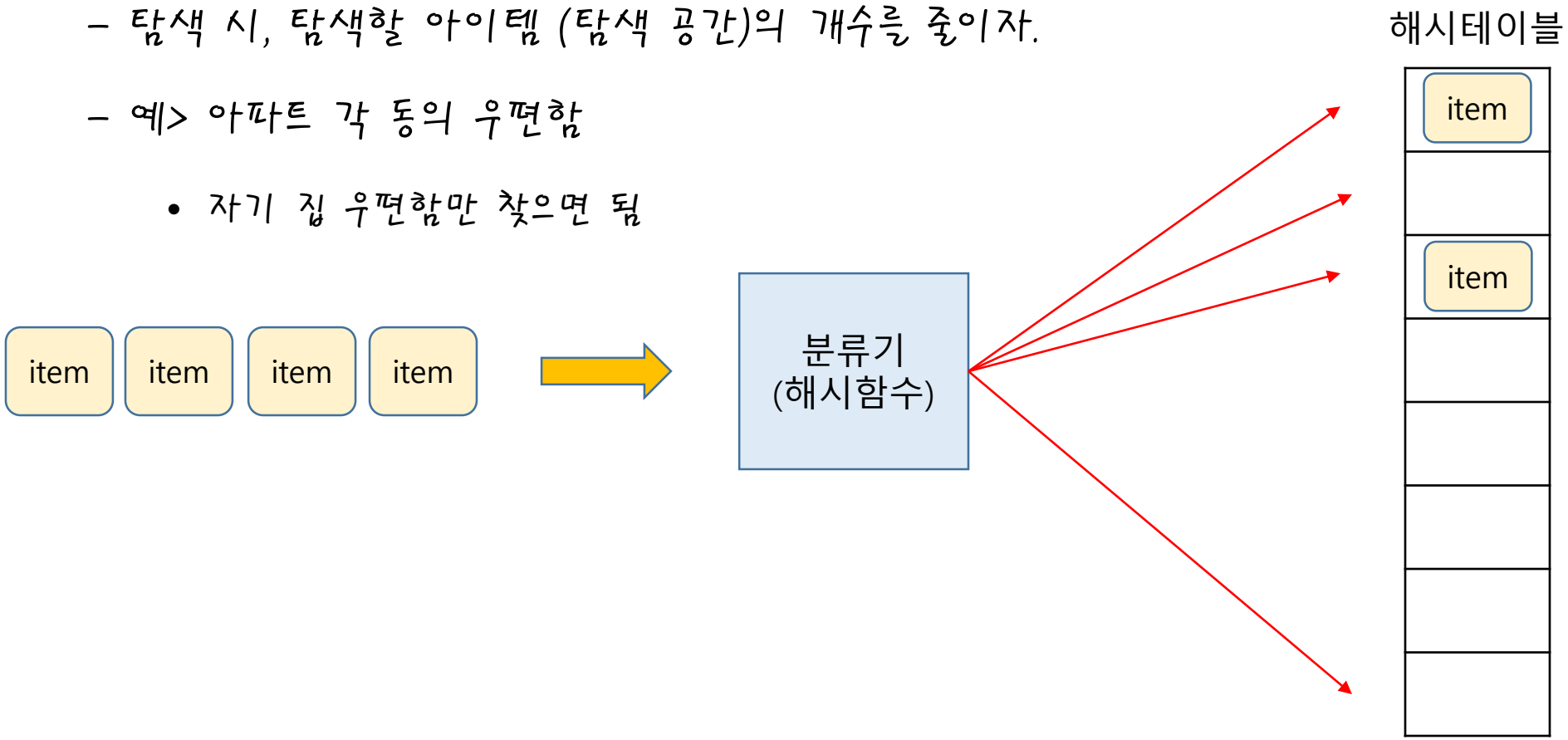
# Hashing (해싱)

# 해싱

- 연결 리스트, 스택, 큐 등은 모두 선형 구조
  - 구현 용이
  - 스택, 큐 등은 LIFO, FIFO 등의 정책을 구현하기에 효율적인 자료구조
  - 탐색에는 적합하지 않음
    - 선형 (순차) 탐색의 비효율성
  - 예> 아파트 우편물 배송 시스템
    - 큐로 구현한다면??? 우리 집 우편물 찾기 매우 어려움.
    - 해결책?

# 해싱

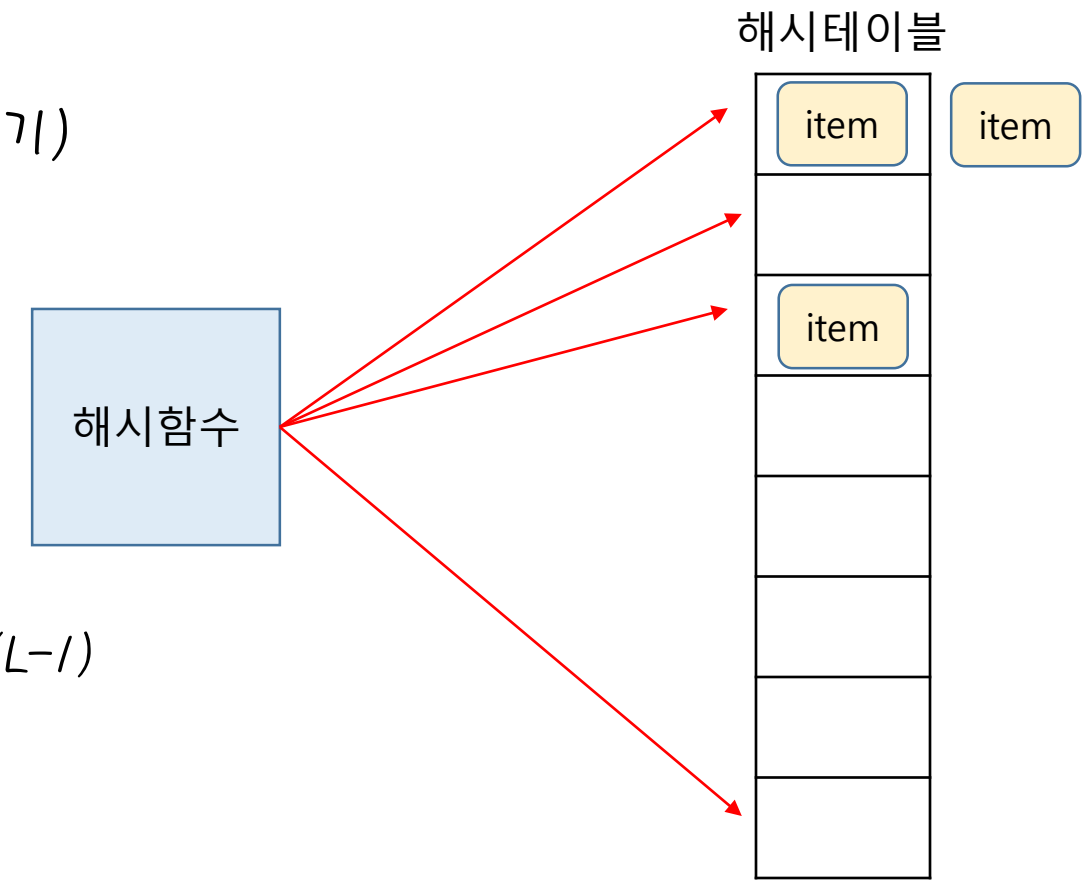
- 핵심 아이디어
  - 탐색 시, 탐색할 아이템 (탐색 공간)의 개수를 줄이자.
  - 예> 아파트 각 동의 우편함
    - 자기 집 우편함만 찾으면 됨



# 해시함수

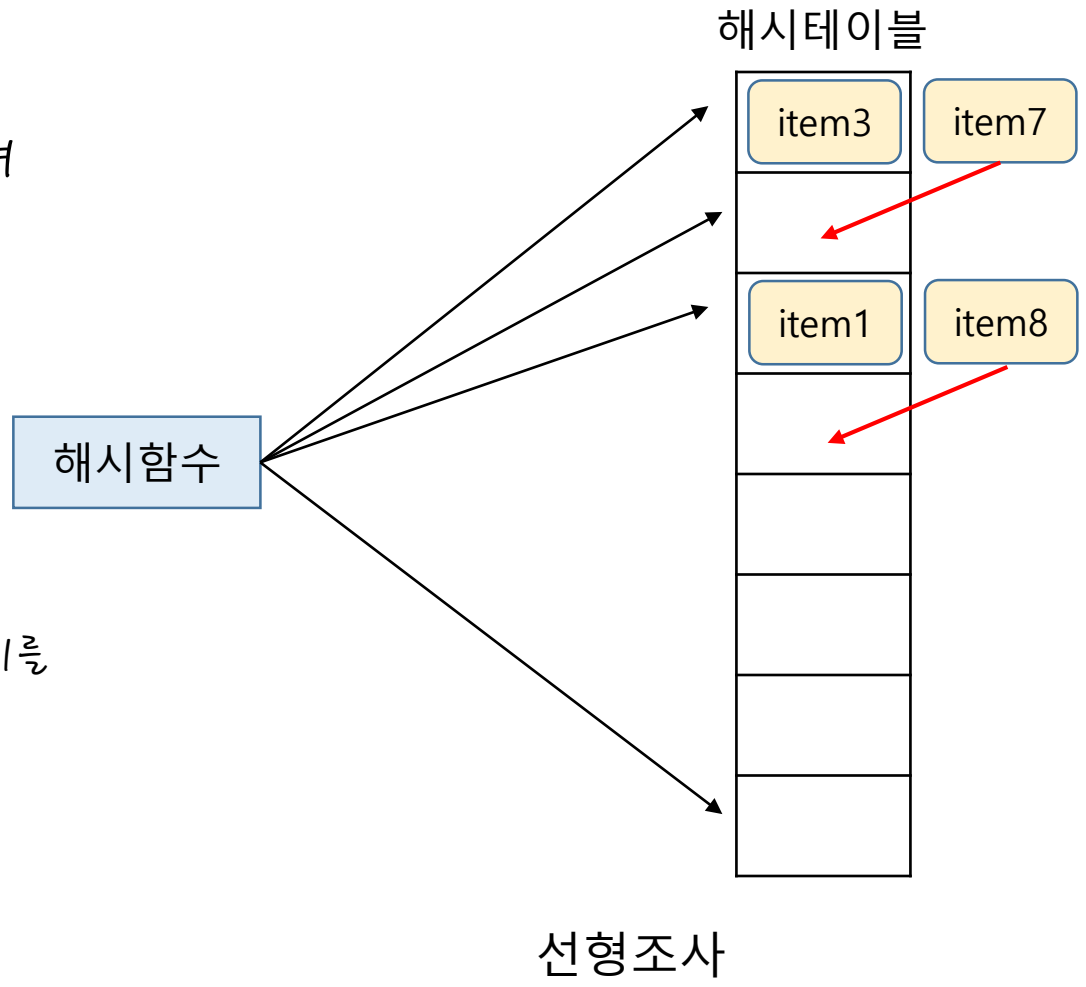
- 이상적인 해시함수의 조건
  - 균등 분산 (충돌 방지, 탐색 공간 줄이기)
  - low 계산 오버헤드

- 해시함수의 예
  - % (나머지 연산)
    - key % 해시테이블크기 (L) => 0 – (L-1)
  - 기타
    - (key를 이용한 여러 연산) % L



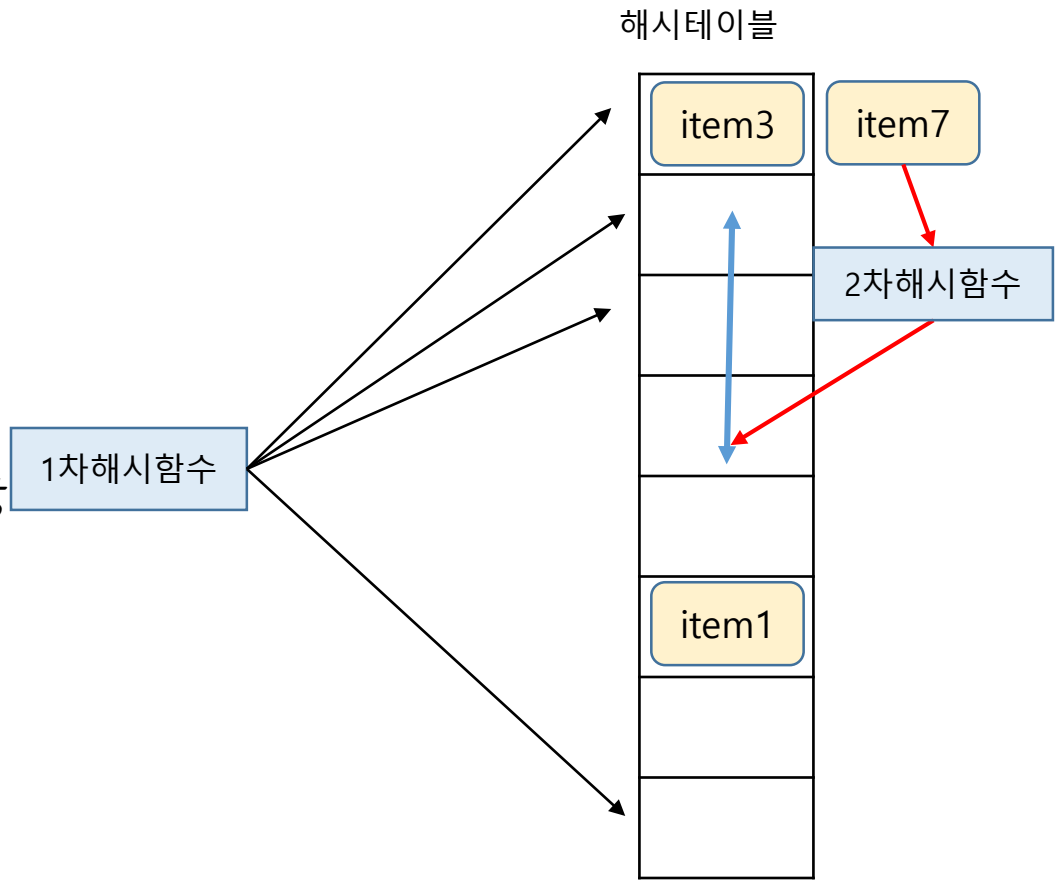
# 충돌 문제의 해결 방안

1. 선형조사 (linear probing)
   - 충돌이 발생하면, 순차적으로 다음의 빈 자리를 탐색하여 저장
   - 검색
     1. 해시함수를 통해 타겟 원소 발견
     2. 타겟 원소가 찾는 아이템인지 체크
     3. 아니면 다음 원소로 진행
     4. 2-3 작업을 빈 자리를 만날 때까지 (혹은 처음 위치에 이를 때까지) 반복
   - clustering (군집화) 발생 가능 & 성능 저하



# 충돌 문제의 해결 방안

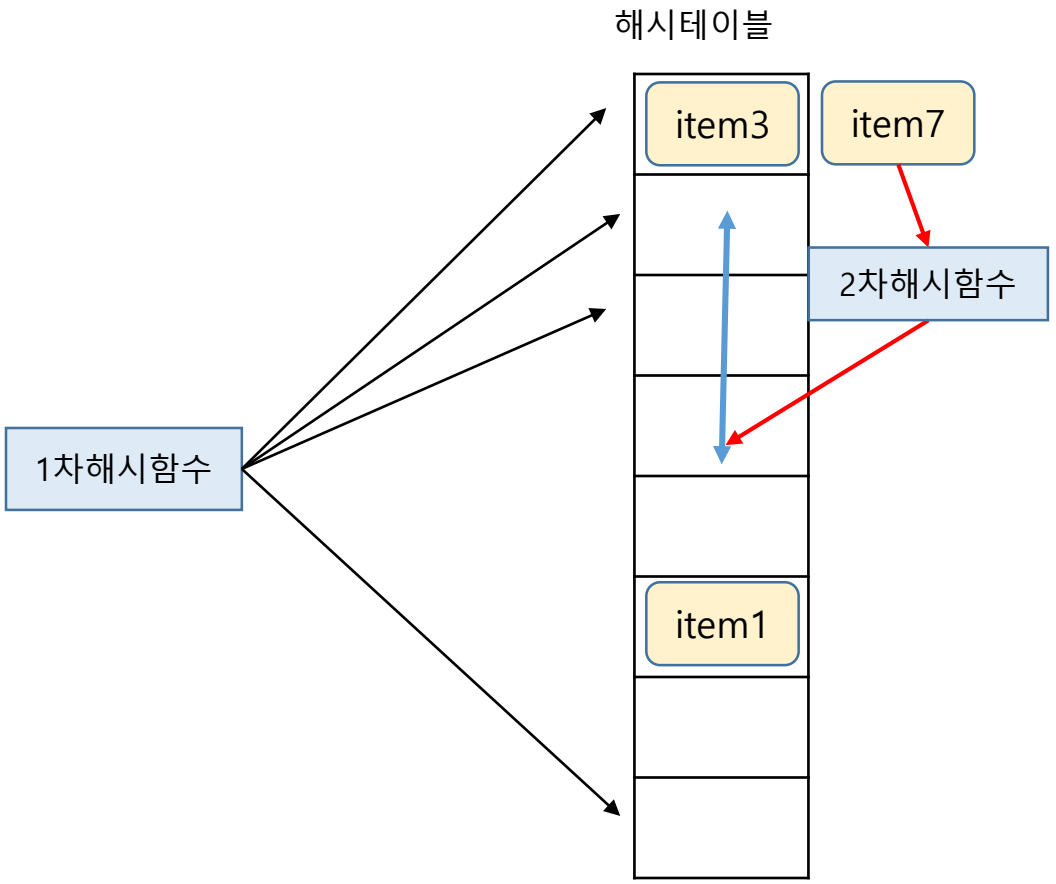
2. 이중해싱 (double hashing)
   - 충돌이 발생하면, 2차 해시함수를 적용하여 해당 distance 만큼 떨어진 위치를 다음 자리로 결정
   - 저장
     1. 1차 해시함수로 타겟 자리 결정 & 비어 있으면 저장
     2. 충돌 발생하면 2차 해시함수 적용하여 거리 계산
     3. 현재 타겟 자리에 거리를 더한 위치를 타겟 자리로 결정
     4. 빈 타겟 자리를 발견할 때까지 (혹은 처음 위치에 이를 때까지) 2-3번 반복.




이중해싱

# 충돌 문제의 해결 방안

2. 이중해싱 (double hashing)
   - 검색
     1. 1차 해시함수를 통해 타겟 원소 발견
     2. 타겟 원소가 찾는 아이템인지 체크
     3. 아니면 2차 해시함수를 적용하여 다음 원소로 진행
     4. 2-3 작업을 빈 자리를 만날 때까지 (혹은 처음 위치에 이를 때까지) 반복
   - clustering (군집화) 완화 (분산 저장)
   - 2차 해시함수의 결과값, distance와 해시테이블의 크기 L이 서로 소일 때 좋은 성능을 보임 => 해시테이블 크기 L을 소수로 선택하면 항상 서로 소가 됨.




이중해싱

# 충돌 문제의 해결 방안

3. 체이닝 (chaining)

- 해시테이블의 각 원소가 여러 item을 저장

  1. 2차원 배열을 활용하여 여러 원소를 저장할 수 있음 (단 저장 가능한 최대 개수가 배열의 column 값에 의해 결정됨)

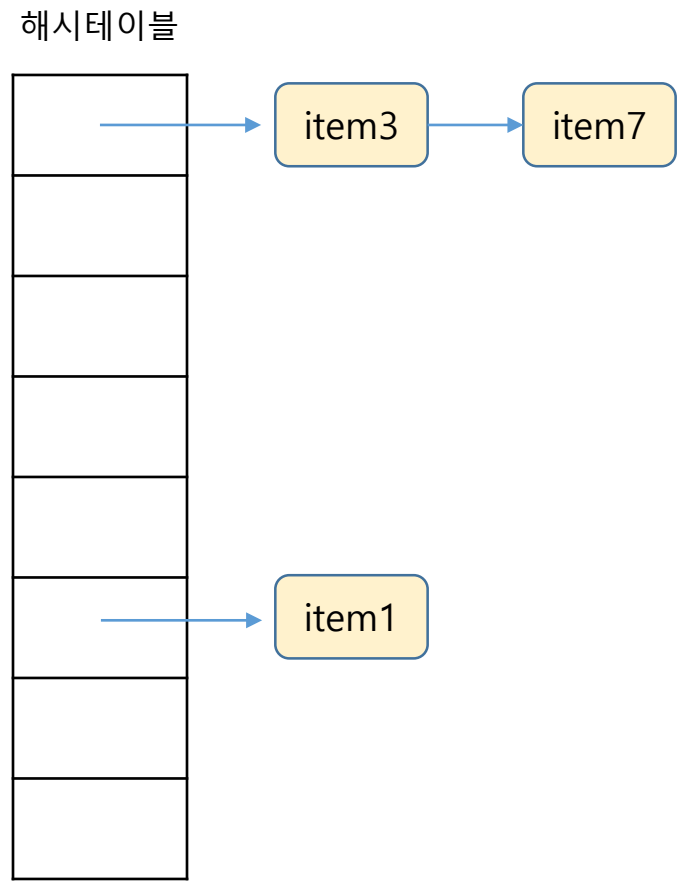
  2. 1차원 배열을 사용하되, 배열의 각 원소를 연결 리스트의 헤더로 사용. 동일한 원소로 해시된 item들을 연결 리스트로 연결

- 저장

  1. 1차 해시함수로 타겟 자리 결정
  2. 아이템을 기존의 연결 리스트에 추가

- 검색

  1. 1차 해시함수로 타겟 자리 결정
  2. 타겟 연결 리스트에서 검색




체이닝

# 선형조사 구현

- 클래스 LinearProbing 정의
  - 선형조사 방식의 해시테이블
  - two member variables
    - tablesize: 해시테이블의 크기
    - table: 해시테이블. 각 원소는 (key, value)의 tuple
  - 생성자, hash(), add(), search(), remove(), print_table()

```
def __init__(self, size):
    self.tablesize = size
    self.table = [(None, None)] * size
```

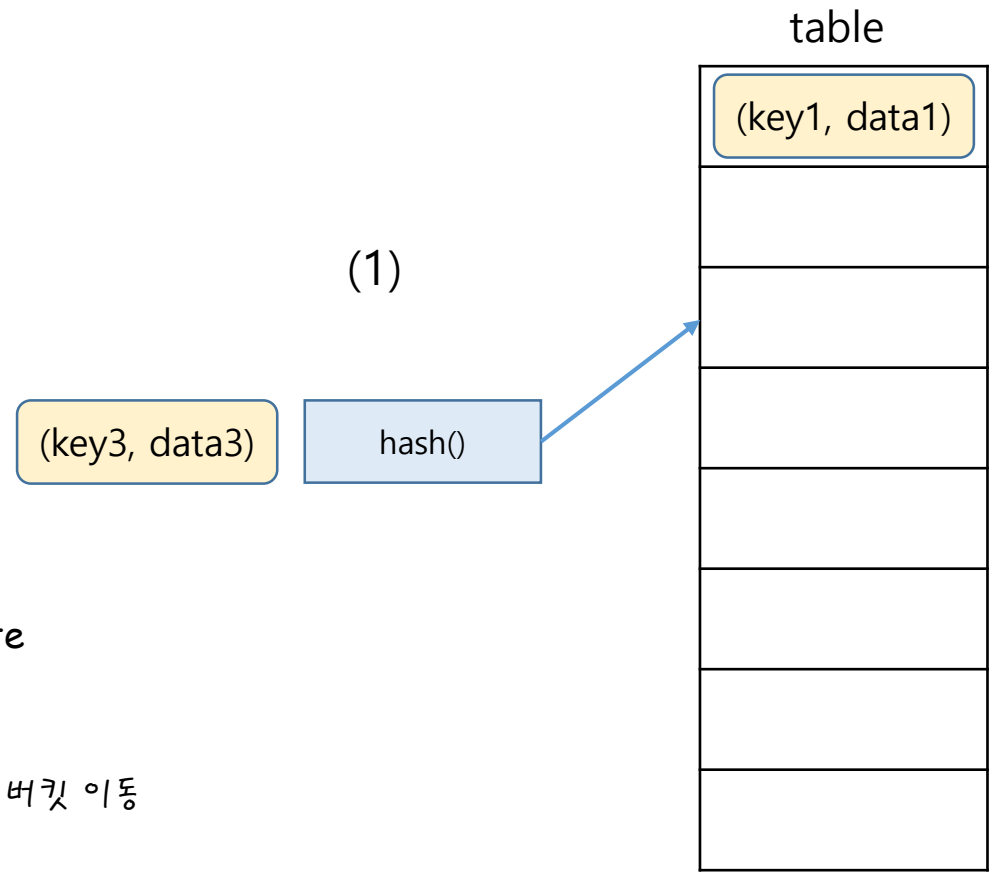
# 선형조사 구현

- 클래스 LinearProbing 정의
  - hash()

```
def hash(self, key):
    return key % self.tablesize
```

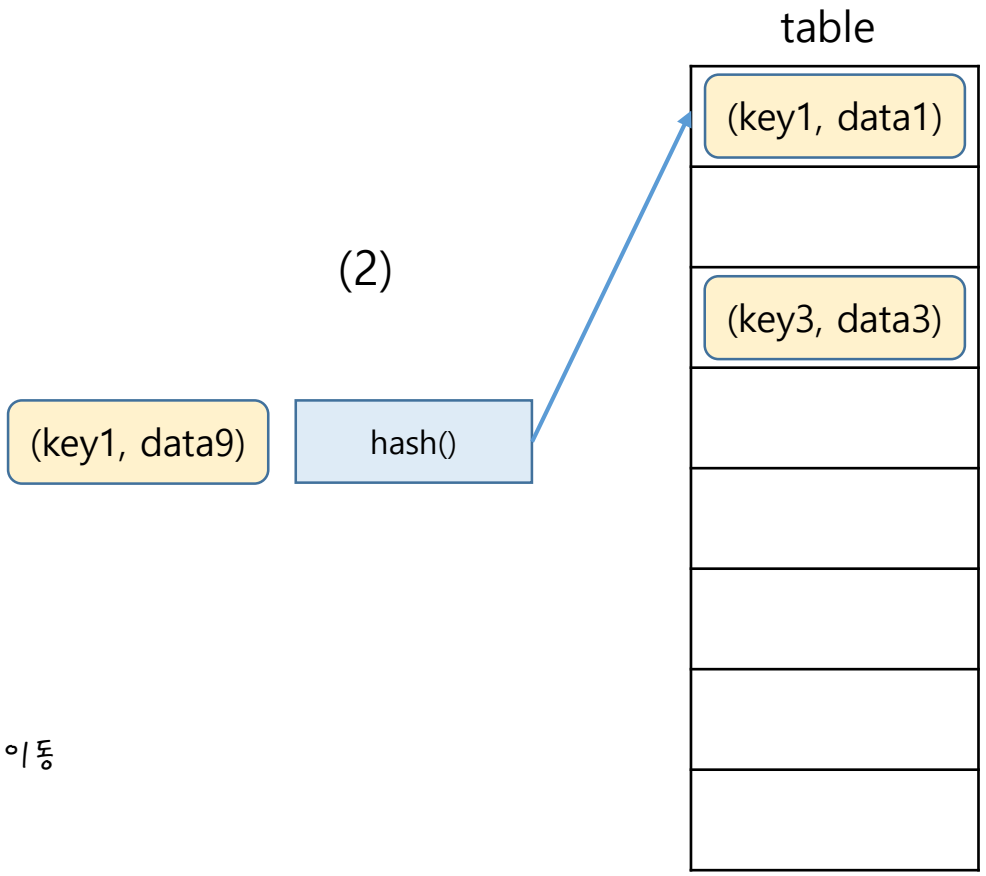
- 클래스 LinearProbing 정의
  - add()

```
def add(self, key, value):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fvalue == None:  # (1) 빈 버킷 발견 & 추가
            self.table[position] = (key, value)
            return  True
        elif fkey == key:    # (2) 동일 데이터 발견 & update
            self.table[position] = (key, value)
            return  True
        position = (position + 1) % self.tablesize  # 다음 버킷 이동

        if position == initial_position: # 추가할 빈 버킷 없음
            return False
```



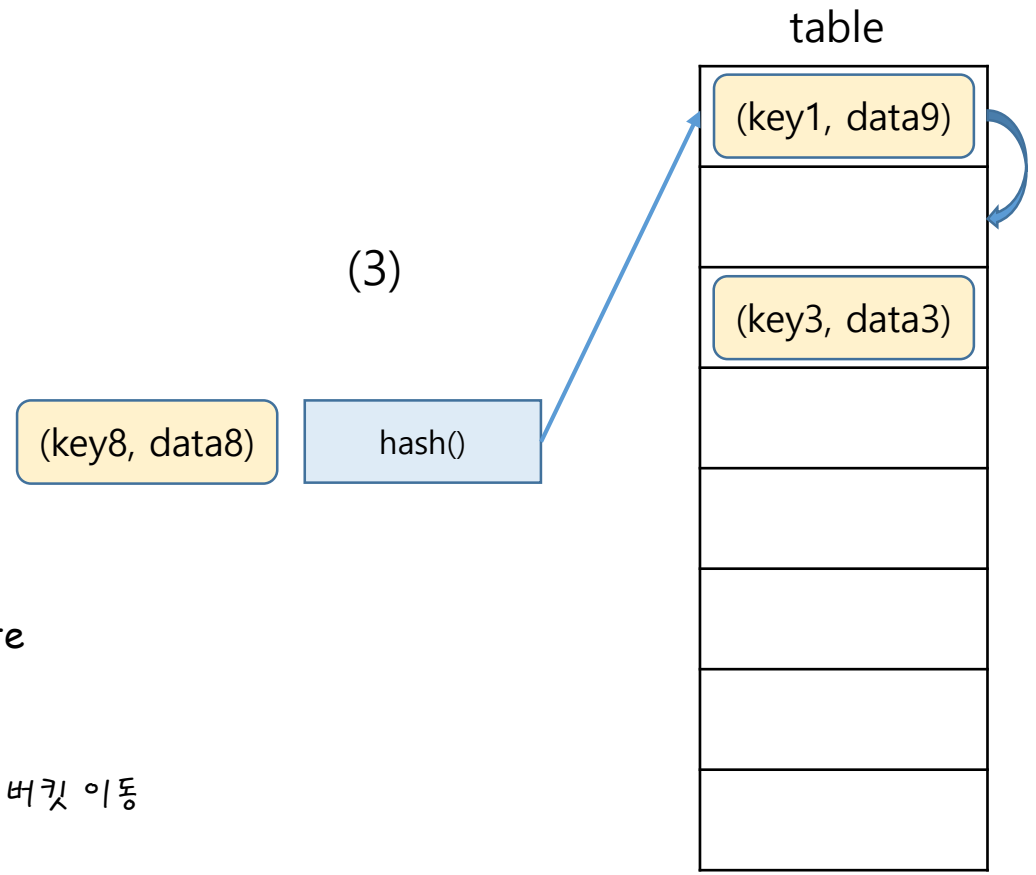
- 클래스 LinearProbing 정의
  - add()

```
def add(self, key, value):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fvalue == None:  # (1) 빈 버킷 발견 & 추가
            self.table[position] = (key, value)
            return  True
        elif fkey == key:    # (2) 동일 데이터 발견 & update
            self.table[position] = (key, value)
            return  True
        position = (position + 1) % self.tablesize  # 다음 버킷 이동

        if position == initial_position: # 추가할 빈 버킷 없음
            return False
```



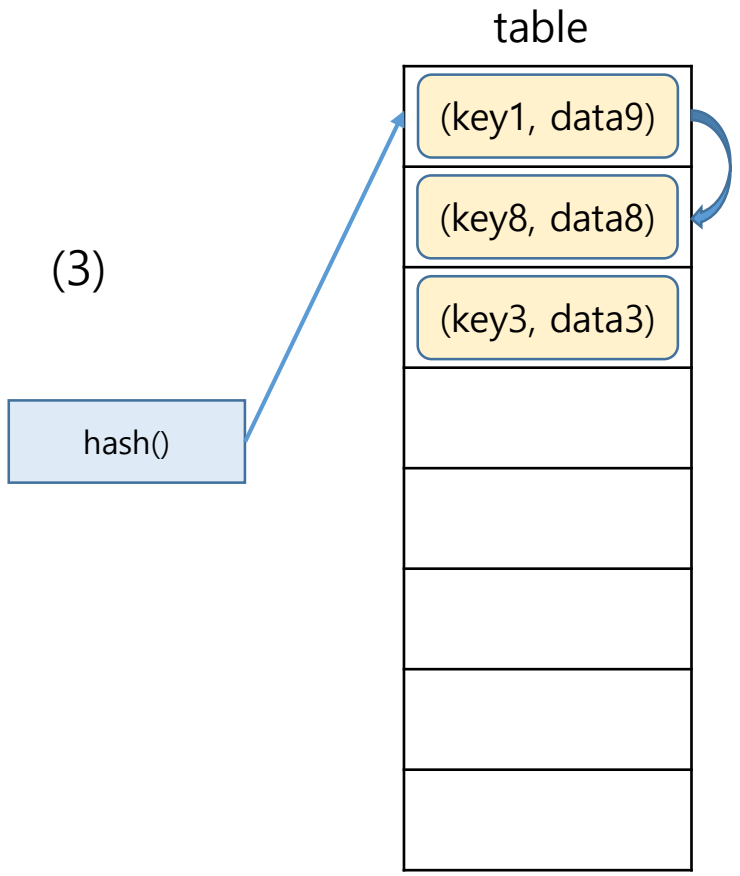
- 클래스 LinearProbing 정의
  - add()

```
def add(self, key, value):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fvalue == None:  # (1) 빈 버킷 발견 & 추가
            self.table[position] = (key, value)
            return  True
        elif fkey == key:    # (2) 동일 데이터 발견 & update
            self.table[position] = (key, value)
            return  True
        position = (position + 1) % self.tablesize  # 다음 버킷 이동

        if position == initial_position: # 추가할 빈 버킷 없음
            return False
```



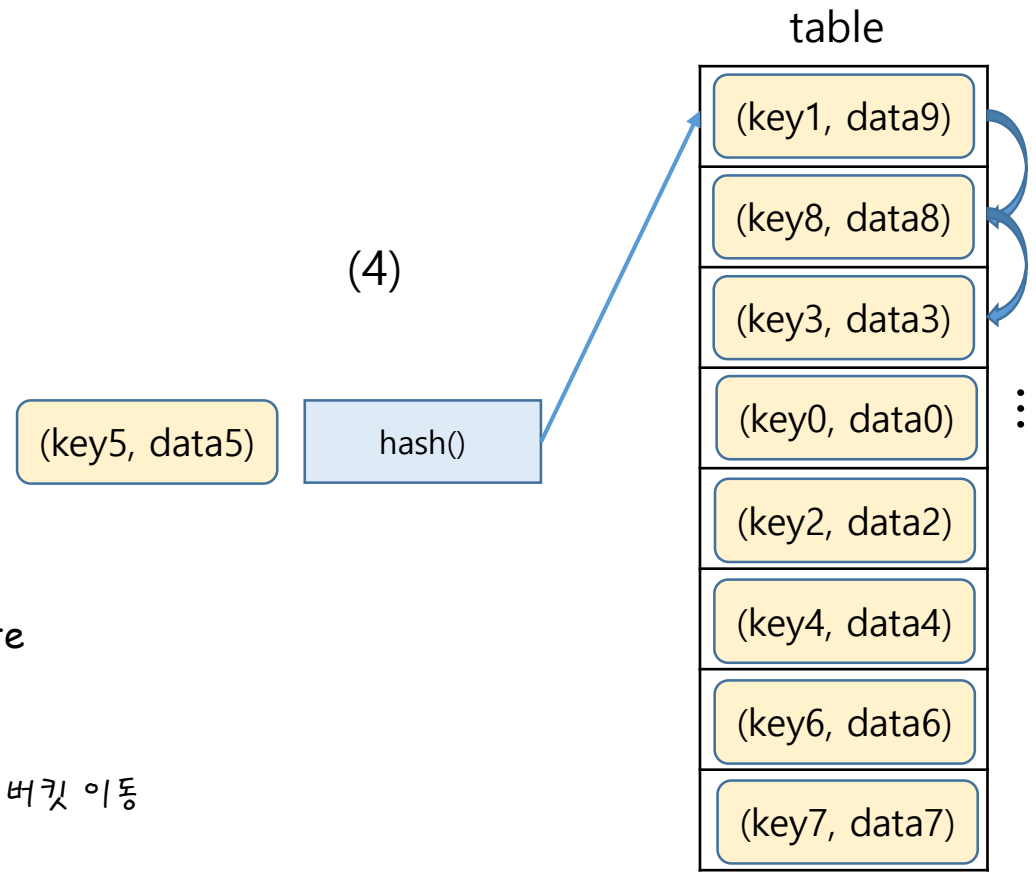
- 클래스 LinearProbing 정의
  - add()

```
def add(self, key, value):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fvalue == None:  # (1) 빈 버킷 발견 & 추가
            self.table[position] = (key, value)
            return  True
        elif fkey == key:    # (2) 동일 데이터 발견 & update
            self.table[position] = (key, value)
            return  True
        position = (position + 1) % self.tablesize  # 다음 버킷 이동

        if position == initial_position: # 추가할 빈 버킷 없음
            return False
```



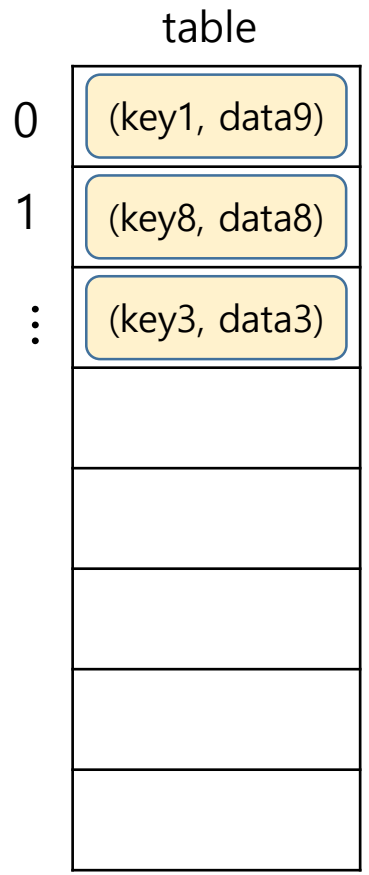
- 클래스 LinearProbing 정의
  - add()

```
def add(self, key, value):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fvalue == None:  # (1) 빈 버킷 발견 & 추가
            self.table[position] = (key, value)
            return  True
        elif fkey == key:    # (2) 동일 데이터 발견 & update
            self.table[position] = (key, value)
            return  True
        position = (position + 1) % self.tablesize  # 다음 버킷 이동

        if position == initial_position: # 추가할 빈 버킷 없음
            return False
```



- 클래스 LinearProbing 정의
  - print_table()

```
def print_table(self):
    i = 0
    for tuple in self.table:
        print(i, tuple)
        i += 1
    print()
```

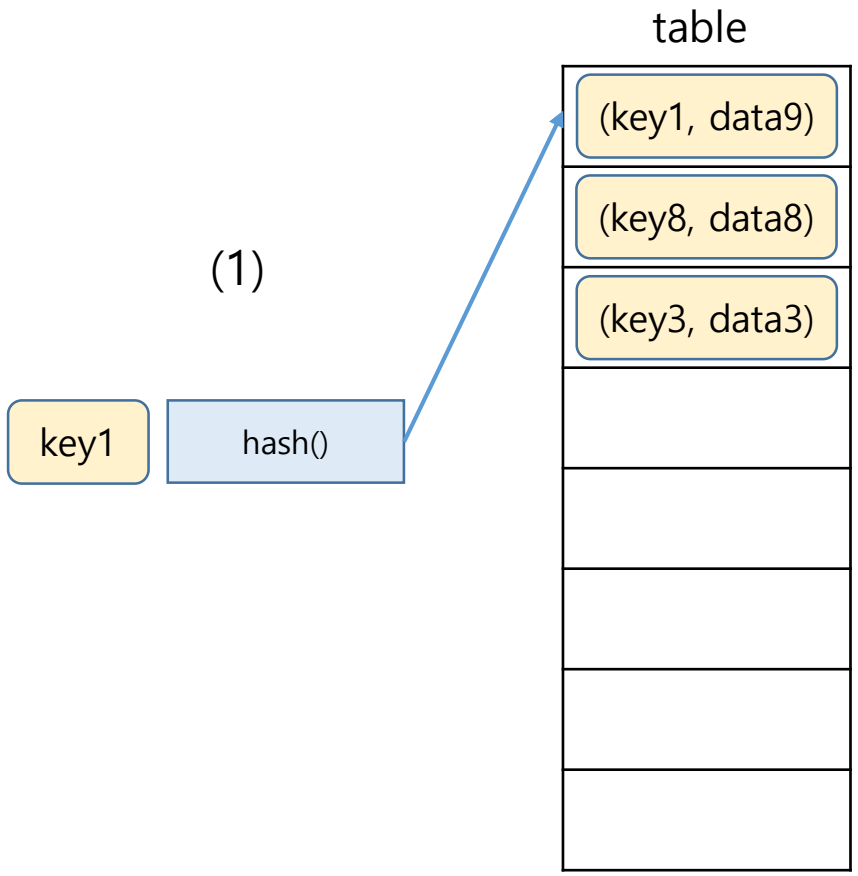


# 해시 활용 프로그램1

```
t = LinearProbing.LinearProbing(7)
t.put(7, 'grape')
t.put(1, 'apple')
t.put(2, 'banana')
t.print_table()
t.put(15, 'orange')
t.print_table()
```

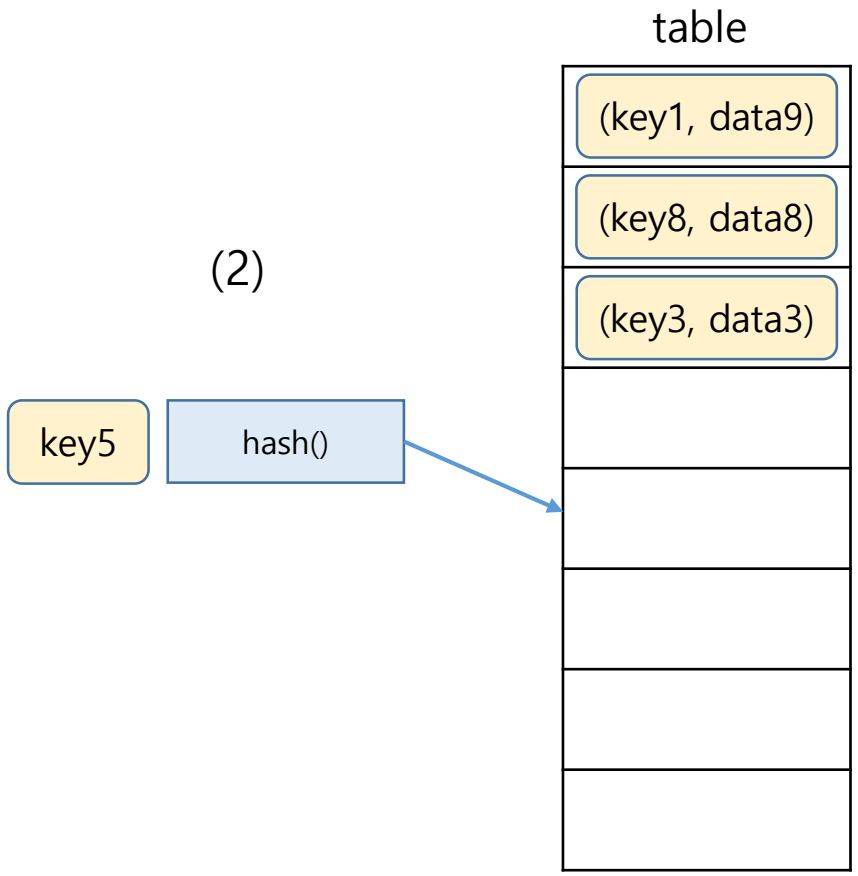
- 클래스 LinearProbing 정의
  - search()

```
def search(self, key):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fkey == key:        # (1)
            return fvalue
        if fkey == None:
            return None

        position = (position + 1) % self.tablesize
        if position == initial_position:
            return None
```



- 클래스 LinearProbing 정의
  - search()

```
def search(self, key):
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fkey == key:
            return fvalue
        if fkey == None:          # (2)
            return None

        position = (position + 1) % self.tablesize
        if position == initial_position:
            return None
```

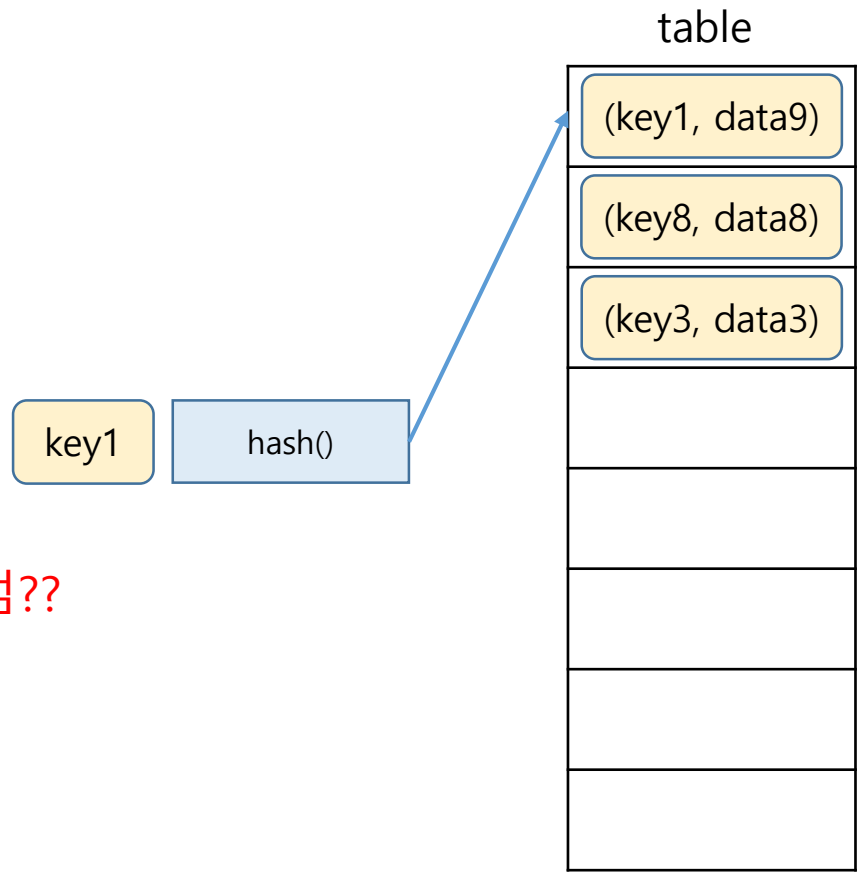


# 해시 활용 프로그램2

```
import LinearProbing

t = LinearProbing.LinearProbing(7)
t.put(7, 'grape')
t.put(1, 'apple')
t.put(2, 'banana')
t.print_table()
t.put(15, 'orange')
t.print_table()

print('탐색 결과:')
print('1의 data = ', t.search(1))
print('15의 data = ', t. search(15))
```

- 클래스 LinearProbing 정의
  - remove()

```
def remove(self, key): # 삭제 연산
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fkey == key:
            self.table[position] = (None, None)
            return True
        elif fkey == None:
            return False

        position = (position + 1) % self.tablesize
        if position == initial_position:
            return False
```



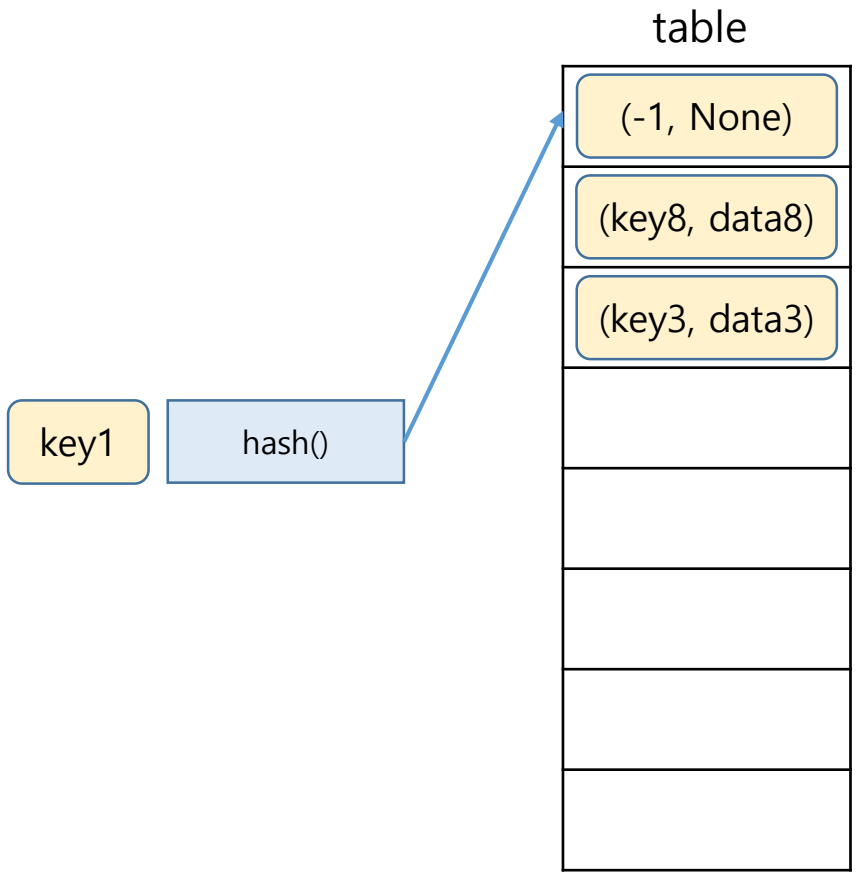

문제점??

# 해시 활용 프로그램3

```
t = LinearProbing(7)
t.add(7, 'grape')
t.add(1, 'apple')
t.add(2, 'banana')
t.add(15, 'orange')
t.print_table()

print('탐색 결과:')
print('1의 data = ', t.search(1))

t.remove(1)
t.print_table()
print('1의 data = ', t.search(1))
print('15의 data = ', t.search(15))          # orange를 찾을 수 없음!!
```

- 클래스 LinearProbing 정의
  - remove2()

```
def remove(self, key): # 삭제 연산
    initial_position = self.hash(key)
    position = initial_position

    while True:
        (fkey, fvalue) = self.table[position]
        if fkey == key:
            self.table[position] = (-1, None)
            return fvalue
        elif fkey == None:
            return None

        position = (position + 1) % self.tablesize
        if position == initial_position:
            return None
```



# 해시 활용 프로그램4

```
t = LinearProbing(7)
 t.add(7, 'grape')
 t.add(1, 'apple')
 t.add(2, 'banana')
 t.add(15, 'orange')
 t.print_table()

 print('탐색 결과:')
 print('1의 data = ', t.search(1))

 t.remove(1)
 t.print_table()
 print('1의 data = ', t.search(1))
 print('15의 data = ', t.search(15))
```

# Homework candidate

- Chaining 해시테이블 구현하기
  - insert(), search(), remove(), print_table()