□ 프로그램 구현 환경

windows 10 운영체제에서 Eclipse Jee Oxygen를 이용해 java언어로 작성.

□ 함수 설명 및 시간 복잡도 \*소스코드번호 \* N:트리 안에 데이터 수

|  |  |
| --- | --- |
| **자료구조**  **함 수** | **AVL Tree** |
| InOrder | root부터 Inorder travesal방식으로 출력  모든 노드를 방문하므로 시간 복잡도 ⨁(N) |
| Find | root부터 시작해 데이터를 찾아 해당 데이터를 가진 노드를 반환  재귀방식으로 데이터와 root의 데이터를 비교해서 왼쪽, 오른쪽자식으로 호출  최악의 경우 가장 밑에 있는 데이터일 경우 O(lnN) |
| Delete | 데이터를 가진 노드를 삭제하는 메서드  세 가지 경우로 나눠서 진행. 삭제노드의 자식들이 모두 없을 경우  삭제노드의 자식 중 하나만 없을 경우. 자식 모두 있을 경우  root의 왼쪽과 오른쪽 자식트리를 확인해서 각각의 경우 처리하는데  둘 다 null일 경우에는 root의 서브트리를 null로 설정  한 쪽만 null일 경우 다른 자식을 삭제시킨 노드를 대체시키면 되므로 null이  아닌 자식 리턴, 둘 다 null이 아닌 경우에 삭제할 노드보다 큰 수 중 가장 작은  노드를 찾는 데 가장 작은 노드를 cNode, cNode의 부모를 pNode로 저장해놓고 삭제할 노드의 데이터를 cNode의 데이터로 바꾸고 cNode의 상태에 따라 4경우로 나눠서 진행. cNode가 삭제노드와 일직선일 경우 삭제노드의 오른쪽을 cNode의 오른쪽 자식이나 null로 해주고 일직선이 아닐 경우 cNode의 자식이나 null을 pNode의 왼쪽으로 붙여준다.  균형 맞추지 않고 삭제만으로의 시간복잡도는  데이터를 찾는 시간O(lnN) + 찾는 데이터 보다 큰 데이터 중 가장 작은 데이터 찾는 시간인데 최악의 경우 마지막 노드까지 찾으므로 O(ln(H-K))  (depthK에서 depthH까지 노드 수)이므로 이 함수 시간복잡도는 O(ln(N)) |
| Erase | Delete메서드 실행 후에 root부터 무너진 곳을 찾아가며 무너진 지점부터 균형 맞추는 메서드. IsAVL메서드로 전체 트리 중에 균형이 무너진 지점 찾아  그 노드 반환해주고 없으면 NULL리턴한다. unb노드가 무너진 지점 부모노드이며  가장 최초부모가 무너졌을 시 최초루트를 균형맞춘 후 최초부모 재설정  무너진 지점 데이터를 임시로 저장해 놓고 무너진 서브트리를 ReRotate메서드로  균형 맞춘 서브트리를 임시로 저장해 놓은 데이터를 원래 트리에서 찾아  균형 맞춘 서브트리로 대체시킨다.  시간복잡도는 Delete연산시간(lnN)+균형 무너진 곳 찾는 시간 IsAVL 최악의 경우  전부 탐색하므로(N)\*while문으로 전체 트리 균형맞춰질 때까지 탐색 K\*  root트리에 무너진 지점 서브트리를 균형 조절한 서브트리로 바꾸는 시간  (cNode, pNode 설정 시간(lnN)+서브트리 바꾸는 시간(1))  = lnN + KNlnN = O(NlnN) |

|  |  |
| --- | --- |
| IsAVL | Tree의 균형이 무너진 노드 확인하여 없을 시 NULL리턴 있을 시 무너진 노드  반환한다. 최악의 경우 모든 노드 탐색하기 때문에 N시간  O(N) |
| ReRotate | 어떤 노드 삭제 후 무너진 균형 맞추기 위한 메서드.  IsAVL로 무너진 노드 포인트를 인자로 받아 그 노드 중심으로 해서  회전방법 결정 해 조정한 후 균형맞춘 서브트리를 반환한다. 자식들 높이 값 비교  후 이동해서 회전방법 결정. 높이에 따라 이동해가기 때문에 O(lnN)  회전하는데 걸리는 시간은 상수시간이므로. |
| SingleRotateLeft  (L-L 회전) | 그림입니다.노드를 연결해 주기만 하면 되므로 시간 복잡도 O(1) |
| SingleRotateRight  (R-R 회전) | SingleRotateLeft와 마찬가지 방향만 다르게 해서 |
| DoubleRotateWith  Left (L-R회전) | 그림입니다.  두 번의 SingleRotate메서드 호출 O(1)\*O(1) = O(1) |
| DoubleRotateWith  Right(R-L회전) | DoubleRotateWIthRight와 마찬가지 방향만 다르게 해서 |
| Insert | 삽입할 데이터를 비교해가며 Insert함수를 재귀호출해 현재 노드와 부모 노드를  설정해가며 내려가다가 null인 노드를 만나면 그 자리가 삽입될 자리이므로  새로운 노드를 만들고 리턴시켜 다시 위쪽으로 서브트리를 설정해가며  올라가는데 올라가다가 왼쪽과 오른쪽 높이를 비교해서 높이차가 2가 나면  삽입으로 인해 균형이 무너졌으므로 데이터의 위치에 따라 균형방법을 선택해 맞춘다. 데이터의 위치를 찾는 시간 lnN + 균형조절하는 시간 1이므로  Insert의 시간복잡도는 O(lnN) |
| Height | e의 노드와 왼쪽, 오른쪽 노드의 키 값 출력 메서드  e를 가진 노드 찾는 시간 O(lnN)이므로 시간복잡도는 O(lnN) |
| resetHeight | 노드들 키 재설정해주는 메서드  모든 노드 탐색해서 키 설정해주므로 O(N) |