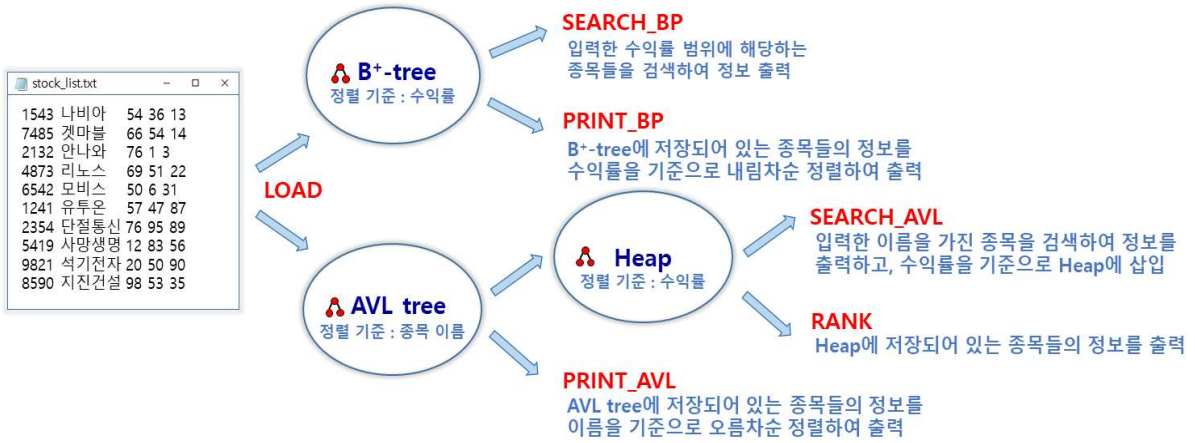
Data Structure Lab. Project #2

2018년 10월 10일

Due date: 2018년 11월 11일 일요일 23:59:59

주식 정보 검색 시스템은 종목들의 주식 정보를 저장하고, 다양한 기준을 적용하여 효율적으로 검색할 수 있는 시스템이다.

본 프로젝트는 다양한 자료 구조를 활용하여 주식 정보 시스템을 구축하는 것으로, 주식들의 주가 데이터를 수익률 또는 종목 이름순으로 정렬하여 각각 B^+-tree와 AVL tree에 저장한다. B^+-tree는 전체 데이터 출력 기능과 특정 종목들 중 원하는 수익률에 포함되는 종목들의 정보 검색 기능이 있고 AVL tree는 전체 데이터 출력 기능과 특정 종목의 수익률 정보 검색 기능이 있다. 이 때, AVL tree는 검색한 주가 데이터를 Heap을 이용해 수익률순으로 출력하는 기능을 추가적으로 가진다. B^+-tree와 AVL tree는 삽입 및 탐색 기능만 있을 뿐, 삭제 및 수정 기능은 구현하지 않는다.

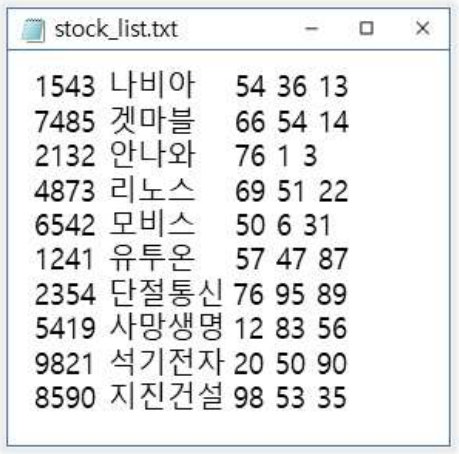


[그림 1] 주식 정보 검색 시스템

□ Program implementation

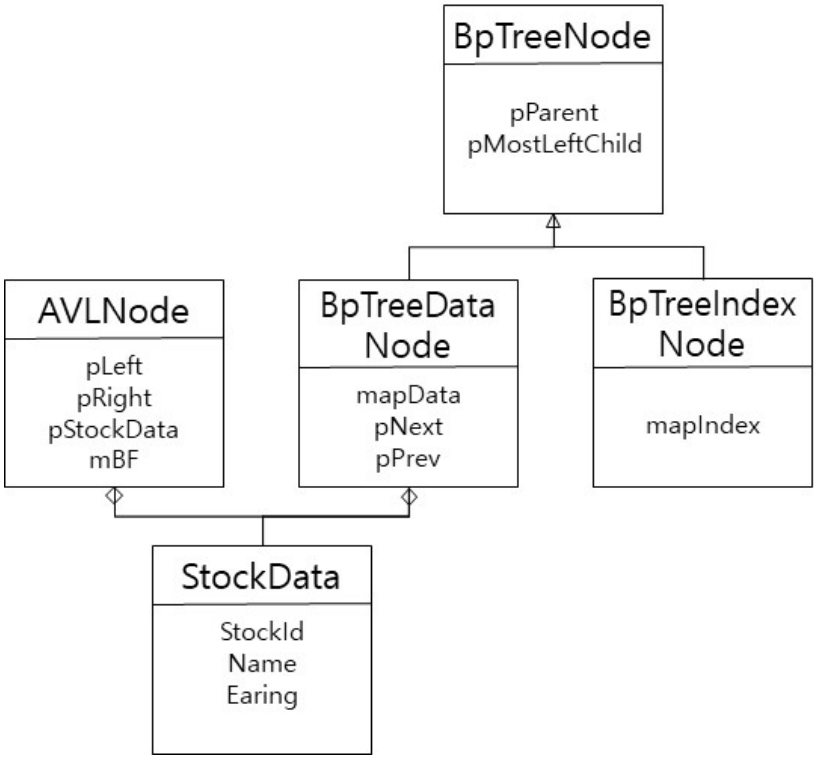
1. 주식 정보 저장

주식 정보 검색 시스템은 종목들의 주식 정보가 저장되어 있는 텍스트 파일(stock_list.txt)을 읽어, 해당 정보를 알맞은 자료구조에 저장한다. 텍스트 파일에는 고유번호, 종목 이름, 시가, 종가, 거래량이 순서대로 저장되어 있다. 고유번호, 종목 이름은 '주식 정보 클래스(StockData)'에 저장하며, 시가, 종가, 거래량은 '주식 정보 클래스'에 저장한다. 이 때, 각 주식 정보 클래스에 있는 시가, 종가에 따른 수익률 순서대로 수익률을 저장한다. 모든 종목들은 세가지의 데이터를 가지고 있으며, 시가, 종가를 이용해 수익률을 계산하여 주식 정보 클래스에 저장한다. 수익률은 종가에서 시가를 뺀 값에 시가를 나누어 계산하는데, 항상 소수점 둘째 자리까지 반올림하여 저장한다. 텍스트 파일에서 중복된 고유번호의 입력은 없다고 가정한다. 종목들의 주식 정보가 저장되어 있는 텍스트 파일의 예시와 주식 정보 클래스는 다음과 같다. 각 변수에 대한 자세한 내용은 '구현 시 반드시 정의해야하는 Class 및 멤버 변수'에 작성되어 있다.



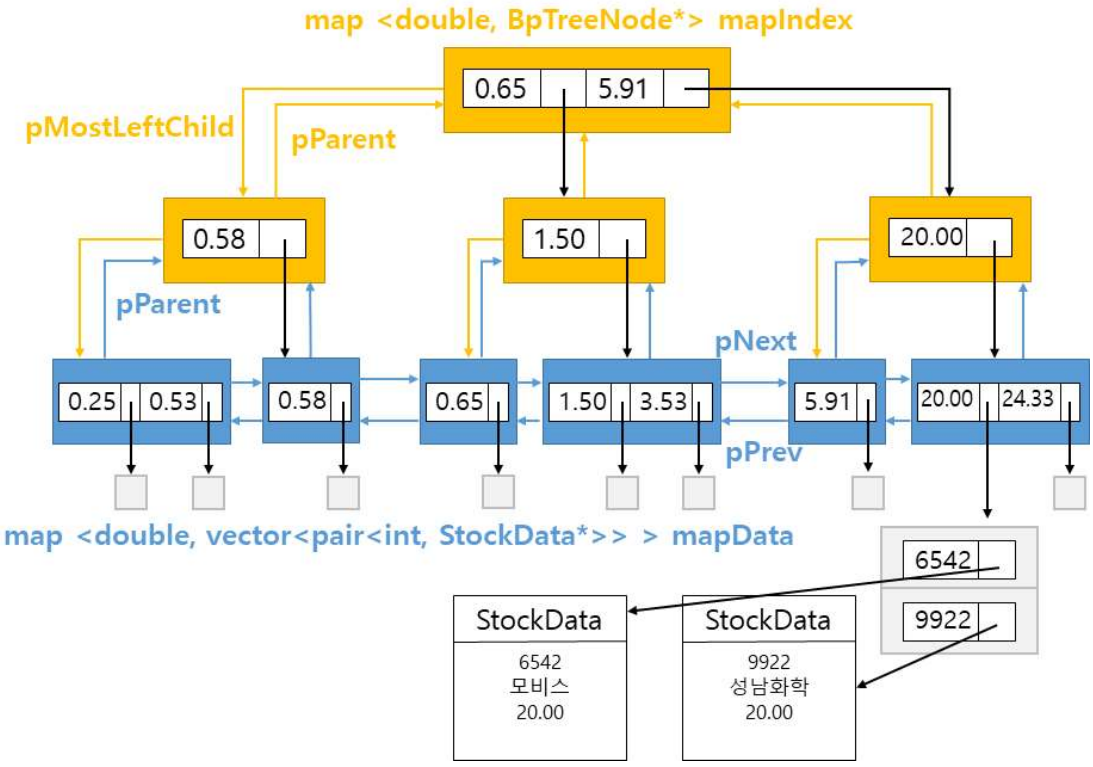
[그림 2] 주식들의 주가 정보가 저장되어 있는 텍스트 파일의 예

주식들의 주가 정보는 주식 정보 클래스를 이용한 객체로 저장되고 B^+-tree와 AVL tree를 구성하는 노드들은 해당 객체 정보를 가진다. B^+-tree와 AVL tree는 서로 다른 노드를 이용하여 구성되며 주식 정보 클래스와 B^+-tree 노드 클래스, AVL tree 노드 클래스의 관계는 다음과 같다. 각 변수에 대한 자세한 내용은 '구현 시 반드시 정의해야하는 Class 및 멤버 변수'에 작성되어 있다.



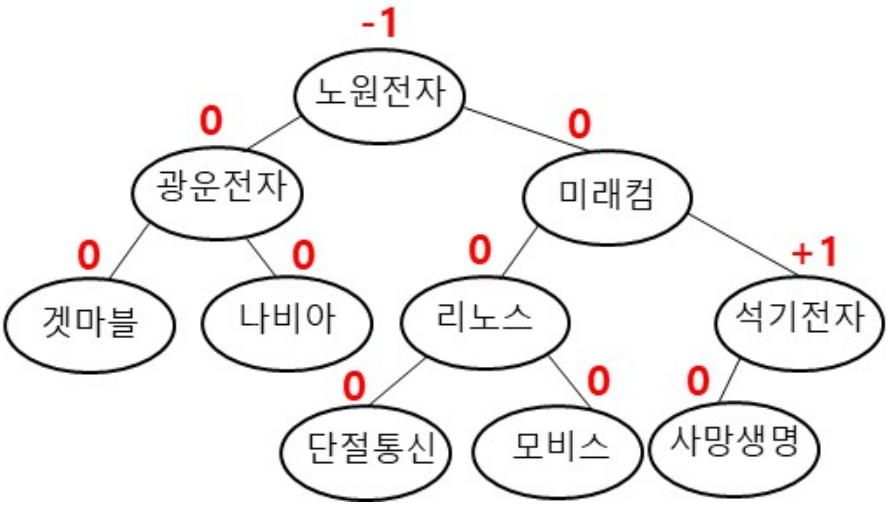
[그림 4] 각 트리의 노드 클래스와 주식 정보 클래스의 관계

B⁺-tree는 수익률을 기준으로 종목들의 객체 정보를 저장하고 있으며, 수익률이 같은 경우 고유번호의 오름차순으로 정렬한다. B⁺-tree(BpTree)는 인덱스 노드(BpTreeIndexNode)와 데이터 노드(BpTreeDataNode)로 구성되며, 각 노드 클래스는 B⁺-tree 노드 클래스(BpTreeNode)를 상속받는다. 데이터 노드는 단말 노드로, 종목들의 객체 정보를 map 컨테이너 형태로 가지고 있다. 인덱스 노드는 비단말 노드로, 인덱스와 자식 노드를 가리킬 포인터를 map 컨테이너 형태로 가지고 있으며 가장 왼쪽 자식을 가리키는 포인터를 따로 가지고 있다. 자세한 노드 클래스 정보는 '구현 시 반드시 정의해야하는 Class 및 멤버 변수'에 작성되어 있다. B⁺-tree의 차수 ORDER(m)는 고정되어 있지 않으며 멤버 변수로 변경이 가능하다. B⁺-tree 구성의 예는 다음과 같다.



[그림 5] B+-tree 구성의 예

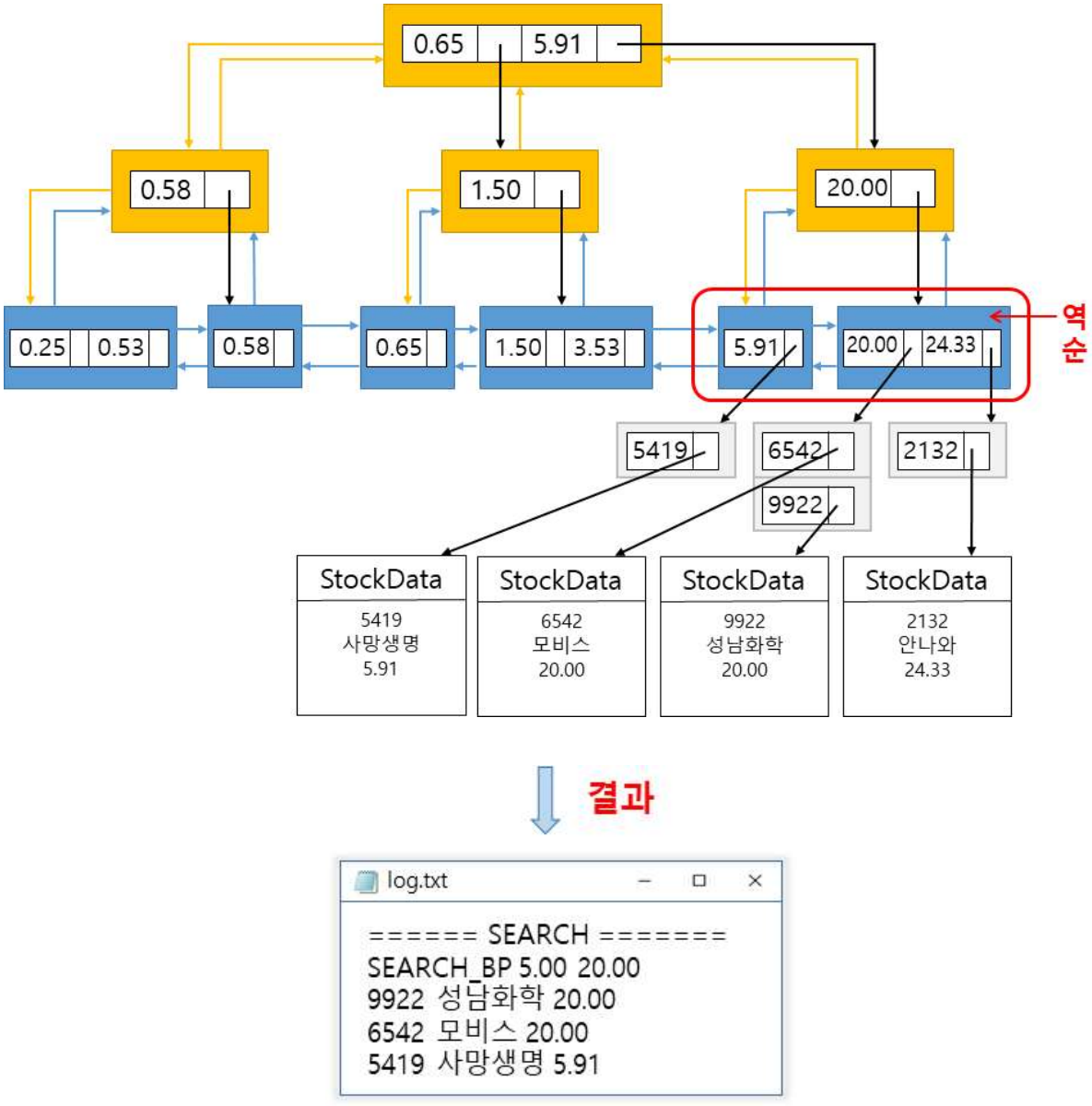
AVL tree는 종목의 이름을 기준으로 종목들의 객체 정보가 저장되어 있다. 이 때, 종목 이름이 같은 경우는 없는 것으로 가정한다. AVL tree는 각 노드마다 balance factor를 가지고 있으며, 이를 활용하여 트리의 균형을 유지한다. AVL tree 구성의 예시는 다음과 같다.



[그림 6] AVL tree 구성의 예

2. B⁺-tree 검색

주식 정보 검색 시스템은 B⁺-tree를 통해 종목들 중 원하는 수익률 범위에 포함되는 종목들의 정보를 검색할 수 있다. 검색은 수익률을 기준으로 역순으로 탐색하면서 주식 정보를 출력한다. 만약 주식 종목 중 수익률이 5.00이상 20.00이하인 종목을 검색한다면, 그 결과는 다음과 같다.



[그림 7] B⁺-tree 검색의 예

3. AVL tree 검색

주식 정보 검색 시스템은 AVL tree를 통해 특정 이름을 가진 주식의 정보를 검색할 수 있다. 검색을 수행할 때마다 그 결과는 출력함과 동시에 vector로 이루어진 Max-Heap에 삽입한다. Heap의 정렬 기준은 수익률(MAX) 및 고유번호(MIN)을 기준으로 한다. RANK 명령을 수행할 때, Heap에 저장된 모든 주식 데이터들이 출력 및 삭제된다.

본 프로젝트에서 Max-heap은 표준 템플릿 라이브러리(STL)를 이용하여 구현한다. 시퀀스 컨테이너인 vector를 이용하며, 아래 멤버 함수를 사용하여 구현한다.

- void make_heap(vector의 시작, vector의 끝) : vector의 모든 원소를 heap에 삽입
- void push_heap(vector의 시작, vector의 끝) : vector의 마지막 원소를 heap에 삽입
- void pop_heap(vector의 시작, vector의 끝) : heap의 가장 위에 있는 원소가 vector의 마지막으로 이동

□ Functional Requirements

주식 정보 검색 시스템은 명령어를 통해 동작하며, 실행할 명령어가 작성되어 있는 텍스트 파일(command.txt)을 읽고 명령어에 따라 순차적으로 동작한다. 모든 명령어는 반드시 대문자로 입력하며, 각 명령어의 사용법과 기능은 다음과 같다.

명령어	명령어 사용법 및 기능
LOAD	사용법) LOAD ✓ 주식 정보 검색 시스템에 종목들의 주식 정보를 불러오는 명령어로, 종목들의 주식 데이터가 저장되어 있는 텍스트 파일(stock_list.txt)을 읽어 B^+-tree와 AVL tree를 구성한다. ✓ 범위(1000~9999)가 맞지 않는 고유번호의 입력, 주가 데이터가 아닌 다른 문자열의 입력은 없다고 가정한다. ✓ 중복된 고유번호을 입력하는 경우, 고유번호/종목이름/3개의 주가 데이터 중 하나라도 입력하지 않은 경우는 없다고 가정한다. ✓ 텍스트 파일이 존재하지 않거나, 트리가 이미 구성되어 있는 경우, 출력 파일(log.txt)에 에러 코드(100)를 출력한다.
SEARCH_BP	사용법) SEARCH_BP 시작범위 끝범위 예시) SEARCH_BP 3.00 3.60 ✓ 입력한 수익률 범위(-1~100)에 속하는 종목들을 B^+-tree에서 검색한 후, 해당 종목의 정보를 수익률의 내림차순으로 출력 파일(log.txt)에 출력한다. ✓ 수익률의 시작 범위와 끝 범위는 항상 소수점 둘째짜리까지 입력한다고 가정하며, 범위(-1.00~1.00)가 맞지 않는 수익률의 입력, 시작 범위가 끝

	범위보다 큰 경우의 입력은 없다고 가정한다. ✓ 입력한 조건의 종목이 존재하지 않거나, 시작 범위, 끝 범위 중 하나라도 입력하지 않은 경우, 출력 파일(log.txt)에 에러 코드(200)를 출력한다.
SEARCH_AVL	사용법) SEARCH_AVL 이름 예시) SEARCH_AVL 사망생명 ✓ 입력한 이름을 가진 주식의 주가 데이터를 검색하는 명령어로, 종목의 이름을 AVL tree에서 검색하여 출력 파일(log.txt)에 출력한다. ✓ 한글이 아닌 이름의 입력은 없다고 가정한다. ✓ 입력한 이름의 주식이 존재하지 않거나, 이름을 입력하지 않은 경우, 출력 파일(log.txt)에 에러 코드(300)를 출력한다.
RANK	사용법) RANK ✓ AVL tree에서 검색하였던 종목들을 수익률의 내림차순, 수익률이 같을 경우 고유번호의 오름차순으로 출력하는 명령어로, Heap에 삽입되어 있는 종목들의 수익률 정보를 삭제하면서 출력 파일(log.txt)에 출력한다. ✓ Heap에 저장되어 있는 정보가 없을 경우, 출력 파일(log.txt)에 에러 코드(400)를 출력한다.
PRINT_BP	사용법) PRINT_BP ✓ B^+-tree를 탐색하여, 저장되어 있는 모든 정보를 출력하는 명령어로, 출력 파일(log.txt)에 수익률의 내림차순으로 출력한다. 수익률이 같을 경우, 고유번호의 오름차순으로 출력한다. ✓ B^+-tree에 저장되어 있는 정보가 없을 경우, 출력 파일(log.txt)에 에러 코드(500)를 출력한다.
PRINT_AVL	사용법) PRINT_AVL ✓ AVL tree를 탐색하여, 저장되어 있는 모든 정보를 출력하는 명령어로, 출력 파일(log.txt)에 종목 이름의 오름차순으로 출력한다. ✓ AVL tree에 저장되어 있는 정보가 없을 경우, 출력 파일(log.txt)에 에러 코드(600)를 출력한다.
EXIT	사용법) EXIT ✓ 주식 정보 검색 시스템을 종료하는 명령어로, 종료 시 할당된 메모리를 모두 해제한다.

□ Error Code

명령어에 인자(parameter)가 모자라거나 필요 이상으로 입력 받을 경우, 또는 각 명령어마다 에러 코드를 출력해야 하는 경우, 출력 파일(log.txt)에 명령어 별로 알맞은 에러 코드를 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 출력하여 저장한다. 각 명령어별 에러 코드는 다음과 같다.

명령어	에러 코드
LOAD	100
SEARCH_BP	200
SEARCH_AVL	300
RANK	400
PRINT_BP	500
PRINT_AVL	600

□ Print Format

명령어 동작이 성공하였거나 실패한 경우, 출력 파일(log.txt)에 해당 내용을 지정된 형식에 맞추어 출력한다. 출력 파일(log.txt)이 이미 존재할 경우, 기존 내용 뒤에 이어서 추가로 저장한다. 명령어별 출력 형식은 다음과 같다.

명령어	출력 형식
에러(ERROR)	======== ERROR ======== 100 ======================
LOAD	======== LOAD ======== Success ======================
SEARCH_BP	======= SEARCH ======= SEARCH_BP 5.00 20.00 9922 성남화학 20.00 6542 모비스 20.00 5419 사망생명 5.92 ======================
SEARCH_AVL	======= SEARCH ======= SEARCH_AVL 성남화학 9922 성남화학 20.00 ======================

RANK	========== RANK ========== 2132 안나와 24.33 시가: 3 종가: 76 거래량: 3 수익률: 24.33 9922 성남화학 20.00 시가: 2 종가: 42 거래량: 64 수익률: 20.00 ... ==========
PRINT_BP	========== PRINT ========== 2132 안나와 24.33 시가: 3 종가: 76 거래량: 3 수익률: 24.33 9922 성남화학 20.00 시가: 2 종가: 42 거래량: 64 수익률: 20.00 6542 모비스 20.00 시가: 4 종가: 84 거래량: 31 수익률: 20.00 ... ==========

PRINT_AVL	==================== PRINT ==================== 9922 성남화학 20.00 시가: 2 종가: 42 거래량: 64 수익률: 20.00 2132 안나와 24.33 시가: 3 종가: 76 거래량: 3 수익률: 24.33 ... ==

☐ Program Behavior Examples

command.txt

```
LOAD
SEARCH_BP 5.00 20.00
SEARCH_AVL 성북화학
SEARCH_AVL 성남화학
SEARCH_AVL 단절통신
SEARCH_AVL 사망생명
SEARCH_AVL 모비스
RANK
PRINT_BP
PRINT_AVL
EXIT
```

실행 결과 화면 및 log.txt

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

======= SEARCH =======
SEARCH_BP 5.00 20.00
9922 성남화학 20.00
6542 모비스 20.00
5419 사망생명 5.92
======================

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

```
======== SEARCH =======
SEARCH_AVL 성남화학
9922 성남화학 20.00
=========================

======== SEARCH =======
SEARCH_AVL 단절통신
2354 단절통신 0.25
=========================

======== SEARCH =======
SEARCH_AVL 사망생명
5419 사망생명 5.92
=========================


======== SEARCH =======
SEARCH_AVL 모비스
6542 모비스 20.00
=========================

==================== RANK ====================
6542 모비스 20.00
시가: 4
종가: 84
거래량: 31
수익률: 20.00

9922 성남화학 20.00
시가: 2
종가: 42
거래량: 64
수익률: 20.00

5419 사망생명 5.92
시가: 12
종가: 83
거래량: 56
수익률: 5.92

2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25
=============================================
```

```
==================== PRINT ====================
2132 안나와 24.33
시가: 3
종가: 76
거래량: 3
수익률: 24.33

9922 성남화학 20.00
시가: 2
종가: 42
거래량: 64
수익률: 20.00
…
===============================================

==================== PRINT ====================
2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25

6542 모비스 20.00
시가: 4
종가: 84
거래량: 31
수익률: 20.00
…
===============================================
```

□ 구현 시 반드시 정의해야하는 Class 및 멤버 변수

1. StockData : 주식 정보 클래스

항목	내용	비고
StockId	고유번호	1000번부터 9999번까지만 있는 것으로 가정
Name	이름	이름은 반드시 한글로 저장한다고 가정
Earning	수익률	수익률 = (종가-시가)/시가
Opening_Price	시가	int
Closing_Price	종가	int
Volume	거래량	int

6. BpTreeNode : B^+-tree의 노드 클래스

항목	내용	비고
pParent	상위 BpTreeNode를 가리킴	
pMostLeftChild	하위 BpTreeNode 중 가장 왼쪽 노드를 가리킴	

7. BptreeIndexNode : B^+-tree의 인덱스 노드 클래스

항목	내용	비고
mapIndex	인덱스 값을 가지고 있는 map 컨테이너	map<double, BpTreeNode*> (수익률, 하위 B^+-Tree 노드*)

8. BptreeDataNode : B^+-tree의 데이터 노드 클래스

항목	내용	비고
mapData	주식 정보 데이터를 가지고 있는 map 컨테이너	map<double, vector<pair<int, StockData*>>> (수익률, map(고유번호, 주식 정보 객체*))
pNext	다음 BpTreeNode를 가리킴	
pPrev	이전 BpTreeNode를 가리킴	

9. AVLNode : AVL tree의 노드 클래스

항목	내용	비고
pRight	오른쪽 하위 노드를 가리킴	
pLeft	왼쪽 하위 노드를 가리킴	
pStockData	주식 정보 객체를 가리킴	
mBF	balance factor	

10. Bptree : B^+-tree 클래스

항목	내용	비고
root	B^+-tree의 루트	
order	B^+-tree의 차수(m)	멤버 변수로 변경 가능

11. AVLtree : AVL tree 클래스

항목	내용	비고
root	AVL tree의 루트 노드를 가리킴	

12. Manager : 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

항목	내용	비고
avl	AVL Tree	
bp	B^+-Tree	

□ Files

- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ stock_list.txt : 종목들의 주식 정보를 저장하고 있는 파일
- ✓ log.txt : 프로그램 출력 결과를 모두 저장하고 있는 파일

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 압축파일(project2.tar.gz)를 이용하여 구현하며 작성된 소스 파일의 이름과 클래스와 함수 이름 및 형태를 절대 임의로 변경하지 않는다.
- ✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
- ✓ 제시된 Class를 각 기능에 알맞게 모두 사용한다.
- ✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- ✓ 채점 시 코드를 수정해야 하는 일이 없도록 한다.
- ✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점)
- ✓ 프로그램은 반드시 리눅스에서 동작해야한다. (컴파일 에러 발생 시 감점)
  - 제공되는 Makefile을 사용하여 테스트 하도록 한다.
- ✓ 소멸자는 제공한 소스코드를 사용하며, 따로 구현하지 않아도 된다.

□ 제출기한 및 제출방법

1. 소스 코드 제출

- ✓ 제출기한
  - **2018년 11월 11일 일요일 23:59:59 까지 제출**
- ✓ 제출방법
  - **개인별 제출**
  - 소스코드(Makefile과 텍스트 파일 제외)를 모두 압축하여 제출
    - 확장자가 .cpp, .h가 아닌 파일은 제출하지 않음
  - 1차 프로젝트와 동일한 방법으로 제출
- ✓ 제출 형식 (제출 형식과 다를시 감점)
  - 학번_DS_project2.tar.gz
  - .zip, .tar 형식 제출시 감점
  - 학번_DS_project2.tar.gz 압축파일에는 반드시 .cpp, .h만 있어야 함

2. 보고서 제출

✓ 제출기한

- 2018년 11월 11일 일요일 23:59:59 까지 제출

✓ 제출방법

- **개인별 제출**
- 보고서 파일을 pdf로 변환하고, zip형식으로 압축하여 제출
  - 보고서 파일 확장자가 pdf가 아닐 시 감점
- 제출 방법 추후 공지

✓ 보고서 작성 형식

- 하드카피는 제출하지 않음
- 보고서는 한글로 작성
- 보고서에는 소스코드를 포함하지 않음
- 반드시 포함해야하는 항목
  - Introduction : 프로젝트 내용에 대한 설명
  - Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명(모든 명령어에 대하여 각각 그릴 것)
  - Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명(B^+-tree 삽입, AVL tree 삽입, Heap push, Heap pop에 대하여 각각 예시를 들어 설명할 것)
  - Result Screen : 모든 명령어에 대하여 각각 결과화면을 캡처하고 동작을 설명
  - Consideration : 고찰 작성

※ 고찰 작성법

· 1page 이상 작성

· 자신이 스스로 프로젝트를 진행하면서 배운 점, 어려웠던 점, 해결 방안 등을 작성

· (예: AVLtree class의 Insert()에서 root가 없을 때의 예외처리를 하지 않았다.)