Data Structures_Porject_02

학　　과: 컴퓨터공학과

담당교수: 최상호 교수님

학　　번: 2020202037

성　　명: 염정호

1. Introduction

1)목적

본 프로젝트는 B+ tree, Heap, Selection tree) 자료구조를 이용한. 도서관이 프로그램을 구현해 보고 해당 프로그램을 구현해하고 각각의 자료구조에서 어떠한 방식으로 데이터가 삽입, 정렬, 삭제, 출력 되는지 확인해본다.

2)세부사항

- 도서명, 도서분류코드, 저자, 발행연도, 대출 권수를 관리하며, 이를 이용해 대출중인 도서와 대출 불가 도서에 대한 정보를 제공할 수 있다.

- loan_data.txt파일을 읽어 책 정보를 불러온다. 각 도서에 대한 정보는 /t으로 구분되며 도서명은 항상 고유하고 소문자로 이루어져 있다.

-분류코드의 경우 000~600번대 까지 존재하며 각 분류코드에 따른 대출 권수는 다음을 따른다.

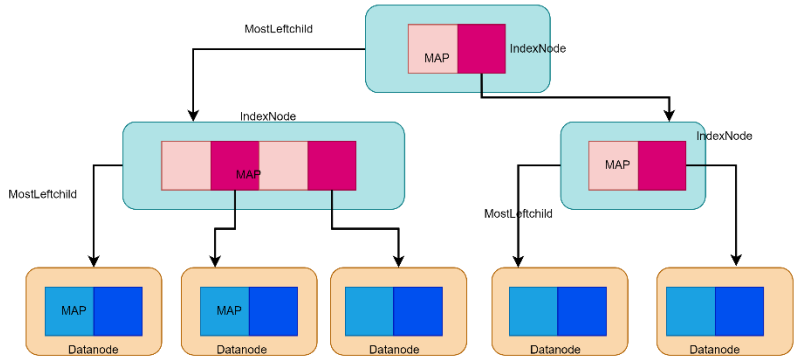
분류코드	대출 가능 권수
000/100/200/	3
300/400	4
500/600/700	2

1) B+TREE

-B+ tree의 차수는 3으로 정한다.

-ADD명령어를 이용해 대출 권수를 추가하거나 직접적으로 데이터를 넣을 수 있으며 코드 별 대출 권수를 초과한 경우 즉 대출 가능 도서가 없는 경우에는 selection트리로 데이터를 전달하고 b+ tree에서 제거한다.

-B+tree에 저장되는 데이터의 경우 map형태로 도서명을 key로 가지고 있으면 책에 관한 데이터의 경우 lonabookdata class에 저장되어있다.



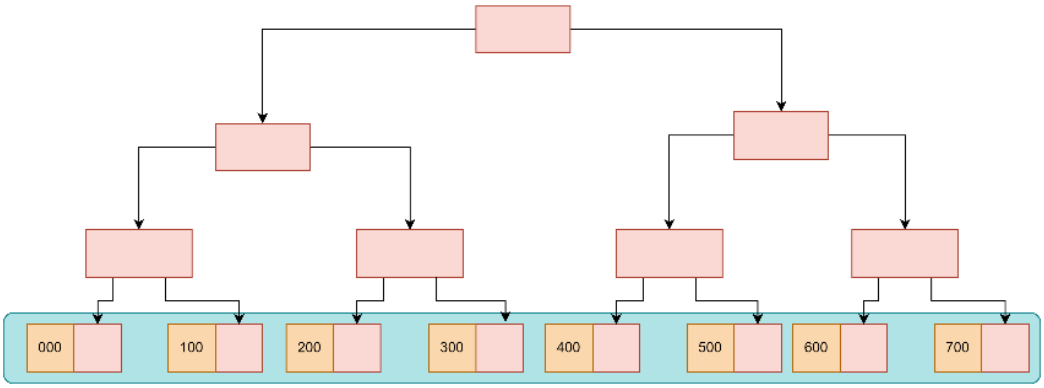
위와 같이 각 노드 안에 맵 형태의 데이터가 포함되어 key값을 기준으로 정렬되며 가장 아래에는 데이터 노드들이 위치하게 된다.

2) Seletion tree

-도서명을 기준으로 min winner tree의 형태를 가지고 있으며 대소문자 구분이 없다.

-ADD명령어를 통해 B+tree에서 삭제된 데이터가 들어오게 되며 각 데이터는 heap의 형태로 seletion트리의 run에 저장되어 있다. 여기서 run의 개수는 분류코드의 개수와 같다.

-seletion내부에 저장되는 데이터 또한 loanbookdata형태로 저장되어진다.



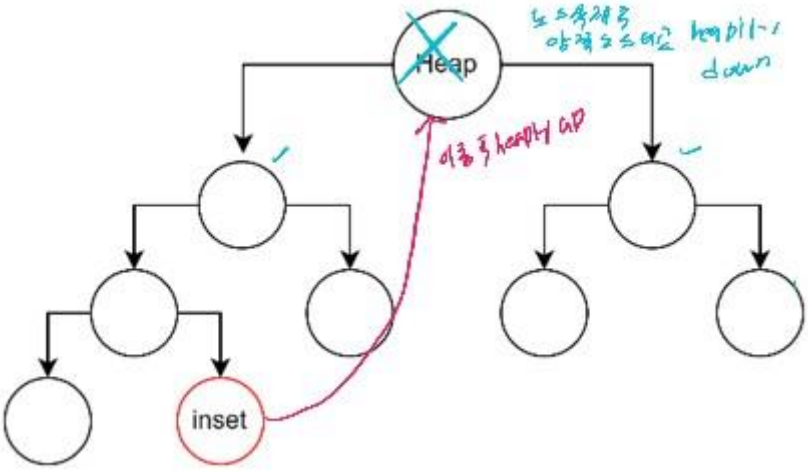
셀렉션 트리의 경우 처음 시작부터 16개의 노드를 가지고 생성되며 마지막 8개 노드는 맵 형태로 구현 되어 분류코드 8가지를 키로 지니게 된다.

양쪽 힙의 루트 값이 셀렉션 트리 마지막 단 데이터가 되며 부모 노드는 양쪽의 값을 비교하여 더 작은 쪽을 선택하게 되어 가장 작은 값이 맨 위로 올라가게 되는 것이다.

3) LoanBookHeap

-도서명을 기준으로 min heap으로 구성되며 B+tree로부터 대출 불가도서 데이터를 받아와 구축된다.

- 데이터가 새로 들어올 때 마다 재정렬 되며 힙이 재정렬 되는 경우 seletiontree또한 재정렬 된다.



힙의 경우 완전 이진트리를 유지해야 하기 때문에 구조 내에서 가장 마지막 단에 데이터가 삽입되며 해당 데이터를 부모와 비교해 조건에 따라 부모와 교체되면서 위치를 찾아가고 루트의 삭제의 경우 가장 마지막에 삽입된 데이터가 루트로 이동해 값을 타고 내려오며 정렬하게 된다.

각 헤더파일과 명령어에 대한 설명은 다음 표의 내용과 같다.

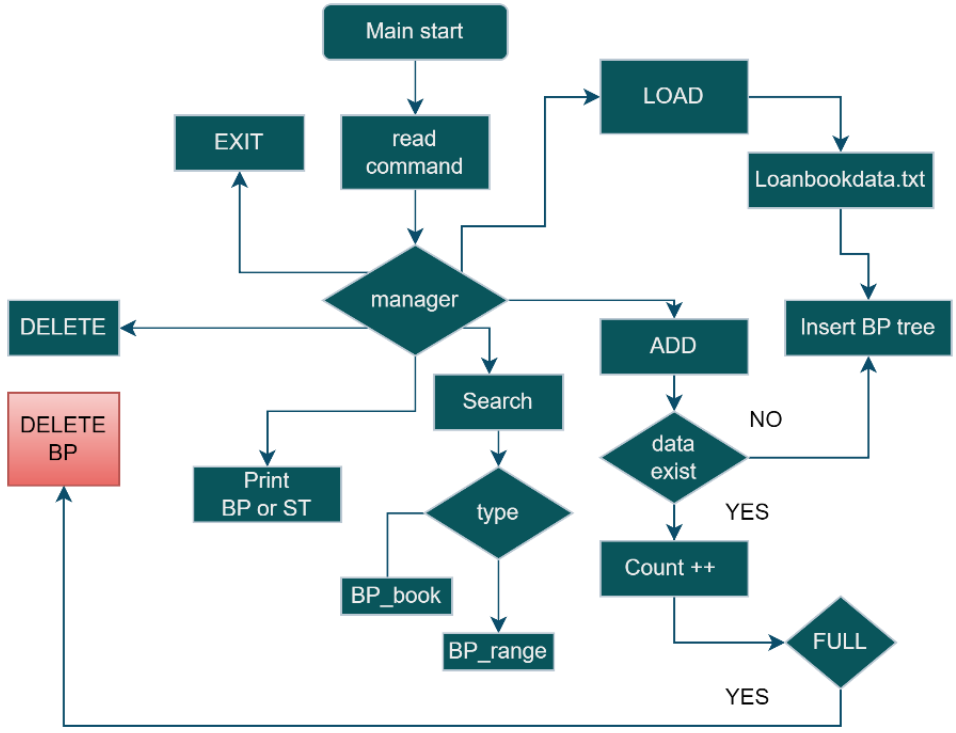
헤더파일	명령어	기능
Manager	생성자	Bp tree와 Seletion tree 생성
	run	Command.txt에 저장된 명령어를 읽어오고 실행시킨다. 인자가 존재하는 명령어의 경우 인자와 명령어를 따로 구분해 저장한 뒤 실행시켜 준다.
	LOAD	Loanbookdata.txt로부터 데이터를 읽어와 BP tree에 저장하는 명령어로 이미 데이터가 존재하거나 파일이 존재하지 않을 경우 에러코드를 출력한다.
	ADD	데이터를 직접 추가해주는 명령어로 이미 존재하는 데이터의 경우 대출 권수를 증가시켜 주며 존재하지 않는 데이터의 경우 해당 도서를 추가한다.
	SEARCH_BP _BOOK	이름을 인자로 도서를 검색하는 명령어로 해당 도서를 btree에서 탐색한다.
	SEARCH_BP _RANGE	인자를 2개 받아 비피트리의 데이터를 검색하는 명령어로 a b와 같이 입력 받을 시 해당 알파벳 사이에 존재하는 데이터를

		모두 출력한다.
	PRINT_BP	bp트리 내의 모든 데이터를 출력한다.
	PRINT_ST	코드를 입력받아 해당 코드에 해당하는 도서를 셀렉션 트리에서 출력한다.
	DELETE	셀렉션 트리의 루트 노드를 제거하는 명령어로 루트 제거시 힙에서도 같이 제거되며 두 저장공간 모두 재정렬 시켜준다
	BP_DELETE	BPTREE에서 대출불가 도서를 삭제하고 데이터를SELETIONTREE로 넘겨주는 명령어이다
	printErrorCode	명령어별 에러 발생시 에러코드 출력
	PrintSucessCode	성공문구 출력
BpTree	Insert	데이터를 비피트리에 추가하는 명령어로 노드의 데이터가 차수 이상일 경우 해당 노드를 분할한다.
	excessDataNode	데이터 노드가 차수를 넘어선지 확인
	excessIndexNode	인덱스 노드가 차수를 넘어선지 확인
	splitDataNode	데이터노드가 정해진 차수 이상의 데이터를 가진 경우 해당 데이터 노드를 분할 이때 인덱스 노드가 생기게 되는데 해당 인덱스 노드의 차수 또한 확인해야 한다.
	splitIndexNode	인덱스 노드를 분할하는 함수로 해당 인덱스를 분할 후 상위 인덱스를 생성하거나 분할 되는데 이 때 생성된 인덱스가 차수를 넘을 경우 함수를 다시 호출해 분할한다.
	getRoot	루트 노드 데이터를 불러온다.
	SearchDataNode	데이터 노드를 탐색하는 함수로 도서명을 인자로 받아 해당 도서가 존재해야 하는 데이터 노드로 이동한다. 데이터 노드로 이동하는 것이기 때문에 실제 데이터가 존재하는지 확인을 위해 맵을 이용해야 한다.
	print	비피 트리의 데이터를 모두 출력하기 위해 사용되는 명령어 해당 트리의 가장 왼쪽 노드로 이동하여 맵과 노드를 이동하며 가장 오른쪽 끝 노드 까지 출력
	searchBook	SearchDataNode명령어를 통해 데이터가 존재해야 하는 위치를 찾고 해당 위치에 데이터가 존재하는지 확인하는 명령어
	searchRange	범위를 입력받은 뒤 해당 시작 범위부터 끝 범위까지 탐색하며 데이터 출력
	deletenode	add에서 대출가능 권수를 초과한 데이터들을 관리할 때 사용하는 명령어이다. 각 명령어가 어떤 상황에서 쓰이는지는 아래에서 설명한다.
	del_data_next_size2	
	del_data_pre_size2	
	change_indexdata	
	underflow_happend	
	merge_front_index	
	merge_back_index	
Seletion Tree	생성자	8개의 분류코드를 가진 셀렉션 트리 생성
	SetRoot	루트노드 설정
	Insert	셀렉션 트리에 데이터를 추가하는 명령어로 해당 셀렉션 트리에 힙 데이터가 존재하지 않는다면 새로운 힙 데이터를 만들고 이미 존재하는 경우 셀렉션 트리와 힙 모두의 데이터를 갱신한다.
	Delete	셀렉션 트리의 루트 노드에 있는 데이터를 삭제하는 명령어로 셀렉션 트리에서 삭제된 데이터는 힙에서도 삭제되며 두 데이터 모두 재정렬 된다.
	printBookData	입력받은 코드에 대한 도서정보를 heap에서 정렬된 순서대로 출력한다.
LoanBookHeap	setRoot	루트 노드를 설정한다
	getRoot	루트 노드를 반환한다.
	heapifyUp	데이터 입력 후 데이터 정렬
	heapifyDown	데이터 출력 후 데이터 정렬
	Insert	힙에 데이터 추가
	div	입력 할 노드 위치를 찾는 함수
	countdown	힙의 개수 감소
	getCount	힙 데이터 개수 반환
	getLast	힙 위치 탐색에 사용되는 배열에서 마지막 주소 반환

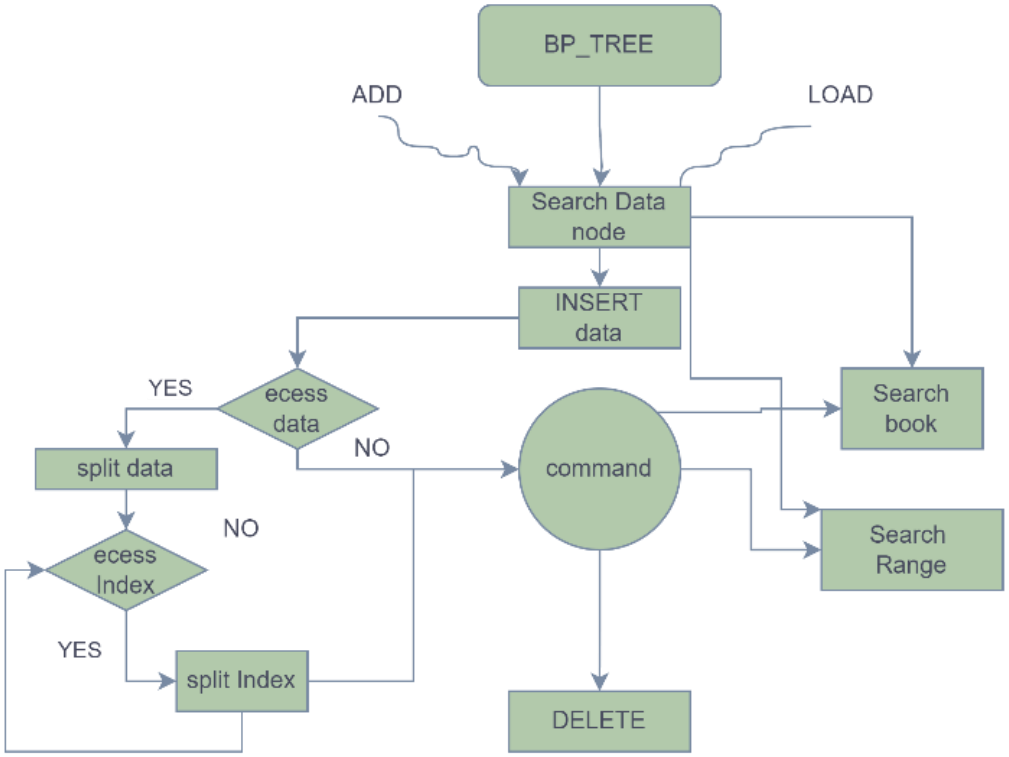
여기서는 각 데이터 별 간단한 기능만 설명하고 각 함수의 동작 과정의 경우 알고리즘에서 설명한다.

2. Flowchart

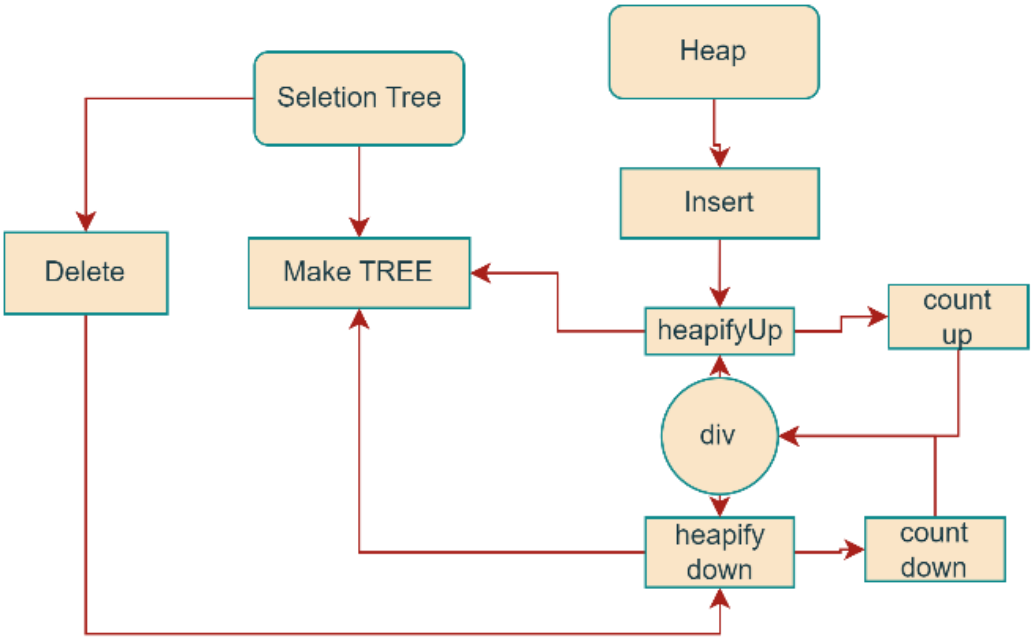
Mannager



BPTREE



Seletion Tree & Heap



Div = 데이터가 삽입될 or 마지막으로 삽입된 위치를 count를 통해 구한다.

3. Algorithm

Main{

차수(3) 지정 후 매니저 파일 생성

Command.txt를 불러와 명령어를 읽고 실행

}

Manager{

Run:

Log.txt파일 생성

Command.txt파일을 읽어와 명령어를 구분하고

해당 명령어 실행, 오류 발생시 에러코드 출력

Load:

이름, 분류코드, 저자, 연도, 대출권수로 구분된 데이터를 읽어와

비피트리 생성/ 데이터 추가

ADD:

데이터 존재시 대출 권수 추가,

대출 불가능 도서로 변할 시 데이터 삭제

데이터가 존재하지 않을 시 데이터 생성

}

BPTree{

Insert:

Root에 데이터가 존재하지 않으면 root에 데이터 삽입

그 이외의 경우 searchDataNode를 통해 삽입될 데이터 노드 탐색 후 삽입.

데이터 노드 분할이 필요한 경우 분할 실행,

➔ 인덱스 노드 생성/필요시 재귀적으로 분할 진행

splitDataNode:

분할받을 데이터 노드 생성

현재 데이터에서 중간에 있는 노드를 기준으로 2개의 노드를 분할 받는다.

분할 뒤 현재 노드에서 분할해준 노드 데이터 삭제

인덱스 존재하지 않을시 생성

이미 상위 인덱스가 존재할 시 해당 인덱스에 인덱스 데이터 추가

/ 필요시 분할

splitIndexNode:

새로운 인덱스 데이터 1개 생성

분할할 인덱스 노드의 마지막 부분을 분할받는다.

상위 노드가 존재할시 중간 노드를 상위 노드로 이동,

존재하지 않을 시 새로운 노드를 생성해 값 전달-

➔ 해당 노드를 root로 변경

searchDataNode

MostLeftChild가 널값일 경우 해당 노드는 데이터 노드가 된다.

Key를 기준으로 데이터 노드가 나올 때 까지 탐색

인덱스 노드의 데이터가 작을 때

데이터가 하나인 경우 MostLeftChild로 이동

데이터가 2개 인 경우

->첫 데이터 ->MostLeftChild

2번째 데이터 -> 이전 데이터의 맵 데이터로 이동

인덱스 노드 데이터가 더 클 때

데이터가 하나인 경우

맵 데이터가 가리키는 곳으로 이동

데이터가 두개인 경우

첫 데이터 -> 다음 데이터로 이동

두번째 데이터, 해당 데이터의 맵 주소로 이동

데이터 노드를 만날 때 까지 반복 실행

searchBook

해당 도서가 존재할 데이터로 이동

해당 데이터에서 map find진행

searchRange

처음 인자의 데이터 노드로 이동 후

조건에 맞는 데이터 출력

두번째 인자 범위를 넘어갈 시 종료

print

전체 비피 트리의 가장 왼쪽 데이터 노드로 이동후

전체 데이터노드를 순회하면서 출력

deletenode

조건 분할 후 실행

1. 처음 노드가 아닌 경우(완)
   맵에서 해당 데이터 삭제
2. 첫 데이터이지만 데이터가 2개 이상인 경우(완)
   맵에서 해당 데이터 삭제 -> 데이터 순위 변경
   부모 노드를 조사하며 해당 노드가 존재하는 경우
   변경된 첫 데이터가 해당 데이터 키를 대신한다.

3. 첫 데이터이면서 데이터가 1개인 경우
   1) 인덱스 부모 노드가 존재하지 않는다면 데이터가 단 1개 남은 것.
      삭제 후 루트를 널 값으로 변경
   2) 어떤 노드가 같은 부모를 공유하는지 확인
      같은 부모를 공유하는 데이터가 2개 이상이라면 데이터 분할 실행
      삭제되는 노드의 다음 노드인지 이전 노드인가에 따라 가장 마지막 혹은 가장 처음 데이터를 분할 받음
      기존 데이터를 부모 노드에 사용하고 있었다면 지금 분할 받은 데이터로 키값 교체
   3) 같은 부모를 공유하는 데이터 또한 데이터가 하나지만 인덱스 노드의 데이터가 2개인 경우
      현재 데이터 기준으로 앞뒤 데이터 끼리 이어준다.
      현재 데이터 삭제, 인덱스 노드에서도 해당 데이터 삭제
   4) 파트너, 부모 모두 데이터가 하나인 경우. ->언더 플로우 발생

del_data_pre_size2

이전 파트너 데이터 노드가 2개 일 때 분할 실행

del_data_next_size2

다음 파트너 데이터 노드가 2개 일때 분할 실행

change_indexdata

삭제할 데이터의 부모노드부터 root노드 까지 삭제된 데이터를 key로 사용하는 경우가 존재한다면 해당 key를 현재 data의 처음 값으로 교체

underflow_happend

언더플로우가 발생해야 하는 경우에 실행

부모의 양쪽 데이터 노드가 1인 경우이다.

경우의 수를 3개로 나눈다(부모노드의 부모노드 = 조상노드)

현재 부모 노드가 조상노드의

1) 왼쪽 merge_front_index

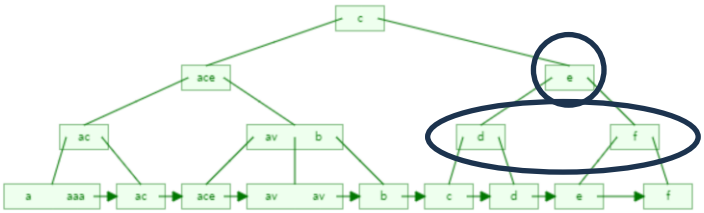
->앞의 부모 노드 인덱스 수가 2개 일 때

앞의 부모 노드의 가장 왼쪽 데이터 노드를 현재 부모노드한테 넘겨준다

현재 부모노드를 지우고 조상 노드의 key값을 받아온다
조상 노드의 key값을 다음 부모 노드의 가장 왼쪽 인덱스로 변경한다
다음 부모 노드에서 데이터를 삭제하고 Mostleftchild를 다시 설정한다.
->앞의 부모 노드의 인덱수 수가 1개일 때
->미구현

2) 가운데
양쪽 노드 중에 부모 노드 인덱스 수가 2개 인 것을 찾아 분할 실행
이전 노드의 경우-> merge_back_index
다음 노드의 경우-> merge_front_index
양쪽 부모 노드의 인덱수 수가 1개일 때
->미구현

3) 마지막 merge_back_index
->뒤의 부모 노드 인덱스 수가 2개 일 때
현재 조상 노드의 key값을 부모 노드의 mostleftchild로 가져온다.
현재 부모 노드의 mostleftchild 값을 뒤 부모의 가장마지막 데이터 노드를 가져온다.
조상 노드를 삭제한다
조상 노드에 뒤 노드에서 나눠준 인덱스 노드의 key값을 전달 받는다.
뒤 부모에서 나눠준 인덱스와 데이터 삭제;
->뒤의 부모 노드의 인덱수 수가 1개일 때
->미구현
}



위에서 표시한 것과 같이 데이터, 부모인덱스, 조부모 인덱스 까지 데이터가 1개인 경우의 예외처리는 하지 못했습니다.

LoanBookHeap{

Heapify up:
힙에 데이터 삽입 후 부모 노드와 데이터를 비교하며 힙 재정렬
부모 노드가 존재하지 않을 때 종료{
문자열 비교 후
부모 노드 보다 현재 노드가 문자열 순위가 낮을 때 교환 실행
교환을 실행 했다면 부모 노드로 이동 후 부모 노드의 부모 노드와 비교
}
heapifyDown:
힙에서 데이터 삭제 시 루트 노드를 삭제하고 가장 최근에 삽입된 데이터를 루트 노드로 이동시킨 뒤 재정렬
양쪽 노드 중 더 사전 배열이 빠른 쪽을 택해 현재 노드와 비교해 더 자식 노드가 배열이 더 빠른 경우 해당 노드와 교환 교환 시 해당 노드로 이동 후 해당 노드의 자식 노드와 비교
Insert:

힙에서 데이터를 추가하는 명령어로 div를 통해 데이터가 삽입 되어야 할 위치의 부모 노드를 구하고 load배열의 마지막 인자에 따라 왼쪽 노드로 갈지 오른쪽 노드로 이동할지 정하게 된다.

삽입된 데이터는 Heapify up 명령어를 통해 힙 내에서 정렬된다.

Div:

데이터가 삽입되어야 할 위치, 데이터가 삽입된 위치를 구하는 명령어로 해당 위치의 부모 노드를 반환하며

Count를 통해 작업을 수행한다. Count 를 2로 나눈 나머지를 load에 저장하며 해당 배열을 최근에 들어온 순서대로 읽어들어 0이면 왼쪽 1이면 오른쪽으로 이동한다.

}

Selection Tree{

Insert :

힙에다 데이터를 추가하는 함수 처음 들어온 데이터인 경우 새로운 힙 생성.

이미 힙이 존재하는 경우 해당 힙에 데이터 추가, 힙 재정렬 후 샐렉션 트리 비교

부모노드의 자식노드끼리 비교 상대 자식 노드에 데이터가 없다면 자동으로 올라가게 된다.

Delete :

셀렉션 트리의 루트 노드의 데이터를 삭제하는 명령어로 힙에서도 동일한 데이터가 삭제된다.

힙에서 삭제 후 힙에서 가장 최근에 삽입된 노드를 찾아 루트 노드로 보낸 뒤 해당 노드를 재 정렬해준다

PrintBookData :

분류 코드를 인자로 받아 해당 인자에 해당하는 힙 데이터를 출력한다. Deepcopy를 이용해 힙 노드를 복사하고

셀렉션 트리에서 힙들 삭제할 때와 같이 해당 루트 노드를 출력과 재정렬을 반복하며 데이터를 출력하면 된다.

}

4. Result Screen

LOAD , ADD, PRINT

```
LOAD
ADD 1984      100  george orwell1949
ADD 1984      100  george orwell1949
ADD to kill a monchingbird   100  harper lee     1960
ADD educated      100  tara westover      2018
ADD demian   100  hermann hesse
PRINT_BP
```

========LOAD========
SUCCESS
====================
========ADD========
1984/100/george orwell/1949
====================
========ADD========
1984/100/george orwell/1949
====================
========ADD========
to kill a monchingbird/100/harper lee/1960
====================
========ADD========
educated/100/tara westover/2018
====================

성공 메시지 출력

======================
ERROR 200 데이터 부족으로 인한 에러 출력
======================

1984/100/george orwell/1949/2 대출 도서 수 증가
a tale of two cities/400/charles dickens/2012/0
and then there were none/100/dame agatha christie/1972/0
chainsaw man/100/huzimototakeuki/2019/0
demian/100/hermann hesse/1919/0
educated/100/tara westover/2018/1 대출 도서 수 증가
harry potter and the philosopher's stone/600/j. k. rowling/1997/0
i want to eat your pancreas/300/seminoyoru/2015/0
ivan the fool/100/leo tolstoy/1886/0
jujutsu kaisen1/200/akutamekake/2018/0
jujutsu kaisen2/200/akutamekake/2018/0
jujutsu kaisen3/200/akutamekake/2018/0
the little prince/300/antoine de/1943/0
the lord of the rings/500/j. r. r. tolkien/1954/0
the power of now/100/echhart tolle/1997/0
the selfish giant/100/oscar fingal/1888/0
to kill a monchingbird/100/harper lee/1960/1 대출 도서 수 증가
your name/0/sinkaimakoto/2017/0

```
SEARCH_BP   a    b
SEARCH_BP   z    z
SEARCH_BP   1984
LOAD
```

```
========SEARCH_BP========
a tale of two cities/400/charles dickens/2012/0
and then there were none/100/dame agatha christie/1972/0
=========================
=========================
ERROR 300
=========================
```

범위 데이터 출력

데이터 존재하지 않을시 에러 출력

```
========SEARCH_BP========
1984/100/george orwell/1949/2
=========================
=========================
ERROR 100
=========================
```

이름을 통한 데이터 검색

이미 데이터가 존재할 때 로드 진행시 에러

```
LOAD
PRINT_ST    100
PRINT_ST    200
PRINT_ST    300
PRINT_ST    400
PRINT_ST    500
PRINT_ST    600
```

Delete에서 예외처리 못한 부분이 있어 로드를 변경해 데이터를 Stree로 바로 넣어 테스트를 진행했습니다

```
========PRINT_ST========
1984/100/george orwell/1949/0
and then there were none/100/dame agatha christie/1972/0
chainsaw man/100/huzimototakeuki/2019/0
demian/100/hermann hesse/1919/0
educated/100/tara westover/2018/0
ivan the fool/100/leo tolstoy/1886/0
the power of now/100/echhart tolle/1997/0
the selfish giant/100/oscar fingal/1888/0
to kill a monchingbird/100/harper lee/1960/0
=========================
========PRINT_ST========
jujutsu kaisen1/200/akutamekake/2018/0
jujutsu kaisen2/200/akutamekake/2018/0
jujutsu kaisen3/200/akutamekake/2018/0
=========================
========PRINT_ST========
i want to eat your pancreas/300/seminoyoru/2015/0
the little prince/300/antoine de/1943/0
=========================
========PRINT_ST========
a tale of two cities/400/charles dickens/2012/0
=========================
========PRINT_ST========
the lord of the rings/500/j. r. r. tolkien/1954/0
=========================
========PRINT_ST========
harry potter and the philosopher's stone/600/j. k. rowling/1997/0
=========================
=========================
ERROR 700
=========================
```

각 코드별로 데이터가 출력되는 것을 확인 가능하다.

```
LOAD
PRINT_ST    100
DELETE
DELETE
DELETE
PRINT_ST    100
```

```
========PRINT_ST========
1984/100/george orwell/1949/0
and then there were none/100/dame agatha christie/1972/0
chainsaw man/100/huzimototakeuki/2019/0
demian/100/hermann hesse/1919/0
educated/100/tara westover/2018/0
ivan the fool/100/leo tolstoy/1886/0
the power of now/100/echhart tolle/1997/0
the selfish giant/100/oscar fingal/1888/0
to kill a monchingbird/100/harper lee/1960/0
========================
========DELETE========
            Success
========================
========DELETE========
            Success
========================
========DELETE========
            Success
========================
========PRINT_ST========
chainsaw man/100/huzimototakeuki/2019/0
demian/100/hermann hesse/1919/0
educated/100/tara westover/2018/0
ivan the fool/100/leo tolstoy/1886/0
the power of now/100/echhart tolle/1997/0
the selfish giant/100/oscar fingal/1888/0
to kill a monchingbird/100/harper lee/1960/0
========================
```

전체에서 가장 상위 데이터인

1984와 and the ~ 이 삭제

코드에서 데이터가 삭제되는 것을 확인 가능하다.

5. Consideration

이번 프로젝트에서는 비플러스 트리를 이용한 도서 관리 프로그램을 구현했다. 처음 프로젝트를 받았을 땐 MAP이라는 처음보는 자료 형태가 스켈레톤 코드안에 있어서 많이 당황했다. 키 값을 어떻게 불러와야 하고 데이터의 접근이 어떻게 이루어 지는지 알지 못했기 때문이다. 하지만 이번 프로젝트 기간동안 계속 찾아보고 사용하니 되게 편리한 데이터 구조 형태인 것 같았다. key에따라 자동으로 정렬을 해주고, key값만 알고 있다면 find를 이용해 언제든지 내가 원하는 데이터에 접근 할 수 있었다.

이번 bp, heap, selection에서 예상처럼 구현되는 자료형태가 selection트리 하나였다.
Heap의 경우 이진 트리를 이용하는 것이기 때문에 BST와 어느정도 유사할 것이라 생각했지만 배열을 사용하지 않고 링크드리스트를 이용한 경우 구현이 까다로웠다. 항상 완전 이진 트리를 유지하기 때문에 삽입, 삭제할 위치가 정해져 있어 해당 위치를 따로 구하는 코드를 구현해야 했기 때문이다.
비피트리 구현 과정에서 삭제할 때 고려해야 할 상황이 많았다. 이 문제를 해결하면서 깨달은 것은 함수 구현을 잘 해두면 편하다는 것이다. 인덱스 노드가 하나일 때, 두 개 일 때 , 데이터 노드가 개수가 어떻게 됐을 때를 생각하면서 일일이 구현해 보니 몇몇 파일들을 합칠 수 있겠다고 생각했고 실제로 가능했기 때문이다. 이번 과제를 통해 간단한 자료구조 프로그램이라도 생각보다 고려해야 할 사항들이 많았고 또 그 것들을 해결하기 위한 함수를 잘 구현한다면 하나의 함수를 이용해 여러 문제 해결 과정에서 쓸 수 있다는 것을 알게 되었다.