

데이터구조설계 및 실습 프로젝트

학과:컴퓨터정보공학부

학번:2018202074

이름:김상우

Introduction

프로젝트를 크게 요약하면 다음과 같다.

AVL Tree를 만드는 명령어 실행 시 txt파일로부터 NodeLocalID, cityname, country name에 대한 데이터를 받고 그걸 바탕으로 AVL Tree를 제작한다. 이때 AVL Tree는 도시 이름을 나타내는 name을 기준으로 만들어지며 사전 순으로 적을수록 왼쪽으로 간다.(단 대문자와 소문자도 고려하며 실제 기준은 아스키코드이다.)

명령어에 의해 AVL Tree는 추가적인 Data를 받는 것, LocalID를 통해 Data를 찾는 것, 특정 LocalID를 가진 Node를 제거하는 것이 가능하다.

이후 Graph를 제작하라는 명령어가 들어온다면 AVL Tree의 LocalID를 이용해 Graph를 만든다. 이때 Graph의 node로 AVL Tree의 노드를 사용하며 graph내부에는 모든 Node간의 거리를 표현한다. (이때 Graph는 모든 노드들이 연결되어 있는 완전 연결 Graph가 됨을 주의해야 한다.)

이후 MST를 제작하라는 명령어가 들어온다면 Graph의 값을 이용해 MST를 만든다. 이때 MST는 1차원 배열로 MstMatrix[node]=connect_node로 표현된다.

AVL Tree, Graph, MST는 모두 출력을 요구하는 명령어가 존재한다.

위에서 명명된 명령어들은 다음과 같다.

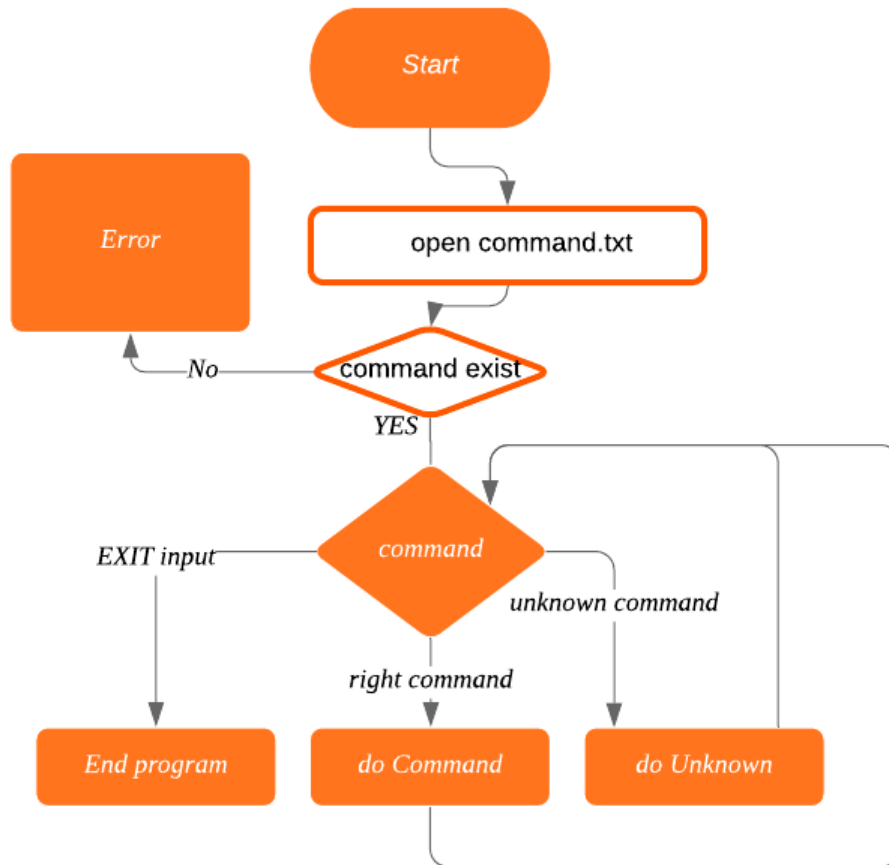
명령어	명령어 설명
LOAD	사용법) LOAD City_list.txt을 읽고 cityname순으로 AVL TREE를 제작한다. 입력되는 LocalID는 1000~9999사이 데이터는 언제나 3개 다 들어옴. error(에러코드 100출력):텍스트파일 존재X 트리가 이미 존재
INSERT	사용법) INSERT LocalID Cityname Countryname 존재하는 AVL Tree에 새로운 정보를 추가한다. Error(에러코드 200출력):AVL tree가 존재 X
PRINT_AVL	사용법)PRINT_AVL AVL tree 탐색 후 저장된 모든 정보를 log.txt에 출력한다. 이때 출력은 오름차순으로 한다. Error(에러코드 300출력):AVL tree가 비어있음
SEARCH_AVL	사용법)SEARCH_AVL LocalID 입력된 도시 위치 정보를 가진 도시 정보의 데이터를 찾는다. 이는 이 데이터(LocalID, cityname, countryname)는 log.txt에 출력된다.

	Error(500출력):입력된 정보가 존재하지 않는다. 위치정보가 입력되지 않았다.
DELETE_AVL	사용법)DELETE_AVL LocalID AVLtree탐색 후 도시위치 정보가 저장된 데이터 삭제 단 AVLTree인 상태를 유지해야만 한다. Error(에러코드 400출력)입력된 위치정보를 가진 정보가 없을 경우
BUILD_GP	사용법)BUILD_GP AVL tree에 저장된 정보들로 완전 연결 Graph를 제작한다. 이미 만들어진 그래프 존재시 지우고 재생성한다. 노드번호는 도시 이름의 오름차순으로 설정한다. Error(에러코드 600출력)AVL tree에 도시정보가 없는 경우
PRINT_GP	PRINT_GP 행렬형식으로 Graph의 연결상태를 log.txt에 출력한다. Error(에러코드 700출력)Graph에 저장된 정보가 없는 경우
BUILD_MST	사용법)BUILD_MST Graph정보를 이용해 최적의 가중치로 모든 노드를 연결하는 tree생성 Kruskal 알고리즘을 사용해야 하며 MST가 존재 시 지우고 생성해야 한다. Error(에러코드 800출력)Graph에 저장된 정보가 없는 경우
PRINT_MST	사용법)PRINT_MST 노드의 연결 순서대로 시작 도시와 끝 도시, 도시간 길이를 출력 최종적으로는 모든 간선의 합을 출력한다. Error(에러코드 900출력)MST에 저장된 정보가 없는 경우
EXIT	사용법)EXIT 시스템 종료

Flowchart & Algorithm

아래 알고리즘 들에서 맨처음의 확인은 개별의 변수를 통해 확인하고 있다.

-Command input



큰 틀은 다음과 같습니다. Command.txt로부터 명령어를 받고 command를 확인해 각각 알맞은 출력을 냅니다. 명령어에 따른 프로그램의 입력 등은 위의 Introduction을 따릅니다.

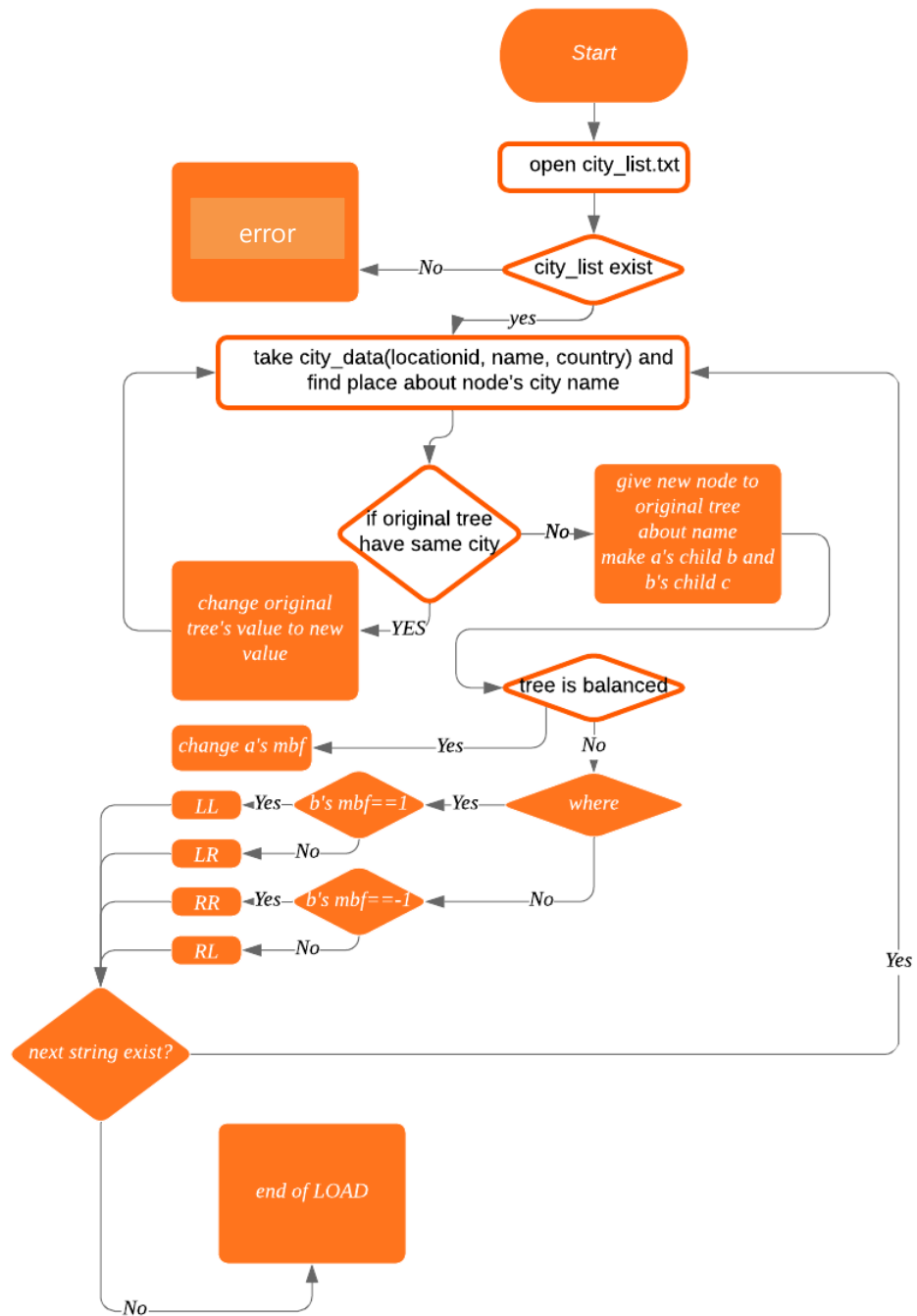
Load부터 PRINT_MST까지가 right command에 해당합니다. Strcmp를 통해 명령어를 확인하고 그에 알맞은 시스템을 실행합니다.

Unknown command가 입력된 경우 Unknown이라는 것을 출력해줍니다. 단 error 출력을 해주는 함수로 출력한다.

EXIT가 명령어로 들어온 경우 코드를 종료합니다.

Right command와 unknown command가 들어온 경우 다시 다음 줄로 넘어가게 되며 이와 동시에 새로운 command를 인식하여 EXIT가 나올 때까지 재실행하게 됩니다.

-LOAD



-Manager.cpp

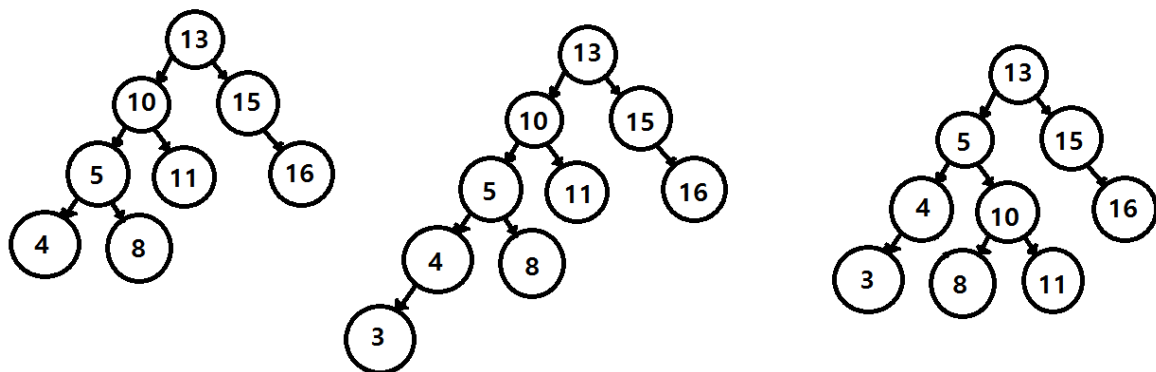
우선 city_list.txt를 읽어옵니다. 텍스트파일이 존재하지 않는다면 error를 출력하고 코드를 종료합니다. 열린 txt파일에서 한 줄씩 값을 읽어옵니다. 한 줄 마다 LocalID, name, country에 해당하는 값들이 존재하며 이들은 tab을 기준으로 나열되어 있기 때문에 strtok(ptr, "wt")를 통해 3개의 데이터로 분류해줍니다. 그 결과로 문자열을 갖게 되므로 숫자로 표현해야 하는 LocalID의 경우 atoi를 통해 int의 형태로 바꾸어 줍니다. 이 3개의 Data를 CityData 내에 넣고 이를 AVLtree내의 Insert함수를 이용해 값을 넣어준다.

-AVLTree.cpp

Citydata*를 받는 Insert함수를 사용한다. 위 함수에선 root가 NULL일 경우 받은 값을 mBF가 0인 root로 만들어 준다.

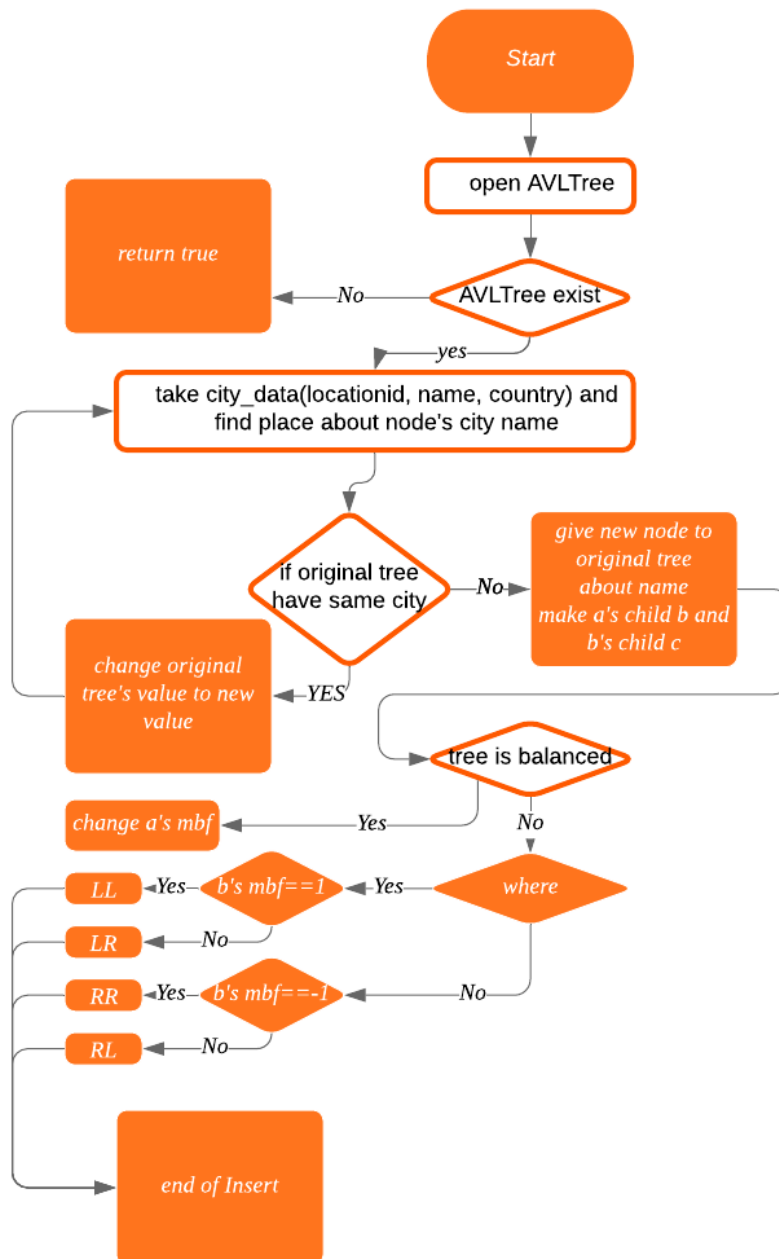
만약 root가 null이 아닌 경우 p를 root로 해준 뒤, cityname끼리 비교하여 적절한 위치에 배치해 줍니다. 만약 이미 root내에 값이 있다면 값을 덮어 씌워줍니다.

이후 높이 계산을 한 후 balance가 유지되고 있다면 mBF 값을 바꾸고 종료한다. 만약 왼쪽이 balance하지 않다면 d의 값을 이용해 d가 1일 때는 LL을, 아닐 때는 LR을 진행하고 오른쪽이 balance하지 않다면 d의 값을 이용해 d가 -1일때는 RR을, 아닐 때는 RL을 진행한다. 이 때 LR 혹은 RL을 진행한다면 c의 mBF값을 switch를 이용해 a와 b의 mBF값을 조율해준다. 이후 경우에 따라 rootsub를 SetLeft혹은 SetRight를 이용해 정돈해 줌으로서 완성할 수 있다.



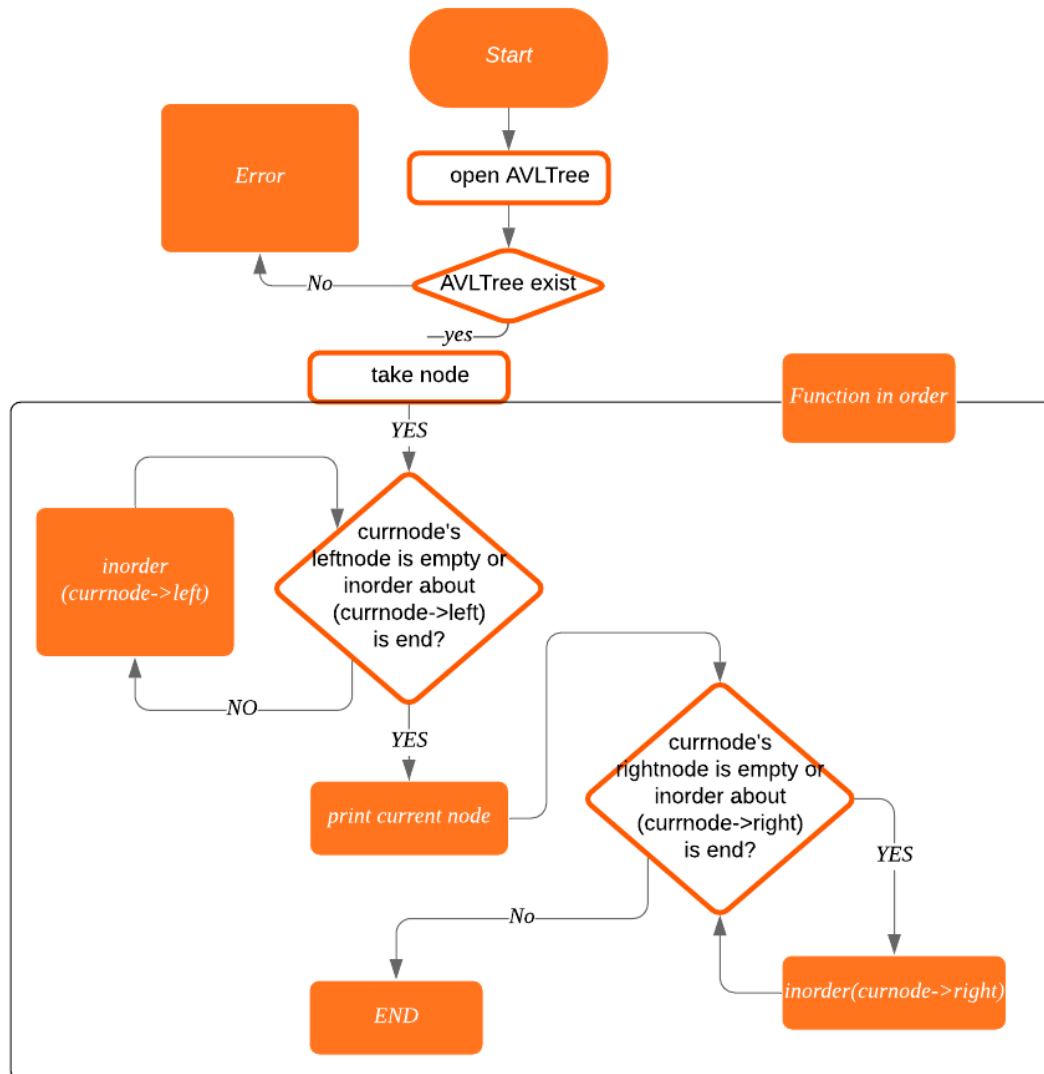
다음은 AVL Tree에서 insert를 해준 예시이다. 3이 insert되면 2번째 그림과 같이 된다. 다만 이 경우 밸런스가 안맞게 된다. 10노드가 -2, 11노드가 -2가 되므로 변경이 필요하다. 이후 위 알고리즘에 따라 10에서 right rotation을 통해 돌리고 그렇게 balance가 맞게 되는 것을 확인할 수 있다.

-Insert



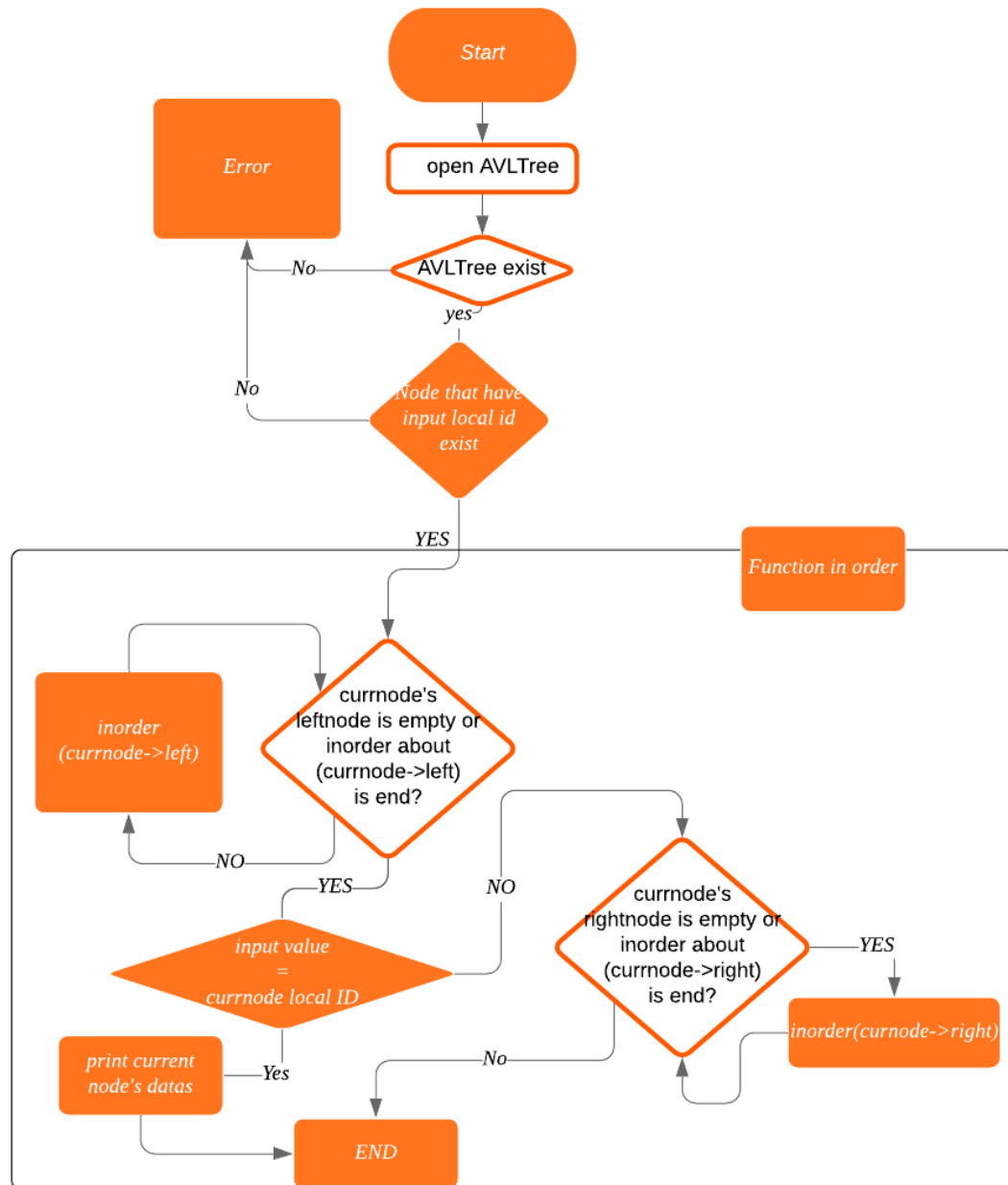
Insert는 load와 유사함을 알 수 있다. 위 경우 명령어를 받아들인 후 AVLTree가 있는지 확인한다. 만약 존재한다면 명령어 이후 있는 문자열을 Data로서 받는 CityData를 avl->Insert함수를 통해 넣어준다. 이후 내용을 LOAD와 동일하다.

-PRINT_AVL



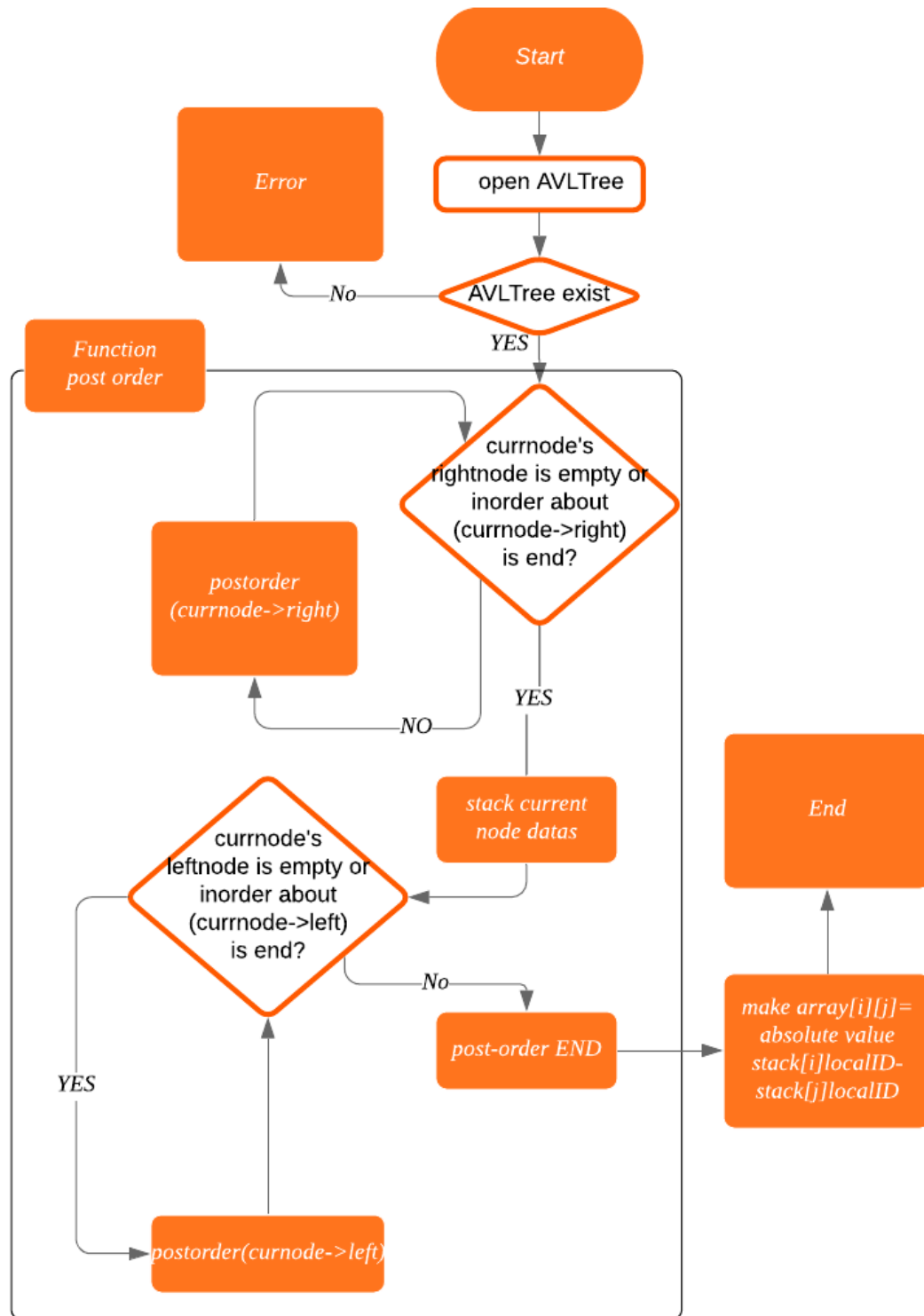
PRINT_AVL의 경우 load의 결과로 AVLTree가 만들어졌는지 확인한다. 만약 있다면 inorder를 통해 AVL Tree 전체를 출력한다. Inorder는 Inorder(왼쪽노드)실행, 현재파일에서의 실행, Inorder(오른쪽 노드)실행 순으로 이루어진다.

-SEARCH_AVL

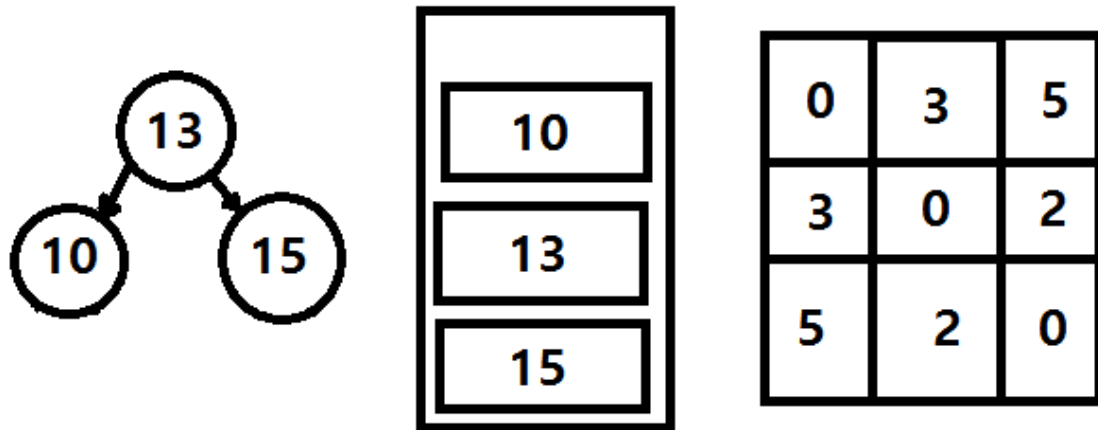


위 flowchart와 같이 PRINT_AVL과 비슷하게 진행됨을 확인할 수 있습니다. Inorder의 형식으로 전체 AVL Tree를 살핍니다. 만약 입력된 값과 받은 node와 같은 값이 들어있는 노드를 출력해줍니다. 이때 error체크를 위해 node와 같은 값이 들어있는 노드가 있는 지 확인을 먼저 해주고 이를 실행합니다.

-BUILD_GP



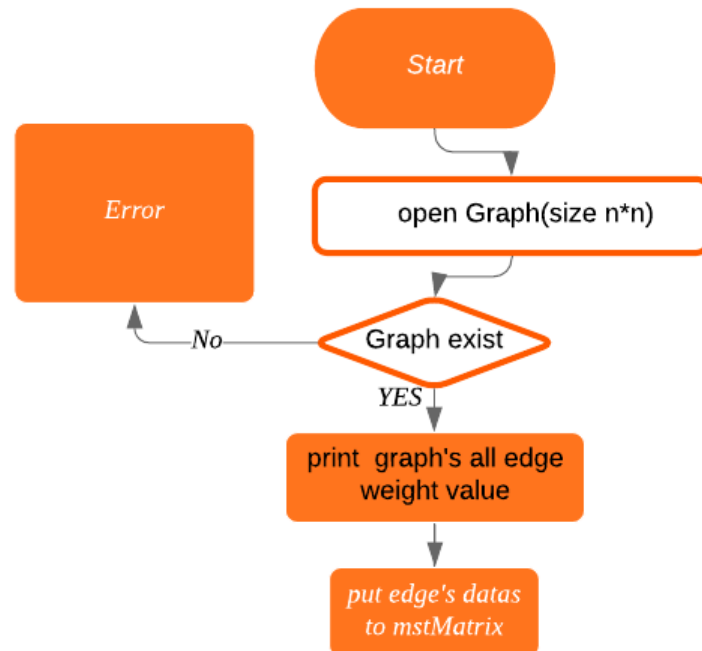
위 플로우차트와 같이 AVLTree가 만들어 주었는지 확인합니다. 이후 post-order를 통해 모든 Node의 값들을 Stack의 형태로 받아옵니다. 이 과정에서 stack의 top과 pop을 이용해 값들을 받아오게 됩니다. 이후 이 값들을 바탕으로 array[n][n]을 제작하고 각각의 array[i][j]에는 stack[i]의 LocalID-stack[j]-LocalID의 절댓값을 넣어줍니다.



아래의 경우를 예를 들어보면 후위 연산을 통해 오른쪽부터 15, 13, 10 순서로 들어오게 된다. 다만 Stack의 특성으로 늦게 들어온 값이 먼저 나가므로 10이 0, 13이 1, 15가 2에 해당하게 되며 이들의 차이의 절댓값을 통해 오른쪽의 그래프가 작성된다.

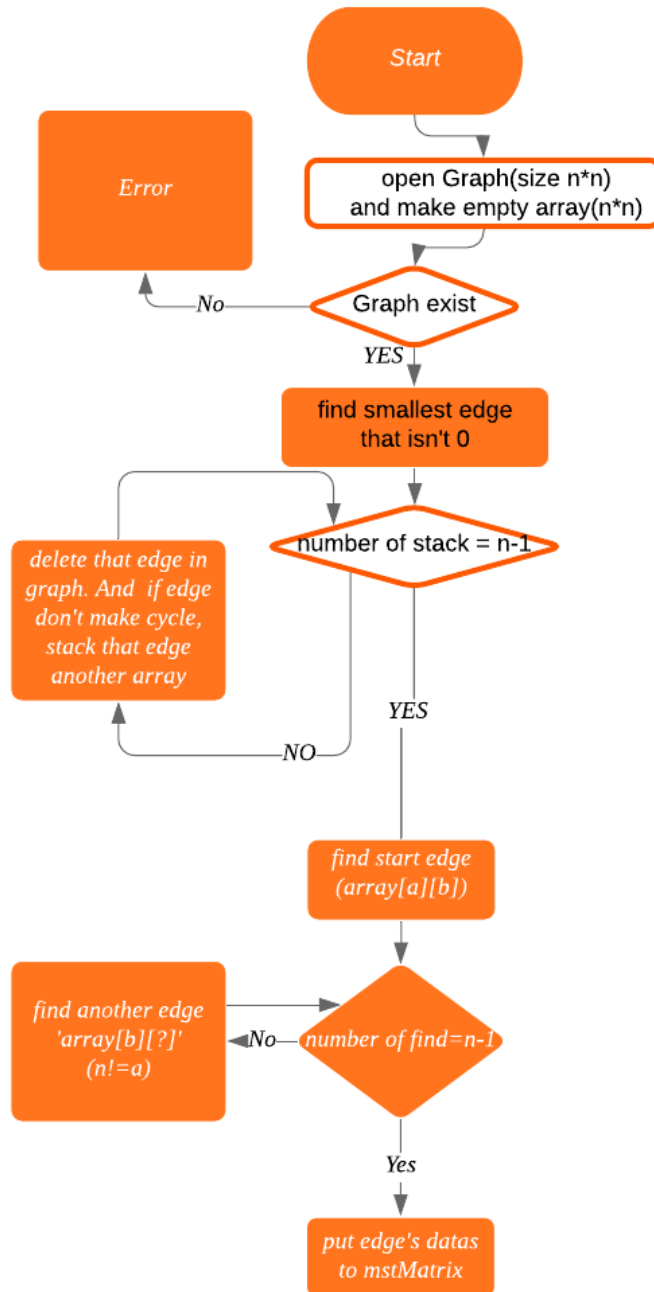
10-10	13-10	15-10
10-13	13-13	15-13
10-15	13-15	15-15

-PRINT_GP



Graph가 있는 지 없는지 확인하고 있을 경우 $N \times n$ 의 배열을 2중반복문을 통해 출력한다. 이 과정에서 `mList[i].begin`과 `mList[i].end`를 통해 시작과 끝을 확인한다. 또한 `*flog<<mList[i]->second<<" "`; 등의 형태로 출력을 한다.

-BUILD_MST



우선 Graph를 열고 Graph가 있는지 확인한다. 만약 Graph가 있다면 Graph와 크기가 같은 빈 행렬을 만든다. 이후 Graph를 2중 반복문을 통해 둘러보면서 0이 아닌 가장 작은 값을 가지는 그래프의 크기와 위치를 확인한다. 만약 그 edge를 추가했을 때 cycle을 구현하지 않는다면 비어있는 행렬에 이를 추가해준다. 이 추가를 (node의 개수-1)만큼 반복한다.

그 결과로 생성된 빈 행렬에서 1개의 edge를 가지는 행을 찾는다. 그 중 위에 있는 행이 node의 시작이 된다. 이 edge의 열에 해당하는 숫자를 행으로 가지는 줄에서 이전 edge의 행을 열로서 가지는 edge가 아닌 edge를 선택한다. 이후 이를 (node의 수 -1)만큼 반복한다. 이때 방문한 행의 순서대로 mstMatrix2를 생성한다. 이후 mstMatrix2[i]와 mstMatrix2[i+1]의 값을 이용해서 mstMatrix를 만든다.

이때 Cycle의 확인은 다음과 같이 한다.

1.X,y값을 받는다.

2.x와 y의 값을 기존의 배열과 비교한다.

X와 y모두 어떠한 배열에도 존재하지 않는다.->x와 y를 가지는 새로운 배열생성

X만 특정 배열에 존재->X을 가지는 배열에 y를 추가한다.(X와 Y를 바꿔도 성립)

X와 Y가 각각 다른 배열에 존재한다.->먼저 X가 속한 배열과 Y가 속한 배열 중 먼저 생성된 배열에 나중에 생성된 배열 전체를 넣어준다.

X와 Y가 같은 배열에 존재한다.->Cycle발생

3.insert의 횟수가 (node의 수-1) 이상이고 배열이 하나가 되었다면 종료한다. 아니면 다음값을 받으러 1로 돌아간다.

예시로 만약 (1,2) ,(2,3), (1,3), (2,4)라는 값을 순서대로 넣는다고 가정하면 다음과 같이 나온다..

1	2		

1,2모두 배열에 없으므로 배열 생성

1	2	3	

2가 배열에서 존재하므로 배열에 3추가

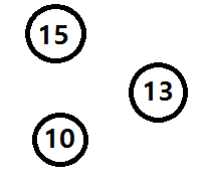
1	2	3	

1,3모두 같은 배열에 존재하므로 cycle발생

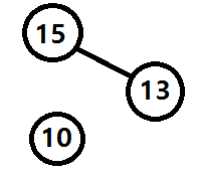
1	2	3	4

2가 배열에서 존재하므로 배열에 4 추가

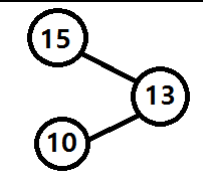
이 경우를 바탕으로 다음 그래프를 크루스칼을 통해 mst로 바꾸면 아래와 같이 된다.

0	3	5	
3	0	2	
5	2	0	

가장 작은 0이 아닌 값=2, cycle생성은 안함

0	3	5	
3	0	0	
5	0	0	

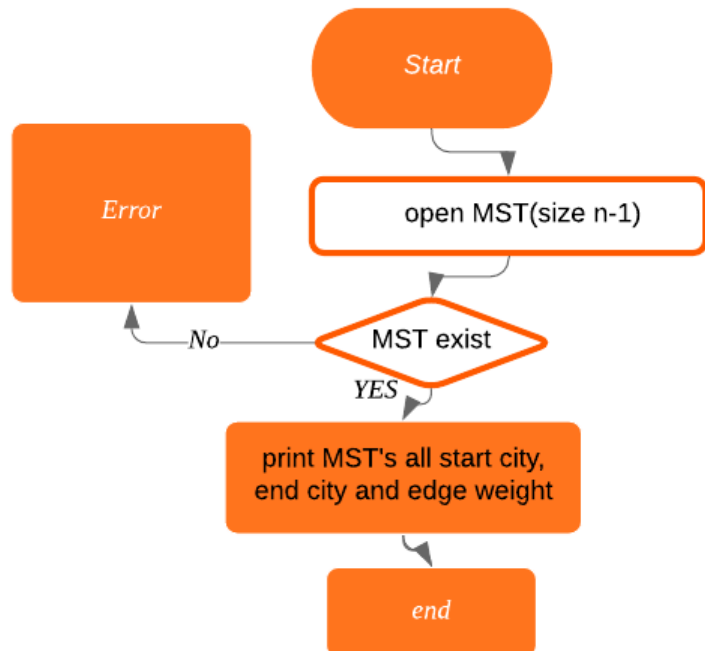
가장 작은 0이 아닌 값=3, cycle생성은 안함

0	0	5	
0	0	0	
5	0	0	

2번(3-1)진행했다. 고로 graph는 완료

(2,3)(2,1)순으로 들어왔으니 2->3->1의 식으로 들어왔다. 고로 배열은 각각 1,3,1이 들어간다.

-BUILD_MST



MST가 존재하는 지 확인하고 존재한다면 각각의 값을 받는다. MstMatrix2의 값을 이용, 위치 좌표를 바탕으로 vertex에서의 위치를 탐색하고 Getname()을 통해서 이름을 받아 출력한다. 이후 graph에서의 좌표를 찾아서 출력해준다. 또한 변수를 추가해서 graph를 통해 출력된 결과들을 계속해서 더하며 값들의 합을 구해준다.

Result Screen

1	LOAD	1	1543	Seoul	Korea
2	INSERT 7474 Incheon Korea	2	7485	Busan	Korea
3	PRINT_AVL	3	2132	Tokyo	Japan
4	SEARCH_AVL 2132	4	4873	NewYork	USA
5	BUILD_GP	5	6542	LA	USA
6	PRINT_GP	6	1241	Chicago	USA
7	BUILD_MST	7	2354	Paris	France
8	PRINT_MST	8	5419	Roma	Italy
9	EXIT	9	9821	Athens	Greece
		10	8590	Hanoi	Vietnam

다음과 같은 comment와 city list는 다음과 같은 내용으로 진행된다.

```
==> command 1) LOAD
Success
==> command 2) INSERT
Success
==> command 3) PRINT_AVL
(9821, Athens, Greece)
(7485, Busan, Korea)
(1241, Chicago, USA)
(8590, Hanoi, Vietnam)
(7474, Incheon, Korea)
(6542, LA, USA)
(4873, NewYork, USA)
(2354, Paris, France)
(5419, Roma, Italy)
(1543, Seoul, Korea)
(2132, Tokyo, Japan)
==> command 4) SEARCH_AVL
(2132, Tokyo, Japan)
```

SEARCH_AVL까지의 결과는 다음과 같다. LOAD와 INSERT가 진행된다.

이후 PRINT_AVL에 Incheon, Korea가 추가되어 있으며 city의 순서는 위에서부터 사전 순으로 배치되어 있음을 알 수 있다. 또한 SEARCH_AVL 2132로 2132를 LocalID로 가지는 Tokyo의 데이터들이 출력된다.

```
==> command 5) BUILD_GP
Success
==> command 6) PRINT_GP
0 2336 8580 1231 2347 3279 4948 7467 4402 8278 7689
2336 0 6244 1105 11 943 2612 5131 2066 5942 5353
8580 6244 0 7349 6233 5301 3632 1113 4178 302 891
1231 1105 7349 0 1116 2048 3717 6236 3171 7047 6458
2347 11 6233 1116 0 932 2601 5120 2055 5931 5342
3279 943 5301 2048 932 0 1669 4188 1123 4999 4410
4948 2612 3632 3717 2601 1669 0 2519 546 3330 2741
7467 5131 1113 6236 5120 4188 2519 0 3065 811 222
4402 2066 4178 3171 2055 1123 546 3065 0 3876 3287
8278 5942 302 7047 5931 4999 3330 811 3876 0 589
7689 5353 891 6458 5342 4410 2741 222 3287 589 0
==> command 7) BUILD_MST
```

위는 BUILD_GP 와 PRINT_GP 가 진행된다. 보면 알 수 있듯이 각 LocalID간의 차로 이루어져 있다. 대표적으로 [1][0]과 [0][1]에 존재하는 2336의 경우 9821-7485로 진행되어 생성된 것이다.

```

==> command 7) BUILD_MST
Success
==> command 8) PRINT_MST
(Athens, Hanoi), 1231
(Hanoi, Busan), 1105
(Busan, Incheon), 11
(Incheon, LA), 932
(LA, Roma), 1123
(Roma, NewYork), 546
(NewYork, Paris), 2519
(Paris, Tokyo), 222
(Tokyo, Seoul), 589
(Seoul, Chicago), 302
Total: 8580

```

위 그래프를 바탕으로 진행된 BUILD_MST와 PRINT_MST를 진행했다. 위 그래프를 통해 0이 아닌 가장 작은 값인 11이 들어가 있음을 확인할 수 있다. 이후 kruskal을 직접 해보면 위 같은 결과가 나옴을 알 수 있다. Total도 위 값들을 합치면 나오는 것을 확인할 수 있다.

-Consideration

이 프로젝트를 진행하면서 데이터구조설계 및 실습에서 배운 AVLtree작성이나 Graph, 크루스칼 등을 비롯해서 값을 넣는 것을 map과 stack의 형태로 사용하면서 새로운 헤더 파일 및 형식에 대한 지식을 배울 수 있었다. 추가적으로 이전에는 Inorder만을 사용하였던 반면 이번엔 stack을 사용하여 post-inorder를 사용하게 되었다. 이를 통해 이전 지식에 대한 복습 또한 겸할 수 있었다.

프로젝트를 진행하면서 가장 크게 느낀 점은 이 같은 많은 양의 프로젝트를 실전에 나가서도 계속하게 될 것이라는 것이다. 이를 통해 프로젝트 제작에 대해서 계획을 짜는 것에 중요성과 효율적인 배분의 필요함을 느낄 수 있었다.

또한 대부분의 버그를 잡아주는 Window의 visual studio와는 다르게 Linux의 경우 오류를 무시하지 않고 진행하는 것에서 환경 변화에 따른 어려움을 느낄 수 있었다. 대표적으로 txt파일에서 읽어들일 때 마지막 줄을 Linux에서는 추가적으로 읽어들여 'w0'을 한번 더 인식해 주어서 해결해 줄 수 있었다.