Thinking Recursively

Part Four

Announcements

- Assignment 2 due right now.
- Assignment 3 out, due next Monday, April 30th at 10:00AM.
  - Solve cool problems recursively!
  - Sharpen your recursive skillset!

A Little Word Puzzle

“What nine-letter word can be reduced to a single-letter word one letter at a time by removing letters, leaving it a legal word at each step?”

Shrinkable Words

- Let's call a word with this property a **shrinkable word**.
- Anything that isn't a word isn't a shrinkable word.
- Any single-letter word is shrinkable
  - A, I, O
- Any multi-letter word is shrinkable if you can remove a letter to form a word, and that word itself is shrinkable.
- So how many shrinkable words are there?

Recursive Backtracking

- The function we wrote last time is an example of **recursive backtracking**.
- At each step, we try one of many possible options.
- If *any* option succeeds, that's great! We're done.
- If *none* of the options succeed, then this particular problem can't be solved.

Recursive Backtracking

```
if (problem is sufficiently simple) {
    return whether or not the problem is solvable
} else {
    for (each choice) {
        try out that choice.
        if it succeeds, return success.
    }
    return failure
}
```

Failure in Backtracking

S	T	A	R	T	L	I	N	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	L	I	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	L	I	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	I	N	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	I	N	G

S	T	R	T	I	N	G

Failure in Backtracking

S T A R T L I N G

S T A R T I N G

S T R T I N G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	I	N	G

Failure in Backtracking

S	T	A	R	T	L	I	N	G

S	T	A	R	T	I	N	G

S	T	A	R	I	N	G

Failure in Backtracking

- Returning false in recursive backtracking does **not** mean that the entire problem is unsolvable!
- Instead, it just means that the current subproblem is unsolvable.
- Whoever made the call to this function can then try other options.
- Only when all options are exhausted can we know that the problem is unsolvable.

Extracting a Solution

- We now have a list of words that allegedly are shrinkable, but we don't actually know how to shrink them!
- Could we somehow have our function tell us if there's a solution?

Output Parameters

- An **output parameter** (or **outparam**) is a parameter to a function that stores the result of that function.
- Caller passes the parameter by reference, function overwrites the value.
- Useful if you need to return multiple values.

CHeMoWIZrDy

CHeMoWIZrDy

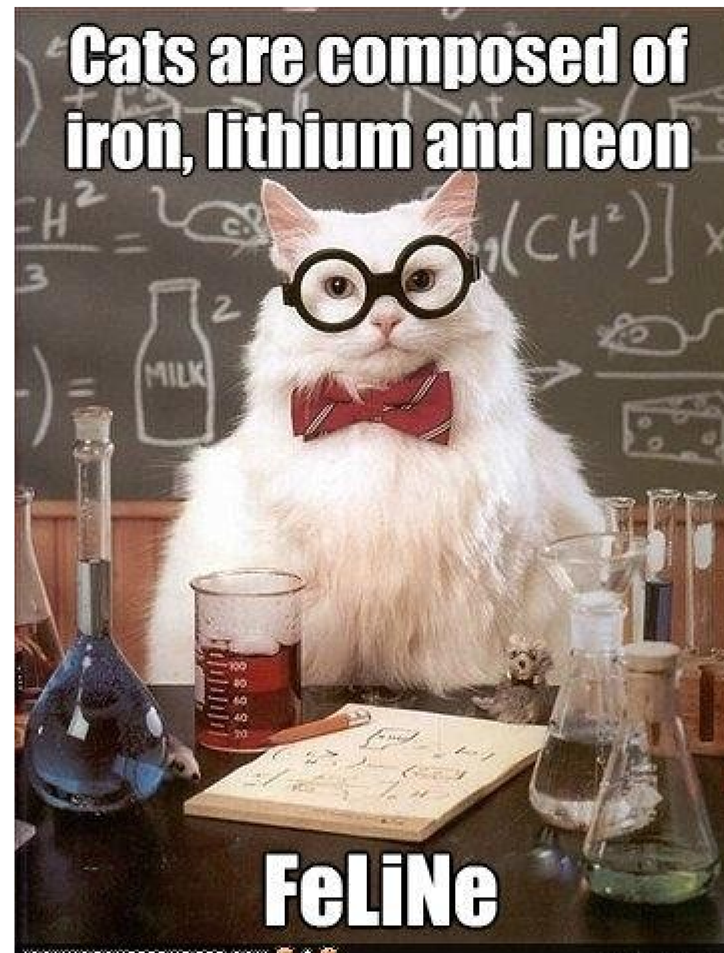
- Some words can be spelled using just element symbols from the periodic table.

CHeMoWIZrDy

- Some words can be spelled using just element symbols from the periodic table.

CHeMoWIZrDy

- Some words can be spelled using just element symbols from the periodic table.



CHeMoWIZrDy

- Some words can be spelled using just element symbols from the periodic table.
- Given a word:
  - Can you spell it out using just element symbols?
  - If so, what does it look like?

RhHeCuRhSiON

- **BaSe CaSe**:
  - The empty string can be spelled using just element symbols.
- **RhHeCuRhSiV STeP**:
  - For each 1-, 2-, or 3-letter prefix:
    - If that prefix is an element symbol, then check if the rest of the word is spellable.
    - If so, then the original word is spellable too.
  - Otherwise, no option works, so the word isn't spellable.

Revisiting an Old Problem

Buying Cell Towers


137


42


95


272


52

Buying Cell Towers

137	42	95	272	52

Buying Cell Towers

137	42	95	272	52

Buying Cell Towers

14 22 13 25 30 11 9

Buying Cell Towers

14	22	13	25	30	11	9

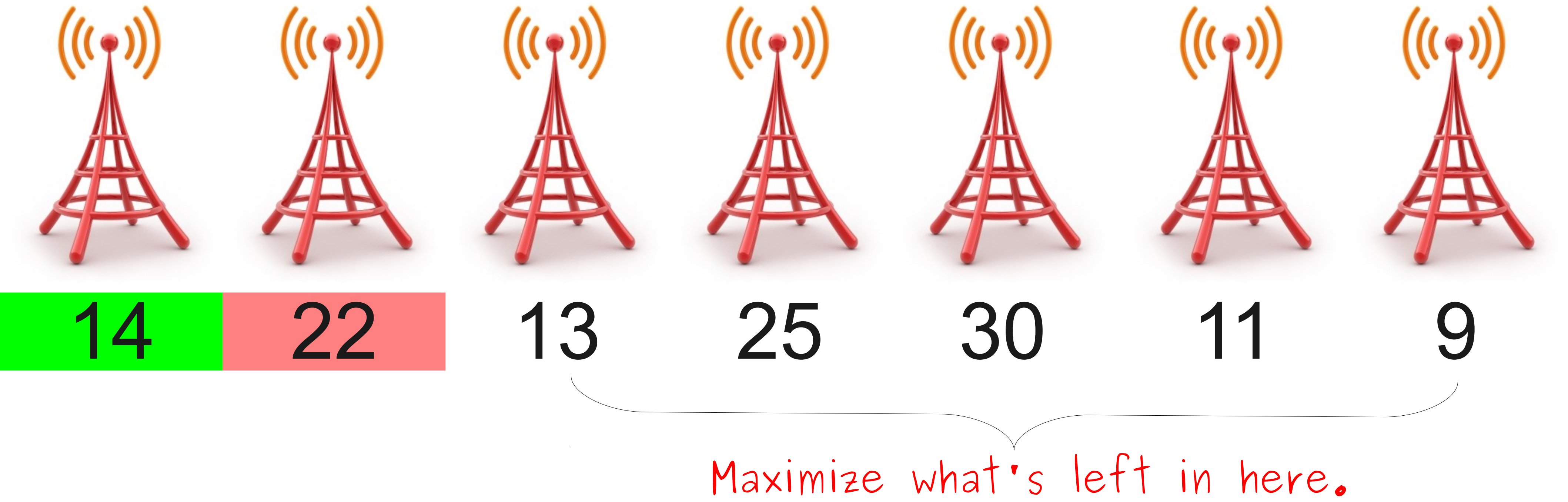
Buying Cell Towers

14	22	13	25	30	11	9

14 22 13 25 30 11 9

14	22	13	25	30	11	9

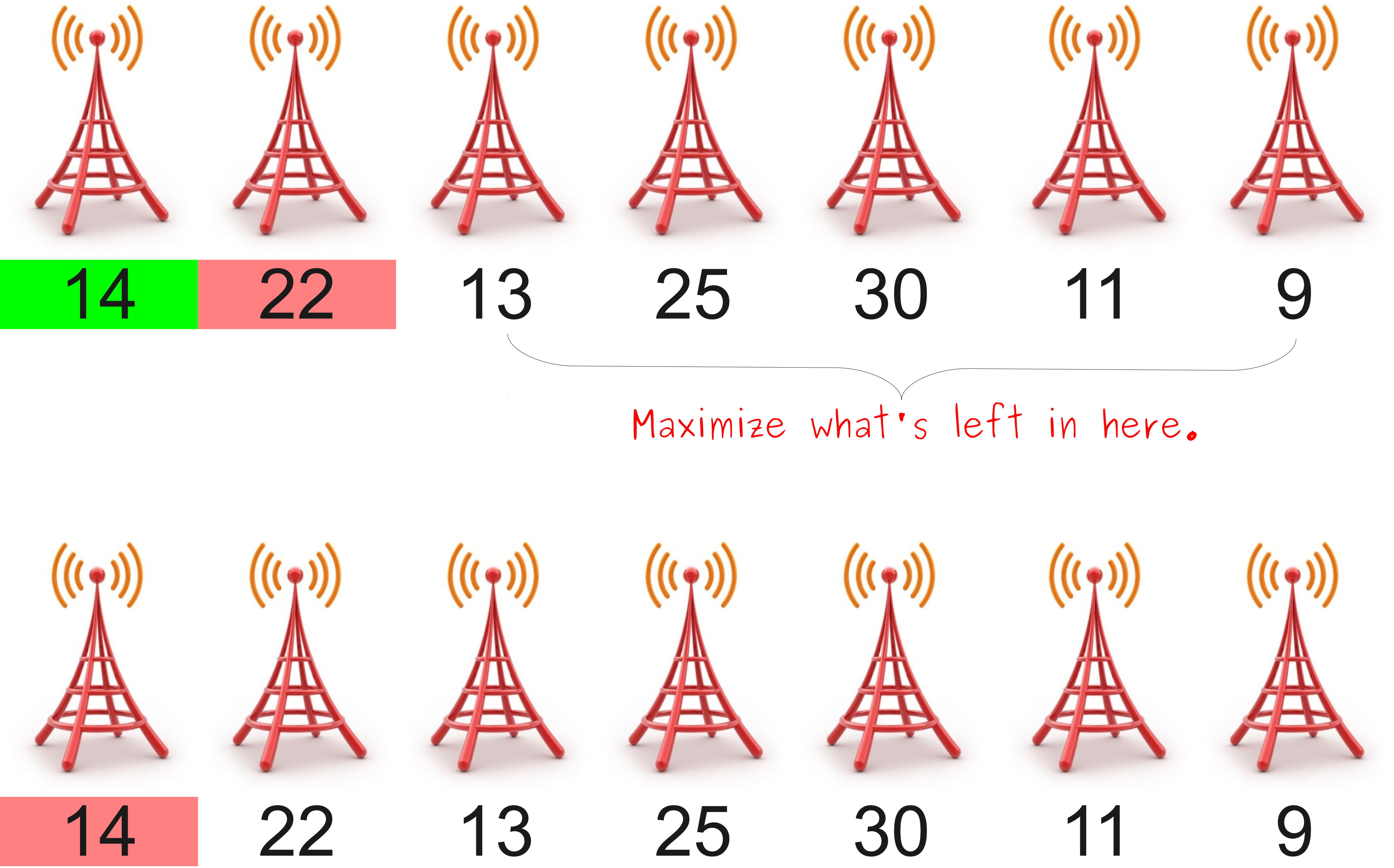
14	22	13	25	30	11	9

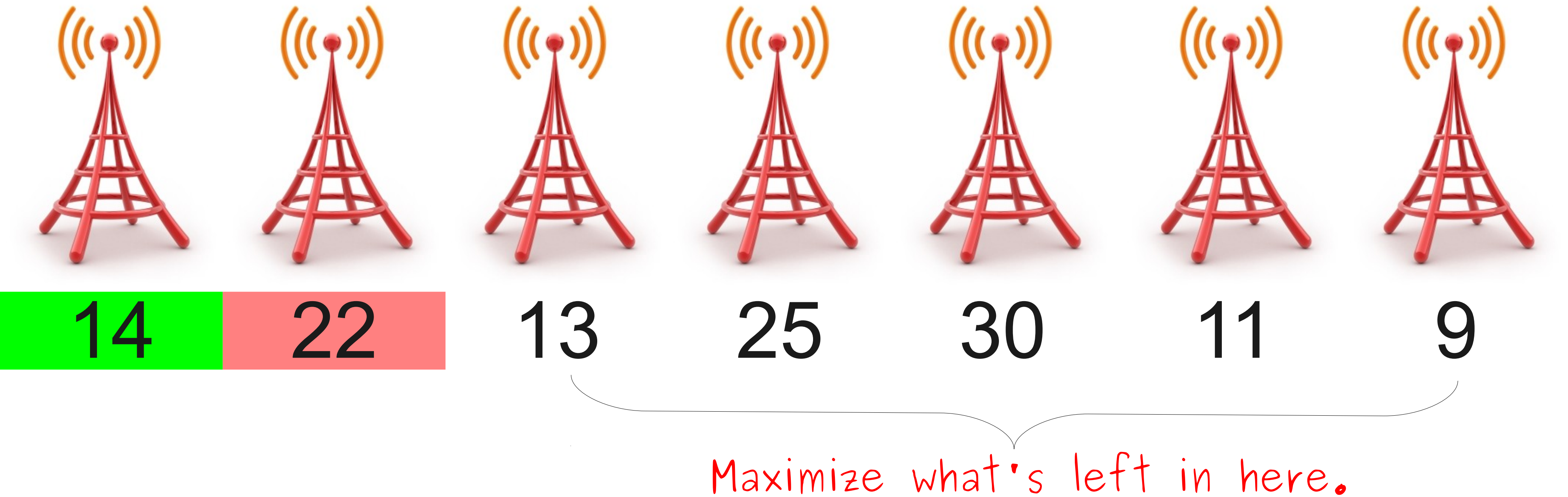

14
22
13
25
30
11
9
Maximize what's left in here.

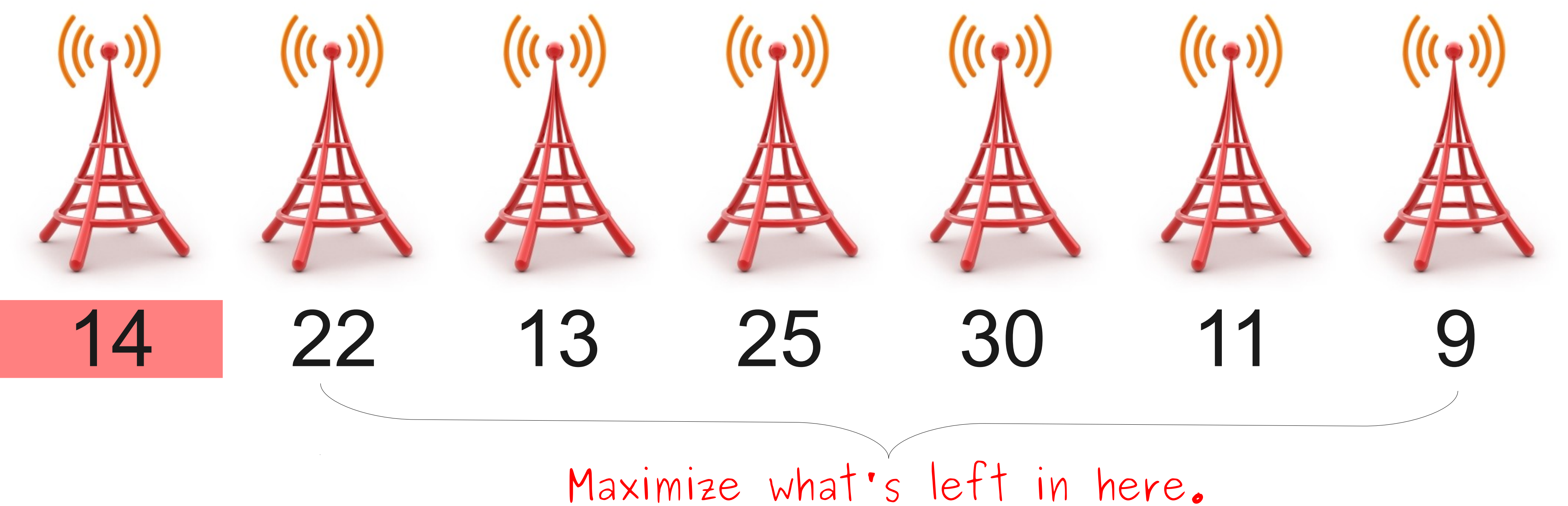
14 22 13 25 30 11 9

Maximize what's left in here.

14 22 13 25 30 11 9


14
22
13
25
30
11
9
Maximize what's left in here.
14
22
13
25
30
11
9

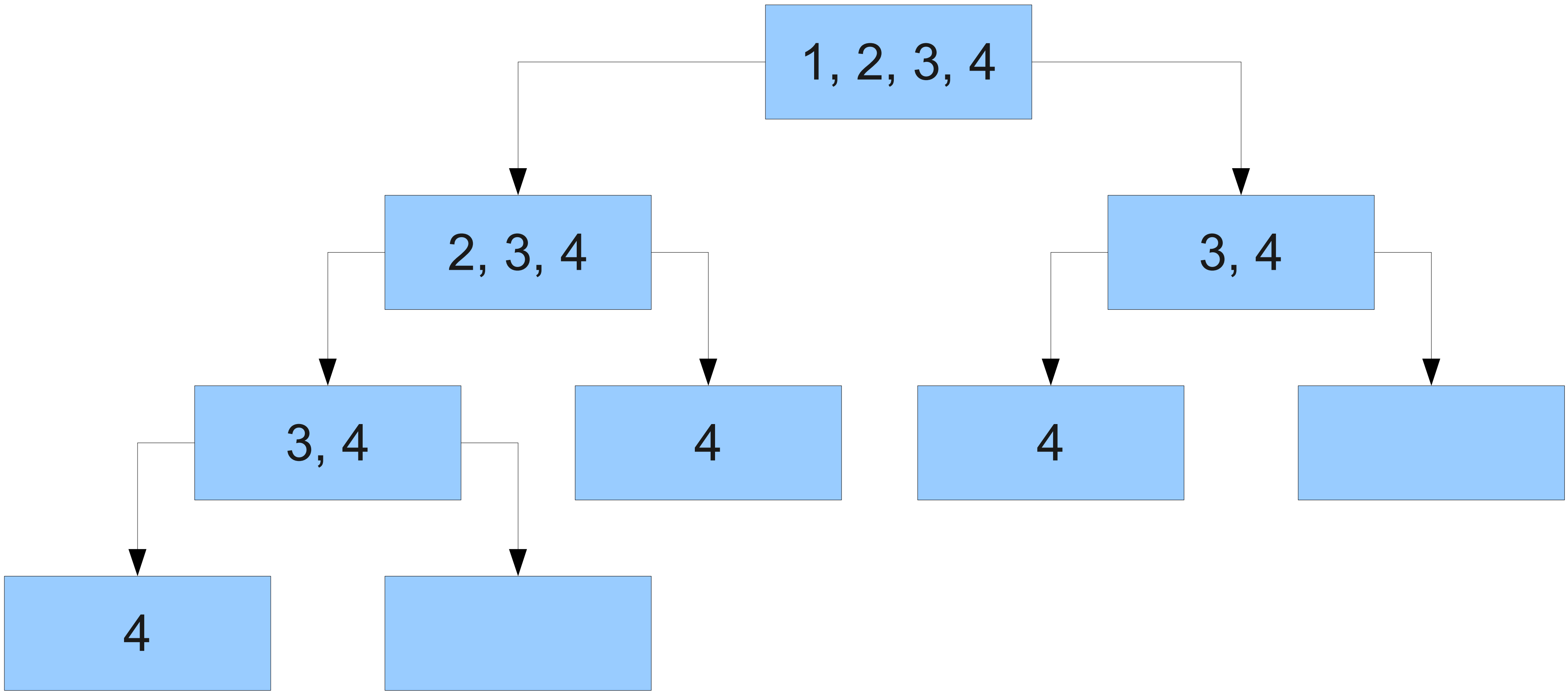

14 22 13 25 30 11 9
Maximize what's left in here.


14 22 13 25 30 11 9
Maximize what's left in here.

Revisiting our Solution

Introduction to Algorithmic Analysis

The Call Tree



Counting Recursive Calls

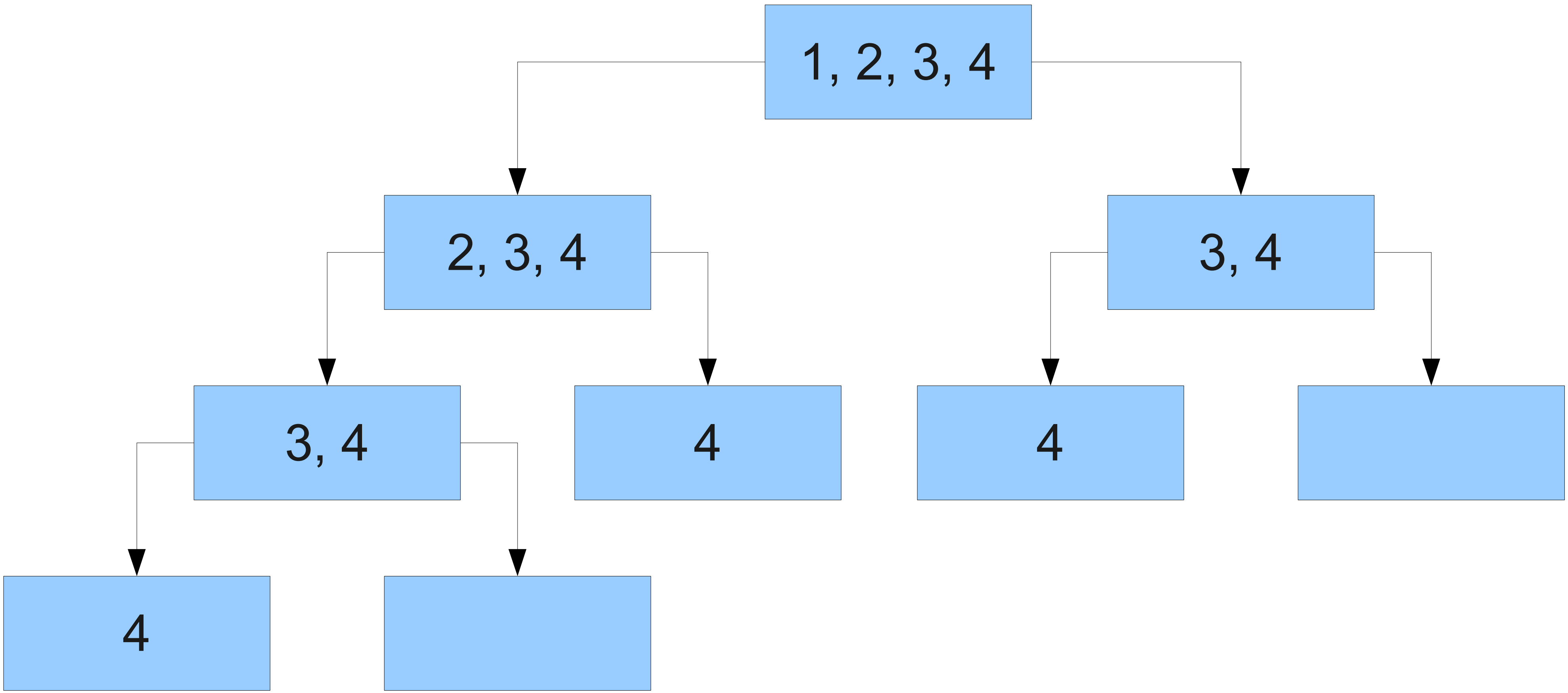
- Let n be the number of cities.
- Let $C(n)$ be the number of function calls made.
  - If $n = 0$, there is just one call, so $C(0) = 1$.
  - If $n = 1$, there is just one call, so $C(1) = 1$.
  - If $n \geq 2$, we have the initial function call, plus the two recursive calls. So $C(n) = 1 + C(n - 1) + C(n - 2)$.

Counting Recursive Calls

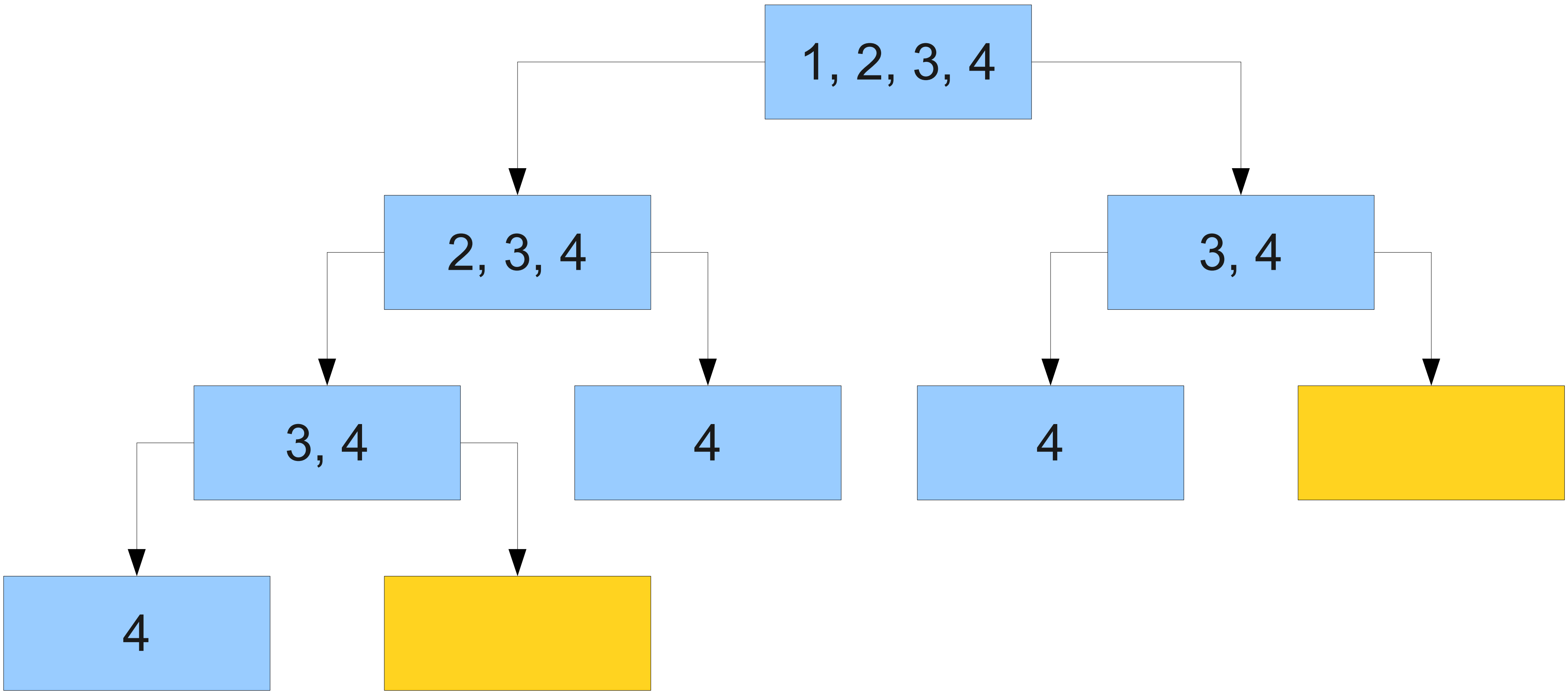
- C(0) = C(1) = 1.
- C(n) = C(n – 1) + C(n – 2)
- This gives the series

  1, 1, 3, 5, 9, 15, 25, 41, 67, 109, 177, 287, 465, 753, 1219, 1973, 3193, 5167, ...
- This function grows very quickly, so our solution will scale very poorly.
- Neat mathematical aside – these numbers are called the **Leonardo numbers**.

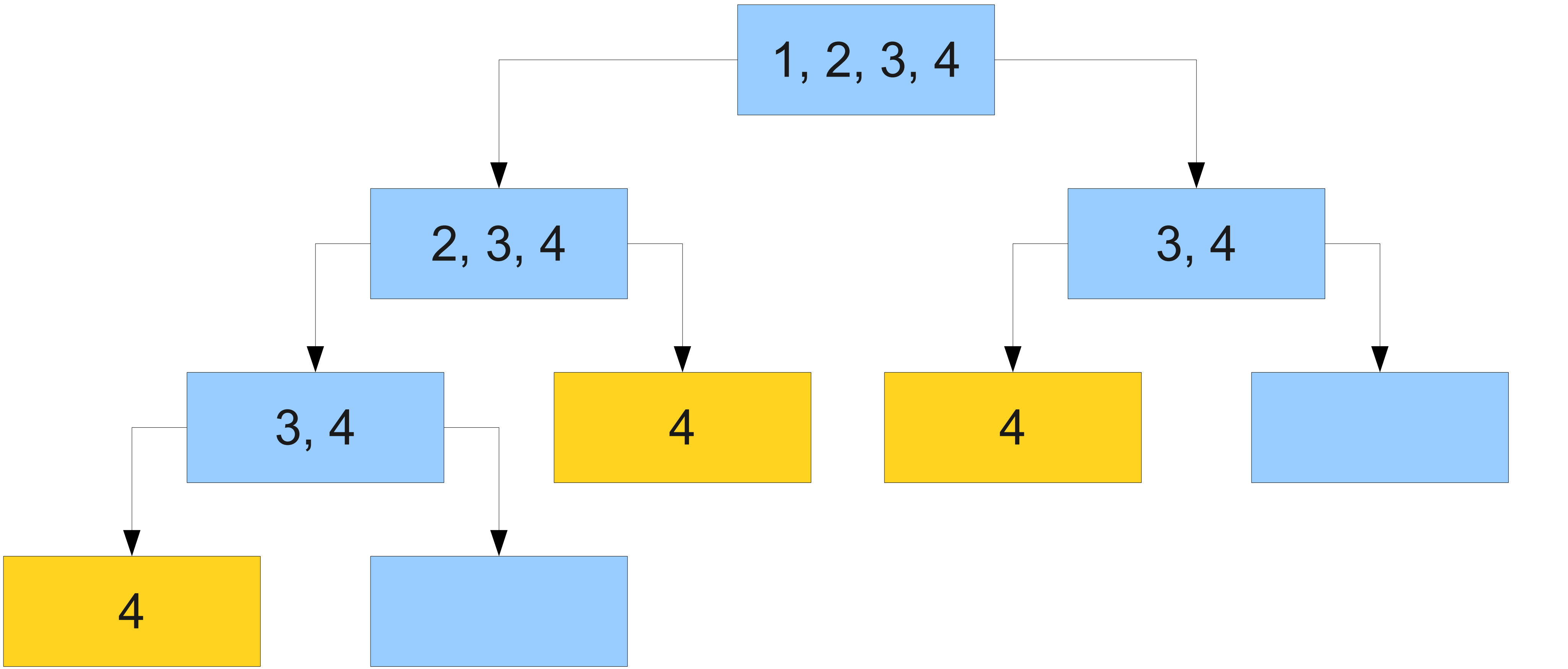
The Call Tree



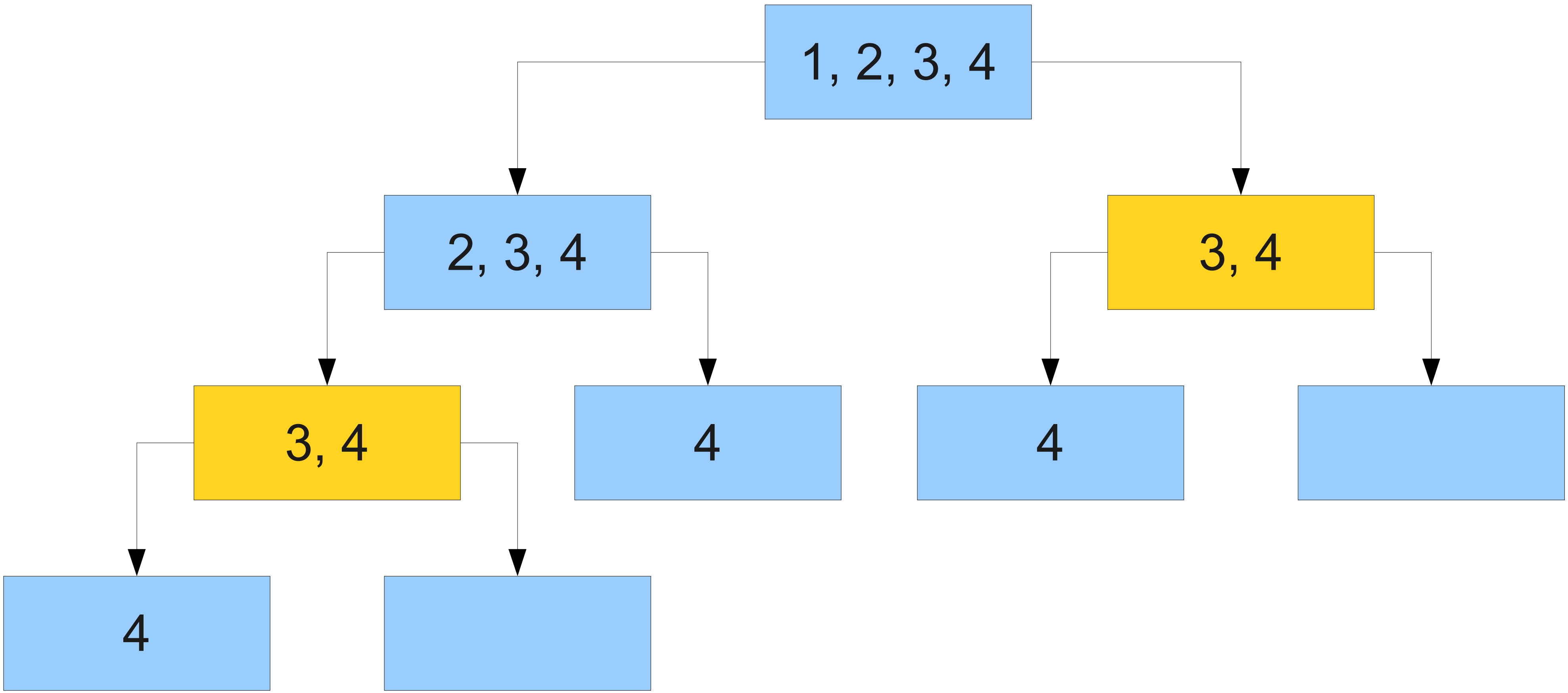
The Call Tree



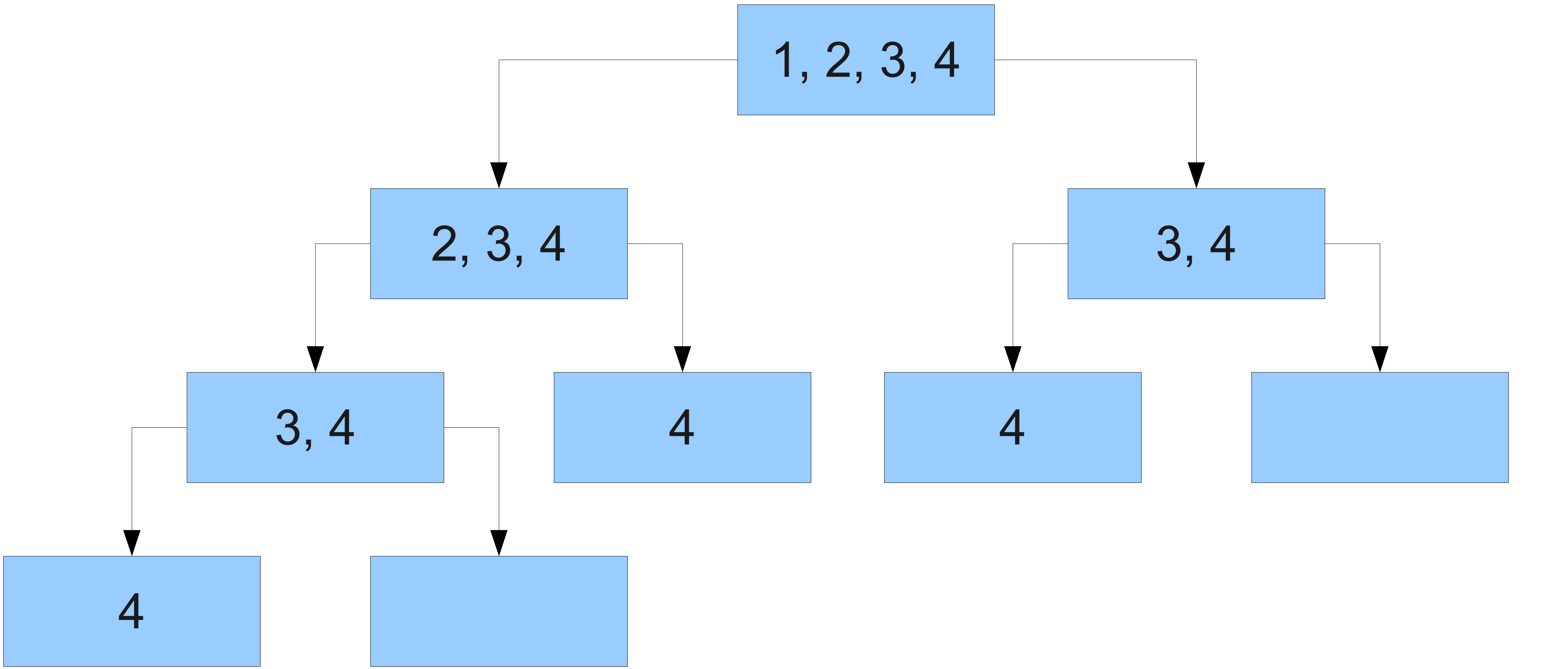
The Call Tree



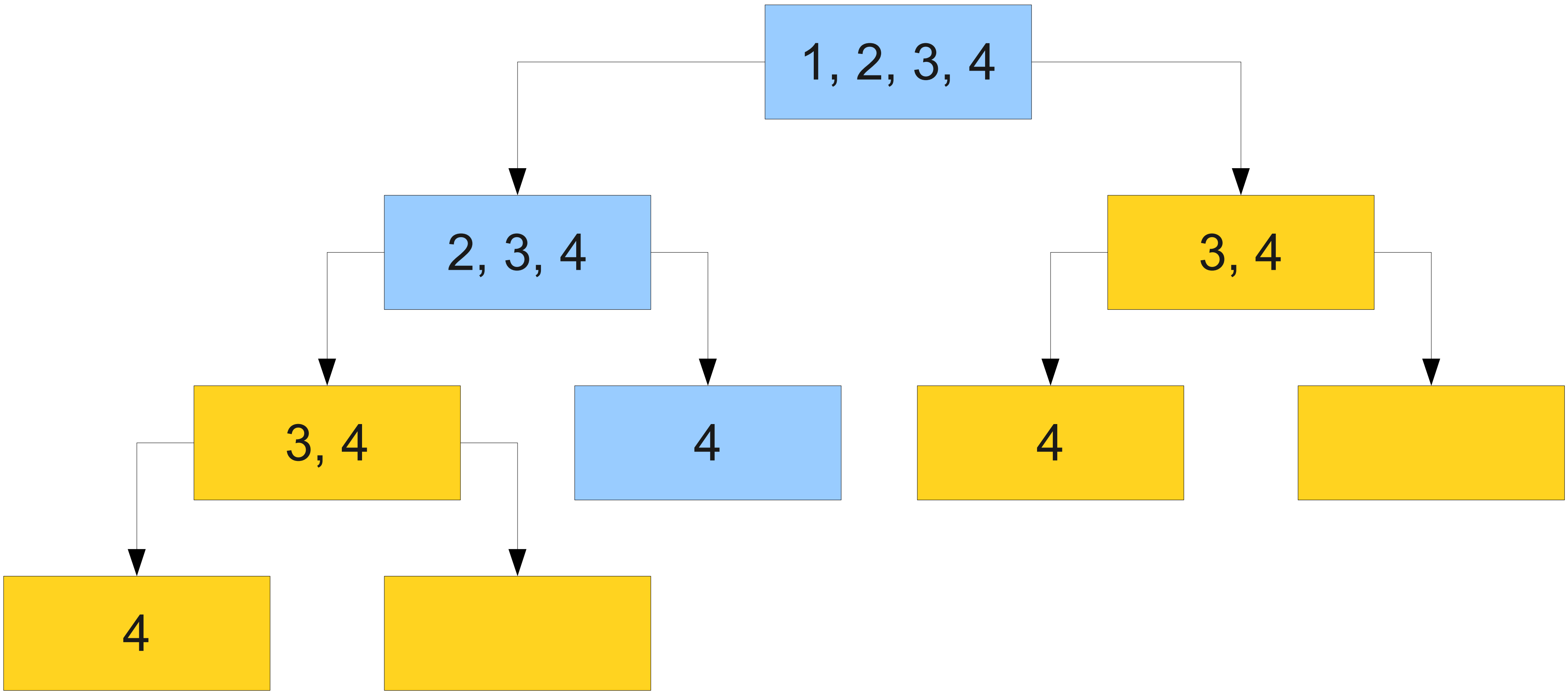
The Call Tree



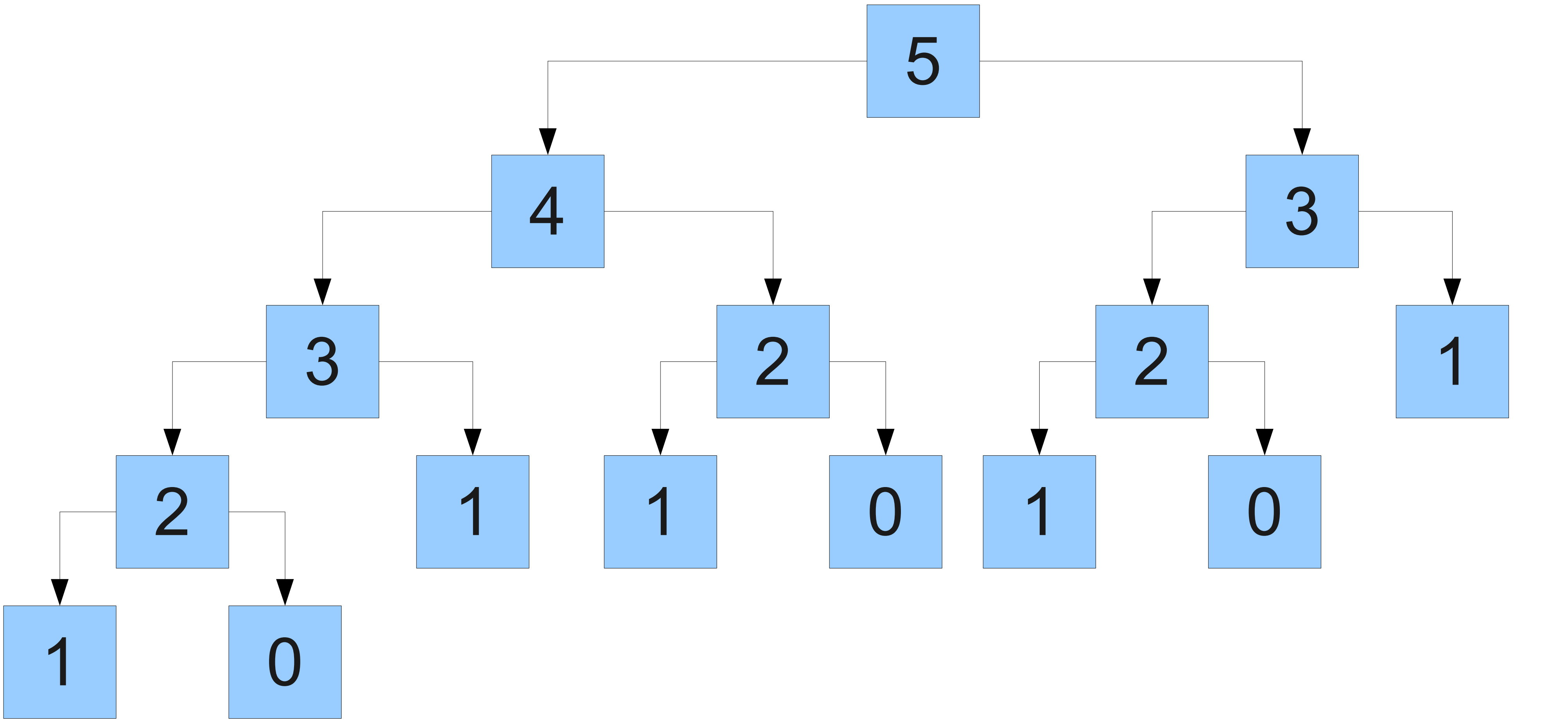
The Call Tree



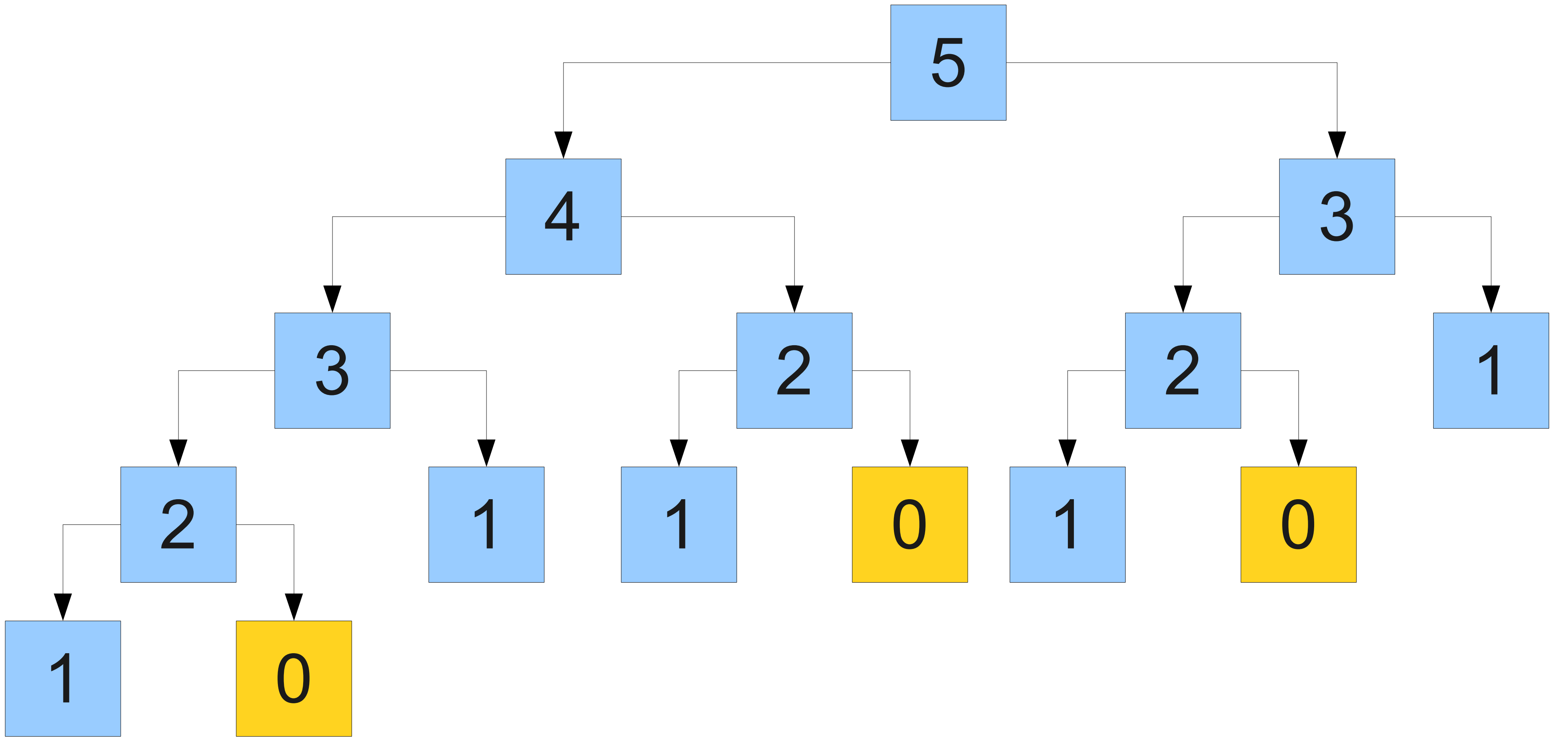
The Call Tree



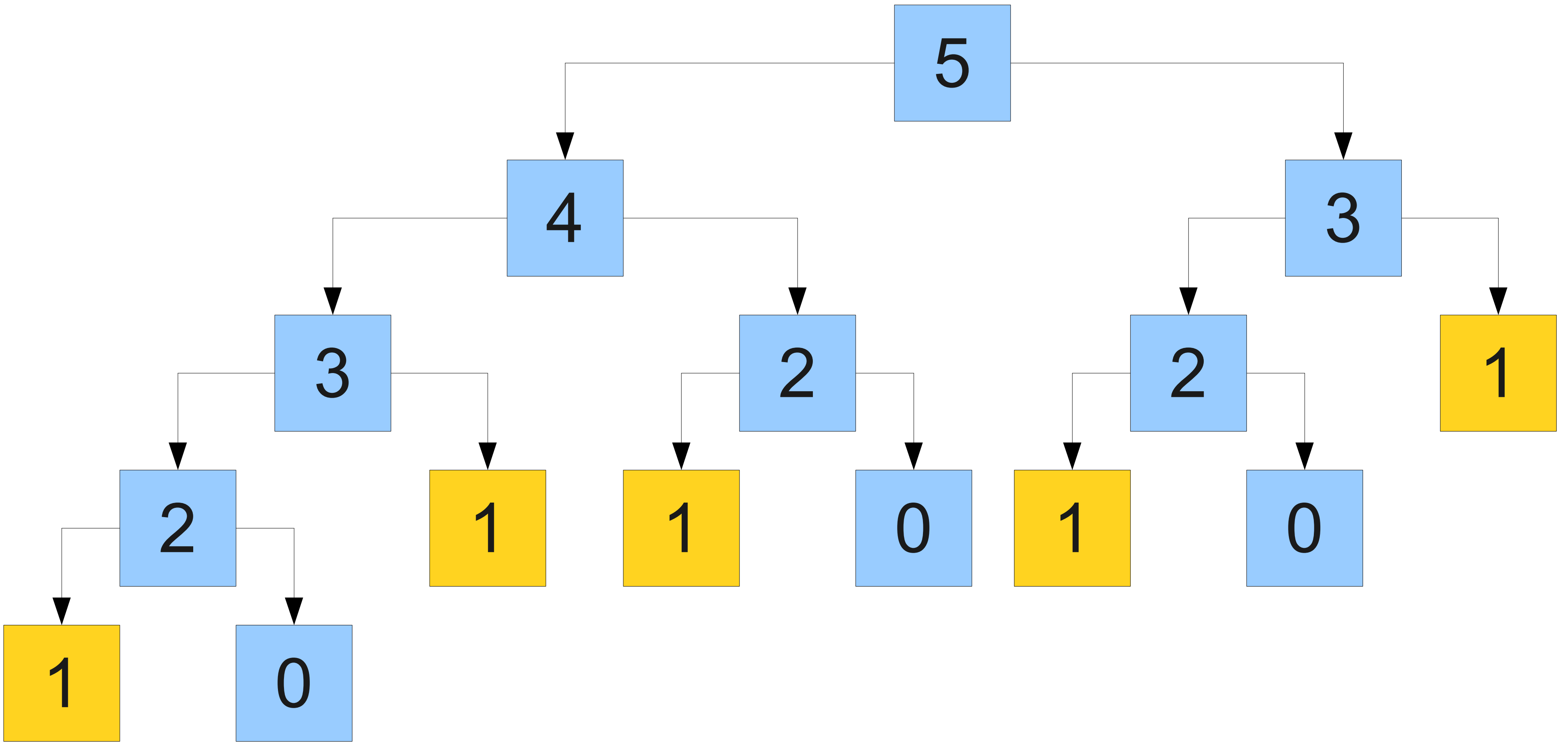
A Bigger Call Tree



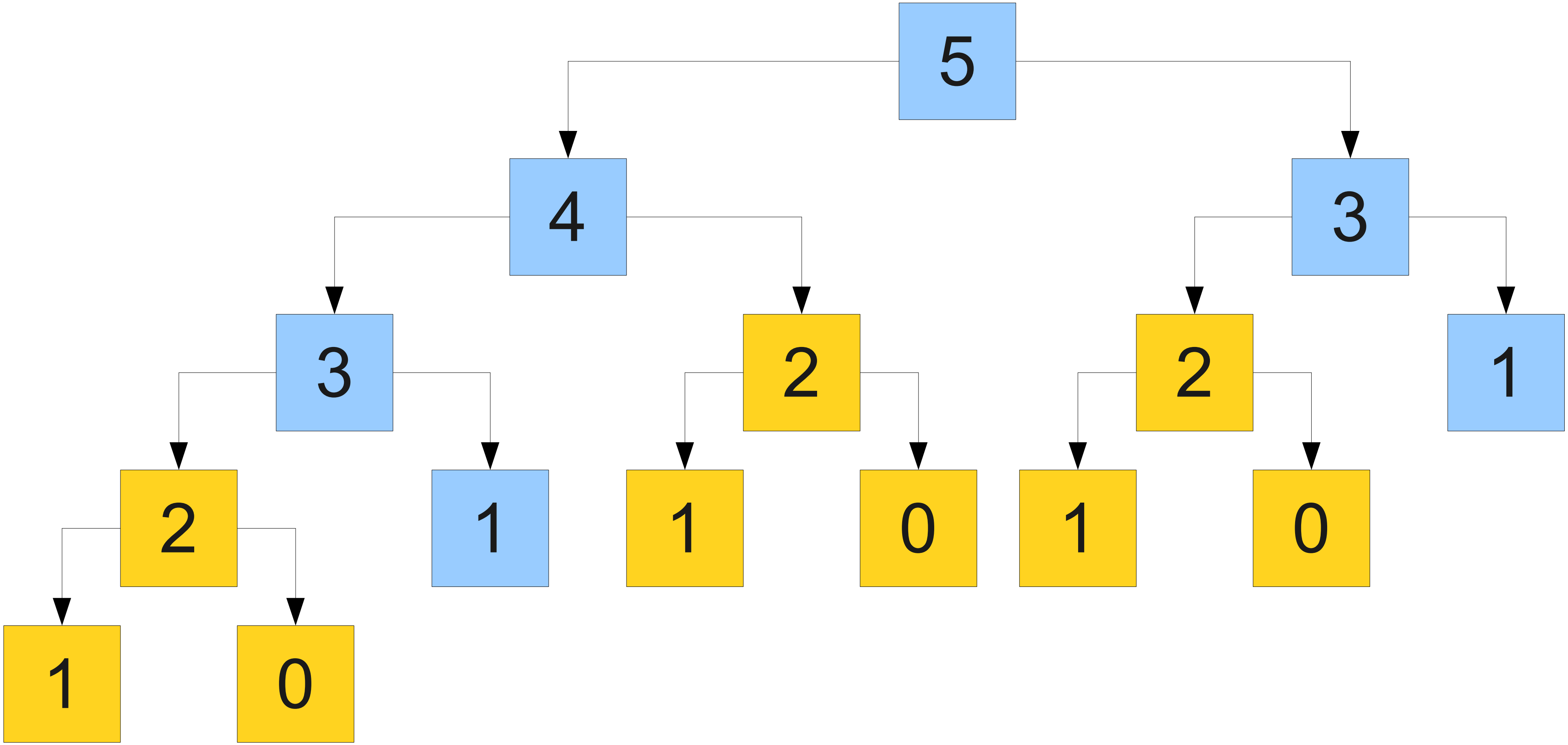
A Bigger Call Tree



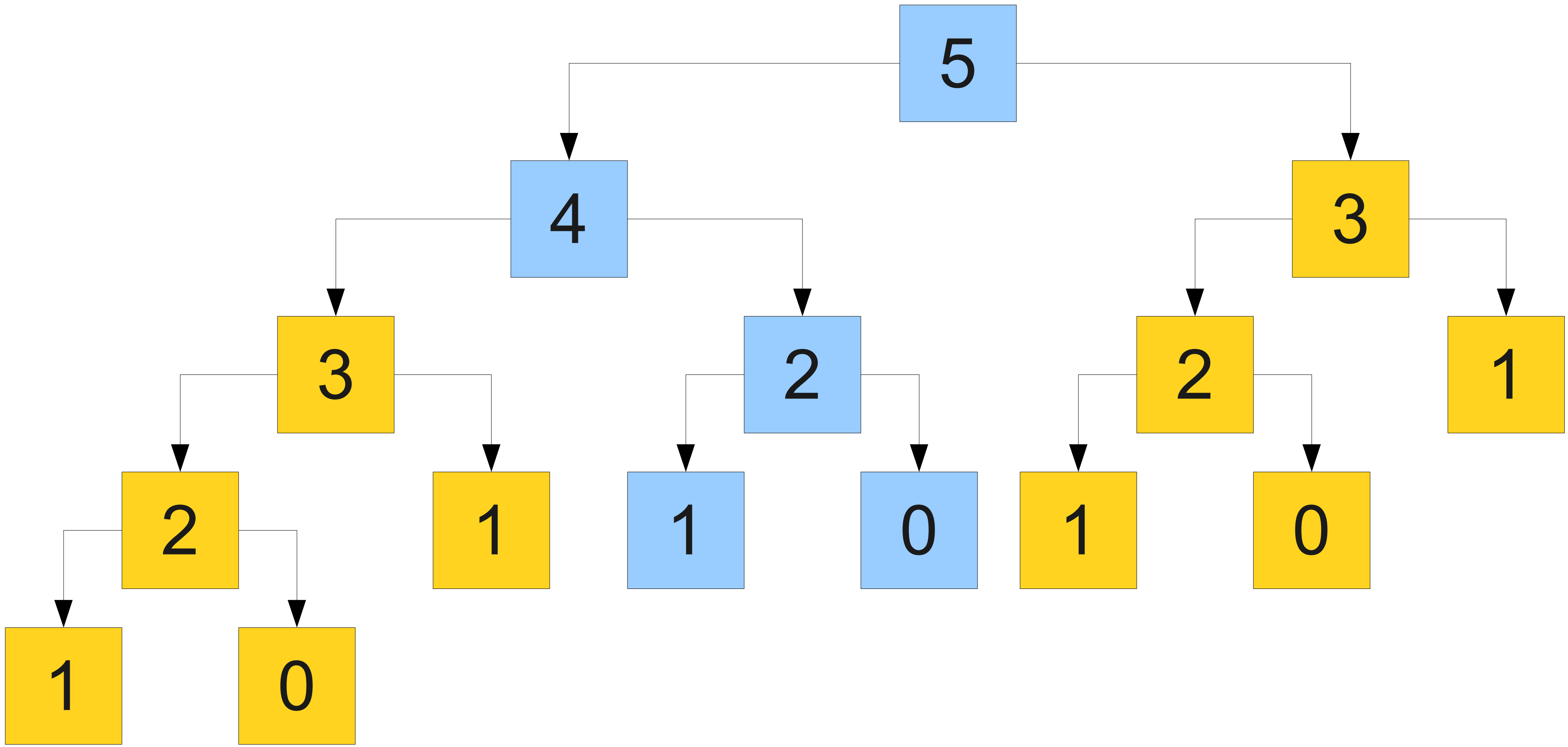
A Bigger Call Tree



A Bigger Call Tree



A Bigger Call Tree



We're doing completely unnecessary work!
Can we do better?

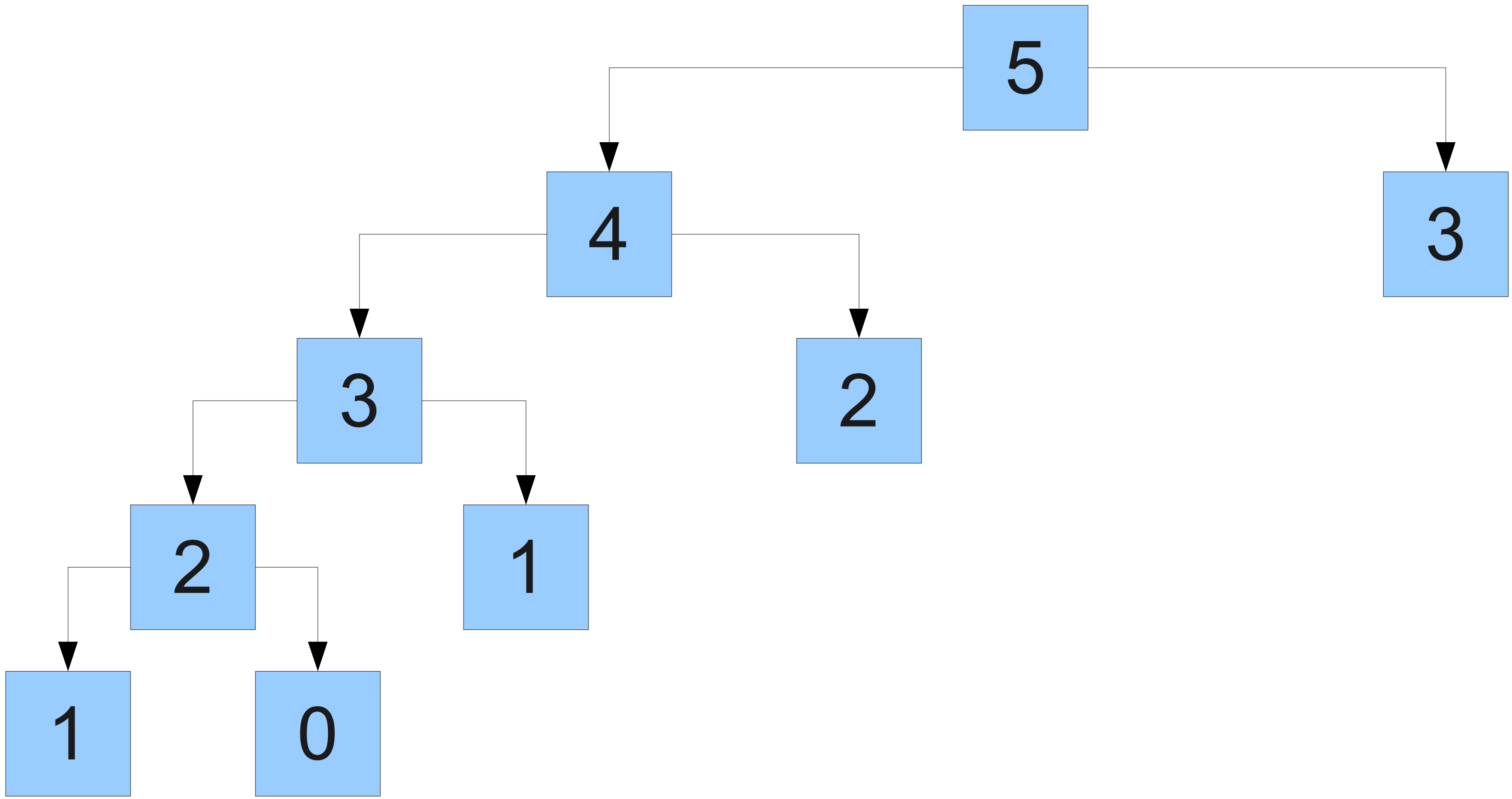
What Just Happened?

- **Remember what values we've computed so far.**
- New base case: If we already computed the answer, we're done.
- When computing a recursive step, record the answer before we return it.
- This is called **memoization**.
  - No, that is not a typo – there's no “r” in memoization.

Memoization

- Memoization is useful if
  - you make a large number of recursive calls
  - with exactly the same arguments.
- Not a “silver bullet” to speed things up, but when applicable can have huge performance implications.

Memoized Recursion



Next Time

- **Algorithmic Analysis**
  - How can we predict the behavior of an algorithm on inputs we haven't seen?
  - How can we quantitatively rank algorithms against one another?