Acta Univ. Sapientiae, Informatica, **1**, 1 (2009) 109–118

Generating and ranking of Dyck words

Zoltán Kása
Sapientia Hungarian University of Transylvania
Department of Mathematics and Informatics,
Târgu Mureş
email: kasa@ms.sapientia.ro

Abstract. A new algorithm to generate all Dyck words is presented, which is used in ranking and unranking Dyck words. We emphasize the importance of using Dyck words in encoding objects related to Catalan numbers. As a consequence of formulas used in the ranking algorithm we can obtain a recursive formula for the nth Catalan number.

1 Introduction

Let $B = \{0, 1\}$ be a binary alphabet and $x_1x_2 \ldots x_n \in B^n$. Let $h : B \to \{-1, 1\}$ be a valuation function with $h(0) = 1$, $h(1) = -1$, and $h(x_1x_2 \ldots x_n) = \sum_{i=1}^{n} h(x_i)$.

A word $X = x_1x_2 \ldots x_{2n} \in B^{2n}$ is called a *Dyck word* [4] if it satisfies the following conditions:

$$h(x_1x_2 \ldots x_i) \geq 0, \text{ for } 1 \leq i \leq 2n-1$$
$$h(x_1x_2 \ldots x_{2n}) = 0.$$

n is the semilength of the word.

AMS 2000 subject classifications: 68R05
CR Categories and Descriptors: G.2.1. [Combinatorics]: Subtopic - Combinatorial algorithms.
Key words and phrases: Dyck words, generating and ranking algorithms, Catalan numbers

2 Lexicographic order

The algorithm that generates all Dyck words in lexicographic order is obvious. Let us begin with 0 in the first position, and add 0 or 1 each time the Dyck-property remains valid. In the following algorithm $2n$ is the length of a Dyck word, n_0 counts the 0s, and n_1 the 1s.

There are the following cases :

Case 1: $(n_0 < n)$ and $(n_1 < n)$ and $(n_0 > n_1)$ (We can continue by adding 0 and 1.)
Case 2: $(n0 < n)$ and $(n_1 < n)$ and $(n_0 = n_1)$ (We can continue by adding 0 only.)
Case 3: $(n_0 < n)$ and $(n_1 = n)$ (We can continue by adding 0 only.)
Case 4: $(n_0 = n)$ and $(n_1 < n)$ (We can continue by adding 1 only.)
Case 5: $(n_0 = n_1 = n)$ (A Dyck word is obtained.)

Let us use the following short notations:

Dyck 0 for
  $x_i \leftarrow 0$
  $n_0 \leftarrow n_0 + 1$
  LexDyckWords(X, i, n_0, n_1)
  $n_0 \leftarrow n_0 - 1$

Dyck 1 for
  $x_i \leftarrow 1$
  $n_1 \leftarrow n_1 + 1$
  LexDyckWords(X, i, n_0, n_1)
  $n_1 \leftarrow n_1 - 1$

The algorithm is the following:

```
LexDyckWords(X, i, n0, n1)
if Case 1
   then i ← i + 1
        Dyck 0
        Dyck 1
if Case 2 or Case3
   then i ← i + 1
        Dyck 0
if Case 4
   then i ← i + 1
        Dyck 1
if Case 5
   then Visit x1x2 ... xn
return
```

The recursive call:
  $x_1 \leftarrow 0$, $n_0 \leftarrow 1$, $n_1 \leftarrow 0$
  LexDyckWords$(X, 1, n_0, n_1)$

For $n = 4$ we obtain:

00001111, 00010111, 00011011, 00011101, 00100111, 00101011, 00101101, 00110011, 00110101, 01000111, 01001011, 01001101, 01010011, 01010101.

This algorithm obviously generates all Dyck words.

3 Generating the positions of 1s

Let $b_1 b_2 \dots b_n$ be the positions of 1s in the Dyck word $x_1 x_2 \dots x_{2n}$. E.g. for $x_1 x_2 \dots x_8 = 01010011$ we have $b_1 b_2 b_3 b_4 = 2478$.

To be a Dyck word of semilength n, the positions $b_1 b_2 \dots b_n$ of 1s of the word $x_1 x_2 \dots x_{2n}$ must satisfy the following conditions:

$$2i \le b_i \le n + i, \quad \text{for} \quad 1 \le i \le n.$$

Following the idea of generating combinations by positions of 0s in the corresponding binary string [5] we propose a similar algorithm that generates the positions $b_1 b_2 \dots b_n$ of 1s.

```
PosDyckWords(n)
for i ← 1 to n
    do b_i ← 2i
repeat
    Visit b_1 b_2 ... b_n
    IND ← 0
    for i ← n − 1 downto 1
        do if b_i < n + i
            then b_i ← b_i + 1
                 for j ← i + 1 to n − 1
                     do b_j ← max(b_{j−1} + 1, 2j)
                 IND ← 1
                 break (for)
until IND = 0
return
```

For $n = 4$ we obtain:

2468, 2478, 2568, 2578, 2678, 3468, 3478, 3568, 3578, 3678, 4568, 4578, 4678, 5678.

The corresponding Dyck words are:

01010101, 01010011, 01001101, 01001011, 01000111, 00110101, 00110011,
00101101, 00101011, 00100111, 00011101, 00011011, 00010111, 00001111.

Because all values of positions that are possible are taken by the algorithm, it generates all Dyck words. Words are generated in reverse lexicographic order.

4 Generating by changing 10 in 01

The basic idea [2] is to change the first occurence of 10 in 01 to get a new Dyck word. We begin with $0101\dots01$.

```
DyckWords(X, k)
i ← k
while i < 2n
      do Let j be the position of the first occurence of 10 in x_i x_{i+1} ... x_{2n},
         or 0 if such a position doesn't exist.
         if j > 0
            then Let Y ← X
                 Change y_i with y_{i+1}.
                 Visit y_1 y_2 ... y_{2n}
                 DyckWords(Y, j − 1)
                 i ← j + 2
return
```

The first call is DyckWords$(X, 1)$, if $X = 0101\dots01$.

For $X = 01010101$, the algoritm generates:

01010101, 00110101, 00101101, 00011101, 00011011, 00010111, 00001111,
00101011, 00100111, 00110011, 01001101, 01001011, 01000111, 01010011.

Can this algorithm always generate all Dyck words? To prove this we show that any Dyck word can be tranformed to $(01)^n$ by several changing of 01 in 10. Let us consider the leftmost subword of the form 0^i1, for $i > 0$. Changing 01 in 10 $(i-1)$ times, we will obtain a leftmost subword of the form $0^{i-1}1$. So, all subwords of this form can be avoided.

5 Ranking Dyck words

Ranking Dyck words means [6] to determine the position of a Dyck word in a given ordered sequence of all Dyck words.

Algorithm PosDyckWords generates all Dyck word in reverse lexicographic order. For ranking these words we will use the following function [7], where $f(i,j)$ represents the number of paths between (0,0) and (i,j) not crossing the diagonal $x = y$ of the grid.

$$f(i,j) = \begin{cases} 1, & \text{for } 0 \le i \le n, j = 0 \\ f(i-1,j) + f(i,j-1), & \text{for } 1 \le j < i \le n \\ f(i,i-1), & \text{for } 1 \le i = j \le n \\ 0, & \text{for } 0 \le i < j \le n \end{cases} \qquad (1)$$

Some values of this function are given in the following table.

j											
9										4862	
8									1430	4862	
7								429	1430	3432	
6							132	429	1001	2002	
5						42	132	297	572	1001	
4					14	42	90	165	275	429	
3				5	14	28	48	75	110	154	
2			2	5	9	14	20	27	35	44	
1		1	2	3	4	5	6	7	8	9	
0	1	1	1	1	1	1	1	1	1	1	
	0	1	2	3	4	5	6	7	8	9	i

It is easy to prove that if C_n is the nth Catalan number then

$$C_{n+1} = f(n+1,n) = \sum_{i=0}^{n} f(n,i), \quad n \ge 0 \qquad (2)$$

$$f(n+1,k) = \sum_{i=0}^{k} f(n,i), \quad n \ge 0, n \ge k \ge 0.$$

Using this function the following ranking algorithm results.

```
RANKING(b_1 b_2 ... b_n)
c_1 ← 2
for j ← 2 to n
    do c_j ← max(b_{j−1} + 1, 2j)
nr ← 1
for i ← 1 to n − 1
    do for j ← c_i to b_i − 1
            do nr ← nr + f(n − i, n + i − j)
return nr
```

For example, if b =4 5 8 9 10, we get c =2 5 6 9 10, and $nr = 1 + f(4,4) + f(4,3) + f(2,2) + f(2,1) = 1 + 14 + 14 + 2 + 2 = 33$.
This algorithm can be used for ranking in lexicographic order too.

6 Unranking Dyck words

The unranking algorithm for a given n will map a number between 1 and C_n to the corresponding Dyck word represented by positions of 1s. Here the Dyck words are considered in reverse lexicographic order too.

```
UNRANKING(nr)
b_0 ← 0
nr ← nr − 1
for i ← 1 to n
    do b_i ← max (b_{i−1} + 1, 2i)
       j ← n + i − b_i
       while (nr ≥ f(n − i, j)) and (b_i < n + i)
             do nr ← nr − f(n − i, j)
                b_i ← b_i + 1
                j ← j − 1
return b_1 b_2 ... b_n
```

If $n = 6$ and $nr = 93$, we will have: $92 - f(5,5) - f(5,4) - f(3,3) - f(2,2) - f(1,1) = 92 - 42 - 42 - 5 - 2 - 1$, so the corresponding Dyck word represented by positions of 1's is: b = 4 5 7 9 11 12. Are changed from the initial values $2i$ the following: position 1 by 2, position 3 by 1, position 4 by 1 and position 5 by 1.

7 Applications of Dyck words

If $\mathcal{O}$ is a set of C_n objects, Dyck words can be used for encoding the objects of $\mathcal{O}$. The importance of such an encoding currently is not suitably accentuated. We present here an encoding and decoding algorithms for binary trees, based on [1].

Algorithm for encoding a binary tree
Let B_L be the left and B_R the right subtree of the binary tree B. $w01$ means the concatenation of word w with 01, and w is considered a global variable.

```
EncodingBT(B)
if B_L ≠ ∅ and B_R = ∅
   then w ← w01
        EncodingBT(B_L)
if B_L = ∅ and B_R ≠ ∅
   then w ← w10
        EncodingBT(B_R)
if B_L ≠ ∅ and B_R ≠ ∅
   then w ← w00
        EncodingBT(B_L)
        w ← w11
        EncodingBT (B_R)
return
```

Call:

```
w ← 0
EncodingBT(B)
w ← w1
```

For all trees of $n = 4$ vertices the result of the algorithm is given in Fig. 1.

Algorithm to decode a Dyck word into a binary tree
At the beginning the root of the generated binary tree is the current vertex. When an edge is drawn, its endvertex becomes the current vertex.

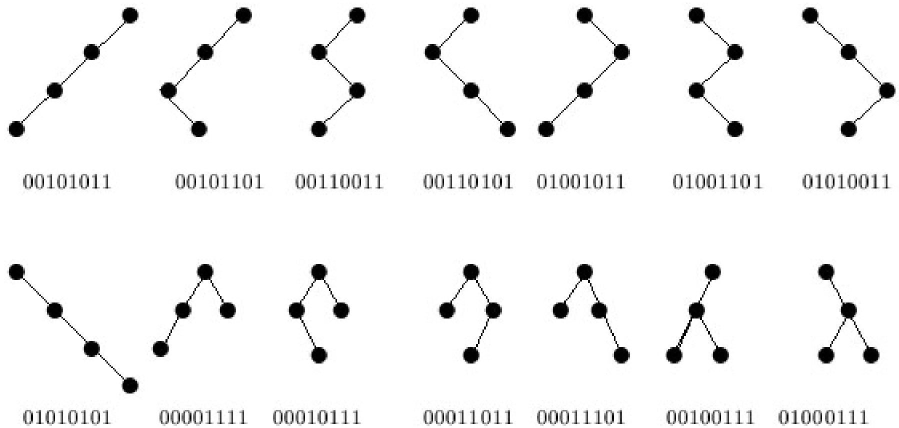

Figure 1: Encoding of binary trees for $n = 4$.

```
DecodingBT(w)
Let ab be the first two letters of w.
Delete ab from w.
if ab = 01
   then draw a left edge from the current vertex
        DecodingBT(w)
if ab = 10
   then draw a right edge from the current vertex
        DecodingBT(w)
if ab = 00
   then put in the stack the position of the current vertex
        draw a left edge from the current vertex
        DecodingBT(w)
if ab = 11
   then get from the stack the position of the new current vertex
        draw a right edge from the current vertex
        DecodingBT(w)
return
```

Call:

delete 0 from the beginning and 1 from the end of the input word w
draw a vertex (the root of the tree) as current vertex
DecodingBT(w)

For some other objects related to Catalan numbers the corresponding coding can be found in [1] and at http://www.ms.sapientia.ro/˜kasa/CodingDyck.pdf.

8 A consequence

As a consequence of formulas (1) and (2) the following formula for the $(n+1)$th Catalan number results:

$$C_{n+1} = 1 + \sum_{k\geq 0} (-1)^k \binom{n-k}{k+1} C_{n-k}. \tag{3}$$

We can prove that

$$f(n, n-k) = \sum_{i=0}^{n} (-1)^i \binom{k-i}{i} C_{n-i}$$

for appropriate n and k, using mathematical induction on n and k, and formula (1) in the form

$$f(n, n-k) = f(n, n-k+1) - f(n-1, n-k+1).$$

Now, from (2)

$$\begin{aligned} C_{n+1} &= \sum_{i=0}^{n} f(n,i) = f(n,0) + \sum_{i=1}^{n} f(n,i) = 1 + \sum_{i=0}^{n-1} f(n, n-i) \\ &= 1 + \sum_{i=0}^{n-1} \left(\sum_{k=0}^{n} (-1)^k \binom{i-k}{k} C_{n-k} \right) \\ &= 1 + \sum_{k=0}^{n} (-1)^k C_{n-k} \left(\sum_{i=0}^{n-1} \binom{i-k}{k} \right) \\ &= 1 + \sum_{k=0}^{n} (-1)^k \binom{n-k}{k+1} C_{n-k}. \end{aligned}$$

In the last line $\binom{k}{k} + \binom{k+1}{k} + \cdots + \binom{n-1-k}{k} = \binom{n-k}{k+1}$ was used.

References

[1] A. Bege, Z. Kása, Coding objects related to Catalan numbers, *Studia Univ. Babeş-Bolyai Infor.* **46**, 1 (2001) 31–40.

[2] A. Bege, Z. Kása, *Algoritmikus kombinatorika és számelmélet*, Presa Universitară Clujeană, 2006.

[3] E. Deutsch, Dyck path enumeration, *Discrete Math.* **204**, 1–3 (1999) 167–202.

[4] P. Duchon, On the enumeration and generation of generalized Dyck words, *Discrete Math.* **225**, 1–3 (2000) 121–135.

[5] D. E. Knuth, *The Art of Computer Programming, Vol. 4. Fasc. 3, Generating All Combinations and Partitions*, Addison-Wesley Reading MA., 2005.

[6] J. Liebehenschel, Ranking and unranking of lexicographically ordered words: An average-case analysis. *J. Autom. Lang. Comb.*, **2**, 4 (1997) 227–268

[7] W. Yang, *Discrete Mathematics*, http://www.cis.nctu.edu.tw/˜wuuyang/, manuscript.

Received: November 18, 2008