Combinatorial Enumeration Algorithm of Binary Trees of a Given Height

Jie Tian†, Ninghan Wang‡, Zhiwei Xu‡
†Dept. of Computer Science, New Jersey Institute of Technology, Newark, USA
‡Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R.China
Email: jt66@njit.edu, {xuzhiwei2001, wangninghan}@ict.ac.cn

Abstract—Combinatorial enumeration of binary trees of a given height is a difficult problem. In this work, we propose an effective and efficient algorithm to enumerate the number of distinct binary trees of an arbitrary height as long as the storage space is sufficient. To solve the problem of combinatorial enumeration of binary trees, we use a dynamic programming method and cache the intermediate results of combinatorial enumeration to reduce the temporal and spatial complexity of this algorithm.

Index Terms—Combinatorial enumeration, Distinct binary trees of a given height, Dynamic programming algorithm

I. INTRODUCTION

Trees are the most important nonlinear structures arising in computer science. In this work, we design an enumeration algorithm to solve the problem of calculating the node number of distinct binary trees of a given height. We can easily calculate the node number of distinct binary trees of height 4 in a manual way. However, the total number is exponentially growing while the height of binary trees increases, and thus we design an algorithm to calculate the node number of distinct binary trees of an arbitrary height.

A. Related Works

Algorithms to generate Catalan-counted objects were not invented until computer programmers developed an appetite for them. H. I. Scoins gave two recursive algorithms for ordered tree generation in the first such algorithms to be published. In the same paper, a generation algorithm of oriented trees [1] was also introduced. His algorithms dealt with binary trees represented as bit strings that were essentially equivalent to Polish prefix notation or to nested parentheses. Then Mark Wells, generated binary trees by representing them as noncrossing set partitions. And Gary Knott gave recursive ranking and unranking algorithms for binary trees [2].

B. Problem Description

In the case where the height of binary trees is 1, the node number of distinct binary trees is obviously equal to 1. And as to distinct binary trees of height 2, there are 3 different cases, appending a left child to the root node, appending a right child to the root node and appending two children to the root node as depicted in Figure 1. However, for the binary trees of height 3, the problem becomes a bit complex. We can not simply calculate the number of the node in the level of 3 unless we explicitly determine the number of the nodes in the level 2. In the rest part of this paper, we design an algorithm to solve the problem.

---

Binary Trees of Height 1

Binary Trees of Height 2

Binary Trees of Height 3

---

Fig. 1. Enumeration Process

II. PROBLEM ANALYSIS AND ALGORITHM DESIGN

A. Formulas derivation

If lower level has n nodes, we have \( \sum_{i=1}^{2n} \binom{n}{i} \) choices to add nodes to a particular level of binary trees. We suppose \( F_{L,n} \) is the enumeration result of the distinct binary trees which has n nodes at height L and can be calculated by a recursive formula.

The calculation process of \( F_{L,n} \) is simple: \( F_{1,1} = 1 \)

We can expand the root node in two ways: (1) add two nodes upon the root node or (2) add one node upon the root node. For the first case, \( F_{2,1} = F_{1,1} \times \binom{2}{1} \). For the other case, \( F_{2,2} = F_{1,1} \times \binom{2}{2} \).

As to binary trees of height 3, we should divide the enumeration process into two conditions.

If there is only one node on level 2, we will have \( F_{2,1} \times \binom{2}{1} \) alternatives to expand one node on level 3; If there are two nodes on level 2, we will have \( F_{2,2} \times \binom{2}{2} \) alternatives on level 3.

Therefore, according to the number of the nodes on level 2, we can get \( F_{3,1} = F_{2,1} \times \binom{2}{1} + F_{2,2} \times \binom{2}{1} \). Similarly, \( F_{3,2} = F_{2,1} \times \binom{2}{2} + F_{2,2} \times \binom{2}{2} \), \( F_{3,3} = F_{2,2} \times \binom{4}{3} \) and \( F_{3,4} = F_{2,2} \times \binom{4}{4} \).
Following this recursive process, only if the lower level has 
$[n/2]$ nodes or more, can next level has n nodes. Therefore, we can obtain a recursive formula as following $F_{L,n} = F_{L-1,[n/2]} \times (2 \times (n/2)) + F_{L-1,[n/2]+1} \times (2 \times ((n/2)+1)) + \ldots + F_{L-1,2^{L-2} \times \left(\begin{array}{c} n \\ n \end{array}\right)}$.

In brief, $F_{L,n} = \sum_{i=[n/2]}^{2^{L-2}} F_{L-1,i} \times \left(\begin{array}{c} n \\ i \end{array}\right)$.

Define $E_L$ as the enumeration of the trees at height L. Level L has 1 to $2^L - 1$ nodes. Therefore $E_L = \sum_{i=1}^{2^L - 1} F_{L,n}$.

Obviously, $E_L = F_{1,1} = 1$. So we can gradually calculate all $E_L$ with the formulas above.

To calculate the combinatorial enumeration of binary trees of a given height, we implement this algorithm by 2 sub-procedures: calculating the combination values and calculating all F values.

B. Combination Calculation

We can use the formula $\left(\begin{array}{c} m \\ n \end{array}\right) = \frac{m!}{n!(m-n)!}$ to compute the combination values. In this work, we employ a method based on the recursive formulas $\left(\begin{array}{c} m \\ n \end{array}\right) = \left(\begin{array}{c} m-1 \\ n \end{array}\right) + \left(\begin{array}{c} m-1 \\ n-1 \end{array}\right)$ and $\left(\begin{array}{c} m \\ n \end{array}\right) = \left(\begin{array}{c} m \\ n-1 \end{array}\right)$ to calculate the combination values more effectively and efficiently. Notice that, to get the enumeration values of binary trees of a given height, we need to calculate the combination values repeatedly. Therefore, to improve the entire performance of this algorithm, we pre-calculate and cache all combination values $\left(\begin{array}{c} m \\ n \end{array}\right)$, where $n, m \in \{1, 2^{L-1}\}$. We use a 2D array to store the combination values.

Algorithm 1 Combination Calculation

Input: 
m, n
Output: 
comResult[m][n]
1: comResult[][] = 0
2: Function combination(m, n);
3: if comResult[m][n]! = 0 then
4: return comResult[m][n];
5: end if
6: if comResult[m][m-n]! = 0 then
7: return comResult[m][n];
8: end if
9: if n == 0 or m == n then
10: return 1;
11: end if
12: comResult[m][n] ← combination(m - 1, n) + combination(m - 1, n - 1);
13: return comResult[m][n];
14: FunctionEnd.

C. Enumeration of binary trees

If we want to compute the enumeration values of binary trees of height L, we must have all F values of height L-1. Therefore, we need to first calculate $F_{2,1}$, $F_{2,2}$, and then calculate $F_{3,1}$, $F_{3,2}$, and so forth. Considering we need to use them during the entire calculation, we use a 2D array to store the results of $F_{L,n}$. Ultimately, we add all $F_{L,n}$ values at height L to get $E_L$.

Algorithm 2 Enumeration of next level

Input: lastlevel
Output: Enumeration of new level
1: F[][] = 0
2: Function NextLevel(lastlevel);
3: newlevel ← lastlevel + 1
4: nodes ← 2(newlevel - 1)
5: for j = 1 → nodes/2 do
6: for i = 1 → 2j do
7: $F[\text{newlevel}][i] ← F[\text{newlevel}][i] + F[\text{lastlevel}][j] \times \text{comResult}[2j][i]$
8: end for
9: end for
10: $R = 0$
11: for i = 0 → nodes do
12: $R ← R + F[\text{newlevel}][i]$
13: end for
14: return $R$
15: Function End.

III. TEMPORAL AND SPATIAL COMPLEXITY ANALYSIS

We state a 2D array to store all combination values. At height L, tree have $2^L - 1$ nodes at most and we use a $2^{L-1} \times 2^{L-1}$ array to store the combination values. Therefore, the spatial complexity is O($4^L$). According to formula $\left(\begin{array}{c} m \\ n \end{array}\right) = \left(\begin{array}{c} m-1 \\ n \end{array}\right) + \left(\begin{array}{c} m-1 \\ n-1 \end{array}\right)$, the algorithm computes every combination value once. Therefore, time complexity is O($4^L$).

Enumeration process of binary trees uses the combinations of the same scale, i.e., the space complexity is O($4^L$). The time complexity is mainly decided by the two for-loops. The total operations of these two loops are $2^2 + 2^2 + \ldots + (2^{L-1})^2 = 2^3 + 2^3 + \ldots + (2^{L-2})^2 = 2^{2L-4} + 2^{L-3} + 2^{L-1} + 1 = \frac{2^{2L-4} + 2^{L-3} + 2^{L-1} + 1}{6}$. The total time complexity is O($8^L$).

REFERENCES