**REPORT**

**[Prac 5]**



**과 목 : 알고리즘06**

**담당교수 : 주종화 교수님**

**학 과 : 컴퓨터공학과**

**학 번 : 2021111971**

**이 름 : 이재혁**

텍스트, 클립아트이(가) 표시된 사진

자동 생성된 설명

**프로그램 구현 환경**

프로그래밍언어 : C++

텍스트 에디터 : Visual Studio Code

운영체제 : MAC OS

**문제 1 : 레드블랙트리 삽입**

현재 트리에 value가 4인 노드 삽입하는 과정을 보이시오

**텍스트, 도표, 친필, 폰트이(가) 표시된 사진

자동 생성된 설명**

**레드블랙트리 삽입과정**

1. 이진탐색트리처럼 노드를 삽입 후 적색으로 설정
2. 레드블랙트리의 성질에 맞게 트리 수정

**트리 수정 case**

Case 1

삽입한 노드(기준 노드)의 삼촌이 적색인 경우

부모와 삼촌의 노드를 흑색으로 변경,

조부모를 적색으로 변경 후 조부모를 기준으로 다시 트리 수정

Case 2 & Case 3  
삼촌이 흑색이며 기준 노드가 오른쪽(Case 2), 왼쪽 (Case 3) 인 경우

Case2는 Case3로 변환 후 우회전 후 부모(B)와 오른쪽 자식(C)의 색 변경

* 부모가 조부모의 왼쪽 자식인 경우의 예시이고,

부모가 조부모의 오른쪽 자식인 경우는 오른쪽으로

노드를 모아서 좌회전 적용

**텍스트, 스크린샷, 친필, 도표이(가) 표시된 사진

자동 생성된 설명**

|  |  |
| --- | --- |
| **텍스트, 도표, 친필, 폰트이(가) 표시된 사진  자동 생성된 설명** | 노드 4 삽입    적색 노드의 자식이 적색 노드이므로 트리조정 필요 |
|  | 삽입한 노드 기준으로 삼촌 노드가 적색 노드   * Case1   부모 노드와 조부모 노드 색 변경 |
|  | 조부모 노드를 기준 노드로 변경 후 트리 확인  적색 노드의 자식 노드가 적색 노드, 트리조정 필요  삼촌 노드가 흑색이고 기준 노드가 오른쪽 자식   * Case2   좌회전 적용 후 우회전 |

|  |  |
| --- | --- |
| **도표, 라인, 원이(가) 표시된 사진  자동 생성된 설명** | 좌회전  기준 노드의 부모 노드가 왼쪽 자식이 되고, 원래의 왼쪽 자식은 부모 노드의 오른쪽 자식으로 설정  조부모 노드와 연결 |
| **도표, 라인, 원이(가) 표시된 사진  자동 생성된 설명** | Case3로 변경 후 우회전 적용  기준 노드의 부모 노드가 오른쪽 자식이 되고 원래의 오른쪽 자식은 부모 노드의 왼쪽 자식으로 설정 |
| **도표이(가) 표시된 사진  자동 생성된 설명** | 회전 후 노드의 색을 변경해 삽입을 완료한다. |

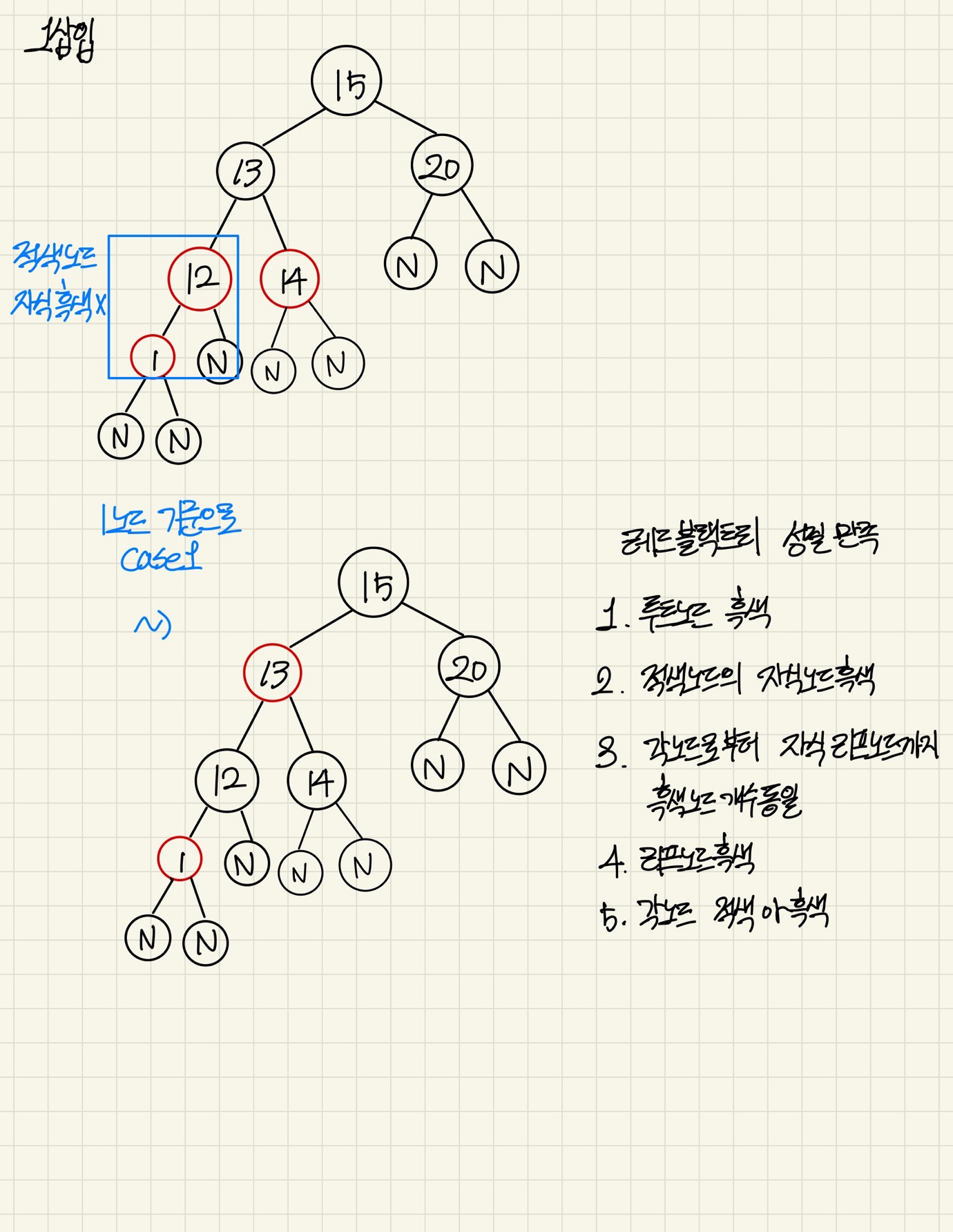
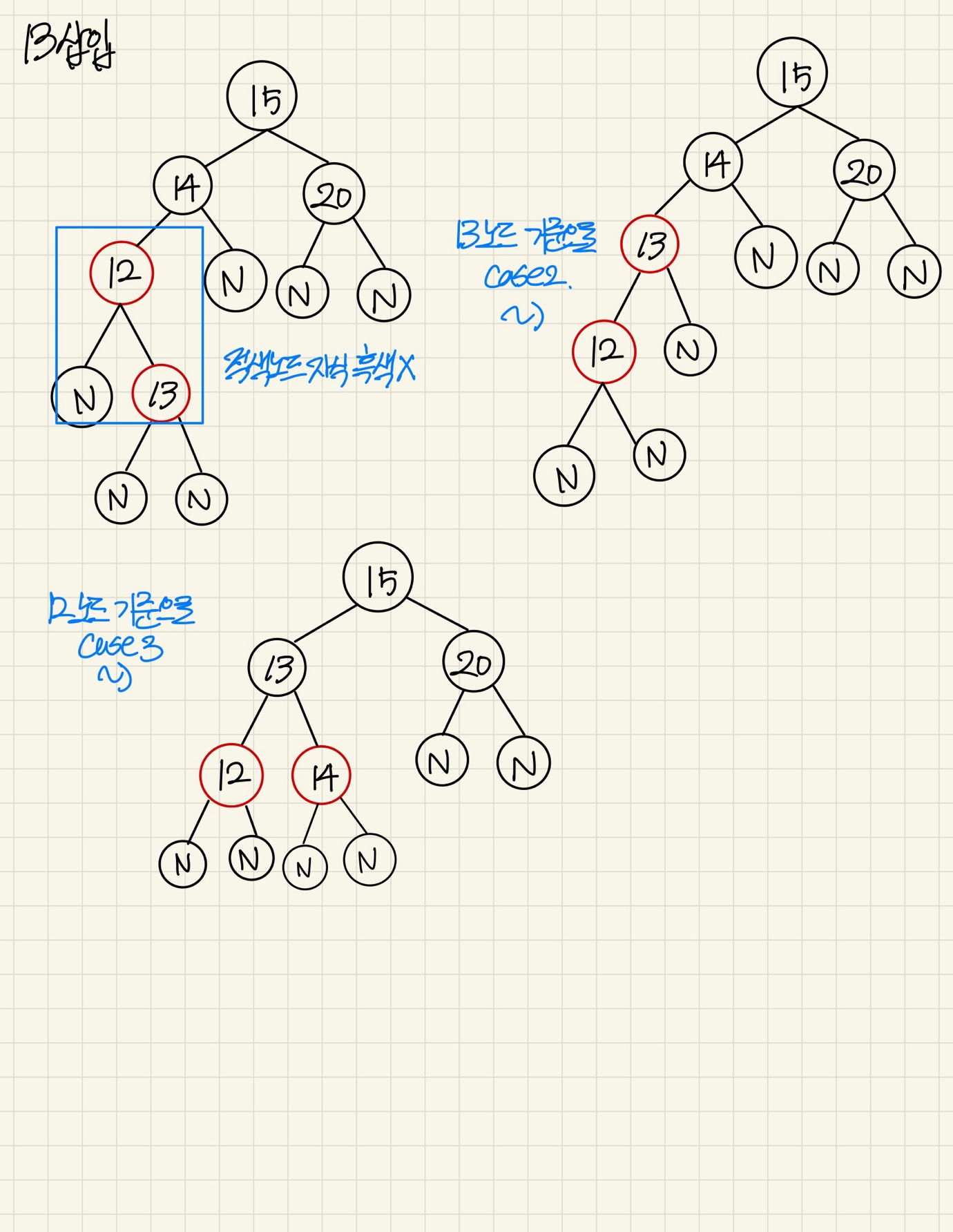
**문제 2 : 레드블랙트리 생성**

빈 레드블랙 트리에 키 20, 15, 14, 12, 13, 1을 차례로

삽입

**텍스트, 친필, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

****

**문제 3**

레드블랙 트리의 삽입을 구현

**프로그램 설계**

노드 한개의 클래스를 구현해 root노드에서 메소드를 실행해 노드의 삽입 진행

T.nil은 NULL포인터로, 만약 연결된 포인터가 없다면 흑색 노드로 판단

**RBNode 클래스**

**멤버 변수**

color : 노드의 색, RED || BLACK의 값을 가짐

value : 노드가 가지는 값

parent : 부모 노드를 가리키는 포인터

left : 왼쪽 자식 노드를 가리키는 포인터

right : 오른쪽 자식 노드를 가리키는 포인터

**멤버 메소드**

생성자 : value를 매개변수로 노드를 생성한다.

적색 노드로 생성하는데, 루트 노드라면

흑색으로 생성

toggleColor() : 현재 노드의 색을 변환

print() : 트리를 출력하는 메소드

텍스트, 폰트, 그래픽, 그래픽 디자인이(가) 표시된 사진

자동 생성된 설명

가운데 부모 노드를 기준으로 같은 깊이의

들여쓰기는 자식 노드를 의미한다.

먼저 출력된 노드가 오른쪽 노드이며

트리가 왼쪽으로 누운 상태로 출력된다.

insert(int v) : 전달받은 매개변수로 노드를 삽입하는 메소드

이진 트리의 성질에 맞게 노드를 삽입 후,

레드 블랙 트리의 성질에 맞게 트리 수정

turnLeft() : 현재 노드를 기준으로 좌회전

부모 노드와 조부모 노드를 확인

부모 노드가 루트 노드일 경우

조부모 노드의 연결부분은 생략

turnRight() : 현재 노드를 기준으로 우회전

부모 노드와 조부모 노드를 확인

부모 노드가 루트 노드일 경우

조부모 노드의 연결부분은 생략

fixTree(RBNode \*z)

매개변수로 전달받은 노드를 기준으로 트리 수정

1. 현재 노드가 루트 노드일 경우 루트 노드의 색만 흑색으로 설정
2. 현재 입력 받은 노드는 적색이기 때문에, 부모 노드가 적색인지 확인
3. 부모가 적색일 경우 부모가 조부모 노드의 어떤 자식인지 확인
4. 삼촌 노드의 색을 확인 후 case에 맞게 트리 수정

getRoot() : 최상위 노드를 반환하는 메소드

트리를 조정하면 루트의 위치가 바뀌기 때문에

최상위 노드를 찾아 반환해 루트 노드 최신화

buildTree(int n) : 매개변수를 전달받아 해당하는 개수의

노드를 가지는 트리 생성

**소스코드**

// 2021111971 이재혁 문제 3

#include <iostream>

#include <string>

using namespace std;

class RBNode {

private:

string color; // Node의 색

int value; // Node의 값

RBNode \*parent; // 부모노드

RBNode \*left; // 왼쪽 자식노드

RBNode \*right; // 오른쪽 자식노드

public:

RBNode(int value = 0, int isRoot = 0) {

// 매개변수가 2개라면 root 노드

if (isRoot) { // root 노드라면 흑색으로 설정

color = "BLACK";

} else {

color = "RED";

}

this->value = value; // 값 설정

RBNode \*left = NULL;

RBNode \*right = NULL;

RBNode \*parent = NULL;

// 아직 연결 X

}

void toggleColor() { // 현재 설정된 색상 반전

if (this->color == "RED") {

this->color = "BLACK";

} else {

this->color = "RED";

}

}

void print(int n = 0) { // 트리 출력 n은 현재 출력할 노드의 깊이

// 같은 부모를 가지는 형제를 같은 들여쓰기로 표현

// 자식2(오른쪽) -> 같은 들여쓰기 형제관계

// 부모

// 자식1(왼쪽)

// 트리가 왼쪽으로 누운 상태

if (this->right) { // 오른쪽 자식이 존재하면

// 깊이를 1추가해 노드 출력

this->right->print(n + 1);

} else { // 오른쪽 자식이 없다면

// 깊이를 1추가해 Nil임을 표시

for (int i = 0; i < n + 1; i++) {

cout << " ";

}

cout << "Nil" << endl;

}

for (int i = 0; i < n; i++) { // 현재 깊이 만큼 들여쓰기

cout << " ";

}

// 현재 노드의 값, 색 출력

cout << this->value << "(" << this->color << ")" << endl;

// 자식 노드 출력

if (this->left) { // 왼쪽 자식이 존재하면

// 깊이를 1 추가해 노드 출력

this->left->print(n + 1);

} else { // 왼쪽 자식이 없다면

// 깊이를 1추가해 Nil임을 표시

for (int i = 0; i < n + 1; i++) {

cout << " ";

}

cout << "Nil" << endl;

}

}

void insert(int v) { // 노드 삽입

if (this->value < v) {

// 삽입할 노드의 값이 현재 값보다 클 때

if (this->right) {

// 오른쪽 자식이 있다면, 왼쪽 자식에서 삽입

this->right->insert(v);

} else {

// 오른쪽 자식이 없다면, 새로운 노드를 생성해 연결

RBNode \*node = new RBNode(v);

this->right = node;

node->parent = this;

// 삽입노드 기준으로 트리조정

fixTree(node);

}

} else {

// 삽입할 노드의 값이 현재 노드의 값보다 작을 때

if (this->left) {

// 왼쪽 자식이 있다면, 왼쪽 자식에서 삽입

this->left->insert(v);

} else {

// 왼쪽 자식이 없다면, 새로운 노드를 생성해 연결

RBNode \*node = new RBNode(v);

this->left = node;

node->parent = this;

// 삽입노드 기준으로 트리조정

fixTree(node);

}

}

}

void turnLeft() {

RBNode \*p = this->parent; // 현재 노드의 부모노드

RBNode \*gp = this->parent->parent; // 현재 노드의 조부모노드

if (gp == NULL) { // 부모 노드가 루트면 -> 현재 노드가 루트가 됨

// 부모 노드의 오른쪽 자식을 현재 노드의 왼쪽 자식으로 변경

p->right = this->left;

if (this->left) {

// 왼쪽 자식이 존재하면 부모노드 변경

this->left->parent = p;

}

// 현재 노드의 왼쪽 자식을 부모노드로 설정

this->left = p;

p->parent = this;

// 현재 노드의 부모 노드 Nil로 설정됨 -> root 노드가 됨

this->parent = gp;

} else {

if (p == gp->left) {

// 부모가 조부모의 왼쪽 자식일 경우

// 부모 노드의 오른쪽 자식을 현재 노드의 왼쪽 자식으로 변경

p->right = this->left;

if (this->left) {

// 왼쪽 자식이 존재하면 부모노드 변경

this->left->parent = p;

}

// 현재 노드의 왼쪽 자식을 부모노드로 설정

this->left = p;

p->parent = this;

// 부모 노드가 조부모 노드의 왼쪽 자식이었기 때문에

// 조부모 노드의 왼쪽 자식으로 변경

gp->left = this;

this->parent = gp;

} else {

// 부모가 조부모의 오른쪽 자식일 경우, 회전 후

// 조부모 노드의 연결부분만 오른쪽 자식으로 변경

p->right = this->left;

if (this->left) {

this->left->parent = p;

}

this->left = p;

p->parent = this;

gp->right = this;

this->parent = gp;

}

}

}

void turnRight() {

RBNode \*p = this->parent; // 현재 노드의 부모노드

RBNode \*gp = this->parent->parent; // 현재 노드의 조부모노드

if (gp == NULL) { // 부모 노드가 루트면 -> 현재 노드가 루트가 됨

// 부모 노드의 왼쪽 자식을 현재 노드의 오른쪽 자식으로 변경

p->left = this->right;

if (this->right) {

// 오른쪽 자식이 존재하면 부모노드 변경

this->right->parent = p;

}

// 현재 노드의 오른쪽 자식을 부모 노드로 연결

this->right = p;

p->parent = this;

// 현재 노드의 부모 노드 Nil로 설정됨 -> root 노드가 됨

this->parent = gp;

} else {

if (p == gp->left) { // 부모가 조부모의 왼쪽 자식일 경우

// 부모 노드의 왼쪽 자식을 현재 노드의 오른쪽 자식으로 변경

p->left = this->right;

if (this->right) {

// 오른쪽 자식이 존재하면 부모노드 변경

this->right->parent = p;

}

// 현재 노드의 오른쪽 자식을 부모 노드로 연결

this->right = p;

p->parent = this;

// 부모 노드가 조부모 노드의 왼쪽 자식이었기 때문에

// 조부모 노드의 왼쪽 자식으로 변경

gp->left = this;

this->parent = gp;

} else {

// 부모가 조부모의 오른쪽 자식일 경우, 회전 후

// 조부모 노드의 연결부분만 오른쪽 자식으로 변경

p->left = this->right;

if (this->right) {

this->right->parent = p;

}

this->right = p;

p->parent = this;

gp->right = this;

this->parent = gp;

}

}

}

void fixTree(RBNode \*z) { // 삽입한 노드 기준으로 트리 조정

if (z->parent) {

// 조정하고자 하는 노드가 루트 노드가 아닐경우 조정

// 루트 노드의 조정은 색만 확인

RBNode \*p = z->parent; // 기준 노드의 부모노드

RBNode \*gp = p->parent; // 기준 노드의 조부모노드

if (p->color == "RED") { // 부모노드가 RED -> 조정필요

if (p == gp->left) { // 부모가 왼쪽 자식일 경우

if (gp->right && gp->right->color == "RED") {

// 삼촌노드가 적색 노드일 경우

p->toggleColor();

if (gp->right) {

gp->right->toggleColor();

// 삼촌노드 흑색으로 설정

}

gp->toggleColor(); // 조부모 노드 적색으로 설정

this->fixTree(gp); // 조부모 노드 기준으로 트리 수정

} else { // 삼촌이 Nil이거나, 흑색노드인 경우

if (z == p->right) { // Case 2인 경우

z->turnLeft(); // Case 3가 됨

z->turnRight();

z->toggleColor(); // 조부모가 된 z 흑색으로 설정

z->right

->toggleColor(); // 자식이된 노드 적색으로 설정

} else { // Case 3의 경우 부모를 기준으로 우회전

p->turnRight();

p->toggleColor(); // 조부모가 된 p 흑색으로 설정

p->right

->toggleColor(); // 자식이된 노드 적색으로 설정

}

}

} else { // 부모가 오른쪽 자식일 경우

if (gp->left && gp->left->color == "RED") {

// 삼촌노드가 적색 노드일 경우

p->toggleColor(); // 부모노드 흑색으로 설정

if (gp->left) {

gp->left->toggleColor();

// 삼촌노드 흑색으로 설정

}

gp->toggleColor();// 조부모 노드 적색으로 설정

this->fixTree(gp); // 조부모 노드 기준으로 트리 수정

} else { // 삼촌이 Nil이거나, 흑색노드인 경우

if (z == p->left) { // Case 2인 경우

z->turnRight(); // Case 3가 됨

z->turnLeft();

z->toggleColor(); // 조부모가 된 z 흑색으로 설정

z->right

->toggleColor(); // 자식이된 노드 적색으로 설정

} else { // Case 3의 경우 부모를 기준으로 좌회전

p->turnLeft();

p->toggleColor(); // 조부모가 된 p 흑색으로 설정

p->left

->toggleColor(); // 자식이된 노드 적색으로 설정

}

}

}

}

}

// 루트 노드 흑색으로 설정

RBNode \*tmp = this;

while (tmp->parent) {

tmp = tmp->parent;

}

tmp->color = "BLACK";

}

// 트리가 조정되면 root노드가 가리키는 값이 최상위 노드가 아닐 수 있음

RBNode \*getRoot() {

RBNode \*tmp = this; // 현재 노드부터

while (tmp->parent) { // 부모가 Nil일 때 까지, 즉 루트노드까지 이동

tmp = tmp->parent;

}

return tmp; // 루트 노드 반환

}

RBNode \*buildTree(int n) { // 트리 생성

RBNode \*root;

int input;

cout << "노드 입력 >> ";

cin >> input;

cout << "-----Root 생성-----" << endl;

root = new RBNode(input, 1); // 루트 생성

root->print();

for (int i = 0; i < n - 1; i++) {

cout << "노드 입력 >> ";

cin >> input;

cout << "-----" << input << " 삽입-----" << endl;

root->insert(input); // 입력받은 값으로 노드 삽입

root = root->getRoot(); // 루트 노드 최신화

root->print(); // 트리 확인

}

return root; // 루트 노드 반환

}

};

int main() {

RBNode \*root;

int input;

cout << "노드 개수 입력 >> ";

cin >> input;

root = root->buildTree(input);

}

// 20 15 14 12 13 1

**결과 분석**

|  |  |
| --- | --- |
| 텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진  자동 생성된 설명 | **텍스트, 친필, 스크린샷이(가) 표시된 사진  자동 생성된 설명** |
|  | **텍스트, 친필, 스크린샷이(가) 표시된 사진  자동 생성된 설명** |
|  | **텍스트, 친필, 스크린샷이(가) 표시된 사진  자동 생성된 설명** |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |