Problem Tanımı:

Elimizde tablo var ve boyutunu kullanıcıdan aldığımız load factor ve gireceği input sayısına göre belirlenir. Tablonun boyutu asal olmalıdır. Open address metodu ile tablo oluşturulur. Elemanları da Horner Kuralı kullanılarak hesaplanır. Kullanıcı tabloya eleman ekleyebiliyor, eleman silebiliyor, arayabiliyor, tabloyu yeniden şekillendirebiliyor ve tabloyu listeleyebiliyor.

Problem Çözümü:

Kullanıcıdan gerekli girişler alındıktan sonra tablonun boyutu belli oluyor. Her kayıt için kullanıcı adı(userName) ve silinip silinmediği bilgisi(deleted) için struct yapısı oluşturulur. Tanımladıktan sonra tablonun boyutu belli olan boyut kadar malloc() ile yer açılır. Ondan sonra başlangıç değerleri verilir. En başta tablonun boş sayılacağı için her indeskteki userName = '\0' ve deleted = 1 olarak ilişkilendirilir. Gerekli işlemler yapılmadan önce girilen ismin anahtar olarak karşılığı var. Horner Kuralı ile anahtar değeri hesaplanılır. Bu kuralda asal sayımızın değeri 31 olarak belirlendi. Hash fonksiyonları anahtar değerini kullanarak indeks hesaplanılır. Şekil-1'de kullanılan hash fonksiyonları mevcut. Çakışmayı en aza indirmek için double hashing metodu kullanılır.

$$h(key, i) = (h1(key) + i*h2(key)) \% m$$
$$h1(key) = key \% m$$
$$h2(key) = 1 + (key \% m2)$$
h2 fonksiyonundaki m2 değeri: m2 = m-2

m: tablo boyutu, key: Horner Kuralı ile belirlenen anahtar değeri

Şekil-1

Burada verilen i değerinin başlangıç değeri sıfırdır. Hash fonksiyonu ile hesaplanan indeks değeri bazen bizim ihtiyacımızı karşılamıyor. Ondan dolayı i değişkenin değerini bir artırarak yeni indeks değer hesaplanılır. Duruma göre i artırılabilir ama eğer i değeri tablo uzunluğundan eşit veya fazla olmaya başlarsa bir anlamı kalmaz. Modunu aldığımız için zaten hesapladığı indeks değerler daha önceden hesaplanılmıştı.

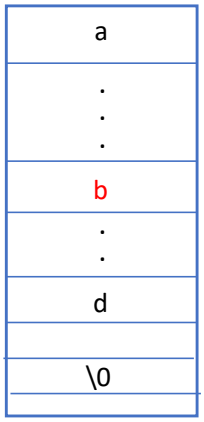
Öncelikle işlemlerin algoritmaları tasarlanırken şu karar verildi. Eleman eklemek ya da silmek isteniliyorsa arama işlemini kullanılarak ekleme ya da silme işlemlerine yardımcı olunabileceği düşünüldü. **Arama işlemi** tasarlanırken 3 işlem için de düşünülerek karar verildi.
Ondan dolayı önce arama işlemini ele alalım. Tablomuzun bir kısmı dolu olsun. Misal 'Sado' değerini aramak istenilirse önce anahtar değeri hesaplanılarak sonra da hash mekanizması ile hangi tablonun indeksine bakılacak ona karar verilir. Olasılıkları ele alalım:
1)Eğer ilk hash mekanizması ile hesaplanan indeksteki userName değeri '\0' ise demek ki daha önce burada herhangi ekleme ya da silme işlemi yapılmamış. Bundan sonraki indekslere bakmaya gerek kalınmadı. 'Sado' elemanı tabloda bulunulmuyor. Bu sonuca varılır. Arama fonksiyonun dönüş değeri hesaplanan indeks olur.
2)Eğer ilk hash mekanizması ile hesaplanan indeksteki userName değeri ile 'Sado' değeri ile aynı değilse hash mekanizması tarafından yeni indeks üretilir. Bu adım userName değeri '\0' ya da 'Sado' olana kadar devam eder.

2.1)userName değeri '\0' değeri ise 'Sado' elemanı tabloda bulunulmuyor .

2.2)userName değeri 'Sado' değeri ise deleted değerine bakılır. Eğer 1 ise demek ki tabloda sadece fiziksel olarak var. Yani aslında tabloda yok. Eğer 0 ise tabloda 'Sado' elemanı bulunur.

Arama fonksiyonun dönüş değeri en son hesaplanan indeks olur.

3)Bazı özel durumları şekil üzerinde anlatalım:
3.1)Elimde birbirinden farklı a, b, c, d değerleri var ve hepsi de aynı anahtar değerine sahip. a ,b ,d değerleri tabloya eklenmiş sonra da b değeri silinmiş. Biz de c değerini tabloda aramaya çalışalım.



Hash Table

İlk adımda c değeri ve a değeri karşılaştırılıyor. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

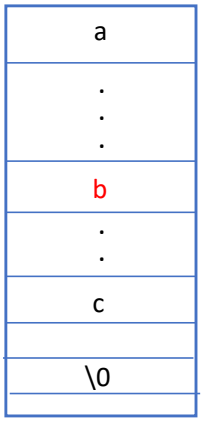
İkinci adımda b değeri ve c değeri karşılaştırılıyor. Ek olarak b değeri sadece fiziksel olarak bulunduğu için indeks akılda tutuluyor. Yapacağımız hamle ileride ekleme/silme işlemleri için lazım olacak. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

Üçüncü adımda d değeri ve c değeri karşılaştırılıyor. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

Dördüncü adımda '\0' değeri ve a değeri karşılaştırılıyor. Tablonun geçerli indeksi boş olduğu için şu anlaşılıyor. Bundan sonra c indeksi olamayacağı için artık ilerlememize gerek kalmadı.

En sonunda dönen değer aklımızda tuttuğumuz değer(sadece fiziksel olarak bulunan b değerinin adresi) olacak.

3.2)Elimde aynı değerler olsun yine . Bu sefer a ,b ,c değerleri tabloya eklenmiş sonra da b değeri silinmiş. Biz de tabloda var olan c değerini tabloda aramaya çalışalım.



Hash Table

İlk adımda c değeri ve a değeri karşılaştırılıyor. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

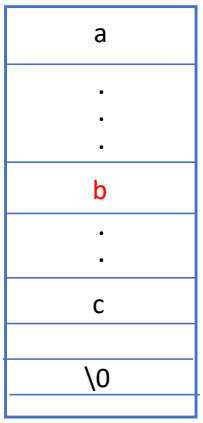
İkinci adımda b değeri ve c değeri karşılaştırılıyor. Ek olarak b değeri sadece fiziksel olarak bulunduğu için indeks akılda tutuluyor. Yapacağımız hamle ileride ekleme/silme işlemleri için lazım olacak. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

Üçüncü adımda c değeri ve c değeri karşılaştırılıyor. İkisinin değeri aynı olduğu için hash fonksiyonu yeni değer üretmez.

En sonunda da c değerinin tablodaki indeks değeri dönülür.

**Değinilmek istenilen eğer sadece fiziksel olarak var olan ilk değer hafızaya alınır ama sonradan aynı değer ile karşılaşılmışsa en son hesaplanan hash fonksiyonun ürettiği sayı dönülür.

3.3)Elimde aynı değerler olsun yine . Bu sefer a ,b ,c değerleri tabloya eklenmiş sonra da b değeri silinmiş. Biz de tabloda b değerini tabloda aramaya çalışalım.



Hash Table

İlk adımda c değeri ve a değeri karşılaştırılıyor. İkisinin değeri aynı olmadığı için hash fonksiyonu yeni değer üretiliyor. Sıradaki indekse gidiliyor.

İkinci adımda b değeri ve b değeri karşılaştırılıyor. Ama tabloda sadece fiziksel olarak var. İkisi de aynı olduğu için hash fonksiyonu yeni değer üretmez.

Dönüş değer olarak da sadece fiziksel olarak bulunan b değeri özelleşmiş şekilde döner.
Dönüş değeri = (-1) * (b değerinin adresi) – 2
İleride lazım olacak.

**Değinilmek isteyen en eğer sadece fiziksel olarak var olan bir değeri arayıp bulunursa onun özelleşmiş şekilde dönülür. Kesin negatif değer dönülür

Eleman ekleme işlemini ele alalım. Öncelikle tabloda eleman varsa ekleme yapılamaz. Eleman eklenileceği zaman sadece fiziksel olarak tabloda bulunuluyorsa deleted = 0 yapılmalı.
Eleman ekleme işlemi yapılırken arama algoritmasından dönen indeks değerlerini kullanıyor olacağız. Ondan sonra uygulamadan dönen değerlere göre ekleme yapılır ya da yapılmaz.
Eğer dönen değer pozitif ise o değer tabloda hemen var diyemeyebiliriz. Öncelikle tablodaki indeks ile eklenmek istenen değer aynı ise tabloda var olunduğu için ekleme yapılamaz. Eğer değerler farklı ise ekleme yapılır. Arama algoritmasında dönen değer aynı zamanda hafızada tutulan değer olabiliyordu. **Hafızada tutulan değer**, eğer aranan eleman yok ise ilk uygun eklenebilecek indeksi hafızada tutuluyordu.
Eğer dönen değer negatif ise tek durum bulunur. O da tabloda sadece fiziksel olarak bulunan değerin özelleşmiş şekilde döner. Sonra özelleşmiş şekilde dönen değerden gerçek indeks bulunulup deleted kısmı = 0 yapılır.

Eleman silme işini ele alalım. Tabloda var olan eleman silinebilir. Olmayan eleman silinemez. Silme işleminde eğer tabloda silinmek istenen değer varsa deleted = 1 yapılır. Yani sadece fiziksel olarak tabloda bulunur. Arama fonksiyonundan dönen değer ile silme işlemlerini ele alınacak.
Eğer dönen değer negatif ise silinmek istenen eleman yoktur.
Eğer dönen değer pozitif ise silinmek istenen değer olunabilir. Öncelikle kontrol edilmeli. Eğer silinmek istenen değer ile indeksteki değer aynı değilse silinmez.
Aksine aynı ise deleted = 1 yapılır.

Tabloyu yeniden düzenlemek işlemini ele alalım. Öncelikle yeniden aynı boyutta bir tablo elde edilir. Sonra eski tablonun ilk değerinden başlanılır. Eğer eski tabloda boş ise herhangi bir işlem yapılmaz yeni tablo için. Eğer eski tabloda değer dolu ise eleman ekleme fonksiyonu ile yeni tabloya eklenilir.
Tablo boyutu kadar devam edilir.

Karşılaşılan Sorunlar:

Öncelikle Horner Kuralı ile anahtar değeri hesaplanırken çok büyük sayılar elde ediliyordu. Ondan dolayı anahtar değerini direkt hesaplanılmadı. Hash mekanizmasında isim karakter karakter ilerlerken asal sayının üssünü almadan önce asal sayının modunu alıp öyle karesi alındı. Ondan sonra her harfin ASCII karşılığı olan sayının modu alındı. Ondan sonra elimizdeki iki sayı toplanıp bir daha mod alındı. Böylelikle küçük sayılarla uğraşılmış olundu. Integer tipi bize yeterli olundu.

Başka bir sorun da ekleme, silme ve arama işlemi için hepsine ayrı ayrı kod yazılırsa çok uzayacağı düşünüldü. Silme ya da ekleme işlemi yapmak için aslında arama işlemine benzer işlemler yapılır. Ondan dolayı Arama fonksiyonu 3 işleme de hizmet edilmesi gerekiyordu. Dönen değerlerin anlamına göre işlemler yapıldı ya da yapılmadı. Silme ve ekleme fonksiyonları ondan dolayı kısa olundu.

Tabloya ekleme yapılmak istenirken tabloda eleman varsa ekleme yapılmamalı. En başta tüm tablo geziliyordu. O zaman hash mekanizmasının ne anlamı kaldı diye düşünüldü. Sonra şu fark edildi. Eğer ben eleman arıyorsam ve hash mekanizması ile üretilen indeksteki değer '\0' ise demek ki bu indekste herhangi bir ekleme ya da silme işlemi yapılmadığına göre bundan sonra da o elemanın var olma ihtimali kalmadığı fark edildi. Ondan en başta hash tablosu ilişkilendirilirken deleted = 1 ve userName = '\0' yapıldı

Silme işleminde ise tabloda sadece fiziksel olarak varsa aslında tabloya ekleme yapılmayacak deleted = 0 yapılmalı. Bundan dolayı arama algoritmasını ona göre tasarlamam gerektiği düşünüldü. Eğer arama sırasında aranılan değer tabloda varsa normalde pozitif değer dönülmesi gerekir ama son adımlarda deleted değişkeni 1 ise özelleşmiş şekilde dönülür.

Algoritma Karmaşıklığı:

Ekleme, silme ve arama işlemleri arama fonksiyonuna bağlı olduğu için aşağıda arama fonksiyonun sözde kodu verilip onun üzerinden anlatılacaktır.

Input:

hashTable: isim değerlerini içeren struct dizisinin adresi

name: aranacak isim

m: struct dizisini boyutu

Output:

Negatif ya da pozitif indeks değerleri döner.

Negatif değer çıkarsa demek ki tabloda sadece fiziksel olarak o değer var. Özelleşmiş şekilde o değerin tablodaki adresi döner.

Pozitif değer çıkarsa:

1) Arama algoritmasını ben aynı zamanda eleman eklemek için de kullanılıyor. Eğer dönen indeks değer aranan değerle uyuşmuyorsa demek ki ben tablonun o indeksine ekleme yapılabilir. Aranılan değer ile indekteki değer bir olmadığına göre silme işlemi ve arama işlemleri başarısız olur.

2)Eğer aranılan eleman ile indeksteki eleman bir ise o zaman arama ve silme işlemi başarılı olur. Ekleme işlemi haliyle başarısız olur. Var olan eleman eklenemez.

```
1   int searchUserName(Node *hashTable, char *name, int m)
2       i <-- 0
3       hashValue <-- hEval(name, m, i)
4       first <-- m;
5           while( strcmp( hashTable[hashValue].userName , name ) != 0  and i < m)
6               if hashTable[hashValue].userName[0] = '\0'
7                   if first = m
8                       return hashValue
9                   ENDIF
10
11                  return first
12              ENDIF
13
14              if(hashTable[hashValue].deleted = 1 and first = m)
15                  first <-- hashValue;
16              ENDIF
17              i <-- i + 1
18              hashValue <-- hEval(name, m, i)
19          ENDWHILE
20
21          if i >= m
22              return first
23          ENDIF
24
25          if hashTable[hashValue].deleted = 1
26              return ( hashValue * -1) - 2
27          ENDIF
28
29          return hashValue;
30  ENDWHILE
```

Yukarıda verilen sözde kod değinilen işlemlerin ana karmaşıklığını oluşturur. Eğer ben eleman aramak istiyorsam en iyi ve ortalama ihtimallerde O(1) zaman karmaşıklığında eleman ekleyebilirim. En iyi ihtimalle hash mekanizması ile ilk hesaplanan indeks değerinde aramak istediğim değer bulunursa sabit sayıda bulmuş olunur. Ortalama ihtimallerde ise birazcık daha fazla hash mekanizması ile indeks hesaplanır ama N, kullanıcının girdiği input sayısı , sayısının yanında sabit sayı olarak kabul edilir.

Eğer tabloda var olan değerler tablonun boyutuna yakınsa ve hesaplanan indekslerde hep başka değerler varsa N sayısına yakın olacağı için O(N) olacaktır.

Ekleme ve silme işlemlerinde de aynı mantık olunduğu için arama işlemi detaylı açıklanmıştır.

Çıktılar:

```
Load Factor: 0.8
Kac tane giris olacak: 5
Tablonun boyutu(Onerilen = 7): 7
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod
1
Ismi giriniz: sado
sado ismi 5. adrese eklendi
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
3
1)Normal Mod
2)Debug Mod
2
Ismi giriniz: selda
        Adimlar
h1(selda) = 5
h2(selda) = 2
h(key, i) = (h1(key) + i*h2(key)) mod m, i = 0
5. adreste selda kelimesi bulunamadi. (key : 5, user name: sado deleted: 0; i = 0)
0. adreste selda kelimesi bulunamadi. (key : 0, user name:  deleted: 1; i = 1) Bu adreste herhangi eleman yok.
0. adres donus yapacak. (user name:  deleted: 1; i = 1).
Uygun islem gerekli kontrol dogrultusunda yapilir/yapilmaz.
selda ismi bulunamadi
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod
2
Ismi giriniz: selda
        Adimlar
h1(selda) = 5
h2(selda) = 2
h(key, i) = (h1(key) + i*h2(key)) mod m, i = 0
5. adreste selda kelimesi bulunamadi. (key : 5, user name: sado deleted: 0; i = 0)
0. adreste selda kelimesi bulunamadi. (key : 0, user name:  deleted: 1; i = 1) Bu adreste herhangi eleman yok.
0. adres donus yapacak. (user name:  deleted: 1; i = 1).
Uygun islem gerekli kontrol dogrultusunda yapilir/yapilmaz.
selda ismi 0. adrese eklendi
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
2
1)Normal Mod
2)Debug Mod
2
Ismi giriniz: sado
        Adimlar
h1(sado) = 5
h2(sado) = 4
h(key, i) = (h1(key) + i*h2(key)) mod m, i = 0
5. adres donus yapacak. (user name: sado deleted: 0; i = 0).
Uygun islem gerekli kontrol dogrultusunda yapilir/yapilmaz.
sado ismi 5. adresten silindi.
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
3
1)Normal Mod
2)Debug Mod
1
Ismi giriniz: selda
selda ismi 0. adreste bulundu.
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
2
1)Normal Mod
2)Debug Mod
2
Ismi giriniz: selda
        Adimlar
h1(selda) = 5
h2(selda) = 2
h(key, i) = (h1(key) + i*h2(key)) mod m, i = 0
5. adreste selda kelimesi bulunamadi. (key : 5, user name: sado deleted: 1; i = 0)
0. adres donus yapacak. (user name: selda deleted: 0; i = 1).
Uygun islem gerekli kontrol dogrultusunda yapilir/yapilmaz.
selda ismi 0. adresten silindi.
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod
2
Ismi giriniz: selda
        Adimlar
h1(selda) = 5
h2(selda) = 2
h(key, i) = (h1(key) + i*h2(key)) mod m, i = 0
5. adreste selda kelimesi bulunamadi. (key : 5, user name: sado deleted: 1; i = 0)
-2. adres donus yapacak. (user name: selda deleted: 1; i = 1).
Uygun islem gerekli kontrol dogrultusunda yapilir/yapilmaz.
Arama ve silme islemi gerceklecesekse basarisiz olur. Silinmis bir isim eklenecekse basarili olur
selda ismi 0. adrese eklendi
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod
1
Ismi giriniz: sehmus
sehmus ismi 6. adrese eklendi
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod

1
Ismi giriniz: sakarya
sakarya ismi 1. adrese eklendi
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
5
        HashTable Degerler
adress: 0 name: selda
adress: 1 name: sakarya
adress: 6 name: sehmus
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
1
1)Normal Mod
2)Debug Mod
1
Ismi giriniz: yunus
yunus ismi 2. adrese eklendi
```

```
        Islemler
1)Ekleme
2)Silme
3)Arama
4)Tabloyu duzenlemek
5)Tabloyu Listele
6)Cikis
4
        Tablo Duzenlenmesi
Eski tablo: (adres: 0, isim: selda) ve yeni tablo: (adres: 5, isim: selda)
Eski tablodaki ve yeni tablodaki yeri ayni. (adres: 1, isim: sakarya)
Eski tablo: (adres: 2, isim: yunus) ve yeni tablo: (adres: 6, isim: yunus)
3. adreste daha onceden herhangi bir isim eklenmemis ve yokmus
4. adreste daha onceden herhangi bir isim eklenmemis ve yokmus
sado ismi daha onceden 5. adreste bulunuyormus ama silindigi icin yeni tabloda yer alamaz.
Eski tablo: (adres: 6, isim: sehmus) ve yeni tablo: (adres: 3, isim: sehmus)
```