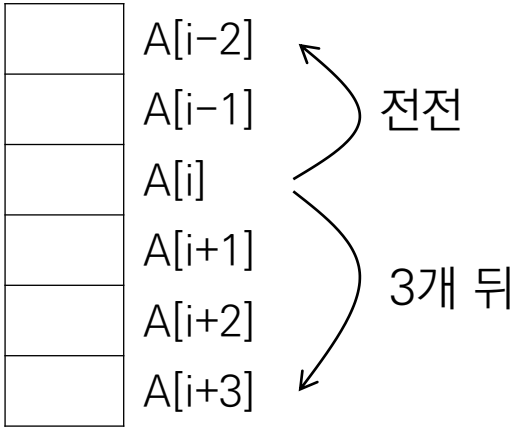
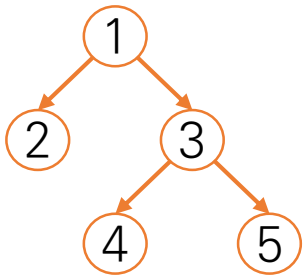
이진 트리

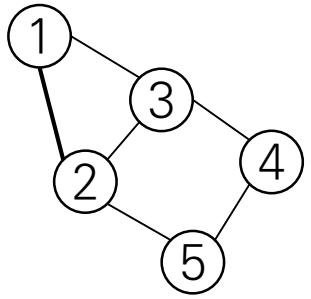
자료구조

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



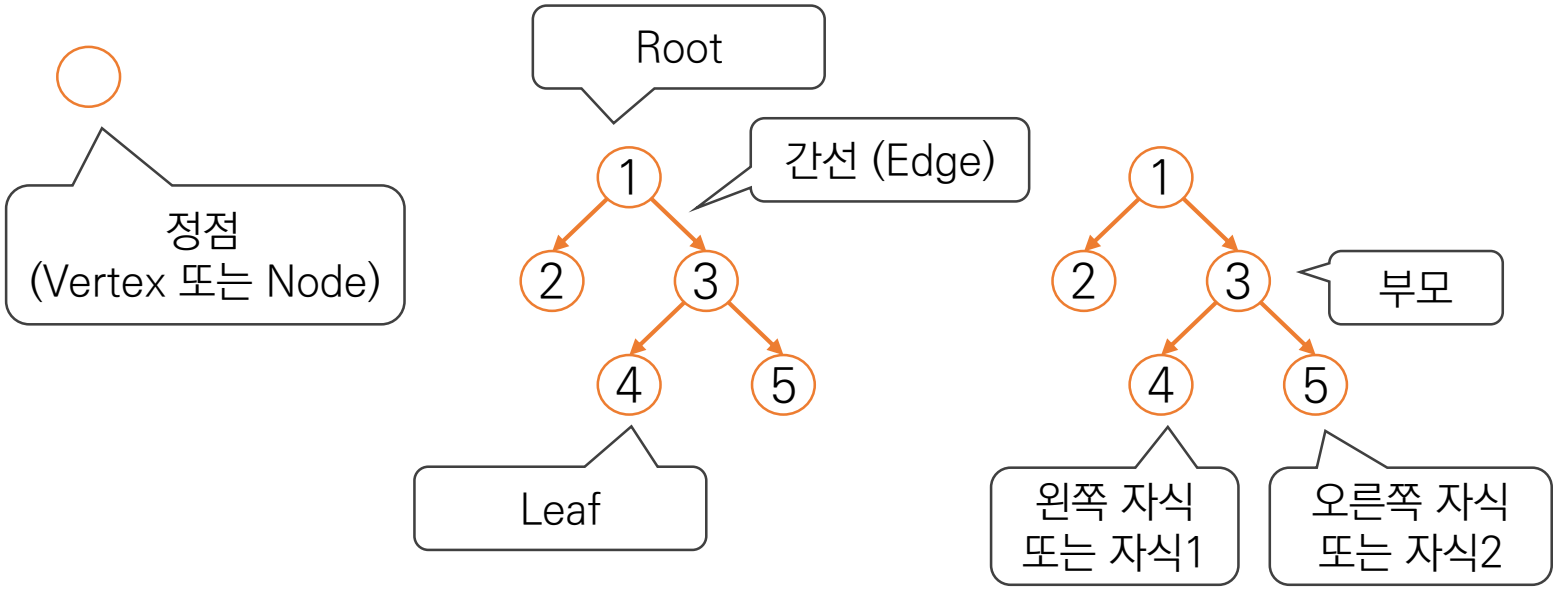
- 비선형 자료구조
  - 1:N 관계를 표시하는 경우.
    - 이진트리, 힙.
  - N:N 관계를 표시하는 경우.
    - 그래프.





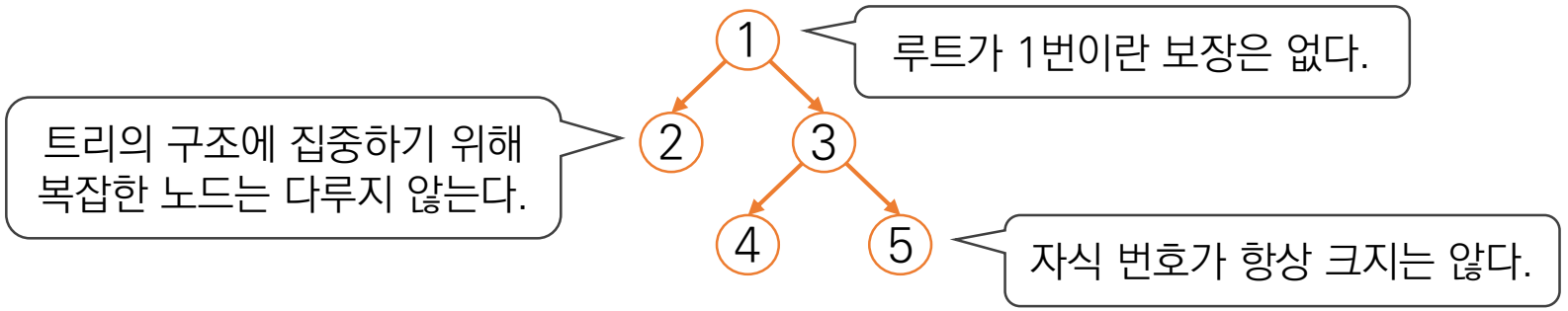
이진트리

- 1:2 관계로 나타낸 트리 구조.



■ 이진트리 (계속)

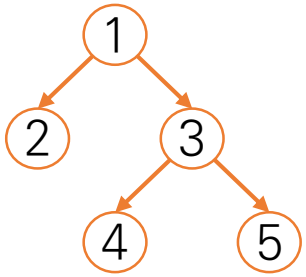
- 조상
  - 어떤 정점에서 루트까지의 경로에 있는 정점들.
  - 4의 조상 : 3, 1
- 자손
  - 어떤 정점 아래에 있는 정점들.
  - 3의 자손 : 4, 5



▪ 트리 정보

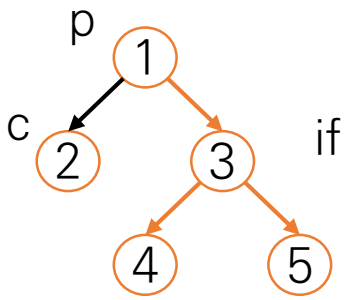
- 입력

```
4 <- 간선의 개수 N
1 2 1 3 3 4 3 5 <- 부모 자식 순
```



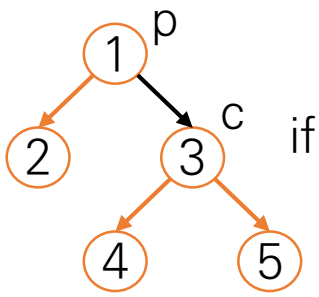
■ 부모 번호를 인덱스로 자식 번호를 저장.

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



if (c1[p] == 0)
c1[p] ← c

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



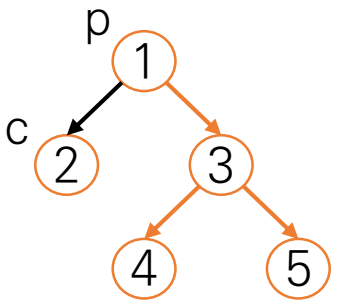
if (c1[p] != 0)
c2[p] ← c

```
for i : 1 -> N
        read p, c;
        if( c1[p] == 0)
                c1[p] = c;
        else
                c2[p] = c;
```

- 자식 번호를 인덱스로 부모 번호를 저장.

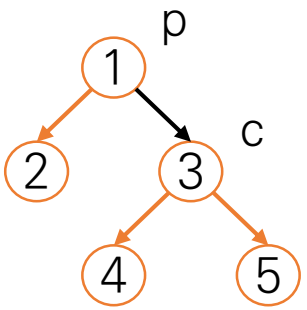
자식	c	0	1	2	3	4	5
부모	a	0	0	1	0	0	0

a[c] ← p



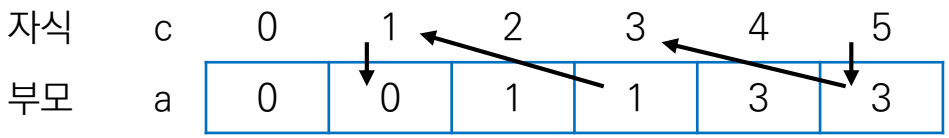
자식	c	0	1	2	3	4	5
부모	a	0	0	1	1	0	0

a[c] ← p

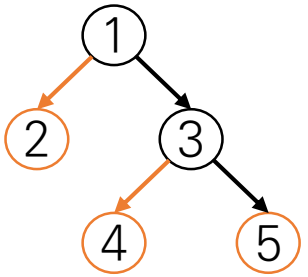



```
for i : 1 -〉 N
      read p, c;
      pa[c] = p;
```

- 루트 찾기
  - 부모가 없는 노드를 찾으면 됨.
  - 부모 노드의 번호가 0인 노드를 찾음.
- 조상 노드 찾기



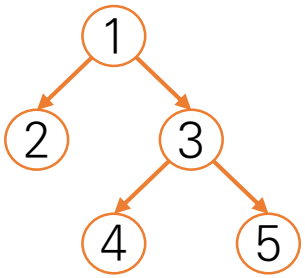
5번 노드의 조상 찾기




```
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);
}
```

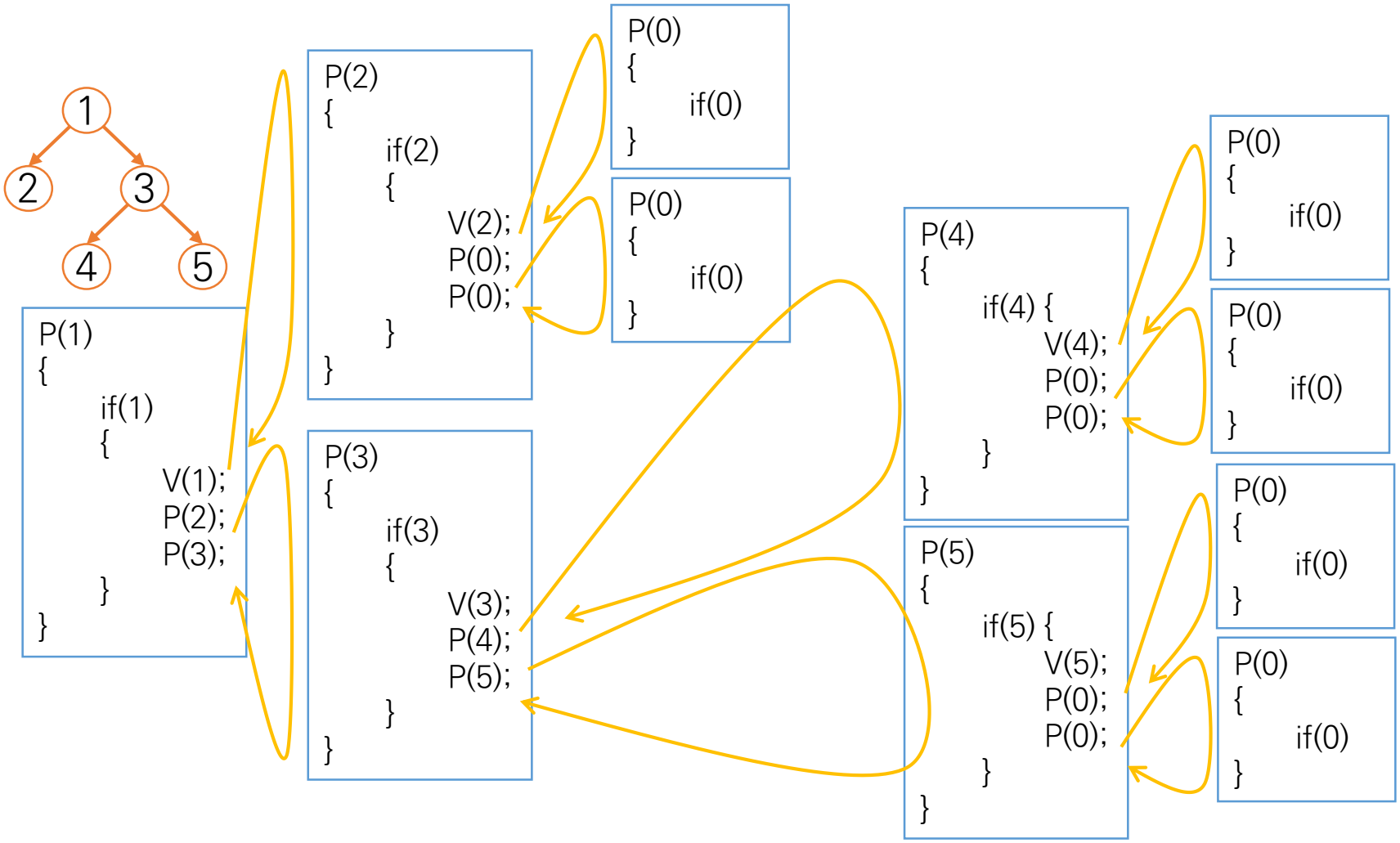
```
DLR(2)
{
    Visit( );
    if Left
        DLR( );
    if Right
        DLR( );
}
```

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

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

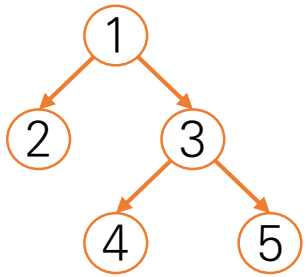
```
DLR(5)
{
    Visit( );
    if Left
        DLR( );
    if Right
        DLR( );
}
```

- 이진 트리 순회 2


1
2
3
4
5
P(1)
{
if(1)
{
V(1);
P(2);
P(3);
}
}
P(2)
{
if(2)
{
V(2);
P(0);
P(0);
}
}
P(0)
{
if(0)
}
P(0)
{
if(0)
}
P(3)
{
if(3)
{
V(3);
P(4);
P(5);
}
}
P(4)
{
if(4) {
V(4);
P(0);
P(0);
}
}
P(0)
{
if(0)
}
P(0)
{
if(0)
}
P(5)
{
if(5) {
V(5);
P(0);
P(0);
}
}
P(0)
{
if(0)
}
P(0)
{
if(0)
}

연습

- 1번 노드부터 이진트리를 순회하고 방문한 노드의 개수를 출력하시오.



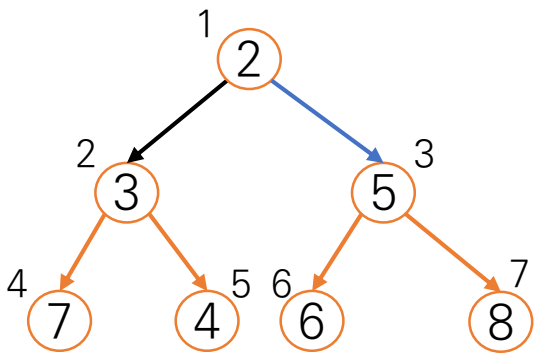
〈참고〉

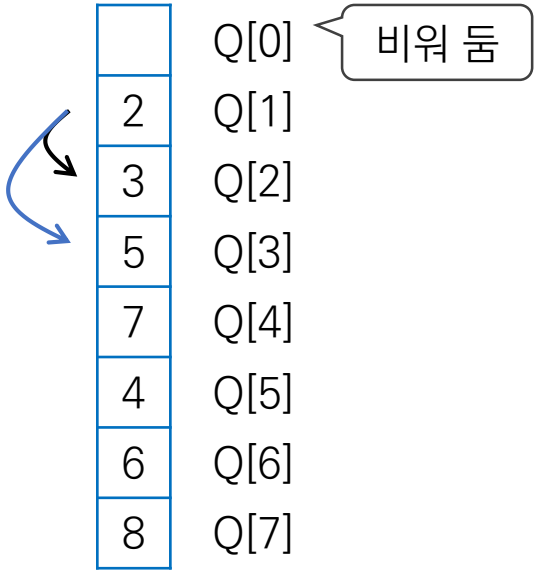
```
P(1)
{
        if(1)
        {
                cnt++;
                P(2);
                P(3);
        }
}
```

포화/완전 이진트리

■ 포화 이진트리

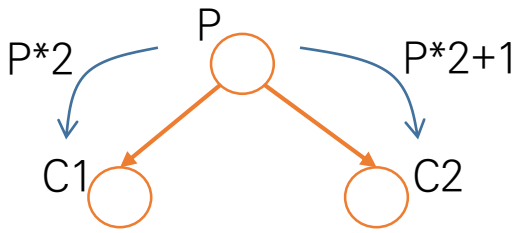
- 특정 높이까지 꽉 차 있는 이진 트리.
- 노드 번호는 위->아래, 왼쪽->오른쪽 순서.
- 노드 번호를 배열의 인덱스로 사용해 저장.





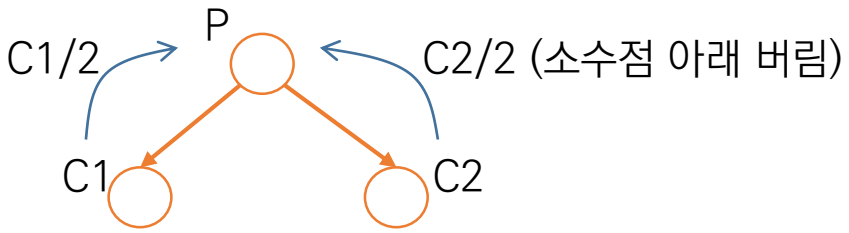
포화 이진트리와 저장 방법

▪ 부모-자식 노드 번호 계산



부모->자식

1->2, 1->3

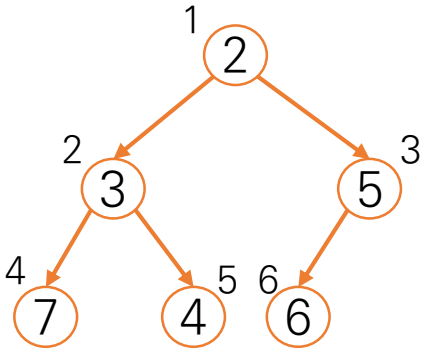


자식->부모

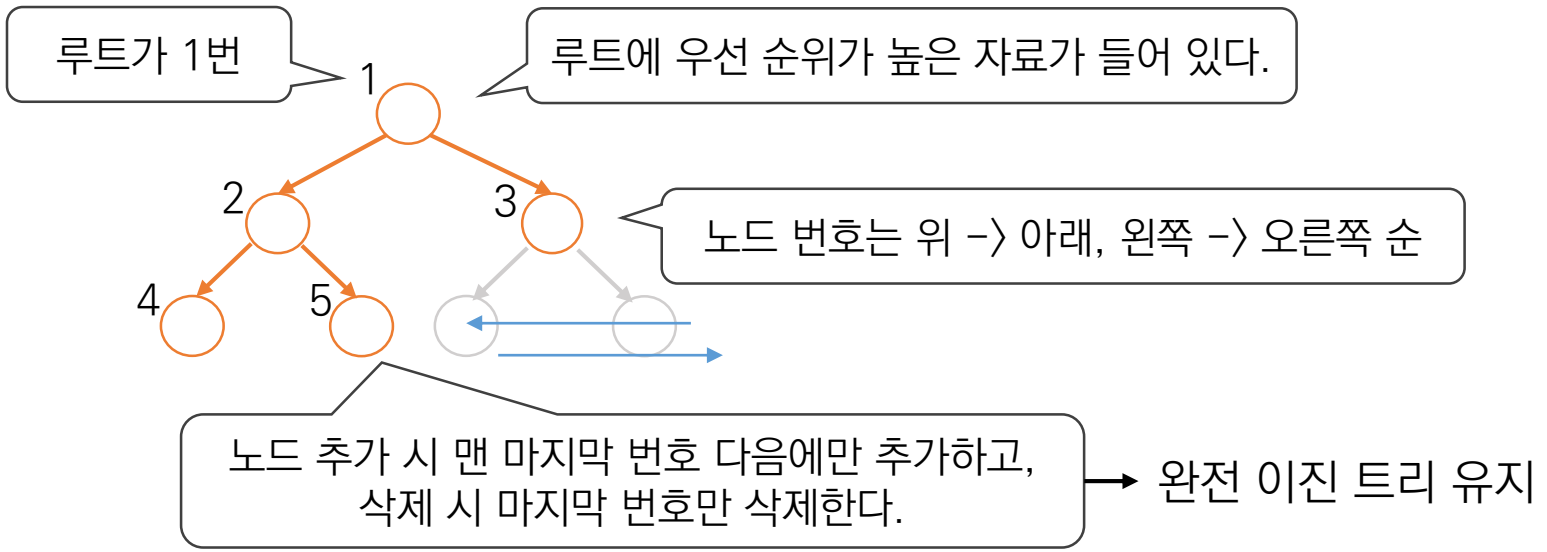
2->1, 3->1

▪ 완전 이진트리

- 오른쪽 끝부터 노드가 비어있는 이진 트리.
- 포화 이진트리와 같은 방법으로 저장.
- 마지막 노드 번호를 별도로 관리.



- 이진 힙 (binary heap)
  - 완전 이진트리 유지.
  - 루트에 최소값이 저장되는 최소 힙.
  - 루트에 최대값이 저장되는 최대 힙.

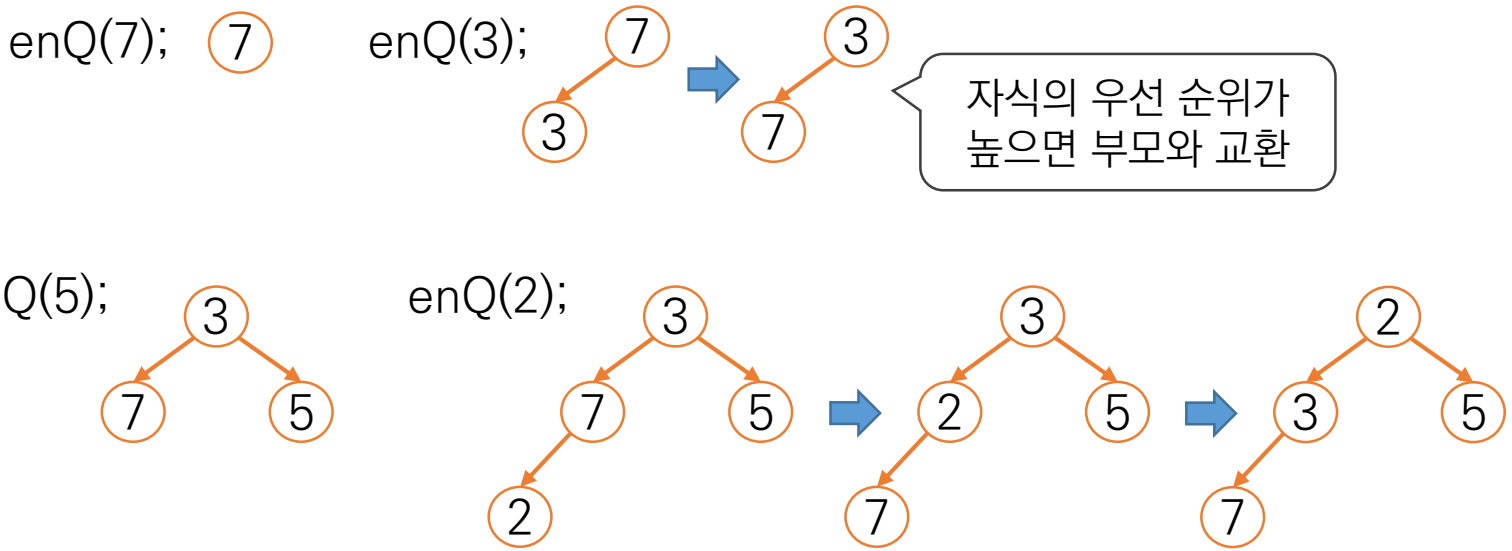


■ 최소 힙 생성

- 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}

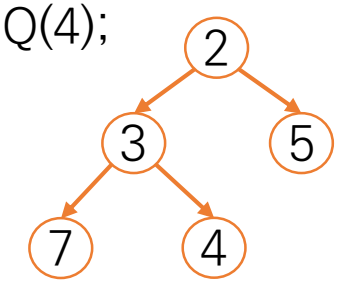


■ 최소 힙 생성(계속)

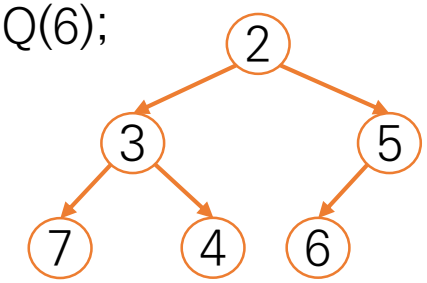
조건 : 완전 이진 트리 유지. 부모 < 자식

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

enQ(4);



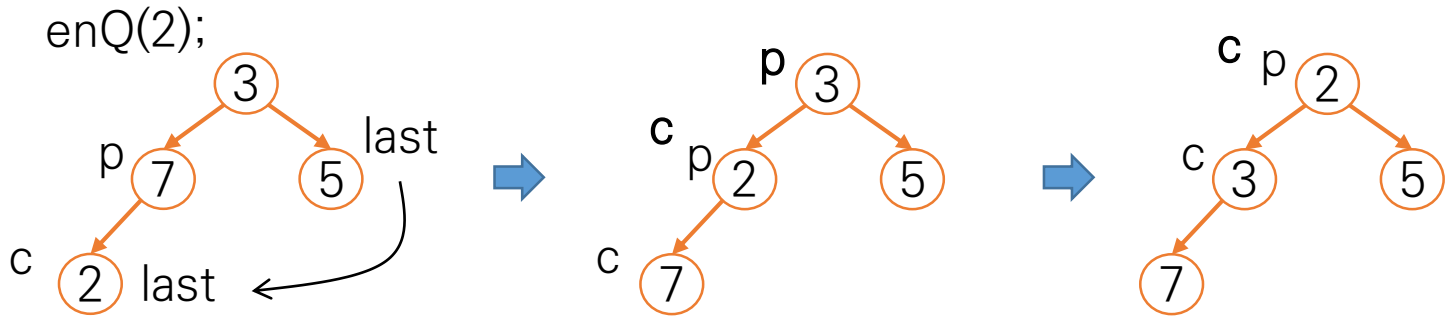
enQ(6);



■ 최소 힙 우선순위 큐의 연산

- enQ( )

조건 : 완전 이진 트리 유지. 부모 〈 자식

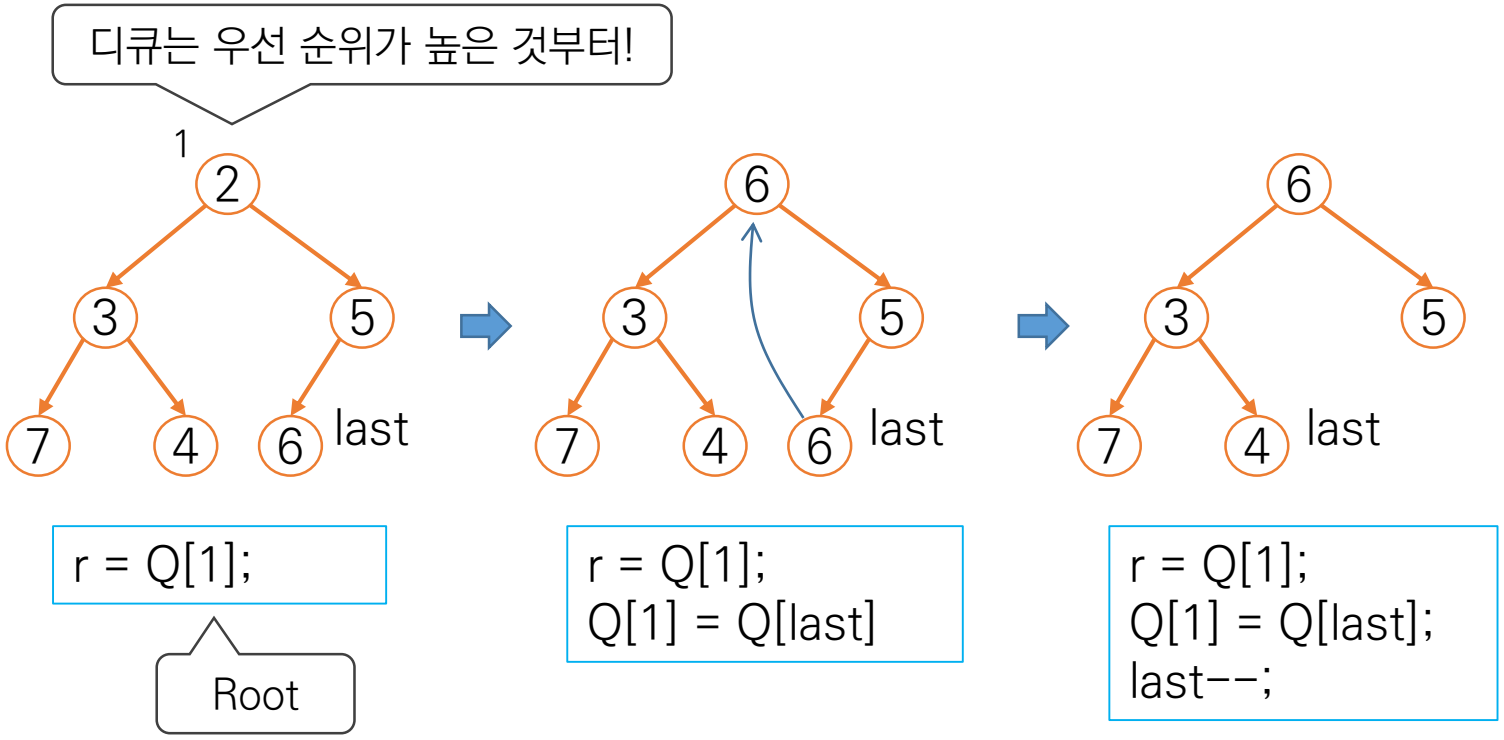



```
c = ++last;
p = c / 2;
Q[c] = data;
```

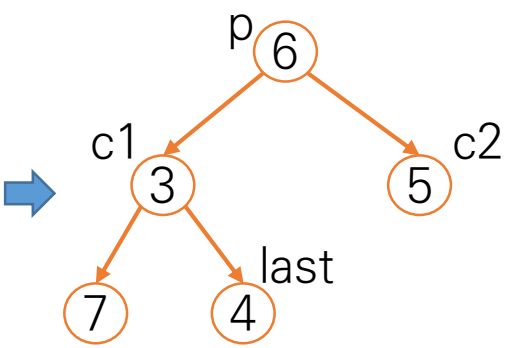
```
while( Q[p] 〉 Q[c] && c 〉 1)
{
      swap(Q[p], Q[c]);
      c= p;
      p = p/2;
}
```

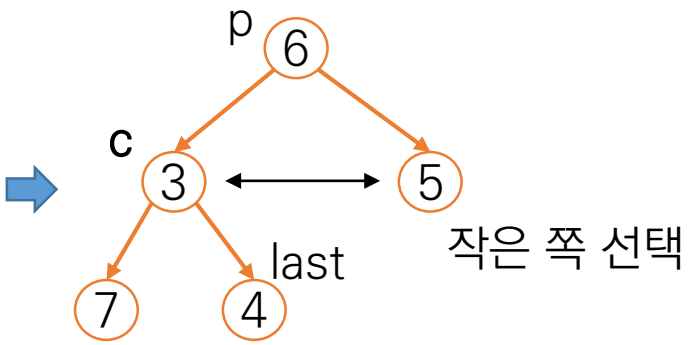
- deQ( )

조건 : 완전 이진 트리 유지. 부모 < 자식



- deQ( )계속






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

자식 노드 번호 계산

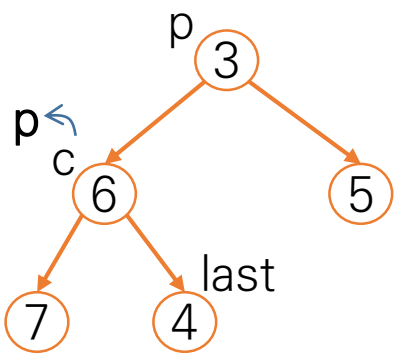
```
      if(c2<=last)
      {
            c = Q[c1]<Q[c2]?c1:c2;
            if( Q[c] < Q[p] )
                  swap(Q[p], Q[c]);
```

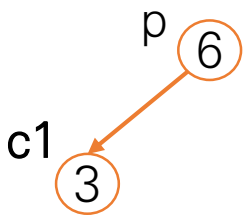
양쪽 자식이 있는 경우

작은 쪽과 자리바꿈

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;
```

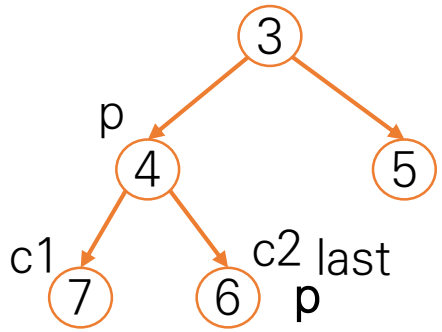
바꾼 쪽을 새로운 부모로

```
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

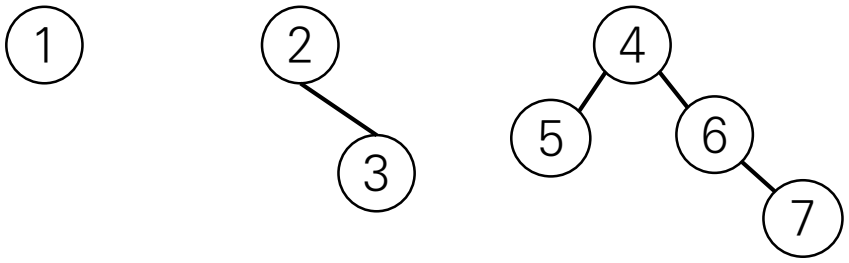
- N개의 정수가 입력된다. 앞에서 설명한 방식대로 최소힙을 구현해 저장하고, 마지막 노드의 조상 노드가 갖고 있는 숫자들의 합을 출력하라. N과 N개의 정수가 차례대로 주어진다.

```
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.
- 서로 소 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
      - p[rep(3)] = rep(2) // 3의 대표원소를 2의 대표원소로 대체

노드	1	2	3	4	5	6	7
대표	1	2	3->2	4	5	6	7

- 4 5, 4 6

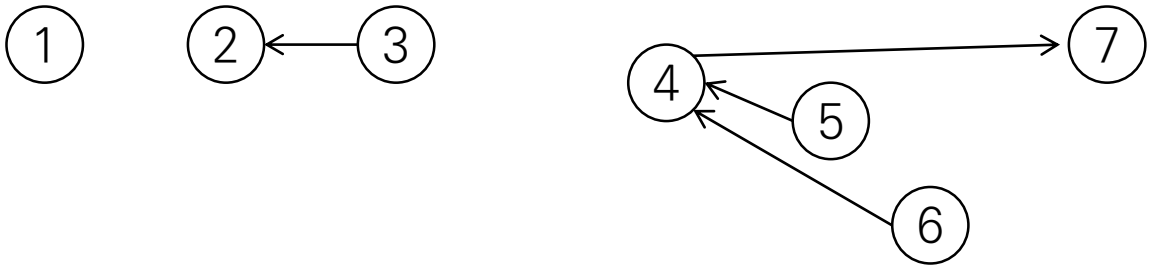
p[rep(5)] = rep(4), p[rep(6)] = rep(4)

노드	1	2	3	4	5	6	7
대표	1	2	2	4	5->4	6->4	7

- 7 6

p[rep(6)] = rep(7)

노드	1	2	3	4	5	6	7
대표	1	2	2	4->7	4	4	7



- 5와 7이 같은 트리에 속해 있는지 확인 하는 방법.
  - 5의 대표값과 7의 대표값을 비교.
  - 5의 대표값.

노드	1	2	3	4	5	6	7
대표	1	2	2	7	4	4	7

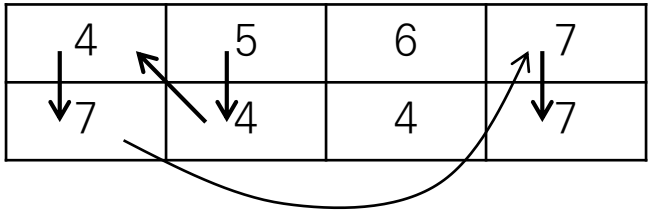
  - 7의 대표값.

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	7

  - 대표값이 같으므로 같은 트리에 속함.

- 대표값 찾기

```
rep( n )
    while( p[n] != n )
        n = p[n]
    return n
```



- 트리의 수
  - 인덱스==대표 원소인 개수를 확인

노드	1	2	3	4	5	6	7
대표	1	2	2	7	4	4	7

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
```