Important: Before reading G_LIVROS, please read or at least skim the program for GB_BOOKS.

1. Some graphs for block decomposition. I've cut out parts of a couple of programs in the Stanford GraphBase to put together this source. You can generate some interesting graphs with this program to check your programs for finding cut vertices and for producing block decompositions of graphs. You should read *gb_books* to understand how these graphs are generated. The accompanying program *g_livros_simples* generates the same graphs as this program, but prints out the vertices with numerical identifiers. Try both programs to understand what I mean.

2. We permit command-line options in typical UNIX style so that a variety of graphs can be studied: The user can say '`-t`⟨title⟩', '`-n`⟨number⟩', '`-x`⟨number⟩', '`-f`⟨number⟩', '`-l`⟨number⟩', '`-i`⟨number⟩', '`-o`⟨number⟩', and/or '`-s`⟨number⟩' to change the default values of the parameters in the graph generated by $book(t, n, x, f, l, i, o, s)$. In case the user wishes to generate a bipartite graph, he or she should say '`-b`'; with this parameter, the graph will be generated by $bi\_book(t, n, x, f, l, i, o, s)$.

The `-v` and `-V` options will tell you more about the vertices in the graph. The `-V` option prints a fuller explanation than `-v`; it also shows each character's weighted number of appearances.

The special command-line option `-g`⟨filename⟩ overrides all others. It substitutes an external graph previously saved by *save_graph* for the graphs produced by *book*.

```
#include "gb_graph.h"     /* the GraphBase data structures */
#include "gb_books.h"     /* the book routine */
#include "gb_io.h"     /* the imap_chr routine */
#include "gb_save.h"     /* restore_graph */
  ⟨Subroutines 4⟩
  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 */
    char *t = "anna";     /* the book to use */
    unsigned long n = 0;     /* the desired number of vertices (0 means infinity) */
    unsigned long x = 0;     /* the number of major characters to exclude */
    unsigned long f = 0;     /* the first chapter to include */
    unsigned long l = 0;     /* the last chapter to include (0 means infinity) */
    long i = 1;     /* the weight for appearances in selected chapters */
    long o = 1;     /* the weight for appearances in unselected chapters */
    long s = 0;     /* the random number seed */
    long bipartite = 0;     /* whether to use bi_book() */
    ⟨Scan the command-line options 3⟩;
    if (filename) g = restore_graph(filename);
    else if (bipartite) g = bi_book(t, n, x, f, l, i, o, s);
    else g = book(t, n, x, f, l, i, o, s);
    if (g ≡ Λ) {
      fprintf(stderr, "Sorry,␣can't␣create␣the␣graph!␣(error␣code␣%ld)\n", panic_code);
      return −1;
    }
    if (verbose) ⟨Print the cast of selected characters 5⟩;
    ⟨Print the vertices and edges of g 7⟩
    return 0;     /* normal exit */
  }
```

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

```
while (−− argc) {
  if (strncmp(argv[argc], "-t", 2) ≡ 0) t = argv[argc] + 2;
  else if (sscanf(argv[argc], "-n%lu", &n) ≡ 1) ;
  else if (sscanf(argv[argc], "-x%lu", &x) ≡ 1) ;
  else if (sscanf(argv[argc], "-f%lu", &f) ≡ 1) ;
  else if (sscanf(argv[argc], "-l%lu", &l) ≡ 1) ;
  else if (sscanf(argv[argc], "-i%ld", &i) ≡ 1) ;
  else if (sscanf(argv[argc], "-o%ld", &o) ≡ 1) ;
  else if (sscanf(argv[argc], "-s%ld", &s) ≡ 1) ;
  else if (strcmp(argv[argc], "-v") ≡ 0) verbose = 1;
  else if (strcmp(argv[argc], "-V") ≡ 0) verbose = 2;
  else if (strcmp(argv[argc], "-b") ≡ 0) bipartite = 1;
  else if (strncmp(argv[argc], "-g", 2) ≡ 0) filename = argv[argc] + 2;
  else {
    fprintf(stderr,
        "Usage:␣%s␣[-ttitle][-nN][-xN][-fN][-lN][-iN][-oN][-sN][-v][-V][-b][-gfoo]\n",
        argv[0]);
    return −2;
  }
}
if (filename) verbose = 0;
```

This code is used in section 2.

4. ⟨ Subroutines 4 ⟩ ≡

```
char *filename = Λ;     /* external graph to be restored */
char code_name[3][3];

char *vertex_name(v, i)     /* return (as a string) the name of vertex v */
    Vertex *v;
    char i;     /* i should be 0, 1, or 2 to avoid clash in code_name array */
{
  if (filename) return v→name;     /* not a book graph */
  code_name[i][0] = imap_chr(v→short_code/36);
  code_name[i][1] = imap_chr(v→short_code % 36);
  return code_name[i];
}
```

This code is used in section 2.

5. ⟨ Print the cast of selected characters 5 ⟩ ≡

```
{
  register Vertex *v;

  printf("The␣characters␣(vertices)␣are:\n\n");
  for (v = g→vertices; v < g→vertices + g→n; v++) {
    if (verbose ≡ 1) printf("%s=%s\n", vertex_name(v, 0), v→name);
    else printf("%s=%s,␣%s␣[weight␣%ld]\n", vertex_name(v, 0), v→name, v→desc,
          i * v→in_count + o * v→out_count);
  }
  printf("\n");
}
```

This code is used in section 2.

6. Printing out the graph. We print out the graph in a rather simple way: we just print the adjacency lists. Unlike *g_livros_simples*, this program prints out the names of the vertices in full.

7. ⟨ Print the vertices and edges of *g* 7 ⟩ ≡

```
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);
  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("%s\n", v→name);
    for (a = v→arcs; a; a = a→next) printf("␣␣->␣%s\n", a→tip→name);
  }
  printf("\n");
}
```

This code is used in section 2.

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

June 1, 2007 at 11:01

G_LIVROS

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).