HASHING

By,

Durgesh B Garikipati

Ishrath Munir

Hashing

- Sorting was putting things in nice neat order
- Hashing is the opposite
  - Put things in a “random” order
  - Algorithmically determine position of any given item
  - Improve search from $O(\log_2 N)$ to $O(1)$

HOW?

- Place everything in an array based on search key value
  - Run the search key through an address calculator (the hash function)
  - Insert/retrieve from location specified by the hash function (address calculator)
- This scheme is termed hashing
  - The function is the hash function
  - The array is called a hash table

Why not discard other data structures?

The mapping not always one-to-one

- A major drawback
- Results in collisions - two items mapped to same location
  - which adds complexity
- Ordered operations do not work well
- Making array large enough to avoid collisions seldom practical

Ordered Operations

- We said earlier that hash tables don’t work well for ordered operations
  - Now you should know enough to understand why
- Problems:
  - Display the users stored in a hash table alphabetically
  - Finding the lowest/highest values in a database
    - To do these operations, we need to retrieve all the data in the hash table, sort or compare as we retrieve.

Typical Hash Function

- Must
  - Be fast and easy to compute
  - Place items evenly through the hash table
    - More important, because the biggest performance loss is due to resolving collisions

Different Hash Functions

- Division and Remainder method
  hash (key) = key % ARRAY_SIZE
  - Can distribute array items evenly by choosing a prime number for ARRAY_ SIZE
- Folding
  Add the digits
  - hash(001364825) = 1+3+6+4+8+2+5 = 29
    - Produces a normal distribution – too bunched up

Continued..

- Universal hashing
  -select hash function at random from designed class of functions
  - Diff address for the same inputs
- Digit Rearrangement
  - taking part of the original value or key , reversing of order, using that sequence of digits

Collision Resolving

- Collision
  4567 % 101 = 22 = 7597 % 101 : collision
  - Two approaches to resolve
    - Alter the hash table to allow multiple entries per location
    - Allocate another location within the hash table

Restructure Hash Table

- Multiple items stored in a single array location
  - Two schemes to do this
    - Buckets
    - Separate chaining

Buckets

- Each array location is itself an array called a bucket
  - Problem is choosing size of these arrays
    - If too small, will fill up and must probe
    - If too large, wasted space

Chaining

- Hash table is an array of linked lists
  - Each array element is a pointer to a linked list, the chain
    - Insertion: place at beginning of chain
    - Deletion/retrieval: search the appropriate chain

Chaining

- Insertion is O(1)
- Deletion/retrieval depend on chain length

- Searching
  - $(1 + \alpha)$ for successful search
  - $(1+ \alpha)$ for unsuccessful search
  - Best case: O(1), worst case: O(N)

Open Addressing

- Open Addressing

Probe for another empty, open location

- Linear Probing
- Quadratic probing
- Double Hashing

Linear probing

- Probe sequentially until an empty spot is found

Given key K

First slot probed T[h'(k)], if collision continue

T[h'(k)+1]….. Up to T[m-1]

\- Wrap around if you get to the end
  - Additions straightforward
  - Deletions pose a problem

Problems of Linear Probing

- Items tend to cluster together in consecutively occupied locations
  - Termed primary clustering
  - A cluster tends to increase in size
  - Clusters can merge into larger clusters
  - Increase the average search time
  - Primary clustering makes linear probing inefficient

Quadratic probing

- Uses a hash function

$h(k,i) = (h'(k)+C_1 i + C_2 i^2)$ mod m

$C_1 C_2$ - auxillary constants

i = 0 ,……,m-1

\- Positions are offset in a quadratic manner
  - Virtually eliminates primary clusters
  - Suffers from secondary clustering since same probe sequence used to resolve collision at original location

Double Hashing

- Best methods of open addressing
- Uses a hash function

$h(k,i) = (h_1(k) + ih_2(k))$ mod m

h_1, h_2 auxillary hash functions

- Drastically reduces clustering
- Previous methods are key independent
- Double hashing uses key-dependent probe sequences

Continued

- Choose first function hash() as usual
- Follow these guidelines for hash2()
  - hash2(key) != 0
  - hash2() != hash()

Hashing Efficiency

- Involves the load factor α
  - α = average # of elements
  - α is a measure of how full the table is
  - As the hash table fills
    - α increases
    - Chance of collision increases
    - Search time increases
    - Hashing efficiency decreases

Linear Probing

- As collisions increase
  - Probe sequence increases
  - Search time increases
  - α should not exceed 2/3 for linear probing
- Best case : O(1)
- Worst Case : O(1+ α)

Quadratic probing ,Double Hashing

- On average, both require fewer comparisons than linear probing
  - For α = 2/3, avg unsuccessful search might require at most 3 compares, successful 2
- All three open addressing schemes suffer when unable to predict number of insertions and deletions as table may be too small

Good Hash Function

- Easy and fast to compute
- Scatter data evenly through table
  - Check for various types of data
    - Random data
    - Nonrandom data
  - Two general principles for even distribution
    - Use the whole key
    - If modulo math, base should be prime

Questions

?

References

1) Introduction to Algorithms
Thomas H. Cormen, Charles E.Leiserson, Ronald L. Rivest

2) http://www.palfrader.org/hashing/

3)http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/hash_tables.html

4)http://64.233.179.104/search?q=cache:9dqRIPQhL5gJ:www.npsnet.org/~mcdowell/CS3971/hashing+hashing%2Befficiency&hl=en