

November 24, 2020 at 13:25

1. Intro. This program computes the $2 \times n$ whirlpool permutation that corresponds to a given up-up-or-down-down permutation of $\{1, 2, \dots, 2n - 1\}$, which appears on the command line. So it's essentially the inverse of the program WHIRLPOOL2N-ENCODE, and its output is a suitable input to that program.

By “up-up-or-down-down permutation” of length $2n - 1$, I mean a permutation $p_1 \dots p_{2n-1}$ such that $p_{2k-1} < p_{2k}$ if and only if $p_{2k} < p_{2k+1}$, for $1 \leq k < n$.

(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];    /* where we build the answer */
int g[2 * maxn];    /* the given permutation */
int w[2 * maxn];    /* workspace */
int used[2 * maxn];

main(int argc, char *argv[])
{
    register int i, j, k, n, nn, t, x, y, saven;
    ⟨ Process the command line 2 ⟩;
    ⟨ Prepare to grow 4 ⟩;
    for (n = 1; n < saven; n++) ⟨ Grow the solution from n to n + 1 5 ⟩;
    ⟨ Print the answer 6 ⟩;
}
```

2. \langle Process the command line 2 $\rangle \equiv$

```

if (argc & 1) {
    fprintf(stderr, "Usage: %s a1 a2 . . . a(2n-1)\n", argv[0]);
    exit(-1);
}
nn = argc, n = saven = nn/2;
if (n > maxn) {
    fprintf(stderr, "Recompile me: This program has maxn=%d!\n", maxn);
    exit(-99);
}
for (k = 1; k < nn; k++) {
    if (sscanf(argv[k], "%d", &g[k])  $\neq$  1) {
        fprintf(stderr, "Bad perm element '%s'!\n", argv[k]);
        exit(-2);
    }
    if (g[k]  $\leq$  0  $\vee$  g[k]  $\geq$  nn) {
        fprintf(stderr, "Perm element '%d' out of range!\n", g[k]);
        exit(-3);
    }
    if (used[g[k]]) {
        fprintf(stderr, "Duplicate perm entry '%d'!\n", g[k]);
        exit(-4);
    }
    used[g[k]] = 1;
}
 $\langle$  Verify the up-up-or-down-down criteria 3  $\rangle$ ;

```

This code is used in section 1.

3. \langle Verify the up-up-or-down-down criteria 3 $\rangle \equiv$

```

for (k = 2; k < nn; k += 2) {
    if ((g[k - 1] < g[k])  $\neq$  (g[k] < g[k + 1])) {
        fprintf(stderr, "Not up-up-or-down-down! (%d %d %d)\n", g[k - 1], g[k], g[k + 1]);
        exit(-6);
    }
}

```

This code is used in section 2.

4. Here I compress the “uncompressed” numbers in the given permutation.

⟨ Prepare to grow 4 ⟩ \equiv

```

a[0] = 1;
for (k = 1; k < nn; k++) used[k] = 0;
for (k = 2; k < nn; k += 2) {
    x = g[k - 1], y = g[k];
    for (t = 0, j = 1; j < x; j++)
        if (used[j]) t++;
    g[k - 1] -= t;
    for (t = 0, j = 1; j < y; j++)
        if (used[j]) t++;
    g[k] -= t;
    used[x] = used[y] = 1;
}
g[nn - 1] = 1;

```

This code is used in section 1.

5. ⟨ Grow the solution from n to $n + 1$ 5 ⟩ \equiv

```

{
    x = g[nn - n - n - 1], y = g[nn - n - n];
    t = y - (x < y ? 2 : 1) - a[0] + n + n;
    for (k = n - 1; k ≥ 0; k--)
        a[k + 1] = (a[k] + t) % (n + n), a[k + saven + 1] = (a[k + saven] + t) % (n + n);
    for (k = 1; k ≤ n; k++) a[k] += (a[k] < x - 1 ? 1 : 2), a[k + saven] += (a[k + saven] < x - 1 ? 1 : 2);
    a[0] = x;
}

```

This code is used in section 1.

6. ⟨ Print the answer 6 ⟩ \equiv

```

for (k = 0; k < nn; k++) printf("␣%d", a[k]);
printf("\n");

```

This code is used in section 1.

7. Index.

a: [1](#).
argc: [1](#), [2](#).
argv: [1](#), [2](#).
exit: [2](#), [3](#).
fprintf: [2](#), [3](#).
g: [1](#).
i: [1](#).
j: [1](#).
k: [1](#).
main: [1](#).
maxn: [1](#), [2](#).
n: [1](#).
nn: [1](#), [2](#), [3](#), [4](#), [5](#), [6](#).
printf: [6](#).
saven: [1](#), [2](#), [5](#).
sscanf: [2](#).
stderr: [2](#), [3](#).
t: [1](#).
used: [1](#), [2](#), [4](#).
w: [1](#).
x: [1](#).
y: [1](#).

- ⟨ Grow the solution from n to $n + 1$ 5 ⟩ Used in section 1.
- ⟨ Prepare to grow 4 ⟩ Used in section 1.
- ⟨ Print the answer 6 ⟩ Used in section 1.
- ⟨ Process the command line 2 ⟩ Used in section 1.
- ⟨ Verify the up-up-or-down-down criteria 3 ⟩ Used in section 2.

WHIRLPOOL2N-DECODE

	Section	Page
Intro	1	1
Index	7	4