Matematické algoritmy

Pole a matice

Při deklaraci pole nejprve uvedeme typ hodnot, které do něj budeme ukládat, za něj pár hranatých závorek (označení, že se jedná o pole) a název proměnné tohoto pole.

Při inicializaci za rovnítko vepíšeme klíčové slovo **new**, následované typem hodnoty, která bude v poli uložena a v hranatých závorkách uzavřenou celočíselnou hodnotu, jež definuje, jak bude pole velké.

Příklad

```
//pole osmi integeru
int[] array = new int[8];

//pole ctyr retezcu
String[] stringArray = new String[4];
```

Přístup k prvkům pole

Pro přístup k hodnotám uloženým v poli používáme hranaté závorky vepsané za jméno proměnné daného pole. V těchto

závorkách uvádíme index (pořadí počítané od nuly) zvolené hodnoty.

Příklad

```
//pole osmi integeru
int[] array = new int[8];
array[0] = 5; //prvni hodnota je 5
array[1] = 3; //druha je 3
array[2] = array[1]; //do treti priradime
hodnotu na indexu 1, tj. 3

array[1] = 4; //array[1] == 4, array[2] ==
3 (V minulem prikazu jsme pouze priradili
```

```
hodnotu. Nerekli jsme, ze navzdy budou tyto
hodnoty totozne)

//pole ctyr retezcu
String[] stringArray = new String[4];
stringArray[0] = "Skakal";
stringArray[1] = "pes";
stringArray[2] = "pres";
stringArray[3] = "oves";
System.out.println(stringArray[0]);
//skakal
```

Zjednodušená inicializace

Takováto inicializace hodnotami je velmi zdlouhavá a nešikovná. Proto máme při deklaraci s inicializací možnost vytvořit pole alternativním způsobem, kdy na pravou stranu přiřazení napíšeme do složených závorek čárkami oddělené hodnoty, na něž má být pole na jednotlivých indexech inicializováno.

Příklad

```
String[] stringArray2 = {"Skakal", "pes",
"pres", "oves"}; //pole ctyr retezcu
```

Příklad

Kvadratická rovnice může 0 až 2 řešení v oboru reálných čísel v závislosti na tom, jesli je diskriminant záporný, nulový nebo kladný.

```
/**
 * Resi kvadratickou rovnici o jedne
nezname ve tvaru
 * ax^2 + bx + c = 0
 * @param a
 * @param b
 * @param c
 * @return pole realnych korenu
```

```
 */
public static double[]
solveQuadraticEquation(double a, double b,
double c) {
    double d = b*b - 4*a*c; //diskriminant
    if (d < 0) {
        double[] result = new double[0];
        return result;
    } else if (d == 0) {
        double[] result = {-b/2*a};
        return result;
    } else {
        double[] result = {(-b +
Math.sqrt(d))/(2*a), (-b -
Math.sqrt(d))/(2*a)};
```

```
            return result;
        }
}
```

Pole a cyklus *for*

Struktura *for-each* smyčky je podobná *for* cyklu, s tím rozdílem, že v závorkách nalezneme nejprve typ proměnné, která je uložena v kolekci, přes níž iterujeme. Za typem následuje název proměnné, pod níž budou v jednotlivých iteracích zpřístupněny uložené objekty, dvojtečka a název proměnné procházené kolekce.

```
/**
 * Vypise retezce ulozene v danem poli
 * @param array pole
 */
public static void print(String[] array){
    for (String s : array) {
        System.out.println(s);
    }
}
```

Samozřejmě, že jde také použít klasický cyklus *for*:

```
/**
 * Vypise cela cisla ulozena v danem poli
 * @param array pole
 */
public static void print(int[] array) {
    for (int i = 0; i < array.length; i++)
{
        System.out.println(array[i]);
    }
}
```

Matice

Matice je vícedimenzionální pole.

Při deklaraci vícerozměrných polí postupujeme obdobně jako u pole jednorozměrného, jediným rozdílem je počet závorek, které uvádíme (co dvojice závorek – to jeden rozměr). Na pravé straně přiřazení **vždy musíme uvést velikost prvního rozměru** (samotného pole polí). Velikost ostatních rozměrů vyplnit nemusíme, ale Java pak tyto rozměry nezinicializuje (to musíme udělat ručně).

Neuvedení dalších rozměrů se může hodit, pokud chceme vytvořit nějakým způsobem nepravidelnou matici – například ve tvaru trojúhelníku.

Pro vícerozměrná pole platí následující poučka: dvojrozměrné pole je časté, trojrozměrné je neobvyklé a více jak trojrozměrné

je obvykle chyba návrhu aplikace (jenom málokdo si umí taková data představit).

Příklad

```
//pole 3 radky, 4 sloupce
int[][] array2d = new int[3][4];

int[][] array2d2 = new int[3][]; //druhy
rozmer neuvadime
for(int i = 0; i < array2d2.length; i++){
//trojuhelnikova matice
    array2d2[i] = new int[i + 1];
}
```

Příklad

```
/**
 * Vypise matici (dvojrozmerne pole)
 * @param array matice
 */
public static void print(int[][] array) {
    for (int i = 0; i < array.length; i++)
{ //pruchod pres pole poli
        for (int j = 0; j <
array[i].length; j++) { //pruchod samotnym
polem (radkem)
```

```
                System.out.print(array[i][j] +
" "); //bez odradkovani
            }
            System.out.println("");
//odradkovani
        }
}
```

Inicializace se dá zjednodušit také.

```
int[][] setting = {
        {0, 0, 4, 0, 3, 6, 9, 2, 7},
        {1, 0, 0, 0, 0, 5, 0, 0, 0},
        {0, 0, 0, 2, 0, 0, 0, 0, 4},
        {0, 0, 5, 0, 0, 0, 0, 6, 0},
```

```
    {6, 4, 0, 0, 0, 0, 0, 8, 5},
    {0, 7, 0, 0, 0, 0, 2, 0, 0},
    {5, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 0, 7, 0, 0, 0, 0, 2},
    {4, 3, 7, 9, 2, 0, 5, 0, 0}
};
```

Násobení dvou matic

Maticové násobení vlastně je rozšíření obvyklého násobení dvou čísel na případ dvou matic.

Pokud A je matice o rozměrech $m \times n$ a B je matice $n \times p$, pak jejich součin A*B je matice s rozměry $m \times p$, definovaná vztahem

$$(A \cdot B)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}.$$

Příklad

Mějme matice

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Jejich součin je

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} (1 \cdot 1 + 2 \cdot 3 + 3 \cdot 5) & (1 \cdot 2 + 2 \cdot 4 + 3 \cdot 6) \\ (4 \cdot 1 + 5 \cdot 3 + 6 \cdot 5) & (4 \cdot 2 + 5 \cdot 4 + 6 \cdot 6) \end{pmatrix} = \begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$$

```
for(int i = 0; i < matA.length; i++) {
```

```
    for(int j = 0; j < matB[0].length; j++)
{
        int sou = 0;
        for(int s = 0; s<matB.length; s++){
            sou += (matA[i][s]*matB[s][j]);
            }
        vynasmat[i][j] = sou;
        }
    }
```

Při deklaraci pole nejprve uvedeme typ hodnot, které do něj budeme ukládat, za něj pár hranatých závorek (označení, že se jedná o pole) a název proměnné tohoto pole. Při inicializaci za rovnítko vepíšeme klíčové slovo new následované typem hodnoty, která bude v poli uložena a v hranatých závorkách

uzavřenou celočíselnou hodnotu, jež definuje, jak bude pole velké.

Hledání GCD (největší společný dělitel).

GCD dvou čísel je největší kladné celé číslo, které dělí obě čísla plně, tedy bez jakéhokoliv zbytku.

Eukleidův algoritmus je efektivní způsob, jak najít GCD dvou čísel a je ho docela snadné realizovat pomocí rekurze v programu Java.

Příklad

Mějme dvě čísla, 133 a 15, jejich nejvyšší společný dělitel (**gcd**) nalezneme takto:

133 vydělíme 15, zbyde 13:

$$133 = 8*15 + 13$$

Nyní si musíme uvědomit tyto věci. Jak levá, tak pravá strana rovnice musí být dělitelná gcd (protože hledáme gcd 133 a 15 a obě strany jsou rovny 133).

Dalším poznatkem je, že násobek patnácti je nevýznamný, protože gcd je faktorem samotného čísla 15, čili násobek (zde 8) klidně můžeme vynechat.

Posledním a nejdůležitějším poznatkem je, že pokud **gcd** dělí 15 a zároveň je celá pravá strana dělitelná **gcd**, tak i zbytek (v našem případě 13) musí být dělitelný gcd.

Když všechna tato pravidla poskládáme, tak z nich plyne, že dělení se zbytkem můžeme opakovat pro redukovaná čísla. Jelikož se levá strana vždy musí zmenšit a nemůže klesnout na 0, tak algoritmus je konečný.

Další postup je proto analogický s prvním krokem, pouze na nižších číslech. Postupně dostáváme:

15 = 1 * 13 + 2

13 = 6 * 2 + 1

2 = 2 * 1 + 0

Nyní, když nám vyšel zbytek 0, tak již vidíme, které nejvyšší číslo dělí jak levou, tak pravou stranu, je to číslo 1 (protože z předchozího kroku jsme věděli, že námi hledané číslo musí dělit čísla 2 a 1).<EN>When we combine all these rules, follows that dividing with a remainder can be repeated for the reduced numbers. Since the left side must always be smaller and can not drop to 0, then the algorithm is finite.

Another approach is therefore analogous to the first step, only the lower numbers. Gradually, we get:

15 = 1 * 13 + 2

13 = 6 * 2 + 1

2 = 2 * 1 + 0

Now, when we obtained the result 0, we see that the highest number that divides both the left and right side, it is the number 1 (because in the previous step, we found that the wanted number must divide both numbers 2 and 1).

Jiný příklad

Čísla 140 a 15:

140 = 9*15 + 5

15 = 3*5 + 0

Nejvyšším společným dělitelem čísel 140 a 15 je číslo 5.

```
public class GCDExample {


    public static void main(String args[]){

        //Enter two number whose GCD needs
to be calculated.
        Scanner scanner = new
Scanner(System.in);
        System.out.println("Please enter
first number to find GCD");
        int number1 = scanner.nextInt();
```

```
        System.out.println("Please enter
second number to find GCD");
        int number2 = scanner.nextInt();

        System.out.println("GCD of two
numbers " + number1 +" and "
                            + number2 +" is
:" + indGCD(number1,number2));
     }

    /*
     * Java method to find GCD of two
number using Euclid's method
     * @return GDC of two numbers in Java
     */
```

```
    private static int findGCD(int number1,
int number2) {
        //base case
        if(number2 == 0){
            return number1;
        }
        return findGCD(number2,
number1%number2);
    }
}
```

Výsledek

```
Please enter first number to find GCD
54
Please enter second number to find GCD
24
```

```
GCD of two numbers 54 and 24 is :6
```

Největší společný dělitel (GCD)

Nejmenší společný násobek (LCM), Euklidův algoritmus

Faktoriál, Rekurze, Faktorizace

Fibonacciho posloupnost

Fisher-Yatesův algoritmus