2018-CE-212 Habiba Atique

**HASHING**

**(This document contains experimental conclusions of the lab)**

**Question # 1:**

Complete the functions so that they call tallyProbes on every successful insertion, with the number of probes required for that insertion and return that value as well. And in case of failure in insertion update 'numFailures'.

**Solution:**

All the functions i.e. linsert, qAdd, LExists and qExists are made and hash functions are also defined

hFLin1:

* converts 1st character of key(string) to equivalent key(integer)
* **hash function: (ch+pos) mod HTableSize**

hFLin2:

* converts key(string) to equivalent key(integer)
* **hash function: (ch+pos) mod HTableSize**

hFQuad1:

* converts 1st character of key(string) to equivalent key(integer)
* **hash function: (ch+pos2) mod HTableSize**

hFQuad2:

* converts key(string) to equivalent key(integer)
* **hash function: (ch+pos2) mod HTableSize**

Linsert:

* inserts the word from engmix.txt (as passed by int main) into ‘key\_arr[j]’ where ‘j’is the index returned by linear probing I.e. hFLin1 or hFLin2(used in actual)
* updtes the total no. of probes using **updateprobes** function
* returns no. of collisions when key is inserted
* returns -1 specifiying a failure in insertion of key

Qinsert:

* inserts the word from engmix.txt (as passed by int main) into ‘key\_arr[j]’ where ‘j’is the index returned by quadratic probing I.e hFQuad1 or hFQuad2(used in actual)
* updtes the total no. of probes using **updateprobes** function
* returns no. of collisions when key is inserted
* returns -1 specifiying a failure in insertion of key

LExits:

* uses hFLin1 or hFLin2(used in actual) to see if the passed string exists at the index return by linear probing
* returns true in case if key exists

QExists:

* uses hFQuad1 or hFQuad2(used in actual) to see if the passed string exists at the index return by linear probing
* returns true in case if key exists

**------------------------------------------------------------------------------------------------------------------------------------------**

**Question # 2**

(a)

**Load Factor**

**Solution:**

Keeping load factor 50% i.e. by keeping size of hash table double than the words in engmix.txt

in case of linear addressing collisions reduced than for case of 100% load factor but clustering could not be reduced enough as compared to quadratic probing as in case of quadratic probing collisions reduced a lot more than for case of 100% load factor and clustering reduced comparatively more than in case of linear addressing because linear addressing produces significant clustering even at low load factor.

Keeping load factor 60% is less effective than that of 50% load factor case and similarly keeping load factor 70% is even less effective than the case of 60% load factor.

Load factor affects quadratic probing more than linear probing and hence, low load factor decreases clustering significantly in quadratic probing and reduces collisions more than it does to linear probing.

(b)

Experiment with different hashfunctions for both methods, to pick your best (one with less average probing or probing rate).

**Solution:**

**I had 4 functions 2 for linear probing and 2 for quadratic probing as:**

**hFLin1: (ch + pos) % HTableSize**

where, ch= integer value of 1st character of key(string) only

**hFLin2: (ch + pos) % HTableSize**

where, ch= sum of integer values of all characters of key(string) only

**hFQuad1: (ch+pos2) % HTableSize**

where, ch= integer value of 1st character of key(string) only

**hFQuad2: (ch+pos2) % HTableSize**

where, ch= sum of integer values of all characters of key(string) only

**The best case is for hFQuad2 because:**

* in quadratic probing there is less clustering than linear probing so less collisions while inserting or searching words
* considering only 1st character of word will divide the whole of data into only 26 parts (since there exist only 26 alphabetical characters) and collisions will occur on the basis of these 26 characters. So, for a smart and better approach, considering the whole string will most of the times, give a different number when key is converted to equivalent integer value and so, the clustering will be less after taking mod and so, there will be less collisions while inserting and searching words from hashtable.

(c)

**Solution:**

**I will choose quadratic probing ovser linear addressing because of less clustering and hence less time to insert and search a value.**

**Because each time it fails to search or insert at some index it goes to index + 2nd power of next value thus reducing clustering and so the time to search and insert.**

**------------------------------------------------------------------------------------------------------------------------------------------**

**Question # 3:**

in case of failure to find any value, is there any way to identify if failure resulted because no try was made to insert the value or because the try was done but it resulted in failure.

**Solution:**

**Yes,** In case of failure to find any value there are 2 possibilities (According to my program):

* The status at the index proposed by hashfunction in status array be 0 represents that there was no attempt made to insert value
* The status at index proposed by hashfunction in status text file be -1 represents that there was a failure in attempt of storing value there because in case of failure insert functions returns -1