A-009

定数葉数無順序木の列挙に対する再帰的アプローチについて
A Recursive Approach to the Enumeration of Unordered Trees with a Fixed Number of Leaves

豊島 海羅† 正代 隆義‡
Kaira Teshima Takayoshi Shoudai

1. はじめに

特徴的な構造を持つ順序木・無順序木を全て列挙したリストは，ある仮説の反例の発見や特定の基準に基づいた最適な対象を発見することに有用である．そのため，順序木や無順序木を，重複なく全て列挙するアルゴリズムが開発されてきた[1]．順序木では比較的容易に列挙可能な特徴的構造でも，無順序木では，各頂点の子に兄弟関係が存在しないため，順序木の生成と同様な手法は使えない．

$T(n,k)$を，頂点数$n \geq 1$，葉数$k \geq 1$の無順序木の個数と定める．無順序木の列挙において，Ishikawa ら[2]は任意の頂点数n，葉数kの無順序木に対する列挙アルゴリズムを提案した．本論文では，頂点数n，葉数kの無順序木に対し，根の子を根とする部分木の構成を長さk以下の可変長の有限列$[(h_1,a_1,t_1),\dots,(h_w,a_w,t_w)]$ $(1 \leq w \leq k)$として，重複なく完全に列挙する問題を考える．この有限列は相異なる(h_i,a_i)と個数$t_i \geq 1$の組から成り，

- 各$i=1,\dots,w$に対して$T(h_i,a_i) > 0$，
- $\sum_{i=1}^{w} h_i \cdot t_i = n-1$，$\sum_{i=1}^{w} a_i \cdot t_i = k$

を満たす．本研究では，整数nをk個に分ける順序付き分割の数$P(n,k)$を再帰的に構成する方法を基に，その考え方を発展させて無順序木の部分構造を構成する手法を検討する．

2. 固定長降順整数分割

Knuth[3]のアルゴリズム P（Algorithm P）**は，**整数の非負整数分割（partition）を列挙するための効率的なアルゴリズムである．アルゴリズム P は，辞書式順または共通の規則に従って順序を保ちながらすべての分割を生成する．本論文では，正の整数nと自然数kが与えられたとき，集合$P(n,k)$を，長さkの非負整数列のうち，すべての要素の和がnに等しく，かつ各要素が非増加順に並んでいるもの全体からなる集合として次のように定義する:

$$P(n,k) = \{ (a_1,a_2,\dots,a_k) \in \mathbb{Z}^k \mid a_1 \geq a_2 \geq \cdots \geq a_k \geq 0 \text{ かつ} a_1+a_2+\cdots+a_k = n \}.$$

例えば，$P(4,2) = \{(4,0),(3,1),(2,2)\}$となる．

図 1 に示すアルゴリズムは，集合$P(n,k)$の要素を重複なく列挙するための再帰的手法である．関数 P_Sub は，降順の非負整数列を次の 3 種類の操作を用いて再帰的に生成する: 正規化，前方展開，末尾統合．正規化は，列が$[a,1,1,\dots,1]$ $(a \geq 2)$のとき，$[a-1,0,0,\dots,0]$に変換する．変換後に列の要素が 1 減少するため，後に出力する際にはその分の補正を行う．この補正値を offset と呼び，正規化が適用されるたびに 1 ずつ増加させ．最終的に出力される列は，offset を全要素に加えることで元の列を復元する．前方展開は，先頭要素を 1 減らし，その分を後方に分配する操作である．末尾統合は，列の末尾 x[last]が 1 であり，x[last - 2]と x[last - 1]の差が 1 以上ある場合に適用される．正規化と最後尾が 1 での末尾統合が非対象のときに，列の各要素に offset を加算したものが出力される．

```
関数 P_Sub(x, r, offset, last):
  # 出力条件：正規化と最後尾が1での末尾統合が非対象
  if len(x) < 3 or x[r - 1] == 0 or
     (not(x[r - 1] == 1 and x[r - 3] >= x[r - 2] + 1) # 末尾統合非対象
      and not(x[1] == 1 and x[r - 1] == 1)): # 正規化非対象
     yield [v + offset for v in x]
  # 正規化の適用：[a,1,1,...,1] を [a-1,0,...,0] に変換
  if len(x) >= 2 and x[1] == 1 and x[r - 1] == 1:
     x[0] = x[0] - 1; x[1:] = [0] * (r - 1)
     offset = offset + 1; last = 0
     yield from P_Sub(x, r, offset, last)
  # 前方展開の適用：x[last+1] = 0 かつ展開条件を満たすとき
  if len(x) >= 2 and x[0] > x[1] and not(x[0] == 1 and x[1] == 0):
     if (last + 1 != r - 1 or x[1] == 1
       or (x[1] > 1 and x[last - 1] >= x[last] + 1)):
        x[0] = x[0] - 1; x[last + 1] = x[last + 1] + 1; last = last + 1
        yield from P_Sub(x, r, offset, last)
  # 末尾統合の適用：x[last] = 1 かつその左隣との差が 1 以上
  if last >= 2 and x[last] == 1 and x[last - 2] >= x[last - 1] + 1:
     x[last] = 0; x[last - 1] = x[last - 1] + 1; last = last - 1
     yield from P_Sub(x, r, offset, last)

関数 P_Main(n, k):
  x := [n, 0, 0, ..., 0] (長さ k); offset = 0; last = 0
  yield from P_Sub(x, k, offset, last)
```

図 1：関数 P_Sub(n, k)と P_Main(n, k)．yield は値を返して処理を中断・再開できる命令である．

本章で示した関数 P_Main(n, k)により列挙される各数列に，次章で述べる無順序木の処理のために，コストを定める．数列$s=(a_1,a_2,\dots,a_k) \in P(n,k)$に対して，次のようなコスト関数を定義する:

$$cost(s) = \sum_{i=1}^{k} f(a_i), \quad f(a) = \begin{cases} a+1, & a \geq 2 \text{ のとき,} \\ a, & \text{その他.} \end{cases}$$

このコストは，次章で扱う定数葉数無順序木の列挙において，構造の枝刈り条件として用いられる．すなわち，構成される数列のコストがあらかじめ定められた上限を超える場合には，その分岐以下の探索を打ち切る．

3. 定数葉数無順序木の再帰的構造

3.1 例: 葉数 5 の無順序木の構造

最初に例として，頂点数をn，葉数kを 5 としたときの左深さ優先無順序木の数を考える．根の子を頂点とする部分木が持つ葉数の場合の集合は$P(5,5)$で表される．例えば，根が 3 つの子を持つならば，それらを根とする 3 つの部分

†福岡工業大学大学院工学研究科情報工学専攻
Department of Computer Science and Engineering, Fukuoka Institute of Technology
‡福岡工業大学情報工学部情報工学科 Department of Computer Science and Engineering, Fukuoka Institute of Technology

```
関数 T_Sub(n, k):
  # 特殊ケース：唯一の数列 [1,1,...,1] に対応
  if k == n - 1:
    yield [(1, 1, k)]
    return
  # ステップ 1：数列 s を P(k, k) から列挙
  for s in P(k, k):
    cost = 0
    # ステップ 2：数列 s に対してコストを計算
    # a == 1 のとき 1，a ≥ 2 のとき a + 1
    for a in s:
      if a == 1: cost = cost + 1; else: cost = cost + (a + 1)
    # cost > n - 1 の時点で打ち切る
    if cost > n - 1: break
    # ステップ 3：残余 m を計算，s から (a, c) の形式へ変換
    m = n - cost - 1
    count_form = to_count_form(s) # a > 0 のみを対象
    # ステップ 4：Q(m, l) により残余を分割，q を列挙
    for q in Q(m, len(count_form)):
      triples = [ ]
      # ステップ 5：各 (a, c, d) に対して配分列 p を列挙
      for i in range(len(count_form)):
        (a, c) = count_form[i]; d = q[i]
        for p in P(d, c):  # d を c 個に分配する
          for e in p:  # 各 e に対して h を構成する
            if a == 1: h = a + e; else: h = a + e + 1
            triples.append((h, a))
      # ステップ 6：triple リストをカウント形式に変換
      count_form_p = to_count_form(triples)
      yield count_form_p

関数 T_Main(n, k):
  if k == n - 1: return 1  # 特殊ケース：数列 [1, ..., 1] のみ
  total = 0
  for triples in T_Sub(n, k):  # triple (h, a, t) の列を列挙
    product = 1
    for (h, a, t) in triples:
      value = T_Main(h, a) # 再帰的に T を評価
      product = product × C(value, t)  # 重複組合せ
    total = total + product  # 合計に加算する
  return total
```

図 2：関数 T_Sub(n,k)と T_Main(n,k)

木が持つ葉の数は非増加順に(3,1,1)か(2,2,1)の 2 パターンしかない．このように根の子を根とする部分木が持つ葉の数は，次の 7 パターンで表される:

$$(5),(4,1),(3,1,1),(2,1,1,1),(1,1,1,1,1),(3,2),(2,2,1).$$

各数列に 0 を付加して長さを 5 に統一すれば，これらのパターンが$P(5,5)$で生成できることがわかる．ただし，頂点数nによっては，実現できないパターンがあることに注意する．例えば，根の子の数 3 で，それらの子が子孫とする葉の数(2,2,1)は$n = 6,7$の場合は実現できない．これは cost([2,2,1,0,0])=8 から，(2,2,1)のために少なくとも 8 頂点必要であることが判断できる．

3.2 定数葉数無順序木の部分構造の列挙

関数$Q(m,r)$は，非負整数mを順序付きr個の非負整数に分割する方法を列挙する[3]．列挙数は$\binom{m+r-1}{r-1}$で与えられる．関数$C(n,k) = \binom{n+k-1}{k}$は，$n$種類から重複を許して$k$個選ぶ組合せの数で，部分木の重複選択に用いる．

関数 T_Sub(n, k) (図 2)は，頂点数が n，葉数がkである無順序木の構成に対応する中間的な構造表現として，三つ組(triple)のリスト(triples)を列挙する補助関数である．出力される三つ組リストは，次の形式の組からなる:

$$[(h_1,a_1,t_1),(h_2,a_2,t_2),\dots,(h_w,a_w,t_w)].$$

ここで，各組(h_i,a_i,t_i)は次を表す: h_iは部分木の頂点数，a_iは部分木の葉数，t_iはその構造の部分木の出現数を表す．T_Sub(n, k)は最初に，構造列s (長さk，総和kの非負整数列)を$P(k,k)$により列挙し，各コストを計算する．コストが$n-1$を超えないならば，残余頂点数$m = n - cost - 1$を計算し，数列を形式(a,c) (cは値aの出現回数を表す)に変換する．異なる(a,c)が全部でr形式あれば，$Q(m,r)$により，mを順序付きで分割する．さらに各(a,c,d)に対して$P(d,c)$で内部割当を列挙する．各割当から部分木サイズ $h = a + e$ ($a = 1$のとき)または$h = a + e + 1$ (それ以外)を構成し，対応する(h,a)を集めて三つ組(h,a,t)に集約する．これにより，数列に対応する全ての部分構造の候補が得られる．なお，関数 T_Main(n, k)では，各三つ組に対して部分木の重複選択数を重複組合せ $C(T(h,a),t) = \binom{T(h,a)+t-1}{t}$により評価する．以上，本アルゴリズムについて，次の命題が成り立つ:

命題 任意の 3 つ組リストに対して、次の式が成り立つ:

$$\sum_{i=1}^{w} h_i \cdot t_i = n-1, \qquad \sum_{i=1}^{w} a_i \cdot t_i = k.$$

関数 T_Main(n, k)は，T_Sub(n, k)により得られた triple リストに対して，各部分構造(h_i,a_i)における再帰的な部分木の個数$T(h_i,a_i)$を評価し，それらの重複組合せの積を合計することで，全体の無順序木の構成数を求める．具体的には，各三つ組に対して次式により評価を行い，合計を取る:

$$T(n,k) = \sum_{triples} \prod_{i=1}^{w} C(T(h_i,a_i),t_i).$$

この計算により，部分木の組合せによる頂点数n，葉数kの無順序木全体の数が正確に計算される．例として頂点数$n = 32$に対する関数$T(n,k)$の計算結果を示す: $T(32,k)$ ($k = 1,\dots,32$) : [1, 240, 17665, 607445, 11681631, 139551650, 1109643254, 6169083918, 24885909121, 74995707572, 172870609803, 310810391963, 443079133769, 507792236830, 473329912016, 362343773669, 229607170694, 121187922747, 53524094575, 19841915619, 6183126522, 1619414666, 355792533, 65314335, 9954713, 1247856, 126938, 10279, 645, 30, 1, 0], 合計 2809934352700 は頂点数32の無順序木の数と等しい (参照: OEIS, https://oeis.org/A000081/b000081.txt).

4. おわりに

本研究では，頂点数nと葉数kを指定した無順序木の部分構造の再帰的列挙アルゴリズム T_Sub(n,k)を与え，それを用いて，$T(n,k)$の計算結果を示した．

今後の課題は，定数葉数無順序木データに頻出する無順序木構造パターンの効率的な抽出を目的とした機械学習アルゴリズムの設計と解析である．具体的には，グラフ畳み込みネットワークと計算論的学習モデルを融合した定数葉数無順序木データのための高精度で効率的な機械学習モデルの開発および実装があげられる．

謝辞 本研究は JSPS 科研費 JP21K12021, JP24K15074, JP24K15090 の助成を受けたものです．

参考文献

[1] 岡本 吉央: 列挙の基本と基礎的なアルゴリズム，電子情報通信学会誌, 95 (6), pp.477-483 (2012).
[2] Ishikawa M. et al.: Enumerating All Rooted Trees Including k Leaves, IEICE Trans. Inf & Syst., E95-D (3), pp.763-768 (2012).
[3] Knuth D. E. 著，筧 一彦，小出 洋訳: The Art of Computer Programming Volume 4A Combinatorial Algorithms Part1 日本語版, KADOKAWA (2017).