트리 1

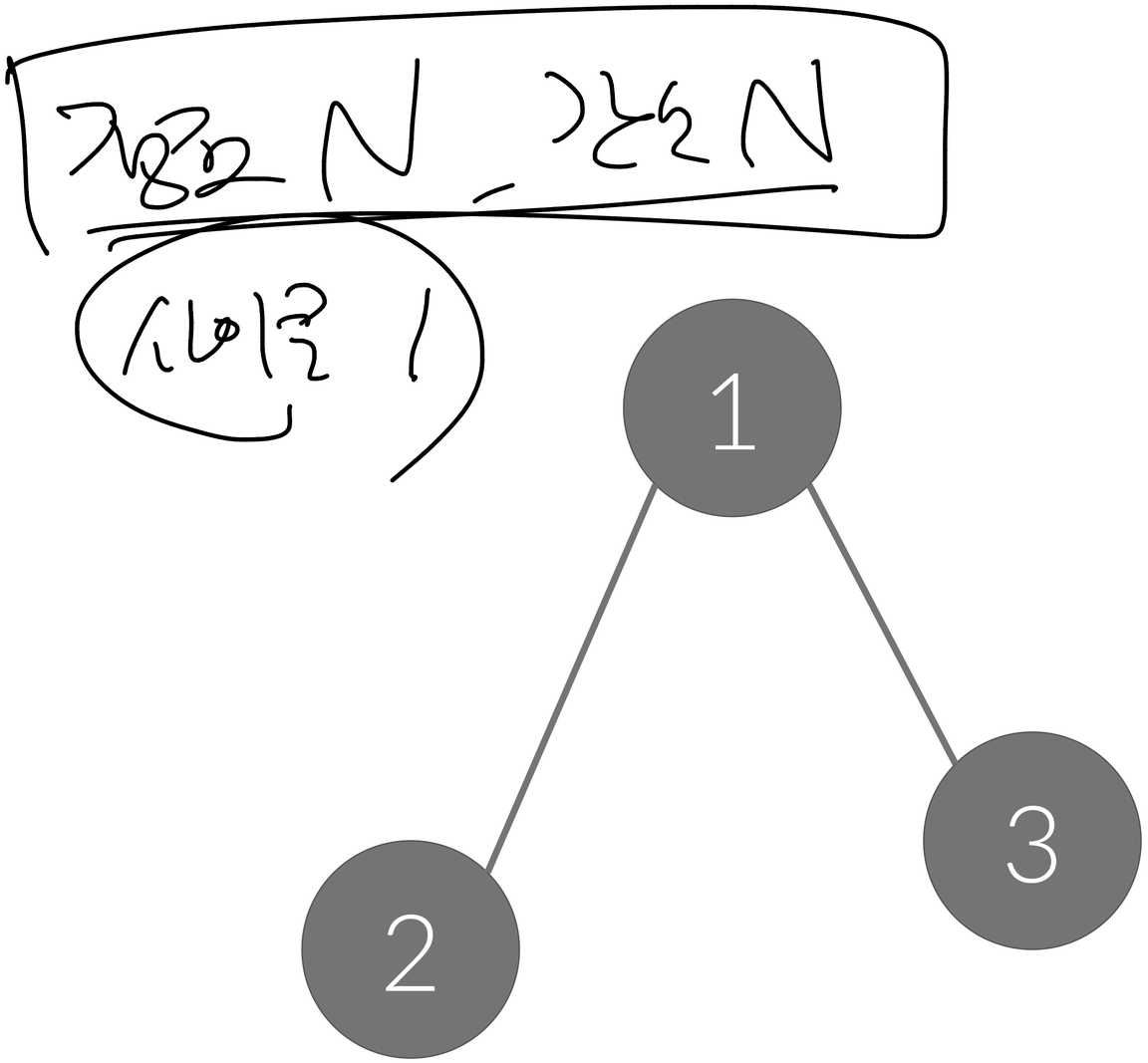
최백준 choi@startlink.io

트리

트리

Tree

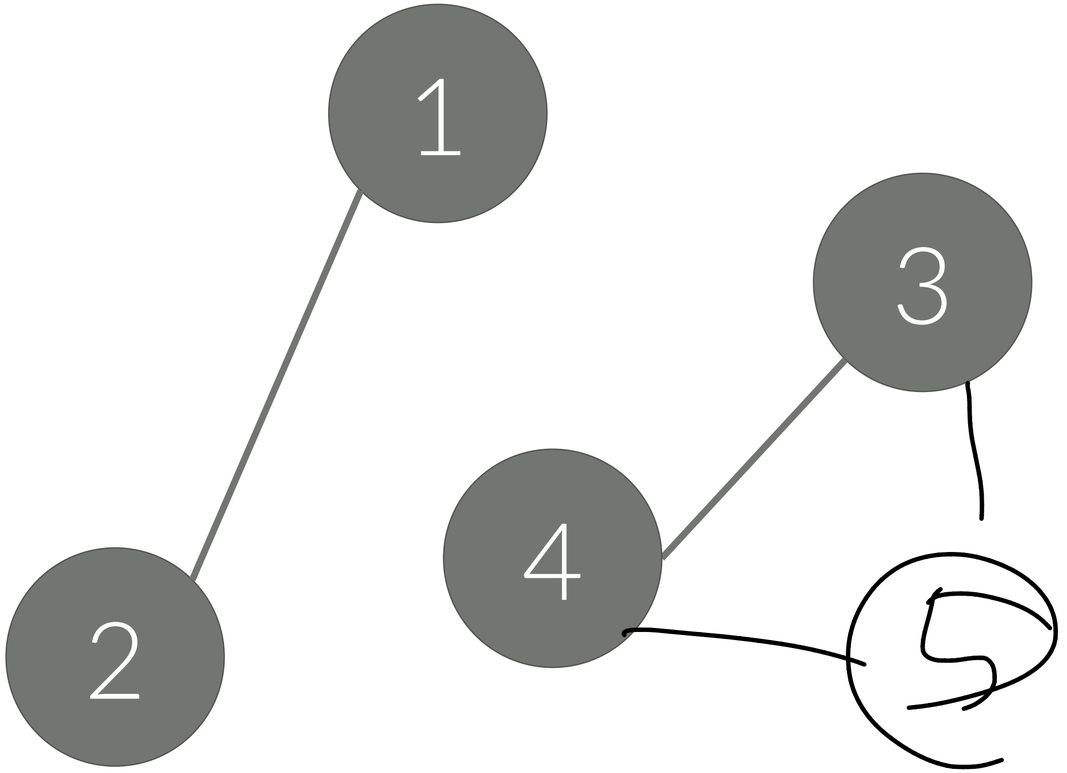
- 자료구조의 일종
- 사이클이 없는 연결 그래프
- 정점의 개수: V
- 간선의 개수: V-1



트리

Tree

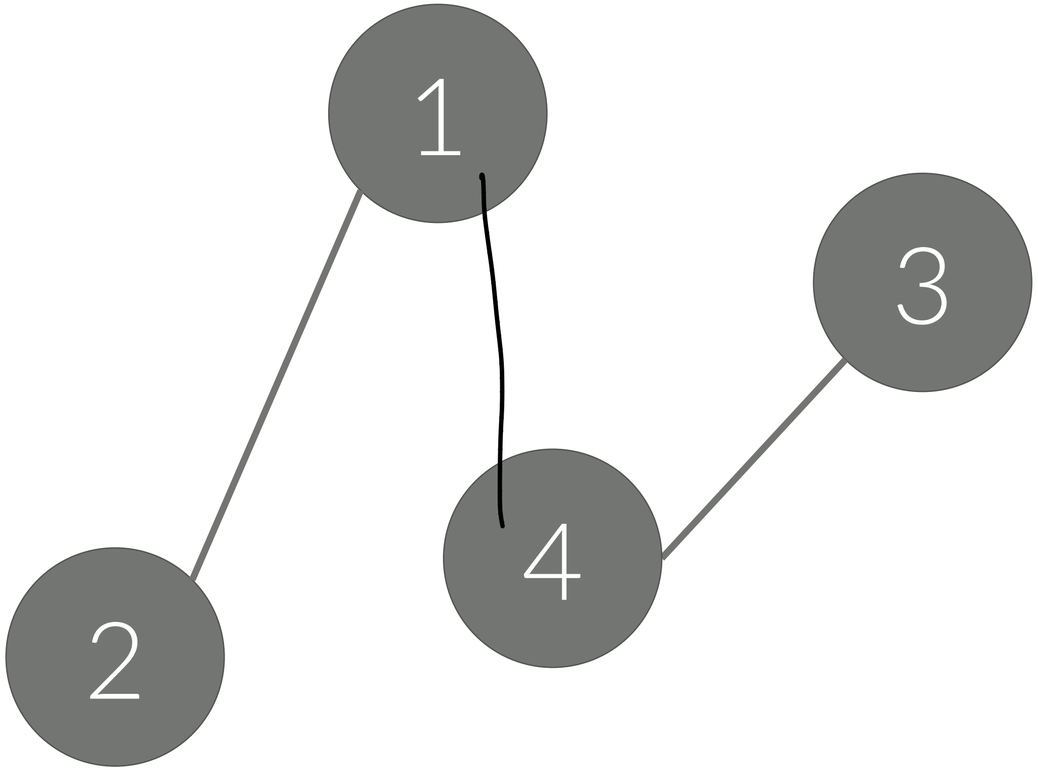
- 정점의 개수: V
- 간선의 개수: V-1
- 이면 트리일까? 아니다



트리

Tree

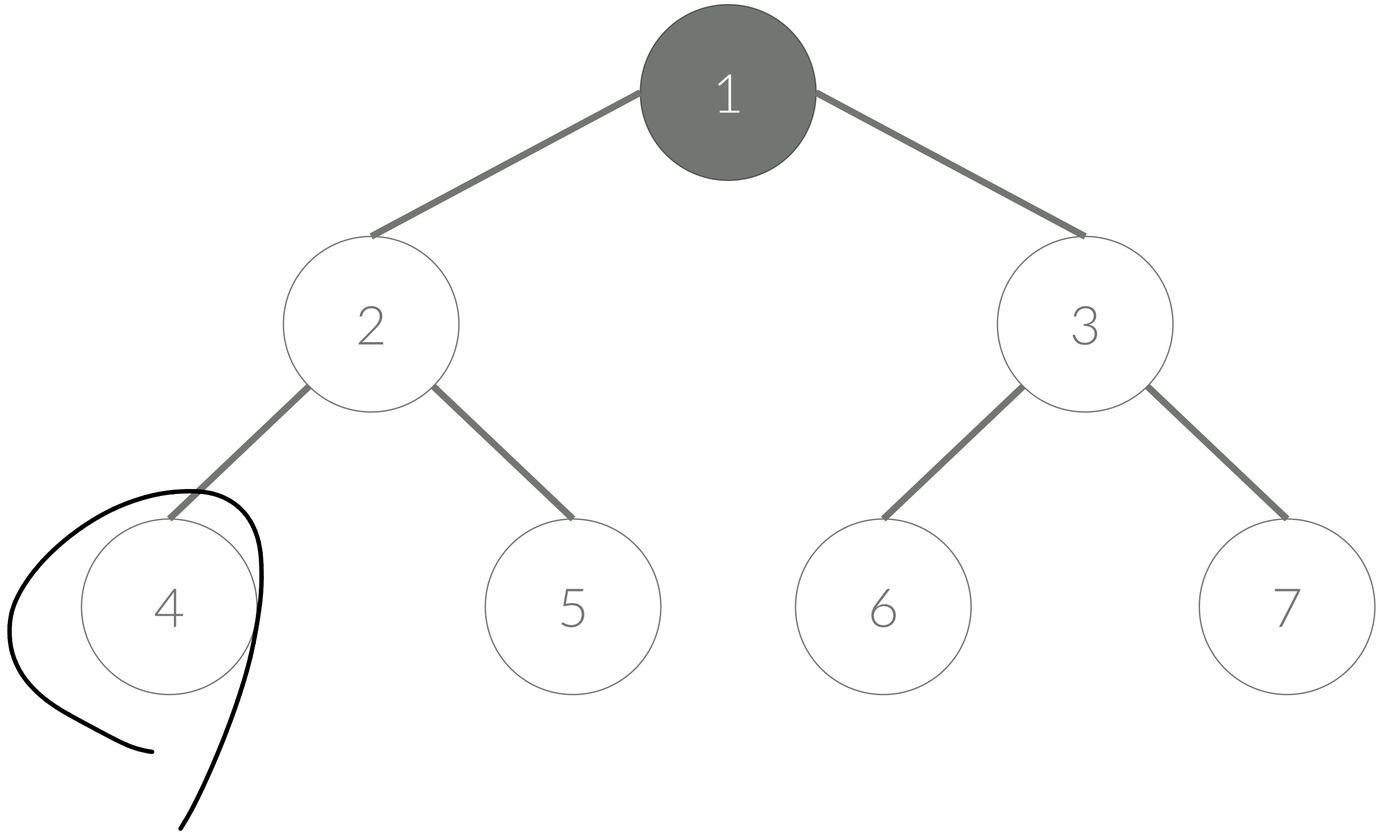
- 정점의 개수: V
- 간선의 개수: V-1
- 모든 정점이 연결되어 있음
- 이면 트리일까? 맞다



루트 있는 트리

Rooted Tree

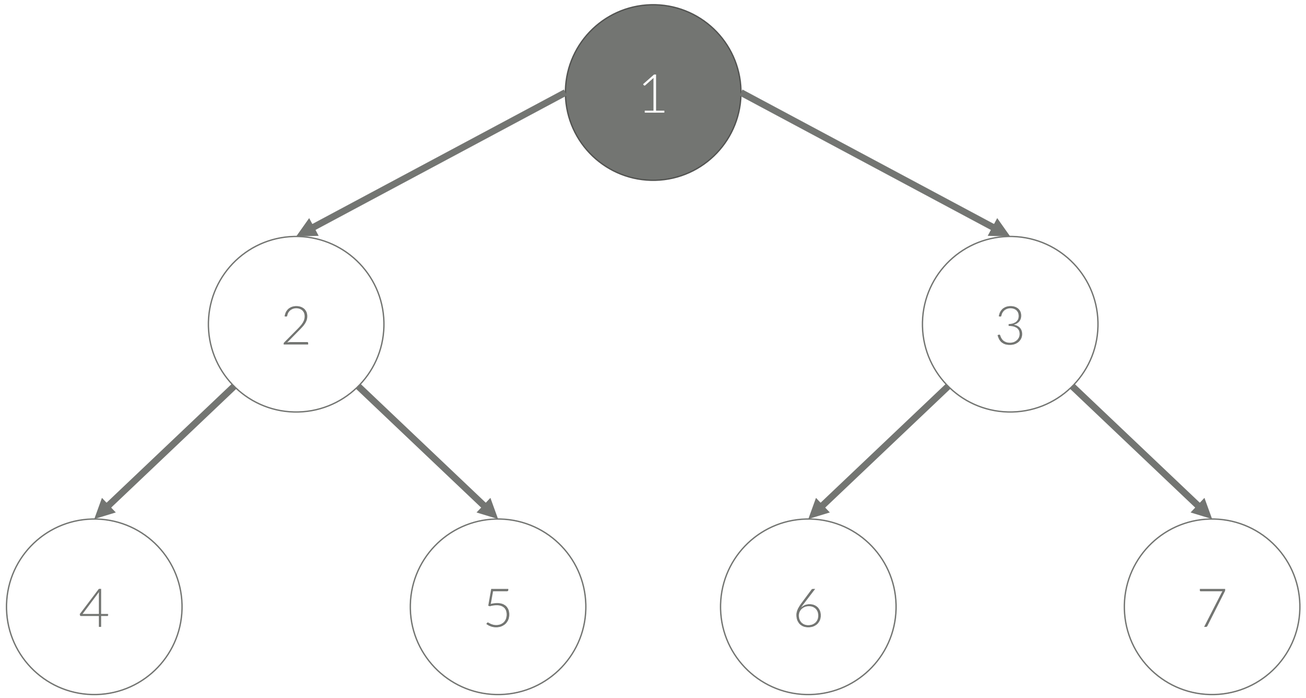
- 루트가 있는 트리
- 1번이 루트이다



루트 있는 트리

Rooted Tree

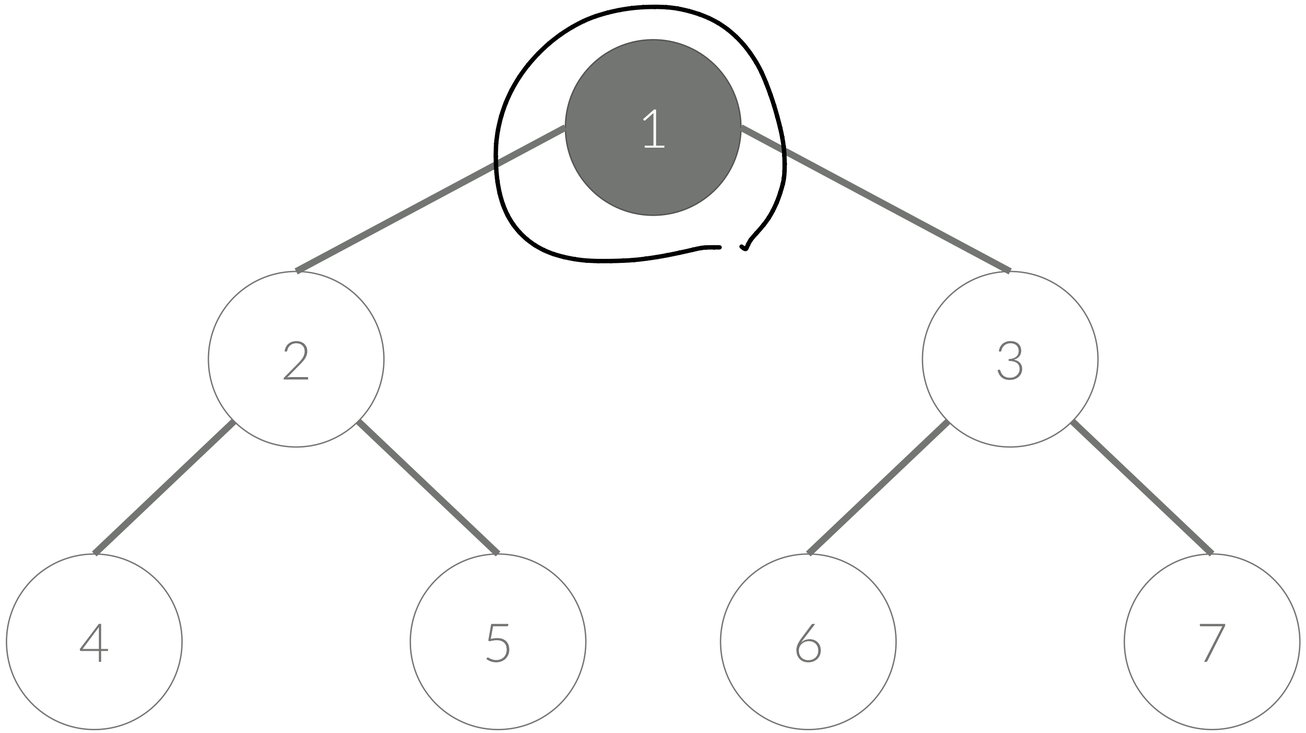
- 루트부터 아래로 방향을 정할 수 있다



부모

Parent

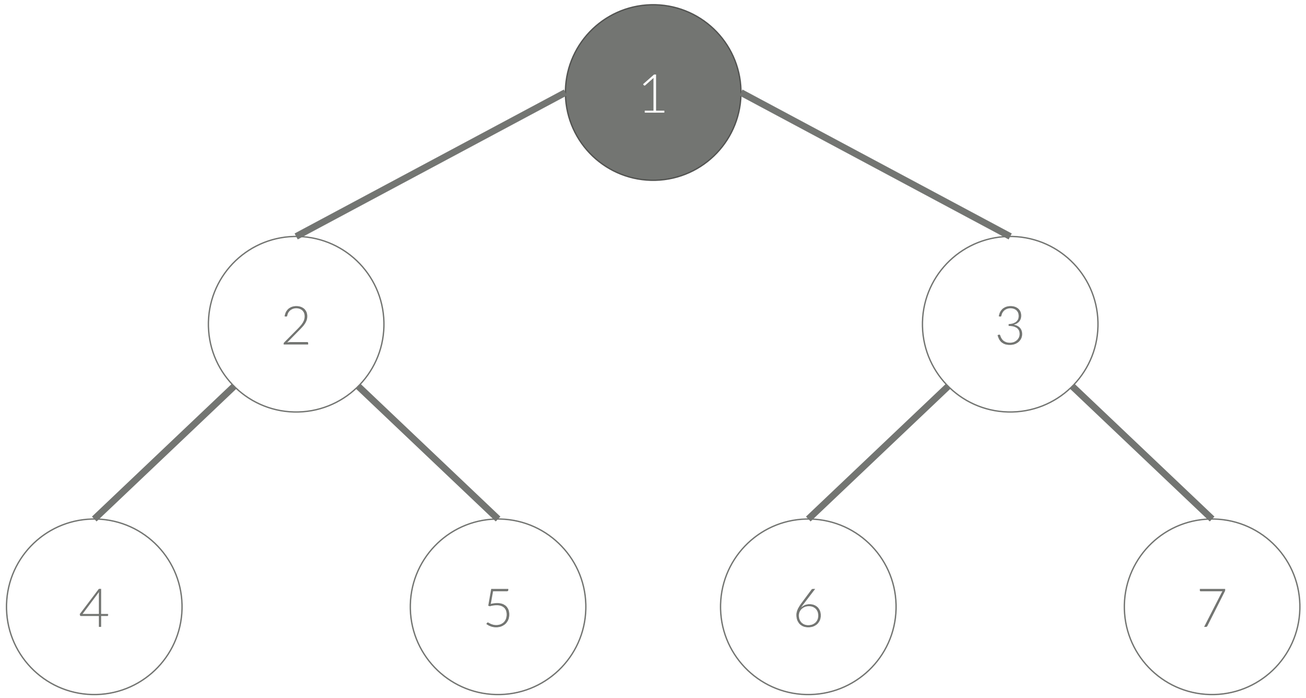
- 1은 2의 부모
- 2는 4의 부모



자식

Children

- 2는 1의 자식
- 4는 2의 자식
- 3의 자식: 6, 7

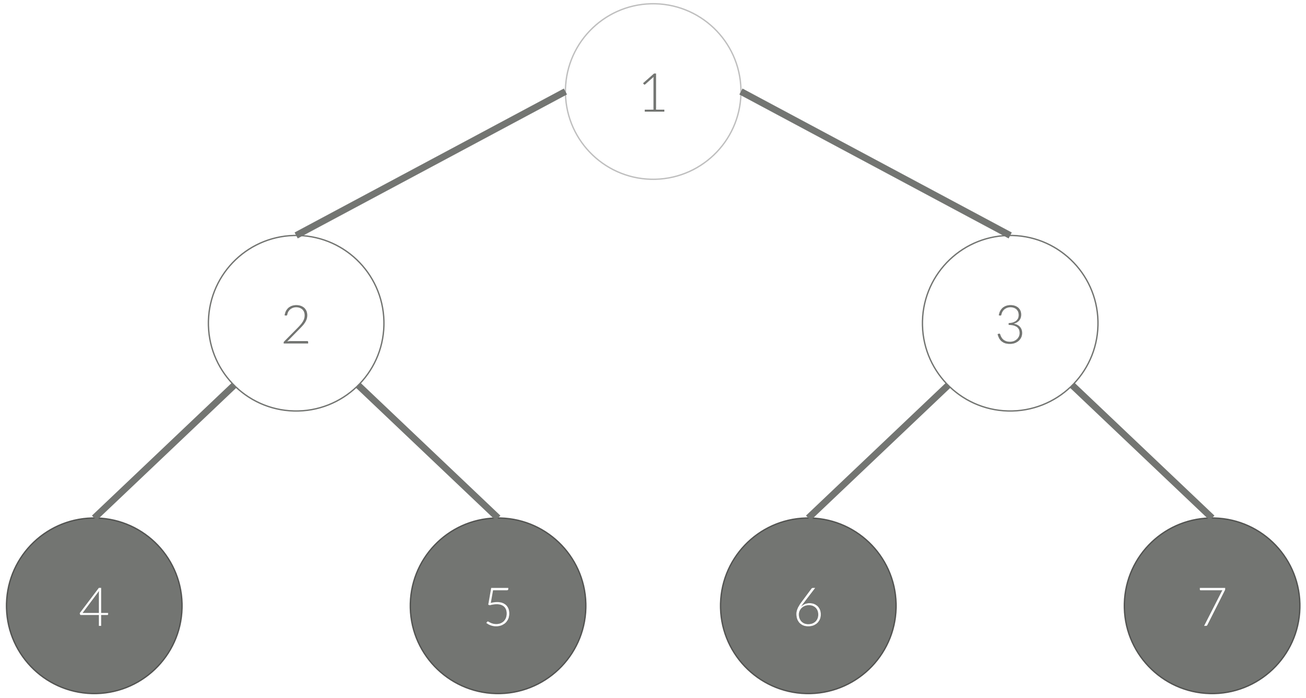


단말 정점

Terminal Node

Leaf Node

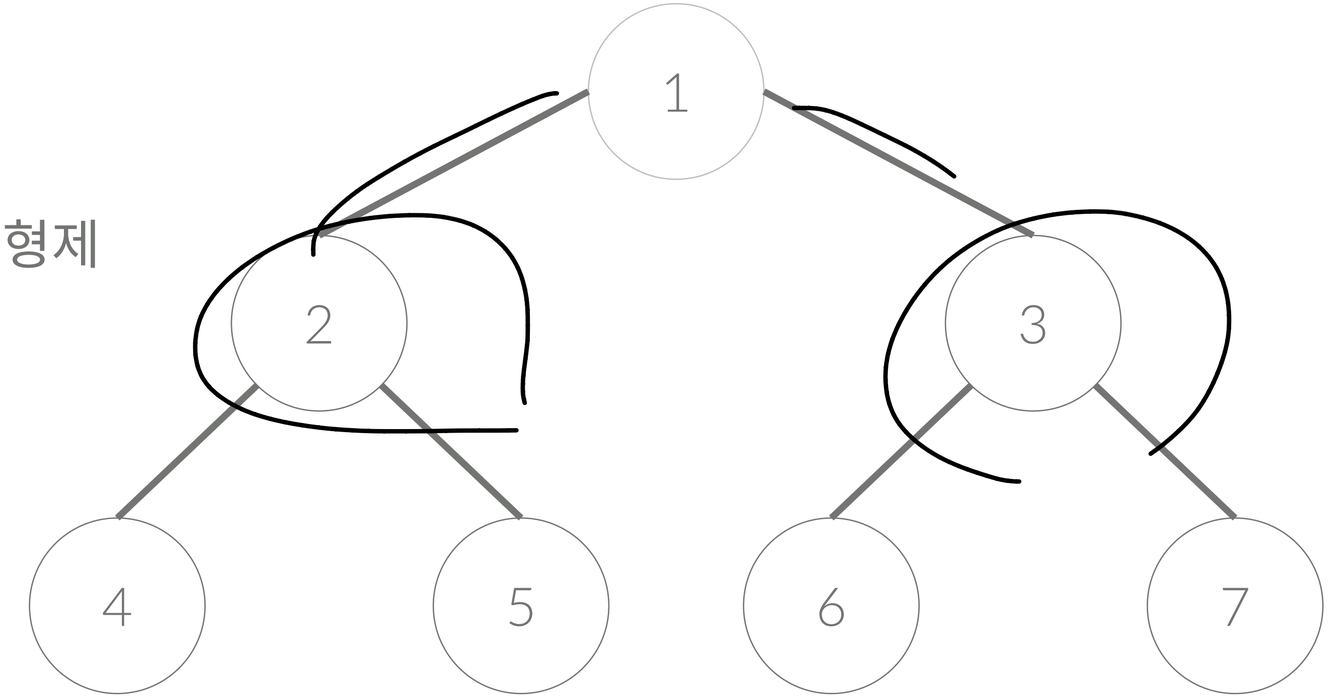
- 4, 5, 6, 7



형제

Sibling

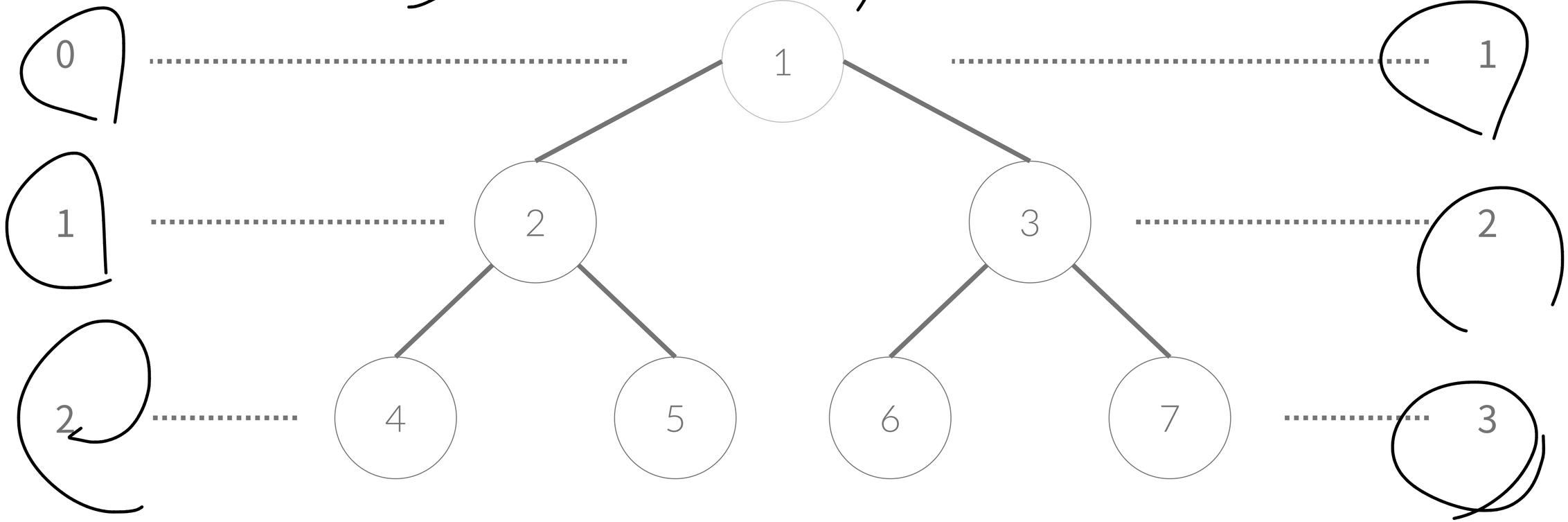
- 4와 5는 형제
- 6과 7은 형제
- 2와 3도 형제
- 같은 부모를 가지면 형제



깊이

Depth

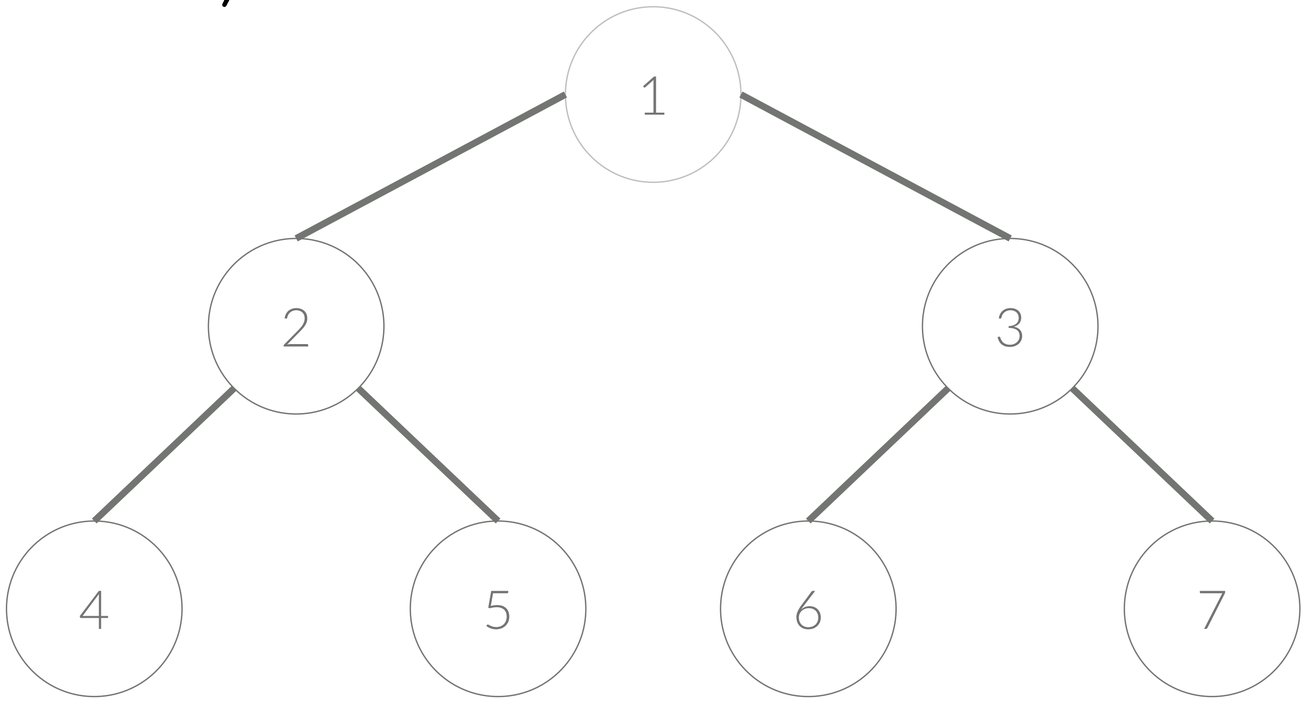
- 루트에서 부터 거리 (루트의 깊이를 0으로 하는 경우와 1로 하는 경우가 있다)



높이

Height

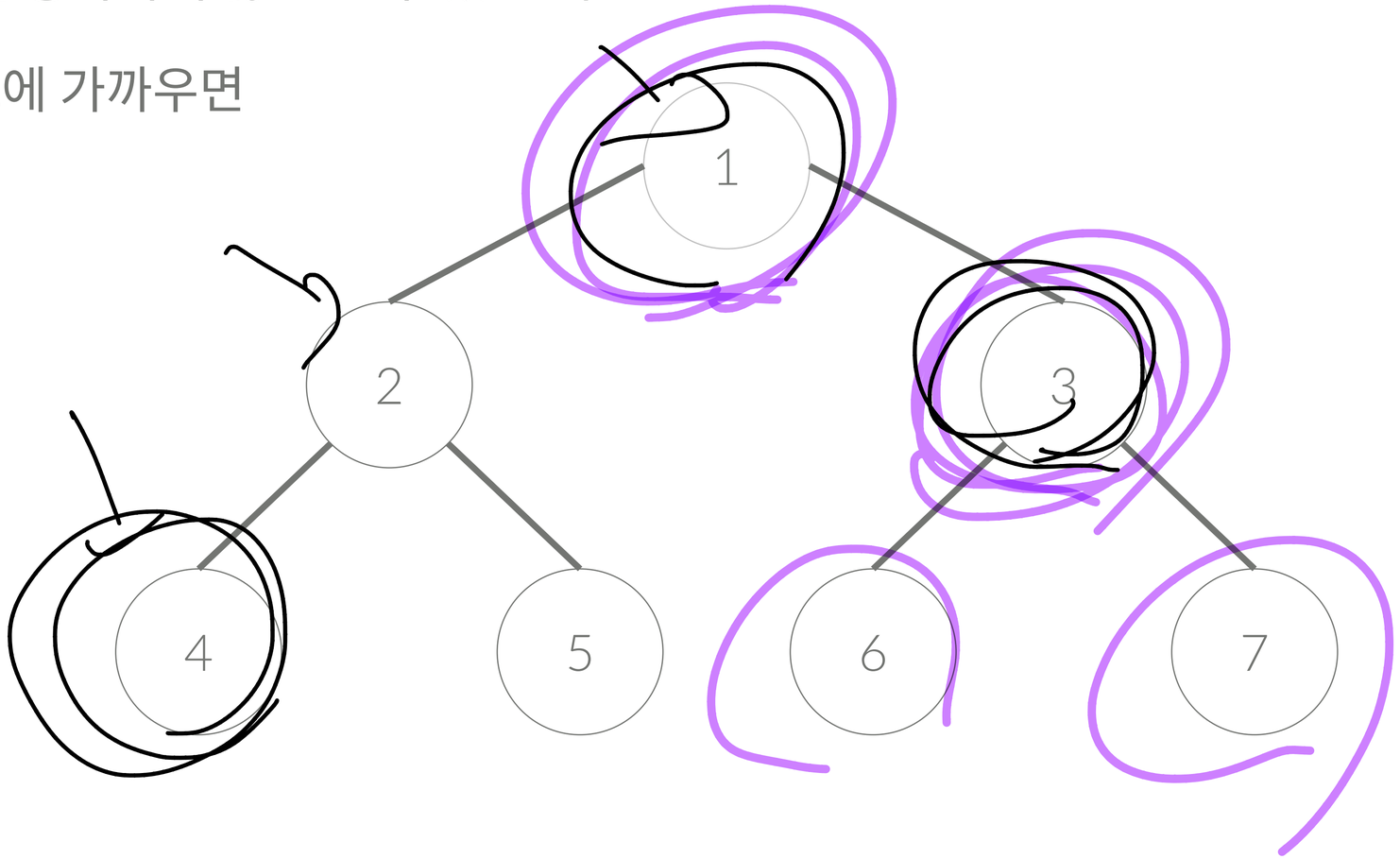
- 깊이 중 가장 큰 값 2 또는 3



조상, 자손

Ancestor, Descendent

- p -> q로 루트를 통과하지 않고 갈 수 있을 때
- p가 q보다 루트에 가까우면
- p는 q의 조상
- q는 p의 자손

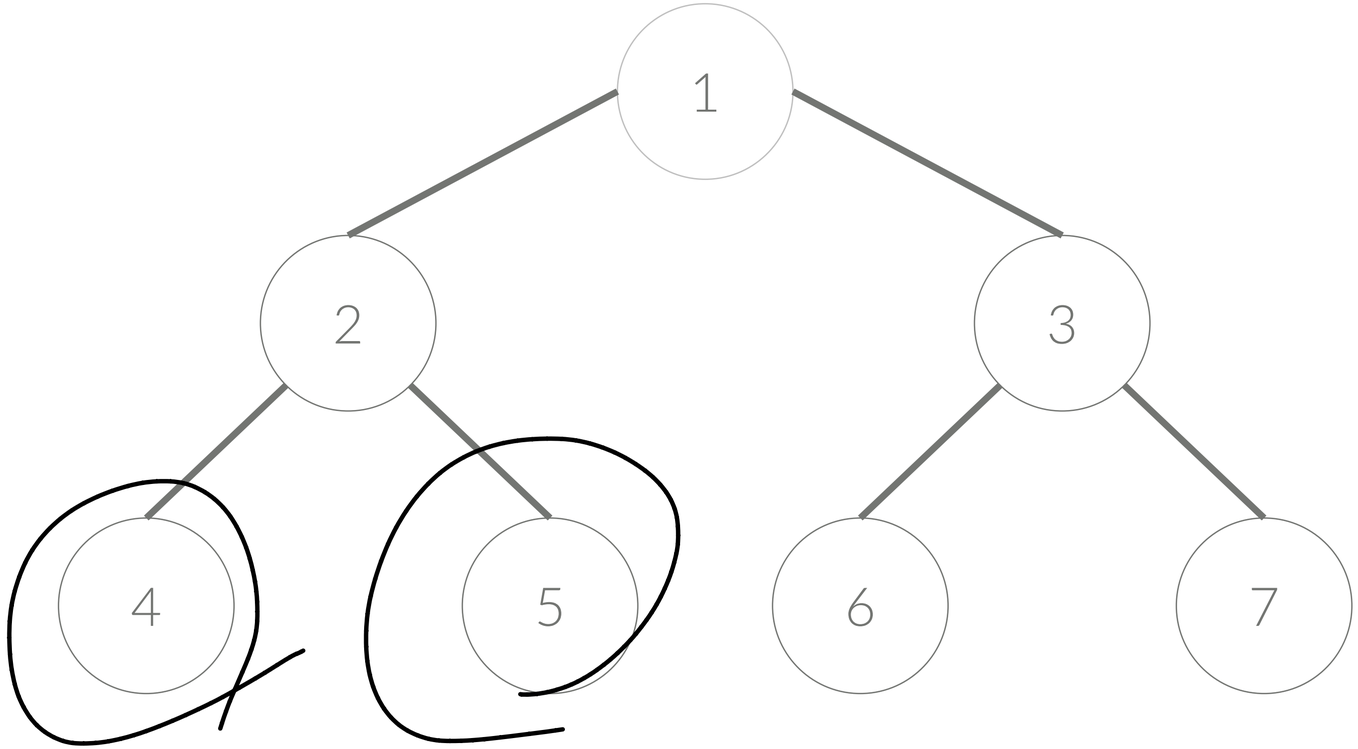


이진 트리

이진 트리

Binary Tree

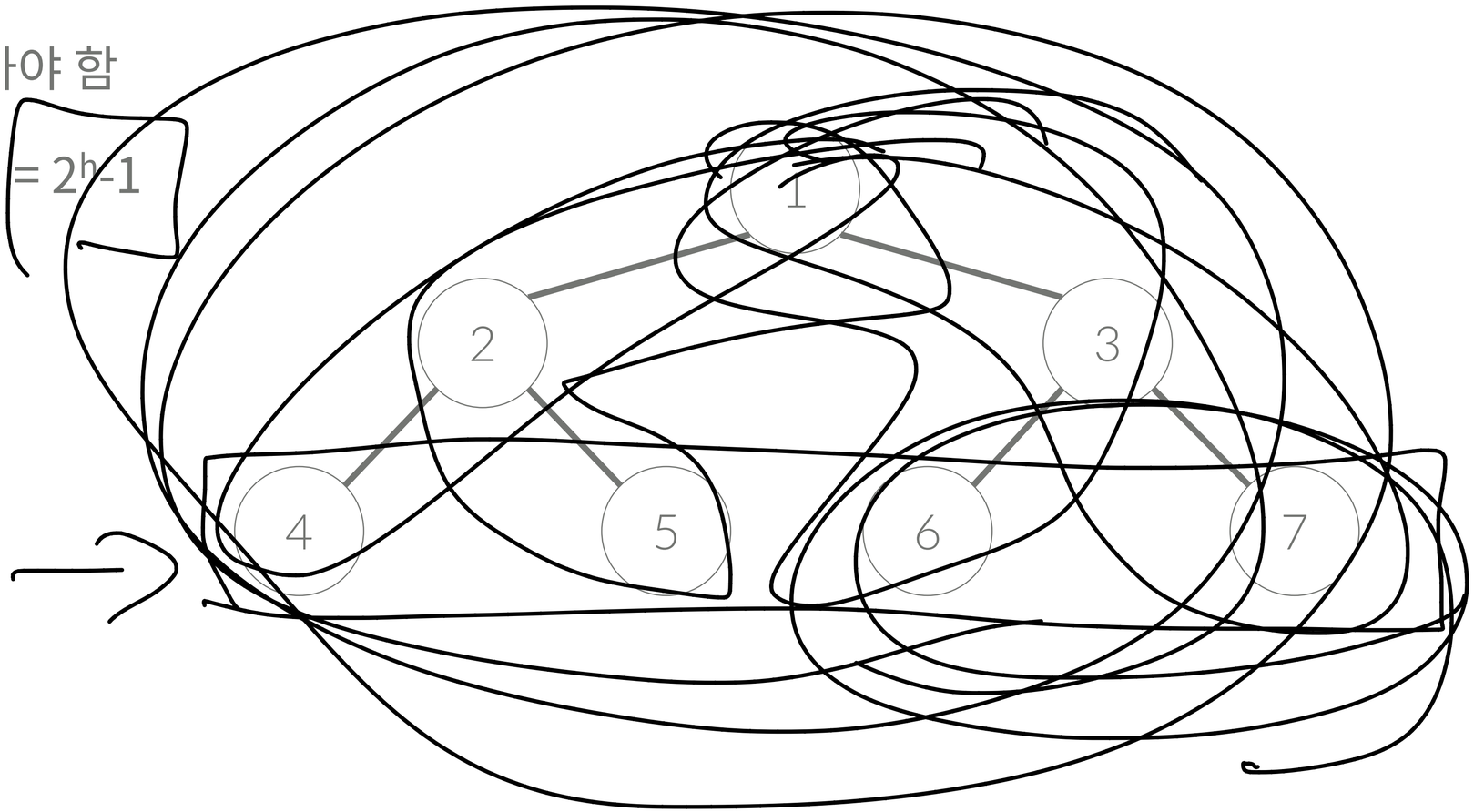
- 자식을 최대 2개만 가지고 있는 트리



포화 이진 트리

Perfect Binary Tree

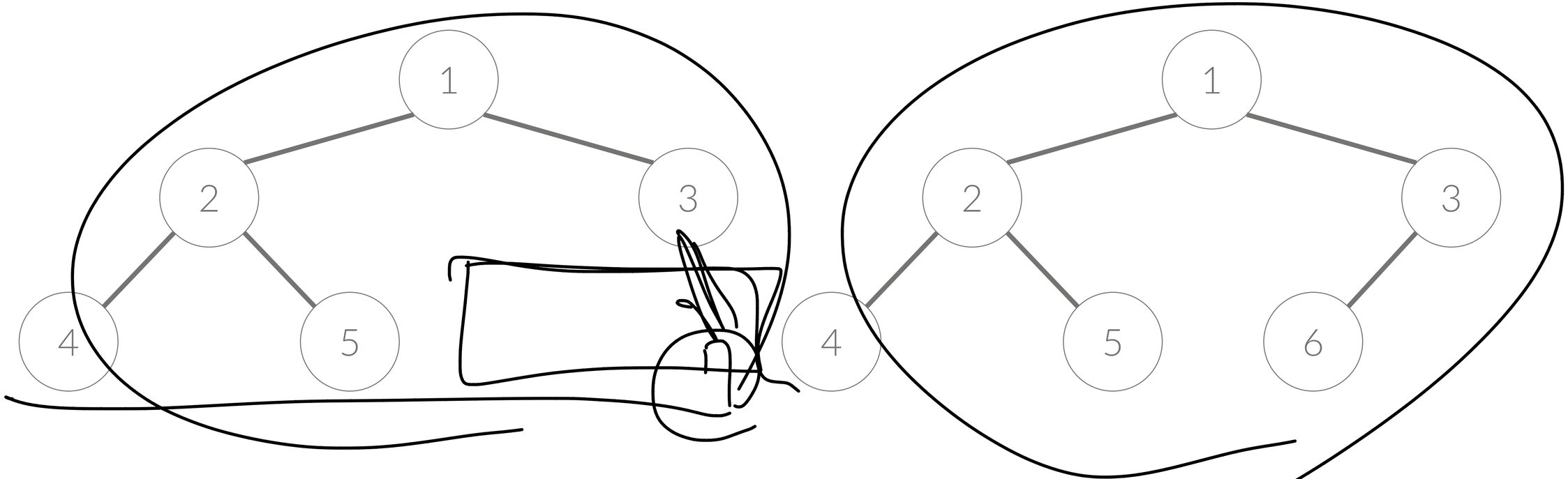
- 리프 노드를 제외한 노드의 자식의 수: 2
- 리프 노드의 자식의 수: 0
- 모든 리프 노드의 깊이가 같아야 함
- 높이가 h인 트리의 노드 개수 = 2^h-1



완전 이진 트리

Complete Binary Tree

- 리프 노드를 제외한 노드의 자식의 수: 2
- 리프 노드의 자식의 수: 0
- 마지막 레벨에는 노드가 일부는 없을 수도 있음
- 오른쪽에서부터 몇 개가 사라진 형태



트리의 표현

트리의 표현

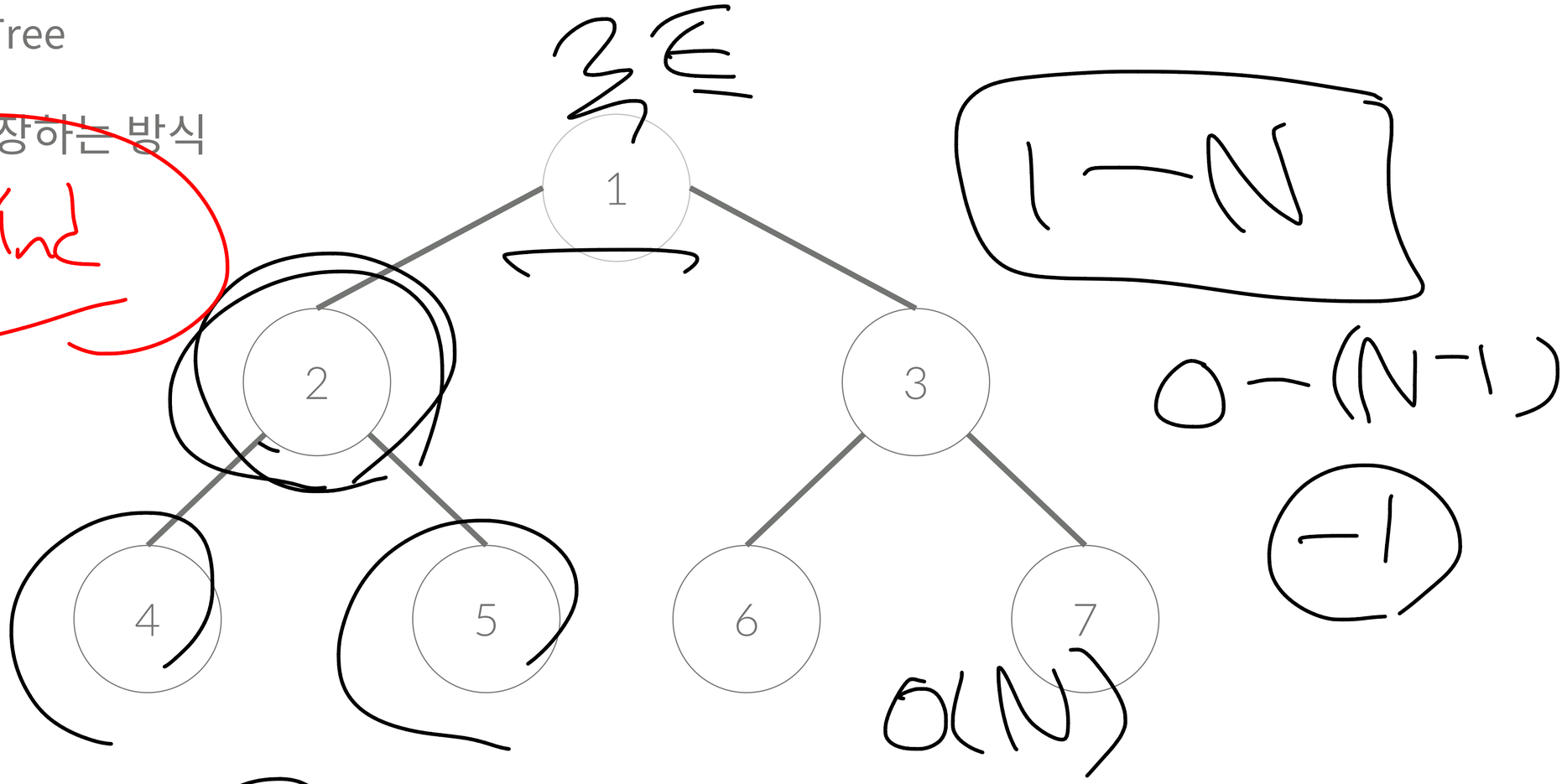
Representation of Tree

- 트리는 그래프이기 때문에, 그래프의 표현과 같은 방식으로 저장할 수 있다.
- 또는
- 트리의 모든 노드는 부모를 하나 또는 0개만 가지기 때문에 부모만 저장하는 방식으로 저장할 수 있다
- 부모가 0개인 경우는 트리의 루트인데, 이 경우 부모를 -1이나 0으로 처리하는 방식을 사용한다

트리의 표현

Representation of Tree

- 트리의 부모만 저장하는 방식

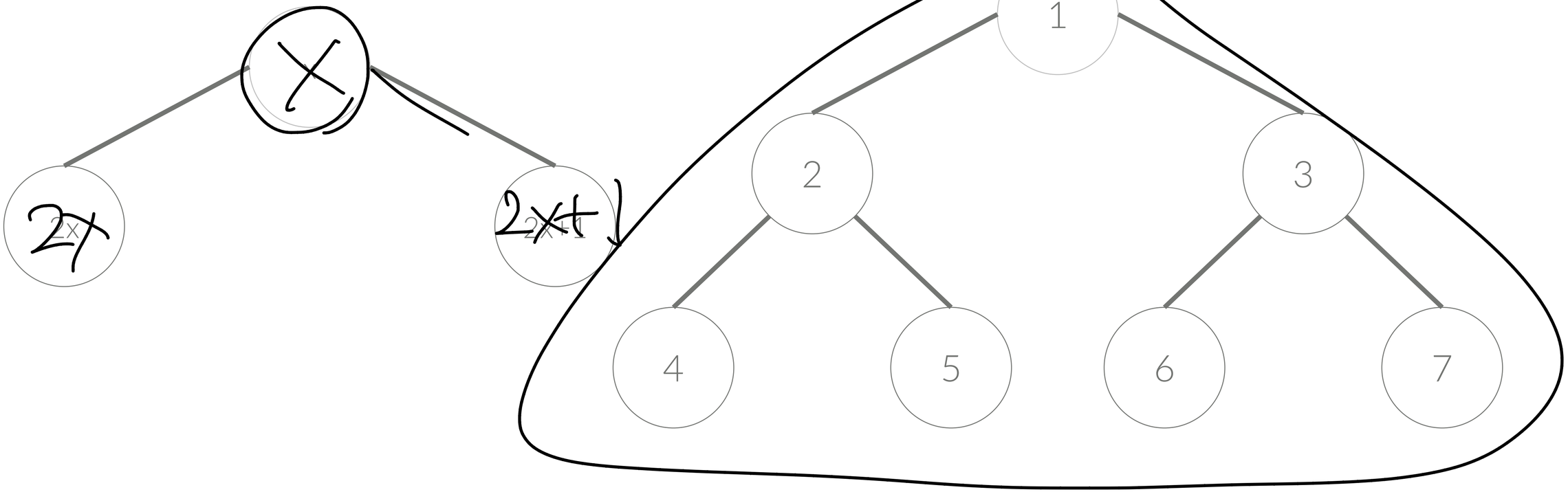


i	1	2	3	4	5	6	7
parent[i]	0	1	1	2	2	3	3

트리의 표현

Representation of Tree

- 완전 이진 트리의 경우에는 배열로 표현할 수 있다.
- 부모의 노드가 x인 경우에 자식의 노드는 2*x, 2*x+1 로 나타내면 된다.



트리의 표현

Representation of Tree

- 이진 트리의 경우에는 구조체나 클래스를 이용할 수도 있다

```
struct Node {
    Node *left;
    Node *right;
}
```

트리의 순회

트리의 순회

Tree Traversal

- 트리의 모든 노드를 방문하는 순서이다.
- 그래프의 경우에는 DFS와 BFS가 있었다
- 트리에서도 위의 두 방법을 사용할 수 있다.
- DFS는 아래와 같이 3가지 출력 순서가 있다
- 프리오더
- 인오더
- 포스트오더
- 세 방법의 차이는 노드 방문 처리를 언제 할 것인가이다.

트리의 순회

Tree Traversal

- 프리오더
  - 노드 방문
  - 왼쪽 자식 노드를 루트로 하는 서브 트리 프리오더
  - 오른쪽 자식 노드를 루트로 하는 서브 트리 프리오더
- 인오더
- 포스트오더

트리의 순회

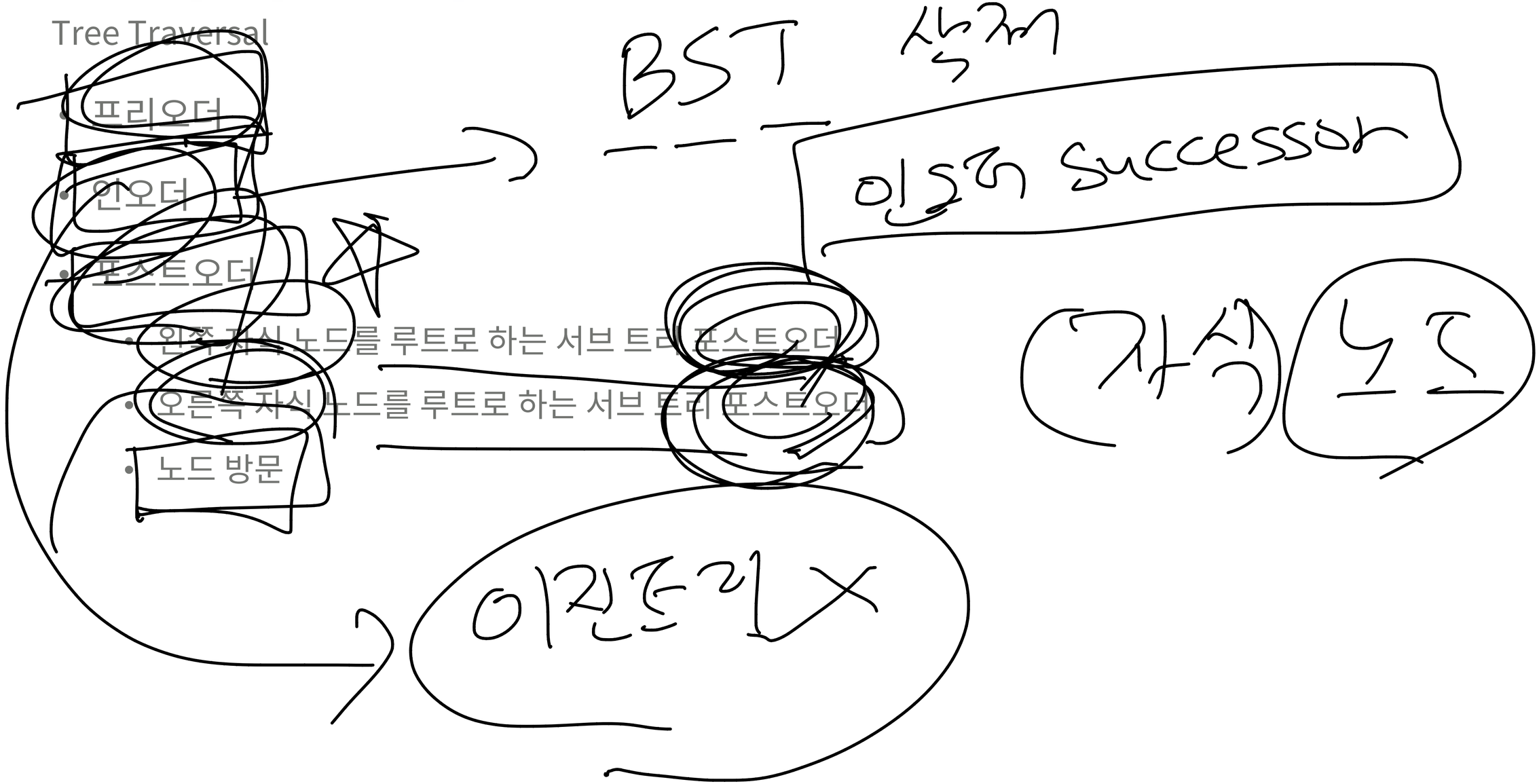
Tree Traversal

- 프리오더
- 인오더
  - 왼쪽 자식 노드를 루트로 하는 서브 트리 인오더
  - 노드 방문
  - 오른쪽 자식 노드를 루트로 하는 서브 트리 인오더
- 포스트오더

트리의 순회

Tree Traversal

- 프리오더
- 인오더
- 포스트오더
  - 왼쪽 자식 노드를 루트로 하는 서브 트리 포스트오더
  - 오른쪽 자식 노드를 루트로 하는 서브 트리 포스트오더
  - 노드 방문



트리 순회

https://www.acmicpc.net/problem/1991

- 이진 트리의 프리오더, 인오더, 포스트오더 순서를 출력하는 문제

트리 순회

https://www.acmicpc.net/problem/1991

- 소스: http://codeplus.codes/5579a90defb9426fb5f04745246cb20c

트리의 높이와 너비

https://www.acmicpc.net/problem/2250

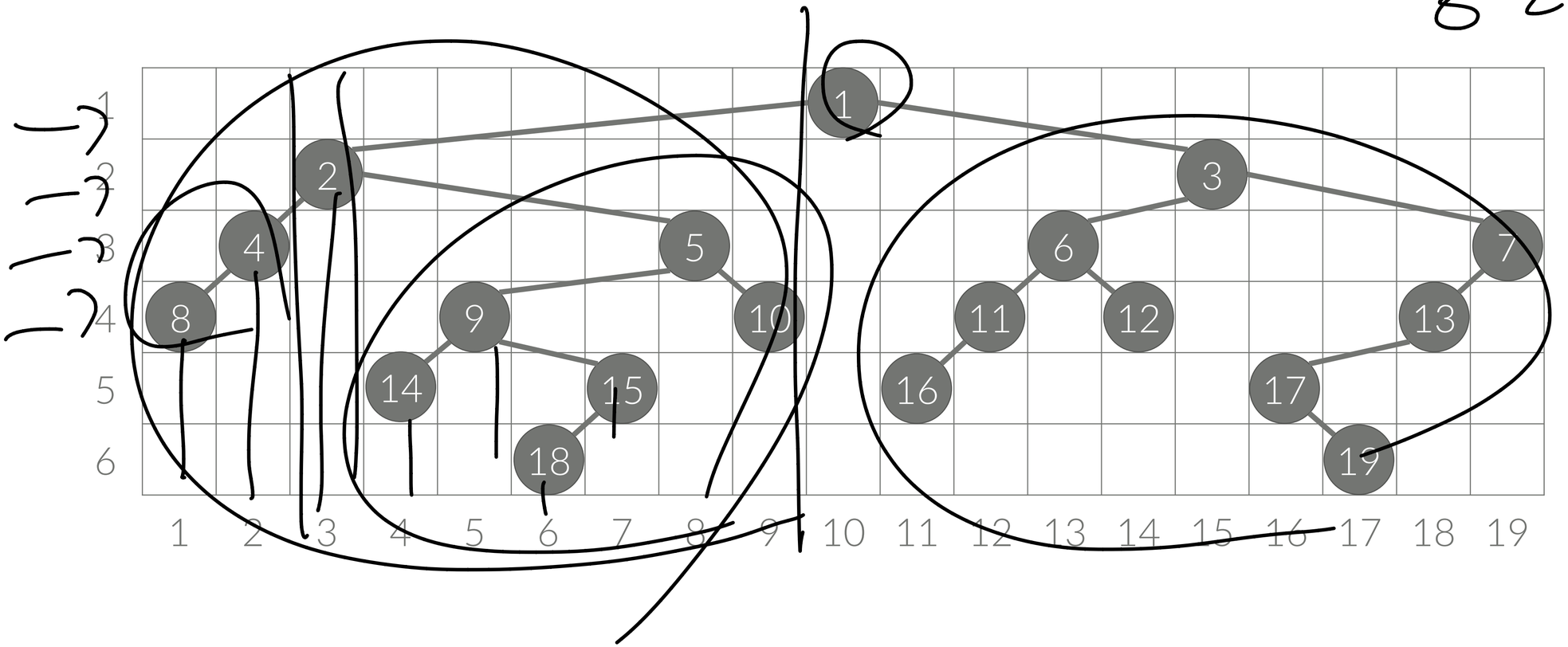
- 이진 트리를 다음과 같은 규칙에 따라 격자에 그리려고 한다

1. 이진트리에서 같은 레벨(level)에 있는 노드는 같은 행에 위치한다.
2. 한 열에는 한 노드만 존재한다.
3. 임의의 노드의 왼쪽 부트리(left subtree)에 있는 노드들은 해당 노드보다 왼쪽의 열에 위치하고, 오른쪽 부트리(right subtree)에 있는 노드들은 해당 노드보다 오른쪽의 열에 위치한다.
4. 노드가 배치된 가장 왼쪽 열과 오른쪽 열 사이엔 아무 노드도 없이 비어있는 열은 없다.

트리의 높이와 너비

https://www.acmicpc.net/problem/2250

- 이진 트리를 다음과 같은 규칙에 따라 격자에 그리려고 한다



트리의 높이와 너비

https://www.acmicpc.net/problem/2250

- 각 노드가 배치되는 순서는 인오더와 같다
- 인오더를 수행하면서, 몇 번인지, 몇 번째 레벨인지 기록한다

트리의 높이와 너비

https://www.acmicpc.net/problem/2250

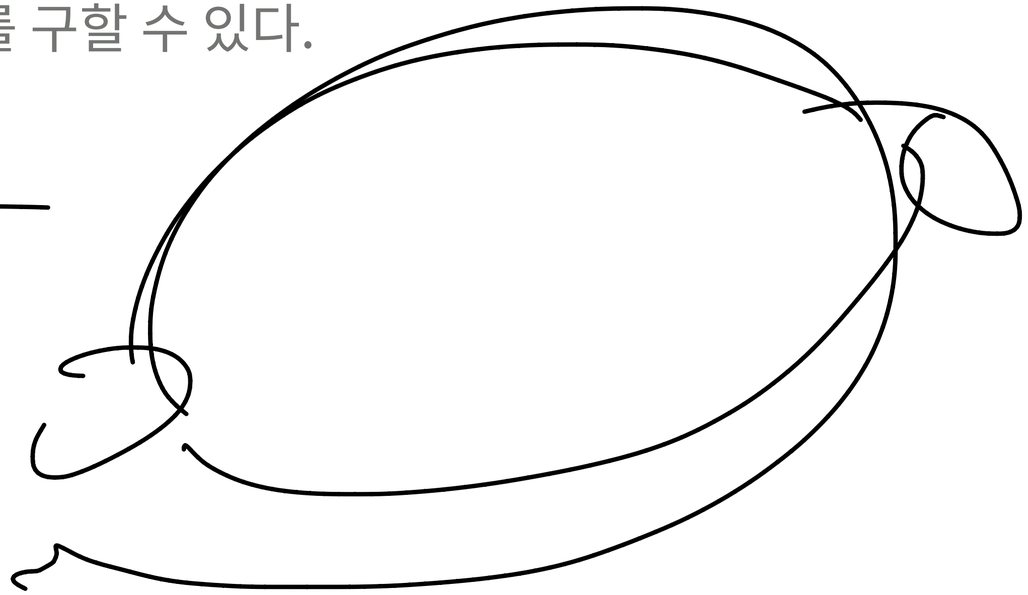
- 소스: http://codeplus.codes/0d1a8f4e8d1c453d9f0c0a923ae00124

트리의 탐색

트리의 탐색

BFS

- 트리의 탐색은 DFS/BFS 알고리즘을 이용해서 할 수 있다.
- 트리는 사이클이 없는 그래프이기 때문에
- 임의의 두 정점 사이의 경로는 1개이다.
- 따라서, BFS 알고리즘을 이용해서 최단 거리를 구할 수 있다.
- 이유: 경로가 1개라 찾은 그 경로가 최단 경로



트리의 부모 찾기

https://www.acmicpc.net/problem/11725

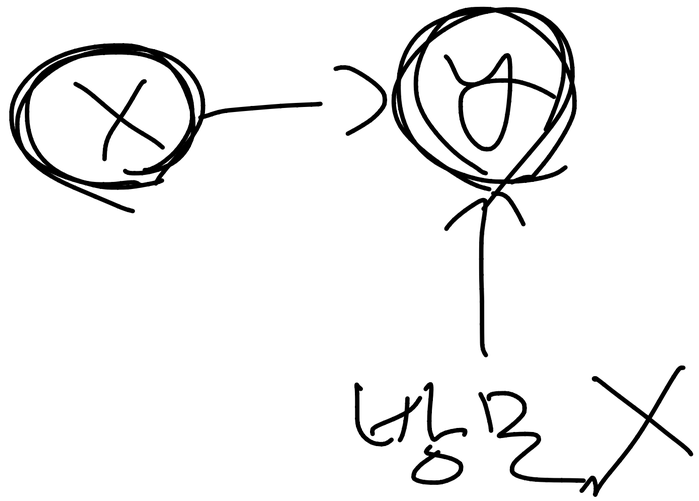
- 그래프로 트리를 입력받고
- 루트를 1이라고 정했을 때
- 각 노드의 부모를 찾는 문제

- BFS 탐색으로 해결할 수 있다.

트리의 부모 찾기

https://www.acmicpc.net/problem/11725

```
queue<int> q;
depth[1] = 0; check[1] = true; parent[1] = 0; q.push(1);
while (!q.empty()) {
    int x = q.front(); q.pop();
    for (int y : a[x]) {
        if (!check[y]) {
            depth[y] = depth[x] + 1;
            check[y] = true;
            parent[y] = x;
            q.push(y);
        }
    }
}
```



트리의 부모 찾기

https://www.acmicpc.net/problem/11725

- 소스: http://codeplus.codes/c447cc9e46dc4d22b2c499b7a5b69a2c

트리의 지름

트리의 지름

Diamater

- 트리에 존재하는 모든 경로 중에서 가장 긴 것의 길이를 트리의 지름이라고 한다
- 트리의 지름은 탐색 2번으로 구할 수 있다.

1. 한 정점 s에서 모든 정점까지의 거리를 구한다. 이 때, 가장 먼 거리인 정점을 u라고 한다.
2. u에서 모든 정점까지의 거리를 구한다. 이 때, 가장 먼 거리인 정점 v를 구한다.

- d(u, v)를 u와 v사이의 거리라고 했을 때, d(u, v)가 트리의 지름이다

트리의 지름

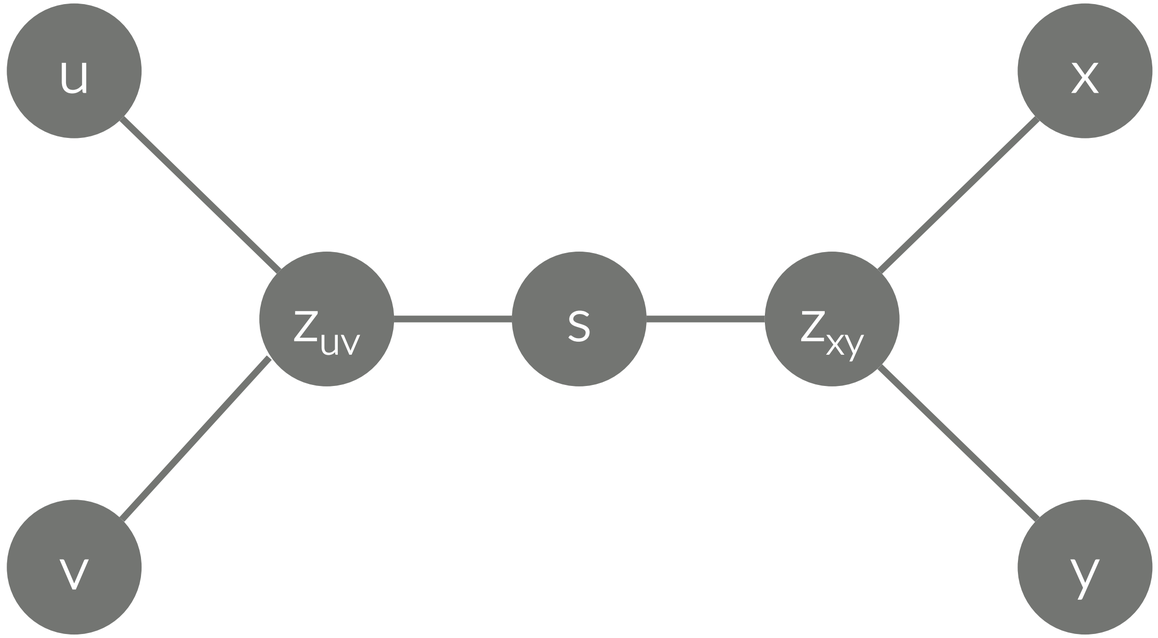
Diamater

- d(u, v)를 트리의 지름이라고 하자.
- 한 정점 s에서 BFS 탐색을 시작해서 가장 거리가 멀었던 정점을 x라고 하고
- x에서 BFS 탐색을 시작해서 가장 거리가 멀었던 정점을 y라고 하자.
- wlog d(s, u) ≥ d(s, v) 이다. 이때, d(s, x) ≥ d(s, y)는 반드시 만족해야 한다.
- 이제 d(x, y) = d(u, v)임을 증명하면 된다.

트리의 지름

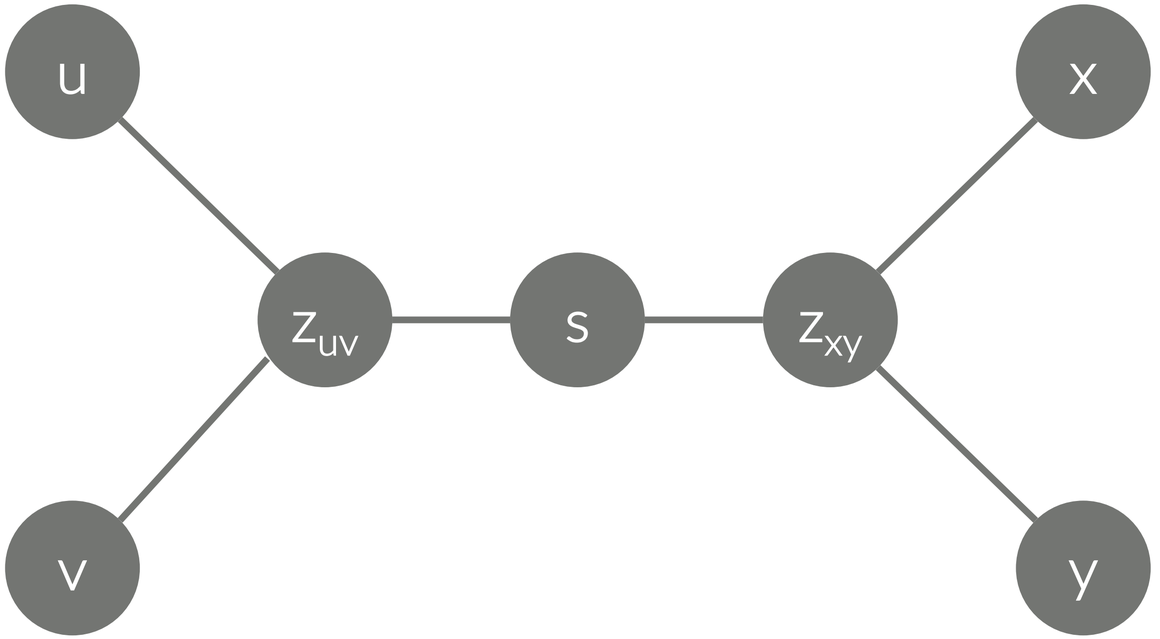
Diamater

- 그래프



트리의 지름

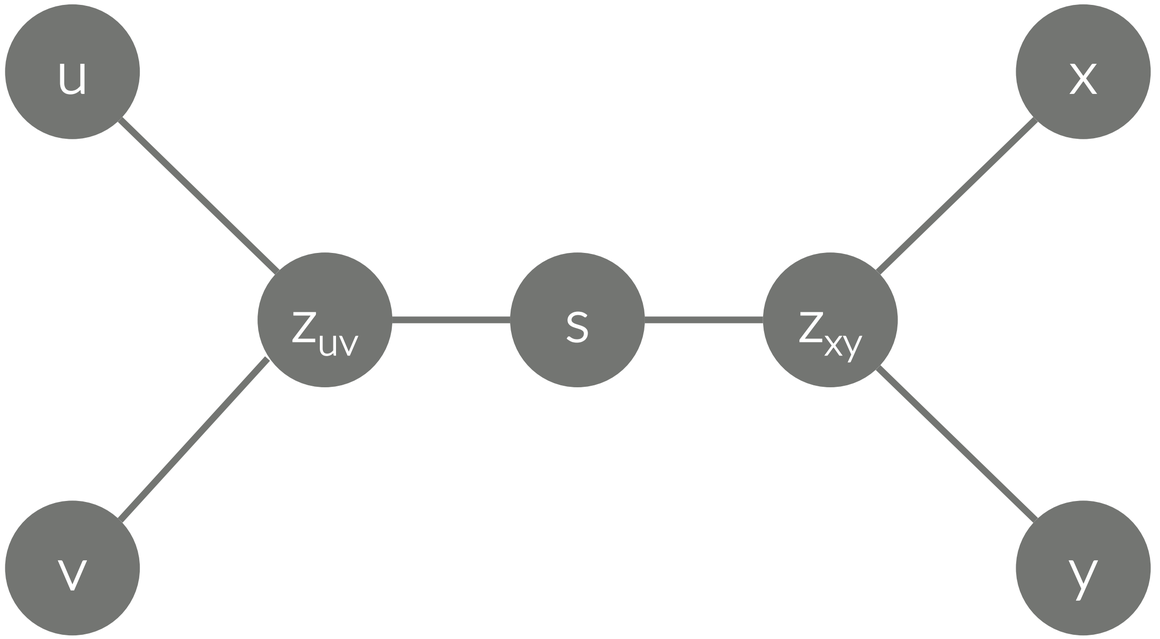
Diamater



1. $d(z_{uv}, y) \leq d(z_{uv}, v)$
- 그렇지 않으면, d(u, v) < 트리의 지름
2. $d(z_{uv}, x) \leq d(z_{uv}, u)$
- 그렇지 않으면, d(u, v) < 트리의 지름
3. $d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u)$
- 그렇지 않으면, 첫 번째 BFS에서 x가 가장 먼 정점이 아님
4. $d(z_{xy}, y) \geq d(v, z_{uv}) + d(z_{uv}, z_{xy})$
- 그렇지 않으면, 두 번째 BFS에서 y가 가장 먼 정점이 아님

트리의 지름

Diamater



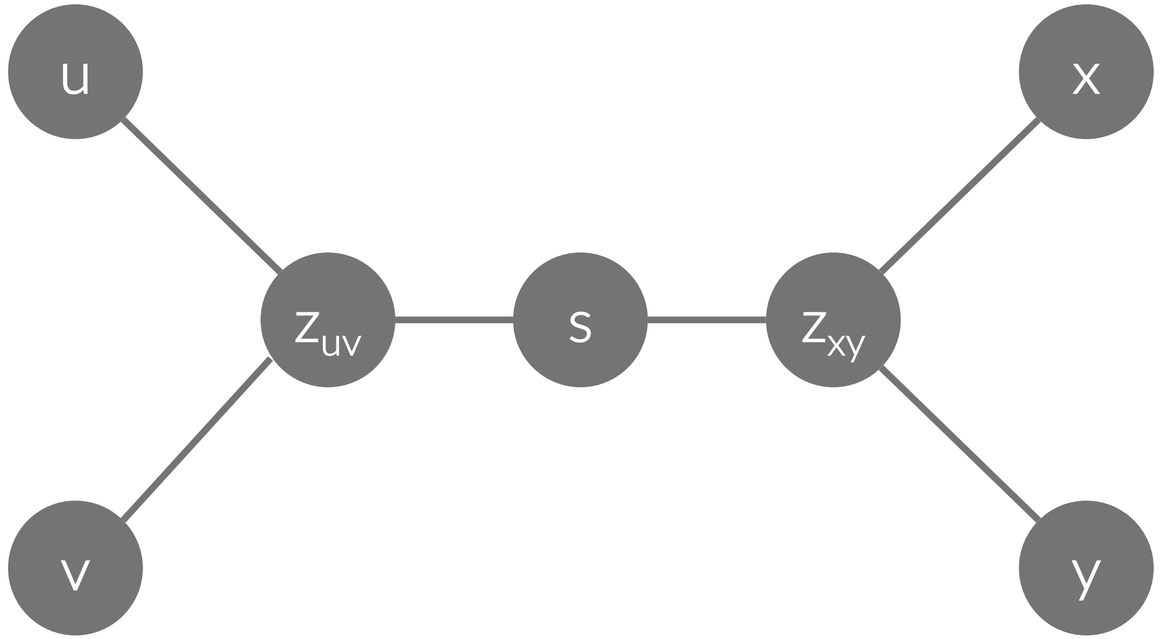
1. $d(z_{uv}, y) \leq d(z_{uv}, v)$
2. $d(z_{uv}, x) \leq d(z_{uv}, u)$
3. $d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u)$
4. $d(z_{xy}, y) \geq d(v, z_{uv}) + d(z_{uv}, z_{xy})$

- 1과 2 때문에
- $d(u,v) = d(z_{uv}, v) + d(z_{uv}, u) \geq d(z_{uv}, x) + d(z_{uv}, y) = d(x, y) + 2d(z_{uv}, z_{xy}) \geq d(x, y)$
- 3과 4 때문에
- $d(z_{xy}, y) + d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u) + d(v, z_{uv}) + d(z_{uv}, z_{xy})$

트리의 지름

Diamater

1. $d(z_{uv}, y) \leq d(z_{uv}, v)$
2. $d(z_{uv}, x) \leq d(z_{uv}, u)$
3. $d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u)$
4. $d(z_{xy}, y) \geq d(v, z_{uv}) + d(z_{uv}, z_{xy})$



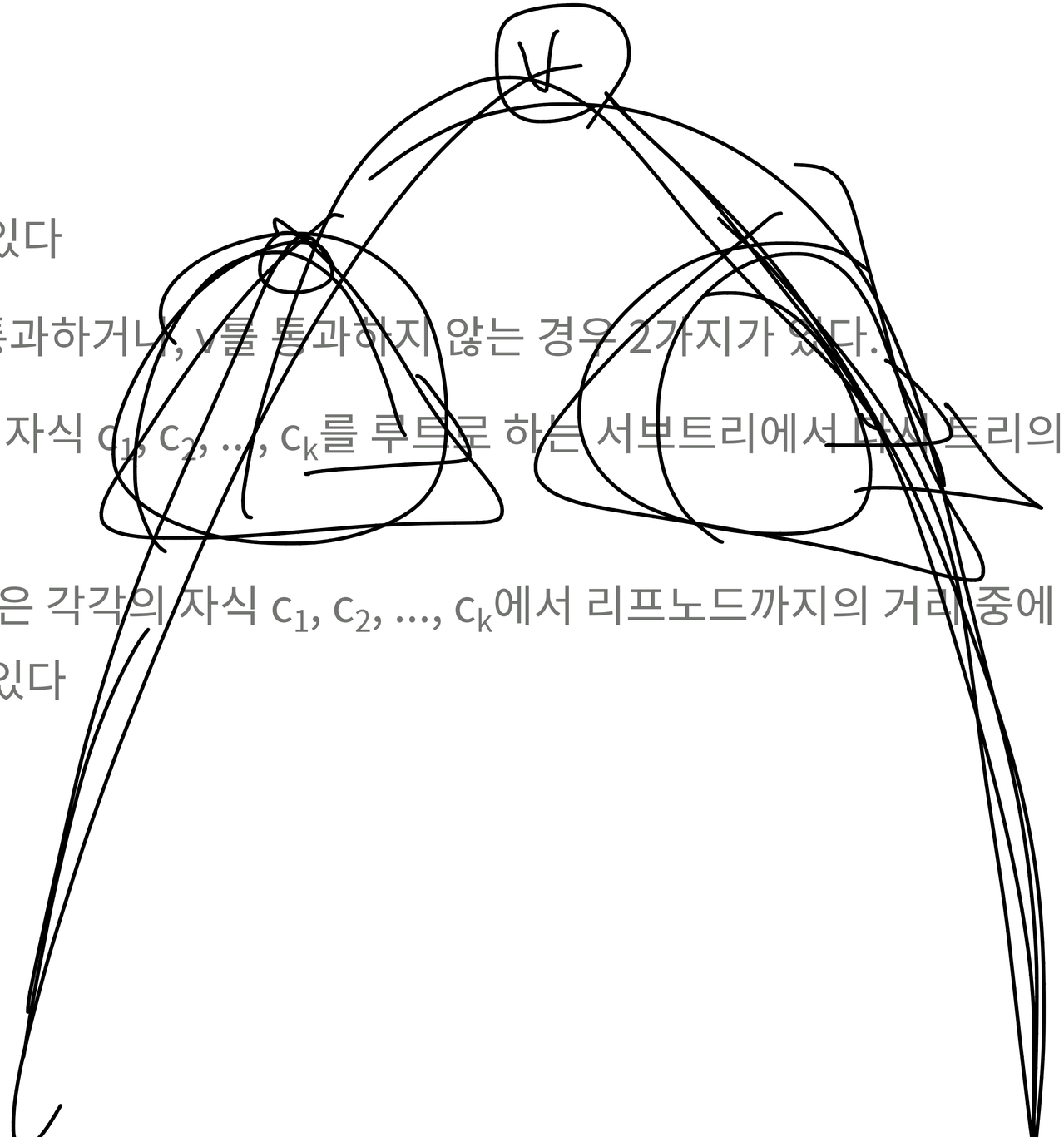
- $d(u,v) = d(z_{uv}, v) + d(z_{uv}, u) \geq d(z_{uv}, x) + d(z_{uv}, y) = d(x, y) + 2d(z_{uv}, z_{xy}) \geq d(x, y)$
- $d(z_{xy}, y) + d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u) + d(v, z_{uv}) + d(z_{uv}, z_{xy})$

- $d(x, y) = d(z_{xy}, y) + d(z_{xy}, x) \geq 2d(s, z_{uv}) + d(v, z_{uv}) + d(u, z_{uv}) \geq d(u, v)$
- 따라서, $d(u, v) = d(x, y)$

트리의 지름

Diamater

- 포스트 오더를 이용해서도 구할 수 있다
- 루트가 v일 때, 트리의 지름은 v를 통과하거나, v를 통과하지 않는 경우 2가지가 있다.
- v를 통과하지 않는 경우에는 각각의 자식 c_1, c_2, ..., c_k를 루트로 하는 서브트리에서 마찬가지의 지름을 구한다
- v를 통과하는 경우에는 트리의 지름은 각각의 자식 c_1, c_2, ..., c_k에서 리프노드까지의 거리 중에 가장 큰 값 2개를 이용해서 만들 수 있다



트리의 지름

https://www.acmicpc.net/problem/1167

- 트리의 지름을 구하는 문제

트리의 지름

https://www.acmicpc.net/problem/1167

- 탐색 2번 이용한 소스: http://codeplus.codes/40b58ad7a1514aa1aa3cef299e44000a
- 포스트 오더 이용한 소스: http://codeplus.codes/fb41c67d9691473e95c5e12070059e61

트리의 지름

https://www.acmicpc.net/problem/1967

- 트리의 지름을 구하는 문제

트리의 지름

https://www.acmicpc.net/problem/1967

- 탐색 2번 이용한 소스: http://codeplus.codes/6dcc22310dd54123ac497ef7a6df7bd4
- 포스트 오더 이용한 소스: http://codeplus.codes/25f2016bb5d644c483381270b505b6b3

끝

코드 플러스

https://code.plus

- 슬라이드에 포함된 소스 코드를 보려면 "정보 수정 > 백준 온라인 저지 연동"을 통해 연동한 다음, "백준 온라인 저지"에 로그인해야 합니다.
- 강의 내용에 대한 질문은 코드 플러스의 "질문 게시판"에서 할 수 있습니다.
- 문제와 소스 코드는 슬라이드에 첨부된 링크를 통해서 볼 수 있으며, "백준 온라인 저지"에서 서비스됩니다.
- 슬라이드와 동영상 강의는 코드 플러스 사이트를 통해서만 볼 수 있으며, 동영상 강의의 녹화와 다운로드, 배포와 유통은 저작권법에 의해서 금지되어 있습니다.
- 다른 경로로 이 슬라이드나 동영상 강의를 본 경우에는 codeplus@startlink.io 로 이메일 보내주세요.
- 강의 내용, 동영상 강의, 슬라이드, 첨부되어 있는 소스 코드의 저작권은 스타트링크와 최백준에게 있습니다.