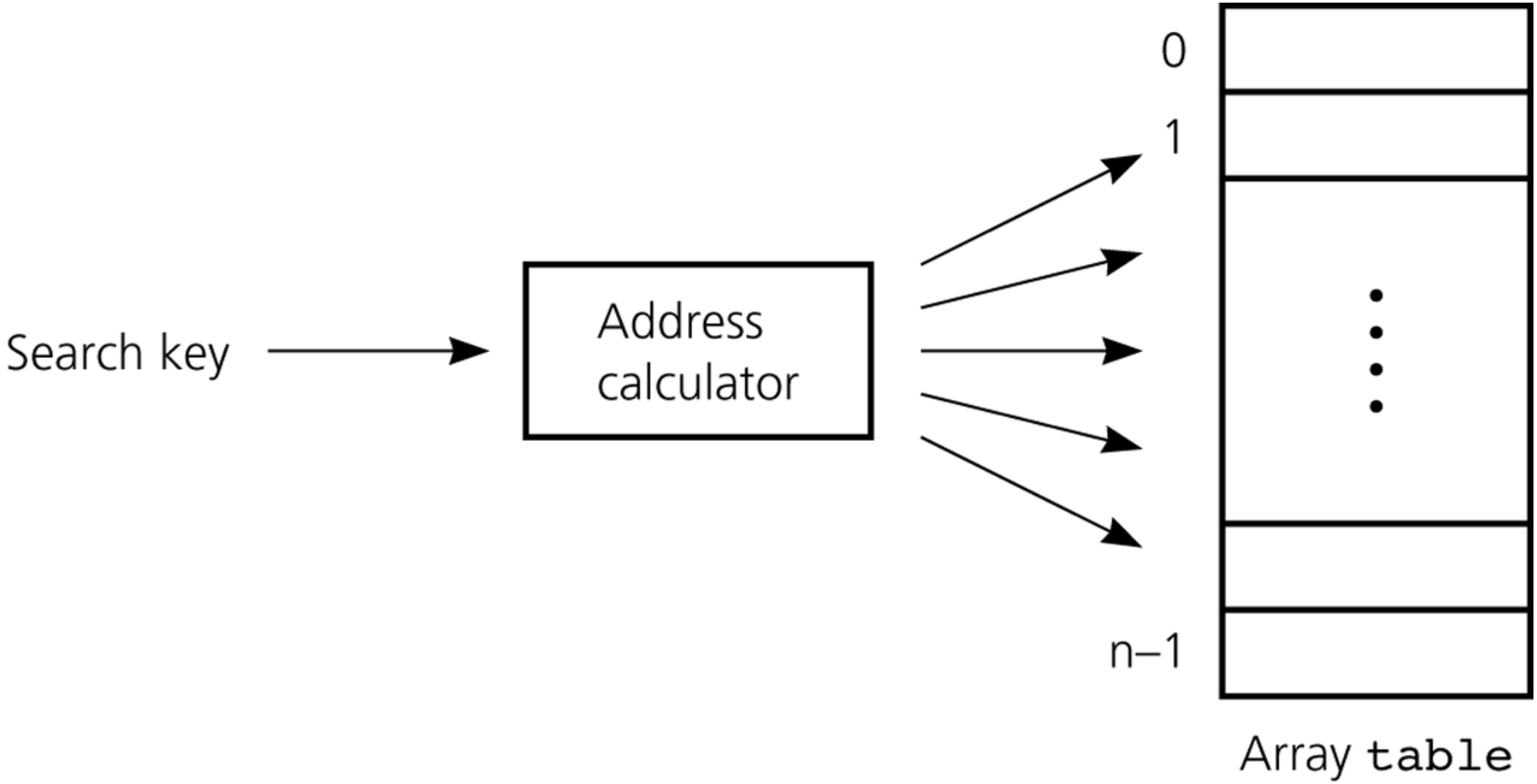
Ch. 12 Hash Tables

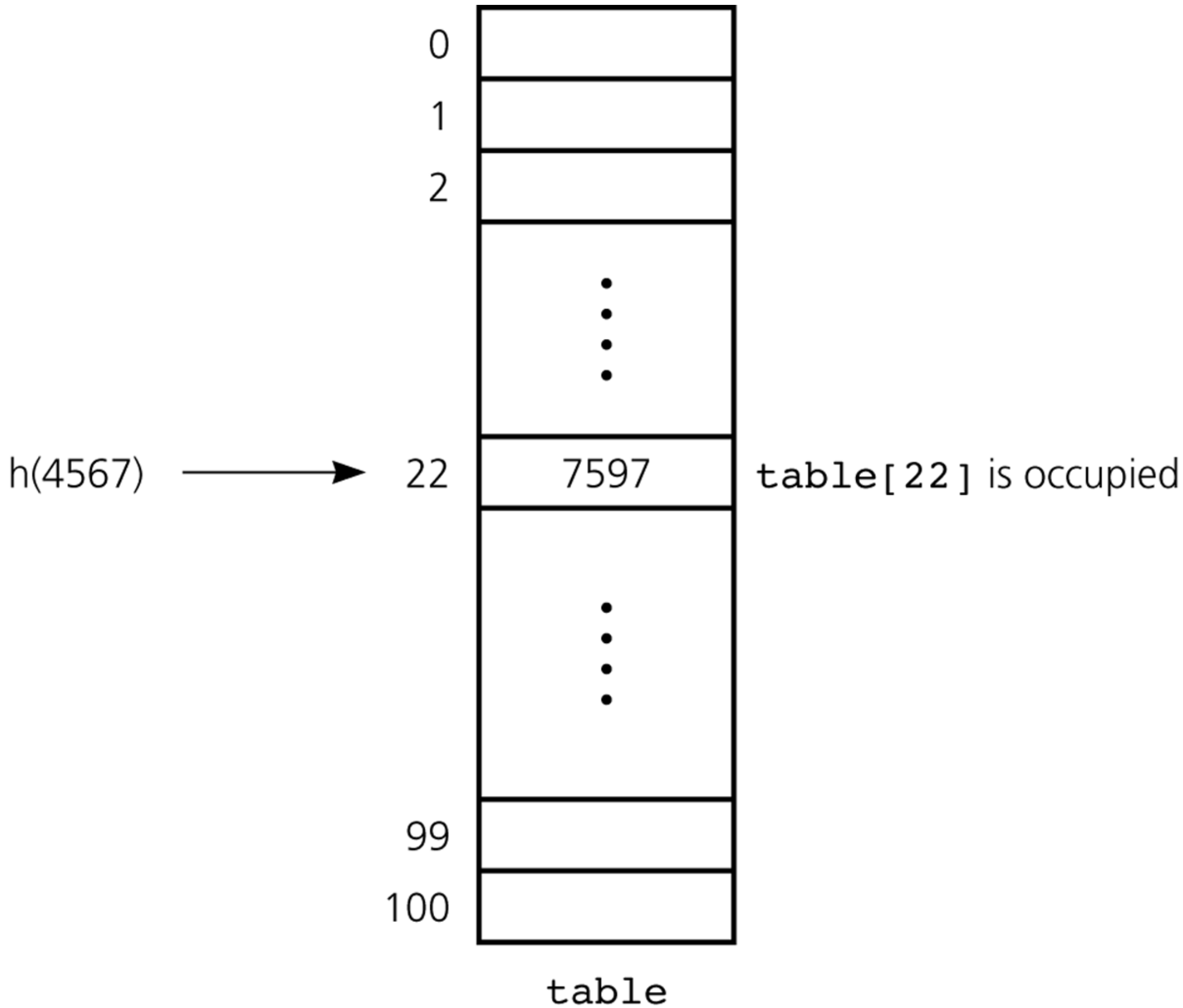
- Array or linked list
  - Overall $O(n)$ time
- Binary search trees
  - Expected $\theta(\log n)$-time search, insertion, and deletion
  - But, $\theta(n)$ in the worst case
- Balanced binary search trees
  - Guarantees $O(\log n)$-time search, insertion, and deletion
  - Red-black tree, AVL tree
- Balanced k-ary trees
  - Guarantees $O(\log n)$-time search, insertion, and deletion w/ smaller constant factor
  - 2-3 tree, 2-3-4 tree, B-trees
- Hash table
  - Expected $\theta(1)$-time search, insertion, and deletion

- Stack, queue, priority queue
  - do not support *search* operation
  - i.e., do not support *dictionary*
- But, hash table does not support finding the minimum (or maximum) element
- Applications that need radically fast operations
  - 119 emergent calls and locating caller's address
  - Air flight information system
  - 주민등록 시스템

Address calculator



Collision



Insert

```
tableInsert(x)
{  // A[ ]: hash table, x: new key to insert
    if (A[h(x)] is not occupied) {
        A[h(x)] = x;
    } else {
        Find an appropriate index i by a collision-resolution method;
        A[i] =x ;
    }
}
```

Hash Functions

- Toy functions
  - Selection digits
    - $h(001364825) = 35$
  - Folding
    - $h(001364825) = 1190$
- Modulo arithmetic
  - $h(x) = x \bmod tableSize$
  - *tableSize* is recommended to be prime
- Multiplication method
  - $h(x) = (xA \bmod 1) * tableSize$
  - A: constant in $(0, 1)$
  - *tableSize* is not critical, usually 2^p for an integer p

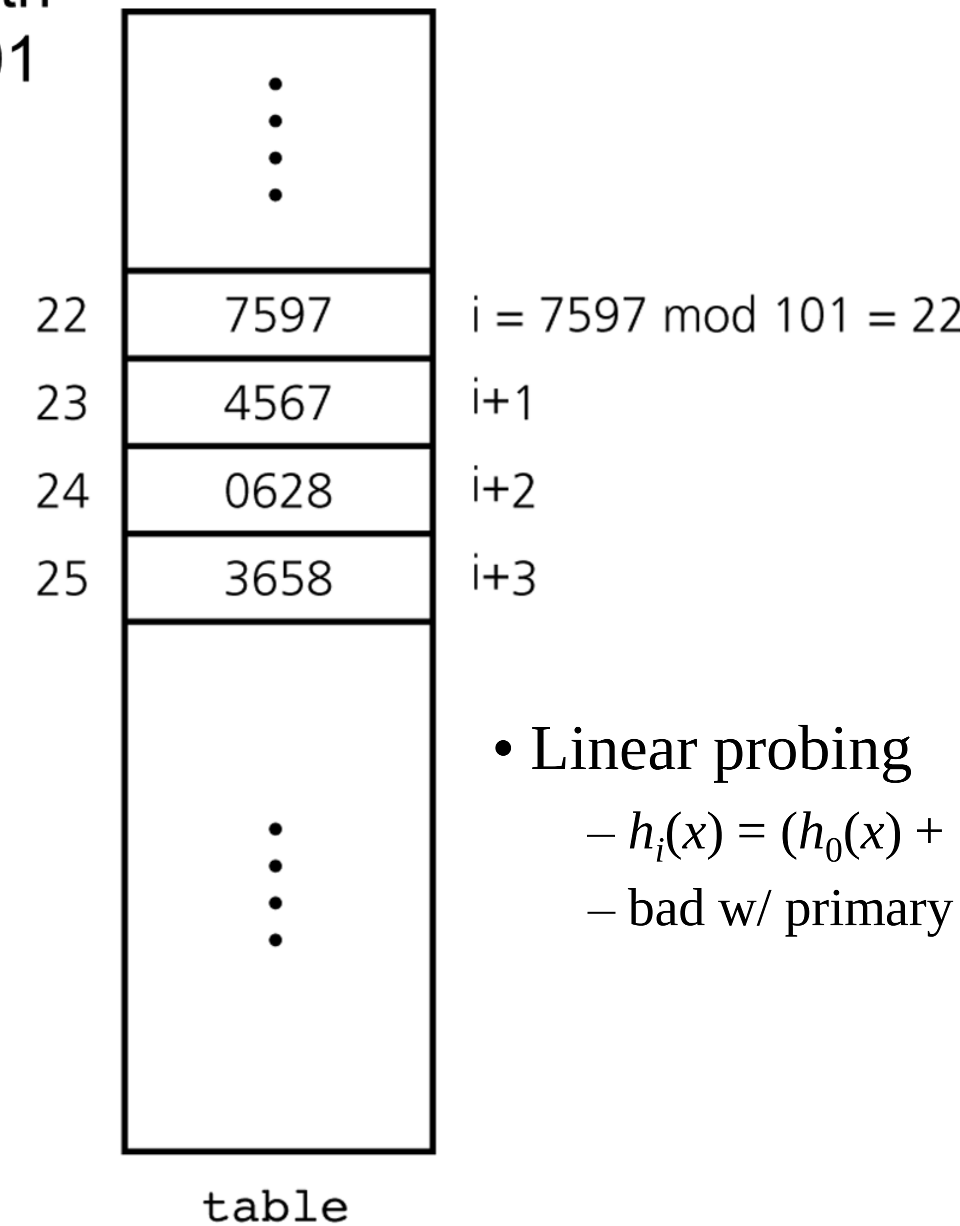
Collision Resolution

- Collision
  - The situation that two keys are mapped into the same location in the hash table
- Collision resolution
  - resolves collision by a seq. of hash values
  - $h_0(x)(=h(x)), h_1(x), h_2(x), h_3(x), \ldots$

Collision-Resolution Methods

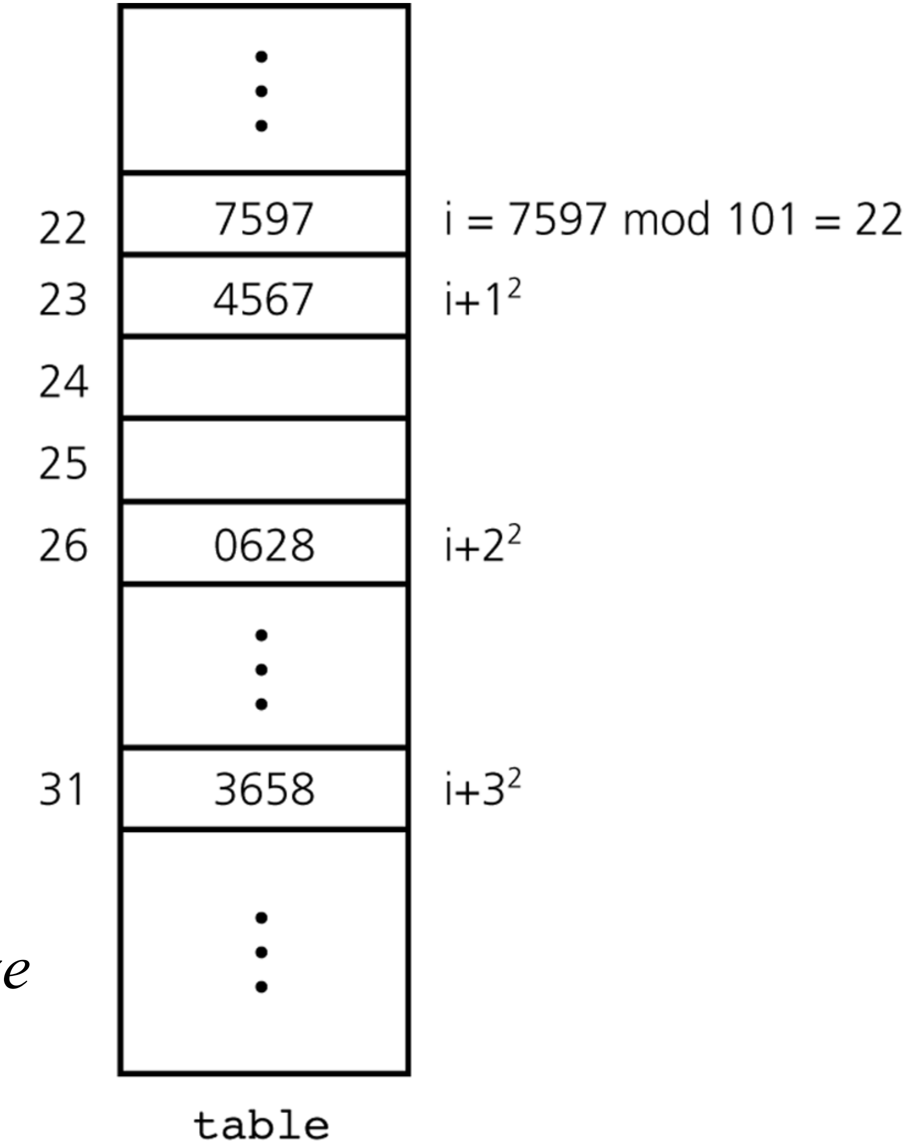
- Open addressing (resolves in the array)
  - Linear probing
    - $h_i(x) = (h_0(x) + i) \bmod tableSize$
  - Quadratic probing
    - $h_i(x) = (h_0(x) + i^2) \bmod tableSize$
  - Double hashing
    - $h_i(x) = (\alpha(x) + i \bullet \beta(x)) \bmod tableSize$
    - $\alpha(x)$, $\beta(x)$: hash functions
- Separate chaining
  - Each *table*[*i*] is maintained by a linked list

Linear probing with $h(x) = x \bmod 101$



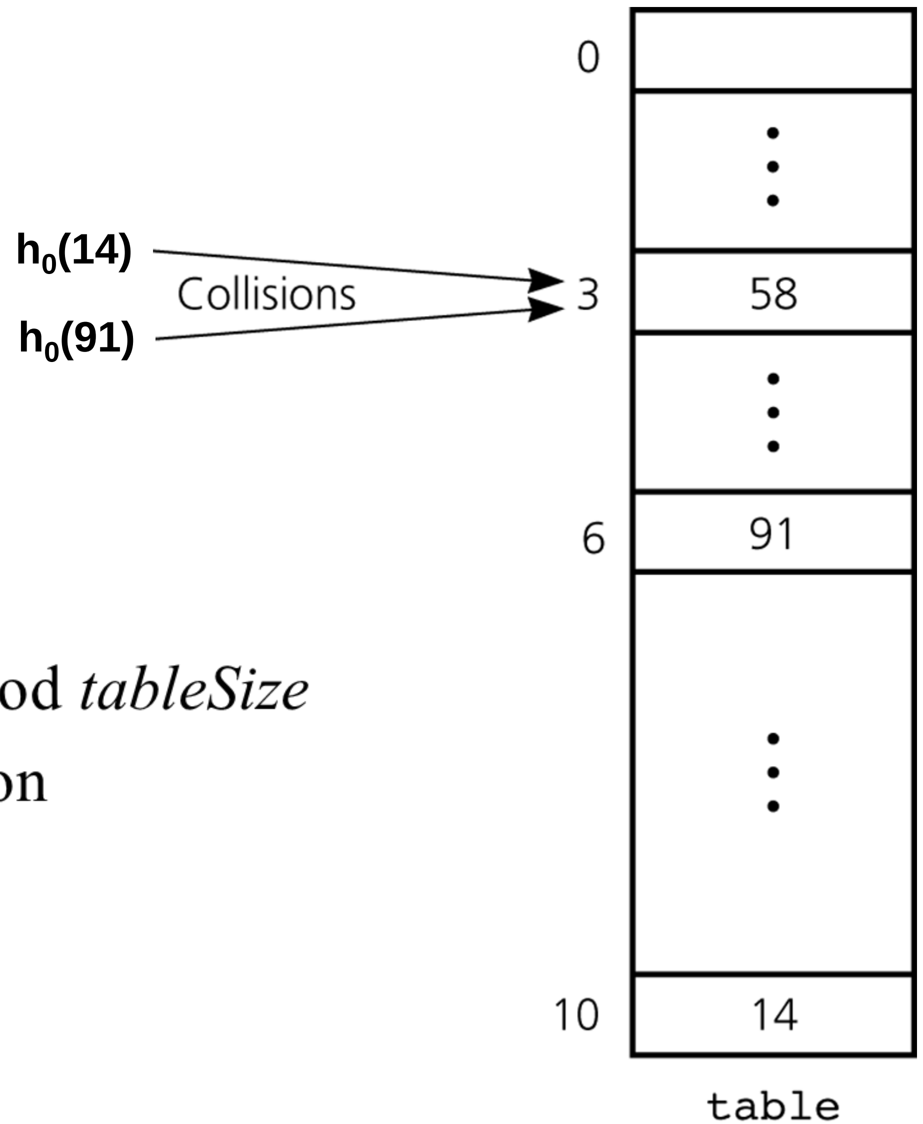
- Linear probing
  - $h_i(x) = (h_0(x) + i) \bmod tableSize$
  - bad w/ primary clustering

Quadratic probing with
$h(x) = x \bmod 101$



- Quadratic probing
  - $h_i(x) = (h_0(x) + i^2) \bmod tableSize$
  - bad w/ secondary clustering

Double hashing during the insertion of 58, 14, and 91



- Double hashing
  - $h_i(x) = (h_0(x) + i \bullet \beta(x)) \bmod \mathit{tableSize}$
  - $\beta(x)$: another hash function

Double Hashing의 예

$$h_i(x) = (h(x) + i\,f(x)) \bmod m$$

예: 입력 순서 15, 19, 28, 41, 67

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

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

$h_1(67) = 3$

$h_1(28) = 8$

$h_1(41) = 10$

$$h(x) = x \bmod 13$$
$$f(x) = (x \bmod 11) + 1$$
$$h_i(x) = (h(x)+i\,f(x)) \bmod 13$$

삭제시 조심할 것

0	13
1	1
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

(a) 원소 1 삭제

0	13
1	
2	15
3	16
4	28
5	31
6	38
7	7
8	20
9	
10	
11	
12	25

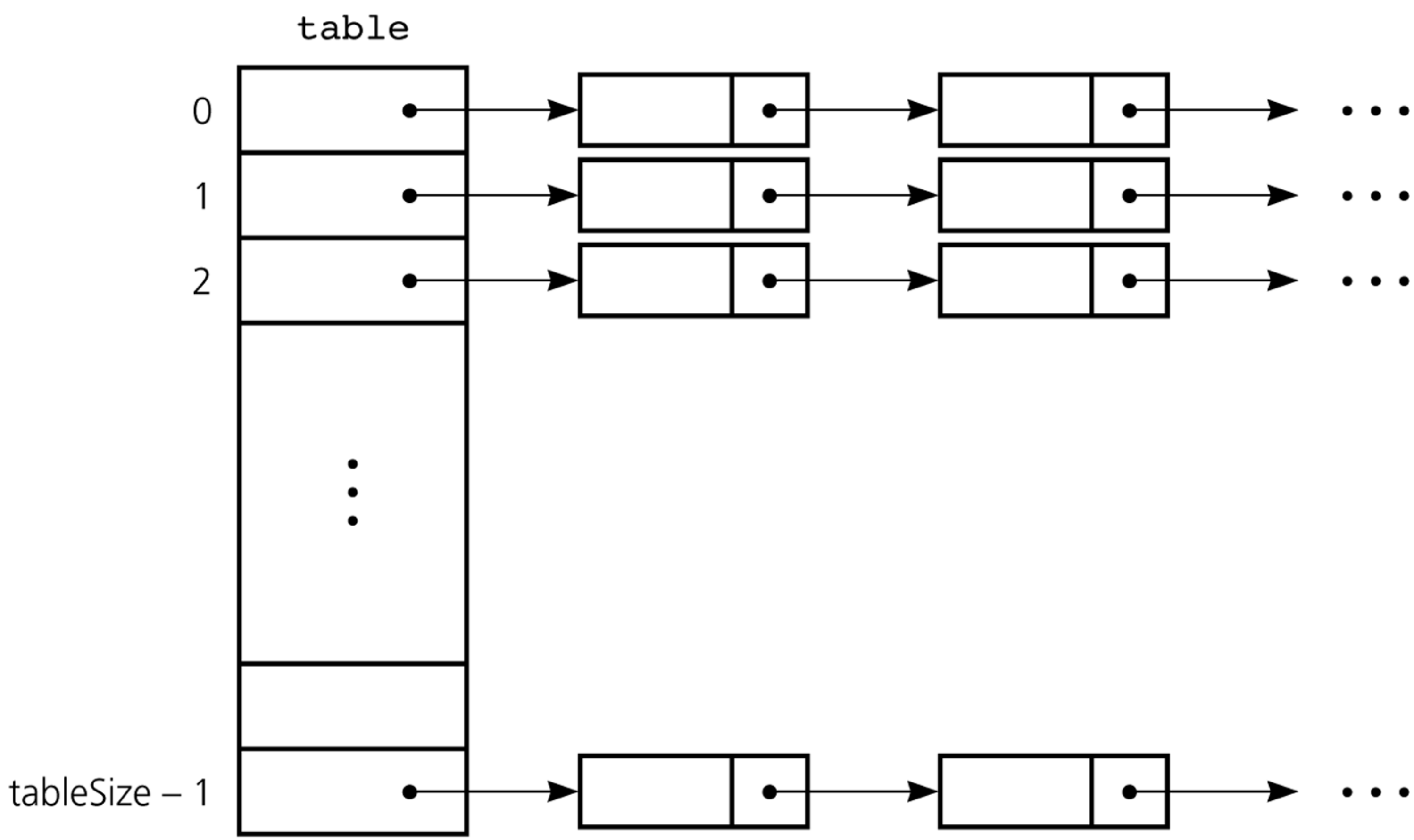
(b) 38 검색, 문제발생

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

(c) 표식을 해두면 문제없다

- Increasing the size of hash table
  - Load factor α
    - The rate of occupied slots in the table
    - A high load factor harms performance
      - We need to increase the size of hash table
  - Increasing the hash table
    - Roughly double the table size
    - Rehash all the items on the new table

Separate chaining

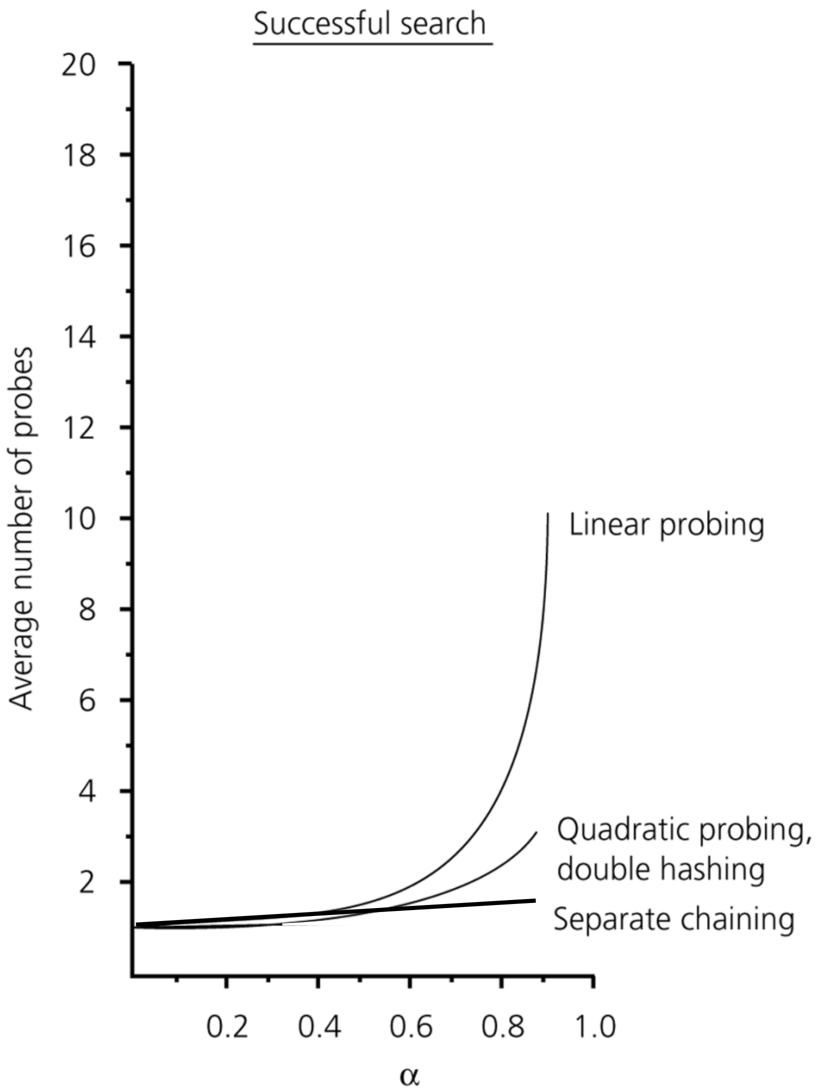


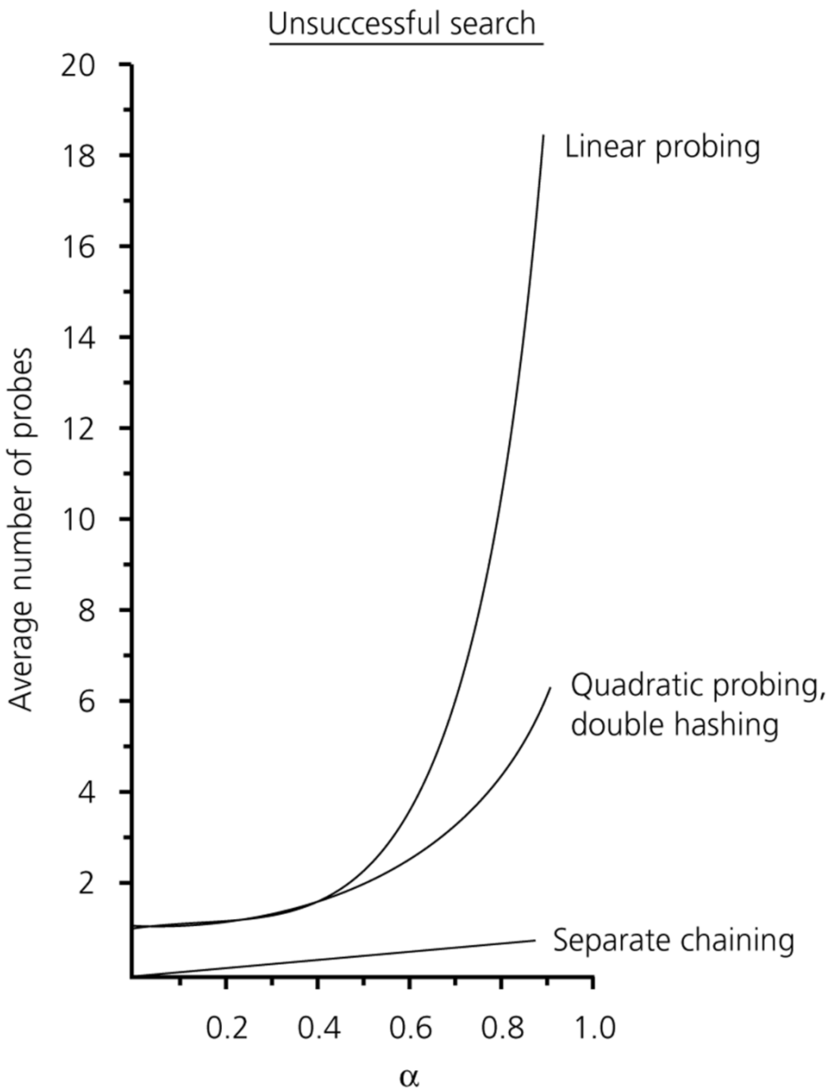
Each location of the hash table contains a reference to a linked list

The Efficiency of Hashing

- Approximate average # of comparisons for a search
  - Linear probing
    - $\frac{1}{2}(1 + {}^{1}/_{(1-\alpha)})$ for a successful search
    - $\frac{1}{2}(1 + {}^{1}/_{(1-\alpha)^2})$ for an unsuccessful search
  - Quadratic probing and double hashing
    - ${}^{-\ln(1-\alpha)}/_{\alpha}$ for a successful search
    - ${}^{1}/_{1-\alpha}$ for an unsuccessful search
  - Separate chaining (except the access for the indexing array)
    - $1 + \alpha/2$ for a successful search
    - α for an unsuccessful search

The Relative Efficiency of Collision-Resolution Methods





Good Hash Functions

- should be easy and fast to compute
- should scatter the data evenly on the hash table

Observation

- Load factor가 낮을 때는 probing 방법들은 대체로 큰 차이가 없다.
- Successful search는 insertion할 당시의 궤적을 그대로 밟는다.