Hashing

Bir liste ya da dizide eleman aranacağı ya da eleman ekleneceği zaman o elemanın var olup olmadığının kontrolü sıra sıra yapılmaktadır. Bu işlem çok büyük veri kümelerinde fazla zaman almaktadır. Bunun yerine kullanılacak optimizasyonlar arama zamanını düşürebilecektir. Hashing yöntemi de bunlardan biridir.

Veri Yapısı	Ekleme İşlemi	Arama İşlemi
Sıralı Olmayan Dizi	O(1)	O(n)
Sıralı Dizi	O(n)	O(logn)
Liste	O(n)	O(n)
Dengesiz İkili Arama Ağacı	O(n)	O(n)
Dengeli İkili Arama Ağacı	O(logn)	O(logn)

* Yukarıdaki karmaşıklık değerleri her durumda garanti edilen değerlerdir.

Tanım: Bir hash fonksiyonu yardımı ile elde edilen anahtarın gösterdiği yere eleman eklenir ve aynı şekilde arama da bu fonksiyonun vereceği anahtar değeri ile gerçekleştirilecektir. Hashing bir dizi ile gerçekleştirildiği düşünülürse, hashing fonksiyonun döndürdüğü değer dizinin indeks değeri olacaktır.

Hash fonksiyonu nasıl olmalıdır sorusunun cevabı, hızlı ve kolay hesaplanabilir olmalıdır. Hash fonksiyonun içerisinde rastgelelik bulunamaz çünkü bir elemanın hash değeri her hesaplamada aynı çıkmalıdır.

Örnek Hash Fonksiyonu:

h(x) = x%5;

h(1) = 1, h(5) = 0, h(12)=2, h(103)=3

```
int Hash(int deger,int diziUzunluk){
   return deger % diziUzunluk;
}
```

Örnek Hash Fonksiyonu: Eğer dizide string ifadeleri tutuyorsak bu durumda string ifadeye uygun bir hash fonksiyonu belirlemeliyiz. İlk etapta akla ASCII kodlarından yararlanmak gelebilir.

“abcd” => 97+98+99+100 = 394

Elde edilen değer 394 olabileceği gibi çok uzun string’lerde çok daha büyük değerlerde çıkabilir. Bu durumda böyle büyük indeks değerleri ile uğraşmak doğru olmayacaktır. Yapılabilecek bir çözüm değerin, dizi boyutu kaç ise modunu alıp çıkan değeri indeks olarak kabul etmek olabilir.

Dizi uzunluğu = 10 ise

Hash(“abcd”) = 394%10 = 4

4. indekse bu değer eklenir.

```
int ASCIIDonustur(string deger)
{
   int asciiDeger=0;
   for (int i = 0; i < deger.length(); i++)
   {
      asciiDeger += int(deger.at(i));
   }
   return asciiDeger;
}

int Hash(string deger,int diziUzunluk){
   int ascii = ASCIIDonustur(deger);
   return ascii % diziUzunluk;
}
```

Çeşitli veri kümeleri karşımıza çıkabilir bu durumda hash fonksiyonuna hangi değerin sokulacağı iyi seçilmelidir. Örneğin Türkiye'deki kişilerden oluşan bir veri kümesinde T.C. Kimlik numarasını almak doğru bir seçim olacaktır. Ama örneğin doğum yılı aynı indekste çakışacak birçok kişi anlamına gelecektir. Bir başka örnekte telefon numaraları seçilecekse alan kodunun seçilmesi iyi hash değerlerinin elde edilemeyeceği anlamına gelmektedir. Bunun yerine alan kodunun dışındaki kısım seçilebilir.

Çarpışma

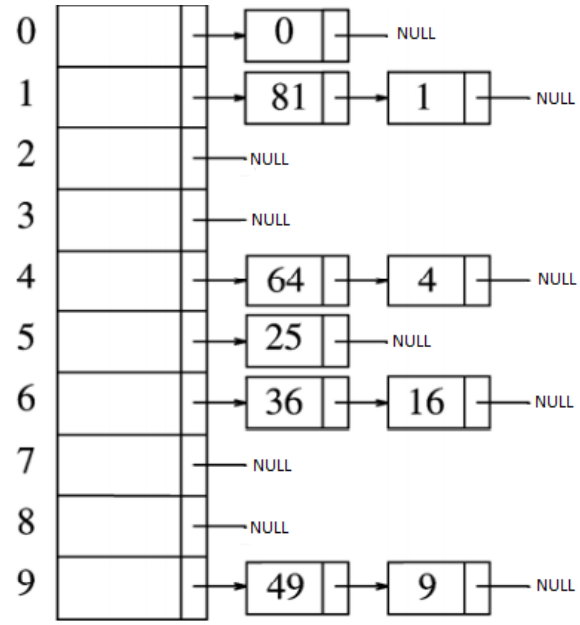
Dizi boyutunun küçük olması ve aynı hash anahtarını veren değerler aynı indekse konulmaya çalışıldığında çarpışma denilen durum ortaya çıkar. Çarpışma durumunu ortadan kaldırmak için birkaç yöntem bulunmaktadır. Burada bir kaçından bahsedilecektir.

- Zincirleme
- Açık Adresleme
- İkili Hashing
- Ve diğerleri

Zincirleme (Chaining)

Çarpışma olan değerler aynı indeksteki bir liste içerisinde tutulur. Örneğin aşağıdaki hash tablosuna sayılar "0, 1, 4, 9, 16, 25, 36, 49, 64, 81" şeklinde gönderiliyor. Dizimizin uzunluğu 10 olduğu için hash fonksiyonumuz aşağıdaki gibi olacaktır.

h(sayi) = sayi mod10 => örnek h(81) = 81 mod10 = 1 (81 sayısı 1. İndekste tutulur.)

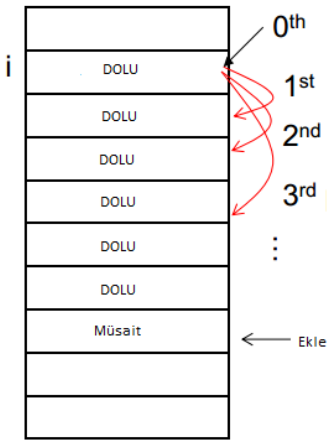


Liste bulunmasaydı hash fonksiyonu hesaplandıktan sonra bir adımda ekleme ya da arama yapılır. Fakat liste işin içerisine girdiğinde ekleme ve arama işlemi o indekste listenin uzunluğuna bağlı olarak değişecektir.

Önemli: Burada dizinin uzunluğu bize bir ipucu vermektedir. Seçilecek olan sayı aynı indekste liste uzunluğunu en aza indirecek şekilde seçilmelidir. Bu noktada asal sayılar iyi sonuçlar elde etmek için seçilecek sayılardır. Tabi ki burada dizi uzunluğunu da büyük seçmek gerekecektir.

Açık Adresleme (Open Adressing)

Çarpışma gerçekleştiğinde en yakın ileri bir indeks bulunup eleman oraya eklenir. Örneğin aşağıdaki hash tablosunda bir elemanın hash değeri 0 çıkmış fakat 0 ve ileriki indeksler dolu olduğu için müsait yazan yere kadar ötelenmiştir.



Buradaki sıkıntı dizi uzunluğu küçük ise dizi çabuk dolacak ve eleman eklemeye yer kalmayacaktır. Eldeki veri kümesine göre dizi boyutu belirlenmeli, çok mantıksız olabilecek durumlarda farklı yöntemler kullanılmalıdır (örneğin zincirleme gibi).

```
int Hash(int deger,int diziUzunluk){
   return deger % diziUzunluk;
}
int fonk(){
   int hashDeger = Hash(150,23);
   int i= hashDeger;
   for( ; dizi[i] != NULL; i++); //Boş olan yere kadar gidilir.

  dizi[i] = 150;
}
```

İkili Hashing (Double Hashing)

Bu yöntemde iki kere hash işlemi yapılır. Bunun amacı belli indekslerdeki kümelemenin önüne geçmektir. İlk hash yapılıp yerleşeceği indeks değeri belli olduktan sonra eğer o indeks değeri dolu değilse direk eklenir. Dolu ise ikinci hash değeri hesaplanır. İkinci hash değeri ne kadar ileri öteleneceğini gösterir.

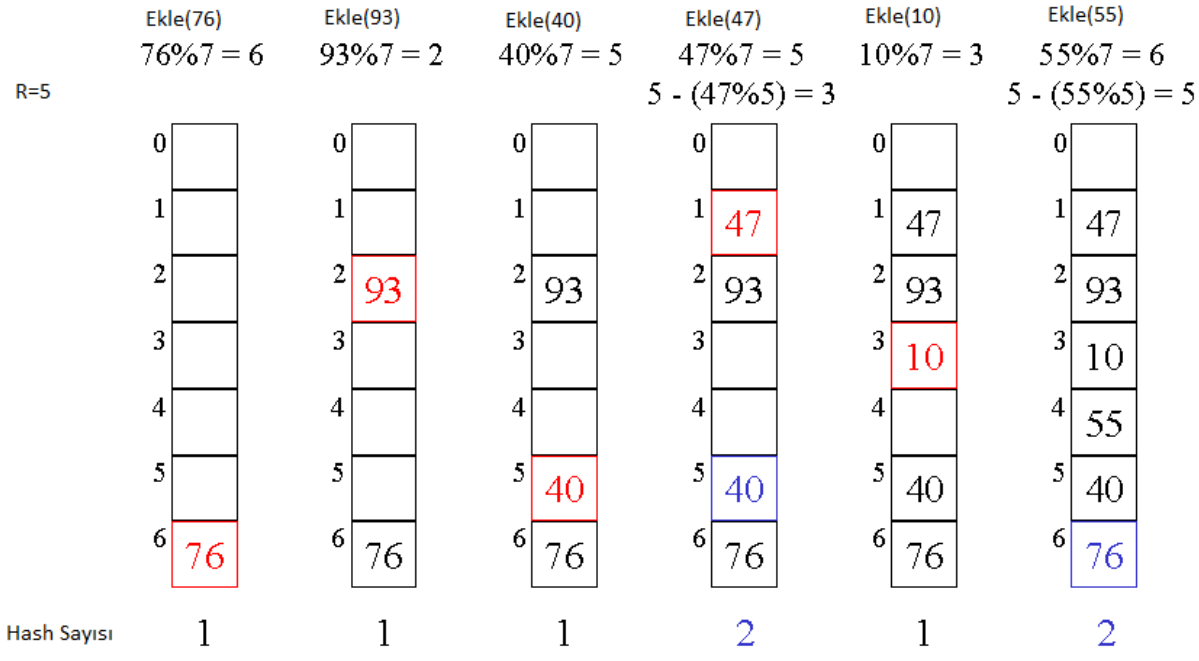
Örnek:

Hash1(değer) = değer % Dizi Boyutu

Hash2(değer) = R - (değer % R)

R: Dizi boyutuna en yakın asal sayı seçilir. (Örnek 10 uzunluğundaki bir dizide R=7 olur çünkü 7, 10'a en yakın asal sayıdır.)

Aşağıdaki örnekte 7 elemanlı bir dizi hash tablosu olarak kullanılacak, 76 sayısı eklendiğinde 7 ile bölümünden kalan 6 'dır. Dolayısıyla 6. indekse yerleşecektir. 6. indeks boş olduğu için 2. hash fonksiyonunu hesaplamaya gerek kalmaz. 93 sayısı da aynı şekilde bir hash fonksiyonu ile eklenir. Fakat 47 sayısı 40 ile aynı indekse sahip olduğu için çarpışma gerçekleşecektir. İkinci hash fonksiyonu hesaplanır. 7 sayısına en yakın asal sayı 5 olduğu için R=5 seçilir. 47 sayısı için ikinci hash fonksiyonu 3 değerini döndürecektir. Bu değer normal olması gereken indeksten kaç adım öteleneceğini gösterir. Dolayısıyla 1 numaralı indekse yerleşir.



İkili hashing yukarıda bahsedilen kurallar tek yol olmayıp farklı ikili hashing uygulamaları ile karşılaşılabilir. Fakat bu ders kapsamında sorumlu tutulacağınız hashing kuralı bu dosyada anlatıldığı şekliyledir. Eğer ikili hashing yaptıktan sonra yine çarpışma olursa ikinci hash fonksiyonunun hesaplamış olduğu öteleme değeri kadar tekrar öteleme yapılır.

Hazırlayan
Arş. Gör. M. Fatih ADAK