December 16, 2023 at 15:25
Changed sections for computing the crossing number.

1 The main program

Each PSLA for n lines has a unique parent with $n-1$ lines. This defines a tree structure on the PSLAs. The principle of the enumeration algorithm is a depth-first traversal of this tree.

Change: We are keeping statistics for several independent characteristics, one of which (the *crossing_number*) can rise to high values (see `MAX_CROSSINGS` in Section 3). Therefore, we reduce the maximum number `MAXN` of pseudolines from 15 to what we really need.

6   **#define** `MAXN` 12   /* The maximum number of pseudolines for which the program will work. */

⟨ Include standard libaries 10 ⟩
⟨ Types and data structures 4 ⟩
⟨ Global variables 5 ⟩
⟨ Subroutines 27 ⟩
⟨ Core subroutines for recursive generation 15 ⟩

int *main*(**int** *argc*, **char** *∗argv*[ ])
{
   ⟨ Parse the command line 13 ⟩;
#if *readdatabase*   /* reading from the database */
   ⟨ Read all point sets of size *n_max* + 1 from the database and process them 77 ⟩
   return 0;
#endif
#if *enumAOT*
   ⟨ Initialize statistics and open reporting file 52 ⟩;
   ⟨ Start the generation 16 ⟩;
   ⟨ Report statistics 54 ⟩;
#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 fixed vertex on the hull (3 possibilities).

In addition, we keep

- the number of halving-lines, *num_halving_lines*.

- the crossing number, *crossing_number*.

PSLAcount counts OAOT of point sets with a marked point on the convex hull, but no specified traversal direction. http://oeis.org/A006245 (see below) is the same sequence with n shifted by 0. *xPSLAcount* counts OAOT of point sets with a marked point on the ... ?

51   **#define** `NO_MIRROR` 0
   #define `MIRROR_WITH_FIXED_VERTEX` 1
   #define `MIRROR_WITHOUT_FIXED_VERTEX` 2

⟨ Global variables 5 ⟩ +≡
   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 $n = 16$. */
   long long unsigned *xPSLAcount*[`MAXN` + 2];

long long unsigned $classcount[\texttt{MAXN}+2][\texttt{MAXN}+2][\texttt{MAXN}+2][3][\texttt{MAX\_HALVING\_LINES}+1][\texttt{MAX\_CROSSINGS}+1]$;
  /∗ This is a huge array: If it were full it would take about 2 GBytes of main memory. But it is very
  sparsely filled. With the `gcc` compiler, the system reports less than 30 MBytes of used storage, and
  often only 2 or 3 MBytes. This might just depend on the available memory for swapping. With `clang`
  compiler, the observed memory usage was also less than 3 MBytes. Only in ⟨ Report statistics 54 ⟩ is the
  full array scanned once. ∗/
int $num\_halving\_lines$;   /∗ global variable; (This is not the cleanest way to do it.) ∗/
long long unsigned $numComparisons \leftarrow 0$, $numTests \leftarrow 0$;   /∗ profiling ∗/

53 ¶ ⟨ Gather statistics about the AOT, collect output 53 ⟩ ≡
  int $hulledges[\texttt{MAXN}+1]$;
  int $hullsize \leftarrow upper\_hull\_PSLA(n, hulledges)$;   /∗ Determine the extreme points: ∗/

  int $rotation\_period$;
  boolean has_fixed_vertex;
  boolean $has\_mirror\_symmetry$;
  int $n\_points \leftarrow n+1$;   /∗ number of points of the AOT ∗/
  boolean $lex\_smallest \leftarrow is\_lex\_smallest\_P\_matrix(n, hulledges, hullsize, \&rotation\_period,$
    $\&has\_mirror\_symmetry, \&has\_fixed\_vertex)$;

  if ($lex\_smallest$) {
    $countU[n\_points]{+}{+}$;   /∗ We count to contribution from this AOT to the various counters $countO$,
        $PSLAcount$, $xPSLAcount$ according to the symmetry information. ∗/
    if ($has\_mirror\_symmetry$) {
      $countO[n\_points]{+}{+}$;
      $PSLAcount[n]\mathrel{+}=rotation\_period$;
      if (has_fixed_vertex) $xPSLAcount[n]\mathrel{+}=rotation\_period/2+1$;
          /∗ works for even and odd $rotation\_period$ ∗/
      else $xPSLAcount[n]\mathrel{+}=rotation\_period/2$;
    }
    else {
      $countO[n\_points]\mathrel{+}=2$;
      $PSLAcount[n]\mathrel{+}=2*rotation\_period$;
      $xPSLAcount[n]\mathrel{+}=rotation\_period$;
    }

    int $crossing\_number \leftarrow count\_crossings(n)$;

    $assert(num\_halving\_lines \leq \texttt{MAX\_HALVING\_LINES})$;
    $classcount[n\_points][hullsize][rotation\_period][\neg has\_mirror\_symmetry$?
        $\texttt{NO\_MIRROR} : has\_fixed\_vertex$? $\texttt{MIRROR\_WITH\_FIXED\_VERTEX} :$
        $\texttt{MIRROR\_WITHOUT\_FIXED\_VERTEX}][num\_halving\_lines][crossing\_number]{+}{+}$;
  }
  #**if** 0   /∗ debugging ∗/
    $printf(\texttt{"found\_n=\%d.\_\%Ld\_"}, n\_points, countO[n\_points])$;
    $print\_small(S, n\_points)$;
  #**endif**
  This code is used in chunk 18.

¶ The statistics gathered in the *classcount* array are written to a *reportfile* so that a subsequent program can
conveniently read and process it.

55 ⟨ Report statistics 54 ⟩ +≡
  if ($strlen(fname)$) {
    $fprintf(reportfile, \texttt{"\#\_N\_max=\%d/\%d"}, n\_max, n\_max+1)$;
    if ($parts \neq 1$) $fprintf(reportfile, \texttt{",\_split-level=\%d,\_part\_\%d\_of\_\%d"}, split\_level, part, parts)$;
    $fprintf(reportfile, \texttt{"\textbackslash nx\_N\_hull\_period\_mirror-type\_halving-lines\_crossing-number\_NUM\textbackslash n"})$;
    for_int_from_to ($n, 0, n\_max+1$) {
      char $c \leftarrow \texttt{'T'}$;   /∗ total count ∗/

      if ($parts \neq 1 \wedge n > split\_level+1$) $c \leftarrow \texttt{'P'}$;   /∗ partial count ∗/
      for_int_from_to ($k, 0, n$)
        for_int_from_to ($p, 0, n$)

```
          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␣␣%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);
  }
```

3 Extension: Compute crossing-number for each AOT

What range of values should we anticipate for the number of halving-lines? By https://oeis.org/A076523, a set with $n = 12$ points 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. I could not find the bounds for small odd n in the literature or on the internet. After running the program once with a larger safety margin, it was found that a set with $n = 11$ points has at most 24 halving-lines. (The program checks if the bound is not violated.) With $n = 13$ points and a bound 40 on the halving-lines we should be on the safe side.

58 **#define** MAX_HALVING_LINES 40   /∗ For $n = 12$ point, MAX_HALVING_LINES ← 24 would be sufficient. ∗/
    #define MAX_CROSSINGS (MAXN + 1) ∗ MAXN ∗ (MAXN − 1) ∗ (MAXN − 2)/24
         /∗ crossing-number goes up to $\binom{n}{4}$ for n points ∗/

¶ Find the number of crossings on each level of the wiring diagram.
   This algorithm is like the program for drawing the wiring diagram, except that it does not draw anything. The wires run on n *tracks*, which are labeled from 1 to n from top to bottom.

   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 crossing between consecutive tracks p and $p + 1$, $1 \le p \le n - 1$.

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

   In addition, the program stores the number of halving-lines in the global variable $num\_halving\_lines$.

59 ⟨ Subroutines 27 ⟩ +≡
    int *count_crossings* (**int** n)
    {
      int $line\_at$[MAXN + 1];
      int $next\_crossing$[MAXN + 1];
      int $num\_crossings\_on\_level$[MAXN + 1];
      for_int_from_to $(i, 0, n)$ {   /∗ Initialize: ∗/
        $line\_at[i] \leftarrow i$;   /∗ which line is on the i-th track, $1 \le i \le n$. Tracks are numbered $p = 1 \dots n$ from
            top to bottom. ∗/
        $next\_crossing[i] \leftarrow$ SUCC$(i, 0)$;
            /∗ current crossing on each line; the first crossing with line 0 "at ∞" is counted separately: ∗/
        $num\_crossings\_on\_level[i] \leftarrow 1$;   /∗ The initial value counts the crossing with line 0 ∗/
      }
      $next\_crossing[0] \leftarrow 1$;   /∗ sentinel values to ensure that the loop breaks at $p = 0$ ∗/

      int $p \leftarrow n$;

      while (*true*) {   /∗ Main loop ∗/
          /∗ Invariant throughout: The lines at levels $p + 1, \dots, n$ want to cross *upwards* (or are finished.) ∗/
        while ($next\_crossing[p] < line\_at[p]$) p−−;   /∗ **while** p wants to cross upwards ∗/
        if ($p \equiv 0$) **break**;
        do {   /∗ The line $i \equiv line\_at[p]$ on track p wants to cross down and the line $j \equiv line\_at[p + 1]$ on
            track $p + 1$ wants to cross up. (In this case, we must actually have $next\_crossing[i] \equiv j$ and
            $next\_crossing[j] \equiv i$.) ∗/
          $num\_crossings\_on\_level[p]$++;
```

```
/* update the data structures to CARRY OUT the crossing p ↔ p + 1: */
    int i ← line_at[p];
    int j ← line_at[p + 1];
    int temp ← SUCC(i, next_crossing[p]);

    next_crossing[p] ← SUCC(j, next_crossing[p + 1]);
    next_crossing[p + 1] ← temp;
    line_at[p] ← j;
    line_at[p + 1] ← i;

    p++;
  } while (next_crossing[p] > line_at[p]);    /* while p wants to cross downwards */
  p −= 2;    /* We need not check p − 1, because this would be an immediate back-crossing at the same
      level. */
}
```

/* Number of halving-lines is stored in the global variable *num_halving_lines*: */
if $(n \% 2)$ /* n odd, number of points even: */
  $num\_halving\_lines \leftarrow num\_crossings\_on\_level[(n + 1)/2]$;
else   /* n even, number of points odd: */
  $num\_halving\_lines \leftarrow num\_crossings\_on\_level[n/2] + num\_crossings\_on\_level[n/2 + 1]$;

/* compute crossing number according to the formula: */

int $crossing\_formula \leftarrow -(n + 1) * n * (n - 1)/2$;

for_int_from_to $(p, 1, n)$ $crossing\_formula$ += $num\_crossings\_on\_level[p] * (n + 1 - 2 * p) * (n + 1 - 2 * p)$;
return $crossing\_formula/4$;
}