Bölüm 2: Diziler

Veri Yapıları

Diziler Nedir?

- Diziler, aynı türdeki verileri düzenli bir şekilde saklamak için kullanılan veri yapılarıdır.
- Diziler, aynı isimle tanımlanan ve bir veya daha fazla veriyi saklayabilen değişkenlerdir.
- İndeksleme ile hızlı erişim sağlarlar.

Dizi Kullanımı

- Dizi elemanları, indeks numaraları ile erişilebilir.
- İlk elemanın indeksi genellikle 0'dan başlar.
- Elemanlara erişmek için dizinin adını ve indeksi kullanabilirsiniz.

- Örnek:
- `Dizi = [10, 20, 30, 40, 50]`
- `İlkEleman = Dizi[0] # İlk eleman 10'dur.`

Dizi Avantajları

- Verileri düzenli ve sıralı bir şekilde saklar.
- Elemanlara hızlı erişim sağlar (indeksleme).
- Verileri gruplandırarak düzenler.

Dizi Dezavantajları

- Sabit boyuttadır, boyutu değiştirilemez.
- Ekleme veya çıkarmalar zaman alabilir (bazı durumlarda tüm diziyi kaydırma gerekebilir).

Dizi Türleri

- Bir boyutlu diziler (tek boyutlu diziler): Elemanlar yalnızca bir satırda saklanır.
- İki boyutlu diziler (çok boyutlu diziler): Elemanlar satır ve sütunlarda saklanır.
- Üç boyutlu diziler ve daha fazlası: Daha karmaşık veri düzenlemelerini destekler.

Dizi Kullanım Alanları

- Programlama dillerinde yaygın olarak kullanılır.
- Verileri düzenlemek ve işlemek için idealdir.
- Matrisler, resimler ve ses verileri gibi yapıları temsil etmek için kullanılır.

Diziler

- Görsel, beş elemanlı bir diziyi temsil ediyor. Dizideki her eleman, bir hücre içinde bulunuyor. Bu görselleştirme dizinin elemanlarını bir arada ve düzenli bir şekilde gösterir.

indis	0	1	2	3	4
eleman	13	-1	4	2	7

Dizi Tanımlama

- `VeriTipi[] diziAdi = new VeriTipi[diziBoyutu];`

- VeriTipi: Dizide saklanacak verilerin türünü belirtir.
- diziAdi: Dizinin adını temsil eder.
- diziBoyutu: Dizinin kaç eleman içereceğini belirtir.

- Örnek:
- `int[] sayilar = new int[5];`

Dizi Elemanlarına Erişim

- Dizi elemanları, indeks numarasıyla erişilir.
- İndeksler 0'dan başlar.

- Örnek:
- `int ilkEleman = sayilar[0]; // İlk elemanı alır`
- `int ikinciEleman = sayilar[1]; // İkinci elemanı alır`

Dizi İlklendirme

- Dizi elemanlarına başlangıç değerleri atanabilir.
- İlklendirme sırasında dizi boyutunu belirtmeye gerek yoktur.
- İnitializers, bir dizinin ilk değerlerini atamak için kullanılır.
- Dizi tanımlandığında, başlatıcılar dizinin başlangıç değerlerini belirlemek için kullanılır.

- Örnek:
- `int[] sayilar = {10, 20, 30, 40, 50};`

Dizi Boyutunu Almak

- `length` özelliği ile dizinin boyutu alınabilir.
- Bu özellik dizinin kaç eleman içerdiğini verir.

- Örnek:
- ```
  int diziBoyutu = sayilar.length; // Dizi boyutu: 5
  ```
  ```

Boyut Belirtilmediğinde

- Dizi boyutu belirtilmediğinde, başlatıcılar boyutu belirler.
- Başlatıcı sayısı, dizinin boyutunu belirler.

- Örnek:
- ```
  int[] n = {1, 2, 3, 4, 5}; // 5 başlatıcı olduğu için n 5 elemanlı bir dizi olur.
  ```
  ```

Örnek Kod Parçası

```
public class DiziOrnegi {
    public static void main(String[] args) {
        // Bir diziyi tanımlama ve başlatma
        int[] sayilar = new int[5]; // 5 elemanlı bir tamsayı dizisi
tanımlandı

        // Diziyi başlatma
        sayilar[0] = 10;
        sayilar[1] = 20;
        sayilar[2] = 30;
        sayilar[3] = 40;
        sayilar[4] = 50;
```

Örnek Kod Parçası

```
        // Dizinin elemanlarına erişme
        System.out.println("Dizinin birinci elemanı: " + sayilar[0]);
        System.out.println("Dizinin üçüncü elemanı: " + sayilar[2]);

        // Dizinin tüm elemanlarını döngü kullanarak yazdırma
        System.out.print("Dizi Elemanları: ");
        for (int i = 0; i < sayilar.length; i++) {
            System.out.print(sayilar[i] + " ");
        }
    }
}
```

OrtalamaHesaplama

```
import java.util.Scanner;

public class OrtalamaHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();

        // Kullanıcıdan elemanları al ve dizi oluştur
        double[] dizi = new double[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextDouble();
        }
```

OrtalamaHesaplama

```
        // Dizi elemanlarını topla
        double toplam = 0;
        for (int i = 0; i < boyut; i++) {
            toplam += dizi[i];
        }
        // Ortalama hesapla
        double ortalama = toplam / boyut;
        // Sonucu yazdır
        System.out.println("Dizi Elemanları: ");
        for (int i = 0; i < boyut; i++) {
            System.out.print(dizi[i] + " ");
        }
        System.out.println("\nOrtalama: " + ortalama);
        scanner.close();
    }
}
```

OrtaDegerHesaplama

```
import java.util.Arrays;
import java.util.Scanner;

public class OrtaDegerHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();
        // Kullanıcıdan elemanları al ve dizi oluştur
        double[] dizi = new double[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextDouble();
        }
        // Diziyi sırala
        Arrays.sort(dizi);
```

OrtaDegerHesaplama

```
        // Ortanca değeri hesapla
        double ortanca;
        if (boyut % 2 == 0) {  // Çift boyutlu dizi için ortanca hesabı
            int orta1 = boyut / 2 - 1;
            int orta2 = boyut / 2;
            ortanca = (dizi[orta1] + dizi[orta2]) / 2;
        } else {  // Tek boyutlu dizi için ortanca hesabı
            int orta = boyut / 2;
            ortanca = dizi[orta];
        } // Sonucu yazdır
        System.out.println("Dizi Elemanları (sıralı): ");
        for (int i = 0; i < boyut; i++) {
            System.out.print(dizi[i] + " ");
        }
        System.out.println("\nOrtanca Değer: " + ortanca);
    }
}
```

ModHesaplama

```
import java.util.Arrays;
import java.util.Scanner;

public class ModHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dizi boyutunu girin: ");
        int boyut = scanner.nextInt();
        // Kullanıcıdan elemanları al ve dizi oluştur
        int[] dizi = new int[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemanı girin: ");
            dizi[i] = scanner.nextInt();
        }
        // Diziyi sırala
        Arrays.sort(dizi);
```

ModHesaplama

```
// Mod değeri ve tekrar sayısını bulma
int enCokTekrarEden = dizi[0];
int enCokTekrarSayisi = 1;
int mevcutTekrarEden = dizi[0];
int mevcutTekrarSayisi = 1;

for (int i = 1; i < boyut; i++) {
    if (dizi[i] == dizi[i - 1]) {
        mevcutTekrarSayisi++;
    } else {
        mevcutTekrarSayisi = 1;
        mevcutTekrarEden = dizi[i];
    }
    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
        enCokTekrarSayisi = mevcutTekrarSayisi;
        enCokTekrarEden = mevcutTekrarEden;
    }
}
```

ModHesaplama

```
// Sonucu yazdır
        System.out.println("Dizi Elemanları (sıralı): " + Arrays.toString(dizi));
        System.out.println("Mod Değeri: " + enCokTekrarEden + " (Tekrar Sayısı: " +
                                                enCokTekrarSayisi + ")");
        scanner.close();
    }
}
```

Çok Boyutlu Diziler Nedir?

- Çok boyutlu diziler, bir dizi içinde birden fazla boyutta veri saklamak için kullanılan veri yapılarıdır.
- Java'da iki boyutlu diziler (matrisler) ve daha fazlasını kullanabiliriz.

İki Boyutlu Diziler (Matrisler)

- İki boyutlu diziler, satır ve sütunlarla tanımlanan tablo benzeri yapılar oluşturur.
- Matrisler, örneğin bir satranç tahtası veya bir resim gibi iki boyutlu verileri temsil etmek için kullanışlıdır.

- Örnek:
- `int[][] matris = new int[3][3]; // 3x3 boyutunda bir matris`

İki Boyutlu Dizilerin İlklendirilmesi

- İki boyutlu dizilere başlangıç değerleri atanabilir.
- İlklendirme sırasında matrisin boyutu ve başlangıç değerleri belirtilmelidir.

- Örnek:
- `int[][] matris = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };`

CokBoyutluDiziOrnegi

```
public class CokBoyutluDiziOrnegi {
    public static void main(String[] args) {
        // İki boyutlu bir dizi (matris) tanımlama ve başlatma
        int[][] matris = new int[3][3]; // 3x3 boyutunda bir matris tanımlandı

        // Matrisi başlatma
        for (int satir = 0; satir < 3; satir++) {
            for (int sutun = 0; sutun < 3; sutun++) {
                matris[satir][sutun] = satir * 3 + sutun + 1;
            }
        }
```

CokBoyutluDiziOrnegi

```
        // Matrisi ekrana yazdırma
        System.out.println("Matrisin İçeriği:");
        for (int satir = 0; satir < 3; satir++) {
            for (int sutun = 0; sutun < 3; sutun++) {
                System.out.print(matris[satir][sutun] + " ");
            }
            System.out.println(); // Yeni satıra geçme
        }
    }
}
```

Üç Boyutlu ve Daha Fazla Boyutlu Diziler

- Java'da üç boyutlu diziler ve daha fazla boyutlu diziler de kullanılabilir.
- Daha fazla boyut, daha karmaşık veri yapılarına izin verir.

- Örnek:
- `// 3x3x3 boyutunda üç boyutlu dizi`
- `int[][][] üçBoyutluDizi = new int[3][3][3];`

UcBoyutluDiziOrnegi

```
public class UcBoyutluDiziOrnegi {
    public static void main(String[] args) {
        // Üç boyutlu bir dizi tanımlama ve başlatma
        int[][][] ucBoyutluDizi = new int[3][3][3]; // 3x3x3 boyutunda
bir üç boyutlu dizi tanımlandı

        // Üç boyutlu diziyi başlatma
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                for (int z = 0; z < 3; z++) {
                    ucBoyutluDizi[x][y][z] = x * 9 + y * 3 + z + 1;
                }
            }
        }
```

UcBoyutluDiziOrnegi

```
        // Üç boyutlu diziyi ekrana yazdırma
        System.out.println("Üç Boyutlu Dizinin İçeriği:");
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                for (int z = 0; z < 3; z++) {
                    System.out.print(ucBoyutluDizi[x][y][z] + " ");
                }
                System.out.println(); // Yeni satıra geçme (y ekseni)
            }
            System.out.println(); // Yeni satıra geçme (x ekseni)
        }
    }
}
```

Algoritma Karmaşıklığı Nedir?

- Algoritma karmaşıklığı, bir algoritmanın çalışma süresini veya kaynak kullanımını analiz eden bir ölçüdür.
- Temel olarak, bir algoritmanın ne kadar hızlı veya verimli çalıştığını belirler.
- Algoritmaların zaman karmaşıklığı, işlemlerinin girdinin boyutuna göre nasıl büyüdüğünü gösterir.

Zaman Karmaşıklığı - Lineer Arama

- **Örnek**: Bir dizinin içinde belirli bir elemanı arama.
- Sıradan bir (lineer) arama algoritması kullanalım.

```
public boolean lineerArama(int[] dizi, int hedef) {
    for (int eleman : dizi) {
        if (eleman == hedef) {
            return true;
        }
    }
    return false;
}
```

Zaman Karmaşıklığı - Lineer Arama

- Lineer arama algoritması, en kötü durumda tüm diziyi dolaşır.
- Algoritmanın zaman karmaşıklığı O(n) olur (n dizinin boyutunu temsil eder).

- **İyi Durum**: Hedef erken bulunur, arama hızlı biter.
- **Kötü Durum**: Hedef dizinin sonunda veya hiç bulunmaz, arama uzun sürer.

Zaman Karmaşıklığı - Eşleştir

- **Örnek**: İki dizi arasında çift elemanları eşleştirme.
- İç içe iki döngü kullanarak bir çift eleman arama algoritması kullanalım.

```
public void eslestir(int[] dizi1, int[] dizi2) {
    for (int eleman1 : dizi1) {
        for (int eleman2 : dizi2) {
            if (eleman1 == eleman2) {
                System.out.println("Eşleşen Çift Eleman: " + eleman1);
            }
        }
    }
}
```

Zaman Karmaşıklığı - Eşleştir

- İç içe döngüler, her elemanı her iki dizide karşılaştırır.
- Algoritmanın zaman karmaşıklığı O(n^2) olur (n dizilerin boyutunu temsil eder).
- O(n^2) algoritmalar büyük veri setleri için verimsizdir.

Statik Diziler

- Statik diziler, yığına (stack) oluşturulur ve otomatik bellek süresine sahiptir.
- Belleği manuel olarak yönetmeye gerek yoktur, ancak bu diziler işlevleri sona erdiğinde yok olur.
- Derleme zamanında sabit bir boyuta sahiptirler.

```
String[] plaka = {"", "Adana", "Adıyaman", "Afyon"};
System.out.println(plaka[1]);
```

Dinamik Diziler

- Dinamik diziler, dinamik bellek süresine sahiptir ve yığına değil, heap (serbest bellek) üzerinde depolanır.
- Çalışma zamanında istediğiniz boyuta sahip olabilirler, ancak bellek tahsisi ve serbest bırakma işlemini kendiniz yapmanız gerekir.

```
int[] dinamikDizi = new int[10]; // Bellek tahsis edilir
// ...
dinamikDizi = null; // Bellek serbest bırakılır
```

Heap ve Yığın Belleği Karşılaştırması

- Heap belleği dinamik veri yapıları ve nesneler için kullanılır.
- Yığın belleği yerel değişkenler ve işlev çağrıları için kullanılır.
- Bellek yönetimi heap belleği için manuel, yığın belleği için otomatiktir.
- Bellek serbest bırakma işlemi (örneğin, delete veya System.gc() kullanarak) manuel olarak yapılabilir.

İndis Sınır Dışı Hatası (IndexOutOfBoundsException)

- Endeks dışı hatalar, bir dizinin belirtilen sınırlarının dışına çıkmaya çalıştığınızda ortaya çıkar.
- Bu hatalar programlarınızın çökmesine neden olabilir.

```
// Dizi uzunluğunu kontrol etme
if (index >= 0 && index < dizi.length) {
    int deger = dizi[index];
    // indis dışı hatayı önler
    } else {
    // indiz dışı hatayı ele alma veya hata mesajı gösterme
}
```

Dizi Elemanına Erişme Zaman Karmaşıklığı

- O(1), sabit zaman karmaşıklığıdır ve işlem süresi girdi boyutundan bağımsızdır.
- Bir dizinin belirli bir elemanına erişmek, genellikle sabit zaman alır çünkü dizi elemanlarının adresi hesaplanabilir.

```
int[] dizi = {10, 20, 30, 40, 50};
// Bir dizinin belirli bir elemanına erişme
int eleman = dizi[2]; // 30
// Bu işlem O(1) karmaşıklığına sahiptir.
```

Dizi Elemanı Arama Zaman Karmaşıklığı

- O(n), lineer zaman karmaşıklığıdır ve işlem süresi girdi boyutuna doğru orantılıdır.
- Bir dizide eleman aramak lineer zaman alır çünkü her elemanı kontrol etmeniz gerekebilir.

```
int[] dizi = {10, 20, 30, 40, 50};
// Bir dizide belirli bir elemanı arama
int aranan = 30;
boolean bulundu = false;

for (int eleman : dizi) {
    if (eleman == aranan) {
        bulundu = true;
        break;
    }
} // Bu işlem O(n) karmaşıklığına sahiptir.
```

Dizi Sıralama Zaman Karmaşıklığı

- Sıralama işleminin zaman karmaşıklığı, kullanılan sıralama algoritmasına bağlıdır.
- Kabarcık Sıralama (Bubble Sort): O(n^2)
- Seçim Sıralama (Selection Sort): O(n^2)
- Ekleme Sıralama (Insertion Sort): O(n^2)
- Hızlı Sıralama (Quick Sort): O(n log n)
- Birleştirme Sıralama (Merge Sort): O(n log n)

Seçim Sıralama (Selection Sort)

- Her adımda dizideki en küçük elemanı bulup dizinin başına yerleştirerek çalışır.

```
int[] dizi = {64, 25, 12, 22, 11};

for (int i = 0; i < dizi.length - 1; i++) {
    int minIndex = i;
    for (int j = i + 1; j < dizi.length; j++) {
        if (dizi[j] < dizi[minIndex]) {
            minIndex = j;
        }
    }
    // Minimum elemanı bulduk, şimdi swap işlemi yapalım
    yerDegistir(dizi[minIndex], dizi[i])
}
```

Alan Karmaşıklığı Nedir?

- Bir algoritmanın/veri yapısının ne kadar bellek kullandığını ifade eder.
- Dizi alan karmaşıklığı, bir dizinin bellekte kapladığı alanı ifade eder.
- Bir dizinin alan karmaşıklığı, dizinin boyutu ve eleman türüne göre değişir.
- Örnekler:
  - Bir tamsayı dizisi (int[]) için, her bir tamsayı 4 byte gerektirir
  - Bir karakter dizisi (char[]) için, her bir karakter 2 byte gerektirir (Unicode için).
  - Bir nesne dizisi (Object[]) için, her nesne referansı 4 byte (32 bit sistemlerde) veya 8 byte (64 bit sistemlerde) gerektirir.
- Bir int[] dizisi, 100 elemandan oluşuyorsa ve her int 4 byte bellek gerektiriyorsa, Alan karmaşıklığı = 100 (eleman sayısı) x 4 (her bir int'in bellek gereksinimi) = 400 byte'dır.

Matris Transpozu

Transpoz matrisi, orijinal matrisin satırlarını sütunlara ve sütunları satırlara dönüştürerek oluşturulur.

```
public class MatrixTranspose {
    public static void main(String[] args) {
        int[][] matris = {
            {1, 2, 3},
            {4, 5, 6}
        };

        // Orijinal matrisi yazdır
        System.out.println("Orijinal Matris:");
        yazdirMatris(matris);
```

Matris Transpozu

```
int satirSayisi = matris.length;
int sutunSayisi = matris[0].length;

// Transpoz matrisi oluştur
int[][] transpozMatris = new int[sutunSayisi][satirSayisi];

for (int i = 0; i < satirSayisi; i++) {
    for (int j = 0; j < sutunSayisi; j++) {
        transpozMatris[j][i] = matris[i][j];
    }
}
```

Matris Transpozu

```
        // Transpoz matrisi yazdır
        System.out.println("Transpoz Matris:");
        yazdirMatris(transpozMatris);
    }
    // Matrisi ekrana yazdırmak için yardımcı fonksiyon
    public static void yazdirMatris(int[][] matris) {
        for (int i = 0; i < matris.length; i++) {
            for (int j = 0; j < matris[i].length; j++) {
                System.out.print(matris[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Diziyi Tersine Çevirme

```
public static void tersineCevir(int[] dizi) {
    int baslangic = 0;
    int bitis = dizi.length - 1;

    while (baslangic < bitis) {
        // Baslangic ve bitis elemanlarini degistir
        int gecici = dizi[baslangic];
        dizi[baslangic] = dizi[bitis];
        dizi[bitis] = gecici;
        // Baslangic indeksi arttir, bitis indeksi azalt
        baslangic++;
        bitis--;
    }
}
```

Tic Tac Toe (X-Oyunu)

- 3x3'lük bir oyun tahtası kullanılarak oynanır.
- Oyuncular sırayla hamle yapar.
- 3 tane yan yana ya da çapraz elemanlarını yerleştiren oyunu kazanır.
- Oyun sonlandığında kazanan veya berabere sonucu bildirilir.

```
Tebrikler, X oyuncusu kazandı!
-------------
| X | O |   |
-------------
| X | O |   |
-------------
| X |   |   |
-------------
```

Tic Tac Toe (X-Oyunu)

```
public class TicTacToe {
    public static void main(String[] args) {
        char[][] tahta = new char[3][3]; // 3x3'lük oyun tahtası
        char oyuncu = 'X'; // İlk oyuncu X ile başlar
        boolean oyunDevamEdiyor = true;
        tahtaDoldur(tahta); // Tahtayı başlangıç durumuyla doldur
        while (oyunDevamEdiyor) {
            tahtayiGoster(tahta);
            hamleYap(tahta, oyuncu);
            oyunDevamEdiyor = oyunDevamEdiyorMu(tahta, oyuncu);
            oyuncu = (oyuncu == 'X') ? 'O' : 'X'; // Oyuncu değişimi
        }
        tahtayiGoster(tahta);
    }
```

Tic Tac Toe (X-Oyunu)

```
public static void tahtaDoldur(char[][] tahta) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            tahta[i][j] = ' ';
        }
    }
}
```

Tic Tac Toe (X-Oyunu)

```
public static void tahtayiGoster(char[][] tahta) {
    System.out.println("-------------");
    for (int i = 0; i < 3; i++) {
        System.out.print("| ");
        for (int j = 0; j < 3; j++) {
            System.out.print(tahta[i][j] + " | ");
        }
        System.out.println("\n-------------");
    }
}
```

Tic Tac Toe (X-Oyunu)

```
public static void hamleYap(char[][] tahta, char oyuncu) {
        Scanner scanner = new Scanner(System.in);
        int satir, sutun;

        do {
            System.out.print("Sıra " + oyuncu + " oyuncusunda. Satır ve sütun seçin (1-3): ");
            satir = scanner.nextInt() - 1;
            sutun = scanner.nextInt() - 1;
        } while (satir < 0 || satir > 2 || sutun < 0 || sutun > 2 || tahta[satir][sutun] != ' ');

        tahta[satir][sutun] = oyuncu;
    }
```

Tic Tac Toe (X-Oyunu)

```
public static boolean oyunDevamEdiyorMu(char[][] tahta, char oyuncu) {
        // Kazanan durumlarını kontrol et
        if ((tahta[0][0] == oyuncu && tahta[0][1] == oyuncu && tahta[0][2] == oyuncu) ||
            (tahta[1][0] == oyuncu && tahta[1][1] == oyuncu && tahta[1][2] == oyuncu) ||
            (tahta[2][0] == oyuncu && tahta[2][1] == oyuncu && tahta[2][2] == oyuncu) ||
            (tahta[0][0] == oyuncu && tahta[1][0] == oyuncu && tahta[2][0] == oyuncu) ||
            (tahta[0][1] == oyuncu && tahta[1][1] == oyuncu && tahta[2][1] == oyuncu) ||
            (tahta[0][2] == oyuncu && tahta[1][2] == oyuncu && tahta[2][2] == oyuncu) ||
            (tahta[0][0] == oyuncu && tahta[1][1] == oyuncu && tahta[2][2] == oyuncu) ||
            (tahta[0][2] == oyuncu && tahta[1][1] == oyuncu && tahta[2][0] == oyuncu)) {
            System.out.println("Tebrikler, " + oyuncu + " oyuncusu kazandı!");
            return false;
        }
```

Tic Tac Toe (X-Oyunu)

```
// Berabere durumunu kontrol et
boolean berabere = true;
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (tahta[i][j] == ' ') {
            berabere = false;
            break;
        }
    }
    if (!berabere) {
        break;
    }
}
```

Tic Tac Toe (X-Oyunu)

```
            if (berabere) {
                System.out.println("Oyun berabere bitti.");
                return false;
            }

            return true;
        }
}
```

Soru

Bir dizinin en kötü durumda hangi işlemi yapmak için O(n) zaman karmaşıklığına sahip olduğunu belirtir misiniz?

A) Diziyi sıralama

B) Dizinin ortalamasını hesaplama

C) Belirli bir elemanın indeksini arama

D) Diziyi tersine çevirme

Soru

Hangi tür dizi, her elemanın kendisinden önceki tüm elemanlarla karşılaştırıldığı sıralama algoritmaları için en kötü durumu temsil eder?

A) Sıralı dizi

B) Rastgele dizi

C) Ters sıralı dizi

D) Çok boyutlu dizi

Soru

Bir dizinin elemanlarını hızlı bir şekilde sıralamak için hangi algoritma kullanılır?

A) Kabarcık sıralama

B) Hızlı sıralama

C) Birleştirme sıralama

D) Seçme sıralama

Soru

Bir dizinin en iyi durumda hangi işlemi yapmak için O(1) zaman karmaşıklığına sahiptir?

A) Diziyi sıralama

B) Dizinin ortalamasını hesaplama

C) Belirli bir elemanın indeksini arama

D) Diziyi tersine çevirme

Soru

Dinamik bir dizi (dynamic array) ve bağlı liste (linked list) arasındaki temel fark nedir?

A) İkisi de aynıdır ve birbirlerinin yerine kullanılabilirler.

B) Dinamik dizi, sabit boyutlu iken bağlı liste dinamik olarak büyüyebilir.

C) Bağlı liste, sabit boyutlu iken dinamik dizi dinamik olarak büyüyebilir.

D) Dinamik dizi ve bağlı liste her zaman aynı boyuttadır.

Soru

Bir dizide hangi işlem O(1) zaman karmaşıklığına sahiptir?

A) Bir elemanın diziden silinmesi
B) Dizinin ortalamasının hesaplanması
C) Dizinin tüm elemanlarının sıralanması
D) Bir elemanın dizinin sonuna eklenmesi

Soru

Dizi arama algoritmalarından "Binary Search" işlemi için hangi koşulun geçerli olması gerekir?

A) Dizi önceden sıralanmalıdır.

B) Dizi rastgele elemanlar içermelidir.

C) Dizi ters sıralı olmalıdır.

D) Dizi sırasız olmalıdır.

Soru

Bir dizinin sonuna bir eleman eklemek için O(1) zaman karmaşıklığına sahip olan veri yapısı nedir?

A) Bağlı liste (linked list)

B) Yığıt (stack)

C) Kuyruk (queue)

D) Dinamik dizi (dynamic array)

Soru

Dizilerin dezavantajlarından biri nedir?

A) Sabit boyutta olmaları
B) Elemanları hızlı bir şekilde sıralama yetenekleri
C) Dinamik boyutlandırılabilir olmaları
D) Verileri rastgele erişme yetenekleri

Soru

Dizi (array) veri yapısının temel avantajlarından biri nedir?

A) Dinamik boyutlandırılabilir olması
B) Elemanların rastgele erişim sağlaması
C) Elemanları sıralama yeteneği
D) Hafif olması

Soru

Bir dizinin en kötü durumda hangi işlemi yapmak için O(n^2) zaman karmaşıklığına sahip olduğunu belirtir misiniz?

A) Diziyi sıralama

B) Dizinin ortalamasını hesaplama

C) Belirli bir elemanın indeksini arama

D) Diziyi tersine çevirme

Soru

Hangi veri yapısı, her elemanın kendisinden önceki elemanla bağlantılı olduğu bir dizi olarak tanımlanır?

A) İkili arama ağacı (binary search tree)

B) Bağlı liste (linked list)

C) Kuyruk (queue)

D) Yığıt (stack)

Soru

Hangi dizi erişim şekli, her elemanın sabit bir sürede erişilebildiği bir dizi türünü temsil eder?

A) Dinamik dizi (dynamic array)

B) Dairesel dizi (circular array)

C) Çok boyutlu dizi (multidimensional array)

D) Sabit boyutlu dizi (static array)

Soru

Dizilerin hangi zayıf yönü, elemanlar eklemek ve çıkarmak için gereken zamanın dizinin boyutuna bağlı olarak değişebilir olmasıdır?

A) Dizilerin rastgele erişim sağlayamaması

B) Dizilerin elemanları sıralayamaması

C) Dizilerin boyutunun sabit olması

D) Dizilerin boyutunun dinamik olmaması

Soru

2D bir dizi (array) içindeki elemanlara erişmek için kullanılan indisler nelerdir?

A) Yükseklik ve genişlik

B) Sıra ve sütun

C) İç ve dış döngü

D) İlk ve son indeks

Soru

2D dizilerin bir elemanına erişmek için kullanılan yöntem nedir?

A) Diziyi tersine çevirme
B) İki boyutlu koordinat sistemi kullanma
C) Dizi elemanlarını sıralama
D) Tüm elemanları tekrarlayarak erişme

Soru

3D bir dizinin her elemanına erişmek için kaç tane indis gerekir?

A) 1
B) 2
C) 3
D) 4

Soru

3D bir dizideki bir elemana erişmek için kullanılan indisler nelerdir?

A) Yükseklik, derinlik ve genişlik
B) Sıra, sütun ve katman
C) İç, dış ve üst döngü
D) İlk, orta ve son indeks

SON