Generating and ranking of Dyck words

Zoltán Kása
Sapientia Hungarian University of Transylvania
Department of Mathematics and Informatics,
Târgu Mureș
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 n-th Catalan number.

1 Introduction

Let $B = \{0, 1\}$ be a binary alphabet and $x_1x_2\ldots x_n \in B^n$. Let $h : B \rightarrow \{-1, 1\}$ 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: ($n_0 < n$) 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:

\[
\begin{align*}
\text{Dyck 0 for} & \quad x_1 \leftarrow 0 \\
& \quad n_0 \leftarrow n_0 + 1 \\
& \quad \text{LEXDYCKWORDS}(X, i, n_0, n_1) \\
& \quad n_0 \leftarrow n_0 - 1
\end{align*}
\]

\[
\begin{align*}
\text{Dyck 1 for} & \quad x_1 \leftarrow 1 \\
& \quad n_1 \leftarrow n_1 + 1 \\
& \quad \text{LEXDYCKWORDS}(X, i, n_0, n_1) \\
& \quad n_1 \leftarrow n_1 - 1
\end{align*}
\]

The algorithm is the following:

\[
\begin{align*}
\text{LEXDYCKWORDS}(X, i, n_0, n_1) \\
& \quad 1 \text{ if Case 1} \\
& \quad 2 \quad \text{then } i \leftarrow i + 1 \\
& \quad 3 \quad \text{ Dyck 0} \\
& \quad 4 \quad \text{ Dyck 1} \\
& \quad 5 \text{ if Case 2 or Case 3} \\
& \quad 6 \quad \text{then } i \leftarrow i + 1 \\
& \quad 7 \quad \text{ Dyck 0} \\
& \quad 8 \text{ if Case 4} \\
& \quad 9 \quad \text{then } i \leftarrow i + 1 \\
& \quad 10 \quad \text{ Dyck 1} \\
& \quad 11 \text{ if Case 5} \\
& \quad 12 \quad \text{then Visit } x_1 x_2 \ldots x_n \\
& \quad 13 \text{ return}
\end{align*}
\]

The recursive call:

\[
\begin{align*}
& \quad x_1 \leftarrow 0, \ n_0 \leftarrow 1, \ n_1 \leftarrow 0 \\
& \quad \text{LEXDYCKWORDS}(X, 1, n_0, n_1)
\end{align*}
\]
Generating and ranking of Dyck words

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 \( b_1b_2\ldots b_n \) be the positions of 1s in the Dyck word \( x_1x_2\ldots x_{2n} \). E.g. for \( x_1x_2\ldots x_8 = 01010011 \) we have \( b_1b_2b_3b_4 = 2478 \).

To be a Dyck word of semilength \( n \), the positions \( b_1b_2\ldots b_n \) of 1s of the word \( x_1x_2\ldots x_{2n} \) must satisfy the following conditions:

\[
2i \leq b_i \leq n + i, \quad \text{for } 1 \leq i \leq n.
\]

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 \( b_1b_2\ldots b_n \) of 1s.

\[
\text{PosDyckWords}(n)
\]

\[
\begin{array}{l}
\text{for } i \leftarrow 1 \text{ to } n \\
\text{do } b_i \leftarrow 2i \\
\text{repeat} \\
\text{Visit } b_1b_2\ldots b_n \\
\text{IND } \leftarrow 0 \\
\text{for } i \leftarrow n - 1 \text{ downto } 1 \\
\text{do if } b_i < n + i \\
\text{then } b_i \leftarrow b_i + 1 \\
\text{for } j \leftarrow i + 1 \text{ to } n - 1 \\
\text{do } b_j \leftarrow \max(b_{j-1} + 1, 2j) \\
\text{IND } \leftarrow 1 \\
\text{break (for)} \\
\text{until IND } = 0 \\
\text{return}
\end{array}
\]

For \( n = 4 \) we obtain:

\[
2468, 2478, 2568, 2578, 2678, 3468, 3478, 3568, 3578, 3678, 4568, 4578, 4678, \\
5678.
\]
The corresponding Dyck words are:

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

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 occurrence of 10 in 01 to get a new Dyck word. We begin with 0101\ldots01.

```latex
\textbf{DYCKWords}(X, k)
\begin{align*}
&1\quad i \gets k \\
&2\quad \textbf{while } i < 2^n \\
&3\quad \quad \textbf{do} \quad \text{Let } j \text{ be the position of the first occurrence of } 10 \text{ in } x_i x_{i+1} \ldots x_{2^n}, \\
&\quad \quad \quad \text{or } 0 \text{ if such a position doesn't exist.} \\
&4\quad \quad \textbf{if } j > 0 \\
&5\quad \quad \quad \textbf{then} \quad \text{Let } Y \gets X \\
&6\quad \quad \quad \text{Change } y_i \text{ with } y_{i+1}. \\
&7\quad \quad \quad \text{Visit } y_1 y_2 \ldots y_{2^n} \\
&8\quad \quad \quad \textbf{DYCKWords}(Y, j - 1) \\
&9\quad \quad i \gets j + 2 \\
&10\quad \textbf{return}
\end{align*}
```

The first call is \textbf{DYCKWords}(X, 1), if \(X = 0101 \ldots 01\).

For \(X = 01010101\), the algorithm generates:

\[
01010101, 00110101, 00110111, 00011101, 00010111, 00001111, 00101011, 00100111, 00110011, 01001101, 01001011, 01000111, 01010011.
\]

Can this algorithm always generate all Dyck words? To prove this we show that any Dyck word can be transformed 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.

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

(1)

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

| j |     |     |     |     |     |     |     |     |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
|   | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| 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   |
| 0 | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |

|     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | i |
|-----|---|---|---|---|---|---|---|---|---|---|---|
| 0   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |   |
| 1   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | i |

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

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

(2)

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

Using this function the following ranking algorithm results.
RANKING($b_1b_2\ldots b_n$)
1  $c_1 \leftarrow 2$
2  for $j \leftarrow 2$ to $n$
3      do $c_j \leftarrow \max(b_{j-1} + 1, 2j)$
4  $nr \leftarrow 1$
5  for $i \leftarrow 1$ to $n - 1$
6      do for $j \leftarrow c_i$ to $b_i - 1$
7          do $nr \leftarrow nr + f(n - i, n + i - j)$
8  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($nr$)
1  $b_0 \leftarrow 0$
2  $nr \leftarrow nr - 1$
3  for $i \leftarrow 1$ to $n$
4      do $b_i \leftarrow \max(b_{i-1} + 1, 2i)$
5          $j \leftarrow n + i - b_i$
6          while ($nr \geq f(n - i, j)$) and ($b_i < n + i$)
7              do $nr \leftarrow nr - f(n - i, j)$
8      $b_i \leftarrow b_i + 1$
9          $j \leftarrow j - 1$
10 return $b_1b_2\ldots b_n$

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 $O$ is a set of $C_n$ objects, Dyck words can be used for encoding the objects of $O$. 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)
1  if $B_L \neq \emptyset$ and $B_R = \emptyset$
2      then $w \leftarrow w01$
3      ENCODINGBT($B_L$)
4  if $B_L = \emptyset$ and $B_R \neq \emptyset$
5      then $w \leftarrow w10$
6      ENCODINGBT($B_R$)
7  if $B_L \neq \emptyset$ and $B_R \neq \emptyset$
8      then $w \leftarrow w00$
9      ENCODINGBT($B_L$)
10     $w \leftarrow w11$
11     ENCODINGBT($B_R$)
12  return
```

Call:
$w \leftarrow 0$
ENCODINGBT(B)
$w \leftarrow w1$

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)
1 Let $ab$ be the first two letters of $w$.
2 Delete $ab$ from $w$.
3 if $ab = 01$
4 then draw a left edge from the current vertex
5    DECODINGBT(w)
6 if $ab = 10$
7 then draw a right edge from the current vertex
8    DECODINGBT(w)
9 if $ab = 00$
10 then put in the stack the position of the current vertex
11 draw a left edge from the current vertex
12    DECODINGBT(w)
13 if $ab = 11$
14 then get from the stack the position of the new current vertex
15 draw a right edge from the current vertex
16    DECODINGBT(w)
17 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)
8 A consequence

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

\[
C_{n+1} = 1 + \sum_{k \geq 0} (-1)^k \binom{n-k}{k+1} C_{n-k}.
\] (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)

\[
C_{n+1} = \sum_{i=0}^{n} f(n, i) = f(n, 0) + \sum_{i=1}^{n} f(n, i) = 1 + \sum_{i=0}^{n-1} f(n, n-i)
\]

\[
= 1 + \sum_{i=0}^{n-1} \left( \sum_{k=0}^{n} (-1)^k \binom{i-k}{k} C_{n-k} \right)
\]

\[
= 1 + \sum_{k=0}^{n} (-1)^k C_{n-k} \left( \sum_{i=0}^{n-1} \binom{i-k}{k} \right)
\]

\[
= 1 + \sum_{k=0}^{n} (-1)^k \binom{n-k}{k+1} C_{n-k}.
\]

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. Babeș-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