# DS_Project_2_2021_2

**2018202060**

**컴퓨터정보공학부**

**이준형**

# Contents

# Introduction

프로젝트에 내용에 대한 설명

이번 데이터구조 2차 프로젝트는 B+-Tree, AVL Tree, STL을 이용해 코로나 예방접종 관리 프로그램을 구현하는 것입니다. input_data.txt와 command.txt로 부터 이름, 백신 명, 접종 횟수, 나이, 지역명을 관리하며, 접종 대상자 및 접종 완료자에 대한 정보를 제공할 수 있습니다. B+-Tree를 이용하여 접종예정자를 관리하며, AVL Tree를 이용하여 접종 완료자를 관리합니다. STL을 이용해 접종 완료자를 다양한 정렬 방법으로 출력할 수 있습니다.

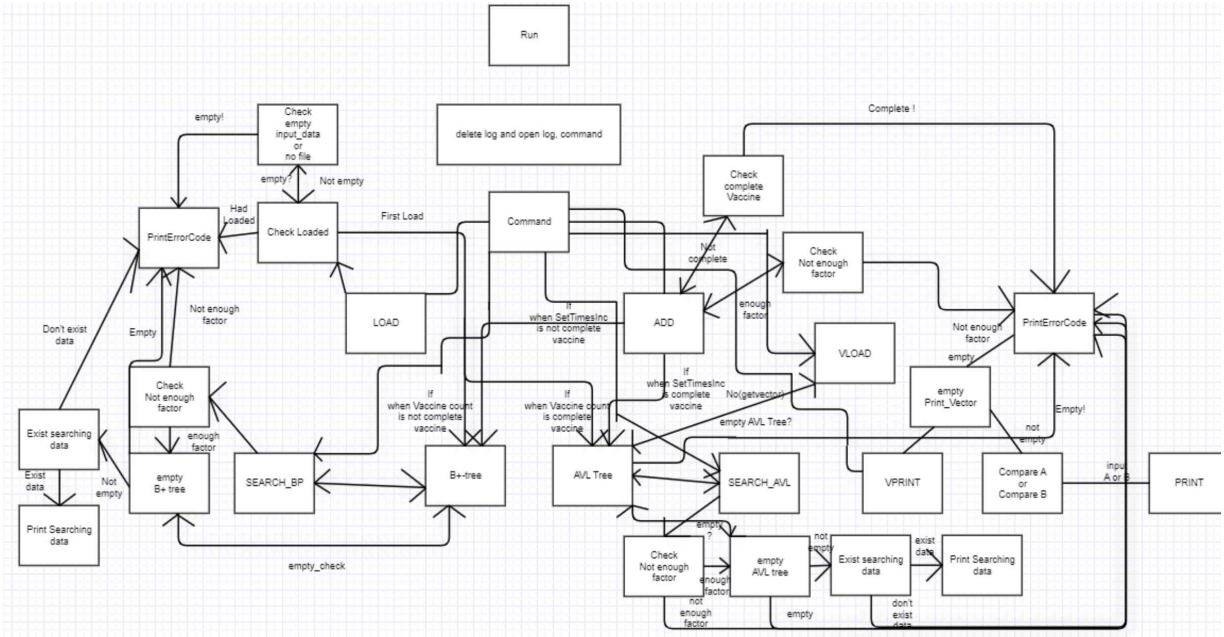
프로그램은 이름, 백신이름, 접종횟수, 나이, 지역명이 담긴 파일을 “LOAD”을 통해 해당 정보를 클래스 노드에 저장합니다. Janssen 접종 대상자는 횟수가 1회인 경우 AVL Tree로 가고, 나머지 백신(Pfizer, Moderna, AstraZeneca)는 백신 접종 횟수가 2회인 경우 AVL Tree로 insert해줍니다. LOAD를 받았을때는 input_data.txt에서 띄어쓰기 단위로 구분합니다. 유저들의 데이터를 담아둘 B+tree는 3차로 구현합니다. LOAD 혹은 ADD로 b+tree에 데이터를 저장합니다. 인덱스 노드와 데이터 노드로 구성을 하며, 각 노드클래스는 b+ tree노드를 상속받습니다. 데이터노드는 이름, 백신이름, 접종횟수, 나이, 지역명을 저장하는 vaccinationData를 map컨테이너 형태로 가지고 있어야합니다. ADD 명령어로 B+tree에 저장된 데이터의 접종횟수를 업데이트 후에, 접종 완료자를 필터링 하여 AVL Tree에 저장합니다. 이때 B+Tree에서 삭제하지 않습니다.AVL tree는 vaccinationData class로 구성되어있습니다. AVL Tree의 특징인 balance factor로 균형을 유지해야합니다. B+tree는 대소문자를 구별하되, AVL Tree는 대소문자를 구별하지 않습니다.

이번 프로젝트의 command는 총 8개입니다. 1. LOAD는 input_data.txt에서 정보를 불러오는것 2. VLOAD는 AVL Tree로부터 Print_vector에 STL을 이용하여 vector로 불러오는것 3. ADD는 데이터를 직접 추가하는것 4. SEARCH_BP 는 두가지 형태가 있는데, 알파벳 두개를 받았을때는 그 알파벳 B+Tree안 범위안에 있는 이름을 출력하는것, 두번째는 해당 사용자의 정보를 찾는것 5. SEARCH_AVL은 AVL Tree에서 사용자의 정보를 찾는것 6. VPRINT는 VLOAD로 불러온 vector를 출력하는 것 7. PRINT_BP는 B+Tree에 있는 노드를 전부 출력하는것 8. EXIT은 메모리를 해제하며, 프로그램을 종료하는것입니다. command를 넣고 나서의 결과는 log.txt에 저장해야합니다.

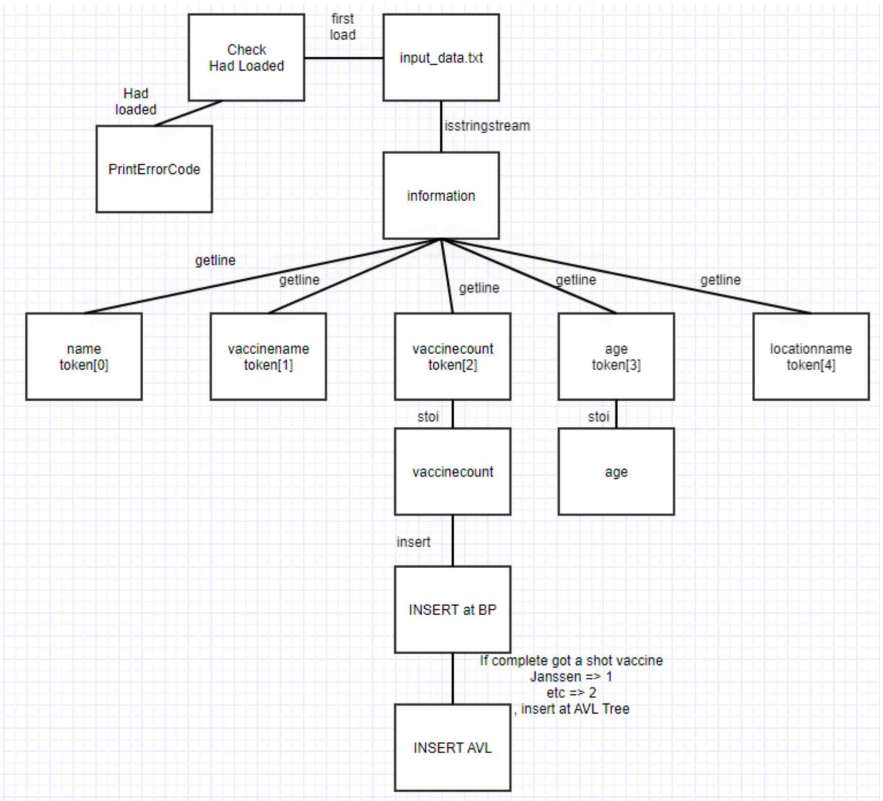
# Flowchart

설계한 프로젝트의 플로우차트를 그리고 설명 이번

프로젝트에서 제가 생각한 프로젝트의 Flowchart입니다.

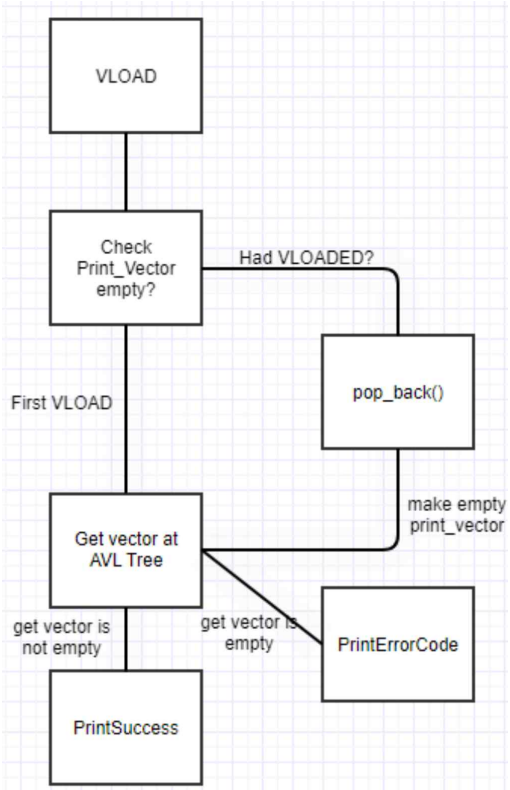



제가 구현한 LOAD명령어의 Flowchart입니다.



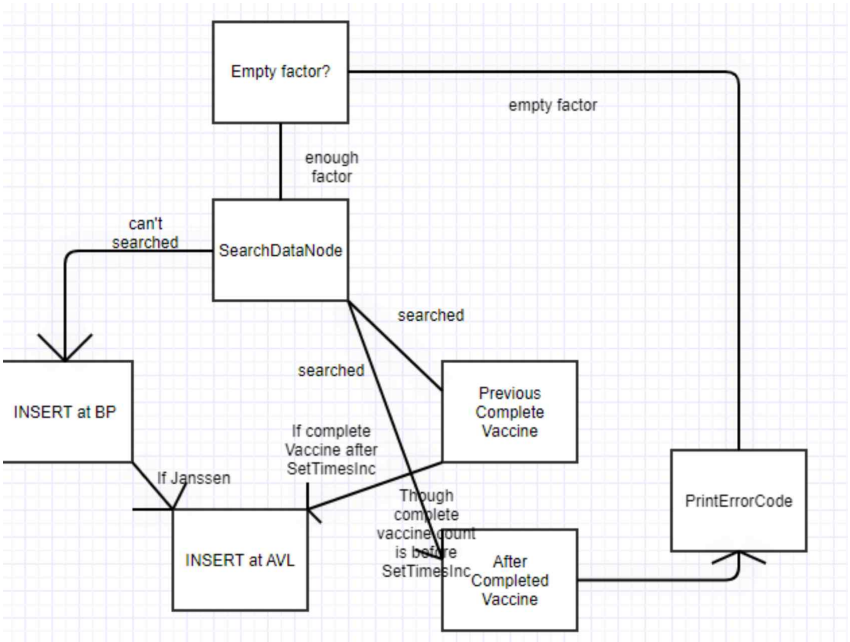

LOAD를 입력 받았을때, input_data.txt에서 한줄씩 입력을 받아서 getline을 사용할 수 있게 istringstream을 이용해서 string 의 크기 5인 배열을 만들어 각 데이터 정보를 입력받아 stoi를 이용해 백신접종횟수와 나이를 정수로 바꾸고 B+tree에 INSERT를 하고, 그 백신접종횟수가 완료인경우에는 AVL Tree에 insert해주었습니다.

제가 구현한 VLOAD입니다.



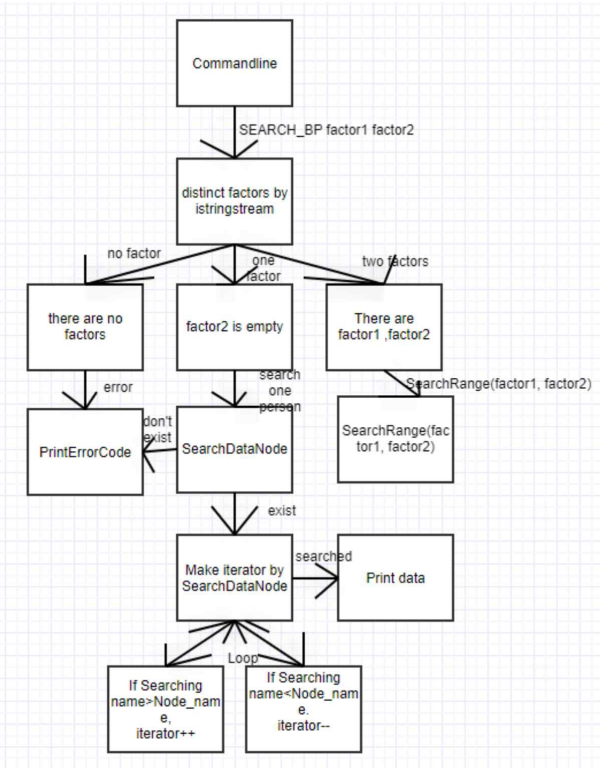

VLOAD를 입력받았을때 기존에 VLOAD를 입력받았다면 Print_vector에 node가 들어가 있으므로 pop_back()이라는 함수를 통해 Print_Vector의 사이즈만큼 pop_back시킨 후, AVL Tree에 있는 node를 Print_Vector에 return했습니다.

제가 구현한 ADD명령 입니다.



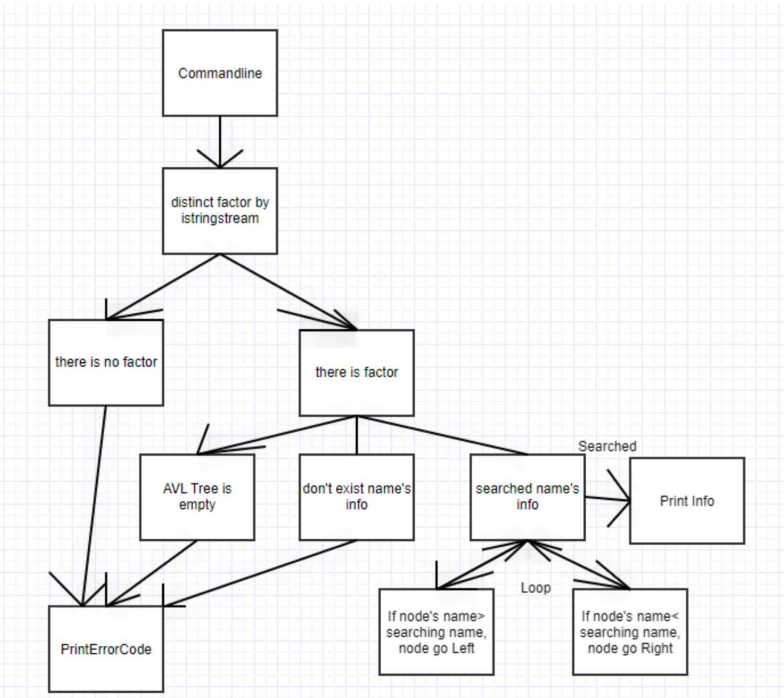

ADD를 받기전에 인자가 충분히 받아졌는지 확인 후, 충분하다면 그 인자 중 이름을 SearchDataNode함수를 통해서 이름을 찾고, 이름을 찾지 못했을 경우 B+ Tree에 삽입을 하는데 만약 ADD의 인자로 Janssen이 온다면 AVL Tree에 바로 넣어주었습니다. 만약에 SearchDataNode를 통해 찾았을 경우, settimesinc로 백신카운트를 1씩 증가하고, 백신접종횟수가 완료되었을경우 AVL Tree에 삽입해주고,SearchDataNode를 통해 찾았는데, 백신접종횟수가 이미 완료된경우 에러코드를 출력하게 했습니다.

제가 구현한 SEARCH_BP입니다.



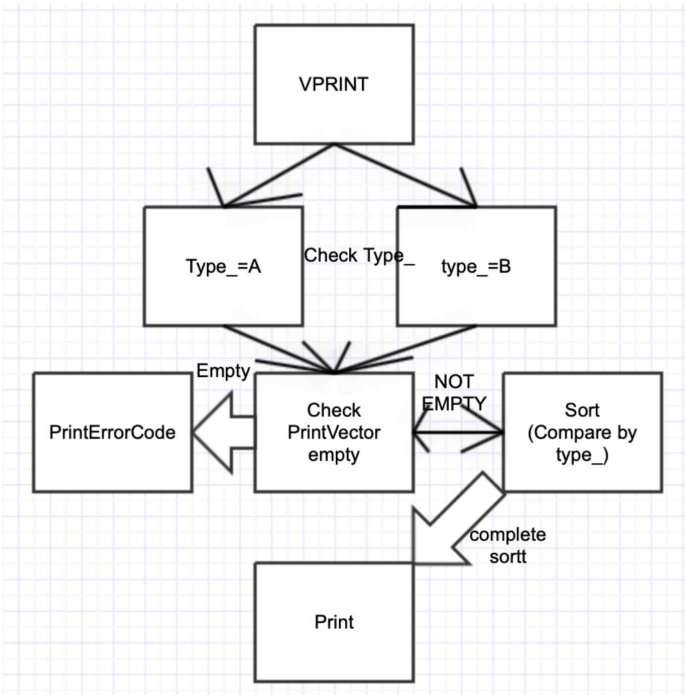

먼저 istringstream으로 인자가 몇개인지 확인을 합니다. 인자가 없는 경우, 에러를 출력합니다. 인자가 1개인경우 SearchDataNode를 통해서 data가 있는지 없는지 확인을 하고, 없는 경우 에러코드를 출력하고 있는경우 iterator를 통해 찾는 이름과 크기를 비교해서 iterator를 증가 혹은 감소시켜서 찾는 이름의 정보를 찾은 후, 출력시켰습니다. 인자가2개인경우, SearchRange를 통해서 찾는데, 이것은 알고리즘의 설명에 적겠습니다.

제가 구현한 SEARCH_AVL입니다.



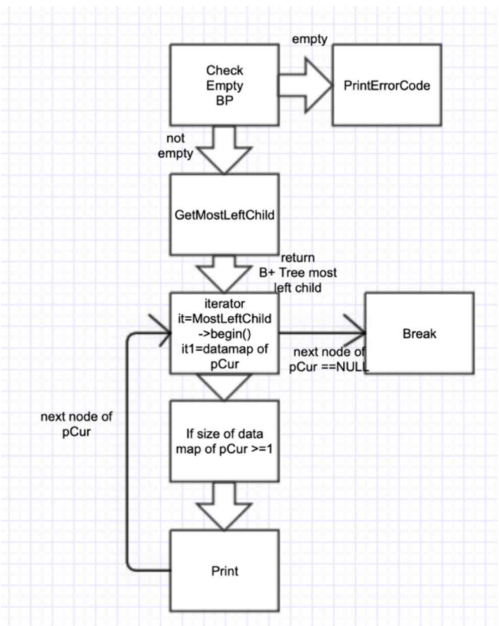

SEARCH_BP처럼 istringstream을 통해 인자를 구별합니다. 인자가없는경우 에러를 출력합니다. 인자가 있는경우, AVL Tree가 비어있는지 확인을 하고, 비어있다면 에러출력, 비어있지않다면 해당이름의 정보가 존재하는지 확인합니다. 존재하지 않는경우 에러를 출력하고, 찾았다면, BST의 searching algorithm처럼 찾는 이름이 작다면 pLeft로 가고, 찾는 이름이 크다면 pRight로 가게 해서 찾는경우, 정보를 출력합니다.

제가 구현한 VPRINT입니다.




VPRINT를 입력받았을때, Type의 종류를 파악합니다.(원래 Type이 있고 없고도 하려했으나, 주어진 프로젝트제안서에는 없었습니다.) 그리고 PrintVector가 비었는지 안 비었는지 확인하고, 비었을경우 에러코드를 출력시킵니다. 비어있지 않은경우에는 type에 따라, 문자열과 나이를 비교하여 정렬한 후, 출력합니다. 정렬하는 방식은 type이 A인경우 처음에 백신이름순으로 정렬, 백신이름이 같은경우 나이로 정렬 나이도 같은경우에는  이름으로 정렬을 했습니다. Type B인경우 사는지역, 나이, 이름 순으로 정렬했습니다.

제가 구현한 PRINT_BP입니다.




먼저 BP Tree가 비었는지 확인하고, 비어있는 경우에는 에러코드를 출력합니다. 비어 있지 않는 경우에는 맨 왼쪽의 자식 노드를 가져와서 그 자식노드를 iterator형 변수에 저장을 시키고, 그 변수에 담긴 데이터맵의 정보를 출력하고, 맨 왼쪽의 자식노드를 다음 노드를 가르키게 하다가, 맨 왼쪽의 자식노드를 가르키던 변수가 NULL을 만나면 출력을 멈추고 반복문을 break 했습니다.

# Algorithm

## 프로젝트에서 사용한 알고리즘의 동작을 설명

이번 데이터구조실습 2차 프로젝트의 주제에서 제가 생각하고 사용한 자료구조는 B+ tree, AVL Tree, QUEUE, Vector, String, pair, map 먼저 B+ tree에 관해서 설명드리겠습니다. B+ Tree는 차수(order)란것이 존재합니다. 그 차수만큼 노드를 가져 한덩어리를 만든다고 생각했습니다. 그래서 BST보다 더 적은 높이를 가지는 것이 장점입니다.

1) Insert

LOAD나 ADD명령어로 Insert함수를 사용하면서 B+ Tree를 채웁니다. 먼저 B+ tree에 key와 value를 설정해야해서, 사람이름을 key, 그사람에 관한 정보노드를 value로 잡았습니다. 처음 Insert를 하기전에 B+ Tree의 root는 존재하지 않기때문에, DataNode를 할당한 후, key 와 value를 넣어서 datamap을 완성했습니다. Insert를 사용을 한 적이 있다면, 해당 key가 들어있는 datanode를 찾아서 데이터노드가 차수를 초과하면 split을 하게 했습니다.

2) searchDataNode

key값을 입력받아서 그 key에 맞는 value를 반환해주는 함수입니다. 먼저 B+ Tree의 제일 왼쪽 자식을가르키는 노드를 생성하고, 그 노드의 인덱스를 받을 map형 iterator 변수를 생성하고 그 iterator형 변수의 Key값보다 클 경우 다시 제일 왼쪽자식을 부르고 작을경우, iterator형 변수의 index를 늘리거나 줄이면서 찾은 후, 찾을 경우 그 노드를 반환합니다

3) splitDataNode

처음에 프로젝트에서 주어진 차수를 이용해 쪼개지는 지점을 선언하고, 가운데 데이터, 두번째 가운데 데이터, 맨왼쪽 iterator형 변수를 선언했습니다. Insert에서 넘겨준 데이터노드를 맨왼쪽 데이터를 가르킬 iterator형 변수에 담아주고, for문을 통해서 다음노드를 가르키다가 i가 쪼개지는 지점보다 1작을때 가운데 데이터를 가르킬 iterator형 변수에 맨왼쪽노드를 가르킬 iterator형 변수를 넣어줍니다. 그리고 두번째 가운데 데이터를 가르킬 iterator형 변수에 가운데 데이터를 가르킬 iterator형 변수를 넣어줍니다. 그 후 counting을 시키면서(차수와 쪼개지는 지점의 차보다 작을때동안)가운데 데이터를 가르킬 iterator형 변수의 first(key), second(value)를 datamap에 삽입을 시켜주고, 다음 노드를 가르키게합니다. 그리고 다시 카운팅을 하면서 앞에서 증가된 맨왼쪽을 가르키는 iterator형 변수를 감소시키면서 그거와 같은 key값을 가지는 map을 delete해줍니다. 이때 delete는 STL<map>에 erase함수를 이용했습니다. 그 후에 B+ Tree의 특징인 datamap을 연결리스트처럼 setnext setprev를 해주어서 doubly linked list형태처럼 만들었습니다. 그 후, 부모노드가 없으면 부모노드와 자식노드를 만들어 주었고, 부모노드가 있다면 insert해서 했던것처럼 인덱스노드가 초과되었는지 확인하고 이번에는 인덱스 노드를 split을 하게 했습니다.

4) splitIndexNode

이것도 splitDataNode처럼 맨 왼쪽의 인덱스와 가운데인덱스를 가르킬 iterator형 변수를 선언하고, splitDataNode에서 받은 노드의 인덱스맵을 맨왼쪽의 인덱스를 가르키는 iterator형 변수에 담아두고, 앞에서 한것처럼 쪼개지는 지점과 차수를 보면서 맨왼쪽을 가르키는 iterator형 변수를 증가시킵니다. 그리고 맨마지막에 입력받은 노드의 왼쪽 자식을 가운데를 가르킬 iterator형 변수의 second(value)로 하게하고, 그 맨마지막에 입력받은 노드를 부모노드로 선언했습니다. 그 후 차수와 쪼개지는 지점의 차를 이용해서 iterator형 변수를 증가시키면서 부모노드를

채워줍니다. 그러다가 중복되는 key값이 나오면 delete를 erase함수를 통해 지워줍니다. 그 후 datanode를 쪼갤때처럼  부모노드와 맨왼쪽자식노드를 set해주면서 delete를 해줍니다. 그러고 indexnode가 초과했는지 확인을 하면서 계속 split을 하게 했습니다.

5) exceedDataNode

datanode가 초과했는지 초과하지 않았는지 확인하는 함수입니다. Map으로 선언했는데 다른 클래스타입으로 어떻게 바꿔야 될지 몰라서 조교님께 여쭤봐서<dynamic_cast>라는 키워드를 통해서 B+ 데이터노드를 캐스팅해서 인자로받은 노드의 데이터 맵을 가져올 수 있었습니다. 그리고 그것의 사이즈가 차수-1보다 높은 경우 초과했다고 생각했고, 아닌 경우는 초과하지 않았다고 생각했습니다.

6) exceedIndexNode

인덱스노드의 초과를 확인하는 방법도 데이터노드의 초과를 확인하는 방법과 동일합니다.

7) SearchRange

알파벳 범위안에 속하는 사람의 정보를 찾아도 '어떻게 저장하지?' 라는 생각을 하게 되었습니다. 프로젝트제안서에서 얼핏 본것같은 stack과 queue을 이용해도된다라는 글을 통해서, 큐를 이용해서 범위안에 드는 사람의 정보를 push해주었습니다. 여기서 혹시 인자를 두개 받았는데, 모두 알파벳한글자가 아닐 수 있기때문에 string함수중에서 .at(0)을 이용해서 맨 앞글자를 범위로 생각하게했습니다. Break를 만날동안 범위안에 start문자열의 첫글자보다 크거나 같은 사람의 이름이 담긴 VaccinationData를 push했습니다. 그렇게 앞에서 B+ Tree의 특징을 이용한 getnext를 통해서 next가 없을때까지 탐색을 하게 했습니다. 그러고 탐색이 끝나면 큐가 비었으면 에러를 출력하고, 비어있지 않았으면 빌때까지 사람의 정보를 출력하면서 pop시켜주었습니다.

8) Print

앞에서 PRINT_BP에서 설명했듯이 먼저 BP Tree가 비었는지 확인하고, 비어있는 경우에는 에러코드를 출력합니다. 비어 있지 않는 경우에는 맨 왼쪽의 자식 노드를 가져와서 그 자식노드를 iterator형 변수에 저장을 시키고, 그 변수에 담긴 데이터맵의 정보를 출력하고, 맨 왼쪽의 자식노드를 다음 노드를 가르키게 하다가, 맨 왼쪽의 자식노드를 가르키던 변수가 NULL을 만나면 출력을 멈추고 반복문을 break 했습니다.

마지막으로 AVL Tree에 대해서 설명드리겠습니다.
AVL Tree는 B+ Tree에 order가 있듯이 bf(balance factor)를 통해서 트리의 균형을 조정합니다. Bf를 통해서 LL, LR, RL, RR과 같은 회전을 가지게 됩니다. 균형있는 트리이기때문에 탐색과 삽입 삭제가 모두 O(log n)이라는 복잡도를 가지는 장점이 있습니다.

1) Insert

BST처럼 사람 이름의 크기를 비교해서 트리를 구성했습니다. 그리고 4개의 회전을 고안했습니다. LL, LR, RL, RR이 있습니다. 트리를 구성하면서 BF(balance factor)을 set하게 했습니다. 받은 인자의 노드가 새로 생성한 노드의 사람의 이름보다 클경우 a의 오른쪽으로 지정하고, bf=-1로 생각했습니다. 작을경우는 왼쪽으로 설정하고 bf=1로 했습니다. 그렇게 하고, 인자로 받은 노드와 같을때까지 반복문을 통해서 인자로받은 노드와 새로 만든 노드의 이름을 비교해서 BF 변수를 설정했습니다.
그렇게 하고 균형이 맞혀졌는지(bf가 -1, 0, 1인지 확인)을 하고 맞혀졌다면 괜찮지만 균형이 맞지 않다면 b(a의 왼쪽노드)의 bf를 확인해서 LL회전을 하게 한 후, Bf를 다시

설정했습니다. LL은 서브트리의 높이는 변하지 않고, 왼쪽으로 돌린다는 개념으로 접근했습니다. LR은 왼쪽노드의 오른쪽노드를 오른쪽노드로 보내고, 그 자리를 기존의 오른쪽노드의 왼쪽노드로 설정하고, 왼쪽노드를 오른쪽노드의 오른쪽노드를 가르키고, 기존의 오른쪽노드의 왼쪽을 기존의 왼쪽노드로 하고, 기존노드를 오른쪽노드로 했습니다. 그렇게 하고, 기존의 bf를 확인하고, switch문을 통해 bf에 따른 경우를 나눠 bf를 다시 설정했습니다. RR과 RL또한 LL과 LR과 방향만 바뀐 유사한 알고리즘입니다.

2) Search

BST에서 search알고리즘과 같습니다. 찾아야될 이름이랑 반환할 이름이 같을때 까지 반복문을 사용하면서 찾아야될 이름이 작으면서 자식노드가 있을경우 왼쪽으로가고, 이름이 크면 오른쪽으로 가고 자식노드가 없는상태인데 그 노드가 찾는 이름을 갖고 있지 않은경우도 반환하면서 error를 출력시켰습니다.

3) GetVector

AVLNODE 클래스형 변수를 선언하고 root를 가르키게 했습니다. 앞서서 B+ Tree에서 SearchRange에서 썻던 큐를 선언했고, root를 가르키는 변수가 끝날때 까지 push를 해주었습니다. 그리고 stl<vector>에 있는 push_back을 이용해서 q에 있는 백신데이터를 담아주면서 큐를 팝 시켰습니다.

AVL Tree의 알고리즘은 insert빼면은 1차 프로젝트때 구현한 BST와 똑같은 것을 확인할 수 있습니다.

# Result Screen

모든 명령어에 대해 결과화면을 캡처하고 동작을 설명

input_data.txt

```
1   Denny Pfizer 0 32 Gyeonggi
2   Tom Pfizer 1 38 Seoul
3   Emily Moderna 0 21 Incheon
4   John Pfizer 1 17 Seoul
5   Erin AstraZeneca 0 51 Daegu
```

아래와 같은 input_data.txt로 결과 화면을 보여드리겠습니다.

1) LOAD

command.txt

```
1   LOAD
2   LOAD
3   PRINT_BP
```

다음과 같은 명령어를 주었습니다.

log.txt

```
1   ======== LOAD ========
2   Success
3   ======================
4   ========== ERROR ==========
5   100
6   ==========================
7
8   ======= PRINT_BP =======
9   Denny Pfizer 0 32 Gyeonggi
0   Emily Moderna 0 21 Incheon
1   Erin AstraZeneca 0 51 Daegu
2   John Pfizer 1 17 Seoul
3   Tom Pfizer 1 38 Seoul
```

LOAD가 두번째로 들어온 경우, ERROR가 출력된것을 확인할 수 있습니다.

2)ADD

command.txt

```
1   ADD Elsa Janssen 49
2   LOAD
3   ADD Elsa Janssen 49 Busan
4   PRINT_BP
5   PRINT_AVL
```

다음과 같은 명령어를 주었습니다.

log.txt

```
1   ========== ERROR ==========
2   300
3   ==========================
4
5   ======== LOAD ========
6   Success
7   ======================
8   ======== ADD ========
9   Elsa Janssen 49 Busan
0   ======================
1   ======= PRINT_BP =======
2   Denny Pfizer 0 32 Gyeonggi
3   Elsa Janssen 1 49 Busan
4   Emily Moderna 0 21 Incheon
5   Erin AstraZeneca 0 51 Daegu
6   John Pfizer 1 17 Seoul
7   Tom Pfizer 1 38 Seoul
8
9   ======= SEARCH_AVL =======
0   Elsa Janssen 1 49 Busan
1   ==========================
```

add의 인자가 부족할 경우 에러를 출력합니다.하지만 이번프로젝트에서 LOAD가 되어야 ADD를 할 수 있게 설정했습니다.

3)VLOAD

command.txt
```
1   LOAD
2   VLOAD
3   ADD Elsa Janssen 49 Busan
4   VLOAD
```
이러한 명령어를 주었습니다.

log.txt
```
1   ======== LOAD ========
2   Success
3   ======================
4   ========== ERROR ==========
5   200
6   ============================
7
8   ======== ADD ========
9   Elsa Janssen 49 Busan
0   ======================
1   ======== VLOAD ========
2   Success
3   ======================
```
첫 VLOAD는 AVL NODE가 비어있기때문에 Elsa가 얀센을 맞아서 AVL Node로 갔기 때문에 VLOAD가 된것을 확인할 수 있습니다.

4)SEARCH_BP start end  SEARCH_BP name

command.txt
```
1   LOAD
2   SEARCH_BP b e
3   SEARCH_BP B E
4   SEARCH_BP LEE
5   SEARCH_BP John
```
이와 같은 명령어를 주었습니다.

log.txt
```
1   ======== LOAD ========
2   Success
3   ======================
4   ========== ERROR ==========
5   400
6   ============================
7
8   ======= SEARCH_BP =======
9   Denny Pfizer 0 32 Gyeonggi
0   Emily Moderna 0 21 Incheon
1   Erin AstraZeneca 0 51 Daegu
2   ======================
3   ========== ERROR ==========
4   400
5   ============================
6
7
8   ======= SEARCH_BP =======
9   John Pfizer 1 17 Seoul
0   ====================
```
대소문자 구별해서 search_range가 된것을 확인할 수 있고 단일 이름인데 잘 출력 된것을 확인 할 수 있습니다.

5) SEARCH_AVL

command.txt
```
ADD Elsa Janssen 49
LOAD
ADD Elsa Janssen 49 Busan
PRINT_BP
PRINT_AVL
```
이와 같은 명령어를 주었습니다.(2_ADD사진과 동일)

log.txt
```
========== ERROR ==========
300
==========================

======== LOAD ========
Success
======================
======== ADD ========
Elsa Janssen 49 Busan
======================
======= PRINT_BP =======
Denny Pfizer 0 32 Gyeonggi
Elsa Janssen 1 49 Busan
Emily Moderna 0 21 Incheon
Erin AstraZeneca 0 51 Daegu
John Pfizer 1 17 Seoul
Tom Pfizer 1 38 Seoul

======= SEARCH_AVL =======
Elsa Janssen 1 49 Busan
==========================
```
Elsa가 안센 접종횟수가 1이므로 제대로 출력 된것을 확인 할 수 있습니다.

6) VPRINT A, VPRINT B

command.txt
```
LOAD
ADD Elsa Janssen 49 Busan
ADD Tommy Pfizer 38 Seoul
VPRINT
PRINT_BP
SEARCH_BP Tom
ADD John Pfizer 17 Seoul
ADD Emily Moderna 21 Incheon
ADD Emily Moderna 21 Incheon
ADD Emily Moderna 21 Incheon
SEARCH_AVL John
VLOAD
VPRINT A
ADD Tom Pfizer 38 Seoul
VLOAD
VPRINT B
EXIT
```
프로젝트 제안서에 나와있는 명령어를 예시로 들겠습니다.

```
======== LOAD ========
Success
======================
======== ADD ========
Elsa Janssen 49 Busan
======================
======== ADD ========
Tommy Pfizer 38 Seoul
======================
========== ERROR ==========
600
==========================
```

```
======= VPRINT A =======
Elsa Janssen 1 49 Busan
Emily Moderna 2 21 Incheon
John Pfizer 2 17 Seoul
==================
======== ADD ========
Tom Pfizer 38 Seoul
======================
======== VLOAD ========
Success
======================

======= VPRINT B =======
Elsa Janssen 1 49 Busan
Emily Moderna 2 21 Incheon
Tom Pfizer 2 38 Seoul
John Pfizer 2 17 Seoul
==================
```
VPRINT A었을때는 백신이름 순, 나이순, 이름순으로 되있고, VPRINT B는 지역이름순, 나이순(내림차순), 이름 순 으로 되있는것을 확인할 수 있습니다.

7) PRINT_BP

command.txt
```
1   PRINT_BP
2   LOAD
3   PRINT_BP
4   EXIT
```
아래와 같은 명령어를 주었습니다.

log.txt
```
1   ========== ERROR ==========
2   700
3   ============================
4
5   ======== LOAD ========
6   Success
7   ======================
8   ======= PRINT_BP =======
9   Denny Pfizer 0 32 Gyeonggi
0   Emily Moderna 0 21 Incheon
1   Erin AstraZeneca 0 51 Daegu
2   John Pfizer 1 17 Seoul
3   Tom Pfizer 1 38 Seoul
4   ==========EXIT=========
5
6   ======================
```
아래와 같은 결과를 가집니다.

8) EXIT

command.txt
```
1   PRINT_BP
2   LOAD
3   PRINT_BP
4   EXIT
```
7)과 같은 명령어를 주었습니다.

log.txt
```
1   ========== ERROR ==========
2   700
3   ============================
4
5   ======== LOAD ========
6   Success
7   ======================
8   ======= PRINT_BP =======
9   Denny Pfizer 0 32 Gyeonggi
0   Emily Moderna 0 21 Incheon
1   Erin AstraZeneca 0 51 Daegu
2   John Pfizer 1 17 Seoul
3   Tom Pfizer 1 38 Seoul
4   ==========EXIT=========
5
6   ======================
```
EXIT이 된것을 확인할 수 있습니다.

# Consideration

고찰 작성

이번 프로젝트는 자료구조 중 B+ Tree, AVL Tree, Map, Vector, pair, String, Queue를 사용해서 코로나 백신접종 정보관리 프로그램을 구현했습니다. 1차 프로젝트보다 더 많은 자료구조를 사용하면서 점점 더 컴퓨터공학인으로써 성장하는 나 자신이 뿌듯했습니다. 프로젝트를 진행하는 중 dynamic_cast같은 부분을 실습시간에 대면으로 조교님께 질문할 수 있어서 좋았습니다. 그리고 프로젝트 조건에 log.txt 내용을 지웠다가 해야하는 부분에서 파일입출력이 헷갈렸는데 저에게 18학번 이민재 친구가 ios::trunc라는 키워드로 도움을 주었습니다. 하지만 이번 프로젝트에 아쉬운점이 있었습니다. 시험이 끝나고 바로 도입을 해서 마음이 성급했던점이 ADD함수가 LOAD되지않고는 사용할 수 없게 구현했던 점과 메모리 누수를 잡지 못한점이 너무 아쉽습니다. 메모리 누수는 valgrind 라는 것은 조교님께서 알려주셔서 사용했지만, 메모리누수가 어디서 났는지 tracking을 하고 수정을했지만 변경이 되지않아서 메모리 누수부분에 대해서 공부를 해야겠다는 생각을 하게 되었습니다. 앞으로 진행하게 될 3차 프로젝트는 더욱 더 메모리 누수부분에 대해서 학습을 진행한 후, 진행해야될것 같습니다. 그리고 AVL Tree같은 경우에는 교수님께서 LL과 LR코드를 주셔서 반대로 RR과 RL코드를 짜면 되기때문에 편리했고, search 또한 BST와 비슷한 알고리즘이어서 쉽게 구현할 수 있었지만 B+ Tree같은 경우에는 교수님께서 설명을 해주신것을 제대로 복습하지 않아서 강의자료와 같은 동기들과 선배분들께 질문하면서 겨우겨우 해결했던 점이었습니다. 실습과목일수록 수업을 더 중요하게 들어야 된다는 생각을 하게 되었습니다. 메모리 누수공부, 수업을 더 열심히 듣기, 모르는것 바로 물어보기를 실천하면서 이번 프로젝트의 고찰을 마치겠습니다.

2018202060
컴퓨터정보공학부
이준형