데이터 구조

검색(Search)

Contents

❖ 학습목표

- 자료 검색의 개념을 알아본다.
- 순차 검색의 개념과 알고리즘을 알아본다.
- 이진 검색의 개념과 알고리즘을 알아본다.
- 해싱의 원리를 알아보고 다른 검색 알고리즘과의 차이를 살펴본다.

❖ 내용

1. 검색의 이해

❖ 검색search

- 컴퓨터에 저장한 자료 중에서 원하는 항목을 찾는 작업
  - 검색 성공 - 원하는 항목을 찾은 경우
  - 검색 실패 – 원하는 항목을 찾지 못한 경우
- 탐색 키를 가진 항목을 찾는 것
  - 탐색 키search key - 자료를 구별하여 인식할 수 있는 키
- 삽입/삭제 작업에서의 검색
  - 원소를 삽입하거나 삭제할 위치를 찾기 위해서 검색 연산 수행

1. 검색의 이해

- 검색 방법
  - 수행 위치에 따른 분류
    - 내부 검색Internal Search : 메모리 내의 자료에 대해서 검색 수행
    - 외부 검색External Search : 보조 기억 장치에 있는 자료에 대해서 검색 수행
  - 검색 방식에 따른 분류
    - 비교 검색 방식comparison search method
    - 계산 검색 방식non-comparison method

표 10-1 검색 방법에 따른 분류

검색 방법	설명	종류
비교 검색	검색 대상의 키를 비교하여 검색한다.	순차 검색, 이진 검색, 트리 검색
계산 검색	계수적인 성질을 이용한 계산으로 검색한다.	해싱

2. 순차 검색

❖ **순차 검색**sequential search**, 선형 검색**linear search

- 일렬로 된 자료를 처음부터 마지막까지 순서대로 검색하는 방법
- 가장 간단하고 직접적인 검색 방법
- 배열이나 연결 리스트로 구현된 순차 자료 구조에서 원하는 항목을 찾는 방법
- 검색 대상 자료가 많은 경우에 비효율적이지만 알고리즘이 단순하여 구현이 용이함

2. 순차 검색

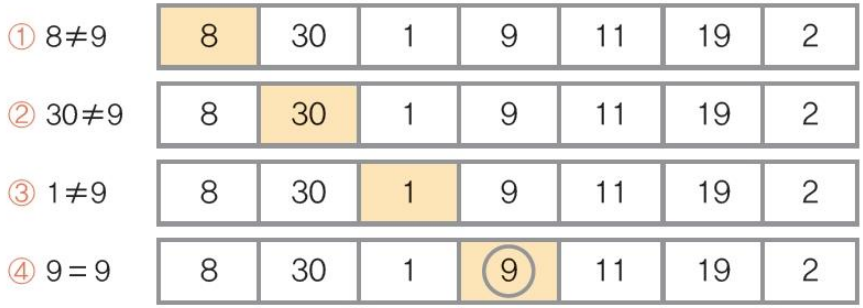
❖ 정렬되어 있지 않은 자료를 순차 검색

- 검색 방법
  - 첫 번째 원소부터 시작하여 마지막 원소까지 순서대로 키 값이 일치하는 원소가 있는지를 비교하여 찾는다.
    - 키 값이 일치하는 원소를 찾으면 그 원소가 몇 번째 원소인지를 반환
    - 마지막 원소까지 비교하여 키 값이 일치하는 원소가 없으면 찾은 원소가 없는 것이므로 검색 실패

2. 순차 검색

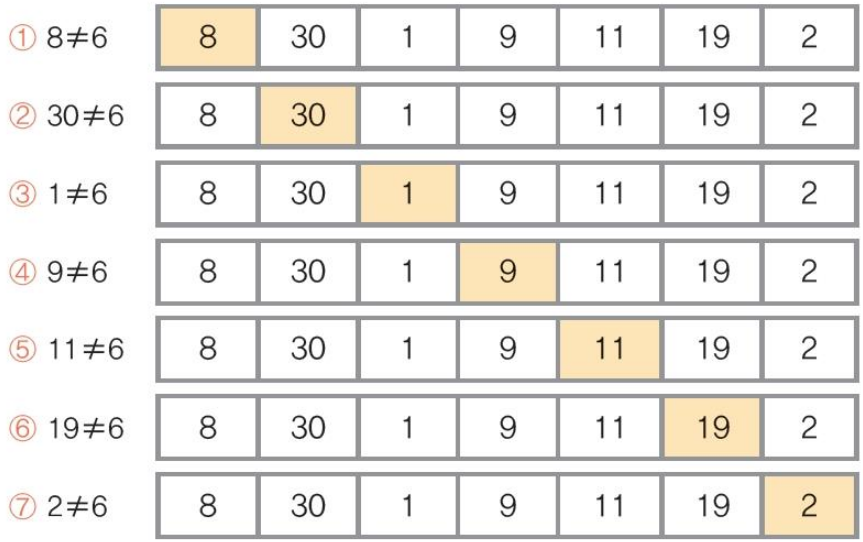
8	30	1	9	11	19	2

(a) 정렬되어 있지 않은 자료의 예



① 8≠9	8	30	1	9	11	19	2
② 30≠9	8	30	1	9	11	19	2
③ 1≠9	8	30	1	9	11	19	2
④ 9=9	8	30	1	9	11	19	2

(b) 검색 성공의 예 : 9 검색



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

(c) 검색 실패의 예 : 6 검색

그림 10-1 정렬되어 있지 않은 자료에서의 순차 검색 예

2. 순차 검색

- 정렬되지 않은 자료의 순차검색 알고리즘

알고리즘 10-1 정렬되지 않은 자료의 순차 검색

```
sequentialSearch1(a[], n, key)
    i ← 0;
    while (i<n and a[i]≠key) do {
        i ← i + 1;
    }
    if (i<n) then return i;
    else return -1;
end sequentialSearch1()
```

- 비교 횟수 - 찾고자 하는 원소의 위치에 따라 결정
  - 찾는 원소가 첫 번째 원소라면 비교횟수는 1번, 두 번째 원소라면 비교횟수는 2번, 세 번째 원소라면 비교횟수는 3번, 찾는 원소가 i번째 원소이면 i번
- 평균 비교 횟수 : $\frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n} \times \frac{(nn+1)}{2} = \frac{n+1}{2}$
- 평균 시간 복잡도 : O(n)

2. 순차 검색

- [예제 10-1] 정렬되지 않은 자료를 순차 검색하기 : 교재 582p
- 실행 결과

```
《 검색 대상 자료 》
  8  30   1   9  11  19   2
9를 검색하라! -》  4번째에 검색 성공!
6를 검색하라! -》  8번째에 검색 실패!
```

2. 순차 검색

❖ 정렬된 자료를 순차 검색

- 원소의 키값이 찾는 키값보다 크면 찾는 원소가 없는 것이므로 더 이상 검색을 수행하지 않아도 검색 실패를 알 수 있음

그림 10-2 정렬된 자료에서의 순차 검색 예

2. 순차 검색

- 정렬된 자료의 순차검색

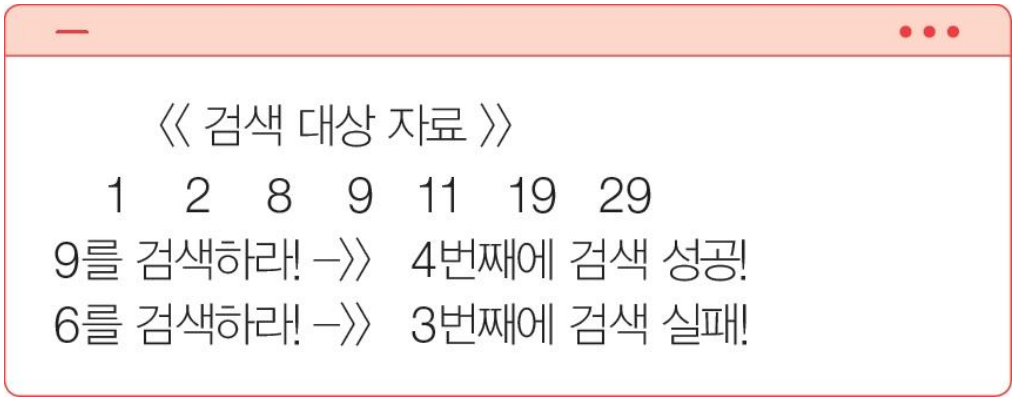
알고리즘 10-2 정렬된 자료의 순차 검색

```
sequentialSearch2(a[], n, key)
    i ← 0;
    while (a[i]<key) do {
        i ← i + 1;
    }
    if (a[i] = key) then return i;
    else return  -1;
end sequentialSearch2()
```

- 비교 횟수 - 찾고자 하는 원소의 위치에 따라 결정
  - 검색 실패의 경우에 평균 비교 횟수가 반으로 줄어듦
- 평균 시간 복잡도 : O(n)

2. 순차 검색

- [예제 10-2] 정렬된 자료를 순차 검색하기 : 교재 584p
- 실행 결과



2. 순차 검색

❖ 색인 순차 검색index sequential search

- 정렬되어있는 자료에 대한 인덱스 테이블index table을 추가로 사용하여 탐색 효율을 높인 검색 방법
- 인덱스 테이블
  - 배열에 정렬되어있는 자료 중에서 일정한 간격으로 떨어져있는 원소들을 저장한 테이블
    - 자료가 저장되어있는 배열의 크기가 n이고 인덱스 테이블의 크기가 m일 때, 배열에서 n/m 간격으로 떨어져있는 원소와 그의 인덱스를 인덱스 테이블에 저장
- 검색 방법
  - indexTable[i].key ≤ key < indexTable[i+1].key를 만족하는 i를 찾아서 배열의 어느 범위에 있는지를 먼저 알아낸 후에 해당 범위에 대해서만 순차 검색 수행

2. 순차 검색

- 색인 순차 검색 (예) 검색 대상 자료 : {1, 2, 8, 9, 11, 19, 29}
  - 키값이 6인 원소를 검색하려면 먼저 인덱스 테이블의 키값을 검색.
    - 키값 6은 indexTable[0].key ≤ 6 < indexTable[1].key로 인덱스 테이블의 키값 1과 9 사이에 있음.
    - 따라서 검색 범위는 indexTable[0].index ≤ 검색 범위 < indexTable[1].index가 되므로 자료가 저장되어 있는 배열의 인덱스 0번부터 2번까지를 검색 범위로 정하고 순차 검색을 수행

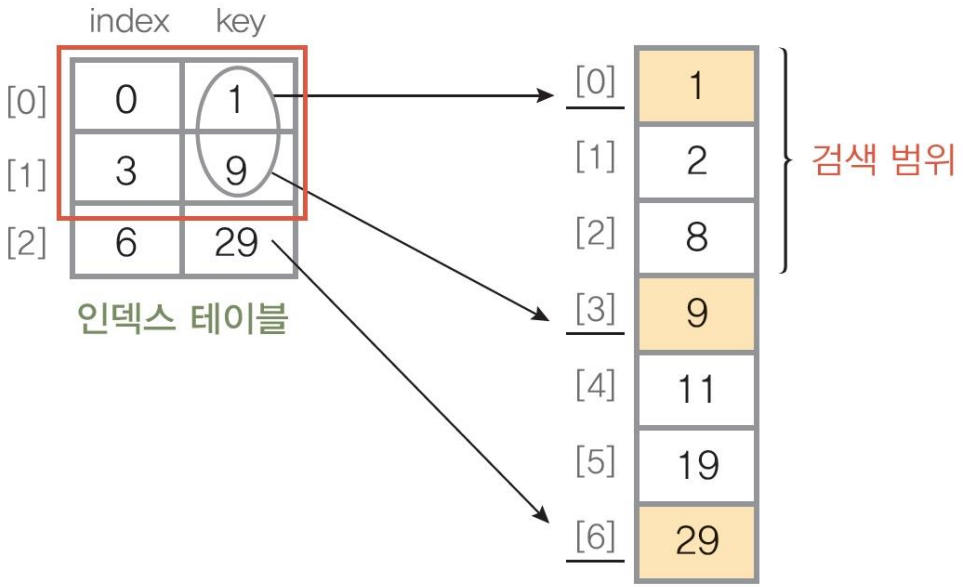


그림 10-3 색인 순차 검색의 예

2. 순차 검색

- 색인 순차 검색 알고리즘

알고리즘 10-3 정렬된 자료의 색인 순차 검색

```
indexSearch(a[], n, key)
    for (i ← 0; i < m; i ← i+1) do
        if ((indexTable[i].key <= key) and (indexTable[i + 1].key > key)) then {
            begin ← indexTable[i].index;
            end ← indexTable[i + 1].index;
            break;
    }
    sequentialSearch2(a[], begin, end, key);
end indexSearch()
```

2. 순차 검색

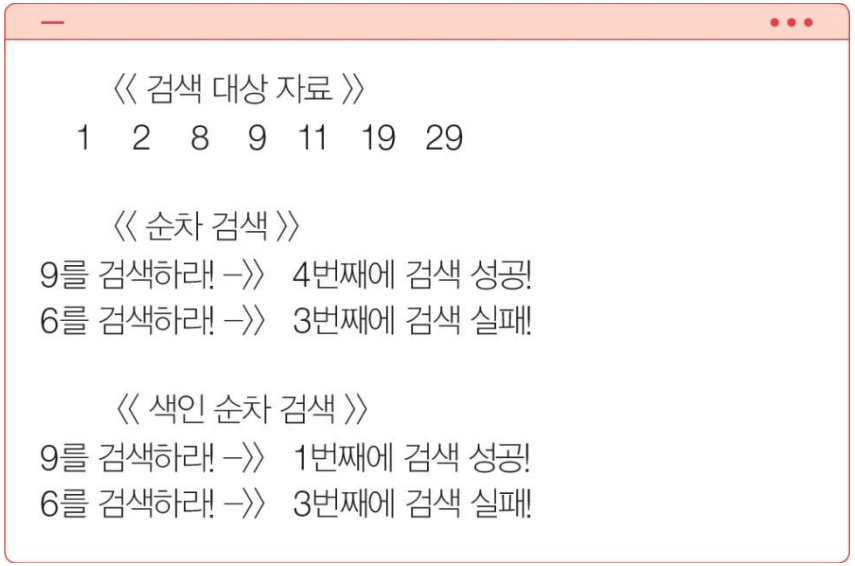
- 색인 순차 검색을 위해 인덱스 테이블 알고리즘

알고리즘 10-4 인덱스 테이블

```
makeIndexTable(a[], n, key)
    n ← size / m;                          // 인덱스 테이블에 들어가는 배열 원소의 간격 계산
    if (size mod  m > 0) then n ← n + 1;
    for (i ← 0; i< m; i ← i +1) do {       // 인덱스 테이블 채우기
        indexTable[i].index ← i*n;
        indexTable[i].key ← a[i*n];
    }
end makeIndexTable()
```

2. 순차 검색

- [예제 10-3] 자료 색인 순차 검색하기 : 교재 586p
- 실행 결과



2. 순차 검색

- 색인 순차 검색의 성능
  - 인덱스 테이블의 크기에 따라 결정
    - 인덱스 테이블의 크기가 줄어들면 배열의 인덱스를 저장하는 간격이 커지므로 배열에서 검색해야 하는 범위도 커짐
    - 인덱스 테이블의 크기를 늘리면 배열의 인덱스를 저장하는 간격이 작아지므로 배열에서 검색해야 하는 범위는 작아지겠지만 인덱스 테이블을 검색하는 시간이 늘어나게 됨
  - 색인 순차 검색의 시간 복잡도 : O(m + n/m)
    - 배열의 크기 : n, 인덱스 테이블의 크기 : m

3. 이진 검색

- **이진 검색**binary search**, 이분 검색, 보간 검색**interpolation search
  - 자료의 가운데에 있는 항목을 키 값과 비교하여 다음 검색 위치를 결정하여 검색을 계속하는 방법
    - 찾는 키 값 > 원소의 키 값 : 오른쪽 부분에 대해서 검색 실행
    - 찾는 키 값 < 원소의 키 값 : 왼쪽 부분에 대해서 검색 실행
  - 키를 찾을 때까지 이진 검색을 순환적으로 반복 수행함으로써 검색 범위를 반으로 줄여가면서 더 빠르게 검색
  - 정복 기법을 이용한 검색 방법
    - 검색 범위를 반으로 분할하는 작업과 검색 작업을 반복 수행
  - 정렬되어있는 자료에 대해서 수행하는 검색 방법

3. 이진 검색

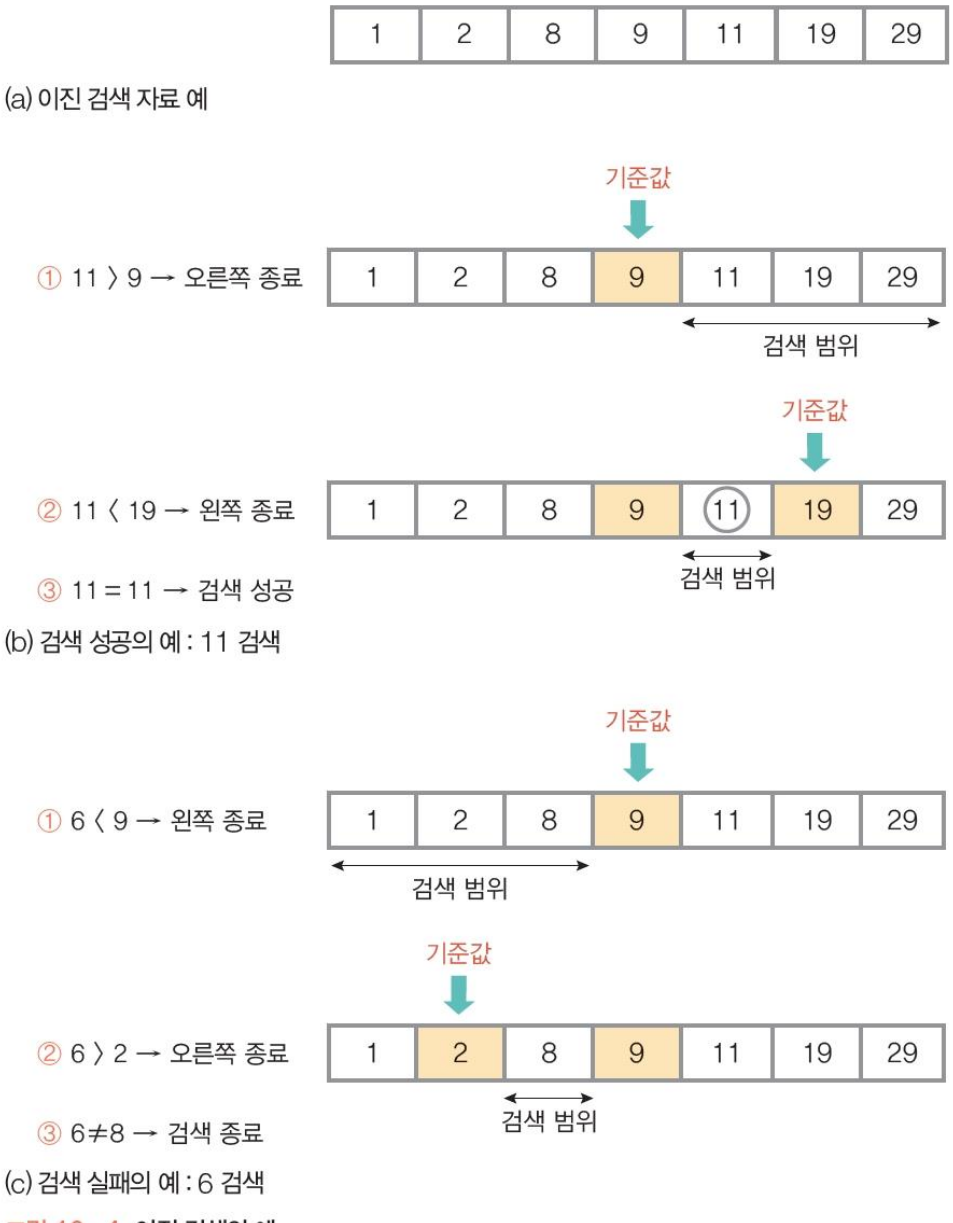


그림 10-4 이진 검색의 예

3. 이진 검색

- 이진 검색 알고리즘

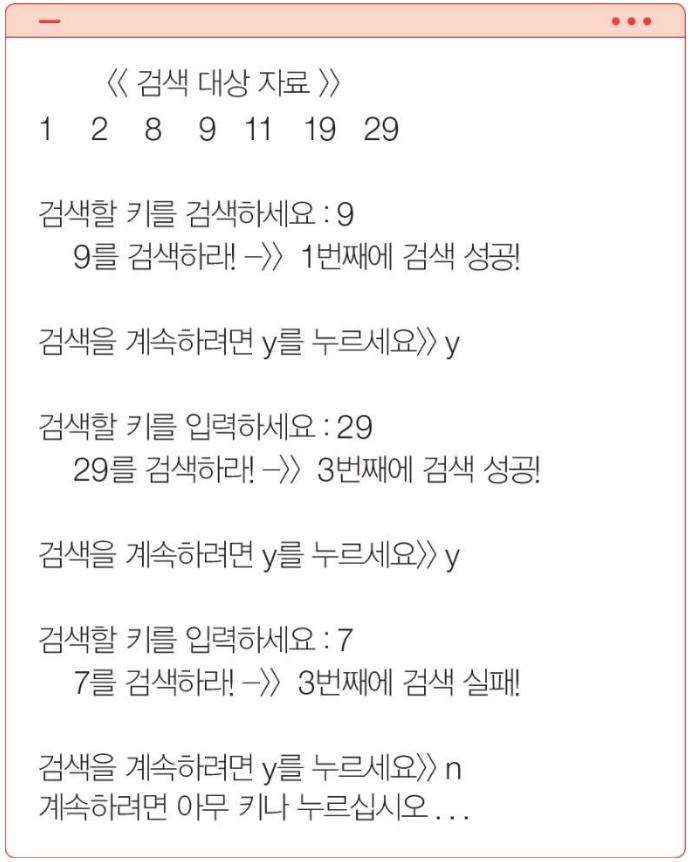
알고리즘 10-5 이진 검색

```
binarySearch(a[], begin, end, key)
    middle ←(begin + end) / 2;
    if (key = a[middle]) then return  1;
    else if (key < a[middle]) then binarySearch(a[], begin, middle - 1, key);
    else if (key > a[middle]) then binarySearch(a[], middle + 1, end, key);
    else return  -1;
end binarySearch()
```

- n개의 자료에 대한 이진 검색의 메모리 사용량은 n이 됨.
- 이진 검색에서 검색 범위를 1/2로 분할하면서 비교하는 연산에 대한 시간 복잡도는 $O(\log_2 n)$

3. 이진 검색

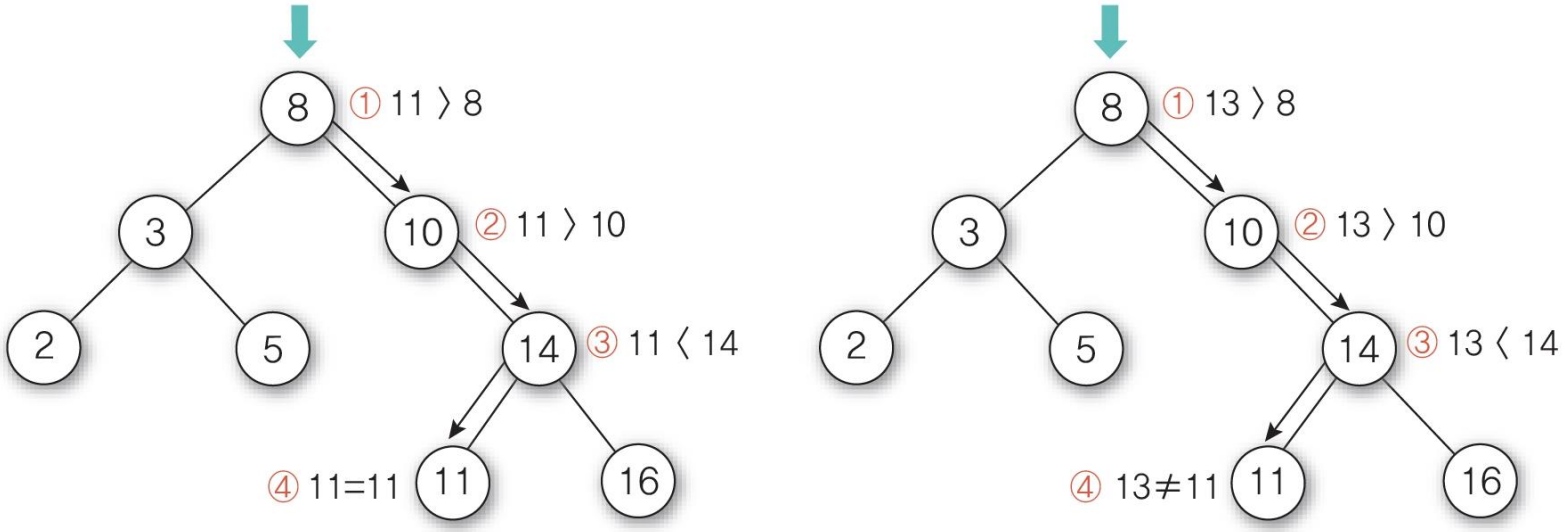
- [예제 10-4] 정렬된 자료를 이진 검색하기 : 교재 591p
- 실행 결과



4. 이진 트리 검색

❖ 이진 트리 검색binary tree search

- 7장에서 설명한 이진 탐색 트리를 사용한 검색 방법
- 원소의 삽입이나 삭제 연산에 대해서 항상 이진 탐색 트리를 재구성하는 작업 필요

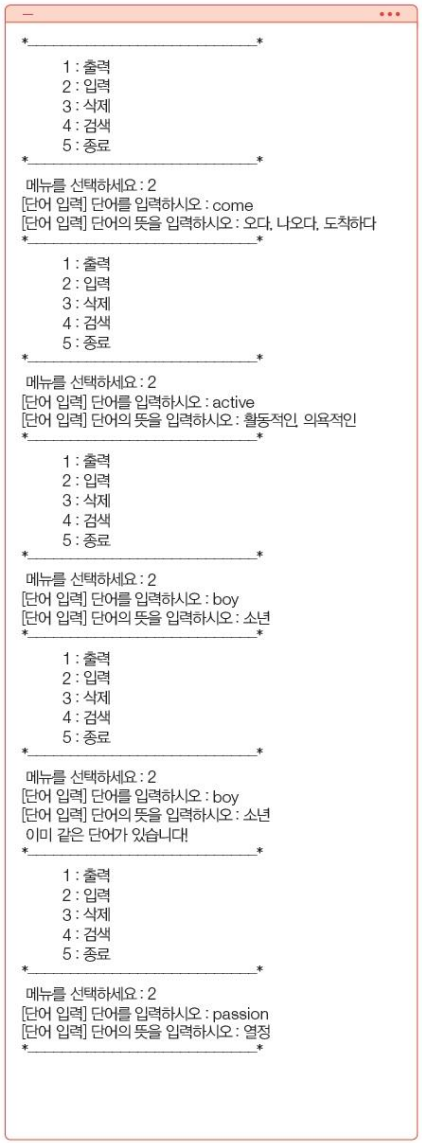


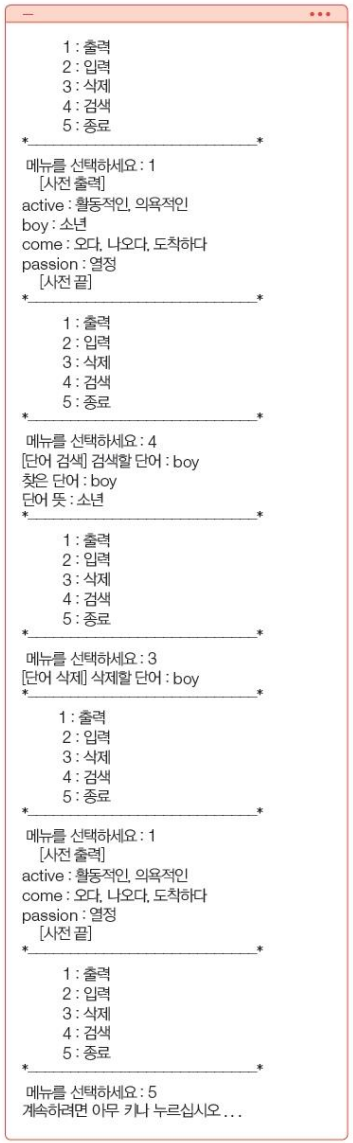
(a) 검색 성공의 경우 (b) 검색 실패의 경우

그림 10-5 이진 트리 검색의 예 : 11 검색과 13 검색

4. 이진 트리 검색

- [예제 10-5] 이진 검색을 이용해 영어 사전 구현하기 : 교재 593.p
- 실행 결과





5. 해싱

❖ 해싱hashing

- 산술적인 연산을 이용하여 키가 있는 위치를 계산하여 바로 찾아가는 계산 검색 방식
- 검색 방법
  - 키 값에 대해서 해시 함수를 계산하여 주소를 구하고,
  - 구한 주소에 해당하는 해시 테이블로 바로 이동
    - 해당 주소에 찾는 항목이 있으면 검색 성공, 없으면 검색 실패
- 해시 함수hashing function
  - 키 값을 원소의 위치로 변환하는 함수
- 해시 테이블hash table
  - 해시 함수에 의해 계산된 주소의 위치에 항목을 저장한 표

5. 해싱

- 해싱 검색 수행 방법

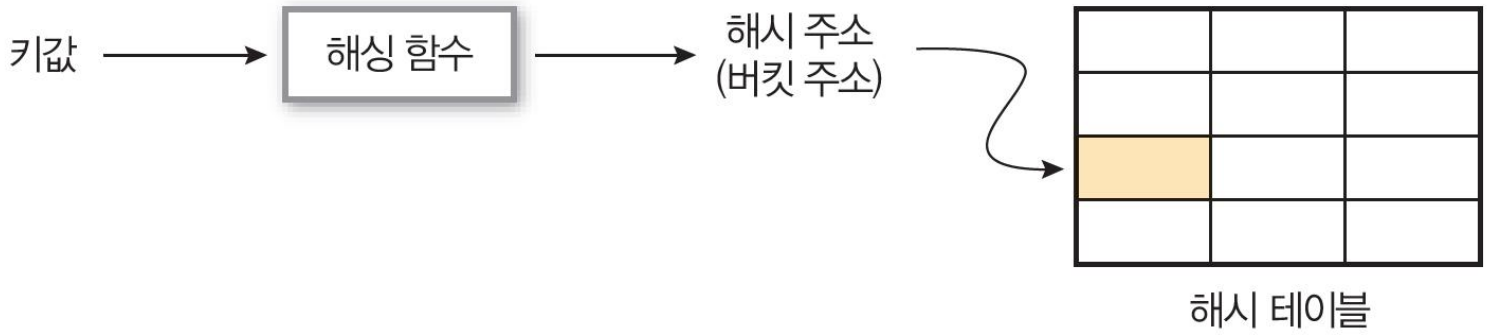
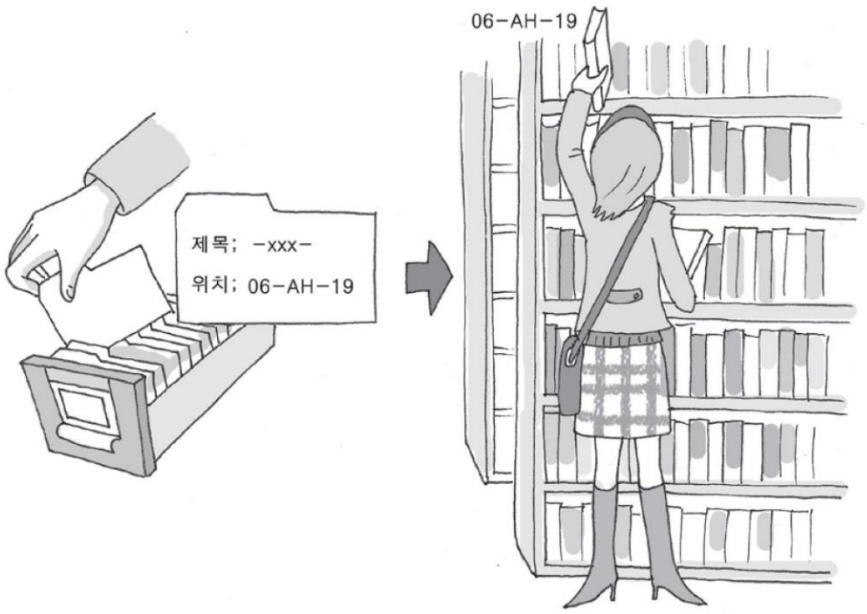


그림 10-6 해싱 검색의 수행 방법

5. 해싱

- 해싱의 예
  - 도서관에서의 도서 검색



(a) 위치 번호를 보고 책 찾기

(b) 도서 검색 시스템으로 책 찾기

그림 10-7 해싱의 예

5. 해싱

- 해싱의 예
  - 강의실 좌석 배정

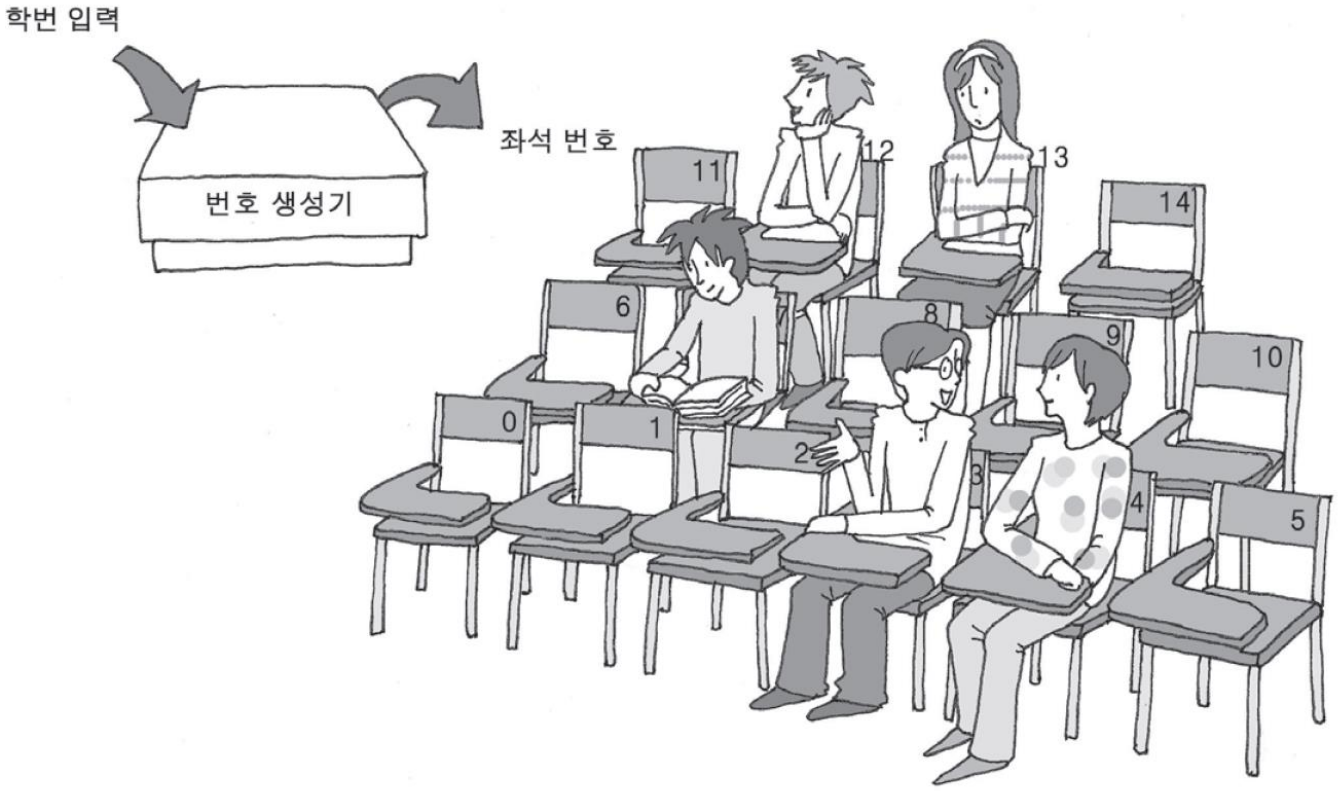


그림 10-8 해싱의 예 : 학번에 따라 강의실 좌석 배정

5. 해싱

❖ 해싱 관련 용어

- **동거자 synonym**
  - 서로 다른 키 값을 가지지만 해싱 함수에 의해 같은 버킷에 저장된 키 값들
- **충돌collision**
  - 서로 다른 키 값에 대해서 해싱 함수에 의해 주어진 버킷 주소를 같은 경우
  - 충돌이 발생한 경우에 비어있는 슬롯에 동거자 관계로 키 값 저장
  - 오버플로 : 버킷에 비어있는 슬롯이 없는 포화 버킷 상태에서 충돌이 발생하여 해당 버킷에 키 값을 저장할 수 없는 상태

그림 10-9 동거자의 예 : 같은 좌석에 앉은 짝꿍

그림 10-10 오버플로의 예

5. 해싱

키 값 밀도와 적재 밀도

키 값 밀도

- 사용 가능한 전체 키 값들 중에서 현재 해시 테이블에 저장되어서 실제 사용되고 있는 키 값의 개수 정도
- 키값 밀도 = 실제 사용 중인 키값의 개수 / 사용 가능한 키값의 개수

적재 밀도

- 해시 테이블에 저장 가능한 키 값의 개수 중에서 현재 해시 테이블에 저장되어서 실제 사용되고 있는 키 값의 개수 정도
- 적재 밀도 = 실제 사용 중인 키값의 개수 / 해시 테이블에 저장 가능한 전체 키값의 개수
  = 실제 사용 중인 키값의 개수 / (버킷 개수 × 슬롯 개수)

5. 해싱

❖ 해시 함수

- 해시 함수의 조건
  - 해시 함수는 계산이 쉬워야 함
    - 비교 검색 방법을 사용하여 키 값의 비교연산을 수행하는 시간보다 해싱 함수를 사용하여 계산하는 시간이 빨라야 해싱 검색을 사용하는 의미가 됨
  - 해시 함수는 충돌이 적어야 함
    - 충돌이 많이 발생한다는 것은 같은 버킷을 할당 받는 키 값이 많다는 것이므로 비어있는 버킷이 많은데도 어떤 버킷은 오버플로가 발생할 수 있는 상태가 되므로 좋은 해시 함수가 될 수 없음
  - 해시 테이블에 고르게 분포할 수 있도록 주소를 만들어야 함

5. 해싱

- 중간 제곱 함수
  - 키 값을 제곱한 결과 값에서 중간에 있는 적당한 비트를 주소로 사용
  - 제곱한 값의 중간 비트들은 대개 키의 모든 값과 관련이 있기 때문에 서로 다른 키 값은 서로 다른 중간 제곱 함수 값을 갖게 됨

  - (예) 키 값 00110101 10100111에 대한 해시 주소 구하기

```
                          00110101 10100111
X                         00110101 10100111
-------------------------------------------
000010110011 11101001 001011110001
```

해시 주소

5. 해싱

- 제산Division 함수
  - 함수는 나머지 연산자 mod(C에서의 %연산자)를 사용하는 방법
  - 키 값 k를 해시 테이블의 크기 M으로 나눈 나머지를 해시 주소로 사용
  - M으로 나눈 나머지 값은 0~(M-1)이 되므로 해시 테이블의 인덱스로 사용
  - 해시 주소는 충돌 발생 없이 고르게 분포하도록 생성되어야 하므로 키 값을 나누는 해시 테이블의 크기 M은 적당한 크기의 소수prime number 사용

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

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

5. 해싱

- 접지 함수
  - 키의 비트 수가 해시 테이블 인덱스의 비트 수보다 큰 경우에 주로 사용
  - 이동 접지 함수
    - 각 분할 부분을 이동시켜서 오른쪽 끝자리가 일치하도록 맞추고 더하는 방법
    - (예) 해시 테이블 인덱스가 3자리이고 키 값 k가 12312312312인 경우

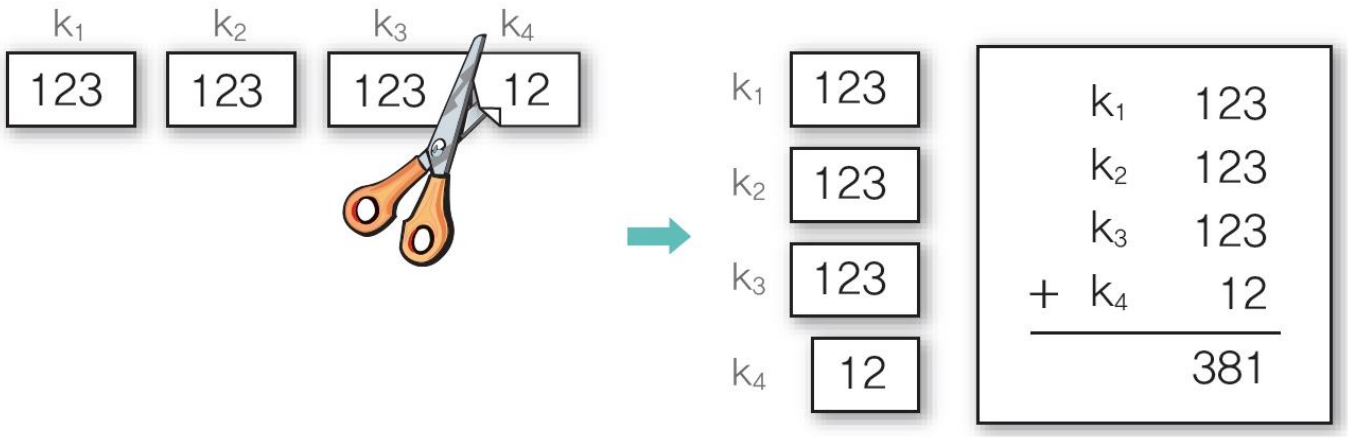


그림 10-11 이동 접지 함수의 사용 예

5. 해싱

- 경계 접지 함수
  - 분할된 각 경계를 기준으로 접으면서 서로 마주보도록 배치하고 더하는 방법
  - (예) 해시 테이블 인덱스가 3자리이고 키 값 k가 12312312312인 경우

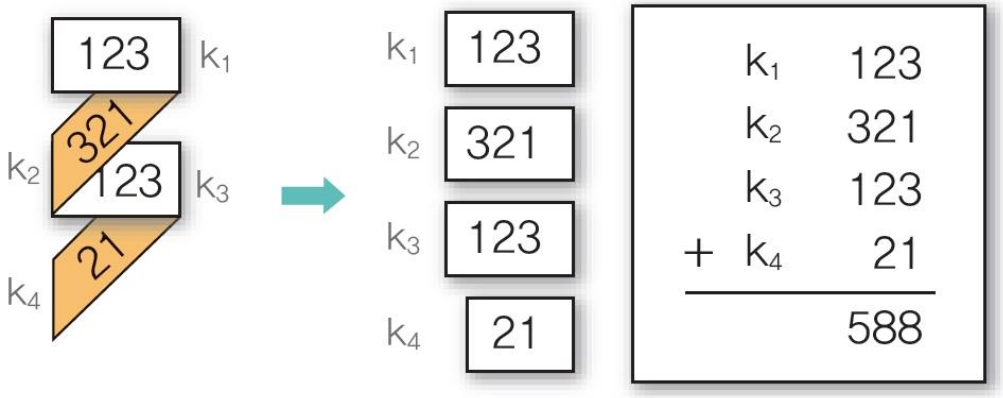


그림 10-12 경계 접지 함수의 사용 예

5. 해싱

- 숫자 분석 함수
  - 키 값을 이루고 있는 각 자릿수의 분포를 분석하여 해시 주소로 사용
  - 각 키 값을 적절히 선택한 진수로 변환한 후에 각 자릿수의 분포를 분석하여 가장 편중된 분산을 가진 자릿수는 생략하고, 가장 고르게 분포된 자릿수부터 해시 테이블 주소의 자릿수만큼 차례로 뽑아서 만든 수를 역순으로 바꾸어서 주소로 사용

  - (예) 키 값이 학번이고 해시 테이블 주소의 자릿수가 3자리인 경우

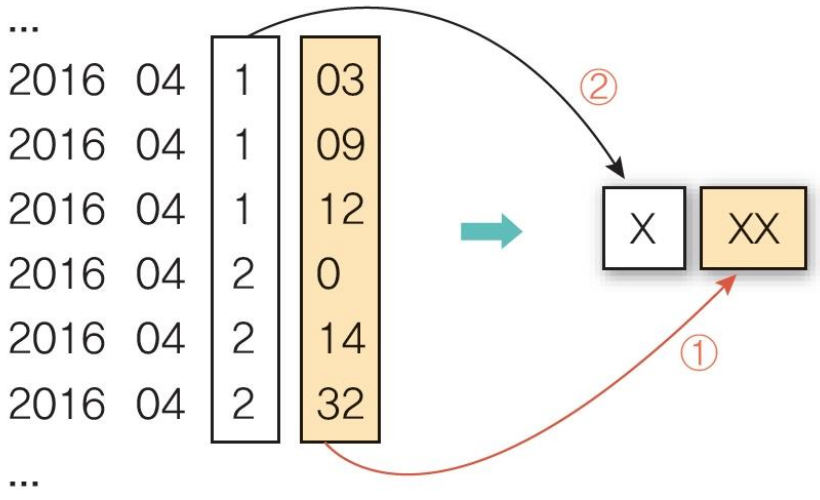


그림 10-13 숫자 분석 함수의 사용 방법

5. 해싱

- 진법 변환 함수
  - 키 값이 10진수가 아닌 다른 진수일 때, 10진수로 변환하고 해시 테이블 주소로 필요한 자릿수만큼만 하위자리의 수를 사용하는 방법
- 비트 추출 함수
  - 해시 테이블의 크기가 2^k일 때 키 값을 이진 비트로 놓고 임의의 위치에 있는 비트들을 추출하여 주소로 사용하는 방법
  - 이 방법에서는 충돌이 발생할 가능성이 많으므로 테이블의 일부에 주소가 편중되지 않도록 키 값들의 비트들을 미리 분석하여 사용해야 함

5. 해싱

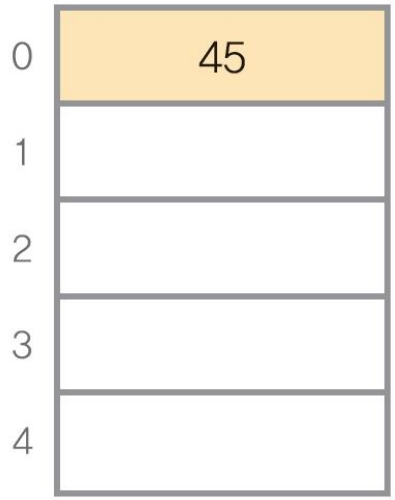
❖ 해싱에서 오버플로를 처리 방법

- 선형 개방 주소법(선형 조사법linear probing)
  - 해싱 함수로 구한 버킷에 빈 슬롯이 없어서 오버플로가 발생하면, 그 다음 버킷에 빈 슬롯이 있는지 조사
    - 빈 슬롯이 있으면 - 키 값을 저장
    - 빈 슬롯이 없으면 - 다시 그 다음 버킷을 조사
    - 이런 과정을 되풀이 하면서 해시 테이블 내에 비어있는 슬롯을 순차적으로 찾아서 사용하여 오버플로 문제를 처리하는 방법

5. 해싱

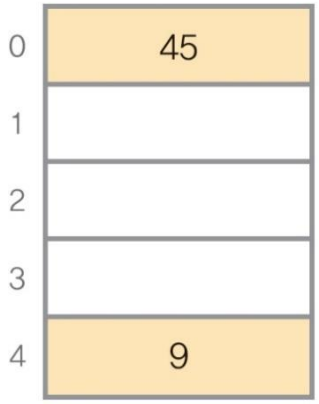
- 선형 개방 주소법을 이용한 오버플로 처리 예
  - 해시 테이블의 크기 : 5
  - 해시 함수 : 제산함수 사용. 해시 함수 h(k) = k mod 5
  - 저장할 키 값 : {45, 9, 10, 96, 25}

① h(45) = 45 mod 5 = 0 → 해시 테이블 0번에 키값 45 저장

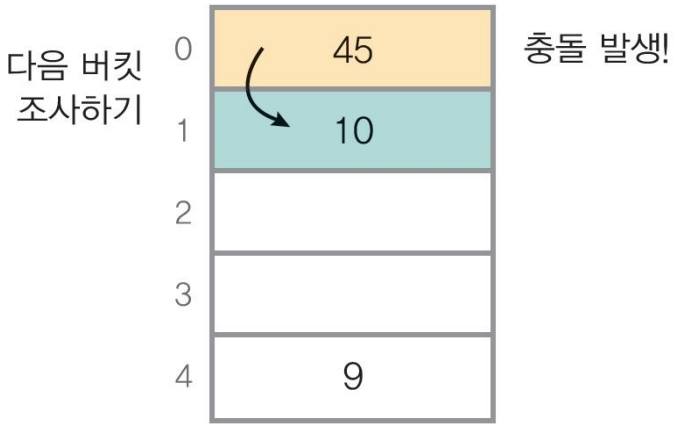


5. 해싱

② h(9) = 9 mod 5 = 4 → 해시 테이블 4번에 키값 9 저장

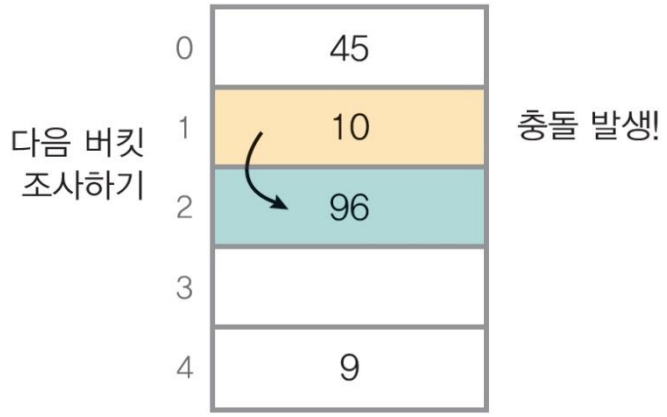


③ h(10) = 10 mod 5 = 0 → 충돌 발생! → 다음 버킷 중에서 비어 있는 버킷 1에 키값 10을 저장

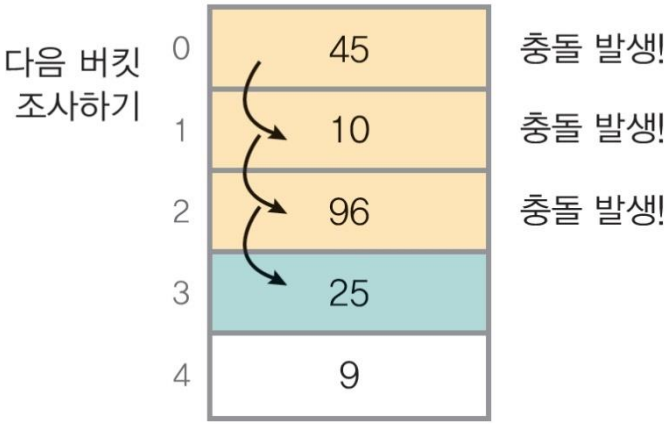


5. 해싱

④ h(96) = 96 mod 5 = 1 → 충돌 발생! → 다음 버킷 중에서 비어 있는 버킷 2에 키값 96을 저장



⑤ h(25) = 25 mod 5 = 0 → 충돌 발생! → 다음 버킷 중에서 비어 있는 버킷 3 에 키값 25를 저장



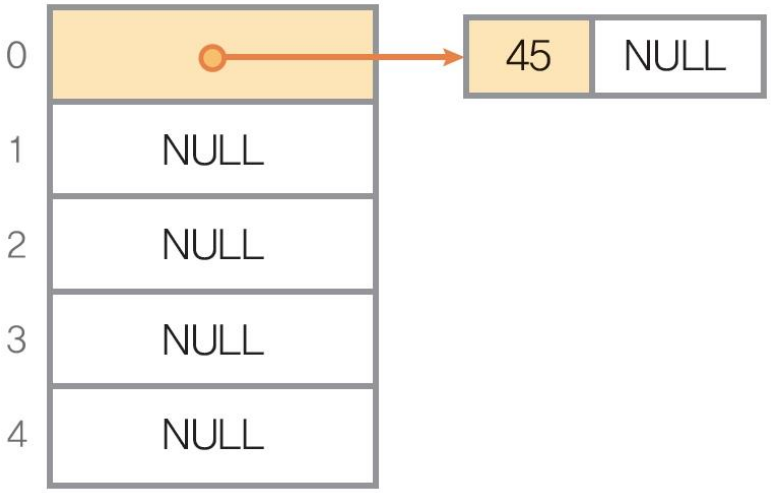
5. 해싱

- 체이닝
  - 해시 테이블의 구조를 변경하여 각 버킷에 하나 이상의 키 값을 저장할 수 있도록 하는 방법
  - 버킷에 슬롯을 동적으로 삽입하고 삭제하기 위해서 연결 리스트 사용
    - 각 버킷에 대한 헤드 노드를 1차원 배열로 만들고 각 버킷에 대한 헤드노드는 슬롯들을 연결 리스트로 가지고 있어서 슬롯의 삽입이나 삭제 연산을 쉽게 수행 가능
    - 버킷 내에서 원하는 슬롯을 검색하기 위해서는 버킷의 연결 리스트를 선형 검색

5. 해싱

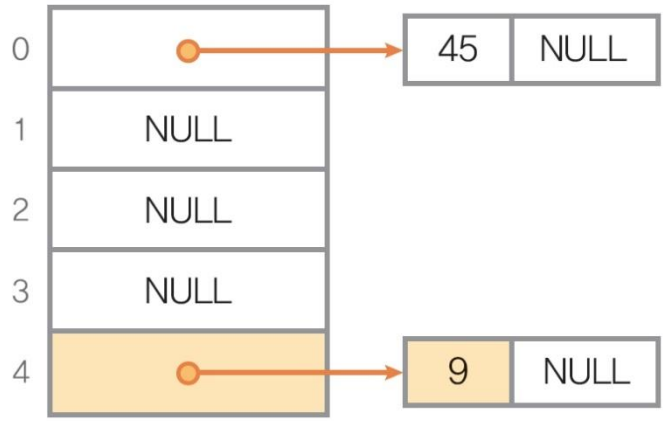
- 체이닝을 이용한 오버플로 처리 예
  - 해시 테이블의 크기 : 5
  - 해시 함수 : 제산함수 사용. 해시 함수 h(k) = k mod 5
  - 저장할 키 값 : {45, 9, 10, 96, 25}

① h(45) = 45 mod 5 = 0 → 해시 테이블 0번에 노드를 삽입, 키값 45를 저장

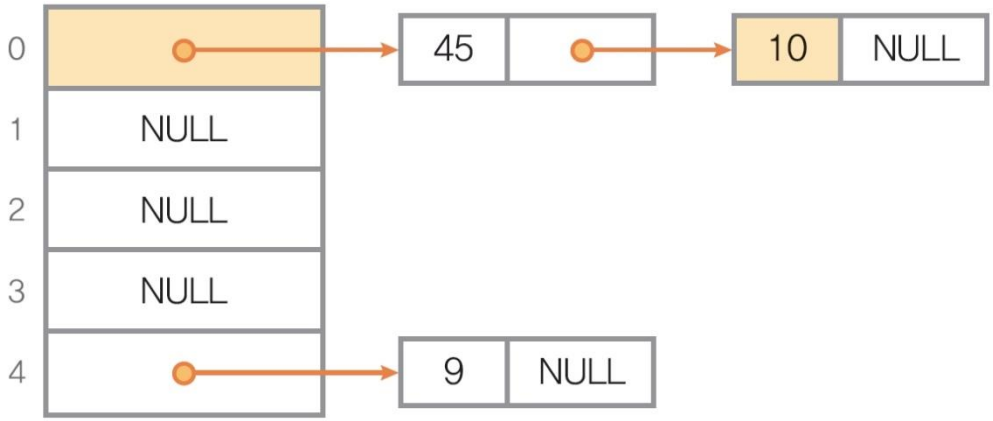


5. 해싱

② h(9) = 9 mod 5 = 4 → 해시 테이블 4번에 노드를 삽입, 키값 9를 저장

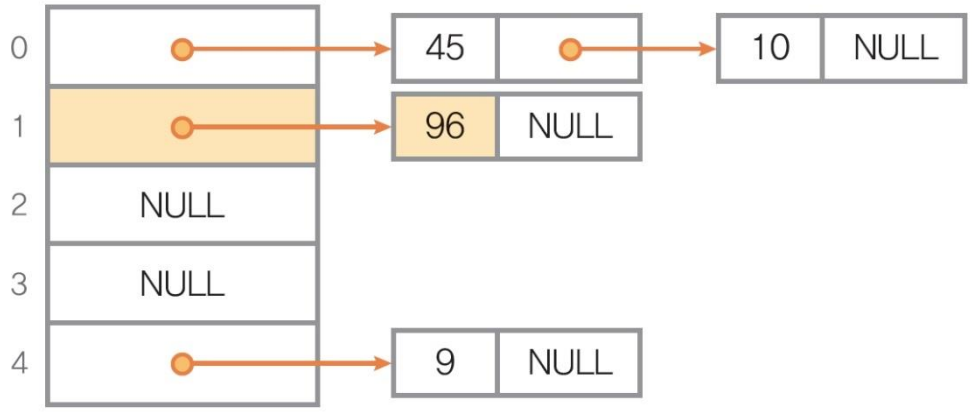


③ h(10) = 10 mod 5 = 0 → 해시 테이블 0번에 노드를 삽입, 키값 10을 저장



5. 해싱

④ h(96) = 96 mod 5 = 1 → 해시 테이블 1번에 노드를 삽입, 키값 96을 저장



⑤ h(25) = 25 mod 5 = 0 → 해시 테이블 0번에 노드를 삽입, 키값 25를 저장