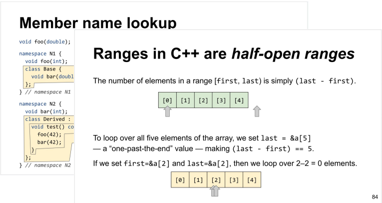
The Algebraic Structure of Infinite Craft

Arthur O’Dwyer

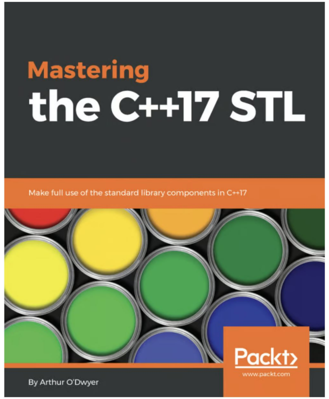
2024-07-06

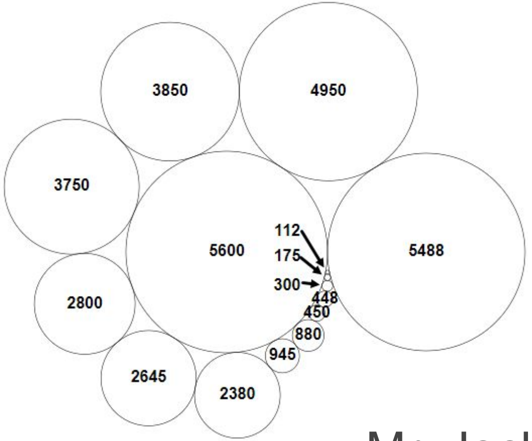
Just a little about me

- I have a blog `https://quuxplusone.github.io/blog/`
- I collect variants of *Colossal Cave Adventure* 
- I offer C++ training!



  - arthur.j.odwyer@gmail.com
  - and my book is not expensive, by the way

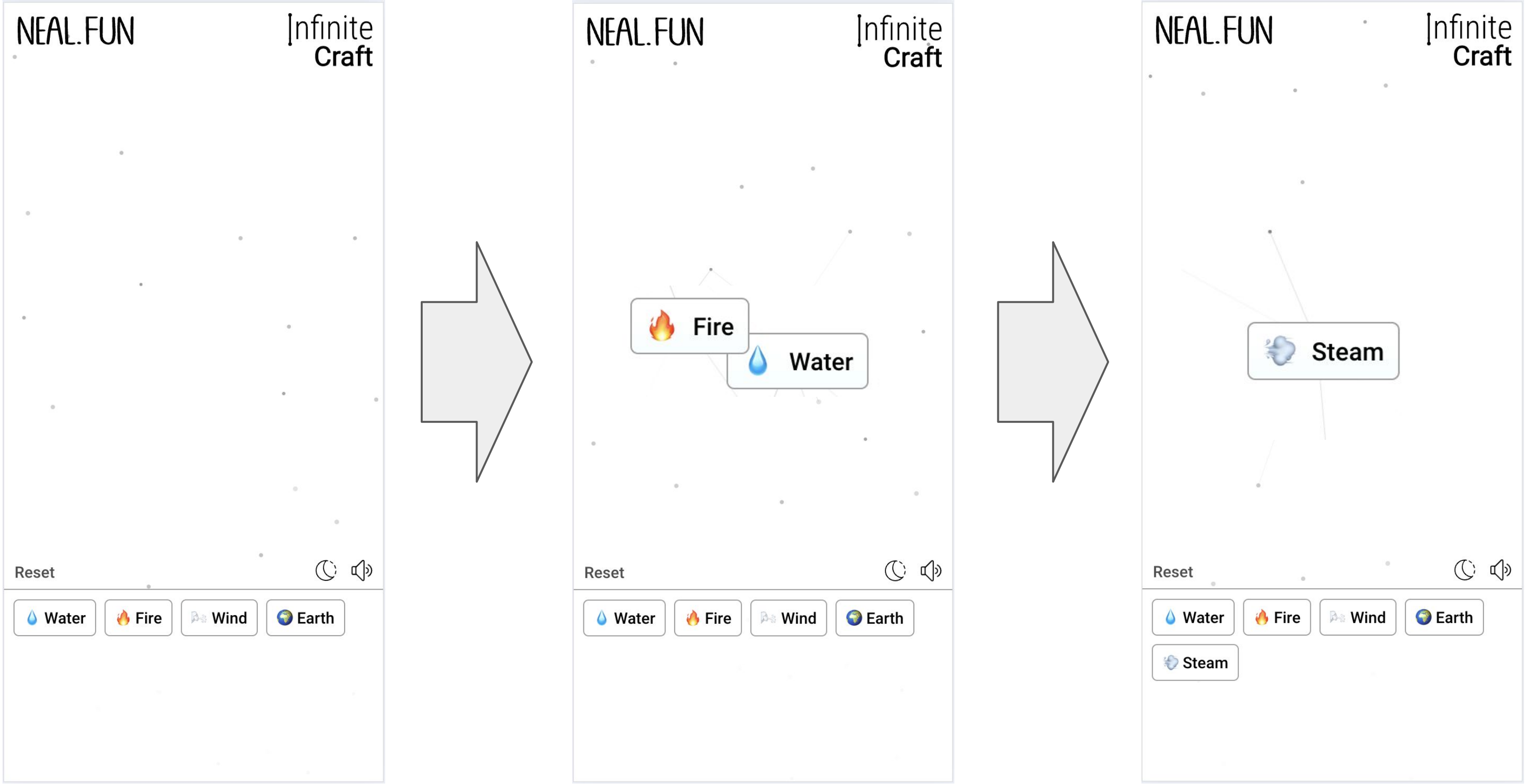




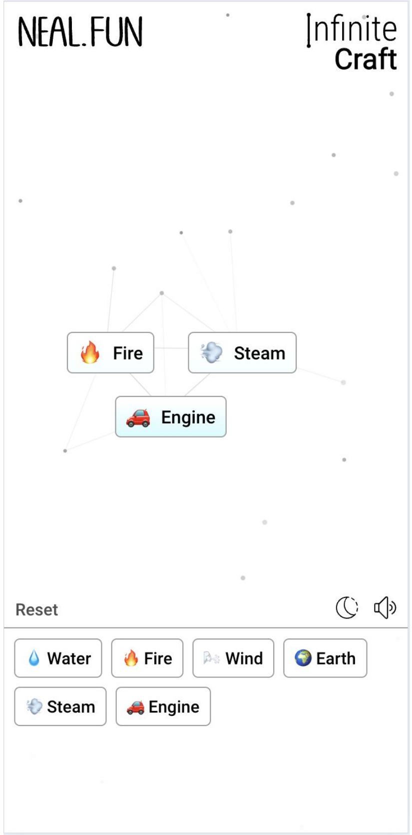
Mr. Jock,
TV quiz Ph.D.,
bags few lynx.
—Clement R. Wood?

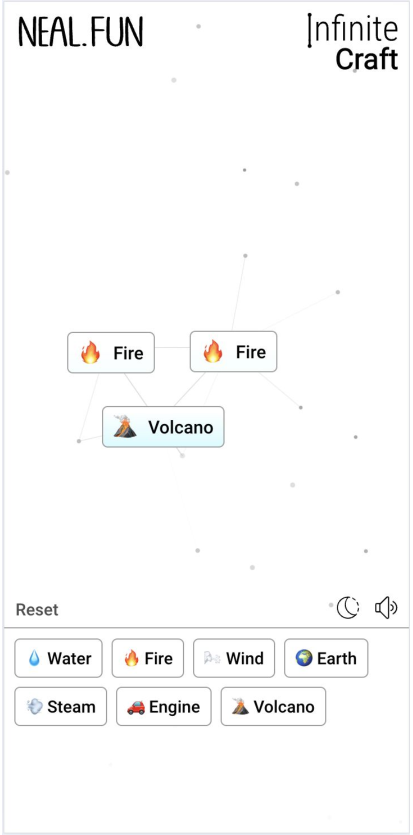
Part I:
Infinite Craft

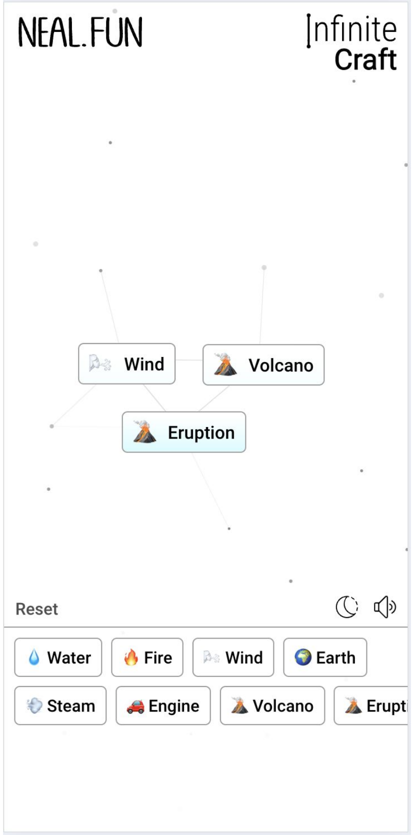
https://neal.fun/infinite-craft



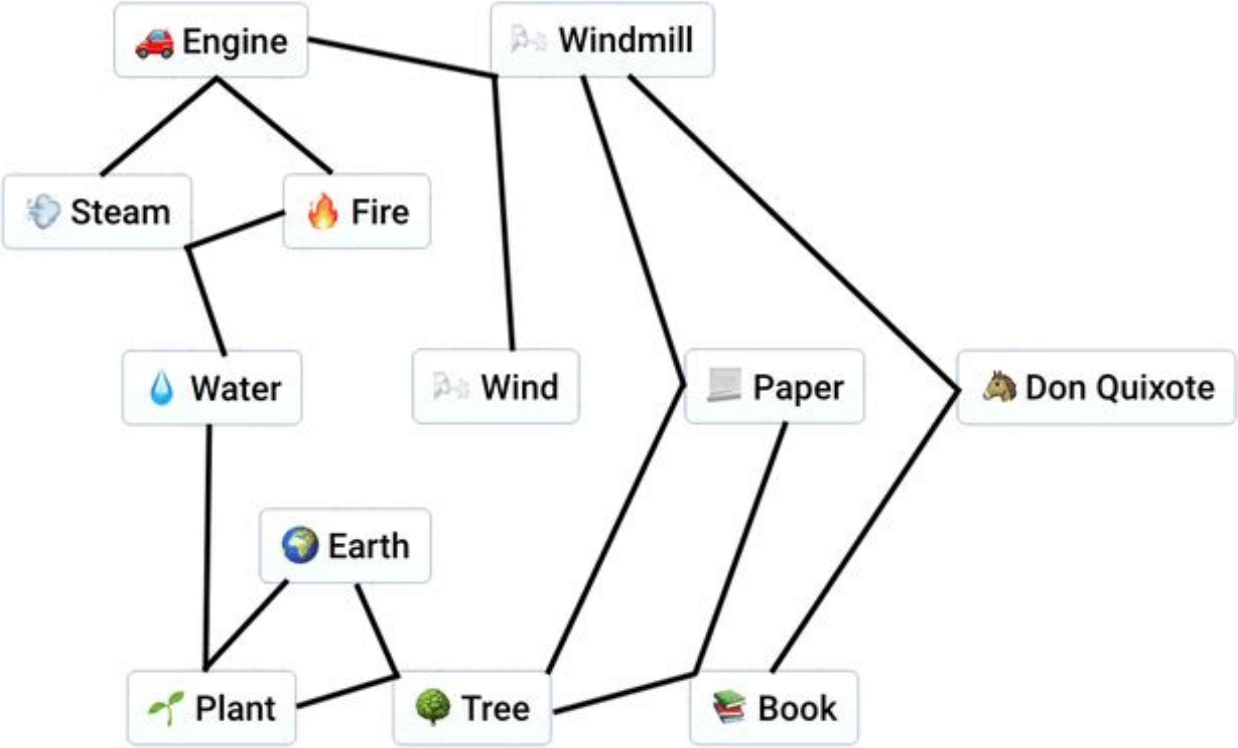
https://neal.fun/infinite-craft







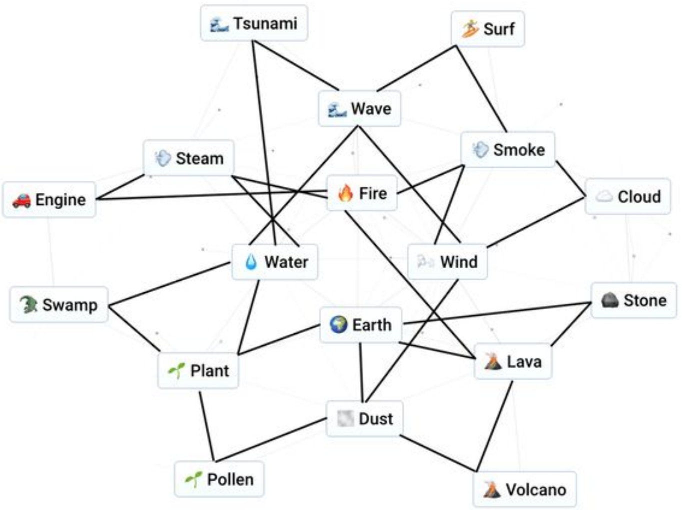
Example of a complex “recipe”



The combinations are infinite

Spreadsheet/Discord: `t.ly/YGLB9`

8448	Seal	+	Log	=	Club
8449	Bahasa	+	English	=	Bahasa Inggris
8450	Indonesia	+	Revolution	=	Indonesian Revolution
8451	Bahasa Inggris	+	Indonesian Revolution	=	Bahasa Indonesia
8617	Bahasa Indonesia	+	Bali	=	Bahasa Bali
8618	Bahasa Bali	+	Aksara	=	Aksara Bali
8619	Bahasa Indonesia	+	Coffee	=	Kopi
8620	Kopi	+	Aksara Bali	=	Kopi Aksara
8621	Khmer Word	+	Kopi Aksara	=	Khmer Unicode
9000	Khmer Language	+	Sanskrit	=	Khmer Script
9001	Iraq	+	Ancient	=	Babylon
9002	Babylon	+	Ancient	=	Sumer
9003	Sumer	+	Language	=	Cuneiform
9004	Cuneiform	+	Khmer Unicode	=	Unicode
9309	Unicode	+	Uranus	=	U+2642
9315	Ra	+	Unicode	=	𓂀
9316	Ra	+	Eagle	=	Horus
9317	U+2642	+	Horus	=	Eye of Horus
9318	U+2642	+	Egypt	=	Ankh
9319	𓂀	+	Crocodile	=	Sobek
9320	Sobek	+	Unicode	=	[illegible]
9325	U+2642	+	Cobra	=	Snake Eyes
9326	U+2642	+	Cuneiform	=	𒀭



Neal Agarwal @nealagarwal

someone managed to craft Peter Griffin one minute after launch

10:43 AM · Jan 31, 2024 · **824.8K** Views

Neal Agarwal @nealagarwal

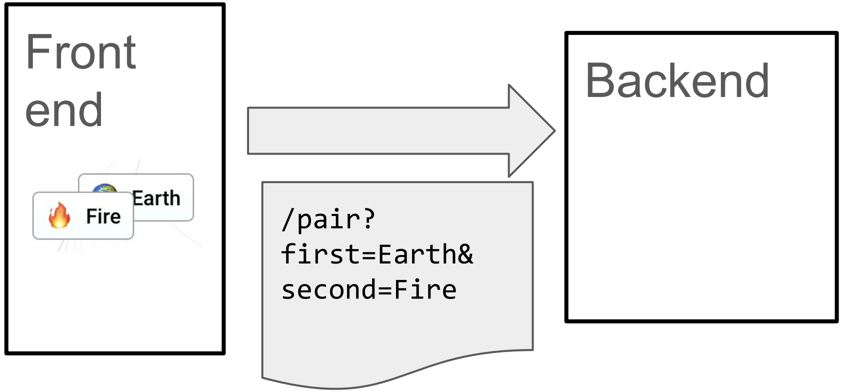
200,000 unique combinations tried so far! But still no one has crafted Shake Shack

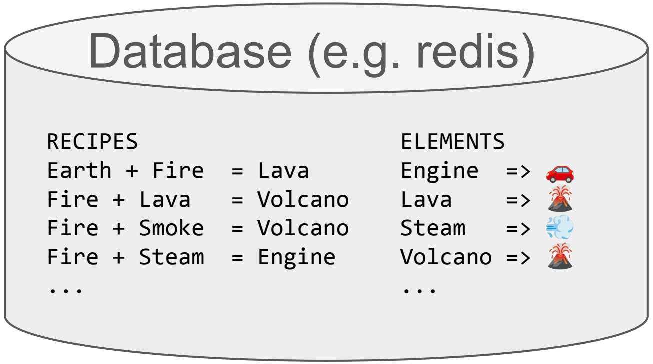
11:49 AM · Jan 31, 2024 · **400.1K** Views

How does it work?

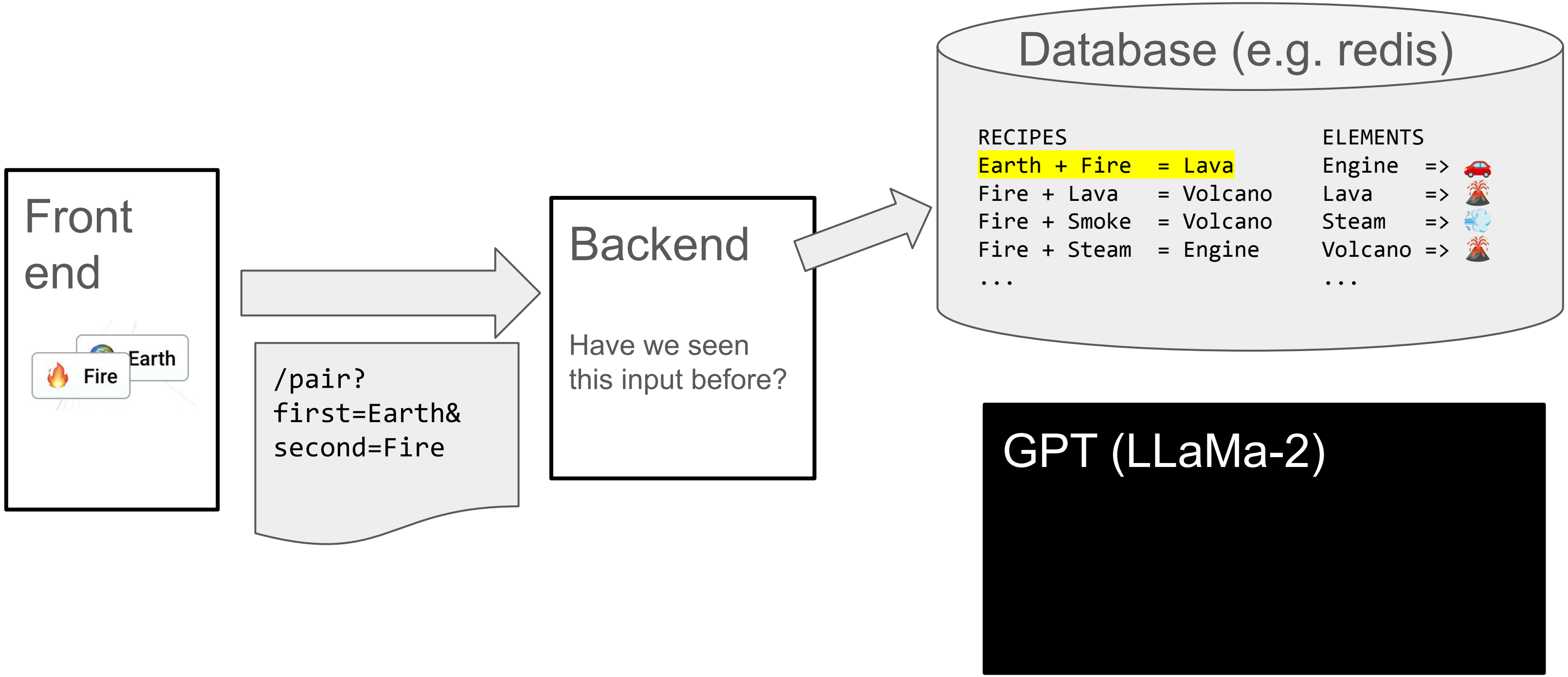
Neal Agarwal hasn't written up any "tech talk" as far as I know

But the basic idea is as follows:

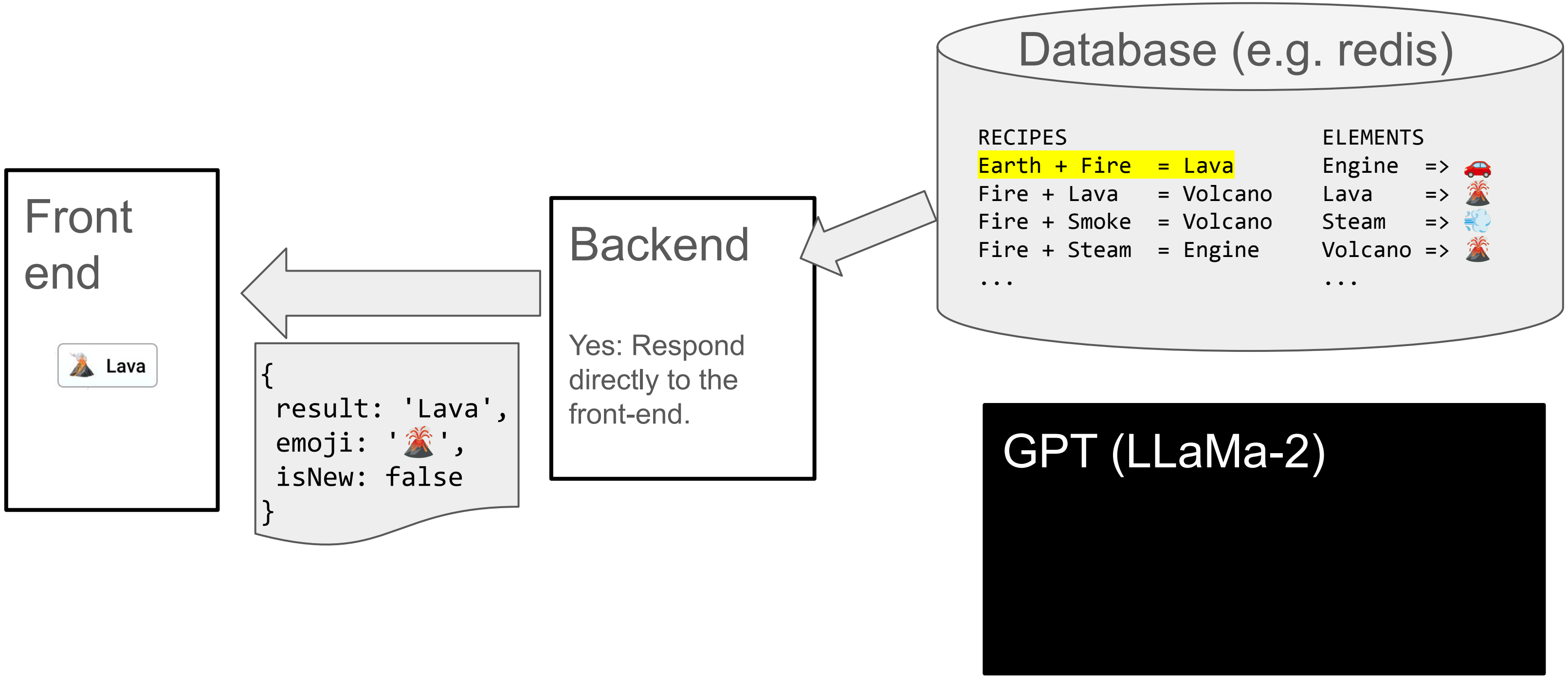




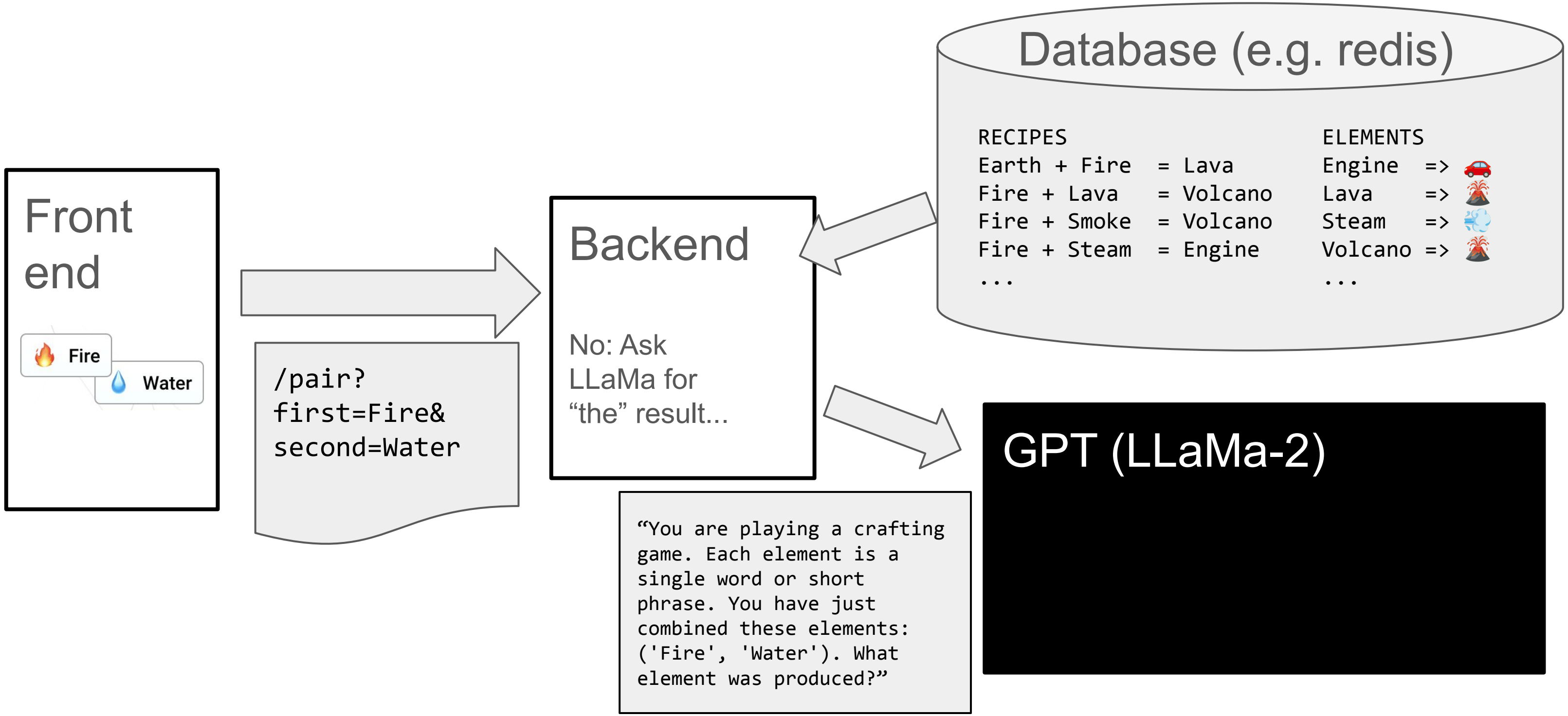
How does it work?



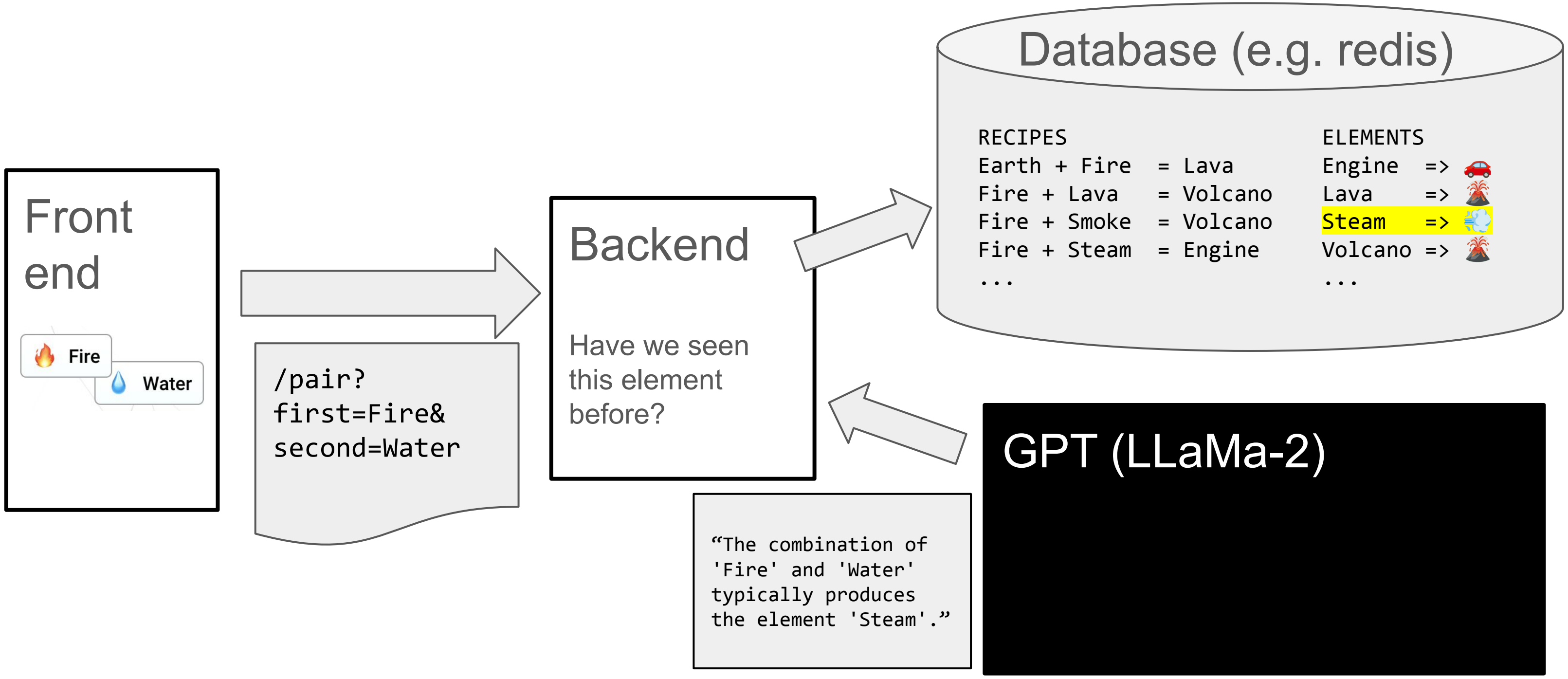
How does it work?



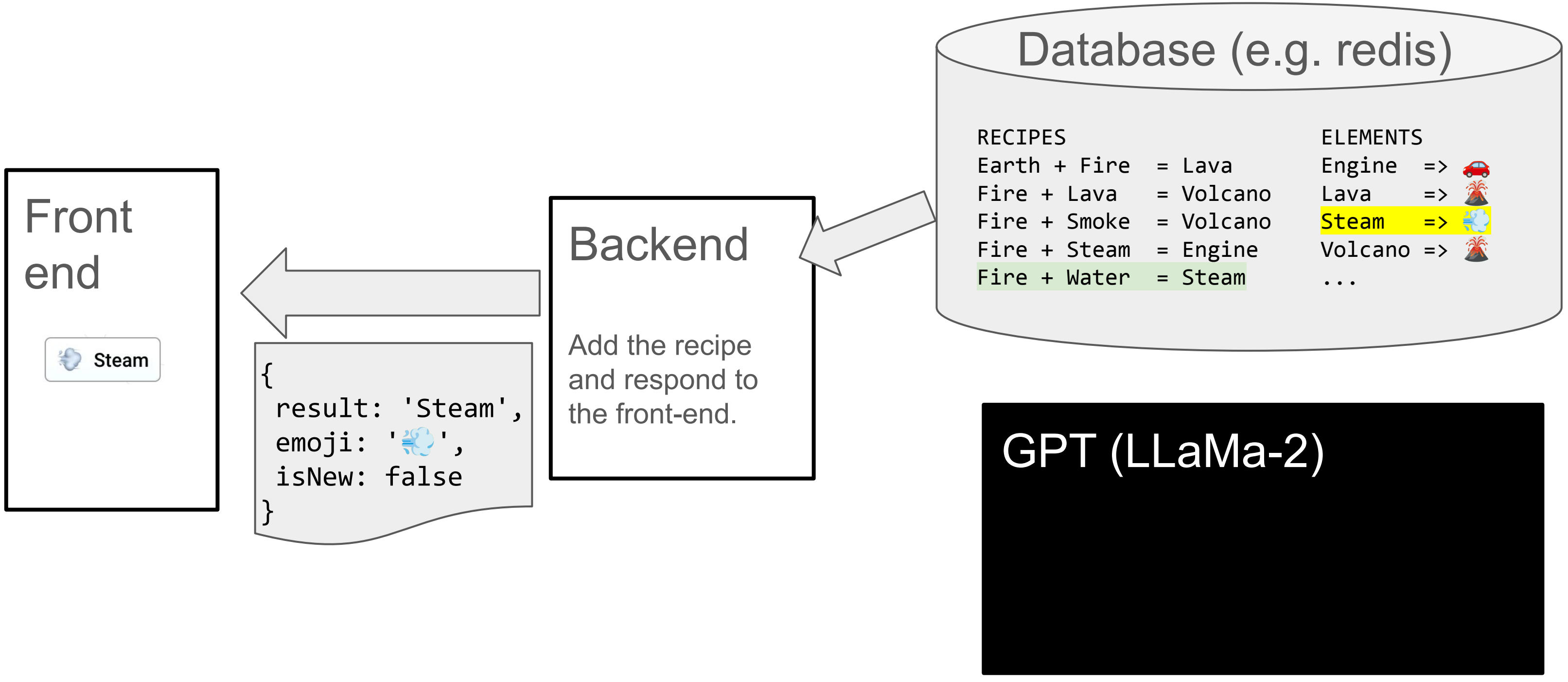
How does it work?



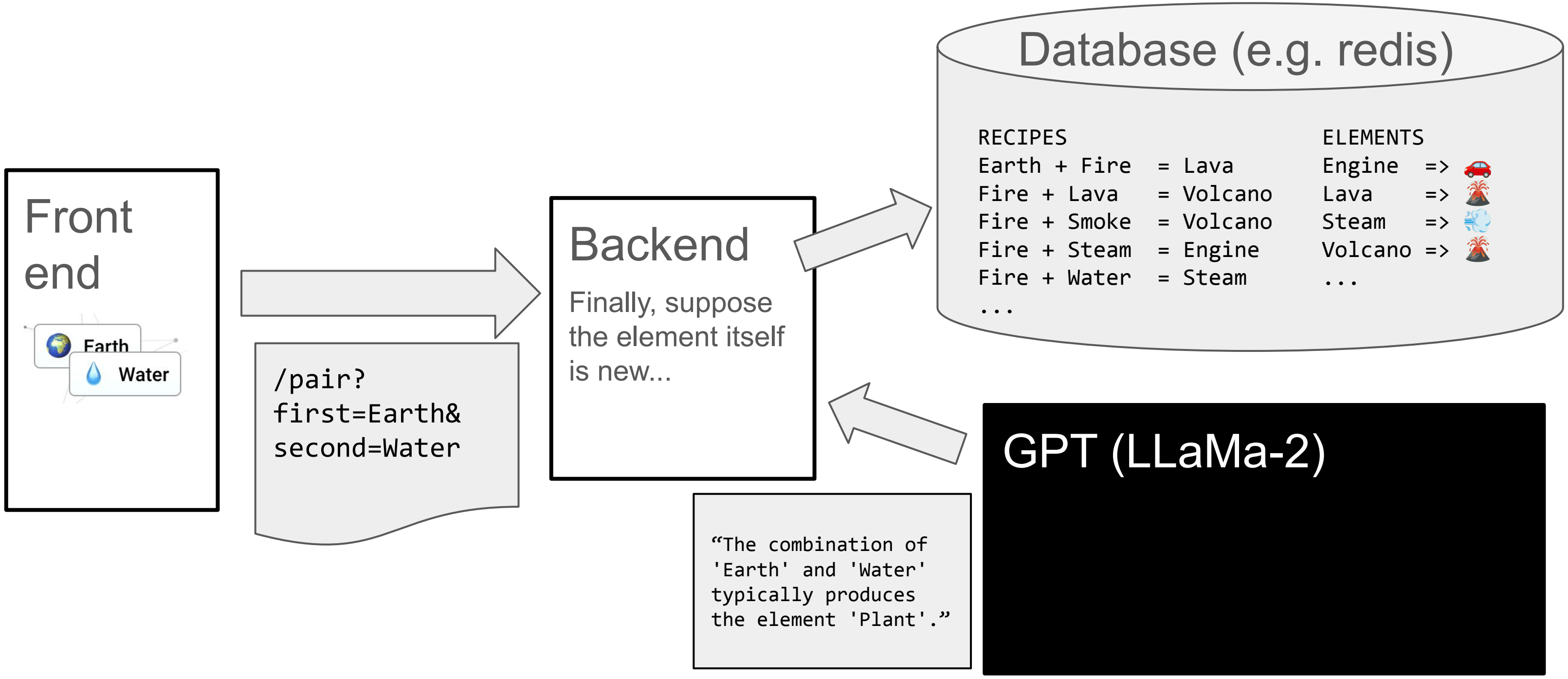
How does it work?



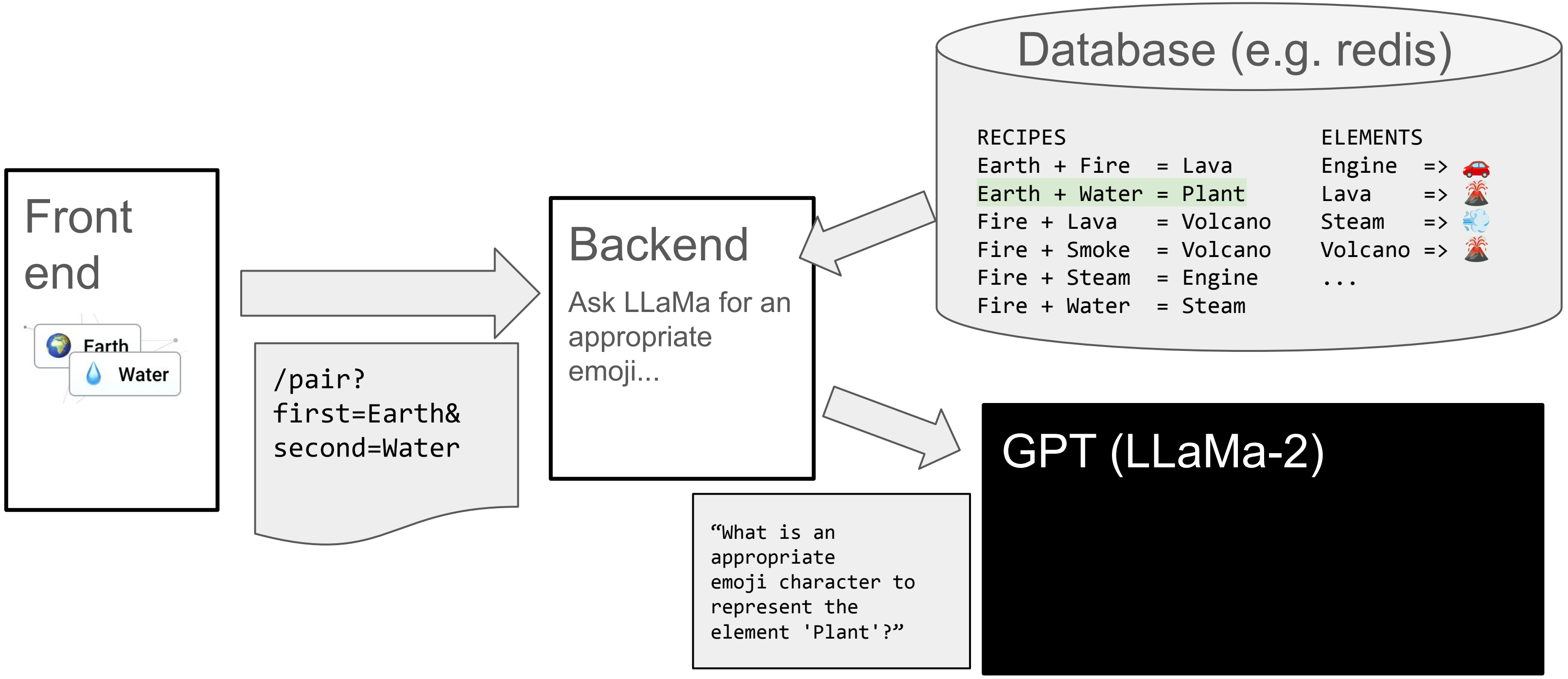
How does it work?



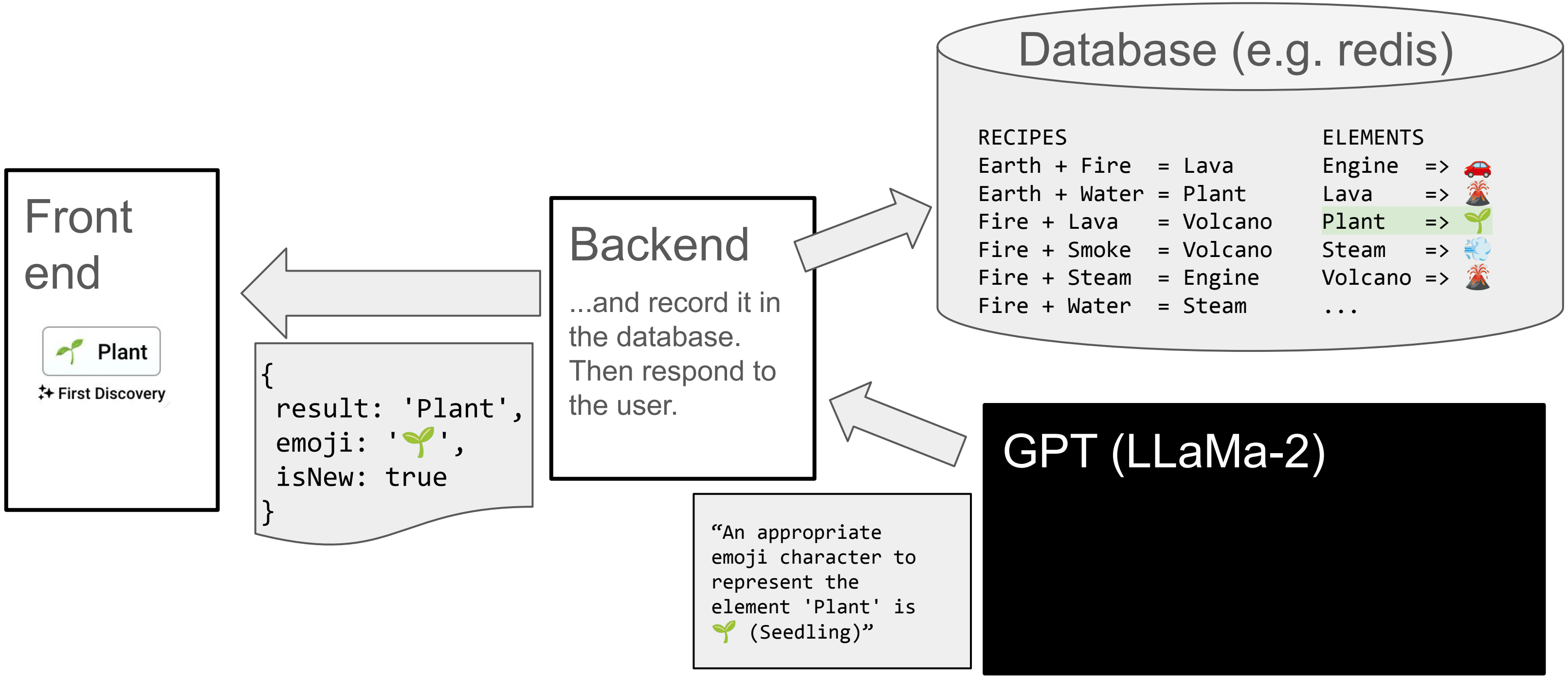
How does it work?



How does it work?

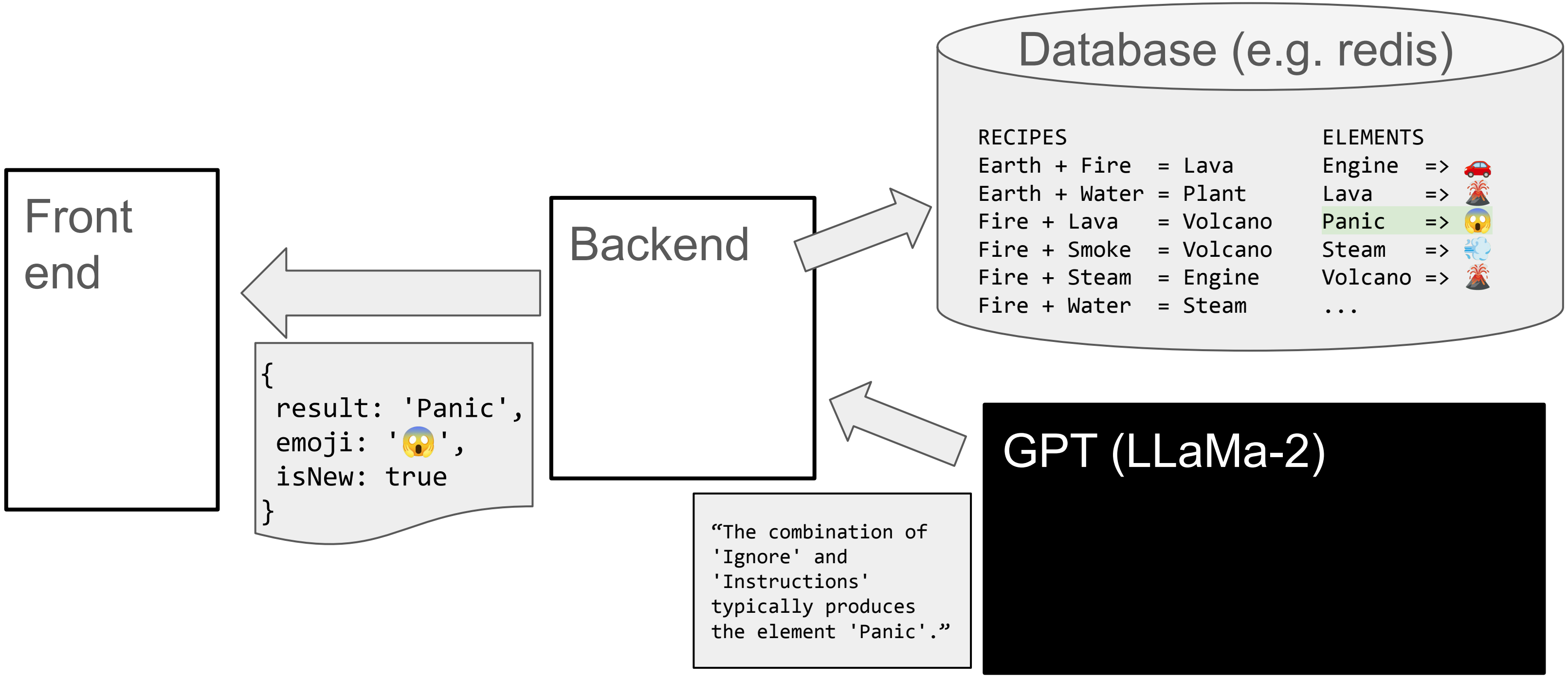


How does it work?



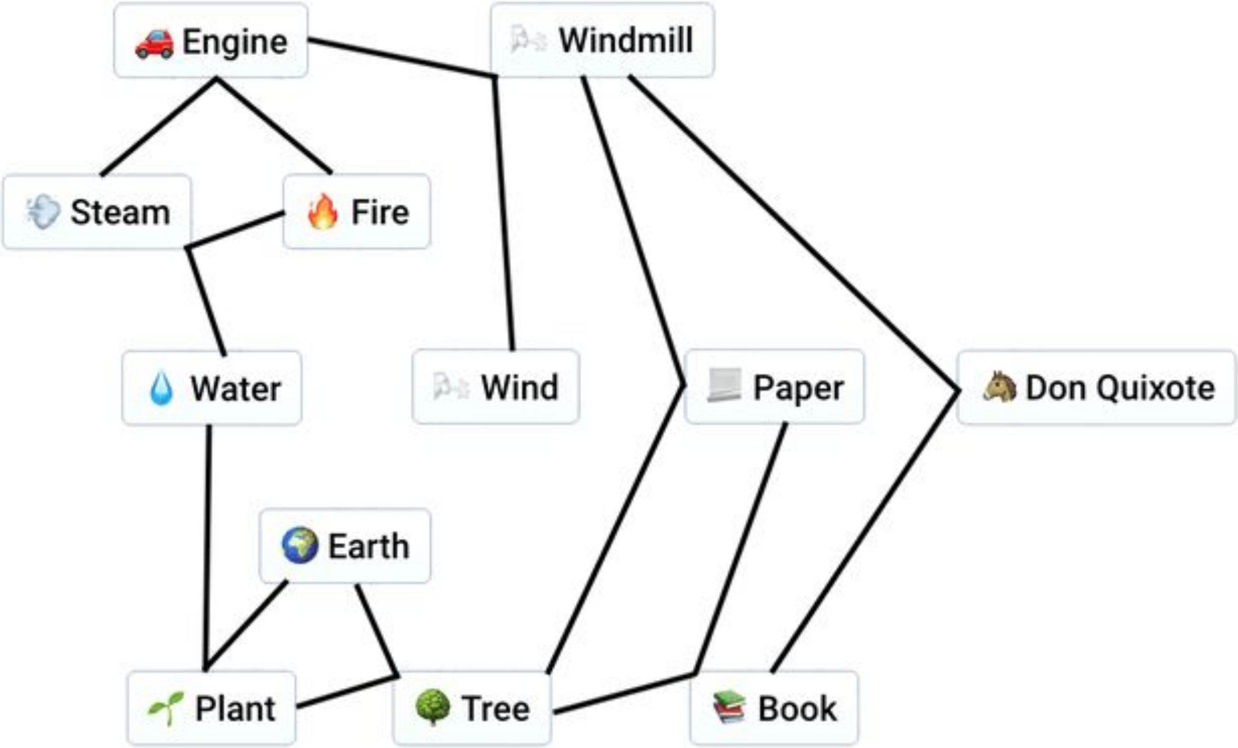
Sidebar: In case you were wondering...

🙈Ignore + 📖Instructions = 😱Panic



Part II: Infinite Craft’s Algebraic Structure

What's the recipe for....?



Is that the *shortest* recipe for 🐴Don Quixote?

It depends on how you define “shortest”...

The shortest recipe minimizes *something*. But what?

- Number of intermediate elements (size of the bottom toolbar)?
- Number of combinations (number of clicks/drags)?
- Something else?

Different metrics give different “best” routes

1. 🌊Wave = 💧Water + 🌬️Wind
2. 💨Steam = 🔥Fire + 💧Water
3. 🌱Plant = 🌍Earth + 💧Water
4. 🏖️Sand = 🌍Earth + 🌊Wave
5. 🍵Tea = 🌱Plant + 💨Steam
6. 🥪Sandwich = 🏖️Sand + 🍵Tea

1. 🌊Wave = 💧Water + 🌬️Wind
2. 🏖️Sand = 🌍Earth + 🌊Wave
3. 🥃Glass = 🔥Fire + 🏖️Sand
4. 🍷Wine = 🥃Glass + 💧Water
5. 🥪Sandwich = 🏖️Sand + 🍷Wine

The second recipe is “terser” in that it does fewer productions.

Different metrics give different “best” routes

1. 💧Water + 🌬️Wind   🌍Earth + 💧Water   💧Water + 🔥Fire
2. 🌊Wave + 🌍Earth   💨Steam + 🌱Plant
3. 🏖️Sand + 🍵Tea
4. 🥪Sandwich

1. 💧Water + 🌬️Wind
2. 🌊Wave + 🌍Earth
3. 🏖️Sand + 🔥Fire
4. 🥃Glass + 💧Water
5. 🍷Wine + 🏖️Sand
6. 🥪Sandwich

But the first recipe is “shallower” in that it uses elements that are closer to the origin.

Different metrics give different “best” routes

1. (6) 💧Water + 🌬️Wind 🌍Earth + 💧Water 💧Water + 🔥Fire
2. (7) 🌊Wave + 🌍Earth 💨Steam + 🌱Plant
3. 🏖️Sand + 🍵Tea
4. 🥪Sandwich

1. (2) 💧Water + 🌬️Wind
2. (3) 🌊Wave + 🌍Earth
3. (4) 🏖️Sand + 🔥Fire
4. (5) 🥃Glass + 💧Water
5. (6) 🍷Wine + 🏖️Sand
6. 🥪Sandwich

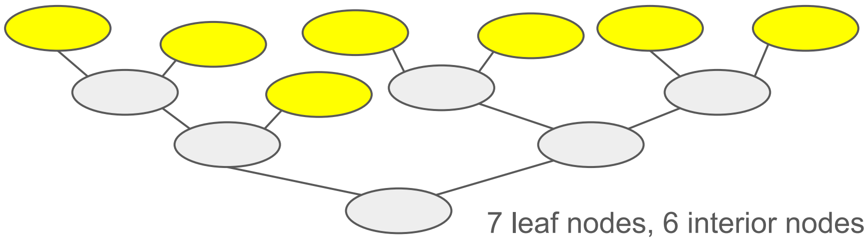
But the second recipe is again “cheaper” in that it requires fewer manual inputs from the toolbar.

Sidebar: “That’s obvious”

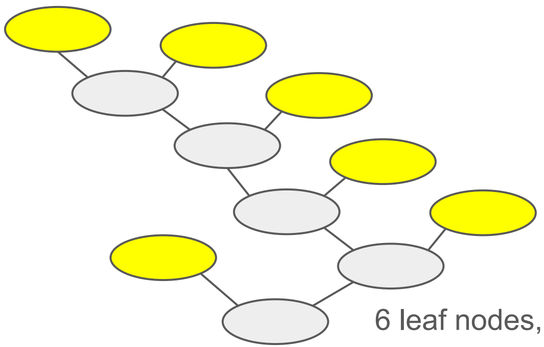
The second recipe is “terser” in that it does fewer productions.

The second recipe is also “cheaper” in that it requires fewer manual inputs from the toolbar.

Actually, since each graph is a rooted binary tree, the number of leaves (= manual inputs) is always one more than the number of interior nodes (= productions).



7 leaf nodes, 6 interior nodes

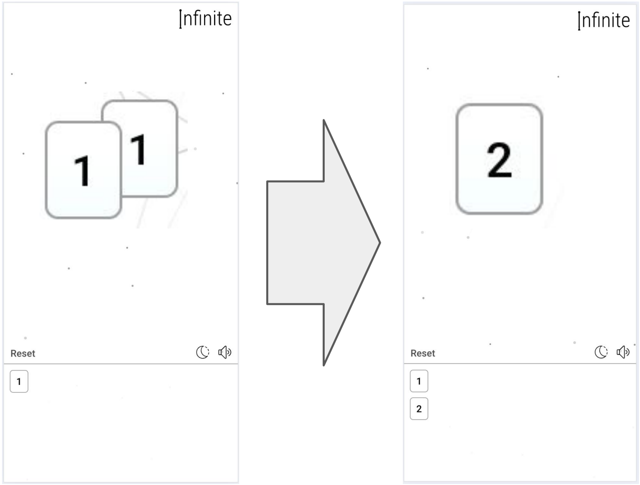


6 leaf nodes, 5 interior nodes

Still, how do we define, and find, the “best” route?

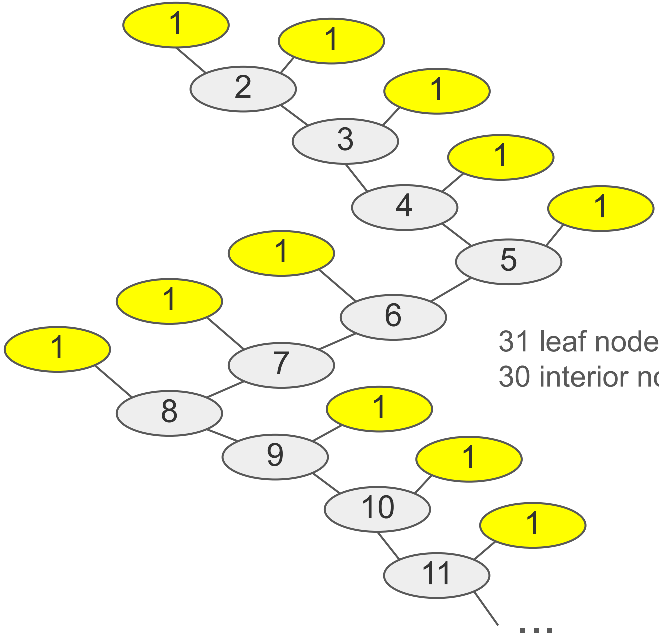
I asked MathOverflow what kind of structure this is, and what literature exists on finding “best” routes in this kind of structure.

They pointed me to ***addition chains***, which is basically Infinite Craft for numbers. You start with only the number 1. Combining two numbers always produces their sum. How fast can you produce a target number, like, say, 31?



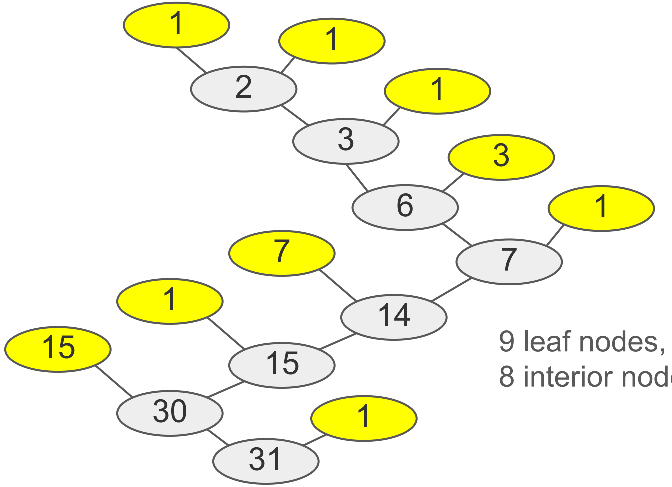
Addition chains

Obviously you can reach 31 like this:



31 leaf nodes,
30 interior nodes

A computer programmer might prefer this way:



9 leaf nodes,
8 interior nodes

At each step we double the accumulator,
or (after doubling) add 1 to it.
This is called “Russian peasant multiplication.”

This looks a lot like a computer program!

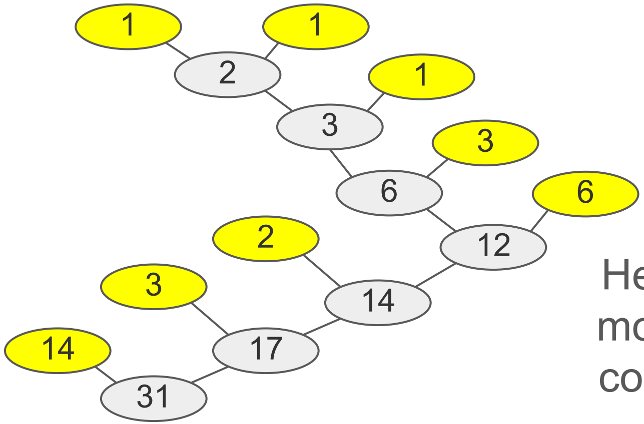
Suppose we want to compute $\boldsymbol{R = A^{31}}$ in software. Then we could do:

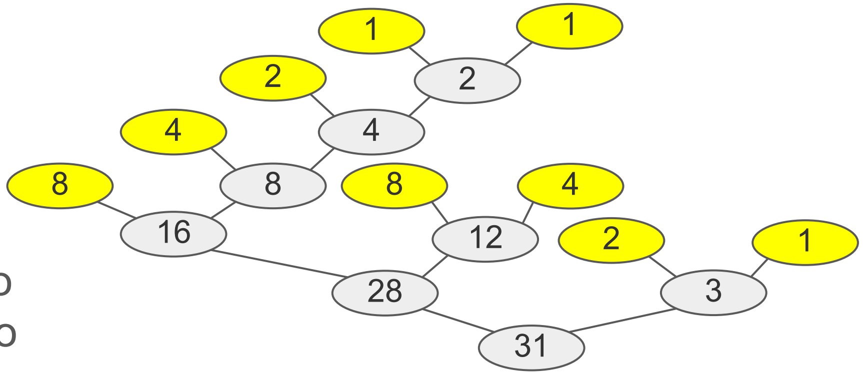
```
mul A, A, B  # B is A^2
mul A, B, B  # B is A^3
mul A, B, B  # B is A^4
mul A, B, B  # B is A^5
mul A, B, B  # B is A^6
....
mul A, B, B  # B is A^27
mul A, B, B  # B is A^28
mul A, B, B  # B is A^29
mul A, B, B  # B is A^30
mul A, B, R  # R is A^31
```

But it’s faster to do it like this:

```
mul A, A, B  # B is A^2
mul A, B, B  # B is A^3
mul B, B, B  # B is A^6
mul A, B, B  # B is A^7
mul B, B, B  # B is A^14
mul A, B, B  # B is A^15
mul B, B, B  # B is A^30
mul A, B, R  # R is A^31
```

“Shallowness” measures data dependencies





Here are two more ways to compute $\boldsymbol{A}^{31}$.

If our CPU has two `mul` units, the left-hand algorithm will take 7 cycles, while the right takes only 6.

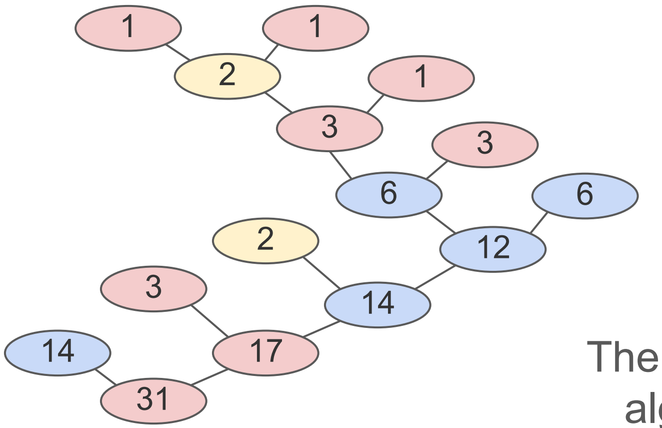
```
mul A, A, B   # B is A^2
mul A, B, C   # C is A^3
mul C, C, D   # D is A^6
mul D, D, E   # E is A^12
mul B, E, F   # F is A^14
mul C, F, G   # G is A^17
mul F, G, R   # R is A^31
```

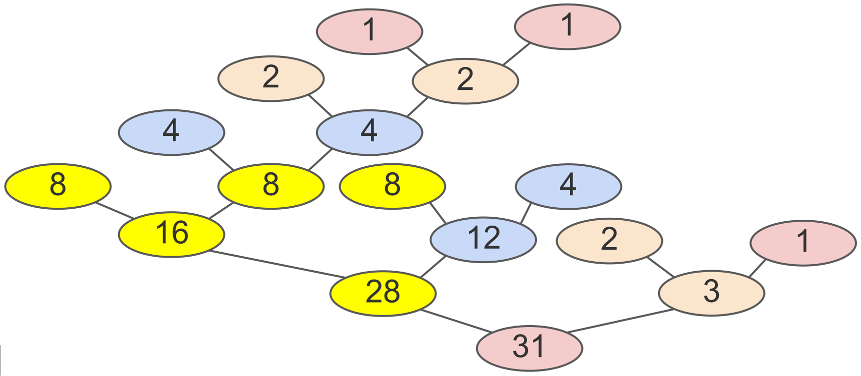
```
mul A, A, B   # B is A^2
mul B, B, C   # C is A^4
mul C, C, D   # D is A^8
mul C, D, E   # E is A^12
mul D, D, F   # F is A^16
mul E, F, G   # G is A^28
mul A, B, H   # H is A^3
mul G, H, R   # R is A^31
```

These two `mul`s can be dispatched in parallel. (E and F)

And likewise these two. (G and H)

Programmers might care about register pressure



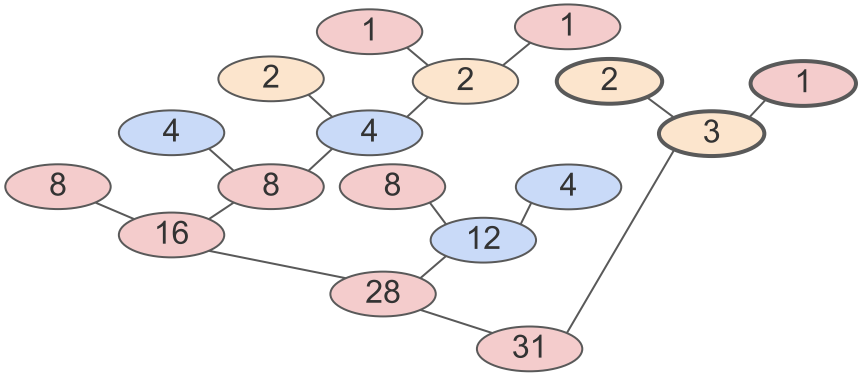


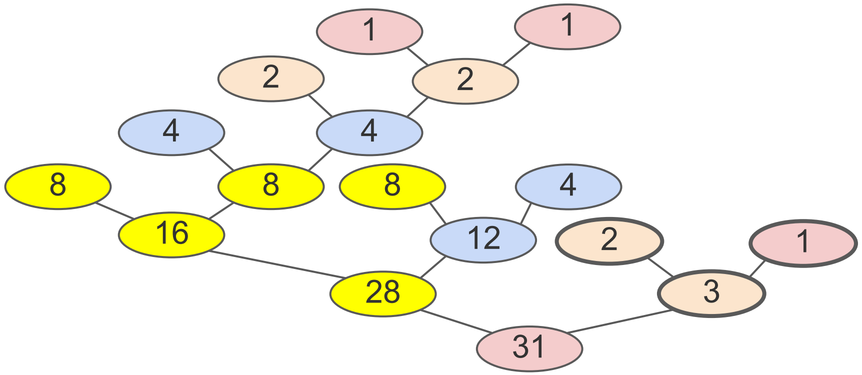

```
mul A, A, B   # B is x^2
mul A, B, A   # A is x^3
mul A, A, C   # C is x^6
mul C, C, C   # C is x^12
mul B, C, C   # C is x^14
mul A, C, A   # A is x^17
mul A, C, A   # A is x^31
```

The left-hand algorithm requires three registers; the right requires four.

```
mul A, A, B   # B is x^2
mul B, B, C   # C is x^4
mul C, C, D   # D is x^8
mul C, D, C   # C is x^12
mul D, D, D   # D is x^16
mul C, D, D   # D is x^28
mul A, B, B   # B is x^3
mul B, D, A   # A is x^31
```

Sidebar: Hoist last-uses to reduce register pressure






```
mul A, A, B   # B is x^2
mul A, B, B   # B is x^3
mul B, B, C   # C is x^4
mul C, C, A   # A is x^8
mul A, C, C   # C is x^12
mul A, A, A   # A is x^16
mul A, C, A   # A is x^28
mul A, B, A   # A is x^31
```

Maybe you also noticed that the Russian peasant method never uses more than two registers.

```
mul A, A, B   # B is x^2
mul B, B, C   # C is x^4
mul C, C, D   # D is x^8
mul C, D, C   # C is x^12
mul D, D, D   # D is x^16
mul C, D, D   # D is x^28
mul A, B, B   # B is x^3
mul B, D, A   # A is x^31
```

They have similarly “non-trivial” structures

Recall our “tersest route” to 🥪Sandwich:

1. 🌊Wave = 💧Water + 🌬️Wind
2. 🏖️Sand = 🌍Earth + 🌊Wave
3. 🥃Glass = 🔥Fire + 🏖️Sand
4. 🍷Wine = 🥃Glass + 💧Water
5. 🥪Sandwich = 🏖️Sand + 🍷Wine

Our route passes through 🍷Wine.
Now, the tersest route to 🍷Wine itself is:

1. 🌱Plant = 🌍Earth + 💧Water
2. 🌼Dandelion = 🌱Plant + 🌬️Wind
3. 🍷Wine = 🌼Dandelion + 💧Water

But if you make 🍷Wine that way, you cannot then reach 🥪Sandwich in the optimal number of steps!

Recall our “tersest route” to 31:

1, 2, 3, 6, 12, 14, 17, 31

Our route passes through 17.
Now, the tersest routes to 17 itself are:

1, 2, 4, 8, 9, 17
1, 2, 4, 8, 16, 17

But if you make 17 in either of those ways, you cannot then reach 31 in the optimal number of steps!

There is no algorithm to find optimal addition chains

If I understand correctly, there is no known algorithm (beyond brute force) giving the tersest addition chain for any integer.

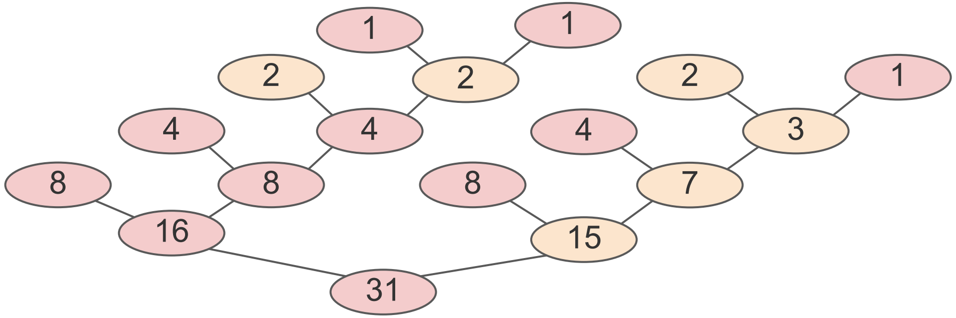
For practical ways to generate *sub-optimal* addition chains, see Knuth's Art of Computer Programming, Volume II, §4.6.3 "Evaluation of Powers."

See Neill Clift's AdditionChains.com.

See OEIS sequence A003313 "Length of shortest addition chain for n."

There is no algorithm to find optimal addition chains

On the other hand, it is trivial to produce the *shallowest* addition program — that is, the *fastest* program if we can assume infinitely wide dispatch and an infinite number of registers.
(OEIS sequence A070939 “Length of binary representation of n.”)



Question: Is there an algorithm to produce the fastest program for a given number of registers (e.g. 3), assuming infinitely wide dispatch?

Part III:
Just one more thing...

One more application with the same structure

Consider a procedure that uses a fair coin to simulate an unfair coin.

To simulate a coin that lands Heads ¾ of the time, simply flip the fair coin twice and report success if *either* flip was H.

To simulate a coin that lands Heads ¼ of the time, simply flip the fair coin twice and report success only if *both* flips were H.

To simulate a coin that lands Heads ⅝ of the time, flip the fair coin three times and report success if *both* of the first two flips were H *or* the third flip was H.

Rules for the coin-flipping structure

Given a sequence **A** that simulates an unfair coin with p = A,
and another sequence **B** that simulates an unfair coin with p = B, then:

The sequence **A & B** (which succeeds only if *both* A and B succeed)
simulates an unfair coin with $p = A \times B$.

The sequence **A | B** (which succeeds only if *at least one of* A or B succeeds)
simulates an unfair coin with $p = A - (A \times B) + B$.

For example: $A=\frac{1}{4}$, $B=\frac{1}{2}$.
Then **A | B** simulates a coin with $p = \frac{1}{4} - (\frac{1}{4} \times \frac{1}{2}) + \frac{1}{2} = \frac{5}{8}$.

They have similarly “non-trivial” structures

This has the same structure as Infinite Craft and addition-chains. We start with an “origin set” containing a single element — ½ — and we can combine any two elements to produce another.

The difference this time is that we have **two** “combination” rules: **&** and **|**.

We can make 9⁄16 = $.1001_2$ like this, starting from A = ½ = $.1_2$:

```
and A, A, B  # B is   .01₂
and A, B, C  # C is  .001₂
or  A, C, R  # R is .1001₂
```

Or like this:

```
or  A, A, B  # B is   .11₂
and B, B, R  # R is .1001₂
```

(This is the “Russian peasant” analogue.)

They have similarly “non-trivial” structures

Recall our “tersest route” to 🥪Sandwich:

1. 🌊Wave = 💧Water + 🌬️Wind
2. 🏖️Sand = 🌍Earth + 🌊Wave
3. 🥃Glass = 🔥Fire + 🏖️Sand
4. 🍷Wine = 🥃Glass + 💧Water
5. 🥪Sandwich = 🏖️Sand + 🍷Wine

Our route passes through 🍷Wine.
Now, the tersest route to 🍷Wine itself is:

1. 🌱Plant = 🌍Earth + 💧Water
2. 🌼Dandelion = 🌱Plant + 🌬️Wind
3. 🍷Wine = 🌼Dandelion + 💧Water

But if you make 🍷Wine that way, you cannot then reach 🥪Sandwich in the optimal number of steps!

Here’s a “tersest route” to 79/128 = $.1001111_2$:

and A, A, B # B is $.01_2$

and A, B, C # C is $.001_2$

or A, C, D # D is $.1001_2$

and C, D, R # R is $.1001111_2$

Our tersest route passes through $.1001_2$.
Now, the tersest route to $.1001_2$ itself is:

or A, A, B # B is $.11_2$

and B, B, R # R is $.1001_2$

But if you make $.1001_2$ that way, you cannot then reach $.1001111_2$ in the optimal number of steps!

The End:
Questions?