UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Elaborato d'Esame

PROGETTAZIONE E ANALISI DI ALGORITMI

MASSIMO NOCENTINI

Donatella Merlini
Maria Cecilia Verri

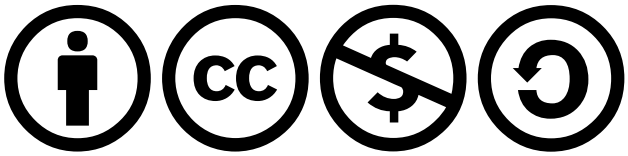
Anno Accademico 2012-2013

CONTENTS

LICENZE

Solo questa sezione dedicata alle licenze verrà scritta completamente in inglese.

Text contents



Sources

Part I

ANALYSIS PART

1

LECTURES NOTES

1.1 BEFORE GENERATING FUNCTIONS

We study two algorithms that are based on checks between keys, those are MERGESORT and QUICKSORT.

1.1.1 *MERGESORT algorithm*

The MERGESORT algorithm is independent from the keys present in the input vector and its methodology behaves always the same. Let $n = 2^m$ be the length of the vector to be ordered, we can define a function C which count the number of checks needed to order the input vector. We define the function C using the method used by the algorithm at each step:

$$C(2^m) = 2C(2^{m-1}) + 2^m$$

Solving the recurrence[1] we obtain $C(n) \in O(n \log n)$. We can observe that if we use a method (like the ones we're studying in this section) based on checks between keys, isn't possible to do better than build a "checks tree", this allow use to have a lower bound for the complexity of those algorithms.[2]

1.1.2 *QUICKSORT algorithm*

The QUICKSORT algorithm depends on the distribution of the keys in the input vector. For what follow we assume to have a probability space $\Omega = D_n$, where D_n is the set of permutation of length n without repetition over $\{1, \dots, n\}$. We focus on the simpler variant where the pivot is chosen as the right-most key[3]. We study the behavior of an application to the vector $(20, 25, 7, 3, 30, 8, 41, 18)$, reporting in Table 1 the steps performed. We can observe that in order to move the `pivot` element in its final position, it is necessary for two keys $(7, 25)$ to be checked twice. Hence, given a vector of length n, the number of checks performed before recurring on left and right partitions is $(n-1)+2$ (where $n-1$ is explained by the `pivot` isn't indexed neither with `i` nor with `j`).

We can analyze the number of checks performed, partitioning it in the following cases:

1 put here the proof
2 in the slides of the first lecture maybe there's more material about this topic
3 report here the code

Table 1: Quicksort example

20	25	7	3	30	8	41	18		
↑ i					↑ j		↑ pivot	→	{20, 41, 8}
8	25	7	3	30	20	41	18		
	↑ i		↑ j				↑ pivot	→	{25, 30, 3}
8	3	7	25	30	20	41	18		
		↑ j	↑ i				↑ pivot	→	{7, 25, 7}
8	3	7	18	30	20	41	25		
			↑ pivot					→	recursion

WORST CASE when the vector is already ordered (in one of the two directions). In this case we got:

$$C(n) = (n-1) + 2 + C(n-1)$$

recurring only on one partition because the other have to be empty. We can expand the recurrence, fixing $C(0) = 0$:

$$\begin{aligned} C(n) &= (n+1) + C(n-1) = (n+1) + n + C(n-2) = \\ &= (n+1) + n + (n-1) + \ldots + 2 + C(0) = \\ &= \sum_{k=2}^{n+1} k + C(0) = \sum_{k=1}^{n+1} k - 1 + C(0) = \frac{(n+1)(n+2)}{2} - 1 \end{aligned}$$

so $C(n) \in O(n^2)$.

BEST CASE when the partition phase puts the pivot in the middle, hence the QUICKSORT recurs on balanced partitions. In this case we have the same complexity of MERGESORT, hence $C(n) \in O(n \log n)$

We explain the average case in a dedicated section.

1.1.3 *QUICKSORT: On the average number of checks*

To study this case we have to consider all elements of Ω (recall that $w \in \Omega \rightarrow (w[i] \in \{1, \ldots, n\}) \wedge (\forall i \neq j : w[i] \neq w[j])$). First of all we can suppose that j is the pivot, hence a generic w will have this structure:

$$w = (C_{j-1} \quad C_{n-j} \quad j)$$

where C_k is a vector of length k. We can consider the probability to have j as $pivot$ considering the uniform distribution on Ω:

$$\mathbb{P}(w \in \Omega : w[n] = j) = \frac{(n-1)!}{n!} = \frac{1}{n}$$

Our goal here is to build a function $C(n)$ which counts the average number of checks during an execution of the algorithm given an input vector of length n. In order to do that observe that every keys $j \in \{1, \ldots, n\}$ can be the $pivot$, we can write:

$$C(n) = (n+1) + \frac{1}{n}\sum_{j=1}^{n} C(j-1) + C(n-j)$$

Observing the sum when j runs:

$$\begin{aligned} j = 1 &\to C(0) + C(n-1) \\ j = 2 &\to C(1) + C(n-2) \\ &\cdots \\ j = n-1 &\to C(n-2) + C(1) \\ j = n &\to C(n-1) + C(0) \end{aligned}$$

Hence we can rewrite:

$$C(n) = (n+1) + \frac{2}{n}\sum_{j=0}^{n-1} C(j)$$

Now we do some manipulation:

$$\begin{aligned} C(n) &= (n+1) + \frac{2}{n}\sum_{j=0}^{n-1} C(j) \\ nC(n) &= n(n+1) + 2\sum_{j=0}^{n-1} C(j) \end{aligned}$$

Subtract the previous $(n-1)$ term to both members:

$$\begin{aligned} nC(n) - (n-1)C(n-1) &= n(n+1) + 2\sum_{j=0}^{n-1} C(j) \\ &\quad - \left((n-1)((n-1)+1) + 2\sum_{j=0}^{(n-1)-1} C(j) \right) \\ &= n(n+1) + 2\sum_{j=0}^{n-1} C(j) \\ &\quad - n(n-1) - 2\sum_{j=0}^{n-2} C(j) \\ &= n(n+1-(n-1)) + 2C(n-1) \\ &= 2(n + C(n-1)) \end{aligned}$$

Getting $nC(n) = 2n + (n+1)C(n-1)$, divide both member by $n(n+1)$:

$$\frac{C(n)}{n+1} = \frac{2}{n+1} + \frac{C(n-1)}{n}$$

We arrived at a recurrence $A(n) = b(n) + A(n-1)$, where $A(n) = \frac{C(n)}{n+1}$ and $b(n) = \frac{2}{n+1}$. So we expand, fixing $C(0) = 0$:

$$\begin{aligned} \frac{C(n)}{n+1} = \frac{2}{n+1} + \frac{C(n-1)}{n} &= \frac{2}{n+1} + \frac{2}{n} + \frac{C(n-2)}{n-1} \\ &= \frac{2}{n+1} + \frac{2}{n} + \ldots + \frac{2}{3} + \frac{2}{2} + \frac{C(0)}{1} \\ &= \frac{2}{n+1} + \frac{2}{n} + \ldots + \frac{2}{3} + 1 \\ &= 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3} \right) + 1 \\ &= 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3} \right) + 2\frac{1}{2} + 2 + 1 - 2\frac{1}{2} - 2 \\ &= 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3} + \frac{1}{2} + 1 \right) - 2 \\ &= 2(H_{n+1} - 1) \end{aligned}$$

Having recognized the harmonic numbers $H_{n+1} = \left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3} + \frac{1}{2} + 1\right)$, the final result is

$$C(n) = 2(n+1)(H_{n+1} - 1)$$

In order to bound $C(n)$ we have to recall that $H_n \sim \ln(n) + \gamma$, hence $C(n) \in O(n \log n)$.

From a practical point of view, to avoid the worst case, when a sorting problem is approached with the QUICKSORT algorithm we can choose to do one of the following actions before starting the sorting process:

- shuffling the input vector and proceed with the algorithm described above;
- choose the $pivot$ element at random, move it in the right-most position and proceed with the algorithm described above.

Each of the two tricks require linear time in the dimension of the input vector and allow us to use the result of the average case and working with $O(n\log n)$ number of checks.

1.1.4 *QUICKSORT: On the average number of swaps*

Our goal here is to build a function $S(n)$ which counts the average number of swaps during an execution of the algorithm given an input vector of length n.

The recurrence for the average number of swaps is as follow:

$$S(n) = \frac{n-2}{6} + \frac{1}{n}\sum_{j=1}^{n} S(j-1) + S(n-j)$$

We proceed the proof in two stages, first studying

$$one : \mathbb{E}\left[K_j\right] = \frac{(j-1)(n-j)}{n-1}$$

where K_j is a random variable that depends on j, after

$$two : \frac{n-2}{6} = \frac{1}{n}\sum_{j=1}^{n} \mathbb{E}\left[K_j\right]$$

In what follow, suppose to have a probability space Ω as defined in the previous sections and an uniform distribution above it.

Proof of one. Again, suppose the $pivot$ is $j \in \{1, \dots, n\}$. We define the random variable $K_j : \Omega \to \mathbb{R}$ such that:

$$w[n] = j \to K_j(w) = s$$
$$w[n] \neq j \to K_j(w) = 0$$

where s is the number of swaps performed by the partitioning phase of QUICKSORT given an input vector w of length n (hence the variable K_j counts the number of swaps). Now we can compute the probability to have k swaps when j is the $pivot$:

$$\mathbb{P}\left(K_j = k\right) = \frac{\binom{n-j}{k}\binom{j-1}{k}(n-j)!(j-1)!}{(n-1)!}$$

We can justify the above formula in the following steps:

- we have k swaps when $|\{w_i : w_i < j\}| = k$ and $|\{w_i : w_i > j\}| = k$
- we can choose k keys from $\{w_i : w_i < j\}$ in $\binom{j-1}{k}$ ways and, for each choice, there are $(j-1)!$ ways to sort them;
- we can choose k keys from $\{w_i : w_i > j\}$ in $\binom{n-j}{k}$ ways and, for each choice, there are $(n-j)!$ ways to sort them;
- the total number of possible permutation of n keys excluding the $pivot$ (which is fixed in the right-most position) is $(n-1)!$

Now we study the mean of the variable K_j:

$$\mathbb{E}\left[K_j\right] = \sum_{k \geqslant 0} k\mathbb{P}\left(K_j = k\right)$$

Using $\binom{n}{m} = \frac{n!}{m!(n-m)!}$, we can rewrite:

$$\mathbb{P}\left(K_j = k\right) = \binom{n-j}{k}\binom{j-1}{k}\frac{(n-j)!(j-1)!}{(n-1)!} = \binom{n-j}{k}\binom{j-1}{k}\binom{n-1}{j-1}^{-1}$$

We put the previous rewrite of $\mathbb{P}\left(K_j = k\right)$ into the definition of $\mathbb{E}\left[K_j\right]$:

$$\mathbb{E}\left[K_j\right] = \sum_{k \geqslant 0} k\mathbb{P}\left(K_j = k\right) = \sum_{k \geqslant 0} k\frac{\binom{n-j}{k}\binom{j-1}{k}}{\binom{n-1}{j-1}}$$

Using the following rewrite for $\binom{j-1}{k}$:

$$\binom{j-1}{k} = \frac{(j-1)!}{k!(j-1-k)!} = \frac{(j-1)}{k}\frac{(j-2)!}{(k-1)!(j-1-k)!} = \frac{(j-1)}{k}\binom{j-2}{k-1}$$

And $\binom{n}{m} = \binom{n}{n-m}$ implies $\binom{j-2}{k-1} = \binom{j-2}{j-2-(k-1)} = \binom{j-2}{j-k-1}$, then:

$$\begin{aligned}\mathbb{E}\left[K_j\right] &= \sum_{k \geqslant 0} k\mathbb{P}\left(K_j = k\right) = \frac{1}{\binom{n-1}{j-1}}\sum_{k \geqslant 0} k\binom{n-j}{k}\binom{j-1}{k} \\ &= \frac{1}{\binom{n-1}{j-1}}\sum_{k \geqslant 0} k\binom{n-j}{k}\frac{j-1}{k}\binom{j-2}{j-k-1} = \frac{j-1}{\binom{n-1}{j-1}}\sum_{k \geqslant 0}\binom{n-j}{k}\binom{j-2}{j-k-1}\end{aligned}$$

Now we recognize the Vandermonde result:

$$\sum_{k \geqslant 0}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$$

which can be proved directly because the ipergeometric distribution has exactly the same structure, and being a distribution, it sum up to 1. So we use this result applying it to $\sum_{k \geqslant 0}\binom{n-j}{k}\binom{j-2}{j-k-1} = \binom{n-2}{j-1}$, obtaining:

$$\begin{aligned}\mathbb{E}\left[K_j\right] &= \frac{j-1}{\binom{n-1}{j-1}}\binom{n-2}{j-1} = \frac{(j-1)(n-2)!(j-1)!(n-j)!}{(n-1)!(j-1)!(n-j-1)!} = \\ &= \frac{(j-1)(n-2)!(j-1)!(n-j)(n-j-1)!}{(n-1)(n-2)!(j-1)!(n-j-1)!} = \frac{(j-1)(n-j)}{n-1}\end{aligned}$$

□

Now we proceed with the second proof:

Proof of two. Let us start with:

$$\frac{1}{n}\sum_{j=1}^{n}\mathbb{E}\left[K_j\right] = \frac{1}{n}\sum_{j=1}^{n}\frac{(j-1)(n-j)}{n-1} = \frac{1}{n(n-1)}\sum_{j=1}^{n}(j-1)(n-j) =$$

$$= \frac{1}{n(n-1)}\sum_{j=1}^{n}(j(1+n)-j^2-n)$$

$$= \frac{1}{n(n-1)}\left((n+1)\sum_{j=1}^{n}j-\sum_{j=1}^{n}j^2-n\sum_{j=1}^{n}1\right)$$

$$= \frac{1}{n(n-1)}\left(\frac{n(n+1)^2}{2}-\frac{n(n+1)(2n+1)}{6}-n^2\right) = \ldots = \frac{n-2}{6}$$

□

We are now ready to solve the main recurrence using the same strategy for the average number of checks, fixing $S(0) = S(1) = S(2) = 0$[4]:

$$nS(n)-(n-1)S(n-1) = \frac{2n-3}{6}+2S(n-1)$$

$$\frac{S(n)}{n+1} = \frac{S(n-1)}{n}+\frac{2n-3}{6n(n+1)} = \sum_{k=3}^{n}\frac{2k-3}{6k(k+1)}$$

1.1.5 *Final consideration*

We've finished our treatment of sorting algorithms based on checks between keys. We've seen theoretical results under a probability space Ω composed by permutations of n objects. This allow us to remark that if we setup a simulation where we apply one algorithm seen above to the entire space Ω with vectors of a given dimension n, then the average number of checks and swaps must be equal to the theoretical results obtained in the previous formulas.

1.2 GENERATING FUNCTIONS BASICS

1.3 ELEMENTS OF COMBINATORICS CALCULUS

1.3.1 *Permutations* P_n

Given a vector $(a_1, \ldots, a_n)$ the number of permutations (or the possible ways to order the elements a_i) is $n!$. A simple proof of that is as follow: for the position

4 put after the displaymath environment the proof that the specified quantity is actually n logn

1 are available n elements, for the second position (after fixing the first) are available $n-1$ elements arriving to the position n where only one element is left, hence $n!$.

It is interesting to show how to add a new element a_{n+1} to existing permutations from the set $\{a_1, \ldots, a_n\}$, this is indeed another proof (by induction) to the number $n!$ justified above. Having the permutations of length n by hypothesis, we study how is it possible to add a_{n+1} into a given permutation $w = (a_1, a_2, \ldots, a_n)$:

$$
\begin{aligned}
&(a_{n+1}, a_1, a_2, \ldots, a_n)\\
&(a_1, a_{n+1}, a_2, \ldots, a_n)\\
&\ldots\\
&(a_1, a_2, \ldots, a_{n+1}, a_n)\\
&(a_1, a_2, \ldots, a_n, a_{n+1})
\end{aligned}
$$

Those new vectors are $n+1$ because in the original permutation $w = (a_1, a_2, \ldots, a_n)$ there are $n-1$ "internal holes" between the consecutive elements a_i and a_{i+1}, plus the left-most and right-most "external holes" $(\square, a_1, ...)$ and $(..., a_n, \square)$. So far we've considered only one permutation w of n elements, repeating the same reasoning for the others w_i (which are $n! - 1$), it must exists $n+1$ new vectors for each of them. So the number of permutation of $n+1$ elements is $(n+1)n! = (n+1)!$ (what we would have expected).

1.3.2 *Dispositions* $D_{n,k}$

A special case of permutations are dispositions, that is the number of different ways in which k elements can be chosen from a set of length n, with $n \geqslant k$. We have n elements available for the first choice, $n-1$ for the second choice arriving to $n-k+1$ elements for the k-th choice, hence $D_{n,k} = n(n-1)\cdots(n-k+1)$. We can write an example to show how to build dispositions. Let $\Omega = \{a, b, c, d\}$ and we want to build $D_{4,2}$, that is the set of ordered pairs, paying attention that $\forall i : (i, i) \notin D_{n,2}$. The following pairs are the candidates (the pairs of the form (i, i) aren't listed because after we choose i for the first position it isn't possible to choose it again for the second):

$$
\begin{array}{ccc}
(a,b) & (a,c) & (a,d)\\
(b,a) & (b,c) & (b,d)\\
(c,a) & (c,b) & (c,d)\\
(d,a) & (d,b) & (d,c)
\end{array}
$$

We've build the previous matrix because we take care about the order in which the pairs are built. In the next section we'll see that there exists another set that doesn't take care of the order.

1.3.3 *Combinations* $C_{n,k}$

A special case of dispositions are combinations. With combinations, notation $C_{n,k}$, we mean a set of sets $\{\{a_{i_1}, \ldots, a_{i_k}\}\}$ without regard to the order in which $a_{i_1}, \ldots, a_{i_k}$ appear. We can see combinations like dispositions modulo the symmetric relation: $(j,i) \in C_{n,2} \rightarrow (i,j) \notin C_{n,2}$ and $(i,i) \notin C_{n,2}, \forall i)$. The following sub matrix is to combinations as the previous one is to dispositions:

$$\begin{matrix} (a,b) & (a,c) & (a,d) \\ & (b,c) & (b,d) \\ & & (c,d) \end{matrix}$$

The combinations matrix is obtained by the disposition matrix, breaking the symmetry, only half of them belongs to the dispositions $C_{4,2}$. Hence $C_{n,k} = \binom{n}{k} = \frac{D_{n,k}}{k!} = \frac{n(n-1)\cdots(n-k+1)}{k!} = \frac{n!}{k!(n-k)!)}$.

1.3.4 *Binomial coefficients*

In this section we study the binomial coefficients, and in the particular the writing $(a+b)^n, n \in \mathbb{N}$. It is known that:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

Now we study $(a+b)^r = \sum_{k=0}^{r} \binom{r}{k} a^k b^{r-k}, r \in \mathbb{R}$, in particular we have to pay attention to the binomial coefficient when $r \in \mathbb{R}$:

$$\binom{r}{k} = \frac{r(r-1)\cdots(r-k+1)}{k!}$$

The previous product make sense, instead we would have had problem with $\frac{r!}{k!(r-k)!}$. Now do simple manipulation:

$$(a+b)^r = \left(a\left(1+\frac{b}{a}\right)\right)^r = a^r\left(1+\frac{b}{a}\right)^r$$

In general we can call $z = \frac{b}{a}$ and reduce to study the function $f(z) = (1+z)^r$, which can be expanded with Taylor:

$$f(z) = \sum_{k \geqslant 0} \frac{f^{(k)}(0)}{k!} z^k$$

Study the derivatives of $f(z)$:

$$
\begin{aligned}
&f^{(0)}(0) = 1 \\
&f^{(1)}(z) = r(1+z)^{r-1} \quad \rightarrow \quad f^{(1)}(0) = r \\
&f^{(2)}(z) = r(r-1)(1+z)^{r-2} \quad \rightarrow \quad f^{(2)}(0) = r(r-1) \\
&f^{(k)}(z) = r(r-1)\cdots(r-k+1)(1+z)^{r-k} \quad \rightarrow \quad f^{(k)}(0) = r(r-1)\cdots(r-k+1)
\end{aligned}
$$

Now we can substitute the previous into $f(z)$:

$$f(z) = \sum_{k\geqslant 0} \frac{f^{(k)}(0)}{k!} z^k = \sum_{k\geqslant 0} \frac{r(r-1)\cdots(r-k+1)}{k!} z^k = \sum_{k\geqslant 0} \binom{r}{k} z^k$$

Exercise 1.3.1. *Compute* $\binom{\frac{1}{2}}{3}$.

Sol. It is sufficient to build a function $f(z) = (1+z)^r$ with the right value for r and then expand $f(z)$ with Taylor, the coefficient of z^3 is the desired value $\binom{r}{3}$.

In this case $r = \frac{1}{2}$, so $f(z) = \sqrt{1+z} = \binom{\frac{1}{2}}{0} z^0 + \binom{\frac{1}{2}}{1} z^1 + \binom{\frac{1}{2}}{2} z^2 + \binom{\frac{1}{2}}{3} z^3 + \dots$ □

Now suppose to have $\binom{-n}{k}, n \in \mathbb{N}$:

$$\binom{-n}{k} = \frac{-n(-n-1)\cdots(-n-k+1)}{k!}$$

At the numerator there are k terms, so factor -1:

$$
\begin{aligned}
\binom{-n}{k} &= \frac{(-1)^k(n+k-1)\cdots(n+1)n}{k!} = \frac{(-1)^k(n+k-1)\cdots(n+1)n(n-1)!}{k!(n-1)!} \\
&= \frac{(-1)^k(n+k-1)!}{k!(n-1)!} = (-1)^k \binom{n+k-1}{k}
\end{aligned}
$$

1.3.5 *Central Binomial Coefficients and Catalan Numbers*

We introduce in the following paragraph a very important class of numbers, the Catalan numbers. We proceed step by step, making an observation on a rewriting for $\binom{n}{k}$ where $n, k \in \mathbb{N}$.

Given a set of n elements $a_1, \dots, a_n$ it is possible to generate the set S of the combinations of length k. Now choose an element, let say a_j, and partition S in two set:

$$
\begin{aligned}
S^+_{a_j} &= \left\{(a_{\pi(1)}, \dots, a_{\pi(k)}) | \exists i : a_j = a_{\pi(i)}\right\} \\
S^-_{a_j} &= \left\{(a_{\pi(1)}, \dots, a_{\pi(k)}) | \forall i : a_j \neq a_{\pi(i)}\right\}
\end{aligned}
$$

where $\pi : \{1, \dots, k\} \to \{1, \dots, n\}$ is one-to-one. The following holds:

$$\binom{n}{k} = \left|S^+_{a_j}\right| + \left|S^-_{a_j}\right| = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Proof. We've chosen a_j to partition S, hence

- $\left|S^+_{a_j}\right| = \binom{n-1}{k-1}$ because $S^+_{a_j}$ contains sets which everyone contains a_j. But we can build $S^+_{a_j}$ from scratch choosing $k-1$ elements (a_j is contained in every set of $S^+_{a_j}$, hence we have to build combinations of length $k-1$) from $\{a_1, \ldots, a_n\} \setminus \{a_j\}$ (a_j is an implicit choice)
- $\left|S^-_{a_j}\right| = \binom{n-1}{k}$ because $S^-_{a_j}$ contains sets which everyone doesn't contains a_j. But we can build $S^-_{a_j}$ from scratch choosing k elements (a_j isn't contained in any set of $S^-_{a_j}$, hence we have to build combinations of length k) from $\{a_1, \ldots, a_n\} \setminus \{a_j\}$ (a_j is an impossible choice)

□

The recurrence $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ allow us to build an infinite matrix reported in Table 2 with interesting numbers written inside a ticket as $\boxed{j}$. The

Table 2: Central Binomial Coefficients

	0	1	2	3	4	5	6	...	k
0	$\boxed{1}$								
1	1	1							
2	1	$\boxed{2}$	1						
3	1	3	3	1					
4	1	4	$\boxed{6}$	4	1				
5	1	5	10	10	5	1			
6	1	6	15	$\boxed{20}$	15	6	1		
⋮								⋱	
⋮								$\binom{n-1}{k-1}$	$\binom{n-1}{k}$
n									$\binom{n}{k}$

numbers $\binom{2h}{h}$ are called *Central Binomial Coefficients*, while $\frac{1}{h+1}\binom{2h}{h}$ are called *Catalan Numbers*.

We finish this section on combinatorics with some exercises.

Exercise 1.3.2. *Compute* $\binom{-\frac{1}{2}}{k}$.

Sol. Using the approach followed in Exercise 1.3.1, we can build the function $f(z) = \frac{1}{\sqrt{1+z}}$ and expand it with Taylor. Here we use another approach that follow the definition:

$$\begin{aligned}
\binom{-\frac{1}{2}}{k} &= \frac{-\frac{1}{2}(-\frac{1}{2}-1)(-\frac{1}{2}-2)\cdots(-\frac{1}{2}-k+1)}{k!} = \\
&= \frac{-\frac{1}{2}(-\frac{3}{2})(-\frac{5}{2})\cdots(-\frac{2k-1}{2})}{k!} = \frac{(-1)^k 135\cdots(2k-1)}{2^k k!} = \\
&= \frac{(-1)^k 135\cdots(2k-1)\quad 246\cdots 2k}{2^k k!\quad 246\cdots 2k} = \frac{(-1)^k (2k)!}{2^k k!\quad 2^k(123\cdots k)} = \\
&= \frac{(-1)^k (2k)!}{2^k k!\quad 2^k k!} = \frac{(-1)^k}{4^k}\binom{2k}{k}
\end{aligned}$$

□

2 SEQUENTIAL SEARCH SIMULATION

We've repeated the simulation for the *sequential search* problem as did during the class. We want to study the mean of checks performed by the sequential search algorithm before the desired element is found in the given permutation. After we compare the mean of checks obtained by simulation with the theoretical mean value for the same input dimension.

The sequential search algorithm cited above is a very simple searching method: it consume a permutation of integer of a fixed dimension n and a target integer; produce true if the target belongs to the given permutation, else otherwise (in our case we augment the information returned with the number of checks needed in the run). The searching strategy consists of starting from the very left and moving one step to right every time the target is missed.

2.1 IMPLEMENTATION

We don't report here the description for the implementation of the sequential search algorithm because is very simple. Instead, we focus the explanation on the main simulation procedure.

The simulation function consume three parameters:

- $numdimensions$, the number of dimensions that we would like to test;
- $interval$, the multiplier to build the permutation vector to be used during the search;
- $attempts$, the number of application of the sequential search algorithm to a given permutation.

We use the number of dimensions $numdimensions$ to build a vector $dimensions$ such that both of the following hold:

$$length(dimensions) = numdimensions$$
$$dimensions[i] = i * interval \quad \forall i \in \{1, \ldots, numdimensions\}$$

For a given dimension $n \in \{interval, 2 * interval, \ldots, numdimensions * interval\}$, we sample a permutation of integers from $\{1, \ldots, n\}$ using the uniform distribution. For each permutation we apply the sequential search algorithm: the number of application is proportional to n (specifically $2n$) and after every application we record the number of checks needed to hit the target. When we run out all the iterations we compute the mean and the variance of the checks, storing them into auxiliary vectors.

2.2 RESULTS

In Table 3 we report a summary table for a simulation invoked with arguments $numdimensions = 50, interval = 20, attempts = 50$. Observing the table we can see that the simulation is quite correct, the theoretical values matches the simulated ones with little differences.

We've used the standardized means to plot them against the normal distribution in order to verify the Central Limit Theorem in Figure 1. The dotted curve represent the normal distribution, the blue one represent the sampling means distribution instead. Observing the plot we can say that the approximation is quite good, however the max dimension is $n = 50$, this allow us to apply the theorem but we could gain precision if we'll repeat our simulation for a greater value of n.

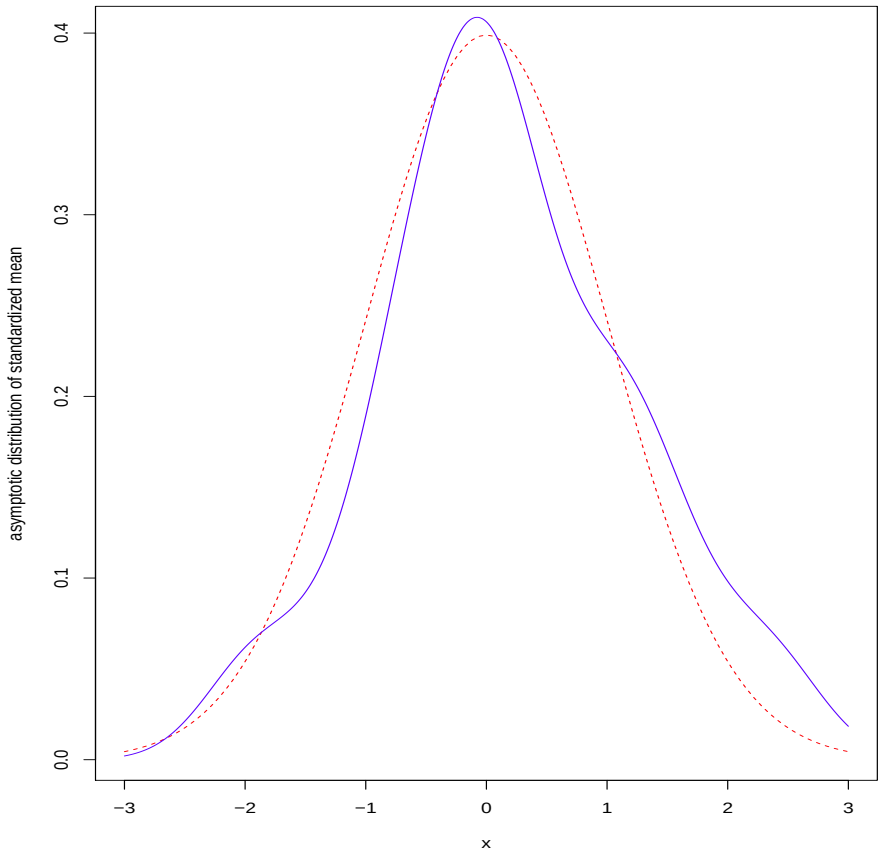


Figure 1: Plot of standardized means

Also we've build a second plot with a regression of the sampling means in Figure 2. The red line is drawn after computing the intercept and the coefficient using an hybrid model. Our attempt to explain the data consists of a mixture up to the second degree, as follow:

$$
\begin{aligned}
dims &= \sum_{i}^{n} dimensions[i] \\
means &= \sum_{i}^{n} mean[i] \\
dim\_squares &= \sum_{i}^{n} dimensions[i]^2 \\
mean\_squares &= \sum_{i}^{n} means[i]^2 \\
dim\_mean &= \sum_{i}^{n} means[i] * dimensions[i] \\
intercept &= \frac{dim\_squares * means - dims * dim\_mean}{n * dim\_squares - dims^2} \\
coefficient &= \frac{n * dim\_mean - dims * means}{n * dim\_squares - dims^2}
\end{aligned}
$$

where n is our $numdimensions$ cited above, hence the red line is defined as $means = coefficient * dimensions + intercept$. Using an R interpreter we can see our numerical output:

```
coefficient = 0.49, intercept = 0.66
square of correlation index = 0.999
```

The regression is almost perfect, hence there exists a strong linear relation between means and distribution (from a statistical point of view we can conclude with the following observation: for an increment of the dimensions of one unit we get an increment of about a half unit on the number of checks).

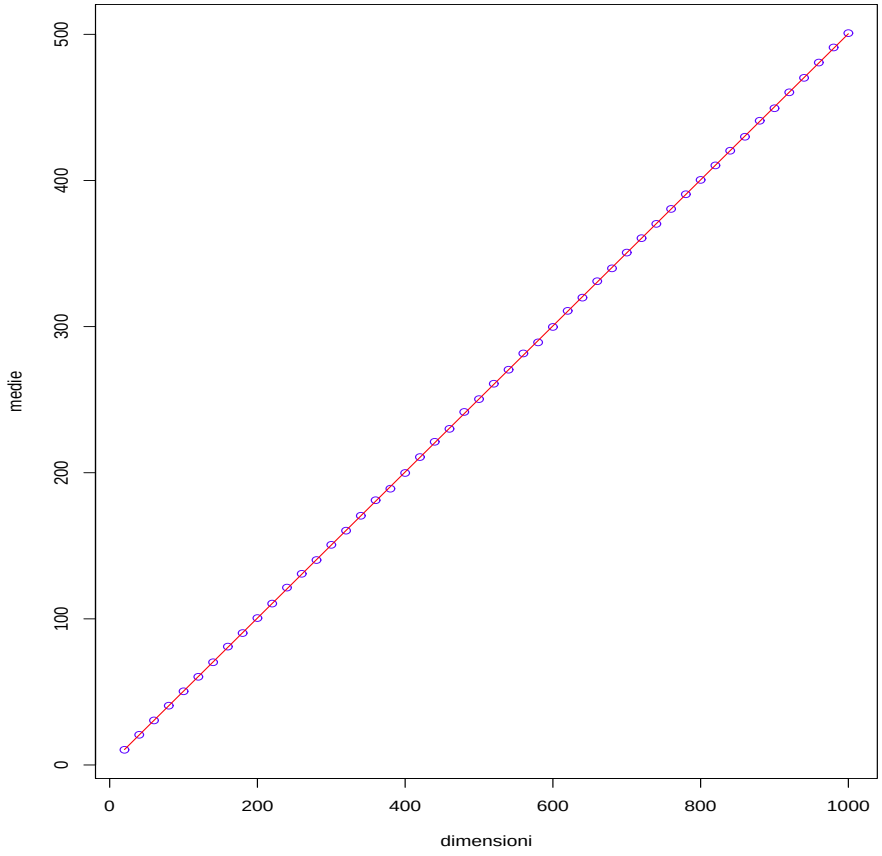


Figure 2: Plot of means regression

Table 3: Sequential search summary

	dimensions	theo means	means	theo vars	vars	theo var of vars	var of vars	stand. means	stand. vars
1	20.00	10.50	10.66	33.25	34.34	877.80	975.55	1.22	1.64
2	40.00	20.50	20.65	133.25	135.02	14177.80	14994.21	0.82	0.94
3	60.00	30.50	30.48	299.92	302.46	71900.02	74572.74	-0.10	0.74
4	80.00	40.50	41.09	533.25	531.28	227377.80	221994.03	2.28	-0.37
5	100.00	50.50	50.52	833.25	833.01	555277.80	553196.84	0.07	-0.03
6	120.00	60.50	60.13	1199.92	1189.70	1151600.02	1122368.13	-1.18	-1.04
7	140.00	70.50	70.42	1633.25	1621.51	2133677.80	2078575.00	-0.22	-0.95
8	160.00	80.50	79.80	2133.25	2155.97	3640177.80	3782358.53	-1.92	1.51
9	180.00	90.50	91.06	2699.92	2696.73	5831100.02	5837985.82	1.44	-0.18
10	200.00	100.50	100.26	3333.25	3315.86	8887777.80	8760281.48	-0.59	-0.82
...									
41	820.00	410.50	409.84	56033.25	56005.31	2511768877.80	2520823155.84	-0.80	-0.16
42	840.00	420.50	419.93	58799.92	59131.53	2765932400.02	2833053962.68	-0.68	1.83
43	860.00	430.50	429.49	61633.25	61801.47	3038913677.80	3061925029.95	-1.20	0.89
44	880.00	440.50	441.59	64533.25	64686.44	3331619377.80	3349812105.46	1.27	0.79
45	900.00	450.50	448.81	67499.92	67476.72	3644977500.02	3620068539.03	-1.95	-0.12
46	920.00	460.50	461.97	70533.25	70360.59	3979937377.80	3963315767.79	1.68	-0.83
47	940.00	470.50	470.75	73633.25	73791.33	4337469677.80	4381479940.27	0.28	0.74
48	960.00	480.50	481.49	76799.92	76684.82	4718566400.02	4704539498.17	1.11	-0.52
49	980.00	490.50	490.41	80033.25	80176.91	5124240877.80	5142289190.74	-0.10	0.63
50	1000.00	500.50	500.43	83333.25	83210.10	5555527777.80	5524755813.88	-0.08	-0.52

3

GENERATING BINARY TREES AT RANDOM

Explain the project and what we want to accomplish.

3.1 ATKINSON AND SACK ALGORITHM

Describe briefly the algorithm here.

3.2 IMPLEMENTATION USING R

```
generate.tree <- function(number_of_nodes){

  word_dimension <- 2 * number_of_nodes

  universe <- 1:word_dimension
  sample <- sample(universe, size=number_of_nodes)
  w = rep(0, word_dimension)
  for (i in 1:word_dimension) {
    w[i] <- ifelse(any(sample == i), 1, -1)
  }

  phi=phi(w)
  list(word=w, phi=phi, as_brackets = brackets_of_word(
      phi))
}

split.word <- function(w){
  if(length(w) == 0){
    return(list(u=c(), v=c()))
  }

  u_index_set <- 1:match(0, cumsum(w))
  list(u=w[u_index_set], v=w[-u_index_set])
}

phi <- function(w){
  if(length(w) == 0){
```

```
    return(w)
  }

  split <- split.word(w)

  if(all(cumsum(split$u) > -1)){
    return (c(split$u, phi(split$v)))
  }
  else{
    t = split$u[-c(1, length(split$u))]
    return (c(1, phi(split$v), -1, -t))
  }
}
```

Part II

ALGORITHMS PART

4

DIVIDE ET IMPERA

Put here some exercises on some topics of interest.

5 APPENDIX

5.1 ORIGINAL ARTICLE ABOUT RANDOM BINARY TREES GENERATION

Information Processing Letters 41 (1992) 21–23
North-Holland

21 January 1992

Generating binary trees at random

M.D. Atkinson and J.-R. Sack

School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6

Communicated by S.G. Akl
Received 21 December 1990
Revised 17 September 1991

Abstract

Atkinson, M.D. and J.-R. Sack, Generating binary trees at random, Information Processing Letters 41 (1992) 21–23.

We give a new constructive proof of the Chung–Feller theorem. Our proof provides a new and simple linear-time algorithm for generating random binary trees on n nodes; the algorithm uses integers no larger than $2n$.

Keywords: Analysis of algorithms, binary trees, bracket sequences, data structures

1. Introduction

Methods for generating binary trees on n nodes have been considered by several authors (see [4,8] and [6] also for additional references). In most cases the focus has been on generating all binary trees in some order or on ranking and unranking them. The number of binary trees on n nodes is the Catalan number $\binom{2n}{n}/(n+1)$ which is exponential in n ($\approx 4^n$) and so these computations cannot be carried out very easily unless n is small. Unranking algorithms allow binary trees to be generated uniformly at random and this is often more useful than being able to list all the possible trees. Unfortunately, numbers which are exponential in n enter these calculations and this makes them impracticable unless n is small.

The problem of generating binary trees uniformly at random without introducing exponentially large numbers was overcome by Arnold and Sleep [1] and Martin and Orr [6]; they gave linear time algorithms which used integers of size $O(n)$ for generating a random binary tree. We shall present a new and simpler solution having the same advantages. Our solution is based on a constructive version of the Chung–Feller theorem on coin-tossing, which has not previously been applied to this area. Our treatment also provides a new proof of the Chung–Feller theorem.

The set of binary trees on n nodes is well known to be in one-to-one correspondence with many other sets of combinatorial objects including rooted (ordered) trees with n branches, triangulations of a convex $(n+2)$-gon, lattice paths from $(0, 0)$ to (n, n) which do not cross the diagonal, and well-formed bracket sequences with n pairs of brackets. The one-to-one correspondences are explicit and efficient to compute in linear time; thus a uniform random generator for binary trees gives rise to a uniform random generator for all these other objects, and vice versa. We shall focus on generating well-formed sequences of brackets.

2. Terminology

We begin with some terminology. It is convenient to denote the left and right bracket symbols by λ and ρ respectively. Thus a bracket sequence (well formed or not) corresponds to a word over

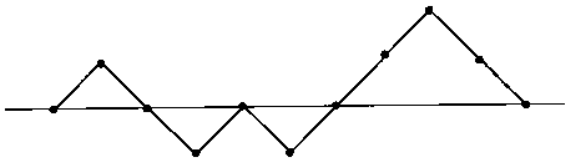


Fig. 1. The zigzag diagram corresponding to the word $\lambda\rho\rho\lambda\rho\lambda\lambda\lambda\rho\rho$.

the alphabet $\{\lambda, \rho\}$. A word such as $\lambda\rho\rho\lambda\rho\lambda\lambda\lambda\rho\rho$ may be pictured as a zigzag diagram drawn from some base line where each upward edge represents λ and each downward edge represents ρ (see Fig. 1).

A word is said to be *balanced* if it contains equal numbers of λ's and ρ's. Balanced words are precisely those whose diagram returns to the base line. A balanced word w is said to be *reducible* if it may be written $w = w_1w_2$ with w_1, w_2 each balanced and nonempty, otherwise w is *irreducible*.

A balanced word is defined to have *defect* i if its diagram has precisely $2i$ links below its base line. Defect 0 words are called *well-formed* and corresponds to well-nested bracket sequences. Observe that the defect of a word is easily found by a summation: we scan the word of length k from left to right regarding each λ as $+1$, each ρ as -1, and computing the partial sums 0, $s_1, \ldots, s_k$; the final sum is zero and the number of negative interim sums s_j at odd indices j is the defect. We call this calculation *partial summation*.

For any word w let w^* denote the result of replacing all occurrences of λ by ρ and ρ by λ. The following two results are immediate consequences of these definitions.

Lemma 1. *If a balanced word w is irreducible, then one of w and w^* is well-formed; in fact, $w = \lambda u\rho$ where u is well-formed, or $w = \rho u\lambda$ where u^* is well-formed. If a balanced word w is well-formed, then $\lambda w\rho$ is irreducible.*

Lemma 2. *A balanced word w has a unique factorisation as $w = w_1w_2 \cdots w_k$, where each w_i is irreducible. If w is well-formed, so is each w_i.*

3. The algorithm

Let B_n denote the set of $\binom{2n}{n}$ balanced words of length $2n$, and let B_{ni} denote the subset of balanced words of defect i. Clearly B_n is the disjoint union of $B_{n0}, B_{n1}, \ldots, B_{nn}$. The Chung–Feller Theorem, see [2, Theorem 2A] and [3, p.94], states that these subsets all have the same size. The central idea of our algorithm is to use a new constructive proof of this theorem which depends on explicit 1–1 correspondences between these sets.

Our algorithm has the following form:

Algorithm RANDOM BRACKET SEQUENCE

Input: An integer n.

Output: A well-formed word of length $2n$ over the alphabet $\{\lambda, \rho\}$.

1. Generate a uniformly random combination L of n integers from $\{1, 2, \ldots, 2n\}$
2. Define a random member $x = (x_1x_2 \cdots x_{2n})$ of B_{2n} by the rule $x_i = \lambda$ if $i \in L$, $x_i = \rho$ if $i \notin L$.
3. Return the well-formed member of B_{2n} to which x corresponds.

Steps 1 and 2 of the algorithm are straightforward. To generate a combination of n integers from $\{1, 2, \ldots, 2n\}$ uniformly at random, and hence a member of B_n, we may use the technique described, for example, in [5, p.137] (see also [7, pp. 189–198] for more discussion on this topic). This technique takes only linear time and uses only integers less than $2n$.

To implement step 3 we need to define some suitable correspondences. We denote by $|w|$ the length of w and by card(S) the cardinality of a set S. We now define a map $\Phi_n : B_n \to B_{n0}$. The definition is inductive. For $n = 0$, we define Φ_0 in the only way possible: it maps the empty string to the empty string. For $n > 0$ and $w \in B_n$ we begin by expressing w as $w = uv$, where u is irreducible, $|u| = r > 0$, $|v| = s \geqslant 0$; then we define Φ_n by the rules

$\Phi_n(w) = u\Phi_s(v)$ if u is well-formed,

$\Phi_n(w) = \lambda\Phi_s(v)\rho t^*$ if $u = \rho t\lambda$ is not well-formed.

Theorem 3. *Φ_n is $n+1$ to 1 onto B_{n0} and is bijective on each B_{ni}.*

Corollary 4. (Chung and Feller) $\text{card}(B_{ni}) = \text{card}(B_{n0})$.

Corollary 5. *If w is a random variable distributed uniformly in B_n, then $\Phi_n(w)$ is distributed uniformly in B_{n0}.*

Proof. It is sufficient to show that Φ_n is a bijection $B_{ni} \to B_{n0}$ for each i. Suppose that $w_1, w_2 \in B_{ni}$ and that they have the same image under Φ_n. For $k = 1,2$ put $w_k = u_k v_k$, where u_k irreducible, and let $|u_k| = r_k$, $|v_k| = s_k$. There are four possibilities:

(1) u_1, u_2 are each well-formed. Then $v_1 \in B_{s_1,i}$, $v_2 \in B_{s_2,i}$, and $u_1\Phi_{s_1}(v_1) = \Phi_n(w_1) = \Phi_n(w_2) = u_2\Phi_{s_2}(v_2)$. By Lemma 2, $u_1 = u_2$, $\Phi_{s_1}(v_1) = \Phi_{s_2}(v_2)$ and so $v_1 = v_2$ by induction.

(2) Neither of u_1 and u_2 are well-formed, say $u_1 = \rho t_1 \lambda$ and $u_2 = \rho t_2 \lambda$. Then $v_1 \in B_{s_1,i-r_1}$, $v_2 \in B_{s_2,i-r_2}$, and $\lambda\Phi_{s_1}(v_1)\rho t_1^* = \lambda\Phi_{s_2}(v_2)\rho t_2^*$. The leading subwords $\lambda\Phi_{s_1}(v_1)\rho t_1^*$ and $\lambda\Phi_{s_2}(v_2)\rho t_2^*$ are irreducible, therefore equal. Therefore, by induction, $v_1 = v_2$ and $t_1 = t_2$, so $u_1 = u_2$.

(3) u_1 is well-formed and u_2 is not well-formed, say $u_2 = \rho t_2 \lambda$. Then $v_1 \in B_{s_1,i}$, $v_2 \in B_{s_2,i-r_2}$, and $u_1\Phi_{s_1}(v_1) = \lambda\Phi_{s_2}(v_2)\rho t_2^*$. By Lemma 2 again, $u_1 = \lambda\Phi_{s_2}(v_2)\rho$ and, taking lengths, $r_1 = s_2 + 2$ and $s_1 = r_2 - 2$. But $r_2 \leqslant i$ since $v_2 \in B_{s_2,i-r_2}$ and $i \leqslant s_1$ since $v_1 \in B_{s_1,i}$ from which it follows that $i \leqslant s_1 \leqslant r_2 - 2 < r_2 \leqslant i$, a contradiction.

(4) u_1 is not well-formed and u_2 is well-formed. This case is impossible for the same reasons as case 3.

This proves that Φ_n is one-to-one on B_{ni}. To prove that it maps B_{ni} onto B_{n0} it is enough to show that these sets have the same size. But we have seen that $\text{card}(B_{ni}) \leqslant \text{card}(B_{n0})$ for each i and if any of these inequalities were strict we would have the contradiction

$$\binom{2n}{n} = \text{card}(B_n) = \sum_{i=0}^{n} \text{card}(B_{ni}) < (n+1)\text{card}(B_{n0}) = \binom{2n}{n}. \qquad \square$$

Lemma 6. *If $w \in B_n$, $\Phi_n(w)$ can be determined in $O(n)$ time.*

Proof. We follow the inductive definition of $\Phi_n(w)$. Let $T(n)$ be the total number of operations required. The decomposition $w = uv$, where u is irreducible, can be found in $O(r)$ steps, where $r = |u|$ by partial summation; the first partial sum equal to zero defines u. Then $\Phi_{n-r}(v)$ must be calculated and so we obtain the recurrence $T(n) = O(r) + T(n-r) = O(n)$. $\square$

Note that the computation of $\Phi_n(w)$ requires no integers larger than $2n$. Thus step 3 of our algorithm can be implemented in linear time, with integers of size $O(n)$, by applying the function Φ_n.

The above discussion provides the proof of the following theorem:

Theorem 7. *Algorithm RANDOM BRACKET SEQUENCE is an unbiased random binary tree generator. It is executes in linear time and uses integers of size $O(n)$.*

Acknowledgment

We acknowledge helpful discussions with D. Horrocks, A. Knight and U. Martin.

References

[1] D.B. Arnold and M.R. Sleep, Uniform random number generation of n balanced parenthesis strings, *ACM Trans. Programming Languages Systems* **2** (1980) 122–128.

[2] K.L. Chung and W. Feller, Fluctuations in coin-tossing, *Proc. Nat. Acad. Sci. U.S.A.* **35** (1949) 605–608.

[3] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1 (Wiley, New York, 3rd ed., 1968).

[4] G.D. Knott, A numbering system for binary trees, *Comm. ACM* **20** (1977) 113–115.

[5] D.E. Knuth, *Semi-numerical Algorithms, The Art of Computer Programming*, Vol. 2 (Addision-Wesley, Reading, MA, 2nd ed., 1981).

[6] H.W. Martin and B.J. Orr, A random binary tree generator, in: *Computing Trends in the 1990's, ACM Seventeenth Computer Science Conf.*, Louisville, KY (ACM, New York, 1989) 33–38.

[7] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice* (Prentice Hall, Englewood Cliffs, NJ, 1977).

[8] M. Solomon and R.A. Finkel, A note on enumerating binary trees, *J. ACM* **27** (1980) 3–5.