

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

1. Intro. A quickie to find a longest string that avoids the interesting set of “unavoidable” m -ary strings of length n constructed by Champarnaud, Hansel, and Perrin.

Their construction can be viewed as finding the minimum number of arcs to remove from the de Bruijn graph of $(n - 1)$ -tuples so that the resulting graph has no oriented cycles. (Because each n -letter string corresponds to an arc that must be avoided.)

This program constructs the graph and finds a longest path.

```
#define m 2      /* this many letters in the alphabet */
#define n 20     /* this many letters in each string */
#define space (1 << (n - 1)) /*  $m^{n-1}$  */
#include <stdio.h>
char avoid[m * space]; /* nonzero if the arc is removed */
int deg[space]; /* outdegree, also used as pointer to next level */
int link[space]; /* stack of vertices whose degree has dropped to zero */
int a[n + 1]; /* staging area */
int count; /* the number of vertices on the current level */
int code; /* an  $n$ -tuple represented in  $m$ -ary notation */
main()
{
    register int d, j, k, l, q;
    register int top; /* top of the linked stack */
    < Compute the avoid and deg tables 2 >;
    for (d = 0; count; d++) {
        printf("Vertices at distance %d: %d\n", d, count);
        for (l = top, top = -1, count = 0; l ≥ 0; l = link[l])
            < Decrease the degree of  $l$ 's predecessors, and stack them if their degree drops to zero 5 >
        }
        < Print out a longest path 6 >;
    }
}
```

2. Algorithm 7.2.1.1F gives us the relevant prime powers here.

```
< Compute the avoid and deg tables 2 > ≡
for (j = 0; j < space; j++) deg[j] = m;
count = d = 0;
top = -1;
for (j = n; j; j--) a[j] = 0;
a[0] = -1, j = 1;
while (1) {
    if (n % j ≡ 0) < Generate an  $n$ -tuple to avoid 3 >;
    for (j = n; a[j] ≡ m - 1; j--) ;
    if (j ≡ 0) break;
    a[j]++;
    for (k = j + 1; k ≤ n; k++) a[k] = a[k - j];
}
printf("m=%d, n=%d: avoiding one arc in each of %d disjoint cycles\n", m, n, d);
```

This code is used in section 1.

3. At this point $\lambda = a_1 \dots a_j$ is a prime string and $\alpha = a_1 \dots a_n = \lambda^{n/j}$. The crux of the Champarnaud/Hansel/Perrin method is to find the shortest prime μ such that α has the form $\mu^{\lfloor n/|\mu| \rfloor} \beta$, and to avoid the string $\beta \mu^{\lfloor n/|\mu| \rfloor}$.

We have $\mu = \lambda$ and $\beta = \epsilon$ if $j < n$. Otherwise we can use Duval's algorithm to discover all the prime prefixes of α , stopping when one of them has the desired form. (The resulting algorithm is quite pretty, if I do say so myself.)

```

⟨ Generate an  $n$ -tuple to avoid 3 ⟩ ≡
{
  d++;
  if ( $j < n$ )  $l = j, q = n$ ;
  else
    for ( $l = 1, k = 2$ ; ;  $k++$ ) {
      /* at this point  $a_1 \dots a_l$  is prime, and  $a_1 \dots a_{k-1}$  is its  $(k-1)$ -extension */
      if ( $a[k-l] < a[k]$ ) {
         $q = l * (\text{int})(n/l)$ ;
        if ( $k > q$ ) break;
         $l = k$ ;
        if ( $k \equiv n$ ) break;
      }
    }
  for ( $code = 0, k = q + 1$ ;  $k \leq n$ ;  $k++$ )  $code = m * code + a[k]$ ;
  for ( $k = 1$ ;  $k \leq q$ ;  $k++$ )  $code = m * code + a[k]$ ;
  ⟨ Avoid the  $n$ -tuple encoded by code 4 ⟩;
}

```

This code is used in section 2.

```

4. ⟨ Avoid the  $n$ -tuple encoded by code 4 ⟩ ≡
  avoid[code] = 1;
   $q = code / m$ ;
  deg[q]--;
  if (deg[q] ≡ 0) deg[q] = -1, link[q] = top, top = q, count++;

```

This code is used in section 3.

```

5. ⟨ Decrease the degree of  $l$ 's predecessors, and stack them if their degree drops to zero 5 ⟩ ≡
  for ( $j = m - 1$ ;  $j \geq 0$ ;  $j--$ ) {
     $k = l + j * space$ ;
    if ( $\neg avoid[k]$ ) {
       $q = k / m$ ;
      deg[q]--;
      if (deg[q] ≡ 0) deg[q] = l, link[q] = top, top = q, count++;
    }
  }
}

```

This code is used in section 1.

6. Here I apologize for using a dirty trick: The current value of k happens to be the most recent value of l , a vertex with no predecessors.

⟨ Print out a longest path [6](#) ⟩ \equiv

```
printf("Path:");
for (code = k, j = 1; j < n; j++) {
    code = code * m, q = code / space;
    printf("□%d", q);
    code -= q * space;
}
while (deg[k] ≥ 0) {
    printf("□%d", deg[k] % m);
    k = deg[k];
}
```

This code is used in section [1](#).

7. Index.

a: [1](#).

avoid: [1](#), [4](#), [5](#).

code: [1](#), [3](#), [4](#), [6](#).

count: [1](#), [2](#), [4](#), [5](#).

d: [1](#).

deg: [1](#), [2](#), [4](#), [5](#), [6](#).

j: [1](#).

k: [1](#).

l: [1](#).

link: [1](#), [4](#), [5](#).

m: [1](#).

main: [1](#).

n: [1](#).

printf: [1](#), [2](#), [6](#).

q: [1](#).

space: [1](#), [2](#), [5](#), [6](#).

top: [1](#), [2](#), [4](#), [5](#).

- ⟨ Avoid the n -tuple encoded by *code 4* ⟩ Used in section 3.
- ⟨ Compute the *avoid* and *deg* tables 2 ⟩ Used in section 1.
- ⟨ Decrease the degree of l 's predecessors, and stack them if their degree drops to zero 5 ⟩ Used in section 1.
- ⟨ Generate an n -tuple to avoid 3 ⟩ Used in section 2.
- ⟨ Print out a longest path 6 ⟩ Used in section 1.

UNAVOIDABLE

	Section	Page
Intro	1	1
Index	7	4