데이터구조실습

Project2

담당교수 : 이형근 교수님

학 번 : 2021202058

성 명 : 송채영

1. Introduction

이번 프로젝트에서는 B+-tree. 선택 트리(Selection Tree), Heap 을 이용해 도서 대출 관리 프로그램을 구현하는 것을 목표로 한다. 도서명, 도서 분류 코드, 저자, 발행 연도, 대출 권수의 정보를 관리하고 이 정보들을 통해 대출 중인 도서와 대출 불가 도서에 대해 관리한다. 해당 정보들은 loan_book.txt 에 저장되어 있으며, LOAD 명령어를 통해 정보를 저장한다. 도서 분류 코드는 000 부터 600 까지 존재한다. 도서분류코드가 000 에서 200 까지는 3 권까지 대출할 수 있으며, 300 부터 400 까지는 4 권, 500 부터 700 까지는 2 권까지 대출할 수 있다.

B+-Tree 는 대출 중인 도서를 관리하기 위한 자료구조이다. B+-tree 의 차수는 3 이며 도서명을 기준으로 대출 중인 도서를 정렬하고 저장하며, B+-tree 는 인덱스 노드와 데이터 노드로 구성되어 있다. loan_book.txt 파일에 저장된 데이터를 B+-tree 에 저장하며 새로 추가된 데이터가 B+-tree 에 이미 존재하는 경우 대출 권수를 업데이트 하며, 데이터가 없는 경우는 node 를 새로 추가한다. 도서가 모두 대출되어 대출 불가 상태가 되는 경우는 Selection Tree 로 전달한 후 B+-tree 에서 해당 도서를 지운다.

LoanBookHeap 은 Min Heap 으로 구현되어 있고, 도서명을 기준으로 정렬되며 새로운 데이터가 추가될 때는 왼쪽 자식 노드부터 생성한다. B+-tree 에 저장된 데이터 중 대출 불가 도서를 가져와 Heap 을 구축하며, 기존에 Heap 이 존재하지 않는 경우에는 새로 구축하도록 한다. 이때 Min Heap 의 특성에 따라 재정렬을 하도록 한다. 재정렬시에는 모든 부모 노드의 도서명이 자식 노드의 도서명과 작거나 같은 경우에 정렬이 되며, 정렬이 되는 경우 Selection Tree 도 같이 정렬된다.

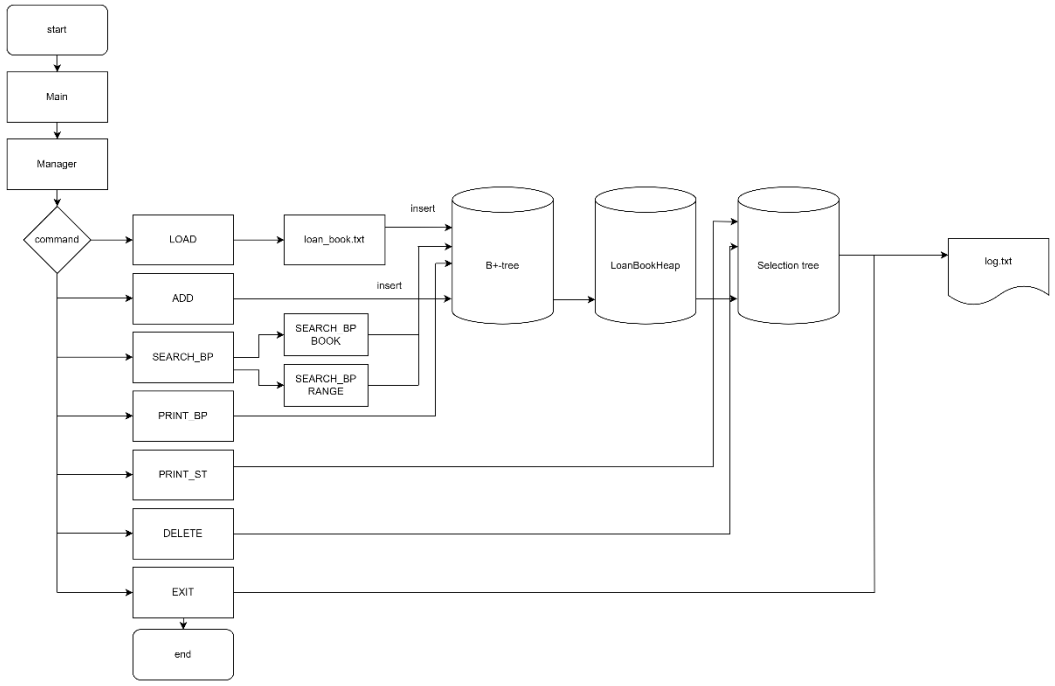
Selection tree 는 대출 불가 도서를 관리하기 위한 자료구조이다. Selection tree 는 도서명을 기준으로 Min Winner Tree 를 구성하고 있으며 각 run 은 도서 분류 코드 개수와 동일하게 설정된다. 대출 불가 도서는 도서 분류 코드에 따라 Min Heap 으로 구현된 Selection tree 의 각 run 에 저장되도록 한다.

프로젝트에서 구현해야 하는 흐름을 간략하게 살펴보면 다음과 같다.
LOAD 명령어는 텍스트 파일 "loan_book.txt"에서 데이터 정보를 읽어올 수 있으며 데이터 파일에 정보가 존재하는 경우, 해당 정보를 B+-tree 자료구조에 저장한다. 이어서 ADD 명령어는 B+-tree 에 데이터를 직접 추가하기 위한 명령어로 도서 이름, 분류 코드, 저자, 발행 연도를 추가로 입력한다. SEARCH_BP 명령어는 B+-tree 에 저장된 데이터를 검색하는 명령어로 1 개 또는 2 개의 인자를 입력한다. 1 개가 입력되는 경우 해당 도서의 검색 결과를 출력하며, 2 개가 입력되는 경우 시작 단어로 시작하는 도서부터 끝 단어로

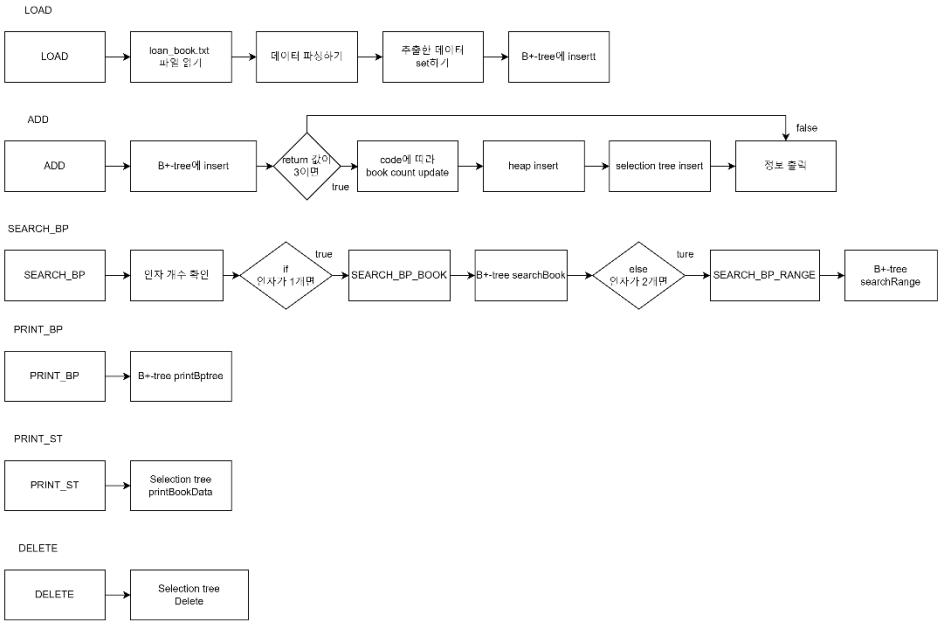
시작하는 도서 사이의 모든 도서들을 출력한. PRINT_BP 명령어는 B+-tree 에 저장된 데이터를 도서명을 기준으로 오름차순으로 출력한다. PRINT_ST 명령어는 Selection tree 에서 인자로 받은 도서분류코드에 해당하는 데이터를 오름차순으로 출력하는 명령어이다. 저장된 회원정보를 찾아 출력하는 명령어이다. DELETE 명령어는 Selection Tree 에서 root node 에 저장된 도서를 제 거하는 명령어이다. 마지막으로 EXIT 명령어는 프로그램 상의 메모리를 해제하며 프로그램을 종료한다.

전체적인 흐름과 명령어에 대한 추가적인 내용은 Flowchart 에서 설명할 것이며, B+-tree, Selection tree, Heap 에 대한 내용은 Algorithm 에서 설명하겠다.

2. Flowchart



이번 프로젝트의 전체적인 흐름을 나타낸 Flowchart 이다. command.txt 를 통해 명령어를 입력 받고 이를 manager.h 에서 관리한다. 각각의 command 는 각 명령어에 맞는 동작을 취하며 위의 흐름도와 같이 동작한다. Manager.cpp 의 Run 함수에서 command.txt 파일을 parsing 하여 command 를 추출한 후 command 와 명령어를 비교하여 각각에 맞는 동작을 한다. 모든 명령어는 성공 시 success 를 출력하며, 실패 시 Error code 를 lo.txt 에 출력한다. 명령어들은 동작을 마친 뒤 종료되지 않고 command 파일이 끝까지 읽힐 때까지 작동한다. 그에 대한 흐름은 log.txt 에서 직접 파악할 수 있으며 성공여부와 실패여부 역시 log 에 남는다.



LOAD, ADD, SEARCH_BP, PRINT_BP, PRINT_ST, DELETE 명령어의 flow chart 이다.

LOAD 는 텍스트 파일의 데이터 정보를 불러오는 명령어로, loan_book.txt 파일에 저장된 data 를 불러온다. 텍스트 파일에 데이터가 존재하지 않는 경우 에러코드를 출력하였다. 이후 loan_book.txt 파일에서 한 줄을 읽어와 tab 단위로 데이터를 추출하며 추출한 데이터를 각각 도서명, 분류 코드, 저자, 발행 연도, 대출 권수 변수에 저장하였다. 추출한 도서명, 분류 코드, 저자, 발행연도에 새로운 값을 할당한 후 대출 권수도 count 수만큼 update 해준 후 b+-tree 에 insert 하였다.

ADD 는 B+-tree 에 도서를 직접 추가하는 명령어로, 도서명, 분류 코드, 저자, 발행 연도를 인자로 가진다. run 함수에서 새로운 도서명, 분류 코드, 저자, 발행연도를 할당한 후 ADD 함수에서 인자로 받은 temp 를 b+-tree 에 insert 하였다. 이때 반환 값이 3 이면 b+-tree 에서 제거되었다는 뜻으로 000~700 에 해당하는 코드에 따라 count 수를 업데이트 해주었다. 제거된 정보들은 heap 에 insert 해주었고 이후 다시 selection tree 에 insert 한 후 정보를 출력하였고, 만약 제거되지 않았다면 정보를 출력해주었다.

SEARCH_BP 는 B+-tree 에 저장된 데이터를 검색하는 명령어로, 1 개 또는 2 개의 인자를 가진다. run 함수에서 인자의 개수를 확인한 후 인자가 1 개라면 SEARCH_BP_BOOK 함수를, 인자가 2 개라면 SEARCH_BP_RANGE 함수를 실행하였다. SEARCH_BP_BOOK 함수에서는 B+-tree searchBook 을, SEARCH_BP_RANGE 함수에서는 B+-tree search Range 를 실행한다. 인자가 1 개면 해당 도서의 검색 결과를 출력하며, 인자가 2 개일 경우 첫번째는 시작 단어, 두 번째는 끝 단어에 해당하고 시작단어로 시작하는 도서명에서부터 끝 단어로 시작하는 도서명 사이에 있는 도서명들을 모두 출력한다.

PRINT_BP 는 B+-tree 에 저장된 데이터를 도서명을 기준으로 오름차순으로 출력하는 명령어로, B+-tree 의 printBptree 를 호출하여 데이터를 출력한다.

PRINT_ST 는 Selection tree 에서 인자로 받은 도서 분류 코드에 해당하는 데이터를 오름차순으로 출력하는 명령어로, Selection tree 의 printBookData 를 호출하여 데이터를 출력한다.

마지막으로 DELETE 는 Selection Tree 에서 root node 에 저장된 도서를 제거하는 명령어로 Selection tree 의 Delete 를 호출한다.

전체적인 코드의 흐름과 각 명령어의 흐름을 설명하였다. 각 명령어에서 호출한 함수들은 Alogoritm 에서 설명하도록 하겠다.

3. Algorithm

- B+-tree

  B+-tree 자료구조를 이용하여 insert, search, print, delete 를 구현하였다. 우선 insert 는 새로운 데이터(newData)를 B+tree 에 삽입하는 함수로, 트리가 비어 있을 경우 새로운 데이터를 담을 리프 노드를 생성하고, 해당 데이터를 삽입하였다. 트리가 비어있지 않을 경우 데이터를 삽입할 위치를 찾아가며 적절한 리프 노드를 찾아 삽입하였다. 삽입 후에 노드가 데이터를 저장할 수 있는 용량을 초과하게 되면, 노드를 분할, 즉 split 해야 하는데 이때 인덱스 노드도 필요한 경우 split 하도록 구현하였다. 이어서 excessDataNode 함수는 주어진 데이터 노드가 넘치는지 넘치지 않는지 확인하는 함수로, 데이터 노드에 저장된 데이터 개수가 트리의 차수보다 크면 true 를 반환하고 그렇지 않으면 false 를 반환한다. excessIndexNode 함수는 주어진 인덱스 노드가 넘치는지 넘치지 않는지 확인하는 함수로, 인덱스 노드가 저장된 인덱스 개수가 트리의 차수보다 크면 true 를 반환하고 그렇지 않으면 false 를 반환한다. 다음으로 splitDataNode 함수이다. 해당 함수는 데이터 노드를 분할하는 함수로, 데이터 노드의 데이터를 반으로 나누어 새로운 데이터 노드를 생성하고 나머지 반은 기존 노드에 남겨두도록 구현하였다. 이렇게 함으로써 데이터 노드 간의 관계를 유지하도록 하였다. splitIndexNode 함수는 인덱스 노드를 분할하는 함수로, 인덱스 노드의 인덱스를 반으로 나누어 새로운 인덱스 노드를 생성하고 나머지 반은 기존 노드에 남겨두며 부모 노드와의 연결 및 인덱스 노드 간의 관계를 조정하며 B+-tree 의 균형을 유지하도록 구현하였다. searchDataNode 함수는 주어진 name 을 가진 데이터가 있는지 검색하는 함수이다. 왼쪽에서 시작하여 가장 왼쪽의 리프 노드까지 이동한 후 해당 노드에서 데이터를 찾았다. 만약 데이터가 존재하면 해당

데이터의 리프 노드를 반환하고 존재하지 않으면 NULL 을 반환하도록 하였다. 다음으로 search 함수이다. 우선 searchBook 함수는 주어진 name data 를 활용하여 해당 도서를 B+-tree 에서 검색하는 함수로, 트리를 왼쪽으로 이동해 가장 왼쪽의 리프 노드에 도달한 후 해당 리프부터 순차적으로 도서를 검색하며 도서를 찾으면 true, 찾지 못하면 false 를 반환한다. Search Range 함수는 주어진 시작 도서이름과 끝 도서 이름 사이의 도서들을 검색하여 출력하는 함수로, 트리를 왼쪽으로 이동하여 가장 왼쪽의 리프 노드에 도달한 후, 해당 리프 노드부터 순차적으로 도서를 검색하며, 범위 내의 도서를 찾으면 해당 도서 정보를 출력하고 찾은 여부를 found 에 저장하였다. 범위 내에 더 이상 도서가 없을 때까지 반복하며 found 값을 반환하였다. 다음으로 print 함수이다. 주어진 노드부터 시작해 B+-tree 의 모든 도서 정보를 출력하는 함수로, 왼쪽으로 이동해 가장 왼쪽의 리프 노드에 도달한 후 해당 리프 노드부터 순차적으로 도서를 출력하도록 하였다. 마지막으로 delete 함수이다. 주어진 도서 정보를 B+-tree 에서 삭제하는 함수로, 트리를 왼쪽으로 이동하여 가장 왼쪽의 리프 노드에 도달한 후 해당 리프 노드부터 순차적으로 도서를 검색한다. 이때 찾은 도서를 삭제하고, 필요한 경우 데이터 노드를 조정하여 B+-tree 의 형태를 유지하도록 하였다.

- LoanBookHeap

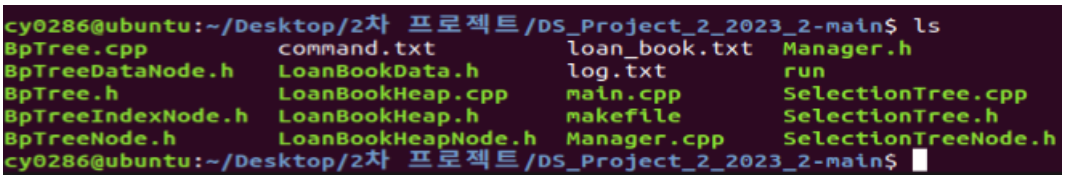
Heap 자료구조를 이용하여 도서 정보를 heap 에 삽입하는 insert 와 Min Heap 형태를 유지하도록 삽입 시 부모 비교하는 함수와 삭제 시 자식과 비교하는 함수를 구현하였다. 우선 heapifyUP 함수와 heapifyDown 함수를 통해 Min Heap 의 상태를 유지하도록 하였는데, 각각의 구현방법은 다음과 같다. 우선 heapifyUp 함수는 삽입한 노드가 부모 노드보다 작은 도서 이름을 가지도록 상위로 이동시키며 힙의 속성을 만족하도록 하였다. 부모 노드와 비교하며 조건이 만족할 때까지 반복하였다. 다음으로 heapifyDown 함수의 경우 삭제 연산시 마지막 노드를 루트로 이동하고 힙 속성을 만족하게 하기 위해 하위로 이동하는데, 현재 노드가 자식 노드들 중에서 작은 도서 이름을 가진 노드와 교환하는 과정을 최소값을 찾을 때까지 반복하였다. Insert 함수는 새로운 도서 정보를 담은 노드를 생성하고, 해당 노드의 도서 정보를 설정하였다. 이때 힙이 비어있으면 새로운 노드가 루트가 되고, 비어있지 않다면 마지막 노드로 삽입 후 heapifyup 을 호출하여 힙 속성을 만족하도록 하였다.

- Selection tree

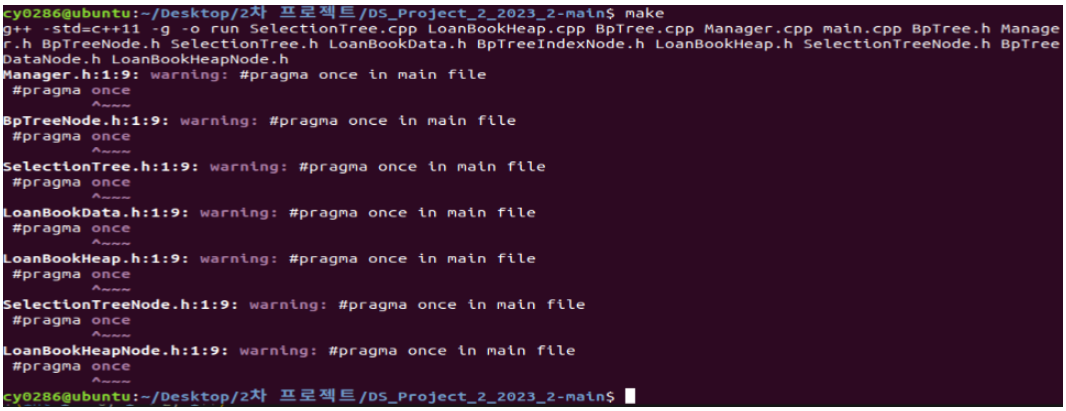
Selection tree 자료구조를 이용하여 insert, print, delete 를 구현하였다. 해당 자료구조는 이진 탐색 트리를 바탕으로 하기 때문에 각 노드가 최대 두개의 자식을

가지며 왼쪽 자식은 현재 노드보다 작은 값을, 오른쪽 자식은 현재 노드보다 큰 값을 갖는 트리 구조이다. 이번 프로젝트에서는 Min Winner Tree 를 구성하는데, 각 노드는 자식 중에서 가장 작은 값을 가지는 노드가 되는 구조를 가져야 한다. 우선 insert 함수는 새로운 도서 정보를 이진 탐색 트리에 삽입한다. 새로운 노드를 생성해 해당 노드의 도서 정보를 설정하며, 트리가 비어 있다면 새로운 노드가 루트가 되고 트리가 비어 있지 않다면 적절한 위치를 찾으며 삽입한다. 이어서 delete 함수는 주어진 도서 코드에서 해당하는 노드를 이진 탐색 트리에서 삭제하며 삭제할 노드가 자식이 없는 경우, 삭제할 노드가 하나의 자식을 가지는 경우, 삭제할 노드가 두 개의 자식을 가지는 경우로 나누어 구현하였다. 삭제할 노드가 두개의 자식을 가지는 경우 오른쪽 서브트리에서 가장 작은 값을 찾는 노드를 반환하기 위해 findSuccessor 함수를 구현하였다. 마지막으로 print 함수는 주어진 도서 코드에 해당하는 노드의 정보를 출력하며, 재귀적으로 탐색하며 현재 노드의 도서 코드와 주어진 도서코드를 비교해 이동 방향을 결정하는 findNode 함수를 추가적으로 구현하였다.

4. Result Screen



프로젝트 파일에서 ls 명령어로 프로젝트 폴더 안에 있는 파일들을 확인하였다.

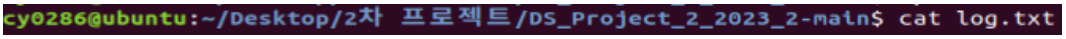


Make 명령어를 통해 컴파일을 한 사진이다. 스켈레톤 코드 안에 #pragma once 가 있어 warning 이 뜨는 것 이외에는 코드가 잘 컴파일 된 것을 확인할 수 있다.

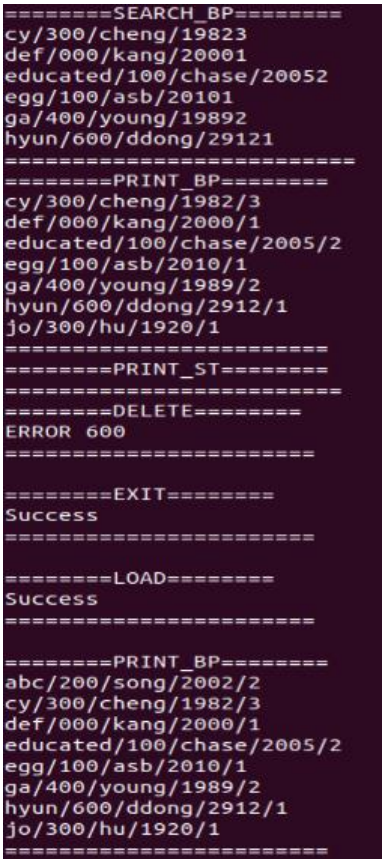
Cat 명령어를 사용해 임시로 구현한 command.txt 파일에 있는 명령어를 출력해보았다.

Cat 명령어를 사용해 임시로 구현한 loan_book.txt 파일에 있는 data 를 출력해보았다. Command.txt 파일과 loan_book.txt 파일 모두 tab 단위로 구분되어 있는 것을 확인할 수 있다.

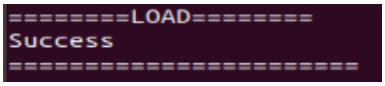
실행하였을 때 결과화면이다.



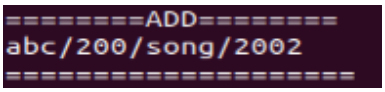
Cat 명령어를 사용해 log.txt 파일에 출력된 결과를 확인해보았다.



우선 출력 포맷에 맞게 출력한 것을 확인할 수 있으며, log.txt 파일이 이미 존재할 경우 텍스트 파일 가장 뒤에 이어서 추가로 저장하는 것을 확인할 수 있다.

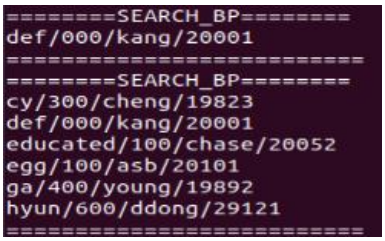


LOAD 명령어 부분이다. loan_book.txt 파일의 데이터들이 성공적으로 load 됐다는 것을 의미한다.

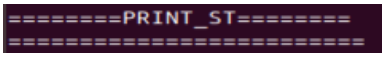


ADD 명령어 부분이다. Command.txt 파일을 바탕으로 성공적으로 데이터가 추가된 것을 알 수 있다.

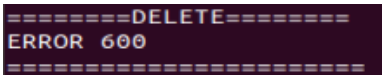
PRINT_BP 명령어를 통해 확인해보면, LOAD 됐을 때 abc 도서가 존재하지만, ADD 명령어로 abc 도서를 추가했을 경우, 도서분류코드가 200 인 도서는 3 권까지 빌릴 수 있기 때문에 사라진 것을 확인할 수 있다. 이를 통해 LOAD 와 ADD 명령어가 잘 작동하는 것을 확인하였다.



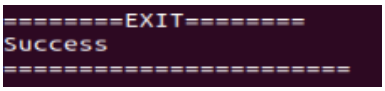
SEARCH_BP 명령어 부분이다. Command.txt 파일에서 첫번째 경우는 인자가 1 개이므로 해당 도서명을 출력하면 되며, 두번째 경우는 인자가 2 개이므로 c 로 시작하는 도서명부터 h 로 시작하는 도서명 사이에 있는 모든 도서들을 출력해야 한다. 결과를 보면 알맞게 출력이 된 것을 볼 수 있다.



PRINT_ST 명령어 부분이다. 에러코드는 출력하지 않으나, 노드를 제대로 출력하지 못하는 것을 확인하였다.



DELETE 명령어 부분이다. PRINT_ST 가 잘 출력되지 않아 delete 가 안 되는 것을 확인하였다.



마지막으로 EXIT 부분이다. 프로그램이 성공적으로 종료되었음을 알 수 있다.

5. Consideration

우선 이번 프로젝트에서 selection tree 에서 인자로 받은 도서분류코드에 해당하는 데이터를 출력하는 PRINT_ST 명령어와 selection tree 에서 root node 에 저장된 도서를 제거하는 명령어인 DELETE 명령어의 결과가 잘 나오지 않아 아쉬웠다. Selection tree 부분은 Min winner tree 형태를 구현하고 insert 를 했어야 됐다는 것을 구현 막바지에 깨달았지만, 수정해서 제출하기까지 시간이 부족했다. 프로젝트 제출 기한과 점수를 떠나서 완성해서 구현해보고 싶다.
프로젝트는 기본적으로 작년 2차 프로젝트를 진행했던 코드 중 B+-tree 부분을 참고하며 구현해 큰 어려움이 있지는 않았지만, 구현하면서 segmentation fault 가 여러 번 떠 어려움이 많았다. Linux 내 vscode 로 진행하였는데, 디버깅을 할 수 없어 하나하나 출력해보며 어느 부분에서 segmentation 오류가 뜨는지 확인해야 했기에 복잡하고 어려웠다. 또한 부가적으로 구현해야 하는 부분, 예를 들어 ADD 했을 때 대출 도서 불가인 경우 B+-tree 에서 삭제하고 selection tree 로 넘겨야 하는데 이런 부분을 구현할 때 아이디어가 많이 필요했던 것 같다. 또한 B+-tree 에서 지우는 부분에서 datanode 와 indexnode 를 모두 지우고 연결해주는 과정이 필요했는데, 이부분을 정확하게 구현하지 못해 일부 예외 케이스의 경우 출력이 잘 되지 않는 경우도 생기는 것 같았다.

또한 이번 프로젝트에서 STL 을 많이 사용하였는데 작년에 map, set, list 등의 STL 들을 실습했던 경험이 있어 어렵지 않고 수월하게 진행할 수 있었고, 어떻게 사용되는지도 다시 한번 복습할 수 있었다.