Important: Before reading GIRTH, please read or at least skim the program for GB_RAMAN.

1. Introduction. This demonstration program uses graphs constructed by the *raman* procedure in the GB_RAMAN module to produce an interactive program called `girth`, which computes the girth and diameter of a class of Ramanujan graphs.

The girth of a graph is the length of its shortest cycle; the diameter is the maximum length of a shortest path between two vertices. A Ramanujan graph is a connected, undirected graph in which every vertex has degree $p+1$, with the property that every eigenvalue of its adjacency matrix is either $\pm(p+1)$ or has absolute value $\le 2\sqrt{p}$.

Exact values for the girth are of interest because the bipartite graphs produced by *raman* apparently have larger girth than any other known family of regular graphs, even if we consider graphs whose existence is known only by nonconstructive methods, except for the cubic "sextet" graphs of Biggs, Hoare, and Weiss [*Combinatorica* **3** (1983), 153–165; **4** (1984), 241–245].

Exact values for the diameter are of interest because the diameter of any Ramanujan graph is at most twice the minimum possible diameter of any regular graph.

The program will prompt you for two numbers, p and q. These should be distinct prime numbers, not too large, with $q > 2$. A graph is constructed in which each vertex has degree $p + 1$. The number of vertices is $(q^3 - q)/2$ if p is a quadratic residue modulo q, or $q^3 - q$ if p is not a quadratic residue. In the latter case, the graph is bipartite and it is known to have rather large girth.

If $p = 2$, the value of q is further restricted to be of the form $104k + (1, 3, 9, 17, 25, 27, 35, 43, 49, 51, 75, 81)$. This means that the only feasible values of q to go with $p = 2$ are probably 3, 17, and 43; the next case, $q = 107$, would generate a bipartite graph with 1,224,936 vertices and 3,674,808 arcs, thus requiring approximately 113 megabytes of memory (not to mention a nontrivial amount of computer time). If you want to compute the girth and diameter of Ramanujan graphs for large p and/or q, much better methods are available based on number theory; the present program is merely a demonstration of how to interface with the output of *raman*. Incidentally, the graph for $p = 2$ and $q = 43$ turns out to have 79464 vertices, girth 20, and diameter 22.

The program will examine the graph and compute its girth and its diameter, then will prompt you for another choice of p and q.

2. Here is the general layout of this program, as seen by the C compiler:

```
#include "gb_graph.h"    /* the standard GraphBase data structures */
#include "gb_raman.h"    /* Ramanujan graph generator */
  ⟨ Preprocessor definitions ⟩
  ⟨ Global variables 3 ⟩
  main( )
  {
    printf("This␣program␣explores␣the␣girth␣and␣diameter␣of␣Ramanujan␣graphs.\n");
    printf("The␣bipartite␣graphs␣have␣q^3-q␣vertices,␣and␣the␣non-bipartite\n");
    printf("graphs␣have␣half␣that␣number.␣Each␣vertex␣has␣degree␣p+1.\n");
    printf("Both␣p␣and␣q␣should␣be␣odd␣prime␣numbers;\n");
    printf("␣␣or␣you␣can␣try␣p␣=␣2␣with␣q␣=␣17␣or␣43.\n");
    while (1) {
      ⟨ Prompt the user for p and q; break if unsuccessful 4 ⟩;
      g = raman(p, q, 0L, 0L);
      if (g ≡ Λ) ⟨ Explain that the graph could not be constructed 5 ⟩
      else {
        ⟨ Print the theoretical bounds on girth and diameter of g 10 ⟩;
        ⟨ Compute and print the true girth and diameter of g 12 ⟩;
        gb_recycle(g);
      }
    }
    return 0;    /* normal exit */
  }
```

3. ⟨ Global variables 3 ⟩ ≡

```
  Graph *g;    /* the current Ramanujan graph */
  long p;    /* the branching factor (degree minus one) */
  long q;    /* cube root of the graph size */
  char buffer[16];    /* place to collect what the user types */
```

See also section 11.

This code is used in section 2.

4.

```
#define prompt(s)
        { printf(s); fflush(stdout);    /* make sure the user sees the prompt */
          if (fgets(buffer, 15, stdin) ≡ Λ) break; }
```

⟨ Prompt the user for p and q; **break** if unsuccessful 4 ⟩ ≡

```
  prompt("\nChoose␣a␣branching␣factor,␣p:␣");
  if (sscanf(buffer, "%ld", &p) ≠ 1) break;
  prompt("OK,␣now␣choose␣the␣cube␣root␣of␣graph␣size,␣q:␣");
  if (sscanf(buffer, "%ld", &q) ≠ 1) break;
```

This code is used in section 2.

5. ⟨ Explain that the graph could not be constructed 5 ⟩ ≡

```
printf("␣Sorry,␣I␣couldn't␣make␣that␣graph␣(%s).\n",
      panic_code ≡ very_bad_specs ? "q␣is␣out␣of␣range" :
      panic_code ≡ very_bad_specs + 1 ? "p␣is␣out␣of␣range" :
      panic_code ≡ bad_specs + 5 ? "q␣is␣too␣big" :
      panic_code ≡ bad_specs + 6 ? "p␣is␣too␣big" :
      panic_code ≡ bad_specs + 1 ? "q␣isn't␣prime" :
      panic_code ≡ bad_specs + 7 ? "p␣isn't␣prime" :
      panic_code ≡ bad_specs + 3 ? "p␣is␣a␣multiple␣of␣q" :
      panic_code ≡ bad_specs + 2 ? "q␣isn't␣compatible␣with␣p=2" :
      "not␣enough␣memory");
```

This code is used in section 2.

6. Bounds. The theory of Ramanujan graphs allows us to predict the girth and diameter to within a factor of 2 or so.

In the first place, we can easily derive an upper bound on the girth and a lower bound on the diameter, valid for any n-vertex regular graph of degree $p+1$. Such a graph has at most $(p+1)p^{k-1}$ points at distance k from any given vertex; this implies a lower bound on the diameter d:

$$1 + (p+1) + (p+1)p + (p+1)p^2 + \cdots + (p+1)p^{d-1} \;\ge\; n.$$

Similarly, if the girth g is odd, say $g = 2k+1$, the points at distance $\le k$ from any vertex must be distinct, so we have

$$1 + (p+1) + (p+1)p + (p+1)p^2 + \cdots + (p+1)p^{k-1} \;\le\; n;$$

and if $g = 2k+2$, at least p^k further points must exist at distance $k+1$, because the $(p+1)p^k$ paths of length $k+1$ can end at a particular vertex at most $p+1$ times. Thus

$$1 + (p+1) + (p+1)p + (p+1)p^2 + \cdots + (p+1)p^{k-1} + p^k \;\le\; n$$

when the girth is even.

In the following code we let $pp = p^{dl}$ and $s = 1 + (p+1) + \cdots + (p+1)p^{dl}$.

⟨ Compute the "trivial" bounds gu and dl on girth and diameter 6 ⟩ ≡

```
s = p + 2; dl = 1; pp = p; gu = 3;
while (s < n) {
  s += pp;
  if (s ≤ n) gu++;
  dl++;
  pp *= p;
  s += pp;
  if (s ≤ n) gu++;
}
```

This code is used in section 10.

7. When $p > 2$, we can use the theory of integral quaternions to derive a lower bound on the girth of the graphs produced by *raman*. A path of length g from a vertex to itself exists if and only if there is an integral quaternion $\alpha = a_0 + a_1 i + a_2 j + a_3 k$ of norm p^g such that the a's are not all multiples of p, while a_1, a_2, and a_3 are multiples of q and $a_0 \not\equiv a_1 \equiv a_2 \equiv a_3 \pmod 2$. This means we have integers (a_0, a_1, a_2, a_3) with

$$a_0^2 + a_1^2 + a_2^2 + a_3^2 = p^g,$$

satisfying the stated properties mod q and mod 2. If a_1, a_2, and a_3 are even, they cannot all be zero so we must have $p^g \ge 1 + 4q^2$; if they are odd, we must have $p^g \ge 4 + 3q^2$. (The latter is possible only when g is odd and $p \bmod 4 = 3$.) Since n is roughly proportional to q^3, this means g must be at least about $\frac{2}{3}\log_p n$. Thus g isn't too much less than the maximum girth possible in any regular graph, which we have shown is at most about $2\log_p n$.

When the graph is bipartite we can, in fact, prove that g is approximately $\frac{4}{3}\log_p n$. The bipartite case occurs if and only if p is not a quadratic residue modulo q; hence the number g in the previous paragraph must be even, say $g = 2r$. Then $p^g \bmod 4 = 1$, and a_0 must be odd. The congruence $a_0^2 \equiv p^{2r} \pmod{q^2}$ implies that $a_0 \equiv \pm p^r$, because all numbers relatively prime to q^2 are powers of a primitive root. We can assume without loss of generality that $a_0 = p^r - 2mq^2$, where $0 < m < p^r/q^2$; it follows in particular that $p^r > q^2$. Conversely, if $p^r - q^2$ can be written as a sum of three squares $b_1^2 + b_2^2 + b_3^2$, then $p^{2r} = (p^r - 2q^2)^2 + (2b_1q)^2 + (2b_2q)^2 + (2b_3q)^2$ is a representation of the required type. If $p^r - q^2$ is a positive integer that cannot be represented as a sum of three squares, a well-known theorem of Legendre tells us that $p^r - q^2 = 4^t s$, where $s \equiv 7 \pmod 8$. Since p and q are odd, we have $t \ge 1$; hence $p^r - 2q^2$ is odd. If $p^r - 2q^2$ is a positive odd integer, Legendre's theorem tells us that we can write $2p^r - 4q^2 = b_1^2 + b_2^2 + b_3^2$; hence $p^{2r} = (p^r - 4q^2)^2 + (2b_1q)^2 + (2b_2q)^2 + (2b_3q)^2$. We conclude that the girth is either $2\lceil \log_p q^2 \rceil$ or $2\lceil \log_p 2q^2 \rceil$. (This explicit calculation, which makes our program for calculating the girth unnecessary or at best redundant in the bipartite case, is due to G. A. Margulis and, independently, to Biggs and Boshier [*Journal of Combinatorial Theory* **B49** (1990), 190–194].)

A girth of 1 or 2 can occur, since these graphs might have self-loops or multiple edges if p is sufficiently large.

```
⟨Compute a lower bound gl on the girth 7⟩ ≡
  if (bipartite) { long b = q * q;
    for (gl = 1, pp = p; pp ≤ b; gl++, pp *= p) ;      /* iterate until $p^g > q^2$ */
    gl += gl;
  } else { long b1 = 1 + 4 * q * q, b2 = 4 + 3 * q * q;      /* bounds on $p^g$ */
    for (gl = 1, pp = p; pp < b1; gl++, pp *= p) {
      if (pp ≥ b2 ∧ (gl & 1) ∧ (p & 2)) break;
    }
  }
```

This code is used in section 10.

8. Upper bounds on the diameter of any Ramanujan graph can be derived as shown in the paper by Lubotzky, Phillips, and Sarnak in *Combinatorica* **8** (1988), page 275. (However, a slight correction to their proof is necessary—their parameter l should be odd when x and y lie in different parts of a bipartite graph.) Their argument demonstrates that $p^{(d-1)/2} < 2n$ in the nonbipartite case and $p^{(d-2)/2} < n$ in the bipartite case; therefore we obtain the upper bound $d \le 2\log_p n + O(1)$, which is about twice the lower bound that holds in an arbitrary regular graph.

```
⟨Compute an upper bound du on the diameter 8⟩ ≡
  { long nn = (bipartite ? n : 2 * n);
    for (du = 0, pp = 1; pp < nn; du += 2, pp *= p) ;
    ⟨Decrease du by 1, if $pp/nn \ge \sqrt{p}$ 9⟩;
    if (bipartite) du++;
  }
```

This code is used in section 10.

9. Floating point arithmetic might not be accurate enough for the test required in this section. We avoid it by using an all-integer method analogous to Euclid's algorithm, based on the continued fraction for $\sqrt{p}$ [*Seminumerical Algorithms*, exercise 4.5.3–12]. In the loop here we want to compare nn/pp to $(\sqrt{p}+a)/b$, where $\sqrt{p}+a>b>0$ and $p-a^2$ is a multiple of b.

```
⟨ Decrease du by 1, if pp/nn ≥ √p 9 ⟩ ≡
  { long qq = pp/nn;
    if (qq * qq > p) du−−;
    else if ((qq + 1) * (qq + 1) > p) {     /* qq = ⌊√p⌋ */
      long aa = qq, bb = p − aa * aa, parity = 0;
      pp −= qq * nn;
      while (1) {
        long x = (aa + qq)/bb, y = nn − x * pp;
        if (y ≤ 0) break;
        aa = bb * x − aa;     /* now 0 < aa < √p */
        bb = (p − aa * aa)/bb;
        nn = pp; pp = y;
        parity ⊕= 1;
      }
      if (¬parity) du−−;
    }
  }
```

This code is used in section 8.

10.
```
⟨ Print the theoretical bounds on girth and diameter of g 10 ⟩ ≡
  n = g→n;
  if (n ≡ (q + 1) * q * (q − 1)) bipartite = 1;
  else bipartite = 0;
  printf("The␣graph␣has␣%ld␣vertices,␣each␣of␣degree␣%ld,␣and␣it␣is␣%sbipartite.\n", n, p + 1,
      bipartite ? "" : "not␣");
  ⟨ Compute the "trivial" bounds gu and dl on girth and diameter 6 ⟩;
  printf("Any␣such␣graph␣must␣have␣diameter␣>=␣%ld␣and␣girth␣<=␣%ld;\n", dl, gu);
  ⟨ Compute an upper bound du on the diameter 8 ⟩;
  printf("theoretical␣considerations␣tell␣us␣that␣this␣one's␣diameter␣is␣<=␣%ld", du);
  if (p ≡ 2) printf(".\n");
  else {
    ⟨ Compute a lower bound gl on the girth 7 ⟩;
    printf(",\nand␣its␣girth␣is␣>=␣%ld.\n", gl);
  }
```

This code is used in section 2.

11. We had better declare all the variables we've been using so freely.

```
⟨ Global variables 3 ⟩ +≡
  long gl, gu, dl, du;     /* theoretical bounds */
  long pp;     /* power of p */
  long s;     /* accumulated sum */
  long n;     /* number of vertices */
  char bipartite;     /* is the graph bipartite? */
```

12. Breadth-first search. The graphs produced by *raman* are symmetrical, in the sense that there is an automorphism taking any vertex into any other. Each vertex V and each edge P corresponds to a 2×2 matrix, and the path $P_1P_2 \ldots P_k$ leading from vertex V to vertex $VP_1P_2 \ldots P_k$ has the same properties as the path leading from vertex U to vertex $UP_1P_2 \ldots P_k$. Therefore we can find the girth and the diameter by starting at any vertex v_0.

We compute the number of points at distance k from v_0 for all k, by explicitly forming a linked list of all such points. Utility field *link* is used for the links. The lists terminate with a non-null *sentinel* value, so that we can also use the condition $link \equiv \Lambda$ to tell if a vertex has been encountered before. Another utility field, *dist*, contains the distance from the starting point, and *back* points to a vertex one step closer.

```
#define link w.V    /* the field where we store links, initially Λ */
#define dist v.I    /* the field where we store distances, initially 0 */
#define back u.V    /* the field where we store backpointers, initially Λ */

⟨Compute and print the true girth and diameter of g 12⟩ ≡
  printf("Starting␣at␣any␣given␣vertex,␣there␣are\n");
  { long k;    /* current distance being generated */
    long c;    /* how many we've seen so far at this distance */
    register Vertex *v;    /* current vertex in list at distance k − 1 */
    register Vertex *u;    /* head of list for distance k */
    Vertex *sentinel = g→vertices + n;    /* nonzero link at end of lists */
    long girth = 999;    /* length of smallest cycle found, initially infinite */

    k = 0;
    u = g→vertices;
    u→link = sentinel;
    c = 1;
    while (c) {
      for (v = u, u = sentinel, c = 0, k++; v ≠ sentinel; v = v→link) ⟨Place all vertices adjacent to v onto
            list u, unless they've been encountered before, increasing c whenever the list grows 13⟩;
      printf("%8ld␣vertices␣at␣distance␣%ld%s\n", c, k, c > 0 ? "," : ".");
    }
    printf("So␣the␣diameter␣is␣%ld,␣and␣the␣girth␣is␣%ld.\n", k − 1, girth);
  }
```

This code is used in section 2.

13. ⟨Place all vertices adjacent to v onto list u, unless they've been encountered before, increasing c whenever the list grows 13⟩ ≡

```
  { register Arc *a;
    for (a = v→arcs; a; a = a→next) { register Vertex *w;    /* vertex adjacent to v */
      w = a→tip;
      if (w→link ≡ Λ) {
        w→link = u;
        w→dist = k;
        w→back = v;
        u = w;
        c++;
      } else if (w→dist + k < girth ∧ w ≠ v→back) girth = w→dist + k;
    }
  }
```

This code is used in section 12.

14. Index. Finally, here's a list that shows where the identifiers of this program are defined and used.

January 30, 2026 at 12:17

GIRTH

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