Binary Search Tree 와 AVL트리 비교

2014152019 컴퓨터공학과 심지섭

1. introduction

알고리즘 수업에서 배운 Binary Search Tree와 AVL트리. 이 두가지 data structure의 성능을 비교해보고자 이 과제를 수행하게 되었다. Binary Search Tree와 AVL트리는 어떤 연관관계가 있기 때문에 둘을 비교해 보는 것일까? 우선 둘은 이진트리라는 공통점이 있다. 현재 노드보다 작으면 왼쪽, 크면 오른쪽 자식으로 들어가는 특성 때문에 탐색을 할 때 노드의 root부터 leaf까지 한번의 흐름으로 진행이 된다 그래서 이진트리의 big-O는 이진트리의 높이이다. 이 높이를 줄이는 것이 탐색시간을 줄이는데 도움이 되는데 AVL트리가 높이를 줄이는 기능을 가지고 있는 data structure 이다. 그래서 Binary Search Tree와 AVL트리의 높이와 탐색시간을 조사하여 이 둘의 성능차이를 증명해보고자 한다.

Geeksforgeeks라는 홈페이지에서 얻을 수 있는 Binary Search Tree와 AVL트리구현 코드를 사용하여 Binary Search Tree와 AVL트리의 높이와 탐색시간을 측정해보고자 한다. 측정내용은 다음과 같다.

**-높이에 관하여**

|  |  |
| --- | --- |
| 100개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 100개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 200개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 200개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 500개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 500개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 1000개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 1000개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 1500개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 1500개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 2000개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 2000개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 2500개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 2500개의 노드를 가진 AVL트리  100개의 높이의 평균 |
| 3000개의 노드를 가진 Binary Search Tree  100개의 높이의 평균 | 3000개의 노드를 가진 AVL트리  100개의 높이의 평균 |

**-탐색횟수에 관하여**

|  |  |
| --- | --- |
| 100개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 100개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 200개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 200개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 500개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 500개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 1000개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 1000개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 1500개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 1500개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 2000개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 2000개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 2500개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 2500개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |
| 3000개의 노드를 가진 Binary Search Tree  100개의 특정 노드 탐색횟수의 평균 | 3000개의 노드를 가진 AVL트리  100개의 특정 노드 탐색횟수의 평균 |

측정한 결과 Binary Search Tree의 높이의 평균과 탐색횟수의 평균보다 AVL트리의 높이의 평균과 탐색횟수의 평균이 모두 적을 것으로 예상한다.

2. BST와 AVL트리에 대한 설명

**1) Tree?**

트리란 노드로 이루어진 자료구조로, 그래프의 일종이다. 그래프에 트리가 속한다는 것은 공집합도 트리라고 할 수 있다는 것을 말한다. 트리는 사이클이 없고 엣지의 수가 노드의 수 -1 이며, 하나의 부모에 여러 개의 자식이 있을 수 있다. 트리의 높이는 루트노드부터 가장 말단의 leaf노드까지의 높이다.

**2) Binary Tree?**

트리중 부모노드가 0-2개의 자식노드만 가질 수 있는 트리를 이진트리라 한다. 자식이 최대 두개밖에 있을 수 없기 때문에 왼쪽자식, 오른쪽자식으로 구분할 수 있는 것이 탐색에서 유용하게 작용한다. Binary Tree를 순회하는 방법에는 3가지가 있는데, 전위순회(Preorder), 중위순회(Inorder), 후위순회(Postorder) 세가지가 있다.

순회방식은 다음과 같다.

|  |  |
| --- | --- |
| Preorder | 현재 노드 -> 왼쪽 자식 노드 -> 오른쪽 자식 노드 |
| Inorder | 왼쪽 자식 노드 -> 현재 노드 -> 오른쪽 자식 노드 |
| Postorder | 왼쪽 자식 노드 -> 오른쪽 자식 노드 -> 현재 노드 |

**3) Binary Search Tree**

Binary Tree의 부모 노드에 자식 노드를 할당함에 있어 특정한 조건을 추가한 트리를 말한다.

왼쪽자식은 부모 노드보다 작다, 오른쪽자식은 부모 노드보다 크다는 조건을 만족하는 트리를 Binary Search Tree라고 한다.

**4) AVL트리**

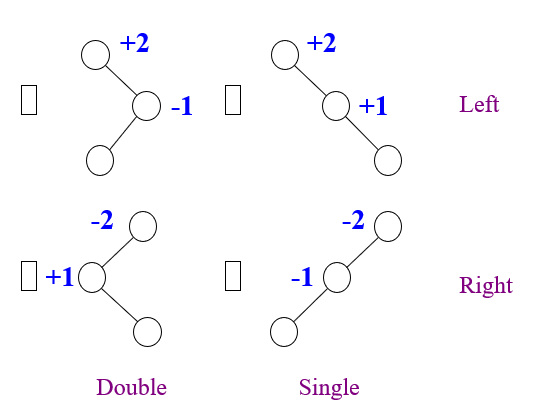
Binary Search Tree중 모든 노드에서 왼쪽과 오른쪽의 높이차이가 1 이하인 트리를 말한다.

트리에 노드를 Insert하다가 높이차이가 2이상이 되면 바로 높이를 조정해서 모든 노드에서의 왼쪽과 오른쪽의 높이차이를 1 이하로 유지한다.

높이차이가 2 이상이 되는 경우는 4가지 경우가 있다.

Single Left Rotation, Double Left Rotation, Single Right Rotation, Double Right Rotation

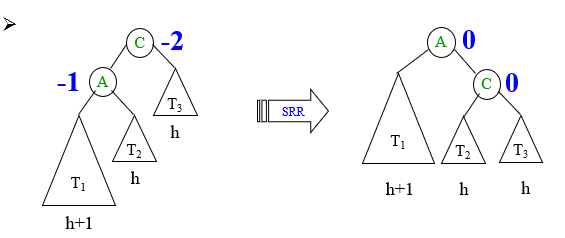
4가지 경우를 구별하는 기준은 다음과 같다.

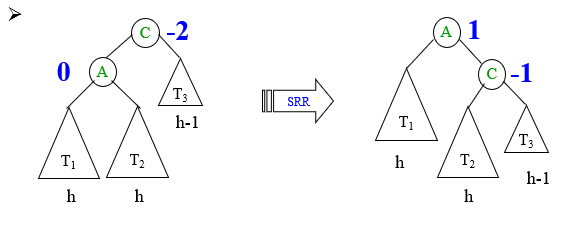


Single Left Rotation과 Double Left Rotation은 높이가 (+2)일 때 사용되며,

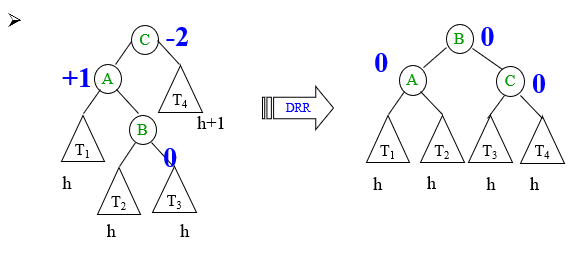
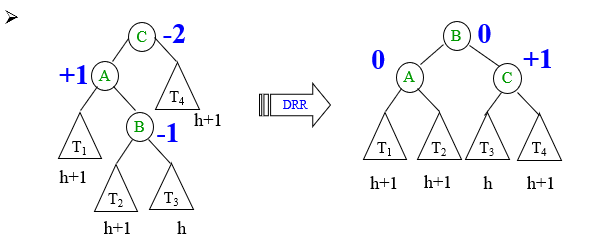
Single Right Rotation과 Double Right Rotation은 높이가 (-2)일 때 사용된다.

**-Single Right Rotation의 예이다.**



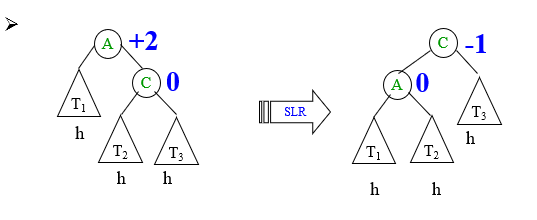


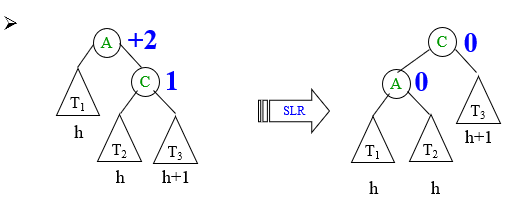
**-Double Right Rotation의 예이다.**



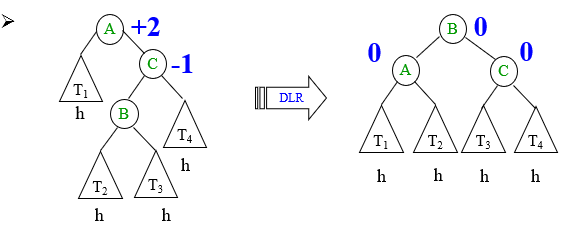
Double Right Rotation은 A에 Single Left Rotation을 취하고, C에 Single Right Rotation을 취하면 된다.

**-Single Left Rotation의 예이다.**





**-Double Left Rotation의 예이다.**



Double Left Rotation은 C에 Single Right Rotation을 취하고, A에 Single Left Rotation을 취하면 된다.

3. simulation

Binary Search Tree와 AVL트리의 높이를 구하기 위해서는 루트 노드의 Height값을 구하면 된다. 트리의 Depth와 Height는 서로 반대이기 때문에 Height값은 루트 노드가 가지고 있다.

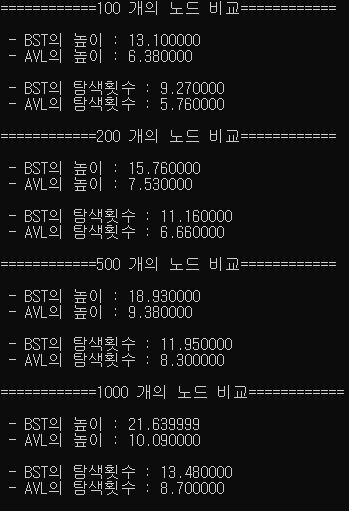
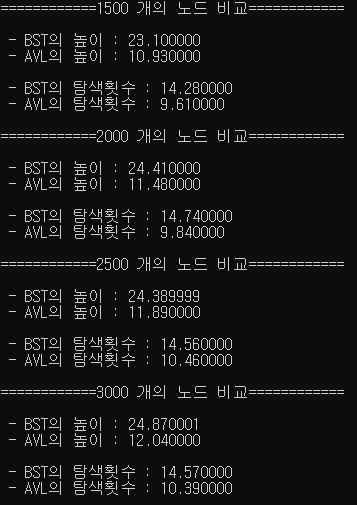
Binary Search Tree와 AVL트리의 탐색시간을 구하기 위해서 Search함수에 count변수를 추가하였다. Count변수는 Search함수가 재귀호출 될 때 마다(노드를 거칠 때 마다)1씩 증가하여 값을 찾아냈을 때 리턴된다.

100개~3000개의 노드를 가진 트리를 각각 100개씩 만들어야 하기 때문에, 100개~3000개의 노드를 가진 트리 각각을 만드는 것을 100번 반복하도록 했고, Height와 탐색시간을 따로 저장한 후 트리를 바로 삭제함으로 불필요한 메모리의 유지를 줄였다.

메모리의 삭제는 Postorder를 사용했는데 leaf노드부터 차례차례 삭제해 가기 위함이다.

마지막으로 Height와 탐색시간이 입력된 배열의 원소들에 100을 나눈다.(평균을 구하기 위함) 이전에 예상했던 것처럼 BST보다 AVL트리가 무조건적으로 Height와 탐색시간이 더 적을까?

실행 결과는 다음과 같다.

노드의 수가 많아질수록 높이의 변화가 작아지지만, BST와 AVL트리의 높이차는 증가하는 것을 볼 수 있다.

노드 안에 있는 랜덤 값을 찾는 탐색시간은 높이와 비례할 것이라 예상하였는데 높이가 증가할수록 탐색시간이 늘어나는 것을 확인할 수 있다.

4. 결론

* Binary Search Tree와 AVL트리를 동일한 데이터를 사용하여 만들었을 때, Binary Search Tree의 Height보다, AVL트리의 Height가 더 작다.
* Binary Search Tree와 AVL트리를 동일한 데이터를 사용하여 만들고 동일한 노드를 탐색했을 때, Binary Search Tree의 탐색시간 보다, AVL트리의 탐색시간이 더 작다.
* 트리의 높이가 증가할수록, 데이터를 탐색하는 시간도 증가한다.
* 트리를 만드는 시간을 배제했을 때, Binary Search Tree와 AVL트리중 탐색에 더 효율적인 트리는 AVL트리이다.
* 노드의 수가 증가할수록 트리의 높이의 변화는 더 적어진다. (트리의 높이가 lgn 임을 볼 수 있다.)