/**
***Title: HashTables**

***Author: Melis Atun**

***ID: 21901865**

***Assignment: 4**

***Description: hw.pdf report**
***/**

QUESTION 2

PART 1: Describe briefly your design of the HashTable class and how you implement the collision resolution strategies (e.g., how to decide when to stop probing).

Note: You should carefully design the stopping conditions to avoid infinite loops where probing can cycle through the same sequence of array indices even though the hash table is not completely full. You should think about this before actually implementing the methods.

Before implementing the hash table methods, I studied the topic from the lecture slides. Then, I first wrote the insert function and tested it. However, as I expected, I got many errors because I was not checking the corner cases and I was not paying attention to the variables in the function calls or in the loops. Also, my test function got into an infinite loop for a few times because I did not pay attention to when to stop probing. After these infinite loop mistakes, I realized that I need to stop probing when the item is inserted into the correct place. After testing the insert function for many different arrays and made sure that it is working properly, I moved on to other functions (remove and search). Those functions were a bit easier to write because they resembled insert function (same corner cases, for example) and they were actually easier and less complicated than the insert function because there were not any collisions in remove and insert functions unlike the insert function. This is because when a bucket is full, I needed to resolve this by linear, quadratic and double hashing methods but when removing an item, this is not the case because I just needed to find the item and remove it directly by making its place equal to -1.

PART 2:

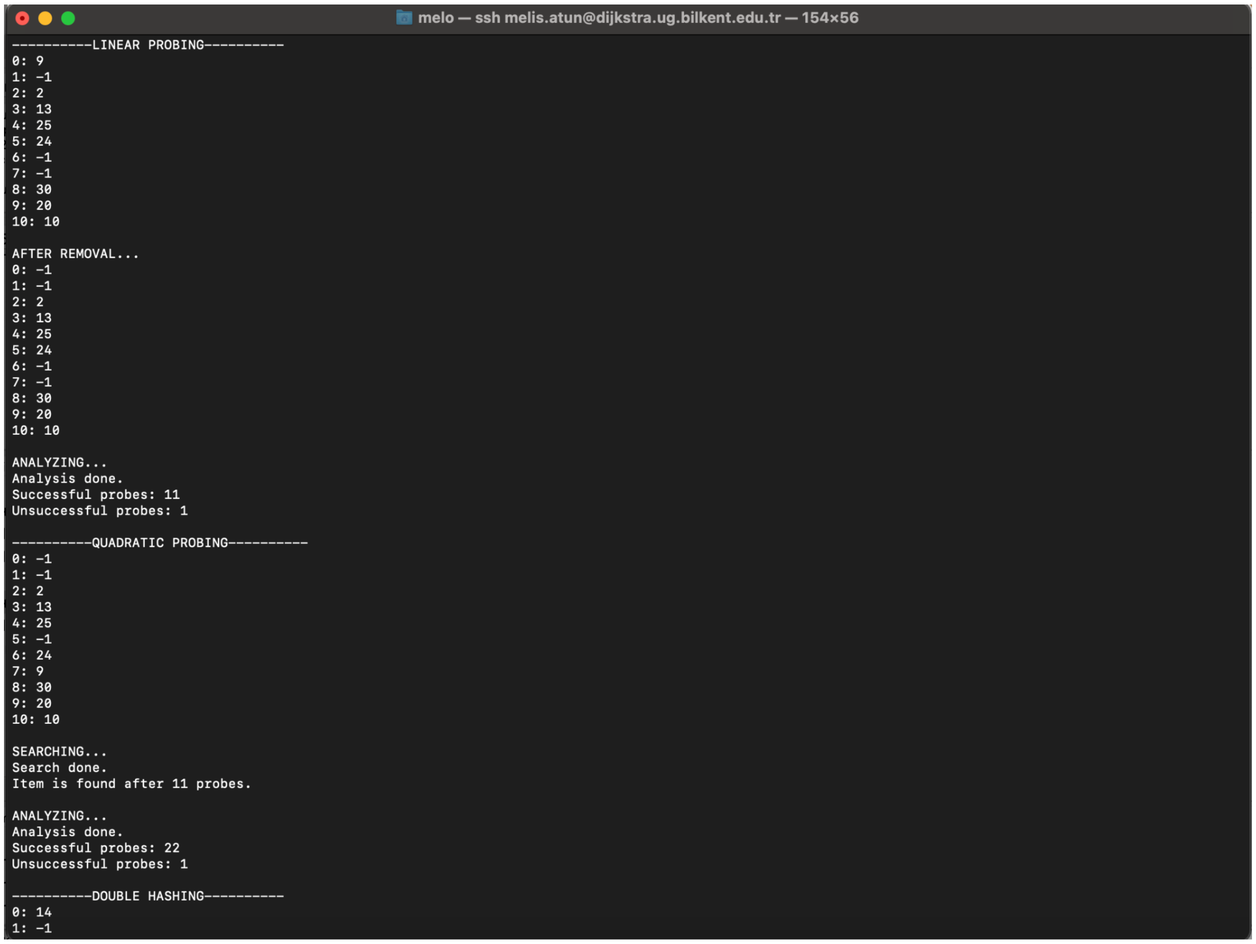
- **table size used: 11**
- **driver code that I have used for my testing purposes (also in the main.cpp file):**

```
int main() {

   CollisionStrategy collision;

   HashTable h1(11, collision);

   int probes = 0;

   int successful = 0;

   int unsuccessful = 0;

   cout << endl;

   cout << "----------LINEAR PROBING----------" << endl;

   collision = LINEAR;

   h1.insert(20); //9

   h1.insert(30); //8

   h1.insert(2); //2

   h1.insert(13); //3

   h1.insert(25); //4

   h1.insert(24); //5

   h1.insert(10); //10

   h1.insert(9); //0
```

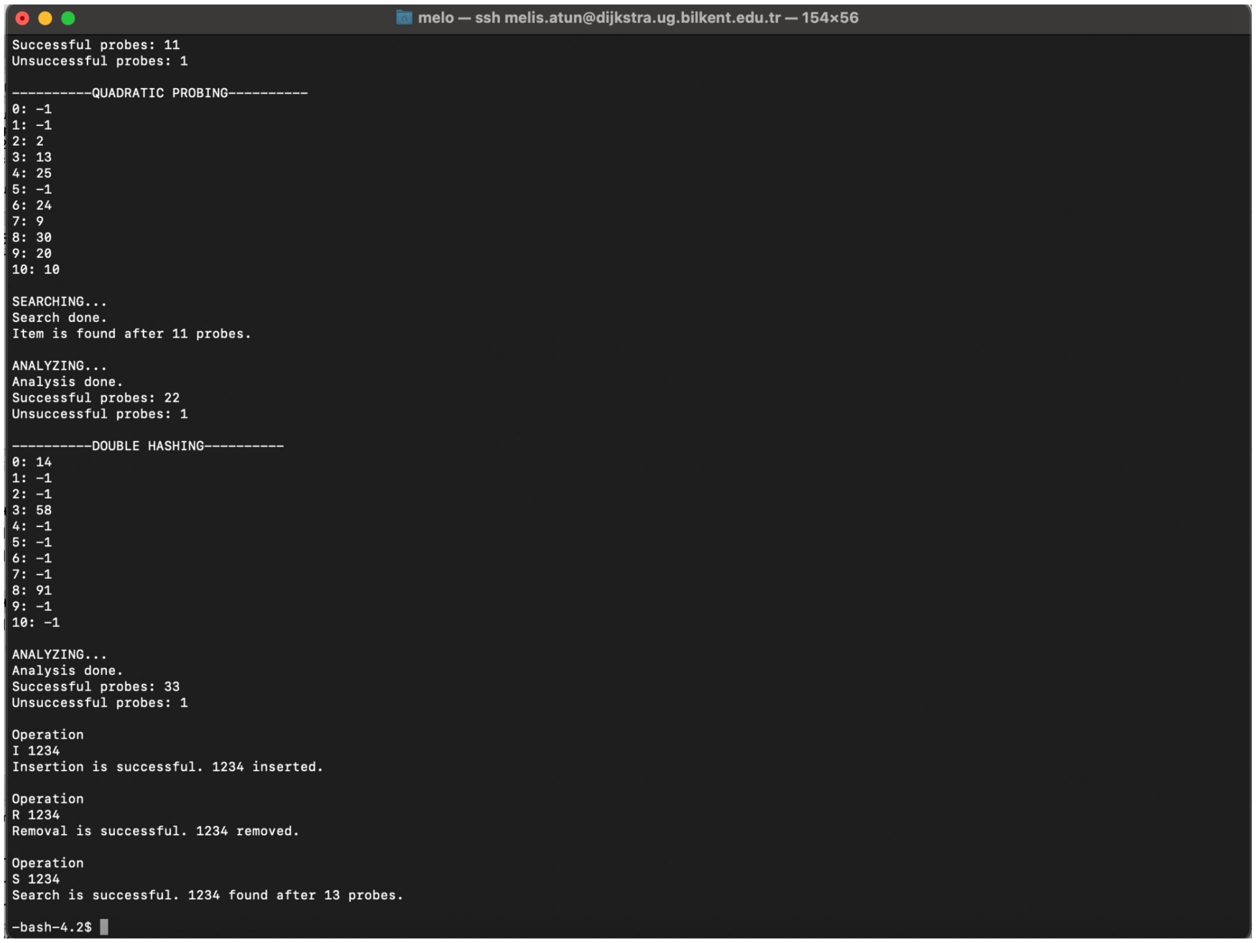
```
h1.display();

cout << endl;

cout << "AFTER REMOVAL..." << endl;

h1.remove(9);

h1.remove(30);

h1.remove(10);

h1. display();

cout << endl;

cout << "ANALYZING..." << endl;

h1.analyze(successful, unsuccessful);

HashTable h2(11, collision);

cout << endl;

cout << "----------QUADRATIC PROBING----------" << endl;

collision = QUADRATIC;

h2.insert_helper(20); //9

h2.insert_helper(30); //8

h2.insert_helper(2);  //2

h2.insert_helper(13); //3

h2.insert_helper(25); //4

h2.insert_helper(24); //6

h2.insert_helper(10); //10

h2.insert_helper(9);  //7

h2.display();

cout << endl;
```

```
   cout << "SEARCHING..." << endl;

   h2.search(13, probes); //random item from the above array to test the search
function

   cout << "Search done." << endl;

   cout << "Item is found after " << probes << " probes." << endl;

   cout << endl;

   cout << "ANALYZING..." << endl;

   h2.analyze(successful, unsuccessful);

   HashTable h3(11, collision);

   cout << endl;

   cout << "----------DOUBLE HASHING----------" << endl;

   collision = DOUBLE;

   h3.insert_helper2(58); //3

   h3.insert_helper2(14); //0

   h3.insert_helper2(91); //8

   h3.display();

   cout << endl;

   cout << "ANALYZING..." << endl;

   h3.analyze(successful, unsuccessful);

   /*

   * QUESTION 2, PART 2 TEST BELOW

   */

   HashTable test(20, collision);

   cout << endl;
cout << "Operation" << endl;

cout << "I 1234" << endl;

test.insert(1234);

if (test.insert(1234) == true) {

    cout << "Insertion is successful. 1234 inserted." << endl;

}

else {

    cout << "Insertion is unsuccessful. 1234 not inserted." << endl;

}

cout << endl;

cout << "Operation" << endl;

cout << "R 1234" << endl;

test.remove(1234);

if (test.remove(1234) == true) {

    cout << "Removal is successful. 1234 removed." << endl;

}

else {

    cout << "Removal is unsuccessful. 1234 not removed." << endl;

}

cout << endl;

cout << "Operation" << endl;

cout << "S 1234" << endl;

test.search(1234, probes);

if (test.search(1234, probes) == true) {
        cout << "Search is successful. 1234 found after " << probes << " probes." << endl;
    }
    else {
        cout << "Search is unsuccessful. 1234 not found after " << probes << " probes." << endl;
    }
    cout << endl;
```

- **Output of main function (continued on the other page):**



```
----------LINEAR PROBING----------
0: 9
1: -1
2: 2
3: 13
4: 25
5: 24
6: -1
7: -1
8: 30
9: 20
10: 10

AFTER REMOVAL...
0: -1
1: -1
2: 2
3: 13
4: 25
5: 24
6: -1
7: -1
8: 30
9: 20
10: 10

ANALYZING...
Analysis done.
Successful probes: 11
Unsuccessful probes: 1

----------QUADRATIC PROBING----------
0: -1
1: -1
2: 2
3: 13
4: 25
5: -1
6: 24
7: 9
8: 30
9: 20
10: 10

SEARCHING...
Search done.
Item is found after 11 probes.

ANALYZING...
Analysis done.
Successful probes: 22
Unsuccessful probes: 1

----------DOUBLE HASHING----------
0: 14
1: -1
```



melo — ssh melis.atun@dijkstra.ug.bilkent.edu.tr — 154×56

```
Successful probes: 11
Unsuccessful probes: 1

----------QUADRATIC PROBING----------
0: -1
1: -1
2: 2
3: 13
4: 25
5: -1
6: 24
7: 9
8: 30
9: 20
10: 10

SEARCHING...
Search done.
Item is found after 11 probes.

ANALYZING...
Analysis done.
Successful probes: 22
Unsuccessful probes: 1

----------DOUBLE HASHING----------
0: 14
1: -1
2: -1
3: 58
4: -1
5: -1
6: -1
7: -1
8: 91
9: -1
10: -1

ANALYZING...
Analysis done.
Successful probes: 33
Unsuccessful probes: 1

Operation
I 1234
Insertion is successful. 1234 inserted.

Operation
R 1234
Removal is successful. 1234 removed.

Operation
S 1234
Search is successful. 1234 found after 13 probes.

-bash-4.2$ 
```

PART 3: Compare the empirical performance values (average number of probes for successful and unsuccessful searches as given by the analyze function) and the theoretical average number of probes that can be obtained using the formulas given in the course slides (according to the collision resolution scheme selected and the load factor for the resulting hash table after executing all insert and remove operations specified in the input text file). Briefly discuss your observations.

-FOR LINEAR PROBING-

successful search formula: $\frac{1}{2}[1 + 1 / (1 - \alpha)]$ where $\alpha = N / M$

unsuccessful search formula: $\frac{1}{2}[1 + 1 / (1 - \alpha^2)]$

-FOR QUADRATIC PROBING & DOUBLE HASHING-

successful search formula: $[-log_e(1 - \alpha)] / \alpha$

unsuccessful search formula: 1 / (1 - α)

Results:

No, my results do not agree with theoretical results completely. For instance, for the linear probing array example in my main function, I calculated that the result must be 1.9 for successful probes and 2.8 for unsuccessful probes theoretically (I also checked this from the course slides). However, my results are not compatible with these results. I expect that this occurs due to some experimental errors.