SAKARYA MESLEK YÜKSEKOKULU

BİLGİSAYAR PROGRAMCILIĞI PROGRAMI

Algoritma ve Programlamaya Giriş Ders Notları

Ders 13: Sıralama ve Arama Algoritmaları

Merge Sort, Quick Sort, Bubble Sort, Selection Sort, Insertion Sort, Shell Sort
Binary Search, Linear Search

Hazırlayan
Öğr. Gör. İsmail ÖYLEK

Sakarya
2020

Sıralama Algoritmaları

Sıralama algoritmaları, bilgisayar bilimlerinde ya da matematikte kullanılan, verilen bir listenin elemanlarını belirli bir sıraya sokan algoritmalardır. En çok kullanılan sıralama türleri, sayı büyüklüğüne göre sıralama ve alfabetik sıralamadır. Sıralama işleminin verimli yapılması, arama ve birleştirme algoritmaları gibi çalışması için sıralanmış dizilere gereksinim duyan algoritmaların başarımının yüksek olması için önemlidir. Sıralama algoritmaları, bilgisayarlarda tutulan verilerin düzenlenmesini ve kullanıcı tarafından daha rahat algılanmasını da sağlamaktadır.

Sıralama algoritmaları, tanımı çok yalın olmasına karşın çözümü çok karmaşık olan bir işi gerçekleştirdikleri için, üzerinde en fazla araştırma yapılan bilgisayar bilimi konularındandır. Sıralama algoritmalarında performansı öne çıkarak 2 önemli durum vardır. Bunlardan birincisi hafıza, ikincisi zamandır [Kaynak: 4]. Bu dokümanda 6 farklı sıralama algoritmasına yer verilmiştir.

1. Birleştirmeli Sıralama (Merge Sort): Böl ve kazan (*divide and conquer*) stratejisi kullanılır. Bu sıralama algoritmasında, fazladan hafıza kullanılır. Diğer bahsedilen algoritmalara göre çok daha hızlıdır. Yapılan işlem kısaca şöyledir:
Setimiz, iki veri kalana kadar parçalara ayrılır (sürekli yarıya bölünerek) ve her ikili parça kendi içinde sıralanır. Her iki parça sıralandıkça, diğer ikili parçayla kıyaslanıp sıralanır ve böylece elimizde sıralı veri dizisi oluşur [Kaynak: 2].

2. Hızlı Sıralama (Quick Sort): Birleştirmeli sıralama algoritmasında olduğu gibi, çabuk sıralama algoritması da böl ve kazan (*divide and conquer*) stratejisini kullanır. Çabuk sıralama performans olarak diğer sıralamalara göre çok iyidir. Özellikle büyük boyutlu setlerde bu algoritmayı kullanmak çok önemlidir. Fakat çabuk sıralama algoritmasını koda dökebilmek zordur. Bu yüzden, sıralamaya ihtiyaç duyan kişi, bu algoritmayı yazmak için harcayacağı zamanın, sıralama yaparken kazanacağı zamana oranını iyi hesaplaması gerekir. Çabuk sıralamada yapılan işlem kısaca şöyledir: Veri seti iki parçaya ayrılır, bu işlem için bir veri pivot olarak seçilir, kalan iki parça ise, pivottan küçükler ve pivottan büyüklerin oluşturduğu parçalar olur. Bu iki parçaya da yine çabuk sıralama algoritması uygulanır. Yani çabuk sıralama algoritması, kendi içerisinde yine kendini çağırır (*özyineleme - recursion*) [Kaynak: 2].
Örnek…

27 63 1 72 64 58 14 9 ➔ Pivot 9 olsun.
1 9 63 72 64 58 14 27 ➔ 9'dan küçükler 9'un soluna, büyükler sağına yerleştirildi. Yeni pivot 27 olsun.
1 9 14 27 64 58 72 63 ➔ 27'den küçükler 27'nin soluna, büyükler sağına yerleştirildi. Yeni pivot 63.
1 9 14 27 58 63 72 64 ➔ 63'ten küçükler 63'ün soluna, büyük sağına yerleştirildi.

1 9 14 27 58 63 64 72 ← Sıralama tamamlandı.

3. Kabarcık Sıralama (Bubble Sort): Sıralama algoritmaları arasında en yavaş çalışan olarak değerlendirilir. Aslında bilgisayar bilimi eğitiminde, sıralama algoritması hakkında bir temel oluşturulması için öğretilmesi dışında pek kullanılmaz. Elemeli sıralamada yapılan işlem kısaca: veri setinin ilk iki elemanı okunup karşılaştırılır. İlk veri ikinciden küçükse, ikinci ve üçüncü veri karşılaştırılır; büyükse ikisi yer değiştirir. Bu işlemler son veriye ulaşana kadar devam eder ve doğru sıralamaya ulaşana kadar işlemler tekrar baştan başlar [Kaynak: 2].
Örnek…

(5 1 4 2 8) ➔ (1 5 4 2 8) ➔ burada, ilk iki veri karşılaştırılıyor ve ikinci küçük olduğu için yer değiştiriyorlar.
(1 5 4 2 8) ➔ (1 4 5 2 8)
(1 4 5 2 8) ➔ (1 4 2 5 8)
(1 4 2 5 8) ➔ (1 4 2 5 8) ➔ son elemana ulaşılıyor ama setimiz hala sıralı değil, o yüzden tekrar baştan başlıyoruz.
(1 4 2 5 8) ➔ (1 4 2 5 8)
(1 4 2 5 8) ➔ (1 2 4 5 8) ➔ Sıralama tamamlandı. Ama döngü tamamlanana dek kontrol yapılır.
(1 2 4 5 8) ➔ (1 2 4 5 8)
(1 2 4 5 8) ➔ (1 2 4 5 8)

Kabarcık Sıralama (Bubble Sort) için Örnek Kodlama ➔ [Kaynak: 1]

```
int[] array = new int[6] { 5, -3, 0, 8, -6, 4 };
bool swapped = true;
int temporary;
int j = 0;

// Sıralanmadan diziyi ekranda gorelim
Console.WriteLine("Dizi sıralanmadan önce:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}
// Diziyi Sıralayalım
while (swapped)
{
    swapped = false;
    j++;

    for (int i = 0; i < array.Length - j; i++)
    {
        if (array[i] > array[i + 1])
        {
            temporary = array[i];
            array[i] = array[i + 1];
            array[i + 1] = temporary;
            swapped = true;
        }
    }
}
// Sıralanmıs diziyi ekranda gorelim
Console.WriteLine("\nDizi sıralandıktan sonra:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}
Console.ReadKey();
```

4. Seçmeli Sıralama (Selection Sort): Basit ve yazılması kolay bir algoritmadır. Ama kabarcık sıralamaya göre daha iyi olsa da diğer algoritmalara göre yavaş çalışır. Seçmeli sıralamada yapılan işlem kısaca şöyledir: Veri seti sırayla okunur ve en küçük olan eleman, setimizin ilk elemanıyla yer değiştirir. İkinci elemandan itibaren set bir daha okunur ve en küçük veriyle ikinci veri yer değiştirir. Bu işlem son veriye gelene kadar devam eder [Kaynak: 2].
Örnek…

31 25 12 22 11 ➔ veri setimiz
11 25 12 22 31 ➔ en küçükleri olan 11, ilk sıradaki veri ile yer değiştirdi.
11 12 25 22 31
11 12 22 25 31 ➔ Sıralama tamamlandı.

Seçmeli Sıralama (Selection Sort) için Örnek Kodlama ➔ [Kaynak: 1]

```
int[] array = new int[10] { -4, 1, 8, 6, -9, 2, 0, -5, -7, 3 };
Console.WriteLine("Sıralamadan önce dizi:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}

int temporary;
int min;
// Seçerek Sıralama / i indeks, min ise minumum elemanın indeksidir.
for (int i = 0; i < array.Length; i++)
{
    min = i;
    for (int j = i + 1; j < array.Length; j++)
    {
        if (array[j] < array[min])
        {
           min = j;
        }
    }
    temporary = array[i];
    array[i] = array[min];
    array[min] = temporary;
}

Console.WriteLine("\nSıralamadan sonra dizi:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}
Console.ReadKey();
```

5. Eklemeli Sıralama (Insertion Sort): Kabarcık sıralamaya göre daha hızlı olan bu sıralama algoritması, küçük verili setler için daha uygundur. Seçmeli sıralama algoritmasına benzer ama ona oranla daha hızlıdır. Bu sıralama iki şekilde yapılabilir. Birincisi; Kabarcık ve Seçmeli sıralama algoritmalarında olduğu gibi aynı set içerisinde sıralama. Diğer yöntem ise fazladan bir boş dizi seti daha kullanılarak yapılan sıralama. İkinci yöntemde fazladan dizi ve dolayısıyla fazladan hafıza kullanıldığı için, ilk yöntem daha çok tercih edilir. Eklemeli sıralamada yapılan işlem kısaca şöyledir: Veriler sırayla okunur ve kendinden önceki verilerle kıyaslanıp en uygun yere alınır. Bu algoritma yerden sırayla çekilen iskambil kartlarının sıralanmasına benzetilebilir [Kaynak: 2].
Örnek…

34 8 64 51 32 21 ➔ ilk listemiz
8 34 64 51 32 21 ➔ 8, 34 ün önüne eklendi.
8 34 64 51 32 21 ➔ 64, kendisinden önceki rakamlardan büyük olduğu için yerinde kaldı.
8 34 51 64 32 21
8 32 34 51 64 21
8 21 32 34 51 64 ➔ Sıralama tamamlandı.

Eklemeli Sıralama (Insertion Sort) için Örnek Kodlama ➔ [Kaynak: 1]

```
int[] array = new int[10] { -4, 1, 8, 6, -9, 2, 0, -5, -7, 3 };

Console.WriteLine("Sıralanmadan önce dizi: ");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}

// Insertion sort ile diziyi sıralayalım
int temporary, j;
for (int i = 1; i < array.Length; i++)
{
    temporary = array[i];
    j = i - 1;
    while (j >= 0 && array[j] > temporary)
    {
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = temporary;
}

Console.WriteLine("\nSıralandıktan sonra dizi: ");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}
Console.ReadKey();
```

6. Kabuk Sıralama (Shell Sort): İsmini algoritmayı bulan Donald Shell'den almaktadır. Aslında Meta Sort Algoritması da denilebilir. Çünkü başka bir sıralama algoritması üzerinde çalışır. Bu başka sıralama algoritması genelde Insertion Sort olmak ile birlikte başka herhangi bir algoritma da olabilir. Bu sebeple bu algoritmaya sıralama algoritması demektense başka sıralama algoritmalarının verimliliğini artıran bir algoritma demek de yanlış olmaz [Kaynak: 3].

Örnek…

5, 7, 9, 3, 4, 6, 8, 10, 11, 1, 0, 2 sıralanacak olan dizi olsun. Shell Sort'ta ilk olarak gap'ler belirlenir. Aslında gap, mevcut diziden iki boyutlu yeni diziyi kaçarlı sütun halinde oluşturacağınıza karar vermek içindir. Burada gap 4 olsun. Gap'lerin belirlenmesiyle aşağıdaki sütün yapısı oluşturulur.

5	7	9	3
4	6	8	10
11	1	0	2

Daha sonra her sütün kendi arasında sıralanır. İşte bu sıralama işleminde herhangi başka bir sıralama algoritması da kullanılabilir.

5	7	9	3	4	1	0	2
4	6	8	10 ➔	5	6	8	3
11	1	0	2	11	7	9	10

Her sütun kendi arasında sıralandıktan sonra tekrar satırlar yan yana getirilerek dizi haline çevrilir.

4, 1, 0, 2, 5, 6, 8, 3, 11, 7, 9, 10

Yukarıdaki aşama tekrar uygulanır. Ancak bu kez gap bir önceki gap'in yarısı olur. Yani 4/2'den 2'şerli sütunlar üzerinde işlem yapılır.

4	1		0	1
0	2		4	2
5	6	➔	5	3
8	3		8	6
11	7	9	7	
9	10	11	10	

Tekrar dizi satırlar yan yana gelecek şekilde birleştirilir ve yeni bir dizi oluşturulur.

0, 1, 4, 2, 5, 3, 8, 6, 9, 7, 11, 10 ← Sıralama tamamlandı.

Bu kez üstteki aşamalar 1 kolondan oluşan son dizi için yapılır ki bu da sıralanmış diziyi oluşturur.

Kabuk Sıralama (Shell Sort) için Örnek Kodlama ← [Kaynak: 1]

```
int[] array = new int[10] { -4, 1, 8, 6, -9, 2, 0, -5, -7, 3 };

Console.WriteLine("Sıralamadan önce dizi:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}

int j;
int temporary;
int gap = array.Length / 2;

// Kabuk sıralama,
while (gap > 0)
{
    for (int i = gap; i < array.Length; i++)
    {
        temporary = array[i];
        j = i;
        while (j >= gap && array[j - gap] > temporary)
        {
array[j] = array[j - gap];
j -= gap;
        }
        array[j] = temporary;
    }
    gap = gap / 2;
}

Console.WriteLine("\nSıralamadan sonra dizi:");
for (int i = 0; i < array.Length; i++)
{
    Console.WriteLine("array[" + i + "] = " + array[i]);
}
Console.ReadKey();
```

Arama Algoritmaları

Bilgisayar bilimlerinde, çeşitli veri yapılarının (*data structures*) üzerinde bir bilginin aranması sırasına kullanılan algoritmaların genel ismidir. Örneğin bir dosyada bir kelimenin aranması, bir ağaç yapısında (*tree*) bir düğümün (*node*) aranması veya bir dizi (*array*) üzerinde bir verinin aranması gibi durumlar bu algoritmaların çalışma alanlarına girer.

Yapısal olarak arama algoritmalarını iki grupta toplamak mümkündür.

- Uninformed Search (*Bilmeden arama*)
- Informed Search (*Bilerek arama*)

Arama işleminin bilmeyerek yapılması demek, arama algoritmasının, probleme özgü kolaylıkları barındırmaması demektir. Yani her durumda aynı şekilde çalışan algoritmalara '*Uninformed Search*' (*bilmeden arama*) ismi verilir. Arama işleminin bilerek yapılması ise, algoritmanın probleme ait bazı özellikleri bünyesinde barındırması ve dolayısıyla arama algoritmasının problem bazlı değişiklik göstermesi demektir [Kaynak: 5]. Bu dokümanda 2 farklı arama algoritmasına yer verilmiştir.

1. İkili Arama (Binary Search): Bilgisayar bilimlerinde bir bilgi kaynağı veya veri yapısı üzerinde problemi her adımda iki parçaya bölerek yapılan arama algoritmasının ismidir. "Divide and Conquer" mantığıyla çalışır. Binary Search kullanılabilmesi için verilerin sıralı olması şarttır. Binary Search'te veri kümesinin en ortasına bakılır. Eğer aranılan değer en ortadaki değerden küçükse aramaya küçük taraf yani sol taraftan, büyükse sağ taraftan devam edilir. Aranan değer bulunana kadar bu işlem küçük parçalarda da uygulanır. Böylece her bir aramada veri kümesinin yarısı elenmiş olur[Kaynak: 3].

Örnek…

Elimizde **9, 7, 6, 4, 3, 2, 1** sıralı dizisi olsun. Bu dizi arasında 6 sayısını aralım.

Dizinin ortasına bakılır. 4 ? 6 ➔ 4 < 6. Aranan değer büyük olduğundan dizinin sağ tarafı dikkate alınmaz ve sol taraf üzerinden arama işlemine devam edilir. Böylece veri kümemizi yarı yarıya küçültmüş oluruz.

9 , 7 , 6 .. bu kez elimizde kalan dizinin tam ortasına bakılır. 7 ? 6 ➔ 7 > 6. Aranan değer küçük olduğundan arama işlemine sağ taraftan devam edilir.

6 = 6 ← Aranan değer bulunmuş olur.

DİZİDE ORTANCA DEĞERİN BULUNMASI

9, 7, 6, 4, 3, 2, 0,1,-1,-2 sıralı dizimiz olsun. Dizinin indisleri ise 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Ortanca değer bulunurken uç değerlerin indisleri toplamı 2'ye bölünür. 4. veya 5. index'i ortanca değer alabilirsiniz.

9, 7, 6, 4, 3, 2, 0, 1, -1, -2 ➔ (0 + 9)/ 2 = 4.5 ortanca değer 4. veya 5. indis alınıp aranan değer ile kıyaslanır.

Aranan değer bulunamazsa tekrar uç değerlere aynı işlem uygulanır.

9, 7, 6, 4, 3 ➔ (0+4)/2 = 2 aramaya sol tarafta devam edersek ortanca değer 2. indis alınıp aranan değer ile kıyaslanır.

Aranan değer bulunamazsa tekrar uç değerlere aynı işlem uygulanır.

9, 7, 6 ➔ (0+1)/2 = 0.5 aramaya sol tarafta devam edersek ortanca değer 1. indis alınıp aranan değer ile kıyaslanır.

9 ← Arama tamamlandı.

Eğer aranan değer dizide bulunamazsa bu durumun uyarısı verilir.

2. Doğrusal Arama (Linear Search): 'Sıralı Arama' olarak da bilinen ve belirli bir değer için bir veri listesini aramaya uygun olan bir arama algoritmasıdır. Eşleşme bulana kadar listenin bütün elemanlarını bir kere kontrol ederek çalışır. Doğrusal arama O(n) karmaşıklığı ile çalışır. En iyi durum ilk karşılaştırmada aranan değerin bulunmasıdır. En kötü durum ise tüm listenin tüm elemanlarını karşılaştırma işlemine sokmaktır [Kaynak: 3].

Kaynak:

1. https://apocalyptra.wordpress.com/
   Erişim: Aralık 2020
2. https://e-bergi.com/y/siralama-algoritmalari/
   Erişim: Aralık 2020
3. http://cagataykiziltan.net/algoritmalar/1-siralama-algoritmalari/6-hizli-siralama/
   Erişim: Aralık 2020
4. http://ceng.harran.edu.tr/demoday/2019/ParallelSortingAlgorithms/Dokumanlar.html
   Erişim: Aralık 2020
5. http://bilgisayarkavramlari.com/2009/11/23/arama-algoritmalari-search-algorithms
   Erişim: Aralık 2020