Bölüm 39

Binary Search

(Yarılama)

39.1 Dizide Bir Öğe Arama

İkil aramayı (yarılama yöntemi) sıralı veri kümelerinde sık sık kullanırız. Örneğin, sözlükte bir sözcüğü ararken, sözlüğün bütün sayfalarını baştan sona doğru aramayız. Alfabetik sıralı olduğunu bildiğimiz için, sözlüğü rasgele açarız. Aradığımız sözcüğün ilk harfleri ile açtığımız sayfadaki sözcüklerin ilk harflerini karşılaştırırız. Aradığımız sözcük o sayfada değilse, sayfaları ileriye ya da geriye doğru çevirerek, aradığımız sözcüğü içeren sayfayı buluruz. Bu algoritma $O(logn)$ grubuna aittir.

İkil arama algoritması, sözlükte arama yaparken izlenen yöntemi izleyen basit bir algoritmadır. İkil (binary) Arama Algoritması aranan veriyi, sıralı dizinin ortanca terimi ile karşılaştırır. Üç durumdan ancak birisi ortaya çıkabilir:

Sıralı dizinin terimleri

$$a[0] < a[1] < a[2] < \ldots < a[n/2] < \ldots < a[n]$$

olsun. Basitliği sağlamak için, diziyi `a[sol...sağ]` simgesiyle gösterelim. Aranan değer x olsun. Ortanca terimin indisini ort= (sol+sağ)/2 ile gösterelim. İkil arama algoritması şöyle çalışır. x öğesi ile `a[ort]` ortanca terimini karşılaştırır.

1. x= a[ort] ise, aranan terim bulunmuş olur.
2. x < a[ort] ise, algoritma a[sol...ort-1] altdizisinde aramaya başlar.
3. x > a[ort] ise, algoritma a[ort+1...sağ] altdizisinde aramaya başlar.
4. Her adımda, yukarıdaki üç işlemden yalnızca birisi yapılarak, altdizi tek öğeli olana kadar devam edilebilir.
5. Bu adımlardan her hangi birisinde x= ortanca olursa, aranan bulunmuş olur; değilse aranan öğenin a[] dizisi içinde olmadığı sonucuna varılır.

Şimdi bu algoritmanın sankikodlarını (pseudo code) yazabiliriz.

a[sol...sağ] sıralı dizisi içinde bir x öğesinin olup olmadığı aranacaktır.

1. sol <= sağ için tekrarla:
   (a) ort= (sol+sağ)/2 //ortancanın indisi
   (b) x= a[ort] ise, aranan terimin indisi ort'dur.

(c) x < a[ort] ise, algoritmayı a[sol...ort-1] altdizisine uygula.

(d) x > a[ort] ise, algoritmayı a[sol+1...sağ] altdizisine uygula.

2. Aranan değer dizide yok; -1 değeri ver.

Bu yalancı kodların dediği işi yapan bir java metodu yazalım:

39.1.1 While döngüsü ile Binary Search Algoritması

İkil arama yapan bir metodu, *while döngüsü* kullanarak yazabiliriz:

```
public static int BinarySearch(Comparable[] a, int sol,
 int sağ,
         Comparable aranan) {
    int t = sol , r = sağ;
    while (sol <= sağ){
      int  ort = (sol+sağ)/2 ;
  int comp = aranan.compareTo(a[ort]);
  if (comp == 0) return ort; // aranan bulundu, return
   index.
    else if (comp < 0) sağ = ort-1 ;
      else  // comp > 0
        sol = ort + 1
    }
    return -1;        // aranan bulunamadı
}
```

Tabii, burada `Comparable` yerine java'da mukayese edilebilen veri tipleri alınabilir. Şimdi yukarıdaki metodu bir java uygulama programı içine koyalım. Örneğimizde `Object[]` yerine `int[]` alınmıştır.

Program 39.1.1.

```
public class Demo {
  public static final int BULUNAMADI = -1;

  /**
   * standard binary arama için her adımda iki mukayese
    yapar
   * BULUNAMADI ise -1 yazar.
   * Bulunduysa, o terimin indisini yazar
   */
  public static int binarySearch(Comparable[] a,
   Comparable x) {
    int sol = 0;
    int sağ = a.length - 1;
    int mid;

    while (sol <= sağ) {
      mid = (sol + sağ) / 2;

      if (a[mid].compareTo(x) < 0)
      sol = mid + 1;
      else if (a[mid].compareTo(x) > 0)
      sağ = mid - 1;
      else
      return mid;
    }

    return BULUNAMADI; // BULUNAMADI = -1
  }

  // Test program

  public static void main(String[] args) {
    int UZUNLUK = 8;
    Comparable[] a = new Integer[UZUNLUK];
    for (int i = 0; i < UZUNLUK; i++)
    a[i] = new Integer(i * 2);

    for (int i = 0; i < UZUNLUK * 2; i++)
    System.out.println("Bulunan : " + i + " indisi : "
    + binarySearch(a, new Integer(i)));

  }
}
```

```
/*
Bulunan : 0 indisi :   0
Bulunan : 1 indisi :  -1
Bulunan : 2 indisi :   1
Bulunan : 3 indisi :  -1
Bulunan : 4 indisi :   2
Bulunan : 5 indisi :  -1
Bulunan : 6 indisi :   3
Bulunan : 7 indisi :  -1
Bulunan : 8 indisi :   4
Bulunan : 9 indisi :  -1
Bulunan : 10 indisi :  5
Bulunan : 11 indisi : -1
Bulunan : 12 indisi :  6
Bulunan : 13 indisi : -1
Bulunan : 14 indisi :  7
Bulunan : 15 indisi : -1
*/
```

39.1.2 Recursive metot ile Binary Search algoritması

Yukarıda while döngüsü ile ikil arama yapan bir metot yazmıştık. Aynı işi recursive (özyineli, kendi kendini çağıran) bir metot ile de yapabiliriz:

```
int rBinarySearch(int[] arr, int alt, int üst, int
 aranan) {
  if (alt <= üst) {
    int ort = (alt + üst) / 2; // ortanca terimin
     indisi.
    if (aranan == arr[ort])
      return ort; // aranan bulundu
    else if (aranan < arr[ort])
      // dizinin sol yarısına uygula
      return rBinarySearch(arr, alt, ort - 1, aranan);
```

```
9         else
            // dizinin sağ yarısına uygula
            return rBinarySearch(arr, ort + 1, üst, aranan);
        }
        return -(alt + 1); // aranan bulunamadı
14    }
```

Bunu bir java uygulama programı içine alalım:

Program 39.1.2.

```
1  public class Demo {
     int[] a = { 1, 11, 22, 33, 44, 55, 66, 77, 88, 99 };
     int x = 44;

     int binarySearch(int[] arr, int alt, int üst, int
      aranan) {
6      if (alt <= üst) {
         int ort = (alt + üst) / 2; // ortanca terimin
          indisi.
         if (aranan == arr[ort])
         return ort; // aranan bulundu
         else if (aranan < arr[ort])
11       // dizinin sol yarısına uygula
         return binarySearch(arr, alt, ort - 1, aranan);
         else
         // dizinin sağ yarısına uygula
         return binarySearch(arr, ort + 1, üst, aranan);
16     }
       return -(alt + 1); // aranan bulunamadı
     }

     public static void main(String[] args) {
21     Demo bs = new Demo();
       // bs.alt = 0; bs.üst = 9;
       int s = bs.binarySearch(bs.a, 0, bs.a.length, bs.x);
       System.out.println(s);
     }
26 }
```

```
/*
```

```
    4
*/
```

39.2 Örnekler

Program 39.2.1.

```
public class Demo {
  String[] a = { "Yüce", "Ulusoy", "Aygün", "Volkan",
   "Çelik", "Can", "Alp","Baydar", "Kazan", "Demir" };
  String x = "Demir";

  int rBinarySearch(String[] str, int alt, int üst,
   String aranan) {
    if (alt <= üst) {
      int ort = (alt + üst) / 2; // ortanca terimin
       indisi.
      if (aranan.compareTo(str[ort]) == 0)
      return ort; // aranan bulundu
      else if (aranan.compareTo(str[ort]) < 0)
      // dizinin sol yarısına uygula
      return rBinarySearch(str, alt, ort - 1, aranan);
      else
      // dizinin sağ yarısına uygula
      return rBinarySearch(str, ort + 1, üst, aranan);
    }
    return -1; // aranan bulunamadı
  }
  public static void main(String[] args) {
    Demo bs = new Demo();
    int s = bs.rBinarySearch(bs.a, 0, bs.a.length, bs.x);
    System.out.println(s);
  }
}
```

```
/**
    -1
*/
```

Program 39.2.2.

```
import java.util.ArrayList;

public class Demo {
  public static void main(String[] args) {

    ArrayList arrayList = new ArrayList();

    arrayList.add("1");
    arrayList.add("2");
    arrayList.add("3");
    arrayList.add("4");
    arrayList.add("5");
    arrayList.add("1");
    arrayList.add("2");

    boolean bulundu = arrayList.contains("4");
    System.out.println("4 ambarda mı ? " + bulundu);

    int index = arrayList.indexOf("4");
    if(index == -1)
      System.out.println("4  ambarda değil.");
    else
      System.out.println("4 ün indisi :" + index);

    int lastIndex = arrayList.lastIndexOf("1");
    if(lastIndex == -1)
      System.out.println("1 ambarda değil");
    else
      System.out.println("1  in sonuncu indisi :" +
       lastIndex);
  }
}
```

```
/**
    4 ambarda mı?       true
    3 ün indisi         :3
    1 in sonuncu indisi :5
*/
```

Program 39.2.3, bağlı bir liste yaratıp, harfleri ambara koyuyor. Sonra `shuffle()` metodu ile onları karıyor. Karma içinden bir harfi

`binarySearch()` metodu ile arıyor.

Program 39.2.3.

```
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class Demo {
  public static void main(String args[])
  {
    List<Character> linkedList = new
     LinkedList<Character>();

    for (char n = 'A'; n <= 'Z'; n++)
    linkedList.add(n);

    Collections.shuffle(linkedList);

    for (Character x : linkedList)
    System.out.print(x + " ");
    System.out.println();

    Collections.sort(linkedList);

    for (Character x : linkedList)
    System.out.print(x + " ");

    System.out.println("\nK harfini ara");
    int i = Collections.binarySearch(linkedList, 'F');

    if (i >= 0) {
      System.out.println("K nın indisi : " + i);
      System.out.println("Bulunan Nesne :" +
       linkedList.get(i));
    }
  }
}
```

```
/**
    L X B H W P I J F M S Y U Z O C K D A V T E R Q N G
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

```
    K harfini ara
    K nın indisi : 5
    Bulunan Nesne : F
*/
```

Özet

Terimleri büyüklük sırasına dizilmiş bir dizide bir verinin olup olmadığını aramak için *yarılama yöntemi* (binary search) kullanılır. Bu yöntem *doğrusal arama* yöntemine göre daha hızlıdır; performansı, n bileşeni olan bir dizi için $O(logn)$'dir. Yarılama yönteminde, dizinin ortanca terimi ile aranan veri karşılaştırılır. Aranan veri dizinin ortanca teriminden küçükse alt yarı dizide, büyükse üst yarı dizide aramaya devam edilir. Aranan veri ortanca terime eşitse, aranan bulunmuştur. Algoritma, bulduğu o terimin indisini verir. Terim bulunamamışsa, alt ya da üst yarı dizide aynı yöntem özyineli (recursive) olarak uygulanır. Yarı dizinin uzunluğu tek terime indiği halde, aranan veriye raslanmamışsa, arananın dizide olmadığı sonucu çıkar; algoritma -1 verir.