Cycle Finding Problem

1. Introduction

Assume that you have a function **f: S → S** and any initial value $\mathbf{x_0} \in \mathbf{S}$
(in this visualization, we are restricted to **f(x) = (A*x^2 + B*x + C) % M** and $\mathbf{x_0}$ **% M** hence the function **f** has domain and range ∈ [0..**M**-1]).

The sequence of iterated function values:
$\{\mathbf{x_0}, \mathbf{x_1 = f(x_0)}, \mathbf{x_2 = f(x_1)}, ..., \mathbf{x_i = f(x_{i-1})}, ...\}$
must eventually use the same value twice (cycle), i.e. $\mathbf{a \neq b}$ such that $\mathbf{x_a = x_b}$.
Afterwards, the sequence must continue by repeating the cycle of values from $\mathbf{x_a}$ to $\mathbf{x_{b-1}}$.

Let **mu (μ)** be the smallest index and let **lambda (λ)** (the cycle length) be the smallest positive integer such that $\mathbf{x_\mu = x_{\mu+\lambda}}$.

The cycle-finding problem: Find such **μ** and **λ**, given **f** and $\mathbf{x_0}$.

2. Custom Input

You can define custom function $\mathbf{f(x) = (A*x^2 + B*x + C) \% M}$ here.

1. Random: The function will be $\mathbf{f(x) = (x^2 - 1) \% M}$ and only **M** and the $\mathbf{x_0}$ are randomly generated.
2. Custom: You can specify the coefficients **A**, **B**, **C** of **f(x)** (ranging from -999 to 999), the modulo value **M** (ranging from 10 to 1000) and the initial value $\mathbf{x_0}$ (ranging from 0 to **M-1**).

You can also set custom $\mathbf{x_0}$, which must be ∈ [0..**M**-1].

3. Floyd's Tortoise-Hare Algorithm

Floyd's Tortoise-Hare Cycle-Finding is one algorithm that can solve this problem efficiently in both time and space complexities.

It just requires O(**μ+λ**) time and O(**1**) space to do the job.

3-1. The Visualization

This is the visualization of Floyd's Tortoise-Hare Cycle-Finding algorithm.

The shape of **rho (ρ)** of the sequence of iterated function values defined by **f(x)** and $\mathbf{x_0}$ nicely visualizes **μ** and **λ**.

VisuAlgo uses green vertices to represent the tortoise (t) and orange vertices to represent the hare (h).

3-2. Finding kλ (a multiple of λ)

We start from $\mathbf{x_0}$.

While **tortoise's pointer** != **hare's pointer**, we advance **tortoise/hare** by one step/two steps to their next values by calling **f(tortoise)/f(f(hare))**.

3-3. Finding μ (the start of the cycle)

We reset hare back to $\mathbf{x_0}$ and keep tortoise at its current position.

Then, we iteratively advance both pointers to their next values by one step as this maintains the gap of tortoise and hare by **kλ**.

The first time both pointers are equal tells us the value of **μ**.

3-4. Finding λ (the length of the cycle)

Again, we let tortoise stays in its current position and set hare next to it.

Then, we iteratively move hare to the next value by one step.

The first time both pointers are equal tells us the value of **λ**.

4. Try the Animation!

The animation of this algorithm should clear up most questions involving this algorithm.

We will expand this e-Lecture slides to contain more information about this algorithm later.

5. Implementation

You are allowed to use/modify our implementation code for Floyd Cycle-Finding Algorithm:

UVa00350.cpp
UVa00350.java
UVa00350.py
UVa00350.ml