Data Structure Project

Project #2

담당교수 : 이기훈

제출일 : 2018. 11.04

학과 : 컴퓨터정보공학부

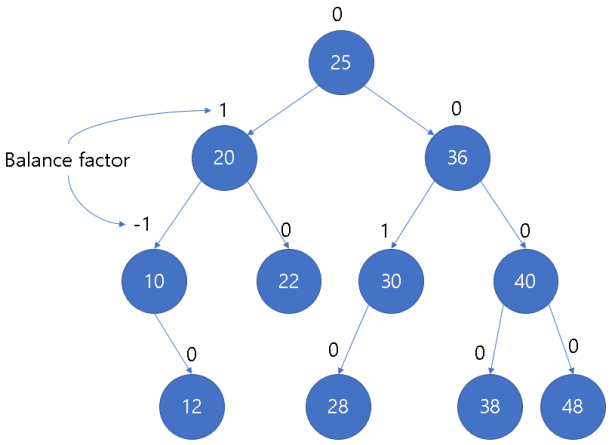
학번 : 2015722025

이름 : 정용훈

1. Introduction

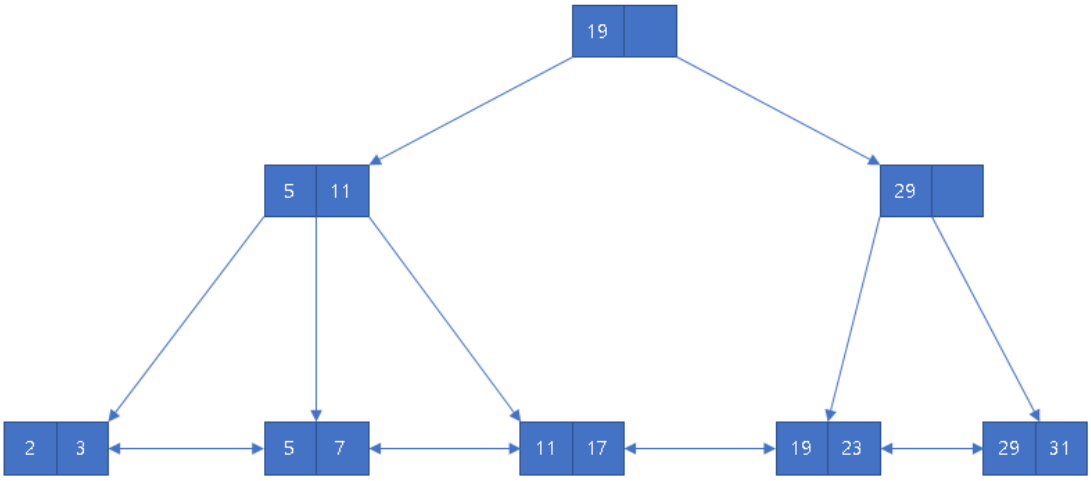
해당 프로젝트는 주식 정보를 정해진 자료구조데로 저장하고 명령어에 맞게 검색하며 출력하는 것을 목적으로 합니다. 사용된 자료구조는 크게 3가지로 나뉘며 AVL tree, Heap, B+ tree로 구성이 되어있습니다. 각 종목의 정보는 stock_list.txt파일에 저장되어 있으며 명령어를 인식하여 받는 정보를 AVL tree와 B+ tree바로 구현하게 됩니다. AVL tree의 경우 종목의 이름을 기준으로 tree가 구현되며 기존 Binary tree와 같은 삽입 과정을 거치다가 트리의 양쪽 높이가 기준이상 차이가 나게 되면 일정 조건에 맞는 rotation을 동작 할 수 있도록 합니다. 즉 기준 노드에서 balance factor가 |2|이상 차이가 나게 되면 Node들의 위치를 바꾸어 항상 tree가 균형이 잡혀 있을 수 있도록 해주는 동작을 뜻합니다. 그리고 AVL tree가 구현되는 동시에 B+ tree도 함께 구현이 되는데 B+ tree란 B tree의 구조와는 다르게 모든 데이터들이 leaf에 남아있는 것을 의미합니다. B tree와 마찬가지로 데이터가 들어오면서 order에 맞게 데이터가 split되며 트리가 구성되지만 leaf를 봤을 때 모든 데이터들이 존재하면 leaf는 linked list로 구성되는 것을 확인할 수 있습니다. 이는 데이터를 검색하는데 굉장히 효율적이며 실제로 이번 프로젝트에서도 B+ tree의 Search 함수를 구현하는 것이 포함 되어있습니다. 마지막으로 Heap의 구현입니다. Heap은 명령어 Load를 통하여 구현되는 것이 아니라 AVL tree에서 Search를 통하여 찾게 된 Node를 Max Heap 형태로 구현하는 것을 목적으로 둡니다. 즉 AVL Search 명령이 없다면 Heap은 아무 정보를 가지고 있지 않은 NULL 상태를 유지합니다. Max Heap이란 자식의 노드가 부모의 노드보다 값이 작기만 하면 성립되는 tree로 아래 그림을 보면 쉽게 이해할 수 있습니다. 또한 AVL tree와 B+ tree와는 다르게 vector를 통해 array와 비슷한 형태로 tree를 구현하는 것을 확인 할 수 있습니다. 또한 이번 프로젝트에서는 데이터의 삭제, 수정은 구현하지 않습니다. 이유는 트리를 구현하는 것 자체로 난이도가 높은 편이기 때문에 삭제 수정 구현은 하지 않고 Search와 Insert구현에 중점을 둡니다.

AVL tree



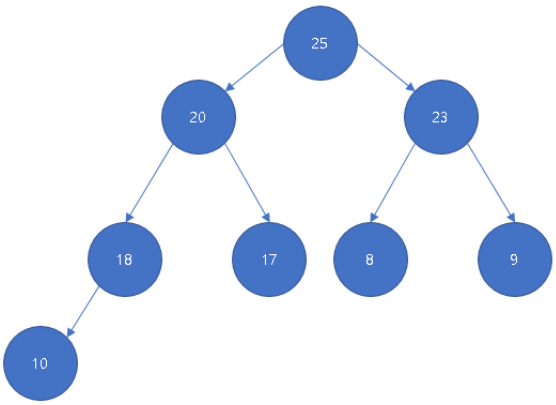
해당 이미지는 구현된 AVL tree로 단순히 봤을 때는 Binary트리와 다르지 않지만 각 노드는 Balance factor를 가지고 있고 factor를 기준으로 rotation을 하게 됩니다. Insert 동작의 자세한 동작 방법은 아래 알고리즘 문항에서 자세히 다루겠습니다. 다음으로는 B+ tree의 구조를 이미지로 나타낸 것입니다.

B+ tree



해당 이미지는 B+ tree의 구조를 나타낸 이미지입니다. 지금까지 배웠던 tree와는 다르게 노드 안에 여러가지 데이터들이 존재할 수 있으며 이외의 구조는 앞서 배운 tree들과 비슷합니다. Leaf를 보면 tree의 전체 data가 담겨있는 것을 확인 할 수 있습니다. 또한 leaf들은 보는 것과 같이 linked list로 연결되어 데이터를 찾는데 좀 더 수월합니다. 삽입과정은 AVL tree와 마찬가지로 알고리즘 문항에서 좀 더 자세하게 다루겠습니다.

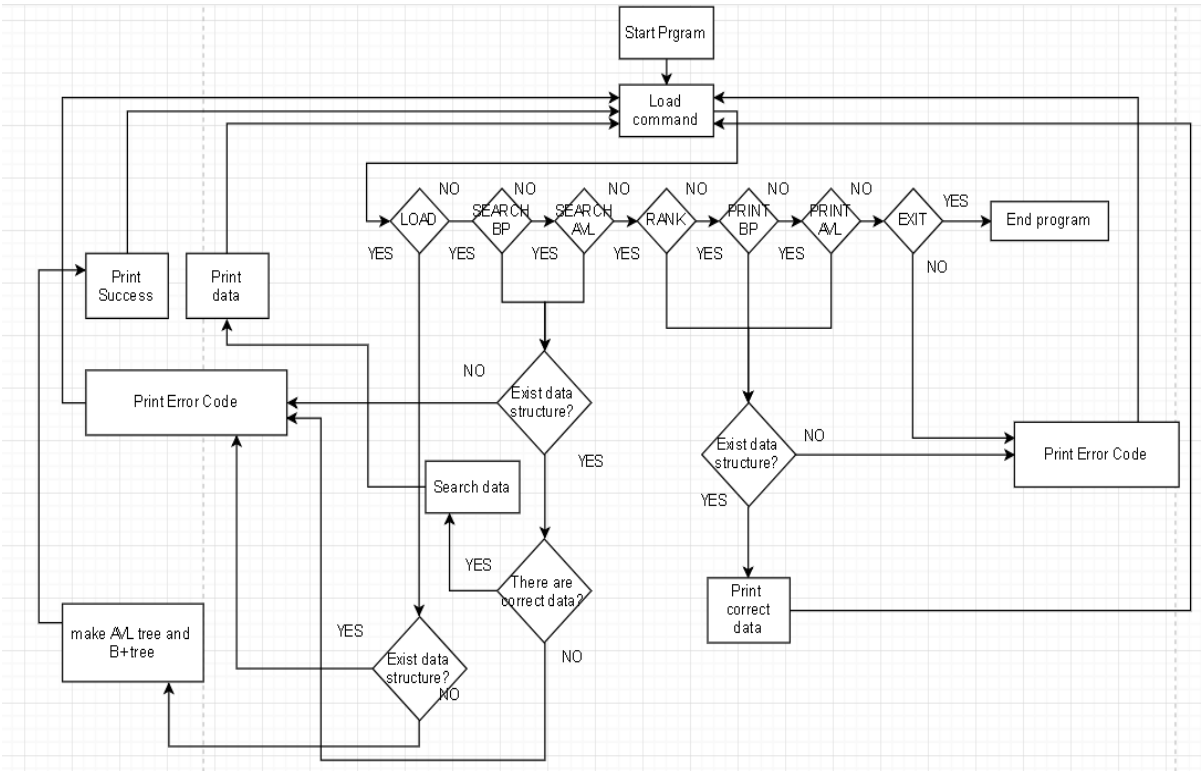
Max heap



해당 이미지는 Max Heap입니다. 단순히 부모 노드가 자식 노드보다 데이터의 크기가 크면 조건을 만족하고 insert를 하게 되면 level순서데로 왼쪽부터 데이터가 들어가기 시작합니다. 만약 insert되는 값이 부모보다 크게 되면 간단하게 swap을 통하여 바꿔주면 문제가 해결됩니다. 이번 프로젝트에서는 가장 구현하기 수월한 자료 구조입니다.

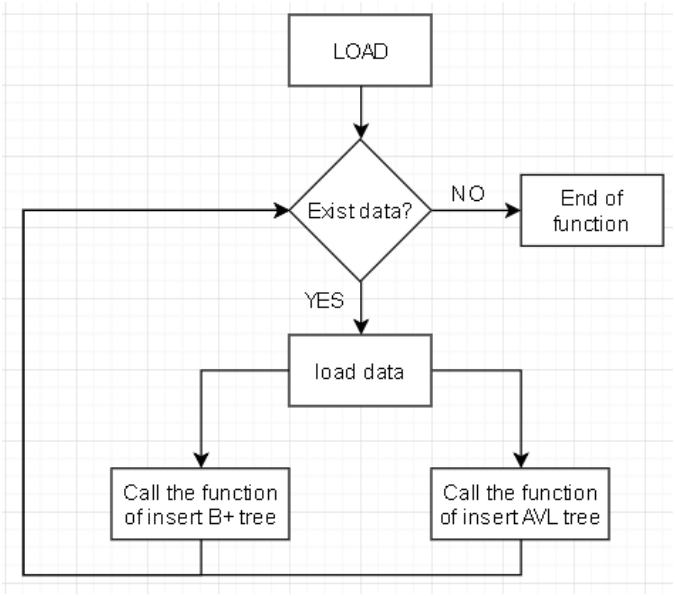
2. Flow Chart

전체 동작에 대한 Flow chart



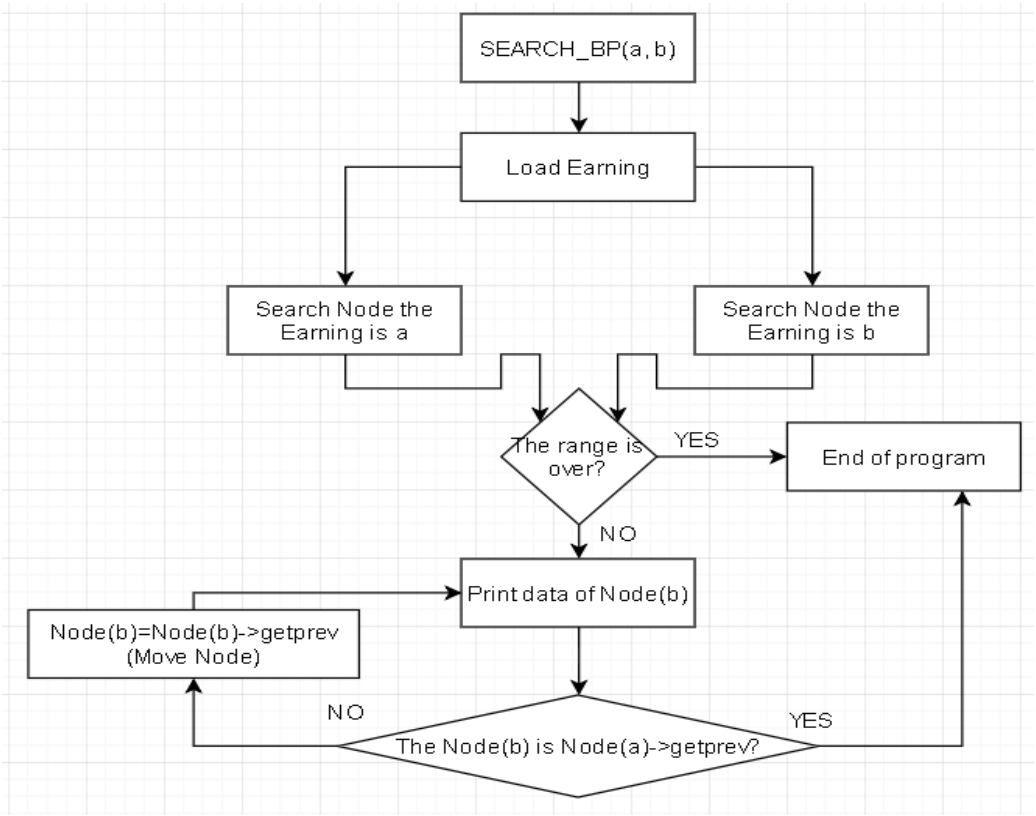
다음 flow chart는 프로그램의 전체적인 동작을 나타내는 이미지입니다. Project1에 비해 구현되는 동작이 적고 Insert, Print, Search의 동작이 알고리즘구현의 대부분을 차지 하고 있기 때문에 chart 가 비교적 간단하게 나온 것을 확인할 수 있습니다. Flow chart를 참고하면 어떤 명령어가 있고 해당 명령어를 받았을 때 어떤 부분에서 예외처리가 되며 명령어가 어떤 동작을 하는지 확인할 수 있습니다. 명령어의 자세한 알고리즘은 3번 문항에서 자세하게 다루겠습니다.

LOAD



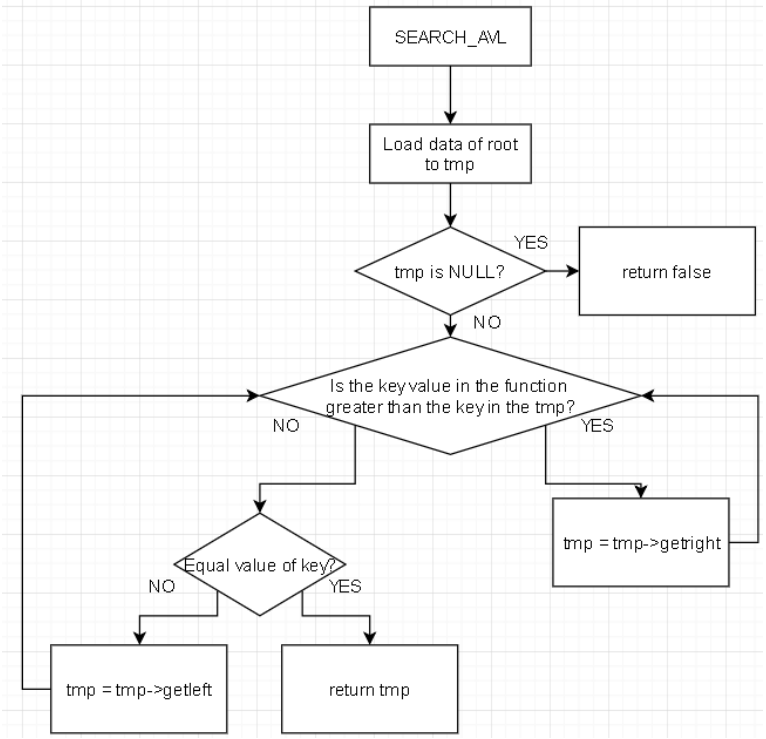
Load 명령어는 data를 담고 있는 txt파일을 불러와 데이터가 존재하는지에 대한 유무를 판단하여 데이터가 있으면 해당 데이터를 기반으로 B+ tree와 AVL tree를 구성할 수 있도록 설계 되어있습니다. 만약 더이상 데이터가 없다면 함수를 종료하게 됩니다.

SEARCH_BP



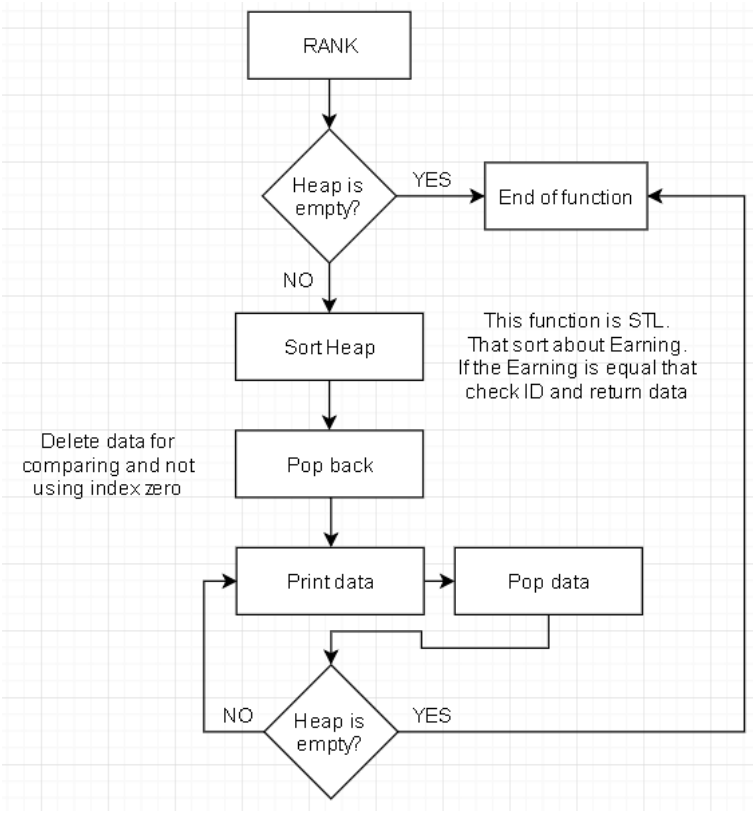
SEARCH_BP에 대한 flow chart입니다. Earning을 통하여 노드를 찾는 과정은 제공받은 코드에서 구현되어 있기 때문에 flow chart를 따로 작성하지 않았습니다. 해당하는 끝 노드의 정보를 Print하는 과정은 iterator를 사용하여 들어온 Earning과 비교하여 출력 여부를 결정하며 범위 안에 있는 노드들의 Print는 멤버가 모두 출력된 경우 노드를 다음으로 넘겨주며 조건에 벗어날 때까지 순차적으로 Print해주면 됩니다. Print함수는 노드에 구성되어 있는 Map이 Stock Data를 가지고 있기 때문에 Print함수에 해당 정보를 넘겨주면 쉽게 정보를 Print할 수 있습니다.

SEARCH_AVL



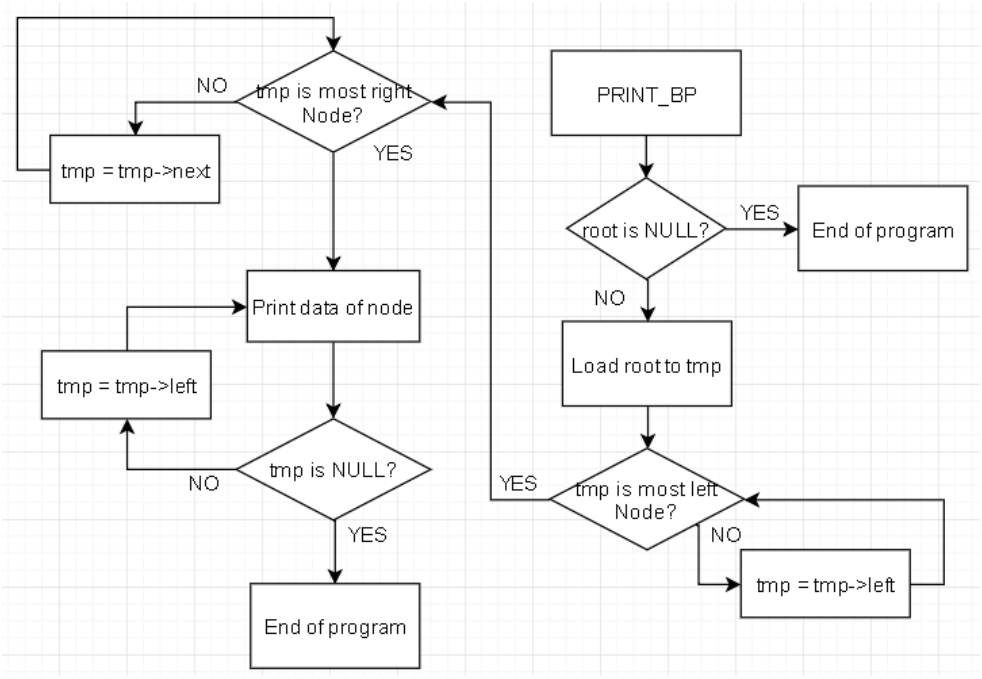
AVL tree의 Search 알고리즘은 프로젝트 1에서도 구현하였던 기본적인 Binary tree의 Search와 동일합니다. 비교대상이 되는 값을 key값이라고 하였을 때 찾게 될 값을 기준으로 노드의 key값이 크면 노드를 왼쪽으로 작으면 오른쪽으로 이동시켜 값이 같을 때 해당 노드를 반환해주면서 함수를 종료해주면 됩니다.

RANK



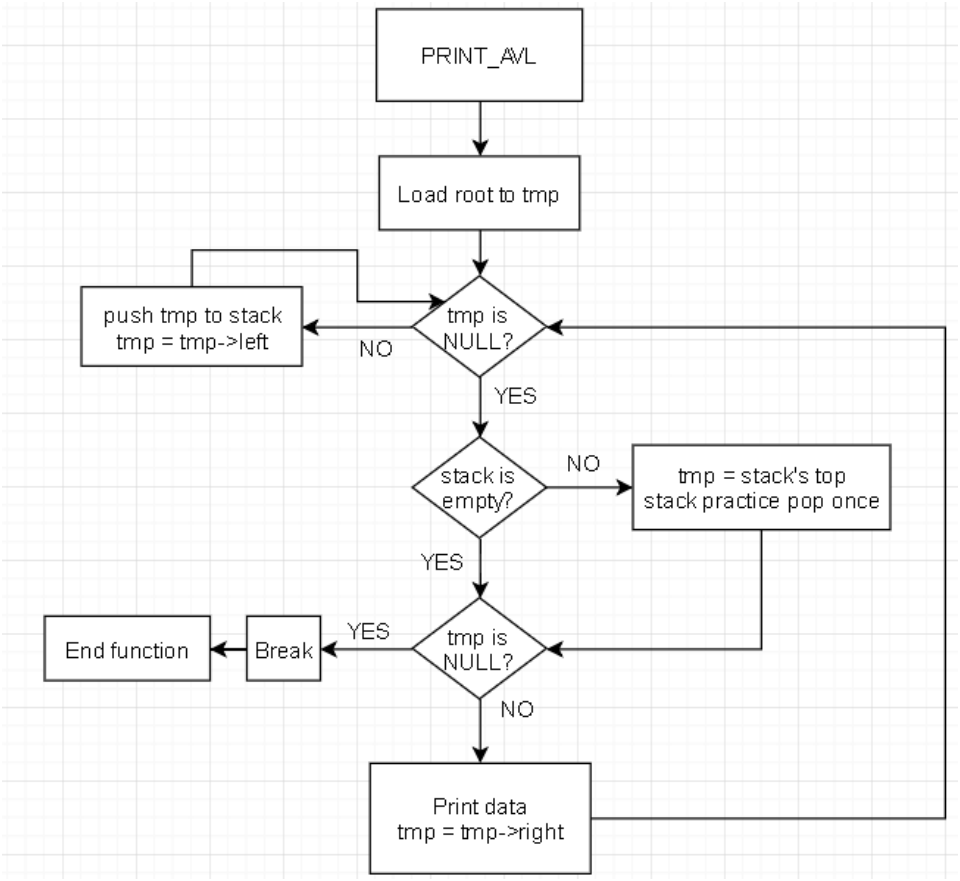
RANK 함수의 대한 Flow chart입니다. 해당 함수에서는 STL과 관련된 sort함수와 Pop함수를 사용하였기에 많은 도움이 되었습니다. 출력을 위해 정보들이 담겨있는 Heap을 STL함수를 사용하여 sort시킨 후 인덱스 번호 0을 사용하지 않기 위해 넣어두었던 정보를 지워준 후 데이터를 Print하기 시작합니다. Print된 정보들은 순차적으로 지워주며 RANK함수가 끝나게 되면 Heap에 정보는 남아있지 않게 됩니다.

PRINT_BP



B+ tree의 Print Flow chart입니다. 왼쪽 끝 노드로 가서 linked list를 통하여 오른쪽 끝으로 pointer를 보내고 순차적으로 데이터를 프린트하며 왼쪽으로 이동시킵니다. (내림차순으로 print해야 하기 때문에) 왼쪽으로 보내고 오른쪽으로 보내는 이유는 root Node에서 오른쪽 노드로 가기 위해서는 Map에 접근해서 이동시켜야 하므로 불편한 부분이 있어 접근이 쉬운 왼쪽으로 노드를 보내 linked list를 통해 오른쪽 끝 노드를 가리키게 만들었습니다. 미리 선언한 tmp노드가 왼쪽 끝 노드에 도착하고 NULL을 가질 때 동안 반복하면 B+ tree의 모든 정보를 Print할 수 있습니다.

PRINT_AVL



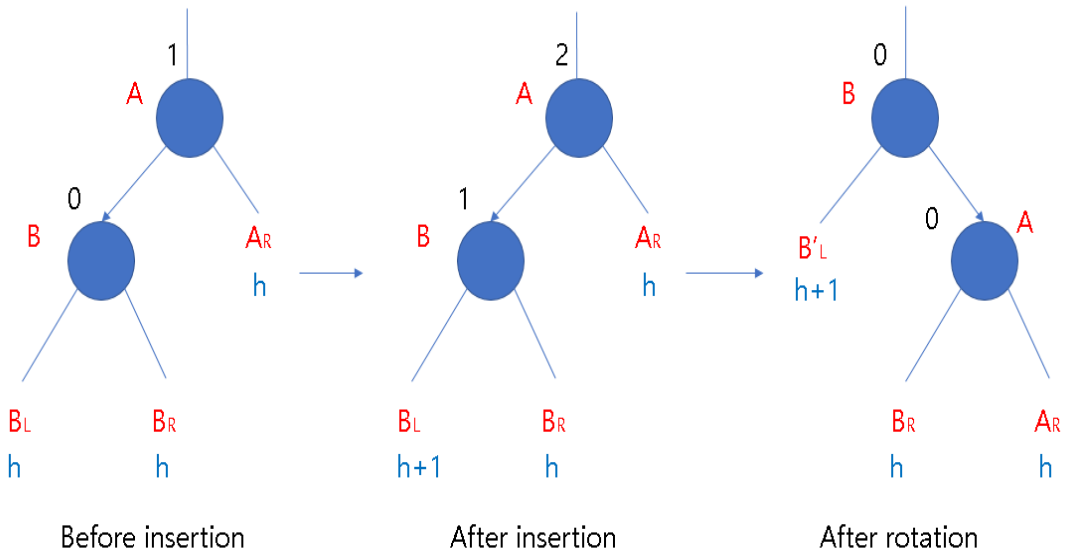
AVL Tree의 Print 동작입니다. 재귀함수를 사용한 방법이 아니라 반복문을 사용하여 Stack을 필수적으로 사용하였습니다. (처음 주어진 코드에서 Stack이 선언 되어있었습니다.) 눈으로만 알고리즘을 보면 tmp Node의 이동과 출력과정이 헷갈릴 수 있지만 직접 알고리즘을 하나씩 써가며 Stack에 저장되는 정보를 push하고 pop하면서 확인해보면 알고리즘을 쉽게 이해할 수 있습니다. 첫번째 프로젝트에서 다뤘던 함수이기 때문에 익숙한 함수였습니다.

3. Algorithm

Insert AVL

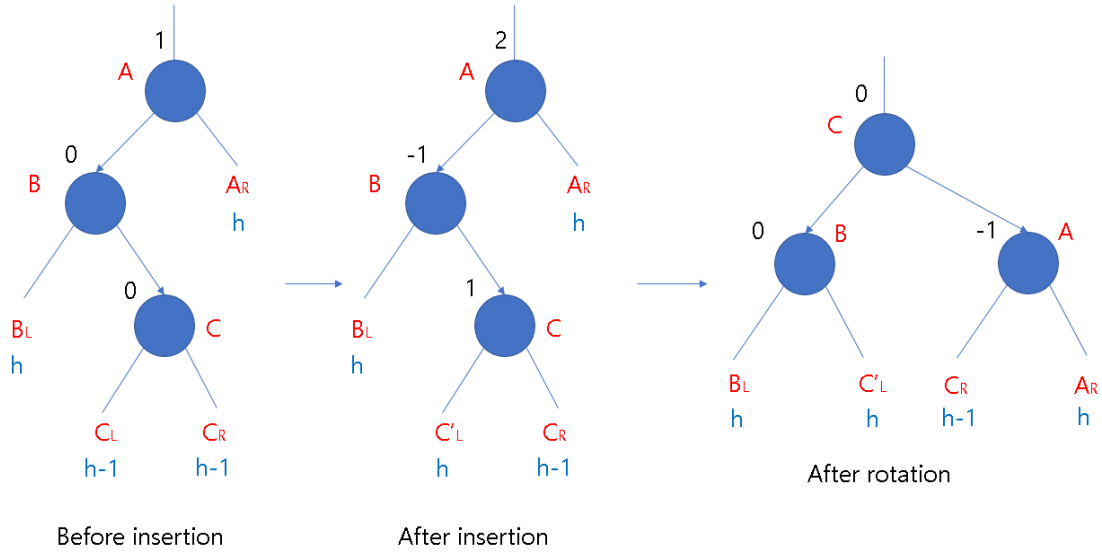
처음 설명할 알고리즘으로는 AVL의 Insert과정입니다. AVL을 Insert하고 Balance factor를 기준으로 rotation을 하는 표현을 사용하는데 개인적으로는 노드들의 위치를 맞는 조건에 바꿔준다는 표현이 저에게는 보다 이해하기 쉬운 표현이었습니다. AVL의 rotation에는 4가지 종류로 나뉘는데 LL, RR, LR, RL입니다. Balance factor를 기준으로 새로 들어온 노드의 위치에 따라 종류가 나뉘며 다음 다룰 내용은 각 경우마다 노드의 위치가 어떻게 바뀌는지 설명하도록 하겠습니다.

(LL rotation)



해당 알고리즘은 수업시간에 사용한 PDF를 참고하였으며 실제로 코드에서도 같은 알고리즘으로 프로그램이 구현된 것을 확인할 수 있습니다. 간단한 설명으로는 A가 가리키는 왼쪽 노드를 B의 오른쪽 노드로 설정하고 B의 가리키는 오른쪽 노드를 A 노드로 설정해주면 위 그림과 같은 방식으로 노드의 이동이 일어나며 Balance가 받게 됩니다. RR rotation도 마찬가지로 방향만 반대일 뿐 동작은 동일합니다.

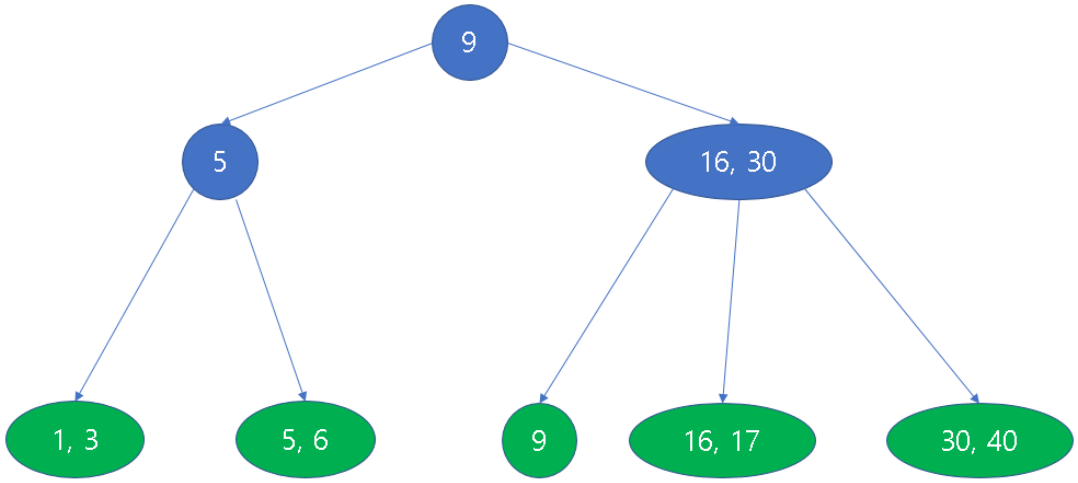
(LR rotation)



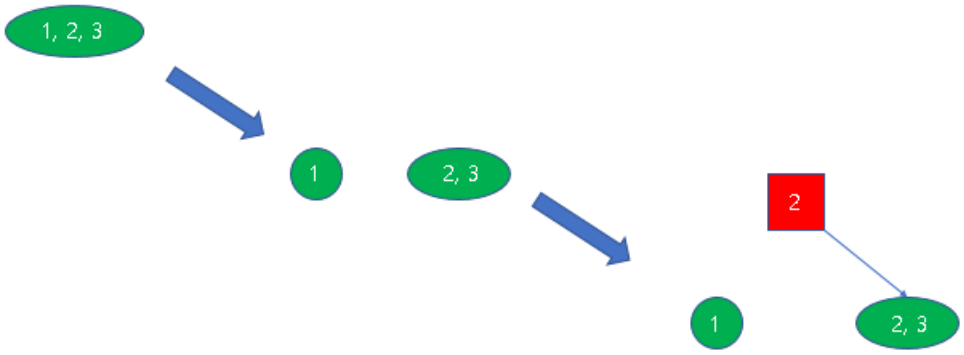
다음 이미지는 LR rotation을 나타낸 이미지입니다. Balance factor를 기준으로 새로 들어오 노드의 위치가 왼쪽->오른쪽에 있기 때문에 LR을 뜻하며 노드의 위치가 바뀌는 알고리즘을 간단하게 설명하면 B노드다 가리키는 오른쪽 노드가 C가 가리키는 왼쪽 노드로 바뀌며 A가 가리키는 왼쪽 노드는 C의 오른쪽 노드로 바뀌게 됩니다. 그 후 C노드를 올려 주어야 하기 때문에 C노드의 왼쪽 노드는 B노드로 설정하며 오른쪽 노드는 A노드로 설정하면 Balance문제가 해결됩니다. 마찬가지로 RL인 경우도 LR인 경우와 동작이 반대로 동작하므로 알고리즘을 이해하는데 크게 무리가 없으며 코드 또한 이와 마찬가지로 동일하기 때문에 오히려 코드를 보고 공부하는 것이 큰 도움이 됩니다.

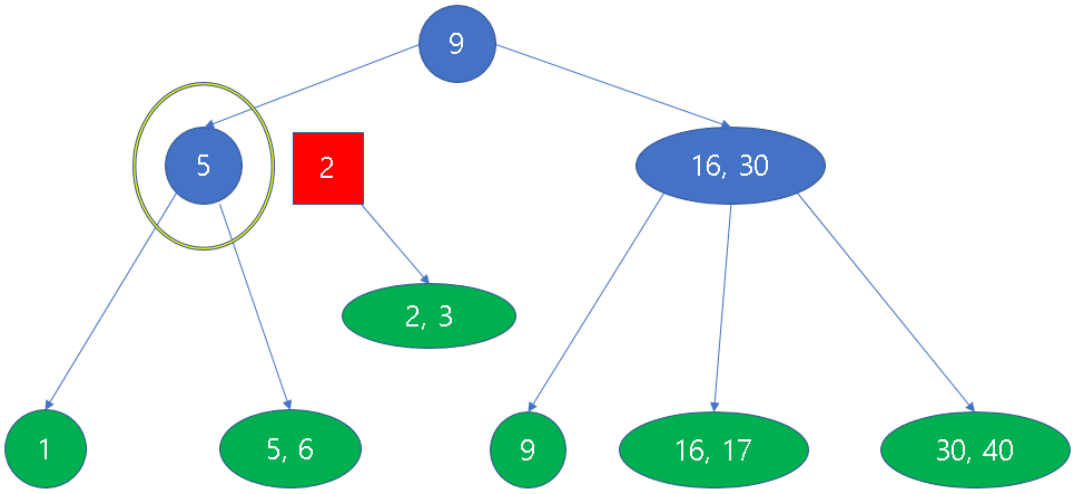
Insert B+

B+ tree의 삽입을 구현하는데 가장 큰 부분은 split입니다. split이란 tree에 삽입된 정보가 있는 노드의 data개수를 판단하여 수용개수를 초과하였을 때 해당 노드의 data중 중간 값을 떼어 위로 올리면서 tree가 형성될 수 있도록 하는 방식을 뜻합니다. 다음 그림의 순서는 B+ tree의 Insert과정을 나타낸 것입니다.

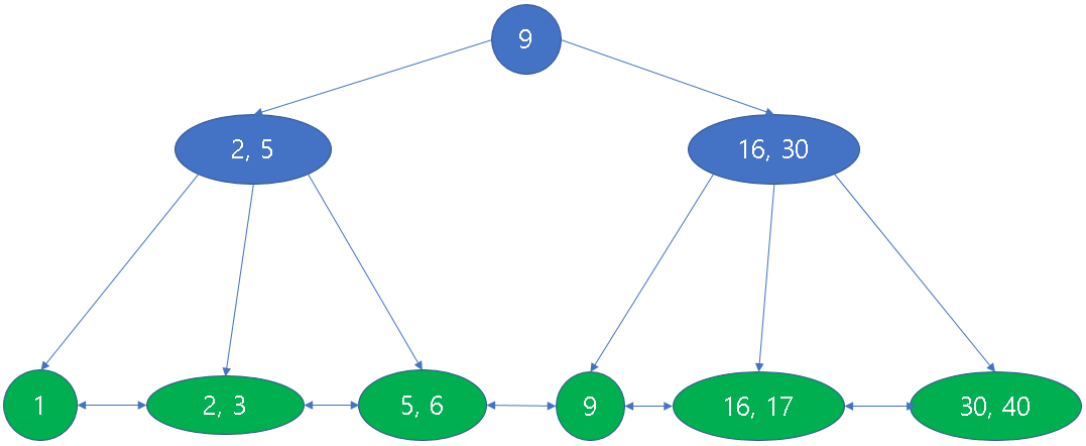


Insert 2



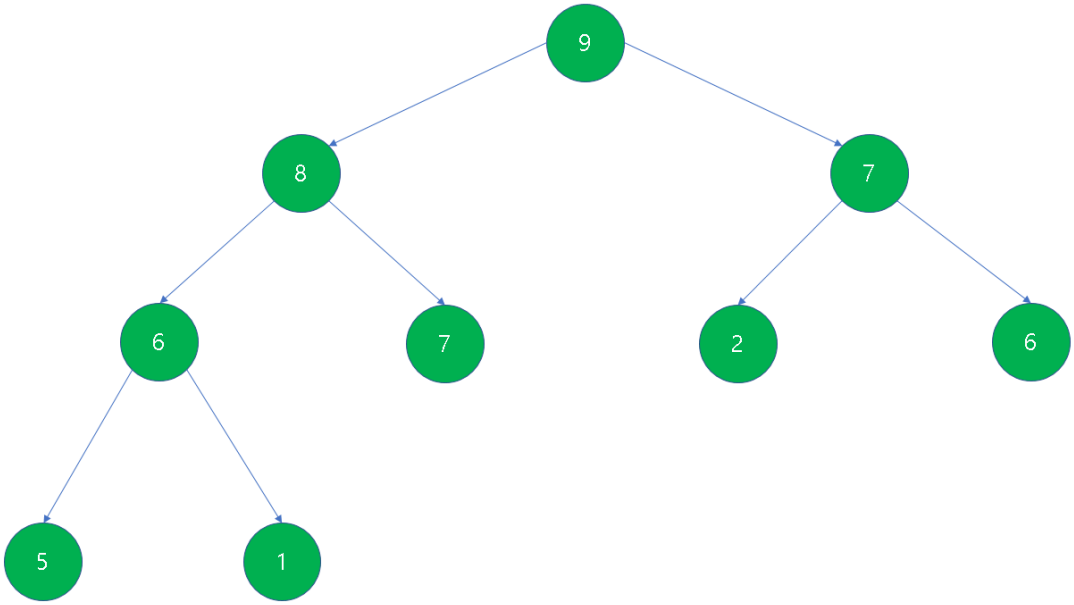


Result

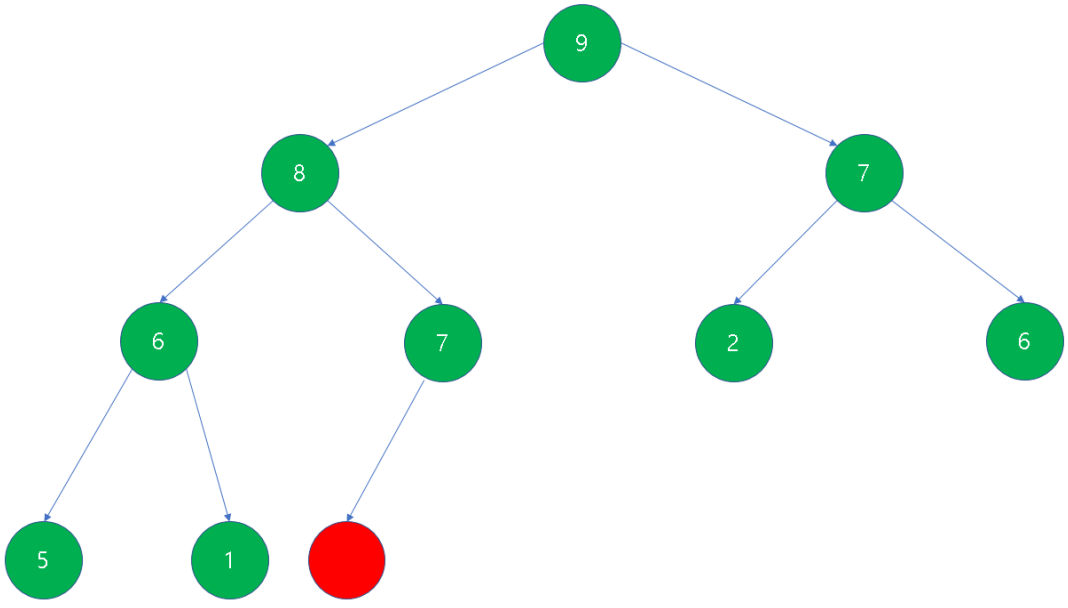


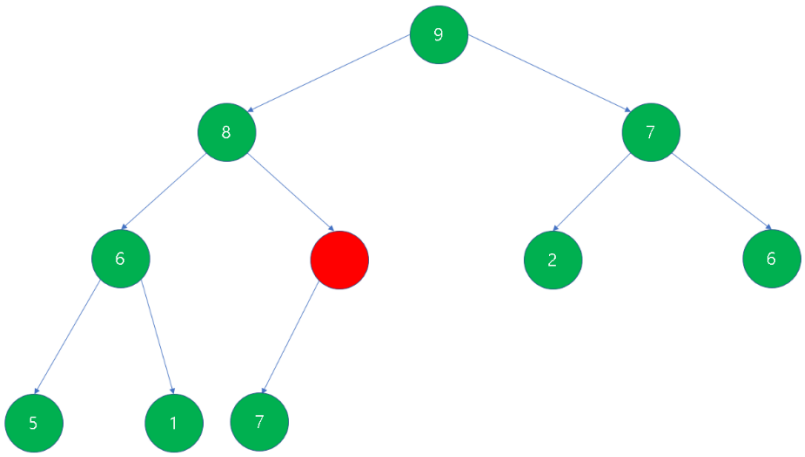
들어간 노드를 판단하여 개수가 3개면 계속해서 split이 일어나면 B+ tree를 만족할 수 있습니다. 최종적인 B+ tree의 모양은 위 이미지와 같습니다.

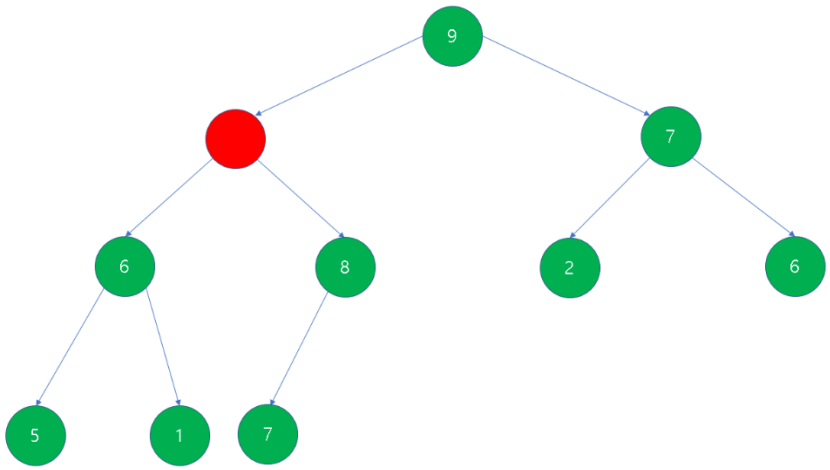
Insert Max-Heap

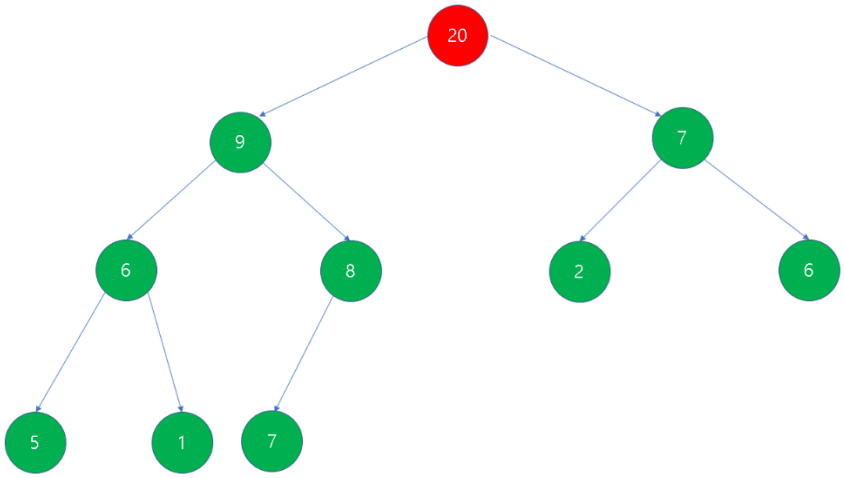


New element is 20



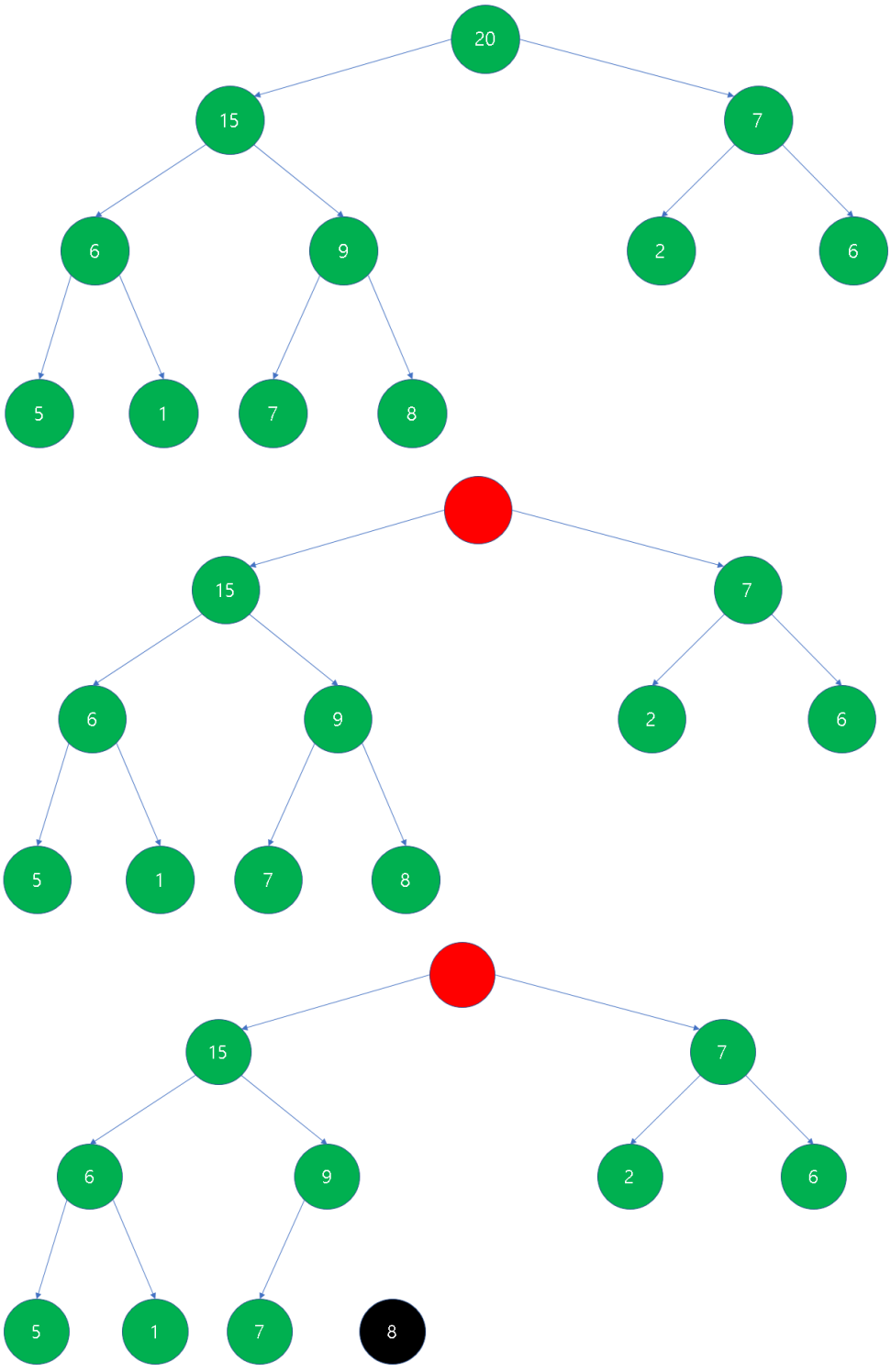




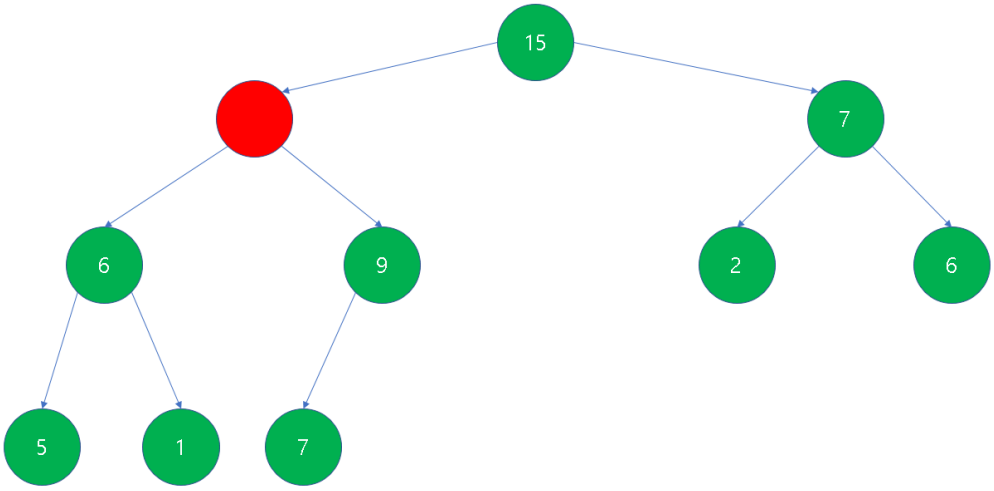


위 과정은 Max Heap의 insert과정으로 leaf로 data가 먼저 들어간 후 해당하는 위치하는 부분의 부모들과 데이터를 비교하여 데이터가 순차적으로 올라가는 것을 확인할 수 있습니다.

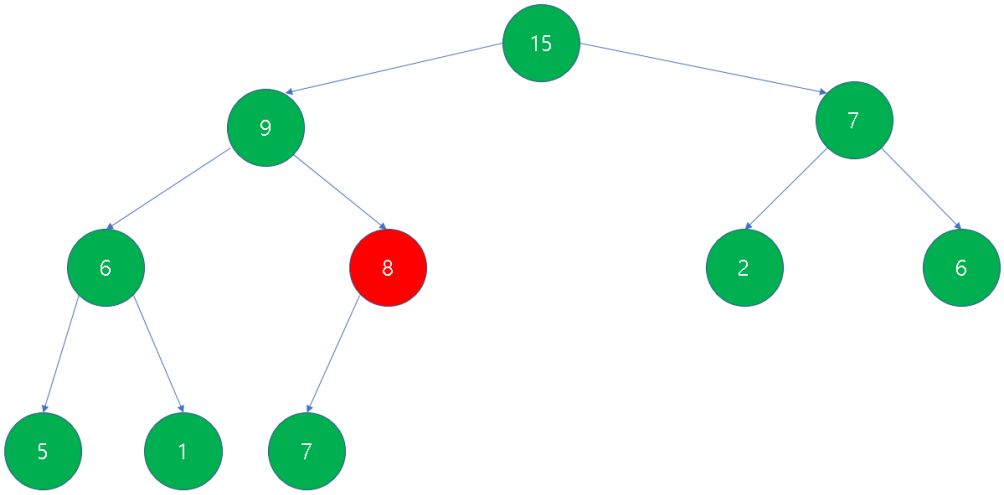
Pop Max-Heap



Reinsert 8 into the heap.



Reinsert 8 into the heap.



Reinsert 8 into the heap.

위 이미지는 Pop동작을 한번 완료한 tree의 형태입니다. Heap은 root를 먼저 제거한 후 root를가 될 data를 올려주어야 하기 때문에 leaf에서 하나를 없앤 후 데이터를 비교하고 위로 올려주며 결과적으로 data하나가 소멸되는 것을 확인할 수 있다.

4. Result Screen

프로그램 검증에 사용된 명령어

```
LOAD list
SEARCH_AVL 사망생명
SEARCH_AVL 풍선항공
SEARCH_BP -0.7 9
PRINT_BP
PRINT_AVL
RANK
LOAD
PRINT_AVL
PRINT_BP
SEARCH_AVL 유투온
SEARCH_AVL 단절통신
SEARCH_AVL 사망생명
SEARCH_AVL 없는정보
SEARCH_AVL 석기전자
SEARCH_AVL 지진건설
SEARCH_AVL 데이터
SEARCH_AVL 파산은행
SEARCH_AVL 풍선항공
SEARCH_BP -0.99 20
SEARCH_BP -0.11 -0.12
SEARCH_BP 50.00 100.00
SEARCH_BP 7.00 7.10
RANK hi
RANK
RANK
PRINT_AVL world
PRINT_BP HI
EXIT
```

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

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

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

======== ERROR ========
200
=======================

======== ERROR ========
500
=======================

======== ERROR ========
600
=======================

======== ERROR ========
400
=======================

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

해당 명령은 정상적으로 LOAD가 완료되기 전까지 결과를 나타낸 결과입니다. 우선 첫 load시 명령어는 쓸데없는 인자가 포함되므로 ERROR 100이 잘 출력이 되었으며 load가 되지 않았으므로 Search할 정보가 없어 AVL search에서 2개의 에러코드가 출력됩니다. 또한 BP search에서도 에러코드가 출력되며 Print AVL, BP명령에서도 에러가 뜨는 것을 확인할 수 있습니다. 마지막으로는 RANK또한 에러가 출력되는 것을 확인할 수 있습니다. 그리고 마지막 Success는 LOAD명령어가 정확히 인식하여 Tree구축 수행을 완료한 것입니다. 다음으로 확인할 명령어들은 Load가 성공적으로 완료된 후 수행되는 명령어와 출력화면입니다.

PRINT_AVL 해당 명령은 AVL Tree에 있는 데이터들을 종목 이름의 오름차순으로 정렬하여 출력하는 방식입니다.

```
==================== PRINT ====================
7485 겟마블 -0.18
시가: 66
종가: 54
거래량: 14
수익률: -0.18

7654 광운전자 3.53
시가: 17
종가: 77
거래량: 94
수익률: 3.53

1543 나비아 -0.33
시가: 54
종가: 36
거래량: 13
수익률: -0.33

1111 노원전자 -0.60
시가: 58
종가: 23
거래량: 50
수익률: -0.60

2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25

4873 리노스 -0.26
시가: 69
종가: 51
거래량: 22
수익률: -0.26

6542 모비스 -0.88
시가: 50
종가: 6
거래량: 31
수익률: -0.88

5984 미래콤 0.58
시가: 43
종가: 68
거래량: 65
수익률: 0.58

5419 사망생명 5.92
시가: 12
종가: 83
거래량: 56
수익률: 5.92

9821 석기전자 1.50
시가: 20
종가: 50
거래량: 90
수익률: 1.50
```

9922 성남화학 20.00
시가: 2
종가: 42
거래량: 64
수익률: 20.00

2132 안나와 -0.99
시가: 76
종가: 1
거래량: 3
수익률: -0.99

8416 엔마이너스 -0.74
시가: 78
종가: 20
거래량: 61
수익률: -0.74

1241 유투온 -0.18
시가: 57
종가: 47
거래량: 87
수익률: -0.18

1702 제로에너지 0.53
시가: 30
종가: 46
거래량: 91
수익률: 0.53

8590 지진건설 -0.46
시가: 98
종가: 53
거래량: 35
수익률: -0.46

4375 케이더블유 0.00
시가: 86
종가: 86
거래량: 3
수익률: 0.00

3287 파산은행 -0.53
시가: 51
종가: 24
거래량: 38
수익률: -0.53

9234 풍선항공 -0.53
시가: 80
종가: 38
거래량: 13
수익률: -0.53

2311 한국화학 0.65
시가: 52
종가: 86
거래량: 53
수익률: 0.65

PRINT_BP 다음은 PRINT_BP 명령어로 B+ tree에 구축된 정보들을 불러오는 방식입니다. 해당 데이터들은 수익률을 기반으로 내림차순으로 정렬되어 출력되며 수익률이 같은 경우 고유번호의 내림차순으로 정렬하게 됩니다.

```
                                                    2311 한국화학 0.65
                                                    시가: 52
                                                    종가: 86
                                                    거래량: 53
                                                    수익률: 0.65
========== PRINT ==========
9922 성남화학 20.00                                  5984 미래콤 0.58
시가: 2                                             시가: 43
종가: 42                                            종가: 68
거래량: 64                                          거래량: 65
수익률: 20.00                                       수익률: 0.58

5419 사망생명 5.92                                   1702 제로에너지 0.53
시가: 12                                            시가: 30
종가: 83                                            종가: 46
거래량: 56                                          거래량: 91
수익률: 5.92                                        수익률: 0.53

7654 광운전자 3.53                                   2354 단절통신 0.25
시가: 17                                            시가: 76
종가: 77                                            종가: 95
거래량: 94                                          거래량: 89
수익률: 3.53                                        수익률: 0.25

9821 석기전자 1.50                                   4375 케이더블유 0.00
시가: 20                                            시가: 86
종가: 50                                            종가: 86
거래량: 90                                          거래량: 3
수익률: 1.50                                        수익률: 0.00
```

7485 겟마블 -0.18
시가: 66
종가: 54
거래량: 14
수익률: -0.18

1241 유투온 -0.18
시가: 57
종가: 47
거래량: 87
수익률: -0.18

4873 리노스 -0.26
시가: 69
종가: 51
거래량: 22
수익률: -0.26

1543 나비아 -0.33
시가: 54
종가: 36
거래량: 13
수익률: -0.33

8590 지진건설 -0.46
시가: 98
종가: 53
거래량: 35
수익률: -0.46

9234 풍선항공 -0.53
시가: 80
종가: 38
거래량: 13
수익률: -0.53

3287 파산은행 -0.53
시가: 51
종가: 24
거래량: 38
수익률: -0.53

1111 노원전자 -0.60
시가: 58
종가: 23
거래량: 50
수익률: -0.60

8416 엔마이너스 -0.74
시가: 78
종가: 20
거래량: 61
수익률: -0.74

6542 모비스 -0.88
시가: 50
종가: 6
거래량: 31
수익률: -0.88

2132 안나와 -0.99
시가: 76
종가: 1
거래량: 3
수익률: -0.99

==========

```
SEARCH_AVL 유투온
SEARCH_AVL 단절통신
SEARCH_AVL 사망생명
SEARCH_AVL 없는정보
SEARCH_AVL 석기전자
SEARCH_AVL 지진건설
SEARCH_AVL 데이터
SEARCH_AVL 파산은행
SEARCH_AVL 풍선항공
```

다음은 SEARCH_AVL 명령어로 해당 데이터를 가지고 있는 노드를 찾게 됩니다. 노드가 없는 경우 ERROR code를 출력합니다.

```
======= SEARCH =======
1241 유투온 -0.18
시가: 57
종가: 47
거래량: 87
수익률: -0.18
======================
```

```
======= SEARCH =======
2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25
======================

======= SEARCH =======
5419 사망생명 5.92
시가: 12
종가: 83
거래량: 56
수익률: 5.92
======================

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

======= SEARCH =======
9821 석기전자 1.50
시가: 20
종가: 50
거래량: 90
수익률: 1.50
======================
```

```
======= SEARCH =======
8590 지진건설 -0.46
시가: 98
종가: 53
거래량: 35
수익률: -0.46
======================

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

======= SEARCH =======
3287 파산은행 -0.53
시가: 51
종가: 24
거래량: 38
수익률: -0.53
======================

======= SEARCH =======
9234 풍선항공 -0.53
시가: 80
종가: 38
거래량: 13
수익률: -0.53
======================
```

```
SEARCH_BP -0.99 20
SEARCH_BP -0.11 -0.12
SEARCH_BP 50.00 100.00
SEARCH_BP 7.00 7.10
```

다음 명령어는 SEARCH_BP입니다. B+ Search같은 경우 범위를 기반으로 데이터들을 찾습니다. 해당 범위안에 데이터가 존재하면 출력하고 없는 경우는 ERROR code를 출력합니다.

```
======== SEARCH ========
9922 성남화학 20.00
시가: 2
종가: 42
거래량: 64
수익률: 20.00

5419 사망생명 5.92
시가: 12
종가: 83
거래량: 56
수익률: 5.92

7654 광운전자 3.53
시가: 17
종가: 77
거래량: 94
수익률: 3.53

9821 석기전자 1.50
시가: 20
종가: 50
거래량: 90
수익률: 1.50

2311 한국화학 0.65
시가: 52
종가: 86
거래량: 53
수익률: 0.65

5984 미래콤 0.58
시가: 43
종가: 68
거래량: 65
수익률: 0.58

1702 제로에너지 0.53
시가: 30
종가: 46
거래량: 91
수익률: 0.53

2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25

4375 케이더블유 0.00
시가: 86
종가: 86
거래량: 3
수익률: 0.00

7485 겟마블 -0.18
시가: 66
종가: 54
거래량: 14
수익률: -0.18

1241 유투온 -0.18
시가: 57
종가: 47
거래량: 87
수익률: -0.18

4873 리노스 -0.26
시가: 69
종가: 51
거래량: 22
수익률: -0.26

1543 나비아 -0.33
시가: 54
종가: 36
거래량: 13
수익률: -0.33

8590 지진건설 -0.46
시가: 98
종가: 53
거래량: 35
수익률: -0.46

9234 풍선항공 -0.53
시가: 80
종가: 38
거래량: 13
수익률: -0.53

3287 파산은행 -0.53
시가: 51
종가: 24
거래량: 38
수익률: -0.53

1111 노원전자 -0.60
시가: 58
종가: 23
거래량: 50
수익률: -0.60

8416 엔마이너스 -0.74
시가: 78
종가: 20
거래량: 61
수익률: -0.74

6542 모비스 -0.88
시가: 50
종가: 6
거래량: 31
수익률: -0.88

2132 안나와 -0.99
시가: 76
종가: 1
거래량: 3
수익률: -0.99

------------------------

======== ERROR ========
200
========================

======== ERROR ========
200
========================

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

```
RANK hi
RANK
RANK
```

다음 명령어는 RANK와 관련된 명령어로 처음은 인자가 들어갔으므로 ERROR code가 나오며 다음 RANK는 정상출력 그 다음 RANK는 삭제되며 출력되기 때문에 정보가 없으므로 ERROR가 나와야 합니다.

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

=================== RANK ====================
5419 사망생명 5.92
시가: 12
종가: 83
거래량: 56
수익률: 5.92

9821 석기전자 1.50
시가: 20
종가: 50
거래량: 90
수익률: 1.50

2354 단절통신 0.25
시가: 76
종가: 95
거래량: 89
수익률: 0.25

1241 유투온 -0.18
시가: 57
종가: 47
거래량: 87
수익률: -0.18

8590 지진건설 -0.46
시가: 98
종가: 53
거래량: 35
수익률: -0.46

3287 파산은행 -0.53
시가: 51
종가: 24
거래량: 38
수익률: -0.53

9234 풍선항공 -0.53
시가: 80
종가: 38
거래량: 13
수익률: -0.53

=============================================

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

```
PRINT_AVL world
PRINT_BP HI
EXIT
```

다음은 필요 없는 인자를 받았을 경우와 EXIT명령어입니다. EXIT명령어 같은 경우 ERROR code출력이 PDF에 명시되지 않았기 때문에 해당 예외처리는 하지 않았습니다.

```
======== ERROR ========
600
========================

======== ERROR ========
500
========================

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

5. Consideration

해당 과제의 목적은 전 프로젝트와 마찬가지로 설계시간에 배운 tree의 구조 3가지를 구현하는데 목적을 둡니다. 첫번째 프로젝트를 완료하고 배운 새로운 tree의 구조로는 AVL tree, B+ tree, Heap이 있었습니다. B+ tree를 제외한 나머지 구조는 관련 시험문제와 PDF에 제공된 코드가 있었기에 이해하기도 쉬우며 설계 전 직접 실행해보며 자료구조에 대하여 충분히 학습하고 익숙해진 상황이었습니다. 가장 큰 문제는 B+의 Insert구현을 포함한 Split함수에 대한 구현이었습니다. 특히 처음 구현당시 Data Node와 Index Node의 역할을 헷갈려 구현하는데 혼동이 많았습니다. 제공된 PDF의 Split과정을 보고 개념을 다시 잡았으며 알고리즘의 구현도 PDF와 동일하게 구현하게 되었습니다. B+ tree같은경우 PDF를 참고한 것뿐만 아니라 Google를 통해 정보를 상당히 많이 찾게 되었고 많은 경우의 코드해석과 order가 다른 경우 동작하는 B+ tree도 공부하는 계기가 되었습니다. 하지만 이번 프로젝트에서는 order3을 이용하여 각 노드가 데이터를 최대 두개 가질 수 있도록 설계하였으며 size를 체크하여 3이되면 split함수를 통하여 Node를 분리해주는 작업을 하게 되었습니다. 다음으로 생소했던 함수구현은 Print와 Search입니다. 특히 B+에서의 Search는 범위를 정하여 그 범위에 해당하는 KEY값을 비교하여 해당하는 데이터들을 모두 출력할 수 있는데 이는 leaf에 모든 데이터들이 linked list로 연결되어 있기 때문에 검색속도가 효율적일 수 있습니다. 하지만 문제는 구현하는 방식이 지금까지 해왔던 방식과는 많이 달랐습니다. 기존 해왔던 Node의 Print는 해당 노드의 포인터가 다음 노드를 가리키는 현태로 되어있지만 B+ tree에서의 노드는 왼쪽 노드를 가리키며 가운데 범위 노드와 오른쪽 노드는 노드의 구성이 되는 Map에서 가리키게 되도록 하였으며 이 역할을 하는 Node가 Index Node의 역할입니다. 알맞은 출력을 위해서는 Map의 사용법과 출력을 Data Node를 기반으로 하기 때문에 vector를 알아야 하며 각각의 iterator또한 알고 있어야 구현할 수 있는 함수입니다. 특히 vector는 array를 대신하여 사용할 수 있는 STL로써 지금까지는 array에 대한 사용 빈도가 높았지만 좀더 편리하게 사용할 수 있는 STL이란 것을 배우는 계기가 되었습니다. 이번 프로젝트를 진행하면서 새로운 데이터구조를 구현하는 것이 가장 큰 목적이었지만 다양한 STL을 쓰면서 개발자가 보다 편하고 효율적으로 프로그램을 작성할 수 있다는 것도 배우게 되었습니다. 마지막으로 리눅스에 관련된 고찰입니다. 이번 프로그램을 작성하는데 첫번째 프로젝트와는 달리 한글이 출력format에 포함이 되는 경우가 생기게 되었습니다. 이러한 이유로 cpp파일이 Linux로 넘어오면서 글이 깨지는 경우가 생겼는데 이로 인하여 Linux내에서 출력이 원활하지 않은 것이 확인되었습니다. 주석에도 한글이 포함되어 출력이 깨지게 되면 이 또한 출력이 원활 하지 않은 것이 확인 되었습니다. 이에 대한 해결방안으로는 윈도우 환경에서만 파일을 수정하는 것이 아니라 Linux내에서 파일을 직접 수정하는 방법으로 바로 한글과 호환될 수 있도록 수정하여 출력 format에 이상이 없도록 수정하게 되었습니다.