ALGORİTMA ANALİZİ 3. ÖDEV

HASHING

Adı Soyadı: Büşra Medine GÜRAL

Öğrenci Numarası: 20011038

Mail: medine.gural@std.yildiz.edu.tr

Dersin Eğitmeni: M. Elif KARSLIGİL

1. Problem Tanımı

İlgili ödevde, kullanıcı isimlerini saklamak için bir hash tablosu oluşturulması gerekmektedir. Hash tablosu, Horner Kuralı kullanılarak sayı karşılıkları hesaplanacak ve çakışma problemini çözmek için double hashing yöntemi kullanılacaktır. Hash fonksiyonları bölme yöntemini kullanarak belirlenecektir. Ayrıca, silme işlemi sadece deleted değişkenini 1 yaparak gerçekleştirilecek ve bu sayede daha sonra aynı isimle arama yapılırsa "Bu kişi bulunmamaktadır." mesajı verilecektir. Kullanıcı tabloyu düzenlemek isterse, silinmemiş elemanlar yeni bir hash tablosuna taşınacaktır. Kullanıcı programı debug ve normal modda başlatabilecektir, böylece debug modda daha detaylı bir işlem akışı görebilecektir.

2. Problem Çözümü

Tabloya eklenecek kullanıcı isimlerini saklamak için bir hash tablosu oluşturulmaktadır. Bu tablonun boyutu, kullanıcıdan alınan eleman sayısı ve load factor kullanılarak belirlenmektedir. Eleman sayısı load factor'e bölünerek çıkan sonuçtan daha büyük veya eşit bir asal sayı tablo boyutu olarak seçilmektedir. Hash tablosu, çakışmaları önlemek için iki farklı hash fonksiyonunun kombinasyonu olan double hashing yöntemini kullanmakta ve hash fonksiyonlarını bölme yöntemiyle belirlemektedir. Her kullanıcı adının sayı karşılığı ise Horner Kuralı kullanılarak hesaplanmaktadır. Horner Kuralı için belirlenen sabit sayı 31'dir.

Program, kullanıcının seçtiği normal veya debug modda çalıştırılabilmektedir. Debug modda, işlem adımları daha detaylı bir şekilde ekrana yazdırılmaktadır. Her iki modda da kullanıcıya ekleme, arama, silme ve tabloyu düzenleme gibi işlemleri gerçekleştirebileceği bir alt menü sunulmaktadır. Kullanıcıdan alınan işleme göre ilgili fonksiyonlar çağrılmaktadır.

Ekleme işlemi, çakışma durumlarına göre uygun bir konum bulmak için double hashing yöntemi kullanılarak gerçekleştirilmektedir. Yerleştirilecek eleman için hesaplanan indeks değeri hash tablosunda kontrol edilmekte, eğer ilgili satır dolu ise değer tekrardan hash fonksiyonundan geçirilmektedir. Yapılan kontrollerin ardından yerleştirilecek satıra ulaşıldığında daha önceden o kişinin silinip silinmediğine bakılmaktadır.

Silme işlemi önce arama işlemi gerçekleştirilerek ilgili kullanıcının bulunması ve bulunduğu takdirde deleted değişkeninin değiştirilmesiyle gerçekleşmektedir. Kullanıcı aranırken ekleme işleminde olduğu gibi hash fonksiyonu yardımıyla yerleştirildiği göz bulunmakta, ardından o kişinin deleted niteliği kontrol edilmektedir.

Son olarak tablo aktarımında ise yeni bir hash tablosu oluşturularak başlangıç değerleri ayarlanmaktadır. Sonrasında eski tablonun her bir gözü taranarak silinmeyen elemanlar, ekleme işleminde yapıldığı gibi güncel tabloya sırasıyla ve yeni değerleriyle aktarılmaktadır.

3. Karşılaşılan Sorunlar

Horner kuralından dolayı çıkan büyük sayılar bir int tipindeki değişkenin boyutunu aştığından program, birçok kez çalışmayı durdurmaktaydı. Bu sorunun çözümü için hash tablosunun boyutu ile mod alınması ya da long int kullanılması gerekmekteydi. 2. tercih seçilerek ilgili fonksiyonda yapılan mod işlemi bu sorunu çözmüştür. Ek olarak rehash fonksiyonu, herhangi bir kişinin silinmediği durumda hata vermekteydi. Bunun sebebi tek for döngüsü içinde, yeni tablonun başlangıç değerlerinin atanması ve rehash işleminin gerçekleştirilmesinin denenmesiydi. Yeni tablo henüz oluşturulmamışken bir username'in yerleştirilmeye çalışılması programın çalışmasını durdurmaktaydı, ayrı döngüler ile bu sorun da çözüldü.

4. Karmaşıklık Analizi

Koddaki her bir fonksiyon için karmaşıklık incelendiğinde:

- firstHashFunc ve secondHashFunc fonksiyonlarında sadece mod alma işlemi gerçekleşmektedir. Sabit bir işlem olduğundan O(1) karmaşıklığa sahiptir.
- hornerMethod fonksiyonunda, parametre olarak verilen n uzunluklu karakter dizisinin her bir elemanı için matematiksel işlemler gerçekleştiğinden karmaşıklık O(n) olmaktadır.
- insertToHash fonksiyonu, minimum load factor'e bağlı olarak zaman karmaşıklığını O(1)'e indirgemektedir. Her bir elemanın birkaç çakışma olması durumunda dahi sabit bir adım sayısı ile tabloya yerleştiğinden karmaşıklık sabit olacaktır. Ancak en kötü durumda tablonun tamamı dolu olduğundan karmaşıklık O(n) olmaktadır.
  Hash algoritmalarındaki insert fonksiyonunun karmaşıklığının O(1) olmasının yanı sıra ödev için tasarlanan algoritmadaki insert fonksiyonunda Horner kuralının kullanılması sebebiyle karmaşıklık O(n)'e çıkmaktadır.
- searchInHash fonksiyonu, insert fonksiyonunda olduğu gibi load factor etkisiyle karmaşıklığı O(1) yapmaktadır ancak en kötü durumda, bir elemanın aranması için tablonun tamamı kontrol edildiğinden karmaşıklık O(n) olacaktır.
- rehash fonksiyonu, eski tablodaki silinmemiş tüm elemanları yeni tabloya aktardığından her bir eleman için insert işlemi yapmaktadır. Algoritmadaki insert işleminin karmaşıklığı O(n) olduğundan rehash işleminin karmaşıklığı O(n^2) olmaktadır.
- printHash fonksiyonu, hash tablosunu ekrana yazdırmak için tüm tabloyu dolaştığından karmaşıklığı O(n) olmaktadır.

rehash
input:
oldTable - the old hash table
tableSize - size of the new hash table
option – normal mode or debug mode

```
newTable = createTable(tableSize)
for i=0 to tableSize
   newTable[i].deleted = 0
   newTable[i].userName = ""
for i=0  to tableSize
   if oldTable[i].userName is not empty then
      if oldTable[i].deleted is not 1 then
         call insertToHash(newTable, tableSize, oldTable[i].userName, option)
   end if
end for
call freeMemory(oldTable)
return newTable
```

insertToHash
input:
table - hash table
tableSize - size of the hash table
name - name of the element to be inserted
option - normal mode or debug mode

```
i = 1
```

```
    hornerInt = hornerMethod(name, tableSize)
    hIndex1 = calculateFirstHashIndex(hornerInt, tableSize)
    hIndex2 = calculateSecondHashIndex(hornerInt, tableSize)
    hashIndex = hIndex1
    while table[hashIndex].userName is not empty and table[hashIndex].userName != name and i < tableSize
do
        hashIndex = (hashIndex + hIndex2) % tableSize
        i = i + 1
    end while
    if i >= tableSize then
        print("The insertion operation could not be performed because the table is full.")
    else
        if table[hashIndex].userName is empty then
            table[hashIndex].userName = allocateMemory(name)
            copyName(table[hashIndex].userName, name)
            print("Your element is placed at address {}.".format(hashIndex))
        else
            if table[hashIndex].deleted is 1 then
                table[hashIndex].deleted = 0
                print("Your element is placed at address {}.".format(hashIndex))
            else
                print("The insertion operation could not be performed because the element already exists in the
table.")
            end if
        end if
    end if
```

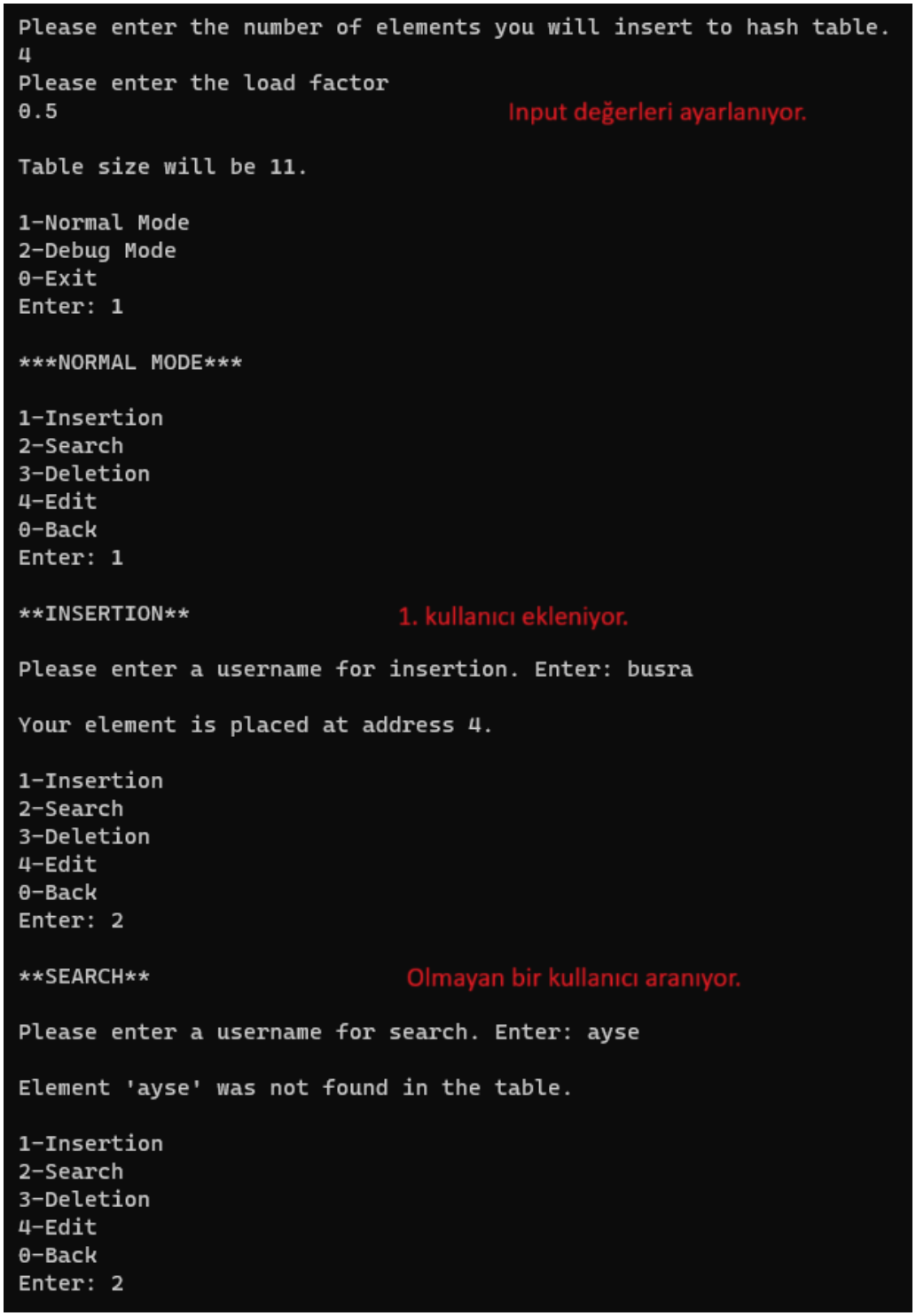
hornerMethod:
input:

```
    name - name of the element to be converted to int
    tableSize - size of the table

    res = name[0] - ASCIIValue('a') + 1
    i = 1
    while name[i] is not '\0' do
        res = res * HORNERS_CONST + (name[i] - ASCIIValue('a') + 1)
        res = res % tableSize
        i = i + 1
    end while
    return res
```

5. Ekran Görüntüleri

Aşağıdaki ekran görüntülerinde, koddaki prime fonksiyonları yorum satırlarından çıkarılmış haliyle program çalıştırılmıştır. Kullanıcı girişli ‘table size’ durumunda tablo boyutuna 11 değeri verildiğinde yine aynı sonuçlara ulaşılmaktadır.



1. kullanıcı aranıyor.

```
Please enter a username for search. Enter: busra

The element 'busra' is found at address 4.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 3

**DELETION**
```

Olmayan bir kullanıcı siliniyor.

```
Please enter a username for deletion. Enter: fatma

Element 'fatma' is not deleted because it does not exist in the table.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 0

1-Normal Mode
2-Debug Mode
0-Exit
Enter: 2
```

Normal moddan debug moduna geçiş yapılıyor.

```
***DEBUG MODE***

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 1

**INSERTION**
```

2. kullanıcı ekleniyor.

```
Please enter a username for insertion. Enter: ahmet

D: After Horner's Method: ahmet -> 3
D: h1('ahmet') = 3
D: h2('ahmet') = 4
D: h('ahmet', 0) = 3
D: The word 'ahmet' is placed at address 3.

Your element is placed at address 3.
```

```
3-Deletion
4-Edit
0-Back
Enter: 2

**SEARCH**
                                    1. kullanıcı aranıyor.
Please enter a username for search. Enter: busra

D: After Horner's Method: busra -> 4
D: h1('busra') = 4
D: h2('busra') = 5
D: The word 'busra' is found at address 4.
The element 'busra' is found at address 4.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 3

**DELETION**              2. kullanıcı siliniyor.

Please enter a username for deletion. Enter: ahmet

D: After Horner's Method: ahmet -> 3
D: h1('ahmet') = 3
D: h2('ahmet') = 4
D: The word 'ahmet' is found at address 3.
The 'ahmet' element at address 3 was deleted.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 1

**INSERTION**
                                    3. kullanıcı ekleniyor.
Please enter a username for insertion. Enter: fatma

D: After Horner's Method: fatma -> 0
D: h1('fatma') = 0
D: h2('fatma') = 1
D: h('fatma', 0) = 0
D: The word 'fatma' is placed at address 0.

Your element is placed at address 0.
```

```
1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 4

The table is edited.
```

Tablo aktarımı yapılıyor.

```
D: Username: fatma
D: Old Address: 0
D: Deleted: NO
D: After Horner's Method: fatma -> 0
D: h1('fatma') = 0
D: h2('fatma') = 1
D: h('fatma', 0) = 0
D: The word 'fatma' is placed at address 0.

Your element is placed at address 0.


D: Username: ahmet
D: Old Address: 3
D: Deleted: YES
```

2. kişi daha önceden silindiği için yeni adres değeri yok.

```
D: Username: busra
D: Old Address: 4
D: Deleted: NO
D: After Horner's Method: busra -> 4
D: h1('busra') = 4
D: h2('busra') = 5
D: h('busra', 0) = 4
D: The word 'busra' is placed at address 4.

Your element is placed at address 4.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 1
```

```
**INSERTION**
Please enter a username for insertion. Enter: ahmet

D: After Horner's Method: ahmet -> 3
D: h1('ahmet') = 3
D: h2('ahmet') = 4
D: h('ahmet', 0) = 3
D: The word 'ahmet' is placed at address 3.

Your element is placed at address 3.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 4

The table is edited.

D: Username: fatma
D: Old Address: 0
D: Deleted: NO
D: After Horner's Method: fatma -> 0
D: h1('fatma') = 0
D: h2('fatma') = 1
D: h('fatma', 0) = 0
D: The word 'fatma' is placed at address 0.

Your element is placed at address 0.


D: Username: ahmet
D: Old Address: 3
D: Deleted: NO
D: After Horner's Method: ahmet -> 3
D: h1('ahmet') = 3
D: h2('ahmet') = 4
D: h('ahmet', 0) = 3
D: The word 'ahmet' is placed at address 3.

Your element is placed at address 3.


D: Username: busra
D: Old Address: 4
D: Deleted: NO
D: After Horner's Method: busra -> 4
D: h1('busra') = 4
D: h2('busra') = 5
D: h('busra', 0) = 4
D: The word 'busra' is placed at address 4.

Your element is placed at address 4.
```

Silinen 2. kişi yeniden ekleniyor.

Tablo aktarımı yapılıyor.

Silinen 2. kişi yeniden eklendiğinden yeni tabloda adresi var.

```
**INSERTION**

Please enter a username for insertion. Enter: omer

D: After Horner's Method: omer -> 6
D: h1('omer') = 6
D: h2('omer') = 7
D: h('omer', 0) = 6
D: The word 'omer' is placed at address 6.

Your element is placed at address 6.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 1

**INSERTION**

Please enter a username for insertion. Enter: melih

D: After Horner's Method: melih -> 8
D: h1('melih') = 8
D: h2('melih') = 9
D: h('melih', 0) = 8
D: The word 'melih' is placed at address 8.

Your element is placed at address 8.

1-Insertion
2-Search
3-Deletion
4-Edit
0-Back
Enter: 1

**INSERTION**

Please enter a username for insertion. Enter: ozlem

D: After Horner's Method: ozlem -> 6
D: h1('ozlem') = 6
D: h2('ozlem') = 7
D: h('ozlem', 0) = 6
D: The word 'ozlem' is not placed at the address 6.
D: The word 'ozlem' is placed at address 2.

Your element is placed at address 2.
```

4. kullanici ekleniyor.

5. kullanici ekleniyor.

6. kullanici eklenirken 6. adres dolu olduğundan yeniden hash indeksi hesaplanıyor.

Debug moddayken olmayan bir eleman arandığında: