Hashing 1.2

Mayank Pratap Singh

Week 4

1 Hash Tables

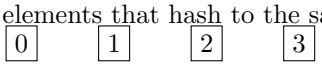
A hash table is a data structure that implements an associative array, a structure that can map keys to values. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

2 Handling Collisions

Collisions occur when two keys hash to the same index. There are several strategies to handle collisions:

2.1 Chaining

Chaining uses linked lists to handle collisions. Each bucket contains a list, and all elements that hash to the same bucket are inserted into that list.



2.2 Open Addressing

Open addressing tries to find another open slot within the hash table when a collision occurs. The primary methods are:

2.2.1 Linear Probing

Linear probing resolves collisions by checking the next slot sequentially until an empty slot is found.

Formula:

$$\mathrm{LP}(k, i) = (h(k) + i)\%m$$

where k is the key, i is the number of probes, and m is the size of the table.

Example: Values: 50, 75, 99, 20, 35, 88, 45, 23, 55, 67

Hash function:

$$h(k) = k\%10$$

Insert 50:

$$\mathrm{LP}(50, 0) = (0 + 0)\%10 = 0$$

0
50

1
45

2
20

3

4

5
75

6

7
35

8
88

9
99

2.2.2 Quadratic Probing

Quadratic probing uses a quadratic function to find the next slot.

Formula:

$$\mathrm{QP}(k, i) = (h(k) + c_1 i + c_2 i^2)\%m$$

where c_1 and c_2 are constants.

Example:

$$\text{QP}(20, 1) = (0 + 1 + 1)\%10 = 2$$

2.2.3 Double Hashing

Double hashing uses a second hash function to determine the probe sequence.

Formula:

$$\text{DH}(k, i) = (h_1(k) + i \cdot h_2(k))\%m$$

where h_1 and h_2 are hash functions.

Example:

$$h_1(k) = k\%10, \quad h_2(k) = 1 + (k\%9)$$

$$\text{DH}(20, 1) = (0 + 1 \cdot 5)\%10 = 5$$

3 Complexity Analysis

- **Insertion**: $O(1)$ in the average case, $O(m)$ in the worst case. - **Deletion**: $O(1)$ in the average case, $O(m)$ in the worst case. - **Search**: $O(1)$ in the average case, $O(m)$ in the worst case.

4 Applications of Hashing

Hashing is widely used in: - Databases for indexing - Caches - Password storage - Symbol tables in compilers

5 Example Implementation in Python

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [None] * size

    def hash_function(self, key):
        return key % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        while self.table[index] is not None:
            index = (index + 1) % self.size
        self.table[index] = (key, value)

    def search(self, key):
```

```
        index = self.hash_function(key)
        while self.table[index] is not None:
            if self.table[index][0] == key:
                return self.table[index][1]
            index = (index + 1) % self.size
        return None

    def delete(self, key):
        index = self.hash_function(key)
        while self.table[index] is not None:
            if self.table[index][0] == key:
                self.table[index] = None
                return True
            index = (index + 1) % self.size
        return False

# Example Usage
ht = HashTable(10)
ht.insert(50, 'A')
ht.insert(75, 'B')
print(ht.search(50))  # Output: 'A'
print(ht.search(75))  # Output: 'B'
ht.delete(50)
print(ht.search(50))  # Output: None
```