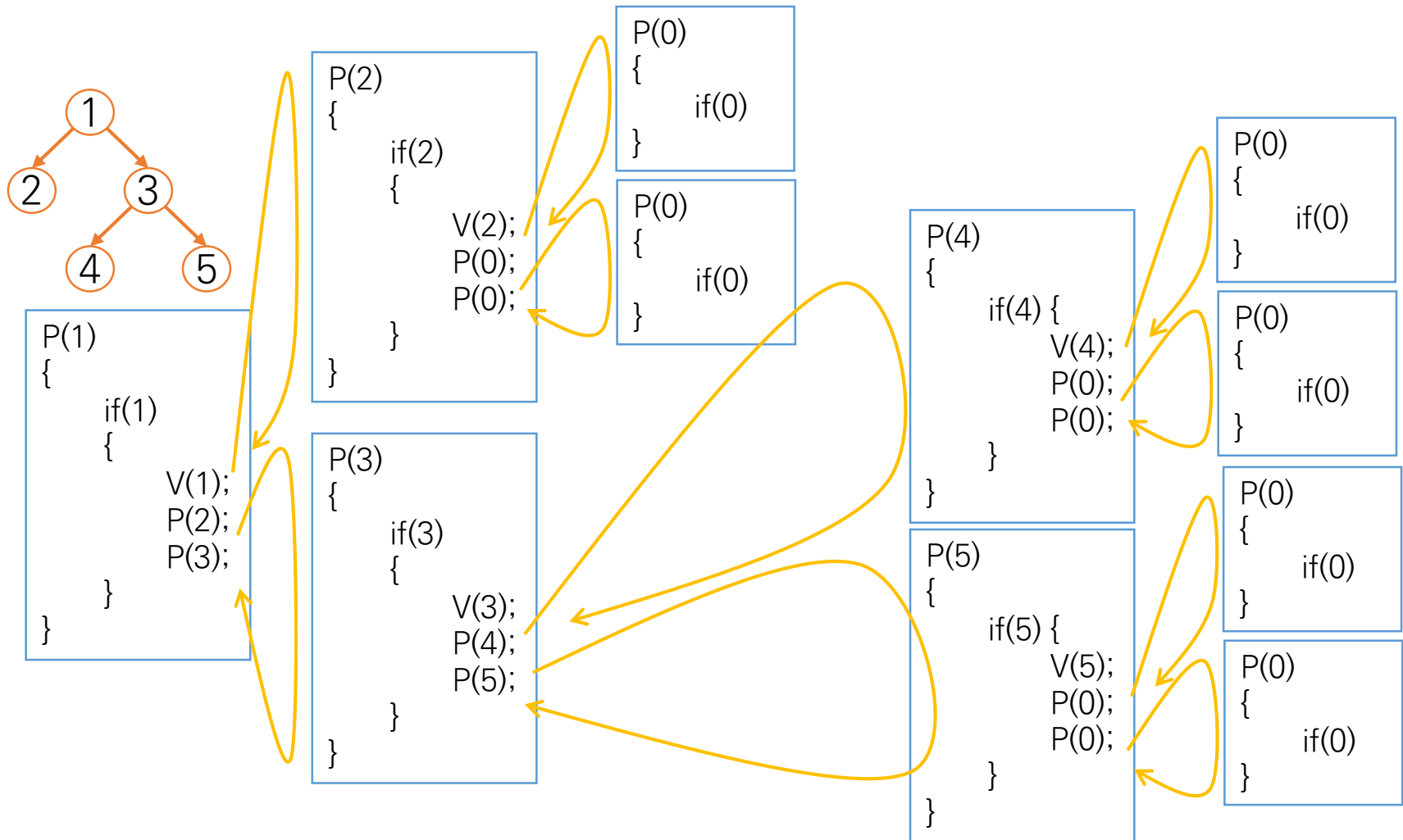
```
DLR(3)
{
  Visit( );
  if Left
    DLR(4);
  if Right
    DLR(5);
}
```

```
DLR(4)
{
  Visit( );
  if Left
    DLR( );
  if Right
    DLR( );
}
```

```
DLR(5)
{
  Visit( );
  if Left
    DLR( );
  if Right
    DLR( );
}
```

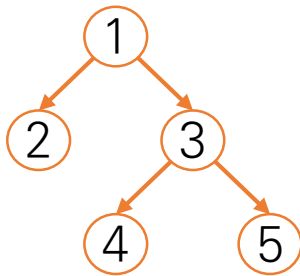


이진 트리 순회 2



연습

- 1번 노드부터 이진트리를 순회하고 방문한 노드의 개수를 출력하시오.



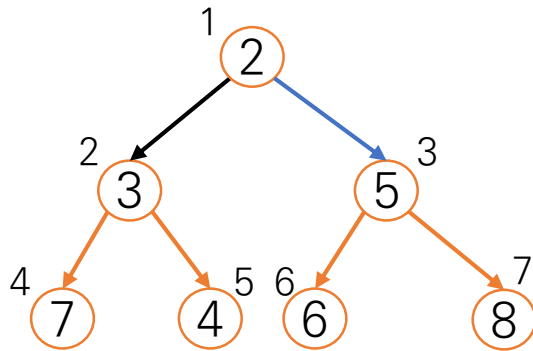
〈참고〉

```
P(1)
{
    if(1)
    {
        cnt++;
        P(2);
        P(3);
    }
}
```

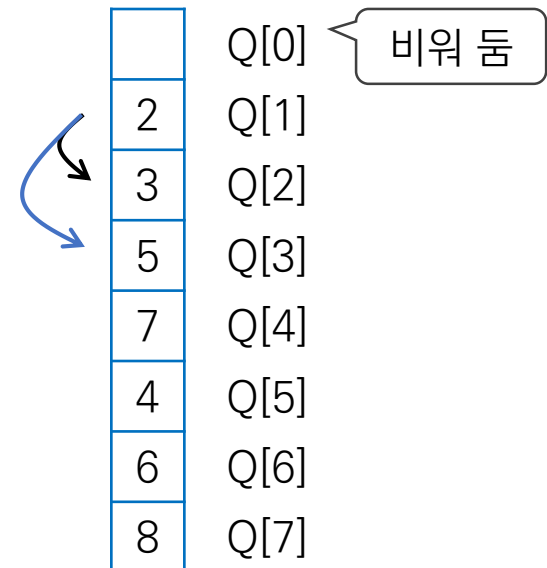

포화/완전 이진트리

■ 포화 이진트리

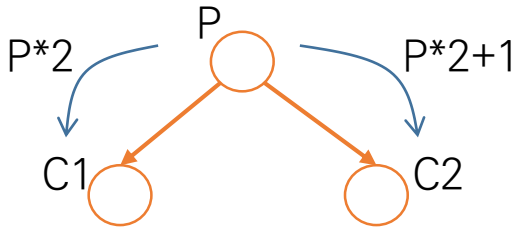
- 특정 높이까지 꽉 차 있는 이진 트리.
- 노드 번호는 위->아래, 왼쪽->오른쪽 순서.
- 노드 번호를 배열의 인덱스로 사용해 저장.



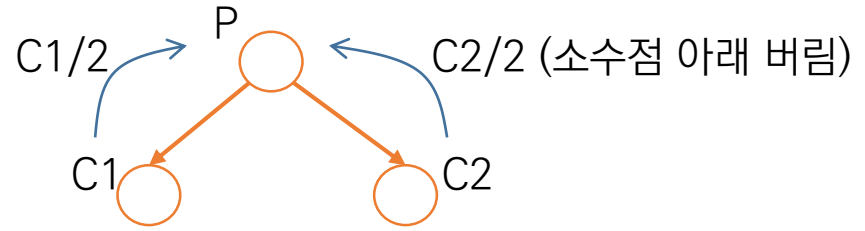
포화 이진트리와 저장 방법



■ 부모-자식 노드 번호 계산



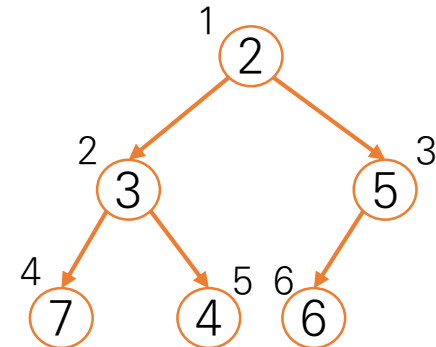
부모->자식



자식->부모

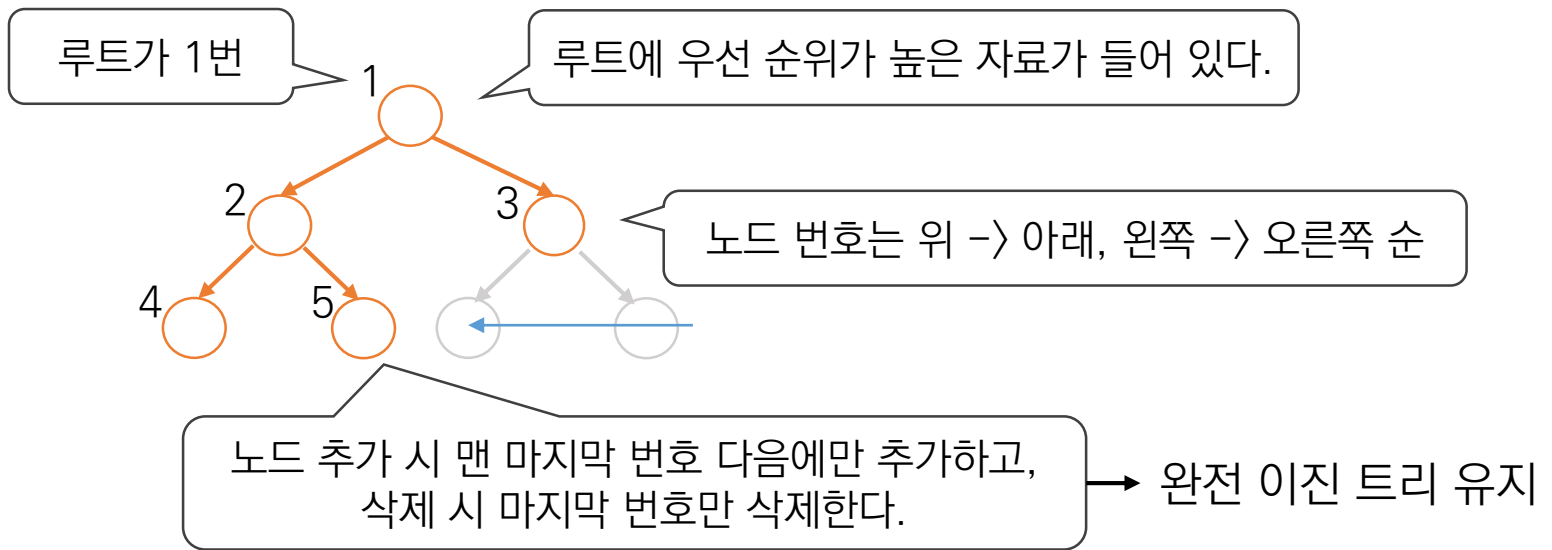
■ 완전 이진트리

- 오른쪽 끝부터 노드가 비어있는 이진 트리.
- 포화 이진트리와 같은 방법으로 저장.



■ 이진 힙 (binary heap)

- 완전 이진트리 유지.
- 루트에 최소값이 저장되는 최소 힙.
- 루트에 최대값이 저장되는 최대 힙.

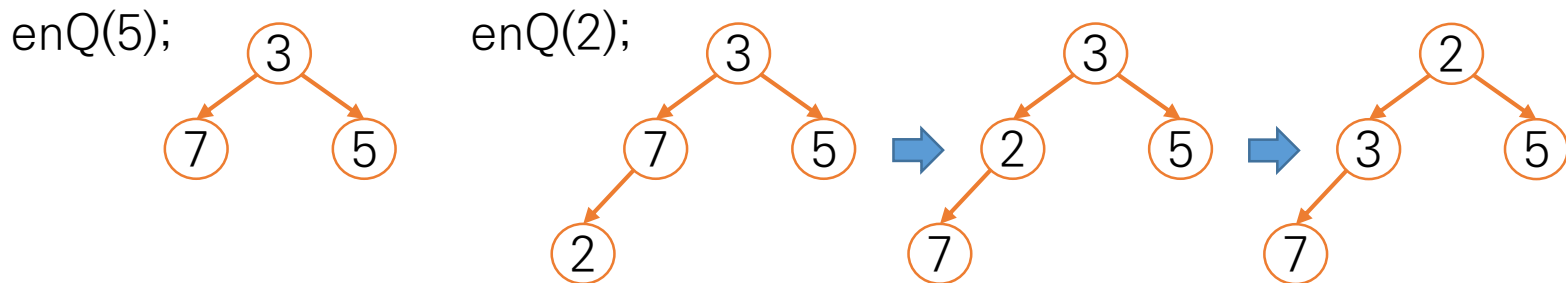
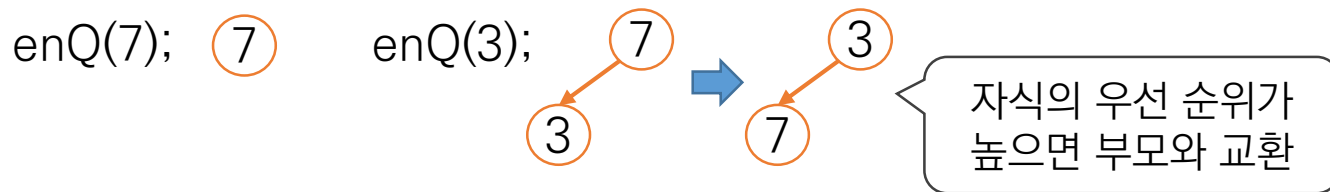


■ 최소 힙 생성

- 작은 값이 우선 순위가 높음. 루트에 가장 작은 값 저장.
- 이진 큐는 우선 순위 큐의 구현에 사용.

조건 : 완전 이진 트리 유지. 부모 < 자식

{7, 3, 5, 2, 4, 6}

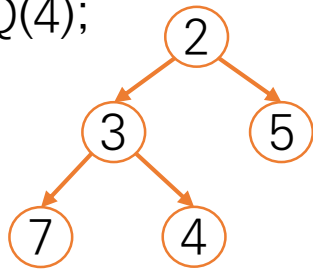


■ 최소 힙 생성(계속)

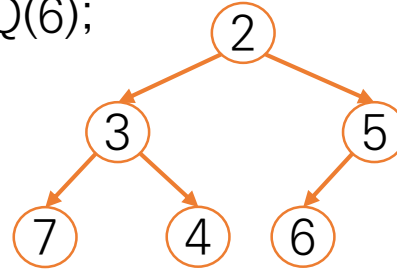
조건 : 완전 이진 트리 유지. 부모 < 자식

{7, 2, 5, 3, 4, 6}

enQ(4);



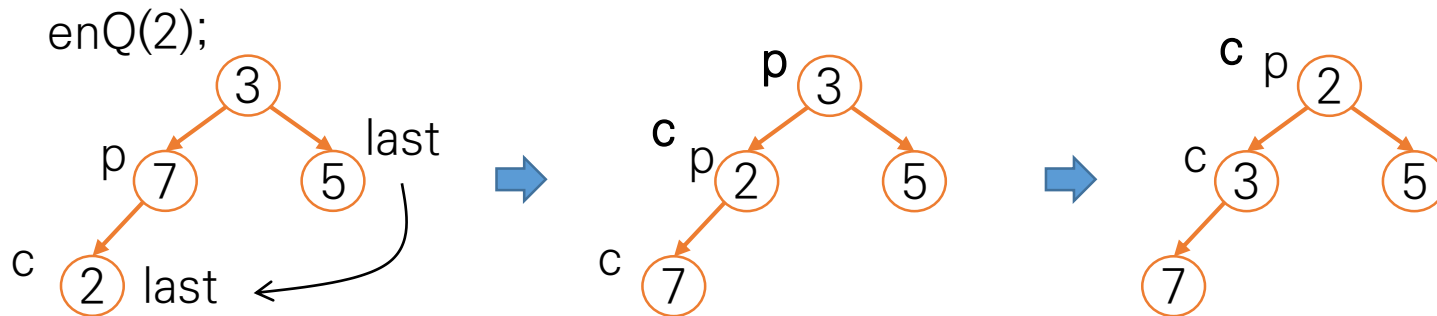
enQ(6);



■ 최소 힙 우선순위 큐의 연산

- enQ()

조건 : 완전 이진 트리 유지. 부모 < 자식



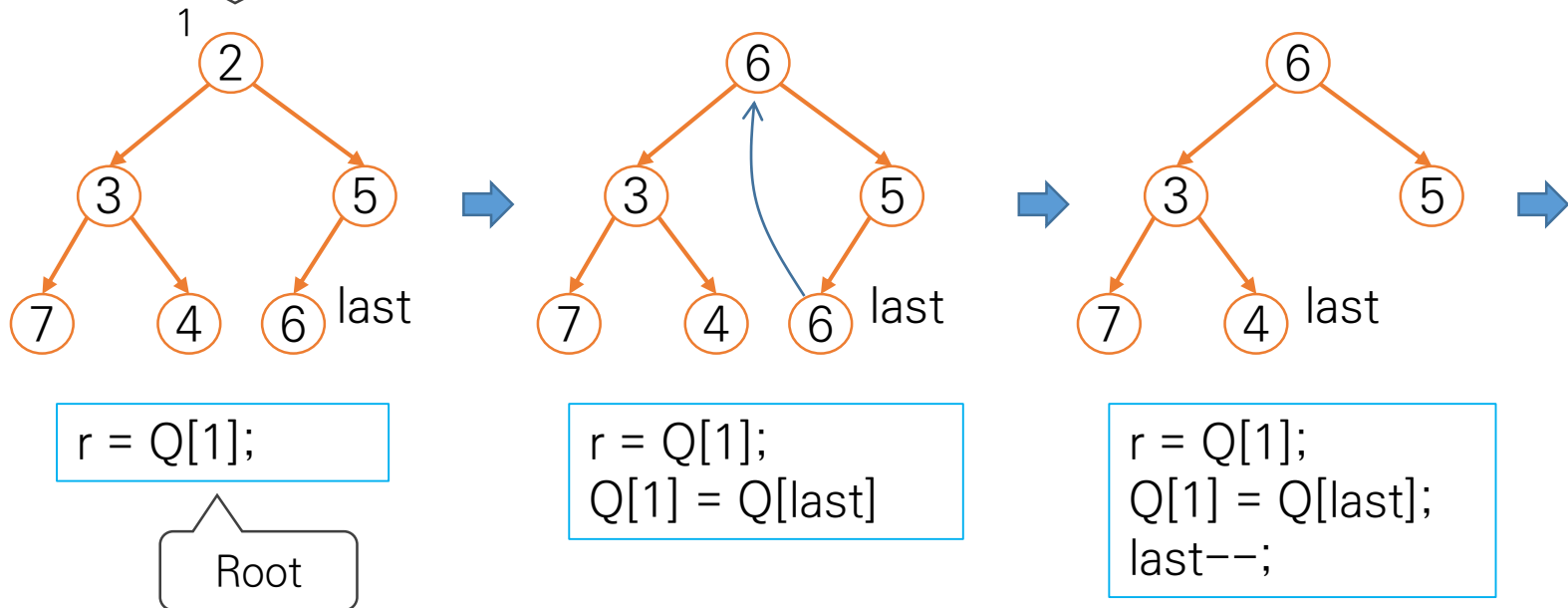
```
c = ++last;  
p = c / 2;  
Q[c] = data;
```

```
while( Q[p] > Q[c] && c > 1)  
{  
    swap(Q[p], Q[c]);  
    c = p;  
    p = p/2;  
}
```

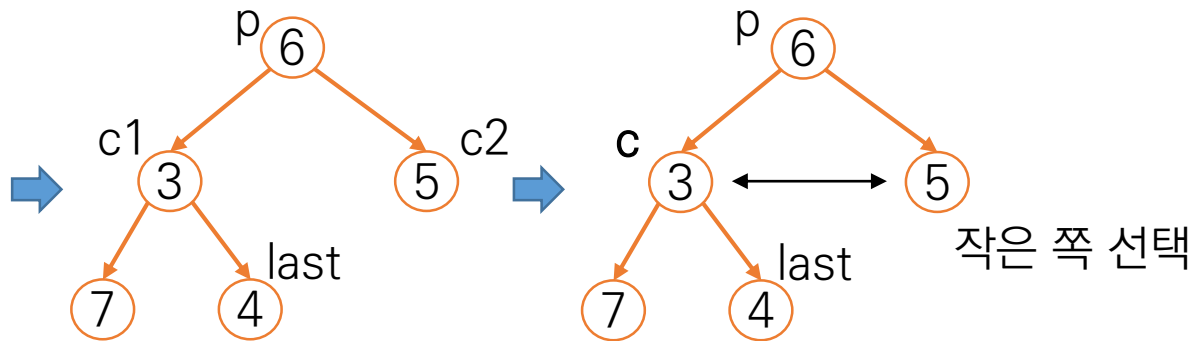
- deQ()

조건 : 완전 이진 트리 유지. 부모 < 자식

디큐는 우선 순위가 높은 것부터!



- deQ()계속



```
p = 1;
while(p < last)
{
    c1 = p * 2;
    c2 = p * 2 + 1;
```

자식 노드 번호 계산

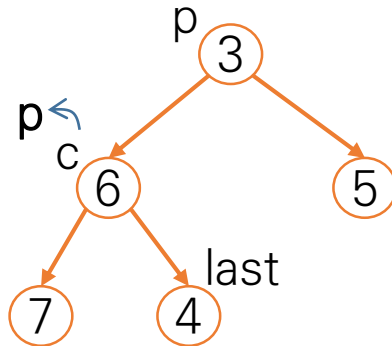
```
if(c2 <= last)
{
    c = Q[c1] < Q[c2] ? c1 : c2;
    if( Q[c] < Q[p] )
        swap(Q[p], Q[c]);
```

작은 쪽과 자리바꿈

양쪽 자식이
있는 경우

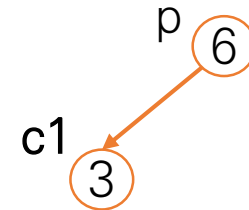
우선순위 큐

- deQ()계속



```
if(c2<=Last)
    c = Q[c1]<Q[c2]?c1:c2;
    if( Q[c] < Q[p] )
        swap(Q[p], Q[c]);
        p = c;
    else
        break;
```

바꾼 쪽을 새로운
부모로



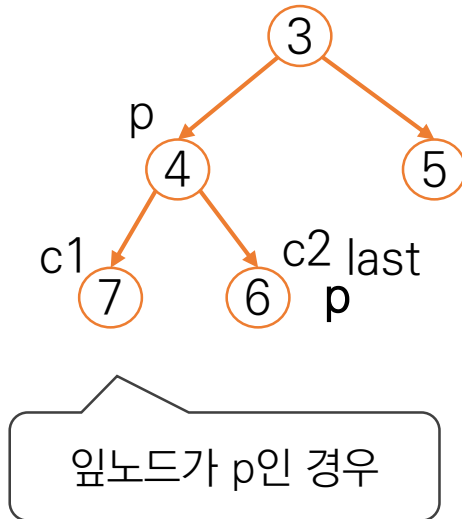
```
while(p<last)
    c1 = p * 2;
    c2 = p * 2 + 1;
    if(c2<=last)
        { ... }
    else if(c1<=last)
        if(Q[c1]<Q[p])
            swap(Q[p], Q[c1]);
            p = c1;
        else
            break;
```

왼쪽 자식만
있는 경우



우선순위 큐

- deQ()계속



```
while(p < last)
{
    c1 = p * 2;
    c2 = p * 2 + 1;
    if(c2 <= last)
    { ... }
    else if(c1 <= last)
    { ... }
    else
        break;
}
```



연습

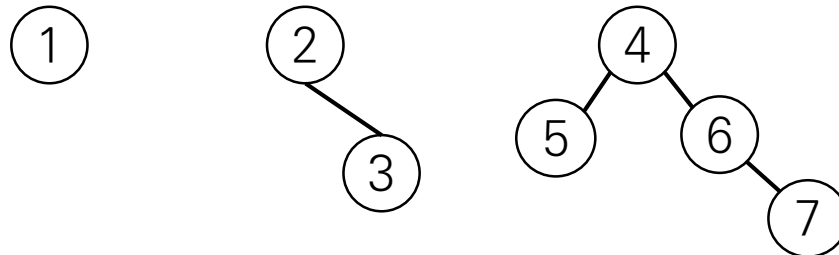
- N개의 정수가 입력된다. 앞에서 설명한 방식대로 최소힙을 구현해 저장하고, 마지막 노드의 조상 노드가 갖고 있는 숫자들의 합을 출력하라. N과 N개의 정수가 차례대로 주어진다.

5 5 4 3 2 1

7 2 6 10 8 5 11 7

같은 트리에 속한 노드인지 확인하는 방법

- 트리의 대표 원소를 이용.
 - 간선으로 연결된 노드 중에서 대표 노드를 지정.
- 크루스칼(Kruskal) 알고리즘으로 최소비용신장트리(MST)를 찾을 때 필요한 기술.
- 서로소 집합과 관련.
- 1, 6번 노드는 같은 트리에 속해있는가?



■ 초기 조건 : 7개의 노드

- 처음에는 자기 자신이 트리의 대표 노드.



노드	1	2	3	4	5	6	7
대표	1	2	3	4	5	6	7

- 간선 정보가 주어지면 대표 원소를 갱신.
- 2 3

노드	1	2	3	4	5	6	7
대표	1	2	2	4	5	6	7



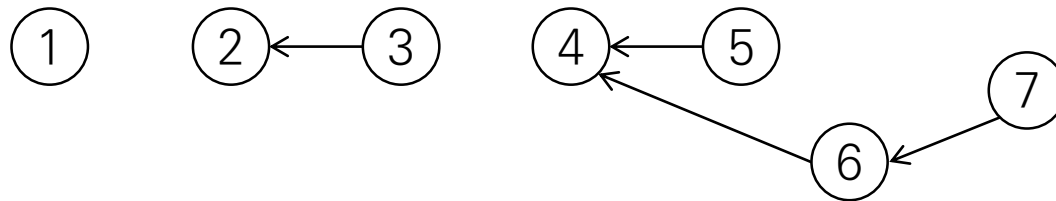
- 4 5

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	6	7



- 4 6, 6 7

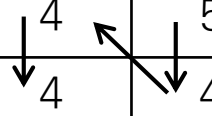
노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	6



■ 5와 7이 같은 트리에 속해 있는지 확인 하는 방법.

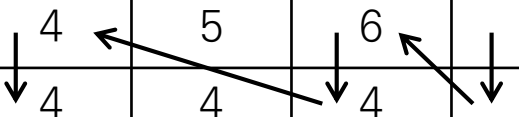
- 5의 대표값과 7의 대표값을 비교.
- 5의 대표값.
 - 노드번호와 대표값이 같은 4.

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	6



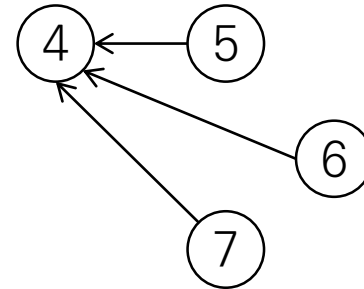
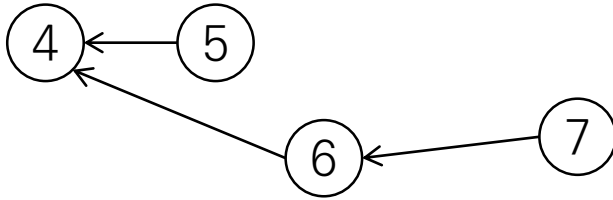
- 7의 대표값.
 - 노드번호와 대표값이 같은 4.

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	6



- 대표값이 같으므로 같은 트리에 속함.
-

■ 깊이 줄이기



노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	6

- 저장된 노드의 부모가 대표 번호가 아니면 대표로 바꿈.

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	4

연습

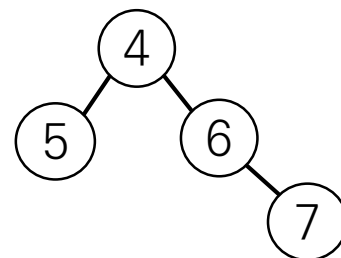
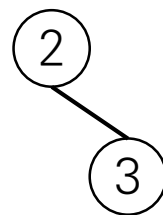
- 1번 부터 N번까지의 노드는 서로 다른 이진 트리에 속할 수 있다고 한다. 트리 정보가 주어지면 몇 개의 트리가 생성 되는지 출력하고, 주어진 노드가 같은 트리에 속하면 1, 아니면 0을 출력한다. 에지의 수와 부모, 자식 정보, 찾을 노드 번호가 주어진다.

입력

4
2 3 4 5 4 6 6 7
3 6

출력

3 0



백트래킹

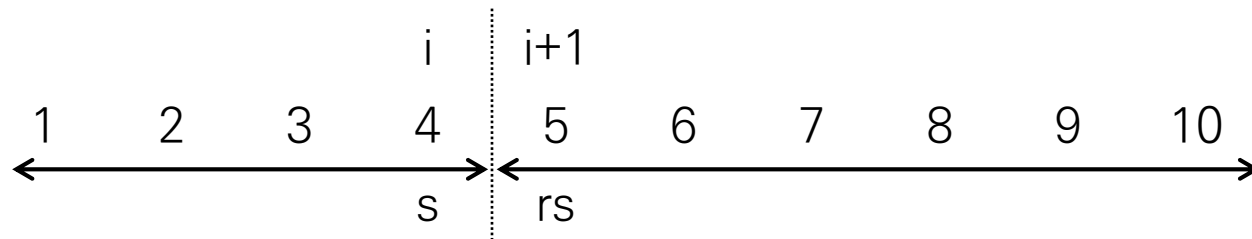
- 재귀의 반복 횟수 줄이기 -

부분 집합의 합

- $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ 의 부분 집합 중 합이 7인 부분 집합의 개수를 구하려면.
 - 각 요소의 포함 여부를 활용해 부분 집합을 구한다음 합이 7인지 검사한다. -> 모든 부분집합을 다 검사해야 함.
 - 모든 요소의 포함 여부를 고려하지 않고도 해를 구할 방법이 필요함.
 - 일부 요소만 고려해서 가능성이 없으면 나머지 요소를 고려할 필요가 없음.
 - 재귀호출을 트리 구조로 나타냈을 때, 일부 노드를 방문할 필요가 없음.
-> '가지치기'라고 부름.
-

■ 부분 집합의 합 문제에서의 가지치기.

- 찾고 있는 합 ts 포함 여부를 고려한 구간 i , 고려한 구간에서 부분 집합에 포함된 요소의 합 s , 남은 구간의 합 rs 라고 한다..



- $s > ts$ 인 경우 리턴. -> 고려 구간의 합이 찾는 수를 초과함.
- $s+a[i+1] > ts$ 인 경우 리턴. -> 남은 구간의 요소 중 가장 작은 수를 더해도 초과 하는 경우..
- $s+rs < ts$ 인 경우 리턴. -> 남은 모든 숫자를 더해도 모자람.

연습

■ 문제설명

1부터 N까지의 정수를 원소로 갖는 집합이 있다.

이 집합의 모든 부분 집합에 대해 원소의 합이 K인 경우의 수를 출력하시오.

N이 5, K=3이면 주어진 집합은 {1,2,3,4,5}가 되고,

원소의 합이 3인 경우는 {1,2}와 {3}이므로 2를 출력한다.

$1 \leq N \leq 100$, $1 \leq K$ 인 정수.

우선 두 번째 까지만
입력해본다.

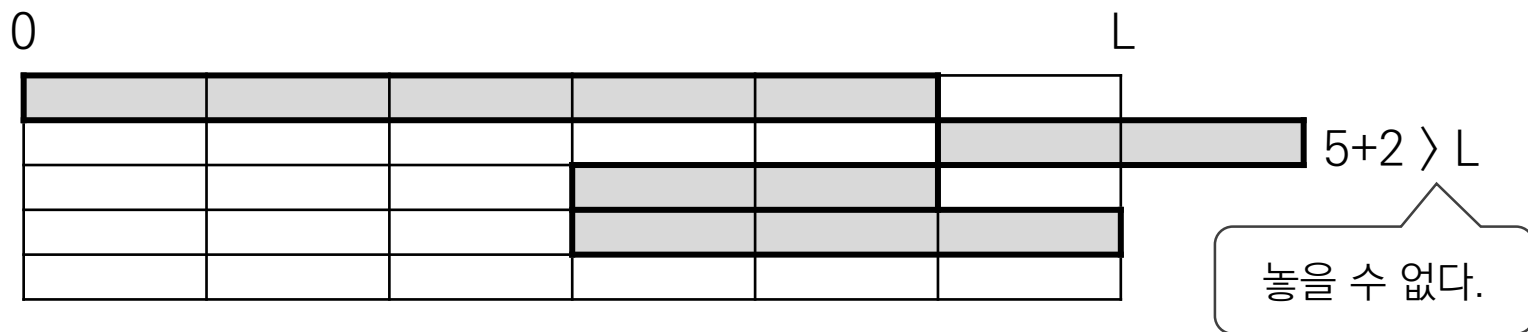
3 <- 빈복 입력 횟수
10 7 <- N과 K
10 53
100 5050

#1 5
#2 1
#3 1

연습

- 그림처럼 0부터 L까지 1cm 마다 눈금이 그려진 판이 있다. 짧은 막대의 길이를 이용해 L에 맞는 길이를 만들 수 있으면 1, 없으면 0을 출력하라. 단, 막대의 위치는 눈금 판을 벗어날 수 없다.

$L = 6, a[] = \{ 2, 3, 5 \}$

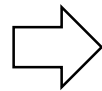


연습

- 다음 문제의 풀이에 백트래킹을 적용해 보세요.
 - 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

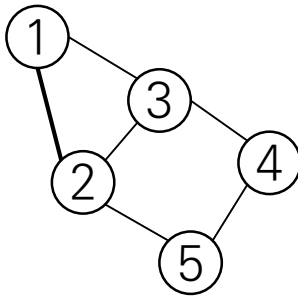
배정에 순열을 활용

그래프

- 그래프 탐색 -

인접행렬을 이용한 그래프 저장

- 무향 그래프 : 간선의 방향이 없는 경우.

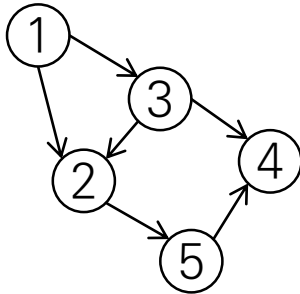


M	1	2	3	4	5
1	0	1	1	0	0
2	1				
3	1				
4	0				
5	0				

인접은 1, 아닌 경우 0

정점 수, 간선 수	5 6	read V, E
간선의 정점 반복	1 2 1 3 2 3 3 4 2 5 5 4	for i : 1 → E read n1, n2; M[n1][n2] = 1; M[n2][n1] = 1;

- 유향 그래프 : 간선에 방향이 있는 경우



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

도착

출발

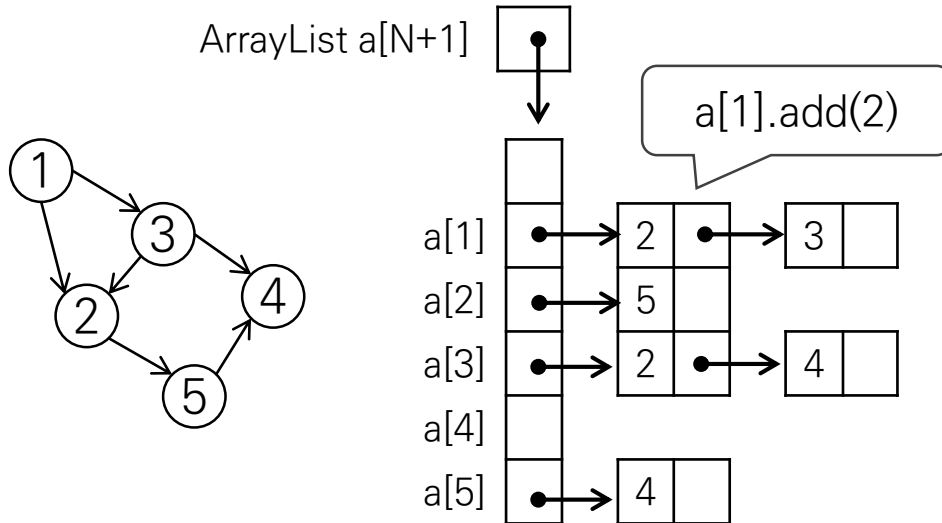
출발 → 도착 노드로 주어지는 경우

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

read n1, n2;
M[n1][n2] = 1;

■ 리스트를 이용한 저장

- 노드 개수가 많은 경우.



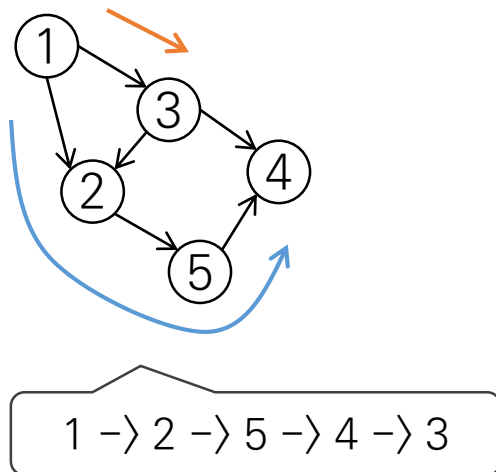
```
static ArrayList<Integer>[ ] a;  
...  
a = new ArrayList[N+1];  
for(int i = 0; i<=N; i++)  
    a[i] = new ArrayList<>();  
for(int i = 0; i<M; i++)  
{  
    int n1 = sc.nextInt();  
    int n2 = sc.nextInt();  
    a[n1].add(n2);  
    // a[n2].add(n1); // 무향  
}
```

```
// t의 인접 노드 번호 n 읽기  
for(int i = 0; i<A[t].size();i++)  
{  
    int n = A[t].get(i);  
}
```

탐색

■ 깊이 우선 탐색 (DFS)

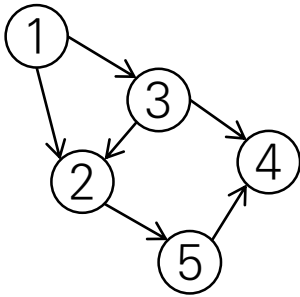
- 2개 이상의 선택이 가능할 때, 정해진 순서에 따라 다음 노드 선택.
- 더 이상 갈 수 없으면 가장 가까운 이전 갈림길에서 다른 방향 선택.
- 지나온 경로를 저장해야 함.



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

■ 재귀를 사용한 DFS

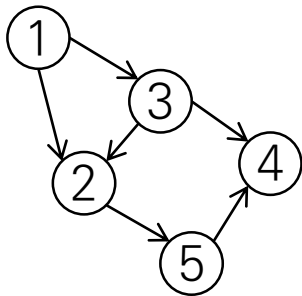
- 재귀의 각 단계가 방문중인 노드 번호를 저장.
- 방문한 노드에서 방문하지 않은 인접 노드 중 번호가 작은 곳으로 이동.



```
DFS(n)
  V[n] = 1           // 방문 표시
  visit(n)           // 노드에 대해 처리할 일
  for i : 1 -> N
    if( M[n][i] == 1 && V[i] == 0 )
      DFS(i) // 인접하고 방문하지 않은 노드로 이동
```

✓반복 구조의 DFS

- 지나온 노드를 스택에 저장하거나, 방문하지 않고 남겨놓은 노드를 스택에 저장.



DFS(s)

시작 노드 s의 모든 인접 i에 대해 push(i)

while(stack_is_not_empty())

 n = pop()

 visit(n)

 n에 대해 i가 인접이고 아직 방문하지 않은 곳이면

 push(i)

Stack : 마지막으로 저장한 데이터를 먼저 꺼내서 사용하는 자료 구조.

✓배열로 구현한 Stack

```
static int s[ ] = new int[STACK_SIZE];  
static top = -1;
```

```
push(int n)  
(  
    if( top == STACK_SIZE -1 )  
        Error(); // 디버깅용 메시지  
    else  
        s[++top] = n;  
}
```

```
int pop( )  
{  
    if(top == -1)  
        Error(); // 디버깅용 메시지  
    else  
        return s[top--];  
}
```

// 코드로 직접 구현

```
static int s[ ] = new int[STACK_SIZE];  
static top = -1;
```

```
// push(n)  
s[++top] = n;
```

```
// pop( )  
n = s[top--];
```

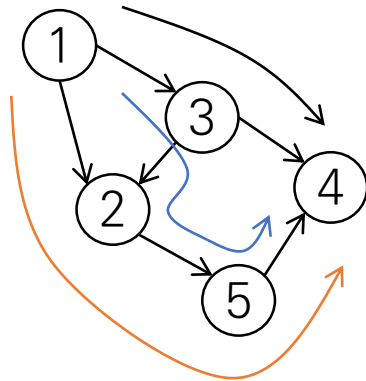
✓ Stack Class 활용

```
import java.util.Stack;

public class Solution {
    static Stack<Integer> s;
    public static void main(String[] args) {
        s = new Stack<>();
        s.push(new Integer(1));
        s.push(new Integer(2));
        s.push(new Integer(3));
        while(!s.empty())
        {
            System.out.println(s.pop());
        }
    }
}
```


■ DFS 응용

- 1에서 4번 노드에 도착할 수 있는 경로의 수 찾기.



가능한 경로

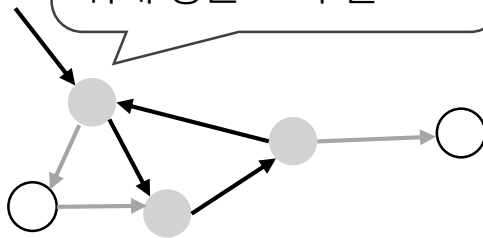
1 → 2 → 5 → 4

1 → 3 → 2 → 5 → 4

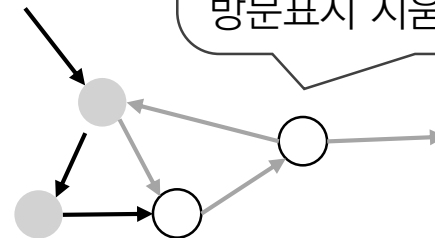
1 → 3 → 4

접근하는 경로가 다르면
중복을 허용해야 함.

되돌아 가는 것을 막기
위해 방문 표시 필요.



다른 경로에서의 접근을 위해
갈림길로 되돌아갈 때는
방문표시 지움.



- 1에서 4번 노드에 도착할 수 있는 경로의 수 찾기.

```
// n = 1, k = 4
```

```
DFS(n, k)
```

```
    if( n == k )
```

```
        cnt++;
```

```
    else
```

```
        V[n] = 1;           // 방문 표시
```

```
        for i : 1 → N
```

```
            if( M[n][i] == 1 && V[i] == 0 )
```

```
                DFS(i, k);    // 인접하고 방문하지 않은 노드로 이동
```

```
        V[n] = 0;           // 방문 표시 삭제
```

for문 밖에서 처리

- 1에서 4번 노드에 도착할 수 있는 최단 거리 찾기.
 - 모든 경로를 찾는 것이 기본.
 - 지나온 간선의 수를 인자로 전달.

```
// 호출 조건 : n = 1, k = 4, e = 0, min = INF
```

```
DFS(n, k, e)
    if( n == k )
        if( min > e )
            min = e;
    else
        V[n] = 1;                // 방문 표시
        for i : 1 → N
            if( M[n][i] == 1 && V[i] == 0 )
                DFS(i, k, e+1);    // 인접하고 방문하지 않은 노드로 이동
        V[n] = 0;                // 방문 표시 삭제
```

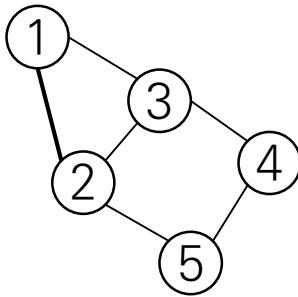
- 1에서 4번 노드에 도착할 수 있는 최단 거리 찾기.
 - 호출을 줄이려면 중단 조건 추가.

```
// 호출 조건 : n = 1, k = 4, e = 0, min = INF
```

```
DFS(n, k, e)
    if( n == k )
        if( min > e )
            min = e;
    else if( e >= min )           // 현재까지 거리가 기존의 min보다 크면 다른 경로
        return;
    else
        V[n] = 1;                // 방문 표시
        for i : 1 -> N
            if( M[n][i] == 1 && V[i] == 0 )
                DFS(i, k, e+1);    // 인접하고 방문하지 않은 노드로 이동
        V[n] = 0;                // 방문 표시 삭제
```

연습

- 다음 그래프를 DFS로 탐색하고 방문 순서를 출력하시오.



입력

노드 수, 간선 수

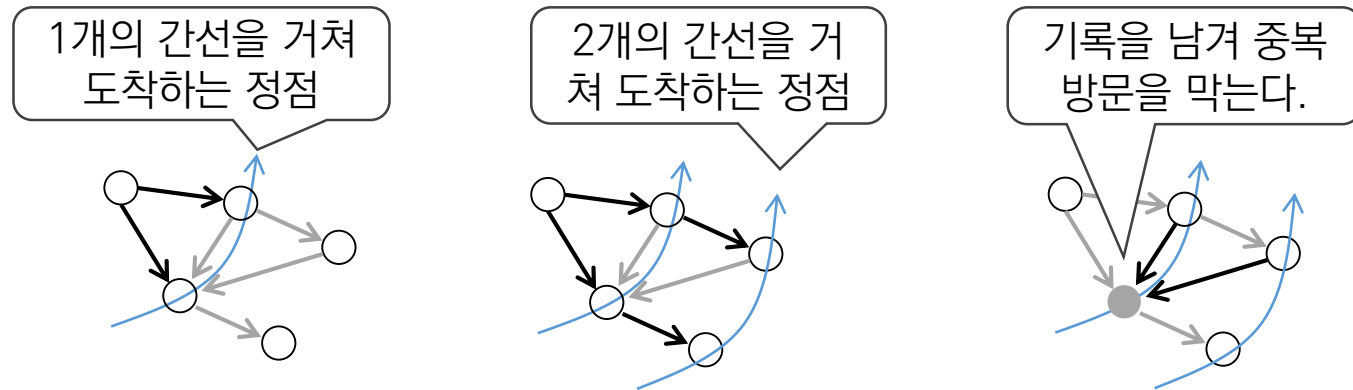
간선 정보

5 6

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

■ BFS (너비 우선 탐색)

- 시작 정점부터 거쳐가는 간선의 수가 같은 순서로 탐색하는 방식.



- 시작에서 n 개의 간선을 지나 도착하는 정점의 인접 정점은 $n+1$ 개의 간선을 지나 도착하게 됨.
- 거리가 n 인 정점들을 처리할 때 $n+1$ 인 인접 정점들을 저장함.
- 거리 n 인 정점들을 처리하면, 저장해둔 $n+1$ 정점들을 꺼내 처리함.

■ BFS

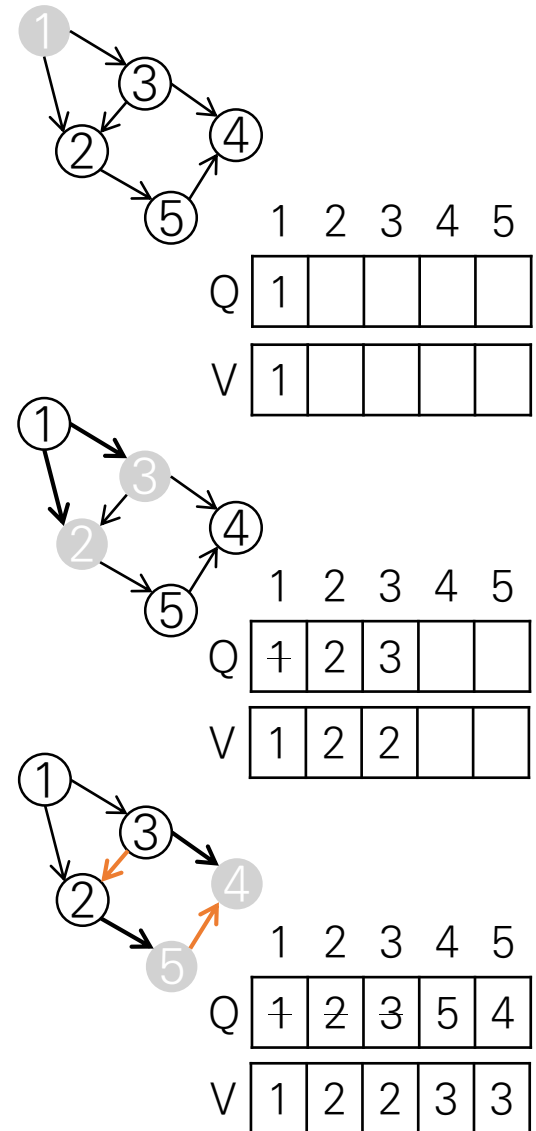
BFS(s)

```

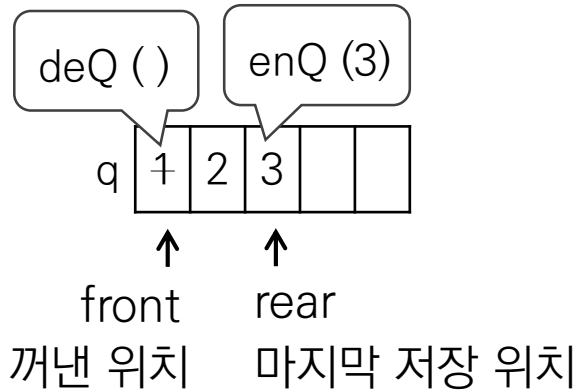
    enQ(s)                // 시작점 enqueue
    V[s] = 1              // 방문 표시
    while( is_not_emptyQ() )
        n = deQ()
        visit(n)          // 노드에 대해 처리할 일
        for i : 1 → N
            // i가 인접이면서 방문안한 노드면
            if( A[n][i] == 1 && V[i] == 0 )
                enQ(i)
                V[i] = V[n] + 1

```

‘V[i] = 1’ 대신 ‘V[i] = V[n] + 1’로
표시하면 인접한 정점으로 부터의 거
리를 알 수 있다.



✓배열을 이용한 큐(queue)



```
enQ(int n)
    if( rear == Q_SIZE-1 )
        Error!    // 가득참
    else
        q[ ++rear ] = n

int deQ()
    if( front == rear ) // 비었음
        Error!
    else
        return q[ ++front ]
```

```
int front = -1;
int rear = -1;
int q[Q_SIZE];
```

```
q[ ++rear ] = 1; //enqueue
q[ ++rear ] = 2;
q[ ++rear ] = 3;
```

```
while( front != rear )    // 큐가 비어있지 않으면
    print(q[ ++front ]); // dequeue
```


✓ Queue 클래스

```
import java.util.LinkedList;  
import java.util.Queue;  
...
```

```
Queue <Integer> q = new LinkedList<>();
```

```
q.add(1); // enqueue( )
```

```
q.add(2);
```

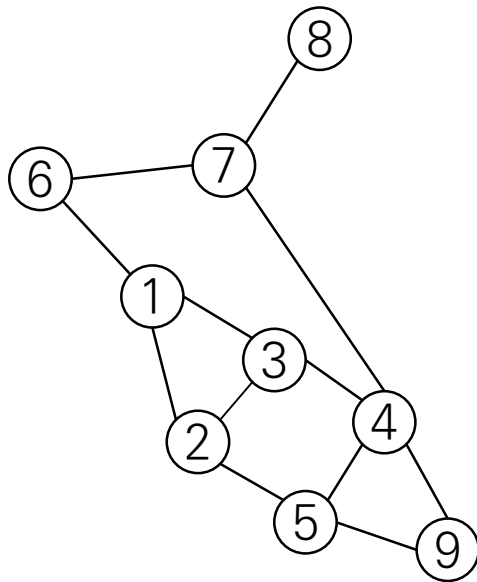
```
q.add(3);
```

```
while(!q.isEmpty())
```

```
    System.out.println(q.poll());    // dequeue( )
```

연습

- 다음 그래프의 1번 노드에서 다른 노드까지 최소한의 간선을 거쳐 도착한다고 할 때, 지나는 간선 수의 총 합을 구하시오.



각 노드까지의 거리

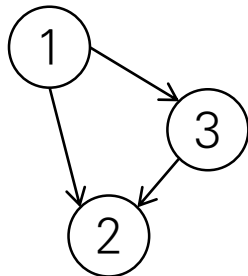
1	2	3	4	5	6	7	8	9
0	1	1	2	2	1	2	3	3

9 12

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

위상 정렬

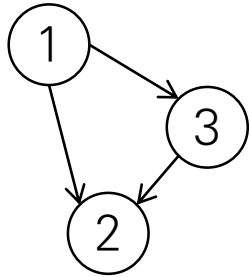
- 앞의 1, 3이 처리되어야 2번을 처리할 수 있는 경우.
 - 정점의 진입 차수를 활용.
 - 진입 차수가 0인 정점부터 시작.
 - 정점을 처리할 때 인접 정점에 처리되었음을 알림.
 - 인접 정점의 진입 차수를 하나 줄임.
 - 진입 차수가 0이 되면 다음 번에 처리할 차례가 됨.



A	1	2	3
1	0	1	1
2	0	0	0
3	0	1	0

I	0	2	1
---	---	---	---

A : 인접행렬
I : 진입 차수



A : 인접행렬
I : 진입 차수

```

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

Q	1						
---	---	--	--	--	--	--	--

I	0	2	1
---	---	---	---

Q	1	3					
---	---	---	--	--	--	--	--

I	0	2->1	1->0
---	---	------	------

Q	1	3	2				
---	---	---	---	--	--	--	--

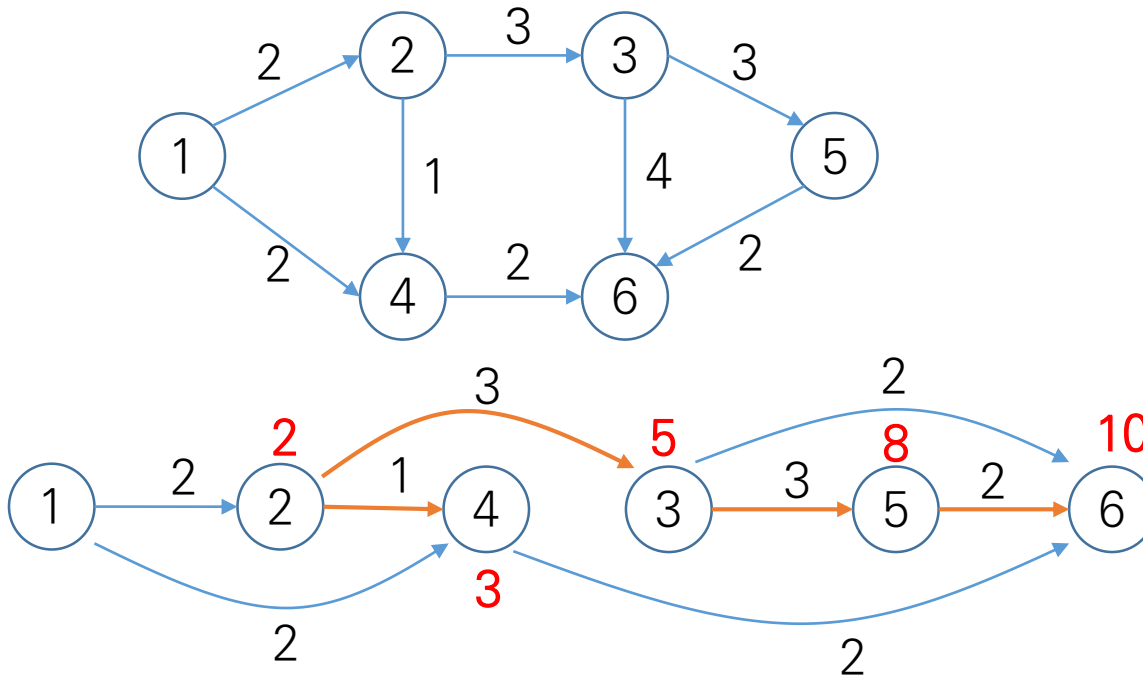
I	0	1->0	0
---	---	------	---

Q	1	3	2				
---	---	---	---	--	--	--	--

I	0	0	0
---	---	---	---

■ 최장거리 구하기

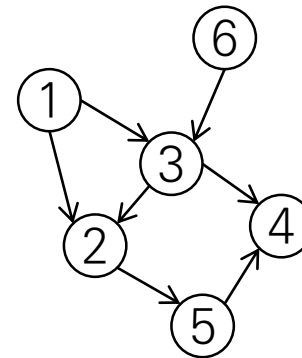
- 조건 : 사이클이 없는 방향성 그래프(DAG)
- 시작 노드 1, 도착 노드 6.
- 위상 정렬을 사용해 거리 계산 순서를 결정한다.



연습

- 6명이 사람이 각자 갖고 있는 100원짜리 동전의 개수를 비교했더니 다음과 같은 조건이 되었다. 1번과 6번이 갖고 있는 금액이 100원 이었다면, 가장 많은 동전을 가진 사람은 최소한 몇 개의 동전을 갖고 있었는가?

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



미로

- 2차원 배열 활용 -

오른쪽과 아래로만 이동하는 경우

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

	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

출발

도착

■ 다음 칸의 좌표 계산.

- 현재 위치가 (0, 0)이면, 갈 수 있는 칸은 (0, 1)이나 (1, 0)이다.
- 재귀 호출의 각 단계에서는 현재 위치를 저장한다.
- 오른쪽 칸으로 가는 경우와 아래로 가는 경우를 나누어 호출한다.
- 현재 칸까지의 숫자의 합을 항상 구한다.

	0	1
0	5	6
1	2	7

A 2x2 grid representing a small map. The columns are indexed 0 and 1, and the rows are indexed 0 and 1. The values in the cells are 5, 6, 2, and 7 respectively. A black dot is placed at the center of the cell (0,0) which contains the number 5. From this dot, two arrows originate: one pointing horizontally to the right towards the cell (0,1) containing 6, and one pointing vertically downwards towards the cell (1,0) containing 2.

```
f( row, col, sum)
{
    if( 도착점이면 )
        if( sum+m[row][col] < min )
            min = sum + m[row][col];
    else
        if(col+1<N)
            f( row, col + 1, sum + m[row][col]);
        if(row+1<N)
            f( row + 1, col, sum + m[row][col]);
}
```

- 모든 칸을 지나지 않고 답을 찾는 방법.
 - 도착점이 아닌 경우, 지나온 숫자의 합이 min보다 크면 return.

```
f( row, col, sum)
{
    if( 도착점이면 )
        if( sum+m[row][col] < min )
            min = sum + m[row][col];
        else if( sum + m[row][col] > min )
            return;
    else
        f( row, col + 1, sum + m[row][col]);
        f( row + 1, col, sum + m[row][col]);
}
```

미로의 응용

■ 배열에 저장한 미로

- 1-벽, 0-통로, S-출발, G-도착
- 도착 가능 여부 판단, 최단 거리 구하기, 경로의 수 구하기 등.

	0	1	2	3	4	column
0	S	1	1	1	1	
1	0	0	1	0	G	
2	1	0	1	0	1	
3	1	0	0	0	0	
4	1	0	1	1	0	
row						

■ 미로에서의 이동

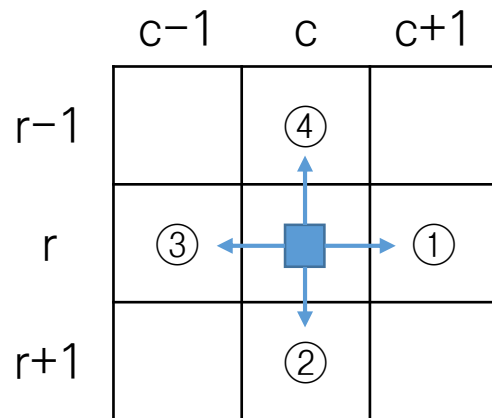
	c	0	1	2
r	0	① → ×		
	1	② → ③ → ④		
	2	1	G	

	r	c	
①	0	0	오른쪽 불가. 아래 가능.
②	1	0	오른쪽 가능.
③	1	1	오른쪽 가능.

	c	0	1	2
r	0	①		×
	1	②	③ ← ×	④ → ×
	2	1	G	×

	r	c	
④	1	2	오른쪽/아래 불가. 왼쪽 이미 방문. 위쪽 불가.
③	1	1	이전 위치로 되돌아감. 아래 가능.
G	2	1	도착

■ 이동할 칸의 좌표 계산



//크기가 N×N인 2차원 배열 일 때,
//현재 위치 (r, c)에서 새 좌표로 이동하기.

```
if ( c+1 < N )  
    next( r, c+1 ); // ①  
if ( r+1 < N )  
    next( r+1, c ); // ②  
if ( c > 0 )  
    next( r, c-1 ); // ③  
if ( r > 0 )  
    next( r-1, c ); // ④
```

벽으로 둘러싸인 미로의 경우 유효
좌표 검사가 필요없다.

■ 반복문을 이용한 이동할 칸의 좌표 계산

```
int dr[] = { 0, 1, 0, -1 };  
int dc[] = { 1, 0, -1, 0};
```

...

```
for( i = 0 ; i < 4 ; i++ )  
{
```

```
    nr = r + dr[i];
```

```
    nc = c + dc[i];
```

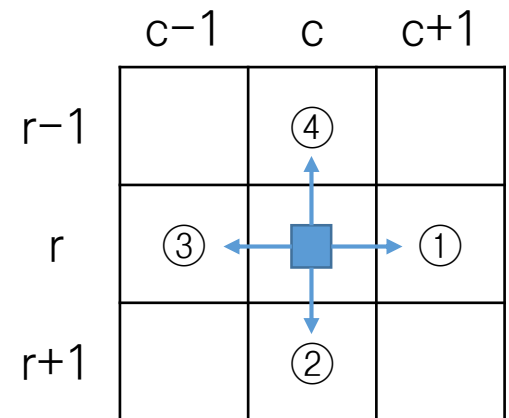
```
    if(( nr >= 0 )&&( nr < N )&&( nc >= 0 )&&( nc < N ))
```

```
    {
```

```
        next( nr, nc );
```

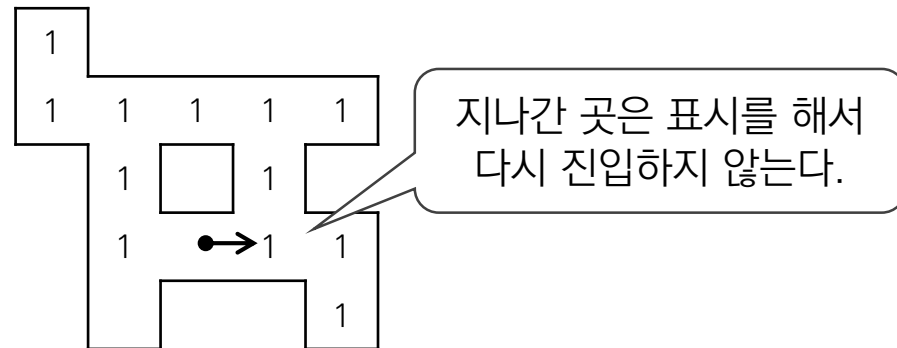
```
    }
```

```
}
```



■ 경로의 존재만 확인하는 경우

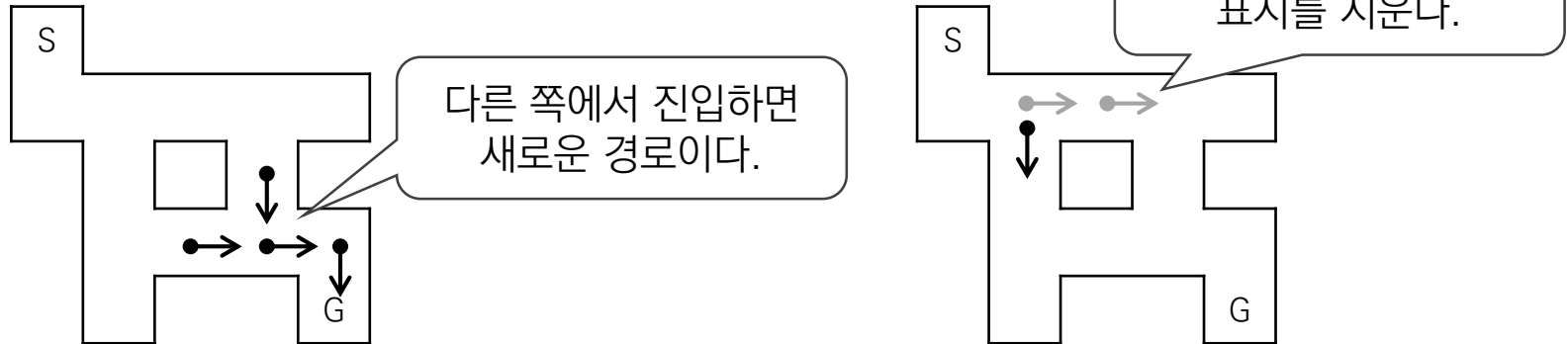
- 한번 지나간 칸은 표시를 해서 다시 들어가지 않는다.
- 일단 목적지에 도착하면 나머지 경로는 확인하지 않는다.



```
map[r][c] = 1; // 현재 위치를 벽으로 변경
for i : 1 -> 4
    //새 좌표 계산
    ...
    next( nr, nc );
```

■ 경로의 수를 찾는 경우.

- 목적지에 도착 가능한 모든 경로를 지나야 한다.



```
map[r][c] = 1; // 현재 위치를 벽으로 표시
for i : 1 -> 4
    //새 좌표 계산
    ...
    next( nr, nc );
map[r][c] = 0; // 되돌아 가기 전에 표시 지움
```

방문 표시가 없으면 무한 반복의 위험이 있다.

연습

- 숫자가 적혀 있는 2차원 배열이 있다. 이웃한 칸으로 움직여 1 2 3 4 5 6 3 이란 수열을 찾을 수 있으면 1, 없으면 0을 출력하라. 배열안의 숫자는 1번씩만 사용할 수 있고, 대각선으로는 이동할 수 없다.

0	0	0	0	0
0	1	2	0	0
3	6	3	0	0
0	5	4	0	0
0	0	0	0	0

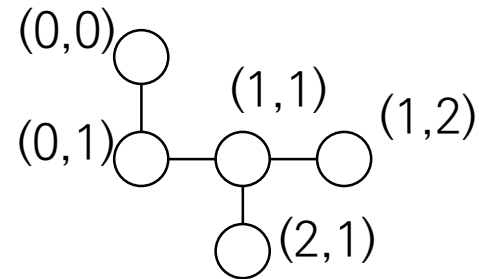
수열을 찾을 수 있는 경우

0	0	0	0	0
0	1	2	0	0
0	0	3	6	0
0	0	4	5	0
0	0	0	0	0

수열을 찾을 수 없는 경우

■ 미로와 그래프.

	0	1	2
0	S		
1	0	0	0
2		0	



상하좌우만 인접할 수 있기 때문에
인접행렬이 필요 없음.

■ BFS를 적용하는 경우

	0	1	2
0	0		
1	1	2	3
2		3	

시작 위치부터 거리가 같은 곳을
찾거나 최단 거리를 찾을 수 있다.

✓ 좌표를 큐에 저장하기.

배열

```
// enqueue
rear++;
qr[rear] = row;
qc[rear] = col;

// dequeue
front++;
row = qr[front];
col = qc[front];
```

Point Class

```
import java.awt.Point;

Queue<Point> q = new LinkedList<>();

q.add(new Point(row, col)); // enqueue

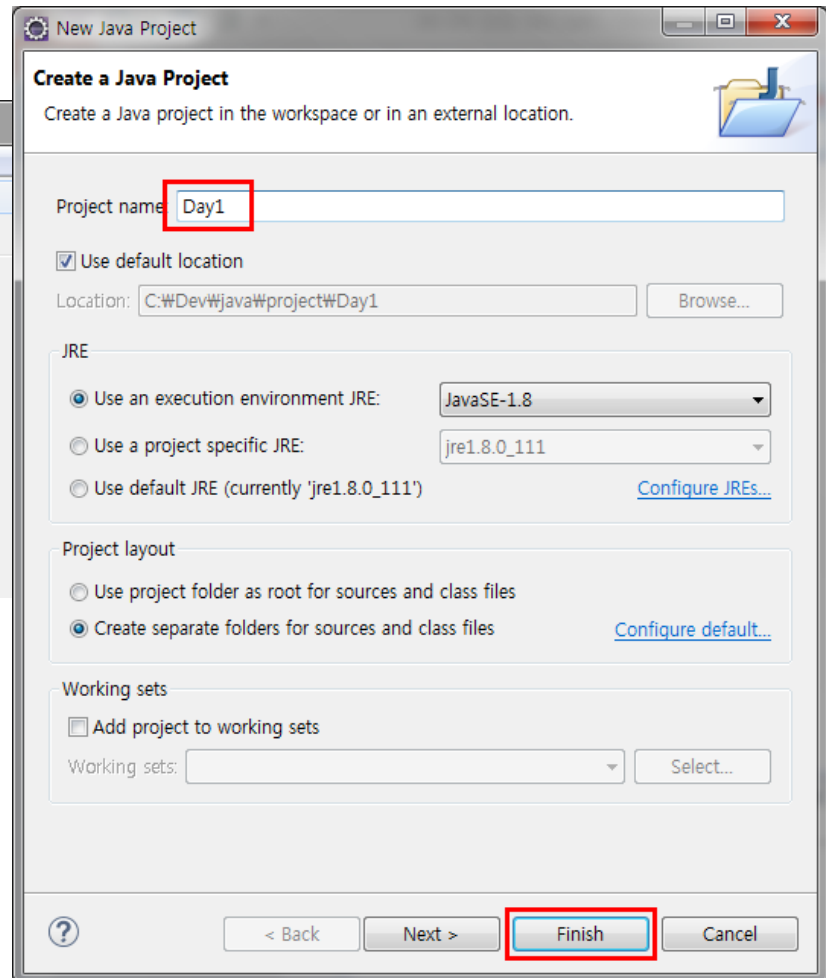
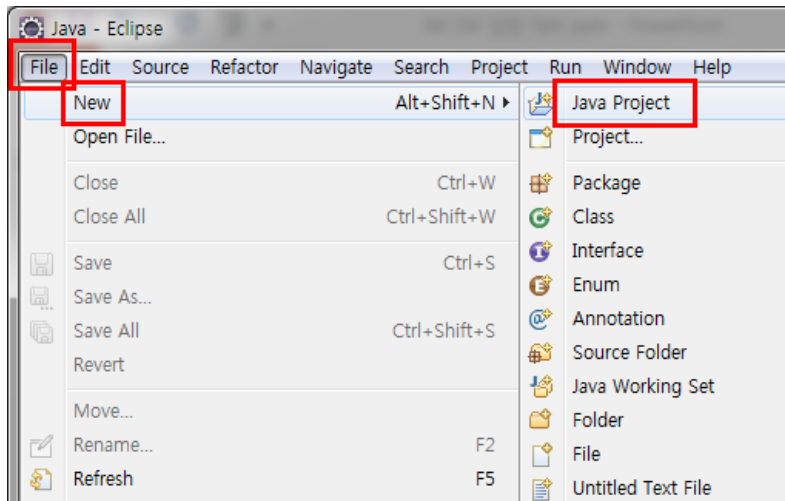
Point p = q.poll();        // dequeue

row = p.x;
col = p.y;
```

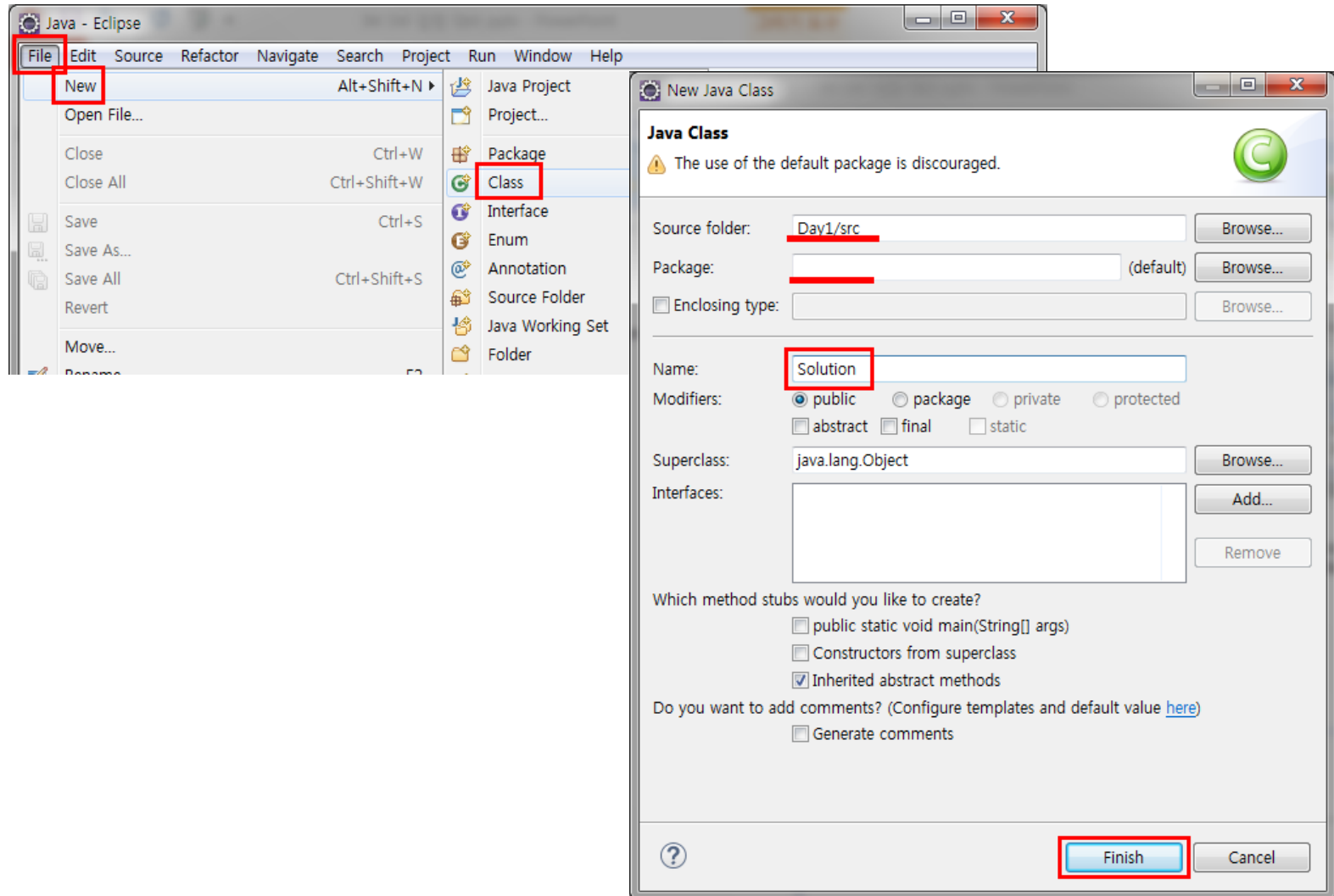
부록

Eclipse

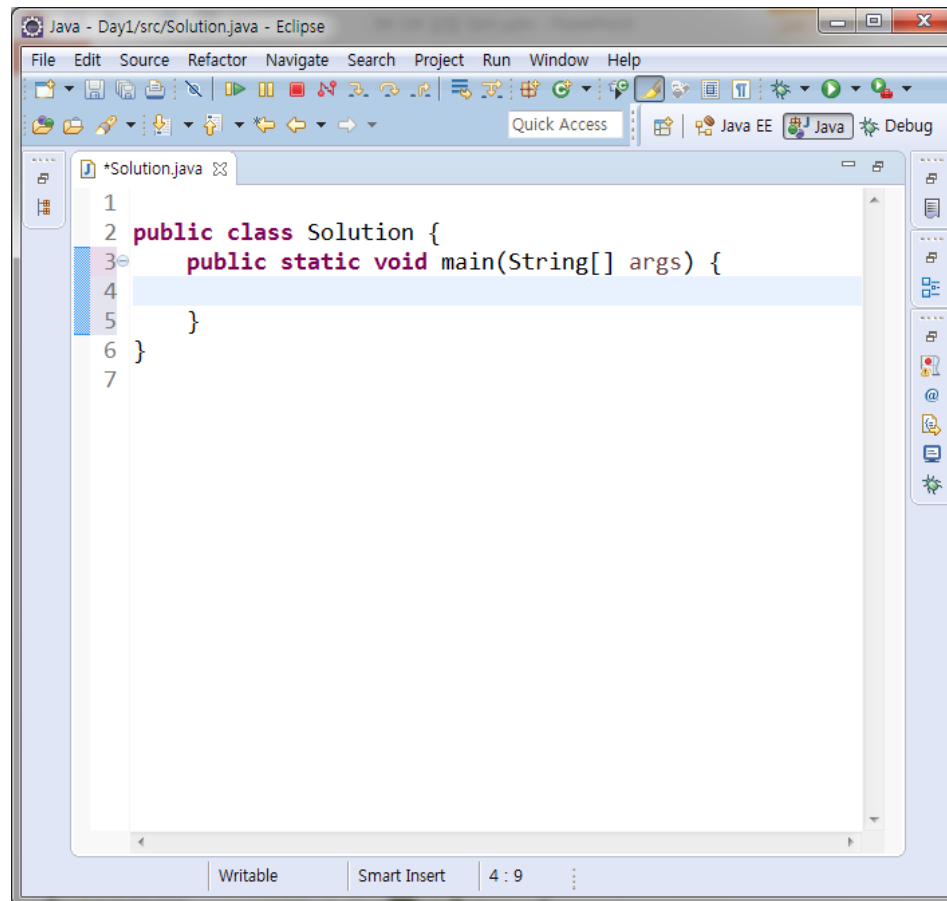
■ Project 만들기



■ Class 생성



■ Class 작성



The screenshot shows the Eclipse IDE interface. The title bar reads "Java - Day1/src/Solution.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, editing, and running. The editor window displays the following Java code:

```
1  
2 public class Solution {  
3     public static void main(String[] args) {  
4  
5     }  
6 }  
7
```

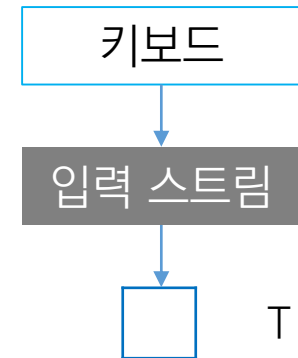
The status bar at the bottom indicates "Writable", "Smart Insert", and "4 : 9".

■ 콘솔 입력에 의한 초기화

```
import java.util.Scanner;

class Solution {
    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();
        ...
    }
}
```



■ 파일에서 가져오기.

```
import java.util.Scanner;  
import java.io.FileInputStream;
```

```
class Solution {  
    public static void main(String args[]) {
```

```
        Scanner sc = new Scanner(new FileInputStream("input.txt"));
```

```
        int T = sc.nextInt();
```

```
        ...
```

프로젝트 폴더에
있어야 함.

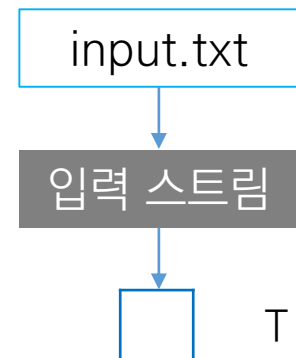
input.txt

10

// 다음 형식도 가능

```
System.setIn(new FileInputStream("input.txt"));
```

```
Scanner sc = new Scanner(System.in);
```



■ 변수의 출력

```
System.out.print(T);  
System.out.println();  
System.out.println(T);  
System.out.printf("%d", T)
```

서식문자	출력 형태
%d	부호 있는 10진수 정수
%o	부호 없는 8진수 정수
%x	부호 없는 16진수 정수
%f	부호 있는 10진수 실수
%e	e 표기법 기반의 실수
%s	문자열
%c	문자

■ 실습

- char, byte, float, double 형 변수를 선언하고 값을 출력해보기.

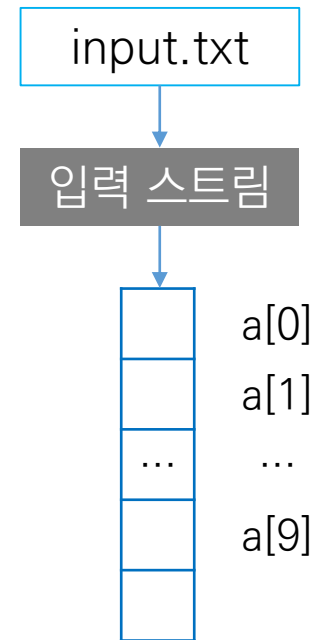
■ 정수형 데이터 입력

- 키보드로는 디버깅을 위한 다량의 데이터 입력이 어려움.

```
import java.util.Scanner;
import java.io.FileInputStream;

class Solution {
    public static void main(String args[]) {
        System.setIn(new FileInputStream("input.txt"));
        Scanner sc = new Scanner(System.in);

        Scanner sc = new Scanner(System.in);
        for(int i = 0; i < 10; i++)
        {
            a[i] = sc.nextInt();
        }
    }
    ...
}
```



■ 공백 없는 숫자 입력

- ‘0에서 9까지의 숫자가 적힌 5장의 카드’의 예

```
int N = sc.nextInt();
String str = sc.next();

for(int i = 0; i < N; i++)
{
    a[i] = str.charAt(i) - '0';
}
```

Text.txt

5 49679

a

4	9	6	7	9
---	---	---	---	---

■ Eclipse 디버거

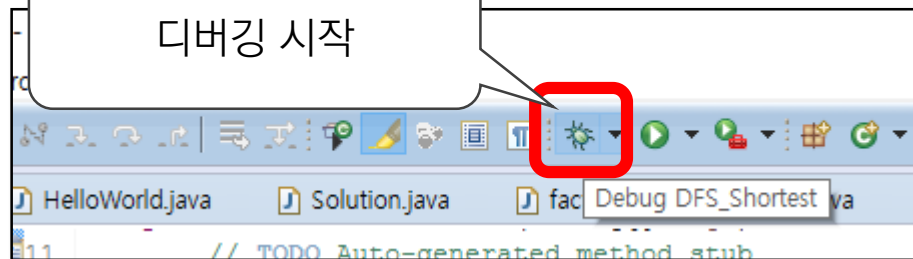
중단점(break point)를
설정할 위치를 더블 클릭.

```
17     for(int tcase=1;tcase<=T;tcase++)
18     {
19         N = sc.nextInt();
20         max = 99999;
21         for(int i=0;i<N;i++)
22         {
23             String line = sc.next();
24             for(int j=0;j<N;j++)
25             {
```

중단점(break point)
생성.

```
17     for(int tcase=1;tcase<=T;tcase++)
18     {
19         N = sc.nextInt();
20         max = 99999;
21         for(int i=0;i<N;i++)
22         {
23             String line = sc.next();
24             for(int j=0;j<N;j++)
25             {
```

디버깅 시작

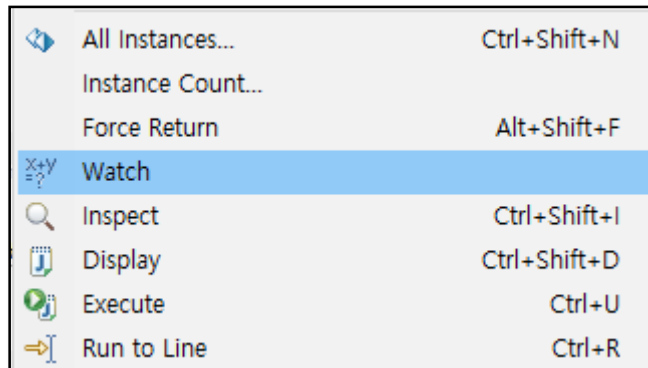


■ Eclipse 디버거 (계속)

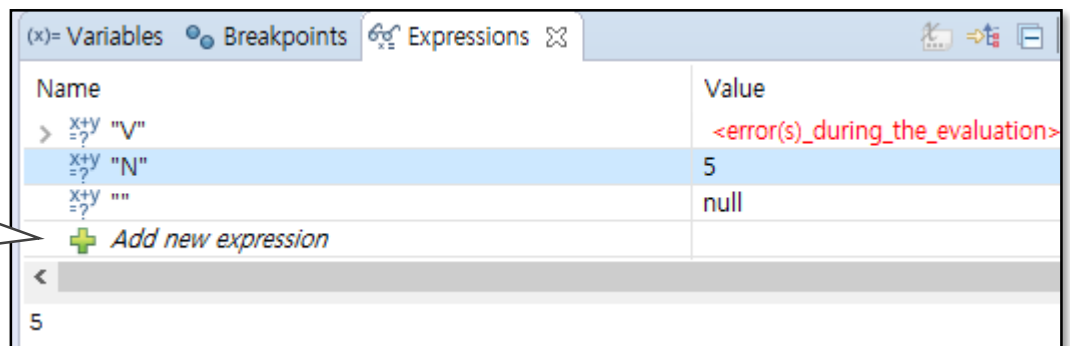
중단점에서 실행 멈춤.

```
17         for(int tcase=1;tcase<=T;tcase++)
18         {
19             N = sc.nextInt();
20             max = 99999;
21             for(int i=0;i<N;i++)
22             {
23                 String line = sc.next();
24                 for(int j=0;j<N;j++)
25                 {
```

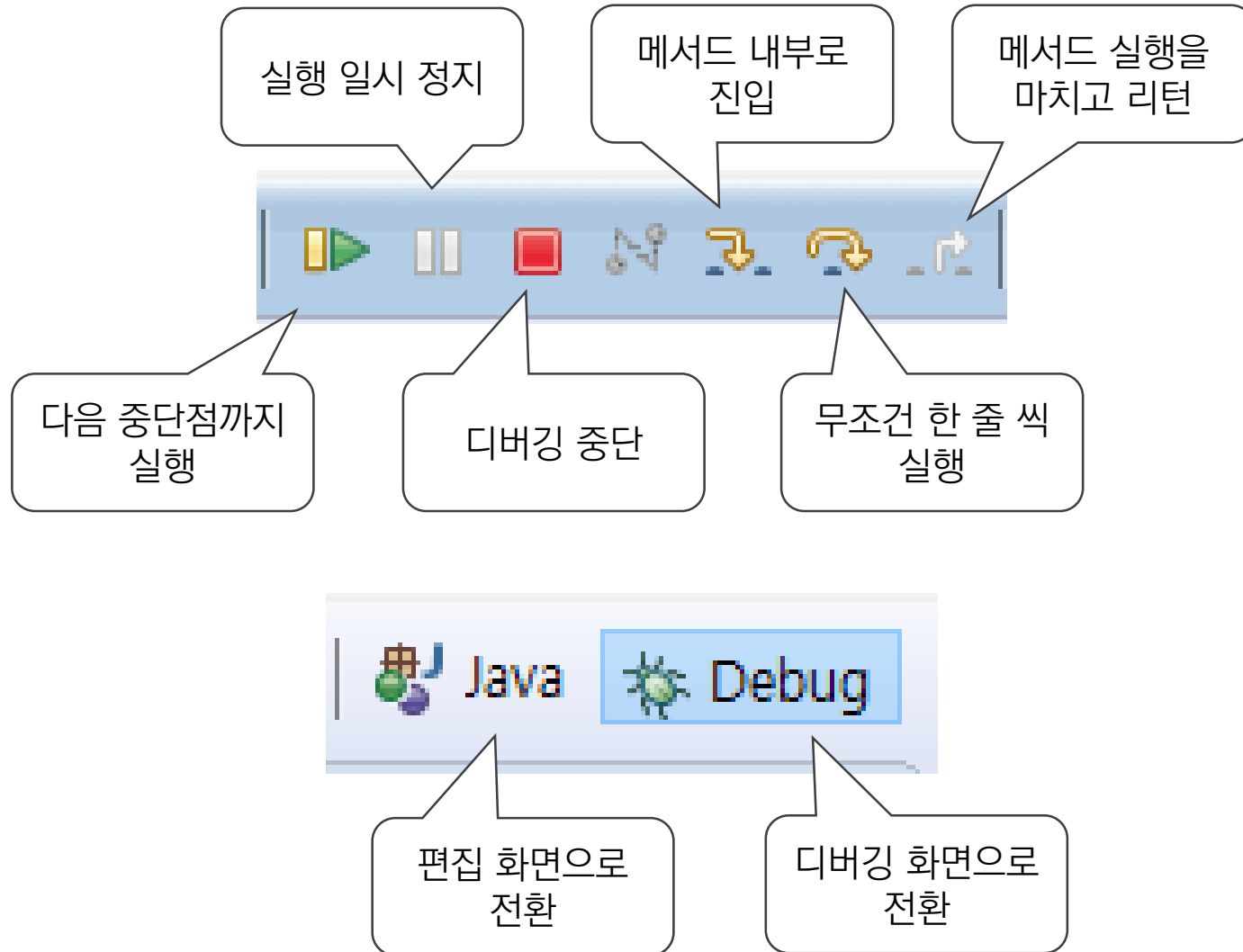
편집기에서 오른쪽 클릭 후
Watch 선택.



Expressions 창에
변수를 추가.



■ Eclipse 디버거 (계속)



■ 조건에 의한 중단

