1. Intro. Given m and n, this program does a depth-first search on the random digraph with vertices $\{1,\ldots,n\}$ that has m arcs, where each arc $u\longrightarrow v$ goes from a uniformly random vertex u to a uniformly random vertex v.

By depth-first search I mean Algorithm 7.4.1.1D. That algorithm converts a given digraph into what Tarjan called a "jungle," consisting of an oriented forest plus nontree arcs called back arcs, forward arcs, and cross arcs. My goal is to understand the distribution of those different flavors of arcs.

Actually two other parameters are given on the command line: The number of repetitions, *reps*, and the random seed, *seed*.

```
#define maxm 10000000
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gb_flip.h"
  int m, n, reps, seed;      /* command-line parameters */
  ⟨ Type definitions 8 ⟩;
  ⟨ Global variables 4 ⟩;
  ⟨ Subroutines 9 ⟩;
  main(int argc, char *argv[])
  {
    register int i, j, k, r;
    ⟨ Local variables for depth-first search 6 ⟩;
    ⟨ Process the command line 2 ⟩;
    for (r = 0; r < reps; r++) {
      ⟨ Generate the random arcs 3 ⟩;
      ⟨ Do a depth-first search 5 ⟩;
      ⟨ Update the statistics 12 ⟩;
    }
    ⟨ Print the statistics 13 ⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
  if (argc ≠ 5 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1 ∨ sscanf(argv[3], "%d",
        &reps) ≠ 1 ∨ sscanf(argv[4], "%d", &seed) ≠ 1) {
    fprintf(stderr, "Usage: %s m n reps seed\n", argv[0]);
    exit(-1);
  }
  if (m > maxm ∨ n > maxm) {
    fprintf(stderr, "Recompile me: I can only handle m,n<=%d!\n", maxm);
    exit(-2);
  }
  gb_init_rand(seed);
  printf("Depth-first search model A, seed %d.\n", seed);
```

This code is used in section 1.

3. The arcs from vertex v are $tip[k]$ for $arcs[v] \le k < arcs[v+1]$.

Uniform random numbers allow us some flexibility to mix and match. First I figure out how many arcs have a given source u, then I generate the targets.

```
⟨ Generate the random arcs 3 ⟩ ≡
  for (k = 0; k < m; k++) arcs[k] = 0;
  for (k = 0; k < m; k++) arcs[gb_unif_rand(n)]++;
  for (j = k = 0; k < n; j += i, k++) i = arcs[k], arcs[k] = j;
  if (j ≠ m) printf("I'm␣confused!\n");
  arcs[n] = j;
  for (k = 0; k < m; k++) tip[k] = gb_unif_rand(n);
```

This code is used in section 1.

4.

```
⟨ Global variables 4 ⟩ ≡
  int arcs[maxm + 1];    /* where arcs begin in the tip table */
  int tip[maxm];         /* tips of the arcs */
```

See also sections 7 and 11.

This code is used in section 1.

5.

```
⟨ Do a depth-first search 5 ⟩ ≡
d1: roots = backs = forwards = loops = crosses = maxlev = 0;
  for (w = 0; w < n; w++) par[w] = post[w] = 0;
  p = q = 0;
d2: while (w) {
    v = w = w − 1;
    if (par[v]) continue;
  d3: par[v] = n + 1, level[v] = 0, arc[v] = arcs[v], pre[v] = ++p, roots++;
  d4: if (arc[v] ≡ arcs[v + 1]) {
      post[v] = ++q, v = par[v] − 1;
      goto d8;
    }
  d5: u = tip[arc[v]++];
  d6: if (par[u]) {    /* nontree arc */
      if (pre[u] > pre[v]) forwards++;
      else if (pre[u] ≡ pre[v]) loops++;
      else if (¬post[u]) backs++;
      else crosses++;
      goto d4;
    }
  d7: par[u] = v + 1, level[u] = level[v] + 1, v = u, arc[v] = arcs[v], pre[v] = ++p;
    if (level[u] > maxlev) maxlev = level[u];
    goto d4;
  d8: if (v ≠ n) goto d4;
  }
```

This code is used in section 1.

6.

```
⟨ Local variables for depth-first search 6 ⟩ ≡
  register int a, u, v, w, p, q, roots, backs, forwards, loops, crosses, maxlev;
```

This code is used in section 1.

7. ⟨ Global variables 4 ⟩ +≡

```
int par[maxm];      /* parent pointers plus 1, or 0 */
int pre[maxm];      /* preorder index */
int post[maxm];     /* postorder index, or 0 */
int arc[maxm];      /* the current next arc to examine */
int level[maxm];    /* tree distance from the root */
```

8. Statistics. I'm keeping the usual sample mean and sample variance, using the general purpose routines that I've had on hand for more than 20 years.

```
⟨ Type definitions 8 ⟩ ≡
  typedef struct {
    double mean, var;
    int n;
  } stat;
```

This code is used in section 1.

```
9. ⟨ Subroutines 9 ⟩ ≡
  void record_stat(q, x)
      stat *q;
      int x;
  {
    register double xx = (double) x;
    if (q→n++) {
      double tmp = xx − q→mean;
      q→mean += tmp/q→n;
      q→var += tmp * (xx − q→mean);
    }
    else {
      q→mean = xx;
      q→var = 0.0;
    }
  }
```

See also section 10.

This code is used in section 1.

```
10. ⟨ Subroutines 9 ⟩ +≡
  void print_stat(q)
      stat *q;
  {
    printf("%g␣+-␣%g", q→mean, q→n > 1 ? sqrt(q→var/(q→n − 1)) : 0.0);      /* standard deviation */
  }
```

```
11. ⟨ Global variables 4 ⟩ +≡
  stat rootstat, backstat, forwardstat, loopstat, crossstat, maxlevstat;
```

```
12. ⟨ Update the statistics 12 ⟩ ≡
  record_stat(&rootstat, roots);
  record_stat(&backstat, backs);
  record_stat(&forwardstat, forwards);
  record_stat(&loopstat, loops);
  record_stat(&crossstat, crosses);
  record_stat(&maxlevstat, maxlev);
```

This code is used in section 1.

13. ⟨ Print the statistics 13 ⟩ ≡

```
printf("During␣%d␣repetitions␣with␣%d␣vertices␣and␣%d␣arcs␣I␣found\n", reps, n, m);
print_stat(&rootstat);
printf("␣roots;\n");
print_stat(&backstat);
printf("␣back␣arcs;\n");
print_stat(&forwardstat);
printf("␣nonloop␣forward␣arcs;\n");
print_stat(&loopstat);
printf("␣loops;\n");
print_stat(&crossstat);
printf("␣cross␣arcs.\n");
print_stat(&maxlevstat);
printf("␣was␣the␣maximum␣level.\n");
```

This code is used in section 1.

14. Index.

RANDOM-DFS-A