FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

Tınaztepe Yerleşkesi, Buca-Kaynaklar, Dokuz Eylül Üniversitesi, İZMİR, TÜRKİYE

CME2201, Data Structures, Assignment 1 Report
Date:12.08.2022
Prepared By: Sefa Çelik

Performance monitoring:

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP	**9,590,245**	**11423ms**	**112989.8ns**	**28800ns**	**2156700ns**
		DH	**1,126,561**	**10940ms**	**61691.8ns**	**6600ns**	**676600ns**
	PAF	LP	**644**	**9657ms**	**734.6ns**	**200ns**	**21800ns**
		DH	**485**	**9818ms**	**772.2ns**	**100ns**	**17500ns**
α=80%	SSF	LP	**9,590,245**	**10948ms**	**108890.5ns**	**29000ns**	**921100ns**
		DH	**1,126,561**	**10619ms**	**63637.2ns**	**7300ns**	**772500ns**
	PAF	LP	**2496**	**10089ms**	**918.7ns**	**300ns**	**16700ns**
		DH	**1205**	**10222ms**	**1040.7ns**	**200ns**	**23200ns**

- Looking at the table, it is seen that between load factor 0.5 and load factor 0.8, load factor 0.5 is much better in collision count and search time, and it is seen that indexing times are generally very close to each other.

- Looking at the table, it is seen that between SSF and PAF, PAF produces much better results than SSF in all areas.

- Looking at the table, it is seen that between linear probing and double hashing, double hashing produces more successful results in every subject.

If we come to a general conclusion, the first thing we need to be aware of is that our application is a search engine application. For this reason, the first comparison factor to be looked at should be search time. In the second place, since the data structure we use is HashTable, we should measure the success in hashing. For this, we have to compare the collision numbers.

When we look at these priorities, we see that our preferred option in this application is load factor 0.5, PAF and DH because it provides a much lower searching time and our collision numbers are much lower than others.

The reasons why Load Factor 0.8, SSF, Linear Probing were put into the background:

-Load factor 0.8 is inefficient because it compresses data into a smaller array. This causes more collisions to occur.

- The reason why SSF is inefficient is that the number of collisions, which is caused by the high probability of the keys they produce to be the same with each other, reaches very high numbers.

- Linear probing is inefficient because it shifts one unit for each key. For this reason, the keys are positioned close to each other and the number of collisions increases.

Codeing Part:

We can describe the code under 4 main classes.

The first class is the valueNode class. This class contains one integer and one dictionary. This class is created for the valus of the hashTable.

The second class is the FileReader class. In this class, the given sport file has been converted to a dictionary named counter. The word that comes in the key part of this dictionary is kept and another dictionary is kept in the value part. In the other dictionary created in the Value section, the number of words in which txt was kept. Later, this counter dictionary was converted to hashTable. While reading these files, another dictionary was created that keeps the number of words in each txt. This txtSize dictionary was used to find the most logical txt in the future.

The third class is the MaxPercantageSearcher class. This class first calculates the highest value in a dictionary and then the 3 highest values. It then converts them to percentages using the txtSize dictionary and returns the 3 txts with the highest percentage.

The fourth class is the SearchEngine class, which is the main class. In this class, it takes 3 words as input and searches and sends the most logical txt to the screen with its ratio.