Veri Yapıları

Queue

Liste ve Dizide Avantajları;

Ekleme yani enqueue listede avantajlidir çünkü dizide boyut sabit olduğundan aşma durumunda yeni olusturulan diziye kopyalanması yük olacaktır.

Clear için ise dizi avantajlidir çünkü listedeki gibi tüm düğümleri dönerek silmek gerekmez.

Ayrica dizi yapisinda baştan eleman silindiğinden dolayı yeni bir dizi oluşturulana kadar ölü alanlar oluşur.

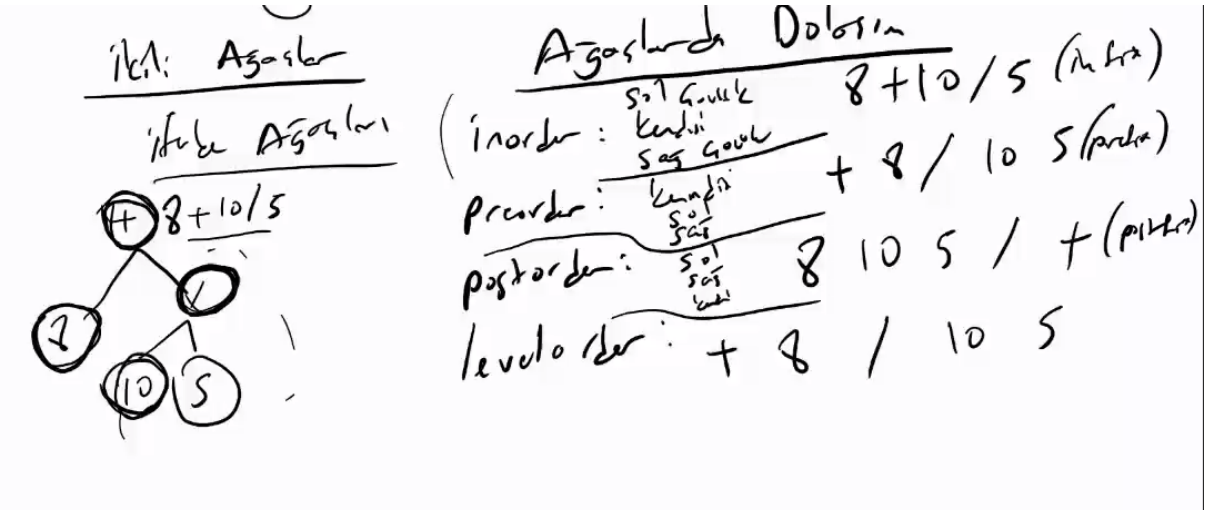
Binary Search Tree (BST) ve AVL Ağacı

BST için;

Dizi üzerinde tasarlandığında dezavantaji her halukarda dizide çocuk icin yer ayırıyor olmasıdır. Diğer dezavantaji ise silme işlemi varsa kesinlikle kullanılmamalıdır.

Liste üzerinde tasarlandığında avantajı silme işlemi dizideki gibi karmaşık değildir, boyut dinamiktir.
Dezavantaji ise düğümlere sıçramak yüktür.

Ağaçlarda dolaşım şekilleri;



Heap

Sıralı olarak ilerlendiğinden dolayı BST gibi arada bir boşluk oluşmayacağından ve indekslerde hızlı işlemler gerçekleştirebildiği için heap, dizide gerçekleştirilir.

Minimum ve maksimum heap türü vardır.

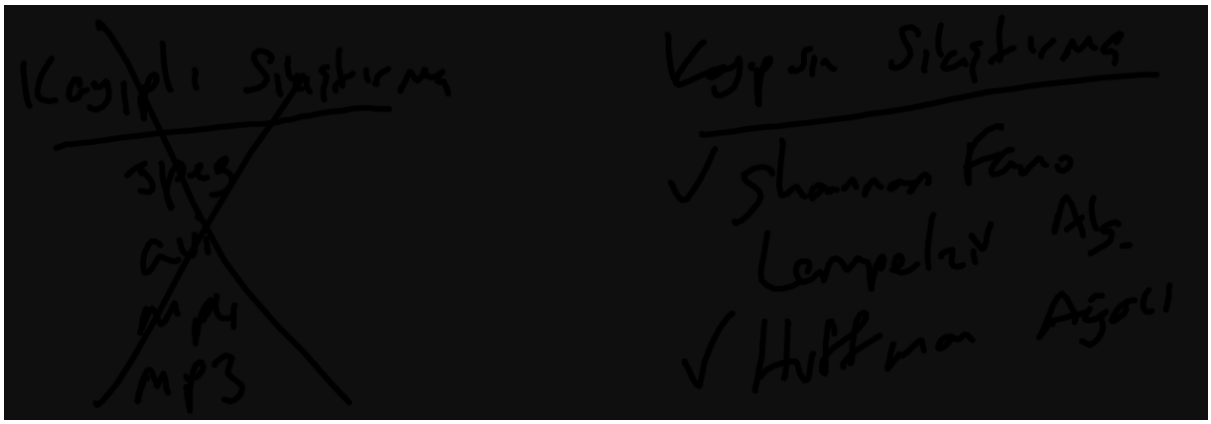
Minimum heap diziye küçükten büyüğe, maksimum heap ise büyükten küçüğe sıralanır.
Eklemeler ise her seviyede soldan sağa yapılır.
Ebeveyn her zaman çocuktan küçük olmak zorundadır, dolayısıyla en küçük değer köktür.
Sıralarken yani heapsort esnasında sıralanan düğüm alındığı için o ağaçtan sökülür ve yeniden ağaç düzenlemesi yapılır, olayı budur.

Dosya Sıkıştırma Yapıları

Küçük değer mutlaka sola gelmelidir. (frekans)
Sınavda düğümler verilip genel ağacı oluştur denebilir, önemli olan küçüğün solda olması!!
Sürekli frekansı en küçük 2 taneyi kombine ediyoruz t1,t2.. olarak.
Oluşan t'ler de kombinlenecek değerler içerisinde sayılır, yani 15 frekanslı x ile t2 kombinlenebilir
keza t'ler de kombinlenebilir.

En son ağaç oluşturulduktan sonra kodlamayı yapmak için sol cocuk 0, sag cocuk ise 1'dir.

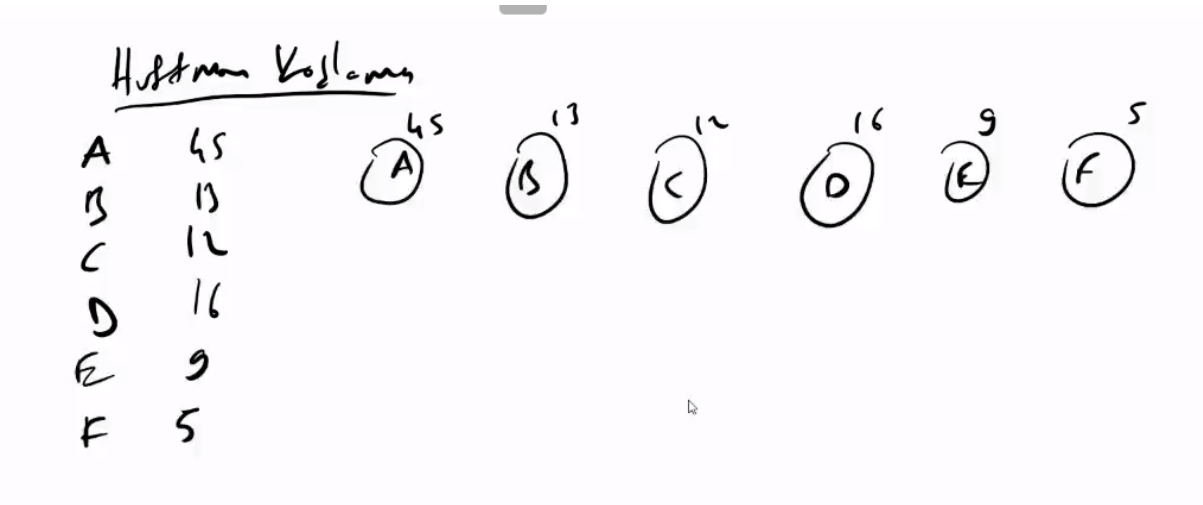
Kayıplı ve kayıpsız sıkıştırma türleri;



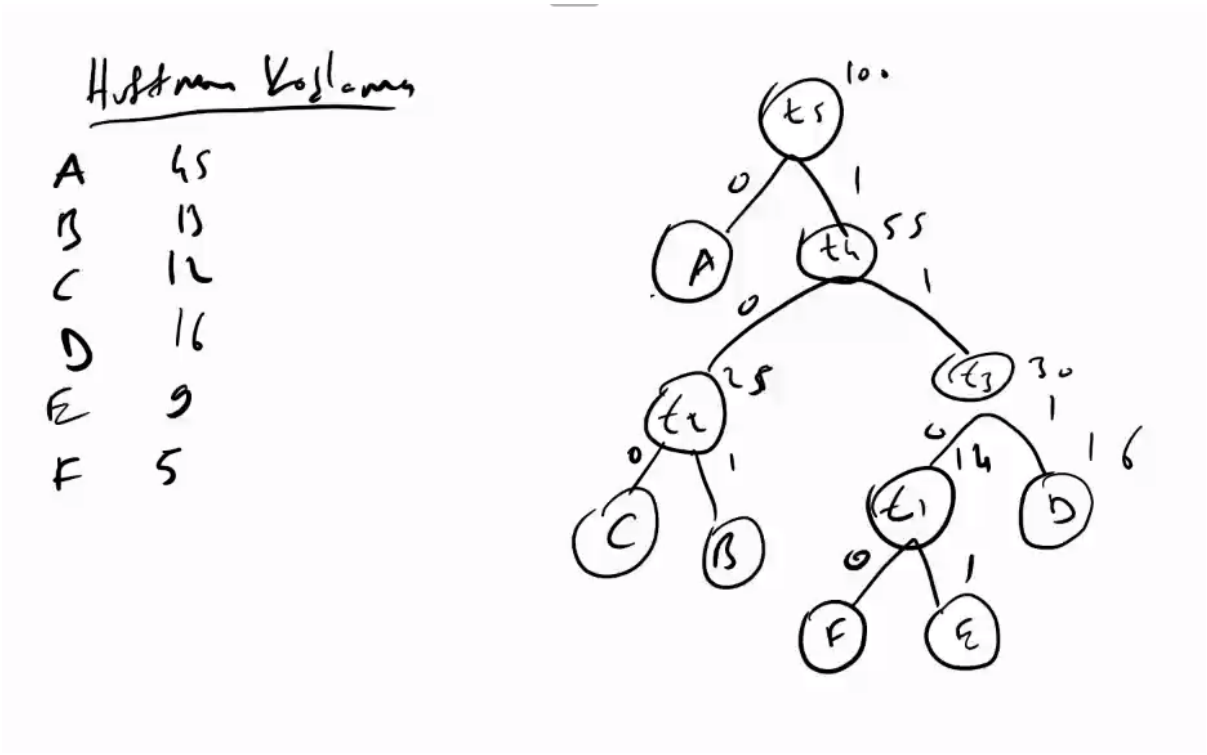
Huffman

Örnek için huffman öncesi bitini hesaplamada frekanslar toplanır 45+13... ve 8 ile çarpılır =800bit.
Sıkıştırıldıktan sonraki bitini hesaplamada örneğin a 3. aşamada 0 çıkmış kodlaması yani 1 bit 45*1(frekansbitsayisi)+ f 1101 4 bit 5*4..... toplanır =224bit olur.

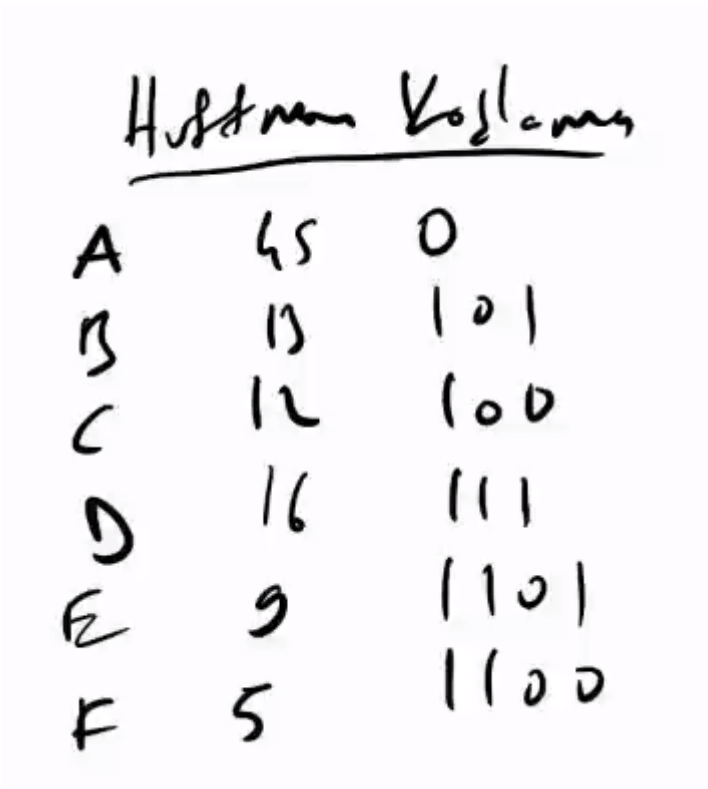
1) Soru böyle verilir;



2) Sorudan yola çıkarak ağaç yapısı oluşturulur;



3) Son olarak ise oluşturduğumuz ağaçtan kodlamaları çıkartılır.

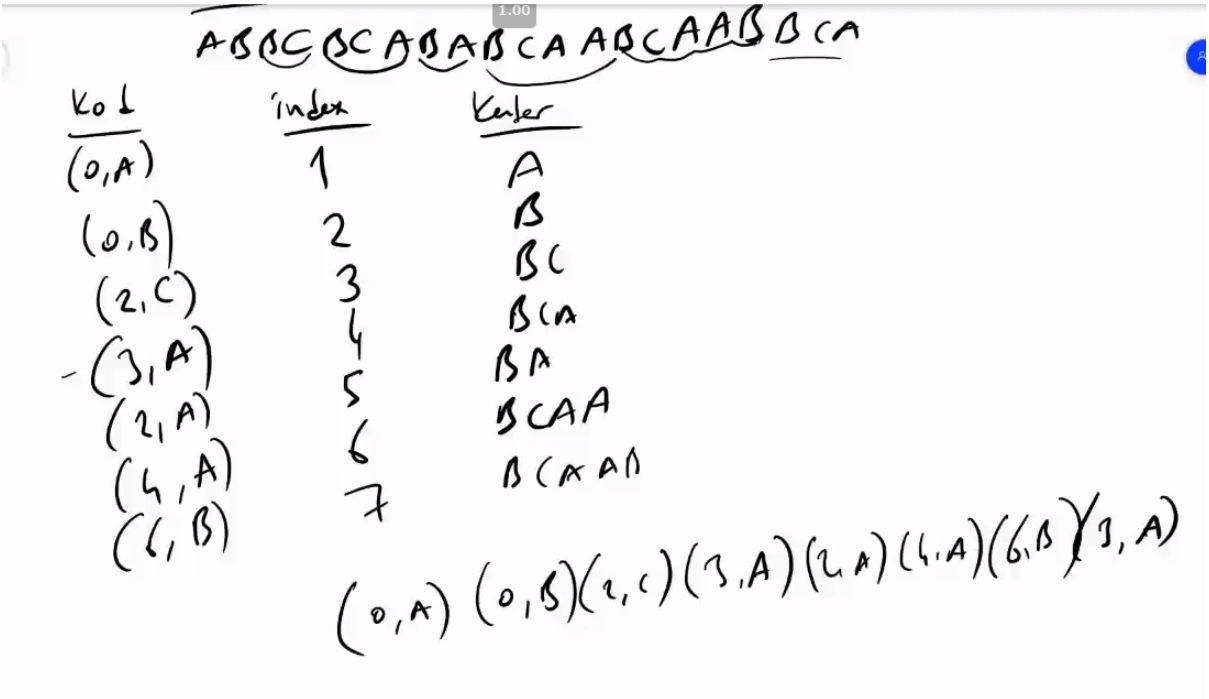


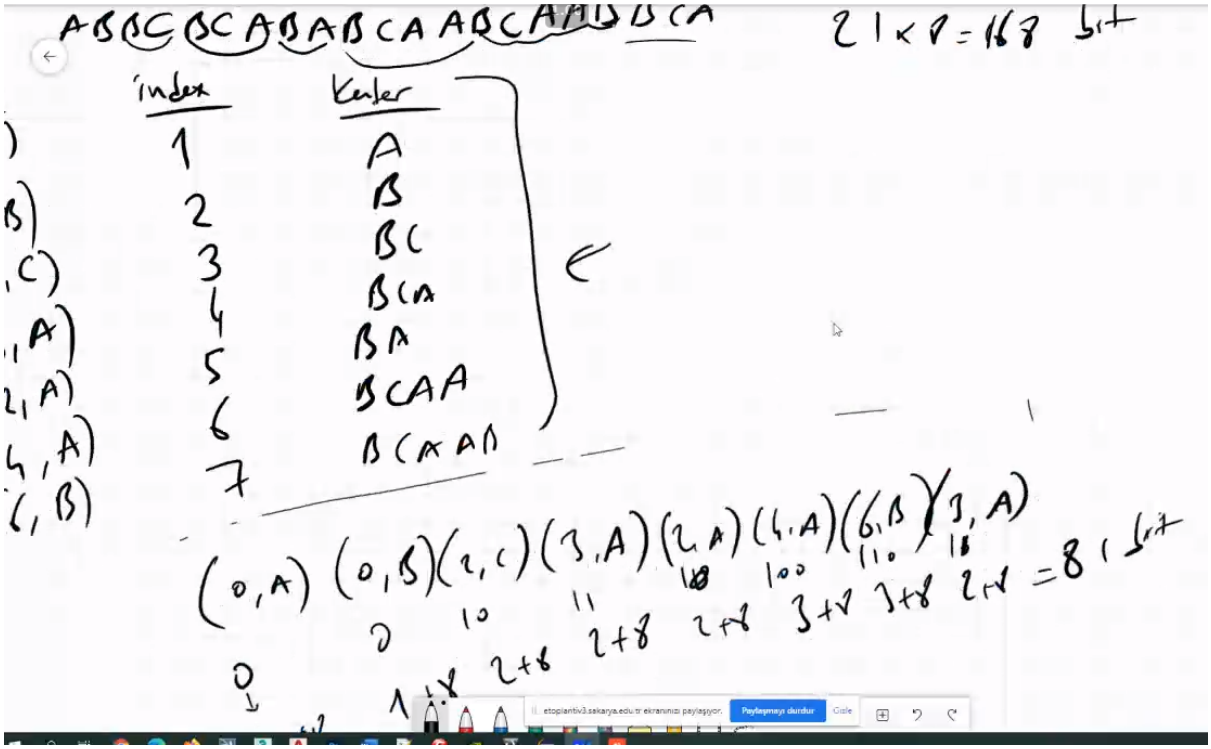
Lempelziv

Soruda yalnızca en üstteki harf grubu verilir. Eğer örneğin b'yi ilk defa görüp b'yi aldıktan sonra tekrar b'yi görmüşüz ve b'yi index 2'de gördüğümüz için bundan sonrakinin 2'den başladığını ve sonrasında C geldiğini kodda belirtmişiz ve yeni katarımız BC olmuş. Aynı mantıkla verilenin tamamı bitiriliyor ve kodlar, indexler, katarlar oluşturuluyor.

En son ise tüm kodlar soldan sağa yazılır (sonda tekrar BCA gelmiş ve bunu daha önceden gördüğümüz için yeni katar oluşturmuyoruz, sadece bu aşamada yerine yazıyoruz) ve indeksleri 2 tabanında yazarak bit sayısı ile 8'i topladığımızda tümünün toplamı sıkıştırma sonrası (Lempelziv) sonrası bit sayısını yani 81'i bize veriyor.

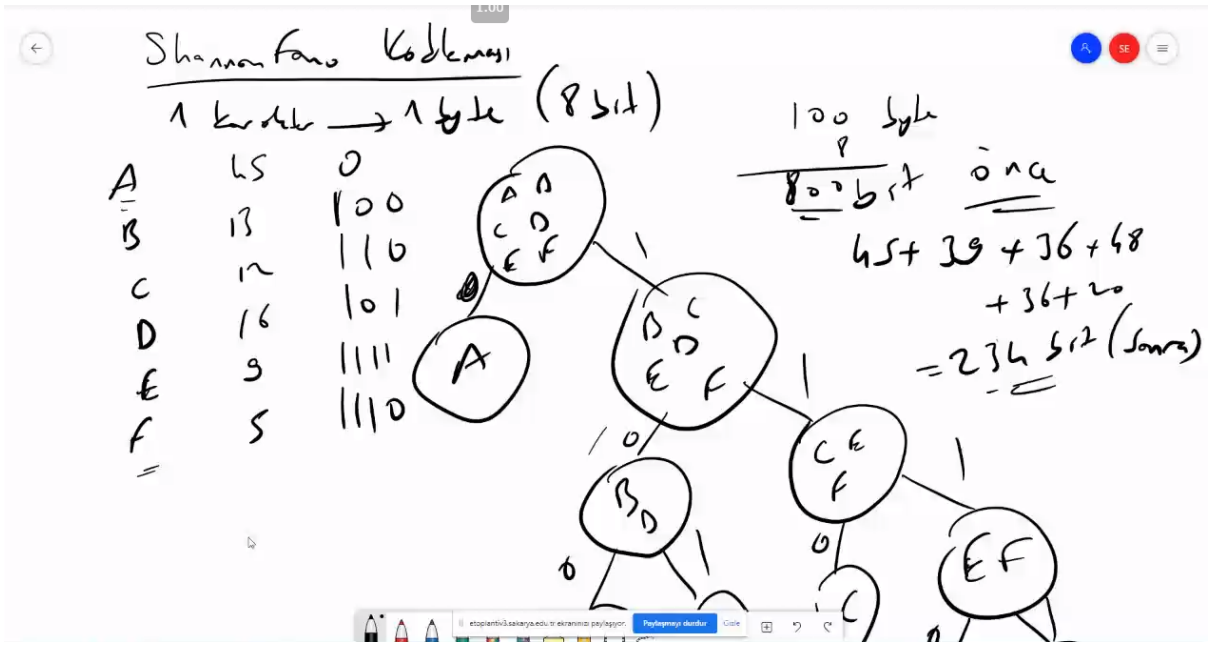
Sıkıştırma öncesi bit sayısı ise verilen harf grubundaki harf sayısı *8 yani 21*8=168 bittir.





Shannon Fano

Bunun mekaniğini hatırlayamadım. Büyük ihtimalle verilen karakterler ağaçta tek kalana kadar dallandırıyoruz ve en son kodları çıkıyor, onlardan da yeni bit frekans*kodunbitsayısı'ndan hesaplanıyor. Örneğin B'nin frekansı 13 ve kodu 3 bit çıkmış, 13*3=39, böyle böyle hepsini toplayınca 234 bit etmiş. Sıkıştırma öncesi bit ise huffman ile aynı zaten.



Hashing Türleri

3 türü vardır;

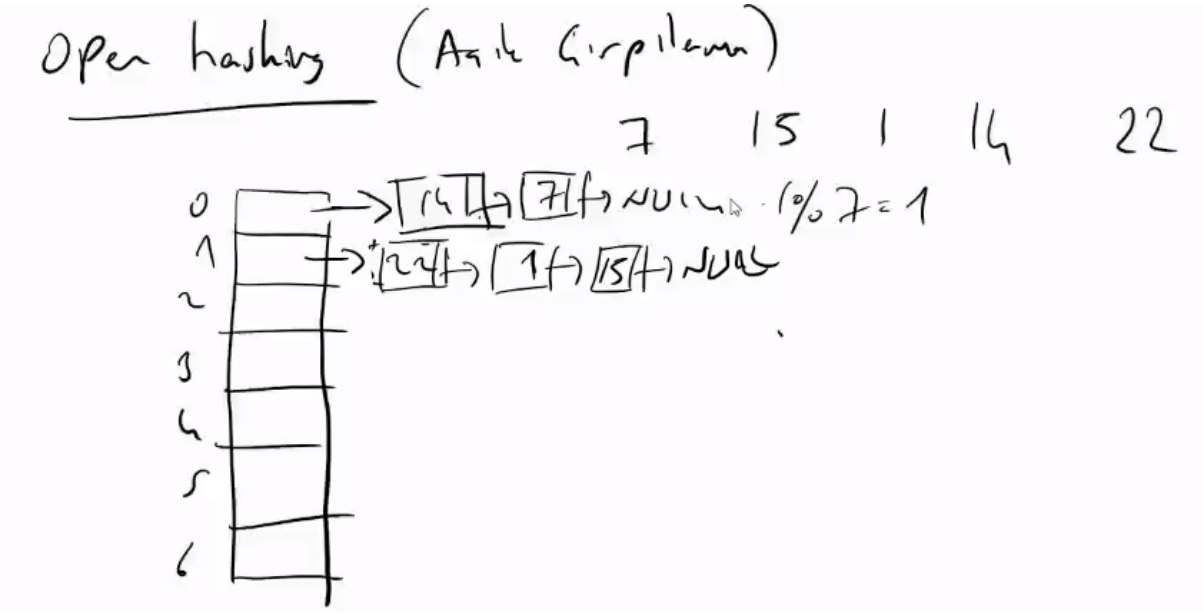
Open Hashing → yer sıkıntısı bulunma durumunda
Closed Hashing → yerim bol varsa ve çarpışma olasılığı düşükse
Double Hashing → belli bölgelerdeki kümelenmelerin önüne geçmek için

Bu yöntemler için eğer veri sayı değil de string ise stringdeki boşluk sayısına veya sesli harf sayısına ve her karakterin ascisine göre çözümler üretilebilir.

Open Hashing

Temel olayı dizi boyutuyla modunu aldığında eğer eklenecek sayılarda modu alındıgındaki sonuçlar eşit çıkarsa aynı listeye eklemektir.

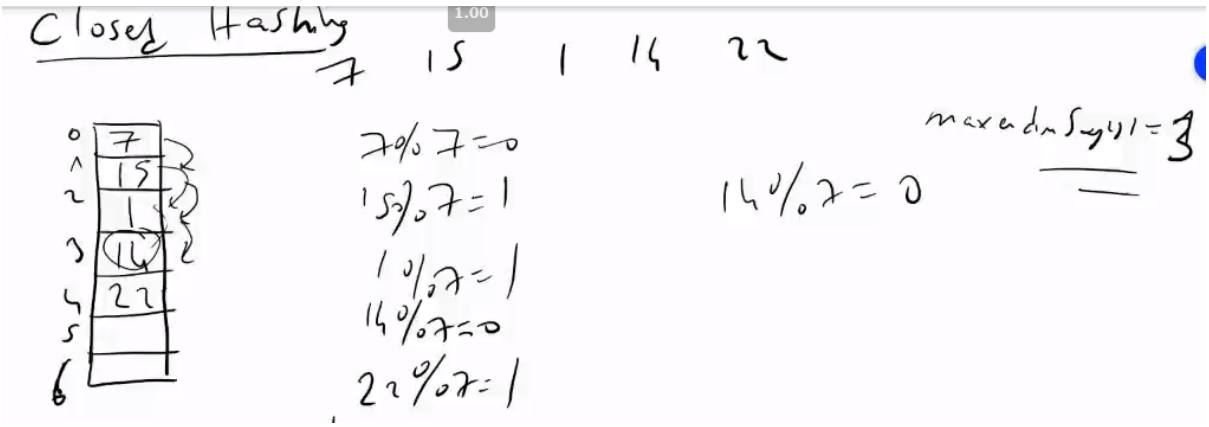
İlk elemanı elemanın sayısının indeksindeki liste dizisine ekliyoruz ve sonrakileri o ilk elemanla modunu alarak sonraki liste dizilerine aynı şekilde yolluyoruz. Eğer ilk elemanla modu 0'a o elemanın oldugu listeye ekliyoruz.



Closed Hashing

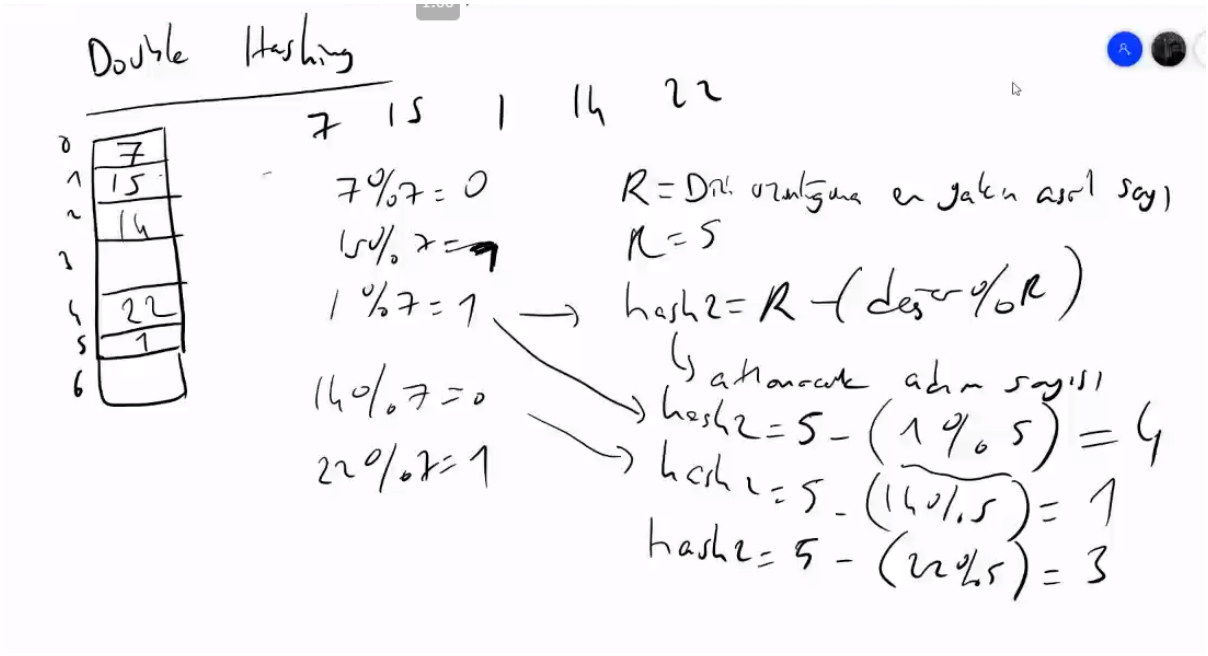
Open hashing aksine dizi boyutuyla modunu alıp 0 çıkmayanları diğer dizi indeksindeki listeye eklemek yerine diğer bir liste dizisine atar her birini her aynı

gelen modlama sonucunda maxadımsayısı tutulur.



Double Hashing

R → Dizi uzunluguna en yakın asal sayı
Örneğin dizi uzunluğu 7 ise r 5 olur.
Örneğin 7 15 1 eklenirse 15%7=1 ve 1%7=1 olduğunda çarpışma olacağından dolayı hash2 çağırılır
`hash2=r-(deger%R)` → atlanacak adım sayısını döndürür hash2, örneğin 3 döndürürse 3'er 3'er adım atlanır.
Örnekteki durum için `hash2=5-(1%5)=4` yani 4 adım atlayacaksın demek oldu. Dizinin 0. indeksinde 7 vardı, 1.de 15, 1+4=5. indekse 1 konuldu. (burada tekrar eden mod sonucunun oldugu indeks baz alınarak atlama yapılır yani 15 1. indekste oldugu icin 1+4)


Double Hashing
7 15 1 14 22
0
1
2
3
4
5
6
7
15
14
22
1
7%7 = 0
15%7 = 1
1%7 = 1
14%7 = 0
22%7 = 1
R = Dizi uzunluğuna en yakın asal sayı
R = 5
hash2 = R - (değer%R)
atlanacak adım sayısı
hash2 = 5 - (1%5) = 4
hash2 = 5 - (14%5) = 1
hash2 = 5 - (22%5) = 3