Cs202-hw4

Hashing

Alper Kandemir

21703062

Sec-01

Part 1:

CollisionStrategy option = choosing LINEAR, QUADRATIC, DOUBLE

int* hashTable = integer array of items

int tableSize = size of table

bool* isEmpty = false means the index is occupied, true means the index is empty

HashTable(const int tableSize, const CollisionStrategy option):

Constructor takes table size and collision option. Initialize the int array that keeps items and the bool array that keeps index availability. Initialize all items to zero and all availabilities to true.

~HashTable():

Delete items array and availability array.

bool insert(const int item):

Unique items are allowed and zero cannot be added. The function takes the item and check its proper index to be added. It uses primary hashing first if there is collision it uses collision hashing. If the addition successful the function returns true else return false.

bool remove(const int item):

Checks the item in the array one by one. If the item is found the function removes the item (put zero to the index) and return true. Also, the index's availability will be true.

bool search(const int item,int&numProbes):

Searching according to collision option and return true if the item is found and return the number of probes.

void display( ):

Prints the table one by one if the index is free, it prints nothing.

void analyze(double& numSuccProbes,double& numUnSuccProbes);

Finds the average number of probes for successful and unsuccessful searches according the collision option. The function starts a search for occupied index's items and calculate the probes than add to numSuccProbes, and divide to size of occupied index's items for successful search. For unsuccessful search, it counts the number of probes needed to reach an empty location in the array for each index. Then adds the result to numUnSuccProbes, and divide to table size. Return double results.

int hashPrimary(const int item);

Hash(key)=key mod tableSize. Returns index according to hashing

int collisionHash(const int item);

hi(key) = hash(key) + f(i) mod tableSize. f(i) changes according to collision strategy. Returns index according to certain hashing type.

int doubleHash(const int item);

Helper function for collision hash when the option is DOUBLE. f(i) = i hash2(key) where hash2(key) = reverse(key) is the secondary hash function that reverses the digits of the key (e.g., reverse (1234) = 4321).

Collision Resolution Strategies:

I used table size to check end condition for loops to prevent infinite looping because using table size as end value is enough to visit all indexes once.

Loop goes to 0 to tablesize

Linear:

(index + (i + 1)) % tableSize

Loop goes to 1 to tablesize

Quadratic:

(index+(i * i)) % tableSize

Loop goes to 1 to tablesize

Double:

(i*doubleHash(item)) % tableSize ----> doubleHash gets reverse of the item.

Part 2:

Size:15

```
------------------------------LINEAR------------------------------
20 inserted
30 inserted
2 inserted
13 inserted
25 inserted
24 inserted
10 inserted
9 inserted
20 found after 1
25 found after 1
9 found after 4
1 inserted
0 cannot be added
0 not inserted
Item:2 is already exist
2 not inserted
20 removed
30 removed
2 removed
2 not removed
1234 inserted
1234 found after 1
0:
1: 1
2:
3:
4: 1234
5:
6:
7:
8:
9: 24
10: 25
11: 10
12: 9
13: 13
14:
Table size: 15
successful search: 1.57143
unsuccessful search: 2.13333
```

```
------------------------------QUADRATIC--------------------------------
20 inserted
30 inserted
2 inserted
13 inserted
25 inserted
24 inserted
10 inserted
9 inserted
20 found after 1
25 found after 1
9 found after 4
1 inserted
0 cannot be added
0 not inserted
Item:2 is already exist
2 not inserted
20 removed
30 removed
2 removed
2 not removed
1234 inserted
1234 found after 1
0:
1: 1
2:
3: 9
4: 1234
5:
6:
7:
8:
9: 24
10: 25
11: 10
12:
13: 13
14:
Table size: 15
successful search: 1.57143
unsuccessful search: 1.46667
```

```
------------------------------DOUBLE--------------------------------
20 inserted
30 inserted
2 inserted
13 inserted
25 inserted
24 inserted
10 inserted
9 inserted
20 found after 1
25 found after 1
9 found after 3
1 inserted
0 cannot be added
0 not inserted
Item:2 is already exist
2 not inserted
20 removed
30 removed
2 removed
2 not removed
1234 inserted
1234 found after 3
0:
1: 10
2: 1234
3: 9
4: 1
5:
6:
7:
8:
9: 24
10: 25
11:
12:
13: 13
14:
Table size: 15
successful search: 2.28571
unsuccessful search: -1
```