

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

1. Intro. A simple program to find the vacillating tableau loop that corresponds to a given restricted growth string, given in the standard input file.

The program also computes the dual restricted growth string.

Apology: I wrote the following code in an awful hurry, so there was no time to apply spit and/or polish.

```
#define maxn 1000
#include <stdio.h>
char buf[maxn]; /* the restricted growth string input */
int last[maxn]; /* table for decoding the restricted growth string */
int a[maxn], b[maxn]; /* rook positions */
int p[maxn][maxn], q[maxn][maxn]; /* tableaux */
int r[maxn]; /* row lengths */
int dualins[maxn], dualdel[maxn]; /* row changes for the dual */
int verbose = 1;

main(int argc, char *argv[])
{
    register int i, j, k, m, n, xi, xip;
    while (fgets(buf, maxn, stdin)) {
        (Build the rook table 2);
        (Make the inverse rook table 3);
        (Compute and print the intermediate tableaux 4);
        (Compute the dual rook table 9);
        (Print the restricted growth string of the dual 12);
    }
}
```

2. (Build the rook table 2) \equiv

```
printf("Given: %s", buf);
for (k = 0, m = -1; (buf[k] ≥ '0' ∧ buf[k] ≤ '9') ∨ (buf[k] ≥ 'a' ∧ buf[k] ≤ 'z'); k++) {
    j = (buf[k] ≥ 'a' ? buf[k] - 'a' + 10 : buf[k] - '0');
    if (j > m) {
        if (j ≠ m + 1) {
            buf[k] = 0;
            fprintf(stderr, "Bad form: %s%d should be %s%d!\n", buf, j, buf, m + 1);
            continue;
        }
        m = j, last[m] = 0;
    }
    a[k + 1] = last[j], last[j] = k + 1;
}
n = k;
```

This code is used in section 1.

3. (Make the inverse rook table 3) \equiv

```
for (k = 1; k ≤ n; k++) b[k] = 0;
for (k = 1; k ≤ n; k++)
    if (a[k]) b[a[k]] = k;
```

This code is used in section 1.

```

4. #define infty 1000 /* infinity (approximately) */
⟨ Compute and print the intermediate tableaux 4 ⟩ ≡
  ⟨ Initialize the tableaux 5 ⟩;
  for ( $k = 1$ ;  $k \leq n$ ;  $k++$ ) {
    ⟨ Possibly delete  $k$  7 ⟩;
    ⟨ Possibly insert  $k$  6 ⟩;
  }

```

This code is used in section 1.

```

5. ⟨ Initialize the tableaux 5 ⟩ ≡
  for ( $k = 1$ ;  $k \leq n$ ;  $k++$ ) {
     $r[k] = q[0][k] = q[k][0] = 0, p[0][k] = p[k][0] = \text{inf}ty$ ;
    for ( $j = 1$ ;  $j \leq n$ ;  $j++$ )  $q[k][j] = \text{inf}ty, p[k][j] = 0$ ;
  }

```

This code is used in section 4.

6. Here's Algorithm 5.1.4I, but with order reversed in the p tableau. We insert $b[k]$ into p and k into q .

I wouldn't actually have to work with both p and q ; either one would suffice to determine the vacillation. But I compute them both because I'm trying to get familiar with the whole picture.

```

⟨ Possibly insert  $k$  6 ⟩ ≡
  if ( $b[k]$ ) {
     $i1$ :  $i = 1, xi = b[k], j = r[1] + 1$ ;
    while (1) {
       $i2$ : while ( $xi > p[i][j - 1]$ )  $j--$ ;
       $xip = p[i][j]$ ;
       $i3$ :  $p[i][j] = xi$ ;
       $i4$ : if ( $xip$ )  $i++, xi = xip$ ;
      else break;
    }
     $q[i][j] = k$ ;
     $r[i] = j$ ;
     $dualins[k] = j$ ;
  } else  $dualins[k] = 0$ ;
  ⟨ Print the tableau shape 8 ⟩;

```

This code is used in section 4.

7. And here's Algorithm 5.1.4D, applied to the q tableau. We delete k from p and $a[k]$ from q . The error messages here won't be needed unless I have made a mistake.

⟨Possibly delete k 7⟩ \equiv

```

if ( $a[k]$ ) {
  for ( $i = 1, j = 0; r[i]; i++$ )
    if ( $p[i][r[i]] \equiv k$ ) {
       $j = r[i], r[i] = j - 1, p[i][j] = 0;$ 
       $dualdel[k] = j;$ 
      break;
    }
  if ( $\neg j$ ) {
     $fprintf(stderr, "I\_couldn't\_find\_d\_in\_p!\n", k);$ 
     $exit(-1);$ 
  }
 $d1: xip = infty;$ 
  while (1) {
     $d2: \text{while } (q[i][j + 1] < xip) \ j++;$ 
     $xi = q[i][j];$ 
     $d3: q[i][j] = xip;$ 
     $d4: \text{if } (i > 1) \ i--, xip = xi;$ 
    else break;
  }
  if ( $xi \neq a[k]$ ) {
     $fprintf(stderr, "I\_removed\_d,\_not\_d,\_from\_q!\n", xi, a[k]);$ 
     $exit(-2);$ 
  }
} else  $dualdel[k] = 0;$ 
⟨Print the tableau shape 8⟩;

```

This code is used in section 4.

8. If *verbose* is nonzero, we also print out the contents of p and q .

⟨Print the tableau shape 8⟩ \equiv

```

for ( $i = 1; r[i]; i++$ )  $printf("%d", r[i]);$ 
if ( $verbose \wedge i > 1$ ) {
   $printf("\_");$ 
  for ( $i = 1; r[i]; i++$ ) {
    if ( $i > 1$ )  $printf(";");$ 
    for ( $j = 1; j \leq r[i]; j++$ )  $printf("%s%d", j > 1 ? ", " : "", p[i][j]);$ 
  }
   $printf("), (");$ 
  for ( $i = 1; r[i]; i++$ ) {
    if ( $i > 1$ )  $printf(";");$ 
    for ( $j = 1; j \leq r[i]; j++$ )  $printf("%s%d", j > 1 ? ", " : "", q[i][j]);$ 
  }
   $printf(")");$ 
}
if ( $i \equiv 1$ )  $printf("\_0\n");$  else  $printf("\n");$ 

```

This code is used in sections 6 and 7.

9. Now for the dual, I'll work only with q .

```

⟨ Compute the dual rook table 9 ⟩ ≡
  for ( $k = 1$ ;  $k \leq n$ ;  $k++$ ) {
    if ( $dualdel[k]$ ) ⟨ Dually delete  $k$  11 ⟩;
    if ( $dualins[k]$ ) ⟨ Dually insert  $k$  10 ⟩;
  }

```

This code is used in section 1.

10. ⟨ Dually insert k 10 ⟩ ≡
 $i = dualins[k], j = r[i] + 1, r[i] = j, q[i][j] = k$;

This code is used in section 9.

11. ⟨ Dually delete k 11 ⟩ ≡

```

{
   $i = dualdel[k], j = r[i], r[i] = j - 1, xip = infty$ ;
  while (1) {
    while ( $q[i][j + 1] < xip$ )  $j++$ ;
     $xi = q[i][j]$ ;
     $q[i][j] = xip$ ;
    if ( $i > 1$ )  $i--$ ,  $xip = xi$ ;
    else break;
  }
   $a[k] = xi$ ;
}

```

This code is used in section 9.

12. ⟨ Print the restricted growth string of the dual 12 ⟩ ≡

```

for ( $k = 1, m = -1$ ;  $k \leq n$ ;  $k++$ )
  if ( $a[k]$ )  $buf[k - 1] = buf[a[k] - 1]$ ;
  else  $m++, buf[k - 1] = (m > 9 ? 'a' + m - 10 : '0' + m)$ ;
printf("Dual: %s", buf);

```

This code is used in section 1.

13. Index.

a: [1](#).
argc: [1](#).
argv: [1](#).
b: [1](#).
buf: [1](#), [2](#), [12](#).
dualdel: [1](#), [7](#), [9](#), [11](#).
dualins: [1](#), [6](#), [9](#), [10](#).
d1: [7](#).
d2: [7](#).
d3: [7](#).
d4: [7](#).
exit: [7](#).
fgets: [1](#).
fprintf: [2](#), [7](#).
i: [1](#).
infty: [4](#), [5](#), [7](#), [11](#).
i1: [6](#).
i2: [6](#).
i3: [6](#).
i4: [6](#).
j: [1](#).
k: [1](#).
last: [1](#), [2](#).
m: [1](#).
main: [1](#).
maxn: [1](#).
n: [1](#).
p: [1](#).
printf: [2](#), [8](#), [12](#).
q: [1](#).
r: [1](#).
stderr: [2](#), [7](#).
stdin: [1](#).
verbose: [1](#), [8](#).
xi: [1](#), [6](#), [7](#), [11](#).
xip: [1](#), [6](#), [7](#), [11](#).

- ⟨ Build the rook table 2 ⟩ Used in section 1.
- ⟨ Compute and print the intermediate tableaux 4 ⟩ Used in section 1.
- ⟨ Compute the dual rook table 9 ⟩ Used in section 1.
- ⟨ Dually delete k 11 ⟩ Used in section 9.
- ⟨ Dually insert k 10 ⟩ Used in section 9.
- ⟨ Initialize the tableaux 5 ⟩ Used in section 4.
- ⟨ Make the inverse rook table 3 ⟩ Used in section 1.
- ⟨ Possibly delete k 7 ⟩ Used in section 4.
- ⟨ Possibly insert k 6 ⟩ Used in section 4.
- ⟨ Print the restricted growth string of the dual 12 ⟩ Used in section 1.
- ⟨ Print the tableau shape 8 ⟩ Used in sections 6 and 7.

VACILLATE

	Section	Page
Intro	1	1
Index	13	5