Data Structure and Algorithms [CO2003]

Chapter 9 - Hash

Lecturer: Duc Dung Nguyen, PhD.
Contact: nddung@hcmut.edu.vn

Faculty of Computer Science and Engineering
Hochiminh city University of Technology

Contents

Outcomes

- **L.O.5.1** - Depict the following concepts: hashing table, key, collision, and collision resolution.
- **L.O.5.2** - Describe hashing functions using pseudocode and give examples to show their algorithms.
- **L.O.5.3** - Describe collision resolution methods using pseudocode and give examples to show their algorithms.
- **L.O.5.4** - Implement hashing tables using C/C++.
- **L.O.5.5** - Analyze the complexity and develop experiment (program) to evaluate methods supplied for hashing tables.
- **L.O.1.2** - Analyze algorithms and use Big-O notation to characterize the computational complexity of algorithms composed by using the following control structures: sequence, branching, and iteration (not recursion).

Basic concepts

- Sequential search: $O(n)$
- Binary search: $O(\log_2 n)$

$\rightarrow$ Requiring several key comparisons before the target is found.

Search complexity:

Size	Binary	Sequential (Average)	Sequential (Worst Case)
16	4	8	16
50	6	25	50
256	8	128	256
1,000	10	500	1,000
10,000	14	5,000	10,000
100,000	17	50,000	100,000
1,000,000	20	500,000	1,000,000

Is there a search algorithm whose complexity is $O(1)$?

Is there a search algorithm whose complexity is $O(1)$?

YES

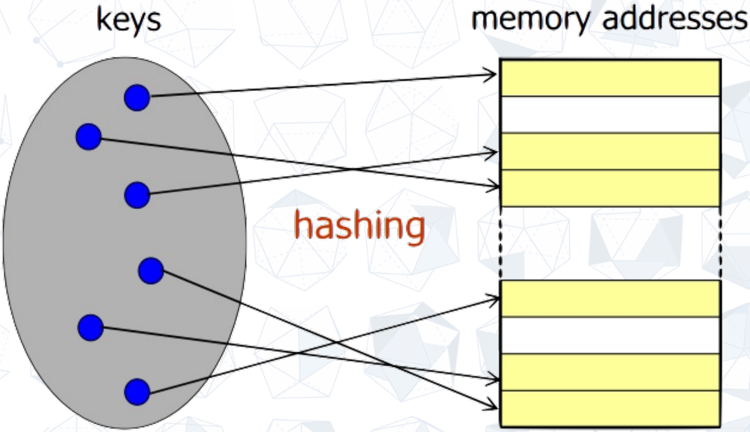
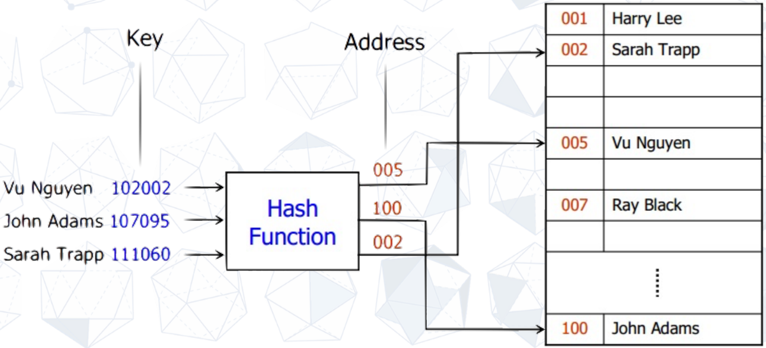


Figure 1: Each key has only one address

Basic concepts



Basic concepts

- Home address: address produced by a hash function.
- Prime area: memory that contains all the home addresses.

Basic concepts

- Home address: address produced by a hash function.
- Prime area: memory that contains all the home addresses.
- Synonyms: a set of keys that hash to the same location.
- Collision: the location of the data to be inserted is already occupied by the synonym data.

- Home address: address produced by a hash function.
- Prime area: memory that contains all the home addresses.
- Synonyms: a set of keys that hash to the same location.
- Collision: the location of the data to be inserted is already occupied by the synonym data.
- Ideal hashing:
    - No location collision
    - Compact address space

Basic concepts

Insert A, B, C

hash(A) = 9

hash(B) = 9

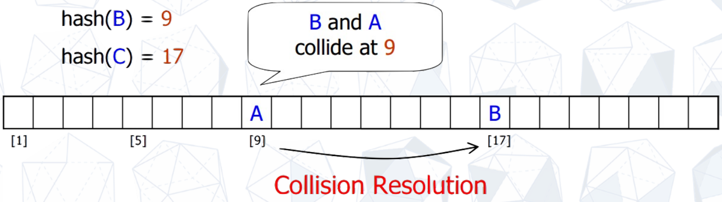
hash(C) = 17

Insert A, B, C

hash(A) = 9

hash(B) = 9

hash(C) = 17

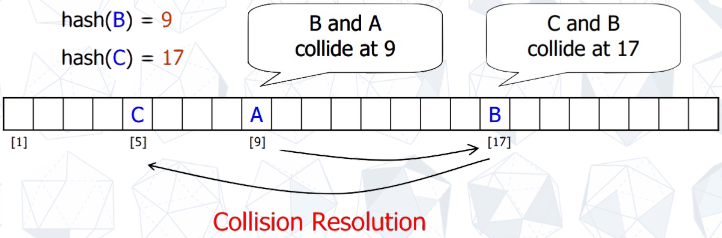


Insert A, B, C

hash(A) = 9

hash(B) = 9

hash(C) = 17

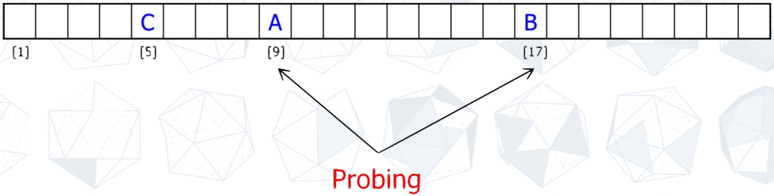


Searh for B

hash(A) = 9

hash(B) = 9

hash(C) = 17



Hash functions

Hash functions

- Direct hashing
- Modulo division
- Digit extraction
- Mid-square
- Folding
- Rotation
- Pseudo-random

The address is the key itself:

$hash(Key) = Key$

Direct Hashing

- Advantage: there is no collision.
- Disadvantage: the address space (storage size) is as large as the key space.

Modulo division

$$Address = Key\ mod\ listSize$$

- Fewer collisions if $listSize$ is a prime number.
- Example:
  Numbering system to handle 1,000,000 employees
  Data space to store up to 300 employees
  $hash(121267) = 121267\ mod\ 307 = 2$

$$Address = selected\ digits\ from\ Key$$

Example:

379452 → 394

121267 → 112

378845 → 388

160252 → 102

045128 → 051

Mid-square

$$Address = middle\ digits\ of\ Key^2$$

Example:

$9452 * 9452 = 89340304 \rightarrow 3403$

Mid-square

- Disadvantage: the size of the Key^2 is too large.
- Variations: use only a portion of the key.

  Example:

  379452: 379 * 379 = 143641→364 121267: 121 * 121 = 014641→464 045128: 045 * 045 = 002025→202

The key is divided into parts whose size matches the address size.

Example:
Key = 123|456|789
fold shift
123 + 456 + 789 = 1368
→ 368

The key is divided into parts whose size matches the address size.

Example:

Key $= 123|456|789$

fold shift

$123 + 456 + 789 = 1368$

$\rightarrow 368$

fold boundary

$321 + 456 + 987 = 1764$

$\rightarrow 764$

Rotation

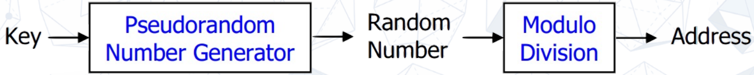
- Hashing keys that are identical except for the last character may create synonyms.
- The key is rotated before hashing.

original key	rotated key
600101	160010
600102	260010
600103	360010
600104	460010
600105	560010

- Used in combination with fold shift.

original key	rotated key
600101 → 62	160010 → 26
600102 → 63	260010 → 36
600103 → 64	360010 → 46
600104 → 65	460010 → 56
600105 → 66	560010 → 66

Spreading the data more evenly across the address space.



$$y = ax + c$$

For maximum efficiency, a and c should be prime numbers.

Pseudo-random

Example:

Key = 121267

a = 17

c = 7

listSize = 307

Address = ((17*121267 + 7) mod 307

= (2061539 + 7) mod 307

= 2061546 mod 307

= 41

Collision resolution

Collision resolution

- Except for the direct hashing, none of the others are one-to-one mapping
  $\rightarrow$ Requiring collision resolution methods
- Each collision resolution method can be used independently with each hash function

Collision resolution

- Open addressing
- Linked list resolution
- Bucket hashing

Open addressing

When a collision occurs, an unoccupied element is searched for placing the new element in.

Open addressing

Hash function:

$h : U \to \{0, 1, 2, ..., m-1\}$

set of keys addresses

Hash and probe function:

$hp : U \times \{0, 1, 2, ..., m-1\} \rightarrow \{0, 1, 2, ..., m-1\}$

set of keys   probe numbers   addresses

Algorithm hashInsert(ref T <array>, val k <key>)
Inserts key k into table T.

```
i = 0
while i < m do
    j = hp(k, i)
    if T[j] = nil then
        T[j] = k
        return j
    else
        i = i + 1
    end
end
return error: "hash table overflow"
```

End hashInsert

```
Algorithm hashSearch(val T <array>, val k <key>)
Searches for key k in table T.

i = 0
while i < m do
    j = hp(k, i)
    if T[j] = k then
        return j
    else if T[j] = nil then
        return nil
    else
        i = i + 1
    end
end
return nil
End hashSearch
```

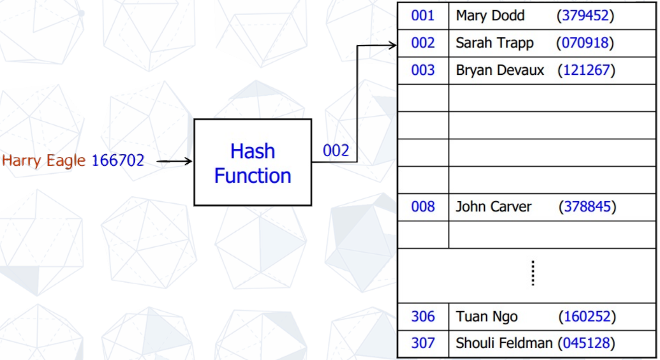
There are different methods:

- Linear probing
- Quadratic probing
- Double hashing
- Key offset

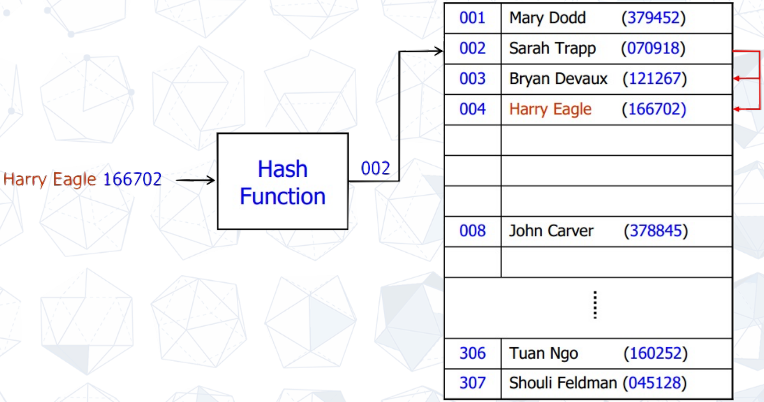
Linear Probing

- When a home address is occupied, go to the next address (the current address + 1):
  $hp(k, i) = (h(k) + i) \; mod \; m$

- When a home address is occupied, go to the next address (the current address + 1):
$hp(k, i) = (h(k) + i)\ mod\ m$



Linear Probing



Linear Probing

- Advantages:
  - quite simple to implement
  - data tend to remain near their home address (significant for disk addresses)
- Disadvantages:
  - produces primary clustering

Quadratic Probing

- The address increment is the collision probe number squared:
  $hp(k, i) = (h(k) + i^2)\ mod\ m$

Quadratic Probing

- Advantages:
  - works much better than linear probing
- Disadvantages:
  - time required to square numbers
  - produces secondary clustering
    $h(k_1) = h(k_2) \to hp(k_1, i) = hp(k_2, i)$

- Using two hash functions:
  $$hp(k,i) = (h_1(k) + ih_2(k))\ mod\ m$$

Key Offset

- The new address is a function of the collision address and the key.

$$offset = [key/listSize]$$
$$newAddress = (collisionAddress + offset) \ mod \ listSize$$

- The new address is a function of the collision address and the key.

$$offset = [key/listSize]$$
$$newAddress = (collisionAddress + offset)\ mod\ listSize$$

$$hp(k, i) = (hp(k, i-1) + [k/m])\ mod\ m$$

Hash and probe function:

$hp : U \times \{0, 1, 2, ..., m-1\} \rightarrow \{0, 1, 2, ..., m-1\}$

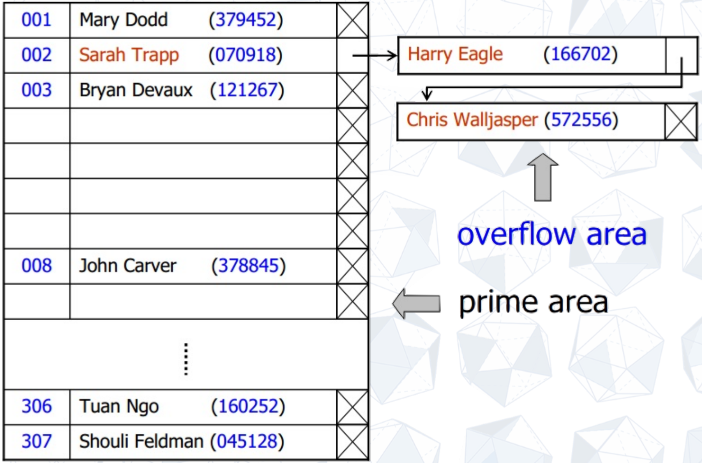
set of keys probe numbers addresses

$\{hp(k, 0), hp(k, 1), \ldots, hp(k, m-1)\}$ is a permutation of $\{0, 1, \ldots, m-1\}$

Linked List Resolution

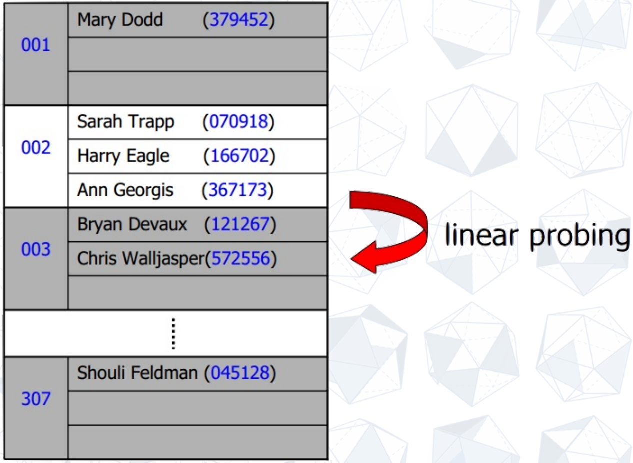
- Major disadvantage of Open Addressing: each collision resolution increases the probability for future collisions.
  $\rightarrow$ use linked lists to store synonyms

Linked list resolution



Bucket hashing

- Hashing data to buckets that can hold multiple pieces of data.
- Each bucket has an address and collisions are postponed until the bucket is full.


001
Mary Dodd (379452)
002
Sarah Trapp (070918)
Harry Eagle (166702)
Ann Georgis (367173)
003
Bryan Devaux (121267)
Chris Walljasper(572556)
307
Shouli Feldman (045128)
linear probing