1* Components. This simple demonstration program computes the largest connected component of the GraphBase graph of five-letter words, and saves it in file `word_giant.gb`. It modifies the edge lengths so that alphabetic distances are used (as in the `-a` option of LADDERS).

```
#include "gb_graph.h"      /* the GraphBase data structures */
#include "gb_save.h"      /* the save_graph routine */
#include "gb_basic.h"      /* the induced routine */
#include "gb_words.h"      /* the words routine */
  ⟨ Preprocessor definitions ⟩
  main( )
  { Graph *g = words(0_L, 0_L, 0_L, 0_L);      /* 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 */
    for (v = g→vertices; v < g→vertices + g→n; v++) {
      n++;
      ⟨ Add vertex v to the component structure, and change the lengths of edges that connect it to
          previous vertices 2* ⟩;
    }
    ⟨ Mark all vertices of the giant component 5* ⟩;
    save_graph(induced(g, "giant", 0, 0, 0), "word_giant.gb");
    return 0;      /* normal exit */
  }
```

2* The arcs from v to previous vertices all appear on the list v→*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, and change the lengths of edges that connect it to previous
      vertices 2* ⟩ ≡
  ⟨ Make v a component all by itself 3 ⟩;
  a = v→arcs;
  while (a ∧ a→tip > v) a = a→next;
  for ( ; a; a = a→next) { register Vertex *u = a→tip;
    register int k = a→loc;      /* where the words differ */
    register char *p = v→name + k, *q = u→name + k;
    if (*p < *q) a→len = (a − 1)→len = *q − *p;
    else a→len = (a − 1)→len = *p − *q;      /* alphabetic distance */
    m++;
    ⟨ Merge the components of u and v, if they differ 4* ⟩;
  }
```

This code is used in section 1*.

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) {
      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 (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 set the *ind* field to 1 in the giant component, so that the *induced* routine will retain those vertices.

```
⟨ Mark all vertices of the giant component 5* ⟩ ≡
  for (v = g→vertices; v < g→vertices + g→n; v++)
    if (v→master→size + v→master→size < g→n) v→ind = 0;
    else v→ind = 1;
```

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.

The following sections were changed by the change file: 1, 2, 4, 5, 6.

WORD_GIANT

This program was obtained by modifying WORD_COMPONENTS in the Stanford GraphBase.
Only sections that have changed are listed here.