# **DOKUZ EYLUL UNIVERSITY**

# **ENGINEERING FACULTY**

# **DEPARTMENT OF COMPUTER ENGINEERING**

# **HOMEWORK I**

# **INVERTED INDEX BY USING HASH TABLES**

**by**

**Fatma KAHRAMAN**

**Lecturers**

**Ass. Prof Zerrin IŞIK**

**Res.Asst. Altuğ YİĞİT**

**14.11.2019**

|  |
| --- |
| **CONTENTS** |

**Page**

Contents 1

Linear Probing 3

Double Hashing 3

Inverted Index by using Hash Tables 4

Reference 5

# **LINEAR PROBING**

A simple method for collision handling with open addressing is linear probing. Linear probing: handles collisions by placing the colliding item in the next (circularly) available table cell. Each table cell inspected is referred to as a “probe”. Colliding items lump together, causing future collisions and leading a longer sequence of probes.

h(x)=(x+j) mod N, j = 0, 1, ... , N – 1 and N: table size

The load factor  = n/N affects the performance of a hash table. it is important that the load factor,be kept below 1.

# **DOUBLE HASHING**

Double hashingis a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs. Double hashing can be done using :

h2(k)=(h(k) + j d(k)) mod N , j = 0, 1, ... , N – 1 and N: table size

d2(k) = q - k mod q where q < N

The secondary hash function d(k) cannot have zero values. The table size N must be a prime to allow probing of all the cells.

The load factor  = n/N affects the performance of a hash table. it is important that the load factor,be kept below 1.

# **INVERTED INDEX BY USING HASH TABLES**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Load Factor** | **Hash Function** | **Collision Handling** | **Collision Count** | **Indexing Time** | **Avg. Search Time** | **Min. Search Time** | **Max. Search Time** |
| α=50% | SSF | LP | 2.713.026.720 | 66945 ms | 0,462 ms | 0,002 m s | 1,624 ms |
| DH | 738.119.455 | 32645 ms | 0,184 ms | 0,003 ms | 0,963 ms |
| PAF | LP | 579.807.488 | 25788 ms | 0,268 ms | 0,001 ms | 2,083 ms |
| DH | 228.912.771 | 19057 ms | 0,098 ms | 0,001 ms | 0.823 ms |
| α=80% | SSF | LP | 96.689.209 | 14186 ms | 0,014 ms | 0,002 ms | 0,143 ms |
| DH | 64.032.745 | 11798 ms | 0,015 ms | 0.001 ms | 0,097 ms |
| PAF | LP | 94.043.891 | 12594 ms | 0,014 ms | 0,003 ms | 0,138 ms |
| DH | 56.885.440 | 12781 ms | 0,014 ms | 0,001 ms | 0,047 ms |

***Table 1.*** *Performance matrix*

The project is implemented as Linear Probing and Double Hashing as SSF and PAF , which have hash table functions. There is too much collisions in the Linear Probing when the load factor is 50%. When the load factor is 50%, the first hashtable is resized. The number of collisions increases during resizing. In Linear Probing , it handles collisions by placing the colliding element in the next (circular) existing table cell. Also, the SSF is a linear function, the integer value it generates is smaller. Since there are many the number of collisions, the indexing time of the words takes longer in the hashtable. There is fewer collision count using PAF than using SSF in the probe hashing and double hashing . PAF is a polynomial function, the integer value it generates is big numbers. In PAF, different keys can be produced for each value. This results in fewer collisions. When load factor is 50% according to PAF, the index time takes less long than SSF.

The Duble Hashing works faster than the Linear Probing in performance. Because Double Hashing formula places any subsequent value in a position that is far from itself and where there is less chance of a collisions. Thus, each value is placed in the table without too much collisions. When load factor change as 80%, there is collusion count decreases. Also, Double hashing works again faster than Linear Probing. As a result, this indicates that the load factor is not a proper when the load factor is 50%. Because there are so less collisions when the load factor is 80%. The optimum value for the load factor is 80%.

# **REFERENCE**

1. <https://www.geeksforgeeks.org/double-hashing/>

# Data Structures and algorithms in Java, Sixth Edition, GoodRich MT,

# Tamassia R, Goldwasser MH, Wiley, 2015