Polynome

$$\sum_{i=0}^{n} a_i x^i$$

Gliederung

- 10719 – Quotient Polynomial (Polynomdivison)
- 10586 – Polynomial Remains (Polynomdivision)
- Polynomial GCD (Euklidscher Algorithmus)
- 498 – Polly the Polynomial
- 10268 – 498‘ (Ableitung)
- $\int 498$ (Integration)
- Newtonverfahren für Polynome
- Überraschung!

10719 – Quotient Polynomial

Sei p(x) ein beliebiges Polynom von Grad n. Dann existiert ein k, so dass man p(x) auch als

$$p(x) = (x - k)q(x) + r$$

schreiben kann.

In diesem Problem soll das Ergebnispolynom q(x) und der Rest r berechnet werden.

$$\Rightarrow \frac{p(x)}{(x-k)} = q(x) + \frac{r}{(x-k)}$$

Beispiel:
$p(x) = x^3 - 7x^2 + 15x - 8$ mit $k = 3$

$\Rightarrow$ Berechne $\mathrm{p}(x) : (\mathrm{x} - 3)$

Rechenbeispiel Polynomdivison

$$(x^3 - 7x^2 + 15x - 8) : (x - 3) = x^2 - 4x + 3 + \frac{1}{(x-3)}$$

$$\underline{-(x^3 - 3x^2)}$$

$$-4x^2 + 15x$$

$$\underline{-(-4x^2 + 12x)}$$

$$3x - 8$$

$$\underline{-(3x - 9)}$$

$$1$$

Wichtigstes Codefragmente

```
resultPolynom = new int[coefficients.length];
int adder = 0;

//Horner Schema
for (int i = 0; i < coefficients.length; i++) {
    resultPolynom[i] = coefficients[i] + adder;
    adder = resultPolynom[i] * divisor;
}

//Generiere Output
StringBuilder sb = new StringBuilder("q(x):");
for (int i = 0; i < resultPolynom.length - 1; i++) {
    sb.append(" " + resultPolynom[i]);
}

System.out.println(sb);
System.out.println("r = " + resultPolynom[resultPolynom.length - 1]);
```

10586 – Polynomial Remains

Gegeben sei ein Polynom a(x) mit Grad n und eine Ganzzahl k.

Berechnen Sie das Polynom r(x), welches bei Division von a(x) mit $\left(x^k + 1\right)$ herauskommt.

$$\Rightarrow \frac{a(x)}{(x^k + 1)} = r(x)$$

Wichtigstes Codefragmente

```
if (k <= n && k != 0) {
    resultPolynom = new long[n - k + 1];
    remainderPolynom = new long[n + 1];

    for (int i = 0; i < n - k + 1; i++) {
        if (polynom[i] == 0)
            continue;

        resultPolynom[i] = polynom[i];
        remainderPolynom[i] = resultPolynom[i];
        remainderPolynom[i + k] = resultPolynom[i];

        polynom[i] = polynom[i] - remainderPolynom[i];
        polynom[i + k] = polynom[i + k] - remainderPolynom[i + k];

        remainderPolynom[i] = 0;
        remainderPolynom[i + k] = 0;
    }

    printArrayReverseNotZero(polynom);
} else if(k == 0) {
    System.out.println("0");
} else {
    printArrayReverseNotZero(polynom);
}
```

Polynomial GCD

Gegeben seien zwei Polynome, a(x) und b(x).

Berechnen Sie das Polynom, das den größten gemeinsamen Teiler von a(x) und b(x) dastellt.

⇒Euklidscher Algorithmus

Wichtigste Codefragmente, Main

```
polynomA = readDoubleArr(br.readLine());
polynomB = readDoubleArr(br.readLine());
double[] gcd = gcdOfPolynoms(polynomA, polynomB);

double gcdOfPolyElements = moreEuclid(gcd);
for (int i = 0; i < gcd.length; i++) {
    gcd[i] = gcd[i] / gcdOfPolyElements;
}

StringBuilder b = new StringBuilder("Case " + numberOfCase + ": ");
b.append(gcd.length - 1 + " ");
for (int i = gcd.length - 1; i >= 0; i--)
    b.append((int) gcd[i] + " ");

System.out.println(b.substring(0, b.length() - 1));
```

Wichtigste Codefragmente, Euklid für Polynome

```
do {
    int gradeA = polynomA.length - 1;
    int gradeB = polynomB.length - 1;
    double tmp;
    double[] kor;

    if (gradeA < gradeB) {
        kor = polynomA;
        polynomA = polynomB;
        polynomB = kor;
    } else {
        tmp = polynomA[gradeA] / polynomB[gradeB];
        double[] div = new double[gradeA + 1];
        if (gradeA == gradeB) {
            for (int i = gradeB; i >= 0; i--) {
                div[i] = polynomB[i] * tmp;
            }
        } else {
            int grade = gradeA - gradeB;
            for (int i = gradeB; i >= 0; i--) {
                div[i + grade] = polynomB[i] * tmp;
            }
        }
        double[] sub = new double[gradeA];
        for (int i = 0; i < gradeA; i++) {
            sub[i] = polynomA[i] - div[i];
        }

        if (sub.length == 0) {
            break;
        }

        polynomA = polynomB;
        polynomB = deleteLeadingZero(sub);
    }
} while (polynomB.length != 0);
return polynomA;
```

Wichtige Codefragmente, Euklid allgemein

```
public static double euclid(double a, double b) {
    if (b == 0)
        return a;
    else
        return euclid(b, a % b);
}

public static double moreEuclid(double[] poly) {
    double[] gcd;
    gcd = new double[poly.length - 1];

    for (int i = 0; i < gcd.length; i++) {
        gcd[i] = euclid(poly[i], poly[i + 1]);
    }

    if (gcd.length == 2)
        return euclid(gcd[0], gcd[1]);
    else
        return moreEuclid(poly);
}
```

498 - Polly the Polynomial

Gegeben sei ein Polynom p(x) und eine Zahlenmenge M.

Berechne den Wert des Polynoms p(x) für jedes $x \in M$ und gebe diese Werte aus.

Wichtigstes Codefragment

```
for (long x : xValues) {
    long sum = 0;
    for (int cOffset = 0; cOffset < coefficients.length; cOffset++) {
        sum += coefficients[cOffset]
                * Math.pow(x, coefficients.length - cOffset - 1);
    }

    sb.append(sum + " ");
}

System.out.println(sb.substring(0, sb.length() - 1));
```

10268 – 498‘

Gegeben sei ein Polynom p(x) und eine Zahlenmenge M.

Berechne den Wert des Polynoms p‘(x) für jedes $x \in M$ und gebe diese Werte aus.

Wichtigste Codefragmente

```
public static long[] derivePolynom(long[] arr) {
    if (arr.length > 1) {
        long[] result = new long[arr.length - 1];

        for (int i = 0; i < arr.length - 1; i++) {
            result[i] = arr[i] * (arr.length - i - 1);
        }

        return result;
    } else {
        return new long[] { 0 };
    }
}

public static long hornerScheme(long[] polynom, long valueForX) {
    long adder = 0;
    long result = 0;

    for (int i = 0; i < polynom.length; i++) {
        result = polynom[i] + adder;
        adder = result * valueForX;
    }

    return result;
}
```

$\int 498$

Gegeben sei ein Polynom p(x) und eine Zahlenmenge M.

Berechne $\int p(x)$ für jedes $x \in M$ und gebe diese Werte aus.

Wichtigstes Codefragment

```java
public static long[] integratePolynom(long[] arr, long c) {
    long[] result = new long[arr.length + 1];

    for (int i = 0; i < arr.length; i++) {
        result[i] = arr[i] / (result.length - i);
    }

    result[arr.length] = c;

    return result;
}
```

Newtonverfahren zur Nullstellenbestimmung von Polynomen

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Überraschung!

Integrieren, Ableiten, Einlesen und berechnen von Polynomwerten.

Jetzt downloaden unter:

http://trampi.000-n-000.de/Polynomfunctions.java

(danke für eure Aufmerksamkeit ☺)