**HW5: BST VS BBST(AVLT)** 202211390 최준원

|  |
| --- |
| 1/ Random Insert, Random Find  위의 두 tree에 대해서, 1~10만까지의 integer를 random하게 발생시켜서 insert하는 작업을 10만번 수행한 후 수행 시간을 측정하여 1회 평균 수행시간을 구하시오. 그리고, 1~10 만까지의 숫자를 random하게 발생시켜 find하는 작업을 10만번 수행하는 실험을 하여 수행 시간을 측정하여, 1회 평균 수행시간을 구하시오. 이러한 실험을 20만, 30만, ..., 100만까지 수행(범위도 같이 증가)하여, 1회 수행 시간에 대한 time complexity의 간단한 결론을 제시하시오. 단, random한 수의 범위만이 아니라, 종류도 10만(20만, 30만, ....)개이어야 실험결과가 제대로 나올 수 있습니다. 예를 들어, data = (rand() \* 111) % 100000 이라고 해도, data의 범위는 0~10만이지만, 그 종류는 32767개입니다. 큰 수 범위에 대한 random number 생성에 대한 방법을 조사해서 이용하세요.  <스크린샷>  텍스트, 스크린샷, 흑백, 디자인이(가) 표시된 사진  자동 생성된 설명텍스트, 스크린샷, 메뉴, 흑백이(가) 표시된 사진  자동 생성된 설명  위의 결과값을 그래프로 정리해보면 아래와 같다.  위 그래프의 값들은 Binary Search Tree와 Balanced Binary Search Tree(AVL Tree)에 random한 정수 10만개를 random하게 Insert하고 10만개를 random하게 find하는 operation을 수행하고 그 소요 시간을 10만으로 나누어 1회 operation에 걸리는 평균 시간을 구한 값들이다. 여기서insert할 정수의 범위는 수행시마다 달라지나, 그 수는 10만개로 고정이고, find는 해당 트리에 insert가 시도된 횟수만큼이 범위이다. 예를 들면, 0인 경우에 10만개의 값을 insert하고 find할때의 정수의 범위는 1 ~ 100000이다. 100000인 경우에는 insert할 정수의 범위는 100001 ~ 200000이고, find할 정수의 범위는 1 ~ 200000이다. 이러한 식으로 random하게 insert하고 find할 10만개 정수의 대한 범위를 정한다. 그래프에서의 분포와 같이 원소의 수가 늘수록 수행 시간이 선형적으로 증가하지 않는데, 이는 random한 insert를 수행했기 때문으로, 한쪽으로 몰아 insert하는 경우(최악의 경우)보다는 함수가 재귀적으로 호출되는 빈도가 반드시 적어 Time Complexity가 O(logn)과 비슷하게 결정된다. Insert에서는 상대적으로 Balanced Binary Search Tree가 시간이 더 오래걸리는데, 이는 트리의 높이의 균형을 맞추기 위해 노드에 회전을 수행하기 때문이다. 이 추가적인 처리로 인해 수행시간이 더 걸리지만, 높이 균형이 맞춰졌기 때문에 이 또한 Time Complexity가 O(logn)이다. Find에서는 반대로 Binary Search Tree가 더 시간이 많이 걸리는데, 이는 위에서 말한 높이의 균형으로 설명할 수 있다. Balanced Binary Search Tree는 이미 Insert시 Tree를 정리(?)하기 때문에 나중에 find 수행 시 비교적 일정한 높이로 참조할 수 있다. 하지만 Binary Search Tree는 그렇지 않기 때문에 특정 값을 find하기 위해 더 많은 recursive call을 해야할 수도 있다. 따라서 Find에서는 Binary Search Tree의 시간이 좀 더 걸리는 경향을 보인다. |
| 2/ Ordered Insert, Random Find  위의 두 tree에 대해서, 1, 2, 3, ..., 1000까지의 integer를 순서대로 insert하는 작업을 수행한 후, 수행 시간을 측정하여, 1회 평균 수행시간을 구하시오. 그리고, 1~1000사이의 숫자를 random하게 발생시켜 find하는 작업을 1000번 수행하여 수행 시간을 측정하여, 1회 평균 수행시간을 구하시오. 이러한 실험을 2000번, 3000번, 4000회, 5000회(범위도 같이 증가)를 실하여, 1회 수행시간에 대한 time complexity의 대한 간단한 결론을 제시하시오.  <스크린샷>  텍스트, 스크린샷, 흑백, 디자인이(가) 표시된 사진  자동 생성된 설명  위의 결과값을 그래프로 정리해보면 아래와 같다.  위 그래프의 값들은 Binary Search Tree와 Balanced Binary Search Tree(AVL Tree)에 정수 1000개를 순서대로 Insert하고 1000개를 random하게 find하는 operation을 수행하고 그 소요 시간을 1000으로 나누어 1회 operation에 걸리는 평균 시간을 구한 값들이다. 여기서도 insert할 정수의 범위가 수행시마다 달라지고, 그 수는 1000개로 고정이다. 예를 들면, 0인 경우에 1000개의 값을 insert하고 find할때의 정수의 범위는 1 ~ 1000이다. 1000인 경우에는 1001 ~ 2000이다. 이때 find할 정수의 범위는 1 ~2000인 것이다. 이러한 식으로 순서대로 insert하고 find할 1000개 정수의 대한 범위를 정한다. 1번과는 다르게 insert하는 값들이 순서가 정해져 있어(1씩 커지는 순서) 트리에 insert할 때마다 오른쪽 한 방향으로만 insert될 것이다. 따라서 Balanced Binary Tree는 오른쪽 맨 끝에 삽입하는 경우 트리의 높이 차가 2가 넘는 순간마다 왼쪽 방향으로 Rotate을 한다. 그래서 이로 인한 시간이 더 걸린다고 생각할 수 있지만, 기본적으로 Binary Search Tree에 형성되는 트리가 오른쪽으로 편향된 삽입으로 인해서 Skewed Tree가 되기 때문에, 시간이 갈수록 Insert나 find하는 시간이 선형적으로 증가한다. 위에 그래프에서도 Binary Search Tree에서의 수행시간이 선형적으로 증가함이 보인다. 즉, Binary Search Tree의 경우 2개의 수행 모두 최악의 경우로 O(n)의 Time Complexity를 가진다. Balanced Binary Search Tree의 경우, insert되고 find되는 요소의 수가 Binary Search Tree만큼의 시간을 재기엔 턱없이 부족한 것인지 이 코드를 실행한 컴퓨터에서는 유의미한 시간을 잴 수가 없었다. (위에 나오는 조그만 숫자의 차이는 그저 clock(); 함수의 값이 1ms만큼 변하는 부분을 구간 내에 두고 있기 때문에 발생하는 변화이다.) 실제로 1번문제에서의 결과가 나온 것도 있고, 디버깅해보았을 때 필요한 함수를 호출하지 않았다든가 하는 오류는 없었기 때문에 시간을 더 세세히 젤 수 없어 발생하는 문제라고 잠정적인 결론을 내려본다. |