情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-AL-126 No.2
2009/9/15

グラフ的列の列挙

菊地洋右[†1]　山中克久[†2]　中野眞一[†3]

単純グラフ G の各点の次数を降順に並べたものを G の次数列という。与えられた整数列 D がある単純グラフの次数列であるとき D はグラフ的列である。本文は、高々 n 個の整数からなるグラフ的列を、重複も抜けもれなく、列挙するアルゴリズムを提案する。

Ruskey ら[3] は長さちょうど n のグラフ的列を列挙するアルゴリズムを提案し、それが Constant Amortized Time(CAT) アルゴリズムであることを実験的に示した。それに対して本論文では長さちょうど n のグラフ的列を列挙するアルゴリズムを提案し、それが CAT アルゴリズムである証明を与えた。

本論文のアルゴリズムと Ruskey らのアルゴリズムは同様の方法で列挙を行っているが、本論文のアルゴリズムは 1 つのグラフ的列を最悪の場合でも定数時間で生成する。また、本論文では出力にかかる時間も考慮に入れて、計算時間について考察を行った。提案するアルゴリズムはすべてのグラフ的列を 1 つあたり $O(1)$ 時間で列挙する。ただし、出力は直前のグラフ的列からの差分である。

Enumerating All Graphical Sequences

Yosuke Kikuchi ,[†1] Katsuhisa Yamanaka [†2]
and Shin-ichi Nakano [†3]

The degree sequence of a simple graph is a sequence of degrees of vertices in the graph in decreasing order. If an integer sequence D is the degree sequence of some simple graph, then D is *graphical*. This paper proposes a simple algorithm to generate all graphical sequences with length at most n. Ruskey et al. gives an algorithm to generate all graphical sequences. Their algorithm generates each graphical sequence "possibly" in constant amortized time (CAT), and they have shown the efficiency by an experiment, and the problem where there is an algorithm to generate each graphical sequence in CAT has remained open. In this paper we design a simple algorithm to generate all graphical sequences, in CAT for each.

図 1　$S = (5, 3, 3, 3, 2, 2)$ を次数列としてもつ非同型なグラフの例
Fig. 1　Two graphs sharing the degree sequence $(5, 3, 3, 3, 2, 2,)$.

1. はじめに

単純グラフ G の各点の次数を降順に並べたものを G の次数列 (degree sequence) という。一方、与えられた整数列 D がある単純グラフの次数列であるとき D はグラフ的列 (graphic sequence, graphical sequence) である。ここで、次数列やグラフ的列は、降順であることに注意しよう。グラフから次数列は一意に定まるが、グラフ的列からグラフは一意に定まるとは限らない。例えば、図 1 にグラフ的列 $D = (5, 3, 3, 3, 2, 2)$ を次数列とする 2 つのグラフを示す。一方 $(3, 2, 2, 2)$ のようにグラフ的でない列もある。

本文は、高々n 個の整数からなるグラフ的列を、重複も抜けもなく、列挙するアルゴリズムを提案する。さらに、そのアルゴリズムの出力を工夫することでちょうど n 個の整数からなるグラフ的列を列挙するアルゴリズムを提案する。Ruskey ら[3] も同様の研究を行っている。Ruskey らは、彼らのアルゴリズムが Constant Amortized Time(CAT) アルゴリズムであることを実験によって示した。このように、証明はないが実験的に Constant Amortized Time(CAT) であることが示されたアルゴリズムを alley CAT(野良猫) とよんでいて、猫たちは良い家 (証明) を探している。　本論文ではその野良猫に家を提供する。さらに高々 n 個の整数からなるグラフ的列を列挙するアルゴリズムは CAT ではなく、1 つのグラフ的列を最悪でも定数時間で生成できることを示す。このアルゴリズムは、列挙する対象の集合に木構造を上手に定義し、この木構造に基づいて列挙する。このような木構造に基づく高速列挙アルゴリズムが幾つも知られている[1),4)–6)] が、本文はこの手法をグラフ的列の高速列

†1 津山工業高等専門学校情報工学科
†2 電気通信大学大学院情報システム学研究科情報システム基盤学専攻
†3 群馬大学大学院工学研究科情報工学専攻

挙に応用できるように工夫する。

2. 木 構 造

Havel, Erdös and Gallai, Hakimi は整数列がグラフ的列となる必要十分条件を与えた[2) [pp.12–15]。すべての高々n 点のグラフの次数列の集合、すなわち高々n 個の整数からなるグラフ的列の集合を S_n とする。例えば $S_3 = \{(0), (0,0), (1,1), (0,0,0), (2,1,1), (1,1,0), (2,2,2)\}$ である。

集合 S_n に木構造を定義する。$D \in S_n \setminus \{(0)\}$ としよう。1 個の数字からなるグラフ的列は (0) のみであるので、D は $k > 1$ 個の整数からなる。$D = (d_1, d_2, \ldots, d_k)$ とする。$d_1 \geq d_2 \geq \cdots \geq d_k$ である。次のようにして D から得られる新しい整数列を $P(D)$ としよう。

場合 1: $d_k = 0$ のとき。

D から最後の整数 0 を除去して得られる整数列を $P(D)$ とする。すなわち、$P(D) = (d_1, d_2, \ldots, d_{k-1})$ である。

場合 2: $d_k \neq 0$ のとき。

D の最初の整数 d_1 を除去し、2 番目から $1 + d_1$ 番目までの d_1 個の整数をそれぞれ 1 だけ減らし、さらに、降順にソートして得られる整数列を $P(D)$ とする。

後の補題に示すように、$d_1 \geq k$ のとき D はグラフ的列ではないので、$1 + d_1 \leq k$ であるとしてよい。また、場合 2 において、もし $d_{1+d_1} > d_{2+d_1}$ ならばソートは不要であることに注意しよう。一方、$d_{1+d_1} = d_{2+d_1}$ のときは $P(D)$ において $d_{1+d_1} < d_{2+d_1}$ となるので、ソートが必要となる。しかし、ソートの代わりに次のようにしてもよい。d_{1+d_1} と同じ値をもつ D の部分列が a 番目から始まり b 番目で終るとする。すなわち $d_a = d_{1+d_1} = d_b$ であるとする。D の a 番目から $1 + d_1$ 番目までの $c = 1 + d_1 - a + 1$ 個の整数をそれぞれ 1 だけ減らすかわりに、$b - c + 1$ 番目から b 番目までの c 個の整数をそれぞれ 1 だけ減らすことによって、D から簡単に $P(D)$ が求められる。すなわち、場合 2 では、D の先頭の整数 d_1 を除去したのち、d_2 から d_{a-1} までと d_{b-c+1} から d_b までの、2 つもしくは 1 つの部分列の整数を、それぞれ 1 だけ減らすことにより、$P(D)$ が得られる。$a = 2$ や $b = 1 + d_1$ のとき、扱う部分列は 1 つになることに注意しよう。また、D と $P(D)$ のいずれにおいても d_{b-c+1}=d_b である。

場合 1 のとき明らかに $P(D)$ はグラフ的列である。また、次の補題より、場合 2 のときも $P(D)$ はグラフ的である。

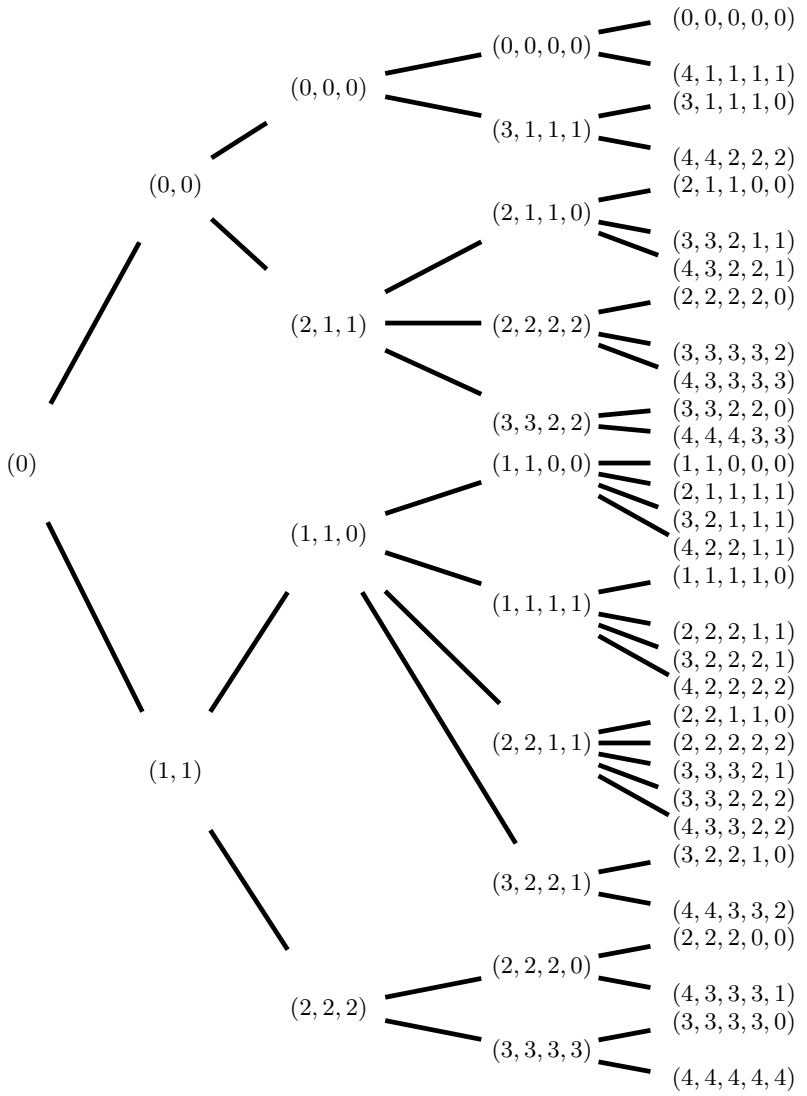

図 2 家系木 T_5
Fig. 2 The family tree T_5.

補題 1 [2] *[p.13]* $D = (d_1, d_2, \ldots, d_k)$ がグラフ的列であるのは $D = (d_2 - 1, d_3 - 1, \ldots, d_{1+d_1} - 1, d_{2+d_1}, d_{3+d_1}, \ldots, d_k)$ がグラフ的列であるときであり、また、そのときにかぎる。

どちらの場合も、$P(D)$ は D より整数の個数が 1 つ少ないことに注意しよう。$P(D)$ を D の親とし、D を $P(D)$ の子とする。D の親 $P(D)$ は一意であるのに対し、$P(D)$ の子は複数あるかもしれない。

任意の $D \in S_n \setminus \{(0)\}$ について、親、その親、その親のように繰り返し親を求めると、$D, P(D), P(P(D)), \ldots$ なるグラフ的列の列が、一意に定まる。親を求めるたびに 1 つずつ整数の個数が減り、最後には 1 つの整数からなるグラフ的列になる。1 つの整数からなるグラフ的列は (0) だけであるので、この列は必ずグラフ的列 (0) で終了する。すなわち集合 S_n に (0) を根とする木構造を定義したことになる。例を図 2 に示す。この木を S_n の家系木とよび T_n と書こう。T_n の深さ k の各点は、$k+1$ 個の整数からなるグラフ的列に対応する。

3. 列挙アルゴリズム

もし T_n 中の点 v が与えられたとき、v の子をすべて列挙するアルゴリズムがあれば、このアルゴリズムをグラフ的列 (0) から再帰的に実行することにより、T_n のすべての点、すなわち S_n 中のすべてのグラフ的列を列挙できる。本節ではそのような子を列挙するアルゴリズムを与える。

与えられたグラフ的列を $D = (d_1, d_2, \ldots, d_{k-1})$ としよう。D の子は、(i) 列 D の最後に整数 0 を追加したもの、もしくは、(ii) まず列 D の先頭に整数 $x > 0$ を追加し、次に、合計 x 個の整数からなる、高々2 つの D の部分列を選び、部分列の整数をそれぞれ 1 だけ増加したもの、のいずれかである。いずれの場合も、D の子は k 個の整数からなる。(i) により得られる列を $C[0] = (d_1, d_2, \ldots, d_{k-1}, 0)$ としよう。これを**タイプ 0** とよぶ。(ii) により得られる列は、高々2 つの部分列が D のどこにあるかによって、次の 3 つのタイプがある。

タイプ 1: $C[x]$

部分列が 1 つで、D の 1 番目からの x 個の整数 $d_1, d_2, \ldots, d_x$ のもの。$x \leq k-1$ であることに注意しよう。

タイプ 2: $C[x, s]$

部分列が 1 つで、D の $s \neq 1$ 番目からの x 個の整数 $d_s, d_{s+1}, \ldots, d_{s+x-1}$ のもの。$d_1 = d_{s-1} = 1 + d_s$ であり、d_1 や d_{s-1} には $+1$ しないことに気をつけよう。よって $x < k-1$ である。

タイプ 3: $C[x, r, s]$

部分列が 2 つで、D の 1 番目から $r \geq 1$ 番目までと $s > r+1$ 番目から $t = s+x-r-1$ 番目までの計 x 個の整数のもの。d_{r+1} や d_{s-1} には $+1$ しないことに気をつけよう。よって、$x < k-1$ である。

D の子は $C[0], C[x], C[x, s], C[x, r, s]$ のいずれかである。しかし、これらのすべてが D の子になるわけではない。それぞれの場合について考察しよう。D が n 個の整数からなるとき、すなわち、もし $k-1 = n$ ならば、いずれのタイプも $n+1$ 個の整数からなるので D の子でない。すなわち D は子を持たない。よって $k-1 < n$ としよう。

タイプ 0: $C[0] = (d_1, d_2, \ldots, d_{k-1}, 0)$ のもの。

$C[0]$ の親は D なので、$C[0]$ は D の子である。

タイプ 1: $C[x] = (x, 1+d_1, 1+d_2, \ldots, 1+d_x, d_{x+1}, d_{x+2}, \ldots, d_{k-1})$ のもの。

さらに 2 つのタイプにわけて考える。

タイプ 1(a): $C[x]$ の最後の整数 d_{k-1} が 0 のもの。

$C[x]$ の親は、$C[x]$ から最後の整数 0 を削除して得られる整数列であり、D ではない。すなわちタイプ 1(a) は D の子でない。

タイプ 1(b): $C[x]$ の最後の整数 d_{k-1} が 0 でないもの。

$x \leq d_1$ なる x のとき、$C[x]$ は降順でない。よって $C[x]$ は D の子でない。$x \geq k$ なる x のとき、$C[x]$ は定義できない。$d_1 < x < k$ なる x のとき、$C[x]$ は降順にソートされており、$C[x]$ の親は D であるので、$C[x]$ は D の子である。

すなわち、D の最後の整数 d_{k-1} が 0 でないときは、$d_1 < x < k$ なる各 x について $C[x]$ は D の子である。D の d_{k-1} が 0 のときは、$C[k-1]$ のみが D の子である。

タイプ 2: $C[x, s] = (x, d_1, d_2, \ldots, d_{s-1}, 1+d_s, 1+d_{s+1}, \ldots, 1+d_{s+x-1}, d_{s+x}, d_{s+x+1}, \ldots, d_{k-1})$ のもの。

さらに 2 つのタイプにわける。

タイプ 2(a): $C[x, s]$ の最後の整数 d_{k-1} が 0 のもの。

$C[x, s]$ の親は、$C[x, s]$ から最後の整数 0 を削除して得られる整数列であり、D ではない。すなわち、タイプ 2(a) は D の子でない。

タイプ 2(b): $C[x, s]$ の最後の整数 d_{k-1} が 0 でないもの。

$x < d_1$ または $d_{s-1} = d_s$ のとき、$C[x, s]$ は降順でない。よって D の子でない。$s+x-1 > k-1$ のとき $C[x, s]$ は定義できない。$d_1 > d_{s-1}$ もしくは $d_{s-1} \geq 2 + d_s$ もしくは $d_s > d_{s+x-1}$ のとき、$C[x, s]$ の親は D でない。よって、$d_1 = d_{s-1} = 1 + d_s$ のときの

みタイプ 2 の子がある可能性がある。このとき $d_t = d_s$ なる各 $t \geq s$ について $x = t - s + 1$ としたとき, $x \geq d_1$ なる $C[x, s, t]$ は D の子である。

タイプ 3: $C[x, r, s] = (x, 1 + d_1, 1 + d_2, \ldots, 1 + d_r, d_{r+1}, d_{r+2}, \ldots, d_{s-1}, 1 + d_s, 1 + d_{s+1}, \ldots, 1 + d_{s+x-r-1}, d_{s+x-r}, d_{x+x-r+1}, \ldots, d_{k-1})$ のもの。

さらに 2 つのタイプにわけて考える。

タイプ 3(a): $C[x, r, s]$ の最後の整数 d_{k-1} が 0 のもの。

$C[x, r, s]$ の親は、$C[x, r, s]$ から最後の整数 0 を削除して得られる整数列であり、D ではない。すなわち、タイプ 3(a) は D の子でない。

タイプ 3(b): $C[x, r, s]$ の最後の整数 d_{k-1} が 0 でないもの。

$x \leq d_1$, または $d_{s-1} = d_s$ のとき、$C[x, r, s]$ は降順でない。よって D の子でない。$s+x-r-1 > k-1$ のとき $C[x, r, s]$ は定義できない。$d_{r+1} > d_{s-1}$、もしくは、$d_{s-1} \geq 2+d_s$ のとき、$C[x, r, s]$ の親は D でない。

$d_{r+1} = d_{s-1} = 1 + d_s$ なる各 r, s についてタイプ 3 の子がある可能性がある。このとき $d_s = d_t$ なる各 $t \geq s$ について $x = r + t - s + 1$ としたとき, $x > d_1$ なる $C[x, r, s]$ は D の子である。

以上の場合わけから、グラフ的列を列挙するアルゴリズムが得られる。

Procedure find-all-children$(D = (d_1, d_2, \ldots, d_{k-1}), k - 1)$
begin
D を出力

if $k - 1 = n$
then return %子なし

%タイプ 0 の子
find-all-children$(C[0], k)$

%タイプ 1 の子
if $d_{k-1} \neq 0$ **then**
  for $x = 1 + d_1$ **to** $k - 1$
    find-all-children$(C[x], k)$
else
  find-all-children$(C[k - 1], k)$

% タイプ 2 の子 (d_1 や d_{s-1} は +1 しない)
if $d_{k-1} = 0$ **then**
  if $d_1 = 1$ **then** % $d_1 = 1$ かつ $d_{k-1} = 0$
  begin
  $1 = d_{s-1} = 1 + d_s$ なる $s < k - 1$ を選ぶ
  $x = k - 1 - s + 1$ とする。
  if $x \geq d_1$ **then**
    find-all-children$(C[x, s], k)$
  end
else % $d_{k-1} \geq 1$
  if $d_1 = d_{s-1} = 1 + d_s$ なる s がある **then**
  begin
  そのような s を選ぶ
  for $t \geq s$ かつ $d_s = d_t$ なる各 t について降順に **do**
    begin
    $x = t - s + 1$
    if $x < d_1$ **then break**
    find-all-children$(C[x, s], k)$
    end
  end

% タイプ 3 の子 (d_1 に +1 する)(d_{r+1} や d_{s-1} に +1 しない)
if $d_{k-1} = 0$ **then** % 子は必ず d_{k-1} に +1 する
  if $1 = d_{s-1} = 1 + d_s$ なる s がある **then**
    for $d_{r+1} = d_{s-1}$ なる各 r について降順に **do**
      begin
      $x = r + k - 1 - s + 1$
      if $x \leq d_1$ **then break**
      find-all-children $(C[x, r, s], k)$

Vol.2009-AL-126 No.2
2009/9/15

end
else % $d_k \neq 0$
% 下のループはひとつも子をもたない s 以降は処理を打ち切り
for $d_{s-1} = 1 + d_s$ なる 各 s について降順に **do**
% 下のループはひとつも子をもたない r 以降は処理を打ち切り
for $d_{r+1} = d_{s-1}$ なる各 r について降順に **do**
for $d_s = d_t$ かつ $t \geq s$ なる各 t について降順に **do**
begin
$x = r + t - s + 1$
if $x \leq d_1$ **then break**
find-all-children $(C[x, r, s], k)$
end
end

次の定理がいえる。

定理 1 上のアルゴリズムは S_n 中のグラフ的列を列挙する。計算時間はグラフ列 1 個あたり平均 $O(1)$ 時間である。計算領域は $O(n)$ である。

証明 家系木の定義と子の場合分けにより、アルゴリズムは S_n 中すべてのグラフ的列を列挙する。

次にデータ構造について説明しよう。図 3 参照。

現在のグラフ的列 $D = (d_1, d_2, \ldots, d_{k-1})$ の各整数を、直前の整数からの差分に置き換え、配列 DIFF[n..-n] に格納する。ただし、便宜的に、d_1 の直前の整数を $d_0 = n$ とみなす。子を生成するとき、列の先頭や末尾に数字を追加するので、DIFF のインデックスを $-n$ から $+n$ までとすることに注意しよう。また、現在のグラフ的列の先頭と末尾のインデックスを変数 HEAD と TAIL に格納する。また、グラフ的列の最後の整数の値が 0 であるかどうか判定する必要があるのでこれを変数 LST に格納する。

また、プログラム中で $d_{s-1} = 1 + d_s$ なる s や, $d_{s-1} > d_s$ なる s を参照するので、この条件を満たす s のみを双方向リストとして配列上に格納する。それぞれ、JST-One-List[-n..+n] と NON-Zero-List[-n..+n] とする。また、それぞれのリストの両端を変数に格納する。

以上のデータ構造に必要な計算領域は $O(n)$ である。また、$O(n)$ のスタックを準備する。

子を生成するとき、これらのデータ構造の更新は、定数時間しかかからない。このときの更新をスタックに記憶しておけば、子のデータ構造から元の親のデータ構造に、定数時間で戻すこともできる。

また、タイプ 3 の 2 箇所のループの打ち切りも、フラグを用意することで簡単に実現できる。

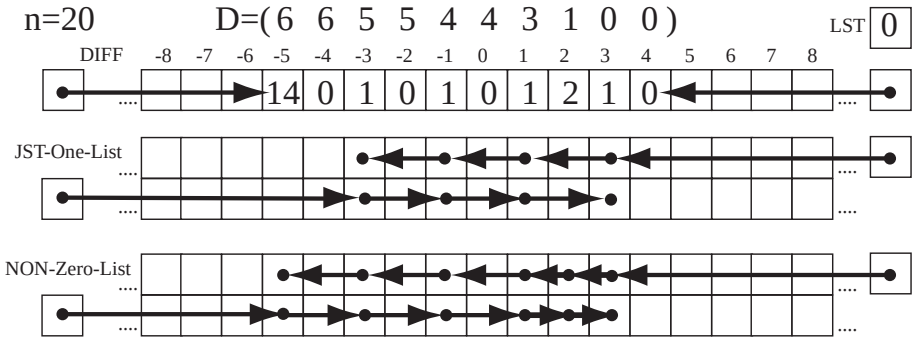

図 3 データ構造の説明図
Fig. 3 Explanation for data structure

上記のデータ構造により、子が z 個あるとき $O(z)$ 時間でこれらを列挙できる。ただし出力は、直前のグラフ的列の差分列からの、修正操作の列、すなわち差分である。

以上により、アルゴリズムは S_n 中のすべてのグラフ的列を $O(|S_n|)$ 時間で列挙する。すなわち、すべてのグラフ的列を、1 個あたり平均 $O(1)$ 時間で列挙する。 ■

アルゴリズムは、すべてのグラフ的列を、1 個あたり平均 $O(1)$ 時間で列挙するが、次のグラフ的列を $O(1)$ 時間で出力するとは限らないことに注意しよう。例えば、大きな部分木の最後の点を出力した後、次の点を出力するためには、多くの時間を単に深い再帰呼び出しから戻るために必要とするかもしれない。しかし、家系木中の深さが、偶数の点に対応するグラフ的列は、子のグラフ的列より前に出力し、奇数の点に対応するグラフ的列は、子のグラフ的列の後に出力するようにアルゴリズムを改造することにより[7]、次のグラフ的列を、1 個あたり $O(1)$ 時間で列挙できる。

4. n 個の整数からなるグラフ的列の列挙

ちょうど n 個の整数からなるグラフ的列の集合を $S_{=n}$ とする。3 章のアルゴリズムを、家系木の葉に対応する列のみ列挙するように改造すると $S_{=n}$ 中の列を列挙するアルゴリズ

Vol.2009-AL-126 No.2
2009/9/15

ムが得られる。次の定理がいえる。

定理 2 ちょうど n 個の整数からなるグラフ的列を、1 個あたり平均 $O(1)$ 時間で列挙できる

証明 $k-1 < n$ なる列 $D = (d_1, \ldots, d_{k-1}) \in S_n$ はタイプ 0 の子 $C[0]$ とタイプ 1 の子 $C[k-1]$ をもつ。すなわち D は子を 2 つ以上もつので、$|S_n| \leq 2|S_{=n}|$ である。

定理 1 より、$O(|S_n|)$ 時間で S_n 中のすべての列を列挙できる。よって、$|S_n| \leq 2|S_{=n}|$ より、$O(|S_{=n}|)$ 時間で $S_{=n}$ 中のすべての列を列挙できる。すなわち、長さがちょうど n のグラフ的列を、1 個あたり平均 $O(1)$ 時間で列挙できる。 ■

5. ま と め

高々n 個の整数からなるグラフ的列を、重複も抜けもなく、高速に列挙するアルゴリズムを設計した。このアルゴリズムは、すべてのグラフ的列を、1 個あたり $O(1)$ 時間で列挙する。

平面グラフや連結グラフのすべてのグラフ的列の高速列挙アルゴリズムを開発することが今後の課題である。

参 考 文 献

1) D.Avis and K.Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 6, 21–46, 1996.
2) G.Chartrand and L.Lesniak, *Graphs & Digraphs*, 4th ed., Chapman & Hall, 2005.
3) F.Ruskey, P.Eades, B.Cohen and A.Scott, *Alley CATs in search of good homes*, Congressus Numerantium., 102, 97–110, 1994.
4) S.Nakano, *Enumerating floorplans with n rooms*, Proc. ISAAC2001, LNCS 2223, 104–115, 2001.
5) S.Nakano, *Efficient generation of plane trees*, Inf. Process. Lett., 84, 167–172, 2002.
6) S. Nakano, *Efficient generation of triconnected plane triangurations*, Computational Geometry, Theory and Applications, 27, 109–122, 2004.
7) S.Nakano and T.Uno, *Constant time generation of trees with specified diameter*, Proc. WG 2004, LNCS 3353, 33-45, 2004.