



Working with Bit Vectors

brian d foy
brian.d.foy@gmail.com

Perl is a high level language, so I don't have to play with bits and bytes to get my job done. The trade-off, however, is that I have to let Perl manage how it stores everything. What if I want to control that?

Memory Bigfoot

Perl trades memory for speed, meaning that it will gladly gobble up memory, waste it even some might say, so it can run faster. By keeping more stuff in memory, it can easily look them up. Most of the time that's a good thing. I can easily create arrays and hashes, manipulate them, and so on. I don't have to think about it.

Perl containers (or aggregates, which are terms for things that can have more than one item) group scalars. Scalars can be of any size up to available memory, but no matter their size, they have a bit of overhead. Even a scalar variable with no value takes up some space. The `Devel::Size` module can tell me how much, although the actual numbers may vary on your platform and version of Perl (mostly since the size of various low-level data types, such as an int, depend on the architecture).

```
use Devel::Size qw(size);

my $scalar;

print "Size of scalar is " .
      size( $scalar ) . " bytes\n";
```

On my Powerbook G4 running Perl 5.8.4, this scalar takes up 12 bytes, and it doesn't even have a value.

```
Size of scalar is 12 bytes.
```

I could use `Devel::Peek` to see some of this.

```
use Devel::Peek;

my $scalar;

print Dump( $scalar );
```

The output shows me the Perl has already set up some infrastructure to handle the scalar value.

```
SV = NULL(0x0) at 0x1807058
  REFCNT = 1
  FLAGS = (PADBUSY,PADMY)
```

Even with nothing in it, the scalar variable has a reference count and the scalar flags. Now, imagine an array of several hundred or thousand scalar values, each with their own scalar overhead.

Doing It Myself

I don't need to use Perl's arrays to store my data. If I have enough data, and another way to store it and then access it, I can save a lot of memory. If that doesn't impact the runtime too much, I should use that. Perl more than one way to do this, of course. I'm going to spend most of my article showing bit vectors, but along the way I'll show a couple of related techniques.

First, the easiest thing I can do use a long string where each character (or other number of characters) represents an element. I'll pretend that I'm working with DNA (the biological sort, although you should probably use BioPerl for this sort of thing), and I'll use the letters T, A, C, and G to represent the base pairs that make up the DNA strand. Instead of storing the sequence as an array of scalars each holding one character (or even objects representing that base), I store them as sequential characters in a single string where I only get the scalar overhead once.

```
my $strand = 'TGACTTTAGCATGACAGATACAGGTACA';
```

I can then access the string with `substr()`, which I give a starting position and a length.

```
my $codon = substr( $strand, 3, 3 );
```

I can even change values since I can use `substr()` as an lvalue.

```
substr( $strand, 2, 3 ) = 'GAC';
```