데이터구조실습 프로젝트#2 보고서

제출일자: 2023년 11월 16일 (목)

학 과: 컴퓨터공학과

담당교수: 이기훈 교수님

실습분반: 금요일 5, 6

학 번: 2022202092

성 명: 연선우

1. Intoroduction

본 프로젝트는 B+-Tree, 선택 트리 및 힙과 같은 자료 구조를 활용하여 효율적인 도서 대출 관리 시스템을 개발하는 것을 목표로 한다. 이 프로그램은 도서 제목, 분류 코드, 저자, 발행 연도 및 대출 횟수와 같은 도서 관련 정보를 관리한다. B+-Tree를 활용하여 현재 대출 중인 도서를 저장하고, 선택 트리 및 힙은 대출 불가 도서를 관리하는 데 사용된다. 입력값은 command.txt, loan_book.txt로 전달되고, 출력은 log.txt에 출력된다. 만약 log.txt가 이미 존재한다면 다음 결과는 이어서 작성하게 된다.

B+-tree로 대출 가능 및 대출 중인 도서를 관리한다. 각각의 book code마다 최대 대출권수가 정해져 있는데, 해당 권수 이상 대출되게 되면 해당 도서 정보를 B+-tree에서 꺼내어 LoanBook Heap 및 Selection Tree에 전달한다. B+-tree의 경우 command.txt의 LOAD 및 ADD에 의해 값이 추가된다. LOAD의 경우 프로그램 실행 당 한번만 실행될 수 있기에 재호출하게 되면 에러코드인 100이 출력된다. ADD 명령어의 경우 한 권씩 도서 정보를 업데이트하며, 중복된 책 제목이 있는 경우 해당 도서의 대출 권수를 한권 늘려준다. 그러다 앞서 말한 최대 대출 권수에 도달하면 B+-tree에서 제거하고 다른 자료구조로 해당 도서를 옮기는 작업을 수행한다. 옮긴 이후의 작업은 아래에 서술하겠다.

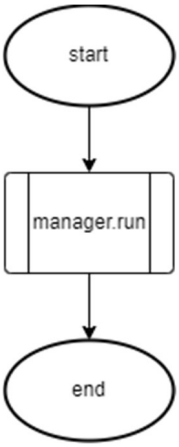
그렇게 이동한 도서는 우선 LoanBook Heap에 저장되어 각각의 코드에 맞는 min heap을 구성한다. 그렇게 총 8개의 Heap이 구성되어 각각 run으로서 Selection tree의 leaf node에 연결된다. 다시 Heap으로 돌아와 새로운 노드가 추가될 때마다 재정렬을 실행한다. 재정렬이 실행되어 root가 바뀐 경우 해당 루트에 저장된 도서 정보를 selection tree의 leaf node에 전달하여 min winner tree로 재구성되게 된다.

Selection tree는 8개의 run에서 가장 작은 값들을 leaf node로 전달받아 min winner tree를 구성한다. 그렇게 selection tree의 root node에는 항상 가장 작은 값을 가지는 도서가 존재하게 된다. 이렇게 구성된 selection tree는 DELETE 명령어를 실행할 때 사용되게 된다. DELETE 명령을 입력 받으면 selection tree의 root부터 시작하여 같은 값을 가지는 자식노드로 이동하며 결국에는 해당 run에 접근하게 된다. 그렇게 접근한 run에서 해당 도서를 삭제하고 힙을 재정렬하여 selection tree의 leaf node에 전달하며 다시 min winner tree 재정렬을 실행하여 기존과 같은 형태를 유지한다.

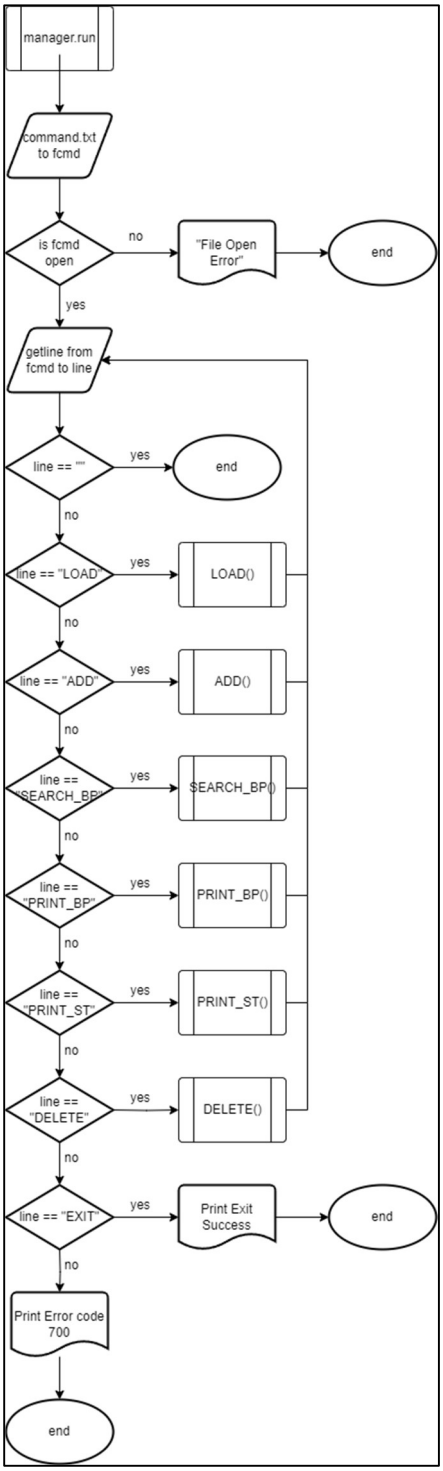
그 외에 SEARCH_BP, PRINT_BP, PRINT_ST는 각각 B+-tree에 저장된 모든 데이터를 출력, B+-tree에서 찾는 도서명의 정보를 출력하거나 알파벳의 범위를 전달하여 해당 범위 내의 정보들을 출력한다. 마지막으로 PRINT_ST는 코드와 함께 입력되어 selection tree의 8개의 run 중에서 입력 받은 코드에 맞는 도서들을 모두 출력한다.

2. Flowchart

Main



Main 함수에서는 모든 명령어를 처리하는 manager 클래스의 run 함수를 호출한다. 해당 함수에서 반환되면 프로그램을 종료한다.



manager.run

manager 클래스의 run 함수 순서도이다. command 매개변수로 전달된 파일을 fcmd로 열고 log.txt 파일을 ios::app 모드로 연다. 해당 모드는 log.txt 파일이 존재하는 경우 뒤에 이어서 로그를 작성해주는 역할을 한다. 만약 파일 열기에 실패하면 "File Open Error"를 flog 파일에 기록하고 함수를 종료한다. 파일 열기를 체크한 후, 무한 루프를 돌면서 command.txt로부터 한 줄씩 명령어를 읽고 해당하는 작업을 실행한다.

명령어가 "LOAD"라면, 입력 형식을 검사하여 입력 형식이 올바르지 않으면 에러 코드를 출력합니다. 올바르게 입력되었으면 LOAD 멤버함수를 실행한다.

명령어가 "ADD"라면, 해당 명령어와 함께 전체 줄을 ADD 함수에 전달한다.

명령어가 "SEARCH_BP"라면, 탭을 기준으로 나뉜 데이터를 추출한다. 데이터가 없는 경우 에러 코드를 출력하고, 데이터가 있는 경우 첫 번째 데이터를 이용하여 SEARCH_BP_BOOK 함수를 호출하거나, 두 개의 데이터를 이용하여 SEARCH_BP_RANGE 함수를 호출한다.

명령어가 "PRINT_BP"라면, PRINT_BP 멤버함수를 호출하여 Bptree의 인자들을 오름차순으로 출력한다.

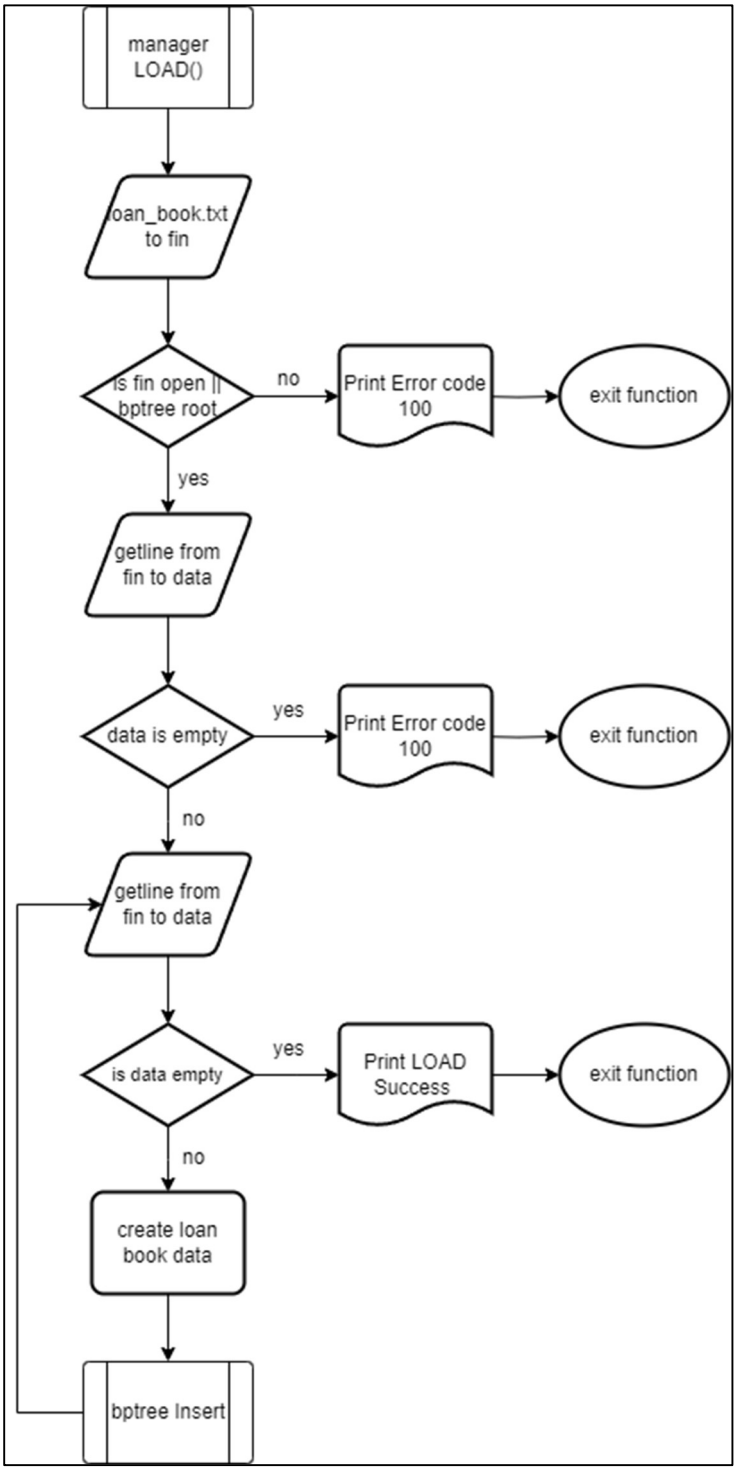
명령어가 "PRINT_ST"라면, 해당 명령어와 함께 전체 줄을 PRINT_ST 함수에 전달합니다. 파싱한 코드에 맞는 도서정보를 출력한다.

명령어가 "DELETE"라면, DELETE 함수를 호출하여 Selection Tree의 루트를 삭제하는 연산을 수행한다.

명령어가 "EXIT"라면, 성공 코드를 출력한 후, fcmd와 flog 파일을 닫고, bptree와 stree를 삭제한 뒤 프로그램을 종료한다.

그 외의 경우에는 에러 코드를 출력한다.

아무 것도 입력되지 않은 경우 반복문을 빠져나오며 fcmd와 flog 파일을 닫고 bptree와 stree를 삭제한 뒤 함수를 종료한다.



Manager.LOAD

LOAD 함수는 "loan_book.txt" 파일에서 대출 도서 정보를 읽어와 B+-트리에 저장하는 역할을 수행한다.

먼저, ifstream 객체인 fin을 선언하고 "loan_book.txt" 파일을 연다. 파일 열기에 실패하거나 이미 B+-트리에 데이터가 있는 경우에는 에러 코드 100을 출력하고 해당 함수를 빠져나간다.

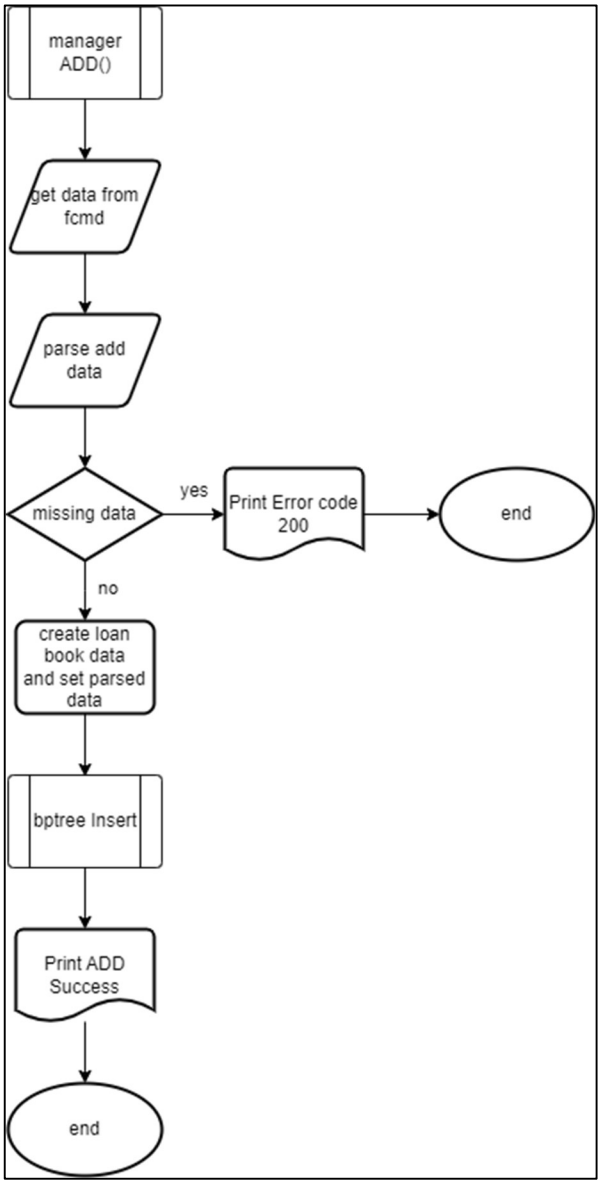
그렇지 않은 경우, 파일의 끝까지 반복하면서 각 줄을 읽어와서 데이터를 후처리한다. 만약 파일이 끝에 도달하거나 더 이상 읽어올 대출 도서 데이터가 없는 경우 반복문을 종료한다.

각 줄에서 탭 문자를 기준으로 데이터를 추출하여 도서명, 분류 코드, 저자, 출판연도, 대출 권수의 데이터를 파싱하여 BpTree Insert 함수에 인자로 전달하여저장한다.

모든 데이터를 처리한 후에는 "LOAD"에 대한 성공 코드를 출력한다.

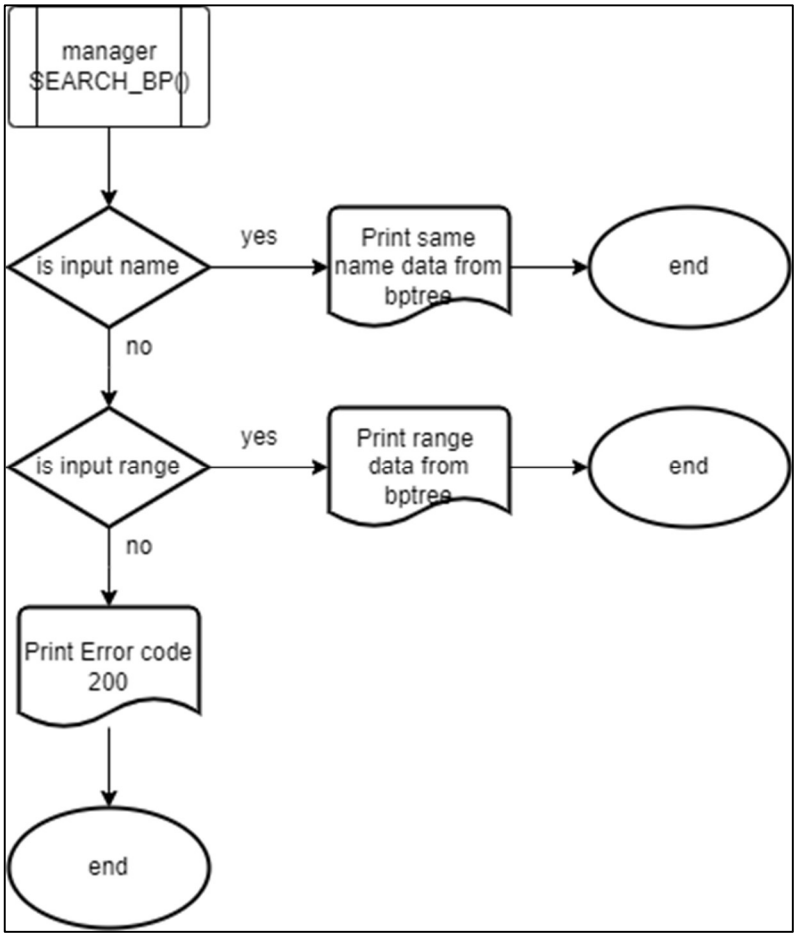
Manager.ADD

이 함수는 전달된 문자열에서 데이터 필드를 추출하여 새로운 도서 정보를 생성하고 B+-트리에 추가한다. index 배열을 선언하고 초기값을 -1로 설정하여 line 문자열에서 탭 문자(₩t)의 인덱스를 찾아 index 배열에 저장한다. 이때, 필드가 누락되었거나 잘못된 형식인 경우 해당하는 에러 코드 200을 출력하고 함수를 종료한다. 이후, 이러한 방식으로 미리 찾은 인덱스를 사용하여 name, code, author, year 변수에 적합한 값을 저장하여 LoanBookData 객체를 생성하고 추출한 데이터를 이용하여 도서 정보를 설정한다. 그리고 이를 B+-트리에 삽입한다. 성공적으로 삽입되면 로그 파일(flog)에 "ADD" 성공을 기록하고 해당 함수를 빠져나간다.



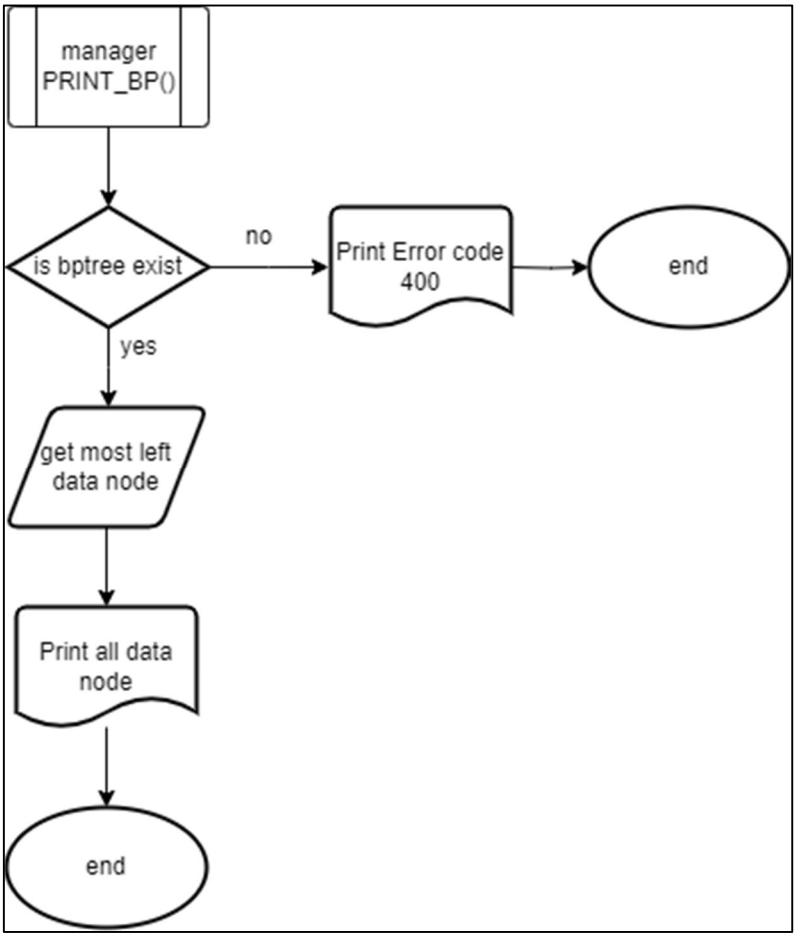
Manager.SEARCH_BP

해당 함수는 이름을 입력 받았냐, 알파벳 두개를 입력 받았냐에 따라 호출하는 함수가 달라진다. 이름을 입력 받은 경우 SEARCH_BP_BOOK함수를 호출하여 동일한 도서명을 가진 정보를 출력한다. 알파벳을 두개 입력받은 경우엔 도서명이 해당하는 두 알파벳의 범위 내에 있는 알파벳으로 시작하는 경우 모두 출력한다. 두 경우 모두 출력할 도서정보가 없으면 에러코드 300을 출력하고 함수를 빠져나간다.



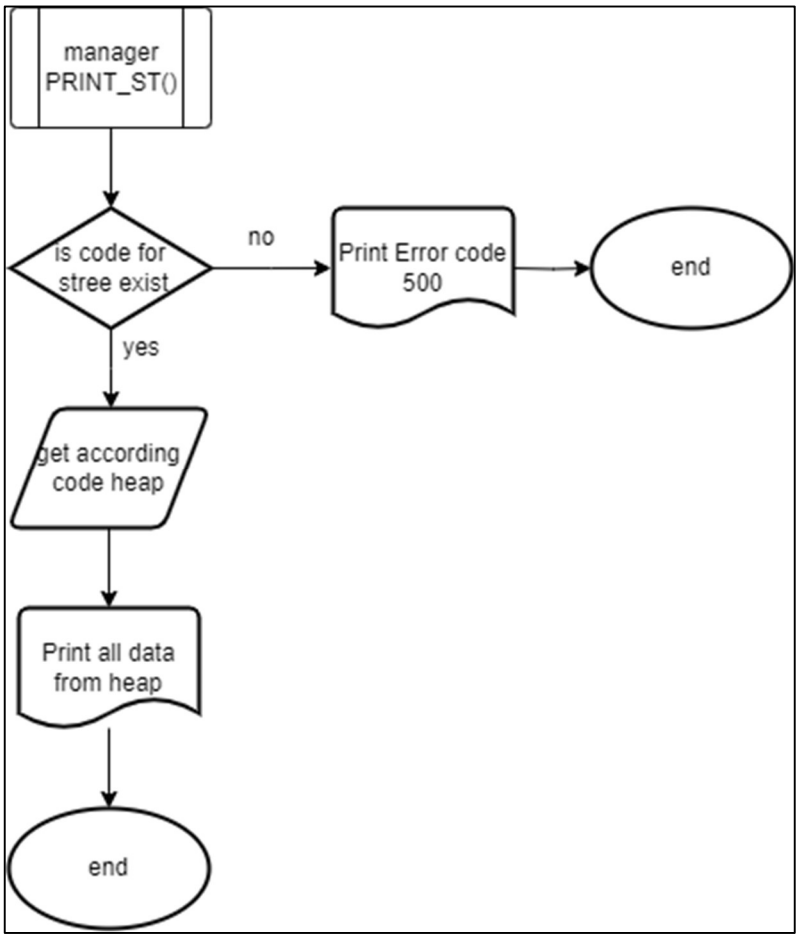
Manager.PRINT_BP

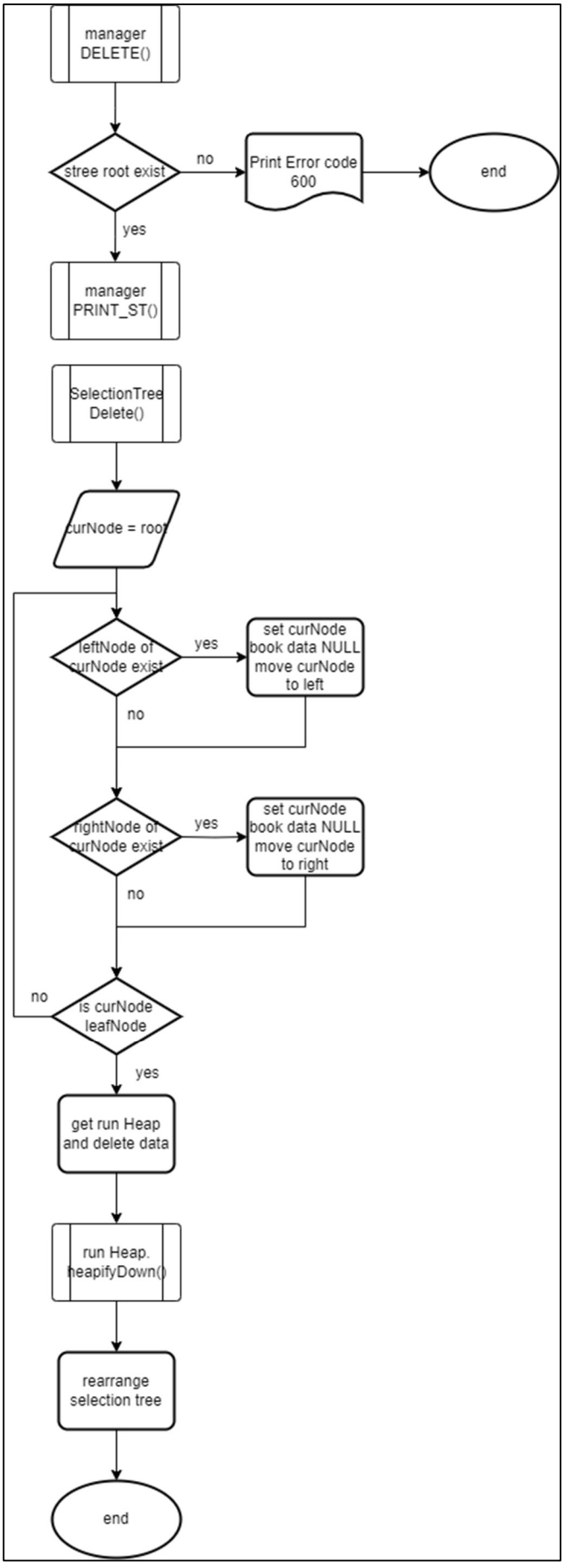
이 함수는 B+-트리에 저장된 도서 정보를 출력하는 역할을 수행한다. 먼저, B+-트리에 데이터가 저장되어 있는지 확인하고 없으면 에러코드 400을 출력한다. 데이터가 저장되어 있으면 데이터 노드를 찾기 위해 B+-트리의 루트 노드를 기준으로 반복해서 가장 왼쪽 자식 노드로 이동한다. 그렇게 얻은 data node를 순차적으로 접근하며 해당 노드에 저장된 맵에 있는 데이터를 하나씩 출력한다. 더 이상 출력한 데이터가 없으면 다음 노드로 이동하며 다음 노드가 없을 때까지 해당 작업을 반복한다.



Manager.PRINT_ST

PRINT_ST 함수는 분류 코드를 입력받아 해당 코드에 대한 도서 정보를 출력하는 기능을 한다. 주어진 문자열에서 탭 문자(₩t)의 인덱스를 찾아서 코드를 추출한다. 추출된 코드를 정수형으로 변환하여 selection tree에 저장된 해당 코드에 적합한 heap을 가져와 임시 저장한다. 이 때 추출된 코드를 검사하여 유효한 코드인지 확인하고 유효한 코드인 경우 해당하는 printBookData 함수를 호출하여 도서 정보들을 출력한다. 유효하지 않은 경우에는 에러 코드 500을 출력한다.





Manager.DELETE

마지막으로 DLETE 함수는 select tree에서 도서 정보를 삭제하는 역할을 수행한다. 먼저, select tree의 루트 노드에 도서 정보가 있는지 확인한다. 루트 노드의 getBookData 함수를 호출하여 도서 정보의 존재 여부를 확인한다. 도서 정보가 존재하는 경우, selection tree의 Delete 함수를 호출하여 도서 정보를 삭제한다. 그리고 성공 결과를 출력한다. 도서 정보가 존재하지 않는 경우 에러코드 600을 출력한다.

여기까지 기본적인 main함수와 프로그램 진행을 관리하는 manager 함수의 순서도이다. manager파일에서 각각의 Bptree, Selection Tree, Heap에 접근하여 데이터를 관리한다. 해당 알고리즘들의 동작은 다음 Algorithm 파트에서 다룰 예정이다.

3. Algorithm

B+-tree

- b+-tree는 데이터를 효율적으로 저장하고 검색하기 위한 트리 자료 구조이다. 주로 데이터베이스 인덱스에서 사용되며, 검색 속도를 높이고 데이터를 효율적으로 관리하는 데 도움을 준다.
- 노드 구성: 크게 index node와 data node로 나뉘어 구성되는데, 모든 data node들은 leaf node에 위치한다. 또한 각각의 data node는 앞뒤로 연결된 이중 연결 리스트의 형태를 갖는다. 그리고 index node들은 leaf node를 제외한 레벨에 존재하게 된다. 이때 해당 tree의 order에 따라 각각의 노드들이 수용할 수 있는 data의 개수가 정해지게 된다. 예를 들어 이번 프로젝트에서 쓰인 order가 3인 트리의 경우 각각의 노드가 가질 수 있는 data의 최대 개수는 2개이다. 3개가 되는 순간 해당 노드는 split되게 된다. Split이 실행되면 data node는 2개의 노드로 나뉘고, index node는 3개의 node로 나뉘게 된다. 기존 노드의 parent가 존재하면 split한 data를 합병하며 이와 같은 과정을 반복하며 tree를 구성한다.
- 특징: B+-tree는 complete binary tree 형태를 이루고 있다. 덕분에 데이터의 개수가 많아져도 tree의 level 자체는 log scale로 증가하여 낮은 level을 유지할 수 있다. tree 구조에서 level은 complexity와 관련하여 중요한 요소이므로 탐색 속도를 효율적으로 유지할 수 있다. 또한 data를 leaf node에만 저장하기에 순차접근이 용이하다는 장점이 있다. 이러한 특징 덕분에 삽입 및 삭제 연산이 효율적으로 이뤄질 수 있다.
- 주요연산: B+-tree의 주된 연산으로는 삽입, 검색, 삭제가 있다. 각각을 더 자세히 서술하면 다음과 같다.
  - 삽입(ADD, LOAD): 트리에 값을 추가하는데, 해당 값이 이미 존재하는지 먼저 확인한다. 이미 존재하는 값이면 해당 노드로 접근하여 대출권수를 증가시켜주고, 그렇지 않다면 새로운 data node를 만들어 기존 트리에 연결해주며 필요한 경우 해당하는 index node도 생성하여 tree에 삽입해준다.
  - 검색(SEARCH_BP): 특정 값을 찾기 위해 트리를 탐색한다. 인자로 책 제목이나 알파벳 두개를 입력 받을 수 있다. 두 경우 모두 루트노드에서 시작하여 현재 노드의 값과 비교하여 더 작은 쪽 또는 더 큰 쪽으로 이동하면서 검색을 진행한다. 결과적으로 입력 받은 값과 인접한 data를 가지는 data node를 반환 받는다. 이름을 입력 받은 경우 해당 노드에 찾는 책 제목이 있는지 확인하여 동일한 책제

목을 가진 정보를 출력하고, 알파벳 두 개를 입력 받은 경우 range 탐색으로 두 알파벳 사이에 존재하는 문자들로 시작하는 모든 정보들을 출력한다.

  - ■ 삭제(delete): 삭제의 경우 이번 프로젝트에서는 대출 권수를 초과한 책들만 tree 구조에서 삭제하여 selection tree와 heap으로 이동시킨다. Data node가 삭제되는 경우 크게 두가지 경우가 있는데, data가 data node의 맵의 두번째 요소일 경우와 첫번째인 경우로 나뉜다. 전자의 경우 데이터를 그냥 삭제해주면 되지만, 두번째 경우에는 또 underflow가 일어나지 않는 경우와 underflow가 발생하는 경우로 나뉜다. Underflow가 발생하면 트리 구조의 전반적인 수정이 필요하다.

- 해당 프로젝트에서는 대출도서를 저장한 Bptree에서 해당 알고리즘을 이용하였다.

Selection tree

- 일반적으로 포화 이진 트리 구조를 가진다. 리프 노드에 해당하는 run의 개수에 따라 level이 정해지는데, 예를 들어 run이 8개면 포화 이진트리 구조를 만족시키기 위해 level 3의 트리를 구성하게 된다.
- 선택트리에는 크게 두 종류가 있는데, 승자 트리와 패자 트리이다. 각각의 노드에 저장되는 것이 승자면 승자트리, 패자면 패자트리가 된다. 이러한 트리는 다시 한번 승자가 큰 값인지 작은 값인지에 따라 max와 min으로 나뉘게 된다. 승자 트리의경우 형제와 비교하여 부모노드로 전달되지만, 패자트리의 경우 부모와 비교한다.
- 이러한 선택트리는 효율적인 선택을 위해 사용된다. 이번 프로젝트에서는 대출불가 도서 중 가장 작은 값을 선택하여 삭제하는 연산을 수행하기 위해 선택트리가 사용되었다. 각각의 run은 일반적인 선택트리와 다르게 배열이 아니라 Heap으로 정보들을 저장한다. Min loser tree를 구성하기 위해서는 각각의 run도 오름차순으로 정렬되어야 하는데, 이러한 구조를 만족시키기 위해 run 또한 min heap으로 구현한 것을 알 수 있다.
- 해당 자료구조에서는 삭제 기능을 구현했는데, 패자 트리이므로 root node부터 시작해서 같은 값을 가지는 자식노드로 계속 연결해 들어가며 run에 해당하는 Loan Book Heap의 root를 삭제한다. 이후 해당 힙을 재정렬하여 새로운 run으로 선택트리도 재정렬한다.

Heap

- 데이터 구조 중 하나로 완전 이진트리 구조를 가진다. 일반적으로 우선순위 큐를 구현하는데 사용되는데, 그 종류는 부모 노드가 자식노드보다 큰 최대 힙과 부모 노드가 더 작은 최소 힙이 있다.
- 일반적으로 배열을 사용하여 구현하는데, 부모 노드와 자식 노드 간의 관계를 유지하

면서 배열에 요소를 저장한다. 부모 노드와 자식 노드의 관계를 대략적으로 서술하면, 왼쪽 자식의 인덱스는 부모 인덱스의 두배, 오른쪽 자식의 인덱스는 부모 인덱스의 두배에 1을 더한 값, 부모 인덱스는 자식 인덱스를 2로 나눈 값이 된다. 일반적인 c에서 나머지는 버리므로 위와 같은 관계가 성립한다.

- 반정렬 상태를 유지하는데, 이는 큰 값이 상위 레벨에 있고 작은 값이 하위 레벨에 있다는 의미이다. 하지만 이는 부모와 자식간의 관계이므로 일반적인 중위 순회 등을 통해 오름차순이나 내림차순으로 출력할 수 없다.
- 또 다른 특징으로 힙 트리는 BST와 다르게 중복된 값을 허용한다.
- 대신 여러 개의 값들 중에서 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조로서 항상 루트 노드에는 가장 작거나 큰 값을 유지하게 된다. 따라서 힙에서는 삽입과 삭제가 중요하게 다뤄지는데, 해당 작업이 수행될 때마다 재정렬이 동반되기 때문이다.
- 주요 연산: 큐의 주요 연산으로는 insert와 delete가 있다.
  - insert: 힙에 새로운 요소를 추가하는 동작으로, 일반적으로 힙의 마지막 부분에 새로운 노드가 추가되게 된다. Min heap인지 Max heap인지에 따라 다르지만 공통적으로 새로운 노드가 삽입되면 재정렬 과정인 heapify가 수행된다. 삽입의 경우 마지막 노드를 기준으로 부모노드와 비교하며 교환을 진행한다. 그렇게 루트 노드까지 재정렬을 수행하면 최종적으로 루트에 가장 크거나 작은 값이 위치하게 된다.
  - delete: 힙에서의 삭제의 경우엔 항상 루트 노드를 삭제하게 된다. 가장 크거나 작은 값을 삭제하게 되고, 가장 마지막 리프노드에 위치한 데이터를 루트노드로 이동시킨다. 그리고 나서 다시 heap 데이터 구조를 유지하도록 heapify 작업을 수행한다. Insert 동작과 마찬가지로 그 결과 항상 가장 크거나 작은 값이 root에 위치하게 된다.
- 해당 프로젝트에서는 선택 트리의 각각의 run에 heap 자료구조가 위치하게 된다. Min loser tree를 구성하는 이번 프로젝트의 특성상 각각의 run의 자료들도 오름차순으로 정렬되어야 하기 때문이다. 각각의 heap의 root값을 선택트리의 리프노드로 전달하여 패자트리를 구성하고, DELETE 명령어가 입력되면 선택 트리의 루트부터 시작해 해당하는 run에 접근해 해당 데이터를 삭제하는 작업을 수행한다. 이후 heap을 재정렬해 새로운 루트값을 입력 받아 선택 트리도 재정렬하게 된다.

4. Result Screen

LOAD

loan_book.txt

```
the odyssey 600 Bram Stoker 1908    1
great expectations  400 George Orwell   1915    1
to kill a mockingbird   700 Harper Lee  1913    1
madame Bovary   100 Victor Hugo 1947    1
don Quixote 200 Miguel de Cervantes 1971    1
jane Eyre   600 Anthony Burgess 1924    1
pride and Prejudice 700 Charlotte Brontë    1998    1
slaughterhouse-Five 100 J.R.R. Tolkien  2006    1
wuthering Heights   400 Bram Stoker 1997    1
emma    300 George Orwell   1977    1
heart of Darkness   200 Anthony Burgess 1976    1
brave New World 100 Joseph Conrad   1995    1
dracula 200 Charles Dickens 1915    1
little Women    300 Oscar Wilde 2005    1
lolita  500 George Orwell   1922    1
sense and Sensibility   500 Anthony Burgess 1959    1
war and Peace   600 Miguel de Cervantes 1953    1
the Picture of Dorian Gray  500 Charles Dickens 1940    1
the Adventures of Huckleberry Finn  700 J.R.R. Tolkien  2017    1
the Great Gatsby    300 J.R.R. Tolkien  1931    1
frankenstein    600 Charlotte Brontë    2010    1
moby Dick   000 Herman Melville 1851    1
=========LOAD=========
Success
======================
```

정상적으로 loan_book.txt에 데이터가 존재하는 경우

```
=========LOAD=========
Success
======================

======================
ERROR 100
======================
```

자료구조에 이미 데이터가 들어가 있는 경우

```
======================
ERROR 100
======================
```

텍스트 파일이 존재하지 않은 경우

loan_book.txt파일을 열어 해당 파일에 있는 데이터들을 후처리하여 bptree에 삽입하는 명령어이다. 해당 파일이 존재하지 않거나 이미 데이터가 존재하면 에러코드를 출력한다. 이때 ADD 명령어로 데이터가 추가된 경우에도 고려하여 프로그램을 설계하였다.

ADD

```
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD madame Bovary   100 Victor Hugo 1947
ADD madame Bovary   100 Victor Hugo 1947
```

```
=========ADD=========
great expectations/400/George Orwell/1915
=======================

=========ADD=========
great expectations/400/George Orwell/1915
=======================

=========ADD=========
great expectations/400/George Orwell/1915
=======================

=========ADD=========
madame Bovary/100/Victor Hugo/1947
=======================

=========ADD=========
madame Bovary/100/Victor Hugo/1947
=======================
```

일반적인 ADD 명령어의 경우

```
ADD great expectations  400 George Orwell
ADD great expectations  400     1915
ADD great expectations      George Orwell   1915
ADD     100 Victor Hugo 1947
ADD madame Bovary   100 Victor Hugo 1947
```

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

=======================
ERROR 200
=======================

=======================
ERROR 200
=======================

=======================
ERROR 200
=======================

=========ADD=========
madame Bovary/100/Victor Hugo/1947
=======================
```

잘못된 인자 전달 (발행연도, 저자, 분류코드, 제목이 각각 없는 경우)

ADD 함수는 command 파일에서 데이터를 하나씩 추가하는 함수이다. 발행연도, 저자, 분류코드, 제목의 4가지 인자를 추가로 입력 받아 데이터를 후처리한 후 기존 데이터베이스에 추가한다. 이러한 인자 중 하나라도 없을 경우 에러 코드를 출력한다. 이번 프로젝트에서 ADD 함수는 매우 중요한 역할을 취하는데, 그 이유는 bptree에서 selectiontree로 데이터가 넘어가는 경우는 ADD 명령어가 들어오는 경우에만 가능하기 때문이다. ADD로 새로운 책이 입력되면 bptree에 새로운 노드로서 입력해주고, 기존에 존재하던 책 제목이면 대출 권수를 늘려준다. 이때 각각의 분류코드에 해당하는 최대 대출권수에 도달하면 bptree에서 제거한 후 selection tree와 loan book heap으로 전달한다.

SEARCH_BP

```
SEARCH_BP   a   c
SEARCH_BP   b   e
SEARCH_BP   a   y
SEARCH_BP   o   p
```

기본적인 범위 입력

```
=========SEARCH_BP=========
brave new world/100/Joseph Conrad/1995/1
==========================

=========SEARCH_BP=========
brave new world/100/Joseph Conrad/1995/1
don quixote/200/Miguel de Cervantes/1971/1
dracula/200/Charles Dickens/1915/1
emma/300/George Orwell/1977/1
==========================

=========SEARCH_BP=========
brave new world/100/Joseph Conrad/1995/1
don quixote/200/Miguel de Cervantes/1971/1
dracula/200/Charles Dickens/1915/1
emma/300/George Orwell/1977/1
frankenstein/600/Charlotte Bront/2010/1
great expectations/400/George Orwell/1915/1
heart of darkness/200/Anthony Burgess/1976/1
jane eyre/600/Anthony Burgess/1924/1
little women/300/Oscar Wilde/2005/1
lolita/500/George Orwell/1922/1
madame bovary/100/Victor Hugo/1947/1
moby dick/000/Herman Melville/1851/1
pride and prejudice/700/Charlotte Bront?/1998/1
sense and sensibility/500/Anthony Burgess/1959/1
slaughterhouse-five/100/J.R.R. Tolkien/2006/1
the adventures of huckleberry Finn/700/J.R.R. Tolkien/2017/1
the great gatsby/300/J.R.R. Tolkien/1931/1
the odyssey/600/Bram Stoker/1908/1
the picture of dorian gray/500/Charles Dickens/1940/1
to kill a mockingbird/700/Harper Lee/1913/1
war and peace/600/Miguel de Cervantes/1953/1
wuthering heights/400/Bram Stoker/1997/1
==========================

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

입력에 대한 출력

```
SEARCH_BP    emma
SEARCH_BP    emm
```

기본적인 이름 이름

```
=========SEARCH_BP=========
emma/300/George Orwell/1977/1
==========================

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

입력에 대한 출력

```
SEARCH_BP    a    c    d
SEARCH_BP
```

잘못된 입력

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

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

잘못된 입력에 대한 출력

Search bp 함수는 b+-tree에 저장된 데이터를 검색하여 출력하는 함수이다. 인자가 한 개 한 경우와 두개인 경우로 나뉘는데, 인자가 한 개 입력된 경우엔 책 제목으로 인식하여 동일한 제목을 가진 책의 데이터를 출력한다. 같은 책 제목이 없으면 오류코드 300을 출력한다. 또 다른 경우는 두개의 알파벳을 입력 받는 경우이다. 두 알파벳을 범위로 지정하여 해당 범위 내의 알파벳으로 시작하는 책 제목을 가지는 모든 데이터를 출력한다. 해당 범위에 데이터가 없는 경우 마찬가지로 오류코드 300을 출력한다. 마지막으로 1개나 2개가 아닌 경우에도 오류코드 300을 출력한다.

PRINT_BP

```
=========PRINT_BP=========
brave new world/100/Joseph Conrad/1995/1
don quixote/200/Miguel de Cervantes/1971/1
dracula/200/Charles Dickens/1915/1
emma/300/George Orwell/1977/1
frankenstein/600/Charlotte Bront/2010/1
great expectations/400/George Orwell/1915/1
heart of darkness/200/Anthony Burgess/1976/1
jane eyre/600/Anthony Burgess/1924/1
little women/300/Oscar Wilde/2005/1
lolita/500/George Orwell/1922/1
madame bovary/100/Victor Hugo/1947/1
moby dick/000/Herman Melville/1851/1
pride and prejudice/700/Charlotte Bront?/1998/1
sense and sensibility/500/Anthony Burgess/1959/1
slaughterhouse-five/100/J.R.R. Tolkien/2006/1
the adventures of huckleberry Finn/700/J.R.R. Tolkien/2017/1
the great gatsby/300/J.R.R. Tolkien/1931/1
the odyssey/600/Bram Stoker/1908/1
the picture of dorian gray/500/Charles Dickens/1940/1
to kill a mockingbird/700/Harper Lee/1913/1
war and peace/600/Miguel de Cervantes/1953/1
wuthering heights/400/Bram Stoker/1997/1
==========================
```

앞선 LOAD 이후 PRINT_BP 출력

```
==========================
ERROR 400
==========================
```

데이터 없는 경우

해당 함수는 bptree에 저장된 데이터를 도서명 기준 오름차순으로 모두 출력한다. 만약 저장되어 있는 값이 없는 경우엔 에러 코드 400을 출력한다. Bptree의 특징 상 첫번째 데이터 노드에 접근하여 순차적으로 출력하기만 하면 되는 간단한 명령어이다.

PRINT_ST

```
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD madame bovary   100 Victor Hugo 1947
ADD madame bovary   100 Victor Hugo 1947
```

```
=========PRINT_ST=========
great expectations/400/George Orwell/1915/4
==========================

=========PRINT_ST=========
madame bovary/100/Victor Hugo/1947/3
==========================
```

기본적인 입력

```
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD great expectations  400 George Orwell   1915
ADD madame bovary   100 Victor Hugo 1947
ADD madame bovary   100 Victor Hugo 1947
PRINT_ST    400
PRINT_ST    100
DELETE
ADD moby dick   000 Herman Melville 1851
ADD moby dick   000 Herman Melville 1851
ADD brave new world 100 Joseph Conrad   1995
ADD brave new world 100 Joseph Conrad   1995
ADD heart of darkness   200 Anthony Burgess 1976
ADD heart of darkness   200 Anthony Burgess 1976
ADD little women    300 Oscar Wilde 2005
ADD little women    300 Oscar Wilde 2005
ADD little women    300 Oscar Wilde 2005
ADD lolita  500 George Orwell   1922
ADD frankenstein    600 Charlotte Bront?    2010
ADD to kill a mockingbird   700 Harper Lee  1913
PRINT_ST    000
PRINT_ST    100
PRINT_ST    200
PRINT_ST    300
PRINT_ST    400
PRINT_ST    500
PRINT_ST    600
PRINT_ST    700
```

```
=========PRINT_ST=========
moby dick/000/Herman Melville/1851/3
==========================

=========PRINT_ST=========
brave new world/100/Joseph Conrad/1995/3
madame bovary/100/Victor Hugo/1947/3
==========================

=========PRINT_ST=========
heart of darkness/200/Anthony Burgess/1976/3
==========================

=========PRINT_ST=========
little women/300/Oscar Wilde/2005/4
==========================

==========================
ERROR 500
==========================

=========PRINT_ST=========
lolita/500/George Orwell/1922/2
==========================

=========PRINT_ST=========
frankenstein/600/Charlotte Bront/2010/2
==========================

=========PRINT_ST=========
to kill a mockingbird/700/Harper Lee/1913/2
==========================
```

모든 selection tree에 데이터가 있는 경우

```
PRINT_ST    700
PRINT_ST    70
```

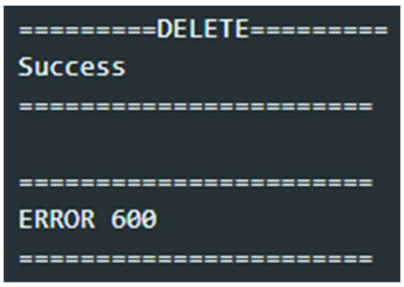
```
=========PRINT_ST=========
to kill a mockingbird/700/Harper Lee/1913/2
==========================

==========================
ERROR 500
==========================
```

잘못된 분류코드를 입력하는 경우

해당 명령어는 Selection tree의 분류코드를 인자로 받아, 해당 분류코드를 가지는 loan book heap에 접근하여 해당 도서들을 출력한다. 출력은 도서명을 기준으로 오름차순으로 출력하고, selection tree에 저장된 데이터가 없는 경우나 분류 코드에 해당하는 heap이 존재하지 않는 경우 에러코드 500을 출력한다. Heap의 자료구조 특성상 일반적인 순회로는 오름차순 출력이 안되기 때문에 map STL을 이용하여 도서명을 key로 설정하여 모든 데이터들을 입력해주고 순차적으로 출력하는 방식으로 구현하였다.

DELETE



Selection tree에 삭제할 노드가 있는 경우와 없는 경우

DELETE함수는 Selection tree의 루트 노드에 해당하는 값을 패자트리 알고리즘을 따라 내려가서 해당하는 run에 있는 heap node를 삭제하는 명령을 수행한다. 해당 힙을 재정렬하고 루트 노드를 다시 selection tree의 리프노드에 삽입하면서 selection tree도 재정렬시킨다.

5. Consideration

- B+-tree의 구현과정에서 가장 큰 고충을 겪었다. 처음 접해보는 데이터구조여서 어떤 식으로 구현해야 할지 고민을 많이 한 것 같다. 특히 주어진 스켈레톤 코드를 제대로 이해하지 못한 상태로 시작하였다가 구조를 완전히 잘못 짜서 갈아 엎기도 했었다. 실습시간에 알려주신 내용을 기반으로 split하는 로직을 이용해 Bptree를 완성할 수 있었다. 상속자에 대한 개념을 이론수업만으로 확실히 이해하지 못한 상태였는데, 이번 프로젝트를 계기로 적용해보며 확실히 이해할 수 있었다.

  추가적으로 Bptree의 delete의 경우 underflow가 발생했을 때 노드들을 재설정하는 기능은 시간 관계상 포기할 수밖에 없었다. 구현방법에 대한 이해는 되었는데, 직접 적용해보며 익힐 시간이 없어 아쉬움이 남는다.
- 마지막에 log를 연결해서 출력하는 기능을 구현하는데 고민을 했었다. 그 결과 ios::app을 log.txt 파일을 열 때 두번째 인자로서 넣어주면 연결해서 데이터가 출력된다는 사실을 알게 되었다. 파일 입출력에 관해서는 아직 미숙한 부분이 많다는 것을 새삼 깨닫게 되었고, 버퍼 관리에 있어서도 개념을 확실히 잡으면 좋겠다는 생각이 들었다.
- PRINT_ST에서 heap에 데이터를 출력하는 경우 heap이라는 데이터의 특성에 대해 고찰을 많이 했다. 오름차순으로 출력을 할 경우 max heap으로 재구성한 후 root부터 빼내어 출력하고 재정렬하는 힙정렬 방식을 생각해냈었다. 이를 위해 임시로 힙 객체를 동적할당해 모두 복사한 후 다시 출력하는 과정을 거쳤는데, 이는 비효율적이라는 생각이 들었다. 그래서 깃허브 issue에 나온 것처럼 map STL을 이용하여 자료들을 삽입한 후 다시 출력하며 해당 기능을 구현하였다.
- 이전 프로젝트에서 Window의 visual studio로 진행하다 Linux로 옮기자 컴파일러 차이로 다른 결과가 나와 고생을 많이 했었다. 이번 프로젝트는 그러한 경험을 토대로 처음부터 리눅스 운영체제로 vscode를 열어 코딩하였다.
- 구현을 모두 마치고 소멸자를 호출하는 과정에서 문제를 발견하였다. 시간을 할애해 계속해서 생각해보았는데, 프로그램 구현 과정에서 소멸자와 맞지 않는 부분이 있음을 파악하였다. 이를 고치기 위해서 소멸자를 고치는 방식으로 접근하였지만, 결국 해당 함수들을 고쳐야된다는 결과가 도출되었다. 고치는 과정은 두렵지 않았지만, 해당 오류를 찾아낸 시점이 너무 늦었기에 1차 프로젝트처럼 늦게 제출하지 않도록 일부 메모리 누수는 안고 가기로 하였다...