CDM

Induction

Klaus Sutner

Carnegie Mellon University

The Concept

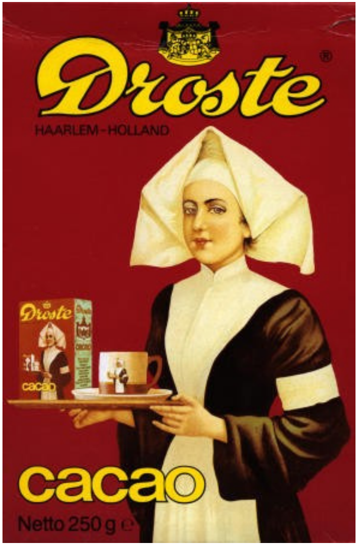
Recall that we are dealing with entities that are similar to a part of themselves.

This notion has striking examples in geometry, programming, and discrete mathematics.

- In geometry the use of recursion has lead to a whole new class of objects, in particular self-similar figures and fractals. As it turns out, these are just as important in describing real world phenomena as classical Euclidean objects.
- Recursion in programming is a standard tool that often leads to elegant, concise solutions of otherwise messy problems. Programming languages themselves are often defined in terms of recursive rules.
- Many objects in discrete mathematics are defined by recursion and are manipulated by operations that are similarly defined by recursion. Recursive equations are a central tool in many combinatorial problems.

First a few geometric examples.

The Droste Effect

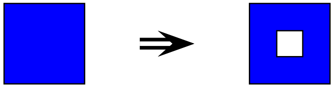


Recursion in Geometry

Roughly, we

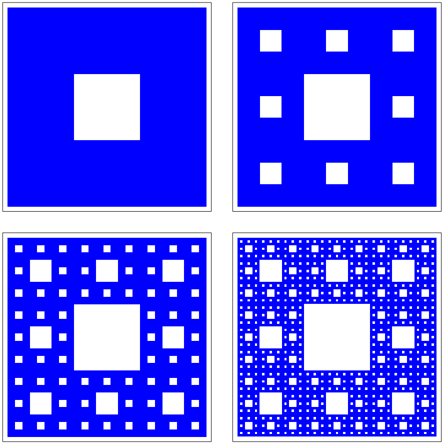
- start with a simple geometric figure (the axiom),
- replace the figure by a new one, which consists of several scaled-down copies of the original (the generator).

In the limit often produces self-similar fractals. In the following example, the axiom is a square and the generator is a punctured square:



Note that the generator can be thought of as consisting of 8 squares of size $1/3$ the original.

Sierpinski Carpet



Incidentally, this type of pattern is useful in antenna design.

Taking Things to the Limit

In the continuous world of geometry we actually do not need a halting condition, the replacement operation can continue indefinitely (at the very least ω steps).

For example, for the Sierpinski carpet there is a clear sense in which the nth generation set S_n converges to a limit, as n tends to infinity.

In fact, in this particular case

$$\lim S_n = \bigcap S_n \subseteq [0,1]^2$$

The properties of these limit sets are quite fascinating and their systematic study was one of the major accomplishments of 20th century geometry.

For example, one interesting consequence of this recursive construction is that, in the limit, it often leads to figures with non-integral dimension.

Non-Integral Dimension

To see why, consider the following method to measure dimension of a classical geometric figure such as a line, square or cube: count the number of copies of the figure, scaled down by $1/2$, needed to cover the original figure. For lines, squares and cubes these counts are 2, 4, 8, respectively.

The dimensions are therefore $\log_2 2 = 1$, $\log_2 4 = 2$ and $\log_2 8 = 3$.

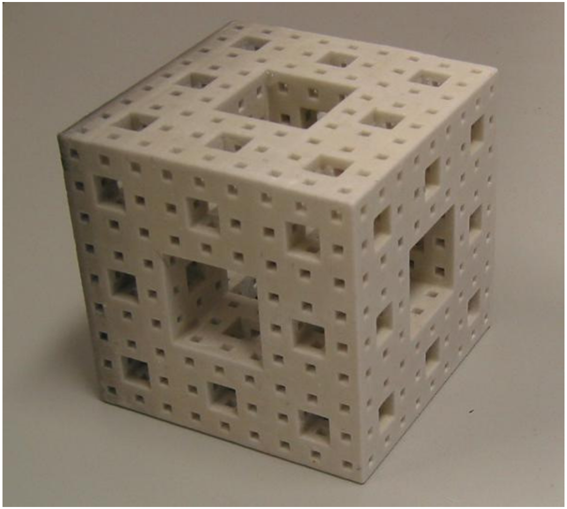
Of course, the shrink factor 2 is arbitrary, in general we shrink by $1/k$ and consider $\log_k \#$copies. For lines, square and cubes this makes no difference.

For the Sierpinski carpet we get dimension

$$\log_3 8 \approx 1.89$$

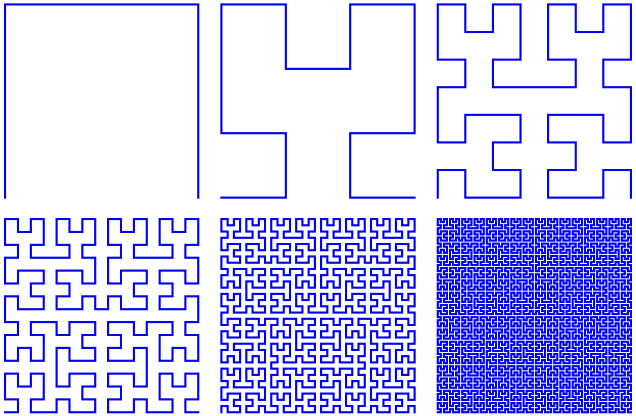
This makes good intuitive sense: the Sierpinski carpet is not as massive as a square, but has significantly higher dimension than a simple line. For a more careful discussion of non-integral dimensions see Hausdorff-Besicovic.

Menger Sponge



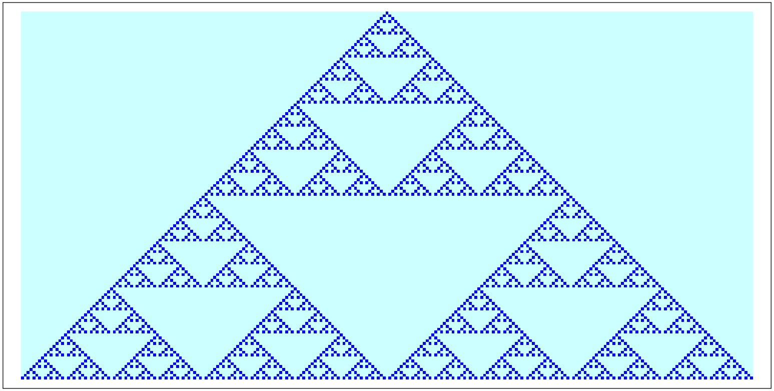
Note that is not entirely clear how one would go about building a 3-D model (this one was done on a ZCorp Z406 rapid prototyping machine).

A Hilbert Curve



In the limit, this curve fills the whole unit square. As a consequence, the unit interval has the same cardinality as the unit square.

A Cellular Automaton



This is elementary cellular automaton number 90 with local rule $\rho(x, y, z) = x + z \bmod 2$. The pattern has dimension $\log_2 3 \approx 1.59$.

Towers of Hanoi

Here is classic (though somewhat tired) example of recursion: 64 golden disks of different sizes are stacked on three diamond pegs. Unspecified monks are trying to move the disks from one peg to another, one disk at a time, without ever placing a larger disk on top of a smaller. A slightly more affordable version is shown below.

The Recursive Solution

Everyone knows how to solve this problem recursively. Let $n =$ number of disks. The case $n = 1$ is trivial; just move one disk, done.

The general case $n > 1$ can be reduced to $n - 1$. Here is the method for moving from peg a to c.

- Move $n - 1$ disks from peg a to peg b.
- Move nth disk from peg a to peg c.
- Move $n - 1$ disks from peg b to peg c.

Note that correctness is a bit subtle: moving the $n - 1$ small disks in a game of size n really corresponds to a game of size $n - 1$: the recursive approach works because the game itself carries an inductive structure.

Computationally speaking, recursion, for all its elegance and simplicity, is a big gun. One might wonder if there is a non-recursive solution. More generally, what is the simplest program to compute the moves?

A Run

peg 1	peg 2	peg 3	peg 1	peg 2	peg 3
5, 4, 3, 2, 1				5	4, 3, 2, 1
5, 4, 3, 2	1		1	5	4, 3, 2
5, 4, 3	1	2	1	5, 2	4, 3
5, 4, 3		2, 1		5, 2, 1	4, 3
5, 4	3	2, 1	3	5, 2, 1	4
5, 4, 1	3	2	3	5, 2	4, 1
5, 4, 1	3, 2		3, 2	5	4, 1
5, 4	3, 2, 1		3, 2, 1	5	4
5	3, 2, 1	4	3, 2, 1	5, 4	
5	3, 2	4, 1	3, 2	5, 4, 1	
5, 2	3	4, 1	3	5, 4, 1	2
5, 2, 1	3	4	3	5, 4	2, 1
5, 2, 1		4, 3		5, 4, 3	2, 1
5, 2	1	4, 3	1	5, 4, 3	2
5	1	4, 3, 2	1	5, 4, 3, 2	
5		4, 3, 2, 1		5, 4, 3, 2, 1	

Note how disk 5 moves only once. Is there any pattern in the recursive solution that one can exploit?

Better Questions

- Peg-to-Peg (PtP): move all disks from one peg to another.
- Arbitrary-to-Peg (AtP): move all disks from some arbitrary configuration to one peg.
- Arbitrary-to-Arbitray (AtA): move all disks from some arbitrary configuration to some other arbitrary configuration.

Since there are multiple solutions, it is natural in each case to ask for the solution requiring the least number of moves.

And, one would like good algorithms to compute the optimal solution.

Pushing further, one could ask about the average number of moves for the AtP or AtA version (we won't go there).

The Framework: Configuration Space

We need to find a formal way to express the configurations of our game: a complete description of the state of affairs at any particular time.

We will call this collection of all possible configurations $\mathcal{H}_n$.

Of course, we also need to figure out how these configurations are related; in particular when a configuration C can change into a configuration C' in a single move.

Since the rules are symmetric, we should think of $\mathcal{H}_n$ as an undirected graph, the Hanoi graph of order n.

Representation 1: Position Vectors

Arguably the most obvious way to represent configurations is to think of them as n-vectors of peg positions:

$$C : [n] \to \nvDash$$

We will often think of these vectors as words over the digit alphabet $\nvDash = \{0, 1, 2\}$.

It will be convenient later to index these words as in $x = x_{n-1}x_{n-2}\dots x_1x_0$ where x_{n-1} indicates the position of the largest disk, and x_0 the smallest one.

So there are 3^n configurations.

Moves

All moves are of the following kind:

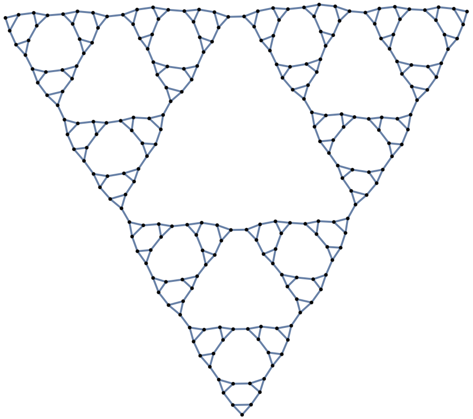
- xab^i to xcb^i where $a \neq b \neq c$.

We allow $i = 0$, in which case xa simply turns into xc. These moves are always possible, but $i > 0$ fails when we are in one of the 3 one-peg configurations.

So the nodes in $\mathcal{H}_n$ have degree 3, except for the one-peg configurations with degree 2.

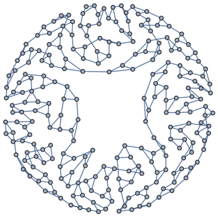
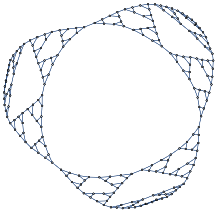
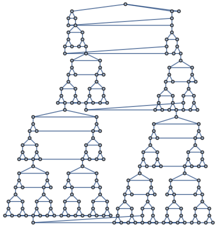
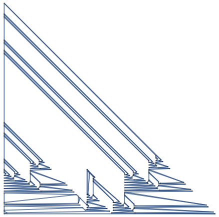
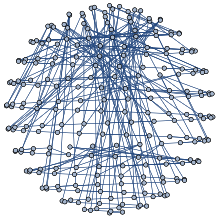
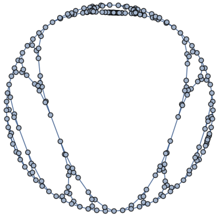
It is natural to ask if we can draw a useful picture of, say, $\mathcal{H}_5$?

Automatic Graph Rendering

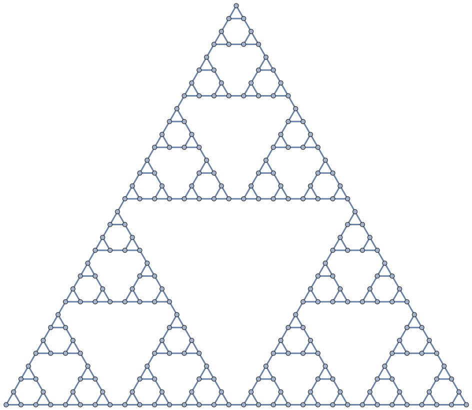


Upside down, but not bad overall. It clearly shows a nested triangular structure.

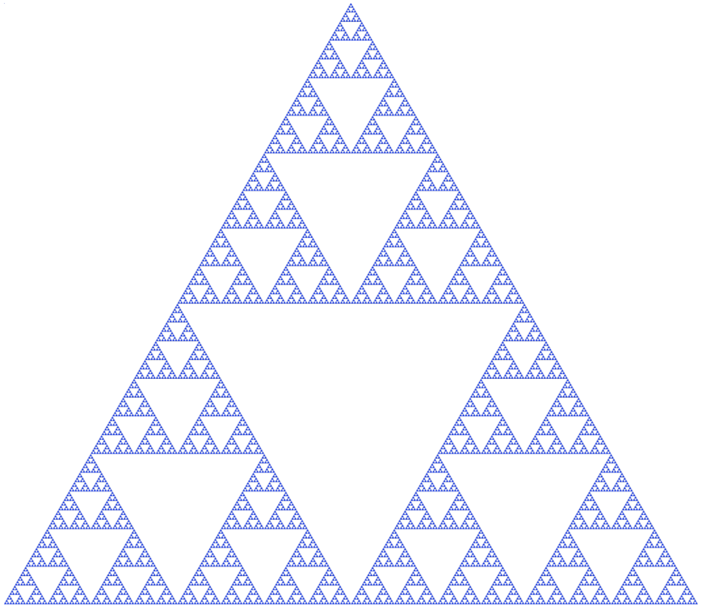
Other Layouts

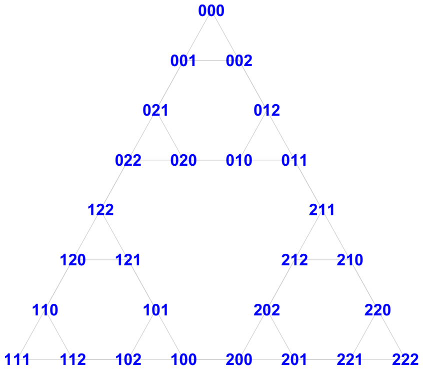
Hand Crafted



A Fractal



Labeled



Same coordinates, but with vertex labels. Ponder deeply.

Sanity Check

The doctored picture is pretty (mathematicians have slightly less taxing standards than the drama department), but it's worth checking what exactly it represents and how. Visuals sometimes can be deceiving.

- Does the picture properly represent $\mathcal{H}_n$? Why and how?
- Where in the picture is the solution for PtP?
- Where is recursion in this picture?
- What do the symmetries mean?

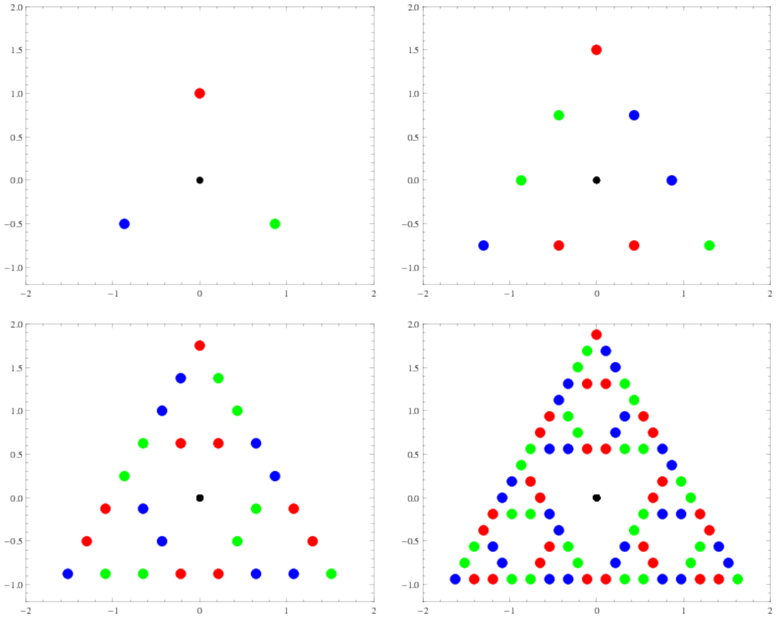
Exercise

Answer all these questions in detail.

Exercise (Hard)

Develop an algorithm to draw the perfect picture (for arbitrary n).

Hint



Application: PtP

The graph $\mathcal{H}_n$ provides a straightforward solution to PtP: the shortest path from one peg-only configuration to another clearly leads along one of the three sides of the main triangle.

It is obvious that the length of this path is $2^n - 1$. Right?

This is actually quite remarkable: we are saying there is a solution of this length, and there is no shorter solution–it does not matter how clever your solution is; even if it is non-computable, it cannot beat ours.

Really?

The last result seems compelling, but it is based on “visual evidence”: we stare at a particular embedding of a very regular graph and we “see” the shortest path right away.

Thanks to millions of years of evolution, humans excell at visual computation, but it still is a good idea to verify our insights in a more formal, logical manner.

Exercise

Prove that the shortest path between vertex 1^n and 2^n in $\mathcal{H}_n$ has length $2^n - 1$. Use only formal properties of the graph (a blind theorem prover must be able to understand your arguments).

Fractal Coordinates

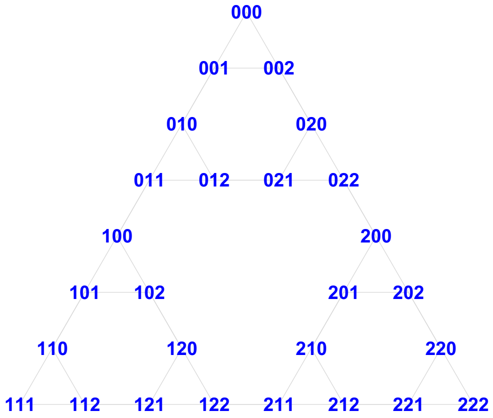
The hand-drawn (sic) picture of $\mathcal{H}_5$ from above suggests another coordinate system: the graph is composed of 3 similar, smaller copies, one on **top**, one on the **left**, and one on the **right**. If we were dealing with the continuous limit, the three copies would indeed be scaled versions of the main one.

To keep notation simple, we write t, l and r for the three directions.

So, we can uniquely label every vertex in $\mathcal{H}_n$ by a word of length n over $\{t, l, r\}$.

But we won't: naming directions is just syntactic sugar, it is cleaner to use labels $\{0, 1, 2\}$. This is the old battle between equality and isomorphism.

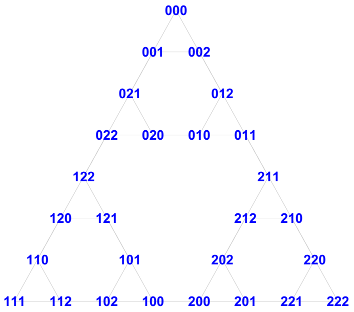
Sierpinski Labels

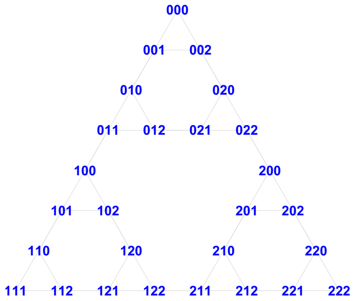


Dire Warning

We will call the new graph on $\nvDash^n$ the Sierpinski graph $\mathcal{S}_n$ of order n.

Warning: $\mathcal{H}_n$ and $\mathcal{S}_n$ are not the same.





Sierpinski Edges

The edges in $\mathcal{S}_n$ are defined like so:

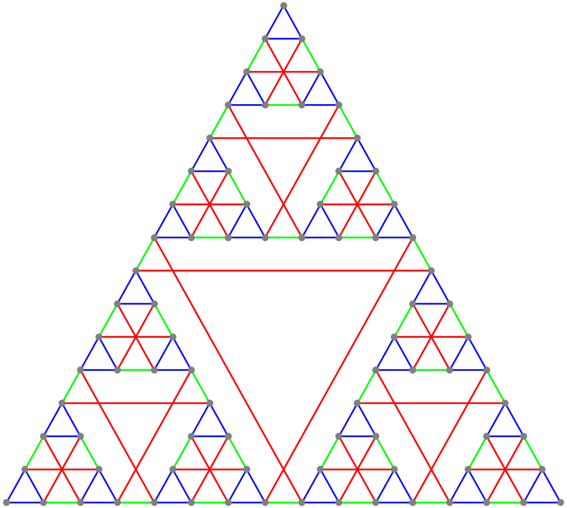
- xab^i to xba^i where $a \neq b$.

We allow $i = 0$ which produces the small triangle edges.

Note that $\mathcal{S}_n$ has a very simple recursive structure: edges are of the form (ax, ay) where $(x, y) \in \mathcal{S}_{n-1}$, plus glue edges (ab^{n-1}, ba^{n-1}).

Huge Surprise: $\mathcal{H}_n$ and $\mathcal{S}_n$ are isomorphic. This is even more surprising since they share the same vertex set and some of the edges.

Hanoi and Sierpinski



blue: triangle edges, green: Hanoi glue, red: Sierpinski glue

Automaticity

In an intuitive sense, both $\mathcal{H}_n$ and $\mathcal{S}_n$ are extremely simple computable structures: the vertex set is just $\nvDash^n$, and the edges are defined by very simple rules.

In fact, for both graphs we have:

- There is a finite state machine that recognizes the vertex set.
- There is a finite state machine that recognizes the edge set.

It is tempting to ask whether the isomorphism between $\mathcal{H}_n$ and $\mathcal{S}_n$ can be computed by a finite state machine?

A Linear Time Isomorphism

Let $\mathfrak{S}_3$ be the symmetric group on $\{0,1,2\}$ and write τ_a for the transposition fixing a: $\tau_0 = (0,2,1)$, $\tau_1 = (2,1,0)$ and $\tau_2 = (1,0,2)$.

τ_a acts on $\nvDash^\star$ pointwise. Define

$$\alpha(ax) = a\,\alpha(\tau_a(x))$$

Hence we have

$$\alpha(uv) = w\,\alpha(\tau_u(v))$$

where τ_u is the obvious abbreviation for the product of various transpositions. Note that α can easily be computed in linear time and constant space.

Lemma

α is an isomorphism from $\mathcal{S}_n$ to $\mathcal{H}_n$. Additionally, α has order 2: $\alpha^2 = I$.

A Transducer

The natural algorithm to compute α essentially maintains an element of $\mathfrak{S}_3$ to avoid linear time rewrites of the tail of the argument.

We can make this explicit by building a transducer on state set $\mathfrak{S}_3$ and transitions

$$p \xrightarrow{a/p_a} \tau_a p$$

In fact, this is an example of an invertible transducer.

Exercise

Prove the lemma. Then show that the transducer really computes α.

Transducer Diagram

$$\begin{tikzcd}
 & \tau_0 \arrow[r, <->, "1:2"] \arrow[dl, <->, "0:0"'] \arrow[ddr, <->, "2:1"] & \sigma \arrow[dr, <->, "2:0"] \arrow[ddl, <->, "0:1"'] & \\
1 \arrow[rrr, <->, "1:1"] \arrow[dr, <->, "2:2"'] & & & \tau_1 \arrow[dl, <->, "0:2"] \\
 & \tau_2 \arrow[r, <->, "1:0"'] & \sigma^2 &
\end{tikzcd}$$

Who Cares?

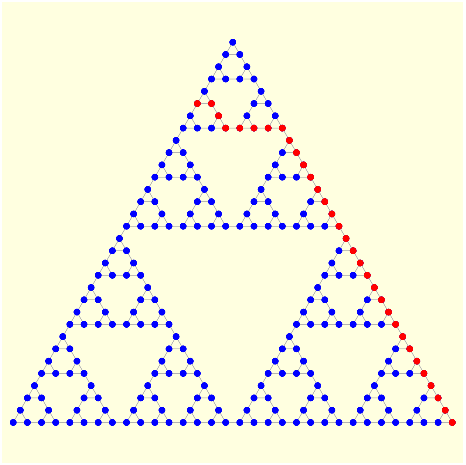
So why should anyone care about the isomorphism between $\mathcal{H}_n$ and $\mathcal{S}_n$?

Graph theorists, of course, but is there anybody else?

The isomorphism provides a computationally cheap way to go from $\mathcal{H}_n$ to $\mathcal{S}_n$, which fact can be exploited to attack shortest path problems: instead of finding a shortest path in $\mathcal{H}_n$ from x to y, we find a shortest path in $\mathcal{S}_n$ from $\alpha(x)$ to $\alpha(y)$.

The second task is easier than the first.

AtP



An optimal solution of a AtP problem.

AtP Algorithm

The AtP problem is slightly easier than the general problem. Suppose we are in configuration $x \in \nvDash^n$ and we want to move all disks to peg a.

Here is a recursive solution:

Case $x_{n-1} = a$: move $n-1$ small disks to a.

Case $x_{n-1} = b \neq a$: let c be the third peg.

- Move $n-1$ disks to peg c.
- Move nth disk to peg a.
- Move $n-1$ disks to peg a.

Note that the last part is the same as in the standard peg-to-peg problem.

Path Lengths in AtP

So how many steps does it take to get from anywhere to one peg?

Phrased differently: what is the distance between an arbitrary configuration and a one-peg-configuration.

Let's try to count:

$$N_d = \text{ number of configurations at distance } d.$$

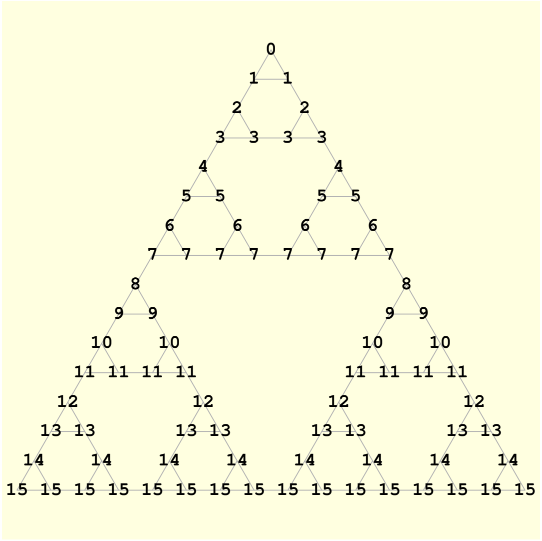
By inspection:

d	0	1	2	3	4	5	6	7	8
N_d	1	2	2	4	2	4	4	8	2

Not exactly clear, except that we seem to get powers of 2.

Note the values at 2, 4, 8.

Distance Picture



Digit Sums

It seems the answer depends on the binary expansion of d.

Let's write $\mathsf{ds}(x)$ for the digit sum of x, the number of 1's in the binary expansion.

Claim

$$N_d = 2^{\mathsf{ds}(d)}$$

Proof. (by geometry)

Induction on k: show the claim holds for all $0 \leq d < 2^k$, for all k.

Base case $k = 0$ is trivial.

Step: Use the picture to show that for $d' = 2^k + d$, $0 \leq d < 2^k$ we have $N_{d'} = 2N_d$. Then

$$2^{\mathsf{ds}(d')} = 2 \cdot 2^{\mathsf{ds}(d)} = 2N_d = N_{d'}$$

□

Distance Function

Similarly it is easy to compute the distance $\Delta_a(x)$ from an arbitrary configuration x to a one-peg configuration a^n in the Sierpinski graph $\mathcal{S}_n$.

$$\Delta_a(x) = \sum_{x_i \neq a} 2^i$$

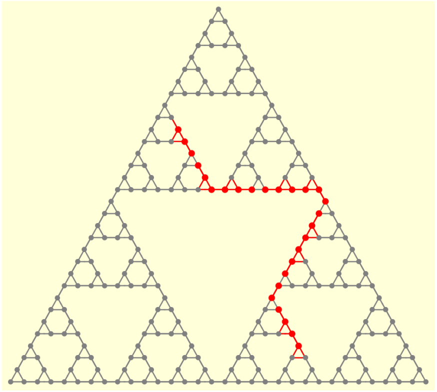
In general, the distance is given by

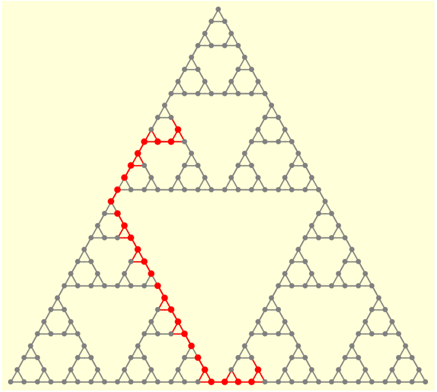
$$\text{dist}(ax, by) = 1 + \min\left(\Delta_a(x) + \Delta_b(y), 2^{|x|} + \Delta_c(x) + \Delta_c(y)\right)$$

Here $a \neq b \neq c$.

To see why, take a look at two typical shortest paths.

Two AtA Cases





Some AtA solutions move the largest disk once, some twice–whence the min operator.

Automaticity

Note that $0 \leq \Delta_a(x) < 2^n$. There is a finite state transducer that computes $\Delta_a(x)$ as an n-bit binary number, written MSD first.

Safe addition of n-bit numbers (no overflow) as well as the computation of the minimum can similarly be handled by a transducer.

In conjunction with the automatic isomorphism we have the following fact.

Lemma

The distance between two configurations in a Hanoi graph can be computed by a finite state transducer.

AtA

Exercise

Find a way to determine whether a AtA problem is of type I or type II.

Exercise

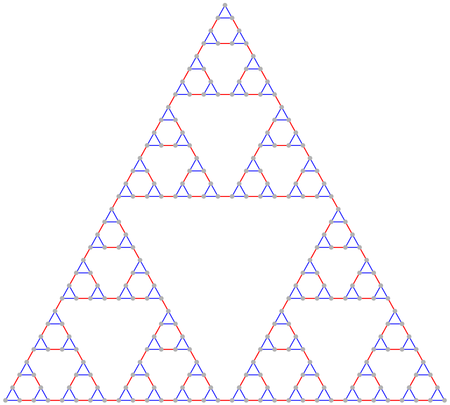
Construct a transducer that computes the distance of two AtA configurations.

Exercise

Produce an elegant algorithm to solve AtA.

A Nonrecursive PtP Solution

Here is another picture of the Hanoi graph, this time distinguishing between small and medium disk moves (triangle versus glue edges).



blue: triangle edges, red: glue edges

Alternating Moves

It is clear from the picture that on any shortest path, blue and red edges alternate.

Medium moves are forced: there is only one peg the disk can go to.

Small disk moves could be clockwise or counterclockwise.

In each blue triangle row, clockwise or counterclockwise moves alternate (ignore the horizontal edges), and each row starts with a clockwise move.

Hence there is a surprisingly simple solution for the peg-to-peg problem:

Alternately move the smallest disk clockwise, and the medium disk as forced, until all disks are on the target peg.

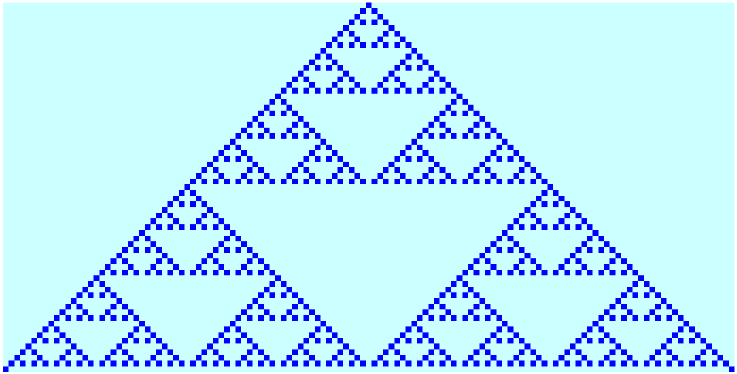
Exercise

Prove that the non-recursive method really works.

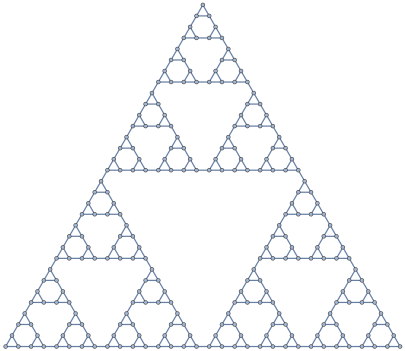
Pascal's Triangle

```
                              1
                           1     1
                        1     2     1
                     1     3     3     1
                  1     4     6     4     1
               1     5     10    10    5     1
            1     6     15    20    15    6     1
         1     7     21    35    35    21    7     1
      1     8     28    56    70    56    28    8     1
   1     9     36    84    126   126   84    36    9     1
1     10    45    120   210   252   210   120   45    10    1
```

Pascal modulo 2



. . . and Hanoi



Hanoi graphs have an uncanny similarity to Pascal’s triangle modulo 2.

Could this be mere coincidence?

Odd Binomials

We claim that the odd binomial coefficients $\binom{a}{b}$ where $0 \leq b \leq a \leq n = 2^n - 1$ correspond to possible configurations in n-disks Hanoi.

When is a binomial coefficient odd?

Definition

Let $0 \leq x \leq y$. Write $x = x_k x_{k-1} \dots x_1 x_0$ and $y = y_k y_{k-1} \dots y_1 y_0$ for the binary expansions, both padded to k bits. Define x to be bitwise less-than-or-equal y ($x \sqsubseteq y$) if $\forall\, i\, (x_i \leq y_i)$

Theorem (Lucas, 1878)

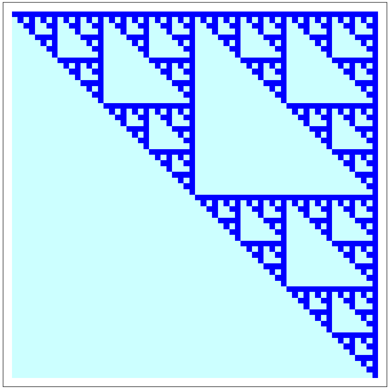
$\binom{y}{x}$ *is odd if, and only if,* $x \sqsubseteq y$.

Thinking about bitvectors, this means $S_x \subseteq S_y$ iff $x \sqsubseteq y$ iff $\binom{y}{x}$ is odd.

Déjà vu all over

Note that $\sqsubseteq$ is just the product order on $\mathbf{2}^k$.

We have seen a plot of this relation before (below for $k = 6$).



Again, this looks vaguely like a Hanoi graph, but rotated. With a little bit of effort, it too can be made to look very much like $\mathcal{H}_6$.

But Why?

As a first sanity check we have to check that the number of odd binomial coefficients $\binom{a}{b}$, $0 \leq b \leq a < 2^n$, is 3^n.

Recall the binary digit sum $\mathsf{ds}(x)$. An easy induction on n shows

$$\sum_{b \sqsubseteq a < 2^n} 1 = \sum_{a < 2^n} 2^{\mathsf{ds}(a)} = 3^n.$$

By Lucas' theorem there are $2^{\mathsf{ds}(a)}$ odd binomial coefficients of the form $\binom{a}{b}$.

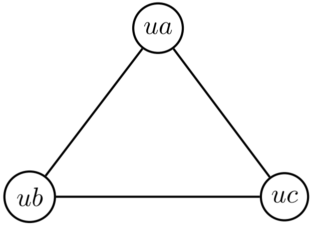
Hence there is a bijection between the odd binomials and Hanoi configurations. But we are looking for more: we want a natural isomorphism, a bijection that preserves adjacencies, preferably one that is easy to compute.

Hanoi Adjacency

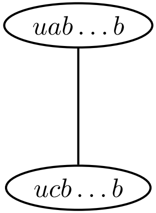
In a Hanoi graph, adjacency is defined by the "move one disk" operation.

Let $a, b, c \in \nvDash$, pairwise distinct.

Moving the smallest disk:



Moving the medium top disk:

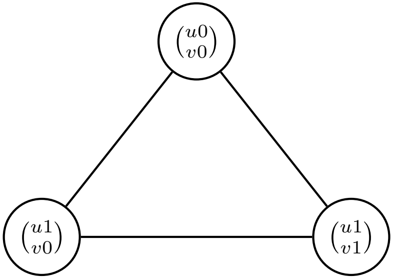


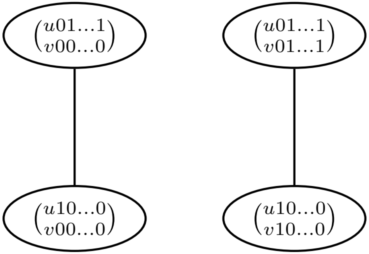
Binomial Adjacency

But for binomials, adjacency is inherited from Pascal's triangle. Since we have to transform the image a bit to get a good match, it is harder to keep track of things. Let $v \sqsubseteq u$ and write numbers in binary.

Basic triangles:

Connecting edges:




$\binom{u01\ldots1}{v00\ldots0}$
$\binom{u10\ldots0}{v00\ldots0}$
$\binom{u01\ldots1}{v01\ldots1}$
$\binom{u10\ldots0}{v10\ldots0}$

There are similarities, but it is not clear how to connect these pieces.

A Trick

Instead of converting to Hanoi configurations, let's just try to convert to Sierpinski.

We want an isomorphism

$$\text{binom} \xrightarrow{\beta} \nvDash^n$$

This turns out to be really easy: write a and b in binary, padded to n digits.

$$\begin{pmatrix} a_1 a_2 \dots a_n \\ b_1 b_2 \dots b_n \end{pmatrix}$$

Since $b_i \leq a_i$ there are 3 possible pairs (a_i, b_i). Replace (a_i, b_i) by $a_i + b_i \in \nvDash$.

Exercise

Show that β really is an isomorphism.

Controlling Complexity

As the geometry examples show, recursion can be used to generate complexity from very simple rules.

Often one is interested in the opposite direction:

> Complicated problems may have simple recursive solutions.

Requirements:

- Simple starting point.
- A way to obtain a difficult solution in terms of a less difficult one (or perhaps several less difficult ones).

This is in stark contrast to an explicit solution (which may be preferable but very hard to obtain).

Programming Languages

We can describe arithmetic expressions recursively as follows:

$$\begin{aligned}
\text{expr} &= \text{integer} \\
\text{expr} &= \text{expr} + \text{expr} \\
\text{expr} &= \text{expr} - \text{expr} \\
\text{expr} &= \text{expr} \times \text{expr} \\
\text{expr} &= \text{expr} \,/\, \text{expr} \\
\text{expr} &= (\ \text{expr}\ )
\end{aligned}$$

Programming languages can be (almost) completely specified in terms of such recursive rules (so-called context-free grammars).

Even better: the rules can be translated into a compiler automatically.

Recursive Datatypes

We can rephrase this definition of expressions as follows:

Atoms	Every integer is an expression.
Constructors	Expressions are closed under the constructors plus, minus, times, divide and parenthesize.

Traditionally recursive datatypes used to be called “inductively defined sets”.

The advantage of this type of definition is that it makes it easy to define operations on the datatype, and to prove properties of the datatype and/or the operations:

Atoms	Define the operation on atoms.
Constructors	Define the operation on compound objects obtained from constructors (assuming the operation is already defined on the subobjects.

More on this in the section on Induction.

Recursion in CS

Prime example: recursively defined functions.
The computation is similar to a sub-computation.

Requirements:

- Need exit condition.
- Calls must be on “smaller” values.

Smaller can mean many things here:
$n - 1$, $n/2$, shorter list, subtree, ...

Need a well-ordering: no infinite descending chains

$$x_0 \succ x_1 \succ x_2 \succ x_3 \succ \ldots \succ x_n \succ x_{n+1} \succ \ldots$$

Otherwise the computation goes on forever.

Usual Examples

Factorial

```
int  fac( int x )
{
        if( x == 0 ) return 1;
        return  x * fac( x-1 );
}
```

Euclidean Algorithm

```
int  gcd( int x, int y )
{
        if( y == 0 ) return x;
        return  gcd( y, x % y );
}
```

Fibonacci Numbers

$$\begin{aligned} F_0 &= 0, \qquad F_1 = 1, \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

A few values:

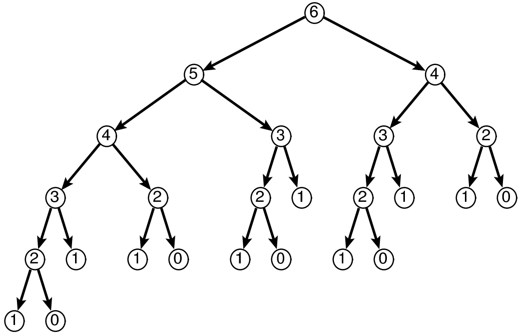
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233$

$F_{100} = 354224848179261915075$

```
int  fibo( int n )
{
        if( n < 2 ) return n;
        return  fibo(n-1) + fibo(n-2);
}
```

Disaster Strikes

This implementation is enormously wasteful! It calls itself endlessly (well, very often). E.g., $\text{fibo}(10)$ causes 55 calls to $\text{fibo}(1)$.



This type of tree is actually useful to build a data structure (see Fibonacci heaps).

Counting Calls

So how many calls are there really?

$$C_n = \text{ number of calls in } \mathrm{fibo}(n)$$

Clearly, $C_0 = C_1 = 1$ and

$$C_n = C_{n-1} + C_{n-2} + 1$$

for $n \geq 2$. This looks very similar to the definition of the Fibonacci numbers themselves.

So one might suspect that the solution has something to do with Fibonacci numbers.

We'll have more to say about recurrence relations in a while, for the time being, compute a table, perhaps there is some pattern.

A Table

In the next table, $S_n = \sum_{i \leq n} F_n$.

n	C_n	F_n	S_n
0	1	0	0
1	1	1	1
2	3	1	2
3	5	2	4
4	9	3	7
5	15	5	12
6	25	8	20
7	41	13	33
8	67	21	54
9	109	34	88
10	177	55	143
11	287	89	232
12	465	144	376

From the table, it looks like

$$C_n = F_{n-1} + S_n$$

The Solution

A closer look shows that

$$S_n = F_{n+2} - 1$$

But then

$$\begin{aligned} C_n &= F_{n-1} + S_n \\ &= F_{n-1} + F_{n+2} - 1 \\ &= F_{n-1} + F_n + F_{n+1} - 1 \\ &= 2 \cdot F_{n+1} - 1 \end{aligned}$$

So far, this is just a conjecture, prompted by the table.

But, it's now straightforward to **prove** this conjecture by induction on n.

Finding the right answer is often much harder than showing that it is in fact correct once it has been found.

Summary

- Induction and recursion are standard concept in mathematics and computer science.
- Induction is one of the most important proof technique.
- Some simple induction proofs (most notably on the naturals) can be automated.
- In computer science, inductively defined structures are known as recursive datatypes and are used extensively.
- Recursive (inductively defined) operations on recursive datatypes are easy to implement and their properties can be established via inductive proofs.