Design of Algorithms: Assignment 2 Report

Part 4: Load Factor and Collisions

A Jupyter Notebook was used to process the results, with Python code and the Pandas and Matplotlib libraries used to produce the visualizations and correlation values. The utility program cmdgen.c was used to generate the sequences of commands for each number of inserts, shown in **Figure 1**.

As shown in **Figure 1**, the number of inserts for each input to linear.c was increased by 200 for each test. As a demonstration of the relationship between inserts and collisions, I calculated their Pearson Correlation to be 0.953, showing a near linear relationship (**Figure 2**). This gives a reference point for the experiments on the relationship between the load factor, number of collisions and length of average probe sequence, since the number of inserts is the independent variable in this case.

The brackets either side of **Figure 1** demonstrate the periodic nature of load factor, collisions and length of average probe sequence values. The table is split by the total number of slots in the hash table, and shows that every time the hash table doubles in size, the number of collisions, load factor and length of average probe sequence decrease. We can therefore see how load factor follows the same periodic progression of values as number of collisions and length of average probe sequence.

	Inserts	Collisions	Total Load	Load Factor (%)	Length of Avg Probe Sequence	Number of Slots
0	200	88	200	78.125	4.863636	256
1	400	159	399	79.930	4.094340	512
2	600	195	596	58.203	2.984615	1024
3	800	291	788	76.953	3.786942	1024
4	1000	427	982	95.898	18.482436	1024
5	1200	334	1176	57.422	2.464072	2048
6	1400	476	1369	66.846	3.232101	2048
7	1600	610	1561	76.221	4.631148	2048
8	1800	737	1753	85.596	8.531886	2048
9	2000	857	1934	94.434	10.653442	2048
10	2200	495	2124	51.855	2.024242	4096
11	2400	585	2292	55.957	2.194872	4096
12	2600	719	2499	61.011	2.499305	4096
13	2800	816	2691	65.698	2.726716	4096
14	3000	977	2861	69.849	2.968270	4096
15	3200	1085	3068	74.902	3.608295	4096
16	3400	1196	3211	78.394	4.075251	4096
17	3600	1358	3415	83.374	5.494109	4096
18	3800	1508	3586	87.549	7.004642	4096
19	4000	1640	3753	91.626	11.200610	4096

Figure 1 – Table showing statistics of experiments with varying number of inserts

	Inserts	Collisions
Inserts	1.000000	0.952905
Collisions	0.952905	1.000000

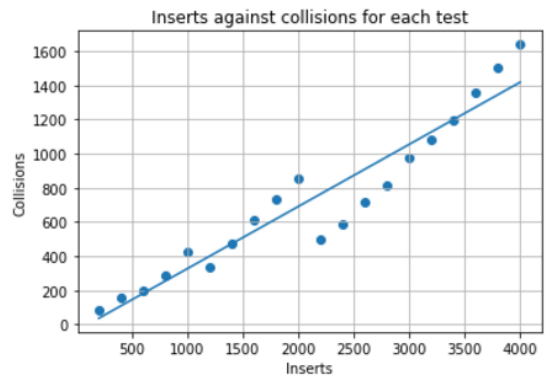

Figure 2 – Inserts vs. Collisions

	Load Factor (%)	Collisions
Load Factor (%)	1.000000	0.407358
Collisions	0.407358	1.000000

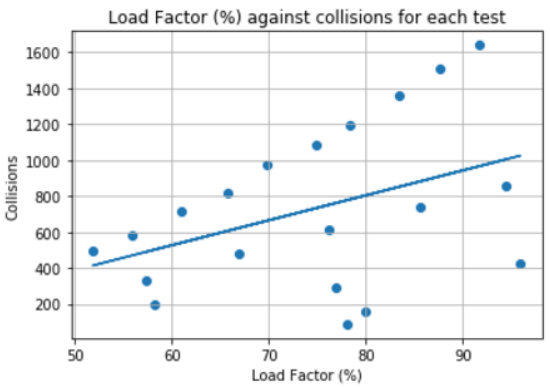

Figure 3 – Load Factor vs. Collisions

Figure 3 shows the relationship between the hash table's load factor and the number of collisions. The Pearson Correlation of 0.407 is indicative of a moderate correlation, which could be explained by the previous analysis showing the periodicity of the collected statistics. While the effect that doubling the hash table size has on load factor and collisions cannot be explicitly seen in **Figure 3**, the positive trends of three visible groups of values are indicative of how load factor interacts with collision number, with regards to the growing hash table size. Since the raw figures of collisions are relative to the number of inserts, which is not reflected in **Figure 3**, the three groups are visible.

The strong correlation between load factor and length of average probe sequence is shown in **Figure 4**, with a Pearson Correlation of 0.812. While the line of best fit does not show this, there appears to be an exponential increase in length of average probe sequence as load factor increases. When load factor approaches 100, the hash table doubles in size, resulting in the length of average probe sequence decreases, along with load factor.

	Load Factor (%)	Length of Avg Probe Sequence
Load Factor (%)	1.0000	0.8123
Length of Avg Probe Sequence	0.8123	1.0000

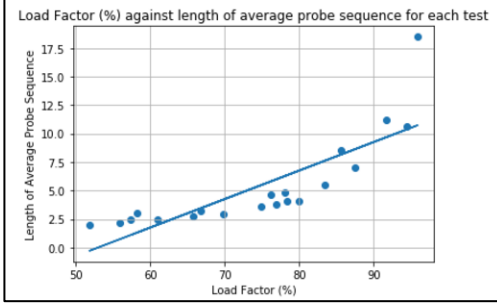

Figure 4 – Load Factor vs. Length of Avg Probe Sequence

Part 5: Keys Per Bucket

The same method as in part 4 was used to produce the visualizations. To generate the input file, cmdgen.c was used to create an input file with 5,000 inserts and 5,000 lookups, and this file was run with xtndbln.c with input bucket sizes ranging from 4 (default bucket size) to 1004 buckets. The bucket sizes were incremented by 100 buckets each time.

By using bucket size as the independent variable in this case, I could experiment with the performance of my single key extendible hash table (via CPU Time) to show its relationship with the number of keys per bucket.

	Bucketsize	CPU Time
Bucketsize	1.000000	-0.541366
CPU Time	-0.541366	1.000000

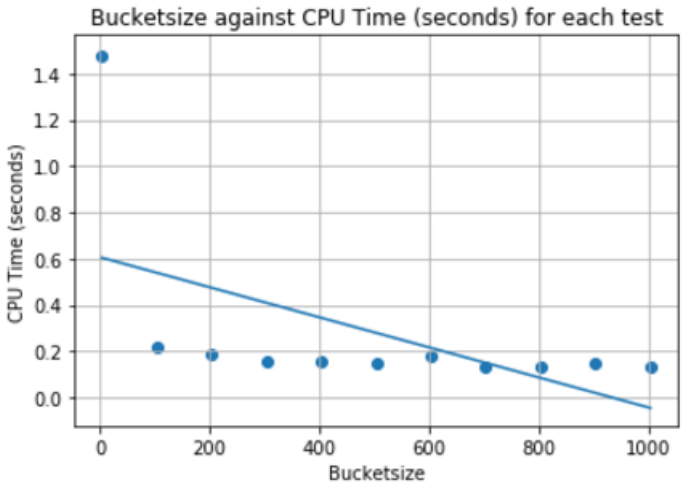

Figure 5 – Bucket size against Hash Table Performance (CPU Time)

The trend visible in **Figure 5** seems to illustrate an exponential increase in CPU Time as the bucket size decreases. The massive drop in CPU Time from 4 to 104 buckets shows that a higher bucket size results in a better performance of the hash table, as CPU Time decreases. However, it appears that the decrease in CPU Time approaches an asymptote, and the CPU Time does not decrease significantly beyond a bucket size of 204. In fact, the CPU Time remains relatively stagnant after that point, fluctuating around 0.135 seconds.

The Pearson Correlation of -0.541 is indicative of a moderate to strong negative correlation between CPU Time and the keys per bucket. As a result of this experimentation, we can say that there definitely is a relationship between the number of keys per bucket (bucket size) and the performance of the hash table, however the benefit of increasing bucket size rapidly decreases after around 104 keys per bucket.