2019-2020 SPRING CS202 HW4

Rüzgar Ayan 21801984 Section 2

Part1.

For the implementation of HashTable class, I have used two arrays both of the length of tableSize. The first one called ***table*** is simply the array that holds the items. The second one ***states*** is an array that can have the values {OCCUPIED, EMPTY, DELETED} for each position. As described in the course slides, this array is used to solve problems that occur after deletions. In the constructor, all elements of this array are initialized to EMPTY.

Insert: For inserting, the method probes until a non-OCCUPIED position is found or until the number of probes is equal to the size of table. Because for all 3 collision strategies, the hash function will repeatedly give the same positions after probing a number of *tableSize*. Therefore, there is no point of checking for new non-OCCUPIED positions after that. When such position is found, its state is set to OCCUPIED and the item is put into the position and returned true. If no position is found after *tableSize* probes, return false.

Remove: This method probes until we find an empty cell or we find the item we want or we probe *tableSize* times. Finding an empty cell and probing *tableSize* times (for the same reason as in inserting) means that we won't find our item anymore, therefore we return false. If we find cell the cell containing our item, we set its state to DELETED and just to guarantee we set the value in that cell to 0, then we return true.

Search: The conditions to stop probing are the same as remove function's stop conditions. We return false if we have probed *tableSize* times or found and empty cell. We return true if we have found a cell with the searched item's value.

Display: Simply iterate over all cells, print the item value inside only if the state of that cell is OCCUPIED.

Analyze: For successful searches, just take the items in the table already and use our search function to get the *numProbes*, and then find the average of the result. For unsuccessful searches, as the assignment file suggested, I counted the number of probes required for finding the next EMPTY cell, up to at most tableSize cells.

Part2.

The input file contents:

- S 1
- R 1
- I 1
- S 1
- R 1
- I 2
- I 3
- I 4
- S 2
- I 13
- I 24
- I 35
- S 13
- S 24
- S 35
- R 35
- I 250
- I 350
- I 900
- I 1500
- S 900
- S 1500
- S 132
- S 141
- R 1000

Output from the driver function:

```
1 not found after 1 probes
1 not removed
1 inserted
1 found after 1 probes
1 removed
2 inserted
3 inserted
4 inserted
2 found after 1 probes
13 inserted
24 inserted
35 inserted
13 found after 1 probes
24 found after 1 probes
35 found after 1 probes
35 removed
250 inserted
350 inserted
900 inserted
1500 inserted
900 found after 4 probes
1500 found after 3 probes
132 not found after 7 probes
141 not found after 11 probes
1000 not removed


The size of the table used is 13.
0: 13
1:
2: 2
3: 3
4: 4
5: 250
6: 900
7: 1500
8:
9:
10:
11: 24
12: 350
Average number of probes successful searches: 1.77778
Average number of probes unsuccessful searches: 5.30769
```

Part3.

The simulation in Part 2 is done with **Linear Probing**. Its empirical results are as follows:

(Empirical) Average number of probes successful searches: 1.77778
(Empirical) Average number of probes unsuccessful searches: 5.30769

For theoretical values, we have the load factor $\propto = \frac{9}{13} \approx 0.692$. And using the formulas given in the slides, we will get the following values:

(Theoretical) Average number of probes successful searches $= \frac{1}{2}\left(1 + \frac{1}{1-\propto}\right) = 2.125$
(Theoretical) Average number of probes unsuccessful searches $= \frac{1}{2}\left(1 + \frac{1}{(1-\propto)^2}\right) = 5.78125$

The reason for the difference between the empirical and theoretical result can be that we deleted some values from the table, which decreases the load factor. But if we don't put another item into this DELETED cell, the required number of probes will remain the same although the load factor decreases and this might cause differences in empirical and theoretical values.

Next, doing the same simulation with **Quadratic Probing**, we get the empirical results:

(Empirical) Average number of probes successful searches: 1.66667
(Empirical) Average number of probes unsuccessful searches: 4.92308

Again with the load factor $\propto = \frac{9}{13} \approx 0.692$, we get the theoretical results:

(Theoretical) Average number of probes successful searches $= \frac{-\ln(1-\propto)}{\propto} = 1.7025$
(Theoretical) Average number of probes unsuccessful searches $= \frac{1}{1-\propto} = 3.25$

When using **Double Hashing**, we have:

(Empirical) Average number of probes successful searches: 1.25

This time we have $\propto = \frac{8}{13}$, and:

(Theoretical) Average number of probes successful searches $= \frac{-\ln(1-\propto)}{\propto} = 1.55$

Overall for all 3 collision strategies, these theoretical results are not very good. But this was what I expected since the table I have used (*tableSize* = 13) was really small. Probably if we were to use bigger tables, the theoretical results would be much closer to the real values.