

Listing subtrees of a binary tree

東京都立大学 足田輝雄

1. はじめに

ここでは, rooted ordered binary trees とその subtrees を対象とする. subtree としては, もとの tree と root を共有するもののみを考える. k 個の頂点からなる tree や subtree を単に k -tree, k -subtree と書く.

目標は, 1 つの tree と k の値とを与えられて, その tree の k -subtrees をすべてリストアップするアルゴリズムを与えることである.

k -trees をすべて生成するアルゴリズムはいろいろ与えられている [1, 3, 4, 6, 7, 8, 9]. 1 つの tree を与えられて, 頂点の数は無視して, すべての subtrees を生成するアルゴリズムは Ruskey [5] にある.

我々の方針は単純で, 次の 3 点に要約できる.

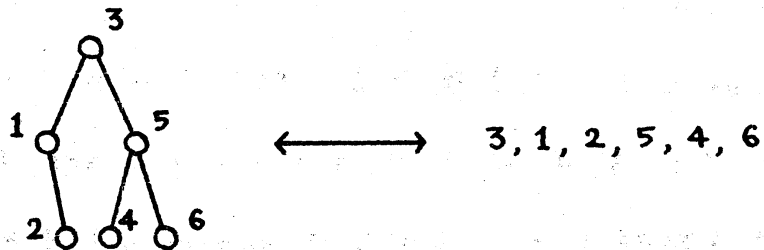
- 1) k -trees を順にすべて生成して, 与えられた tree の中にそれらが埋め込むことができるかどうかテストし, で

さるものをリストアップする。

- 2) その際、1つの k -tree を完全に作り出してから埋め込みをテストするのではなく、作成と埋め込みのテストとを並行的に行なう。つまり、埋め込みない時はそのことができるだけ早くわかるようにする。
- 3) tree の表現方法として tree sequence と称するものを用いる。これにより、 k -tree は、 $1, 2, \dots, k$ を置換した長さ k の列として表現される。

2. tree sequences の生成

tree sequence とは、 k -tree の頂点に $1, 2, \dots, k$ をこの順に inorder でラベル付けして、それらを今度は preorder で読んで並べたものである。



tree sequences の性質はよくわかってゐる。次のような性質が成り立つ。

Property 1. tree sequence とは, $1 \rightarrow$ のスタックを用いて, 列 $1, 2, \dots, k$ に変換できるような列である.

Property 2. a_1, a_2, \dots, a_k が tree sequence

$\iff i < j < m, a_m < a_i < a_j$ となるような i, j, m は存在しない.

Property 3. tree sequences 全体と, ordered binary trees 全体とは 1対1 に対応する.

Remark. Knuth [2] の, 2.2.1 節 Exercise 2 (P238) や, 2.3.1 節 Ex. 6 (P329) の "stack による sequence" は, 我々の tree sequence の逆置換にあたる. 性質は, 従って, ほとんど同様である.

tree sequences (や上の stack sequences) の生成法はすでにいくつかある. 我々の Algorithm 1. は単純なもので, 基本的には backtracking である. 列 a_1, a_2, \dots, a_k を左から順に決めていくが, $1 \rightarrow$ 要素を決めると, その left son と right son それぞれの, 位置と, とり値の範囲とが決まるので, 範囲の下限と上限とを p と q とに記録して, 先へ進む.

Algorithm 1. Tree permutation generation.

Generate all tree permutations of length k lexicographically in an array a . Auxiliary arrays p and q of length k are used, where $p[i]$ and $q[i]$ together designate the range of values $a[i]$ can currently take.

1. (Initialize.)

```

i ← 0;
p[1] ← 1; q[1] ← k;

```

2. (Loop. Proceed one position forward.)

```

i ← i + 1;
a[i] ← p[i];

```

3. (If reached to the right end, output the current content of a , go back to the position where another candidate still exists, and set this candidate to a .)

```

if i = k then
    output(a);
    repeat i ← i - 1; if i = 0 then stop; endif; until a[i] < q[i];
    a[i] ← a[i] + 1;
endif;

```

4. (Set p and q for the left son of $a[i]$, if it exists.)

```

if p[i] < a[i] then
    p[i+1] ← p[i]; q[i+1] ← a[i] - 1;
endif;

```

5. (Set p and q for the right son of $a[i]$, if it exists.)

```

if a[i] < q[i] then
    p[i+a[i]-p[i]+1] ← a[i] + 1; q[i+a[i]-p[i]+1] ← q[i];
endif;

```

6. (Repeat loop.)

```

goto 2.;

```

この方法においても, tree sequences を全部, 可能な
 $\binom{2k}{k} / (k+1)$ 個生成するとき, 1個あたりの平均実行時
 間は定数で押えられる. 証明は略する.

3. k-subtrees のリストアップ

第1節に述べた方針 1), 2), 3) に沿ったアルゴリズムが
 Algorithm 2. である. 説明すべき点はあまりないが, 次のこ
 とを注意しておく.

- 1) Γ の tree sequence において, ある要素の left son は
 (もしあれば) その直後にある.
- 2) Γ の right son はふつう離れた所にある.
- ii) そこで Algorithm 2. では,
 - i) 与えられた tree b の中での, 各頂点の right son の
 位置,
 - ii) 生成する k-tree a の中での, (right) son の
 father の位置,

これらの情報を, a か b に埋め込めることができるかど
 うかテストしていく際に必要とする. i), ii) 以外の情報
 は不要である. なお, i) は, b からのプロセスによ
 り, あらかじめ用意されているものとする.

Algorithm 2. Listing all subtrees on k nodes of a binary tree.

This lists all subtrees on k nodes of a given rooted ordered binary tree on n nodes. Subtrees are assumed to share the root with the given tree.

Input: The value k , and arrays b and r of length n .

The tree should be given as a tree permutation in b , and also an array r should be given, where $r[j]$ is the index in b of the right son of $b[j]$ if it exists and 0 otherwise. Subtrees are generated in lexicographical order in an array a of length k as tree permutations.

Auxiliary arrays f , p and q of length k are used, where $f[i]$ is the index in a of the father node of $a[i]$ (used only when $a[i]$ is the right son), and p and q are as in Algorithm 1. An array m of length k is also used, where $m[i]$ indicates the index in b at which $a[i]$ currently matches.

1. (Initialize.)

```

i ← 0;
p[1] ← 1; q[1] ← k;

```

2. (Loop. Proceed one position forward.)

```

i ← i + 1;
a[i] ← p[i];

```

3. (Examine whether $a[i]$ matches to some node in b , and return the result in a Boolean variable $match$ and $m[i]$.)

```

if i = 1 then                                (Case 1. a[i] is the root.)
    match ← true; m[1] ← 1;
elseif a[i-1] > a[i] then                    (Case 2. a[i] is the left son of a[i-1].)
    if m[i-1] = n then match ← false;
    elseif b[m[i-1]] < b[m[i-1]+1] then match ← false;
    else match ← true; m[i] ← m[i-1] + 1;
    endif;
else                                           (Case 3. a[i] is the right son of a[f[i]].)
    if r[m[f[i]]] = 0 then match ← false;
    else match ← true; m[i] ← r[m[f[i]]];
    endif;
endif;

```

4. (If no node in b matches to a[i], find another candidate.)

if not match then

repeat $i \leftarrow i - 1$; if $i = 0$ then stop; endif; until $a[i] < q[i]$;

$a[i] \leftarrow a[i] + 1$;

goto 3.;

endif;

5. (If reached to the right end, output a, and find another candidate.)

if $i = k$ then

output(a);

repeat $i \leftarrow i - 1$; if $i = 0$ then stop; endif; until $a[i] < q[i]$;

$a[i] \leftarrow a[i] + 1$;

endif;

6. (Set p and q for the left son of a[i], if it exists.)

if $p[i] < a[i]$ then

$p[i+1] \leftarrow p[i]$; $q[i+1] \leftarrow a[i] - 1$;

endif;

7. (Set f, p and q for the right son of a[i], if it exists.)

if $a[i] < q[i]$ then

$f[i+a[i]-p[i]+1] \leftarrow i$;

$p[i+a[i]-p[i]+1] \leftarrow a[i] + 1$; $q[i+a[i]-p[i]+1] \leftarrow q[i]$;

endif;

8. (Repeat loop.)

goto 2.;

4. 高さ h の k -trees の個数

Algorithm 2. の応用として, 高さ h の complete binary tree (頂点数 $n = 2^{h+1} - 1$) の k -subtrees の個数を調べることにより, 高さが h 以下の k -trees の個数がわかる. これから, 高さがちょうど h の k -trees の個数がわかる. 結果は Table 1. の通りである.

h	0	1	2	3	4	5	6	7	8	9	10	11	12	total
1	1													1
2	0	2												2
3	0	1	4											5
4	0	0	6	8										14
5	0	0	6	20	16									42
6	0	0	4	40	56	32								132
7	0	0	1	68	152	144	64							429
8	0	0	0	94	376	480	352	128						1430
9	0	0	0	114	844	1440	1376	832	256					4862
10	0	0	0	116	1744	4056	4736	3712	1920	512				16796
11	0	0	0	94	3340	10856	15248	14272	9600	4352	1024			58786
12	0	0	0	60	5976	27672	47104	50784	40576	24064	9728	2048		208012
13	0	0	0	28	10040	67616	140640	172640	156864	110592	58880	21504	4096	742900

Table 1. The numbers of rooted ordered binary trees on k nodes of height h , for $1 \leq k \leq 13$.

References

- /1/ G. D. Knott, A numbering system for binary trees, *Comm. ACM* 20 (1977) 113-115.
- /2/ D. E. Knuth, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms* (Addison-Wesley, Reading, Mass., 1968, Second ed., 1973).
- /3/ A. Proskurowski, On the generation of binary trees, *J. ACM* 27 (1980) 1-2.
- /4/ D. Rotem and Y. L. Varol, Generation of binary trees from ballot sequences, *J. ACM* 25 (1978) 396-404.
- /5/ F. Ruskey, Listing and counting subtrees of a tree, *SIAM J. Comput.* 10 (1981) 141-150.
- /6/ F. Ruskey and T. C. Hu, Generating binary trees lexicographically, *SIAM J. Comput.* 6 (1977) 745-758.
- /7/ I. Semba, Generation of stack sequences in lexicographical order, *J. Inform. Process.*, to appear.
- /8/ M. Solomon and R. A. Finkel, A note on enumerating binary trees, *J. ACM* 27 (1980) 3-5.
- /9/ A. E. Trojanowski, Ranking and listing algorithms for k-ary trees, *SIAM J. Comput.* 7 (1978) 492-509.