(See https://cs.stanford.edu/~knuth/programs.html for date.)

1. Intro. I'm trying to find a nice partitioning of the Boolean functions associated with tictactoe.

For each of the $\binom{18}{6} = 18564$ choices of six bit coordinates, I compute a score as follows: Count the number of "care" positions that match each setting of the other 12 bits. The score is the sum of squares of those counts.

I minimize the score, in order to spread the cares around rather evenly.

```
#define bitcode v.I   /* binary representation of this position */
#define cases 4520
#include "gb_graph.h"
#include "gb_save.h"
  int care[cases];
  char a[1 ≪ 18];
  int count[65];
  main()
  {
    register int j, k, m, x, y, minj;
    register Graph *g = restore_graph("/tmp/tictactoe.gb");
    register Vertex *v;
    for (k = 0, v = g→vertices; v < g→vertices + g→n; v++)
      if (v→arcs) care[k++] = v→bitcode;
    if (k ≠ cases) {
      fprintf(stderr, "There␣are␣%d␣cases,␣not␣%d!\n", k, cases);
      exit(−1);
    }
    minj = #7fffffff;   /* note Gosper's hack in the following line */
    for (m = #3f; m < 1 ≪ 18; x = m & −m, y = m + x, m = y + (((y ⊕ m)/x) ≫ 2)) {
      ⟨Compute stats for mask m 2⟩;
    }
  }
```

2. ⟨Compute stats for mask m 2⟩ ≡

```
  x = #3ffff − m;
  for (k = 0; k < cases; k++) a[care[k] & x]++;
  for (k = 1; k ≤ 64; k++) count[k] = 0;
  for (j = k = 0; k < cases; k++) {
    y = a[care[k] & x];
    if (y) {
      j += y * y, count[y]++;
      a[care[k] & x] = 0;
    }
  }
  if (j ≤ minj) {
    minj = j;
    printf("%05x␣gives␣score␣%d;␣", m, j);
    for (k = 1; k ≤ 64; k++) printf("%4d", count[k]);
    printf("\n");
  }
```

This code is used in section 1.

3. Index.

⟨ Compute stats for mask m 2 ⟩ Used in section 1.

TICTACTOE5