Hashing

Shiv Shankar Dayal
March 1, 2024

Introduction to Hashing

Hashing is a technique to compute some value from a given input. This value can be of fixed length or variable length. We can treat this as a mathematical function, which is called a *hash function*. The output of these hash functions is called *keys*, the input is called *value* and the table where keys and values are stored is called the hash table. The main purpose of a hash function is to perform a fast search. The trees which we have studied are at best $O(\log N)$.

Consider most common words of English. These are A, AND, ARE, AS, AT, BE, BUT, BY, FOR, FROM, HIS, IN, IS, IT, ON, OR, OF, ON, OR, WAS, WHICH, WITH, HAD, HAVE, HE, HER, THAT, THE and THIS. Let us consider a function $f(x)$, which can transform these 29 words between 10 and 28. We will need about 39 table locations to store the 29 keys.

However, it is very hard to find a good hash function. There are about $39^{29} = 10^{46}$ possible functions from a 29-value into a 39-value set and only $39!/10! = 5.6 \times 10^{39}$ of them will give distinct values for each argument.

You can study birthday paradox in details which says that if 23 people are present in a room then there is bound to be two people having same birthday. If we select a random function that maps 23 keys into a table of size 365, the probability that no two keys map to same location is only $.4927$.

Contd.

The duplicate values are called collisions. The problem of finding a good hash function is so severe that GCC uses red-black trees as storage medium for C++ map/hash table and the argument which is put forth by GCC documentation is that with collisions most hash functions worsen to $O(\log N)$ complexity.

Properties of a good hash function

Following are desirable qualities of a hash function:

- Efficient to compute so that insertion in hash table is easy,
- Keys should be uniformly distributed,
- Minimum collision i.e. high periodicity(when different inputs produce same value then it is called a collision), and
- Number of items in the table divided by the size of the table i.e. load factor should be low.

Some simple hash functions

- **Division Method:** $f(x) = x \mod M$, here x is the input value and M is the hash table size. A prime M is suitable as that makes sure that distribution is even. However, it leads to poor performance as consecutive keys map to consecutive locations.
- **Middle Sqaure Method:** In this case we square the key x and extract m middle digits from the square. The benefit is that result is not dominated by distribution of top and bottom digits. However, the size of key and collisions are limitations are the problem.
- **Digit Folding Method:** In this method, we divide the key x into parts $x_1, x_2, \ldots, x_n$, where each part has the same number of digits except for the last part that can have less digits than the other parts; finally we add the parts.
- **Multiplication Method:** We choose a constant value $A \in (0, 1)$, multiply key with A, extract the fractional part, multiply the result with M, and take the floor of result.

Collision resolution techniques

- Separate chaining/Open hashing
- Open addressing/Closed hashing
  - Linear probing
  - Quadratic probing
  - Double hashing

Separate chaning/Open hashing

Each cell of the hash table points to a linked list of records that have the same hash function value. Hash function is $x \mod 5$. Input values are $12, 16, 22, 25, 35, 45, 63$.

$|25 -> 35 -> 45|16|12 -> 22|63|||$

Linear probing

Next available slot is taken in case of collision. Hash function is $x \mod 5$. Input values are $12, 16, 22, 25, 35$.

$|25|16|12|22|35|$

Quadratic probing

If the slot $hash(x)\%M$ is full then try $hash(x) + 1^2$, next try $hash(x) + 2^2$, next try $hash(x) + 3^3$ and so on.

Double hashing

$$h(x, y) = (h1(x) + y * h2(x))\%M$$