1. Intro. Given p and n, this program does a depth-first search on the random digraph with vertices $\{1,\ldots,n\}$ whose arcs from each vertex u are independently generated as follows: "With probability p, generate a new arc $u\longrightarrow v$, where v is uniformly random, and repeat this process. Otherwise stop."

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 ars, forward arcs, and cross arcs. My goal is to understand the distribution of those different flavors of arcs.

The probability p is specified as a rational number, by giving numerator and denominator (for example `5` and `6` for 5/6). And two other parameters must also be given on the command line: The number of repetitions, *reps*, and the random seed, *seed*.

```
#define maxm 100000000
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gb_flip.h"
  int pnum, pden, n, reps, seed;      /* command-line parameters */
  ⟨ Type definitions 6 ⟩;
  ⟨ Global variables 5 ⟩;
  ⟨ Subroutines 7 ⟩;
  main(int argc, char *argv[])
  {
    register int i, j, k, r;
    ⟨ Local variables for depth-first search 4 ⟩;
    ⟨ Process the command line 2 ⟩;
    for (r = 0; r < reps; r++) {
      ⟨ Do a depth-first search 3 ⟩;
      ⟨ Update the statistics 10 ⟩;
    }
    ⟨ Print the statistics 11 ⟩;
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
  if (argc ≠ 6 ∨ sscanf(argv[1], "%d", &pnum) ≠ 1 ∨ sscanf(argv[2], "%d", &pden) ≠ 1 ∨ sscanf(argv[3],
        "%d", &n) ≠ 1 ∨ sscanf(argv[4], "%d", &reps) ≠ 1 ∨ sscanf(argv[5], "%d", &seed) ≠ 1) {
    fprintf(stderr, "Usage: %s pnum pden n reps seed\n", argv[0]);
    exit(-1);
  }
  if (n > maxm) {
    fprintf(stderr, "Recompile me: I can only handle n<=%d!\n", maxm);
    exit(-2);
  }
  gb_init_rand(seed);
  printf("Depth-first search model B, probability %d/%d, seed %d.\n", pnum, pden, seed);
```

This code is used in section 1.

3. ⟨ Do a depth-first search 3 ⟩ ≡

```
d1: roots = backs = forwards = loops = crosses = maxlev = arcs = 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, pre[v] = ++p, roots++;
  d4: if (gb_unif_rand(pden) ≥ pnum) {
      post[v] = ++q, v = par[v] − 1;
      goto d8;
    }
  d5: arcs++, u = gb_unif_rand(n);
  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, 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.

4. ⟨ Local variables for depth-first search 4 ⟩ ≡

```
register int a, u, v, w, p, q, roots, backs, forwards, loops, crosses, maxlev, arcs;
```

This code is used in section 1.

5. ⟨ Global variables 5 ⟩ ≡

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

See also section 9.

This code is used in section 1.

6. 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 6 ⟩ ≡
  typedef struct {
    double mean, var;
    int n;
  } stat;
```

This code is used in section 1.

7.

```
⟨ Subroutines 7 ⟩ ≡
  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 8.

This code is used in section 1.

8.

```
⟨ Subroutines 7 ⟩ +≡
  void print_stat(q)
       stat *q;
  {
    printf("%g␣+-␣%g", q→mean, q→n > 1 ? sqrt(q→var/(q→n − 1)) : 0.0);      /* standard deviation */
  }
```

9.

```
⟨ Global variables 5 ⟩ +≡
  stat arcstat, rootstat, backstat, forwardstat, loopstat, crossstat, maxlevstat;
```

10.

```
⟨ Update the statistics 10 ⟩ ≡
  record_stat(&arcstat, arcs);
  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.

11. ⟨ Print the statistics 11 ⟩ ≡

```
printf("During␣%d␣repetitions␣with␣%d␣vertices␣I␣found\n", reps, n);
print_stat(&arcstat);
printf("␣arcs;\n");
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.

12. Index.

RANDOM-DFS-B