

1. Intro. Given m , n , and t , I calculate the number of matrices with $0 \leq a_{i,j} < t$ for $0 \leq i < m$ and $0 \leq j < n$ whose histoscape is a three-valent polyhedron.

(More generally, this program evaluates all matrices such that the $(m-1)(n-1)$ submatrices

$$\begin{pmatrix} a_{i-1,j-1} & a_{i-1,j} \\ a_{i,j-1} & a_{i,j} \end{pmatrix}$$

for $1 \leq i < m$ and $1 \leq j < n$ are not “bad,” where badness is an arbitrary relation.)

The enumeration is by dynamic programming, using an auxiliary matrix of t^{n+1} 64-bit counts. (If necessary, I’ll use double precision floating point, but this version uses unsigned integers.)

It’s better to have $m \geq n$. But I’ll try some cases with $m < n$ too, for purposes of testing.

```
#define maxn 10
#define maxt 16
#define o mems++
#define oo mems += 2
#define ooo mems += 3
#include <stdio.h>
#include <stdlib.h>
int m, n, t; /* command-line parameters */
char bad[maxt][maxt][maxt][maxt]; /* is a submatrix bad? */
unsigned long long *count; /* the big array of counts */
unsigned long long newcount[maxt]; /* counts that will replace old ones */
unsigned long long mems; /* memory references to octabytes */
int inx[maxn+1]; /* indices being looped over */
int tpow[maxn+2]; /* powers of t */
main(int argc, char *argv[])
{
    register int a, b, c, d, i, j, k, p, q, r, pp;
    <Process the command line 2>;
    <Compute the bad table 3>;
    for (i = 1; i < m; i++)
        for (j = 1; j < n; j++) <Handle constraint (i,j) 5>;
    <Print the grand total 7>;
}
```

2. $\langle \text{Process the command line 2} \rangle \equiv$

```

if ( $argc \neq 4 \vee sscanf(argv[1], "%d", \&m) \neq 1 \vee sscanf(argv[2], "%d", \&n) \neq 1 \vee sscanf(argv[3], "%d", \&t) \neq 1$ )
{
    fprintf(stderr, "Usage: %s m n t\n", argv[0]);
    exit(-1);
}
if ( $m < 2 \vee m > maxn \vee n < 2 \vee n > maxn$ ) {
    fprintf(stderr, "Sorry, m and n should be between 2 and %d!\n", maxn);
    exit(-2);
}
if ( $t < 2 \vee t > maxt$ ) {
    fprintf(stderr, "Sorry, t should be between 2 and %d!\n", maxt);
    exit(-3);
}
for ( $j = 1, k = 0; k \leq n + 1; k++$ )  $tpow[k] = j, j *= t;$ 
 $count = (\text{unsigned long long} *) \text{malloc}(tpow[n + 1] * \text{sizeof}(\text{unsigned long long}));$ 
if ( $\neg count$ ) {
    fprintf(stderr, "I couldn't allocate t^%d=%d entries for the counts!\n", n + 1, tpow[n + 1]);
    exit(-4);
}

```

This code is used in section 1.

3. $\langle \text{Compute the bad table 3} \rangle \equiv$

```

for ( $a = 0; a < t; a++$ )
    for ( $b = 0; b \leq a; b++$ )
        for ( $c = 0; c \leq b; c++$ )
            for ( $d = 0; d \leq a; d++$ ) {
                if ( $d > b$ ) goto nogood;
                if ( $a > b \wedge c > d$ ) goto nogood;
                if ( $a > b \wedge b \equiv d \wedge d > c$ ) goto nogood;
                continue;
            }
    nogood:  $bad[a][b][c][d] = 1;$ 
     $bad[a][c][b][d] = 1;$ 
     $bad[b][d][a][c] = 1;$ 
     $bad[b][a][d][c] = 1;$ 
     $bad[d][c][b][a] = 1;$ 
     $bad[d][b][c][a] = 1;$ 
     $bad[c][a][d][b] = 1;$ 
     $bad[c][d][a][b] = 1;$ 
}

```

This code is used in section 1.

4. Throughout the main computation, I'll keep the value of p equal to $(inx[n] \dots inx[1]inx[0])_t$.

$\langle \text{Increase the } inx \text{ table, keeping } inx[q] = 0 \ 4 \rangle \equiv$

```

for ( $r = 0; r \leq n; r++$ )
    if ( $r \neq q$ ) {
         $ooo, inx[r]++, p += tpow[r];$ 
        if ( $inx[r] < t$ ) break;
         $oo, inx[r] = 0, p -= tpow[r + 1];$ 
    }

```

This code is used in sections 5 and 6.

5. Here's the heart of the computation (the inner loop).

```

⟨ Handle constraint (i, j) 5 ⟩ ≡
{
  if (j ≡ 1) ⟨ Get set to handle constraint (i, 1) 6 ⟩
  else q = (q ≡ n ? 0 : q + 1);
  while (1) {
    o, b = (q ≡ n ? inx[0] : inx[q + 1]);
    o, c = (q ≡ 0 ? inx[n] : inx[q - 1]);
    for (d = 0; d < t; d++) o, newcount[d] = 0;
    for (o, a = 0, pp = p; a < t; a++, pp += tpow[q]) {
      for (d = 0; d < t; d++)
        if (o, -bad[a][b][c][d]) ooo, newcount[d] += count[pp];
    }
    for (o, d = 0, pp = p; d < t; d++, pp += tpow[q]) oo, count[pp] = newcount[d];
    ⟨ Increase the inx table, keeping inx[q] = 0 4 ⟩;
    if (p ≡ 0) break;
  }
  fprintf(stderr, "done with %d, %d. %lld, %lld mems\n", i, j, count[0], mems);
}

```

This code is used in section 1.

6. And here's the tricky part that keeps the inner loop easy. I don't know a good way to explain it, except to say that a hand simulation will reveal all.

```

⟨ Get set to handle constraint (i, 1) 6 ⟩ ≡
{
  if (i ≡ 1) {
    for (o, p = tpow[n + 1]; p > 0; p--) o, count[p - 1] = 1;
    q = 0;
  } else {
    q = (q ≡ n ? 0 : q + 1);
    while (1) {
      for (o, a = 0, pp = p, newcount[0] = 0; a < t; a++, pp += tpow[q]) o, newcount[0] += count[pp];
      for (a = 0, pp = p; a < t; a++, pp += tpow[q]) o, count[pp] = newcount[0];
      ⟨ Increase the inx table, keeping inx[q] = 0 4 ⟩;
      if (p ≡ 0) break;
    }
    q = (q ≡ n ? 0 : q + 1);
  }
}

```

This code is used in section 5.

7. ⟨ Print the grand total 7 ⟩ ≡

```

for (newcount[0] = 0, p = tpow[n + 1] - 1; p ≥ 0; p--) o, newcount[0] += count[p];
printf("Altogether %lld 3VPs (%lld mems).\n", newcount[0], mems);

```

This code is used in section 1.

8. Index.

a: [1](#).
argc: [1](#), [2](#).
argv: [1](#), [2](#).
b: [1](#).
bad: [1](#), [3](#), [5](#).
c: [1](#).
count: [1](#), [2](#), [5](#), [6](#), [7](#).
d: [1](#).
exit: [2](#).
fprintf: [2](#), [5](#).
i: [1](#).
inx: [1](#), [4](#), [5](#).
j: [1](#).
k: [1](#).
m: [1](#).
main: [1](#).
malloc: [2](#).
maxn: [1](#), [2](#).
maxt: [1](#), [2](#).
mems: [1](#), [5](#), [7](#).
n: [1](#).
newcount: [1](#), [5](#), [6](#), [7](#).
nogood: [3](#).
o: [1](#).
oo: [1](#), [4](#), [5](#).
ooo: [1](#), [4](#), [5](#).
p: [1](#).
pp: [1](#), [5](#), [6](#).
printf: [7](#).
q: [1](#).
r: [1](#).
sscanf: [2](#).
stderr: [2](#), [5](#).
t: [1](#).
tpow: [1](#), [2](#), [4](#), [5](#), [6](#), [7](#).

- ⟨ Compute the *bad* table 3 ⟩ Used in section 1.
- ⟨ Get set to handle constraint $(i, 1)$ 6 ⟩ Used in section 5.
- ⟨ Handle constraint (i, j) 5 ⟩ Used in section 1.
- ⟨ Increase the *inx* table, keeping $inx[q] = 0$ 4 ⟩ Used in sections 5 and 6.
- ⟨ Print the grand total 7 ⟩ Used in section 1.
- ⟨ Process the command line 2 ⟩ Used in section 1.

HISTOSCAPE-COUNT

	Section	Page
Intro	1	1
Index	8	4