Hashing

Monday, March 12, 2018 11:35 PM

	Sorted Array	Balanced BST	Hashing
Insertion	O(n)	O(log n)	O(1) avg
Deletion	O(n)	O(log n)	O(1) avg
Retrieval	O(log n)	O(log n)	O(1) avg

Direct Addressing Table

- Keys must be integer values and range is small/ dense (minimal gaps)

Hashing

- Map large integers -> small, non-integer -> integer
- Collision: different keys may not be hashed to different buckets

Hash Functions

- Fast to compute, even scatter
- Minimal collisions, less space
- Perfect 1-1 mapping (no collision, but all keys must be known)
    - Used by compiler/ interpreter to check for reserved words (GNU gperf)
    - Minimal: table size same as #keywords supplied
- Uniform hash function: keys evenly distributed -> can reduce size of hashtable

Division method

- Modulo arithmetic
- m = hash table size (power 2 = take n bits, power 10 = take n digits)
- Pick prime number close to power of 2 (reduce collisions)

Multiplication method

- Multiply by A, A between 0 and 1 (e.g. Phi)
- Multiply integer part by m (table size)

Hashing Strings

- Add letters and modulo m
    - Does not consider position of characters
    - Can shift sum by multiplying (e.g. sum = sum*37 + c)

Collision Resolution

- Birthday paradox
- Minimise clustering, always find empty slot if it exists
- Different probe sequences even for same initial probe (minimise secondary clustering)
- Should be fast

Separate Chaining

- LL to store collided values
- Load factor = no. of keys/ size (can be bounded)
    - $\alpha = n/m$

- Find $O(1 + \alpha)$
  Insert $O(1)$
  Delete $O(1 + \alpha)$
- Binding may require reconstruction (rehash all keys into a bigger table, increase m and reduce a)

Linear Probing

- Look for the next empty slot
- Delete checks for empty slot
  - Cannot simply remove key value (creates gap)
  - Lazy deletion: mark as occupied/ deleted/ empty in status array
- Probing continues until empty slot (consecutive occupied slots)
  - Primary Clustering problem
- Modified: increase probe sequence distance by d, which is co-prime to m
  - Avoids primary clustering, and yet can cover all slots (co-prime)

Quadratic Probing

$$\text{hash(key)}$$
$$(\ \text{hash(key)} + \mathbf{1}\ )\ \%\ m \qquad \text{jump } \mathbf{1^2}$$
$$(\ \text{hash(key)} + \mathbf{4}\ )\ \%\ m \qquad \text{jump } \mathbf{2^2}$$
$$(\ \text{hash(key)} + \mathbf{9}\ )\ \%\ m \qquad \text{jump } \mathbf{3^2}$$

- Probe sequence increases by a square number (k + 1, k + 4, 9…)
- For load factor < 0.5 and m is prime, there will always be an empty slot
  - Odd number of steps -> changing index parity
- How to make sure there will be a termination? (see lec)
  - Since the probe distance changes, the probing will not be circular
- Secondary clustering: same initial position -> same probe sequence

Double Hashing

- Secondary hash function determines probe distance for collisions
  - Fixed probe distance for each number (similar to linear probing/ modified linear probing)
  - Cannot evaluate to 0, else there is no jumping
  - E.g. hash2(k) = 5 - (k%5)

Comparison with BST

- Ordered traversal of all items
  - BST = O(n)
  - Hashing = O(n log n) - take out all and sort
- Hashing is not good for searching for min/ max values, range search
- Hashing is usually used to retrieve a particular item given a key

Hashtable<K,V>

- `extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable`
- Non-null objects can be used as key/ value
- Dictionary is abstract parent class
- Default load factor = 0.75, size 11
- get(k), put(k,v), contains(v)