

(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 Mykkeltveit in 1972.

His 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.

I hacked it from the previous program UNAVOIDABLE, which uses a different set of strings.

```
#define m 2      /* this many letters in the alphabet */
#define n 20     /* this many letters in each string, assumed greater than 2 */
#define space (1 << (n - 1)) /*  $m^{n-1}$  */
#include <stdio.h>
#include <math.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 */
double sine[n]; /* imaginary parts of the  $n$ th roots of unity */
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 */
    double u = 2 * 3.1415926535897932385 / (double) n;
    register double s;

    for (j = 0; j < n; j++) sine[j] = sin(j * u);
    <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 Mykkeltveit's method is to compute an exponential sum  $s(a) = \sum a_j \omega^{(j-1)}$ , where  $\omega = e^{2\pi i/n}$ , and to avoid the "first" cyclic shift of the  $a$  array for which the imaginary part of  $s(a)$  is positive. (If no such shift exists, an arbitrary shift is chosen.)

```

⟨ Generate an n-tuple to avoid 3 ⟩ ≡
{
  d++;
  if (j < n) q = n;
  else {
    for (q = 1; ; q++) {
      for (l = 1, s = 0.0; l ≤ n; l++) s += a[l] * sine[(l - 1 + n - q) % n];
      if (s < .0001) break;
    }
    for (q++; q < n + n; q++) {
      for (l = 1, s = 0.0; l ≤ n; l++) s += a[l] * sine[(l - 1 + n + n - q) % n];
      if (s ≥ .0001) break;
    }
    if (q > n) q -= n;
  }
  for (code = 0, k = q + 1; k ≤ n; k++) code = m * code + a[k];
  for (k = 1; k ≤ 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.  $\langle$  Decrease the degree of  $l$ 's predecessors, and stack them if their degree drops to zero 5  $\rangle \equiv$

```

for ( $j = m - 1$ ;  $j \geq 0$ ;  $j--$ ) {
     $k = l + j * space$ ;
    if ( $\neg avoid[k]$ ) {
         $q = k/m$ ;
         $deg[q]--$ ;
        if ( $deg[q] \equiv 0$ )  $deg[q] = l, link[q] = top, top = q, count++$ ;
    }
}

```

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.

$\langle$  Print out a longest path 6  $\rangle \equiv$

```

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

```

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](#).  
*s*: [1](#).  
*sin*: [1](#).  
*sine*: [1](#), [3](#).  
*space*: [1](#), [2](#), [5](#), [6](#).  
*top*: [1](#), [2](#), [4](#), [5](#).  
*u*: [1](#).

- ⟨ 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.

UNAVOIDABLE2

	Section	Page
Intro .....	<a href="#">1</a>	1
Index .....	<a href="#">7</a>	4