

자료구조2 실습 레포트



과목명		자료구조2 실습
담당교수		홍 민 교수님
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20204059
이름		이예빈

목 차

1. 학생 정보 이진 트리를 전위, 중위, 후위로 출력하는 프로그램

1.1 문제 분석

1.2 소스 코드

1.3 소스 코드 분석

1.4 실행창

1.5 느낀점

2. 완전 이진 트리 검증 프로그램

2.1 문제 분석

2.2 소스 코드

2.3 소스 코드 분석 – 레벨 순회, 그리고 단말 노드 판별

2.3.1 추가 – 더 효율적인 방식(단말 노드의 개수 확인)

2.4 실행창

2.5 느낀점

3. 느낀점

1. 학생 정보 이진 트리를 전위, 중위, 후위로 출력하는 프로그램

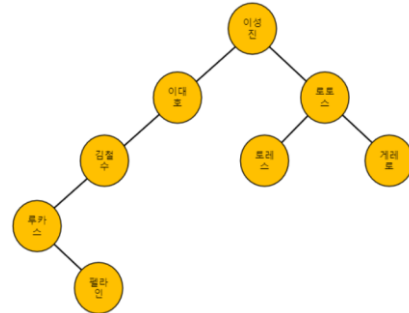
이진 트리 1

- 파일 data.txt에 아래와 같은 학번과 이름들이 저장되어 있다.
 - data.txt 파일에서 데이터를 입력
 - 학번을 이용하여 insert_node함수로 데이터를 이진 트리에 저장
 - 저장된 정보를 전위, 중위, 후위순으로 출력

```
data.txt - 텍스트
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
50 이상진
46 이대호
44 김철수
27 루카스
32 멜라인
84 로토스
71 도레스
86 게레로

C:\Users\Sungjin\source\repos\Project1\Debug\Project1.exe
전위 순회 : 50 이상진 → 46 이대호 → 44 김철수 → 27 루카스 → 32 멜라인 → 84 로토스 → 71 도레스 → 86 게레로
중위 순회 : 27 루카스 → 32 멜라인 → 44 김철수 → 46 이대호 → 50 이상진 → 84 로토스 → 71 도레스 → 86 게레로
후위 순회 : 32 멜라인 → 27 루카스 → 44 김철수 → 46 이대호 → 71 도레스 → 86 게레로 → 84 로토스 → 50 이상진
```

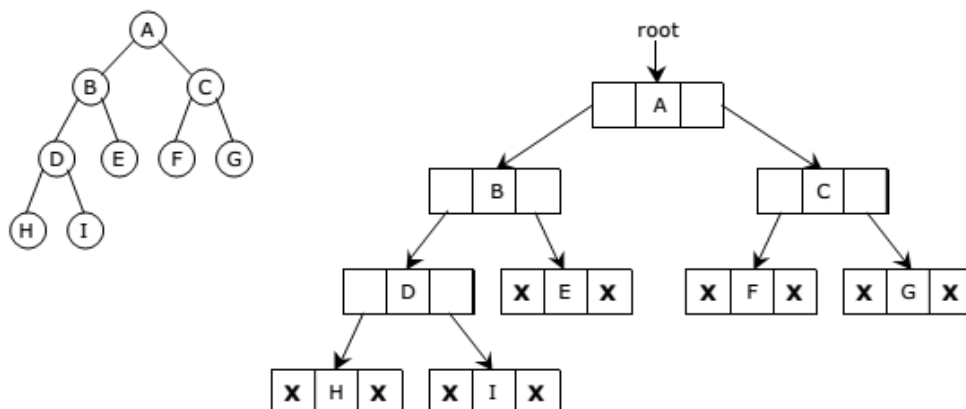
data.txt에서 읽어 만든 트리



1.1 문제 분석

이 문제는 파일의 학생의 정보(학번과 이름)를 입력 받아, 키 값(학번)을 이용해 이진 트리에 각 노드로서 저장하고, 저장된 이진 트리를 전위, 중위, 후위 순으로 출력하는 문제이다.

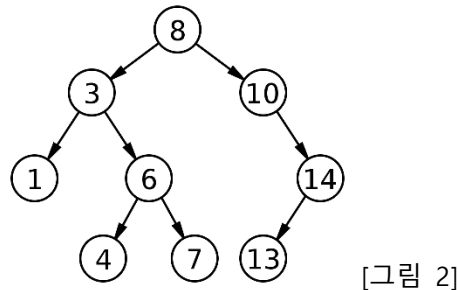
우선 이진 트리에 대해서 살펴보자.



[그림 1]

이진 트리는 위 그림과 같이 모든 노드가 최대 2개의 서브 트리(자식 노드)를 가지는 트리이다. 이번 문제에서는 이진 트리를 '링크 표현법'을 통해 나타내자. 링크 표현법에서는 트리에서의 각

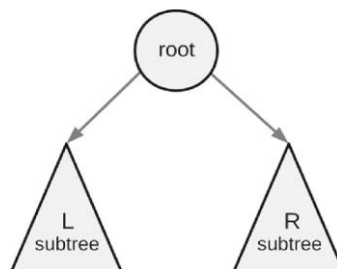
노드가 구조체로 표현되고, 각 구조체는 데이터 필드, 그리고 왼쪽/오른쪽 자식 노드를 가리키는 포인터 2개를 포함한 링크 필드로 구성된다.



[그림 2]

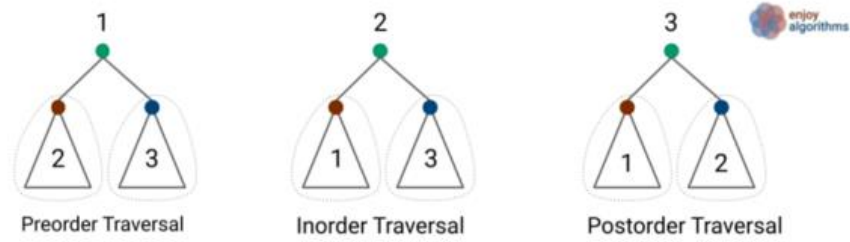
특히, 이진 탐색에 사용되는 **이진 탐색 트리**에서 한 루트 노드의 왼쪽 서브 트리에는 루트 노드의 키 값보다 작은 값들이, 오른쪽 서브 트리에는 루트 노드의 키 값보다 큰 값들이 데이터 필드에 저장되어 있다. 위 그림과 같이, 레벨 1의 루트 노드인 8의 왼쪽 서브 트리에는 모두 8보다 작은 값들이, 그리고 오른쪽 서브 트리에는 모두 10보다 큰 값들이 링크(간선)로 연결되어 있다. 이는 모든 노드들의 서브 트리에도 해당이 된다.

이진 트리 탐색에서 중요한 것은 **탐색** 그리고 **순회**이다.



탐색은, 일반적으로 원하는 값을 트리로부터 찾고자 하는 상황에서도 필요하지만, 이번 문제의 요구사항과 같이 새로운 데이터를 노드로 삽입하고자 할 때에도 매우 중요하다. 위에 설명한 것과 같이, 루트 노드의 데이터보다 작은 값이면 왼쪽 서브 트리에, 큰 값이면 오른쪽 서브 트리에 저장해야 하므로 적절히 각 노드를 순회하며 삽입할 위치를 찾아내야 한다. 예를 들어, 위 [그림 2]에서 숫자 5를 삽입하기 위해서는 8 -> 3 -> 6 -> 4를 따라가며 결국에는 4의 오른쪽 자식 노드에 삽입해야 할 것이다. 만약 숫자 9를 삽입하기 위해서는 8 -> 10을 따라가며 10의 왼쪽 자식 노드에 삽입해야 할 것이다.

이 문제의 해결 과정에서는 순환적인 탐색을 사용한다. 순환의 과정은 데이터의 키 값을 비교하며 서브 트리를 따라 아래로 내려가고, 가장 아래 null값이 탐색되면 해당 위치에 새로운 노드를 동적 할당 받아 삽입하도록 한다.



순회에는 여러가지 종류가 있지만, 이번 문제에서 요구한 바와 같이 전위, 중위, 후위 순회를 살펴보자. 세 가지 순회는 루트 노드를 자식 노드와 비교하여 언제 방문하느냐에 따라 구분된다. 루트를 서브 트리에 앞서 먼저 방문하면 전위 순회(preorder traversal): VLR, 왼쪽과 오른쪽 서브 트리 중간에 방문하면 중위 순회(inorder traversal): LVR, 그리고 서브 트리 방문 후 루트를 방문하면 후위 순회(postorder traversal): LRV라고 한다.

이진 트리의 모든 노드를 메모리 **해제**하고자 할 때는, 간단히 구현할 수 있는 후위 순회를 이용한다. 자식 노드들을 먼저 방문한 뒤 각 노드들의 메모리를 해제하고 다시 루트 노드로 돌아온 뒤 해당 노드의 메모리를 해제하는 과정을 반복하도록 한다.

1.2 소스 코드

```

1  /*
2  →   작성자:..이예빈 (20204059)
3  →   작성일:..2021.09.26
4  →   프로그램명:..파일에 저장되어 있는 학생 데이터를 읽어,
5  →   →   →   학번을 이용해 이진 트리에 저장하고 전위/중위/후위 순으로 출력하는 프로그램
6  */
7
8  #include<stdio.h>
9  #include<stdlib.h>
10 #include<string.h>
11
12 //학생 구조체
13 typedef struct Student{
14     int id;→ //학번 (정수값 key)
15     char* name;→//이름 (문자 포인터형)
16 }Student;
17
18 //트리 노드 구조체
19 typedef struct TreeNode{
20     Student data;
21     struct TreeNode* left, **right;
22 }TreeNode;
23
24 //함수 원형 선언
25 void error(const char* message);
26 TreeNode* new_node(Student data);
27 TreeNode* insert_node(TreeNode* node, Student new_data);
28 void preorder(TreeNode* root);
29 void inorder(TreeNode* root);
30 void postorder(TreeNode* root);
31 void clear(TreeNode* root);
32
33 int main(){
34     FILE* fp;
35     TreeNode* root==NULL; → //이진 탐색 트리의 루트 노드
36     Student tmp; → → → //학생 임시 구조체
37     char name[30]; → → → //파일에서 이름 임시로 읽을 문자열
38
39     //파일 열기
40     fp=fopen("data.txt", "rt");
41     if (fp==NULL) error("파일 열기 실패");
42
43     //파일에서 데이터 읽고, 이진 트리에 삽입
44     while (!feof(fp)) {
45         fscanf(fp, "%d %s", &tmp.id, name);
46
47         tmp.name=(char*) malloc(sizeof(name)); →//문자 배열 공간 메모리 동적 할당
48         if (tmp.name==NULL) error("문자열 동적 할당 실패 \n");
49         strcpy(tmp.name, name); →//문자열 복사
50
51         root=insert_node(root, tmp); →//구조체 전달하여 트리에 노드 삽입
52     }
53
54     //전위, 주위, 후위 출력
55     printf("전위 순회:"); preorder(root); printf("\b\b..\n");
56     printf("중위 순회:"); inorder(root); printf("\b\b..\n");
57     printf("후위 순회:"); postorder(root); printf("\b\b..\n");
58
59     clear(root); → //메모리 해제
60     fclose(fp); → //파일 닫기
61     return 0;
62 }

```

```

63 void error(const char* message) {
64     fprintf(stderr, "%s\n", message);
65     exit(1);
66 }
67
68 //트리의 새로운 학생 노드 동적 할당
69 TreeNode* new_node(Student data) {
70     TreeNode* temp = (TreeNode*) malloc(sizeof(TreeNode));
71     if (temp == NULL) error("동적 할당 실패\n");
72     else {
73         temp->data = data; //학생 구조체를 노드의 데이터 필드에 대입
74         temp->left = temp->right = NULL; //링크 필드는 모두 NULL로 대입 (단말 노드)
75     }
76     return temp;
77 }
78
79 //insert_node 함수 (탐색 후 삽입)
80 TreeNode* insert_node(TreeNode* node, Student new_data) {
81     //삽입할 위치 탐색 후 새로운 노드의 주소 리턴
82     if (node == NULL) return new_node(new_data);
83
84     if (new_data.id < node->data.id) { //key가 더 작은 경우 왼쪽 서브트리 탐색
85         node->left = insert_node(node->left, new_data);
86     }
87     else if (new_data.id > node->data.id) { //key가 더 큰 경우 오른쪽 서브트리 탐색
88         node->right = insert_node(node->right, new_data);
89     }
90     return node; //변경된 루트 포인터 반환
91 }
92
93 //전위 순회
94 void preorder(TreeNode* root) {
95     if (root != NULL) {
96         printf("%d %s -> ", root->data.id, root->data.name);
97         preorder(root->left);
98         preorder(root->right);
99     }
100 }
101 //중위 순회
102 void inorder(TreeNode* root) {
103     if (root != NULL) {
104         inorder(root->left);
105         printf("%d %s -> ", root->data.id, root->data.name);
106         inorder(root->right);
107     }
108 }
109 //후위 순회
110 void postorder(TreeNode* root) {
111     if (root != NULL) {
112         postorder(root->left);
113         postorder(root->right);
114         printf("%d %s -> ", root->data.id, root->data.name);
115     }
116 }
117
118 //후위 순회하며 노드의 메모리 해제
119 void clear(TreeNode* root) {
120     if (root == NULL) return;
121     clear(root->left);
122     clear(root->right);
123     free(root);
124 }

```

1.3 소스 코드 분석

```

/*
→  작성자.:이예빈 (20204059)
→  작성일.:2021.09.26
→  프로그램명.:파일에 저장되어 있는 학생 데이터를 읽어,
→  →  →  학번을 이용해 이진 트리에 저장하고 전위/중위/후위 순으로 출력하는 프로그램
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

```

1. 주석에 작성자, 작성일, 프로그램명을 쓴다. 파일에 저장되어 있는 학생 데이터(학번과 이름)을 읽어, 학번을 key값으로 이용하여 이진 트리에 저장하고, 전위, 중위, 후위 순으로 데이터들을 출력하는 프로그램이다. 이후 필요한 헤더파일들을 추가한다. 표준 입출력, 메모리 동적 할당에 필요한 malloc 함수를 위해 표준 라이브러리, 문자열을 복사하기 위한 strcpy 함수를 위해 string 헤더 파일들을 추가한다.

```

//학생 구조체
typedef struct Student {
→  int id; → //학번 (정수값 key)
→  char* name; → //이름 (문자 포인터형)
} Student;

//트리 노드 구조체
typedef struct TreeNode {
→  Student data;
→  struct TreeNode* left, **right;
} TreeNode;

```

2. 학번(정수형) id와 이름을 저장할 문자 포인터형 name을 멤버로 하는 학생 구조체 Student를 정의한다. 그리고 데이터 필드에는 학생 data를, 링크 필드로는 왼쪽 자식 노드와 오른쪽 자식 노드를 가리킬 자체 참조 구조체 포인터 left, right를 멤버로 하는 트리 노드 구조체 TreeNode를 정의한다.

```

//함수 원형 선언
void error(const char* message);
TreeNode* new_node(Student data);
TreeNode* insert_node(TreeNode* node, Student new_data);
void preorder(TreeNode* root);
void inorder(TreeNode* root);
void postorder(TreeNode* root);
void clear(TreeNode* root);

```

3. 프로그램에서 필요한 함수의 원형을 미리 선언한다. 프로그램 실행 도중 오류가 난 경우 에러 메시지를 출력하고 프로그램을 종료하는 error 함수, 학생 데이터를 바탕으로 새로운 노드를 동적

할당하여 반환하는 new_node 함수, 새로운 노드를 이진 트리에 삽입하는 insert_node 함수, 전위/중위/후위로 이진 트리의 데이터들을 출력하는 preorder, inorder, postorder 함수, 그리고 마지막으로 이진 트리의 모든 노드들을 메모리 해제하는 clear 함수이다.

main 함수의 동작을 보기 전에, 각 함수들이 어떻게 작동하는지 자세히 살펴보도록 하자.

```
void error(const char* message) {
    fprintf(stderr, "%s\n", message);
    exit(1);
}
```

4. 파일 열기 또는 동적 메모리 할당 과정에서 오류가 발생한 경우 에러 문자열을 매개 변수로 받아 출력하고 프로그램을 종료하도록 하는 error 함수이다.

```
//트리의 새로운 학생 노드 동적 할당
TreeNode* new_node(Student data) {
    TreeNode* temp = (TreeNode*) malloc(sizeof(TreeNode));
    if (temp == NULL) error("동적 할당 실패\n");
    else {
        temp->data = data; //학생 구조체를 노드의 데이터 필드에 대입
        temp->left = temp->right = NULL; //링크 필드는 모두 NULL로 대입 (단말 노드)
    }
    return temp;
}
```

5. 트리의 새로운 노드를 동적 할당하는 new_node 함수이다. 학생 데이터를 담은 Student형 구조체를 매개 변수로 받아, 이를 새로운 노드의 데이터 필드에 대입하게 된다.

우선 TreeNode형 temp에 노드 크기만큼의 메모리를 동적 할당한다. 동적 할당에 성공하면, 매개 변수로 받은 학생 구조체의 데이터를 노드의 데이터 필드에 대입하고, 새로운 노드는 항상 단말 노드이기 때문에 자식 노드가 없다는 점을 고려하여, left와 right 링크 필드는 NULL 값으로 초기화한다. 이후 완성된 노드를 반환한다.

```

//.insert_node.함수. (탐색.후.삽입)
TreeNode*.insert_node(TreeNode*.node, Student.new_data){
→ //삽입할.위치.탐색.후.새로운.노드의.주소.리턴
→ if.(node==NULL).return.new_node(new_data);

→ if.(new_data.id<.node->data.id){ //key가.더.작은.경우.왼쪽.서브트리.탐색
→   node->left=.insert_node(node->left, new_data);
→ }
→ else if.(new_data.id>.node->data.id) //key가.더.큰.경우.오른쪽.서브트리.탐색.
→   node->right=.insert_node(node->right, new_data);

→ return node; //변경된.루트.포인터.반환
}

```

6. 이진 트리에 새로운 노드를 삽입하는 insert_node 함수이다. 우선 노드의 루트를 시점으로, 삽입하려는 노드의 id값이 루트 노드의 데이터 필드의 id값(노드의 key값)보다 크다면 오른쪽 서브트리로 이동하고, 작다면 왼쪽 서브 트리로 이동하여 삽입할 위치를 탐색한다. 이를 반복하며 node가 NULL이라면 삽입할 위치를 발견했다는 뜻이므로 해당 위치에 new_node함수를 호출하여 동적 할당한 노드 주소를 반환하여 삽입을 완성한다. 마지막으로 변경된 루트 포인터를 반환한다.

```

//.전위.순회
void preorder(TreeNode*.root){
→ if.(root!=NULL){
→   printf("%d.%s.->", root->data.id, root->data.name);
→   preorder(root->left);
→   preorder(root->right);
→ }
}

```

7. 전위 순회를 하며 데이터를 출력하는 preorder 함수이다. 루트 노드의 데이터를 먼저 출력한 뒤, 루트의 왼쪽 자식 노드와 오른쪽 자식 노드를 preorder 함수로 다시 전달하여 순환을 반복하며 순회를 하는 함수이다.

```

//.중위.순회
void inorder(TreeNode*.root){
→ if.(root!=NULL){
→   inorder(root->left);
→   printf("%d.%s.->", root->data.id, root->data.name);
→   inorder(root->right);
→ }
}

```

8. 중위 순회를 하며 데이터를 출력하는 inorder 함수이다. 먼저 왼쪽 자식 노드를 inorder 함수로 전달하여 루트 노드가 null 값에 도달할 때까지 순환을 반복하고 루트 노드의 데이터를 출력한다. 이후 다시 오른쪽 자식 노드를 동일하게 반복하도록 한다.

```

//.후위.순회
void postorder(TreeNode* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d %s -> ", root->data.id, root->data.name);
    }
}

```

8. 후위 순회를 하며 루트 노드의 데이터를 출력하는 postorder 함수이다. 왼쪽 자식 노드, 오른쪽 자식 노드를 postorder 함수에 인수로 전달하여 순환을 반복하고, 이후 루트 노드의 데이터를 후위에 출력하도록 한다.

```

//.후위.순회하며.노드의.메모리.해제
void clear(TreeNode* root) {
    if (root == NULL) return;
    clear(root->left);
    clear(root->right);
    free(root);
}

```

9. 후위 순회하며 노드의 모든 메모리들을 해제하는 clear 함수이다. 후위 순회를 통해 자식 노드를 방문하여 메모리를 해제하고 다시 루트 노드에 돌아와 해당 노드의 메모리를 해제하게 된다.

다음은 main함수이다.

```

int main() {
    FILE* fp;
    TreeNode* root = NULL; // .이진.탐색.트리의.루트.노드
    Student tmp; // .학생.임시.구조체
    char name[30]; // .파일에서.이름.임시로.읽을.문자열

    // .파일.열기
    fp = fopen("data.txt", "rt");
    if (fp == NULL) error("파일.열기.실패");
}

```

10. 파일 포인터 fp, 그리고 NULL로 초기화된 이진 트리의 루트 포인터 root, 파일에서 데이터를 읽어 임시로 저장할 학생 구조체 변수 tmp, 그리고 파일에서 이름 데이터를 읽을 때 문자열을 저장해 놓을 name 배열을 선언한다.

11. data.txt 파일을 읽기 모드로 읽는다. 파일 열기를 실패한 경우의 코드도 작성한다.

```

→ //파일에서 데이터 읽고, 이진 트리에 삽입
→ while (!feof(fp)) {
→     fscanf(fp, "%d %s", &tmp.id, &name);

→     tmp.name = (char*) malloc(sizeof(name)); → 문자 배열 공간 메모리 동적 할당
→     if (tmp.name == NULL) error("문자열 동적 할당 실패\n");
→     strcpy(tmp.name, name); → 문자열 복사

→     root = insert_node(root, tmp); → 구조체 전달하여 트리에 노드 삽입

```

12. 각 학생의 학번, 그리고 이름이 각 줄에 저장되어 파일을 읽는다. 우선 학번은 tmp 구조체의 id 멤버로 저장하도록 하고, name 문자열에 이름을 저장하도록 한다.

13. 이후 tmp 구조체의 name 포인터 멤버에 파일에서 읽은 name 문자열의 크기만큼의 메모리를 동적 할당하도록 한다. 문자열에 대한 메모리 동적 할당 실패한 경우의 코드도 작성한다. 그리고 나서, name의 문자열을 tmp 구조체의 name에 복사하여 한 학생 데이터를 담은 구조체를 완성하도록 한다.

14. 다음으로 insert_node 함수에 이진 트리의 루트 포인터 root와 완성된 tmp 학생 구조체를 인수로 전달한다. 변경된 루트 포인터를 다시 리턴 받아 root에 저장하도록 한다.

```

→ //전위, 주위, 후위 출력
→ printf("전위 순회:"); preorder(root); printf("\b\b\n");
→ printf("중위 순회:"); inorder(root); printf("\b\b\n");
→ printf("후위 순회:"); postorder(root); printf("\b\b\n");

```

15. 모든 데이터를 파일로부터 읽고 나서, 완성된 이진 트리를 전위/중위/후위 순으로 출력하도록 한다. 루트 포인터 root를 인수로 전달하여 preorder, inorder, postorder을 호출하여 이진 트리를 각각 출력한다. 각 함수를 호출한 뒤에는, 마지막에 출력되는 "->" 문자 2개를 제거하기 위해 backspace 문자를 사용하였다. 백스페이스 문자를 두 개 출력하여 커서를 두 번 앞으로 이동시키고, 공백 space를 두 번 출력하도록 하여 마지막에 출력되는 '-'와 '>'를 각각 화면에 출력되지 않도록 한다.

```

→ clear(root); → //메모리 해제
→ fclose(fp); → //파일 닫기
→ return 0;
}

```

16. 마지막으로 루트 포인터 root를 clear 함수로 전달하여 이진 트리의 모든 노드들을 메모리 해제한다. 파일을 닫고, 프로그램 실행을 종료하도록 한다.

1.4 실행창

```
data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V)
50 이성진
46 이대호
44 김철수
27 루카스
32 펠라인
84 로토스
71 토레스
86 레게로

Microsoft Visual Studio 디버그 콘솔
전위 순회 : 50 이성진 -> 46 이대호 -> 44 김철수 -> 27 루카스 -> 32 펠라인 -> 84 로토스 -> 71 토레스 -> 86 레게로
중위 순회 : 27 루카스 -> 32 펠라인 -> 44 김철수 -> 46 이대호 -> 50 이성진 -> 71 토레스 -> 84 로토스 -> 86 레게로
후위 순회 : 32 펠라인 -> 27 루카스 -> 44 김철수 -> 46 이대호 -> 71 토레스 -> 86 레게로 -> 84 로토스 -> 50 이성진
C:\Users\이예빈\Desktop\CSE 2021-2\자료구조2\code files\자료구조2\실습\20204059-이예빈-실습4주차_1번\Debug\20204059-이예빈
```

1.5 느낀점

이번 1번 문제를 통해 링크로 표현하는 이진 트리의 구조에 대해 확실하게 알게 되었다. 지난번 인덱스를 활용하여 순서대로 노드들을 저장하는 배열 표현법의 이진 트리는 부모 노드와 자식 노드 간의 관계를 사용해야 하는 상황에서는 매우 유용했지만, 새로운 노드를 삽입하거나 삭제해야 하는 경우 매우 번거로워진다는 단점을 지니고 있었다. 이번 링크 표현법의 이진 탐색 트리를 통해서, 트리의 탐색을 통해 새로운 노드의 삽입이 자유로울 수 있다는 점을 새로 배웠다.

또한 이전에 스택에 대해 배울 때 수식 표기법에서 사용된 전위, 중위, 후위를 링크로 표현한 이진 트리의 순회에서 조금 더 명확하게 배울 수 있어서 새로웠다. 이번 문제에서는 순환을 이용하여 전위, 중위, 후위 순회를 하며 노드의 데이터들을 출력하였다. 후에 트리의 데이터 개수가 많고 높이가 더 높아질수록 순환을 함에 따라 스택에 쌓이는 데이터가 많아질 것이므로, 다음 과제에서는 반복을 사용한 순회를 사용하여 프로그램을 작성할 것이다.

2. 완전 이진 트리 검증 프로그램

이진 트리 2

- data.txt에서 정수로 이루어진 정보를 읽어와 이진 트리를 생성하고 생성된 트리가 완전 이진 트리인지 아닌지 검증하시오.
- 파일 data.txt에 저장되어 있는 정보를 사용할 것
- insert_node를 이용하여 트리 생성

data1.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움(H)
8 3 10 14 2 5 9 11 16 4 6

```
C:\WINDOWS\system32\cmd.exe
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Inserted 11
Inserted 16
Inserted 4
Inserted 6
Preorder >> 8
완전 이진 탐색트리가 아닙니다. 계속해
```

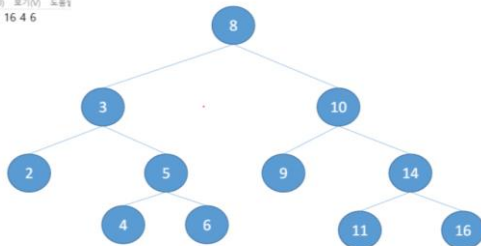
data2.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움(H)
8 3 10 14 2 5 9

```
C:\WINDOWS\system32\cmd.exe
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Preorder >> 8 3 2 5 10 9 14
완전 이진 탐색트리입니다.
계속하려면 아무 키나 누르십시오 . . .
```

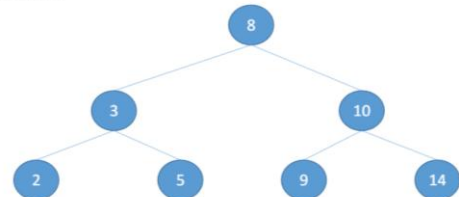
data1.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움(H)
8 3 10 14 2 5 9 11 16 4 6



data2.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움(H)
8 3 10 14 2 5 9



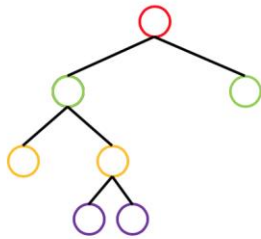
2.1 문제 분석

이 문제는 파일에 저장되어 있는 정수 데이터들로 이진 탐색 트리를 생성하고, 생성된 트리가 완전 이진 트리인지 검증하는 프로그램이다. 파일 입출력, 이진 탐색 트리를 만들고, 새로운 노드를 동적 할당 받아 삽입 시에 삽입 위치를 탐색하여 삽입하는 것, 그리고 트리 노드들의 메모리 해제는 1번 이진 트리 문제와 그 해결 방식이 동일하다.

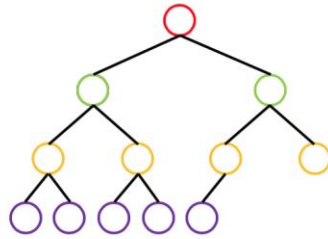
이번 문제에서 주목해야 부분은 완전 이진 탐색 트리를 판별하는 과정이다.

우선 완전 이진 트리가 무엇인지부터 살펴보자.

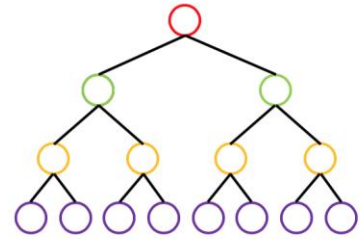
정 이진 트리 Full binary tree
적정 이진 트리 Proper binary tree



완전 이진 트리 Complete binary tree



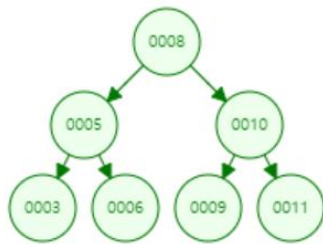
포화 이진 트리 Perfect binary tree



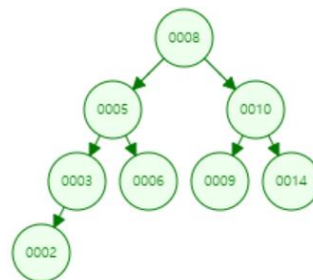
<https://sean-ma.tistory.com>

완전 이진 트리는 마지막 레벨을 제외한 모든 레벨에 노드들이 모두 채워져 있으며, 마지막 레벨에는 노드들이 왼쪽부터 차례대로 채워져 있는 트리이다. 위의 그림에서 두번째, 세번째 트리가 완전 이진 트리이다. 특히 세번째 트리는 마지막 레벨의 노드들이 모두 채워진 포화 이진 트리라 불린다.

다음의 몇 가지 이진 트리 예시를 보면서, 조금 더 명확하게 완전 이진 트리 여부를 판별해보자.

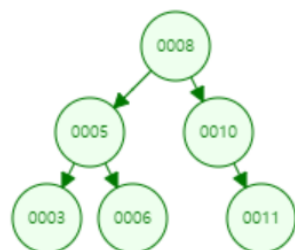


[그림 1]

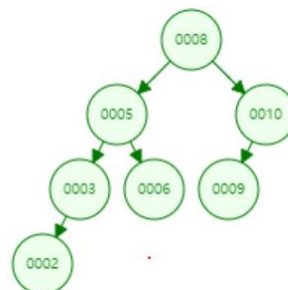


[그림 2]

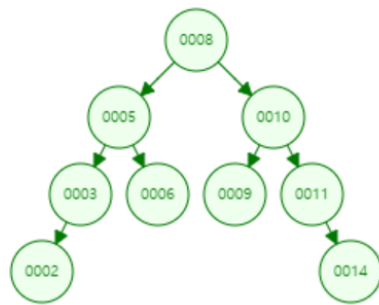
[그림 1]과 같은 경우는 완전 이진 트리 중에서도, 마지막 3레벨까지 모든 노드들이 채워진 포화 이진 트리이다. [그림 2]는 레벨3까지 모든 노드들이 채워져 있으며, 마지막 4레벨에는 맨 왼쪽에 노드가 레벨3의 첫번째 노드의 왼쪽 자식 노드로 붙어 있으므로, 이 또한 완전 이진 트리라고 할 수 있다.



[그림 3]



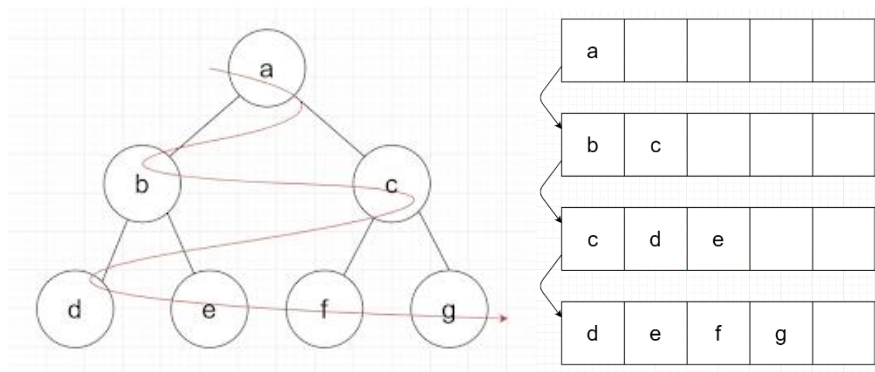
[그림 4]



[그림 5]

위의 그림 3개는 완전 이진 트리가 아닌 트리들의 예시이다. [그림 3]의 경우 마지막 레벨 3은 왼쪽부터 차례대로 노드가 채워져 있는 것이 아니므로 완전 이진 트리의 조건에 부합하지 않는다. [그림 4]의 경우, 레벨 3의 모든 노드들이 채워져 있지 않은 상태로 레벨 4에 노드가 존재하므로 이 또한 완전 이진 트리가 아니다. 마지막 그림 5]의 경우도 [그림 3]과 마찬가지로, 마지막 레벨 4에 있는 마지막 노드로 인해 완전 이진 트리라 할 수 없다.

완전 이진 트리의 정의에 따라, 트리의 '레벨' 개념이 매우 중요하다. 마지막 레벨을 제외한 나머지 레벨의 노드들이 모두 채워져 있는지, 마지막 레벨이 모두 채워져 있는 것이 아니라면, 마지막 레벨의 노드들이 왼쪽부터 차례대로 채워져 있는지를 검증하여 완전 이진 트리임을 판별해야 한다. 따라서 이 문제에서는 큐 배열 구조를 활용한 레벨 순회를 통해 각 노드들을 레벨별로 왼쪽부터 순회하면서 완전 이진 트리의 조건에 부합하는지 판별하도록 프로그램을 작성하는 것으로 하자.



레벨 순회는 위 그림에 나타난 것처럼 이진 트리를 레벨 1의 루트 노드부터 시작하여, 각 레벨의 모든 노드들을 왼쪽부터 차례대로 순회한다. 레벨 순회를 위해서는, 일반적인 큐 배열 구조가 사용된다. 레벨 순회를 하며 '완전 이진 트리'를 판별하기 위해서는 순회를 하며, 각 노드의 자식 노드의 상태를 확인하는 과정이 필요하다.

레벨 순회는 루트 노드를 큐에 삽입함으로 시작된다. 이후, 루트 노드를 큐로부터 꺼냄과 동시에 루트 노드의 왼쪽 자식 노드와 오른쪽 자식 노드가 존재하는 경우 각각 차례대로 큐에 삽입한다. 큐에 삽입함으로써, 다음 레벨의 순회 때 삽입된 이 자식 노드들을 이용할 수 있게 되는 것이다. 큐에서 다시 노드를 꺼내어 다시 자식 노드들을 삽입할 때, 완전 이진 트리의 정의에 따라 왼쪽 자식 노드 오른쪽 자식 노드의 여부에 따라 각 조건을 살펴보면 완전 이진 트리의 조건에 부합하는지 판단하도록 한다.

큐로부터 노드를 꺼낸 후 다음 조건들을 거치게 된다.

1. 왼쪽 자식 노드가 없는 경우
 - A. 오른쪽 자식 노드가 있는 경우 -> 완전 이진 트리 X
 - B. 오른쪽 자식 노드가 없는 경우
 - i. 큐에 삽입되어 있는 모든 노드들이 단말 노드인지 판별 -> 단말 노드가 아닌 노드가 발견되면 완전 이진 트리 X
 - ii. (추가) - 또는 큐에 현재까지 삽입한 노드의 개수가 기존 노드 개수와 동일한지 판별 -> 동일하다면 완전 이진 트리 O
2. 왼쪽 자식 노드가 있지만, 오른쪽 자식 노드가 없는 경우
 - A. 큐에 삽입되어 있는 모든 노드들이 단말 노드인지 판별 -> 단말 노드가 아닌 노드가 발견되면 완전 이진 트리 X
 - B. (추가) - 또는 큐에 현재까지 삽입한 노드의 개수가 기존 노드 개수와 동일한지 판별 -> 동일하다면 완전 이진 트리 O

단말 노드는 왼쪽/오른쪽 서브 트리가 모두 null값인 노드이다. 단말 노드를 확인하는 과정은 마지막 레벨 전 레벨의 모든 노드들이 채워져 있는지, 그리고 마지막 레벨의 노드들이 모두 왼쪽부터 채워져 있는지를 조사하기 위해서는 단말 노드 여부를 판별하는 것이 매우 중요하다.

(추가) - 단말 노드를 확인하는 방법도 있지만, 현재까지 큐에 삽입한 노드의 개수가 이진 트리에 있는 전체 노드 개수와 동일한 경우를 확인하여 완전 이진 트리를 판별할 수도 있다. 위의 조건과 같이 왼쪽 자식 노드가 더 이상 발견되지 않는 경우, 완전 이진 트리이라면 마지막 레벨의 마지막 노드까지 모두 삽입했다는 것이다. 왼쪽 자식 노드는 있지만, 오른쪽 자식 노드가 없는 경우도 마찬가지로, 완전 이진 트리일 경우라면 가장 마지막 레벨의 마지막 노드까지 모두 삽입한 경우인 것이므로, 큐에 삽입한 노드의 개수들이 트리의 전체 노드 개수와 동일한 것으로 완전 이진 트리를 판별해 볼 수도 있다. (이는 2.3.1 소스코드 분석 2에 추가적으로 작성해 놓았다.)

2.2 소스 코드

```

1  /*
2  →  작성자.: 이예빈 (20204059)
3  →  작성일.: 2021.09.27
4  →  프로그램명.: 파일의 정수들을 이용해 이진 탐색 트리를 만들고,
5  →  →  →  레벨 순회를 하며 완전 이진 탐색 트리인지 판별하는 프로그램
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #define MAX_QUEUE_SIZE 100
11 #define FALSE 0
12 #define TRUE 1
13
14 typedef struct TreeNode { →  →  // TreeNode: 트리의 노드
15     int data;
16     struct TreeNode* left, *right;
17 }TreeNode;
18
19 typedef TreeNode* element; →  →  // element: 트리 노드 포인터
20 typedef struct { →  →  →  →  // QueueType: 트니 도드 포인터에 대한 큐 배열
21     element data[MAX_QUEUE_SIZE];
22     int front, rear;
23 }QueueType;
24
25 // 함수 원형 선언
26 void error(char* message);
27 int is_empty(QueueType* q);
28 int is_full(QueueType* q);
29 void enqueue(QueueType* q, element item);
30 element dequeue(QueueType* q);
31 TreeNode* new_node(int item);
32 TreeNode* insert_node(TreeNode* node, int key);
33 int is_leaf(TreeNode* node);
34 int isComplete(TreeNode* ptr);
35 void clear(TreeNode* ptr);
36
37 int main() {
38     FILE* fp;
39     TreeNode* root = NULL;
40     int tmp;
41
42     // 파일 열기
43     fp = fopen("data.txt", "rt");
44     if (fp == NULL) error("파일 열기 실패");
45
46     // 파일로부터 정수 읽어 트리 구조 완성
47     while (!feof(fp)) {
48         fscanf(fp, "%d", &tmp);
49         root = insert_node(root, tmp);
50     }
51
52     // 완전 이진 탐색 트리인지 판별
53     if (isComplete(root)) printf("완전 이진 탐색 트리입니다.\n");
54     else printf("완전 이진 탐색 트리가 아닙니다.\n");
55
56     clear(root); →  // 메모리 해제
57     fclose(fp); →  // 파일 닫기
58     return 0;
59 }
60

```

```

61 void error(char* message) {
62     fprintf(stderr, "%s\n", message);
63     exit(1);
64 }
65 void init_queue(QueueType* q) { q->front == q->rear == 0; }
66 int is_empty(QueueType* q) { return (q->front == q->rear); }
67 int is_full(QueueType* q) { return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front); }
68
69 void enqueue(QueueType* q, element item) {
70     if (is_full(q))
71         error("큐가 포화상태입니다.");
72     q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
73     q->data[q->rear] = item;
74 }
75
76 element dequeue(QueueType* q) {
77     if (is_empty(q))
78         error("큐가 공백상태입니다.");
79     q->front = (q->front + 1) % MAX_QUEUE_SIZE;
80     return q->data[q->front];
81 }
82
83 // 트리의 새로운 노드 동적 할당
84 TreeNode* new_node(int item) {
85     TreeNode* temp = (TreeNode*) malloc(sizeof(TreeNode));
86     temp->data = item;
87     temp->left = temp->right = NULL;
88     return temp;
89 }
90
91 // 이진 탐색트리에서 탐색 후 특정 위치에 새로운 key를 갖는 노드 삽입
92 TreeNode* insert_node(TreeNode* node, int key) {
93     // 탐색에 성공하여 새로운 노드의 주소를 반환
94     if (node == NULL) {
95         printf("Inserted %d\n", key);
96         return new_node(key);
97     }
98
99     // 새로운 노드를 삽입할 위치 탐색
100     if (key < node->data)
101         node->left = insert_node(node->left, key);
102     else if (key > node->data)
103         node->right = insert_node(node->right, key);
104
105     return node; // 변경된 루트 포인터 반환
106 }
107
108 // 단말 노드 판별
109 int is_leaf(TreeNode* node) {
110     // 왼쪽/오른쪽 자식 노드가 모두 NULL인 경우 단말 노드
111     if (node->left == NULL && node->right == NULL) return TRUE;
112     else return FALSE;
113 }
114

```

```

115 // 레벨 순회를 하며 완전 이진 탐색 트리인지 판별
116 int isComplete(TreeNode* ptr) {
117     QueueType q;
118     init_queue(&q);
119
120     if (ptr == NULL) return;
121
122     enqueue(&q, ptr);
123     while (!is_empty(&q)) {
124         ptr = dequeue(&q);
125
126         if (ptr->left)
127             enqueue(&q, ptr->left);
128         if (ptr->right)
129             enqueue(&q, ptr->right);
130
131         // 왼쪽 자식 노드가 없는 경우
132         if (ptr->left == NULL) {
133             // 왼쪽 자식 노드가 없으면서, 오른쪽 자식 노드는 있는 경우
134             if (ptr->right) return FALSE;
135
136             // 큐에 있는 나머지 노드들이 단말 노드인지 확인하기
137             else {
138                 while (!is_empty(&q)) {
139                     ptr = dequeue(&q);
140                     if (is_leaf(ptr) == 0) return FALSE;
141                 }
142                 return TRUE;
143             }
144         }
145         // 왼쪽 자식 노드는 있지만, 오른쪽 자식 노드가 없는 경우
146         if (ptr->left && ptr->right == NULL) {
147             // 큐에 있는 나머지 노드들이 단말 노드인지 확인하기
148             while (!is_empty(&q)) {
149                 ptr = dequeue(&q);
150                 if (is_leaf(ptr) == FALSE) return FALSE;
151             }
152             return TRUE;
153         }
154     }
155 }
156
157 // 후위 순회하며 노드의 메모리 해제
158 void clear(TreeNode* root) {
159     if (root == NULL) return;
160     clear(root->left);
161     clear(root->right);
162     free(root);
163 }

```

2.3 소스 코드 분석

```

/*
 * 작성자: .이예빈 (20204059)
 * 작성일: .2021.09.27
 * 프로그램명: .파일의.정수들을.이용해.이진.탐색.트리를.만들고,
 *           → → → 레벨.순회를.하며.완전.이진탐색.트리인지.판별하는.프로그램
 */

```

```

#include<stdio.h>
#include<stdlib.h>
#define MAX_QUEUE_SIZE 100
#define FALSE 0
#define TRUE 1

```

1. 작성자, 작성일, 프로그램명을 주석에 쓴다. 이 프로그램은 파일의 정수들을 읽어 이진 탐색 트리를 만들고, 해당 트리가 이진탐색 트리인지 판별하는 프로그램이다. '레벨 순회'를 통해 이진탐색 트리를 구현하는 프로그램이다.

2. 필요한 표준 입출력, 표준 라이브러리 헤더 파일을 추가한다. 레벨 순회에 필요한 큐 배열을 위해, 최대 큐의 배열 크기인 MAX_QUEUE_SIZE 를 100 으로 정의하고, 코드의 가독성을 위해 FALSE 와 TRUE 를 각각 0 과 1 로 정의하도록 한다.

```

typedef struct TreeNode { → → //TreeNode:트리의.노드
    int data;
    struct TreeNode* left, *right;
}TreeNode;

typedef struct { → → //element:트리.노드.포인터
    element data[MAX_QUEUE_SIZE];
    int front, rear;
}QueueType;

```

2. 트리 노드의 구조체 TreeNode 는 데이터 필드로는 정수형 변수 data 를, 그리고 링크 필드로는 자체 참조 구조체 포인터인 left 와 right 을 갖는다.

이 트리 노드의 포인터를 element 로 정의한다. 그리고, MAX_QUEUE_SIZE(100) 크기의 element 형 배열 data, 그리고 큐 배열의 전단과 후단을 의미하는 정수형 front 와 rear 을 멤버로 갖는 큐 구조 QueueType 을 선언한다.

```

//.함수.원형.선언
void error(char* message);
int is_empty(QueueType* q);
int is_full(QueueType* q);
void enqueue(QueueType* q, element item);
element dequeue(QueueType* q);
TreeNode* new_node(int item);

TreeNode* insert_node(TreeNode* node, int key);
int is_leaf(TreeNode* node);
int isComplete(TreeNode* ptr);
void clear(TreeNode* ptr);

```

3. 프로그램에서 사용되는 함수들의 원형을 선언한다. 오류 메시지를 출력하고 프로그램을 종료하는 error 함수, 큐의 공백 상태를 판별하는 is_empty, 큐의 포화 상태를 판별하는 is_full, 큐에 요소를 삽입하는 enqueue, 큐로부터 요소를 삭제하여 반환하는 dequeue 함수까지 레벨 순회에 필요한 큐 구조의 입출력 함수들이 있다.

4. 새로운 노드를 동적 할당하는 new_node 함수, 새로운 노드를 이진탐색 트리에 삽입하는 insert_node 함수, 노드가 단말 노드인지 판별하는 is_leaf 함수, 트리가 완전 이진탐색 트리인지 판별하는 isComplete, 마지막으로 트리의 모든 노드를 메모리 해제하는 clear 함수이다.

우선, main 함수의 동작을 살펴보기 전에 프로그램에서 사용되는 함수들을 살펴보도록 하자.

```

void error(char* message){
    → fprintf(stderr, "%s\n", message);
    → exit(1);
}

void init_queue(QueueType* q){ q->front = q->rear = 0; }
int is_empty(QueueType* q){ return (q->front == q->rear); }
int is_full(QueueType* q){ return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front); }

void enqueue(QueueType* q, element item){
    → if (is_full(q))
    →     error("큐가 포화상태입니다.");
    → q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    → q->data[q->rear] = item;
}

element dequeue(QueueType* q){
    → if (is_empty(q))
    →     error("큐가 공백상태입니다.");
    → q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    → return q->data[q->front];
}

```

4. 오류 메시지를 출력하고 프로그램 실행을 종료하는 error 함수, 그리고 위에서 언급했던 큐 배열의 입출력 함수들이다. 이 함수들은 간단히 살펴보고 가자.

5. 큐 구조는 `init_queue` 를 통해서, 전단인 `front` 와 후단인 `rear` 이 각각 배열의 인덱스 0 번을 가리키는 것으로 초기화된다. 큐의 전단과 후단이 같은 곳을 가리키는 경우라면 공백 상태를 의미하고, `rear` 이 `front` 보다 하나 앞를 가리키고 있다면, 포화 상태임을 의미한다. `enqueue` 를 통해 `rear` 의 인덱스가 앞으로 이동하며 요소를 해당 위치에 삽입하게 되고, `dequeue` 를 통해 `front` 의 인덱스가 하나 앞으로 이동하며 가리키게 되는 곳의 요소를 반환하게 된다.

```
//.트리의.새로운.노드.동적.할당
TreeNode* new_node(int item) {
    →   TreeNode* temp = (TreeNode*) malloc(sizeof(TreeNode));
    →   temp->data = item;
    →   temp->left = temp->right = NULL;
    →   return temp;
}
```

6. 삽입하고자 하는 정수 데이터 `item` 을 매개 변수로 전달받아, 트리의 새로운 노드를 동적 할당하여 반환하는 `new_node` 함수이다. 우선 `TreeNode` 형 포인터 `temp` 에 노드의 크기만큼의 메모리를 동적 할당 받는다. 데이터 필드에는 매개 변수로 전달받은 `item` 을 대입하고, 링크 필드인 `left` 와 `right` 은 각각 `NULL` 을 초기화하여 하나의 단말 노드를 완성시킨다. 그리고 나서 완성한 노드 `temp` 를 반환한다.

```
//.이진.탐색트리에서.탐색.후.특정.위치에.새로운.key를.갖는.노드.삽입
TreeNode* insert_node(TreeNode* node, int key) {
    →   //.탐색에.성공하여.새로운.노드의.주소를.반환
    →   if (node == NULL) {
    →       →   printf("Inserted %d\n", key);
    →       →   return new_node(key);
    →   }

    →   //.새로운.노드를.삽입할.위치.탐색
    →   if (key < node->data)
    →       →   node->left = insert_node(node->left, key);
    →   else if (key > node->data)
    →       →   node->right = insert_node(node->right, key);

    →   return node; //.변경된.루트.포인터.반환
}
```

7. 이진 탐색 트리에 정수형 `key` 를 데이터 필드로 하는 새로운 노드를 삽입하는 함수이다. 이진 트리의 특성에 맞게, 루트 노드부터 시작하여 각 루트 노드의 `data` 보다 `key` 값이 작으면 왼쪽 자식 노드로 이동, 그렇지 않다면 오른쪽 자식 노드로 이동하여 삽입할 위치를 탐색한다. 탐색을 반복하여 `node` 가 `NULL` 값인 경우면, 탐색에 성공한 것이므로 `new_node` 함수를 호출하여 새로 동적 할당한 노드의 주소를 탐색에 성공한 위치에 대입하도록 한다.


```

//.단말.노드.판별
int is_leaf(TreeNode* node) {
    → //.왼쪽/오른쪽.자식.노드가.모두.NULL인경우.단말노드
    → if (node->left == NULL && node->right == NULL) return TRUE;
    → else return FALSE;
}

```

8. 노드가 단말 노드인지 판별하는 is_leaf 함수이다. 해당 노드의 왼쪽, 오른쪽 자식 노드가 모두 NULL 값이라면 TRUE 를, 그렇지 않다면 FALSE 를 반환한다.

이제 다음은 이번 프로그램에서 가장 핵심적인 isComplete 함수를 살펴보자.

isComplete 함수는 이진 탐색 트리를 레벨 1의 루트 노드부터 시작하여, 각 레벨을 순회하며 완전 이진 탐색 트리인지 판별하는 함수이다. 코드를 자세히 살펴보자.

```

//.레벨.순회를.하며.완전.이진탐색.트리인지.판별
int isComplete(TreeNode* ptr) {
    → QueueType q;
    → init_queue(&q);

    → if (ptr == NULL) return;

    → enqueue(&q, ptr);
    → while (!is_empty(&q)) {
        → ptr = dequeue(&q);

        → if (ptr->left)
        →     enqueue(&q, ptr->left);
        → if (ptr->right)
        →     enqueue(&q, ptr->right);
    }
}

```

8. 이진 트리의 루트 노드를 매개 변수 ptr 로 전달받는다. 레벨 순회에 필요한 큐 배열 구조 QueueType 형 변수 q 를 선언하고, init_queue 를 호출하여 배열을 초기화 시킨다. ptr 이 NULL 값이라면 함수를 종료하도록 한다.

9. 우선 q 배열에 루트 노드를 삽입한다. 그리고 나서, 큐에 삽입되어 있는 노드를 dequeue 함과 동시에, 그 노드의 자식 노드들을 왼쪽 노드부터 차례대로 큐에 삽입하도록 한다. 자식 노드들을 삽입한다는 것은, 다음 레벨에 있는 노드들을 모두 큐에 삽입하는 것과 동일하며, 이는 다음 레벨 순회에 왼쪽 노드부터 시작하여 차례대로 모든 노드들을 순회할 수 있게 하는 것이다. 이 과정을 큐가 공백일 때까지, 즉 모든 레벨의 모든 노드들을 순회할 수 있게 반복하도록 한다.

다음은 큐로부터 dequeue 한 ptr 노드를 이용하여 완전 이진탐색 트리인지 판별하는 조건문들이다. 각 조건을 자세하게 살펴보도록 하자. 핵심은 ptr 노드의 왼쪽 자식 노드와 오른쪽 자식 노드의 존재 여부, 그리고 큐에 남아있는 다른 노드들의 자식 노드들을 살펴보는 것이다.

```

→ //왼쪽.자식.노드가.없는.경우
→ if (ptr->left==NULL)·{
→     //왼쪽.자식.노드가.없으면서,·오른쪽.자식.노드는.있는.경우
→     if (ptr->right)·return·FALSE;

→     //큐에.있는.나머지.노드들이.단말.노드인지.확인하기
→     else·{
→         while·(!is_empty(&q))·{
→             ptr·=·dequeue(&q);
→             if·(is_leaf(ptr)·==·FALSE)·return·FALSE;
→         }
→         return·TRUE;
→     }
→ }

```

우선 ptr의 왼쪽 자식 노드가 없는 경우이다.

10. 첫번째의 경우는, 왼쪽 자식 노드가 없으면서, 오른쪽 자식 노드가 있는 경우이다. 이러한 경우는 왼쪽 자식 노드부터 채워져야 하는 완전 이진 탐색 트리의 조건과 부합하지 않으므로 바로 FALSE를 반환하도록 한다.

아래 [그림 1]에서, 레벨 2의 두번째 노드는 오른쪽 자식 노드만 갖고 있으므로 이 트리는 완전 이진 탐색 트리의 조건에 부합하지 않는다.)

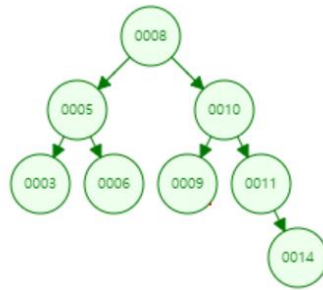


[그림 1]

11. 두번째의 경우는, ptr의 오른쪽 자식 노드도 왼쪽 자식 노드와 마찬가지로 없는 경우이다. 이때에는, 큐에 삽입되어 있는 노드들, 즉 ptr 노드와 같은 레벨에 있는 오른쪽 노드들의 자식 노드 여부를 판별하는 것이다.

다음 [그림 2]에서, 레벨 3의 첫번째 노드가 그 예이다. 왼쪽, 오른쪽 자식 노드가 모두 없는 상황에서, 레벨 3에 있는 나머지 3개의 노드들이 모두 단말 노드가 아닌지 판별이 필요하다.

레벨 3의 마지막 노드는 단말 노드가 아니므로, 이 트리는 완전 이진탐색 트리의 조건에 부합하지 않음을 확인할 수 있다.



[그림 2]

```

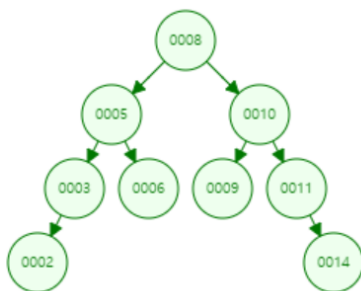
→ //왼쪽.자식.노드는.있지만,.오른쪽.자식.노드가.없는.경우
→ if(ptr->left.&&ptr->right==NULL){
→ //큐에.있는.나머지.노드들이.단말.노드인지.확인하기.
→ while(!is_empty(&q)){
→ ptr=>dequeue(&q);
→ if(is_leaf(ptr)==FALSE)·return·FALSE;
→ }
→ return·TRUE;
→ }

```

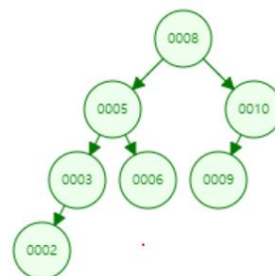
12. 이번에는, 왼쪽 자식 노드는 존재하지만, 오른쪽 자식 노드가 없는 경우이다. 이러한 경우에도 큐에 삽입되어 있는 나머지 노드들이 단말 노드인지 판별할 필요가 있다.

다음의 [그림 3]에서, 레벨 3의 첫번째 노드는 왼쪽 자식 노드가 있지만 오른쪽 자식 노드가 없다. 이 때, 큐에 삽입되어 있는 노드들(레벨 3의 오른쪽 노드들)을 모두 꺼내어 단말 여부를 확인한다. 레벨 3의 맨 오른쪽 노드가 단말 노드가 아니므로, 이 트리는 완전 이진 탐색 트리의 조건에 부합하지 않게 된다.)

[그림 4]에서는 레벨 2의 두번째 노드의 오른쪽 자식 노드가 없는 경우이다. 따라서 큐에 삽입되어 있는 레벨 3의 노드들이 단말 노드인지 확인할 필요가 있다. 레벨 3의 첫번째 노드가 단말 노드가 아니므로, 완전 이진 탐색 트리가 아닌 것으로 판별한다.



[그림 3]



[그림 4]

```
//·후위·순회하며·노드의·메모리·해제
void·clear(TreeNode*·root)·{
    if·(root·==·NULL)·return;
    clear(root->left);
    clear(root->right);
    free(root);
}
```

13. 마지막으로 후위 순회를 하며 트리의 모든 노드를 메모리 해제하는 함수이다. 루트 노드의 왼쪽 자식 노드 left 를 인수로 하여 clear 함수를 순환하여 호출하고, 다음으로 right 을 인수로 하여 순환 호출한다. 그리고 해당 루트 노드의 메모리를 해제하도록 한다.

이는 자식 노드들부터 우선 순회하여 메모리를 해제한 뒤 다시 루트 노드로 이동하는 후위 순회를 이용한 함수이다.

이제 마지막으로 main 함수의 동작을 살펴보자.

```
int·main()·{
    FILE*·fp;
    TreeNode*·root·==·NULL;
    int·tmp;

    //·파일·열기
    fp·=·fopen("data.txt",·"rt");
    if·(fp·==·NULL)·error("파일·열기·실패");
}
```

14. 파일 포인터 fp, NULL 값으로 초기화된 이진 탐색 트리의 루트 노드 포인터 root, 그리고 파일로부터 읽는 정수를 임시로 저장할 tmp 를 선언한다.

그리고 파일을 연다. 파일 열기를 실패한 경우의 코드도 포함한다.

```
//·파일로부터·정수·읽어·트리·구조·완성
while·(!feof(fp))·{
    fscanf(fp,·"%d",·&tmp);
    root·=·insert_node(root,·tmp);
}
```

15. 파일로부터 정수를 읽을 때마다 tmp 에 저장하고, insert_node 함수에 트리의 루트 노드 포인터 root 와 tmp 를 인수로 전달하여 새 노드를 트리에 삽입하도록 한다. 변경된 루트 포인터를 반환 받아 root 에 다시 대입한다.

```

→ //·완전·이진탐색·트리인지·판별
→ if·(isComplete(root))·printf("완전·이진탐색·트리입니다·.\n");
→ else·printf("완전·이진탐색·트리가·아닙니다·.\n");

→ clear(root); → //·메모리·해제
→ fclose(fp); → //·파일·닫기
→ return·0;
}

```

16. isComplete 함수에 root 포인터를 인수로 전달하여, 해당 이진 탐색 트리가 완전 이진 탐색 트리인지 판별하도록 한다. isComplete 에서 TRUE 를 반환하면, 완전 이진탐색 트리이다. 그리고 트리의 모든 노드를 메모리 해제하고, 파일을 닫고 프로그램의 전체 실행을 종료한다.

2.3.2 (추가) 큐에 삽입한 노드의 개수를 이용하는 방법

위에서 사용한 방법은 큐에 삽입한 노드들의 단말 여부를 통해서, 마지막 레벨이 아닌 레벨에 모든 노드들이 꽉 채워져 있는지, 그리고 마지막 레벨에 노드가 존재한다면 왼쪽부터 채워져 있는지 여부를 판별해보고자 했다.

위 방법은 나머지 노드들의 단말 여부를 하나하나 살펴봐야 한다는 점에서 조금은 비효율적이다. 이번에 사용하는 방법은 조금 더 간단한 방법이다. 바로 트리의 전체 노드 개수를 미리 세고, 큐에 노드를 삽입한 횟수를 세어 완전 이진 트리의 여부를 판별해야 하는 상황에서 그 횟수가 전체 트리의 노드 개수와 일치하는지 판별하는 방법이다. 즉, 더이상 자식 노드가 발견되지 않아 마지막 레벨로 추정되는 상황에서, 트리에 노드를 삽입한 횟수가 이진 트리의 전체 노드의 개수가 일치하는지 확인을 하는 방법이다.

변경된 소스 코드를 살펴보자.

다음과 같이 함수를 정의할 때, 단말 노드의 여부를 판별하는 is_leaf 함수는 주석 처리하도록 한다. 그리고 isComplete 함수는 루트 노드 포인터 뿐 아니라, 두번째 매개 변수로 전체 트리의 노드 개수를 전달받는다.

```
//.int.is_leaf(TreeNode*.node);
int.isComplete(TreeNode*.ptr, .int.node_count);
```

다음은 main 함수에서 변경된 부분이다. 트리의 노드 개수를 셀 정수형 변수 node_count 를 선언하고 0 으로 초기화한다. 파일에서 정수를 읽을 때마다 node_count 를 증가시키도록 한다. isComplete 함수를 호출할 때에는 node_count 를 두번째 인수로 전달한다.

```
int.tmp, .node_count.=0;

//.파일.열기
fp.=fopen("data.txt", ".rt");
if.(fp==.NULL).error("파일.열기.실패");

//.파일로부터.정수.읽어.트리.구조.완성
while.(!feof(fp)).{
→ fscanf(fp, "%d", .&tmp);
→ node_count++;
→ root.=insert_node(root, .tmp);
}

//.완전.이진탐색.트리인지.판별
if.(isComplete(root, .node_count)).printf("\n완전.이진탐색.트리입니다..\n");
else.printf("\n완전.이진탐색.트리가.아닙니다..\n");
```

다음으로, 일부 변경된 isComplete 함수를 살펴보자.

```
//.레벨.순회를.하며.완전.이진탐색.트리인지.판별
int isComplete(TreeNode* ptr, int node_count) {
    QueueType q;
    init_queue(&q);
    int count = 1;

    if (ptr == NULL) return;

    enqueue(&q, ptr);
    while (!is_empty(&q)) {
        ptr = dequeue(&q);

        if (ptr->left) {
            enqueue(&q, ptr->left);
            count++;
        }
        if (ptr->right) {
            enqueue(&q, ptr->right);
            count++;
        }

        //.왼쪽.자식.노드가.없는.경우
        if (ptr->left == NULL) {
            //.왼쪽.자식.노드가.없으면서,.오른쪽.자식.노드는.있는.경우
            if (ptr->right) return FALSE;
            //.왼쪽.자식.노드가.없고,.오른쪽.자식.노드.또한.없는.경우
        } else {
            //.그동안.큐에.삽입한.노드의.개수가.트리의.노드.개수와.동일하다면.
            if (node_count == count) return TRUE;
            else return TRUE;
        }

        //.왼쪽.자식.노드는.있지만,.오른쪽.자식.노드가.없는.경우
        if (ptr->left && ptr->right == NULL) {
            //.그동안.큐에.삽입한.노드의.개수가.트리의.노드.개수와.동일하다면.
            if (node_count == count) return TRUE;
            else return FALSE;
        }
    }
}
```

우선 맨 처음 루트 노드를 큐에 삽입하는 경우를 위해, int 형 변수 count 는 1 로 초기화하여 선언한다. 그리고 나서, 노드를 큐에서 dequeue 하여 해당 노드의 자식 노드들을 삽입할 때마다 count 의 값을 증가시킨다. 이를 통해 순회하게 되는 노드의 개수를 알 수 있게 된다.

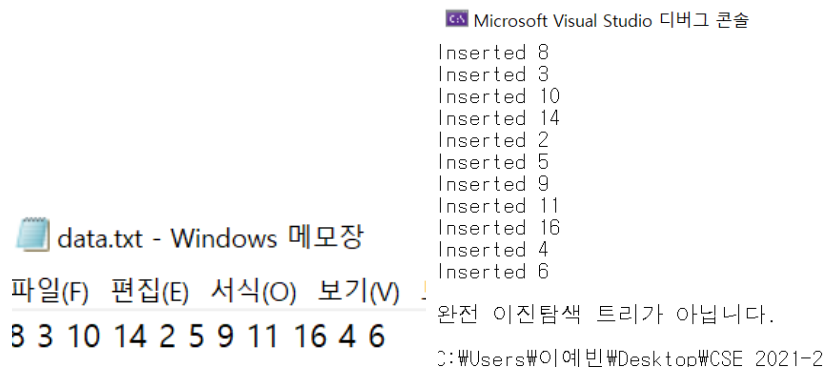
완전 이진 트리를 판별하는 조건은 첫번째 방식과는 동일하다. 다만, 이진 탐색 트리의 여부를 판별해야 하는 상황에서 큐에 남아있는 모든 노드들의 단말 여부를 판별하는 첫번째 방식과 다르게, 마지막 레벨의 마지막 노드인지 판별하는 상황에서 count 의 값과 node_count 의 값을

2021-09-27

비교하여, 그 값이 같다면 완전 이진 트리를 성사시키지 않는 문제의 추가 노드가 없음을 의미하게 `hel` 는 것이다.

2.4 실행창

예시 결과창 1

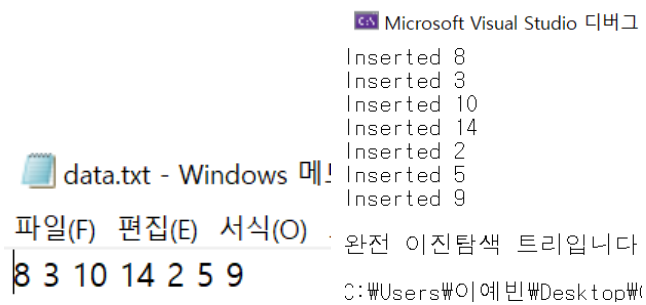


```

Microsoft Visual Studio 디버그 콘솔
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9
Inserted 11
Inserted 16
Inserted 4
Inserted 6

data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V)
8 3 10 14 2 5 9 11 16 4 6
  
```

예시 결과창 2

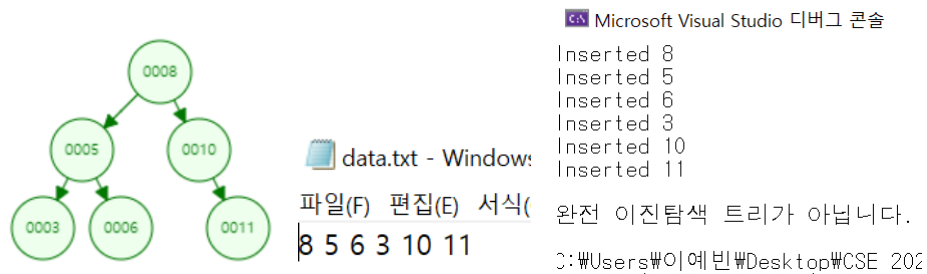


```

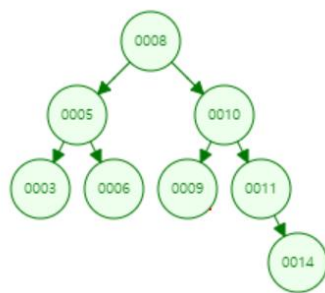
Microsoft Visual Studio 디버그
Inserted 8
Inserted 3
Inserted 10
Inserted 14
Inserted 2
Inserted 5
Inserted 9

data.txt - Windows 메모장
파일(F) 편집(E) 서식(O)
8 3 10 14 2 5 9
  
```

왼쪽 자식 노드가 없는 경우 1



왼쪽 자식 노드가 없는 경우 2



data.txt - Windows 메모장
 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
 8 5 3 6 10 9 11 14

Microsoft Visual Studio 디버그 콘솔

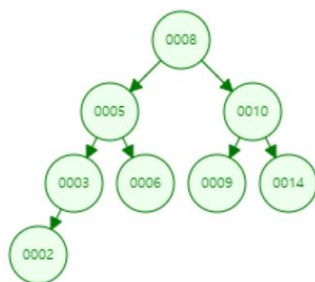
Inserted 8
 Inserted 5
 Inserted 3
 Inserted 6
 Inserted 10
 Inserted 9
 Inserted 11
 Inserted 14

완전 이진탐색 트리가 아닙니다.

C:\Users\이예빈\Desktop\CSE 2021-2\

오른쪽 자식 노드가 없는 경우 1 - 완전 이진 트리 O

(마지막 레벨에 첫번째 노드만 있는 경우)



data.txt - Windows 메모장
 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
 8 5 6 3 10 14 2 5 9

Microsoft Visual Studio 디버그 콘솔

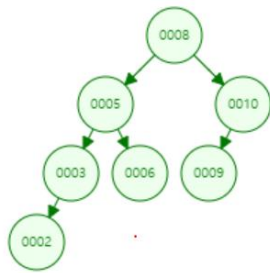
Inserted 8
 Inserted 5
 Inserted 6
 Inserted 3
 Inserted 10
 Inserted 14
 Inserted 2
 Inserted 9

완전 이진탐색 트리입니다

C:\Users\이예빈\Desktop\

오른쪽 자식 노드가 없는 경우 2 - 완전 이진 트리 X

(단말 노드 여부 확인)



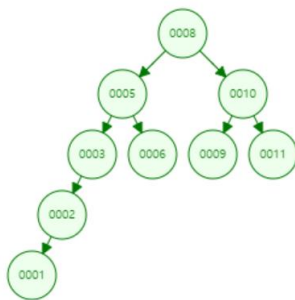
```

Microsoft Visual Studio 디버그 콘솔
Inserted 8
Inserted 5
Inserted 6
Inserted 3
Inserted 10
Inserted 9
Inserted 2

data.txt - Windows
파일(F) 편집(E) 서식(O)
8 5 6 3 10 9 2 5
완전 이진탐색 트리가 아닙니다.
C:\Users\이예빈\Desktop\CSE 20:
  
```

오른쪽 자식 노드가 없는 경우 3 - 완전 이진 트리 X

(단말 노드 여부 확인)



```

Microsoft Visual Studio 디버그 콘솔
Inserted 8
Inserted 5
Inserted 6
Inserted 3
Inserted 10
Inserted 11
Inserted 9
Inserted 2
Inserted 1

data.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기
8 5 6 3 10 11 9 2 1
완전 이진탐색 트리가 아닙니다.
C:\Users\이예빈\Desktop\CSE 2021
  
```

2.5 느낀점

이번 문제를 해결하기 위해 완전 이진 트리의 개념을 다시 살펴보았다. 어떠한 순회를 통해, 어떠한 조건을 거쳐 완전 이진 트리를 판별해야 할까 라는 고민에서 시간을 많이 투자해야 했던 것 같다. 트리의 전체 노드 개수 또는 전체 높이를 이용하는 방법이 가장 처음 떠올라 이를 활용한 여러 가지 아이디어들을 떠올려 보았지만, 배열 큐 구조를 이용한 레벨 순회라는 점, 그리고 각 노드의 링크를 이용하여 자식 노드들의 여부를 쉽게 알 수 있다는 점에 있어서 큐에 삽입된 노드들의 단말 여부를 판별해보는 방식이 가장 효율적이라고 생각하였다.

그러나, 단말 여부를 확인하는 방법은, 트리의 높은 레벨 위치에서 판별하여, 큐에 삽입된 노드가 많은 경우에는 매우 비효율적인 아이디어가 될 수도 있다. 따라서 전체 노드의 개수를 구하고, 큐에 삽입한 노드의 전체 개수(마지막 레벨까지 도달한 경우)를 결과적으로 비교하여 이 값이 다른 경우라면 완전 이진 트리에 부합하지 않도록 만드는 추가적인 노드가 누군가의 자식 노드로 달려있음을 의미하게 되는 아이디어가 떠올라 급히 추가적인 문제 해결 과정을 작성하게 되었다.

큐의 구조, 레벨 순회와 이진 트리의 개념을 모두 연결시켜야 했기 때문에 더 효율적인 아이디어를 도출하는데 많은 시간이 걸린 것 같다. 앞으로는 개념을 확실히 짚고 넘어가야 할 것 같다.

교수님께서 트리의 레벨을 이용하는 방법도 언급하셨는데, 레벨을 활용한 방법은 떠오르지 않아 무척이나 아쉽다. 더 많은 고민을 하고, 시간을 투자하여 더 효율적인 방법을 발견했으면 하는 바람이 있다.

3. 느낀점

4주차 과제를 통해 이진 탐색 트리의 개념을 확실하게 공부할 수 있었다. 이진 탐색 트리의 탐색과 삽입, 그리고 삭제 연산을 비롯하여 이진 트리에 대해 그동안 2, 3주동안 공부를 했지만 이제서야 트리라는 개념에 익숙해진 것 같아 앞으로 더 많은 복습이 필요할 것 같다. 첫번째 문제를 해결하는 과정에서는 큰 어려움이 없었지만, 완전 이진 탐색 트리를 판별해야 하는 두번째 문제에서는 많은 시간을 투자했어야 했다. 비록 조금 더 효율적인 방법을 미리 발견하지 못했다는 아쉬운 점은 있지만, 최대한 공부한 내용, 고민한 아이디어들을 레포트에 담을 수 있어서 만족한다.

다음에는 순환을 이용한 순회, 반복을 이용한 순회를 비교해보고 싶다.