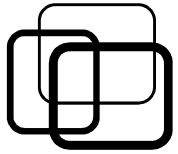
Search Algorithms (cont)

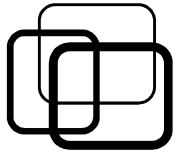
Inst. Nguyễn Minh Huy

Contents



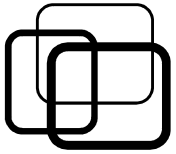
- Hash function.
- Hash table.
- Collision handling.

Contents



- **Hash function.**
- Hash table.
- Collision handling.

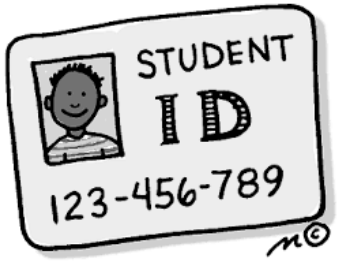
Hash function



- Basic concepts:

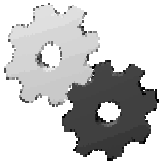
Student identity:

- Name?
- Age?
- Hair color?
- Hobbies?



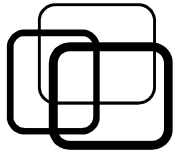
Number identity
IS SIMPLER!!

- A hash function is…
  - A mapping function.
  - Convert key to integer.

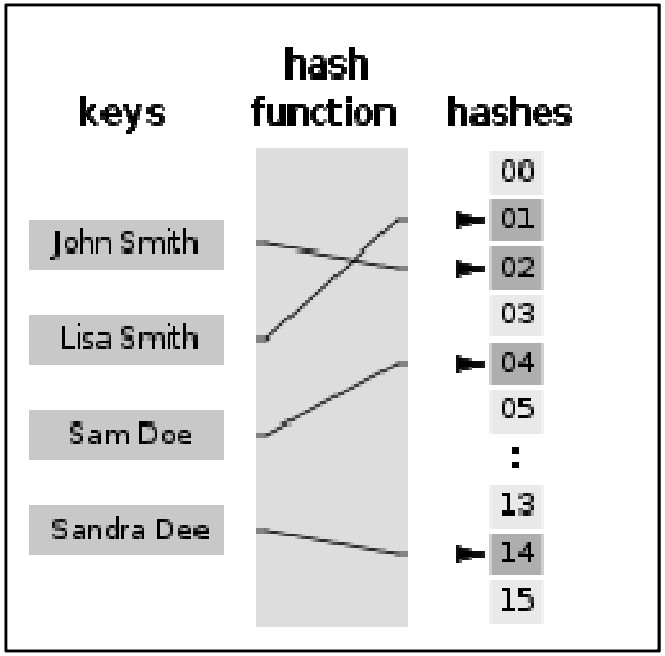


$$H: U \longrightarrow V$$
$$k \longrightarrow h = H(k)$$

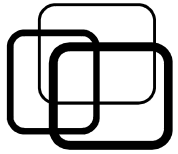
Hash function



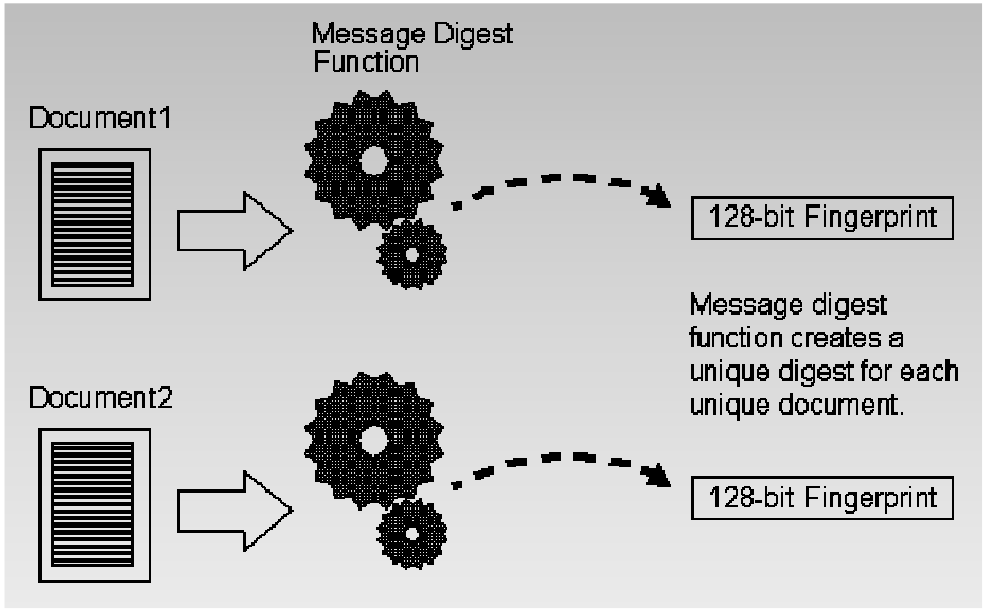
- Characteristics:
  - The key:
    - Variable length.
    - Big or complex value.
    - Big number, string, object.
  - The hash:
    - Fixed length.
    - 32-bit, 64-bit, 128-bit integer.
    - Key space BIGGER hash space.
  - The function:
    - Key mapped to unique hash (perfectly).
    - Fast to compute: H(k) ~ O(1).
    - Hard to invert: $H^{-1}(k)$ ~ O(key space).



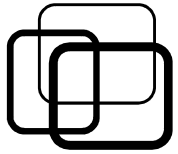
Hash function



- Hash function applications:
  - Hash table indexing (see below).
  - Checksum.
  - Cryptograph.

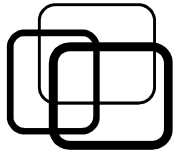


Hash function



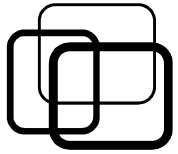
- Hash function examples:
  - Big integer?
  - String?
  - Object?

Contents



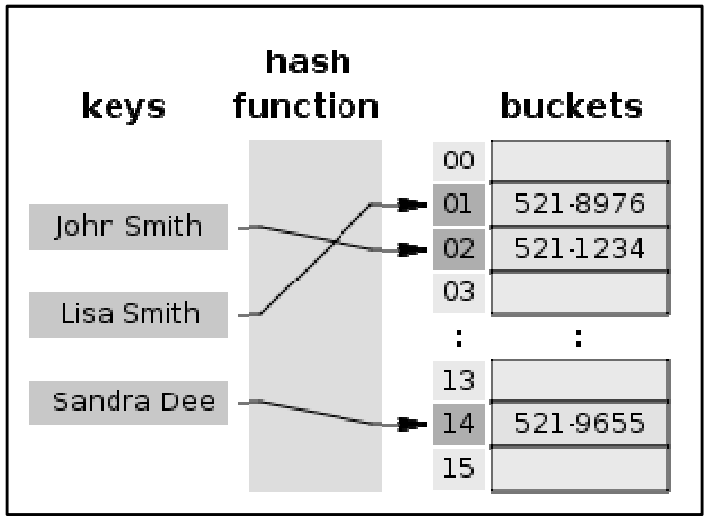
- Hash function.
- **Hash table.**
- Collision handling.

Hash table

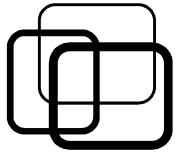


- Hash table idea:
  - An O(1) search data structure.
  - How to achieve O(1) search?
    - Given array A.
    - Given hash function H.
    - Given element k.
    - h = H(k) is the hash.
    - k is stored at h in A.

  ➔ Element is stored at its hash.



Hash table

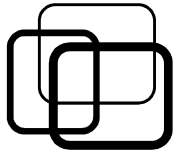


- ADT Hash table:
  - Values: array of elements.

```
class HashTable {
    vector<int *> m_elements;
};
```

  - Operations:
    - Initialize: create empty table.
    - Contains: check if a key exists.
    - Find: look for a key.
    - Add: insert a key.
    - Remove: delete a key.

Hash table



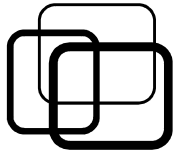
- Hash table implementation:
  - Find: look for a key.

```
FindKey( array A, key k ) {
    Get hash h = H( k ).
    if A[ h ] is empty  →  return not found.
    else  →  return found.
}
```

  - Add: insert a key.

```
AddKey( array A, key k ) {
    Get hash h = H( k ).
    if A[ h ] is empty  →  Store k at h.
    else  →  Resolve collision.
}
```

Hash table

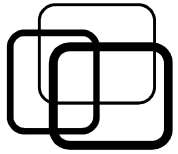


- Complexity analysis:

Operations	Average	Worst
Search	O(1)	O(n)
Add	O(1)	O(n)
Remove	O(1)	O(n)

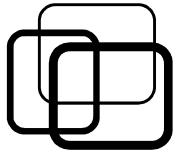
  - Additional space: 4*n bytes.
  - Efficient way to store **un-ordered, intensively searched** array.

Contents

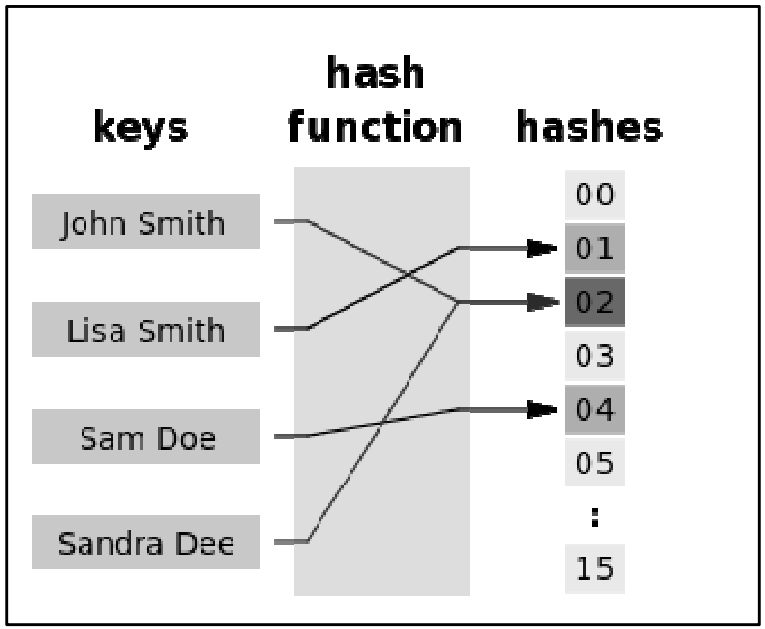


- Hash function.
- Hash table.
- **Collision handling.**

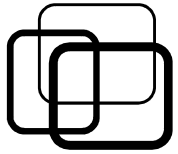
Collision handling



- Collision problem:
  - Key space BIGGER hash space.
    - ➔ Two keys have same hash.
  - Definition:
    - Given hash function H.
    - If two keys k1 ≠ k2.
    - But H(k1) = H(k2).

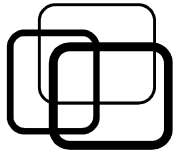


Collision handling

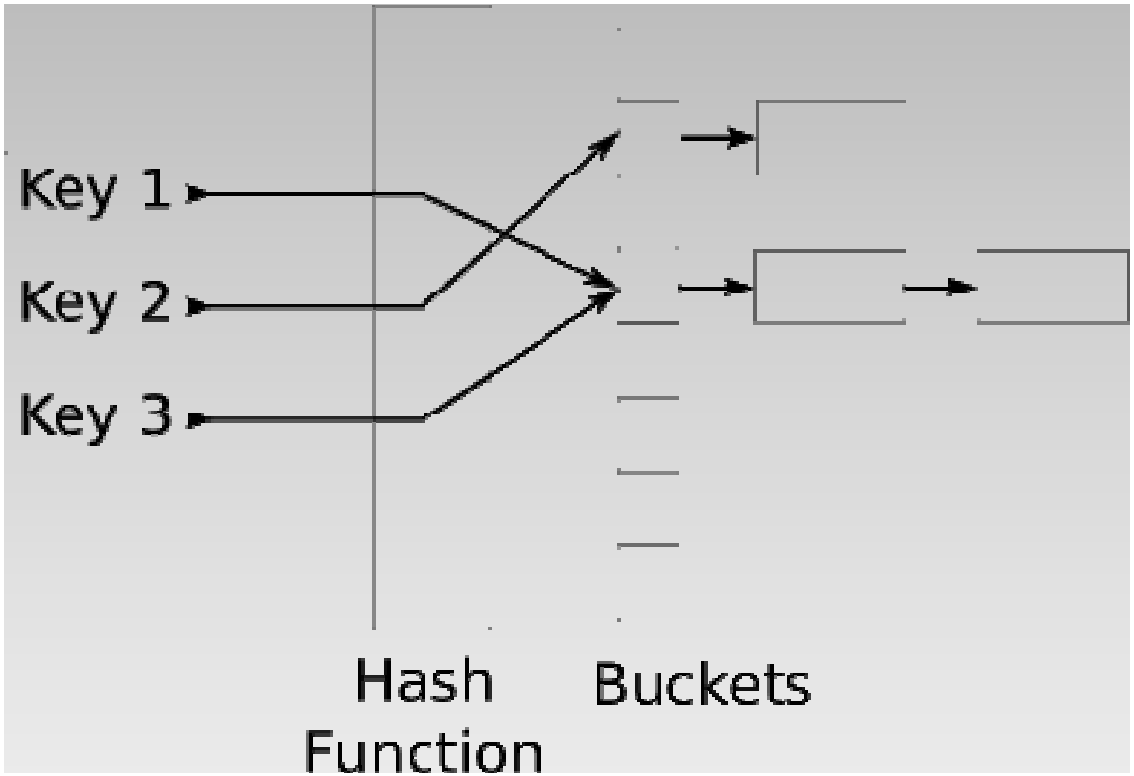


- Collision resolutions:
  - Chaining.
  - Open addressing:
    - Linear probing.
    - Quadratic probing.
    - Double hashing.

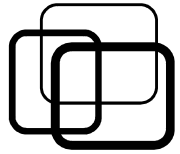
Collision handling



- Collision resolutions:
  - Chaining:
    - Collided keys are linked by singly linked list.

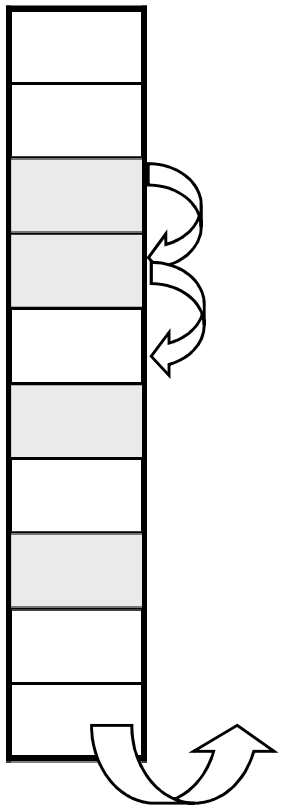


Collision handling

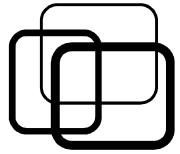


- Collision resolutions:
  - Open addressing – Linear probing:
    - Collided keys are added next to each other.
    - Probing function:

      P(k, i) = (H(k) + i) mod N.



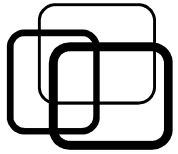
Collision handling



- Collision resolutions:
  - Open addressing – Quadratic probing:
    - Collided keys are added in quadratic increment.
    - Probing function:

      $P(k, i) = (H(k) + i^2) \bmod N.$

Collision handling



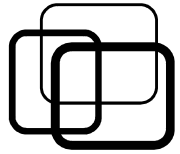
- Collision resolutions:
  - Open addressing – Double hashing:
    - Collided keys are added by second hash function.
    - Probing function:

      $P(k, i) = (H(k) + i * H_2(k)) \bmod N.$
    - Second hash function:

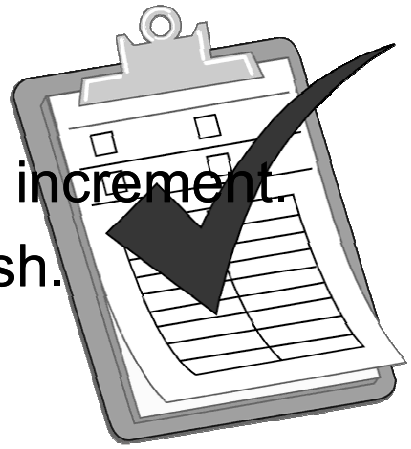
      $H_2(k) = R + (k \bmod R)$

      R is prime number.

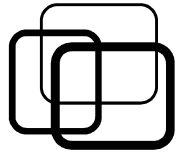
Summary



- Hash function:
  - Mapping function converting key to integer.
- Hash table:
  - Array storing element at its hash.
- Collision handling:
  - Chaining: link collides by singly linked list.
  - Open addressing:
    - Linear probing: stores collides in sequence.
    - Quadratic probing: stores collides in quadratic increment.
    - Double hashing: stores collides by second hash.



Practice



- ## Practice 4.1:

  Implement ADT **HashTable** (of integer) with values and operations.

  Try using different collision resolutions:

  - Chaining.
  - Linear probing.
  - Double hashing.