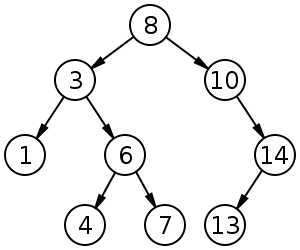
이진탐색트리 한계점과 극복방법. + 순회 방법/순서 조사



이진탐색트리 (Binary Search Tree)

효율적인 탐색을 위해 데이터를 저장하는 규칙을 가지며 해당 규칙은 특정 데이터의 위치를 찾는데 사용된다. 이진탐색과 연결리스트 둘의 장점을 챙긴 자료구조다. 탐색의 효율을 높이고 자료의 삽입과 삭제도 가능하게 하는 것이 핵심이다.

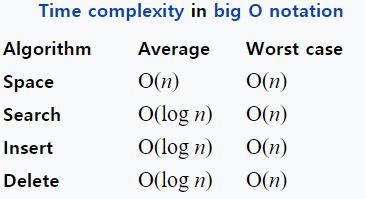
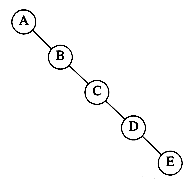
이진 탐색트리의 특징

- 이진 탐색 트리의 노드에 저장된 키는 유일하다. (검색 목적 자료구조이기 때문)

=> 클래스는 SortedSet<T> (정렬이 보장되어 있다.) 중복값을 허용하지 않는다.

- 각 노드의 왼쪽 자식 노드들은 부모보다 작고, 오른쪽 자식 노드들은 부모보다 크다.

- 부모 노드의 왼쪽과 오른쪽 서브트리도 이진탐색트리다.



▲ 이진탐색트리 시간복잡도와 편향 트리의 구조

이진 탐색트리의 한계점

이진 트리의 탐색 연산의 시간 복잡도는 보통 O(logN)이지만 높이에 따라 O(H)의 시간 복잡도를 갖는다. 트리의 높이가 더해질수록 추가할 수 있는 노드의 수는 두 배씩 증가하기 때문이다. 이 때 저장 순서로 인해 노드가 한 쪽으로만 추가되어 **편향 트리가 될 경우 탐색 영역이 절반으로 줄어들지 않아 시간복잡도가 증가**된다.

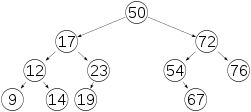
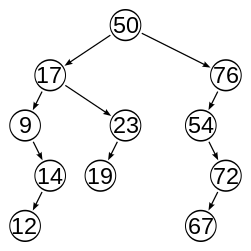
배열보다 많은 메모리를 사용하며 데이터를 저장하였지만 탐색에 필요한 시간복잡도가 같에 되어 비효율적이 된다. 이것이 이진 탐색트리의 한계점이라고 할 수 있다.

- 균형트리 : 모든 하위 트리의 높이 차가 1 이하인 트리

- 자가균형 이진탐색트리 : 트리에서 노드의 삽입이나 삭제와 같은 연산이 일어날 때 자동으로 균형 유지

한계점 극복 방법

편향 트리의 현상을 막기 위해 자가 균형기능을 추가한 트리의 사용이 일반적인 해결방법이다. 노드의 균형을 맞추기 위해 삽입과 삭제가 일어나는 경우에 자동으로 그 높이(루트에서부터 내려갈 수 있는 최대 레벨)를 작게 유지하는 방법이다. 대표적인 방법으로는 AVL Tree, Red-Black Tree 등 이 있다.

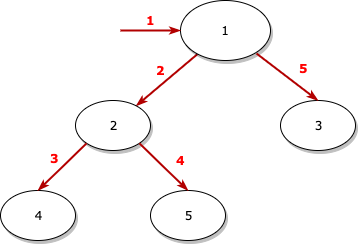


왼쪽은 균형이 맞지 않는 트리의 예이고 오른쪽은 높이 균형을 맞춘 후의 상태이다. 탐색의 이동 횟수 평균이 확실히 줄어든걸 알 수 있다.

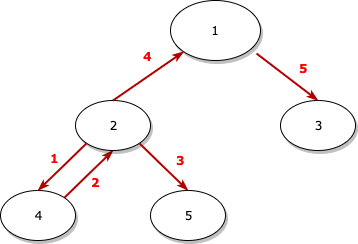
이진 탐색트리의 순회 방법 ( + 순서 )

이진 탐색트리는 특정 값을 탐색하는 것에 빠른 성능을 보여준다. 트리의 부모가 최대 두 개의 자식 노드만을 가지는 이진트리에서 DFS 기반 탐색 알고리즘인 전위 순회, 중위 순회, 후위 순회를 적용하여 전체 노드를 탐색할 수 있다.

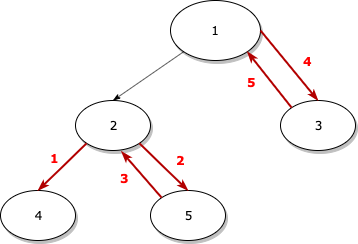
1. 전위순회 : 노드, 왼쪽, 오른쪽



전위순회는 루트 노드 -> 왼쪽 노드 -> 오른쪽 노드 순서로 트리의 노드들을 방문하는 방법이다. 위 그림과 같이 전위 순회를 진행하면 1 -> 2 -> 4 -> 5 -> 3 순서로 노드를 방문한다.



2. 중위순회 : 왼쪽, 노드, 오른쪽 => 중위 순회는 왼쪽 노드 -> 루트 노드 -> 오른쪽 노드 순서로 트리의 노드들을 방문하는 방법이다. 이진 탐색트리에서 중위순회를 진행할 경우 오름차순으로 정렬된다. 4 -> 2 -> 5 -> 1 -> 3 순서로 노드를 방문하게 된다.



3. 후위순회 : 왼쪽, 오른쪽, 노드 => 후위 순회는 왼쪽 노드 -> 오른쪽 노드 -> 루트 노드 순서로 트리의 노드들을 방문하는 방법이다. 전위, 중위 순회들과 다른 점은 루트 노드를 가장 마지막에 방문한다는 점이다. 4 -> 5 -> 2 -> 3 -> 1 순서로 노드를 방문한다.