

# 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

S T A R T L I N G

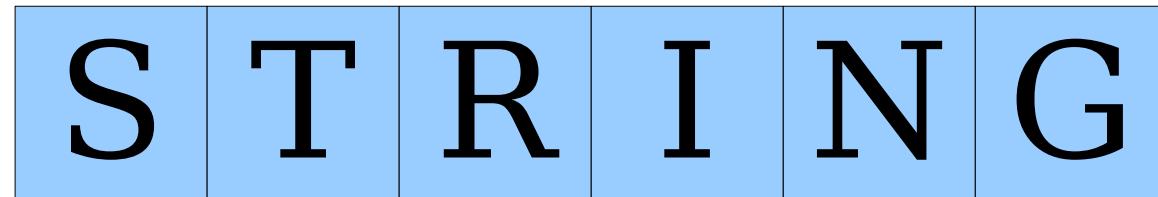
# One Solution

S T A R T I N G

# One Solution

S T A R I N G

# One Solution



S T R I N G

# One Solution

S	T	I	N	G
---	---	---	---	---

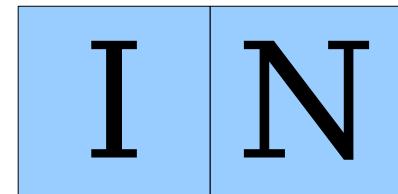
# One Solution

S	I	N	G
---	---	---	---

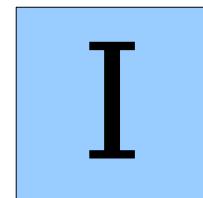
# One Solution

S	I	N
---	---	---

# One Solution



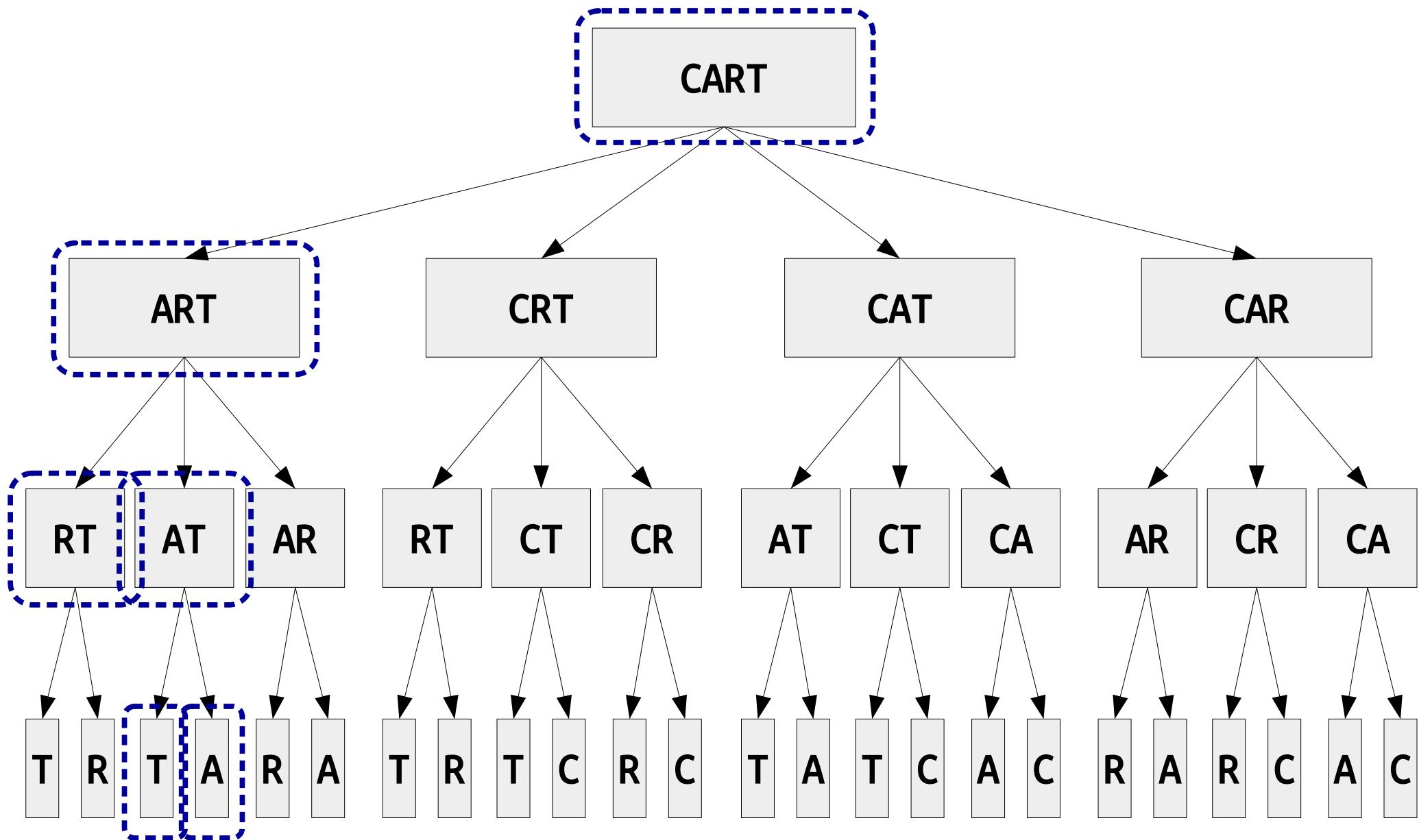
# One Solution



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;
}
```

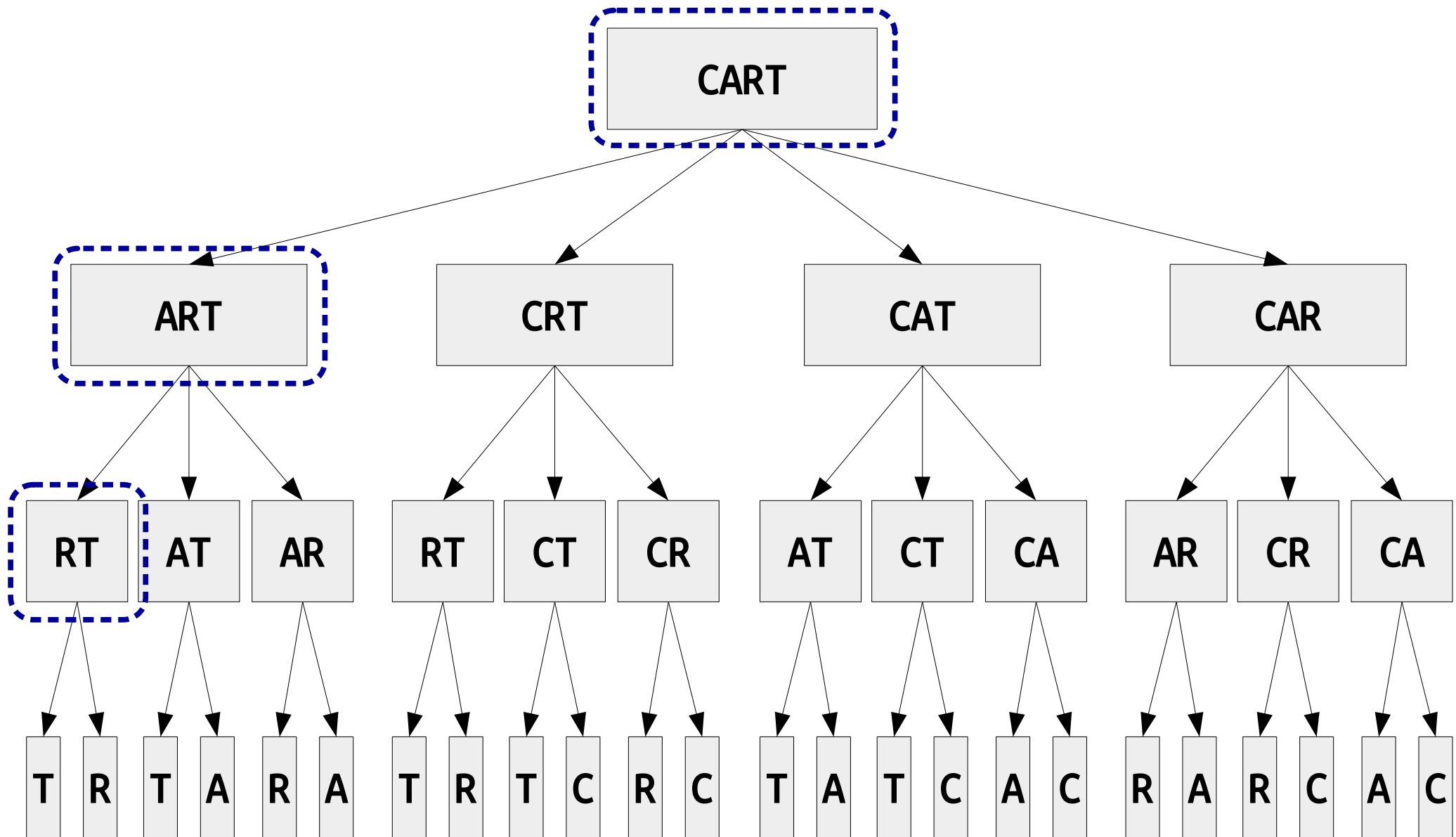
```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    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;  
}
```

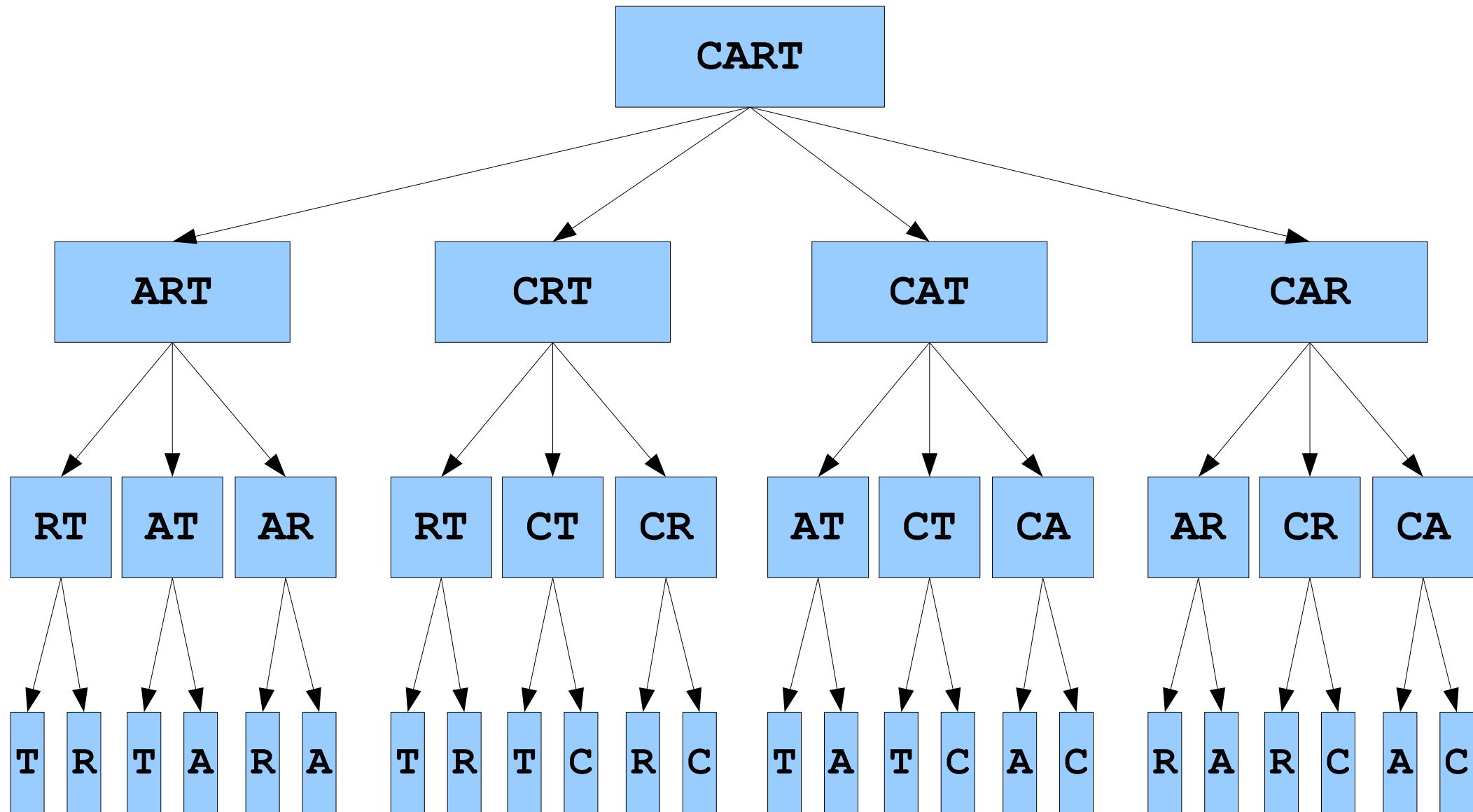
Note that if the recursive call succeeds, then we return success. If it doesn't succeed, that doesn't mean we've failed - it just means we need to try out the next option.

How do we know we're correct?

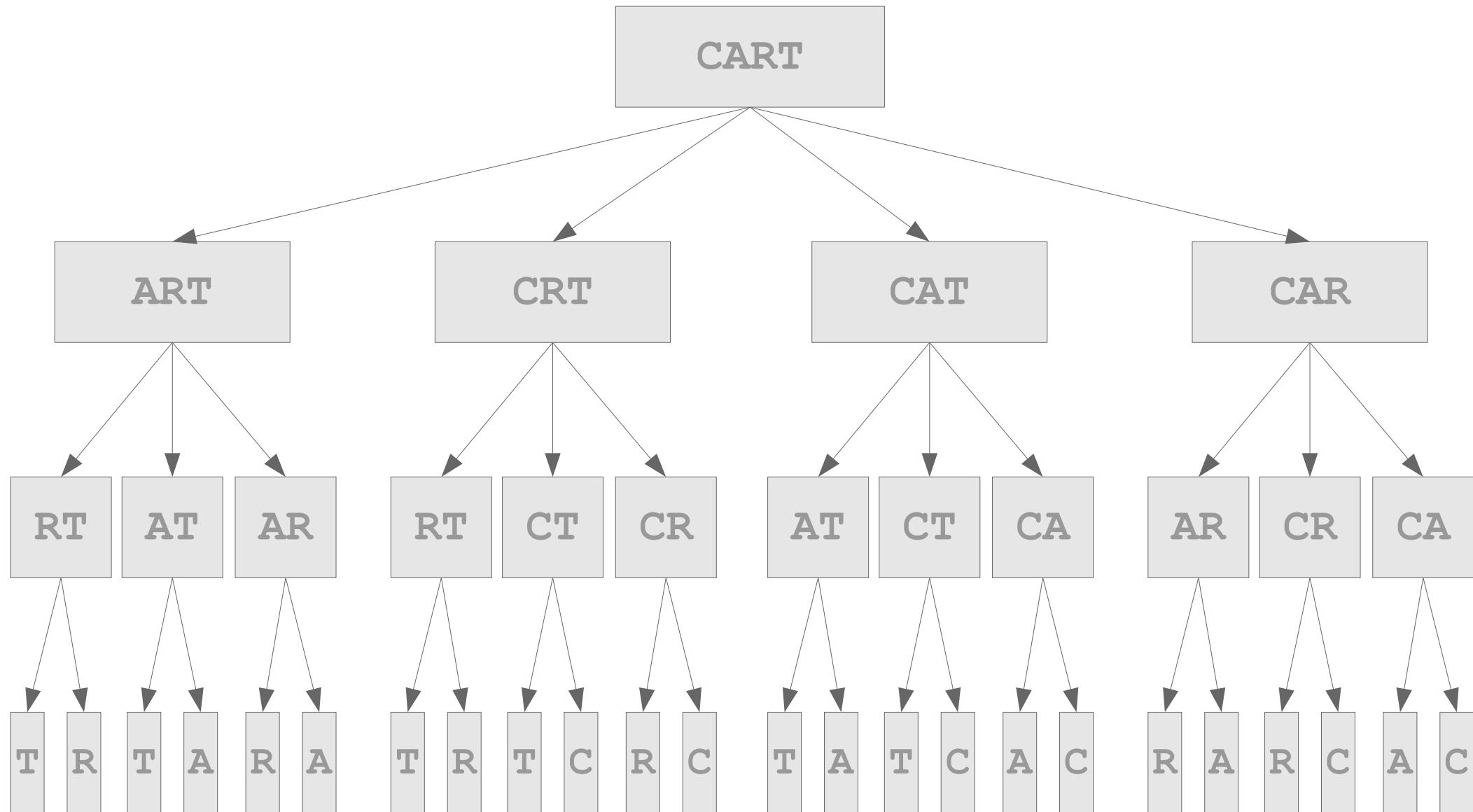
# Optional<T>

- 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.

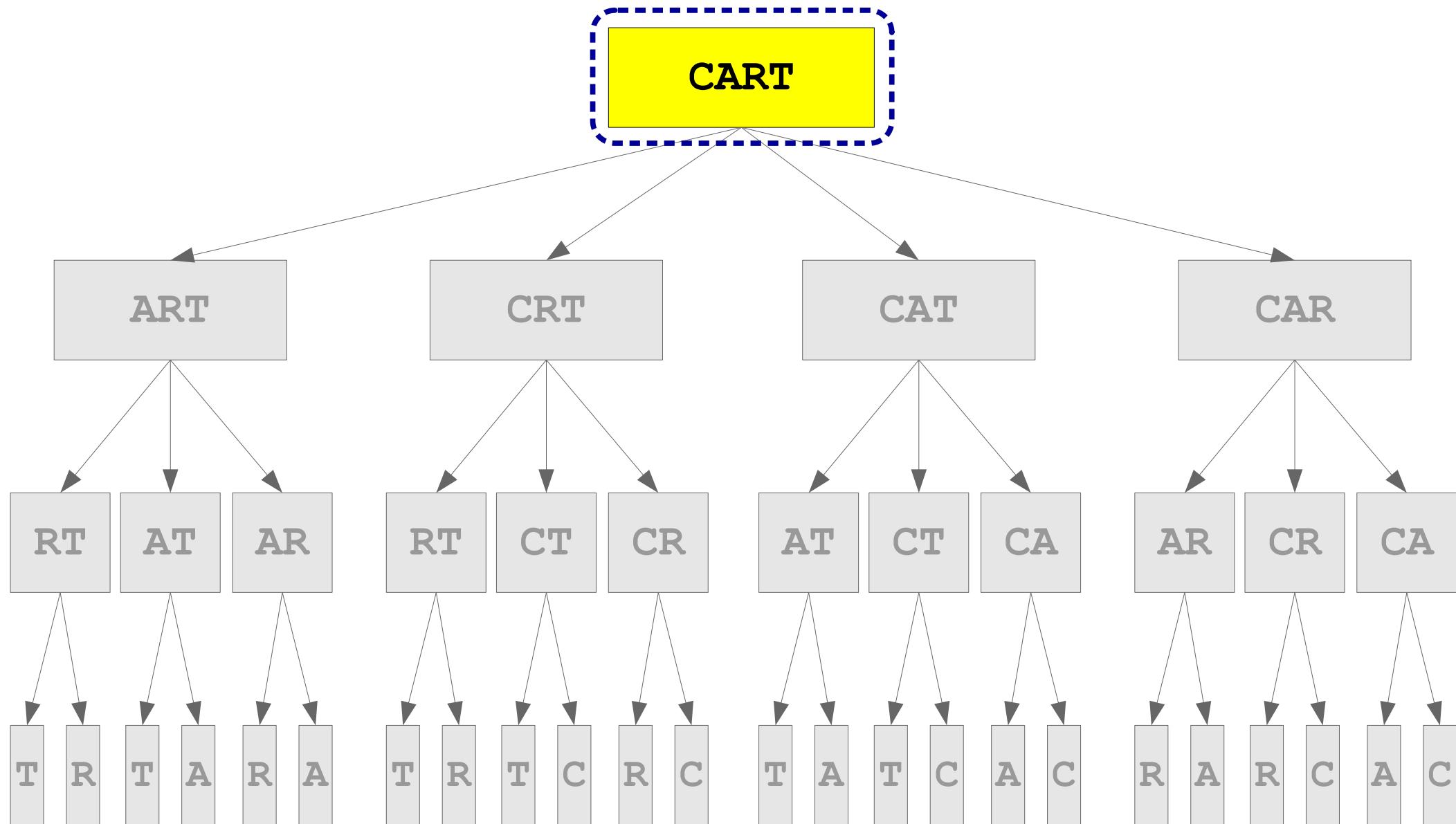
# Generating the Answer



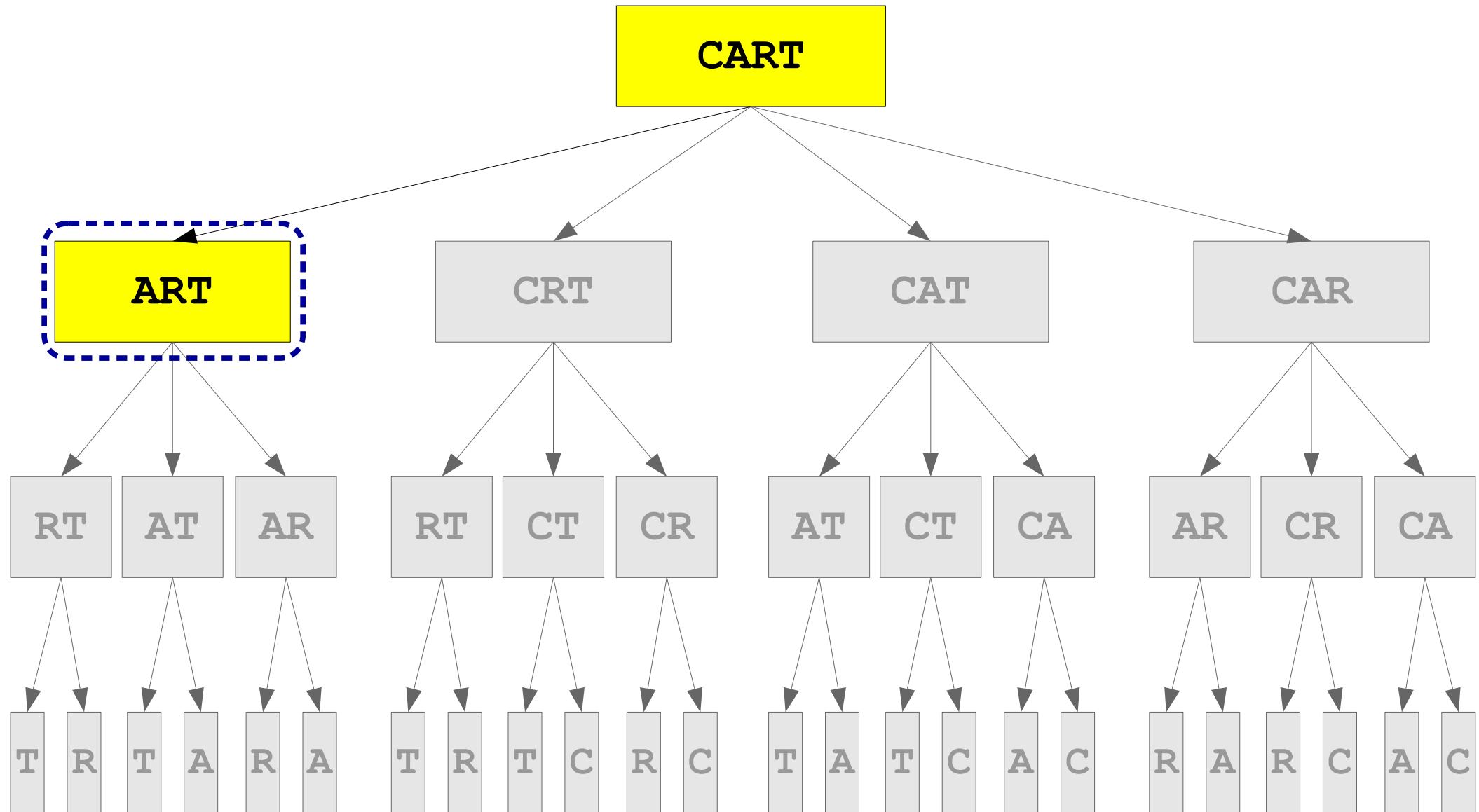
# Generating the Answer



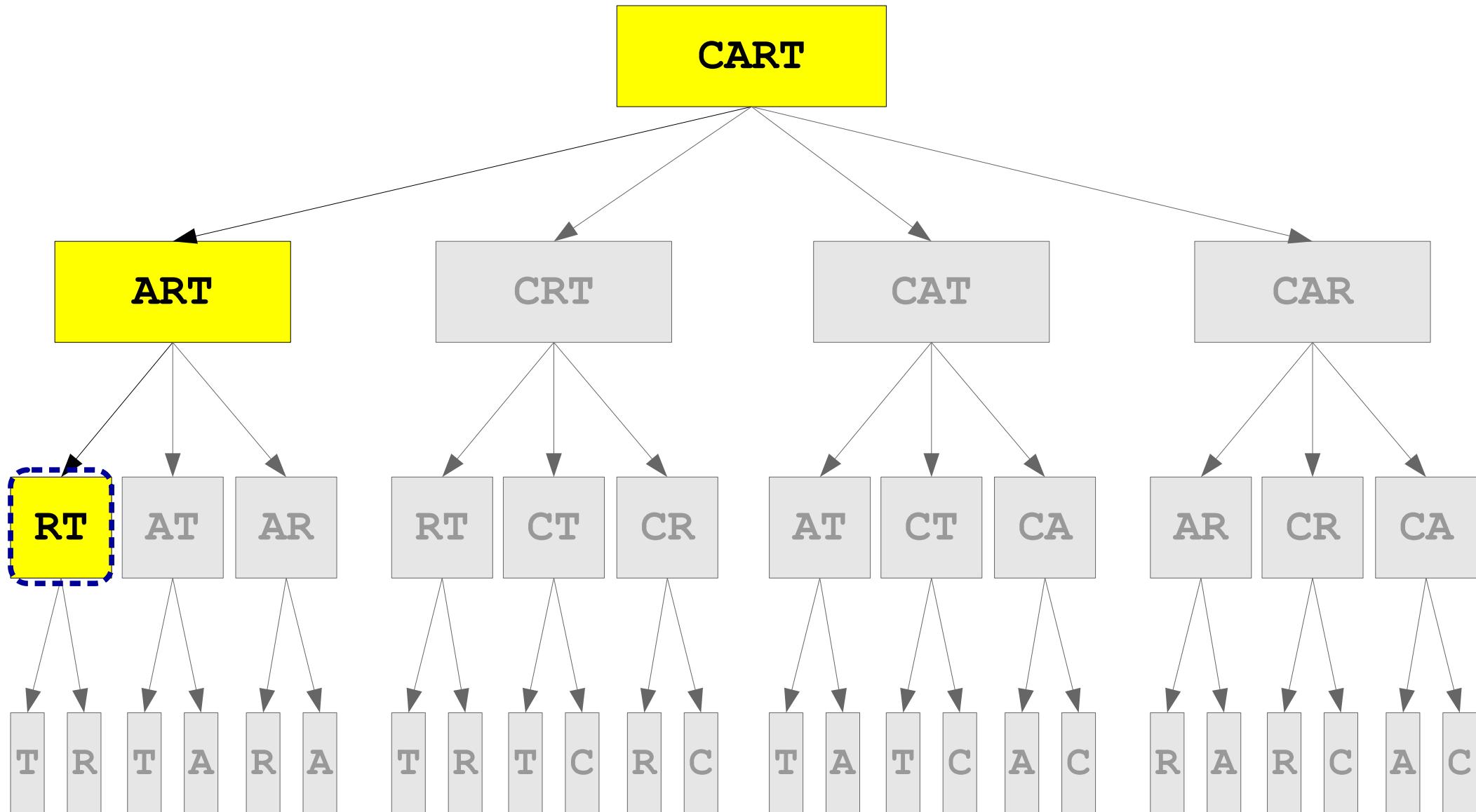
# Generating the Answer



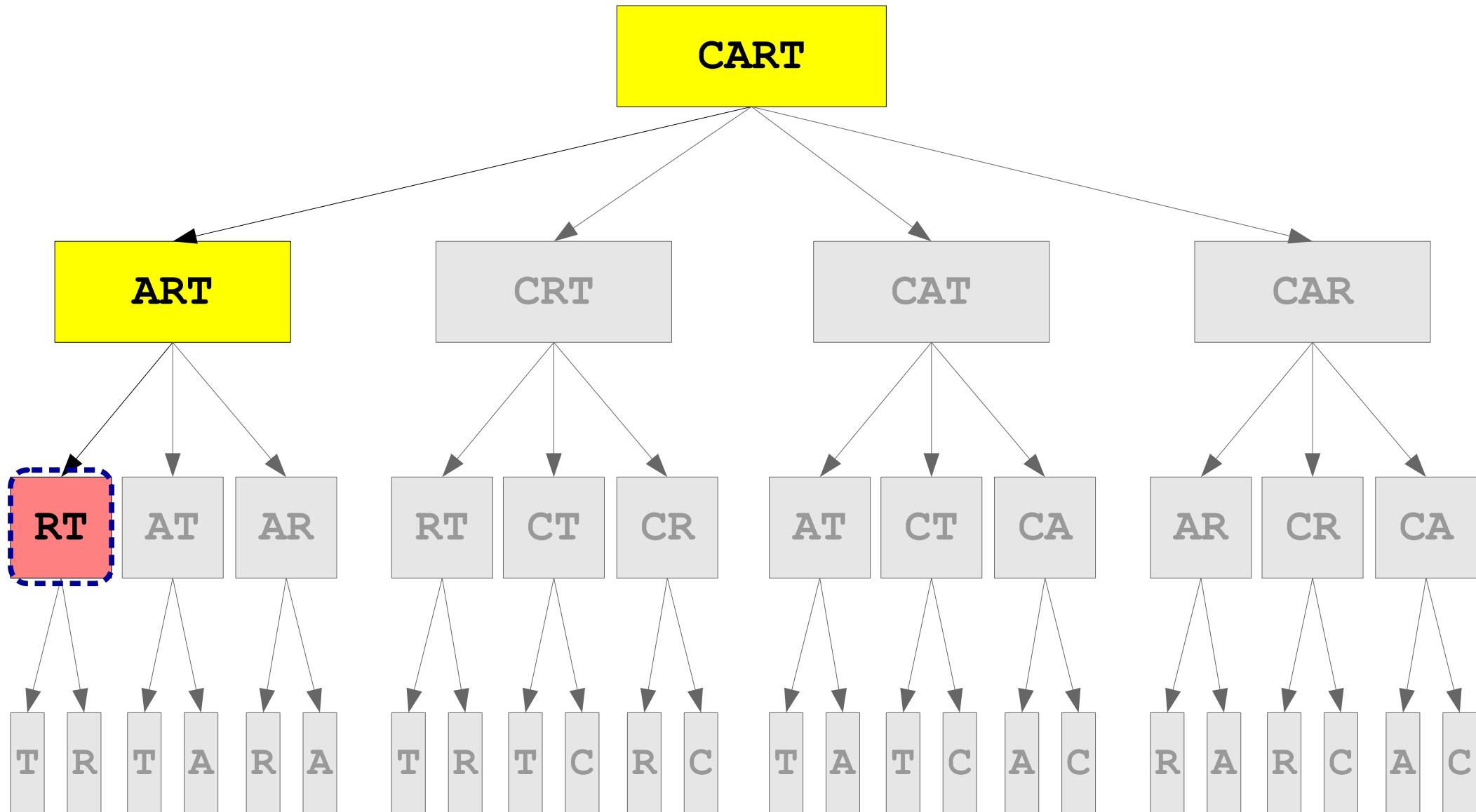
# Generating the Answer



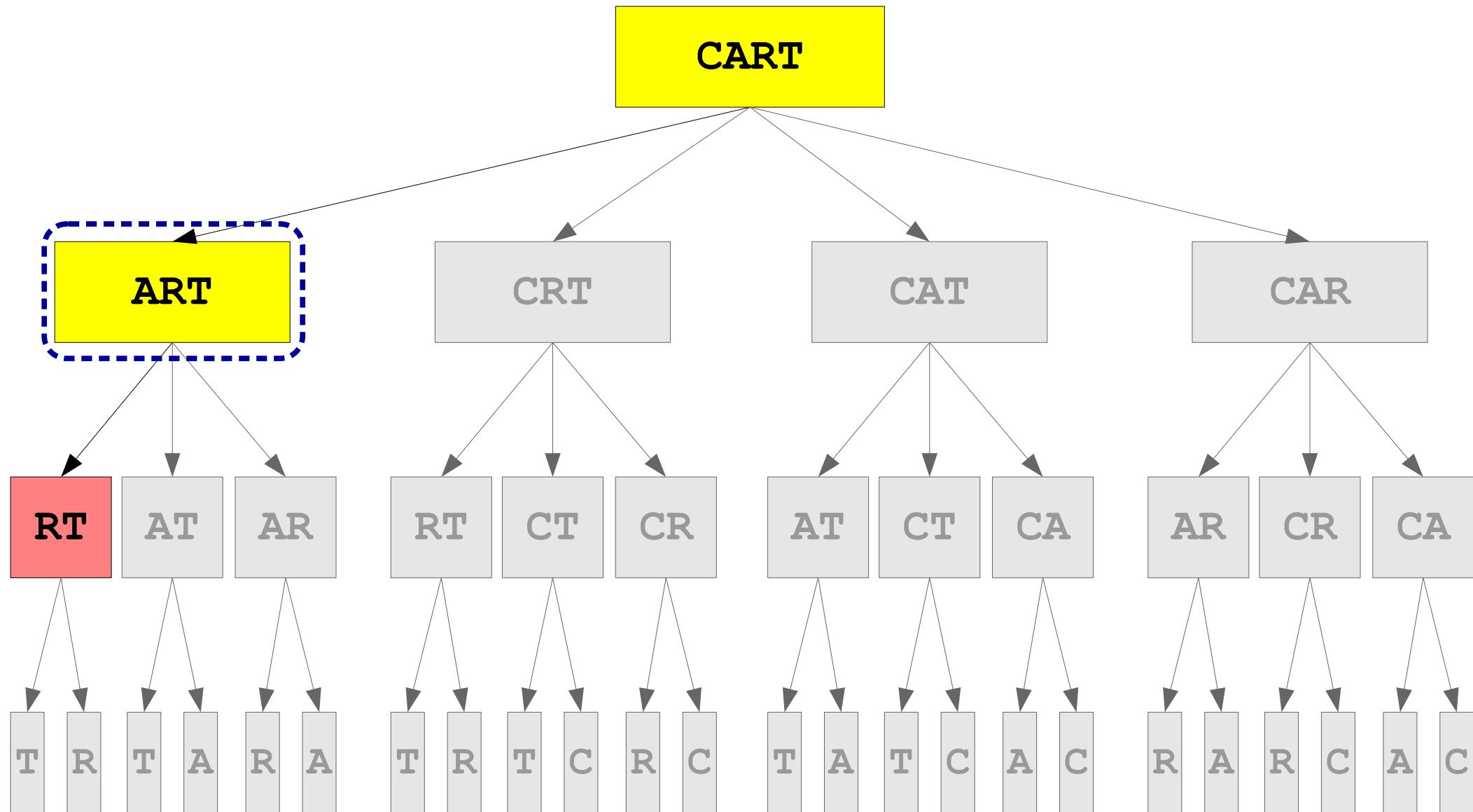
# Generating the Answer



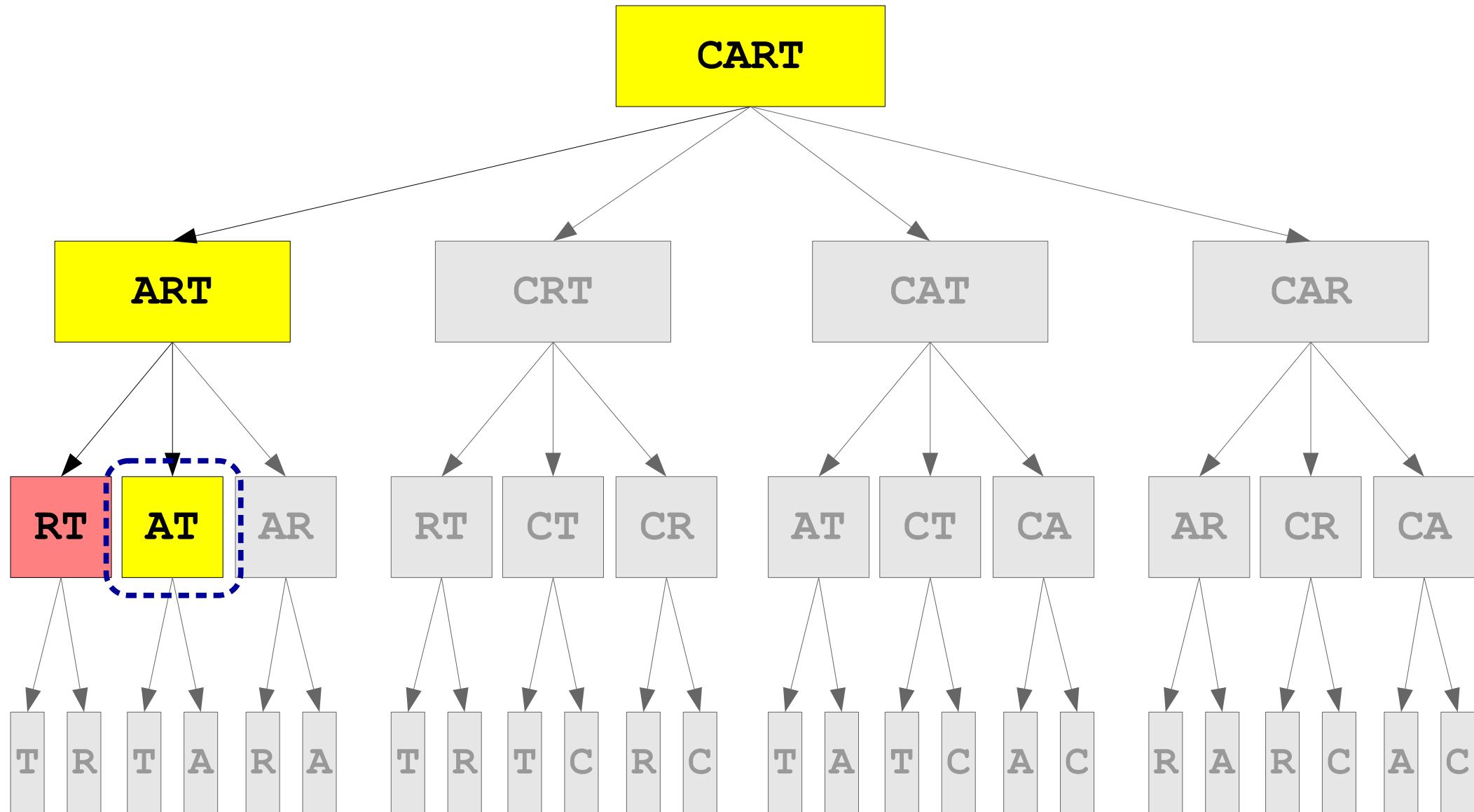
# Generating the Answer



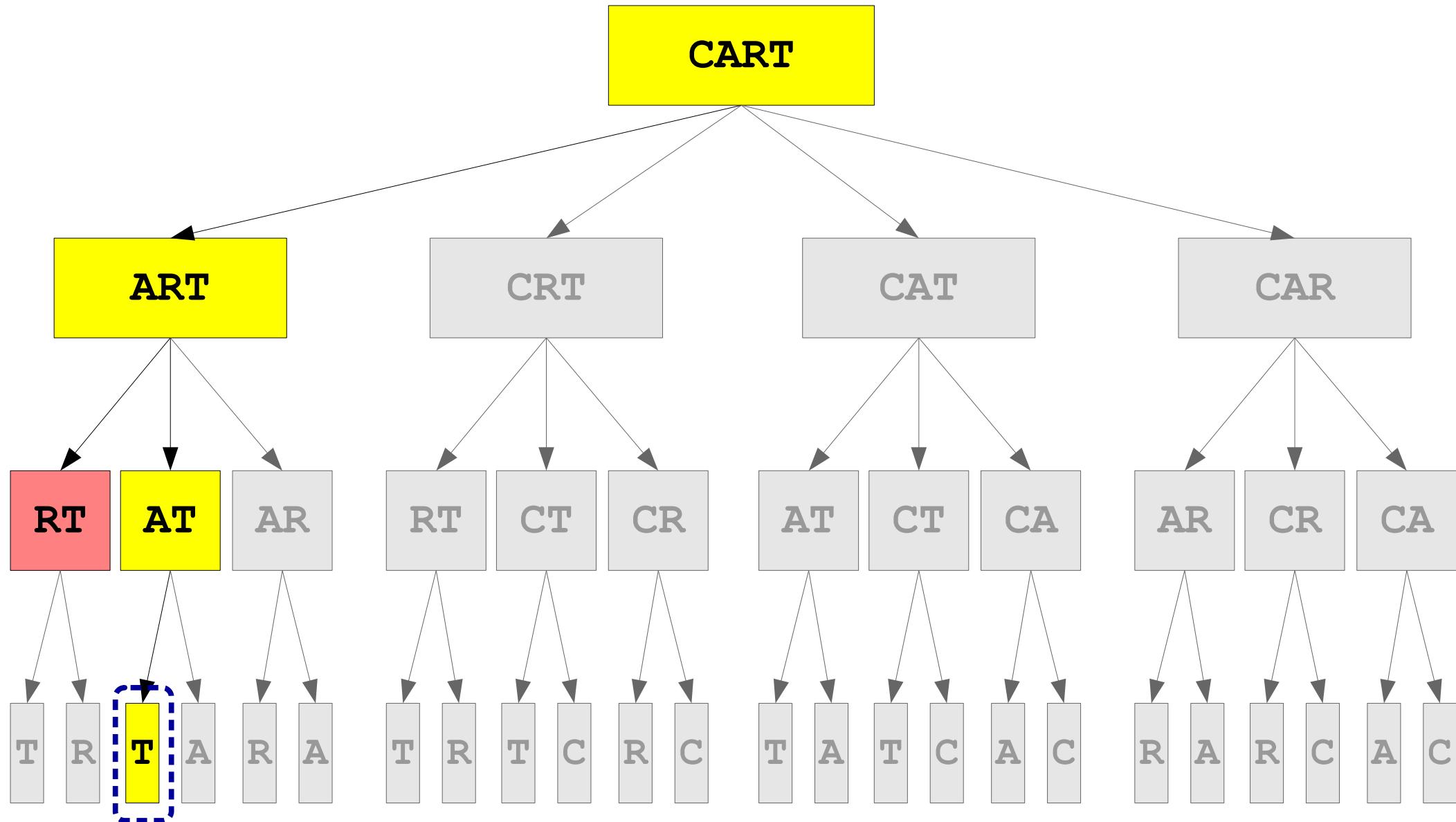
# Generating the Answer



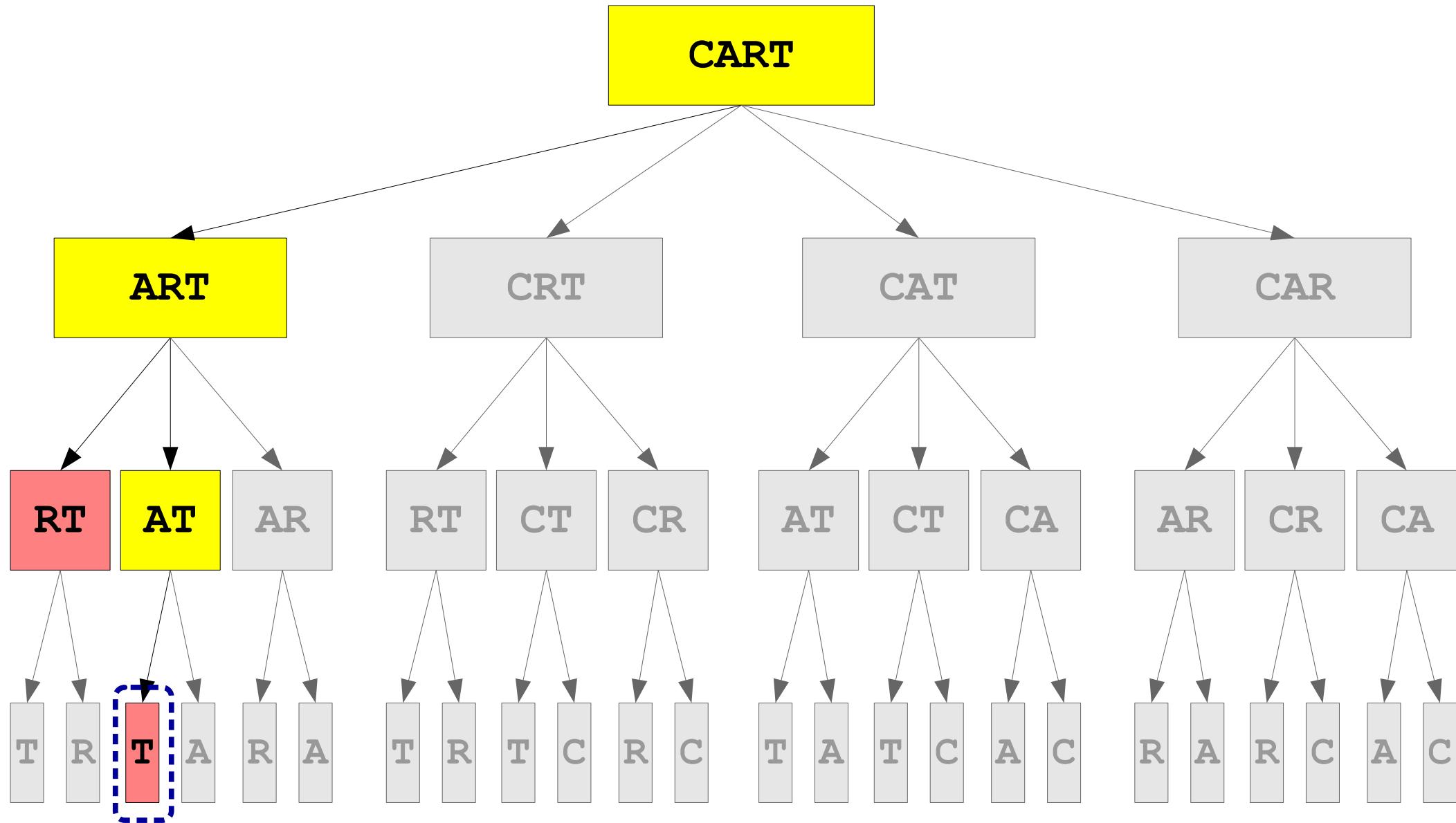
# Generating the Answer



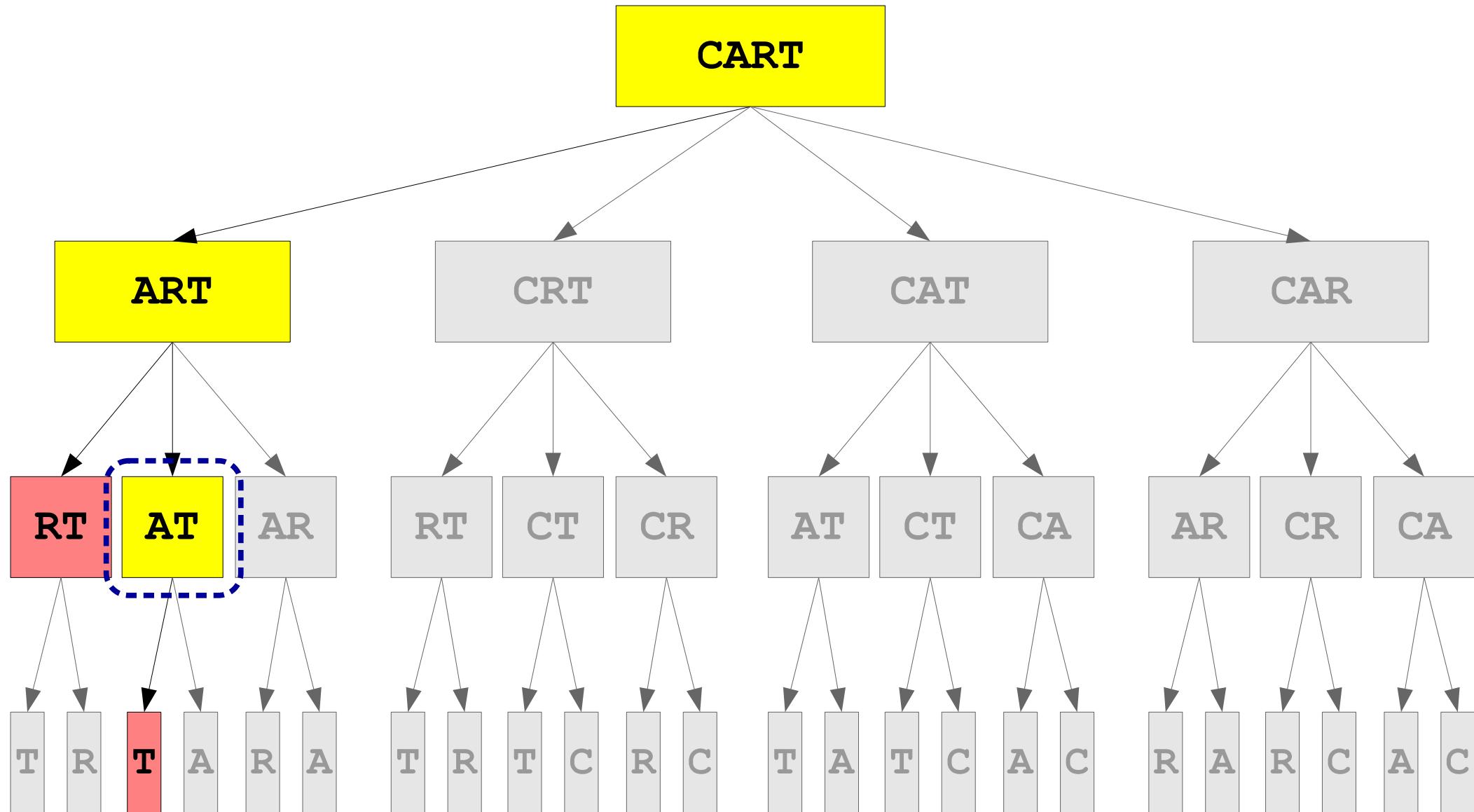
# Generating the Answer



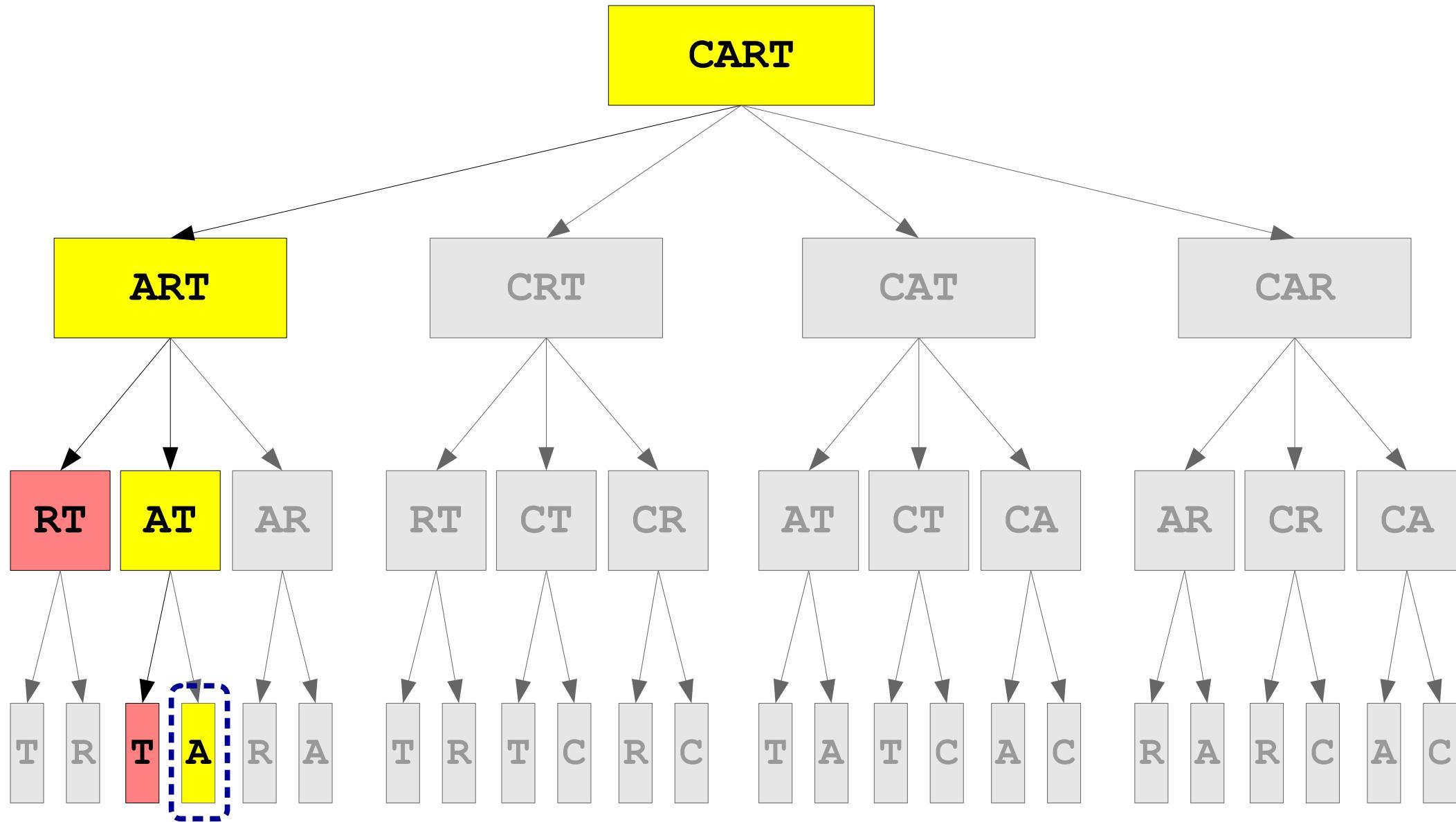
# Generating the Answer



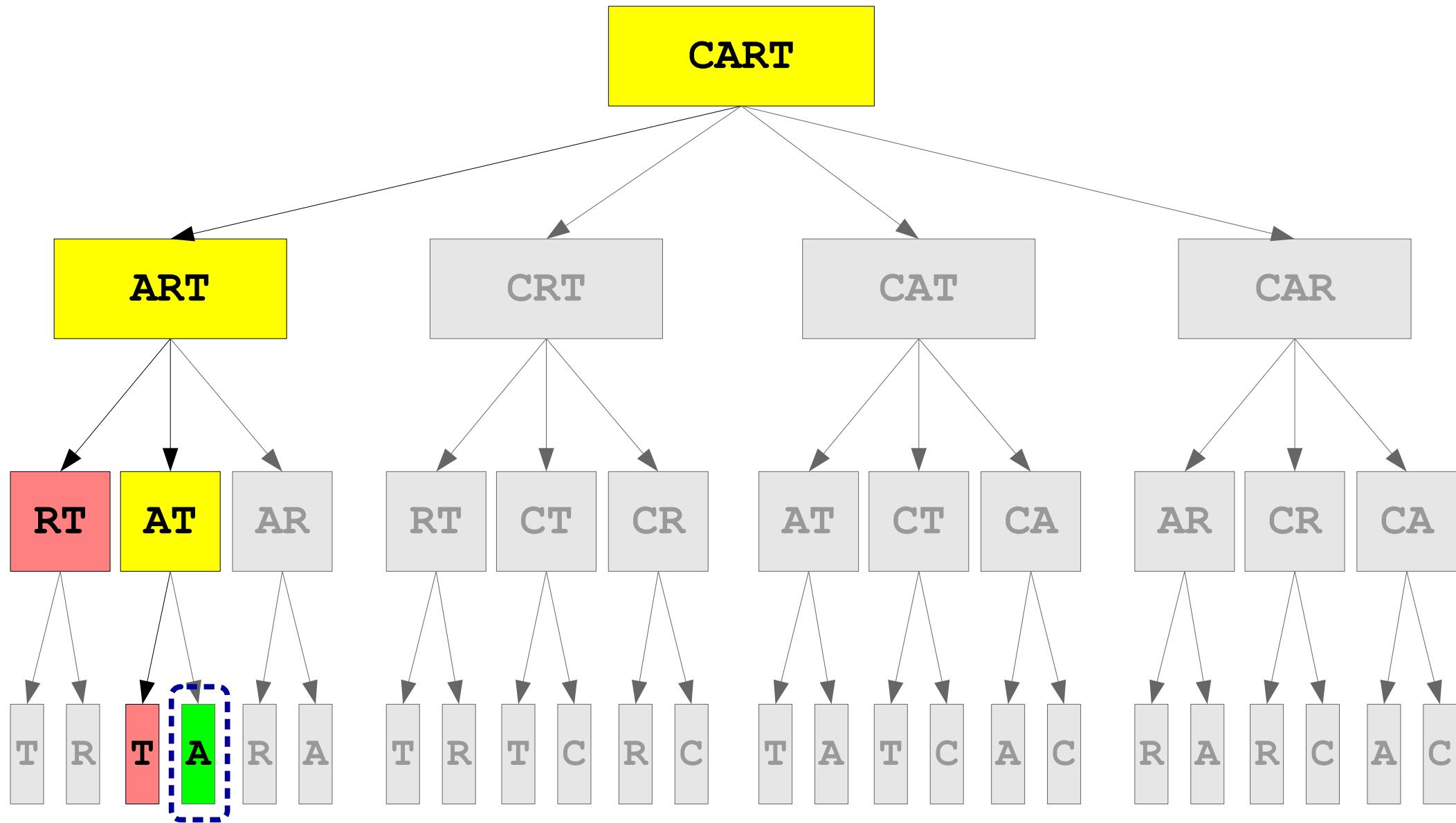
# Generating the Answer



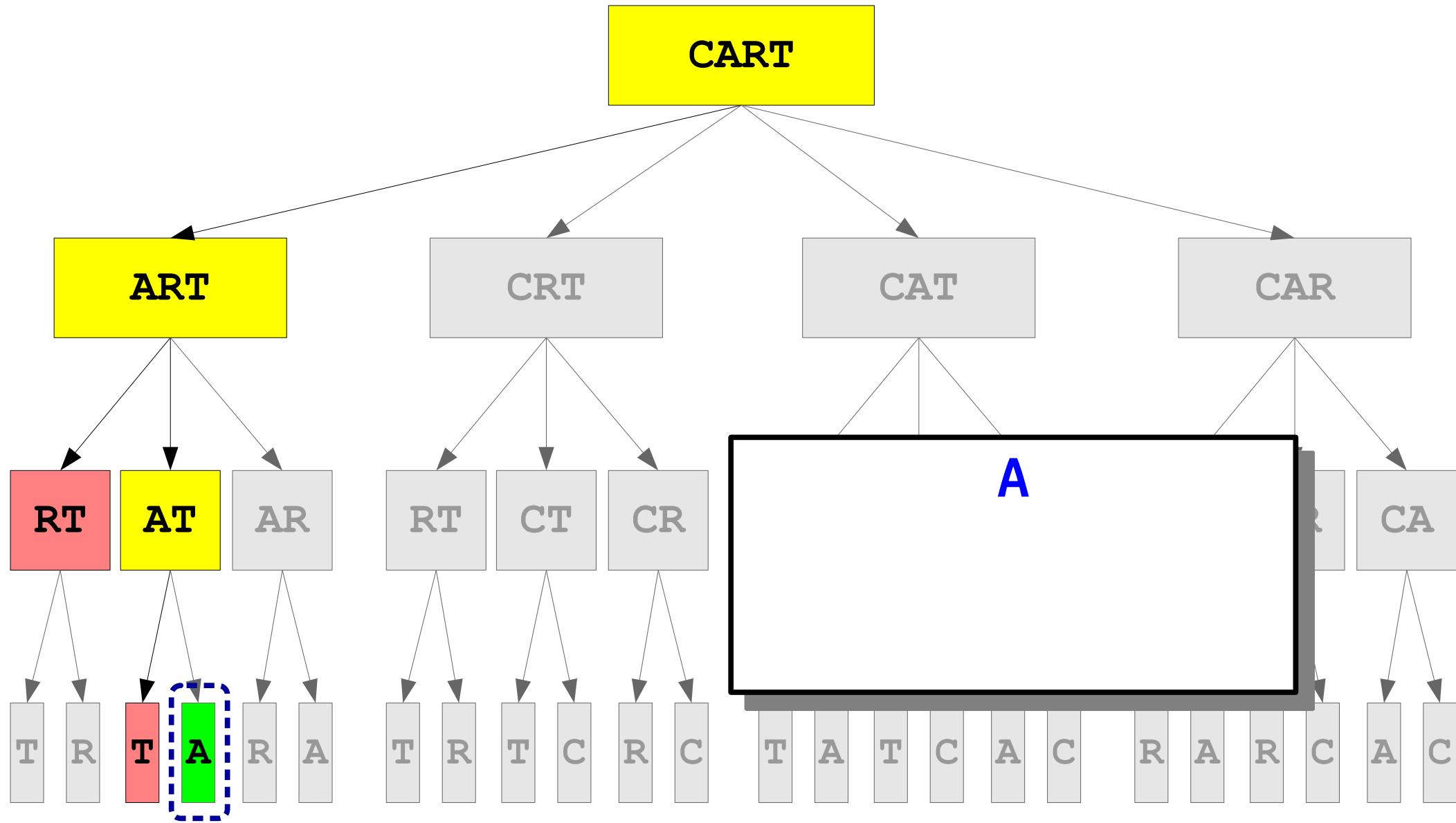
# Generating the Answer



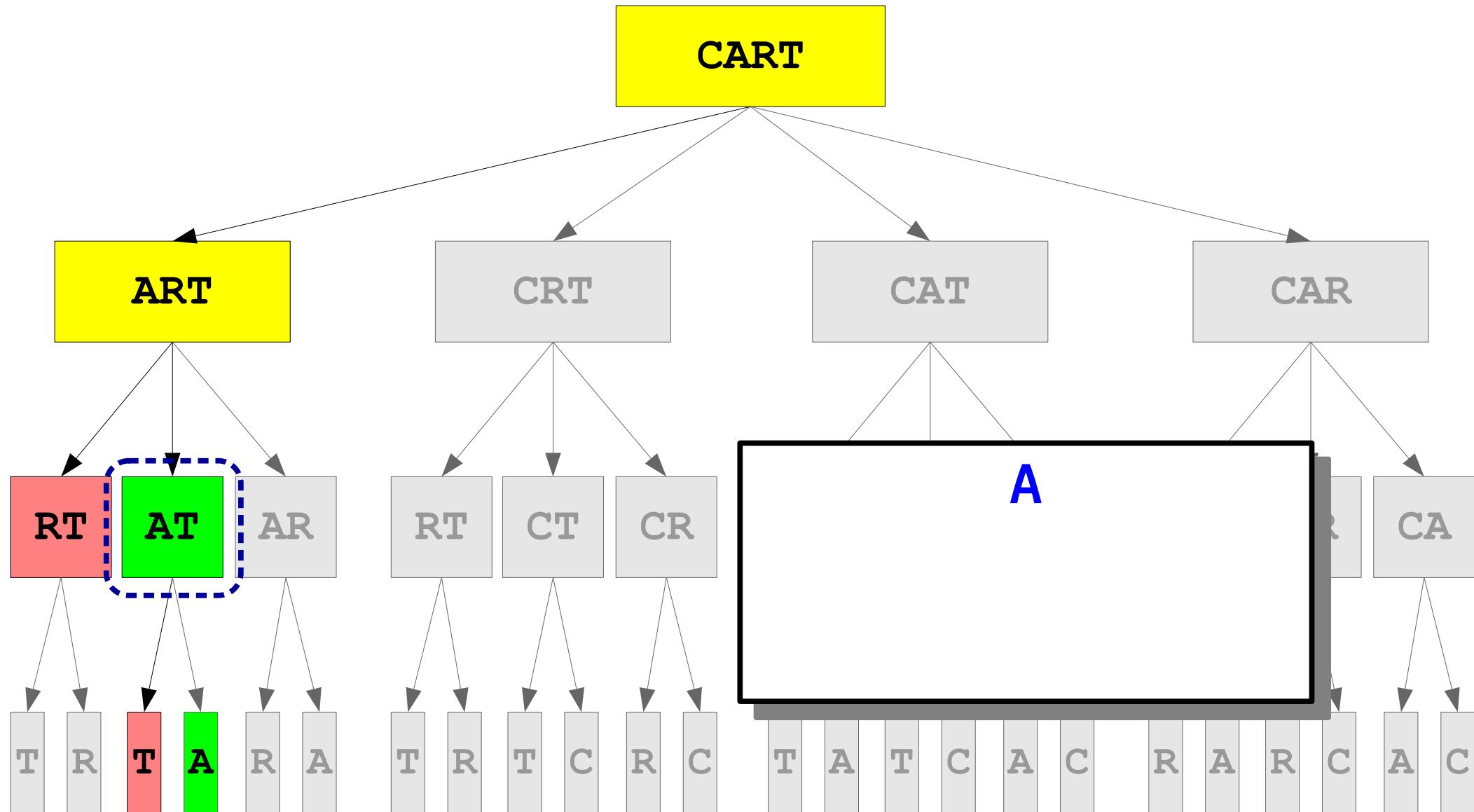
# Generating the Answer



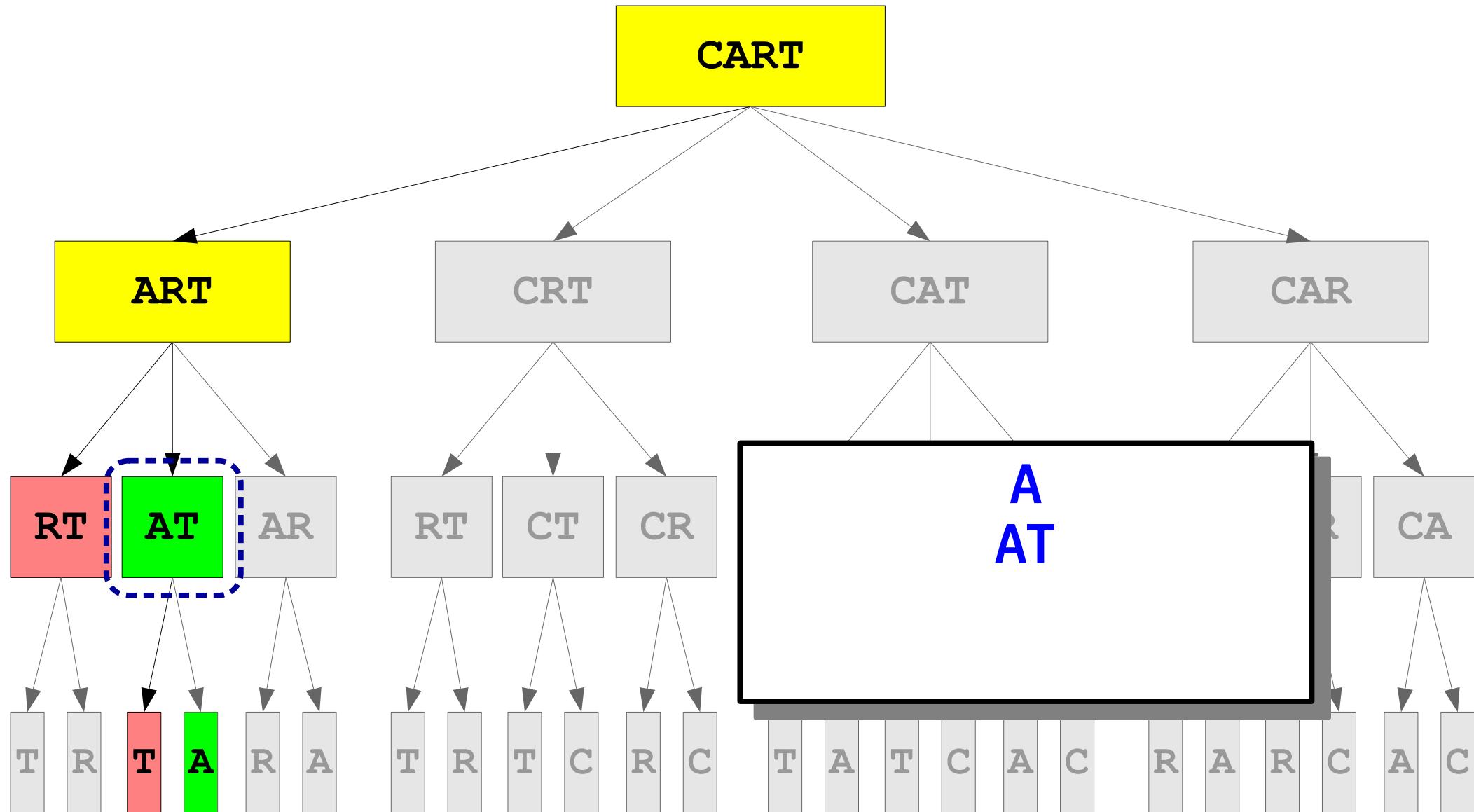
# Generating the Answer



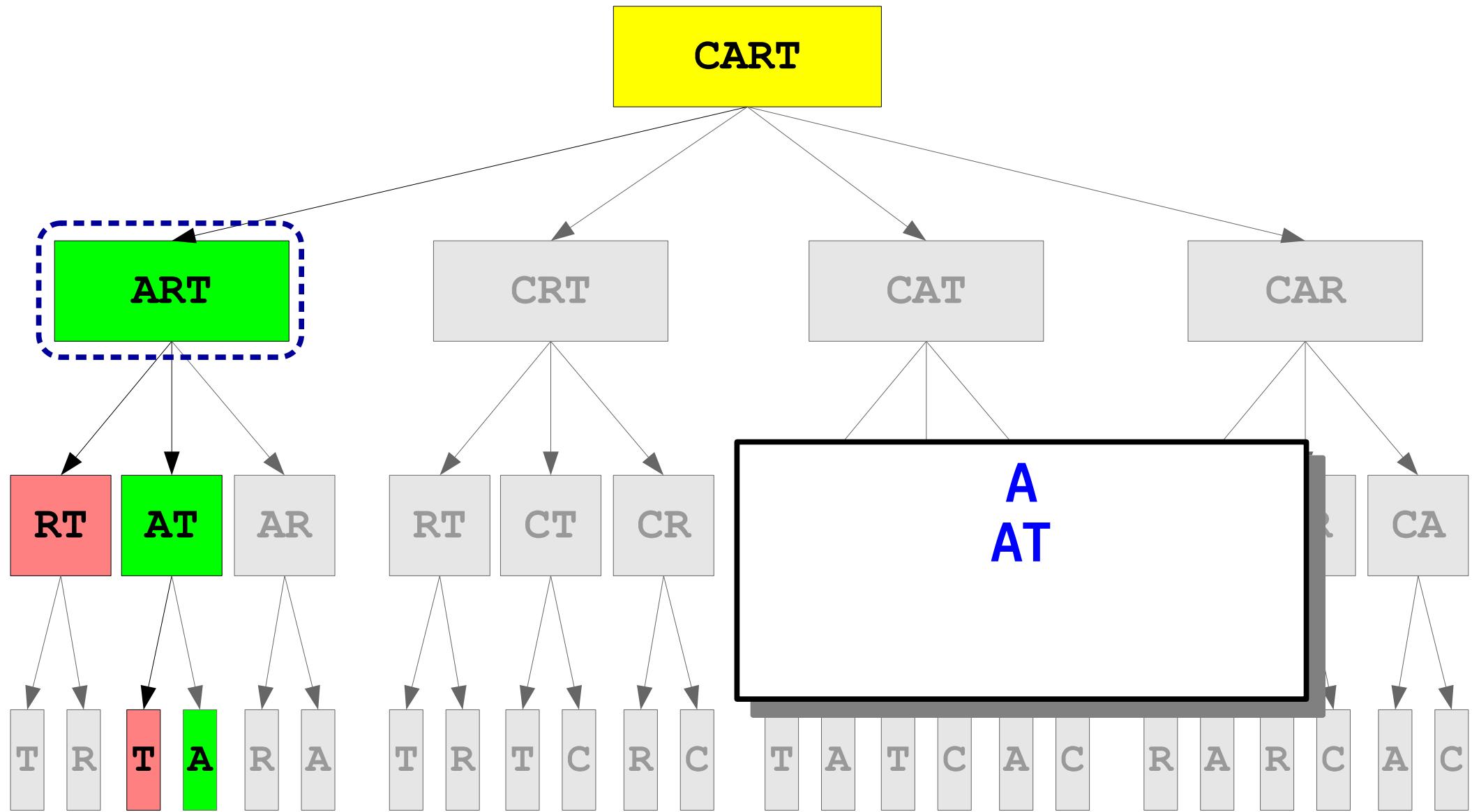
# Generating the Answer



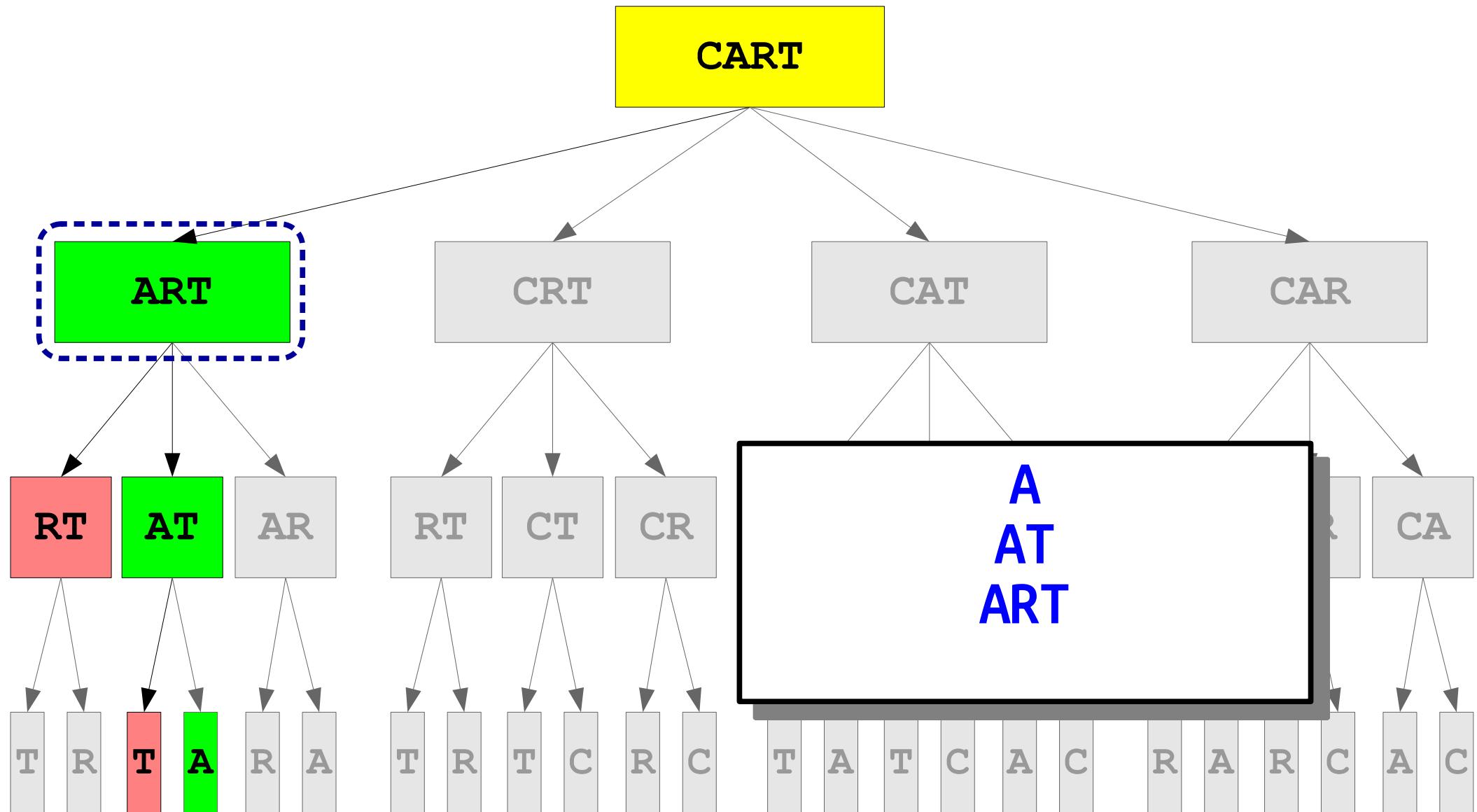
# Generating the Answer



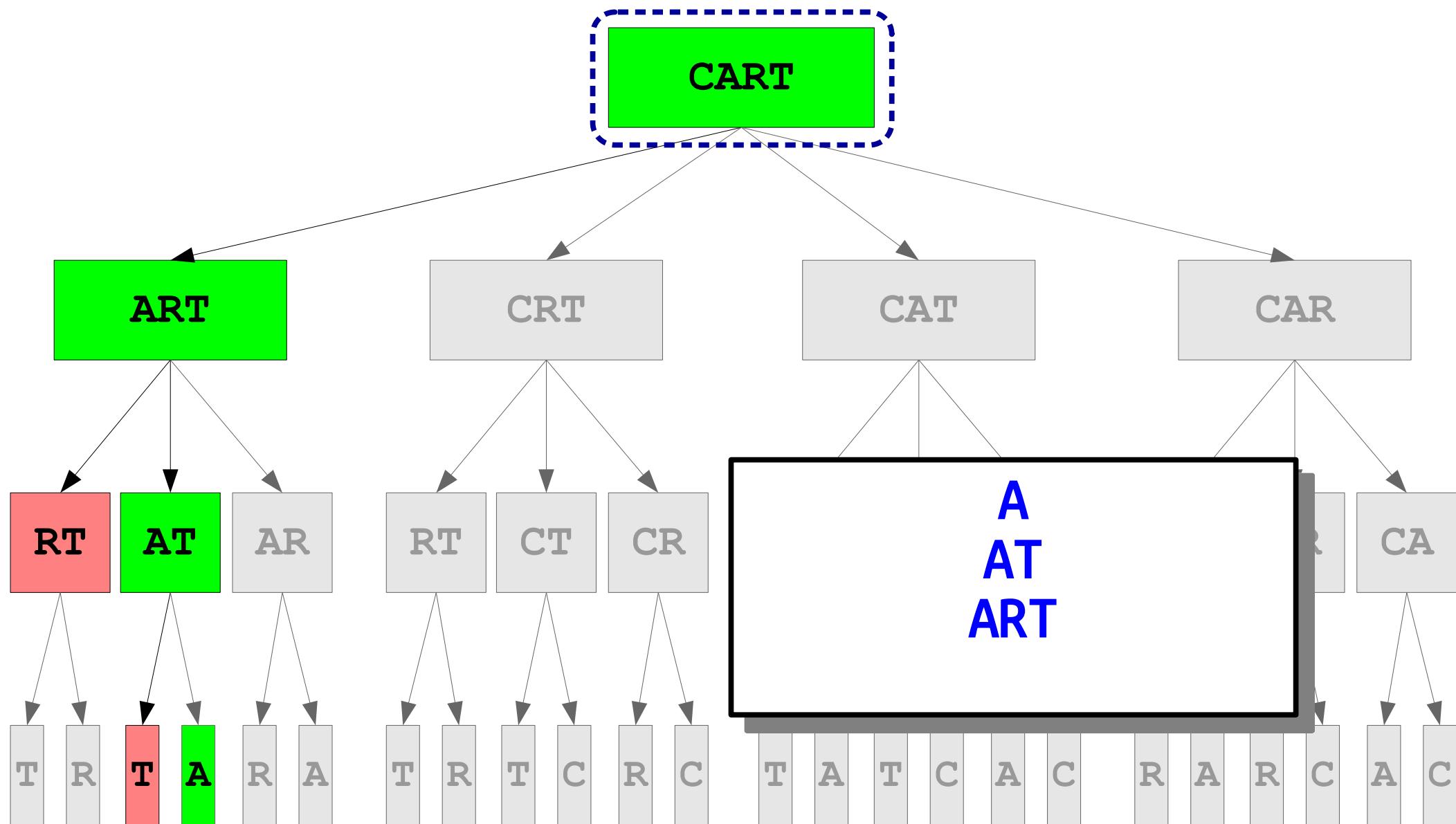
# Generating the Answer



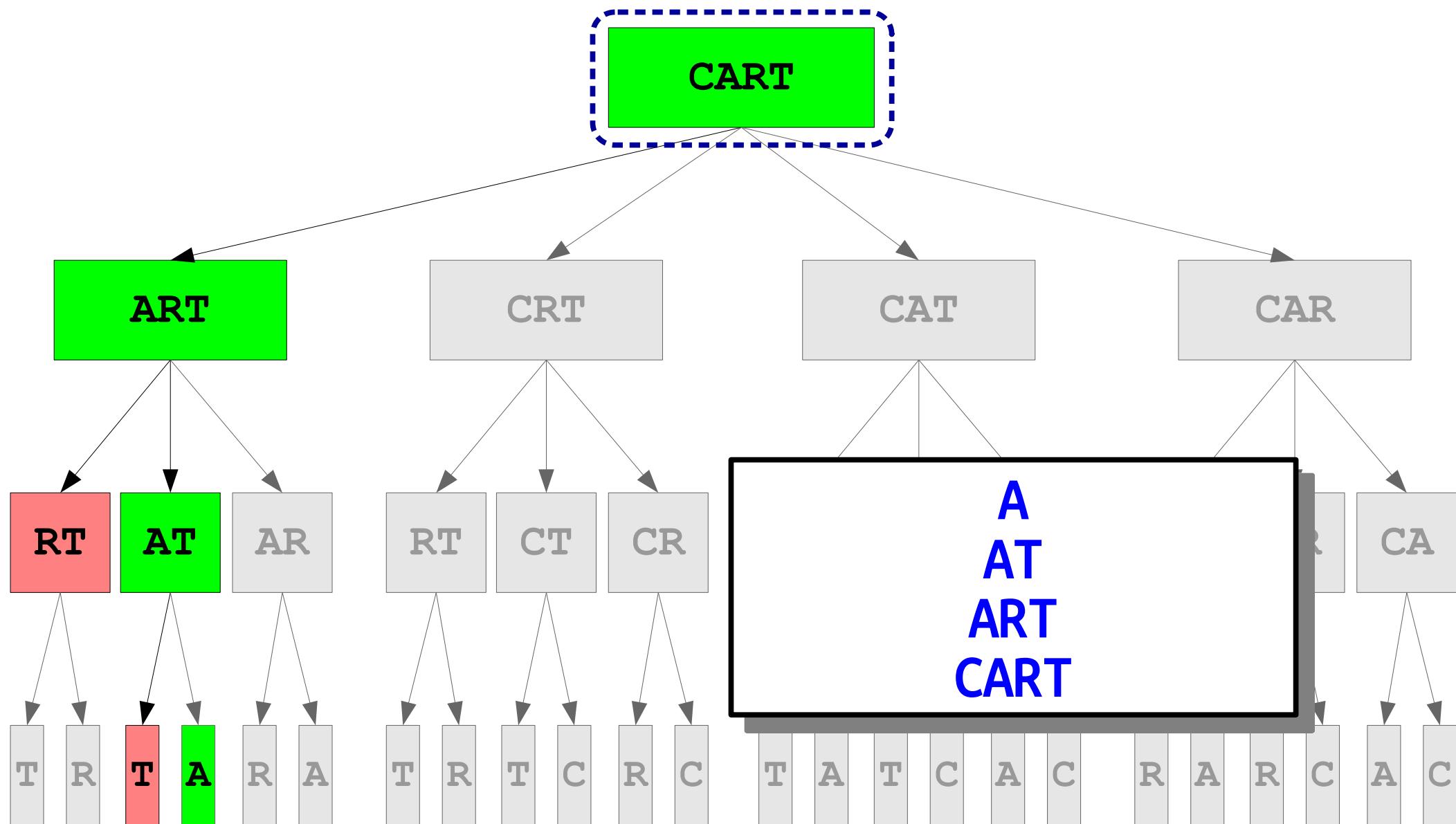
# Generating the Answer



# Generating the Answer



# Generating the Answer



# Another Backtracking Example

# A Great Tool of Science

1 <b>H</b> Hydrogen	2 <b>He</b> Helium
3 <b>Li</b> Lithium	4 <b>Be</b> Beryllium
11 <b>Na</b> Sodium	12 <b>Mg</b> Magnesi...
19 <b>K</b> Potassium	20 <b>Ca</b> Calcium
37 <b>Rb</b> Rubidium	38 <b>Sr</b> Strontium
55 <b>Cs</b> Caesium	56 <b>Ba</b> Barium
87 <b>Fr</b> Francium	88 <b>Ra</b> Radium
4 <b>Sc</b> Scandium	21 <b>Y</b> Yttrium
22 <b>Ti</b> Titanium	23 <b>V</b> Vanadium
24 <b>Cr</b> Chromium	25 <b>Mn</b> Mangan...
26 <b>Fe</b> Iron	27 <b>Co</b> Cobalt
28 <b>Ni</b> Nickel	29 <b>Cu</b> Copper
30 <b>Zn</b> Zinc	31 <b>Ga</b> Gallium
32 <b>Ge</b> Germani...	33 <b>As</b> Arsenic
34 <b>Se</b> Selenium	35 <b>Br</b> Bromine
36 <b>Kr</b> Krypton	37 <b>Xe</b> Xenon
40 <b>Zr</b> Zirconium	41 <b>Nb</b> Niobium
42 <b>Mo</b> Molybde...	43 <b>Tc</b> Techneti...
44 <b>Ru</b> Ruthenium	45 <b>Rh</b> Rhodium
46 <b>Pd</b> Palladium	47 <b>Ag</b> Silver
48 <b>Cd</b> Cadmium	49 <b>In</b> Indium
50 <b>Sn</b> Tin	51 <b>Sb</b> Antimony
52 <b>Te</b> Tellurium	53 <b>I</b> Iodine
53 <b>Br</b> Bromine	54 <b>Xe</b> Xenon
55 <b>Cs</b> Caesium	56 <b>Ba</b> Barium
57 <b>La</b> Lanthan...	72 <b>Hf</b> Hafnium
73 <b>Ta</b> Tantalum	74 <b>W</b> Tungsten
75 <b>Re</b> Rhenium	76 <b>Os</b> Osmium
77 <b>Ir</b> Iridium	78 <b>Pt</b> Platinum
79 <b>Au</b> Gold	80 <b>Hg</b> Mercury
81 <b>Tl</b> Thallium	82 <b>Pb</b> Lead
83 <b>Bi</b> Bismuth	84 <b>Po</b> Polonium
85 <b>At</b> Astatine	86 <b>Rn</b> Radon
87 <b>Fr</b> Francium	88 <b>Ra</b> Radium
89 <b>Ac</b> Actinium	104 <b>Rf</b> Rutherford...
105 <b>Db</b> Dubnium	106 <b>Sg</b> Seaborg...
107 <b>Bh</b> Bohrium	108 <b>Hs</b> Hassium
109 <b>Mt</b> Meitneri...	110 <b>Ds</b> Darmsta...
111 <b>Rg</b> Roentge...	112 <b>Cn</b> Coperni...
113 <b>Nh</b> Nihonium	114 <b>Fl</b> Flerovium
115 <b>Mc</b> Moscovi...	116 <b>Lv</b> Livermor...
117 <b>Ts</b> Tenness...	118 <b>Og</b> Oganess...
58 <b>Ce</b> Cerium	59 <b>Pr</b> Praseod...
60 <b>Nd</b> Neodym...	61 <b>Pm</b> Prometh...
62 <b>Sm</b> Samarium	63 <b>Eu</b> Europium
64 <b>Gd</b> Gadolini...	65 <b>Tb</b> Terbium
66 <b>Dy</b> Dysprosi...	67 <b>Ho</b> Holmium
68 <b>Er</b> Erbium	69 <b>Tm</b> Thulium
70 <b>Yb</b> Ytterbium	71 <b>Lu</b> Lutetium
90 <b>Th</b> Thorium	91 <b>Pa</b> Protacti...
92 <b>U</b> Uranium	93 <b>Np</b> Neptunium
94 <b>Pu</b> Plutonium	95 <b>Am</b> Americium
96 <b>Cm</b> Curium	97 <b>Bk</b> Berkelium
98 <b>Cf</b> Californi...	99 <b>Es</b> Einsteini...
100 <b>Fm</b> Fermium	101 <b>Md</b> Mendelev...
102 <b>No</b> Nobelium	103 <b>Lr</b> Lawrenc...

# Oooh! Letters!

1	<b>H</b> Hydrogen	2	<b>He</b> Helium
3	<b>Li</b> Lithium	4	<b>Be</b> Beryllium
11	<b>Na</b> Sodium	12	<b>Mg</b> Magnesi...
19	<b>K</b> Potassium	20	<b>Ca</b> Calcium
37	<b>Rb</b> Rubidium	38	<b>Sr</b> Strontium
55	<b>Cs</b> Caesium	56	<b>Ba</b> Barium
87	<b>Fr</b> Francium	88	<b>Ra</b> Radium
	<b>Sc</b> Scandium	39	<b>Y</b> Yttrium
	<b>Ti</b> Titanium	22	<b>V</b> Vanadium
	<b>Nb</b> Zirconium	23	<b>Cr</b> Chromium
	<b>Mo</b> Niobium	24	<b>Mn</b> Molybde...
	<b>Tc</b> Technetium	25	<b>Fe</b> Iron
	<b>Ru</b> Ruthenium	26	<b>Co</b> Cobalt
	<b>Rh</b> Rhodium	27	<b>Ni</b> Nickel
	<b>Pd</b> Palladium	28	<b>Cu</b> Copper
	<b>Ag</b> Silver	29	<b>Zn</b> Zinc
	<b>Cd</b> Cadmium	30	<b>Ga</b> Gallium
	<b>In</b> Indium	31	<b>Ge</b> Germani...
	<b>Sn</b> Tin	32	<b>As</b> Arsenic
	<b>Sb</b> Antimony	33	<b>Se</b> Selenium
	<b>Te</b> Tellurium	34	<b>Br</b> Bromine
	<b>I</b> Iodine	35	<b>Kr</b> Krypton
	<b>Xe</b> Xenon	36	<b>Rn</b> Radon
	<b>La</b> Lanthan...	57	<b>At</b> Astatine
	<b>Hf</b> Hafnium	72	<b>Ta</b> Tantalum
	<b>W</b> Tungsten	73	<b>Re</b> Rhenium
	<b>Os</b> Osmium	74	<b>Ir</b> Iridium
	<b>Pt</b> Platinum	75	<b>Au</b> Gold
	<b>Hg</b> Mercury	76	<b>Tl</b> Thallium
	<b>Pb</b> Lead	77	<b>Pb</b> Bismuth
	<b>Bi</b> Bismuth	80	<b>Po</b> Polonium
	<b>At</b> Astatine	81	<b>At</b> Rhenium
	<b>Rf</b> Rutherford	104	<b>Db</b> Dubnium
	<b>Sg</b> Seaborg...	105	<b>Bh</b> Bohrium
	<b>Hs</b> Hassium	106	<b>Mt</b> Meitneri...
	<b>Ds</b> Darmsta...	107	<b>Rg</b> Roentge...
	<b>Rg</b> Copernic...	108	<b>Cn</b> Copernic...
	<b>Nh</b> Nihonium	109	<b>Nh</b> Nihonium
	<b>Fl</b> Flerovium	110	<b>Mc</b> Moscov...
	<b>Mc</b> Moscov...	111	<b>Lv</b> Livermor...
	<b>Ts</b> Tenness...	112	<b>Ts</b> Tenness...
	<b>Og</b> Oganess...	113	<b>Og</b> Oganess...
	<b>Ce</b> Cerium	58	<b>Pr</b> Praseod...
	<b>Th</b> Thorium	59	<b>Nd</b> Neodym...
	<b>Pa</b> Protacti...	60	<b>Pm</b> Prometh...
	<b>U</b> Uranium	61	<b>Sm</b> Samarium
	<b>Np</b> Neptunium	62	<b>Eu</b> Europium
	<b>Pu</b> Plutonium	63	<b>Gd</b> Gadolini...
	<b>Am</b> Americium	64	<b>Tb</b> Terbium
	<b>Cm</b> Curium	65	<b>Dy</b> Dysprosi...
	<b>Bk</b> Berkelium	66	<b>Ho</b> Holmium
	<b>Cf</b> Californi...	67	<b>Er</b> Erbium
	<b>Es</b> Einsteini...	68	<b>Tm</b> Thulium
	<b>Fm</b> Fermium	69	<b>Yb</b> Ytterbium
	<b>Md</b> Mendelev...	70	<b>Lu</b> Lutetium
	<b>No</b> Nobelium	71	<b>Lr</b> Lawrenc...

# Oooh! Letters!

**Br**ea**k**ing  
**B**a**d**



# 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

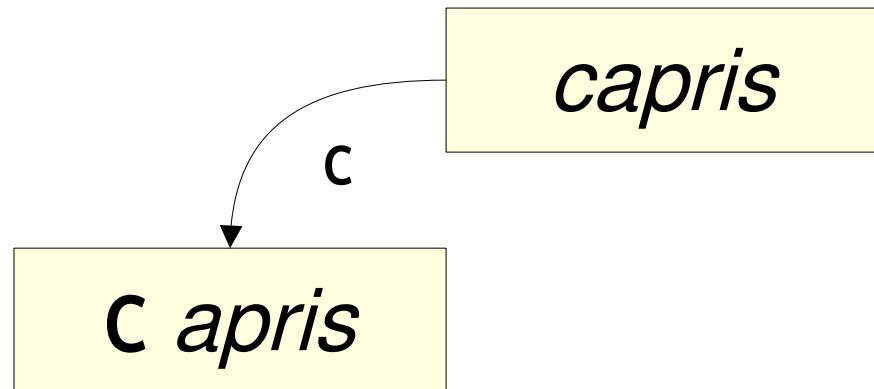
- Given a word, can we spell it using only symbols from the periodic table?
- And, if so, how?

NoTiCe ThAt

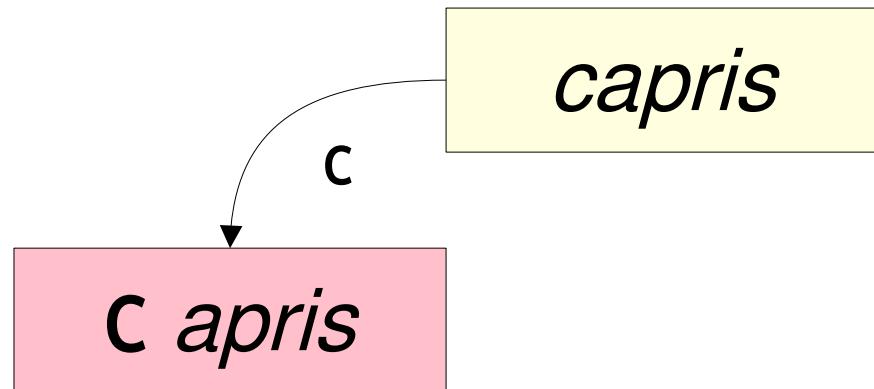
# NoTiCe ThAt

*capris*

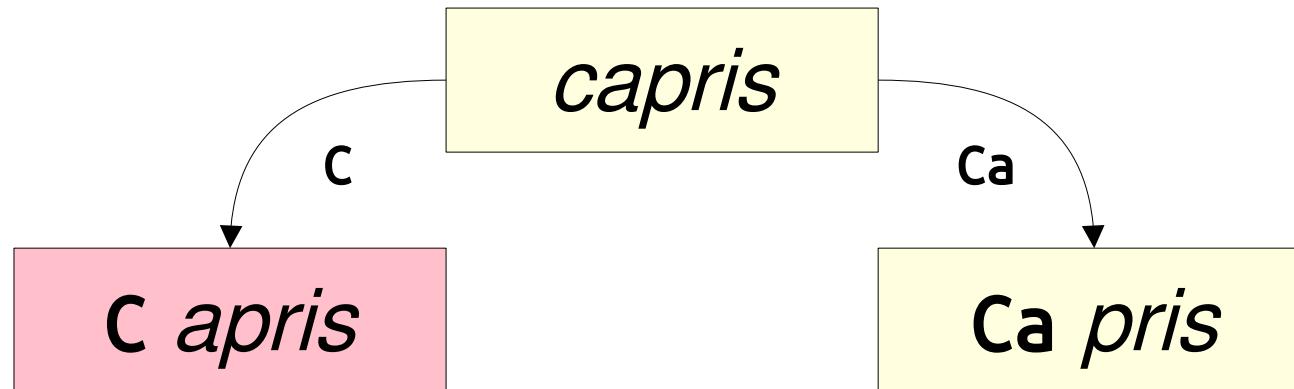
# NoTiCe ThAt



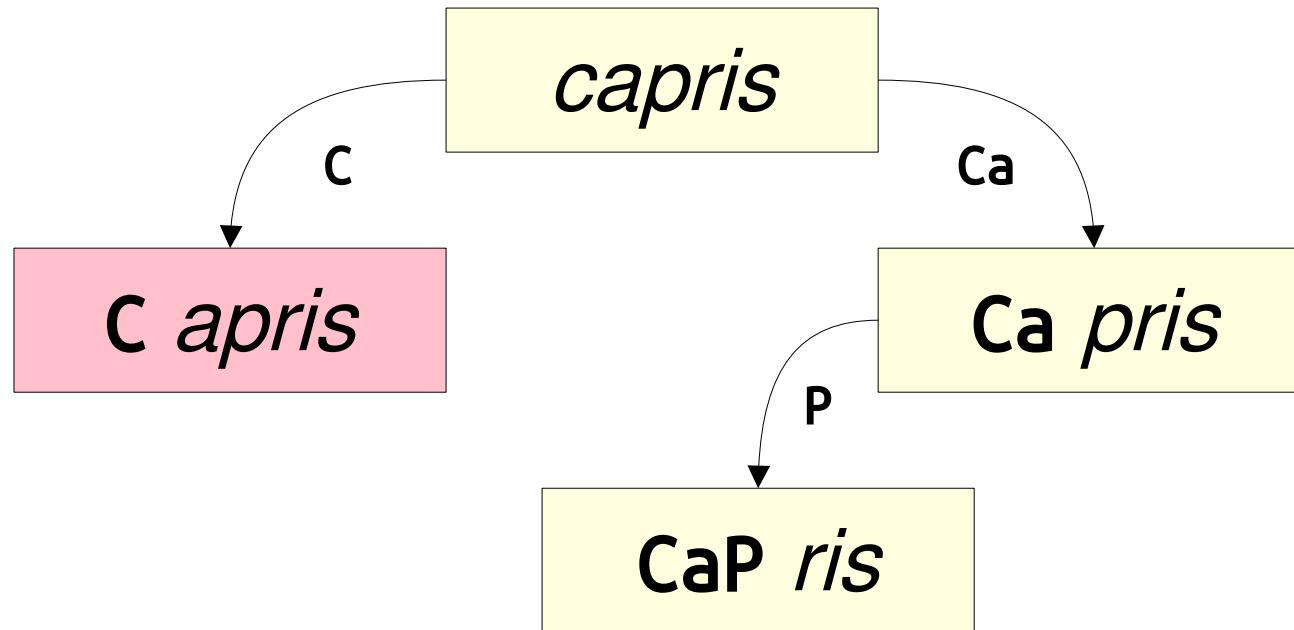
# NoTiCe ThAt



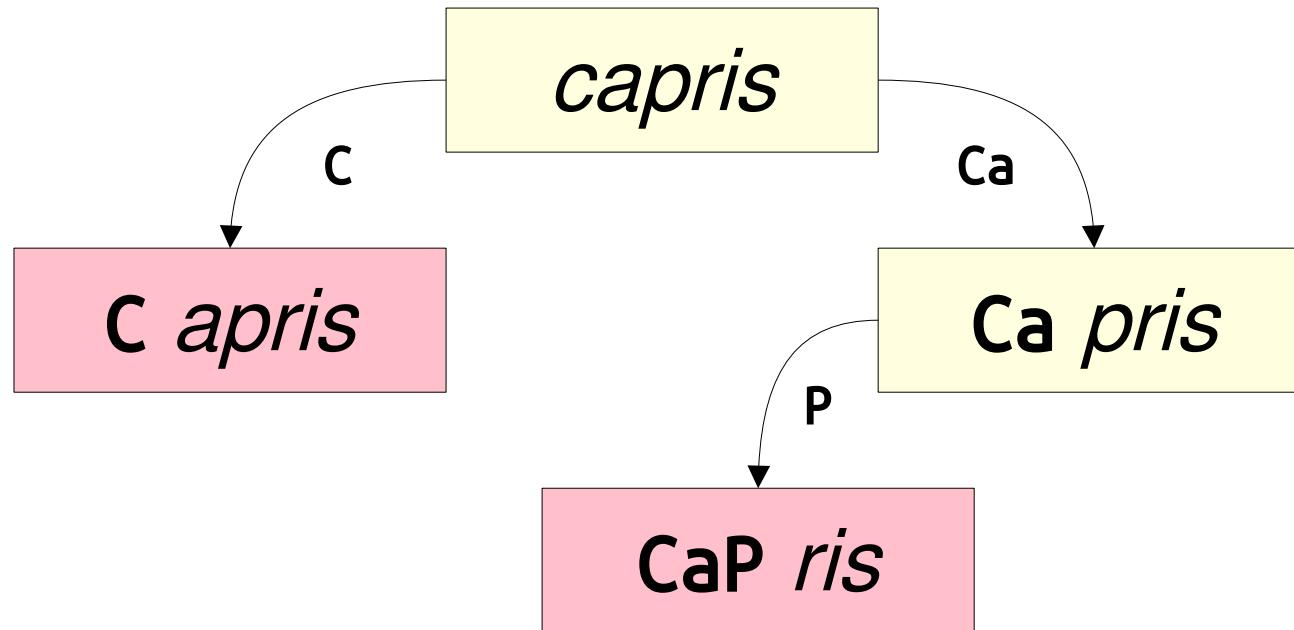
# NoTiCe ThAt



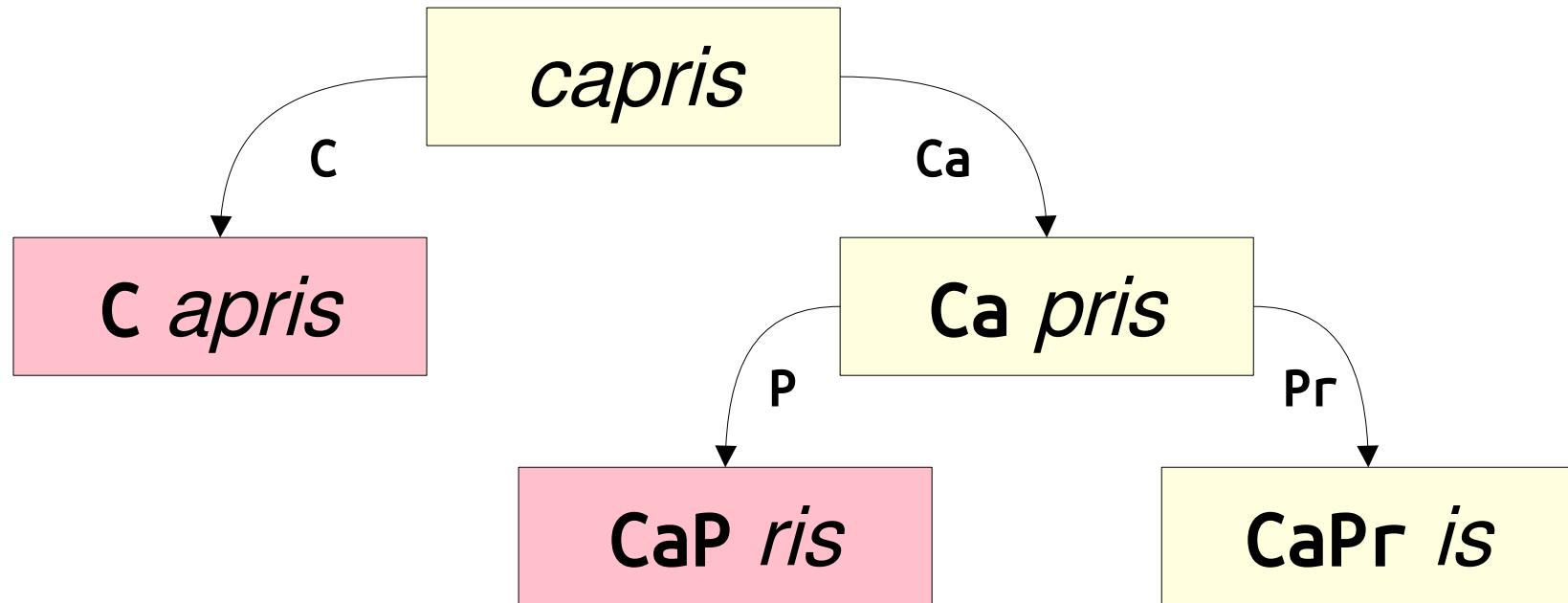
# NoTiCe ThAt



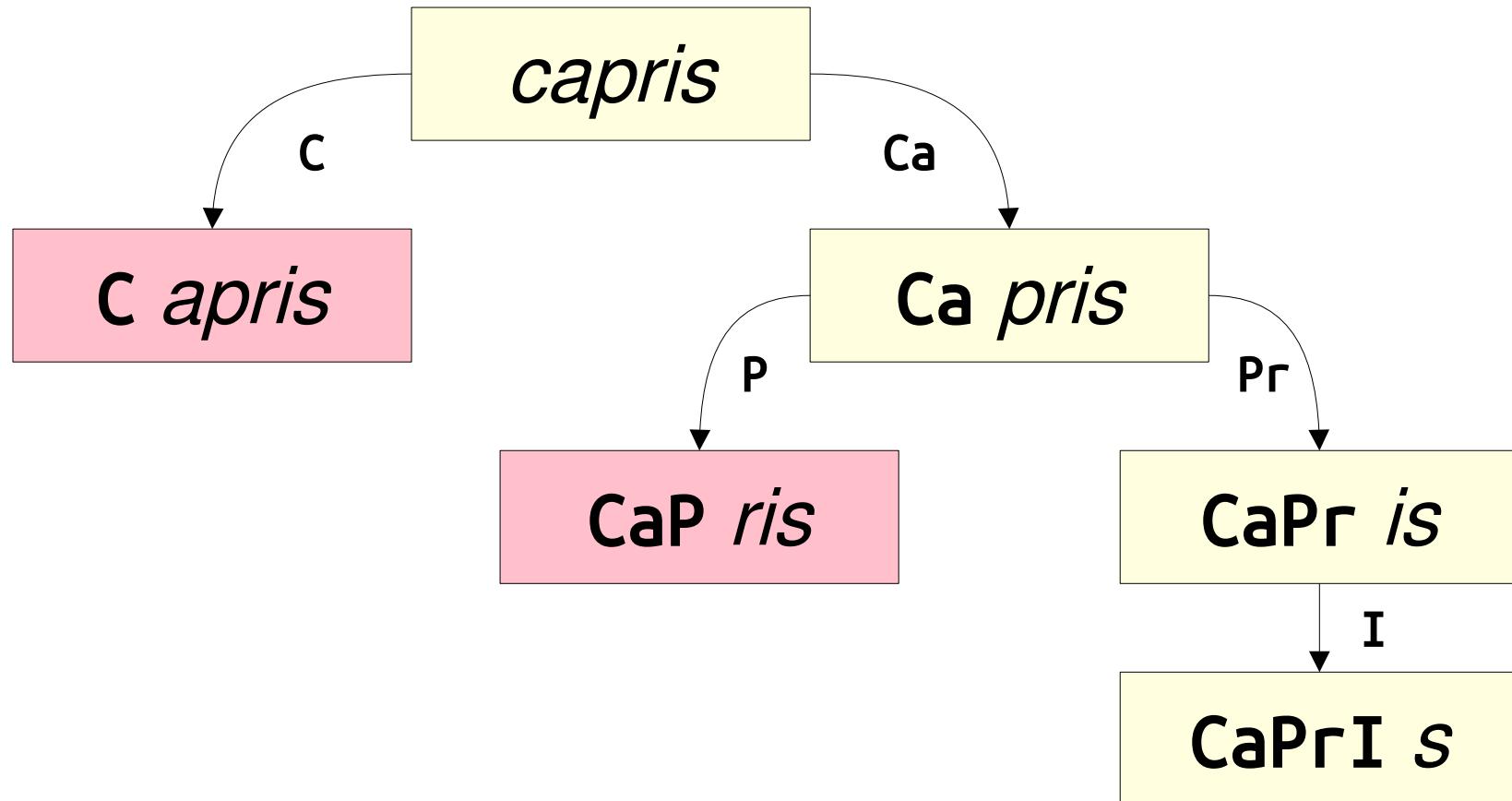
# NoTiCe ThAt



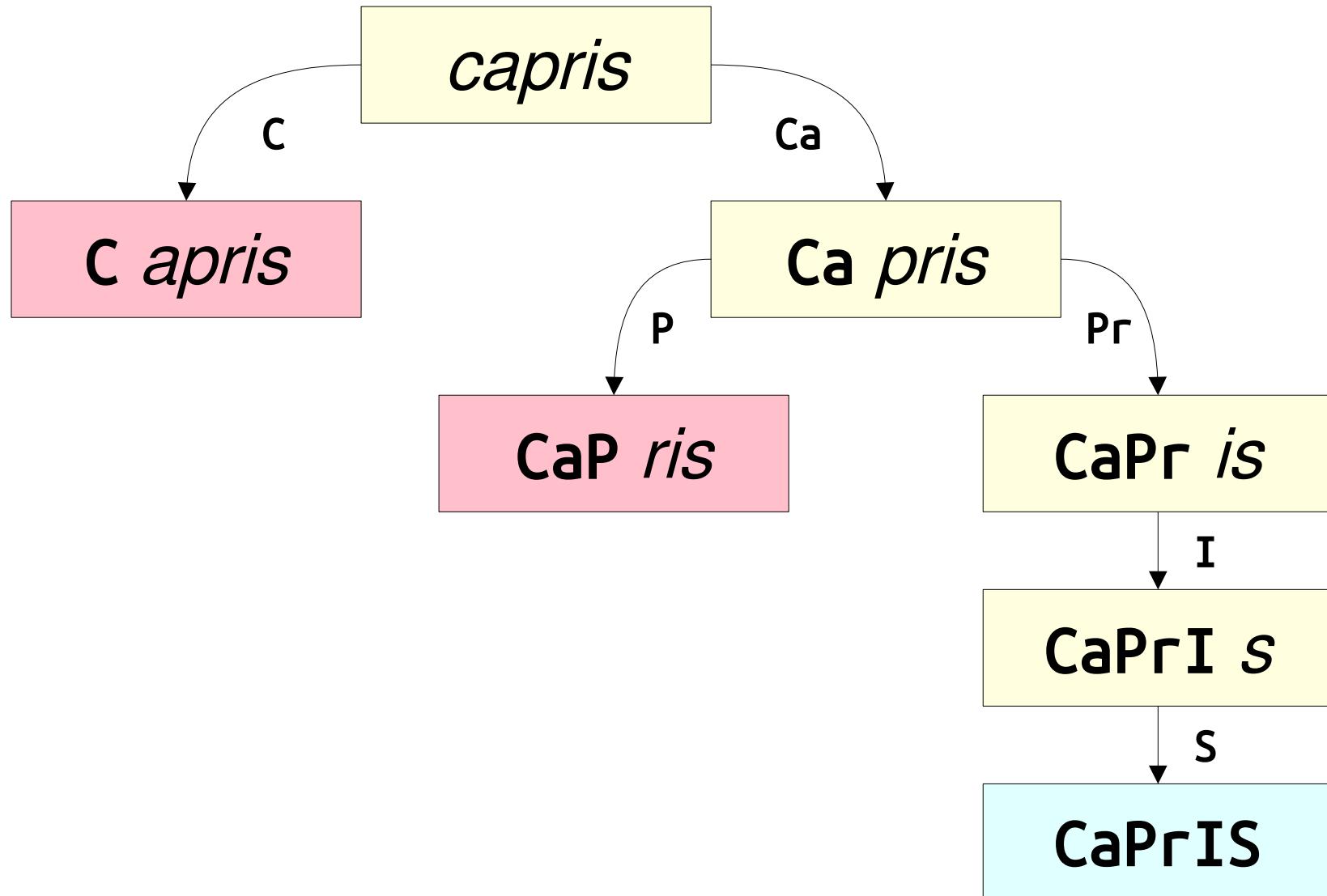
# NoTiCe ThAt



# NoTiCe ThAt



# NoTiCe ThAt



# 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!