

1. Intro. This program generates DLX3 data that finds all “reduced dissections” of an $m \times n$ rectangle into subrectangles.

The allowable subrectangles $[a..b] \times [c..d]$ have $0 \leq a < b \leq m$, $0 \leq c < d \leq n$; so there are $\binom{m+1}{2} \cdot \binom{n+1}{2}$ possibilities.

Furthermore we require that every $x \in (0..m)$ occurs at least once among the a ’s; also that every $y \in (0..n)$ occurs at least once among the c ’s. (Otherwise the dissection could be collapsed into a smaller one, by leaving out that coordinate value.)

[I hacked this program from MOTLEY-DLX, because I thought of that one first — although logically speaking, this one is simpler and I probably should have considered it earlier.]

```
#define maxd 36      /* maximum value for m or n */
#define encode(v) ((v) < 10 ? (v) + '0' : (v) - 10 + 'a')    /* encoding for values < 36 */
#include <stdio.h>
#include <stdlib.h>
int m, n;           /* command-line parameters */
main(int argc, char *argv[])
{
    register int a, b, c, d, j, k;
    < Process the command line 2 >;
    < Output the first line 3 >;
    for (a = 0; a < m; a++)
        for (b = a + 1; b ≤ m; b++) {
            for (c = 0; c < n; c++)
                for (d = c + 1; d ≤ n; d++) { < Output the line for [a..b] × [c..d] 4 > }
        }
}
```

```
2. < Process the command line 2 > ≡
if (argc ≠ 3 ∨ sscanf(argv[1], "%d", &m) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage: %s s m n\n", argv[0]);
    exit(-1);
}
if (m > maxd ∨ n > maxd) {
    fprintf(stderr, "Sorry, m and n must be at most %d!\n", maxd);
    exit(-2);
}
printf("|_redrect-dlx%d%d\n", m, n);
```

This code is used in section 1.

3. The main primary columns jk ensure that cell (j, k) is covered, for $0 \leq j < m$ and $0 \leq k < n$. And there are primary columns xa and yc for the at-least-once conditions.

I also include primary columns xab and yed ; these are unrestricted, so they don't affect the number of solutions. They are, however, useful for compressing the output because they name the subrectangles of a solution.

⟨ Output the first line 3 ⟩ \equiv

```

for ( $j = 0$ ;  $j < m$ ;  $j++$ )
    for ( $k = 0$ ;  $k < n$ ;  $k++$ )  $printf$  (" $\square\%c\%c$ ",  $encode(j)$ ,  $encode(k)$ );
for ( $a = 1$ ;  $a < m$ ;  $a++$ )  $printf$  (" $\square 1:\%d|x\%c$ ",  $n$ ,  $encode(a)$ );
for ( $c = 1$ ;  $c < n$ ;  $c++$ )  $printf$  (" $\square 1:\%d|y\%c$ ",  $m$ ,  $encode(c)$ );
for ( $a = 0$ ;  $a < m$ ;  $a++$ )
    for ( $b = a + 1$ ;  $b \leq m$ ;  $b++$ )  $printf$  (" $\square 0:\%d|x\%c\%c$ ",  $n$ ,  $encode(a)$ ,  $encode(b)$ );
for ( $c = 0$ ;  $c < n$ ;  $c++$ )
    for ( $d = c + 1$ ;  $d \leq n$ ;  $d++$ )  $printf$  (" $\square 0:\%d|y\%c\%c$ ",  $m$ ,  $encode(c)$ ,  $encode(d)$ );
 $printf$  (" $\backslash n$ ");

```

This code is used in section 1.

4. ⟨ Output the line for $[a..b] \times [c..d]$ 4 ⟩ \equiv

```

for ( $j = a$ ;  $j < b$ ;  $j++$ )
    for ( $k = c$ ;  $k < d$ ;  $k++$ )  $printf$  (" $\square\%c\%c$ ",  $encode(j)$ ,  $encode(k)$ );
if ( $a$ )  $printf$  (" $\square x\%c$ ",  $encode(a)$ );
if ( $c$ )  $printf$  (" $\square y\%c$ ",  $encode(c)$ );
 $printf$  (" $\square x\%c\%c\square y\%c\%c\backslash n$ ",  $encode(a)$ ,  $encode(b)$ ,  $encode(c)$ ,  $encode(d)$ );

```

This code is used in section 1.

5. Index.

a: [1](#).
argc: [1](#), [2](#).
argv: [1](#), [2](#).
b: [1](#).
c: [1](#).
d: [1](#).
encode: [1](#), [3](#), [4](#).
exit: [2](#).
fprintf: [2](#).
j: [1](#).
k: [1](#).
m: [1](#).
main: [1](#).
maxd: [1](#), [2](#).
n: [1](#).
printf: [2](#), [3](#), [4](#).
sscanf: [2](#).
stderr: [2](#).

- ⟨ Output the first line [3](#) ⟩ Used in section [1](#).
- ⟨ Output the line for $[a..b] \times [c..d]$ [4](#) ⟩ Used in section [1](#).
- ⟨ Process the command line [2](#) ⟩ Used in section [1](#).

REDRECT-DLX

	Section	Page
Intro	1	1
Index	5	3