CSC345 Discussion 9

Hashing

Hash Table

Hash Table

A hash table is a generalization of an ordinary array.

Features: Array size is generally chosen to be comparable to (perhaps, a small multiple of) our bound on dictionary size

Worst-case performance is generally poor
However, the average-case performance is extremely good

Terminology

Hash Function: Maps key values to positions

Hash Table: The array that holds the records

Slot: A position in the hash table

Collisions

A collision occurs if two keys k1 and k2 (used in the dictionary) hash to the same location, that is, h(k1) = h(k2) .

Note: Collisions are unavoidable if the size of the dictionary is greater than the table size.

There are several different kinds of hash tables that use different ways to deal with collisions.

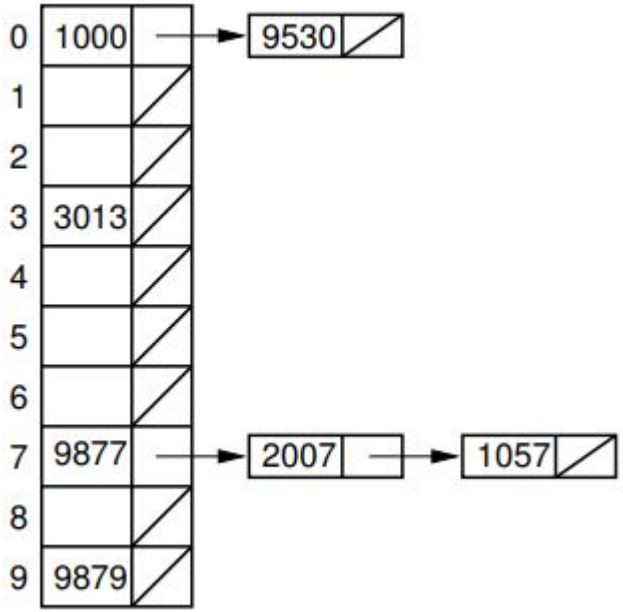
Open Hashing (chaining)

Closed Hashing (open addressing)

Open Hashing

Open Hashing

Collisions are stored outside the table.

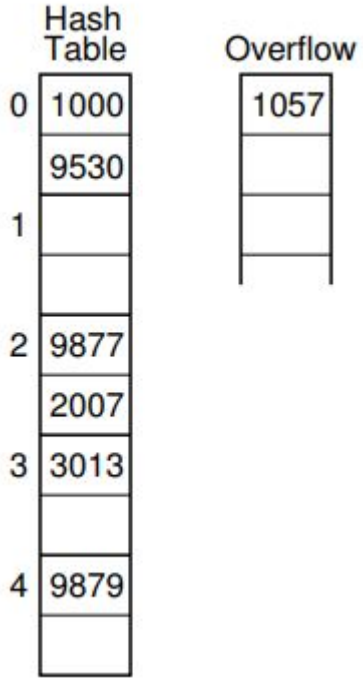


Closed Hashing

Closed Hashing

Storing one of the records at another slot in the table

An example of Bucket Hashing



When is Hashing Bad?

When is Hashing Bad?

Hashing is not good for applications where multiple records with the same key value are permitted.

Hashing is not a good method for answering range searches.

In other words, we cannot easily find all records (if any) whose key values fall within a certain range. Nor can we easily find the record with the minimum or maximum key value, or visit the records in key order.

When is Hashing Good?

When is Hashing Good?

Hashing is most appropriate for answering the question, “What record, if any, has key value K?”

For applications where access involves only exact-match queries, hashing is usually the search method of choice because it is extremely efficient when implemented correctly.

Problem

What is the average case cost for a search on a properly tuned hash system that stores n records:

Θ(logn)
Θ(1)
Θ(n^2)
Θ(n)

Problem

What is the average case cost for a search on a properly tuned hash system that stores n records:

Θ(logn)
Θ(1)
Θ(n^2)
Θ(n)

Problem

(Recall that M refers to the size of a hash table.)
A hash function must:

Return the position of a free slot in the table
Return a value between 0 and M-1
Return a value between 1 and M
Return values in equal distribution throughout the table

Problem

(Recall that MM refers to the size of a hash table.)
A hash function must:

Return the position of a free slot in the table
Return a value between 0 and M-1
Return a value between 1 and M
Return values in equal distribution throughout the table

A hash function should always give you a legal index into the hash table array.

Problem

Assume that you have a seven-slot closed hash table (the slots are numbered 0 through 6). Show the final hash table that would result if you used the hash function h(k) = k mod 7 and linear probing on this list of numbers: 3, 12, 9, 2. After inserting the record with key value 2, list for each empty slot the probability that it will be the next one filled.

Problem

Assume that you are hashing key K to a hash table of n slots (indexed from 0 to $n-1$). For each of the following functions $\mathbf{h}(K)$, is the function acceptable as a hash function (i.e., would the hash program work correctly for both insertions and searches), and if so, is it a good hash function? Function **`Random(n)`** returns a random integer between 0 and $n-1$, inclusive.

(a) $\mathbf{h}(k) = k/n$ where k and n are integers.
(b) $\mathbf{h}(k) = 1$.
(c) $\mathbf{h}(k) = (k + \text{Random}(n)) \bmod n$.
(d) $\mathbf{h}(k) = k \bmod n$ where n is a prime number.