HW -5 (Implementing Hashing)

5 Absolute Marks

DeadLine Jan 9, 2014 11:59PM

Email your Code and ReadMe to the following email address: Bitf10a009@pucit.edu.pk

No Viva will be conducted for the homework submitted late (even 1 minute)

Home work is individual, No grouping is allowed

Q – 1: Write a function *generateDistinctRandomKeys* which return an array "D" of n distinct non-negative integers generated randomly in the range [0 – 2^{32}-1] (use 64 if you have 64 bit OS)

D = generateDistinctRandomKeys(n){

}

Q – 2: Call the above function and get the data array "D". Write a program that hashes the values of array "D" one by one into the hash table using the following hash function:

$h(k) = ((A * k) \bmod 2^{32}) rsh(32 - r)$; Where the size of the table is: $m = 2^r$

(Use 64 instead of 32 if you have 64 bit OS)

(a) At the start of the program, generate the Value of the constant A randomly in the range (2^{31} - 2^{32}). Make sure the value generated randomly should not be too close to the either of the extents of the range, otherwise generate it again.
(b) Start by initializing the table of size 1. Use table doubling to expand the table whenever the number of keys already hashed in the table is equal to m. Use chaining to resolve collisions if any.
(c) Count the total number of collisions in the hash table after all the keys have been hashed.

Q – 3: Perfect Hashing: Call the function in Q-1 and get the data array "D". Write a program that hashes the values of array "D" one by one into the hash table using the following Universal hash function:

$$h(k) = ((a * k + b) \bmod p) \bmod m$$ Where p is any prime number greater than m

(a) Let the total number of keys be n, i.e. size (D) == n. Now declare the hash table of size m = n^2. Pick any prime number p greater than m.

(b) Generate positive integer "a" randomly in the range [1 – p-1]. Now generate nonnegative integer "b" randomly in the range [0 – p-1]. (If you are done till now, you have picked the hash function randomly from the universal hash family.)

(c) Now insert all the keys in the hash table one by one using the above hash function. Resolve collisions using chaining.

(d) Count the total number of collisions in the hash table after all the keys have been hashed.

(e) If there are collisions in part (d), then repeat part (b) and generate another hash function randomly.

(f) Using the new hash function, count the collisions if any. If no collisions found, you are done by finding a perfect hash function for your data, other wise, keep on repeating step (b) until you find a perfect hash function.

(g) Repeat Q-3 by getting the new Data array D.