알 고 리 즘 설 계 과 제

설계과제명	BST와 AVL의 분석
제출날짜	2019년 12월 9일
학 과	컴퓨터공학과
학 번	2017152049
이 름	정하림

목차

1장. 개요

1.1 수행중인 시뮬레이션의 소개

1) 시뮬레이션 개요

- 이 시뮬레이션은 BST와 AVL 트리의 노드 수 증가에 따른 평균적인 높이와 검색시의 비교 횟수를 측정하기 위해 설계되었다.

- 중복 없는 난수를 활용하여 노드 개수를 100 – 1000 – 2000 – 3000 – 5000 – 7500 – 10000 개로 점차 늘려가며 BST와 AVL트리에 삽입하고 연산 결과로 도출되는 트리의 높이와 랜덤한 특정수를 검색하는것에 수행되는 비교횟수를 구하여 BST와 AVL트리를 비교해 본다.

- 트리에서 높이와 비교횟수를 구하는 것은 해당 트리의 주목적인 탐색영역에서의 효율을 확인하기 위함이다. 높이와 비교횟수를 통해 실제 높이와 비교횟수의 상관관계를 확인해보고 평균 검색시간을 도출해내어 BST와 그를 보완한 AVL트리의 검색시간을 비교해보고 실제로 확인해볼 예정이다.

2) 입력할 값

- 이진검색트리는 삽입의 순서에 따라 모양이 결정된다. 키가 총 n개라면 키의 절대값은 중요하지 않고 키의 상대적인 크기에 따라 트리의 모양이 결정된다. n개의 키가 만들 수 있는 상대적 순서는 모두 n!가지이다. 때문에 rand()%최대노드갯수 +1을 통해 생성한 난수값 즉 실제 key값은 1~노드최대 노드 개수 만큼 범위를 지니게 했다. 이 값들을 중복되지 않게 생성하여 배열에 넣어 순차적으로 노드를 생성하도록 하였다.

3) 예상되는 결과

- 이진 탐색트리에서 최악의 탐색시간은 리프노드까지 탐색하는 것으로 O(h)이다
- 이진탐색트리의 이상적인 높이는 lg(n)이고 가장 나쁜 결과는 O(n)이된다. 때문에 이진탐색트리의 성공적인 탐색결과는 O(lg n)에 근접할 것이다.
  퀵정렬과 마찬가지로 평균은 최악의 경우보다 최상의 경우에 가까울 것이다.
- AVL트리는 이진탐색트리에서 균형을 보완하여 lg n에 해당하는 높이를 지니고 평균적으로 O(lg n)에 해당하는 값이 나올 것이다.

2장. 알고리즘 개념 및 차이점

2.1 BST 알고리즘의 개념

1) 트리(Tree)의 개념

1-1) 트리란?

- 트리는 나무를 거꾸로 세운 모습을 모형화한 자료구조로 계층적 표현이나 검색을 쉽게 만들어주는 도구이다.

1-2) 트리의 정의

- 1개 이상의 노드를 갖는 자료의 집합이다.
- 루트라고 불리는 특별한 노드가 있다.
- 루트 이외의 나머지 노드들은 원소가 중복되지않는 n개의 subtree를 가진다.
- 사이클이 없는 그래프이며 계층구조를 이룬다.

2) 이진 트리(Binary Tree)의 개념

2-1) 이진트리란?

- 일반 트리 중에서도 자식 노드의 수가 2개 이하인 트리는 컴퓨터에서 다루기 쉽고 많이 활용된다.

2-2) 이진트리의 정의

- 이진트리는 유한개의 노드로 구성된 트리이다.
- **비어있거나** 혹은 **루트노드와 2개의 부속트리로 구성**된다.

2-3) 이진트리의 성질

- 이진트리는 일반트리와 달리 노드가 하나도 없는경우에도 성립한다.
- 각 노드는 0,1,2개의 자식노드를 가지며 왼쪽 자식과 오른쪽 자식은 서로 다른 자식으로 구분된다
- 이진트리의 최대 레벨이 i인 경우 트리의 최대 노드갯수는 2의 (i-1)승 개다.
- 차수가 0인 노드(리프노드)는 차수가 2인 노드의 수보다 하나 더 많다. n0 = n2 + 1

2-4) 이진트리의 종류

- 편향 이진트리 : 모든 노드가 부모노드의 왼쪽 자식이거나 오른쪽 자식인 이진 트리
- 포화 이진트리 : 이진트리가 보유할 수 있는 최대노드를 보유한 이진트리 n = 2^(h+1)-1
- 완전 이진트리 : n<=2^(h+1)-1 인 이진트리 중 마지막 레벨에서 왼쪽부터 채워진 트리

3) 이진 탐색트리(BST)의 개념

3-1) 이진 탐색트리란?

- 이진 탐색트리란 이진트리 구조에 검색 대상 데이터를 저장하는 방법이다. 비어있는 트리에 key값들을 저장하되, key값이 작은 것을 왼쪽 트리로, key값이 크면 오른쪽 트리로 저장한다. 모든 노드는 key값을 지니며 같은 key를 가지는 경우는 없다. 왼쪽 subtree의 모든 노드의 key값은 루트의 key값보다 작고 오른쪽 모든 노드의 key값은 루트의 key값보다 크다. 즉 왼쪽 자식은 부모보다 작은 값 오른쪽 자식은 부모보다 큰값으로 정렬된 트리이다.

3-2) 이진탐색트리의 문제점

- 이진탐색트리는 트리의 높이를 하나씩 더해갈수록 추가할 수 있는 노드의 수가 두배씩 증가하게 되므로 이진탐색트리의 높이는 평균적으로 O(lg n)의 시간복잡도를 가진다. 그런데 이러한 이진탐색트리는 균형이 맞지 않을수록 O(n)에 가까운 시간복잡도를 가지게된다.
  만약 이진탐색트리가 한쪽으로만 진행되는 편항이진트리의 형태를 하게될 경우 높이가 O(n)일것이며 비교횟수가 높이의 영향을 받으므로 최악의 경우 탐색에 O(n)의 시간이 소요된다. 이것을 극복하기위해 균형이진트리가 등장하였고 그중 AVL을 설명하려한다.

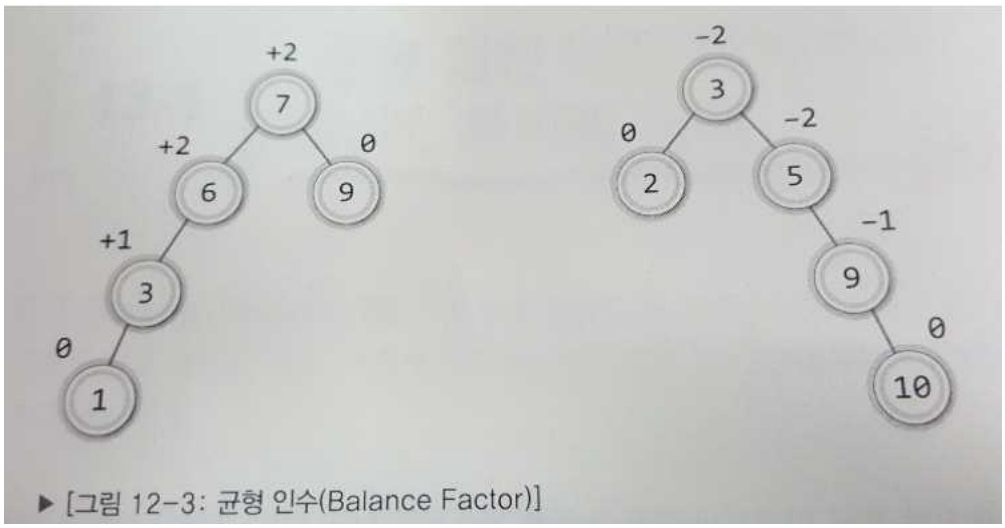
2.2 AVL 알고리즘의 개념

1) AVL트리의 개념

1-1) AVL 트리란?

- ABL트리는 G.M. Adelson-Velskii와 E. M Landis에 의해 1960년대에 고안되었다.
- 트리의 이름은 이들의 이름에서 유래되었다. AVL트리는 노드가 추가되거나 삭제될 때 트리의 균형상태를 파악하여 구조를 변경하며 균형을 잡는 균형 이진 탐색트리이다.

1-2) 균형을 잡는 방법



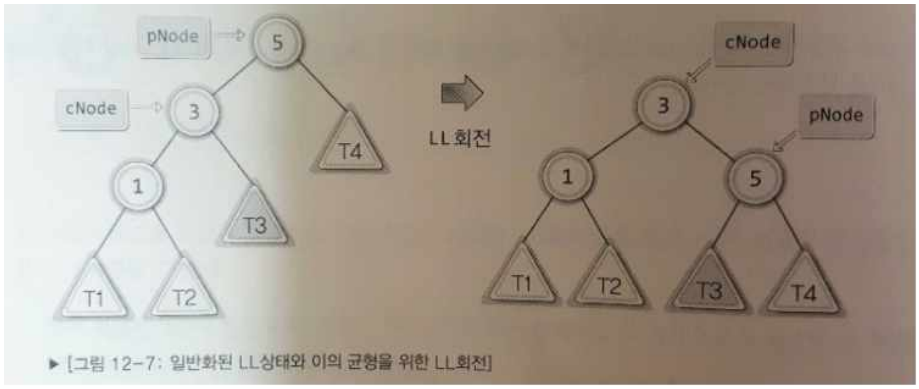
▶ [그림 12-3: 균형 인수(Balance Factor)]

- 균형의 정도를 표현하기위해 균형인수(Balance Factor)라는 것을 사용하는데 이 균형인수는 다음과같이 계산된다.
- 균형인수 = 왼쪽 서브트리 높이 – 오른쪽 서브트리 높이
- 도출된 균형인수의 값에 따라 균형인수의 절대값이 2 이상인 경우 트리의 균형을 바로잡기위한 리밸런싱이 일어난다.

2) AVL 로테이션

- AVL 트리의 균형이 무너진 상태는 총 4가지로 정리된다. 그리고 각 리벨런싱 방법에도 차이가 있다.

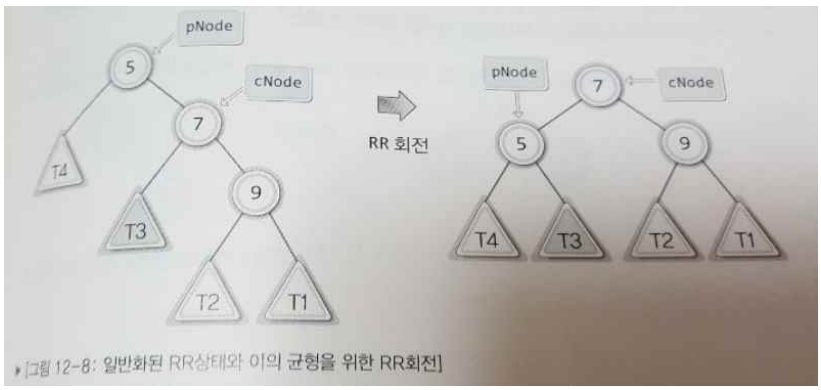
2-1) LL회전



▸ [그림 12-7: 일반화된 LL상태와 이의 균형을 위한 LL회전]

- 자식노드 두 개가 왼쪽으로 연이어 연결되어 균형이 무너져 균형인수가 2이상이 될 경우를 Left Left 상태 즉 LL상태라고 칭하며 이러한 불균형 해소를 위해 등장한 리벨런싱 방법이 LL회전이다.
- LL회전의 핵심은 균형인수가 2인 노드를 균형인수가 1인 노드의 오른쪽 자식이 되게하는 것에 있다.

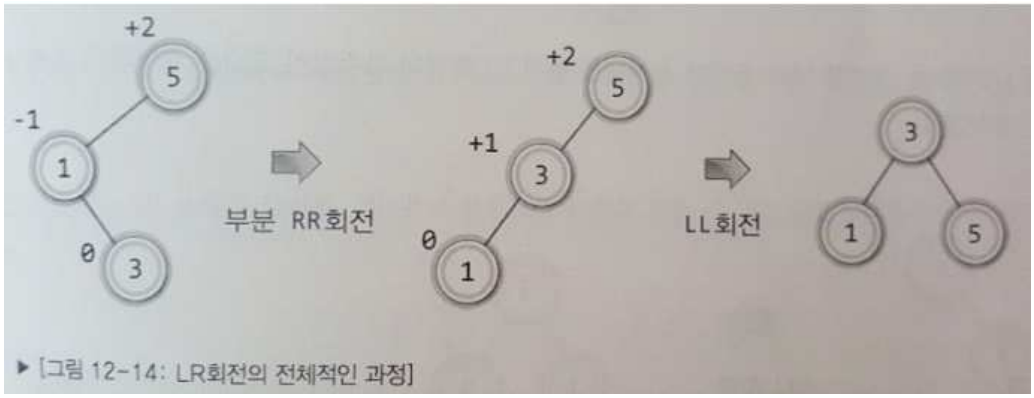
2-2) RR회전



▸ [그림 12-8: 일반화된 RR상태와 이의 균형을 위한 RR회전]

- 앞서 설명한 LL상태에서 반대방향으로 회전을 요구하는 상태를 RR상태라고 한다.
- 즉 자식노드 두 개가 오른쪽으로 연이어 연결되어 균형이 무너진 상태로 이를 RR회전을 통해 해결할 수 있다.

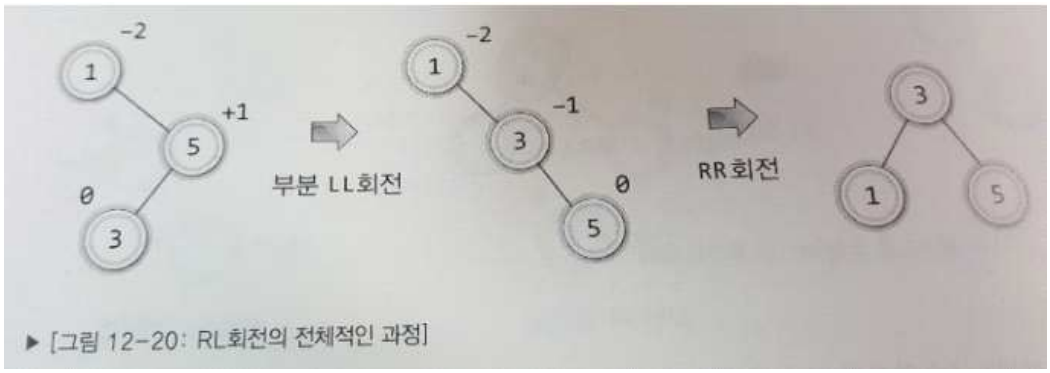
2-3) LR회전



▶ [그림 12-14: LR회전의 전체적인 과정]

- LR회전은 LR상태에서의 리벨런싱을 위한 회전방법으로 LR상태란 자식노드가 왼쪽으로 하나 그리고 오른쪽으로 하나 연결되어 균형인수를 2 이상으로 무너트린 상태를 뜻한다.
- 이러한 상태는 한번의 회전으로 균형을 잡아줄 수 없어 두단계를 거쳐서 리밸런싱하게된다.
- 먼저 LR상태를 부분적 RR회전을 통해 부모와 자식노드의 관계를 바꿔주어 LL상태로 만들어 준 후 LL회전을 통해 균형을 잡아주게 된다.
- 정리하자면 LR회전은 부분적 RR회전과 LL회전의 조합으로 LR상태를 리벨런싱 한다.

2-4) RL회전



▶ [그림 12-20: RL회전의 전체적인 과정]

- RL회전은 역시 LR회전과 반대로 부분적인 LL회전을 통해 트리를 RR상태로 만들어주고 RR회전을 통해 리밸런싱을 완성한다.

3장. 분석

3.1 차이점 도출을 위한 시뮬레이션 과정

1) 시뮬레이션 설명

1-1) 시뮬레이션 상세 정보

- 노드의 값은 전부 랜덤으로 1 ~ 노드 개수-1 개만큼 구성했다.
- 랜덤값 구성부터 insert작업까지를 전부 아래 함수에 넣어 작업을 100번 반복했다.

```
struct result_t* play_hundred_times(struct bst_node* bst_root, struct avl_node* avl_root);
```

- 또한 작업을 특정한 값을 랜덤으로 지정하여 해당 값에 대한 탐색을 진행하였다.
- 생성된 트리의 높이와 탐색의 과정에서 발생한 비교횟수는 아래 구조체형식에 넣었다.

```
struct result_t
{
        double bst_sum_height;
        double bst_sum_count;
        double avl_sum_height;
        double avl_sum_count;
};
```

- 100번 반복한 후 해당 결과값을 100으로 나누어 평균 높이와 평균 비교횟수를 출력하였다

2) 시뮬레이션 결과

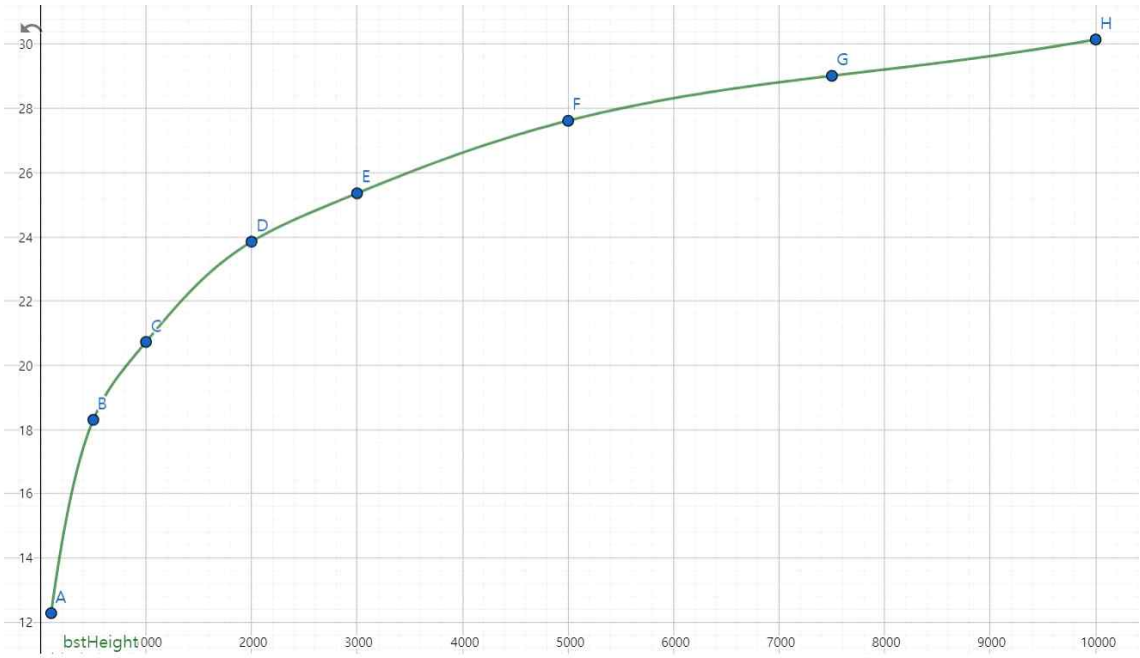
노드 개수(개)	BST 높이	BST 비교횟수	AVL 높이	AVL 비교횟수
100	12.28	7.45	7.0	5.97
500	18.3	10.84	9.92	8.24
1000	20.73	12.04	10.99	9.33
2000	23.85	12.8	12.01	9.85
3000	25.36	14.37	12.97	10.44
5000	27.62	15.14	13.95	11.79
7500	29.02	16.06	14.37	11.94
10000	30.15	16.92	15.01	12.56

4장. 분석 결과

4.1 결과값 분석

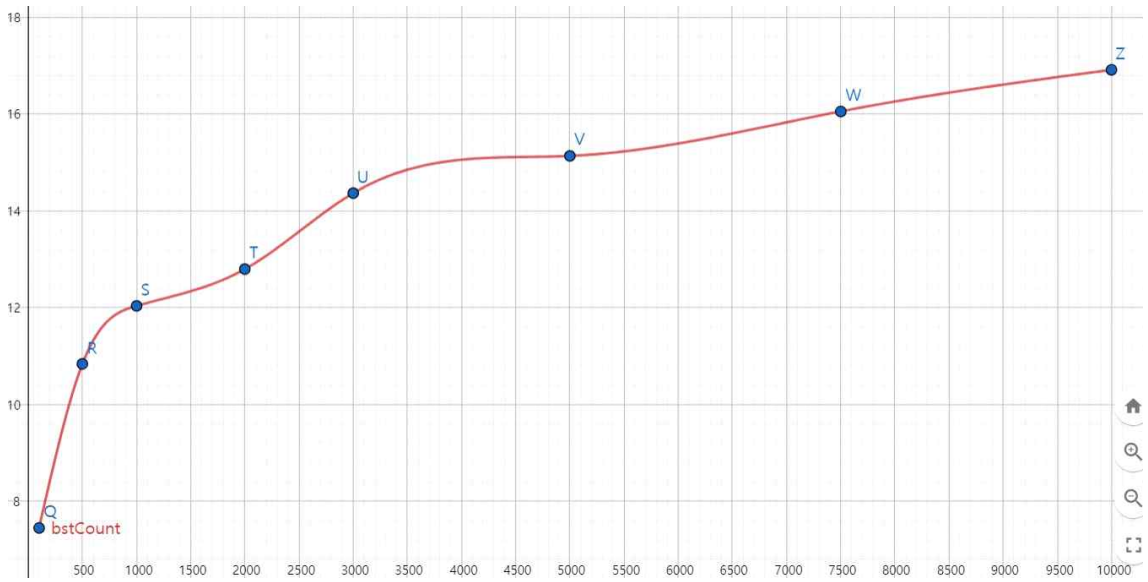
1) BST 그래프

1-1) BST 높이 그래프



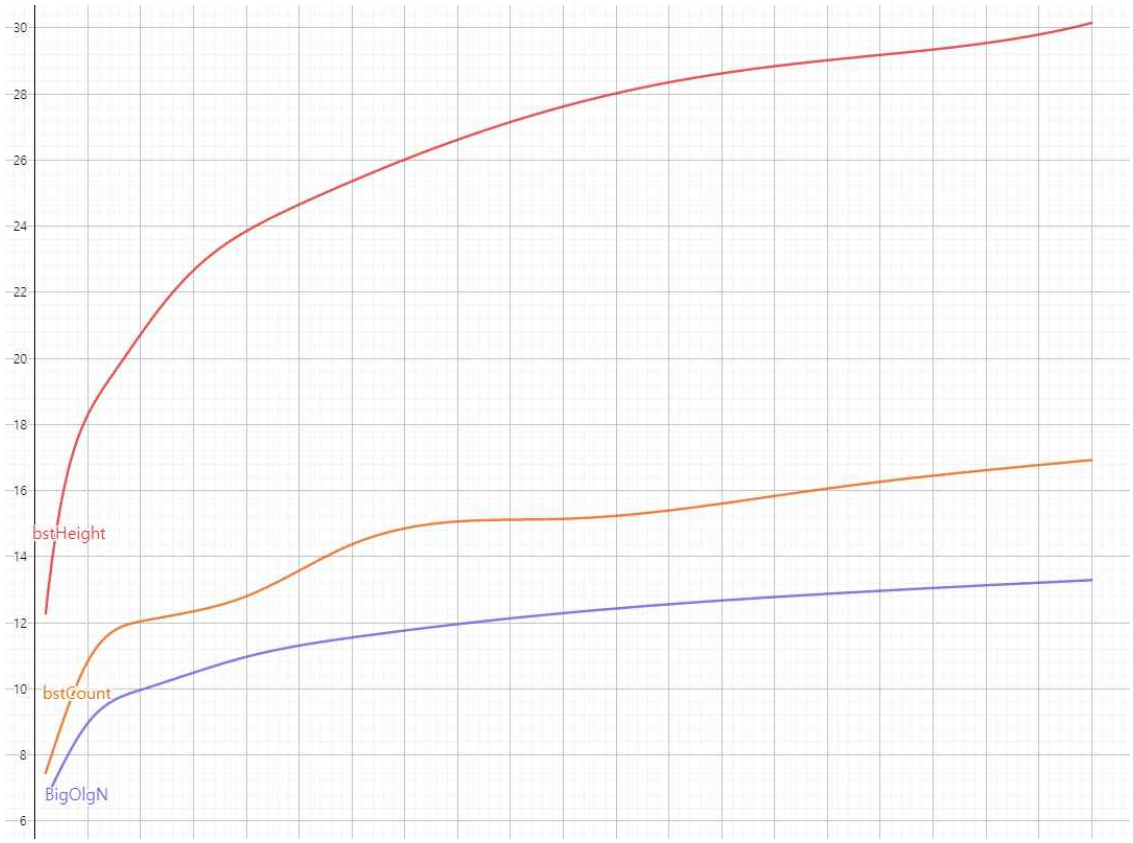
- BST에서 노드의 개수와 높이 평균값 간의 상관관계를 표현한 그래프이다.
- BST의 높이는 (lg n < h < n) 의 범위값을 지닐 수 있다.
- 높이값은 평균적으로 lg n보다 크지만 최악의 경우 O(n)에 도달하게 되는데 평균적으로 log함수의 형태를 띄는 것을 볼 수 있다. 이 그래프는 임의의 수로 구성한 BST의 평균높이가 퀵정렬처럼 최악의 경우가 드물다는 것을 의미하는 것 같다.

1-2) BST 비교 횟수 그래프



- BST에서 노드의 개수와 탐색 비교횟수 평균값 간의 상관관계를 표현한 그래프이다.
- 비교횟수의 평균이 고르지않아 그래프가 다소 울퉁불퉁한 같다. 그러나 역시 log함수의 형태를 띄어 BST의 효율성을 증명하였다.

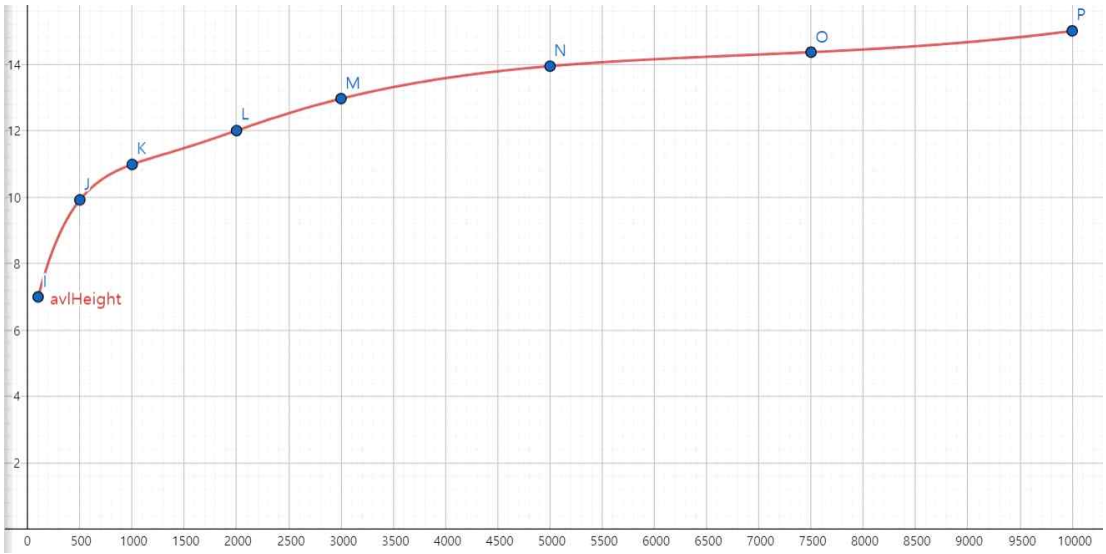
1-3) BST 분석 그래프



- 세 그래프가 모두 동일한 추세를 띄고 있다.
- bst의 비교횟수가 편향그래프가 되었을 때 n값에 가까워질수 있는데도 불구하고 평균 비교횟수 값은 lg n의 그래프와 굉장히 유사하다.

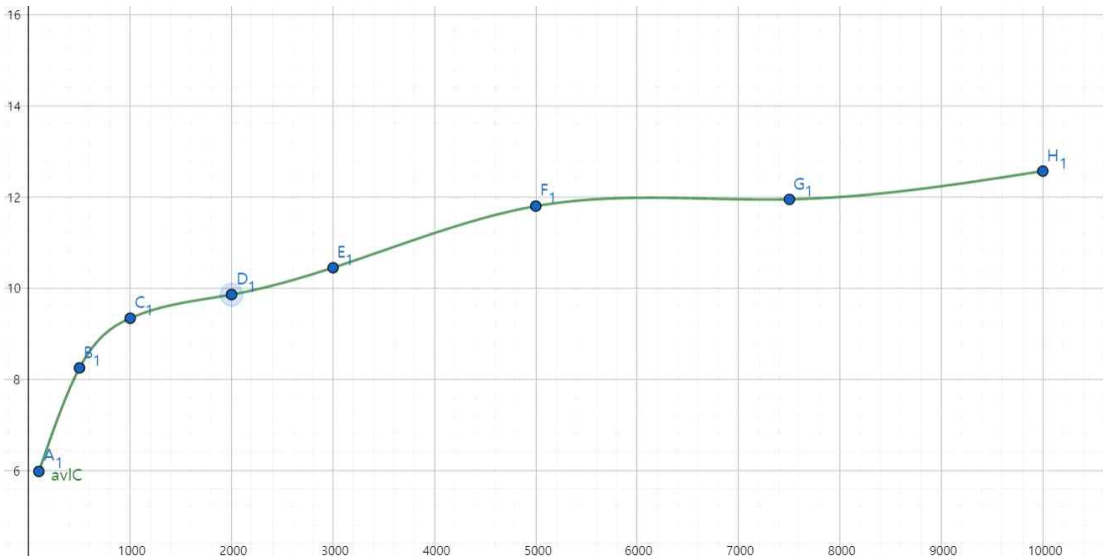
2) AVL 트리 그래프

2-1) AVL트리 높이 그래프



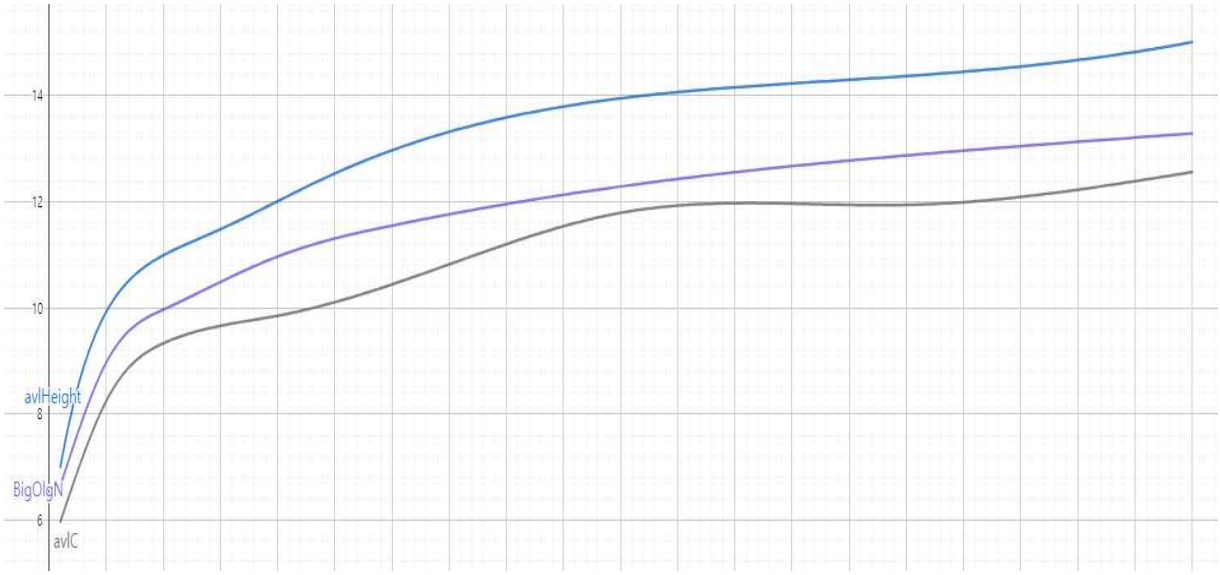
- AVL 트리의 평균 높이와 노드수의 상관관계를 표현한 그래프이다.
- 노드 수가 많아지더라도 확연하게 증가폭이 감소하여 효율적이다.

2-2) AVL 트리 비교횟수 그래프



- AVL 트리의 평균 비교 횟수와 노드 개수의 상관관계를 표현한 그래프이다.

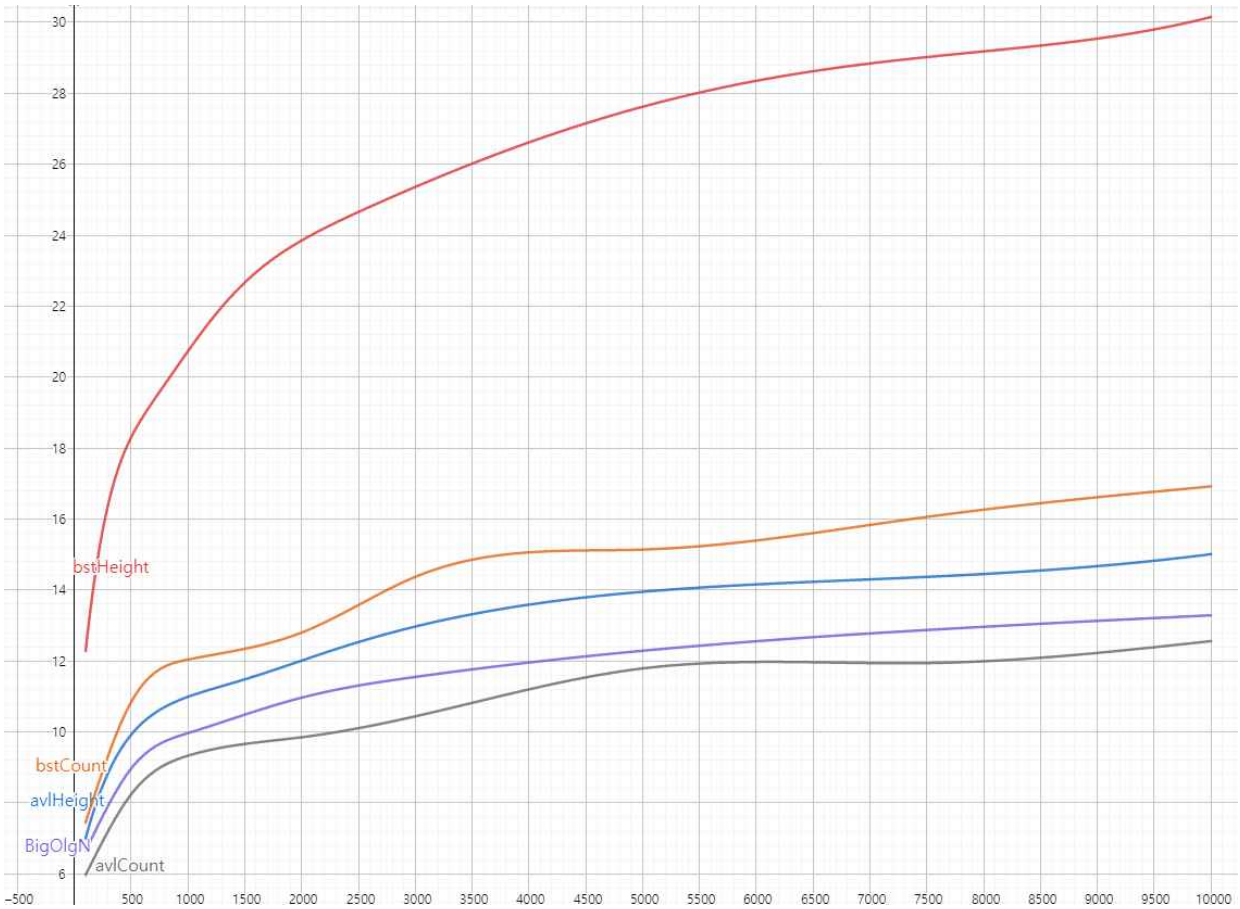
2-3) AVL 분석 그래프



- avl의 평균 높이, 비교횟수, O(lg n)의 그래프를 한꺼번에 비교하였다
- 높이와 비교횟수 모두 lgN의 값과 크게 차이나지 않았다.
- 그래프가 훨씬 안정적임을 볼 수 있다.

4.2 분석을 통해 도출한 결론

1) 최종 비교 그래프



- 평균 높이 측면에서는 bst보다 균형인수를 통해 높이의 균형을 맞춰주는 avl 트리가 2배가량 효율적이였다.
- 그러나 bst와 avl의 평균 비교횟수에서는 높이만큼의 극명한 차이가 발생하지 않았다.
- 그래프 분석 결과 avl의 평균 비교횟수는 O(lgN)의 원소가 될수 있었다.

4.3 총평

1) 표본을 100번 밖에 반복하지 않아서 오차가 발생할 수 있으나 그래프들이 전부 로그함수 그래프의 형태를 띄게 되었고 높이의 균형을 맞춘 avl트리의 경우에는 반복횟수와 노드갯수가 적음에도 불구하고 탐색 비교횟수가 O(lg n)에 성립하는 그래프가 도출되었다.

2) 그래프의 높이 측면에서 BST를 보완한 AVL트리가 훨씬 효율적이었음을 직접 확인해 보았다. 비교횟수 측면에서는 높이만큼의 차이가 나지는 않았지만 avl의 비교횟수가 O(lg n)의 원소가 되는 것을 확인할 수 있었다.

3) 간단한 수이지만 5000개이상의 데이터를 다뤄본 것이 처음이였는데 노트북 환경에서는 연산속도가 너무 오래걸려서 학교 리눅스 서버에서 코드를 실행하게되었고 학교서버의 고성능을 체감할 수 있었다.

참고자료

Introduction to Algorithms Thomas H Cormen
알기쉬운 자료구조-c언어 박우창
C C++로 배우는 자료구조론 주우석
윤성우의 열혈 자료구조 윤성우
자료구조와 알고리즘 김유성
그래프 작성 https://www.geogebra.org/