Fakultät und Binomialkoeffizient

Ac 2013-2022

1. Die Fakultät (eine natürliche Zahl) :

Die Fakultät n! (sprich: n Fakultät) ist so definiert: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n \ , \ \text{wobei} \ \ 0! = 1$

Die **rekursive** Definition ist:

Falls n = 0
  dann n! = 1
  sonst n! = n·(n-1)!

Die Ergebnisse für n! **wachsen enorm stark**, wesentlich stärker als die der Exponentialfunktionen.
So übertrifft bereits 70! = 1,197857166... · 10^{100} das Fassungsvermögen gewöhnlicher Taschenrechner.

Beispiele:

0! = 1 3! = 6 5! = 120 7! = 5040 15! = 1307674368000 100! ≈ 9,3326215443·10^{157}

1000! ≈ 4,0238726007·10^{2567} 10000! ≈ 2,8462596809·10^{35659}

JAVA-Methoden (iterativ und rekursiv) :

```
public static long nFakLong(int n) {
    long fak = 1L;
    for ( int i = 2; i <= n; i++ )
       fak = fak * i;
    return fak;
}
```

Hinweis: Diese long-Methode arbeitet bis n = 20, da 20! = 2432902008176640000 = 2,432·10^18 .
Bei n = 21 erfolgt ein Überlauf des Long-Zahlbereichs (max. 9,223372036854775807·10^18)

Daher ist eine "Langzahl-Methode" (BigInteger) besser geeignet:

```
public static BigInteger nFakBig(int n) {
    BigInteger fakBig = BigInteger.ONE;
    for ( int i = 2; i <= n; i++ )
       fakBig = fakBig.multiply(BigInteger.valueOf(i));
    return fakBig;
}
```

Schließlich noch die rekursive Methode, die aber **nachteilig** ist wegen eines möglichen „Stack-Overflows“ !

```
public static long nFakRek(int n) {
    if ( n > 1 )
       return nFakRek = n*nFakRek(n-1);
    else
       return 1L;
}
```

Anzahl der Ziffern einer Fakultät

Will man die Anzahl der Ziffern einer Fakultät herausfinden, so gibt es u.a. folgende 2 Möglichkeiten, die auf die Logarithmierung zurückgehen:

1) Aus obiger Definition ergibt sich durch Logarithmieren der beiden Seiten:

$$(1)\ \lg(n!) = \lg(1\cdot 2\cdot 3\cdot \ldots \cdot (n-1)\cdot n) = \lg(1)+\lg(2)+\lg(3)+\ldots+\lg(n-1)+\lg(n)$$

Wegen lg(1) = 0 folgt dann: $\lg(n!) = \sum_{i=2}^{n} \lg(i)$

Aus dem Ergebnis für lg(n!) kann man sofort auf die Anzahl der Stellen von n! schließen, denn es gilt:

$$(*)\ \text{Anzahl der Stellen von n!} = \mathrm{Vk}\left(1+\sum_{i=2}^{n}\lg(i)\right)$$

Anm.: Vk bedeutet "Vorkommaanteil".

2) Stirling-Formel: $(2)\ \ln(n!) \approx n\cdot(\ln(n)-1) + \frac{\ln(2\pi n)}{2} + \frac{1}{12n}$ (modifizierte Stirling-Formel)

Auch hier wird Formel (*) angewandt, nur dass statt der Summe der Term der Stirling-Formel eingesetzt werden muss !
Während Formel (1) exakt gilt, jedoch für große n aufwendig zu berechnen ist, gilt Formel (2) nur näherungsweise, ist aber auch für große n schnell zu berechnen.

Die folgende Tabelle gibt einen Einblick in die Brauchbarkeit der beiden Formeln unter Anwendung von Formel (*) :

n	**Anzahl der Ziffern n!**	**Summenformel 1)**	**mod. Stirling-Formel 2)**
1	1	1	1
10	7	7	7
100	158	158	158
1 000	2568	2568	2568
10 000	35660	35660	35660
100 000	456574	456574	456574
1 000 000	5565709	5565709	5565709
10 000 000	65657060	65657060	65657060

Beide Formeln sind also sehr brauchbar !

Eine Formel für die Anzahl der Endnullen von n! ist: $\text{Anzahl der Endnullen von n!} = \sum_{k=1}^{\mathrm{Vk}(\log_5 n)} \mathrm{Vk}\left(\frac{n}{5^k}\right)$

Beispiel: n=125: Anzahl Endnullen von 125! = [125/5] + [125/25] + [125/125] = 25 + 5 + 1 = **31** .

Prüfung: 125! = 188267717688892609974376770249160085759540364871492425887598231508353156
331613598866882932889495923133646405445930057740630161919341380597818883457558547055524326375565007131770880000000000000000000000000000000 (31 Endnullen !)

Der Binomialkoeffizient :

Der Binomialkoeffizient „n über k“ ist für n ≥ k so definiert:

$$\binom{n}{k}=\frac{n!}{k!\cdot(n-k)!} \quad \text{mit} \quad \binom{n}{n-k}=\binom{n}{k} \quad \text{und} \quad \binom{n}{0}=\binom{n}{n}=1 \ .$$

Durch Kürzen von n! gegen (n-k)! lässt sich die Formel noch vereinfachen:

Wegen $\frac{n!}{(n-k)!}=n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-(k-2))\cdot(n-(k-1))$ folgt:

$$\binom{n}{k}=\frac{n!}{k!\cdot(n-k)!}=\frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-(k-2))\cdot(n-(k-1))}{1\cdot2\cdot\ldots\cdot(k-2)\cdot(k-1)\cdot k}$$

$$=\frac{n}{1}\cdot\frac{n-1}{2}\cdot\frac{n-2}{3}\cdot\ldots\cdot\frac{n-(k-2)}{k-1}\cdot\frac{n-(k-1)}{k} \rightarrow \text{Umorientierung des Zählers}$$

$$=\frac{n-k+1}{1}\cdot\frac{n-k+2}{2}\cdot\frac{n-k+3}{3}\cdot\ldots\cdot\frac{n-k+(k-1)}{k-1}\cdot\frac{n-k+k}{k}=\prod_{i=1}^{k}\frac{n-k+i}{i}$$

Der Vorteil ist, dass hier alle Zwischenergebnisse **natürliche Zahlen** sind und die Rechnung schnell ist.

Rekursiv gilt: $\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}=\frac{n-(k-1)}{k}\cdot\binom{n}{k-1},\ k>0\ ;\ \binom{n}{0}=1$

Für wachsende n und k steigen die Ergebnisse stark an !
Das maximal Mögliche im (positiven) Long-Zahlenbereich ist "60 über 30" (s. unten) !

Beispiele:

$$\binom{5}{2}=\frac{5}{1}\cdot\frac{4}{2}=10$$

$$\binom{11}{4}=\frac{11}{1}\cdot\frac{10}{2}\cdot\frac{9}{3}\cdot\frac{8}{4}=330$$

$$\binom{18}{13}=\binom{18}{18-13}=\binom{18}{5}=\frac{18}{1}\cdot\frac{17}{2}\cdot\frac{16}{3}\cdot\frac{15}{4}\cdot\frac{14}{5}=8568$$

$$\binom{60}{30}=\frac{60}{1}\cdot\frac{59}{2}\cdot\frac{58}{3}\cdot\frac{57}{4}\cdot\frac{56}{5}\cdot\ldots\cdot\frac{32}{29}\cdot\frac{31}{30}=118264581564861424\approx1{,}18\cdot10^{17}$$

$$\binom{61}{30}=\frac{61}{1}\cdot\frac{60}{2}\cdot\frac{59}{3}\cdot\frac{58}{4}\cdot\frac{57}{5}\cdot\ldots\cdot\frac{33}{29}\cdot\frac{32}{30}=232714176627630544\approx2{,}3\cdot10^{17}$$

Während der Rechnung wird bereits der Long-Zahlenbereich (max. $9223372036854775807\approx9{,}2\cdot10^{18}$) überschritten, und zwar bei der Multiplikation mit 32 !

JAVA-Methoden (iterativ) :

```
public static long nUeberK(int n,int k) {
   if (n < k) return 0L;
   if (n < 2*k) k = n-k;
   if (k == 1) return n;
   if (k == 0) return 1;
   long nminusk = n - k;
   long bin = nminusk + 1;
   for (int i=2; i<=k; i++)
      bin = bin*(nminusk+i)/i;
   return bin;
}

public static BigInteger nUeberKBig(int n, int k) {
   if (n < k) return BigInteger.ZERO;
   if (n < 2*k) k = n - k;
   if (k == 1) return new BigInteger(String.valueOf(n));
   if (k == 0) return BigInteger.ONE;
   long nminusk = n - k;
   BigInteger binBig = BigInteger.valueOf(nminusk + 1);
   for (int i=2; i<=k; i++)
      binBig = binBig.multiply(BigInteger.valueOf(nminusk +i)).
               divide(BigInteger.valueOf(i));
   return binBig;
}
```

JAVA-Methode (rekursiv) :

```
public static long nUeberKRek(int n,int k) {
   if (n<k) return 0;
   if (n<2k) k = n-k;
   if (k=1) return n;
   if (k=0) return 1;
      else return nUeberKRek(n-1,k-1) + nUeberKRek(n-1,k);
}
```

Approximative Berechnungen (STIRLING-Formel) :

Nicht immer sind natürliche Zahlen zur Berechnung von Fakultät oder Binomialkoeffizient geeignet. Selbst der JAVA-Datentyp BigInteger tut sich bei sehr großen Zahlen mitunter schwer . Z.B. dauert die Berechnung von 1 000 000 ! auf derzeit üblichen PCs mitunter einige Minuten !!

Als Ausweg können **Approximationen** mithilfe der sog. **Stirling-Formel** (James Stirling) dienen:

Stirling-Reihenentwicklung für n! :

$$\ln(n!) = n\cdot(\ln(n)-1)+\frac{\ln(2\pi n)}{2}+\frac{1}{12n}-\frac{1}{360n^3}+\frac{1}{1260n^5}-\frac{1}{1680n^7}+\ldots$$

Je genauer die Approximation mit der Stirling-Reihenentwicklung sein soll, umso mehr Reihenglieder werden benötigt und müssen ermittelt werden!

Bricht man die STIRLING-Reihe nach dem zweiten Summanden ab, so ist der absolute Fehler < 1/(12n).
Für n >1000 genügt der erste Summand, um den relativen Fehler kleiner als 1% zu halten !
Für sehr große n wird der relative Fehler noch kleiner.

Wie erhält man n! , wenn n sehr groß ist ?

Zuerst muss eine Approximation für z = ln(n!) gemäß der Stirling-Reihe berechnet werden.
Wir beschränken uns dabei auf die Glieder der Reihe bis einschließlich 1/(12n) .

Beispiel: $\ln(1\,000\,000!) \approx 1000000 \cdot (\ln(1000000) - 1) + \ln(2\pi \cdot 1000000)/2 + 1/(12 \cdot 1000000)$
$\approx 12815518{,}384658$

Der "normale Weg" wäre dann die Bildung von e^z, was n! liefern würde.

Leider aber ist es kaum möglich, e^ln(n!) für **sehr große n** in angemessener Zeit zu berechnen bzw. mit dem normalen Datentyp “double” darzustellen, weil diese Zahl für normale Rechner zu groß ist.
Für das Beispiel 1 000 000 ! ergäbe sich ca. $8{,}26393112 \cdot 10^{5565708}$.
Der Datentyp “double” endet bei ca. 10^{308} .

Man muss daher eine Umformung vornehmen, um von ln(n!) auf das gesuchte n! zu schließen .
Man rechnet daher zunächst um auf die Basis 10: **$\lg(n!) = \ln(n!) / \ln(10)$**

Für obiges Beispiel ergibt sich: **$\lg(1000000!) \approx 12815518{,}384658 / \ln(10) \approx 5565708{,}91718664$**

Jetzt hat man eine neue Dezimalzahl, deren Vorkommateil (Vk) den Exponenten der Zehnerpotenz von n! anzeigt. Der Exponent ist also expo := Vk(lg(n!)). Man notiert dann einfach den String 10^expo .
Für obiges Beispiel notiert man also **10^5565708** .

Die Mantisse steckt im Nachkommateil (Nk) von lg(n!), und zwar **berechnet** man :
mantisse = 10^Nk(lg(n!)) . Für obiges Beispiel notiert man $10^{0{,}91718664} \approx$ **8,2639296**.

Der gesuchte Gesamtstring setzt sich dann zusammen aus: “mantisse” und ”·10^expo” .
Das Ergebnis ist dann **1000000! ≈ 8,26393·10^5565708** .

Fazit: Approximationsformel für n! nach ausgerechnetem ln(n!) :

Erst $\boxed{\lg(n!) = \ln(n!)/\ln(10)}$ berechnen, dann $\boxed{n! \approx 10^{\mathrm{Nk}(\lg(n!))} \cdot 10^{\mathrm{Vk}(\lg(n!))}}$

beim 1.Faktor muss die gesamte Potenz ausgerechnet werden, beim 2. nur der Exponent ! Beispiele (Anwendung der obigen Formeln) :

ln(56!) ≈ 56·(ln(56)-1) + ln(112π)/2 + 1/672 – 1/63221760 ≈ 172,3527971 (Stirling-Reihenentwicklung)
Also ist lg(56!) = 172,3527971/ln(10) ≈ 74,85186872 und wegen $10^{0{,}85186872} \approx 7{,}109986$ gilt
dann: **$56! \approx 7{,}109986 \cdot 10^{74}$**

$\ln(500!) \approx 500 \cdot (\ln(500)-1) + \ln(1000\pi)/2 + 1/6000 - 1/4{,}5 * 10^{10} \approx 2611{,}330458$
Also ist **$500! \approx 1{,}2201368 \cdot 10^{1134}$**

$\ln(12000!) \approx 12000 \cdot (\ln(12000)-1) + \ln(24000\pi)/2 + 1/6{,}2208 * 10^{14} - 1/3{,}1352832 * 10^{23} \approx 100717{,}5584$
Also ist **$12000! \approx 1{,}2018584 \cdot 10^{43741}$**

Ebenso
$50000! \approx 3{,}3473205 \cdot 10^{213236}$

Stirling-Reihenentwicklung für “n über k” :

Unter Verwendung der Stirling-Formel für n! (siehe oben)

$$\ln(n!) = n\cdot(\ln(n)-1)+\frac{\ln(2\pi n)}{2}+\frac{1}{12n}-\frac{1}{360n^3}+\frac{1}{1260n^5}-\frac{1}{1680n^7}+\ldots$$

kann man auch eine Approximationsformel für den Binomialkoeffizienten “n über k” herleiten.

Zunächst eine einfache Umformung für ln(n!) :

$$\ln\binom{n}{k} = \ln\frac{n!}{k!(n-k)!} = \ln(n!) - \ln(k!) - \ln((n-k)!) \text{ Logarithmengesetze !}$$

Nimmt man obige Reihe bis zum Glied 1/(12n), so ergibt sich

$$\ln\binom{n}{k} = \ln(n!) - \ln(k!) - \ln((n-k)!) \quad \text{mit}$$

$$\ln(n!) \approx n\cdot(\ln(n)-1)+\frac{\ln(2\pi n)}{2}+\frac{1}{12n}$$

$$\ln(k!) \approx k\cdot(\ln(k)-1)+\frac{\ln(2\pi k)}{2}+\frac{1}{12k}$$

$$\ln((n-k)!) \approx (n-k)\cdot(\ln(n-k)-1)+\frac{\ln(2\pi(n-k))}{2}+\frac{1}{12(n-k)}$$

Außerdem ist $\lg\binom{n}{k} = \frac{\ln\binom{n}{k}}{\ln(10)}$

und es gilt schließlich: $\boxed{\binom{n}{k} \approx 10^{Nk(\lg\binom{n}{k})}\cdot 10^{Vk(\lg\binom{n}{k})}}$

beim 1.Faktor muss die gesamte Potenz ausgerechnet werden, beim 2.Faktor nur der Exponent !

Beispiele :

1. Approximation für $\binom{300}{150}$

$$\ln\binom{300}{150} = \ln(300!) - \ln(150!) - \ln(150!) = \ln(300!) - 2\cdot\ln(150!) \quad \text{mit}$$

$$\ln(300!) \approx 300\cdot(\ln(300)-1) + \frac{\ln(2\pi 300)}{2} + \frac{1}{12\cdot 300} \approx 1414,90585$$

$$\ln(150!) \approx 150\cdot(\ln(150)-1) + \frac{\ln(2\pi\cdot 150)}{2} + \frac{1}{12\cdot 150} \approx 605,0201059$$

Also $\ln\binom{300}{150} \approx 204,8656382$

Dann ist $\lg\binom{300}{150} = \frac{\ln\binom{300}{150}}{\ln(10)} \approx \frac{204,8656382}{\ln(10)} \approx 88,97201622$

Schließlich: $\binom{300}{150} = 10^{\mathrm{Nk}(\lg\binom{n}{k})} \cdot 10^{\mathrm{Vk}(\lg\binom{n}{k})} \approx 10^{0,97201622}\cdot 10^{88} \approx 9,375970263\cdot 10^{88}$

Eine genauere Berechnung (iterativ) ergibt den Wert $9,3759702772\cdot 10^{88}$

2. Approximation für $\binom{1000000}{500000}$

$\binom{1000000}{500000} \approx 7,89957877184\cdot 10^{301026}$ (ohne Rechnung)

Genauere Berechnung (iterativ) : $7,8995787722769708417702 \cdot 10^{301026}$