EDAA45 Programmering, grundkurs

Läsvecka 10: Matriser, Typparametrar

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

10 Matriser, Typparametrar

- Veckans labb: `maze`
- Matriser
- Typparametrar

Veckans labb: maze

Veckans labb: `maze`

Grunduppgift:

- Implementera en algoritm som hittar ut ur en labyrint.
- En labyrint representeras av en **matris**,
  närmare bestämt en **vektor av vektorer** med **booelska** värden:
  `Vector[Vector[Boolean]]`

Veckans labb: `maze`

Grunduppgift:

- Implementera en algoritm som hittar ut ur en labyrint.
- En labyrint representeras av en **matris**,
  närmare bestämt en **vektor av vektorer** med **booelska** värden:
  `Vector[Vector[Boolean]]`
  Där de två olika sanningsvärdena representerar följande:
  - **`true`** om det **finns en vägg** på en viss plats i matrisen
  - **`false`** om det **inte** finns en vägg på en viss plats i matrisen

Veckans labb: `maze`

Grunduppgift:

- Implementera en algoritm som hittar ut ur en labyrint.
- En labyrint representeras av en **matris**,
  närmare bestämt en **vektor av vektorer** med **booelska** värden:
  `Vector[Vector[Boolean]]`
  Där de två olika sanningsvärdena representerar följande:
  - **`true`** om det **finns en vägg** på en viss plats i matrisen
  - **`false`** om det **inte** finns en vägg på en viss plats i matrisen
- Använd enkel idé (som inte ger kortaste vägen):
  Behåll vänster hand i kontakt med väggen och gå tills du når utgången.
- Vad krävs av labyrinten för att detta ska fungera?

Veckans labb: `maze`

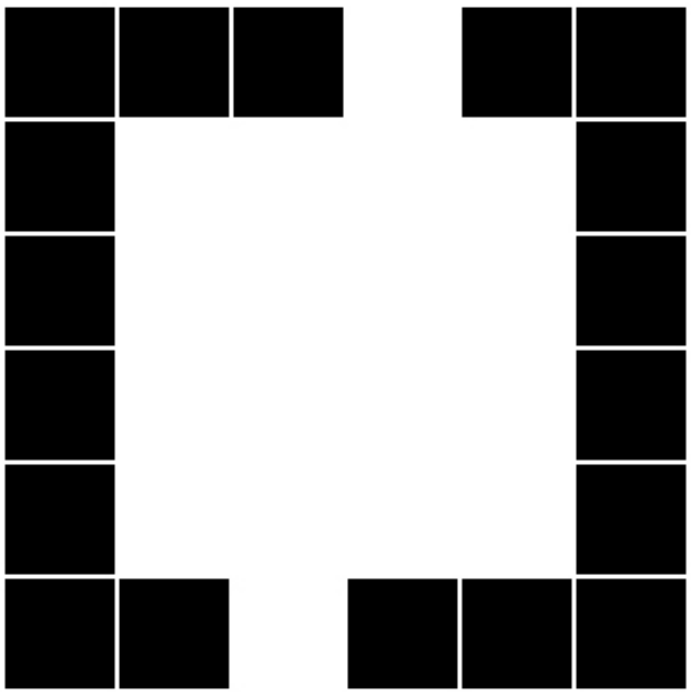
Grunduppgift:

- Implementera en algoritm som hittar ut ur en labyrint.
- En labyrint representeras av en **matris**,
  närmare bestämt en **vektor av vektorer** med **booelska** värden:
  `Vector[Vector[Boolean]]`
  Där de två olika sanningsvärdena representerar följande:
  - **`true`** om det **finns en vägg** på en viss plats i matrisen
  - **`false`** om det **inte** finns en vägg på en viss plats i matrisen
- Använd enkel idé (som inte ger kortaste vägen):
  Behåll vänster hand i kontakt med väggen och gå tills du når utgången.
- Vad krävs av labyrinten för att detta ska fungera?

Extrauppgift:

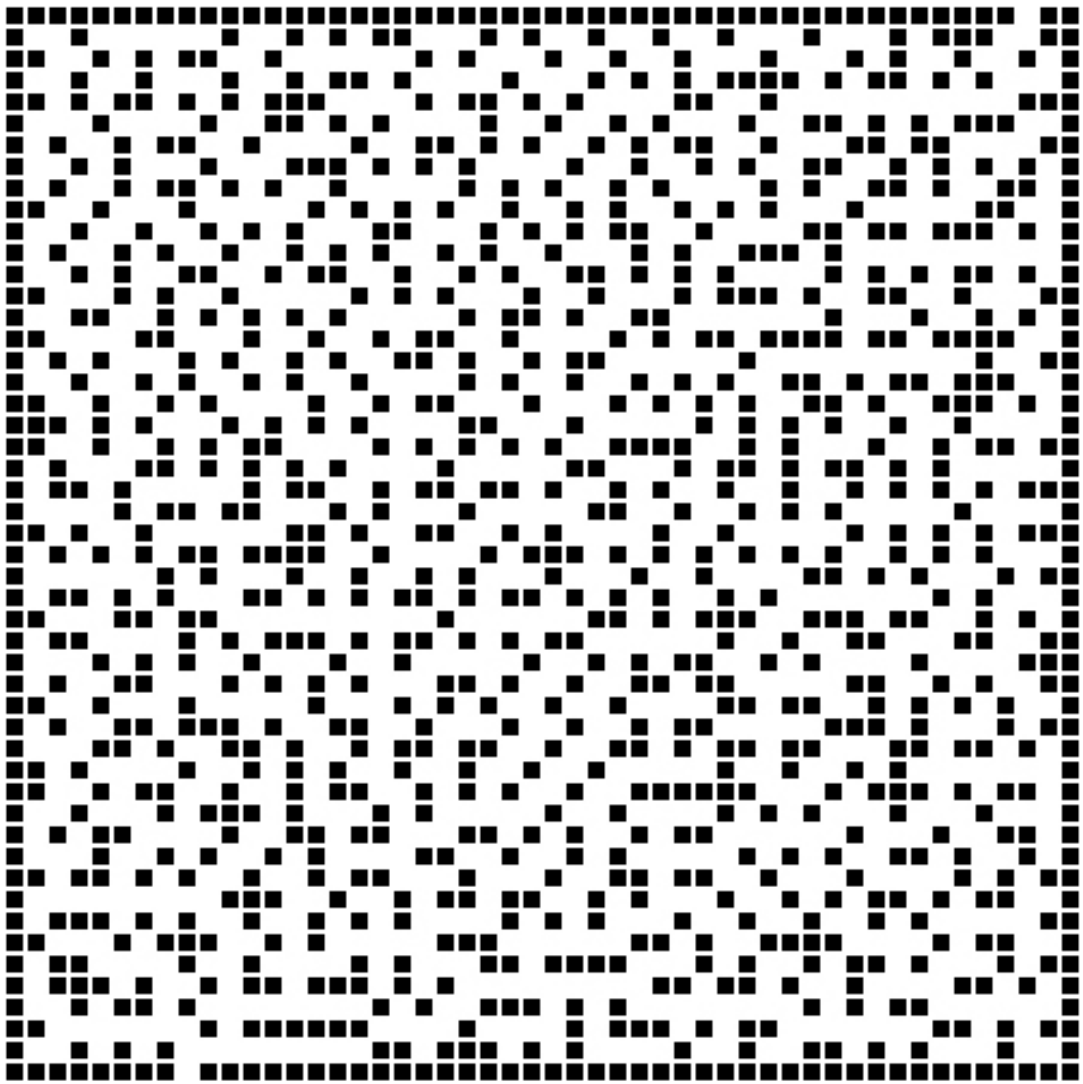
- Generera slumpmässig labyrint
- Algoritmen (*Prims algoritm*) är given i pseudokod

Labyrint som booelsk matris



true	true	true	false	true	true
true	false	false	false	false	true
true	false	false	false	false	true
true	false	false	false	false	true
true	false	false	false	false	true
true	true	false	true	true	true

Slumpmässig labyrint



Matriser

Vad är en matris?

- En **matris** inom **matematiken** innehåller **rader** med lika många tal och **kolumner** med lika många tal.
- En matris av dimension $m \times n$ har $m \cdot n$ stycken element.
- En matris $M_{2,5}$ ritas inom matematiken ofta så här:

$$M = \begin{pmatrix} 5 & 2 & 42 & 4 & 5 \\ 3 & 4 & 18 & 6 & 7 \end{pmatrix}$$

Vad är en matris?

- En **matris** inom **matematiken** innehåller **rader** med lika många tal och **kolumner** med lika många tal.
- En matris av dimension $m \times n$ har $m \cdot n$ stycken element.
- En matris $M_{2,5}$ ritas inom matematiken ofta så här:

$$M = \begin{pmatrix} 5 & 2 & 42 & 4 & 5 \\ 3 & 4 & 18 & 6 & 7 \end{pmatrix}$$

- Indexering inom matematiken sker från 1 (men oftast från 0 i datorprogram).
- Vad har talet 42 för index i matrisen M ovan?

En matris med array av arrayer

Inom programmering används ordet **matris** ofta för att beteckna en **nästlade struktur** i två dimensioner, till exempel en instans av typen `Array[Array[Int]]`

```
1 scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
2 xss: Array[Array[Int]] = Array(Array(5, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
```

En matris med array av arrayer

Inom programmering används ordet **matris** ofta för att beteckna en **nästlade struktur** i två dimensioner, till exempel en instans av typen `Array[Array[Int]]`

```
1  scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
2  xss: Array[Array[Int]] = Array(Array(5, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
```

Man indexerar i en nästlad sekvens med upprepad `apply`:

```
1  scala> xss(0)(2)
2  res0: ???                    // Vad är typ och värde?
3
4  scala> xss.apply(0).apply(2)
5  res1: ???                    // Vad är typ och värde?
6
7  scala> xss(0)
8  res2: ???                    // Vad är typ och värde?
```

En matris med array av arrayer

Inom programmering används ordet **matris** ofta för att beteckna en **nästlade struktur** i två dimensioner, till exempel en instans av typen `Array[Array[Int]]`

```
1  scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
2  xss: Array[Array[Int]] = Array(Array(5, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
```

Man indexerar i en nästlad sekvens med upprepad `apply`:

```
1  scala> xss(0)(2)
2  res0: Int = 42
3
4  scala> xss.apply(0).apply(2)
5  res1: Int = 42
6
7  scala> xss(0)
8  res2: Array[Int] = Array(5, 2, 42, 4, 5)
```

Uppdatering av en förändringsbar nästlad struktur

Man kan förändra en array av arrayer ”på plats” med tilldelning:

```
1  scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
2
3  scala> xss(0)(0) = 100
4
5  scala> xss
6  res0: ???
7
8  scala> xss(0)(2) = xss(0)(2) - 1
9
10 scala> xss
11 res1: ???
12
13 scala> xss(1) = Array.fill(5)(-1)
14
15 scala> xss
16 res2: ???
```

Uppdatering av en förändringsbar nästlad struktur

Man kan förändra en array av arrayer ”på plats” med tilldelning:

```
1  scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
2
3  scala> xss(0)(0) = 100
4
5  scala> xss
6  res0: Array[Array[Int]]=Array(Array(100, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
7
8  scala> xss(0)(2) = xss(0)(2) - 1
9
10 scala> xss
11 res1: Array[Array[Int]]=Array(Array(100, 2, 41, 4, 5), Array(3, 4, 18, 6, 7))
12
13 scala> xss(1) = Array.fill(5)(-1)
14
15 scala> xss
16 res2: Array[Array[Int]]=Array(Array(100, 2, 41, 4, 5), Array(-1,-1,-1,-1,-1))
```

Några olika sätt att skapa förändringsbara matriser

Det jobbiga, primitiva sättet:

```
scala> val xs = new Array[Array[Int]](2)
xs: Array[Array[Int]] = Array(null, null)

scala> for (i <- xs.indices) {xs(i) = new Array[Int](5)}

scala> xs
res0: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))

scala> println(xs)
[[I@196a99d0
```

Enklare sätt:

```
scala> val xs = Array.ofDim[Int](2,5)
xs: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))
```

Enklare och tydligare sätt, där initialvärdet anges explicit:

```
scala> Array.fill(2,5)(0)
res37: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))
```

Exempel på skapande av oföränderlig nästlad struktur

Om du kan beräkna initialvärde direkt, använd `Vector.fill`:

```
def fill[A](n1: Int, n2: Int)(elem: => A): Vector[Vector[A]]
```

```
1  scala> Vector.fill(2,5)(scala.util.Random.nextInt(6) + 1)
2  res0:
3    typ???
4    värde???
```

Om du kan beräkna initialvärde ur index, använd `Vector.tabulate`:

```
def tabulate[A](n1: Int, n2: Int)(f: (Int, Int) => A): Vector[Vector[A]]
```

```
1  scala> Vector.tabulate(5,2)((x,y) => x + y + 1)
2  res1:
3    typ???
4    värde???
```

Exempel på skapande av oföränderlig nästlad struktur

Om du kan beräkna initialvärde direkt, använd `Vector.fill`:

```
def fill[A](n1: Int, n2: Int)(elem: => A): Vector[Vector[A]]
```

```
scala> Vector.fill(2,5)(scala.util.Random.nextInt(6) + 1)
res0:
  scala.collection.immutable.Vector[scala.collection.immutable.Vector[Int]] =
  Vector(Vector(1, 2, 6, 2, 1), Vector(1, 4, 3, 3, 2))
```

Om du kan beräkna initialvärde ur index, använd `Vector.tabulate`:

```
def tabulate[A](n1: Int, n2: Int)(f: (Int, Int) => A): Vector[Vector[A]]
```

```
scala> Vector.tabulate(5,2)((x,y) => x + y + 1)
res1:
  scala.collection.immutable.Vector[scala.collection.immutable.Vector[Int]] =
  Vector(Vector(1,2), Vector(2,3), Vector(3,4), Vector(4,5), Vector(5, 6))
```

Uppdatering av en oföränderlig nästlad struktur

Uppdatering av endimensionell struktur med `xs.updated`:

```
def updated[A](index: Int, elem: A): Vector[A]
```

```
1  scala> var xs = Vector.tabulate(5)(x => x + 1)
2  xs: typ??? = värde???
3
4  scala> xs = xs.updated(1, 42)
5  xs: typ??? = värde???
```

Uppdatering av nästlad struktur i två dimensioner:

```
1  scala> var xss = Vector.tabulate(2, 5)((x,y) => x + y + 1)
2  xss:
3    typ??? =
4    värde???
5
6  scala> xss = xss.updated(0, xss(0).updated(1, 42))
7  xss:
8    typ??? =
9    värde???
```

Uppdatering av en oföränderlig nästlad struktur

Uppdatering av endimensionell struktur med `xs.updated`:

```
def updated[A](index: Int, elem: A): Vector[A]
```

```
scala> var xs = Vector.tabulate(5)(x => x + 1)
xs: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3, 4, 5)

scala> xs = xs.updated(1, 42)
xs: scala.collection.immutable.Vector[Int] = Vector(1, 42, 3, 4, 5)
```

Uppdatering av nästlad struktur i två dimensioner:

```
scala> var xss = Vector.tabulate(2, 5)((x,y) => x + y + 1)
xss:
  scala.collection.immutable.Vector[scala.collection.immutable.Vector[Int]] =
  Vector(Vector(1, 2, 3, 4, 5), Vector(2, 3, 4, 5, 6))

scala> xss = xss.updated(0, xss(0).updated(1, 42))
xss:
  scala.collection.immutable.Vector[scala.collection.immutable.Vector[Int]] =
  Vector(Vector(1, 42, 3, 4, 5), Vector(2, 3, 4, 5, 6))
```

Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats:

```
scala> val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)

scala> for (???) {
         for (???) {
           print(xss(i)(j) + " ")
         }
         println
       }

1 2 3 4 5
2 3 4 5 6
```

Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats:

```
scala> val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)

scala> for (i <- xss.indices) {
         for (j <- xss(i).indices) {
           print(xss(i)(j) + " ")
         }
         println
       }

1 2 3 4 5
2 3 4 5 6
```

Övningsexempel: Yatzy

Skapa en funktion `roll` som ger utfallet av n st tärningskast:

```
scala> import scala.util.Random

scala> def roll(n: Int): Vector[Int] = ???
```

Skapa en funktion `isYatzy` som ger **`true`** om alla utfall är lika:

```
scala> def isYatzy(xs: Vector[Int]): Boolean = ???
```

Du kan anta att xs.length > 0
Tips: använd metoden xs.forall:

```
def forall[A](p: A => Boolean): Boolean
```

Övningsexempel: Yatzy

Skapa en funktion `roll` som ger utfallet av n st tärningskast:

```
scala> import scala.util.Random

scala> def roll(n: Int): Vector[Int] = Vector.fill(n)(Random.nextInt(6) + 1)
```

Skapa en funktion `isYatzy` som ger **`true`** om alla utfall är lika:

```
scala> def isYatzy(xs: Vector[Int]): Boolean = xs.forall(x => x == xs(0))
```

Du kan anta att xs.length > 0
Tips: använd metoden xs.forall:

```
def forall[A](p: A => Boolean): Boolean
```

Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats: (vad har xss för typ?)

```
1  scala> val xss = Vector.fill(100)(roll(5))
2
3  scala> for (???) {
4           for (???) {
5             print(s"($i)($j) == " + xss(i)(j) + " ")
6           }
7           println(isYatzy(???))
8         }
9
10 (0)(0) == 5 (0)(1) == 3 (0)(2) == 4 (0)(3) == 1 (0)(4) == 3 false
11 (1)(0) == 3 (1)(1) == 3 (1)(2) == 6 (1)(3) == 3 (1)(4) == 1 false
12 (2)(0) == 3 (2)(1) == 4 (2)(2) == 2 (2)(3) == 2 (2)(4) == 1 false
13 (3)(0) == 5 (3)(1) == 2 (3)(2) == 6 (3)(3) == 5 (3)(4) == 1 false
14 (4)(0) == 4 (4)(1) == 6 (4)(2) == 4 (4)(3) == 1 (4)(4) == 4 false
15 (5)(0) == 3 (5)(1) == 4 (5)(2) == 6 (5)(3) == 5 (5)(4) == 1 false
16 (6)(0) == 4 (6)(1) == 6 (6)(2) == 2 (6)(3) == 2 (6)(4) == 6 false
17 (7)(0) == 2 (7)(1) == 5 (7)(2) == 3 (7)(3) == 6 (7)(4) == 2 false
18 (8)(0) == 4 (8)(1) == 4 (8)(2) == 6 (8)(3) == 1 (8)(4) == 4 false
19 (9)(0) == 3 (9)(1) == 3 (9)(2) == 3 (9)(3) == 3 (9)(4) == 3 true
20 (10)(0) == 1 (10)(1) == 2 (10)(2) == 4 (10)(3) == 3 (10)(4) == 3 false
21 (11)(0) == 6 (11)(1) == 5 (11)(2) == 4 (11)(3) == 1 (11)(4) == 5 false
```

Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats: (xss är en `Vector[Vector[Int]]`)

```
scala> val xss = Vector.fill(100)(roll(5))

scala> for (i <- xss.indices) {
         for (j <- xss(i).indices) {
           print(s"($i)($j) == " + xss(i)(j) + " ")
         }
         println(isYatzy(xss(i)))
       }

(0)(0) == 5 (0)(1) == 3 (0)(2) == 4 (0)(3) == 1 (0)(4) == 3 false
(1)(0) == 3 (1)(1) == 3 (1)(2) == 6 (1)(3) == 3 (1)(4) == 1 false
(2)(0) == 3 (2)(1) == 4 (2)(2) == 2 (2)(3) == 2 (2)(4) == 1 false
(3)(0) == 5 (3)(1) == 2 (3)(2) == 6 (3)(3) == 5 (3)(4) == 1 false
(4)(0) == 4 (4)(1) == 6 (4)(2) == 4 (4)(3) == 1 (4)(4) == 4 false
(5)(0) == 3 (5)(1) == 4 (5)(2) == 6 (5)(3) == 5 (5)(4) == 1 false
(6)(0) == 4 (6)(1) == 6 (6)(2) == 2 (6)(3) == 2 (6)(4) == 6 false
(7)(0) == 2 (7)(1) == 5 (7)(2) == 3 (7)(3) == 6 (7)(4) == 2 false
(8)(0) == 4 (8)(1) == 4 (8)(2) == 6 (8)(3) == 1 (8)(4) == 4 false
(9)(0) == 3 (9)(1) == 3 (9)(2) == 3 (9)(3) == 3 (9)(4) == 3 true
(10)(0) == 1 (10)(1) == 2 (10)(2) == 4 (10)(3) == 3 (10)(4) == 3 false
(11)(0) == 6 (11)(1) == 5 (11)(2) == 4 (11)(3) == 1 (11)(4) == 5 false
```

Iterera över nästlad struktur med nästlad foreach

Iterera med nästlad foreach-sats:

```
1  scala> val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)
2
3  xss.foreach{ xs => ??? ; println }
4
5  1 2 3 4 5
6  2 3 4 5 6
```

Iterera över nästlad struktur med nästlad foreach

Iterera med nästlad foreach-sats:

```
1  scala> val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)
2
3  xss.foreach{ xs => xs.foreach{ x => print(x + " ") }; println }
4
5  1 2 3 4 5
6  2 3 4 5 6
```

Nästlade for-uttryck

Iterera med **nästlad for-yield**:

```
scala> val xss = for (i <- 1 to 2) yield {
                   for (j <- 1 to 5) yield i + j + 1
                 }
xss:
  scala.collection.immutable.IndexedSeq[
    scala.collection.immutable.IndexedSeq[Int]] =
      ???
```

Om man skriver så här får man en endimensionell struktur:

```
scala> val xs = for (i <- 1 to 2; j <- 1 to 5) yield i + j + 1
xs:
  scala.collection.immutable.IndexedSeq[Int] =
    ???
```

Nästlade for-uttryck

Iterera med **nästlad for-yield**:

```
scala> val xss = for (i <- 1 to 2) yield {
                   for (j <- 1 to 5) yield i + j + 1
                 }
xss:
  scala.collection.immutable.IndexedSeq[
    scala.collection.immutable.IndexedSeq[Int]] =
      Vector(Vector(3, 4, 5, 6, 7), Vector(4, 5, 6, 7, 8))
```

Om man skriver så här får man en endimensionell struktur:

```
scala> val xs = for (i <- 1 to 2; j <- 1 to 5) yield i + j + 1
xs:
  scala.collection.immutable.IndexedSeq[Int] =
    Vector(3, 4, 5, 6, 7, 4, 5, 6, 7, 8)
```

Nästlade map-uttryck

Iterera med **nästlade map-uttryck**:

```
scala> val xss = (1 to 2).map(i => (1 to 5).map(j => i + j + 1))
xss:
  scala.collection.immutable.IndexedSeq[
    scala.collection.immutable.IndexedSeq[Int]] =
      ???
```

Nästlade map-uttryck

Iterera med **nästlade map-uttryck**:

```
scala> val xss = (1 to 2).map(i => (1 to 5).map(j => i + j + 1))
xss:
  scala.collection.immutable.IndexedSeq[
    scala.collection.immutable.IndexedSeq[Int]] =
      Vector(Vector(3, 4, 5, 6, 7), Vector(4, 5, 6, 7, 8))
```

Matris som Array med Array med heltal i Java

```
public class ArrayMatrix {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

Matris som Array med Array med heltal i Java

```
public class ArrayMatrix {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

Övning: skriv en metod `fillRnd` som fyller en heltalsmatris med slumptal 1 till n:

Matris som Array med Array med heltal i Java

```
public class ArrayMatrix {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

Övning: skriv en metod `fillRnd` som fyller en heltalsmatris med slumptal 1 till n:
`public static void fillRnd(int[][] m, int n){ /* ??? */ }`

Matris som Array med Array med heltal i Java

```
public class ArrayMatrix {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

Övning: skriv en metod `fillRnd` som fyller en heltalsmatris med slumptal 1 till n:

```
public static void fillRnd(int[][] m, int n){ /* ??? */ }
```

Tips: använd en nästlad for-sats och:

```
(int) (Math.random * n + 1)   // (int) motsvarar Scalas asInstanceOf[Int]
```

Om veckans övningar

- Träna på att iterera i nästlade strukurer
- Fortsätt jobba med Yatzy-exemplet
- Övning 2f) ger träning i att skapa en **imperativ** algoritm:
  lös `isYatzy` med **`while`**-sats (kunde varit del av en tenta...)
- Extrauppgiften 7 är en bra träning på matriser där du ska bygga ett enkelt yatzy-spel i terminalen (kunde varit del av en tenta...)
- Uppgift 3 är en förberedelse inför nästa veckas labb: `survey` då vi ska analysera enkäter och kombinera matriser & registrering & sortering.

Grunduppgift 3, utgör början på labb survey

Specification **Table**

```
object Table {
  /** Creates a new Table from fileName with columns split by sep */
  def fromFile(fileName: String, separator: Char = ';'): Table = ???
}
case class Table(
  data: Vector[Vector[String]],
  headings: Vector[String],
  sep: String){
  /** A 2-tuple with (number of rows, number of columns) in data */
  val dim: (Int, Int) = ???

  /** The element in row r an column c of data, counting from 0 */
  def apply(r: Int, c: Int): String = ???

  /** The row-vector r in data, counting from 0 */
  def row(r: Int): Vector[String]= ???

  /** The column-vector c in data, counting from 0 */
  def col(c: Int): Vector[String] = ???

  /** A map from heading to index counting from 0 */
  lazy val indexOfHeading: Map[String, Int] = ???

  /** The column-vector with heading h in data */
  def col(h: String): Vector[String] = ???
```

Fördjupningsuppgift 8: skapa en generisk matris-klass

```
case class Matrix[T](data: Vector[Vector[T]]){

  def foreachRowCol(f: (Int, Int, T) => Unit): Unit =
    for (r <- data.indices) {
      for (c <- data(r).indices) {
        f(r, c, data(r)(c))
      }
    }

  def map[U](f: T => U): Matrix[U] = Matrix(data.map(_.map(f)))

  /** The element at row r and column c */
  def apply(r: Int, c: Int): T = ???

  /** Gives Some[T](element) at index (r, c) if within index bounds, else None */
  def get(r: Int, c: Int): Option[T] = ???

  /** The row vector of row r */
  def row(r: Int): Vector[T] = ???

  /** The column vector of column c */
  def col(c: Int): Vector[T] = ???

  /** A new Matrix with element at row r and col c updated */
  def updated(r: Int, c: Int, value: T): Matrix[T] = ???
}
object Matrix {
  def fill[T](rowSize: Int, colSize: Int)(init: T): Matrix[T] =
    new Matrix(Vector.fill(rowSize)(Vector.fill(colSize)(init)))
}
```

Typparametrar

Vad är en typparameter?

- En **typparameter** gör det möjligt att ge ett **typargument**
- En **fri** typparameter kan bindas till vilken typ som helst
- Bindingen sker vid **kompileringstid**
- En typparameter är **fri** om den **inte** fått något värde i omslutande deklarationer, annars **bunden**.

Exempel: **generisk** metod:

```
def tnirp[A](x: A):Unit = println(x.toString.reverse)
```

Vad är en typparameter?

- En **typparameter** gör det möjligt att ge ett **typargument**
- En **fri** typparameter kan bindas till vilken typ som helst
- Bindingen sker vid **kompileringstid**
- En typparameter är **fri** om den **inte** fått något värde i omslutande deklarationer, annars **bunden**.

Exempel: **generisk** metod:

```
def tnirp[A](x: A):Unit = println(x.toString.reverse)
```

Exempel: **generisk** klass:

```
class Cell[A](var value: A){
  override def toString = s"Cell($value)"
  def concat(x: A): Cell[String] = new Cell(value.toString + x)   // A bunden
  def tnirp[B](x: B): Unit = println(x.toString.reverse)          // B fri
}
```

Vad är en typparameter?

- En **typparameter** gör det möjligt att ge ett **typargument**
- En **fri** typparameter kan bindas till vilken typ som helst
- Bindingen sker vid **kompileringstid**
- En typparameter är **fri** om den **inte** fått något värde i omslutande deklarationer, annars **bunden**.

Exempel: **generisk** metod:

```
def tnirp[A](x: A):Unit = println(x.toString.reverse)
```

Exempel: **generisk** klass:

```
class Cell[A](var value: A){
  override def toString = s"Cell($value)"
  def concat(x: A): Cell[String] = new Cell(value.toString + x)   // A bunden
  def tnirp[B](x: B): Unit = println(x.toString.reverse)          // B fri
}
```

- **Skuggning kan förekomma**: Om `tnirp` i `Cell` hade använt namnet A på sin typparameter hade den **skuggat** klassens typparameter och blivit en ny fri typparameter.

Exempel: Generisk funktion

Vad händer här?

```
1
2  scala> def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
3  ???
4
5
6
7  scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse
8
9  scala> skrikBaklänges("gurka är gott")
10 res0: ???
```

Exempel: Generisk funktion

Vad händer här?

```
1
2  scala> def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
3  <console>:11: error: not found: type T
4         def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
5                                ^
6
7  scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse
8
9  scala> skrikBaklänges("gurka är gott")
10 res0: ???
```

Exempel: Generisk funktion

Vad händer här?

```
1
2  scala> def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
3  <console>:11: error: not found: type T
4         def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
5                                ^
6
7  scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse
8
9  scala> skrikBaklänges("gurka är gott")
10 res0: String = TTOG RÄ AKRUG
```

Exempel: Generisk case-klass

```
1  scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse
2
3  scala> case class Grönsak(whatever: A)
4  ???
5
6
7  scala> case class Grönsak[A](whatever: A)
8
9  scala> Grönsak("gurka")
10 res1: ???
11
12 scala> skrikBaklänges(Grönsak(42))
13 res2: ???
14
15 scala> Grönsak[Int](42)
16 res3: ???
17
18 scala> Grönsak[String](42)
19 ???
20
21
22
23                        ^
```

Exempel: Generisk case-klass

```
scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse

scala> case class Grönsak(whatever: A)
<console>:11: error: not found: type A
       case class Grönsak(whatever: A)
                                    ^
scala> case class Grönsak[A](whatever: A)

scala> Grönsak("gurka")
res1: Grönsak[String] = Grönsak(gurka)

scala> skrikBaklänges(Grönsak(42))
res2: String = )24(KASNÖRG

scala> Grönsak[Int](42)
res3: Grönsak[Int] = Grönsak(42)

scala> Grönsak[String](42)
<console>:14: error: type mismatch;
 found   : Int(42)
 required: String
       Grönsak[String](42)
                       ^
```

Fallgrop: likhet av array

```
scala> Vector.fill(5)(42) == Vector.fill(5)(42)
res0: ???

scala> Array.fill(5)(42) == Array.fill(5)(42)
res1: ???
```

Fallgrop: likhet av array

```
scala> Vector.fill(5)(42) == Vector.fill(5)(42)
res0: Boolean = true

scala> Array.fill(5)(42) == Array.fill(5)(42)
res1: Boolean = false  // AAAARRGH!!! :(
```

Primitiva arrayer har en equals-metod som ger referenslikhet, **inte** innehållslikhet.

Kolla likhet av array-matris med nästlad while

```
scala> def isEqual(xss: Array[Array[Int]], yss: Array[Array[Int]]) = {
         var i = 0
         var allEqual = true
         while (???) {
           var j = 0
           while (???) {
             if (xss(i)(j) != yss(i)(j)) ???
             j += 1
           }
           i += 1
         }
         allEqual
       }

scala> val (xss, yss) = (Array.fill(5,2)(42), Array.fill(5,2)(42))

scala> isEqual(xss, yss)

scala> yss(4)(1) = 0

scala> isEqual(xss, yss)
```

Kolla likhet av array-matris med nästlad while

```
1  scala> def isEqual(xss: Array[Array[Int]], yss: Array[Array[Int]]) = {
2           var i = 0
3           var allEqual = true
4           while (i < xss.length && allEqual) {
5             var j = 0
6             while (j < xss(i).length && allEqual) {
7               if (xss(i)(j) != yss(i)(j)) allEqual = false
8               j += 1
9             }
10            i += 1
11          }
12          allEqual
13        }
14
15 scala> val (xss, yss) = (Array.fill(5,2)(42), Array.fill(5,2)(42))
16
17 scala> isEqual(xss, yss)
18
19 scala> yss(4)(1) = 0
20
21 scala> isEqual(xss, yss)
```

Fördjupning: Fallgrop typradering (eng. *type erasure*)

Informationen om typerna i typparametrar raderas innan kodgenerering av prestandaskäl och **typinformationen finns ej vid runtime**.

```
1  scala> val xs = Vector(1,2,3)
2  xs: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)
3
4  scala> val ys = xs.map(_.toDouble)
5  ys: scala.collection.immutable.Vector[Double] = Vector(1.0, 2.0, 3.0)
6
7  scala> def hasDoubles[T](xs: Vector[T]): Boolean = xs match {
8           case _: Vector[Int] => false
9           case _: Vector[Double] => true
10        }
11
12 <console>:13: warning: ???
13
14
15                         ^
16 <console>:14: warning: ???
17
18
19                         ^
20 <console>:14: warning: ???
```

Fördjupning: Fallgrop typradering (eng. *type erasure*)

Informationen om typerna i typparametrar raderas innan kodgenerering av prestandaskäl och **typinformationen finns ej vid runtime**.

```
scala> val xs = Vector(1,2,3)
xs: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)

scala> val ys = xs.map(_.toDouble)
ys: scala.collection.immutable.Vector[Double] = Vector(1.0, 2.0, 3.0)

scala> def hasDoubles[T](xs: Vector[T]): Boolean = xs match {
         case _: Vector[Int] => false
         case _: Vector[Double] => true
       }

<console>:13: warning: non-variable type argument Int in type pattern scala.co
is unchecked since it is eliminated by erasure
                case _: Vector[Int] => false
                        ^
<console>:14: warning: non-variable type argument Double in type pattern scala
is unchecked since it is eliminated by erasure
                case _: Vector[Double] => true
                        ^
<console>:14: warning: unreachable code: case _: Vector[Double] => true
```

Fördjupning: Dynamisk typtest vid typpradering

Typtest vid körtid med nästlad matchning:

```
scala> def hasDoubles2[T](xs: Vector[T]): Boolean = xs match {
         case x +: xs => x match {
           case _: Double => true
           case _ => false
         }
         case _ => false
       }

scala> hasDoubles2(Vector(1.0))    // funkar!
```

Typtest vid körtid med match och gard med `isInstanceOf`:

```

scala> def hasDoubles3[T](xs: Vector[T]): Boolean = xs match {
         case x +: xs if x.isInstanceOf[Double] => true
         case _ => false
       }

scala> hasDoubles3(Vector(1.0))    // funkar!
```

Typparametrar på tentan?

- Det ingår att kunna använda färdiga generiska strukturer med specifik typer, t.ex. `Vector[Int]`
- Det ingår att kunna skapa strukturer med specifika typparametrar, t.ex. en case-klass som tar en vektor med en specifik typ:
  ```
  case class X(x: Vector[Int])
  ```
- Det ingår **inte** på tentan att kunna skapa generiska metoder eller klasser, t.ex.:
  ```
  def f[T](x: Vector[T]): Vector[T] = ???
  ```
  Mer om generiska strukturer fortsättningskursen!