Design of the HashTable

The HashTable uses modular aritmatic rules respect to the size of the table when the key code of the values are calculated. The following rules are used in order to insert, remove and search any element to the Table.

- ## Linear Table:

  - ### Insertion:

    When insertion function is called for any integer value, it return false if the table is full. Otherwise, firstly take the mod of the particular value respect to the size of the table and check that wheter its key values is avaliable in the table or not. If it is avaliable in the table, insert it directly to its key value, otherwise check the next place in order to check its avaliablity and the same proccess is used until we check all place on to the Table. If there is no place, it means that it is full and automatically it returns falls.

```
while(arr[(key +i)%size]!=NULL)
{
    if (arr[(key +i)%size]->isDeleted)
    {
        arr[(key +i)%size]->setItem(newItem);
        arr[(key +i)%size]->setIsDeleted(false);
        return true;
    }
    i++;
    if(i == size){
        return false;
    }
}
HashTableItem* item = new HashTableItem(newItem);
arr[(key +i)%size] = item;
return true;
```

- **Remove:**

When remove function is called for any integer value, firstly its key value which is described in Insertion part is calculated and check to the this particular place of table respect to the key value. If it is null which means that this place is never used before, the function returns false. Otherwise if this place is empty but it has used for some other values or it is not empty, checking its next place. For the next place, same proccess is used. When all places on the table is checked and the value is not found function return false, otherwise the value is deleted from the table. For doing this seperation between null and deleted place, objects are used in the implementation part. When any object is removed, its isDelete feature becomes true. Therefore, while traversing the table, it can be understood that the place is used before or not.

```
for(int i = 0; i < size; i++)
{
    if(arr[(key + i)%size] == NULL)
    {
        return false;
    }
    if(arr[(key + i)%size]->getItem() == item){
        arr[(key + i)%size]->setIsDeleted(true);
        return true;
    }
}
return false;
```

- **Search:**

When the search function is called for any item, firstly the key value is calculated for particular item. The first step of searching is check that if the key place is null or not. If it is null, it means that the place is never be inserted before so the function returns false. Otherwise if the place is empty but used and removed or it is not empty, the next place is checked and same proccess is used. Its algorithm is almost same with the remove part of the table. By using this algorithm the check number is calculated in order to use it in analyze part of the table.

- **Quadratic Table**

  - **Insertion:**

    When the instertion funciton is called, firstly its key value is calculated with the same idea. The differences between Quadratic Table and Linear Table is, if the checking place is not avaliable its direcrly the next place is not checked in every step unlike the Linear Table. The places that are checked is calculated with respect to the square of the number of checking. It means that for every check, the number of checks' square is calculated, it added to the key value and new results key is checked that it is avaliable or not. The other difference is in Linear Table, every place (which equals to the size) is checked in order to find a null or empty place in order to insert the item. However, in Quadratic Table it is known that the half of the integers' square have the same value of the other half of the integers' square symmetrically in modular articmatic. Therefore the stopping case is not the size unlike Linear Table, it equals to size/2.

```
while(arr[(key + i*i)%size]!=NULL)
{

    if(arr[(key + i*i)%size]->isDeleted)
    {
        arr[(key + i*i)%size]->setItem(newItem);
        arr[(key + i*i)%size]->setIsDeleted(false);

        return true;
    }
    i++;
    if(i == (size/2) + 1)
        return false;
```

➢ Remove:

When the remove function is called, firstly its key value is calculated with the same idea. After that the same idea is used. Checking the all places with respect to the Quadratic table key rule until finding the item. When the item is found, it is deleted from the table with the same idea of Linear Table. If we encounter a null place while checking avaliable places, the funciton return false or if it is not found, the difference is same with the insertion function's difference. Stopping condition again equals to the size/2 unlike size because of the same idea which is described in the insertion part.

```
for(int i = 0; i <= size/2; i++)
{
    if(arr[(key + i*i)%size] == NULL)
    {
        return false;
    }
    if(arr[(key + i*i)%size]->getItem() == item){
        arr[(key + i*i)%size]->setIsDeleted(true);
        return true;
    }
}
return false;
```

➢ Search:

When the search function is called, again its key value is calculated with the same idea. After that same idea is used with the remove part. Checking all places with respect to the Quadratic table key rule until finding the item. When the item is found, function returns true, otherwise check the other particular places which are calculated with the key rule. Again stopping condition again equals to the size/2 because of the same idea.

- ## Double Hash Table:

  - ### Insertion:

    In the Double Hash Table the idea is similar to the linear table. The places are checked again one by one ,however, places are calculated with another hash function. In the implementation, the other hash function is reversing the integer value. Firstly the key value of the item is calculated like all other function and check the key values corresponding place. If the place is avaliable means empty or null, the item is inserted otherwise the second key valuse is calculated with respect to the second hash funtion. Every check step, the place is calculated as key plus the number of check times second key. It can be thought that the places are checked linearly, however the value does not increase one by one, it increases second key by second key in every step. The other proccesses are the same with the linearly insertion. The stopping condition eqauls to the size like linearly table because it is known that the size of the table will be prime number so every step is corresponding different place by the modular aritmatic rules.

  - ### Remove:

    When the remove function is called, the idea is same with the other ones. Firstly the checking the key value that it is the particular item or not. If it is, deleting the item from the list with the same implementation idea of Linear Table. Otherwise, checking the all places which are calculated respect to the second hash function. How to go to the next plave is described in the insertion part. If the item is found, remove it. If it is not found the function returns false. Its stopping condition is same with the linear Table. If any place equals to null it returns false, otherwise its check the all places on the table.

- **Search:**

In the search function, the idea is the same with the remove. The places are calculated with respect to the first and second key value linearly second key by second key. If there is a null value, the funciton return false, otherwise it search every places until finding the item. The idea of the search is the same with the other functions.

- **Analyze Function:**

In the anaylze function, firstly the number of success probes is calculated by search every item in the array one by one and sum the all number of probes values. When we divide this sum to the numer of the item, the avarage number of success probes is determined. Secondly, when the number of unseccessful probes is calculated, we are trying the numbers that not included in the Table starting from the zero to the number of size. If the number between them is in the table, we add the size to the this number until the map does not contain this number. The size is adding because when we add size to the number, the key value is not changed. The number of probes is given by the search method. Thus, sum all of these number and divide it to the size. Therefore, the avarage number of unsucessfull probes is calculated.

Muhammed Emre YILDIZ
21702825

Linear Table with 11 Size

```
25 cannot be removed.
25 cannot be removed.
25 cannot be found in 1 number of probes.
25 cannot be found in 1 number of probes.
21 cannot be removed.
21 cannot be found in 2 number of probes.
22 cannot be found in 1 number of probes.
32 cannot be removed.
37 cannot be removed.
34 cannot be removed.
10 cannot be found in 3 number of probes.
10 cannot be removed.
23 cannot be removed.
21 cannot be removed.
45 cannot be removed.
25 cannot be inserted.
22 is found in 1 number of probes.
---------------------------------
0:
1:
2:
3: 25
4:
5:
6: 17
7: 6
8:
9: 9
10: 43
---------------------------------
The size of the table is: 11
Average Successful Search: 1.2
Average Unsuccessful Search: 3.18182
```

```
I 32
R 25
R 25
S 25
R 32
S 25
R 21
S 21
S 22
I 7
I 17
I 43
R 32
I 22
R 37
R 34
I 25
S 10
R 7
R 10
R 23
I 9
R 21
I 33
R 45
I 25
I 45
S 22
R 33
R 45
I 6
R 22
```

Quadratic Table with 11 Size

```
34 cannot be removed.
33 is found in 3 number of probes.
35 cannot be found in 2 number of probes.
33 is found in 3 number of probes.
22 cannot be removed.
44 is found in 2 number of probes.
17 cannot be inserted.
52 cannot be removed.
47 cannot be found in 3 number of probes.
5 cannot be found in 1 number of probes.
---------------------------------
0:
1: 44
2: 12
3: 25
4:
5:
6: 17
7:
8: 8
9: 19
10:
---------------------------------
The size of the table is: 11
Average Successful Search: 1.5
Average Unsuccessful Search: 2.81818
```

```
I 11
I 35
I 22
I 33
R 34
S 33
R 22
I 44
R 35
S 35
S 33
R 22
I 52
I 12
R 11
S 44
I 17
R 33
I 17
R 52
I 8
I 25
R 52
S 47
S 5
I 19
```

Double Table with 11 Size

```
13 is found in 4 number of probes.
12 cannot be removed.
35 cannot be found in 6 number of probes.
14 cannot be removed.
21 cannot be inserted.
18 cannot be removed.
28 cannot be removed.
17 cannot be inserted.
11 cannot be found in 11 number of probes.
23 cannot be removed.
18 cannot be inserted.
--------------------------------
0: 31
1: 12
2: 24
3: 32
4: 17
5: 5
6: 16
7: 13
8: 28
9: 20
10:
--------------------------------
The size of the table is: 11
Average Successful Search: 3.1
Average Unsuccessful Search: -1
```

```
I 11
I 24
I 35
I 13
S 13
I 5
I 16
R 35
I 20
R 11
I 31
R 12
I 21
I 12
S 35
R 14
I 21
I 32
R 18
I 17
R 28
I 17
I 28
S 11
R 23
I 18
R 21
```

Part III

- **Linear Table**

Theoretical:

$$Load\ Factor\ \alpha: \frac{5}{11} = 0.454545$$

$$Avarage\ number\ of\ successful\ prob = \frac{1}{2}\left[1 + \frac{1}{1-\alpha}\right] = 1.4166$$

$$Avarage\ number\ of\ unsuccessful\ prob = \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)^2}\right] = 2.1805$$

Analyze Function Results:

$$Avarage\ number\ of\ successful\ prob = 1.2$$

$$Avarage\ number\ of\ successful\ prob = 3.18182$$

The values of the Theoretical and experimental results are similar to each other in successful prob. However, in the unseccessful prob the formula is given to avarage cases. When the input file contains similar key value items and less null values, in unsuccesful prob we check more places in order to return false. In remove and search parts of the report, the algorithm is described. In conclusion, alpha is calculated respect to the empty spaces ,however, search function operate with respect to the null spaces not empty spaces. That is the reason of the diffrence between these two values.

- **Quadratic Table**

  Theoretical:

$$Load\ Factor\ \alpha: \frac{6}{11} = 0.54545$$

$$Avarage\ number\ of\ successful\ prob = \frac{-log_e(1-\alpha)}{\alpha} = 1.4455$$

$$Avarage\ number\ of\ unsuccessful\ prob = \frac{1}{1-\alpha} = 2.2$$

Analyze Function Results:

$$Avarage\ number\ of\ successful\ prob = 1.5$$

$$Avarage\ number\ of\ successful\ prob = 2.8181$$

Again results are similar with each other. Because the values are uniformly distributed so Avarage number of successful prob is almost the same with theoretical result. However because of the same reason in unsuccessful result there is a small difference between these two results. The difference is less than the linear Table because Quadratic table is more optimal then the Linear Table.

- **Double Hash Table**

Theoretical:

$$Load\ Factor\ \alpha: \frac{10}{11} = 0.909090$$

$$Avarage\ number\ of\ successful\ prob = \frac{-log_e(1-\alpha)}{\alpha} = 2.6376$$

$$Avarage\ number\ of\ unsuccessful\ prob = -1$$

Analyze Function Results:

$$Avarage\ number\ of\ successful\ prob = 3.1$$

$$Avarage\ number\ of\ successful\ prob = -1$$

It can be said that again the results are similar with each other. However, the difference between them is increased in comparison with the Quadratic Table. Because, in the input file of Double Hash Table, the values are not uniformly distributed and it is much closer to the worst case than the Quadratic Table. In conclusion, these theoretical result formulas are used for uniformly distributed values. If the Table is regulated by considering the uniformly distribution, the experimental results would be more similar to theoretical ones.