Hashing

Yu-Tai Ching
Department of Computer Science
National Chiao Tung University

- Dictionary problem, maintain a dictionary and support operations Get, $Insert$, and $Delete$.
- Using an array.
- Using a balance tree
- Hashing, all 3 operations can be done in $O(1)$ expected time.

Static Hashing

- In Static Hasing, dictionary pair stored in a table ht, called the *hash table*.
- b buckets in ht, $ht[0]$, $ht[1]$, ..., $ht[b-1]$.
- each bucket is capable of holding s dictionary pairs, i.e., there are s slots. If $s = 1$, a bucket can hold one pair.
- Hashing function h maps a key to the address of the pair stored in ht,
- $h(k)$ map k to an integer in the range 0 to $b-1$.
- $h(k)$ is the hash or home address of k.

Definition: The *key density* of a hash table is the ratio n/T, where n is the number of pairs in the table and T is the total number of possible keys.

- An example, keys are at most 6 characters (decimal digit or an upper case character) long, and first cahracter must be a letter.
- The number of possible keys is $T = \sum_{0 \leq i \leq 5} 26 \times 36^i > 1.6 \times 10^9$.
- We usually use a very small portion of these, thus key density n/T is very small.
- Thus number of buckets b is much less than T, and h could map different keys to the same bucket.

Definition: The *loading density* or **loading factor** of a hash table is $\alpha = n/(sb)$.

- The number of bucket, b, usually of the same magnitude as the number of keys (much less than T). Thus there are chances that k_1 and k_2 are mapped into the same bucket.
- k_1 and k_2 are synonyms w.r.t. h if $h(k_1) = h(k_2)$.
- Idea case will be that each pair stored in its home bucket.
- "Collision" home bucket of a new pair is not empty.
- "Overflow" home bucket for a new pair is full.

An Example

- A hash table, $b = 26$ buckets and $s = 2$.
- Assume there are $n = 10$ distinct keys, each key begins with a letter.
- Loading factor $\alpha = 10/52 = 0.19$.
- Hashing function h maps the begining letter A - Z to 0-25.
- Home buckets for GA, D, A, G, L, A2, A1, A3, A4 and E are 6, 3, 0, 6, 11, 0, 0, 0, 0, and 4.
- A, A1, A2, A3, and A4 are synonyms.
- figure 8.1

- What is the lower bound for search operation in the dictionary problem, if only comparison operation is allow.
- Search by static hashing, if no overflow occur, time required is to compute the hash function (nothing to do with n).
- if the hashing function is easy to compute (can be computed in $O(1)$ time), a search can be done in constant time.

- We would like to have the hash function easy to compute,
- Minimize collision, for a random input, the probability of $h(k) = i$ is $1/b$.
- A hash function satisfies this property called a “uniform hash function”.

Division

- The most widely used hash function,
- use the "modulo" operation, $h(k) = k\%D$.
- So the bucket size is D, from 0 to $D - 1$.
- If D is divisible by 2, odd keys go to odd bucket, even keys go to even bucket.
- There are bias for the case that D is divisible by small prime number.
- In practice, we choose D that has no prime factor smaller than 20.

Mid-Square

- Given a key, k, compute the home bucket by
  1. first taking sqaure of k,
  2. using an appropriate number of bits from the middle of the square
- bucket size depends on the number of bits taken
- if r bits are taken, bucket size is $2^r - 1$.

Folding

Givne a key k, the home bucket is computed by

- partition k into several parts.
- the partitions are then added to get the address.
- another approach: folding at the boundary,
- If $k = 12320324111220$, partition into 3 decimal digits long, we have $P_1 = 123$, $P_2 = 203$, $P_3 = 241$, $P_4 = 112$, $P_5 = 20$,
  1. $h(k) = \sum P_i = 123 + 203 + 241 + 112 + 20 = 699$.
  2. Or $h(k) = 123 + 302 + 241 + 211 + 20 = 897$.

Overflow Handling

- Two popular ways to handle overflow
  1. Open addressing,
  2. Chaining
- Chaining, a bucket is a pointer to a list of keys having same home bucket. If collide, insert the pair into the list.

Open Addressing

- Linear probing, searching for the bucket $(h(k) + i)\%b$ to find the next available space, a problem- tend to be clustered.
- Quadratic probing, searching for the bucket $(h(k) + i^2)\%b$ and $(h(k) - i^2)\%b$.
- rehashing, use a series of hash function: h_1, h_2, . . . , h_m.

Secure Hashing Function

- In computer security, message authentification,
  - A message M transmitted over an insecure channel from A to B.
  - We want B to be confident that the received message is the original message that was transmitted and not forgery.
  - Assume that we have a means to transmit messages much smaller than M securely.
    - encrypt the small message
    - or transmit the small message through a more expensive but far more secure channel.

Secure Hashing Function

- If M is altered along in insecure channel, B receives a different message M'.
- B computes $h(M')$, if $h(M') \neq h(M)$, then didn't not receive correct message.
- h should be designed so that hard to find M' such that $h(M') = h(M)$.