



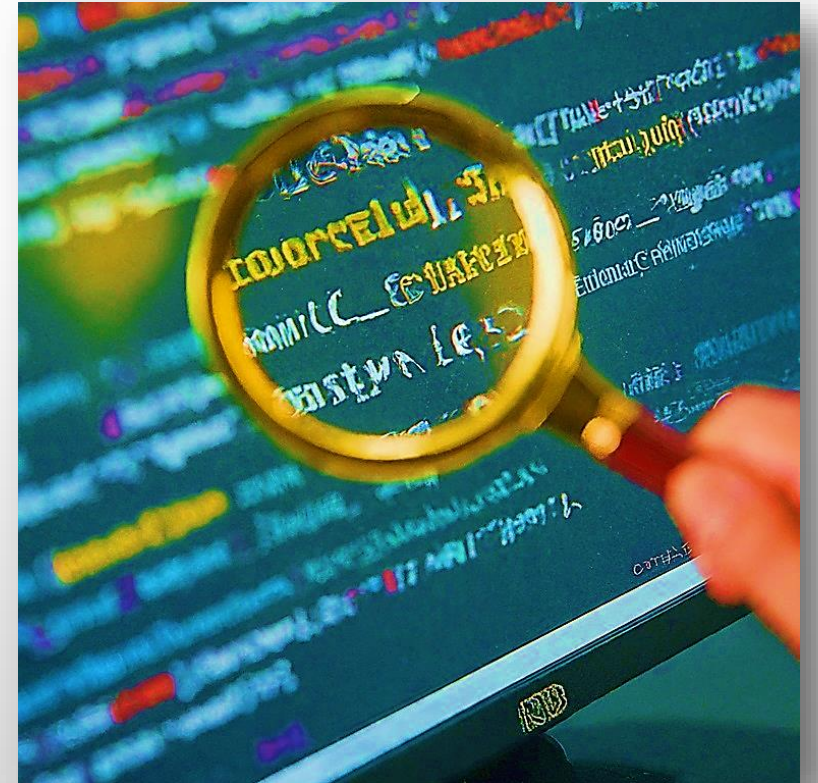
Bölüm 3: Arama Algoritmaları

Algoritmalar



Arama Algoritmaları

- Bir veri kümesinde istenilen değeri bulmak için kullanılır.
- Arama kriterine göre veriyi tarar ve
 - eşleşen öğeyi bulmaya çalışır.
- Web sitelerinde, müzik çalarlarda, akıllı telefonlarda yaygın kullanılır.





Arama Algoritmalarının Çeşitleri

- Farklı arama algoritmaları, farklı çalışma prensiplerine sahiptir.
- Doğrusal Arama (Linear Search):
 - Verileri tek tek karşılaştırarak arar.
- İkili Arama (Binary Search):
 - Veri kümesini ikiye bölerek ve arama alanını daraltarak arar.
- Hash Arama (Hash Search):
 - Veriyi hızlı erişim için bir hash tablosuna yerleştirir ve
 - Aramayı tablo üzerinden yapar.



- [illegible]



Doğrusal Arama

- Listedeki her öğeyi tek tek kontrol ederek arama yapar.
- Kayıp bir eşyayı bulmak için odayı sistematik olarak tarama gibi.
- Aranan değer, listenin başından başlanarak her öğe ile karşılaştırılır.
- Eğer aranan değer bulunursa, konumu döndürülür.
- Eğer aranan değer listede yoksa, başarısız sonuç döndürülür.



Avantajları

- Kodlaması ve anlaşılması kolay.
- Karmaşık veri yapıları gerektirmez.
- Ön hazırlık süreci yoktur.
- Küçük veri kümelerinde hızlı bir şekilde arama yapmak için idealdir.
- Örneğin, bir telefon numarasını küçük bir rehber listesinde aramak.



Doğrusal Arama Algoritması

- Parametreler:
 - dizi: Aranacak öğelerin bulunduğu dizi.
 - n: Dizinin boyutunu temsil eden tamsayı.
 - aranan: Aranacak öğe.
- Dönüş Değeri:
 - Aranan öğenin dizideki indisi, eğer öğe bulunmazsa -1.



Doğrusal Arama Algoritması

```
def dogrusal_arama(dizi, n, aranan):  
    i = 0  
    while i < n and dizi[i] != aranan:  
        i += 1  
  
    if i < n:  
        return i  
    else:  
        return -1
```




Doğrusal Arama Algoritması

- `dogrusal_arama()` fonksiyonu, *dizi*, *n* ve *aranan* parametrelerini alır.
- *i* değişkeni, dizideki öğeleri dolaşmak için kullanılır.
- *while* döngüsü,
 - *i* değeri *n*'den küçük ve
 - `dizi[i]` değeri aranan değerine eşit olmadığı sürece devam eder.
- *i* değeri her döngüde 1 artırılır.
- *if* ifadesi, aranan değerinin dizide bulunup bulunmadığını kontrol eder.
- Eğer aranan dizide bulunursa, *i* değeri (öğenin indisi) döndürülür.
- Aksi takdirde, -1 döndürülür.



İkili Arama

- Kütüphanede kitap aramaya benzer.
- Kitaplar, yazarın soyadına göre alfabetik olarak sıralanmıştır.
- Aranan kitabı bulmak için tüm rafları tek tek aramak yerine,
 - önce orta sıradaki bölüme gidilir.
- Eğer kitap alfabetik olarak orta sıranın solundaysa,
 - sol taraftaki raflar aranmaya devam edilir.
- Sağdaysa, sağ taraftaki raflar kontrol edilir.





Avantajlar

- Önceden sıralı listelerde arama yaparken hızlıdır.
- Her adımda, listenin kontrol edilmesi gereken kısmını yarıya indirir.
- Örneğin, 1000 öğelik bir listede
 - doğrusal arama ortalama 500 kontrol yaparken,
 - ikili arama sadece 10 adımda aramayı tamamlar.
- Büyük ve sıralı veri kümelerinde arama için ideal.





İkili Arama Algoritması

```
def ikili_arama(dizi, n, aranan):  
    bas = 0  
    son = n - 1  
    while bas <= son:  
        orta = (bas + son) // 2  
        if aranan < dizi[orta]:  
            son = orta - 1  
        elif aranan > dizi[orta]:  
            bas = orta + 1  
        else:  
            return orta  
    return -1
```



İkili Arama Algoritması

- `ikili_arama()` fonksiyonu, `dizi`, `n` ve aranan parametrelerini alır.
- `bas` ve `son`, arama yapılacak dizinin alt ve üst sınırlarını temsil eder.
- `while`, `bas` değeri `son` değerinden küçük eşit olduğu sürece devam eder.
- `orta` değişkeni, her seferinde dizinin ortasındaki öğenin indisini tutar.
- `if` ifadesi, aranan değerini `dizi[orta]` değeri ile karşılaştırır.
- Aranan değer ortadan küçükse, arama dizinin alt yarısında devam eder.
- Aranan değer ortadan büyükse, arama dizinin üst yarısında devam eder.
- Aranan değer ortaya eşitse, fonksiyon `orta` değerini döndürür.
- Aranan değer bulunamamışsa fonksiyon `-1` döndürür.



Hash Tablo Arama Algoritması

- Bir anahtarın kilide tam olarak oturması gibi çalışır.
- Veriler, anahtar kelimeler ve bunlara karşılık gelen değerlerden oluşur.
- Anahtar kelimeler, hash fonksiyonu ile benzersiz değerlere dönüştürülür.
- Hash değeri, hash tablosundaki verilerin yerini işaret eder.
- Aranan anahtar kelimenin hash değeri ile konumda (kova) arama yapılır.
- Eğer kova boş değilse, anahtar kelime kovadaki değerlerle karşılaştırılır,
 - aranan değer bulunursa işlem tamamlanır.





Avantajları

- Ortalama durumda çok hızlı arama yapar.
- Verilerin önceden sıralanmasına gerek yoktur.
- Büyük veri kümeleri aramalarında idealdir.
- Hash fonksiyonu iyi seçilmişse, aranılan bilgiye doğrudan erişilebilir.
- Hash fonksiyonu çakışmalara yol açabilir.
 - iki farklı anahtar kelimenin aynı hash değerine sahip olması.
- Çakışmalar olduğunda, ek işlem adımları gerekebilir. 😞



Hash Tablo Arama Algoritması

- Veri Yapıları:
 - hash_table: Anahtar-değer çiftlerini saklayan hash tablosu.
 - hash_fonksiyonu: Bir anahtarı hash tablosunda bir indekse dönüştüren fonksiyon.
- Parametreler:
 - hash_table: Aranacak öğelerin bulunduğu hash tablosu.
 - anahtar: Aranılan öğenin anahtarı.



Hash Tablo Arama Algoritması

```
def hash_arama(hash_table, anahtar):  
    index = hash_fonksiyonu(anahtar)  
    while hash_table[index] is not None:  
        if hash_table[index][0] == anahtar:  
            return hash_table[index][1]  
        index = (index + 1) % len(hash_table)  
    return None
```



Hash Fonksiyonu

```
def hash_fonksiyonu(anahtar):  
    toplam = 0  
    for karakter in anahtar:  
        toplam += ord(karakter)  
    return toplam % len(hash_table)
```



Hash Tablo Arama Algoritması

- hash_arama() fonksiyonu, hash_table ve anahtar parametrelerini alır.
- hash_fonksiyonu kullanılarak anahtar bir indekse dönüştürülür.
- while döngüsü,
 - hash_table[index] değeri None olana kadar veya
 - tablo sonuna ulaşana kadar devam eder.
- Her döngüde, hash_table[index] konumundaki anahtar ile karşılaştırılır.
- Eğer anahtarlar eşleşirse, bu konumdaki değer döndürülür.
- Eşleşme yoksa bir sonraki indekse bakılır (döngüsel arama).
- Döngü sonunda None döndürülür (anahtar bulunamadı).



Aradeğer (Interpolation) Arama

- Sadece önceden sıralanmış veri setlerinde kullanılabilir.
- Verilerin listenin içinde eşit aralıklarla dağıldığını varsayar.
- Listenin başlangıç değerinden sonra her eleman arasındaki fark aynıdır.
- Bu varsayıma dayanarak, aranan değerın konumu tahmin edilir.
- Tahmin edilen konum kontrol edilir ve eğer yanlışsa,
 - arama daha dar bir aralıkta (sağda veya solda) devam ettirilir.



Aradeğer (Interpolation) Arama

- En küçük ve en büyük eleman kullanılarak tahmini konum hesaplanır.
- Tahmini konumdaki değer kontrol edilir.
- Eğer tahmini konumdaki değer
 - aranan değer ise, arama başarıyla tamamlanır ve konum döndürülür.
 - aranan değerden küçükse, arama listenin sağ yarısında devam ettirilir.
 - aranan değerden büyükse, arama listenin sol yarısında devam ettirilir.
- Bu adımlar, aranan değer bulunana kadar veya listenin tüm elemanları kontrol edilene kadar tekrarlanır.



Aradeğer (Interpolation) Arama

```
def interpolation_arama(dizi, n, aranan):  
    bas = 0  
    son = n - 1  
    while bas <= son:  
        pozisyon = bas + (((aranan - dizi[bas]) * (son - bas)) // (dizi[son] - dizi[bas]))  
        if dizi[pozisyon] == aranan:  
            return pozisyon  
        elif dizi[pozisyon] < aranan:  
            bas = pozisyon + 1  
        else:  
            son = pozisyon - 1  
    return -1
```




Aradeğer (Interpolation) Arama

- interpolation_arama() fonksiyonu, dizi, n ve aranan parametrelerini alır.
- bas ve son değişkenleri, arama yapılacak dizinin alt ve üst sınırlarını temsil eder.
- Eğer aranan değer uç değerlerden küçük veya büyükse -1 döndürülür.
- pozisyon değişkeni, interpolation hesaplaması ile tahmini bir indis hesaplar.
- while döngüsü bas ve son değerleri kesişene kadar devam eder.
- Her döngüde, hesaplanan pozisyon kontrol edilir.
- Aranan değer dizi[pozisyon] ile eşleşirse pozisyon döndürülür.
- dizi[pozisyon] aranandan küçükse, arama pozisyon+1 indisinden devam eder.
- dizi[pozisyon] aranandan büyükse, arama pozisyon-1 indisine kadar devam eder.
- Döngü sonunda aranan değer bulunamamışsa -1 döndürülür.



Atla Ara Bul (Jump) Algoritması

- Engelli koşuda engelleri aşarak ilerleyen bir atlet gibi çalışır.
- Listeyi önceden belirlenen büyüklükte parçalara ayırır.
- İlk olarak listenin başından büyük bir adım atlar ve o konumdaki öğeyi kontrol eder.
- Eğer aranan değer bu konumdaysa, arama başarıyla sona erer.
- Eğer atlanan konumdaysa, atlanan aralığa geri dönülerek o aralıkta doğrusal arama yapılır.
- Bu sayede, tek tek tüm öğeleri kontrol etmek yerine daha hızlı bir şekilde arama gerçekleştirilebilir.



Atla Ara Bul (Jump) Algoritması

```
def jump_search(dizi, eleman, n):  
    atlama = int(math.sqrt(n))  
    konum = 0  
    while dizi[min(atlama, n)-1] < eleman:  
        konum = atlama  
        atlama += int(math.sqrt(n))  
        if konum >= n:            return -1  
    while dizi[konum] < eleman:  
        konum += 1  
        if konum == min(atlama, n):    return -1  
    if dizi[konum] == eleman:        return konum  
    return -1
```



Atla Ara Bul (Jump) Algoritması

- `atlama_arama` fonksiyonu, `dizi`, `eleman` ve `n` değerlerini parametre alır.
- Atlama mesafesi `math.sqrt(n)` ile hesaplanır.
- `konum` değişkenine başlangıçta sıfır atanır.
- Döngü içerisinde `konum` değeri, atlama mesafesi kadar artırılır.
- Eğer bulunan değer aranan değerden (x) büyükse, döngüden çıkılır.
- Döngü sonunda, aranan değer olabileceği aralığın sonu işaret edilir.
- İkinci bir döngü ile `konum`dan itibaren doğrusal arama ile taranır.
- Eğer x değeri bulunursa, bulunduğu indeks değeri döndürülür.
- Dizi içerisinde x değeri yoksa `-1` döndürülür.



Üstel (Exponential) Arama

- Define avcısının harita işaretlerini takip ederek defineye yaklaşması gibi.
- Listeyi, üstel olarak büyüyen parçalara ayırır.
- Örneğin, listenin uzunluğu 1024 ise,
 - ilk arama 1024. konuma, yani listenin sonuna yakın bir konuma yapılır.
 - aranan değer burada değilse, atlama mesafesi geriye çekilir ve
 - arama 512 ile 1023 arasındaki kısımda devam eder.
 - her adımda atlama mesafesi küçülerek arama alanı daraltılır.



Üstel (Exponential) Arama

```
def usluArama(dizi, x, n):  
    konum = 1  
    while konum < n and dizi[konum - 1] < x:  
        konum *= 2  
    for i in range(konum // 2, n):  
        if dizi[i] == x:  
            return i  
    return -1
```



Üstel (Exponential) Arama

- usluArama fonksiyonu, dizi, x ve n değerlerini parametre alır.
- konum değişkeni başlangıçta 1 olarak atanır.
- Döngü içerisinde konum değeri,
 - dizi[konum - 1] değeri x değerinden küçük olduğu sürece 2 ile çarpılır.
- Döngü sonunda, aranılan değer olabileceği aralığın sonu işaret edilir.
- konum değeri 2 ile bölünerek arama aralığı daraltılır.
- Doğrusal arama (linear search) ile kalan kısım taranır.
- Eğer x değeri bulunursa, bulunduğu indeks değeri döndürülür.
- Dizi içerisinde x değeri yoksa -1 döndürülerek bulunamadığı belirtilir.



SON