자료구조론

12장 검색(search)

❑ 이 장에서 다룰 내용

- 검색
- 순차 검색
- 이진 검색
- 이진 트리 검색
- 해싱

❑ 검색

❖ 검색(search, 탐색)

- 컴퓨터에 저장한 자료 중에서 원하는 항목을 찾는 작업
- 즉, 원하는 검색 키를 지닌 항목을 찾는 것
  - 검색 키(search key) – 자료를 구별하여 인식할 수 있는 키
  - 검색 성공 – 원하는 항목을 찾은 경우
  - 검색 실패 – 원하는 항목을 찾지 못한 경우
- 삽입/삭제 작업에서의 검색
  - 원소를 삽입할 위치를 찾기 위해서 검색 연산 수행
  - 삭제할 원소를 찾기 위해서 검색 연산 수행

검색 방법

- 수행 위치에 따른 분류
  - 내부 검색 – 메모리 내의 자료에 대해서 검색 수행
  - 외부 검색 – 보조 기억 장치에 있는 자료에 대해서 검색 수행
- 검색 방식에 따른 분류
  - 비교 검색
    - 검색 대상의 키를 저장된 자료의 키와 비교하여 검색하는 방법
    - 순차 검색, 이진 검색, 이진탐색트리의 검색
  - 계산 검색
    - 계수적인 성질을 이용한 계산으로 검색하는 방법
    - 해싱
- 검색 방법의 선택
  - 자료 구조의 형태와 자료의 저장 상태(정렬 유무 등)에 따라 최적의 검색이 달라진다.

❑ 순차 검색

❖ 순차 검색(sequential search)

- 선형 검색(linear search)
- 일렬로 된 자료를 처음부터 순서대로 검색하는 방법
- 배열이나 연결 리스트로 구현한 선형 리스트에서 원하는 항목을 찾을 때 사용할 수 있다.
- 장점: 알고리즘이 단순하여 구현이 쉬움
- 단점: 검색 대상 자료가 많은 경우에 검색 시간이 오래 걸림

❖ 세가지 순차 검색

- 정렬되지 않은 배열의 순차 검색
- 정렬된 배열의 순차 검색
- 색인(index) 순차 검색

❑ 순차 검색

❖ 정렬되지 않은 배열의 순차 검색

- 검색 방법
  - 첫번째 원소부터 시작하여 순서대로 원소를 검색
    - ➢ 키 값이 일치하는 원소를 찾으면 그 원소의 인덱스를 반환
    - ➢ 마지막 원소까지 비교하여 키 값이 일치하는 원소가 없으면 찾은 원소가 없는 것이므로 검색 실패
- 예) (a) 검색 성공 (b) 검색 실패

8	30	1	9	11	19	2

9를 검색하는 경우

① 9 ≠ 8	**8**	30	1	9	11	19	2
② 9 ≠ 30	8	**30**	1	9	11	19	2
③ 9 ≠ 1	8	30	**1**	9	11	19	2
④ 9 = 9	8	30	1	**(9)**	11	19	2

6을 검색하는 경우

① 6 ≠ 8	**8**	30	1	9	11	19	2
② 6 ≠ 30	8	**30**	1	9	11	19	2
③ 6 ≠ 1	8	30	**1**	9	11	19	2
④ 6 ≠ 9	8	30	1	**9**	11	19	2
⑤ 6 ≠ 11	8	30	1	9	**11**	19	2
⑥ 6 ≠ 19	8	30	1	9	11	**19**	2
⑦ 6 ≠ 2	8	30	1	9	11	19	**2**

❑ 순차 검색

- 정렬되지 않은 배열의 순차검색 알고리즘

```
sequentialSearch1(a[], n, key)   // a[0…n-1] 에서 key를 검색하여
                                 //              일치하는 인덱스를 리턴
    i←0;
    while (i<n and a[i]≠key) do {
        i←i+1;
    }
    if (i<n) then return i;    // 검색 성공
    else return -1;            // 검색 실패
end sequentialSearch1()
```

- 비교 횟수
  - ➢ 찾는 원소가 첫 번째 원소라면 비교횟수는 1번, 두 번째 원소라면 2번, 세 번째 원소라면 3번, … , 마지막 원소라면 n번
  - ➢ 정렬되지 않은 원소에서의 순차 검색의 평균 비교 횟수
    = (1+2+3+ … + n)/n = (n+1)/2

➔ 평균 시간 복잡도 : O(n)

❑ 순차 검색

❖ 정렬된 배열의 순차 검색

- 검색 방법
  - 첫번째 원소부터 순서대로 검색하되, 원소 값이 찾는 키 값보다 크면 그 뒤로는 없는 것이므로 더 이상 검색을 수행하지 않고 검색 종료
- 예)

(a) 검색 성공의 경우

(b) 검색 실패의 경우

❑ 순차 검색

- 정렬된 배열의 순차검색 알고리즘

```
sequentialSearch2(a[], n, key)   // a[0…n-1] 에서 key를 검색하여
                                 //          일치하는 인덱스를 리턴
    i←0;
    while (i<n and a[i]<key) do {
        i←i+1;
    }
    if (i<n and a[i]=key) then return i;    // 검색 성공
    else return -1;                         // 검색 실패
end sequentialSearch2()
```

- 비교횟수
  - ➢ 검색 성공의 경우는 정렬되지 않은 배열의 순차 검색과 동일
  - ➢ 검색 실패의 경우에 평균 비교 횟수가 반으로 줄어든다.

➔ 평균 시간 복잡도 : O(n)

❑ 이진 검색

❖ 이진 검색(binary search)

- 정렬된 전체 원소들 중에서 어떤 키 값을 찾기 위해 일단 가운데 위치에 저장된 원소와 비교해보고 다음 검색 대상을 결정하는 방법
  - 검색 키 = 가운데 원소의 키 ➔ 검색 성공
  - 검색 키 > 가운데 원소의 키 ➔ 뒤쪽 부분에서 검색
  - 검색 키 < 가운데 원소의 키 ➔ 앞쪽 부분에서 검색

  키를 찾을 때까지 이진 검색을 재귀적으로 수행한다.

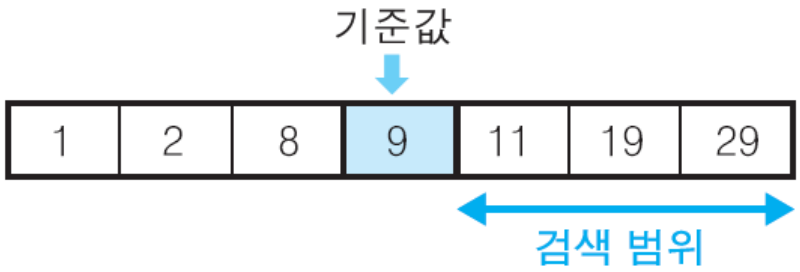
  즉, 검색 범위를 반으로 줄여가면서 이진 검색한다.
- divide-and-conquer 기법
  - 검색 범위를 반으로 분할하는 작업과 검색 작업을 반복 수행
- 이진 검색 알고리즘을 적용하기 위한 자료구조 조건
  - 원소들이 정렬되어 있어야 함
  - 특정 위치를 직접 접근 가능해야 함 (배열-순차자료구조)

❑ 이진 검색

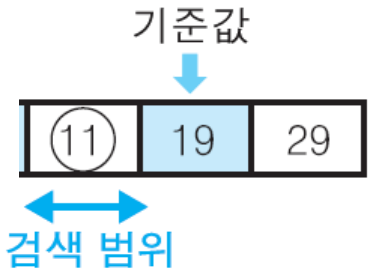
- 예 : 검색 성공의 경우

11을 검색하는 경우

① 11 > 9 → 뒤쪽 검색



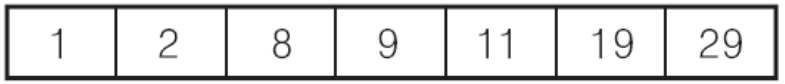
② 11 < 19 → 앞쪽 검색



③ 11 = 11 → 검색 성공

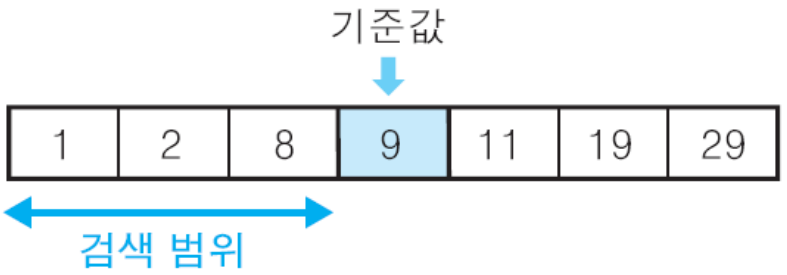
❑ 이진 검색

- 예 : 검색 실패의 경우

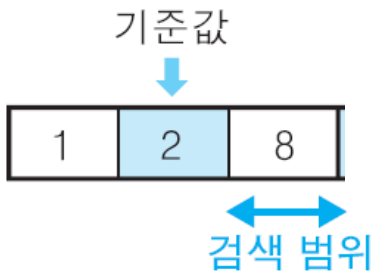


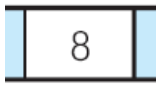
6을 검색하는 경우

① 6 < 9 → 앞쪽 검색



② 6 > 2 → 뒤쪽 검색





③ 6 ≠ 8 → 검색 종료

❑ 이진 검색

❖ 이진 검색 알고리즘

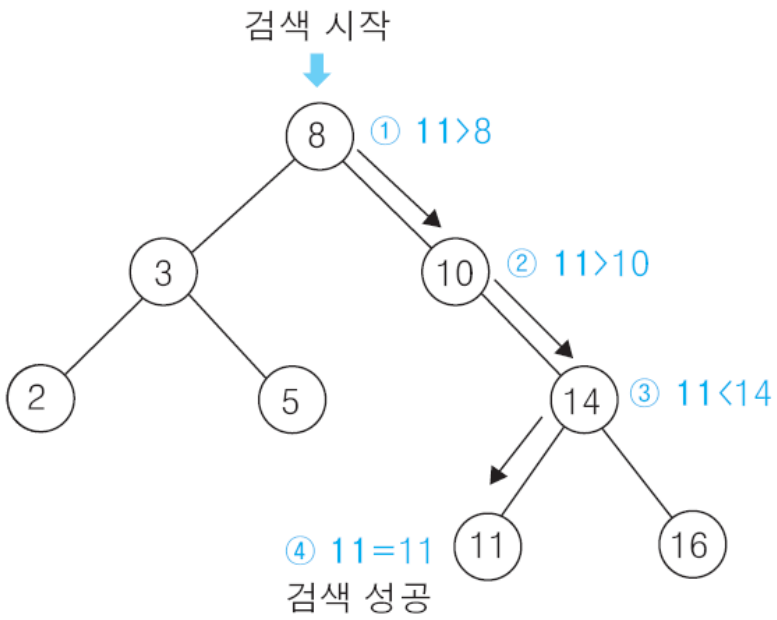
```
binarySearch(a[], low, high, key) // a[low...high] 에서 key를 검색하여
                                  //              일치하는 인덱스를 리턴
    if (low > high) return -1;
    else {
        middle ← (low+high)/2;
        if (key = a[middle]) then
            return middle;
        else if (key < a[middle]) then
            return binarySearch(a, low, middle-1, key);
        else
            return binarySearch(a, middle+1, high, key);
    }
end binarySearch()
```

- 검색 시간 복잡도 : $O(\log_2 n)$
- 단, 원소의 삽입/삭제시 배열을 오름차순 정렬 상태로 유지하기 위한 추가 작업 필요

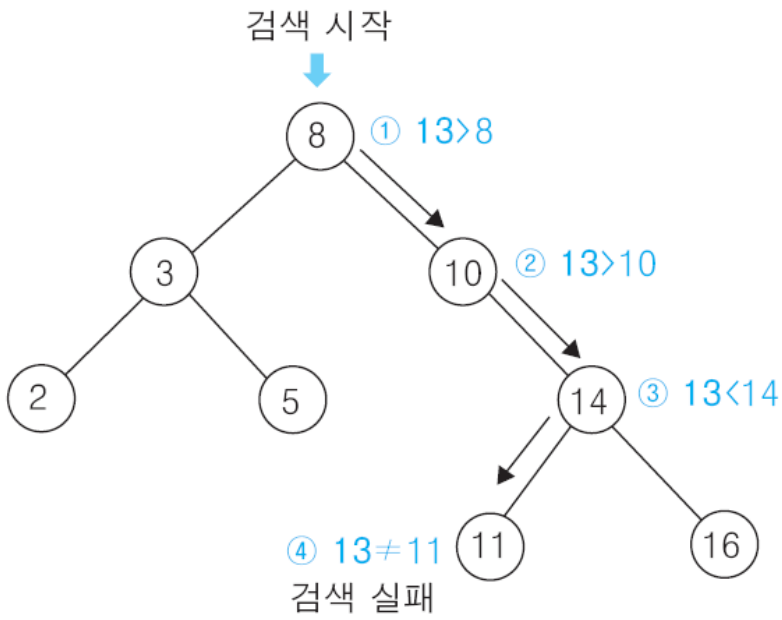
❑ 이진 트리 검색

❖ 이진 트리 검색

- 이진 탐색 트리(binary search tree)를 사용한 검색 방법 (8장)



(a) 검색 성공의 경우



(b) 검색 실패의 경우

- 시간 복잡도 : 트리의 균형이 맞는 경우 $O(\log_2 n)$

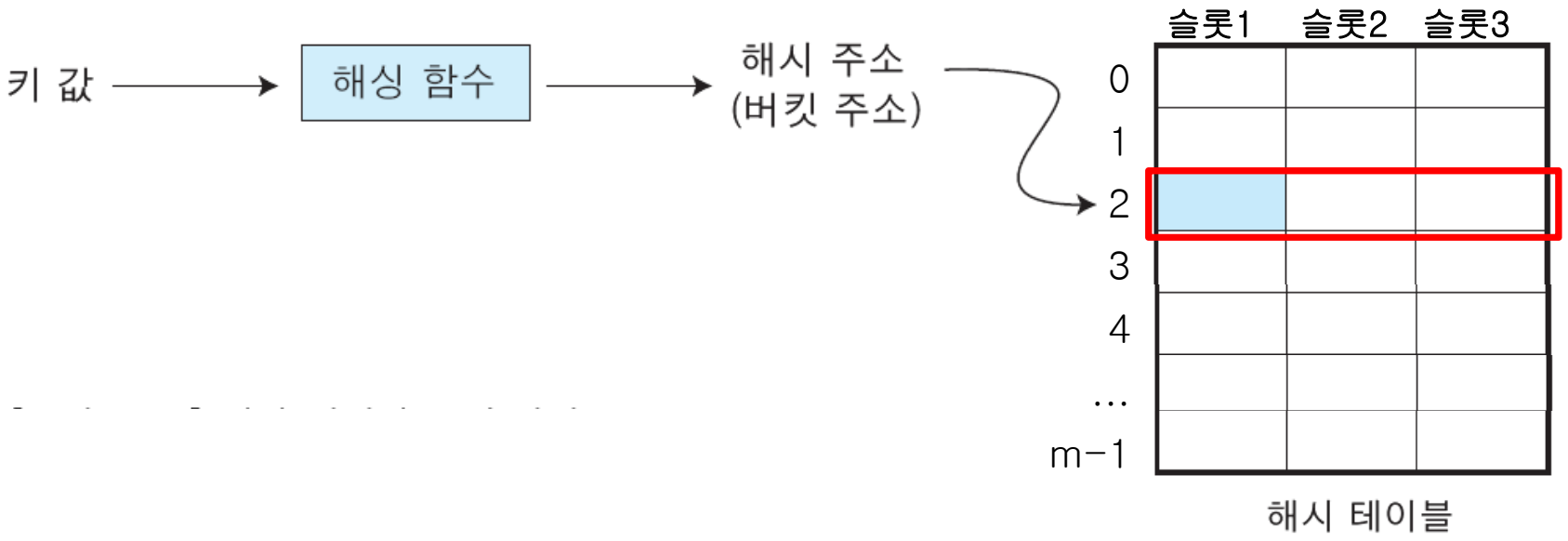
❑ 해싱

❖ 해싱(hashing)

- 찾고자 하는 원소가 저장된 위치를 산술적인 연산을 통해 계산하여 바로 찾아가는 방법
- 검색 방법
  - 검색 키 값을 해시 함수에 대입하여 주소를 구하고,
  - 구한 주소에 해당하는 해시 테이블로 바로 이동
  - 해당 주소에 찾는 항목이 있으면 검색 성공, 없으면 검색 실패
- 해시 함수(hash function)
  - 원소의 키 값을 원소의 위치로 변환하는 함수
- 해시 테이블(hash table)
  - 해시 함수에 의해서 계산한 위치에 따라 원소를 저장한 테이블

❑ 해싱

- 해싱 수행 방법



해시 테이블의 크기(즉, 버킷 수) = m

❑ 해싱

❖ 해싱 용어

- 충돌(collision)
  - 서로 다른 키 값에 대해서 해시 함수를 사용하여 구한 주소(버킷 주소)가 같은 경우
  - 충돌이 발생한 경우 처리 방법
    - ➢ 해당 **버킷**에 비어있는 **슬롯**이 있으면 synonym 관계로 저장
    - ➢ 해당 버킷이 포화 상태이어서 더 이상 키 값을 저장할 수 없으면 ➔ **오버플로우**(overflow)
- 동의어(synonym), 동거자
  - 서로 다른 키 값을 가지지만 해시 함수에 의해서 같은 버킷에 저장된 원소들

❑ 해싱 – 해시 함수

❖ 바람직한 해시 함수의 조건

- 키값들을 해시 테이블에 고르게 분포할 수 있도록 주소를 계산해야 한다.
  - 즉, 가능한 충돌을 적게 발생시키는 함수이어야 한다.
  - 충돌이 많이 발생한다는 것은 같은 버킷을 할당받는 키 값이 많다는 것 ➔ 비어있는 버킷이 많은데도 어떤 버킷은 오버플로우가 발생할 수 있는 상태가 되므로 좋은 해시 함수가 아님
- 계산이 쉬워야 한다.
  - 비교 검색 방법을 사용하여 키 값의 비교연산을 수행하는 시간보다 해시 함수를 사용하여 계산하는 시간이 빨라야 해싱을 사용하는 의의가 있다.

❑ 해싱 – 해시 함수

❖ 해시 함수의 종류

- bit extraction(비트 추출) 방법
- mid-square(중간-제곱) 방법
- division(나누기) 방법
- multiplication(곱하기) 방법
- folding(접지) 방법
- …

❑ 해싱 – 해시 함수

❖ division(나누기) 방법

- 나머지 연산을 사용하는 방법
- 키 값 k를 해시 테이블의 크기 m으로 나눈 나머지를 해시 주소로 사용

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

- m으로 나눈 나머지 값은 0~(m-1)이므로 해시 테이블의 인덱스로 사용할 수 있다.
- 해시 주소는 충돌이 발생하지 않고 고르게 분포하도록 생성되어야 하므로 m은 적당한 크기의 소수(prime number)

❖ multiplication(곱하기) 방법

- 곱하기 연산을 사용하는 방법
- 키 값 k와 정해진 실수 α를 곱한 결과에서 소수점 이하 부분만을 테이블의 크기 m과 곱하여 그 정수 값을 해시 주소로 사용

❑ 해싱 – 충돌 처리

❖ 충돌 처리 방법

: 버킷 당 슬롯이 하나라고 가정하면, 충돌 = 오버플로우

- 개방 주소법(open addressing)
  - 선형 조사(linear probing) : linear open addressing
  - 이차 조사(quadratic probing)
  - 이중 해싱(double hashing)
- 체이닝(chaining)

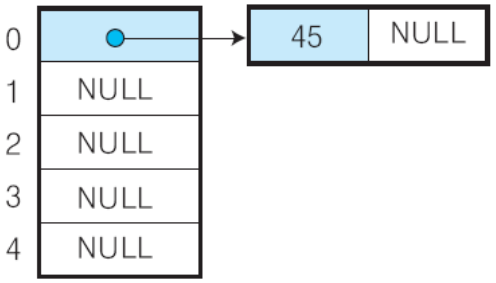
❑ 해싱 – 충돌 처리

❖ 체이닝(chaining)

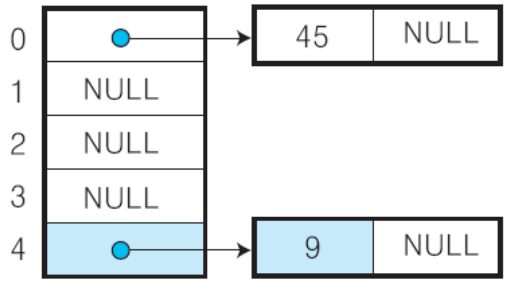
- 하나의 버킷에 여러 키값을 저장할 수 있도록 각 버킷을 연결 리스트로 구현
  - 즉, synonym들을 하나의 연결 리스트에 저장
  - 각 버킷에 대해 헤드노드를 둠
  - 해시 테이블은 헤드노드들의 1차원 배열
- 저장 방법
  - 해시 함수로 버킷 번호를 얻은 후, 해당 버킷의 연결리스트에 원소 삽입
- 검색 방법
  - 해시 함수로 버킷 번호를 얻은 후, 버킷 내에서 원하는 키 값을 찾으려면 버킷의 연결 리스트를 선형 검색

❑ 해싱 – 충돌 처리

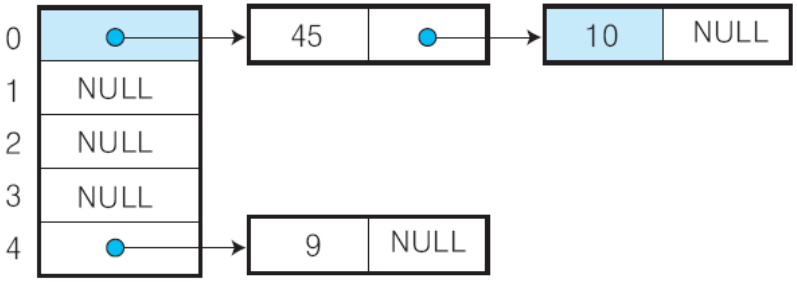
저장할 키 값 : 45, 9, 10, 96, 25
해시 함수 : division method



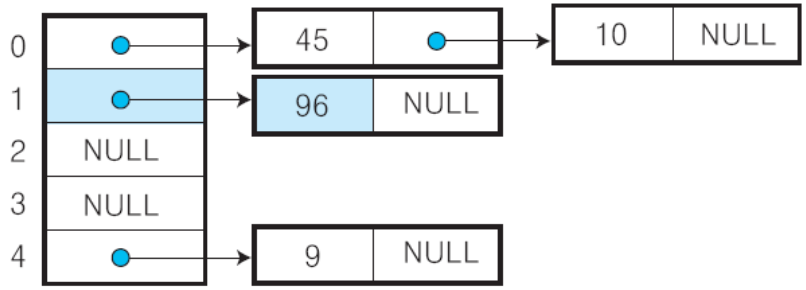
(1) 1단계



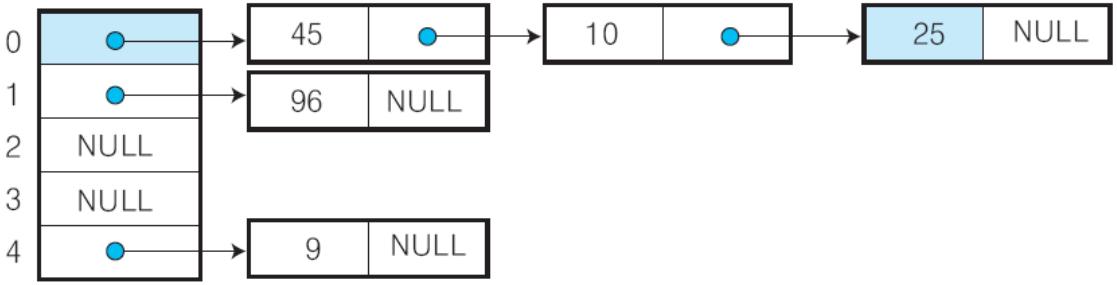
(2) 2단계



(3) 3단계



(4) 4단계



(5) 5단계

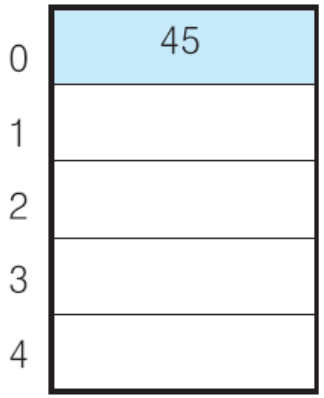
❑ 해싱 – 충돌 처리

❖ 선형 개방 주소법(linear open addressing)

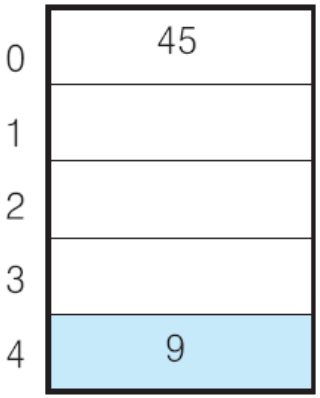
- 저장 방법
  - 충돌이 일어나면 다음 버킷을 조사한다.
  - 다음 버킷이 비어있으면 키 값을 저장
  - 버킷이 가득 차있으면 다시 다음 버킷을 조사
  - 이런 과정을 되풀이 하여 비어있는 버킷을 찾아 저장함
- 검색 방법
  - 버킷에 저장된 값이 검색 키 값과 다르면 바로 다음 버킷을 계속 조사한다.
  - 다음 버킷이 비어있으면 검색 실패

❑ 해싱 – 충돌 처리

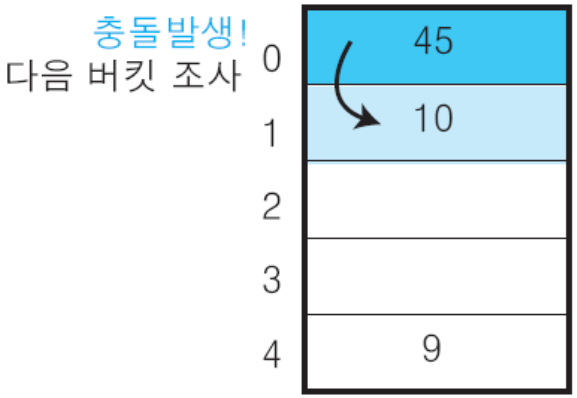
저장할 키 값 : 45, 9, 10, 96, 25
해시 함수 : division method



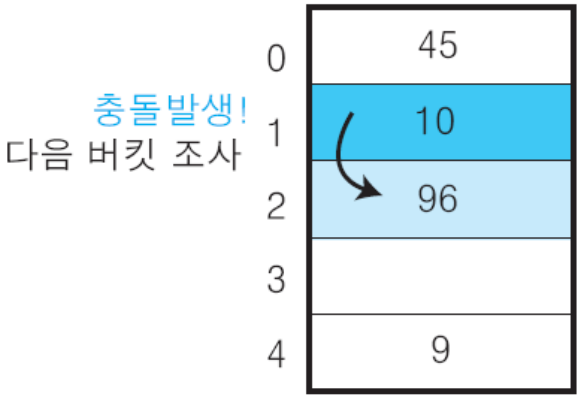
(1) 1단계



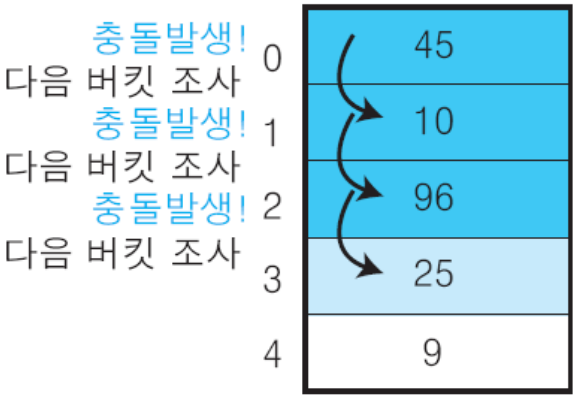
(2) 2단계



(3) 3단계



(4) 4단계



(5) 5단계