Important: Before reading WORD_COMPONENTS, please read or at least skim the program for GB_WORDS.

1. Components. This simple demonstration program computes the connected components of the GraphBase graph of five-letter words. It prints the words in order of decreasing weight, showing the number of edges, components, and isolated vertices present in the graph defined by the first n words for all n.

```
#include "gb_graph.h"    /* the GraphBase data structures */
#include "gb_words.h"    /* the words routine */
  ⟨Preprocessor definitions⟩
  main()
  { Graph *g = words(0L, 0L, 0L, 0L);    /* the graph we love */
    Vertex *v;    /* the current vertex being added to the component structure */
    Arc *a;    /* the current arc of interest */
    long n = 0;    /* the number of vertices in the component structure */
    long isol = 0;    /* the number of isolated vertices in the component structure */
    long comp = 0;    /* the current number of components */
    long m = 0;    /* the current number of edges */
    printf("Component␣analysis␣of␣%s\n", g→id);
    for (v = g→vertices; v < g→vertices + g→n; v++) {
      n++, printf("%4ld:␣%5ld␣%s", n, v→weight, v→name);
      ⟨Add vertex v to the component structure, printing out any components it joins 2⟩;
      printf(";␣c=%ld,i=%ld,m=%ld\n", comp, isol, m);
    }
    ⟨Display all unusual components 5⟩;
    return 0;    /* normal exit */
  }
```

2. The arcs from v to previous vertices all appear on the list $v\to arcs$ after the arcs from v to future vertices. In this program, we aren't interested in the future, only the past; so we skip the initial arcs.

```
⟨Add vertex v to the component structure, printing out any components it joins 2⟩ ≡
  ⟨Make v a component all by itself 3⟩;
  a = v→arcs;
  while (a ∧ a→tip > v) a = a→next;
  if (¬a) printf("[1]");    /* indicate that this word is isolated */
  else { long c = 0;    /* the number of merge steps performed because of v */
    for ( ; a; a = a→next) { register Vertex *u = a→tip;
      m++;
      ⟨Merge the components of u and v, if they differ 4⟩;
    }
    printf("␣in␣%s[%ld]", v→master→name, v→master→size);    /* show final component */
  }
```

This code is used in section 1.

3. We keep track of connected components by using circular lists, a procedure that is known to take average time $O(n)$ on truly random graphs [Knuth and Schönhage, *Theoretical Computer Science* **6** (1978), 281–315].

Namely, if v is a vertex, all the vertices in its component will be in the list

$$v,\ v\to link,\ v\to link\to link,\ \ldots,$$

eventually returning to v again. There is also a master vertex in each component, $v\to master$; if v is the master vertex, $v\to size$ will be the number of vertices in its component.

```
#define link z.V    /* link to next vertex in component (occupies utility field z) */
#define master y.V    /* pointer to master vertex in component */
#define size x.I    /* size of component, kept up to date for master vertices only */
```

⟨ Make v a component all by itself 3 ⟩ ≡

```
v→link = v;
v→master = v;
v→size = 1;
isol++;
comp++;
```

This code is used in section 2.

4. When two components merge together, we change the identity of the master vertex in the smaller component. The master vertex representing v itself will change if v is adjacent to any prior vertex.

⟨ Merge the components of u and v, if they differ 4 ⟩ ≡

```
u = u→master;
if (u ≠ v→master) { register Vertex *w = v→master, *t;
  if (u→size < w→size) {
    if (c++ > 0) printf("%s␣%s[%ld]", (c ≡ 2 ? "␣with" : ","), u→name, u→size);
    w→size += u→size;
    if (u→size ≡ 1) isol--;
    for (t = u→link; t ≠ u; t = t→link) t→master = w;
    u→master = w;
  } else {
    if (c++ > 0) printf("%s␣%s[%ld]", (c ≡ 2 ? "␣with" : ","), w→name, w→size);
    if (u→size ≡ 1) isol--;
    u→size += w→size;
    if (w→size ≡ 1) isol--;
    for (t = w→link; t ≠ w; t = t→link) t→master = u;
    w→master = u;
  }
  t = u→link;
  u→link = w→link;
  w→link = t;
  comp--;
}
```

This code is used in section 2.

5. The *words* graph has one giant component and lots of isolated vertices. We consider all other components unusual, so we print them out when the other computation is done.

```
⟨ Display all unusual components 5 ⟩ ≡
  printf("\nThe␣following␣non-isolated␣words␣didn't␣join␣the␣giant␣component:\n");
  for (v = g→vertices; v < g→vertices + g→n; v++)
    if (v→master ≡ v ∧ v→size > 1 ∧ v→size + v→size < g→n) { register Vertex *u;
      long c = 1;     /* count of number printed on current line */
      printf("%s", v→name);
      for (u = v→link; u ≠ v; u = u→link) {
        if (c++ ≡ 12) putchar('\n'), c = 1;
        printf("␣%s", u→name);
      }
      putchar('\n');
    }
```

This code is used in section 1.

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

WORD_COMPONENTS

© 1993 Stanford University

This file may be freely copied and distributed, provided that no changes whatsoever are made. All users are asked to help keep the Stanford GraphBase files consistent and "uncorrupted," identical everywhere in the world. Changes are permissible only if the modified file is given a new name, different from the names of existing files in the Stanford GraphBase, and only if the modified file is clearly identified as not being part of that GraphBase. (The CWEB system has a "change file" facility by which users can easily make minor alterations without modifying the master source files in any way. Everybody is supposed to use change files instead of changing the files.) The author has tried his best to produce correct and useful programs, in order to help promote computer science research, but no warranty of any kind should be assumed.

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