Magic Numbers and Extensions

Tim Clem

Spring 2007

University Honors in Mathematics
Advisor: Dan Kalman

Many ideas in mathematics surely began with a simple observation. The research in this paper is no different. Inspired by a peculiar property of numbers on calculator keypads, the concept of "magic" numbers was formed. Further investigation of "magicity" yielded the concept of a magic vector. These latter constructs encompass magic numbers but are far more generalized. The ultimate goal of this research is to categorize and predict these vectors.

1 The Beginning: Magic Numbers

We will begin with a discussion of magic numbers, their properties, and various approaches to investigating them.

1.1 Repunits and Other Notation

First, we will briefly introduce the concept of repunits and describe some notation used throughout the paper. As the name suggests, a *repunit* of length n, or $\mathbf{U}_n$, is the n-digit number formed by repeating the unit (i.e., 1) n times. For example, $\mathbf{U}_4 = 1111$.

Mathematically speaking,

$$\mathbf{U}_n = \sum_{i=1}^{n} 10^{i-1} = \frac{10^n - 1}{9}$$

Of course, this concept can be extended to other bases. In a positive base b, we have the following analogous formula:

$$\mathbf{U}_n^{(b)} = \sum_{i=1}^{n} b^{i-1} = \frac{b^n - 1}{b - 1}$$

For this paper, $\mathbf{U}_n$ (with no superscript) refers to a repunit in base 10.

The following is an important result that will be used throughout the paper.

Theorem 1. *Let u and v be positive, non-zero integers such that $u \mid v$. Then, for any positive base b, $\boldsymbol{U}_u^{(b)} \mid \boldsymbol{U}_v^{(b)}$.*

Proof. Let $k = \frac{v}{u} - 1$. Consider the base-b integer below.

$$\begin{aligned} R &= 1\overbrace{\overbrace{0\ldots01}^{u \text{ digits}}\overbrace{0\ldots01}\cdots\overbrace{0\ldots01}}^{k \text{ groups}} \\ &= 1 + b^u + b^{2u} + \cdots + b^{ku}. \end{aligned}$$

Multiplying R by $\mathbf{U}_u^{(b)}$ yields

$$\mathbf{U}_u^{(b)} R = \mathbf{U}_u^{(b)} + \mathbf{U}_u^{(b)} b^u + \mathbf{U}_u^{(b)} b^{2u} + \cdots + \mathbf{U}_u^{(b)} b^{ku}$$

Note that this product has $u + uk = u + u(\frac{v}{u} - 1) = v$ digits, all of which are the unit. That is, $\mathbf{U}_u^{(b)} R$ is exactly $\mathbf{U}_v^{(b)}$. Thus $\mathbf{U}_u^{(b)} \mid \mathbf{U}_v^{(b)}$. ■

For convenience in this paper, we will often write a string of integers in the form $d_1 d_2 \cdots d_k$. This refers to the place-value representation of a k-digit number in the appropriate base, where the d_i are the individual digits. It is assumed when this form is used, then, that d_i is a valid digit in base b. That is, for all i, $0 \le d_i \le b - 1$. This notation can be expressed as the sum below.

$$d_1 d_2 \cdots d_k = \sum_{i=1}^{k} d_i b^{i-1}$$

1.2 Numeric Keypads and Magic

The keypad on any standard calculator or computer keyboard was the ultimate inspiration for the idea of magic numbers. To wit, any "circle" of digits on a numeric keypad is divisible by the repunit half its length. We'll come back to the meaning of "circle" later, but for now let's investigate an example. In Figure 1, a circle is highlighted on a keypad. The path connecting the keys can be used to define a number one digit at a time. For instance, starting at 2 and proceeding around counterclockwise yields 2684.

As was stated, 2684 is divisible by $\mathbf{U}_2 = 11$. But it gets even more interesting. As the arrows in the figure suggest, it doesn't matter where on the circle we start or in which direction we traverse the four digits. For example, 4862, 4268 and 8624 are also divisible by 11. Indeed, every combination of cyclic shifts and reflections of the digits 2684 is divisible by 11. This property of a number makes it *magic*.

Let's flesh out these details more systematically. Let σ be a cyclic shift of the integers from 1 to $2k$ such that $\sigma(1) = 2k$ and $\sigma(i) = i - 1$ for $1 < i \le 2k$. Let ρ be a reflection

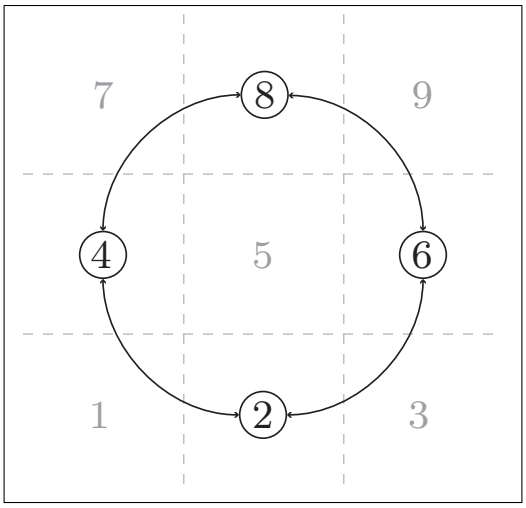

Figure 1: A magic number of length 4 on a standard numeric keypad.

of the same set such that $\rho(i) = 2k - i + 1$. Then define G, the dihedral group of degree $2k$ generated by σ and ρ.

Finally, we will define the action of $g \in G$ on an integer $N = \sum_{i=1}^{2k} d_i b^{i-1}$ as $g(N) = \sum_{i=1}^{2k} d_{g(i)} b^{i-1}$. With this framework in place, we arrive at the following definition.

Definition 1. Let N be a positive integer with in some positive base b. Then N is **magic** with respect to a divisor d if and only if $d \mid g(N)$ for all $g \in G$. In this paper, if no particular divisor is specified, *magic* means magic with respect to the repunit half the number's length, as in the example above.

1.3 Properties of Magic Numbers

The above definition is not particularly friendly. It provides no insight on how to determine if a given number is magic, other than testing all of its shifts and reflections. Luckily, we can derive some constraints on the divisor and a more workable test.

Theorem 2. Let $N = d_1 d_2 \cdots d_k$ be a positive integer in base $b > 0$. Let n be a positive integer such that $n \mid k$. If $\mathbf{U}_n^{(b)} \mid N$, then $\mathbf{U}_n^{(b)} \mid \sigma(N)$.

Proof. By the previous definition of σ, we know it cycles the leading digit of N to

the beginning of the number. That is,

$$\begin{aligned}\sigma(N) &= d_2d_3\cdots d_kd_1\\ &= bN - b^kd_1 + d_1\\ &= bN - d_1(b^k - 1)\\ &= bN - d_1(b-1)(b^{k-1} + b^{k-2} + b^{k-3} + \cdots + 1)\\ &= bN - d_1(b-1)\mathbf{U}_k^{(b)}\end{aligned}$$

Since $\mathbf{U}_n^{(b)} \mid N$, it must be that $\mathbf{U}_n^{(b)} \mid bN$.

By assumption, we know $n \mid k$. But by Theorem 1, this means $\mathbf{U}_n^{(b)} \mid \mathbf{U}_k^{(b)}$. So, $\mathbf{U}_n^{(b)} \mid d_1(b-1)\mathbf{U}_k^{(b)}$. Thus $\mathbf{U}_n^{(b)} \mid \sigma(N)$. ■

This building block allow us to simplify our magicity test.

Corollary 1. Let N be a positive integer in some positive base b. Then, N is magic with respect to $\mathbf{U}_n^{(b)}$ if and only if $\mathbf{U}_n^{(b)} \mid N$ and $\mathbf{U}_n^{(b)} \mid \rho(N)$.

Proof. Assume $\mathbf{U}_n^{(b)} \mid N$ and $\mathbf{U}_n^{(b)} \mid \rho(N)$. By Theorem 2, $\mathbf{U}_n^{(b)} \mid N$ implies that $\mathbf{U}_n^{(b)} \mid \sigma(N)$. Thus, if $\mathbf{U}_n^{(b)} \mid \rho(N)$, it follows that $\mathbf{U}_n^{(b)} \mid g(N)$ for all $g \in G$, since σ and ρ generate G and both preserve divisibility by $\mathbf{U}_n^{(b)}$. Hence N is magic.

The converse follows from the definition of magic. ■

Finally, we can place a limit on the divisors with respect to which a number can be magic.

Theorem 3. Let $N = d_1d_2\cdots d_k$, with not all $d_i = 0$, be a positive integer in some positive base b. If N is magic with respect to $\mathbf{U}_n^{(b)}$, then $n \mid k$.

Proof. By the definition of magic, the following hold.

$$N = d_1d_2\cdots d_k \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \tag{1}$$

$$\sigma(N) = d_2d_3\cdots d_kd_1 \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \tag{2}$$

Suppose $n \nmid k$. Then, by the division algorithm, there exist integers $\ell \geq 0$ and $0 < r < n$ such that $k = \ell n + r$.

Note that $\sigma(N)$ (**(2)** above) will still be magic if padded with $n - r$ zeroes, so that it has a length which is a multiple of n. That is,

$$\overbrace{00\cdots0}^{n-r}d_2d_3\cdots d_kd_1 \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

So the following shift is also divisible by $\mathbf{U}_n^{(b)}$:

$$d_1 \overbrace{00\cdots0}^{n-r} \overbrace{d_2 d_3 \cdots d_k}^{k-1} \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

Subtracting **(1)** from the previous equation yields

$$d_1 \overbrace{00\cdots0}^{n-r-1} (-d_1) \overbrace{00\cdots0}^{k-1} \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

Or, with shifting,

$$d_1 \overbrace{00\cdots0}^{n-r-1} (-d_1) \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

So, $d_1 b^{n-r} - d_1 = d_1(b^{n-r} - 1) \equiv 0 \pmod{\mathbf{U}_n^{(b)}}$. Note that $b^{n-r} - 1 = (b-1)\mathbf{U}_{n-r}^{(b)}$, and since $\mathbf{U}_n^{(b)} \nmid (b-1)$ and $\mathbf{U}_n^{(b)} \nmid \mathbf{U}_{n-r}^{(b)}$, it must be that $d_1 \equiv 0 \pmod{\mathbf{U}_n^{(b)}}$.

But since d_1 is a digit in base b, it must be the case that $0 \leq d_1 \leq b-1$. Therefore $d_1 = 0$.

Since we can apply this development to all shifts of N, we conclude $d_i = 0$ for all $1 \leq i \leq k$. This is a contradiction, so it must be that $n \mid k$. ■

We now know, for example, that a 10-digit number cannot be magic with respect to $\mathbf{U}_4$, since $4 \nmid 10$. Let's continue our exploration of magicity by returning to numeric keypads.

1.4 More on "Circles": The Diameter Property

It was stated earlier that this magic property holds for all "circles" on numeric keypads. What exactly is a circle in this sense? To clarify this idea, we can represent the process of entering digits on a keypad using simple images like those in Figure 2. In each of these diagrams, there are nodes on a number of the keys, and a path connecting these. Define a *traversal* of this path to mean following one node to the next, entering a digit for each key that is crossed. Then all the paths in Figure 2 have the following interesting property: traversals of them define numbers that are magic with respect to a particular repunit. In fact, in each of these paths, the number of nodes is even (say $2n$), and the number produced by a traversal is magic with respect to $\mathbf{U}_n$. Also note that by the definition of magic, each path can be traversed starting with any node and in either direction, and in each case the number produced is divisible by $\mathbf{U}_n$.

However, as Figure 2 shows, it's not exactly obvious what gives this magic property to a particular traversal. The cyclic nature of magic numbers lends a clue to finding the similarities in these patterns. If we evenly space the digits in a magic number around a circle, an interesting phenomenon becomes apparent. Namely, the sums of the digits opposite each other are all equal (see Figure 3). This leads to a definition.

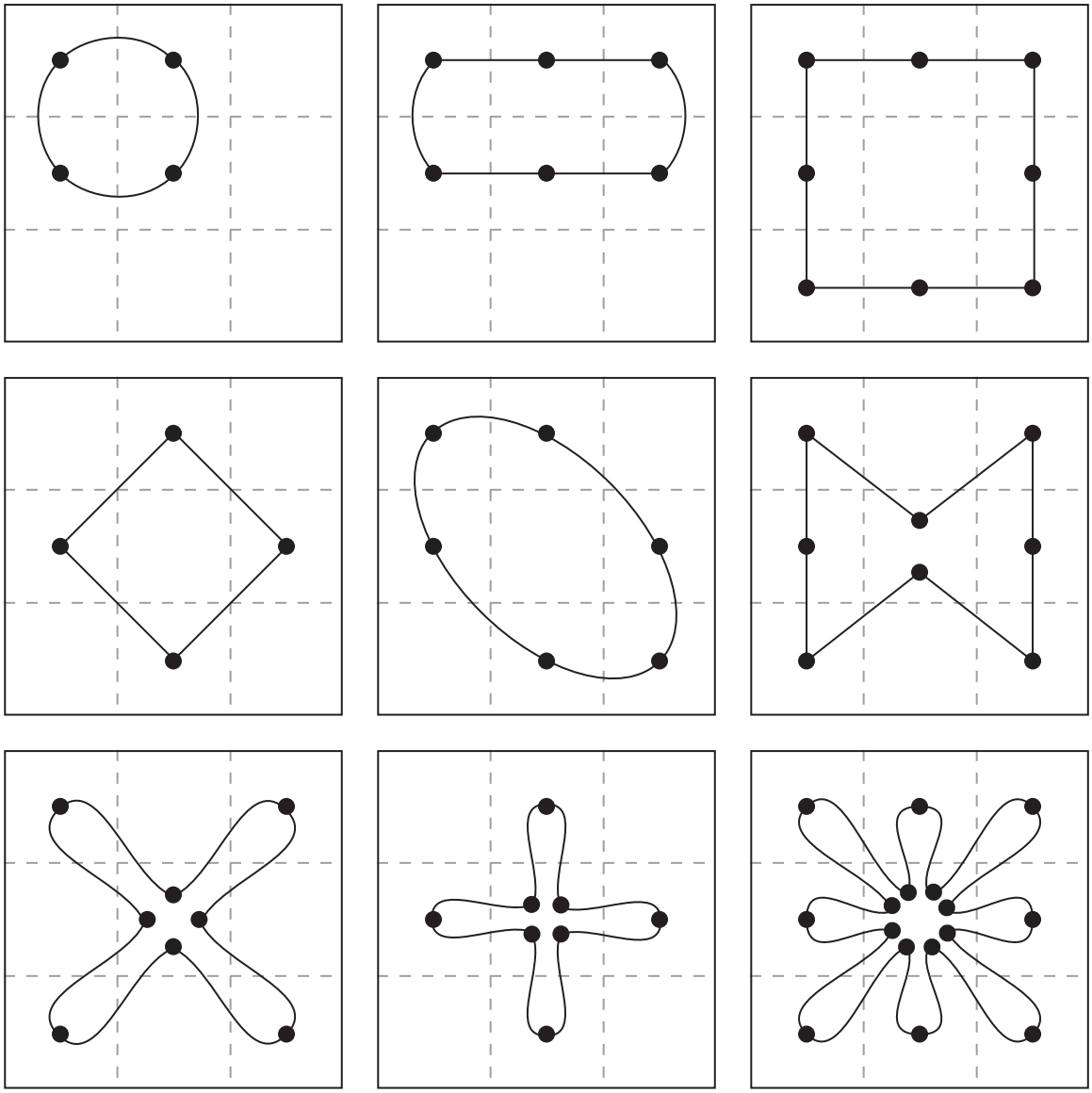

Figure 2: These traversals, among others, also yield magic numbers.

Definition 2. Let $N = d_1 d_2 \cdots d_{2k}$ for some k be a positive integer. Define $a_i = d_i + d_{k+i}$, where $1 \leq i \leq k$. If all the a_is are equal, then N has the **diameter property**. We will call the number $\sum_{i=1}^{k} a_i b^{i-1}$ the **diagonal sum** of the number N.

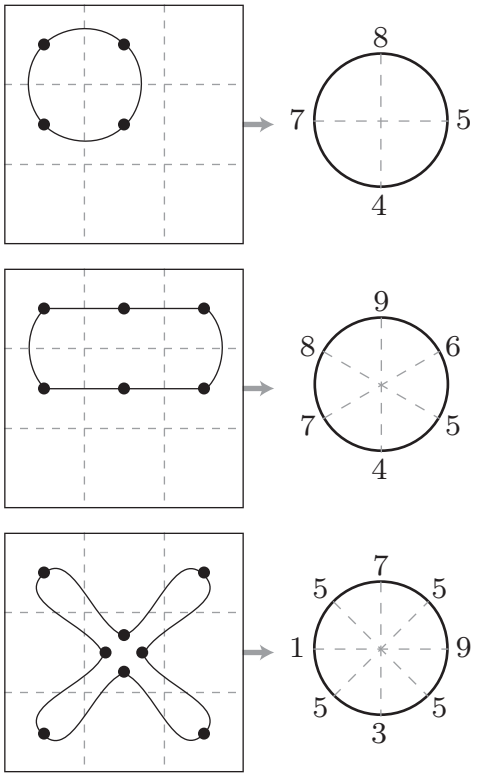

Figure 3: The diameter property illustrated for several magic numbers and their corresponding keypad configurations.

Is this property enough to ensure magicity? We can prove that it is, but we need to go through some intermediate steps. First, note the following identity relating powers of a base and repunits.

Lemma 1. Let b and n be positive, non-zero integers. Then $b^n \equiv 1 \pmod{\mathbf{U}_n^{(b)}}$.

Proof. From the previous discussion of repunits, we have

$$\mathbf{U}_n^{(b)} = \frac{b^n - 1}{b - 1}.$$

Multiplying both sides by $b - 1$ and adding 1 yields

$$\begin{aligned} b^n &= \mathbf{U}_n^{(b)}(b-1) + 1 \\ &\equiv 1 \pmod{\mathbf{U}_n^{(b)}} \end{aligned}$$

■

Theorem 4. Let $N = d_1 d_2 \cdots d_{2k}$ for some k be a positive integer in base $b > 0$. Then N is congruent to its diagonal sum modulo $\mathbf{U}_k^{(b)}$.

That is, first define $a_i = d_i + d_{k+i}$, where $1 \leq i \leq k$. Then,

$$N \equiv \sum_{j=1}^{k} a_j b^{j-1}.$$

Proof. Writing N as an explicit place-value sum, we have

$$N = \sum_{j=0}^{2k-1} d_{2k-j} b^j.$$

Separating this into two sums, we get

$$N = \sum_{j=0}^{k-1} d_{2k-j} b^j + \sum_{j=k}^{2k-1} d_{2k-j} b^j.$$

Shifting bounds and factoring,

$$\begin{aligned} N &= \sum_{j=0}^{k-1} d_{2k-j} b^j + \sum_{j=0}^{k-1} d_{k-j} b^{j+k} \\ &= \sum_{j=0}^{k-1} d_{2k-j} b^j + b^k \sum_{j=0}^{k-1} d_{k-j} b^j. \end{aligned}$$

By the lemma, $b^k \equiv 1 \pmod{\mathbf{U}_k^{(b)}}$. So,

$$\begin{aligned} N &\equiv \sum_{j=0}^{k-1} d_{2k-j} b^j + \sum_{j=0}^{k-1} d_{k-j} b^j \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv \sum_{j=0}^{k-1} (d_{k-j} + d_{2k-j}) b^j \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv \sum_{j=0}^{k-1} a_{k-j} b^j \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv \sum_{j=1}^{k} a_j b^{j-1} \pmod{\mathbf{U}_k^{(b)}} \end{aligned}$$

■

Corollary 2. If a number $N = d_1 d_2 \cdots d_{2k}$ exhibits the diameter property, it is magic.

Proof. As previously, let $a_i = d_i + d_{k+i}$, where $1 \leq i \leq k$. However, since N has the diameter property, all the a_is are equal to some constant, namely a_1.

The previous theorem shows that $N \equiv \sum_{j=1}^{k} a_j b^{j-1} \pmod{\mathbf{U}_k^{(b)}}$. But, in this case, this means

$$\begin{aligned} N &\equiv \sum_{j=1}^{k} a_1 b^{j-1} \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv a_1 \sum_{j=1}^{k} b^{j-1} \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv a_1 \mathbf{U}_k^{(b)} \pmod{\mathbf{U}_k^{(b)}} \\ &\equiv 0 \pmod{\mathbf{U}_k^{(b)}} \end{aligned}$$

Thus $\mathbf{U}_k^{(b)} \mid N$. The fact that $\mathbf{U}_k^{(b)} \mid \rho(N)$ follows from symmetry—it is clear that flipping the digits of N preserves the a_is. Hence N is magic by Corollary 1. ■

The question still remains, however: why do these traversals of a numeric keypad exhibit the diameter property? Consider the numbers on a keypad as the values of a function over the x-y plane (see Figure 4). From this perspective, the function defining the keypad is is $p(x, y) = x + 3(y - 1)$ for $x, y \in \{1, 2, 3\}$.

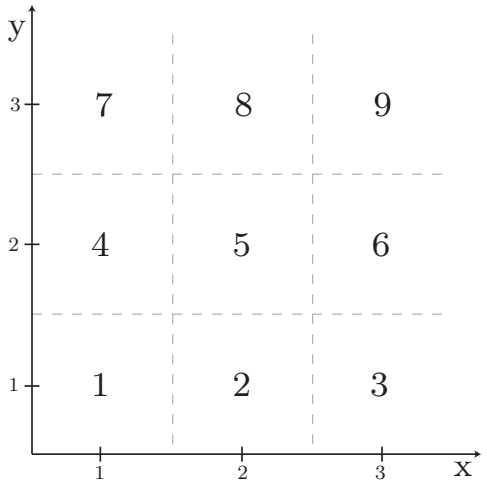

Figure 4: The numeric keypad considered as a function on the x-y plane.

Notably, this function is *linear*. Thus, its average value between two points is the

function evaluated at the midpoint. That is,

$$\frac{p(x,y)+p(u,v)}{2} = p\left(\frac{x+y}{2}, \frac{y+v}{2}\right), \text{ or}$$
$$p(x,y)+p(u,v) = 2p\left(\frac{x+y}{2}, \frac{y+v}{2}\right).$$

So, the sum of the values assumed by p at opposite ends of a diameter is twice the value at the midpoint of the diameter. Thus, if the diagonals of a figure all cross in a single point, the figure has the diameter property. A quick check of the traversals in Figure 2 show that this is the case for all those examples.

Now that we have a deeper understanding of the mechanism behind the diameter property, we can search for non-circles that still generate magic numbers. Figure 5 is one such traversal. By selecting the same number of alternating 1s and 2s as 5s and 6s, one generates a number with a diagonal sum equal to a string of 7s. For instance, both 1265 and 121656 are magic.

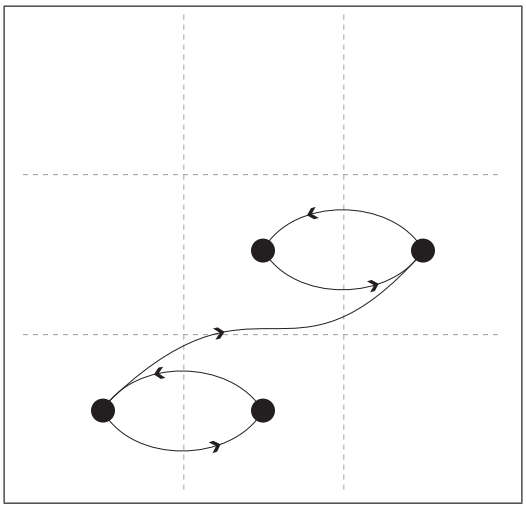

Figure 5: This decidedly non-circular traversal satisfies the diameter property.

2 Magic Vectors

2.1 Wrapping

Theorem 4 showed that a $2k$-digit number is divisible by $\mathbf{U}_k$ if and only if its diagonal sum is. This implies a new treatment of magic number with respect to circles: wrapping the number around twice (see Figure 6). That is, rather than arranging a

$2k$ number around $2k$ points on a circle, use only k, with 2 digits at each point. By summing the digits at each point, we construct the digits of the diagonal sum. Thus, if the sums at each point are equal, the number has the diameter property and is magic by Corollary 2. This is called the *wrapping property*.

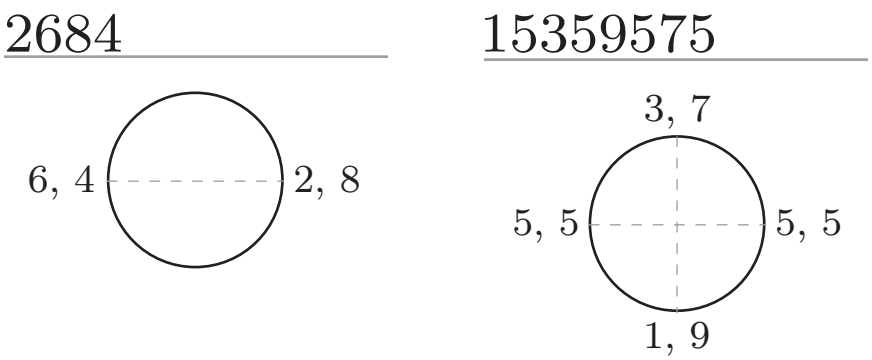

Figure 6: The wrapping property illustrated for two magic numbers.

But why focus only on diameters? In Figure 7, a 6-digit number is wrapped around a circle, and triangles are used to connect the digits rather than diameters. The sums of the numbers at the vertices of the two triangles are both 7, and 212135 is magic with respect to $\mathbf{U}_2 = 11$.

As the figure suggests, this "triangle property" is analogous to wrapping the number 3 times around a circle with two points. This idea generalizes to a sort of "polygon property." Namely, for a circle with with n points, we can create $\frac{n}{k}$ regular k-gons (assuming $k \mid n$) and add the digits at the vertices of each k-gon. This process, however, corresponds to increasing the number of wrappings around a circle. So, rather than dealing with polygons, we can concentrate on the simpler idea of multiple wrappings.

We have seen that the diameter property and diagonal sums (wrapping twice) deal with magicity in terms of the repunit half the length of the original number. In comparison, wrapping a number k times over n points deals with $\mathbf{U}_n$. In Figure 7, the six-digit number was wrapped 3 times over 2 points. In this case, the number turns out to be magic with respect to $\mathbf{U}_2$.

Why consider these wrappings? As we will see, they constitute a way to abstract and simplify magicity. Also, there is a more general form of Theorem 4 which applies to them.

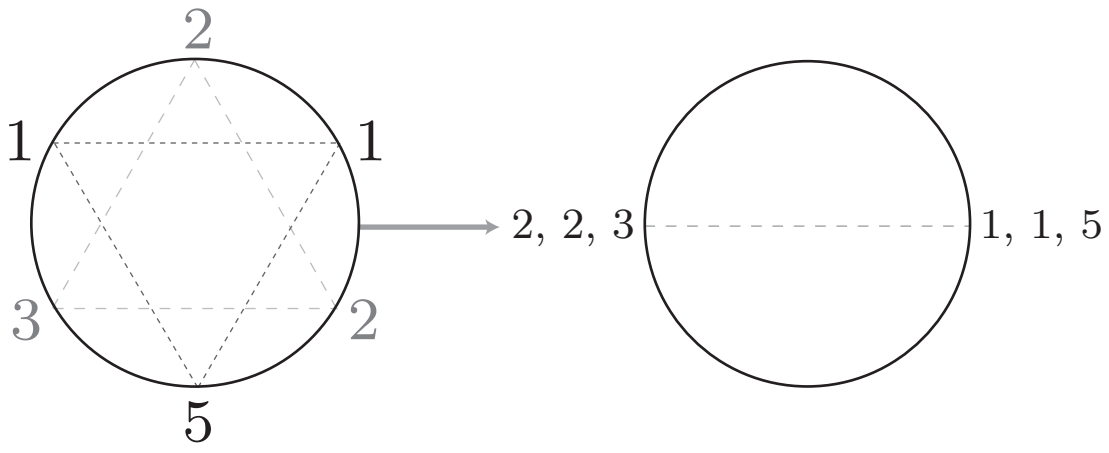

Figure 7: We can switch from diameters to triangles, which is equivalent to wrapping the number three times around a circle with two points.

Theorem 5. Let $N = d_1 d_2 \cdots d_k$ be a k-digit number in base $b > 0$. Let n be a positive integer such that $n \mid k$, and define $c = \frac{k}{n}$. Finally, for $1 \leq i \leq n$, define

$$a_i = \sum_{j=0}^{c-1} d_{i+nj}.$$

Then,

$$N \equiv \sum_{t=1}^{k} a_t b^{t-1} \pmod{\mathbf{U}_n^{(b)}}$$

The statement of this theorem is fairly inscrutable at first, but it's really just a generalization of Theorem 4 where N has been split into n pieces, rather than just 2. The a_is correspond to the sum of every n^{th} digit of N, or the result of sums on a circle using a regular n-gon, rather than diameters, to connect the points. So, if $n = 2$, this is exactly Theorem 4. For $n = 3$, we're constructing the "triangle sums" we saw in Figure 7. For greater values of n, the "polygon property" applies.

Proof. We begin by writing N as in Theorem 4.

$$N = \sum_{\alpha=0}^{k-1} d_{k-\alpha} b^{\alpha}.$$

Splitting this sum into c sums of n digits, we have

$$N = \sum_{\alpha=0}^{n-1} d_{k-\alpha} b^{\alpha} + \sum_{\alpha=n}^{2n-1} d_{k-\alpha} b^{\alpha} + \cdots + \sum_{\alpha=(c-1)n}^{nc-1} d_{k-\alpha} b^{\alpha}.$$

Shifting bounds and factoring yields

$$\begin{aligned} N &= \sum_{\alpha=0}^{n-1} d_{k-\alpha} b^{\alpha} + \sum_{\alpha=0}^{n-1} d_{k-\alpha-n} b^{\alpha+n} + \cdots + \sum_{\alpha=0}^{n-1} d_{k-\alpha-(c-1)n} b^{\alpha+(c-1)n} \\ &= \sum_{\alpha=0}^{n-1} d_{k-\alpha} b^{\alpha} + b^{n} \sum_{\alpha=0}^{n-1} d_{k-\alpha-n} b^{\alpha} + \cdots + b^{(c-1)n} \sum_{\alpha=0}^{n-1} d_{k-\alpha-(c-1)n} b^{\alpha}. \end{aligned}$$

Lemma 1 showed that $b^n \equiv 1 \pmod{\mathbf{U}_n^{(b)}}$. So, we can rewrite the powers of the base in this last expression. Modulo $\mathbf{U}_n^{(b)}$, they are all 1:

$$N \equiv \sum_{\alpha=0}^{n-1} d_{k-\alpha} b^{\alpha} + \sum_{\alpha=0}^{n-1} d_{k-\alpha-n} b^{\alpha} + \cdots + \sum_{\alpha=0}^{n-1} d_{k-\alpha-(c-1)n} b^{\alpha} \pmod{\mathbf{U}_n^{(b)}}.$$

Factoring and combining these sums, we have

$$N \equiv \sum_{\alpha=0}^{n-1} b^{\alpha} \left(d_{k-\alpha} + d_{k-\alpha-n} + \cdots + d_{k-\alpha-(c-1)n} \right) \pmod{\mathbf{U}_n^{(b)}}.$$

But note that the coefficient of b^{α} is simply $a_{k-\alpha}$. Thus we really have the following

$$\begin{aligned} N &\equiv \sum_{\alpha=0}^{k-1} a_{k-\alpha} b^{\alpha} \pmod{\mathbf{U}_n^{(b)}} \\ &\equiv \sum_{t=1}^{k} a_t b^{k-t} \pmod{\mathbf{U}_n^{(b)}}. \end{aligned}$$

■

Corollary 3. Using the same notation as the previous theorem, N is magic with respect to $\mathbf{U}_n^{(b)}$ if and only if

$$\sum_{t=1}^{k} a_t b^{t-1} \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \text{ and} \tag{3}$$

$$\sum_{t=1}^{k} a_t b^{k-t} \equiv 0 \pmod{\mathbf{U}_n^{(b)}}. \tag{4}$$

Proof. First, assume N is magic with respect to $\mathbf{U}_n^{(b)}$. Then, by the definition of magic, $N \equiv 0 \pmod{\mathbf{U}_n^{(b)}}$. But, by the previous theorem, $N \equiv \sum_{t=1}^{k} a_t b^{t-1} \pmod{\mathbf{U}_n^{(b)}}$. Hence **(3)** is satisfied. Now note that $\sum_{t=1}^{k} a_t b^{k-t}$ is the result of applying Theorem 5 to the reflection of N, or $\rho(N)$. But by the definition of magic, $\rho(N)$ is also equivalent to $0 \pmod{\mathbf{U}_n^{(b)}}$, satisfying condition **(4)**.

Now assume conditions **(3)** and **(4)**. By the previous theorem, $\sum_{t=1}^{k} a_t b^{t-1} \equiv N \pmod{\mathbf{U}_n^{(b)}}$. But since **(3)** is true, this means $N \equiv 0 \pmod{\mathbf{U}_n^{(b)}}$. Similarly, **(4)** and the previous theorem imply that $\rho(N) \equiv 0 \pmod{\mathbf{U}_n^{(b)}}$. Thus, by Corollary 1, N is magic with respect to $\mathbf{U}_n^{(b)}$. ■

2.2 Magic Vectors

With Corollary 3, we can stop focusing on magic numbers per se and move on to a more streamlined representation: *magic vectors*. Begin by wrapping the digits of an nk-digit number k times around n points on a circle. Then, add up the digits that fall at each point, and present the result as an n-element vector. See Figure 8 for an example.

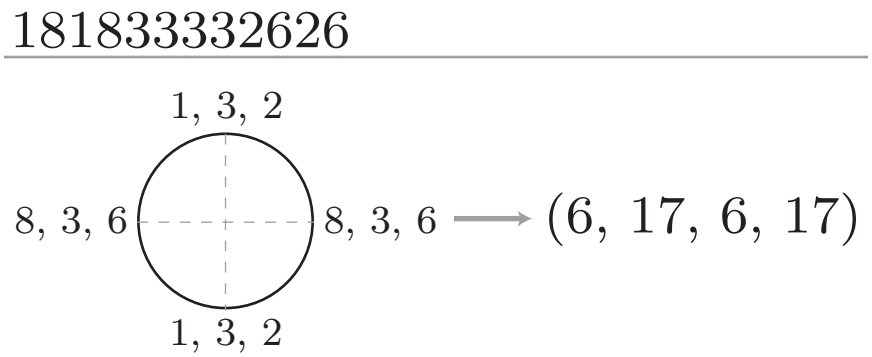

Figure 8: This 12-digit number is magic with respect to $\mathbf{U}_4$ and wraps to the magic vector $(6, 17, 6, 17)$. Note that there are 3 digits at each point, so the largest a component could be is $3 \cdot 9 = 27$.

This is exactly the same as a vector containing the a_i from Theorem 5. So, we can associate these vectors and the sums we saw in the previous theorems by a dot product. That is, $\sum_{i=1}^{n} a_i b^{i-1} = (a_1, a_2, \ldots, a_n) \cdot (b^{n-1}, b^{n-2}, \ldots, 1)$.

Thus, we define a *magic vector* as the n-element list of the a_is from a magic number. Translating the results of Corollary 3 to the language of vectors and dot products, we arrive at the following working definition.

Definition 3. Let $v = (v_1, v_2, \ldots, v_n)$ be an integer vector in some positive base b. Then v is a **magic vector** if and only if the following properties hold:

$$\sum_{i=1}^{n} v_i b^{i-1} = v \cdot (b^{n-1}, b^{n-2}, \ldots, b, 1) \equiv 0 \pmod{\mathbf{U}_n^{(b)}}, \text{ and}$$
$$\sum_{i=1}^{n} v_i b^{n-i} = v \cdot (1, b, \ldots, b^{n-2}, b^{n-1}) \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

If all v_i are such that $0 \leq v_i \leq (b-1)k$ for some positive integer k, then v is said to be magic in $[n, k]^{(b)}$.

Note that if a vector is magic in $[n, k]^{(b)}$, then it is also magic in $[n, \ell]^{(b)}$ for all $\ell \geq k$, since the k only serves as an upper bound for the vector's components.

For the purposes of this paper, when a base is not specified, $[n, k]$ refers to vectors in base 10.

2.3 Properties of Magic Vectors

All the magic numbers we saw in our earlier discussion of calculator keypads wrap to vectors of the form $(c, c, \ldots, c)$ for some constant integer c. But the number in Figure 8 generated $(6, 17, 6, 17)$ in $[4, 3]$. Let's check that this vector is magic by computing the dot products above.

$$\begin{aligned}
(6, 17, 6, 17) \cdot (1000, 100, 10, 1) &= 6000 + 1700 + 60 + 17 \\
&= 7777 \\
&= 7 * 1111 \\
&\equiv 0 \pmod{\mathbf{U}_4} \\
(6, 17, 6, 17) \cdot (1, 10, 100, 1000) &= 6 + 170 + 600 + 17000 \\
&= 17776 \\
&= 16 * 1111 \\
&\equiv 0 \pmod{\mathbf{U}_4}
\end{aligned}$$

So $(6, 17, 6, 17)$ really is a non-constant magic vector. What are these variant vectors, and where do they arise? First, let's investigate some basic properties of magic vectors.

One interpretation of these vectors, especially given the dot products with powers of the base, is to consider them as "digits" in a base-b number. We can't properly call

them digits because, in a typical setting, the digits of a base-b number fall between 0 and $b-1$. However, the elements of a magic vector in $[n,k]^{(b)}$ fall between 0 and $(b-1)k$. Luckily, a number of our previous results, namely Theorem 2 and its corollary, do not rely on the maximum size of a "digit". Thus, they hold by analogy for magic vectors. That is, if v is a magic vector in $[n,k]^{(b)}$, then all combinations of shifts and reflections of v are also magic. Similarly, we will continue to use G to represent the group generated by cyclic shifts (σ) and reflections (ρ) of v.

It is also worthy of note that the magicity of vectors is preserved under linear combinations. This result is proven in the following theorem.

Theorem 6. Let u and v be magic vectors of length n, and let c be an integer. Then the following are true:

1. cv is magic
2. $u+v$ is magic

Proof. These properties follow directly from the linearity of the dot product. If we let B be the vector of increasing powers of b, then the following follow from the definition of magic.

$$(cv)\cdot B = c(v\cdot B) \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \text{ and}$$
$$(u+b)\cdot B = u\cdot B + v\cdot B \equiv 0 \pmod{\mathbf{U}_n^{(b)}}.$$

The same follows if we let B be the decreasing powers of b. ■

2.4 Basic Magic Vectors

In this section we will discuss three types of common magic vectors, *constant vectors*, *alternating "11" vectors*, and *repunit multiple vectors*. Combined linearly, the first two can account for all the vectors we have seen thus far. However, they are by no means a comprehensive survey of all magic vectors.

Theorem 7. Any vector whose elements are all 1 is magic.

Proof. Let v be a vector of length n with all elements equal to 1. Then, for any

positive base b, consider the dot products

$$\begin{aligned} v \cdot (b^{n-1}, b^{n-2}, \ldots, 1) &= \sum_{i=1}^{n-1} b^{n-i} \\ &= \mathbf{U}_n^{(b)} \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \\ v \cdot (1, b, \ldots, b^{n-1}) &= \sum_{i=1}^{n-1} b^{n-1} \\ &= \mathbf{U}_n^{(b)} \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \end{aligned}$$

■

Note that by linearity, this means any vector of the form $(c, c, \ldots, c)$ for some constant c is also magic.

Theorem 8. The vector $v = (0, b+1, 0, b+1, \ldots, 0, b+1)$, where v is of even length, is a magic vector in $[n, k]^{(b)}$ for all $k > 1$.

Proof. Consider the dot products

$$\begin{aligned} (0, b+1, \ldots, 0, b+1) \cdot (b^{n-1}, b^{n-2}, \ldots, b, 1) &= 0 + (b+1)b^{n-2} + \cdots + 0 + (b+1)1 \\ &= b^{n-1} + b^{n-2} + \cdots + b + 1 = \mathbf{U}_n^{(b)} \\ &\equiv 0 \pmod{\mathbf{U}_n^{(b)}} \\ (0, b+1, \ldots, 0, b+1) \cdot (1, b, \ldots, b^{n-2}, b^{n-1}) &= 0 + (b+1)b + \cdots + 0 + (b+1)b^{n-1} \\ &= b + b^2 + \cdots + b^{n-1} + b^n = b\mathbf{U}_n^{(b)} \\ &\equiv 0 \pmod{\mathbf{U}_n^{(b)}}. \end{aligned}$$

■

So, in $[4, k]$, $k > 1$, $(0, 11, 0, 11)$ is a magic vector. This is why we call these *alternating "11" vectors*. By Theorem 7, we also have that $(6, 6, 6, 6)$ is a magic vector. Combined with Theorem 6, this explains the vector $(6, 17, 6, 17)$ we saw earlier.

Finally, we will discuss a trivial sort of magic vector, *repunit multiples*. From the definition of a magic vector in $[n, k]^{(b)}$, it is simple to see that adding abritrary multiples of the $\mathbf{U}_n^{(b)}$ to any element of a vector will preserve magicity.

Theorem 9. Let n and b be positive, nonzero integers. Let $\{c_i \colon 0 \leq i \leq n\}$ be a set of integers such that $0 \leq c_i \mathbf{U}_n^{(b)} \leq (b-1)k$ for all c_i. Then, the vector $v = (c_1\mathbf{U}_n^{(b)}, c_2\mathbf{U}_n^{(b)}, \ldots, c_n\mathbf{U}_n^{(b)})$ is magic in $[n, k]^{(b)}$.

Proof. Note that $v = \mathbf{U}_n^{(b)}(c_1, c_2, \ldots, c_n)$. Consider the dot products below.

$$\begin{aligned}
&\left[\mathbf{U}_n^{(b)}(c_1, c_2, \ldots, c_n)\right] \cdot (b^{n-1}, b^{n-2}, \ldots, b, 1) \\
&\quad = \mathbf{U}_n^{(b)} \left[(c_1, c_2, \ldots, c_n) \cdot (b^{n-1}, b^{n-2}, \ldots, b, 1)\right] \\
&\quad \equiv 0 \pmod{\mathbf{U}_n^{(b)}} \\
&\left[\mathbf{U}_n^{(b)}(c_1, c_2, \ldots, c_n)\right] \cdot (1, b, \ldots, b^{n-1}) \\
&\quad = \mathbf{U}_n^{(b)} \left[(c_1, c_2, \ldots, c_n) \cdot (1, b, \ldots, b^{n-1})\right] \\
&\quad \equiv 0 \pmod{\mathbf{U}_n^{(b)}}
\end{aligned}$$

■

As an example of this theorem, consider the trivial magic vector $(1, 1, 1)$. If we add $(111, 0, 0)$, the resultant vector $(112, 1, 1)$ is also magic.

2.5 Sets of Generators

Now that we have seen some generic magic vectors, and we know that Theorem 6 allows us to create new magic vectors through their linear combinations, it would be nice if we could find certain magic vectors that "generate" the others. Let's define exactly what we're looking for.

Definition 4. A set of magic vectors $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ in $[n, k]^{(b)}$ is a **set of generators for** $[n, k]^{(b)}$ if and only if any magic vector v in $[n, k]^{(b)}$ can be expressed as follows

$$v = w_1 g_1(\gamma_1) + w_2 g_2(\gamma_2) + \cdots + w_p g_p(\gamma_p) + \mathbf{U}_n^{(b)}(c_1, c_2, \ldots, c_n),$$

for some integer weights w_i, c_i, and $g_i \in G$, the group of cyclic shifts and rotations of v.

This definition allows us to concisely specify the elements in $[n, k]^{(b)}$ by listing the generators thereof. Note that it encompasses the "repunit multiples" class of magic vectors. The set of generators for a certain combination of n, k and b is not necessarily unique, so there may be many equally valid options for its representation. With this framework in place, we will turn to a study the generators for certain classes of magic vectors in base 10.

3 Base-10 Magic Vectors

By fixing a base b, we can represent the space of magic vectors in $[n,k]^{(b)}$ as a two-dimensional grid with one axis representing n and the other k. In this section we will investigate particular portions of this table for $b = 10$. See Figure 9 for a graphical representation of the results at present. Since $U_1 = 1$, all one-element vectors are trivially magic, and are not included on the table. First, we will categorize several columns of the table.

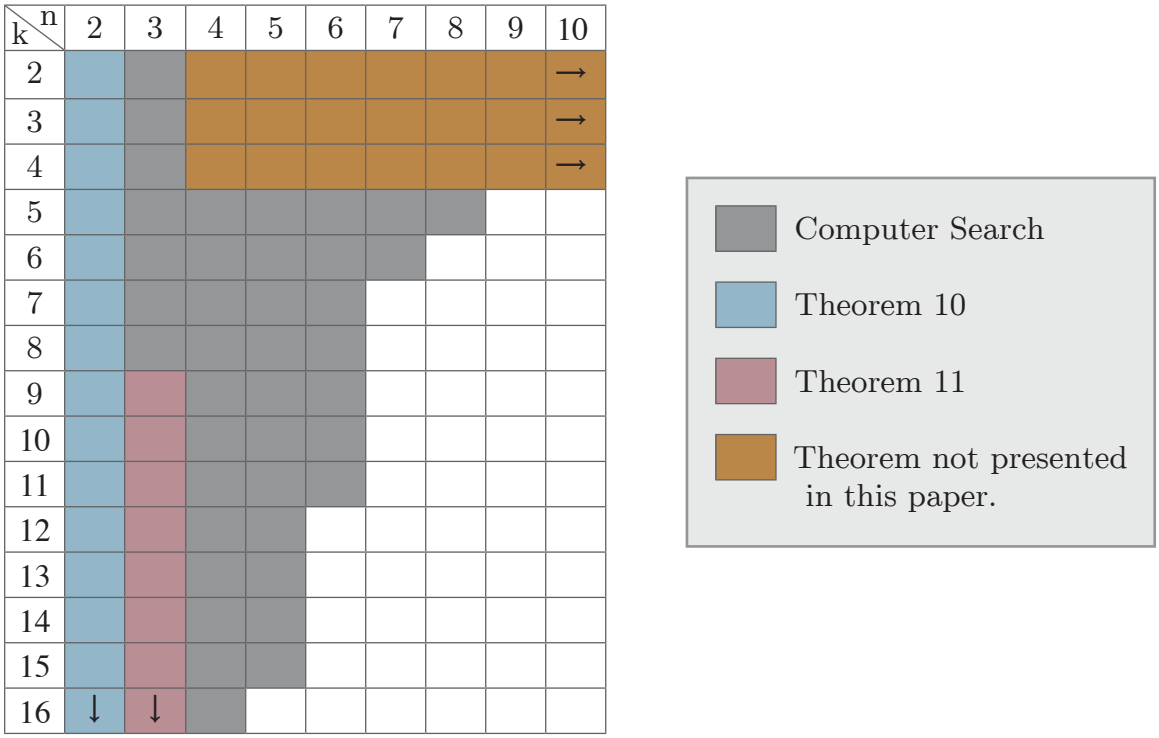

Figure 9: This chart represents the current status of magic vectors in base 10. Blank squares represents combinations of n and k where the magic vectors are currently unknown. The arrows indicate proofs that handle a certain case infinitely.

3.1 Magic Vectors in $[2,k]$

The situation for two-element vectors is summarized by the following theorem.

Theorem 10. The set $\Gamma = \{(1,1),(0,11)\}$ is a set of generators for $[2,k]$.

Proof. We have already seen that $(1,1)$ and $(0,11)$ are magic vectors, so it is obvious that all vectors generated by Γ are magic. Thus it suffices to show that all magic vectors in $[2,k]$ can be generated by the elements of Γ.

Let $v = (r, s)$ be a magic vector. If $r = s$, then $v = r(1, 1)$, which satisfies the theorem.

Otherwise, without loss of generality, let $r \leq s$. Then, since (r, r) is magic, so is $v' = v - r = (0, s - r)$.

By the definition of magic vectors, we have

$$v' \cdot (10, 1) = s - r \equiv 0 \pmod{11} \text{ and}$$
$$v' \cdot (1, 10) = 10s - 10r \equiv 0 \pmod{11}.$$

Rewriting either of these equations yields $s \equiv r \pmod{11}$. Thus for some integer k, we have $s = 11k + r$. This means our original vector v can be written as $(r, 11k + r) = r(1, 1) + (0, 11)$. ■

This theorem completely classifies all 2-element magic vectors. Expanding to account for reflections, they are of the form (c, c), $(c, c + 11)$ or $(c + 11, c)$ for some constant c.

Note that since $U_2 = 10 + 1$, the $(0, 11)$ vector is technically unnecessary since it is covered by the repunit multiples in the definition of a generating set. However, it is included here and in Figure 10 to provide cohesion with the representation of other generating sets, where these are two different cases.

3.2 Magic Vectors in $[3, k]$

The method of proof in the last section can be expanded to classify all magic vectors with 3 components. First, though, a lemma.

Lemma 2. Let v be a 3-element magic vector. Then any permutation of the elements of v is also magic.

Proof. Since v is magic, $g(v)$ is magic for all g in the group G. In this case, we have seen that G is the dihedral group of degree 3. However, this group is known to be identical to the permutation group S_3. Hence any permutation of the elements of v is the result of some element of G and also magic. ■

Theorem 11. The set $\Gamma = \{(1, 1, 1), (0, 37, 74)\}$ is a generating set for $[3, k]$ for any choice of k.

Proof. First, we will show that all vectors generated by the above set are magic. By Theorem 7, we know that $(1, 1, 1)$ is magic. So by linearity, it suffices to show that $(0, 37, 74)$ is magic.

Consider the dot products below.

$$(0, 37, 74) \cdot (1, 10, 100) = 370 + 7400 = 7770 \equiv 0 \pmod{111}$$
$$(0, 37, 74) \cdot (100, 10, 1) = 370 + 74 = 444 \equiv 0 \pmod{111}$$

Thus $(0, 37, 74)$ is magic.

Now we turn to the converse. For $k \leq 8$, the computer data show that all magic vectors are constant (see Figure 10). So, we will concentrate on the case of $k \geq 9$. Since the elements of a vector in $[n, k]$ must be $9k$ or less, $k \geq 9$ is the first setting where 74 becomes a valid element for a vector.

Assume $v = (q, r, s)$ is magic. By the lemma we can, without loss of generality, let $q \leq r \leq s$. Moreover, since (q, q, q) is magic, so is $v' = (0, r - q, s - q)$. Let $r' = r - q$ and $s' = s - q$.

So, by the definition of a magic vector,

$$\begin{aligned} 10r' + s' &\equiv 0 \pmod{111} \text{ and} \\ 10r' + 100s' &\equiv 0 \pmod{111}. \end{aligned} \tag{5}$$

Or, moving the s' terms to the other side,

$$10r' \equiv -s' \equiv 110s' \pmod{111} \text{ and} \tag{6}$$
$$10r' \equiv -100s' \equiv 11s' \pmod{111}. \tag{7}$$

Modulo 111, 100 is the multiplicative inverse of 10. Thus, multiplying both sides of **(6)** by 100 yields $r' \equiv 11s' \pmod{111}$. But, as **(7)** shows, it is also the case that $11s' \equiv 10r' \pmod{111}$. Thus, $10r' \equiv r' \pmod{111}$, or

$$9r' \equiv 0 \pmod{111}. \tag{8}$$

This is a *linear congruence*. See any other number theory text for a discussion of solving such problems. For our purposes, we note that since $\gcd(9, 111) = 3$ and $3 \mid 0$, **(8)** has 3 incongruent solutions modulo 111. One solution is obviously 0, so the others are $0 + \frac{111}{3} = 37$ and 74. We will consider each of these cases individually.

Case I: $r' \equiv 0 \pmod{111}$
Substituting this value of r' into **(5)**, we find that $s' \equiv 0 \pmod{111}$. Working back to our original vector, we have

$$r' = r - q \equiv 0 \pmod{111}$$
$$s' = s - q \equiv 0 \pmod{111}$$

meaning

$$\begin{aligned} r &\equiv q \pmod{111} \\ s &\equiv q \pmod{111}. \end{aligned}$$

So, for some integers b and c,

$$v = (q, q + 111b, q + 111c) = q(1, 1, 1) + 111(0, b, c),$$

which is generated by Γ.

Case II: $r' \equiv 37 \pmod{111}$
Again, substituting into **(5)**, we find $370 + s' \equiv 0 \pmod{111}$. Reducing 370 modulo 111 yields $37 + s' \equiv 0 \pmod{111}$. Thus, $s' \equiv 74 \pmod{111}$. Working back to the original vector,

$$\begin{aligned} r' = r - q &\equiv 37 \pmod{111} \\ s' = s - q &\equiv 74 \pmod{111} \end{aligned}$$

meaning

$$\begin{aligned} r &\equiv q + 37 \pmod{111} \\ s &\equiv q + 74 \pmod{111}. \end{aligned}$$

So, for some integers b and c,

$$v = (q, q + 37 + 111a, q + 74 + 111b) = q(1, 1, 1) + (0, 37, 74) + 111(0, b, c),$$

which is generated by Γ.

Case III: $r' \equiv 74 \pmod{111}$
From **(5)** we have $740 + s' \equiv 0 \pmod{111}$. Reducing 740 modulo 111 yields $74 + s' \equiv 0 \pmod{111}$. Thus, $s' \equiv 37 \pmod{111}$.

So, we have $v' = (0, r', s')$ with the congruences above. But, by the lemma, this can be shifted to $(0, s', r')$ without loss of magicity. In this form, **Case II** applies directly. ■

So we now have a complete classification of the vectors in $[3, k]$. For instance, in $[3, 34]$, $(74, 37, 0) + (113, 113, 113) + (111, 111, 0) = (298, 261, 224)$ is magic.

This method of proof becomes increasingly difficult for values of n greater than 3. Similar difficulty arises as the value of k increases. So, we will turn to some interesting phenomena discovered using the computer search.

3.3 Magic Vectors in $[6, k]$, $k \geq 8$

In $[6, 8]$, two previously unseen magic vectors appear: $(0, 7, 37, 34, 64, 71)$ and $(0, 4, 64, 68, 27, 41)$. The nature of these vectors and how to classify vectors in $[6, k]$ is unclear at present.

It is interesting to note that in $[6, 9]$, the same vectors appear, along with $(0, 37, 74, 0, 37, 74)$. It's no coincidence that this is $(0, 37, 74)$ repeated—it's fairly clear that concatenating r copies of an n-element magic vector yields an rn-element magic vector.

3.4 Magic Vectors in $[n, k]$, $n > 2$, $k \geq 12$

In the previous sections, we saw that increases in k can lead to unexpected new magic vectors. For $[3, 9]$, 74 became a legal vector element, and the new magic vector $(0, 37, 74)$ arose.

Similarly, when $k \geq 12$, 100 and 101 become legal elements. This gives rise to new vectors analagous to the alternating "11"s we saw earlier. For instance, in $[4, 12]$, $(0, 0, 101, 101)$, $(0, 1, 101, 1)$ and $(0, 100, 101, 100)$ are magic. Similar patterns arise as k increases to allow numbers like 111, 1000, 1001 and 1011. In $[6, 124]$, for example, $(0, 0, 1111, 0, 0, 110)$ is magic.

These patterns can be likened to a sort of "masking" phenomenon. In the dot product with powers of the base, the "holes" left by 0s in a vector get filled in by the 1s in other components. The same was the case in Theorem 8, but these vectors have more variation. Take the example of $(0, 0, 101, 101)$. Let's express this as $(0, 0, 10^2+1, 10^2+1)$ Then, one of the dot products is

$$\begin{aligned}(0, 0, 10^2 + 1, 10^2 + 1) \cdot (10^3, 10^2, 10, 1) &= 10^3(0) + 10^2(0) + 10(10^2 + 1) + 10^2 + 1 \\ &= 10^3 + 10^2 + 10 + 1 \\ &= 1111.\end{aligned}$$

Interestingly, this pattern is not limited to vectors of the above form. In $[5, 12]$, $(0, 0, 101, 91, 101)$ is magic. Carrying it through one of the dot products yields a similar result:

$$\begin{aligned}&(0, 0, 10^2 + 1, 9 \cdot 10 + 1, 10^2 + 1) \cdot (10^4, 10^3, 10^2, 10, 1) \\ &\quad = 10^4(0) + 10^3(0) + 10^2(10^2 + 1) + 10(9 \cdot 10 + 1) + 10^2 + 1 \\ &\quad = 10^4 + (9 \cdot 10^2 + 10^2) + 10^2 + 10 + 1 \\ &\quad = 10^4 + 10^3 + 10^2 + 10 + 1 \\ &\quad = 11111.\end{aligned}$$

n = 1	
any k	(1)
n=2	
any k	(1, 1) (0, 11)
n=3	
k ≤ 8	(1, 1, 1)
k ≥ 9	(0, 37, 74)
Fom now on, all results are partial.	
n=4	
k ≤ 11	(1, 1, 1, 1) (0, 11, 0, 11)
k = 12	(0, 0, 101, 101) (0, 1, 101, 1)
k = 13	(0, 10, 0, 111) (0, 111, 101, 111) (0, 1, 101, 1) (0, 112, 101, 11)
k = 14, 15, 16	(0, 122, 101, 122)
n=5	
k ≤ 11	(1, 1, 1, 1, 1)
k = 12	(0, 0, 101, 91, 101) (0, 10, 0, 101, 101)
k = 13	(0, 10, 10, 0, 111) (0, 111, 101, 101, 111)
k = 14	(0, 111, 111, 0, 121) (0, 121, 10, 10, 121)
n=6	
k ≤ 7	(1, 1, 1, 1, 1, 1) (0, 11, 0, 11, 0, 11)
k = 8	(0, 7, 37, 34, 64, 71) (0, 4, 64, 68, 27, 41)
k = 9	(0, 37, 74, 0, 37, 74) (0, 52, 27, 79, 64, 15)
k = 10	(0, 85, 37, 11, 74, 48) (0, 82, 64, 45, 37, 18)
k = 11	(0, 1, 91, 92, 91, 1)
n=7	
k ≤ 6	(1, 1, 1, 1, 1, 1, 1)
n=8	
k ≤ 5	(1, 1, 1, 1, 1, 1, 1, 1) (0, 11, 0, 11, 0, 11, 0, 11)

Figure 10: This table shows the generators the certain values of n and k. The generators are cumulative. For example, the generators for $[4, 13]$ inclue those for $[4, k]$, $k < 13$. For $k < 4$, the results are complete. After 4, the listed values are the only ones currently known.

4 Computer Search

As Figure 9 shows, many of the generating vectors in Figure 10 were the result of a computerized search for magic vectors. Given n and k, the program steps through n-element vectors with digits up to $9k$. Each of these is tested using the dot product condition in the definition of magic vector.

If implemented naively, this algorithm would have to check $(9k+1)^n$ vectors. However, due to the cyclic nature of these vectors, many of these are redundant. For instance, (a, b, c) is magic if and only if (c, b, a) is, so there's no reason to check both.

Avoiding all repeats would involve caching the previous tests. This in itself is a time- and memory-expensive change, and was ruled out. That said, we can still take some shortcuts. The one implemented in the C code in Appendix A operates with the following in mind: any magic vector can be shifted such that its maximal element is on the far right. By ensuring this is always the case, we can greatly speed up the search.

Even with this improvement, the search program can easily take hours or days given certain values of n and k. On a 2 gHz Intel processor, the search in $[6, 11]$ took about 5.5 hours, and $[8, 5]$ took several days. Despite these shortcomings, the data produced by the program were invaluable in discerning patterns in $[3, k]$ and uncovering the new generators in $[6, 8]$ and $[n, 12]$.

Since we are only interested in a generative subset of vectors, not the raw findings of the brute-force search, a post-processing program was written to collapse equivalent vectors into one representative. The Haskell code for this program is in Appendix B.

5 Future Study

There are still many open questions in the realm of magic numbers and vectors. First and foremost are explaining the $[6, 8]$ phenomenon and filling in more of the base-10 chart. The latter will likely require improved proof methods.

The search program could definitely be optimized and parallelized to enable it to take advantage of cluster-computing environments. This would push the frontier of feasible calculations, possibly uncovering new patterns and elucidating old ones.

Research in bases other than 10 is wide open. There are certainly insights to be gained by investigating these territories. It is especially interesting to note that repunits in binary (base 2) are the well-known Mersenne Numbers.

An interesting development could be based around prime repunits. In this case, the full theory of Linear Algebra applies to the magic vector criteria, which is very promising. Unfortunately, prime repunits are relatively rare, so this may not shed much light on the subject. For instance, in base 10, the only known prime repunits $\mathbf{U}_n$ are for $n = 2, 19, 23, 317, 1031, 49081, 86453$, and 109297.

Appendix A: Search Program

This is fairly standard C code and should compile cleanly as long as the compiler supports the 64-bit integer type *uint64_t*.

search.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h> //for memset

#include "search.h"
#include "cmdline.h"

#ifdef SIGHANDLER
#include "signals.h"
#endif

INTTYPE *testvector;
unsigned int nval = 0;

#ifdef DEBUG
unsigned int tests = 0, shortcuts = 0;
#endif


INTTYPE Un(unsigned int n, unsigned int base) {
    // (base^n - 1) / (base - 1)
    int i;
    INTTYPE basepow = 1;

    for(i = 0; i < n; i++)
        basepow *= base;

    return (basepow - 1) / (base - 1);
}


// initialize and set v to powers of ten from 0 ... n - 1
void init_powers(INTTYPE v[], unsigned int n, unsigned int base) {
    size_t i;
```

```
    v[0] = 1;

    for(i = 1; i < n; i++)
        v[i] = v[i - 1] * base;
}
```

/* We pre-compute Un and the powers to save time when re-calling this with the same n, as we will be doing. */

```
int is_magic(INTTYPE v[], unsigned int n, INTTYPE powers[], INTTYPE Un)
{
    size_t i;
    INTTYPE forward = 0, reverse = 0;

#ifdef DEBUG
    tests++;
#endif

    for(i = 0; i < n; i++)
        forward += v[i] * powers[i];

    if(forward % Un) {
#ifdef DEBUG
        shortcuts++;
#endif
        return 0;
    }

    for(i = 0; i < n; i++)
        reverse += v[i] * powers[n - i - 1];

    return !(reverse % Un);
}


void show_vector(INTTYPE v[], size_t n) {
    size_t i;

    putchar('(');

    for(i = 0; i < n - 1; i++)
        printf(PRINTF " ", v[i]);

    printf(PRINTF ")", v[n - 1]);
}
```

```
// returns true if there are more vectors
int next_vector_fast(INTTYPE v[], unsigned int n, unsigned int k, unsigned
int base) {
    unsigned int max = k * (base - 1);
    size_t i = n - 1, j;
    INTTYPE maxelt = 0;

    while(v[i] == max) {
        if(i == 0) return 0;  // this means everything is max

        v[i] = 0;
        i--;
    }

    v[i]++;


    // Track back, find the max element, and set the far right element
to it.
    for(j = 0; j <= i; j++)
        if(v[j] > maxelt) maxelt = v[j];

    v[n - 1] = maxelt;

    return 1;
}

int next_vector_exhaustive(INTTYPE v[], unsigned int n, unsigned int k,
unsigned int base) {
    unsigned int max = k * (base - 1);
    size_t i = n - 1;

    while(v[i] == max) {
        if(i == 0) return 0;  // this means everything is max

        v[i] = 0;
        i--;
    }

    v[i]++;

    return 1;
}
```

```
int main(int argc, char *argv[]) {
    INTTYPE *powers, U;
    unsigned int base, k;
    int mode, searchmode;
    char *vectorstr;
    next_vector_fn next_vector = next_vector_fast;

#ifdef SIGHANDLER
    setup_handler();  // The ctrl+c resume stuff
#endif


//// Parse command-line options & initialize test vector
    parse_options(argc, argv, &nval, &k, &base, &mode, &vectorstr, &searchmode);

    testvector = (INTTYPE*) malloc(sizeof(INTTYPE) * nval);

    if(mode == MODE_RESUME || mode == MODE_CHECK) {
        if(!scan_vector(vectorstr, testvector, nval)) {
            fprintf(stderr, "Error in resume string.\n\n");
            usage(argv[0]);
        }
    }
    else
        memset(testvector, 0, sizeof(INTTYPE) * nval);

    if(searchmode == SEARCH_EXHAUSTIVE)
        next_vector = next_vector_exhaustive;
////


//// Initialize powers of the base and U_n
    powers = (INTTYPE*) malloc(sizeof(INTTYPE) * nval);
    init_powers(powers, nval, base);
#ifdef DEBUG
    printf(">> Base powers: ");
    show_vector(powers, nval);
    putchar('\n');
#endif

    U = Un(nval, base);
#ifdef DEBUG
```

```
    printf(">> U_%u: " PRINTF "\n", nval, U);
#endif
////
```

//// The actual test

```
    if(mode == MODE_CHECK) {
        if(is_magic(testvector, nval, powers, U))
            puts("Vector is magic.");
        else
            puts("Vector is not magic.");

        return 0;
    }


    do {
#ifdef SHOWVECTS
        printf(">> ");
        show_vector(testvector, nval);
        putchar('\n');
#endif

        if(is_magic(testvector, nval, powers, U)) {
            show_vector(testvector, nval);
            putchar('\n');
        }
    } while(next_vector(testvector, nval, k, base));
////


#ifdef DEBUG
    printf("tests: %u\nshortcuts: %u\ndiff: %u\n", tests, shortcuts, tests
- shortcuts);
#endif


    free(testvector);
    free(powers);

    return 0;
}
```

search.h

```
// #define DEBUG
// #define SHOWVECTS  // show each vector as it's generated
#define SIGHANDLER

#define INTTYPE uint64_t
#define PRINTF "%llu"
#define DEFAULTK 1
#define DEFAULTBASE 10

#define MODE_SEARCH 0
#define MODE_RESUME 1
#define MODE_CHECK  2

#define SEARCH_FAST       0
#define SEARCH_EXHAUSTIVE 1

typedef int (*next_vector_fn)(INTTYPE[], unsigned int, unsigned int, unsigned
int);



INTTYPE Un(unsigned int, unsigned int base);
void init_powers(INTTYPE v[], unsigned int n, unsigned int base);
int is_magic(INTTYPE v[], unsigned int n, INTTYPE powers[], INTTYPE Un);
void show_vector(INTTYPE v[], size_t n);
int next_vector_fast(INTTYPE v[], unsigned int n, unsigned int k, unsigned
int base);
int next_vector_exhaustive(INTTYPE v[], unsigned int n, unsigned int k,
unsigned int base);
```

cmdline.h

```
#include <stdio.h>
#include <getopt.h>

void usage(const char *program_name) {
    fprintf(stderr, "Usage: %s [options] n [k]\n", program_name);
    fprintf(stderr, "Options:\n");
    fprintf(stderr, "  -r  --resume \"(x x ... x)\"\tResume search from
given vector.\n");
    fprintf(stderr, "  -c  --check  \"(x x ... x)\"\tCheck if the given
vector is magic.\n");
```

```
    fprintf(stderr, "  -b  --base   x              \tWork in the given base.\n");
    fprintf(stderr, "  -e  --exhaustive            \tSearch exhaustively,
without optimizations.\n");

    exit(1);
}


void parse_options(int argc, char *argv[], unsigned int *n, unsigned int
*k, unsigned int *base, int *mode, char **vectorstr, int *searchmode) {
    static struct option long_opts[] = {
            {"resume", 1, NULL, 'r'},
            {"check", 1, NULL, 'c'},
            {"base", 1, NULL, 'b'},
            {"exhaustive", 0, NULL, 'e'},
            {0, 0, 0, 0}
        };
    const char* const short_opts = "r:c:b:e";

    const char* program_name = argv[0];
    int next_opt;

    *mode = MODE_SEARCH;
    *searchmode = SEARCH_FAST;
    *base = DEFAULTBASE;

    if(argc == 1) usage(program_name);

    while((next_opt = getopt_long(argc, argv, short_opts, long_opts, NULL))
!= -1) {
        switch(next_opt) {
            case 'r':
                *mode = MODE_RESUME;
                *vectorstr = optarg;
            break;

            case 'c':
                *mode = MODE_CHECK;
                *vectorstr = optarg;
            break;

            case 'e':
                *searchmode = SEARCH_EXHAUSTIVE;
            break;
```

```
            case 'b':
                *base = atoi(optarg);
            break;

            case '?':
            default:
                usage(program_name);
        }
    }

    argc -= optind;
    argv += optind;

    if(argc == 0) usage(program_name);

    *n = atoi(argv[0]);

    if(argc > 1) *k = atoi(argv[1]);
    else *k = DEFAULTK;
}


int scan_vector(const char *str, INTTYPE v[], unsigned int n) {
    INTTYPE num;
    size_t i;

    if(sscanf(str, " (" PRINTF, &num) != 1)
        return 0;

    v[0] = num;

    for(i = 1; i < n; i++) {
        str = strnstr(str, " ", strlen(str)) + 1;

        if(sscanf(str, " " PRINTF, &num) != 1)
            return 0;

        v[i] = num;
    }

    return 1;
}
```

signals.h

```
#include <signal.h>
#include <string.h> //for memset

extern INTTYPE *testvector;
extern unsigned int nval;

void sig_handler(int signal_number) {
    if(testvector) {
        printf("\nAborting. Resume with the command line argument:\n-r \"");
        show_vector(testvector, nval);
        printf("\"\n");
    }

    exit(1);
}

void setup_handler() {
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = &sig_handler;

    sigaction(SIGINT, &sa, NULL);
}
```

Appendix B: Haskell Post-Processor

Haskell is a purely functional language with roots in ML. Interpreters and interpreter-compilers are available for most operating systems. Check *http://haskell.org* for details.

Post.hs

```
module Main where

import System.Environment (getArgs)
import Data.List (nubBy, minimum, intersectBy, deleteFirstsBy, delete)


-- Cyclic rotation of a vector
```

```
rot :: [a] -> [a]
rot (x:xs) = xs ++ [x]
rot [] = []

-- All cyclic rotations of a given vector
rots :: [a] -> [[a]]
rots l = take (length l) (iterate rot l)

-- Generate all reflections of the list of vectors
refls :: [[a]] -> [[a]]
refls = map reverse

-- Find the set of possible adjustments of the second vector that might equal
-- the first.  That is:
--  * rotations of l2 with first digit equal to that of l1
--  * reflections of rotations of l2 with last digit equal to the first of l1
-- The two are equivalent (element-wise) if l1 is somewhere in this list.
possibilities :: Eq a => [a] -> [a] -> [[a]]
possibilities l1 l2 = possrots ++ possrefls
                      where allrots = rots l2
                            possrots  = filter
                                        (\v -> head v == head l1)
                                        allrots
                            possrefls = refls $ filter
                                        (\v -> last v == head l1)
                                        allrots

-- Returns true if the given vectors differ only by some vector of multiples
-- of eleven.
offByEleven :: (Integral a, Eq a) => [a] -> [a] -> Bool
offByEleven l1 l2 | ll1 /= ll2 = False
                  | odd ll1    = False
                  | otherwise  = diff rl1 rl2 == zero ||
                                 diff rl2 rl1 == zero
  where reduce n l = map (`mod` n) l
        diff = zipWith (-)
        ll1 = length l1
        ll2 = length l2
        zero = take ll1 (repeat 0)
        rl1  = reduce 11 l1
        rl2  = reduce 11 l2

-- The equivalence test for two vectors
equiv :: Integral a => [a] -> [a] -> Bool
equiv l1 l2 | offByEleven l1 l2 = True
```

```
            | otherwise = elem l1 (possibilities l1 l2)

-- Normalize a vector by subtracting a constant vector (x x ... x).
-- This takes care of the constant cases by bringing them all down to (0 0 .. 0).
normalize :: (Ord a, Num a) => [a] -> [a]
normalize l = map (subtract (minimum l)) l

-- Preferred standard form: rotated so a zero falls in the first slot.
stdform :: Num a => [a] -> [a]
stdform xs = (snd s) ++ (fst s)
             where s = break (== 0) xs

-- Glued together: a list of vectors to its (semi-)unique subset
-- This fixes up some of our mess along the way, like replacing (0 0 ... 0)
-- with (1 1 ... 1), and adding back the alternating elevens vector, if
-- appropriate.
uniq :: [[Int]] -> [[Int]]
uniq vs = fixed
  where raw = (map stdform) . (nubBy equiv) . (map normalize) $ vs
        n = length (head vs)
        zeroes  = take n (repeat 0)
        ones    = take n (repeat 1)
        elevens = if (even n) then [take n (cycle [0, 11])] else []
        fixed = [ones] ++ elevens ++ (delete zeroes raw)

-- Completely glued together: a loaded file to its unique vectors
processFile :: String -> [[Int]]
processFile = uniq . toVects
--processFile = (map stdform) . (nubBy equiv) . (map normalize) . toVects


-- Reads a vector from a string
readVect :: ReadS [Int]
readVect = readParen False (\r -> [pr | ("(",s)  <- lex r,
                                        pr       <- readl s])
           where readl  s = [([],t)   | (")",t)  <- lex s] ++
                            [(x:xs,u) | (x,t)    <- reads s,
                                        (xs,u)   <- readl' t]
                 readl' s = [([],t)   | (")",t)  <- lex s] ++
                            [(x:xs,v) | (x,u)    <- reads s,
                                        (xs,v)   <- readl' u]

-- Converts a string of many vectors into a list of vectors
toVects :: String -> [[Int]]
toVects ""   = []
```

```
toVects "\n" = []
toVects s    = [(fst parse)] ++ toVects (snd parse)
               where parse = head (readVect s)

-- Pretty pring a vector
pshow :: Show a => [a] -> String
pshow (x:xs) = "(" ++ show x ++ pshow' xs
               where pshow' (y:ys) = " " ++ show y ++ pshow' ys
                     pshow' []     = ")"
pshow [] = ""

-- Convert a list of vectors to a newline-delimited string
showVects :: Show a => [[a]] -> String
showVects = unlines . (map pshow)


-- Main:
--  * If given one argument, just print that file's unique vectors
--  * If given two arguments, show their intersection and symmetric
--    differences, or a notice if they contain the same information.
main = do
        args <- getArgs
        if (length args == 1) then do
          file <- readFile (head args)
          putStr (showVects (processFile file))
          else do
           file1 <- readFile (head args)
           file2 <- readFile (head (tail args))
           let min1 = processFile file1
               min2 = processFile file2
               overlap = intersectBy equiv min1 min2
               diff1 = deleteFirstsBy equiv min1 overlap
               diff2 = deleteFirstsBy equiv min2 overlap
           putStrLn "Intersection:"
           putStrLn (showVects overlap)
           if (null diff1 && null diff2) then do
             putStrLn "Files contain the same vectors."
             else do
               putStrLn ((head args) ++ " \\ Intersection:")
               putStrLn (showVects diff1)
               putStrLn ((head (tail args)) ++ " \\ Intersection:")
               putStrLn (showVects diff2)
```