Das Sieb des Eratosthenes

(Eratosthenes von Kyrene ca. 276-194 v.Chr.)

Autor: Jochen Ziegenbalg
Email: ziegenbalg.edu@gmail.com
Internet: https://jochen-ziegenbalg.github.io/root/

Literaturhinweis

J. Ziegenbalg, Algorithmen von Hammurapi bis Gödel, 4. Auflage, Springer Spektrum Wiesbaden 2016

Interaktive Simulation: https://jochen-ziegenbalg.github.io/root/Simulationen

Einführung

Jede natürliche Zahl n besitzt die „trivialen" Teiler 1 und n selbst; jede von 1 verschiedene natürliche Zahl besitzt also mindestens zwei Teiler. Zahlen, die genau diese beiden trivialen Teiler besitzen, nennt man *Primzahlen*. Nach dieser Definition wird die Zahl 1 also nicht zu den Primzahlen gerechnet. Dies hat gute Gründe; einer davon ist, daß sonst der Fundamentalsatz der Zahlentheorie nicht gelten würde. Die Primzahlen sind einer der ältesten und interessantesten Untersuchungsgegenstände der Mathematik. Sie stellen u.a. die Bausteine dar, aus denen die natürlichen Zahlen (multiplikativ) aufgebaut sind.

Der **Fundamentalsatz der Zahlentheorie** besagt:

> Jede natürliche Zahl n $(n > 1)$ ist als Produkt von Primzahlen darstellbar: $n = p_1 \cdot p_2 \cdot \ldots \cdot p_s$. Abgesehen von der Reihenfolge der Faktoren ist diese Darstellung eindeutig.

Auch für andere Zahlensysteme oder algebraische Systeme sind Primzahlen, Primelemente oder dem Primzahlbegriff nachgebildete Begriffe von zentraler Bedeutung. Eine faszinierende Eigenschaft der Primzahlen ist die Unregelmäßigkeit, mit der sie in der Zahlenreihe auftreten. Gesetzmäßigkeiten in der Primzahlreihe zu entdecken, war schon immer eine wichtige Forschungsrichtung in der Mathematik.

Schon im Altertum war man bestrebt, einen möglichst guten Überblick über die Primzahlen zu gewinnen. Euklid zeigte, daß es unendlich viele Primzahlen gibt. Der griechische Mathematiker *Eratosthenes von Kyrene* gab das folgende Verfahren an, um alle Primzahlen bis zu einer bestimmten vorgegebenen Zahl n zu bestimmen.

Es sei hier am Beispiel $n = 20$ erläutert.

1. Schreibe alle Zahlen von 1 bis 20 auf:
2. Streiche die Zahl 1 (sie wird aus guten Gründen nicht zu den Primzahlen gerechnet):
3. Unterstreiche die Zahl 2:
4. Streiche alle echten Vielfachen von 2; also die Zahlen 4, 6, 8, 10, 12, 14, 16, 18 und 20.
5. Unterstreiche die erste freie (d.h. noch nicht unterstrichene oder gestrichene) Zahl; in diesem Fall also die Zahl 3.
6. Streiche aus den verbleibenden Zahlen alle echten Vielfachen von 3; also die Zahlen 9 und 15.
7. Unterstreiche die kleinste freie Zahl; in diesem Fall also die Zahl 5.
8. Streiche aus den verbleibenden Zahlen alle echten Vielfachen der Zahl 5.
   Da die in Frage kommenden Zahlen 10, 15 und 20 bereits gestrichen sind, tritt in diesem Fall (Maximum=20) keine Veränderung auf.
9. Setze das Verfahren sinngemäß so lange fort, bis jede der Zahlen entweder unterstrichen oder gestrichen ist.
10. Ende des Verfahrens. Die unterstrichenen Zahlen sind die Primzahlen zwischen 1 und 20.

Durch dieses Verfahren werden, wenn man so will, also genau die Primzahlen „ausgesiebt“. Man nennt das Verfahren deshalb auch das *Sieb des Eratosthenes* bzw. kurz das *Siebverfahren* (englisch: *sieve*).

Aufgaben:

(a) Führen Sie das Siebverfahren von Hand für die natürlichen Zahlen von 1 bis 200 durch.

(b) Geben Sie eine allgemeine Beschreibung des Siebverfahrens, die von der Zahl 20 unabhängig ist; die Obergrenze sei allgemein mit a bezeichnet.

(c) Zeigen Sie: Ist die Zahl a zerlegbar, z.B. mit von 1 verschiedenen Faktoren x und y, so ist einer der Faktoren kleiner oder gleich $\sqrt{a}$.

(d) Aufgabenteil (c) hat zur Folge, daß das Siebverfahren *erheblich* verkürzt werden kann, denn man ist mit dem Streichen der Vielfachen schon fertig, wenn die unterstrichene Zahl erreicht hat. Formulieren Sie den Algorithmus so, daß diese Verbesserung der „Laufzeiteffizienz“ realisiert wird.

Bemerkungen:

(1.) Gelegentlich kann man lesen, daß das Sieb des Eratosthenes dazu dient, *die* Primzahlen (d.h. *alle* Primzahlen) zu ermitteln. Ein Blick auf den Algorithmus genügt aber, um festzustellen, daß er nur dann funktionieren kann, wenn man sich von vornherein auf einen *endlichen* Zahlenabschnitt beschränkt (im Beispiel: die natürlichen Zahlen von 1 bis 20). Das Sieb des Eratosthenes liefert also stets nur die Primzahlen bis zu einer bestimmten, von vornherein festzulegenden oberen Grenze. Diese Grenze läßt sich jedoch durch mehrere „Läufe“ des Verfahrens immer weiter nach oben verschieben. Die (unendliche) Menge der Primzahlen wird durch das Sieb des Eratosthenes also als „potentiell“ unendliche Menge erschlossen. Es sei an dieser Stelle nochmals an die weise Formulierung von Euklid erinnert: *Es gibt mehr Primzahlen als jede vorgelegte Anzahl von Primzahlen*.

(2.) Obwohl die eingangs geschilderte Version durchaus noch einige Beschleunigungsmöglichkeiten zuläßt, ist das Siebverfahren ein sehr langsamer Algorithmus. Gerade aufgrund seiner Langsamkeit wurde er lange Zeit benutzt, um die Geschwindigkeit von Computern zu messen, denn schnelle Algorithmen laufen u.U. so schnell, daß man eine Zeitmessung nicht sinnvoll vornehmen kann. Die renommierte amerikanische Computerzeitschrift BYTE benutzte jahrelang ein auf dem Sieb des Eratosthenes basierendes Verfahren in diesem Sinne für „benchmark tests“ (Geschwindigkeitstests für Hard- und Software). Das Sieb des Eratosthenes ist also im zweifachen Sinne klassisch zu nennen: im ursprünglichen Sinne von Eratosthenes zur Ermittlung von Primzahlen und neuerdings auch noch als benchmark test. Das von BYTE verwendete Programm ist ohne weiteres in verschiedene Programmiersprachen zu übertragen; es war jedoch im Hinblick auf Ergebnis und Ausgabemeldung fehlerhaft. Das Programm war somit zu

überhaupt nichts nutze - außer als Maß für die Laufzeitgeschwindigkeit verschiedener Computersysteme. Das unten dargestellte Programm lehnt sich so gut es geht an die BYTE-Version an, ohne allerdings dessen Fehler zu übernehmen.

Bei der Obergrenze 1000 läßt sich auf modernen Computern die benötigte Zeit kaum messen. Das Ausdrucken der Primzahlen benötigt oft mehr Zeit als ihre Ermittlung. Man kann die Obergrenze erhöhen, indem man einen anderen Wert für die `Obergrenze` einsetzt. Allerdings wird bei sehr großen Werten von `Obergrenze` die Speicherkapazität des Computers überschritten.

Als benchmark hat das Siebverfahren den Nachteil, daß mit ihm nur wenige Systemkomponenten eines Computers getestet werden - im wesentlichen nur die arithmetisch-logische Einheit des Prozessors (englisch „arithmetic-logic unit" ALU). Inzwischen wurden neuere Testverfahren entwickelt, mit denen auch andere Komponenten (wie Ein- und Ausgabeoperationen, Zugriffsgeschwindigkeit auf diverse Speicherbereiche - und vieles mehr) getestet werden können.

Das Sieb-Programm

```
SiebDesEratosthenes[Obergrenze_] :=
 Module[{L = Table[t, {t, 1, Obergrenze}], i = 2, k},
  L = ReplacePart[L, 0, 1];
  While[i * i ≤ Obergrenze,
   k = i + i;
   While[k ≤ Obergrenze,
    L = ReplacePart[L, 0, k];
    k = k + i];
   i = i + 1];
  Return[Select[L, Positive] ] ]
```

Kommentar aus dem Help-Browser von *Mathematica*:
`ReplacePart[`*expr*`,` *new*`,` *n*`]` yields an expression in which the n^{th} part of *expr* is replaced by *new*.

```
SiebDesEratosthenes[1000]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263,
 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457,
 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,
 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769,
 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881,
 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997}
```

Das Programm in der Syntax von Maxima

```
Eratosthenes(UpperLimit) :=
 block([E, i, k],
   E : makelist(j, j, 1, UpperLimit),
   E[1] : 0,
   i : 2,
   while (i*i <= UpperLimit) do
    (k : i+i,
     while k <= UpperLimit do
       (E[k] : 0,
        k : k+i),
     i : i+1),
   E : delete(0, E),
   E);
```

Hilfsprogramme