1. Intro. What's the lexicographically smallest solution to the ∞-queens problem? I mean, consider the sequence q_0, q_1, …, where q_n is the least nonnegative integer not in the sets $\{q_k \mid 0 \le k < n\}$, $\{q_k + k - n \mid 0 \le k < n\}$, $\{q_k - k + n \mid 0 \le k < n\}$.

Inspired by Eq. 7.2.2–(6), I maintain bit vectors a, b, c to record the occurrences of q_k, $q_k + k$, and $q_k - k$, for previously calculated values. Thus, for example, we'll have $c_r = 1$ if and only if $q_k - k = r$ for some $k < n$, when I'm calculating q_n. The value of q_n will be the smallest $k > 0$ such that $a_k = 0$ and $b_{k+n} = 0$ and $c_{k-n} = 0$.

It turns out (as conjectured by Neil Sloane in 2016, and proved by him and Jeffrey Shallit shortly thereafter) that this sequence has lots of beautiful structure, which greatly facilitates the computation. Let ϕ be the golden ratio. Then the subscripts of a will range from 0 to approximately ϕn; the subscripts of b will range from 0 to approximately $\phi^2 n$; and the subscripts of c will range from approximately $-\phi^{-2}n$ to approximately $\phi^{-1}n$. In particular, since c has n bits equal to 1, and $\phi^{-1}n + \phi^{-2}n = n$, almost all the bits of c will be equal to 1. Furthermore, a will begin with a string of 1s, whose length is approximately $\phi^{-1}n$. Therefore we only need to look at a bounded number of bits when we're computing q_n.

Nice, huh?

This implementation uses one byte per bit. Of course I could make n eight times larger by packing the bits.

Another feature is an *exact* computation of the discrepancies between q_n and ϕn or $\phi^{-1}n$. For example, this program "knows" that $q_{F_{40}} = F_{41} = \phi F_{40} + \phi^{-40}$.

```
#define slack 10     /* approximation when we allocate memory */
#define phi 1.61803398874989484820
#define tickmax 25     /* I hope to need at most this many ticks per round */
#define deltamax 10
#define o ticks++
#define pausethresh 999999995

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
  int goal;     /* command-line parameter */
  char *a, *b, *c;
  long long int maxmema, maxmemb, minmemc, maxmemc;
  int ticks;
  int tickhist[tickmax + 1];     /* histogram of run times */
  int deltalominint, deltalomaxint, deltahiminint, deltahimaxint;
  long long deltalominfrac, deltalomaxfrac, deltahiminfrac, deltahimaxfrac;
  int deltalomin[deltamax + 1], deltalomax[deltamax + 1], deltahimin[deltamax + 1], deltahimax[deltamax + 1];
  ⟨Subroutines 7⟩;
  main(int argc, char *argv[])
  {
    register int j;
    register long long k, n, q, r, s, t, nphiint, nphifrac;
    ⟨Process the command line 2⟩;
    ⟨Allocate the arrays 3⟩;
    r = t = 0, s = 1;
    for (n = nphiint = nphifrac = 1; n ≤ goal; n++) {
      ⟨Determine q = q_n, or goto done if out of memory 4⟩;
      printf("%lld\n", q);
      ⟨Record statistics about q 8⟩;
      ⟨Advance nphiint and nphifrac 6⟩;
    }
  done: ⟨Print the final stats 9⟩;
```

```
}
```

2. ⟨ Process the command line 2 ⟩ ≡

```
if (argc ≠ 2 ∨ sscanf(argv[1], "%d", &goal) ≠ 1) {
   fprintf(stderr, "Usage:␣%s␣n\n", argv[0]);
   exit(−1);
}
```

This code is used in section 1.

3. ⟨ Allocate the arrays 3 ⟩ ≡

```
maxmema = ((int)(phi ∗ goal) + slack);
maxmemb = (maxmema + goal);
maxmemc = (maxmema − goal);
minmemc = (goal − maxmemc + 2 ∗ slack);
a = (char ∗) calloc(maxmema, sizeof(char));
if (¬a) {
   fprintf(stderr, "Can't␣allocate␣array␣a!\n");
   exit(−2);
}
b = (char ∗) calloc(maxmemb, sizeof(char));
if (¬b) {
   fprintf(stderr, "Can't␣allocate␣array␣b!\n");
   exit(−2);
}
c = (char ∗) calloc(minmemc + maxmemc, sizeof(char));
if (¬c) {
   fprintf(stderr, "Can't␣allocate␣array␣c!\n");
   exit(−2);
}
```

This code is used in section 1.

4. In this algorithm, s is the least positive integer such that $a_s = 0$; t is the greatest integer such that $t = 0$ or $c_t = 1$; r is the greatest nonnegative integer $\le t$ such that $c_{-r} = 0$.

⟨ Determine $q = q_n$, or **goto** *done* if out of memory 4 ⟩ ≡

```
ticks = 0;
for (k = s; k ≤ n − r; k++) {
  if (k + n ≥ maxmemb) goto done;
  if (o, b[k + n] ≡ 0) {
    if (k − n + minmemc < 0) goto done;
    if (o, c[k − n + minmemc] ≡ 0) {
      if (o, a[k] ≡ 0) {
        q = k;
        o, a[k] = 1;
        if (k ≡ s)
          for (s = k + 1; o, a[s] ≡ 1; s++) ;
        o, b[k + n] = 1;
        o, c[k − n + minmemc] = 1;
        if (k − n ≡ −r)
          for (r = n − k + 1; ; r++) {
            if (r > minmemc) goto done;
            if (o, c[minmemc − r] ≡ 0) break;
          }
        goto got_q;
      }
    }
  }
}
t++;
if (t ≥ maxmemc) goto done;
o, c[t + minmemc] = 1;
q = n + t;
if (q ≥ maxmema) goto done;
o, a[q] = 1;
if (q + n ≥ maxmemb) goto done;
o, b[q + n] = 1; got_q:
```

This code is used in section 1.

5. I had special fun writing the next part of this program, which expresses the value of $n\phi$ as an integer plus $\sum_{k\ge1} x_k\phi^{-k}$, with $x_kx_{k+1} = 0$ for all k. For example, $9\phi = 14 + \phi^{-2} + \phi^{-4} + \phi^{-7}$. We maintain the integer part in *nphiint*, and the fractional part in *nphifrac*, where the latter is the *binary* integer $(\ldots x_3x_2x_1)_2$.

This fractional part has a nice connection with the *negaFibonacci number system*, which is described in equation 7.1.3–(147) of *The Art of Computer Programming*. For example, $9 = F_{-7} + F_{-4} + F_{-2}$ is the negaFibonacci representation of 9; hence we have $9\phi = (F_6 - F_4 - F_2)\phi = F_7 - F_5 - F_3 - ((-\phi)^{-7} + (-\phi)^{-4} + (-\phi)^{-2} = 14 + \phi^{-7} + \phi^{-4} + \phi^{-2}$.

Furthermore, equation 7.1.3–(149) shows a nice way to go from the negaFibonacci representation of n to its successor. And exercise 7.1.3–45 shows that it's surprisingly easy to compare the fractional parts, even if they are ordered lexicographically from right to left instead of from left to right(!).

6. ⟨ Advance *nphiint* and *nphifrac* 6 ⟩ ≡

```
nphiint++;
if (nphifrac & #3) nphiint++;
{
  register long long y, z;
  y = nphifrac ⊕ #aaaaaaaaaaaaaaaa;
  z = y ⊕ (y + 1);
  z = z | (nphifrac & (z ≪ 1));
  nphifrac ⊕= z ⊕ ((z + 1) ≫ 2);
}
```

This code is used in section 1.

7. ⟨ Subroutines 7 ⟩ ≡

```
int compfrac(long long x, long long y)
{
  register int long long d = (x − y) & (y − x);   /* Rockicki's hack */
  return ((d & y) ≠ 0);   /* 1 if x^R < y^R, 0 otherwise */
}
```

See also section 10.

This code is used in section 1.

8. ⟨ Record statistics about q 8 ⟩ ≡

```
if (n > pausethresh) debug("watch␣me␣now");
if (ticks ≥ tickmax) tickhist[tickmax]++;
else tickhist[ticks]++;
if (q ≥ n) {
  if (q > nphiint) {
    if (q − nphiint > deltahimaxint ∨ (q − nphiint ≡ deltahimaxint ∧ compfrac(nphifrac, deltahimaxfrac)))
      {
      deltahimaxint = q − nphiint, deltahimaxfrac = nphifrac;
      fprintf(stderr, "n=%lld,␣deltahimax=%d,%llx\n", n, deltahimaxint, deltahimaxfrac);
    }
    j = q − nphiint − 1;
    if (j ≥ deltamax) deltahimax[deltamax]++;
    else deltahimax[j]++;
  } else {
    if (q − nphiint < deltahiminint ∨ (q − nphiint ≡ deltahiminint ∧ compfrac(deltahiminfrac, nphifrac)))
      {
      deltahiminint = q − nphiint, deltahiminfrac = nphifrac;
      fprintf(stderr, "n=%lld,␣deltahimin=%d,%llx\n", n, deltahiminint, deltahiminfrac);
    }
    j = nphiint − q;
    if (j ≥ deltamax) deltahimin[deltamax]++;
    else deltahimin[j]++;
  }
} else if (q > (nphiint − n)) {
  if (q − (nphiint − n) > deltalomaxint ∨ (q − (nphiint − n) ≡ deltalomaxint ∧ compfrac(nphifrac,
        deltalomaxfrac))) {
    deltalomaxint = q − (nphiint − n), deltalomaxfrac = nphifrac;
    fprintf(stderr, "n=%lld,␣deltalomax=%d,%llx\n", n, deltalomaxint, deltalomaxfrac);
  }
  j = q − (nphiint − n) − 1;
  if (j ≥ deltamax) deltalomax[deltamax]++;
  else deltalomax[j]++;
} else {
  if (q − (nphiint − n) < deltalominint ∨ (q − (nphiint − n) ≡ deltalominint ∧ compfrac(deltalominfrac,
        nphifrac))) {
    deltalominint = q − (nphiint − n), deltalominfrac = nphifrac;
    fprintf(stderr, "n=%lld,␣deltalomin=%d,%llx\n", n, deltalominint, deltalominfrac);
  }
  j = (nphiint − n) − q;
  if (j ≥ deltamax) deltalomin[deltamax]++;
  else deltalomin[j]++;
}
```

This code is used in section 1.

9. ⟨ Print the final stats 9 ⟩ ≡

```
fprintf(stderr, "OK,␣I␣computed␣%lld␣elements␣of␣the␣sequence.\n", n − 1);
fprintf(stderr, "tick␣histogram:");
for (j = 0; j ≤ tickmax; j++) fprintf(stderr, "␣%d", tickhist[j]);
fprintf(stderr, "\n");
fprintf(stderr, "deltalo␣histogram:");
for (j = deltamax; j ≥ 0; j−−) fprintf(stderr, "␣%d", deltalomin[j]);
fprintf(stderr, "␣|");
for (j = 0; j ≤ deltamax; j++) fprintf(stderr, "␣%d", deltalomax[j]);
fprintf(stderr, "\n");
fprintf(stderr, "deltahi␣histogram:");
for (j = deltamax; j ≥ 0; j−−) fprintf(stderr, "␣%d", deltahimin[j]);
fprintf(stderr, "␣|");
for (j = 0; j ≤ deltamax; j++) fprintf(stderr, "␣%d", deltahimax[j]);
fprintf(stderr, "\n");
```

This code is used in section 1.

10. ⟨ Subroutines 7 ⟩ +≡

```
void debug(char *m)
{
  fprintf(stderr, "%s!\n", m);
}
```

11. Index.

INFTY-QUEENS