November 24, 2020 at 13:24

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 nth 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 $\lambda = a_1 \ldots a_j$ is a prime string and $\alpha = a_1 \ldots a_n = \lambda^{n/j}$. 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. ⟨ Decrease the degree of l's predecessors, and stack them if their degree drops to zero 5 ⟩ ≡

$$
\begin{array}{l}
\textbf{for } (j = m - 1;\ j \ge 0;\ j{-}{-})\ \{\\
\quad k = l + j * \mathit{space};\\
\quad \textbf{if } (\neg \mathit{avoid}[k])\ \{\\
\qquad q = k/m;\\
\qquad \mathit{deg}[q]{-}{-};\\
\qquad \textbf{if } (\mathit{deg}[q] \equiv 0)\ \ \mathit{deg}[q] = l, \mathit{link}[q] = \mathit{top}, \mathit{top} = q, \mathit{count}{+}{+};\\
\quad \}\\
\}
\end{array}
$$

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 ⟩ ≡

$$
\begin{array}{l}
\mathit{printf}(\texttt{"Path:"});\\
\textbf{for } (\mathit{code} = k, j = 1;\ j < n;\ j{+}{+})\ \{\\
\quad \mathit{code} = \mathit{code} * m, q = \mathit{code}/\mathit{space};\\
\quad \mathit{printf}(\texttt{"␣\%d"}, q);\\
\quad \mathit{code} \mathrel{-}= q * \mathit{space};\\
\}\\
\textbf{while } (\mathit{deg}[k] \ge 0)\ \{\\
\quad \mathit{printf}(\texttt{"␣\%d"}, \mathit{deg}[k] \,\%\, m);\\
\quad k = \mathit{deg}[k];\\
\}
\end{array}
$$

This code is used in section 1.

7. Index.

UNAVOIDABLE2