Important: Before reading GB_ROGET, please read or at least skim the programs for GB_GRAPH and GB_IO.

1. Introduction. This GraphBase module contains the *roget* subroutine, which creates a family of graphs based on Roget's Thesaurus. An example of the use of this procedure can be found in the demo program ROGET_COMPONENTS.

⟨ `gb_roget.h` 1 ⟩ ≡
  extern Graph $*roget()$;

See also section 12.

2. The subroutine call $roget(n, min\_distance, prob, seed)$ constructs a graph based on the information in `roget.dat`. Each vertex of the graph corresponds to one of the 1022 categories in the 1879 edition of Peter Mark Roget's *Thesaurus of English Words and Phrases*, edited by John Lewis Roget. An arc goes from one category to another if Roget gave a reference to the latter among the words and phrases of the former, or if the two categories were directly related to each other by their positions in Roget's book. For example, the vertex for category 312 ('ascent') has arcs to the vertices for categories 224 ('obliquity'), 313 ('descent'), and 316 ('leap'), because Roget gave explicit cross-references from 312 to 224 and 316, and because category 312 was implicitly paired with 313 in his scheme.

The constructed graph will have $\min(n, 1022)$ vertices; however, the default value $n = 1022$ is substituted when $n = 0$. If n is less than 1022, the n categories will be selected at random, and all arcs to unselected categories will be omitted. Arcs will also be omitted if they correspond to categories whose numbers differ by less than $min\_distance$. For example, if $min\_distance > 1$, the arc between categories 312 and 313 will not be included. (Roget sometimes formed clusters of three interrelated categories; to avoid cross-references within all such clusters, you can set $min\_distance = 3$.)

If $prob > 0$, arcs that would ordinarily be included in the graph are rejected with probability $prob/65536$. This provides a way to obtain sparser graphs.

The vertices will appear in random order. However, all "randomness" in GraphBase graphs is reproducible; it depends only on the value of a given *seed*, which can be any nonnegative integer less than 2^{31}. For example, everyone who asks for $roget(1000, 3, 32768, 50)$ will obtain exactly the same graph, regardless of their computer system.

Changing the value of *prob* will affect only the arcs of the generated graph; it will change neither the choice of vertices nor the vertex order.

#define `MAX_N` 1022 /* the number of categories in Roget's book */

3. If the *roget* routine encounters a problem, it returns Λ (`NULL`), after putting a code number into the external variable *panic_code*. This code number identifies the type of failure. Otherwise *roget* returns a pointer to the newly created graph, which will be represented with the data structures explained in GB_GRAPH. (The external variable *panic_code* is itself defined in GB_GRAPH.)

#define $panic(c)$ { $panic\_code = c$; $gb\_trouble\_code = 0$; **return** Λ; }

4. The C file `gb_roget.c` has the following general shape:

```
#include "gb_io.h"      /* we will use the GB_IO routines for input */
#include "gb_flip.h"      /* we will use the GB_FLIP routines for random numbers */
#include "gb_graph.h"      /* and we will use the GB_GRAPH data structures */
  ⟨Preprocessor definitions⟩
  ⟨Private variables 7⟩
  Graph *roget(n, min_distance, prob, seed)
      unsigned long n;      /* number of vertices desired */
      unsigned long min_distance;      /* smallest inter-category distance allowed in an arc */
      unsigned long prob;      /* 65536 times the probability of rejecting an arc */
      long seed;      /* random number seed */
  { ⟨Local variables 5⟩
    gb_init_rand(seed);
    if (n ≡ 0 ∨ n > MAX_N) n = MAX_N;
    ⟨Set up a graph with n vertices 6⟩;
    ⟨Determine the n categories to use in the graph 8⟩;
    ⟨Input roget.dat and build the graph 10⟩;
    if (gb_trouble_code) {
      gb_recycle(new_graph);
      panic(alloc_fault);      /* oops, we ran out of memory somewhere back there */
    }
    return new_graph;
  }
```

5. ⟨Local variables 5⟩ ≡

```
  Graph *new_graph;      /* the graph constructed by roget */
```

See also section 9.

This code is used in section 4.

6. Vertices.

⟨ Set up a graph with n vertices 6 ⟩ ≡

```
new_graph = gb_new_graph(n);
if (new_graph ≡ Λ) panic(no_room);   /* out of memory before we're even started */
sprintf(new_graph→id, "roget(%lu,%lu,%lu,%ld)", n, min_distance, prob, seed);
strcpy(new_graph→util_types, "IZZZZZZZZZZZZZ");
```

This code is used in section 4.

7. The first nontrivial thing we need to do is find a random selection and permutation of n vertices. We will compute a *mapping* table such that *mapping*[k] is non-Λ for exactly n randomly selected category numbers k. Moreover, these non-Λ values will be a random permutation of the vertices of the graph.

⟨ Private variables 7 ⟩ ≡

```
static Vertex *mapping[MAX_N + 1];   /* the vertex corresponding to a given category */
static long cats[MAX_N];   /* table of category numbers that have not yet been used */
```

This code is used in section 4.

8. During the loop on v in this step, k is the number of categories whose *mapping* value is still Λ. The first k entries of *cats* will contain those category numbers in some order.

⟨ Determine the n categories to use in the graph 8 ⟩ ≡

```
for (k = 0; k < MAX_N; k++) cats[k] = k + 1, mapping[k + 1] = Λ;
for (v = new_graph→vertices + n − 1; v ≥ new_graph→vertices; v−−) {
  j = gb_unif_rand(k);
  mapping[cats[j]] = v;
  cats[j] = cats[−−k];
}
```

This code is used in section 4.

9. ⟨ Local variables 5 ⟩ +≡

```
register long j, k;   /* all-purpose indices */
register Vertex *v;   /* current vertex */
```

10. Arcs. The data in `roget.dat` appears in 1022 lines, one for each category. For example, the line

```
312ascent:224 313 316
```

specifies the arcs from category 312 as explained earlier. First comes the category number, then the category name, then a colon, then zero or more numbers specifying arcs to other categories; the numbers are separated by spaces.

Some categories have too many arcs to fit on a single line; the data for these categories can be found on two lines, the first line ending with a backslash and the second line beginning with a space.

```
⟨Input roget.dat and build the graph 10⟩ ≡
  if (gb_open("roget.dat") ≠ 0) panic(early_data_fault);
      /* couldn't open "roget.dat" using GraphBase conventions */
  for (k = 1; ¬gb_eof(); k++)
    ⟨Read the data for category k, and put it in the graph if it has been selected 11⟩;
  if (gb_close() ≠ 0) panic(late_data_fault);  /* something's wrong with "roget.dat"; see io_errors */
  if (k ≠ MAX_N + 1) panic(impossible);  /* we don't have the right value of MAX_N */
```

This code is used in section 4.

11. We check that the data isn't garbled, except that we don't bother to look at unselected categories.

The original category number is stored in vertex utility field *cat_no*, in case anybody wants to see it.

```
#define cat_no u.I   /* utility field u of each vertex holds the category number */
⟨Read the data for category k, and put it in the graph if it has been selected 11⟩ ≡
  {
    if (mapping[k]) {  /* yes, this category has been selected */
      if (gb_number(10) ≠ k) panic(syntax_error);  /* out of synch */
      (void) gb_string(str_buf, ':');
      if (gb_char() ≠ ':') panic(syntax_error + 1);  /* no colon found */
      v = mapping[k];
      v→name = gb_save_string(str_buf);
      v→cat_no = k;
      ⟨Add arcs from v for every category that's both listed on the line and selected 13⟩;
    } else ⟨Skip past the data for one category 14⟩;
  }
```

This code is used in section 10.

12. ⟨gb_roget.h 1⟩ +≡

```
#define cat_no u.I   /* definition of cat_no is repeated in the header file */
```

13. **#define** *iabs*(x) ((x) < 0 ? −(x) : (x))

⟨ Add arcs from v for every category that's both listed on the line and selected 13 ⟩ ≡

```
j = gb_number(10);
if (j ≡ 0) goto done;    /* some categories lead to no arcs at all */
while (1) {
  if (j > MAX_N) panic(syntax_error + 2);    /* category code out of range */
  if (mapping[j] ∧ iabs(j − k) ≥ min_distance ∧ (prob ≡ 0 ∨ ((gb_next_rand() ≫ 15) ≥ prob)))
    gb_new_arc(v, mapping[j], 1L);
  switch (gb_char()) {
  case '\\': gb_newline();
    if (gb_char() ≠ '␣') panic(syntax_error + 3);    /* space should begin a continuation line */
        /* fall through to the space case */
  case '␣': j = gb_number(10); break;
  case '\n': goto done;
  default: panic(syntax_error + 4);    /* illegal character following category number */
  }
}
done: gb_newline();
```

This code is used in section 11.

14. We want to call *gb_newline*( ) twice if the current line ends with a backslash; otherwise we want to call it just once. There's an obvious way to do that, and there's also a faster and trickier way. The author apologizes here for succumbing to some old-fashioned impulses. (Recall that *gb_string* returns the location just following the '\0' it places at the end of a scanned string.)

⟨ Skip past the data for one category 14 ⟩ ≡

```
{
  if (*(gb_string(str_buf, '\n') − 2) ≡ '\\') gb_newline();    /* the first line ended with backslash */
  gb_newline();
}
```

This code is used in section 11.

15. Index. Here is a list that shows where the identifiers of this program are defined and used.

GB_ROGET

Preliminary work on the Stanford GraphBase project was supported in part by National Science Foundation grant CCR-86-10181.