데이터구조설계/실습 보고서

Project #1

제출 일자: 2022년 10월 13일 (금)

학 과: 컴퓨터정보공학부

담당교수: 이기훈교수님

학 번: 2019202021

성 명: 정 성 엽

1. Introduction

A. 제목

Project #1 Raw file 편집

B. 목적

링크드 리스트(Linked List), 이진 탐색 트리(Binary Search Tree), 큐(Queue), 스택(stack) 데이터 구조를 사용하여 csv파일로 부터 읽은 raw file의 이름, 고유번호를 정렬하고 후에 이진 탐색 트리를 통해 찾은 선택값을 통해 raw file을 편집하는 프로그램을 제작한다.

C. 데이터 구조

i. Linked List

- LOAD 또는 ADD 명령어를 통해 csv파일로부터 raw file의 이름, 고유번호를 받고 해당 directory에 대해 linked list 통해 연결한다. 100개가 넘을 경우 가장 오래된 Node를 삭제하고 새로 받는다.

ii. Binary Search Tree

- MOVE 명령어를 통해 이미 만들어진 linked list로부터 데이터를 가져와 BST로 저장된다. 저장되는 순서는 Linked List에서 가장 마지막에 제작된 순서로 불러온다. 저장되는 위치의 관계는 각 파일의 고유번호를 기준으로 하며 이때 고유 번호는 중복되지 않는다.

- 비교하는 BST의 고유 번호보다 새로 저장 할 BST의 고유 번호가 더 큰 경우 해당 Tree의 upper_next로 이동하도록 한다. 아니라면 lower_next 로 이동하도록 한다.

iii. Queue

- SEARCH 명령어를 통해 이미 만들어진 Binary Search Tree 에서 post-order 순으로 queue에 저장되도록 한다. Queue의 각 노드는 raw file의 이름 , 고유 번호, 그리고 directory를 저장하고 있다.

- EDIT 명령어를 통해 선택한 raw_file에 저장된 각 픽셀 값을 queue 형식으로 받아서 다시 pop할 때 밝기에 변화를 주어 저장한다.

iv. Stack

- SEARCH 명령어 사용 시 post-order 구현을 위해 stack 구조를 사용한다. 먼저 큰 값을 먼저 찾고 후에 사용하지 않은 작은 값들을 하나씩 pop하여 출력한다.

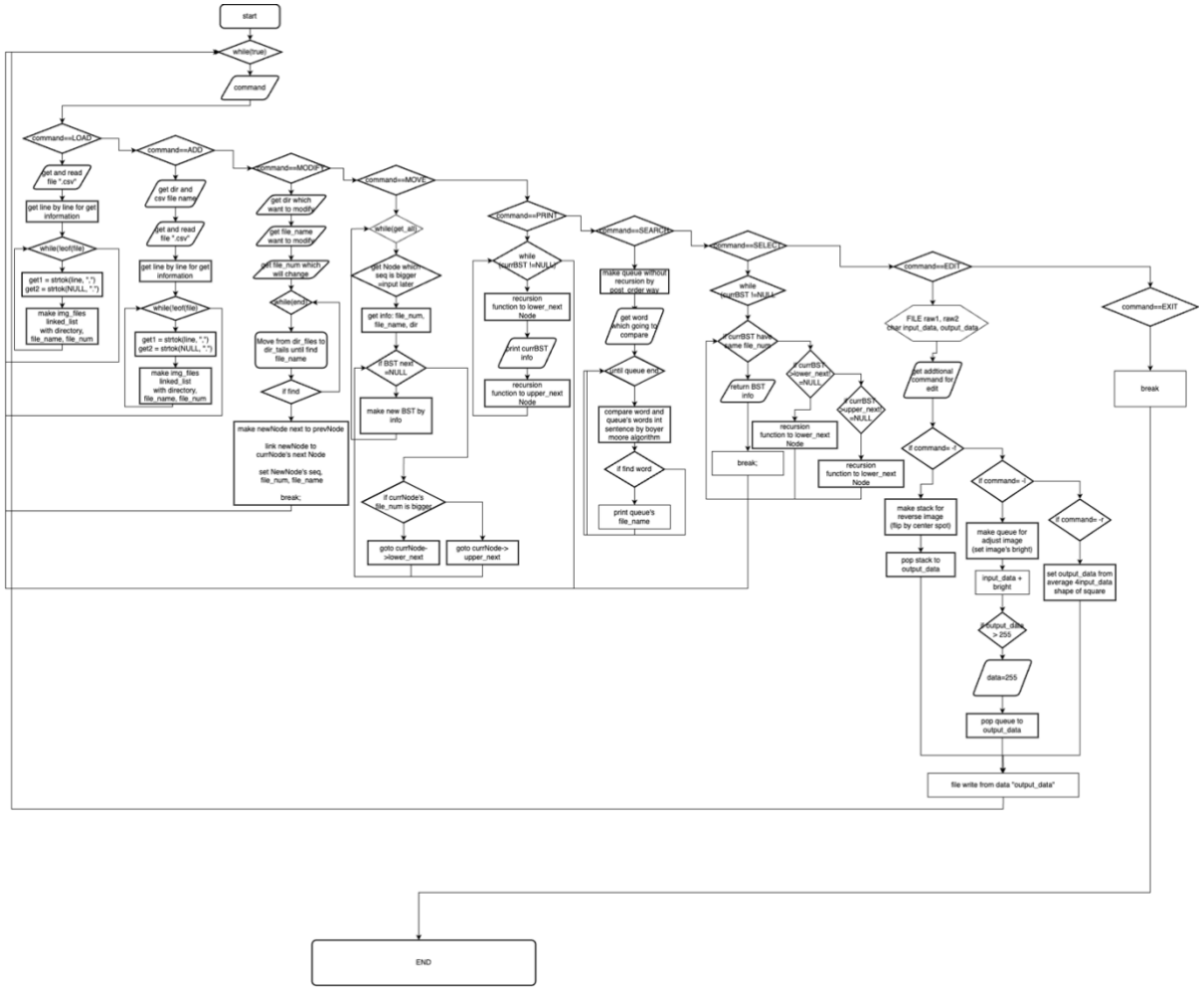
- EDIT 명령어를 통해 raw file에 저장된 각 픽셀 값을 stack 형식으로 받고 마지막부터 pop하여 순서에 변화를 주어 저장한다.

D. 주요 Class 정의 및 정리

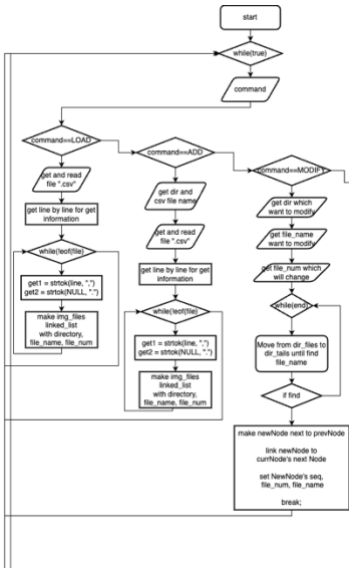
Class	멤버 함수	설명
Loaded_LIST	Loaded_List_Node *img_files Loaded_List_Node *img_tail	Img_files에 대한 디렉토리와 tail 설정
	Loaded_List_Node *new_files Loaded_List_Node *new_tail	New_files에 대한 디렉토리와 tail 설정
	void InsertNode	LOAD 명령어 또는 ADD 명령어를 통해 링크드 리스트 노드를 만들고 데이터를 저장해 둔다
	void delete_img_files void delete_new_files	LOAD 명령어 또는 ADD 명령어를 통해 이미 만들어진 경우 존재하는 링크드리스트를 삭제한다.
	void find_name	MODIFY 명령어를 통해 파일 이름을 찾고 기존 노드는 삭제하고 새로운 노드를 넣는다.
	void ModifyNode	MODIFY 명령어 사용 시 directory에 따라 find_name이 수행하도록 한다.
	int return_dir char* return_name int return_num void delete_tail	MOVE 명령어 사용시 링크드 리스트에 저장된 파일 이름, 주소, 고유 번호를 BST로 옮기기 위해 값을 반환하고 반환한 노드는 삭제한다.
Loaded_List_Node	int seq	순서를 저장한다
	char num[4]	고유 번호를 저장한다
	char name[100]	이름을 저장한다
	char dir[100]	주소를 저장한다.
	Loaded_List_Node *next Loaded_List_Node *down	Loaded_List_Node의 next와 down을 정의한다.
Database_BST	void Move_Create_BST	MOVE 명령어를 사용시 반환해서 가져온 값을 BST 형식으로 저장한다.
	int BST_exist	명령어 사용시 BST 존재 유무를 파악한다
	void print_all void print_BST_in	PRINT 명령 사용시 in-order 방식으로 출력한다
	void make_queue void post_queue void search_queue	SEARCH 명령어 사용시 post-order방식으로 BST로부터 queue를 만들고 만든 queue로부터 검색을 한다
	void select_BST void select_BST_dir Database_BST_Node*select_BST_pre (Database_BST_Node *Node, int num)	SELECT 명령 사용시 찾으려는 파일의 이름과 디렉토리를 반환하여 후에 EDIT 명령 사용시 해당 파일 이름과 디렉토리를 찾을 수 있도록 한다.

Database_BST_Node	char dir[100]	주소를 저장한다
	char name[100]	파일 이름을 저장한다
	int num	고유 번호를 저장한다.
	Database_BST_Node *upper_next	노드 고유 번호 보다 큰 쪽
	Database_BST_Node *lower_next	노드 고유 번호보다 작은 쪽
Manager	LOAD	img_files/filesnumbers.csv로 부터 고유번호와 파일 이름을 받고 링크드 리스트 제작 함수를 사용한다.
	ADD	입력받은 주소로부터 고유번호와 파일 이름을 받고 링크드 리스트 제작 함수를 사용한다.
	MODIFY	입력 받은 파일 이름이 같은 노드의 고유 번호를 수정한다.
	MOVE	linked list의 정보를 가져와 BST로 만든다
	PRINT	BST의 모든 노드를 출력한다.
	SEARCH	같은 단어가 있는 BST 노드를 출력한다
	SELECT	고유번호가 같은 BST를 선택한다.
	EDIT	선택한 raw_file을 뒤집거나, 밝기를 조절하거나, 사이즈를 조절 한다.

2. Flowchart



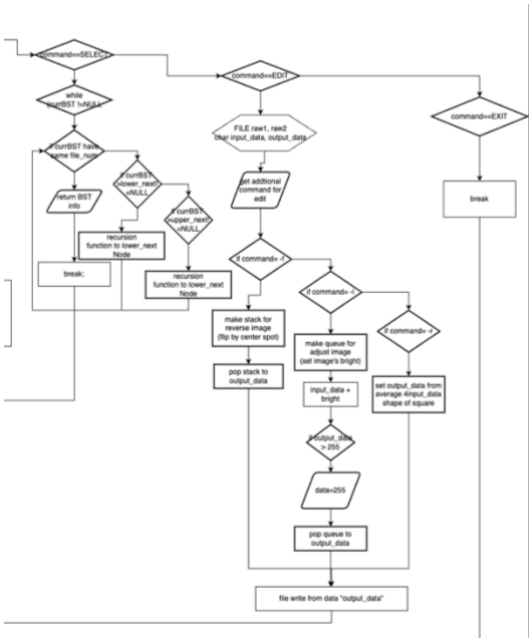
프로그램의 전체적 흐름을 나타내는 위의 그림과 같다. 전체적으로 프로그램을 설명하면 EXIT 명령이 나오기 전까지 반복 되는 것으로 각 명령은 Manage class 에서 관리하고 있으며 Manage class에서는 LOAD, ADD, MODIFY, MOVE, PRINT, SEARCH, SELECT, EDIT을 다룬다. 위의 flow에서 입력이 부족하거나 찾질 못하면 각 명령에 해당하는 에러코드를 출력한다.



LOAD와 ADD 명령어에서는 같은 Loaded_List의 insert Node를 공유하고 있기에 같은 구조의 flow를 가지고 있다. MODIFY 명령어는 먼저 수정될 위치의 dir, 수정될 파일 이름, 수정할 번호를 입력 받고 해당 경로에 이름이 같은 Linked list를 찾기 위해 첫번째 node부터 제일 끝 노드까지 이동하여 비교하며 만약 찾았다면 해당 고유 번호를 수정한다.

MOVE 명령어는 마지막에 들어간 linked list의 노드부터 해당 노드의 값을 반환 받아 BST로 제작하는 것으로 BST를 만들 때 입력하려는 새로운 BST가 기존 BST보다 값이 큰 경우 upper_next로 아닌 경우 lower_next로 이동하고 해당 위치가 null인 경우 자리 잡는다. Print는 in-order 방식으로 recursion하여 출력하고 SEARCH는 recursion 없이 post-order을 하기위

해 stack을 만들고 들어간 순서 역으로 pop하여 queue 로 만든다. 만든 queue로는 하나씩 넘어가며 해당 word가 있는 경우 파일 이름을 출력한다. word를 비교하는 방식은 아래 알고리즘의 보이어 무어 방식으로 비교하여 연산을 줄인다.



SELECT 명령어인 경우 BST를 pre-order 방식으로 읽고 만약 찾으려는 번호와 해당 노드의 고유 번호가 같은 경우 값을 manage class 변수에 저장하여 후에 Edit에 사용한다. EDIT 명령어인 경우 추가적 명령어를 더 받고 -f 인 경우 stack을 이용하여 역순으로 저장하여 raw file을 수정하고 -l 인경우 밝기를 입력 받고 해당만큼 밝기를 추가한다. 만약 255보다 큰 경우 255로 고정한다. -r인 경우 정사각형 모양으로 4개의 Pixel의 평균은 1개의 pixel에 저장하여 output_data에 저장하고 raw file을 만든다.

3. Algorithm

A. LOAD, ADD data to Linked list

i. 지정된 주소의 csv파일로부터 csv파일에 저장된 고유번호와 파일 이름의 정보를 한줄 한줄 읽고 ","와 "."를 기준으로 나누어 각각 고유번호와 파일 이름만을 변수에 저장한다.

ii. LOAD인 경우 img_files 디렉토리 뒤로 linked list가 연결되도록 한다.

iii. ADD인 경우 new_files의 디렉토리 뒤로 linked list가 연결되도록 하며 img_files와 new_files 노드 또한 연결되도록 한다.

iv. LOAD나 ADD 명령어가 중복 사용되면 기존 노드를 삭제한다.

B. MODIFY data in Linked list

i. 변경할 노드의 주소, 파일 이름, 그리고 변경할 고유 번호를 입력 받는다.

ii. 해당하는 노드의 디랙토리로 이동하고 linked_list를 통해 노드 하나 하나 비교하여 파일 이름이 같은 노드가 있는 경우 멈춘다.

iii. 이전 노드 또한 저장하고 멈췄던 노드의 seq(순서)를 따로 뽑아내고 이전 노드와 멈췄던 노드의 다음을 linked list로 연결한다. 멈췄던 노드는 삭제한다.

iv. 만약 찾지 못했으면 Error 코드를 출력한다.

C. MOVE data from linked list to BST

i. 명령어 실행 시 linked list에 저장해 두었던 seq 를 비교하여 가장 큰 seq 값을 가진 노드를 먼저 BST로 옮긴다.

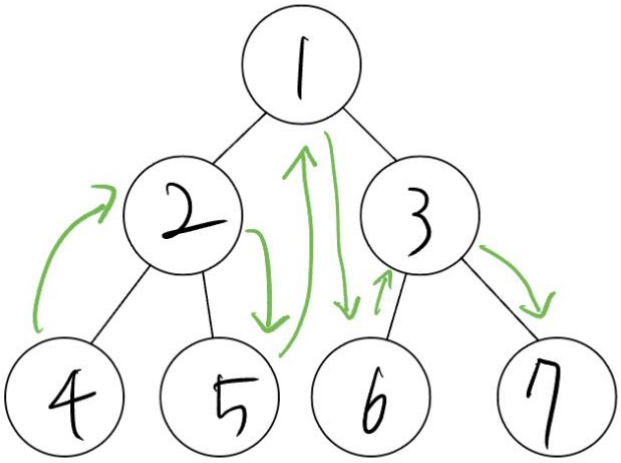
ii. 노드의 값을 먼저 제작한 반환 함수를 통해 주소, 파일 이름, 고유 번호를 임시 변수에 저장한다.

iii. 고유 번호를 기준으로 BST를 제작하며 기존 BST 노드의 고유번호 보다 큰 경우는 upper_next로 이동하고 아닌 경우 lower_next 로 이동하여 노드를 이동한다. 만약 크거나 작아서 이동했을 때 해당 노드가 NULL 인 경우 새로운 노드를 저장한다.

D. PRINT data from BST

i. in-order 방식으로 BST를 이동하여 해당 노드의 값을 출력한다.

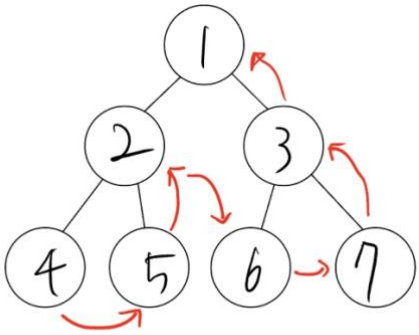
ii. in-order 방식을 아래와 같다.



E. SEARCH data from BST

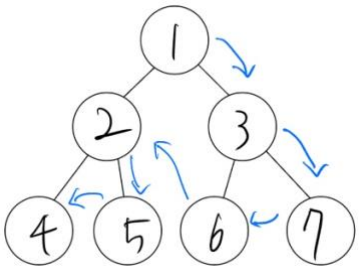
i. 먼저 모든 BST에 대하여 post-order 방식으로 queue를 제작한다. 제작 알고리즘은 아래와 같다.

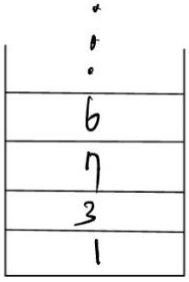
ii. 출력 순서



4->5->2->6->7->3->1

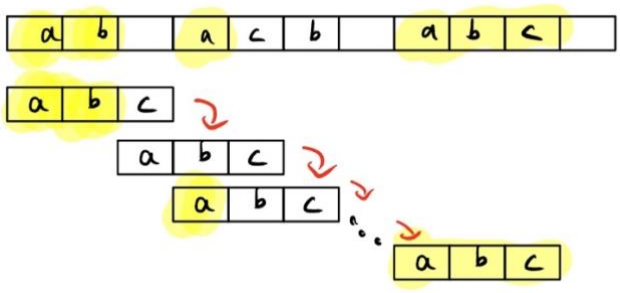
iii. stack





stack을 해당 순서로 쌓아서 역으로 출력하면 post-order 방식으로 출력할 수 있다. 출력된 결과는 queue에 저장하였다.

iv. 만약 BST를 queue로 구현했다면 queue 노드에 저장된 파일 이름과 입력 받은 단어를 비교하도록 보이어 무어 알고리즘을 사용한다.



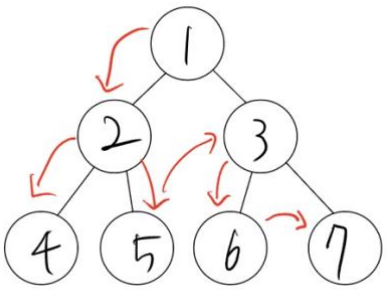
먼저 비교한 값과 비교할 값이 같은 것이 있다면 모두 같다면 출력하면 되지만 앞의 일부만 같다면 같은 일부 개수만큼 뒤로 이동하여 다시 비교한다.

F. SELECT from BST by number

i. 먼저 모든 BST에 대해서 pre-order 방식으로 경로를 찾는다.

ii. 별도의 변수에 파일 이름과 디렉토리, num을 저장한다.

iii. pre-order 방식은 아래와 같다.



G. EDIT raw files

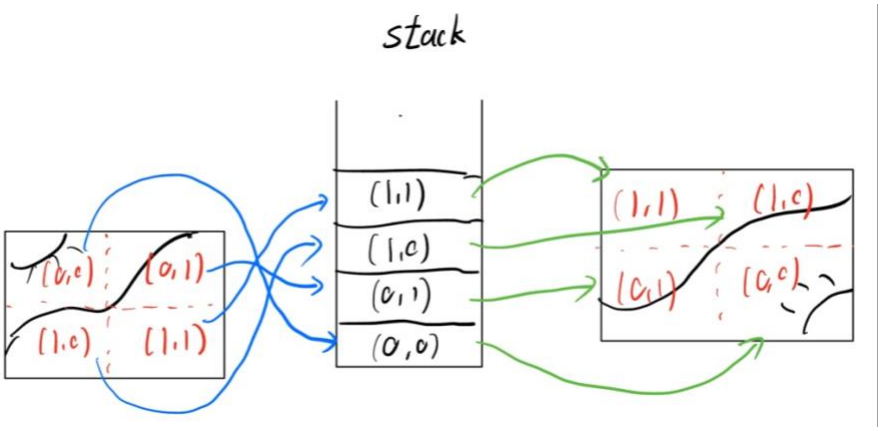
i. SELECT 명령어를 통해 저장되었던 선택한 노드의 디렉토리, 파일 이름, 고유 번호 값을 불러와서 해당 경로의 images 폴더의 해당 파일 이름을 찾아서 파일을 연다.

ii. 열은 파일의 픽셀 단위를 input_data에 하나 하나씩 저장한다.

iii. 만약 flipped 방식을 사용할 경우

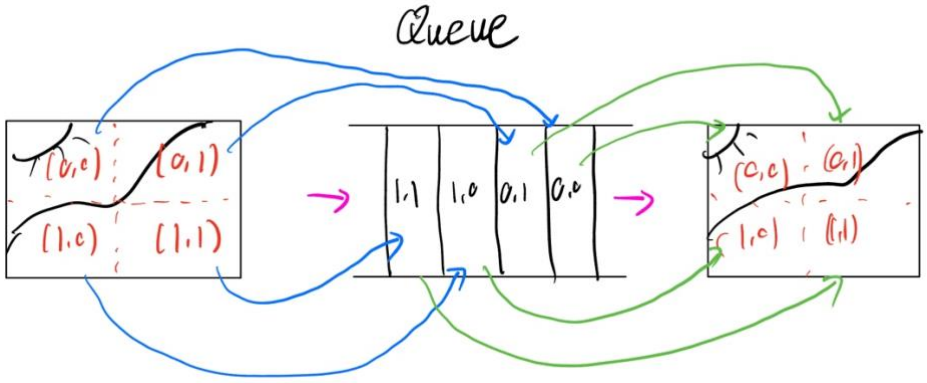
- input_data에 저장된 값을 stack 형식으로 쌓아서 저장하고 저장이 끝났다면 output_data에 역순으로 하나 하나씩 저장한다.

- 아래 그림은 이해를 위한 예시 그림이다.



iv. 만약 adjusted 방식을 사용할 경우

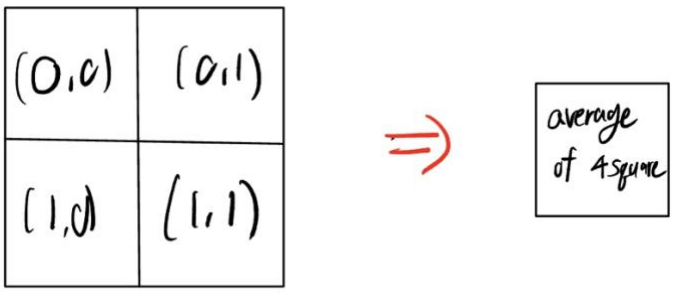
- input_data에 저장된 값을 Queue 형식으로 순차적으로 저장하고 저장한 후에는 저장된 input_data 값에 입력 받은 밝기 변화 값을 더하여 새로운 output_data에 저장한다.



v. 만약 resized 방식을 사용할 경우

- 크기가 4분의1로 줄어든 다는 것은 너비와 높이는 각각 절반 씩 줄어든다는 것이다.

- 직사각형이 아닌 정사각형의 4개의 픽셀 값을 평균 잡아서 output_data 배열에 하나 하나씩 저장하여 128*128 크기의 이미지를 생성한다.



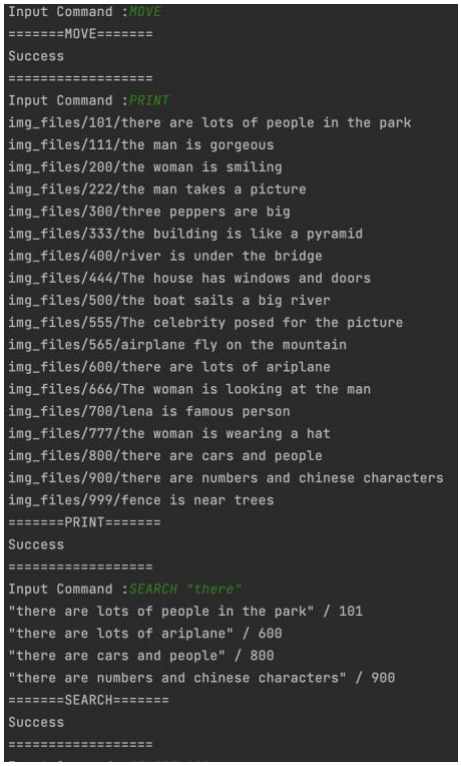
위 그림은 설명을 위한 그림이며 이 과정을 input_data의 모든 배열을 4개씩 평균 내서 output_data 에 저장한다.

4. Result Screen

A. 동작 확인

```
Input Command :LOAD
=======LOAD=======
fence is near trees/999
there are lots of people in the park/101
the man is gorgeous/111
the woman is smiling/200
the man takes a picture/222
three peppers are big/300
the building is like a pyramid/333
river is under the bridge/400
The house has windows and doors/444
the boat sails a big river /500
The celebrity posed for the picture/555
there are lots of ariplane/600
The woman is looking at the man /666
lena is famous person/700
the woman is wearing a hat/777
there are cars and people/800
airplane fly on the mountain/888
there are numbers and chinese characters/900
==================
Input Command :ADD new_files new_filesnumbers.csv
=======ADD=======
Success
==================
Input Command :MODIFY img_files "airplane fly on the mountain" 889
=======MODIFY=======
Success
==================
Input Command :MODIFY img_files "airplane fly on the mountain" 565
=======MODIFY=======
Success
==================
Input Command :
```

LOAD 명령어와 ADD 명령어가 정상적으로 들어갔고 MODIFY 또한 정상적으로 작동했음을 볼 수 있다.



MOVE 명령어를 통해 BST가 만들어졌고, PRINT 명령어를 통해 in-order 방식으로 출력하여 고유번호 오름차 순으로 출력됨을 확인했다.

또한 SEARCH 명령어에서 “there”을 검색하여 해당 단어가 존재하는 모든 BST의 file_name 과 고유 번호를 출력하였다.

```
Input Command :SELECT 600
=======SELECT=======
Success
==================
Input Command :EDIT -f
=======EDIT=======
Success
==================
Input Command :EDIT -l 25
=======EDIT=======
Success
==================
Input Command :EDIT -r
=======EDIT=======
Success
==================
Input Command :EXIT
=======EXIT=======
Success
==================

종료 코드 0(으)로 완료된 프로세스
```

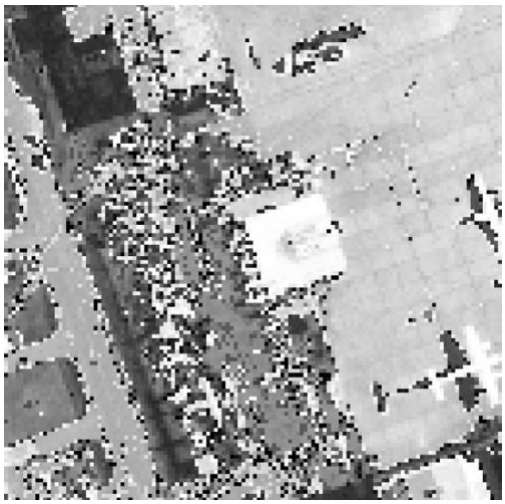
SELECT 명령어를 통해 고유번호 600번을 지정하였고, EDIT 명령어를 통해 raw file을 flip, adjust, resize 하였다. 이 때 EDIT -f 는 비교적 시간이 오래 걸렸지만 정상적으로 작동함을 확인하였다.

EDIT의 결과는 아래와 같다.

<원본>

<flipped image>

<adjusted image>



<resized image>

위의 원본과 아래 편집된 raw file을 하나씩 비교해 보면 flipped image는 원본의 중심을 점대칭한 이미지가 생성되었으며, adjusted image에서는 밝기는 30 정도 밝아진 사진이 나옴을 확인 할 수 있다. resized image에서는 4칸의 정사각형을 1칸의 픽셀로 평균값을 내어 저장 했음을 볼 수 있다.

5. Consideration

이번 프로젝트를 진행하는데 있어 아직 Skeleton code를 참고하여 해당 코드 flow를 익히고 후에 g++ 환경에서 적응하는데 실패하였다. 기존 1학기 객체지향프로그래밍실습에서는 main.cpp 파일 내부에 class 파일을 모두 두어 한번에 처리하였기에 이에 익숙해져 먼저 main.cpp 파일에 모든 class를 설정하고 후에 각 class를 분리하여 헤더와 class 파일로 만들어 두었다. 이 과정에서 문제가 많이 발생하였고 주석 또한 일부분 날라가서 다시 쓰는 사고가 일어났다.

그 외에 img_files에서 csv 파일을 불러온 후 LOAD하는 것이나 ADD를 하는 것은 비교적 간단하였고 다만 BST를 만들고 난 후 SEARCH에서는 post-order 방식으로 탐색을 하되 다만 recursion을 사용하면 안되었기에 먼저 post-order 방식의 패턴을 찾아내었고 해당 패턴을 stack에 저장하여 후에 다시 출력하도록 하였다. 보이어 무어 알고리즘에서는 post-order을 끝내고 나니 비교적 간단하였다. 문장에서 스펠링 하나하나 비교하는 것과 다르게 같은 스펠 하나를 찾고 그 뒤에 맞은 개수 만큼 카운트하여 뒤로 이동하거나 맞는 경우 출력하는 것으로 그림으로 설명이 잘 되었기에 금방 만들었다.

MODIFY 명령어는 후에 MOVE를 하는 과정에서 오류가 발생하곤 했는데, img_list 노드를 처음부터 출력하여 확인했을 때 해당 노드 파일의 고유 번호, 파일 이름 모두 정상적으로 변경됨을 확인했으나 BST로 옮기는 과정에서 예상컨데 새로 next를 설정한 노드가 제대로 설정되지 않아 중간에 끊겨 오류가 발생한 것으로 추측하고 있다. Clion 환경에서 이를 디버그 하여 확인은 했으나 이미 만든 코드에서 대체 무엇이 잘못 되었는지 찾을 충분한 시간이 없어 수정하지 못했으며 다만 MODIFY 없이 진행하는 경우 EDIT 명령까지 정상 작동함을 확인했다.

EDIT 명령에서는 flip, adjust, resize를 진행하는데 이는 input_file에서 input_data를 배열로 가져와 stack 또는 queue로 두고 변경하는 점이 이해하기 좋았다. 다만 Flip을 하는 경우 진행하는 동안 시간이 꽤 걸렸는데 이는 확인해봐야할 사항 중 하나이다.

Visual studio 또는 Clion에서의 개발과 디버그에 너무 익숙해져 있기에 linux 환경에서 g++을 사용하는데 적응을 하지 못하였고 linux 환경으로서의 완성은 하지 못하였으나 해당 시간을 많이 잡아먹으면서 메모리 누수는 찾아서 조치할 생각조차 하지 못했다. 또한 log.txt를 만들어야 함을 Issue를 읽지 않아서 모르고 다 콘솔 출력으로 설정하였다.

이번 1차 프로젝트에서는 과제에서 원하는 형식을 제대로 이해 못하고 완성하지 못한 코드를 만들었지만 앞으로 리눅스 환경에서 개발하는 것을 적응하고 디버거, 그리고 메모리 관리를 할 수 있도록 고쳐 나갈 것이다.