**알고리즘 2진탐색트리 조사**

**2016156007**

**소프트웨어학과**

**김영민**

목차

[**1. 소개** 3](#_Toc57414467)

[**1) 리포트의 목적** 3](#_Toc57414468)

[**2. 알고리즘 설명 및 특성** 3](#_Toc57414469)

[**1) 이진 탐색 트리** 3](#_Toc57414470)

[**2) 이진 탐색 트리의 특성** 3](#_Toc57414471)

[**3) 이진 탐색 트리의 종류** 3](#_Toc57414472)

[**3-1) 배열을 활용한 이진 탐색 트리** 3](#_Toc57414473)

[**3-2) 연결 리스트를 활용한 이진 탐색 트리** 4](#_Toc57414474)

[**3-3) 균형 이진 탐색 트리(AVL tree)** 5](#_Toc57414475)

[**3-4) KD 트리(K-D tree)** 7](#_Toc57414476)

[**3-5) 레드 블랙 트리(Red-black tree)** 7](#_Toc57414477)

[**3-6) 스플레이 트리(Splay tree)** 10](#_Toc57414478)

[**3-7) B- 트리(B tree)** 11](#_Toc57414479)

[**3-8) B플러스 트리(B+ tree)** 12](#_Toc57414480)

[**3. 비교** 13](#_Toc57414481)

[**1) 모든 알고리즘의 비교** 13](#_Toc57414482)

[**4. 결론** 15](#_Toc57414483)

# **1. 소개**

## **1) 리포트의 목적**

a) 이진 탐색 트리의 종류와 각각의 알고리즘에 대해서 조사

b) 조사한 해당 트리 알고리즘의 특성에 대해 조사

c) 조사한 모든 이진 탐색 트리의 알고리즘을 비교

# **2. 알고리즘 설명 및 특성**

## **1) 이진 탐색 트리**

a) 자식 노드를 두개 이하로 가지는 이진 트리를 탐색을 목적으로 사용하는 자료구조

## **2) 이진 탐색 트리의 특성**

a) 이진 탐색 트리의 모든 노드들은 키 값을 가짐

b) 모든 노드의 왼쪽에는 해당 노드보다 작은 키 값, 오른쪽에는 큰 키 값을 배치

c) 수의 크기 비교를 해야 하기 때문에 키 값이 중복되는 노드는 없도록 함

## **3) 이진 탐색 트리의 종류**

### **3-1) 배열을 활용한 이진 탐색 트리**

a) 배열을 사용하여 이진 탐색 트리를 구현

b) 순서대로 첫 번째 들어오는 숫자는 1번 인덱스에 삽입

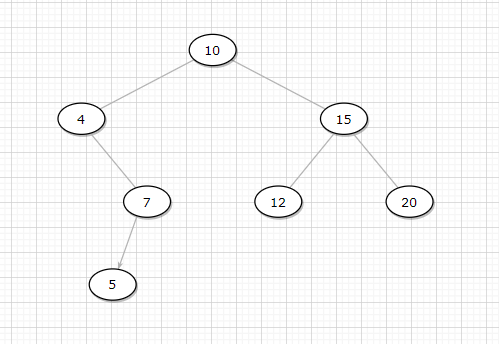
c) 이후 배열에 들어오는 순서대로 1번 인덱스부터 시작하여 x 번 인덱스의 숫자 보다 작은 값은 2x 번 인덱스에 이동시키고 큰 값은 2x + 1 번 인덱스에 이동시켜 빈자리를 찾을 때까지 반복

d) 탐색 속도는 좋으나 배열로 구현한 트리는 삽입, 삭제시에 배열의 공간을 임의적으로 조정이 어려움

[예제]

입력되는 값: 10 4 15 7 5 12 20

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 값 |  | 10 | 4 | 15 |  | 7 | 12 | 20 |  |  | 5 |



[사용한 모든 그림은 drawio 툴을 사용하여 작성함]

### **3-2) 연결 리스트를 활용한 이진 탐색 트리**

a) 이러한 배열에 단점을 보완하기 위해 연결 리스트를 활용하여 이진 탐색 트리를 구현

b) 포인터를 활용하여 노드들을 서로 연결시켜줘 배열처럼 비는 공간이 없음

c) 모든 노드의 왼쪽 서브 트리는 해당 노드의 키 값 보다 작은 값, 오른쪽 서브 트리는 해당 노드의 키 값보다 큰 값으로 구성함

d) 단 들어오는 값의 순서에 따라 탐색 시간의 시간 복잡도가 달라질 수 있음

e) 트리의 형태는 배열을 사용했을 때와 동일함

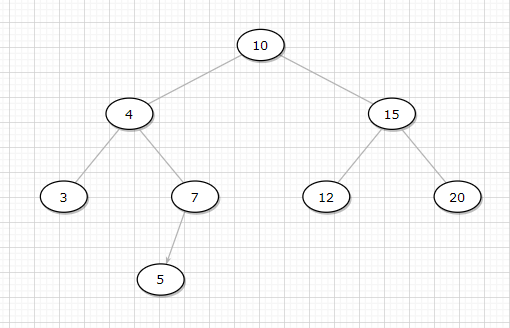
### **3-3) 균형 이진 탐색 트리(AVL tree)**

a) 이진 탐색 트리의 경우 들어오는 값이 한쪽으로만 들어갈 경우 탐색 시간에 악영향을 줄 수 있어 이를 보완하기 위해 균형 이진 탐색 트리를 활용

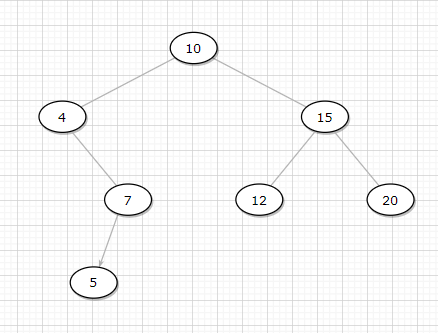
b) 트리의 삽입, 삭제를 진행할 때 각 노드의 왼쪽 서브 트리 높이와 오른쪽 서브 트리 높이의 차가 최대 1만큼만 차이가 나도록 트리를 구성함

c) 트리 높이의 균형이 깨질 때마다 트리를 재구성함

**[균형이 잡힌 AVL 트리]**

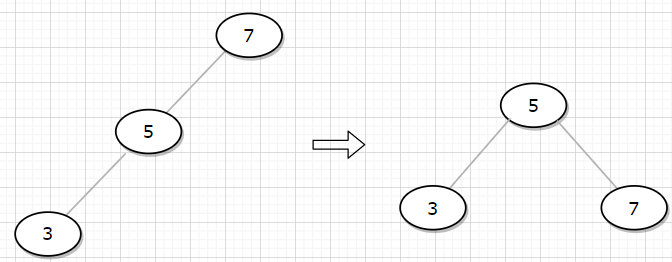


**[균형이 깨진 AVL 트리]**

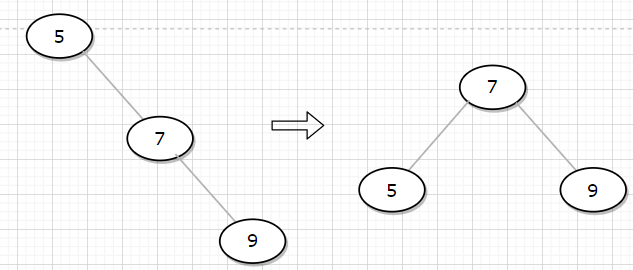


**트리의 균형이 무너질 때 유형 4가지**

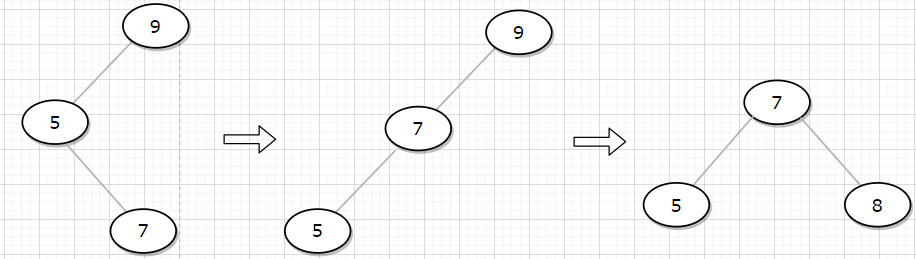
**[LL]**



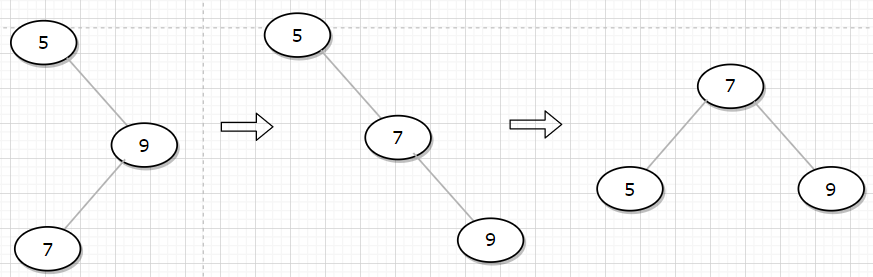
**[RR]**

****

**[LR]**

****

**[RL]**

****

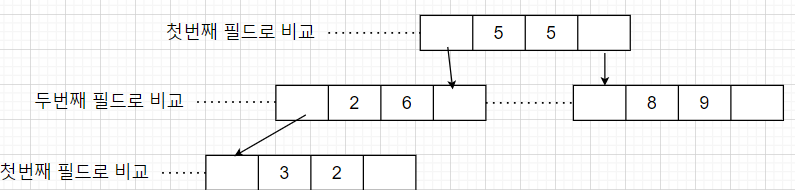
### **3-4) KD 트리(K-D tree)**

a) 이진 탐색 트리를 확장하여 두개 이상의 키 값을 사용하는 트리

b) 알고리즘은 이진 탐색 트리와 유사하지만 트리의 레벨에 따라서 필드를 번갈아 비교해서 트리를 생성

c) 동일한 레벨에 있는 값들은 동일한 필드만 이용한다.

[예시 - 키 값이 2개인 KD 트리]



### **3-5) 레드 블랙 트리(Red-black tree)**

1. 이진 탐색 트리에서 데이터가 한쪽으로만 편중될 경우에 탐색 속도가 떨어지는 문제를 개선하기 위해 레드 블랙 트리 사용
2. 노드에 검은색과 빨간색을 부여해서 트리의 균형을 효율적으로 유지시킴

c) 레드 블랙 트리도 이진 탐색 트리의 특성을 가짐

d) 레드 블랙 트리의 조건

- 모든 노드는 빨강색과 검은색 중 하나의 색을 가짐

- 루트 노드는 검은색을 가짐

- 모든 리프 노드는 검은색을 가짐

- 빨강색 노드의 자식은 모두 검은색을 가짐

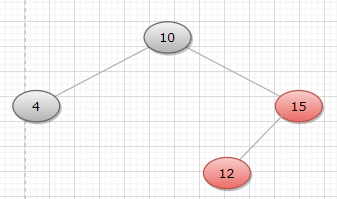
- 모든 노드들은 해당 노드부터 리프 노드까지 모든 경로에서 같은 개수의 검은색 노드가 존재

e) 위의 규칙을 지키면서 배열하되 문제가 생겼을 경우 Restructuring, Recoloring을 진행

f) 빨강 노드를 추가했을 때 부모 또한 빨강인 경우 삼촌 노드의 색이 검정이면 Restructuring, 빨강이면 Recoloring을 진행

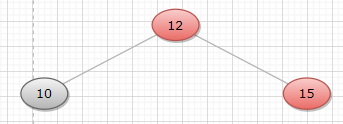
**Restructuring, Recoloring 예시**

**[Restructuring]**

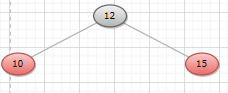


- 노드 12가 추가되었을 때 빨강색 노드는 빨강색 자식을 가질 수 없으므로 문제 발생

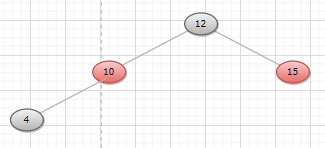
- 12의 삼촌 노드의 색이 검정이므로 Restructuring 진행



- 15의 자식, 부모, 본인을 정렬하고 가운데 값을 부모로 다시 트리를 작성

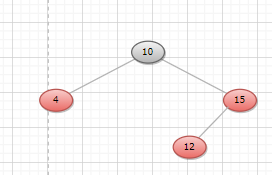


- 이후 부모인 12를 검정, 자식인 10, 15를 빨강으로 변경



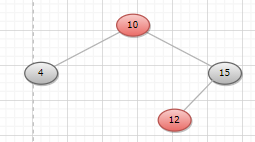
- 4를 다시 10의 자식으로 추가하여 완성

**[Recoloring]**

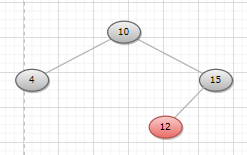
****

- 노드 12가 추가되었을 때 빨강색 노드는 빨강색 자식을 가질 수 없으므로 문제 발생

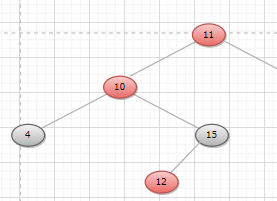
- 12의 삼촌 노드의 색이 빨강이므로 Recoloring 진행



- 삽입된 노드의 부모와 부모의 형제를 검정으로 바꾸고 부모의 부모를 빨강으로 바꾼다.



- 부모의 부모 노드가 루트 노드라면 검정색이여야 하므로 다시 변경



- 부모의 부모 노드가 루트 노드가 아니고 위에 노드가 더 존재하는데 빨강이라면 한단계 더 올라가 다시 상황에 맞게 Restructuring, Recoloring을 수행

- 최악의 경우 루트 노드까지 반복해서 진행

### **3-6) 스플레이 트리(Splay tree)**

1. 레드 블랙 트리의 구현이 어려움 단점을 보완하여 레드 블랙 트리에 근접한 성능을 내면서 구현이 단순한 트리
2. 최근에 접근한 노드를 이후 빠르게 재사용할 수 있도록 스플레이를 통해 루트로 올려 줌
3. 가능성은 낮지만 최악의 경우 노드가 한쪽으로 쏠려 높이가 n이 될 수 있음
4. 스플레이 하기

- 스플레이 방법은 2가지로 스텝 한번으로 수행(ZIG 스텝), 스텝 두번으로 수행(ZIG-ZIG 스텝, ZIG-ZAG 스텝) 방법이 있음

- 스텝 두번으로 수행하는 방법은 AVL 트리와 유사

- ZIG-ZIG 스텝 -> [LL], [RR]

- ZIG-ZAG 스텝 -> [LR], [RL]

e) 스플레이 과정

- 접근한 노드가 루트 노드라면 종료

- 접근한 노드의 부모가 루트라면 ZIG 스텝 수행

- 접근한 노드와 부모의 부모 노드의 관계를 파악하여 상황에 맞춰 ZIG-ZIG 스텝, ZIG-ZAG 스텝 수행

- 접근한 노드가 루트 노드가 될 때까지 이를 반복 수행

### **3-7) B- 트리(B tree)**

a) 이진 탐색 트리의 형태를 차용하고 하나의 노드에 여러 자료로 구성하는 트리 구조

b) 삽입, 삭제시에도 트리의 균형을 유지하도록 함

c) B 트리의 규칙

- 노드의 자료수가 n개 일 때 자식의 수는 n + 1개

- 자료는 노드에 정렬된 상태로 저장

- 각각 노드의 왼쪽 서브 트리는 작은 값, 오른쪽 서브 트리에는 큰 값이 위치

- 루트 노드는 적어도 2개 이상의 자식을 가짐

- 루트 노드를 제외한 모든 노드들은 해당 공간의 절반은 자료로 채워져 있어야 함(소수점은 버림)

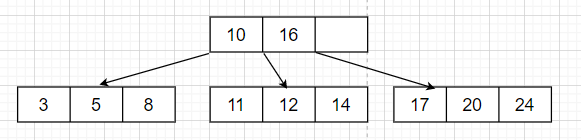
- 리프 노드는 모두 같은 레벨

- 입력된 데이터에 중복은 없음

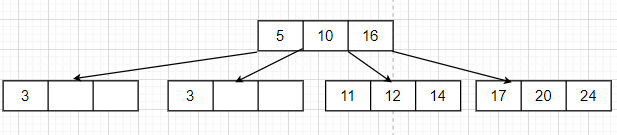
d) 탐색의 경우 이진 탐색 트리와 동일

e) 삽입의 경우 먼저 삽입될 부모 노드의 위치를 찾아 삽입

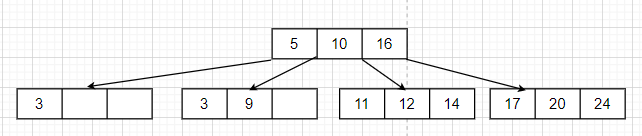
f) 삽입하려고 할 때 노드가 너무 많은 경우 노드를 분리하는 과정 수행 후 삽입



- 9를 삽입하고 싶으나 이미 노드가 꽉 차 있어 불가능



- 리프 노드의 중간 값인 5를 부모 노드로 올려주고 노드를 2개로 분할



- 이후 9를 삽입

g) 삭제의 경우에는 반대로 노드의 데이터 값이 공간의 절반 이하로 내려간 경우 형제 노드에게서 빌리거나 형제와 결합하거나, 대체 키를 찾아 대체

### **3-8) B+ 트리(B+ tree)**

a) B- 트리의 변형으로 B- 트리와 달리 인덱스 노드와 리프 노드가 분리됨

b) B+ 트리의 리프 노드는 서로 연결되어 있어 B- 트리의 단점인 중위 순회 성능을 올림

c) 삽입의 경우 B- 트리와 거의 유사하고 삭제의 경우 리프 노드만 삭제

d) B- 트리보다 트리의 높이가 낮아질 수 있다는 장점이 있음

e) B- 트리와 달리 탐색의 경우 반드시 루트 노드까지 내려가야 하는 단점이 있음

# **3. 비교**

## **1) 모든 알고리즘의 시간 복잡도 비교**

|  |  |  |  |
| --- | --- | --- | --- |
| **트리** | **탐색/삽입/삭제** | **평균** | **최악** |
| **이진 탐색 트리** | **탐색** | O(logN) | O(N) |
| **삽입** | O(logN) | O(N) |
| **삭제** | O(logN) | O(N) |
| **AVL 트리** | **탐색** | O(logN) | O(logN) |
| **삽입** | O(logN) | O(logN) |
| **삭제** | O(logN) | O(logN) |
| **KD 트리** | **탐색** | O(logN) | O(N) |
| **삽입** | O(logN) | O(N) |
| **삭제** | O(logN) | O(N) |
| **레드 블랙 트리** | **탐색** | O(logN) | O(logN) |
| **삽입** | O(logN) | O(logN) |
| **삭제** | O(logN) | O(logN) |
| **스플레이 트리** | **탐색** | O(logN) | O(logN) |
| **삽입** | O(logN) | O(logN) |
| **삭제** | O(logN) | O(logN) |
| **B- 트리** | **탐색** | O(logN) | O(logN) |
| **삽입** | O(logN) | O(logN) |
| **삭제** | O(logN) | O(logN) |
| **B+ 트리** | **탐색** | O(logN) | O(log n + log L) |
| **삽입** | O(logN) | O(M\*log n + log L**)** |
| **삭제** | O(logN) | O(M\*log n + log L) |

# **4. 결론**

a) 이진 탐색 트리의 성능을 개선하고자 여러가지 다양한 트리가 개발되었음

b) 현재까지 개발된 트리 중 레드 블랙 트리가 가장 좋은 성능을 내는 것으로 알려짐

c) 스플레이 트리의 경우 레드 블랙 트리와 AVL 트리보다 간단하면서도 레드 블랙 트리에 근접하는 우수한 성능을 내는 트리로 실용적임

d) B-와 B+ 트리의 경우 데이터베이스에서 우수한 성능을 냄