CS 261P – Data Structures

Lab Assignment Project 1

Hashing Algorithms

Rishabh Saxena | UCInetID: rsaxena3 | StudentID: 72844211

Introduction

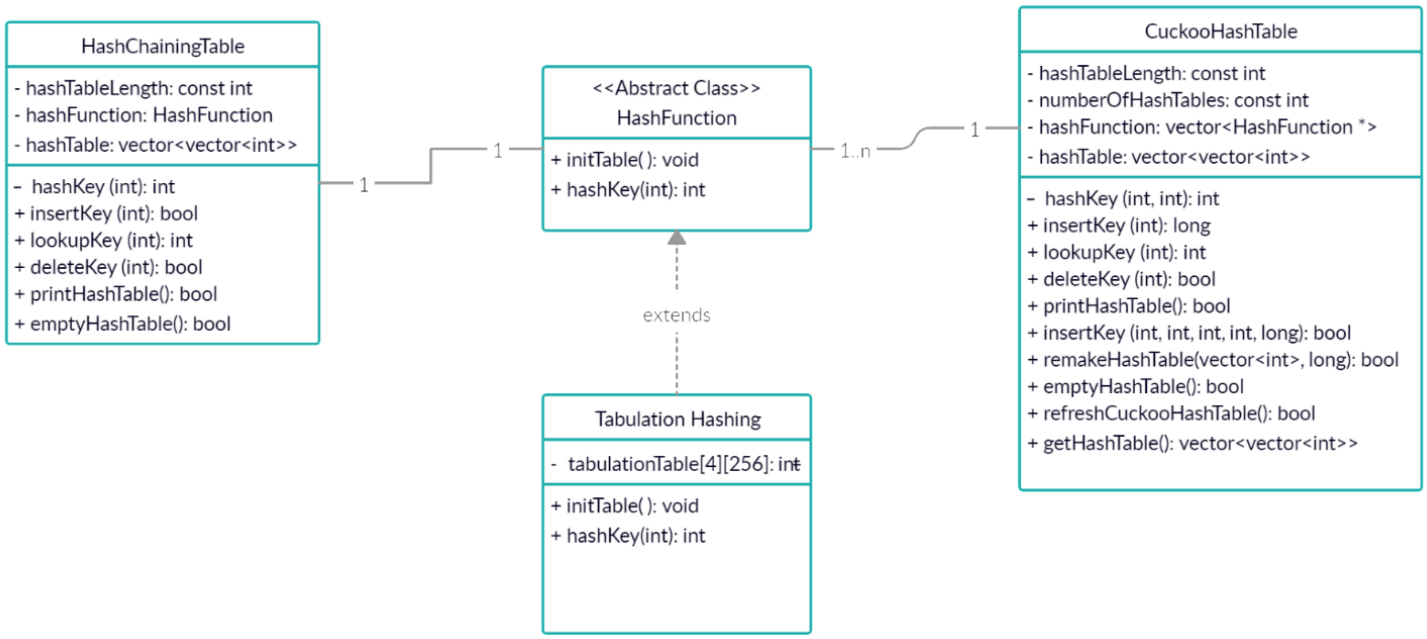
This project was implemented as part of the CS261P Lab Assignment. The goal of this project was to implement Two Hashing Algorithms namely Linear Probing/Chained Hashing and Cuckoo Hashing. I have chosen to implement Chained Hashing along with cuckoo hashing.

Key features of the implemented project:

- **OOP implementation:** The entire project has been implemented using object-oriented features. Classes have been defined for each Hashing algorithms and hash function schemes. The project illustrated key OOP principles like abstraction, data hiding, encapsulation, and polymorphism.
- **Tabulation Hashing:** Both the hashing algorithms use Tabulation hashing to hash the incoming keys of data. This was an interesting hashing algorithm discussed in lectures and was pretty interesting to implement. It has an additional advantage which is very useful for cuckoo hashing.
- **Testing Suites:** I have implemented classes like MainTest.cpp and CuckooHighLoadFactorFailureTest.cpp which have easily changeable parameters and can be used to perform testing on the hash table of various sizes and generate any number of load factors to collect the data on.

Design:

UML Class Diagram showing class relationships



Following are some of the design decisions made during the design phase of the project:

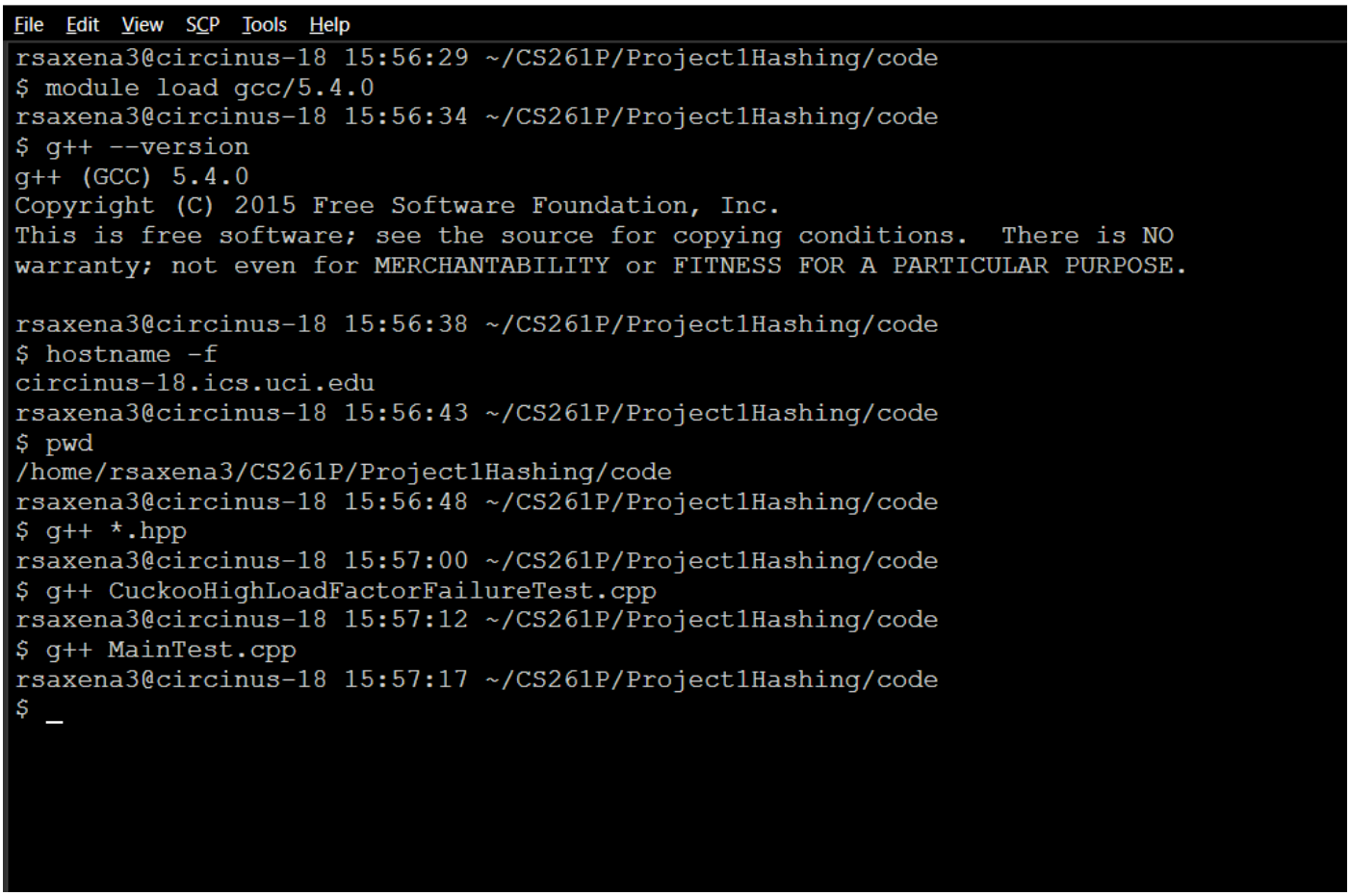
- I am going to analyze worst-case time.
- I have calculated the time taken in the actual clock as well as the number of keys accessed in an insert operation. For Hash chaining, the number of keys accessed is calculated for a subsequent lookup operation.
- The data was randomly generated and the data used is dumped in a CSV file for each test
- The access pattern for the data is inherently random because of the randomness of the keys that are generated.
- I have used the Tabulation Hashing function for both the algorithms. For Cuckoo hashing, it helps in easily generating a new hash function on cycle detection by reinitializing the internal tables with a new set of random values.

Implementation:

If someone wants to tweak the test parameters in the 2 test classes MainTest.cpp and CuckooHighLoadFactorFailureTest.cpp, they will have to edit the #define parameters in the file and recompile the code and execute. The data will be generated in the data directory inside the source code folder.

For the number of alphas provided random values of alpha were generated in the specified range. The insertion was done of only a key (and not a key, value pair). These keys were generated randomly and dumped into an inputDataDump.csv file.

The program was developed entirely in openlab. The following screenshot shows that the project was compiled in openlab using the version of C++ specified in the project description. It also shows the hostname and the present working directory of the project.



During the implementation of Cuckoo Hashing, it was observed that cuckoo Hashing starts performing really poorly after load factor is increased from 50%. After reading on the original paper of Cuckoo hashing, I realized that the original authors always intended cuckoo hashing to be used for a maximum load factor of 50%. Therefore in my analysis, I have only considered load factors less than 50%. I did some additional analysis on the performance of hashing over 50% load factor (CuckooHighLoadFactorFailureTest.cpp).

Additional analysis details:

Hash Table Size: 100
Number of Cuckoo Hash Tables: 2
Alphas Max: 0.9
Alpha Increment: 0.025

alpha	Keys to insert	timeTaken (micro sec)	accessCount
0.5	100	0.026105	190
0.525	105	0.024247	171
0.55	110	0.107286	200
0.575	115	0.03342	246
0.6	120	0.124055	308
0.625	125	0.224644	288
0.65	130	0.13973	357
0.675	135	1.25791	388
0.7	140	0.913779	329
0.725	145	3.32824	595
0.75	150	5.37403	725
0.775	155	71.6765	691
0.8	160	350.198	790
0.825	165	2830.89	908
0.85	170	78501.4	1242

It can be observed that both time taken and access count both increase exponentially as the load factor is reaching 1. Even for such a hash table of 2 tables of size 100 each, the program **COULD NOT** hash the keys for a load factor of 0.875 or greater.

Thus for bigger sized tables in my analysis, I have only considered load factors less than equal to 0.5 (50% filled).

Analysis:

Colab link of analysis

https://colab.research.google.com/drive/1rCDrt1v3hkPP1uRrFZPglYArQmJ4KPAi

Hash Chaining testing paramters:

Hash Table Size: 10000
Alphas generated: 1000
Max Value of Alpha: 5 (For hash chaining alpha can go beyond 1)

Cuckoo Hashing Testing parameters:

Hash Table Size: 10000
Number of Cuckoo Hash Tables: 2
Alphas generated: 1000

I have first plotted 4 graphs that scatter plots in the Cartesian plane to visualize the distribution of data collected.

1. Load Factor vs Time Taken for Hash Chaining
2. Load Factor vs Keys Accessed for Hash Chaining
3. Load Factor vs Time Taken for Cuckoo Hashing
4. Load Factor vs Keys Accessed for Cuckoo Hashing

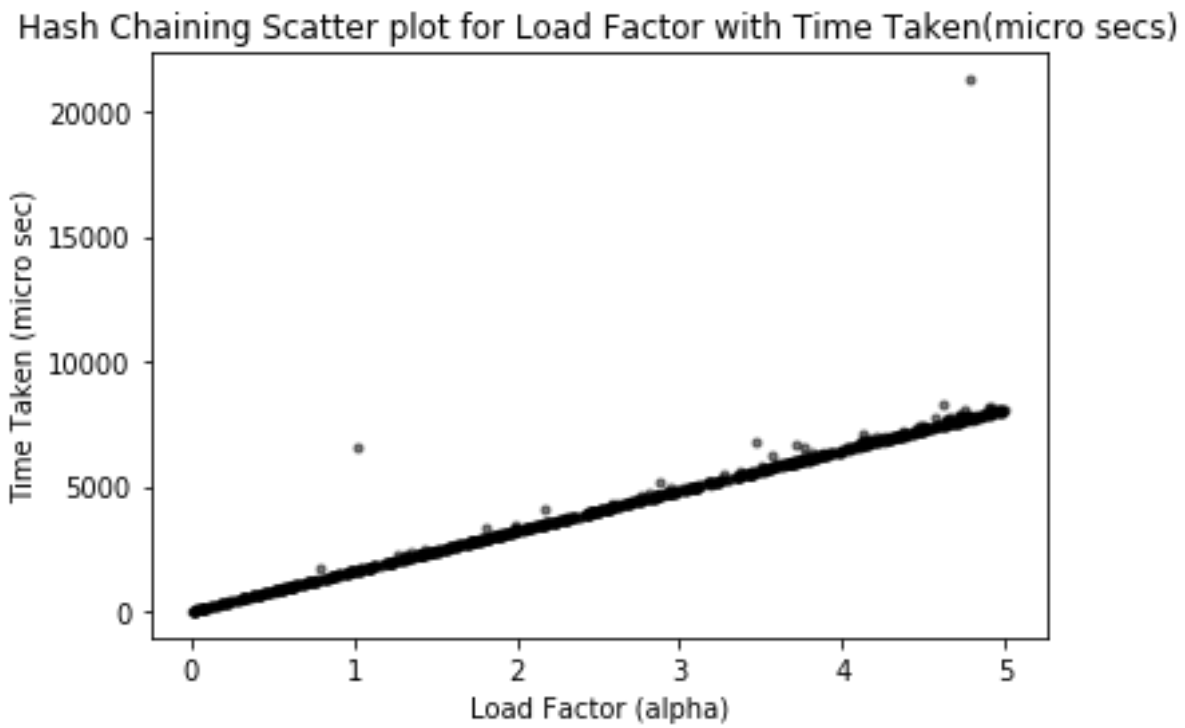
This is followed by the log-log plots for the 2 hashing algorithms. This will again be a set of 4 graphs.

Then I have tabulated the slopes of the log-log plot, which represents the order of exponential relationship between the load factor and performance measure.

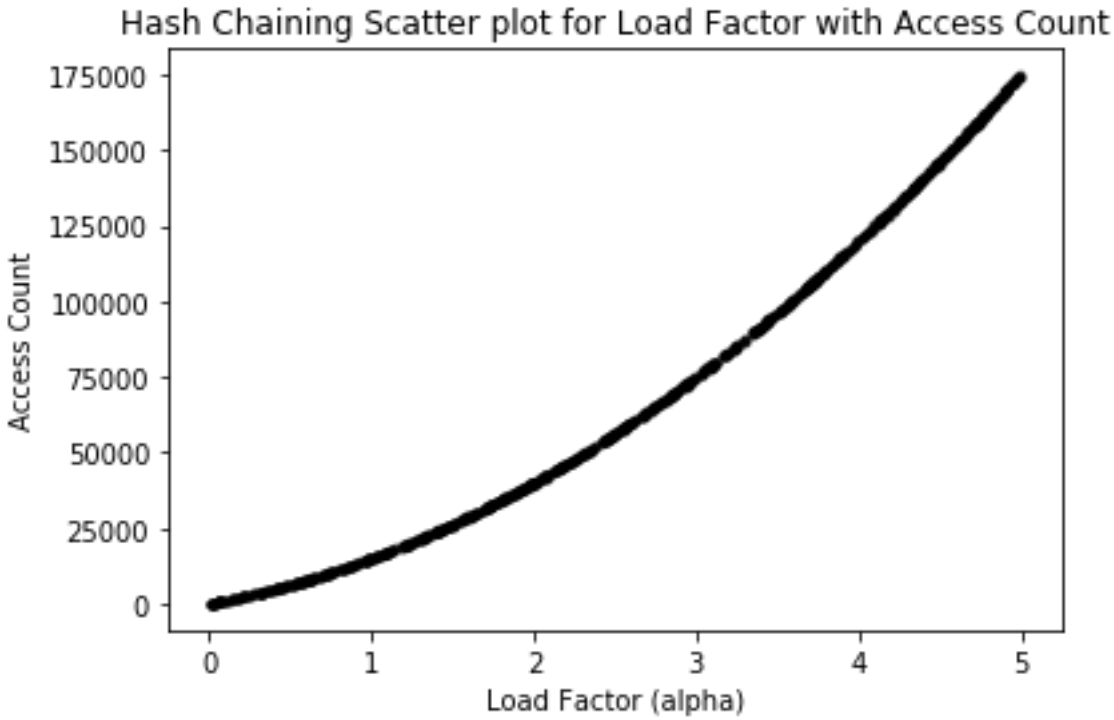
Then I have visually analyzed the 4 lines obtained and drawn some inferences from it in the later sections.

(please move forward to the next page)

Hash Chaining

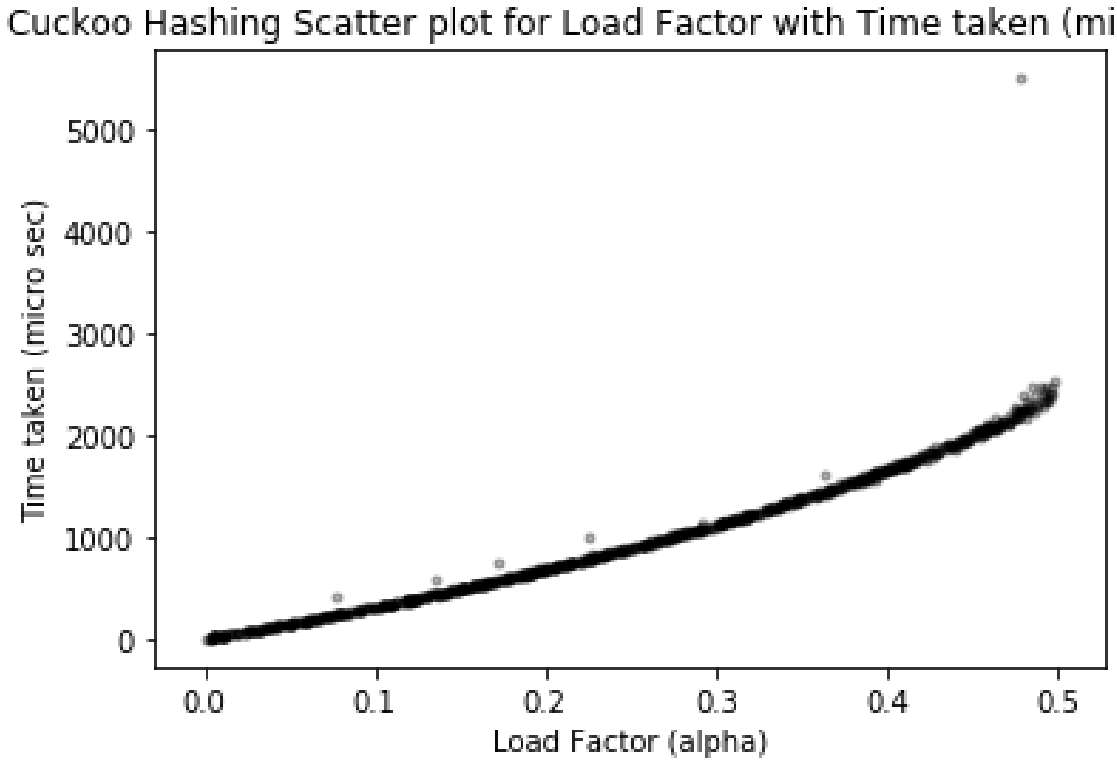


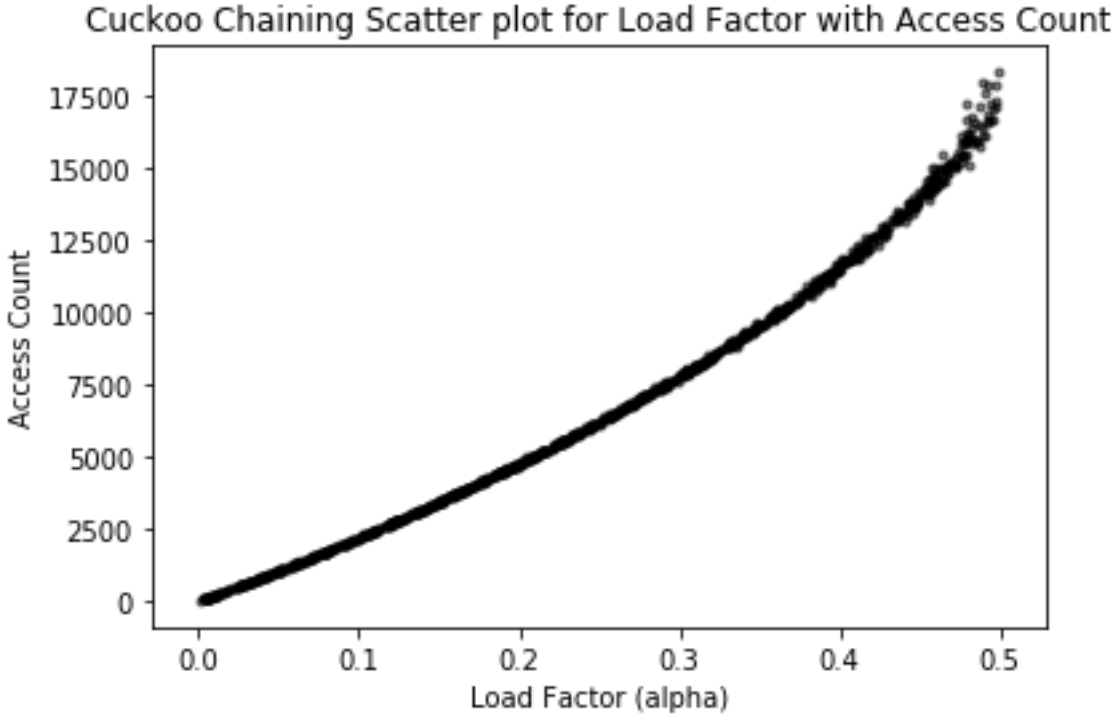
The load factor is varied between 0 and 5 for hash Chaining. It can be seen that time taken for insertion is linearly dependent on load factor. This is because the implementation is using vectors and adding value at the end of a vector is an O(1) operation. Therefore insertion becomes O(1). However, lookup cannot be done in O(1) as illustrated in next graph.



This graph very easily illustrates how lookup operation time increases as load factor is increased becase the chain sizes start becoming very big, the lookup performance degrades.

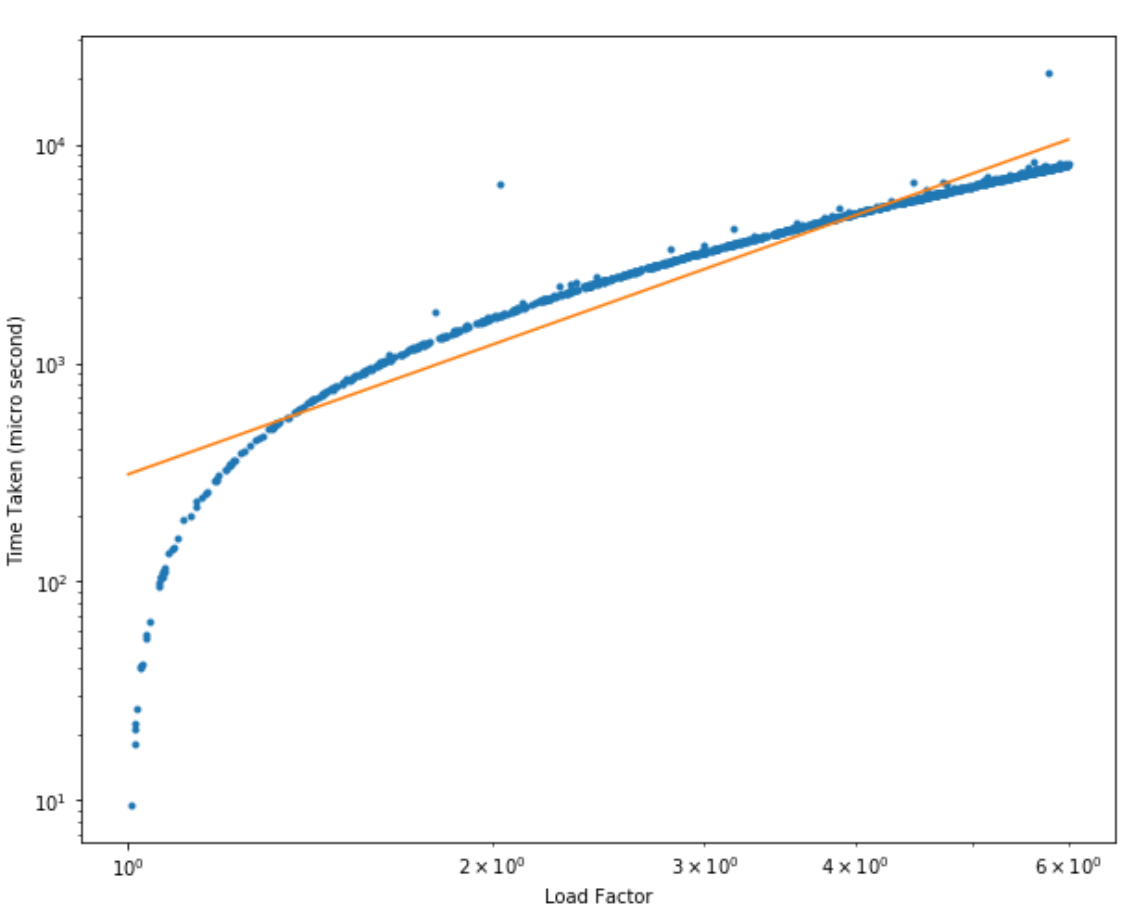
Cuckoo Hashing

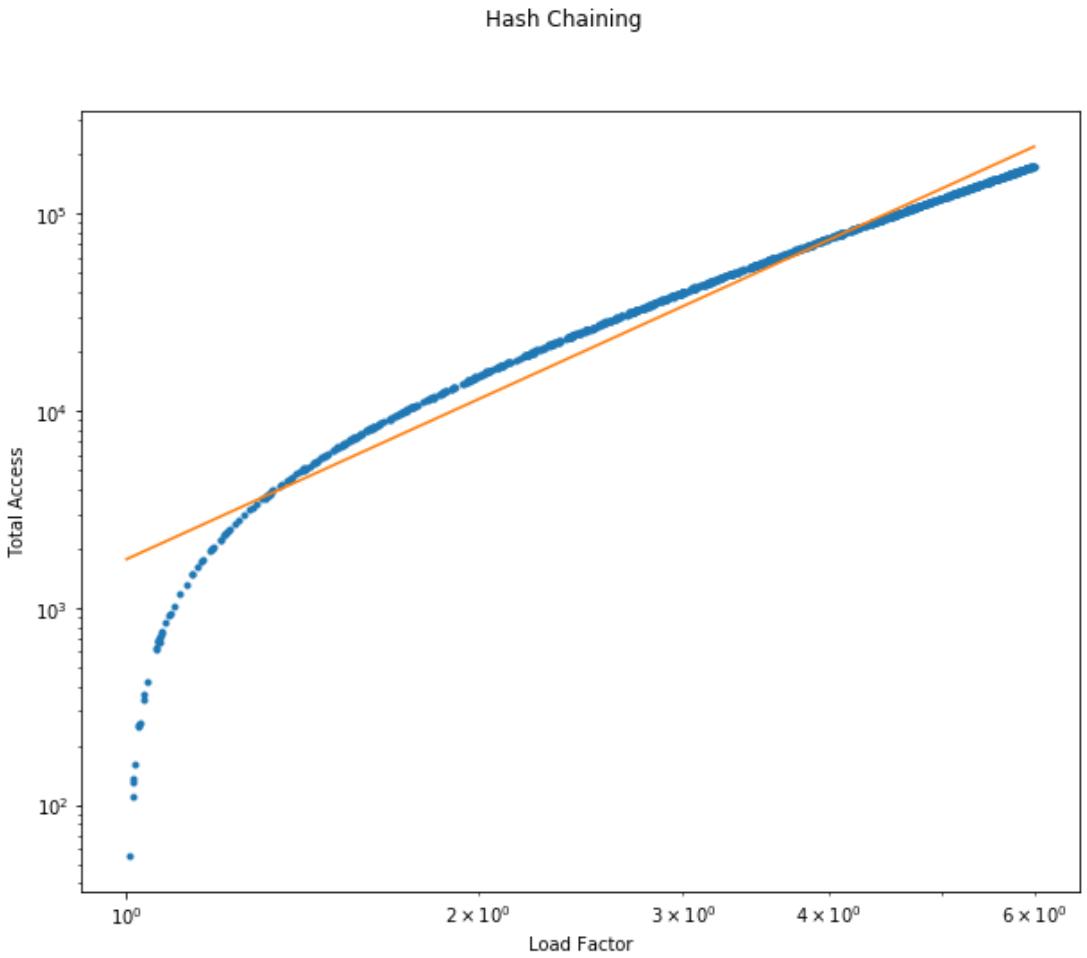




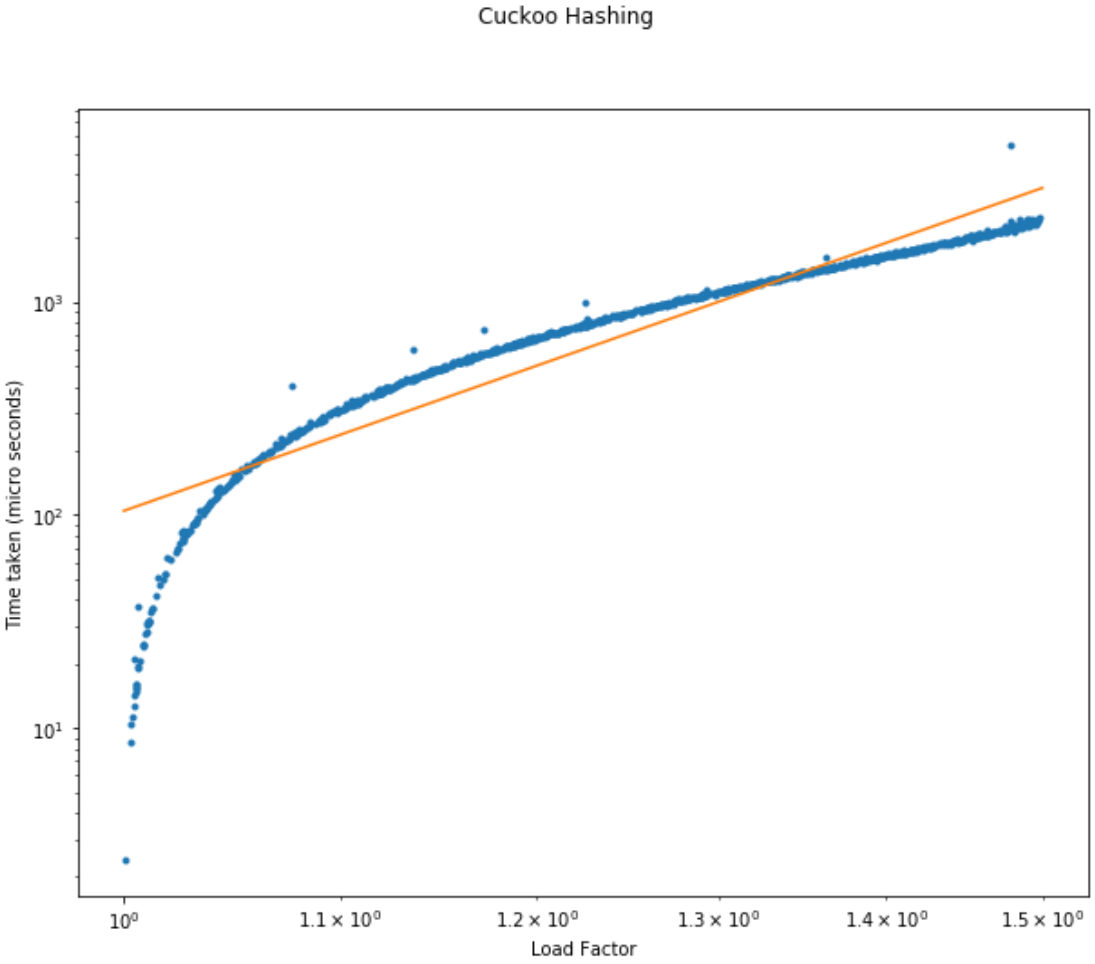
After observing both graphs we can see that the time taken and kets accessed increases at a steep rate as alpha increases. Therefore the relation is not linear.

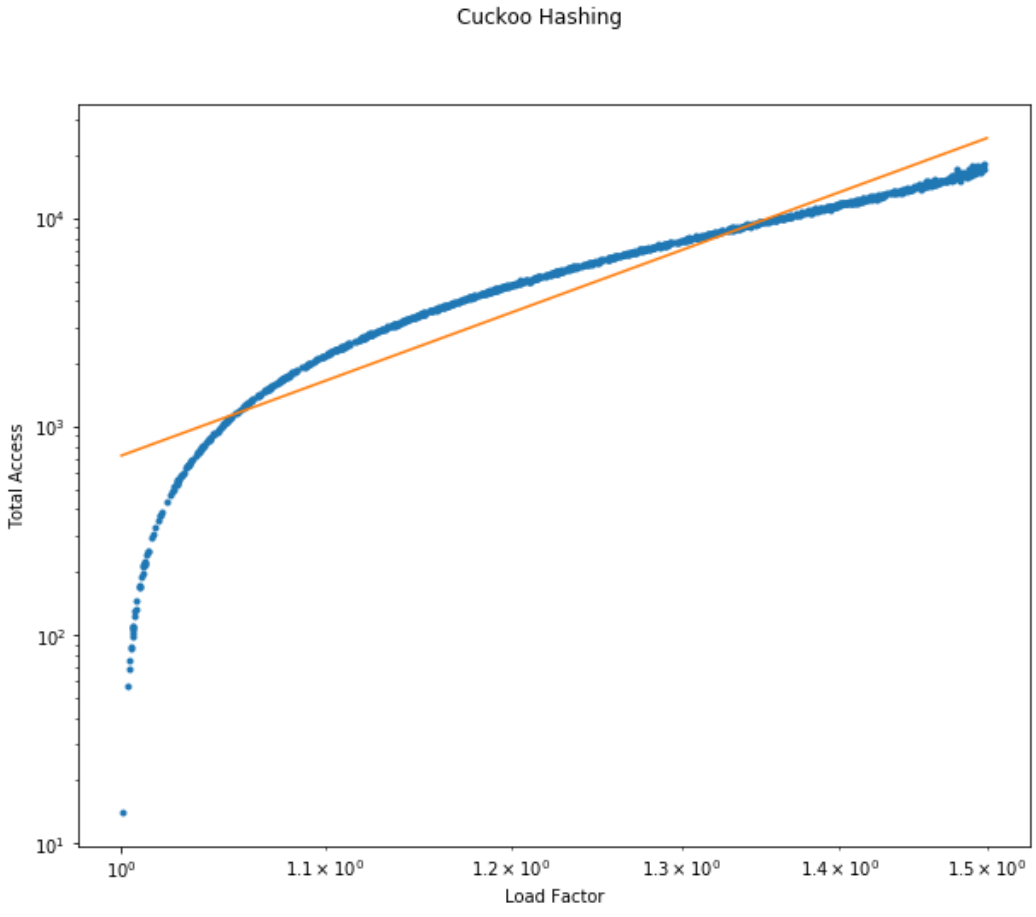
Log-Log Plots Hash Chaining





Log-Log Plots Cuckoo Hashing





Observations

By definition of log-log plot:

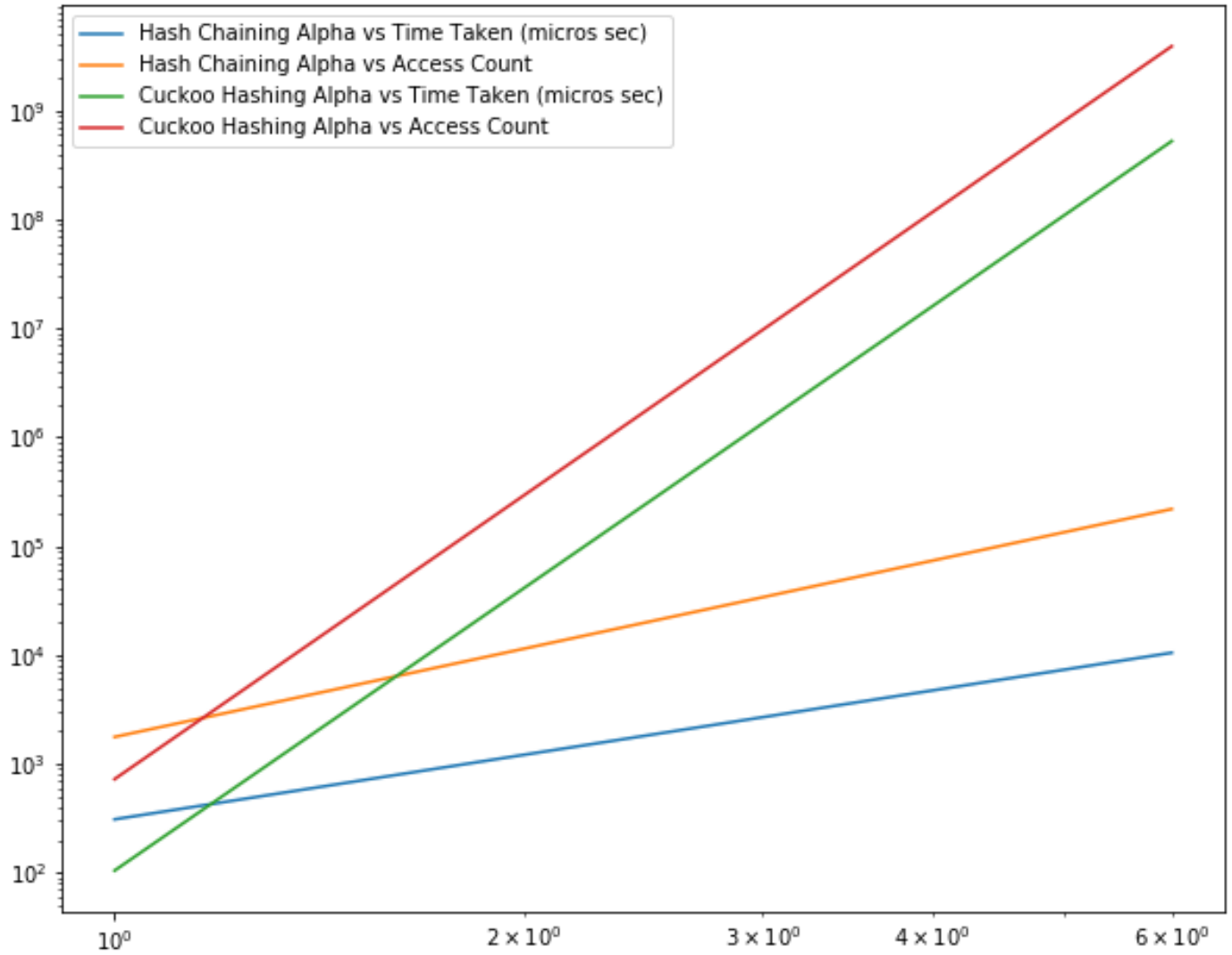
$$\log y = k \log x + \log a$$

i.e. $y = a.x^k$

Therefore the slope of the graph gives us the exponent of the monomial equation and tells us about the growth of our algorithm as the load factor increases.

Algorithm	Performance Measure	Slope	Intercept
Hash Chaining	Time taken (micro sec)	1.9666	2.4908
Hash Chaining	Access count	2.6909	3.2474
Cuckoo Hashing	Time taken (micro sec)	8.6128	2.0198
Cuckoo Hashing	Access count	8.6527	2.8602

Plotting this information in a graph:



Inference

From the previous graph the following things can be interpreted:

- Cuckoo has a very steep slope compared to Hash Chaining, therefore as load factor increases it starts performing very poorly for insert operation compared to hash chaining.
- However, Cukoo hashing has a lower y-intercept compared to Hash Chaining. Observing the point of intersection of the 2 lines, it can be easily seen that despite the steep slope, cuckoo hashing performs better for smaller load factors (less than 50%). Therefore we can conclude that for insert operations on sparse hash table, Cuckoo hashing performs better than hash chaining
- It can also be seen that the lines obtained for the same algorithm with different performance measures have similar slopes. Therefore we can conclude that the inferences drawn are independent of the performance measure used.

Conclusion

In conclusion, we can say that Cuckoo Hashing performs better when the load factor is smaller and starts degrading very poorly compared to Hash Chaining when load factor increases. There is a performance drop even in Hash chaining for high load factors but it is of a much smaller order.