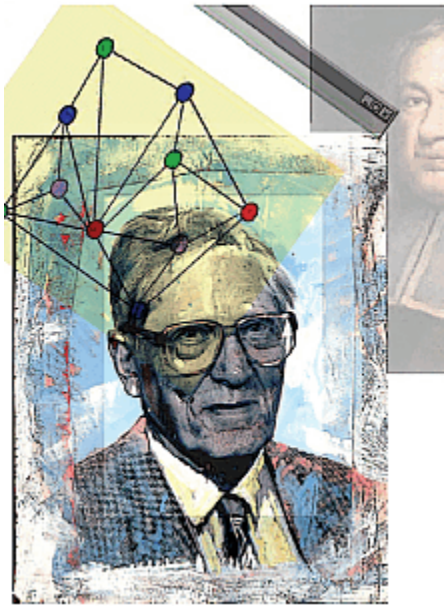
Zwei Maple 7-Prozeduren zur Erzeugung von de Bruijn-Folgen[1]



Gegeben seien ein Alphabet A mit s Buchstaben und eine natürliche Zahl n.
Ein Wort der Länge s^n heißt ***de Bruijn-Folge***, wenn es – zyklisch gelesen – alle Wörter der Länge n als Teilwörter enthält.
Pate für diese Namensgebung ist der holländische Mathematiker und Informatiker *Nicolaas Govert de Bruijn* (* 9. 7. 1918), der die Existenz solcher Wörter in seiner Arbeit „A combinatorial Problem“ (Indagationes Mathematicae 8/1946, Seiten 461-467) bewies. Für $s = 2$ sind diese Folgen insbesondere in der Codierungstheorie von Bedeutung.

De Bruijn-Folgen kann man über Eulersche Kreise in einem passend gewählten gerichteten Graphen erhalten. Die Knoten des Graphen sind die s^{n-1} Wörter der Länge n-1, die Kanten die s^n Wörter der Länge n. Der Anfangsknoten einer Kante ist das Teilwort, das durch Weglassen des letzten Buchstaben entsteht; den Endknoten erhält man durch Weglassen des ersten Buchstaben. Dieser Graph hat Schlingen, aber keine Mehrfachkanten. Schlingen sind die Wörter, deren Buchstaben alle gleich sind. Jeder Knoten ist Anfangsknoten und Endknoten von jeweils s Kanten; also ist der Graph pseudo-symmetrisch. Zwei Knoten $a_1a_2\ldots a_{n-1}$ und $b_1b_2\ldots b_{n-1}$ sind durch den gerichteten Kantenzug

$$a_1a_2\ldots a_{n-1}b_1,\ a_2a_3\ldots a_{n-1}b_1b_2,\ \ldots,\ a_{n-1}b_1b_2\ldots b_{n-1}$$

verbunden; damit ist der Graph auch zusammenhängend und es gibt einen Eulerschen Kreis aus s^n Kanten. Nimmt man die letzten Buchstaben dieser Kanten in der gegebenen Reihenfolge, so hat man eine de Bruijn-Folge.

Im Folgenden wird dies durch zwei Prozeduren des Computeralgebrasystems Maple ausgeführt. Die erste, einfachere arbeitet mit *Listen*, die zweite etwas kompliziertere mit *Arrays*. Die Schwierigkeit mit Arrays liegt darin, dass man nicht wie bei Listen die Länge bei Bedarf abfragen kann, sondern mit fester vorgegebener Länge arbeiten muss. Der Übergang zu Arrays erweist aber als notwendig, wenn man mit größeren Werten von s und n arbeiten will. Schon bei $s = 5$ und $n = 4$ liefert die Prozedur mit Listen die Fehlermeldung:

```
Error, (in anhaengen) assigning to a long list, please use arrays
```

Zunächst werden die Listen der Knoten und Kanten, das heißt, der Wörter der Längen n-1 und n, hergestellt. In der Maple-Terminologie sind das Zeichenketten. Das geschieht mit zwei Unterprozeduren. Die erste, `anhaengen`, verlängert jedes Wort einer gegebenen Liste von Wörtern um einen Buchstaben, und zwar um jeden Buchstaben des Alphabets.

---

[1] erstellt für eine zusammen mit Nikolaj Vladislavovič Malakhovskij gehaltenen Vorlesung zur Angewandten Graphentheorie an der Kaliningrader Staatlichen Universität.

```
anhaengen:=proc(WW,AA)
   local n1, n2, nw, A1, W1;
   nw:=nops(WW); W1:=[seq(a[j],j=1..nw)];
   for n1 to nw do A1:=AA;
      for n2 to nops(AA) do
         A1:=subsop(n2=cat(WW[n1],AA[n2]),A1)
      end do;
      W1[n1]:=A1
   end do;
   for n1 to nw-1 do
      W1:=subsop(1=NULL,2=[op(W1[1]),op(W1[2])],W1)
   end do;
   W1:=W1[1]
end proc:
```

Hier steht `WW` für die vorgebene Liste der zu verlängernden Wörter und `AA` für die Liste der anzuhängenden Buchstaben.

Damit bilden wir die Liste der Wörter der Länge `nn` aus den Buchstaben des Alphabets `AA`:

```
nwoerter:=proc(AA,nn)
   local n1, W;
   W:=AA;
   for n1 to nn-1 do W:=anhaengen(W,A) end do;
end proc:
```

Jetzt bilden wir mit der Prozedur `VAA` die Inzidenzlisten, die für jeden Knoten angeben, zu welchen Kanten er der Anfangsknoten ist:

```
VAA:=proc(AA,nn)
   local k; global E, V, VVA;
   V:=nwoerter(AA,nn-1); E:=nwoerter(AA,nn);
   VVA:=[seq(va[j],j=1..nops(V))];
   for k to nops(V) do VVA[k]:=anhaengen([V[k]],AA) end do
end proc:
```

Jetzt erzeugen wir mit der Prozedur `kreis` einen Kreis in unserem Graphen, der in einem vorgegebenen Knoten `vv` beginnt und die in den eventuell schon verkürzten Inzidenzlisten `B` vorkommenden Kanten benutzt, wobei der Liste `L` die Knoten des Kreises angehängt werden, von denen noch nicht verbrauchte Kanten ausgehen. Dabei verstehen wir hier unter einem Kreis eine alternierende Folge von Knoten und Kanten, das heißt, wir ergänzen die Kantenfolge um den Anfangsknoten der ersten Kante, den Endknoten der letzten Kante – da ein Kreis vorliegt, ist dieser gleich dem Anfangsknoten der ersten Kante – und fügen zwischen zwei Kanten den gemeinsamen Knoten ein. Das hat für das weitere Vorgehen gewisse Vorteile.

```
kreis:=proc(vv)
   local e, w, nn, nv, nw; global B, K, L, V, n;
   v:=vv; K:=[v];
   member(v,V,'nv');
   while B[nv]<>[] do
      e:=VVA[nv][1]; B[nv]:=subsop(1=NULL,B[nv]);
      v:=substring(e,2..n); nv:=indces(v,V);
      if member(v,L)=false and B[nv]<> [] then
         L:=[op(L),v]
      end if;
      K:=[op(K),e,v]
   end do;
end proc:
```

Damit kommen wir zu der endgültigen Prozedur, die zunächst einen Eulerschen Kreis bestimmt und dann daraus die de Bruijn-Folge distilliert.

```
deBruijn:=proc(AA,nn)
   local  C, v, nk, nm; global E, B, K, L, V, VVA, n;
   VAA(AA,nn); B:=VVA; K:=[V[1]]; L:=[];
   kreis(V[1]);
   C:=K;
   while L<>[] do
      v:=L[nops(L)];
      L:=subsop(nops(L)=NULL,L);
      kreis(v);
      C:=subsop(indces(v,C)=op(K),C)
   end do; # jetzt ist der Eulersche Kreis konstruiert
   for nm to (nops(C)+1)/2 do C:=subsop(nm=NULL,C) end do;
   # damit werden die Knoten aus dem Kreis entfernt
   for nm to nops(C) do
      C:=subsop(nm=substring(C[nm],nn),C)
   end do; # die Folge der Kanten wird durch die Folge der
             letzten Buchstaben ersetzt
   for nm to nops(C)-1 do
      C:=subsop(1=NULL,2=cat(C[1],C[2]),C)
   end do; # die Folge der Buchstaben wird zu einer
             Zeichenkette zusammengefasst, einer Liste
             aus einem Element
   op(C) # das einzige Element dieser Liste wird entnommen.
end proc:
```

Angewandt auf das Alphabet `A:=["a","b","c","d"]` und die Wortlänge `n` liefert diese Prozedur die de Bruijn-Folge

```
"abaaacaaadaabbaabcaabdaacbaaccaacdaadbaadcaaddababacabadabbbabbcabbdabc   \
    babccabcdabdbabdcabddacacadacbbacbcacbdaccbacccaccdacdbacdcacddadadb   \
    badbcadbdadcbadccadcdaddbaddcadddbbbbcbbbdbbccbbcdbbdcbbddbcbcbdbcc   \
    cbccdbcdcbcddbdbdccbdcdbddcbdddccccdccddcdcddddaaa"
```

ein Wort aus $4^4 = 256$ Buchstaben.

Nun entwickeln wir die Prozeduren unter Verwendung von Arrays, die – wie schon gesagt – vorgegebene Längen benötigen. Bei der Prozedur anhaengen wird jetzt zunächst eine Matrix erstellt, deren Zeilen hinterher in ein Array zusammengefasst werden.

```
anhaengen:=proc(ss,mm,AA,WW)
    local W1, W2, i, j, k; global m;
    W1:=array(1..ss,1..mm);
    for i to ss do
       for j to mm do W1[i,j]:=cat(WW[j],AA[i]) end do
    W2:=array(1..ss*mm);
    for k to ss*mm do
       if irem(k,mm)=0 then W2[k]:=W1[iquo(k,mm),mm]
       else W2[k]:=W1[iquo(k,mm)+1,irem(k,mm)]
       end if
    end do;
    m:=ss*mm;
    eval(W2)
end proc:
```

Die Herstellung der Woerter ist nun einfach:

```
nwoerter:=proc(ss,nn,AA)
   local W, m; global mn;
   W:=AA;
   for m to nn-1 do mn:=ss^m;
      W:=anhaengen(ss,mn,AA,W)
   end do; mn:=ss^nn;
   eval(W)
end proc:
```

Das gleiche gilt für die Inzidenzlisten der Anfangsknoten:

```
VAA:=proc(ss,nn,AA)
   local n1;
   global nnv, V, VV;
   VV:=array(1..nnv);
   for n1 to nnv do
      VV[n1]:=anhaengen(s,1,AA,[V[n1]])
   end do
end proc:
```

Eine Indexermittlung wird benötigt:

```
indces:=proc(v,ll,c::array)
   local ni, nv;
   for ni to ll do
      if c[ni]=v then nv:=ni end if
   end do;
   nv
 nd proc:
```

Nun können wir einen Kreis konstruieren:

```
kreis:=proc(vv)
   local KK, V1, e, p, nv, w;
   global K, L, M, V, VV, lk, n, nnv;
   w:=vv; nv:=indces(w,nnv,V); K:=[w]; lk:=1;
   while L[nv]<0 do
      e:=VV[nv][1];
      L[nv]:=L[nv]-1;
      if L[nv]<0 then V1:=array(1..L[nv]);
         for p to L[nv] do V1[p]:=VV[nv][p+1] end do;
         VV[nv]:=eval(V1)
         else VV[nv]:=array([])
      end if;
      w:=substring(e,2..n);
      nv:=indces(w,nnv,V); M[nv]:=1;
      KK:=array(1..lk+2); for p to lk do KK[p]:=K[p] end do;
      KK[lk+1]:=e; KK[lk+2]:=w; lk:=lk+2; K:=eval(KK)
   end do;
   M[indces(K[lk],nnv,V)]:=0;
   eval(K), lk
end proc:
```

Schließlich erhalten wir die gesuchte de Bruijn-Folge mit der Prozedur:

```
deBruijn:=proc(ss,nn,AA)
   local CC, E, KK, W, k, p, q, r, t, w;
   global C, K, L, M, V, glk, lk, mn, nne, nnv;
   V:=nwoerter(ss,nn-1,AA); nnv:=mn;
   E:=nwoerter(ss,nn,AA); nne:=mn;
   VAA(ss,nn,AA);
   L:=array(1..nnv); M:=array(1..nnv);
   for p to nnv do L[p]:=s; M[p]:=0 end do;
   CC:=kreis(V[1]); C:=CC[1]; glk:=CC[2]; q:=1;
   while q=1 do
      k:=indces(1,nnv,eval(M));
      kreis(V[k]); t:=indces(K[1],glk,C);
```

```
      KK:=array(1..glk+lk-1);
      for p to t do KK[p]:=C[p] end do;
      for p from t to t+lk-1 do KK[p]:=K[p-t+1] end do;
      for p from t+lk-1 to glk+lk-1 do
         KK[p]:=C[p-lk+1]
      end do;
      C:=eval(KK); glk:=glk+lk-1; r:=0;
      for p to nnv do if L[p]=0 then M[p]:=0 end if end do;
      for p to nnv do r:=r+M[p] end do;
      if r=0 then q:=0 end if
   end do; r:=(glk-1)/2;
   W:=array(1..r);
   for p to r do W[p]:=substring(C[2*p],n) end do;
   w:=W[1]; for p to r-1 do w:=cat(w,W[p+1]) end do; w
end proc:
```

Damit kann man in erträglicher Zeit eine de Bruijn-Folge zu Wörtern der Länge 5 über einem Alphabet von 5 Buchstaben erhalten.