Zeitschrift: Elemente der Mathematik
Herausgeber: Schweizerische Mathematische Gesellschaft
Band: 33 (1978)
Heft: 2

Artikel: Algorithmische Kombinatorik mit Kleinrechnern
Autor: Specker, E.
DOI: https://doi.org/10.5169/seals-32936

ELEMENTE DER MATHEMATIK

Revue de mathématiques élémentaires - Rivista di matematica elementare

Zeitschrift zur Pflege der Mathematik
und zur Förderung des mathematisch-physikalischen Unterrichts

El. Math. | Band 33 | Heft 2 | Seiten 25-56 | 10. März 1978

Algorithmische Kombinatorik mit Kleinrechnern[1])

Einleitung

Am Beispiel der Klasseneinteilungen endlicher Mengen werden die drei Grundprobleme der Kombinatorik erläutert und für den betrachteten Fall durch Angabe von Algorithmen «gelöst». Diese Algorithmen sind auf Kleinrechnern programmierbar, und es werden entsprechende Programme für den HP-25 angegeben.
Eine Klasseneinteilung der Menge E ist eine Menge K von nichtleeren Teilmengen von E mit der Eigenschaft, dass es zu jedem a mit $a \in E$ genau ein A gibt mit $a \in A$ und $A \in K$. Es ist also etwa $\{\{1, 3\}, \{2\}\}$ eine Klasseneinteilung von $\{1, 2, 3\}$. Bezeichnen wir sie kürzer mit «(13/2)», so sieht eine Liste aller Klasseneinteilungen der Menge $\{1, 2, 3\}$ folgendermassen aus:

(1/2/3) (12/3) (13/2) (1/23) (123).

Für die vorgesehene algorithmische Behandlung eignet sich diese Darstellung nicht. In Anlehnung an die in der Poëtik übliche Bezeichnung der Reimschemen wollen wir vielmehr die Klasseneinteilungen der Menge $\{1, 2, \ldots, n\}$ (– die im folgenden mit «E_n» bezeichnet sei) durch Abbildungen von E_n in E_n repräsentieren. Und zwar wird der Klasseneinteilung K von E_n die folgende (rekursiv definierte) Funktion f_K zugeordnet:

$$f_K(1) = 1;$$

$f_K(j)$ ist für $2 \leqslant j \leqslant n$ auf Grund einer Fallunterscheidung definiert:

1. Gibt es ein i mit $i < j$, so dass i, j zur selben Menge von K gehören, so sei i_0 das kleinste solche i und

$$f_K(j) = f_K(i_0).$$

1) Ausarbeitung eines Vortrages, gehalten im Rahmen des mathematikdidaktischen Seminars an der ETH Zürich, WS 1976/77.

2. Gibt es kein i mit $i<j$, so dass i,j zur selben Menge von K gehören, so sei

$$f_K(j)=\operatorname{Max}_{i<j} f_K(i)+1\,.$$

Werden die Abbildungen von E_n in E_n in der üblichen Weise durch Folgen dargestellt, so sind den 5 Klasseneinteilungen von E_3 die folgenden Folgen zugeordnet:

$$\langle 1,2,3\rangle\,; \quad \langle 1,1,2\rangle\,; \quad \langle 1,2,1\rangle\,; \quad \langle 1,2,2\rangle\,; \quad \langle 1,1,1\rangle\,.$$

(Die Anordnung ist beibehalten.)

Aus der Folge f_K kann die Klasseneinteilung als Menge der «Niveauflächen» zurückgewonnen werden. Unter allen Folgen f (mit positiven Werten), deren Menge von Niveauflächen gleich K ist, steht die Folge f_K in der alphabetischen Ordnung an erster Stelle.
Folgen f_K werden «Reimschemen» genannt. Reimschemen sind also etwa

$$\langle 1,2,1,3,1,4\rangle\,, \quad \langle 1\rangle\,, \quad \langle 1,1,2,1,2\rangle\,.$$

Die Folge $\langle 2,1,4,1\rangle$ ist kein Reimschema; das Reimschema der Klasseneinteilung von E_4, welche von dieser Folge induziert wird, ist $\langle 1,2,3,2\rangle$.
Klasseneinteilungen werden im folgenden durch Reimschemen dargestellt, und die drei Grundprobleme der Kombinatorik sind auf diese Darstellung bezogen.
Das erste Problem ist das Problem ERKENNEN.
Die Lösung dieses Problems besteht in der Angabe eines Algorithmus, der die Eingabe einer Folge f mit JA oder NEIN beantwortet, je nachdem ob f ein Reimschema ist oder nicht.
Das zweite Problem ist das Problem ERZEUGEN.
Die Lösung besteht in der Angabe eines Algorithmus, der die Eingabe einer natürlichen Zahl n mit der Aufzählung sämtlicher Reimschemen der Länge n beantwortet.
Das dritte Problem ist das Problem ZÄHLEN.
Die Lösung besteht in der Angabe eines Algorithmus, der die Eingabe einer natürlichen Zahl n mit der Ausgabe der Anzahl der Reimschemen der Länge n beantwortet. (Diese Anzahl von Reimschemen oder verschiedenen Klasseneinteilungen einer n-zahligen Menge wird «n-te Bellsche Zahl» genannt. Es gibt darüber eine schon recht umfangreiche Literatur, welche in [2, 3] zusammengestellt ist.)
Für die genauere Behandlung der beiden ersten Probleme ist noch anzugeben, in welcher Form Folgen dargestellt werden. Da der den Programmen zugrunde gelegte Rechner HP-25 nur 8 Speicher besitzt, erscheint es am günstigsten, sich auf Folgen der Zahlen $1,2,\ldots,9$ zu beschränken und solche Folgen durch Zahlen mit der entsprechenden Dezimalziffernfolge darzustellen - die Folge $\langle 1,2,1,3\rangle$ also durch die Zahl 1213. Es ergibt sich daraus allerdings eine Beschränkung für die Länge n der Reimschemen, und zwar $n\leqslant 10$ für das erste und $n\leqslant 9$ für das zweite Problem.

Ein Vorteil der Darstellung von Folgen durch Dezimalziffern liegt darin, dass das letzte Glied und das Anfangsstück einer Folge durch eine Befehlsfolge berechnet werden können, welche unabhängig ist von der Länge der Folge - dass also in einem gewissen Sinn indirekt adressiert werden kann. Ist etwa $\langle f_1,\ldots,f_n\rangle$ in R_0 gespeichert, so speichert die folgende Befehlsfolge $\langle f_1,\ldots,f_{n-1}\rangle$ in R_0 und f_n ins X-Register:

(1) 1 (2) 0 (3) STO ÷ (4) RCL 0 (5) g FRAC
(6) STO − 0 (7) *

Abgesehen von dieser speziellen Behandlung der Folgen unterscheiden sich die Algorithmen für die beiden ersten Probleme kaum von den Algorithmen, die man für grössere Rechner aufstellen würde.
Wesentlich anders verhält es sich für den Algorithmus, der die Anzahl B_n (n-te Bellsche Zahl) der Reimschemen der Länge n bestimmt. Der natürliche Algorithmus für die Berechnung von B_n benützt nämlich eine Speicherzahl, die linear mit n wächst, und eignet sich somit nicht für den HP-25. Er wird deshalb ersetzt durch einen Algorithmus, der mit beschränkter Speicherzahl auskommt und so - wenigstens theoretisch - die Berechnung von B_n für $1\leqslant n\leqslant 33$ auf dem HP-25 erlaubt. Da aber die Einsparung an Speicherplatz durch eine Rechenzeit erkauft wird, welche exponentiell mit n wächst, so ist schon einige Geduld nötig: Die Berechnung von B_{16} dauert etwa 30 Tage, jene von B_{33} schon etwa 30000 Jahre.
Der hier beschriebene Algorithmus für die Bellsche Funktion lässt sich ohne weiteres übertragen auf beliebige Funktionen F, für welche eine Rekursionsformel gilt von der Gestalt

$$F(n)=\sum_{k=0}^{n-1}F(k)q_{kn},$$

mit Koeffizienten q_{kn}, welche in Abhängigkeit von k,n leicht zu berechnen sind. P. Henrici hat die Methode in [1] auf Algorithmen übertragen, welche Koeffizienten von Potenzreihen berechnen.
Es ist vielleicht nicht ganz überflüssig, darauf hinzuweisen, dass es sich bei den besprochenen Grundproblemen der Kombinatorik um «echte Probleme» handelt. Dies in dem Sinne, dass es viele Lösungen gibt und dass die Kriterien, nach welchen diese Lösungen zu vergleichen sind, nicht in der Problemstellung selbst enthalten sind. Von Extremfällen abgesehen können Algorithmen erst dann verglichen werden, wenn bekannt ist, für welche Art Rechner sie bestimmt sind. Ferner ist zu fragen, ob es genügt, dass der Programmierer allein sein Programm versteht - und wenn ja, ob nur heute oder auch im nächsten Jahr.
Dem aufmerksamen Leser dürfte bei den ersten beiden Programmen nicht entgehen, dass versucht worden ist, diesen Gesichtspunkt zu berücksichtigen. Beim dritten Programm war dies kaum möglich, die Anzahl der zur Verfügung stehenden Programmschritte ist dafür wohl allzu knapp.

1. Erkennen

Der Algorithmus ERKENNEN akzeptiert eine Eingabe x genau dann, wenn x (gelesen als Folge der Ziffern) ein Reimschema ist. Es werden also akzeptiert 1, 11, 12, 121314; nicht akzeptiert werden 2, 132, 141312 sowie alle x, die keine natürlichen Zahlen sind oder die Ziffer 0 enthalten.
Nach der Definition der einer Klasseneinteilung K zugeordneten Funktion f_K (gemäss welcher der Funktionswert an der Stelle j entweder gleich dem Wert an einer früheren Stelle ist oder aber gleich der kleinsten natürlichen Zahl verschieden von $f(1), \ldots, f(j-1)$) ist klar, dass f genau dann ein Reimschema ist, wenn gilt

$$f(1)=1; \qquad f(j+1)\leqslant \mathrm{Max}\,(f(1),\ldots,f(j))+1 \qquad (j=1,2,\ldots).$$

Es ist leicht möglich, diese Bedingungen algorithmisch nachzuprüfen.

Soll der Algorithmus ERKENNEN aber für den HP-25 programmiert werden und gleichzeitig leicht erklärbar sein, so empfiehlt sich ein etwas anderer Ansatz. Wir fragen dazu zunächst nach der Bedingung dafür, dass Folgen wie

$$g^j=\langle 1, g_2, \ldots, g_{m-1}, 3, j\rangle \qquad (j=1,2,\ldots)$$

Reimschemen seien.
Für $j:=1,2,3,4$ gilt offenbar: g^j ist ein Reimschema, genau wenn $\langle 1, g_2, \ldots, g_{m-1}, 3\rangle$ ein Reimschema ist.
Für $j:=5,6,7,\ldots$ dagegen gilt: g^j ist ein Reimschema, genau wenn $\langle 1, g_2, \ldots, g_{m-1}, j\rangle$ ein Reimschema ist.
Entsprechend gilt allgemein: $\langle 1, f_2, \ldots, f_{m-1}, f_m, f_{m+1}\rangle$ ist ein Reimschema, genau wenn

$$\langle 1, f_2, \ldots, f_{m-1}, f_m^*\rangle$$

ein Reimschema ist, wobei $f_m^*=f_m$ oder $f_m^*=f_{m+1}$, je nachdem ob $f_{m+1}\leqslant f_m+1$ oder $f_m+1<f_{m+1}$.
Ferner ist $\langle f_1\rangle$ genau dann ein Reimschema, wenn $f_1=1$.
Diese Bedingungen ergeben einen durchsichtigen Algorithmus. Es wird dazu für natürliche Zahlen x mit $10\leqslant x$ eine Funktion t definiert, so dass gilt: Besitzt x die Dezimalziffernfolge $\langle f_1, \ldots, f_{m-1}, f_m, f_{m+1}\rangle$, so besitzt $t(x)$ die Dezimalziffernfolge $\langle f_1, \ldots, f_{m-1}, f_m^*\rangle$ (wobei f_m^* aus f_m, f_{m+1} gemäss der obigen Definition zu berechnen sind). Der Algorithmus verläuft nun so, dass zu x die Werte $t(x)$, $t^2(x)$, ... berechnet werden, bis ein k mit $t^k(x)<10$ erreicht ist; dafür wird dann $t^k(x)=1$ nachgeprüft.

Das Programm ERKENNEN realisiert diesen Algorithmus für den HP-25.
Es ist im Programmschritt 00 ein Wert x einzugeben und mit R/S zu beginnen. Für die Eingabe 121314 werden dann kurz die sukzessiven t-Werte angezeigt:

12134, 1213, 123, 12, 1;

der Rechner hält im Schritt 00 mit Ausgabe 121314, was AKZEPTIEREN bedeuten soll.
Die Eingabe 141312 ergibt:

14131, 1413, 143, 14, 4; ERROR,

was ABLEHNUNG bedeutet. (Das Programm ist so konzipiert, dass es auch vom Schritt 45, in dem es bei ERROR anhält, mit einer neuen Eingabe durch R/S aufgerufen werden kann.)
Enthält die Eingabe x die Ziffer 0, so wird x ebenfalls verworfen. Da bei der Berechnung von $t(x)$ nur die zwei letzten Stellen von x isoliert werden, so geschieht dies, sobald eine der beiden letzten Ziffern von $t^k(x)$ gleich 0 ist. Auf die Eingabe 1210314 folgen somit die Ausgaben 121034, 12103, ERROR.
Ist x negativ oder nicht ganz, so wird es ebenfalls verworfen; dies wird gleich zu Anfang geprüft und ERROR erscheint ohne andere Ausgaben.

2. Erzeugen

Der zweite Algorithmus erzeugt bei Eingabe einer natürlichen Zahl n, $1 \leqslant n \leqslant 9$, die sämtlichen Reimschemen der Länge n, und zwar in der natürlichen Reihenfolge. Für $n{:=}3$ also:

111, 112, 121, 122, 123.

Am Schluss wird noch die Anzahl der Schemen angegeben, für $n{:=}3$ also 5.0.
Die n-stellige Zahl $11\cdots1$ ist leicht als $[10^n/9]$ zu berechnen, und die wesentliche Aufgabe besteht somit darin, zu einem Schema das nächste zu bilden. Wir betrachten den Anfang für $n{:=}4$:

1111, 1112, 1121, 1122, 1123, 1211, 1212.

Man bemerkt, dass in den meisten Fällen auf f einfach $f+1$ folgt. In dem obigen Beispiel gibt es zwei Ausnahmen: (1112; 1121) und (1121; 1211); auch in diesen Fällen wird eine Ziffer um 1 erhöht. Was vor dieser Ziffer steht, wird nicht verändert; in der neuen Folge steht nach der erhöhten Ziffer $1\cdots1$. Wann ist nun $f+1$ der Nachfolger von f? Offenbar genau wenn $f+1$ selbst ein Reimschema ist, oder - gleichbedeutend - wenn die letzte Ziffer von f kleiner oder gleich einer früheren Ziffer ist. In 121312 erkennt man, dass die letzte Ziffer erhöht werden kann auf Grund der Ziffer 3. Falls für eine Folge f - wie etwa 1123 - feststeht, dass die letzte Ziffer nicht erhöht werden darf, so gehe man zu f^- über, der Folge, die aus f durch Streichen der letzten Ziffer entsteht. Ist - wie im Falle 1123 - auch bei dieser Folge die letzte Ziffer nicht zu erhöhen, so werde das Verfahren iteriert. Im betrachteten Fall gelangt man so zu 11; hier kann die letzte Ziffer zu 12 erhöht werden und der Nachfolger von 1123 wird als 1211 erhalten, indem noch «11» angefügt wird.

Um diesen Prozess einfach beschreiben und programmieren zu können, führen wir eine Funktion w ein, welche Paare $\langle f,p\rangle$ (wobei f ein Reimschema und p eine Potenz von 10 ist) in ebensolche Paare oder aber in eine Zahl abbildet. Die Funktion w wird durch eine Fall-Unterscheidung definiert:

1. $w(\langle 1,p\rangle)=0$ (für alle p).
2. Ist $(f+1)$ ein Reimschema, so ist $w(\langle f,p\rangle)=(f+1)\cdot p+[p/9]$.
3. $w(\langle f,p\rangle)=\langle [f/10], 10p\rangle$ in den übrigen Fällen.

($[f/10]$ stellt das Schema dar, welches durch Streichen der letzten Stelle entsteht.)
Man überlegt sich nun leicht, dass es zu jedem Reimschema f eine natürliche Zahl gibt, so dass

$$w^k(\langle f,1\rangle)$$

eine Zahl ist.
Ist diese Zahl verschieden von 0, so ist sie der gesuchte Nachfolger von f; andernfalls ist f das letzte Reimschema derselben Länge wie f - im Fall 4 also 1234.
Statt eines formalen Beweises betrachten wir zwei Spezialfälle: $\langle 1,1,2,3\rangle$ und $\langle 1,2,3,4\rangle$.

$$w(\langle 1123,1\rangle)=\langle 112,10\rangle;$$
$$w(\langle 112,10\rangle)=\langle 11,100\rangle;$$
$$w(\langle 11,100\rangle)=(11+1)\cdot 100+\left[\frac{100}{9}\right]=1211.$$

$$w(\langle 1234,1\rangle)=\langle 123,10\rangle;$$
$$w(\langle 123,10\rangle)=\langle 12,100\rangle;$$
$$w(\langle 12,100\rangle)=\langle 1,1000\rangle;$$
$$w(\langle 1,1000\rangle)=0.$$

Der Ablauf des Programmes ist damit klar vorgezeichnet. Das Paar $\langle f,p\rangle$ wird mit Hilfe von zwei Speichern dargestellt und die Funktion w in einer Schleife berechnet, welche durchlaufen wird, bis der Wert eine Zahl ist.

3. Zählen

Der Algorithmus ZÄHLEN berechnet zu der natürlichen Zahl n die Anzahl B_n der Reimschemen der Länge n - oder auch die Anzahl der Klasseneinteilungen einer n-zahligen Menge. Es ist $B_1=1, B_2=2, B_3=5$; B_0 ist gleich 1 zu setzen.
Die Funktion B erfüllt eine einfache Rekursionsgleichung. Um sie zu erhalten, zerlegen wir die Menge R_n aller Reimschemen der Länge n in n disjunkte Teilmengen R_n^j $(0\leqslant j\leqslant n-1)$. Das Schema f gehört zu R_n^j, wenn f an genau $(j+1)$ Stellen den Wert 1 hat. Die Anzahl von R_n^j bestimmt sich nun leicht folgendermassen: An erster Stelle von f steht 1; für die übrigen j Stellen mit 1 stehen somit

noch $(n-1)$ Möglichkeiten zur Wahl, und es gibt somit $\binom{n-1}{j}$ Verteilungen der Werte 1 auf die n Stellen. An den restlichen $(n-j-1)$ Stellen stehen die Werte 2,3,... Dafür gibt es B_{n-j-1} Möglichkeiten, denn offenbar werden die Teilfolgen mit diesen Werten durch Subtraktion von 1 bijektiv auf die Reimschemen der Länge $(n-j-1)$ abgebildet. Die Anzahl von R_n^j ist somit

$$B_{n-j-1}\binom{n-1}{j},$$

und es gilt

$$B_n=\sum_0^{n-1} B_{n-j-1}\binom{n-1}{j}.$$

Wir setzen $k:=n-j-1$ und erhalten

$$(*)\ B_n=\sum_{k=n-1}^{0} B_k\binom{n-1}{k};\qquad B_0=1.$$

Aus diesen Beziehungen lassen sich die Werte $B_1, B_2, \ldots$ schrittweise berechnen. Dabei müssen aber für die Berechnung von B_n die Werte $B_{n-1}, B_{n-2}, \ldots, B_1$ gespeichert sein, und die Rekursion liefert somit in einer direkten Anwendung keinen Algorithmus, der sich für einen Rechner wie den HP-25 eignet.
Es soll nun gezeigt werden, wie $(*)$ umgeformt werden kann, so dass der Speicherbedarf zur Berechnung von B_n nicht von n abhängt, und man also mit den 8 Speichern des HP-25 auskommt.
Zur bequemeren Darstellung schreiben wir

$$B_n=\sum_{k=n-1}^{0} B_k q_{kn},\qquad B_0=1.$$

Es gibt dann offenbar ein Polynom F_n (in $\binom{n}{2}$ Variablen), so dass gilt

$$B_n=F_n(q_{01}, q_{02}, \ldots, q_{12}, \ldots, q_{(n-1)n}).$$

Zur genaueren Analyse von F_n betrachten wir die Fälle kleiner n

$$B_1=q_{01}$$
$$B_2=q_{01}q_{12}+q_{02}$$
$$B_3=q_{01}q_{12}q_{23}+q_{02}q_{23}+q_{01}q_{13}+q_{03}$$
$$B_4=q_{01}q_{12}q_{23}q_{34}+q_{02}q_{23}q_{34}+q_{01}q_{13}q_{34}+q_{03}q_{34}+q_{01}q_{12}q_{24}+q_{02}q_{24}+q_{01}q_{14}+q_{04}.$$

Aus diesen Beispielen ergeben sich die folgenden (leicht induktiv zu beweisenden) Aussagen: F_n ist eine Summe von 2^{n-1} Monomen; diese Monome sind Produkte

$$q_{0k_1}q_{k_1k_2}\cdots q_{k_mn}\qquad (0<k_1<\cdots<k_m<n),$$

und zwar tritt jedes solche Produkt genau einmal auf.

Für die folgende Berechnung empfiehlt es sich, die Summanden mit den Zahlen j, $2^{n-1} \leqslant j < 2^n$, zu indizieren. Man erhält dann

$$B_n = \sum_{2^{n-1}}^{2^n-1} P_j,$$

und die Produkte P_j lassen sich vergleichsweise einfach als Funktion von j berechnen.
Wird nämlich j in der Basis 2 dargestellt:

$$j = (1 a_{n-1} \ldots a_1)_2,$$

und sind $k_1, k_2, \ldots, k_m, n$ die (der Grösse nach geordneten) Stellen, an denen die Ziffer 1 steht, so erhalten wir jedes Produkt

$$q_{0k_1} q_{k_1 k_2} \cdots q_{k_m n}$$

genau einmal, wenn j die Werte von $2^n - 1$ bis 2^{n-1} durchläuft. Als Beispiel berechnen wir P_{356}.
Es ist

$$(356)_{10} = (101100100)_2,$$

die Menge der Stellen mit 1 somit $\{3, 6, 7, 9\}$ und P_{356} somit

$$q_{03} q_{36} q_{67} q_{79} = \binom{2}{0} \binom{5}{3} \binom{6}{0} \binom{8}{7} = 80.$$

(Die Darstellung $B_n = \Sigma P_j$ legt es nahe, nach der kombinatorischen Bedeutung der Zahlen P_j zu fragen. Für P_{356} findet man: P_{356} ist die Anzahl der Reimschemen der Länge 9, welche aus den Zahlen 1, 2, 3, 4 gebildet sind und für welche die Zahl 1 zweimal $(2 = 9 - 7)$, die Zahl 2 einmal $(1 = 7 - 6)$, die Zahl 3 dreimal $(3 = 6 - 3)$ und die Zahl 4 dreimal $(3 = 3 - 0)$ auftritt.)

Auf Grund der Darstellung

$$B_n = \sum_{2^n-1}^{2^{n-1}} P_j$$

ergibt sich der Aufbau des Algorithmus fast zwangsläufig:
In einer äusseren Schleife durchläuft j die Werte von $2^n - 1$ bis 2^{n-1} und die - in der inneren Schleife - berechneten Werte P_j werden aufaddiert.
In der inneren Schleife wird zu j das Produkt

$$q_{0k_1} q_{k_1 k_2} \cdots q_{k_m n}$$

berechnet.

In einer höheren Programmiersprache stellt sich der beschriebene Algorithmus etwa folgendermassen dar:

```
begin  read (n);      B:=0;
  for j:=2^(n-1) to 2^n - 1 do
  begin  h:=j;  k1:=0;  k2:=0;  P:=1;
    while  h>0  do
    begin  k2:=k2+1;
      if  odd(h)  then
      begin  P:=P* (k2 choose k1);  k1:=k2  end;
      h:=[h/2]
    end;
    B:=B+P
  end;
  write (B)
end.
```

Im vorgelegten HP-25 Programm ist der Algorithmus schwieriger zu erkennen. Es müssen nämlich die Binomialkoeffizienten q_{ij} selbst rekursiv als Produkte berechnet werden, und aus Ersparnisgründen sind diese Produkte zu einem einzigen zusammengefasst, so dass die einzelnen q_{ij} gar nicht «greifbar» sind.
Für ein genaueres Verständnis ist es deswegen von Vorteil, einen Fall zu betrachten, bei dem die Koeffizienten der Rekursion direkt zu berechnen sind. Ein solcher Fall ist

$$C_n=\sum_{0}^{n-1} C_k r_{kn}, \qquad C_0=1$$

mit

$$r_{kn}=k+\frac{n}{10}.$$

Dieses Beispiel hat folgenden zusätzlichen Vorteil: Wird im Programm r_{ij} in einer Pause angezeigt (am besten im Format FIX 1), so sind sowohl Argumente als auch Funktionswert ersichtlich: 3.5 bedeutet r_{35} hat den Wert 3.5. (Es ist dafür $n\leqslant 9$ zu wählen.)
Um ein Programm für diesen Fall zu erhalten, sind im Programm ZÄHLEN in den Schritten (30) bis (41) folgende Ersetzungen vorzunehmen:

(30) RCL 6 (31) RCL 3 (32) 1 (33) 0 (34) ÷ (35) +
(36) f FIX 1 (37) f PAUSE (38) f FIX 2 (39) STO * 4 (40) GTO 17
(41) g NOP

Im Ablauf dieses Programmes werden kurz angezeigt: Die Zahlen j aus der äusseren Schleife für

$$C_n=\sum_{j} Q_j,$$

die Werte r_{ij}, ihre Produkte Q_j und am Schluss die Summe C_n. (Die Werte r_{ij} erscheinen im Format FIX 1, die andern Werte in FIX 2.)
Im eigentlichen Programm ZÄHLEN werden als Zwischenwerte nur j und P_j angezeigt.
Der Beginn des Programmes bedarf noch einer kurzen Erklärung. Die Zahlen j in ΣP_j werden kombinatorisch gedeutet, indem der Rest bei Division durch 2 getestet wird; dafür ist wesentlich, dass j eine echte ganze Zahl ist. Daher darf 2^n zu Beginn des Programmes nicht einfach mit y^x berechnet werden (es ist schon $2^2 \neq 4$). Es ist für die genaue Berechnung von 2^n die folgende Befehlsfolge gewählt: fy^x, $g \rightarrow H$, f INT. Wegen des Befehles $g \rightarrow H$ bedingt dies $n \leqslant 16$; es ist dies zu verschmerzen, wenn man bedenkt, dass schon bei $n{:=}16$ der Ablauf etwa einen Monat dauert. Wird auf die Anzeige der Werte j, P_j verzichtet, so kann mit zwei zusätzlichen Programmschritten 2^n exakt berechnet werden für $n \leqslant 33$ (d.h. für alle n mit $2^n < 10^{10}$). Aus Gründen der Platzersparnis ist auch das Ende etwas seltsam: Zu Beginn ist der Inhalt von R_0 gleich 2^n, jener von R_1 gleich 0.
Bei jedem Durchgang durch die «grosse Schleife» wird R_0 um 1 erniedrigt, R_1 um 1 erhöht; abgebrochen wird, sobald die Inhalte von R_0, R_1 gleich sind.

4. Programme für HP-25

ERKENNEN

→00		17	STO − 1	34	GTO 36
01	f FIX 0	18	RCL 1	35	STO + 1
→02	STO 0	19	g $x=0$	→36	RCL 1
03	STO 1	20	GTO 40	37	STO 2
04	STO 2	21	1	38	f PAUSE
05	g $x<0$	22	0	39	GTO 10
06	GTO 44	23	÷	→40	RCL 2
07	g FRAC	24	g FRAC	41	1
08	g $x\neq 0$	25	g $x=0$	42	f $x=y$
09	GTO 44	26	GTO 44	43	GTO 48
→10	1	27	-	→44	0
11	0	28	1	45	g $1/x$
12	STO ÷ 1	29	0	46	f LAST x
13	RCL 1	30	*	47	GTO 02
14	g FRAC	31	2	→48	RCL 0
15	g $x=0$	22	$x \rightleftarrows y$	49	GTO 00
16	GTO 44	33	f $x<y$		

Eingabe: x; R/S; Ausgabe: x oder ERROR, je nachdem ob x ein Reimschema darstellt oder nicht. Erscheint x zunächst als «mögliches Reimschema», so werden vor endgültiger Ausgabe die zur Prüfung verwendeten Zahlen kurz angezeigt.

ERZEUGEN

→00		17	0	34	1
01	f REG	18	÷	35	STO + 0
02	f FIX 0	19	STO 1	36	RCL 5
03	1	20	g FRAC	37	STO * 0

```
  04  0
  05  x⇄y
  06  f y^x
→07   9
  08  ÷
  09  f INT
  10  STO+0
  11  1
  12  STO 5
  13  STO+6
  14  RCL 0
  15  f PAUSE
→16   1
  21  STO−1
  22  STO 2
→23   1
  24  0
  25  STO÷1
  26  RCL 2
  27  RCL 1
  28  g x=0
  29  GTO 39
  30  g FRAC
  31  STO−1
  32  f x<y
  33  GTO 23
  38  GTO 07
→39   1
  40  0
  41  STO÷0
  42  STO*5
  43  RCL 0
  44  f INT
  45  STO 0
  46  g x≠0
  47  GTO 16
  48  RCL 6
  49  f FIX 1
```

Eingabe: n (n ganz, $1 \leqslant n \leqslant 9$); R/S; Ausgabe: Reimschemen der Länge n (mit Pausen), Halt mit Anzahl der Schemen. Für die Eingabe n:=3 somit 111; 112; 121; 122; 123; 5.0.
(Schemen werden in Format 0, die Anzahl im Format 1 angegeben.)

ZÄHLEN

```
→00
  01  f REG
  02  2
  03  x⇄y
  04  f y^x
  05  g→H
  06  f INT
  07  STO 0
→08   0
  09  STO 3
  10  1
  11  STO−0
  12  STO+1
  13  STO 4
  14  RCL 0
  15  STO 5
  16  f PAUSE
→17   RCL 3
  18  STO 6
→19   1
  20  STO+3
  21  2
  22  STO÷5
  23  RCL 5
  24  g x=0
  25  GTO 42
  26  g FRAC
  27  g x=0
  28  GTO 19
  29  STO−5
  30  RCL 3
  31  STO 7
→32   RCL 6
  33  g x=0
  34  GTO 17
  35  STO÷4
  36  1
  37  STO−6
  38  STO−7
  39  RCL 7
  40  STO*4
  41  GTO 32
→42   RCL 4
  43  STO+2
  44  f PAUSE
  45  RCL 0
  46  RCL 1
  47  f x≠y
  48  GTO 08
  49  RCL 2
```

Eingabe: n (n ganz, $1 \leqslant n \leqslant 16$); Ausgaben: 2^n-1, P_{2^n-1}, 2^n-2, P_{2^n-2}, ..., 2^{n-1}, P_{2^n-1}, B_n. Dabei ist B_n die Anzahl der Reimschemen der Länge n; P_j ($2^{n-1} \leqslant j \leqslant 2^n-1$) sind die Summanden, mit deren Hilfe B_n dargestellt wird.

E. Specker, Zürich

LITERATURVERZEICHNIS

1 P. Henrici: Computational Analysis with the HP-25 Pocket Calculator. John Wiley, New York 1977.
2 G.-C. Rota: The number of partitions of a set. Amer. Math. Monthly *71*, 498-504 (1964).
3 N. J. A. Sloane: A Handbook of Integer Sequences. Academic Press, New York 1973.