CS202

Osman Buğra Aydın

21704100

Homework04

Section03

Part1-) Description Of HashTable Class

HashTable class has 5 properties which are private and 12 methods which are all public. Besides, there are 3 enums.

Enum

CollisionStrategy { LINEAR, QUADRATIC, DOUBLE } defination stands for the strategy that hashTable class will use.

AddressCalculation { INSERT, FIND } defination exists to decide for which method between insert and find addressCalculator method will work.

StatusOfCell { EMPTY, DELETED, OCCUPIED } defination stands for the current situation of cell of hashTable class.

Properties

Integer *table is for the hash table.

StatusOfCell *statusOftable is to decide if a cell is empty, decide or occupied.

Integer tableSize is the number of whole cells in hash table

CollisionStrategy strategy is to decide hash table's strategy.

Integer curAmountElements is the number of occupied cells in hash table

Methods

Constructer and deconstructer are classic initilaziton of a class. Memory is allocated in constructer and deallocated in deconstructer.

Insert method calls the addressCalculator method to find appropiate location to insert. After taking the location, item will be inserted, unless the value of location is -1. Also, the number of current elements will be increased by one.

Remove method calls the addressCalculator method to find appropiate location to remove. After taking the location, item will be removed, unless the value of location is - 1.Also, the number of current elements will be decreased by one.

Search methods calls the addressCalculator method to find the item that is given by the parameter. The number of probes will be also decided in the addressCalculator method. It will return true, unless the calculated location is -1.

Display method creates a loop from the beginning of the statusOftable which contains the status of all cells to the end of the array. If a cell is not occupied, it will not print anything. Otherwise, it prints the element according to counter.

Analyze method creates a loop from the beginning of the table to the end of the array. If a cell is occupied, addressCalculator will be called to find how many probes is needed to find current item and probe will be summed. In addition, if strategy is not double, addressCalculator will be called to find how many probes is for counter to determine unsuccessful searchs. Finally, summation of probes will be divided by number of current elements where as number of unsuccessful probes will be divided by tableSize unless strategy is double.

AddressCalculation will calculate address according to strategy and option which can be insert or find. At the beginning, it checks if table is full and option is insert, then returns -1. If table is empty and option is find, then return -1. There are three conditions for strategy. In each of them location is calculated as it is described in hw4.pdf file. For all of the strategies in inserting, if option is insert and the calculated location of statusOftable is not occupied, loop will stop and location will be returned. The number of probes will be equal to the counter in the loop. For all of the strategies in searching, there are two conditions. First one, if option is find and the item in the hashTable according to counter is equal to target item and the cell in the statusOftable according to counter is equal to occupied, then the target item will be found. Second one, if option is find and the cell in the statusOftable according to counter is not equal to occupied, the number of probes will be equal to zero and return -1. Stopping conditions of these three different collision stategies are similar. For linear, it will be finished if counter equals to tableSize because it would look at the all cells in hashTable. For quadratic, it will be finished if counter equals to the half of the tableSize because after the half the counter^2 % tableSize value will be same. There will be simetric results thats why it is reduced to the half of the tableSize. For double, it will be finished if counter equals to tableSize. It would check the every cell if the reverse of the number is prime to original number.

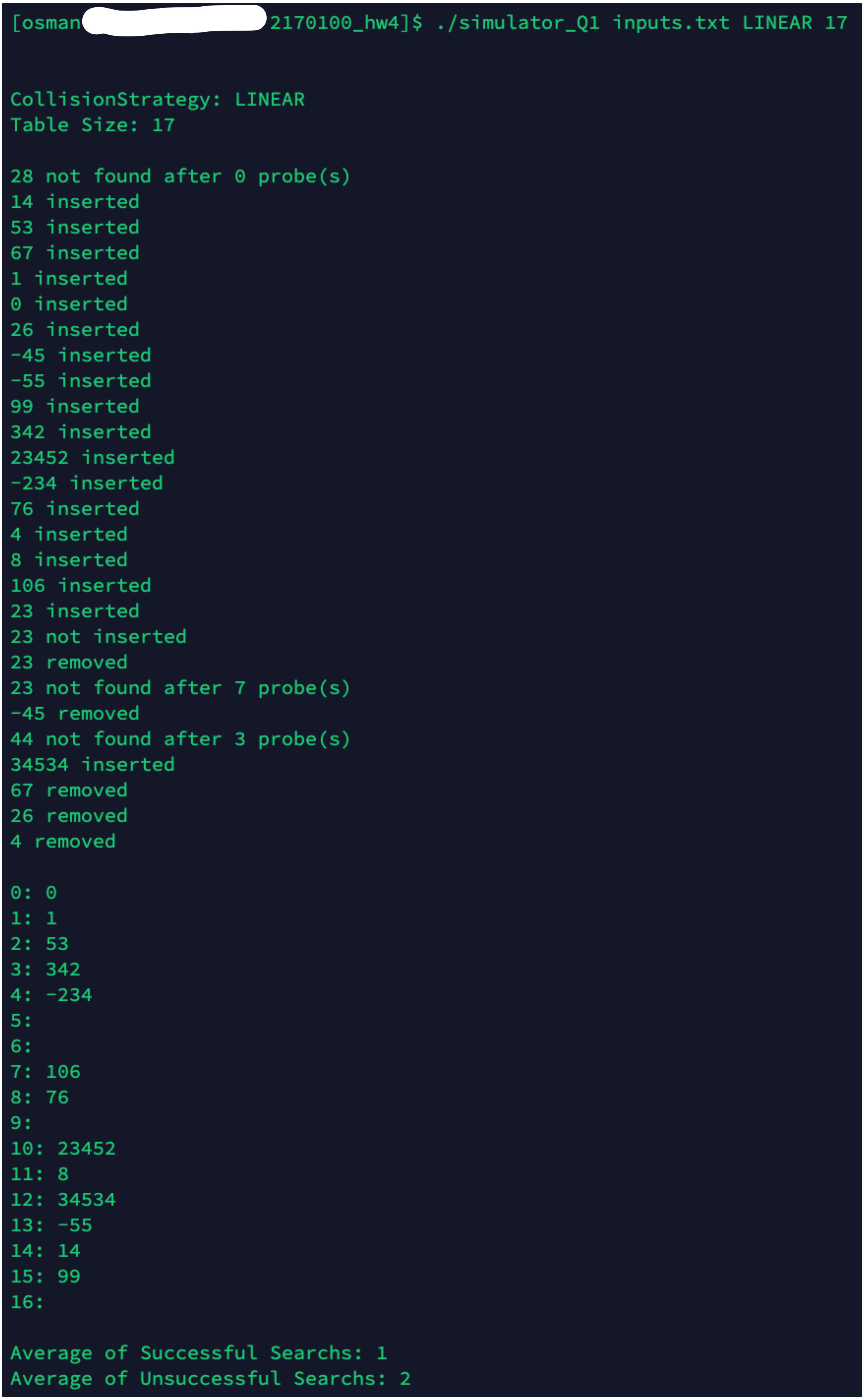
SecondHash method will compute the reverse of an integer by taking modulo and multipying by 10.

IsFull method returns true if the number of current elements is equal to tableSize.

IsEmpty method returns true if the number of current elements is equal to zero.

CalculateModulo method return the modulo of a value which can be negative and positive.

Part2-) Examples Of HashTable



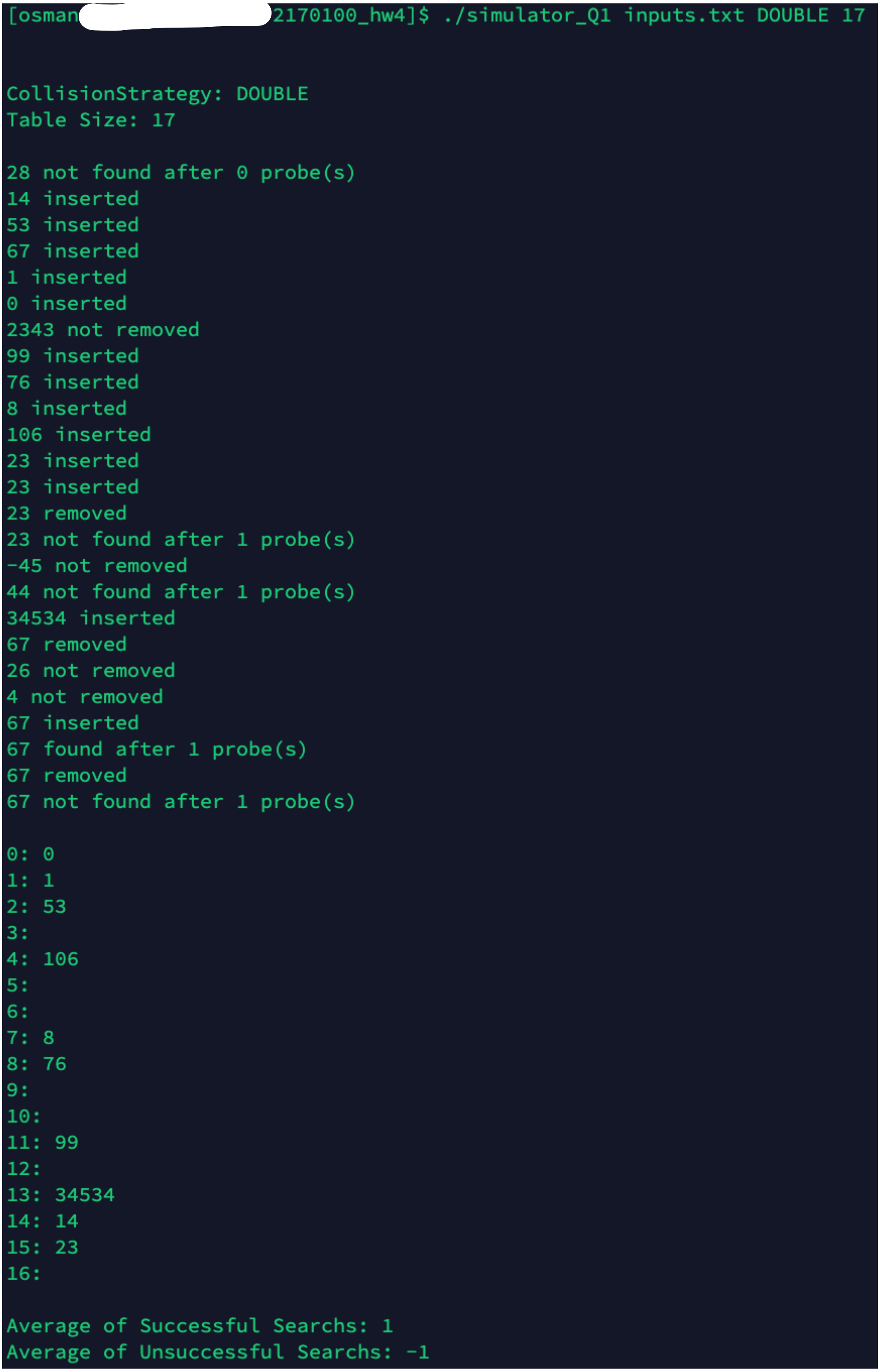
```
[osman[redacted]2170100_hw4]$ ./simulator_Q1 inputs.txt LINEAR 17


CollisionStrategy: LINEAR
Table Size: 17

28 not found after 0 probe(s)
14 inserted
53 inserted
67 inserted
1 inserted
0 inserted
26 inserted
-45 inserted
-55 inserted
99 inserted
342 inserted
23452 inserted
-234 inserted
76 inserted
4 inserted
8 inserted
106 inserted
23 inserted
23 not inserted
23 removed
23 not found after 7 probe(s)
-45 removed
44 not found after 3 probe(s)
34534 inserted
67 removed
26 removed
4 removed

0: 0
1: 1
2: 53
3: 342
4: -234
5:
6:
7: 106
8: 76
9:
10: 23452
11: 8
12: 34534
13: -55
14: 14
15: 99
16:

Average of Successful Searchs: 1
Average of Unsuccessful Searchs: 2
```



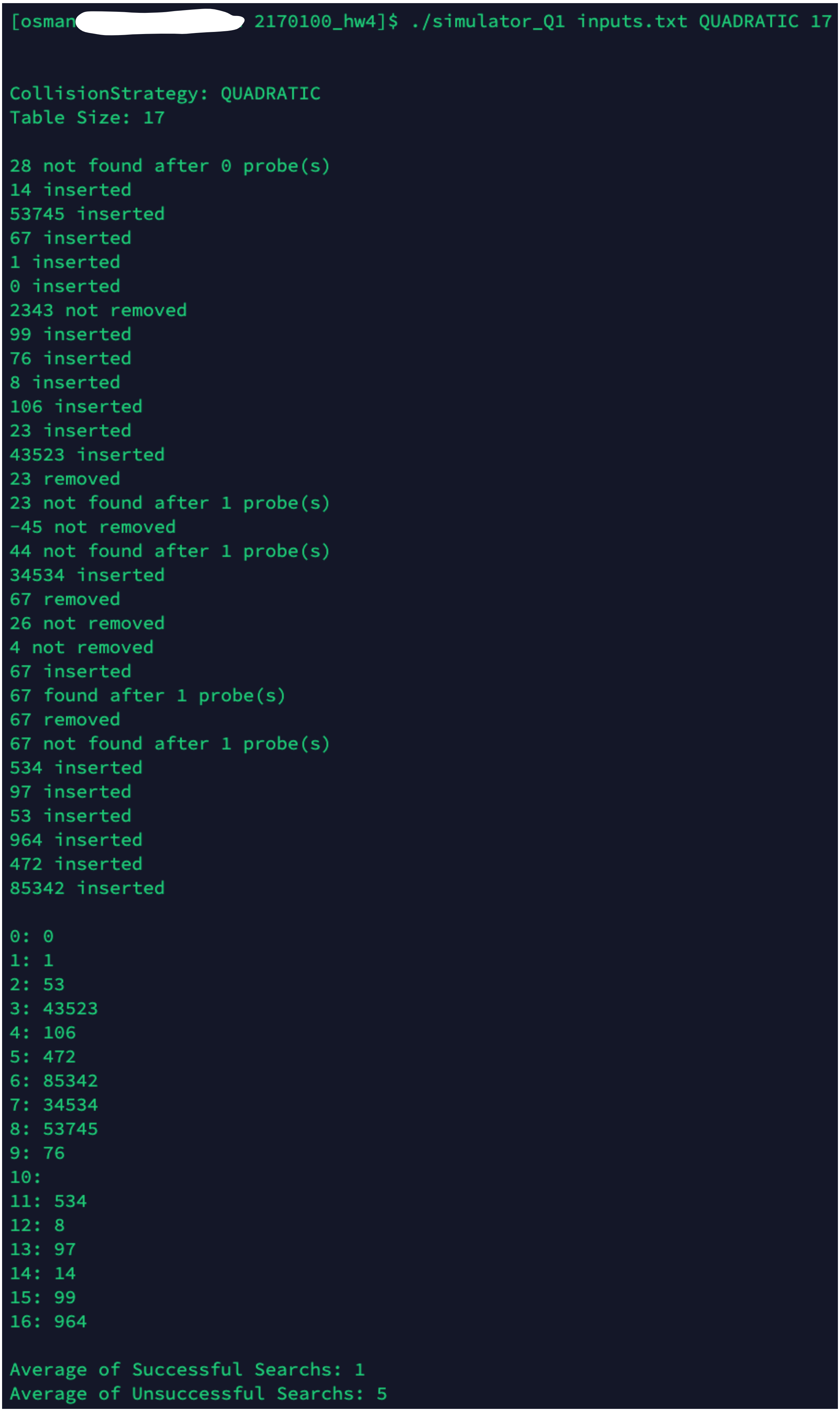
```
[osman[redacted]2170100_hw4]$ ./simulator_Q1 inputs.txt DOUBLE 17


CollisionStrategy: DOUBLE
Table Size: 17

28 not found after 0 probe(s)
14 inserted
53 inserted
67 inserted
1 inserted
0 inserted
2343 not removed
99 inserted
76 inserted
8 inserted
106 inserted
23 inserted
23 inserted
23 removed
23 not found after 1 probe(s)
-45 not removed
44 not found after 1 probe(s)
34534 inserted
67 removed
26 not removed
4 not removed
67 inserted
67 found after 1 probe(s)
67 removed
67 not found after 1 probe(s)

0: 0
1: 1
2: 53
3:
4: 106
5:
6:
7: 8
8: 76
9:
10:
11: 99
12:
13: 34534
14: 14
15: 23
16:

Average of Successful Searchs: 1
Average of Unsuccessful Searchs: -1
```



```
[osman[redacted] 2170100_hw4]$ ./simulator_Q1 inputs.txt QUADRATIC 17


CollisionStrategy: QUADRATIC
Table Size: 17

28 not found after 0 probe(s)
14 inserted
53745 inserted
67 inserted
1 inserted
0 inserted
2343 not removed
99 inserted
76 inserted
8 inserted
106 inserted
23 inserted
43523 inserted
23 removed
23 not found after 1 probe(s)
-45 not removed
44 not found after 1 probe(s)
34534 inserted
67 removed
26 not removed
4 not removed
67 inserted
67 found after 1 probe(s)
67 removed
67 not found after 1 probe(s)
534 inserted
97 inserted
53 inserted
964 inserted
472 inserted
85342 inserted

0: 0
1: 1
2: 53
3: 43523
4: 106
5: 472
6: 85342
7: 34534
8: 53745
9: 76
10:
11: 534
12: 8
13: 97
14: 14
15: 99
16: 964

Average of Successful Searchs: 1
Average of Unsuccessful Searchs: 5
```

Part3-)Performance Values & Observation

Empirical values

Linear probing

Average of successful probing: 1
Average of unsuccessful probing: 2

Quadratic probing

Average of successful probing: 1
Average of unsuccessful probing: 5

Double probing

Average of successful probing: 1

Theoritical values

Linear probing

Average of successful probing: 2.625
Average of unsuccessful probing: 9.531

Quadratic probing

Average of successful probing: 9
Average of unsuccessful probing: 145

Double probing

Average of successful probing: 3.448

Observation

There are some differences between empricial and theoritical values in both successful and unsuccessful search. There are several reasons for that. First one is the improvements in implementation. For example, if hashTable is empty, it will not even go in the loop. It just returns location -1 which means it could not find the target item in the hashTable. For quadratic strategy, searching is enhanced beacuse counter goes to the half of the tableSize instead of going full size. The reason why unsuccessful searching of quadratic strategy is bigger than unsuccessful searching of linear is the load factor is much more bigger. Load factor is computed by number of elements in the array divided by tableSize. If load factor increases, it means there will be a more inefficient search. Quadratic searching is better in successful searching beacuse their averages of successful searching are equal to eachother even if the load factor is much more higher than linear. That is because clustering is a trade between quadratic and linear. In addition to this, it can be inferred from last sentences double hashing is the best for successful searching in theory. It seems that they are equal in the empirical data but the difference will be obtained if different input data is used. The trade in the double hashing is computing and cache performance is weak. There are more computing because there is a second hashing function. However, the clustering problem is solved.