**1) Performance Matrix**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Load Factor** | **Hash Function** | **Collision Handling** | **Collision Count** | **Indexing Time** | **Avg. Search Time** | **Min. Search Time** | **Max. Search Time** |
| α=50% | SSF | LP | 8255207 | 2877693000 ns | 35882 ns | 200 ns | 102500 ns |
| DH | 953996 | 539911800 ns | 6744 ns | 200 ns | 77100 ns |
| PAF | LP | 378 | 188350300 ns | 276 ns | 100 ns | 11900 ns |
| DH | 279 | 185208000 ns | 248 ns | 100 ns | 10700 ns |
| α=80% | SSF | LP | 8255207 | 2565118200 ns | 37284 ns | 300 ns | 122100 ns |
| DH | 953996 | 519738400 ns | 7350 ns | 200 ns | 99600 ns |
| PAF | LP | 1500 | 184894800 ns | 324 ns | 100 ns | 19600 ns |
| DH | 897 | 183203300 ns | 309 ns | 100 ns | 17100 ns |

**Note: The table was filled in by reading the 511 text file, not removing the stop words, and ignoring overflows in the PAF function (the Horner Rule was not used).**

In the table, we examined 3 different conditions that we used when creating our hash table and 8 different combinations of these three different conditions. These three conditions are "Load Factor", "Hash Function" and "Collision Handling Techniques".

Firstly, if we examine the change of load factor; In general, as the load factor increases, we observe that the number of collisions remains the same or increases under the same conditions, which shows that the load factor and the number of collisions are directly proportional. As the load factor increases, the occupancy rate of the table also increases. This causes hash table entries to be more congested. The fact that the entries are more congested causes the entries to be more difficult to find and the number of collisions to increase in cases such as adding entries to the hash table or searching. If we examine the indexing times according to the load factor, we see that the times are very close to each other under the same conditions. As the load factor increases, it causes more collisions in the entries ​​while adding to the hash table, but since the number of rehashes will decrease, so indexing times are close to each other under the same conditions. Since the number of rehashes has a more effect on indexing time, when the load factor increases, indexing time decreases slightly under the same conditions. If we look at the effect of load factor on search times, we observe that max search time, min search time and average search time increase in general as the load factor increases. The most important condition when searching is that the number of collisions is low. As the load factor increases, the number of collisions also increases, so the search times also increase.

Secondly, if we examine the variation of hash function; We have two different hash functions, these are "simple summation function" (SSF) and the "polynomial accumulation function" (PAF). SSF generates numbers that are close to each other and smaller. This causes the keys to be produced close to each other. PAF, on the other hand, generates numbers that are very large and not close to each other, and the keys become very different numbers from each other. Generating various keys from each other ensures that the hash table indexes to which the entries must be added are different from each other. This ensures that the indexing time, collision number, average search time, max search time and min search time of the PAF are much less than the SSF under the same conditions.

Thirdly, if we examine the collision handling techniques; We have two different collision handling techniques, linear probing and double hashing. In these techniques, different ways are used when indexing keys in the hash table. In linear probing, if the index corresponding to the key is full, the appropriate index is tried to be found by increasing the index one by one. In a double hash, if the index corresponding to the key is full, the index is incremented according to a key-dependent function to try to find the appropriate index. When we examine the table, we see that all the values ​​in the table decrease when we choose the Double Hash technique. Searching for the appropriate index with the method specific to the keys ensures that the keys are not accumulate in indexes close to each other and are easier to find. This reduces the number of collisions, indexing time, average search time, max search time and min search time under the same conditions.

As a result, when we examine all these cases, the best case for both collision count and search time is to use the **Polynomial Accumulation Function(PAF)** with the **Double Hashing** technique and set the **load factor to 50%** .

**2) The Algorithm of My Hash Table**

In this assignment, we had to assign specific two parameter to each word as key and value to fill in the hash table. I obtained the integer values of the words in the text files according to the polynomial accumulation function and assigned these values as the key of the words. I created a new Array List for each different word and assigned these Array Lists as the value of the word. I defined the nodes of these Array Lists as objects of the List Entry class that I created. I have created 4 different variables in the List Entry class as "word", "count", "totalWordCount" and "fileName". I have stored which word the key belongs to in the "word" variable, how many times the word appears in the current read text file in the "count" variable, how many words are in the current read text file in the "totalWordCount" variable, and the name of the current read text file in the "fileName" variable. In this way, when a new addition is made to the hash table, if the index where the key corresponds is full, I have it checked whether the “word” variable in the entries of the Array List that forms the value of that index and the newly added object are the same word. If the words are the same, I added the new object to the end of the Array List or if the same words are in the same text file, I have increased the number of counts in the node where the text file is located. If the words are different, I had the key get a new index value with the double hash function and rechecked these check steps until found the same word or encountered an null index. If null index was encountered, I create a new array list as a value and assign these key and value to the index that is result of probe function in hash table. Thus, I have stored all the different words in text files in different indexes in my hash table. Also, I have stored all the necessary values for the same words in text files in Array Lists, which I keep as values.

Finally, I found the keys of three different words taken from the user and searched the indexes corresponding to these keys. I searched until the word was found or until a null index was encountered, and if the word was found, I returned the value of the word. Since the returned values ​​are array lists, I created a loop that including all these array lists and checked the “fileName” in the entries of the array lists. In this check, I created an array list for the file names, and if the file names are the same, I summed up the number of searched words contained in these files and added them to the array list by assigning a new count value. If the file names are different, I created a new entry for the file name and added this entry to the array list. After these steps, I checked the array list consisting of file names from beginning to end and returned the name of the text file, which is the maximum ratio of the total number of searched words to the total number of words in the text file, as a result and displayed it on the screen.

**Ozan Kalkan**

**2020510044**