1. Intro. Michael Simkin defined a curious mapping from an $n \times n$ square grid to a $2N \times 2N$ square grid that has been truncated to a diamond of width $2N$, then rotated 45°. He uses this when $n \geq N^2$.

For $1 \leq i, j \leq n$, let cell (ij) of the $n \times n$ grid be the open set

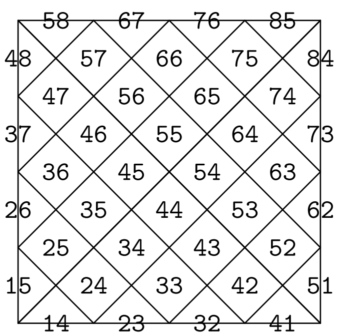
$$(ij) = \{(x, y) \mid i - 1 < nx < i,\ j - 1 < ny < j\}.$$

(Everything has been scaled down to fit in the unit square $[0\,..\,1] \times [0\,..\,1]$.

For $1 \leq I, J \leq 2N$, let Cell $[IJ]$ of the truncated-rotated $2N \times 2N$ grid be the closed set

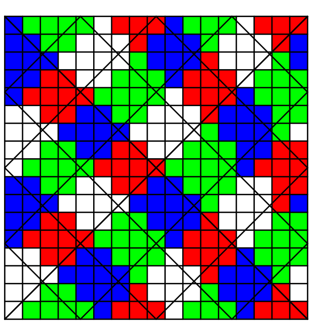
$$[IJ] = \{(x, y) \mid 0 \leq x, y \leq 1,\ I - 1 \leq N(x + y) \leq I,\ J - 1 \leq N(1 + y - x) \leq J\}.$$

(These formulas are shifted from Simkin's, but they're more convenient for programming.) Here, for example, are the Cells when $N = 4$:



Notice that each Cell $[IJ]$ is either a diamond of area $1/(2N^2)$, or an isosceles right triangle of area $1/(4N^2)$, or empty. When $I \leq N$, the nonempty Cells occur for $J = N + 1 - I$ (a triangle pointing up), $N + 1 - I < J < N + I$ (a diamond), and $J = N + I$ (a triangle pointing right). When $I > N$, the nonempty Cells occur for $J = I - N$ (a triangle pointing left), $I - N < J < 3N + 1 - I$ (a diamond), and $J = 3N + 1 - I$ (a triangle pointing down). So the number of diamonds is $0 + 2 + \cdots + (2N-2) + (2N-2) + \cdots + 2 + 0 = 2N^2 - 2N$; and the number of triangles is $4N$. Cells with constant I or J lie on the same diagonal. The center point of $[IJ]$ is $z_{IJ} = (I - J + N, I + J - N - 1)/(2N)$.

The rule for mapping $(ij) \mapsto [IJ]$ is to find the smallest I such that $(ij) \cap [IJ]$ has positive area. If there are two such $[IJ]$ with the same I, choose the one with *larger* J; it lies northwest of the other. For example, here's the mapping when $N = 4$ and $n = 17$:



To simplify calculations, I essentially construct an $nN \times nN$ grid. Each of its pixels (when shrunk down by a factor of nN to match the unit square) belongs to a unique (ij). And each pixel either belongs fully to a unique $[IJ]$, or is split on a diagonal between $[IJ]$ and $[I(J+1)]$, or is split on a diagonal between $[IJ]$ and $[(I+1)J]$, or is split by both diagonals between $[IJ]$, $[(I+1)J]$, $[I(J+1)]$, and $[(I+1)(J+1)]$.

This program outputs a METAPOST file that depicts the assignments, as in the example above.

2. The user specifies N and n on the command line, in that order.

```
#define maxN 16
#define maxn 512
#define encode(t) ((t) < 10 ? '0' + (t) : (t) < 36 ? 'a' + (t) - 10 : (t) < 62 ? 'A' + (t) - 36 : '?')
#include <stdio.h>
#include <stdlib.h>
  int N, n;     /* command-line arguments */
  int nn, Nnn;     /* n + n and N * nn */
  int ass[maxn][maxn];     /* if (ij) ↦ [IJ], ass[i - 1][j - 1] = (I ≪ 16) - J */
  int IJcount[2 * maxN][2 * maxN];
  FILE *MPfile;
  char MPfilename[64];
  ⟨Subroutines 4⟩;
  main(int argc, char *argv[])
  {
    register i, j, k, x, y;
    ⟨Process the command line 3⟩;
    ⟨Compute the assignments 5⟩;
    ⟨Output the assignments 6⟩;
    ⟨Output the many-to-one sizes 7⟩;
    ⟨Output the METAPOST file 8⟩;
  }
```

3. ⟨Process the command line 3⟩ ≡

```
  if (argc ≠ 3 ∨ sscanf(argv[1], "%d", &N) ≠ 1 ∨ sscanf(argv[2], "%d", &n) ≠ 1) {
    fprintf(stderr, "Usage:␣%s␣N␣n\n", argv[0]);
    exit(-1);
  }
  if (N < 1 ∨ N > maxN) {
    fprintf(stderr, "Recompile␣me:␣At␣present␣N␣must␣be␣between␣1␣and␣%d!\n", maxN);
    exit(-2);
  }
  if (n < 1 ∨ n > maxn) {
    fprintf(stderr, "Recompile␣me:␣At␣present␣n␣must␣be␣between␣1␣and␣%d!\n", maxn);
    exit(-2);
  }
  if (n < N * N) fprintf(stderr, "Warning:␣n␣is␣less␣than␣N^2!\n");
  nn = n + n, Nnn = N * nn;
```

This code is used in section 2.

4. Subroutine $IJset(x, xd, y, yd, i, j)$ determines the coordinates I and J that correspond to a given point $((x + xd/2, y + yd/2)/(nN)$ of the unit square, and stores them in $ass[i][j]$ in the form $(I \ll 16) - J$, unless a smaller value is already stored there.

$\langle$ Subroutines 4 $\rangle \equiv$

```
void IJset(int x, int xd, int y, int yd, int i, int j)
{
  register int I, J, acc;
  I = (x + x + xd + y + y + yd + nn)/nn;
  J = (Nnn + y + y + yd - x - x - xd + nn)/nn;
  acc = (I << 16) - J;
  if (acc < ass[i][j]) ass[i][j] = acc;
}
```

This code is used in section 2.

5. This is BRUTE FORCE.

$\langle$ Compute the assignments 5 $\rangle \equiv$

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) ass[i][j] = N << 17;   /* ∞ */
for (x = 0; x < n * N; x++)
  for (y = 0; y < n * N; y++) {
    i = x/N, j = y/N;   /* check each pixel of nN × nN grid */
    IJset(x, 0, y, 1, i, j);   /* set the Cell for (x, y + 1/2) */
    IJset(x, 2, y, 1, i, j);   /* set the Cell for (x + 1, y + 1/2) */
    IJset(x, 1, y, 0, i, j);   /* set the Cell for (x + 1/2, y) */
    IJset(x, 1, y, 2, i, j);   /* set the Cell for (x + 1/2, y + 1) */
  }
```

This code is used in section 2.

6. I give the assignments for the top row first ($j = n$), in order to mimic Cartesian coordinates instead of matrix coordinates.

```
#define Ipart(a) (((a) >> 16) + 1)
#define Jpart(a) (-(a) & #ffff)
```

$\langle$ Output the assignments 6 $\rangle \equiv$

```
for (j = n - 1; j ≥ 0; j--) {
  for (i = 0; i < n; i++) printf("␣%c%c", encode(Ipart(ass[i][j])), encode(Jpart(ass[i][j])));
  printf("\n");
}
```

This code is used in section 2.

7. $\langle$ Output the many-to-one sizes 7 $\rangle \equiv$

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) {
    k = ass[i][j];
    IJcount[Ipart(k) - 1][Jpart(k) - 1]++;
  }
for (j = N + N - 1; j ≥ 0; j--) {
  for (i = 0; i < N + N; i++) printf("%4d", IJcount[i][j]);
  printf("\n");
}
```

This code is used in section 2.

8. ⟨ Output the METAPOST file 8 ⟩ ≡

```
sprintf(MPfilename, "/tmp/queenon-partition-%d-%d.mp", N, n);
MPfile = fopen(MPfilename, "w");
if (¬MPfile) {
  fprintf(stderr, "I␣can't␣open␣file␣`%s'␣for␣writing!\n", MPfilename);
  exit(−5);
}
⟨ Output the METAPOST preamble 9 ⟩;
⟨ Output color codes for the rows 10 ⟩;
⟨ Output the METAPOST postamble 11 ⟩;
fprintf(stderr, "OK,␣I've␣written␣the␣METAPOST␣file␣`%s'.\n", MPfilename);
```

This code is used in section 2.

9. ⟨ Output the METAPOST preamble 9 ⟩ ≡

```
fprintf(MPfile, "%%␣produced␣by␣%s␣%d␣%d\n", argv[0], N, n);
fprintf(MPfile, "N=%d;␣n=%d;\n", N, n);
fprintf(MPfile, "numeric␣h,u;␣u=1cm;␣n*h=N*u;\n");
fprintf(MPfile, "primarydef␣x!y␣=␣(x*u,y*u)␣enddef;\n");
fprintf(MPfile, "\n");
fprintf(MPfile, "string␣ch;\n");
fprintf(MPfile, "picture␣pic[];\n");
fprintf(MPfile, "pic[ASCII␣\"W\"]=nullpicture;\n");
fprintf(MPfile, "currentpicture:=nullpicture;\n");
fprintf(MPfile, "fill␣(0,0)--(h,0)--(h,h)--(0,h)--cycle␣withcolor␣red;\n");
fprintf(MPfile, "pic[ASCII␣\"R\"]=currentpicture;\n");
fprintf(MPfile, "fill␣(0,0)--(h,0)--(h,h)--(0,h)--cycle␣withcolor␣blue;\n");
fprintf(MPfile, "pic[ASCII␣\"B\"]=currentpicture;\n");
fprintf(MPfile, "fill␣(0,0)--(h,0)--(h,h)--(0,h)--cycle␣withcolor␣green;\n");
fprintf(MPfile, "pic[ASCII␣\"G\"]=currentpicture;\n");
fprintf(MPfile, "currentpicture:=nullpicture;\n");
fprintf(MPfile, "\n");
fprintf(MPfile, "newinternal␣ny;\n");
fprintf(MPfile, "def␣row␣expr␣s␣=\n");
fprintf(MPfile, "␣␣ny:=ny+1;\n");
fprintf(MPfile, "␣␣for␣j=0␣upto␣length␣s-1:\n");
fprintf(MPfile, "␣␣␣␣ch:=substring(j,j+1)␣of␣s;\n");
fprintf(MPfile, "␣␣␣␣draw␣pic[ASCII␣ch]␣shifted␣(j*h,ny*h);\n");
fprintf(MPfile, "␣␣endfor\n");
fprintf(MPfile, "enddef;\n");
fprintf(MPfile, "\n");
fprintf(MPfile, "beginfig(0)\n");
fprintf(MPfile, "ny:=-1;\n");
```

This code is used in section 8.

10. ⟨ Output color codes for the rows 10 ⟩ ≡

```
for (j = n - 1; j >= 0; j--) {
  fprintf(MPfile, "row␣\"");
  for (i = 0; i < n; i++) {
    k = ass[i][j];
    switch (2 * (Ipart(k) & #1) + (Jpart(k) & #1)) {
    case 0: fprintf(MPfile, "W"); break;
    case 1: fprintf(MPfile, "R"); break;
    case 2: fprintf(MPfile, "G"); break;
    case 3: fprintf(MPfile, "B"); break;
    }
  }
  fprintf(MPfile, "\"\n");
}
```

This code is used in section 8.

11. ⟨ Output the METAPOST postamble 11 ⟩ ≡

```
fprintf(MPfile,
    "for␣i=0␣upto␣n:␣draw␣(0,i*h)--(n*h,i*h);␣draw␣(i*h,0)--(i*h,n*h);␣endfor\n");
fprintf(MPfile, "for␣i=0␣upto␣N-1:\n");
fprintf(MPfile, "␣␣draw␣0!i--(N-i)!N;\n");
fprintf(MPfile, "␣␣draw␣i!0--N!(N-i);\n");
fprintf(MPfile, "␣␣draw␣0!(N-i)--(N-i)!0;\n");
fprintf(MPfile, "␣␣draw␣i!N--N!i;\n");
fprintf(MPfile, "endfor\n");
fprintf(MPfile, "endfig;\n");
fprintf(MPfile, "bye.\n");
```

This code is used in section 8.

12. Index.

QUEENON-PARTITION