데이터구조설계

Project 2

담당교수 최상호 교수님

학 과 컴퓨터정보공학부

학 번 2015722035

성 명 최한솔

날 짜 2021. 11. 12 (금)

광운대학교
KwangWoon University

Ⅰ. Introduction

본 프로젝트에서는 B+ Tree, AVL Tree, STL Vector를 이용하여 코로나 19 예방접종 관리 프로그램을 구현한다. 예방접종에 대한 정보는 접종자의 이름, 접종 백신 명, 접종 횟수, 접종자의 나이, 접종자의 거주지역명에 대한 멤버 변수를 가진 Vaccination Data 클래스로 구성된다. 백신에 따른 접종 횟수를 통해 접종을 완료한 사람과 완료하지 않은 사람으로 구분하여 각각을 B+ Tree와 AVL Tree를 활용하여 관리한다.

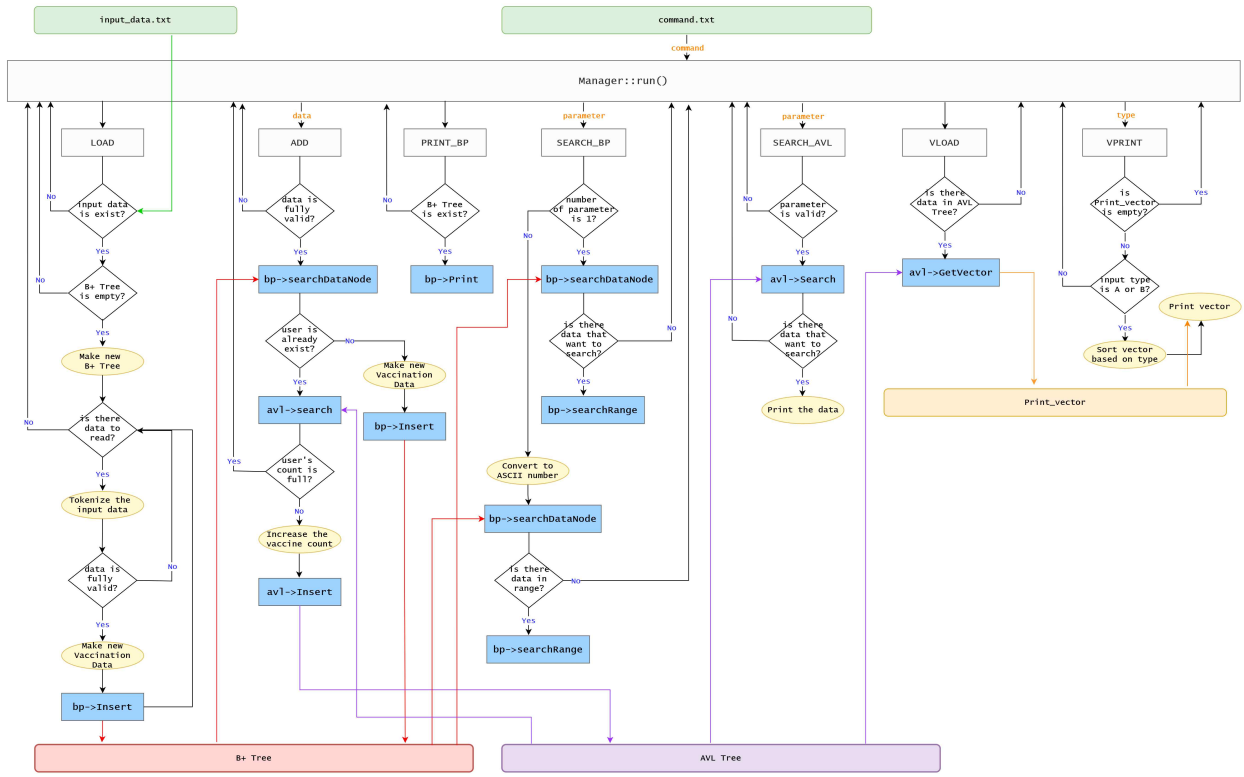
B+ Tree는 접종을 모두 완료하지 않은 접종자의 관리를 수행한다. LOAD 명령어를 통해서 모든 접종자 정보를 입력받아 최초로 B+ Tree를 구성하게 되며, ADD 명령어를 통해서 추가적인 접종자의 정보를 입력받아 B+ Tree에 적용하게 된다. B+ Tree를 구성하는 인덱스 노드 클래스와 데이터 노드 클래스는 B+ Tree 노드 클래스를 상속받는다. 인덱스 노드는 map 컨테이너 형태로 접종자의 이름에 해당하는 키값과 자식 노드 포인터를 저장하며, 데이터 노드는 map 컨테이너 형태로 접종자의 이름과 Vaccination Data 클래스 포인터를 저장한다. 또한, 모든 데이터 노드들은 doubly linked list로 연결된다.

AVL Tree는 접종을 모두 완료한 접종자의 관리를 수행한다. ADD 명령어를 통해 추가적인 접종자의 정보를 B+ Tree에 입력받는 과정에서, 접종을 모두 완료한 접종자의 경우에 AVL Tree에 삽입한다. 또한, SEARCH_BP, SEARCH_AVL, PRINT_BP 명령어를 통해서 각각의 트리에서 정보를 검색하거나 출력할 수 있다.

STL Vector는 모든 접종 완료자의 정보를 저장하고 이를 출력하는 역할을 수행한다. VLOAD 명령어를 통해 AVL Tree에 존재하는 모든 접종 완료자의 정보를 읽어와서 저장한다. 그리고 VPRINT 명령어를 통해 2가지 방식의 출력을 수행하게 된다.

이와 같이, 본 프로젝트는 B+ Tree와 AVL Tree를 통해 접종자 혹은 접종 완료자의 예방접종 정보를 효율적으로 구축, 관리하며, STL Vector를 통해 접종 완료자의 정보를 조건에 맞는 방식으로 출력하는 예방 접종 관리 프로그램을 구현하는 것을 목표로 한다.

Ⅱ. Flow Chart



예방접종 관리 프로그램의 전체적인 플로우 차트는 위의 그림과 같다. command.txt 파일에서 명령어를 읽어와 이에 해당하는 모듈을 실행시킨다. 명령어에는 LOAD, ADD, PRINT_BP, SEARCH_BP, SEARCH_AVL, VLOAD, VPRINT 등이 존재한다.

1. LOAD

input_data.txt 파일의 데이터를 읽어와 B+ Tree를 구축하는 모듈이다. 파일에서 읽어오는 데이터는 예방 접종자의 이름, 접종 백신의 이름, 접종 횟수, 접종자의 나이, 접종자의 거주지역 이름에 해당하는 정보로 이루어진다. 파일에서 읽어온 데이터에서 해당 정보들이 모두 온전히 존재하는 경우에 기존의 B+ Tree가 존재하는지를 확인한다. 기존의 B+ Tree가 존재하지 않는 경우, 새로운 B+ Tree를 생성하고 파일의 모든 데이터를 읽어와 Vaccination Data 객체를 생성, 이를 B+ Tree에 삽입한다.

2. ADD

직접 B+ Tree에 예방 접종자의 정보를 추가하는 모듈이다. 입력받은 데이터의 모든 정보가 온전히 존재하는 경우, 기존의 B+ Tree에 해당 접종자의 정보가 존재하는지를 확인한다. 기존 정보가 존재하는 경우, 해당 접종자의 접종 횟수를 1회 증가시키며 접종을 완료한 경우에는 AVL Tree에 삽입한다. 기존의 B+ Tree에 해당 접종자의 정보가 존재하지 않는 경우, 새로운 Vaccination Data 객체를 생성하여 B+ Tree에 삽입한다.

3. PRINT_BP

B+ Tree를 출력하는 모듈이다. B+ Tree가 존재하는 경우, B+ Tree의 데이터 노드에 접근하여 모든 예방 접종자들의 정보를 출력한다.

4. SEARCH_BP

B+ Tree를 탐색하는 모듈이다. Exact match 방식과 Range search 방식의 2가지 탐색 방법을 제공한다. 해당 모듈이 전달받은 인자의 개수가 1개일 경우, Exact match 방식의 탐색을 수행하고, 전달받은 인자의 개수가 2개일 경우에는 Range search 방식의 탐색을 수행한다. Exact match 탐색 방식은 기존의 B+ Tree에 인자를 통해 입력받은 이름의 접종자가 존재하는지를 확인하고 존재하는 경우, 이에 해당하는 접종자의 정보를 출력한다. Range search 탐색 방식은 두 개의 인자로 각각 하나의 알파벳을 입력받는다. 해당 모듈은 이를 ASCII 숫자로 변환하여 두 알파벳의 사이 범위에 존재하는 알파벳으로 시작되는 접종자의 이름이 존재하는지 확인하고 존재하는 경우, 해당 접종자들의 정보를 출력한다.

5. SEARCH_AVL

입력받은 접종자의 이름을 통해 AVL Tree를 탐색하는 모듈이다.
입력받은 인자가 유효한 경우에 기존의 AVL Tree를 탐색하여 이에 해당하는 접종 완료자의 정보를 출력한다.

6. VLOAD

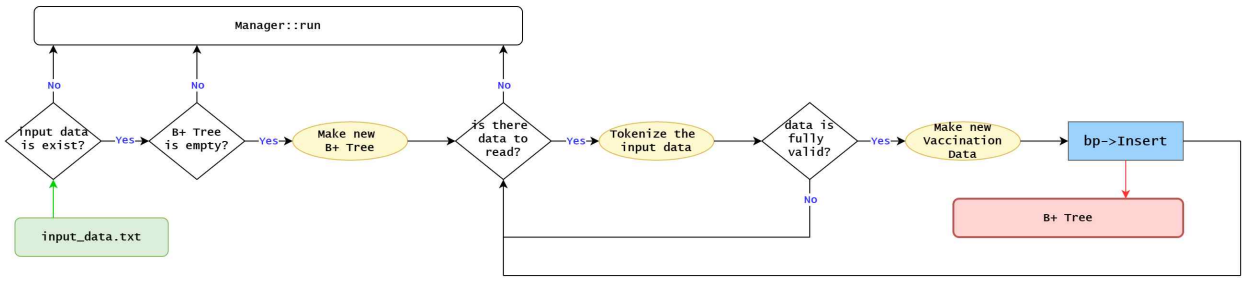
AVL Tree에 존재하는 접종 완료자의 Vaccination Data 객체를 벡터에 저장하는 모듈이다.
기존의 AVL Tree에 데이터가 존재하는 경우에 트리를 순회하며 모든 접종 완료자의 정보를 읽어와 Print_vector에 저장한다.

7. VPRINT

Print_vector를 출력하는 모듈이다.
벡터에 데이터가 존재하는 경우, 인자로 입력받은 타입에 해당하는 방식으로 벡터를 정렬하고 이를 출력한다.

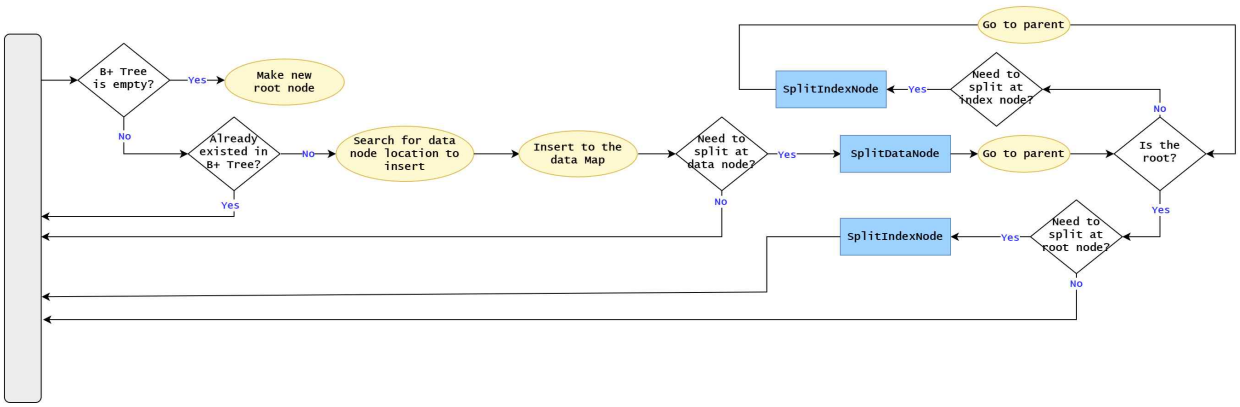
III. Algorithm

1. LOAD



① <input_data.txt> 파일의 데이터를 읽어온다.
  - 파일이 존재하지 않을 경우, 모듈을 종료한다.
② 기존의 B+ Tree가 존재하는지 확인한다.
  - 기존의 B+ Tree가 존재하는 경우, 모듈을 종료한다.
③ 새로운 B+ Tree를 생성한다.
④ <input_data.txt> 파일에 존재하는 데이터를 한 라인씩 읽어온다.
  - 더 이상 읽어올 데이터가 존재하지 않을 경우, 모듈을 종료한다.
⑤ 읽어온 한 라인의 데이터를 접종자의 이름, 백신 명, 접종 횟수, 접종자의 나이, 거주지역을 나타내는 각각의 데이터로 분할한다.
  - 모든 데이터가 온전히 존재하지 않을 경우, ④로 이동하여 다음 라인의 데이터를 읽는다.
⑥ 분할한 각각의 데이터를 통해 새로운 Vaccination Data 클래스의 객체를 생성한다.
⑦ 해당 정보 객체를 B+ Tree에 삽입하고, 다시 ④로 이동하여 반복한다.

[B+ Tree Insert 과정]



① B+ 트리의 루트를 확인한다.
  - 루트가 빈 경우, 새로운 데이터 노드를 생성하여 루트에 저장한 후 모듈을 종료한다.
② 루트가 비지 않은 경우, B+ Tree에 이미 해당 데이터가 존재하는지 탐색한다.
  - 동일 데이터가 존재할 경우, 모듈을 종료한다.
③ 삽입할 데이터의 접종자 이름을 통해 키값을 비교하여 삽입할 데이터 노드 위치를 탐색한다.
④ 탐색 완료된 위치의 데이터 노드에 새로운 데이터를 삽입한다.
⑤ 삽입한 데이터 노드의 위치부터 루트까지 올라가며 노드의 차수를 확인하고, split을 수행한다.

[데이터 노드의 split 과정]

① 새로운 우측 데이터 노드(rightDataNode)를 생성한다.
- split하는 데이터 노드(pDataNode)의 우측 2개의 데이터를 rightDataNode에 저장한다.
- pDataNode에서 우측 2개의 데이터를 삭제한다.
- pDataNode는 split 과정에서 왼쪽 데이터 노드 역할을 수행한다.

② 부모 노드로 rightDataNode의 데이터를 삽입하고, 부모와 자식 노드를 연결한다.

Case 1 : pDataNode가 B+ Tree의 루트 노드인 경우
- 새로운 부모 노드(newIndexNode)를 생성한다.
- newIndexNode에 rightDataNode의 첫 번째 데이터를 삽입한다.
- pDataNode를 newIndexNode의 mostLeftChild로 연결한다.
- newIndexNode와 pDataNode, rightDataNode를 부모와 자식 노드 관계로 연결한다.
- newIndexNode를 B+ Tree의 루트 노드로 설정한다.

Case 2 : pDataNode가 B+ Tree의 루트 노드가 아닌 경우
- pDataNode의 부모 노드에 rightDataNode의 첫 번째 데이터를 삽입한다.
- rightDataNode와 pDataNode의 부모 노드를 부모와 자식 노드 관계로 연결한다.

③ 데이터 노드들 사이를 이중 연결리스트로 연결한다.
- pDataNode의 우측으로 연결된 데이터 노드가 존재할 경우, rightDataNode와 pDataNode의 우측 데이터 노드를 연결한다.
- pDataNode와 rightDataNode를 연결한다.

[인덱스 노드의 split 과정]

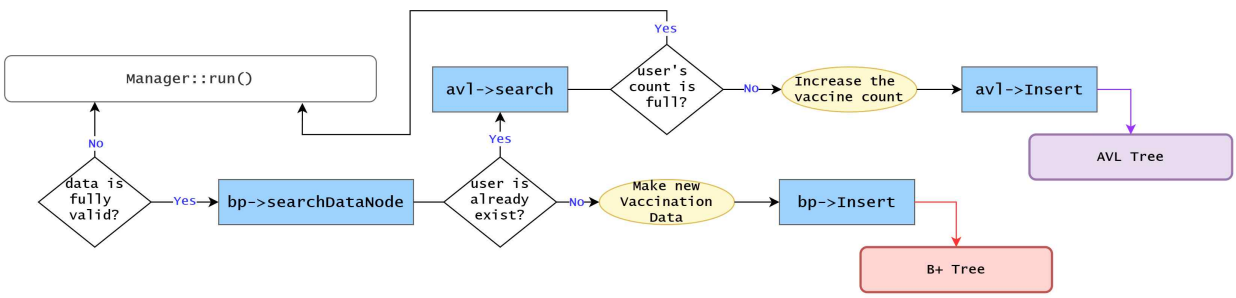
① 새로운 우측 인덱스 노드(rightIndexNode)를 생성한다.
- split하는 인덱스 노드(pIndexNode)의 우측 1개의 데이터를 rightIndexNode에 저장한다.
- pIndexNode에서 우측 2개의 데이터를 삭제한다.
- pIndexNode는 split 과정에서 왼쪽 인덱스 노드 역할을 수행한다.

② 부모 노드로 rightIndexNode의 데이터를 삽입하고, 부모와 자식 노드를 연결한다.

Case 1 : pIndexNode가 B+ Tree의 루트 노드인 경우
- 새로운 루트 노드(newRootNode)를 생성한다.
- newRootNode에 rightIndexNode의 첫 번째 데이터를 삽입한다.
- pIndexNode를 newRootNode의 mostLeftChild로 연결한다.
- newRootNode와 pIndexNode, rightIndexNode를 부모와 자식 노드 관계로 연결한다.
- newRootNode를 B+ Tree의 루트 노드로 설정한다.

Case 2 : pIndexNode가 B+ Tree의 루트 노드가 아닌 경우
- pIndexNode의 부모 노드에 rightIndexNode의 첫 번째 데이터를 삽입한다.
- rightIndexNode와 pIndexNode의 부모 노드를 부모와 자식 노드 관계로 연결한다.

2. ADD



① <command.txt> 파일에서 “ADD”와 함께 읽어온 데이터가 모두 유효한지 확인한다.

- 접종자의 이름, 백신 명, 접종자의 나이, 거주지역에 대해 유효하지 않을 경우, 모듈을 종료한다.

② 기존 B+ Tree에 기존 데이터가 존재하는지 탐색한다.
- 존재하지 않을 경우, 새로운 데이터를 B+ Tree에 삽입한다.

③ 기존 데이터가 존재하는 경우, AVL Tree를 탐색한다.
- AVL Tree에 존재하는 경우, 접종을 완료한 사람이므로 모듈을 종료한다.

④ AVL Tree에 존재하지 않는 경우, 기존 데이터의 접종 횟수를 업데이트한다.

⑤ 기존의 접종자를 접종 완료 상태로 판별하고, AVL Tree에 삽입한다.

[AVL Tree Insert 과정]

① AVL Tree의 루트 노드를 확인한다.
- 루트 노드가 빈 경우, 새로운 루트 노드를 생성하여 접종 완료자의 데이터를 저장한다.

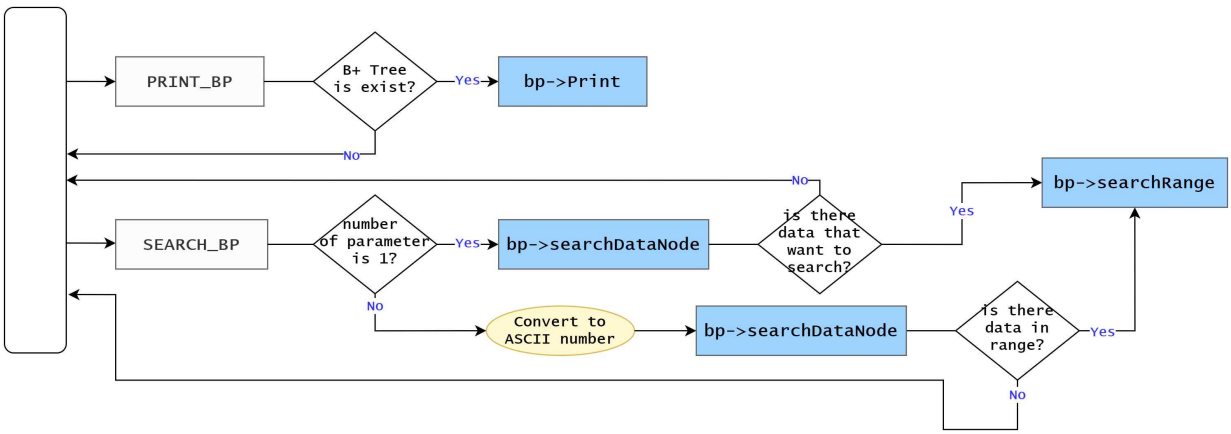
② AVL Tree의 루트 노드가 존재하는 경우, 데이터의 삽입 위치(p)를 탐색한다.
- AVL Tree에 존재하는 기존 데이터의 접종자 이름과 비교하며, 삽입 위치를 탐색한다.
- 삽입 시, Balance Factor가 2 또는 -2가 될 수 있는 노드의 위치(a)를 저장한다.

③ 새로운 데이터를 삽입하고, 삽입으로 인한 Balance Factor를 업데이트한다.

④ pp부터 a까지의 경로에서 Balance Factor를 확인하고, 필요한 경우 rotation을 수행한다.

⑤ Rotation을 수행한 서브 트리를 기존의 AVL Tree에 업데이트한다.

3. PRINT_BP & SEARCH_BP



[B+ Tree Print 과정]

① B+ Tree가 존재하지 않는 경우, 모듈을 종료한다.

② B+ Tree가 존재하는 경우, 이중 연결리스트로 연결된 모든 데이터 노드에 접근하여 출력한다.

[B+ Tree Search 과정]

① 입력받은 인자의 개수를 확인한다.

Case 1 : 인자의 개수가 1개인 경우

② B+ Tree에 인자로 입력받은 접종자 이름의 데이터가 존재하는지 탐색한다.
- 존재하지 않을 경우, 모듈을 종료한다.

③ 존재할 경우, B+ Tree의 Exact match 탐색을 수행한다.

Case 2 : 인자의 개수가 2개인 경우

② 입력받은 2개의 인자(각각 1개의 알파벳)를 ASCII 숫자로 변환하고, 인자 사이의 모든 알파벳으로 시작하는 접종자의 이름의 데이터가 존재하는지 탐색한다.
- 하나라도 존재하지 않을 경우, 모듈을 종료한다.

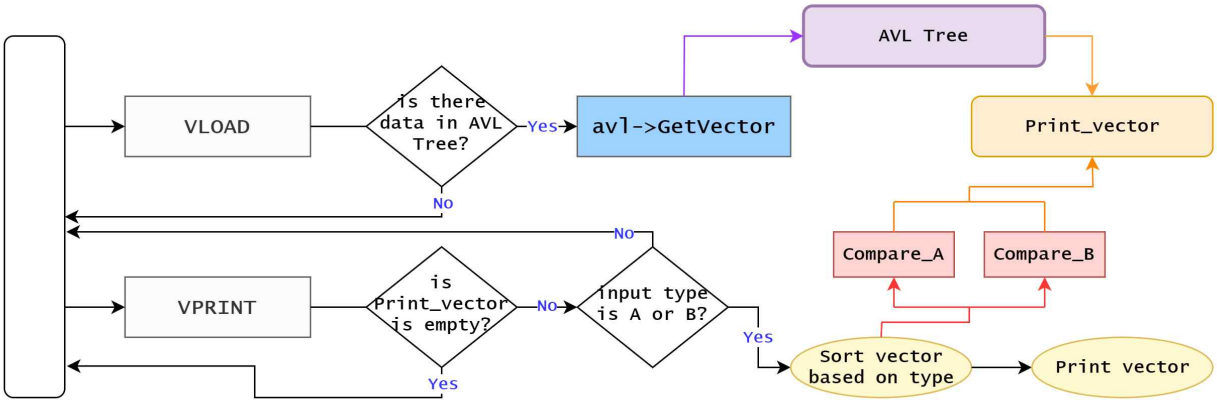
③ 존재할 경우, B+ Tree의 Range 탐색을 수행한다.

[B+ Tree의 Exact match & Range 탐색]

BpTree::SearchRange(string start, string end) 함수로 전달받은 인자에 따라 다른 탐색 방식을 수행

- 인자가 1개(start)인 경우 : 해당 함수는 Exact match 탐색 방식을 수행한다.
- 인자가 2개(start, end)인 경우 : 해당 함수는 Range 탐색 방식을 수행한다.

4. VLOAD & VPRINT



[VLOAD 과정]

① AVL Tree에 기존의 데이터가 존재하는지 확인한다.
  - 기존의 데이터가 존재하지 않을 경우, 모듈을 종료한다.

② 존재하는 경우, AVL Tree의 데이터를 통해 Print_vector를 구축한다.

[Print_vector 구축 과정]

① 벡터 안에 이미 데이터가 존재하는지 확인한다.
  - 벡터 안에 데이터가 존재하는 경우, 기존의 벡터를 초기화한다.

② AVL Tree 노드를 저장할 큐를 생성한다.

③ 큐에 AVL Tree의 가장 상단의 노드 1개를 읽어와 push한다.
  - 읽어올 AVL Tree의 노드가 없는 경우, 모듈을 종료한다.

④ 큐의 가장 앞에 위치한 노드의 데이터(Vaccination Data*)를 벡터에 저장한다.
  - 해당 노드를 큐에서 pop한다.
  - 해당 노드의 왼쪽 자식 노드와 오른쪽 자식 노드를 순차적으로 큐에 push한다.
  - ④의 과정을 지속적으로 반복한다. 읽어올 AVL Tree 노드가 없는 경우, 모듈을 종료한다.

[VPRINT 과정]

① 벡터 안에 데이터가 존재하는지 확인한다.
  - 벡터가 비어있을 경우, 모듈을 종료한다.

② 인자로 입력받은 벡터 출력 타입을 확인한다.
  - 입력받은 인자가 A 또는 B가 아닐 경우, 모듈을 종료한다.

③ 입력받은 인자의 타입에 해당하는 Compare 함수를 통해 벡터를 정렬한다.

④ 벡터의 데이터를 모두 출력한다.

IV. Result Screen

▽ 테스트에 사용한 <input_data.txt> 파일과 <command.txt> 파일의 모습

```
Denny Pfizer 0 32 Gyeonggi
Tom Pfizer 1 38 Seoul
Emily Moderna 0 21 Incheon
John Pfizer 1 17 Seoul
Erin AstraZeneca 0 51 Daegu
Carrick Moderna 0 32 Suwon
Lukas Pfizer 0 23 Busan
Mendes AstraZeneca 0 39 Gyeonggi
Jenny Janssen 0 36 Seoul
Sam Pfizer 0 47 Seoul
Daniel Moderna 0 26 Incheon
Kenny Pfizer 0 33 Seoul
Wayne AstraZeneca 1 55 Seoul
Amy Pfizer 1 38 Gyeonggi
Nicole Moderna 1 34 Daegu
David AstraZeneca 1 43 Seoul
Kevin Janssen 0 20 Busan
```

```
LOAD
PRINT_BP
ADD Elsa Janssen 49 Busan
ADD Tommy Pfizer 38 Seoul
PRINT_BP
SEARCH_BP Tom
SEARCH_BP Q Z
ADD Tom Pfizer 38 Seoul
ADD Jenny Janssen 36 Seoul
ADD Kevin Janssen 20 Busan
ADD Jenny Janssen 36 Seoul
ADD John Pfizer 17 Seoul
ADD Amy Pfizer 38 Gyeonggi
ADD David AstraZeneca 43 Seoul
PRINT_BP
VLOAD
SEARCH_AVL Tom
SEARCH_AVL Jenny
SEARCH_AVL Elsa
ADD Nicole Moderna 34 Daegu
VLOAD
VPRINT A
VPRINT B
LOAD
SEARCH_BP a z
SEARCH_BP A D
EXIT
```

[1] LOAD

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

<input_data.txt> 파일의 데이터를 읽어와 B+ Tree를 구축한다.

[2] PRINT_BP

```
======== PRINT_BP =======
Amy Pfizer 1 38 Gyeonggi
Carrick Moderna 0 32 Suwon
Daniel Moderna 0 26 Incheon
David AstraZeneca 1 43 Seoul
Denny Pfizer 0 32 Gyeonggi
Emily Moderna 0 21 Incheon
Erin AstraZeneca 0 51 Daegu
Jenny Janssen 0 36 Seoul
John Pfizer 1 17 Seoul
Kenny Pfizer 0 33 Seoul
Kevin Janssen 0 20 Busan
Lukas Pfizer 0 23 Busan
Mendes AstraZeneca 0 39 Gyeonggi
Nicole Moderna 1 34 Daegu
Sam Pfizer 0 47 Seoul
Tom Pfizer 1 38 Seoul
Wayne AstraZeneca 1 55 Seoul
=========================
```

구축한 B+ Tree를 출력한다.

[3] ADD Elsa Janssen 49 Busan
[4] ADD Tommy Pfizer 38 Seoul

```
====== ADD ======      ====== ADD ======
Elsa Janssen 49 Busan  Tommy Pfizer 38 Seoul
=================      =================
```

직접 B+ Tree에 데이터를 추가한다.

[5] PRINT_BP

```
======== PRINT_BP =======
Amy Pfizer 1 38 Gyeonggi
Carrick Moderna 0 32 Suwon
Daniel Moderna 0 26 Incheon
David AstraZeneca 1 43 Seoul
Denny Pfizer 0 32 Gyeonggi
Elsa Janssen 0 49 Busan
Emily Moderna 0 21 Incheon
Erin AstraZeneca 0 51 Daegu
Jenny Janssen 0 36 Seoul
John Pfizer 1 17 Seoul
Kenny Pfizer 0 33 Seoul
Kevin Janssen 0 20 Busan
Lukas Pfizer 0 23 Busan
Mendes AstraZeneca 0 39 Gyeonggi
Nicole Moderna 1 34 Daegu
Sam Pfizer 0 47 Seoul
Tom Pfizer 1 38 Seoul
Tommy Pfizer 0 38 Seoul
Wayne AstraZeneca 1 55 Seoul
==========================
```

B+ Tree를 다시 출력했을 때, Elsa와 Tommy에 대한 데이터가 추가된 것을 확인할 수 있다.

[6] SEARCH_BP Tom

```
====== SEARCH_BP ======
Tom Pfizer 1 38 Seoul
=======================
```

Exact match 탐색 방식으로 B+ Tree에 존재하는 Tom에 대한 데이터를 출력한다.

[7] SEARCH_BP Q Z

```
====== SEARCH_BP ======
Sam Pfizer 0 47 Seoul
Tom Pfizer 1 38 Seoul
Tommy Pfizer 0 38 Seoul
Wayne AstraZeneca 1 55 Seoul
======================
```

Range 탐색 방식으로 B+ Tree에 존재하는 데이터 중에서 Q에서 Z 사이 알파벳으로 시작하는 데이터를 모두 출력한다.

[8] ADD Tom Pfizer 38 Seoul
[9] ADD Jenny Janssen 36 Seoul
[10] ADD Kevin Janssen 20 Busan
[11] ADD Jenny Janssen 36 Seoul
[12] ADD John Pfizer 17 Seoul
[13] ADD Amy Pfizer 38 Gyeonggi
[14] ADD David AstraZeneca 43 Seoul

```
====== ADD ======
Tom Pfizer 38 Seoul
=================

====== ADD ======
Jenny Janssen 36 Seoul
=================

====== ADD ======
Kevin Janssen 20 Busan
=================

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

====== ADD ======
John Pfizer 17 Seoul
=================

====== ADD ======
Amy Pfizer 38 Gyeonggi
=================

====== ADD ======
David AstraZeneca 43 Seoul
=================
```

B+ Tree에 데이터를 추가한다. 11번의 명령어에서 Jenny의 접종 횟수가 Janssen의 완료 횟수인 1회에 도달했으므로 수행되지 않는다.

[15] PRINT_BP

```
======== PRINT_BP =======
Amy Pfizer 1 38 Gyeonggi
Carrick Moderna 0 32 Suwon
Daniel Moderna 0 26 Incheon
David AstraZeneca 1 43 Seoul
Denny Pfizer 0 32 Gyeonggi
Elsa Janssen 0 49 Busan
Emily Moderna 0 21 Incheon
Erin AstraZeneca 0 51 Daegu
Jenny Janssen 0 36 Seoul
John Pfizer 1 17 Seoul
Kenny Pfizer 0 33 Seoul
Kevin Janssen 0 20 Busan
Lukas Pfizer 0 23 Busan
Mendes AstraZeneca 0 39 Gyeonggi
Nicole Moderna 1 34 Daegu
Sam Pfizer 0 47 Seoul
Tom Pfizer 1 38 Seoul
Tommy Pfizer 0 38 Seoul
Wayne AstraZeneca 1 55 Seoul
==========================
```

삽입 이전 → 삽입 이후

```
======== PRINT_BP =======
Amy Pfizer 2 38 Gyeonggi
Carrick Moderna 0 32 Suwon
Daniel Moderna 0 26 Incheon
David AstraZeneca 2 43 Seoul
Denny Pfizer 0 32 Gyeonggi
Elsa Janssen 0 49 Busan
Emily Moderna 0 21 Incheon
Erin AstraZeneca 0 51 Daegu
Jenny Janssen 1 36 Seoul
John Pfizer 2 17 Seoul
Kenny Pfizer 0 33 Seoul
Kevin Janssen 1 20 Busan
Lukas Pfizer 0 23 Busan
Mendes AstraZeneca 0 39 Gyeonggi
Nicole Moderna 1 34 Daegu
Sam Pfizer 0 47 Seoul
Tom Pfizer 2 38 Seoul
Tommy Pfizer 0 38 Seoul
Wayne AstraZeneca 1 55 Seoul
==========================
```

8 - 14번의 ADD 명령어로 Amy, David, Jenny, John, Kevin, Tom의 접종 횟수가 증가한 것을 확인할 수 있다.

[16] VLOAD

```
======= VLOAD =======
Success
====================
```

AVL Tree에 존재하는 모든 접종 완료자의 데이터를 벡터에 저장한다.

[17] SEARCH_AVL Tom
[18] SEARCH_AVL Jenny
[19] SEARCH_AVL Elsa

```
====== SEARCH_AVL ======
Tom Pfizer 2 38 Seoul
========================

====== SEARCH_AVL ======
Jenny Janssen 1 36 Seoul
========================

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

Tom은 Pfizer를 2회 접종받았고, Jenny는 Janssen을 1회 접종받았으므로 접종 완료자로 분류되어 AVL Tree에 존재한다. 하지만 Elsa는 Janssen을 0회 접종받았으므로, AVL Tree에 존재하지 않는다.

[20] ADD Nicole Moderna 34 Daegu
[21] VLOAD

```
====== ADD ======
Nicole Moderna 34 Daegu
=================

======= VLOAD =======
Success
====================
```

Nicole에 대한 데이터를 B+ Tree에 추가하였고, 벡터를 다시 생성한다.
이로써 Nicole은 Moderna를 2회 접종받았으므로 AVL Tree에 존재하게 된다.

[22] VPRINT A
[23] VPRINT B

```
======= VPRINT A =======
David AstraZeneca 2 43 Seoul
Kevin Janssen 1 20 Busan
Jenny Janssen 1 36 Seoul
Nicole Moderna 2 34 Daegu
John Pfizer 2 17 Seoul
Amy Pfizer 2 38 Gyeonggi
Tom Pfizer 2 38 Seoul
==========================

======= VPRINT B =======
Kevin Janssen 1 20 Busan
Nicole Moderna 2 34 Daegu
Amy Pfizer 2 38 Gyeonggi
David AstraZeneca 2 43 Seoul
Tom Pfizer 2 38 Seoul
Jenny Janssen 1 36 Seoul
John Pfizer 2 17 Seoul
==========================
```

생성된 벡터를 A 타입과 B 타입으로 출력한다.
A 타입의 경우, 백신 명 오름차순, 나이 오름차순, 접종자의 이름 오름차순으로 정렬된다.
B 타입의 경우, 지역명 오름차순, 나이 내림차순, 접종자의 이름 오름차순으로 정렬된다.

[24] LOAD

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

<input_data.txt> 파일로부터 B+ Tree를 구축한다.
기존에 구축된 B+ Tree가 이미 존재하기 때문에, 정상적으로 실행되지 않는다.

[25] SEARCH_BP a z
[26] SEARCH_BP A D

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

====== SEARCH_BP ======
Amy Pfizer 2 38 Gyeonggi
Carrick Moderna 0 32 Suwon
Daniel Moderna 0 26 Incheon
David AstraZeneca 2 43 Seoul
Denny Pfizer 0 32 Gyeonggi
========================
```

B+ Tree에 대해 range 탐색을 수행한다. 25번째 명령어의 경우, a에서 z사이의 알파벳으로 시작하는 이름의 접종자가 존재하지 않기 때문에 정상적으로 실행되지 않는다.

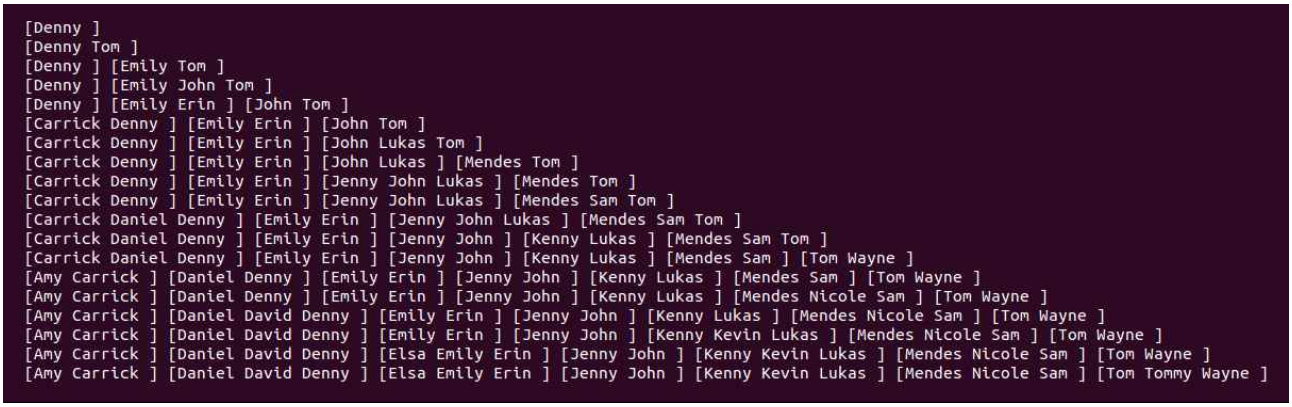
[27] EXIT

```
======= EXIT =======
Success
====================
```

할당받은 자원을 반납하고 프로그램을 종료한다.

Ⅴ. Consideration

이번 프로젝트를 통해서 코로나 19 예방접종 관리 프로그램을 만들면서 B+ Tree와 AVL Tree를 구성하는 방법에 대해 익힐 수 있었고, 상속과 가상 함수의 개념을 익힐 수 있었다. 그리고 벡터를 생성하며 벡터를 원하는 조건에 의해 출력하는 방법에 대하여 익힐 수 있었다.
이번 프로젝트를 진행하면서 가장 많은 시간이 소요되었던 부분은 B+ Tree를 만드는 부분이었다.
처음으로 B+ Tree의 구성을 완료하고 프로그램을 테스트했을 때, 정상적으로 결과가 나와서 B+ Tree를 잘 구성한 것으로 판단하였다. 하지만 B+ Tree의 각 노드들을 순회하면서 모든 데이터 노드들을 출력했을 때, 정상적으로 구성된 것이 아님을 확인할 수 있었다.
이러한 어려움을 겪었던 주된 이유는 map 컨테이너 구조에 대한 미숙함에서 비롯되었다. 첫 번째 프로젝트를 통해 구성했던 이진 탐색 트리는 해당 노드에서 왼쪽 자식 노드와 오른쪽 자식 노드에 대한 포인터를 직관적으로 가지고 있었다, 하지만 이번 프로젝트의 B+ Tree는 map 컨테이너를 활용하여 아래의 노드들을 저장한다는 것, 그리고 인덱스 노드와 데이터 노드들을 구별하여 진행해야 하는 것이 프로젝트를 진행하면서 어려운 부분이었다. 강의를 통해 배운 내용과 학습 자료들을 숙지하면서 B+ Tree의 구조와 특징, 그리고 데이터 노드와 인덱스 노드에 따른 구별되는 split 과정을 정확히 숙지한 이후에 프로젝트를 다시 진행해나갔다.
이번 프로젝트를 통해 구성한 B+ Tree의 데이터 노드들의 모습은 아래와 같다.



각 라인은 LOAD 명령을 수행하면서 input_data.txt 파일의 데이터를 읽어와 새로운 데이터를 하나씩 추가하였을 때의 데이터 노드들의 모습을 의미한다. 그리고 “[ ]”로 묶인 부분은 하나의 데이터 노드가 가지고 있는 데이터 맵의 모습을 의미한다. 이렇게 출력하여 확인함으로써 정상적으로 B+ Tree를 구성한 것을 확인할 수 있었다.

이번 프로젝트를 진행하면서 B+ Tree와 AVL Tree가 가지는 자료구조 적인 특징을 생각해볼 수 있었다. B+ Tree의 경우, 인덱스 노드와 데이터 노드를 별도로 구성하고, 인덱스 노드는 킷값을 가지고 데이터 노드는 실질적인 데이터들을 가지고 있다. 또한, 데이터 노드들이 모두 이중 연결리스트로 연결되어 있기 때문에 B+ Tree를 탐색하거나 출력하는 과정을 훨씬 단순화하고, 성능을 높일 수 있는 장점을 지닐 수 있었다. 또한, AVL Tree의 경우, 각 노드들이 Balance Factor라는 값을 가지고 있고, 이를 고려하여 트리의 모양을 지속적으로 변환하기 때문에 트리의 높이를 최소화시킬 수 있었다. B+ Tree와 마찬가지로 이를 통해서 트리의 높이에 영향을 받는 트리 탐색 과정을 단순화하고, 성능을 높일 수 있을 것이다.

이번 프로젝트를 진행하면서 이전에 많이 사용하지 않았던 map 컨테이너를 활용하는 기회 또한 충분히 가질 수 있었다. STL map의 기능과 특징들을 익힐 수 있었고 특히, 자동으로 컨테이너에 저장된 데이터들을 정렬하여 관리해주기 때문에 B+ Tree의 구성을 더욱 효율적으로 만들어주었다.

이번 프로젝트에서는 지난 프로젝트에서 개선해야 할 부분이라고 생각했던 상위 모듈에서 하위 모듈로 이어지는 설계에서의 미숙함과 코드의 간결성, 효율성 측면의 부족함을 보완하고자 생각하였다. 상위 모듈에서 구현해야 할 기능들을 숙지한 이후에 각 B+ Tree와 AVL Tree의 하위 모듈을 구현하는 방향으로 진행하였고, 지난 프로젝트보다는 이러한 부분에서 조금이나마 개선된 과정과 결과물을 얻을 수 있어 좋은 경험이 되었다.