INHA UNIVERSITY

ICE2004 자료구조론

제 목

자료구조론 과제 5 보고서

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2021년 12월 17일

학부
학년
성명 성시열
학번

[개요 및 구현상 특징]

- 파일은 node.h 헤더파일과 binaryTree.h 헤더파일, binarySearchtree.h 헤더파일. 총 세 개의 헤더파일과 이 헤더파일의 기능을 활용할 수 있는 main.cpp 드라이버 파일로 구성되어있다. 해당 코드의 개요는 헤더파일들이 서로 교집합 관계를 이루며 BST는 BT를 상속받아 사용한다. (node < binaryTree < binarySearchtree))

- node.h 헤더파일은 Binary Search Tree의 요소들을 각각 연결하고 안의 값을 넣을 수 있게 하는 역할을 한다. Node 구조체는 그 노드의 값을 의미하는 element, 부모 노드와 연결할 parent, 그리고 왼쪽, 오른쪽 노드와 연결할 수 있는 left_child, right_child로 구성되어있다. 이후 함수는 각 멤버를 reading하고 writing 할 수 있는 get, set 함수로 되어있고, isExternal()함수를 통해 자식 노드가 비어있다면 가장 바깥쪽 노드임을 나타낼 수 있다.

- binaryTree.h 헤더파일은 Tree의 형태이지만 아직 Binary Search Tree의 조건은 포함이 안된 Binary Tree이다. Tree의 형태이기 때문에 ROOT가 유무에 따라 Tree의 유무도 결정된다. 그렇기 때문에 멤버로 ROOT가 있고, 함수로는 setRoot, getRoot와 Tree의 존재유무를 파악할 수 있는 isEmpty()함수가 있다. 또한, Binary Tree의 3가지 traversal 수행할 수 있는 preorder_traversal, inorder_traversal, postorder_traversal 함수를 생성하였다. 특징으로는 사용자 입장에서 보다 쉽게 함수를 실행하고 안내문을 출력하는 preorder, inorder, postorder 함수까지 생성하였다.

- binarySearchtree.h 헤더파일은 degree가 2인 Binary Tree를 그대로 사용하면서 새로운 조건 하나가 추가된 Tree이다. 그렇기 때문에 BinaryTree 클래스를 상속하여 코드 작성의 효율성을 높였다. 이 클래스는 함수로만 이루어져 있으며 크게 삽입과 삭제를 진행하는 함수로 구성되어 있다. inserter 함수를 통해 원하는 값을 tree에 삽입하는데 이 때 Binary Search Tree의 조건인 왼쪽 자식 노드의 값 <= 자기 자신의 값 <= 오른쪽 자식 노드의 값을 성립하는 코드를 작성한다. remove 함수도 마찬가지로 지우고 싶은 값을 입력하여 제거하는데, 이 때도 Binary Search Tree의 조건이 망가지지 않도록 코드를 구성하였다. 그리고 BinaryTree와 마찬가지로 사용자 입장에서 보다 쉽게 함수를 실행하고 특정 조건에서 실행시킬 수 있게 insert함수와 remove 함수를 생성하였다.
- 모든 헤더파일에서 공통적으로 template <typename T>을 사용하여 자료형을 int에만 제한하지 않고 유연하게 코드를 구성

- main.cpp 파일은 tree에 원하는 노드 개수만큼 삽입하여 tree를 완성하고 3가지 traversal을 출력해본다. 이후 삭제와 삽입 작업도 진행하고 이후의 traversal도 출력한다.

[코드 분석]

1. node.h

코드, 목적 및 상세설명

`bool isExternal() {`
`  return left_child == nullptr && right_child == nullptr;`
`  }`

가장 바깥쪽인 External node인지 확인하는 isExternal(). 뒤의 insert와 remove함수에 활용될 예정. 노드의 left_child와 right_child가 비어있을 때 true, left와 right가 비어있다는 것은 가장 끝 노드라는 것을 의미한다.

2. binaryTree.h

코드, 목적 및 상세설명

`void preorder_traversal(Node<T>* node) {`
`  if (node != NULL) {`
`  cout << node->getEle() << " ";`
`  preorder_traversal(node->getLeft());`
`  preorder_traversal(node->getRight());`
`  }`
`}`

preorder()를 통해 ROOT를 넘겨받아 이 함수에서는 node로 쓰임. 이 함수는 재귀함수이고, node가 NULL일 때까지 left_child와 right_child로 내려가면서 null인 node를 만나면 재귀호출을 멈춘다. 값을 먼저 출력하고 node의 left_child의 node를 다시 preorder_traversal()로 넘겨준 뒤 node의 right_child의 node를 다시 preorder_traversal()로 넘겨줌.

`void inorder_traversal(Node<T>* node) {`
`  if (node != NULL) {`
`  inorder_traversal(node->getLeft());`
`  cout << node->getEle() << " ";`
`  inorder_traversal(node->getRight());`
`  }`
`  }`

inorder()를 통해 ROOT를 넘겨받아 이 함수에서는 node로 쓰임. 이 함수는 재귀함수이고, node가 NULL일 때까지 left_child와 right_child로 내려가면서 null인 node를 만나면 재귀호출을 멈춘다. node의 left_child의 node를 다시 inorder_traversal()로 넘겨준 뒤 값을 출력하고 node의 right_child의 node를 다시 inorder_traversal()로 넘겨줌.

`void postorder_traversal(Node<T>* node) {`
`  if (node != NULL) {`
`  postorder_traversal(node->getLeft());`
`  postorder_traversal(node->getRight());`
`  cout << node->getEle() << " ";.`
`  }`
`  }`

postorder()를 통해 ROOT를 넘겨받아 이 함수에서는 node로 쓰임. 이 함수는 재귀함

수이고, node가 NULL일 때까지 left_child와 right_child로 내려가면서 null인 node를 만나면 재귀호출을 멈춘다. node의 left_child의 node를 다시 postorder_traversal()로 넘겨준 뒤 node의 right_child의 node를 다시 postorder_traversal()로 넘겨주고 값을 출력함.

3. binarySearchtree.h

코드, 목적 및 상세설명

```
void inserter(Node<T>* node, T key) {
        Node<T>* newNode = new Node<T>(key, nullptr, nullptr, nullptr);

        if (key == node->getEle()) {
            return;
        }
        else if (key < node->getEle()) {
            if (node->getLeft() == NULL) {
                node->setLeft(newNode);
            }
            else {
                inserter(node->getLeft(), key);
            }
        }
        else if (key > node->getEle()) {
            if (node->getRight() == NULL) {
                node->setRight(newNode);
            }
            else {
                inserter(node->getRight(), key);
            }
        }
    }
```

inserter 함수 크게 3가지 경우로 나누어 코드를 작성하였다. Key와 node의 element가 같을 때, 작을 때, 클 때.

값이 같을 때는 함수를 끝내고 다를 때는 왼쪽자식과 오른쪽 자식 중 비어있는 자식이 있다면 그 값을 넣어주고 그렇지 않으면 inserter 함수를 호출하였다.

자세한 코드 분석은 주석을 통해 작성하였다.

```
void remover(Node<T>* parentNode, Node<T>* deleteNode) {
if (deleteNode->isExternal()) {
            if (deleteNode == this->getRoot()) {
                this->setRoot(nullptr
            }
            else {
                if (parentNode->getLeft() == deleteNode) {
                    parentNode->setLeft(nullptr);
                }
                else if (parentNode->getRight() == deleteNode)
                    parentNode->setRight(nullptr);
                }
            }
```

```
        }
        else if (deleteNode->getLeft() == NULL || deleteNode->getRight() == NULL) {
            Node<T>* childNode = (deleteNode->getLeft() != NULL) ? deleteNode->getLeft() :
deleteNode->getRight();
if (deleteNode == this->getRoot()) {
                this->setRoot(childNode);
            }
            else {
                if (parentNode->getLeft() == deleteNode) {
                    parentNode->setLeft(childNode);
                }
                else if (parentNode->getRight() == deleteNode)
                    parentNode->setRight(childNode);
                }
            }
        }
else {
            Node<T>* temp = deleteNode;
            Node<T>* down_temp = deleteNode->getRight();
            while (down_temp->getLeft() != NULL) {
                temp = down_temp;
                down_temp = down_temp->getLeft();.
            }

            if (down_temp == temp->getLeft()) {
                temp->setLeft(down_temp->getRight());
            }
            else {
                temp->setRight(down_temp->getRight());
            }

            deleteNode->setEle(down_temp->getEle());
            deleteNode = down_temp;
        }
        delete deleteNode; }
```

remover 함수. 총 3가지 경우로 나누어 코드를 구성하였다. 지우려는 노드가 external node일 때, 지우려는 노드의 child 가 둘 다 NULL 일 때, 하나만 NULL일 때, 둘 다 NULL이 아닐 때.

마지막의 경우 지우려는 노드를 기준으로 한 번 오른쪽으로 간 뒤 쭉 왼쪽으로 내려가서 있는 external node와 값이 바꾼 뒤 그 external node를 제거한다.

자세한 코드 분석은 주석을 통해 작성하였다.

```
void insert(T key) {
        if (this->isEmpty()) {
            Node<T>* newNode = new Node<T>(key, nullptr, nullptr, nullptr);
            this->setRoot(newNode);
        }
        else {
            inserter(this->getRoot(), key);
        }
    }
```

사용자가 사용 시 insert와 넣고 싶은 값을 편하게 넣을 수 있게 하는 인터페이스 역할.

```
void remove(T key) {
        if (this->isEmpty()) {
            cout << "Tree가 비어있습니다." << endl;
            return;
        }
        Node<T>* parentNode = nullptr;
        Node<T>* deleteNode = this->getRoot();.
        while (deleteNode != NULL && deleteNode->getEle() != key) {
            parentNode = deleteNode;

            deleteNode = (key < parentNode->getEle()) ? parentNode->getLeft() : parentNode-
>getRight();
        }

        if (deleteNode == NULL) {
            cout << "값이 Tree에 존재하지 않습니다." << endl;
            return;
        }
        else {
            remover(parentNode, deleteNode);
        }
    }
```

사용자가 사용 시 remove와 넣고 싶은 값을 편하게 넣을 수 있게 하는 인터페이스 역할. remove함수는 insert함수와는 다르게 조건문 하나를 추가하여 tree가 비어있을 때는 출력문을 출력하도록 했다.

4. main.cpp

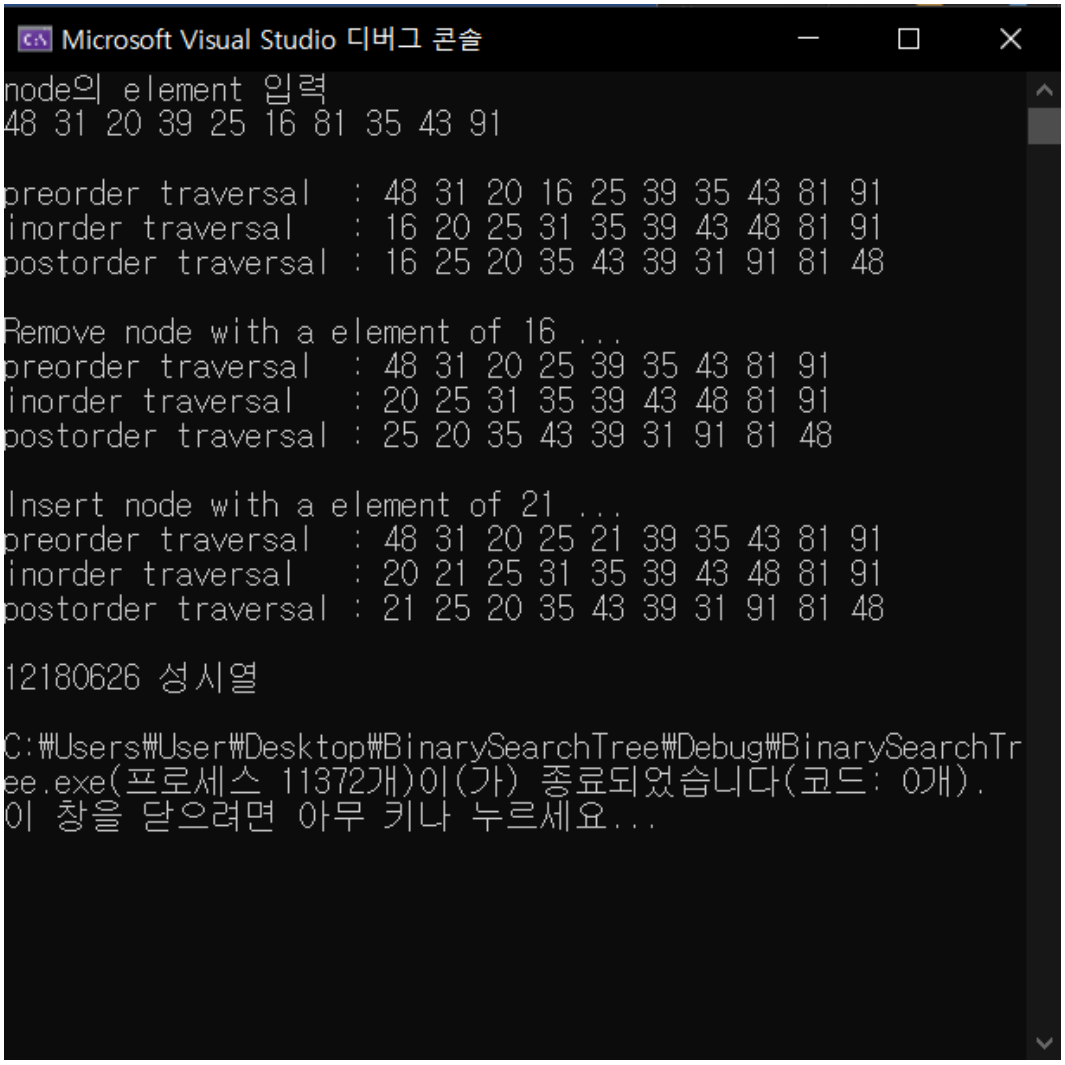
코드, 목적 및 상세설명

```
int n;
    cout << "원하는 node의 개수 : ";
    cin >> n;
    cout << "node의 element 입력" << endl;
    for (int i = 0; i < n; i++) {
        int a;
        cin >> a;
        tree.insert(a);
    }
    cout << endl;
```

for문을 통해 원하는 개수만큼 삽입을 할 수 있게 함.

이 외의 코드에 대한 모든 설명은 주석으로 자세히 작성함.

[실행결과분석]



먼저 원하는 node의 개수를 설정할 수 있도록 정수 값을 입력 받는다. (10 입력)

이후 node의 element를 순서상관 없이 원하는 개수만큼 입력한다.

Binary Search Tree이기 때문에 순서상관 없이 입력하더라도 코드에 의해 자동으로 정렬된다.

이후 정렬된 Tree를 3가지 traversal에 의해 출력한다.

이후 제거함수의 활용을 확인하기 위해 16이라는 원소를 제거하고 배열을 확인한다.

이후 삽입함수의 활용을 확인하기 위해 21이라는 원소를 추가하고 배열을 확인한다.

이후 학번과 이름을 출력한다.

[결론 및 느낀 점, 어려웠던 점]

마지막 자료구조론 과제를 하면서 어려운 점이 많았다. 과제에서 제시한 문제 중 트리 형태로 화면에 출력하지 못했다. 여러 번 시도를 하였으나 삭제와 삽입 과정에서 트리의 모양이 망가지는 것을 해결하지 못해 결국 배열 형태로 출력하였다. 이는 과제 제출이 종료되더라도 반드시 해결해보고 싶은 과제이다.

이번 과제는 단계별로 헤더파일을 작성하고 그 헤더파일을 포함하는 헤더파일 식으로 코드를 구성해보았다. 이렇게 구성한 이유는 Binary Tree까지는 구성하는데 큰 어려움이 없었는데, Binary Search Tree를 구성하는데 어려움이 있어 하나하나 차근히 해보자는 생각에 코드를 구성하게 되었다.

한 한기 동안 자료구조론을 배우면서 다양한 이론에 대해서도 배우게 되었지만, 어떠한 해결해야할 문제, 이루고 싶은 목적이 있을 때 어떻게 코드를 구성해야 하는지 확실하게 알진 못했지만 어느정도 감은 잡힌 느낌이다. 이 과목을 배우면서 한 학기로는 이 과목을 완벽히 이해할 수 없다고 생각했다. 이후에 알고리즘 관련된 과목을 들어야할 때 이 자료구조론이 초석이 될 예정인데 다음학기가 되기 전에 이번 자료구조론 수업을 복습하며 코드를 구성하는 시야를 키워야겠다.