



CS 202

Homework Assignment 4

Section 2

Instructor: Çiğdem Gündüz Demir

Ahmet Işık

a.isik@ug.bilkent.edu.tr

21702226

Q2:

Part 1:

My HashTable class completely contains all the function declarations in the slides. Additionally, I create another enum to determine whether the location is OCCUPIED, EMPTY or DELETED, and an `void unsuccSearch(int index, int & probes)` method for the analyse method. In analyse method I use current values of HashTable as mentioned in the report for successful search. However, for unsuccessful as mentioned in the report, I initiate a search starting from at each array index and counts number of probes until reaching an empty location. Therefore, my `unsuccSearch` method takes the index value as a parameter and counts the number of probes until reaching an empty location. I call this function in analyse function `tableSize` times and find `numUnsuccProbes` for both LINEAR and QUADRATIC search. As mentioned, I determine `numUnsuccProbes = -1` if the search is DOUBLE. I used search function for successful search.

I designed my hashTable by creating a HashNode struct that keeps a int number for key value and Condition current that shows the current condition of that node (EMPTY, DELETED or OCCUPIED). Therefore, my hash array's type is HashNode.

My constructor initializes a hashArr as a new HashNode[tableSize] and fulfil each node's current condition as EMPTY. `numOfElements` variable that keeps the current number of elements in hashArr is initialized with 0, size is equal to tableSize and CollisionStrategy option is implemented by constructor's second parameter. In destructor, objects of hashArr are deleted.

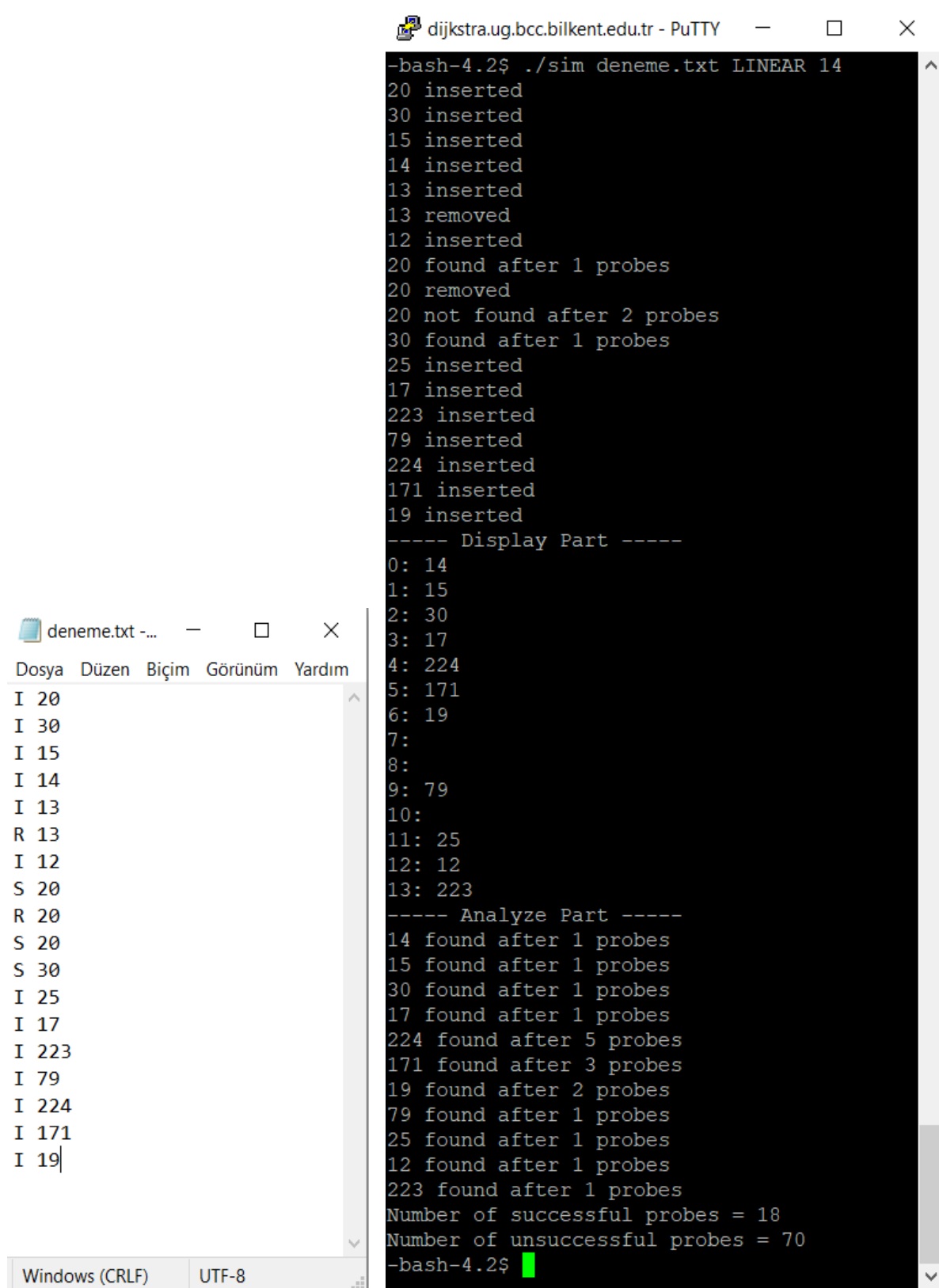
In insert method, firstly I checked whether insertion can be done by checking size. Then, according to the option, I code 3 different insertion algorithms. Each algorithm checks whether the item is inserted or not. All algorithms search EMPTY or DELETED item to insert the item. This loop ends if the searching time is equal to the size of hashArr because for all options, $h_i(key) = (hash(key) + f(i)) \bmod tableSize$ starts to repeat after i value exceeds the hashArr size. Therefore, it will make no sense to continue this loop after size times. So, my stopping condition is $i < size$ where $i = 0$ initially.

Almost same algorithm is used for remove function. This time, I assigned current condition as DELETED if the removing is successful and decrease `numOfElements`. But this time, loop continues until finding an EMPTY location to prevent complications (namely, it does not stop if the location is DELETED which treats like an OCCUPIED location). Same stopping condition is valid for deletion algorithm.

Search algorithm is almost similar with remove algorithm. It counts number of probes until finding the item. If it cannot, its stopping condition is same that $i < size$ where $i = 0$ initially.

Display function shows the item of each array location if location is OCCUPIED.

Part 2:



The image shows two windows from a Windows operating system. The top window is a PuTTY terminal titled "dijkstra.ug.bcc.bilkent.edu.tr - PuTTY". It displays the output of a program named "sim" run with arguments "deneme.txt" and "LINEAR 14". The output shows a sequence of operations on a graph, including inserting and removing nodes and finding paths. The bottom window is a text editor titled "deneme.txt -...". It shows the contents of the "deneme.txt" file, which lists a series of operations on nodes, such as "I 20", "I 30", "I 15", "I 14", "I 13", "R 13", "I 12", "S 20", "R 20", "S 20", "S 30", "I 25", "I 17", "I 223", "I 79", "I 224", "I 171", and "I 19".

```
dijkstra.ug.bcc.bilkent.edu.tr - PuTTY
-bash-4.2$ ./sim deneme.txt LINEAR 14
20 inserted
30 inserted
15 inserted
14 inserted
13 inserted
13 removed
12 inserted
20 found after 1 probes
20 removed
20 not found after 2 probes
30 found after 1 probes
25 inserted
17 inserted
223 inserted
79 inserted
224 inserted
171 inserted
19 inserted
----- Display Part -----
0: 14
1: 15
2: 30
3: 17
4: 224
5: 171
6: 19
7:
8:
9: 79
10:
11: 25
12: 12
13: 223
----- Analyze Part -----
14 found after 1 probes
15 found after 1 probes
30 found after 1 probes
17 found after 1 probes
224 found after 5 probes
171 found after 3 probes
19 found after 2 probes
79 found after 1 probes
25 found after 1 probes
12 found after 1 probes
223 found after 1 probes
Number of successful probes = 18
Number of unsuccessful probes = 70
-bash-4.2$
```

deneme.txt -...
Dosya Düzen Biçim Görünüm Yardım
I 20
I 30
I 15
I 14
I 13
R 13
I 12
S 20
R 20
S 20
S 30
I 25
I 17
I 223
I 79
I 224
I 171
I 19

Windows (CRLF) UTF-8

Picture (1)

Picture (2)

```
dijkstra.ug.bcc.bilkent.edu.tr - PuTTY
-bash-4.2$ ./sim deneme.txt QUADRATIC 14
20 inserted
30 inserted
15 inserted
14 inserted
13 inserted
13 removed
12 inserted
20 found after 1 probes
20 removed
20 not found after 2 probes
30 found after 1 probes
25 inserted
17 inserted
223 inserted
79 inserted
224 inserted
171 inserted
19 inserted
----- Display Part -----
0: 14
1: 15
2: 30
3: 17
4: 224
5: 19
6:
7: 171
8:
9: 79
10:
11: 25
12: 12
13: 223
----- Analyze Part -----
14 found after 1 probes
15 found after 1 probes
30 found after 1 probes
17 found after 1 probes
224 found after 3 probes
19 found after 1 probes
171 found after 3 probes
79 found after 1 probes
25 found after 1 probes
12 found after 1 probes
223 found after 1 probes
Number of successful probes = 15
Number of unsuccessful probes = 49
-bash-4.2$
```

Picture (3)

```
dijkstra.ug.bcc.bilkent.edu.tr - PuTTY
-bash-4.2$ ./sim deneme.txt DOUBLE 14
20 inserted
30 inserted
15 inserted
14 inserted
13 inserted
13 removed
12 inserted
20 found after 1 probes
20 removed
20 not found after 2 probes
30 found after 1 probes
25 inserted
17 inserted
223 inserted
79 inserted
224 inserted
171 inserted
19 inserted
----- Display Part -----
0: 14
1: 15
2: 30
3: 17
4: 224
5: 19
6: 171
7:
8:
9: 79
10:
11: 25
12: 12
13: 223
----- Analyze Part -----
14 found after 1 probes
15 found after 1 probes
30 found after 1 probes
17 found after 1 probes
224 found after 3 probes
19 found after 1 probes
171 found after 2 probes
79 found after 1 probes
25 found after 1 probes
12 found after 1 probes
223 found after 1 probes
Number of successful probes = 14
Number of unsuccessful probes = -1
-bash-4.2$
```

Picture (4)

Part 3

Empirical performance values:

LINEAR:

Successful search: Average number of probes = (number of successful probes / number of items)

By picture (2), Average number of probes = $18 / 11 = 1.63$

Unsuccessful search: Average number of probes = (number of unsuccessful probes / size of table)

Average number of probes = $70 / 14 = 5$

QUADRATIC:

Successful search: Average number of probes = (number of successful probes / number of items)

By picture (3), Average number of probes = $15 / 11 = 1.36$

Unsuccessful search: Average number of probes = (number of unsuccessful probes / size of table)

Average number of probes = $49 / 10 = 3.5$

DOUBLE:

Successful search: Average number of probes = (number of successful probes / number of items)

By picture (4), Average number of probes = $14 / 11 = 1.27$

Unsuccessful search: Average number of probes = -1

Theoretical values:

For the analysis of the average-case efficiency of hashing, the load factor α involves:

$\alpha = (\text{current number of items}) / \text{size of table}$

$\alpha = 11 / 14 = 0.78$ for the values given at pictures

LINEAR:

Successful search: for $\alpha = 0.78$

$$\frac{1}{2} \left[1 + \frac{1}{1-\alpha} \right]$$

$$= 2.77$$

Unsuccessful search: for $\alpha = 0.78$

$$\frac{1}{2} \left[1 + \frac{1}{(1-\alpha)^2} \right]$$

$$= 10.83$$

QUADRATIC & DOUBLE:

Successful search: for $\alpha = 0.78$

$$\left[\frac{1}{\alpha} (\log_e \frac{1}{1-\alpha}) \right] = \frac{-\log_e (1-\alpha)}{\alpha}$$

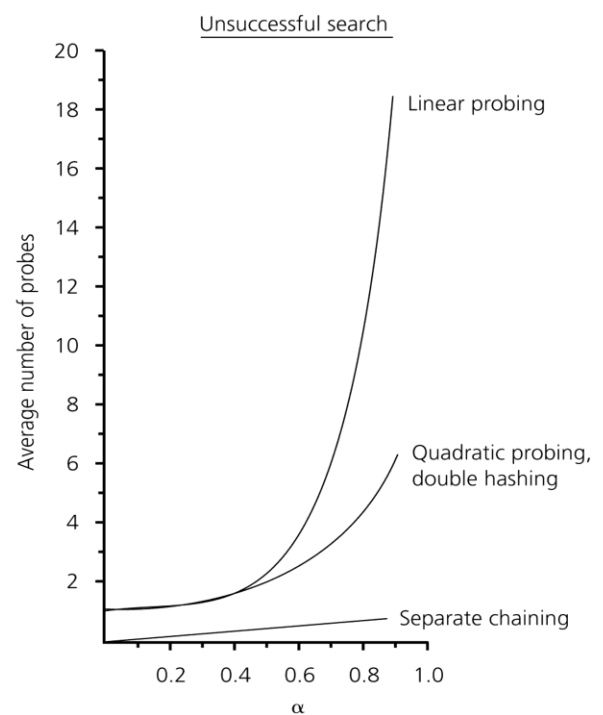
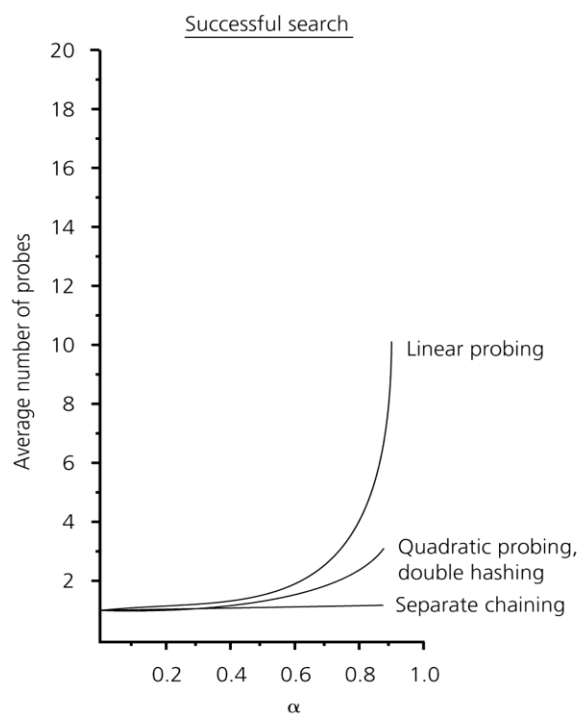
$$= 1.94$$

Unsuccessful search: for $\alpha = 0.78$

$$\frac{1}{1-\alpha}$$

$$= 4.54$$

Theoretical Graph taken by Course slides:



Comparison – Analyse:

I used the number of successful and unsuccessful probes gained by testing my function and calculate the empirical performance values. If we compare for each collision strategy option separately in both empirical and theoretical calculations, we can obviously observe that in both successful and unsuccessful search average number of probes is the highest for linear probing. Quadratic probing and double hashing require a smaller number of probes for searching than linear probing. Their values are too close.

We can see that in each collision strategy, average number of probes is greater in unsuccessful search than successful search. Additionally, my both empirical and theoretical values are confirmed by the theoretical graph table received by Lecture slides.

AVERAGE NUMBER OF PROBES TABLE WITH SIZE 14 & 11 CURRENT ELEMENTS				
	EMPIRICAL PERFORMANCE		THEORETICAL PERFORMANCE	
Collision Strategy	Successful	Unsuccessful	Successful	Unsuccessful
LINEAR	1.63	5	2.77	10.83
QUADRATIC	1.36	3.5	1.94	4.54
DOUBLE	1.27	-1	1.94	4.54