데이터 구조 실습

Project 2

학　　과: 컴퓨터정보공학부

학　　번: 2018202073

성　　명: 기현성

Assigment_1

1) Introduction

이번 프로젝트는 B+ tree, AVL tree. STL vector를 이용하여 코로나 19 예방접종 관리 프로그램을 구현하는 것을 목적으로 하고 있다. 예방접종 관리 프로그램은 이름, 백신 명 , 접종 횟수, 나이, 지역명을 관리하며, 이를 이용하여 접종 대상자 및 접종 완료자에 대한 정보를 제공할 수 있다. B+ tree를 이용하여 접종 대상자에 대해 관리하며, AVL tree를 이용하여 접종 완료자를 관리한다. 이를 Print_vector를 이용하여 접종 완료자를 사전에 정의한 정렬 방법에 따라 출력할 수 있다.

VaccinationData

저장된 파일에서 이름, 백신 명, 접종 횟수, 나이, 지역명을 읽어 클래스에 저장한다. Janssen은 접종 횟수가 1이면 접종 완료자고, 나머지는 접종 횟수가 2회면 접종 완료자다.

B+ tree

차수는 3으로 구현하며, input_data에서 불러온 접종 대상자의 VaccinationData을 map 형태로 저장하며, ADD로 데이터가 존재하면 접종 횟수만 증가시키고, 없으면 추가한다. 이때, 접종 완료자에 대해서는 예외처리를 시행한다.

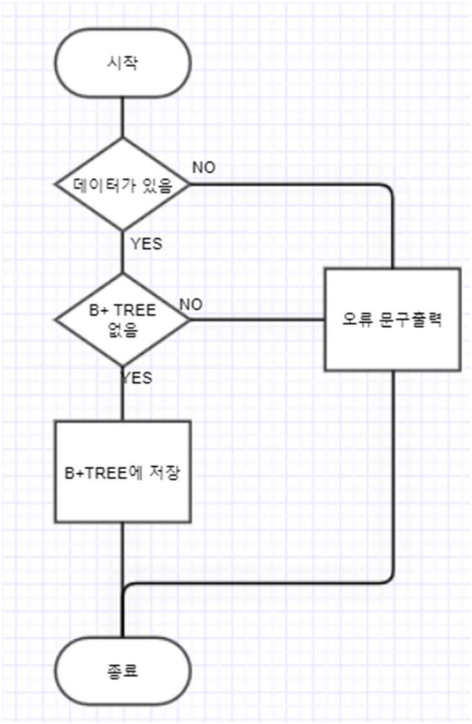
AVL tree

B+ tree에 저장된 데이터가 ADD로 인해 접종 횟수가 증가해 접종 완료자가 되면 그 정보는 AVL Tree에 저장한다. AVL tree에 VacciniationData로 저장하며, balance factor로 균형을 유지한다.

Print_vector

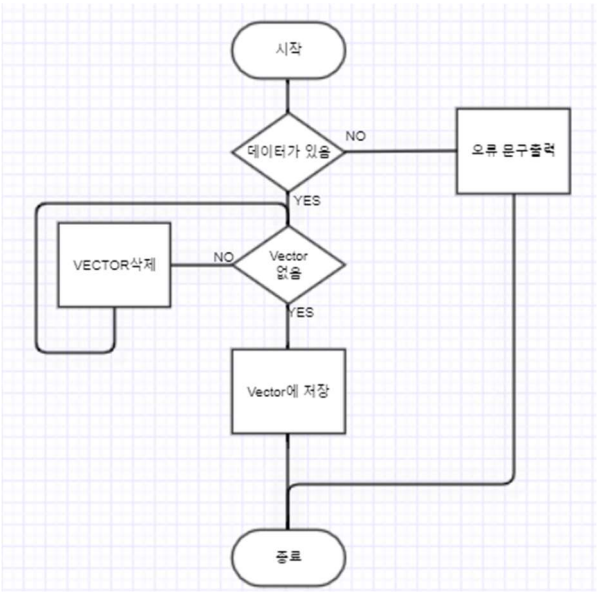
VLOAD를 입력할시 현재 AVL에 저장된 정보를 vector에 저장해준다. VPRINT를 입력해 저장된 정보를 원하는 방식으로 출력을 진행한다. 이때, avl tree에 정보가 없으면 예외처리를 진행한다.

2) Flowchart



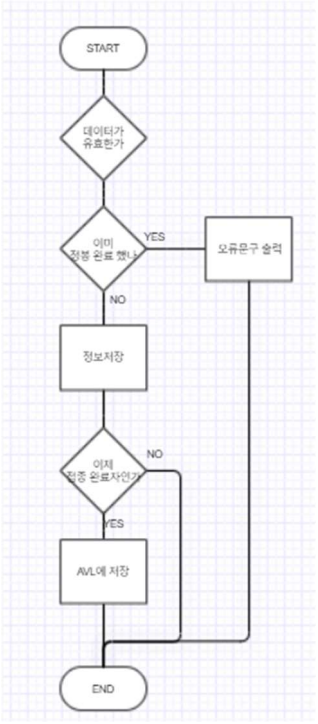
LOAD

데이터가 없거나, 파일이 존재하지 않으면 오류 문구를 출력하고, 이미 B+TREE가 있으면 오류 문구를 출력한다.



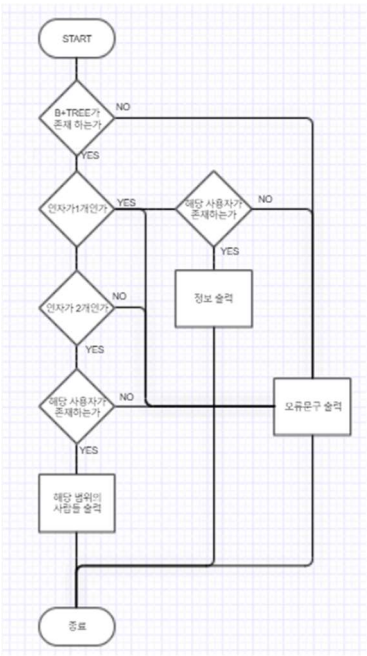
VLOAD

AVL TREE가 없으면 오류 문구 출력 후 종료, VECTOR가 있으면 삭제 후 VECTOR에 저장해주고, VECTOR이 없어도 VECTOR에 정보를 저장한다.

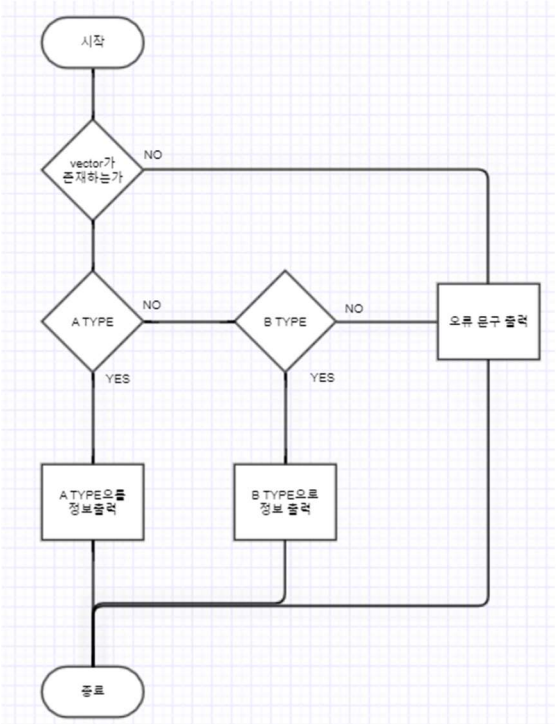


ADD

B+TREE에 정보를 추가하기 위한 명령어로 4개의 인자를 입력 받는다. 입력한 정보가 B+ TREE에 있으면 해당 사용자의 접종 횟수를 1 증가시키고, 만약 방금 추가해서 완료가 되었으면 AVL TREE에 삽입해준다. 만약 이미 접종 완료자인 경우에는 오류문구를 출력한다.



SEARCH_BP 명령어의 인자로 이름을 입력하는 경우에는 B+TREE에 저장된 정보 중 입력한 이름의 정보만을 출력한다. 두 알파벳을 입력한 경우 해당 이니셜 문자의 범위에 해당하는 정보를 출력한다. 이때, 해당하는 정보가 없거나, B+TREE에 정보가 없거나, 잘못된 인자를 입력했을 경우, 오류 문구를 출력한다.



VPRINT

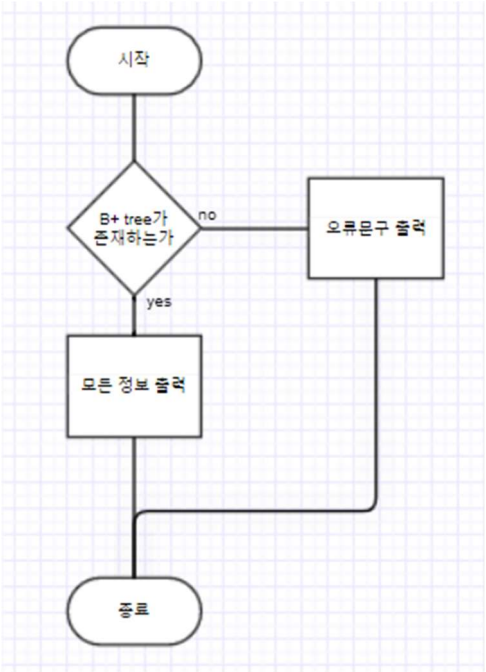
Print_vector에 저장된 데이터를 조건 A, B에 따라 정렬을 하고 출력한다.

조건 A는 백신명 오름차순, 같을 경우 나이 오름차순, 같을 경우 이름 오름차순

조건 B는 지역명 오름차순 같을 경우 나이 내림차순, 같을 경우 이름 오름차순이다.

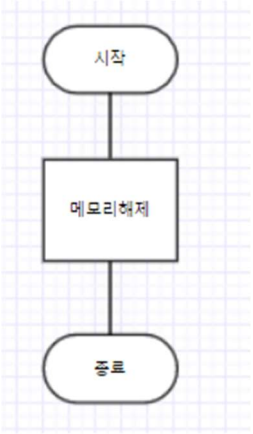
이때 조건은 algorithm의 sort를 이용하여 구현한다.

만약 잘못된 인자를 입력했거나 vector이 존재하지 않으면 오류 문구를 출력한다.



PRINT_BP

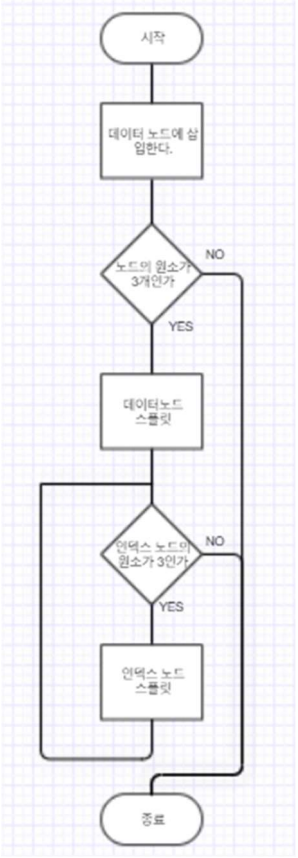
모든 정보를 이름 순서대로 출력하는 메소드로 b+tree가 존재하지 않으면 오류문구를 출력한다.



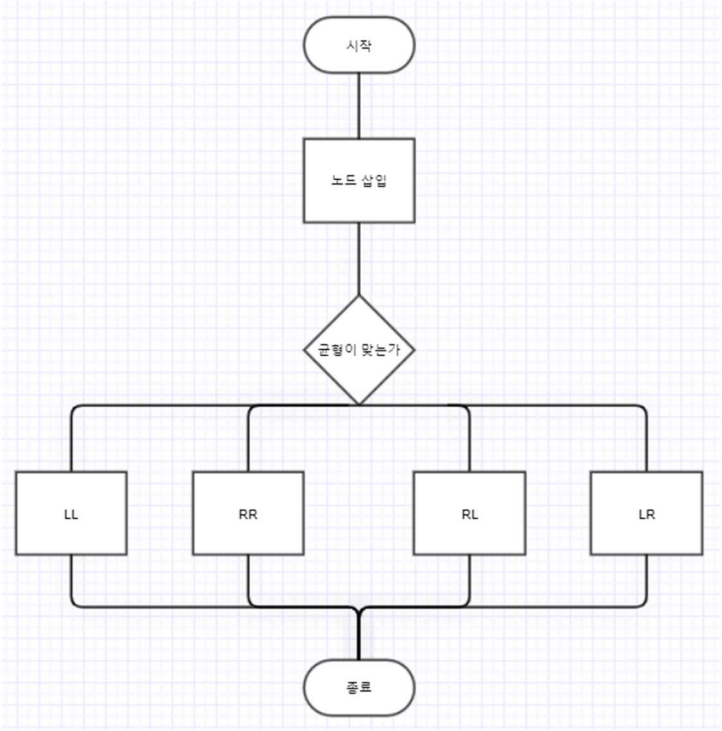
EXIT

프로그램 상의 메모리를 해제하며, 프로그램을 종료한다.

<B+TREE>



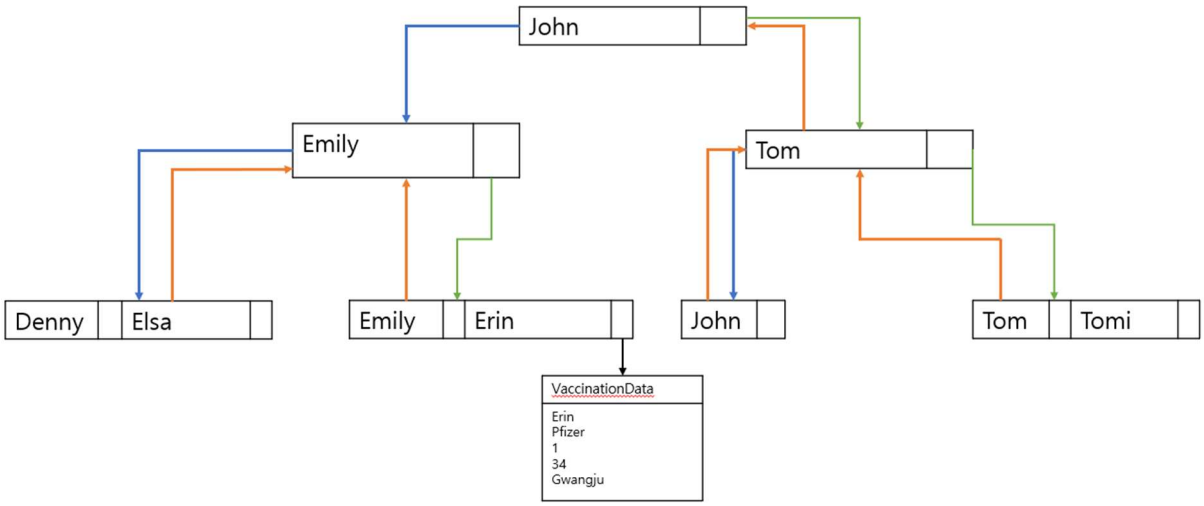
B+TREE에 삽입하고 더 이상 스플릿 되지 않을 때까지 반복한다.



노드를 삽입받고 밸런스팩터에 따라 LL,RR,LR,RL을 진행한다.

3) Algorithm

<B+ Tree>



B+ Tree는 기본적으로 크게 internode와 external node로 구분할 수 있다.

우선 삽입될 데이터 노드의 위치를 찾는 방법은 루트부터 시작해 노드의 맵의 첫 원소의 이름보다 사전 순으로 앞서면 해당 노드의 MostLeftChild로 이동한다. 다음 원소보다 크면 해당 원소의 second로 이동하고, 작으면 이전 원소의 second로 이동한다.

Tree의 root가 없을 때는 terminal node에 넣고 root로 만든다. 차수가 3이므로 3개를 넣었을 때, split 해준다. split 하는 방법에는 두 가지가 있다.

부모가 없을 때)

새 노드의 MLC를 가운데 원소의 second로 한다. 새 부모에 새 노드를 넣고 가운데 원소를 삭제한다.
새 부모의 MLC를 대상 노드로 하고, 대상 노드와 새 노드의 부모를 새 부모로 만든다.
부모가 있을 때)
새 노드의 MLC를 가운데 노드의 second로 한다. 대상 노드의 부모에 새 노드를 삽입한다. iterator를 삭제하고, 새 노드의 부모를 대상 노드의 부모로 정한다.

- INSERT
: 루트가 없으면 새로운 노드를 생성하고 입력받은 정보를 넣어준다. 그 후, 루트로 만든다.
정보의 중복 여부를 보고, 중복되면 중복 플래그를 true로 해주고 메서드를 종료한다. 중복 검사 과정에서 이미 접종 완료자인 것이 확인되면 메서드를 종료한다. Insert가 ADD에 의해 호출되면 바로 접종 횟수를 바로 1 추가해준다. 이때, 백신이 얀센인 경우에는 접종 완료로 해준다. 노드를 비교해가며 새로운 정보가 들어갈 곳을 찾고 삽입해준다. 삽입 후 노드의 사이즈가 3이 되면, 분할해준다.

-exceeddatanode / exceedIndexNode
: 노드의 사이즈를 보고 3 이상이면 true를 반환한다.

-splitDataNode
: 첫 원소를 새로운 노드를 생성해 삽입한다. 만약 부모가 없으면, 새로운 루트 노드를 생성해 기존 노드를 삽입한다. 새로운 루트의 MLC를 새 노드로 한다. 새 노드의 다음 노드를 기존 노드로 설정하고 기존 노드의 이전 노드를 새 노드로 설정한다. 두 노드의 부모를 새 루트 노드로 설정한다. 부모가 있으면, 부모에 삽입하고, 현재 노드의 위치를 새 노드로 대체하고 부모를 지정한다. 기존 노드가 이전 노드가 있으면 이전 노드의 다음 노드를 새 노드로 설정하고 새 노드의 이전 노드를 기존 노드의 이전 노드로 설정한다. 그리고 공통으로, 기존 노드의 이전 노드를 새 노드로 설정하고 새 노드의 다음 노드를 기존 노드로 설정한다. 분할 후 부모의 사이즈가 3이 되면 분할해준다.

-splitIndexNode
: 기존 노드의 첫 원소를 새 노드에 삽입하고 자신의 자식과 노드를 이어준다. 기존 노드의 MLC의 부모를 새 노드로 지정한다. 기존 노드가 부모가 없음. 즉 루트라면, 새로운 루트 노드를 생성해 기존 노드를 삽입하고 새 노드를 MLC로 지정한다. 기존 노드의 MLC를 중간 원소의 자식으로 한다.
기존 노드와 새 노드의 부모를 새 루트로 지정한다.
부모가 있다면, 기존 노드의 MLC를 중간원소의 자식으로 한다. 현재 노드의 위치를 새 노드로 대체하고 부모를 지정한다. 분할을 재귀적으로 반복한다.
-SearchRange
: 인자가 한 개일 때)
데이터 노드까지 내려가 같은 이름에 해당하는 VaccinationData를 찾아 출력한다.

인자가 두 개일 때)
데이터 노드까지 내려가 VaccinationData의 이름의 첫글자가 입력한 알파벳의 범위에 속하면

출력한다.

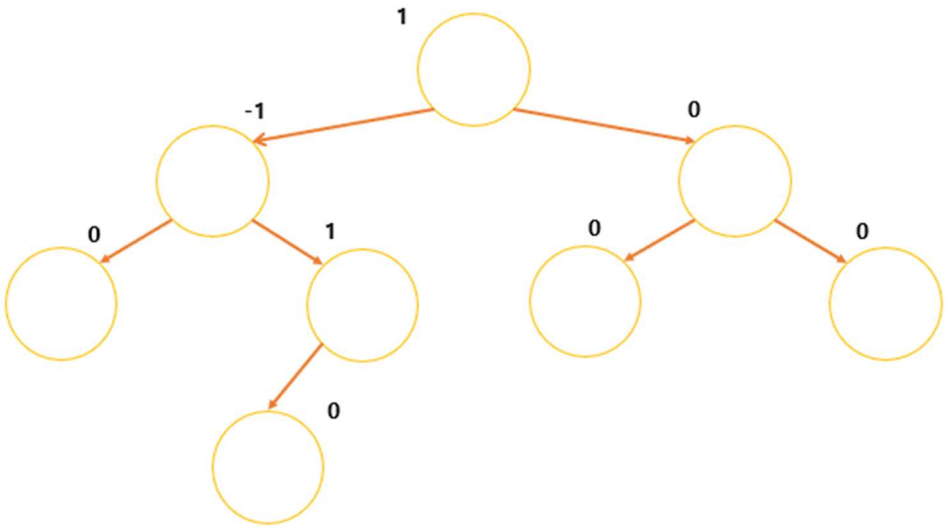
- Print
가장 왼쪽의 데이터 노드로 가서 노드의 next가 없을 때까지 각 노드의 맵과 노드를 이동하면서 출력한다.

dup
: 중복 여부를 판단하는 메서드로, 가장 왼쪽의 데이터 노드까지 내려가 각 노드의 맵의 원소와 하나씩 비교해가면서 해당 원소를 찾는다. 있으면 접종 횟수를 1 증가시키고 메서드를 종료한다. 이때, 이미 접종 완료되었으면 over 플래그를 true로 만들고 그냥 종료한다. 메서드로 인해 접종 횟수가 1 증가하였으면, done 플래그를 true로 만든다.

delete_all
: B+tree의 모든 노드를 삭제하는 메서드로, 루트부터 level order 순회 방식으로 진행한다. 우선 루트를 큐에 넣는다. 리스트의 프론트의 모든 자식을 큐에 넣고 프론트를 삭제시키고 프론트를 팝한다. 리스트의 프론트가 데이터 노드면 반복문을 탈출한다. 큐의 프론트의 맵에 있는 모든 원소를 삭제하고 프론트를 삭제한다. 그 후 팝 시킨다. 큐가 빌 때까지 반복한다.

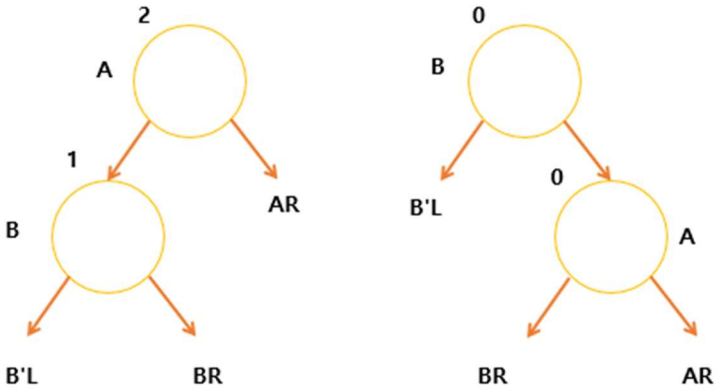
find_root
: 삽입될 위치를 찾는 메서드로, 루트부터 시작해서 맵의 첫 원소보다 작으면 MLC로 이동하고, 크거나 같으면, 다른 원소와 비교한다. 그 원소보다 크거나 같으면 그 원소의 second로 이동하고, 작으면 첫 노드의 second로 이동한다. 재귀적으로 반복해 datanode에 도달하면 해당 datanode를 반환한다.

<AVL Tree>



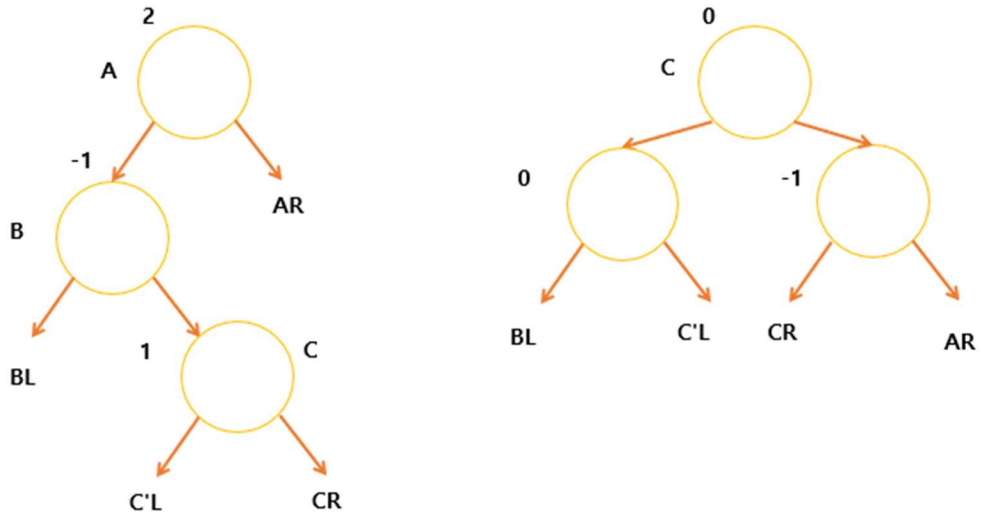
ADD 명령어로 B+ tree에 저장된 데이터가 접종 완료가 되었을 때, 해당 정보는 AVL에 삽입된다. AVL에 아무런 정보가 없다면 해당 노드를 root로 해준다.

AVL은 기본적으로 이진 트리이기때문에 root부터 시작해 크면 오른쪽 작으면 왼쪽으로 이동하면서 삽입될 위치를 찾는다. 하지만, BST와는 달리 BF, Balance Factor을 가지고 있어서, 그것을 이용하여 트리의 균형을 유지해줘야 한다.
Tree의 균형을 맞추는 방법이다.
LL)



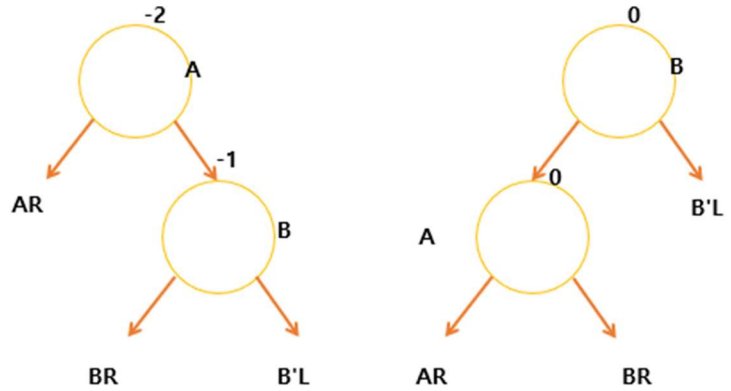
자신과 왼쪽 자식의 balance factor가 1일때 시행하며, 현재 노드의 왼쪽을 왼쪽 자식의 오른쪽으로 해주고, 왼쪽 자식의 오른쪽을 현재 노드로 해준다. 그 후, 현재 노드와 왼쪽 자식의 노드의 balance factor를 0으로 해주고 subroot를 왼쪽 자식으로 해준다.

LR)



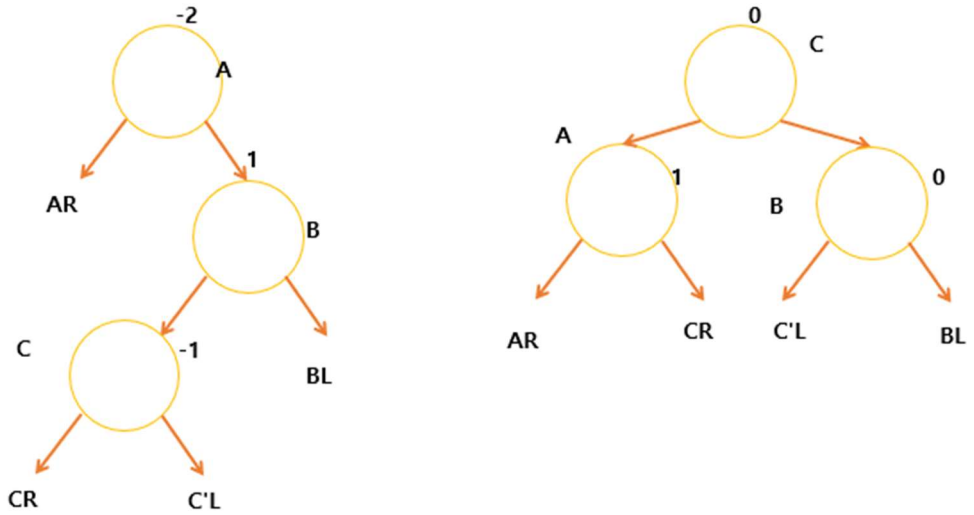
자신의 balance가 1이고 왼쪽 자식의 balance factor가 1이 아닐 때 시행한다. 오른쪽 노드를 왼쪽 노드의 오른쪽 노드로 지정해주고, 왼쪽 노드의 오른쪽 노드를 오른쪽 노드의 왼쪽 노드로 해준다. 현재 노드의 왼쪽 노드를 오른쪽 노드의 오른쪽 노드로 해주고, 오른쪽 노드의 왼쪽을 왼쪽 노드로, 오른쪽 노드의 오른쪽을 현재 노드로 지정해준다. 만약 오른쪽 노드의 bf가 0이면 현재 노드와 왼쪽 노드의 balance factor을 1로 해주고, 1이면 현재 노드의 bf를 -1로 하고 왼쪽 노드의 bf를 0으로 한다. -1이면 현재 노드의 bf를 0으로 해주고 왼쪽 노드의 bf를 1로 해준다. 그 후, 오른쪽 노드의 bf는 0으로 해준다.
간단히 말해서 LL후 RR 주면 된다.

RR}



현재 노드의 오른쪽을 오른쪽 자식의 왼쪽으로 해준다음. 오른쪽 자식의 왼쪽을 현재 노드로 한다. 그 후 현재 노드와 오른쪽 노드의 bf를 모두 0으로 해준다.

RL)



tmp2에 현재 노드의 오른쪽 노드의 왼쪽을 지정한다. 오른쪽 노드의 왼쪽을 tmp2의 오른쪽으로 한다. 현재 노드의 오른쪽을 tmp2의 왼쪽 노드로 한다. tmp2의 오른쪽을 현재 노드의 오른쪽 노드로 왼쪽 노드를 현재 노드로 한다. tmp2가 0이 아니면, 현재 노드와 tmp1의 bf는 0이다. tmp2의 bf가 1이면 cur의 bf를 0으로 tmp1을 -1로한다. tmp2의 bf가 -1이면 cur의 bf를 1로 tmp1의 bf를 0으로 한다. 그 후, tmp2의 bf를 0으로 지정하고 subroot를 tmp2로 한다.

모든 전환이 끝난 후, 부모가 없다면 root를 subroot로 하고, cur이 부모의 왼쪽이면 부모의 왼쪽을, 오른쪽이면 부모의 오른쪽을 subroot로 한다.

insert
: 접종완료자를 avl에 삽입하는 메소드다. 루트가 없으면 루트에 새 정보를 넣어준다.

있으면, 루트부터 시작해 작으면 왼쪽, 크면 오른쪽으로 이동하면서 삽입될 위치를 찾아서 삽입된다. 밸런스 팩터에 따라 LL,RR,LR,RL을 진행해서 균형을 맞춰준다.

-compare
: 모든 문자열을 소문자로 바꿔 비교한다. 같으면 0, 작으면 -1, 크면 1을 반환한다.

-getvector
Print_vector을 불러와 해당 벡터에 avl의 모든 정보를 삽입하는 메소드다. 벡터에 이미 정보가 있으면 모든 정보를 삭제하고 진행한다. 큐를 생성해 루트를 넣는다. 큐의 프론트를 벡터에 넣고 프론트의 자식을 모두 큐에 넣고 자신을 팝 한다. 큐가 빌 때까지 반복한다.

-Search
compare메서드를 이용해 루트부터 비교해가면서 찾는다. 작으면 왼쪽으로 이동하고 크면 오른쪽으로 이동하고, 같으면 해당 정보를 반환한다.

MANAGER
- FindSpace : 입력 받은 문자에서 띄어쓰기를 판별해 명령어의 개수를 판단한다. 만약 띄어쓰기가 연속으로 두개가 나온다면 오류를 반환한다.
-FindDataError : 입력 받은 인자가 오류가 있는지 판별하는 메서드로 다른 문자가 들어오면 오류문구를 출력한다.

4) 결과 화면

```
VLOAD
PRINT_BP
GG
LOAD
LOAD
PRINT_BP
SEARCH_BP Cartier
SEARCH_BP Wooyoungmi
ADD Anne Moderna seven Seoul
ADD Olaf Janssen 4 7 Dokdo
ADD Elsa Janssen 49 Busan
ADD Tommy Pfizer 38 Seoul
PRINT_BP
ADD John Pfizer 17 Seoul
ADD Emily Moderna 21 Incheon
ADD Emily Moderna 21 Incheon
ADD Emily Moderna 21 Incheon
SEARCH_AVL John
SEARCH_AVL Tom
SEARCH_BP B W
VLOAD
VPRINT A
VPRINT B
VPRINT C
ADD Tom Pfizer 38 Seoul
VLOAD
VPRINT A
VPRINT B
EXIT
```

```
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
Happy Moderna 1 23 Gwangju
Edward Pfizer 0 98 Busan
Gucci AstraZeneca 1 43 Ulsan
Chanel Moderna 0 38 Daejeon
```

command.txt와 input_dada.txt

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

vector에 정보가 없을 때 오류문구 출력

```
========== ERROR ==========
700
============================
```

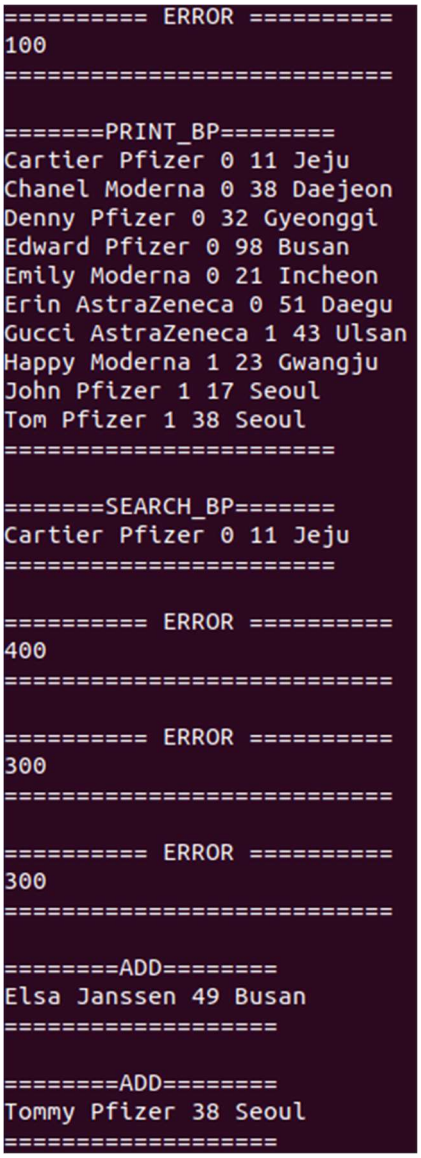
B+ tree에 정보가 없을 때 오류문구 출력

```
========== ERROR ==========
800
============================
```

잘못된 명령어 입력 시 오류문구 출력

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

LOAD가 잘 되었을 때 성공문구 출력



B+TREE에 이미 정보가 있을 때 오류문구 출력

LOAD된 정보들이 잘 출력된 모습

찾고자 하는 정보가 잘 출력된 모습

없는 정보 탐색 시 오류문구 출력

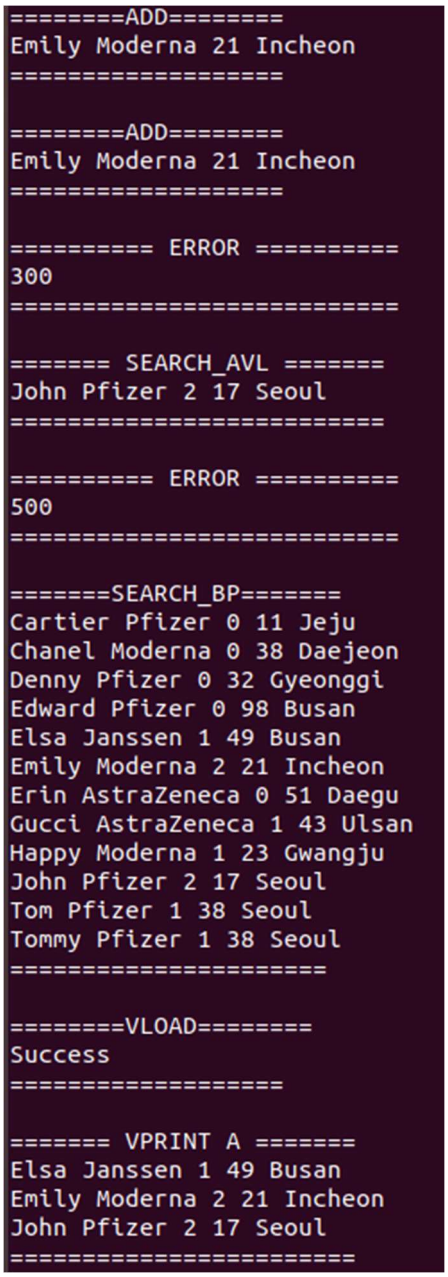
잘못된 인자 입력 시 오류문구 출력

인자를 더 많이 입력 시 오류문구 출력

잘 추가가 된 모습

추가된 정보도 잘 출력된 모습

잘 추가가 된 모습



잘 추가가 된 모습

접종완료자를 추가했을 때 오류가 발생함

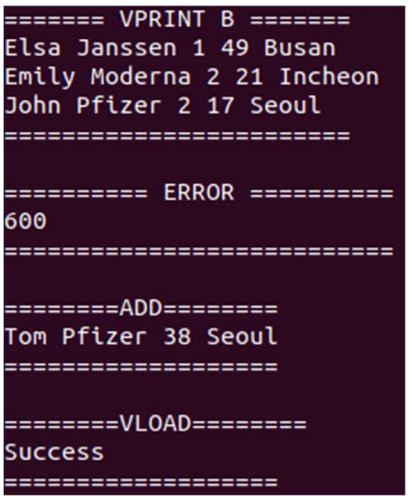
AVL에서 정보탐색이 잘 되었음

AVL에 없는 사람 탐색 시 오류문구 출력

B+트리를 출력한 모습

AVL을 Vector로 잘 로드한 모습

A type로 출력한 모습
백신명 오름차순, 같을 경우 나이 오름차순, 같을 경우 이름 오름차순

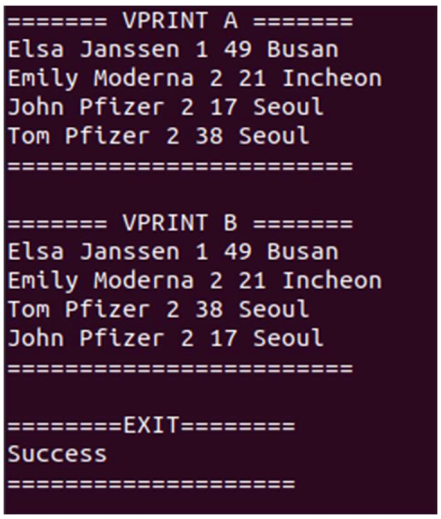


Btype로 출력된 모습
지역명 오름차순 같을 경우 나이 내림차순, 같을 경우 이름 오름차순

잘못된 인자 입력시 오류문구 출력

잘 ADD가 된 모습

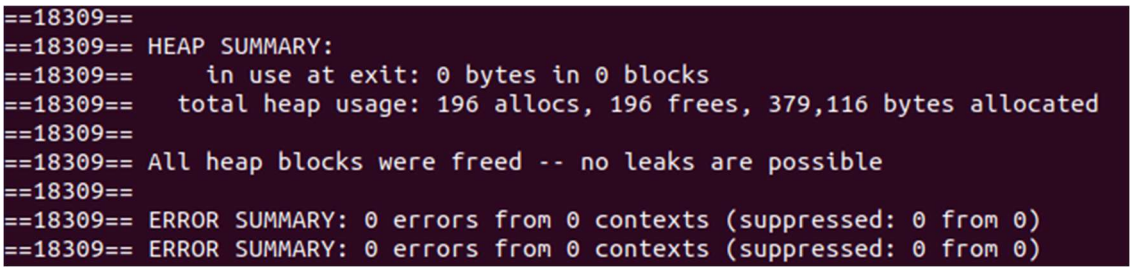
다시 VLOAD함



A type으로 출력한 모습
백신명 오름차순, 같을 경우 나이 오름차순, 같을 경우 이름 오름차순

B type으로 출력된 모습
지역명 오름차순 같을 경우 나이 내림차순, 같을 경우 이름 오름차순

이후에 입력된 명령어가 있지만 실행되지 않고 종료가 된 모습



모든 메모리가 해제된 것을 확인할 수 있다.

5) 고찰

이번 프로젝트는 B+ tree, AVL tree. STL vector를 이용하여 코로나 19 예방접종 관리 프로그램을 구현하는 것을 목적으로 하고 있다. 이번 프로젝트를 진행하면서 가장 어려웠던 부분은 한계 차수 도달 시 노드가 분할하는 것과, 삭제였다. 노드를 분할하는 것에서 인덱스 노드의 분할이 더 어려웠었다. 데이터 노드의 분할 시 가운데 원소와 마지막 원소를 새 노드에 넣어서 분할하는 것과는 달리 마지막 원소를 새 노드에 넣고, 가운데 원소는 부모 혹은 새로운 부모를 생성하여 그곳에 집어 넣은 후, MostLeftChild와 Parent를 지정해주었다. 삭제하는 부분에서는 특히 B+트리의 삭제가 더 어려웠는데, 처음에는 후위 순회로 진행하려 했었다. 일반적은 bst처럼 진행했는데 자꾸 누락된 부분이 생겨서 레벨 오더 순회 방식으로 진행하였다. AVL에서는 회전이 가장 어려웠었다. 이전의 나는 RL이 RR+LL인 것만 알고 RR후 LL을 시행하였다. 그렇게 했더니 코드 볼륨도 커질 뿐만 아니라 잘못된 인자 할당에 의해 내가 원하는 대로 나오지 않을 때가 많았다. 하지만, 그림을 그려 실질적으로 바뀌는 노드는 그렇게 많지 않다는 것을 깨닫고 각 노드에 자식을 바꿔주는 방식으로 진행했더니 전보다 코드 볼륨이 크게 줄어든 것을 확인할 수 있었다.