Data Structure

Lecture#22: Searching 3
(Chapter 9)

U Kang
Seoul National University
In This Lecture

- Motivation of collision resolution policy
- Open hashing for collision resolution
- Closed hashing for collision resolution
Hashing

- Given a key $k$, can we search $k$ in a constant time?
  - Yes!
  - We can do it by hashing. It is faster than binary search, QBS, and sequential search.

- Hash table HT is the array that holds the records
  - HT has $M$ slots (slots numbered from 0 to $M-1$)

- Hash function $h$ maps key $K$ to a number (position)
  - $0 \leq h(K) \leq M - 1$
  - E.g., $h(K) = K \% M$

- Goal of a hashing system: arrange things such that for a given key $K$, and $i = h(K)$, the record for the key $K$ is located in $HT[i]$
  - Then, the searching time would be constant
Hashing

- Goal of a hashing system: arrange things such that for a given key $K$, and $i = h(K)$, the record for the key $K$ is located in HT[$i$]
- Collision: two different keys $k_1$ and $k_2$ map to a slot
  - $h(k_1) = \beta = h(k_2)$

- Finding a record with key value $K$ by hashing:
  - Compute the table location $h(K)$
  - Starting with slot $h(K)$, locate the record containing key $K$ using a collision resolution policy
Collision Resolution

Collision is unavoidable in many cases. How can we insert an item to hash table in case of collision?

Collision resolution techniques
- Open hashing (also called `separate chaining’)
  - Collisions are stored outside the table
- Closed hashing (also called `open addressing’)
  - Collisions are stored at another slot in the table
Open Hashing

- Open hashing (also called ‘separate chaining’)
  - Collisions are stored outside the table
  - Limitation: some slots in the table may not be used
Closed Hashing

- Closed hashing (also called `open addressing’)
  - Collisions are stored at another slot in the table
  - Each record $R$ with key $k_R$ has a home position $h(k_R)$
  - If another record already occupies $R$’s home position, $R$ will be stored at some other slot in the table

- Examples
  - Bucket Hashing
  - Linear Probing
  - …
Bucket Hashing (1)

- Group hash table slots into buckets
  - M slots are divided into B buckets (each bucket: M/B slots)
- Hash function (key->bucket number) assigns each record to the first slot in the bucket that the record is mapped to.
  - If the first slot is empty, insert
  - If the first slot is occupied, find the next empty slot in the bucket
  - If all the slots in the bucket are occupied, store in an overflow bucket

Insertion order: 9877, 2007, 1000, 9530, 3013, 9879, 1057

\[ h(K) = K \mod 5 \]

M = 10, B = 5
Bucket Hashing (2)

- A variation on bucket hashing: hash a key to a slot in the hash table as though bucketing were not being used
  - If the slot is empty, insert
  - If the slot is occupied, find the next empty slot in the bucket
  - If all the slots in the bucket are occupied, store in an overflow bucket

Insertion order:
9877, 2007, 1000, 9530, 3013, 9879, 1057

\[ M = 10, \ B = 5 \]
\[ h(K) = K \mod 10 \]
Bucket Hashing (3)

- Bucket hashing vs open hashing?
  - Bucket hashing has more collision => longer running time to search an item
  - Bucket hashing has less storage requirement => less space

- Limitation of Bucket Hashing
  - If a bucket is full, then all the inserts to the bucket will be stored in the overflow bucket, even when the hash table has many empty areas
Linear Probing

- Closed hashing with no bucketing, and a collision resolution policy can use any slot in the hash table
- If the home position is occupied, the new position is determined by \((\text{home} + \text{probe\_function()}\))
  - The sequence of slots is called `probe sequence`

```java
/*/ Insert record r with key k into HT */
void hashInsert(Key k, E r) {
    int home;         // Home position for r
    int pos = home = h(k);   // Initial position
    for (int i=1; HT[pos] != null; i++) {
        pos = (home + p(k, i)) % M;  // Next probe slot
        assert HT[pos].key().compareTo(k) != 0 :
            "Duplicates not allowed"
    }
    HT[pos] = new KVpair<Key,E>(k, r); // Insert R
}
```
Linear Probing

- Linear probing: move down $i$ slots in the table
  - $p(K, i) = i$

$M = 10$

$h(K) = K \mod 10$

Insertion order:
1001, 9050, 9877, 2037

1059 is added
Linear Probing

Problem of linear probing
- Primary clustering: nonempty slots are clustered, and thus giving unequal probability to empty slots
- E.g., in the figure below, what is the probability that a random key $k$ will be inserted at slot $i$?
  - $P(\text{slot 2}) = 0.6$
  - $P(\text{slot 3}) = P(\text{slot 4}) = P(\text{slot 5}) = P(\text{slot 6}) = 0.1$
Improved Collision Revision

- Use linear probing, but skip slots by a constant $c$
  - $p(K, i) = ci$
  - $c$ should be relatively prime to $M$ (why?)
  - Limitation: one section of slots will be used more, if inputs are skewed
  - E.g., if $c = 2$, and accesses are all odd numbers

- Pseudo-random probing
  - $p(K, i) = \text{Perm}[i - 1]$, where $\text{Perm}$ is an array of length $M-1$ containing a random permutation of the values from 1 to $M-1$

- Quadratic probing
  - $p(K, i) = c_1i^2 + c_2i + c_3$
Performance of Closed Hashing

Cost (# of probe)

Linear Probing Lower Bound

Insert

Delete

Load Factor
Discussion

- How can we make the probability of collision very small?
  - Open hashing vs. closed hashing
  - Time and space tradeoff

- Open hashing vs. bucket hashing
  - Bucket hashing uses space more efficiently, but has more collisions

- Bucket hashing vs linear probing?
What you need to know

- Collision resolution
  - Hard to avoid collision in most cases

- Open hashing
  - Simple, but some slots may not be used

- Closed hashing
  - Open hashing vs. bucket hashing
  - Bucket hashing vs. linear probing
Questions?