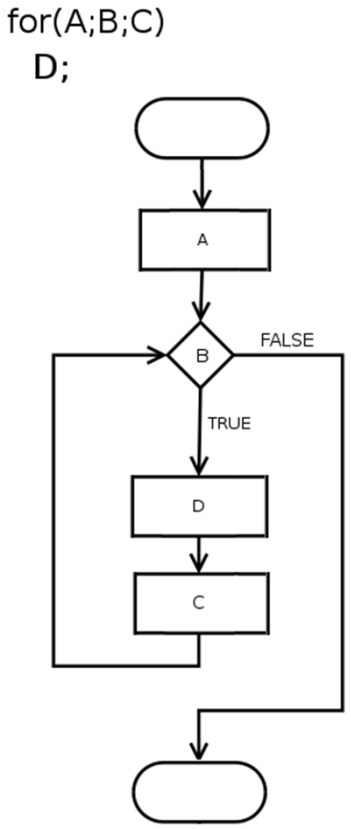
Clickseo.com

자료구조 & 알고리즘

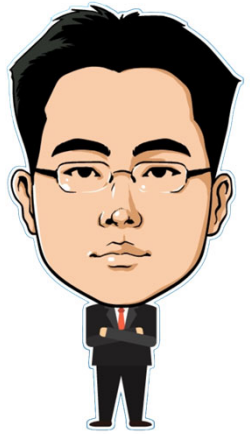


해시 테이블
(Hash Table)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



CC BY NC ND

목 차

cc BY NC ND

- 해시 테이블
- 충돌 해결

해시 테이블

CC BY NC ND

- **해시 테이블**
  - 해시 함수
- **충돌 해결**

해시 테이블 (1/3)

- **시간 복잡도**
  - **Θ(1)** 의 시간 작업을 원한다!!!
  - **배열 또는 연결 리스트:** O(n), 평균 Θ(n)
  - **이진 검색 트리:** 검색, 삽입, 삭제 시 평균 Θ(log n), 최악의 경우 Θ(n)
  - **균형 이진 검색 트리**
    - 검색, 삽입, 삭제 시 최악의 경우 Θ(log n)
    - AVL 트리, RB 트리
  - **균형 다진 검색 트리**
    - 검색, 삽입, 삭제 시 최악의 경우 Θ(log n)
    - 2-3 트리, 2-3-4 트리, B-트리
  - **해시 테이블**
    - **검색, 삽입, 삭제 시 평균 Θ(1)**

해시 테이블 (2/3)

입력 : 25, 13, 16, 15, 7

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

그림 12-1 해시 테이블 예

● 해시 테이블(Hash Tables)

❍ 키를 변환하여 배열의 인덱스로 사용한다.

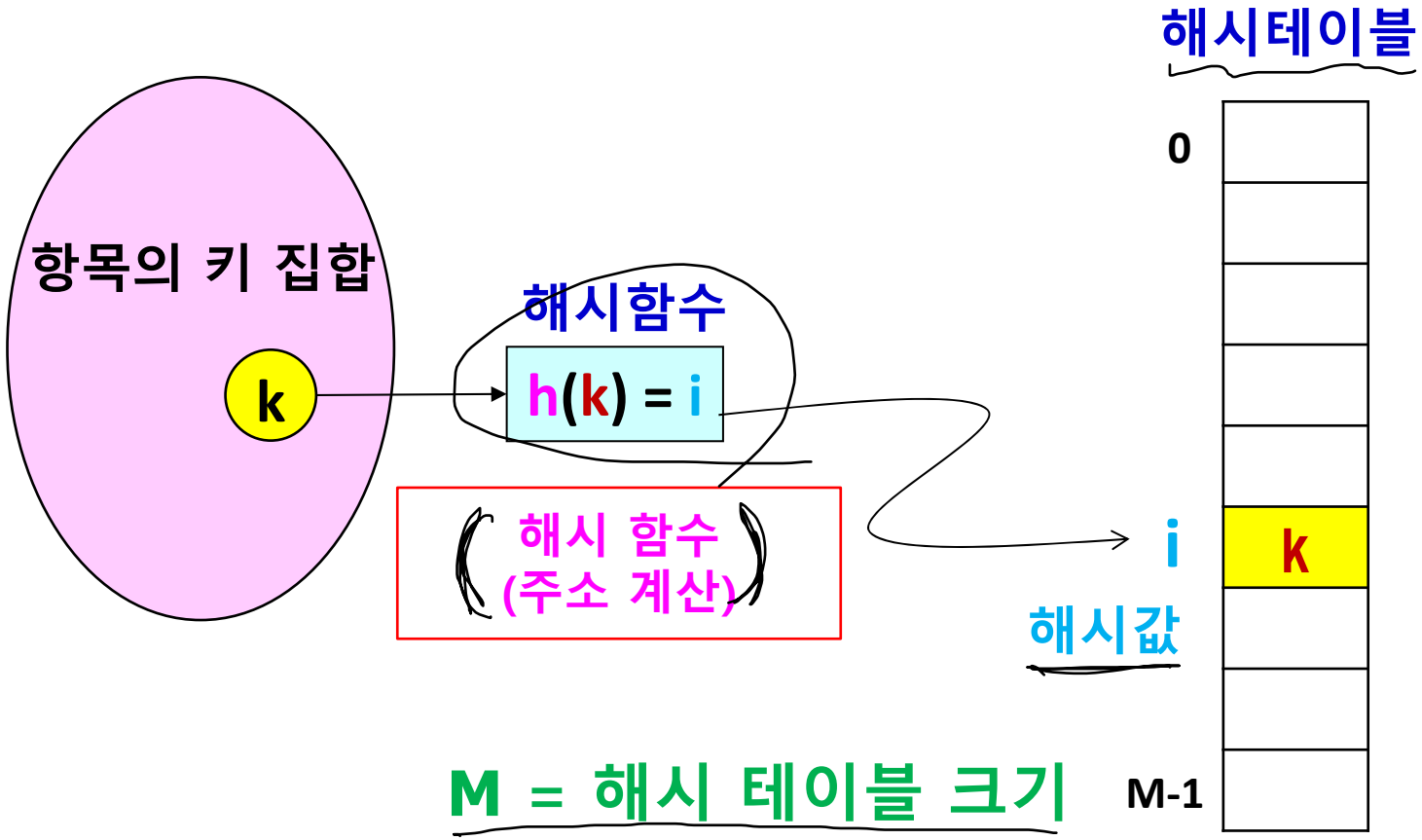
- **Θ(1) : 아주 빠른 검색, 삽입, 삭제 작업을 제공한다.**
  - 자신의 킷값에 의해 위치가 결정된다.
  - 다른 킷값과의 상대적인 크기에 의해 위치가 결정되지 않는다.
- **키를 배열의 인덱스로 그대로 사용하면 메모리 낭비가 심해질 수 있다.**

❍ 해싱(Hashing)

- 키를 간단한 함수를 사용해 변환한 값을 배열의 인덱스로 이용하여 항목을 저장하는 것
- **해시 함수**(Hash Function): 해싱에 사용되는 함수
- **해시 값**(Hash value) **또는 해시주소**: 해시함수가 계산한 값
- **해시 테이블**(Hash Table): 항목이 해시 값에 따라 저장되는 배열

해시 테이블 (3/3)

- 해시 테이블: 주소 계산
  - 주소 계산



CC BY NC ND

충돌 테이블

해시 함수

해시 함수 (1/2)

- **해시 함수**(Hash Functions)
  - **가장 이상적인 해시 함수**
    - 키들을 **균등하게(Uniformly)** 해시 테이블의 인덱스로 변환하는 함수
    - 균등하게 변환 한다는 것은 키들을 해시 테이블에 **랜덤하게 흩어지도록** 저장하는 것을 뜻한다.
    - 해시 함수는 키들을 균등하게 해시 테이블의 인덱스로 변환하기 위해 의미가 부여되어 있는 **키를 간단한 계산을 통해** '뒤죽박죽' 만든 후 해시 테이블의 크기에 맞도록 해시 값을 계산한다.
    - 아무리 균등한 결과를 보장하는 해시 함수라도 함수 계산 자체에 긴 시간이 소요된다면 **해싱의 장점인 연산의 신속성을 상실하므로** 그 가치를 잃어버린다.
  - **일반적으로 키들은 부여된 의미나 특성을 가지므로 키의 가장 앞 부분 또는 뒤의 몇 자리 등을 취하여 해시 값으로 사용하는 방식의 해시함수는 많은 충돌을 야기시킨다.**

해시 함수 (2/2)

해시 함수

장난감 수준의 함수

- 자릿수를 선택: h(001364825) = 35
- 접기Folding
  - h(001364825) = 1190
  - 해시테이블의 크기가 3이라면 1190에서 3자리 수만 을 해시 값으로 이용한다.

나누기 방법에 의한 함수Division Method

- h(key) = key % m ← m: 해시 테이블 사이즈, % : 나머지 연산
- m 은 소수를 권장

곱하기 방법에 의한 함수Multiplication Method

- h(key) = (keyA mod 1) * m
- A : (0, 1) 범위의 상수
- m 이 굳이 소수일 필요 없음. 보통 2^p (p는 양의 정수).

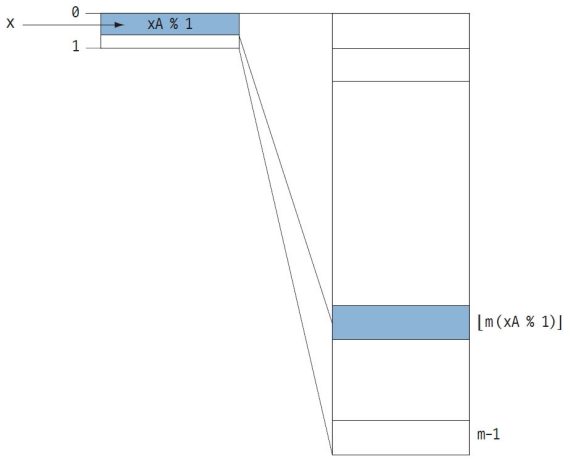


그림 12-3 곱하기 방법의 작동 원리

Clickseo.com

충돌 해결

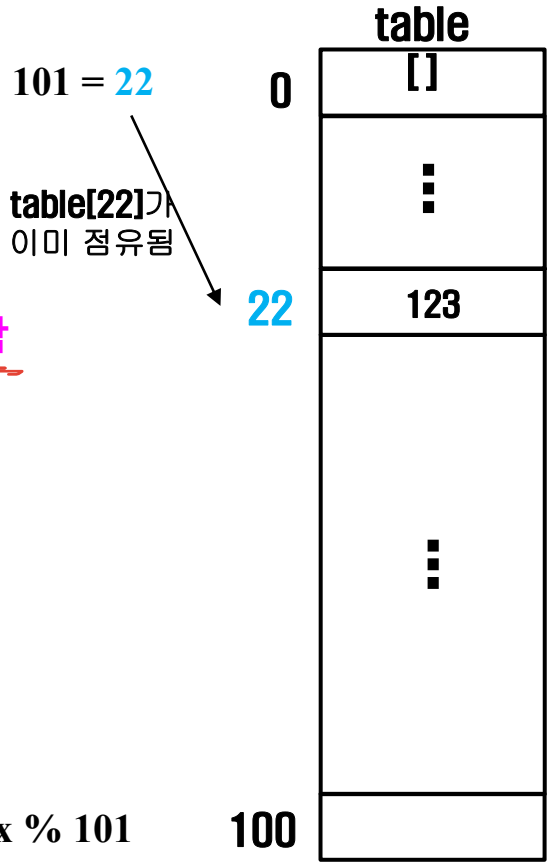
cc BY NC ND

- 해시 테이블
- **충돌 해결**
  - 개방 주소 방식
  - 폐쇄 주소 방식

충돌 해결

충돌 해결(Collision Resolution)

- 충돌 해결



충돌Collision :
어떤 킷값으로 도출된 주소에 이미 다른 키가 자리함

충돌 해결

- 일련의 해시 함수를 생성한다.
- $h_0(x)(=h(x)), h_1(x), h_2(x), h_3(x), \ldots$
- 해시 테이블에서 핵심적인 부분

해시 함수: h(x) = x % 101

CC BY NC ND

충돌 해결

개방형 주소 방식

개방형 주소 방식

- **개방형 주소 방식**(Open Addressing Methods)
  - 해시테이블 전체를 열린 공간으로 가정하고 충돌된 키를 일정한 방식에 따라서 찾아낸 empty 원소에 저장한다.
    - 선형 조사(Linear Probing)
    - 이차 조사(Quadratic Probing)
    - 랜덤 조사(Random Probing)
    - 이중 해싱(Double Hashing)

개방 주소 방법 Open Addressing

- 주어진 배열 안에서 해결

선형 탐색 Linear Probing

$$h_i(x) = (h_0(x)+ai+b) \% m$$

이차원 탐색 Quadratic Probing

$$h_i(x) = (h_0(x)+ai^2+bi+c) \% m$$

더블 해싱 Double Hashing

$$h_i(x) = (h_0(x) + i \cdot f(x)) \% m$$

$f(x)$: 보조 해시 함수

m: 해시 테이블 사이즈, %: 나머지 연산

개방형 주소 방식: 선형 조사 (1/10)

- **선형 조사(Linear probling)**
  - **선형조사는 충돌이 일어난 원소에서부터 순차적으로 검색하여, 처음 발견한 empty 원소에 충돌이 일어난 키를 저장한다.**
    - **h(key) = i 라면,** 해시테이블 **a[i], a[i+1], a[i+2], …, a[i+j]** 를 차례로 검색하여 처음으로 찾아낸 empty 원소에 key를 저장한다.
    - 해시테이블은 1차원 리스트이므로, **i + j** 가 **m** 이 되면 **a[0]** 을 검색한다.
    - **(h(key) + j) % m, j = 0, 1, 2, 3, …**

선형 탐색Linear Probing

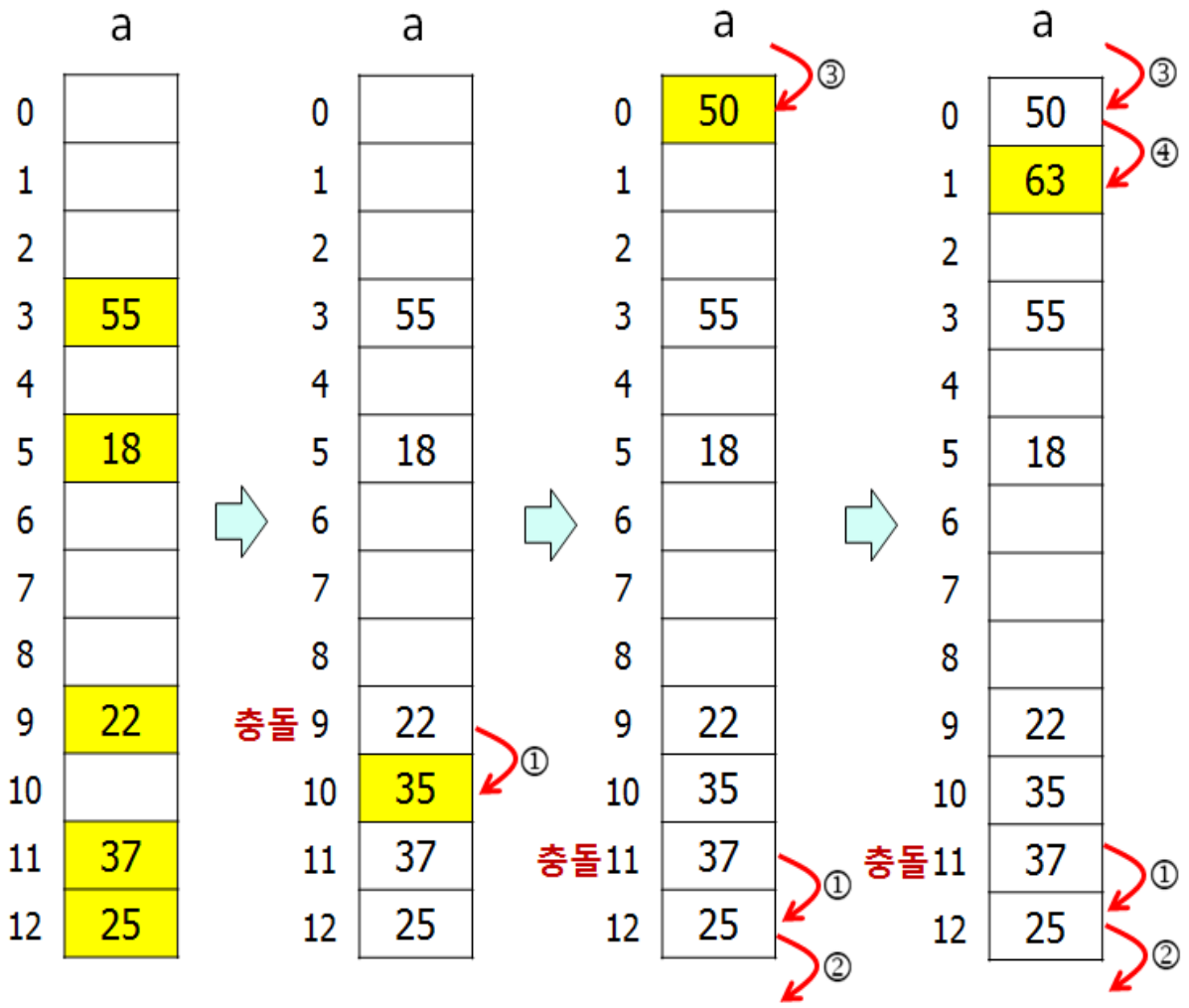
$$h_i(x) = (h_0(x) + i) \% m$$

1차 군집Primary Clustering에 취약

개방형 주소 방식: 선형 조사 (2/10)

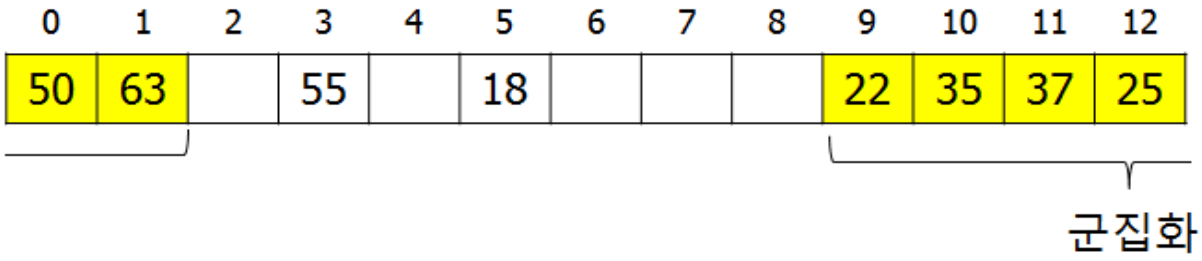
- **선형 조사:** 선형 조사 방식의 키 저장 과정

key	h(key) = key % 13
25	12
37	11
18	5
55	3
22	9
35	9
50	11
63	11



개방형 주소 방식: 선형 조사 (3/10)

- **선형 조사:** 1차 군집화
  - 선형 조사는 순차 탐색으로 empty 원소를 찾아 충돌된 키를 저장하므로 해시 테이블의 키들이 빈틈없이 뭉쳐지는 현상이 발생한다[1차 군집화(Primary Clustering)].
    - 이러한 군집화는 탐색, 삽입, 삭제 연산 시 군집된 키들을 순차적으로 방문해야 하는 문제점을 야기한다.



  - 군집화는 해시 테이블에 empty 원소 수가 적을수록 더 심화되며 해시 성능을 극단적으로 저하시킨다.

개방형 주소 방식: 선형 조사 (4/10)

- **선형 조사**
  - 선형 탐색

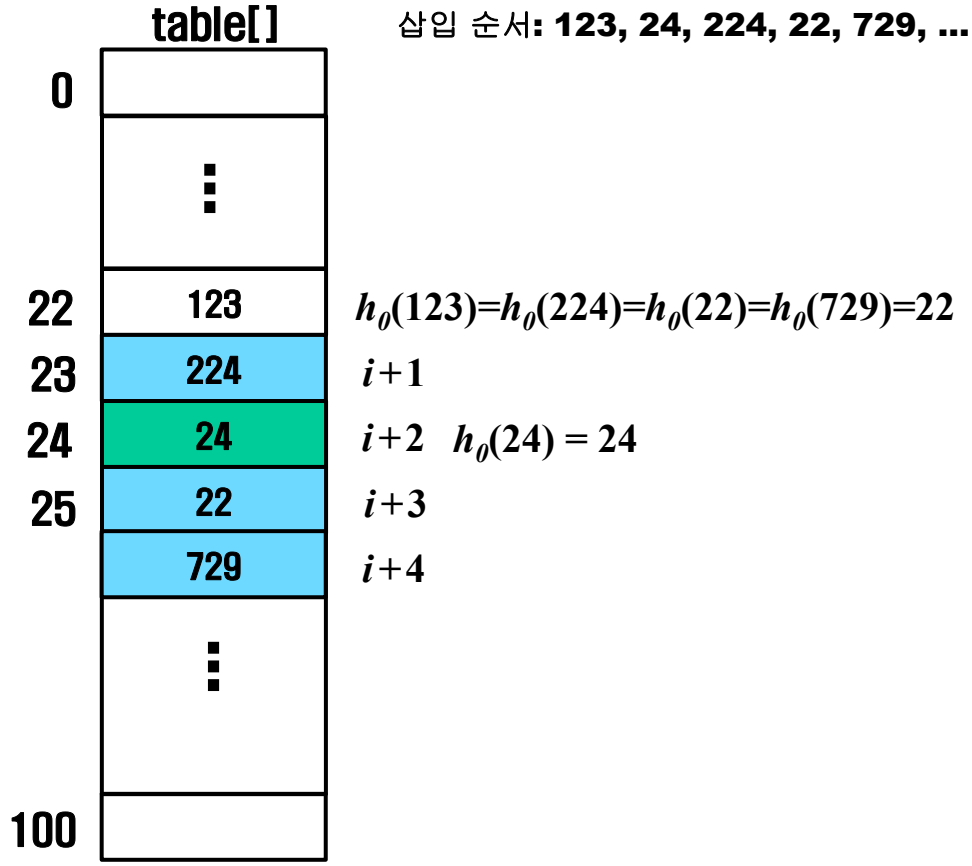
선형 탐색Linear Probing

$h_i(x) = (h_0(x) + i) \% m$

1차 군집Primary Clustering에 취약

i 번째 해시 함수의 예

$h_i(x) = (h_0(x) + i) \% 101$



알고리즘 search / insert / delete

알고리즘 12-2 개방 주소 방법

```
search(x):
    i ← 0
    repeat
        j ← h_i(x)
        if (table[j] = x) return j
        else i++
    until (table[j] = null or i = m)
    return NOT_FOUND ◀ search failed!
```

알고리즘 12-2 개방 주소 방법

```
◀ table[]: 해시 테이블
insert(x):
    i ← 0
    repeat
        j ← h_i(x)
        if (table[j] = null or table[slot] = DELETE)
            table[j] ← x
            return j
        else i++
    until (i = m)
    error "테이블 오버플로우"
```

알고리즘 12-2 개방 주소 방법

```
delete(x):
    i ← 0
    repeat
        j ← h_i(x)
        if (table[j] = x)
            ❶ table[j] ← DELETED;
              break
        else i++
    until (table[j] = null or i = m)
```

삭제시 주의할 점

삭제된 자리를 null 값으로 채우면
검색 시 존재하는 원소를 없다고 대답할 수 있다.

지운 자리에 표식을 해둔다
예: 상수 DELETED 할당

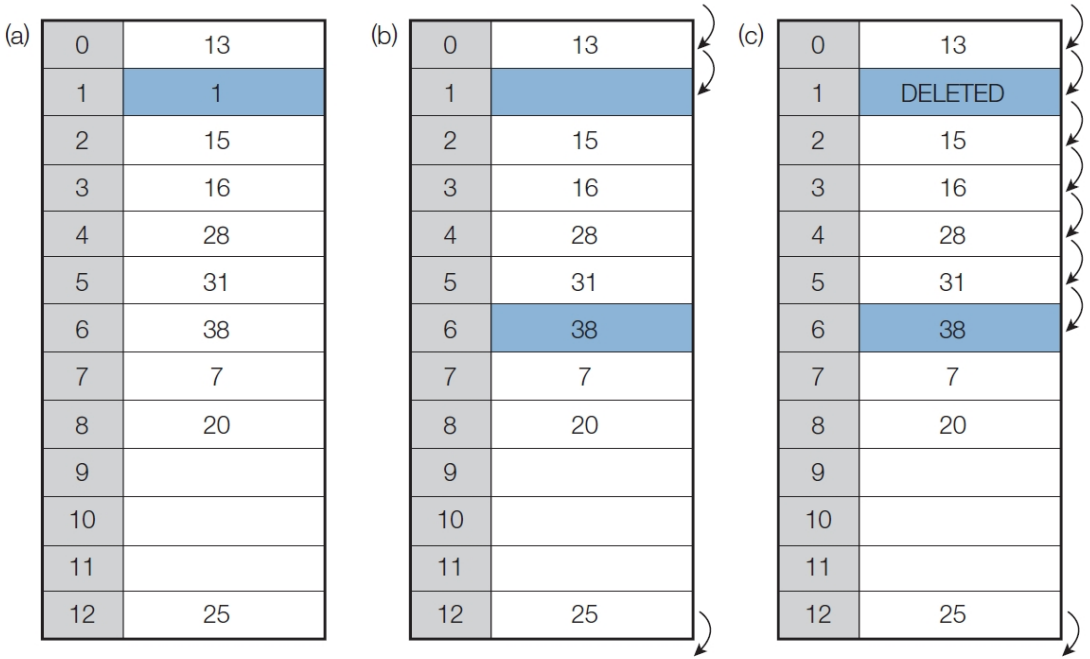
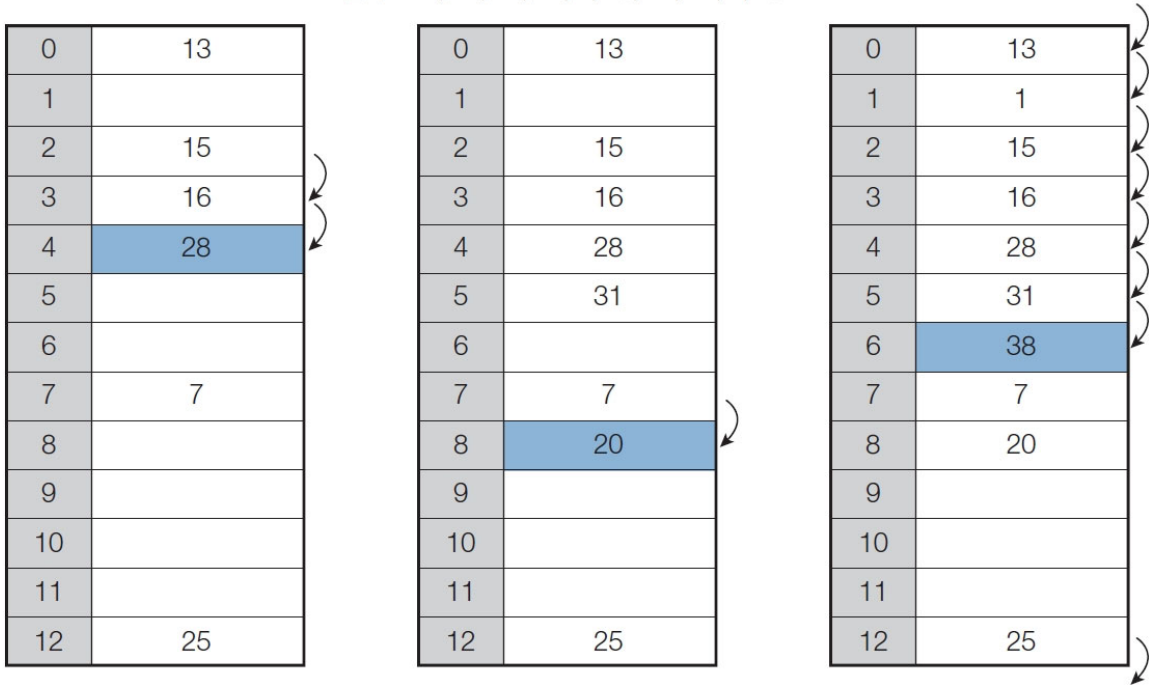


그림 12-5 해시 테이블에서 자료가 삭제될 경우의 처리법

입력 : 25, 13, 16, 15, 7, 28, 31, 20, 1, 38



(a) 키 25, 13, 16, 15, 7, 28 삽입 (b) 키 31, 20 삽입 (c) 키 1, 38 삽입

그림 12-6 선형 탐색의 예

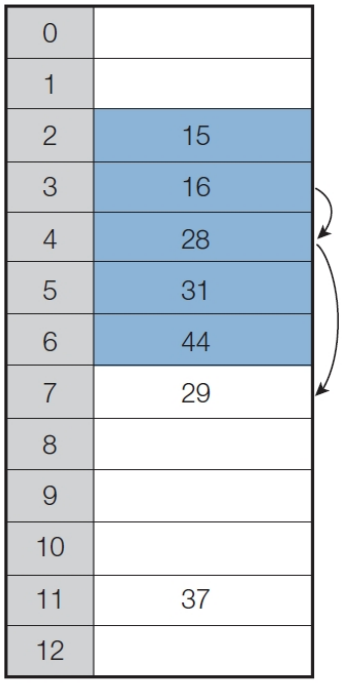
0	
1	
2	15
3	16
4	28
5	31
6	44
7	
8	
9	
10	
11	37
12	

(a) a = 1, b = 0인 경우

0	
1	
2	15
3	
4	17
5	
6	28
7	33
8	43
9	
10	23
11	37
12	

(b) a = 2, b = 0인 경우

그림 12-7 1차 군집의 예



0	
1	
2	15
3	16
4	28
5	31
6	44
7	29
8	
9	
10	
11	37
12	

그림 12-8 1차 군집을 빨리 벗어나는 예

0	
1	
2	15
3	28
4	
5	54
6	41
7	
8	21
9	
10	
11	67
12	

그림 12-9 2차 군집의 예

$h(x) = x \% 13$

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

$h_i(x) = (h(x) + i \cdot f(x)) \% 13$

0		
1		
2	15	$h(15) = h(28) = h(41) = h(67) = 2$
3		
4	67	$h_1(67) = 4$
5		
6	19	
7		
8		
9	28	$h_1(28) = 9$
10		
11	41	$h_1(41) = 11$
12		

그림 12-10 2차 군집에서 해방된 예

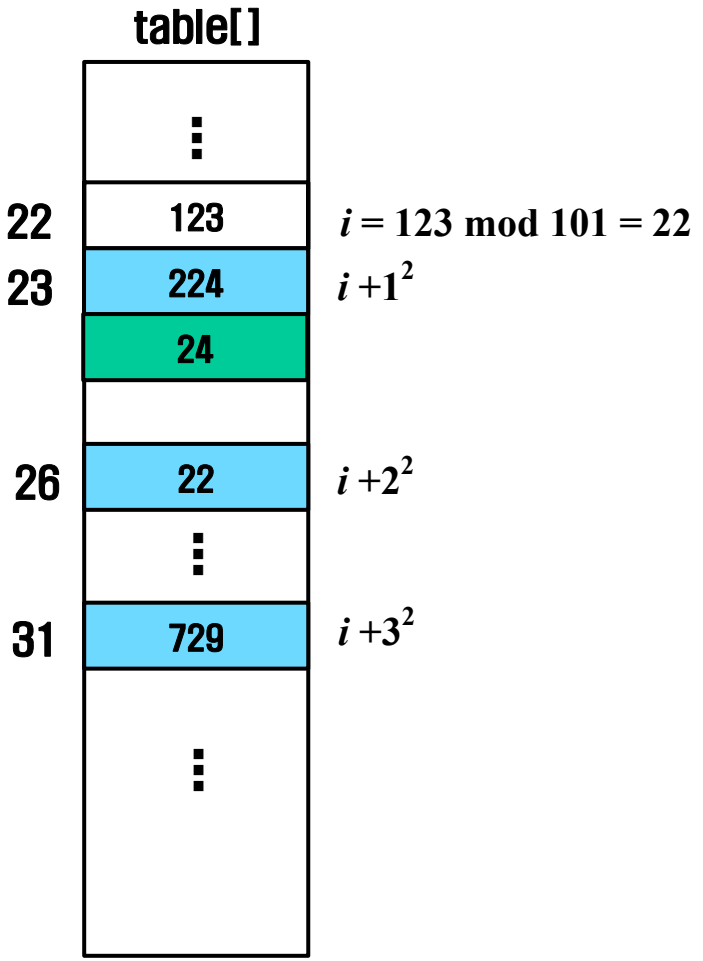
이차원 탐색Quadratic Probing

$h_i(x) = (h_0(x) + i^2) \% m$

2차 군집Secondary Clustering에 취약

i번째 해시 함수의 예

$h_i(x) = (h_0(x) + i^2) \% 101$



개방형 주소 방식: 이차 조사 (1/2)

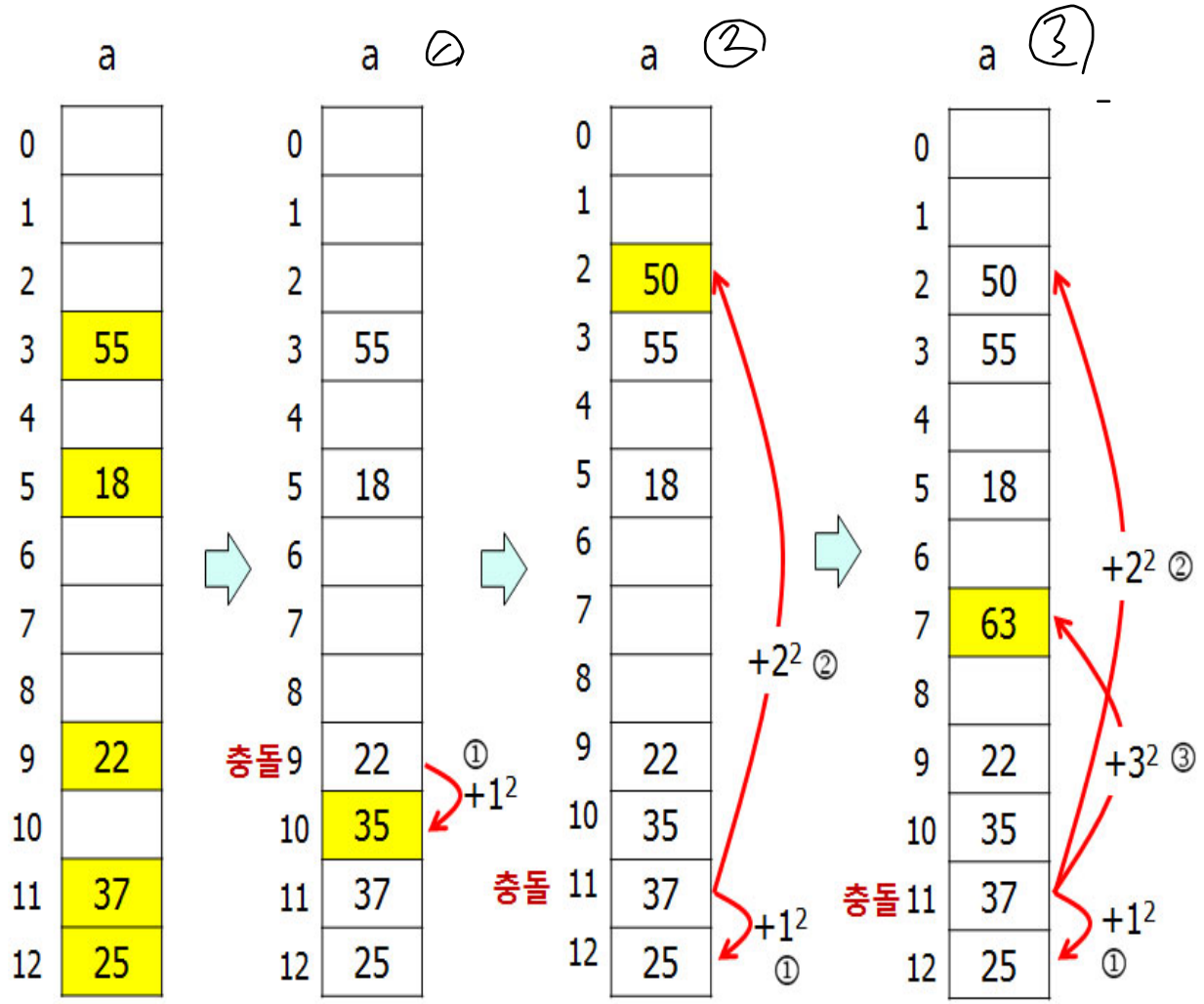
- **이차 조사(Quadratic probling)**
  - 선형 조사와 근본적으로 동일한 충돌해결 방법
    - 충돌 후 배열 a 에서 ...
    - (h(key) + j^2) % M, j = 0, 1, 2, 3, ...
    - 으로 선형 조사보다 더 멀리 떨어진 곳에서 empty 원소를 찾는다.

    - 이차 조사는 이웃하는 빈 곳이 채워져 만들어지는 1차 군집화 문제를 해결하지만,
    - 같은 해시 값을 갖는 서로 다른 키들인 동의어(Synonym)들이 똑같은 점프 시퀀스(Jump Sequence)를 따라 empty 원소를 찾아 저장하므로 결국 또 다른 형태의 군집화인 2차 군집화(Secondary Clustering)를 야기한다.
    - 점프 크기가 제곱 만큼씩 커지므로 배열에 empty 원소가 있는데도 empty 원소를 건너뛰어 탐색에 실패하는 경우도 피할 수 없다.

개방형 주소 방식: 이차 조사 (2/2)

- **이차 조사:** 이차 조사 방식의 키 저장 과정

key	h(key) = key % 13
25	12
37	11
18	5
55	3
22	9
35	9
50	11
63	11



개방형 주소 방식: 랜덤 조사

- **랜덤 조사(Random probling)**
  - **선형 조사와 이차 조사의 규칙적인 점프 시퀀스와는 달리 점프 시퀀스를 무작위화 하여 empty 원소를 찾는다.**
    - 랜덤 조사는 **의사 난수 생성기를 사용하여** 다음 위치를 찾는다.
    - 랜덤 조사 방식도 동의어들이 똑같은 점프 시퀀스에 따라 empty 원소를 찾아 키를 저장하게 되고, 이 때문에 **3차 군집화(Tertiary Clustering)**가 발생한다.

개방형 주소 방식: 이중 해싱 (1/4)

- **이중 헤싱(Double Hashing)**
  - 이중 해싱은 2개의 해시 함수를 사용한다.
    - 하나는 기본적인 해시 함수 h(key)로 키를 해시 테이블의 인덱스로 변환하고, 제2의 함수 d(key)는 충돌 발생 시 다음 위치를 위한 점프 크기를 다음의 규칙에 따라 정한다.
    - (h(key) + j ·d (key)) mod M, j = 0, 1, 2, ...
    - 제 2의 함수 d(key)는 점프 크기를 정하는 함수이므로 0을 리턴 해선 안다.
    - 그 외의 조건으로 **d(key)**의 값과 해시 테이블의 크기 **M** 과 서로소(Relatively Prime) 관계일 때 좋은 성능을 보인다.
    - 하지만 해시 테이블 크기 **M**을 소수로 선택하면, 이 제약 조건을 만족한다.
  - 이중 해싱은 동의어들이 저마다 제2 해시 함수를 갖기 때문에 점프 시퀀스가 일정하지 않다.
  - 따라서 이중 해싱은 모든 군집화 문제를 해결한다.

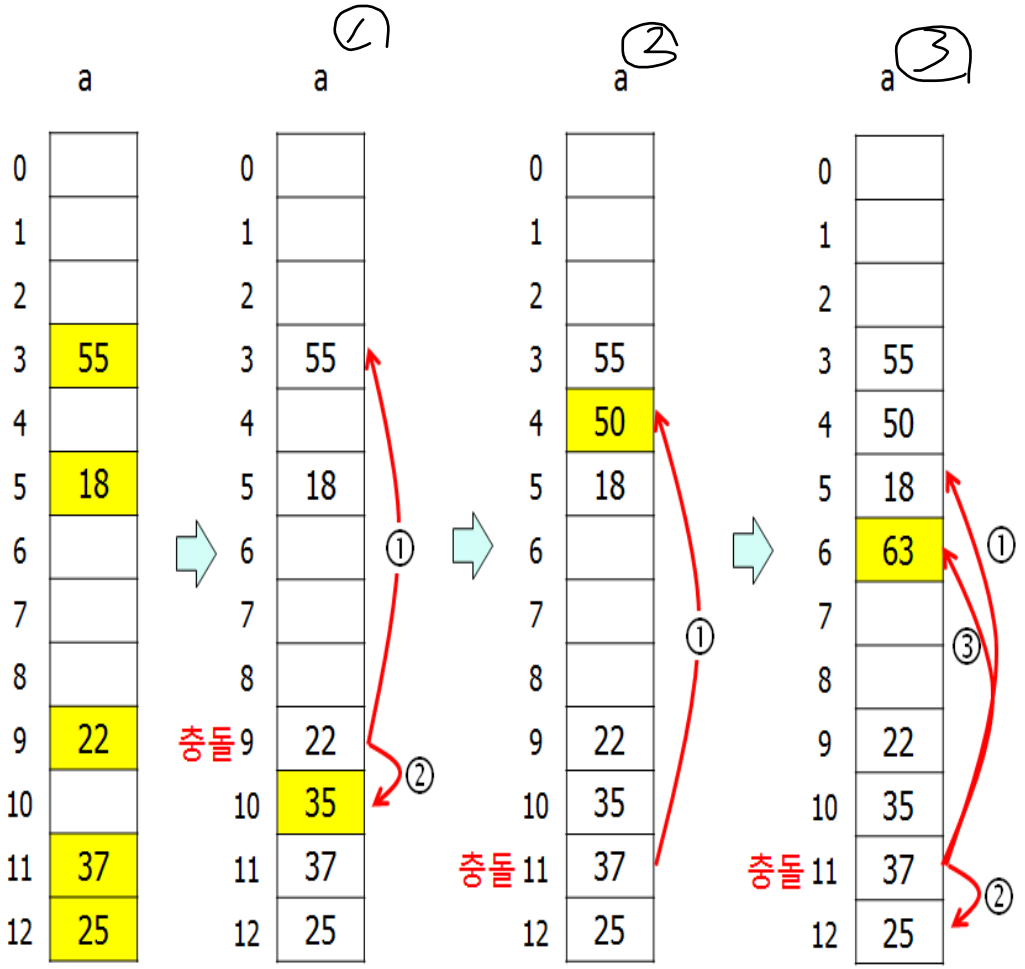
- **h(key) = key % 13**과 **d(key) = 7-(key % 7)** 에 따라, **25, 37, 18, 55, 22, 35, 50, 63**을 해시테이블에 차례로 저장하는 과정

key	h(key)	d(key)	(h(key) + j*d(key)) % 13		
			j=1	j=2	j=3
25	12				
37	11				
18	5				
55	3				
22	9		①	②	
35	9	7	3	10	
50	11	6	4		③
63	11	7	5	12	6

h(key) = key % 13

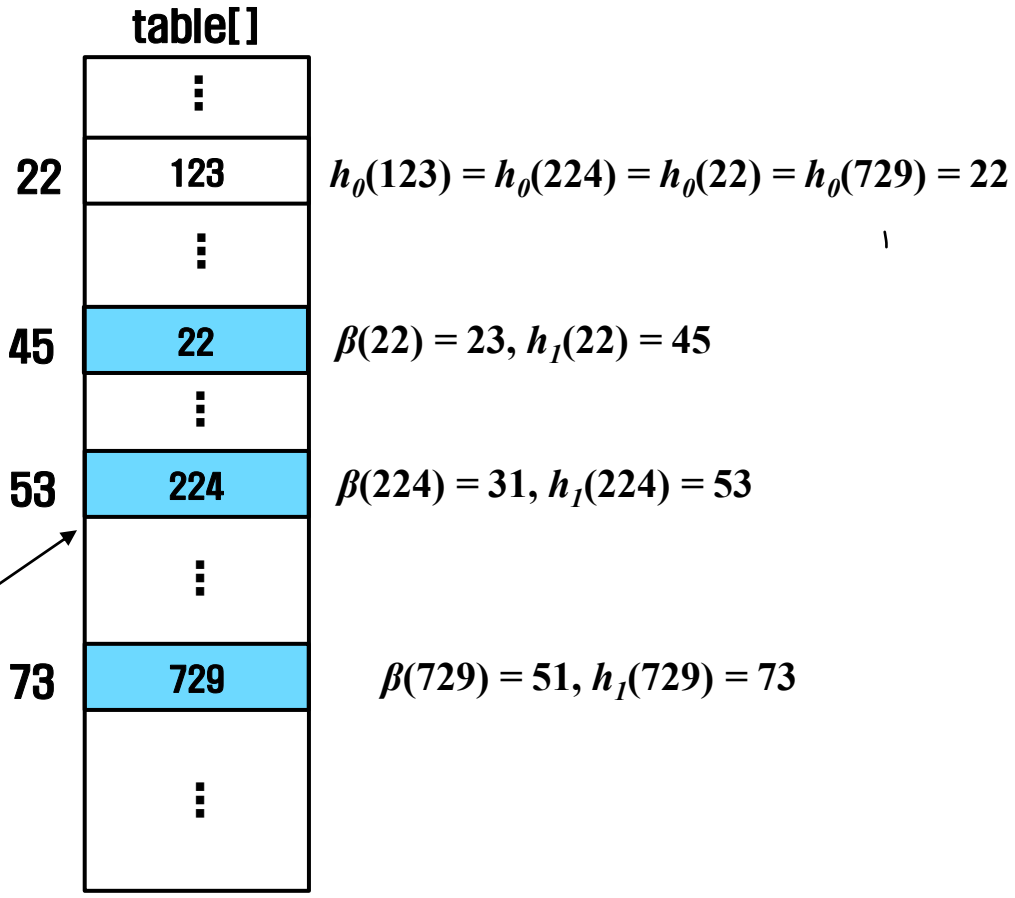
d(key) = 7 – (key % 7)

(h(key) + j*d(key)) % 13, j = 0, 1, ...



더블 해싱 Double hashing

$h_i(x) = (h_0(x) + i \cdot f(x)) \ \%\ 101$



*i*번째 해시 함수의 예

$h_0(x) = x\ \%\ 101$

$f(x) = 1 + (x\ \%\ 97)$

이중해싱의 장점

- 이중해싱은 빈 곳을 찾기 위한 점프 시퀀스가 일정하지 않으며, 모든 군집화 현상을 발생시키지 않는다.
- 또한 해시 성능을 저하시키지 않는 동시에 해시테이블에 많은 키들을 저장할 수 있다는 장점을 가지고 있다.

CC BY NC ND

충돌 해결

폐쇄 주소 방식

폐쇄 주소 방식 (1/2)

- **폐쇄 주소 방식**(Closed Addressing Methods)
  - 키에 대한 해시 값에 대응되는 곳에만 키를 저장한다.
    - 충돌이 발생한 키들은 한 위치에 모여 저장한다.
  - 가장 대표적인 방법: 체이닝(Chaining)

해시 테이블은 연결 리스트의 헤더 노드를 레퍼런스한다

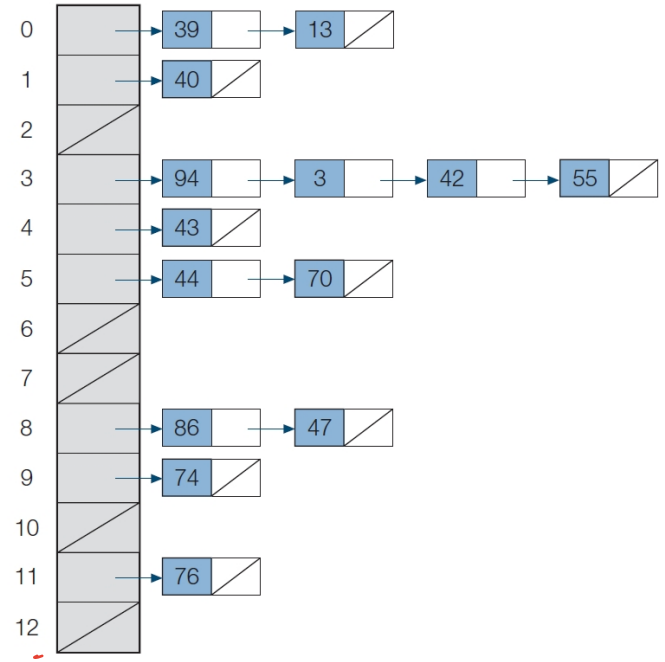


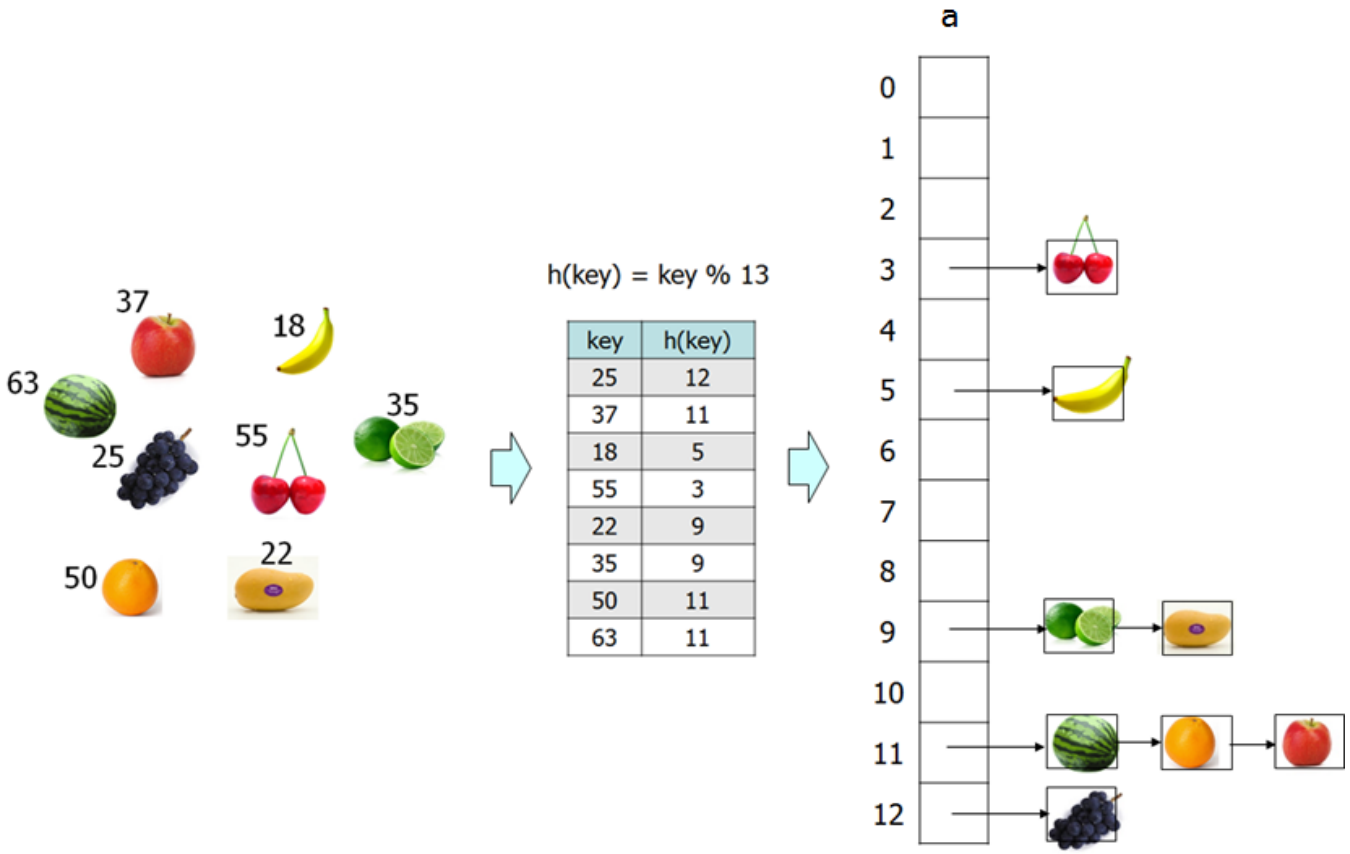
그림 12-4 체이닝을 이용한 충돌 해결의 예

직접 충돌이 일어나지 않은 키들끼리는 간섭하지 않는다.

Clickseo.com

폐쇄 주소 방식 (2/2)

- **폐쇄 주소 방식**(Closed Addressing Methods)



참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

[3] “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 문병로, 한빛아카데미, 2022.

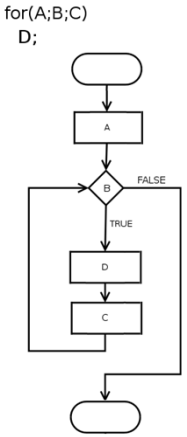
[4] “이것이 취업을 위한 코딩 테스트다 with 파이썬”, 나동빈, 한빛미디어, 2020.

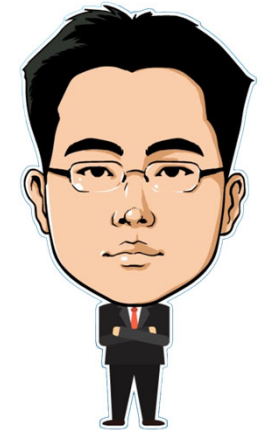
[5] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[6] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.

[7] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[8] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.





이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.