DATA STRUCTURE ALGORITHMS AND APPLICATIONS (CT- 159)

Lecture – 14
Hahsing- Hash Function, Hash Collision Technique, Rehashing

Instructor: Engr. Nasr Kamal

Department of Computer Science and Information Technology

Introduction to Hashing

Suppose that we want to store 10,000 students records (each with a 5-digit ID) in a given container.

- A linked list implementation would take **O(n)** time.
- A height balanced tree would give **O(log n)** access time.
- Using an array of size 100,000 would give **O(1)** access time but will lead to a lot of space wastage.

Is there some way that we could get **O(1)** access without wasting a lot of space?

The answer is hashing.

What is Hashing?

Hashing is a process used to store and retrieve data efficiently. It maps data (keys) to specific locations in a hash table using a mathematical function called a hash function.

Purpose: Enables fast access to data, typically in constant time O(1), making it a common choice for implementing dictionaries, caches, and databases.

Hash Function

A hash function is a mathematical formula that takes input (key) and converts it into a fixed-size integer, which serves as an index for storing the data in a hash table.

- A key can be a number, a string, a record etc.
- The size of the set of keys, **|K|**, to be relatively very large.
- It is possible for different keys to hash to the same array location.
- This situation is called *collision* and the colliding keys are called *synonyms*.

Characteristics: A good hash function should:

- **Deterministic**: Produces the same hash for the same input.
- **Efficient**: Computes quickly.
- **Uniform Distribution**: Minimizes clustering by spreading keys evenly.
- **Minimizes Collisions**: Reduces occurrences where different keys map to the same index.

Example

For a hash table of size nnn, a simple hash function is:

$$\mathrm{hash}(key) = key \ \% \ n$$

If n=10, the key 27 maps to:

27 % 10=7

Example 1: Illustrating Hashing

- Use the function `f(r) = r.id % 13` to load the following records into an array of size 13.

Al-Otaibi, Ziyad	1.73	985926
Al-Turki, Musab Ahmad Bakeer	1.60	970876
Al-Saegh, Radha Mahdi	1.58	980962
Al-Shahrani, Adel Saad	1.80	986074
Al-Awami, Louai Adnan Muhammad	1.73	970728
Al-Amer, Yousuf Jauwad	1.66	994593
Al-Helal, Husain Ali AbdulMohsen	1.70996321	

Example 1: Introduction to Hashing (cont'd)

Name	ID	h(r) = id % 13
Al-Otaibi, Ziyad	985926	6
Al-Turki, Musab Ahmad Bakeer	970876	10
Al-Saegh, Radha Mahdi	980962	8
Al-Shahrani, Adel Saad	986074	11
Al-Awami, Louai Adnan Muhammad	970728	5
Al-Amer, Yousuf Jauwad	994593	2
Al-Helal, Husain Ali AbdulMohsen	996321	1

0	1	2	3	4	5	6	7	8	9	10	11	12
	Husain	Yousuf			Louai	Ziyad		Radha		Musab	Adel	

Hash Tables

- There are two types of Hash Tables: **Open-addressed Hash Tables** and **Separate-Chained Hash Tables**.

- An Open-addressed *Hash Table* is a one-dimensional array indexed by integer values that are computed by an index function called a *hash function*.
- A Separate-Chained *Hash Table* is a one-dimensional array of linked lists indexed by integer values that are computed by an index function called a *hash function*.
- Hash tables are sometimes referred to as ***scatter tables***.
- Typical hash table operations are:

  -Initialization. ***-Searching***

  -Deletion. ***-Insertion.***

Collision in Hashing

A collision occurs when two different keys produce the same hash value, leading them to be stored at the same index in the hash table.

Example: Using **hash(key)=key%10**:

- Key **27: 27%10=7**
- Key **17: 17%10=7**

Both keys map to index **7**, causing a collision.

Collision Hashing Technique

1. **Chaining**

- **Concept**: Store multiple values at the same index using a linked list or dynamic structure.
- **Advantages**:
  - Easy to implement.
  - Handles collisions efficiently even with high load factors.
- **Example**: For **hash(key)=key%10**
- keys **27** and **17** both map to index **7**

Index 7: [27 → 17]

Collision Hashing Technique

2. Open Addressing

Concept: All data is stored within the hash table. When a collision occurs, the algorithm searches for the next available slot.

Methods:

Linear Probing: Increment the index by 1 until an empty slot is found.

- Example: **hash(key)=key%10**
- **Key 27: 27%10=7**
- **Key 17: Collision at 7 → Try 7+1=8**

Quadratic Probing: Probe by adding squares **(1^2, 2^2, etc.)** to the initial index.

Double Hashing: Use a second hash function to determine the step size.

Collision Hashing Technique

3. Rehashing

Concept: Expands the hash table size and applies a new hash function when the table becomes too full (load factor exceeds a threshold).

Steps:

1. Create a new, larger hash table.
2. Recompute the hash for all existing keys using the new hash function.
3. Transfer data to the new table.

Example: Original table size = **5**, new size = **10**.
Key **27,** originally at **27%5=2,** moves to **27%10=7.**

Collision Hashing Technique

4. Double Hashing

Concept: double hashing uses a secondary hash function to calculate the step size for probing. This ensures that collisions are resolved in a way that reduces clustering and distributes entries more uniformly.

Collision Hashing Technique

4. Double Hashing

Steps:

i. Primary Hash Function:
Maps the key k to an initial index in the hash table.
$h1(k)=k \% TableSize$

ii. Secondary Hash Function:
Calculates the step size for probing. The secondary hash function must never return 0 to ensure progress in probing.
$h2(k)=1+(k \% (TableSize-1))$

Collision Hashing Technique

4. Double Hashing

iii. Probing Formula:

To find an open slot after a collision, the position is calculated as:

$h(k,i)=(h1(k)+i \cdot h2(k)) \% TableSize$

where i is the probe number (starts at 0 and increments for each collision).

Summary

- **Hashing**: Efficient data retrieval technique.
- **Hash Function**: Maps keys to table indices.
- **Collisions**: Handled using techniques like chaining and open addressing.
- **Rehashing**: Expands the hash table to maintain performance.