

데이터구조설계

1차 프로젝트 보고서

제출 기한: 2022년 10월 13일 (목)

과목: 데이터구조설계

학부: 컴퓨터정보공학부

학번: 2021202039

이름: 이소영

실습 분반: 금

1. Introduction

본 데이터구조설계 1 차 프로젝트에서는 이진 탐색 트리(Binary Search Tree)와 연결 리스트(Linked List), 그리고 큐(Queue) 자료구조를 이용하여 간단한 사진 파일 편집 프로그램을 구현한다. 이 프로그램은 특정 경로에 저장된 사진 파일에 대한 정보를 링크드 리스트 구조로 저장한 후, 검색에 용이하게 트리 자료 구조로 저장한다. 각 명령어에 따른 구조 이용과 올바른 출력은 각 명령어별로 설명하겠다. 사용 가능한 명령어는 LOAD, ADD, MODIFY, MOVE, PRINT, SEARCH, SELECT, EDIT, EXIT 로 총 9 개이다.

LOAD 는 csv 파일의 데이터 정보를 불러오는 명령어로, csv 파일에 데이터 정보가 존재할 경우 csv 파일을 읽어 링크드 리스트 자료 구조에 파일의 이름과 고유 번호를 저장 후 출력한다. 만약 입력 되는 링크드 리스트 노드 개수가 100 개 넘는다면, 가장 먼저 들어온 노드를 삭제 후 새로운 데이터를 노드로 추가한다. CSV 파일이 존재하지 않으면 에러 코드를 출력한다. csv 파일의 데이터가 정상적으로 링크드 리스트에 저장되면 저장된 노드들을 출력한다. 이때의 링크드 리스트를 Loaded_List 라고 한다.

이때 csv 파일의 이름은 filesnumbers.csv 로, 프로젝트 폴더 내 img_files 에 있는 파일을 사용하였다.

ADD 는 새로운 디렉토리를 탐색하여 링크드 리스트에 데이터를 추가하는 명령어다. 새로운 디렉토리에 추가될 데이터들은 앞서 LOAD 에서 추가된 노드들과 2 차원 링크드 리스트로 연결된다. LOAD 와 마찬가지로 노드 개수가 100 개가 넘어가면, 가장 먼저 들어온 노드를 삭제 후 새로운 데이터를 노드로 추가한다. 예를 들어 "ADD new_files new_filesnumbers.csv"를 입력받으면 new_filesnumbers.csv 파일에 있는 데이터들을 링크드 리스트의 new_files 디렉토리에 삽입한다. 이 csv 파일의 위치는 프로젝트 폴더의 루트 디렉토리에 위치하도록 한다. 인자가 부족하거나, 해당 csv 파일이 존재하지 않거나, Loaded_List 가 존재하지 않는 경우 에러 코드를 출력한다.

MODIFY 는 Loaded_List 에 존재하는 노드의 파일 고유 번호를 수정하는 명령어로, "MODIFY img_files "A CUP ON THE TABLE" 303"와 같이 사용할 수 있다. img_files 라는 디렉토리에 있는 "A CUP ON THE TABLE"이라는 파일의 고유 번호를 303 으로 변경하라는 뜻이다. 이때 노드에 접근하여 번호만 수정하는 것이 아니라, 해당 파일을 의미하는 노드를 삭제 후 새로운 노드를 추가하는 방식으로 고유 번호를 수정한다. 인자가 부족하거나 이미 존재하는 중복된 고유 번호일 경우 에러 코드를 출력한다.

MOVE 는 Loaded_List 의 정보들을 이진 탐색 트리로 옮기는 명령어이다. 본 프로젝트에의 이진 탐색 트리는 Database_BST 라고 하겠다. 파일 고유 번호와 파일 이름을 이용하여 이진 탐색 트리(BST)의 노드를 생성한다. 이때 파일 고유번호가 작은 노드를 부모 노드의 왼쪽에, 큰 노드를

부모 노드의 오른쪽에 저장한다. 노드를 제거할 때는 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동한다. Database_BST 내 노드의 최대 개수는 300 개로, 300 개가 넘게 입력될 경우 트리 내 파일의 고유 번호가 낮은 순서부터 차례로 제거한다. Loaded_List 가 비어 있다면 에러 코드를 출력한다.

PRINT 는 Database_BST 에 저장되어 있는 정보를 트리 중위 순회(in order) 방식으로 탐색하여 출력한다. Database_BST 가 존재하지 않을 경우 에러를 출력한다.

SEARCH 는 특정 단어가 포함된 파일의 이름을 탐색하여 출력하기 위한 명령어로, 인자로 하나 이상의 단어가 포함된다. 인자가 없거나 Database_BST 가 비어있는 경우 에러를 출력한다. 보이어나 알고리즘을 이용하여 다음 규칙에 따라 파일을 탐색하여 출력한다. 첫째로 파일의 이름기반 검색을 위해 트리를 Iterative post-order 방식으로 순회하며 큐(Queue)에 트리 노드 안에 존재하는 파일이름과 고유번호에 대한 정보를 담는다. 둘째로, 모든 노드에 대한 정보가 저장되었다면 순차적으로 POP 을 진행하며 사용자가 SEARCH 명령어에 검색한 단어가 파일명에 있는지 검색한다. 셋째로, 단어가 존재하는 경우 해당 노드의 파일명 전체와 파일 고유번호를 출력한다. 마지막으로, 검색이 완료된 이후의 큐는 반드시 비어 있어야 한다. 예를 들어 "SEARCH "CUP""을 입력받으면 CUP 이라는 단어가 있는 파일을 찾아낸다.

SELECT 명령어는 EDIT 명령을 수행하기 전 편집할 이미지를 선택하는 명령어로, Database_BST 에서 파일 고유 번호를 기반으로 전위 순회(pre-order) 방식을 통해 이미지를 불러온다. 인자가 없거나 존재하지 않는 파일 고유 번호일 경우, 에러 코드를 출력한다.

EDIT 은 SELECT 명령어를 통해 가져온 이미지를 편집하는 명령어로 인자로 편집 옵션과 밝기 변경 정도를 받는다. 편집 옵션은 3 가지로 -f, -r, -l 이 있다. -f 는 점대칭 옵션, -r 은 크기 조정 동작을 수행하고, -l 은 추가 인자로 받은 숫자만큼 밝기 조정 동작을 수행한다. 편집한 이미지는 "Result" 디렉토리에 저장된다.

점대칭(-f)은 각 이미지 픽셀을 Stack 구조에 순서대로 입력하고(Push) 후입선출 (POP)하여 이미지를 점대칭으로 뒤집는다.

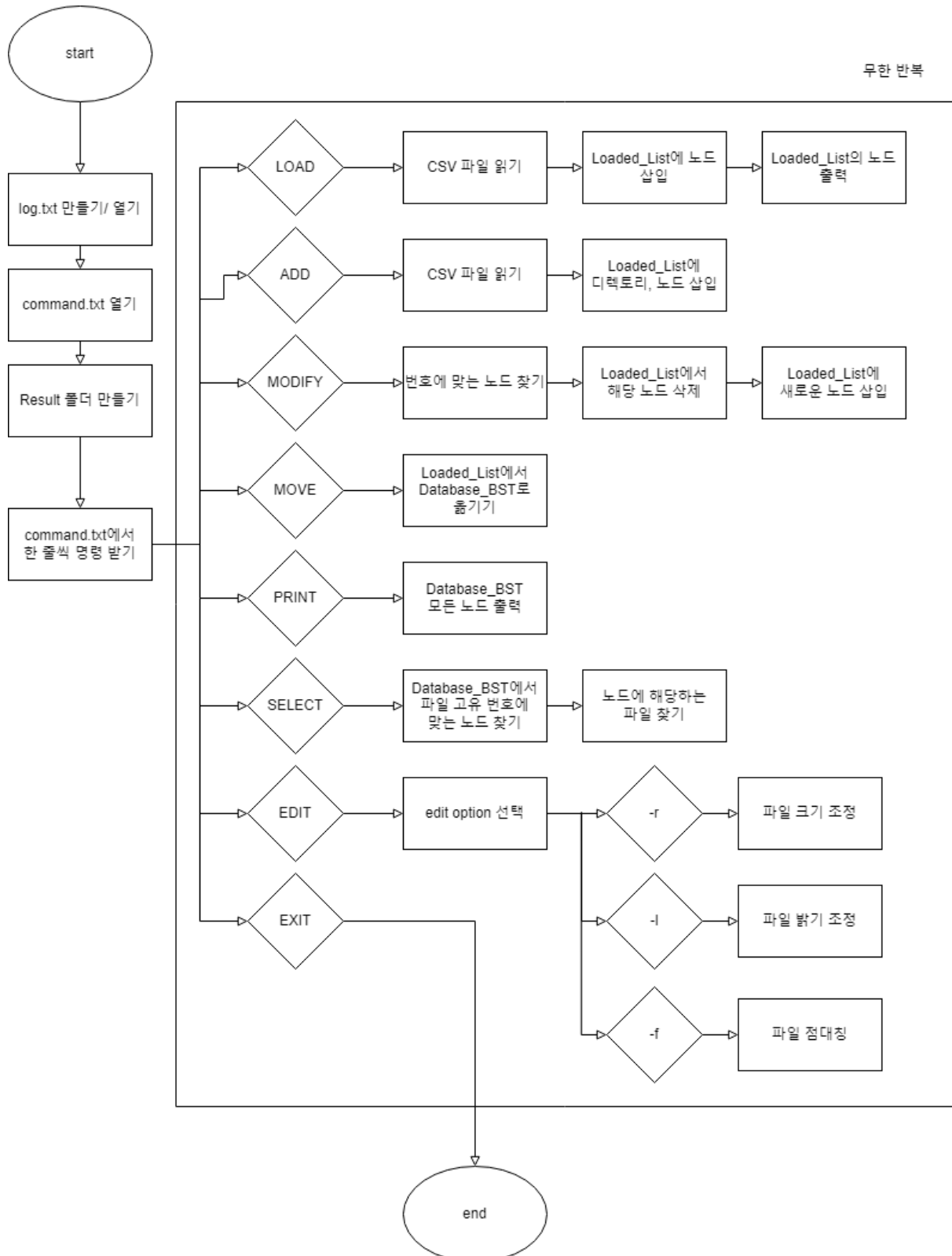
이미지 밝기 조정(-l)은 이미지 픽셀을 알맞게 Queue 에 순서대로 입력하고(Push) 선입 선출(POP) 하며 각 픽셀의 값에 특정 숫자를 더해 이미지를 밝게 만든다.

이미지 크기 조정(-r)은 입력된 이미지의 크기를 4 분의 1 로 축소하는 동작을 진행한다. 인 접한 4 개 셀의 평균값을 구해 하나의 셀에 입력하는 방식으로 구현한다.

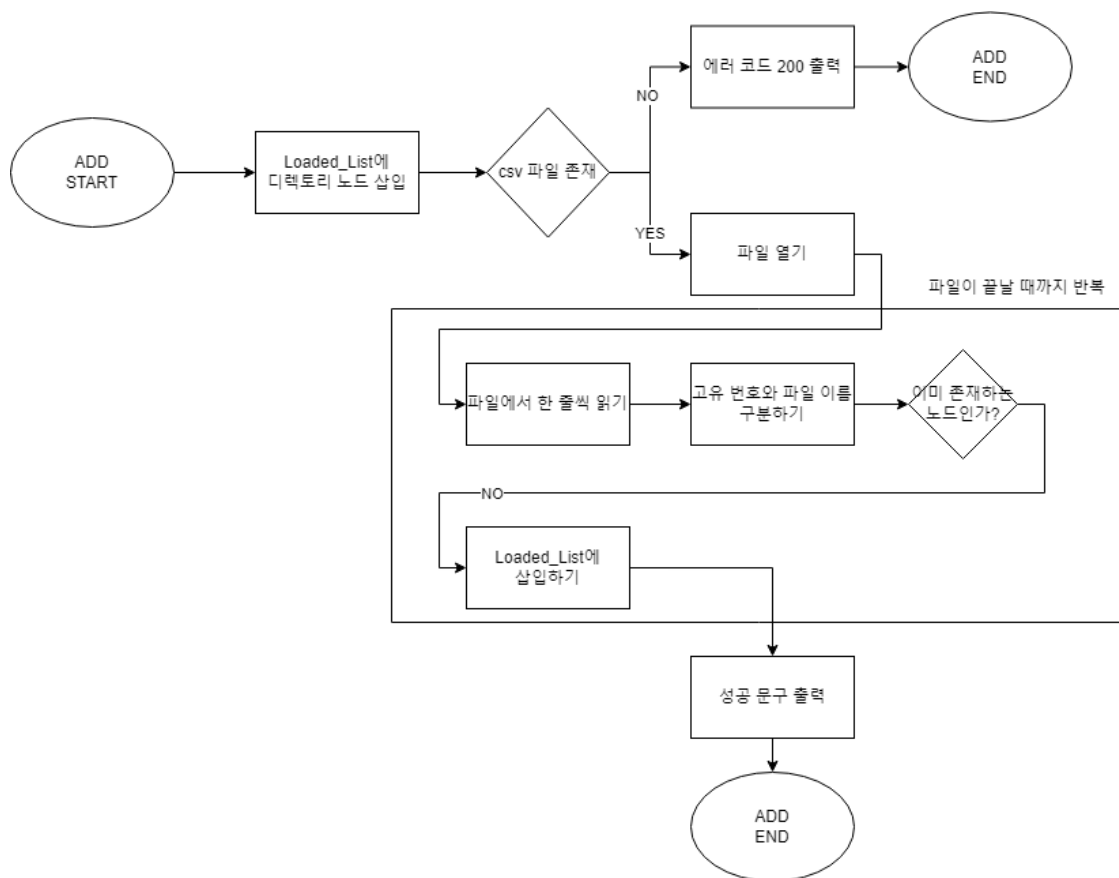
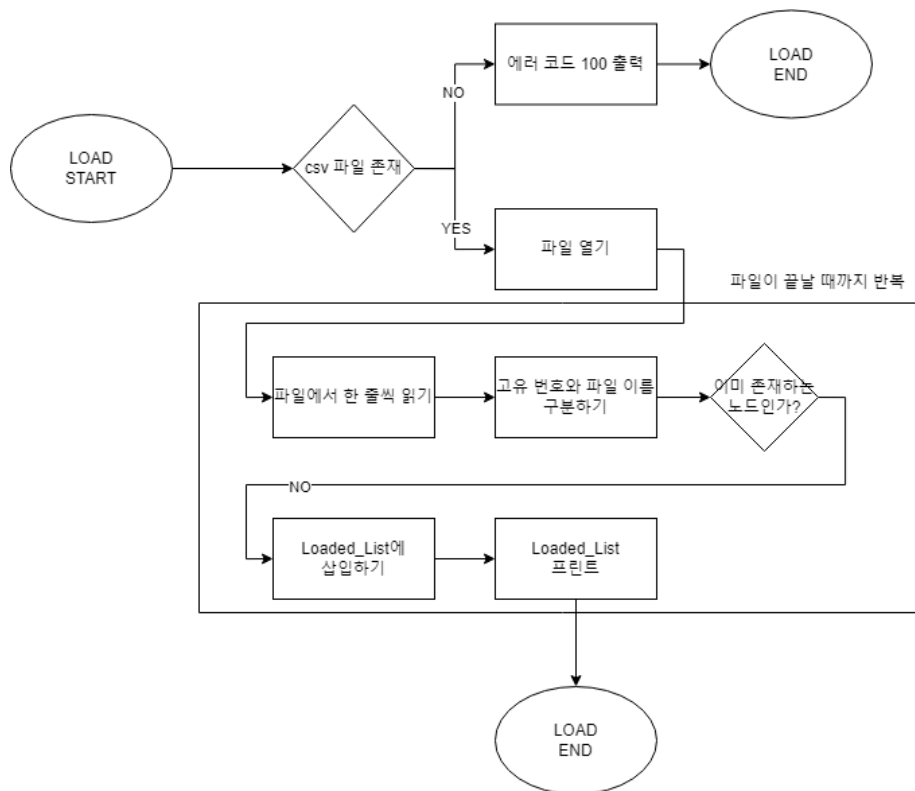
EXIT 명령을 받으면 프로그램 상의 모든 메모리를 해제하며, 프로그램을 종료한다.

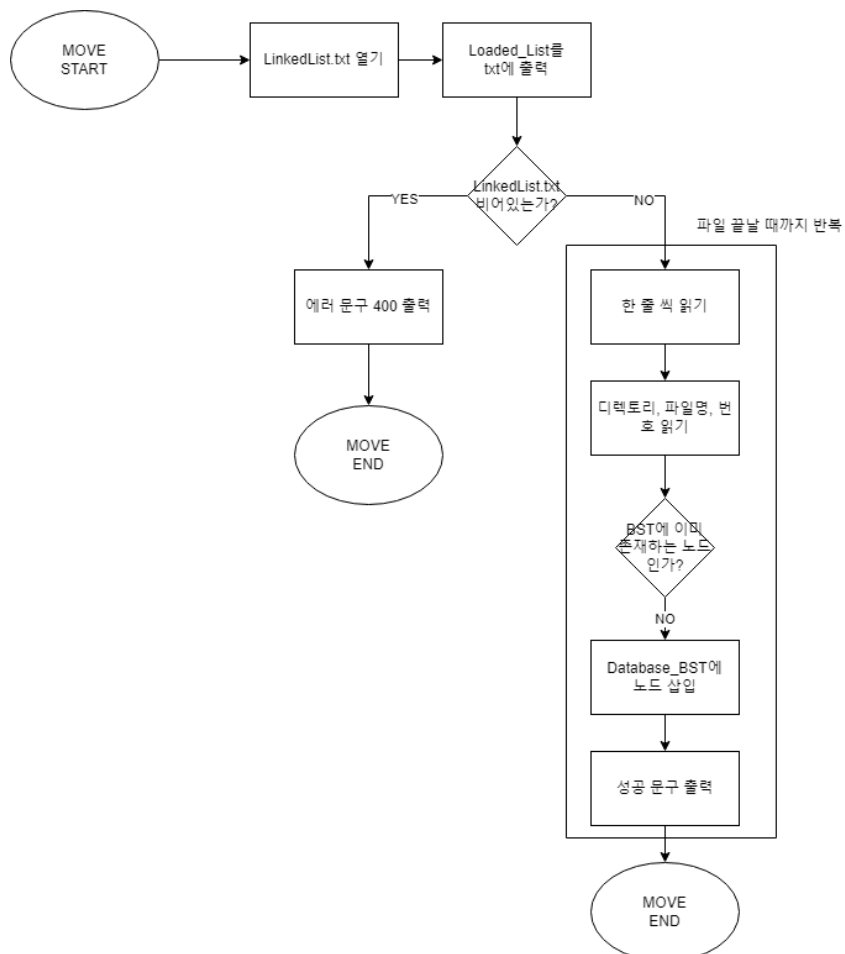
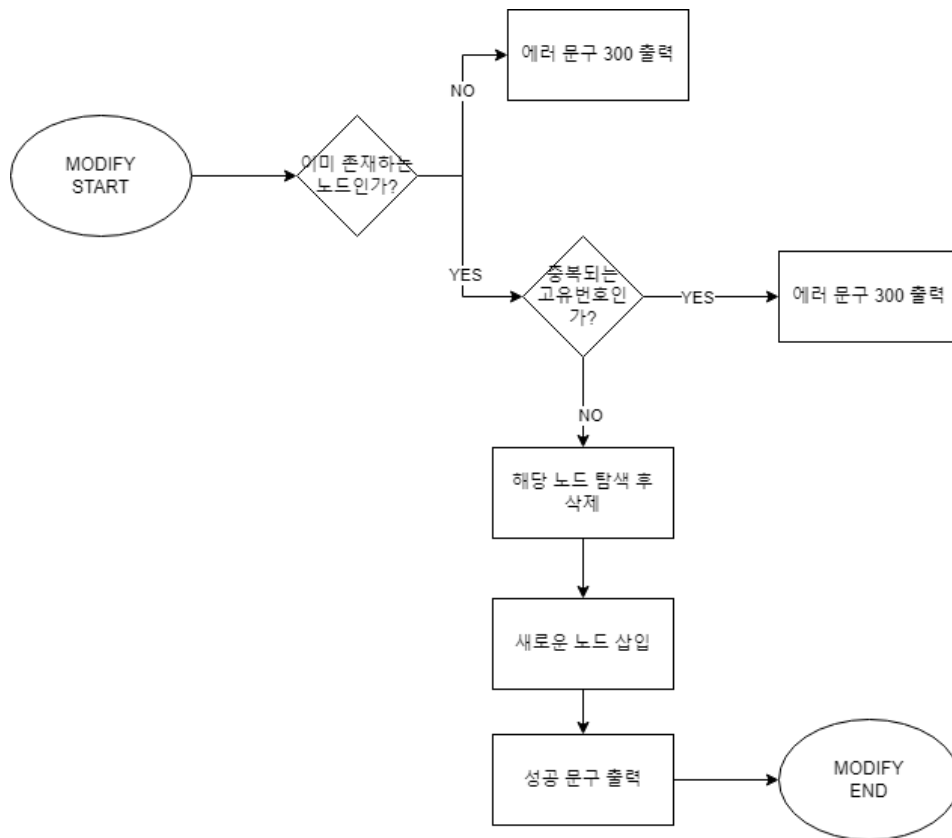
2. Flowchart

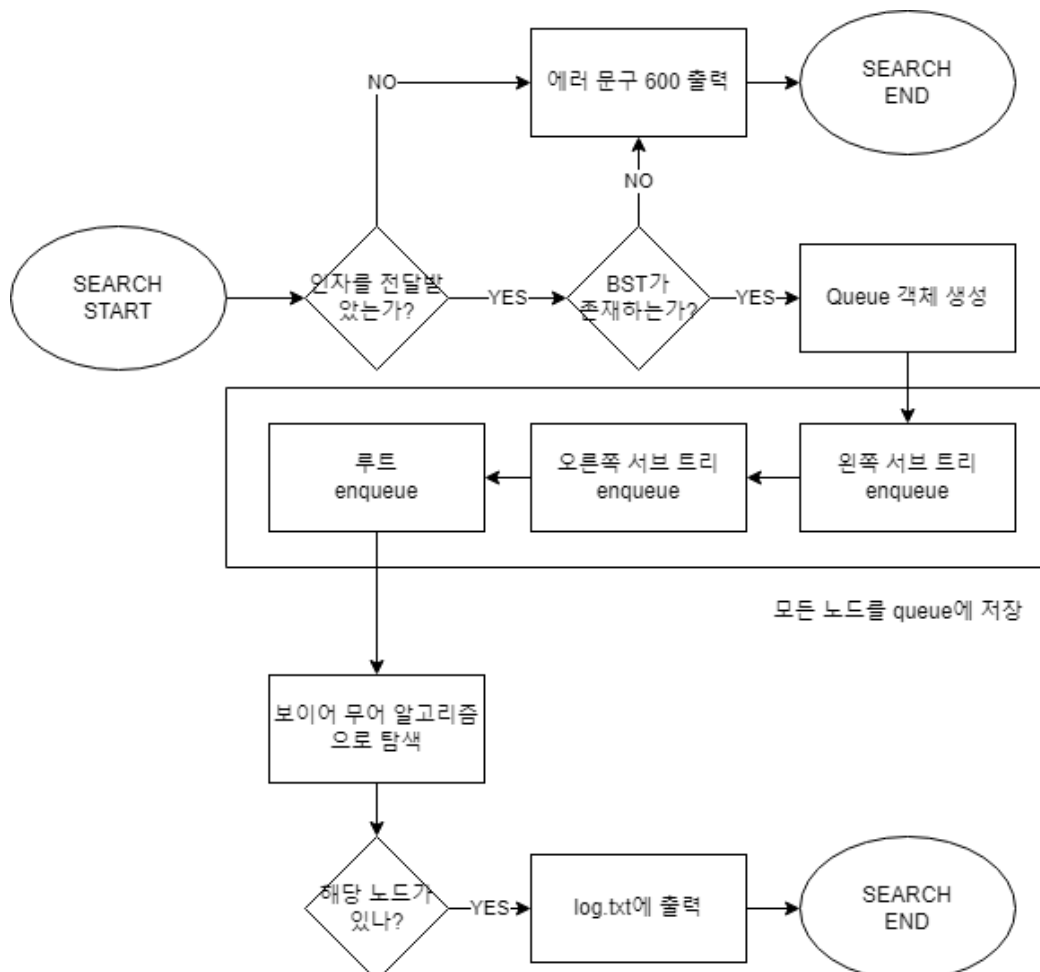
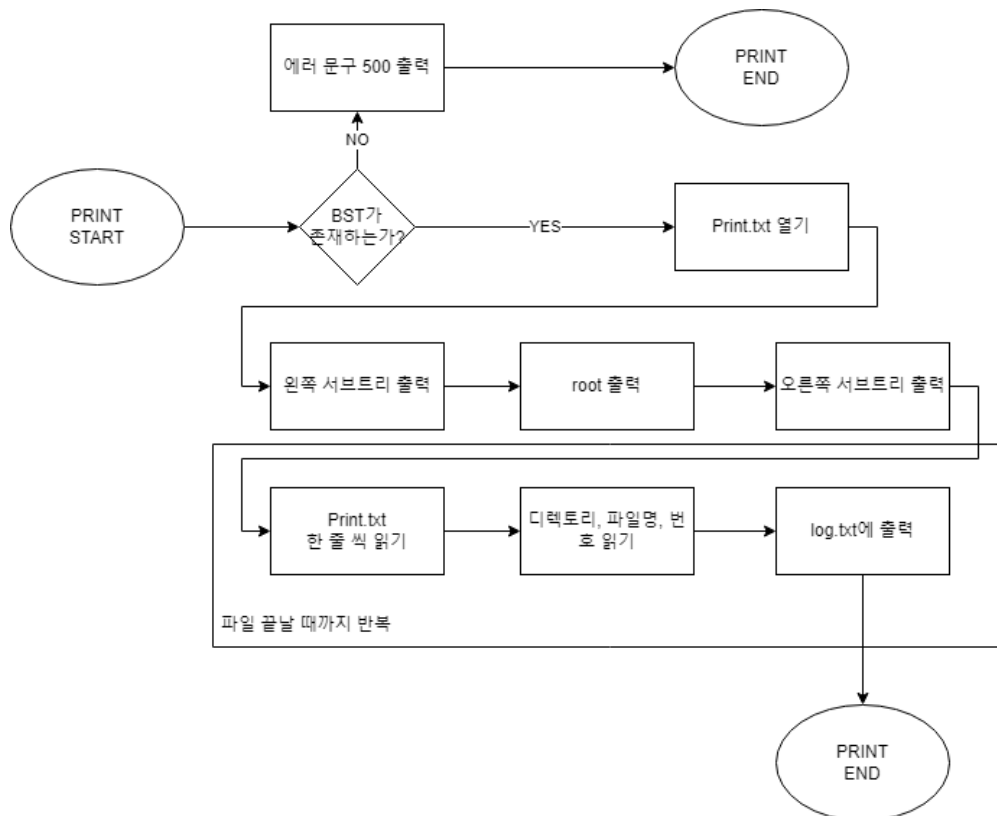
main 코드의 대략적인 흐름은 아래와 같다. command.txt 에서 받은 명령에 따라 어떤 동작을 수행할지 결정된다.

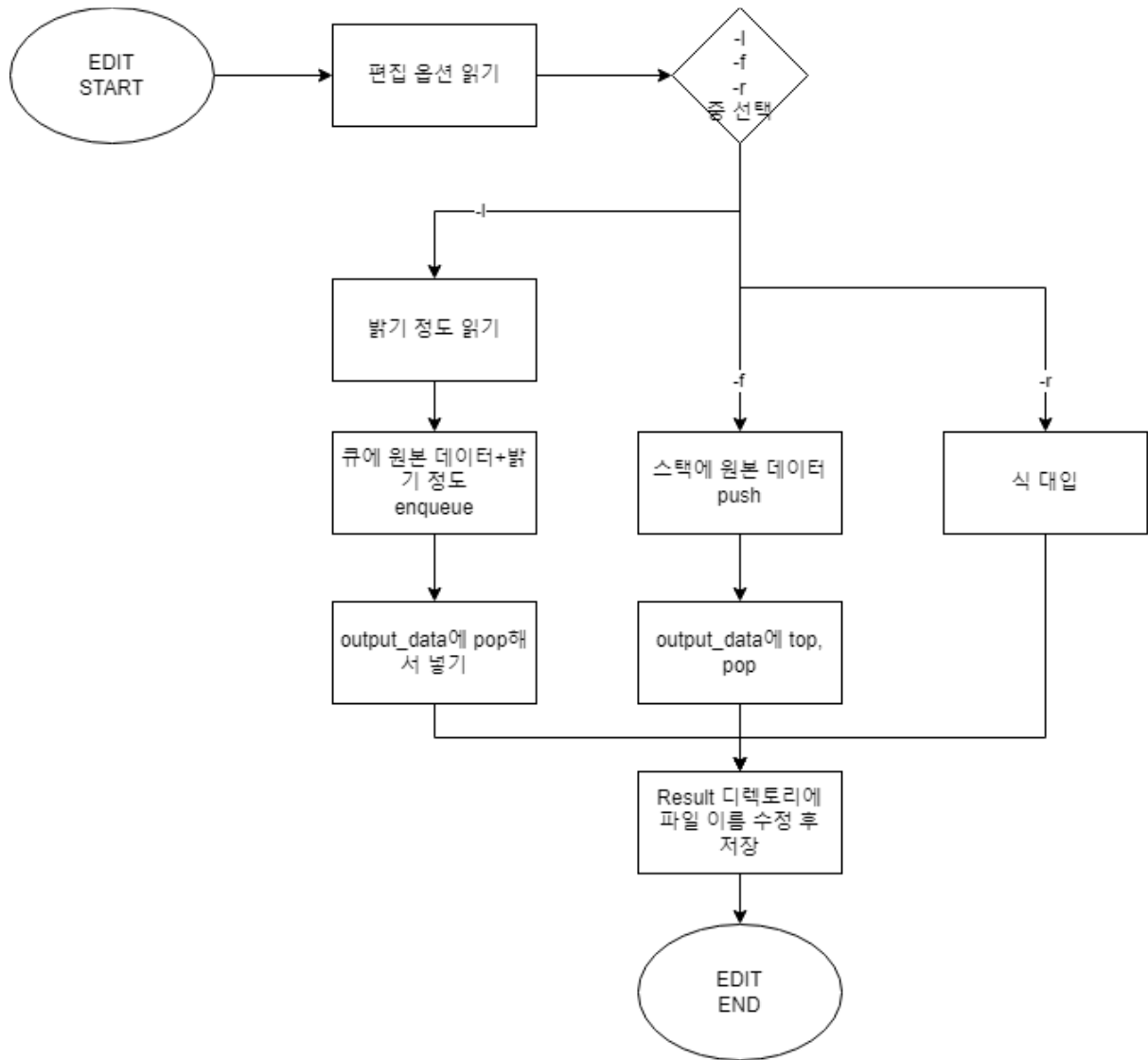
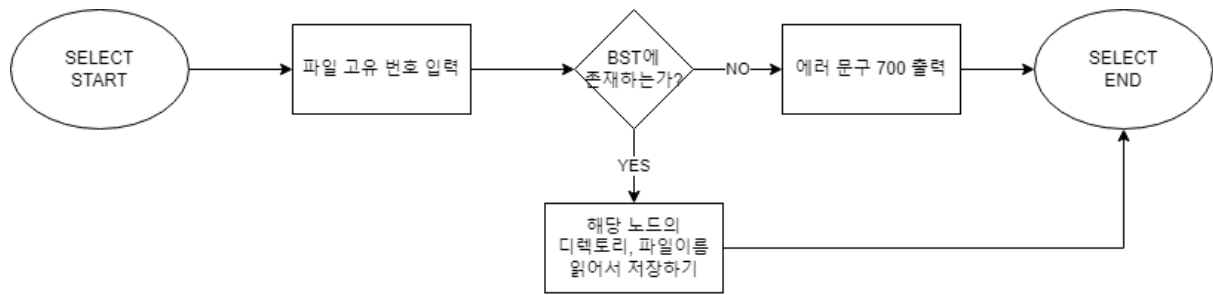


각 명령어의 과정은 아래와 같다.









3. Algorithm

프로젝트에서 사용한 알고리즘의 동작을 설명

main 코드에서 진행되는 순서대로 프로젝트에서 사용한 알고리즘의 동작을 설명하고자 한다.

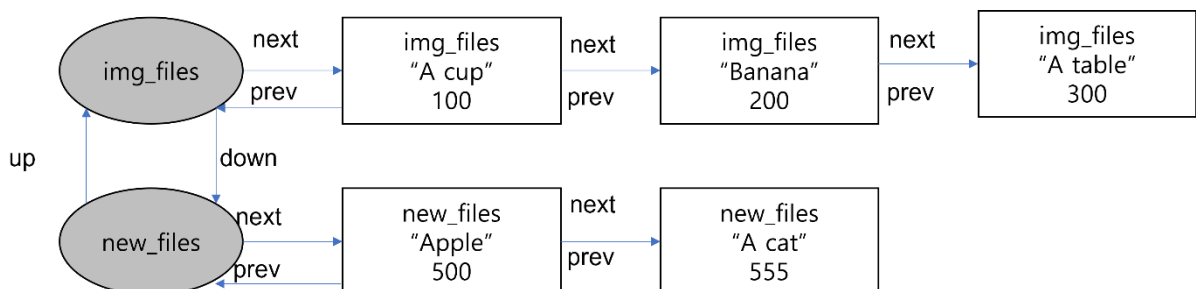
Loaded_List 클래스 객체, List 와 Database_BST 클래스의 객체, BST 를 선언한다.

LOAD 로 불러올 파일들의 디렉토리는 "img_files"로 고정되어 있으므로 "img_files"를 InsertDirecNode() 함수를 이용하여 Loaded_List 에 디렉토리 노드를 만든다. 이 함수를 이해하기 전, Loaded_List 와 Loaded_List_Node 클래스에서는 다음과 같은 변수를 사용한다. 왼쪽 사진은 Loaded_List_Node 의 public 변수로 Loaded_List 에서 접근한 멤버변수이고, 오른쪽 사진은 Loaded_List 의 private 변수들이다.

filename 은 RAW 파일의 이름, filedirec 은 해당 RAW 파일이 있는 디렉토리 이름을, filenum 은 파일의 고유 번호를 의미하고 파일 노드에서 쓰인다. nodenum 은 노드의 개수를 의미하는 변수로 일반적인 RAW 파일을 가리키는 노드가 아닌, 디렉토리 노드에서 쓰인다. head 는 Loaded_List 의 첫 노드로, "img_files" 디렉토리 노드를 가리킨다. tail 은 마지막 노드를 의미하고 이외에 phead, ptail, dhead 는 함수 작성 시 편의를 위해 임의로 선언한 변수들이다. 아래 그림으로 노드들 간의 방향 관계를 이해할 수 있다.

```
class Loaded_List_Node {
public:
    string filename;
    string filedirec;
    int filenum;
    int nodenum;    //for directory node
    //1d linked list
    Loaded_List_Node* next;
    Loaded_List_Node* prev;
    //2d linked list
    Loaded_List_Node* down;
    Loaded_List_Node* up;
};

private:
    Loaded_List_Node* head;
    Loaded_List_Node* tail;
    Loaded_List_Node* phead;
    Loaded_List_Node* ptail;
    Loaded_List_Node* dhead;    //directory head
    int nodenum;
};
```



Loaded_List 에 선언 및 정의된 Loaded_List_Node* InsertDirecNode(string fdirec)의 알고리즘을 살펴보자.

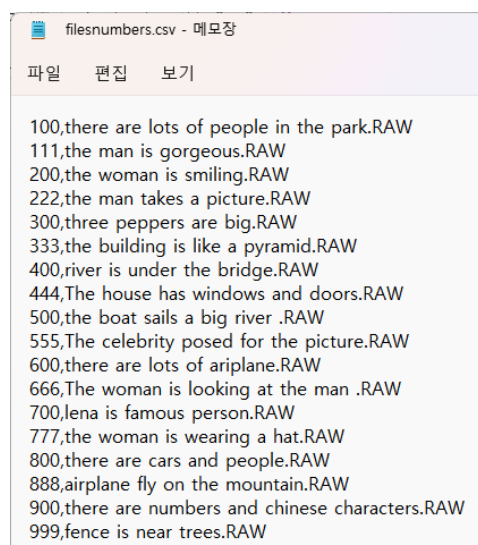
여기서 fdirec 은 새롭게 만들어질 디렉토리 노드의 디렉토리 명을 말한다. 새롭게 디렉토리 노드가 추가될 위치를 찾기 위해 while 문을 이용하며 head 에서 down 을 하며 내려간다. 이때 모든 디렉토리 노드들은 down 과 up 으로 연결됨에 유의한다. 처음으로 NULL 이 나온 곳이 새로운 디렉토리 노드가 생성될 위치다. 새로운 노드, newLoaded_List_Node 의 파일 디렉토리명(filedirec)은 인자로 전달받은 fdirec 이고, next, prev, down, up, down 방향은 모두 NULL 이다. 그러나 적합한 위치를 찾으면 이전 디렉토리 노드의 next 가 newLoaded_List_Node 가 되고, newLoaded_List_Node 의 prev 은 이전 디렉토리 노드가 된다. "img_files" 디렉토리 노드는 head 가 되므로 head 에도 이 새로운 노드를 할당한다.

command.txt 파일로 명령을 받고, log.txt 파일에 출력 결과를 기록하므로 log.txt 파일을 생성 후 command.txt 파일과 함께 연다. 또한 EDIT 으로 편집한 사진들을 Result 폴더에 저장하므로 Reulst 디렉토리를 만든다.

이제 command.txt 에서 명령을 한 줄씩 읽어오는데 더 이상 읽을 명령이 없을 때까지 혹은 EXIT 를 입력받을 때까지 while 문을 이용하여 명령 입력과 동작 수행을 반복한다.

LOAD 를 입력받는다. LOAD 는 img_files\filesnumbers.csv 파일을 불러오는 것이 조건이므로 해당 파일을 연다. 만약 해당 파일이 존재하지 않는다면, 에러 코드를 출력한다.

csv 파일을 메모장 형식으로 열면 아래와 같은 형식이다.



프로그램은 여기서 100 과 같은 숫자를 읽어 파일 고유 번호로, there are lots of people in the park.RAW 를 읽어 파일 이름으로 분류한다.

반복문을 이용하여 파일 고유 번호와 파일 이름을 쉼표나 'W0' 등을 기준으로 구분하여 csv_num_str, csv_name 변수에 저장하였다. 이때 csv_num_str 은 string 데이터 타입인데 파일에서 읽어왔기 때문에 문자열로 저장하였지만 Loaded_List 에 노드로 삽입할 때는 int 형으로 삽입해야 하므로 stoi()를 이용하여 csv_num 에 정수 형태로 저장한다. 이때 stoi() 함수는 txt 파일이 UTF-8(BOM) 형식일 때는 사용할 수 없다. ANSI 혹은 UTF-8 인코딩 파일을 읽어올 때만 사용 가능하므로 빌드 시 주의한다.

LOAD 에서 노드 삽입은 중복된 노드가 아닌지 ExistLoaded_List_Node() 함수로 확인 후 Insert1dLoaede_List_Node() 함수를 사용하여 수행한다.

int ExistLoaded_List_Node(string fname, string fdirec, int fnum)은 인자로 전달받은 변수들로 fdirec 이라는 디렉토리에 파일 이름이 fname 이며 고유 번호가 fnum 인 노드가 있는지 확인하는 함수이다. 해당 노드를 탐색하여 존재하면 1 을 반환하고, 존재하지 않으면 0 을 리턴한다. 즉, 0 을 리턴해야만 중복 노드가 아니므로 Insert1dLoaede_List_Node()가 수행될 수 있다.

Loaded_List_Node* Insert1dLoaede_List_Node(string fname, string fdirec, int fnum)은 1 차원 링크드 리스트(Loaded_List)에 노드를 삽입하는 함수로, head 인 "img_files" 디렉토리에 노드들을 삽입한다. head 부터 시작하여 next 로 다음 노드로 이동하며 NULL 인 곳을 새로운 노드를 삽입할 위치로 인식하여 newNode 를 생성 후 삽입한다. newNode 의 filedirec=fdirec, filename=fname, filenum=fnum 이 되고, next=NULL, prev 은 이전 노드를 가리킨다. 성공적으로 노드 삽입 후에는 head 인 "img_files" 디렉토리 노드의 nodenum 이 1 만큼 증가한다.

위와 같이 filesnumbers.csv 에서 모든 정보를 읽어와 LOAD 를 완료하면, img_files 디렉토리를 log.txt 에 출력하여 LOAD 가 성공적으로 수행됐음을 안내한다. 이때 출력은 void PrintListAfterLoad()로 하는데 이 함수는 head 부터 시작하여 "img_files" 디렉토리 내 모든 노드들을 next 로 이동하며 정보를 출력한다. 이렇게 출력 후 filesnumbers.csv 파일을 닫으며 LOAD 명령이 완료된다.

ADD 는 새로운 csv 파일이 있는 디렉토리와 csv 파일의 이름을 인자로 받는다. while 문을 이용하여 명령에서 이 인자를 찾아낸다. 또한 해당 csv 파일을 열고, 파일이 비어있거나 노드를 삽입할 Loaded_List 가 존재하지 않을 때 에러 코드를 출력한다.

LOAD 에서와 마찬가지로 새로운 디렉토리 노드를 InsertDirecNode() 함수로 만든 후, 반복문을 이용하여 csv 파일에서 파일 이름과 고유 번호를 읽어온 후 중복 노드인지 확인 후 Loaded_List 에 삽입한다.

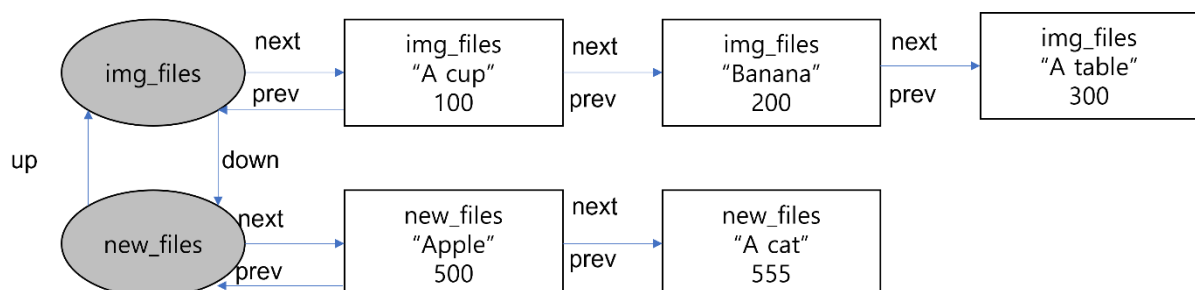
이때는 Loaded_List_Node * Insert2dLoaded_List_Node_withoutIndex(string fname, string fdirec, int fnum) 함수를 사용한다.

이 함수는 Loaded_List_Node* Insert1dLoaded_List_Node (string fname, string fdirec, int fnum)함수와 유사한데, head 인 "img_files" 디렉토리가 아닌 "new_files" 처럼 새로운 디렉토리에 삽입해야 하므로 head 에서 down 으로 해당 Directory Node 를 찾는다. 그 노드에서 next 로 리스트를 채워나간다. 이 외의 과정은 Insert1dLoaded_List_Node()와 같다. 모든 노드가 삽입되면 SUCCESS 문구를 출력한다.

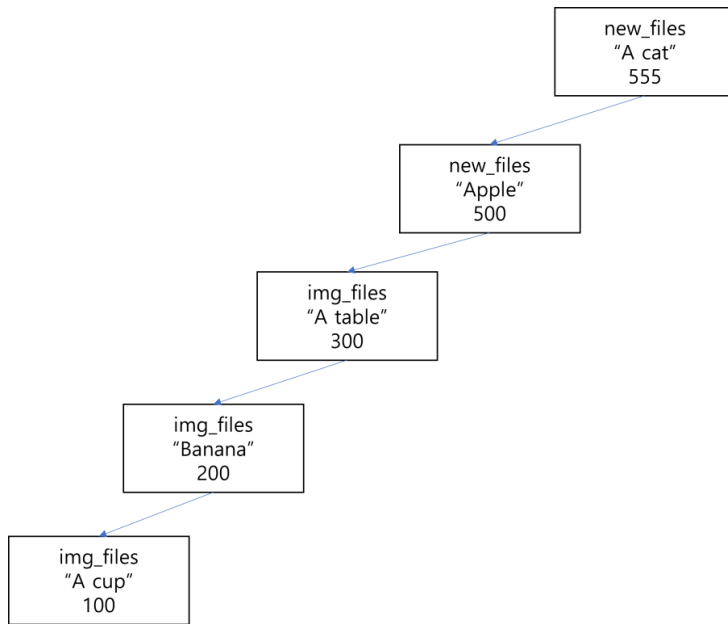
MODIFY 는 인자로 받은 파일 이름에 해당하는 노드를 삭제하고, 또 다른 인자인 고유 번호에 맞는 새로운 노드를 추가하는 명령어이다. 인자로 입력받은 디렉토리 이름과 파일 이름에 해당하는 노드를 삭제 하기 위해 먼저 ExistLoaded_List_Node()로 존재하는 노드인지 확인한다. 존재한다면, Delete2dNode() 함수로 해당 노드를 삭제한다. 노드 삭제는 int Delete2dNode(string fdirec, string fname) 함수에서 수행되는데 해당 노드를 next 와 down 을 이용하여 찾은 후 그 노드의 이전 노드와 삭제하려는 노드의 next 를 연결 후 삭제하려는 노드를 delete 로 할당 해제하여 삭제한다.

그 후 새로운 고유 번호에 따라 새로운 노드를 만들어 삽입한다. 이때 함수는 ADD 에서와 같은 Insert2dLoaded_List_Node_wihoutIndex(...) 함수를 사용한다. MODIFY 가 완료되면 SUCCESS 문구를 출력한다.

MOVE 는 Loaded_List 에 있는 정보를 Database_BST 에 옮기는 명령이다.



이 Loaded_List 를 BST 로 옮기면 아래 그림과 같이 될 것이다.



Database_BST 의 멤버 변수를 살펴보자. 왼쪽 사진은 Database_BST_Node 의 public 변수로 Database_BST 에서 접근이 가능한 변수들이다. left 와 right 는 각 노드의 왼쪽, 오른쪽 등 방향을 가리킨다. filename, filedirec, filenum 은 Loaded_List_Node 에서와 같다. 오른쪽 사진은 Database_BST 의 private 멤버 변수로 BST 의 시작인 root, 마지막 노드인 leaf, BST 에 저장된 총 노드 개수인 nodelist 이 있다.

```

class Database_BST_Node {
public:
    Database_BST_Node* left;
    Database_BST_Node* right;
    string filename;
    string filedirec;
    int filenum;
};

private:
    Database_BST_Node* root;
    Database_BST_Node* leaf;
    int nodelist; //nodelist should be <=300
  
```

Loaded_List 의 마지막 노드 tail 부터 BST 에 저장되므로 tail 노드가 곧 BST 의 root 가 된다. 파일 고유 번호를 기준으로, 부모 노드보다 작으면 왼쪽으로, 크면 오른쪽으로 이동하여 저장된다.

Loaded_List 에 있는 모든 노드들의 정보를 BST 삽입 함수에 전달하기 위하여 Loaded_List 를 LinkedList.txt 에 출력하였다. 이때 void SaveList() 함수를 이용하였다. 그리고 이 txt 파일을 열어 파일 이름, 디렉토리, 고유 번호를 읽어내고 BST 에 InsertNode() 함수로 노드를 삽입하였다. 정보들을 읽어내는 것은 '/'를 기준으로 파일 이름, 디렉토리, 파일 고유 번호를 구분하였다. Loaded_List 에서 삽입할 때와 마찬가지로 BST 에서도 CheckNodeExists()로 중복되는 노드인지 확인 후 void InsertNode(string fdirec,

string fname, int fnum)으로 노드를 삽입하였다. 이 함수는 BST 의 root 부터 시작하여 해당 노드(부모 노드)의 고유 번호보다 인자로 전달받은 고유 번호가 작으면 왼쪽으로, 크면 오른쪽으로 이동하여 적절한 위치를 찾은 후 노드를 삽입한다.

MOVE 가 끝난 후에는 SUCCESS 문구를 출력하고, 예외 상황이 생기면 에러 문구를 출력한다.

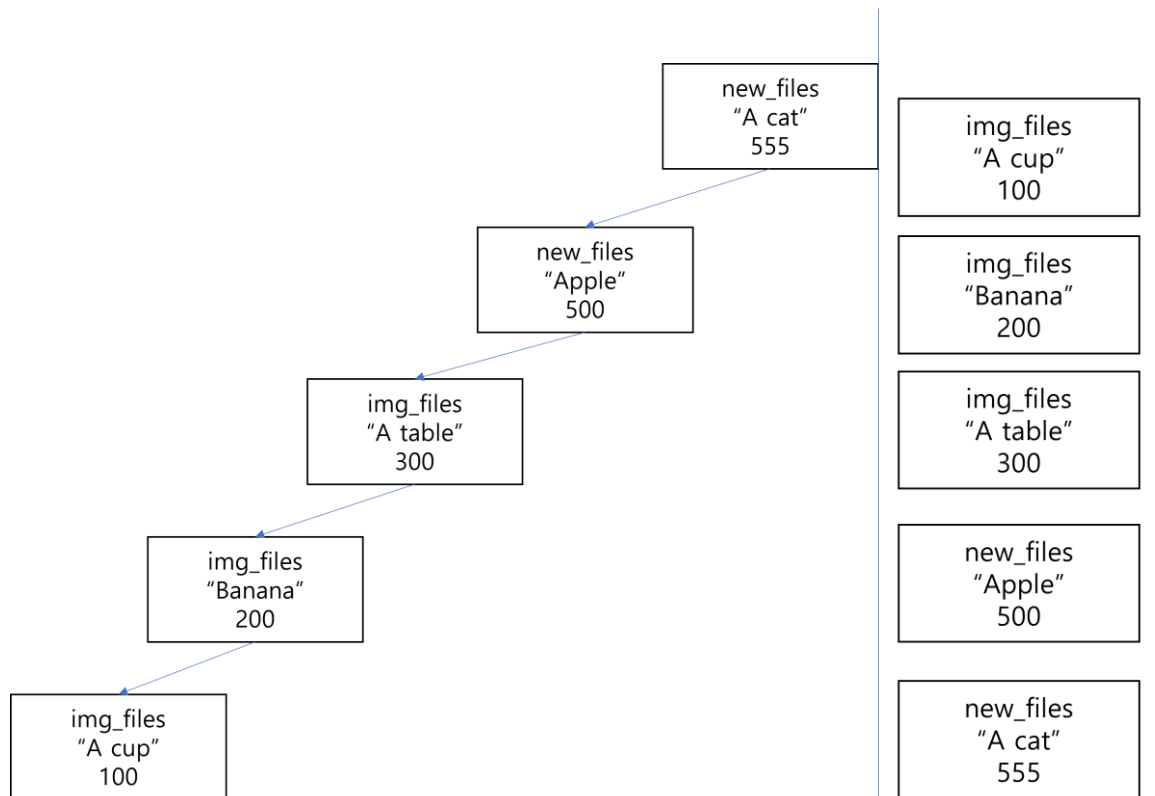
PRINT 는 Database_BST 의 노드들을 트리 중위 순회(In-order) 방식으로 출력한다. 중위 순회는 왼쪽 하위 트리, 루트, 오른쪽 하위 트리 순서대로 순회하는 방식이다. void PrintInorder() 함수와 void Inorder(Database_BST_Node* node)함수에서 진행되는데 PrintInorder()에서 Inorder(root)로 함수를 호출하면 BST 의 root 에서부터 순회를 시작하게 된다. Inorder() 함수는 재귀 함수를 사용하여 구현할 수 있는데 왼쪽, 해당 노드, 오른쪽 노드로 이동함에 유의하여 함수를 호출하거나 해당 노드의 정보를 출력한다.

위 두 함수는 콘솔창에 출력하는 함수로, log.txt 에 기록하기 위해 print.txt 에 출력값을 기록한다. 그리고 print.txt 의 값을 복사하여 log.txt 에 기록함으로써 출력을 마무리한다.

SEARCH 는 특정 단어가 포함된 파일의 이름을 탐색하여 출력하기 위한 명령어이므로 인자로 전달받은 단어가 노드에 있는지 탐색한다. 이때 트리를 반복문을 이용한 후위 순회(Iterative post-order)방식으로 순회하며 큐에 파일 이름과 고유 번호가 담긴 노드를 삽입한다. 모든 노드를 저장했다면 순차적으로 POP 을 진행하며 보이어 무어 알고리즘으로 검색하고자 하는 단어가 노드에 존재하는지 확인한다. 존재하면 출력 후 pop 하며, 모든 노드 확인이 끝나면 큐를 비운다.

보이어 무어 알고리즘은 특정 문자열에서 원하는 패턴을 찾는 알고리즘으로 문장과 같은 character 를 가진 문장 패턴만을 비교해가며 본래 문장에서 찾고자 하는 패턴을 찾아내는 방식이다.

SEARCH 입력을 받아 검색할 단어를 입력 받는다. 큐에 모든 노드들을 Iterative post-order traversal 방식으로 위에서 본 BST 는 아래와 같은 형태로 큐에 담길 것이다. 만약 "A c"라는 단어를 찾는다면 "A cup"과 "A cat"노드를 출력한다. 이때 탐색은 보이어 무어 알고리즘에 따라 파일 노드들을 string 에서 char[] 형으로 변환하여 한 글자 씩 비교한다. 문장의 패턴과 검색하고자 하는 단어의 패턴을 문장의 맨 앞부터 한 칸 씩 이동하면서 대조한다. 패턴이 불일치 하는 경우 일치한 만큼의 문자만큼 이동한 후에 다시 대조함으로써 빠르게 탐색이 가능하다.



SELECT 는 EDIT 명령어를 위해 Database_BST 에서 파일 고유 번호를 기반으로 전위 순회 (pre-order)를 통해 이미지의 경로를 찾아서 이미지를 불러오는 명령어이다. 입력 받은 고유 번호를 해당 고유 번호를 가진 노드가 존재하는지 CheckNodeExists()로 확인한 후 하여 getFileName()과 getFileDirec()으로 해당 파일의 이름과 디렉토리를 찾아낸다.

string getFileDirec(int fnum)은 고유 번호를 fnum 으로 파일 고유 번호를 전달 받고 해당 호가 나올 때까지 root 에서부터 탐색해나간다. 해당 노드를 찾으면 파일이 있는 디렉토리 이름을 반환한다. string getFileName fnum)은 고유 번호를 fnum 으로 파일 고유 번호를 전달 받고 해당 번호가 나올 때까지 root 에서부터 탐색해나간다. 해당 노드를 찾으면 파일이 있는 파일 이름을 반환한다. 이렇게 찾은 디렉토리 이름과 파일 이름은 input_imgage_filedirec 과 input_image_filename 변수에 저장된다.

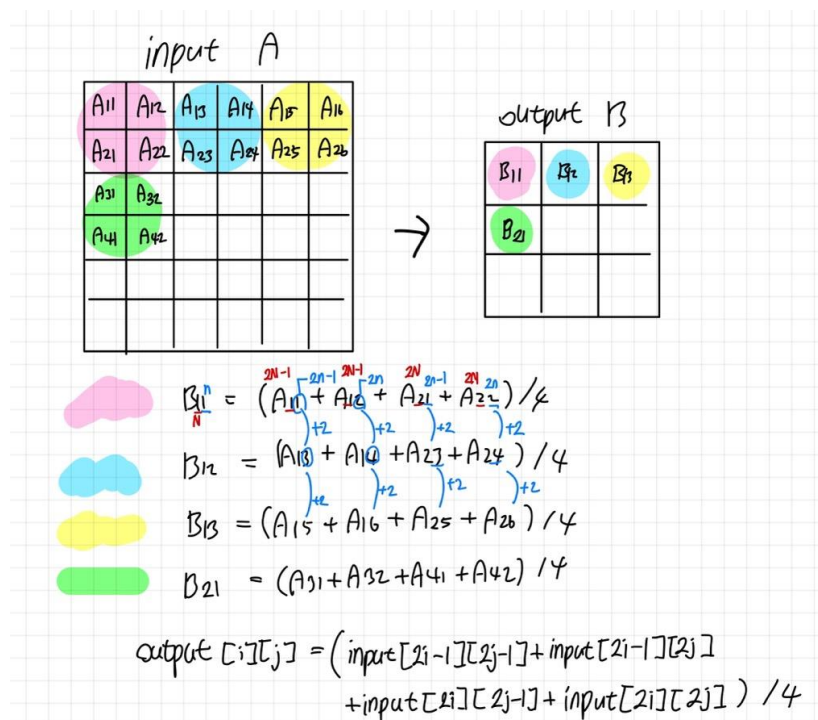
EDIT 은 SELECT 에서 선택된 RAW 파일을 편집하는 명령어이다. 명령어 옵션은 -l, -f, -r 총 3 가지로 각각 밝기 조정, 점대칭, 크기 조정이다.

각 명령어 옵션을 입력받아 어떤 편집을 할지 결정된다면 이미지 파일을 연다. 이때 이 이미지 파일 input_image_file 은 SELECT 에서 구한

input_imgage_filedirec+"/"+input_image_filename+"RAW"

위 형식으로 프로젝트 폴더에서 특정 디렉토리 내 특정 파일을 지목하여 수정할 수 있다.

크기 조정(-r)은 원래 사진의 1/4 크기로 줄이는 기능으로, 원래 이미지 파일이 256*256 의 크기이므로 수정한 이미지는 128*128 이어야 한다. 이를 구현하기 위해 fread 로 input_image_file 에서 데이터를 읽어와 256*256 의 배열로 저장한다. 128*128 크기로 압축하기 위해서는 이 2 차원 배열에서 근접한 4 개의 원소를 더해 평균을 구한 값을 저장하면 된다. 아래 그림으로 규칙을 파악할 수 있다. 이미지 픽셀을 data 라고 표현한다.



i 가 128 만큼, j 가 128 만큼 이중 반복문으로 도는 동안

output_data[i][j]=(input_data[2i-1][2j-1]+input_data[2i-1][2j]+input_data[2i][2j-1]+input_data[2i][2j])/4 값을 할당하면 압축된 이미지 파일을 얻을 수 있다.

이렇게 편집이 완료된 이미지는 Result 디렉토리 내에 "_resized"라는 이름이 추가로 붙어 저장된다.

점대칭(-f)은 각 이미지 픽셀을 스택 구조에 저장한 후 스택에서 꺼내면 점대칭이 된다. 원본 raw 파일의 크기는 256*256 이므로 총 256*256 개의 노드가 스택에 저장된다. 이때 사용되는 클래스는 Stack 과 Node 이다. 반복문으로 256*256 개의 input_data 를 모두 스택에 Push()한다. 이때 Push() 함수는 새로운 노드가 스택의 처음 노드인 head 가 된다. 이후 Pop()하면서 스택의 head node 에 저장된 값을 Top()을 통해 output_data()에

저장한다. Pop()은 head 를 삭제하는 함수이고, Top 은 head node 의 데이터 값을 반환하는 함수이다.

밝기 조정(-l)은 큐에 각 이미지 픽셀을 입력 후 팝하며 특정 숫자를 더해 이미지를 밝게 만든다. 이때 밝기값이 255 이 넘어가는 경우에는 255 로 고정한다. 밝기 조정은 Queue 클래스를 이용해 구현한다. EditEnqueue(double x, int light)로 큐에 노드를 삽입한다. 이때 x 는 input_data, light 는 밝은 정도이다. 새로운 노드는 원래 input_data 의 값에 light 만큼 더해진 값을 얻는다. 이 값은 큐의 마지막 노드인 rear 에 저장된다.

Dequeue()를 통해 큐는 head 부터 차곡차곡 편집된 데이터를 output_data 에 저장한다. 이를 통해 밝기 조절이 가능하다.

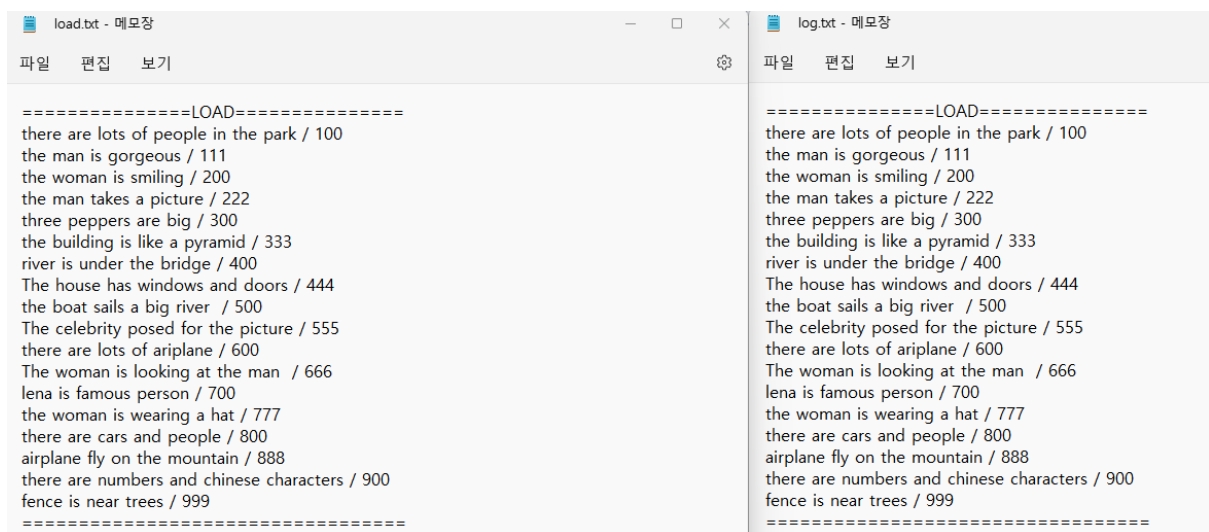
EXIT 를 입력받으면 안내 문구를 출력하고 모든 파일을 닫으며 반복문을 탈출한다.

4. Result Screen

아래 결과 화면들은 윈도우, visual studio 에서 실행한 화면이다.

A. LOAD

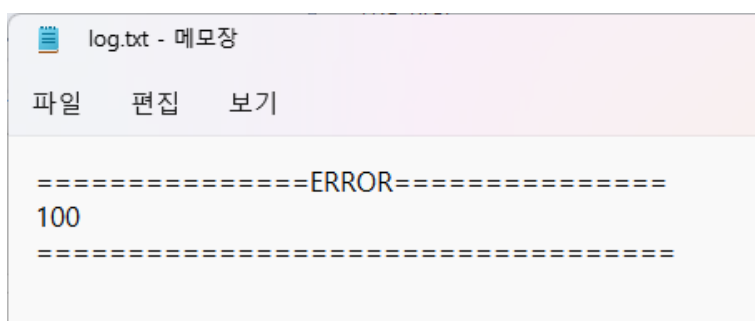
img_files 폴더 내 filesnumbers.csv 파일을 읽고, load.txt 에 기록 후 log.txt 에 복사하는데 성공했다.



The image shows two side-by-side screenshots of a text editor window titled 'load.txt - 메모장' (load.txt - Notepad) and 'log.txt - 메모장' (log.txt - Notepad). Both windows display the same text, which is a list of sentences followed by a number, separated by a slash. The text is enclosed in a block of lines starting and ending with '====='.

```
=====  
there are lots of people in the park / 100  
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  
fence is near trees / 999  
=====
```

해당 파일이 존재하지 않는 경우 아래와 같이 에러 코드를 출력한다.



The image shows a screenshot of a text editor window titled 'log.txt - 메모장' (log.txt - Notepad). The window displays an error message, which is a line of text starting and ending with '====='.

```
=====  
100  
=====
```

B. ADD

새로운 csv 파일을 읽어 Loaded_List 에 저장한 경우 아래와 같이 성공 문구가 출력된다.

```
log.txt - 메모장
파일 편집 보기

=====LOAD=====
there are lots of people in the park / 100
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
fence is near trees / 999
=====
=====ADD=====
SUCCESS
=====

====Directory Folder=====
img_files 18
new_files 6
=====

====Print List=====
img_files / there are lots of people in the park / 100
img_files / the man is gorgeous / 111
img_files / the woman is smiling / 200
img_files / the man takes a picture / 222
img_files / three peppers are big / 300
img_files / the building is like a pyramid / 333
img_files / river is under the bridge / 400
img_files / The house has windows and doors / 444
img_files / the boat sails a big river / 500
img_files / The celebrity posed for the picture / 555
img_files / there are lots of ariplane / 600
img_files / The woman is looking at the man / 666
img_files / lena is famous person / 700
img_files / the woman is wearing a hat / 777
img_files / there are cars and people / 800
img_files / airplane fly on the mountain / 888
img_files / there are numbers and chinese characters / 900
img_files / fence is near trees / 999

new_files / there are flowers / 125
new_files / the man is drinking coffee / 175
new_files / the girl is dancing / 225
new_files / the man takes a picture / 250
new_files / three peppers are big / 325
new_files / the building is like a pyramid / 350
=====

=====ADD=====
SUCCESS
=====
```

위 콘솔 창 화면은 검증을 위한 출력 화면이다. 생성된 노드 개수와 ADD 후 Loaded_List 의 노드들을 볼 수 있다. ===Directory Folder=== 문구 아래로 img_files 디렉토리 노드 다음으로 총 18 개의 노드가 LOAD 시 생성됐음을 확인할 수 있다. ===Print List=== 문구 아래로 현재 Loaded_List 에 있는 노드들을 디렉토리 노드를 제외하고 출력하도록 하였다. 프로젝트 공지 시 주어진 new_filesnumbers.csv 와 함께 테스트하였으며 노드들이 올바르게 삽입 및 연결됐음을 확인할 수 있다.

입력받은 파일이 존재하지 않으면 log.txt 에 아래와 같은 에러 문구를 출력한다.

```
=====ERROR=====
200
=====
```

C. MODIFY

```
MODIFY img_files "A cap on the table" 506
MODIFY new_files "A cup on the table" 404
MODIFY img_files "there are numbers and chinese characters" 950
```

위와 같은 명령을 수행했을 때

```
=====ERROR=====
300
=====

=====ERROR=====
300
=====

=====MODIFY=====
SUCCESS
=====
```

위 2 개는 없는 노드이므로 에러를 출력하고, 마지막은 존재하는 노드이므로 고유 번호를 수정한다. 아래 화면에서 List 의 수정 전 모습과 수정 후 모습을 출력하였는데 숫자가 900 에서 950 으로 바뀌었음을 확인할 수 있다.

| Microsoft Visual Studio 디버그 콘솔 | 선택 Microsoft Visual Studio 디버그 콘솔 |
|--|---|
| <pre>img_files / there are numbers and chinese characters / 950 =====Print List===== img_files / there are lots of people in the park / 100 img_files / the man is gorgeous / 111 img_files / the woman is smiling / 200 img_files / the man takes a picture / 222 img_files / three peppers are big / 300 img_files / the building is like a pyramid / 333 img_files / river is under the bridge / 400 img_files / The house has windows and doors / 444 img_files / the boat sails a big river / 500 img_files / The celebrity posed for the picture / 555 img_files / there are lots of ariplane / 600 img_files / The woman is looking at the man / 666 img_files / lena is famous person / 700 img_files / the woman is wearing a hat / 777 img_files / there are cars and people / 800 img_files / airplane fly on the mountain / 888 img_files / there are numbers and chinese characters / 900 img_files / fence is near trees / 999 new_files / there are flowers / 125 new_files / the man is drinking coffee / 175 new_files / the girl is dancing / 225 new_files / the man takes a picture / 250 new_files / three peppers are big / 325 new_files / the building is like a pyramid / 350</pre> | <pre>=====Print List===== img_files / there are lots of people in the park / 100 img_files / the man is gorgeous / 111 img_files / the woman is smiling / 200 img_files / the man takes a picture / 222 img_files / three peppers are big / 300 img_files / the building is like a pyramid / 333 img_files / river is under the bridge / 400 img_files / The house has windows and doors / 444 img_files / the boat sails a big river / 500 img_files / The celebrity posed for the picture / 555 img_files / there are lots of ariplane / 600 img_files / The woman is looking at the man / 666 img_files / lena is famous person / 700 img_files / the woman is wearing a hat / 777 img_files / there are cars and people / 800 img_files / airplane fly on the mountain / 888 img_files / fence is near trees / 999 img_files / there are numbers and chinese characters / 950 new_files / there are flowers / 125 new_files / the man is drinking coffee / 175 new_files / the girl is dancing / 225 new_files / the man takes a picture / 250 new_files / three peppers are big / 325 new_files / the building is like a pyramid / 350 =====</pre> |

D. MOVE, PRINT

PRINT 명령을 통해 MOVE 명령이 이루어졌는지 확인할 수 있다.

BST 에 옮기기 위해 Loaded_List 를 tail 부터 출력하여 LinkedList 에 저장한 결과다. 위 화면과 비교하였을 때 제대로 저장됐음을 알 수 있다. 오른쪽 화면은 log.txt 로 MOVE 성공 출력 후 PRINT 명령 역시 In-order 에 따라 제대로 출력됐음을 확인할 수 있다.

| LinkedList.txt - 메모장 | |
|--|--|
| 파일 편집 보기 | |
| new_files/the building is like a pyramid/350/ new_files/three peppers are big/325/ new_files/the man takes a picture/250/ new_files/the girl is dancing/225/ new_files/the man is drinking coffee/175/ new_files/there are flowers/125/ img_files/there are numbers and chinese characters/950/ img_files/fence is near trees/999/ img_files/airplane fly on the mountain/888/ img_files/there are cars and people/800/ img_files/the woman is wearing a hat/777/ img_files/lena is famous person/700/ img_files/The woman is looking at the man /666/ img_files/there are lots of ariplane/600/ img_files/The celebrity posed for the picture/555/ img_files/the boat sails a big river /500/ img_files/The house has windows and doors/444/ img_files/river is under the bridge/400/ img_files/the building is like a pyramid/333/ img_files/three peppers are big/300/ img_files/the man takes a picture/222/ img_files/the woman is smiling/200/ img_files/the man is gorgeous/111/ img_files/there are lots of people in the park/100/ | =====MOVE===== SUCCESS ===== =====PRINT===== img_files / "there are lots of people in the park" / 100 img_files / "the man is gorgeous" / 111 new_files / "there are flowers" / 125 new_files / "the man is drinking coffee" / 175 img_files / "the woman is smiling" / 200 img_files / "the man takes a picture" / 222 new_files / "the girl is dancing" / 225 new_files / "the man takes a picture" / 250 img_files / "three peppers are big" / 300 new_files / "three peppers are big" / 325 img_files / "the building is like a pyramid" / 333 new_files / "the building is like a pyramid" / 350 img_files / "river is under the bridge" / 400 img_files / "The house has windows and doors" / 444 img_files / "the boat sails a big river " / 500 img_files / "The celebrity posed for the picture" / 555 img_files / "there are lots of ariplane" / 600 img_files / "The woman is looking at the man " / 666 img_files / "lena is famous person" / 700 img_files / "the woman is wearing a hat" / 777 img_files / "there are cars and people" / 800 img_files / "airplane fly on the mountain" / 888 img_files / "there are numbers and chinese characters" / 950 img_files / "fence is near trees" / 999 ===== |

E. SEARCH

SEARCH 명령은 구현하지 못하여 검색하고자 하는 단어와 에러 코드만 출력하도록 하였다.

```
=====ERROR=====
600
=====
```

```
SEARCH: on the
=====ERROR=====
600
=====
```

F. SELECT

```
SELECT 404
SELECT 800
SELECT 111
```

위의 3 가지 고유 번호를 탐색하게 하였다. 이때 404 는 존재하지 않으므로 에러 코드가 출력하고 나머지 2 개는 SELECT 에 성공했다. 콘솔 화면에 선택된 파일의 정보 역시 출력되게 하여 올바른 파일이 선택됐음을 확인하였다.

```
=====ERROR=====
700
=====

=====SELECT=====
SUCCESS
=====

=====SELECT=====
SUCCESS
=====
```

```
=====ERROR=====
700
=====

input_image_filedirec: ./img_files input_image_filename: there are cars and people
=====SELECT=====
SUCCESS
=====

input_image_filedirec: ./img_files input_image_filename: the man is gorgeous
=====SELECT=====
SUCCESS
=====
```

G. EDIT

SELECT 로 선택된 사진은 위 화면으로 확인할 수 있다. img_files 폴더의 the man is gorgeous.RAW 라는 사진이다. 이 사진으로 결과를 검증하겠다. 원본 사진은 아래와 같다.



a. 점대칭



b. 밝기 조정(+100)



c. 크기 압축



위처럼 모두 편집에 성공했으며 log.txt 에 성공 문구를 출력했다.

```
=====EDIT=====
SUCCESS
=====

=====EDIT=====
SUCCESS
=====

=====EDIT=====
SUCCESS
=====
```

H. EXIT


```
=====EXIT=====
SUCCESS
=====
```

EXIT 에 성공하며 프로그램이 종료됐다.

리눅스 내 log.txt 결과 화면

```
=====LOAD=====
there are lots of people in the park / 100
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
fence is near trees / 999
=====

=====ADD=====
SUCCESS
=====

=====ERROR=====
300
=====

=====ERROR=====
300
=====

=====ERROR=====
300
=====

=====MODIFY=====
SUCCESS
=====

=====ERROR=====
300
=====

=====MOVE=====
SUCCESS
```

```
=====PRINT=====
img_files / "there are lots of people in the park" / 100
img_files / "the man is gorgeous" / 111
new_files / "there are flowers" / 125
new_files / "the man is drinking coffee" / 175
img_files / "the woman is smiling" / 200
img_files / "the man takes a picture" / 222
new_files / "the girl is dancing" / 225
new_files / "the man takes a picture" / 250
img_files / "three peppers are big" / 300
new_files / "three peppers are big" / 325
img_files / "the building is like a pyramid" / 333
new_files / "the building is like a pyramid" / 350
img_files / "river is under the bridge" / 400
img_files / "The house has windows and doors" / 444
img_files / "the boat sails a big river " / 500
img_files / "The celebrity posed for the picture" / 555
img_files / "there are lots of ariplane" / 600
img_files / "The woman is looking at the man " / 666
img_files / "lena is famous person" / 700
img_files / "the woman is wearing a hat" / 777
img_files / "there are cars and people" / 800
img_files / "airplane fly on the mountain" / 888
img_files / "there are numbers and chinese characters" / 950
img_files / "fence is near trees" / 999
=====
```

```
=====ERROR=====
700
=====
```

```
=====SELECT=====
SUCCESS
=====
```

```
=====SELECT=====
SUCCESS
=====
```

```
=====EDIT=====
SUCCESS
=====
```

```
=====EDIT=====
SUCCESS
=====
```

```
=====EDIT=====
SUCCESS
=====
```

```
=====EDIT=====
SUCCESS
=====
```

```
=====EXIT=====
SUCCESS
=====
```

```
█
```

5. Consideration

프로젝트를 진행하면서 코딩에 어려움이 몇 가지 있었다.

ADD 명령어 수행 시 인자가 부족하면 에러코드를 출력해야 한다. ADD에서는 파일 CSV 파일 디렉토리와 CSV 파일 이름 두 가지를 인자로 받아야 하므로

```
cin>>filedirectory;    //파일 디렉토리
```

```
cin>>new_csvfile;      //CSV 파일 이름
```

위와 같이 인자를 받아 만약 둘 중 하나라도 "w0"값을 가진다면 인자가 부족한 것으로 판단하려고 했다. 이런 경우에 catch 문으로 넘겨 에러 코드를 출력하려 했으나 애초에 인자가 입력이 되지 않으면 cin 이 있는 line 이 끝날 수 없어 try 문을 벗어날 수 없다는 것을 디버그 모드를 통해 확인하게 됐다.

ADD 에서 CSV 파일을 올바른 값을 입력받았다고 생각했으나 파일이 열리지 않는 문제가 있었다. for 문을 통해 csv 파일 이름을 입력받던 중 'wr'값이 들어가는 것을 확인하였다. 이 'wr'은 캐리지 리턴으로, 커서가 첫 부분으로 이동 후 출력하게 된다. 따라서 for 문이 끝나는 조건에 'wr'을 만나는 것을 포함하니 문제가 해결되었다.

2 차원 링크드 리스트 구성 시 인덱스를 인자로 받아 몇 번째 노드에 삽입할지를 결정할 수 있도록 Insert2dLoaded_List_Node(...);이라는 노드 삽입 함수를 정의했었다. 디렉토리 폴더 노드에는 오직 디렉토리 내 노드의 개수와 디렉토리의 이름만 저장하고, 실질적인 파일 정보들은 디렉토리 노드의 next 에 저장하고자 하였다. 이때 인덱스를 0 부터 시작하면 head 부터 값이 바뀌기 때문에 오류가 발생하였다. 따라서 링크드 리스트에서는 순서가 무의미하다고 생각하여 인덱스 필요없이 노드를 삽입할 수 있도록 함수를 수정하였다. 이렇게 Insert2dLoaded_List_Node_wihoutIndex(...) 함수를 선언 및 정의하였다.

Load_Linked_List 의 노드들을 어떻게 Database_BST 로 옮길지도 중요한 문제이다. 처음으로 떠오른 방법은 노드 하나씩 get 함수로 정보를 받아 그 정보들을 차례대로 BST 에 넣는 것이었다. 이 방법의 문제는 get 함수로 얻어야 할 변수가 무려 3 개나 되서 데이터 타입을 어떻게 정할 건지에 대한 문제와 같은 노드의 정보들을 한 번에 받아야 되는데 이걸 구현하기에는 어려움이 있다고 판단하였다.

Loaded_List 가 LOAD 후 프린트되는 것에서 아이디어를 얻었다. log.txt 처럼 LinkedList.txt 파일을 만들어 여기에 노드들을 모두 출력하고 이 파일을 읽어와 BST 에 저장하는 방법이다. 이때 디렉터리 노드는 BST 에 저장되지 않도록 주의해야 했다.

이때 Loaded_List 의 tail(마지막 노드)부터 BST 에 저장해야 했기 때문에 LinkedList.txt 에도 tail 부터 저장해야 했다. 그러기 위해 tail 인 노드를 찾고 head 까지 거슬러올라오는 방식으로 출력하여 문제를 해결했다. 이와 비슷하게 print.txt 와 log.txt 등 log.txt 에 편리하게 기록할 방법을 찾고자 노력하였다.

리눅스에서 core dump 를 확인하기 위해 core 파일을 확인하고자 했으나 코어 파일 사이즈가 0 이라 core 파일이 생성되지 않았다. \$ ulimit -c unlimited 명령어로 이 문제를 해결할 수 있었다.

```
세그멘테이션 오류 (core dumped)
soyoung@ubuntu:~/2021202039_DS_project1$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals         (-i) 15409
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes      (-u) 15409
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

```
soyoung@ubuntu:~/2021202039_DS_project1$ ulimit -c unlimited
soyoung@ubuntu:~/2021202039_DS_project1$ ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals         (-i) 15409
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes      (-u) 15409
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

윈도우의 visual studio에서 코드를 작성 후에 우분투ubuntu에서 실행하였는데 윈도우에서는 잘 되던 부분이 우분투에서는 제대로 동작하지 않기도 했다. gdb로 디버깅하며 알아간 결과, 리눅스에서는 NULL 포인터 접근에 대해 더 엄격하다. 윈도우에서는 지정하지 않은 값은 자동으로 NULL값이 되지만 리눅스에서는 그러지 않은 변수가 많아 오류가 많이 발생하였다. 이를 해결하기 위해 next, prev, up, down 등 대부분의 변수의 초기값을 NULL로 지정해주었더니 오류가 해결됐다.

본 프로젝트에서는 C++에서 Linked List와 Binary Search Tree, Stack, Queue 등 여러 자료 구조를 사용하면서 자료 구조에 대한 이해를 높일 수 있었다. 또한 리눅스 우분투를 사용하면서 리눅스의 기본적인 명령어들에 대해 배울 수 있었다.