땅울림
자구 스터디

Made by 규정

6주차

1

과제
리뷰

2

트리의
구현

3

BFS

1. 과제 리뷰

1 과제 리뷰

9934_완전 이진 트리

1 과제 리뷰

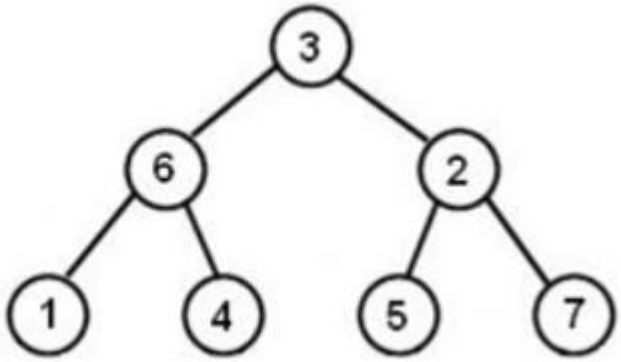
중위 순회로 주어진 완전 이진 트리(포화 이진 트리)를 레벨별로 출력

정확하게 말하면 포화 이진 트리지만, 완전 이진 트리라고 적는 경우도 많음(문제의 조건 잘 확인)

깊이가 K인 완전 이진 트리는 총 2^K-1개의 노드로 이루어져 있다.

포화 이진 트리

1 과제 리뷰



입력 : 1 6 4 3 2 5 7

관찰)

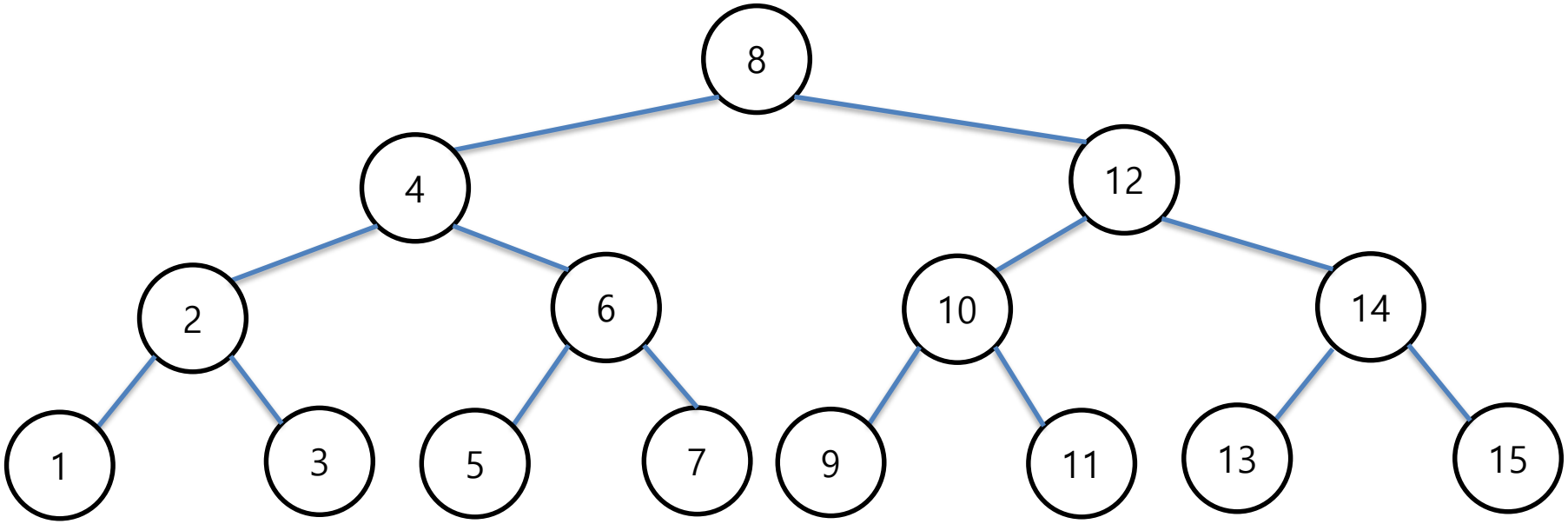
1. 항상 가운데 노드가 루트
2. 루트 노드를 기준으로 양쪽 서브트리의 노드 수가 같음
3. 포화이진트리의 서브트리도 포화이진트리 (노드 수가 2^N-1)

1 과제 리뷰

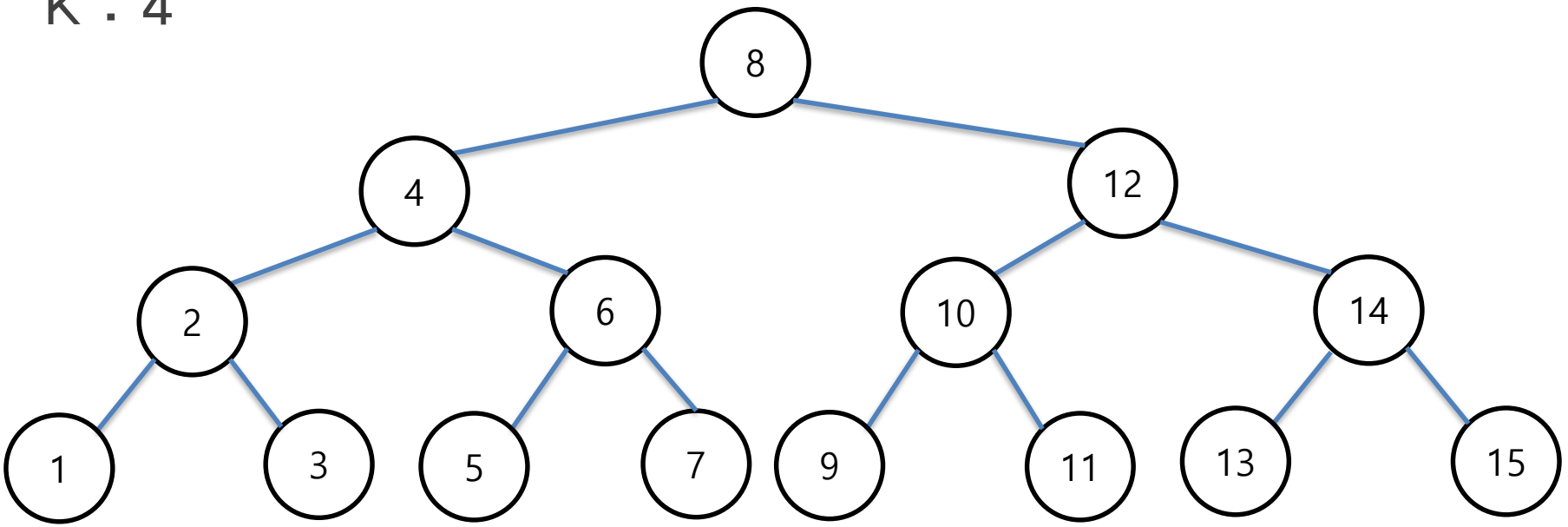
1) 규칙 찾아서 풀기

레벨 3으로는 규칙이 잘 안 보일 수 있으니 4짜리로 보자

왼쪽부터 입력받으므로 입력받는 순서대로 인덱스 부여



K : 4



	레벨별 인덱스	공차(d)
1개	8	.
2개	4 12	8
4개	2 6 10 14	4
8개	1 3 5 7 9 11 13 15	2

시작 위치

필요한 정보

시작 위치 : 8,4,2,1

공차 : x,8,4,2

개수 : 1,2,4,8

int sz=2^K를 구해놓고 시작

```
int start_idx = sz / 2; 시작 위치
int d = sz; 공차
int cnt = 1; 개수
for (int i = 0; i < k; i++)
{
    int idx = start_idx; 출력해야할 위치
    for (int j = 0; j < cnt; j++)
    {
        cout << v[idx] << " ";
        idx += d; 한번 출력하고 공차만큼 더해줌
    }
    start_idx /= 2;
    d /= 2;
    cnt *= 2;

    cout << "\n";
}
```

시작 위치와 공차는 반으로,
개수는 두배로

필요한 정보

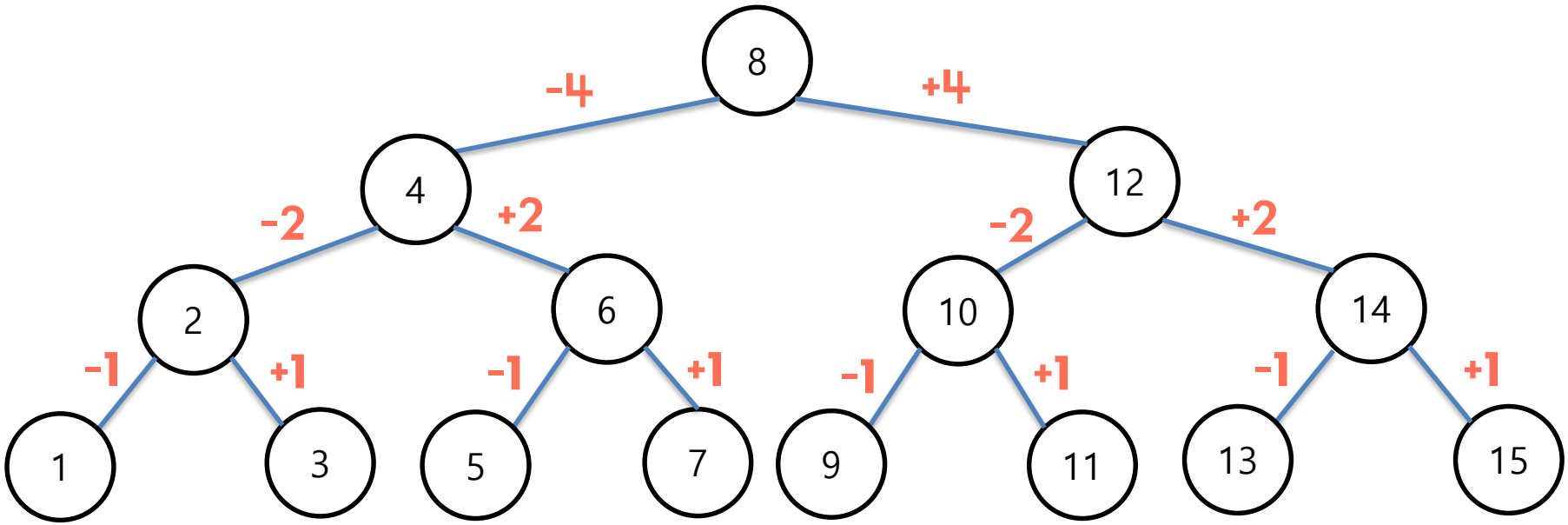
시작 위치 : 8,4,2,1

공차 : x,8,4,2

개수 : 1,2,4,8

2) 재귀함수 이용

K : 4



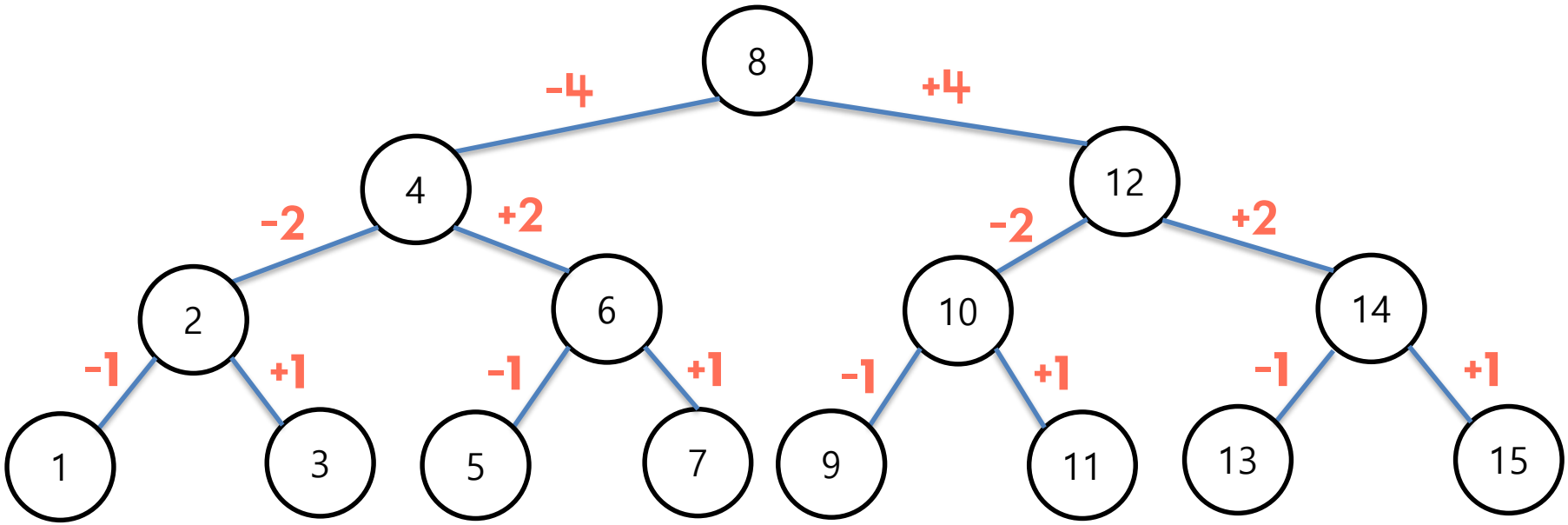
레벨을 거듭할수록 더하고 빼는 수가 반으로 줄어듦

시작 레벨 : 1

시작 인덱스 : $8(2^{K-1})$

처음 더하고 빼는 수 : $4(2^{K-2})$

K : 4

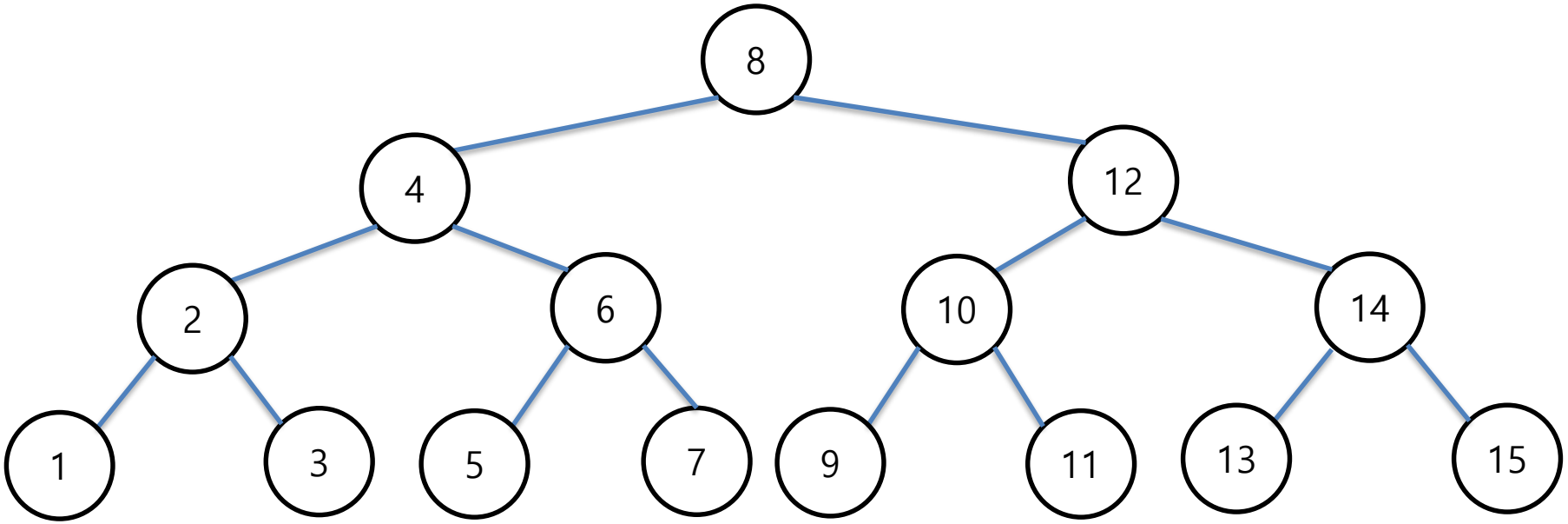


가운데부터 시작해서, 양쪽 서브트리로 재귀적으로 타고들어가면서 각 레벨에 해당하는 수를 저장하면 됨

재귀 함수 : func(레벨, 루트 인덱스, 더하고 빼는 수)

int sz=2^K를 구해놓고, func(1, sz/2, sz/4)로 시작

K : 4



재귀함수는 왼쪽 서브트리가 다 끝나고 나서야 오른쪽 서브트리로 이동하므로 레벨 순서대로 돌지 않음

=〉2차원 벡터를 전역으로 선언해서 각 레벨의 데이터를 순서대로 저장

```
void func(int level, int idx, int dif)
{
    if (level > k) return;
    ans[level].push_back(arr[idx]);
    func(level + 1, idx - dif, dif / 2);
    func(level + 1, idx + dif, dif / 2);
}
```

종료조건

레벨에 해당하는 데이터 입력

왼쪽 서브트리로 이동

오른쪽 서브트리로 이동

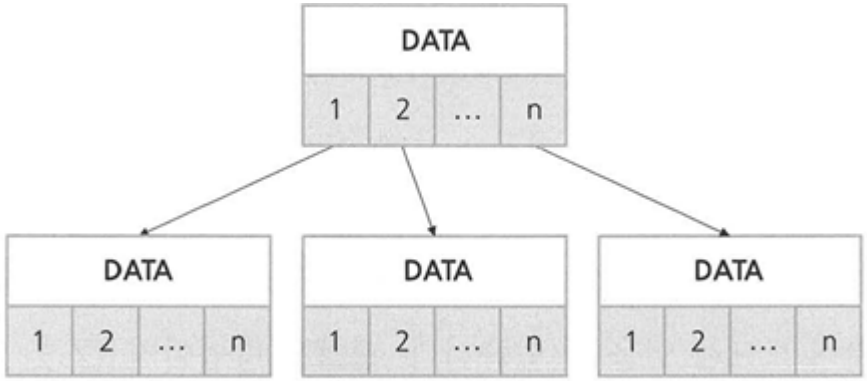
다 돌고나면 ans벡터에 담긴 데이터 레벨별로 출력해주면 끝

2. 트리의 구현

2 N-링크 표현법

N-링크 표현법

- 각 노드가 N개만큼의 링크를 가지고 있음
- 링크들이 각각 자식 노드를 가리킴



2 N-링크 표현법

장점

- 트리의 구조를 그대로 표현해서 직관적임
- 대부분의 상황에서 시간복잡도가 우수함

단점

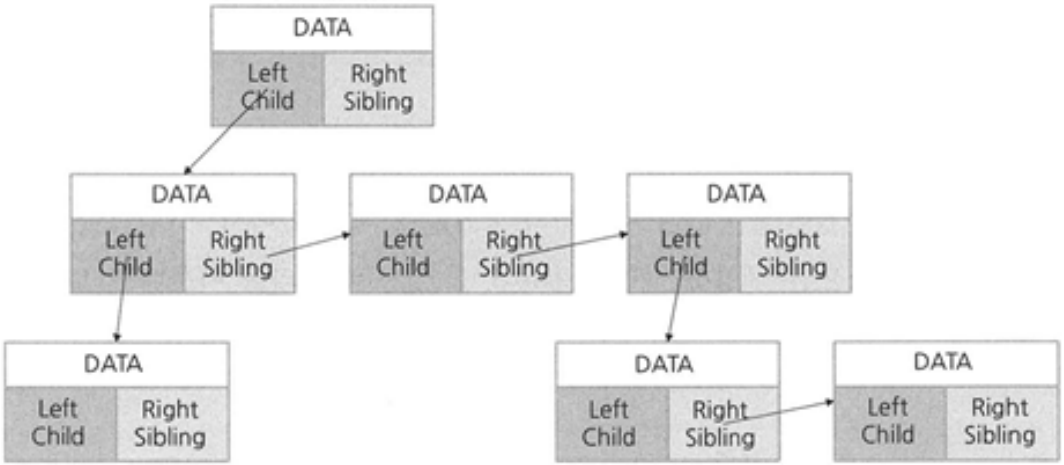
- 모든 노드가 N개의 링크를 가지고 있으므로 메모리 낭비가 심함
- 이를 해결하려면 연결리스트 사용해야하는데, 구현이 개복잡

이런 이유로 N-링크 표현법은 거의 쓰이지 않음

2 왼쪽 자식 - 오른쪽 형제 표현법

왼쪽 자식 – 오른쪽 형제 표현법

- 모든 트리를 이진트리로 구현하는 방법
- left child는 자식을, 오른쪽 child는 형제를 표현



2 왼쪽 자식 - 오른쪽 형제 표현법

장점

- 구현이 매우 간편함(사용도 간편함)
- 메모리 사용에 효과적

단점

- 구조가 바뀌므로 직관적이지 않음
- 자식 노드를 방문할 때 한번에 방문할 수 없음

진짜 웬만하면 거의 다 이 방식 사용

문제가 이진트리로 주어지거나, 변형하면 됨

2 트리의 구현

이론적으로는 이렇지만, 실제로 문제를 풀 때는 배열(벡터)을 사용해서 간단한 형태로 구현 가능

노드 트리 구조체를 굳이 만들 필요도 없음

물론 만들어야 할 때도 있음 ㅎ

대부분이 이진 트리지만, 이진 트리가 아니더라도 당황할 필요 x

2 트리의 구현

문제 : 1991_트리의 구현

입력

N이 26개

첫째 줄에는 이진 트리의 노드의 개수 N(1≤N≤26)이 주어진다. 둘째 줄부터 N개의 줄에 걸쳐 각 노드와 그의 왼쪽 자식 노드, 오른쪽 자식 노드가 주어진다. 노드의 이름은 A부터 차례대로 영문자 대문자로 매겨지며, 항상 A가 루트 노드가 된다. 자식 노드가 없는 경우에는 .으로 표현된다.

2 트리의 구현

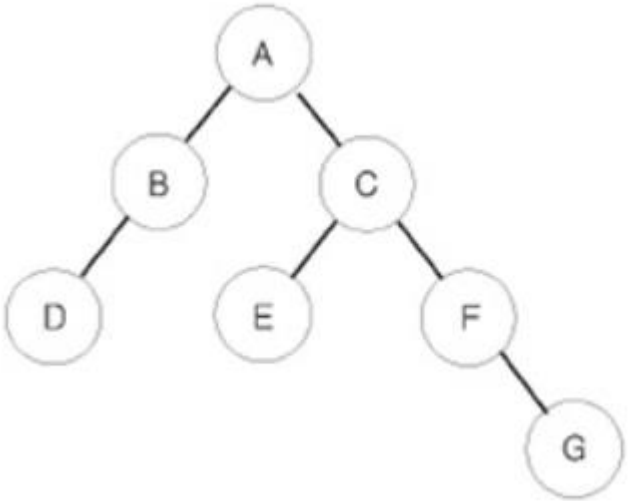
2차원 배열 선언

```
int node[26][2];
```

이진트리이므로 최대 2칸

0	A	B	C
1	B	D	.
2	C	E	F
⋮			
25	Z		

0 : A, 1 : B,… 25 : Z



2 트리의 구현

실습1) 1991_트리의 구현

구조체 만들지 말고 배열(벡터)만 사용해서 풀어보기

2 트리의 구현

이진 트리가 아니더라도 벡터로 간단하게 구현 가능

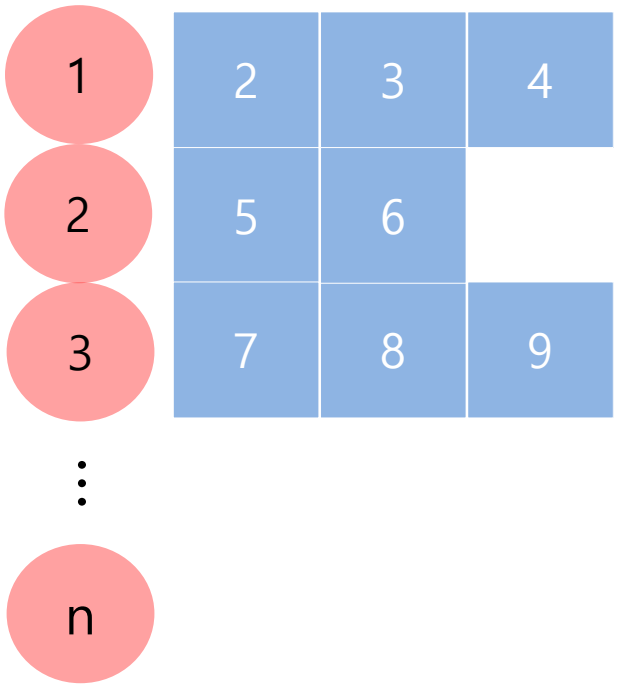
트리나 그래프를 벡터로 표현하는 방법은 정말 많이 쓰이기 때문에 할 줄 알아야함!

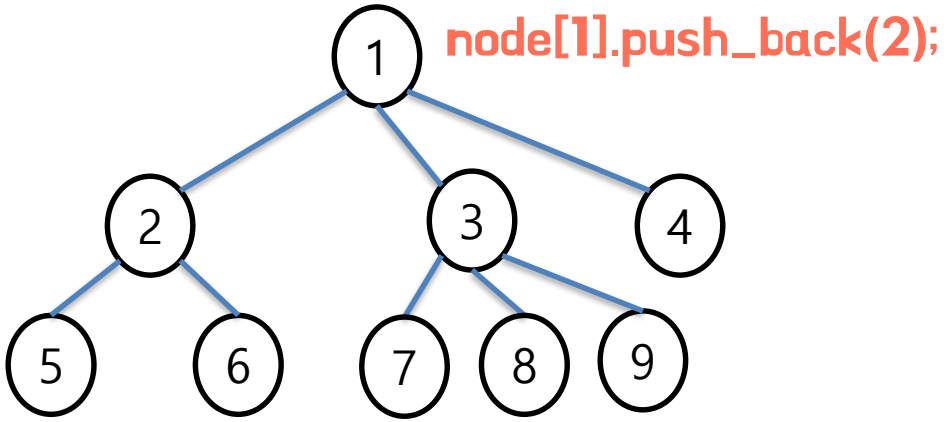
2 트리의 구현

2차원 벡터 선언

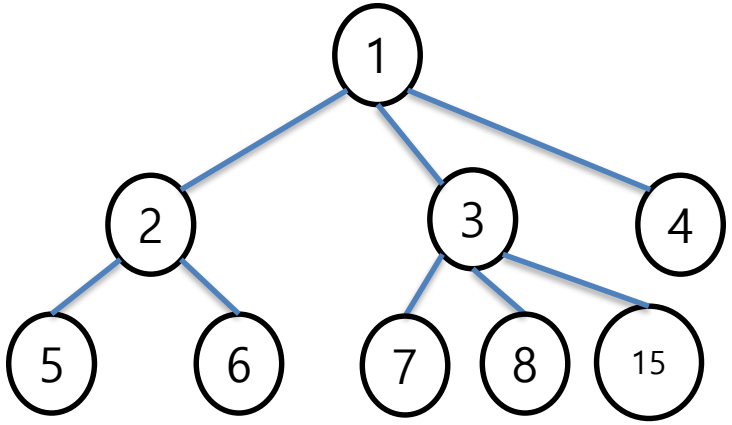
노드의 범위

```
vector <int> node[n+1];
```





2 트리의 구현



실습2)
첫째줄에 노드의 개수(N)가 주어진다.
다음 N-1개의 줄에 트리의 연결 정보(u, v)가 주어진다.
트리를 입력받아 전위순회, 후위순회를 한 결과를 출력해라.

입력 제한 : 1<=N<=10, 1 <= u, v <= 1000

예제 입력

```
9
1 2
1 3
1 4
2 5
2 6
3 7
3 8
3 15
```

예제 출력

```
1 2 5 6 3 7 8 15 4
5 6 2 7 8 15 3 4 1
```

3. BFS

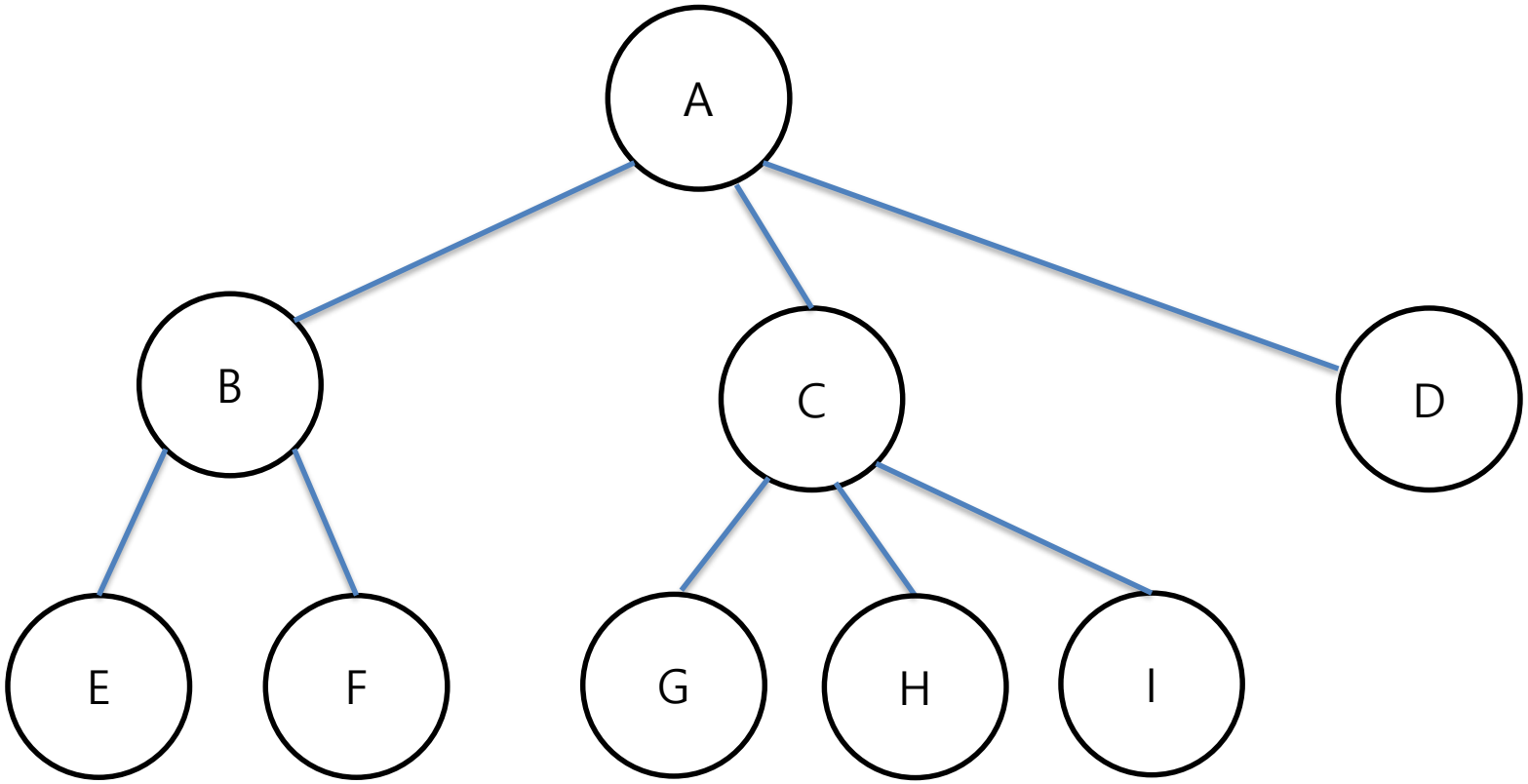
3 BFS

BFS(너비 우선 탐색)

- 레벨별로 타고 내려가면서 탐색하는 방식
- 레벨 오더 탐색이라고도 함
- 큐(queue) 사용해서 구현

3 BFS

위에서 아래로 탐색
같은 레벨에서는 어떤 순서대로 해도 상관없으나,
왼쪽부터 오른쪽으로 가는게 일반적

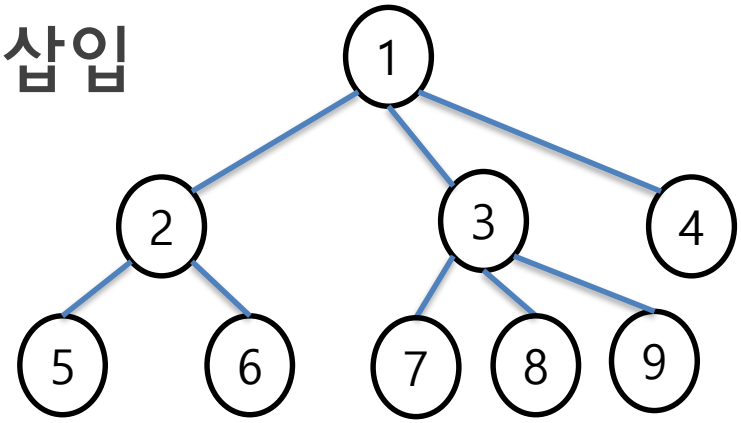


A–B–C–D–E–F–G–H–I

3 BFS

구현 방법 : 큐(queue) 사용

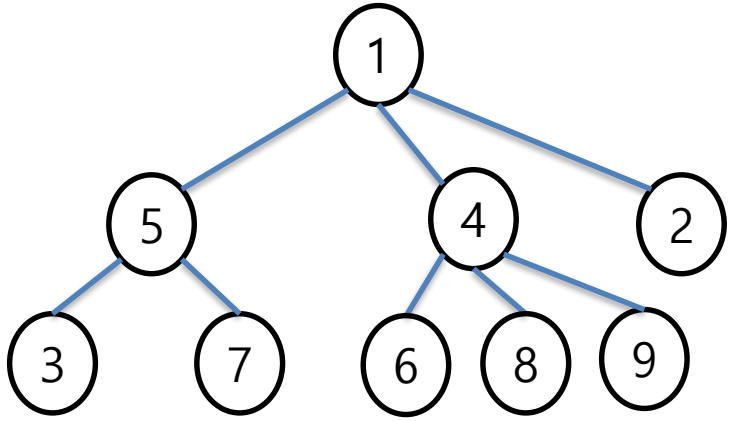
1. 루트부터 시작
2. 큐에서 하나씩 꺼냄
3. 꺼낸 노드의 자식을 모두 큐에 삽입
4. 큐가 빌 때까지 반복



queue

1 2 3 4 5 6 7 8 9

3 BFS



실습2)
첫째줄에 노드의 개수(N)이 주어진다.
다음 N-1개의 줄에 트리의 연결 정보(u, v)가 주어진다.
트리를 입력받아 BFS탐색을 한 결과를 출력해라.

입력 제한 : 1<=N<=100, 1 <= u, v <= 1000

예제 입력

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

예제 출력

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

감사합니다.

Made by 규정