Generating combinatorial objects by ECO method the Lyndon words case

Vincent VAJNOVSZKI
LE2I
Université de Bourgogne
France

'Nothing is more practical than a good theory'

S. Boltzman

Combinatorics deals with *discrete objects*, i.e., objects that can be finitely descibed by construction rules.

More formally, a *class of combinatorial objects* is an at most denumerable set on which a size function is defined, satisfying the following conditions: the size of an element is a non-negative integer; the number of elements of any given size is finite.

bitstring	subset
0 0 0 0 0	$\emptyset$
0 0 0 0 1	{5}
0 0 0 1 0	{4}
0 0 1 0 0	{3}
0 0 1 0 1	{3,5}
0 1 0 0 0	{2}
0 1 0 0 1	{2,5}
0 1 0 1 0	{2,4}
1 0 0 0 0	{1}
1 0 0 0 1	{1,5}
1 0 0 1 0	{1,4}
1 0 1 0 0	{1,3}
1 0 1 0 1	{1,3,5}

An alternative axiomatic presentation is: a class of combinatorial objects is a pair $(\mathcal{O}, |\cdot|)$ where $\mathcal{O}$ is a finite or denumerable set and the mapping $|\cdot| : \mathcal{O} \rightarrow I\!N$ is such that the inverse image of any integer is finite.

Traditionally, combinatorial objects (the elements of a class) are equipped with a combinatorial significance which is a rather intuitive notion.

Primarily, combinatorics treats in the following problems concerning combinatorial objects:

- enumeration
- exhaustive generation
- Gray codes
- ranking and unranking
- random generation
- search
- combinatorial constructors
- ...

Enumeration

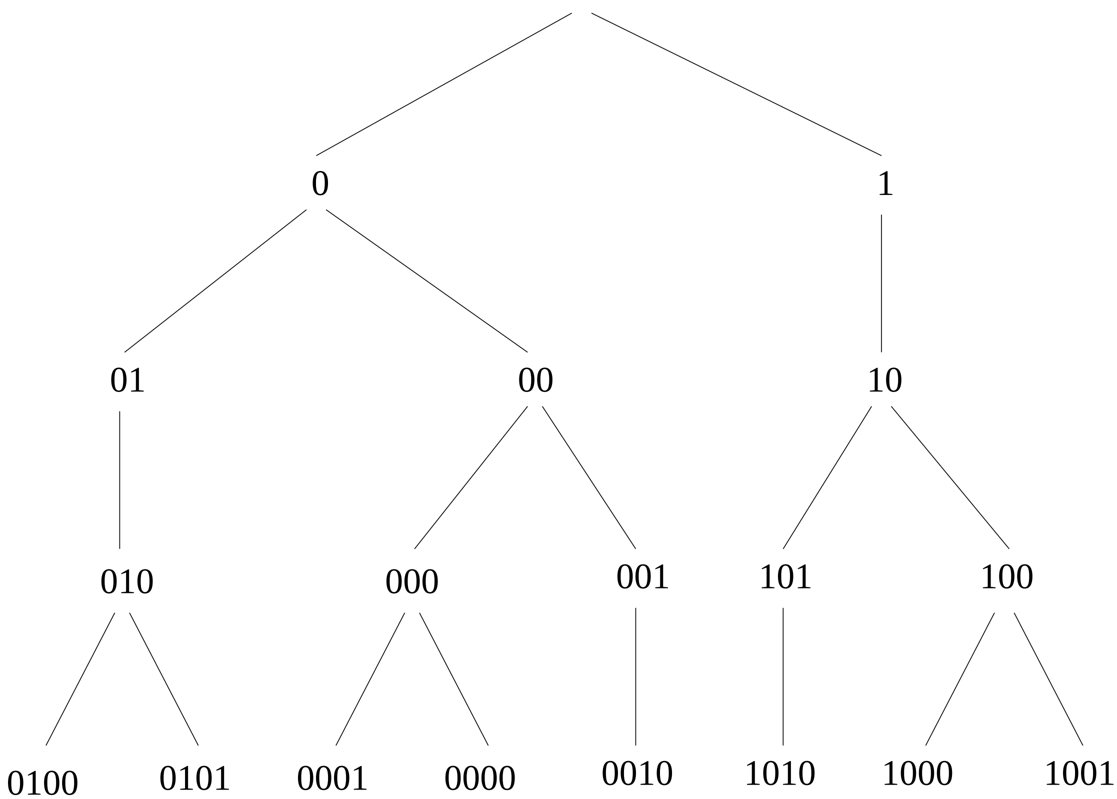
Here the question is to determine the number of combinatorial objects in a given class, and do so for all possible sizes.

$$|\mathcal{O}_n| = ?$$

Exhaustive generation

It consists of listing with no omissions or repetitions all the combinatorial objects with a given size in a particular class.

Exhaustive generation can be used to test hypotheses about a class of objects, solving some NP-complete problems, analysing or proving programs ..., and often an exhaustive generating algorithm exhibits new combinatorial properties of the class being generated.


0
1
01
00
10
010
000
001
101
100
0100
0101
0001
0000
0010
1010
1000
1001

Gray code

If $\mathcal{O}$ is such that successive sequences differ by a number of changes that is bounded independently of the length of the sequences then the list is called a Gray code list (for the combinatorial class).

	bitstring	subset
1	0 1 0 0 $\underline{\mathbf{1}}$	{2,5}
2	0 1 0 $\underline{\mathbf{0}}$ **0**	{2}
3	0 $\underline{\mathbf{1}}$ 0 **1** 0	{2,4}
4	**0** **0** 0 $\underline{\mathbf{1}}$ 0	{4}
5	**0** **0** **0** **0** $\underline{\mathbf{0}}$	$\emptyset$
6	**0** **0** $\underline{\mathbf{0}}$ 0 **1**	{5}
7	**0** 0 **1** 0 $\underline{\mathbf{1}}$	{3,5}
8	$\underline{\mathbf{0}}$ 0 1 0 **0**	{3}
9	**1** 0 1 0 $\underline{\mathbf{0}}$	{1,3}
10	**1** 0 $\underline{\mathbf{1}}$ 0 **1**	{1,3,5}
11	1 0 **0** 0 $\underline{\mathbf{1}}$	{1,5}
12	1 0 **0** $\underline{\mathbf{0}}$ **0**	{1}
13	1 0 0 **1** 0	{1,4}

- solving puzzles: Tower of Hanoi, Brain, Chinese-Rings, Spin-Out
- circuit testing
- analog-digital converters and signal encoding
- data compression
- graphics and image processing
- hashing
- statistics
- Hamiltonian circuits in hypercubes
- Cayley graphs of Coxeter groups
- capanology (the study of bell-ringing)
- continuous space-filling curves
- classification of Venn diagrams
- design of communication codes
- databases
- computable analysis
- experimental design

Random generation

This consists of designing an algorithm that chooses randomly and with uniform probability an object in a certain class. Apart from the theoretical interest, random generation is used principally for producing objects of large size, when exhaustive generation is inefficient.

It has applications in:

- the estimation of the average complexity of algorithms which deal with combinatorial objects;
- verification of combinatorial conjectures and assistance to one intuition (random generation is therefore associated with Monte-Carlo method);
- modeling (in Computer Science, Physics, Statistics)
- etc

In these application areas, one needs to generate objects of very large size, which justifies the desire to achieve very low computational complexities, ideally linear in the size of the object to be constructed.

Search

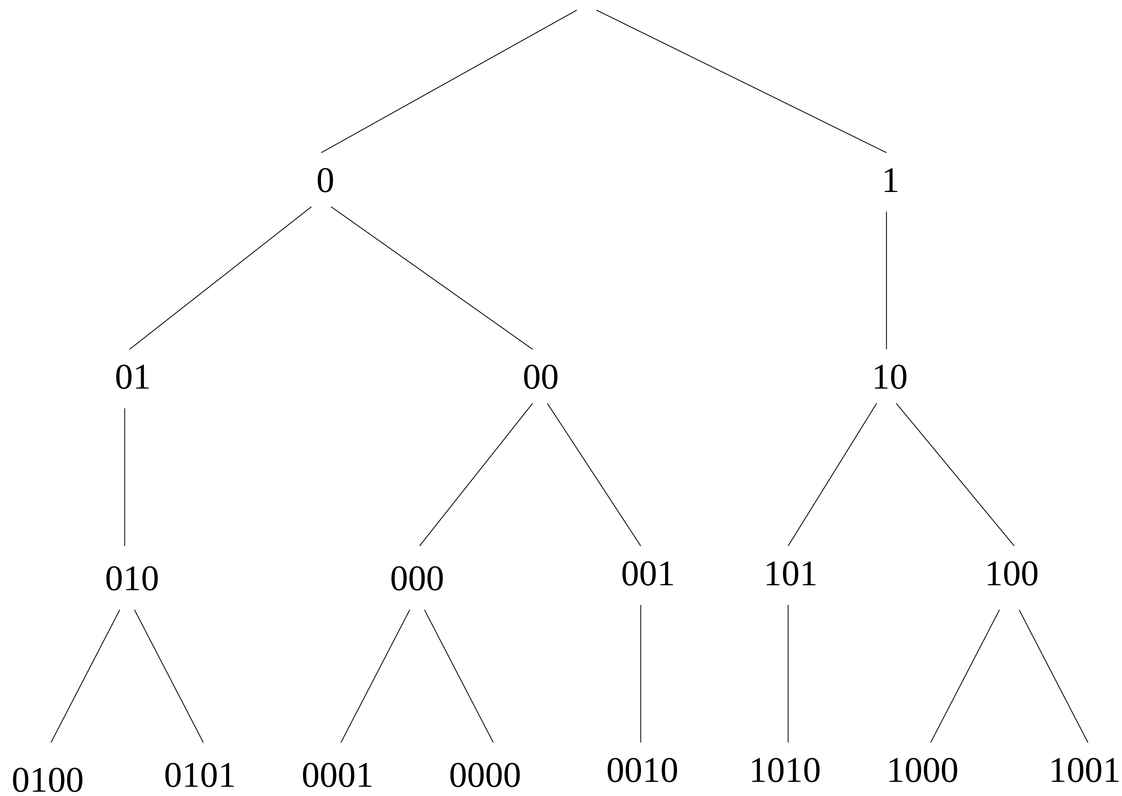
The search is to find one example of an object of a particular class (if it exists). Generally, it is easier to find one example of an object belonging to a class than it is to enumerate the class entirely or generate randomly an object. A variation of a search problem is an optimization problem. Many interesting and important search and optimization problems belong to the class of NP-complete problems.

The ECO method

An operator ϑ on the class $\mathcal{O}$ is a function $\vartheta : \mathcal{O}_n \to 2^{\mathcal{O}_{n+1}}$ with $2^{\mathcal{O}_{n+1}}$ the power set of $\mathcal{O}_n$.
If the operator ϑ satisfies the following conditions:

1. for each $y \in \mathcal{O}_{n+1}$ there exists $x \in \mathcal{O}_n$ such that $y \in \vartheta(x)$
2. if $x_1, x_2 \in \mathcal{O}_n$ and $x_1 \neq x_2$, then $\vartheta(x_1) \cap \vartheta(x_2) = \emptyset$

then $\{\vartheta(x)\}_{x \in \mathcal{O}_n}$ is a partition of $\mathcal{O}_{n+1}$.



Fibonacci

axiom: (2)

rewriting rules:

$(1) \rightsquigarrow (2)$

$(2) \rightsquigarrow (1)(2)$

In addition if

3 for each object $x \in \mathcal{O}_n$ there is an *ordered* list $y_1, y_2, \ldots, y_{|\vartheta(x)|}$ for the set $\vartheta(x) \subset \mathcal{O}_{n+1}$ such that one can compute y_1 from x, y_2 from y_1, $\ldots$, $y_{|\vartheta(x)|}$ from $y_{|\vartheta(x)|-1}$ and back x from $y_{|\vartheta(x)|}$ in time linear on $|\vartheta(x)|$

then ϑ is called *uniform*. In other words on operator $\vartheta(x)$ is uniform if for any n and any $x \in \mathcal{O}_n$ one can compute in constant average time the elements of $\vartheta(x)$ without lose x and with no extra data structures. So, uniformity has an algorithmic meaning. In the following we denote by $\vartheta(x)$ both the set of successors of x and an ordered list for this set.

ECO Inventory

Rational OGF

Fibonacci

axiom: (2)

rewriting rules: $(1) \leadsto (2)$, $(2) \leadsto (1)(2)$

ogf: $\frac{1}{1-z-z^2}$

even Fibonacci

axiom: (2)

rewriting rules: $(k) \leadsto (2)^{k-1}(k+1)$

ogf: $\frac{1-z}{1-3z+z^2}$

odd Fibonacci

axiom: (3)

rewriting rules: $(k) \leadsto (2)^{k-1}(k+1)$

ogf: $\frac{1}{1-3z+z^2}$

Algebraic OGF

Motzkin number

axiom : (1)

rewriting rules: $(k) \rightsquigarrow (1)\ldots(k-1)(k+1)$

ogf: $\frac{1-z-\sqrt{1-2z-3z^2}}{2z^2}$

Catalan number

axiom : (2)

rewriting rules: $(k) \rightsquigarrow (2)\ldots(k)(k+1)$

ogf: $\frac{1-2z-\sqrt{1-4z}}{2z^2}$

Schröder

axiom : (3)

rewriting rules: $(k) \rightsquigarrow (3)\ldots(k)(k+1)^2$

ogf: $\frac{1-3z-\sqrt{1-6z+2z^2}}{4z^2}$

Holonomic transcendental EGF

Permutation

axiom : (1)

rewriting rules: $(k) \rightsquigarrow (k+1)^k$

egf: $\frac{1}{1-z}$

Arrangements

axiom : (1)

rewriting rules: $(k) \rightsquigarrow (k)(k+1)^{k-1}$

egf: $\frac{e^z}{1-z}$

Involutions

axiom : (1)

rewriting rules: $(k) \rightsquigarrow (k-1)^{k-1}(k+1)$

egf: $e^{z+\frac{1}{2}z^2}$

Partial permutations

axiom : (2)

rewriting rules: $(k) \rightsquigarrow (k+1)^{k-1}(k+2)$

egf: $\frac{e^{\frac{z}{1-z}}}{1-z}$

Nonolonomic transcendental EGF

Bell number

axiom : (1)

rewriting rules: $(k) \leadsto (k)^{k-1}(k+1)$

egf: e^{e^z-1}

Bicolored partitions

axiom : (1)

rewriting rules: $(k) \leadsto (k)^{k-2}(k+1)^2$

egf: $e^{2(e^z-1)}$

Necklaces and Lyndon words

The conjugacy class of a string $\boldsymbol{x} = \boldsymbol{uv}$ is the set of all strings $\boldsymbol{vu}$.

- A (binary) *necklace* is a binary string which is minimal with respect to lexicographic order in its conjugacy class
- a *Lyndon word* is an aperiodic necklace
- a *pre-necklace* is a binary string which is the prefix of a some necklace, or equivalently, of some Lyndon word

pre-necklaces	necklaces	Lyndon words
0 1 1 1 1	0 1 1 1 1	0 1 1 1 1
0 1 1 1 0	0 1 0 1 1	0 1 0 1 1
0 1 1 0 1	0 0 0 1 1	0 0 0 1 1
0 1 0 1 0	0 0 0 0 0	0 0 0 0 1
0 1 0 1 1	0 0 0 0 1	0 0 1 0 1
0 0 0 1 1	0 0 1 0 1	0 0 1 1 1
0 0 0 1 0	0 0 1 1 1	
0 0 0 0 0	1 1 1 1 1	
0 0 0 0 1		
0 0 1 0 1		
0 0 1 0 0		
0 0 1 1 0		
0 0 1 1 1		
1 1 1 1 1		

Order relations

- The lexicographical order is defined as: $\boldsymbol{x}$ is less than $\boldsymbol{y}$ iff $x_k = 0$ and $y_k = 1$, where k is the leftmost position with $x_k \neq y_k$; and any proper prefix of a string $\boldsymbol{x}$ is less than $\boldsymbol{x}$.

- The *reflected Gray code order* on $\{0,1\}^n$, due to Frank Gray in 1953, is given by: $\boldsymbol{x} < \boldsymbol{y}$ iff $\sum_{j=1}^{k} x_j$ is even (and $\sum_{j=1}^{k} y_j$ is odd), where k is the leftmost position with $x_k \neq y_k$.

Gray codes

We say that an order relation on a set of strings induces a *k-Gray code* if the set listed in this order yields a k-Gray code.

For example, the reflected Gray code order induces a 1-Gray code on $\{0,1\}^n$; and its restriction to the strings with fixed density (i.e., strings with a constant number of 1's) induces a 2-Gray code, called *revolving door code* by Nijenhuis and Wilf

dual reflected order

$\boldsymbol{x}$ is less than $\boldsymbol{y}$ in *dual reflected order*, denoted by $\boldsymbol{x} \prec \boldsymbol{y}$, if $x_1 x_2 \ldots x_k$, the length k prefix of $\boldsymbol{x}$ contains an odd number of 0's, where k is the leftmost position with $x_k \neq y_k$.

A binary string set $X \subset \{0,1\}^n$ is called absorbent if for any $\boldsymbol{x} \in X$ and any k, $1 \leq k < n$, $x_1 x_2 \ldots x_k 1^{n-k}$ is also a string in X.

Theorems

- The sets of length n Lyndon words, necklaces and pre-necklaces are absorbent.
- If X is an absorbent subset of $\{0,1\}^n$ then X listed in $\prec$ order is a 3-Gray code.
- The $\prec$ order yields a 3-Gray code on the sets of pre-necklaces, necklaces and Lyndon words of length n.

pre-necklaces	necklaces	Lyndon words
0 1 1 1 1	0 1 1 1 1	0 1 1 1 1
0 1 1 1 0	0 1 0 1 1	0 1 0 1 1
0 1 1 0 1	0 0 0 1 1	0 0 0 1 1
0 1 0 1 0	0 0 0 0 0	0 0 0 0 1
0 1 0 1 1	0 0 0 0 1	0 0 1 0 1
0 0 0 1 1	0 0 1 0 1	0 0 1 1 1
0 0 0 1 0	0 0 1 1 1	
0 0 0 0 0	1 1 1 1 1	
0 0 0 0 1		
0 0 1 0 1		
0 0 1 0 0		
0 0 1 1 0		
0 0 1 1 1		
1 1 1 1 1		

Generating algorithm

$$lyn(x_1x_2\dots x_n) =$$
$$\max\{p \,|\, x_1x_2\dots x_p \ \text{ is a Lyndon word}\}$$

Lemma Let $\boldsymbol{x} = x_1x_2\dots x_n$ and $p = lyn(\boldsymbol{x})$

- $\boldsymbol{x}$ is a pre-necklaces iff $\boldsymbol{x} = (x_1x_2\dots x_p)^{\frac{n}{p}}$
- $\boldsymbol{x}$ is a necklaces iff $p|n$
- $\boldsymbol{x}$ is a Lyndon word iff $p = n$

FKM algorithm

procedure $gen\_Lex(t, p)$
if $t > n$ **then** $Print(x)$
else **if** $x[t-p] = 1$
  then $x[t] := a[t-p]$; $gen\_Gray(t+1, p)$;
  else $x[t] := 0$; $gen\_Gray(t+1, p)$
    $x[t] := 1$; $gen\_Gray(t+1, t)$
end

If $Print(a)$ is called when

- $p|n$ then the algorithm lists necklaces
- $p = n$ then the algorithm lists Lyndon words

```
procedure gen_Gray(z, t, p)
if t > n then Print(x)
else  if x[t − p] = 1
      then a[t] := x[t − p]; gen_Gray(z, t + 1, p);
      else  if z is even
            then  x[t] := 0; gen_Gray(z + 1, t + 1, p)
                  x[t] := 1; gen_Gray(z, t + 1, t)
            else  x[t] := 1; gen_Gray(z, t + 1, t)
                  x[t] := 0; gen_Gray(z + 1, t + 1, p)
end
```

procedure $gen\_Gray(z, t, p)$
if $t > n$ **then** $Print(a)$
else **if** $a[t-p] = 1$
  then $a[t] := a[t-p]$; $gen\_Gray(z, t+1, p)$
  else $a[t] := z$; $gen\_Gray(1, t+1, p)$
    $a[t] := 1 - z$; $gen\_Gray(0, t+1, t)$
end.

pre-necklaces	necklaces	Lyndon words
0 1 1 1 1	0 1 1 1 1	0 1 1 1 1
0 1 1 1 0		
0 1 1 0 1		
0 1 0 1 0		
0 1 0 1 1	0 1 0 1 1	0 1 0 1 1
0 0 0 1 1	0 0 0 1 1	0 0 0 1 1
0 0 0 1 0		
0 0 0 0 0	0 0 0 0 0	
0 0 0 0 1	0 0 0 0 1	0 0 0 0 1
0 0 1 0 1	0 0 1 0 1	0 0 1 0 1
0 0 1 0 0		
0 0 1 1 0		
0 0 1 1 1	0 0 1 1 1	0 0 1 1 1
1 1 1 1 1	1 1 1 1 1	

Open problems

- The existence of Gray codes for necklaces or Lyndon words where successive objects differ in at most two positions
- the loopless implementation ($O(1)$ in the worst case) of our algorithm
- the generalization of our results for a k-ary alphabet ($k > 2$)