**알고리즘 중간 과제**

**2015003654 이호영**

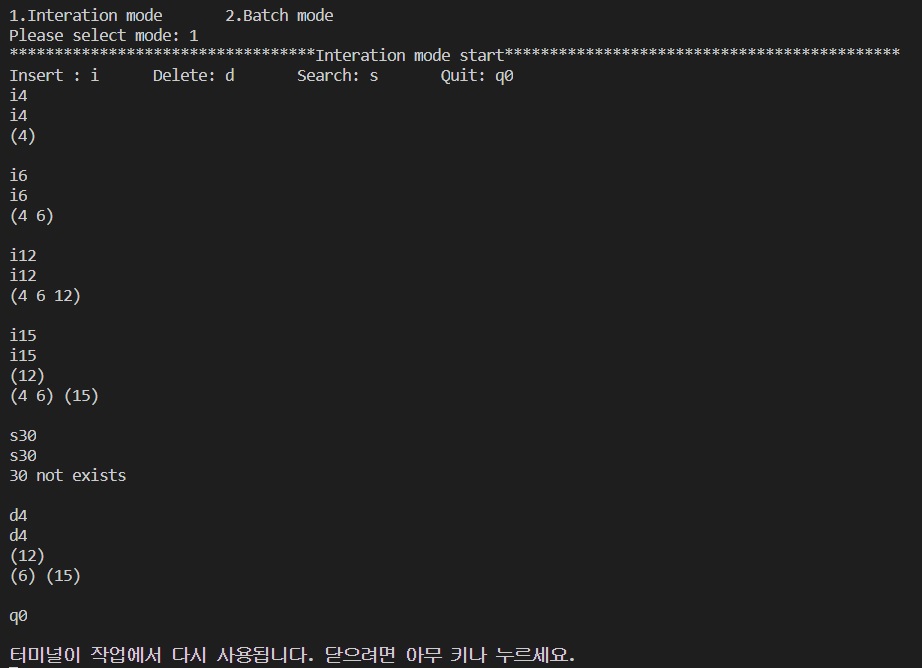
* + 1. **Tree 구현**

1. 구현 환경

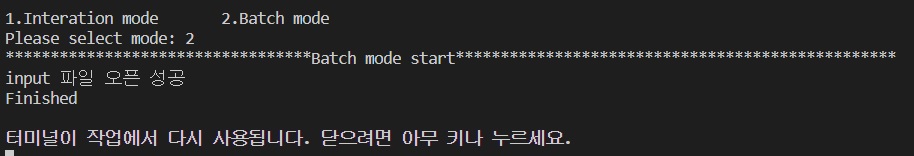
Visual Studio Code version 1.51.1

Input.txt 파일을 읽고 c 파일 경로에 output.txt를 생성하는 방법을 구현하지 못하여 수업을 같이듣는 친구의 도움을 받았습니다. 하지만 프로그램을 다루는 실력이 부족하여 Visual Studio Code로 input.txt과 output.txt 파일을 c파일과 같은 경로에서 읽고 쓰는 것을 구현해내지 못하였습니다. 다음 과제는 visual studio 2017을 설치하여서 모든 기능을 구현하겠습니다. 그리고 tree를 print하는 idea를 또한 친구에게 얻었습니다.

Interaction mode의 예시



Batch mode 예시



1. 소스 코드 설명

먼저 파일은 main.c, 234tree.c, 234tree.h, insert.c, delete.c, search.c로 이루어져 있습니다.

Main.c

Interaction mode와 Batch mode를 선택할 수 있는 메뉴창을 표시하였고 interaction mode는 위의 예시와 같이 i4, s30, d4를 통해서 insert와 delete, search를 할 수 있도록 구현하였습니다. Insert, search, delete 직후에는 printTree 함수로 트리를 출력합니다.

Batch mode는 input txt 파일을 읽고 읽어 들인 명령어를 수행해 output.txt 파일에 명령어와 생성된 트리를 출력하도록 했습니다. Output 파일에 fprintfTree 함수로 출력합니다.

void printNode(TreeNode \*);

void printTree(TreeNode \*);

void fprintNode(TreeNode \*);

void fprintTree(TreeNode \*);

printNode 함수는 한 노드의 키 값들을 모두 출력하는 함수입니다.

printTree 함수는 level이 홀수면 node1의 배열에 노드를 저장하고 짝수면 node2의 배열에 노드를 저장해서 번갈아가며 서로의 배열에 값을 저장하고 level을 하나씩 늘려나가면서 모든 노드들을 배열에 저장합니다.

234tree.c

typedef struct TreeNode

{

    int key[3];

    struct TreeNode \*cNode[4];

    struct TreeNode \*pLink;

} TreeNode;

세 개의 키와 네 개의 자식링크를 가지는 234 tree의 노드 하나를 구현했고, 부모 노드로 가는 링크도 구현했습니다.

234tree.h

#ifndef \_TreeNode

#define \_TreeNode

#include <stdio.h>

typedef struct TreeNode

{

    int key[3];

    struct TreeNode \*cNode[4];

    struct TreeNode \*pLink;

} TreeNode;

#endif

234tree.c를 insert.c, delete.c, search.c에서 동시에 include 시켰을 때 같은 변수나 함수가 여러 번 정의되어서 오류를 일으켰습니다. 그래서 한 번 정의된 것을 다시 정의하지 않기 위해서 헤더 파일을 만들었습니다.

Insert.c

TreeNode \*nodeInit();

void insert(TreeNode \*, int);

TreeNode \*split(TreeNode \*, TreeNode \*);

void propagate(TreeNode \*);

void equalizeNode(TreeNode \*);

nodeInit은 노드의 키 값을 0으로 초기화하고 cNode를 NULL로 초기화하며 동적할당을 통해 공간을 할당 받습니다.

Insert를 구현하는 도중에 여러 가지 경우가 나오는데 split은 한 노드에 키가 4개 들어갈 때 세 번째로 작은 값을 부모 노드로 올리는 함수입니다. 그리고 split을 통해서 원래 tree에 존재하지 않던 부모 노드가 생기게 되는데 그 노드와 원래 부모노드를 결합하는 것이 propagate입니다.

equalizeNode는 원래는 구현하지 않으려 했으나 새로운 노드를 생성할 때 같은 노드가 두 개가 중복되는 현상이 반복되어서 구현하였습니다. 정확한 원리는 잘 모르겠지만 이것저것 해보다가 나온 방법입니다.

Search.c

가장 간단한 함수로, 찾고자 하는 데이터가 있는 노드까지 이동한 후 찾으면 index를 반환하고 찾지 못하면 -1을 반환합니다.

Delete.c

void delete (TreeNode \*, int);

int findSuccessor(TreeNode \*, int);

void reconstruct(TreeNode \*);

int findNum(TreeNode \*);

void fusion(TreeNode \*, TreeNode \*);

void transferR(TreeNode \*, TreeNode \*);

void transferL(TreeNode \*, TreeNode \*);

가장 복잡한 delete입니다. Delete는 지우고자 하는 값과 root를 매개변수로 잡습니다. 그 과정에서 필요한 함수들을 만들었습니다. findSuccessor는 말 그대로 successor를 찾아 그 index를 반환하는 함수입니다. 지우고자 하는 값보다 큰 값들 중 가장 작은 값을 찾는 함수입니다. Reconstruct는 key 값을 지웠을 때 빈 노드가 생겨서 depth 특성이 깨지는 것을 막기 위해 구현했습니다. 빈 노드가 생겼을 때 3,4-node인 형제 노드가 있다면 transfer 함수를 사용합니다. 빈 노드의 위치에 따라 L과 R로 나뉘는데 key를 주는 노드가 빈 노드의 왼쪽에 있으면 R, 오른쪽에 있으면 L을 실행합니다. 부모 노드 값을 빈 노드로 옮기고 형제 노드의 키 값을 하나 부모에게 넘겨줘서 회전하는 모양이 됩니다. 형제 노드들이 모두 2-node이고 부모 노드가 3,4-node일 때 fusion을 실행합니다. Fusion은 부모 노드의 키 하나와 빈 노드의 바로 옆 형제 노드의 키 하나를 결합하는 함수입니다. findNum은 노드의 key 개수를 뽑아내기 위해 만들었습니다.

1. 교과서의 Insert와 Delete 알고리즘과의 비교

Insert는 수업 자료에서와 같이 세번째로 작은 값을 split하여 부모 노드로 올려보내 같은 방법으로 구현하였습니다. Delete는 과제 설명에 나왔듯이 successor를 올려서 지울 숫자를 leaf로 내려 삭제하였으며 predecessor를 이용한 수업 자료와는 다르게 구현하였습니다.