COMP 206 (T4- 2020)

Hashing (Exercise Worksheet)

A hash function $h : \text{int} \rightarrow \text{int}^+$ assigns a positive number $h(k)$ to a numerical key k. The number $h(k)$ is also known as the hash value of the key.

Recall that in a hash table `T` we have an array (table) of size `m` and each of the array (table) entry stores the starting node of a linked list. For our dicussion here we will call the table entries as buckets, where each bucket has a single linked list inside it. Note that a bucket contains all the keys k with the same `h(k)%m` value in it.

Exercise Create a hash table of size 4 and insert the keys `{1,2,3,4,5}` in it. Use the hash function `h(k) = (2*k + 3)%4`. Notice that one of the buckets has at least 2 keys in it. Can you come up with a hash function such that each bucket contains at most 1 key?

The worst case running time of `lookup(T, k)` depends on the number of items in the largest bucket.

Answer the following before you read on.

Exercise Distribute `n=10` items in `m=4` buckets in any way you want. What is the size of the largest bucket (the one having maximum number of items)? Can you redistribute the items from the largest bucket to other buckets so as to reduce the size of the largest bucket (note that some other bucket may become largest now)? What is the smallest number of items possible for the largest bucket? Using your observations try to answer the previous question for any value of `n` and `m`.

If we are storing n keys in the table then the *smallest* worst case time occurs when each of the buckets have roughly equal number of items in them.

Exercise Show that the largest bucket can not have less than `n/m` items in it. (Hint: what is the total number of items in all the buckets when the largest bucket has strictly less than `n/m` items ?)

Due to the exercise above the running time cannot be better than O(n/m) even if we chose a hash function which could distribute all the items in the buckets as evenly as possible. However, we can always resize the table (double m, and insert all the old keys) to ensure that `n/m` is at most some constant `c`, in which case the lookup time would be amortized `O(c) = O(1)` due to a reason similar to unbounded arrays (where we double up the array size as well).

PRNG

Therefore, it is enough to come up with a hash function which can distribute all the keys as evenly as possible in the buckets (i.e. ensure that the largest bucket is as small as possible). Suppose we choose a random number generator to generate hash values and try to insert keys into an empty table using it. The random number generator (RNG) generates a random number in the range [0,m) for each entry that needs to be inserted. Due the property that the RNG will produce numbers with equal probability, the buckets will have roughly equal number of items in them with high probability. However, retrieving the keys is a problem as the RNG generates a hash value irrespective of the key k which was inserted in the table using it. The functions which produce "seemingly" random numbers but depend on some input value are called as pseudo random number generators (PRNG). We will be using PRNGs to generate hash value of keys. Linear congruential generators are a class of PRNGs.

Linear congruential generators (LCG)

Suppose we have 3 numbers `a,c,d`, such that `a!=0` and `gcd(c, d)=1`. One of the simplest form of LCG is:

$$f(x) = (ax + c)\%d.$$

Exercise Write a function `int lcg(int x)` which computes and returns `1664525 * x + 1013904223`. As int is 32 bit long, we have $d = 2^{32}$ automatically.

Exercise Remember that a hash function returns a positive number for any input key. Write a function `int hash_lcg(int x)` which returns the value of absolute value of `lgc(x)`.

Exercise Write a function `int key_index(int k, int m)` which returns the index of key k in a table of size `m`, using the hash function defined above.