November 24, 2020 at 13:25

1.   Intro.   This program computes the up-up-or-down-down permutation of $\{1, 2, \ldots, 2n - 1\}$ that corresponds to a given $2 \times n$ whirlpool permutation. The latter permutation appears on the command line, as a permutation of $\{0, 1, \ldots, 2n - 1\}$, with 0 in the bottom left corner.

   (I've made no attempt to be efficient.)

   (But I didn't go out of my way to be inefficient.)

   (Apologies for doing this hurriedly.)

#define   *maxn*   100

#include <stdio.h>
#include <stdlib.h>
   int $a[2 * maxn]$;
   int $used[2 * maxn]$;
   int $answer[2 * maxn]$;

   $main(\textbf{int}\ argc, \textbf{char}\ *argv[\,])$
   {
      register int i, j, k, n, nn, t, $saven$;

      ⟨Process the command line 2⟩;
      for (; $n > 1$; $n{-}{-}$) ⟨Reduce the problem from n to $n - 1$ 4⟩;
      ⟨Print the answer 5⟩;
   }

2.   ⟨ Process the command line 2 ⟩ ≡
   if $(argc < 5 \vee ((argc \mathbin{\&} 1) \equiv 0))$ {
      $fprintf(stderr, \texttt{"Usage:␣\%s␣a11␣a12␣...␣a1n␣0␣a22␣...␣a2n\textbackslash n"}, argv[0]);$
      $exit(-1);$
   }
   $nn = argc - 1, n = saven = nn/2;$
   if $(n > maxn)$ {
      $fprintf(stderr, \texttt{"Recompile␣me:␣This␣program␣has␣maxn=\%d!\textbackslash n"}, maxn);$
      $exit(-99);$
   }
   for $(k = 0;\ k < nn;\ k\texttt{++})$ {
      if $(sscanf(argv[k+1], \texttt{"\%d"}, \&a[k]) \neq 1)$ {
         $fprintf(stderr, \texttt{"Bad␣matrix␣entry␣`\%s'!\textbackslash n"}, argv[k+1]);$
         $exit(-2);$
      }
      if $(a[k] < 0 \vee a[k] \geq nn)$ {
         $fprintf(stderr, \texttt{"Matrix␣entry␣`\%d'␣out␣of␣range!\textbackslash n"}, a[k]);$
         $exit(-3);$
      }
      if $(used[a[k]])$ {
         $fprintf(stderr, \texttt{"Duplicate␣matrix␣entry␣`\%d'!\textbackslash n"}, a[k]);$
         $exit(-4);$
      }
      $used[a[k]] = 1;$
   }
   if $(a[n])$ {
      $fprintf(stderr, \texttt{"Matrix␣entry␣a21␣should␣be␣zero,␣not␣\%d!\textbackslash n"}, a[n]);$
      $exit(-5);$
   }
   ⟨ Verify the whirlpool criteria 3 ⟩;
This code is used in section 1.

3.   ⟨ Verify the whirlpool criteria 3 ⟩ ≡
   for $(k = n + 1;\ k < nn;\ k\texttt{++})$ {
      if $((((a[k-n-1] < a[k-n]) + (a[k-n] < a[k]) + (a[k] < a[k-1]) + (a[k-1] < a[k-n-1])) \mathbin{\&} 1) \equiv 0)$
         {
         $fprintf(stderr, \texttt{"Not␣a␣vortex!␣(\%d␣\%d␣/␣\%d␣\%d)\textbackslash n"}, a[k-n-1], a[k-n], a[k-1], a[k]);$
         $exit(-6);$
      }
   }
This code is used in section 2.

4. $\langle$ Reduce the problem from n to $n-1$ $_4\rangle \equiv$
```
  {
    register int t, nnp;
    nnp = n + n - 2;
    answer[nnp + 1] = a[0], answer[nnp] = a[1];
    for (k = 1; k < n; k++) {
      t = a[k];
      if (t > answer[nnp + 1]) t−−;
      a[k − 1] = t − 1;
      t = a[k + saven];
      if (t > answer[nnp + 1]) t−−;
      a[k + saven − 1] = t − 1;
    }
    for (t = nnp − a[saven], k = 0; k < n − 1; k++) {
      a[k] = (a[k] + t) % nnp;
      a[k + saven] = (a[k + saven] + t) % nnp;
    }
  }
```
This code is used in section 1.

5.   At this point $n = 1$, and *answer* contains numbers that need to be "uncompressed" because they were the results of a recursive computation on a compressed problem.

$\langle$ Print the answer $_5\rangle \equiv$
```
  n = saven;
  answer[1] = 1;
  for (k = 0; k < nn; k++) used[k] = 0;
  used[answer[nn − 1]] = used[answer[nn − 2]] = 1;
  for (k = nn − 4; k ≥ 0; k −= 2) {
    t = answer[k + 1];
    for (j = 1; j ≤ t; j++)
      if (used[j]) t++;
    answer[k + 1] = t;
    t = answer[k];
    for (j = 1; j ≤ t; j++)
      if (used[j]) t++;
    answer[k] = t;
    used[t] = used[answer[k + 1]] = 1;
  }
  for (k = nn − 1; k; k−−) printf ("␣%d", answer[k]);
  printf ("\n");
```
This code is used in section 1.

6.   Index.

⟨ Print the answer 5 ⟩   Used in section 1.
⟨ Process the command line 2 ⟩   Used in section 1.
⟨ Reduce the problem from n to $n-1$ 4 ⟩   Used in section 1.
⟨ Verify the whirlpool criteria 3 ⟩   Used in section 2.

WHIRLPOOL2N-ENCODE