**Introduction to Algorithm Notes:**

Contents

[Chapter 11. Hash Table 2](#_Toc465587851)

[11. 1 Direct-address Table 2](#_Toc465587852)

[11.2 Hash Table 2](#_Toc465587853)

# Chapter 11. Hash Table

## 11. 1 Direct-address Table

Search, Insert, Delete all take O(1) time.

## 11.2 Hash Table

The downside of direct addressing is obvious: if the universe U is large, storing a table T of size jUj may be impractical, or even impossible, given the memory available on a typical computer.

hash function:

h: U -> {0,1, …, m-1}

problem: collision (different value will derive same key value from h(x.key))

solution: (1) chaining (2) open addressing

(1) Chaining

we place all the elements that hash to the same slot into the same linked list.

performance analysis:

n element, m slot of hash table. Worst case O(n) of search time if all n element has same key. Assume hash function (h()) is ideal. The search time is nh(k)+O(1)

***Theorem 11.1***

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time ‚(1+a), a=n/m, under the assumption of simple uniform hashing.

***Theorem 11.2***

In a hash table in which collisions are resolved by chaining, a successful search takes average-case time ‚(1+a), a=n/m under the assumption of simple uniform hashing.

What does this analysis mean? If the number of hash-table slots is at least proportional to the number of elements in the table, we have n = O(m) and, consequently, a = n/m = O(m)/m = O(1)

So the slot in hast tale bigger or more proportional to the number of elements, the performance is better.

## 1.3 Hash Function

A good hash function satisfies (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the m slots, independently of where any other key has hashed to.

Interpreting keys as natural numbers:

pt -> (112,116)

h(pt) = (112x128)+116 = 14452

(1) The division method

In the division method for creating hash functions, we map a key k into one of m slots by taking the remainder of k divided by m. That is, the hash function is

h(k) = k mode m

When using the division method, we usually avoid certain values of m. For example, m should not be a power of 2, since if m = 2p, then h.k/ is just the p lowest-order bits of k. Unless we know that all low-order p-bit patterns are equally likely, we are better off designing the hash function to depend on all the bits of the key. A prime not too close to an exact power of 2 is often a good choice for m.

(2) The multiplication method

The multiplication method for creating hash functions operates in two steps. First, we multiply the key k by a constant A in the range 0 < A < 1 and extract the fractional part of kA. Then, we multiply this value by m and take the floor of the result. In short, the hash function is

h(k) = floor ( m (kA mod 1) ) where kA mode 1 = kA – floor(kA)



(3) Universal hashing

Let H be a finite collection of hash functions that map a given universe U of keys into the range 0; 1; : : : ;m \_ 1. Such a collection is said to be universal if for each pair of distinct keys k; l in U, the number of hash functions h in H for which h(k) = h(l) is at most jHj=m. In other words, with a hash function randomly chosen from H, the chance of a collision between distinct keys k and l is no more than the chance 1/m of a collision if h(k) and h(l) were randomly and independently chosen from the set 0; 1; : : : ;m \_ 1.

m slots in hash table, choose prime number p (p>m)

Zp (0 … p-1) Zp\* (1…p-1)

choose a in Zp, bin Zp\*

hab(k) = ((ak+b) mod p) mod m

example: p=17, m=6, h3,4(8) = 5

## 11.4 Open addressing