땅울림

05

트리 Tree

자료구조 스터디 5회차

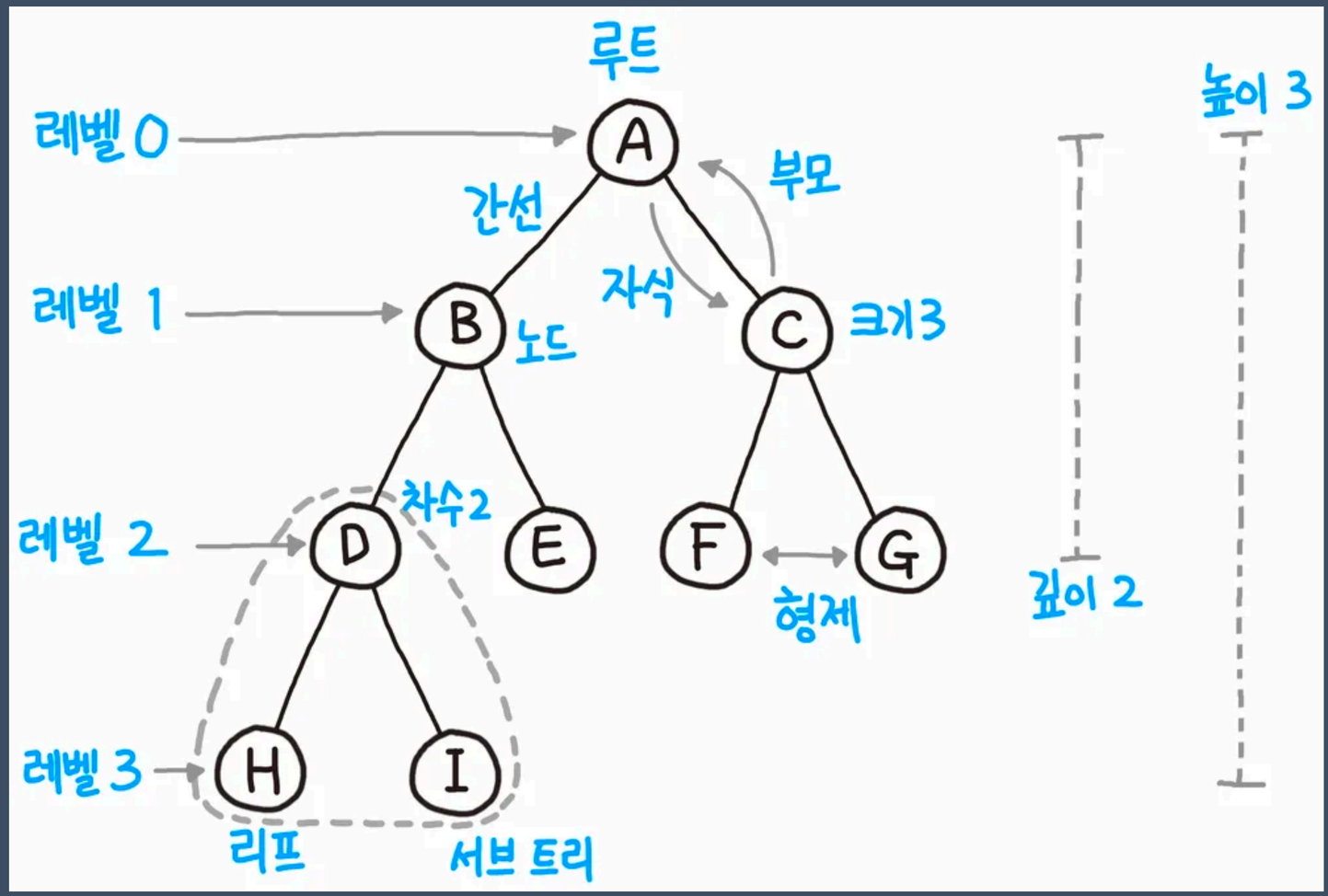
- 트리 개념설명
- 트리 구현방법
- 유형 설명

1

트리 개념 설명

계층 구조를 나타내는 비선형 자료구조
노드(node)와 **간선(edge)**이 존재한다.

- 루트(root)
- 레벨(level)
- 깊이(depth)
- 높이(height)
- 부모(parent)
- 자식(child)
- 조상(ancestor)
- 형제(sibling)
- 리프(leaf)



2

트리 구현 방법

트리 구현
: 트리 멤버함수

insertNode(x, y)
: x 노드의 자식으로 y 노드를 삽입

deleteNode(x)
: x 노드를 삭제
x 노드의 자식 노드들은
x 노드의 부모의 자식 노드가 된다.
(입양이라 생각)

findNode(x)
: x 노드의 포인터값(위치) 반환

노드 클래스 다이어그램

Node
+ parent : Node* + value : Int + childList : vector<Node*>
+ Node(Node*, int) <<create>>

트리 클래스 다이어그램

Tree
+ root : Node* + nodeList : vector<Node*>
+ Tree( ) <<create>> + findNode(int) : Node* + insertNode(int, int) : void + deleteNode(int) : void + 기타 등등

노드 클래스 생성자

+ Node(Node*, int) <<create>>

```
Node(Node* p, int v) {
// p 노드를 부모로 하는 v 값을 가진 노드 생성

    parent = p; // 부모 할당
    value = v; // 값 할당
}
```

트리 클래스 생성자

+ Tree( ) <<create>>

```
Tree() {
    root = new Node(nullptr, 1);
    // root 노드 생성(부모 없음, 값 1)

    nodeList.push_back(root);
    // 트리에 루트 노드 넣기
}
```

인자값에 해당하는 노드의 포인터값을 반환해주는 함수

+ findNode(int) : Node*

```
Node *findNode(int v) {
    for (int i=0; i<nodeList.size(); i++) { // 트리의 노드 리스트를 순회하며
        if (nodeList[i]->value == v) { // 인자값을 가진 노드 찾으면
            return nodeList[i]; // 포인터값 반환
        }
    }
    return nullptr; // 인자값을 가진 노드 없으면 nullptr 반환
}
```

```
findNode 5
findNode(5)의 실행 결과는 0x600000c0c030입니다.
```

삽입연산 (insert x y) : x를 부모로 하는 y값 가지는 노드 생성

+ insertNode(int, int) : void
1. 해당 노드의 부모 노드의 포인터값 알아내기 (findNode(x)) 2. x 값을 가지는 새로운 노드 생성 (Node *newNode = new Node(parentNode, y) 3. 부모 노드의 childList에 새로운 노드 추가 (push_back 사용) 4. 트리의 nodeList에 새로운 노드 추가 (push_back 사용)

삽입연산 (insert x y) : x를 부모로 하는 y값 가지는 노드 생성

+ insertNode(int, int) : void

```
void insertNode(int x, int y) { // 삽입 연산(트리에 노드 추가)
    Node *parentNode = findNode(x);   // 부모가 될 노드의 포인터

    if (parentNode == nullptr) { // x값을 가진 노드가 없을 경우(이상한 경우)
        cout << "-1\n"; // 문제조건따라 에러 처리
    } else { // 올바르게 추가하는 경우
        Node *newNode = new Node(parentNode, y); // y값을 가지는 노드 생성
        parentNode->childList.push_back(newNode); // 부모노드의 '자식 리스트'에 새 노드 추가
        nodeList.push_back(newNode); // '트리'에 새 노드 추가
    }
}
```

삭제연산 (delete x)
: x 노드 삭제 후, 'x 노드의 자식들'은 'x 노드의 부모의 자식들'로 입양 보냄

+ deleteNode(int) : void

1. 해당 노드의 포인터값 알아내기 (findNode(x))
2. 삭제할 노드의 자식들을 부모 노드의 자식으로 변경해주기(입양보내기)
   자신의 childList 순회하며
   (1. 해당 노드 부모의 자식 리스트에 추가)
   (2. 자식의 부모 포인터 업데이트)
3. 부모 노드의 자식 리스트(childList)에서 삭제할 노드 삭제
   parent의 childList 순회하며 해당 노드 발견하면 erase
4. 트리의 노드 리스트(nodeList)에서 해당 노드 삭제
   트리의 nodeList 순회하며 해당 노드 발견하면 erase
5. delete 노드

1.
```
Node *deleteNode = findNode(x);
```

2.
```
// 삭제할 해당 노드의 자식들을 부모 노드의 자식으로 변경해주기
for (int i=0; i<deleteNode->childList.size(); i++) { // 자식 수만큼 반복
    deleteNode->parent->childList.push_back(deleteNode->childList[i]); // 자식 업데이트
    deleteNode->childList[i]->parent = deleteNode->parent; // 부모 업데이트
}
```

3.
```
// 부모 노드의 자식 리스트에서 해당 노드 삭제
for (int i=0; i<deleteNode->parent->childList.size(); i++) {
    if (deleteNode == deleteNode->parent->childList[i]) {
        deleteNode->parent->childList.erase(deleteNode->parent->childList.begin() + i);
        break; // 삭제하고 더이상 반복문 수행할 필요 없으면 break
    }
}
```

4.
```
// 트리의 노드 리스트에서 해당 노드 삭제
for (int i=0; i<nodeList.size(); i++) {
    if(deleteNode == nodeList[i]) {
        nodeList.erase(nodeList.begin() + i);
        break; // 삭제하고 더이상 반복문 수행할 필요 없으면 break
    }
}
```

5.
```
delete deleteNode; // delete 해주기
```

3

트리 문제 유형

트리 구현

+

특정 함수 구현

1

특정 노드의 depth를 출력

21년도 6주차 2번문제
22년도 6주차 2-1번문제
22년도 김영호 6주차 4번 문제

2

특정 노드의 자식들을 출력

21년도 6주차 1번문제
22년도 6주차 1-1번문제

특정 노드의 level을 출력

21년도 6주차 4번문제

특정 노드의 부모를 출력

21년도 6주차 3번문제

같은 부모를 가지는 자식들 출력

22년도 6주차 1-2번문제

어렵

특정 depth에 속하는 노드 중 최대값 출력

22년도 6주차 2-2번문제

특정 노드의 모든 조상들 출력

22년도 김영호 6주차 2번문제

특정 노드의 자식들 중 최대값 출력

22년도 김영호 6주차 1번문제

특정 노드의 자식들 중 최소값 출력

22년도 김영호 6주차 3번문제

1. 특정 노드의 depth 출력

+ printDepth(int) : void

```
Node *curNode = findNode(x);

if (curNode == nullptr) { // 트리에 없는 노드를 가리키면 -1 출력
    cout << "-1\n";
} else {
    int depth = 0;
    while(curNode != root) {
        curNode = curNode->parent;
        depth++;
    }
    cout << depth << "\n";
}
```

2. 특정 노드의 자식들 출력

+ printChilds(int) : void

```
Node *curNode = findNode(x); // 순회용 노드를 인자값에 해당하는 노드로 초기화

if (curNode == nullptr || curNode->childList.empty()) {
    cout << "-1\n";  // 에러 처리
} else {
    for (Node* child : curNode->childList) {
        cout << child->value << " ";
    }
    cout << "\n";
}
```

3. 특정 depth에 속하는 자식들 중 최대값 출력

+ printLargestAtDepth(Node*, int, int&) : void

```
void printLargestAtDepth(Node* root, int depth, int& maxNodeValue) {
    if(root == nullptr) {
        return;
    }
    if(depth == 0) {
        maxNodeValue = max(maxNodeValue, root->value);
    }
    for(Node* child : root->childList) {
        printLargestAtDepth(child, depth-1, maxNodeValue);
    }
}
```

문제 풀어보기~

—

트리
Tree