

1. Intro. A simple program to make “random” squaregraphs, by sort of a “crocheting” technique. (Hacked in haste.)

```
#define maxn 1000
#include <stdio.h>
#include <stdlib.h>
#include "gb_flip.h"
int a[2 * maxn + 4], d[2 * maxn + 8];
int move[8 * maxn];
int count[maxn];
int seed;
int steps;

main(int argc, char *argv[])
{
    register int j, k, m, t, w;
    ⟨ Process the command line 2 ⟩;
    a[0] = 0, a[1] = 1, a[2] = 0, a[3] = 1;
    d[0] = d[1] = d[2] = d[3] = 2;
    w = 4;
    for (j = 0; j < steps; j++) {
        ⟨ Set m to the number of possible moves 3 ⟩;
        k = gb_unif_rand(m);
        ⟨ Make move k 4 ⟩;
        ⟨ Check for pairs 5 ⟩;
    }
    ⟨ Output the result 6 ⟩;
}
```

2. ⟨ Process the command line 2 ⟩ \equiv

```
if (argc  $\neq$  3  $\vee$  sscanf(argv[1], "%d", &steps)  $\neq$  1  $\vee$  sscanf(argv[2], "%d", &seed)  $\neq$  1) {
    fprintf(stderr, "Usage: %s %s %s\n", argv[0]);
    exit(-1);
}
if (steps  $\geq$  maxn) {
    fprintf(stderr, "Sorry, %s should be less than %d!\n", maxn);
    exit(-2);
}
gb_init_rand(seed);
```

This code is used in section 1.

3. ⟨ Set m to the number of possible moves 3 ⟩ \equiv

```
d[w] = d[0], d[w + 1] = d[1], a[w] = a[0], a[w + 1] = a[1];
for (m = 0; m < w; m++) move[m] = m;
for (k = 0; k < w; k++)
    if (d[k + 1] > 3) move[m++] = maxn + k;
for (k = 0; k < w; k++)
    if (d[k + 1] > 3  $\wedge$  d[k + 2] > 3) move[m++] = maxn + maxn + k;
```

This code is used in section 1.

4. $\langle \text{Make move } k \text{ 4} \rangle \equiv$
if ($\text{move}[k] < \text{maxn}$) {
 $w += 2, k = \text{move}[k];$
for ($m = w - 1; m \geq k + 2; m--$) $d[m + 2] = d[m], a[m + 2] = a[m];$
 $d[k + 3] = d[k + 1] + 1, d[k + 2] = d[k + 1] = 2, d[k] = d[k] + 1;$
 $a[k + 3] = a[k + 1], a[k + 2] = j + 2, a[k + 1] = a[k], a[k] = j + 2;$
if ($k + 3 \geq w$)
for ($t = 0; t + w \leq k + 3; t++$) $d[t] = d[w + t], a[t] = a[w + t];$
} **else if** ($\text{move}[k] < \text{maxn} + \text{maxn}$) {
 $k = \text{move}[k] - \text{maxn};$
 $d[k + 1] = 2;$
 $t = a[k + 1], a[k + 1] = a[k], a[k] = t;$
if ($k + 1 \geq w$)
for ($t = 0; t + w \leq k + 1; t++$) $d[t] = d[w + t], a[t] = a[w + t];$
} **else** {
 $k = \text{move}[k] - \text{maxn} - \text{maxn};$
for ($t = 0; t < w; t++$)
if ($a[t] \equiv a[k + 2] \wedge t \neq k + 2$) $a[t] = a[k];$
 $a[k] = a[k + 1], a[k + 1] = a[k + 3], d[k] = d[k] + 1, d[k + 1] = d[k + 3] + 1;$
 $w -= 2;$
for ($t = k + 2; t < w; t++$) $a[t] = a[t + 2], d[t] = d[t + 2];$
}
}

This code is used in section 1.

5. $\langle \text{Check for pairs 5} \rangle \equiv$
for ($k = 0; k < j + 2; k++$) $\text{count}[k] = 0;$
for ($k = 0; k < w; k++$) $\text{count}[a[k]]++;$
for ($k = 0; k < j + 2; k++$)
if ($\text{count}[k] \neq 0 \wedge \text{count}[k] \neq 2$) $\text{fprintf}(\text{stderr}, \text{"count}[\%d] \text{ is } \%d! \backslash \text{n"} , k, \text{count}[k]);$

This code is used in section 1.

6. $\langle \text{Output the result 6} \rangle \equiv$
for ($k = 0; k < w; k++$) {
 $\text{printf}(\text{"\%d"}, a[k]);$
if ($k \% 20 \equiv 19$) $\text{printf}(\text{"\n"});$
}
if ($k \% 20 \neq 0$) $\text{printf}(\text{"\n"});$

This code is used in section 1.

7. Index.

a: [1](#).
argc: [1](#), [2](#).
argv: [1](#), [2](#).
count: [1](#), [5](#).
d: [1](#).
exit: [2](#).
fprintf: [2](#), [5](#).
gb_init_rand: [2](#).
gb_unif_rand: [1](#).
j: [1](#).
k: [1](#).
m: [1](#).
main: [1](#).
maxn: [1](#), [2](#), [3](#), [4](#).
move: [1](#), [3](#), [4](#).
printf: [6](#).
seed: [1](#), [2](#).
sscanf: [2](#).
stderr: [2](#), [5](#).
steps: [1](#), [2](#).
t: [1](#).
w: [1](#).

- ⟨ Check for pairs 5 ⟩ Used in section 1.
- ⟨ Make move k 4 ⟩ Used in section 1.
- ⟨ Output the result 6 ⟩ Used in section 1.
- ⟨ Process the command line 2 ⟩ Used in section 1.
- ⟨ Set m to the number of possible moves 3 ⟩ Used in section 1.

SQUAREGRAPH-RAND

	Section	Page
Intro	1	1
Index	7	3