

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

1. Intro. Sequence M1805, posets with linear order $1 \dots n$. Same as upper triangular $n \times n$ Boolean matrices B such that $I \subseteq B^2 \subseteq B$.

I'm in kind of a hurry tonight, so please excuse terseness. This prog was developed from POSET0, which does the cases up to $n = 9$ in a few seconds, but with much calculation repeated unnecessarily. Therefore I reformulated the method using dynamic programming.

We're given a list of pairs (A, w) where A is an $n \times n$ Boolean matrix and w is a positive weight. The problem is to compute the sum of w times the number of upper triangular B such that $I \subseteq B^2 \subseteq B \subseteq \overline{A}$. The solution is to go through that list and generate all first rows of each A , creating a new list with n decreased by 1.

An $n \times n$ upper triangular Boolean matrix is represented in $\binom{n}{2}$ bits, because we don't care about the diagonal. We want to be able to go up to $n = 12$ at least, on my 32-bit computer, so this program is set up to handle multiple precision. It operates in two phases: Double precision for matrices at first, then single precision when n has been reduced. But in the latter case we need double precision for the weight w .

```
#define n 12
#define memsize 36000000 /* should be a multiple of 12 */
#define thresh (1 << 7) /* boundary between phases; maximum is 1 << 7 */
#define hashsize (1 << 21) /* should be a power of 2 */
#include <stdio.h>
#include "gb_flip.h"
typedef unsigned int uint;
uint mem[memsize]; /* memory pool */
int *hash[1 << 21]; /* heads of hash lists or direct data pointers */
short uni[4][256]; /* random bits for universal hashing */
uint *top, *start, *ostart; /* key places in mem */
int offset[1 << (n - 1)]; /* table showing how bits are mapped */
int t, x, y, z, mask, hmask;
int count, curN;

main()
{
    register int j, k, l, curbits, curaux;
    uint *p;
    <Initialize 2>;
    <Do the first phase 5>;
    <Do the second phase 7>;
}
```

2. First we initialize the *uni* table, for hashing.

```
<Initialize 2> ≡
gb_init_rand(0);
for (j = 0; j < 4; j++)
    for (k = 1; k < 256; k++) uni[j][k] = gb_next_rand();
```

See also section 4.

This code is used in section 1.

3. If *thresh* is $1 \ll k$, the *bits* field will contain rows having k or fewer bits, and the *aux* field will contain the longer rows.

4. $\langle \text{Initialize } 2 \rangle + \equiv$

```
for ( $j = 0, k = 1, l = 2; l \leq \text{thresh}; k++, l \ll = 1$ )  $\text{offset}[l - 1] = j, j += k;$ 
for ( $j = 0; l < 1 \ll n; k++, l \ll = 1$ )  $\text{offset}[l - 1] = j, j += k;$ 
```

5. Data is kept sequentially in *mem*, beginning at *start*; the first available location is *top*. During the first phase the data appears in four-word packets, because we want link fields for hashing.

```
#define wt(p) *p /* first word of packet */
#define aux(p) *(p + 1) /* second word of packet */
#define bits(p) *(p + 2) /* third word of packet */
#define link(p) *(p + 3) /* fourth word of packet (phase one only) */

 $\langle \text{Do the first phase } 5 \rangle \equiv$ 
start = &mem[0];
wt(start) = 1, aux(start) = bits(start) = 0, link(start) = (uint)  $\Lambda$ ;
top = start + 4;
for ( $l = (1 \ll (n - 1)) - 1, \text{curn} = n; l > \text{thresh}; l \gg = 1, \text{curn}--$ ) {
    hmask = (1  $\ll$  offset[l]) - 1;
    for ( $j = 0; j < \text{hashsize}; j++$ ) hash[j] =  $\Lambda$ ;
    count = 0;
    for ( $p = \text{start}, \text{start} = \text{top}; p \neq \text{start}; p = (p \equiv \&\text{mem}[\text{memsize} - 4] ? \&\text{mem}[0] : p + 4)$ ) {
        count++;
        mask = (aux(p)  $\gg$  offset[l]) & l;
        for ( $x = 0; x \leq l; x = ((x \mid \text{mask}) + 1) \& \sim \text{mask}$ ) {
            curbits = bits(p);
            curaux = aux(p);
            for ( $y = x \& (x + 1), t = x \oplus -1; y; y -= z$ ) {
                z = y & -y;
                if ( $z \leq \text{thresh}$ ) curbits |= (t & (z - 1))  $\ll$  offset[z - 1];
                else curaux |= (t & (z - 1))  $\ll$  offset[z - 1];
            }
             $\langle \text{Put curbits and curaux into the new list with weight } w \text{ } 6 \rangle;$ 
        }
    }
    printf(" $\_ \%d\_ \text{item} \%s\_ \text{on} \_ \text{list} \_ \%d; \backslash n$ ", count, count > 1 ? "s" : "", curn);
}
```

This code is used in section 1.

6. \langle Put *curbits* and *curaux* into the new list with weight *w* 6 $\rangle \equiv$

```

{
  register int h;
  register uint *q;
  curaux &= hmask;
  h = uni[0][curbits &#xff] + uni[1][(curbits >> 8) &#xff] + uni[2][(curbits >> 16) &#xff] + uni[3][curbits >> 24];
  h += uni[0][curaux &#xff] + uni[1][(curaux >> 8) &#xff] + uni[2][(curaux >> 16) &#xff] + uni[3][curaux >>
    24];
  h &= hashsize - 1;
  for (q = hash[h]; q; q = (uint *) link(q))
    if (bits(q) &#x26; curbits &#x26; aux(q) &#x26; curaux) goto found;
  q = top;
  if (q &#x26; p) {
    fprintf(stderr, "Sorry, I need more memory!\n");
    exit(-1);
  }
  bits(q) = curbits, aux(q) = curaux, wt(q) = 0;
  link(q) = (uint) hash[h], hash[h] = q;
  top = q + 4;
  if (top &#x26; &mem[memsize]) top = &mem[0];
found: wt(q) += wt(p);
}

```

This code is used in section 5.

7. In the second phase we use the *hash* table as a direct pointer to the data.

\langle Do the second phase 7 $\rangle \equiv$

\langle Repack the data into shorter packets 9 \rangle ;

```

for (; l >>= 1, curn--) {
  hmask = (1 << offset[l]) - 1;
  for (j = 0; j <= hmask; j++) hash[j] = 0;
  count = 0;
  for (p = start, start = top; p &#x2260; start; p = (p &#x26; &mem[memsize - 3] ? &mem[0] : p + 3)) {
    count++;
    mask = (bits(p) >> offset[l]) &#x26; l;
    for (x = 0; x <= l; x = ((x | mask) + 1) &#x26; ~mask) {
      curbits = bits(p);
      for (y = x &#x26; (x + 1), t = x &#x2225; -1; y; y -= z) {
        z = y &#x26; -y;
        curbits |= (t &#x26; (z - 1)) << offset[z - 1];
      }
       $\langle$  Put curbits into the new list with weight w 8  $\rangle$ ;
    }
  }
  printf("%d items on list %d\n", count, curn);
}
printf("...and the solution for %d is %d%09d.\n", n, aux(start), wt(start));

```

This code is used in section 1.

8. $\langle \text{Put } \textit{curbits} \text{ into the new list with weight } w \text{ 8} \rangle \equiv$

```

{
  register uint *q;
  y = curbits & hmask;
  q = hash[y];
  if (¬q) {
    q = top;
    if (q ≡ p) {
      fprintf(stderr, "Sorry, I need more memory!\n");
      exit(-2);
    }
    bits(q) = curbits, wt(q) = aux(q) = 0;
    hash[y] = q;
    top = q + 3;
    if (top ≡ &mem[memsize]) top = &mem[0];
  }
  wt(q) += wt(p), aux(q) += aux(p);
  if (wt(q) ≥ 1000000000) aux(q) += 1, wt(q) -= 1000000000;
}

```

This code is used in section 7.

9. $\langle \text{Repack the data into shorter packets 9} \rangle \equiv$

```

ostart = top;
x = (top - mem) % 3;
if (x) top += 3 - x;
for (p = start, start = top; p ≠ ostart; p = (p ≡ &mem[memsize - 4] ? &mem[0] : p + 4)) {
  wt(top) = wt(p), aux(top) = 0, bits(top) = bits(p);
  top += 3;
  if (top ≡ &mem[memsize]) top = &mem[0];
}

```

This code is used in section 7.

10. Index.

aux: 3, [5](#), 6, 7, 8, 9.

bits: 3, [5](#), 6, 7, 8, 9.

count: [1](#), [5](#), 7.

curaux: [1](#), [5](#), 6.

curbits: [1](#), [5](#), 6, 7, 8.

curn: [1](#), [5](#), 7.

exit: 6, 8.

found: [6](#).

fprintf: 6, 8.

gb_init_rand: 2.

gb_next_rand: 2.

h: [6](#).

hash: [1](#), [5](#), 6, 7, 8.

hashsize: [1](#), [5](#), 6.

hmask: [1](#), [5](#), 6, 7, 8.

j: [1](#).

k: [1](#).

l: [1](#).

link: [5](#), 6.

main: [1](#).

mask: [1](#), [5](#), 7.

mem: [1](#), [5](#), 6, 7, 8, 9.

memsize: [1](#), [5](#), 6, 7, 8, 9.

n: [1](#).

offset: [1](#), 4, [5](#), 7.

ostart: [1](#), 9.

p: [1](#).

printf: 5, 7.

q: [6](#), [8](#).

start: [1](#), [5](#), 7, 9.

stderr: 6, 8.

t: [1](#).

thresh: [1](#), 3, 4, 5.

top: [1](#), [5](#), 6, 7, 8, 9.

uint: [1](#), [5](#), 6, 8.

uni: [1](#), 2, 6.

wt: [5](#), 6, 7, 8, 9.

x: [1](#).

y: [1](#).

z: [1](#).

- ⟨ Do the first phase 5 ⟩ Used in section 1.
- ⟨ Do the second phase 7 ⟩ Used in section 1.
- ⟨ Initialize 2, 4 ⟩ Used in section 1.
- ⟨ Put *curbits* and *curaux* into the new list with weight w 6 ⟩ Used in section 5.
- ⟨ Put *curbits* into the new list with weight w 8 ⟩ Used in section 7.
- ⟨ Repack the data into shorter packets 9 ⟩ Used in section 7.

POSETS

	Section	Page
Intro	1	1
Index	10	5