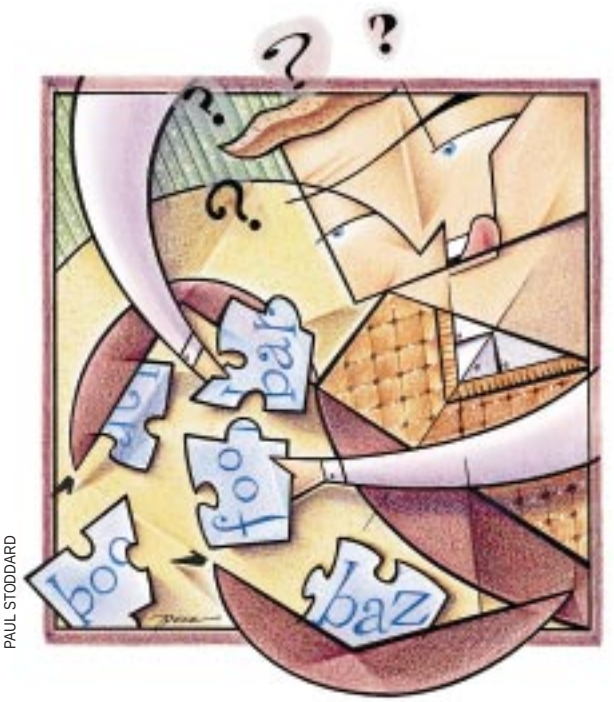
by Jeffreys Copeland and Haemer



PAUL STODDARD

Puzzle Posters

All work and no play makes Jeff a dull boy. – Jeffs

In the past two weeks we've gotten a lot of new machines in our office. Even after they were installed and, mostly, working, they had a few lingering problems. The most persistent problems involved email and, for a variety of reasons, we were stuck with the job of solving them; this is not we hasten to add because we know anything about email.

We quickly became frustrated and did what we often do when we can't solve a problem: We procrastinated by reading Usenet news.

Here's what happened next:

First, we perused `comp.lang.perl.misc`. Among the threads was a flame war initiated by Tom Christiansen, who posted an article stating his views that a) Perl is a glue language, b) any demand that a problem be solved entirely in Perl is misdirected and c) many posters' problems could be solved by installing an operating system from a source other than Microsoft Corp. You can imagine the sorts of responses this generated.

Christiansen is, among other things, the asbestos-coated author of major chunks of the Perl online documentation and coauthor of some key books on Perl. His Web site, `http://www.perl.com`, is a fine starting place for all things Perl-related. Whether you agree with his views or not, he's hard to trivialize.

Skipping past this, we found an interesting puzzle posted by Tim Bunce (see below).

```
A list of names in a specific order is given to a set of messengers in a remote land.
The messengers travel independently to a destination where they give the names
to you. The problem is that the messengers quite often, say 70%, miss out one or
more names and occasionally, say 10%, get the order wrong. Names are never added,
repeated or changed, only missed or reordered. The messengers always think
they've got it right. For example,

        Original list:      foo bar baz boo

        Messenger A says:   foo bar boo
        Messenger B says:   bar boo baz
        Messenger C says:   foo bar baz boo
        Messenger D says:   boo foo bar baz
        Messenger E says:   foo bar baz
        Messenger F says:   foo baz boo

The problem is to find the full list of names and the original order.
Tim
```

We considered a couple of simple-minded solutions, which didn't work. Then we tried a couple of other fixes to the mail problem, which also didn't work. So we went back and read some other responses to Tim's puzzle, which also didn't work. Then, we read one posted by Nathan Torkington, to wit:

```
Tim.Bunce@ig.co.uk (Tim Bunce) (Tim Bunce) writes:
> The problem is to find the full list of names and the
original order.

You INSTALL a FULL SET OF TOOLS, like THE LORD GOD
ALMIGHTY intended. REPENT, ye PRISONER of BILL! The DAY
of JUDGEMENT is AT PERL! Your MESSENGERS are obviously
just POOR substitutes for RELIABLE PIPE COMMUNICATION,
which you'd have if you had a REAL OPERATING SYSTEM and
not a SCURRILOUS PIECE OF TOOL-CHALLENGED COPROPHILIA!

Tom^WNat
:-)
```

For more on Nat, see the biographical material in *Perl Cookbook* by Tom Christiansen and Nathan Torkington, O'Reilly and Associates Inc., 1998, ISBN 1-56592-243-3.

Laughing, we realized that although Nat was teasing Tim, he was right. We, like the other posters, were trying to do a stand-alone solution, when the tool to solve the problem was already available: `tsort`. (If you've never used `tsort`, be patient; we'll talk more about it next month. For now, we'll say that `tsort` produces a single list from a collection of ordered pairs. Each input pair, *[X,Y]*, says *X* must come before *Y* in the final list. This is called *topological sorting*. See the man page on your system for more details. You don't have a man page for `tsort`? Go back and read Nat's message.)

We wrote a few lines of code to solve the problem and sent them off to Tim with copies to Nat and Tom for their amusement. Reasoning that when *X* precedes *Y* in the original list it will still precede *Y* in most garbled messages, we simply order each pair of words, then pipe the result to `tsort`. The next day Tim replied, confessing that Nat had been right for another reason–he really did need an all-Perl version because he needs to run the script on a Microsoft operating system, which has no `tsort`. "That should be easy to write," we said, foolishly, and told him we'd send him a Perl version of `tsort`. We spent some time tying ourselves in recursive knots, trying to write a simple topological sort. Next to this, the email problem looked easy, so we solved that instead.

After we gave up trying to write `tsort` ourselves, we searched the Web and found little more than man pages. (A Python implementation is provided at `http://www.pythonpros.com/arw/kjbuckets/tsort.py`. It requires a special Python extension module for graph operations, which we would have had to get and then translate.) Completely frustrated, we finally consulted our bookshelf. (Remember books?) Our copy of Knuth is missing, so we looked at Jon Bentley's *More Programming Pearls* (Addison-Wesley Publishing Co., 1988, ISBN 0-201-1189-0), and it had what we needed–note the spelling of "Pearls." His chapter on associative arrays reviews and implements topological sorting in less than four pages. That evening, we read the explanation and wrote a version in Perl. The next morning, we came to work bright and early, typed in the code and mailed it to Tim.

So before we launch into code, we think it's worth pointing out the unexpected moral to this story: using a tools approach sometimes keeps you from having to reimplement the wheel. But even when you *have* to reimplement the wheel, a tools approach can break problems into simple pieces.

The Solution

In the rest of this column, we'll talk about the first part of the solution. Figure 1 includes our tools-oriented solution.

Lines 1 and 3: Use the default `-w` flag to gener-

Figure 1. The Tools Approach

```
1   #!/usr/local/bin/perl -wl
2   # $Id: tbpuzzle,v 1.2 1998/06/26 14:02:11 jeff Exp $
3   use strict;

4   die "usage: $0 [filename]\n" if grep /^-./, @ARGV;

5   my $right_of;

6    # $right_of->{X}{Y} is
7    #  Y_right_of_X - X_right_of_Y
8    #
9    # A positive value means that Y is usually to the
10   #  right of X, #  and we'll infer that this was
11   #  the original order.

12  while (<>) {
13   my @message = split;
14   for (my $l = 0; $l < @message; $l++) {
15    for (my $r = $l+1; $r < @message; $r++) {
16     ++$right_of->{$message[$l]}{$message[$r]};
17     --$right_of->{$message[$r]}{$message[$l]};
18    }
19   }
20  }

21  open TSORT, "| tsort" or die "can't open '|tsort' \n";

22  foreach my $l (keys %$right_of) {
23   foreach my $r (keys %{$right_of->{$l}}) {
24    print TSORT "$l\t$r" if ($right_of->{$l}{$r} > 0);
25   }
26  }

27  close TSORT or die "can't close '|tsort' \n";

28  =head1 NAME

29  tbpuzzle - solve the Tim Bunce puzzle
```

```
30  =head1 SYNOPSIS

31    tbpuzzle [filename]

32  =head1 DESCRIPTION

33  =over 2

34  Solves this puzzle, posed by Tim Bunce, <Tim.Bunce@ig.co.uk>
35  in B<comp.lang.perl.misc>:

36  A list of names in a specific order is given to a set of
37  messengers in a remote land.

38  The messengers travel independently to a destination where
39  they give the names to you.

40  The problem is that the messengers tend to be forgetful:
41    They often miss out one or more names (but not all).
42    They occasionally get the order wrong.
43  (The only significance is that most of the messages will
44  have the right ordering of names.)

45  Names are never repeated or changed, only missed or reordered.
46  You don't know how long the original list of names was and
47  it's possible that all the correctly ordered messages are
48  missing one or more names. (The messengers always think
49  they've got it right.)

50  For example:

51    Original list:    foo bar baz boo

52    Messenger A says: foo bar boo     # bar missing
53    Messenger B says: bar boo baz     # reordered

54    Messenger C says: foo bar baz boo # complete
55    Messenger D says: boo foo bar baz # reordered
56    Messenger E says: bar             # foo baz boo missing
57    Messenger F says: foo baz boo     # bar missing
58    Messenger G says: foo bar baz     # boo missing
59    Messenger H says: baz boo         # foo bar missing
60    Messenger I says: bar boo         # foo baz missing

61  The problem is to try find the full list of names and the
62  original order as far as is possible.

63  =head1 OPTIONS AND ARGUMENTS

64  =over 8

65  =item B<filename>

66  Optional input file.
67  Input format is one list of white space-separated
68  names per line.

69  =back

70  =head1 AUTHOR

71    Jeffrey S. Haemer, <jsh@boulder.qms.com>

72  =head1 BUGS

73  Assumes that "reorderings" are only minor disruptions of the
74  original order, not, for example, inversions.

75  =head1 SEE ALSO

76  tsort(1)

77  =cut
```

ate warnings and the lint-like `strict` module to cut down on unintentional errors that even a program can see (line 1 also uses the `-l` flag to automate end-of-line processing). Line 2 is the Revision Control System (RCS) ID. We're not just the presidents of the "RCS Club for Men," we're members.

Lines 5 through 11 set up a pointer to a doubly indexed hash, the meaning of which is explained in the comment. (Yes, yes. We confess that we should say this is a reference to a hash of references to hashes and that `$right_of->{X}{Y}` is just syntactic sugar for `${${$right_of}{X}}{Y}`. If Perl can have syntactic sugar, so can English. (Historical exercise: Who invented the term "syntactic sugar"?)

Lines 12 through 20 read in one line at a time and then use this doubly indexed hash to keep track of how often each element occurs before each of the other elements. Every time we find *X* to the right of *Y*, we increment `$right_of->{X}{Y}`, and every time we find *X* to the left of *Y*, we decrement it. Note that `$right_of->{X}{Y}` will be positive if and only if *X* is found to the right of *Y*, more often than not. This is really quite a robust statistic. It will work so long as the "frequent" deletions leave each pair of nodes represented somewhere in the input data set and the "occasional" rearrangements don't swap the order of any pair of nodes more than half the time.

Lines 21 through 26 pipe the results to `tsort`, which uses the individual, pairwise orderings to come up with an overall order.

Lines 28 through 76 are the man page in "plain old documentation," or pod, format. This is ignored by Perl, but can be processed by tools from the standard distribution into various formats from Web pages to UNIX man pages. Pod-style documentation lets us keep the documentation and the code in the same file, so they're less likely to get out of sync. Note that we've deliberately put an updated form of the original problem statement into the file, so that someone using the program doesn't need to read this column to understand the problem being solved.

Next month, we'll consider `tsort` and discuss why UNIX has one. Until then, happy trails.

Jeffrey Copeland *(copeland@alumni.caltech.edu) lives in Boulder, CO, and works at Softway Systems Inc. on UNIX internationalization. He spends his spare time rearing children, raising cats and being a thorn in the side of his local school board.*

Jeffrey S. Haemer *(jsh@usenix.org) works at QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work.html`.