

데이터구조 설계 및 실습

1 차 프로젝트 보고서

제출 일자 : 2022 년 10 월 13 일 (목)

학과 : 컴퓨터정보공학부
학 번 : 2021202043
이 름 : 이 은 서

1. Introduction

이진탐색트리와 연결리스트, 큐 자료구조를 사용하여 사진파일을 편집하는 프로그램을 구현하는 과제이다. 사진 파일에 대한 경로를 불러와서 링크드 리스트로 저장한 후, 검색을 용이하게 하기 위해 트리 자료구조로 저장한다.

프로젝트에서 요구하는 기능은 다음과 같다.

명령어	간단한 기능 설명
LOAD	csv 파일의 데이터 정보를 불러서 링크드 리스트에 저장
ADD	새로운 디렉토리 경로를 탐색하여 데이터 정보를 링크드 리스트에 추가
MODIFY	링크드 리스트에 존재하는 노드의 파일 고유 번호를 수정
MOVE	링크드 리스트에 저장된 정보를 BST 노드에 옮기고 연결
PRINT	BST 에 저장되어 있는 정보를 출력
SEARCH	특정 단어가 포함된 파일의 이름을 탐색하여 출력
SELECT	EDIT 명령어 전에 선행되어야 하는 명령어로 이미지의 경로를 찾아서 이미지를 부름
EDIT	선택된 이미지를 불러와 점대칭/밝기조정/크기조정 3 가지의 기능을 사용하여 이미지를 편집

1) LOAD

LOAD 명령어는 img_files/filesnumbers.csv 라는 지정된 경로에 접근하여 csv 파일의 데이터 정보를 불러오는 명령어이다. 링크드 리스트 자료구조에 모두 접근하고 저장된 순서대로 파일 이름과 고유 번호를 출력한다. 만약, 노드 개수가 100 개가 넘을 경우 먼저 들어온 순서대로 노드를 삭제한 후 새로운 데이터를 추가한다. 만약, 경로를 찾을 수 없는 경우에는 에러코드 100 을 출력한다. 다음과 같은 명령을 수행하기 위해 연결리스트로 사용하기 위한 노드와 노드들을 연결한 리스트 클래스가 필요하다.

2) ADD

ADD 명령어는 사용자가 직접 디렉토리명과 파일명을 입력하면, 해당 경로가 있는지 탐색하고, 만약 경로가 있다면 링크드리스트에 데이터를 추가하는 명령어이다. LOAD 명령이 선행되어야 하고, ADD 를 통해 새로 저장된 디렉토리명과 파일들은 2 차원 링크드리스트로 구성된다. 즉, 맨 처음 LOAD 를 통해 저장된 img_files 디렉토리명의 파일들의 뒤에 연결되는 것이 아닌, 새로운 디렉토리명의 리스트가 생성되고, 이것이 저장된 디렉토리 리스트끼리 연결되어 2 차원을 구성하는 것이다. 만약, 노드 개수가 100 개가 넘을 경우 먼저 들어온 순서대로 노드를 삭제한 후 새로운 데이터를 추가하고, 인자가 하나라도 존재하지 않거나 링크드리스트가 존재하지 않는 경우 에러코드 200 을 출력한다.

3) MODIFY

MODIFY 명령어는 LOAD 와 ADD 를 통해 저장된 링크드리스트에 존재하는 노드를 탐색하여 파일 고유 번호를 수정하기 위한 명령어이다. 이때, 수정하는 노드는 번호가 바뀌는 것이 아닌 아예 삭제되는 것이고, 수정된 번호가 적힌 새로운 노드를 생성하여 교체한다. 파일 고유번호는 모든 데이터셋에 대해 중복되지 않으며, 인자가 하나라도 없거나 변경할 파일 이름을 찾을 수 없거나 중복되는 고유 번호로 할당할 경우 에러코드 300 을 출력한다.

4) MOVE

MOVE 명령어는 링크드리스트의 정보들을 Binary Search Tree 로 옮기는 명령어이다. 이는 다음에 진행할 SEARCH 명령의 검색을 용이하게 하기 위해 진행하는 과정이며 BST 의 최대 노드 개수는 300 개이다. 300 개가 넘는 경우 고유 번호가 낮은 순서부터 제거하며, 노드가 존재하지 않을 때는 에러코드 400 을 출력한다.

5) PRINT

PRINT 명령어는 BST 에 저장되어 있는 정보를 출력하는 명령어이다. 중위 순회(In-order)방식의 탐색을 이용하여 정보를 출력하고, 만약 BST 가 존재하지 않을 경우에는 에러코드 500 을 출력한다. 이때, 출력하는 정보는 폴더명 / "파일명" / 파일고유번호 다음과 같은 출력 포맷을 지닌다.

6) SEARCH

SEARCH 명령어는 특정 단어가 포함된 파일의 이름을 탐색하여 출력하는 명령어이다. 탐색은 보이어 무어 알고리즘을 사용하며, 보이어무어 알고리즘을 사용하기 위해서는 다음과 같은 규칙을 따른다.

1. 이름기반 검색을 위해 Iterative post-order 방식으로 순회하여 queue 에 트리 노드의 정보를 순차적으로 담는다. Iterative post-order 방식으로 순회할 때는 재귀함수를 사용하지 않는다.
2. 모든 정보가 queue 에 저장되었다면 순차적으로 pop 을 진행하고, 이때 보이어무어 알고리즘을 사용하여 검색한 단어가 파일 명에 있는지 검색한다.
3. 단어가 존재하는 경우에는 해당 노드의 파일명과 고유번호를 출력한다.
4. 검색이 완료된 이후의 큐는 반드시 비어 있어야 한다.

즉, 다음과 같은 규칙을 따르면 사용자가 검색한 단어가 포함된 파일의 이름이 여러개 나올 수 있으며, 끝까지 비교하기 때문에 큐는 비어있게 된다. 만약, BST 가 비어있거나 인자가 없는 경우에는 에러코드 600 을 출력한다.

7) SELECT

SELECT 명령어는 EDIT 명령어를 수행하기 전 선행되는 명령어이며, BST 에서 파일 고유 번호를 기반으로 전위 순회(pre-order)를 통해 이미지의 경로를 찾아 이미지를 불러오는 명령어이다. 인자가 없거나 파일 고유번호가 존재하지 않을 경우에는

에러코드 700 을 출력한다. 이미지를 불러와서 편집을 해야하므로 SELECT 를 통해 찾은 이미지의 경로를 따로 저장하였다가 EDIT 에 사용한다.

8) EDIT

EDIT 명령어는 SELECT 명령어를 통해 가져온 이미지를 편집하는 명령어이다. 이미지 파일을 읽어서 수정하고 저장한다. 인자로는 세 가지의 옵션 -f, -l, -r 이 존재한다.

1. 점대칭 -f

불러온 이미지의 픽셀을 Stack 에 순서대로 입력하고 후입선출하여 이미지를 점대칭으로 뒤집는다. 점대칭을 한 이미지는 파일명 뒤에 "_flipped"를 덧붙여 "Result" 디렉토리에 저장한다.

2. 밝기조정 -l

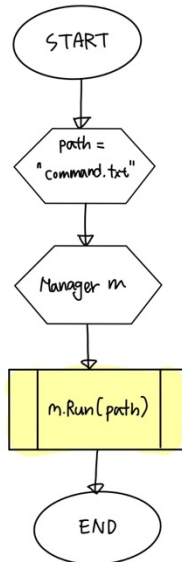
불러온 이미지의 픽셀을 Queue 에 순서대로 입력하고 선입선출한다. 이때 추가적으로 밝기를 조정할 특정 숫자를 더하여 이미지의 밝기를 조절한다. 만약, 밝기가 최대인 픽셀이거나 특정 숫자만큼 더하지 못할 경우에 대해서는 최대 값인 255 를 할당받도록 한다. 밝기조정을 한 이미지는 파일명 뒤에 "_adjusted"를 덧붙여 "Result" 디렉토리에 저장한다.

3. 크기조정 -r

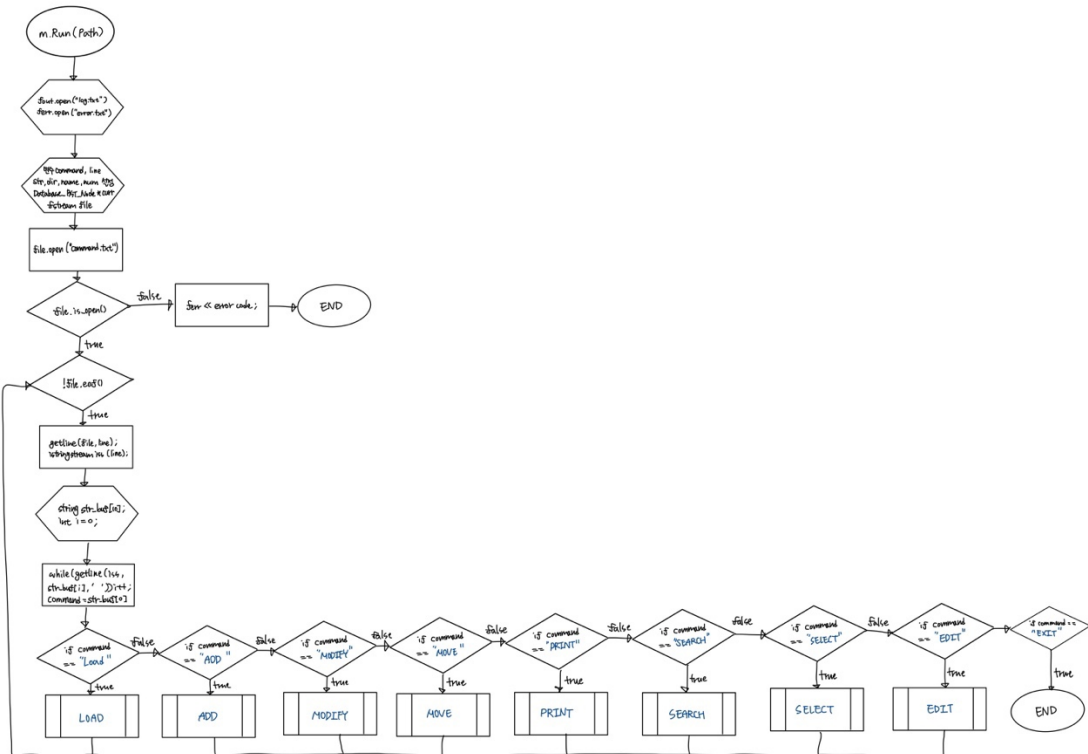
불러온 이미지의 크기를 4 분의 1 로 축소하는 동작을 진행한다. 인접한 4 개의 셀의 평균값을 구하여 하나의 셀에 입력하는 방식으로 구현을 하며, 크기조정을 한 이미지는 파일명 뒤에 "_resized"를 덧붙여 "Result" 디렉토리에 저장한다. 인자가 존재하지 않는 경우 에러코드 800 을 출력한다.

2. Flowchart

<Main>



main.cpp 파일의 경우에는 다음과 같은 순서대로 지니고 있다. command.txt 경로를 받고 Manger 클래스의 인스턴스를 만든 다음 해당 인스턴스에 접근하여 Run 함수를 실행한다. 그리고 Run 함수를 실행한 후 종료 조건을 만나면 프로그램은 끝이 난다. 다음 사진은 Manger 클래스의 Run 함수를 실행할때의 순서도이다.

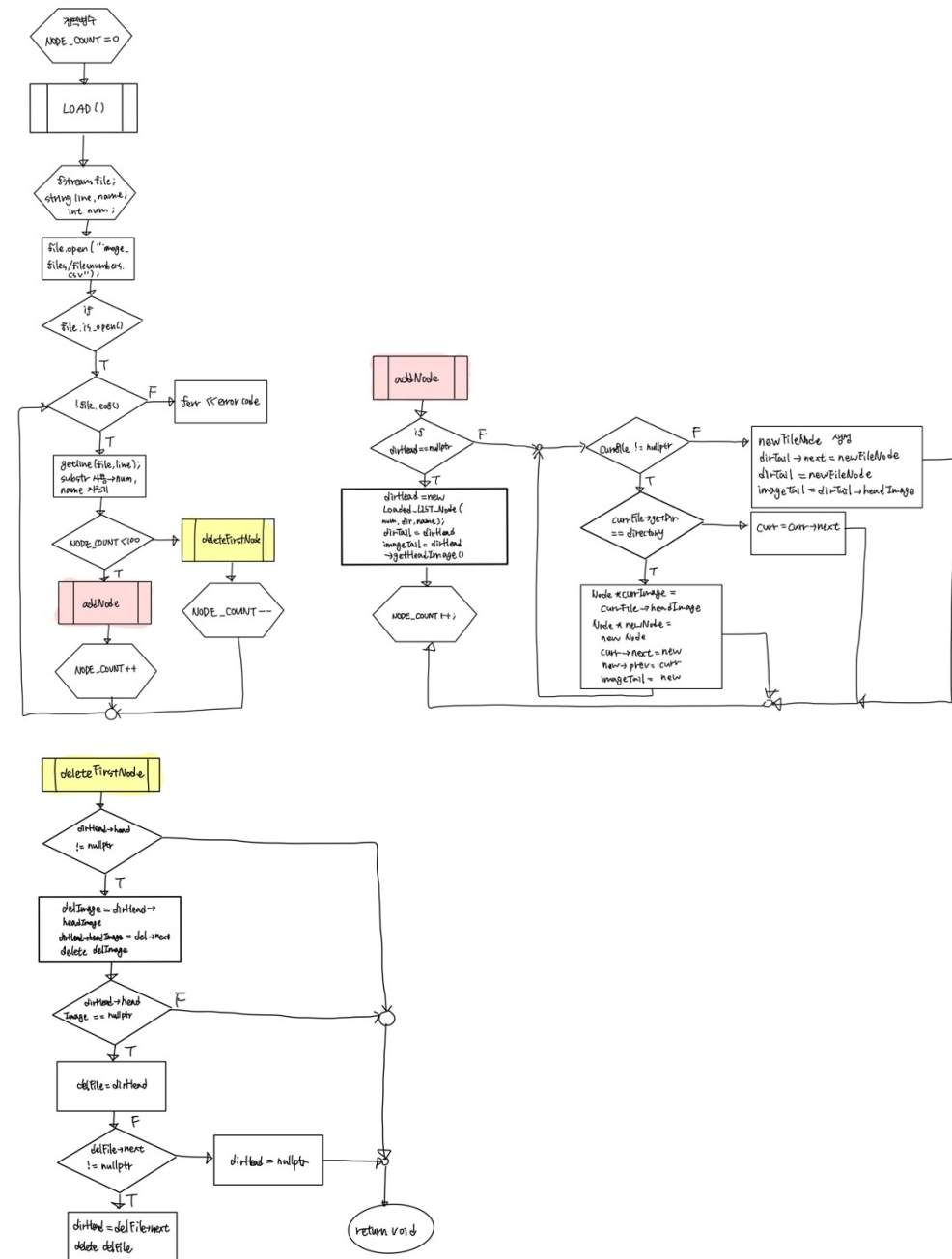


Run 함수를 실행하면 Result 와 Error 를 저장할 파일 두개 "log.txt"와 "error.txt"를 선언한다. 그리고 앞으로 사용할 변수들을 선언한 후 "command.txt"파일을 연다. 파일이

열리지 않으면 에러코드가 에러로그파일에 저장되며 끝나고, 파일이 정상적으로 열리면 파일의 끝에 도달할 때까지 "command.txt"파일을 한줄씩 읽어서 명령을 수행한다. Istringstream 과 getline 을 사용하여 공백과 "를 기준으로 문자열을 자르고 해당 변수에 넣어 명령을 수행한다. 모든 명령어는 공백이 나오기 전 첫번째 문자열에 저장되므로 command 는 항상 str_buf[0]에 저장된다. Command 를 기준으로 총 9 가지의 명령어를 나눌 수 있고, 명령어는 LOAD, ADD, MODIFY, MOVE, PRINT, SEARCH, SELECT, EDIT, EXIT 이 있다. EXIT 의 경우에는 바로 프로그램이 종료되므로 따로 플로우차를 그리지 않았고, 지금부터 명령어들의 플로우차를 차례대로 보일 예정이다.

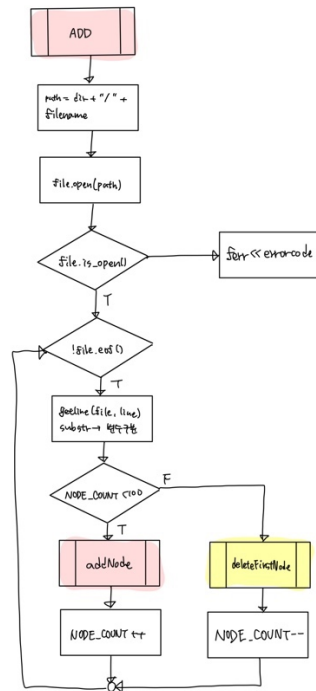
1. LOAD

< LOAD >



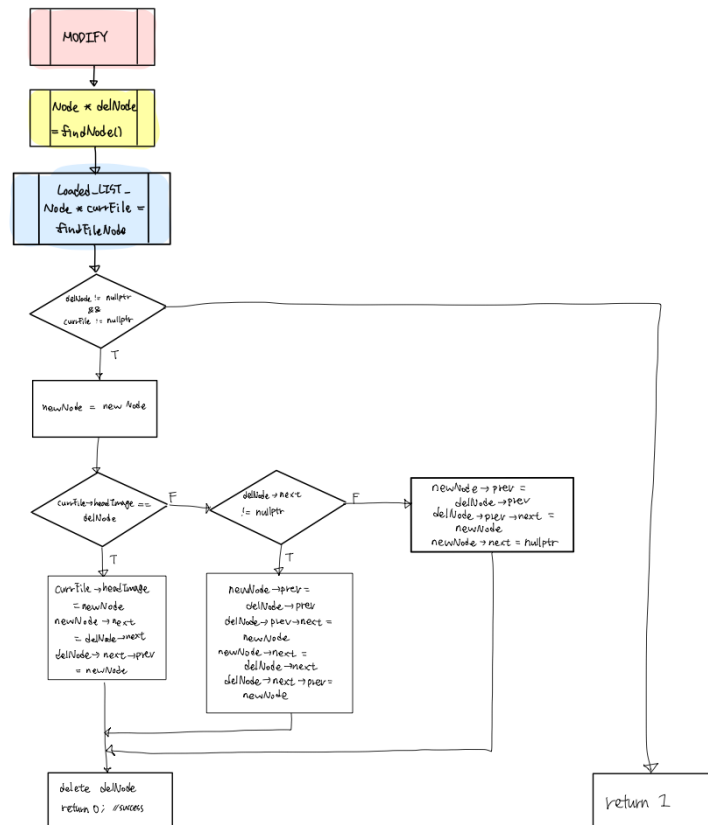
2. ADD

<ADD>



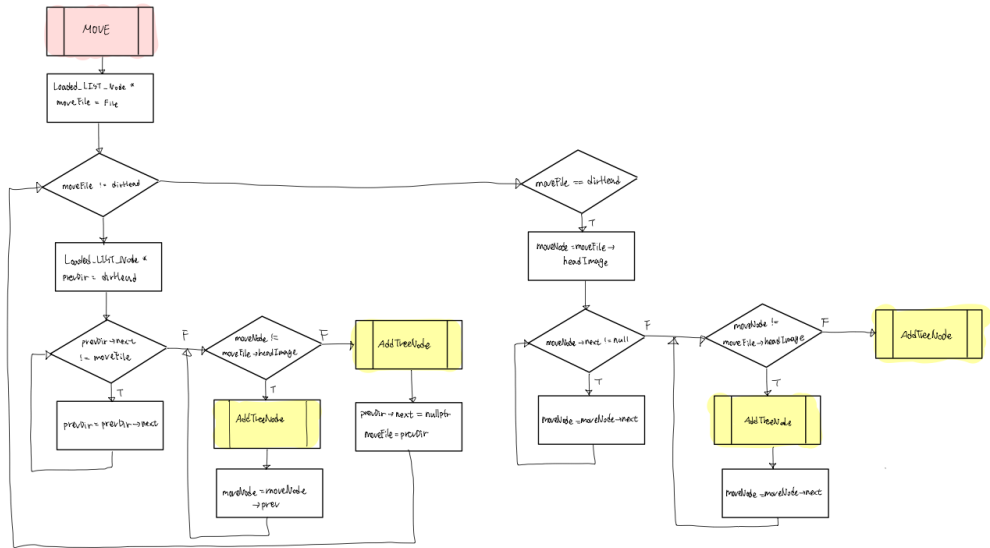
3. MODIFY

<MODIFY>



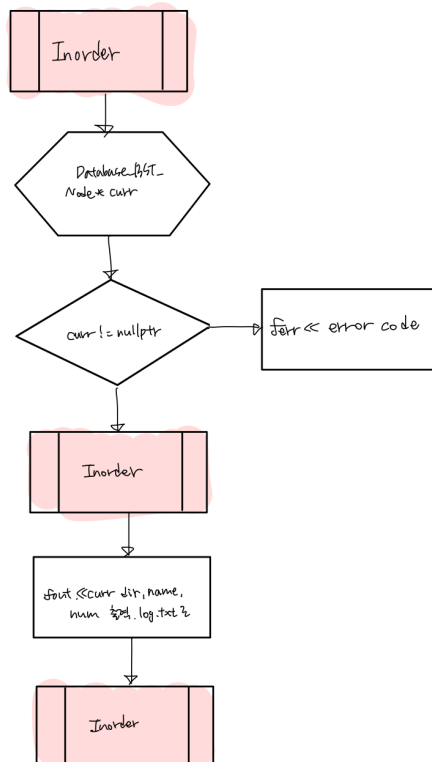
4. MOVE

<MOVE>



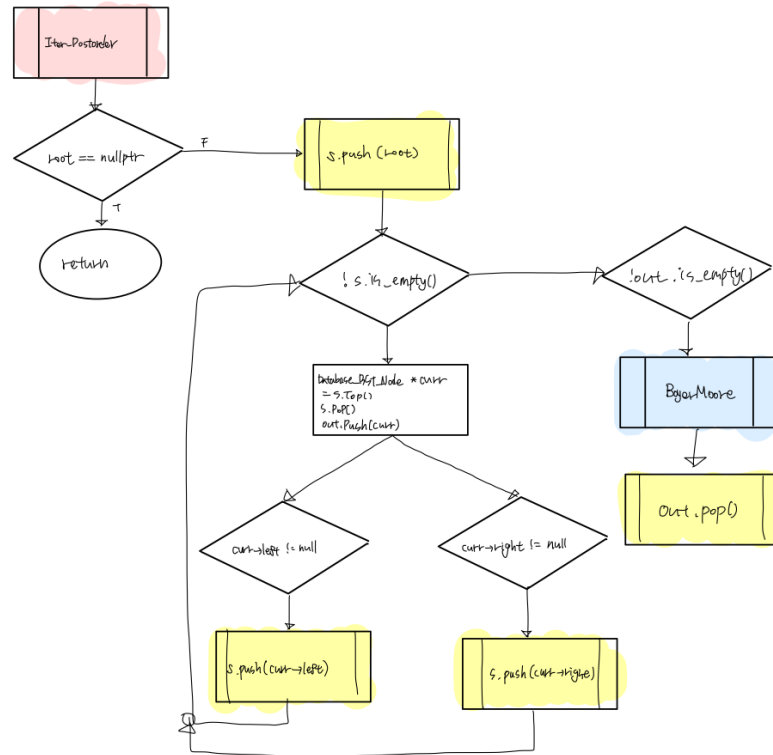
5. PRINT

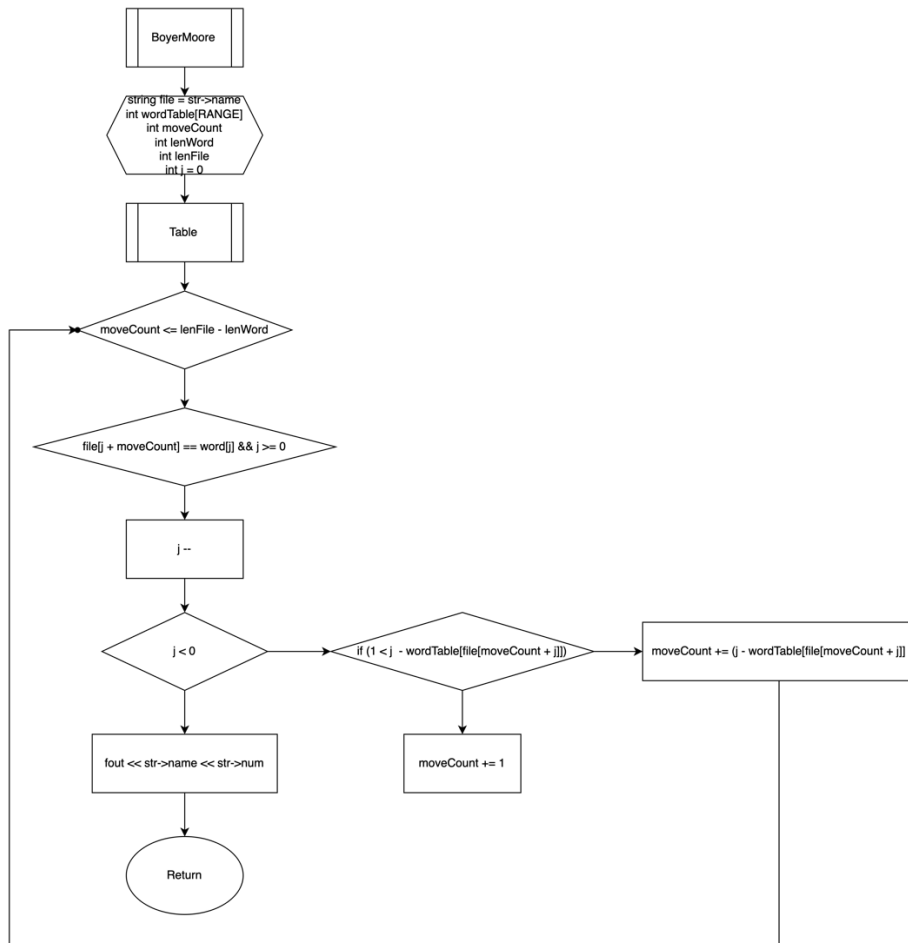
PRINT 는 INORDER 함수를 관리한 것이므로 Inorder 함수의 Flowchart 로 대체한다.



6. SEARCH

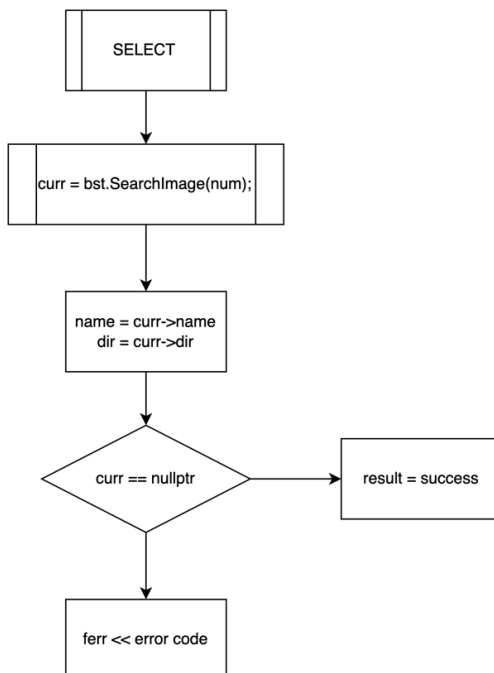
Search 함수는 Iter_Postorder 순회 방식으로 방문하면서 해당 단어를 찾는 것이므로 Iter_Postorder 함수로 대체한다.

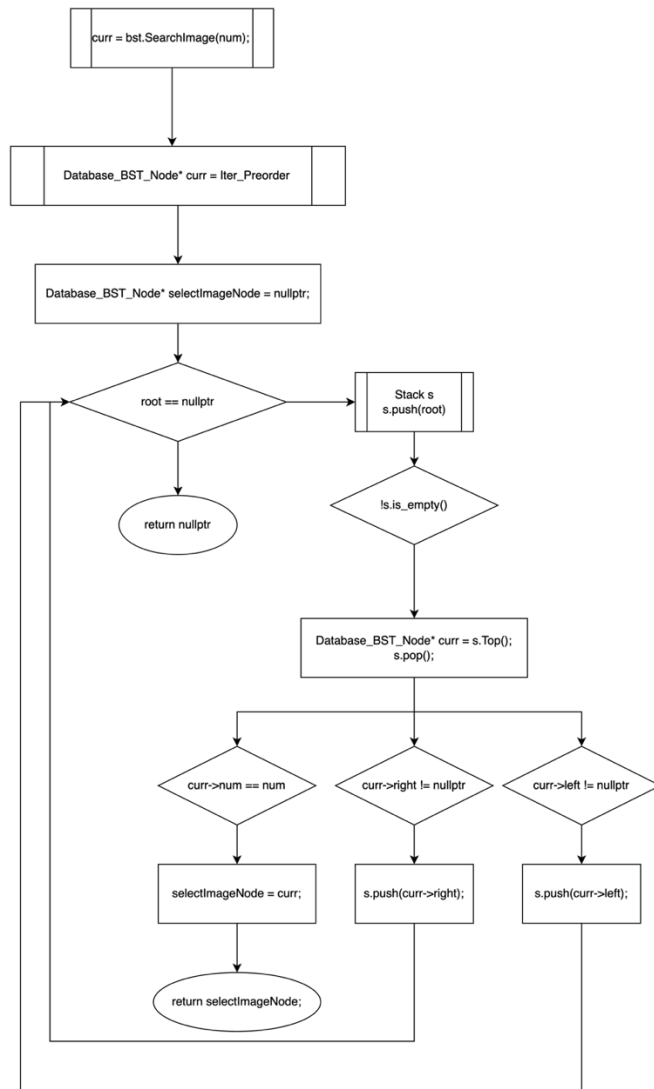




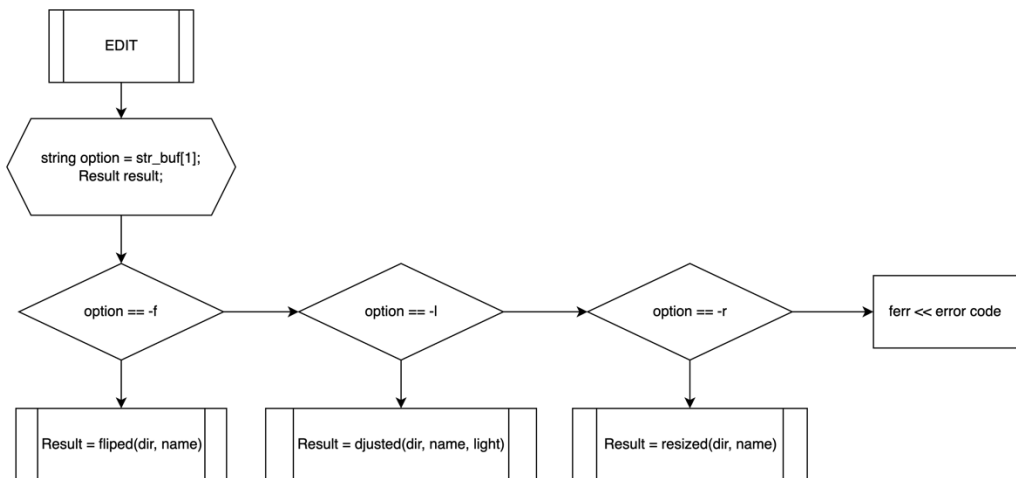
7. SELECT

SELECT 명령어는 숫자를 기준으로 하여 Iter_Preorder 로 순회하여 검색하였을 때 비교하여 나오는 값을 경로로 저장한다.





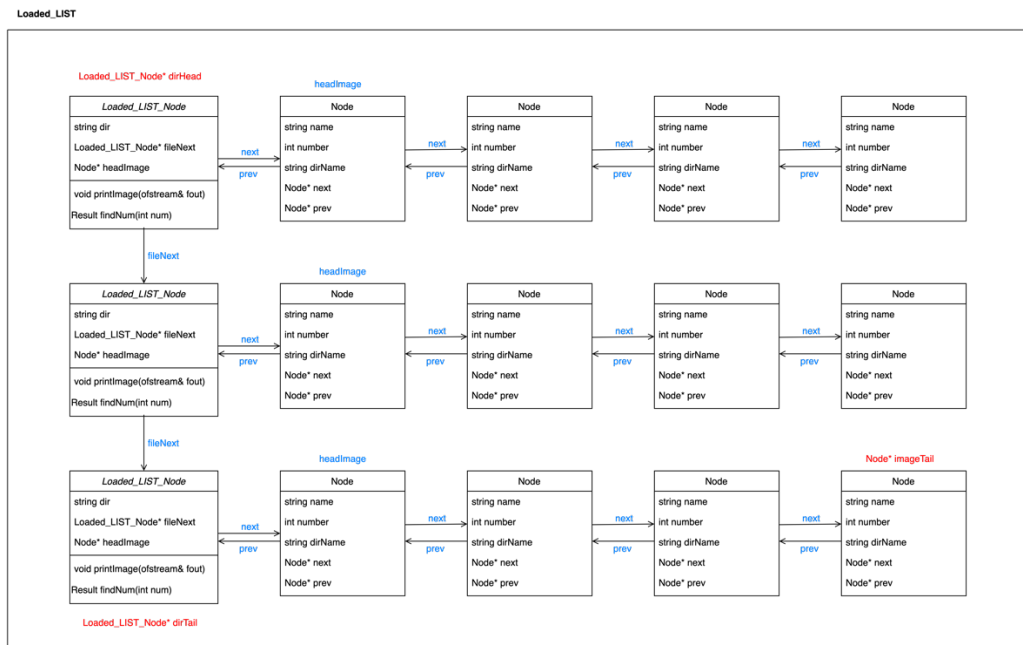
8. EDIT



3. Algorithm

1) 2D Linked-List

파일 이름으로 이루어진 1 차원 양방향 리스트를 구성한 후, 폴더 명을 기준으로 2 차원의 단방향 링크드 리스트를 구현하였다. 각 노드들은 image name, unique number, directory name 3 개의 데이터와 prev, next 포인터 값들을 가지고 있고 양방향으로 연결되어있다. 또한, 폴더명을 기준으로 각 리스트의 head 를 가리키는 노드를 next 포인터 값으로 연결시킨 단방향 링크드리스트를 생성하였다. Loaded_LIST 라는 이름의 링크드리스트이고, 이 링크드리스트는 ADD 명령어를 통해 새로운 디렉토리에 대한 정보를 추가한다. ADD 를 통해 탐색된 디렉토리는 새로운 리스트를 생성하여 2 차원으로 연결되는 것이다. MODIFY 를 통해 Loaded_LIST 에 존재하는 노드의 고유번호를 수정할 수 있고, 수정할 경우에는 기존 노드를 삭제하고 새로운 노드를 추가하는 방식으로 수정한다.



대략적인 그림은 다음과 같다. 각각의 Node 는 이미지들의 정보를 가지고 있는 노드를 의미하고, Loaded_LIST_Node 는 폴더명이 저장되어서 2 차원을 구성하는 연결리스트이다. 그리고 이 전체를 다루고 있는 것이 Loaded_LIST 이다. 다음은 Loaded_LIST class 에서 구현한 함수들이다.

반환 타입	함수명(인자)	기능 설명
void	deleteFirstNode()	노드 개수가 100 개가 넘어갔을 때 처음 들어온 노드부터 삭제하기 위해 만든 함수이다.
int	deleteFileNode(Loaded_LIST_Node* del)	하나의 폴더 리스트에 모든 노드가 삭제되어 어떠한 노드도 존재하지 않을 때 폴더 리스트를 삭제하기 위한 함수이다.
void	addNode(string name, string directory, int num)	파일을 읽어왔을 때 노드를 생성하고 연결하기 위한 함수이다.
Loaded_LIST_Node*	findFileNode(string dir)	해당 폴더명의 폴더 리스트가 있는지 탐색하는 함수이다.

이루어진다. 만약 여러번의 deleteFirstNode 함수가 사용되어 폴더 리스트는 존재하지만 headImage 가 없는 상황일 때는 dirHead->fileNext 인 파란색 폴더 노드를 dirHead 로 설정한 다음 delete 한다.

ㄴ. deleteFileNode

delete 할 폴더 노드가 폴더 노드와 폴더 노드 사이에 있는 경우 / dirTail 인 경우 2 가지의 case 를 나누어 폴더 노드의 삭제를 진행하였다. 만약 첫번째 상황, 폴더와 폴더 사이에 있는 경우에는 삭제할 폴더 노드의 이전 폴더 노드를 구하여 다음 폴더 노드와 연결한 다음 폴더를 삭제하였다. 그리고 두번째 상황, 삭제할 폴더 노드가 dirTail, 마지막 폴더 리스트인 경우에는 폴더의 이전 폴더노드를 구하여 이전 폴더노드를 dirTail 로 설정한 다음 삭제를 진행한다.

ㄷ. addNode

addNode 는 추가할 노드의 폴더명을 기준으로 탐색했을 때 어떠한 폴더 리스트도 없는 경우 / 폴더 리스트는 존재하지만 해당 폴더 명의 리스트가 없는 경우 / 이미 해당 폴더명의 리스트가 존재하는 경우 3 가지를 나누어서 작성하였다. 폴더 리스트가 하나도 존재하지 않는 경우에는 리스트를 새로 생성하고 dirHead 의 headImage 이자 첫 파일 노드를 생성한다. 해당 폴더명의 리스트가 존재하는 경우에는 폴더명과 일치하는 해당 폴더 리스트를 우선적으로 찾고, 그 폴더 리스트에 접근하여 마지막에 있는 파일 노드를 찾은 다음, 파일노드 뒤에 새로운 노드를 넣어 연결을 해준다. 만약, 폴더 리스트가 존재는 하지만 해당 폴더명의 리스트가 없는 경우에는 dirTail 뒤에 폴더 리스트를 새로 생성하고 headImage 이자 첫 노드를 설정해준다.

ㄹ. findFileNode

dirHead 부터 dirTail 까지 순차적으로 접근하며 찾고자 하는 폴더명과 비교하여 반환하는 함수이다.

ㄹ. findNode

imageHead 부터 끝까지 순차적으로 접근하며 찾고자 하는 파일명과 비교하여 반환하는 함수이다.

ㅁ. modifyNode

modifyNode 는 수정할 노드의 폴더명을 탐색하여 해당 폴더 리스트에 접근하고, 그 다음 파일명을 탐색하여 해당 파일 노드에 접근한다. 그러면 접근한 노드의 case 가 총 3 개가 나온다. 가장 먼저 수정할 노드가 headImage 인 경우 / 수정할 노드가 이미지 노드와 이미지 노드 사이에 있는 경우 / 수정할 노드가 맨 마지막 tail 인 경우이다. headImage 의 노드를 수정해야하는 경우에는 수정해서 새로 만든 노드를 headImage 로 설정하고, 새로 만든 노드의 next 를 수정할 노드의 다음 노드와 연결시킨 다음 수정되고 버려진 노드를 delete 한다. 만약, 수정할 노드가

이미지 노드와 이미지 노드 사이에 끼워져 있다면 새로운 노드와 수정할 노드의 이전노드, 새로운 노드와 수정할 노드의 다음노드끼리 next, prev를 연결시킨 다음 수정할 노드를 delete 시킨다. 마지막 tail 노드를 수정할 경우에는 수정할 노드의 이전 노드와 새로운 노드를 서로 연결시키고, 새로운 노드의 next 를 nullptr 로 지정하여 수정한다.

ㄱ. printNode

dirHead 부터 dirTail 까지 Loaded_LIST_Node class 에 있는 printImage 함수에 접근하여 폴더 리스트마다 모든 이미지 노드를 출력하도록 설정하였다.

ㅇ. searchNum

searchNum 은 Loaded_LIST_Node class 에 있는 findNum 함수를 사용하여 각 폴더 리스트에 있는 모든 이미지 노드에 접근하여 해당 num 이 존재하는지 탐색하는 함수이다.

ㅈ. LOAD

Img_files/filesnumbers.csv 경로에 있는 파일을 읽고 링크드리스트에 저장하는 함수이다. Getline 을 사용하여 한 줄씩 파일을 읽고 ','를 기준으로 숫자와 이름을 자른 다음 이름의 뒤에서 '.'을 기준으로 잘라 '.RAW' 확장자를 제외한 이름을 addNode 를 이용하여 리스트에 정보를 저장하는 함수이다.

ㅊ. ADD

LOAD 함수와 비슷한 기능을 지닌다. 한가지 다른점은 사용자로부터 입력받은 폴더명과 파일명을 경로로 설정하여 파일을 읽고 링크드리스트에 저장한다는 것이다. 그 외에는 LOAD 의 기능과 동일하다.

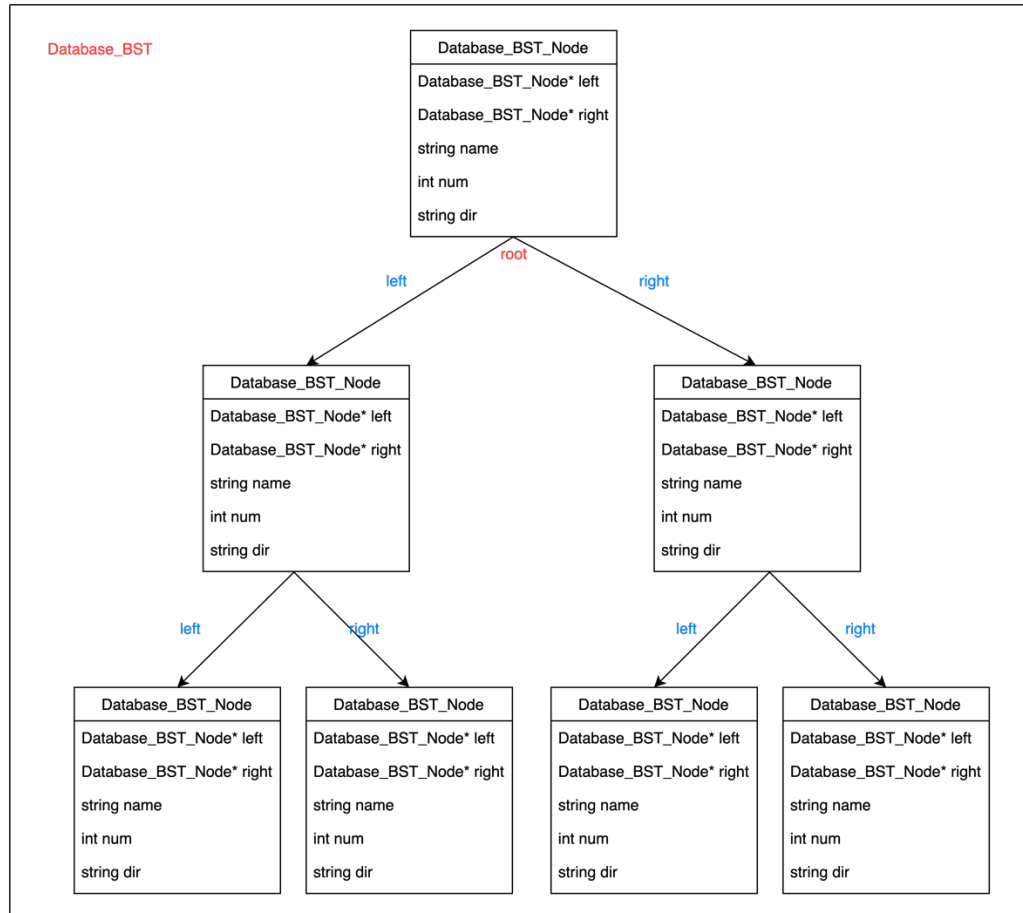
ㅋ. PrintList

dirHead 부터 dirTail 까지 각 폴더 리스트에 접근하여 printNode 함수를 사용하여 모든 이미지 노드를 출력할 수 있도록 총괄하는 함수이다.

ㅌ. MODIFY

modifyNode 함수를 수행한 후 에러가 발생하진 않았는지 상태를 체크하는 총괄 함수이다.

2) Binary Search Tree



앞서 저장한 2 차원 링크드리스트에서 마지막 노드부터 불러와서 BST 구조에 저장하는 알고리즘이다. BST 를 구성하는 class name 은 Database_BST 이고, 각각의 노드들은 다음과 같은 규칙에 따라 연결된다.

1. 부모 노드보다 파일 고유번호가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 위치하도록 한다.
2. 노드를 제거할 때, 양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰노드를 제거되는 노드 위치로 이동시킨다.

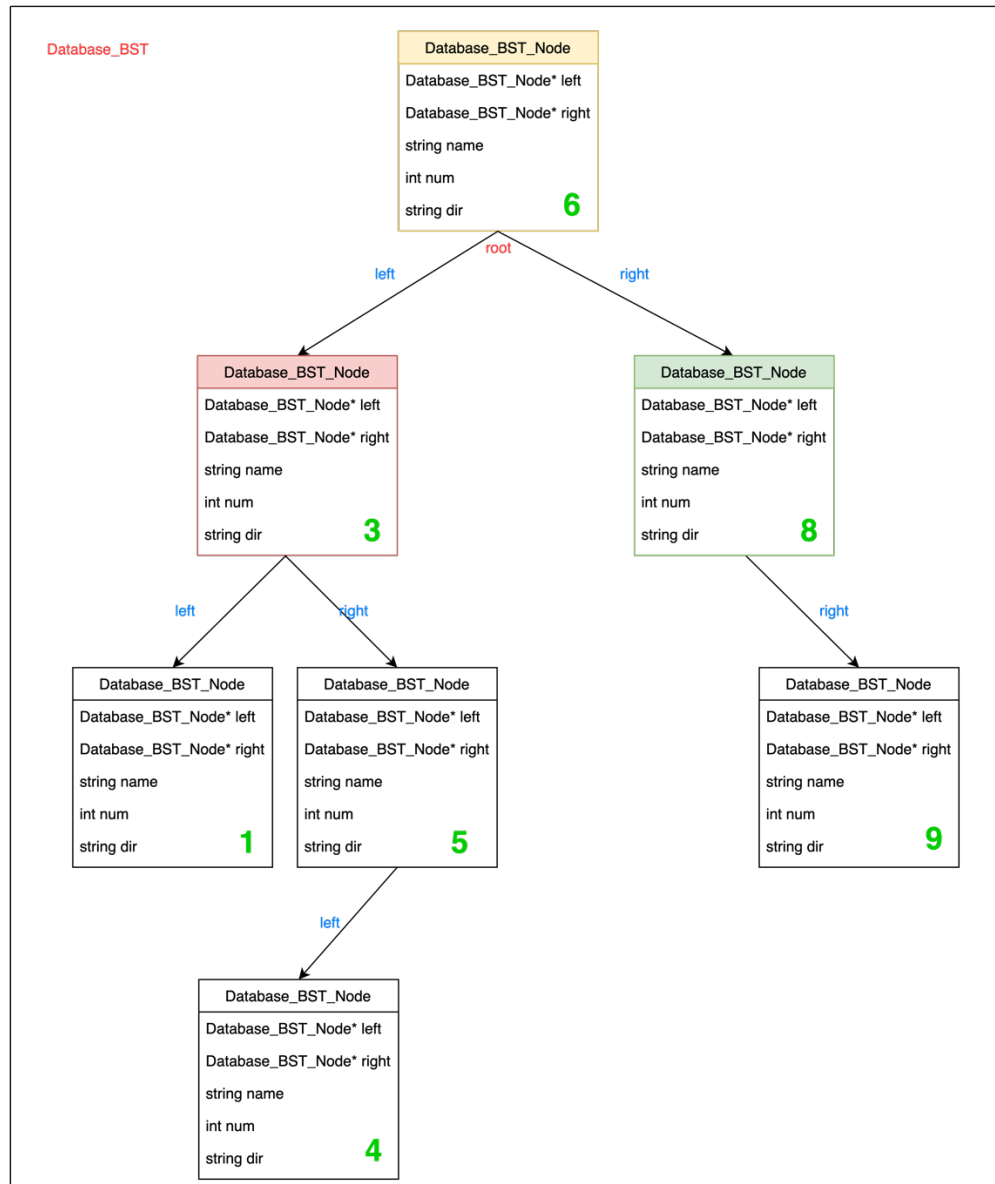
트리에 노드를 저장할 때는 마지막 노드 tail 부터 차례대로 입력해야하므로 링크드리스트를 구현할 때 마지막 노드를 tail 로 지정해주어야 하며, 이러한 BST 구조를 통해 빠른 검색이 가능하도록 한다. 최대 크기는 300 개이며, 이 이상의 노드가 배치되었을 때는 고유번호가 낮은 노드부터 순차적으로 삭제한다. Database_BST class 에서 구현한 함수들은 다음과 같다.

반환 타입	함수명(인자)	기능 설명
void	Inorder(Database_BST_Node* curr, ofstream& fout)	In-order 방식으로 트리를 순회하여 출력하는 함수
void	Iter_Postorder(Stack s, Stack out, string word, ofstream& fout)	Post-order 방식으로 트리를 순회하기 위해 stack 을 사용한다.
Database_BST_Node*	Iter_Preorder(int num)	Pre-order 방식으로 트리를 순회하여 검색하고자 하는 파일 고유번호와 비교하여 해당 노드를 반환

void	AddTreeNode(Node* tail)	BST 에 TreeNode 를 추가
Database_BST_Node *	SearchImage(int num)	파일 고유번호 기반 탐색한 이미지 노드를 반환
void	Print(ofstream& fout)	fout 파일에 BST 에 있는 트리를 Inorder() 함수를 사용하여 In-order 방식으로 트리 순회 후 출력
void	MOVE(Node* moveNode, Loaded_LIST_Node* moveFile, Loaded_LIST_Node* dirHead)	링크드리스트에 저장된 노드를 BST 로 옮기는 함수
void	Table(string str, int size, int table[])	보이어 무어 알고리즘을 사용할 때 최소한으로 단어를 이동하여 비교하기 위해 사용할 배열을 만드는 함수
void	BoyerMoore(string word, Database_BST_Node* str, ofstream& fout)	보이어 무어 알고리즘을 사용하는 함수
Result	BSTisEmpty()	BST 에 값이 있는지 확인하는 함수.

ㄱ. Inorder

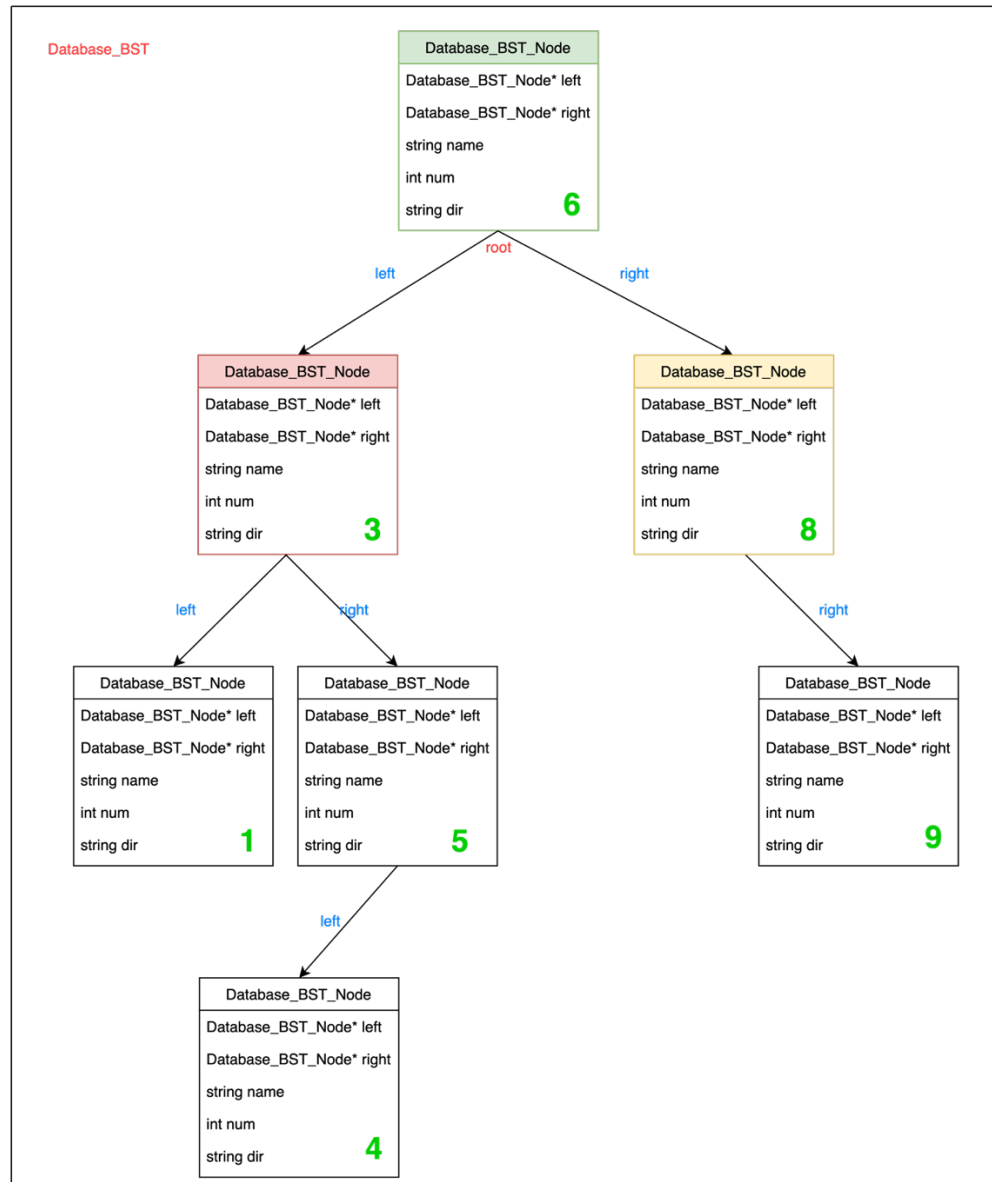
왼쪽 노드에 먼저 접근하여 값을 출력한 다음, 현재 노드의 값을 출력하고, 그 다음 오른쪽 노드에 접근하여 값을 출력하도록 재귀함수를 사용하여 만든 함수이다.



다음은 In-order 순회를 쉽게 이해하기 위해 그린 도식화 그림의 예시이다. 초록색으로 써져있는 번호는 파일고유번호가 아닌 이해를 돕기 위해 임의로 붙인 숫자이다. In-order 는 왼쪽 노드를 먼저 방문한 다음 자신의 노드를 방문한 후 오른쪽 노드를 방문하기 때문에 초록색으로 써져있는 숫자를 기준으로 BST 가 구현되었을 때 In-order 방식으로 노드의 값을 출력하면 1->3->4->5->6->8->9 순서가 된다.

ㄴ. Iter_Postorder

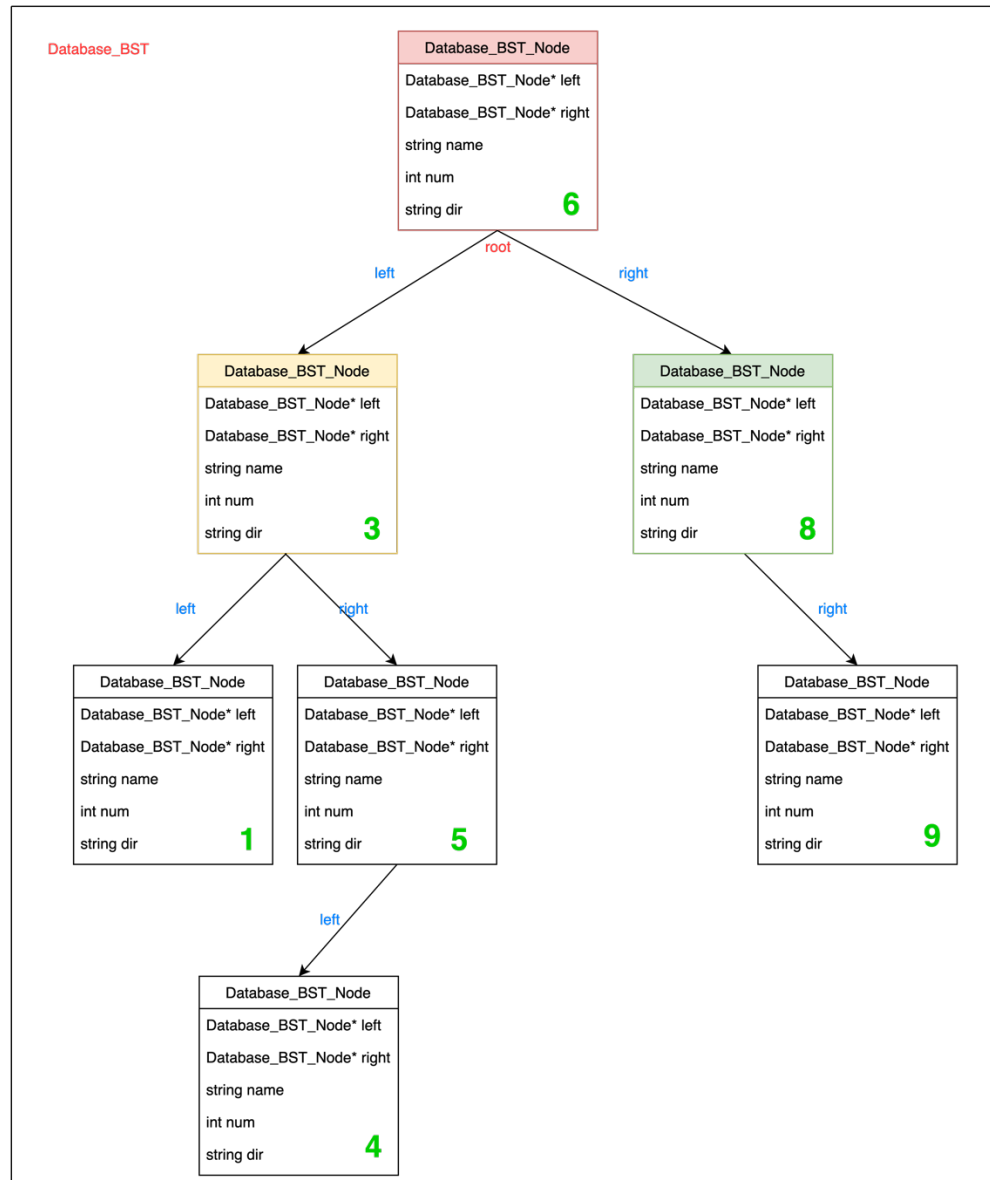
재귀함수를 사용하지 않고 Post-order 방식의 순회를 하기 위해 Stack 을 사용하여 root 먼저 스택에 넣은 후 왼쪽, 오른쪽 순서로 넣은 다음 차례대로 pop 하여 또다른 스택에 저장한다. 그리고 또다른 스택에 저장된 값을 pop 하면 post-order 방식으로 순회하게 된다.



다음은 Post-order 순회를 쉽게 이해하기 위해 그린 도식화 그림의 예시이다. 초록색으로 적힌 번호는 이해를 쉽게 하기 위해 임의로 붙인 숫자이며, 해당 이미지 노드의 고유 번호가 아니다. Post-order 순회의 경우에는 왼쪽 노드를 방문한 다음 오른쪽 노드를 방문하고 그 다음으로 자신의 노드를 방문하는 순으로 순회한다. 따라서 Post-order 순회 방식으로 노드의 값을 출력하면 1->4->5->3->9->8->6 순서가 된다.

ㄷ. Iter_Preorder

Iter_Postorder 와 마찬가지로 재귀함수를 사용하지 않고 Pre-order 방식의 순회를 하기 위해 스택에 push, pop 하는 과정을 수행한다. 차이점은 또다른 스택에 저장하지 않고 오른쪽 노드 다음 왼쪽 노드를 접근하도록 while 을 사용하여 계속 반복하다가 찾고자하는 고유 번호의 노드를 반환한다.



다음은 Pre-order 를 쉽게 이해하기 위해 그린 도식화 그림의 예시이다. 초록색으로 써져있는 번호는 파일고유번호가 아닌 이해를 돕기 위해 임의로 붙인 숫자이다. Pre-order 는 노드를 방문한 다음 왼쪽 노드, 오른쪽 노드 순으로 순회한다. 따라서 초록색으로 써져있는 숫자를 기준으로 BST 가 구현된다고 할 때 6->3->1->5->4->8->9 순으로 방문한다.

ㄹ. AddTreeNode

AddTreeNode 는 root 가 있을때와 없을 때를 구분하고, 없을 때는 해당 노드를 바로 root 에 저장한다. Root 가 있을 때는 이전 노드를 저장하고 새로 추가할 노드의 파일 고유 번호와 현재 방문하고 있는 노드의 파일 고유 번호를 비교하여 새로 추가할 파일 고유번호가 더 작으면 왼쪽으로, 더 크면 오른쪽으로 계속해서 이동하다가 현재 방문하고 있는 노드가 nullptr 이 되면, 이전에 방문한 노드에다가 새로운 노드값을 저장한다.

㉑. SearchImage

SearchImage 는 Iterative post-order 방식으로 순회하여 큐에 노드의 파일 이름과 고유번호에 대한 정보를 담고 빼내면서 이름기반 검색을 하는 함수이다. SearchImage 내에서 찾은 이미지 노드를 반환하고 나중에 SEARCH 함수에서 해당 노드를 인자로 넣어 보이어 무어 알고리즘을 사용한 파일 이름 탐색 기능을 진행한다.

㉒. Print

Print 는 In-order 방식으로 순회하며 트리 노드에 대한 정보를 출력한다.

㉓. MOVE

MOVE 함수에서는 링크드리스트에서 가장 마지막에 저장된 노드부터 차례대로 저장시키기 위해 저장할 폴더 리스트와 파일 노드를 설정하는 함수이다. 여기서 설정된 폴더 리스트와 파일 노드를 인자로 하여 addTreeNode 를 실행한다.

㉔. Table

Table 은 BoyerMoore 함수를 진행할 때 사용된다. 보이어무어 알고리즘에서는 파일 이름 탐색 시 최소한의 반복으로 단어를 비교하여 파일을 찾는데, 여기서 최고의 효율을 내기 위해 이동할 거리를 계산하기 위한 기초 단계로 Table 을 생성하여 해당 단어의 아스키값에 해당하는 자리에 단어의 인덱스를 넣는다. 더 자세한 설명은 3-3) Boyer-Moore algorithm 에서 하고자 한다.

㉕. BoyerMoore

BoyerMoore 함수는 보이어무어 알고리즘을 수행하는 함수로 파일 이름과 검색할 단어를 비교하고 효율적으로 단어를 이동시키면서 비교하는 함수이다. 자세한 설명은 Table 과 마찬가지로 3-3) Boyer-Moore algorithm 에서 하고자 한다.

㉖. BSTisEmpty

BSTisEmpty 함수는 root가 비어있을 때, 즉 BST가 없을 때를 확인하기 위해 만든 함수이다. 비어있는지 체크하여 비어있으면 동작하지 않도록 기능하는 함수들만 있기 때문에 비어 있는 경우에는 에러 코드를, 비어 있지 않을 때는 Success 를 반환하도록 설계한 함수이다.

3) Boyer-Moore algorithm

보이어 무어 알고리즘은 검색할 문자열의 왼쪽에서부터 오른쪽으로 진행하며 비교한다. 일치하지 않는 문자가 나타나면 정해진 규칙에 따라 오른쪽으로 이동해서 다시 진행한다. 모든 문자 지점을 시작으로하여 전체를 비교하는 방식이 아니라 규칙에 따라 효율적으로 움직이기 때문에, 성능이 훨씬 좋은 알고리즘이다. 불일치 패턴이 나타날 경우 오른쪽으로 얼마만큼 이동할 지를 미리 테이블로 작성해두는데 이

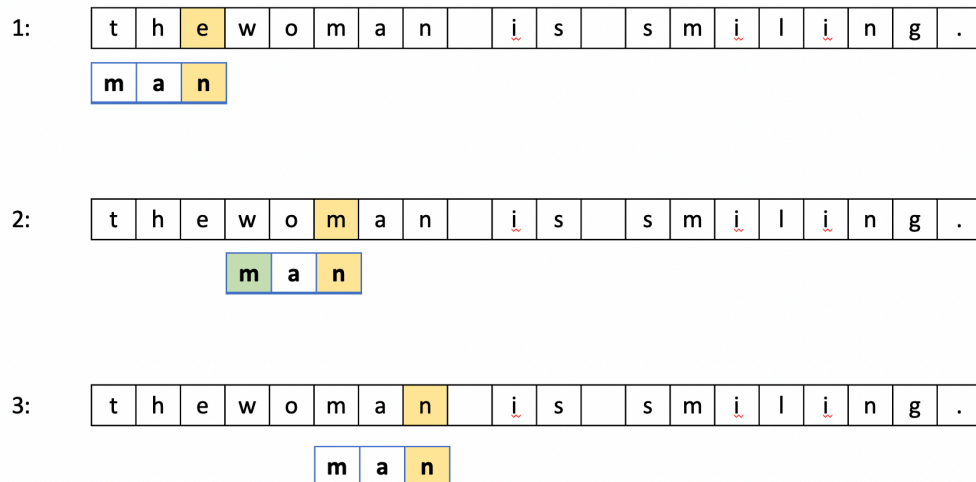
테이블을 스킵 테이블이라 하며 앞서 설명했던 Database_BST 클래스에 정의된 Table 함수가 이것이다. Skip 규칙은 다음과 같다.

1. 찾을 단어의 끝부분부터 비교를 시작한다.
2. 문자가 일치하면 계속해서 다음 문자를 비교한다.
3. 불일치하는 문자가 검색한 단어 문자열에 없으면 검색하는 단어의 길이만큼 이동한다.
4. 불일치하는 문자가 검색한 단어 문자열에 있으면 그 단어가 검색 단어 문자열 뒤에서부터 몇번째에 있는지 찾은 다음 그 수에서 -1 을 한 만큼 이동한다.
5. 처음 문자는 일치했는데, 계속해서 일치하는지 비교하는 과정에서 불일치하는 문자를 만난 경우에는 skip 테이블의 값에서 앞에서 일치한 수만큼 뺀 값을 이동시킨다.

예를 들어 the woman is smiling 이라는 문자열이 있고, 검색할 단어는 'man'이라고 할 때 skip 테이블은 다음과 같다.

m	a	n
2	1	0

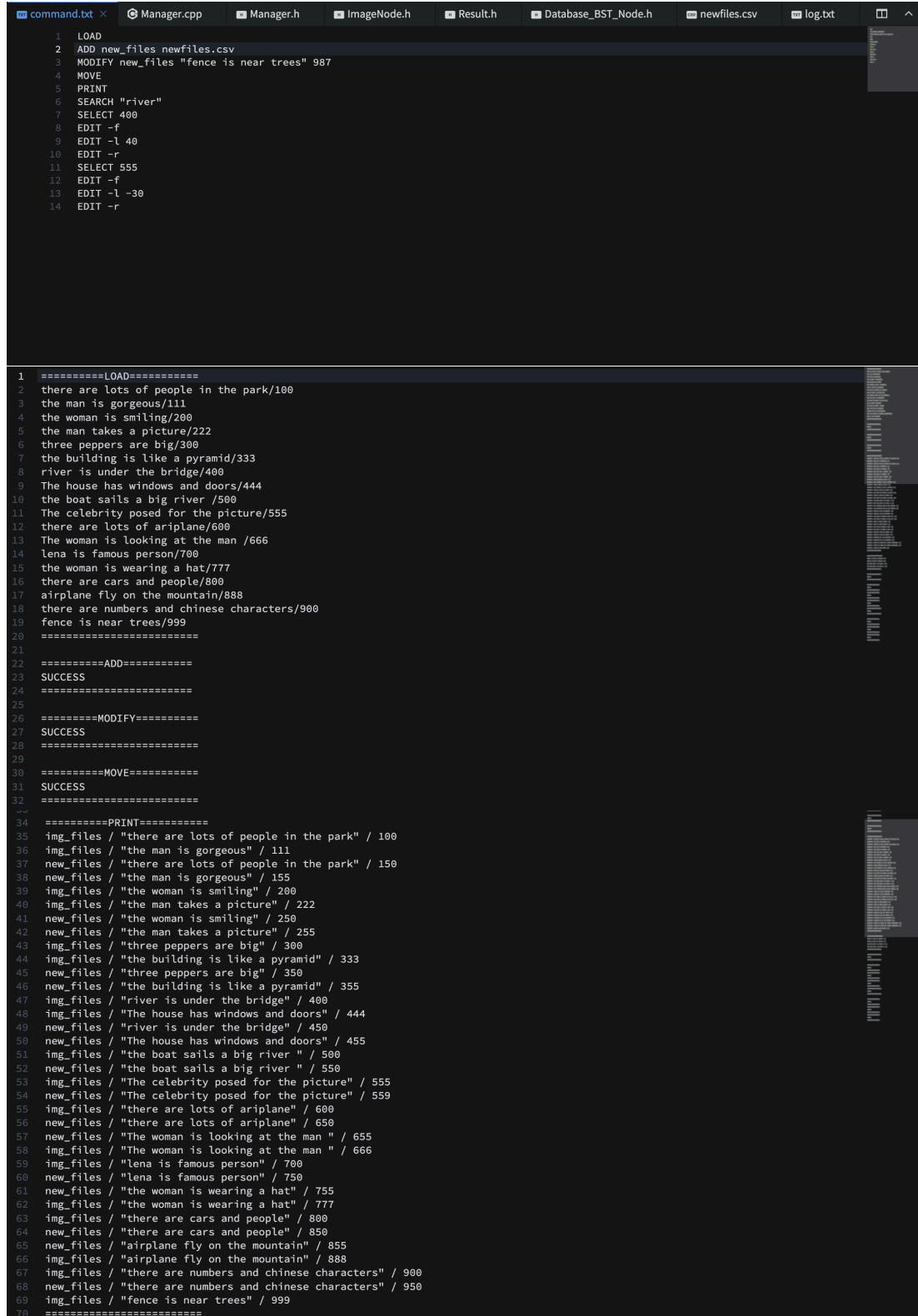
검색할 문자의 뒤부터 인덱스를 세기 때문에 맨 뒤에서부터 왼쪽으로 갈수록 인덱스가 커진다.



다음은 임의의 문자열과 검색할 단어 'man'을 비교하여 그린 그림이다. 1 번째는 검색할 단어와 비교할 문자중에 맨 오른쪽 단어가 일치하지 않고, 단어 문자열에도 존재하지 않는다. 따라서 검색하는 단어의 길이인 3 만큼 옆으로 이동하여 다시 비교한다. 따라서 2 번째 그림이 나타나게 되는데, 2 번째에서 보면 맨 끝 단어가 불일치하지만, 검색할 단어 문자열 중에 있기 때문에, 스킵 테이블 규칙 4 번에 따르면 불일치하는 문자는 스킵테이블 중 3 번째에 위치한 2 번 인덱스인 'm'과 일치하기 때문에 3-1 인 2 만큼 옆으로 이동한다. 따라서 3 번째 그림이 나타나게 되고 처음부터 끝까지 모든 문자열이 검색할 단어의 문자열과 일치하므로 해당 문장이 반환되게 된다.

4. Result Screen

Command.txt 는 다음과 같은 명령어를 적었을 때 log.txt 와 error.txt 는 다음과 같다.



The screenshot shows a code editor with two tabs: 'command.txt' and 'log.txt'. The 'command.txt' tab is active and displays a list of commands for a database system. The 'log.txt' tab is also visible, showing the output of these commands.

```
1 LOAD
2 ADD new_files newfiles.csv
3 MODIFY new_files "fence is near trees" 987
4 MOVE
5 PRINT
6 SEARCH "river"
7 SELECT 400
8 EDIT -f
9 EDIT -l 40
10 EDIT -r
11 SELECT 555
12 EDIT -f
13 EDIT -l -30
14 EDIT -r
```

The 'log.txt' tab shows the output of these commands, including file paths and counts. The output is as follows:

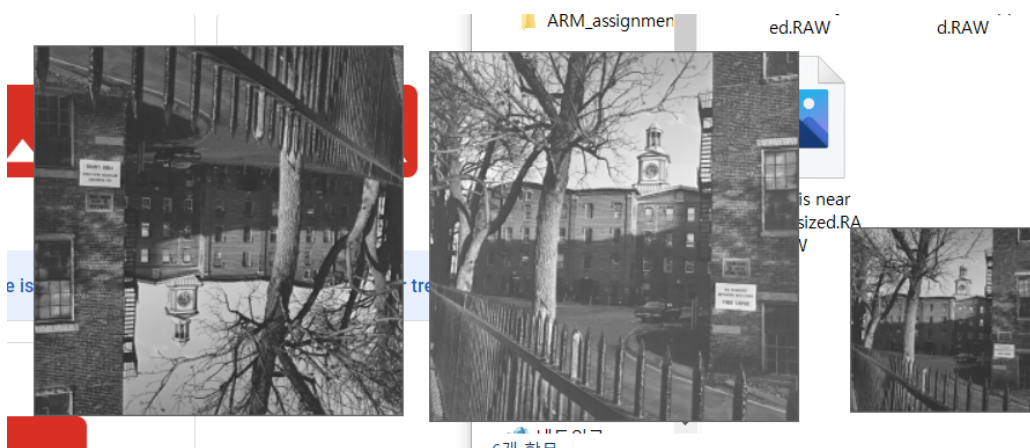
```
1 =====LOAD=====
2 there are lots of people in the park/100
3 the man is gorgeous/111
4 the woman is smiling/200
5 the man takes a picture/222
6 three peppers are big/300
7 the building is like a pyramid/333
8 river is under the bridge/400
9 The house has windows and doors/444
10 the boat sails a big river /500
11 The celebrity posed for the picture/555
12 there are lots of ariplane/600
13 The woman is looking at the man /666
14 lena is famous person/700
15 the woman is wearing a hat/777
16 there are cars and people/800
17 airplane fly on the mountain/888
18 there are numbers and chinese characters/900
19 fence is near trees/999
20 =====
21
22 =====ADD=====
23 SUCCESS
24 =====
25
26 =====MODIFY=====
27 SUCCESS
28 =====
29
30 =====MOVE=====
31 SUCCESS
32 =====
33
34 =====PRINT=====
35 img_files / "there are lots of people in the park" / 100
36 img_files / "the man is gorgeous" / 111
37 new_files / "there are lots of people in the park" / 150
38 new_files / "the man is gorgeous" / 155
39 img_files / "the woman is smiling" / 200
40 img_files / "the man takes a picture" / 222
41 new_files / "the woman is smiling" / 250
42 new_files / "the man takes a picture" / 255
43 img_files / "three peppers are big" / 300
44 img_files / "the building is like a pyramid" / 333
45 new_files / "three peppers are big" / 350
46 new_files / "the building is like a pyramid" / 355
47 img_files / "river is under the bridge" / 400
48 img_files / "The house has windows and doors" / 444
49 new_files / "river is under the bridge" / 450
50 new_files / "The house has windows and doors" / 455
51 img_files / "the boat sails a big river " / 500
52 new_files / "the boat sails a big river " / 550
53 img_files / "The celebrity posed for the picture" / 555
54 new_files / "The celebrity posed for the picture" / 559
55 img_files / "there are lots of ariplane" / 600
56 new_files / "there are lots of ariplane" / 650
57 new_files / "The woman is looking at the man " / 655
58 img_files / "The woman is looking at the man " / 666
59 img_files / "lena is famous person" / 700
60 new_files / "lena is famous person" / 750
61 new_files / "the woman is wearing a hat" / 755
62 img_files / "the woman is wearing a hat" / 777
63 img_files / "there are cars and people" / 800
64 new_files / "there are cars and people" / 850
65 new_files / "airplane fly on the mountain" / 855
66 img_files / "airplane fly on the mountain" / 888
67 img_files / "there are numbers and chinese characters" / 900
68 new_files / "there are numbers and chinese characters" / 950
69 img_files / "fence is near trees" / 999
70 =====
```

```

72 =====SEARCH=====
73 "river is under the bridge" / 400
74 "river is under the bridge" / 450
75 "the boat sails a big river " / 500
76 "the boat sails a big river " / 550
77 =====
78
79 =====SELECT=====
80 SUCCESS
81 =====
82
83 =====EDIT=====
84 success
85 =====
86 =====EDIT=====
87 success
88 =====
89 =====EDIT=====
90 success
91 =====
92 =====SELECT=====
93 SUCCESS
94 =====
95
96 =====EDIT=====
97 success
98 =====
99 =====EDIT=====
100 success
101 =====
102 =====EDIT=====
103 success
104 =====
105

```

LOAD 명령어가 잘 수행되었음을 확인할 수 있고, ADD 명령어는 수행이 잘 되는지 확인하기 위하여 임의로 폴더를 만들고, filenumbers.csv 에서 숫자만 바꿔 newfiles.csv 로 저장하였다. 그리고 ADD 명령어를 수행한 결과 제대로 동작되었음을 확인하였고, MODIFY 와 MOVE 역시 성공적으로 수행하였다. 그 다음 BST 에 제대로 된 값이 들어갔는지 확인하고자 PRINT 명령어를 사용하였고, 그 결과 로그 파일에 링크드리스트에 있던 모든 값들이 들어갔음을 확인하였다. SEARCH 명령어로는 "river"를 사용하여 검색하였고, filenumbers.csv 와 newfiles.csv 는 이름은 같지만 고유번호만 다른 파일이므로 번호는 다르지만 같은 이름인 노드가 2 번씩 검색될 것이라 예상하였고, 예상대로 결과값이 나왔음을 확인하였다. 또한 첫번째 사진과 두번째 사진 모두 사진을 비교하기 위해 하나의 사진으로 EDIT 의 세 명령어 -f, -l, -r 을 사용하였는데 첫번째 사진은 -l 에서 두번째 옵션으로 양수 값 뿐만 아니라 음수 값도 들어가는지 확인하기 위한 용도로 사용되었고, 그 결과 밝아지는 편집 뿐만 아니라 어둡게 하는 편집 역시 가능하다는 것을 확인할 수 있었다.



해당 결과 화면은 EDIT 명령어를 사용하여 하나의 이미지 파일을 -f, -l, -r 옵션을 사용하여 이미지 편집을 수행한 후 .RAW 파일을 여는 프로그램을 통해 변환된 사진을 열었을 때의 결과 화면이다. 이미지 점대칭과 밝기 변화, 1/4 사진 축소 모두 잘 편집되는 것을 확인할 수 있다.

5. Consideration

LOAD 명령어를 구현하고자 CSV 파일을 읽어올 때 문제점이 발생하였다. 파일의 고유 번호와 이름이 한 줄에 '를 기준으로 작성되어있는데 이를 어떻게 나눠서 사용해야하는지 몰라서 많은 고민을 하였다. 또한, stoi 함수를 사용하여 string 한 줄로 읽고 나눈 값을 정수로 변환하고자 하였는데 맨 처음 읽어오는 값이 100 이라 정수와 문자열을 구분할 수 없어 자꾸 오류가 났었다. 결과적으로는 istream과 getline 을 사용하여 string 배열에 각각 값을 넣어둔 다음 변수에 저장하는 식으로 문제를 해결하였다. ADD 명령어를 구현할 때에는 2D 링크드리스트로 연결하고자 링크드로 구성된 2 차원의 연결리스트를 구현하였는데, 그 과정에서 너무 많은 변수와 함수들이 사용되어 복잡해졌고, 오류가 발생하게 되어 문제점을 겪었다. 따라서 간결하게 코드를 사용하고자 할 수 있는 대부분의 중복되는 코드들은 함수로 따로 만들고 사용하는 작업을 수행하였다. MODIFY 명령어를 구현할 때는 노드를 교체해야했는데, 교체할 노드의 이전 값과 다음 값을 알고, 경우의 수를 따져서 새로운 노드로 교체해야해서 생각할 것이 많았다. 또한 MODIFY 명령어를 사용할 때 같이 입력되는 디렉토리명과 파일명, 그리고 바꾸고자 하는 숫자를 구분할 때 파일명이 "로 둘러싸여 있어서 구분하는데 있어 시간이 조금 걸렸다. 하지만 "를 기준으로 구분하는것 역시 LOAD 명령어에서 CSV 파일을 읽어올 때 ,를 기준으로 나누었듯이 istream과 getline 을 사용하여 공백과 큰따옴표를 기준으로 잘라 변수를 구분하였다. MOVE 명령어를 구현할 때에는 링크드리스트에 가장 마지막으로 놓여진 노드부터 차례대로 Binary Search Tree 에 저장되어야 했으므로 마지막 노드를 구하는 것이 필요했다. 하지만 마지막 노드를 구했어도 마지막 이전에 들어온 노드를 구하는 것도 쉽지 않았고, 2 차원 링크드리스트를 일반적인 배열 형태가 아닌 노드 포인터로 연결된 링크드리스트를 사용하여 2 차원을 구현하였기 때문에 맨 처음 디렉토리 노드와 바로 연결되어있는 첫번째 이미지 노드에 접근할 때에는 문제가 계속해서 생겼다. 또한, 마지막 노드부터 차례대로 MOVE 하여 BST 에 넣으면 넣고 더이상 필요 없어진 노드는 삭제했어야 했는데, 이번에도 역시 삭제할 노드가 디렉토리 노드와 바로 연결되어 있는 첫번째 이미지 노드일 경우에는 삭제 후에도 디렉토리 노드가 남아있어 문제가 생겼다. 따라서 이미지 노드가 모두 삭제되어 빈 디렉토리 리스트만 남아있는 경우 파일 리스트도 삭제하는 것을 따로 구현하였다. 여기까지 LOAD, ADD, MODIFY, MOVE, PRINT 까지 총 5 개의 명령어를 구현한 뒤 리눅스 환경에서 제대로 동작하는지 확인하기 위하여 파일을 옮기고 리눅스에 맞는 코드로 작성하는 과정에서 리눅스에 대한 이해가 부족하여 문제점이 자주 발생하였다. MakeFile 이 무엇인지 이해하는 과정부터가 필요했고, g++을 어떤 명령어를 사용하여 실행할 수 있는지 몰라서 헤맸었다. 하지만 결국엔 해결하여 실행까지는 문제없이 완료할 수 있었는데, 파일이 실행되면서 세그먼트 오류가 발생하였다. 즉, 메모리 누수가 발생했다는 것이다. 어디서부터 잘못된 것인지 확인하기에는 명령어를 처음부터 나눠서 코드를 작성하지 않고, main.cpp 에다 모든 명령어들과 클래스들을 한번에 넣어 구현하였기 때문에 잘못된 부분을 찾는 것이 어려웠다. 또한, 어디서부터 어디까지가 관련된 함수인지 구분하는것도 어려워 코드를 이해하는데 시간을 소모하였다. 결국에는 구글링을 통해 리눅스에서 세그먼트 펄스를 확인하는 방법을 찾아서 gdb 를 사용하고,

breakpoint 를 통해 세그먼트가 발생하는 관련 함수 위치를 확인하였다. 그 다음에는 중단점을 활용하여 디버깅을 하고 오류가 발생하는 부분을 찾았다. 원인은 링크드리스트의 노드를 addNode 함수를 사용하여 추가할 때 생성자에 next 값을 nullptr로 초기화하지 않았기 때문이었다. 대부분의 함수들은 각 상황에 맞는 currNode를 설정하고 해당 노드가 nullptr를 만나기 전까지를 조건으로 하여 while문이 돌아가는 경우가 많았는데, next 포인터를 nullptr로 설정하지 않고 링크드리스트 노드를 추가하니 쓰레기값이 들어가게되면서 메모리 누수가 발생하게 된 것이었다. 이 경험을 통해 앞으로는 기능이 비슷한 함수들끼리 모아서 바로바로 cpp 파일과 h 파일을 나눠서 작성해야 된다는 것과 메모리 누수가 발생하지 않도록 생성자와 소멸자, 초기화 부분을 잘 확인해야 된다는 것을 알게되었다. 이런 습관을 가지려고 노력한다면 오류가 발생하는 횟수도 많이 줄어들 것이고 다음에 내가 짠 코드를 다시 보더라도 어디까지 했는지, 어떤 함수가 어떤 기능을 하는지 바로 확인할 수 있어 시간의 효율성을 느낄 수 있을 것이라 기대한다. 다음과 같은 문제점을 겪은 후로는 바로 리눅스 환경에서도 코드가 작동할 수 있도록 goorm idle를 사용하였다. 리눅스 환경을 맞춘 후 해당 idle에서 코드를 작성하였는데, SEARCH 명령어를 수행할 때 queue와 stack에 대한 이해가 부족하여 오류가 많이 발생하였다. 특히 항상 순회를 할 때 재귀함수를 사용하였었는데 반복문을 사용하여 순회하려고 하니 어려웠었다. Iterative Post Order 함수를 만들 때는 두 개의 스택을 사용하였는데, 더 좋은 방법이 있지 않을까 생각해본다. SELECT와 EDIT 함수를 만들 때는 큰 문제점은 없었다. 전체적으로 클래스들과 함수들을 .cpp와 .h로 나누고 #include를 사용하여 관련있는 파일끼리 묶는것이 조금 어려웠다. 그리고 해당 결과값을 항상 콘솔을 활용하여 확인하고, 입력도 콘솔을 사용하였는데, 이를 command, log, error 텍스트 파일을 사용하려고 하니 오류가 발생했었다. 하지만, 파일 입출력에 대해서 다시 공부하고 진행하였더니 문제없이 해결할 수 있었다. 이번 프로젝트를 통해 링크드리스트, 큐, 스택, 보이어무어 알고리즘, BST 등 다양한 자료구조에 대해 알고 구현해보면서 배워갈 수 있어서 좋은 프로젝트가 되었다고 판단한다.