1. 프로그램 소스

#include <stdio.h>

#include <stdlib.h>

typedef struct Node { //이진트리 노드 구조체

struct Node\* left; //왼쪽 자식 노드

struct Node\* right; //오른쪽 자식 노드

}Node;

typedef struct roman { //isEmperor()에서 사용하기 위한 로만함수여부와 자식의 크기차이 저장 구조체

int isroman, issize;

}roman;

Node\* initTree(); //이진트리 만드는 함수

roman\* isEmperor(Node \*root); //황제노드 찾는 함수

Node\*\* makequeue(int\*, int\*); //새로운 큐 생성

void enqueue(struct node \*\*, int \*, struct node\*); //큐 추가입력

Node \*dequeue(struct node \*\*, int \*); //큐 삭제

Node \*\* makequeue(int \*front, int \*rear) {

Node \*\*queue = (Node \*\*)malloc(sizeof(struct node\*) \* 500); //queue의 메모리할당

\*front = \*rear = 0; //queue의 처음도 끝도 0

return queue;

}

void enqueue(struct node \*\*queue, int \*rear, Node \*new) { //큐 추가입력

queue[\*rear] = new; //queue[끝(크기)] 새로 작성

(\*rear)++; //증가

}

Node \*dequeue(Node \*\*queue, int \*front) { //큐 삭제

(\*front)++;

return queue[\*front - 1];

}

Node\* initTree() { //이진트리설정

int i;

Node\* node[29]; //들어가는 노드는 29개.

for (i = 0; i < 29; i++) {

node[i] = (Node\*)malloc(sizeof(Node));//for문 사용하여 node들 동적 메모리할당시키기.

node[i]->left = NULL;

node[i]->right = NULL;

}

node[0]->left = node[1];

node[1]->left = node[3];

node[3]->left = node[7];

node[7]->left = node[13];

node[7]->right = node[14];

node[14]->left = node[23];

node[14]->right = node[24];

node[24]->right = node[25];

node[24]->left = node[26];

node[25]->left = node[27];

node[25]->right = node[28];

node[3]->right = node[8];

node[1]->right = node[4];

node[4]->left = node[9];

node[4]->right = node[10];

node[9]->left = node[15];

node[9]->right = node[16];

node[10]->left = node[17];

node[10]->right = node[18];

node[0]->right = node[2];

node[2]->left = node[5];

node[2]->right = node[6];

node[6]->right = node[12];

node[12]->right = node[22];

node[11]->right = node[20];

node[6]->left = node[11];

node[11]->left = node[19];

node[12]->left = node[21];

//트리 구현

return node[0]; //root노드 반환

}

roman\* isEmperor(Node\* root) { //황제노드 구하기

roman \*A = (roman \*)malloc(sizeof(roman));

roman \*B = (roman \*)malloc(sizeof(roman));

roman \*C = (roman \*)malloc(sizeof(roman));

if ((root->left == NULL) && (root->right == NULL)) { //노드에게 자식이 없는경우(root)

C->isroman = 1; //roman C의 isroman에 1저장

C->issize = 1; //roman C의 issize에 1 저장 (부트리의 크기가 1)

return C;

}

else { //내부노드인경우

if (root->left != NULL) //노드의 왼쪽이 0이 아닌경우

A = isEmperor(root->left); //A가 가르키는 바는 노드의 왼쪽을 isEmperor에 대입한 값

else { //0인경우

A->isroman = 1;

A->issize = 0;

}

if (root->right != NULL) //노드의 오른쪽이 0이 아닌 경우

B = isEmperor(root->right); //B가 가르키는 바는 노드의 오른쪽은 isEmperor에 대입한 값

else { //0인경우

B->isroman = 1;

B->issize = 0;

}

if (((-5 <= ((A->issize) - (B->issize))) && ((A->issize) - (B->issize)) <= 5) && ((A->isroman == 1) || (B->isroman == 1)))

//A와 B의 크기차이가 5이내고 노드의 오른쪽도 왼쪽도 황제노드가 아닌경우

C->isroman = 1;

else

C->isroman = 0;

C->issize = (A->issize) + (B->issize) + 1;

return C;

}

}

int main() {

int cnt = 0, e\_cnt = 0; //레벨, 황제노드 횟수

int rear, front, i;

int cnt1 = 1, cnt2;

Node\* root;

root = initTree(); //root트리 구현

Node \*\*queue = makequeue(&front, &rear);

Node \*temp = root; //temp 임시로 만듦. 내용은 root와 동일

while (temp) {

while ((temp->left != NULL) && (temp->right != NULL)) {

cnt2 = 0;

for (i = 0; i < cnt1; i++) {

cnt++;

if (temp->left) {

enqueue(queue, &rear, temp->left); //큐 입력

cnt2++;

}

if (temp->right) {

cnt2++;

enqueue(queue, &rear, temp->right); //큐 입력

}

temp = dequeue(queue, &front); //temp 큐 삭제

}

cnt1 = cnt2;

isEmperor(temp); //temp에서 황제노드 찾기

if ((isEmperor(temp)->isroman) == 0) { //roman C의 isroman이 0이라면

printf("%d 황제노드\n", cnt);

e\_cnt++; //황제노드 갯수 추가

}

}

break;

}

printf("방문한 내부노드 수 %d개, 황제노드 수 %d개\n", cnt+1, e\_cnt);//cnt는 root에서 다른 내부노드까지 가는 횟수로 root를 세지 못하였기에 +1 한다.

return 0;

}

2. 프로그램 입출력 캡쳐 이미지

