MIT World Peace University
Advanced Data Structures

Assignment 8

Naman Soni Roll No. 10

Contents

1 Problem Statement

Implement Direct access file using hashing (linear probing with and without replacement) perform following operations on it a. Create Database b. Display Database c. Add a record d. Search a record e. Modify a record

2 Objectives

1. To study hashing techniques
2. To implement different hashing techniques
3. To study and implement linear probing with amp; without replacement
4. To study how hashing can be used to model real world problems

3 Theory

3.1 *What is Hashing? Compare hashing with other searching techniques.*

Hashing is a technique of generating a unique code, called a hash, from a given input data, such as a string or a file. The hash function takes the input and returns a fixed-size hash value, which can be used to identify the data uniquely. The process of generating a hash is fast, and the resulting hash is typically much smaller than the original data.

Hashing is commonly used in computer science for searching and indexing. A hash table is a data structure that stores key-value pairs, where the keys are hashed to map to an index in an array. The value associated with the key is stored at that index. When searching for a key, the hash function is used to map the key to an index in the array, and then the value is retrieved from that index.

Compared to other searching techniques, hashing has several advantages:

- **Efficiency:** Hashing has a constant time complexity of O(1) for both insertion and retrieval. This means that the time required to search for a value is independent of the size of the data set.
- **Memory usage:** Hashing uses less memory than other search techniques such as binary search trees, since it only stores the hash values instead of the original data.
- **Collision resolution:**Hashing allows for collision resolution, which means that when two keys hash to the same index, the hash table can still store both keys and their associated values.

However, hashing also has some limitations:

- **Collisions:** If the hash function is not well-designed, it can lead to collisions, where two different keys produce the same hash value. This can result in slower performance and require more memory.
- **Unordered:** Hashing does not maintain any order between the keys. This means that if you need to retrieve the keys in a specific order, hashing may not be the best option.
- **Limited search:** Hashing is only useful for exact match searching. If you need to search for values that are close to a given value, hashing may not be the best technique.

3.2 *Write different hash functions*

Here are some examples of hash functions:

1. **Simple Hash Function:** This is a very basic hash function that adds up the ASCII values of each character in the string and returns the sum modulo some value, such as the size of the hash table.

```python
      def simple_hash(s, table_size):
    sum = 0
    for i in range(len(s)):
        sum += ord(s[i])
    return sum % table_size
```

2. **Polynomial Rolling Hash Function:** This hash function is commonly used in string search algorithms. It generates a hash value by treating the string as a polynomial with a base of a prime number, such as 31, and calculates the hash value using rolling technique.

```python
      def rolling_hash(s, table_size):
    p = 31  # prime number
    hash_value = 0
    power_p = 1
    for i in range(len(s)):
        hash_value = (hash_value + (ord(s[i]) - ord('a') + 1) * power_p) % table_size
        power_p = (power_p * p) % table_size
    return hash_value
```

3. **SHA-256 Hash Function:** This is a cryptographic hash function that produces a fixed-size output of 256 bits. It is widely used in security applications, such as digital signatures and message authentication.

```python
      import hashlib

def sha256_hash(s):
    return hashlib.sha256(s.encode()).hexdigest()
```

4. **MD5 Hash Function:** This is another cryptographic hash function that produces a 128-bit output. It is widely used in checksums and data integrity verification.

```python
      import hashlib

def md5_hash(s):
    return hashlib.md5(s.encode()).hexdigest()
```

3.3 *Explain hash collision resolution techniques.*

Hash collision occurs when two or more keys map to the same hash value or index in a hash table. To deal with hash collisions, there are various techniques that can be used for resolving collisions. Here are some of the most commonly used hash collision resolution techniques:

1. **Chaining:** In this technique, each slot in the hash table contains a linked list of elements that have the same hash value. When a collision occurs, the new element is simply added to the linked list at the appropriate slot. When searching for an element, the hash function is used to find the appropriate slot, and then a linear search is performed on the linked list at that slot. This technique is simple and works well when the hash table is large and the number of collisions is small.

2. **Open addressing:** In this technique, when a collision occurs, the algorithm looks for the next available slot in the hash table and places the element there. There are different methods of finding the next slot, such as linear probing (checking the next slot), quadratic probing (checking the next slot plus a quadratic function of the attempt number), and double hashing (using a second hash function to find the next slot). When searching for an element, the hash function is used to find the initial slot, and then the probing technique is used to find the actual slot where the element is stored. Open addressing works well when the hash table is small and the number of collisions is relatively high.

3. **Robin Hood hashing:** This is a variation of open addressing where elements are placed in a sorted order based on their distance from their original slot. When a new element collides with an existing element, the algorithm compares the distances of the elements from their original slots. If the new element is closer to its original slot, it replaces the existing element, and the existing element is moved to the new slot. If the existing element is closer, then the algorithm continues to probe for the next available slot. This technique can reduce the average probe length and improve cache performance.

4. **Cuckoo hashing:** In this technique, each element is assigned to two hash functions and two hash tables. When a collision occurs in the first hash table, the element is moved to the second hash table using its second hash function. If a collision occurs in the second hash table, the element is moved back to the first hash table using its first hash function. This process continues until either an empty slot is found or a maximum number of retries is reached. Cuckoo hashing has a constant time complexity for insertion, deletion, and lookup, but it requires more memory than other techniques.

4 Implementation

4.1 *Platform*

- 64-bit Mac OS
- Open Source C++ Programming tool like Visual Studio Code

subsection*Test Conditions*

1. Input min 10 elements.
2. Display collision with replacement and without replacement.

5 Conclusion

Thus, we have implemented linear probing with and without replacement.

6 FAQ's

6.1 *Write different types of hash functions.*

Ans. There are several types of hash functions that can be used depending on the specific use case. Here are some of the most commonly used types:

- **Cryptographic hash functions:** These are hash functions that are designed to be very difficult to reverse or compute collisions. They are commonly used for secure communication, digital signatures, and password storage. Examples of cryptographic hash functions include SHA-256, SHA-3, and MD5.
- **Non-cryptographic hash functions:** These are hash functions that are designed for speed and efficiency rather than security. They are commonly used in data structures such as hash tables, bloom filters, and hash-based message authentication codes (HMAC). Examples of non-cryptographic hash functions include MurmurHash, Jenkins Hash, and FNV Hash.

- **Perfect hash functions:** These are hash functions that guarantee no collisions for a given set of keys. They are used when the keys are known in advance and fixed, such as in a compiler symbol table or a command-line interface. Examples of perfect hash functions include Pearson Hash, Minimal Perfect Hashing, and Cuckoo Hashing.
- **Universal hash functions:** These are hash functions that are designed to uniformly distribute keys across the hash table, reducing the likelihood of collisions. They are commonly used in open addressing and chaining collision resolution techniques. Examples of universal hash functions include Tabulation Hashing, Multiplicative Hashing, and Carter and Wegman's Hashing.
- **Pseudo-random permutation functions:** These are hash functions that are designed to behave like a random permutation of the input keys. They are commonly used in block ciphers and stream ciphers. Examples of pseudo-random permutation functions include AES, Blowfish, and ChaCha

6.2 *Explain chaining with amp; without replacement with example.*

Ans. Chaining is a collision resolution technique used in hash tables and hash maps. When two or more keys hash to the same index, chaining stores them in a linked list or another data structure at that index. There are two types of chaining: with replacement and without replacement.

Chaining with replacement: In chaining with replacement, when a collision occurs, the new key-value pair is added to the linked list at the same index as the existing key-value pairs. The linked list is updated to include the new key-value pair, and the hash table or hash map size remains constant. Here is an example of chaining with replacement:

Suppose we have a hash table with 5 slots, and we want to store the following key-value pairs:

- key: 10, value: "apple"
- key: 36, value: "orange"
- key: 49, value: "grape"
- key: 62, value: "pineapple"
- key: 75, value: "mango"

We use a hash function that maps keys to indices in the range [0,4]. Here is the result of the hashing:

- key: 10, index: 0
- key: 23, index: 3
- key: 36, index: 1
- key: 49, index: 4
- key: 62, index: 2
- key: 75, index: 0 (collision with key 10)

The resulting hash table with chaining with replacement would look like this:

Index	Linked List
0	(10, “apple”) -¿ (75, “mango”)
1	(36, “orange”)
2	(62, “pineapple”)
3	(23, “banana”)
4	(49, “grape”)

In this example, key 75 collides with key 10, which is already stored in the linked list at index 0. The new key-value pair (75, ”mango”) is added to the linked list, and the hash table size remains constant.

Chaining without replacement: In chaining without replacement, when a collision occurs, the new key-value pair is added to the next available slot in the hash table or hash map. If there are no available slots, the hash table or hash map is resized. Here is an example of chaining without replacement:

Suppose we have a hash table with 5 slots, and we want to store the same key-value pairs as in the previous example. Again, we use a hash function that maps keys to indices in the range [0,4]. Here is the result of the hashing:

- key: 10, index: 0
- key: 23, index: 3
- key: 36, index: 1
- key: 49, index: 4
- key: 62, index: 2
- key: 75, index: 0 (collision with key 10)

Since we are using chaining without replacement, the hash table needs to be resized to accommodate the new key-value pair. We double the size of the hash table to 10 slots, and the new hash values are:

- key: 10, index: 0
- key: 23, index: 3
- key: 36, index: 1
- key: 49, index: 4
- key: 62, index: 2
- key: 75, index: 5

The resulting hash table with chaining without replacement would look like this: In this example, the new

Index	**Key-Value Pair**
0	(10, “apple”)
1	(36, “orange”)
2	(62, “pineapple”)
3	(23, “banana”)
4	(49, “grape”)
5	(75, “mango”)
6	-
7	-
8	-
9	-

key-value pair (75, “mango”) was added to the next available slot in the resized hash table. The hash table now has twice as many slots as before, and the keys are more evenly distributed across the slots.

6.3 *Explain quadratic probing with example*

Ans. Quadratic probing is a collision resolution technique used in hash tables and hash maps. When two or more keys hash to the same index, quadratic probing stores the new key-value pair in the next available slot. The next available slot is determined by adding a quadratic function of the number of collisions to the original hash value. Here is an example of quadratic probing: