December 3, 2023 at 16:18
Changed sections for computing the crossing number.

1 The main program

3 **#define** `MAXN` 11 /* The maximum number of pseudolines for which the program will work. */

⟨ Include standard libaries 6 ⟩
⟨ Types and data structures 5 ⟩
⟨ Global variables 8 ⟩
⟨ Subroutines 24 ⟩
⟨ Core subroutine for recursive generation 14 ⟩

int *main*(**int** *argc*, **char** ∗*argv*[ ])
{
  ⟨ Parse the command line 9 ⟩;
#if *readdatabase* /* reading from the database */
  ⟨ Read all point sets of size $n\_max + 1$ from the database and process them 70 ⟩
  return 0;
#endif
#if *enumAOT*
  ⟨ Initialize statistics and open reporting file 51 ⟩;
  ⟨ Start the generation 15 ⟩;
  ⟨ Report statistics 53 ⟩;
#endif
  return 0;
}

2 Statistics

Characteristics:

- number h of hull points.
- period p of rotational symmetry on the hull. (The order of the rotation group is h/p.)
- mirror symmetry, with or without fixpoint on the hull (3 possibilities).

PSLAcount gives OAOT of point sets with a marked point on the convex hull. http://oeis.org/A006245 (see below) is the same sequence with n shifted by 0.

50 **#define** `NO_MIRROR` 0
#define `MIRROR_WITH_FIXPOINT` 1
#define `MIRROR_WITHOUT_FIXPOINT` 2

⟨ Global variables 8 ⟩ +≡
long long unsigned *countPSLA*[**MAXN** +2], *countO*[**MAXN** +2], *countU*[**MAXN** +2];
long long unsigned *PSLAcount*[**MAXN** +2]; /* A006245, Number of primitive sorting networks on n elements; also number of rhombic tilings of $2n$-gon. Also the number of oriented matroids of rank 3 on n(?) elements. */
/* 1, 1, 2, 8, 62, 908, 24698, 1232944, 112018190, 18410581880, 5449192389984 ... until $n = 15$. */
long long unsigned *xPSLAcount*[**MAXN** +2];
long long unsigned *classcount*[**MAXN** +2][**MAXN** +2][**MAXN** +2][3][`MAX_HALVING_LINES` + 1][`MAX_CROSSINGS` + 1];
int *num_halving_lines*; /* global variable; this is not clean */
long long unsigned *numComparisons* ← 0, *numTests* ← 0; /* profiling */

52 ¶ ⟨ Gather statistics about the AOT, collect output 52 ⟩ ≡ /* Determine the extreme points: */
small_int *hulledges*[**MAXN** +1];
small_int *hullsize* ← *upper_hull_PSLA*(n, *hulledges*);
small_int *rotation_period*;
boolean *has_fixpoint*;
boolean *is_symmetric*;

```
int n_points ← n + 1;   /* number of points of the AOT */
boolean lex_smallest ← is_lex_smallest_PSLA(n, hulledges, hullsize, &rotation_period, &is_symmetric,
    &has_fixpoint);
if (lex_smallest) {
  countU[n_points]++;
  if (is_symmetric) {
    countO[n_points]++;
    PSLAcount[n] += rotation_period;
    if (has_fixpoint) xPSLAcount[n] += rotation_period/2 + 1;
        /* works for even and odd rotation_period */
    else xPSLAcount[n] += rotation_period/2;
  }
  else {
    countO[n_points] += 2;
    PSLAcount[n] += 2 * rotation_period;
    xPSLAcount[n] += rotation_period;
  }
  int crossing_number ← count_crossings(n);
  assert(num_halving_lines ≤ MAX_HALVING_LINES);
  classcount[n_points][hullsize][rotation_period][¬is_symmetric ? NO_MIRROR : has_fixpoint ?
      MIRROR_WITH_FIXPOINT : MIRROR_WITHOUT_FIXPOINT][num_halving_lines][crossing_number]++;
}
#if 0   /* debugging */
printf("found␣n=%d.␣%Ld␣", n_points, countO[n_points]);
print_small(S, n_points);
#endif
```

This code is used in chunk 14.

¶ written to a file so that a subsequent program can conveniently read and process it.

53 ⟨Report statistics 53⟩ ≡

```
printf("%34s%69s\n", "#PSLA␣visited␣by␣the␣program", "#PSLA␣computed␣from␣AOT");
for_int_from_to (n, 3, n_max + 1) {
  long long symmetric ← 2 * countU[n] − countO[n];
  printf("n=%2d", n);
  if (split_level ≠ 0 ∧ n > split_level) printf("*,"); else printf(",␣");
  printf("#PSLA=%11Ld", countPSLA[n]);
#if 1
  printf(",␣#AOT=%10Ld,␣#OAOT=%10Ld,␣#symm.␣AOT=%7Ld,␣", countU[n], countO[n], symmetric);
  printf("#PSLA=%11Ld,␣#xPSLA=%10Ld", PSLAcount[n], xPSLAcount[n]);
#endif
  printf("\n");
}
if (split_level ≠ 0) printf("*␣Lines␣with␣\"*\"␣give␣results␣from␣partial␣enumeration.\n");
#if profile
printf("Total␣tests␣is_lex_min␣(after␣screening)␣=␣%Ld,␣total␣comparisons␣=␣%Ld,␣averag\
    e␣=%6.3f\n", numTests, numComparisons, numComparisons/(double) numTests);
#endif
printf("passed␣%Ld,␣saved␣%Ld␣out␣of␣%Ld␣=␣%.2f%%\n", cpass, csaved, cpass + csaved,
    100 * csaved/(double)(cpass + csaved));
if (strlen(fname)) {
  fprintf(reportfile, "#␣N_max=%d/%d", n_max, n_max + 1);
  if (parts ≠ 1) fprintf(reportfile, ",␣split-level=%d,␣part␣%d␣of␣%d", split_level, part, parts);
  fprintf(reportfile, "\n#x␣N␣hull␣period␣mirror-type␣halving-lines␣crossing-number␣NUM\n");
  for_int_from_to (n, 0, n_max + 1) {
    char c ← 'T';   /* total count */
    if (parts ≠ 1 ∧ n > split_level + 1) c ← 'P';   /* partial count */
    for_int_from_to (k, 0, n_max + 1)
      for_int_from_to (p, 0, n_max + 1)
        for_int_from_to (t, 0, 2)
```

```
          for_int_from_to (h, 0, MAX_HALVING_LINES)
            for_int_from_to (cr, 0, MAX_CROSSINGS)
              if (classcount[n][k][p][t][h][cr]) fprintf(reportfile, "%c␣%d␣%d␣%d␣%d␣%d␣%d␣␣%Ld\n", c, n, k,
                    p, t, h, cr, classcount[n][k][p][t][h][cr]);
    }
    if (parts ≡ 1) fprintf(reportfile, "EOF\n");
    else fprintf(reportfile, "EOF␣%d,␣part␣%d␣of␣%d\n", split_level, part, parts);
    fclose(reportfile);
    printf("Results␣have␣been␣written␣to␣file␣%s.\n", fname);
  }
```

This code is used in chunk 3.

¶ Input: PSLA with n lines $1\,..\,n$ plus line 0 "at ∞". Output: small λ-matrix B for AOT on $n+1$ points. Line at ∞ corresponds to point 0 on the convex hull.

61 ⟨Subroutines 24⟩ +≡

```
  void convert_to_small_lambda_matrix(small_matrix *B, int n)
  {
    for_int_from_to (i, 0, n) {
      (*B)[i][i] ← 0;
    }
    for_int_from_to (i, 1, n) {
      int level ← i − 1;    /* number of lines above the crossing */
      (*B)[0][i] ← level;
      (*B)[i][0] ← n − 1 − level;
      int j ← SUCC(i, 0);
      while (j ≠ 0) {
        if (i < j) {
          (*B)[i][j] ← level;
          level++;
        }
        else {
          level−−;
          (*B)[i][j] ← n − 1 − level;
        }
        j ← SUCC(i, j);
      }
    }
  }
```

3 Extension: Compute crossing-number for each AOT

By https://oeis.org/A076523, a set with $n = 12$ points (the maximum that the program is set up to deal with), has at most 18 halving-lines. According to S. Bereg and M. Haghpanah, New algorithms and bounds for halving pseudolines, Discrete Applied Mathematics 319 (2022) 194–206, https://doi.org/10.1016/j.dam.2021.05.029, Table 1 on p. 196, the number of halving lines-with for odd numbers n of points are nearly 70 % higher than for the adjacent even values. With a bound of 50 we should be on the safe side. A set with $n = 11$ points has at most 24 halving-lines.

62

```
#define MAX_HALVING_LINES 24
#define MAX_CROSSINGS (MAXN +1) *MAXN *(MAXN −1)*(MAXN −2) / 24
            /* crossing-number goes up to (n choose 4) for n points */
```

¶ How to check for a crossing.

This algorithm is like the program for drawing the wiring diagram, except that it does not draw anything.

The program computes the number of crossings *num_crossings_on_level*[p] at each level p including the crossings with line 0. A crossing at level p is a crossings between consecutive tracks p and $p+1$, $0 \le p \le n-1$.

From this information, there is an easy formula to compute the crossing number of the complete graph K_n when it is drawn on this point set, see Lovász, Vesztergombi, Wagner, and Welzl, *Convex quadrilaterals and k-sets*, DOI:10.1090/conm/342/06138.

63 **#define** CHECK_CROSSING(p)

```
{
  {
    int i ← line_at[p];
    int j ← line_at[p + 1];
    if (i < j ∧ next_crossing[i] > i ∧ next_crossing[j] < j ∧ next_crossing[j] ≠ 0)
        /* Line i wants to cross down and line j wants to cross up. */
        /* (In this case, we must actually have next_crossing[i] ≡ j and next_crossing[j] ≡ i.) */
      crossings[num_crossings++] ← p;
        /* The value p indicates a crossing between tracks p and p + 1. */
  }
}
```

⟨Subroutines 24⟩ +≡

```
int count_crossings(int n)
{
  int next_crossing[MAXN +1];
  int line_at[MAXN +1];
  int num_crossings_on_level[MAXN −1];
  int crossings[MAXN];    /* stack */
  int num_crossings ← 0;    /* Initialize */
  for_int_from_to (i, 1, n) {
    next_crossing[i] ← SUCC(i, 0);
      /* current crossing on each line; The first crossing with line 0 “at ∞” is not considered. */
    line_at[i − 1] ← i;    /* which line is on the p-th track, 0 ≤ p < n. tracks are numbered p = 0 . . . n − 1
        from top to bottom. */
  }
  for_int_from_to (p, 0, n − 1) num_crossings_on_level[p] ← 1;    /* counting the crossing with line 0 */
      /* maintain a stack crossings of available crossings. p ∈ crossings means that tracks p and p + 1 are
        ready to cross */
  for_int_from_to (p, 0, n − 2) CHECK_CROSSING(p)
  while (num_crossings) {    /* Main loop */
    int p ← crossings[−−num_crossings];
    num_crossings_on_level[p]++;
    xE    /* update the data structures to CARRY OUT the crossing */
        int i ← line_at[p];
        int j ← line_at[p + 1];
    next_crossing[i] ← SUCC(i, next_crossing[i]);
    next_crossing[j] ← SUCC(j, next_crossing[j]);
    line_at[p] ← j;
    line_at[p + 1] ← i;    /* Look for new crossings: */
    if (p > 0) CHECK_CROSSING(p − 1)
    if (p < n − 1) CHECK_CROSSING(p + 1)
  }    /* compute result */
  int crossing_formula ← −(n + 1) ∗ n ∗ (n − 1)/2;
  for_int_from_to (p, 0, n − 1)
    crossing_formula += num_crossings_on_level[p] ∗ (n − 1 − 2 ∗ p) ∗ (n − 1 − 2 ∗ p);
      /* global variable num_halving_lines is set. */
  if (n % 2)    /* n odd, number of points even: */
    num_halving_lines ← num_crossings_on_level[(n − 1)/2];
  else    /* n even, number of points odd: */
    num_halving_lines ← num_crossings_on_level[n/2] + num_crossings_on_level[n/2 − 1];
  return crossing_formula/4;
}
```