CS 202

Homework 04

Section 01

Hash Table Implementation

Ali Eren Günaltılı

21801897

28/12/2021

My HashTable Implementation

I use linear and quadratic probing the way everyboy use as they are standartized. But in the implementation of double hashing, some implementation considers prime value as a constant 7 while some others take the prime value as biggest prime value that is smaller than table size. I take the constant 7 version. That means in my implementation, I use this algorithm for double hashing operations.

```
else if (strategy == DOUBLE) {
    int ind = item % table_size;
    if (table[ind] == INT_MIN) {
        table[ind] = item;
    }
    else {
        //ind  is the result of h1(x) we need h2(x) for double hashing.
        int h2_place = 7 - (item % 7);
        int place = ind;
        int i = 1;
        while (table[place] != INT_MIN) {
            place = ind + (h2_place * i);
            if (place >= table_size) {
                place = place % table_size;
            }
            i++;
        }
        table[place] = item;
    }
}
```

I store the enum value from constructor and I use it everytime we are going to make some table operations.

About the collision strategies, I implement two important things to avoid stack over flow. One of them is obviously checking elements until we reach some empty place. I hold the null points by putting INT_MIN inside of them to avoid read access violation due to null pointer exception. In my implementation, I check indices until some index's value is INT_MIN. Also avoid read access violation, I implement the check condition for whether the searched index is currently bigger than the table size or not. If it is, take the modula of the current index and then search it. I didn't implement the recreation of the table respect to its load factor but it is also solution for avoiding infinite loops.

In our cases, same indices might be visited in case of there is no place for new inserted values. As I said previous, we don't implement the recreation of table when the load factor is exceeded. Thus, I check whether the array is full or not in the search and insert methods.

My input txt file

```
I 132
I 534
I 53
I 9
I 82
I 28
I 3
I 55
S 3
R 3
S 3
S 53
R 53
S 53
I 199
I 12
I 280
I 721
I 122

```

For this provided input.txt file output will be like this.

Linear Probing

```
In linear, 3 is founded at 3 probes.
In quadratic, 3 is founded at 4 probes.
In double, 3 is founded at 4 probes.
3 in index 5 is deleted.
3 in index 12 is deleted.
3 in index 15 is deleted.
In linear, 3 is not founded at 3 probes.
In quadratic, 3 is  not founded at 3 probes.
In double, 3 is not founded at 3 probes.
In linear, 53 is founded at 1 probes.
In quadratic, 53 is founded at 1 probes.
In double, 53 is founded at 1 probes.
53 in index 3 is deleted.
53 in index 3 is deleted.
53 in index 3 is deleted.
In linear, 53 is not founded at 1 probes.
In quadratic, 53 is  not founded at 1 probes.
In double, 53 is not founded at 0 probes.
Table size for all tables is : 25
---------- Linear Table ------------
0 :
1 :
2 :
3 :
4 : 28
5 : 280
6 : 55
7 : 132
8 : 82
9 : 534
10 : 9
11 :
12 : 12
13 :
14 :
15 :
16 :
17 :
18 :
19 :
20 :
21 : 721
22 : 122
23 :
24 : 199
For linear suc : 14, unsucc : 59
```

Quadratic Probing

```
For linear suc : 14, unsucc : 59
----------- Quadratic Table -----------
0 :
1 :
2 :
3 :
4 : 28
5 : 55
6 : 280
7 : 132
8 : 82
9 : 534
10 : 9
11 :
12 : 12
13 :
14 :
15 :
16 :
17 :
18 :
19 :
20 :
21 : 721
22 : 122
23 :
24 : 199
---------------
For quadratic suc : 14, unsucc : 36
```

Double Hash Probing

```
--------------Double Table----------------
0 :
1 :
2 :
3 :
4 :
5 : 55
6 :
7 : 132
8 :
9 : 534
10 : 28
11 : 82
12 : 12
13 :
14 : 9
15 :
16 :
17 :
18 :
19 : 280
20 :
21 : 721
22 : 122
23 :
24 : 199
For double suc : 15, unsucc : 22
```

Linear Probing: Succesful = 14, Unsuccessful = 59

Quadratic Table: Successful = 14, Unsuccessful = 36

Double Hash Probing: Successful = 15, Unsuccessful = 22

Experimental Value vs Theoratical Value

First thing to observe is that unsuccessful probing value of linear probing is much much bigger than the other two. This observation is also what we expect from theoratical value. This difference become more obvious when the load factor(a) increases.

Our hash table size is 25 and 11 of the slots are full. That means our load factor is 11/25 which is 0.44. By using this load factor value now we can compare experimental value and theoratical value.

Linear Probing :

$$\frac{1}{2}\left[1+\frac{1}{1-\alpha}\right] \qquad \textit{for a successful search}$$

$$\frac{1}{2}\left[1+\frac{1}{(1-\alpha)^2}\right] \qquad \textit{for an unsuccessful search}$$

Quadratic and Double Hashing

$$\left[\frac{1}{\alpha}(\log_e \frac{1}{1-\alpha})\right] = \frac{-\log_e(1-\alpha)}{\alpha} \qquad \textit{for a successful search}$$

$$\frac{1}{1-\alpha} \qquad \textit{for an unsuccessful search}$$

Where a = 0.44

Theoratical values for Linear Probing’s successful search : 1.3928

Theoratical values for Double Hash Probing and Quadratic Probing’s successful search: 1.3177

Theoratical values for Linear Probing’s unsuccessful search: 3.08264

Theoratical values for Double Hash Probing and Quadratic Probing’s unsuccessful search: 1.7857

What we get from our experiment:

Experimental value for Linear Probing’s successful search: 14, unsuccessful: 59

Experimental value for Quadratic Probing’s successful search: 14, unsuccessful: 36

Experimental value for Double Hashing Probing’s successful search: 15, unsuccessful: 22

As the theoratical values suggests difference between unsuccessful searches are more dramatic. There is a slight difference between linear probing’s successful search rate and other two’s rate as one of them is 1.3928, other of them is 1.3177. We can prove this from our code too. Experimental values for successful searches are too close to each other but for unsuccessful part there is a real change in the efficiency. Using double hash probing and quadratic probing are way more efficient than using linear probing as the theoratical value suggests.

Note: I have to change my INT_MIN to some negative value(-9999) as dijkstra machine cannot compile and gives error to INT_MIN.