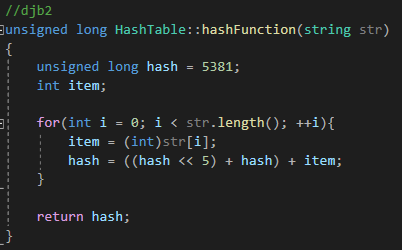
Hashing Function Analysis Report

# Introduction

For this assignment, we were tasked with implementing 3 hashing functions using separate chaining. The goal of this assignment was to judge the quality of hashing functions based on the number of collisions each function had. The number of collisions was determined by the spread of the chain length data for each function, namely the standard deviation. Essentially, the lower the standard deviation of the function’s chain lengths, the better the function was at storing data in an efficient manner. The hashing functions I used for this assignment were DJB2, SDBM, and Lose-Lose.

# DJB2

Figure I: DJB2 Code

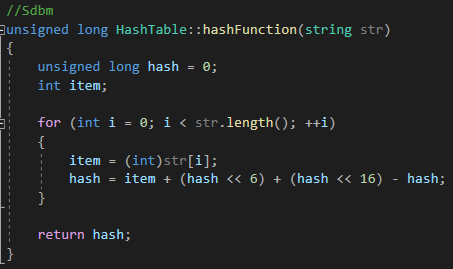


DJB2 is a well-known hashing functions for strings. It utilizes bit manipulation and a set prime number to create the hashing value that used to calculate the index to be used in the table. In my version of the code, I calculated the hashing value as an unsigned long, modulated it against the table size, and cast that value to an int variable.

Given the code requirements and concept behind the algorithm, I predicted that DJB2 would perform decently at distributing the values across the table, and that the number of collisions would be less than Lose-Lose. I also hypothesized that that the number of collisions would be greater than that of SDBM.

# SDBM

Figure II: SDBM Code

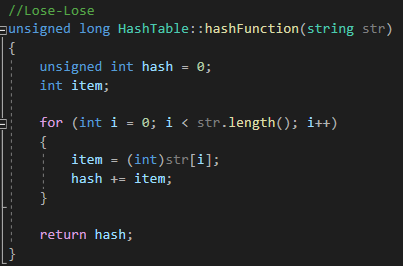


SDBM is another known string hashing function. It utilizes more extensive bitwise operations that DJB2. This algorithm differs from DJB2 further in that hash is not set to a prime number, and it is called more often in the function operation inside of the for loop. In theory, this operation makes the algorithm more sensitive to differences in strings, meaning that it should have good avalanching.

Therefore, I theorized that the spread of data along the table for SDBM would be greater than that of DJB2 and Lose-Lose. Furthermore, I hypothesized that SDBM would have less collisions than both DJB2 and Lose-Lose.

# Lose-Lose

Figure III: Lose-Lose Code



Lose-Lose is a simple hashing function to implement. It is characterized by a very straightforward hashing operation, where the hash value is simple equal to the previous hash value plus the ASCII value of the char at position ‘i’ of the string. However, this results in the hash value not being very different for different strings, meaning that is has bad avalanching. Thus, the distribution of data across the table for Lose-Lose is likely to be concentrated at a certain point or range of points, rather than the whole table.

Given the nature of its operation function, I predicted that Lose-Lose would have the lowest spread of data of the 3 hashing functions and would thus have more collisions.

# Analysis and Conclusion

Presented in Figures IV-VI are the histograms for each hash function, with the chain length data per index coralled into frequency of ocurrence for each chain length.

Figure IV: DJB2 Chain Length Histogram Figure V: SDBM Chain Length Histogram

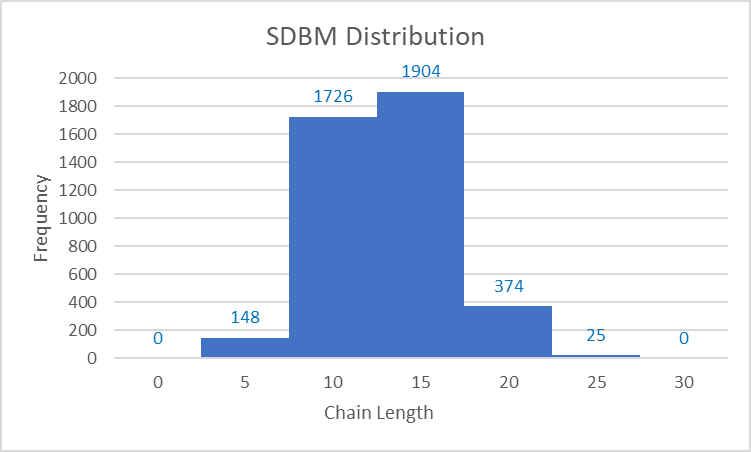
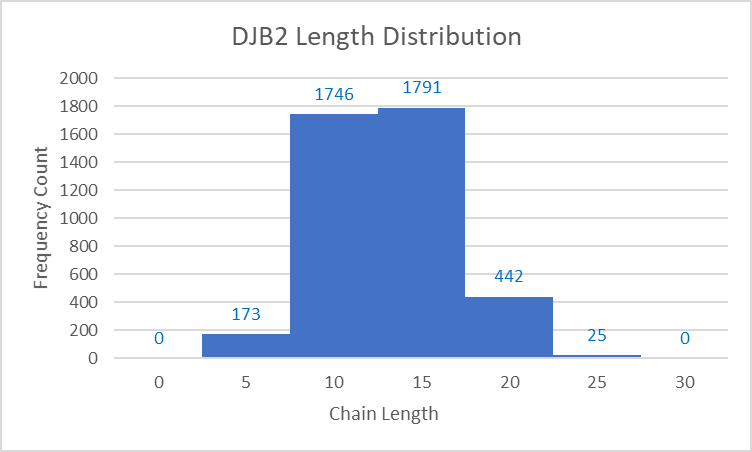
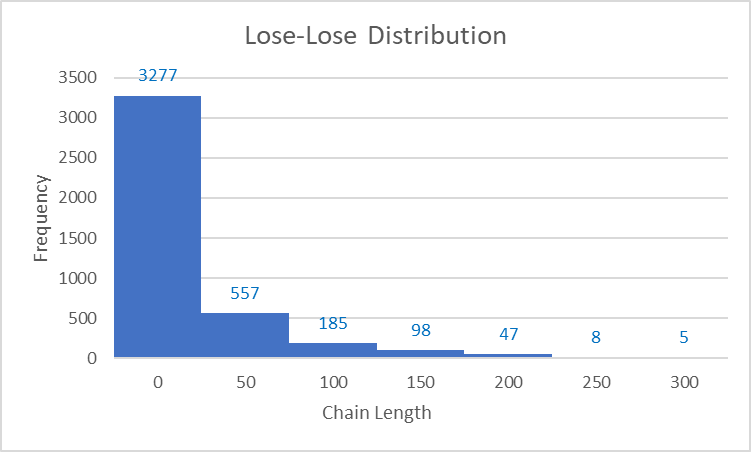


Figure VI: Lose-Lose Chain Length Distribution



As can be seen from Figures IV and V, the distributions of the DJB2 and SDBM histograms are approximately normal and centered around chain lengths of 10-20, meaning that the data was distributed well across the table. Lose-Lose, as depicted in Figure VI, skews right, with chain length values reaching even into the 250-300 length range. Thus, the figure indicates that Lose-Lose did not perform well in organizing the data across the table.

The standard deviations of the chains lengths for each function were also calculated as part of the analysis. Low standard deviation for the chain length indicates that the hashing function was effective in spreading the data out evenly. The standard deviation for Lose-Lose was approximately 32.036, further demonstrating its ineffectiveness as a hashing algorithm. The standard deviations for DJB2 and SDBM were 3.496 and 3.376, respectively. Overall, since SDBM had the lowest standard deviation, albeit by a marginal amount, we can safely conclude that SDBM was the more effective hashing algorithm.