DOKUZ EYLUL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME2201 DATA STRUCTURES
ASSIGNMENT REPORT

INVERTED INDEX BY USING HASH TABLES

by
Rıdvan Özdemir
2017510086

Lecturers
Dr. Zerrin Işık
Ali Cüvitoğlu
Feriştah Dalkılıç
Altuğ Yiğit

IZMIR

15.11.2019

CHAPTER ONE

PROGRESS DESCRIPTION

The aim of the project is to develop a functional hashtable implementation with linear probing and double hashing as abstract data type. This program read some txt files and take the words from them. After the spliting the words and eliminating, words enter the hashcode function that implemented by us and string word convert to integer value. Then, words put into hashtable.

CHAPTER TWO

TASK SUMMARY

2.1. Completed Tasks

Necessary interfaces were implemented.

Abstract data type (abstract class) and generic type were implemented.

AbstractMap, AbstractHashMap, Hashtable according to collision handling were implemented.

DELIMITERS and stop_words txt were read and put into suitable data types.

String words were converted to integer value successfully by using Simple Summation Function(SSF) and Polynomial Accumulation Function(PAF).

Horner method was implemented for using in PAF.

As collision handling, linear probing(LP) and double hashing(DH) were implemented.

All main functions (get, put, resize, remove) were implemented.

Words were inverted into hash table successfully.

1000.txt were read successfully. Minimum, maximum and average search times were found.

2.1. Incomplete Tasks

There is no incompleted task.

2.1. Additional Improvements

Added a functional menu that makes the program easy to use. Also you can add extra word into hashtable if you want. Finally, each folder can be readeble by one by. For example, If you want just read business folder and put it into hashtable and then search a word, you can do this.

CHAPTER THREE

EXPLANATION OF ALGORITHMS

3.1. Algorith and Solution Strategies

First of all I implemented SSF and PAF function to convert to strings. I put english alphabet into an array. To find integer value of words, I splitted them as letters and take it's value in the alphabet from array. After this, I tried convert to some words string to integer. After this operation works successfully, I tried to read txt files and took words from them. After this, I splitted them and eliminated accoridng to stop_words.txt. After all string process operations, I implemented hash table with linear probing and it's all methods. After this, I applied same process for hashtable with double hashing. But my hashtables are not abstract and generic. That's why I implemented entry iterface and abstract classes. Also to keep txr files for words, I added next variable for entries. To keep txt path and txt count for words, I assigned pathTxt variable. Thanks to this variable I read all folder and txt files, and keep them for words. After this, I read 1000.txt file and found search times. Lastly, I added functional menu as additional improvement.

CHAPTER FOUR

PROBLEMS ENCOUNTERED

I encountered infinite loop when resize methods were working in hashtable with double hashing. Because we need prime table size for double hashing. But I did not use table size as as prime number. That’s why I implemented a getPrime function to find a prime number smaller than table size. After this, my problem were solved.

The second problem I encountered was that the program took too long as 15-20 minutes. After a while, I realized, stop_words.txt is reading for every words that put into hashtable. Then, I solved this problem too.

CHAPTER FIVE

CONCLUSION

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP	2.350.118.985	38.778 sn	180,51 microsec	0,3 microsec	1418,8 microsec
		DH	92.800.832	8.293 sn	12,355 microsec	0,3 microsec	104,6 microsec
	PAF	LP	1.751.177	6.354 sn	1,472 microsec	0,2 microsec	67,7 microsec
		DH	1.057.257	6,05 sn	1,456 microsec	0,3 microsec	74,1 microsec
α=80%	SSF	LP	2.627.616.139	41.48 sn	181,62 microsec	0,3 microsec	824,5 microsec
		DH	88.165.513	8.51 sn	14,36 microsec	0,3 microsec	201,8 microsec
	PAF	LP	5.108.321	6.72 sn	1,59 microsec	0,3 microsec	79,4 microsec
		DH	1.606.840	6,37 sn	1,48 microsec	0,3 microsec	83 microsec

Table 1. *Performance matrix*

I completed all expected task of this assignment. I've learned Abstract data types, generic, interface, hashtable and collision handling. This is the first time, I worked on a large data. That's why it was kindly hard for me. After this project I realized, PAF method is faster and stable than SSF method. Because in PAF method, collision probability is lower than SSF. For instance, "cat" and "act" word are the same integer value according to SSF. But according to PAF method, their integer values are different. Other things I realized, Double hashing method has much more less collision count according to linear probing. Because, linear probing looks empty places by one by, if there is a collision. But double hash method is much more complicated and functional. For the load factors, %50 load factor is faster and stable than %80 load factor. Because in the %80 load factor, program will resize when there are more elements in the table according to %50 load factor. Because of this, collision probability in the %80 load factor, is much more than %50 load factor. After all these inferences, I think performance ranking as follows;

%50-DH-PAF > %50-LP-PAF > %80-DH-PAF > %80-LP-PAF > %80-DH-SSF >

%50-DH-SSF > %50-LP-SSF > %80-LP-SSF

REFERENCES

https://www.geeksforgeeks.org/horners-method-polynomial-evaluation/

https://www.javatpoint.com/prime-number-program-in-java

https://www.geeksforgeeks.org/abstract-data-types/

Data Structures and algorithms in Java, Sixth Edition, GoodRich MT, Tamassia R, Goldwasser MH, Wiley, 2015 page.402-422.