§1 RANDOM-TERNARY INTRO 1

1. Intro. Here's an implementation of the Panholzer–Prodinger algorithm, which generates a uniformly random "decorated" ternary tree. (It generalizes the binary method of Rémy, Algorithm 7.2.1.6R, and solves exercise 7.2.1.6–65.) They presented it in *Discrete Mathematics* 250 (2002), 181–195, but without spelling out an efficient implementation.

Although the algorithm is short, it is not easy to discover; the reader who thinks otherwise is invited to invent it before reading further.

I'm using a linked structure as in the presentation of in Rémy's method in Volume 4A: There are 3n + 1 links L_0, L_1, \ldots, L_{3n} , which are a permutation of the integers $\{0, 1, \ldots, 3n\}$. Internal (branch) nodes have numbers congruent to 2 (mod 3). The root is node number L_0 ; the descendants of branch 3k - 1 are the nodes numbered L_{3k-2} , L_{3k-1} , L_{3k} . For example, if n = 3 and $(L_0, L_1, \ldots, L_9) = (5, 0, 1, 3, 2, 6, 7, 8, 4, 9)$, the root is node 5 (a branch node); its left child is node 2 (another branch), its middle child is node 6 (external), and its right child is node 8 (yet another branch).

I also maintain the inverse permutation (P_0, \ldots, P_{3n}) , so that we can determine the parent of each node.

```
#define nmax 1000
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"
                                /* random number generator from the Stanford GraphBase */
  int nn, seed;
                     /* command-line parameters */
  int L[nmax], P[nmax];
                                 /* the links and their inverses */
  main(\mathbf{int} \ argc, \mathbf{char} * argv[])
     register int i, j, k, n, nnn, p, q, r, x;
     \langle \text{Process the command line } 2 \rangle;
     for (n = L[0] = P[0] = 0; n < nn;)
       \langle Extend a solution for n to a solution for n+1, and increase n \ 3 \rangle;
     for (k = 0; k \le 3 * nn; k++) printf("\"\", L[k]);
     printf("\n");
    \langle \text{Process the command line } 2 \rangle \equiv
  if (argc \neq 3 \lor sscanf(argv[1], "%d", \&nn) \neq 1 \lor sscanf(argv[2], "%d", \&seed) \neq 1) {
     fprintf(stderr, "Usage: \_\%s\_n\_seed \n", argv[0]);
     exit(-1);
  qb\_init\_rand(seed);
This code is used in section 1.
```

2 INTRO RANDOM-TERNARY

ξ3

3. #define sanity_checking 1

```
\langle Extend a solution for n to a solution for n+1, and increase n \rangle \equiv
  {
    n++;
    nnn = 3 * n;
    x = gb\_unif\_rand(3 * (nnn - 1) * (nnn - 2));
    p = nnn - (x \% 3);
    q = nnn - ((x+1) \% 3);
    r = nnn - ((x+2) \% 3);
    k = ((\mathbf{int})(x/3)) \% (nnn - 2);
    j = (\mathbf{int})(x/(9*n-6));
    L[p] = nnn, P[nnn] = p;
    L[q] = L[k], P[L[k]] = q, L[k] = nnn - 1, P[nnn - 1] = k;
    \langle \text{ Do the magic switcheroo } 5 \rangle;
    if (sanity_checking) {
       for (i = 0; i < nnn; i++)
         if (P[L[i]] \neq i) {
            fprintf(stderr, "(whoa---the_links_lare_lfouled_lup!)\n");
            exit(-2);
```

This code is used in section 1.

4. Variables j and L_k correspond to P-and-P's nodes y and x; they are random integers with $0 \le j \le 3n+1$ and $0 \le k \le 3n$. The basic idea is to insert a new branch node (node number 3n-1) in place of node L_k ; but this new node has the old node L_k as one of its children (pointed to by L_q), so we haven't really lost anything. Another child, pointed to by L_p , is the leaf 3n. The third child, pointed to by L_r , has to somehow encode the fact that we also need to place the leaf 3n-2 while maintaining randomness.

There are two main cases, depending on whether node number y is a proper ancestor of node number x. The crucial point, proved in the paper, is that we can recover x, y, and p by looking at the switched links.

```
5. \langle Do the magic switcheroo 5\rangle \equiv for (i=k+1-((k+2)\%3);\ i>0 \land i\neq j;\ i=P[i]+1-((P[i]+2)\%3)); if (i>0) \langle Do the harder case 6\rangle else \{ if (j\equiv L[q]) \{ /*\ y=x\ */ L[r]=L[q], P[L[q]]=r, L[q]=nnn-2, P[nnn-2]=q; \} else if (j\equiv nnn-2) \{ /*\ y is the special leaf */ L[r]=nnn-2, P[nnn-2]=r; \} else L[P[j]]=nnn-2, P[nnn-2]=P[j], L[r]=j, P[j]=r; \} This code is used in section 3.
```

6. The "harder case" isn't really hard for the computer; it's just harder for a human being to visualize.

This code is used in section 5.

 $\S 7$ Random-ternary index 3

7. Index.

```
argc: \underline{1}, \underline{2}.
argv: \quad \frac{1}{2}, \quad 2.
exit: \quad 2, \quad 3.
fprintf: 2, 3.
gb\_init\_rand: 2.
gb\_unif\_rand: 3.
i: \underline{1}.
j: \underline{\underline{1}}.
k: \underline{1}.
L: \underline{\mathbf{1}}.
main: \underline{1}.
n: \underline{1}.
nmax: \underline{1}.
nn: \underline{1}, \underline{2}.
nnn: \quad \underline{1}, \ 3, \ 5, \ 6.
P: \underline{\mathbf{1}}.
p: \underline{\mathbf{1}}.
print f: 1.
q: \underline{1}.
r: \underline{1}.
sanity\_checking: \underline{3}.
seed: \underline{1}, \underline{2}.
sscanf: 2.
stderr: 2, 3.
```

 $x: \underline{1}.$

4 NAMES OF THE SECTIONS

RANDOM-TERNARY

```
 \begin{array}{lll} \left\langle \mbox{ Do the harder case } 6 \right\rangle & \mbox{Used in section 5.} \\ \left\langle \mbox{ Do the magic switcheroo 5} \right\rangle & \mbox{Used in section 3.} \\ \left\langle \mbox{ Extend a solution for } n \mbox{ to a solution for } n+1, \mbox{ and increase } n \mbox{ 3} \right\rangle & \mbox{ Used in section 1.} \\ \left\langle \mbox{ Process the command line 2} \right\rangle & \mbox{ Used in section 1.} \end{array}
```

RANDOM-TERNARY

	\ \tag{\tag{\tag{\tag{\tag{\tag{\tag{	Section	. Page
Intro		1	. 1
Index		7	,