ECE345S 2022

Programming Assignment 2 Report

Group Members: Hetav Pandya, Chirag Sethi

a) Hash Function Design Process

The approach we are using for hashing is Open addressing with Linear probing. Based on this choice we have developed a few design constraints:

1) The size of the hashtable needs to be bigger than the number of elements stored
2) The size of the hashtable should be a prime number to achieve a near uniform distribution
3) The size of the hashtable should not be close to a power of 2 or 10 for achieving a near uniform distribution

Based on the above criterias we finalized the size of our hashtable to be 12289 [1]. The design process of our hash function is as follows:

1) We loop over all the elements in an array starting from the left and multiply them by a counter, where the counter ranges from 1, to the length of the password. This is then added to a variable called sum. Note that we use the ASCII code to convert characters into numbers.

$$Sum = Sum + (counter + 1) * ascii(character)$$

2) We then take the modulo of the sum with respect to the size of the hashtable (12289) and then use it as an index to store the password.

$$index = Sum \,\%\, 12289$$

3) However, if the index is already filled we use linear probing to find the next element. Hence, the resultant hash function looks as follows:

$$hash(j) = (sum(password) + j) \,\%\, 12289$$

Our main motivation behind using this hash function is as follows:

1) We wanted them to be scattered in a random fashion hence we used a prime number far away from a power of 2 and 10.
2) We wanted the length of the password along with the character used in them to have an influence on the hashing function.

b) Analysis of load factor and collision rate

To analyze the dependence of collision rate on load factor we produced two types of graphs for load factors ranging from 0.5 to 1. Note that we did not increase the load factor beyond one because we are using open addressing and a load factor greater than one leads to elements being dropped.

We first created five custom password files each containing 1000 passwords with length varying between 6 and 12 inclusive. Below is the graph showing the collisions vs load factor graph for all the five custom password files.

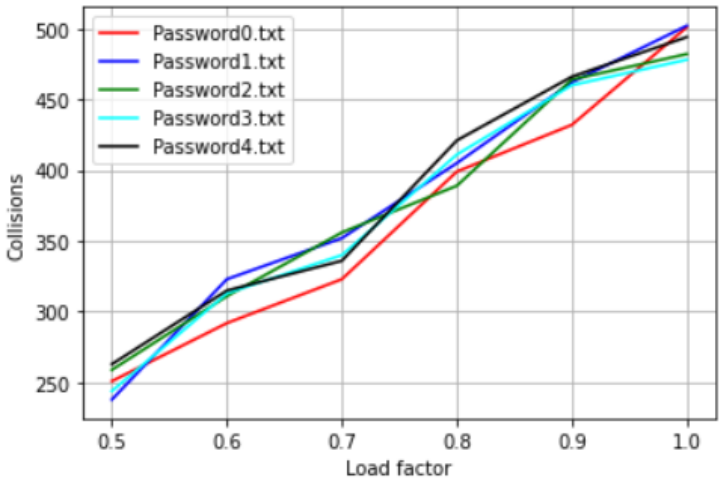

Fig 1.0: Collisions vs Load factor for each password file

Next we generated a plot for the average number of collisions across the files for a given load factor.

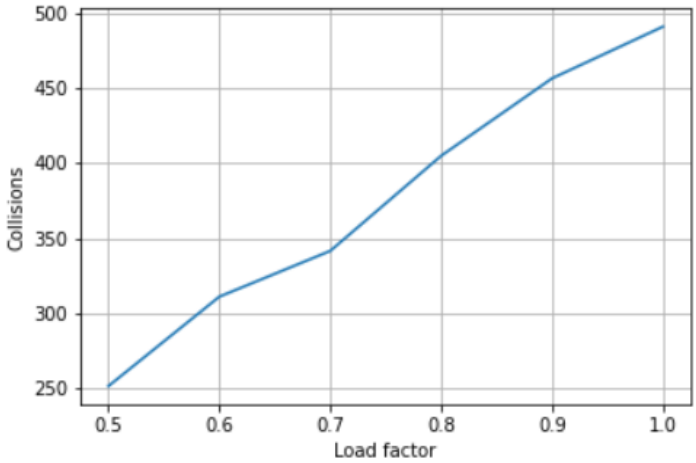

Fig 2.0: Collisions vs Load factor averaged across password files

Here we observe that, as load factor is increased the collision rate increases. This is expected because, as the load factor is brought closer to one, the slots available to insert the element in the hash table decreases. Hence, the chance of collision increases.

References:

[1] *Good hash table primes*. [Online]. Available: https://planetmath.org/goodhashtableprimes. [Accessed: 17-Feb-2022].