The Icon Analyst

In-Depth Coverage of the Icon Programming Language and Applications

June 2000
Number 60

In this issue

Continued Fractions

Continued fractions are part of the "lost mathematics," the mathematics now considered too advanced for high school and too elementary for college.

— Petr Beckmann [1]

A continued fraction is a fraction in which the numerators and denominators may contain (continued) fractions. Displayed in their full laddered form, they are imposing:

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots}}}}}}}$$

See Figure 1 on the next page for other examples.

The numerators and denominators in a continued fraction can themselves be complicated, as evidenced by Figure 1i. Most work on continued fractions deals with *ordinary* continued fractions, in which the numerators and denominators are numbers:

$$a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \cfrac{b_3}{a_4 + \cfrac{b_4}{a_5 + \cfrac{b_5}{a_6 + \cfrac{b_6}{a_7 + \cfrac{b_7}{a_8 + \ldots}}}}}}}$$

Two sequences completely characterize an ordinary continued fraction: $a_1, a_2, a_3, a_4, \ldots$ and $b_1, b_2, b_3, b_4 \ldots$.

A *simple* continued fraction is an ordinary continued fraction in which all the numerators are 1 and all the denominators are integers and positive except possibly a_1:

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{a_5 + \cfrac{1}{a_6 + \cfrac{1}{a_7 + \cfrac{1}{a_8 + \ldots}}}}}}}$$

Only one sequence is needed to characterize a simple continued fraction. For example, the continued-fraction sequence for π is

3, 7, 15, 1, 292, 1, 1, 1, … .

As you' d expect, this sequence is infinite.

We'll stick to simple continued fractions in this article.

There are five important facts about simple continued-fraction sequences:

1. Rational numbers (fractions) have finite sequences. An example is 11/13, which has the sequence 0, 1, 5, 2.
2. Irrational numbers have infinite sequences.
3. Quadratic irrationals have periodic sequences. An example is $\sqrt{7}$, which has the sequence $2,\overline{1,1,1,4}$.

a

$$e-1=1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{4+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{6+\dots}}}}}}}}$$

b

$$\frac{1}{e-2}=1+\cfrac{1}{2+\cfrac{2}{3+\cfrac{3}{4+\cfrac{4}{5+\cfrac{5}{6+\cfrac{6}{7+\cfrac{7}{8+\dots}}}}}}}$$

c

$$\frac{\pi}{2}=1-\cfrac{1}{3-\cfrac{2\cdot 3}{1-\cfrac{1\cdot 2}{3-\cfrac{4\cdot 5}{1-\cfrac{3\cdot 4}{3-\cfrac{6\cdot 7}{1-\cfrac{5\cdot 6}{3-\dots}}}}}}}$$

d

$$\frac{4}{\pi}=1+\cfrac{1^2}{2+\cfrac{3^2}{2+\cfrac{5^2}{2+\cfrac{7^2}{2+\cfrac{9^2}{2+\cfrac{11^2}{2+\cfrac{13^2}{2+\dots}}}}}}}$$

e

$$\sin(x)=\cfrac{x}{1+\cfrac{x^2}{(2\cdot 3-x^2)+\cfrac{2\cdot 3x^2}{(4\cdot 5-x^2)+\cfrac{4\cdot 5x^2}{(6\cdot 7-x^2)+\dots}}}}$$

f

$$\tan(1)=1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{5+\cfrac{1}{1+\cfrac{1}{7+\cfrac{1}{1+\dots}}}}}}}}$$

g

$$\frac{1+\sqrt{5}}{2}=1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\dots}}}}}}}$$

h

$$\log(1+x)=\cfrac{x}{1+\cfrac{1^2x}{2+\cfrac{1^2x}{3+\cfrac{2^2x}{4+\cfrac{2^2x}{5+\cfrac{3^2x}{6+\cfrac{3^2x}{7+\dots}}}}}}}$$

i

$$\sqrt[3]{2}+2=\cfrac{1}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\cfrac{3+\cfrac{1}{3+\cfrac{3+\dots}{3+\dots}}}{3+\dots}}}}}}}$$

Figure 1. A Gallery of Continued Fractions

4. All other irrational numbers have non-periodic sequences. The sequence for π, shown above, is an example.
5. There is a one-to-one correspondence between an irrational number and its simple continued-fraction sequence. Furthermore, any infinite sequence of positive integers represents a unique irrational number. (For rational numbers, there are two equivalent sequences: one that ends … a_m, 1 and one that ends … $a_m - 1$.)

Computing Continued Fractions

Continued fractions are closely related to the familiar Euclidean algorithm for computing the greatest common divisor of two integers. Here's Euclid's algorithm cast as an Icon procedure:

```
procedure gcd(i, j)
   local r

   repeat {
      r := i % j
      if r = 0 then return j
      i := j
      j := r
      }

   return i

end
```

Next, we'll modify to code slightly to get a form that is easily modified to get continued fractions:

```
procedure gcd(i, j)
   local r

   until j = 0 do {
      r := i % j
      i := j
      j := r
      }

   return i                    # previous value of j

end
```

To generate the terms in the continued fraction for *i*/*j*, a line is needed to generate the denominators and the line that returns the greatest common is deleted:

```
procedure cfseq(i, j)
   local r

   until j = 0 do {
```

Srinivasa Ramanujan

1887-1920

Ramanujan stands as one of the greatest mathematical geniuses of all times. It's well to remember that genius is not the same as unusually high intelligence. Webster's 3rd has this definition: "extraordinary intellectual capacity for creative activity of any kind".

Ramanujan would *know* a mathematical result was true — and later prove it if pressed.

Ramanujan grew up in abject poverty in southern India. He lacked a university education but struck out on his own to follow his intense interest in mathematics.

In 1913, Ramanujan wrote a letter to the world-famous mathematician G. H. Hardy at Cambridge University seeking his support. In his letter, he included (without proof) some of his mathematical results. Here's one of several continued fractions in his letter:

$$\cfrac{1}{1+\cfrac{e^{-2\pi\sqrt{5}}}{1+\cfrac{e^{-4\pi\sqrt{5}}}{1+\cfrac{e^{-6\pi\sqrt{5}}}{1+\dots}}}} = \left[\frac{\sqrt{5}}{1+\sqrt[5]{\left(\frac{\sqrt{5}-1}{2}\right)^{\frac{5}{2}}\cdot 5^{\frac{3}{4}}}} - \frac{\sqrt{5}+1}{2}\right]\cdot e^{\frac{2\pi}{\sqrt{5}}}$$

Hardy at first dismissed the letter as the work of a crank. But he couldn't get Ramanujan's results out of his mind. He eventually concluded "They must be true, if they were not true, no one would have the imagination to invent them."

Ultimately Hardy arranged for Ramanujan to come to Cambridge. What followed was a most unusual and fruitful collaboration.

The story of Ramanujan's life is fascinating, poignant in places, and ultimately tragic.

(*continued*)

There are several books about the man and his contributions to mathematics [1-4]. We recommend the first — it's well worth reading.

References

1. *The Man Who Knew Infinity: A Life of the Genius Ramanujan*, Robert Kanigel, Washington Square Press, 1991.

2. *Srinivasa Ramanujan: A Mathematical Genius*, K. Srinivasa Rao, Eastwest Books, 1998.

3. *Ramanujan Letters and Commentary*, Bruce C. Brendt and Robert A. Rankin, American Mathematical Society, 1995.

4. *Ramanujan: The Man and the Mathematician*, S. R. Ranganthan, Asia Publishing House, 1967.

```
      suspend i / j
      r := i % j
      i := j
      j := r
      }
   end
```

This procedure can be adapted to handle real (floating-point) numbers by changing

```
   suspend i / j
```

to

```
   suspend integer(i / j)
```

The problem with trying to compute continued fractions for irrational numbers is that floating-point numbers are finite approximations of real numbers, and hence they really are rational numbers whose values are "close" to the corresponding real numbers. For example, standard 64-bit floating-point encoding for π is

$$7074237752028440/2^{51}$$

The corresponding continued-fraction sequence is, of course, finite:

3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 3, 3, 2, 1, 3, 3, 7, 2, 1, 1, 3, 2, 42, 2

and only the first 13 terms are the same as for the sequence for the actual irrational number:

3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2, 1, …

For what it's worth, as of this writing, the simple continued-fraction sequence for π has been computed to 20,000,000 terms. It is available on the Web <1>.

Convergents

As mentioned earlier, simple continued fractions for irrational numbers are infinite and can be represented by an integer sequence of the form a_1, a_2, a_3, a_4, … .

Finite initial seqences of such a sequence represent rational approximations to the irrational number, getting better as the initial subsequence becomes longer.

The rational numbers formed by initial sequences of an irrational sequence are called convergents.

One of the reasons continued fractions are important is that they often converge to the actual value much more rapidly than other approximation methods, such as power series.

For example, the first 10 convergents for the sequence for π are

3/1
22/7
333/106
355/113
103993/33102
104348/33215
208341/66317
312689/99532
833719/265381
1146408/364913

The fourth convergent, 355/113, already is accurate to six decimal places.

Convergents can be calculated using recurrence relations [5]. Here's a procedure that generates convergents:

```
procedure convergents(seq)
   local prev_p, prev_q, p, q, t

   seq := copy(seq)

   prev_p := [0, 1]
   prev_q := [1, 0]

   while t := get(seq) do {
      p := t * prev_p[2] + prev_p[1]
      q := t * prev_q[2] + prev_q[1]
      suspend rational(p, q, 1)
      prev_p[1] := prev_p[2]
```

```
    prev_p[2] := p
    prev_q[1] := prev_q[2]
    prev_q[2] := q
    }
end
```

Patterns

Simple continued-fraction sequences for rational numbers usually are short and any patterns are accidental.

Since quadratic irrationals have periodic simple continued-fraction sequences, they have patterns that may be of interest in graphic design.

Simple continued-fraction sequences for other irrationals are not periodic and most have no evident patterns.

Some, however, do. An example is tan(1) (see Figure 1f), whose simple continued-fraction sequence is

$$1,\overline{1,2n+1} \qquad n = 1, 2, 3, \ldots$$

Another example is e – 1 (see Figure 1a), whose simple continued-fraction sequence is

$$1,\overline{1,2n,1} \qquad n = 1, 2, 3, \ldots$$

Such sequences have periodic *forms*. The simple continued-fraction sequence for π has no such structure, but there is an ordinary continued-fraction for π/4 (see Figure 1d) that has numerator and denominator sequences with periodic forms:

numerators: $\overline{(2n-1)^2}$ $\quad n = 1, 2, 3, \ldots$

denominators: $1,\overline{2}$

Sequences with periodic forms are on our agenda.

Learning More About Continued Fractions

Much of the literature about continued fractions is highly technical and specialized. There are, however, a few books that are accessible [2-4]. There also are Web resources <2-6>.

Next Time

As mentioned earlier, simple continued-fraction sequences for quadratic irrationals are periodic. We'll take up this topic in the next article in our series on sequences.

References

1. *A History of Pi*, Petr Beckmann, Barnes & Noble, 1993.

2. *Continued Fractions*, C. D. Olds, Mathematical Association of America, 1963.

3. *The Higher Arithmetic*, H. Davenport, Cambridge University Press, 1999.

4. *What is Mathematics?*, Richard Courant and Herbert Robbins, Oxford University Press, 1996.

5. *Numerical Recipes: The Art of Scientific Computing*, William H. Press, Brian P Flannery, Saul A. Teukolsky, and William T. Vettering, Cambridge University Press, 1986, p. 136.

Links

1. http://www.lacim.uqam.ca/piDATA/CFPiTerms20.txt

2. http://mathworld.wolfram.com/topics/ContinuedFractions.html

3. http://archives.math.utk.edu/articles/atuyl/confrac/

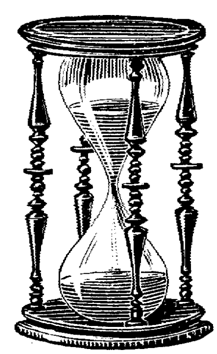
4. http://www.mathsoft.com/asolve/constant/cntfrc/cntfrc.html

5. http://www.mathsoft.com/asolve/constant/pi/frc.html

6. http://www.mathsoft.com/asolve/constant/e/cntfrc.html

Subscription Renewal

For many of you, this is your last issue in your Analyst subscription. If so, you'll find a renewal form in the center of this issue.



Don't miss an issue. Renew now.

Your prompt renewal helps us by reducing the number of follow-up notices we have to send. Knowing where we stand on subscriptions also lets us plan our budget for the next fiscal year.

Drafting Weavable Color Patterns

In two previous articles [1,2], we described the problem of determining if a color pattern can be woven and showed a heuristic method for solving the problem.

What remains is to use the results of a solution to create a draft — threading and treadling sequences and a tie-up.

A solved pattern provides color assignments for columns and rows — colors for the warp and weft threads. From this we can get a drawdown — a grid that shows where warp and weft threads are on top — by looking at the color of each point of intersection. Then from this we can get a draft [3].

At every point in the pattern, there are three possibilities:

1. The corresponding row and column colors are the same, in which case either the warp or weft thread can be on top.
2. The column color is the same as the color of the point, in which case the warp thread is on top.
3. The row color is the same as the color of the point, in which case the weft thread is on top.

The first case, an option point, presents a problem — how to choose? The choice potentially is important, because it can affect the length of floats [4] and the loom resources required. We'll come back to this later.

The output of the heuristic program with the –r (raw) option consists of three lines: the palette color keys for the columns, the color keys for the rows, and a color grid in terms of keys as one long string.

Figure 1 shows the solution for the pattern we used in the last article and which we'll continue to use here.

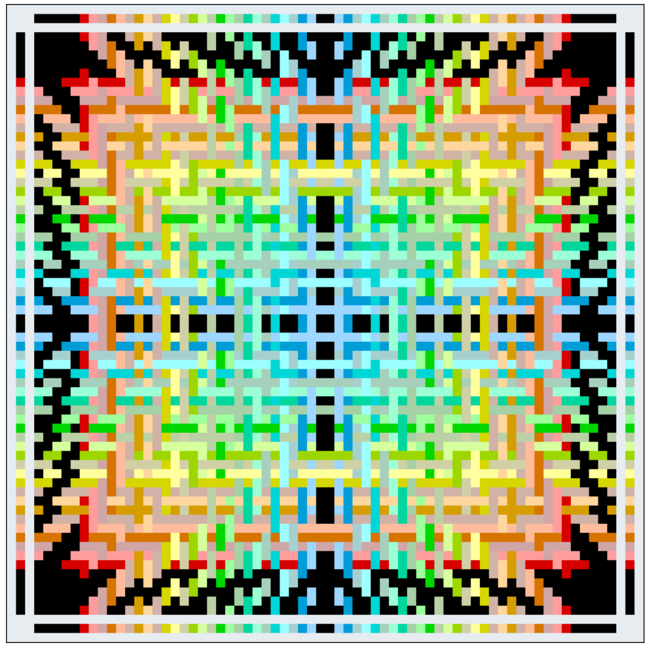


Figure 1. Solved Pattern

Figure 2 shows the output for –r. (Sometimes it's worth looking at raw data to get a feeling for what's really going on.)

```
00000Na@Ob%Pc%Qd.Re:Sf-Tg/Uh'Vi00iV'hU/gT-fS:eR.dQ%cP%bO@aN00000
00000Na@Ob%Pc%Qd.Re:Sf-Tg/Uh'Vi00iV'hU/gT-fS:eR.dQ%cP%bO@aN00000
00000Na@00%Pc%00000:0f0T00U00V0000V00U00T0f0:00000%cP%00@aN00000000000a@O00P0%Q0.00:00-Tg0U00Vi00iV00U0gT-00:00.0Q%0P00O@a000000
0000000@Ob0000Qd.R0000-0g/0h00i00i00h0/g0-0000R.dQ0000bO@000000000000000Ob%0c00d0Re0S0000/0h'000000'h0/0000S0eR0d00c0%bO00000000
00000N000b%Pc%0000e:Sf0T000h'V0000V'h000T0fS:e0000%cP%b000N00000000NNNaNNN%Pc%QN.NN:Nf-TgNUNNVi00iVNNUNgT-fN:NN.NQ%cP%NNNaNNN000
a000aaa@aaaPa%Qd.Ra:aa-Tg/UaaVi00iVaaU/gT-aa:aR.dQ%aPaaa@aaa000a@@000@@@O@@@@@Qd.Re@S@-@g/@@'@i@@i@'@@/g@-@S@eR.dQ@@@@@O@@@000@@
OOO00NOOObOOOOOdOReOSfOOO/Oh'OOOOOO'hO/OOOfSOeROdOOOOObOOON00OOO0bbb0Nabbb%bcbbbbbebSfbbbbUh'bbbbbb'hUbbbbfSbebbbbbcb%bbbaN0bbb0
00%%%Na@%%%Pc%%%%%%:%f%T%%U%%V%00%V%%U%%T%f%:%%%%%%cP%%%@aN%%%00P00PPPa@OPPPP%QP.PP:PP-TgPUPPVi00iVPPUPgT-PP:PP.PQ%PPPPO@aPPP00P
00cccNa@cc%Pc%ccccc:cfcTccUccVc00cVccUccTcfc:ccccc%cP%cc@aNccc00%00%%%a@O%%P%%Q%.%%:%%-Tg%U%%Vi00iV%%U%gT-%%:%%.%Q%%P%%O@a%%%00%
QQ00QQQ@ObQQQQQd.RQQQQ-Qg/QhQQiQQiQQhQ/gQ-QQQQR.dQQQQQbO@QQQ00QQ0dd00dddOb%dcddddRedSdddd/dh'dddddd'hd/ddddSdeRddddcd%bOddd00dd0
..00...@Ob....Qd.R....-.g/.h..i..i..h./g.-....R.dQ....bO@...00..0RR00RRROb%RcRRdRReRSRRRR/Rh'RRRRRR'hR/RRRRSReRRdRRcR%bORRR00RR0
00ee0Neeeb%Pc%eeeee:SfeTeeeh'Ve00eV'heeeTefS:eeeee%cP%beeeN0ee00:00:::a@O::P:%Q:.:::::-Tg:U::Vi00iV::U:gT-:::::.:Q%:P::O@a:::00:
00SS0NSSSb%Pc%SSSSe:SfSTSSSh'VS00SV'hSSSTSfS:eSSSS%cP%bSSSN0SS00000ffNafff%Pc%Qf.ff:ff-TgfUffVi00iVffUfgT-ff:ff.fQ%cP%fffaNff000
--00---@Ob----Qd.R------g/-h--i--i--h-/g------R.dQ----bO@---00--T00TTTa@OTTPT%QT.TT:TT-TgTUTTVi00iVTTUTgT-TT:TT.TQ%TPTTO@aTTT00T
gg00ggg@ObggggQd.Rgggg-gg/ghggiggigghg/gg-ggggR.dQggggbO@ggg00gg0//00///Ob%/c//d/Re/S//////h'//////'h//////S/eR/d//c/%bO///00//0
U000UUa@UUUPU%Qd.RU:UU-Tg/UUUVi00iVUUU/gT-UU:UR.dQ%UPUUU@aUU000U0hhh0Nahhb%hchhhhhehSfhhhhUh'hhhhhh'hUhhhhfShehhhhhch%bhhaN0hhh0
00''0N'''b%Pc%''''e:Sf'T'''h'V'00'V'h'''T'fS:e''''%cP%b'''N0''00V00VVVa@OVVPV%QV.VV:VV-TgVUVVVi00iVVVUVgT-VV:VV.VQ%VPVVO@aVVV00V
ii00iii@ObiiiiQd.Riiii-ig/ihiiiiiiiihi/gi-iiiiR.dQiiiibO@iii00ii000000a@O00P0%Q0.00:00-Tg0U00Vi00iV00U0gT-00:00.0Q%0P00O@a000000
000000a@O00P0%Q0.00:00-Tg0U00Vi00iV00U0gT-00:00.0Q%0P00O@a000000ii00iii@ObiiiiQd.Riiii-ig/ihiiiiiiiihi/gi-iiiiR.dQiiiibO@iii00ii
V00VVVa@OVVPV%QV.VV:VV-TgVUVVVi00iVVVUVgT-VV:VV.VQ%VPVVO@aVVV00V00''0N'''b%Pc%''''e:Sf'T'''h'V'00'V'h'''T'fS:e''''%cP%b'''N0''00
0hhh0Nahhb%hchhhhhehSfhhhhUh'hhhhhh'hUhhhhfShehhhhhch%bhhaN0hhh0U000UUa@UUUPU%Qd.RU:UU-Tg/UUUVi00iVUUU/gT-UU:UR.dQ%UPUUU@aUU000U
0//00///Ob%/c//d/Re/S//////h'//////'h//////S/eR/d//c/%bO///00//0gg00ggg@ObggggQd.Rgggg-gg/ghggiggigghg/gg-ggggR.dQggggbO@ggg00gg
T00TTTa@OTTPT%QT.TT:TT-TgTUTTVi00iVTTUTgT-TT:TT.TQ%TPTTO@aTTT00T--00---@Ob----Qd.R------g/-h--i--i--h-/g------R.dQ----bO@---00--
000ffNafff%Pc%Qf.ff:ff-TgfUffVi00iVffUfgT-ff:ff.fQ%cP%fffaNff00000SS0NSSSb%Pc%SSSSe:SfSTSSSh'VS00SV'hSSSTSfS:eSSSS%cP%bSSSN0SS00
:00:::a@O::P:%Q:.:::::-Tg:U::Vi00iV::U:gT-:::::.:Q%:P::O@a:::00:00ee0Neeeb%Pc%eeeee:SfeTeeeh'Ve00eV'heeeTefS:eeeee%cP%beeeN0ee00
0RR00RRROb%RcRRdRReRSRRRR/Rh'RRRRRR'hR/RRRRSReRRdRRcR%bORRR00RR0..00...@Ob....Qd.R....-.g/.h..i..i..h./g.-....R.dQ....bO@...00..
0dd00dddOb%dcddddRedSdddd/dh'dddddd'hd/ddddSdeRddddcd%bOddd00dd0QQ00QQQ@ObQQQQQd.RQQQQ-Qg/QhQQiQQiQQhQ/gQ-QQQQR.dQQQQQbO@QQQ00QQ
%00%%%a@O%%P%%Q%.%%:%%-Tg%U%%Vi00iV%%U%gT-%%:%%.%Q%%P%%O@a%%%00%00cccNa@cc%Pc%ccccc:cfcTccUccVc00cVccUccTcfc:ccccc%cP%cc@aNccc00
P00PPPa@OPPPP%QP.PP:PP-TgPUPPVi00iVPPUPgT-PP:PP.PQ%PPPPO@aPPP00P00%%%Na@%%%Pc%%%%%%:%f%T%%U%%V%00%V%%U%%T%f%:%%%%%%cP%%%@aN%%%00
0bbb0Nabbb%bcbbbbbebSfbbbbUh'bbbbbb'hUbbbbfSbebbbbbcb%bbbaN0bbb0OOO00NOOObOOOOOdOReOSfOOO/Oh'OOOOOO'hO/OOOfSOeROdOOOOObOOON00OOO
@@000@@@O@@@@@Qd.Re@S@-@g/@@'@i@@i@'@@/g@-@S@eR.dQ@@@@@O@@@000@@a000aaa@aaaPa%Qd.Ra:aa-Tg/UaaVi00iVaaU/gT-aa:aR.dQ%aPaaa@aaa000a
000NNNaNNN%Pc%QN.NN:Nf-TgNUNNVi00iVNNUNgT-fN:NN.NQ%cP%NNNaNNN00000000N000b%Pc%0000e:Sf0T000h'V0000V'h000T0fS:e0000%cP%b000N00000
00000000Ob%0c00d0Re0S0000/0h'000000'h0/0000S0eR0d00c0%bO000000000000000@Ob0000Qd.R0000-0g/0h00i00i00h0/g0-0000R.dQ0000bO@0000000
000000a@O00P0%Q0.00:00-Tg0U00Vi00iV00U0gT-00:00.0Q%0P00O@a00000000000Na@00%Pc%00000:0f0T00U00V0000V00U00T0f0:00000%cP%00@aN00000
```

Figure 2. Solution Output

Here's a program that reads raw output in the form of Figure 2 and produces an ISD [5].

```
link numbers
link options
link patutils
link patxform
link weavutil
link xcode

procedure main(args)
   local warp, weft, pattern, rows, i, j, opts, count
   local threading, treadling, color_list, colors, choice
   local symbols, symbol, drawdown, draft
   local warp_colors, weft_colors, pixels

   opts := options(args, "o+")

   choice := opts["o"] | 1

   (warp := read()  & weft := read() &
      pattern := read()) | stop("*** short file")

   pixels := real(*pattern)

   colors := warp ++ weft

   color_list := [ ]
   warp_colors := []
   weft_colors := []
   drawdown := []

   every put(color_list, PaletteColor("c1", !colors))

   every put(warp_colors, upto(!warp, colors))
   every put(weft_colors, upto(!weft, colors))

   pattern ? {
      while put(drawdown, move(*warp))
      }

   count := 0

   every i := 1 to *weft do {
      every j := 1 to *warp do {
         if weft[i] == warp[j] then {
            count +:= 1
            drawdown[i, j] := case choice of {
               0  :  ?2 – 1                # random
               1  :  "1"                   # warp
               2  :  "0"                   # weft
               3  :  1 – (count % 2)       # alternate
               }
            }
         else if drawdown[i, j] == weft[i]
            then drawdown[i, j] := "0"
         else drawdown[i, j] := "1"
         }
      }

   treadling := analyze(drawdown)
   drawdown := protate(drawdown, "cw")
   threading := analyze(drawdown)

   symbols := table("")

   every pattern := !treadling.patterns do {
      symbol := treadling.rows[pattern]
      symbols[symbol] := repl("0", *threading.rows)
      pattern ? {
         every i := upto('1') do
          symbols[symbol][threading.sequence[i]] := "1"
         }
      }

   symbols := sort(symbols, 3)
   rows := []

   while get(symbols) do
      put(rows, get(symbols))

   draft := isd()

   draft.name := "colorup"
   draft.threading := threading.sequence
   draft.treadling := treadling.sequence
   draft.warp_colors := warp_colors
   draft.weft_colors := weft_colors
   draft.color_list := color_list
   draft.shafts := *threading.rows
   draft.treadles := *treadling.rows
   draft.tieup := rows

   xencode(draft, &output)

end
```

The command-line option –o provides four alternatives for handling option points:

0: random choice

1: chose warp (the default)

2: chose weft

3: chose warp and weft alternately

Figure 3 on the next page shows a warp-choice draft obtained in this manner using –o 1 for the pattern shown Figure 1.

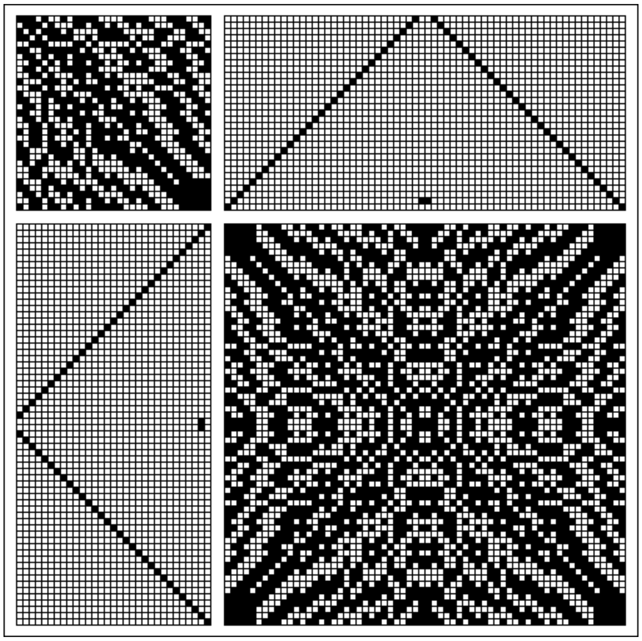


Figure 3. Warp-Choice Draft

Option Points

For many patterns that might be candidates for weaving, the number of option points is large. For Figure 1, 256 of the 4,096 points are option points. So there are 2^{256} possible drafts.

It’s clearly hopeless to explore even a small fraction of possible drafts that result from different choices at option points. Trying each of the four methods usually gives an idea of how important the method used is.

The choices made at option points affects float lengths. See Figures 4 through 7.

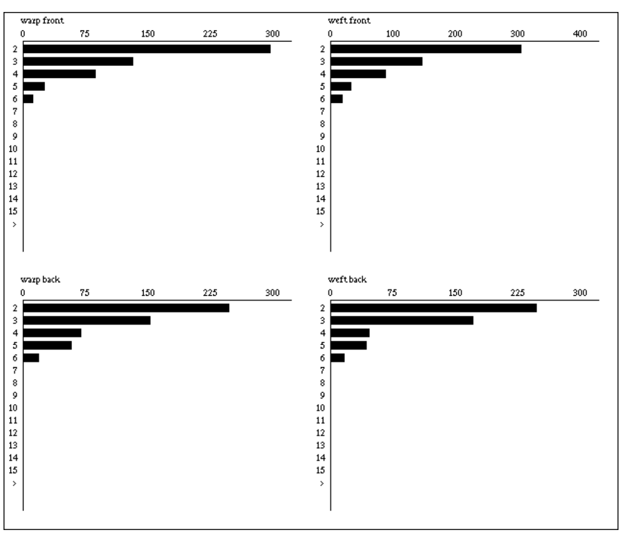


Figure 4. Random-Choice Floats

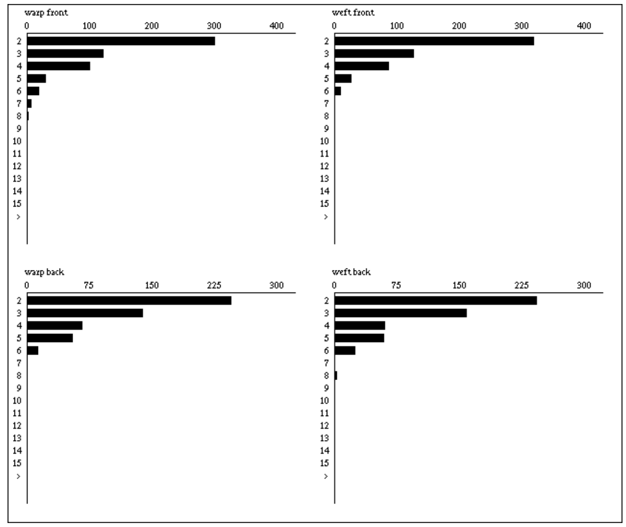


Figure 5. Warp-Choice Floats

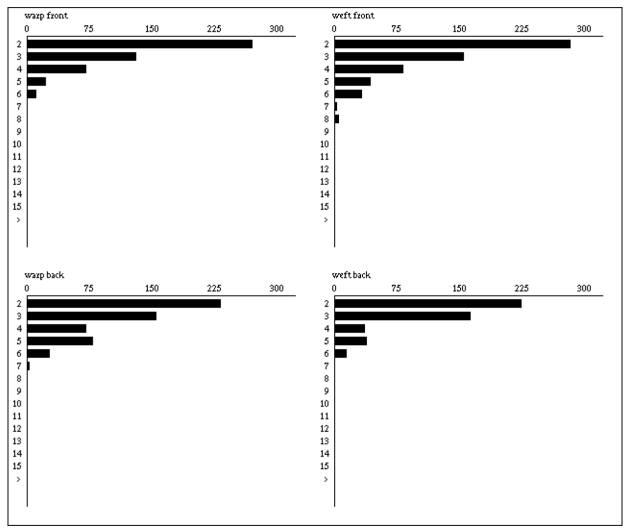


Figure 6. Weft-Choice Floats

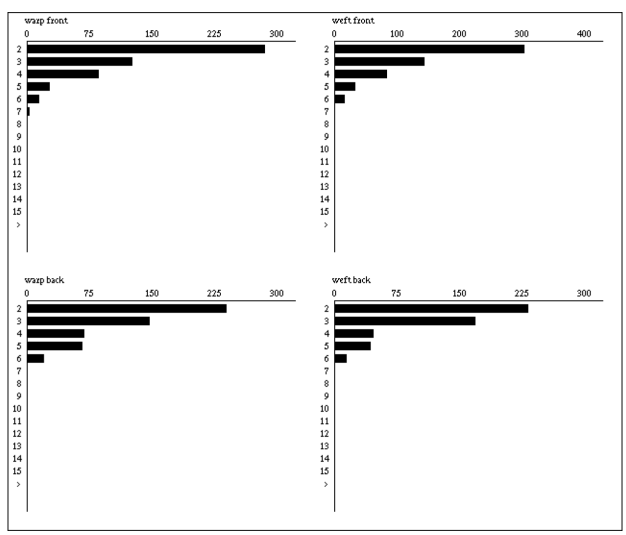


Figure 7. Alternating-Choice Floats

Another, often more important, consideration is the number of shafts and treadles the draft requires. The warp-choice draft shown in Figure 3 requires 31 shafts and 31 treadles. The weft-choice draft, shown in Figure 8, requires only 16 shafts and 16 treadles.

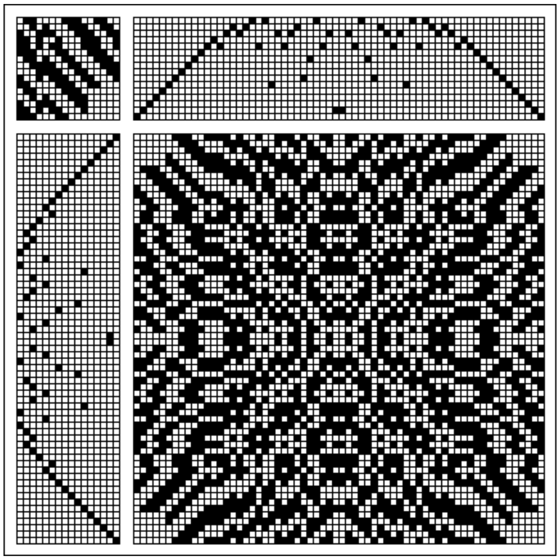


Figure 8. Weft-Choice Draft

Very few weavers have looms that can handle drafts that require 31 shafts and treadles, but many have looms with 16 shafts and treadles. More strikingly, the random-choice draft requires 56 shafts and 52 treadles, while the alternating choice draft requires 62 shafts and 31 treadles, making them out of the question for actual weaving.

Next Time

The question of what color patterns are weavable has led us to develop an application in which the user can create weavable color patterns.

We'll describe this application and comment on the problem from a designer's viewpoint in the next article in this series.

References

1. "Weavable Color Patterns", Icon Analyst 58, pp. 7-10.

2. "Weavable Color Patterns", Icon Analyst 59, pp. 10-15.

3. "Drawups", Icon Analyst 56, pp. 18-20.

4. "Floats", Icon Analyst 59, pp. 1-3.

5. "Weave Draft Representation", Icon Analyst 56, pp. 1-3.

Polyalphabetic Substitution

L oryh wuhdvrq exw kdwh d wudlwru.

— Mxolxv Fdhvdu

The first article in this series [1] described monoalphabetic substitution ciphers in which characters of the plain text are replaced on a one-for-one basis by characters of a cipher alphabet.

Monoalphabetic cryptograms are easy to decrypt because of the one-for-one correspondence. Known letter frequencies and patterns usually reveal a few characters and the rest follow by context.

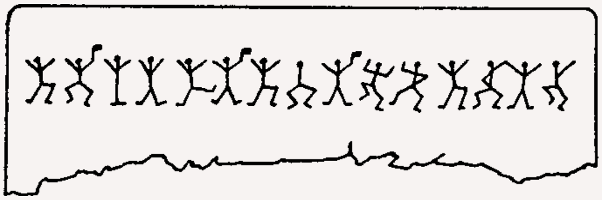
This weakness of monoalphabetic ciphers can be overcome by using multiple alphabets and using different alphabets at different positions in the plain text.

There are many types of such polyalphabetic substitution ciphers. The Vigenére Square is best known and illustrates the principles [2-5].

In a Vigenére Square, there is a cipher alphabet for each character in the plain alphabet. The cipher alphabets can be formed in any manner provided they are distinct.

The simplest Vigenére Square is due to Charles Lutwidge Dodgson (Lewis Carroll). It consists of the plain alphabet successively rotated. For a plain alphabet consisting of the lowercase letters, it looks like this:

```
abcdefghijklmnopqrstuvwxyz
bcdefghijklmnopqrstuvwxyza
cdefghijklmnopqrstuvwxyzab
defghijklmnopqrstuvwxyzabc
efghijklmnopqrstuvwxyzabcd
fghijklmnopqrstuvwxyzabcde
ghijklmnopqrstuvwxyzabcdef
hijklmnopqrstuvwxyzabcdefg
ijklmnopqrstuvwxyzabcdefgh
jklmnopqrstuvwxyzabcdefghi
klmnopqrstuvwxyzabcdefghij
lmnopqrstuvwxyzabcdefghijk
mnopqrstuvwxyzabcdefghijkl
nopqrstuvwxyzabcdefghijklm
opqrstuvwxyzabcdefghijklmn
pqrstuvwxyzabcdefghijklmno
qrstuvwxyzabcdefghijklmnop
rstuvwxyzabcdefghijklmnopq
stuvwxyzabcdefghijklmnopqr
tuvwxyzabcdefghijklmnopqrs
uvwxyzabcdefghijklmnopqrst
vwxyzabcdefghijklmnopqrstu
wxyzabcdefghijklmnopqrstuv
xyzabcdefghijklmnopqrstuvw
yzabcdefghijklmnopqrstuvwx
zabcdefghijklmnopqrstuvwxy
```



Cryptology in Literature

Coded messages have appeared many times as a plot element in popular fiction. A multiply-encoded Latin cryptogram sent Jules Verne's characters on their *Voyage to the Center of the Earth*. Simple substitution ciphers using unusual alphabets figure prominently in two short stories that achieved great popularity.

"The Gold Bug", by Edgar Allen Poe, won a $100 prize and was published in the *Dollar Newspaper* of Philadelphia in 1843. In this improbable but captivating tale, a recluse on a South Carolina island deciphers a message that leads him to Captain Kidd's buried treasure chest. Poe's clear explanation of the cryptanalysis process popularized the subject for the first time.

In "The Adventure of the Dancing Men", by Arthur Conan Doyle, Sherlock Holmes encounters a series of whimsical-looking messages; the first is reproduced above. Holmes breaks the code and sends a message of his own to catch the criminal. Again a straightforward exposition, ostensibly for the benefit of Dr. Watson, brought cryptanalysis to the general public.

In *The Codebreakers* [1], his authoritative book on cryptology, David Kahn cites many other literary examples including a nonfiction treatise on the use of the astrolabe by Geoffrey Chaucer. Kahn analyzes errors that have persisted since the first printing of "Dancing Men" and concludes that we must blame Dr. Watson's transcriptions, for if they appeared in the original messages they would have foreclosed Holmes' method of solution.

Reference

[1] *The Codebreakers*, David Kahn , Macmillan, 1996.

A key with characters from the plain alphabet is used to select the alphabets for enciphering. The characters of the keys are used in order, cyclically. For example, the key kaleidoscope uses the k alphabet to encipher the first character of the plain text, the a alphabet for the second, and so on, continuing with k after the final e. For example, using the Carroll Vigenére Square, the plain text

thaw the casserole for dinner

gives the cryptogram

dhla wvw qpwcecsth xqf hsnyiz

Another classical method of producing a Vigenére Square is to start with a *keyed alphabet* in place of the plain alphabet. A keyed alphabet is constructed by using a string of plain-text characters. The keyed alphabet begins with the distinct characters of the key, followed by the remaining characters of the plain alphabet in their usual order. For example, the key hexamorph with the plain alphabet consisting of lowercase letters produces the keyed alphabet

hexamorpbcdfgijklnqstuvwyz

The resulting Vigenére Square with the alphabets ordered by their first characters is:

```
amorpbcdfgijklnqstuvwyzhex
bcdfgijklnqstuvwyzhexamorp
cdfgijklnqstuvwyzhexamorpb
dfgijklnqstuvwyzhexamorpbc
examorpbcdfgijklnqstuvwyzh
fgijklnqstuvwyzhexamorpbcd
gijklnqstuvwyzhexamorpbcdf
hexamorpbcdfgijklnqstuvwyz
ijklnqstuvwyzhexamorpbcdfg
jklnqstuvwyzhexamorpbcdfgi
klnqstuvwyzhexamorpbcdfgij
lnqstuvwyzhexamorpbcdfgijk
morpbcdfgijklnqstuvwyzhexa
nqstuvwyzhexamorpbcdfgijkl
orpbcdfgijklnqstuvwyzhexam
pbcdfgijklnqstuvwyzhexamor
qstuvwyzhexamorpbcdfgijkln
rpbcdfgijklnqstuvwyzhexamo
stuvwyzhexamorpbcdfgijklnq
tuvwyzhexamorpbcdfgijklnqs
uvwyzhexamorpbcdfgijklnqst
vwyzhexamorpbcdfgijklnqstu
wyzhexamorpbcdfgijklnqstuv
xamorpbcdfgijklnqstuvwyzhe
yzhexamorpbcdfgijklnqstuvw
zhexamorpbcdfgijklnqstuvwy
```

For the plain text and selection key used in the example above, the cryptogram is bdlw agw ppspppkyj ywv mwlaom

Keys that select alphabets in an irregular fashion provide more security than keys that don't, and long keys that use more alphabets provide more security than short keys. A key consisting of a single character obviously is unacceptable. The seven-character key security, in which all the characters are different, is preferable to the seven-character key selects, in which there are only five distinct characters and is equivalent to the five-character key selct.

Using a key to select alphabets in a cyclic manner makes the resulting cryptograms suspectable to cryptographic techniques [6]. One method used to avoid this problem is a *running key* that consists of the plain-text characters of some passage of text known to both the encipherer and decipherer. For example, with case folded, Lincoln's Gettysburg address for the lowercase letters gives the running key

fourscoreandsevenyearsagoourfathersbroughtforth
onthiscontinentanewnationconceived …

For the plain text and keyed alphabet used in the examples above, the cryptogram is

mgua xgf onxfoikxx bud kiqbff

A running key is, of course, just a key at least as long as the message. The advantage of using such a key known to both the encipher and decipherer is that the key itself does not have to be transmitted; only an identification for it.

Another method for choosing cipher alphabets in a noncyclic fashion is *auto-key enciphering*, in which the characters of the plain text are used to select the cipher alphabets. It is, of course, necessary to know where to start. For auto-key enciphering, this key is the first character of the starting alphabet. For the key j, the plain text for our example produces the cryptogram

pehz tem ccufwqtlt fzv dqhmuq

Classical methods were devised to be easy to use and to minimize the possibility of errors. They therefore sometimes seem simplistic. When using computer programs to implement ciphers, these concerns are largely irrelevant

For the Vigenére Square, the alphabets can be constructed using any technique that produces distinct alphabets. The key used for selecting alphabets also can be of any kind as long as many

Letter Frequencies

The frequencies with which letters occur in written material vary from language to language and somewhat depending on the subject matter. However, there is considerable consistency, which aids in decrypting. Here are two lists of letter frequencies based on two large corpora for American and British English.

American			British		
e	577230	12.68	e	588441	12.51
t	418668	9.20	t	435707	9.26
a	364302	8.00	a	378602	8.05
o	345419	7.59	o	357304	7.59
i	330074	7.25	i	342873	7.29
n	323360	7.10	n	333890	7.10
s	293976	6.46	s	307900	6.54
r	281270	6.18	r	288319	6.13
h	255365	5.61	h	255817	5.44
l	188647	4.14	l	194577	4.14
d	181973	4.00	d	186853	3.97
c	133292	2.93	c	145711	3.10
u	125487	2.76	u	127675	2.71
m	112287	2.47	m	119566	2.54
f	106172	2.33	f	108816	2.31
g	89612	1.97	p	94928	2.02
w	88413	1.94	g	91690	1.95
p	85086	1.87	w	88639	1.88
y	81787	1.80	y	81175	1.73
b	70994	1.56	b	72257	1.54
v	45186	0.99	v	46948	1.00
k	30182	0.66	k	30946	0.66
x	10081	0.22	x	9320	0.20
j	6462	0.14	j	7549	0.16
q	5079	0.11	q	5039	0.11

Notice that the only difference in order of frequency of occurrence is for p, g, and w.

It is possible, of course, to deliberately distort letter frequencies in messages. In fact, an entire novel was written without using the letter e [1]. It's said that the missing letter is barely noticeable. The novel also is reported to be dreadful. Don't bother to look for a copy; only a few hundred were printed and existing copies are exceedingly rare.

Incidentally, a composition that deliberately omits certain letters is called a lipogram [2].

References

1. *Gadsby*, Ernest Vincet Wright, Wetzel, 1939.

2. *The Game of Words*, Willard R. Espy, Bramhall House, 1971.

(preferably all) alphabets are used in a nonregular order. Alphabets do not have to be selected by their first character; in fact, it's easier to index into a list. Thus, selection keys can be integer sequences modulo the number of alphabets.

Implementation

A procedure for producing keyed alphabets is:

```
procedure key_alpha(plain, alpha)

   /alpha := &lcase              # default alphabet

   plain := ochars(plain)        # unique characters

   return plain || deletec(alpha, plain)

end
```

The procedures ochars() and deletec() are from the strings module in the Icon program library.

For classical methods, a table is the natural way to represent a Vigenére Square. For a keyed alphabet, it looks like this:

```
alpha := key_alpha(alpha_key, plain_alpha)

vigenere := table()

every 1 to *alpha do {
   vigenere[alpha[1]] := alpha
   alpha := rotate(alpha, 1)
   }
```

The enciphering goes as follows:

```
while plain := read() do {
   crypto := ""
   key := sel_key                # selection key
   every c := !plain do {
      crypto ||:=
         map(c, plain_alpha,  vigenere[key[1]])
      key := rotate(key, 1)
      }
   write(crypto)
   }
```

The deciphering is symmetric:

```
while crypto := read() do
   plain := ""
   key := sel_key
   every c := !crypto do {
      plain ||:=
         map(c, vigenere[key[1]], plain_alpha)
      key := rotate(key, 1)
      }
   write(plain)
   }
```

Auto-key encipheringis similar except for the use of the plain text to construct the selection key:

```
while plain := read() do {
   crypto := ""
   key := sel_key || plain
   every c := !plain do {
      crypto ||:=
       map(c, plain_alpha,  vigenere[key[1]])
      key := rotate(key, 1)
      }
   write(crypto)
   }

     …

while crypto := read() do {
   key := sel_key || plain
   plain := ""
   every c := !crypto do {
      plain ||:=
         map(c, vigenere[key[1]], plain_alpha)
      key := rotate(key, 1)
      }
   write(plain)
   }
```

Next Time

In the next article on classical cryptography, we'll take up polygram substitution, in which substitution is based on groups of characters instead of single characters.

Following that, we'll start on transposition ciphers.

References

1. "Classical Cryptography", Icon Analyst 59, pp. 7-9.

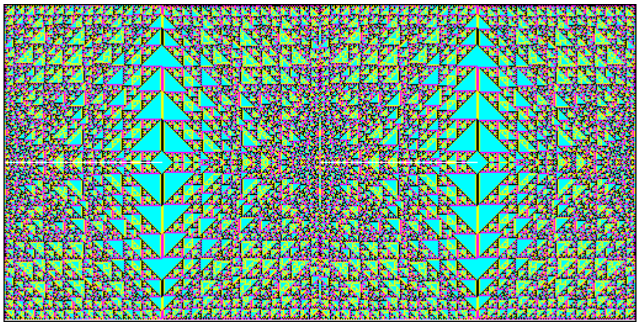
2. *Cryptanalysis: A Study of Ciphers and Their Solution*, Helen Fouché Gaines, Dover, 1956.

3. *The Codebreakers*, David Kahn, Scribner, 1996.

4. *Cryptography: The Science of Secret Writing*, Laurence Dwight Smith, Dover, 1943.

5. *Codes, Ciphers, and Secret Writing*, Martin Gardener, Dover, 1972.

6. *Elementary Cryptanalysis: A Mathematical Approach*, Abraham Sinkov, Random House, 1968.



Graphics Corner — Creating Custom Palettes

In a previous article, we described custom palettes as an addition to Icon's built-in palette mechanism [1]. In this article, we'll describe some tools for creating custom palettes.

Derived Custom Palettes

One way to create a custom palette is to take colors from an existing source.

Color Lists

There are many sources of color lists. For example, we have some left over from numerical carpets [2] and many color lists are available for the popular fractal program Fractint [3].

If a file contains one color specification per line using any of the ways that Icon can represent colors [4], creating a custom palette from these colors is very simple. Here's a program that creates a custom palette database with palettes from color lists whose file names are given on the command line.

```
link basename
link palettes
link xcode

global PDB_

procedure main(args)
   local file, input, clist, name

   every file := !args do {
      input := open(file) | {
         write(&errout, "*** cannot open ", image(file))
         next
         }
      name := basename(file, ".clr")
      clist := []
      while put(clist, read(input))
      close(input)
      makepalette(name, clist) |
         write(&errout, "*** could not make palette for ",
            image(file))
      }
   xencode(PDB_, &output)

end
```

Fracint color lists (called maps) give RGB color specifications one per line, but the range of intensity is from 0 to 255, so it's necessary to change the range for Icon:

```
                   …
      clist := []
      while line := read(input) do {
         line ? {
            tab(upto(&digits))
            color := (tab(many(&digits)) * 257) || ","
            tab(upto(&digits))
            color ||:= (tab(many(&digits)) * 257) || ","
            tab(upto(&digits))
            color ||:= (tab(many(&digits)) * 257)
            }
         put(clist, color)
                   …
```

WIFs [5] contain color palettes that are embedded along with other data. The color range can be specified and varies from file to file. The WIF format is verbose and rather messy to parse, but the color palettes in WIFs are useful in weaving, so it's worth the effort to create custom palettes from them:

```
link basename
link palettes
link xcode

global PDB_

procedure main(args)
   local file, wifname, input, clist, line, range, i

   every file := !args do {
      wifname := basename(file, ".wif")
      input := open(file) | {
         write(&errout, "*** cannot open ", image(file))
         next
         }
      clist := []
      range := &null
      while line := trim(map(read(input))) do {
         if line == "[color table]" then {
            while line := trim(read(input)) do {
               if *line = 0 then break
               line ?:= {
```

```
            if ="[" then break
            tab(upto('=') + 1)
            tab(0)
            }
          put(clist, line)
          }
        }
      else if line == "[color palette]" then {
        while line := trim(map(read(input))) do {
          if *line = 0 then break
          line ? {
            if ="[" then break
            else if ="range=" then {
              tab(upto(',') + 1)
              range := tab(0) + 1
              break
              }
            }
          }
        }
      }
    close(input)

    if (\range ~= 65536) then {   # adjust RGB values
      every i := 1 to *clist do
        clist[i] := color_range(clist[i], range)
      }
    makepalette(wifname, clist)
    }

  xencode(PDB_, &output)

end
```

Images

Images provide handy sources of colors. The following program creates a custom palette database from GIF images whose file names are given on the command line:

```
link basename
link palettes
link graphics
link xcode

global PDB_

procedure main(args)
  local file, name, output, colors, win

  every file := !args do {
    win := WOpen("image=" || file, "canvas=hidden") |
      {
      write(&errout, "*** cannot open image: ",
        image(file))
      next
      }
    name := basename(file, ".gif")
    colors := set()
    every insert(colors, Pixel(win))
    WClose(win)
    makepalette(name, sort_colors(colors))
    }

  xencode(PDB_, &output)

end
```

There are many possibilities for giving a user control over the selection of colors from an image. We'll defer that subject for now.

An Interactive Custom-Palette Application

The creation of custom palettes invites user interaction. There are many issues in the design of such an application. The problem obviously is open-ended and vulnerable to over-generalization and excessive complexity.

The basic functionality we've chosen includes these features:

- creating custom palettes in a variety of ways
- modifying existing custom palettes in a variety of ways
- viewing custom palettes
- saving custom palettes in databases
- loading databases of custom palettes

Figure 1 shows the application interface.

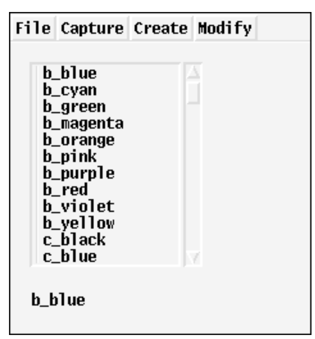


Figure 1. Interface

The scrolling text list displays the names of palettes in the current database. One palette is the current focus of attention and its name is shown at the bottom of the window.

Clicking on a name in the text list brings up a dialog. See Figure 2.

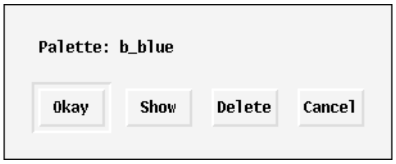


Figure 2. A Palette Dialog

The chosen palette can be approved, in which case it is made the current one, displayed, or deleted. A palette is displayed in a separate window in the style of the palette program in the Icon program library. Figure 3 shows an example.

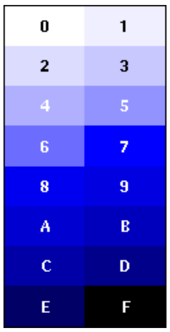


Figure 3. A Custom Palette

The File menu, shown in Figure 4, has items for opening and saving palette databases and quitting the application.

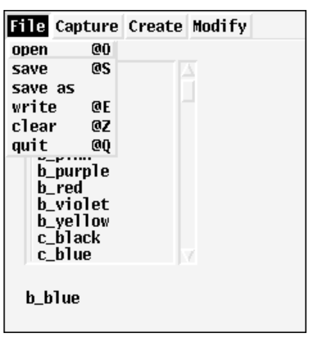


Figure 4. File Menu

The write item (@E) writes a list of colors in the current palette. The clear item (@Z) discards the current database an creates a new, empty one.

The remaining functionality is divided into three parts: capture, creation, and modification.

Capturing Colors

The Capture menu, shown in Figure 5, provides items for creating a custom palette from a file of color specifications or an image.

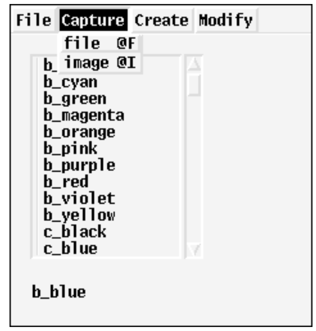


Figure 5. Capture Menu

At present, the methods of capturing colors are limited to getting colors from color list and image files. Possible extensions include specifying the format of a file that contains color specifications and providing various ways by which a user can select desired colors from an image.

Creating New Palettes

The Create menu, shown in Figure 6, provides other ways for creating new palettes.

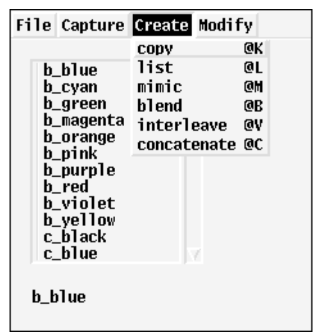


Figure 6. Create Menu

The copy item (@K) in the Create menu makes a copy of a palette already in the database.

The list item (@L) allows the user to enter color specifications in a text dialog. See Figure 7.

Supplementary Material

Supplementary material for this issue of the Analyst, including images and Web links, is available on the Web. The URL is

http://www.cs.arizona.edu/icon/analyst/iasub/ia60/

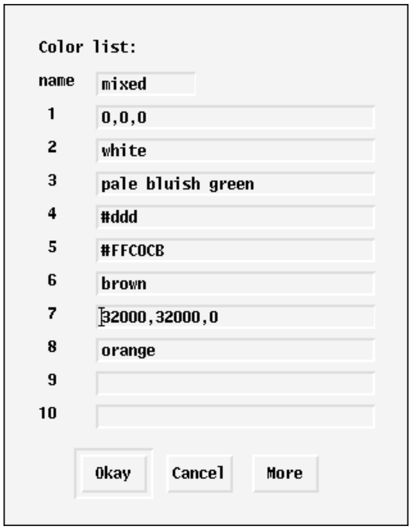


Figure 7. A List Dialog

The More button provides an additional text dialog if more colors are needed.

It's worth noting that it's possible to create a palette with just one color. Such single-colored palettes are useful for creating other kinds of palettes.

The mimic item (@M) allows the creation of a custom palette with the same colors as a built-in palette. See Figure 8. This is useful for creating a custom palette with modifications to a built-in palette.

Figure 8. A Mimic Dialog

The blend item (@B) facilitates the creation of palettes with colors in equally spaced steps between beginning and ending colors. The number of steps between pairs of colors is specified as shown in Figure 9.

Back Issues

Back issues of *The Icon Analyst* are available for $5 each. This price includes shipping in the United States, Canada, and Mexico. Add $2 per order for airmail postage to other countries.

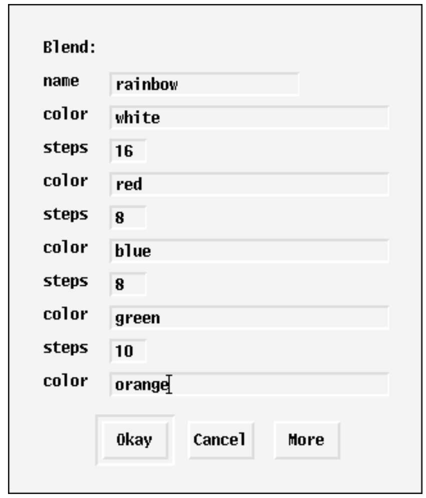


Figure 9. A Blend Dialog

Figure 10 shows the resulting palette.

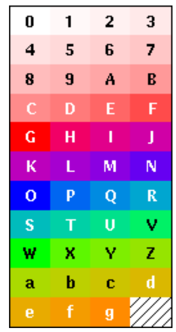


Figure 10. The Rainbow Palette

As usual, it's better to view such images in color. They are available on the Web page for this issue of the *Analyst*.

The interleave item (@V) creates a palette with the colors of two palettes interleaved. Figure 11 shows an example of the dialog.

Figure 11. An Interleave Dialog

If one palette is shorter than the other, it is extended by repetition as necessary.

At present, only two palettes can be interleaved. This limitation could be removed, but a use for the more general feature needs to be demonstrated.

The concatenate item (@C) allows several palettes to be concatenated to form a new palette. See Figure 12.

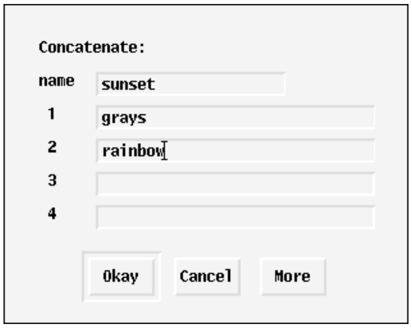


Figure 12. A Concatenation Dialog

Modifying Existing Palettes

The Modify menu, shown in Figure 13, provides ways for changing existing palettes.

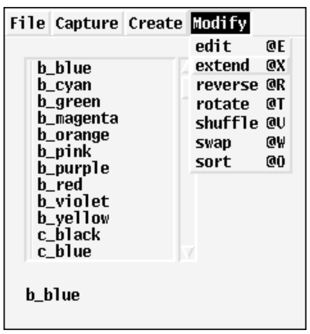


Figure 13. Modify Menu

Items in the Modify menu apply to the current palette.

The edit item (@E) is intended to allow the user to edit a palette interactively. This feature is not yet implemented. See the remarks in the next section.

The extend (@X) item extends a palette to a specified number of colors by repetition. If the specified number is less than the number of colors in the palette, the palette is truncated.

The remaining menu items allow the colors in existing palettes to be re-ordered.

The reverse (@R) item reverses the order of the colors in a palette, while the rotate item (@T) rotates them by a specified amount.

The shuffle item (@U) randomizes the order of the colors in a palette, and the swap (@W) item swaps adjacent colors.

The sort item (@O) sorts the colors of a palette by intensity. Other sorting methods could be added.

Editing Palettes

There are many problems with providing facilities for interactively editing palettes. The relatively weak features in commercial paint programs suggest the problem is fundamentally difficult.

It's easy to imagine things that would be useful, such as:w

- adding new colors
- deleting existing colors
- changing existing colors
- rearranging the order of colors

On the surface, these may seem simple, but it's not so easy to design good methods for doing them, and implementation may be tricky.

We haven't come up with satisfactory specifications for editing palettes. At present the edit item just produces a notice that the facility is not yet implemented.

We're still working on this and plan to add some form of palette editing to a future version of the application.

Implementation

As you might imagine, the program described above is large — too large to list in the Analyst. You'll find a link to the code on the Web page for this issue of the Analyst. A word of caution: The program is functional but still a little raw. We'll update the Web version as the application evolves and, of course, we welcome suggestions as well as reports of problems and bugs.

References

1. "Graphics Corner — Custom Palettes", *Icon Analyst* 58, pp. 10-14.

2. "Anatomy of a Program — Numerical Carpets", *Icon Analyst* 45, pp. 1-10.

3. *Fractal Creations*, Timothy Wegner and Mark Peterson, Waite Group Press, 1991.

4. *Graphics Programming in Icon*, Ralph E. Griswold, Clinton L. Jeffery, and Gregg M. Townsend , Peer-to-Peer Communications, 1998, pp. 139-142.

5. "Weaving Drafts", *Icon Analyst* 53, pp. 1-4.

Message Drafting

In two previous articles [1,2], we described name drafting, a technique for designing weave drafts that uses a name or phrase (string) to produce threading and treadling sequences.

From our viewpoint, name drafting is just a device for producing numerical sequences that well could be produced by other means. We realize, however, that incorporating a name or phrase with personal meaning makes the resulting fabric of special significance.

One limitation of name drafting is that it's not possible to easily recover the string used from the drafting sequences. The encoding techniques used map several characters onto one shaft or treadle. This encoding table is an example:

AEIMQUY	shaft 1
BFJNRVZ	shaft 2
CGKOSW	shaft 3
DHLPTX	shaft 4

There is no inverse; a numerical sequence composed this way could stand for many strings. It might be possible to decrypt it, but it's not possible to decipher it. [3].

In this article we'll present an approach, called message drafting, in which each different character is mapped into a unique sequence.

The method is straightforward. Obvious coding techniques, such as mapping characters into their internal binary representation, could be used except for the fact that the weaving technique that produces attractive patterns requires an odd/even sequence [1].

Our approach is to build a list of odd/even coding patterns based on the number of shafts and treadles used (which must be even). We'll limit ourselves to a maximum of eight shafts or treadles to simplify the code. At the expense of complexity, it could be extended to any even number, but this kind of weaving usually is done on four shafts and treadles and rarely, if ever, on more than eight.

Here's a procedure to produce a code list for enciphering using *n* shafts with a character set of size *k*:

```
procedure mcodes(n, k)

   /k := *&cset

   if n % 2 ~= 0 then fail          # must be even
   if n > 8 then fail               # can't handle

   odds := ""
   evens := ""

   every odds ||:= (1 to n by 2)
   every evens ||:= (2 to n by 2)

   old_codes := [""]
   new_codes := []

   repeat {
      new_codes := []
      every code := !old_codes do {
        every put(new_codes, code || (!odds || !evens))
        if *new_codes >= k then  return new_codes
        }
      old_codes := new_codes
      }

end
```

All the codes must begin with a number of the same parity (we arbitrarily picked odd) so that when they are concatenated, the odd/even sequence is preserved.

To use these codes, it is necessary to map the characters of the message into indices for the list. If the entire character set is used, this is easy. It's only necessary to make the adjustment from 0-origin indexing to 1-origin indexing:

```
code := mcode(n, *&cset)
      …
seqcode := code[ord(c) + 1]
```

If another character set is used, the index of a character in the character set can be used. For example, if &letters is the character set,

```
code := mcode(n, *&letters)
```

```
  ...
seqcode := code[upto(c, &letters)]
```

Using this more general approach, a message-drafting procedure is:

```
procedure mdraft(text, charset, shafts)
   local seq, code

   code := mcodes(shafts, *charset) | fail

   seq := ""

   every seq ||:= code[upto(!text, charset)]

   return seq

end
```


The Icon Analyst

Ralph E. Griswold, Madge T. Griswold,
and Gregg M. Townsend
Editors

The Icon Analyst is published six times a year. A one-year subscription is $25 in the United States, Canada, and Mexico and $35 elsewhere. To subscribe, contact

Icon Project
Department of Computer Science
The University of Arizona
P.O. Box 210077
Tucson, Arizona 85721-0077
U.S.A.

voice: (520) 621-6613

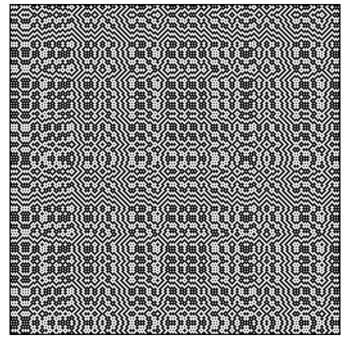
fax: (520) 621-4246

Electronic mail may be sent to:

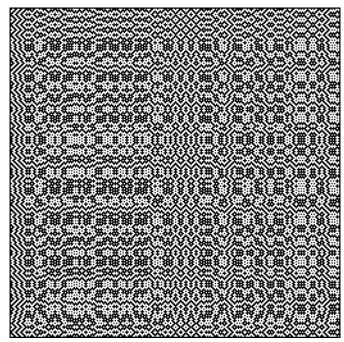
icon–analyst@cs.arizona.edu

THE UNIVERSITY OF
ARIZONA®
TUCSON ARIZONA
and
Bright Forest Publishers
Tucson Arizona

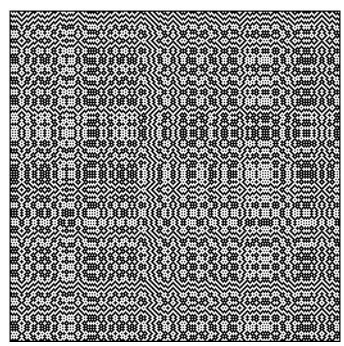
Figure 1 shows some drawdowns for four shafts, treadled as drawn in, with a /2/2 twill [4]. The sequences have been mirrored to add symmetries.



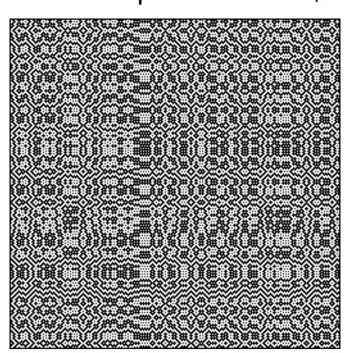
"Life is a beach."



"666 is the number of the beast."



"Fire the stupid #&%*~&^$%!"



"Check the square root of 13."

Figure 1. Drawdowns for Message Drafts

Colored threads can be used to make the weaves appear more visually complex.

It's important to note that long messages lead to large weaves and may, in fact, be a limiting factor.

Cryptographic Possibilities

Message drafting can be thought of as a way to encipher a message into a fabric. There are several aspects to such an enciphering:

- the method used to assign code sequences to characters
- the number of shafts and treadles used
- any modifications made, such as mirroring
- the tie-up
- thread colors

These can be considered to be keys.

For decrypting, fabric analysis techniques [5] can be used to determine the number of shafts and treadles used for a weave, as well as possible tie-ups and threading and treadling sequences (they are not unique).

While various methods can be used to make analysis more difficult and the results more ambiguous, it's probably best to encipher the message before doing message drafting if you wish to keep the message a secret.

Cryptograms in Textiles

Garments have been used to convey messages by their design and pattern since ancient times. Such messages often have been less than secret and sometimes quite blatant.

Incorporating secret messages into textiles and textile-related objects is not new. We can hardly forget Madame DeFarge noting testimony in trials during the French Revolution using stitches in her knitting, as portrayed in Charles Dickens' *A Tale of Two Cities*.

Recently we were reminded of another instance, this one in quilting. Before and during the Civil War, quilts were created that contained stops on the Underground Railroad encoded in their design.

If this idea caught on, it would lend a new meaning to *dress code*.

References

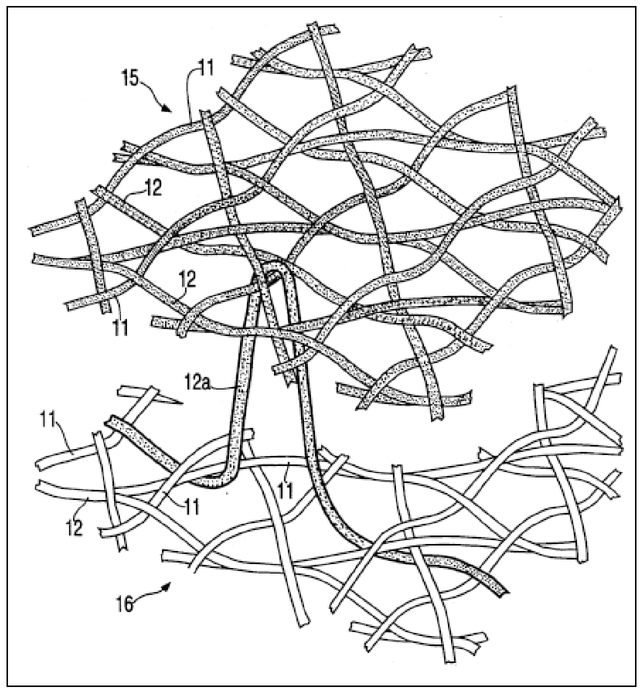
1. "Name Drafting", Icon Analyst 57, pp. 11-14.

2. "Name Drafting Revealed", Icon Analyst 58, pp. 15-16.

3. "Classical Cryptography", Icon Analyst 59, pp. 7-9.

4. "Twills", Icon Analyst 58, pp. 1-2.

5. *From Drawdown to Draft — A Programmer's View*, Ralph E. Griswold, http://www.cs.arizona.edu/patterns/weaving/FabricAnalysis.pdf.



What's Coming Up

Simplicity does not precede complexity, but follows it.

— Alan Perlis

Our plans for the next Analyst include articles on polygram substitution ciphers, continued fractions for quadratic irrationals, and adaptive name drafting.

We didn't have enough room in this issue for the planned article on derived tie-ups, so that's on the agenda for the next issue also.

If we make enough progress on the program for designing color weaves, we'll include that.