CZ2001 Algorithms Example Class 2

Report

Group SS2, Team 4

Mann Singh Arjun (U1822609D)

Padhi Abhinandan (U1823860D)

Ravishankar Amrita (U1822377F)

Taneja Parthasarthi (U1722927B)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCSE)

Contents

Introduction

1.1 Problem Scenario

NTU's Lee Wee Nam library uses "electronic gates" to allow students to enter the library. We assume that these E-Gates record the ID's of the students that enter the library, and our example application is based on searching for specific student ID's using hashing algorithms.

Our program uses open address hashing and double hashing, and we will be comparing between folding and multiplicative congruential (pseudo-random number generator) rehashing functions.

1.2 Data Set

The dataset used is a synthetic dataset for a library which stores the ID's of students who enter the library through the E-Gates. There are a total of 1067 entries in the dataset.

Implementation

2.1 Double Hashing

Double hashing is a technique used in conjunction with open-addressing in hash tables to resolve hash collisions, by using a secondary hash of the key as an offset when a collision occurs.

2.2 Hashing Functions:

A hashing algorithm is a cryptographic hash function. It is a mathematical algorithm that maps data of arbitrary size to a hash of a fixed size.

2.2.1 Auxiliary Hash Function - Division Method

The division method is a hash function that divides an integer key by the hash table size and the remainder will be taken as the hash value. The algorithm for the Division Method is as follows:

$$H(k) = k \bmod h$$

Where: h is some predetermined divisor integer (ie. the table size), k is the preconditioned key and mod stands for modulo.

In this case, we used h = 1069.

```
def Auxiliary(k):
    return (k % hash_size)
```

2.2.2 Folding Method

Given hash table size = 1069, nearest power of 2 is 1024.

2^{10} = 1024, therefore, we use 10 bits from each key to determine their hashed index.

Algorithm:

1) Square the key.
2) Convert the integer into binary.
3) Choose the first 5 and last 5 bits in the binary representation.
4) Convert this new 10 bit binary number back to integer, giving you the hashed index.

```
def Folding(k):
    k = k*k
    c = bin(k)
    c = c[2:]
    c = c[0:5] + c[len(c)-5]
    c = int(c)
    return c
```

2.2.3 Multiplicative Congruential Method (pseudo-number generator):

After choosing a multiplier ‘a’, use f(k) = (a * k) mod h. In this case, a = 6 * ⌊ h / 29 ⌋ + 7

```
def Congruential(k):
  a = 6 * math.floor(hash_size/29) + 7
  return (a*k)%hash_size
```

Statistics

The tables in the following pages show the statistics for both rehashing methods. The total number of successful and unsuccessful searches per method is 10000, and the searches were conducted using random integers. The average CPU time and comparison values are rounded to 4 decimal places, and the CPU used (CPU clock frequency, number of CPU cores/threads) was the same across all searches for consistency (CPU: i7-8750H, 2.2 GHz base clock, up to 4.1 GHz, 6 cores / 12 threads).

Folding Method:

Successful searches:

Load Factor (n / h)	Average CPU time (milliseconds)	Average no. of Key Comparisons
1.00	3.1478	1.9326
0.75	3.6136	1.2473
0.50	1.7444	1.1140
0.25	2.4737	1.0974

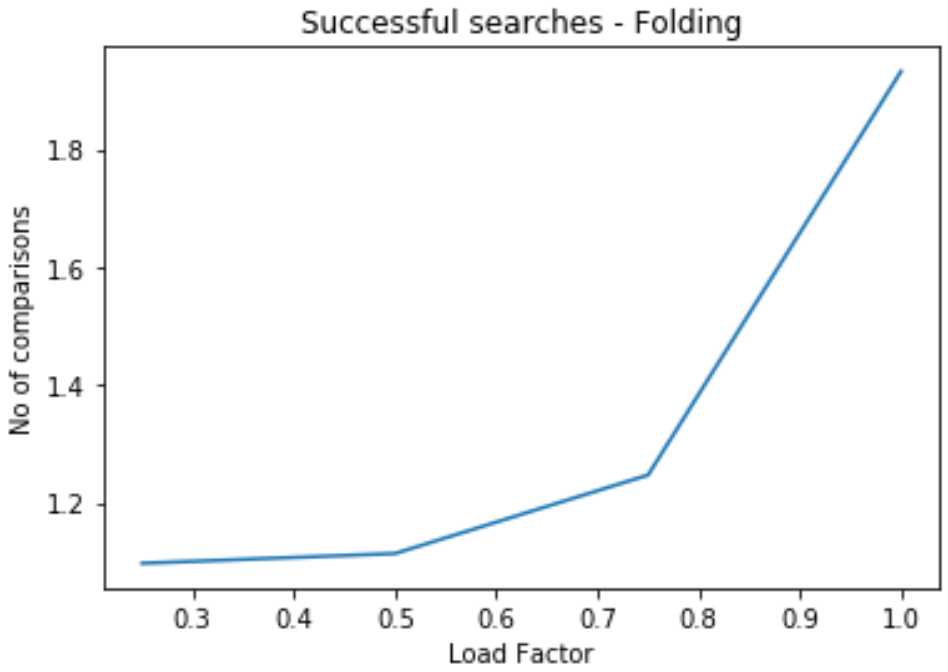
Table 3.1.1

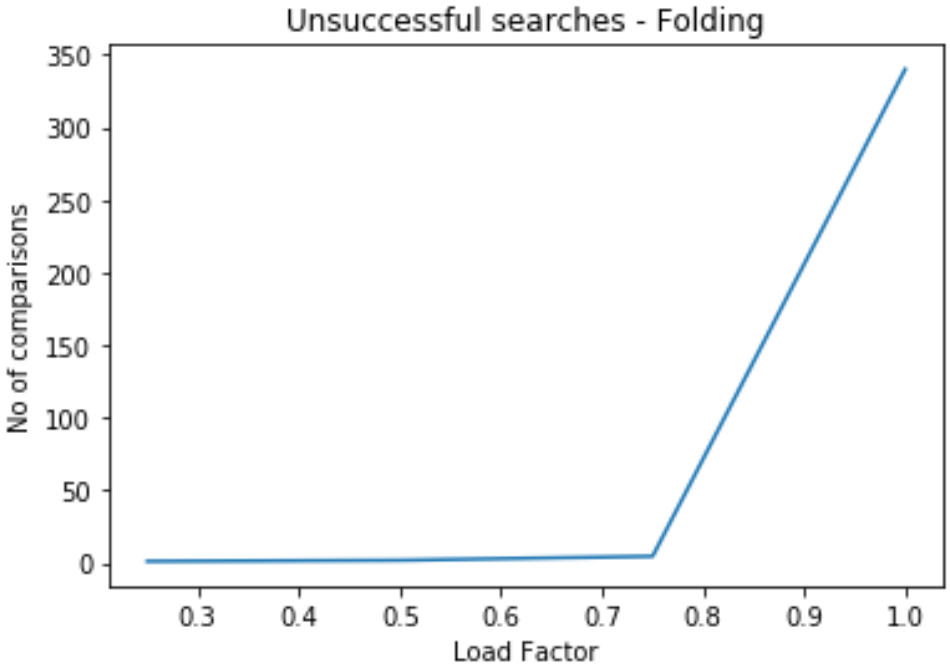
Unsuccessful searches:

Load Factor (n / h)	Average CPU time (milliseconds)	Average Number of Key Comparisons
1.00	608.9336	340.7568
0.75	8.9146	4.5447
0.50	2.3942	1.8200
0.25	0.9403	1.0970

Table 3.1.2

Data Visualisation





Multiplicative Congruential Method

Successful searches:

Load Factor (n / h)	Average CPU time (milliseconds)	Average no. of Key Comparisons
1.00	2.2464	2.0338
0.75	1.5757	1.3514
0.50	1.7678	1.2593
0.25	1.6351	1.1029

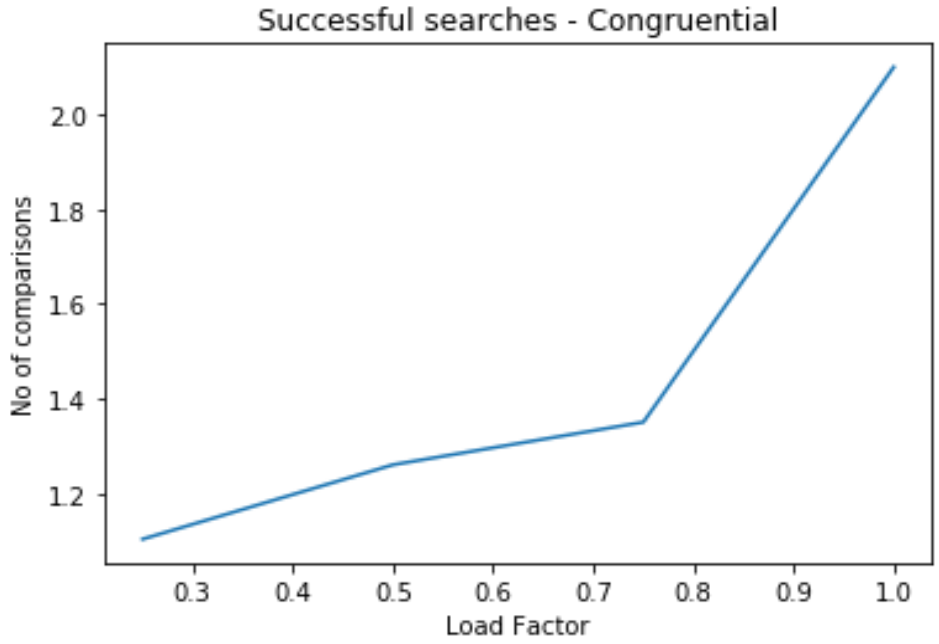
Table 3.2.1

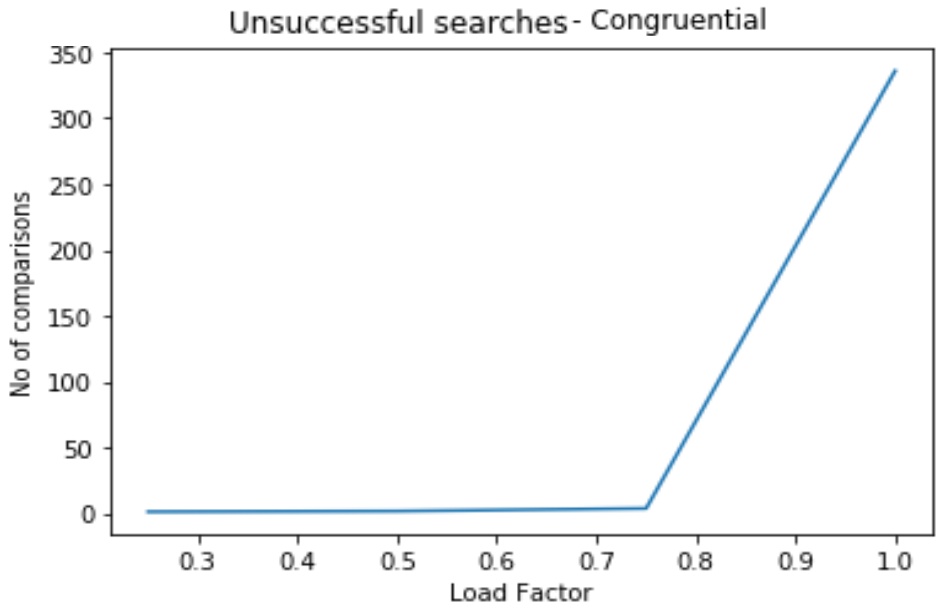
Unsuccessful searches:

Load Factor (n / h)	Average CPU time (milliseconds)	Average Number of Key Comparisons
1.00	526.7234	336.3171
0.75	5.8302	3.6797
0.50	1.6302	1.7478
0.25	0.7840	1.2572

Table 3.2.2

Data Visualisation





Conclusion

Looking at the tables for successful and unsuccessful searches, the average number of comparisons and CPU time taken by the folding method of rehashing is more than those of the multiplicative congruential method of rehashing.
Hence, the multiplicative congruential method of rehashing is more time-effective than the folding method of rehashing.

Moreover, the average number of comparisons and CPU time for searches falls drastically when load factor changes from 1 to 0.75. Reducing the load factor further (0.5, 0.25, etc.) shows less change in the average number of comparisons and CPU time.