Closed Hashing:

Quadratic Probing, Double Hashing, and Analysis

CIT594

Last Time:
Closed Hashing

Closed Hashing

- A *hash system* where *all records are stored in slots inside the hash table*
- Implementations:
    - **Closed hashing with buckets**
    - **Closed hashing with no buckets**

Closed Hashing with No Buckets

Collision Resolution Policy

- The process of finding the proper position in a hash table that contains the desired record
- Used if the hash function did not return the correct position for that record due to a collision with another record
- Mainly used in closed hashing systems with no buckets
- A good collision should ensure that **empty slots** in the table **have** an **equal probability of receiving the next record inserted**

Collision Resolution

- Goal: find a free slot in the hash table when the home position for the record is already occupied
- Uses a probe function

Collision Resolution

- Probe function: function used by a *collision resolution* method to calculate where to look next in the *hash table*
- Probe sequence: the series of *slots* visited by the *probe function* during *collision resolution*.

We will use:

- Hash function: simple mod (**%**)
- `Slot = key % array_size`

Collision Resolution

1. Find home slot
    - `int pos = home = h(K);` where h is the hash function and K is the key
2. Probe sequence (iterative process)
    - `pos = (home + p(k, i)) % M;`
        - Initialize `i` at 1
        - Increment `i` until the slot at `pos` is empty

Probe function

- The probe function returns an offset from the original home position

Collision Resolution Policies

- Linear probing
- Linear probing by steps
- Pseudo-random probing
- Quadratic probing
- Double hashing

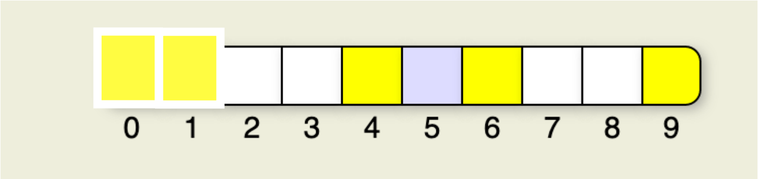
Quadratic Probing

Quadratic Probing

- Eliminates primary clustering
- Probe function is *quadratic*.
- Probe function:
  - $p(k, i) = c_1 i^2 + c_2 i + c_3$
- Simplest form:
  - $p(k, i) = i^2$
  - Probe sequence: the i^{th} value is: $h(K) + i^2$

Quadratic Probing

- Problem: not all slots visited by the “simplest form” probe function



If a value hashes to slot 5. Only the slots in yellow will be visited

Quadratic Probing

- Solution:
  - Length of hash table: power of 2
  - probe function: $p(k, i) = (i^2 + i)/ 2$

All slots will be visited by the probe function

  - Given a hash table of length 8, if a value hashes to slot 0, the probe sequence will be: **1, 3, 6, 2, 7, 5, 4**

Quadratic Probing

- Google SparseHash Tables (https://github.com/sparsehash/sparsehash)

- Facebook Haystack (photo storage system) uses Google SparseHash Tables (https://engineering.fb.com/core-data/needle-in-a-haystack-efficient-storage-of-billions-of-photos/)

Secondary Clustering

- Pseudo-random probing and quadratic probing ignore the key when computing the probe sequence
- Two records with the same home slot will share the same probe sequence
- Secondary Clustering results from the keys hashing to the same slot of the table sharing the same probe sequence

Double Hashing

Double Hashing

- Eliminates secondary clustering
- Probe function uses the original key
- Probe function: `p(k, i) = i*h`$_2$`(k)`
- h_2 is a second hash function

Double Hashing

- Implementation 1:
    - select M to be a prime number
    - $h_2(k) = 1 + (k \% (M-1))$
    - $p(k, i) = i * h_2(k)$

Double Hashing

- Example:

M = 11

A record with key = 0 will generate the probe sequence: 1, 2, 3 , 4, 5, 6, 7, 8, 9, 10

A record with key = 11 will generate the probe sequence: 2, 4 , 6, 8, 10, 1, 3, 5, 7, 9

Double Hashing

- Implementation 2:
  - select *M* to be a power of two
  - h_2 returns an odd number
  - h_2 `(k)= (((`*k/M*`) % (`*M*`/2))*2)+1`
  - `p(k, i) = i * h2(k)`

Double Hashing

- Example:

M = 16

A record with key = 0 will generate the probe sequence: 1, 2, 3 , 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

A record with key = 16 will generate the probe sequence: 3, 6, 9, 12, 15, 2, 5, 8, 11, 14, 1, 4, 7, 10, 13

Deletion

Deletion

- Challenges:
  - Empty slots should not stop the probe sequence when searching
  - The freed slot should be available to a future insertion

Deletion

- A tombstone is used to mark a *slot* in the *hash table* where a record has been deleted.
- **Searching**: if a tombstone is encountered, the probe sequence continues
- **Insertion**: if a tombstone is encountered, that slot is used to store the new record.
- **The insertion procedure must check for duplicates**

Analysis

Analysis of Closed Hashing

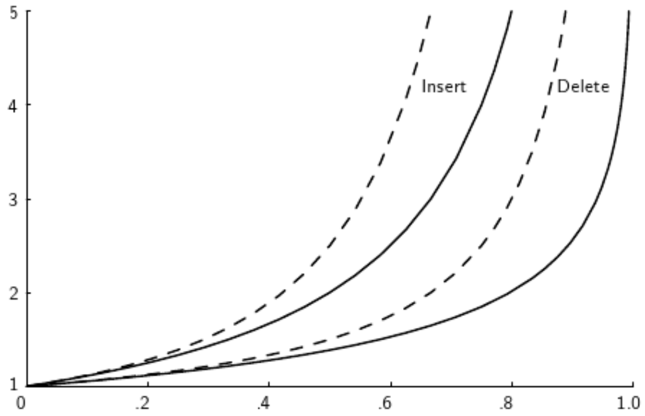
- Assuming that every slot in the table has equal probability of being the home slot for the next record
- The probability of finding the home position is N/M (load factor - α)
- The expected number of probes is:

$$1 + \sum_{i=1}^{\infty} (N/M)^i = 1/(1 - \alpha).$$

- The cost is a function of the load factor

Analysis of Closed Hashing

- Horizontal axis is the value for α
- Vertical axis is the expected number of accesses to the hash table
- Dashed lines show the cost for linear probing
- Solid lines show the cost for "random" probing (a theoretical lower bound)



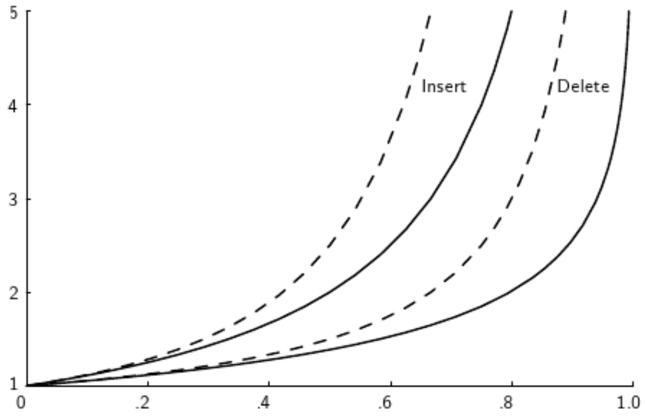
Plot showing the growth rate of the cost for insertion and deletion into a hash table as the load factor.

Analysis of Closed Hashing

- For small values of α, the expected cost is low. It remains below two until the hash table is about half full

- **Rule of thumb:** design a hashing system so that the hash table never gets above about half full

- Select the table size accordingly



Plot showing the growth rate of the cost for insertion and deletion into a hash table as the load factor.