Title	高々2回の交換によるカッコ列の高速生成法 (アルゴリズムと計算の理論)
Author(s)	三河, 賢治; 仙波, 一郎
Citation	数理解析研究所講究録 (1998), 1041: 235-240
Issue Date	1998-04
URL	http://hdl.handle.net/2433/62048
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
1041 巻 1998 年 235-240

高々 2 回の交換によるカッコ列の高速生成法

三河 賢治 (Kenji Mikawa)　仙波 一郎 (Ichiro Semba)
茨城大学工学部情報工学科
E-mail:{mikawa,isemba}@cis.ibaraki.ac.jp

Abstract

n 組の整合したカッコ列は, n 個の内部節点を持つ二分木と一対一対応の関係があり, その生成法について様々な議論がなされている. 近年 (1997), 三河と高岡によって, 1 組のカッコだけを交換してすべてのカッコ列を生成する方法が提案された. 本論文では, 高々 2 組のカッコを交換してすべてのカッコ列を生成する方法を提案する. また実際に計算機による実行結果を与え, 三河と高岡のアルゴリズムよりも高速にカッコ列を生成することを実証する.

1 はじめに

n 組の整合したカッコ列とは, 左カッコ, 右カッコがそれぞれ n 個ずつ存在し, また左から順にカッコ列を並べたとき右カッコの総数は左カッコの総数を超えないものをいう. このようなカッコ列を生成する問題は古くから議論されており, 様々なアルゴリズムが提案されてきた [4, 7, 3]. Ruskey と Proskurowski は偶数組のカッコ列に対し, 隣り合う 2 組のカッコを交換することによってすべてのカッコ列を生成することができることを証明し, 後の論文でそのアルゴリズムを発表した [2, 5]. 近年, 三河と高岡によって正整数 n に対して 1 組のカッコだけを交換してすべてのカッコ列を生成する方法が提案された [1]. この方法は最悪でも $O(1)$ の時間でひとつひとつのカッコ列を生成するが, 交換されるカッコの位置の計算が複雑であり, 実際に計算機に実装して実行時間を比べると他の理論上 $O(n)$ (または平均 $O(1)$) のアルゴリズムよりも遅い場合があった. 本論文では以下の方法に従って, より簡素なアルゴリズムの構築を目指してカッコ列の高速生成を可能とする. (1) それぞれのカッコ列に対して親子関係を与え, (2) その関係を表す木構造を構築し, (3) その木を前順走査することによってカッコ列を順に取り出す. こうして得られたカッコ列を比較すると, 隣り合うどの二つのカッコ列も高々 2 組のカッコだけが異なるように並ぶ. 本論文では, カッコ列を出力する際に必要な手間を考慮しない.

2 定義

まずはじめに長さ $2n$ のカッコ列 X に対して, 以下のように二つの指標を与える [6].

$$p(X) = \max_{n \le i \le 2n} \{i | X_i = \text{`('}\},$$
$$q(X) = \max_{2 \le i < p(X)} \{i | X_i = \text{`)'}\}.$$

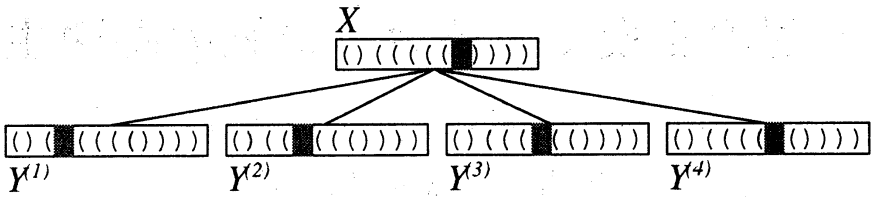

図 1: $X = ()(((((())))))$, $p(X) = 7$, $q(X) = 2$.

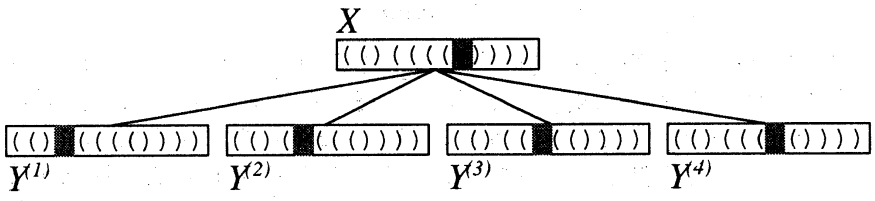

図 2: $X = (()((((()))))$, $p(X) = 7$, $q(X) = 3$.

$p(X)$ は X の中で一番右にある左カッコの位置を示し, $q(X)$ は $X_1 \cdots X_{p(X)}$ の中で一番右にある右カッコの位置をしめす. 便宜的に, $X = (^n)^n$ については $q(X) = 0$ とする. この二つの指標はカッコ列を一意に定めるものではなく, 異なる二つのカッコ列について同じ p, q の組を持つものが存在する場合がある.

次にカッコ列の親子関係を定義する. X と Y をそれぞれ長さ $2n$ のカッコ列とする. 以下のようにして X から Y が作られるとき, Y は X の**子**と呼び, X を Y の**親**と呼ぶ. X に対して, 右カッコ $X_{p(X)+1}$ と $X_{q(X)+1}$ から $X_{p(X)}$ までの左カッコの 1 つを交換して得られるカッコ列を Y とする. $X_1 \cdots X_{q(X)}$ が整合しているならば $X_{p(X)+1}$ と $X_{q(X)+1}$ を交換することはできない (整合性がなくなる) ので, X から $p(X) - q(X) - 1$ 個の子が作られ, そうでなければ $p(X) - q(X)$ 個の子が作られる (図 1, 2 参照).

長さ $2n$ のカッコ列の親子関係を表す木構造を T_n として, その根は深さ 0 であるとしよう. T_n の根に $(^n)^n$ を割り当てて, 他の節点には親子関係を満たすように任意のカッコ列を割り当てる. 子の節点に対応するカッコ列が複数存在する場合は, それらの q の小さい順に左から並べる. 例として, 4 組のカッコ列に対応する T_4 を図 3 に示す. 親子関係に関する補題を以下に与える.

補題 2.1 長さ $2n$ のカッコ列 X は, 深さ i に m 個の子, $Y^{(1)}, \cdots, Y^{(m)}$, を持っていると仮定する. このとき,

(1) $q(Y^{(1)}) = \begin{cases} q(X) + 2 = 2i & X_1 \cdots X_{q(X)} \text{ が整合しているとき,} \\ q(X) + 1 & \text{整合していないとき.} \end{cases}$

(2) $q(Y^{(j)}) = q(Y^{(j-1)}) + 1 \qquad (1 < j \leq m).$

(3) $p(Y^{(j)}) = p(X) + 1 = n + i \qquad (1 \leq j \leq m).$

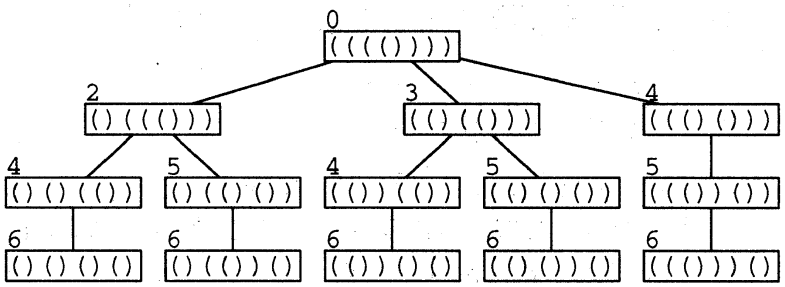

図 3: T_4. 添字は対応するカッコ列の q を示す.

補題 2.1 より T_n 上のどの二つのカッコ列も互いに異なり, また長さ $2n$ の任意のカッコ列に対して, 子を生成するのとは反対の手順を行えば $(^n)^n$ に戻るので, すべてのカッコ列は T_n のいづれかの節点に対応する. このことから T_n によるカッコ列の生成は, 長さ $2n$ のカッコ列をすべて生成することを保証する.

与えられた T_n を前順走査して得られるカッコ列の系列を比べると, 隣り合うどの二つのカッコ列も高々 2 組のカッコだけが異なるように並ぶ. また T_n を後順走査して得られるカッコ列の系列は辞書式順序に並ぶ. この節の終わりに, T_n によるカッコの交換に関わる二つの補題を与える.

補題 2.2 X を T_n の内部節点に対応するカッコ列, Y を前順走査によって X の次に得られるカッコ列とする. このとき Y は, $X_{q(Y)}$ と $X_{p(Y)}$ を交換することによって X から生成される.

補題 2.3 T_n の葉に対応するカッコ列を X, 前順走査によって X の次に得られるカッコ列を Y とする. また Y の左隣りの節点に対応するカッコ列を Z とする. このとき, (1) Z は, $X_{p(X)}$ と $X_{p(Z)}$ を交換することによって X から生成される. (2) Y は, $Z_{q(Z)}$ と $Z_{q(Y)}$ を交換することによって Z から生成される.

3 生成アルゴリズム

前節で構成された T_n を前順走査するだけでよいのだが, ここでは T_n の特性からより高速なアルゴリズムを構築するための議論をしよう. まずはじめに補題 2.1 から導かれる T_n の特性を列挙する. (1) T_n の根から葉までの径路の長さはそれぞれ $n-1$ である. (2) ある節点の一番右の子に対応するカッコ列 X の指標 $q(X)$ は $p(X)-1$ である. (3) ある節点の一番右の子は葉までの径路が唯一である.

T_n 上の深さ i の節点を v, 対応するカッコ列を X, また, v の次にたどる節点を v', 対応するカッコ列を X' としよう. T_n の特性から, 生成アルゴリズムは以下の二つに分けて考えることができる. すなわち, v がある節点の一番右の子である場合とそうではない場合である.

v がある節点の一番右の子である場合, $p(X')$, $q(X')$ はそれぞれ

$$\begin{aligned} p(X') &= p(X)+1, \\ q(X') &= q(X)+1 \quad = \quad p(X')-1 \end{aligned}$$

で求めることができる. また補題 2.2 より, X' は, $X_{p(X')-1}$ と $X_{p(X')}$ を交換して生成される. T_n の葉に対応するカッコ列の指標 p は $2n-1$ であるから, v から葉に至るカッコ列を生成するアルゴリズムは以下に示すとおりである.

A1 **if** $q_i = p-1$ **then begin** { v は一番右の子であるか }
A2 **while** $p < 2n-1$ **do begin** { 葉に至るまでカッコ列を生成する }
A3 $p \leftarrow p+1$; $X_{p-1} \leftrightarrow X_p$; output (X)
A4 **end**;
A5 $i \leftarrow i-1$; $p \leftarrow i+n$ { v の親に節点の深さを戻す }
A6 **end**

次に, v がある節点の一番右の子ではない場合, $p(X')$, $q(X')$, およびカッコの交換位置はそれぞれ, 補題 2.1, 補題 2.2 をそのまま適用すればよい. すなわち, v から葉に至るカッコ列を生成するアルゴリズムは以下に示すとおりである.

B1 **while** $i < n-2$ **do begin** { 深さ $n-2$ の節点に至るまでカッコ列を生成する }
B2 $i \leftarrow i+1$; $p \leftarrow p+1$;
B3 **if** $2i < q_{i-1}+1$ **then** $q_i \leftarrow q_{i-1}+1$ **else** $q_i \leftarrow 2i$;
B4 $X_{q_i} \leftrightarrow X_p$; output (X)
B5 **end**;
B6 $X_{2n-2} \leftrightarrow X_{2n-1}$; output (X) { 葉に対応するカッコ列を生成する }

どちらの場合もアルゴリズムを終了したとき, i は補題 2.3 の交換に対応する節点の深さを示す. 最後に, 完全な生成アルゴリズムを図 4 に与える.

4 アルゴリズムの解析と評価

この節では, 前節のアルゴリズムによってカッコ列の平均交換回数がある定数以下となることを示す. カッコを交換する手順は二つに分けられる. ひとつは, T_n の葉に対応するカッコ列から次のカッコ列を生成するときは 2 組のカッコを交換する. ふたつは, T_n の内部節点に対応するカッコ列から次のカッコ列を生成するときは 1 組のカッコを交換する. カタラン数を $C_n = (2n)!/(n!(n+1)!)$ としよう. 長さ $2n$ のカッコ列の総数は C_n となることが知られている. T_n の葉に対応するカッコ列は, 補題 2.1 からいずれも $S_{2n-2}()$ となる. ここで S_{2n-2} は長さ $2n-2$ のカッコ列である. T_n の葉の総数は C_{n-1} となることは容易に検証できよう. よって長さ $2n$ のカッコ列の平均交換回数を I_n とすると

$$\begin{aligned} I_n &= \frac{C_n + C_{n-1}}{C_n} \quad = \quad 1 + \frac{n+1}{4n-2} \\ &< \frac{7}{5} \end{aligned}$$

```
program GENERATION (input, output);
  var i : integer;
begin
  i ← 0; p ← n; q_0 ← 0;
  repeat
    output (X);
    if q_i = p − 1 then begin
      while p < 2n − 1 do begin
        p ← p + 1; X_{p−1} ↔ X_p; output (X)
      end;
      i ← i − 1; p ← i + n
    end else begin
      while i < n − 2 do begin
        i ← i + 1; p ← p + 1;
        if 2i < q_{i−1} + 1 then q_i ← q_{i−1} + 1 else q_i ← 2i;
        X_{q_i} ↔ X_p; output (X)
      end;
      X_{2n−2} ↔ X_{2n−1}; output (X)
    end;
    X_{2n−1} ↔ X_p; X_{q_i} ↔ X_{q_i+1}; q_i ← q_i + 1
  until i ≤ 0
end.
```

図 4: 生成アルゴリズム

を得る.

最後に, 計算機に実装して実行時間を比べてみよう. 比較されるアルゴリズムは以下の通り (表 1 参照), また使用した計算機は SunSPARK station 20 である.

ALG 1 は仙波のアルゴリズム [6] で, 辞書順にカッコ列を生成する. ALG 2 は Ruskey と Proskurowski のアルゴリズム [5] で, カッコ列を 1 組のカッコが異なるように生成する. このアルゴリズムは再帰形式であるが, n が偶数のとき隣接する 2 つのカッコを交換してすべてのカッコ列を生成するアルゴリズム [2] よりも高速である. ALG 3 は三河と高岡のアルゴリズム [1] で, カッコ列を 1 組のカッコが異なるように生成する. ALG 4 は図 4 に示すアルゴリズムである. ALG 5 は, ALG 4 の改良アルゴリズムである. ALG 4 では, T_n のある内部節点が一番右の子であるかどうか場合分けをしてカッコ列を生成したが, ALG 5 では, T_n のある内部節点が一番右の子の左兄弟であるかどうか場合分けをしてカッコ列を生成する. 計算機による実行結果を表 2 に示す.

表 1: 比較される 5 つのアルゴリズム

	ALG1	ALG2	ALG3	ALG4	ALG5
生成順序	辞書順	その他	その他	その他	その他
形式	反復	再帰	反復	反復	反復
最悪	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$
平均	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
交換回数	3.6	– –	1.0	1.4	1.4

表 2: 実行結果. (単位:ミリ秒)

n	ALG1	ALG2	ALG3	ALG4	ALG5
13	960	1480	1740	770	610
14	3470	5320	6280	2780	2200
15	12550	21480	22750	10070	7950
16	45700	73320	82520	36620	28960

参考文献

[1] K. Mikawa and T. Takaoka, Generation of parenthesis strings by transpositions, in: *Proc. the Computing: the Australasian Theory Symposium, Sydney* (1997) 51–58.

[2] A. Proskurowski and F. Ruskey, Binary tree Gray code, *J. Algorithms* **6** (1985) 239–252.

[3] D. Roelants van Baronaigien, A loopless algorithm for generating binary tree sequences, *Inform. Process. Lett.* **39** (1991) 189–194.

[4] F. Ruskey and T. C. Hu, Generating binary trees lexicographically, *SIAM J. Comput.* **6** (1977) 745–758.

[5] F. Ruskey and A. Proskurowski, Generating binary trees by transpositions, *J. Algorithms* **11** (1990) 68–84.

[6] I. Semba, Generation of all the balanced parenthesis strings in lexicographical order, *Inform. Process. Lett.* **12** (1981) 188–192.

[7] S. Zaks, Lexicographic generation of ordered trees, *Theoret. Comput. Sci.* **10** (1980) 63–82.