데이터구조 및 설계 1차 프로젝트 보고서

2019202050 이강현

1. Introduction

이번 프로젝트는 입력받은 명령어에 기반하여 사진들의 대한 데이터를 linked list, binary search tree를 이용하여 관리하며 사진들을 편집하는 프로그램을 구현하는 것이 목표이다. 간략하게 프로젝트에서 구현하여야 하는 흐름을 보자.

LOAD명령어로 경로내의 폴더 속 사진들의 데이터가 저장되어있는 csv파일을 읽어 linked list 구조내에 저장한다.

ADD로 LOAD에서 불러온 데이터 이외의 파일들을 추가할 수 있도록 한다.

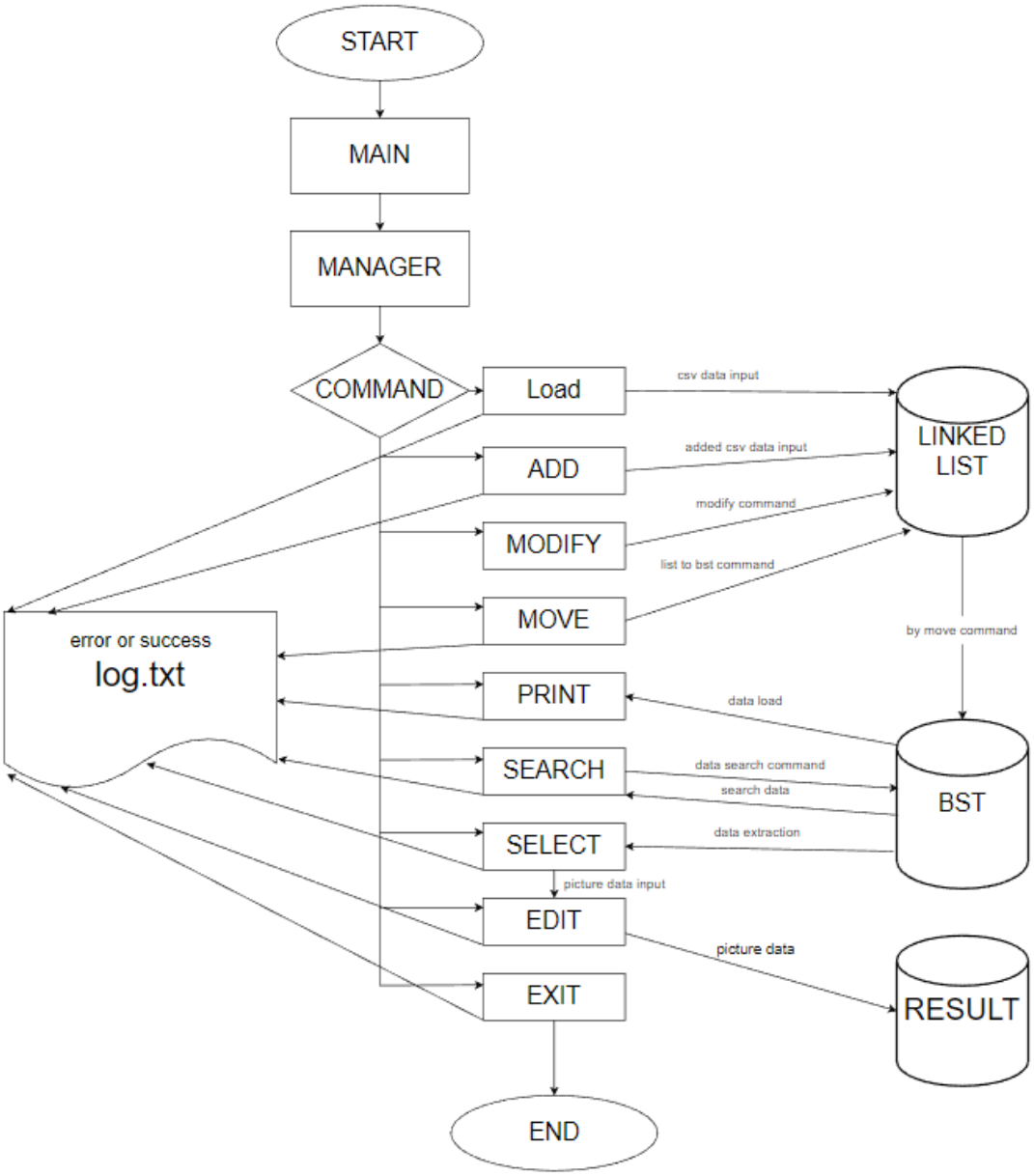
MODIFY로 Linked list 내 파일들의 데이터를 수정할 수 있고 정제된 데이터는 최종적으로 MOVE명령어를 통해 Binary search tree 구조로 이동된다.

PRINT명령어를 통해 binary search tree 내 데이터들을 확인할 수 있고 SEARCH명령어를 통해 어떠한 사진들이 있는지 확인할 수 있다.

다음으로 사진을 편집하기위해 SELECT명령어로 사진을 선택하고 이를 EDIT명령어로 사진의 크기,밝기,점대칭과 같은 작업을 마친 후 디렉토리에 결과물을 저장할 수 있다.

이와 같은 흐름으로 프로그램을 구현해야 하며 핵심 저장구조인 Linked list와 binary search tree를 알맞게 구현하고 관리할 줄 알며 queue나 stack과 같은 자료구조, 탐색을 위한 알고리즘들을 적절히 활용할 줄 아는 것이 이 프로젝트의 과제이다.

2. Flowchart



전체적인 이번 프로젝트의 흐름도이다. Command.txt를 통해 명령어를 입력받고 이를 manager.h에서 관리한다. 따라서 command에 맞는 행동을 취하며 다양한 명령어에 맞게 위의 흐름도와 같이 작동한다. 모든 명령어들은 행동을 마치면 종료되지 않고 command파일이 끝까지 읽힐 때까지 작동한다. 그에 대한 흐름을 log.txt에서 직접 파악할 수 있으며 각각의 성공여부 또한 log.txt에서 파악할 수 있다.

구현한 코드흐름에 따라 설명하면 우선 main 함수에서 프로그램이 실행된다. 그 이후 manager 객체를 생성하고 그 안의 Run함수를 실행하면 command.txt

파일의 내용들을 getline함수를 통해 한 line씩 읽는다.

그 후 line에서 어떤 명령어가 입력되었는지를 판단하기 위해 strtok함수를 통해 띄어쓰기를 구분자로 하여 라인에서 추출한 후 추출한 단어를 각 명령어와 조건문을 사용해 비교한 후 해당 명령을 실행한다.

우선 LOAD에서는 정해진 csv파일 경로에서 데이터를 가져오기 위해 input stream을 열고 여기서도 한line씩 받아온다. 그후 받아온 데이터를 정제하고 nodelist.h에 선언되어 있는 Loaded_List 객체를 생성하여 데이터를 1차적으로 담는다. 이때 데이터가 100개가 넘는 것을 방지하여 nodeisfull함수를 만들어 두었으며 이는 가장 일찍 들어온데이터를 삭제하는 형식으로 구현되어 있다.

ADD에서는 LOAD와 비슷한 방식으로 구현되어있으면 가장 큰 차이점을 새로운 리스트를 생성하여 LOAD에서 생성된 리스트 아래에 2차원으로 붙인다는 차이점을 지닌다.

MODIFY에서는 Linked list에 접근하여 반복문을 통해 수정하려는 고유번호가 기존에 리스트에 존재하는지를 우선 확인하고 없다면 디렉토리명이 같은 리스트를 검색한다. 해당리스트에 도착하면 그중 입력받은 이름과 같은 노드를 삭제하고 그 자리에 새로운 노드를 생성해 끼워넣는다.

MOVE는 우선 linked list의 최하단 리스트에 접근하고 그중 제일 마지막 노드에 접근하여 해당 노드의 데이터는 bst 구조에 삽입하고 리스트의 노드는 삭제된다. 이를 반복하면 리스트의 노드들은 모두 삭제되게 되고 bst의 구조만 남는다. Bst는 300개의 데이터를 받을 수 있으며 따라서 또다시 추가하고 싶은 데이터가 있다면 LOAD와 ADD로 데이터를 추가할 수 있다. 300개의 노드가 넘게 되면 bst는 가장 낮은 고유번호부터 삭제하게 된다.

PRINT는 inorder방식으로 노드에 접근하여 데이터를 log파일에 출력한다.

SEARCH는 스택을 통해 iterative postorder방식으로 노드의 데이터를 정렬한뒤 큐에서 순차적으로 데이터를 사용할 수 있게 한다. 이때 큐에서 순차적으로 나오는 데이터들을 하나하나 보이어 알고리즘을 통해 검사하면서 원하는 단어를 가지고 있는 노드를 출력한다.

SELECT는 편집을 원하는 사진의 고유번호를 인자로 받는다. 인자로 받은 고유번호는 문자열이기에 숫자로 바꾸어주는 atoi함수를 사용하였고 preorder방식으로 노드들을 순회하여 일치하는 노드를 찾으면 selectfile이라는 포인터에 노드

를 가리키게끔 구현하였다.

EDIT에서는 SELECT에서 찾은 노드를 가리키게끔 한 selectfile 이라는 포인터를 통해 데이터를 불러오며 이때 fread를 통해 사진을 픽셀단위로 가져오고 이를 2차원배열에 저장한다. 조건문의 분기를 통해 입력받은 명령어 인자에 맞게 flip,resize,adjust과 같은 사진 편집을 실행한다. 그 후 2차원배열의 저장되어있는 픽셀값들을 fwrite함수를 통해 Result 디렉토리에 저장한다.

EXIT에서는 linked list와 bst에서 동적할당하여 사용한 메모리를 해제해주는 작업을 가진다. MOVE명령어를 통하여 linked list의 메모리는 해제되지만 bst의 메모리는 프로그램의 마지막까지 사용하므로 해제해주는 코드를 구현하였다. 추가적으로 LOAD ADD를 시행하였으나 MOVE를 하지 않고 프로그램이 종료되는 상황을 고려해 조건문을 통해 Linked list가 존재하면 지우도록 구현하였다.

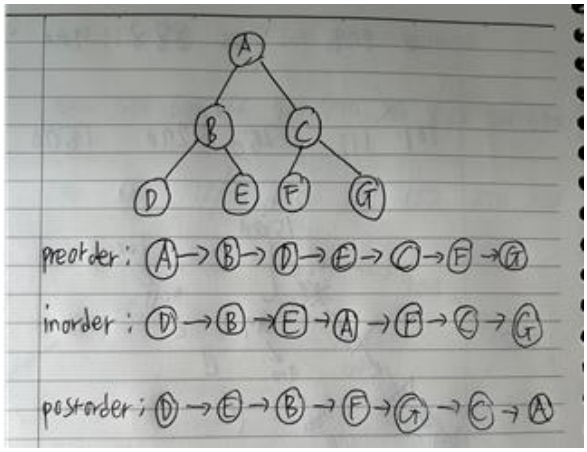
잘못된 명령어가 입력되면 777에러코드를 출력하고 다음 명령어를 받는다.

3. Algorithm

이번 프로젝트에서 사용된 알고리즘은 우선 연결구조에 대한 알고리즘이 중심이 된다. Linked list와 bst구조는 노드속 데이터를 가지고 다음 노드를 가리킬 수 있는 형식이다. 따라서 서로 어디 있든지 주소만 저장되어 있다면 접근이 가능하고 가리키고 있다라는 형식덕분에 직관적으로 노드간의 관계성을 확인하는데 도움이된다. 추가적으로 bst는 노드를 가리키는 방식에서 차이가 있는데 대소를 비교하여 생성된 구조이므로 일렬로 이어진 구조보다 탐색속도에서 더욱 좋은 모습을 보여준다.

특히 이번 프로젝트에서는 binary search tree에서 다양한 탐색 알고리즘을 사용하였다.

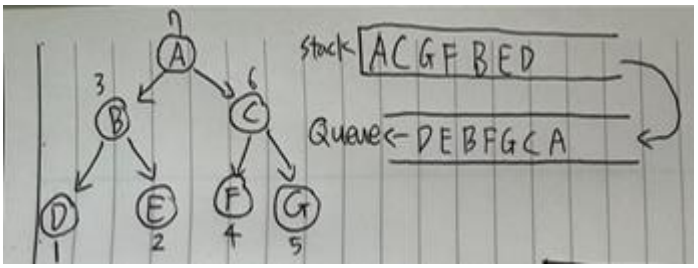
Preorder, inorder, postorder가 모두 사용되면 이는 노드 탐색의 순서에 따라 나뉜다.



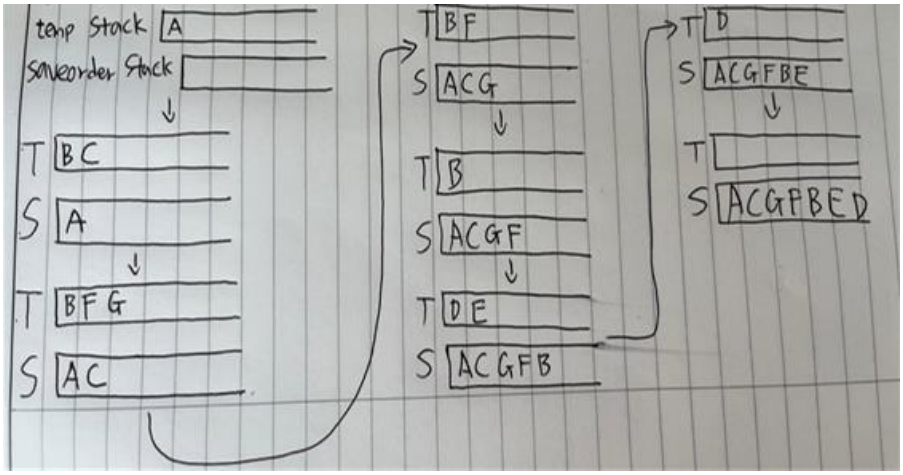
위의 사진과 같은 탐색방법들이 사용되었다.

Queue과 stack 또한 사용되는데 이는 각각 FIFO(first-in,first-out)과 LIFO(last-in, first-out)의 구조이다. 이번 프로젝트에서 stack과 queue의 사용이 극대화 된 부분이 있는데 바로 SEARCH명령어에서 iterative postorder를 이용하여 큐에 데이터를 저장한 후 boyer algorithm을 통한 문자열 비교이다.

우선 iterative postorder는 아래 사진과 같이 작동한다.



postorder탐색의 결과를 얻으려면 사진의 트리에 적힌 번호 순으로 큐에 저장되어야 한다. 따라서 스택에는 역순으로 저장되어야 함을 알 수 있다. 재귀함수를 사용하지 않고 반복문을 통해 이와 같은 결과를 얻기 위해 스택을 2개 사용하여 다음 사진과 같이 접근하였다.

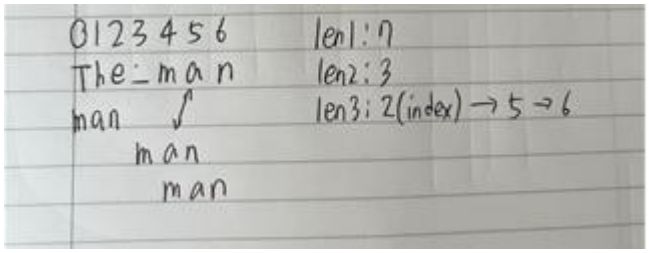


재귀함수를 사용하지 탐색은 노드 탐색 순서를 기억하면서 가지만 반복문을 통한 구현은 순서를 하나하나 저장해두기에는 무리가 있다. 트리의 깊이가 깊을수록 탐색에 어려움이 있을 것이다. 하지만 stack 구조를 사용하여 순서를 저장해둔다면 쉽게 구현할 수 있다. 반복문은 temp stack이 비어질 때까지 반복하며 temp stack에 현재 노드의 위치를 저장하고 이를 pop함과 동시에 saveorder stack에 push하고 현재노드의 left와 right를 다시 temp stack에 넣는 방법을 사용하였다. 이는 스택의 특징인 LIFO을 이용한 방법이며 postorder의 탐색 특성상 왼쪽 트리의 자식이 최우선으로 탐색되므로 역순으로 오른쪽 트리의 자식을 먼저 탐색하여 최종적으로 큐 구조에는 postorder탐색 결과가 저장된다.

다음은 boyer algorithm이다.

Boyer algorithm은 문자열 비교를 위한 알고리즘이다. 문자열속 단어를 찾는 것인데 다음과 같은 방식을 가진다.

1. 단어의 끝과 문자열을 비교한다.
2. 비교 후 같다면 차례대로 단어와 문자열 인덱스를 줄여가며 비교한다.
3. 그 이후 달라진다면 단어의 길이에서 같았던 문자 수만큼을 빼고 skip한다.
4. 단어의 끝이 다르다면 단어의 인덱스만 차례대로 문자열의 문자와 비교한 후 같은 단어가 나온다면 위치를 맞춰주고 아니라면 단어의 길이만큼 skip한다.



Len1은 문자열의 길이 len2는 단어의 길이 len3는 문자열에서 처음 비교할 문자의 인덱스이다.

인덱스 2에서 문자끼리 비교한 후 다르고 단어와 같은 문자가 없으므로 단어 길이만큼 skip한다. 인덱스 5에서는 문자는 다르지만 탐색 후 같은 단어 a를 찾았으므로 a끼리 서로 맞춰준다. 그 결과 man을 찾을 수 있다.

반복문을 통해 구현하였고 len3가 len1보다 작을 때만 작동하게끔 구현하였다.

마지막으로 EDIT에서 사진 편집을 위해 사용된 algorithm이다.

우선 EDIT에서 사진을 편집하는 과정은 for문을 이중으로 사용하여 구현하였고 그 과정에서 사진을 점대칭하는 것은 사진의 모든 픽셀들을 스택에 넣고 다시 꺼내면 역순으로 나오므로 스택을 사용하여 구현하였다. 밝기를 조절하는 것은 큐를 사용하여 하나씩 pop해가며 밝기를 올려주는 것으로 구현하였고 해상도를 낮추는 작업은 반복문의 증감문을 수정하여 256*256이 128*128이 되게끔 하고 인접한 픽셀 4개를 합쳐 평균을 구하는 것으로 구현하였다.

4. Result screen

프로젝트 파일에서 make명령어로 컴파일 후 실행하는모습

```
(base) kang@ubuntu:~/Desktop/data_structure_design/project1$ ls
bst.cpp       command.txt    imagestack.h  Makefile      nodelist.cpp  run
bst.h         imagenode.cpp  img_files     manager.cpp   nodelist.h    stack.h
bstnode.cpp   imagenode.h    log.txt       manager.h     queue.h
bstnode.h     imagequeue.h   main.cpp      new_files     Result
(base) kang@ubuntu:~/Desktop/data_structure_design/project1$ make
g++ -std=c++11 -g -o run manager.cpp bst.cpp bstnode.cpp nodelist.cpp imagenode.
cpp main.cpp imagenode.h imagequeue.h imagestack.h queue.h nodelist.h bst.h bstn
ode.h manager.h stack.h
./run(base) kang@ubuntu:~/Desktop/data_structure_design/project1$ ./run
(base) kang@ubuntu:~/Desktop/data_structure_design/project1$
```

명령어는 다음과 같다.

```
LOAD
ADD new_files new_filenumbers.csv
MODIFY img_files "the building is like a pyramid" 506
MODIFY new_files "the man takes a picture" 404
MOVE
PRINT
SEARCH "s a"
SELECT 404
EDIT -f
EDIT -l 200
EDIT -r
EXIT
```

csv파일은 아래와 같이 설정했다.

Img_files는 기존의 파일 즉, LOAD를 통해 불러와지는 파일이다.

```
700,lena is famous person.RAW
777,the woman is wearing a hat.RAW
800,there are cars and people.RAW
888,airplane fly on the mountain.RAW
900,there are numbers and chinese characters.RAW
999,fence is near trees.RAW
~
```

New_files는 ADD명령어를 통해 추가되는 파일이다.

```
101,there are lots of people in the park.RAW
112,the man is gorgeous.RAW
250,the woman is smiling.RAW
270,the man takes a picture.RAW
1500,the man akes a picture.RAW
~
~
```

Log.txt를 열어보면

```
=======LOAD========
(node.m_name: lena is famous person/img_files/700)
(node.m_name: the woman is wearing a hat/img_files/777)
(node.m_name: there are cars and people/img_files/800)
(node.m_name: airplane fly on the mountain/img_files/888)
(node.m_name: there are numbers and chinese characters/img_files/900)
(node.m_name: fence is near trees/img_files/999)
====================
```

```
========ADD========
SUCCESS
====================
========ERROR========
300
====================
=======MODIFY=======
SUCCESS
====================
========MOVE========
SUCCESS
====================
========PRINT========
(bstnode.m_name: there are lots of people in the park/new_files/101)
(bstnode.m_name: the man is gorgeous/new_files/112)
(bstnode.m_name: the woman is smiling/new_files/250)
(bstnode.m_name: the man takes a picture/new_files/404)
(bstnode.m_name: lena is famous person/img_files/700)
(bstnode.m_name: the woman is wearing a hat/img_files/777)
(bstnode.m_name: there are cars and people/img_files/800)
(bstnode.m_name: airplane fly on the mountain/img_files/888)
(bstnode.m_name: there are numbers and chinese characters/img_files/900)
(bstnode.m_name: fence is near trees/img_files/999)
(bstnode.m_name: the man akes a picture/new_files/1500)
====================
=======SEARCH=======
(bstnode.m_name: there are cars and people/img_files/800)

(bstnode.m_name: there are numbers and chinese characters/img_files/900)

(bstnode.m_name: the man takes a picture/new_files/404)

(bstnode.m_name: the man akes a picture/new_files/1500)

SUCCESS
```

```
=======SELECT=======
SUCCESS
====================
========EDIT========
SUCCESS
====================
========EDIT========
SUCCESS
====================
========EDIT========
SUCCESS
====================
========EXIT========
SUCCESS
=====================
```

Log.txt의 첫번째 줄부터 살펴보면 LOAD명령어가 잘 실행되어 SUCCESS를 나타내는 것을 볼 수 있다. 그 후 실행된 ADD또한 잘 실행되었고 추가적으로 노

드가 100개 이상이 되는 것을 막기위해 설정한 변수값을 조절하여 10만 받도록 수정하였을 때

```
========PRINT========
(bstnode.m_name: there are lots of people in the park/new_files/101)
(bstnode.m_name: the man is gorgeous/new_files/112)
(bstnode.m_name: the woman is smiling/new_files/250)
(bstnode.m_name: the man takes a picture/new_files/404)
(bstnode.m_name: the woman is wearing a hat/img_files/777)
(bstnode.m_name: there are cars and people/img_files/800)
(bstnode.m_name: airplane fly on the mountain/img_files/888)
(bstnode.m_name: there are numbers and chinese characters/img_files/900)
(bstnode.m_name: fence is near trees/img_files/999)
(bstnode.m_name: the man akes a picture/new_files/1500)
====================
```

이와 같이 가장 처음 입력된 img_files의 700의 고유번호를 가진 노드가 삭제된 것과 10개만 저장되어 있는 것을 확인할 수 있다.

또한 bst구조에서도 MOVE를 통해 계속 데이터를 입력받다 300개 이상을 받으려할 때를 고려하여 295개의 노드가 차있고 5개만 받아야한다고 일시적으로 수정해서 확인한 결과 상위 큰 수들 1500,999,900,888만 남고 그전에 저장되어 있던 800이 제일 작으므로 지워지고 700이 들어온 것을 보아 정상적으로 코드가 구현됨을 알 수 있다.

```
========PRINT========
(bstnode.m_name: lena is famous person/img_files/700)
(bstnode.m_name: airplane fly on the mountain/img_files/888)
(bstnode.m_name: there are numbers and chinese characters/img_files/900)
(bstnode.m_name: fence is near trees/img_files/999)
(bstnode.m_name: the man akes a picture/new_files/1500)
====================
```

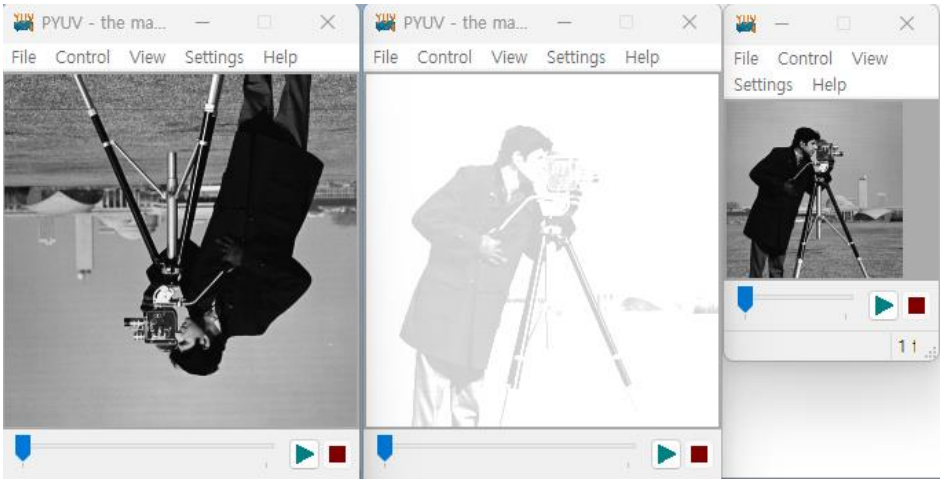
MODIFY 명령어에서 img_files에 있는 "the building is like a pyramid"를 506으로 수정되는 부분은 img_files csv파일에 해당 데이터가 없으므로 오류를 출력하였고 new_files에 "the man takes a picture"를 404로 바꾸라는 명령어는 잘 실행되어 기존의 270의 고유번호를 지니던 파일데이터가 그 다음 PRINT 명령어에서 404로 잘 바뀌어 있는 것을 log에서 확인할 수 있다. 또한 MOVE명령어 또한 잘 실행되어 bst 구조체계로 데이터가 넘어간 것을 확인할 수 있고 그에 따라 inorder로 탐색하여 출력하는 PRINT 명령어도 잘 실행되어 노드의 데이터들을 log에 저장한 것을 확인할 수 있다.

SEARCH 명령어 또한 "s a"문자열이 들어간 데이터들만 잘 간추려서 log에 저장

한 것을 결과화면을 통해 확인할 수 있다.

SELECT 명령어는 아무 노드도 찾지 못하였을 때 처리한 예외처리와는 다르게 파일에 잘 접근하여 아래 EDIT의 결과가 나타났기 때문에 정상적으로 처리된 것을 알 수 있다.

EDIT한 결과는 다음과 같다. 왼쪽부터 flip,adjust,resize이다.



이 수정된 사진들은 프로젝트 pdf에 맞게 "Result"라는 디렉토리에 저장되므로 디렉토리를 만든 상태를 초기로 구성하였다.

EXIT명령어 또한 잘 실행되었다.

5. Consideration

이번 프로젝트에서는 여러가지 명령어를 처리하는 과정에서 다양한 알고리즘들을 접하면서 새로이 공부할 수 있는 내용들이 많았고 기존의 visual studio에서 구현하던 환경과 다른 리눅스 환경에서 코딩을 하려니 많은 어려움이 있었다. 특히 리눅스 환경에서의 디버깅은 make파일과 콘솔창에서 나오는 오류를 기반으로 디버깅 해야했기 때문에 시간이 많이 들었고 세그먼트 오류는 어디서 문제가 생기는 지 알 수 없어 함수내 혹은 라인마다 cout을 통해 시각화하여 디버깅을 했어야 했다. 또한 cpp파일 하나로만 코딩해왔었지만 파일들을 용도에 맞게 분할하고 서로 선언하며 사용하는 것 또한 감을 잡지 못해 어려웠다.

LOAD와 ADD에서는 마찬가지로 코드를 구현하는 첫 부분이었고 linked list와 manager를 나눠서 작업하는게 낯설어서 어려웠다. 또, csv파일 앞부분에 BOM이 있어 첫번째 데이터의 고유번호만 이상한 값이 들어가 최종적으로 bst에서 대소비

교가 올바르게 되지 않는 문제점이 있었다. 이슈리뷰결과 BOM을 포커싱하여 채점하지 않고 파일에서 데이터를 잘 불러오기만 하면 된다는 조교님의 말씀과 UTF-16이나 UTF-32 인코딩은 문서해석에 BOM이 결정적인 역할을 하나 UTF-8에서는 그저 signature 역할을 하고 있다는 구글링 결과에 따라 BOM을 사용하지 않는 csv로 수정하여 코드를 구현하였다.

MODIFY명령어에서는 명령어 인자를 큰따옴표를 활용해서 받아야했고 이 때문에 strtok을 쓸 때 어떻게 써야하는지 고민을 많이 했다. 결국 ₩"를 이용하여 나눠 받았고 또한 숫자를 문자열로 처음에 저장했으나 binary search tree 구조에서 문자열끼리 비교할 때 사전식으로 비교가 되므로 1500이 200보다 더 작다고 판단하는 결과를 낳아 int로 수정하는 과정을 거쳤다. string에서 int로 바뀌는 함수가 작동되지 않아 c.str()을 통해 char형태로 바꾼 후 atoi함수를 사용하였다.

bst탐색 부분에서 대부분 재귀함수로 구현하였지만 iterative postorder는 처음보는 알고리즘이라 어려움이 많았다. 재귀함수를 사용하지 않고 스택을 이용하는 방법이라 감잡기가 어려웠고 하나하나 그려가면서 흐름을 이해하는 것이 필요했다. 따라서 재귀함수를 이용한 방법으로 형성된 bst에서 어떤 순서로 코드가 진행되어야 하는지 미리 알고 역순으로 스택에 어떤 값이 저장되어야 하는지 계산했다. 그렇게 역순으로 그림을 그려서 이해할 수 있었다. EDIT에서는 밝기를 조절하는 부분을 구현할 때 스택과 큐는 템플릿으로 사용하여 예외시에 NULL값을 반환하게끔 구현했는데 이를 SELECT에서는 노드포인터를 반환하고 EDIT에서는 unsigned char를 반환해야 해서 unsigned char는 non-pointer형이라는 오류가 생겼다. 따라서 image편집용 스택과 큐를 새로 구현했다. 마지막으로 메모리를 해제하는 부분에서 일반적으로 linked list는 MOVE를 통해 메모리를 모두 해제하지만 LOAD와 ADD를 하고 MOVE를 처리하지 않은 코드가 종료될때를 대비해서 EXIT에 넣어두었다. 그러나 정상적인 상황에서는 두번 포인터가 FREE된다는 오류가 발생하여 직접 리스트와 BST를 확인하는 조건을 달아주어 해결하였다.

정말 다양한 명령어들을 통해 프로그램을 구현해보았는데 구현을 점차 해나가면서 들었던 생각은 코드를 구현하기전에 전체적인 흐름을 생각하고 만드는 것이 중요하고 코드가 길어지면 길어질수록 더욱 디버깅하기 힘들고 알아보기 힘들기에 코드를 재사용하는 것이 중요하다는 것을 배웠다. 또한 파일을 분할하여 역할에 맞게 구현하는 것이 효율적이라는 것을 느꼈다.