NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC1007
Hash Table

Dr. Liu Siyuan
Email: syliu@ntu.edu.sg
Office: N4-02C-72a

Overview

- Direct address table
- Hashing
  - Hash functions
- Collisions
  - Closed Addressing: Separate Chaining
  - Open Addressing
    - Probing: Linear Probing, Quadratic Probing, Double Hashing
- Delete keys
- Rehashing

Hash Table

- A typical space and time trade-off in algorithms
- A **hash table** is a data structure that allows for efficient lookup, insertion, and deletion of key-value pairs.
- To achieve search time in $O(1)$, we can use hash tables but memory usage will be increased significantly.

Some Applications of Hash Tables

- **Caching**: Hash tables are commonly used in caching applications to store frequently accessed data. The hash table can be used to store key-value pairs, where the key is a unique identifier for the data and the value is the data itself.
- **Databases**: Hash tables are often used in databases to provide fast lookups of data. For example, a hash table can be used to store a table of users, with the user ID as the key and the user's information as the value.
- **Counting**: Hash tables can be used to count occurrences of items in a dataset. Each item can be stored as a key in the hash table, with the value representing the count of occurrences. This can be used in applications like word frequency analysis, where the frequency of words in a document is analyzed.
- **Cryptography**: Hash tables are used in cryptography to store password hashes. When a user logs in, their password is hashed and compared to the hashed value stored in the hash table. This allows for secure authentication without storing the actual passwords in the hash table.

Design a Hash Table

- Suppose you are storing students information? How to find and store data effectively?
- To design a hash table for data retrieval, we need to consider
  - Hash function: a function to map data of arbitrary size to fixed-size values or keys.
  - Collision and its resolutions
  - Delete a key from a hash table
  - Resizing hash table – Dynamic hash table

Direct-Address Table

- Assume that the keys of elements K are drawn from the universe of possible keys U
- No two elements have the same key
- Search time is O(1) but …
    - The array size is enormous
    - |U| >> |K|
    - if the keys are integers between 1 and 1000, the array used to implement the direct-address table must have 1000 elements.

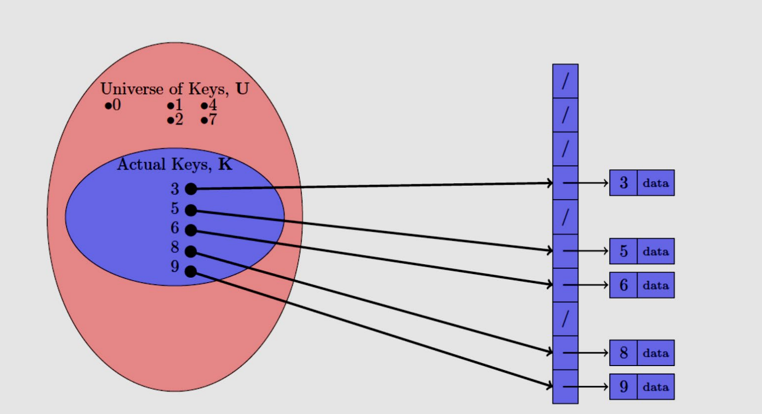


Figure 4.1: Direct Address Table

Hashing

- To reduce the key space to a reasonable size
- Hashing is the process of using a hash function to map data of arbitrary size to fixed-size values of keys

  hash function: {all possible keys} → {0, 1, 2, …, h-1}

- Each key is mapped to a unique index (**hash value/code/address**)
- Search time remains O(1) on the average
- The array is called a **hash table**
- Each entry in the hash table is called a **hash slot**
- When multiple keys are mapped to the same hash value, a **collision** occurs
- If there are n records stored in a hash table with h slots, its **load factor** is $\boldsymbol{\alpha} = \frac{n}{h}$

Hash Functions

- Must map all possible value to the range of the hash table uniquely
- Mapping should achieve an even distribution of the keys
- Easy and fast to compute
- Minimize collision

1. Modulo Arithmetic
2. Folding
3. Mid-square
4. Multiplicative Congruential Method
5. Etc.

Hash Functions

1. Modulo Arithmetic: $\mathrm{H}(k) = k \bmod h$, where *k* is the key and *h* is the hash table size

E.g., h=13, k = 37699 → H(37699) = 37699 mod 13 = 12

For keys are chosen from

- decimal number → h avoid to use powers of 10
- unknown lower p-bit patterns → h avoid to use powers of 2
- "real" data → h should be a prime number but not too close to any power of 2

Hash Functions

1. Modulo Arithmetic: $\mathrm{H}(k) = k \bmod h$

E.g., h=13, k = 37699 → H(37699) = 37699 mod 13 = 12

For keys are chosen from

- decimal number → h avoid to use powers of 10
- unknown lower p-bit patterns → h avoid to use powers of 2
- “real” data → h should be a prime number but not too close to any power of 2

2. Folding
   - Partition the key into several parts and combine the parts in a convenient way
   - Shift folding: Divide the key into a few parts and add up these parts
   - X = abc → (a+b+c) mod h
   - E.g., H(123456789) = (123+456+789) mod 13 = 3
   - We can apply this when the key is super long, e.g., 20 digits

Hash Functions

3. Mid-square
   - The key is squared and the middle part of the result is used as the hash address
     - E.g., k=3121, $k^2 = 3121^2 = 9740641$ → H(k) = 406

4. Multiplicative Congruential Method
   - Pseudo-random number a
   - $H(k) = (a \times k)\ mod\ h$
   - E.g., k=5, a=6, h=13 → $H(k) = (6 \times 5)\ mod\ 13 = 5$

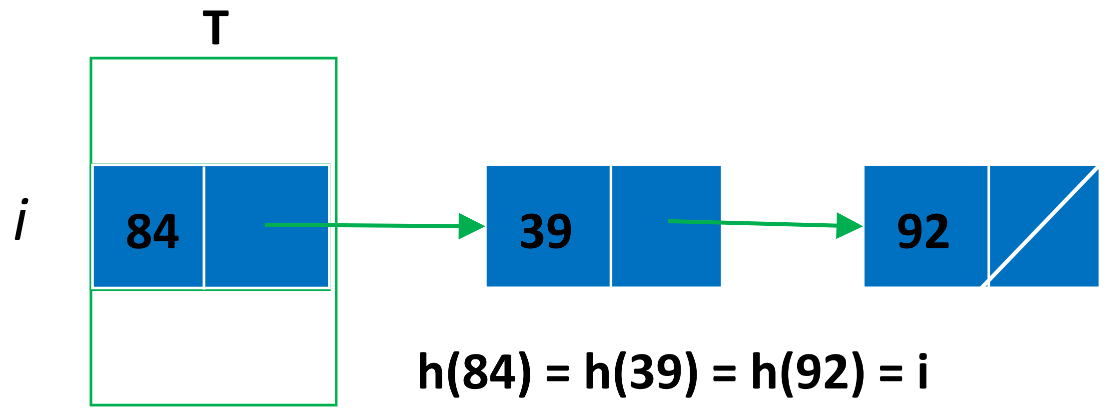
Collision Resolutions

- Closed Addressing Hashing – a.k.a separate chaining

- Open Addressing Hashing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

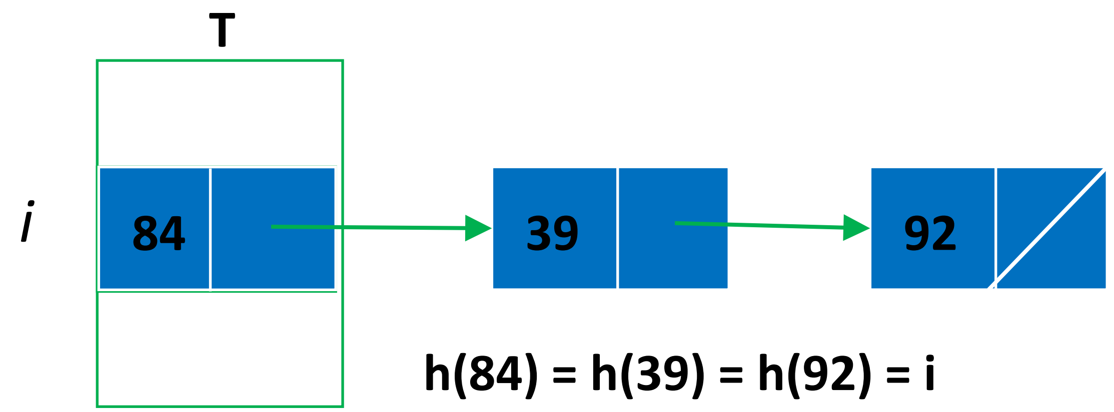
Closed Addressing: Separate Chaining

- When multiple keys are hashed into the same slot, these keys are inserted into to singly-linked list, which is known as a chain



Closed Addressing: Separate Chaining

- When multiple keys are hashed into the same slot index, these keys are inserted into to singly-linked list, which is known as a chain



- During searching, the searched key with hash address i is compared with keys in linked list H[i] sequentially
- In closed address hashing, there will be α number of keys in each linked list on average.

Closed Addressing: Separate Chaining

Time complexity analysis to search a key in the worst-case:

- When all elements are hashed to the same slot
- A linked list contains all n keys
- Its unsuccessful search takes n key comparisons, $\Theta(n)$
- Its successful search, assuming the probability of searching for each item is $\frac{1}{n}$
  $\frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2} = \Theta(n)$
  - It is just like a sequential search

Closed Address Hashing: Separate Chaining

Time complexity analysis to search a key in the average-case:

- All keys are equally likely hashed into h slots, each slot has $\frac{n}{h}$ keyes (i.e., α)
- Its unsuccessful search takes $\frac{n}{h}$ key comparisons, $\Theta(\alpha)$
- Its successful search: $\frac{1}{n/h}\sum_{i=1}^{n/h} i = \Theta(\alpha)$
  - If α is constant (n is proportional to h), then $\Theta(1)$
  - Searching takes constant time averagely

Open Addressing

- Each slot keeps one key
- α cannot be greater than 1
- When collision occurs, probe is required for the alternate slot

1. Linear Probing: probe the next slot

$H(k, i) = (H'(k) + i) \bmod h$ where $i \in [0, h-1]$

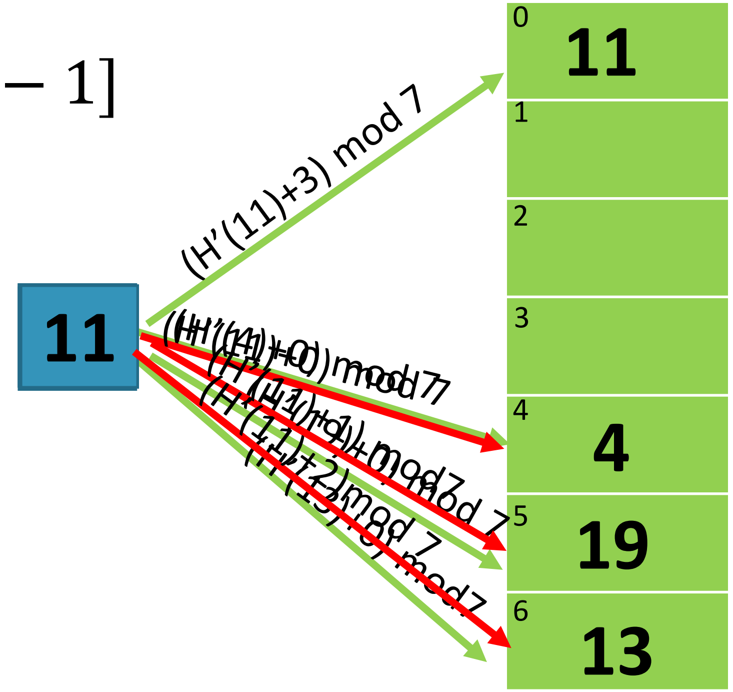
- $H'(k)$ is called auxiliary hash function

e.g., $H(k, i) = (H'(k) + i) \bmod 7$

$H'(k) = k \bmod 7, k \in \{4, 13, 19, 11\}$

Primary clustering:

- A long runs of occupied slots
- Average search time is increased



Open Addressing

2. Quadratic Probing

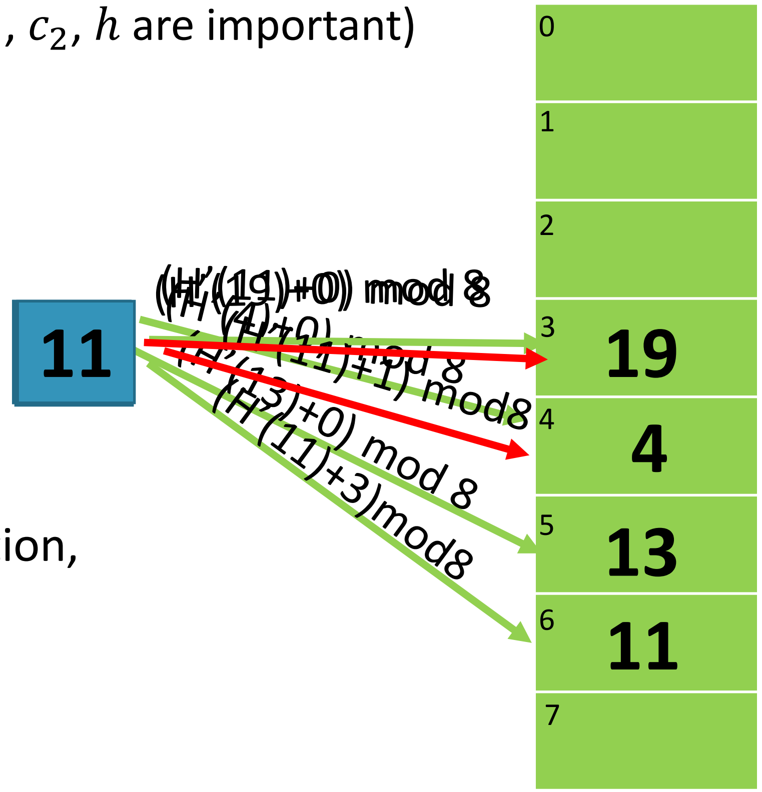
$H(k,i) = (H'(k) + c_1 i + c_2 i^2) \bmod h$, where c_1 and c_2 are constants, $c_2 \neq 0, i \in [0, h-1]$

- $H'(k)$ is auxiliary hash function
- May not all hash table slots be on the probe sequence (selection of c_1, c_2, h are important)
- For $h = 2^m$, a good choice for the constants are $c_1 = c_2 = \frac{1}{2}$

e.g., $H(k,i) = \left(H'(k) + \frac{1}{2}i + \frac{1}{2}i^2\right) \bmod 8$

$H'(k) = k \bmod 8, k \in \{4, 13, 19, 11\}$

i	$(\frac{1}{2}i + \frac{1}{2}i^2)$mod 8
1	1
2	3
3	6
4	2
5	7
6	5
7	4



- Secondary Clustering: if two keys have the same initial probe position, their probe sequences will be the same.
  - Inserting k=3 in the previous example.

Open Addressing

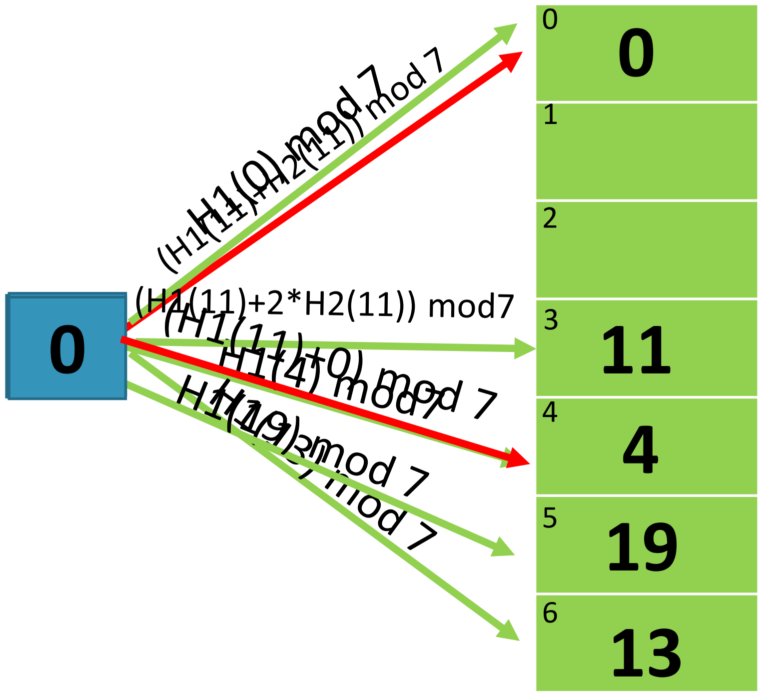
3. Double Hashing: a random probing method

$$H(k,i) = (H_1(k) + iH_2(k)) \bmod h, \text{ where } i \in [0, h-1]$$

- $H_1(k)$ and $H_2(k)$ are auxiliary hash functions
- The hash table size h should be a prime number

eg. $H(k,i) = (H_1(k) + iH_2(k)) \bmod 7$

$$H_1\ (k) = k \bmod 7$$
$$H_2(k) = k \bmod 3 + 1$$
$$k \in \{0, 4, 13, 19, 11\}$$



Time Complexity

Linear Probing

- Successful Search: $\frac{1}{2}(1 + \frac{1}{1-\alpha})$
- Unsuccessful Search: $\frac{1}{2}(1 + (\frac{1}{1-\alpha})^2)$

Double Hashing

- Successful Search: $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$
- Unsuccessful Search: $\frac{1}{1-\alpha}$

*Proof can be found in The Art of Computer Programming by Knuth Donald (1973)

Delete A Key Under Open Addressing

- Leave the deleted key in the table
- Make a marker indicating that it is deleted
- Overwrite it when a new key is inserted to the slot
- May need to do a “garbage collection” when a large number of deletions are done
  - To improve the search time

Rehashing: Expanding the Hash Table

- As α increases, the time complexity also increases

Solution:

- Increase the size of hash table
- Rehash all keys into new larger hash table

Summary

- Hash table: A typical space and time trade-off in algorithms
- Hash functions: Modulo Arithmetic, Folding, Mid-square, Multiplicative Congruential Method
- Collision
  - Closed Addressing: Separate Chaining
  - Open Addressing
    - Linear Probing
    - Quadratic Probing
    - Double Hashing
- Delete keys
  - Marker
  - Garbage collection
- Rehashing all the keys when hash table size is increaed