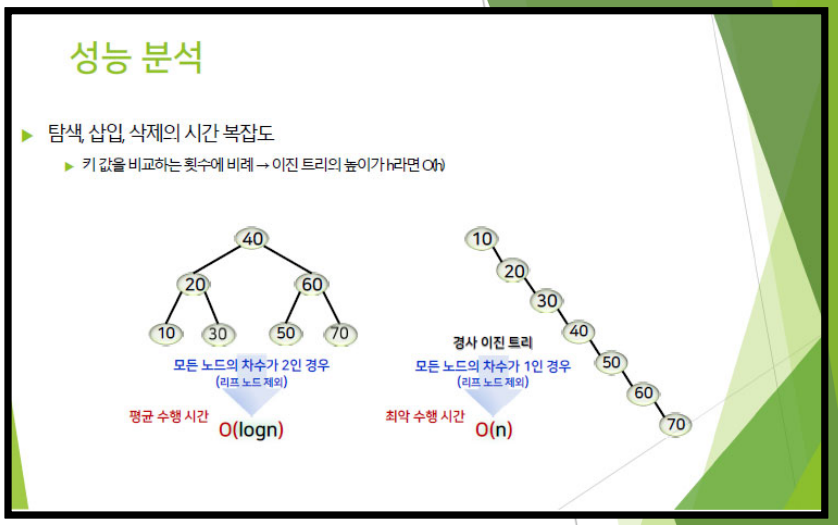
탐색 알고리즘_2

2020년도 2학기 최 희석

목차

- 흑적 트리
- B-트리
- 해싱



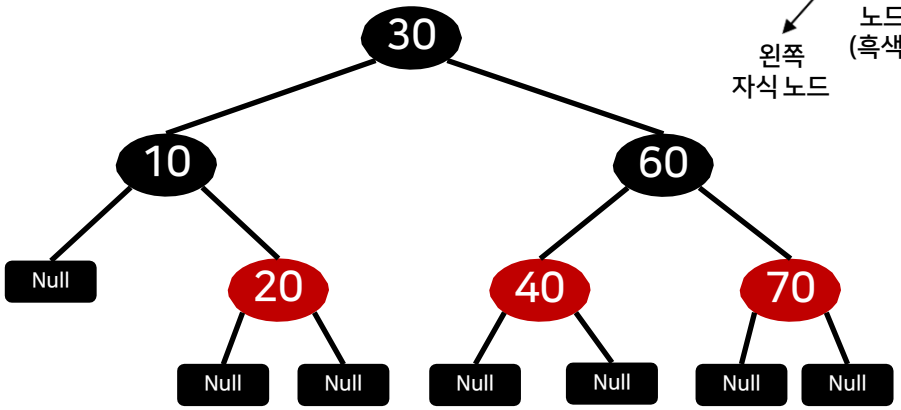
흑적 트리

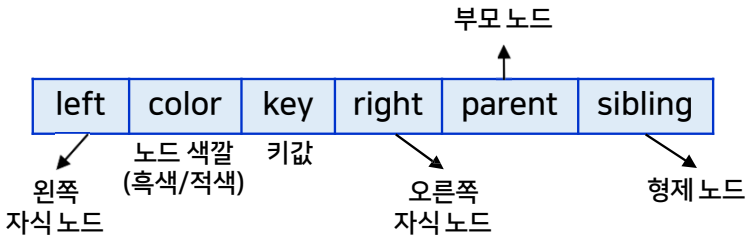
흑적 트리의 개념과 원리

- 흑적 트리 Red-Black Tree
  - 이진 탐색 트리, 균형 탐색 트리

- (성질1) 모든 노드는 흑색이거나 적색이다.
- (성질2) 루트 노드와 리프 노드는 흑색이다.
  - 모든 리프 노드는 Null 노드이다.
- (성질3) 적색 노드의 부모 노드는 항상 흑색이다.
  - 적색 노드가 연달아 나타날 수 없다.
- (성질4) 임의의 노드로부터 리프 노드까지의 경로 상에는 동일한 개수의 흑색 노드가 존재한다.

흑적 트리의 개념과 원리

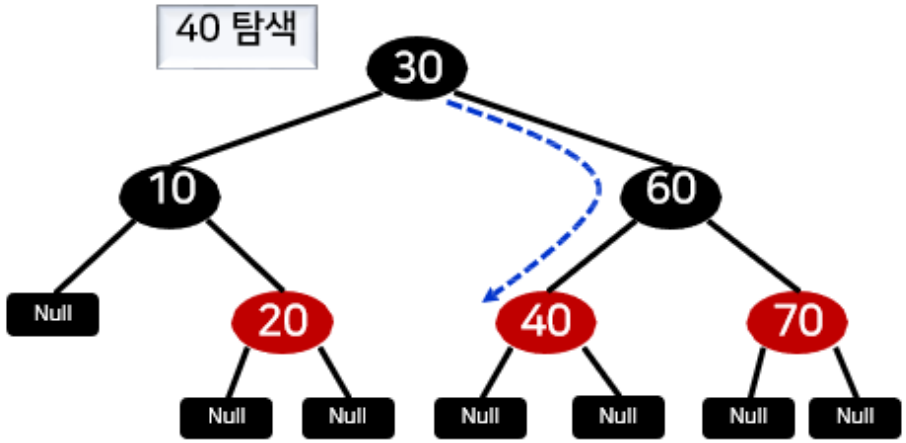




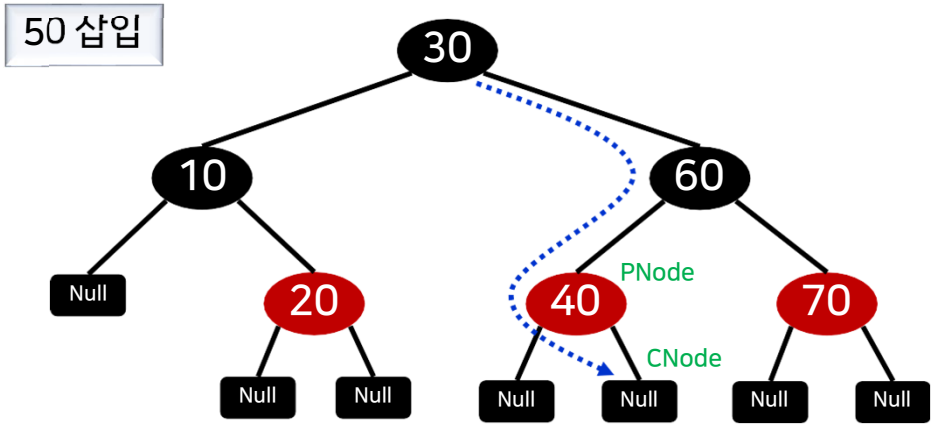
- (성질1) 모든 노드는 흑색이거나 적색이다.
- (성질2) 루트 노드와 리프 노드는 흑색이다.
  - 모든 리프 노드는 Null 노드이다.
- (성질3) 적색 노드의 부모 노드는 항상 흑색이다.
  - 적색 노드가 연달아 나타날 수 없다.
- (성질4) 임의의 노드로부터 리프 노드까지의 경로 상에는 동일한 개수의 흑색 노드가 존재한다.

탐색 연산

- 이진 탐색 트리에서의 탐색 방법과 동일

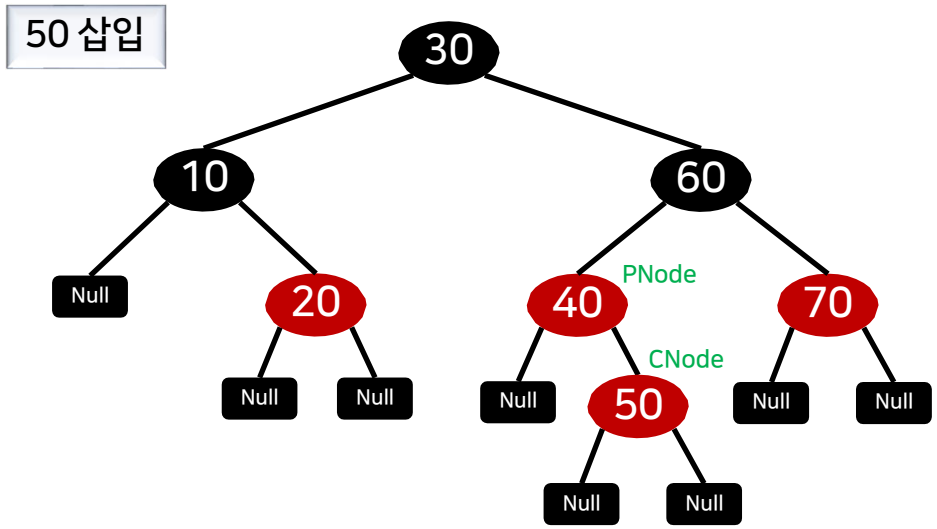


삽입 연산



탐색이 실패한 Null 노드에 적색 노드를 추가, 두 자식 노드를 Null 노드로 만듦

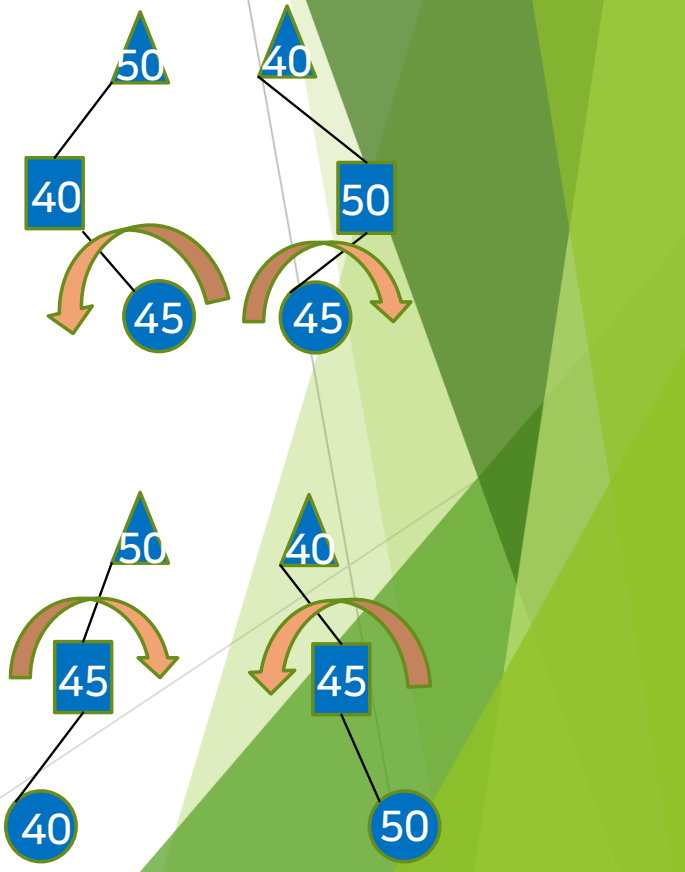
삽입 연산



적색 노드가 연달아 나타나면 흑적 트리의 성질을 만족하도록
루트 노드쪽으로 올라가면서 노드의 구조와 색깔을 조정

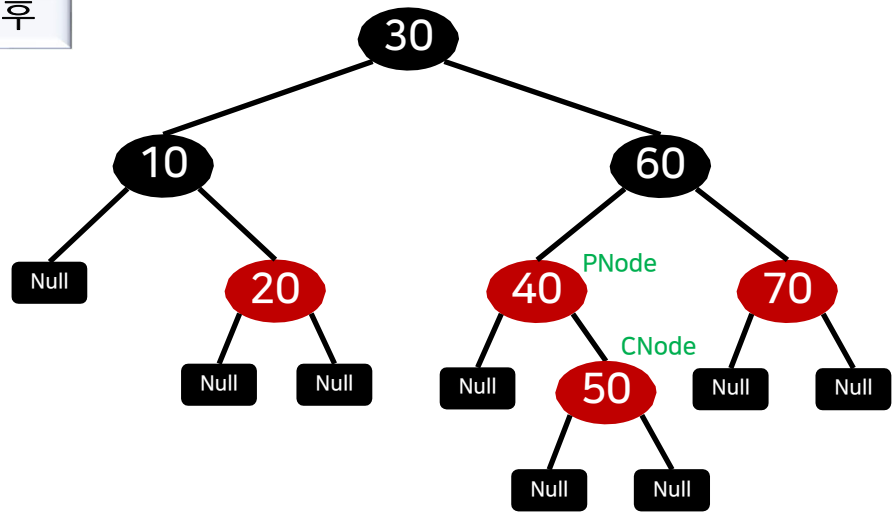
삽입 연산의 적용 규칙

- 적색 노드가 연달아 나타나는 경우에 적용하는 규칙
  - **(규칙 1) 부모 노드의 형제 노드가 적색인 경우**
    - 부모 노드, 부모 노드의 형제 노드, 부모 노드의 부모 노드의 색깔을 모두 변경
  - **(규칙 2) 부모 노드의 형제 노드가 흑색이고, 현재 노드의 키 값이 부모 노드와 부모 노드의 부모 노드의 키 값의 사이인 경우인 경우**
    - 현재 노드와 부모 노드를 회전시킴
  - **(규칙 3) 부모 노드의 형제 노드가 흑색이고, 현재 노드의 키 값보다 부모 노드와 부모 노드의 부모 노드의 키 값이 큰 (또는 작은) 경우**
    - 부모 노드와 부모 노드의 부모 노드를 회전시키고 색깔을 변경



삽입 연산

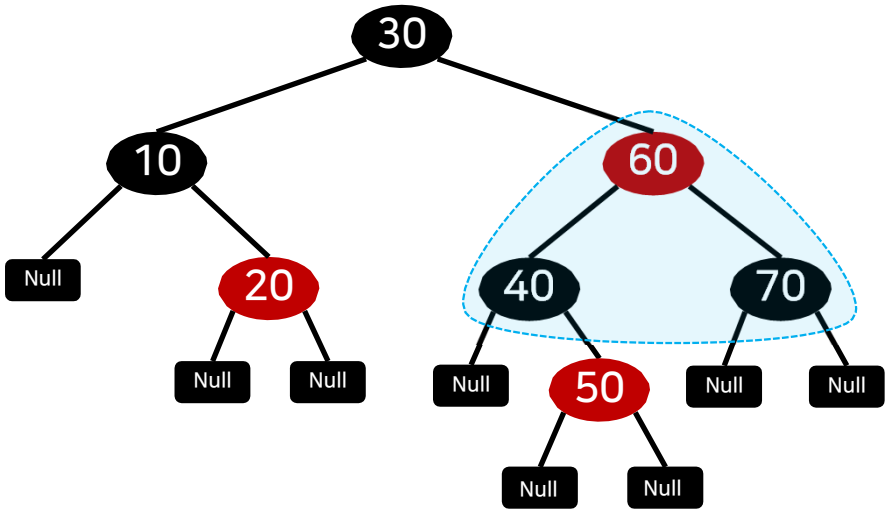
50 삽입후



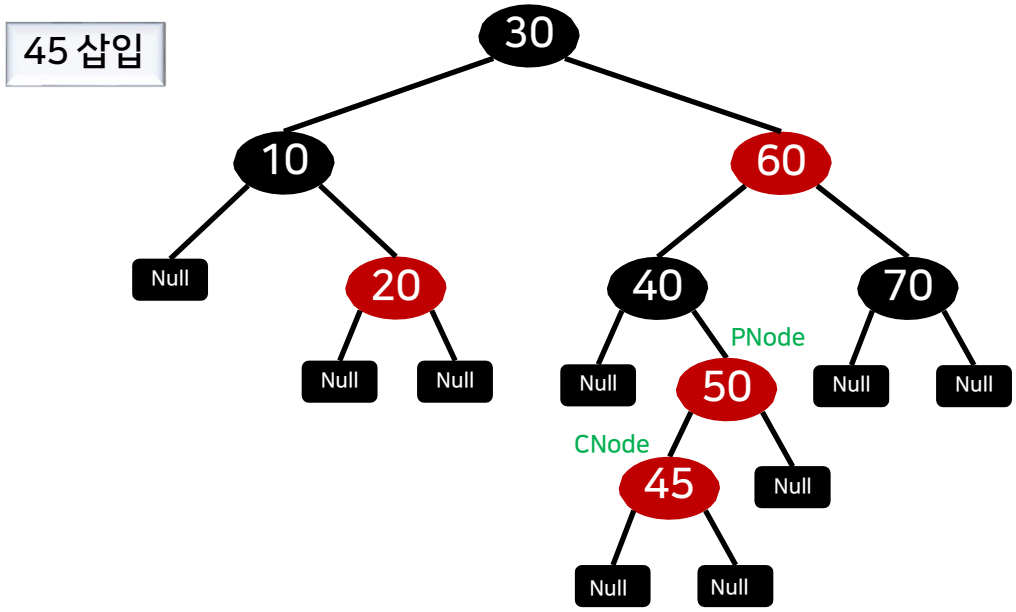
(규칙1) 부모 노드40의 형제 노드70가 적색인 경우

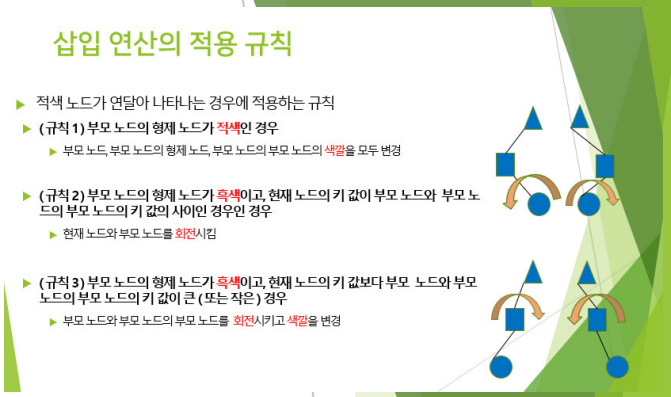
→ 부모 노드40, 부모 노드의 형제 노드70, 부모 노드의 부모 노드60의 색깔 변경

삽입 연산



삽입 연산



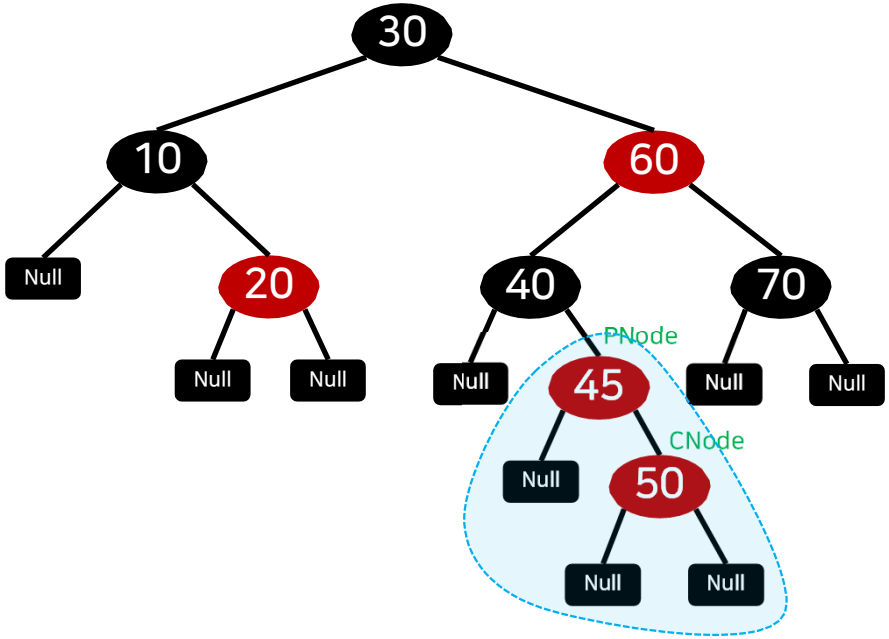


(규칙2) 부모 노드50의 형제 노드Null = 흑색

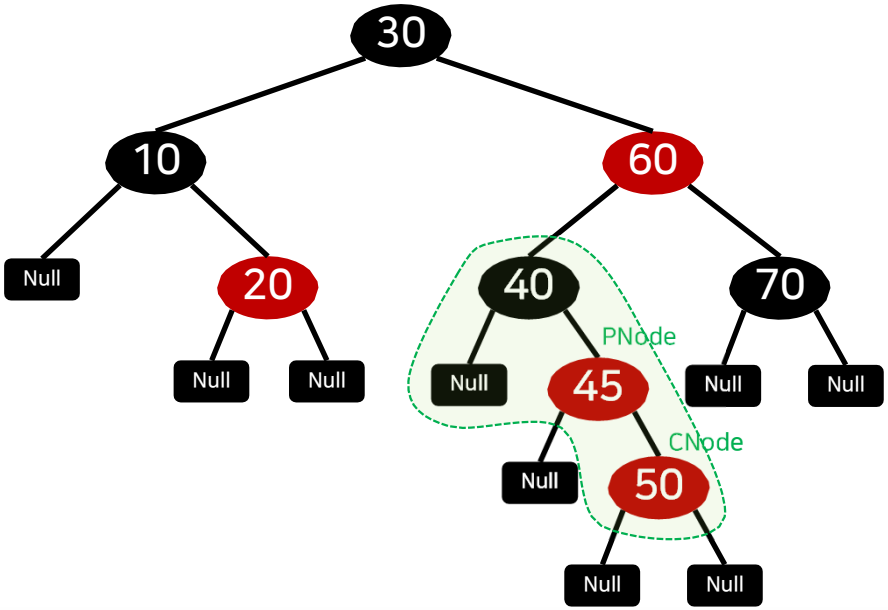
& 현재 노드의 키값45이 부모 노드50와 부모 노드의 부모 노드40의 키값 사이인 경우

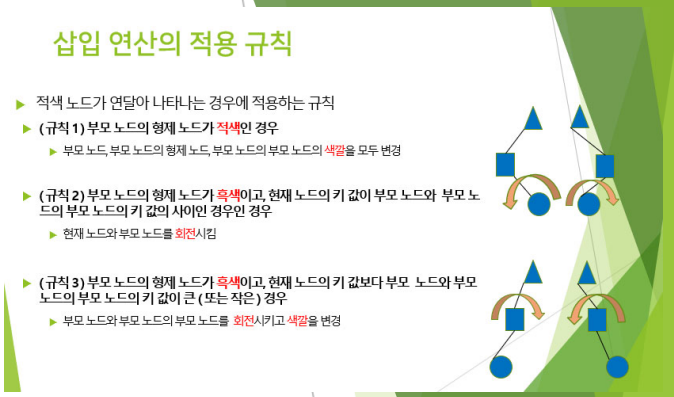
→ 현재 노드45와 부모 노드50를 회전

삽입 연산


30
10
60
Null
20
40
70
PNode
Null
Null
Null
45
Null
Null
CNode
Null
50
Null
Null

삽입 연산

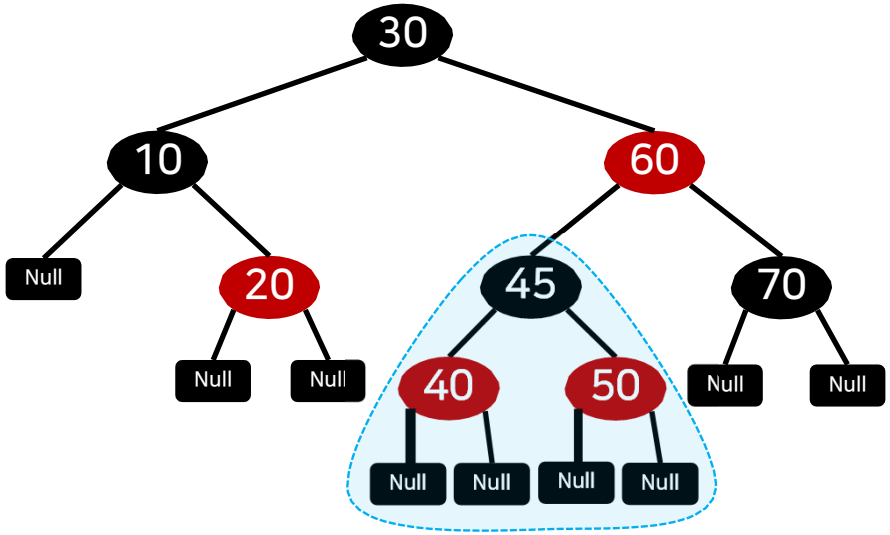




⇒ 부모 노드45와
부모 노드의 부모 노드40를
회전시키고 색깔 변경

(규칙3) 부모 노드45의 형제 노드Null = 흑색
& 현재 노드의 키값50보다 부모 노드45와 부모 노드의 부모 노드40의 키값이 작은 경우

삽입 연산

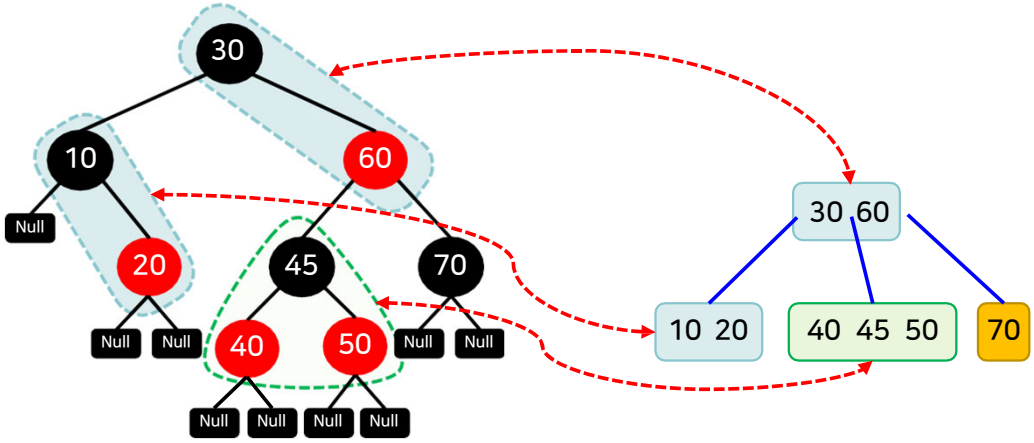

30
10
60
Null
20
45
70
Null
Null
40
50
Null
Null
Null
Null
Null
Null

상입 연산의 적용 규칙

- 탐색 시간 → **O(log n)**
  - 모든 리프 노드의 깊이가 두 배 이상 차이 나지 않으므로 최악의 트리의 높이는 O(logn)
- 삽입/삭제 시간 → **O(log n)**

흑적 트리의 특징

- 사실상 이진 탐색 트리
  - 탐색 연산은 이진 탐색 트리와 동일
  - 삽입 연산은 회전, 색깔 변경과 같은 추가 연산이 필요
- 2-3-4 트리를 이진 탐색 트리로 표현한 것

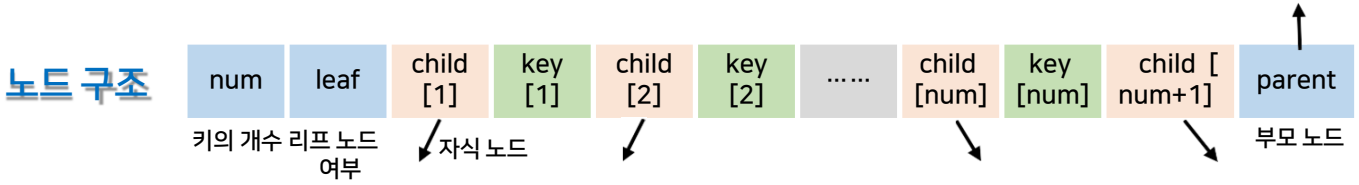


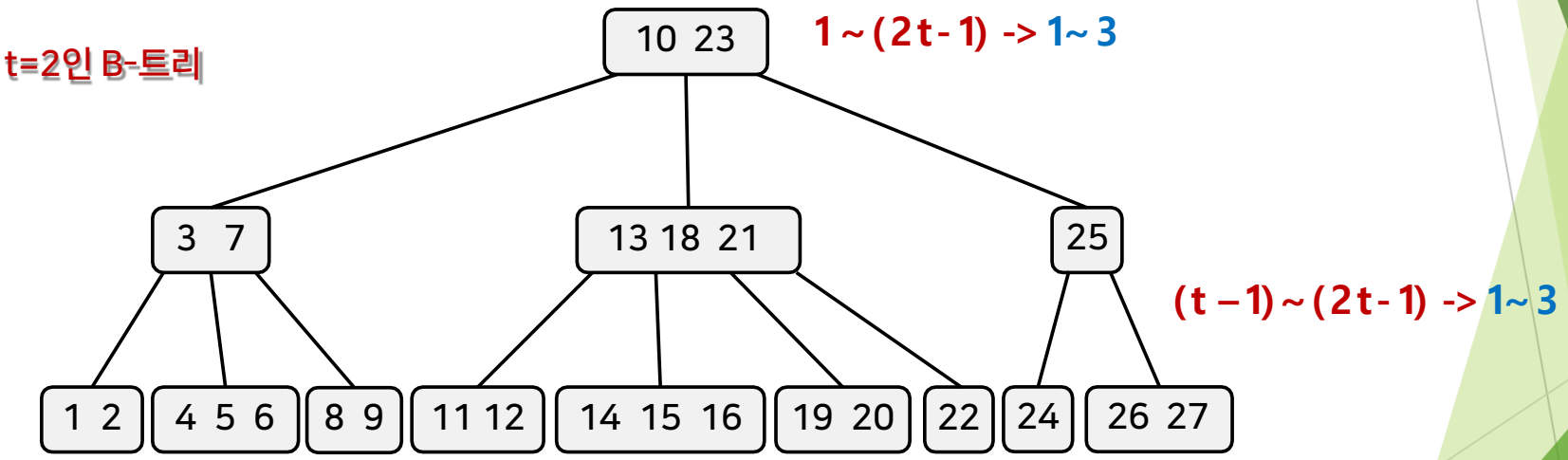
B-트리

B-트리의 개념과 원리

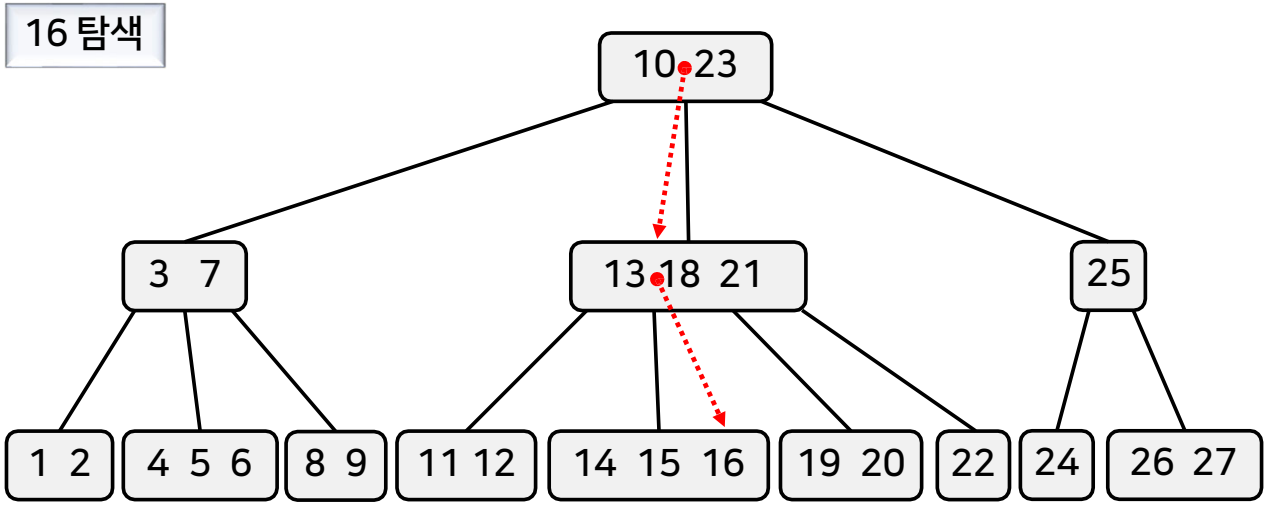
- 균형 탐색 트리
  - **(성질 1)** 루트 노드는 한 개 이상 2t 개 미만의 오름차순으로 정렬된 키를 가짐 **1~(2t-1)**
  - **(성질 2)** 루트 노드가 아닌 모든 노드는 (t−1) 개 이상 2t 개 미만의 오름차순으로 정렬된 키를 가짐 **(t−1)~(2t-1)**
  - **(성질 3)** 내부 노드는 자신이 가진 키의 개수보다 하나 더 많은 자식 노드를 가짐
  - **(성질 4)** 한 노드의 한 키의 왼쪽 서브트리에 있는 모든 키 값은 그 키 값보다 작고, 오른쪽 서브트리에 있는 모든 키 값은 그 키 값보다 크다.
  - **(성질 5)** 모든 리프 노드의 레벨은 동일

B-트리의 개념과 원리


노드 구조
num
leaf
child [1]
key [1]
child [2]
key [2]
……
child [num]
key [num]
child [num+1]
parent
키의 개수
리프 노드 여부
자식 노드
부모 노드


t=2인 B-트리
10 23
1 ~ (2t - 1) -> 1~ 3
3 7
13 18 21
25
(t – 1) ~ (2t - 1) -> 1~ 3
1 2
4 5 6
8 9
11 12
14 15 16
19 20
22
24
26 27

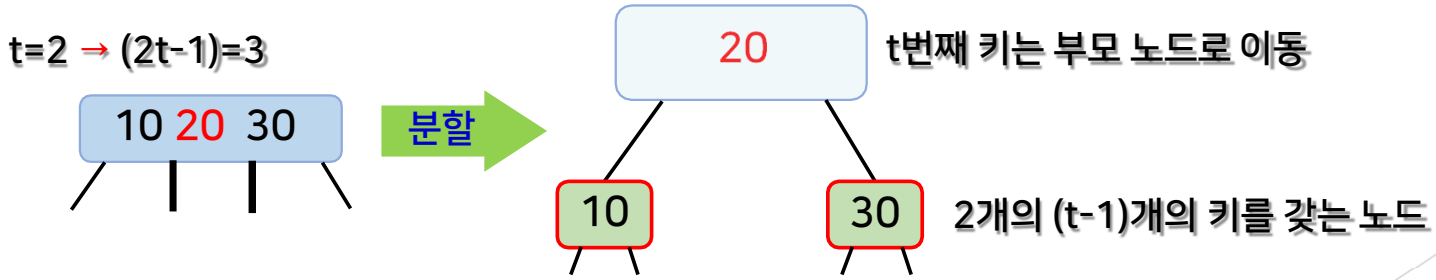
탐색 연산



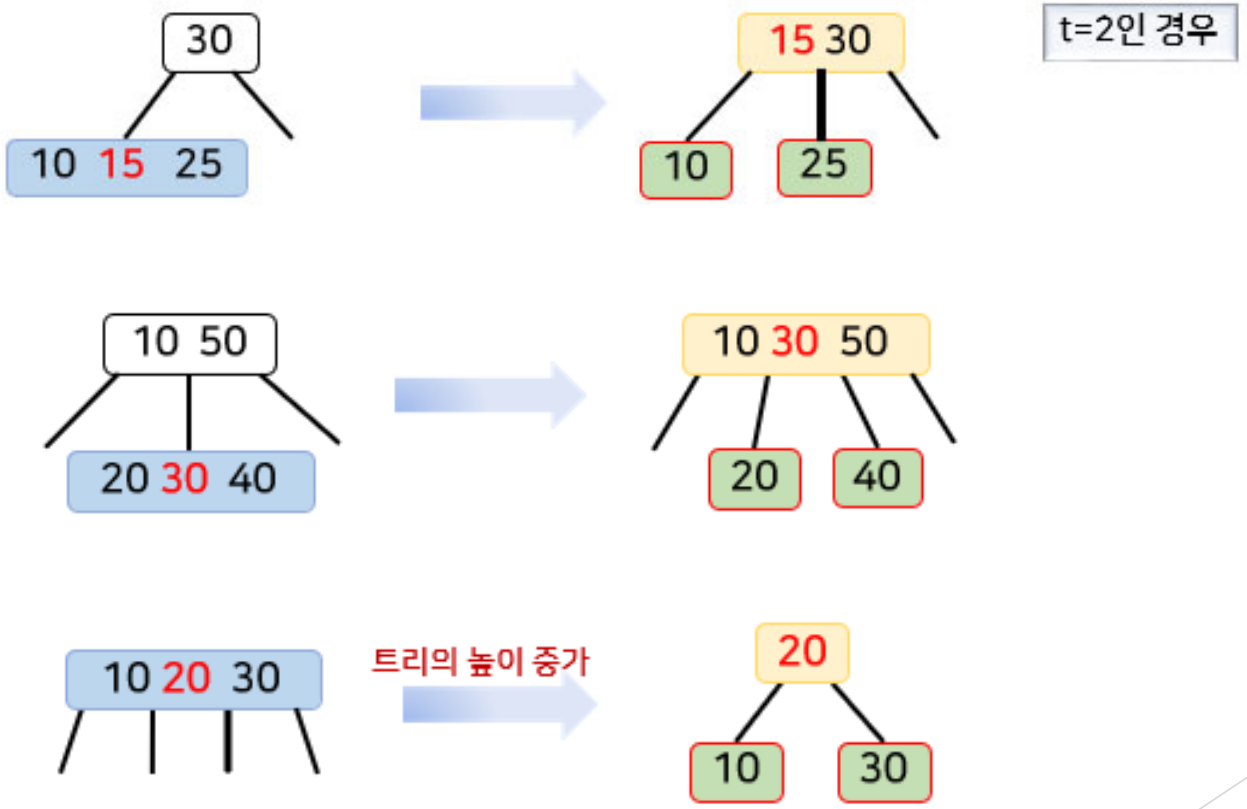
삽입 연산

- 루트 노드에서부터 탐색을 수행하여 리프 노드에도 존재하지 않으면 해당 노드에 추가

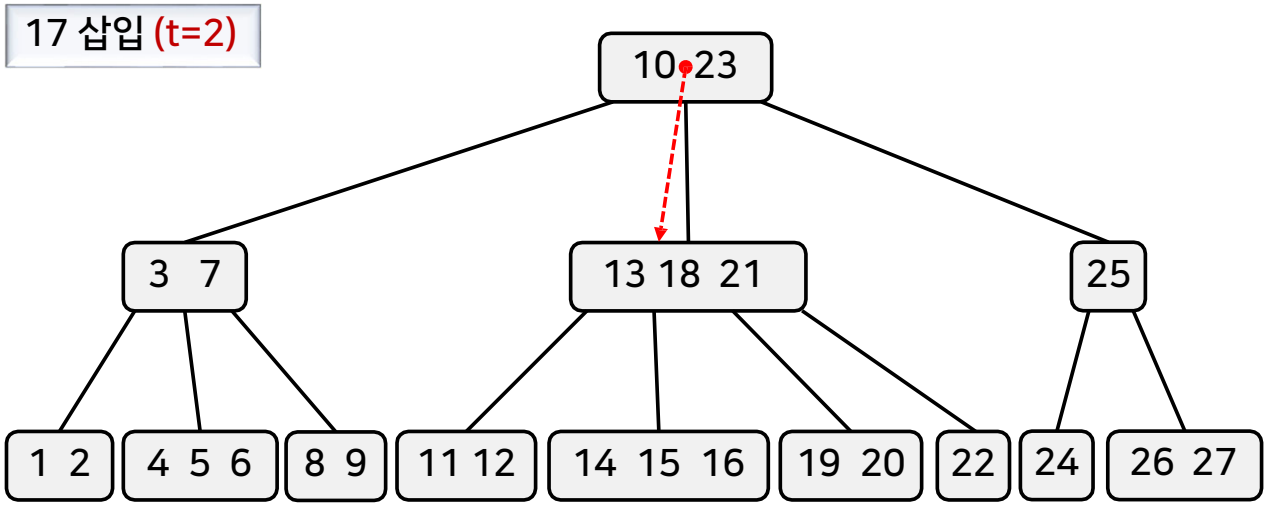
- 노드 분할
  - 삽입을 위한 탐색 과정에서 (2t-1)개의 키를 갖는 노드를 만나면, 이 노드를 (t-1)개의 키를 갖는 두 개의 노드로 분할
    - 삽입으로 인해 노드의 키의 개수가 2t개가 되는 것을 방지



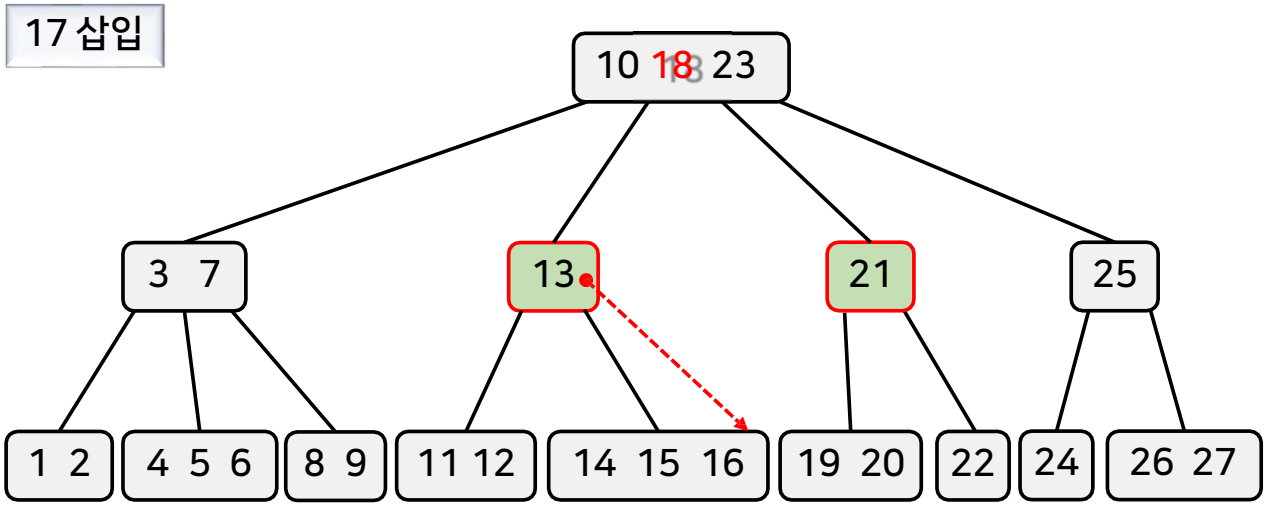
삽입 연산_노드 분할의 유형



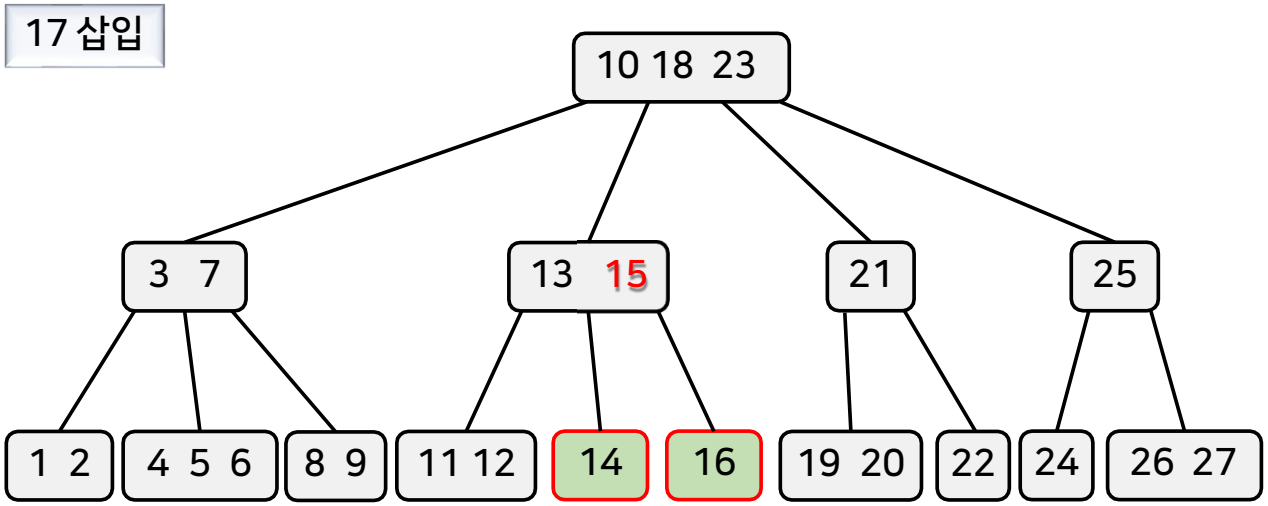
삽입 연산



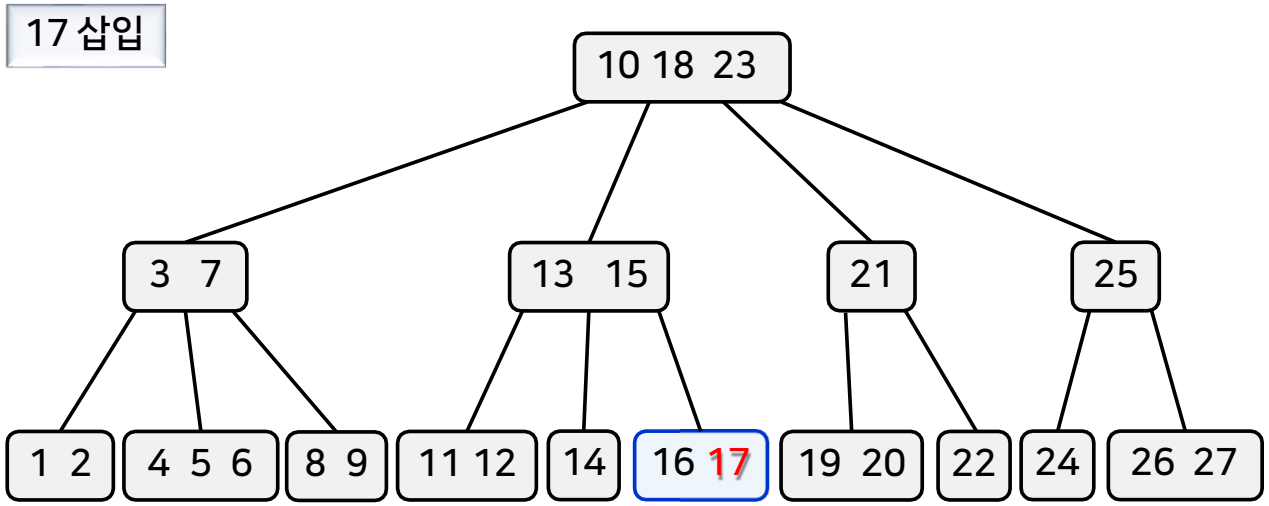
삽입 연산



삽입 연산



삽입 연산



B-트리의 성능 분석

- 탐색, 삽입, 삭제 시간 복잡도
  - 트리의 높이 h, 각 노드에서 키의 위치를 찾는 시간 O(t) → O(th)
    - 각 노드에서는 (t-1) ~ (2t-1)개의 키와 t ~ 2t개의 자식 노드를 가짐
    - 모든 리프 노드의 레벨은 동일
  - 트리의 높이 h → $O(\log_t n)$ (n : 키의 개수)
  - 각 노드에서의 키 관리에 흑적 트리를 이용하면 O(t) → O(log t)
- $O(\log t \log_t n)$ → O(log n)

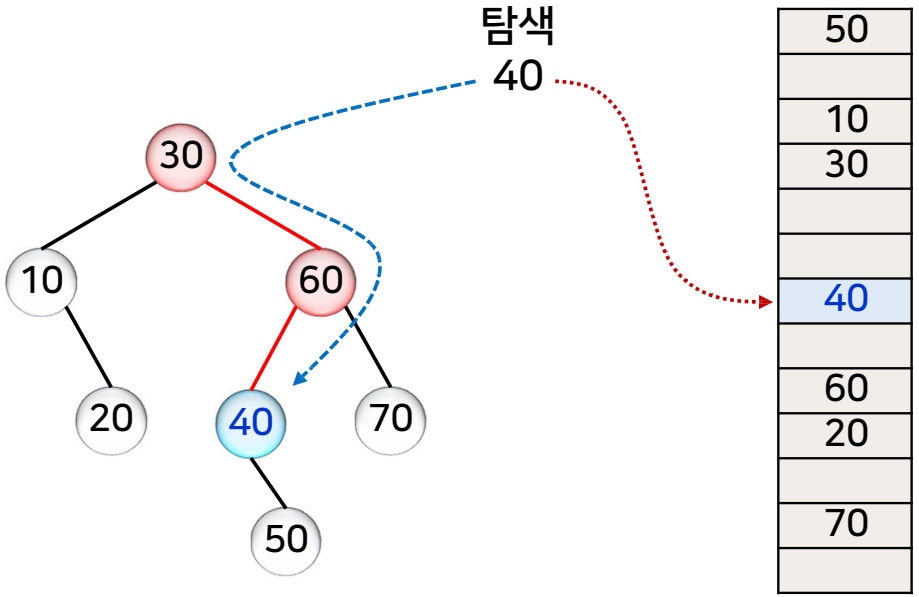
B-트리의 특징

- 내부 탐색과 외부 탐색에 모두 활용
  - 내부 탐색의 경우, t=2 또는 3으로 지정
    - t=2이면 2-3-4 트리
  - 외부 탐색의 경우, t를 충분히 크게 지정
    - 한 노드의 크기가 디스크 한 블록에 저장되도록

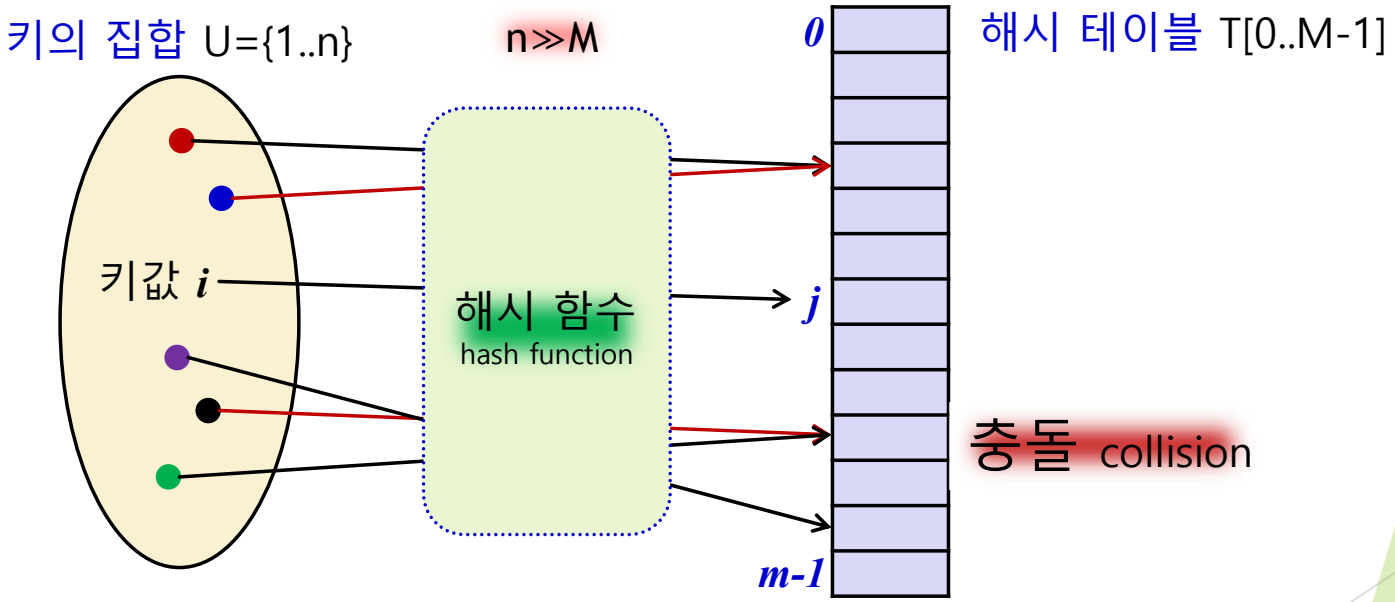
해싱

해싱의 개념

- 탐색 키 값을 기반으로 데이터의 저장 위치를 직접 계산
  - 상수 시간 내에 데이터를 탐색, 삽입, 삭제 가능



해싱의 개념


키의 집합 U={1..n}
n≫M
0
해시 테이블 T[0..M-1]
키값 i
해시 함수
hash function
j
충돌 collision
m-1

해싱의 개념

- 해싱이 적합한 형태의 응용 문제는?
  - 동일한 키 값을 가진 여러 개의 데이터가 존재하는 응용
  - 어떤 범위에 속하는 키 값을 가진 모든 데이터를 탐색하는 문제
  - 최대/최소의 키 값을 가진 데이터를 찾는 문제
  - 키 값의 순서대로 데이터를 방문하는 형태의 문제
  - 특정 키 값을 갖는 데이터를 찾는 문제

해시 함수

- 해시 함수 h : U → { 0, 1, .., M-1 }
  - 키 값을 해시 테이블 주소로 변환하는 함수
  - 종류 → 제산 잔여법 mod function, 비닝 binning, 중간 제곱법 mid-square, 문자열을 위한 함수 등

- 바람직한 해시 함수는
  - 계산이 용이해야 한다.
  - 각 키 값을 테이블의 각 슬롯에 균등하게 사상mappiing시킬 수 있어야 한다.

해시 함수_제산 잔여법

- division remainder hashing, modulo division , 가장 간단한 형태

$$h(k) = k \bmod m$$

K : 키 값, m : 저장공간의 크기

9 ← 108 mod 11

- M의 선택에 주의
  - M= 2^r 이면 h (K)는 하위 r 비트의 값이 됨
  - 키 값의 전체 비트를 주소 계산에 활용하지 못함

```
h (int x) {
  return x % 16;
}
```

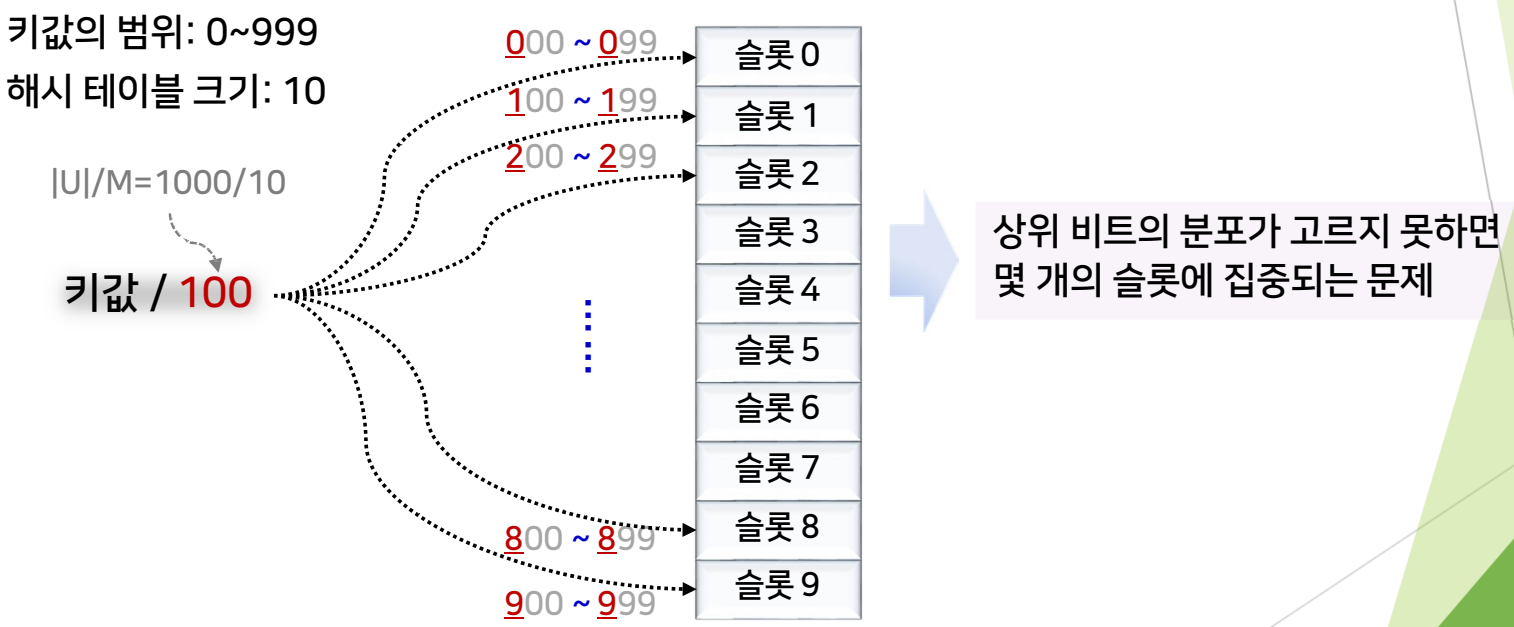
0~ 15 (0000 ~ 1111)

- 39 = 100111 ⇒ 7
- 3751 = 111010100111 ⇒ 7

→ M은 2의 멱수와 상당한 차이가 있는 소수로 선택

해시 함수_비닝

- 키의 집합 U 를 단순히 M 등분하여 각 등분을 각 슬롯으로 해시



해시 함수_중간 제곱법

- Midsqure method, 키 값을 제곱한 결과에서 중간 부분의 적당한 크기의 자릿수를 취하는 방법

```
h (int x) {
  return (x*x /1000) % 100;
}
```

키 값: 4자리 십진수, 해시 테이블 크기: 100

1. 주어진 키값을 제곱한다.

$(4567)^2$ ➡ 20,857,489

2. 제곱된 결과를 키값의 자릿수로 나눈 후, M에 해당하는 하위 2자리 십진수를 취한다.

20,857,489 / 1,000 = 20,857.489 ➡ 20,857.489 %100 ➡ 57

➡ 모든/대부분의 비트가 결과 생성에 기여
→ 상위/하위 자리의 분포에 의해 지배적인 영향을 받지 않음

해시 함수_문자열을 위한 해시 함수(1)

```
h1 (ch x[], int M) { // x: 입력 문자열, M: 테이블 크기
  int xlength = length(x);
  for (sum=0, i=0; i < xlength ; i++)
      sum += x[i];
  return sum % M;
}
```

a:97, b:98 → 97×4+98×4 = 780

780 % 100 = 80

키값 "aaaabbbb"의 주소는?

- M ≪ sum일 때 유용
- 짧은 문자열에 대해서는 비효과적
- 문자열에서 문자의 출현 순서에 무관

해시 함수_문자열을 위한 해시 함수(2)

```
h2 ( ch s[], int M) {
  int intLength = length(s) / 4;
  long sum=0;
  for (j=0; j < intLength; j++) {
    long mult = 1;
    for (k=0; k < 4; k++) {  su
      m += x[j*4+k] * mult;
      mult *= 256;
    }
  }
  char c[] = substring( s[intLength*4 .. length(s)-1] );
  long mult = 1;
  for (k=0; k < length(c); k++) {
      sum += c[k] * mult;
      mult *= 256;
  }
  return ( sum % M );
}
```

키값 "aaaabbbb"의 주소는?

"aaaa" → $\sum_{i=0}^{3}$("a"×256^i) = 1,633,771,873

"bbbb" → $\sum_{i=0}^{3}$("b"×256^i) = 1,650,614,882

3,284,386,755 % 100 = 55

7~12문자 이상의 긴 문자열에 대해 효과적

충돌 해결 방법

- 충돌?
  - 서로 다른 키값 x, y에 대하여 h(x)=h(y)인 경우

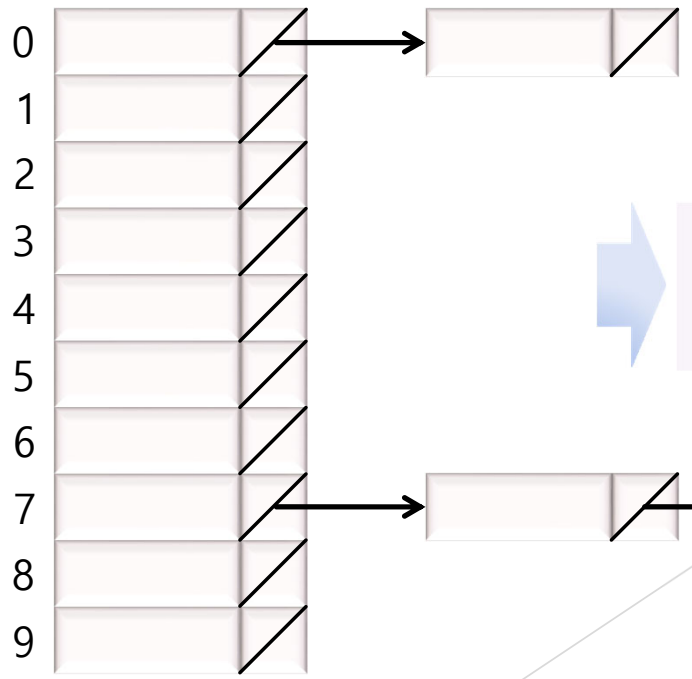
- 충돌 해결 방법
  - 개방 해싱 (연쇄법)
    - 충돌된 데이터를 테이블 밖의 별도의 장소에 저장·관리
  - 폐쇄 해싱 (개방 주소법)
    - 테이블 내의 다른 슬롯에 충돌된 데이터를 저장·관리
    - 버킷 해싱, 선형 탐사, 이차 탐사, 이중 해싱

충돌 해결 방법_연쇄법

- 테이블의 각 슬롯을 연결 리스트의 헤더로 사용

1000 9879 3013 9877 1057 9530 2007

h(k) = k mod 10

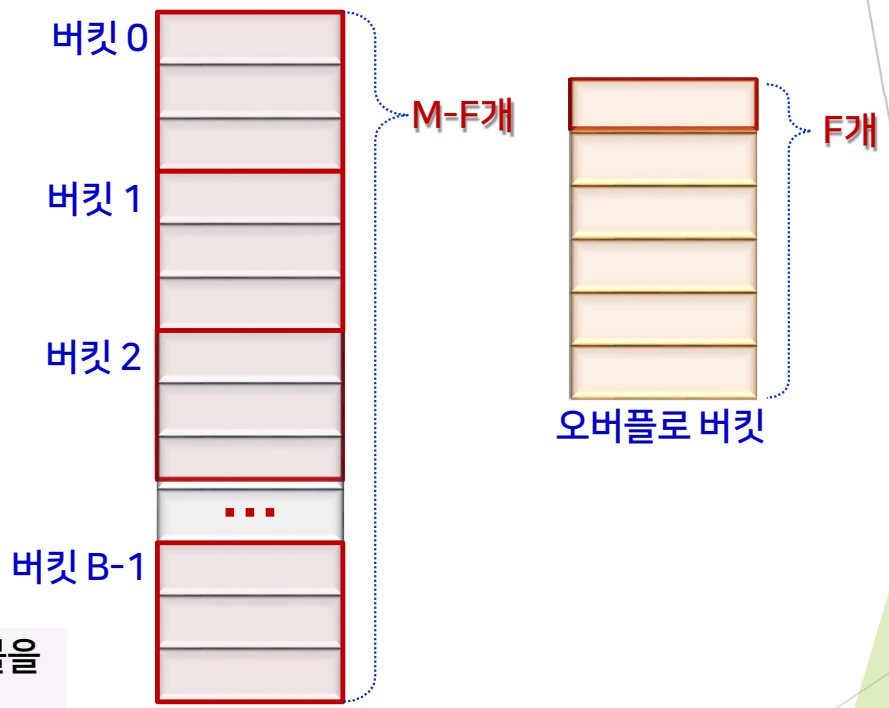


해시 테이블과 연결 리스트가 주기억장치 내에서 유지될 때 적합한 방법

충돌 해결 방법_버킷 해싱

672
1502
352
1122

h(K) = K mod 10



디스크에 저장된 해시 테이블을
구현하는 데 적합

충돌 해결 방법_선형 탐사

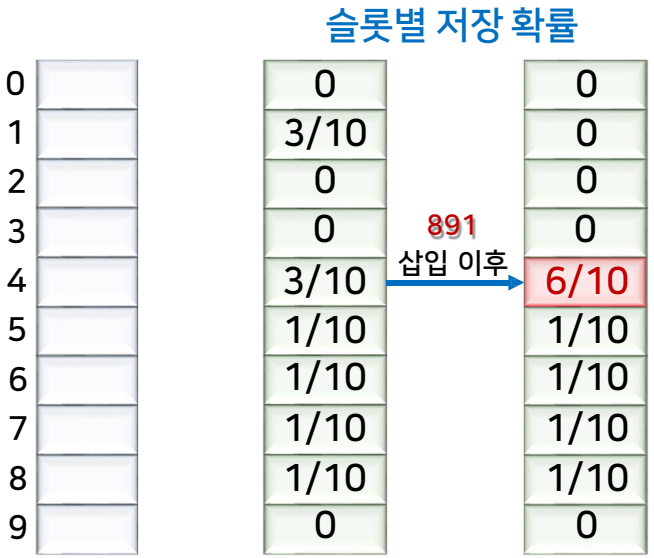
- 탐사 순서 probe sequence
  - 어떤 키 K를 위해서 탐사되는 슬롯의 순서열
    - $p(K,i) \rightarrow p(k,0) = h(K), p(K,1), p(K,2), p(K,3), \ldots$
  - 탐사 순서의 계산 방법에 따라 성능의 차이가 발생
    - 선형 탐사, 이차 탐사, 이중 해싱

- 선형 탐사linear probing
  - $p(K,i) = (h(K) + i) \bmod M, (i = 0, 1, \ldots, M-1)$
    - 빈 슬롯을 찾을 때까지 테이블의 바로 다음 슬롯으로 순차적으로 이동
  - 가장 간단한 방법, 하지만 최악의 방법

충돌 해결 방법_선형 탐사

552
822
700
319
891

h(K)= K mod 10



- 모든 슬롯이 새로운 데이터를 삽입할 후보가 됨
- "1차 클러스터링" 문제 → 긴 탐사 순서를 만들어 평균 탐색 시간의 증가를 초래

  데이터들이 연속된 위치를 점유하여 클러스터를 형성하고, 이것이 점점 커지는 현상

충돌 해결 방법_이차 탐사

- 탐사 순서의 단계에 대한 이차식을 이용
  - $p(K, i) = (h(K) + c_1 \times i^2 + c_2 \times i + c_3) \bmod M$
    - 충돌이 발생하는 횟수(i)의 제곱 형태로 탐사 순서를 결정

- 서로 다른 홈 위치를 갖는 두 키는 서로 다른 탐사 순서를 가짐
  - $p(K, I) = (h(K) + i^2) \bmod 10$
  - $h(k1) = 2 \rightarrow$ 2, 3, 6, 1, …
  - $h(k2) = 5 \rightarrow$ 5, 6, 8, 4, …

충돌 해결 방법_이차 탐사

- 모든 슬롯이 탐사 순서에 사용되지 않음
  - $p(K, I) = (h(K) + i^2) \bmod 10$
    - $p(K,0) = 2$ 라면 슬롯 1, 2, 3, 6, 7, 8 만 탐사 가능
  - 탐사 함수와 테이블 크기가 적절히 조합되면 많은 슬롯의 방문이 가능

- 2차 클러스터링 문제
  - 해시 함수가 특정 홈 위치에 대한 클러스터를 만드는 현상
    - 서로 다른 두 키의 홈 위치가 동일하면 전체 탐사 순서가 동일

충돌 해결 방법_이중 해싱

- 탐사 순서를 원래의 키값을 이용하여 계산
  - 1차/2차 클러스터링 문제 해결
- $p(K, i) = (h_1(K) + i \times h_2(K)) \bmod M$

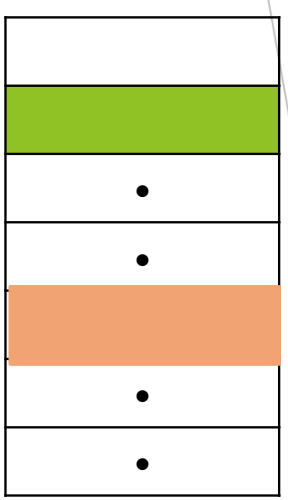
- 서로 다른 두 키의 홈 위치가 동일해도 서로 다른 탐사 순서를 가짐

- 좋은 이중 해싱을 구현하려면
  - 탐사 순서의 모든 상수가 테이블 크기 M과 서로소가 되어야 함
    - M을 소수로 선택하고, h_2가 $1 \le h_2(K) \le M-1$ 의 값을 반환
    - $M = 2^m$ 으로 정하고, h_2 가 1과 2^m 사이의 홀수를 반환

삭제 연산

- 두 가지 고려 사항
  - 데이터의 삭제가 차후의 **탐색을 방해**하지 않아야 한다.
    - 단순히 빈 슬롯을 두면 탐색이 해당 슬롯에서 종료되므로 그 이후의 레코드가 고립된다.
  - 삭제로 인해 해시 테이블의 위치에서 사용할 수 없는 곳을 만들지 않아야 한다.

- 비석 tombstone
  - 삭제된 데이터의 위치에 '비석'이라는 특별한 표시를 하는 방법
    - 탐색 → 비석을 무시하고 탐색을 계속 진행
    - 삽입 → 비석이 표시된 위치를 빈 위치로 간주하여 새 데이터를 삽입
  - 비석의 개수가 증가할수록 평균 탐색 거리가 증가하는 문제



과제 안내

과제

- 흑적트리 구현하기
  - e-Class 업로드
- 양식 (한글, 워드,PDF -> 자유)
- 파일명 (이름_학번_전공)
  - 예) 최희석_2014182009_게임공학

- 질의 응답은 e-Class 질의응답 게시판에 남겨 주시길 바랍니다.