KOREATECH
한국기술교육대학교

07장. 해시 테이블

Youn-Hee Han
LINK@KOREATECH
http://link.koreatech.ac.kr

그에게서 배운 것이 아니라
이미 내 속에 있었던 것이 그와 공명을 한 것이다.

-머레이 겔만

머리 겔만

75개 언어 펼쳐짐 접힘

위키백과, 우리 모두의 백과사전.

머리 겔만(영어: Murray Gell-Mann IPA: [ˈmʌriː ˈgɛl ˈmæn], 1929년 9월 15일 ~ 2019년 5월 24일)은 미국의 물리학자이다. 기묘도, 팔정도, 쿼크 등의 발견에 공헌하였고, 1969년에 노벨 물리학상을 수상하였다.

목차 [숨기기]

생애 [편집]

15세에 예일 대학교에 입학하고 19세에 MIT 대학원에 들어가 21살에 박사학위를 받았다. 입자물리학 이론 영역에서의 업적으로 40세인 1969년 노벨물리학상을 받았다. 1980년대 여러 분야의 연구자들이 모여 복잡계 현상을 연구하는 산타페 연구소를 설립하는데 참여했으며, 1987년부터 동 연구소에서 일했다.[1] 인류학, 언어학 등에도 관심을 보여 관련 연구 프로젝트를 주도하기도 했다.

저서 [편집]

- 겔만, 머리. 《스트레인지 뷰티》. ISBN 89-88907-58-2.

각주 [편집]

1. ↑ 쿼크의 아버지, 언어학에 빠지다 동아사이언스, 2011년 10월 24일

겔만 (2007년 사진)

출생 1929년 9월 15일
미국 뉴욕주 맨해튼

학습 목표

- ◈ 해시 테이블의 발생 동기를 이해한다.
- ◈ 해시 테이블의 원리를 이해한다.
- ◈ 해시 함수 설계 원리를 이해한다.
- ◈ 충돌 해결 방법들과 이들의 장단점을 이해한다.
- ◈ 해시 테이블의 검색 성능을 분석할 수 있도록 한다.

해싱 (Hashing)
- 임의의 값을 더 짧은 길이의 값으로 변환하는 것
- 사전적 의미: 잘게 썰다.

01. 검색 효율의 극단

검색/저장/삭제의 복잡도

- ◈ 배열
  - 평균의 경우: $\Theta(n)$
- ◈ 이진검색트리
  - 최악의 경우: $\Theta(n)$
  - 평균의 경우: $\Theta(logn)$
- ◈ 균형잡힌 이진검색트리(예: 레드블랙트리, AVL트리)
  - 최악/평균의 경우: $\Theta(logn)$
- ◈ B-트리
  - 최악/평균의 경우: $\Theta(logn)$
  - 실제적으로 균형잡힌 이진검색트리보다 효율적
- ◈ 해시 테이블
  - 평균의 경우: $\Theta(1)$

[해싱 (Hashing)]
하나의 문자열을 더 짧은 길이의 값으로 변환하는 것
사전적 의미: 잘게 썰다.

해시 테이블

◈ 해시 테이블 (Hash Table)

- 원소가 저장될 자리가 **원소의 값**에 의해 결정되는 자료구조
  - 이미 저장되어 있는 자료들과의 비교 과정 없음

- 해시의 장점
  - 해시 연산은 매우 짧게 수행
  - 평균 상수 시간(매우 빠른 시간)에 삽입, 삭제, 검색

- 해시 테이블의 크기 (Capacity): m
  - 저장할 수 있는 총 원소의 개수
  - 오른쪽 해시 테이블 예
    - ➢ 크기가 $m = 13$인 해시 테이블에 5개의 원소 저장
    - ➢ 사용된 해시 함수 $h(x) = x \ mod \ 13$

입력 : 25, 13, 16, 15, 7

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

해시 테이블

◈ 적재율 (Load Factor) α

- 해시 테이블의 크기가 m이고, 저장된 원소의 총수가 n일 때 해당 해시 테이블의 적재율 $\alpha = \frac{n}{m}$

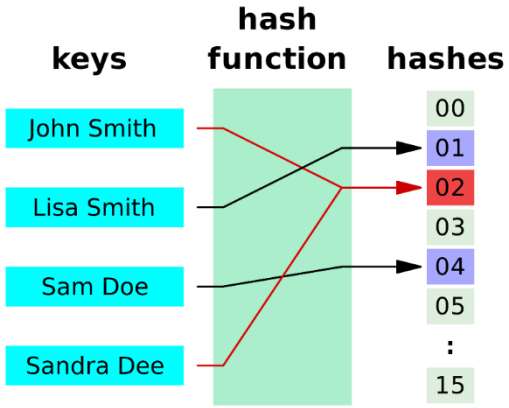
- 오른쪽 해시 테이블의 적재율

$$\alpha = \frac{n}{m} = \frac{5}{13} = 0.38$$

입력 : 25, 13, 16, 15, 7

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

해시 테이블



◈ 충돌 (Collison)

- 해시 테이블의 한 주소에 2 개 이상의 원소를 저장해야 하는 상황

- 오른쪽 해시 테이블에서의 충돌 예
  - 이미 저장되어 있는 원소 2
    - ➢ $h(2) = 2 \ mod \ 13 = 2$
  - 새롭게 저장될 원소 28
    - ➢ $h(28) = 28 \ mod \ 13 = 2$

- 충돌 해결(처리)는 해시 테이블 공부의 핵심

입력 : 25, 13, 16, 15, 7

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

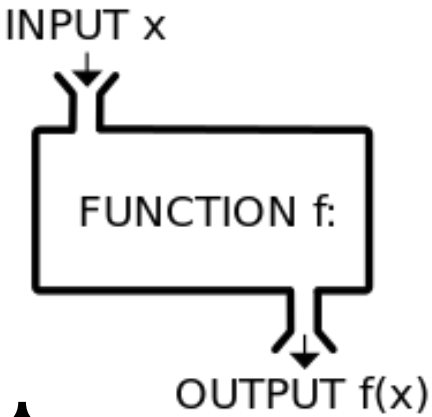
02. 해시 함수

해시 함수

◈ 함수 (Function)

- 두 변수 x, $f(x)$에 대하여 x가 주어지면 그에 따라 $f(x)$의 값이 오직 하나의 값으로 결정될 때, $f(x)$를 x의 함수라고 정의한다.

저장해야 할 원소



저장 주소

◈ 해시 함수 (Hash Function)

- x: 해시 테이블에 저장해야 할 원소
- $f(x)$: 저장해야 할 원소의 x의 해시 테이블 내 주소

◈ 해시 함수에 바라는 성질

- 계산이 간단해야 함
- 입력 원소가 해시 테이블 전체에 골고루 저장되어야 함
  - 즉, 충돌이 적어야 함

해시 함수 – 나누기 방법

◈ 나누기 방법 (Division Method)

$$h(x) = x \ mod \ m$$

- m: 해시 테이블의 크기(Capacity)
  - 해시 테이블 내 주소: $0, 1, 2, \dots, m-1$

- 적절한 m의 선택 방법
  - 2의 멱수(2^p)와 많이 떨어져 있는 소수(Prime Number)
    - 예 $m = 1{,}601$ → $h(1{,}025{,}390) = 1{,}025{,}390 \ mod \ 1{,}601 = 750$
  - 만약 $m = 2^4 = 16$라면?
    - $h(64) = 64 \ mod \ 16 = 0$
    - $h(128) = 128 \ mod \ 16 = 0$
    - …
    - $h(8{,}192) = 8{,}192 \ mod \ 16 = 0$
    - …

많은 충돌 발생

해시 함수 – 곱하기 방법

◈ 곱하기 방법 (Multiplication Method)

$$h(x) = \lfloor m(xA \mod 1) \rfloor$$

- m: 해시 테이블의 크기(Capacity)
  - 보통 2의 멱수로 정함

- A: 저장할 원소 x를 소수(decimal)로 대응시키 위하여 미리 정해 놓은 실수 $(0 < A < 1)$

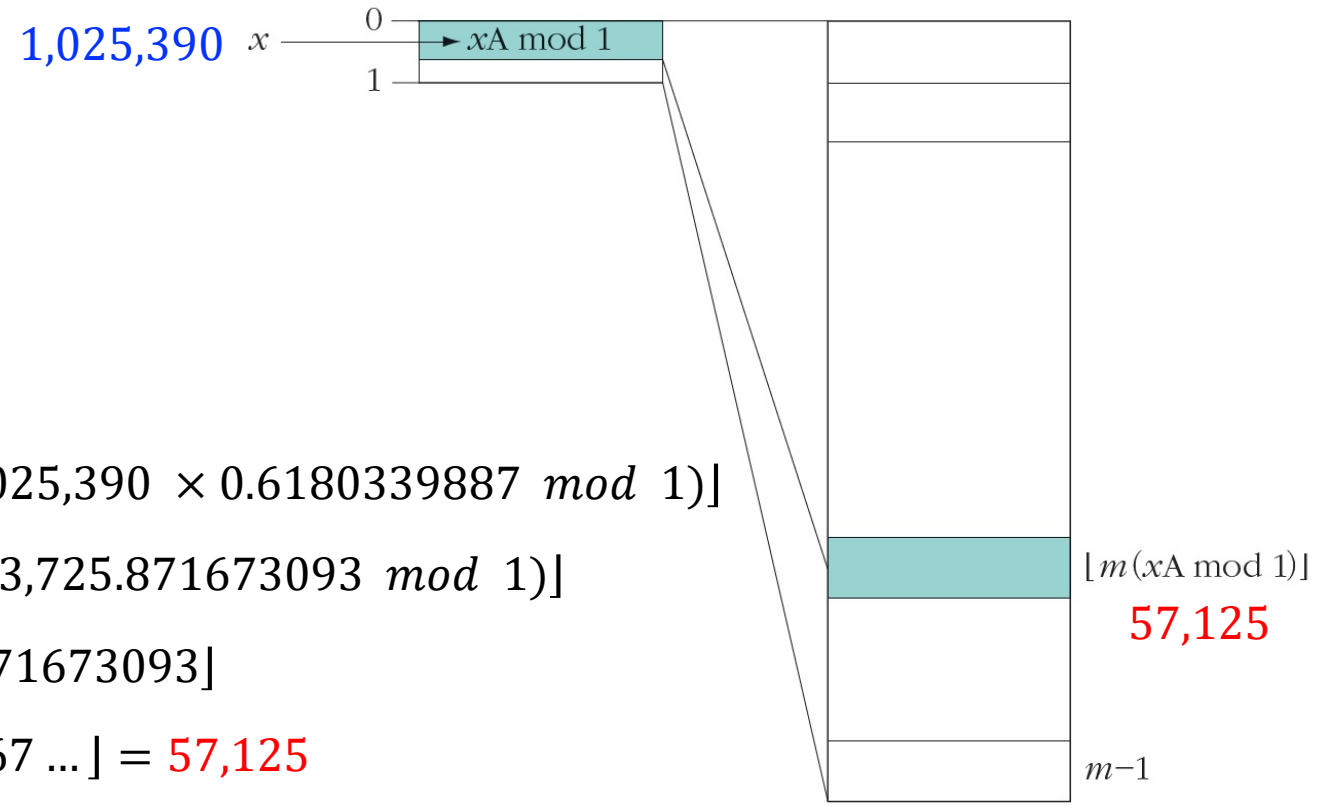
- $xA \mod 1$: x에 A를 곱한 다음 소수부만 취함

해시 함수 – 곱하기 방법

◈ 곱하기 방법 (Multiplication Method)

$$h(x) = \lfloor m(xA \mod 1) \rfloor$$

– 예 $[m = 2^{16} = 65{,}536,\ A = 0.6180339887]$



$$h(1{,}025{,}390) = \lfloor 2^{16} \times (1{,}025{,}390 \times 0.6180339887 \mod 1) \rfloor$$

$$= \lfloor 2^{16} \times (633{,}725.871673093 \mod 1) \rfloor$$

$$= \lfloor 2^{16} \times 0.871673093 \rfloor$$

$$= \lfloor 57{,}125.967 \ldots \rfloor = 57{,}125$$

그림 7–2 곱하기 방법의 작동 과정을 보여주는 예

03. 충돌 해결

충돌 해결

◈ 충돌 (Collision)

- 해시 테이블의 한 주소를 놓고 두 개 이상의 원소가 자리를 다투는 것

입력: 25, 13, 16, 15, 7

29 삽입 시도

$h(29) = 29 \ mod \ 13 = 3$

이미 다른 원소가 차지하고 있음

0	13
1	
2	15
3	16
4	
5	
6	
7	7
8	
9	
10	
11	
12	25

해시함수

$h(x) = x \ mod \ 13$

◈ 두 가지 대표적인 충돌 해결 방법

- 체이닝Chaining
  - 리스트 추가 공간 사용
  - 해시 테이블의 각 주소가 리스트의 헤더 역할 수행
  - 이 리스트에 해당 주소로 들어오는 원소들이 차례로 저장됨

- 개방주소 방법Open Addressing
  - 추가 공간 사용없이 해시 테이블 내부에서 충돌 해결
  - 충돌 발생시 해시 테이블의 다른 자리로 찾아 들어감

체이닝

◈ 체이닝

- 같은 주소로 해싱되는 원소를 모두 하나의 연결 리스트로 관리
- 해시 테이블 크기가 m이면 추가적인 연결 리스트 m개 필요

알고리즘 7-1 체이닝을 사용하는 해시 테이블에서 작업

```
chainedHashInsert(T[], x)
▷ T : 해시 테이블, x : 삽입 원소
{
    리스트 T[h(x)]의 맨 앞에 x를 삽입;
}

chainedHashSearch(T[], x)
▷ T : 해시 테이블, x : 검색 원소
{
    리스트 T[h(x)]에서 x값을 가지는 원소를 검색;
}

chainedHashDelete(T[], x)
▷ T : 해시 테이블, x : 삭제 원소
{
    리스트 T[h(x)]에서 x의 노드를 삭제;
}
```

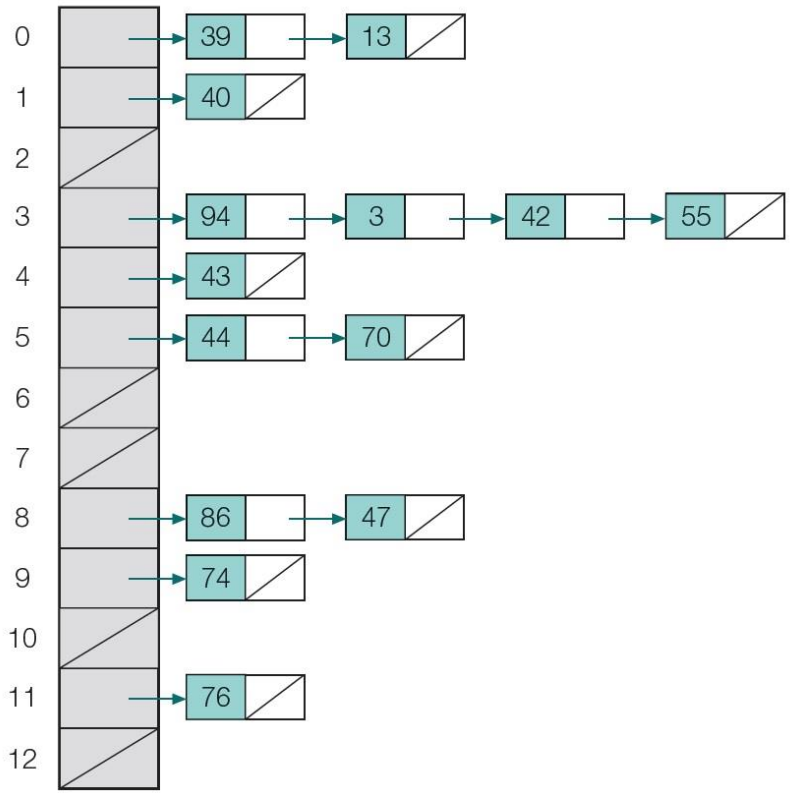
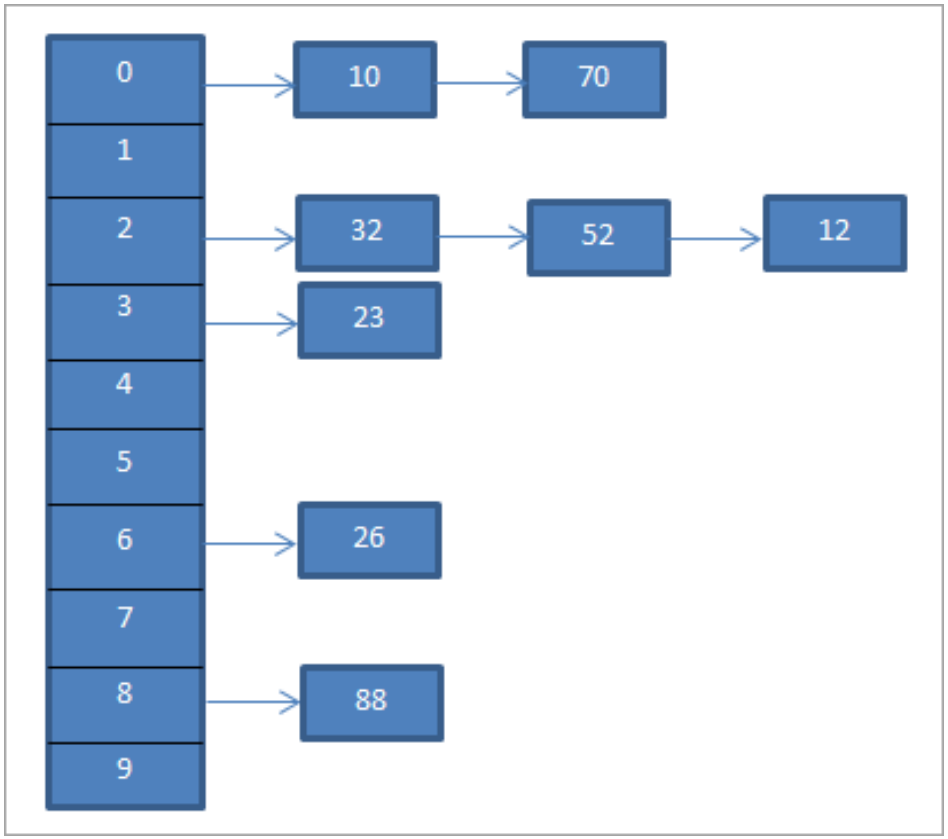


그림 7-3 체이닝을 이용한 충돌 해결을 보여주는 예

체이닝

◈ 체이닝



$h(x) = x \ \ mod \ \ 10$

개방 주소 방법

◈ 개방 주소 방법

- 연결 리스트와 같은 추가 공간 필요 없음
- 충돌 발생하면 어떻게든 주어진 테이블 공간에서 해결

◈ 개방 주소 방법에서 원소 x 삽입 과정

- 해시 함수 $h(x)(= h_0(x))$ 계산
- 충돌 발생이 없다면 해당 자리에 삽입
- 충돌 발생하면 순차적으로 다음 $h_1(x),\ h_2(x),\ h_3(x)$ … 해시 함수들을 사용하여 빈 자리의 주소를 찾고, 빈 자리가 나오면 해당 주소에 삽입

개방 주소 방법

◈ 다음 주소를 결정하는 대표적인 세 가지 방법

- 선형 조사 (Linear Probing)
  - 충돌이 발생한 바로 뒷 자리를 순차적으로 조사
- 이차원 조사 (Quadratic Probing)
  - 바로 뒷 자리를 조사하는 대신 보폭을 2차 함수를 사용하여 넓혀가면서 조사
- 더블 해싱 (Double Hashing)
  - 두 개의 해시 함수 $h(x)$와 $f(x)$ 사용

개방 주소 방법 – 선형 조사

◈ 선형 조사 (Linear Probing)

– 충돌이 발생한 바로 뒷 자리를 순차적으로 조사

$h(x) = h_0(x) = x \ mod \ m$ | $h_i(x) = (h(x) + i) \ mod \ m$ where $i = 1, 2, \ldots$

– **예** 입력 순서 25, 13, 16, 15, 7, 28, 31, 20, 1, 38

$h(x) = h_0(x) = x \ mod \ 13$ | $h_i(x) = (h(x) + i) \ mod \ 13$

$h_0(28) = 28 \ mod \ 13 = 2$

$h_1(28) = (h(28) + 1) \ mod \ 13$
$= 3 \ mod \ 13 = 3$

$h_2(28) = (h(28) + 2) \ mod \ 13$
$= 4 \ mod \ 13 = 4$

$h_0(20) = 20 \ mod \ 13 = 7$

$h_1(20) = (h(20) + 1) \ mod \ 13$
$= 8 \ mod \ 13 = 8$

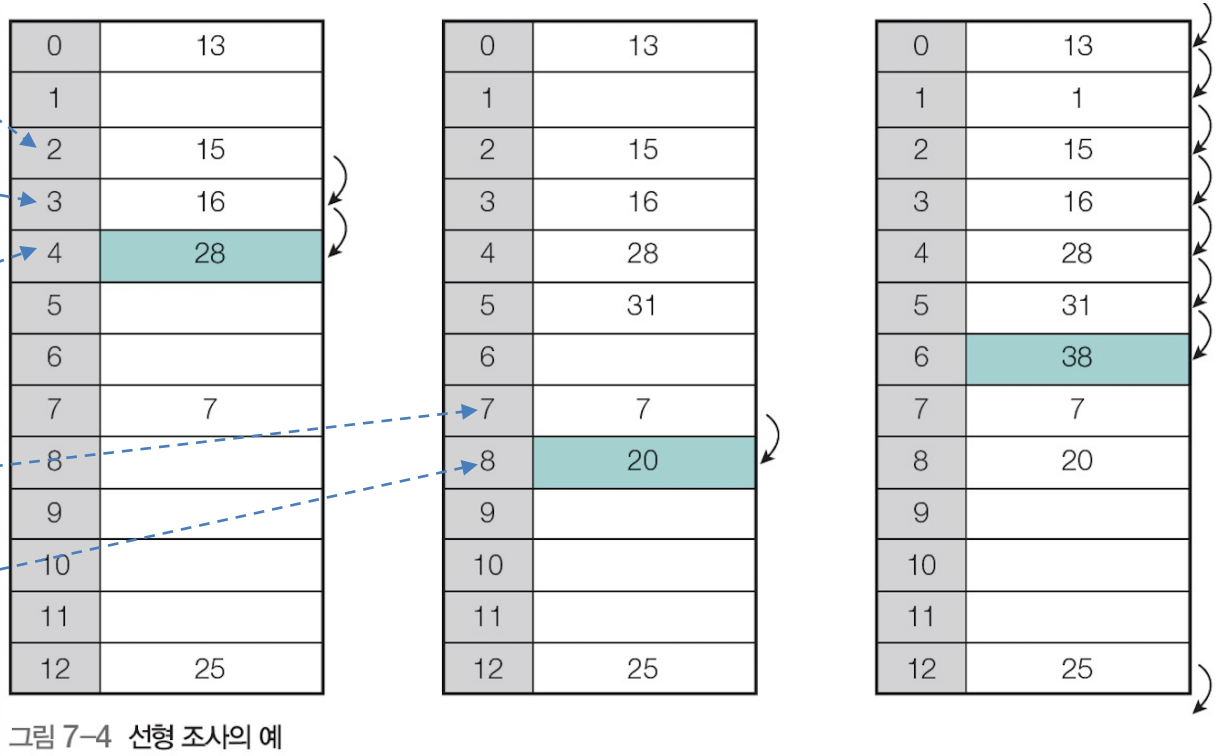


그림 7–4 선형 조사의 예

개방 주소 방법 – 선형 조사

◈ 선형 조사 (Linear Probing)

– 예 입력 순서 25, 13, 16, 15, 7, 28, 31, 20, 1, 38

$h(x) = h_0(x) = x \ mod \ 13$

$h_i(x) = (h(x) + i) \ mod \ 13$

$h_0(38) = 38 \ mod \ 13 = 12$

$h_1(38) = (h(38) + 1) \ mod \ 13$
$= 13 \ mod \ 13 = 0$

$h_2(38) = (h(38) + 2) \ mod \ 13$
$= 14 \ mod \ 13 = 1$

⋮

$h_7(38) = (h(38) + 7) \ mod \ 13$
$= 19 \ mod \ 13 = 6$

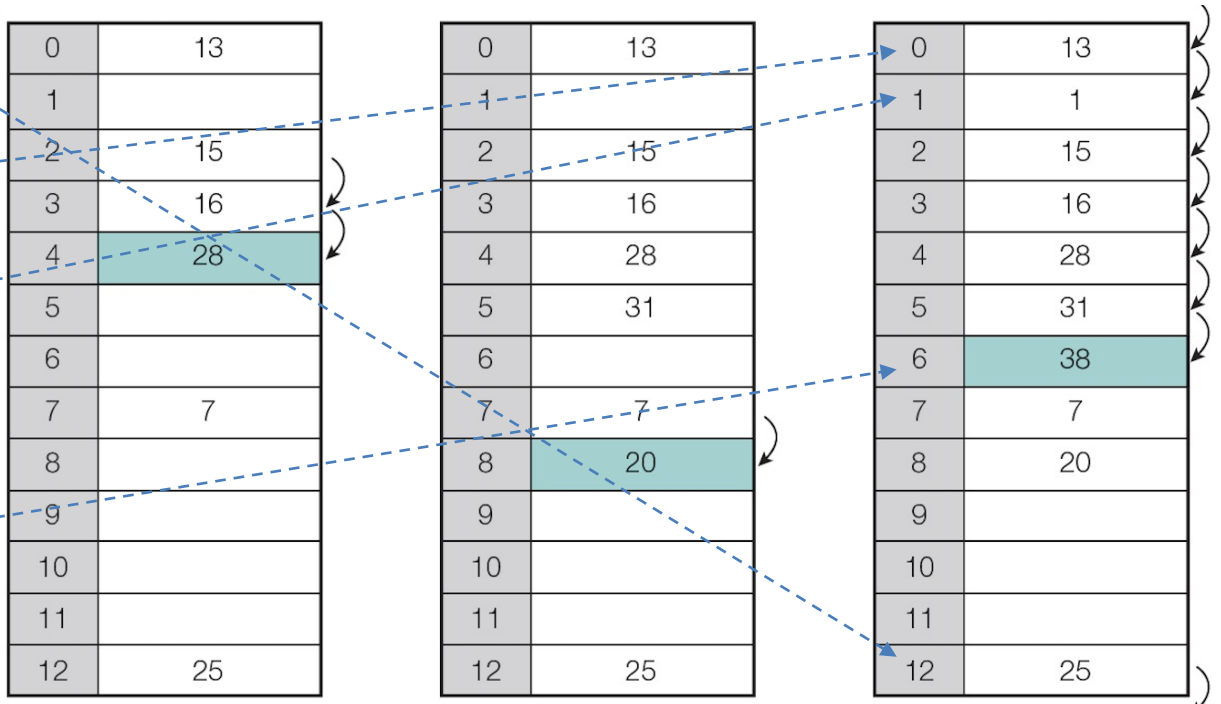


그림 7–4 선형 조사의 예

개방 주소 방법 – 선형 조사

◈ 선형 조사 (Linear Probing)의 단점

– 1차 군집(Primary Clustering) 현상 발생

- 여러 개의 원소가 동일한 초기 해시 함수 값 $h_0(x)$을 가지면 빈 자리를 찾기 위한 조사의 횟수가 너무 많아짐
  - $h_0(15) = 15 \ mod \ 13 = 2$
  - $h_0(16) = 16 \ mod \ 13 = 3$
  - $h_0(28) = 28 \ mod \ 13 = 2$ (Collision)
  - $h_0(31) = 31 \ mod \ 13 = 5$
  - $h_0(44) = 44 \ mod \ 13 = 5$ (Collision)
  - Next $h_0(41) = 41 \ mod \ 13 = 2$ (Collision)

0	
1	
2	15
3	16
4	28
5	31
6	44
7	
8	
9	
10	
11	37
12	

그림 7–5 1차 군집의 예

- 군집 영역이 크면 해당 영역으로 해싱될 확률이 커지므로 군집 영역의 크기가 클 수록 더욱 그 군집의 크기가 커진다.

개방 주소 방법 – 이차원 조사

◈ 이차원 조사 (Quadratic Probing)

– 바로 뒷 자리를 조사하는 대신 보폭을 이차 함수로 넓혀가면서 조사

$$h(x) = h_0(x) = x \ mod \ m$$

$$h_i(x) = (h(x) + c_1 i^2 + c_2 i) \ mod \ m \text{ where } i = 1, 2, \ldots$$

- 즉, $h(x)$, $h(x)+1$, $h(x)+4$, $h(x)+9$, $h(x)+16$, … 식으로 조사 [$c_1 = 1, c_2 = 0$]

– 선형 조사와 달리 특정 영역에 원소가 몰려도 그 영역을 빨리 벗어날 수 있다.

◈ 이차원 조사의 단점

– 2차 군집(Secondary Clustering) 현상 발생

- 여러 개의 원소가 동일한 초기 해시 함수 값 $h_0(x)$을 가지면 이 원소들에 대해 같은 순서로 조사를 수행하게 됨

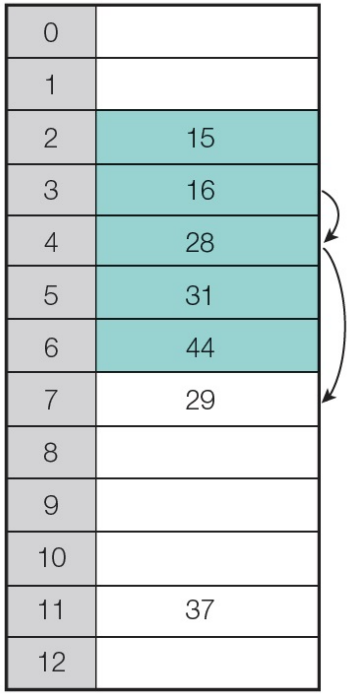


그림 7–6 1차 군집을 빨리 벗어나는 예

개방 주소 방법 – 이차원 조사

◈ 이차원 조사의 단점

– 예 $h(x) = h_0(x) = x \ mod \ 13$ $h_i(x) = (h(x) + i^2) \ mod \ 13$

$h(28) = 28 \ mod \ 13 = 2$

$h_1(28) = (h(28) + 1) \ mod \ 13 = 3 \ mod \ 13 = 3$

$h(41) = 41 \ mod \ 13 = 2$

$h_1(41) = (h(41) + 1) \ mod \ 13 = 3 \ mod \ 13 = 3$

$h_1(41) = (h(41) + 4) \ mod \ 13 = 6 \ mod \ 13 = 6$

$h(67) = 67 \ mod \ 13 = 2$

$h_1(67) = (h(67) + 1) \ mod \ 13 = 3 \ mod \ 13 = 3$

$h_2(67) = (h(67) + 4) \ mod \ 13 = 6 \ mod \ 13 = 6$

$h_3(67) = (h(67) + 9) \ mod \ 13 = 11 \ mod \ 13 = 11$

$h(54) = 54 \ mod \ 13 = 2$

$h_1(54) = (h(54) + 1) \ mod \ 13 = 3 \ mod \ 13 = 3$

$h_2(54) = (h(54) + 4) \ mod \ 13 = 6 \ mod \ 13 = 6$

$h_3(54) = (h(54) + 9) \ mod \ 13 = 11 \ mod \ 13 = 11$

$h_4(54) = (h(54) + 16) \ mod \ 13 = 18 \ mod \ 13 = 5$

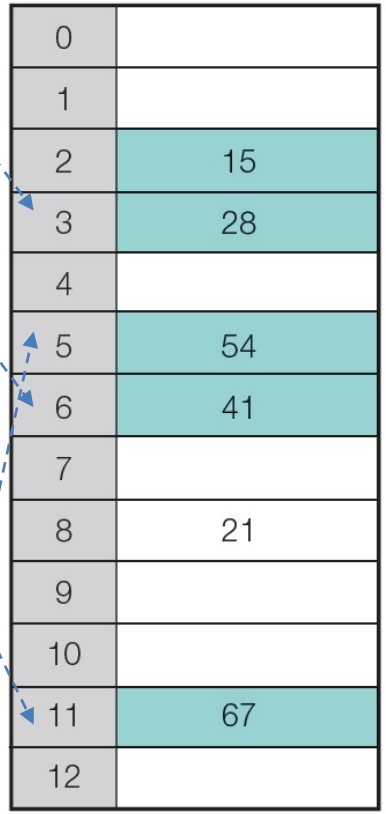


그림 7–7 2차 군집의 예

개방 주소 방법 – 더블 해싱

◈ 더블 해싱 (Double Hashing)

– 두 개의 해시 함수 $h(x)$와 $f(x)$ 사용

$$h(x) = h_0(x) = x \ mod \ m$$

$$h_i(x) = (h(x) + if(x)) \ mod \ m \text{ where } i = 1, 2, \ldots$$

– 임의의 두 원소의 첫번째 해시 함수 값이 동일해도, 두번째 해시 함수 값도 동일할 확률은 매우 작음

→ 서로 다른 보폭으로 주소 점프

→ 2차 군집 문제 발생하지 않음

개방 주소 방법 – 더블 해싱

◈ 더블 해싱 (Double Hashing)

$$h(x) = h_0(x) = x \ mod \ m$$

$$h_i(x) = (h(x) + if(x)) \ mod \ m \text{ where } i = 1, 2, \ldots$$

- [주의]: 두 함수 설정이 적당하지 않으면 해시 테이블 주소 중 일부 밖에 보지 못하는 현상 발생 가능
  - $h(x)$의 해시에 사용하는 m & $f(x)$의 해시에 사용하는 m'
  - m 과 m'은 서로소가 되어야 함
  - 권장하는 두 함수 설정 방법
    - m'는 소수 m보다 다소 작은 소수
    - 예: $m = 13, m' = 11$

$$h(x) = x \ mod \ m \qquad f(x) = 1 + (x \ mod \ m')$$

개방 주소 방법 – 더블 해싱

◈ 더블 해싱 (Double Hashing) 예

$h(x) = x \ mod \ 13$

$f(x) = 1 + (x \ mod \ 11)$

$h_i(x) = (h(x) + if(x)) \ mod \ 13$

$h_1(67) = \big(h(67) + f(67)\big) \ mod \ 13$
$= \big((67 \ mod \ 13) + 1 + (67 \ mod \ 11)\big) \ mod \ 13$
$= (2 + 1 + 1) \ mod \ 13 = 4 \ mod \ 13 = 4$

$h_1(28) = \big(h(28) + f(28)\big) \ mod \ 13$
$= \big((28 \ mod \ 13) + 1 + (28 \ mod \ 11)\big) \ mod \ 13$
$= (2 + 1 + 6) \ mod \ 13 = 9 \ mod \ 13 = 9$

$h_1(41) = \big(h(41) + f(41)\big) \ mod \ 13$
$= \big((41 \ mod \ 13) + 1 + (41 \ mod \ 11)\big) \ mod \ 13$
$= (2 + 1 + 8) \ mod \ 13 = 11 \ mod \ 13 = 11$

0	
1	
2	15
3	
4	67
5	
6	19
7	
8	
9	28
10	
11	41
12	

$h_0(15)=h_0(28)=h_0(41)=h_0(67)=2$

$h_1(67)=4$

$h_1(28)=9$

$h_1(41)=11$

그림 7-8 2차 군집에서 해방된 예

개방 주소 방법 정리

◈ 개방 주소 방법 의사 코드 (알고리즘 7-2)

- 삽입 과 검색

```
hashInsert(T[], x)
{
    i ← 0;
    repeat {
        j ← h_i(x);
        if (T[j] = NIL)
            then {T[j] ← x; return j;}
            else i++;
    } until (i = m)
    error "테이블 오버플로";
}
```

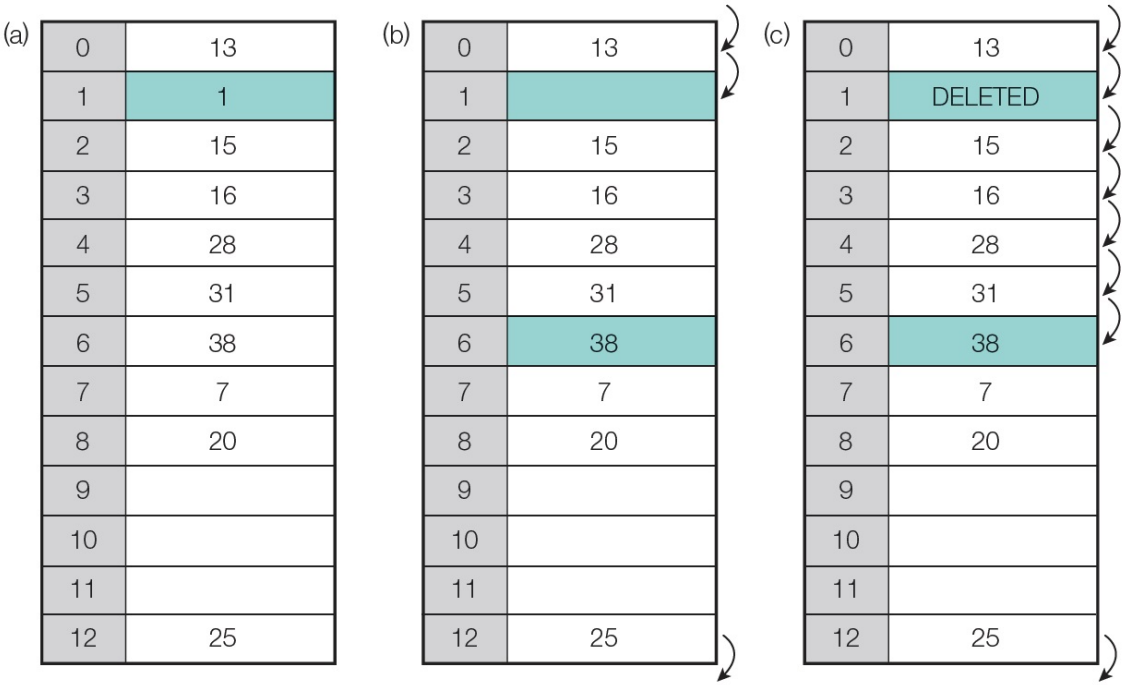
```
hashSearch(T[], x)
{
    i ← 0;
    repeat {
        j ← h_i(x);
        if (T[j] = x)
            then return j;
            else i++;
    } until (T[j] = NIL or i = m)
    return NIL;
}
```

- 개방 주소 방법은 적재율이 1을 넘을 수 없음
  - 적재율을 1이 넘도록 새로운 키 삽입 시도 → Rehashing 필요

개방 주소 방법 정리

◈ 개방 주소 방법 의사 코드 (알고리즘 7-2)

– 삭제



	(a)	(b)	(c)
0	13	13	13
1	1		DELETED
2	15	15	15
3	16	16	16
4	28	28	28
5	31	31	31
6	38	38	38
7	7	7	7
8	20	20	20
9			
10			
11			
12	25	25	25

그림 7-9 해시 테이블에서 자료가 삭제될 경우의 처리 방법

- 삭제된 자리를 비워두면 검색시 오류 발생
  - ➢ 삭제된 원소 자리에 "DELETED"와 같은 태그 저장 필요
- 새로운 원소 삽입시 "DELETED" 를 만나면 이 자리에 새로운 원소 저장

개방 주소 방법 정리

◈ 선형 조사 (Linear Probing) vs.
이차원 조사 (Quadratic Probing) vs.
더블 해싱 (Double Hashing)

- 아래 표는 load factor에 따른 개방 주소 방법의 평균 성공 조사 횟수와 실패 조사 횟수 비교 표이다.

	Linear probing		Quadratic		Double hashing	
LF	success	fail	success	fail	success	fail
10%	1.10	1.10	1.10	1.10	1.10	1.10
20%	1.10	1.30	1.10	1.30	1.10	1.20
30%	1.20	1.50	1.20	1.50	1.20	1.40
40%	1.30	1.90	1.30	1.80	1.70	1.70
50%	1.50	2.50	1.40	2.20	1.40	2.00
60%	1.80	3.60	1.60	2.80	1.50	2.50
70%	2.20	6.10	1.90	3.80	1.70	3.30
80%	3.00	13.00	2.20	5.80	2.00	5.00
90%	5.50	50.50	2.90	11.40	2.60	10.00
95%	10.50	200.50	3.50	22.00	3.20	20.00

04. 해시 테이블에서 검색 시간 분석

적재율과 Rehashing

◈ 적재율(Load Factor) α

- 체이닝의 적재율은 1 이상 가능
- 개방 주소 방법의 적재율는 항상 1 이하

◈ 적재율 관리와 "Rehashing"

- 적재율이 높아지면 해시 테이블의 효율이 줄어듦
- 따라서, 절대율이 미리 정한 임계점을 넘으면 해시 테이블의 크기를 대략 2배로 키우고 모든 원소를 다시 해싱하여 저장하는 작업 필요

체이닝 방법에서 검색 시간 분석

◈ 체이닝 방법 분석을 위한 가정

- 저장된 다른 원소와 관계없이 임의의 원소가 테이블상의 임의의 위치에 해시되어지는 확률은 동일하다고 가정

◈ [정리 7-1]

- 체이닝 방법 기반 해싱에서 적재율이 α일 때, 실패하는 검색의 평균 조사 횟수는 α이다.
  - 정확히는 $\max(1, \alpha)$
  - 오른쪽 해시 테이블의 적재율
    - 14 / 13 ≈ 1.076
    - 실패하는 검색의 평균 조사 횟수 : 1.076

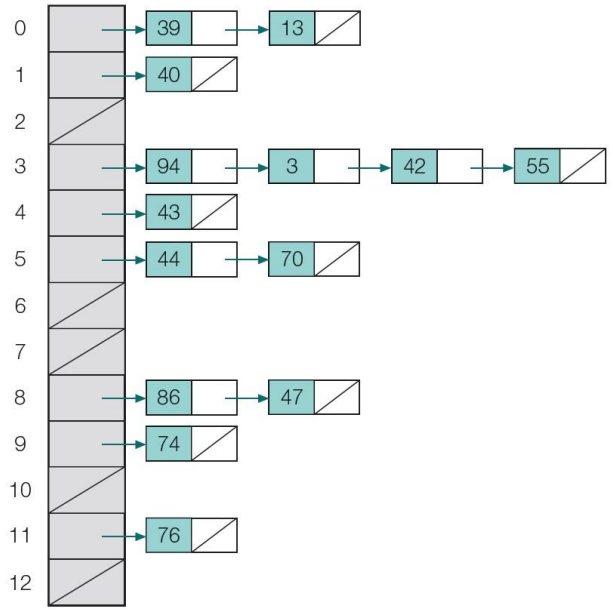


그림 7-3 체이닝을 이용한 충돌 해결을 보여주는 예

체이닝 방법에서 검색 시간 분석

◈ [정리 7-2]

– 체이닝 방법 기반 해싱에서 적재율이 α이고 저장된 원소의 개수가 n개일 때, 성공하는 검색의 평균 조사 횟수는 $\frac{1+\alpha}{2} + \frac{\alpha}{2n}$이다.

- $\alpha = 2, n = 10$ → $\frac{1+\alpha}{2} + \frac{\alpha}{2n} = 1.5 + 0.1 = 1.6$
- $\alpha = \frac{1}{2}, n = 10$ → $\frac{1+\alpha}{2} + \frac{\alpha}{2n} = 0.75 + 0.025 = 0.775$
- $\alpha = \frac{3}{4}, n = 10$ → $\frac{1+\alpha}{2} + \frac{\alpha}{2n} = 0.875 + 0.0375 = 0.9125$
- 오른쪽 해시 테이블의 적재율
  - 14 / 13 = 1.076
  - 성공하는 검색의 평균 조사 횟수 : $\frac{1+1.07}{2} + \frac{1.07}{28} \approx 1.035 + 0.038 \approx 1.073$

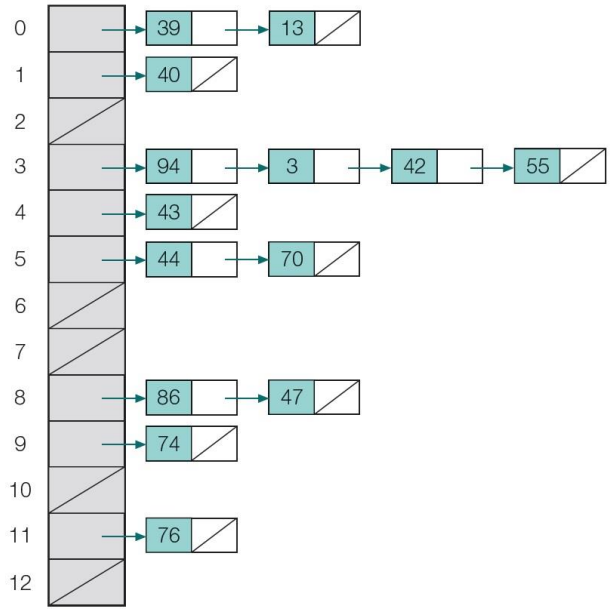


그림 7-3 체이닝을 이용한 충돌 해결을 보여주는 예

개방 주소 방법에서 검색 시간 분석

◈ 개방 주소 방법 분석을 위한 가정

- $h_0(x),\ h_1(x),\ h_2(x),\ h_3(x)$ …가 0부터 $m-1$사이의 수로 이루어진 순열을 이루고, 모든 순열은 같은 확률로 발생
- 즉, 조사시에 테이블의 모든 위치를 중복없이 한 번씩 볼 수 있음
- 조사하는 순서는 임의의 순서
- 더블 해싱이 선형 조사와 이차원 조사보다 위 가정에 가깝게 수행

◈ [정리 7-3]

- 개방 주소 방법을 이용하는 해싱에서 적재율이 $\alpha < 1$일 때, 실패하는 검색의 평균 조사 횟수는 최대 $\frac{1}{1-\alpha}$이다.
  - $\alpha = \frac{1}{2} \rightarrow \frac{1}{1-\alpha} = 2$
  - $\alpha = \frac{3}{4} \rightarrow \frac{1}{1-\alpha} = \frac{4}{1} = 4$

개방 주소 방법에서 검색 시간 분석

◈ [정리 7-4]

- 개방 주소 방법을 이용하는 해싱에서 적재율이 $\alpha < 1$일 때, 성공하는 검색의 평균 조사 횟수는 최대 $\frac{1}{\alpha} log \frac{1}{1-\alpha}$이다.
  - $\alpha = \frac{1}{2}$ → $\frac{1}{\alpha} log \frac{1}{1-\alpha} = 2log2 = 2$
  - $\alpha = \frac{3}{4}$ → $\frac{1}{\alpha} log \frac{1}{1-\alpha} = \frac{4}{3} log4 \approx 1.3 \times 2 \approx 2.6$

개방 주소 방법에서 검색 시간 분석

◈ 체이닝 vs. 개방 주소 방법

- 평균 조사 회수 측면에서 체이닝이 개방 주소 방법보다 효율적
  - 개방 주소 방법은 충돌을 일으키지 않은 원소라도 조사과정에서 확인
  - 체이닝은 충돌을 일으킨 원소들만 확인
  - 그렇다고, 개방 주소 방법의 조사 횟수가 매우 많은 것은 아님
    - ➢ 특히, 평균 조사 횟수가 Capacity 나 실제 저장된 원소의 개수와 무관한 점 주목

- 하지만 체이닝은 연결 리스트 추가 공간 필요

- 적재율이 높지 않으면 추가 공간 요구가 없고, 평균 조사 횟수도 그렇게 높지 않은 개방 주소 방법이 더 매력적
  - 따라서, 적재율을 높지 않게 유지하며 개방 주소 방법 활용 더 선호

Java와 Python에서 해시 테이블 사용

해시 테이블

◈ Java HashMap 사용

```
import java.util.HashMap;
import java.util.Map;

public class Number {
    public static void main(String[] args) throws Exception {
        Map mMap = new HashMap();

        mMap.put("John Smith", "521-1234");
        mMap.put("Lisa Smith", "521-8976");
        mMap.put("Sandra Dee", "521-9655");

        System.out.println("[Phone Numbers]");
        System.out.println("John Smith: " + mMap.get("John Smith"));
        System.out.println("Sandra Dee: " + mMap.get("Sandra Dee"));
    }
}
```

해시 테이블

◈ Java HashMap 사용

```
import java.util.*;
public class HashMapDemo {

   public static void main(String args[]) {

      // Create a hash map
      HashMap hm = new HashMap();

      // Put elements to the map
      hm.put("Zara", new Double(3434.34));
      hm.put("Mahnaz", new Double(123.22));
      hm.put("Ayan", new Double(1378.00));
      hm.put("Daisy", new Double(99.22));
      hm.put("Qadir", new Double(-19.08));

      // Get a set of the entries
      Set set = hm.entrySet();

      // Get an iterator
      Iterator i = set.iterator();

      // Display elements
      while(i.hasNext()) {
         Map.Entry me = (Map.Entry)i.next();
         System.out.print(me.getKey() + ": ");
         System.out.println(me.getValue());
      }
      System.out.println();

      // Deposit 1000 into Zara's account
      double balance = ((Double)hm.get("Zara")).doubleValue();
      hm.put("Zara", new Double(balance + 1000));
      System.out.println("Zara's new balance: " + hm.get("Zara"));
   }
}
```

해시 테이블

◈ Java HashMap 사용

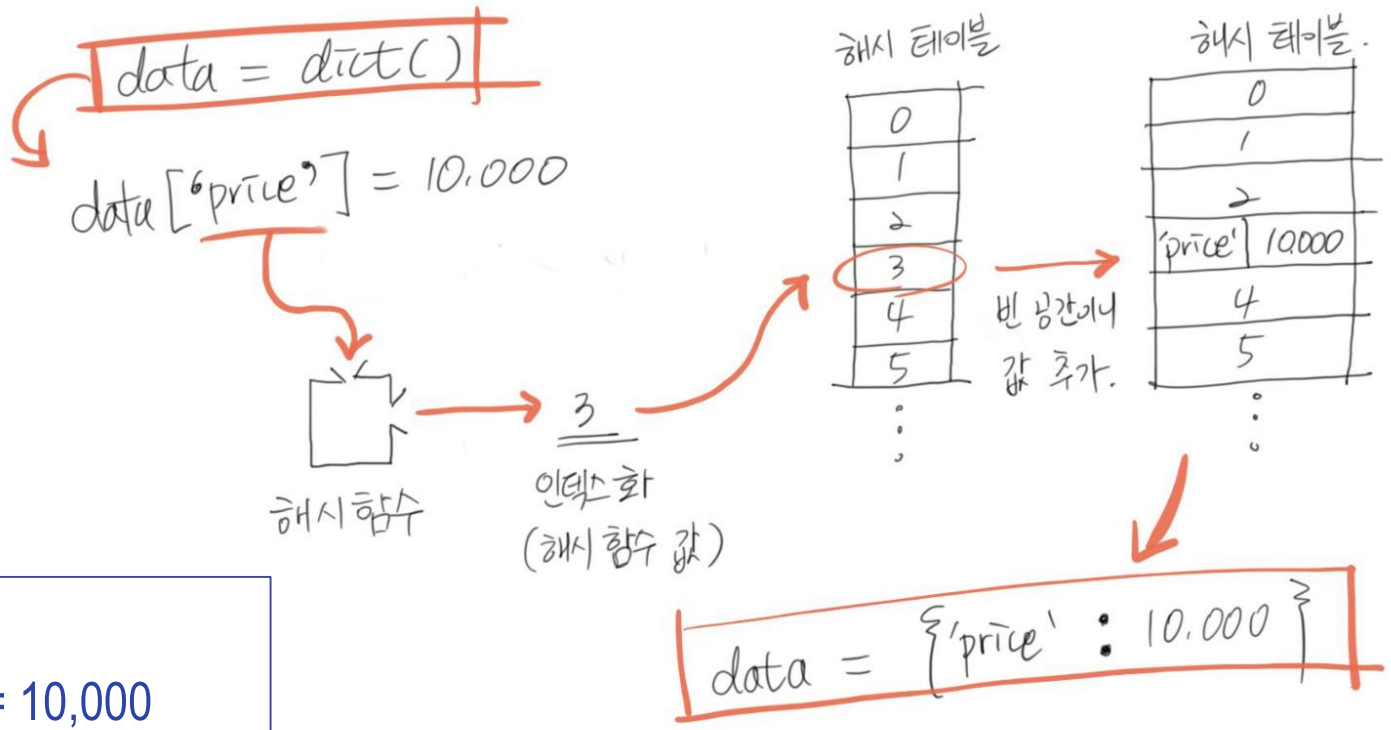
Constructors

Constructor and Description
`HashMap()` Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
`HashMap(int initialCapacity)` Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
`HashMap(int initialCapacity, float loadFactor)` Constructs an empty HashMap with the specified initial capacity and load factor.
`HashMap(Map<? extends K,? extends V> m)` Constructs a new HashMap with the same mappings as the specified Map.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/HashMap.html

해시 테이블

◈ Python 딕셔너리(dictionary) 사용




```
data = dict()
data['price'] = 10,000
```

해시 테이블

◈ Python 딕셔너리(dictionary) 자료구조

```
# 빈딕셔너리 생성
dict1 = {} # {}
dict2 = dict() # {}
```

```
# 특정 key-value쌍을 가진 dictionary 선언

Dog = {
    'name': '동동이',
    'weight': 4,
    'height': 100,
}


'''
{'height': 100, 'name': '동동이', 'weight': 4}
'''
```

```
# dictionary를 value로 가지는 dictionary 선언

Animals = {
    'Dog': {
        'name': '동동이',
        'age': '5'
    },
    'Cat': {
        'name': '야옹이',
        'weight': 3
    }
}


'''
 { 'Dog': { 'name': '동동이', 'age': '5'},
   'Cat': {'name': '야옹이','weight': 3 }}
'''
```

- https://yunaaaas.tistory.com/46

Questions & Answers