Thinking Recursively

Part V

Outline for Today

- ***Recursive Backtracking***
  - Finding a needle in a haystack.
- ***On Tenacity***
  - Computational grit!
- ***Optional<T>***
  - Sending data out of functions.
- ***CHeMoWiZrDy***
  - Having some fun with the periodic table.

A Warm-Up Exercise

What's Wrong With This Code?

```
bool containsE(const string& str) {
    for (char ch: str) {
        return ch == 'e' || ch == 'E';
    }
    return false;
}
```

Answer at
https://pollev.com/cs106bwin23

What's Wrong With This Code?

```
bool containsE(const string& str) {
    for (char ch: str) {
        return ch == 'e' || ch == 'E';
    }
    return false;
}
```

It's exceedingly rare to have an unconditional return statement in a for loop. This almost certainly indicates the presence of a bug.

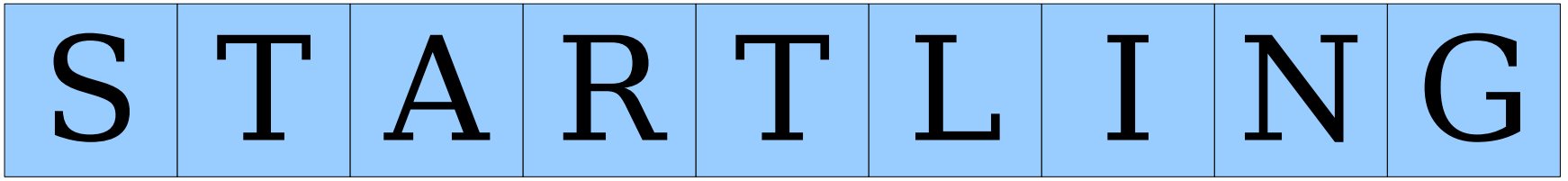
Specifically, this code makes its final decision based on the first character of the string.

Recap from Last Time

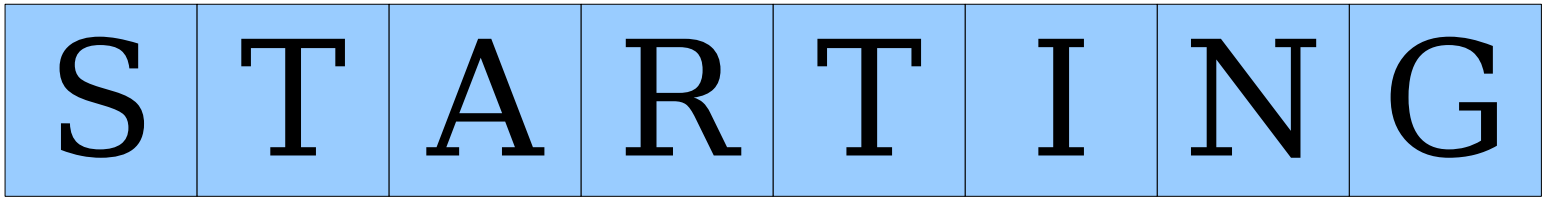
A Little Word Puzzle

“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?”

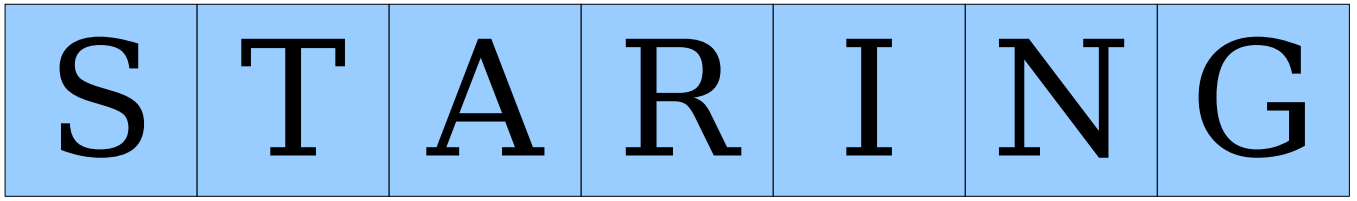
One Solution



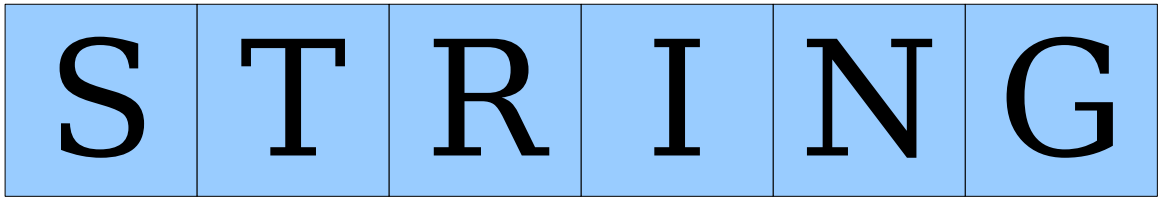
One Solution



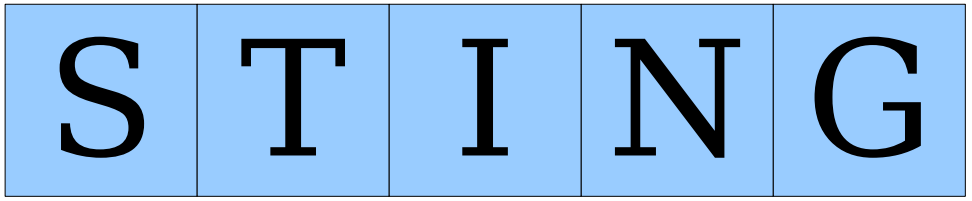
One Solution



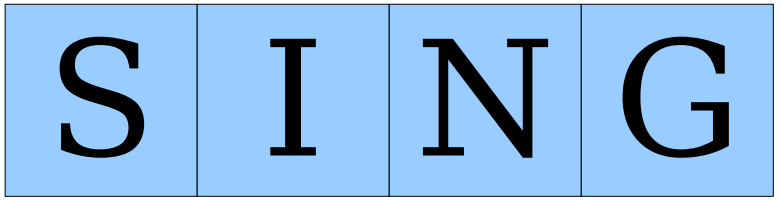
One Solution



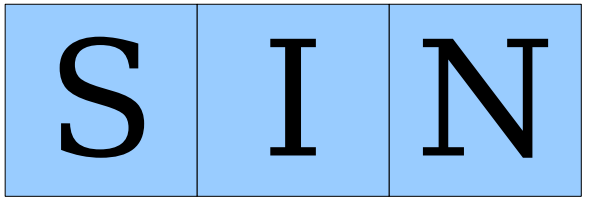
One Solution



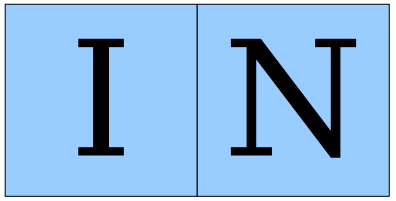
One Solution



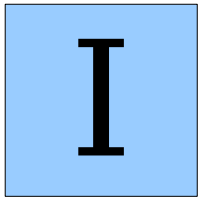
One Solution



One Solution



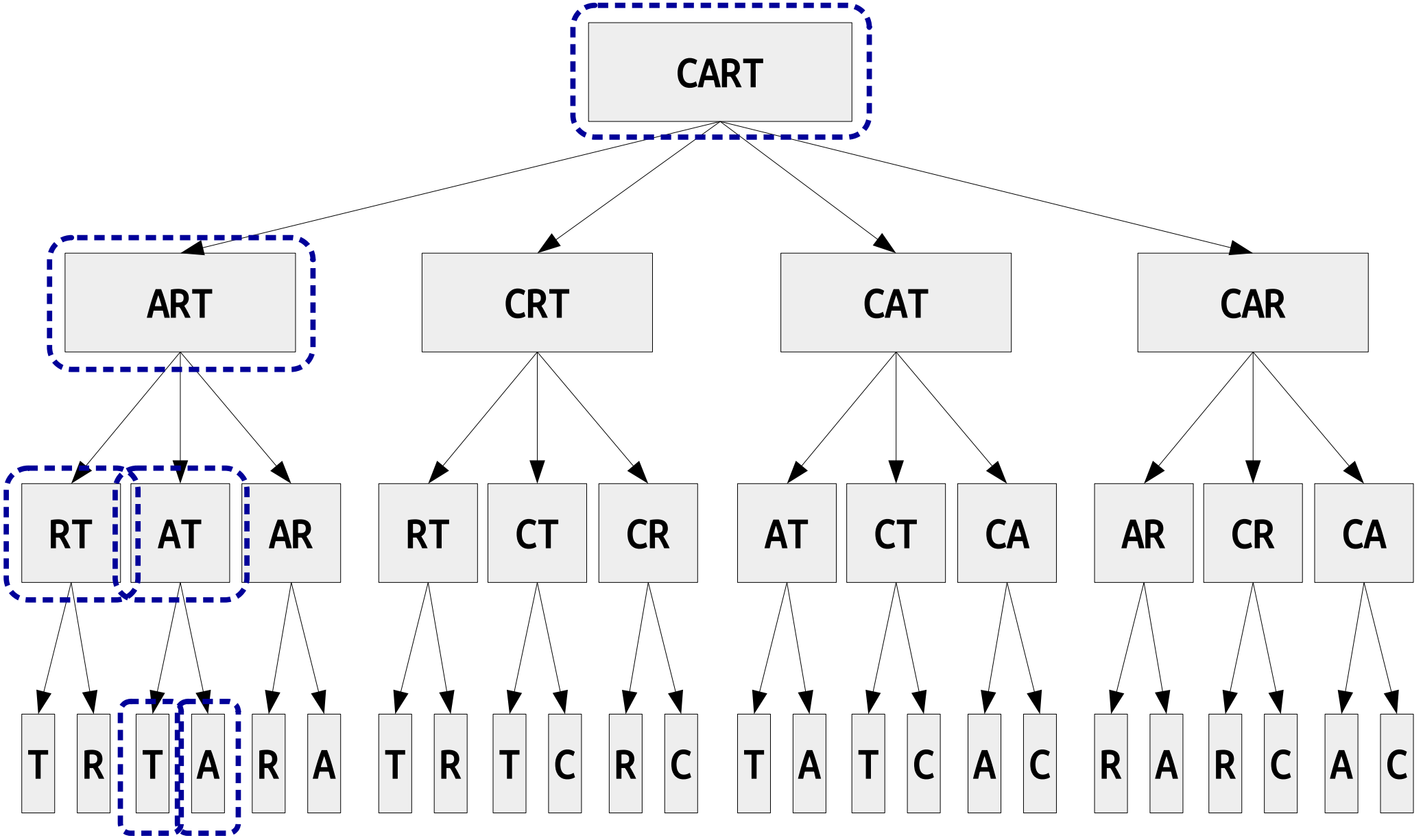
One Solution



New Stuff!

Our Solution, In Action

The Incredible Shrinking Word



```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        if (isShrinkableWord(shrunken, english)) {
            return true;
        }
    }
    return false;
}
```

```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        if (isShrinkableWord(shrunken, english)) {
            return true;
        }
    }
    return false;
}
```

```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        return isShrinkableWord(shrunken, english); // Bad idea!
    }
    return false;
}
```

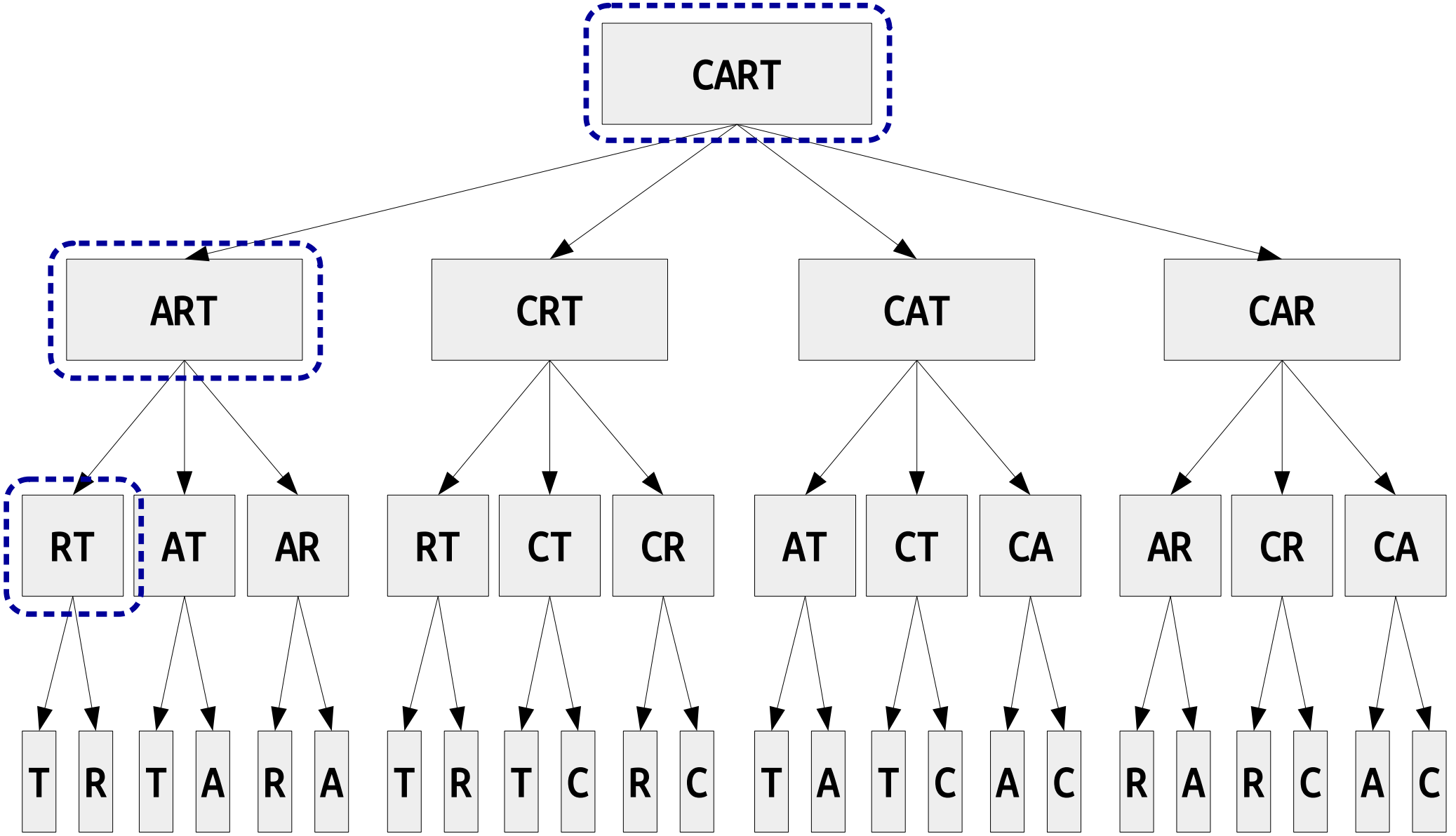
```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        return isShrinkableWord(shrunken, english); // Bad idea!
    }
    return false;
}
```

It's exceedingly rare to have an unconditional return statement in a for loop. This almost certainly indicates the presence of a bug.

Specifically, this code makes its final decision based on the first character it tries removing.

Tenacity is a Virtue



When backtracking recursively,

don't give up if your first try fails!

Hold out hope that something else will work out. It very well might!

Recursive Backtracking

```
if (problem is sufficiently simple) {
    return whether the problem is solvable
} else {
    for (each choice) {
        try out that choice
        if (that choice leads to success) {
            return success;
        }
    }
    return failure;
}
```

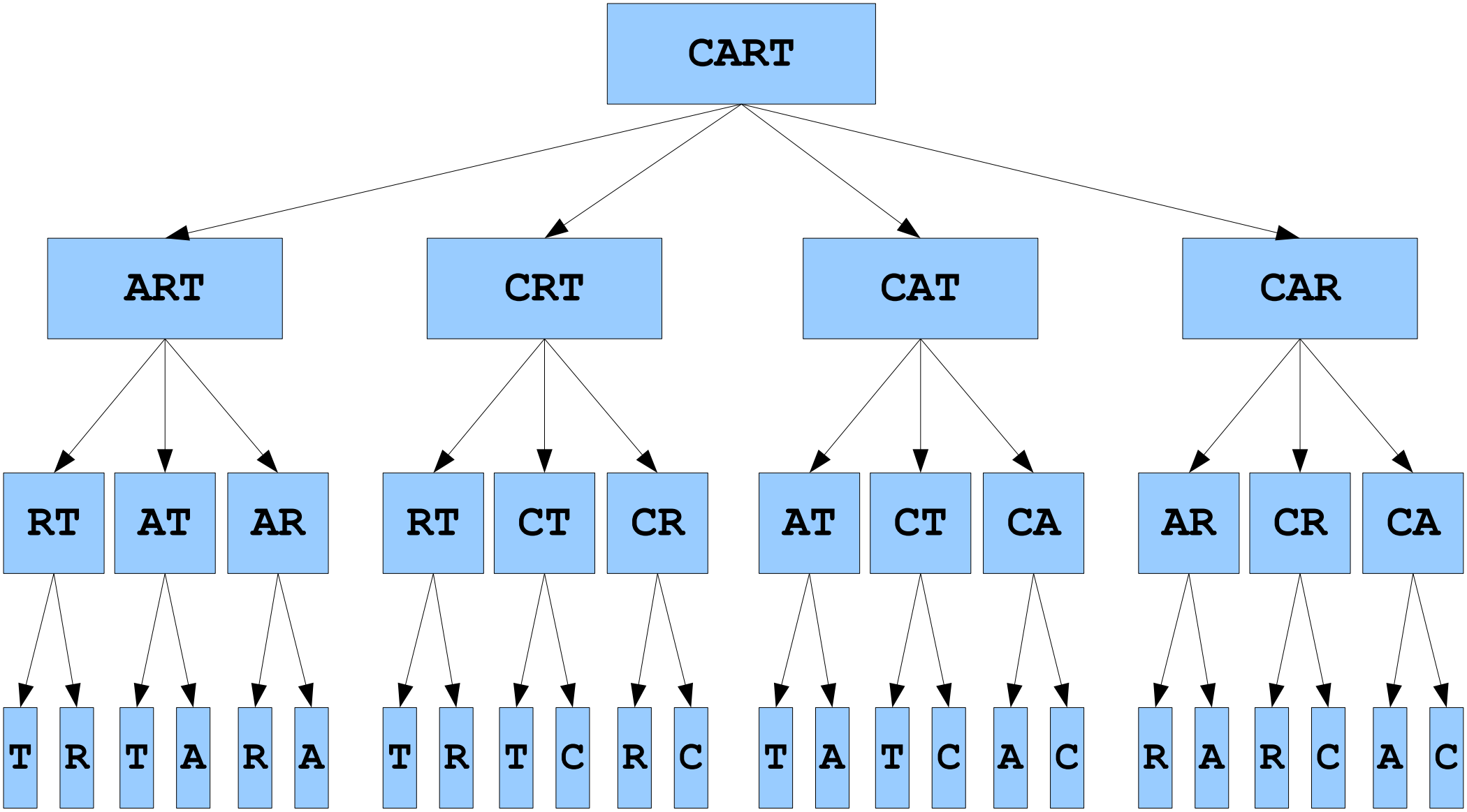
Note that *if* the recursive call succeeds, *then* we return success. If it doesn't succeed, that doesn't mean we've failed – it just means we need to try out the next option.

How do we know we’re correct?

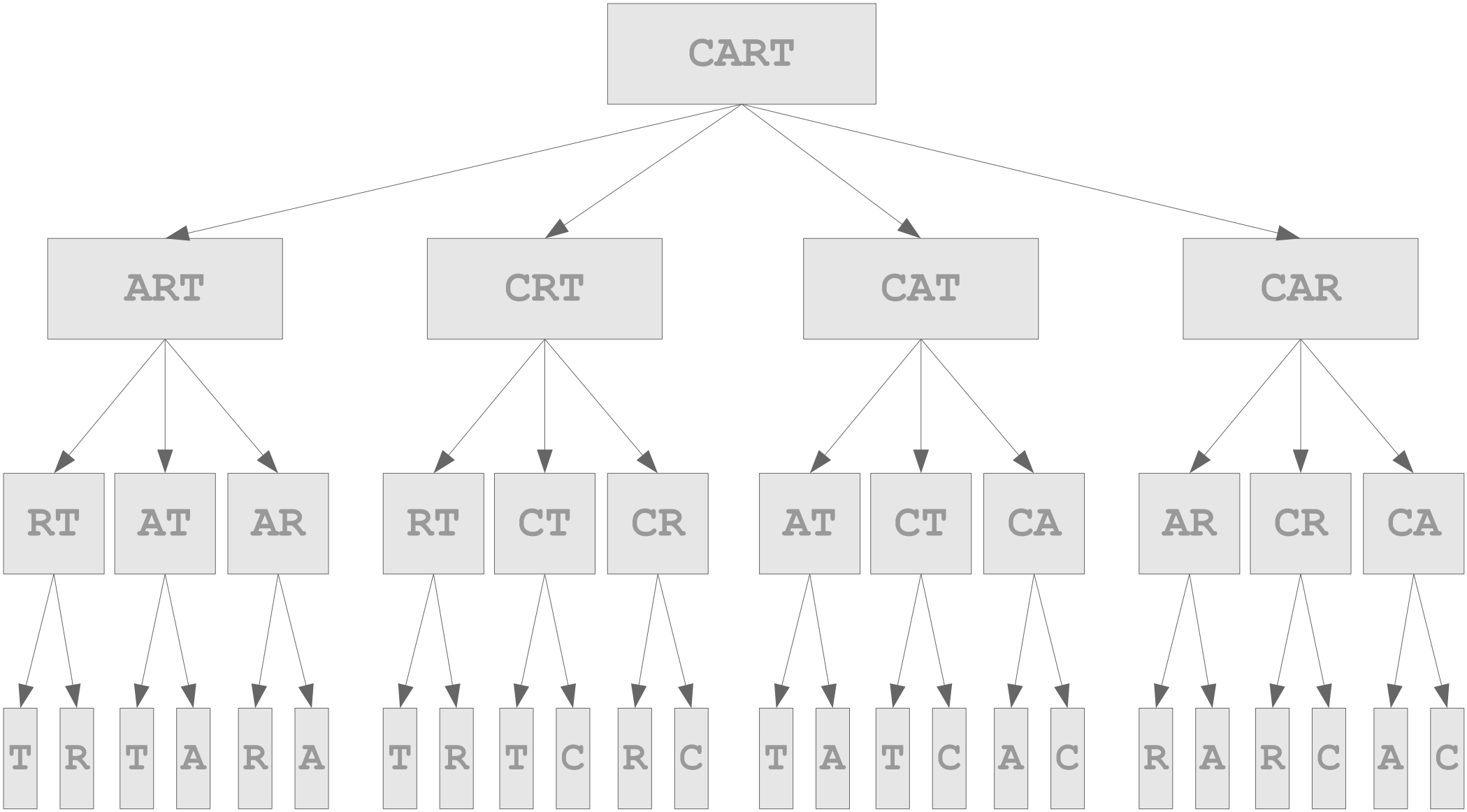
Optional<T>

- The `Optional<T>` type represents either an object of type `T` or is `Nothing` at all.
- It's useful when working with recursive functions that look for something that may or may not exist.
  - If a solution exists, return it as usual.
  - Otherwise, return `Nothing`.
- If the `Optional<T>` is a value of type `T`, you can call the `.value()` function to retrieve the underlying value.

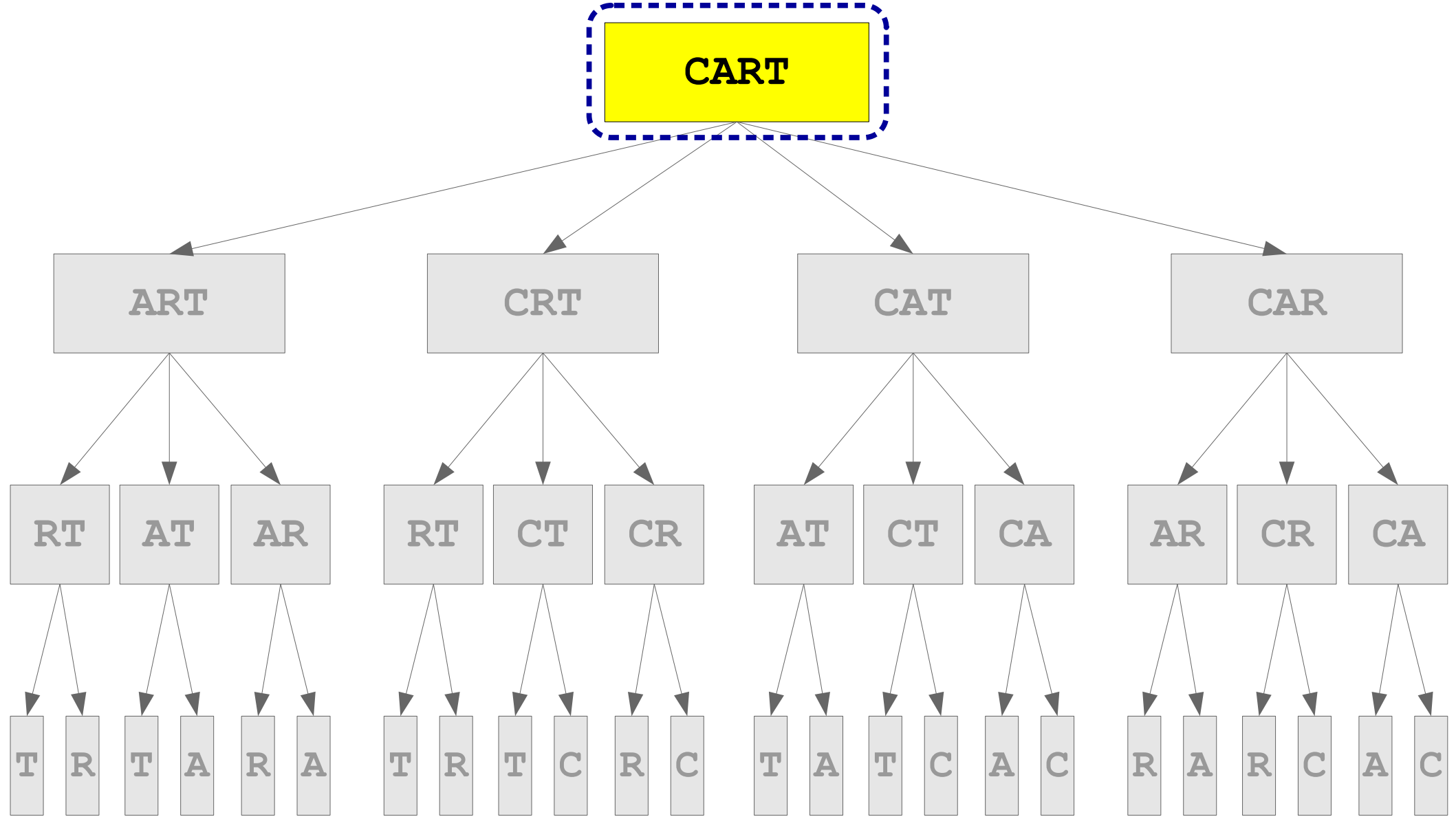
Generating the Answer



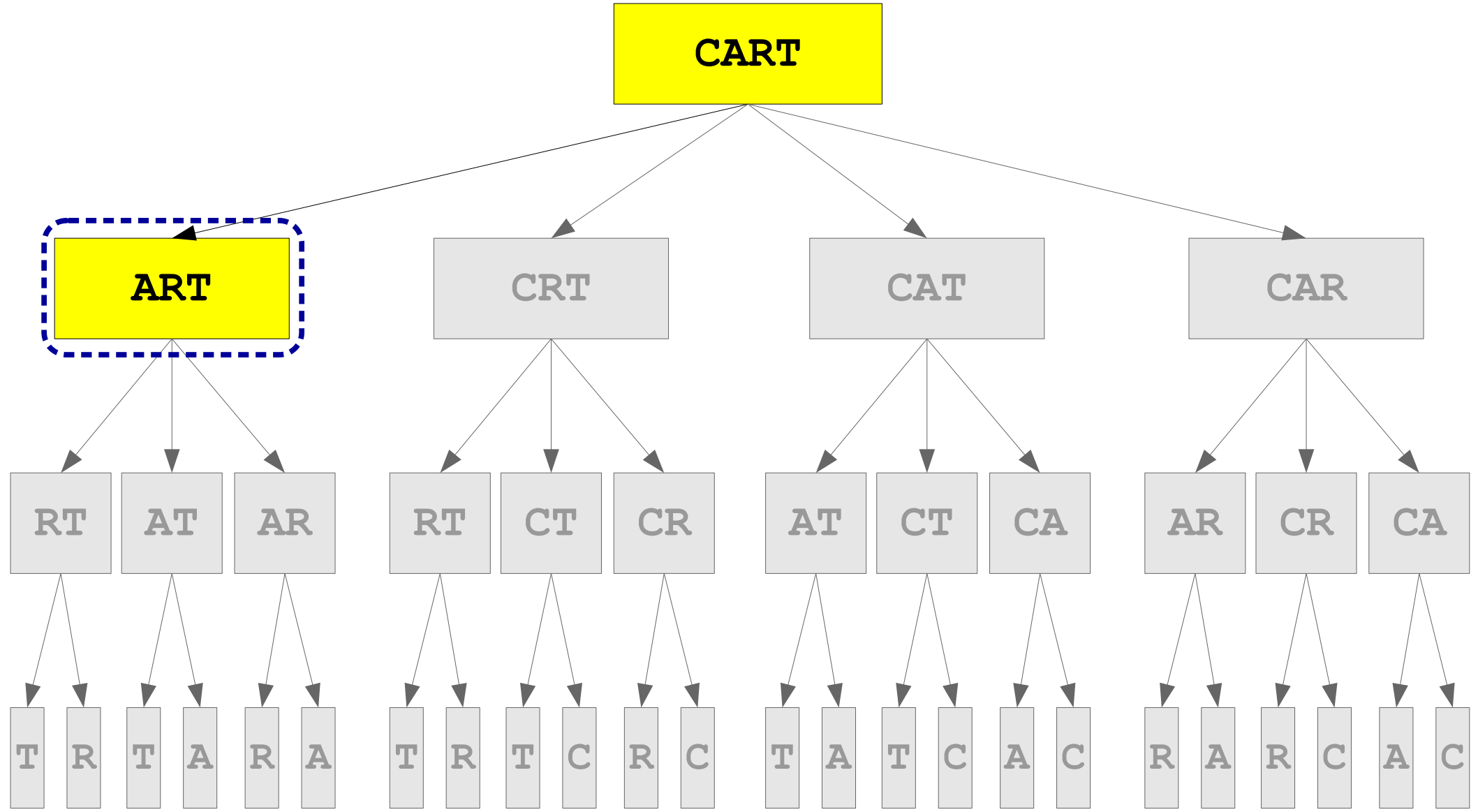
Generating the Answer



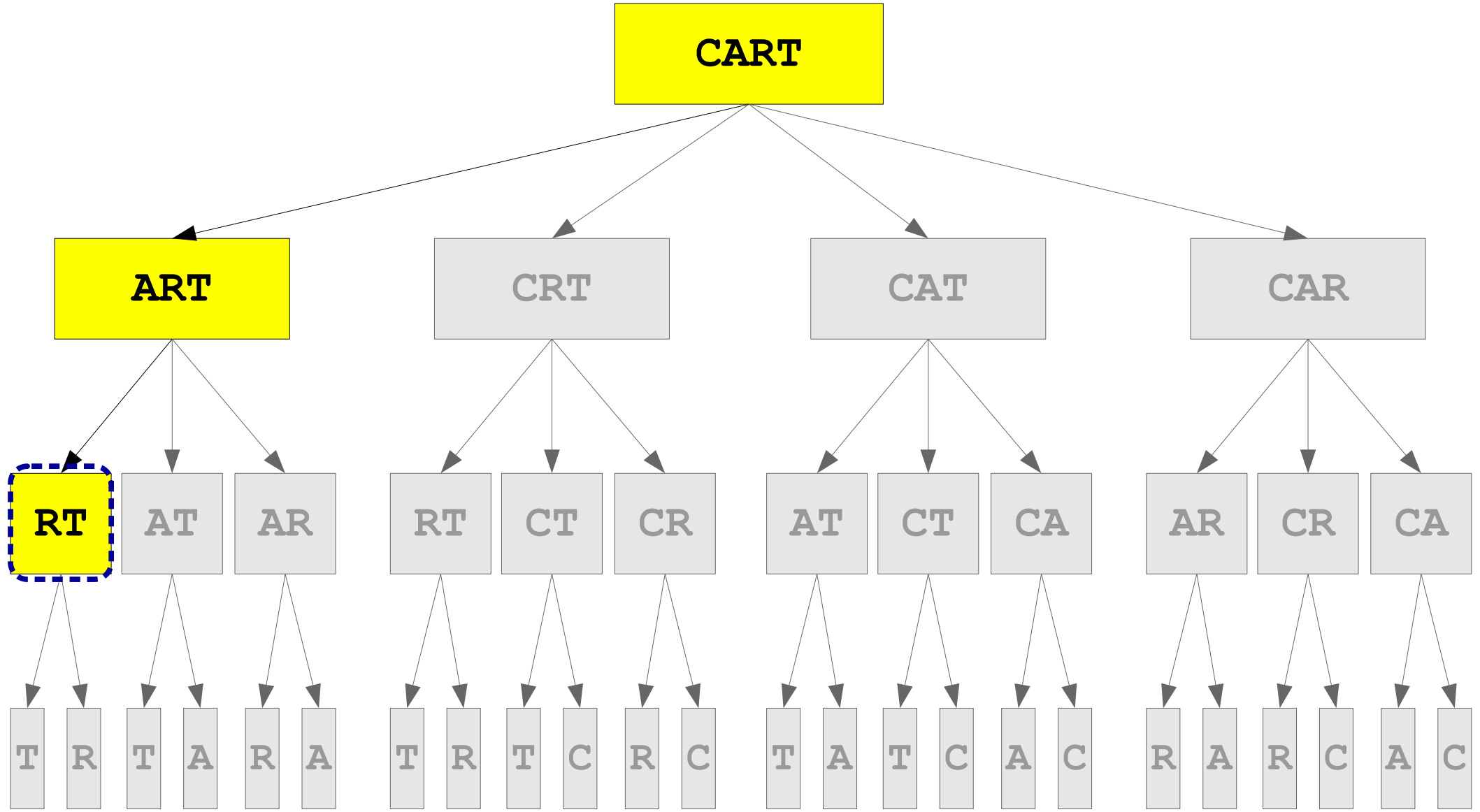
Generating the Answer



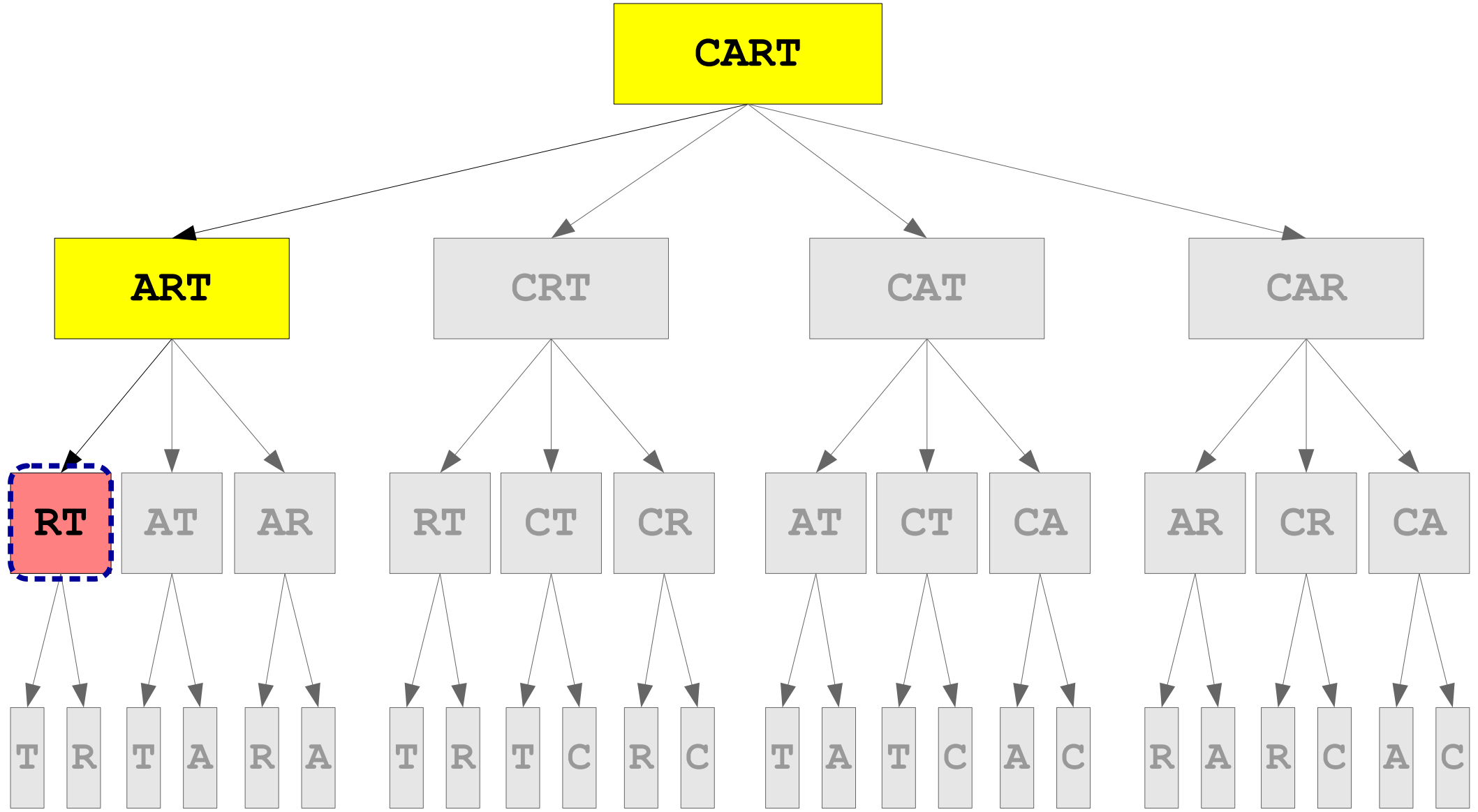
Generating the Answer



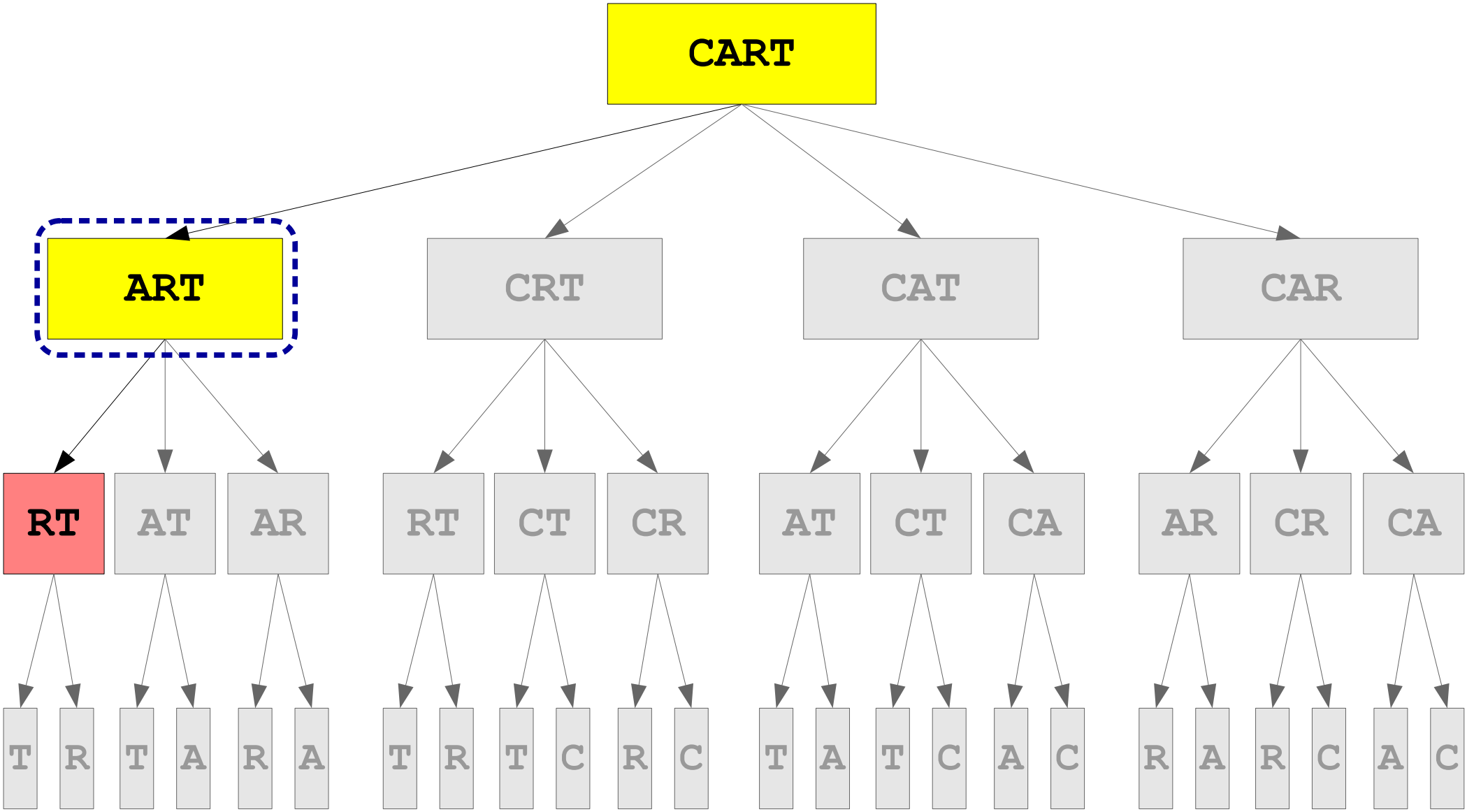
Generating the Answer



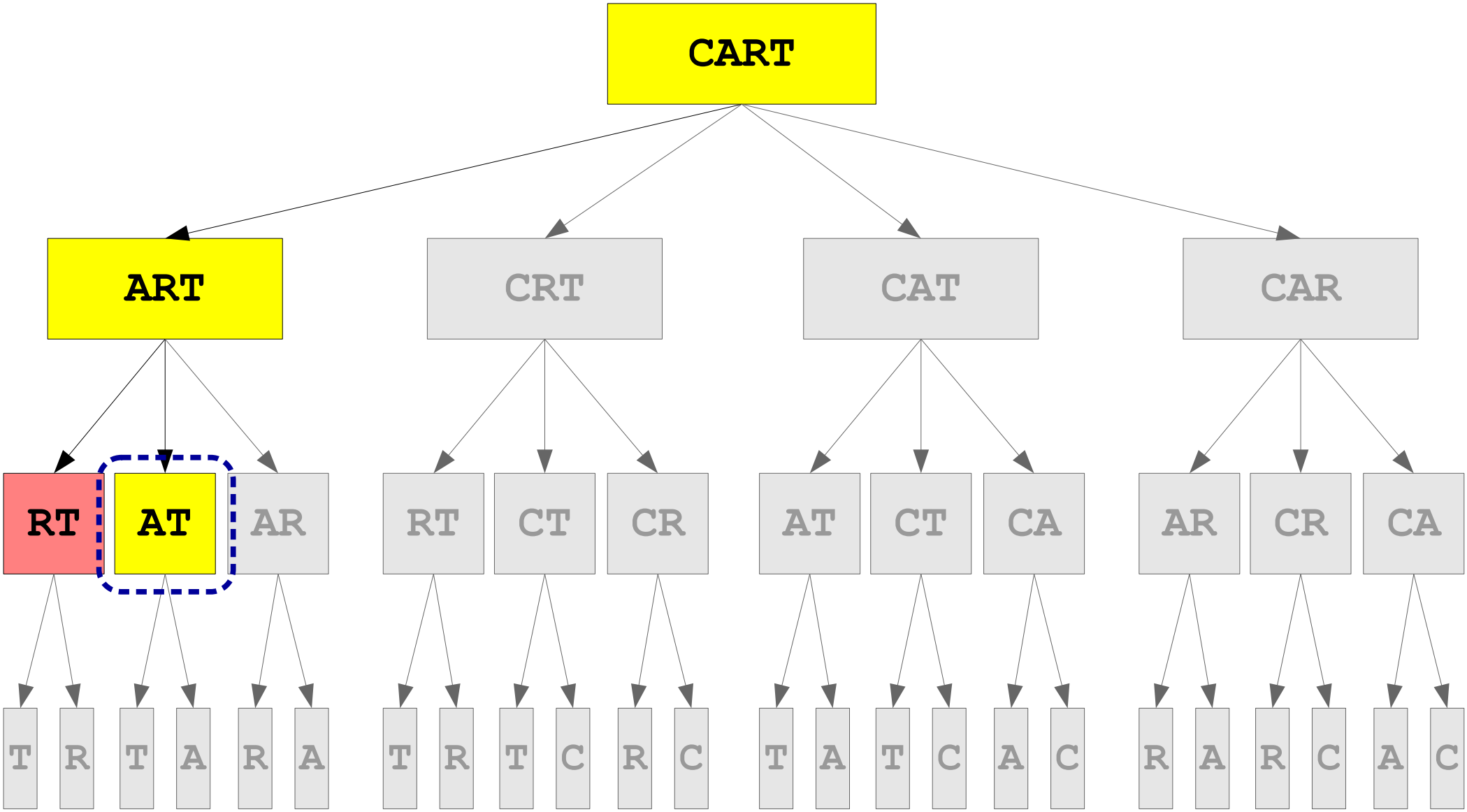
Generating the Answer



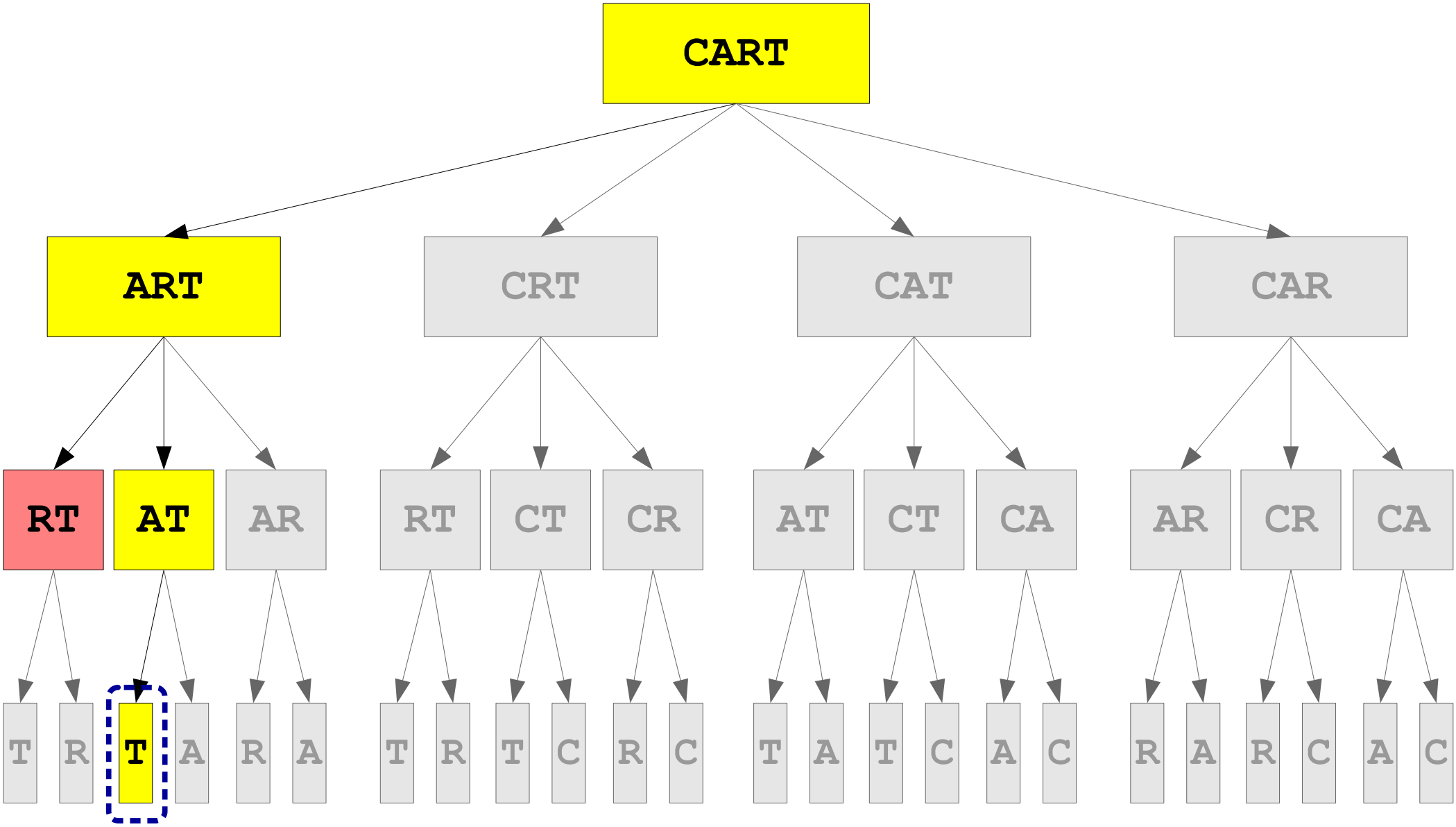
Generating the Answer



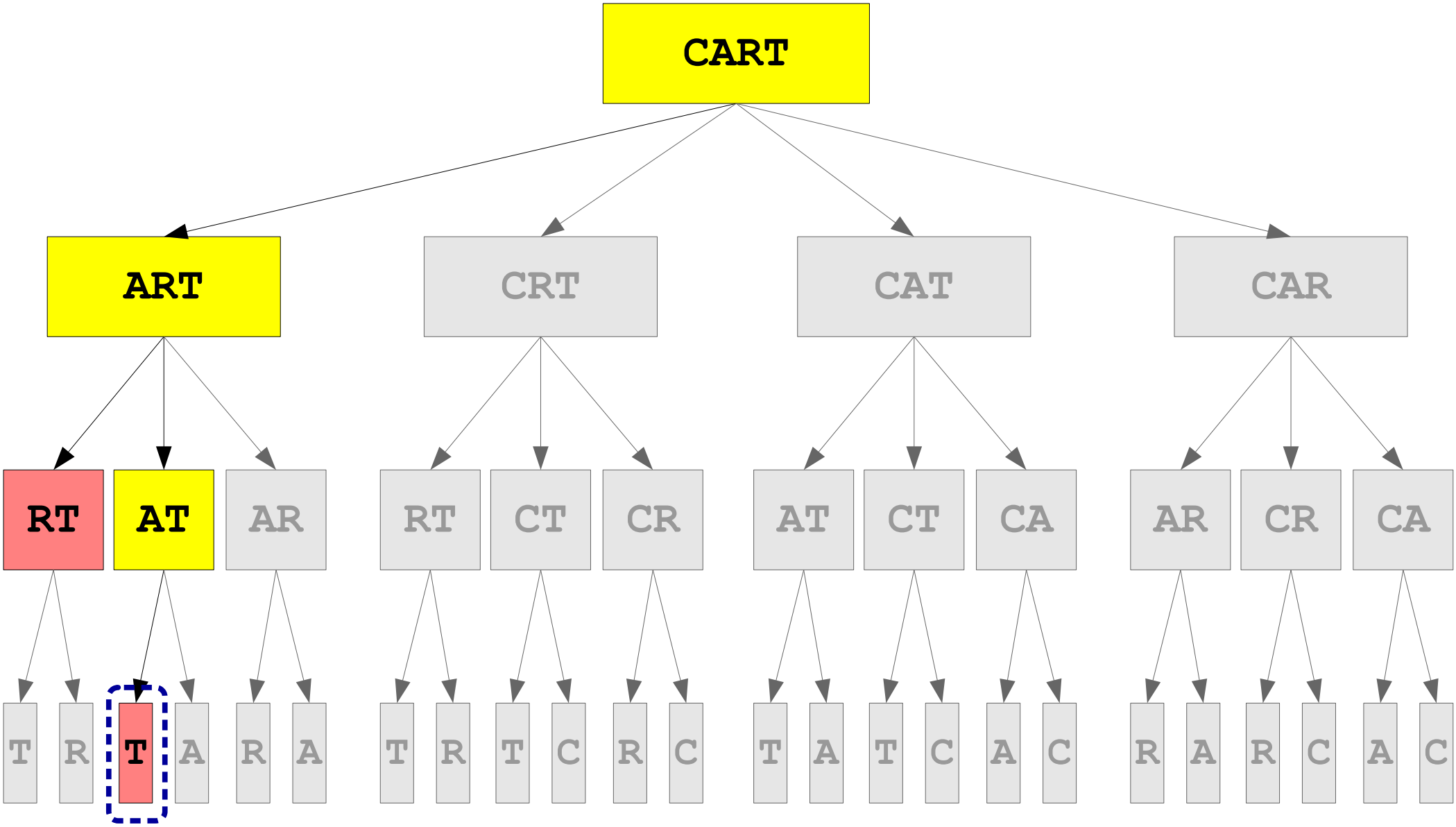
Generating the Answer



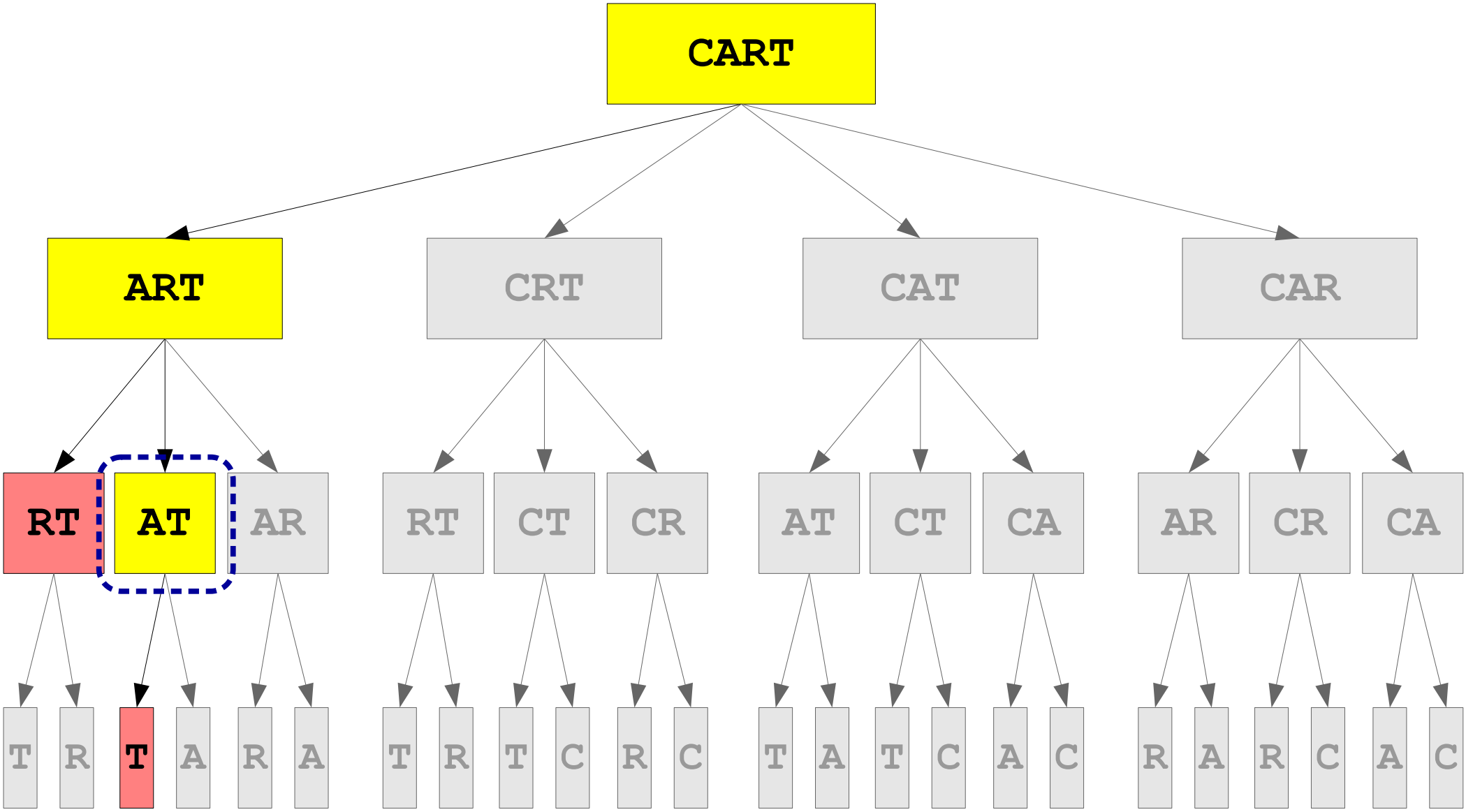
Generating the Answer



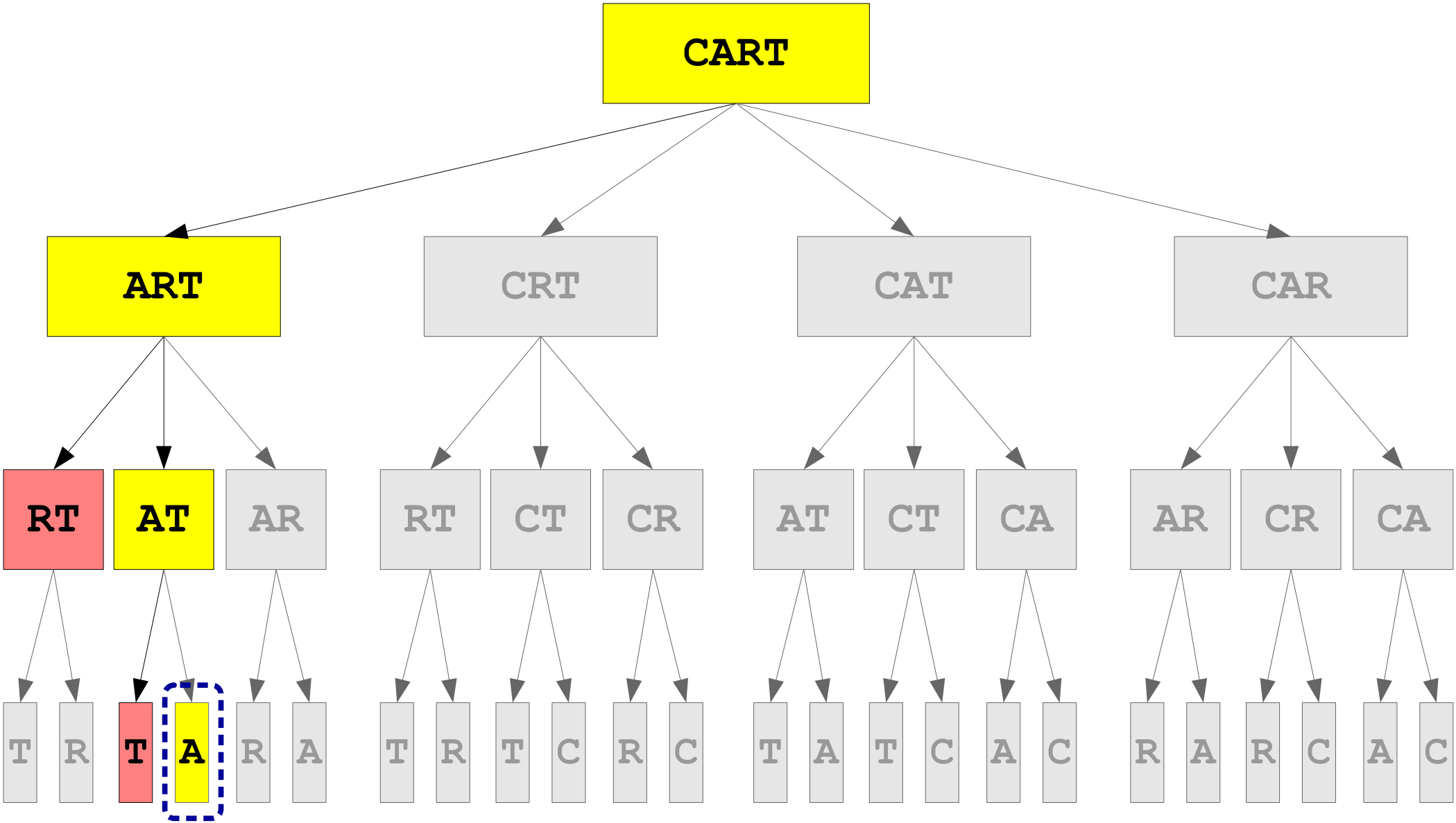
Generating the Answer



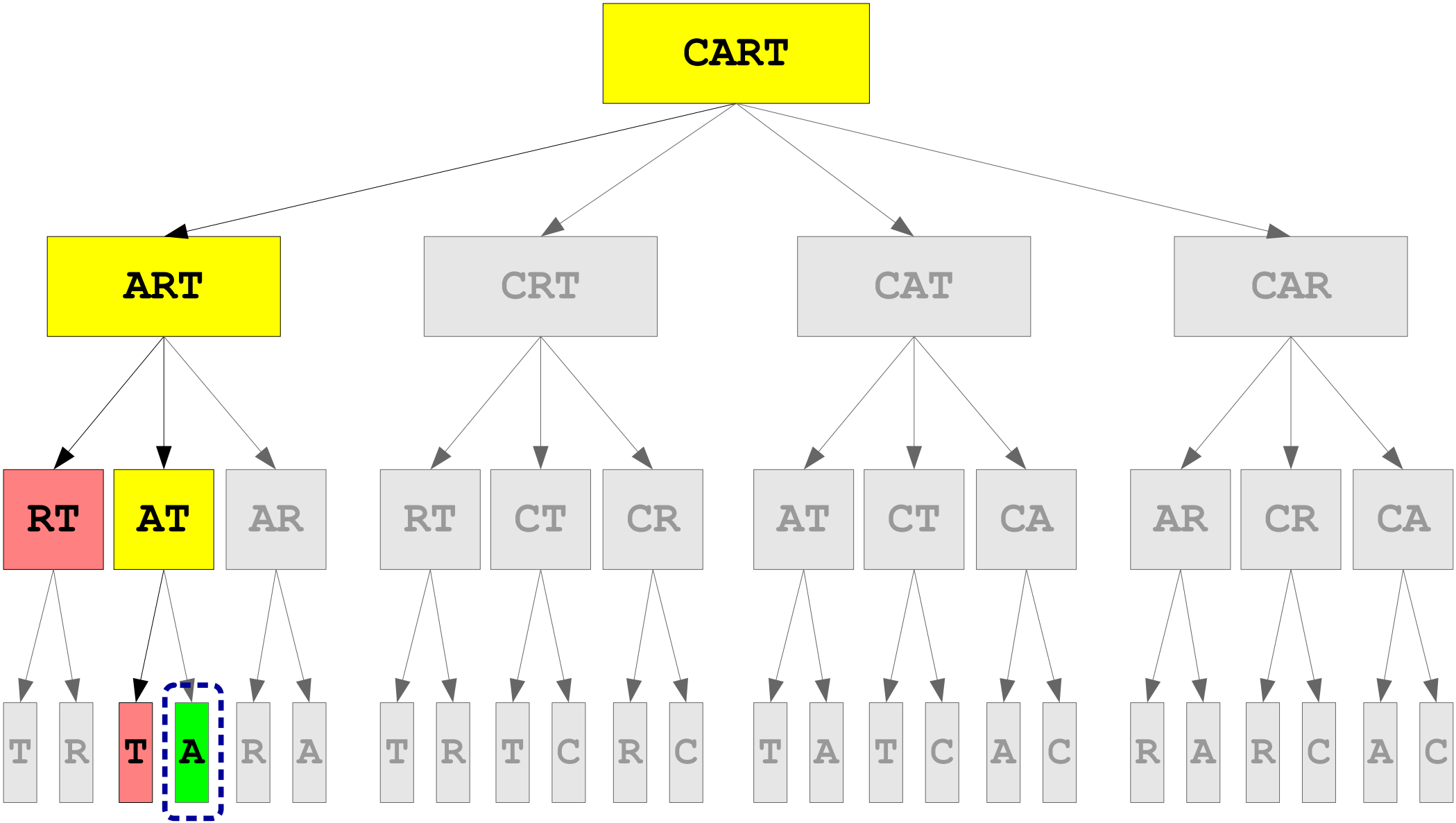
Generating the Answer



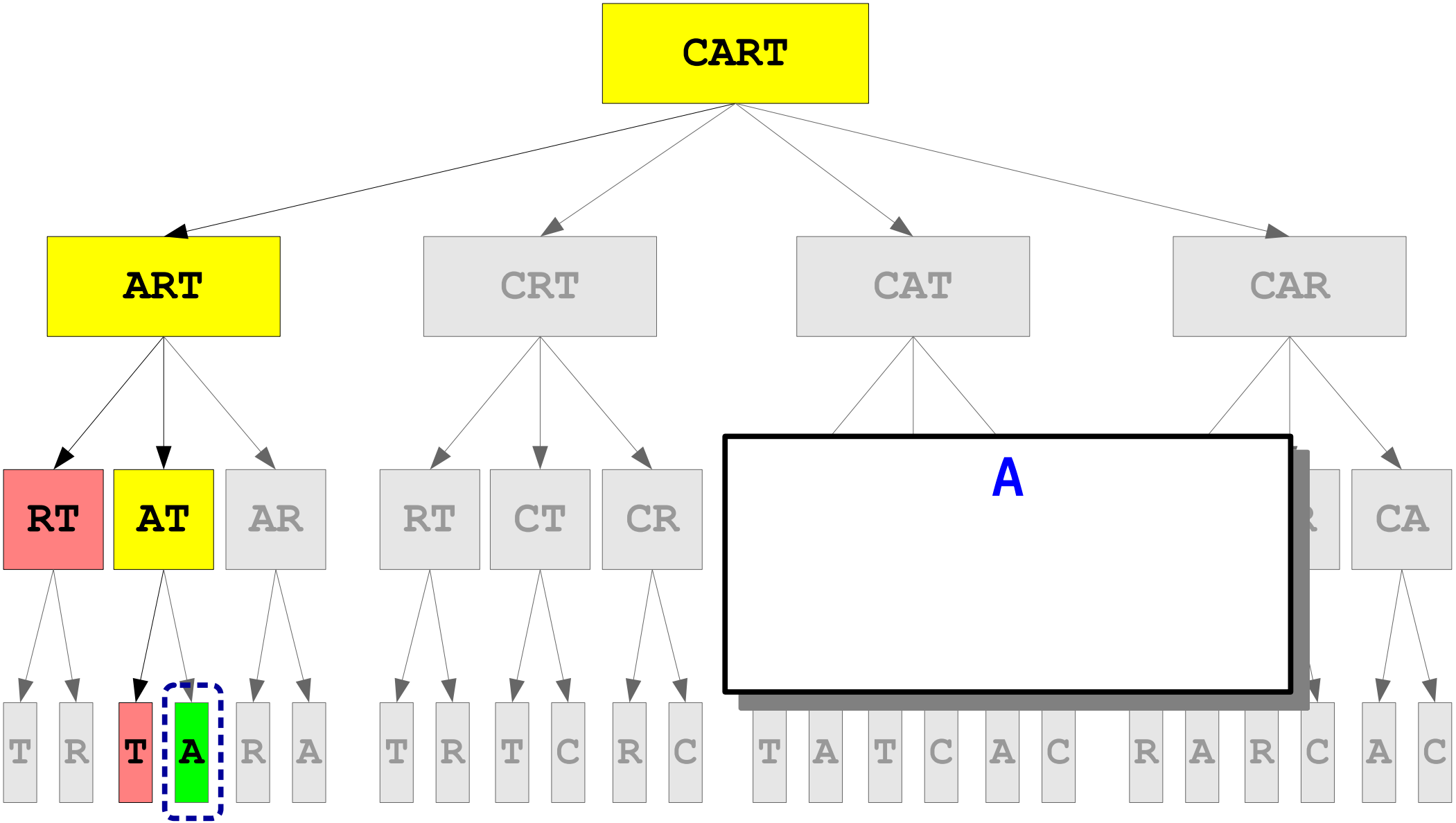
Generating the Answer



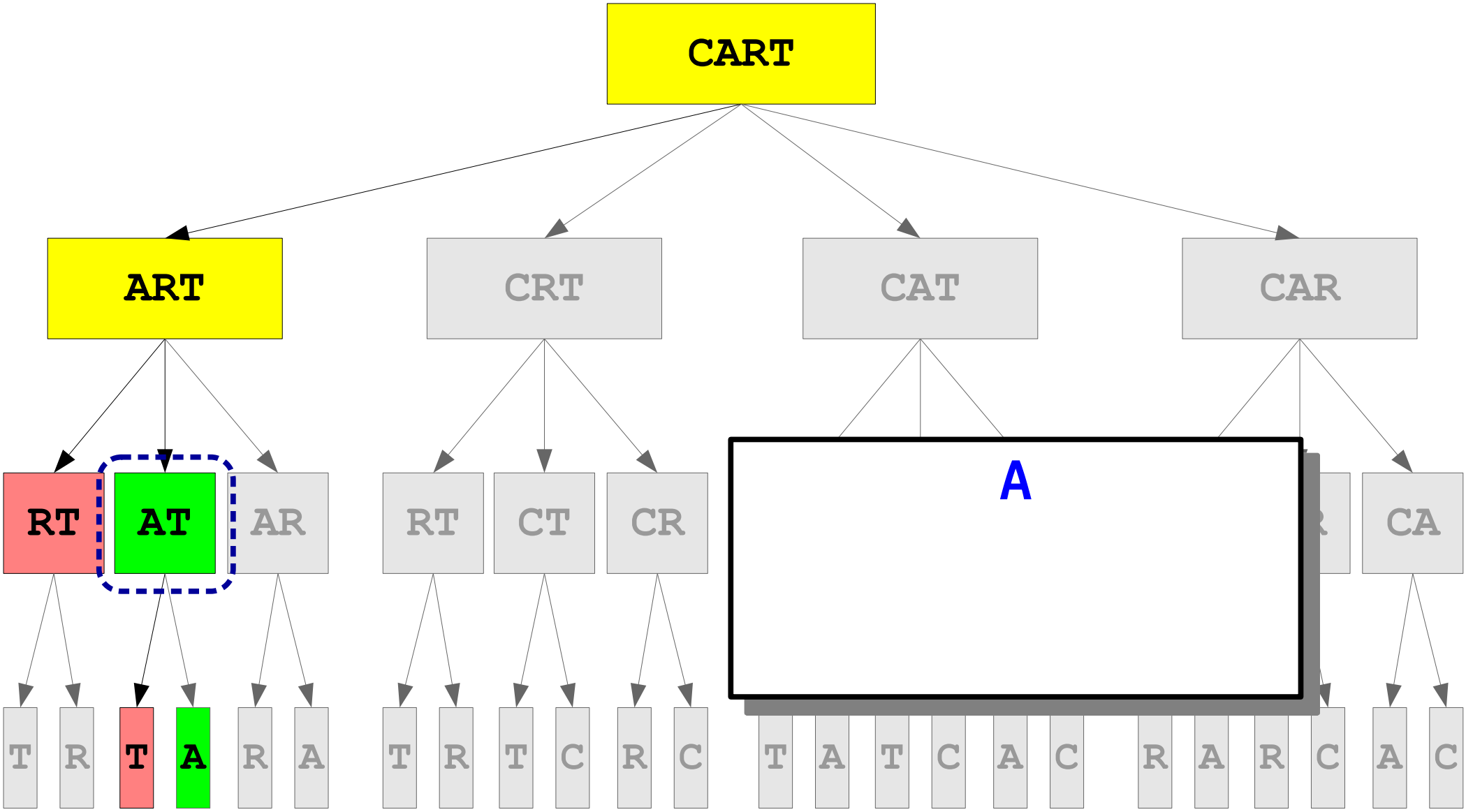
Generating the Answer



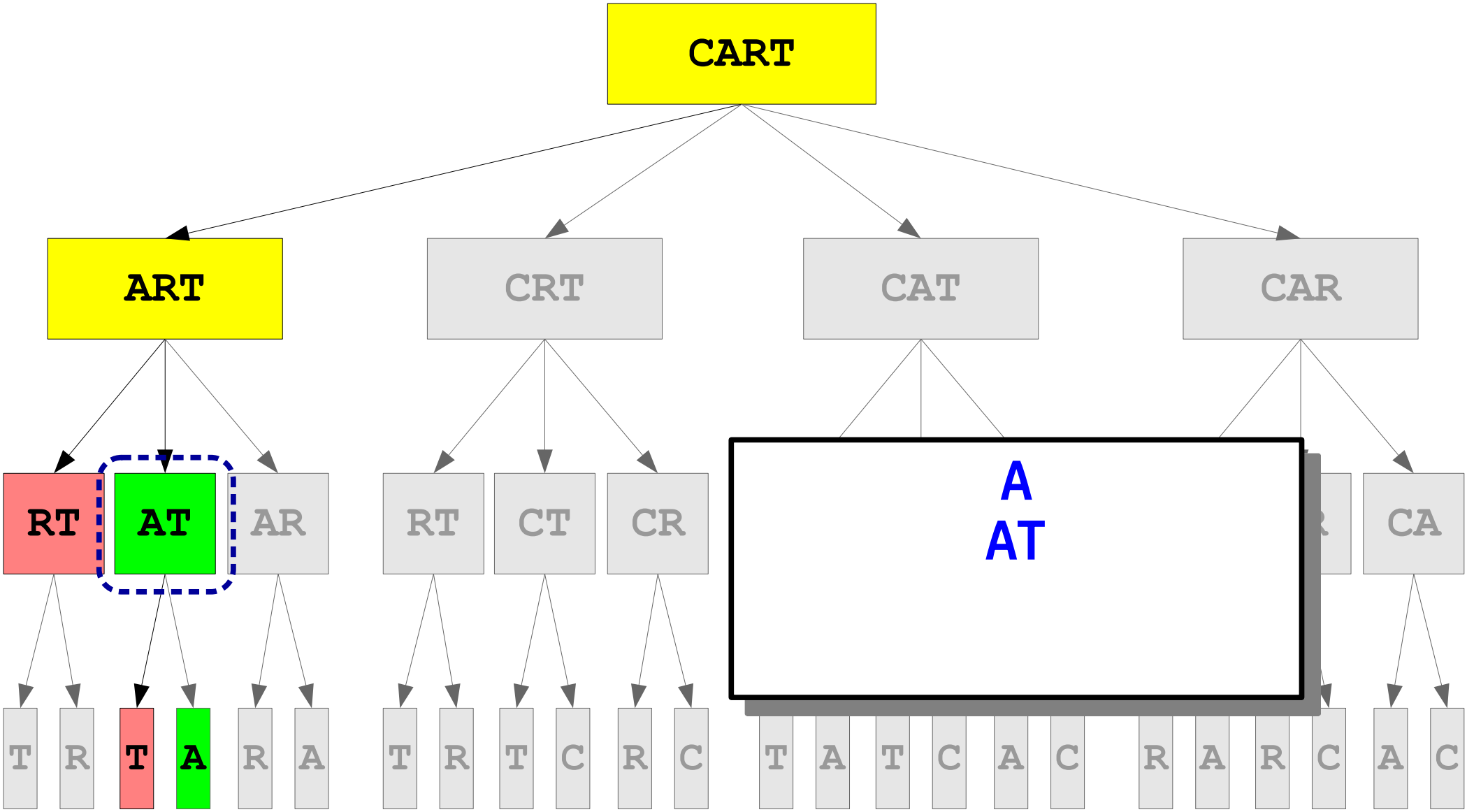
Generating the Answer



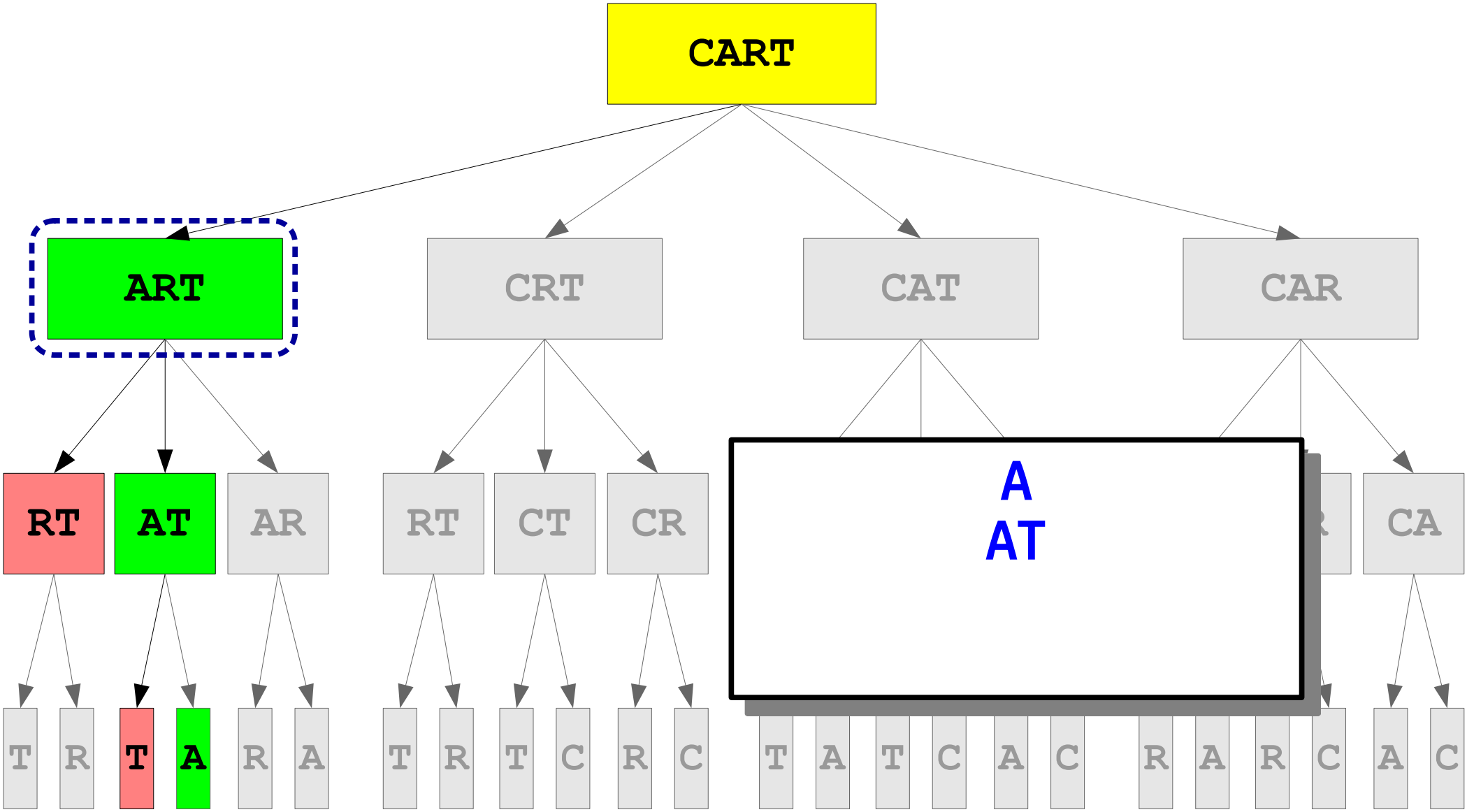
Generating the Answer



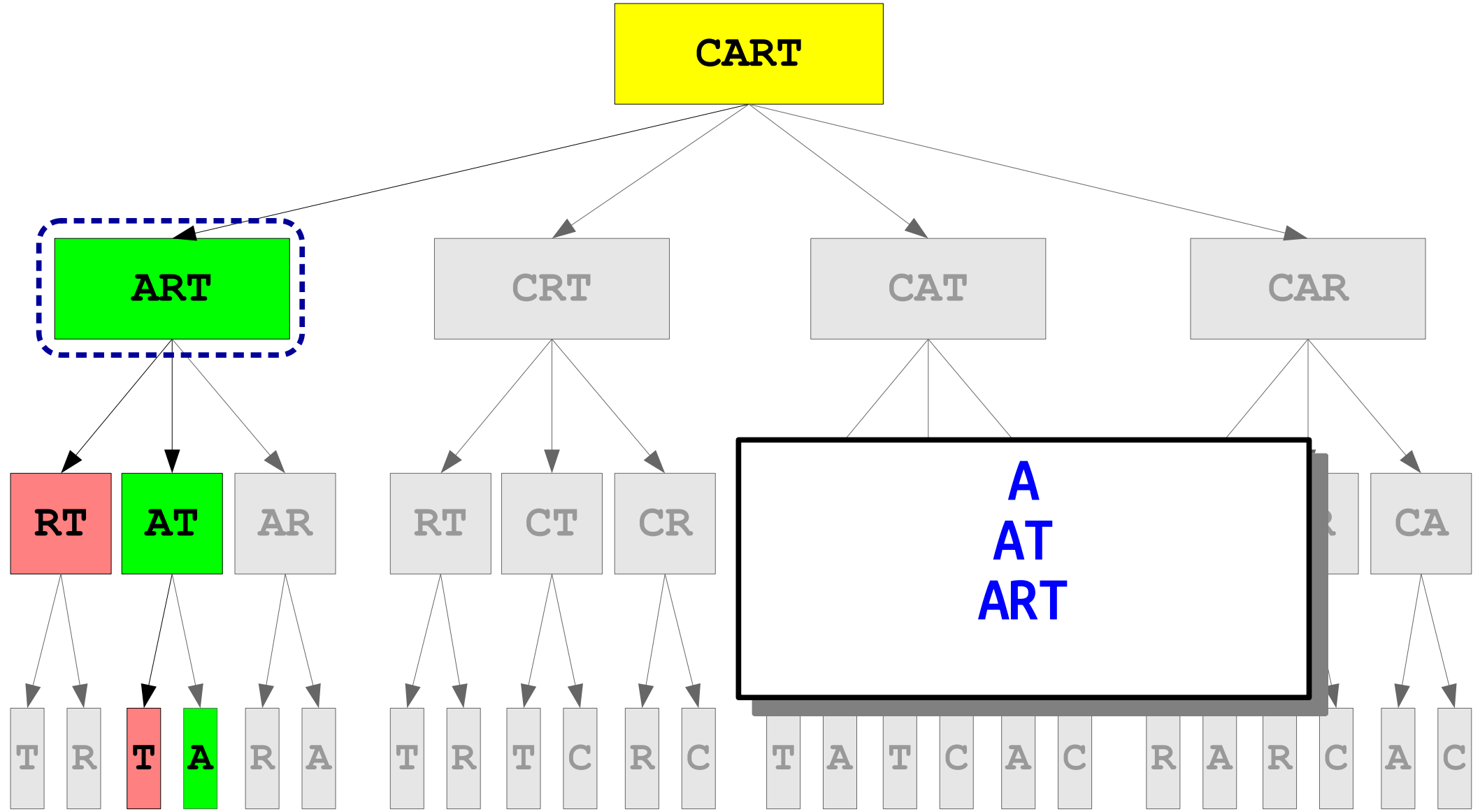
Generating the Answer



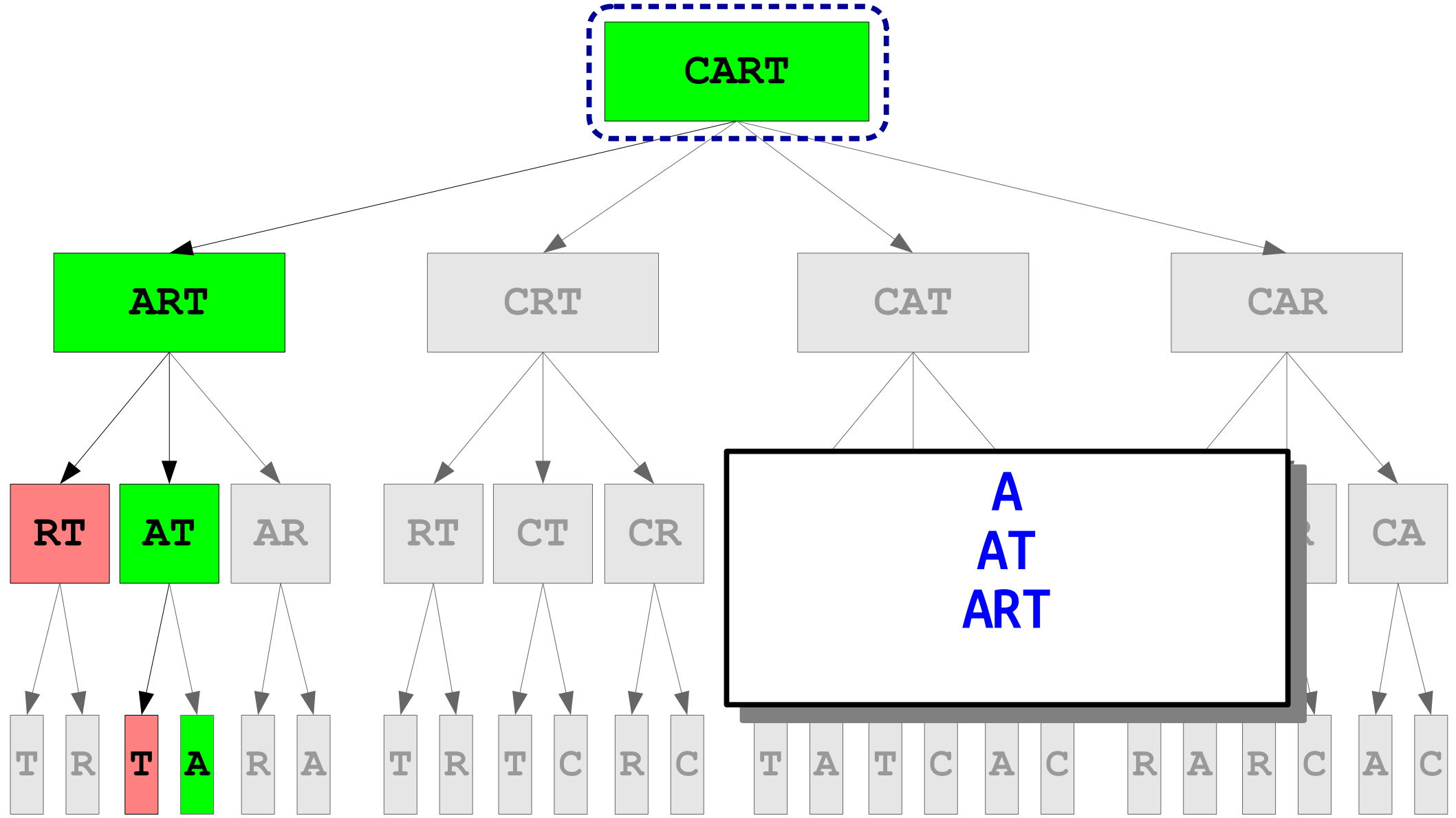
Generating the Answer



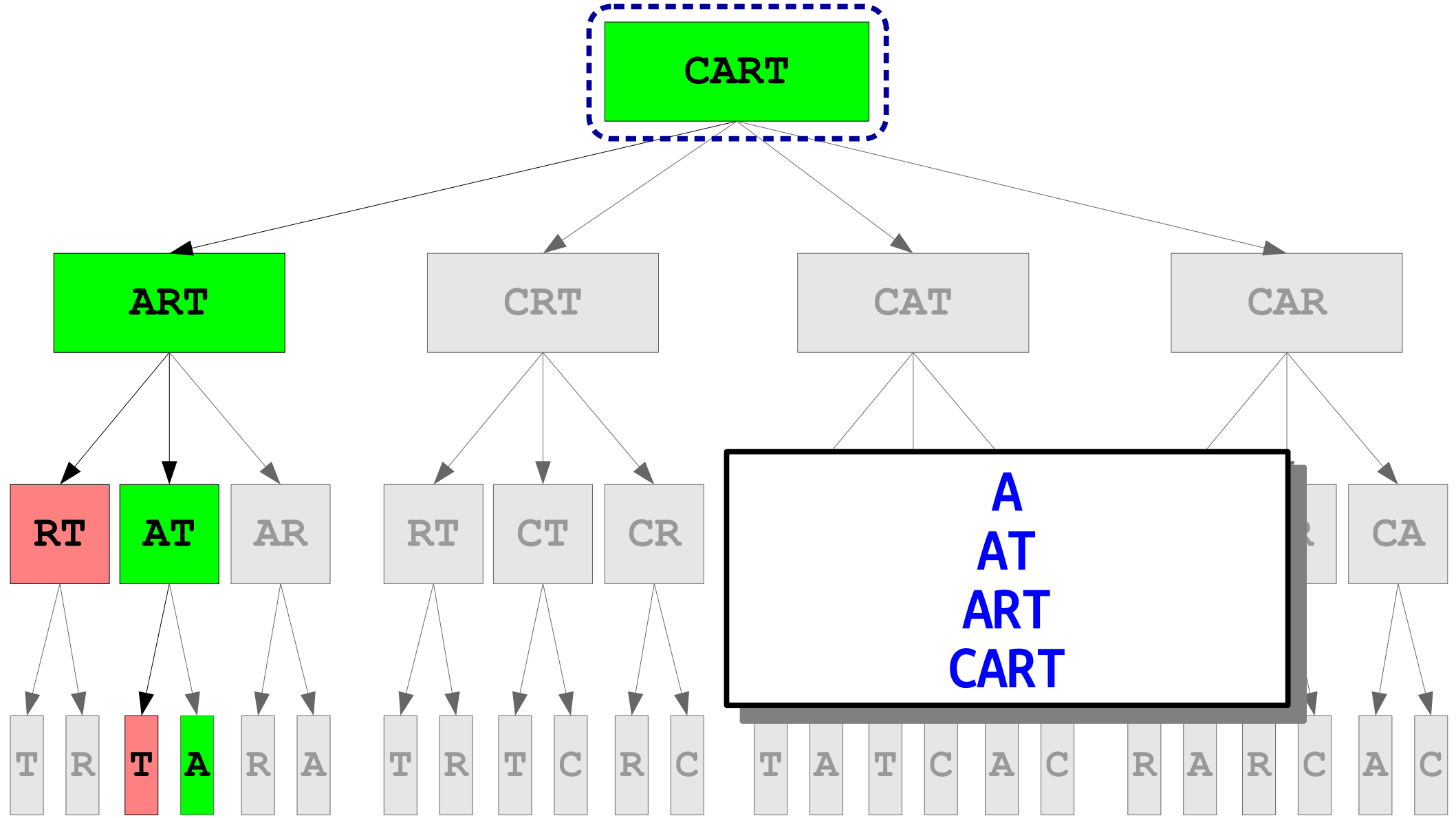
Generating the Answer



Generating the Answer

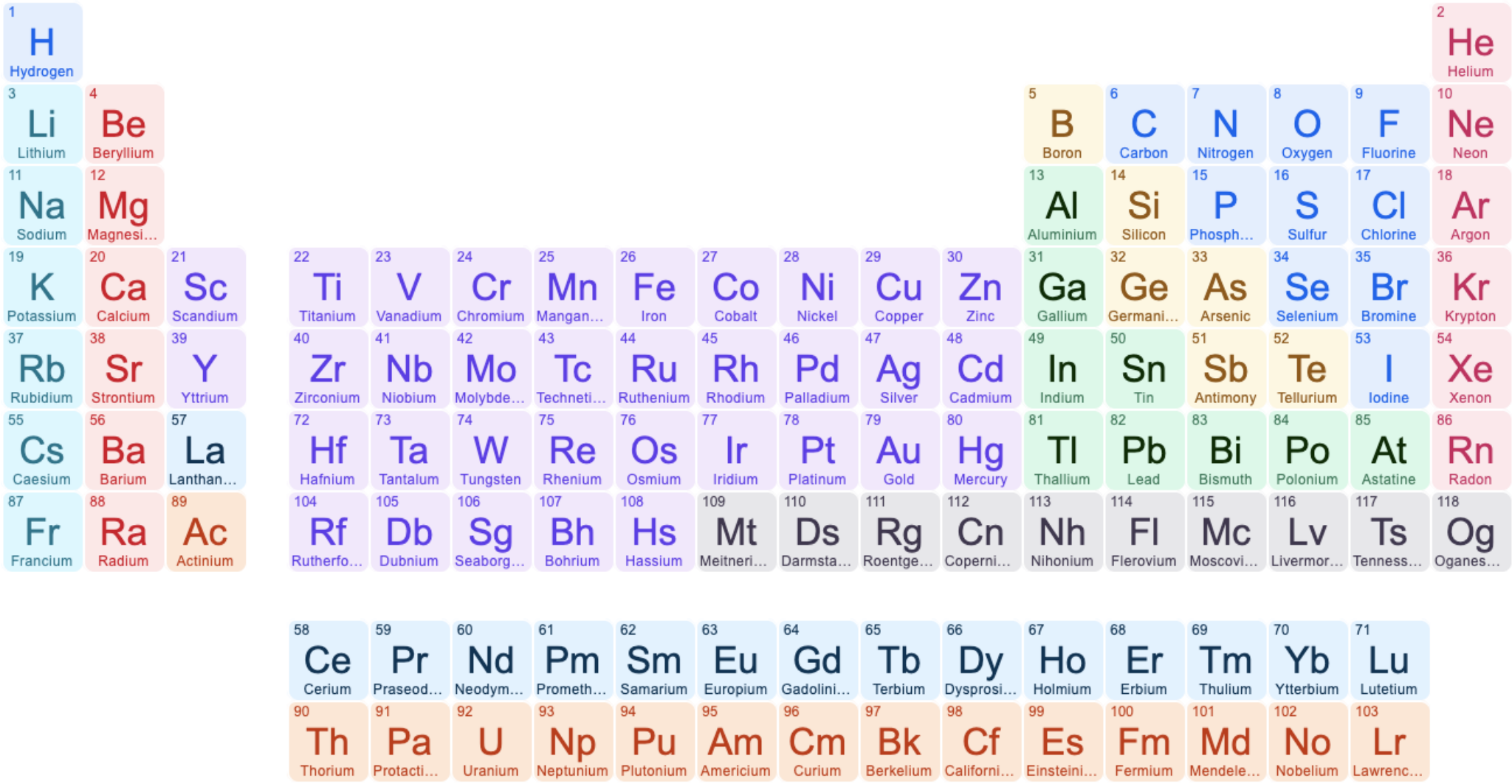


Generating the Answer

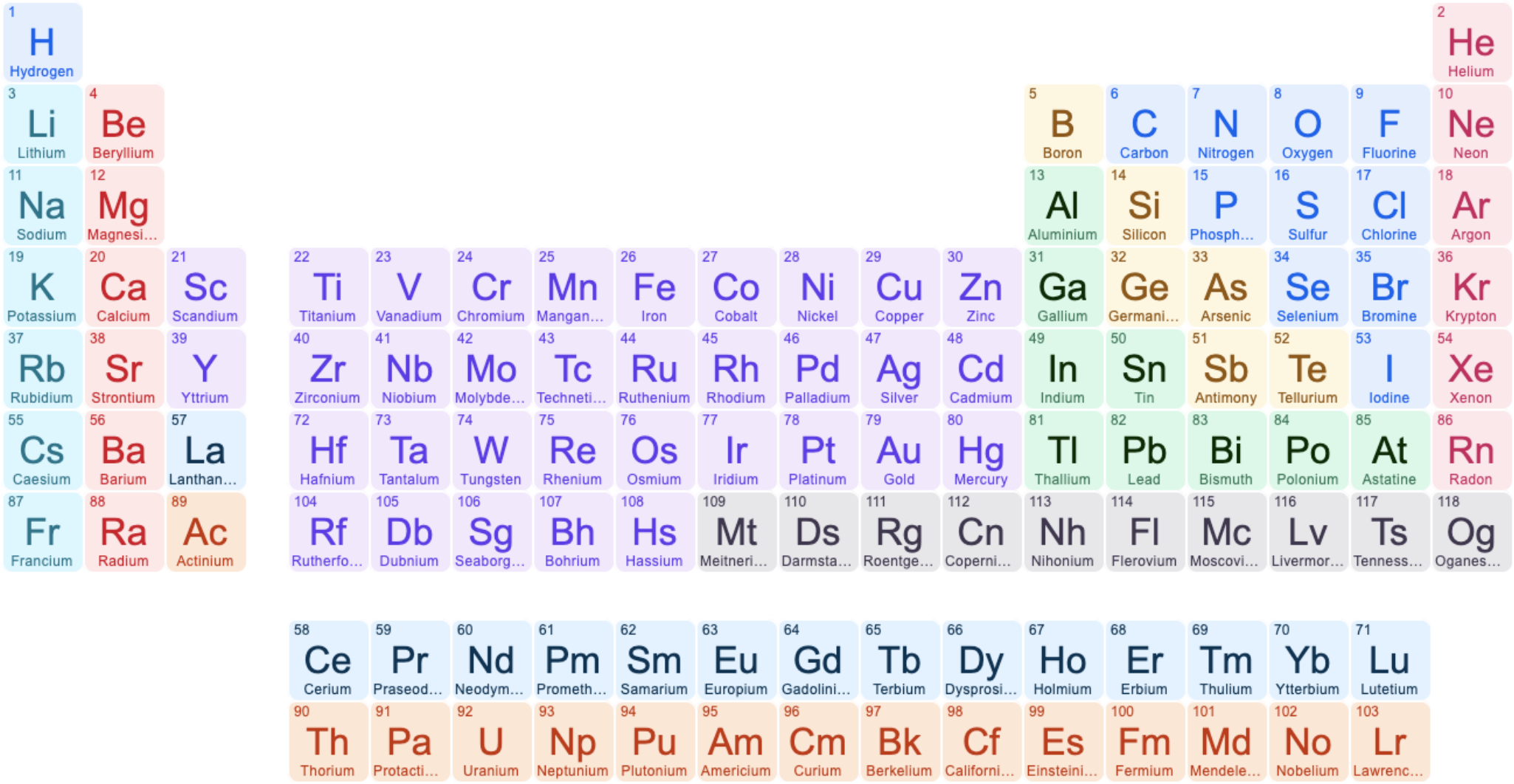


Another Backtracking Example

A Great Tool of Science



Oooh! Letters!



Oooh! Letters!

Can We Do Better?

CHeMoWIZrDy

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

CaNiNe FeLiNe PHYSiCs

UNIVErSITiEs HAlLuCINoGeNiCs

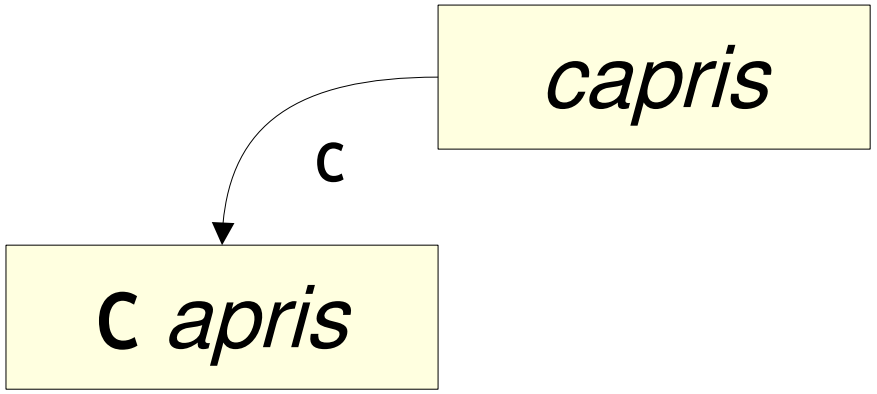
- Given a word, can we spell it using only symbols from the periodic table?
- And, if so, how?

NoTiCe ThAt

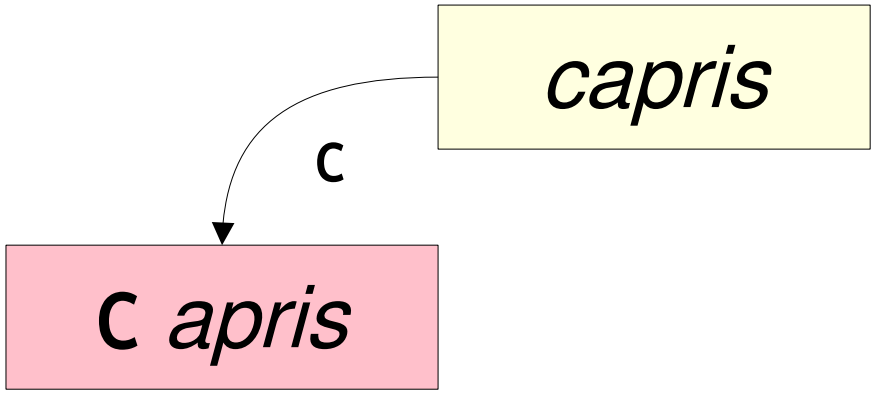
NoTiCe ThAt

capris

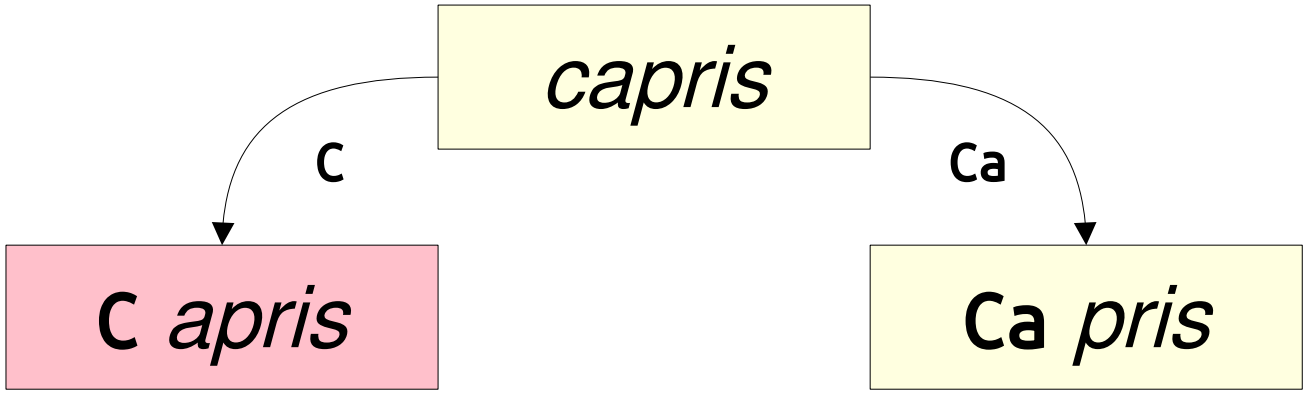
NoTiCe ThAt



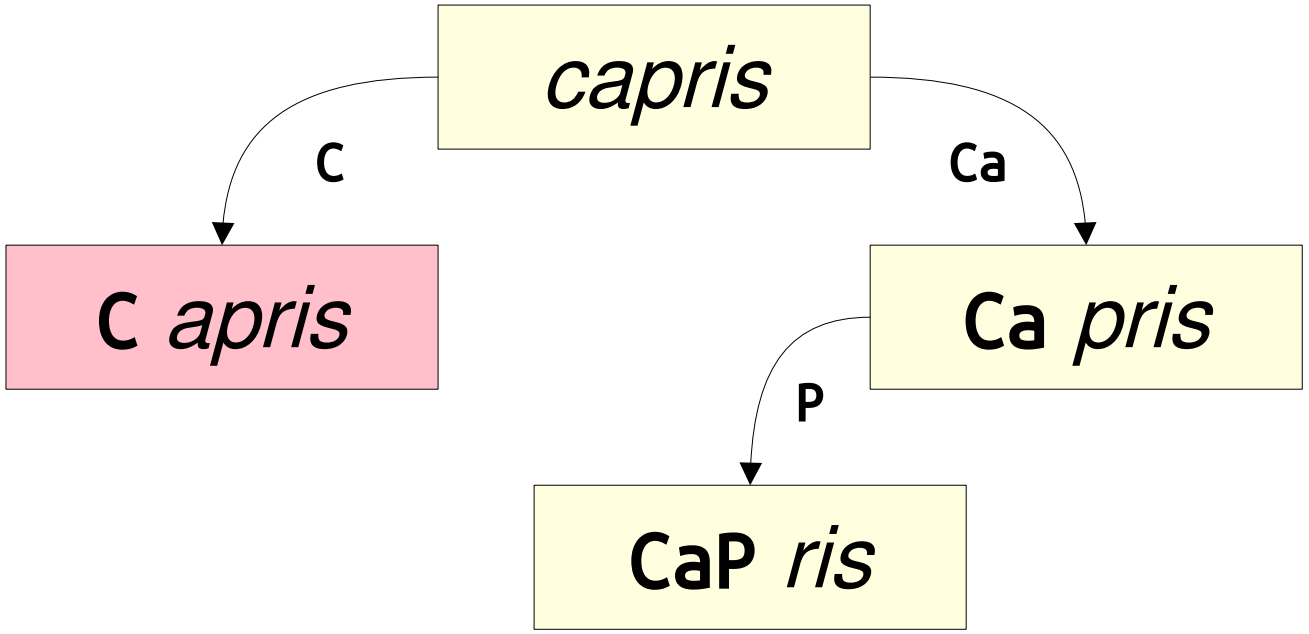
NoTiCe ThAt



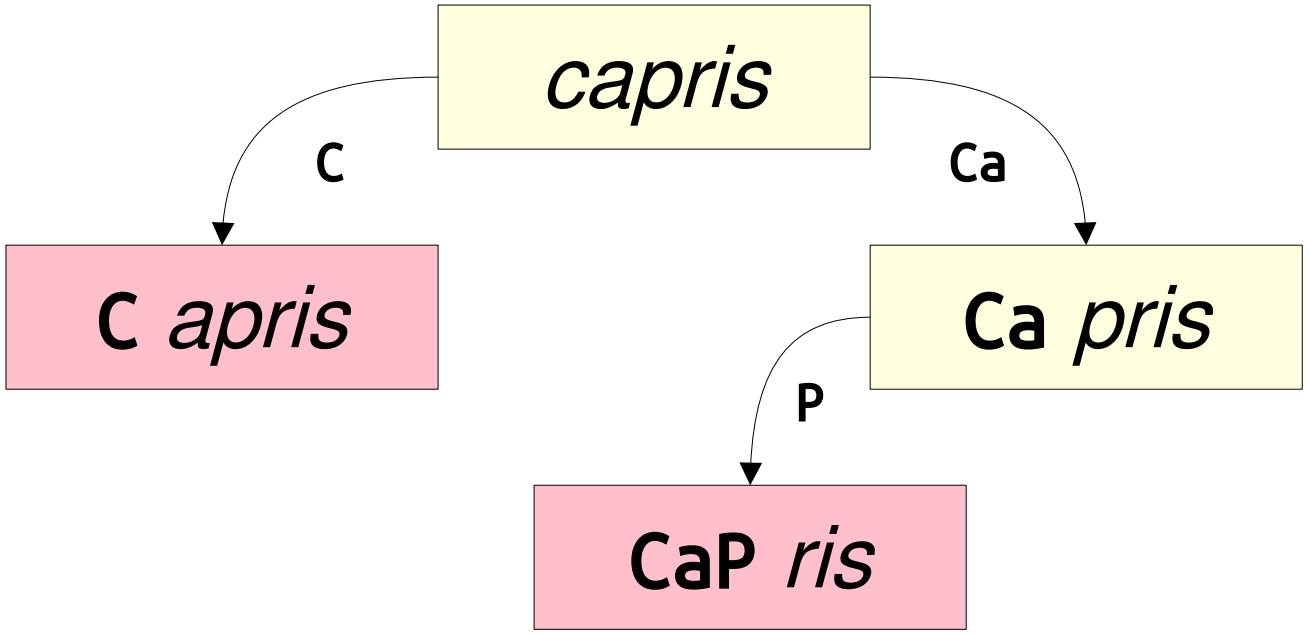
NoTiCe ThAt



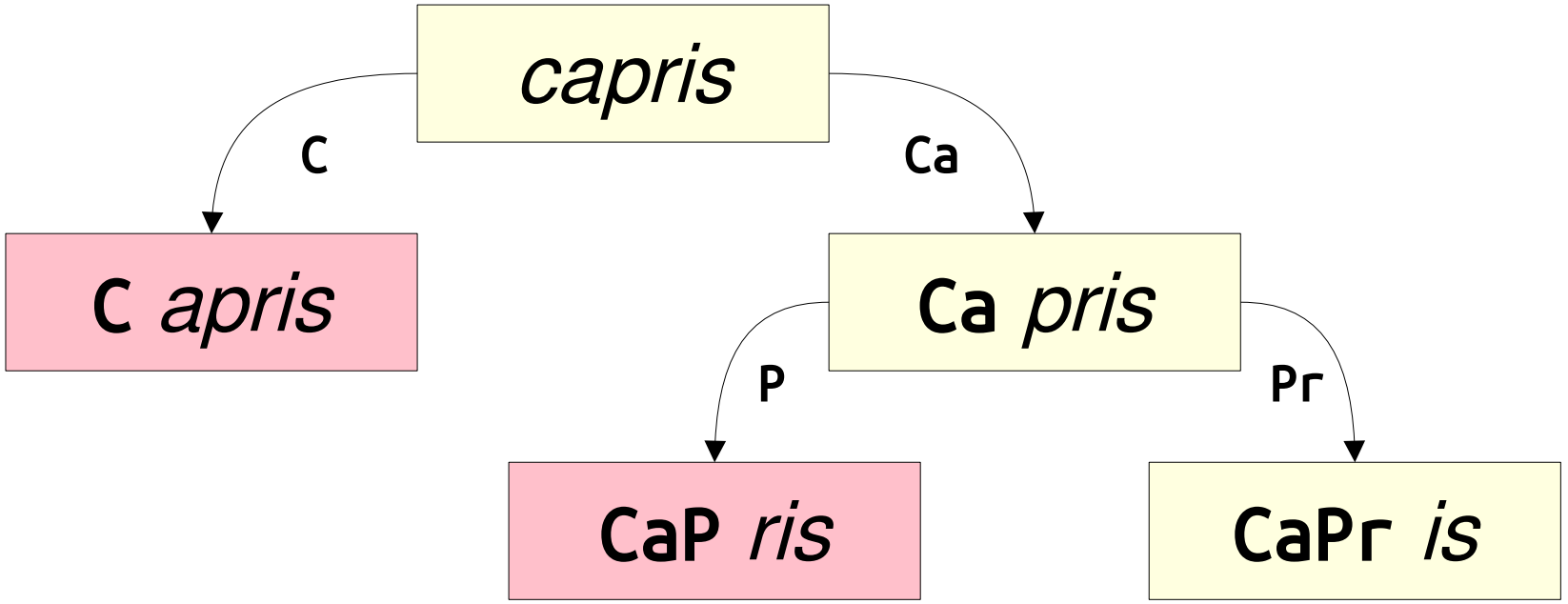
NoTiCe ThAt



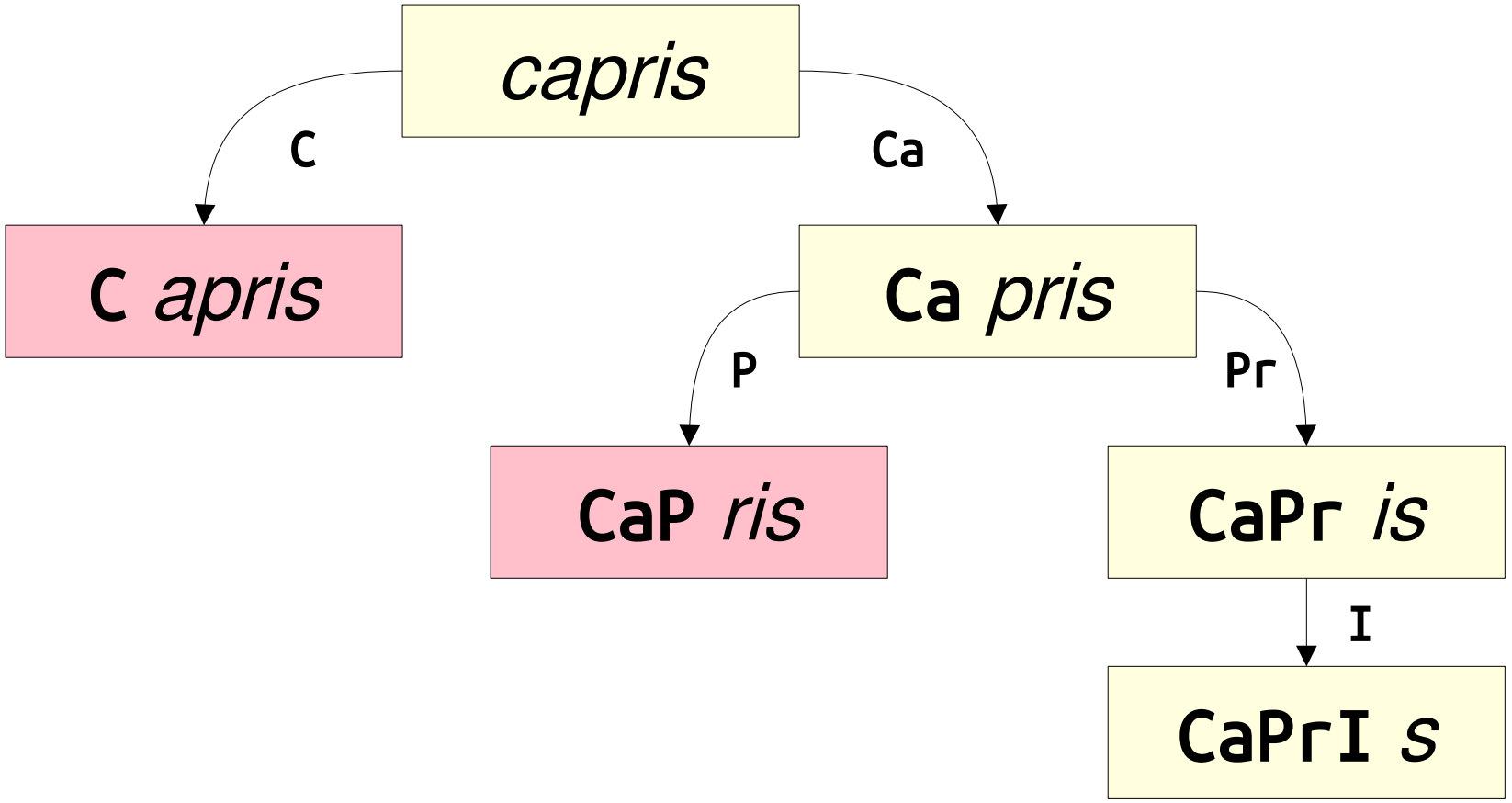
NoTiCe ThAt



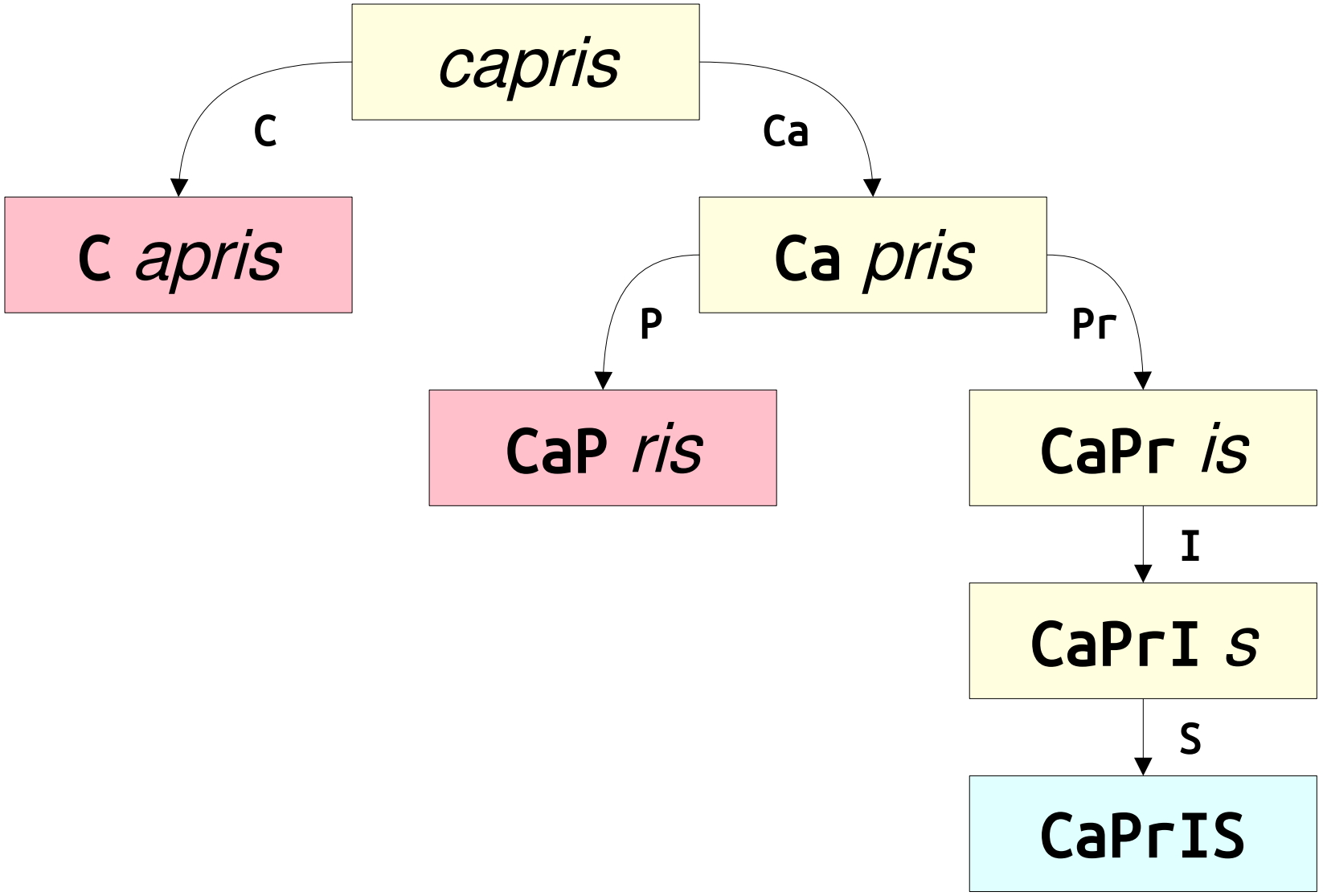
NoTiCe ThAt



NoTiCe ThAt



NoTiCe ThAt


capris
C
Ca
C apris
Ca pris
P
Pr
CaP ris
CaPr is
I
CaPrI s
s
CaPrIS

RhHeCuRhSiON

- **BaSe CaSe:**
  - The empty string can be spelled using just element symbols.
- **RhHeCuRhSiV STeP:**
  - For each element symbol:
    - If the string starts with that symbol, 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.

Closing Thoughts on Recursion

You now know how to use recursion to ***view problems from a different perspective*** that can lead to ***short and elegant solutions***.

You've seen how to use recursion to ***enumerate all objects of some type***, which you can use to find the ***optimal solution to a problem***.

You've seen how to use recursive backtracking to ***determine whether something is possible*** and, if so to ***find some way to do it***.

Congratulations on making it this far!

Your Action Items

- ***Finish Chapter 9.***
  - It’s all about backtracking, and there are some great examples in there!
- ***Finish Assignment 3.***
  - As always, get in touch with us if we can help out!

Next Time

- ***Algorithmic Analysis***
  - How do we formally analyze the complexity of a piece of code?
- ***Big-O Notation***
  - Quantifying efficiency!