4. Versum Sequence Mergers

As we showed in the last issue of the Analyst, a simple observation and a relatively simple calculation allows the determination of all equivalent *n*-digit versum sequence seeds. The result is a dramatic reduction in the amount of computation and storage space that is needed to study versum sequences.

Another potential source of savings is in versum sequences that start out differently but merge to a common term. For example, the sequence for the seed 1 is

1: {2, 4, 8, 16, 77, 154, 605, 1111, 2222, … }

It's obvious that the seed that’s a term in this sequence immediately merges to it. For example, the sequence for 2 ,

2: { 4, 8, 16, 77, 154, 605, 1111, 2222, … }

is just the trailing part of the sequence for the seed 1. Of course, the sequence for 1 merges to the sequence for 2 after one term. To keep track of mergers, we’ll deal with seeds in numerical order and merge the sequence for seed 2 to the sequence for seed 1, rather than the other way around.

Sequences also merge to sequences with smaller seeds after initial terms that do not merge to other sequences. For example, the sequence for 104 is

104: {505, 1010, 1111, … }

Although 505 and 1010 are not terms in sequences with smaller seeds, 1111 is the eighth term in the sequence for seed 1. Thus 1 and 104 have sequences with different “heads” but a common “tail”.

In order to perform computations on versum sequences, it’s necessary to have the heads for all primary seeds, but the tails only for those sequences that do not merge to others. We’ll call the seeds for sequences that do not merge to others *base seeds*. It’s only the sequences for base seeds, then, that need to be carried out to the maximum number of terms needed.

You might wonder if all sequences eventually merge to a single base sequence. Empirical evidence strongly suggest that this is not the case, but it’s unlikely that such a conjecture could be proved or disproved. We’ll be satisfied in referring to base sequences as those that do not merge to others after a relatively large number of terms (like 500). We’ll have more on this later.

Several questions concerning merging versum sequences come to mind:

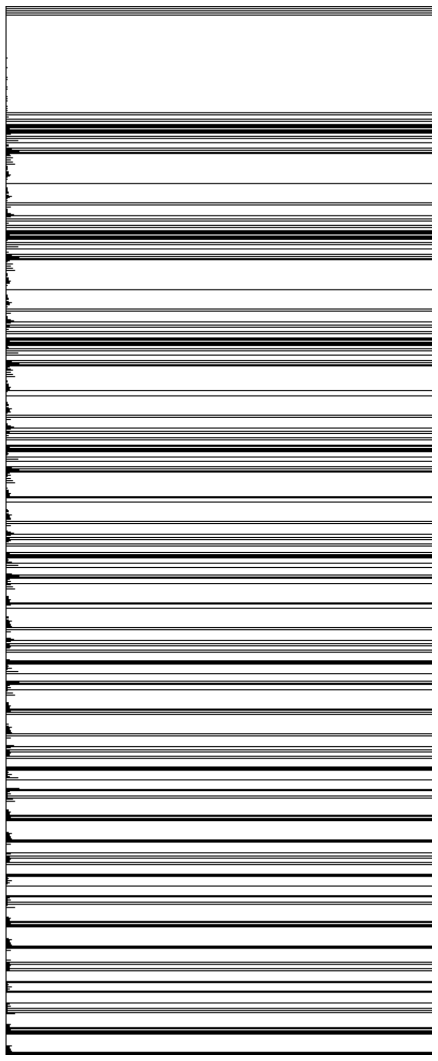
• How common are mergers?

• More specifically, what percentage of *n*-digit primary sequences are base sequences?

• How far out do mergers occur?

• How much space for storing versum sequences can be saved by storing only the heads of non-base sequences?

• How should we organize the data if we store only the heads of non-base sequences?



Versum Mergers for Seeds 1 Through

It doesn’t take much exploration to discover that mergers are quite common and that they usually occur after only a few terms. The image above shows how far sequences go before merger for seeds 1 through 999. Note that this image shows versum mergers for all seeds, many of which are equivalent.

It's clear from the amount of white space in this image (that is, terms after mergers), that there's a lot to gain in using merger information.

The information needed to represent non-base sequences is easily encoded, as in

104: {505, 1010, 1:8 }

where 1:8 is the terminating merger term.

With this in mind, we can create versum sequences, computing all terms for base sequences and only the heads for sequences that merge to others.

The approach we took to computing versum sequences with mergers was to create versum sequences for primary seeds in numerical order, keeping versum terms in a table. Then, when a new term is computed, if it's in the table, the sequence for the corresponding seed merges to a previously computed seed. By taking seeds in numerical order, we assured, as mentioned earlier, that the base seeds are the smallest members of their equivalence classes.

```
link options
link pvseeds

procedure main(args)
   local opts, n, limit, merge_tbl, i, j, k, count, output, name
   local tlist, merge_file, merge

   opts := options(args, "n+l+")

   n := \opts["n"] | 3                          # number of digits; small default
   limit := \opts["l"] | 30                     # maximum number of terms

   merge_file := open("vsq.mrg", "w") |         # non–base sequences go here
      stop("*** cannot open vsq.mrg")

   merge_tbl := table()                         # mergers for terms

   every i := pvseeds(1 to n, 1) do {           # primary seeds through n digits
      j := i
      tlist := []                               # list of terms
      output := &null                           # no output file yet
      every count := 1 to limit do {
         j +:= reverse(j)                       # next term
         if merge := \merge_tbl[j] then {       # term is already in table
            put(tlist, merge)                   # save merge information
            output := merge_file                # file for merger information
            break                               # terminate loop for this seed
            }
         else {                                 # term not in table
            put(tlist, j)                       # add term
            merge_tbl[j] := i || ":" || count   # merger information
            }
         }
      if /output then {                         # no file; new base sequence
         output := open(i || ".vsq", "w") |
            stop("*** cannot open ", i, ".vsq")
         every write(output, !tlist)
         close(output)
         }
      else {                                    # identify the seed
         write(output, i, "=")
         every write(output, !tlist)
         }
      }

end
```

Program to Create Versum Sequences with Merger Information

Before going on, we need to deal with another problem. Although the number of *n*-digit primary seeds is much smaller than the number of all *n*-digit seeds, as *n* gets large, keeping the sequences for all primary seeds in separate files becomes a significant problem for even modest values of *n*. As shown in the article on equivalent versum sequences, the number of primary seeds for *n*=6 is 6,498, for *n*=7, 64,980, and for *n*=8, 123,426.

Some operating systems handle a large number of files better than others, but few can do much with a large number of files in the same directory. All kinds of things go wrong. For example, if the sequences for all 6-digit primary seeds are kept as separate files with the suffix .vsq in one directory, on a UNIX platform, attempting to list them by

```
ls *.vsq
```

is likely to produce the message

```
Arguments too long.
```

You can imagine various things to do about such problems, such as organizing files in subdirectories, combining several sequences in one file, and so on. All of these lead to complexities and maintenance problems. Without taking a Draconian approach (like recomputing versum sequences as needed), the sheer number of sequences becomes the limiting factor as *n* becomes larger.

Dealing with mergers provides an opportunity to reduce the number of files needed for versum sequence information. Since the number of base seeds appears to be small compared to the number of primary seeds, and the number of terms in the heads of non-base sequences appears to be small on average, it seems reasonable to store the sequences for base seeds in separate files and put all the information for non-base seeds in a single file. It remains to be shown that this approach is practical — for example, that the file of non-base information is not monstrously large. It seems worth trying, in any event.

The program on the preceding page uses this approach. The file containing merger information for non-base sequences is named vsq.mrg. The terms in a sequence are kept in a list until it is known whether the sequence is a base sequence or one that merges to one. In the first case, a new file is created and the terms written to it. In the second case, the seed followed by an identifying mark and its terms are appended to vsq.mrg. Some portions of vsq.mrg are shown in the right

```
vsq.mrg
2=
1:2
4=
1:3
6=
3:2
8=
1:4
10=
5:2
11=
5:3
12=
3:3
13=
5:4
14=
7:2
15=
3:4
16=
1:5
17=
5:5
18=
9:2
19=
7:3
29=
7:4
39=
3:5
49=
143
7:6
59=
1:6
 …
409=
1313
1:10
509=
106:2
609=
1515
102:5
709=
100:5
809=
1717
1:11
909=
108:2
110=
7:4
115=
626
1252
3773
5:9
219=
1131
2442
3:9
719=
1636
7997
15994
65945
120901
229922
459844
908798
1806607
7:15
 …
```

It's worth noting that the technique we've used creates multiple merger links, as in

104: {505, 1010, 1111, 1:8}

109: {104:2}

Computing versum sequences and merger information in this way is not without its problems. It's necessary to create sequence information up to the chosen limit for all primary seeds up to the value of *n* chosen, all in one run. If *n* is even moderately large, like 6, and a few hundred terms are needed, the amount of memory to store all terms in all base sequences is *very* large.

Using an DEC Alpha workstation with 96 MB of RAM, we've managed to push *n* to 8 (on an Alpha with "only" 64 MB, swapping brought the machine to its knees, and it's unlikely that the process for *n*=9 would have completed before the machine crashed. Our patience was considerably more limited).

Having done the computation through *n*=8, we have enough information about the number of base sequences to be interesting:

n	*primary seeds*	*base seeds*	*ratio*
1	9	5	0.55556
2	18	0	0.00000
3	180	41	0.22777
4	342	16	0.04938
5	3420	464	0.13567
6	6498	220	0.03385
7	64980	4953	0.07622
8	123426	3061	0.02480
sum	198873	8760	0.04404

Remember that the number of base sequences given here is conjectural. It's certainly possible that some might merge if taken to more terms. However, the evidence strongly suggests otherwise; for *n* through 8 and 500 terms, the maximum number of steps to a merger is only 24 and more than 93% of all mergers occur within 3 steps.

From the information above, it's obvious that much less data is needed for storing only base sequences and merger information. For *n*=1 though 8, it takes "only" about 95 MB of disk space to store all the information; about 5 MB for vsq.mrg and 90 MB for the base sequences. From the ratios above, you can estimate the amount of space that would be required for storing sequences for all primary seeds.

Incidentally, there are suggestions of patterns of digits in the seeds for base sequences. There are too many of them to show for all but the most modest values of *n*. Here are the ones for *n*=3:

100	112	118	133	184	399	739	879	999
102	113	119	135	186	459	759	919	
106	114	122	137	196	539	779	939	
108	116	124	138	199	659	799	959	
111	117	128	166	359	679	859	979	

There are fewer for *n*=4:

1000	1011	1022	1037	1057	1068	1088	6999

1006 1013 1033 1046 1066 1077 4999 8999

There are more striking patterns for larger values of n, but there are too many to show here. Here's a curiosity for you to ponder, however: At least for n=1 through 8, all base seeds that start with a digit greater than 1 end in the digit 9.

Here's a proof by contradiction:

Suppose $a\underline{x}b$ is a base seed, where

a is a digit > 1

$\underline{x}$ is some sequence of digits

b is a digit < 9

Then let

$a' = a - 1$ (still a single digit > 0)

$b' = b + 1$ (still a single digit ≤ 9)

Because

$a' + b' = (a - 1) + (b + 1) = a + b$

then $rs(a\underline{x}b) = rs(a'\underline{x}b')$, where $rs(i) = i + reverse(i)$; that is, $a\underline{x}b$ and $a'\underline{x}b'$ merge.

But, by definition, a base seed is the smallest seed whose sequence does not merge to another sequence. Since $a\underline{x}b$ is larger than $a'\underline{x}b'$, $a\underline{x}b$ cannot be a base seed if $a > 1$ and $b < 9$.

Treating base sequences the way we have has complicated one aspect of using versum sequences. The procedure vsterm(i), which generates the terms in the sequence for seed i, now must deal with vsq.mrg and the way information in it is stored.

As with all the things we're considering, there are various approaches. We chose to have vsterm() read vsq.mrg the first time it is called, putting the information it contains in a table. We chose to have table keyed by non-base seeds whose corresponding values are lists of the terms in the head of the sequence, ending with the merger term.

The code is a bit lengthy and complicated, but it's not that difficult to write:

```
link vprimary

procedure vsterm(i)
   local term, merge_file, tlist, j, k, line
   static input, merge_tbl

   initial {
      merge_tbl := table()

      terms := 0

      merge_file := open("vsq.mrg") | {
         write(&errout, "*** cannot open vsq.mrg")
         fail
         }

      while line := read(merge_file) do {
         line ? {
            j := integer(tab(upto('=')))
            merge_tbl[j] := tlist := []
            while term := read(merge_file) do {
               if term := integer(term) then put(tlist, term)
               else {
                  put(tlist, term)
                  break
                  }
               }
            }
         }
      close(merge_file)
      write(&errout, "\ninitialization done")
      }

   close(\input)

   k := 0
   i := vprimary(i)

   if tlist := \merge_tbl[i] then {
      k := 1
      repeat {
         term := tlist[k]
         if term := integer(term) then suspend term
         else {
            term ? {
               i := integer(tab(upto(':')))
               move(1)
               k := integer(tab(0)) |
               if tlist := \merge_tbl[i] then next
               else break
               }
            }
         k +:= 1
         }
      }

   input := open(i || ".vsq") | {
      write(&errout, "*** cannot find sequence for ", i)
      fail
      }

   every 1 to k – 1 do read(input)

   while term := read(input) do
      suspend integer(term)

   close(input)

end
```

The functionality of vsterm() is the same as it was before, and programs that use it need not be changed for the new way of recording versum sequences.

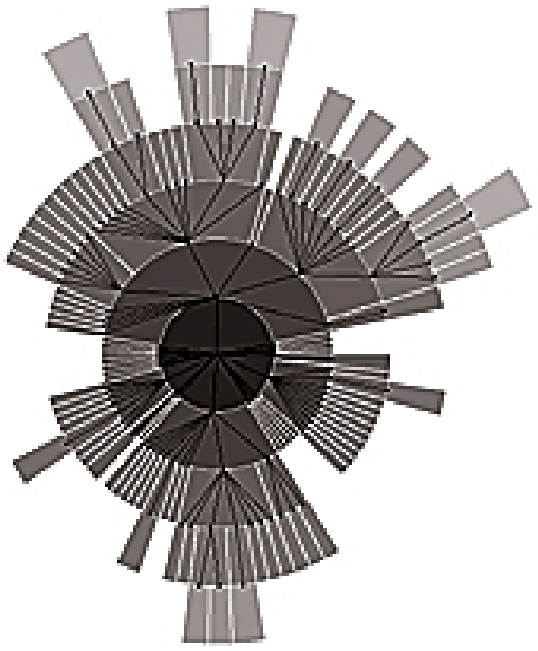
Merger Trees

One thing we began to wonder about when working with versum sequence mergers was the structure of mergers. Clearly, they're trees. And, of course, the nature of the trees depends on the particular way we've cast mergers and computed them.

Here's the program we used to produce string encodings of trees from vsq.mrg:

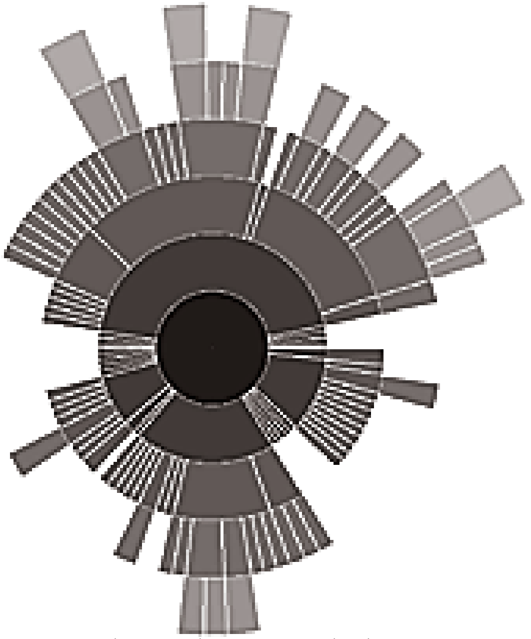
```
record arc(head, tail)

global seed_sets
global seed_names

procedure main(args)
  local n, seeds, arcs, i, x
  local merge_file, base, line, head, tail, node

  n := (0 < integer(args[1])) |
    stop("*** invalid command-line argument")

  merge_file := open("vsq.mrg") |
    stop("*** cannot open vsq.mrg")
  base := open("base." || n) |
    stop("*** cannot open base.", n)

  seeds := set()
  arcs := []
  seed_names := table()
  seed_sets := table()

  #  Process the merger information.

  every line := !merge_file do
    line ? {
      if head := integer(tab(upto('='))) then {
        if *head > n then break
        insert(seeds, head)
        }
      else if tail := integer(tab(upto(':'))) then {
        insert(seeds, tail)
        put(arcs, arc(head, tail))
        }
      }

  #  Create a set for each seed and build
  #  cross reference.

  every name := !seeds do {
    node := set()                   # create new node
    seed_sets[name] := node         # name to node
    seed_names[node] := name        # node to name
    }

  #  Insert the arcs.

  every x := !arcs do
    insert(\seed_sets[x.tail], \seed_sets[x.head])

  #  Output trees for every base name.

  every i := integer(!base) do
    write(tree(\seed_sets[i]))

end

#  Construct strings for merge trees.

procedure tree(node)
  local children, node_names

  children := ""
  node_names := []
  every put(node_names, seed_names[!node])
  every children ||:=
     tree(seed_sets[!sort(node_names)])

  return seed_names[node] || "[" || children || "]"

end
```

These trees gave us a chance to try out a visualization tool that is designed to provide a variety of ways of viewing trees. Here's a visualization of the merger tree leading to the base seed 9 for *n*=1 through 6.



This visualization uses rings to allow "wide" trees to be represented in an understandable way. Labels are omitted to allow the structure to be understood more easily.

The conventional tree diagram superimposed on the rings can be omitted, giving a simpler overall view of the tree:



We can easily see that merge links are at most 6 levels deep, but the tree has considerable breadth. The merger tree for seed 9 is larger than for most base seeds, but its shape is similar to that of most others.

We don't know what the nature of merger trees might say about versum sequences, but we find them interesting, nonetheless.

Acknowledgment

The tree visualization program was written as an honors project by Michael Shipman, a Computer Science senior. You've seen his name before; his project in the recent graphics programming course was one of the best and was featured in a recent *Icon Newsletter*.

Next Time

The observations and techniques described in the last two articles on versum sequences have made it possible to study their properties for much larger values of *n* than otherwise would have been possible.

Now it's time to use this capability to explore at greater length the original motivation for the study of versum sequences: palindromes. We'll tell you in advance that we haven't cracked the "big question": whether all versum sequences contain palindromes. But we do have some interesting results about the location of palindromes in versum sequences and the nature of their structure.

Ralph E. Griswold
Department of Computer Science
The University of Arizona

December 1995
Revised September 30, 1999