ALGORITHM
알고리즘

Lecture 07

탐색 (2)

컴퓨터과학과 | 이관용 교수

한국방송통신대학교

학습목차

01. 레드-블랙 트리

레드-블랙 트리?

이진 탐색 트리, 균형 탐색 트리

〈성질 1〉 모든 노드는 검정이거나 빨강이다.

〈성질 2〉 루트 노드와 리프 노드는 검정이다.

→ 모든 리프 노드는 NULL 노드이다.

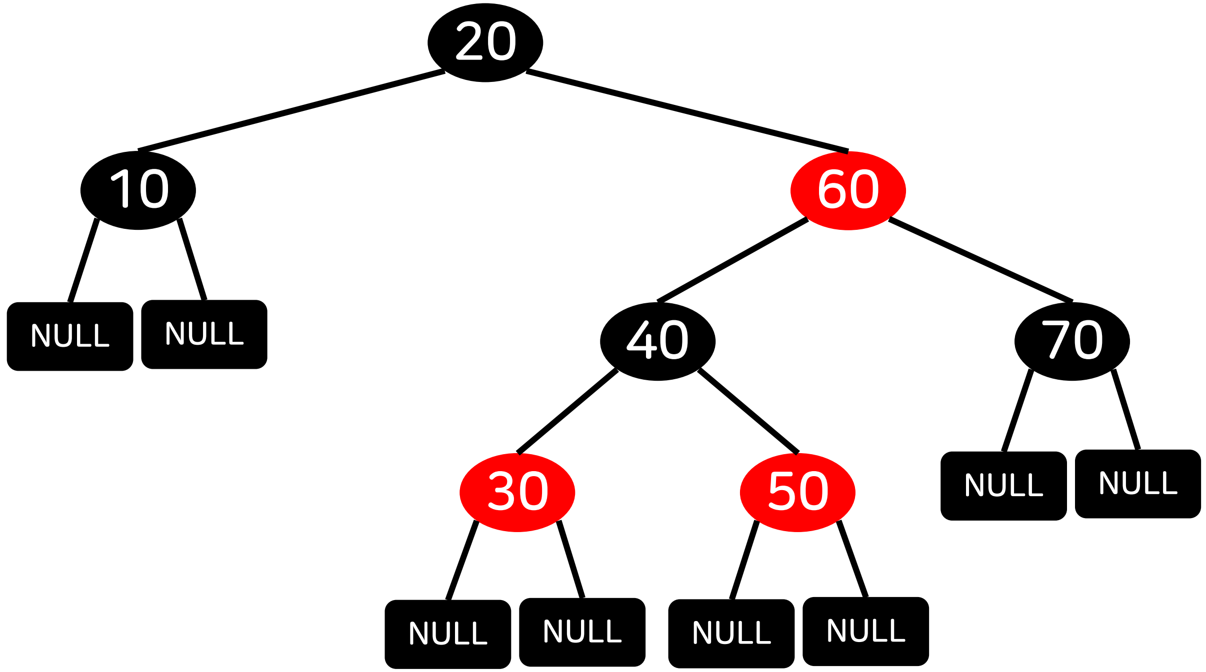
〈성질 3〉 빨강 노드의 부모 노드는 항상 검정이다.

→ 빨강 노드가 연달아 나타날 수 없음

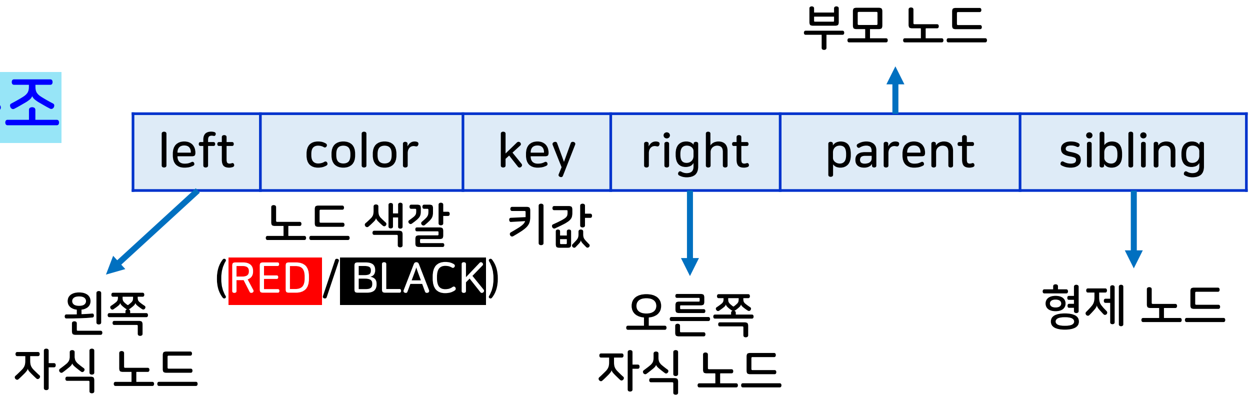
〈성질 4〉 임의의 노드로부터 리프 노드까지의 경로상에는
동일한 개수의 검정 노드가 존재한다.

〈성질 5〉, 〈성질 6〉 → 이진 탐색 트리의 성질

레드-블랙 트리

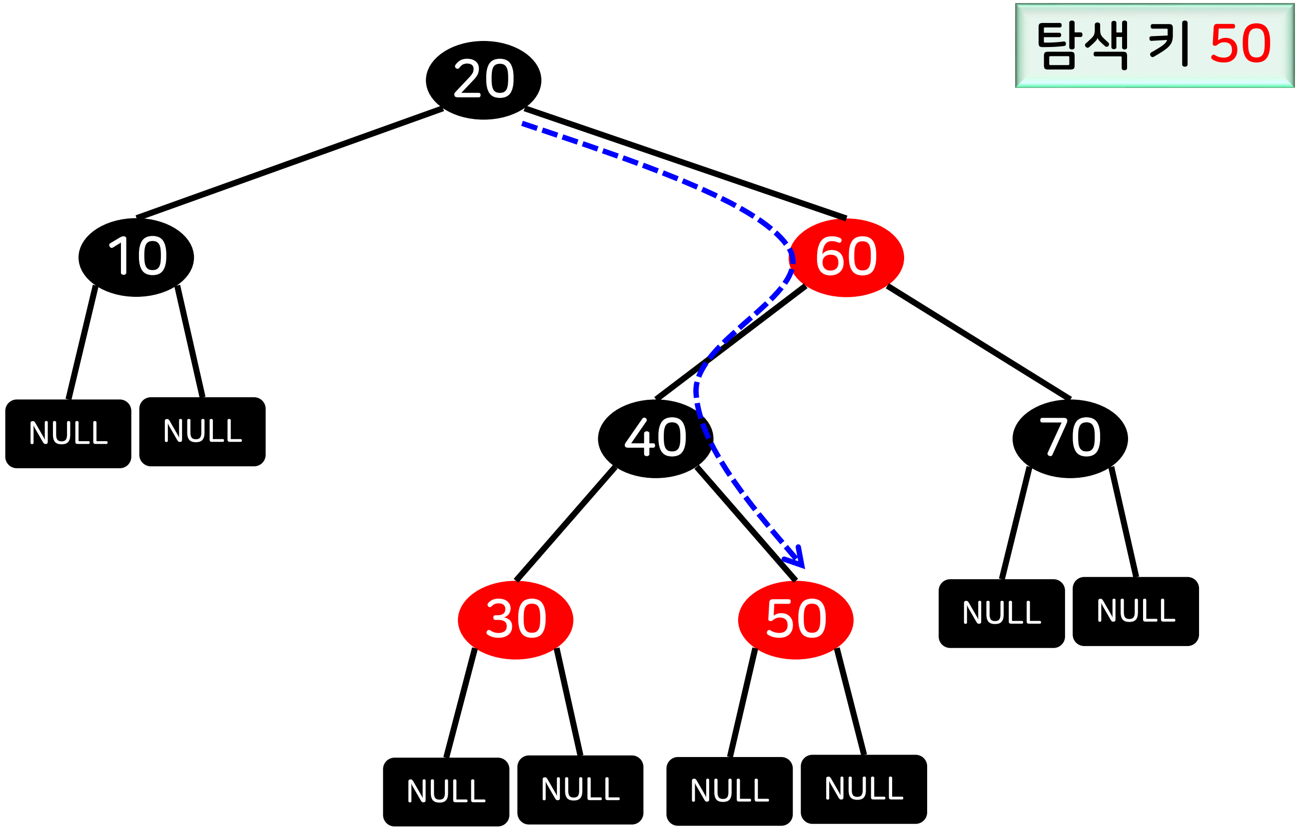

20
10
60
NULL
NULL
40
70
30
50
NULL
NULL
NULL
NULL
NULL
NULL

노드 구조


부모 노드
left
color
key
right
parent
sibling
노드 색깔
(RED/BLACK)
키값
왼쪽
자식 노드
오른쪽
자식 노드
형제 노드

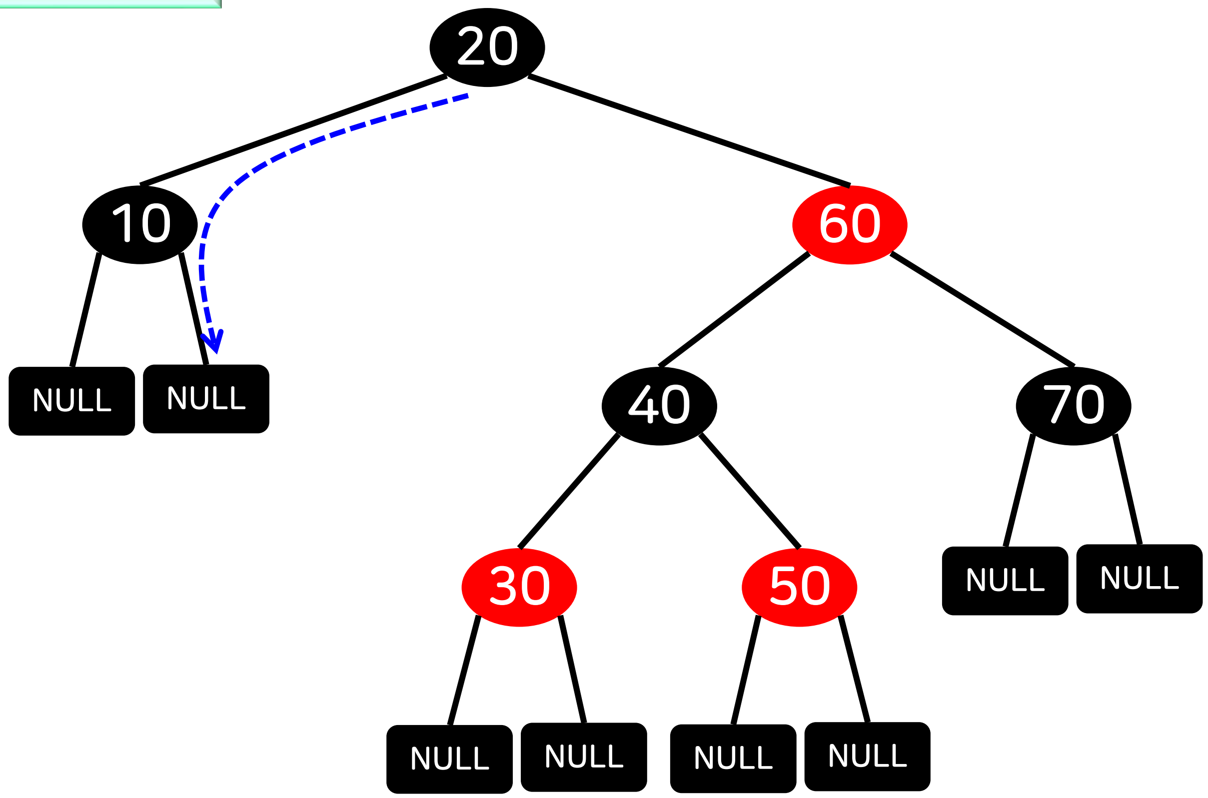
레드-블랙 트리_탐색 연산

이진 탐색 트리의 탐색 방법과 동일



레드-블랙 트리_삽입 연산_1

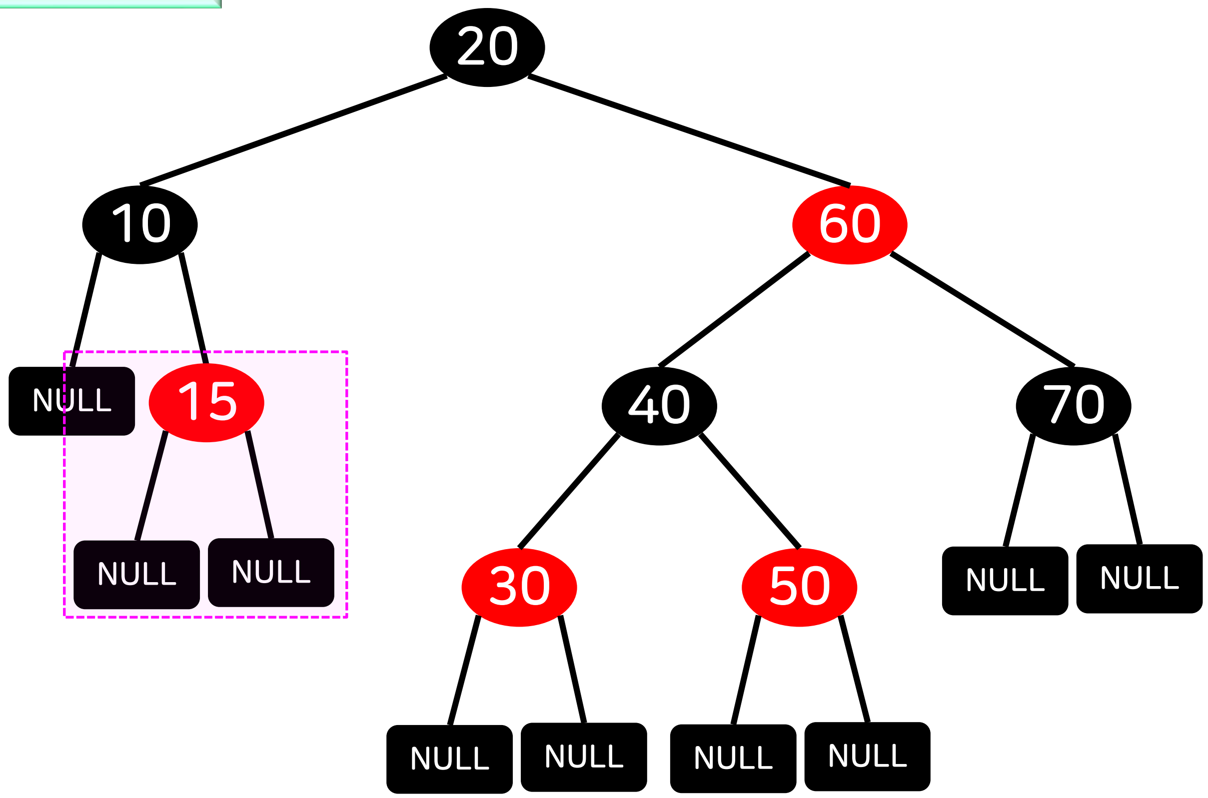
삽입할 원소 15



탐색이 실패한 NULL 노드에 빨강 노드를 추가하고, 두 자식 노드를 NULL 노드로 만듦

레드-블랙 트리_삽입 연산_1

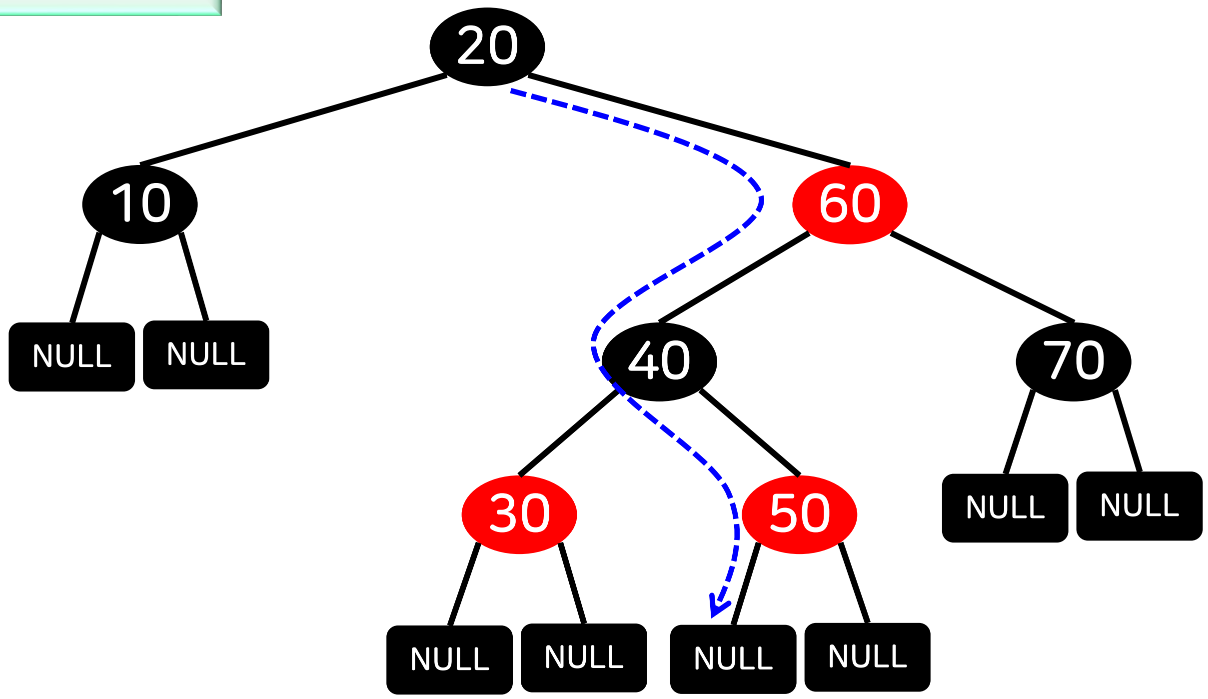
삽입할 원소 15



탐색이 실패한 NULL 노드에 빨강 노드를 추가하고, 두 자식 노드를 NULL 노드로 만듦

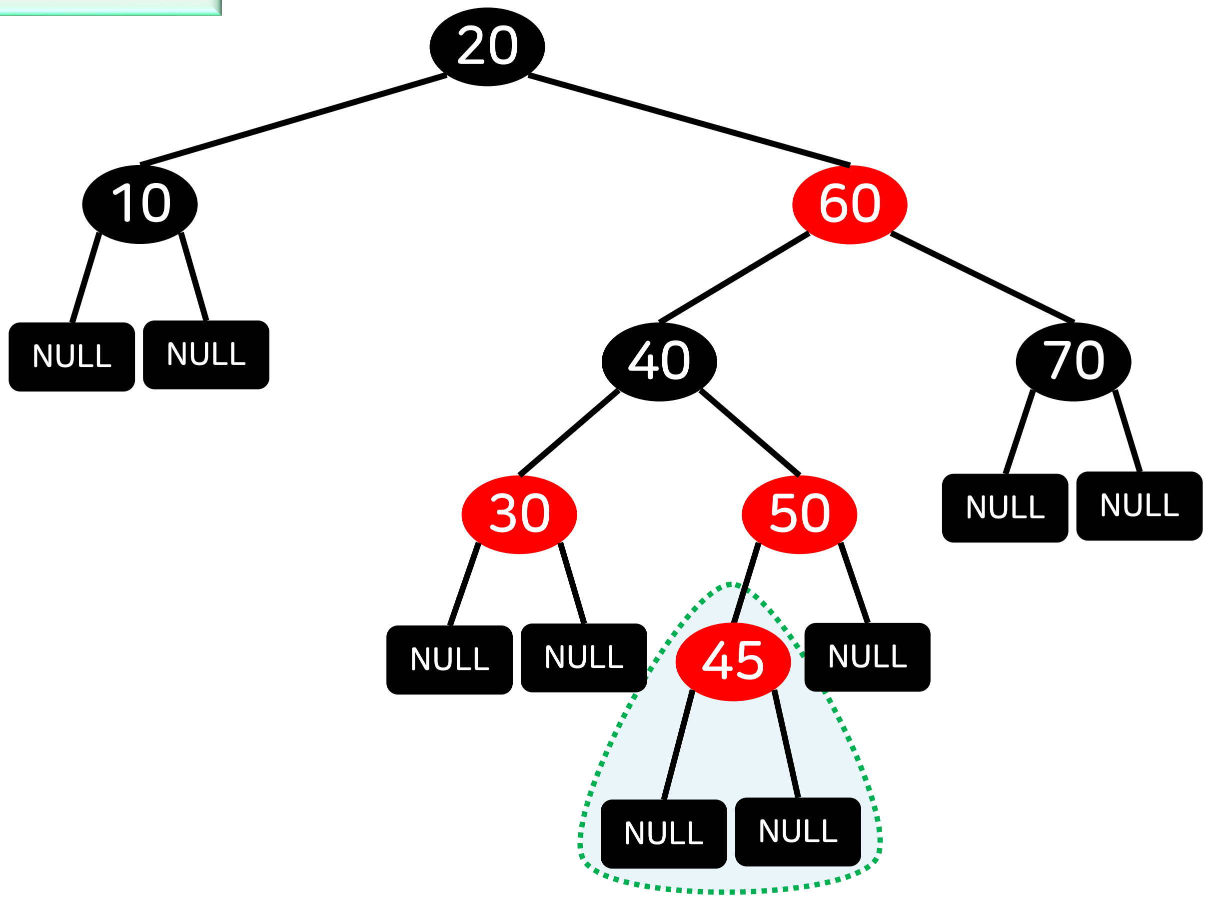
레드-블랙 트리_삽입 연산_2

삽입할 원소 45



레드-블랙 트리_삽입 연산_2

삽입할 원소 45



빨강 노드가 연달아 나타나서 <성질 3>을 만족시키지 못함

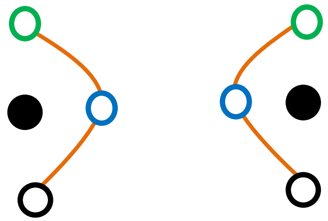
→ 루트 노드쪽으로 올라가면서 노드의 구조와 색깔을 조정해서 성질을 만족시켜야 함

삽입 연산을 위한 규칙

빨강 노드가 연달아 나타나는 경우에 적용하는 규칙

〈규칙 1〉 부모 노드의 형제 노드가 빨강인 경우

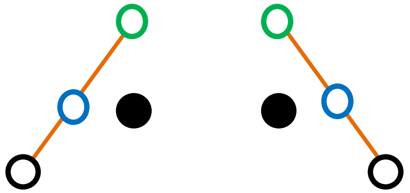
→ 부모 노드, 부모 노드의 형제 노드, 부모 노드의 부모 노드의 색깔을 모두 변경



〈규칙 2〉 부모 노드의 형제 노드가 검정이고,
현재 노드의 키값이 부모 노드와 부모 노드의 부모 노드의 키값의 사이인 경우

→ 현재 노드와 부모 노드를 회전시킴

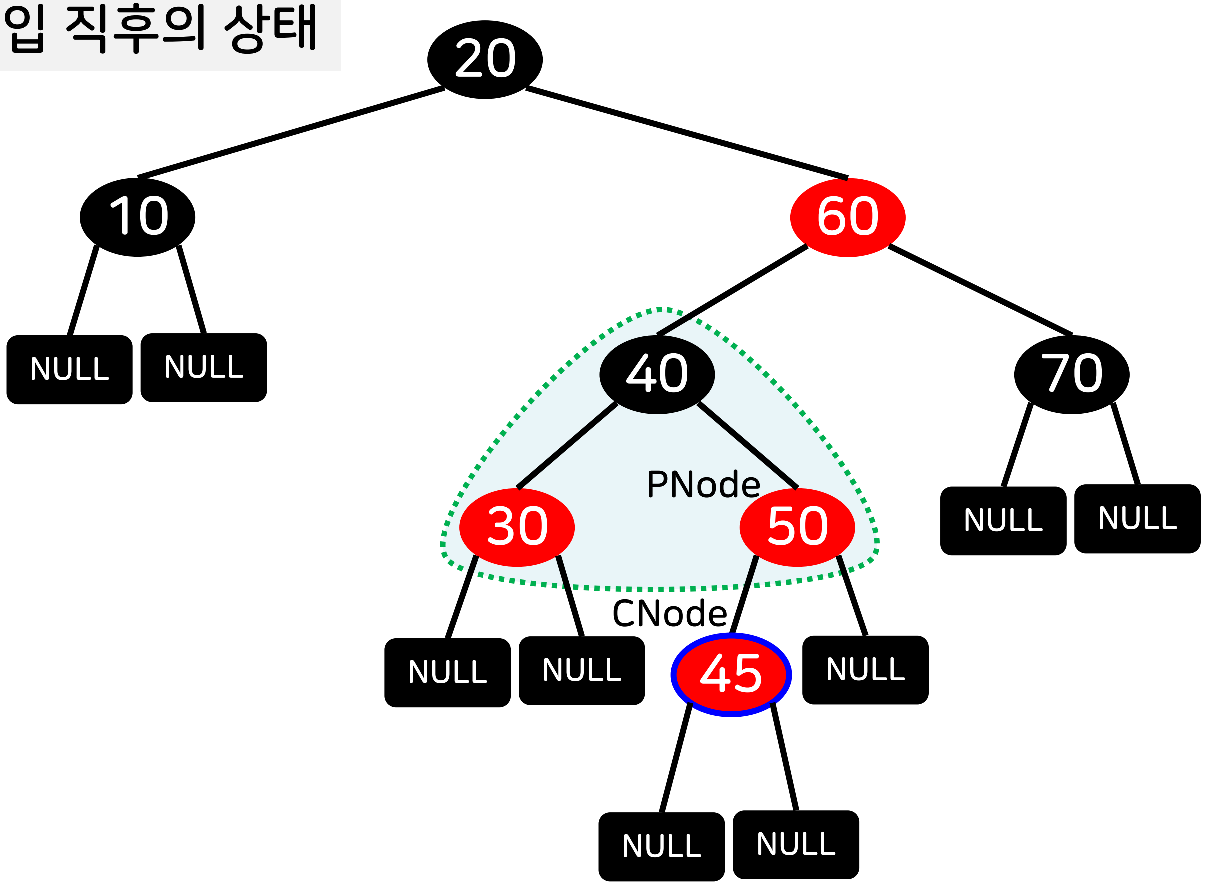
〈규칙 3〉 부모 노드의 형제 노드가 검정이고, 현재 노드의 키값보다
부모 노드와 부모 노드의 부모 노드의 키값이 큰(또는 작은) 경우



→ 부모 노드와 부모 노드의 부모 노드를 회전시키고 색깔을 변경

레드-블랙 트리_삽입 연산_2

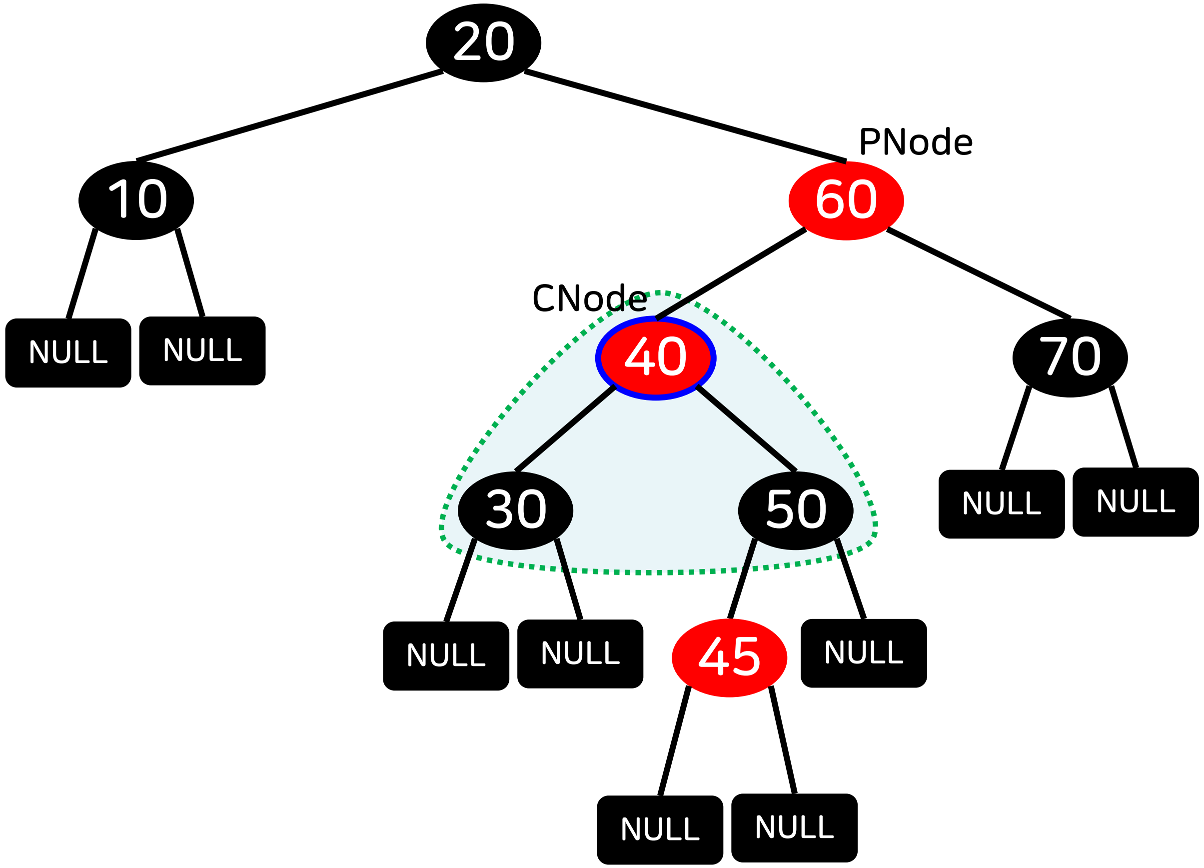
45 삽입 직후의 상태



<규칙1> 부모 노드50의 형제 노드30가 빨강인 경우

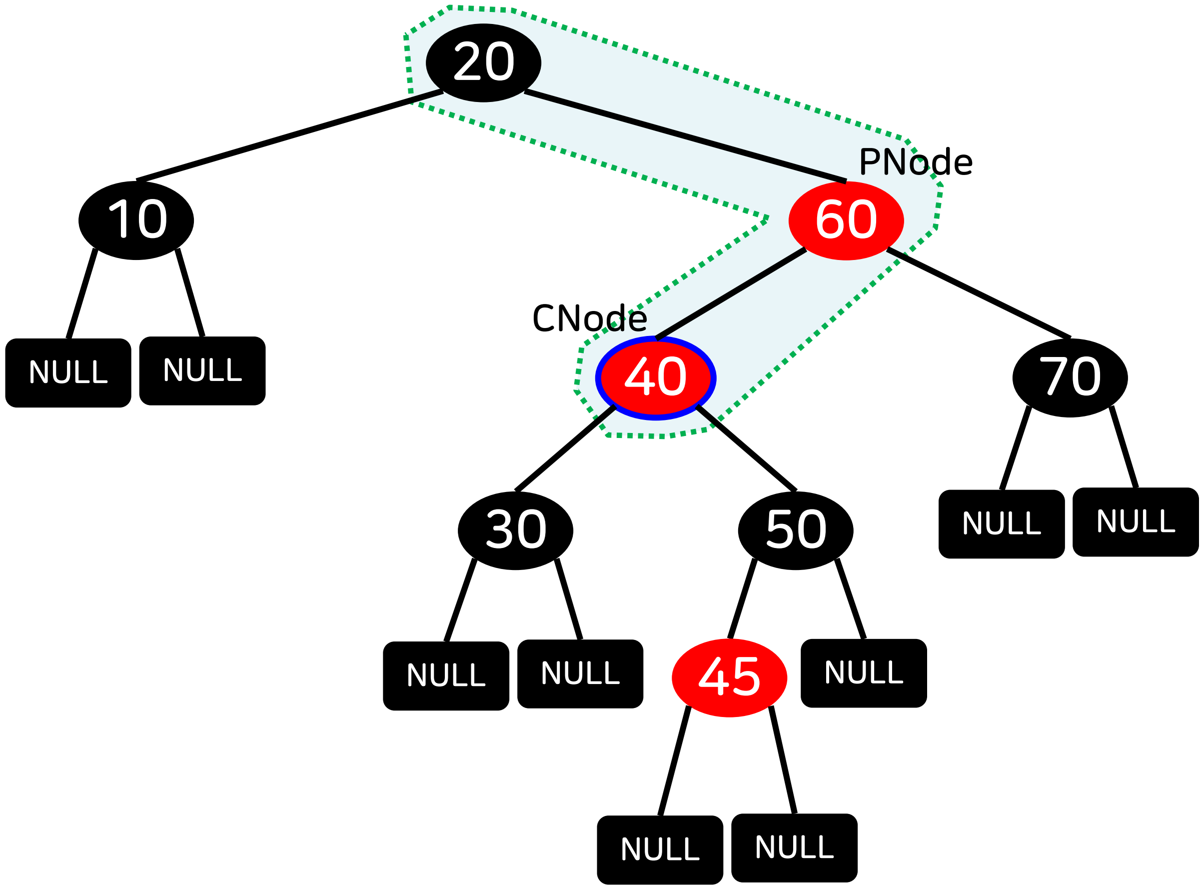
→ 부모 노드50, 부모 노드의 형제 노드30가, 부모 노드의 부모 노드 40의 색깔 변경

레드-블랙 트리_삽입 연산_2



<규칙 1>을 적용한 후의 상태

레드-블랙 트리_삽입 연산_2

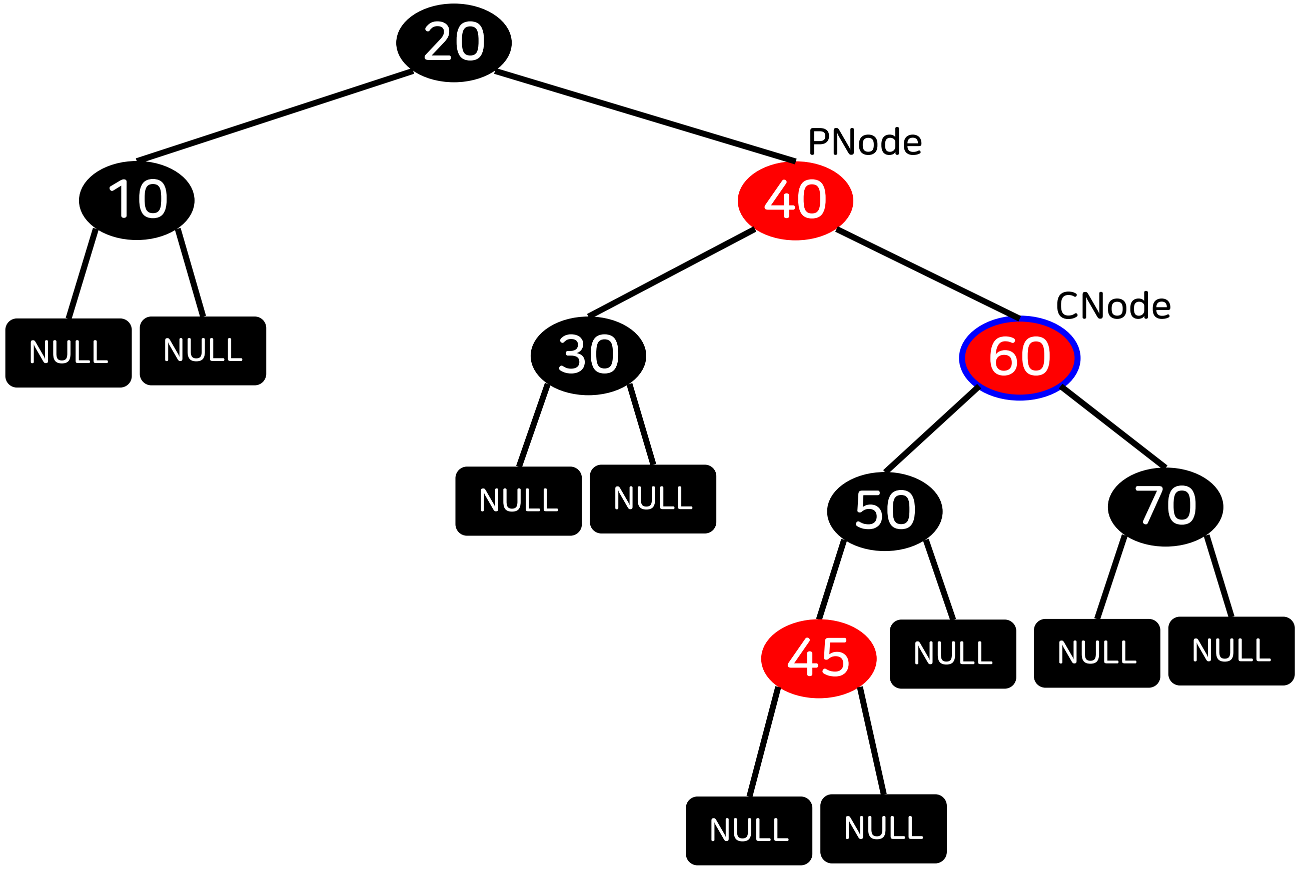


<규칙2> 부모 노드60의 형제 노드10가 검정,

현재 노드의 키값40이 부모 노드60와 부모 노드의 부모 노드20의 키값 사이인 경우

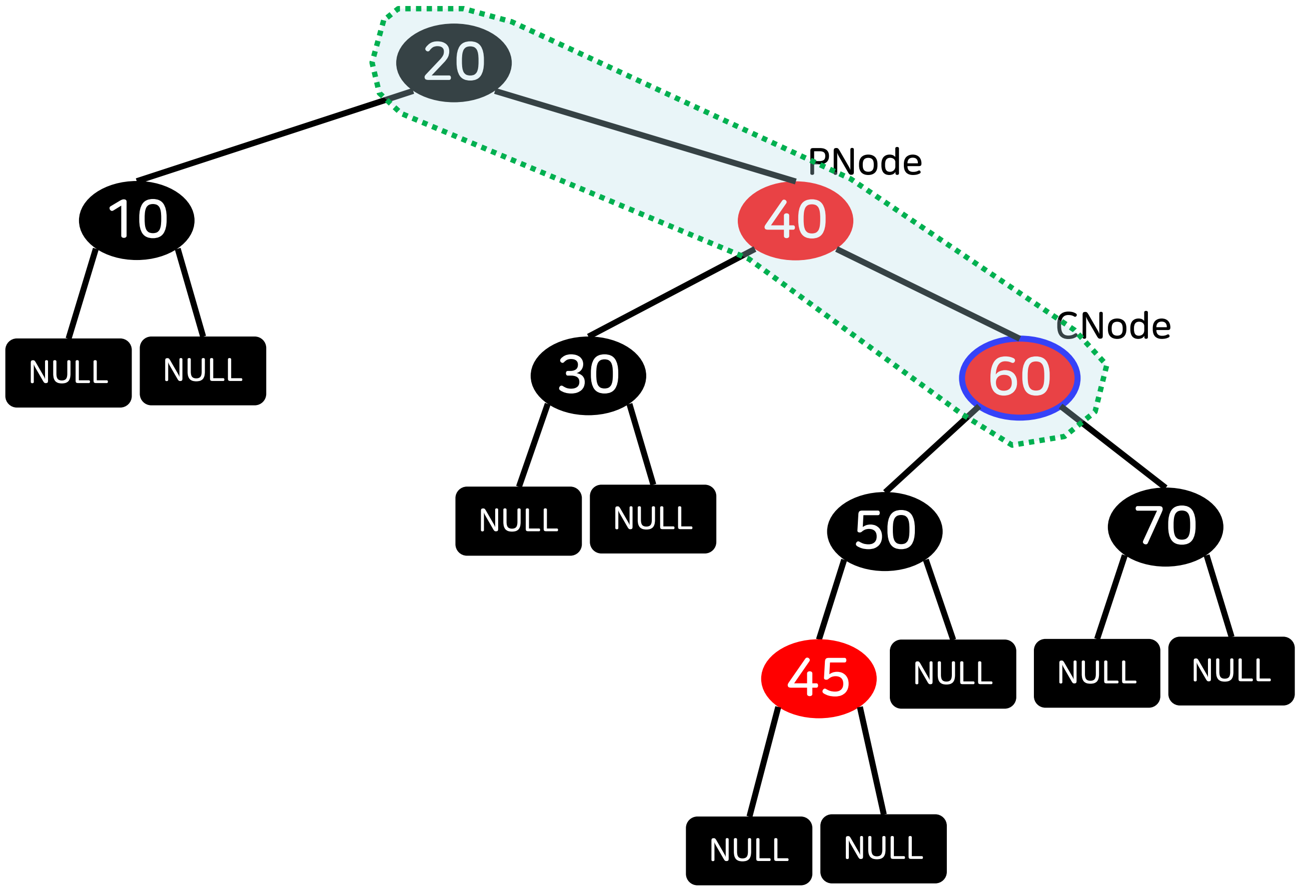
→ 현재 노드40와 부모 노드60을 회전시킴

레드-블랙 트리_삽입 연산_2



<규칙 2>를 적용한 후의 상태

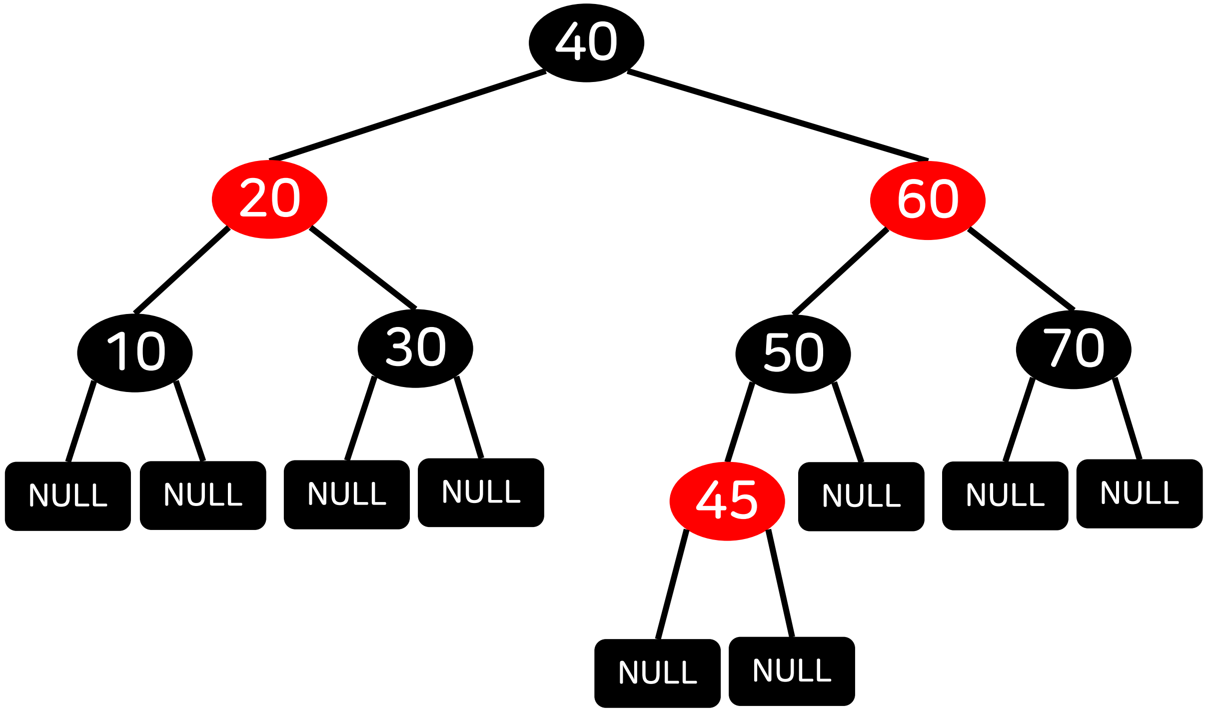
레드-블랙 트리_삽입 연산_2



<규칙3> 부모 노드60의 형제 노드10가 검정,
현재 노드의 키값60보다 부모 노드40와 부모 노드의 부모 노드20의 키값이 작은 경우

→ 부모 노드40와 부모 노드의 부모 노드20을 회전시키고 색깔 변경

레드-블랙 트리_삽입 연산_2



<규칙 3>을 적용한 후의 상태

키값 45를 삽입한 최종 상태

성능과 특징

균형 탐색 트리

- 어떤 두 리프 노드의 레벨 차이가 2배를 넘지 않는 균형 탐색 트리
  - ✓ 〈성질 3〉 → 루트 노드에서 리프 노드의 경로상에는 '빨강 노드의 개수 〈 검정 노드의 개수'
  - ✓ 〈성질 4〉 → 루트 노드에서 리프 노드의 경로상에는 동일한 개수의 검정 노드가 존재

탐색, 삽입, 삭제 연산의 시간 복잡도 → O(logn)

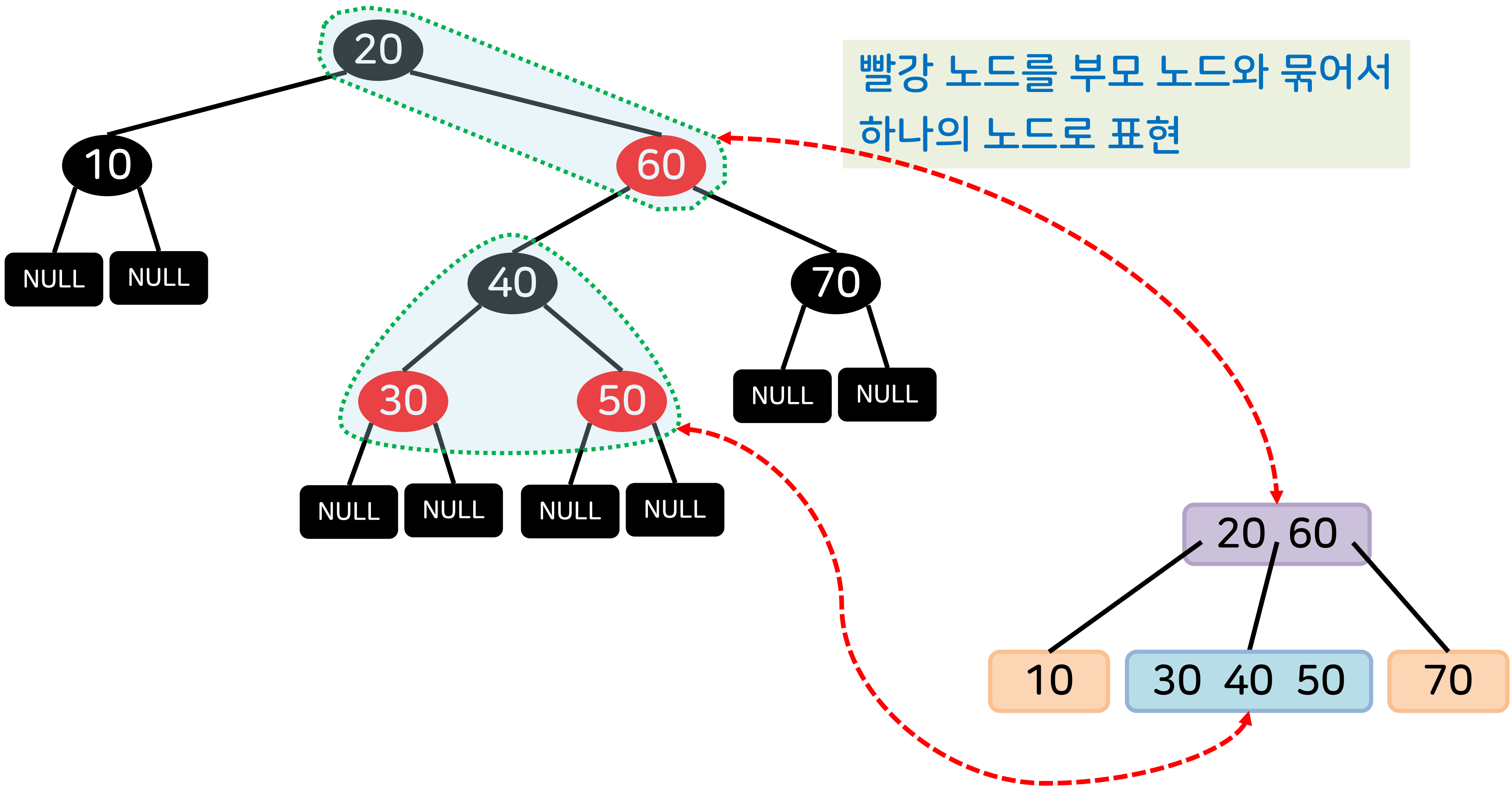
- 최악의 경우 트리의 높이 O(logn)

사실상 이진 탐색 트리

- 탐색 연산은 이진 탐색 트리와 동일
- 삽입 연산은 회전과 색깔 변경과 같은 추가 연산이 필요

성능과 특징

2-3-4 트리를 이진 탐색 트리로 표현한 것



02. B-트리

B-트리?

균형 탐색 트리 (t는 자연수인 상수)

〈성질 1〉 루트 노드는 1개 이상 2t개 미만의 오름차순으로 정렬된 키를 가짐

〈성질 2〉 루트 노드가 아닌 모든 노드는
(t-1)개 이상 2t개 미만의 오름차순으로 정렬된 키를 가짐

〈성질 3〉 내부 노드는 자신이 가진 키의 개수보다 하나 더 많은 자식 노드를 가짐

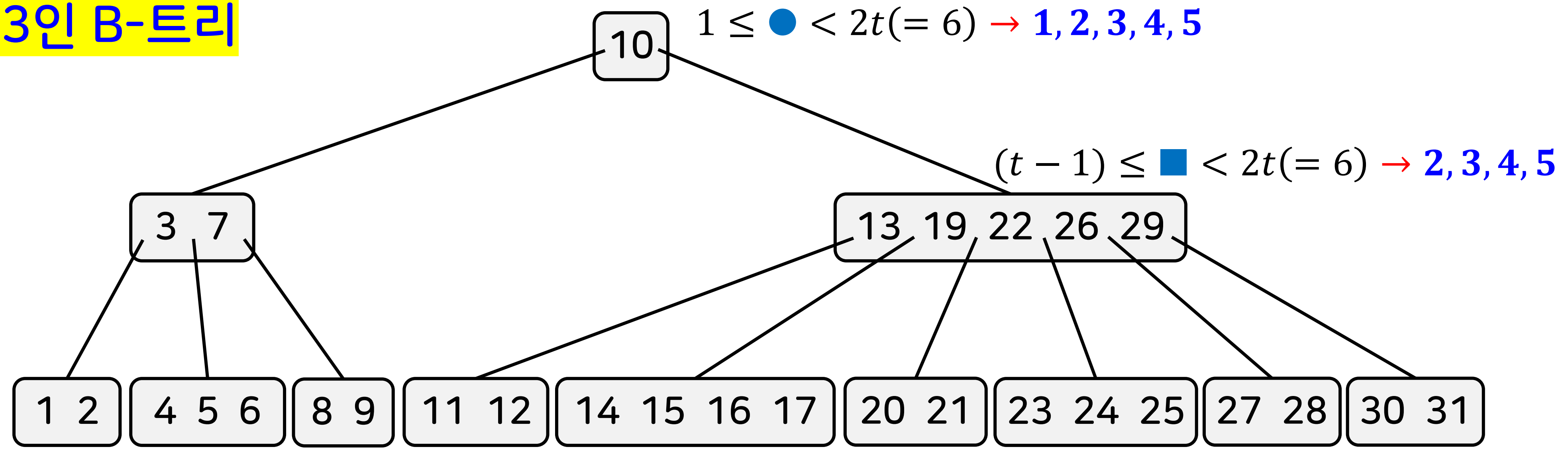
〈성질 4〉 각 노드의 한 키의 왼쪽 서브트리에 있는 모든 키값은 그 키값보다 작음

〈성질 5〉 각 노드의 한 키의 오른쪽 서브트리에 있는 모든 키값은 그 키값보다 큼

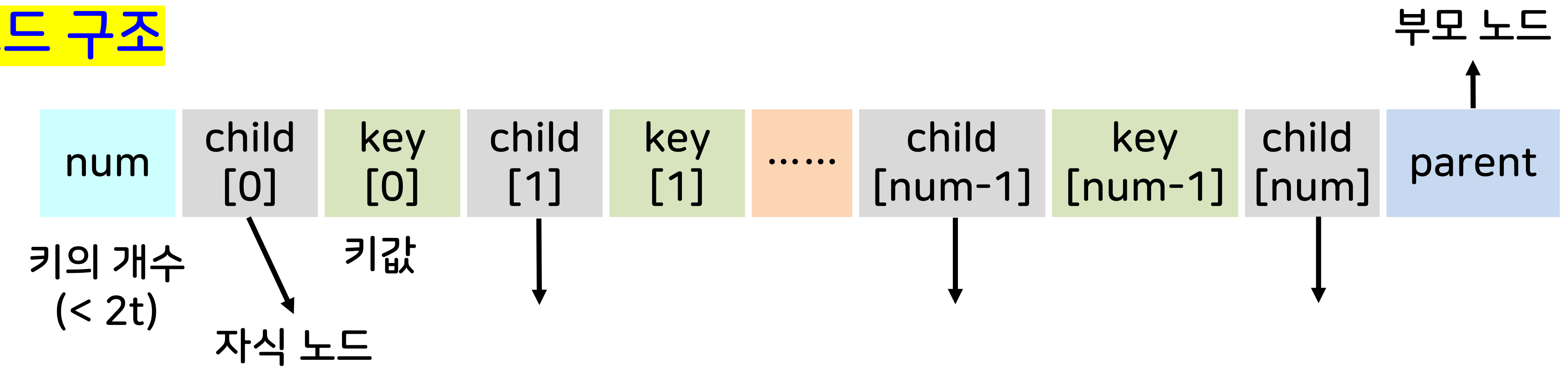
〈성질 6〉 모든 리프 노드의 레벨은 동일함

B-트리

t=3인 B-트리

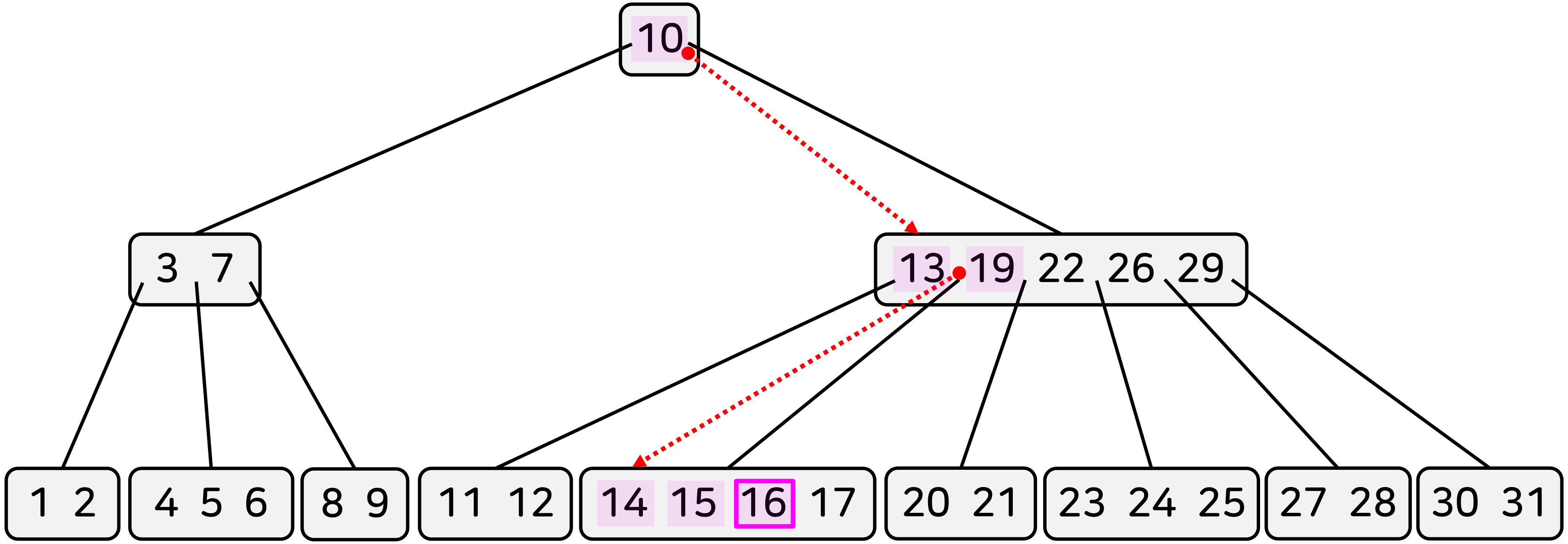


노드 구조



B-트리_탐색 연산

탐색 키 16

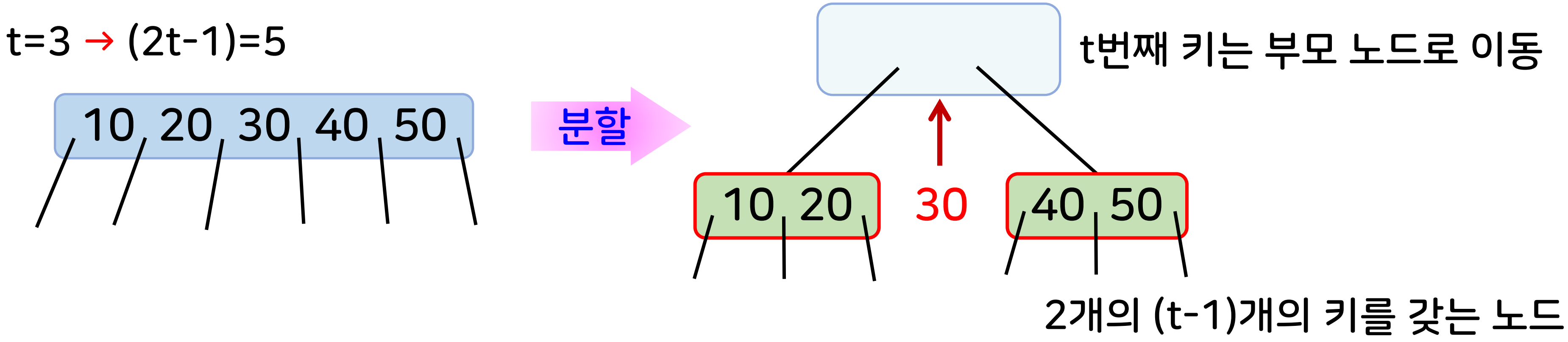


B-트리_삽입 연산

루트 노드에서부터 탐색을 수행하여 리프 노드에도 존재하지 않으면 해당 노드에 추가

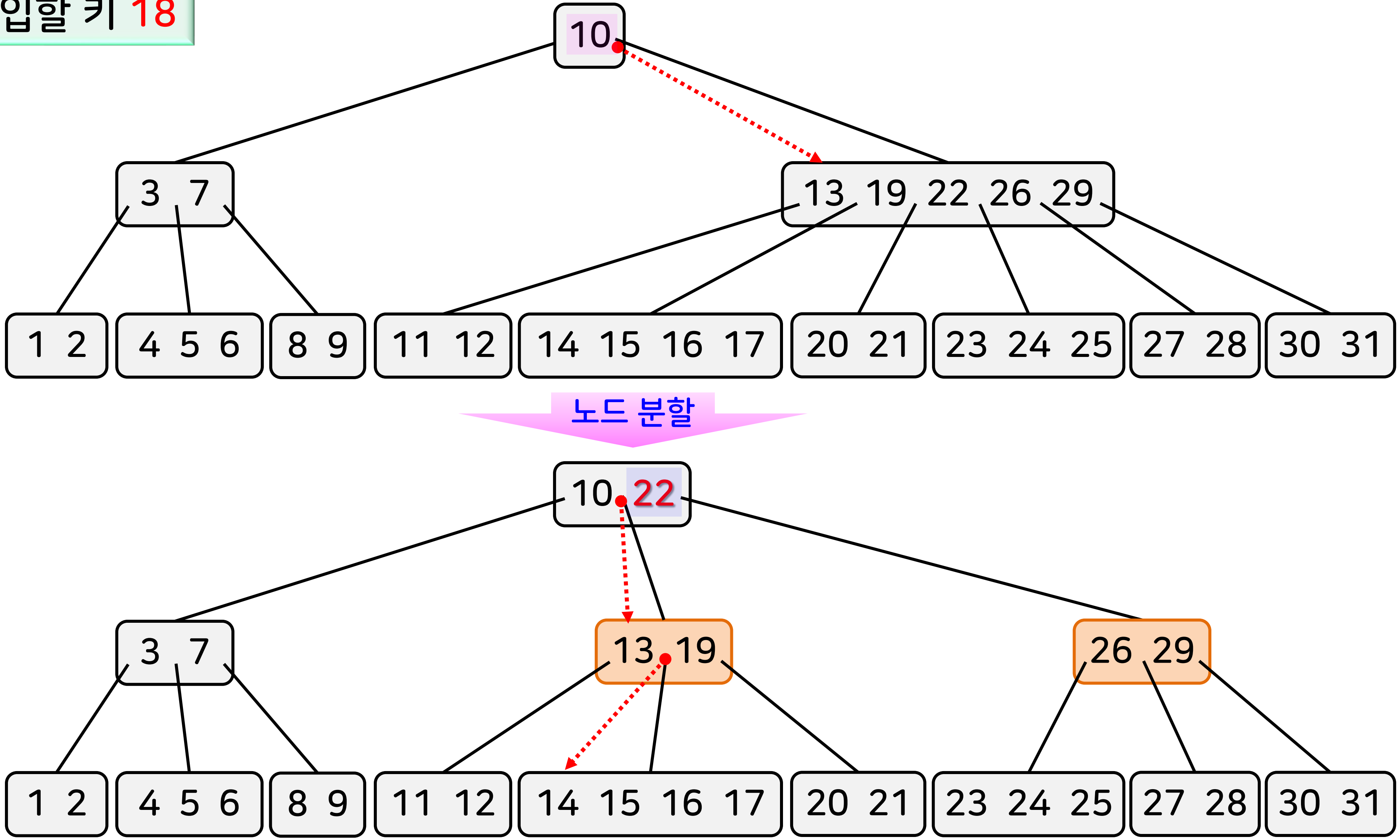
노드 분할

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



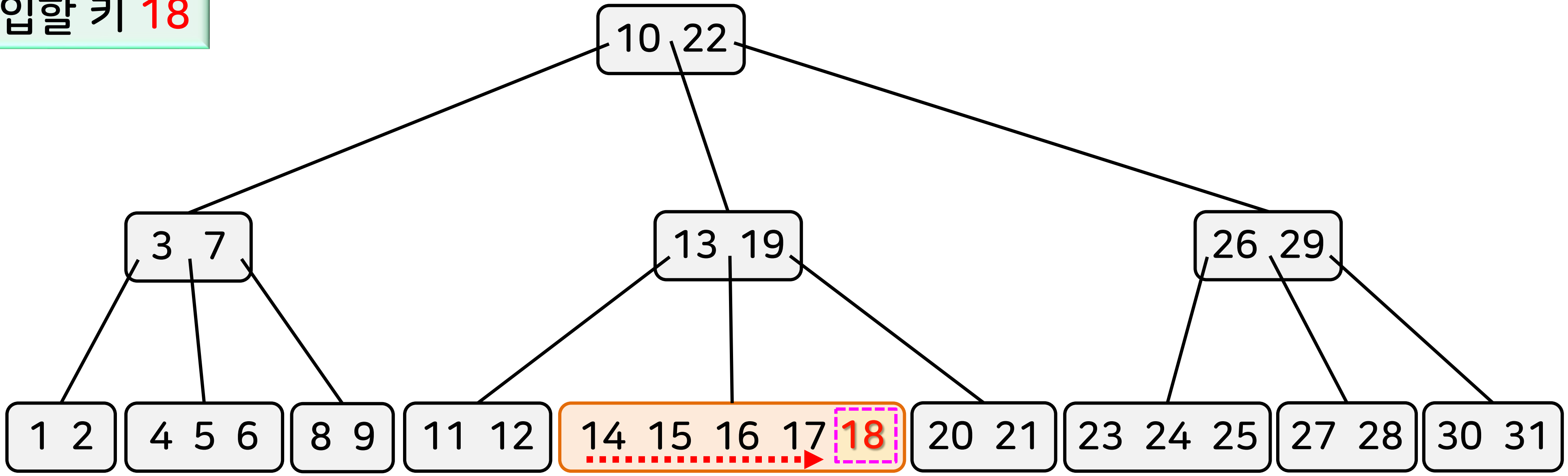
B-트리_삽입 연산

삽입할 키 18



B-트리_삽입 연산

삽입할 키 18



성능과 특징

탐색, 삽입, 삭제 연산의 시간 복잡도 → O(logn)

- 트리의 높이 h, 각 노드에서 키의 위치를 찾는 시간 O(t) → $O(th)$
  - ✓ 각 노드 → 키의 개수: (t-1) ∽ (2t-1)개, 자식 노드의 개수: t ∽ 2t개
  - ✓ 모든 리프 노드의 레벨은 동일
  - ✓ 트리의 높이 h → $O(\log_t n)$ (n: 키의 개수)
  - ✓ 각 노드에서의 키 관리에 레드-블랙 트리를 이용하면 O(t) 시간 → $O(\log t)$

$$O(th) \rightarrow O(\log t \log_t n) \rightarrow O(\log n)$$

성능과 특징

내부 탐색과 외부 탐색에 모두 활용

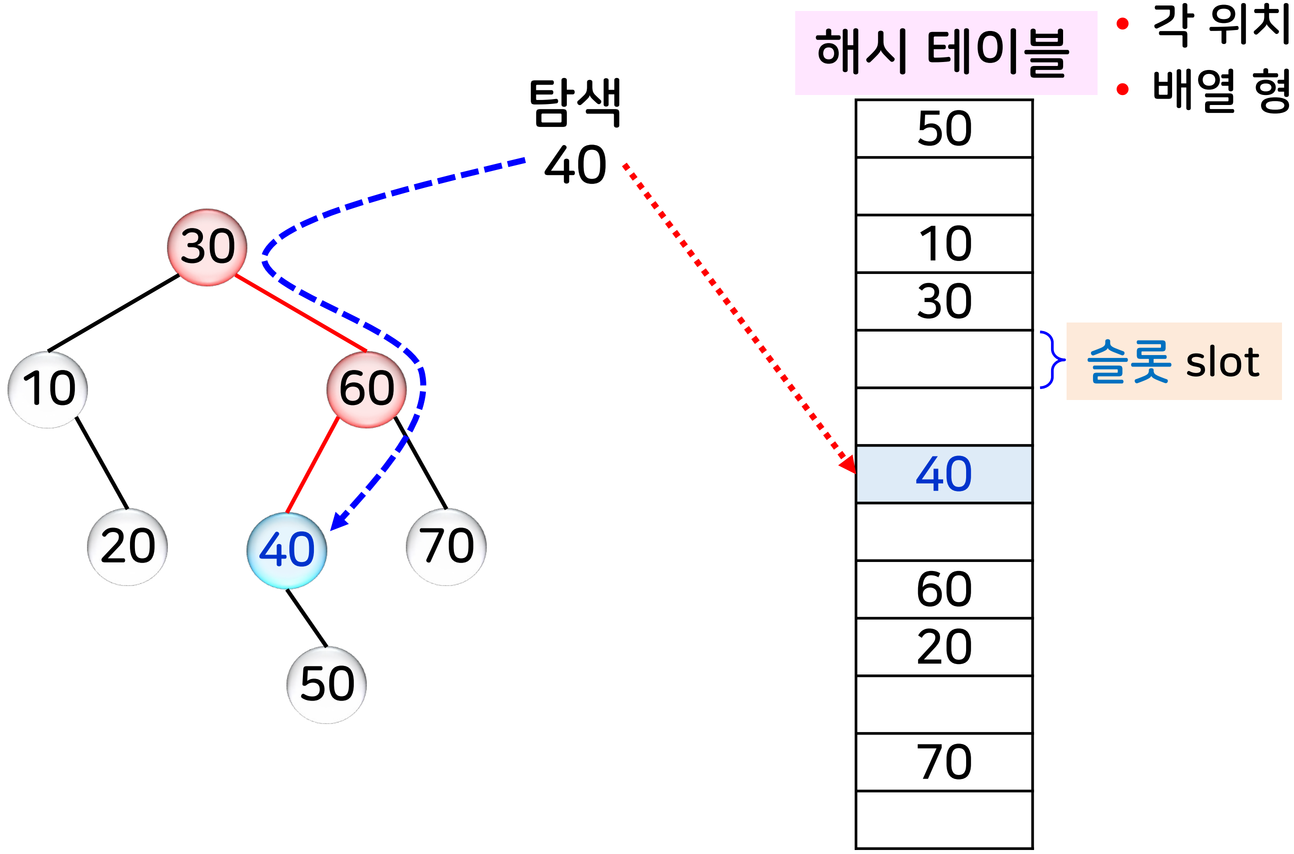
- 내부 탐색의 경우
  - ✓ t=2 또는 t=3 정도의 작은 값으로 지정
  - ✓ t=2 → '2-3-4 트리'
- 외부 탐색의 경우
  - ✓ 디스크를 사용하는 경우라면 t를 충분히 크게 지정

    → 한 노드의 크기가 디스크의 한 블록에 저장되도록 함

03. 해시 테이블

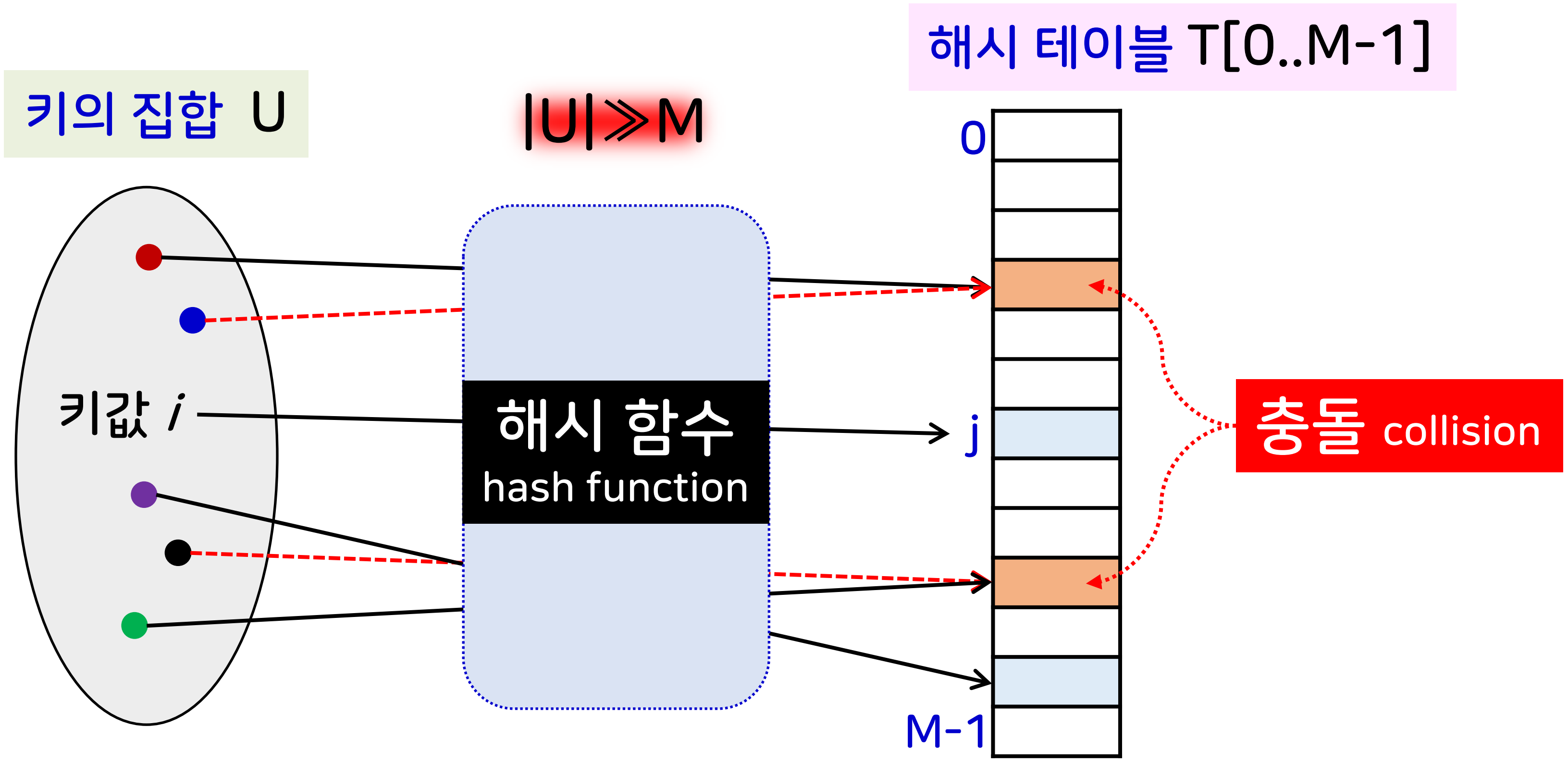
해싱?

키값을 기반으로 데이터의 저장 위치를 직접 계산함으로써 상수 시간 내에 데이터를 저장, 삭제, 탐색할 수 있는 방법



- 각 위치마다 주소가 부여되어 있는 저장공간
- 배열 형태

해싱의 개념


해시 테이블 T[0..M-1]
키의 집합 U
|U|≫M
0
키값 i
해시 함수
hash function
j
충돌 collision
M-1

해싱의 개념

해싱이 적합한 형태의 응용 문제는? ⑤

① 동일한 키값을 가진 여러 개의 데이터가 존재하는 응용

② 어떤 범위에 속하는 키값을 가진 모든 데이터를 탐색하는 문제

③ 최대 또는 최소의 키값을 가진 데이터를 찾는 문제

④ 키값의 순서대로 데이터를 방문하는 형태의 문제

⑤ 특정 키값 K를 갖는 데이터를 찾는 문제

해시 함수?

$h: U \rightarrow \{ 0, 1, \cdots, M-1 \}$

- 키값을 해시 테이블의 주소로 변환하는 함수
- 종류
  - ✓ 제산 잔여법, 비닝, 중간 제곱법, 문자열을 위한 함수(비닝, 단순 합, 가중 합) 등

바람직한 해시 함수?

- 계산이 용이해야 함
- 적은 충돌 발생 → 각 키를 테이블의 각 슬롯에 균등하게 사상시킬 수 있어야 함

해시 함수_제산 잔여법

h(K) = K mod M (K: 키값, M: 해시 테이블의 크기)

- h(123) = 123 mod 11 = 2

M의 선택에 주의해야 함

- M=2^r이면 h(K)는 키값의 하위 r비트의 값이 됨

→ 키값의 전체 비트가 주소 계산에 활용되지 못함

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

0~15 (0000 ~ 1111)

✓ 39 = 100111 → 7
✓ 3751 = 111010100111 → 7

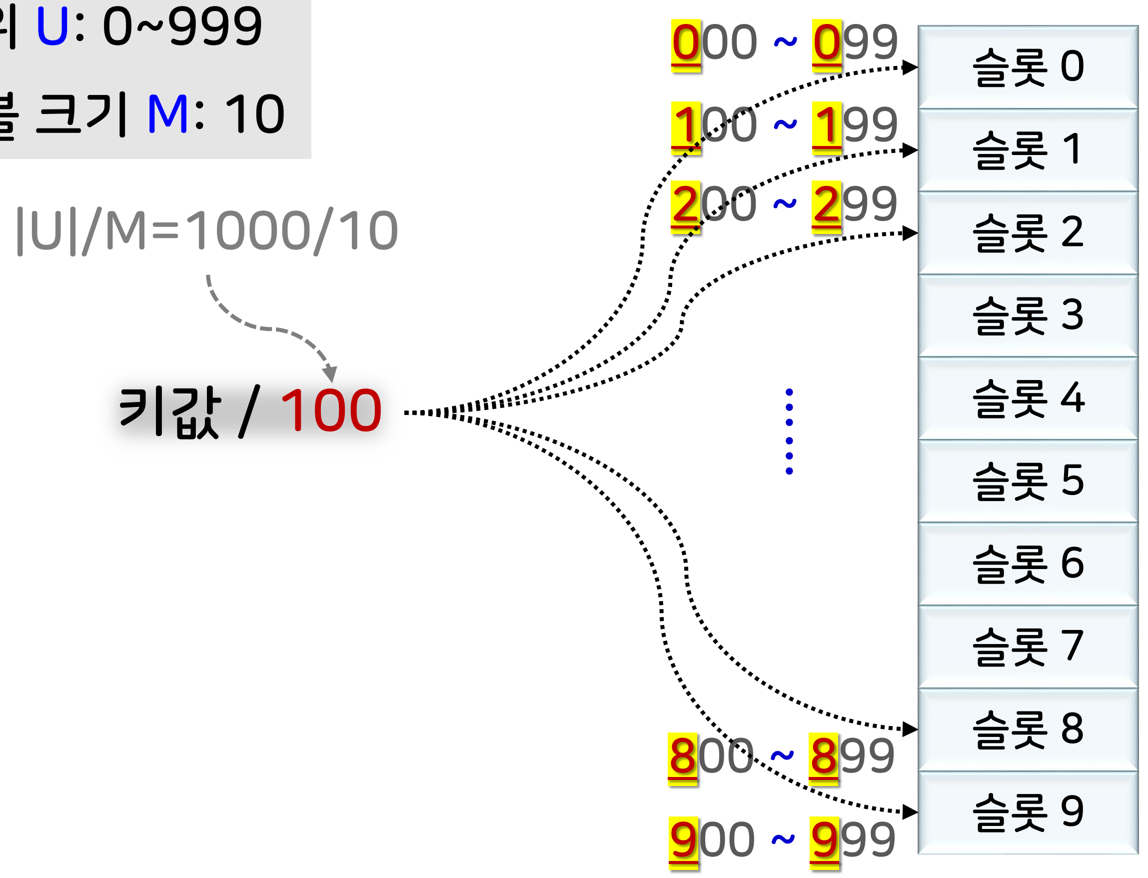
→ M은 2의 거듭제곱과 상당한 차이가 있는 소수로 선택하는 것이 바람직

해시 함수_비닝

U를 단순하게 M 등분하여 각 등분을 각 슬롯으로 해시

키값의 범위 U: 0~999

해시 테이블 크기 M: 10



상위 비트의 분포가 고르지 못하면 몇 개의 슬롯에 집중되는 문제

해시 함수_중간 제곱법

$h(K) = (K^2 / 2^m) \bmod 2^r$

(m: 키값을 제곱한 결과에서 사용하지 않을 하위 비트의 크기, r: 해시 주소로 취할 비트의 크기)

- 키값 → 3자리 십진수
- 해시 테이블의 크기 → 16 (r=4)
- m → 7

```
h (int x) {
  return (x*x / 128) % 16;
}
```

1. 주어진 키값을 제곱한다.

$(123)^2$ ⇒ 15129 → 11 1011 0001 1001

2. 제곱된 결과에서 하위 m를 버리고, M에 해당하는 하위 r비트를 취한다.

11 1011 0001 1001 ⇒ 111 0110 ⇒ 0110 = $(6)_{10}$

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

해시 함수_문자열을 위한 비닝

U를 단순하게 M 등분하여 각 등분을 각 슬롯으로 해시

- 문자열의 앞쪽 일부를 해시 결과로 사용

→ 입력 문자열의 앞쪽의 분포가 고르지 못하면 결과가 슬롯에 고르게 분포되지 못함

```
h (int x[ ]) {
  return ( x[0] - 'A' );
}
```

영어 대문자로 구성된 문자열인 키값 x[ ]를 26개의 슬롯으로 해시하는 함수

문자열의 첫 글자를 기반으로 슬롯을 할당 →

'A--' → 슬롯 0
'B--' → 슬롯 1
…
'Z--' → 슬롯 25

해시 함수_문자열을 위한 단순 합

각 문자의 코드값을 합한 후 제산 잔여법 적용

$$h(K) = \left(\sum_{i=0}^{|K|-1} K[i] \right) \bmod M$$

```
h1 (x[ ], M) {
  int xlength = length(x);
  int sum = 0;
  for (i=0; i < xlength; i++)
    sum += x[i];
  return (sum % M);
}
```

키값 x[ ] = "AAAAAZZZZZ"의 값은?
해시 테이블 크기 M=100

A:65,⋯, Z:90 → 65×5+90×5 = 775

775 % 100 = 75

- M≪sum일 때 유용
- 짧은 문자열에 대해서는 비효과적
- 문자의 출현 순서는 고려되지 않음 → h('ABC') = h('BCA')

해시 함수_문자열을 위한 가중 합

각 문자의 코드값에 자리에 따른 가중치를 곱한 값을 합한 후 제산 잔여법을 적용

$$h(K) = \left(\sum_{i=0}^{|K|-1} K[i] D^{|K|-1-i} \right) \bmod M$$

i번째 자리$(0 \le i \le |K|-1)$의 가중치

```
h2 (x[ ], M, D) {
  int xlength = length(x);
  int wsum = 0;
  for (i=0; i < xlength; i++)
    wsum = (wsum * D + x[i]) % M;
  return (wsum);
}
```

h2 (x[ ] = “ABC”, M=1000, D=256)=?

wsum ← (0 * 256 + 65) % 1000 = 65

wsum ← (65 * 256 + 66) % 1000 = 706

wsum ← (706 * 256 + 67) % 1000 = 803

h2(“ABC”,⋯)=803 ≠ h2(“BCA”,⋯)=593

충돌 해결 방법

충돌?

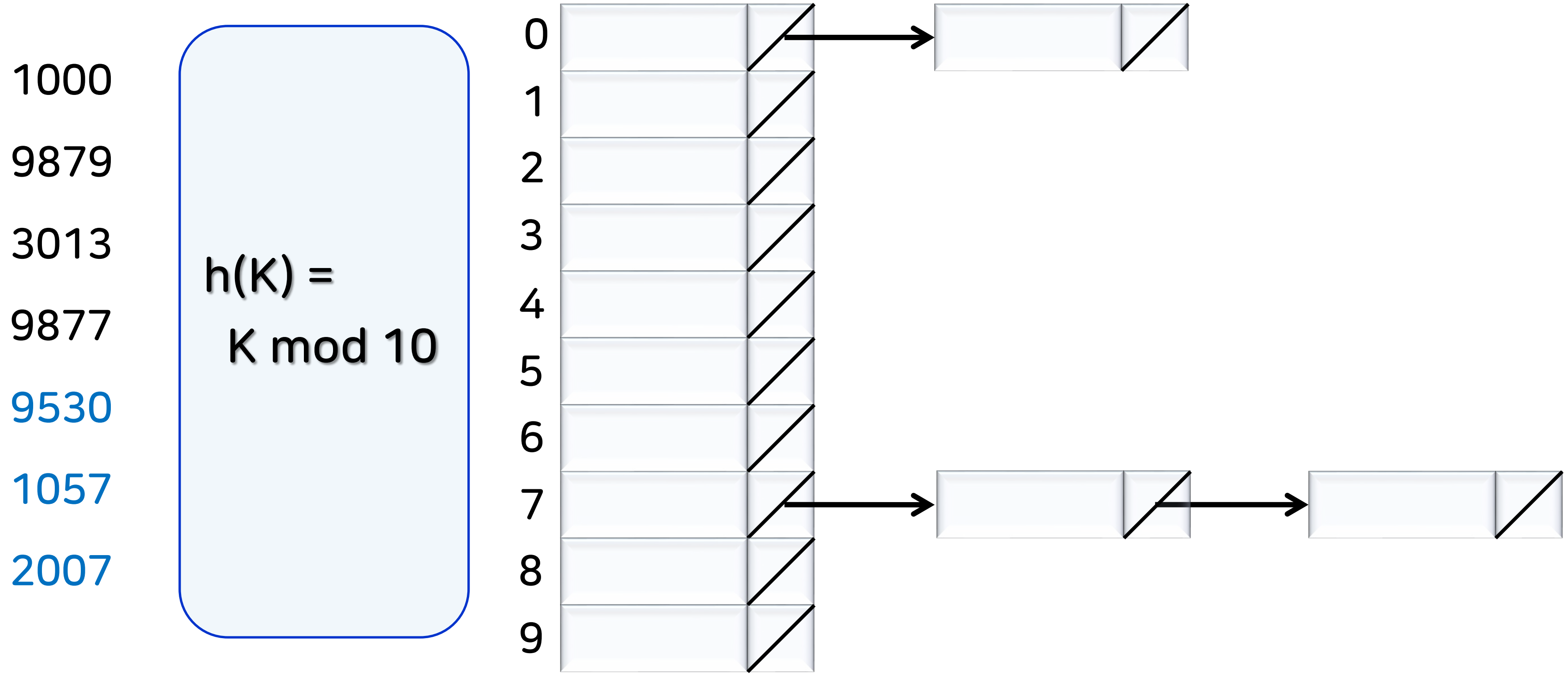
- 서로 다른 키값 x, y에 대하여 h(x)=h(y)인 경우

충돌 해결 방법

- 개방 해싱 ("연쇄법")
  - ✓ 충돌된 데이터를 테이블 밖의 별도의 장소에 저장/관리 → 연결 리스트 사용
- 폐쇄 해싱 ("개방 주소법")
  - ✓ 해시 테이블 내의 다른 슬롯에 충돌된 데이터를 저장/관리
  - ✓ 종류 → 버킷 해싱, 선형 탐사, 이차 탐사, 이중 해싱

충돌 해결 방법_개방 해싱

해시 테이블의 각 슬롯을 연결 리스트의 헤더로 사용



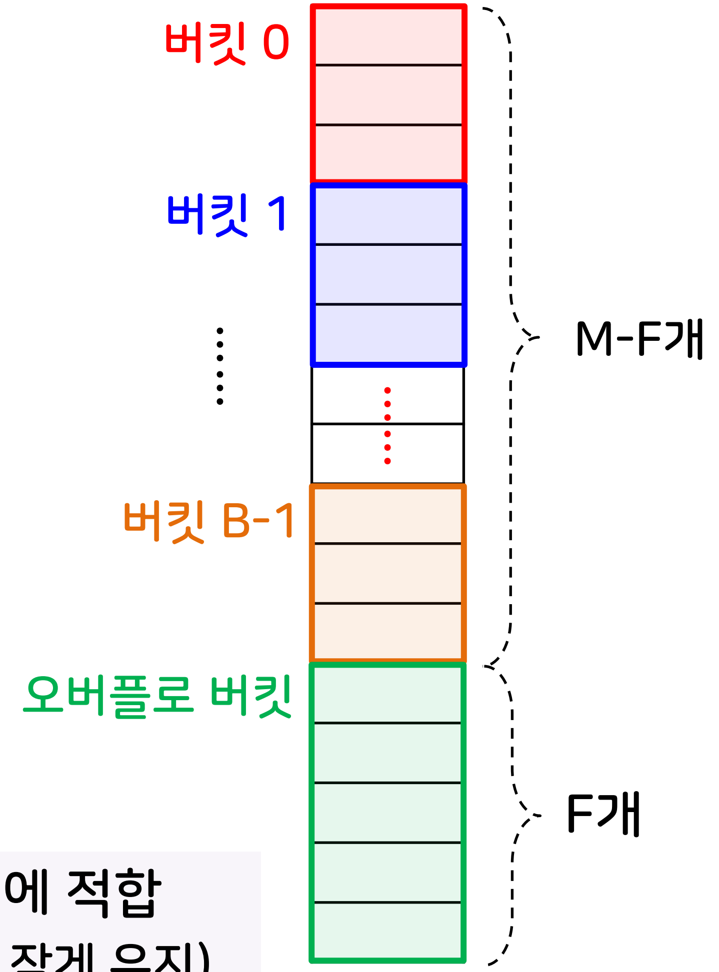
해시 테이블과 연결 리스트가 주기억장치 내에서 유지될 때 적합

충돌 해결 방법_폐쇄 해싱_버킷 해싱

해시 테이블 슬롯을 버킷으로 묶어 버킷 단위로 해싱

671
1501
351
1121

h(K) = K mod 10



디스크에 저장된 해시 테이블 구현에 적합
(버킷 크기 = 디스크 크기, 오버플로는 작게 유지)

충돌 해결 방법_폐쇄 해싱_선형 탐사

탐사 순서 probe sequence

- 어떤 키 K에 대해서 탐사되는 슬롯의 순서열
  - ✓ p(K, i) → p(K, 0)=h(K), p(K, 1), p(K, 2), p(K, 3), …
- 탐사 순서의 계산 방법에 따라 성능의 차이가 발생
  - ✓ 선형 탐사, 이차 탐사, 이중 해싱

선형 탐사 linear probing

- p(K, i) = (h(K) + i) mod M (i=0, 1, 2, …, M-1)
  - ✓ 홈 위치가 사용 중이면 빈 슬롯을 찾을 때까지 테이블의 다음 슬롯으로 순차적으로 이동
- 가장 간단하지만 최악의 방법

충돌 해결 방법_폐쇄 해싱_선형 탐사

h(K)= K mod 10 → p(K, i)= (K + i) mod 10

1000
9879
3013
9877
1057
9530
2007

슬롯별 저장 확률

슬롯	키	저장 확률	1057, 9530 → 저장 확률	키
0	1000	0	0	1000
1		3/10	0	9530
2		1/10	6/10	
3	3013	0	0	3013
4		2/10	2/10	
5		1/10	1/10	
6		1/10	1/10	
7	9877	0	0	9877
8		2/10	0	1057
9	9879	0	0	9879

- 모든 슬롯이 새로운 데이터가 삽입될 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_폐쇄 해싱_이차 탐사

탐사 순서의 계산에 이차식을 이용

- $p(K, i) = (h(K) + c_1 \cdot i^2 + c_2 \cdot i + c_3) \bmod M$
  - ✓ 충돌이 발생하는 횟수의 제곱 형태로 탐사 순서를 결정

서로 다른 홈 위치를 갖는 두 키는 서로 다른 탐사 순서를 가짐

- $p(K, i) = (h(K) + i^2) \bmod 10$
  - ✓ $h(K_1) = 1 \rightarrow 1, 2, 5, 0, \cdots$
  - ✓ $h(K_2) = 4 \rightarrow 4, 5, 8, 3, \cdots$

충돌 해결 방법_폐쇄 해싱_이차 탐사

$h(K) = K \bmod 10$ → $p(K,i) = (K + i^2) \bmod 10$

1000		
9879		
3013		
9877		
1057	p(1057, 0)=7 p(1057, 1)=8	
9530	p(9530, 0)=0 p(9530, 1)=1	
2007	p(2007, 0)=7 p(2007, 1)=8 p(2007, 2)=1 p(2007, 3)=6	

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

충돌 해결 방법_폐쇄 해싱_이차 탐사

모든 슬롯이 탐사 순서에 사용되지 않음

- $p(K, i) = (h(K) + i^2) \bmod 10$
  - ✓ p(K, 0)=1 → 슬롯 0, 1, 2, 5, 6, 7만 탐사 가능 → 슬롯 3, 4, 8, 9는 사용 불가
- 탐사 함수와 해시 테이블 크기가 적절히 조합되면 많은 슬롯의 방문이 가능

2차 클러스터링 문제

- 동일한 홈 위치를 갖는 두 키는 동일한 탐사 순서를 가짐

  → 특정 홈 위치에 대한 클러스터를 만드는 현상

충돌 해결 방법_폐쇄 해싱_이중 해싱

탐사 순서를 원래의 키값을 이용하여 해싱

- $p(K,i) = (h_1(K) + i \cdot h_2(K)) \bmod M$
- 1차/2차 클러스터링 문제 해결
- 서로 다른 두 키의 홈 위치가 동일해도 서로 다른 탐사 순서를 가짐

좋은 이중 해싱을 구하려면

- 탐사 순서에 있는 모든 상수가 해시 테이블의 크기 M과 서로 소가 되어야 함
  - ✓ M을 소수로 선택하고, h_2가 $1 \leq h_2(K) \leq M-1$의 값을 반환하는 방법
  - ✓ 어떤 m에 대해 $M=2^m$으로 정하고, h_2가 1과 2^m 사이의 홀수를 반환하는 방법

삭제 연산

두 가지 고려 사항

- 데이터의 삭제가 차후의 탐색을 방해하지 말아야 함
  - ✓ 단순히 빈 슬롯으로 두면 탐색이 해당 슬롯에서 종료되므로 그 이후의 데이터는 고립됨
- 삭제로 생긴 빈 슬롯은 나중에 삽입 과정에서 사용되어야 함

비석 tombstone

- 삭제된 데이터의 위치에 '비석'이라는 특별한 표시를 하는 방법
  - ✓ 탐색 → 탐색하는 동안 비석을 만나면 탐색을 계속 진행
  - ✓ 삽입 → 비석이 표시된 위치를 빈 위치로 간주하여 새 데이터를 삽입
- 비석의 개수가 증가할수록 평균 탐색 거리가 증가

정리하기

1. 레드-블랙 트리

- 균형 탐색 트리, 성질-6가지
- 연산(탐색, 삽입-노드의 구조나 색깔을 조정하는 3가지 규칙)
- O(logn), 2-3-4 트리를 이진 탐색 트리로 표현한 것

2. B-트리

- 균형 탐색 트리, 성질-6가지, 연산(탐색, 삽입-노드 분할)
- O(logn), 내부 탐색과 외부 탐색에 활용

3. 해시 테이블

- 개념(해싱, 해시 테이블, 해시 함수, 충돌)
- 해시 함수 → 제산 잔여법, 비닝, 중간 제곱법, 문자열을 위한 함수(비닝, 단순 합, 가중 합)
- 충돌 해결 방법 → 개방 해싱, 폐쇄 해싱(→ 버킷 해싱, 선형탐사, 이차 탐사, 이중 해싱)
- 데이터 삭제와 비석

ALGORITHM 알고리즘

다음시간에는

Lecture **08**

그래프 (1)

컴퓨터과학과 | **이관용 교수**