THE FLAMES ALGORITHM

I AND P

1. Introduction

The games of `FLAMES` is apparently a popular game played by teenagers as a way to deal with the unpredictability of their romantic relationships. This belongs to the vast class of prediction devices such as the counting of leaves, numerological tricks, horoscopes and the like. However, unlike many of these other methods, the `FLAMES` game is based on a transparent deterministic algorithm. Even so the allure seems to be based on the seemingly complicated dynamics of the algorithm that can keep the player guessing about the outcome till the very last and adds to the challenge of manipulating the input to produce a desired outcome. One (serious) motivation for studying this game was based on the observation of the similarity of this game to the class of rule-based string rewriting algorithms invented by Emil Post called *tag systems*, and as it turned out the interesting number theoretic angle that the game brings to tag systems.

2. Description of the FLAMES game

The rules of the FLAMES game can be described as follows. Take the names of two people, usually of opposite genders, and cross out all the common letters in the two names. Count the total number of letters that remain in both names after this procedure. Let this number of mismatches be m.

Now write `FLAMES` on a piece of paper. Count through the letters of this string starting from `F` to `S` and cycling back through `F` till m letters have been counted. At this point, cross out the letter in `FLAMES` at which the count ends (say `A`). Restart counting from the next letter (here `M`) through the string `FLMES`. Repeat this till five of the letters of `FLAMES` have been crossed out and only one letter remains.

The final letter that remains is the predicted nature of the relationship between the two people whose names were initially chosen, where `F` = "Friend", `L` = "Love", `A` = "Adore", `M` = "Marry", `E` = "Enemy", `S` = "Support". The rationale (or the lack of it) for these interpretations is beyond the scope of this paper.

To provide a concrete example, let us consider the predicament of *Alice* and *Stevens*. From Table 1, we can see the total number of mismatches $m = 4 + 5 = 9$, where the letters common to both names are shown in bold font.

Date: 13 November, 2005.

Thanks to The Smugbug for bringing the algorithm to the attention of the first author.

TABLE 1. Names and number of mismatched letters

A	L	I	C	**E**			4
S	T	**E**	V	**E**	N	S	5

Using this value of $m = 9$, the iteration through the string `FLAMES` is shown in Table 2. The letter struck out at each iteration is shown in bold font. For the value $m = 9$, the predicted relation between *Alice* and *Stevens* is `S`.

TABLE 2. Trace of execution for $m = 9$

Iteration	Start							Removed
1	F	F	L	**A**	M	E	S	A
2	M	F	**L**		M	E	S	L
3	M	F			**M**	E	S	M
4	E	**F**				E	S	F
5	E					**E**	S	E
6	Result						**S**	[S]

As this prediction may not correspond to the desired/expected relationship between the protagonists often attempts are made to manipulate the result by changing the way the names are presented to change the value of m. Anecdotal evidence seems to suggest that a large component of the intrigue of this game is in finding a manipulation that produces the desired result. One rule for this manipulation is that both names be treated symmetrically. If full names are used then this has to be so for both the persons in question, and if nicknames are used this is done for both names. However, this rule is apparently violated often flagrantly in the desire to get a desired result. The ability to manipulate the outcome is the central motivation of our investigation.

While the outcome of interest in the game is the final remaining letter, here we will focus on the string generated by the letters successively struck out at each iteration for a given m. In the above example, this string is $ALMFES$. Since one letter is always obtained from s at each iteration and this letter is subsequently removed from subsequent iterations (i.e. no chance of repeats), this resultant string is essentially a permutation of the string $s = FLAMES$.

This brings us to the decision problem: *Given a random permutation of s say s, does there exist a value $m \in \mathbb{N}$ for which the algorithm will produce s'?*. In different words, if we were given a random string $ESFLMA$ is there an efficient way to determine if there is some number m that can produce it via the algorithm or not?

3. IDEA

This section provides a sketch of the strategy used to analyze this algorithm.

Given a finite string s, the indices of the symbols range from 1 to $l = |s|$. In short, $s = s_1 s_2 ... s_l$. The concatenation of two strings x and y is denoted by xy. The

operator $pref_i(y)$ returns the prefix string $x = y_1y_2...y_{i-1}$ and the empty string if $i = 1$. Similarly the operator $suff_i(y)$ returns the suffix string $x = y_{i+1}y_{i+2}...y_{|y|}$ and the empty string if $i = |y|$.

With this background, the FLAMES algorithm can be expressed in the simpler form shown by Algorithm 1.

Algorithm 1 FLAMES algorithm

```
1: flames (s,m) {
2: l ← |s|
3: if l = 1 then
4:    Print s
5:    Return
6: else
7:    p ← m mod l
8:    if p = 0 then
9:       p ← l
10:   end if
11:   Print s_p
12:   x ← pref_p(s)
13:   y ← suff_p(s)
14:   flames(yx, m)
15: end if
16: }
```

Let $R_m \in \pi_{\mathbf{s}}$ be the string obtained. From this rewrite of the algorithm, the following results pop out.

Proposition 3.1. *$R_{m'} = R_m$ if and only if $m' \bmod i \equiv m \bmod i$ for every $i \in \{1, 2, \cdots, |s|\}$*

Theorem 3.2. *For any two numbers $m', m \in \mathbb{N}$, the relation $m' \bmod i \equiv m \bmod i$ for every $i \in \{1, 2, \cdots, |s|\}$ is true if and only if $m' = \tau j + m$ where $\tau = LCM(1, 2, \cdots, |s|)$ and $j \in \mathbb{N}$.*

Corollary 3.3. *$R_m \neq R'_m$ for any $1 \leq m, m' \leq \tau(m \neq m')$.*

Corollary 3.4. *The total number of distinct sequences R_m for all $m \in \mathbb{N}$ is equal to $\tau = LCM(1, 2, \cdots, |s|)$.*

4. Synopsis

The `FLAMES` algorithm f is effectively a string rewriting system that is defined on a finite string s (drawn from a language Σ^*) where no characters are repeated. In the canonical form of the game the string $s = FLAMES$, hence the name of the game. The algorithm takes a positive integer $m \in \mathbb{N}(m > 0)$ as input and rewrites s in a series of rule-governed steps to produce a permutation s' of the string s. In the typical form in which the game is played, the last character of the string s' is considered to be the output. However, we will focus on a more interesting variant of the game, the π-`FLAMES` game, where the permutation s' is the output.

The allure of this game is that it seems to behave like a *one-way function*, i.e.:

(1) We are given the function $f_s : \mathbb{N} \to \pi_{\mathbf{s}}$, where the description of f_s is publicly known and does not require any secret information for its operation.[1]
(2) Given $m \in \mathbb{N}$, it is easy to compute $f_s(m)$.
(3) However, given some $s' \in \pi_{\mathbf{s}}$, it (intuitively) seems "hard" to find an m such that $f_s(m)$ is equal to s' or even if such an m exists.

Since $\mathbb{N}$ is enumerable and f is a total function, one obvious way to address the problem of finding $f^{-1}(s')$ is to exhaustively generate the strings corresponding to each $m \in \mathbb{N}$ and check whether $s' = f(m)$, halting when such an m is obtained. However, f_s is neither injective nor surjective so there may be no m such that $s' = f_s(m)$. Consequently, this enumerative procedure is not guaranteed to halt if s' in the range of f_s.

The main result of this investigation is that the function f_s over the domain $\mathbb{N} \bmod \tau$ is injective, where $\tau = LCM(1, 2, \cdots, |s|)$, and the range for this domain is identical to that for the domain $\mathbb{N}$.

So:

(1) When s is known, there is an efficient algorithm for f^{-1} that is linear in τ (i.e. $O(\tau)$). (This algorithm does not involve enumeration and is the equivalent of retracing the number using the known indices of s and s')
(2) When s is unknown but $|s|$ is known, the decision problem associated with f is *effectively decidable*, i.e. given any $s' \in \pi_{\mathbf{s}}$, there is an enumeration-based procedure using f_s that is guaranteed to halt, returning an answer "yes" if $s' = f_s(m)$ for some $m \in \mathbb{N}$ and "no" if no such m exists. More specifically: *if* $s' \neq f_s(i)$ *for any* i *in the range* $[1, \tau] \subset \mathbb{N}$ *then there exists no* $m \in \mathbb{N}$ *for which* $s' = f_s(m)$.

So, one can think of this as a way to encrypt single numbers, where the program f_s is publicly available and the algorithm for f_s is publicly known but the string s is a secret. Alice uses f_s to encrypt a number $m \in \mathbb{N}$ and sends the resultant string $s' = f_s(m)$ to Bob. Since Bob knows the secret key s, he can easily decrypt s' to obtain m.

The question then is: if Oscar intercepts s' can he use f_s to determine the number m that Alice sent to Bob? The answer is a resounding "yes" – Oscar can efficiently find m. And in fact if f_s is publicly available, he can efficiently use it to determine the secret key s. All in all, this is a terribly insecure cryptosystem and there is very little that is really cryptic about it!

[1]The set of all permutations of s is denoted here by $\pi_{\mathbf{s}}$.

5. Disclamer

I think all of the above is correct but if you think we are full of you-know-what then please do let us know why as we're always eager to find out a better solution to all this.