Closed Hashing:

Linear Probing, Linear Probing by Steps, and Pseudo-Random Probing

CIT594

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

Linear Probing

Linear Probing

- Works by moving sequentially through the hash table from the *home slot*.
- Probe function:
    - `p(k, i) = i`
- If home slot is `home`, the probe sequence will be `home + 1, home + 2, home + 3, … home + (M - 1)`

Example

- Hash function: simple mod (%)
- `M = 10`
- `home= key % M`
- `p(key, i) = i`
- `pos = (home + i) % M;`
- `Keys = [9877, 9050, 2037, 1059, 7200, 3348]`

Primary Clustering

- The tendency in certain collision resolution methods to create clustering in sections of the hash table

- Happens when a group of keys follow the same probe sequence during collision resolution

- **primary clustering** lead to **empty slots** in the table to **not have** an **equal probability of receiving the next record inserted**

Primary Clustering

- Linear probing leads to primary clustering
- Linear probing is one of the worst collision resolution methods

Linear Probing by Steps

Linear Probing by Steps

- Goal: avoid primary clustering / improve linear probing
- Idea: skip slots by some constant *c* other than 1
- Probe function:
  - `p(k, i) = c * i`
- ***c* must be relatively prime to *M*** to generate a linear probing sequence that visits all slots in the table

Example

- Hash function: simple mod (%)
- `M = 10`
- `home= key % M`
- `c = 3`
- `p(key, i) = c * i`
- `pos = (home + 3i) % M;`
- `Keys = [9877, 9050, 2037, 1059, 7200, 3348]`

Pseudo-Random Probing

Pseudo-Random probing

- Idea: select the next position on the probe sequence at random from the unvisited slots
- The random sequence should be the same for insertion and searching (impossible for a truly random sequence)

Pseudo-Random probing

- Stores a random permutation of the values 1 through the size of the *hash table*
- The term i of the *probe sequence* is the value of position i in the permutation array

Pseudo-Random probing

- Probe function:
  - `p(k, i) = Permutation[i]`
- **Permutation:**
  - Array of length *M*
  - Stores a value of 0 in position **Permutation[0]**
  - Stores a random permutation of the values from 1 to *M*−1 in slots 1 to *M*−1.

Example

- Hash function: simple mod (%)
- `M = 10`
- `home = key % M`
- `Permutation = [0, 3, 7, 6, 1, 4, 9, 2, 5, 8]`
- `p(key, i) = Permutation[i]`
- `pos = (home + Permutation[i]) % M;`
- `Keys = [157, 273, 17, 913, 110, 258]`