DATA STRUCTURE

Chapter 9: Hashing

Topics: 9.1 Hash Functions | 9.2 Deleting Items from Hash Tables

---

Introduction to Hashing

Hashing is a technique used to store and retrieve data in almost constant time. Instead of searching a list one element after another (linear search, O(n)) or repeatedly dividing it (binary search, O(log n)), hashing uses a mathematical function to calculate directly the position at which a record should be stored. The data structure that stores records in this way is called a **hash table**.

A hash table is an array of fixed size **m**. Each position of the array is called a **slot** or **bucket**. When a record with key **k** has to be stored, a **hash function h** converts the key into an array index h(k), and the record is placed at that index. The same function is used later to search or delete the record, which makes these operations very fast on average.

Basic Terminology

- **Key (k):** the value used to identify a record, e.g. roll number, employee ID, name.
- **Hash function h(k):** a function that maps a key to an index in the range 0 to m − 1.
- **Hash value / hash address:** the index returned by the hash function.
- **Hash table:** an array of m slots that stores the records.
- **Collision:** the situation in which two different keys give the same hash value, i.e. h(k1) = h(k2) with k1 ≠ k2.
- **Load factor (α):** α = n / m, where n is the number of stored records and m is the table size. It tells how full the table is.
- **Synonyms:** keys that hash to the same address.

9.1 Hash Functions

A **hash function** is a function that takes a key as input and returns an integer index within the size of the hash table. Since the number of possible keys is usually much larger than the number of slots, it is not possible to avoid collisions completely, but a good hash function keeps them as few as possible.

Characteristics of a Good Hash Function

- It should be **easy and fast to compute**, ideally in O(1) time.
- It should distribute keys **uniformly** over all the slots of the table.
- It should **minimise collisions**.
- It should be **deterministic**, i.e. the same key must always produce the same hash value.
- It should use **all the parts of the key**, not only a few digits or characters.
- It should always return a value in the range **0 to m − 1**.

Types of Hash Functions

1. Division Method (Modulo Method)

This is the simplest and most commonly used method. The key is divided by the table size m and the remainder is taken as the hash value.

```
h(k) = k mod m
```

- The value of **m should be a prime number** that is not close to a power of 2 or a power of 10. This gives a more even spread of keys.
- If m is a power of 2 (say 2^p), only the lowest p bits of the key are used, which gives poor distribution.
- **Example:** m = 10; h(25) = 5, h(37) = 7, h(42) = 2. For m = 97, h(1234) = 1234 mod 97 = 70.
- **Advantage:** very fast, needs only one division. **Disadvantage:** the result depends heavily on the choice of m.

2. Multiplication Method

The key is multiplied by a constant A (0 < A < 1). The fractional part of the product is then multiplied by m and the floor of the result is taken.

```
h(k) = floor( m × (k·A mod 1) )
```

- Knuth suggested A ≈ 0.6180339887 (the golden ratio conjugate, (√5 − 1) / 2).
- The value of m is **not critical** in this method, so m can be a power of 2.
- **Example:** k = 123, m = 100, A = 0.6180339887. k·A = 76.01818, fractional part = 0.01818, so h(k) = floor(100 × 0.01818) = **1**.

3. Mid-Square Method

The key is squared and a fixed number of digits from the **middle** of the square are extracted as the hash value. The middle digits depend on all the digits of the key, so the distribution is fairly good.

- **Example:** k = 4567, k^2 = 20857489. The middle two digits are 57, so h(k) = **57** for a table of size 100.
- The same positions must be chosen for every key.
- **Disadvantage:** the square of a large key can overflow the integer size of the machine.

4. Folding Method

The key is divided into several parts, each having the same number of digits as the required address (the last part may be shorter). The parts are then added, and the carry, if any, is ignored.

- **Shift folding:** the parts are simply added. For k = 123456789 with 3-digit parts: 123 + 456 + 789 = 1368, ignoring the carry gives **368**.
- **Boundary folding:** alternate parts are reversed before adding. 123 + 654 + 789 = 1566, ignoring the carry gives **566**.
- It is useful when the keys are long, such as phone numbers or ID numbers.

5. Digit Extraction (Digit Selection) Method

Some selected digits are extracted from the key and joined to form the address. The digits chosen are those which are expected to be most evenly distributed.

- **Example:** k = 345678, extract the 2nd, 4th and 6th digits (4, 6, 8), so h(k) = **468**.
- **Disadvantage:** it does not use the whole key, so it works well only when the key pattern is known in advance.

6. Hash Function for Strings

When the key is a string, each character is converted to its ASCII value and combined using a polynomial so that the position of the character also matters.

```
h(s) = ( s[0]·b^(L-1) + s[1]·b^(L-2) + ... + s[L-1] ) mod m
```

- Here L is the length of the string and b is a small prime base such as 31 or 33.
- Simply adding the ASCII values is a poor choice because the strings "abc" and "cba" would give the same value.

7. Universal Hashing

In universal hashing a hash function is chosen **randomly** at run time from a family of functions, for example h(k) = ((a·k + b) mod p) mod m, where p is a prime larger than every key. Because the function is random, no fixed set of keys can be made to always produce many collisions.

Collision Resolution Techniques

Whenever a collision occurs, the record must still be stored somewhere. There are two main families of techniques.

A. Separate Chaining (Open Hashing)

- Every slot of the table holds a **linked list**. All keys that hash to the same slot are added to that list.
- The table can hold more records than the number of slots, so α can be greater than 1.
- Average search time is **O(1 + α)**.
- **Disadvantage:** extra memory is needed for the pointers.

B. Open Addressing (Closed Hashing)

All records are stored inside the table itself. If the slot h(k) is occupied, the other slots are examined one by one according to a **probe sequence** until an empty slot is found. Here α can never exceed 1.

- **Linear probing:** h(k, i) = (h(k) + i) mod m. Simple, but it causes **primary clustering**, i.e. long runs of occupied slots.
- **Quadratic probing:** h(k, i) = (h(k) + c1·i + c2·i²) mod m, for example (h(k) + i²) mod m. It reduces primary clustering but causes **secondary clustering**, because keys with the same initial position follow the same probe sequence.

- **Double hashing:** h(k, i) = (h1(k) + i·h2(k)) mod m, with h2(k) = R − (k mod R), where R is a prime smaller than m. The step size depends on the key, so clustering is the least. It is the best of the three.

Example (linear probing): table size m = 10, h(k) = k mod 10. Insert 23, 43, 13, 33. All of them hash to 3. 23 goes to slot 3, 43 to slot 4, 13 to slot 5, and 33 to slot 6.

Load Factor and Rehashing

- As the load factor increases, collisions increase and the performance drops. In open addressing it is usually kept below **0.5 to 0.7**.
- **Rehashing** means creating a larger table (usually about twice the size, and prime), calculating a new hash value for every existing key and inserting it in the new table.
- It is triggered when the load factor crosses a chosen threshold.

Advantages, Disadvantages and Applications

- **Advantages:** very fast insertion, deletion and search (average O(1)); simple to implement.
- **Disadvantages:** worst case O(n) when many keys collide; records are not kept in sorted order; needs extra space; performance depends on the hash function.
- **Applications:** compiler symbol tables, database indexing, caches, password storage (cryptographic hashes), dictionaries and sets, spell checkers, and finding duplicates.

9.2 Deleting Items from Hash Tables

Deletion means removing a record with a given key from the hash table. To delete a key, the key is first **searched** using its hash function, and then removed. How the removal is done depends on the collision resolution technique used, and it is much more difficult in open addressing than in chaining.

Deletion in Separate Chaining

Deletion is simple because each slot holds an independent linked list.

1. Compute the index i = h(k).
2. Traverse the linked list at slot i to find the node containing the key.
3. If the key is found, unlink the node from the list (adjust the pointer of the previous node) and free it.
4. If the key is not found, report that the key is not present.

- The other keys of the table are not affected in any way.
- Average time is **O(1 + α)**; worst case is O(n) when all keys are in one list.

Deletion in Open Addressing

In open addressing, a search for a key follows its probe sequence and **stops as soon as an empty slot is met**, because an empty slot means the key cannot be further along. Because of

this, simply making the slot of a deleted key empty is **wrong**, as it breaks the probe sequence of other keys that were inserted after it.

Why Simple Deletion Fails: Example

Let m = 10, h(k) = k mod 10 and linear probing. Insert 15, 25 and 35. All hash to slot 5, so 15 is placed in slot 5, 25 in slot 6 and 35 in slot 7.

- Suppose 25 is deleted and slot 6 is made empty.
- Now search for 35: start at slot 5 (holds 15, not a match), then slot 6 is **empty**, so the search stops and wrongly reports that 35 is **not found**, although 35 is in slot 7.

Solution 1: Lazy Deletion using Tombstones

Each slot is given one of three states: **EMPTY**, **OCCUPIED** or **DELETED**. A slot marked DELETED is called a **tombstone**. Instead of emptying the slot, the deleted record is only marked as DELETED.

- **Search:** on meeting a DELETED slot, the search does **not stop**; it continues along the probe sequence. It stops only at an EMPTY slot or when the key is found.
- **Insert:** the first DELETED slot met can be reused to store the new key, but only after confirming that the key is not already present further along the probe sequence.
- **Delete:** search for the key and, if found, mark its slot as DELETED.

In the example above, after deleting 25 the slot 6 is marked DELETED. Searching 35 now goes slot 5, slot 6 (DELETED, continue), slot 7 (found). The search is correct.

Algorithm for Deletion with Tombstones

```
DELETE(T, k):
  i = 0
  repeat
    j = (h(k) + i) mod m
    if T[j] is EMPTY:  return "not found"
    if T[j] is OCCUPIED and T[j].key == k:
        mark T[j] as DELETED
        return "deleted"
    i = i + 1
  until i == m
  return "not found"
```

Drawbacks of Tombstones

- Tombstones are never truly removed, so after many deletions the table may contain **few real records but very few EMPTY slots**.
- Searches become slow because they must continue past all the DELETED slots, and an unsuccessful search may scan the whole table.
- Tombstones count towards the load factor for the purpose of performance.
- **Remedy:** rehash the whole table after a number of deletions. This discards all the tombstones and restores the performance.

Solution 2: Rehashing the Cluster (Backward Shift Deletion)

This method is used with linear probing and does not need tombstones. After the key is removed, the slots that follow it in the same cluster are examined one by one until an empty slot is reached. Any record that can legally move into the hole is shifted back to fill it, or all the records of the cluster are removed and reinserted.

- Deletion is more expensive than the tombstone method, but searches stay fast because no dead slots are left.
- In the earlier example, after deleting 25 (slot 6), record 35 from slot 7 is moved to slot 6, and slot 7 becomes empty. A search for 35 then finds it in slot 6.
- It is harder to apply for quadratic probing and double hashing, so tombstones are preferred there.

Comparison of Deletion Methods

- **Separate chaining:** unlink node from list. Easy, no side effects, average O(1 + α).
- **Open addressing with tombstones:** mark slot DELETED. Simple and fast delete, but performance decays until rehashing.
- **Open addressing with cluster rehash:** shift records back. No dead slots, but delete is slower and more complex.

Important Points to Remember

- Never make a slot EMPTY when deleting from open addressing, unless the cluster is repaired.
- Search continues past DELETED slots but stops at EMPTY slots.
- Deletion in chaining is the same as deletion in a singly linked list.
- Time complexity of deletion is O(1) on average and O(n) in the worst case for all techniques.

Multiple Choice Questions (15)

Choose the correct option for each question. The answer key is given at the end of this section.

Q1. A hash function is used to

(a) Sort the keys
(b) Map a key to an index of the hash table
(c) Encrypt the data
(d) Compress the file

Q2. The average time for searching in a hash table is

(a) O(n)
(b) O(log n)
(c) O(1)
(d) O(n log n)

Q3. In the division method with m = 97, the value of h(1234) is

(a) 70
(b) 68
(c) 72
(d) 74

Q4. In the division method, the table size m should preferably be

(a) A power of 2
(b) An even number
(c) A power of 10
(d) A prime number not close to a power of 2

Q5. In the mid-square method, for the key 25 (square = 625) and a table of size 10, the hash value using the middle digit is

(a) 6
(b) 5
(c) 2
(d) 0

Q6. In shift folding, the key 123456 is split into 2-digit parts (12, 34, 56) and added. Ignoring the carry for a table of size 100, the address is

(a) 12
(b) 2 (i.e. 02)
(c) 102
(d) 56

Q7. When two different keys give the same hash address, it is called a

(a) Collision
(b) Overflow
(c) Underflow
(d) Traversal

Q8. The load factor of a hash table with n records and m slots is

(a) m / n
(b) n × m

(c) $m - n$
(d) n / m

Q9. Primary clustering occurs in

(a) Separate chaining
(b) Linear probing
(c) Double hashing
(d) Perfect hashing

Q10. Which open addressing technique gives the least clustering?

(a) Linear probing
(b) Quadratic probing
(c) Double hashing
(d) All give the same

Q11. In open addressing, making the slot empty when a key is deleted may cause

(a) Faster searching
(b) Less memory use
(c) Table overflow
(d) Unsuccessful search of other keys that are still present

Q12. A slot that is marked as deleted in lazy deletion is called a

(a) Tombstone
(b) Null pointer
(c) Root
(d) Bucket

Q13. The average time for deleting a key in separate chaining is

(a) $O(m)$
(b) $O(n^2)$
(c) $O(1 + \alpha)$
(d) $O(\log n)$

Q14. While searching in a table with tombstones, on reaching a DELETED slot the algorithm

(a) Stops and reports not found
(b) Continues probing the next slot
(c) Reports that the key is found
(d) Rehashes the table

Q15. In open addressing the load factor can never exceed

(a) 0.5
(b) 2
(c) 10
(d) 1

Answer Key

Q1: (b) Q2: (c) Q3: (a) Q4: (d) Q5: (c)
Q6: (b) Q7: (a) Q8: (d) Q9: (b) Q10: (c)
Q11: (d) Q12: (a) Q13: (c) Q14: (b) Q15: (d)

Broad Questions (10)

Q1. What is hashing? Explain the terms hash table, hash function, collision and load factor with suitable examples.

Q2. What are the characteristics of a good hash function? Explain why the table size should be a prime number in the division method.

Q3. Explain the division method and the multiplication method of hashing with examples. Compare the two.

Q4. Explain the mid-square method and the folding method (shift and boundary folding) with numerical examples.

Q5. What is a collision? Explain any two collision resolution techniques with examples.

Q6. Explain linear probing, quadratic probing and double hashing. Discuss primary and secondary clustering.

Q7. What is rehashing? When is it needed and how is it carried out?

Q8. Explain how an item is deleted from a hash table that uses separate chaining. Write the steps and state the time complexity.

Q9. Why does simply emptying a slot fail during deletion in open addressing? Explain with an example and describe how tombstones (lazy deletion) solve the problem.

Q10. Write an algorithm for deletion in a hash table with open addressing using tombstones. Discuss its drawbacks and explain the alternative method of rehashing the cluster.