1. 개요

   Binary Search Tree에 대해 알아보고 기본 BST보다 개선된 Adelson-Velskii and Landis에 대해 알아본다. 이후 실제 코딩을 통한 성능 차이를 비교한다.

2. 이론적 배경

   2.1 BST

   - 특정 노드를 기준으로 왼쪽 서브 트리의 모든 키값은 작아야 하고 오른쪽 서브 트리의 모든 키값은 커야 한다.
   - 시간 복잡도: O(h), BST가 Full / Complete Binary Tree인 경우 O(logn)이지만 한 쪽 방향으로 편향된 경우 O(n)이 된다.
   - 문제점: 주어지는 Input에 따라 최악의 경우 O(n)이 되므로 성능이 매우 저하되는 문제점이 존재한다.

   2.2 AVL

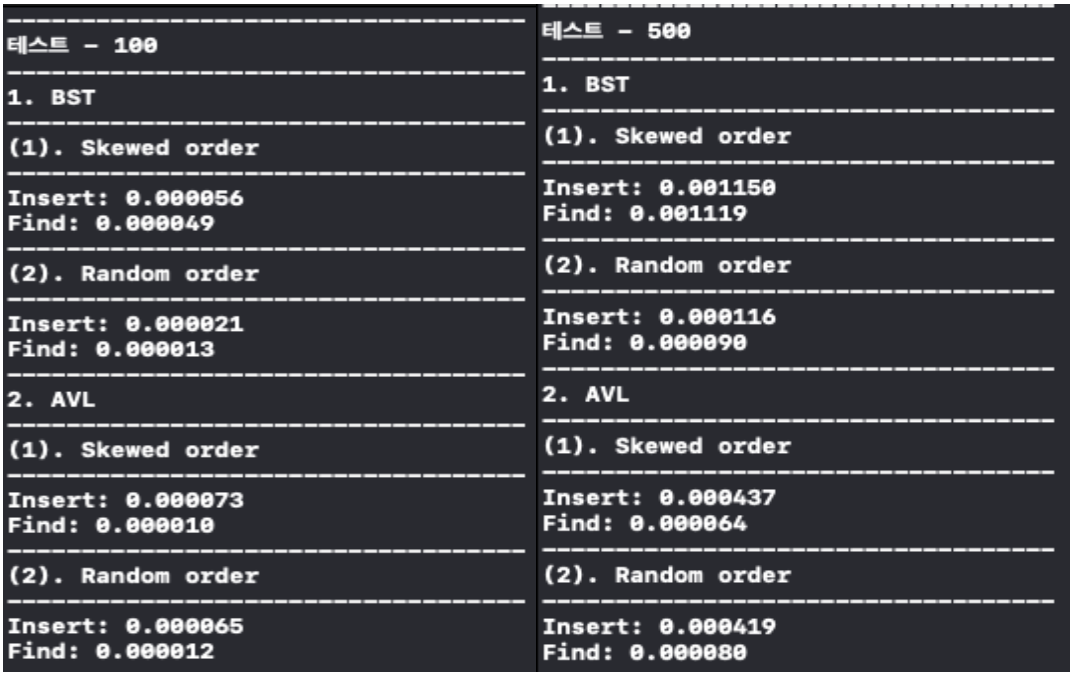
   - Height-balanced binary tree로써 각 노드의 balance factor가 -1, 0, +1 중 하나의 값을 갖는다. LL의 경우 right rotation, RR의 경우 left rotation, LR의 경우 (left + right) rotation, RL의 경우 (right + left) rotation을 한다.
   - insert의 경우 삽입된 노드에서부터 root 노드까지의 경로에 있는 노드의 balance factor가 변하므로 rotate를 통한 rebalance가 필요하다.
   - Random Order의 경우 rotation 확률은 0번(53.4%), 1번(23.3%), 2번(23.2)이다.
   - 시간 복잡도: O(logn), insert의 경우 동일한 키값이 존재하는지 search를 해야 되고 경우에 따라서 rotation 연산을 수행해야 하므로 search 보다 constant time이 더 소요된다.

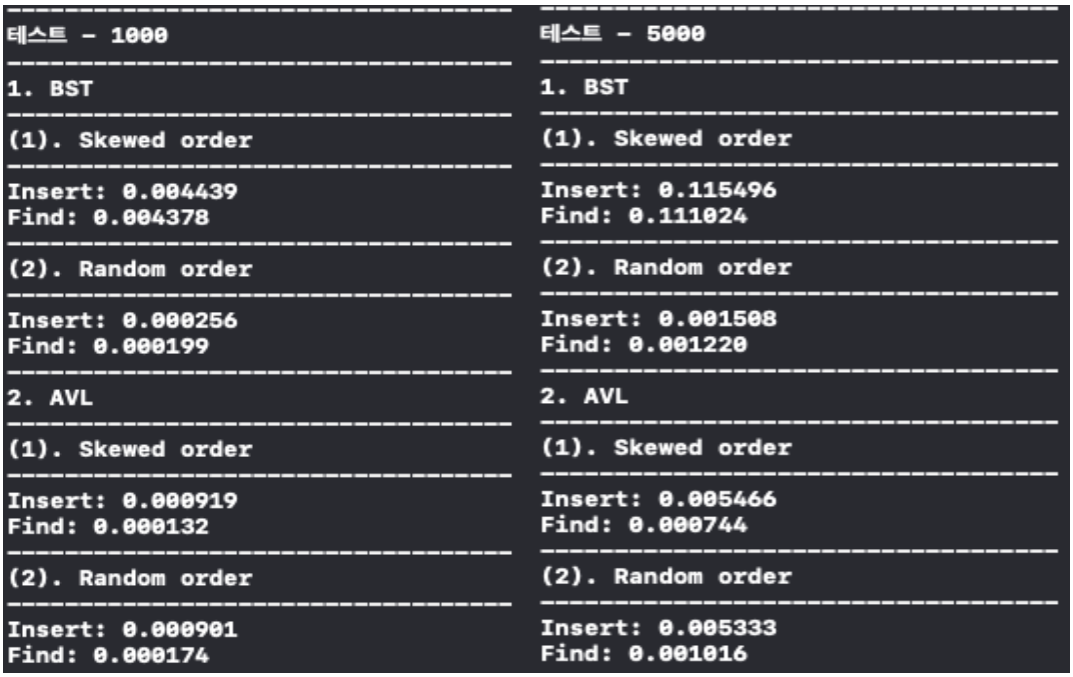
3. 구현 내용

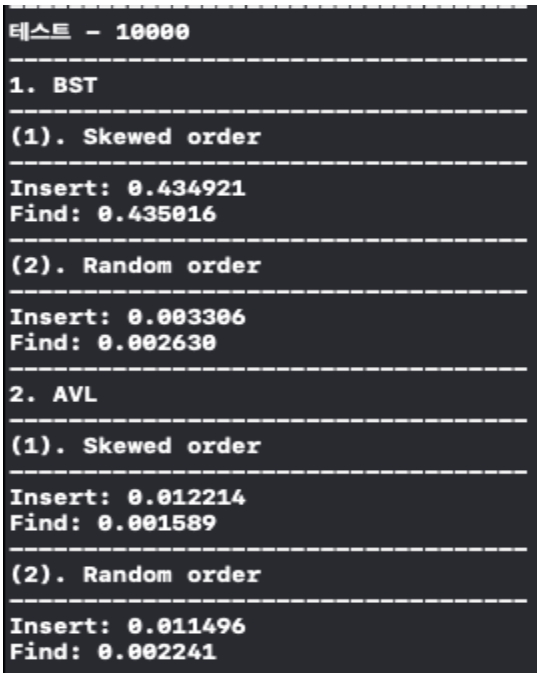
   Entry를 원소로 갖는 Node를 기반으로 LinkedBinaryTree를 구현한다. LinkedBinaryTree를 통해서 BST와 AVL를 구현한다. BST와 AVL에서는 추가적으

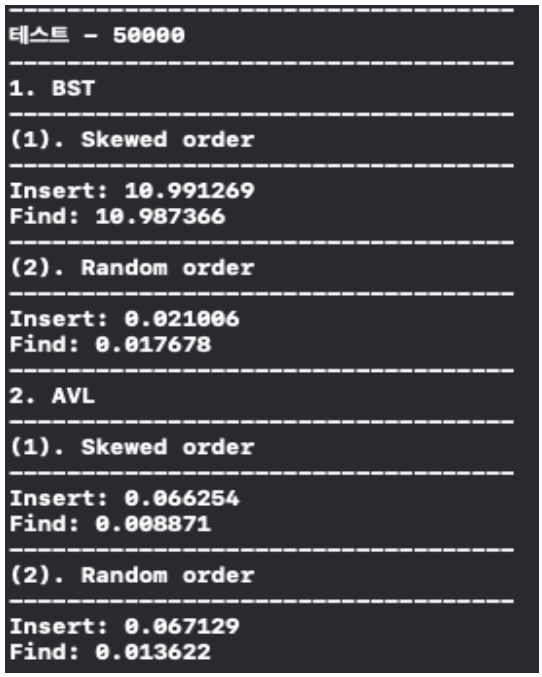
로 필요한 메소드와 변수를 선언한다. 예를 들어 AVL에서는 height 변수를 추가한 AVLEntry를 선언한다.

4. 테스트 케이스






테스트 - 10000
1. BST
(1). Skewed order
Insert: 0.434921
Find: 0.435016
(2). Random order
Insert: 0.003306
Find: 0.002630
2. AVL
(1). Skewed order
Insert: 0.012214
Find: 0.001589
(2). Random order
Insert: 0.011496
Find: 0.002241

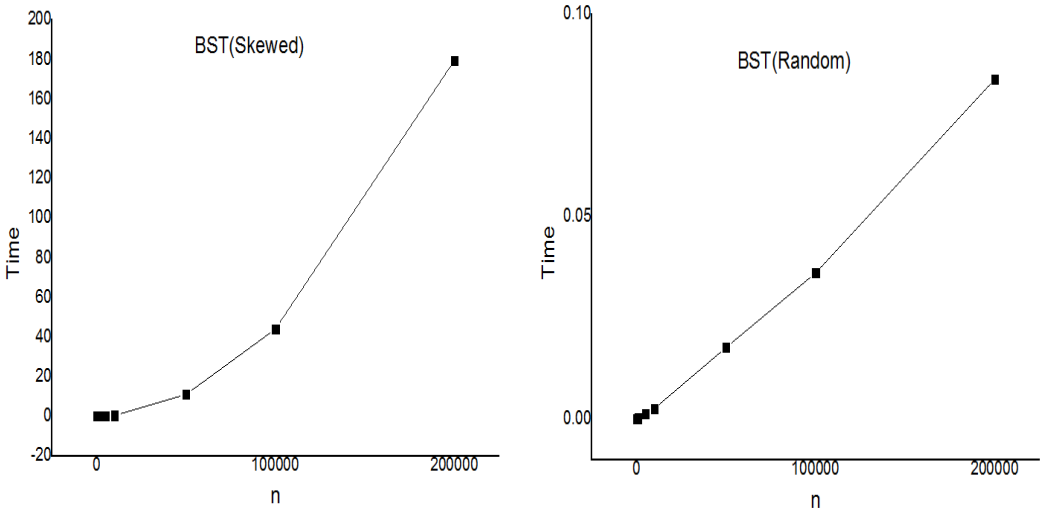

테스트 - 50000
1. BST
(1). Skewed order
Insert: 10.991269
Find: 10.987366
(2). Random order
Insert: 0.021006
Find: 0.017678
2. AVL
(1). Skewed order
Insert: 0.066254
Find: 0.008871
(2). Random order
Insert: 0.067129
Find: 0.013622

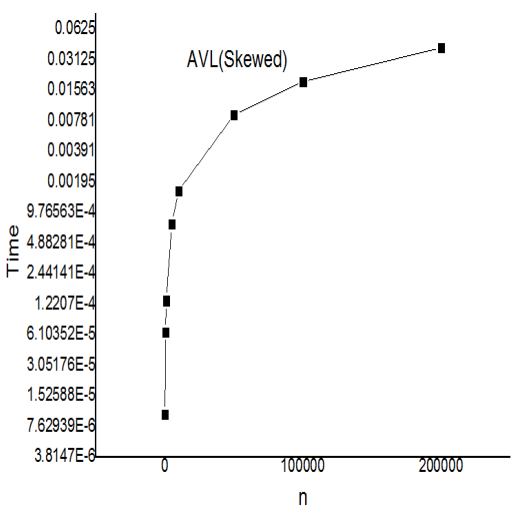

테스트 - 100000
1. BST
(1). Skewed order
Insert: 44.044498
Find: 44.063751
(2). Random order
Insert: 0.042475
Find: 0.036089
2. AVL
(1). Skewed order
Insert: 0.145459
Find: 0.018868
(2). Random order
Insert: 0.143481
Find: 0.030002

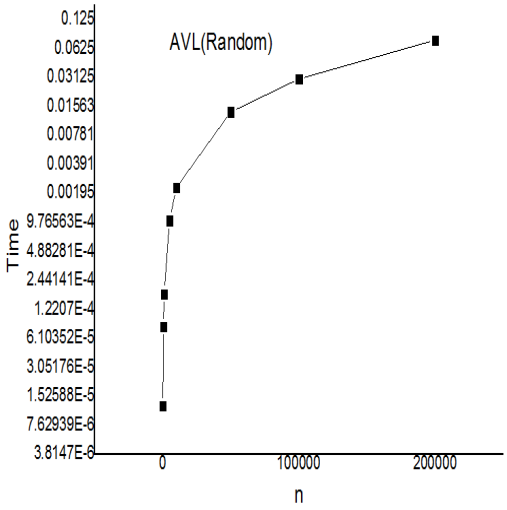

테스트 - 200000
1. BST
(1). Skewed order
Insert: 179.738388
Find: 179.432419
(2). Random order
Insert: 0.095448
Find: 0.083896
2. AVL
(1). Skewed order
Insert: 0.297327
Find: 0.040664
(2). Random order
Insert: 0.318485
Find: 0.075934

	100		500		1000		5000	
	Skewed	**Random**	**Skewed**	**Random**	**Skewed**	**Random**	**Skewed**	**Random**
BST	Insert: 0.000056 Find: 0.000049	Insert: 0.000021 Find: 0.000013	Insert: 0.001150 Find: 0.001119	Insert: 0.000116 Find: 0.000090	Insert: 0.004439 Find: 0.004378	Insert: 0.000256 Find: 0.000199	Insert: 0.115496 Find: 0.111024	Insert: 0.001508 Find: 0.001220
AVL	Insert: 0.000073 Find: 0.000010	Insert: 0.000065 Find: 0.000012	Insert: 0.000437 Find: 0.000064	Insert: 0.000419 Find: 0.000080	Insert: 0.000919 Find: 0.000132	Insert: 0.000901 Find: 0.000174	Insert: 0.005466 Find: 0.000744	Insert: 0.005333 Find: 0.001016

	10000		50000		100000		200000	
	Skewed	**Random**	**Skewed**	**Random**	**Skewed**	**Random**	**Skewed**	**Random**
BST	Insert: 0.434921 Find: 0.435016	Insert: 0.003306 Find: 0.002630	Insert: 10.991269 Find: 10.987366	Insert: 0.021006 Find: 0.017678	Insert: 44.044498 Find: 44.063751	Insert: 0.042475 Find: 0.036089	Insert: 179.738388 Find: 179.432419	Insert: 0.095448 Find: 0.083896
AVL	Insert: 0.012214 Find: 0.001589	Insert: 0.011496 Find: 0.002241	Insert: 0.066254 Find: 0.008871	Insert: 0.145459 Find: 0.018868	Insert: 0.143481 Find: 0.030002	Insert: 0.014765 Find: 0.002660	Insert: 0.297327 Find: 0.040664	Insert: 0.318485 Find: 0.075934







5. 테스트 케이스 분석

- Insert와 Find를 비교해 보면 BST와 AVL 모두에서 Insert가 조금 더 오래 걸리는 것을 볼 수 있다. 이는 Insert의 선결 조건으로 Find가 실행되기 때문이다. 더욱이 AVL에서는 rotation 연산을 통한 restructure, rebalancer 연산이 추가적으로 필요하기 때문에 그 차이가 BST보다 큰 것을 볼 수 있다.
- Find연산에서 대부분 Skewed, Random 모두 AVL이 BST보다 빠른것을 볼 수 있다. 특히 Skewed의 경우 그 차이가 매우 큰 것을 볼 수 있는데 이는 n의 값이 커질수록 Height-balanced 효과가 기하 급수적으로 커지기 때문이다. Random의 경우 rotation를 통한 constant time 소모로 Height-balanced를 유지하는 것이 더 효과적임을 암시한다.
- BST의 경우 Skewed와 Random의 차이가 매우 커지는 것을 볼 수 있다. 앞서 살펴본 바와 같이 Skewed의 경우 시간 복잡도는 O(n)이 된다. 따라서 테스트 케이스의 숫자가 커질수록 Find 연산에 소요되는 시간이 매우 커지는 것을 볼 수 있다. Random의 경우 완벽하지는 않지만 어느 정도 높이가 일정한 tree로 분산이 되므로 월등히 소요시간이 줄어드는 것을 볼 수 있다.
- AVL은 Skewed와 Random의 차이가 크지 않다. 어느 방법을 취하든 Insert 연산에 따른 restructure 연산으로 balance가 일정하게 유지되기 때문이다.

- AVL Insert연산의 경우 대부분 Skewed가 Random보다 더 많은 시간이 소요되는 것을 볼 수 있다. Skewed의 경우 Insert 연산시 일정한 패턴으로 한 번의 rotation 연산이 수반된다. Random의 경우에는 앞서 살펴본 확률로 rotation 연산을 수행하거나(1번 또는 2번) 수행하지 않는다. 이와 같은 이유로 미미한 차이지만 Random이 조금 더 빠른 연산을 수행하는 것으로 생각된다.
- 데이터를 기반으로 그래프를 plotting 한 결과 BST의 경우 skewed, random 모두 직선 형태로 기울기가 나왔다. AVL의 경우 Time의 scale를 log로 하였고 그 결과 log 그래프 형태로 기울기가 나왔다.

6. 고찰

이번 과제를 통해 수업 시간에 배운 Binary search tree를 cpp 언어를 통해 직접 구현해 보았다. 처음에는 문법이 생소하여 구현에 있어 어려움이 따랐지만 수업내용과 교과서를 통해 큰 무리 없이 로직을 구현할 수 있었다. Input으로 주어지는 n의 크기에 따라 소요되는 시간을 직접 측정하고 이를 통해 테이블과 그래프를 생성하였다. 예상한 대로 시간 복잡도 그래프 형태를 관찰할 수 있었다. BST Skewed의 경우 n의 값이 매우 커질 경우 finder 연산의 계속되는 재귀 호출로 segmentation falut 오류가 발생하는 것을 발견할 수 있었다. 따라서 finder 메소드는 while문을 통하여 구현하였다. 이번 과제를 통해 cpp 언어에 익숙해질 수 있었고 BST에 대해 깊이있는 이해를 할 수 있었다.