

# Report

## 데이터 구조 설계

### Project2

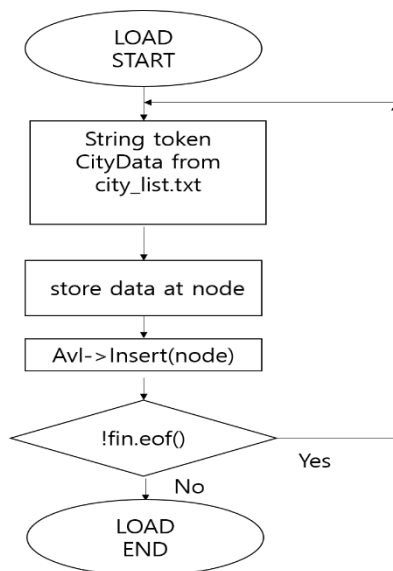
컴퓨터정보공학부 2017202037 오민혁

## < Introduction >

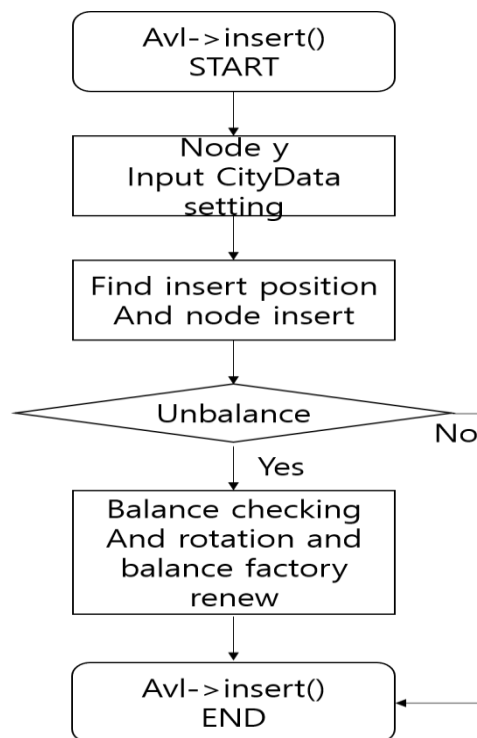
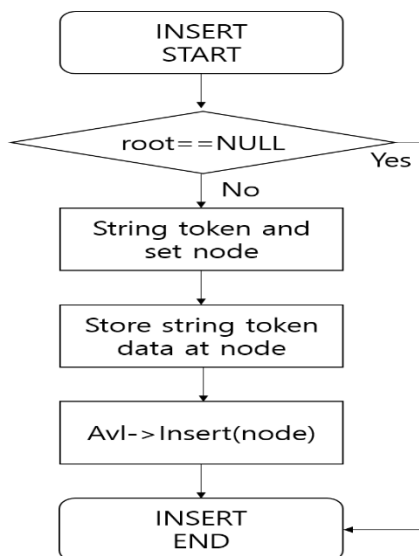
City\_list.txt 파일에 있는 도시들의 지역 번호, 도시 이름, 나라 이름 데이터들을 이용하여 여행 일정 관리 시스템을 구축한다. 이는 도시의 이름을 기준으로 하여 오름차순으로 AVL tree에 저장된다. 그리고 Undirected Complete Graph를 이용하여 최단 거리를 가지는 여행 일정 그래프를 만들어야 한다. 마지막으로 Kruskal 알고리즘을 이용하여 최단 이동거리 그래프(Minimum Spanning Tree)를 구축해야 한다. AVL Tree는 삭제, 삽입, 검색, 출력의 기능을 가지고, Graph와 MST는 출력 기능만을 가진다.

## < Flowchart >

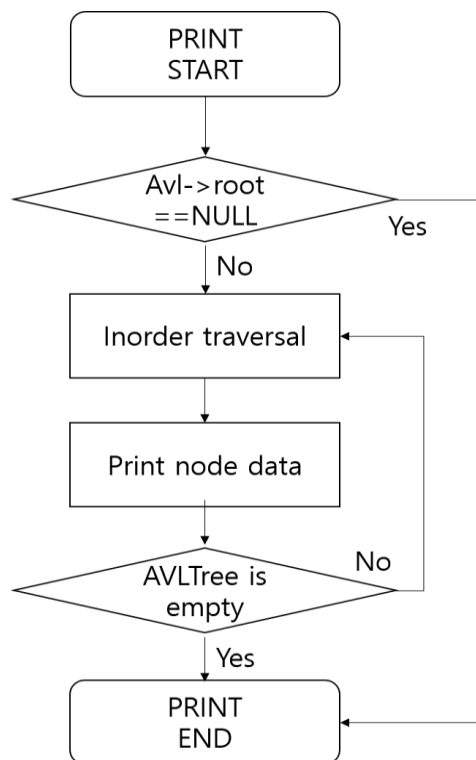
### A. LOAD



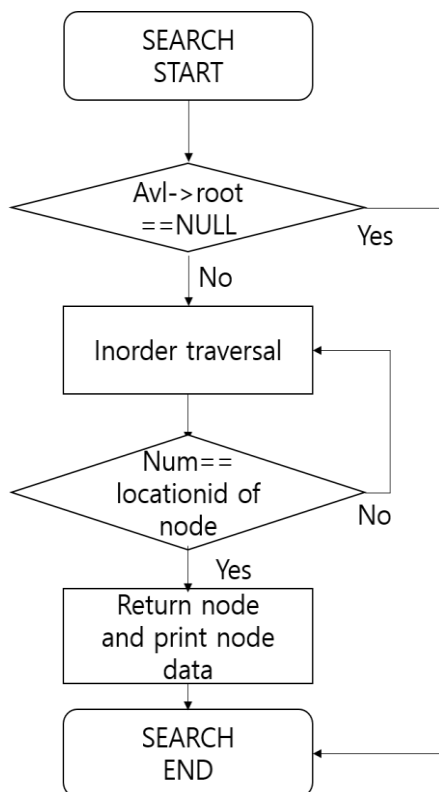
### B. INSERT



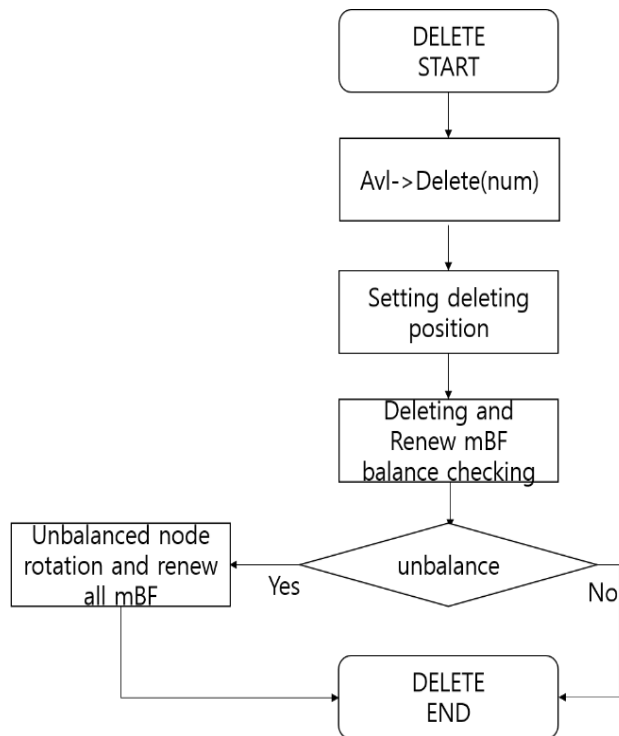
### C. PRINT\_AVL



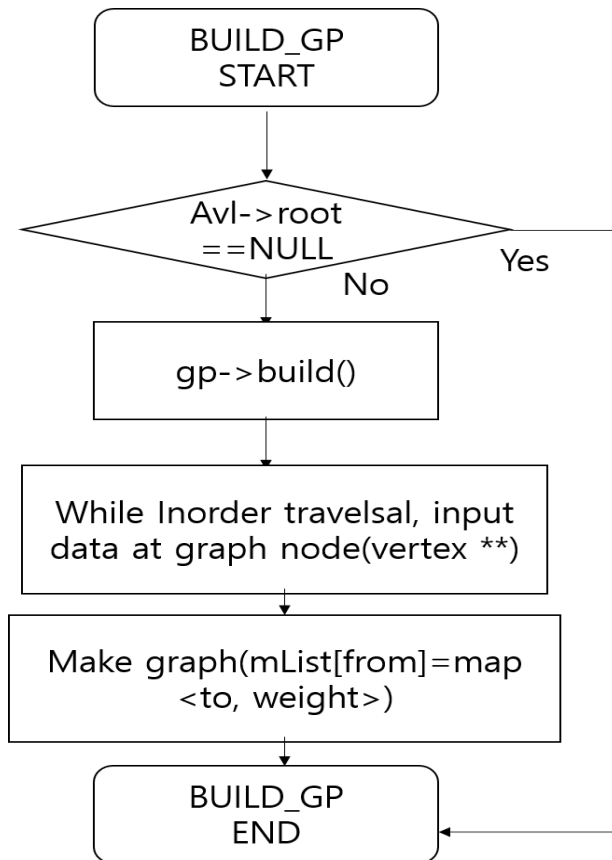
### D. SEARCH\_AVL



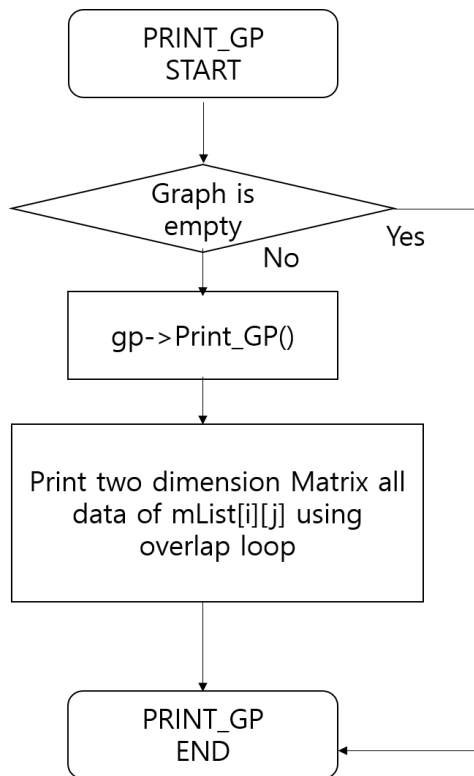
### E. DELETE\_AVL



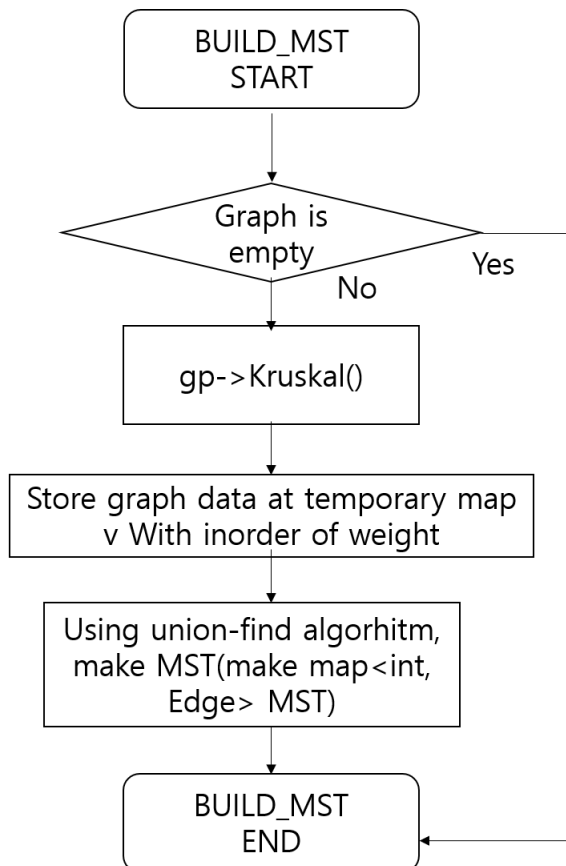
### F. BUILD\_GP



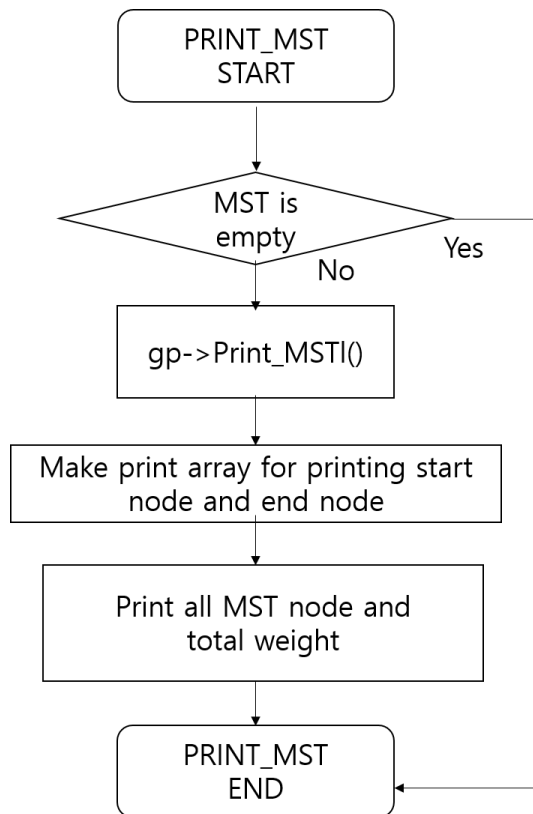
### G. PRINT\_GP



### H. BUILD\_MST



## I. PRINT\_MST



## < Algorithm >

(프로젝트에서 사용한 알고리즘의 동작을 설명(AVL tree삽입, Graph, Kruskal에 대하여 각각 예시를 들어 설명할 것)

### A. AVL Tree 삽입

AVL Tree의 삽입 작동은 우선 AVL 또한 BST 이므로 도시 이름의 순서에 따라 부모노드 보다 작으면 왼쪽 자식으로 삽입되고, 크면 오른쪽 자식으로 삽입되는 것을 이용하여 삽입할 위치를 찾아준다. 그리고 인자로 입력 받은 data를 이용해 Node를 하나 생성하여 찾아준 삽입 위치에 연결해준다. 그 후 밸런스 검사를 하여 왼쪽 Subtree의 left child에 삽입되어 무너진 경우에는 LL Rotation, right child에 삽입되어 밸런스가 무너진 경우에는 LR Rotation, 오른쪽 Subtree의 left child에 삽입되어 무너진 경우에는 RL Rotation, right child에 삽입되어 무너진 경우에는 RR Rotation을 실행하여 주면 된다.

### B. Graph

Graph는 Map library 을 이용하여 만들어주었다. AVL Tree를 inorder 로 순회하면서 data들을 저장하는 vertex 배열에 index에 따라 넣어준다. 그리고 각 vertex들을 연결해줄 2차원 배열 map인 mList를 생성하여 2중 loop를 이용해 <to node, weight>형태로 모든 vertex가 연결되

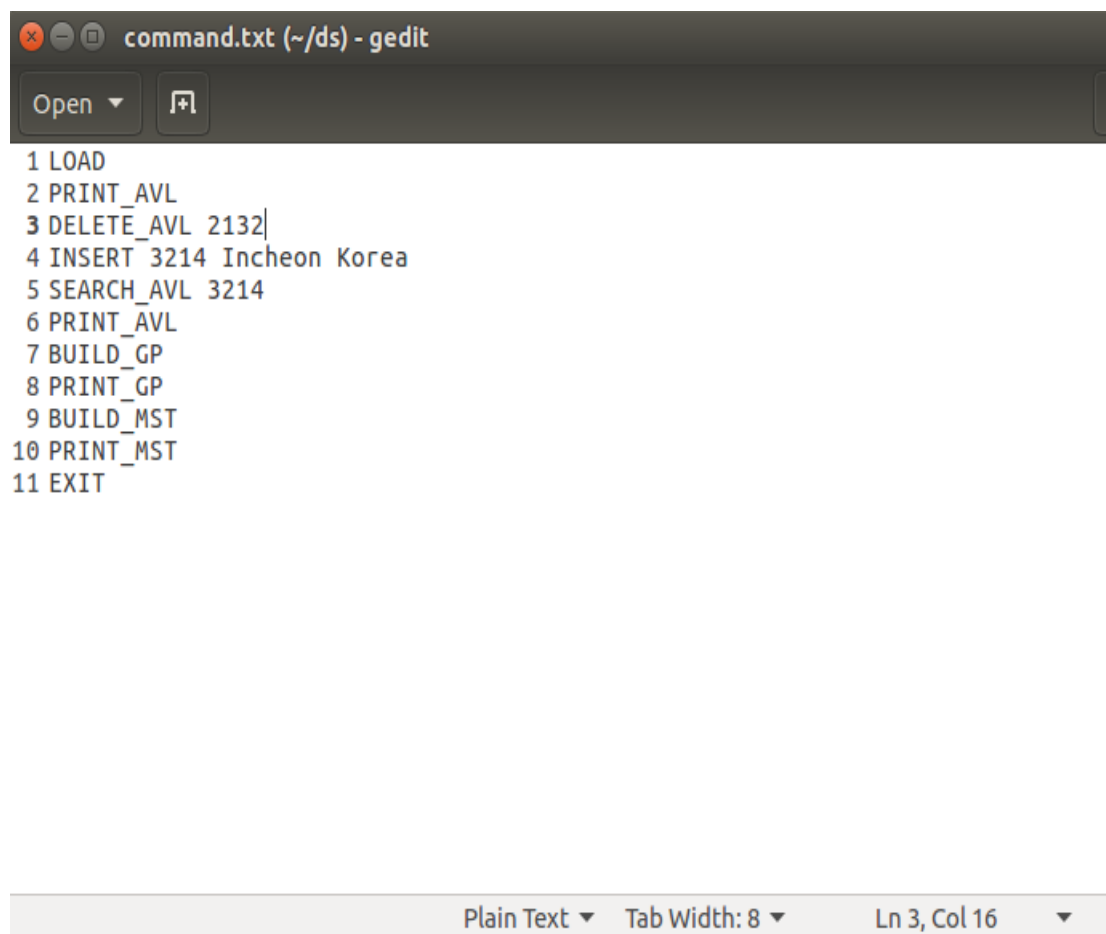
도록 data를 설정해준다.

### C. Kruskal

Kruskal은 union-find algorithm을 이용하여 MST를 만드는 알고리즘이다. Union-find 알고리즘은 각 집합에서 중복되는 원소들이 없게 해주는 것이다. 이를 이용해서 최단 거리로 이루어진 MST를 생성할 수 있다. Kruskal 함수에서 현재 Graph는 정사각형 2차원 배열로 이루어져 있는 상태에서 대각선을 제외하고 upper triangle만을 저장하는 map을 생성한다. 그리고 이 map에서 for문을 이용해 union-find 알고리즘으로 최소 weight 정보만을 걸러내서 map MST에 데이터들을 형식에 맞게 저장한다. 그러면 Minimum Spanning Tree가 map MST에서 이루어져 있는 상태가 된다.

### < Result Screen >

```
bmh@ubuntu:~/ds$ make
g++ -std=c++11 -g -o run AVLNode.cpp AVLTree.cpp CityData.cpp Manager.cpp Graph.
cpp main.cpp AVLNode.h Manager.h Graph.h AVLTree.h CityData.h
bmh@ubuntu:~/ds$ ./run
```



```
command.txt (~/.ds) - gedit
Open
1 LOAD
2 PRINT_AVL
3 DELETE_AVL 2132
4 INSERT 3214 Incheon Korea
5 SEARCH_AVL 3214
6 PRINT_AVL
7 BUILD_GP
8 PRINT_GP
9 BUILD_MST
10 PRINT_MST
11 EXIT
Plain Text Tab Width: 8 Ln 3, Col 16
```

```

1 ==> command 1) LOAD
2 Success
3 ==> command 2) PRINT_AVL
4 ( 9821, Athens, Greece )
5 ( 7485, Busan, Korea )
6 ( 1241, Chicago, USA )
7 ( 8590, Hanoi, Vietnam )
8 ( 6542, LA, USA )
9 ( 4873, NewYork, USA )
10 ( 2354, Paris, France )
11 ( 5419, Roma, Italy )
12 ( 1543, Seoul, Korea )
13 ( 2132, Tokyo, Japan )
14 ==> command 3) DELETE_AVL
15 Success
16 ==> command 4) INSERT
17 ( 3214, Incheon, Korea )
18 ==> command 5) SEARCH_AVL
19 ( 3214, Incheon, Korea )
20 ==> command 6) PRINT_AVL
21 ( 9821, Athens, Greece )|
22 ( 7485, Busan, Korea )
23 ( 1241, Chicago, USA )
24 ( 8590, Hanoi, Vietnam )
25 ( 3214, Incheon, Korea )
26 ( 6542, LA, USA )
27 ( 4873, NewYork, USA )
28 ( 2354, Paris, France )
29 ( 5419, Roma, Italy )
30 ( 1543, Seoul, Korea )
31 ==> command 7) BUILD_GP
32 Success
33 ==> command 8) PRINT_GP
34 0      2336    8580    1231    6607    3279    4948    7467    4402    8278
35 2336    0      6244    1105    4271    943     2612    5131    2066    5942
36 8580    6244    0      7349    1973    5301    3632    1113    4178    302
37 1231    1105    7349    0      5376    2048    3717    6236    3171    7047
38 6607    4271    1973    5376    0      3328    1659    860     2205    1671
39 3279    943     5301    2048    3328    0      1669    4188    1123    4999
40 4948    2612    3632    3717    1659    1669    0      2519    546     3330
41 7467    5131    1113    6236    860     4188    2519    0      3065    811
42 4402    2066    4178    3171    2205    1123    546     3065    0      3876
43 8278    5942    302     7047    1671    4999    3330    811     3876    0
44 ==> command 9) BUILD_MST
45 Success

46 ==> command 10) PRINT_MST
47 ( Athens, Hanoi ), 1231
48 ( Hanoi, Busan ), 1105
49 ( Busan, LA ), 943
50 ( LA, Roma ), 1123
51 ( Roma, NewYork ), 546
52 ( NewYork, Incheon ), 1659
53 ( Incheon, Paris ), 860
54 ( Paris, Seoul ), 811
55 ( Seoul, Chicago ), 302
56 Total: 8580
57 ==> command 11) EXIT
58 Success

```

---

Command.txt에서 볼 수 있듯이 각 명령어들을 한번 씩 실행해보았다. LOAD는 city\_list.txt에서 데이터를 가져와 insert 함수를 통해 avl tree를 생성한다. PRINT\_AVL는 생성된 AVL Tree를 inorder로 순회하며 오름차순으로 출력한다. DELETE\_AVL은 인자로 받은 num값과 동일 한 노드를 Tree에서 삭제한다. INSERT는 도시의 데이터들을 받아서 AVL



Tree에 노드를 삽입한다. SEARCH\_AVL은 인자로 받은 num값과 동일한 node를 찾아서 그 node의 데이터들을 출력해준다. BUILD\_GP 는 AVL Tree로부터 데이터들을 전달받아 완전 연결 그래프를 생성해준다. PRINT\_GP 는 만들어진 그래프들의 weight들을 정사각형 행렬로 출력해준다. BUILD\_MST는 graph의 데이터들을 가지고 Minumum Spanning Tree를 생성해준다. PRINT\_MST는 MST의 data들을 from node ,to node, weight 순서대로 연결하면서 출력해준다. 스크린샷을 보면 확인할 수 있듯이 모든 명령어들이 제대로 작동하고 있다.

### < Consideration >

이번 과제를 하면서 map에 대한 개념을 알게 되었다. 사실 graph를 만들면서 굉장히 많이 공부하고 사용법을 익혀야 하는 개인적으로 까다로운 library였던 것 같다. 그렇지만 graph를 만들때는 map을 대체 할 다른 좋은 library가 없는 것 같다. 또한 이번 과제를 하면서 그동안 귀찮게만 느껴졌던 파일 입출력 시스템의 편의성을 깨닫게 되었다. 이번 프로젝트처럼 규모가 좀 크게 되면 데이터들을 file에 저장하거나 file에서 꺼내 오는 것이 더 편한 것 같다. 그리고 graph를 만들면서 from node와 to node, weight를 멤버로 가지고 있는 Edge class를 선언하게 되었다. 이 class를 만들기까지 많은 생각이 필요했다 어떤 정보를 저장할 때, 그 노드가 저장 할 데이터의 종류가 많게 되면 class를 만들어 public member에 데이터들을 정리하는게 편함을 깨달았다. 이번 프로젝트를 진행하면서 가장 어려웠던 것은 MST를 출력할 때 었다. 출발 노드와 끝 노드를 찾는 것, from node와 to node에 대한 정보를 가져와서 연결하면서 출력해야 하는 것이 굉장히 까다로웠다. 이 알고리즘을 생각하는 것이 가장 많은 시간을 잡아 먹었음은 확실하다. 그리고 insert 하는 것은 강의 자료의 pseudo code 덕분에 쉬웠지만, avl deletion이 너무 어려웠다. 모든 mBF를 갱신하고 균형이 어긋난 노드를 찾고 있으면 회전을 시킨 후 다시 mBF를 갱신하는 알고리즘을 생각하는 것은 어렵지 않았으나, 이를 코드로 구현하자니 많이 힘들었다. 진행했던 프로젝트 중에 이번 프로젝트에서 가장 많은 것을 배웠다.