Data Structures
Hash Table Effieciency and Other Types

Cain Susko

Queen's University
School of Computing

March 11, 2022

Hash Table Effieciency

this effieciency of a hash table os mainly dictated by the complexity of the `locate` and `insert` functions.
These complexities depends on the type of hash table (closed v. open), how full the table is, and how many collisions there are in the hash table.
In the worst case, all elements are hashed into the same bucket (list at index of 'bucket') as the search time is relative to the number of elements in the bucket. In parctice however, this does not happen as hast tables are carefully designed to avoid this.
Witin a hash table, every key k is equally likely to hash to the m buckets.

Open Hash Table Effieiency If you have a good hash function, there are N items distrubuted over M buckets. This means that the *load* of the hash table is:

$$\alpha = \frac{N}{M}$$

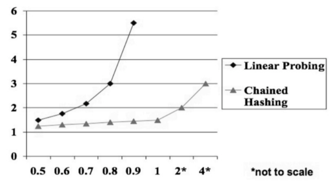
A successful search results in the list in the bucket being searched with the average number of comparisons being:

$$S_\alpha = 1 + \frac{N-1}{2M} = 1 + \frac{\alpha}{2}$$

a unsuccessful search will result in an average complexity of $\Theta(1+\alpha)$

Closed Hash Table Efficeiency within a closed hash table, when it is quite empty, the time complexity is nearly 1. when the hash table is nearly full, the time complexity can be nearly $O(N)$.

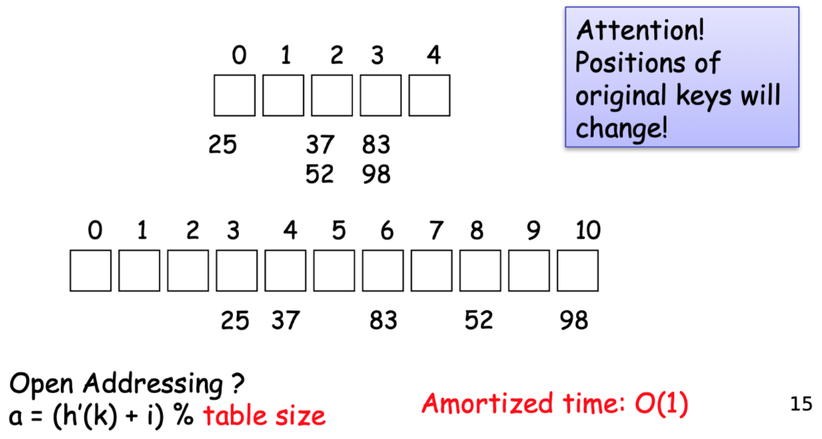
Linear Probing the average number of searches during a successful search as a function of the load factor α is:



there is a tradeoff as the larger the hash table, the faster it is. however that requires more memory, less memeory means slower hash table. in order to resolve both these issues we can implement rehasing

Re Hashing

for seperate chaining we can rehash the function $h_1(x) = m \mod 5$ to be $h_2(x) = m \mod 11$



Quadratic Probing Additionally, a draw on time is when linear probing checks many already-full buckets. To avoid this, we can use something called Quadratic Probing.

Quadratic probing:

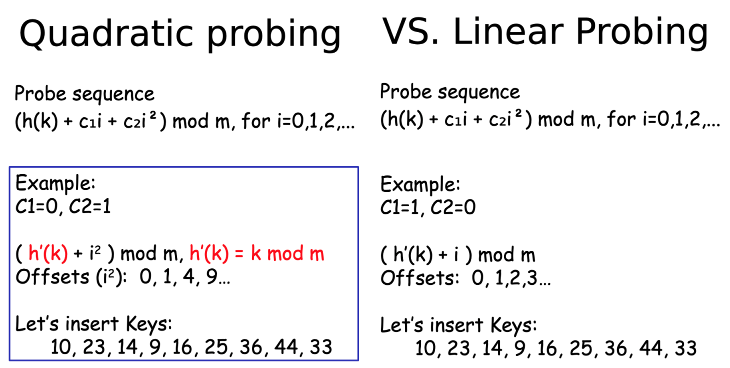
$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

Double hashing:

$$h(k, i) = (h_1(k) + i h_2(k)) \bmod m$$

Note: $c_2 \neq 0$

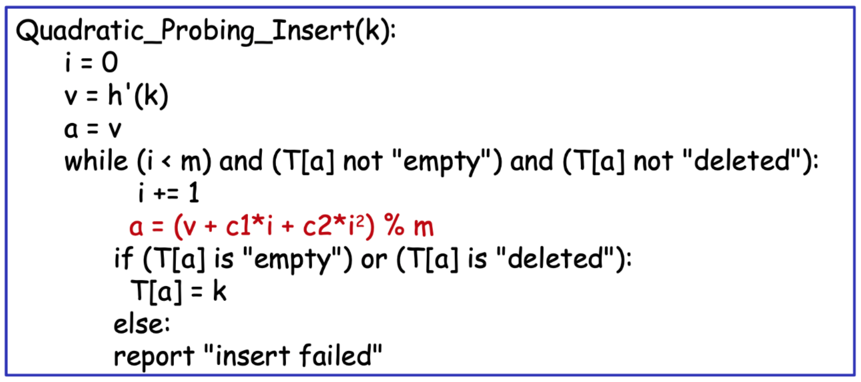
Examples



Quadratic probing VS. Linear Probing

Quadratic probing	Linear Probing
Probe sequence	Probe sequence
$(h(k) + c_1 i + c_2 i^2)$ mod m, for i=0,1,2,...	$(h(k) + c_1 i + c_2 i^2)$ mod m, for i=0,1,2,...
Example:	Example:
C1=0, C2=1	C1=1, C2=0
$(h'(k) + i^2)$ mod m, h'(k) = k mod m	$(h'(k) + i)$ mod m
Offsets (i^2): 0, 1, 4, 9...	Offsets: 0, 1,2,3...
Let's insert Keys: 10, 23, 14, 9, 16, 25, 36, 44, 33	Let's insert Keys: 10, 23, 14, 9, 16, 25, 36, 44, 33

An implementation of quadratic probing would be as follows:



```
Quadratic_Probing_Insert(k):
    i = 0
    v = h'(k)
    a = v
    while (i < m) and (T[a] not "empty") and (T[a] not "deleted"):
          i += 1
         a = (v + c1*i + c2*i²) % m
        if (T[a] is "empty") or (T[a] is "deleted"):
          T[a] = k
        else:
        report "insert failed"
```

Problems with Quadratic Probing There are some problems with Qudratic Probing however, mainly it is that the entire probing sequece is determined by the initial probe. A much larger problem with quadratic probing is that, if given the probe sequence for $h'(k) = 0$ is:

let $(h'(k) + i + i^2)$ mod 12, i.e., c1=c2=1, m=12

$$0, 2, 6, 0, 8, 6, 6, 0, 8, 6, \ldots$$

which will *never* caontain any odd numbers, so our table couls only be half full but the hash function would say it is full...

In order to avoid this, we must control the load factor and table size such that:

$$\alpha \leq 0.5$$
$$prime(N) = \texttt{true}$$

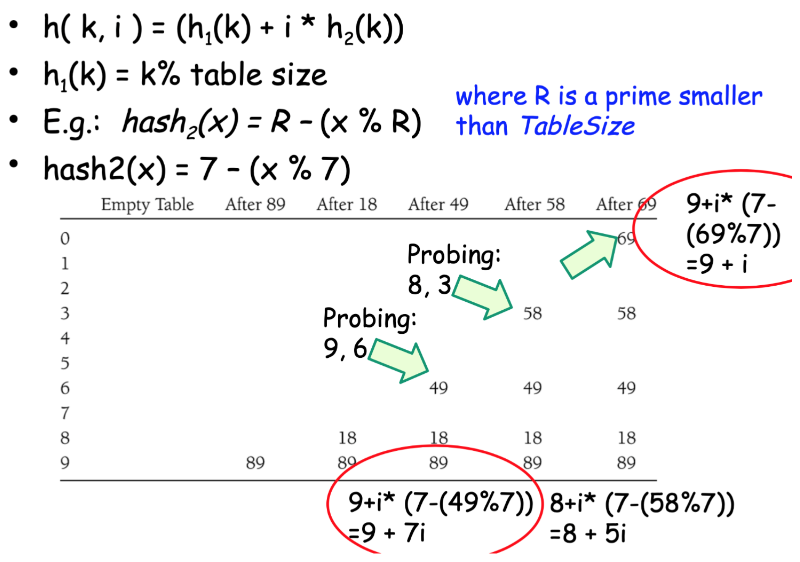
where α is the load factor and N is the number of items contained in the hash table. Another way to avoid this problem altogether is by using double hashing instead of quadratic probing.

Double Hashing

The idea of double hasing is to using a second hash function to spread out the search for an empty slot. double hashing is one of the most efficient collosion avoidance methodes to use. An Example equation is:

$$h(i, k) = (h_1(k) + i * h_2(k)) \mod m$$

due to the fact that $h_2(k)$ and m are relatively prime for all k, each probing sequence **contains all addresses!**



- $h(k, i) = (h_1(k) + i * h_2(k))$
- $h_1(k)$ = k% table size
- E.g.: $hash_2(x) = R - (x \% R)$ where R is a prime smaller than *TableSize*
- hash2(x) = 7 - (x % 7)

	Empty Table	After 89	After 18	After 49	After 58	After 69
0						69
1						
2						
3					58	58
4						
5						
6				49	49	49
7						
8			18	18	18	18
9		89	89	89	89	89

Probing: 9, 6
Probing: 8, 3
9+i* (7-(49%7)) =9 + 7i
8+i* (7-(58%7)) =8 + 5i
9+i* (7-(69%7)) =9 + i

within double factoring, the load factor for the successful and unsuccessful search are:

$$\frac{1}{1-\lambda} \qquad \frac{1}{\lambda}\ln\left(\frac{1}{1-\lambda}\right)$$