November 24, 2020 at 13:23

1.   Intro.   This program tests an amazingly simple algorithm that generates all n-node trees with the property that the kth node in preorder has t_k link fields. (A link field is either zero or it points to a subtree.) In the special case that $t_k = 2$ for $1 \le k \le n$, we get Skarbek's algorithm for binary trees, Algorithm 7.2.1.6B. In the special case that $t_k = t$ for $1 \le k \le n$, we get an algorithm that was sent to me by James Korsh in December 2004. I happened to notice that Korsh's idea works in the general case considered here; but I'll let him have the fun of constructing a formal proof, because the basic insights are essentially his.

   The number of trees generated does not appear to have a simple formula in general. But one can show bijectively that such trees are equivalent to integer sequences $a_1 \ldots a_{n-1}$ with the property that $a_1 \ge \cdots \ge a_{n-1} \ge 0$ and $a_k \le \sum_{j=1}^{n-k}(t_j - 1)$.

   The numbers t_1, ..., t_n are input on the command line.

#define *maxn*   100   /* n should be less than this */

#include `<stdio.h>`
  int $h[maxn]$;   /* the table of degrees; $h_k = t_k - 1$ */
  int $l[maxn][maxn]$;   /* the links (right to left) */
  int *count*;

  $main(\textbf{int}\ argc, \textbf{char}\ *argv[\,])$
  {
    register int j, k, n, r, x, y;
    ⟨ Read the t's from the command line 2 ⟩;
    for $(j = 1;\ j < n;\ j{+}{+})\ l[j][h[j]] = j + 1$;
    while (1) {
      ⟨ Visit the current tree 3 ⟩;
      for $(j = 1, x = l[1][0];\ x \equiv j + 1;\ j = x, x = l[j][0])\ l[j][0] = 0, l[j][h[j]] = x$;
      if $(j \equiv n)$ **break**;
      for $(r = 1;\ l[j][r] \equiv 0;\ r{+}{+})$;
      for $(k = 0, y = l[j][r];\ l[y][0];\ k = y, y = l[y][0])$;
      if (k) $l[k][0] = 0$; **else** $l[j][r] = 0$;
      $l[j][0] = 0, l[j][r - 1] = y, l[y][0] = x$;
    }
  }

2.   **#define** $abort(m)$
      { $fprintf(stderr, \texttt{"\%s!\textbackslash n"}, m)$;
        $exit(j)$; }
⟨ Read the t's from the command line 2 ⟩ ≡
  $n = argc - 1$;
  if $(n \equiv 0)$ {
    $fprintf(stderr, \texttt{"Usage:\_\%s\_t1\_t2\_...\_tn\textbackslash n"}, argv[0])$;
    $exit(0)$;
  }
  if $(n \ge maxn)$ $abort(\texttt{"I\_can't\_handle\_that\_many\_degrees"})$;
  for $(j = 1;\ j \le n;\ j{+}{+})$ {
    if $(sscanf(argv[j], \texttt{"\%d"}, \&h[j]) \ne 1)$ $abort(\texttt{"unreadable\_degree"})$;
    $h[j]{-}{-}$;
    if $(h[j] < 0)$ $abort(\texttt{"Each\_degree\_must\_be\_positive"})$;
    if $(h[j] \ge maxn)$ $abort(\texttt{"Degree\_is\_too\_large"})$;
  }
This code is used in section 1.

3.   For each tree, we print out the array of links, with link 0 last.

$\langle$ Visit the current tree $3 \rangle \equiv$

 $count{+}{+};$
 $printf\,(\texttt{"\%d:"}, count\,);$
 for $(j = 1;\ j \leq n;\ j{+}{+})$ {
  for $(k = h[j];\ k \geq 0;\ k{-}{-})\ printf\,(\texttt{"\_\%d"}, l[j][k]);$
  if $(j < n)\ printf\,(\texttt{";"});$
 }
 $printf\,(\texttt{"\textbackslash n"});$

This code is used in section 1.

4.   Index.

abort: 2.
argc: 1, 2.
argv: 1, 2.
count: 1, 3.
exit: 2.
fprintf: 2.
h: 1.
j: 1.
k: 1.
l: 1.
main: 1.
maxn: 1, 2.
n: 1.
printf: 3.
r: 1.
sscanf: 2.
stderr: 2.
x: 1.
y: 1.

⟨ Read the t's from the command line 2 ⟩   Used in section 1.
⟨ Visit the current tree 3 ⟩   Used in section 1.

LINKED-TREES