Important: Before reading G_RANDOM, please read or at least skim the program for GB_RAND.

1. Random graph generation. I've cut out parts of a couple of programs in the Stanford GraphBase to put together this source. You can generate several types of random graphs with this program to check the programs that you write yourself. You should read *gb_random* to understand how these graphs are generated.

2. We permit command-line options in typical UNIX style so that a variety of graphs can be studied: The user can say '`-n`⟨number⟩', '`-m`⟨number⟩', '`-M`', '`-s`', '`-d`', '`-l`⟨number⟩', '`-L`⟨number⟩', '`-S`⟨number⟩', and/or '`-G`', to change the default values of the parameters in the graph *random_graph*(*n*, *m*, *multi*, *self*, *directed*, *dist_from*, *dist_to* Look at the code below to see what these options work (sorry!). If the user wishes to generate random bipartite graphs, he or she should say '`-b`', and give the parameters '`-u`⟨number⟩' and '`-w`⟨number⟩' to specify how many vertices should be in the two vertex classes U and W (say *n1* and *n2*). This program then makes the call *random_bigraph*(*n1*, *n2*, *m*, *multi*, *dist1*, *dist2*, *min_len*, *max_len*, *seed*).

```
#include "gb_graph.h"     /* the GraphBase data structures */
#include "gb_rand.h"     /* the roget routine */
#include "gb_save.h"     /* restore_graph */
  ⟨ Preprocessor definitions ⟩
  main(argc, argv)
      int argc;     /* the number of command-line arguments */
      char *argv[];     /* an array of strings containing those arguments */
  { Graph *g;     /* the graph we will work on */
    unsigned long n = 100;     /* number of vertices desired */
    unsigned long m = 100;     /* number of arcs or edges desired */
    long multi = 0;     /* allow duplicate arcs? */
    long self = 0;     /* allow self loops? */
    long directed = 0;     /* directed graph? */
    long *dist_from = Λ;     /* distribution of arc sources */
    long *dist_to = Λ;     /* distribution of arc destinations */
    long min_len = 1, max_len = 1;     /* bounds on random lengths */
    long seed = 0;     /* random number seed */
    long save = 0;     /* whether to save the generated graph */
    long bipartite = 0;     /* whether bipartite */
    unsigned long n1 = 100;     /* number of vertices desired */
    unsigned long n2 = 100;     /* number of vertices desired */
    long *dist1 = Λ;     /* distribution on first vertex class */
    long *dist2 = Λ;     /* distribution on second vertex class */
    ⟨ Scan the command-line options 3 ⟩;
    if (bipartite) g = random_bigraph(n1, n2, m, multi, dist1, dist2, min_len, max_len, seed);
    else g = random_graph(n, m, multi, self, directed, dist_from, dist_to, min_len, max_len, seed);
    if (g ≡ Λ) {
      fprintf(stderr, "Sorry,␣can't␣create␣the␣graph!␣(error␣code␣%ld)\n", panic_code);
      return −1;
    }
    ⟨ Print the vertices and edges of g 5 ⟩;
    if (save) save_graph(g, "random.gb");     /* generate an ASCII file for g */
    return 0;     /* normal exit */
  }
```

3. ⟨ Scan the command-line options 3 ⟩ ≡

```
while (−− argc) {
  if (sscanf(argv[argc], "-n%lu", &n) ≡ 1) ;
  else if (sscanf(argv[argc], "-u%lu", &n1) ≡ 1) ;
  else if (sscanf(argv[argc], "-w%lu", &n2) ≡ 1) ;
  else if (sscanf(argv[argc], "-m%lu", &m) ≡ 1) ;
  else if (strcmp(argv[argc], "-M") ≡ 0) multi = 1;
  else if (strcmp(argv[argc], "-s") ≡ 0) self = 1;
  else if (strcmp(argv[argc], "-d") ≡ 0) directed = 1;
  else if (sscanf(argv[argc], "-l%ld", &min_len) ≡ 1) ;
  else if (sscanf(argv[argc], "-L%ld", &max_len) ≡ 1) ;
  else if (sscanf(argv[argc], "-S%ld", &seed) ≡ 1) ;
  else if (strcmp(argv[argc], "-G") ≡ 0) save = 1;
  else if (strcmp(argv[argc], "-b") ≡ 0) bipartite = 1;
  else {
    fprintf(stderr,
        "Usage:␣%s␣[-nN][-mN][-M][-s][-d][-lN][-LN][-SN][-G][-b][-n1N][-n2N]\n", argv[0]);
    return −2;
  }
}
```

This code is used in section 2.

4. Printing out the graph. We print out the graph in a rather simple way: we just print the adjacency lists. For simplicity, the vertices are identified with the integers $0, \ldots, n-1$, where n is the number of vertices in the graph g.

5. ⟨ Print the vertices and edges of g 5 ⟩ ≡

```
if (g ≡ Λ) printf("Something␣went␣wrong␣(panic␣code␣%ld)!\n", panic_code);
else {
  register Vertex *v;    /* current vertex being visited */
  printf("The␣graph␣whose␣official␣name␣is\n\n␣␣%s\n\n", g→id);
  if (directed) printf("has␣%ld␣vertices␣and␣%ld␣arcs:\n\n", g→n, g→m);
  else printf("has␣%ld␣vertices␣and␣%ld␣edges:\n\n", g→n, g→m/2);
  for (v = g→vertices; v < g→vertices + g→n; v++) {
    register Arc *a;    /* current arc from v */
    printf("\n%ld:", v − g→vertices);
    for (a = v→arcs; a; a = a→next) printf("␣%ld", a→tip − g→vertices);
  }
  printf("\n");
}
```

This code is used in section 2.

6. Index. We close with a list that shows where the identifiers of this program are defined and used.

a: 5.
Arc: 5.
arcs: 5.
argc: 2, 3.
argv: 2, 3.
bipartite: 2, 3.
directed: 2, 3, 5.
dist_from: 2.
dist_to: 2.
dist1: 2.
dist2: 2.
fprintf: 2, 3.
g: 2.
Graph: 2.
id: 5.
m: 2.
main: 2.
max_len: 2, 3.
min_len: 2, 3.
multi: 2, 3.
n: 2.
next: 5.
n1: 2, 3.
n2: 2, 3.
panic_code: 2, 5.
printf: 5.
random_bigraph: 2.
random_graph: 2.
restore_graph: 2.
roget: 2.
save: 2, 3.
save_graph: 2.
seed: 2, 3.
self: 2, 3.
sscanf: 3.
stderr: 2, 3.
strcmp: 3.
tip: 5.
UNIX dependencies: 2, 3.
v: 5.
Vertex: 5.
vertices: 5.

⟨ Print the vertices and edges of g 5 ⟩ Used in section 2.
⟨ Scan the command-line options 3 ⟩ Used in section 2.

G_RANDOM

	Section	Page
Random graph generation	1	1
Printing out the graph	4	3
Index	6	4

This file in **not** part of the Stanford GraphBase. I have, however, copied parts of the programs in the GraphBase (I just put together some parts of two different programs and edited the result a bit).