CDM
Iteration

Klaus Sutner
Carnegie Mellon Universality

30-iteration 2017/12/15 23:20

Computational Memes 3

There are several general ideas that are useful to organize computation, perhaps the two most important ones being

- Recursion (self-similarity)
- Iteration (repetition)

Recursion is quite popular and directly supported in many programming languages.

Unfortunately, iteration requires some amount of extra work.

Iteration 4

Definition

Let $f : A \to A$ be an endofunction. The kth power of f (or kth iterate of f) is defined by induction as follows:

$$f^0 = I_A$$
$$f^k = f \circ f^{k-1}$$

Here I_A denotes the identity function on A and $f \circ g$ denotes composition of functions.

Informally, this just means: compose function f $(k-1)$-times with itself.

$$f^k = \underbrace{f \circ f \circ f \circ \ldots \circ f}_{k \text{ terms}}$$

General Laws 5

Without any further knowledge about f there is not much one can say about the iterates f^k. But the following always holds.

Lemma (Laws of Iteration)

- $f^n \circ f = f^{n+1}$
- $f^n \circ f^m = f^{n+m}$
- $(f^n)^m = f^{n \cdot m}$

Exercise

Prove these laws by induction using associativity of composition.

Wurzelbrunft's Idea 6

Prof. Dr. Alois Wurzelbrunft stares at these equations and immediately recognizes a deep analogy to exponentiation.

He also remembers that there is a method for fast exponentiation based on squaring:

$$a^{2e} = (a^e)^2$$
$$a^{2e+1} = (a^e)^2 \cdot a$$

which allows us to compute a^e in $O(\log e)$ multiplications.

Wurzelbrunft's Conclusion:
There is an analogous "fast iteration" method.

Aside

> Good mathematicians see analogies between theorems or theories; the very best ones see analogies between analogies.
>
> S. Banach

So is Wurzelbrunft brilliant?

Fast Iteration

Suppose we want to compute f^{1000}. The obvious way requires 999 compositions of f with itself.

By copying the standard divide-and-conquer approach for fast exponentiation we could try

$$f^{2n} = (f^n)^2$$
$$f^{2n+1} = f \circ (f^n)^2$$

This seems to suggest that we can compute $f^n(x)$ in $O(\log n)$ applications of the basic function f.

After all, it's just like exponentiation, right?

Linear Maps

If the function f in question is linear it can be written as

$$f(x) = M \cdot x$$

where M is a square matrix over some suitable algebraic structure. Then

$$f^t(x) = M^t \cdot x$$

and M^t can be computed in $O(\log t)$ matrix multiplications.

So this is an exponential speed-up over the standard method.

Polynomial Maps

Another important case is when f is a polynomial map

$$f(x) = \sum a_i x^i$$

given by a coefficient vector $\boldsymbol{a} = (a_d, \ldots, a_1, a_0)$.

In this case the coefficient vector for $f \circ f$ can be computed explicitly by substitution. This is useful in particular when computation takes place in a quotient ring such as $R[x]/(x^n - 1)$ so that the expressions cannot blow up.

Again, an exponential speed-up over the standard method.

But Beware of Hasty Conclusions

But we cannot conclude that $f^t(x)$ can always be computed in $O(\log t)$ operations.

The reason fast exponentiation and the examples above work is that we can explicitly compute a representation of $f \circ f$, given the representation of f.

But, in general, there is no fast representation for $f \circ f$, we just have to evaluate f twice.

Just think of f as being given by a piece of C code. We can produce another piece of C code that computes f^2, and more generally for f^t, but the code just evaluates f t-times, in the obvious brute-force way.

Exercise

Ponder deeply. Assume the speed-up trick always works and figure out what that would mean for complexity theory.

Hasty Conclusion I

Speaking about hasty conclusions, here is a simple inductively defined sequence of integers.

$$a_1 = 1$$
$$a_n = a_{n-1} + (a_{n-1} \bmod 2n)$$

Thus, the sequence starts like so:

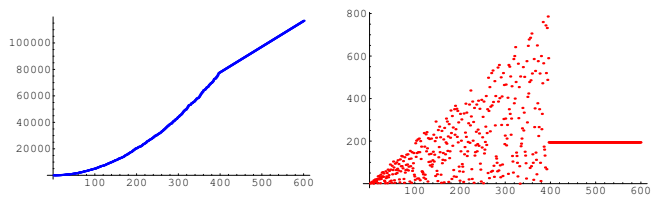
$$1, 2, 4, 8, 16, 20, 26, 36, 36, 52, 60, 72, 92, 100, 110, 124, 146, 148, 182, 204$$

This seems rather complicated. The function appears to be increasing in a somewhat complicated manner.

Alas, there is a rude surprise.

Ultimately Linear

The sequence is ultimately linear: $a_{396+k} = a_{396} + k \cdot 194$ for $k \geq 0$.



The plot on the left is the sequence, on the right (in red) are the forward differences.

Exercise

Figure out why the sequence is ultimately linear.

Hasty Conclusion II

Here is another strange integer sequence:

$$a_n = \lceil 2/(2^{1/n} - 1) \rceil - \lfloor 2n/\log 2 \rfloor$$

This time, the sequence starts like so:

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots$$

and continues like this for a long, long time, for trillions of terms.

Note that it is not so easy to compute the terms. At any rate, it sure looks like the sequence is constant 0. Alas

$$a_{777\,451\,915\,729\,368} = 1$$

Iteration versus Recursion

Iteration can be construed as a special case of recursion.

$$F(0, y) = y$$
$$F(x + 1, y) = f(F(x, y))$$

Then $F(x, y) = f^x(y)$.

Conversely, iteration can be used to express recursion. Suppose

$$\begin{aligned} f(0, \boldsymbol{y}) &= g(\boldsymbol{y}) \\ f(x + 1, \boldsymbol{y}) &= h(x, f(x, \boldsymbol{y}), \boldsymbol{y}) \end{aligned}$$

is defined by primitive recursion.

As Iteration

Define a function $H : \mathbb{N} \times \mathbb{N} \times \mathbb{N}^k \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}^k$ by

$$H(x, z, \boldsymbol{y}) = (x + 1, h(x, z, \boldsymbol{y}), \boldsymbol{y})$$

Then

$$\mathsf{snd}(H^x(0, g(\boldsymbol{y}), \boldsymbol{y})) = f(x, \boldsymbol{y})$$

This is the natural definition, but if we wanted to we could make H unary by coding everything up as a sequence number.

Unary Iteration

More precisely, suppose we have some simple basic functions such as

$$x + y \quad x * y \quad x \dot{-} y \quad \mathsf{rt}(x)$$

Here $\mathsf{rt}(x)$ is the integer part of $\sqrt{x}$. These suffice to set up coding machinery, which can then be used to replace recursion by iteration. It suffices to define functions via

$$f(x) = g^x(0)$$

to get the same class as from the recursions above.

Exercise

Come up with a precise version of this statement and give a detailed proof.

Trajectories and Orbits

Definition

The trajectory or orbit of $a \in A$ under f is the infinite sequence

$$\mathsf{orb}_f(a) = a, f(a), f^2(a), \ldots, f^n(a), \ldots$$

The set of all infinite sequences with elements from A is often written A^ω. Hence the we can think of the trajectory as an operation of type

$$(A \to A) \times A \to A^\omega$$

that associates a function on A and element in A with an infinite sequence over A.

Terminology Warning

Sometimes one is not interested in the actual sequence of points but rather in the set of these points:

$$\{ f^i(a) \mid i \geq 0 \}$$

While the sequence is always infinite, the underlying set may well be finite, even when the carrier set is infinite.

In a sane world one would refer to the sequences as trajectories, and use the term orbit for the underlying sets. Alas, in the literature the two notions are hopelessly mixed up.

So, when we refer to a "trajectory" we will always mean the sequence, but, bending to custom, we will use "orbit" for both.

Digression: Dedekind's Ketten (Chains)

Here is a clever definition due to Dedekind: given an endofunction f and a point a, the corresponding chain is defined to be

$$\bigcap \{ X \subseteq A \mid a \in X, f(X) \subseteq X \}$$

Thus, the chain is the least set that contains a and is closed under f. That is exactly the orbit of a under f, considered as a set.

Who cares?

Dedekind's definition does not require the natural numbers. In fact, it can be used to define them. In Dedekind's view, this means that arithmetic can be reduced to logic.

From Chains to Naturals

Here is how. Suppose we have a function $f : A \to A$ and a point $a \in A$ such that

- f is an injection,
- a is not in the range of f,
- A is the chain of f and a.

Dedekind calls these sets simply infinite.

We can think of a as 0 and, more generally, we can think of $f^n(a)$ as n.

So this is a way of describing the natural numbers, the smallest infinite set, without any hidden references to the naturals.

The Price: Impredicativity

According to Dedekind, the chain C defined by f and a has the form

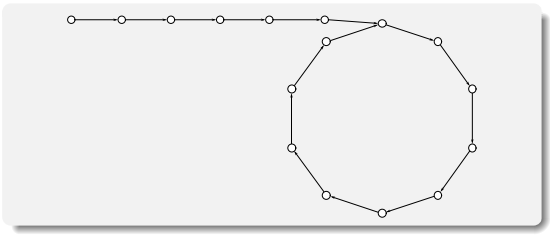
$$C = \bigcap \{ X \subseteq A \mid a \in X, f(X) \subseteq X \}$$

But note that C is one of the X's on the right hand side. So there is some (non-vicious) circularity in this approach. Most mathematicians would not bat an eye when confronted with definitions like this one, they are totally standard.

And the payoff is huge. For example, when Bernstein told Dedekind about his correct proof of the "Cantor-Schröder-Bernstein" theorem, he was shocked to hear that Dedekind had a much better proof, based on his chains.

The Lasso

At any rate, if the carrier set is finite, all trajectories must ultimately wrap around and all orbits must be finite:



What changes is only the length of the transient part and the length of the cycle (in the picture 6 and 10).

Fixed Points, Cycles and Periods

Definition

Let $f : A \to A$ be an endofunction.

- $a \in A$ is a fixed point of f if $f(a) = a$.
- A sequence $a_0, \ldots, a_{n-1}$ in A is a cycle of f if $f(a_i) = a_{i+1 \bmod n}$.
- A cycle of length n is also called an n-cycle.
- The orbit of a under f is periodic if $\exists p > 0 : f^p(a) = a$.
- The orbit of a under f is ultimately periodic $\exists t \geq 0, p > 0 : f^{t+p}(a) = f^t(a)$.

Cycles and fixed points are closely related:

$a_0, \ldots, a_{n-1}$ is an n-cycle of f iff a_0 is a fixed point of f^n.

Transient and Period

If A is finite, then any orbit of $f : A \to A$ must be ultimately periodic:

$$f^t(x) = f^{t+p}(x)$$

for some $t \geq 0$, $p > 0$, which values depend on x.

Definition

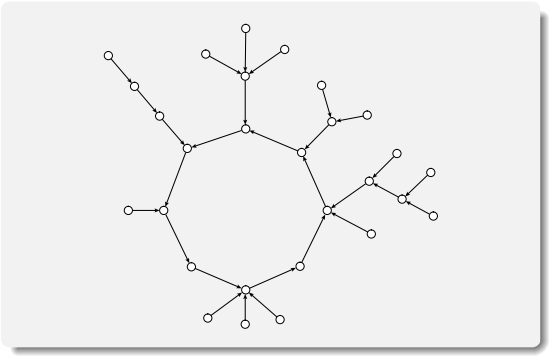
The least t and p such that $f^t(x) = f^{t+p}(x)$ is the transient length and the period length of the orbit of x (wrt. f).

Thus, an orbit is periodic iff the transient is 0.

Also, a function on a finite set has only transients of length 0 iff the function is injective iff it is a permutation.

Limit Cycles

The lasso shows the general shape of any single orbit, but in general orbits overlap. All orbits with the same limit cycle are called a basin of attraction in dynamics.



The Functional Digraph

As the last picture shows, it is natural to think of f as a directed graph on the carrier set where the edges indicate the action of f.

Definition

The functional digraph (or diagram) of $f : A \to A$ is defined as $G_f = \langle A, E \rangle$ where $E = \{ (x, f(x)) \mid x \in A \}$.

Note that every vertex in G_f has outdegree 1, but indegrees may vary.

The non-trivial strongly connected components of the digraph are the limit cycles of the function. The weakly connected components are the basins of attraction.

Analyzing the Diagram

There are several natural parameters associated with the digraph that provide useful information about the function in question.

- Indegree. If all nodes have the same indegree k the function is k-to-1. Otherwise, determine the maximum/minimum indegree, the distribution of values.
- Periods. Count the number of limit cycles, and their length.
- Transients. Determine the length of the transients leading to limit cycles.

At least when the carrier set is finite we would like to be able to determine these parameters easily. Alas, even for relatively simple maps this turns out to be rather difficult.

Reachability

The geometric perspective afforded by the diagram also suggests to study path-existence problems.

Definition

Let f be a function on A and $a, b \in A$ two points in A. Then point y is reachable from x if for some $i \geq 0$:

$$f^i(x) = y.$$

In other words, point y belongs to the orbit of x.

Proposition

Reachability is reflexive and transitive but in general not symmetric.

Reachability is symmetric when A is finite and f injective (and therefore a permutation): each orbit then is a cycle and forms an equivalence class.

Confluence (aka Basins of Attraction)

Definition

Let f be a function on A and $a, b \in A$ two points in A. Points a and b are confluent if for some $i, j \geq 0$:

$$f^i(a) = f^j(b).$$

In other words, the orbits of a and b merge, they share the same limit cycle (which may be infinite and not really a cycle).

Reachability implies confluence but not conversely. For finite carrier sets reachability is the same as confluence iff the map is a bijection.

Confluence is an Equivalence

Proposition

Confluence is an equivalence relation.

Reflexivity and symmetry are easy to see, but transitivity requires a little argument.

Let $f^i(x) = f^j(y)$ and $f^k(y) = f^l(z)$, assume $j \leq k$. Then with $d = k - j \geq 0$ we have

$$f^{i+d}(x) = f^{j+d}(y) = f^k(y) = f^l(z).$$

Each equivalence class contains exactly one cycle of f, and all the points whose orbits lead to this cycle – just as in the last picture.

Small Example

Here is a somewhat frivolous operation on binary lists: given L, replace the first element of L by 0, and then rotate to the left by 2 places.

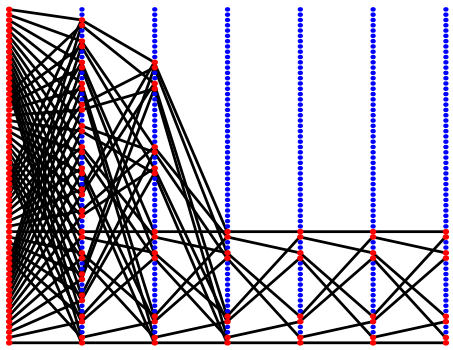
Here is the orbit of a generic list (with symbolic entries) of length 6 under this operation:

$$\begin{array}{ll} 0: & x_1\ x_2\ x_3\ x_4\ x_5\ x_6 \\ 1: & x_3\ x_4\ x_5\ x_6\ 0\ x_2 \\ 2: & x_5\ x_6\ 0\ x_2\ 0\ x_4 \\ 3: & 0\ x_2\ 0\ x_4\ 0\ x_6 \\ 4: & 0\ x_4\ 0\ x_6\ 0\ x_2 \\ 5: & 0\ x_6\ 0\ x_2\ 0\ x_4 \\ 6: & 0\ x_2\ 0\ x_4\ 0\ x_6 \end{array}$$

So both transient and period are 3 in the generic case. But note the for special values of x_2, x_4 and x_6 the period may be shorter (ditto for the transient and x_1, x_3 and x_5).

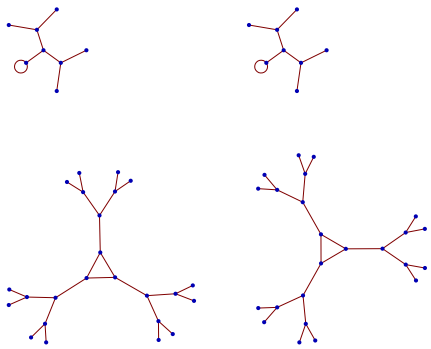
The following picture shows the behavior of all binary lists of length 6 under this operation.

All Orbits

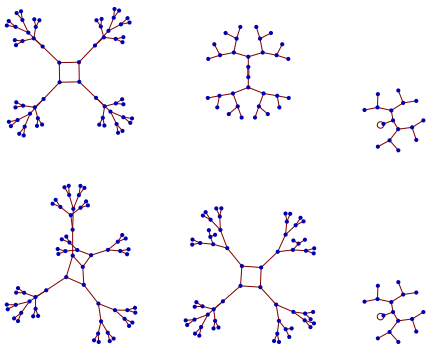


There are two fixed points, and two 3-cycles.

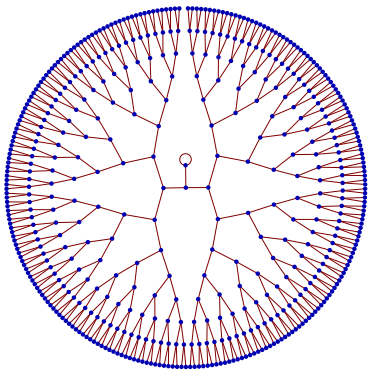
Another View



$n = 8$

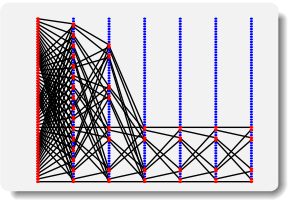


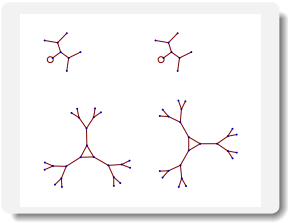
By Contrast $n = 9$



Two Pictures, One Object

The two pictures





represent the same functional digraph, albeit from a different perspective.

Depending on what one is interested one or the other may be preferable.

Example: Exclusive Or

Here is a slightly more ambitious example, though the analysis turns out still to be fairly easy in this case. Consider the map

$$f : \mathbf{2}^n \to \mathbf{2}^n$$

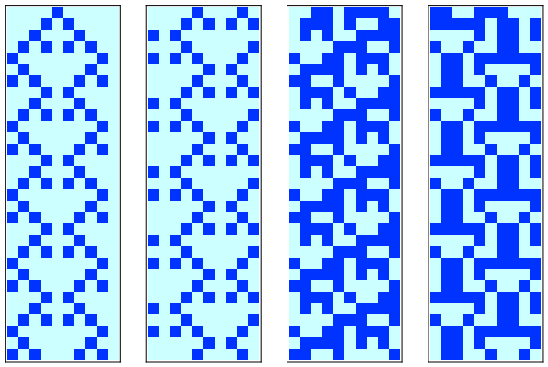
defined by

$$f(\boldsymbol{x}) = L(\boldsymbol{x}) \operatorname{xor} R(\boldsymbol{x})$$

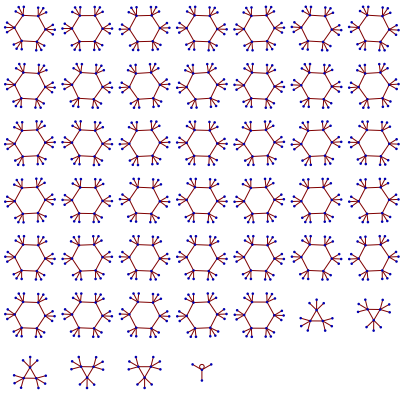
where L and R denote cyclic left- and right-shift, respectively, and xor is bitwise exclusive or.

E.g., for $n = 10$ we have $f(0,0,0,1,1,1,0,0,0,0) = (0,0,1,1,0,1,1,0,0,0)$

Some Orbits, $n = 10$



The Whole Diagram



The Parameters

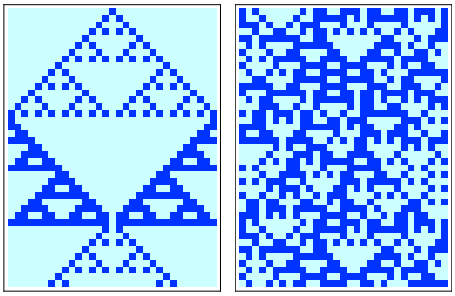
The diagram is highly uniform in this case and can easily be described in terms of the general parameters.

- Every node has indegree 4.
- There are 40 limit cycles of length 6, 5 limit cycles of length 3 and one fixed point.
- The transient lengths for all points not on a limit cycle is 1.

Alas, this is not really interesting. What we really would like to have is an answer in terms of the parameter n.

This is just the special case $n = 10$.

Some Orbits, $n = 31$



A Battle Plan

How would one go about answering this question?

We need at the very least

- a program that takes as input an n-bit pattern and computes transient and period, and
- a program that takes as input n and determines the whole diagram.

Why is the second item listed separately?

Phase Space

Concretely, suppose you have a Boolean function

$$f : \mathbf{2}^{24} \to \mathbf{2}^{24}$$

and you want to compute all transients and periods.

The space in question is $A = \mathbf{2}^{24}$, about 16 million elements.

Exercise

- *What is a plausible algorithm?*
- *What if we had to deal with 32 bits instead, with 64?*

Fixed Points

Fixed points are particularly interesting. Quite a few computational tasks can be rephrased as a fixed point computation. In the right environment this approach produces very elegant algorithms.

Definition

Let $f : A \to A$ and $a \in A$. Write $\mathrm{FP}(f, a)$ for the fixed point on the orbit of a under f if it exists (undefined otherwise).

So our claim is that there are lots of examples where an algorithm boils down to computing $\mathrm{FP}(f, a)$ for the right choice of f.

Note that most current programming language do not directly support the operation FP, though some like Mathematica do.

Binary Expansions

Here is a more utilitarian example. Define the following operation on numbers and binary lists.

$$\begin{aligned} &f : \mathbb{N} \times \mathsf{List}(\mathbf{2}) \to \mathbb{N} \times \mathsf{List}(\mathbf{2}) \\ &f(0, L) = (0, L) \\ &f(x, L) = (x \text{ div } 2, \mathsf{prep}(L, x \bmod 2)) \end{aligned}$$

Then $\mathrm{FP}(f, (x, \mathsf{nil})) = (0, \mathrm{bin}(x))$ where $\mathrm{bin}(x)$ is the binary expansion of x.

In the right environment, this provides a one-liner for conversion:

$$\mathsf{tobin}(x) = \mathsf{last}(\mathrm{FP}(f, (x, \mathsf{nil})))$$

Experimental, Interactive Computing

The one-liner may not seem particularly impressive.

In fact, when constructing large programs it may be preferable to use lots of simple operations, rather than somewhat cryptic and complicated "primitives" such as FP. Kenneth Iverson's APL from the 1960s is a perfect example of a language that tends to produce "write-only" code.

But for quick-and-dirty one-shot computations this is the way to go.

In experimental computing it is important to get results quickly so one try out various possibilities – there is no time to write, debug and compile a complicated program.

One has to rely on an expressive language (including for example list manipulation primitives) together with a large base of algorithms (such as integration, factorization, ...).

Application: Transitive Closure

A classical computational problem is to determine the transitive reflexive closure $\rho^\star$ of a given binary relation $\rho \subseteq A \times A$, A finite.

One way to tackle this is by thinking of ρ as a digraph and use standard graph algorithms. Another is to apply iteration.

Define the square of a relation

$$\mathrm{squ}(\rho) = \rho^2 = \rho \bullet \rho$$

to be the composition of the relation with itself.

Lemma

If ρ is reflexive, then $\mathrm{FP}(\mathrm{squ}, \rho)$ is the transitive reflexive closure of ρ.

Efficiency?

If the carrier set has size n then $\rho^\star = \rho^t$ where $t \leq n-1$. Hence we have to iterate the squaring operation at most $\log_2 n$ times.

How does one implement the squaring operation? If we use Boolean matrices to represent the relations the relational composition comes down to matrix multiplication, a cubic operation with standard algorithms. The whole computation is $O(n^3 \log n)$, slightly underwhelming.

Note, though, for small n, say, $n \leq 256$, one can exploit hardware to make this very fast.

Application: Equivalence Relations

Recall that an equivalence relation E on A can be represented by a selector function $f : A \to A$ such that E is the kernel relation defined by f:

$$x \mathrel{E} y \iff f(x) = f(y)$$

Computationally this is advantageous since we can use a simple array to represent f:

This means we can test $x \mathrel{E} y$ in $O(1)$ steps, and the data structure has size $O(n)$ (assuming, of course, that the elements in the carrier set are constant size).

Exercise

Why is every equivalence relation on A a kernel relation of an endofunction on A?

Union/Find

A more challenging problem is to deal with dynamic equivalence relations: the relation changes over time as more and more equivalent pairs of elements are discovered. The (static) canonical selector function is not well-suited to this situation.

Instead, we use a function $f : A \to A$ such that

$$x \mathrel{E} y \iff \mathrm{FP}(f, x) = \mathrm{FP}(f, y).$$

Thus, an equivalence class consists of one basin of attraction of f.

A priori equivalence testing is just $O(n)$: the transients might be very long. Each tree represents an equivalence class and is directed towards the root.

View From a Tree

Intuitively, we build a collection of trees, initially all consisting of a single node: initially, $f(x) = x$ for all x.

Here is a primitive version (without ranking and path compression). For every new pair (a, b):

- Compute $a_0 = \mathrm{FP}(f, a)$ and $b_0 = \mathrm{FP}(f, b)$.
- If $a_0 = b_0$, do nothing.
- Otherwise, set $f(b_0) = a_0$.

Clearly maintains the fixed point property.

By adding ranked union and path compression we can keep the transients so small that the whole algorithm is essentially linear in the number of operations.

Exercise

Read up on the details of the Union/Find algorithm.

Clever Heuristics

There are two hackish improvements to make this algorithm enormously efficient.

- Ranked union: attach the shallower tree to the deeper one.
- Path compression: attach all elements in the branches to p and q directly to the new root.

Theorem

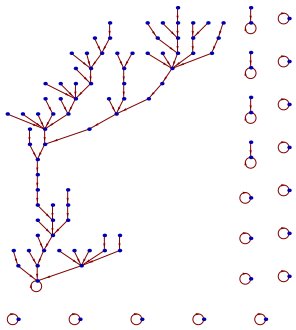
The Union/Find algorithm has essentially linear running time.

The running time is $O(m\,\alpha(n))$ where m is the number of operations and n the size of the carrier set. Here α is the "inverse" of the Ackermann function:

$$\alpha(n) = \min\big(\, k \mid n \leq A(k,k) \,\big)$$

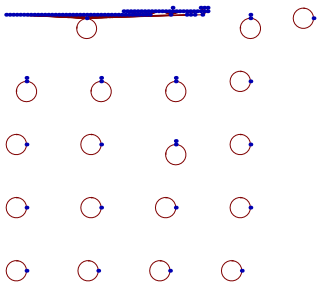
So, in theory $\alpha(n)$ tends to infinity – but in the real world $\alpha(n)$ is a small constant.

Union/Find Example



Carrier set $[100]$, 100 randomly generated pairs, no ranking, no path compression.

Union/Find Example



Carrier set $[100]$, 100 randomly generated pairs, with ranking and path compression.

Decimal Fractions

Computations that end when a fixed point has been reached are particularly elegant, but sometimes one has to to stop after going around the limit cycle for the first time. Note that this makes it quite a bit more difficult to decide when to stop.

A classical example: conversion of a rational number $0 < x < 1$ into a decimal fraction. Since $x = \sum_{i \geq 1} d_i \cdot 10^{-i}$ where $0 \leq d_i \leq 9$ we have the following conversion algorithm:

$$f : \mathsf{List}(\mathbb{N}) \times \mathbb{Q} \to \mathsf{List}(\mathbb{N}) \times \mathbb{Q}$$
$$f(L, x) = (\mathsf{app}(L, \mathsf{floor}(10x)), \mathsf{fract}(10x))$$

Stop when $\mathsf{last}(f^i(\mathsf{nil}, x))$ assumes the same value twice (i.e., we are on the limit cycle).

Example

For $x = \frac{1}{123}$ we get

$$\frac{1}{123}, \frac{10}{123}, \frac{100}{123}, \frac{16}{123}, \frac{37}{123}, \frac{1}{123}, \ldots$$

leading to digits

$$0.00813\,00813\,00813\ldots$$

So the transient is 0 and the period is 5 in this case.

A Long Cycle

But for $x = \frac{1}{1234}$ there is a transient of 1, and a period of 88.

$$\frac{1}{1234}, \frac{5}{617}, \frac{50}{617}, \frac{500}{617}, \frac{64}{617}, \frac{23}{617}, \frac{230}{617}, \frac{449}{617}, \frac{171}{617}, \frac{476}{617},$$
$$\frac{441}{617}, \frac{91}{617}, \frac{293}{617}, \frac{462}{617}, \frac{301}{617}, \frac{542}{617}, \frac{484}{617}, \frac{521}{617}, \frac{274}{617}, \frac{272}{617},$$
$$\frac{252}{617}, \frac{52}{617}, \frac{520}{617}, \frac{264}{617}, \frac{172}{617}, \frac{486}{617}, \frac{541}{617}, \frac{474}{617}, \frac{421}{617}, \frac{508}{617},$$
$$\ldots$$
$$\frac{76}{617}, \frac{143}{617}, \frac{196}{617}, \frac{109}{617}, \frac{473}{617}, \frac{411}{617}, \frac{408}{617}, \frac{378}{617}, \frac{78}{617}, \frac{163}{617},$$
$$\frac{396}{617}, \frac{258}{617}, \frac{112}{617}, \frac{503}{617}, \frac{94}{617}, \frac{323}{617}, \frac{145}{617}, \frac{216}{617}, \frac{309}{617}, \frac{5}{617}$$

A Wild Iteration

We have already seen that iteration can produce very rapidly growing functions (much like recursion). Here is another example where iteration produces a rather perplexing result: every orbit ends in fixed point 0, though it looks like it should diverge towards infinity.

Suppose we write a number in base 2, say

$$266 = 2^8 + 2^3 + 2$$

We can turn this into the complete binary expansion by writing the exponents also in base 2, and so on.

$$266 = 2^{2^{2+1}} + 2^{2+1} + 2$$

where we really should write 2^0 instead of 1, but c'mon.

Base Bump

Now suppose we replace 2 in the representation everywhere by 3:

$$3^{3^{3+1}} + 3^{3+1} + 3$$

Unsurprisingly, this new number is much larger:

$$443426488243037769948249630619149892887 \approx 4 \times 10^{38}$$

Next, we write this number in complete base 3, and bump the base to 4. We get something like 3×10^{616}.

Then we write this number in complete base 4 and bump to 5 ...

Goodstein's Theorem

Obviously, this process leads to a very rapidly increasing sequence of numbers.

Now suppose we follow the base bump by subtracting 1, so the result will be a tiny little bit smaller than with a pure base bump. Call such a sequence a Goodstein sequence.

We expect Goodstein sequences to diverge since the base bump causes a huge increase, subtracting 1 should really not matter much. Alas ...

Theorem

All Goodstein sequences converge to 0.

Example?

Alas, it is very hard to come up with good examples.

Starting at 3 we get the sequence

$$3, 3, 3, 2, 1, 0$$

Starting at 4 we get the sequence

$$4, 26, 41, 60, 83, 109, 139, 173, 211, 253, 299, \ldots$$

It takes some $10^{121,210,695}$ steps to get to 0!

The proof of Goodstein's theorem uses ordinals and cannot be handled in Peano arithmetic. But note that the stopping time of these sequences is clearly computable: just do the arithmetic. Computable functions can be monsters.

Calculating Transients and Periods

How do we compute the transient t and period p of the orbit of $a \in A$ under $f : A \to A$ for finite carrier sets A?

The obvious brute force approach is to use a container to keep track of everything we have already seen:

$$a, f(a), f^2(a), \ldots, f^i(a)$$

and then to compare $f^{i+1}(a)$ to all these previous values.

In most cases, the data structure of choice is a hash table or tree: we can check whether $f^{i+1}(a)$ is already present in expected constant time or logarithmic time, respectively. Memory requirement is linear in the size of the orbit assuming the elements in A require constant space (a fairly safe assumption, if the elements are big use pointers).

Floyd's Trick

A (simplified version of a) classical problem from the early days of Lisp: Suppose we have a pointer-based linked list structure in memory and we want to check if there are any cycles in the structure (as opposed to having all lists end in `nil`).

We can think of this as an orbit problem:

- A is the set of all nodes of the structure,
- $f(x) = y$ means there is a pointer from x to y.

The Problem:

Suppose further the structure consumes 90% of memory, so we cannot afford to build a large hash table or tree.

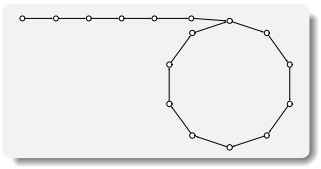
Can we compute transients and periods in $O(1)$ space?

A Memoryless Approach

At first glance, this may seem quite impossible: if we forget already discovered elements we *obviously* cannot detect cycles.

Nonetheless, the following code finds a point on the cycle in the orbit of a.

```
u = f(a);
v = f(u);
while( u != v ) {
    u = f(u);
    v = f(f(v));
}
```



Claim

Upon termination, $u = v$ is a position on the cycle.

Moving Pebbles

Think of two pebbles u and v, moving at speed 1 and 2, respectively.

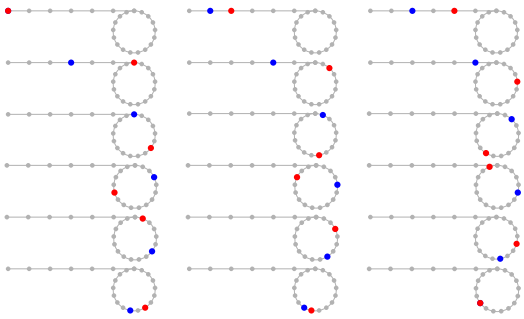
The slow pebble u enters the limit cycle at time t, the transient, when the fast pebble v is already there. From now on, v gains one place on u at each step. So pebble v must catch up at time s where $s \leq t + p$, where p is the period.

Also note that once we have our foothold on the cycle, we can compute the period: run around the cycle one more time, counting.

One can make a nice movie out of this. OK, it is pretty boring after all, but what do you expect.

Example

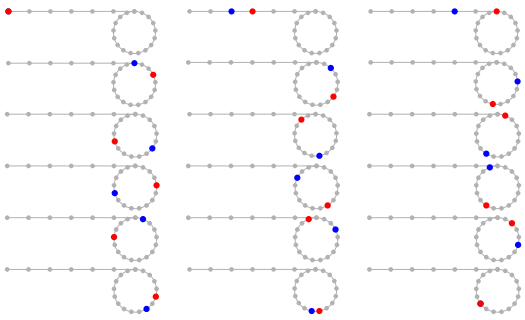
Here the transient is 6, and the period 17.



The pebbles meet at time 17.

Example

Same transient and period, but pebble speeds 2 and 3, respectively.



Again, the pebbles meet at time 17.

How about the Transient?

Suppose we already know p, the period.

```
t = 0;
u = a;
v = iterate( f, a, p );   // v = f^p(a)
while( u != v ) {
  u = f(u);
  v = f(v);
  t++;
}
```

Claim

Upon completion, t is the transient length.

Proof. v is exactly p places ahead of u. So, when u first enters the cycle, v has just gone around once, and they meet. □

Floyd's Trick

Let us assume f to be computable in time $O(1)$ and elements of the carrier set A to take space $O(1)$.

Theorem

One can determine the transient t and period p of a point in A under f in time $O(t+p)$, and space $O(1)$.

Linear time cannot be avoided in general (why?), so this is optimal.

Exercise

The choice of speeds 1 and 2 for the pebbles in Floyd's algorithm is natural, but there are other possibilities. Discuss other choices.

Beware of Permutations

Floyd's cycle finding algorithm is an excellent general purpose tool in particular when the evaluation of the function in question is cheap.

But note that in the special case where the function is known to be a permutation on a finite domain there is, of course, no need to use Floyd's or similar cycle finding algorithms: since the components of the diagram are all cycles we can simply trace a cycle once to determine its length. So the natural method to compute cycle length is automatically memoryless (if we assume the objects in question can be stored in constant space).

Incidentally, determining cycle lengths of permutations is very important for some advanced counting methods, more later.

The Real Challenge

Typically we are not interested in just one map $f : A \to A$ but in a whole family of maps. For example, we may want

- $A = \{0, 1, \ldots, n-1\}$
- $A = \{1, 2, \ldots, n\}$
- $A = \mathbf{2}^n$
- $A = \Sigma^n$
- $A =$ full binary trees on n leaves

We need a description that holds for all values of n, preferably in closed form.

Here are some simple examples.

Modular Addition

Consider the additive function

$$f_k : \mathbb{Z}_n \to \mathbb{Z}_n$$
$$x \mapsto x + k \bmod n$$

f_k is clearly injective, so the orbits are all cycles.

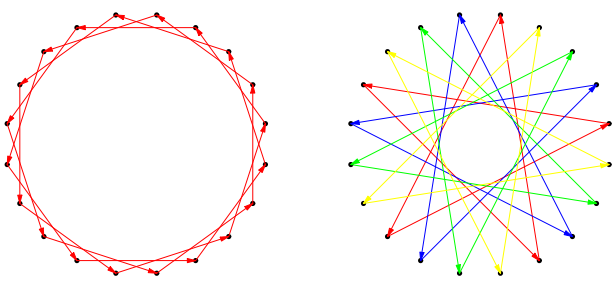
Moreover, since $f_k(x+d) = f_k(x) + d \pmod n$ all the cycles must have the same length.

So how many orbits are there?

Proposition

f_k has $\gcd(n,k)$ distinct orbits, each of length $n/\gcd(n,k)$.

$n = 20$



The stride is $k = 3$ on the left, $k = 8$ on the right.

How About Multiplication?

Can we come up with a similar analysis for the multiplicative function

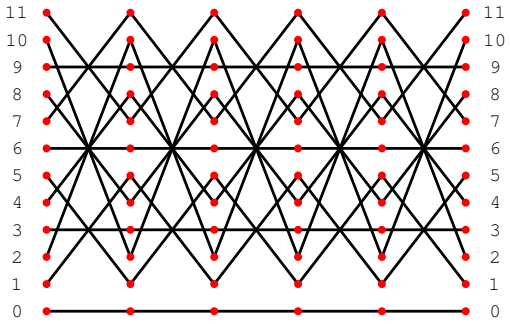
$$g_k(x) = k \cdot x \bmod n$$

One should expect somewhat greater difficulties here: g_k is not injective in general, so the orbits will have transients.

Moreover, the orbits cannot simply be translated into each other, not even the periodic parts.

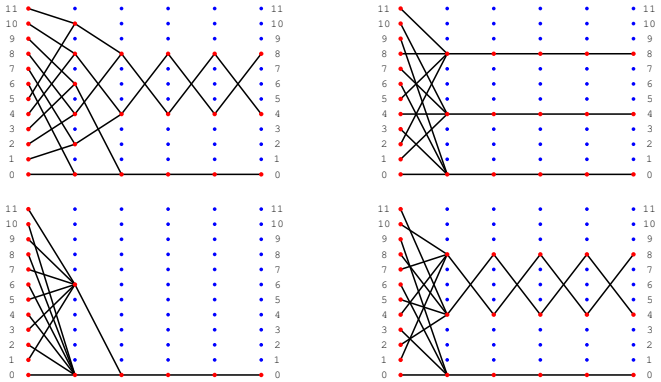
A complete description of the digraph of g_k will be much more complicated than in the additive case.

$n = 12,\ k = 5$



There are 4 fixed points (0, 3, 6, 9) and 4 2-cycles ($(1,5)$, $(2,10)$, $(4,8)$, $(7,11)$).

$n = 12,\ k = 2, 4, 6, 8$



The Easy Case

Suppose n and k are coprime. In this case g_k is injective, all orbits are periodic, all transients are 0.

That means that for all x there is some $p > 0$:

$$x = k^p \cdot x \pmod{n}$$

For this to hold it suffices that $k^p = 1 \pmod{n}$, so the multiplicative order of k in $Z_n^\star$ is an upper bound for the period p. But this bound is not tight, the choice of x also plays a role.

Exercise

Figure out the details.

Example: Riffle Shuffle

Start with an even number of cards, split the deck in half, and then interleave the two halfs (perfect shuffle, alternate one card from each half-deck).

E.g., starting with the deck $[20]$ one obtains

$$11, 1, 12, 2, 13, 3, 14, 4, 15, 5, 16, 6, 17, 7, 18, 8, 19, 9, 20, 10$$

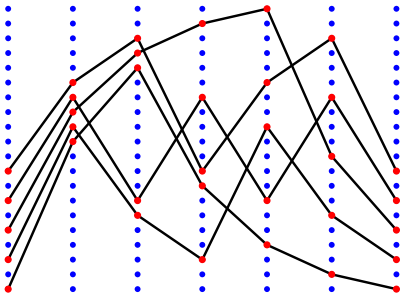
This clearly is a permutation (no cards disappear or are added). Hence, all transients are 0, and we get the cycle decomposition

$$(1, 11, 16, 8, 4, 2), (3, 12, 6), (5, 13, 17, 19, 20, 10), (7, 14), (9, 15, 18)$$

After six riffle shuffles we are back to the original deck of cards: the least common multiple of 6, 3, 6, 2, and 3 is 6.

The Orbits

This is also clear to see when we trace the orbit of the first point in each cycle.

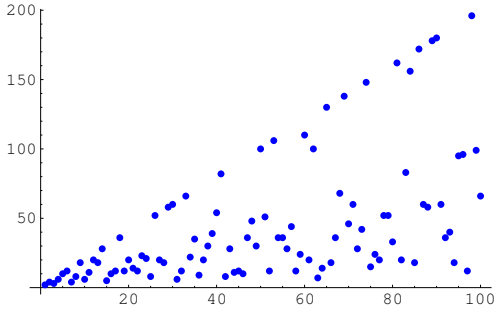


Note that the inverse map is a bit easier to understand: it boils down to multiplication by 2, modulo 21.

Various Deck Sizes

For any n, there must be some number k such that k repetitions of riffle shuffle on $2n$ cards return the deck to its original state.

How does k depend on n?



The relationship is a bit complicated, and we will not pursue the issue here.