FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
Tınaztepe Yerleşkesi, Buca-Kaynaklar, Dokuz Eylül Üniversitesi, İZMİR, TÜRKİYE
08.12.2022
CME 2201 1. Homework Report: Text Based Basic Search Engine
Prepared by: Yusuf Gassaloğlu

1-) Introduction: Developing a simple text-based search engine. The program operates on text files that are kept in the directory. The goal is implementing a **hashing algorithm** in Java to index words of documents given in sports file and read these documents, split it word by word, and index each word to the hash table according to rules. After insertion of all documents, the user will query for a sentence that contains 3 words splitted by a single space. The algorithm must bring most relevant document for the query written by the user.

2-) Project Description: To specify an index corresponding to given string key, firstly generating an integer hash code by using a special function. Then, resulting hash code must be converted to the range 0 to N-1 using a compression function, such as modulus operator (N is the size of hash table). There are two different functions and two different collision handlings,

2.1.1-) Simple Summation Function (SSF):
Generate the hash code of a string *s* with the length *n* simply by the following formula:

$$\mathrm{h(s)} = \sum_{k=0}^{n-1} ch_k$$

As an example:

SSF("Ali") = 1 (A) + 12 (l) + 9 (i) = 22

2.1.2-) Polynomial Accumulation Function (PAF):
Generate the hash code by using the following polynomial:

$$\mathrm{h(s)} = ch_0 * z^{n-1} + ch_1 * z^{n-2} + \ldots + ch_{n-2} * z^1 + ch_{n-1} * z^0$$

where ch_0 is the left most character of the string, characters are represented as numbers in 1-26 (case insensitive), and *n* is the length of the string. The constant *z* is usually a prime number. When the *z* value is chosen as 31, the string "car" has the following hash value:

$$h(\mathrm{car}) = 3 * 31^2 + 1 * 31 + 18 * 1 = 2932$$

Horner rule is used for avoiding overflows.

2.2.1-) Linear Probing (LP): Linear probing handles collisions by placing the colliding item in the next available table cell.

2.2.2-) Double Hashing (DH):

Double hashing uses a secondary hash function *d(k)* and handles collisions by placing an item in the first available cell of the series.

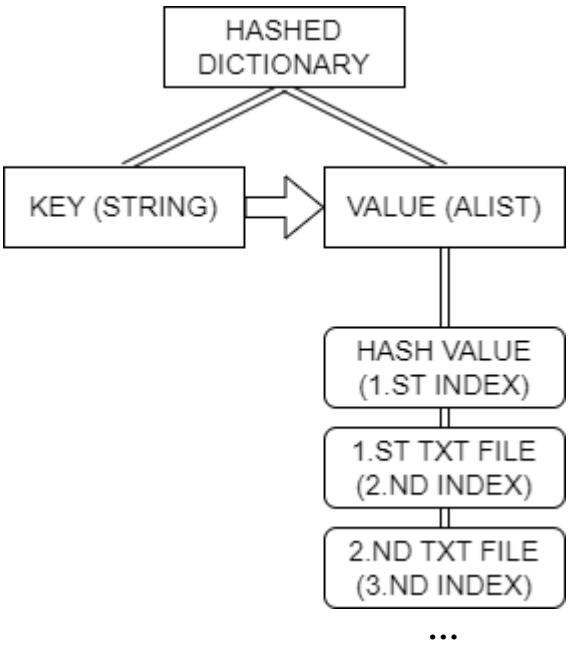
$$d(k) = q - k \bmod q$$

$$h_2(k) = \big(h(k) + j\, d(k)\big) \bmod N$$

where $q < N$ (table size), q is a prime, and $j = 0, 1, \ldots, N-1$.

3-) Algorithms and Solution Strategies:

3.1-) Hash Table Algorithm:



The hashed dictionary algorithm is mentioned on chart. Unique words are stored in hashed dictionary's key. The words hash value and txt files containing the word are stored in the dictionary's value by array list. Hash value of the word is stored in array lists index first and the txt files are begin to stored in arrays remaining indexes.

3.2-) Adding The Word Into The Dictionary:

```
public V add(K key, V value, String HashFunction, String collisionHandling);
```

Add function needs 4 parameters:
'key' an object search key of the new entry, value an object associated with the search key, HashFunction is related hashing function of the word (simple summation function or polynomial accumulation function), collisionHandling is collision handling method of code (linear probing or double hashing)
While adding a word into the table add function controls load ratio. Compares with max load factor if the ratio is bigger than load factor the table is rehashed. And new size will be prime

number *(>oldSize*2)*.

3.3-) Finding Most Relevant File:

The main logic is:

For all words and related files,
RELATED WORD COUNT / TOTAL WORD COUNT IN A FILE

We can find the distribution and density with this formula. First find related words ratios then multiply with each other. Biggest value is most relevant file.

4-) Results (Reading 100 Files):

Load Factor	Hash Function	Collision Handling	Collision Count	Indexing Time	Avg. Search Time	Min. Search Time	Max. Search Time
α=50%	SSF	LP	3145638	48.20 ms	0.091 ms	100 ns	136800 ns
		DH	362438	10.80 ms	0.001 ms	100 ns	72600 ns
	PAF	LP	667	2.150 ms	386 ns	100 ns	4700 ns
		DH	1141	2.336 ms	415 ns	100 ns	1700 ns
α=80%	SSF	LP	3145638	42.21 ms	0.040 ms	100 ns	143700 ns
		DH	362438	10.16 ms	0.010 ms	100 ns	80100 ns
	PAF	LP	5830	1.647 ms	782 ns	100 ns	2700 ns
		DH	9305	1.893 ms	791 ns	100 ns	3200 ns

5-) Conclusion:

As we can see on the table that mentioned above. **PAF & LP & 0.5 load factor** is most efficient way to store data. So Lets examine why.
PAF is distributing words clearly into the hashed dictionary. If we distribute them again by **DH** they locate randomized into the table. But **LP** is increases index by one when there is a collision. So lets visualize it.

1-) Let add 3 values into the table to occupied cells. Red x is if we used **DH,** blue x if we used **LP**, Orange x is collision.

X	X1		X	X2			X	X3	

X x1 x2	X1 x2	X2 x3	X x3	X3			X		

08.12.2022

Yusuf Gassaloğlu
2020510034