

검색

목차

1. 순차 검색
2. 이진 검색
3. 해싱
4. 균형 이진 탐색 트리
5. 다원 탐색 트리

검색이란?

- 검색이란?
 - 저장된 자료 중에서 원하는 자료를 찾는 것
 - 키(Key)
 - 찾고자 하는 자료를 다른 자료들과 구분시켜주는 키
- 검색의 종류
 - (검색 방법)에 따른 분류
 - 비교 검색 방법
 - 검색 키 값을 비교하여 검색
 - 순차 검색, 이진 검색, 트리 검색 등
 - 계산 검색 방법
 - 검색 키를 계산하여 검색
 - 해싱

순차 검색이란?

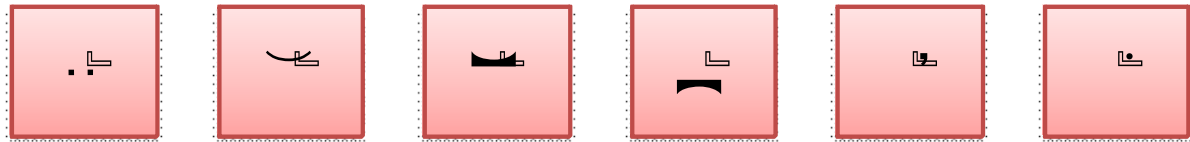
1. 순차 검색

- 순차 검색
 - 검색 키를 차례대로 비교하여 검색
- 종류
 - 검색하려는 자료들이
 - 1) 미리 정렬되어 있지 않은 경우
 - 2) 미리 정렬된 경우
 - 색인(Index)
 - 미리 정렬된 경우

1.1 자료가 미리 정렬되지 않은 경우

1. 순차 검색

- 순차 검색
- 검색 성공의 경우
 - 검색 키: 20
- 검색 실패의 경우
 - 검색 키: 25

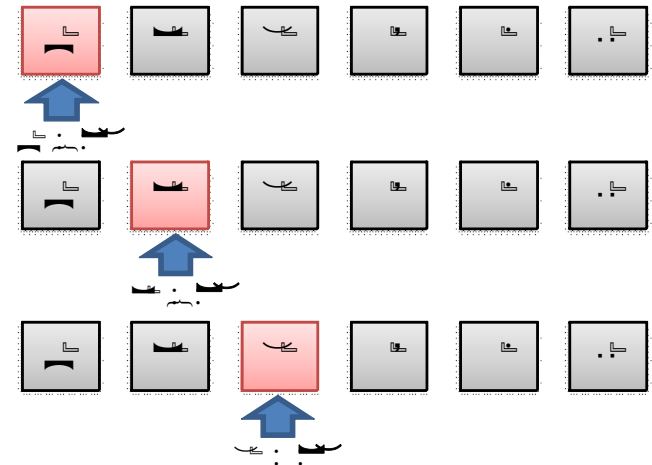
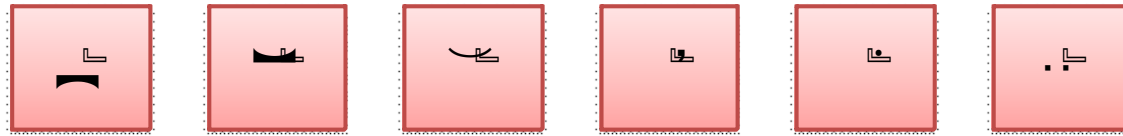


- 평균 비교 연산 횟수

$$O\left(\frac{1}{n} \times (1 + 2 + 3 + \dots + n)\right) = O\left(\frac{n+1}{2}\right) = O(n)$$

1.2 자료가 미리 정렬된 경우 (1/2)

- 순차 검색
 - 검색 실패 여부를 자료의 마지막 원소까지 찾지 않아도 알 수 있다
- 검색 성공의 경우
 - 검색 키: 20
- 검색 실패의 경우
 - 검색 키: 25

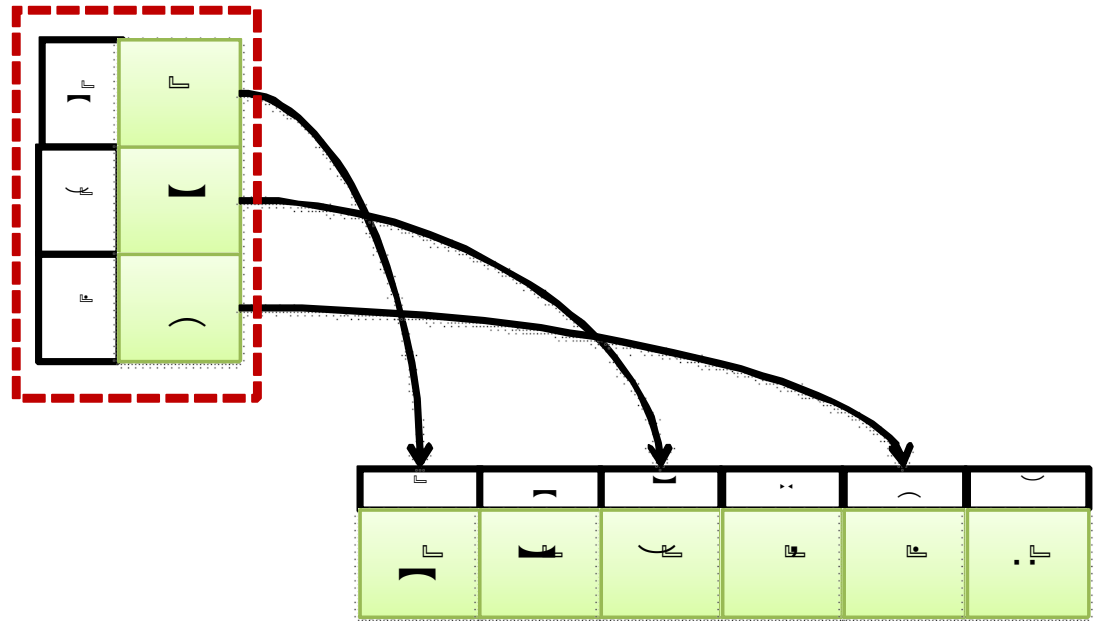


- 평균 비교 연산 횟수

1.3 색인 순차 검색 (1/3)

- 색인 순차 검색(Index Sequential Search)
 - 색인, 인덱스: 특정 키 값을 가지는 자료의 위치
 - 인덱스 테이블(Index Table)
 - 인덱스를 모아둔 테이블
 - 검색 범위 축소

1. 순차 검색



10 < 60



10	0
50	2
70	4

50 < 60



10	0
50	2
70	4

70 > 60



10	0
50	2
70	4

검색 범위

10	0
50	2
70	4

검색 범위

0	1	2	3	4	5
10	20	50	60	70	80

검색 범위



60 = 60

1.3 색인 순차 검색 (2/3)

1. 순차 검색

- 검색 범위

$\text{IndexTable}[\underline{i-1}].\text{key} \leq \text{key} < \text{IndexTable}[i].\text{key}$
 $\Rightarrow \text{IndexTable}[\underline{i-1}].\text{position} \leq \text{검색 범위} < \text{IndexTable}[i].\text{position}$

1.3 색인 순차 검색 (3/3)

1. 순차 검색

- 소스 파일 구성

파일 이름	내용
10_02.vcproj	Visual Studio 프로젝트 파일
seqindexsearch.h	색인 순차 검색 함수 선언
seqindexsearch.c	색인 순차 검색 함수 구현
exampl10_02.c	예제 프로그램

- 평균 비교 연산 횟수

- 검색 대상이 되는 전체 자료의 개수가 n 개
- 인덱스 테이블의 인덱스 개수가 m 개

$$O\left(m + \frac{n}{m}\right)$$

이진 검색 (1/3)

- 이진 검색(Binary Search)

2. 이진 검색

- 검색 범위를 절반으로 감소시켜 가면서 검색
- 미리 정렬되어 있어야 함

- 예)

- 검색키: 60

1	2	3	4	5	6
10	20	30	40	50	60

$$\frac{(1 \text{ 번째 원소} + \text{마지막 원소})}{2} = \frac{0 + 5}{2} = 2.5 \rightarrow 2$$

중간 위치

0	1	2	3	4	5
10	20	50	60	70	80



$50 < 60$

← 검색 범위 →

0	1	2	3	4	5
10	20	50	60	70	80



← 검색 범위 →



$70 > 60$

중간 위치

0	1	2	3	4	5
10	20	50	60	70	80



$60 == 60$

이진 검색 (2/3)

- 의사 코드(Pseudo code)

2. 이진 검색

```
binarySearch(value, start, end, key)
{
    result ← FAIL

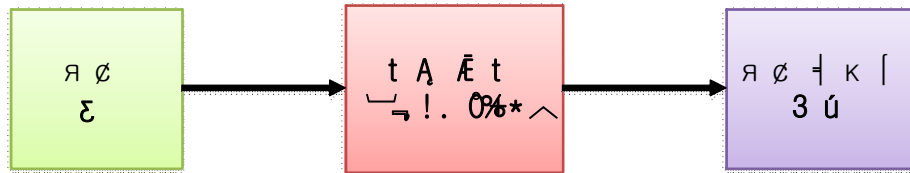
    tempStart ← start
    tempEnd ← end
    while (tempStart <= tempEnd) {
        middle = (tempStart + tempEnd) / 2
        if (key == value[middle]) {
            result ← middle
            break
        }
        else if (key < value[middle]) {
            tempEnd ← middle - 1
        }
        else {
            tempStart ← middle + 1
        }
    }
    return result
}
```

- 평균 비교 연산 횟수
 - $O(\log N)$

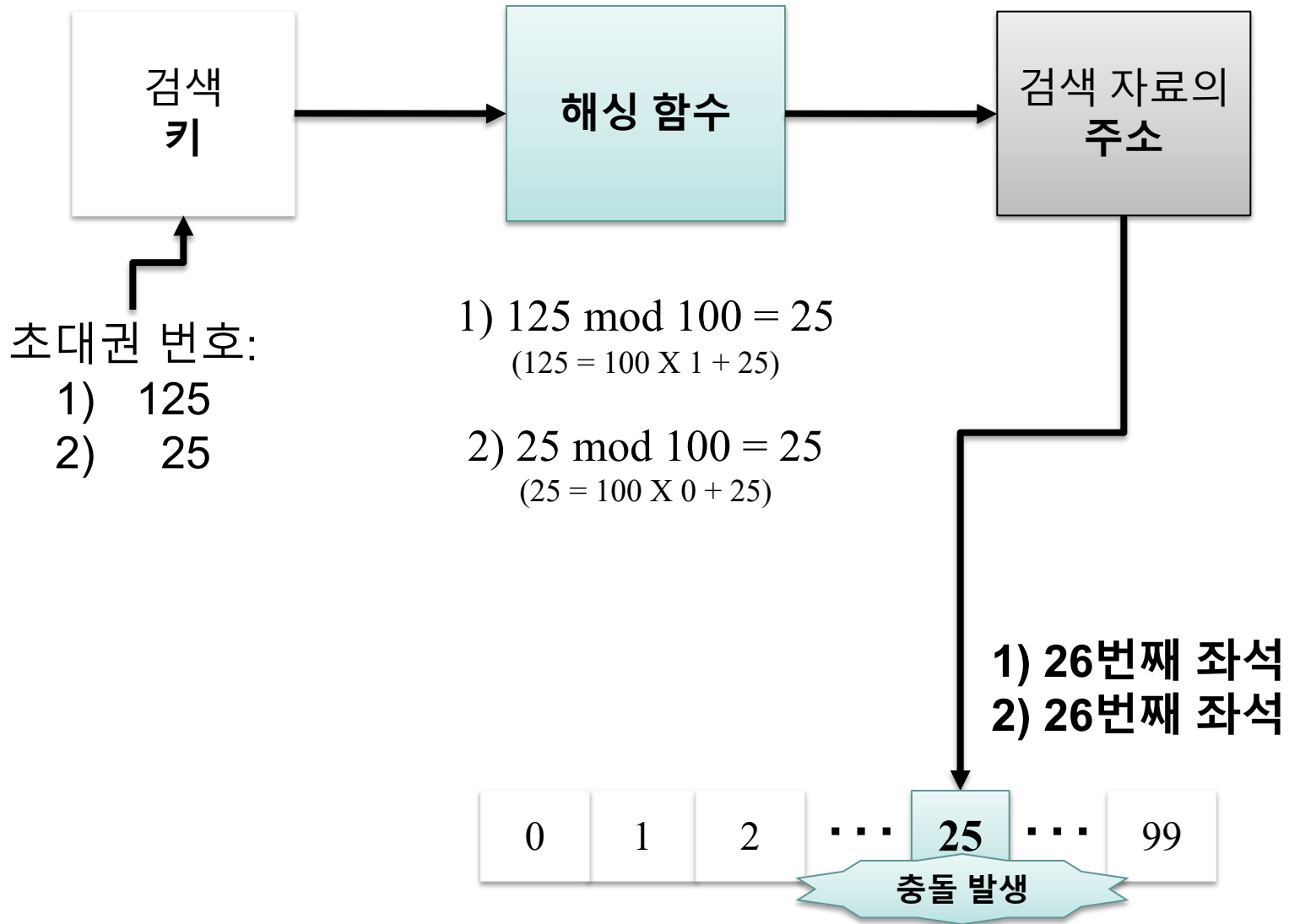
해싱

3. 해싱

- 해싱(Hashing)
 - 산술 연산을 이용한 검색 방식



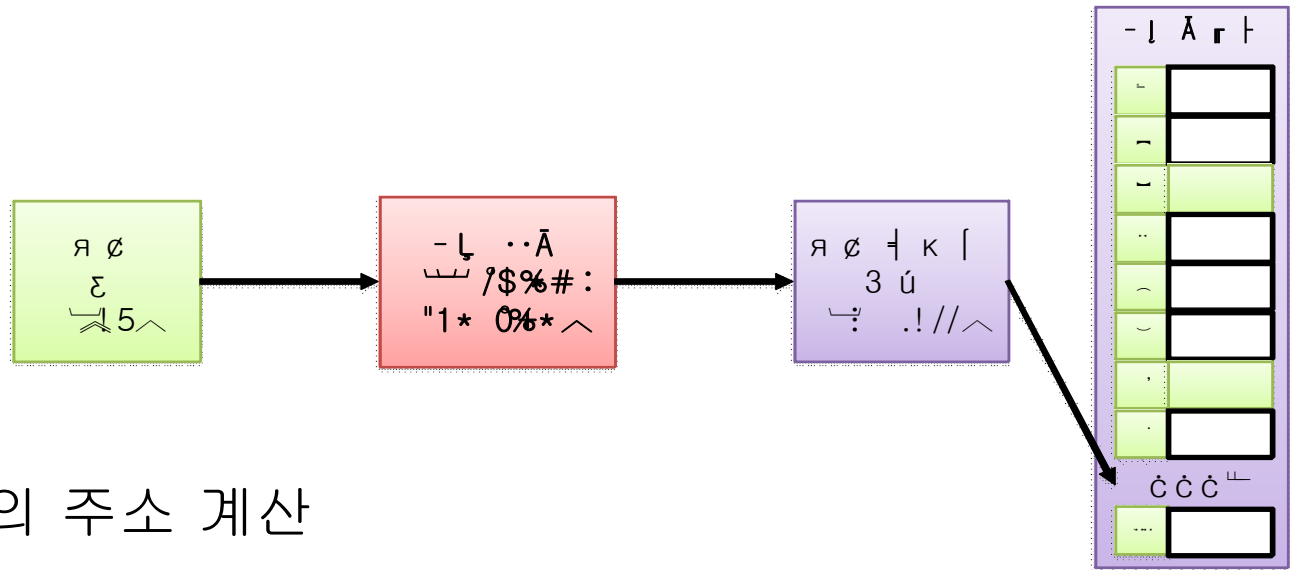
- 평균 시간 복잡도
 - 자료 N개
 - $O(1)$
- 문자열의 경우?



3.1 해싱 검색

- 해싱 검색의 단계

3. 해싱

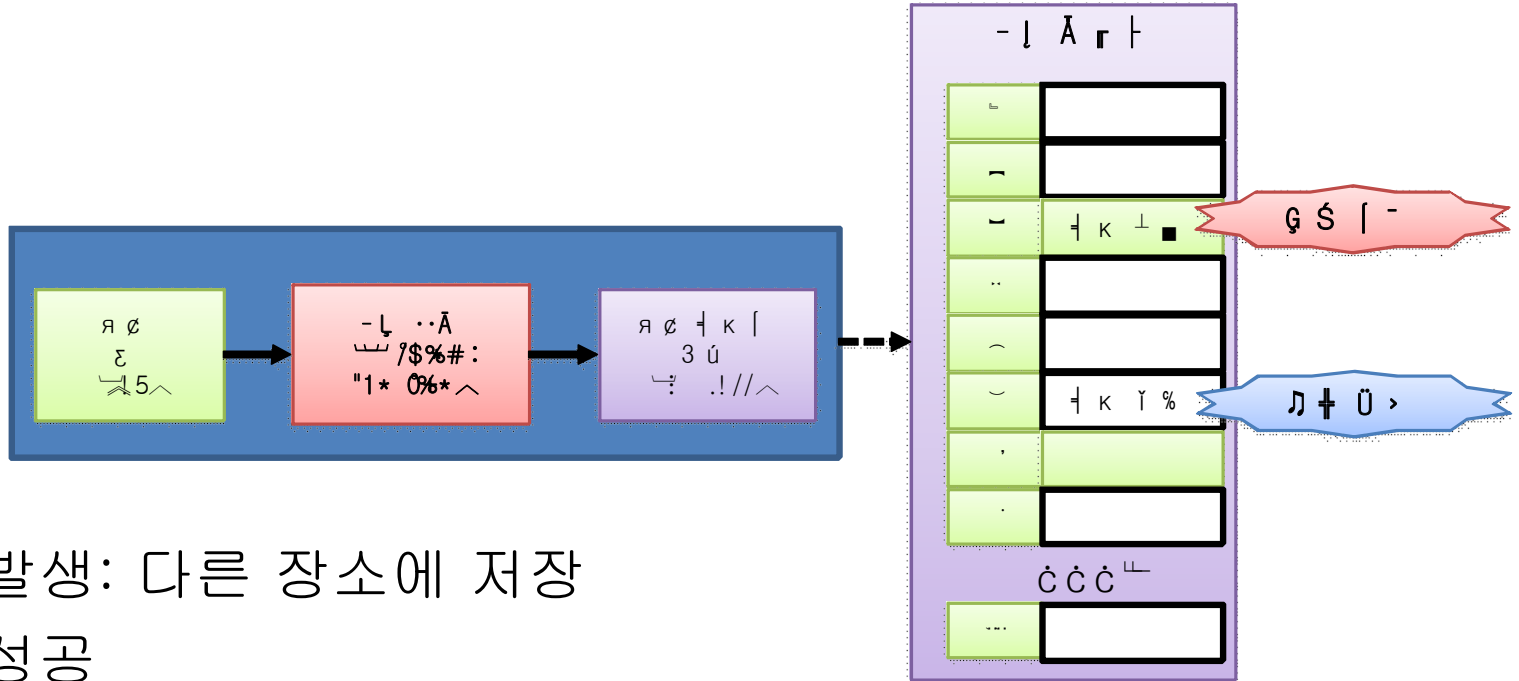


1. 검색 자료의 주소 계산
2. 해시 테이블의 자료 점검
3. 검색 완료

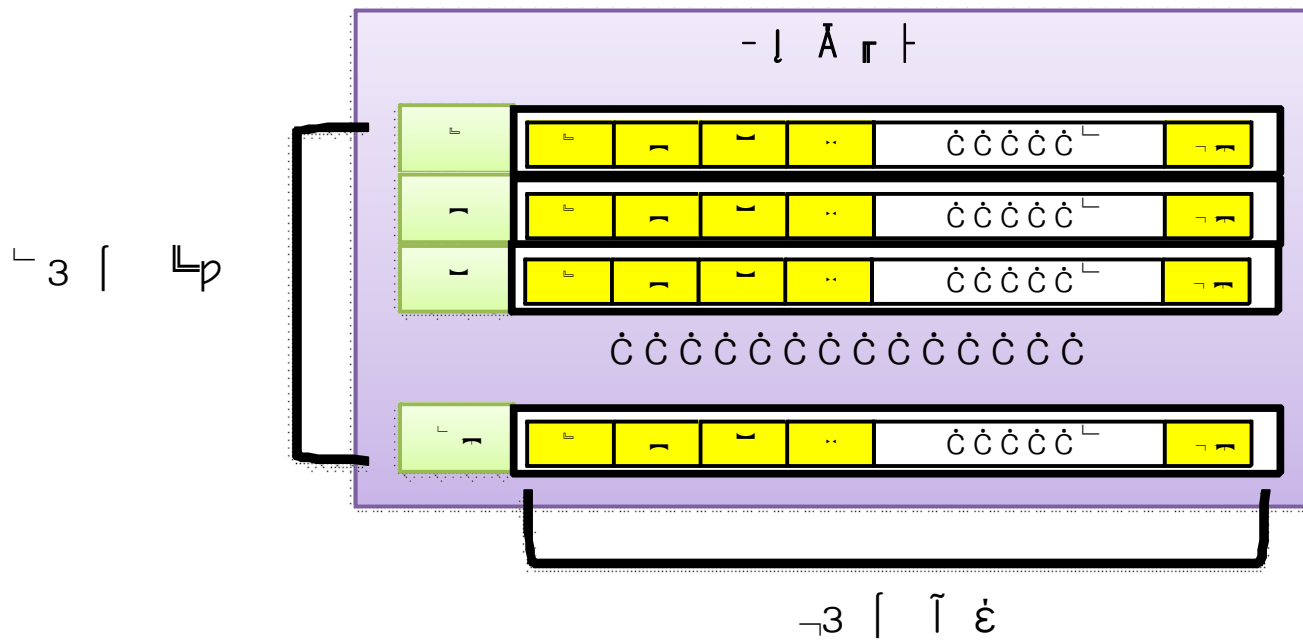
3.2 자료의 저장

- 자료 저장의 경우

3. 해싱



- 충돌 발생: 다른 장소에 저장
- 저장 성공
- 이상적 해싱의 저장 공간 낭비



3.3 해싱 함수 (1/4)

3. 해싱

- 좋은 해싱 함수의 조건
 - 충돌 발생 빈도
 - 해시 테이블 사용률
 - 해싱 함수 계산
- 예
 - 나머지(제산: Divide) 함수
 - 검색 키 값 k : 125
 - 해시 테이블 크기 M : 100

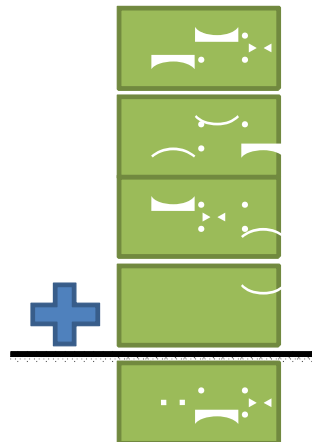
3.3 해싱 함수 (2/4)

3. 해싱

- 접기(접지: Folding) 함수
 - 이동 접기 함수
 - 검색 키 값 k: 1234512345 (10자리)
 - 해시 테이블 크기 M: 999 (3자리)



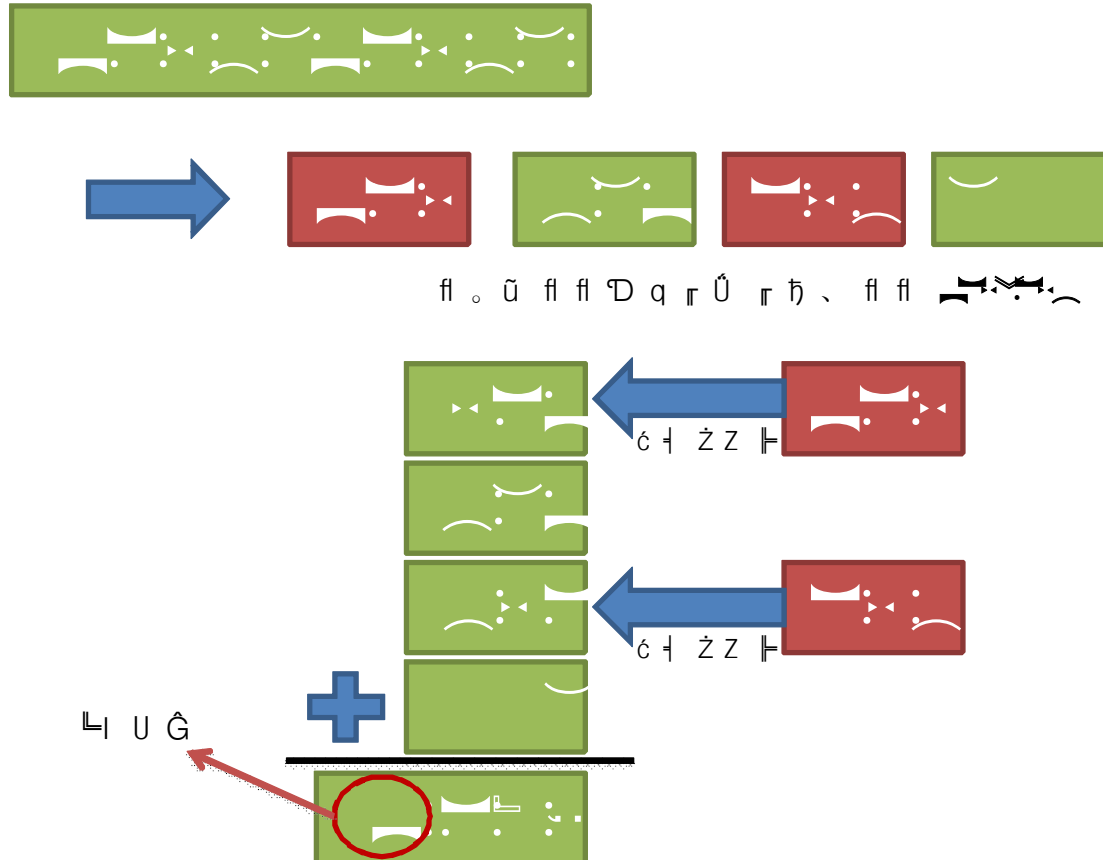
'A -[+ И Ā ǧ ŵ X 3 X fl 。 ū



3.3 해싱 함수 (3/4)

3. 해싱

- 접기(접지: Folding) 함수
 - 경계 접기 함수
 - 검색 키 값 k: 1234512345 (10자리)
 - 해시 테이블 크기 M: 999 (3자리)



3.3 해싱 함수 (4/4)

3. 해싱

- 중간 제곱(Mid-Square) 함수
 - 검색 키 값 k: 9451
 - $9451 * 9451 = 89\underline{3214}01$
- 숫자 분석 방법
 - 검색 키 값: 학번 2010-9025
- 검색 키가 문자열인 경우, 예) ABC
 - 첫 번째 문자 이용
 - 단순 더하기
 - 위치 고려, 호너의 방법(Honor's Method)
 - 예) $ABC \rightarrow ((65 * 31) + 66) * 31 + 67 = 64578$
 $CBA \rightarrow ((67 * 31) + 66) * 31 + 65 = 66498$

3.4 첫 번째 충돌 해결 방법: 개방 주소법 (1/4)

3. 해싱

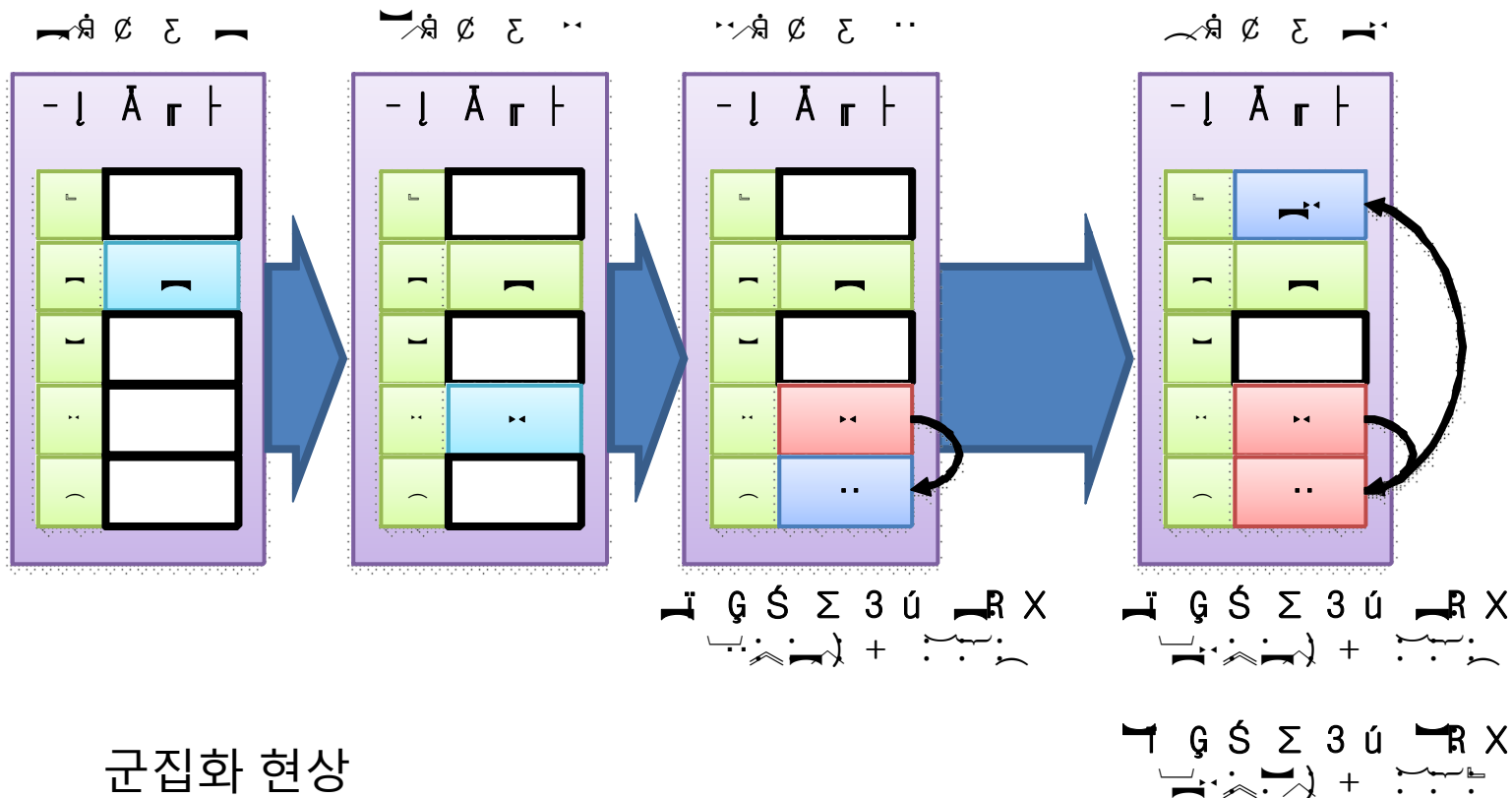
- 충돌 해결 방법
 - 개방 주소법(Open Addressing)
 - 3가지
 - 1) 선형 조사법(Linear Probing)
 - 2) 제곱 조사법(Quadratic Probing)
 - 3) 이중 해싱(Double Hashing)
 - 체이닝(Chaining)
 - 동거자
- 예
 - 해시 테이블 크기 5
 - 해싱 함수: 나머지 함수 mod 5
 - 저장되는 검색 키 값 {1, 3, 8, 13}

3.4 첫 번째 충돌 해결 방법: 개방 주소법 (2/4)

3. 해싱

- 선형 조사법

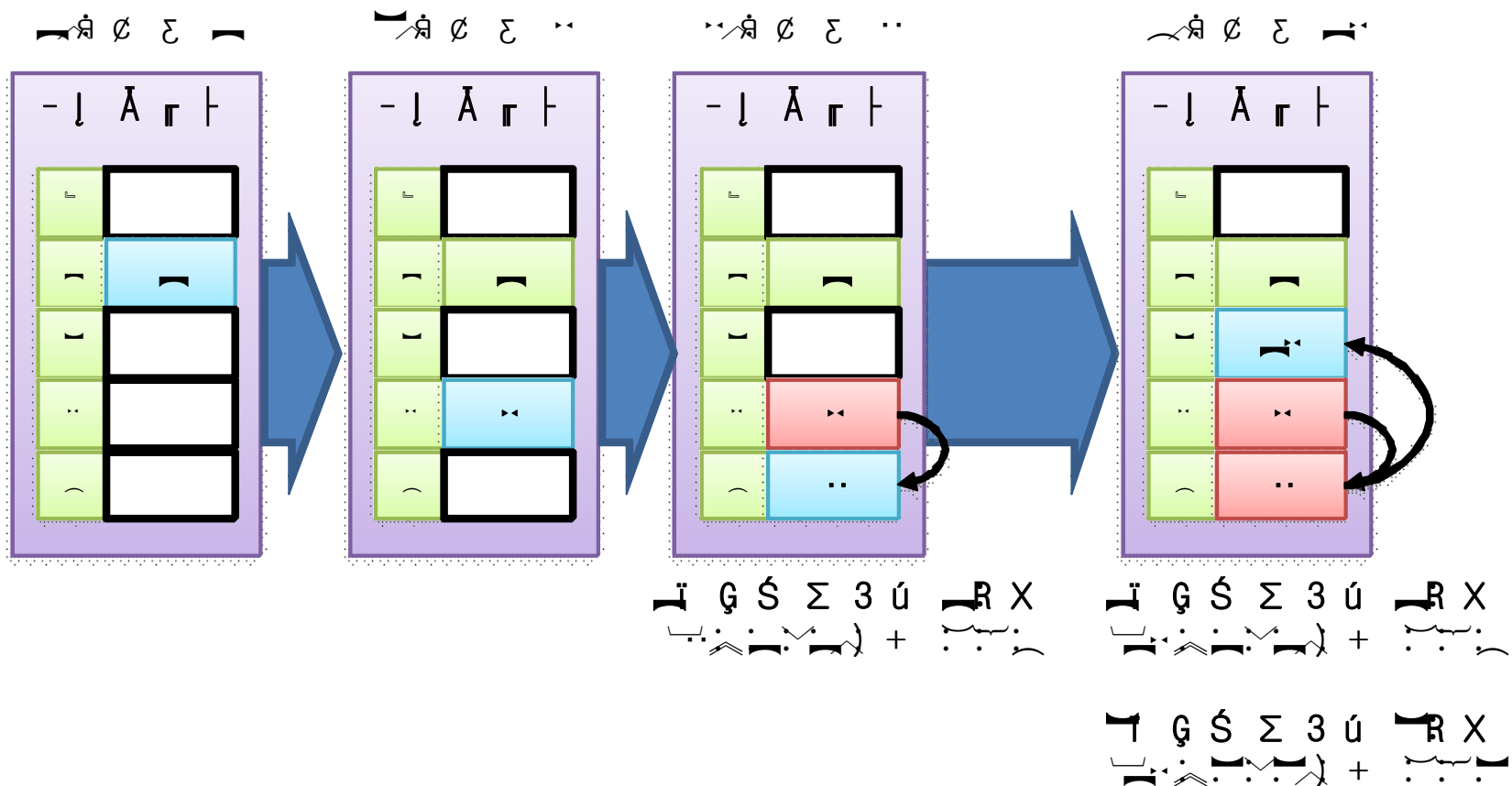
- 일정한 상수를 증가시켜 다시 조사, 군집화 현상 발생가능성 큼
- $h(k) = (k + \text{try_count}) \bmod M$
- 저장되는 검색 키 값 $\{1, 3, 8, 13\}$



3.4 첫 번째 충돌 해결 방법: 개방 주소법 (3/4)

3. 해싱

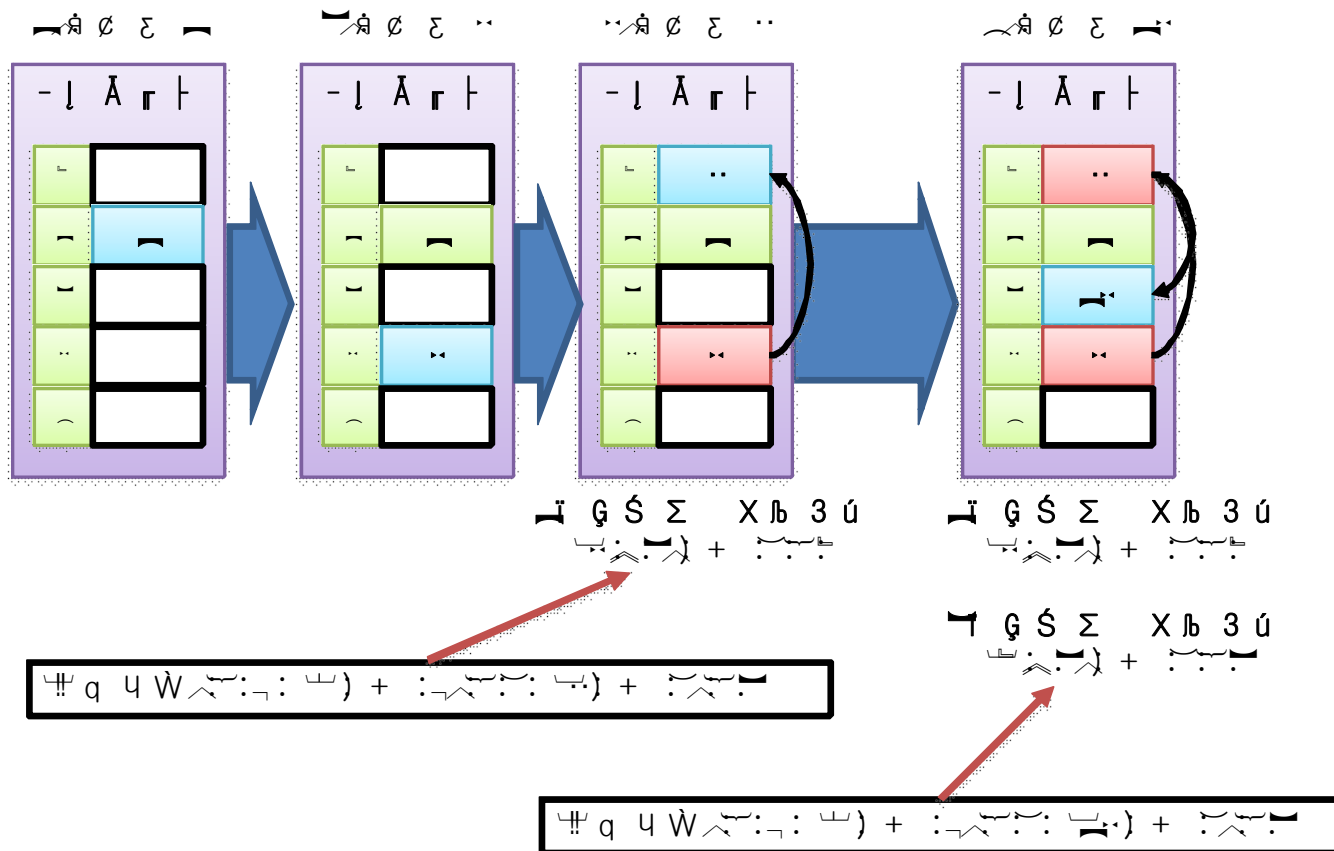
- 제곱 조사법
 - 조사 횟수의 제곱만큼 증가
 - $h(k) = (k + \text{try_count} * \text{try_count}) \bmod M$
 - 저장되는 검색 키 값 {1, 3, 8, 13}



3.4 첫 번째 충돌 해결 방법: 개방 주소법 (4/4)

3. 해싱

- 이중 해싱
 - 원래의 해싱 함수와는 다른 추가적인 해싱 함수 이용
 - $h(k) = (k + \text{조사간격}) \bmod M$
 - $(\text{조사 간격}) = M - (k \bmod M)$



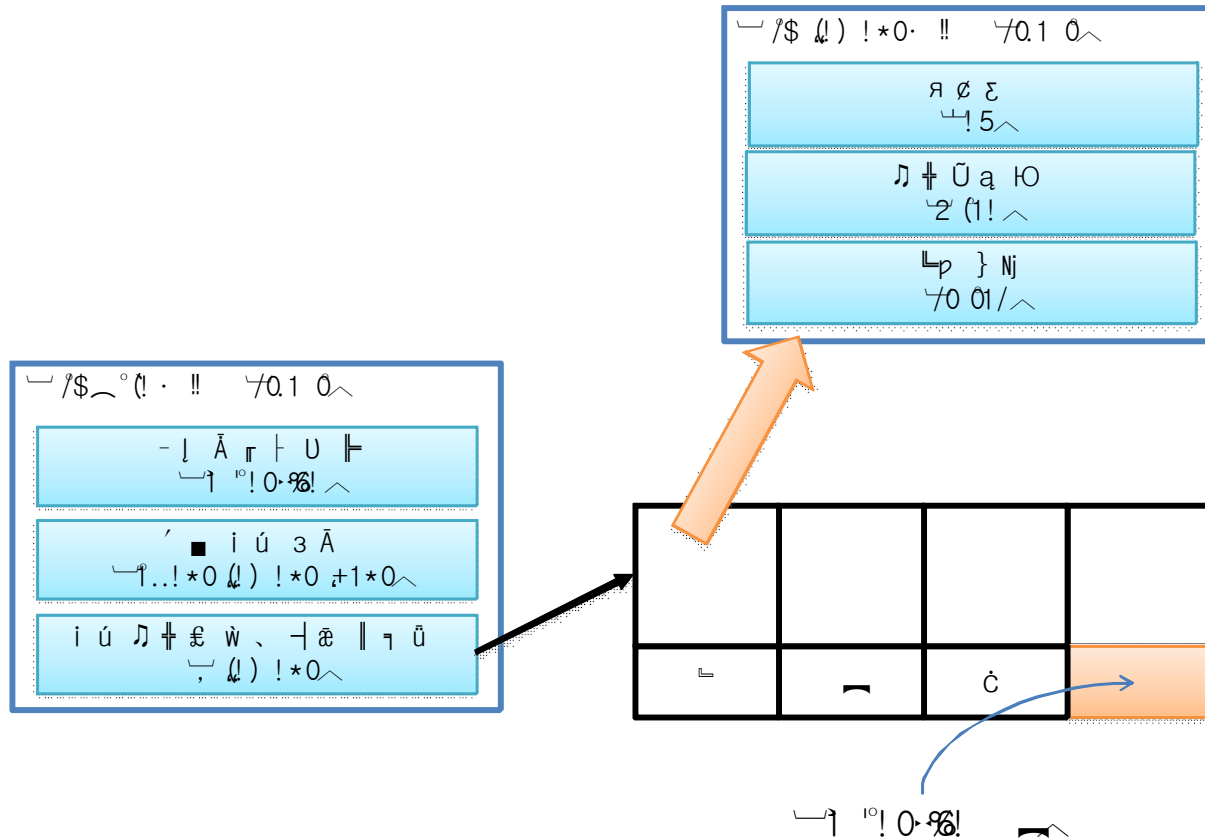
3.5 추상 자료형

- 해시 테이블 생성
- 해시 테이블 삭제
- 자료 추가
- 자료 제거
- 자료 검색
- 자료 개수

3. 해싱

3.6 첫 번째 구현: 개방 주소법 사용

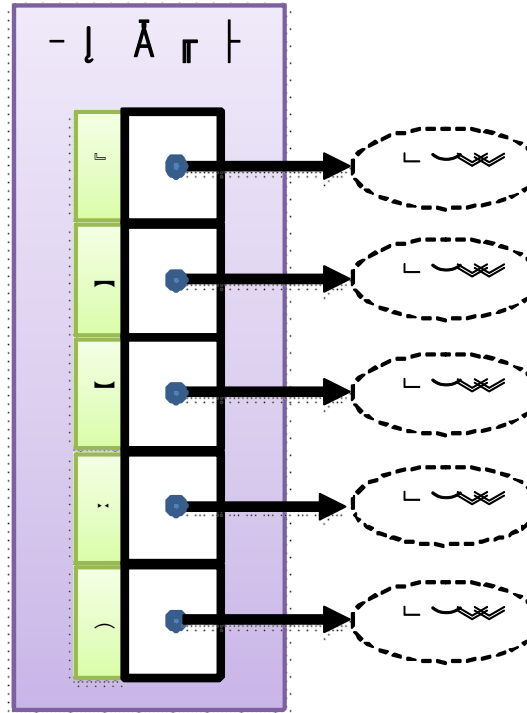
3. 해싱



3.7 두 번째 충돌 해결 방법: 체이닝

3. 해싱

- 체이닝
 - 연결리스트 사용

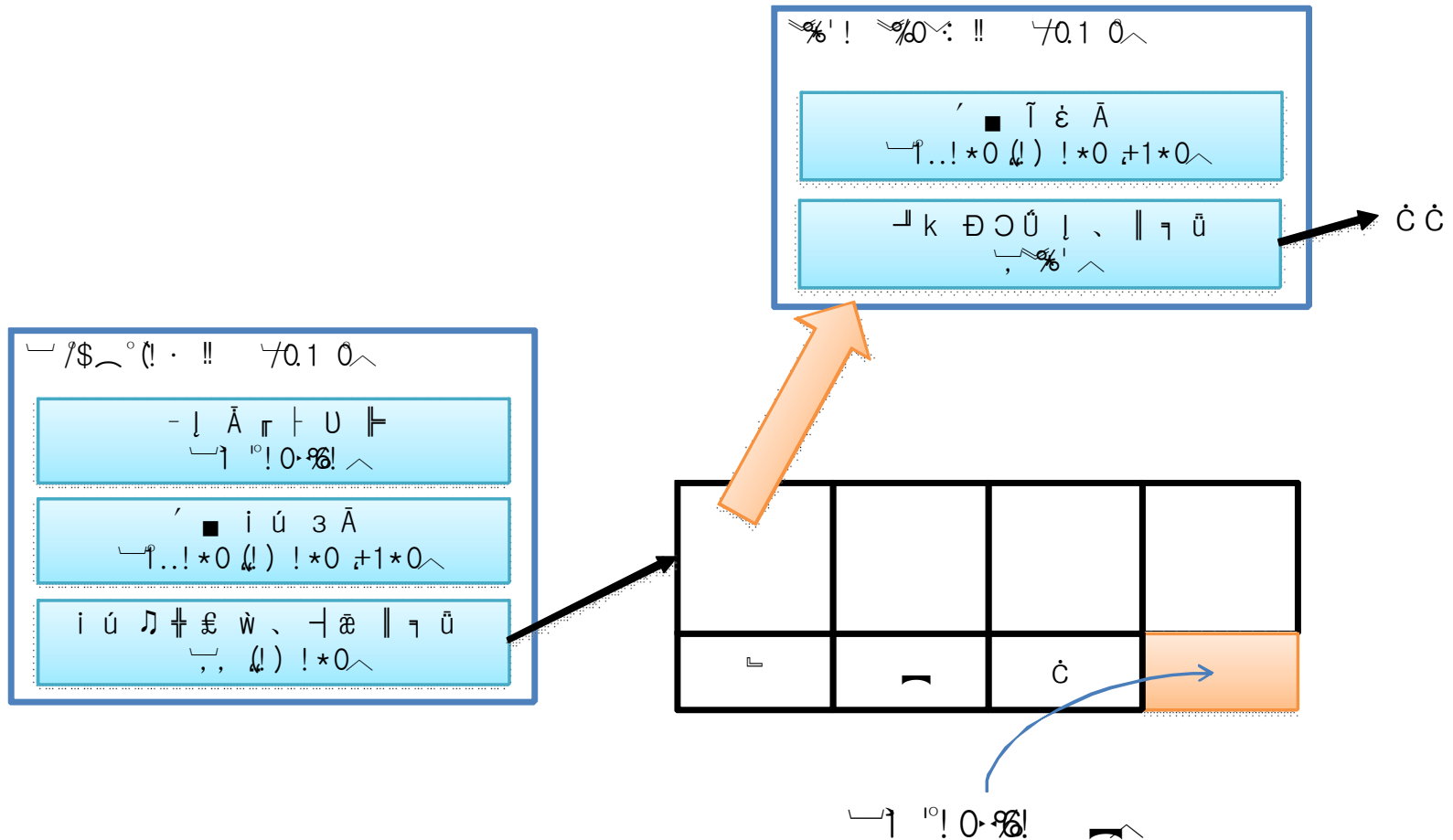


- 예
 - 해싱 함수: 나머지 함수 이용 (mod 5)
 - 저장되는 검색 키 값 {1, 8, 3}

3.8 두 번째 구현: 체이닝 (2/2)

- 해싱 테이블에 대한 구조체 HashTable

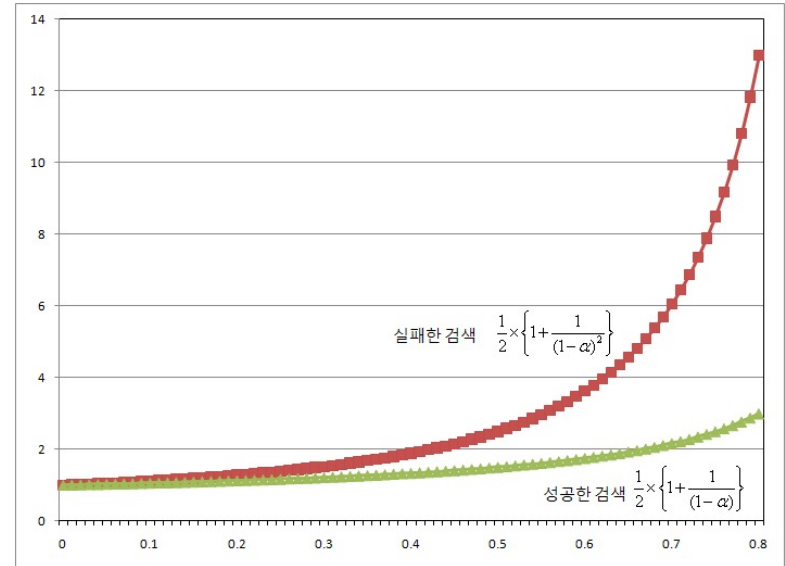
3. 해싱



3.9 해싱의 성능 분석

3. 해싱

- 해싱의 성능
 - 최선의 경우
 - 최악의 경우
- 키 값 밀도
- 적재 밀도



$$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right) \quad \alpha$$

$$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)} \right) \quad 1 + \frac{\alpha}{2}$$

이진 탐색 트리란?

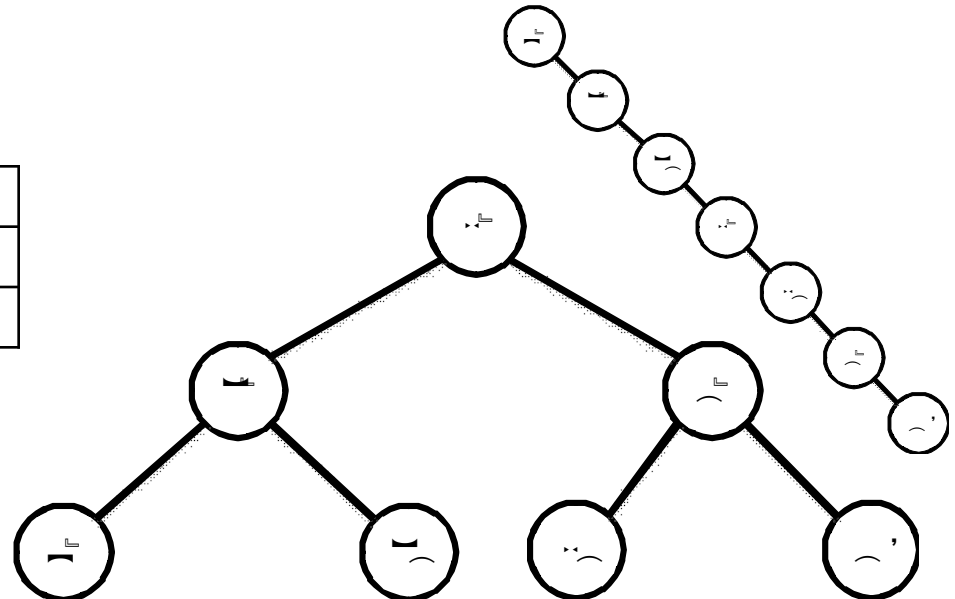
4. 균형 이진 탐색 트리

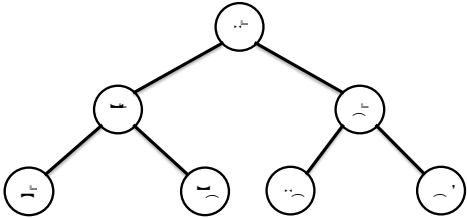
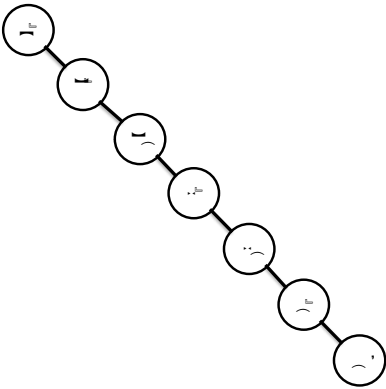
- 특징

1. 트리의 모든 노드는 유일한 키(key)를 가진다.
2. 왼쪽 서브트리에 있는 모든 노드의 키는 루트의 키보다 작다.
3. 오른쪽 서브트리에 있는 모든 노드의 키는 루트의 키보다 크다.
4. 왼쪽 서브트리와 오른쪽 서브트리도 모두 이진 탐색 트리이다.

- 검색의 시간 복잡도

	검색의 시간 복잡도
균형 트리	$O(\log_2 n)$
불균형 트리	$O(n)$

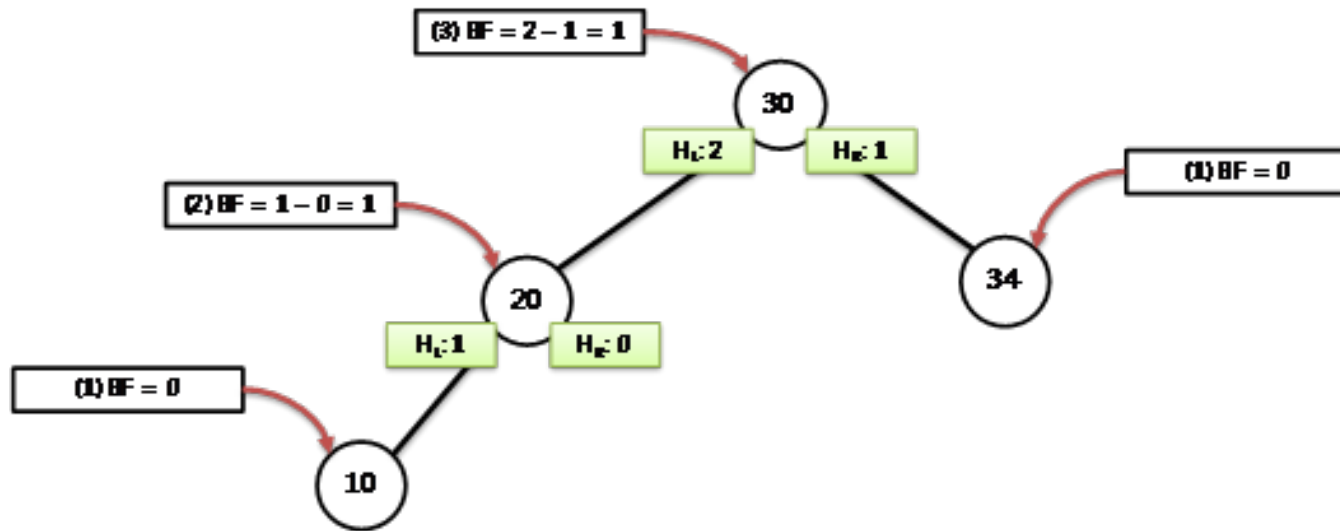


	(a) 균형 이진 탐색 트리	(b) 불균형 이진 탐색 트리
자료 추가 순서	30→20→40→10→24→34→46	10→20→24→30→34→40→46
생성된 이진 탐색 트리	 <pre> graph TD 30((30)) --- 20((20)) 30 --- 40((40)) 20 --- 10((10)) 20 --- 24((24)) 40 --- 34((34)) 40 --- 46((46)) </pre>	 <pre> graph TD 10((10)) --- 20((20)) 20 --- 24((24)) 24 --- 30((30)) 30 --- 34((34)) 34 --- 40((40)) 40 --- 46((46)) </pre>
최대 비교 횟수	3회 예) 검색 키 46 검색	7회 예) 검색 키 46 검색
평균 비교 횟수	$\frac{1+2+2+3+3+3+3}{7} = 2.43$	$\frac{1+2+3+4+5+6+7}{7} = 4$

4.1. AVL 균형 이진 탐색 트리 (1/2)

4. 균형 이진 탐색 트리

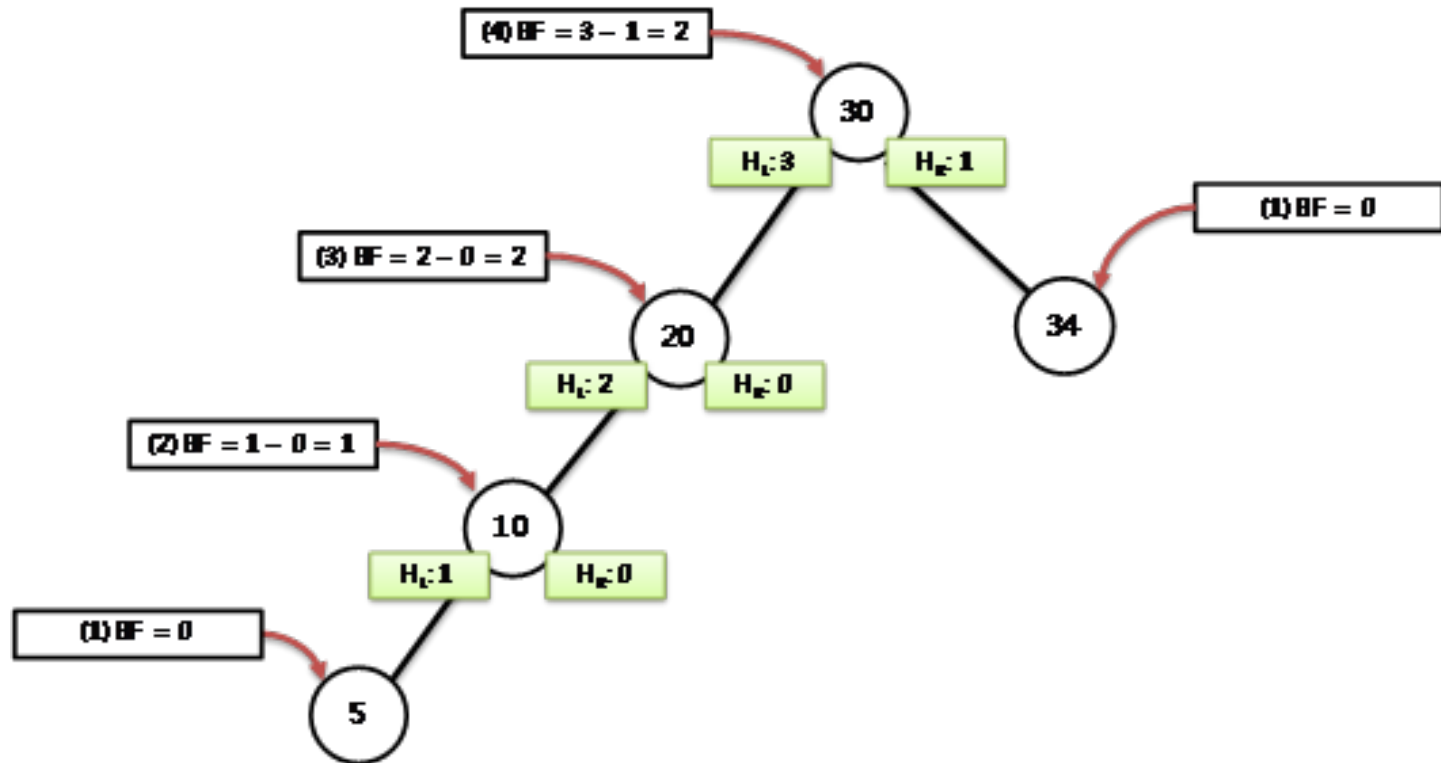
- 제약조건
 - 균형 인수 BF(Balance Factor): 왼쪽 서브트리의 높이에서 오른쪽 서브트리의 높이를 뺀 값
 - $|H_L - H_R| \leq 1$



4.1. AVL 균형 이진 탐색 트리 (2/2)

4. 균형 이진 탐색 트리

- 균형인수 (계속)



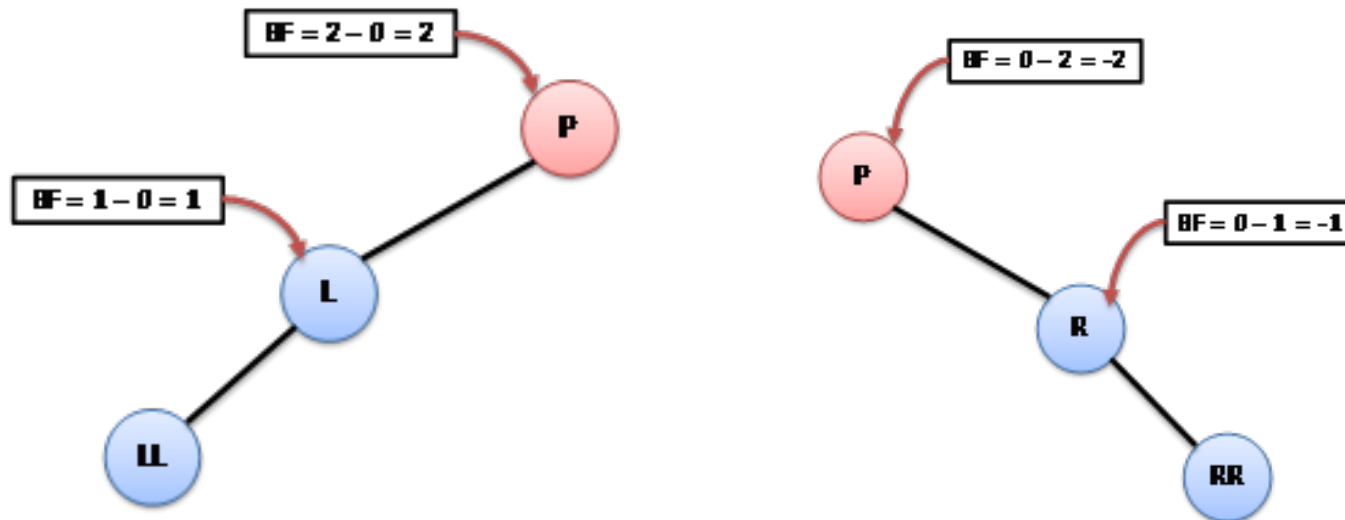
4.2 AVL 균형 유지 연산 (1/6)

4. 균형 이진 탐색 트리

- AVL 트리의 균형이 깨지는 4가지 경우

(1) LL (Left-Left)

(2) RR (Right-Right)



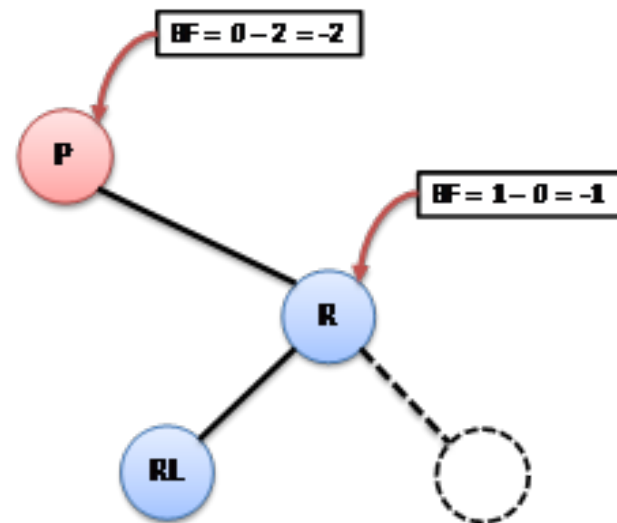
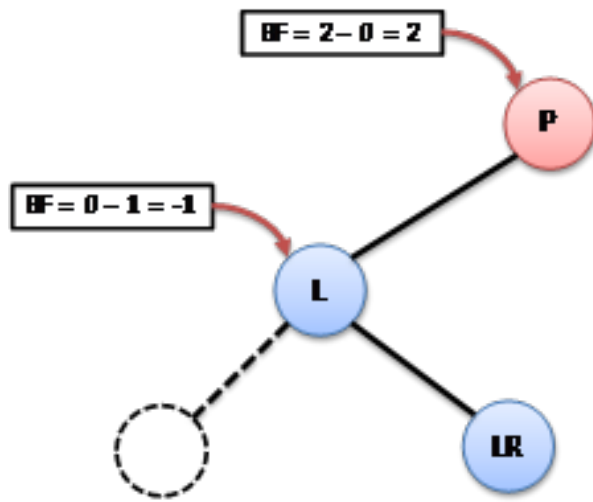
4.2 AVL 균형 유지 연산 (2/6)

4. 균형 이진 탐색 트리

- AVL 트리의 균형이 깨지는 4가지 경우 (계속)

(3) LR (Left-Right)

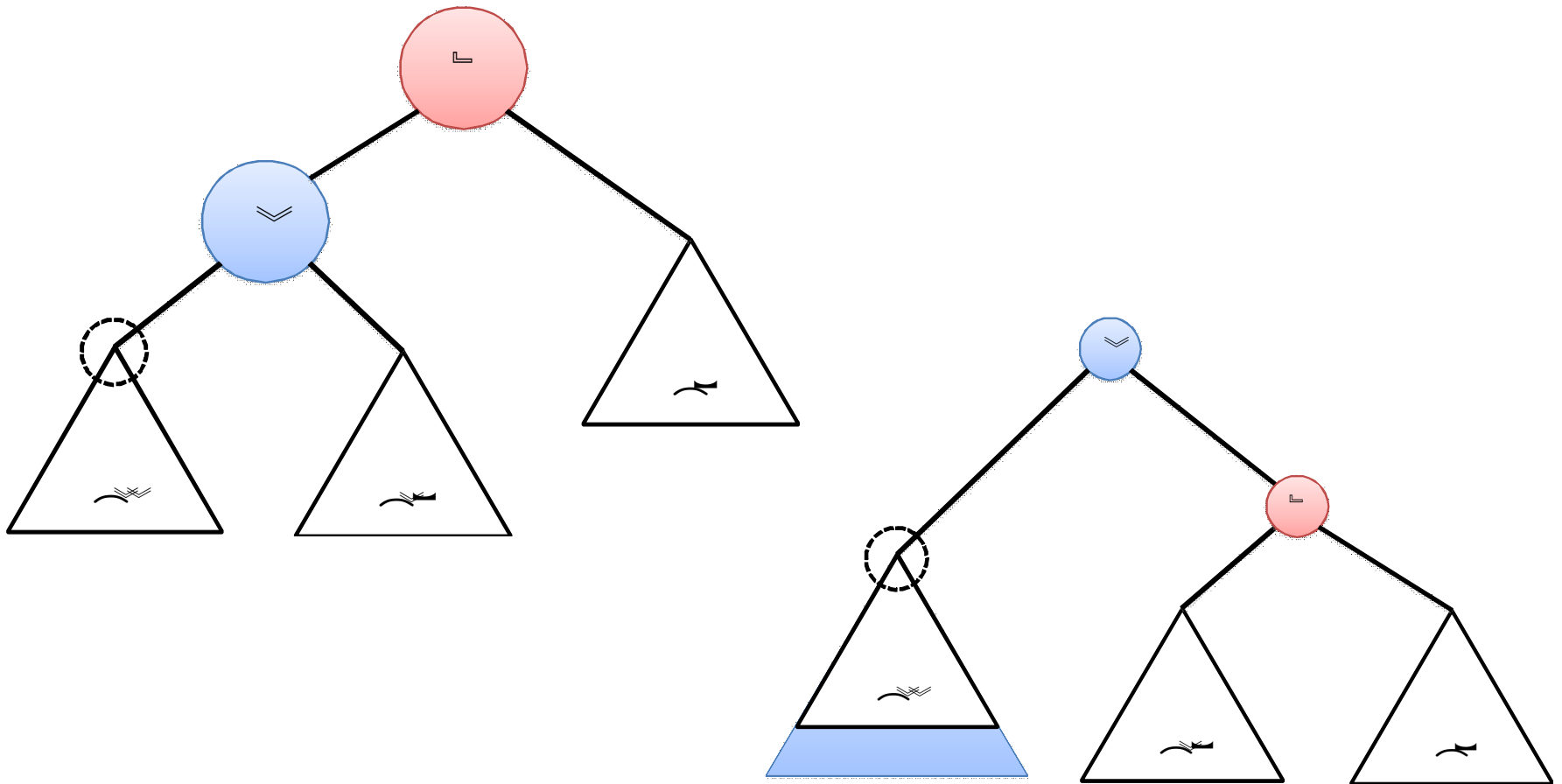
(4) RL (Right-Left)



4.2 AVL 균형 유지 연산 (3/6)

4. 균형 이진 탐색 트리

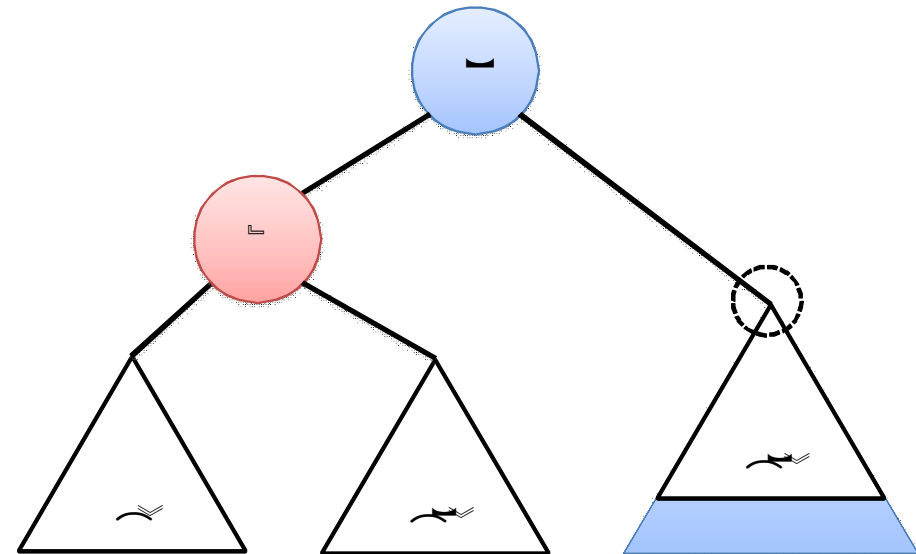
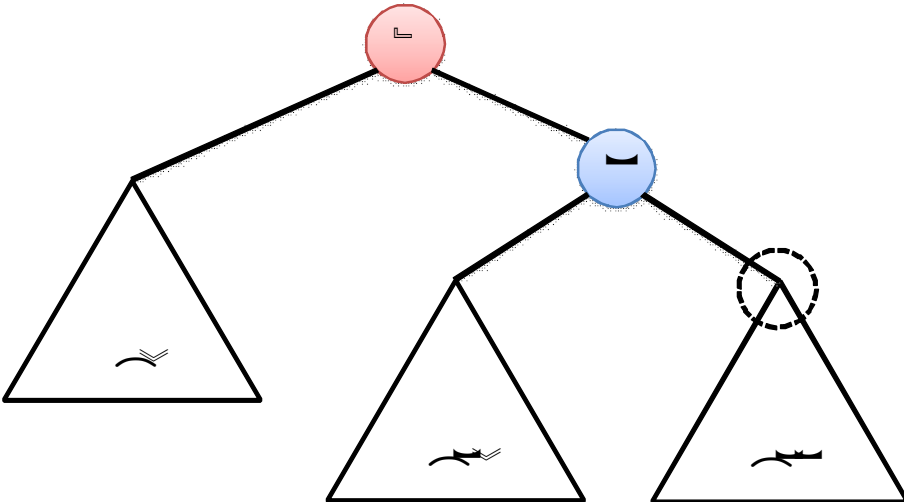
- LL의 경우: LL회전



4.2 AVL 균형 유지 연산 (4/6)

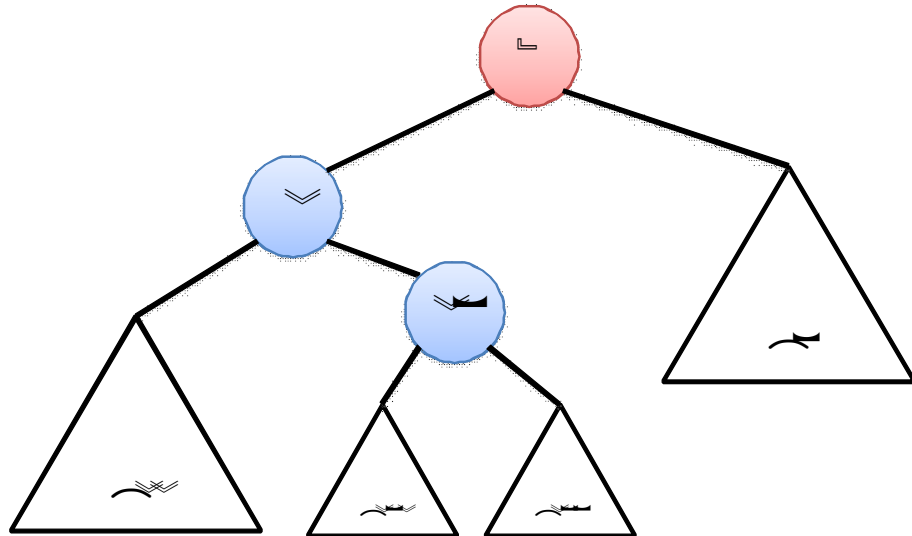
4. 균형 이진 탐색 트리

- RR의 경우: RR회전

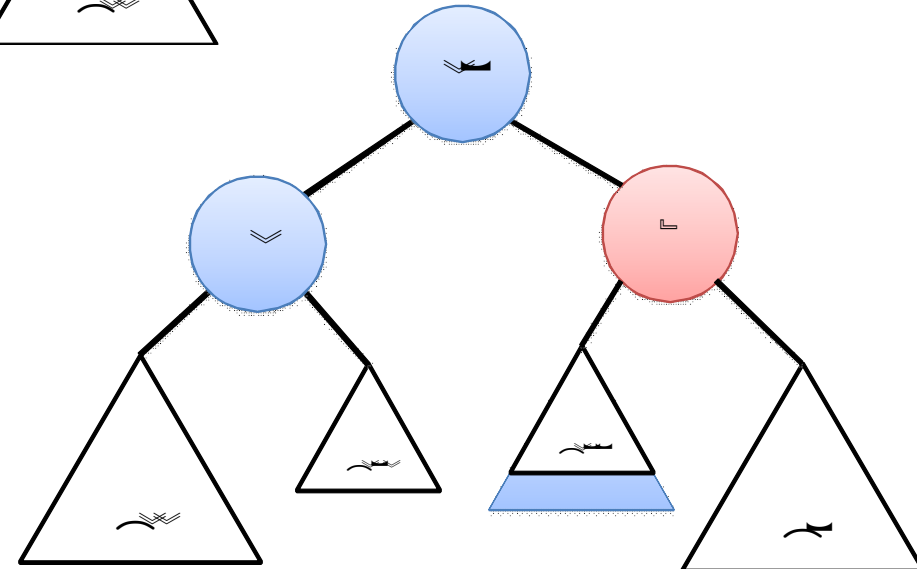
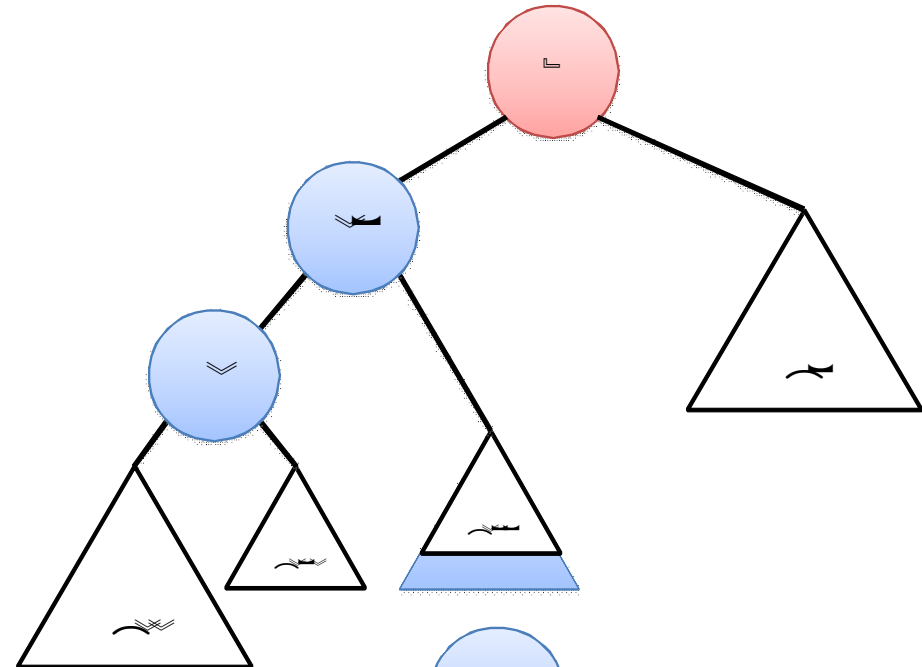


4.2 AVL 균형 유지 연산 (5/6)

- LR의 경우: LR회전



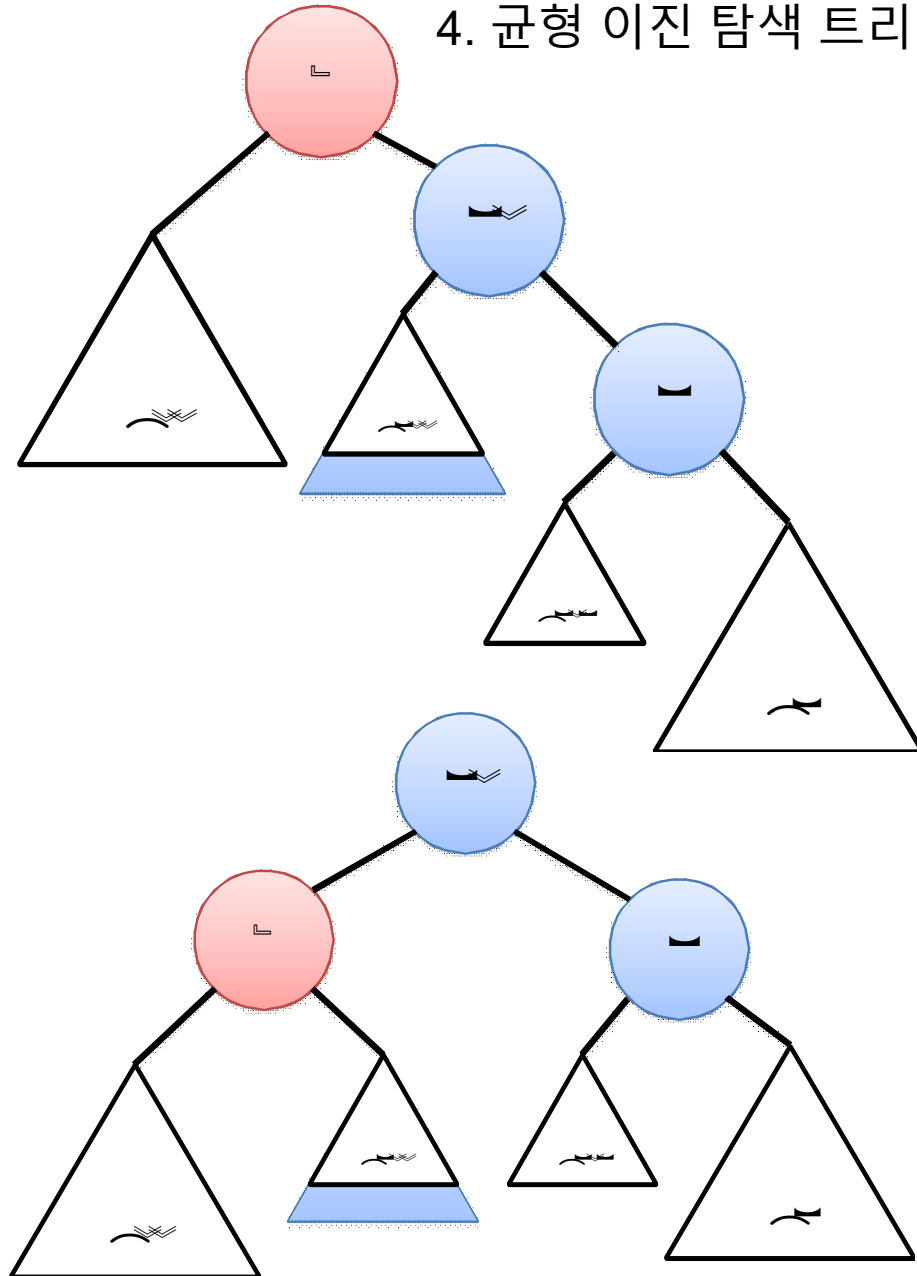
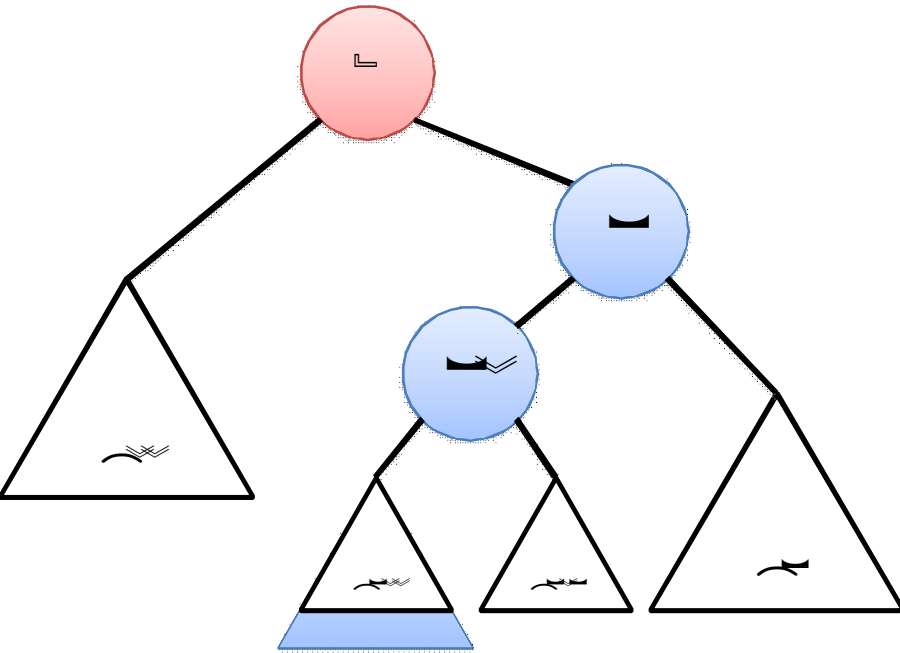
4. 균형 이진 탐색 트리



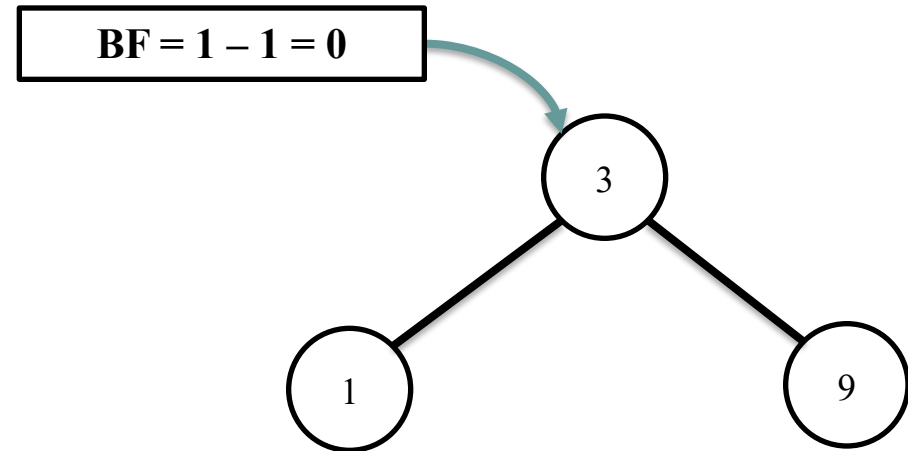
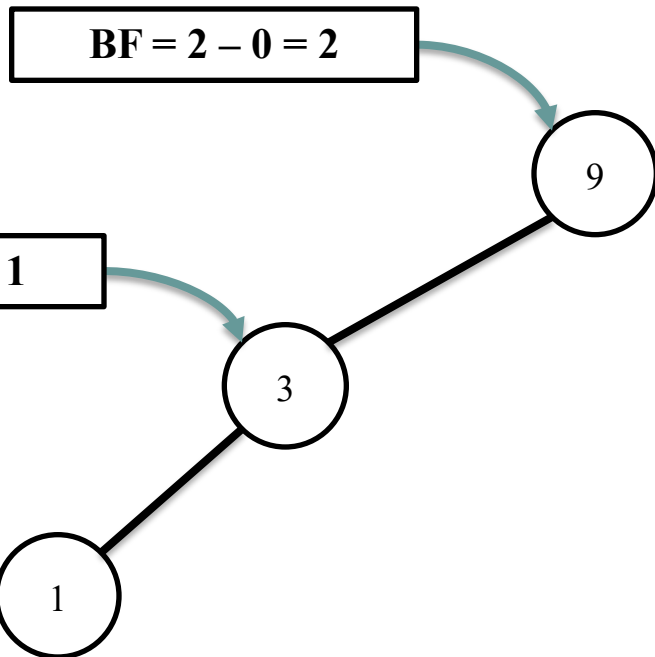
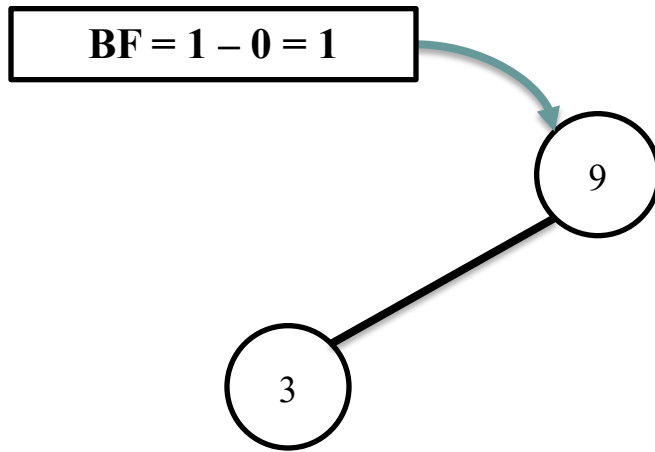
4.2 AVL 균형 유지 연산 (6/6)

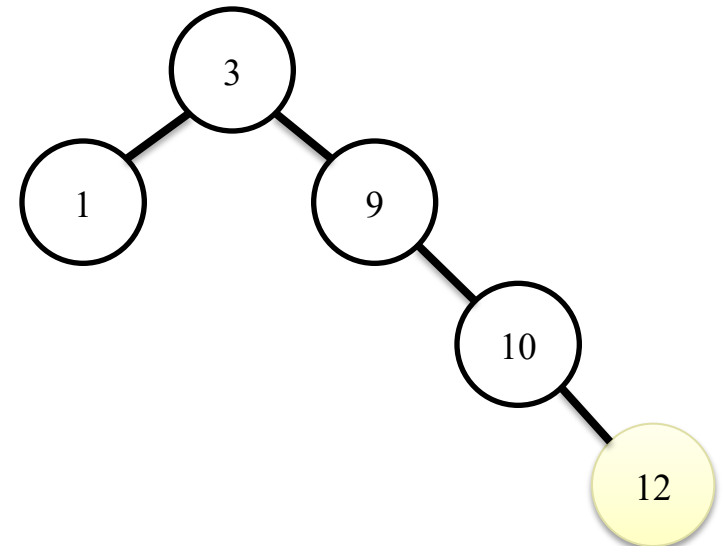
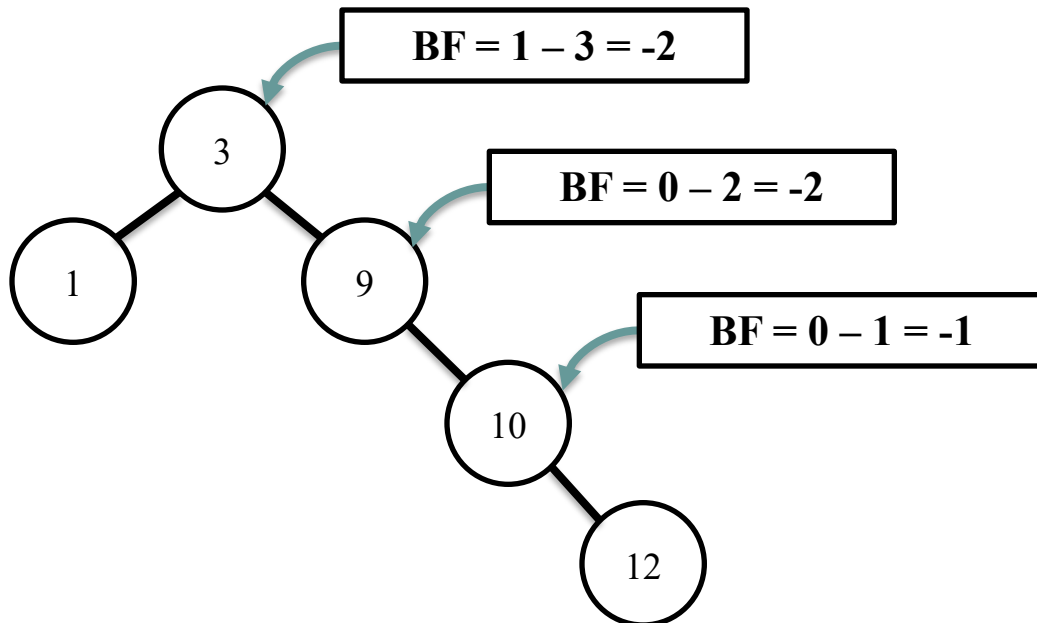
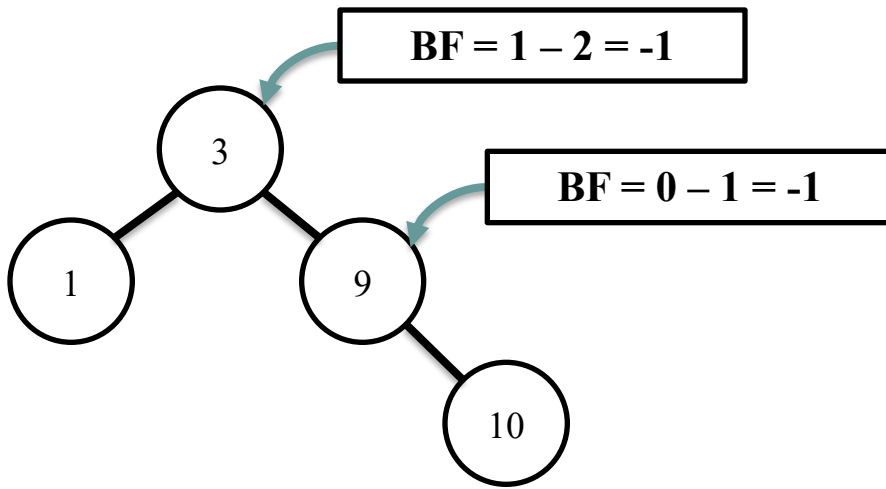
- RL의 경우: RL회전

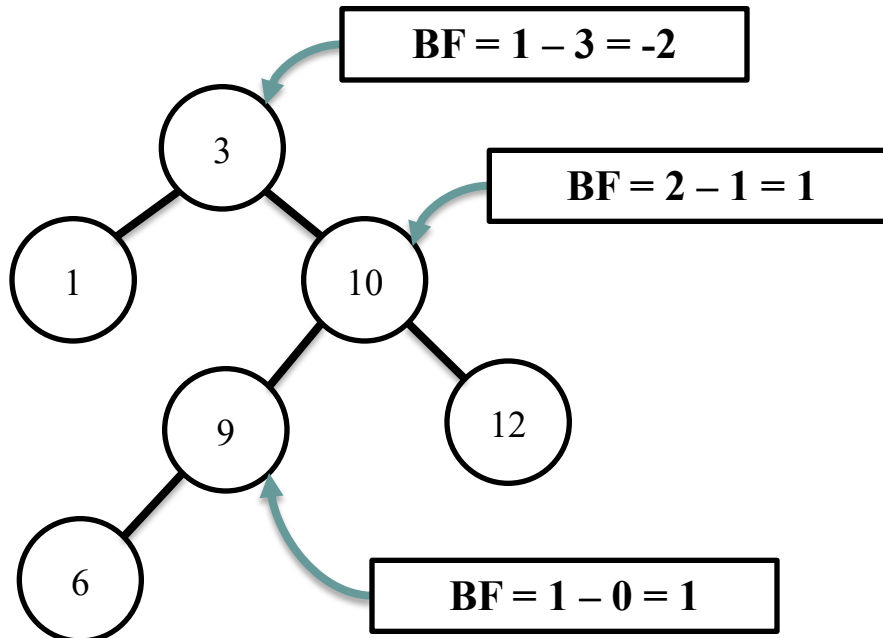
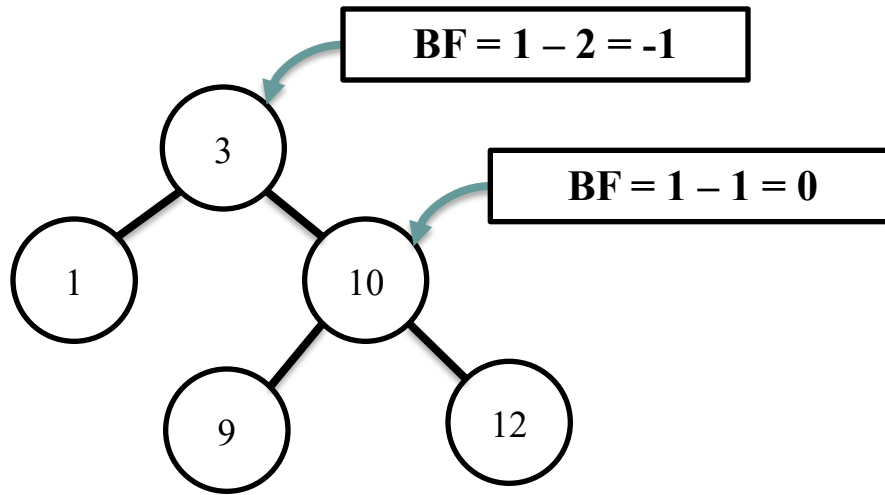
4. 균형 이진 탐색 트리

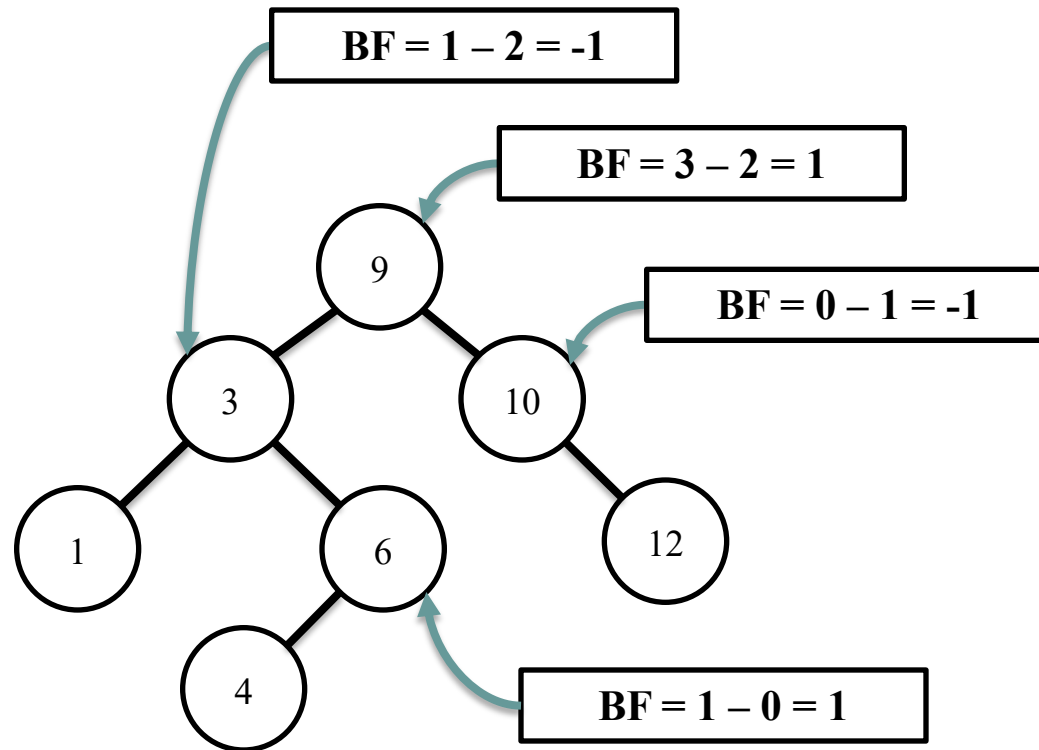
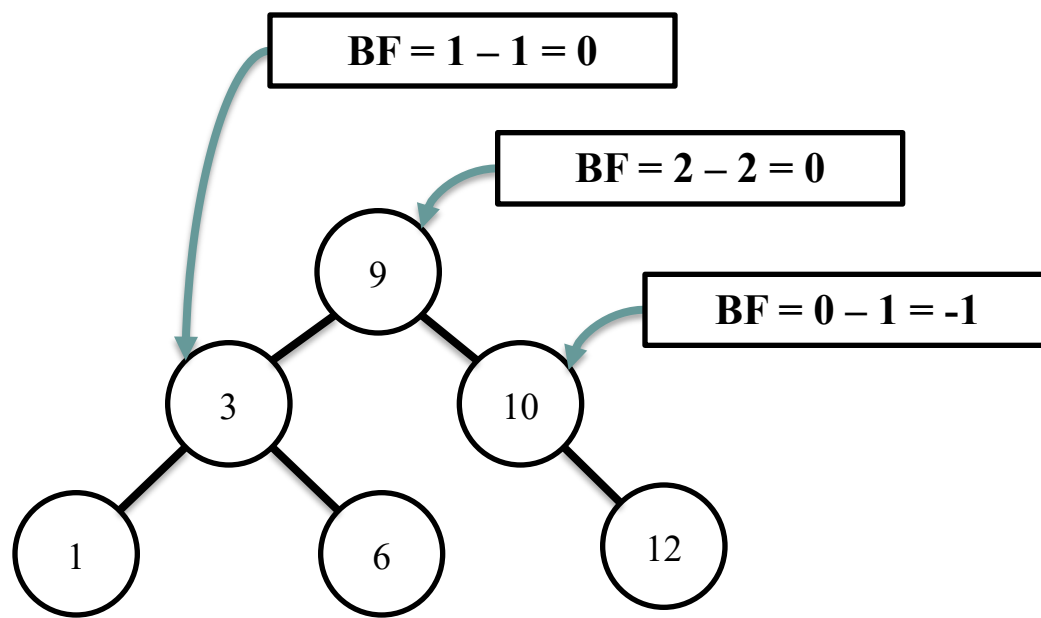


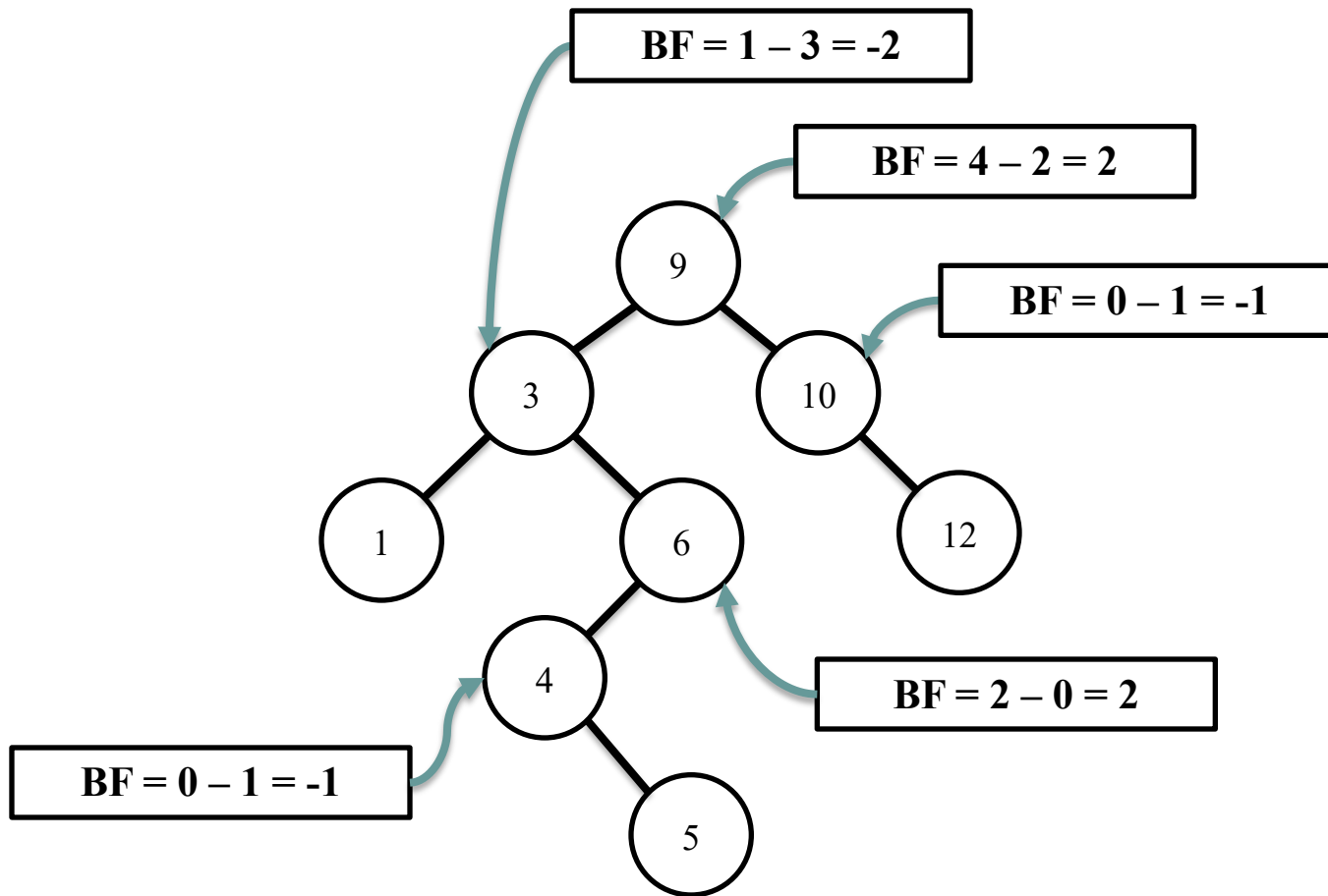
		pChildNode의 balance	
		> 0	< 0
pNode의 balance	> 1	LL 회전	LR 회전
	< -1	RL 회전	RR 회전

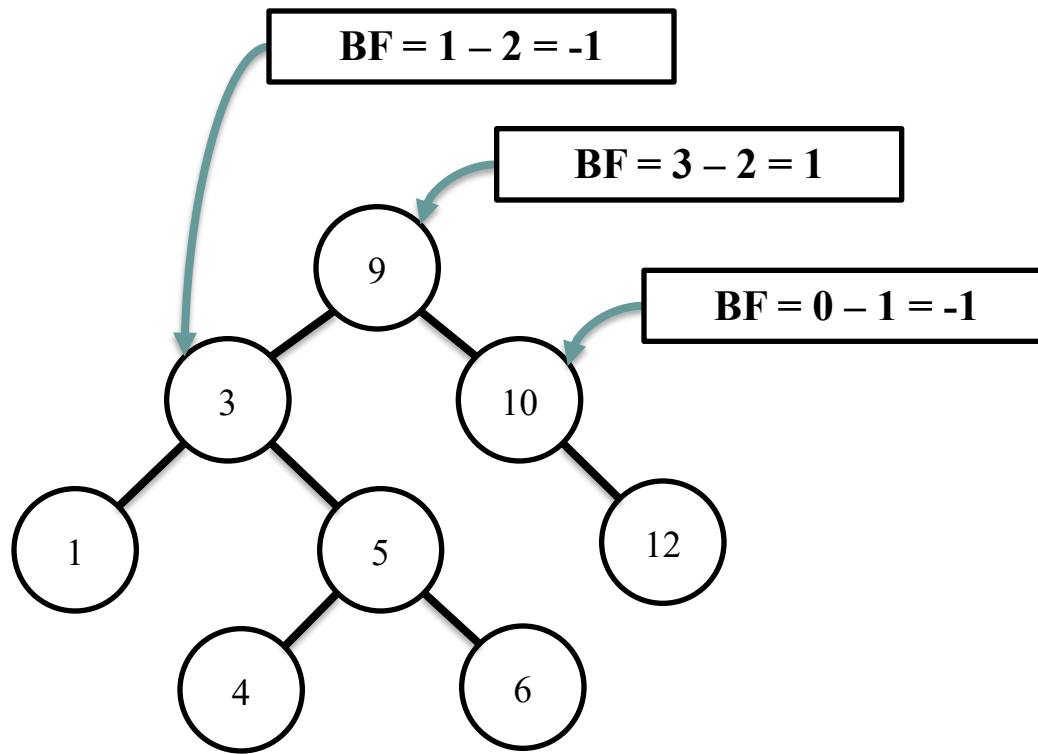












AVL 트리의 추상 자료형 및 구현

- 추상 자료형
 - AVL트리 생성, 삭제
 - 탐색
 - 데이터 추가, 삭제

4. 균형 이진 탐색 트리

※ 4.4 연결 리스트로 구현한 스택

5.1 m-원 탐색 트리

- 다원 탐색 트리

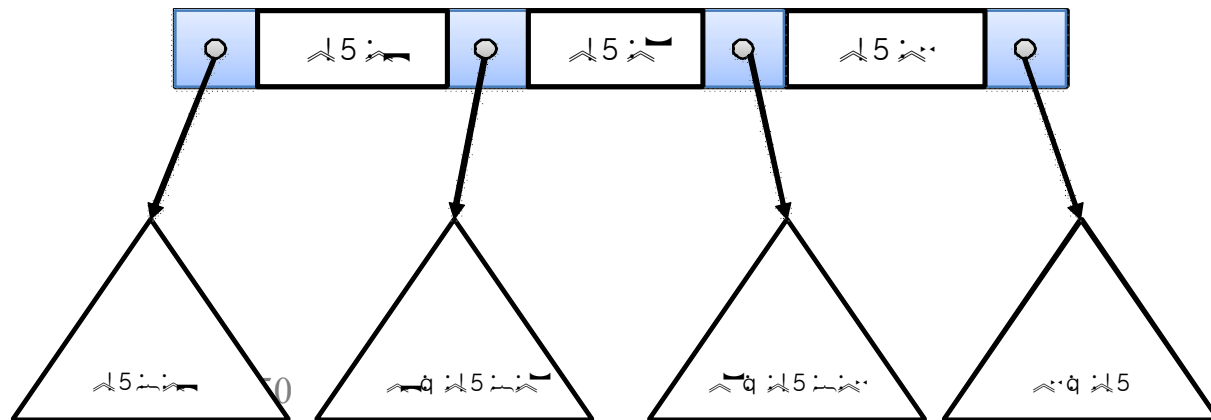
5. 다원 탐색 트리

- 균형 이진 탐색 트리의 문제점

- 1) 높이 문제
- 2) 높은 균형 유지 문제

- m-원 탐색 트리(m-way Search Tree)의 특징

1. 각 노드는 0에서 최대 m 개의 서브트리(sub tree)를 가진다.
2. k개의 서브트리를 가지는 노드는 (k - 1) 개의 자료를 가진다. (단, $k \leq m$)
3. 각 노드 안에서 자료들은 검색 키에 의해 정렬된다.
4. $key\ 1 \leq key\ 2 \leq \dots \leq key\ (k-1)$
5. 다음 조건을 항상 만족한다.
6. $Key\ i \leq (i\text{번째 서브트리 내의 모든 키 값}) < Key\ (i + 1)$



5.2 B-트리

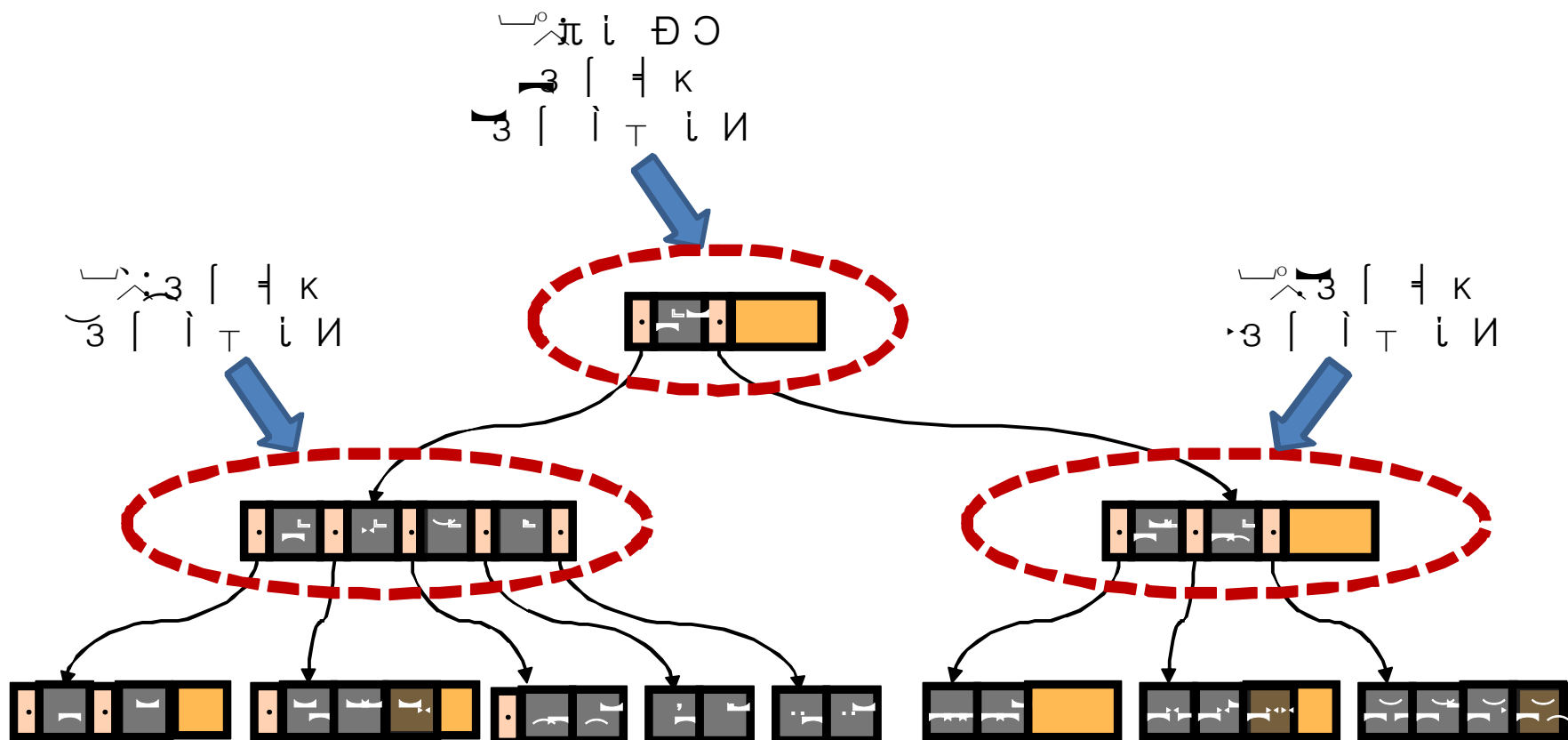
• B-트리의 추가 4가지 특성

5. 다원 탐색 트리

1. 루트 노드는 단말(leaf) 노드이거나, 2에서 m 개의 서브트리를 가진다.
2. 루트 노드를 제외한 모든 내부(internal) 노드는 아래의 개수만큼 서브트리를 가진다.
3. $\lceil m/2 \rceil \leq (\text{서브트리의 개수}) \leq m$
4. 단말 노드는 아래의 개수만큼 자료를 가진다.
5. $\lceil m/2 \rceil - 1 \leq (\text{자료의 개수}) \leq m - 1$
6. 모든 단말 노드는 같은 레벨에 있다. 즉, 트리는 완전한 균형 상태에 있도록 한다.

	서브트리의 개수		자료의 개수	
	최소	최대	최소	최대
3	2	3	1	2
4	2	4	1	3
5	3	5	2	4
m	$\lceil m/2 \rceil$	m	$\lceil m/2 \rceil - 1$	$m - 1$

- $m=5$

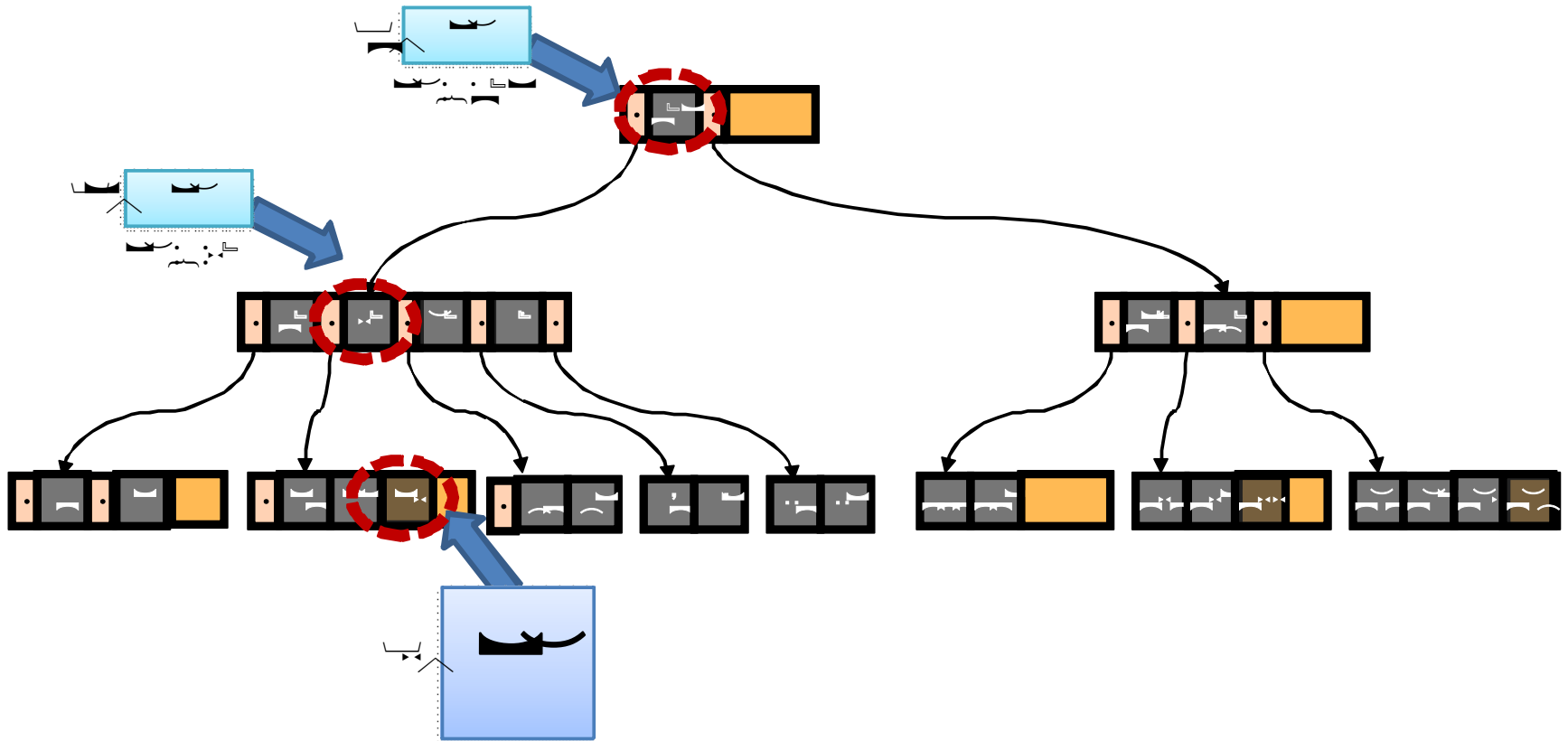


5.3 B-트리에서의 자료 추가 (1/3)

- (1) 저장 위치 찾기
- (2) 자료의 저장과 노드 분할

5. 다원 탐색 트리

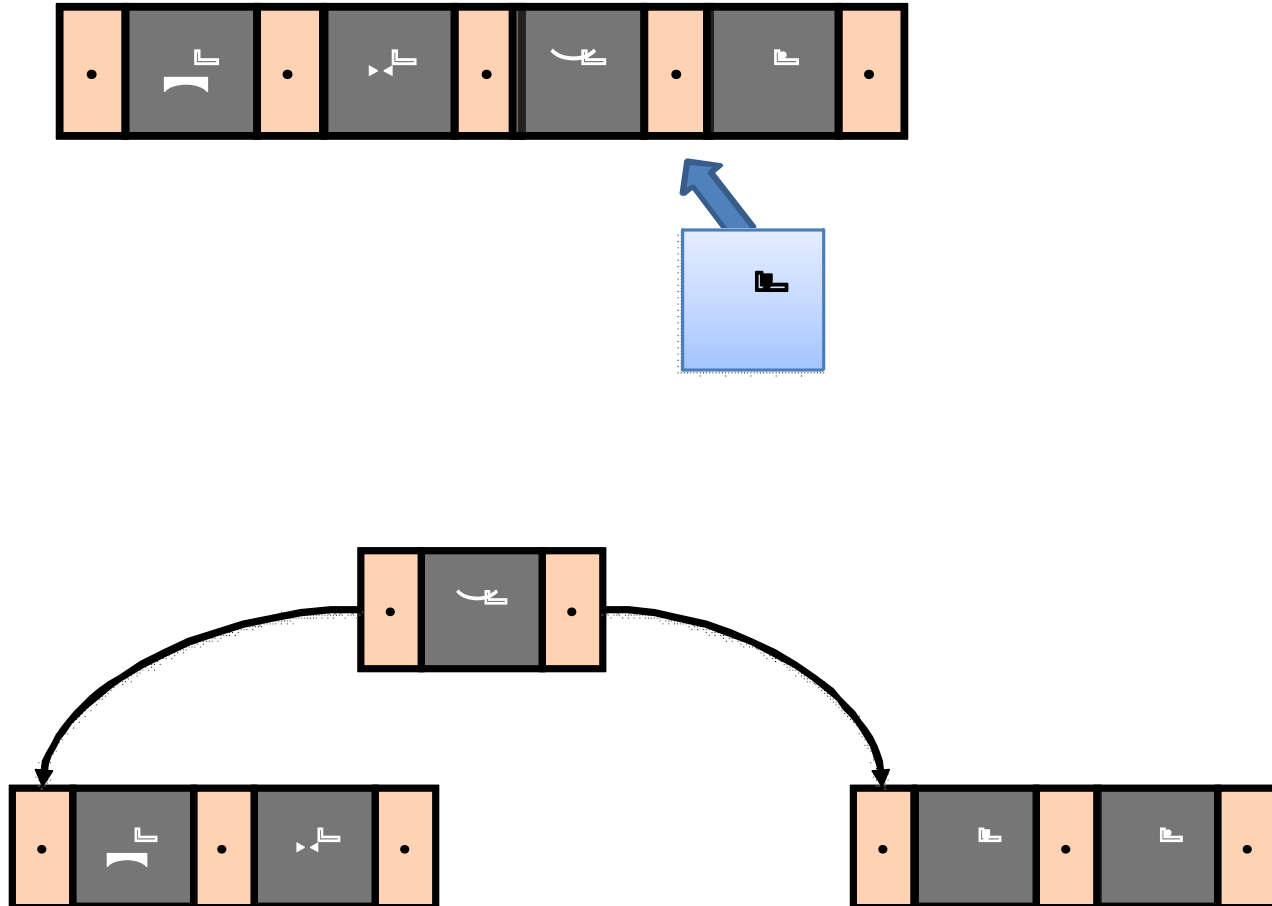
Key $i \leq$ (i번째 서브트리 내의 모든 키 값) $<$ Key $(i + 1)$



5.3 B-트리에서의 자료 추가 (2/3)

- (2)자료의 저장과 노드 분할 - 첫 번째 예
 - $m=5$, 레벨이 1인 B-트리

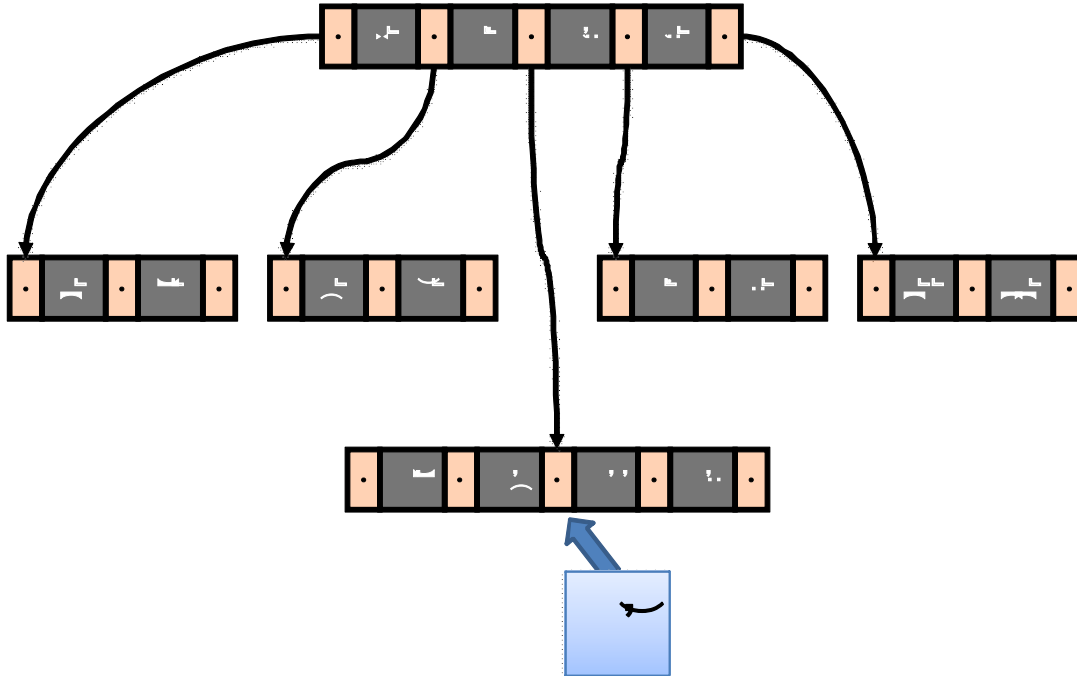
5. 다원 탐색 트리



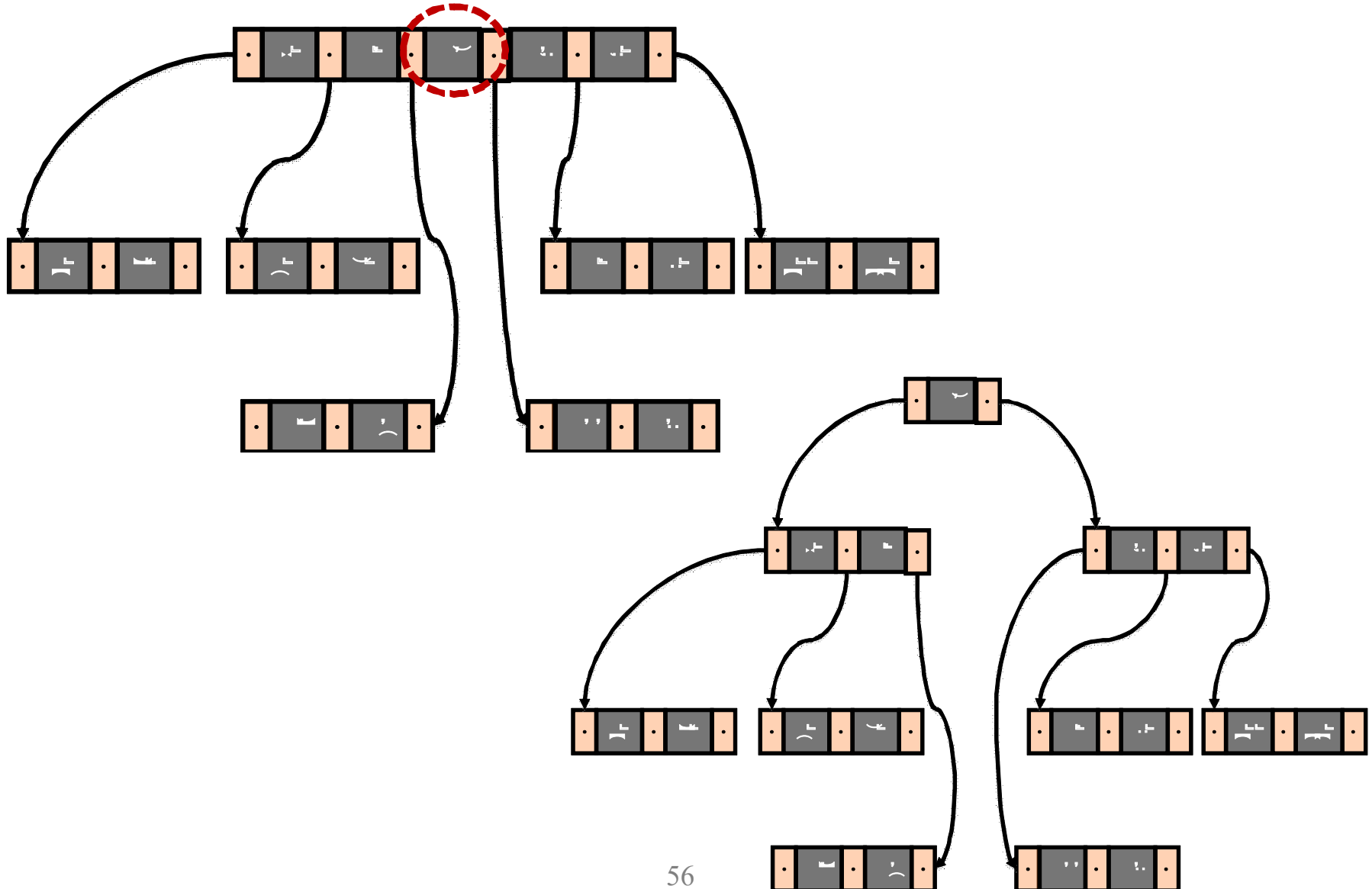
5.3 B-트리에서의 자료 추가 (3/3)

- (2)자료의 저장과 노드 분할 - 두 번째 예
 - $m=5$, 레벨이 2인 B-트리

5. 다원 탐색 트리



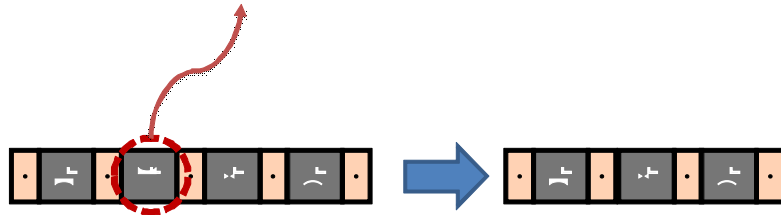
- 순환적 분할 발생



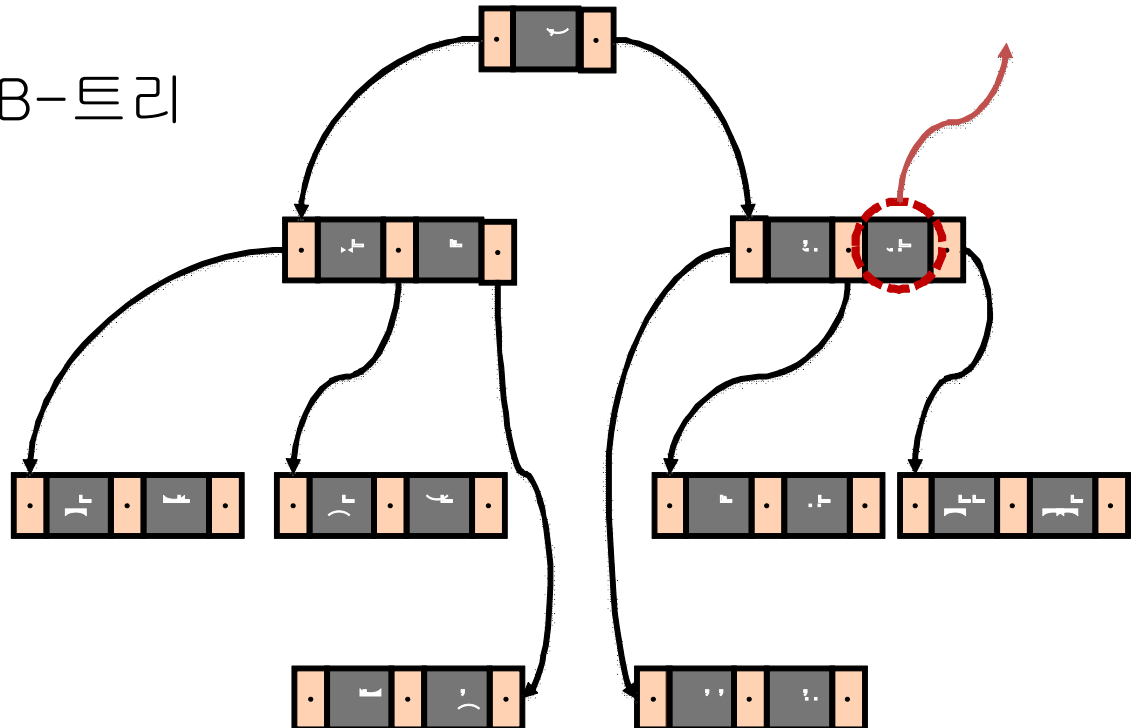
5.4 B-트리에서의 자료 제거 (1/4)

- (1) 저장 위치 찾기
- (2) 자료의 삭제: 말단 노드 여부 점검

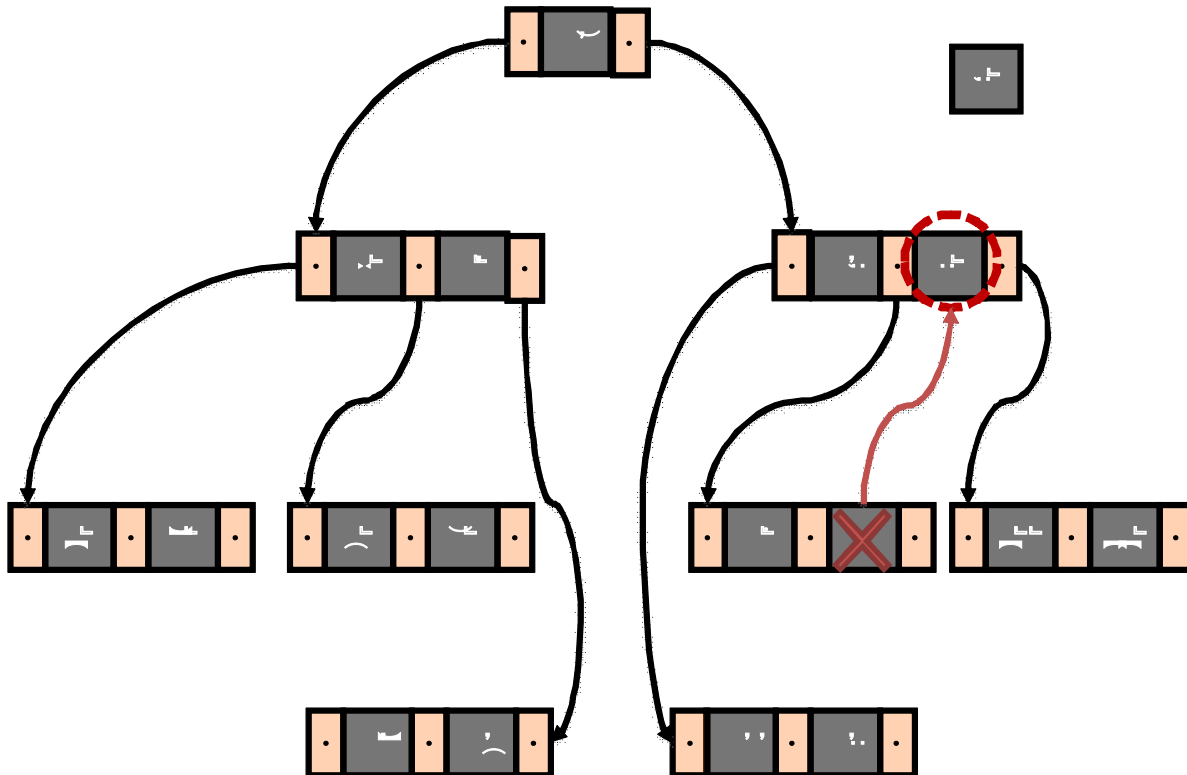
5. 다원 탐색 트리



- $m=5$, 레벨이 3인 B-트리



Key $i \leq$ (i번째 서브트리 내의 모든 키 값) $<$ Key $(i + 1)$



2. 루트 노드를 제외한 모든 내부(internal) 노드는 아래의 개수만큼의 서브트리를 가진다.

$$\lceil m/2 \rceil \leq (\text{서브트리의 개수}) \leq m$$

3. 단말 노드는 아래의 개수만큼의 자료를 가진다.

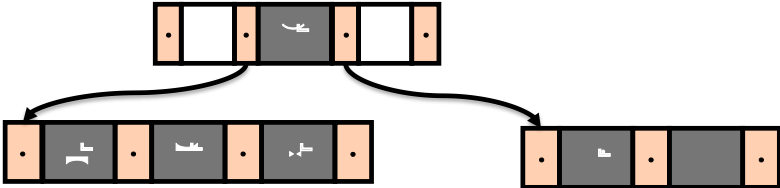
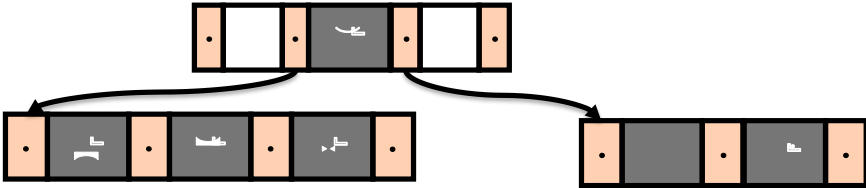
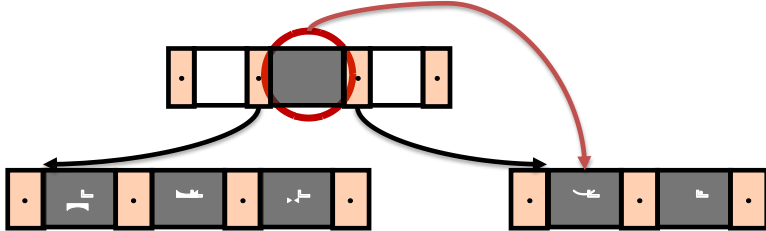
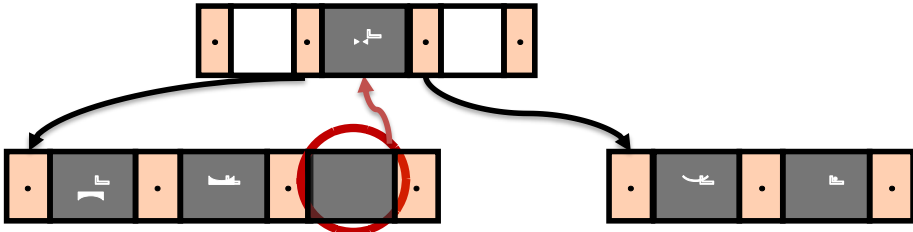
$$\lceil m/2 \rceil - 1 \leq (\text{자료의 개수}) \leq m - 1$$

5.4 B-트리에서의 자료 제거 (2/4)

- (3)균형 유지
 - 왼쪽에서 빌리기
 - 오른쪽에서 빌리기
 - 병합

5. 다원 탐색 트리

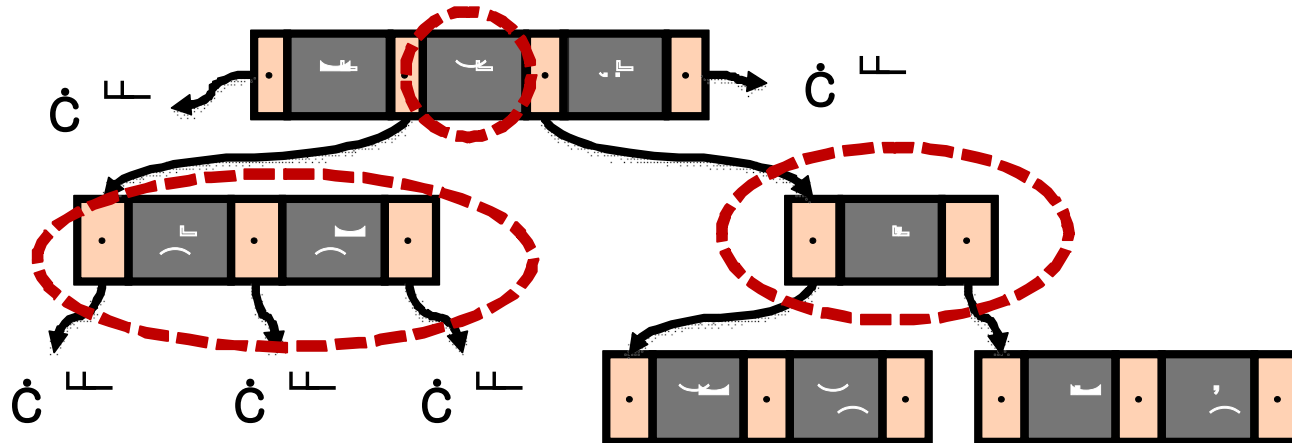
5.4 B-트리에서의 자료 제거 (3/4)

왼쪽에서 빌리기 실행 전		
왼쪽에서 빌리기	(1/3) 오른쪽 노드의 자료를 오른쪽으로 1칸 이동	
	(2/3) 부모 노드에서 오른쪽 노드로 이동	
	(3/3) 왼쪽 노드에서 부모 노드로 이동	

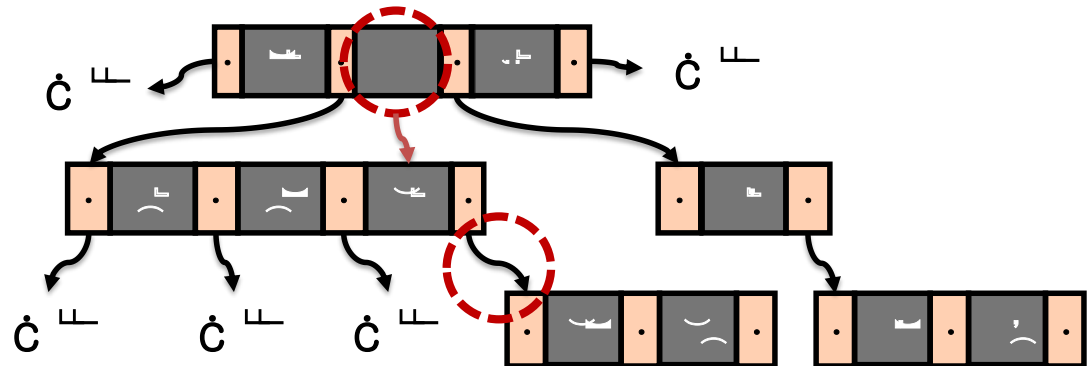
5.4 B-트리에서의 자료 제거 (4/4)

오른쪽에서 빌리기 실행 전		
오른쪽에서 빌리기	(1/3) 부모 노드에 서 왼쪽 노드로 이 동	
	(2/3) 오른쪽 노드 에서 부모 노드로 이동	
	(3/3) 오른쪽 노드 의 자료들 왼쪽으 로 이동	

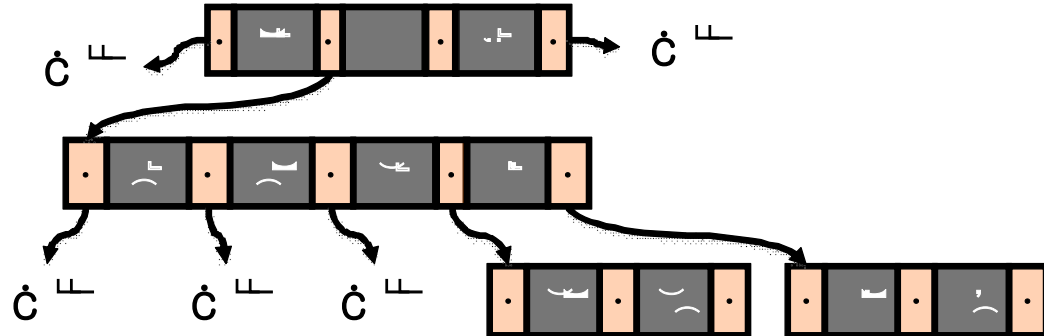
병합 (계속)



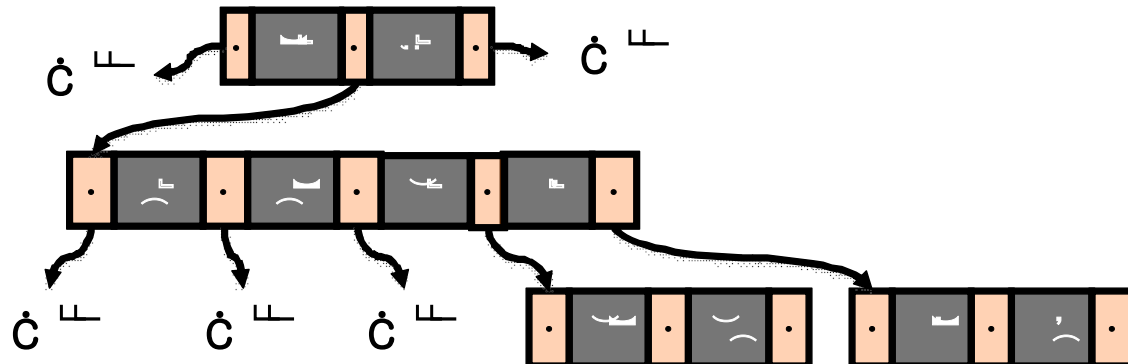
(1/3) 부모 자료를 왼쪽
쪽 노드로 이동



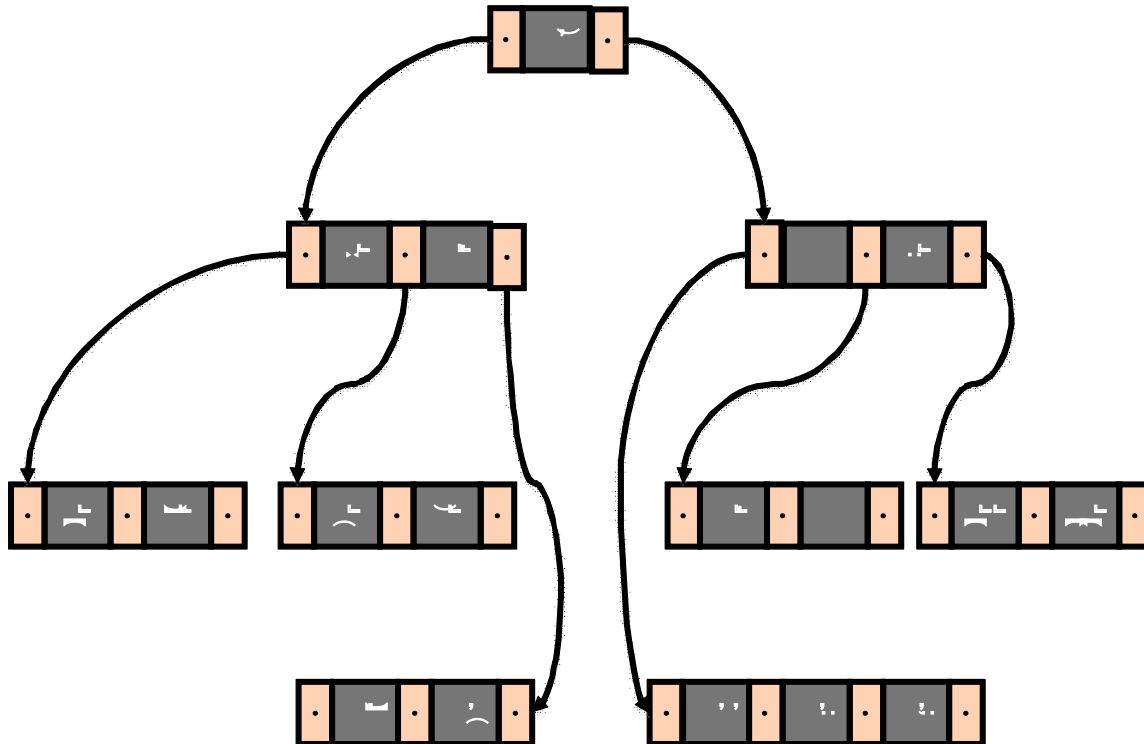
- (2/3) 오른쪽 노드의 자료를 왼쪽 노드로 이동



- (3/3) 부모 노드의 자료를 이동

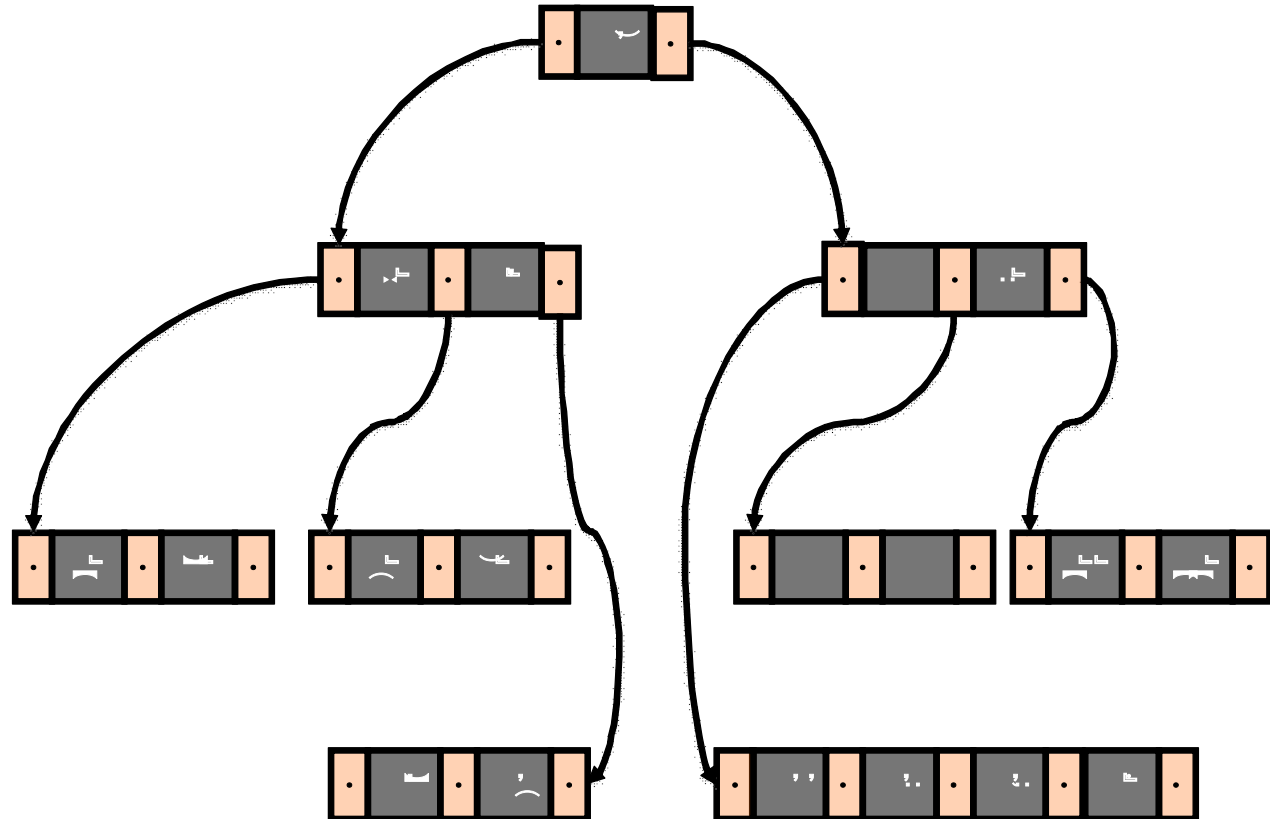


- 첫번째
 - 단계 1 - 부모 자료를 왼쪽 노드로 옮기기

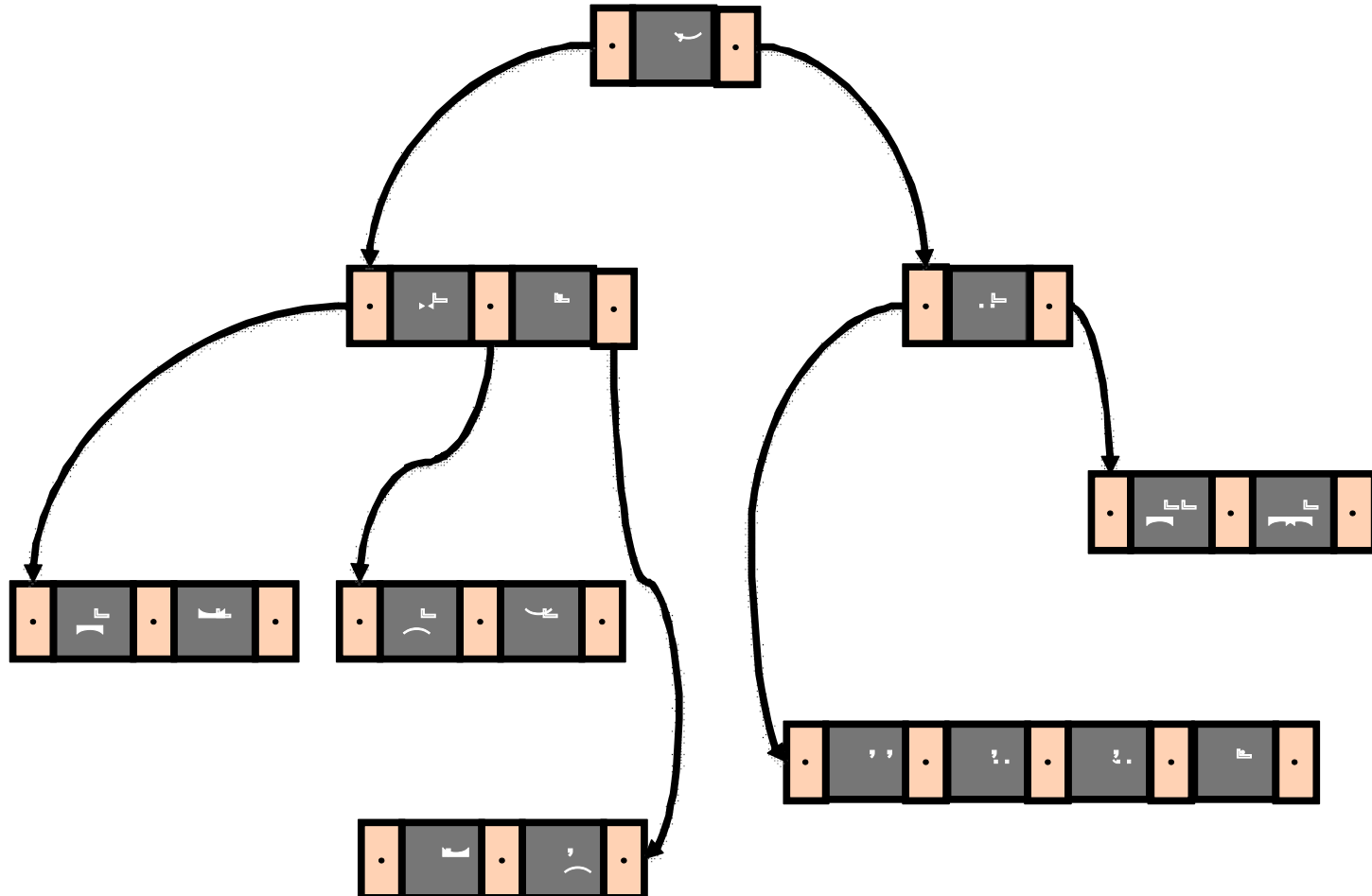


- 첫번째

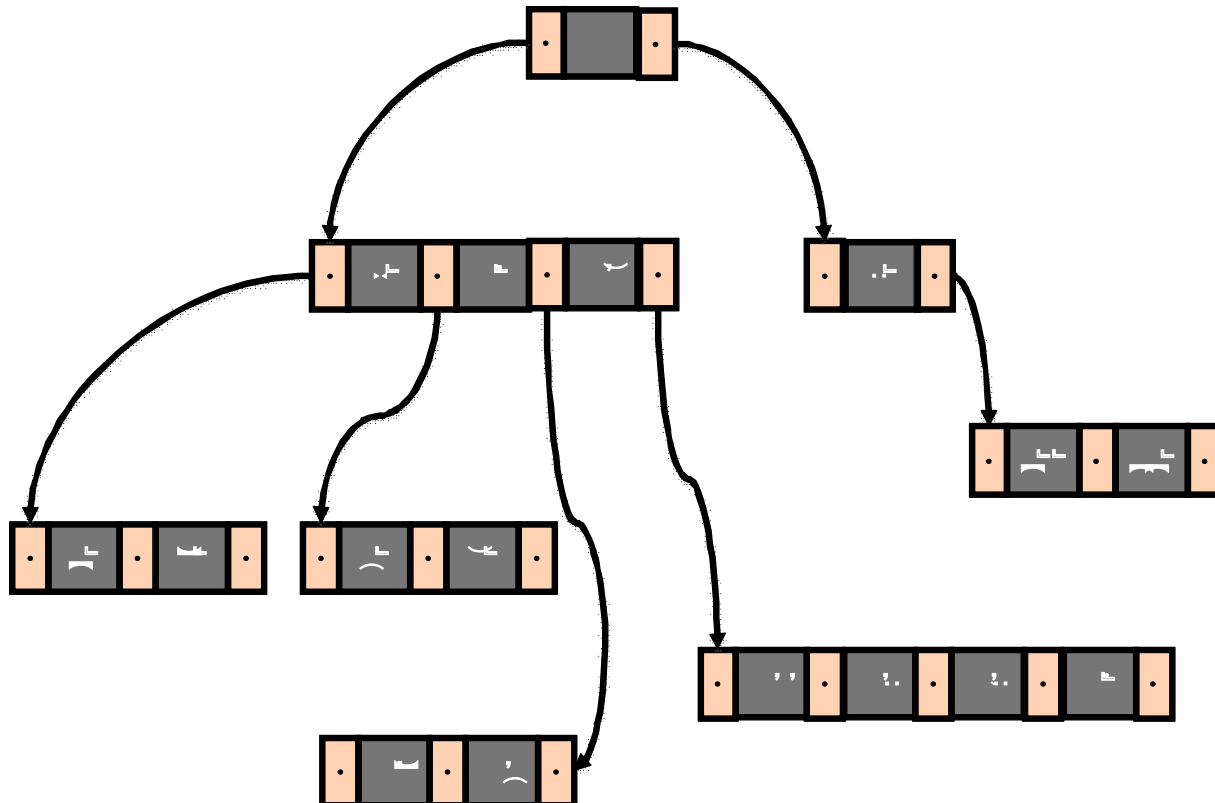
- 단계 2 - 오른쪽 노드의 자료를 왼쪽 노드로 옮기기



- 첫번째
 - 단계 3 - 부모 노드의 자료 이동

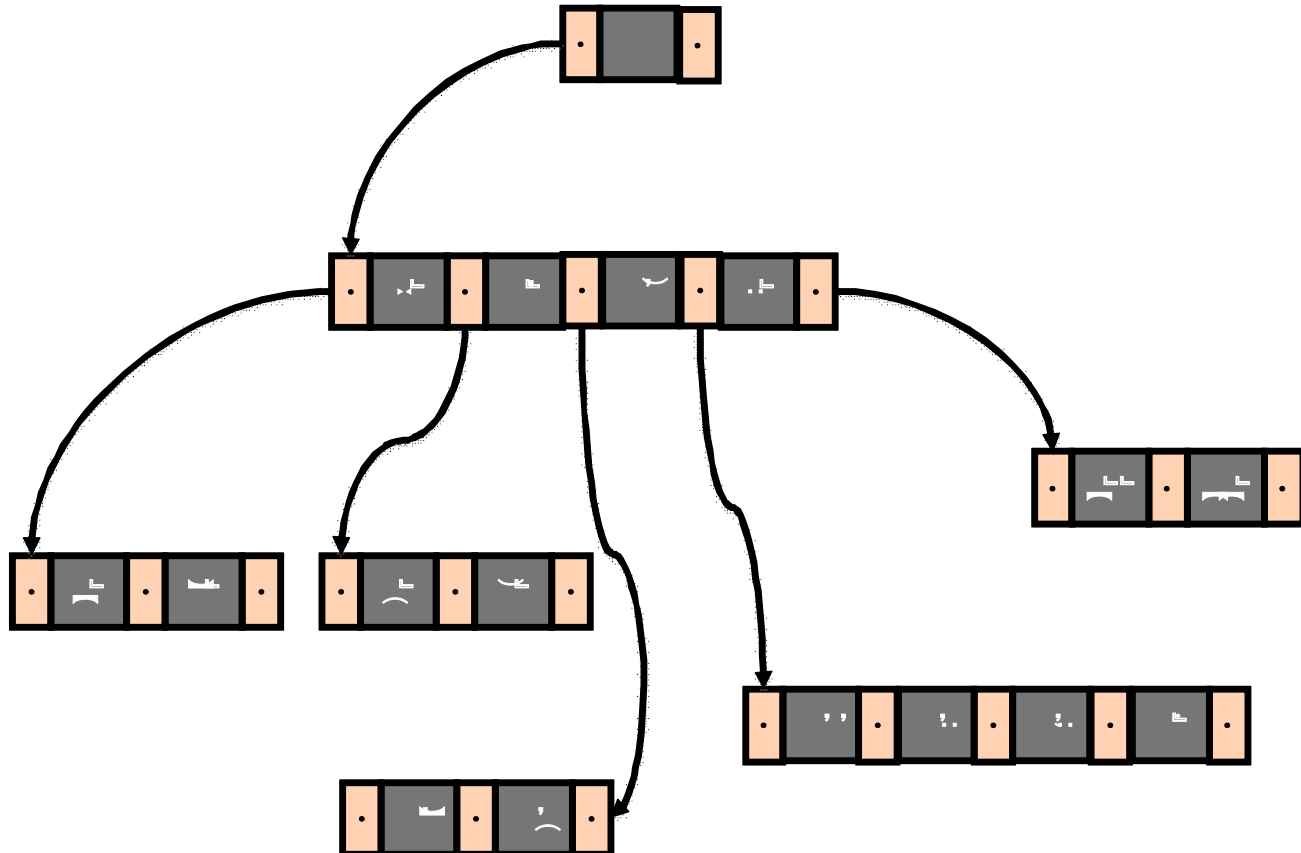


- 두번째
 - 단계 1 - 부모 자료를 왼쪽 노드로 옮기기



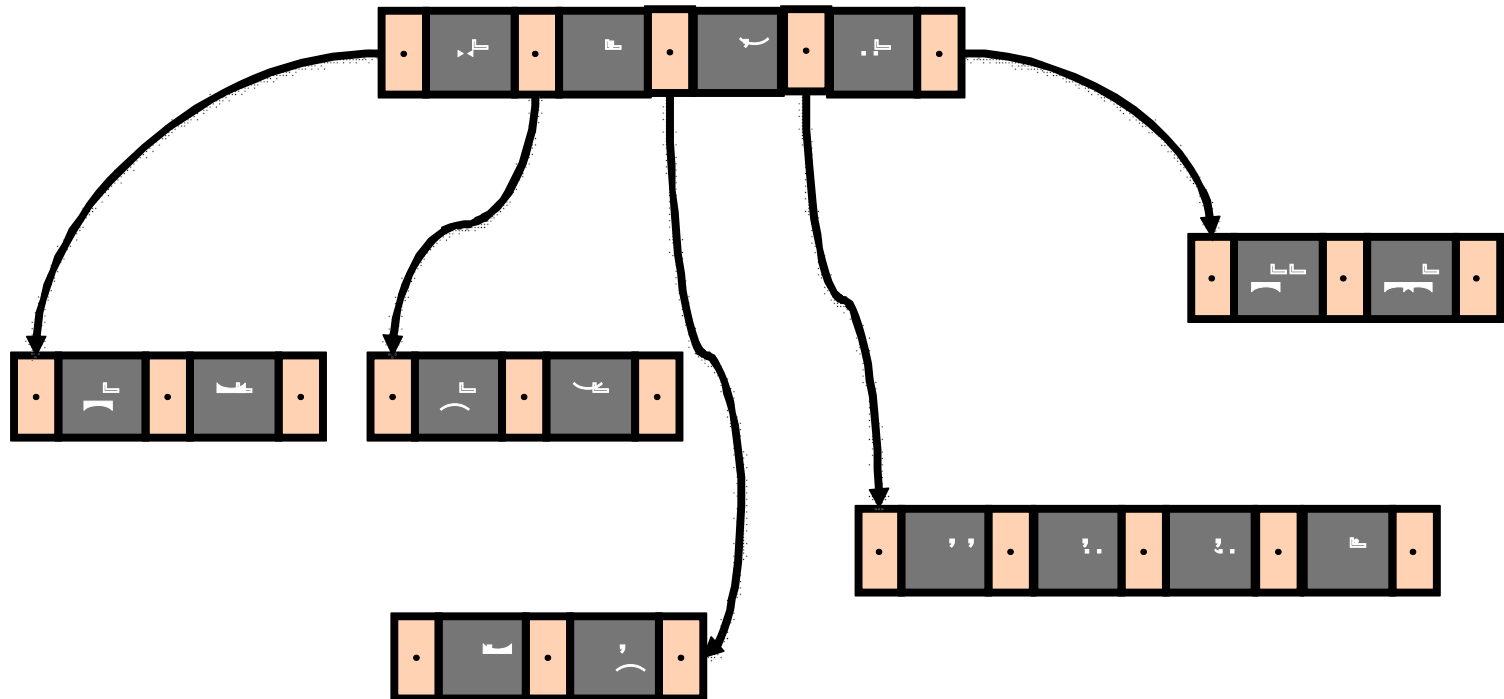
- 두번째

- 단계 2 - 오른쪽 노드의 자료를 왼쪽 노드로 옮기기



- 두 번째

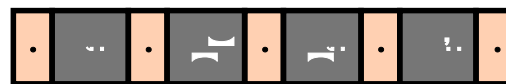
- 단계 3 - 부모 노드의 자료 이동



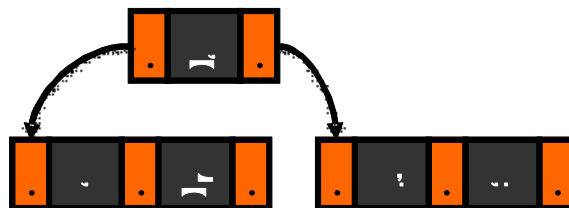
68 → 19 → 12 → 9 → 97 → 85 → 74 → 63

→ 45 → 42 → 57 → 18 → 14 → 17 → 52 → 30 → 22

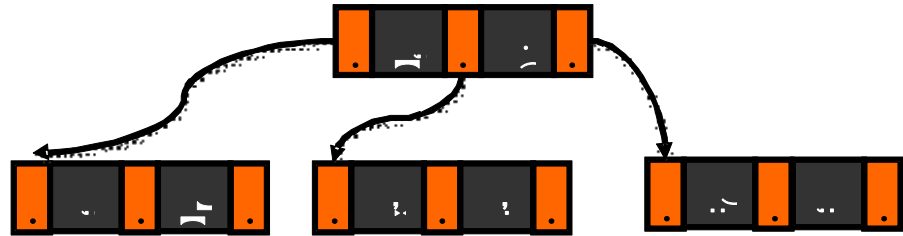
68 → 19 → 12 → 9 추가



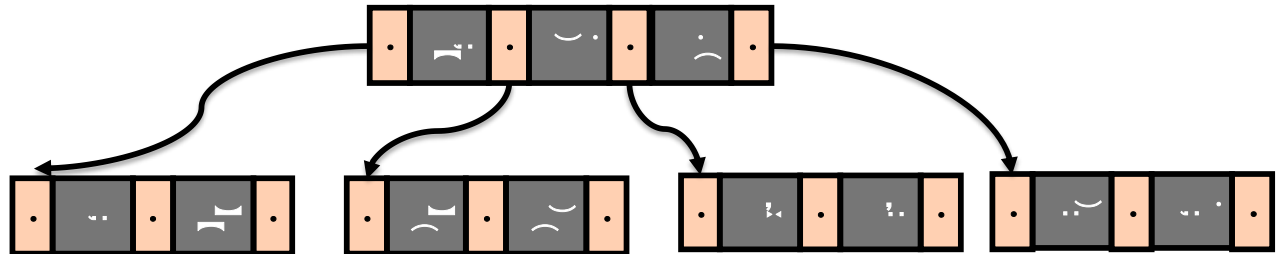
97 추가



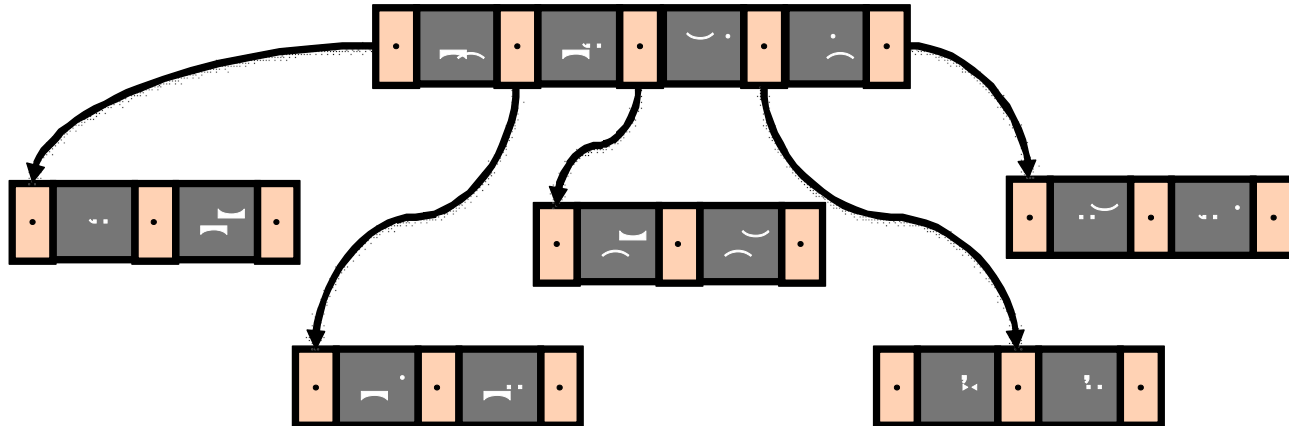
85 → 74 → 63 추가



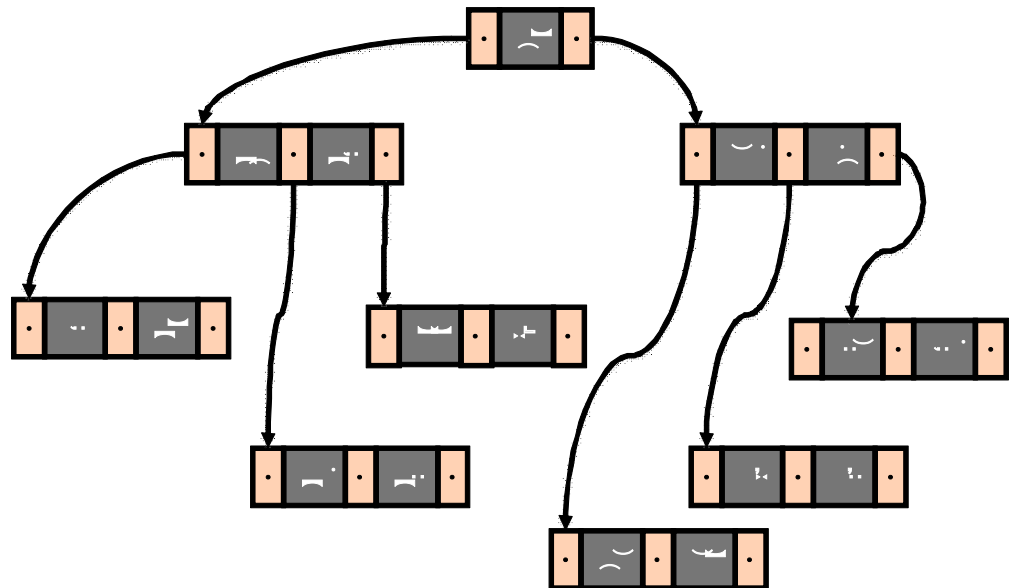
45 → 42 → 57 추가
가



- 18 → 14 → 17 추가

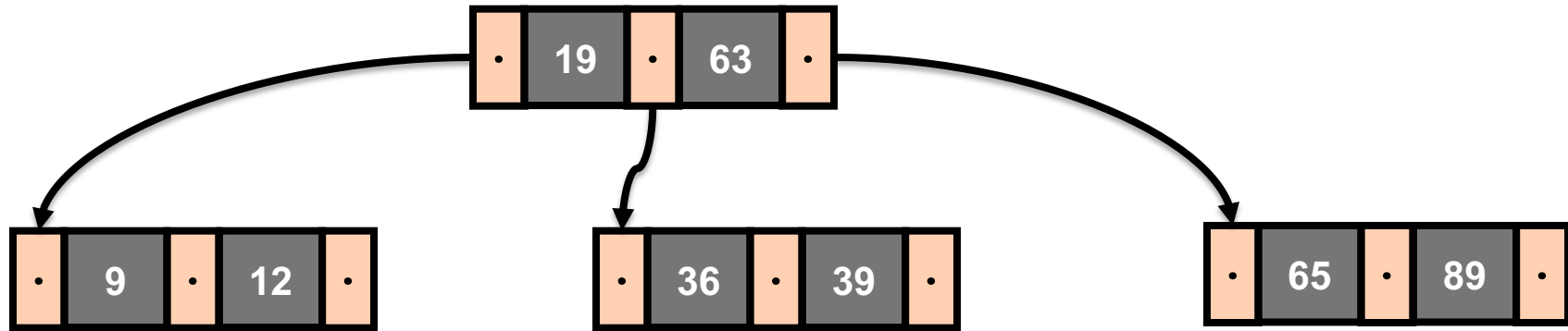


- 52 → 30 → 22 추가

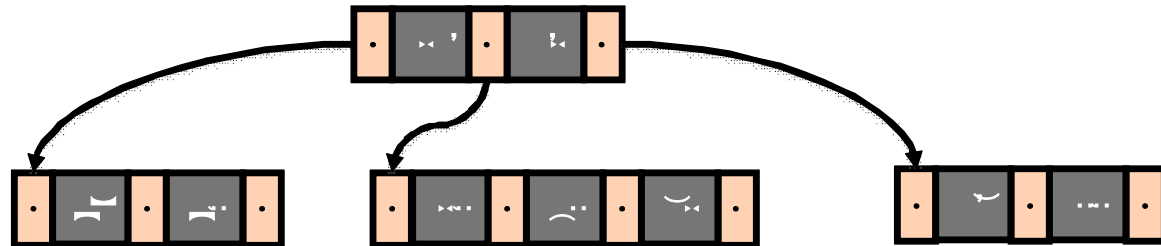


- 삭제 순서:

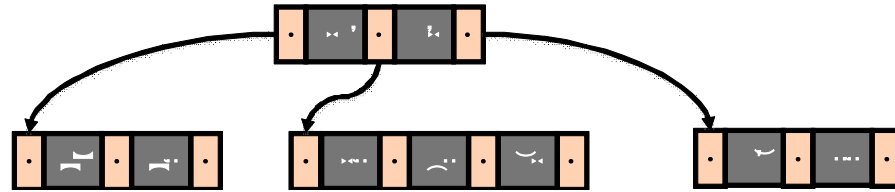
9 → 89 → 39 → 63 → 53 → 36



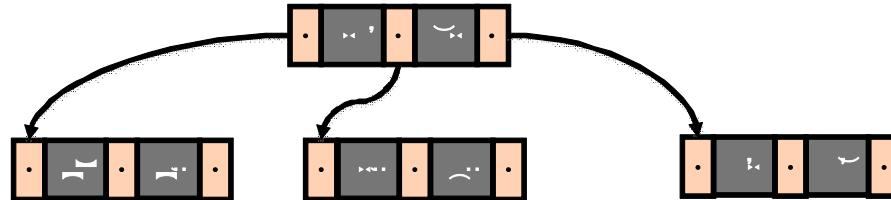
- 9삭제



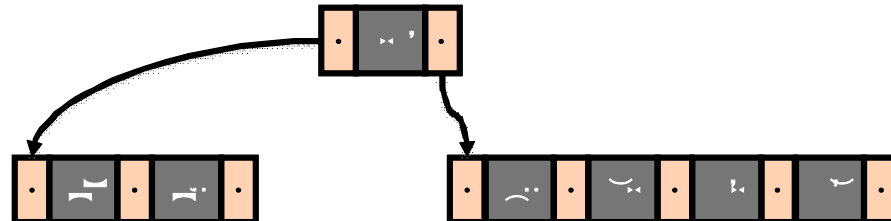
- 9 삭제



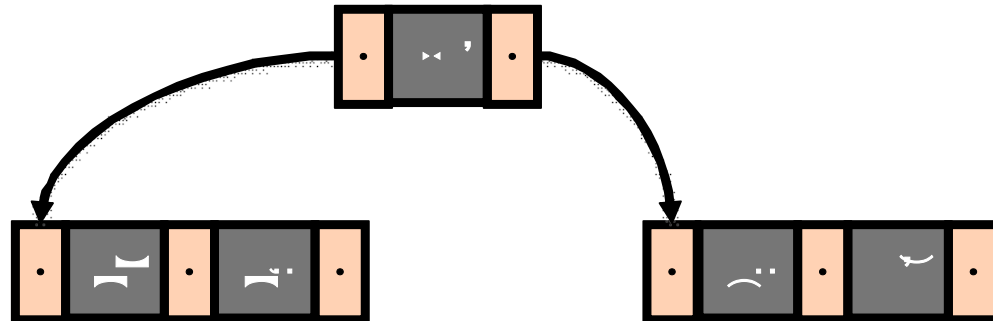
- 89 삭제



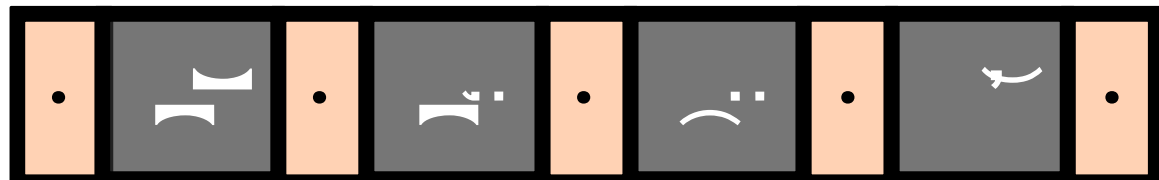
- 39 삭제



- 63→53 삭제



- 36 삭제



B-트리의 추상 자료형 및 구현

5. 다원 탐색 트리

- 소스 파일 구성

파일 이름	내용
10_07.vcproj	Visual Studio 프로젝트 파일
btreedef.h	상수와 구조체 선언
btree.h	B-트리 구현을 위한 구조체와 함수 선언
btree.c	B-트리 함수 구현
btlinkedstack.h	(B-트리 구현을 위한) 연결 스택의 구조체와 함수 선언
btlinkedstack.c	(B-트리 구현을 위한) 연결 스택의 함수 구현
exampl10_07.c	예제 프로그램

- 추상 자료형

- B-트리 생성, 삭제
- 탐색
- 데이터 추가, 삭제

※ 4.4 연결 리스트로 구현한 스택

5.7 간단한 형태의 B-트리

- 2-3트리와 2-3-4트리

5. 다원 탐색 트리

	서브트리의 개수		자료의 개수	
	최소	최대	최소	최대
3	2	3	1	2
4	2	4	1	3
m	$\lceil m/2 \rceil$	m	$\lceil m/2 \rceil - 1$	m - 1

5.8 B-트리의 변형

5. 다원 탐색 트리

- B+ 트리

- 순차적으로(Sequential) 접근 필요

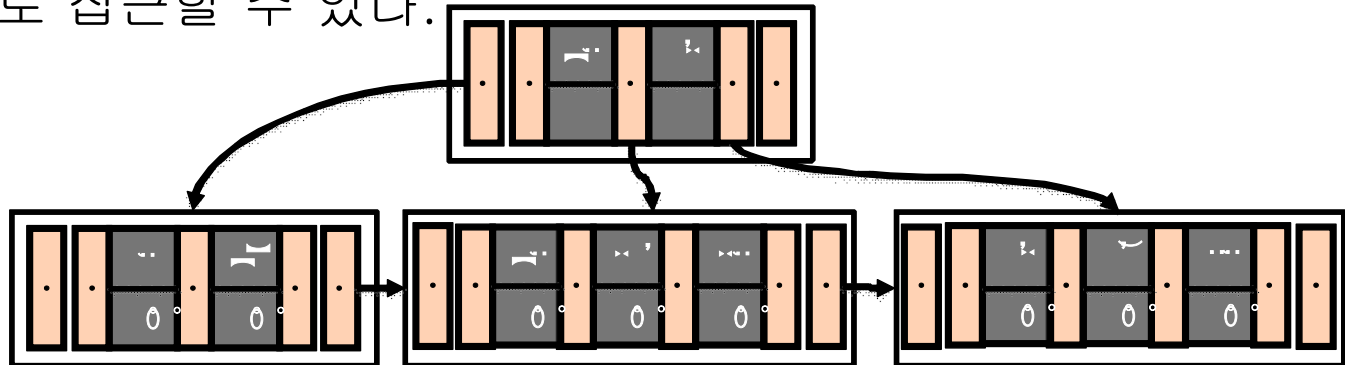
- 중위 순회

- 특성

- 1) 모든 내부(internal) 노드는 (자료의) 키 값만 저장하고, 각 자료의 데이터는 오직 말단(leaf) 노드에만 저장된다.

⇒ 따라서 말단 노드에 저장된 키와 동일한 키를 저장하는 내부 노드도 있을 수 있다.

- 2) 각 말단 노드에는 다음 형제 노드에 대한 포인터를 가지고 있어 순차적으로 접근할 수 있다.

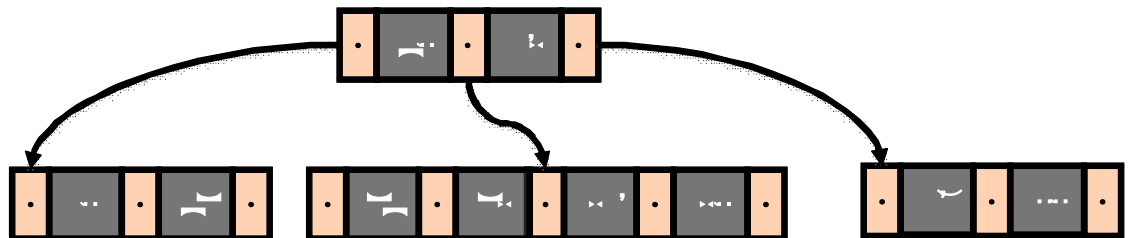


5.8 B-트리의 변형

- B* 트리

5. 다원 탐색 트리

- 대용량의 자료를 저장하기 위해 차수 m 이 5 이상인 B-트리가 주로 사용
 - 전체 노드의 약 50% 정도가 비어 있음
- 특성
 - 1) 루트 노드가 아닌 노드는 최대 저장 공간의 $2/3$ 이상의 자료가 저장되어야 한다.
 - ⇒ B-트리에서는 조건 $(\lceil m/2 \rceil - 1)$ 에 의해 최소 $1/2$ 이상의 자료가 저장
 - 2) 노드에 저장되는 자료가 넘치는 경우(overflow), 일단은 형제 노드들로 재분배시킨다. 모든 형제 노드들이 가득 찬 경우에만 분할이 발생한다.
 - ⇒ B-트리는 중간값을 가지는 자료를 부모 노드로 올려 보내고, 분할.



이번 장에서는

- 순차 검색
- 이진 검색
- 해싱
- 균형 이진 탐색 트리
- 다원 탐색 트리