A NOTE ON RÉMY'S ALGORITHM FOR GENERATING RANDOM BINARY TREES

Erkki Mäkinen and Jarmo Siltaneva

Abstract. This note discusses the implementation of Rémy's algorithm for generating unbiased random binary trees. We point out an error in a published implementation of the algorithm. The error is found by using the χ^2–test. Moreover, a correct implementation of the algorithm is presented.

1. Introduction. Binary trees are essential in various branches of computer science [5]. From time to time, there is a need to generate random binary trees. For example, when writing a program to manipulate binary trees, it is advantageous to have an efficient method to generate random binary trees with some fixed number n of nodes in order to test the program.

Random generation of binary trees is a special case of random generation of combinatorial structures. A seminal work in this area is the book of Nijenhuis and Wilf [8, 12]. The work of Nijenhuis and Wilf was later generalized by Flajolet et al. [4]. A general method for various kinds of trees was introduced by Alonso et al. [1, 2]. A recent survey on random generation of binary trees appears in [6].

This note discusses an algorithm for generating binary trees introduced by Rémy [10]. We show how the correctness of the implementations of such algorithms can be verified by using the χ^2–test. Moreover, we notice that an implementation by Alonso and Schott [2] does not pass the test, i.e. this implementation is incorrect. We also present a correct implementation. This note is based on the Master's Thesis of the second author [11].

2. Rémy's Algorithm. We suppose familiarity with the basic definitions and notations concerning binary trees as given in [5].

Most algorithms for generating random binary trees are based on coding systems of the trees in question. These algorithms then operate on the sequences of valid code words instead of the actual trees [6]. Rémy's algorithm [10] is an exception to this general approach.

To generate a binary tree with n internal nodes and $n + 1$ leaves, Rémy's algorithm proceeds as follows:

- suppose that so far we have a binary tree with k internal nodes and $k+1$ leaves
- randomly select one of the $2k + 1$ nodes; denote the selected node by v
- replace v by a new node

- choose v to be the left or right child of the new node with probability 1/2 for each of the alternatives; the other child of the new node is a new leaf (the subtrees of v are kept unchanged)
- the process of inserting nodes is repeated until the tree has n internal nodes and $n+1$ leaves.

Notice that Rémy's algorithm is an example of the inductive approach for designing algorithms advocated by Manber [7].

The correctness of Rémy's algorithm can be proved by considering binary trees with leaves labeled by numbers $1, \ldots, n+1$. Namely, it is easy to show that the algorithm generates all binary trees with $2n+1$ labeled leaves with probability $n!/(2n)!$ [2, 10].

3. Verifying the Correctness of Implementations. In order to verify whether or not a given implementation of an algorithm generating random binary trees really produces trees with equal probabilities, we can use the χ^2–test for goodness of fit.

Suppose n is the number of internal nodes in the trees to be generated. There are

$$C_n = \binom{2n}{n} \frac{1}{n+1}$$

different trees with n internal nodes. (C_n is known as the nth Catalan number.) Each of these trees should appear equally likely as the output of the algorithm.

Suppose further that we perform a series of k test runs, and each tree s, $s = 1, \ldots, C_n$, is obtained Y_s times as the output of the algorithm. We compare the squares of differences between the observed numbers Y_s and the expected numbers kp_s, where p_s is the probability to obtain a fixed binary tree, i.e. $p_s = 1/C_n$.

In order to be able to use the standard tables of the χ^2 distribution, we divide each square $(Y_s - kp_s)^2$ by kp_s. Notice that in our case all kp_s's are equal. Hence, we obtain the formula

$$V = \frac{1}{k} \sum_{1 \le s \le C_n} \left(\frac{Y_s^2}{p_s} \right) - k = \frac{C_n}{k} \sum_{1 \le s \le C_n} Y_s^2 - k.$$

In our test runs we have "$C(n)-1$ degrees of freedom" present, i.e. the values $Y_1, Y_2, \ldots, Y_{C(n)-1}$ uniquely define $Y_{C(n)}$. Hence, we use the χ^2 distribution with

$\nu = C(n) - 1$ degrees of freedom. We can now compare the given values of V to the tabulated values.

For example, if we generate binary trees with $n = 4$ internal nodes, the table value 22.36 for $\nu = 13$ and $p = 0.95$ means that $V > 22.36$ only 5 percent of the time provided that the number of test runs k is large enough.

When χ^2 has a very large number of degrees of freedom (as happens in our tests even with reasonably small trees), the values are outside the range of standard χ^2 tables. Then we approximate the χ^2–values by using the normal distribution as explained in [3].

Consider now our test runs with the implementation of Rémy's algorithm presented by Alonso and Schott in [2]. In Table I we give the V–values for binary trees generated by up to 100,000 test runs with $n = 4$ ($C_4 = 14$) and $n = 10$ ($C_{10} = 16,796$). In the test runs reported we used the random number generator suggested in [9].

number of test runs	$V(n = 4)$	$V(n = 10)$
1000	326.61	19955.96
2000	606.10	22086.00
5000	1892.72	29822.31
10,000	3740.75	45100.31
20,000	7100.23	74483.09
50,000	18642.27	154914.59
100,000	35168.11	298598.44

Table I. The V–values related to the results of the Alonso & Schott implementation of Rémy's algorithm.

For example, when $n = 4$ and the number of test runs is 100,000, the "correct" frequency of each possible binary tree is 7142.86. The implementation of Alonso and Schott produced the following sequence of 14 frequencies: 4221 - 4224 - 8296 - 4050 - 4258 - 16,482 -8328 -16,577 - 8437 - 4066 - 4282 - 8272 -4270 - 4237. Hence, the two greatest values are more than twice that of the expected frequencey 7143.

Even without a proper null hypothesis (and hence, by being "worthless statisticians") we can conclude that the above V values for the implementation by Alonso and Schott are "too high". Moreover, they steadily increase when the number of test runs increase. Hence, there must be something wrong with the source code. As a matter of fact, by carefully checking the code we find that the implementation calls the random number generator only once during the insertion of a new node. The order of the children of the new node is not random, and as a consequence all binary trees do not appear equally likely as the output of the implementation.

4. A Correct Implementation of Rémy's Algorithm. In this section we present a correct implementation of Rémy's algorithm and test its correctness by using the χ^2–test.

The following piece of code implements Rémy's algorithm. We use an array *tree* for storing the tree to be generated. Each array element contains pointers to its children and an additional pointer to its parent. The index of the root of the tree is maintained in a variable *root*.

Algorithm.

```
begin
    { the initialization of the tree }
    root := 0;
    tree[0].left_child:=nil;
    tree[0].right_child:=nil;
    tree[0].parent:=nil;
{ the "induction step" }
    for i:=1 to 2n−1 step 2 begin
        hit:= random(0,i−1); { the choice of v }
        direction:= random(0,1); { the choice of the order of
                     the children of the new node }
        parent:=tree[hit].parent;
        if parent = nil
        then root:=i
        else if tree[parent].left_child = hit
            then tree[parent].left_child:=i
            else tree[parent].right_child:=i;
        tree[i].parent:=parent;
```

```
        if direction = 0
        then begin
            tree[i].left_child:=i+1;
            tree[i].right_child:=hit;
            end
        else begin
            tree[i].left_child:=hit;
            tree[i].right_child:=i+1;
            end
        { insertion of the new leaf into the position i+1 }
        tree[hit].parent:=i;
        tree[i+1].left_child:=nil;
        tree[i+1].right_child:=nil;
        tree[i+1].parent:=i;
    end { for }
end; { The Algorithm }
```

The function *random(a,b)* returns a random value of appropriate type from the interval $[a, b]$. In our tests *random* is the random number generator of [9].

For the sake of safety, we perform the χ^2-test for the above implementation (see Table II).

number of test runs	$V(n = 4)$	$V(n = 10)$
1000	21.19	16736.70
2000	14.42	16879.11
5000	12.41	16903.49
10,000	18.81	16758.13
20,000	6.77	16457.53
50,000	8.47	16858.91
100,000	14.28	16984.61

Table II. The V−values related to the correct implementation of Rémy's algorithm.

When $n = 4$ and the number of test runs is 100,000, the sequence of frequencies is now 7393 - 7310 - 7076 - 7203 -7139 - 6971 - 7067 - 7055 - 7101 - 7215 - 7086 - 7132 - 7163 - 7089.

These results show that random binary trees are indeed generated.

5. Concluding Remarks. The χ^2–test revealed that the implementation of Rémy's algorithm presented in [2] is erroneous. A correct implementation of the algorithm is given. Similar tests are naturally possible for other algorithms that randomly generate combinatorial systems.

Acknowledgements. The work of Erkki Mäkinen was supported by the Academy of Finland (Project 35025).

References

1. L. Alonso, J. L. Remy, and R. Schott, "A Linear–Time Algorithm for the Generation of Trees," *Algorithmica* 17 (1997), 162–182.
2. L. Alonso and R. Schott, *Random Generation of Trees,* Kluwer, Boston, MA, 1995.
3. G. M. Clarke and D. Cooke, *A Basic Course in Statistics,* Arnold, London, UK, 1998.
4. P. Flajolet, P. Zimmerman and B. Van Cutsem, "A Calculus for the Random Generation of Labeled Combinatorial Structures," *Theoret. Comput. Sci.* 132 (1994), 1–35.
5. D.E. Knuth, *The Art of Computer Programming. Vol. 1, Fundamental Algorithms,* Third Edition, Addison-Wesley, Reading, MA, 1997.
6. E. Mäkinen, "Generating Random Binary Trees – A Survey," *Inf. Sciences* 115 (1999), 123–136.
7. U. Manber, *Introduction to Algorithms: A Creative Approach,* Addison–Wesley, Reading, MA, 1989.
8. A. Nijenhuis and H.S. Wilf, *Combinatorial Algorithms,* Second Edition, Academic Press, New York, NY, 1978.
9. S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard To Find," *Comm. ACM* 31 (1988), 1192–1201.
10. J. L. Rémy, "Un Procédé Itératif de Dénombrement D'Arbes Binairies et son Application à Leur Génération Aléatoire," *RAIRO Inform. Théor,* 19 (1985), 179–195.

11. J. Siltaneva, *Random Generation of Binary Trees,* (In Finnish), Master's Thesis, Dept. of Computer and Information Sciences, University of Tampere, August 2000.

12. H. S. Wilf, *Combinatorial Algorithms: An Update.* SIAM, Philadelphia, PA, 1989.

Erkki Mäkinen
Department of Computer and Information Sciences
P.O. Box 607
FIN–33014
University of Tampere
Finland
email: em@cs.uta.fi

Jarmo Siltaneva
Information Technology Center
City of Tampere
Lenkkeilijänkatu 8
Finn–Medi 2
FIN–33520
Tampere, Finland
email: Jarmo.Siltaneva@tt.tampere.fi