Thinking Recursively

Part V

Outline for Today

- ***Recursive Backtracking***
  - Finding a needle in a haystack.
- ***On Tenacity***
  - Computational grit!
- ***Outparameters***
  - Sending data out of functions.
- ***Dense Crosswords***
  - Solving a challenging word puzzle.

What's Wrong With This Code?

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

Formulate a hypothesis, but ***don't post anything in chat just yet***.

What's Wrong With This Code?

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

Now, ***private chat me your best guess***. Not sure? Just answer "??"

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

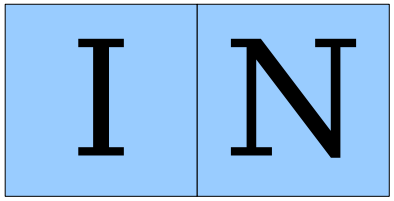
One Solution

S	I	N	G

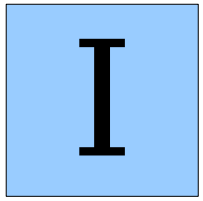
One Solution

S	I	N

One Solution



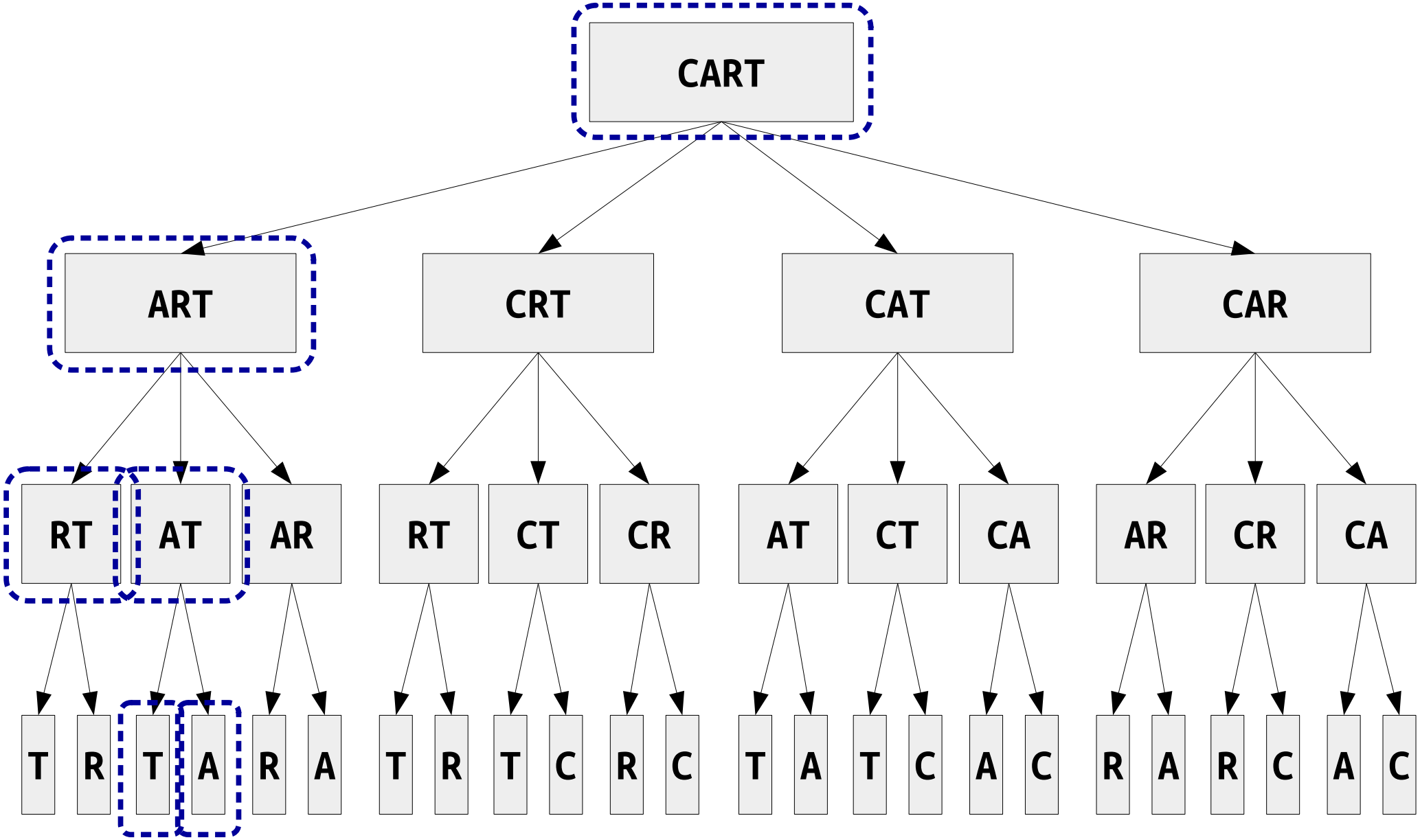
One Solution



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 (isShrinkable(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 (isShrinkable(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 isShrinkable(shrunken, english); // Bad idea!
    }
    return false;
}
```

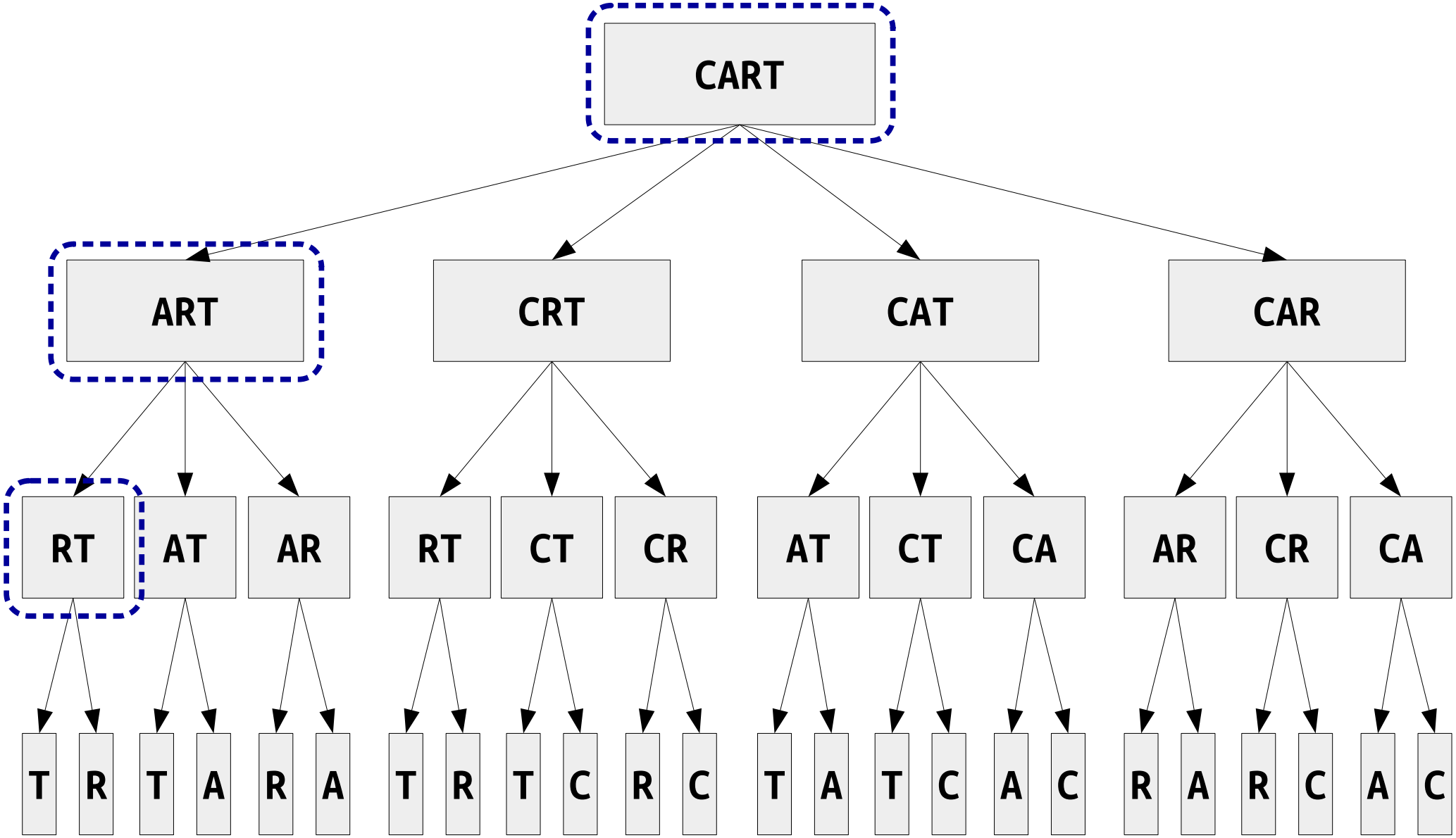
```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        return isShrinkable(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;
}
```

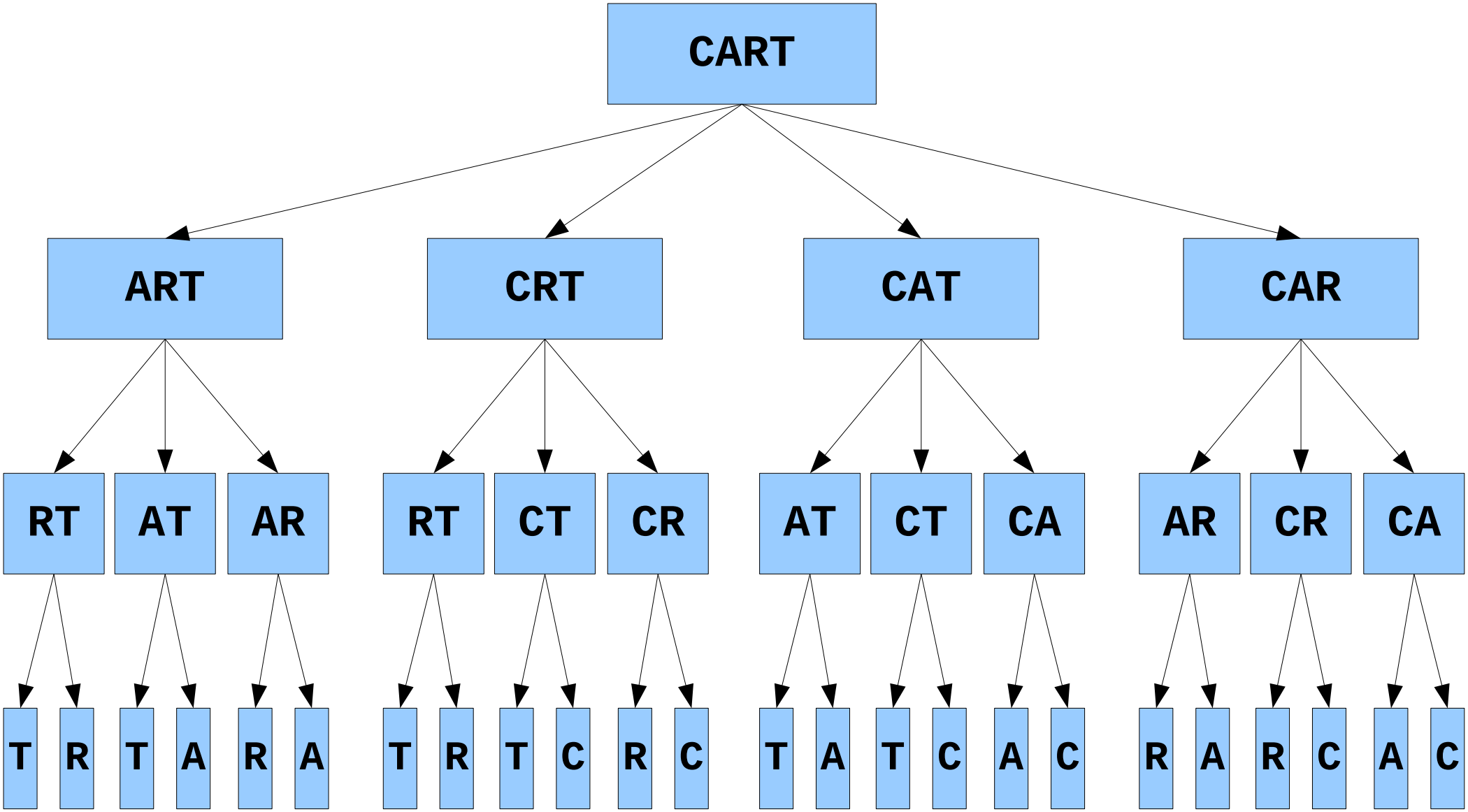
Note that *if* the recursive call succeeds, *then* we return success. If it doesn't succeed, that doesn't mean we've failed – it just means we need to try out the next option.

How do we know we’re correct?

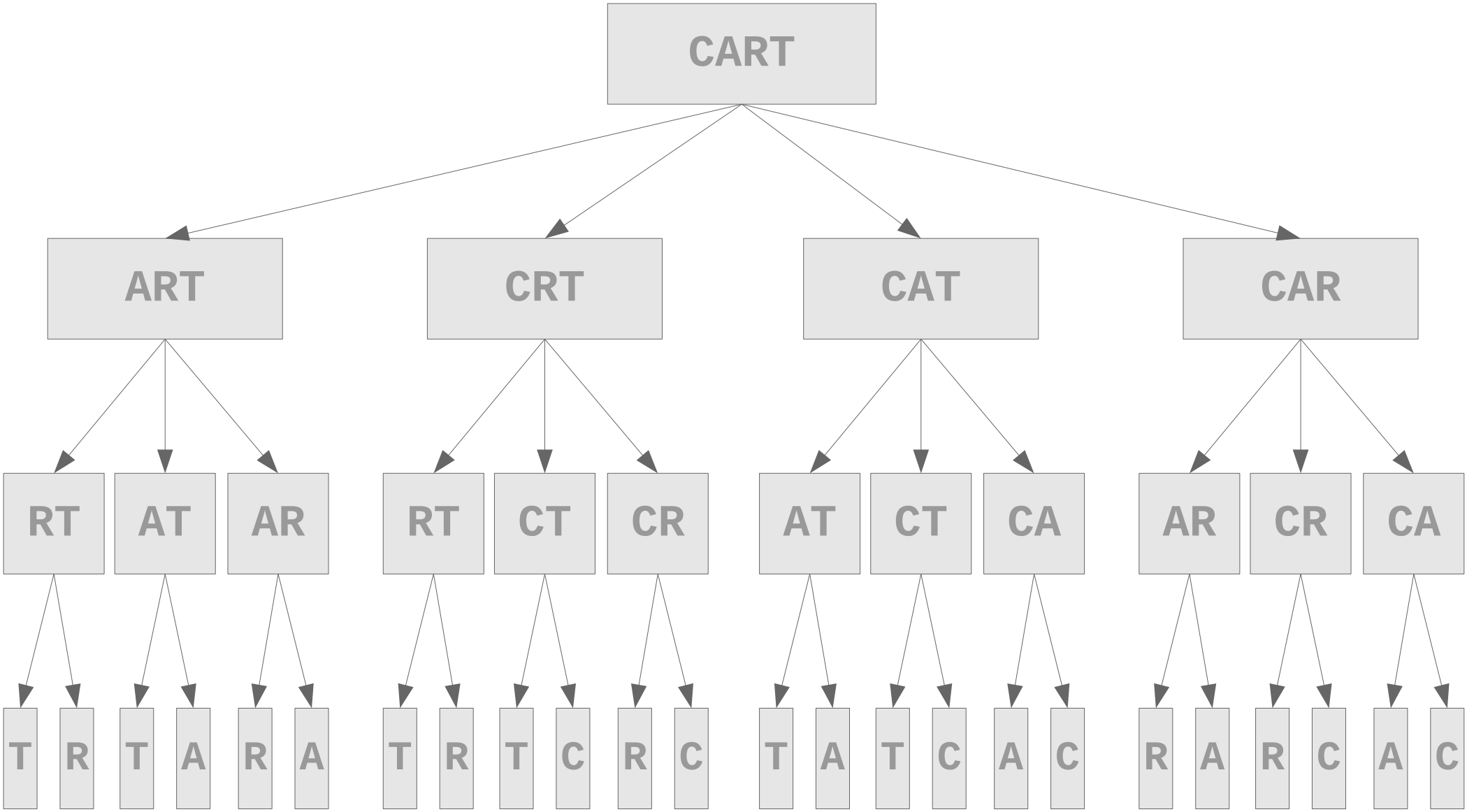
Output Parameters

- An ***output parameter*** (or ***outparam***) is a parameter to a function that stores the result of that function.
- The caller passes the parameter by reference, and the function overwrites the value.
- They're often used with backtracking searches:
  - The return value says whether a solution exists.
  - If one does, it's loaded into the outparameter.
- In general, avoid using outparameters when you can get away with returning values normally.
  - In our case, we need to return both "did we succeed?" and "if so, what did we get?," which means our return value is already used up.

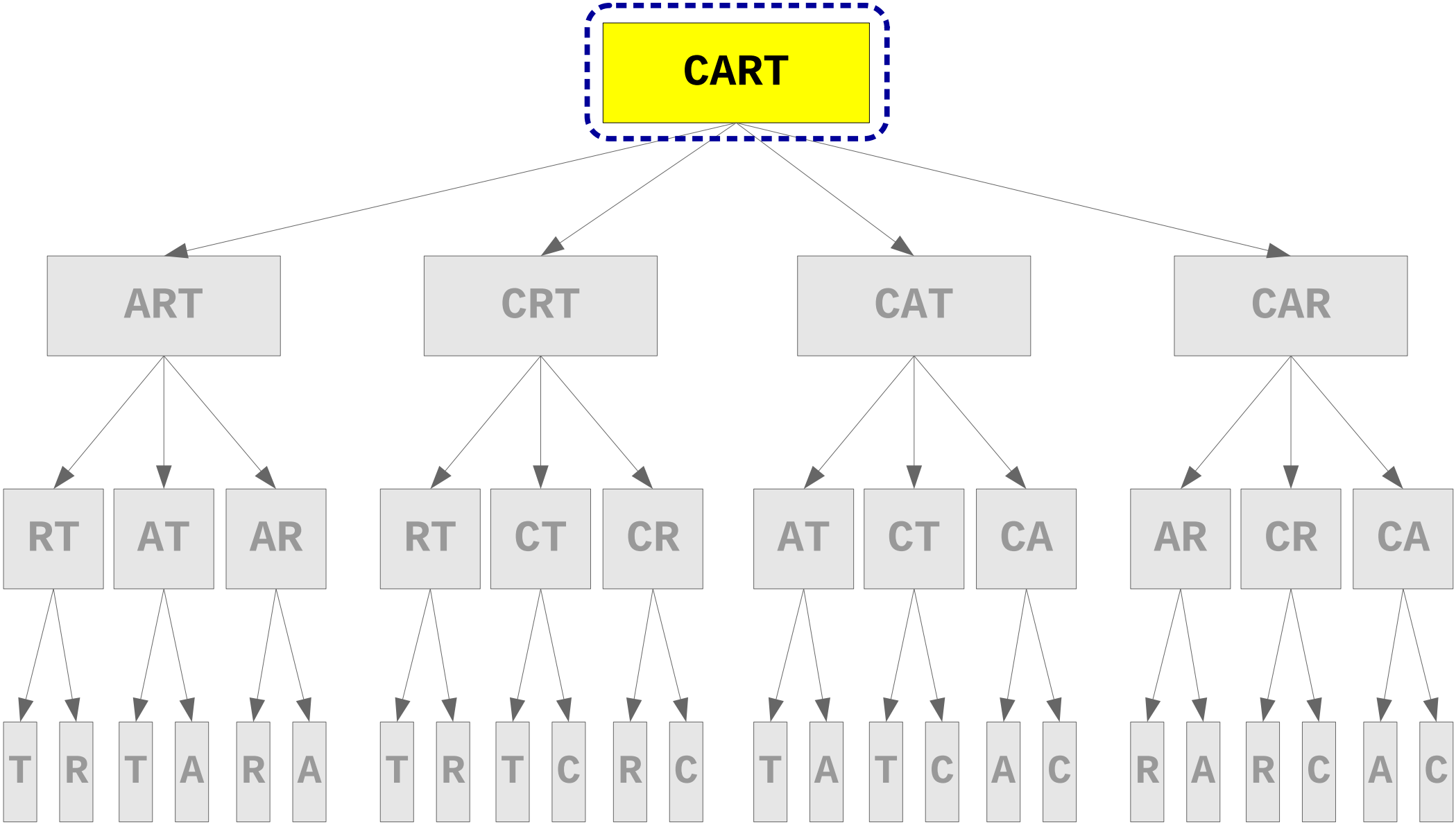
Generating the Answer



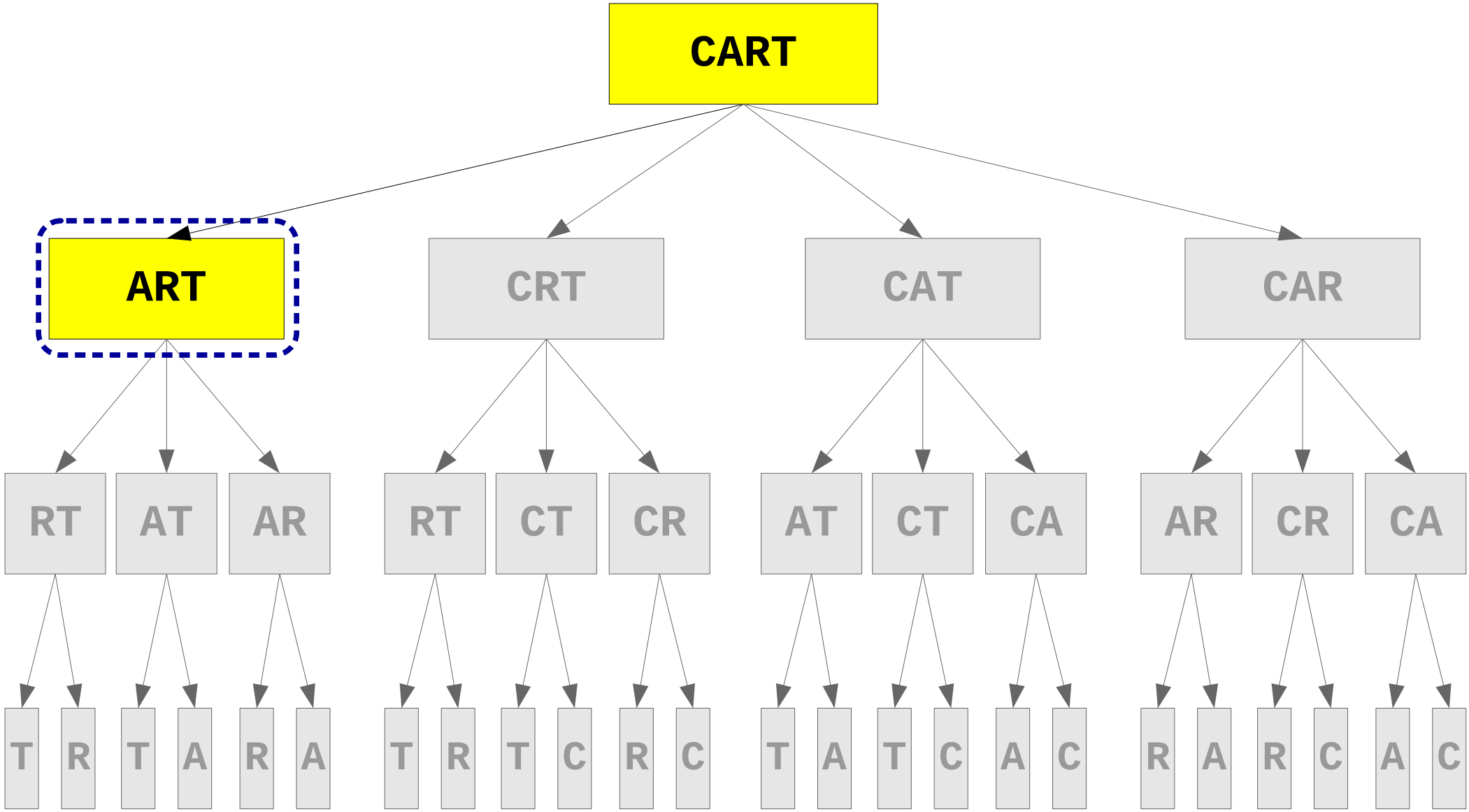
Generating the Answer



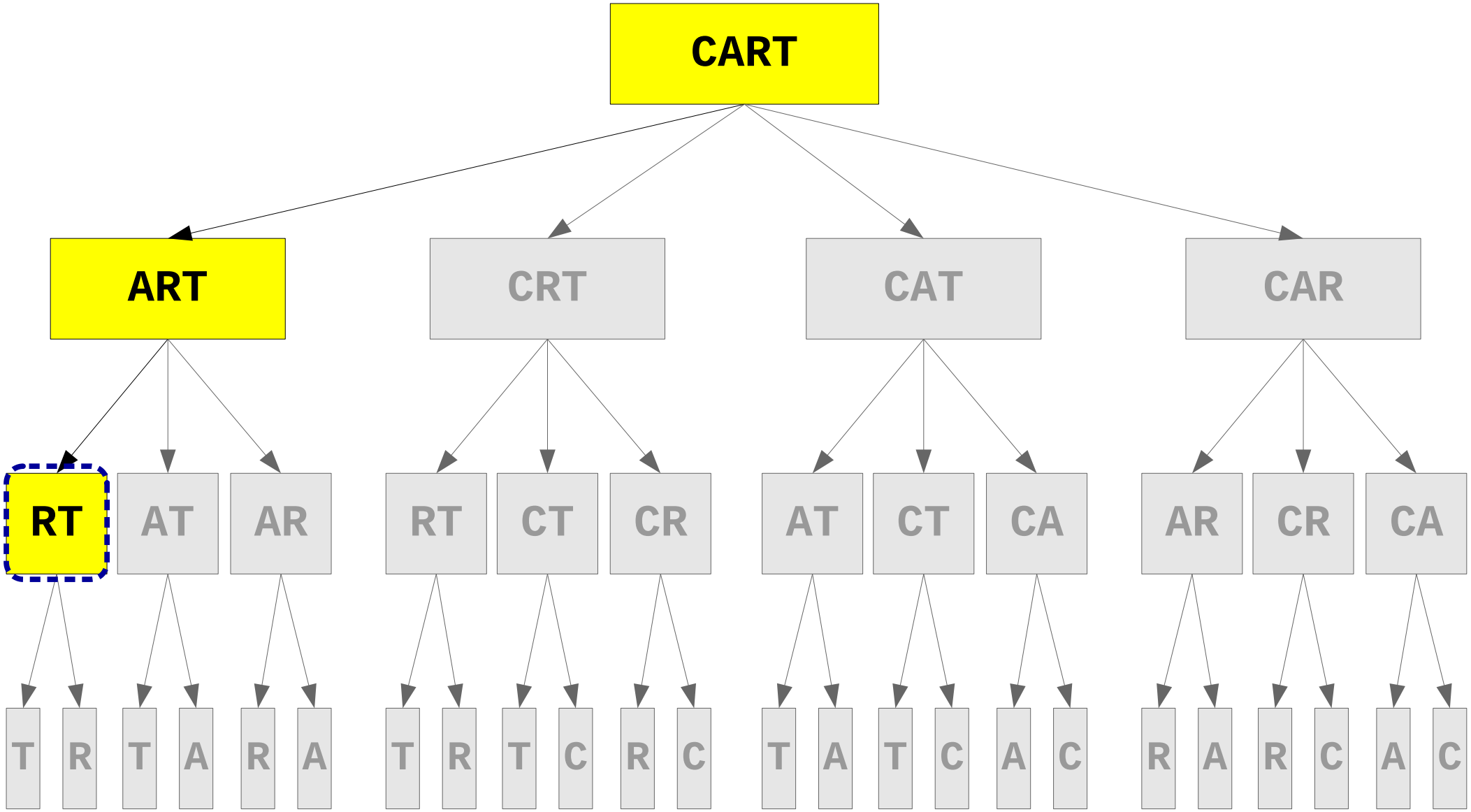
Generating the Answer



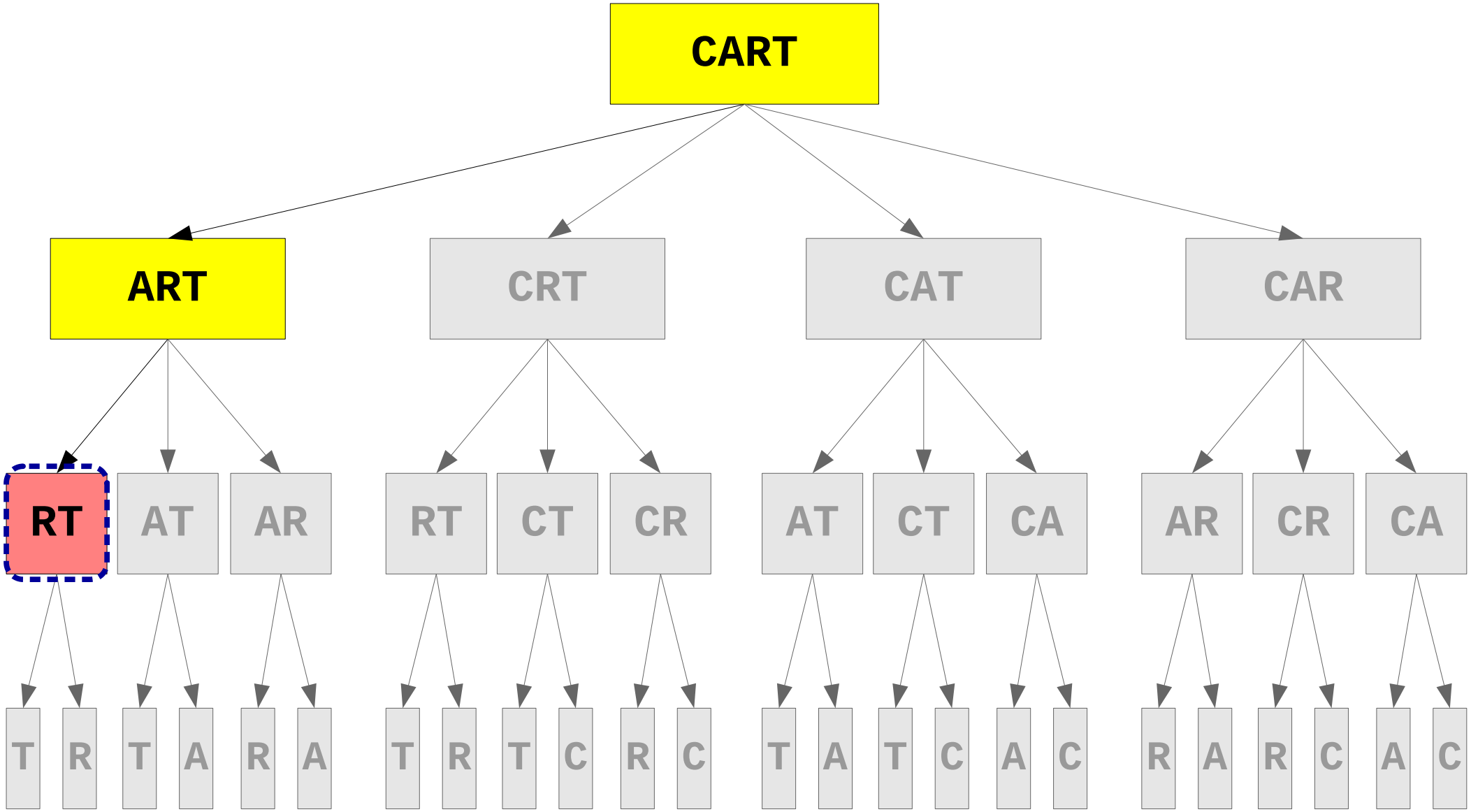
Generating the Answer



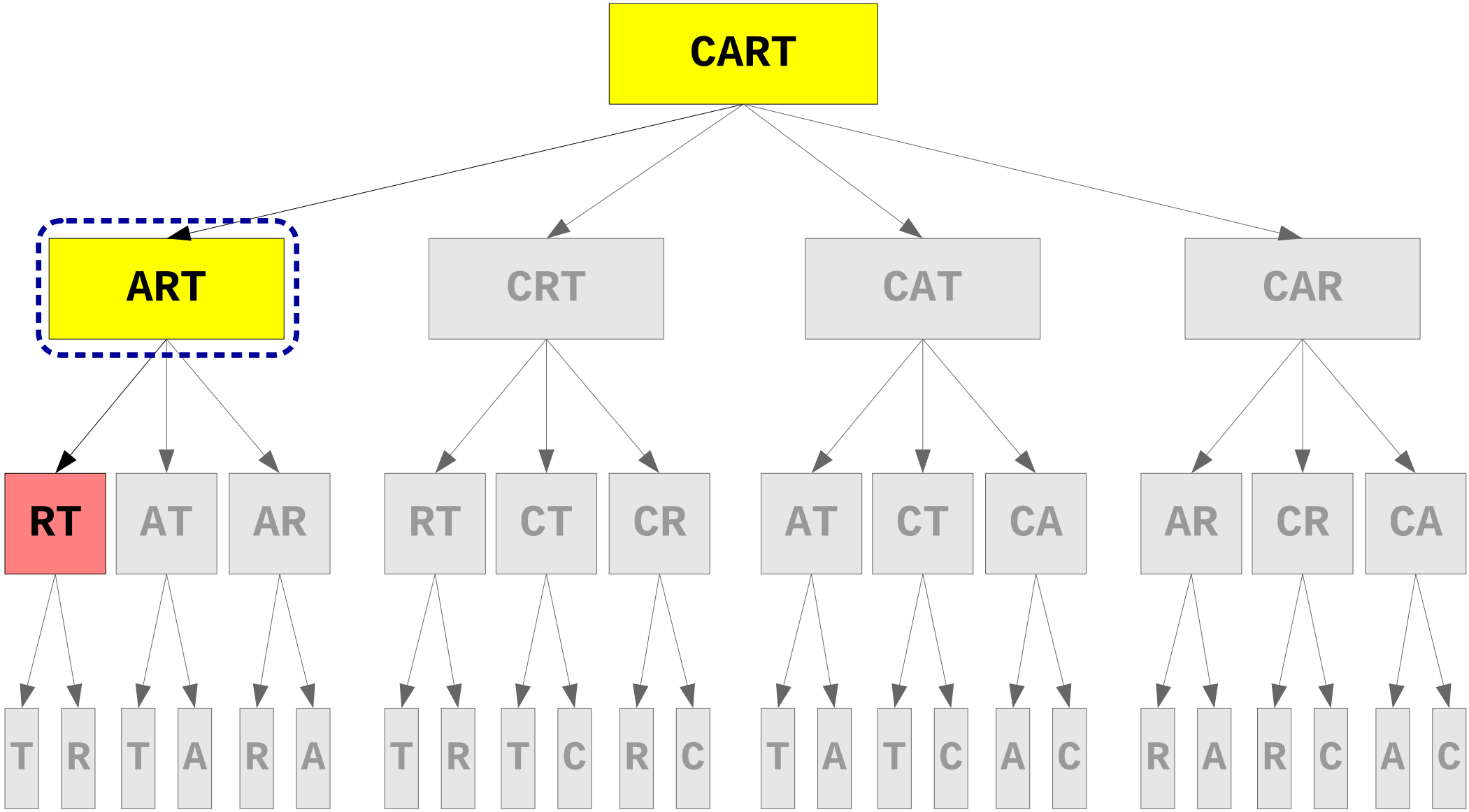
Generating the Answer



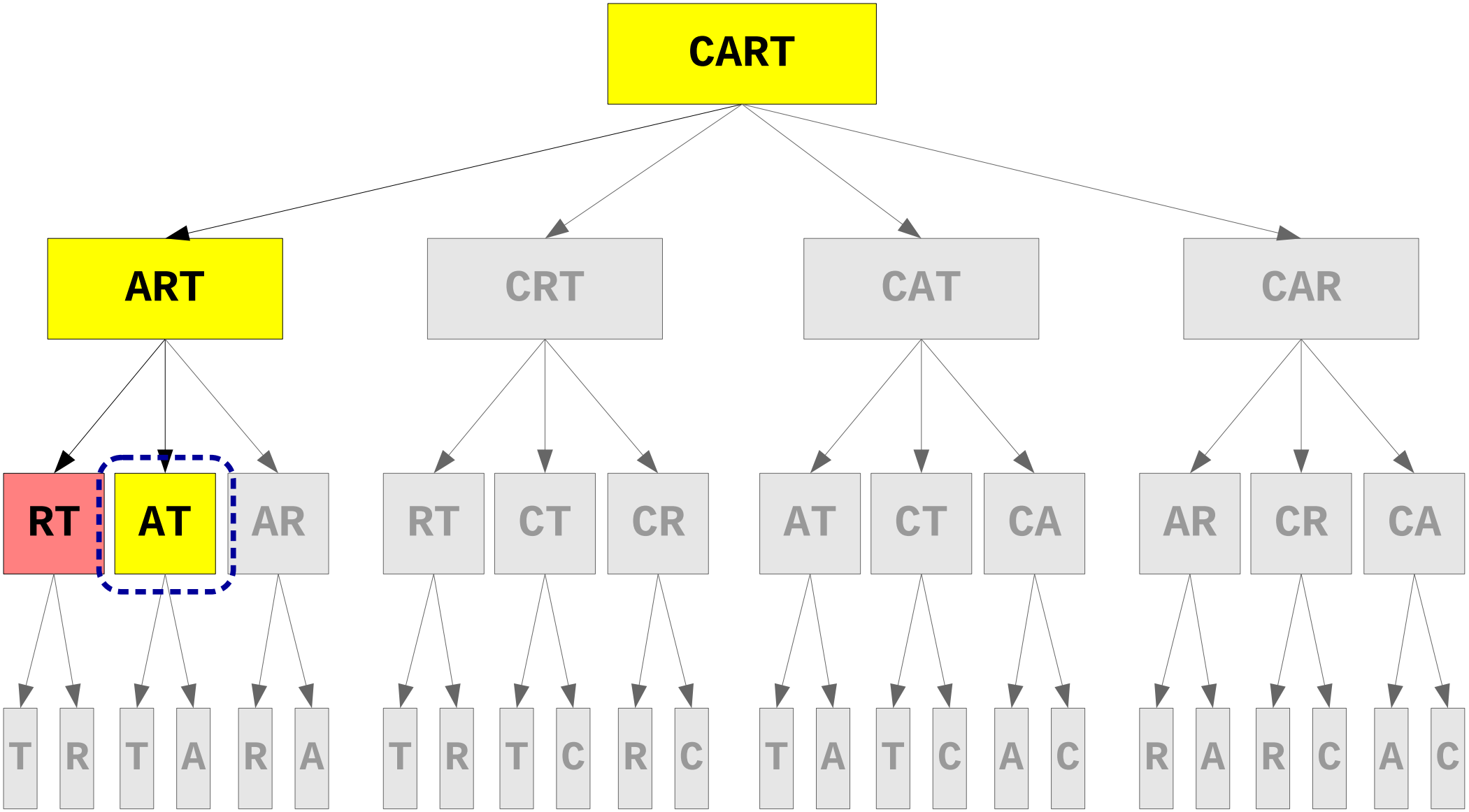
Generating the Answer



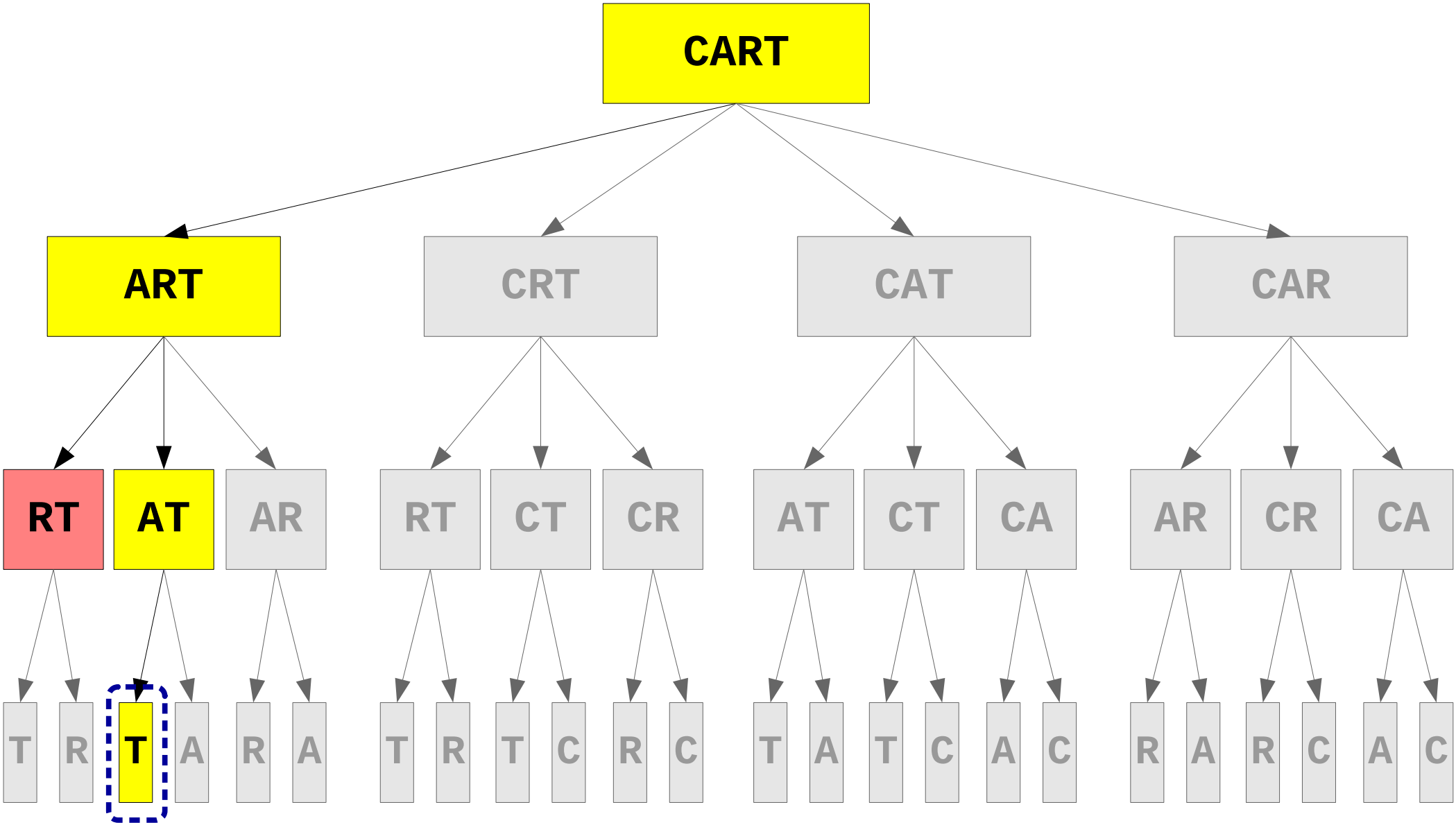
Generating the Answer



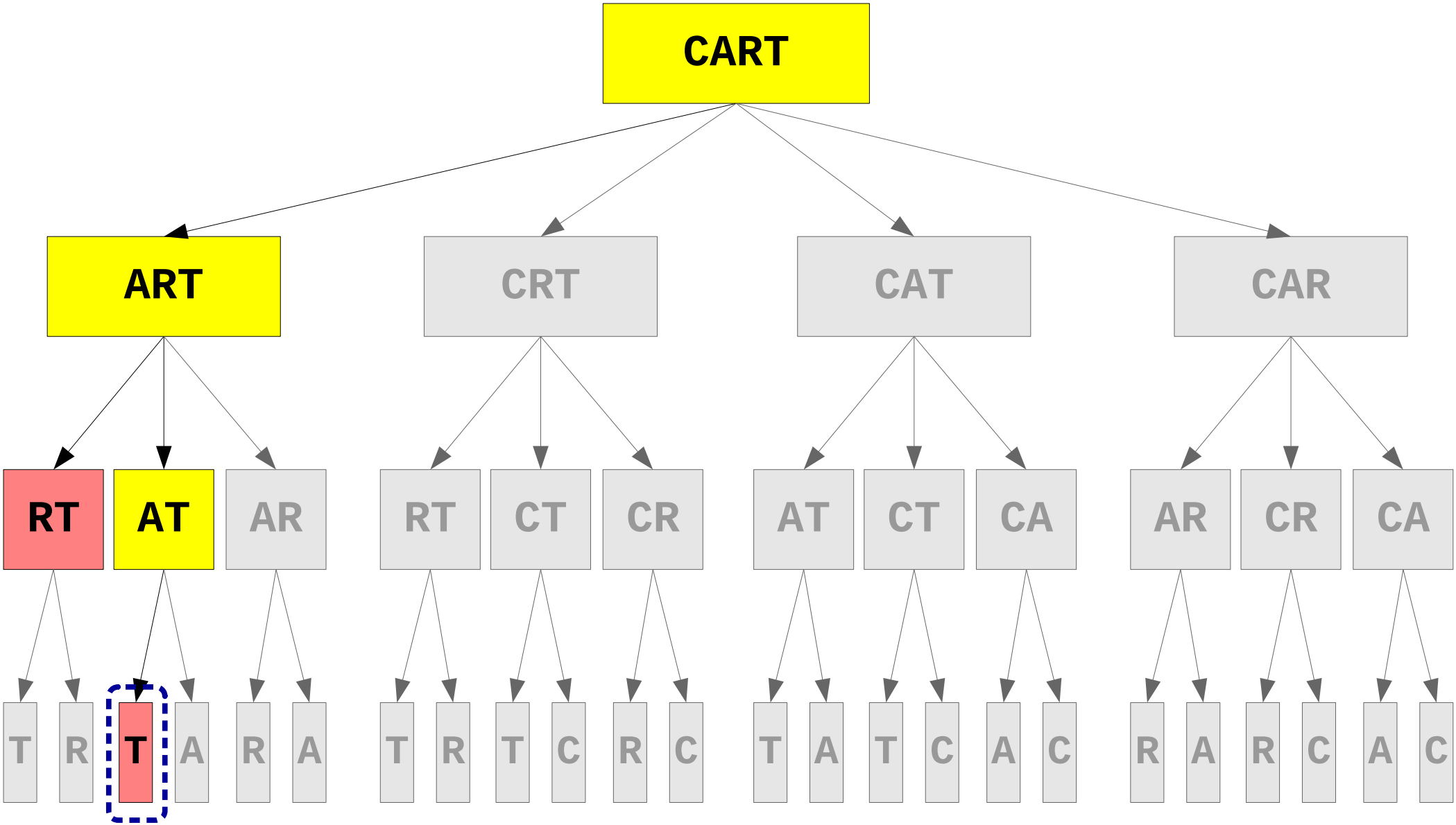
Generating the Answer



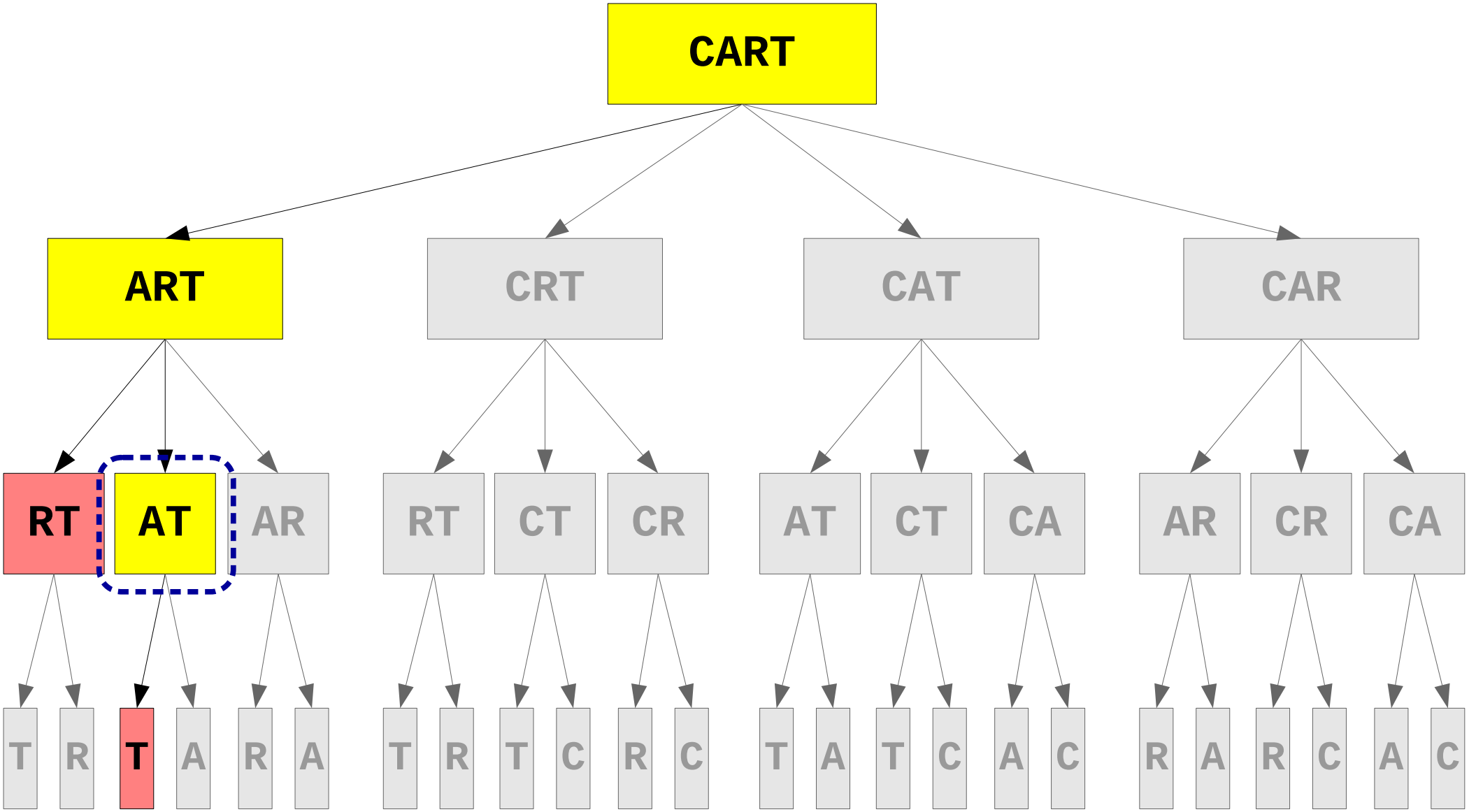
Generating the Answer



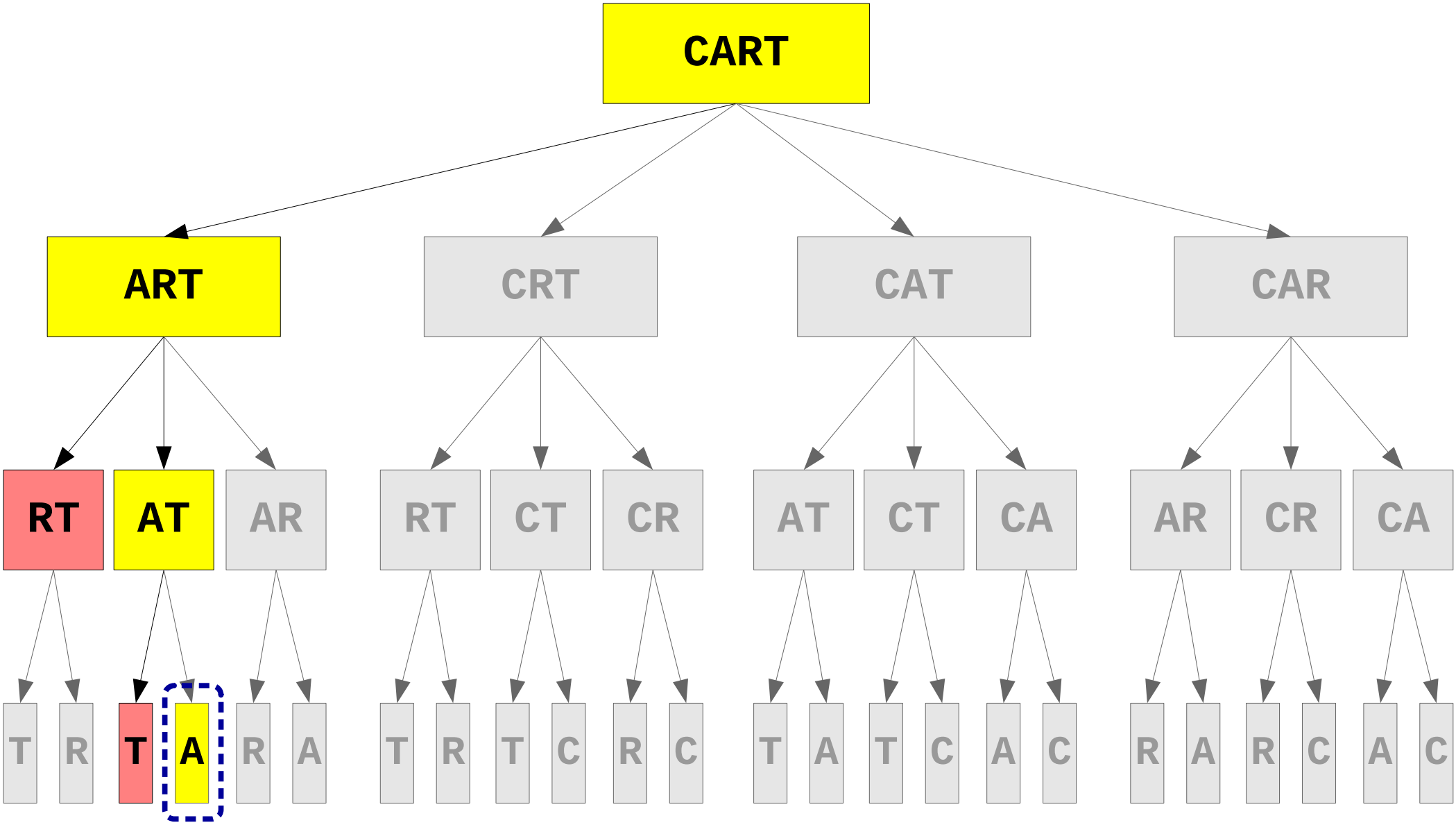
Generating the Answer



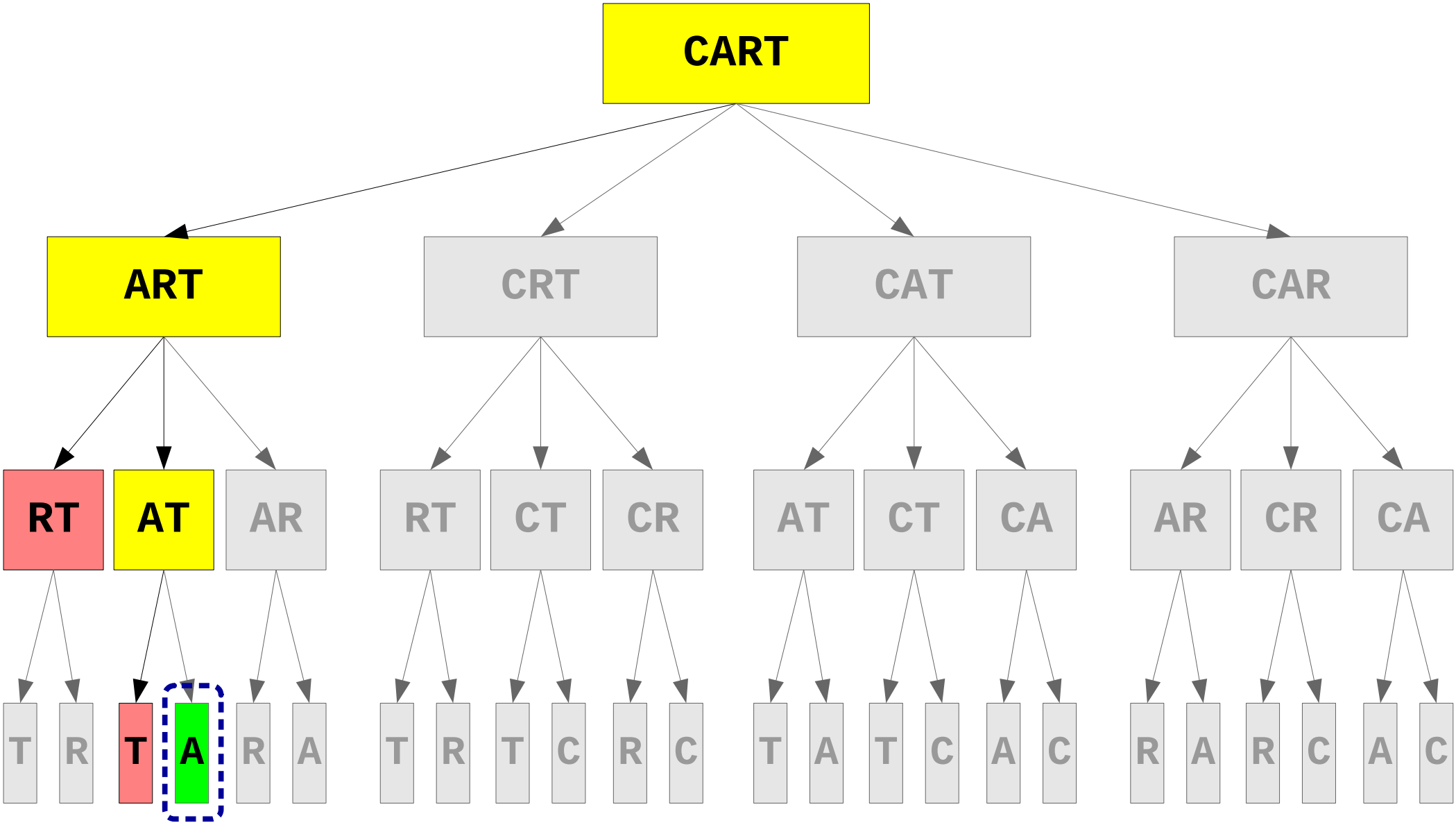
Generating the Answer



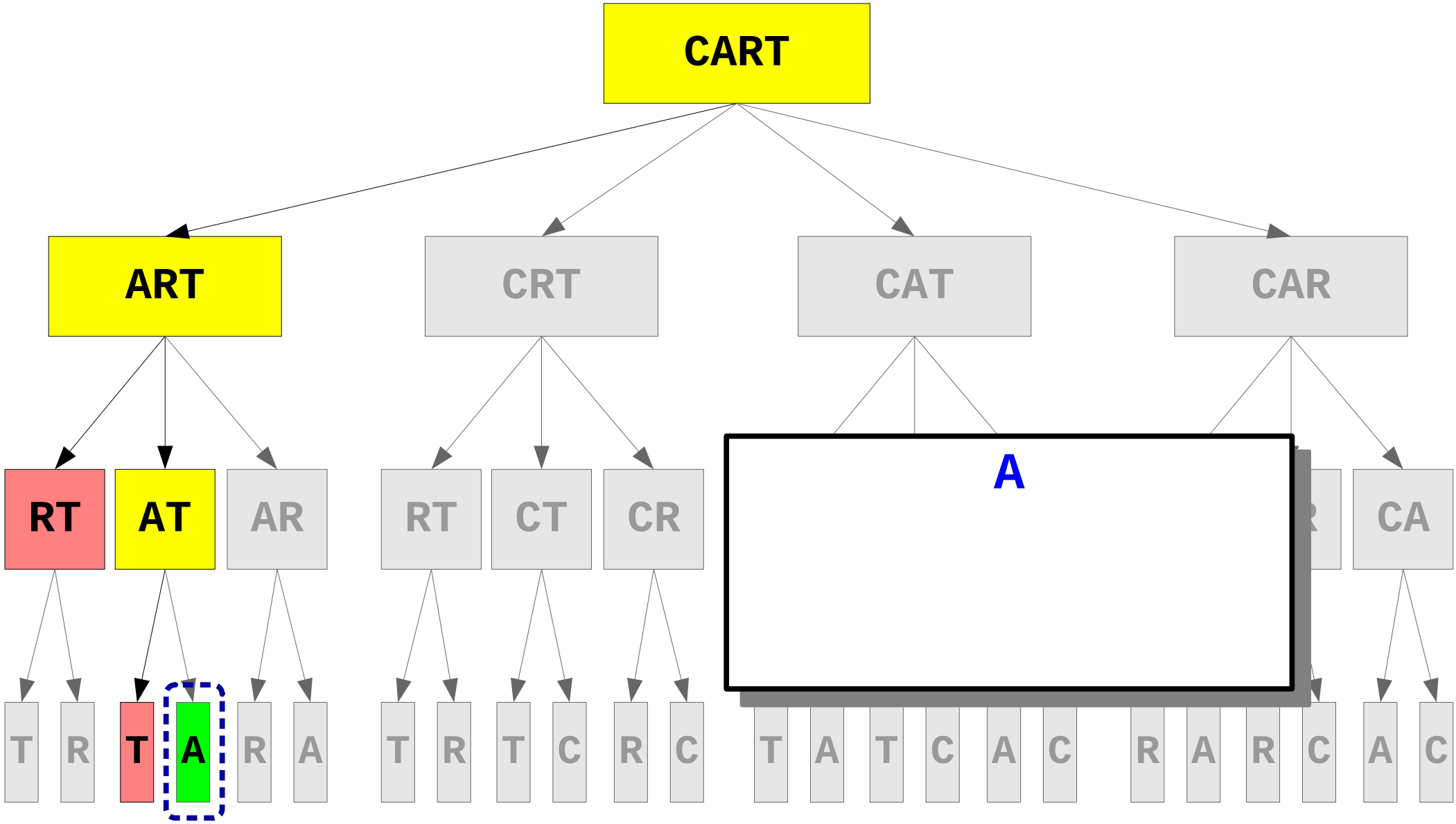
Generating the Answer



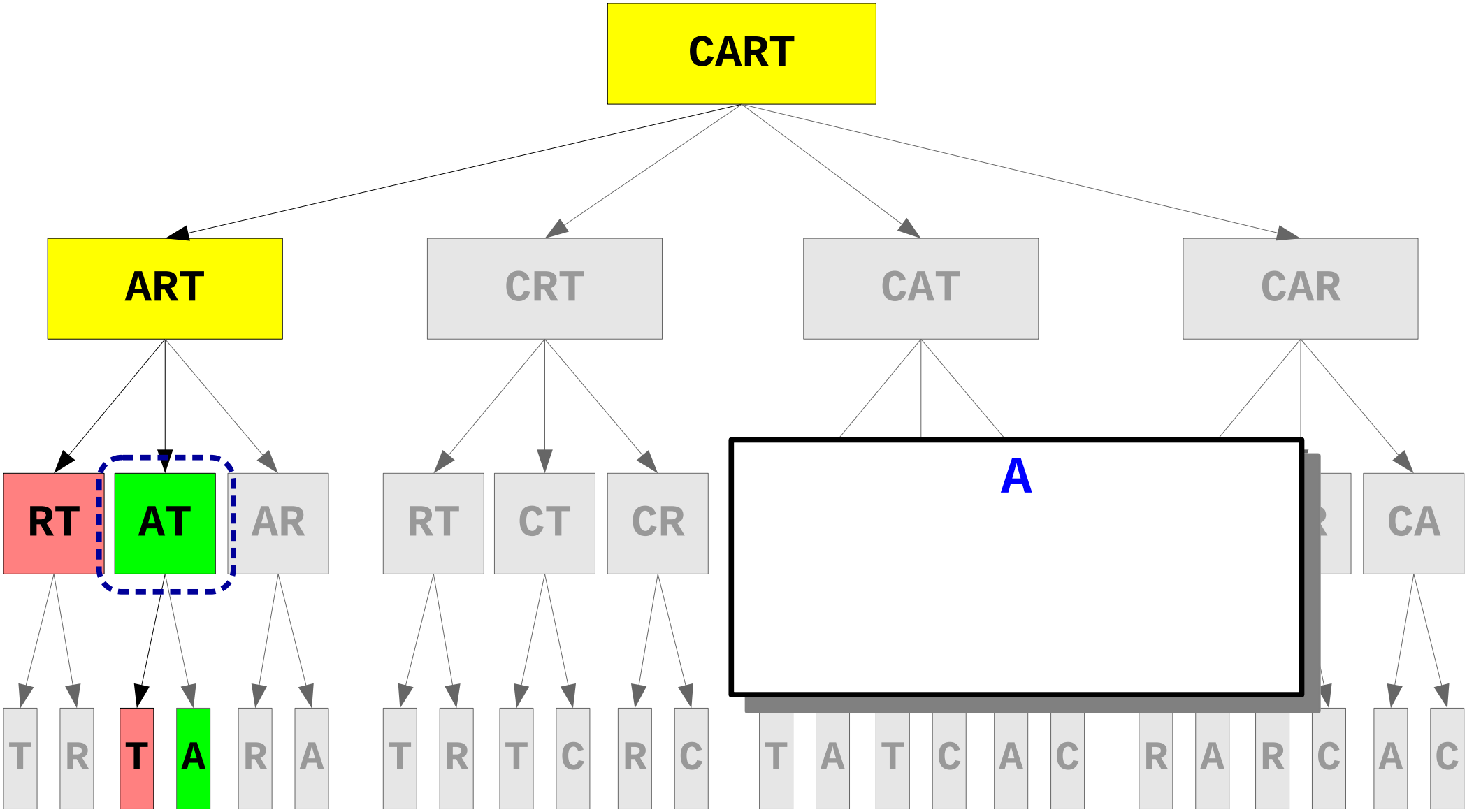
Generating the Answer



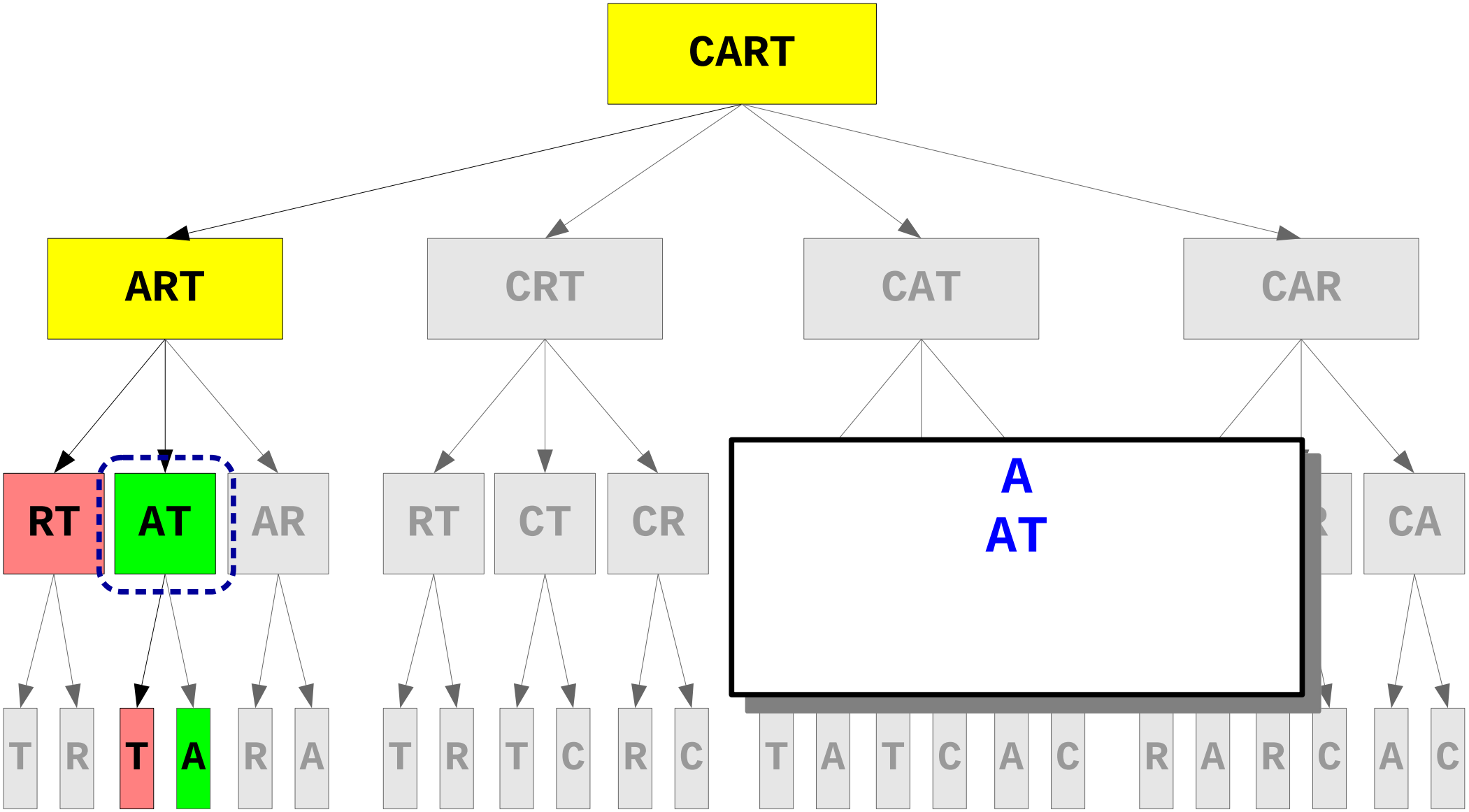
Generating the Answer



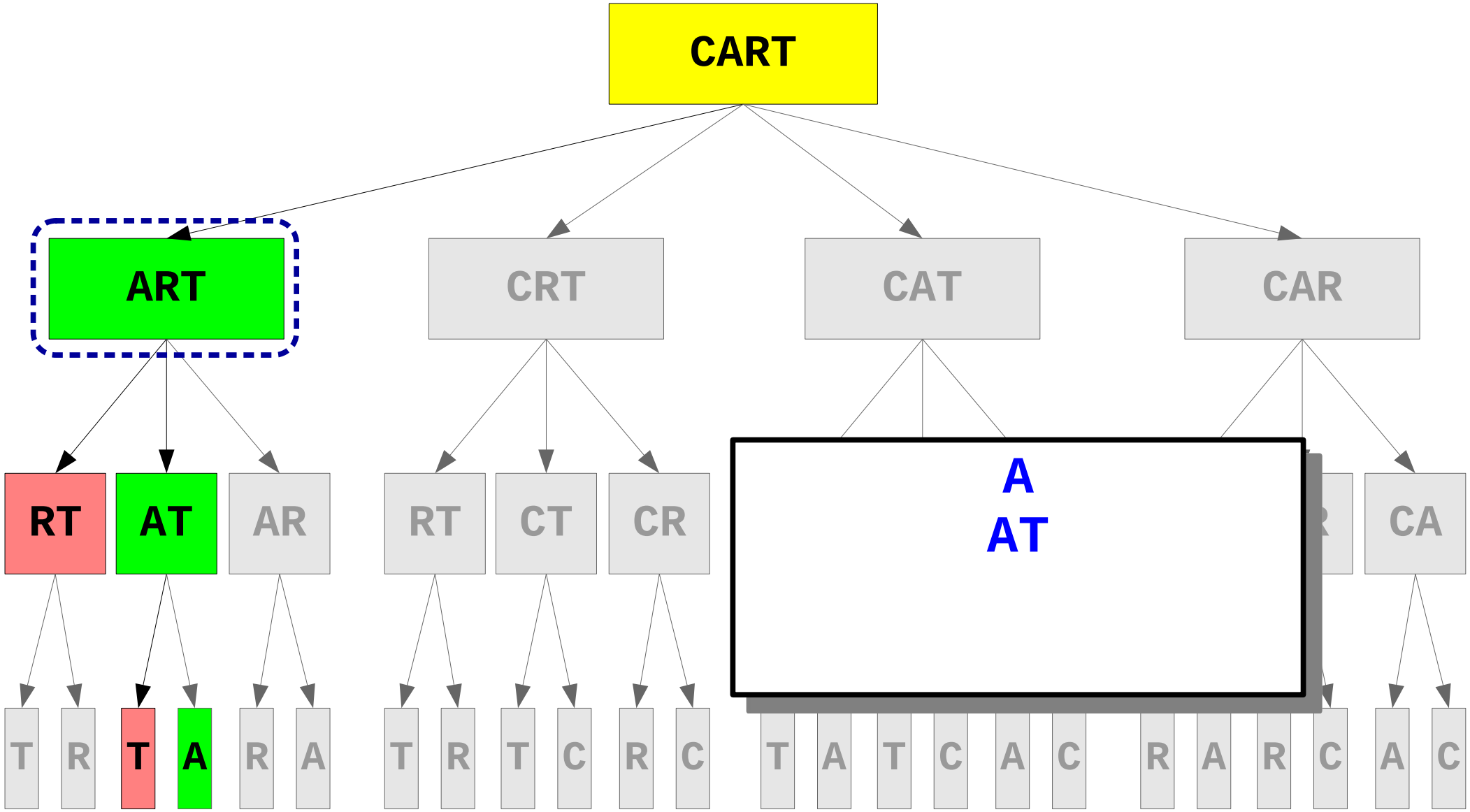
Generating the Answer



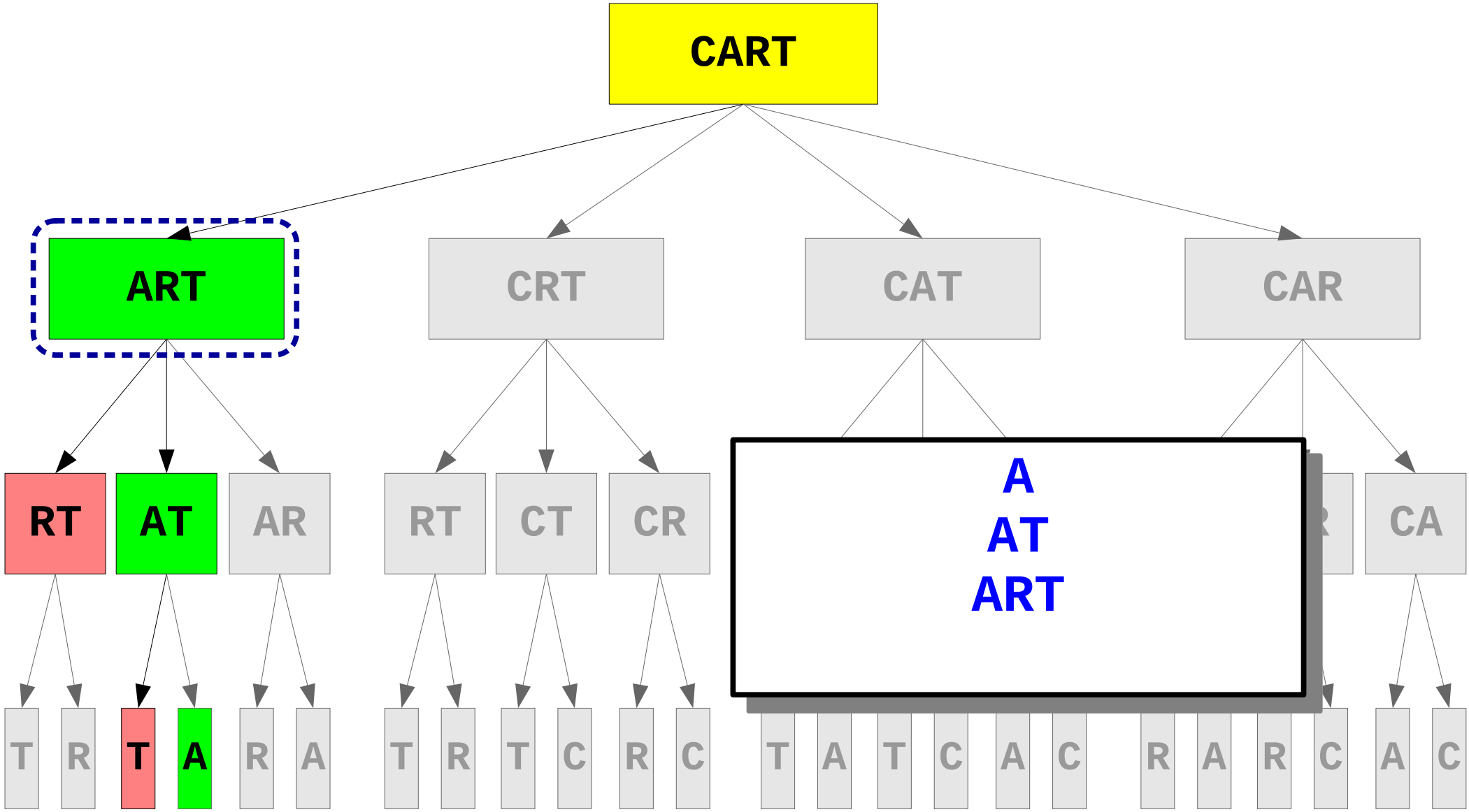
Generating the Answer



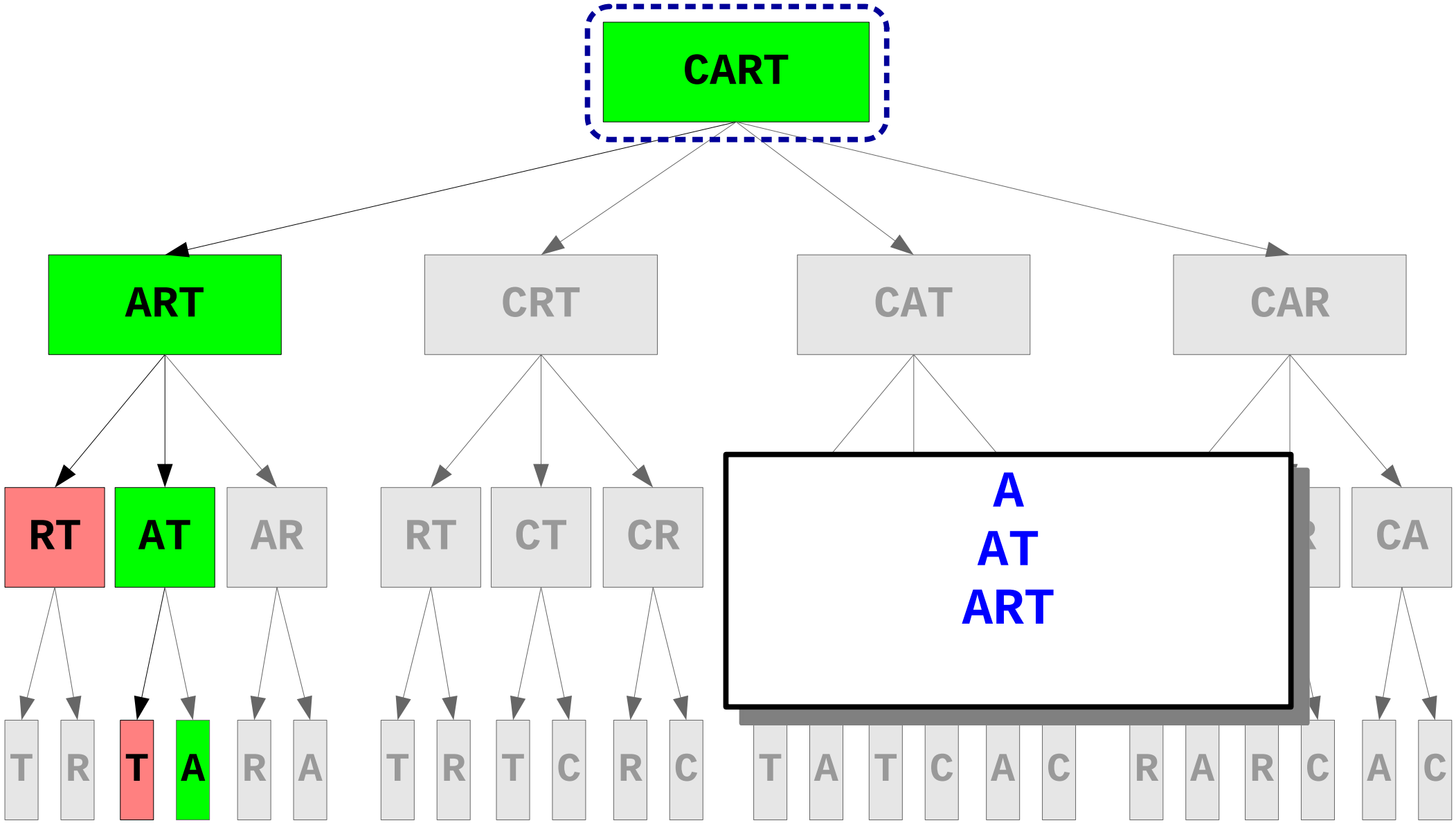
Generating the Answer



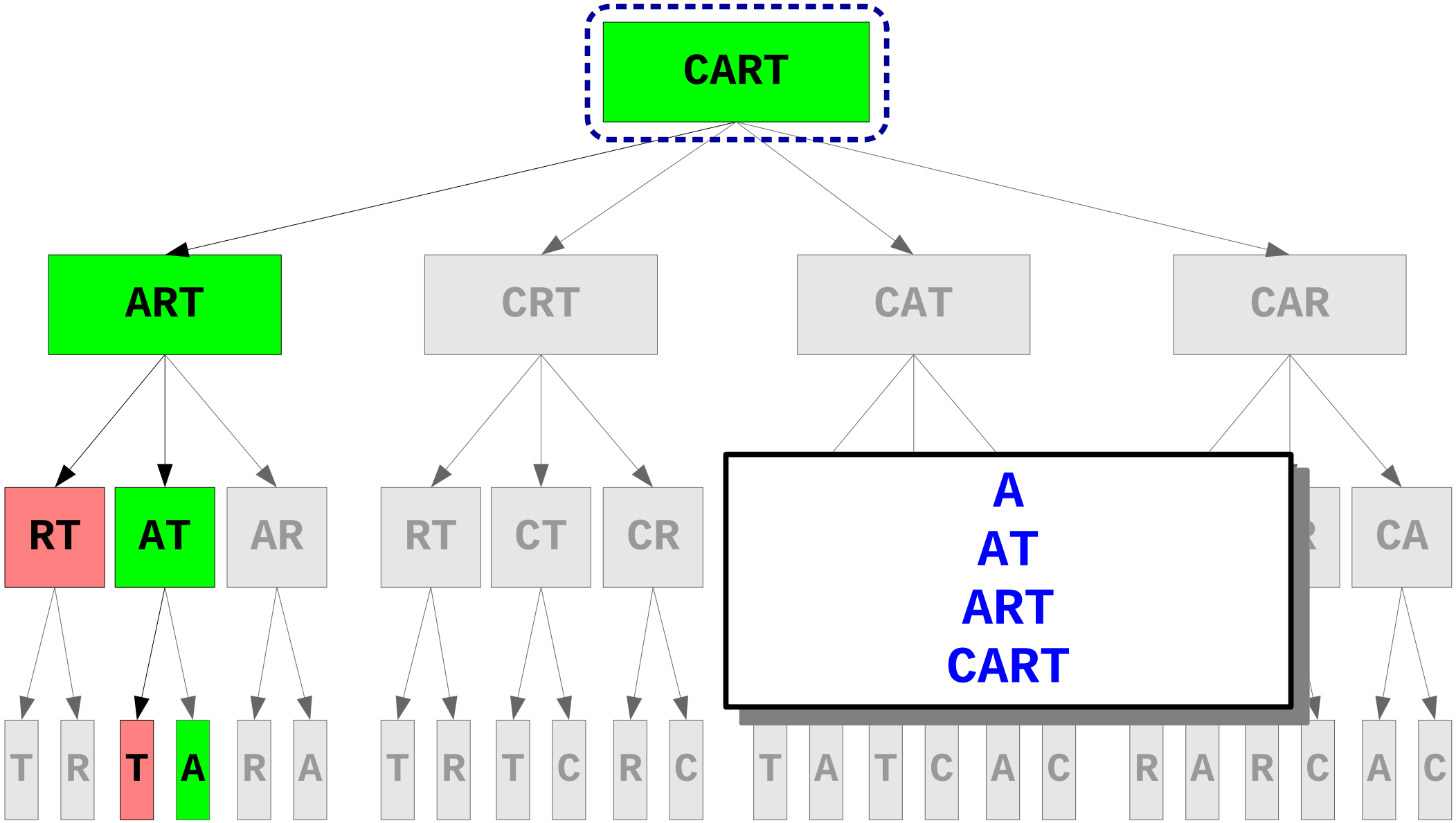
Generating the Answer



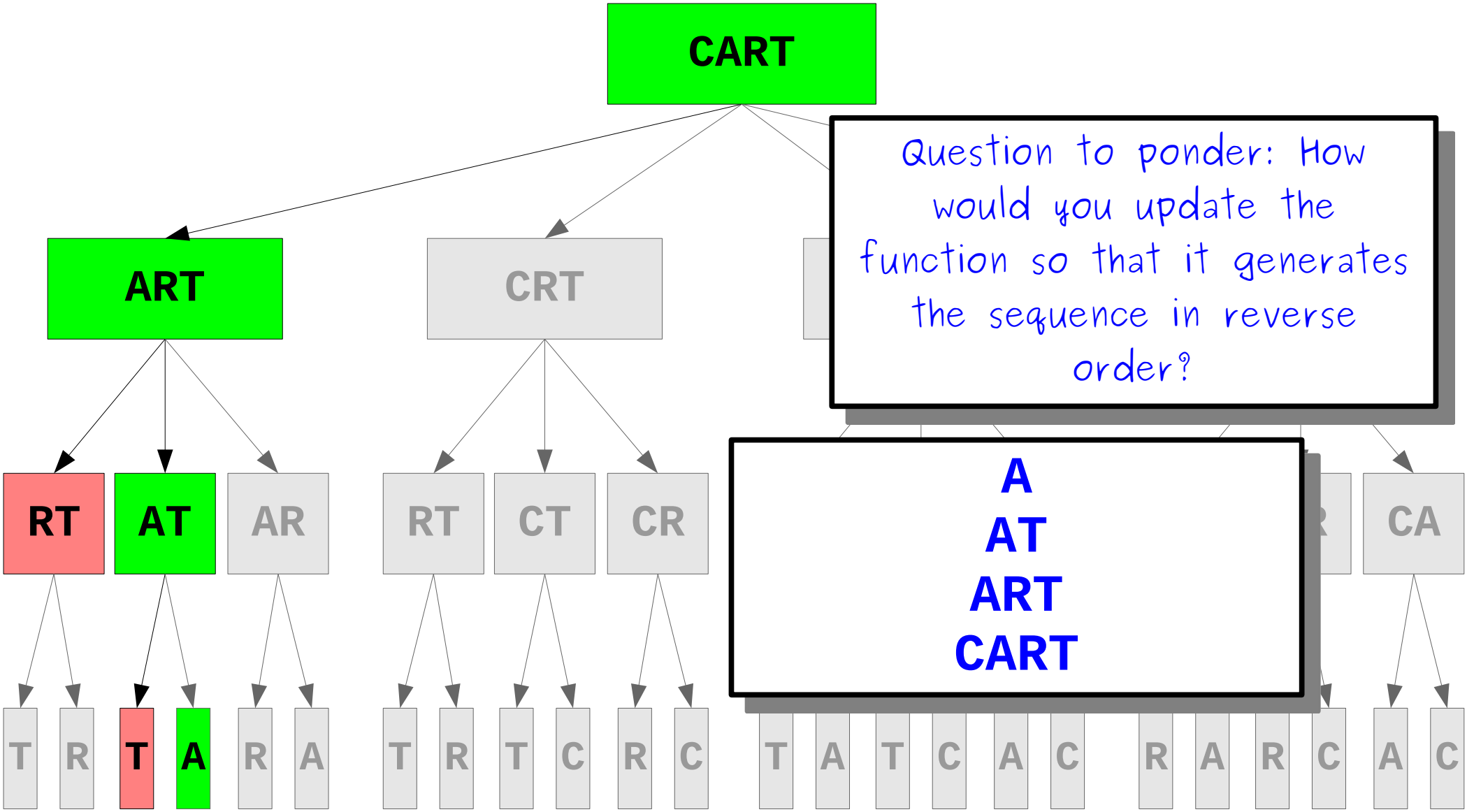
Generating the Answer



Generating the Answer



Generating the Answer



Question to ponder: How would you update the function so that it generates the sequence in reverse order?

A
AT
ART
CART

Time-Out for Announcements!

Assignment 4

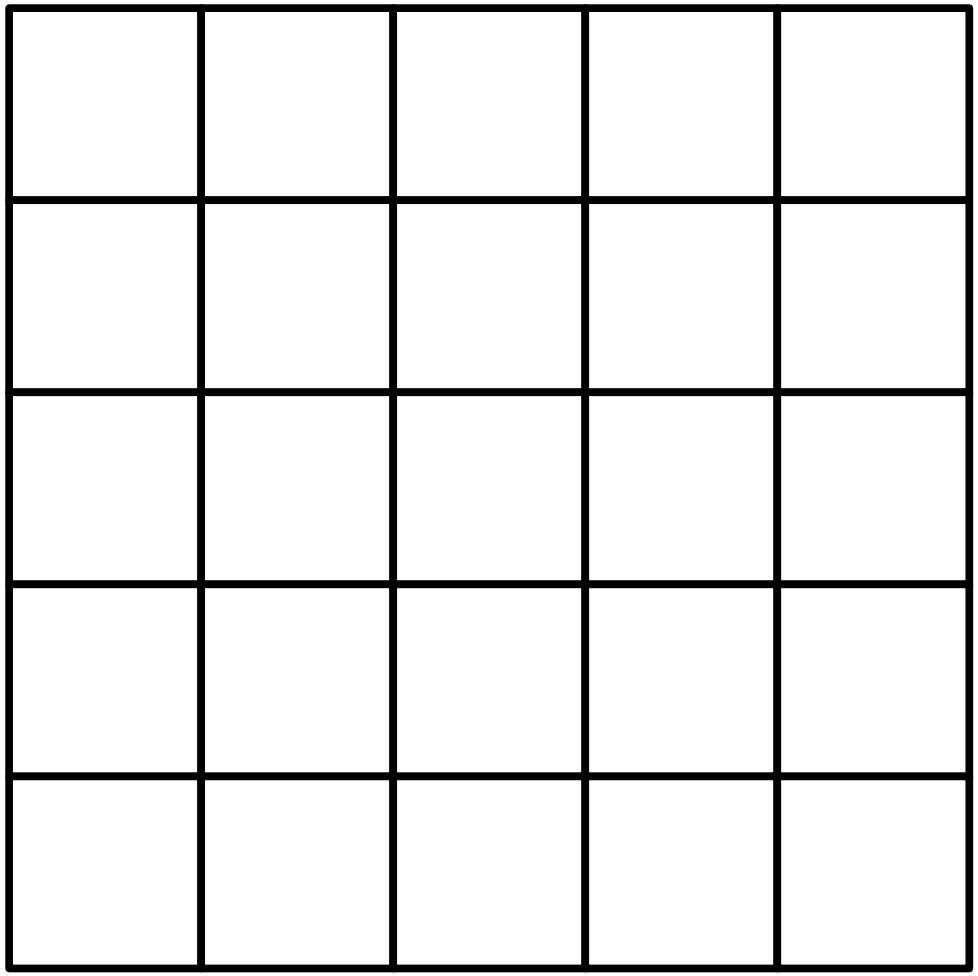
- Assignment 3 was due at 10:30AM today.
  - Need more time? You can submit during the grace period up until tomorrow morning at 10:30AM.
- Assignment 4 (***Recursion to the Rescue!***) goes out today. It's due next Friday at 10:30AM.
  - Play around with backtracking searches!
  - Get a feel for just how powerful recursion is!
- You are welcome to work on this one with a partner.
- We've provided a suggested timetable at the top of the assignment.

Midterm Exam Logistics

- Our first midterm exam runs from 10:30AM Friday, February 4th to 10:30AM Sunday, February 6th, Pacific time.
- It's a take-home coding exam. It will be designed to take three hours to complete, though you'll have the full 48-hour window to complete it.
- We'll post starter files and instructions along the lines of what we've done for the programming assignments.
- Topic coverage is as follows:
  - Primary focus is on material and concepts from Lectures 00 – 10 and Assignments 0 – 3.
  - Secondary focus is on material and concepts from Lectures 11 – 13 and Assignment 4.
- More information can be found online in the "Midterm Information" page on the course website. We'll talk more about the exam over the next week in lecture, too!

Back to CS106B!

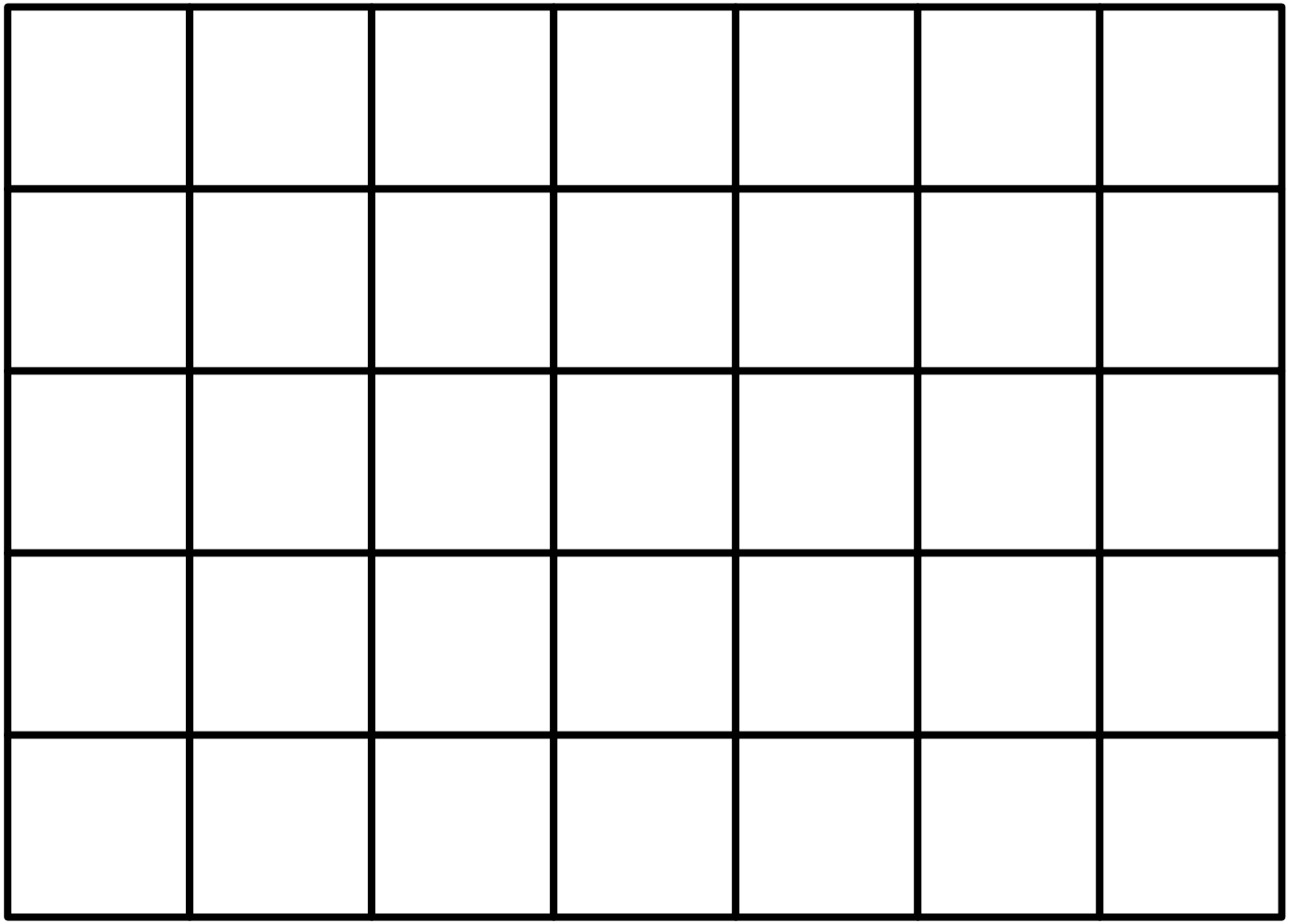
Dense Crosswords



New York Times Mini Crossword, January 24, 2020

s	t	r	a	w
i	r	i	s	h
m	a	s	s	e
o	c	e	a	n
n	e	r	d	s

New York Times Mini Crossword, January 24, 2020



Scoundrel

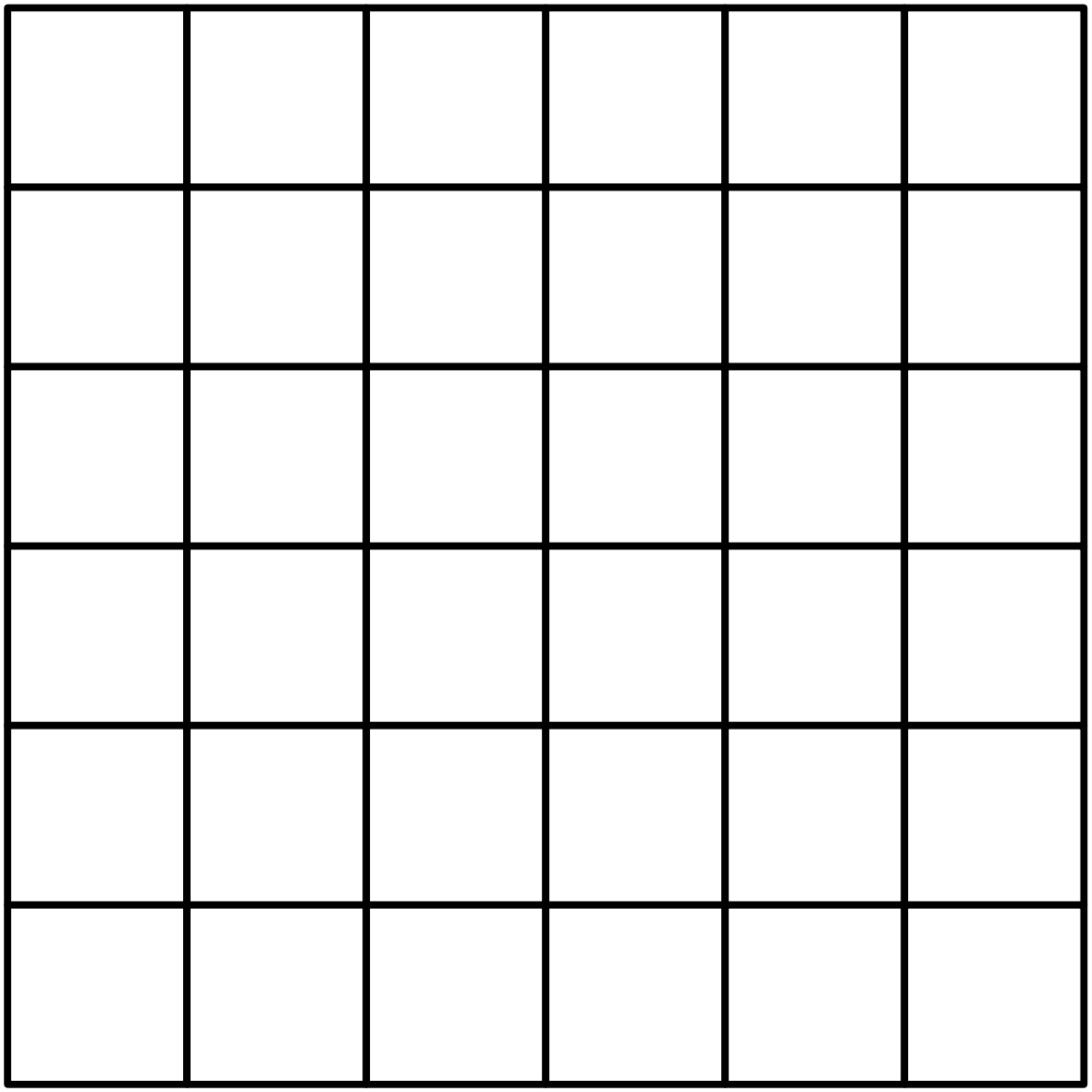
Where current flows in

Tapeworm

p	r	o	g	r	a	m
l	a	d	r	o	n	e
a	v	i	a	t	o	r
c	e	s	t	o	d	e
e	n	t	e	r	e	r

Person who writes odes

More than mere, less than merest



Rose-scented molecule

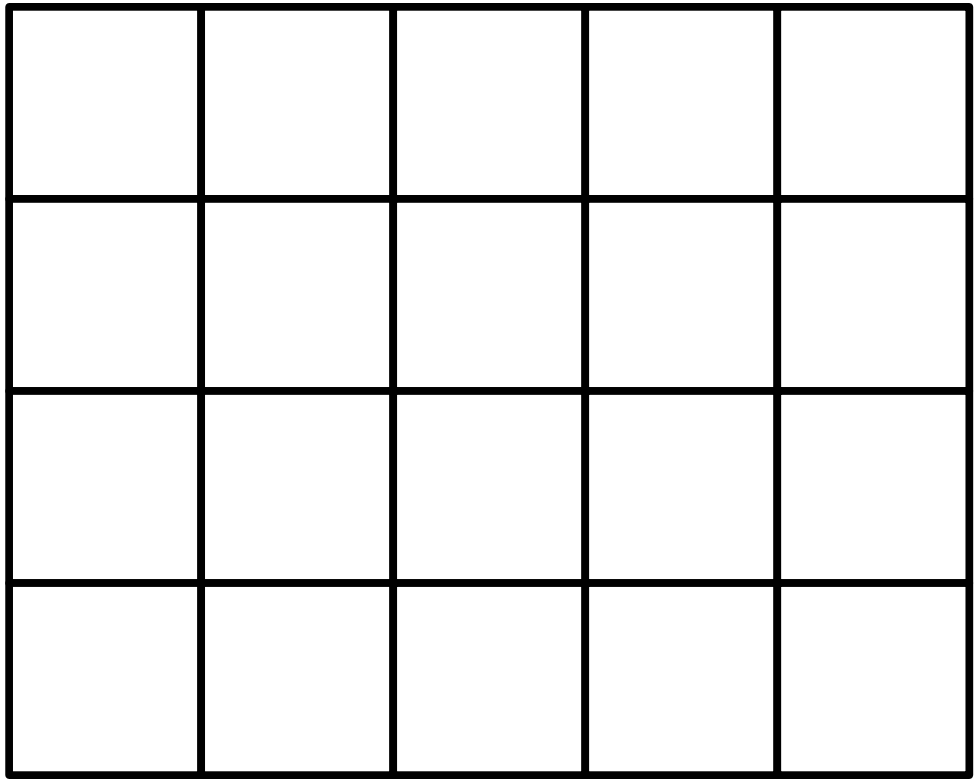
Stuffed grape leaves

Bind with lace

Hilly

Synonym for keyboard

d	i	k	d	i	k
i	o	n	o	n	e
k	n	o	l	l	y
d	o	l	m	a	s
i	n	l	a	c	e
k	e	y	s	e	t



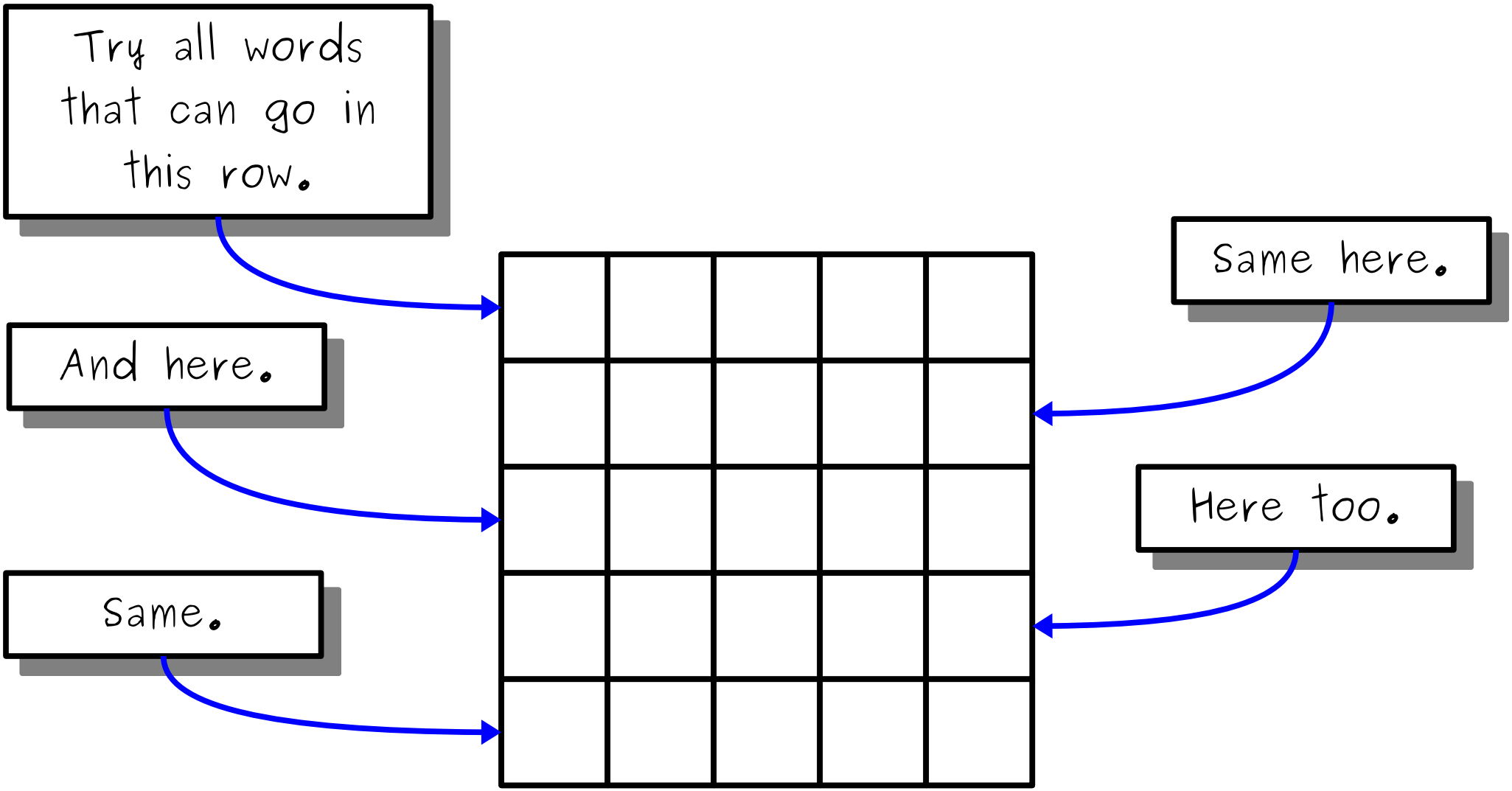
*Thanks for former CS106B student and CS106B SL **Jose Francisco**!*

s	p	l	i	t
e	r	o	d	e
a	e	r	o	s
s	p	e	l	t

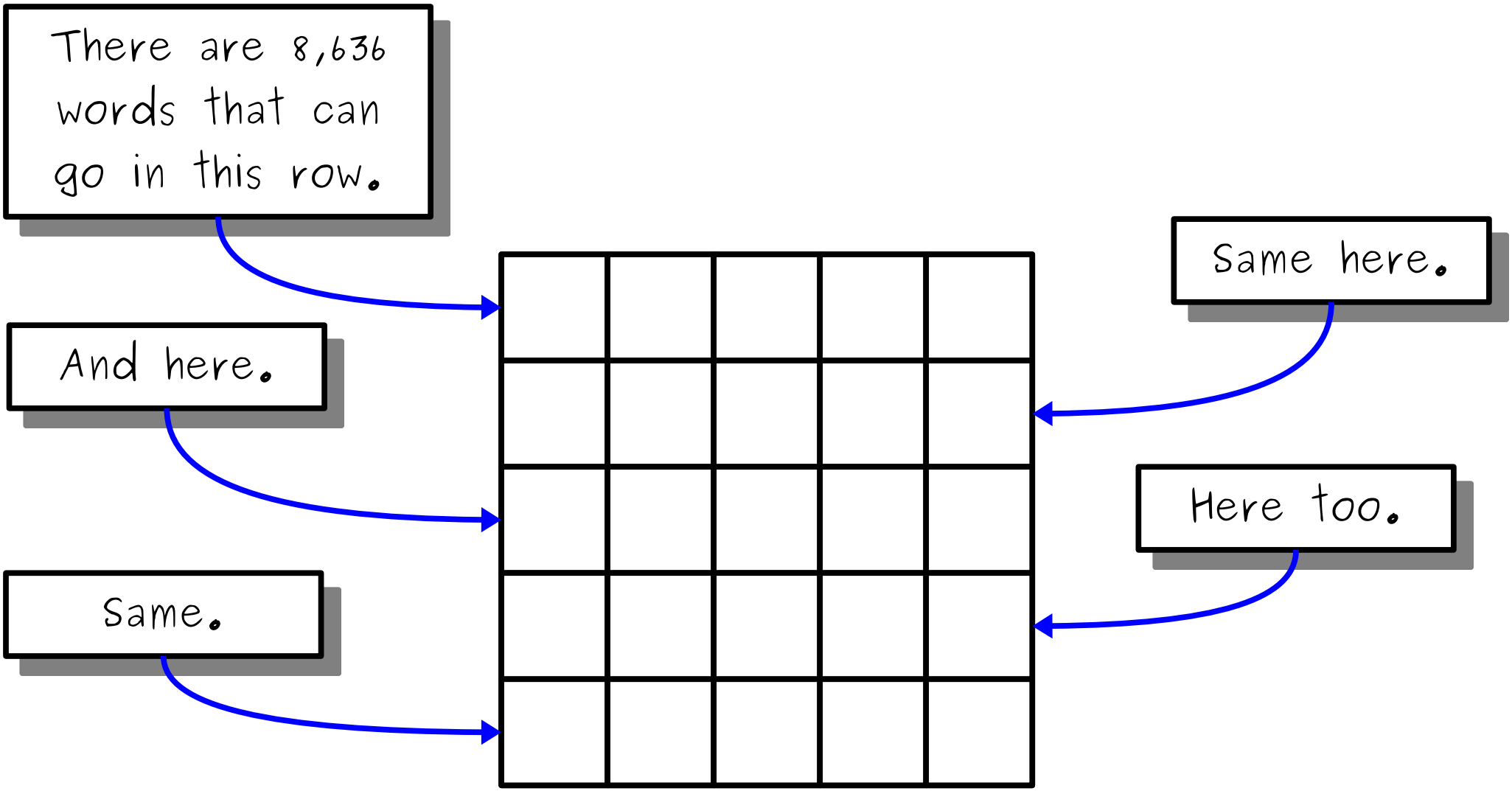
Short version of "aerodynamics"

Type of wheat

*Thanks for former CS106B student and CS106B SL **Jose Francisco**!*



Idea: Fill this in using recursive backtracking.

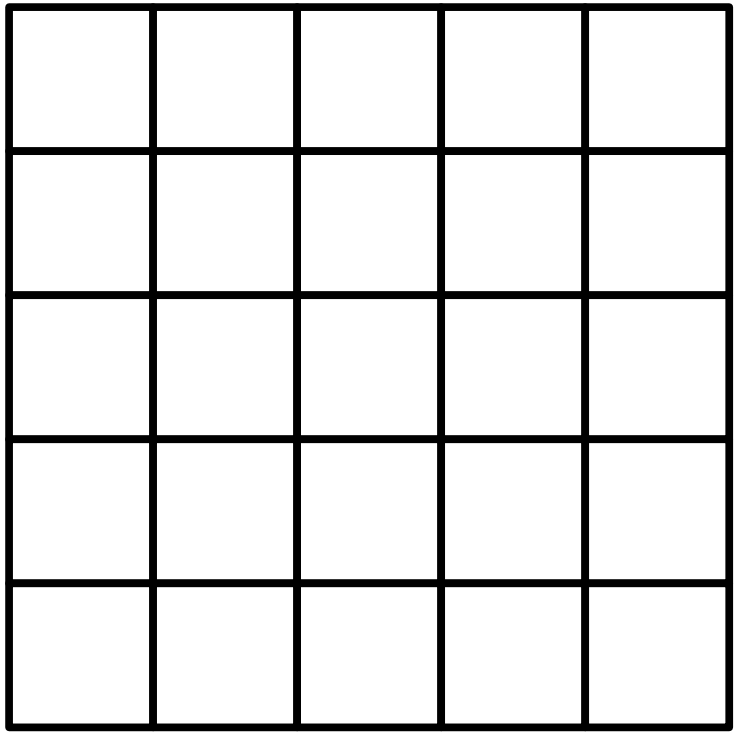


$$8{,}636^5 = 48{,}035{,}594{,}312{,}821{,}554{,}176$$

At one billion grids per second, this will take about ***three hundred years*** to complete.

Speeding Things Up

Generating Dense Crosswords



Generating Dense Crosswords

A	A	H	E	D

Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D

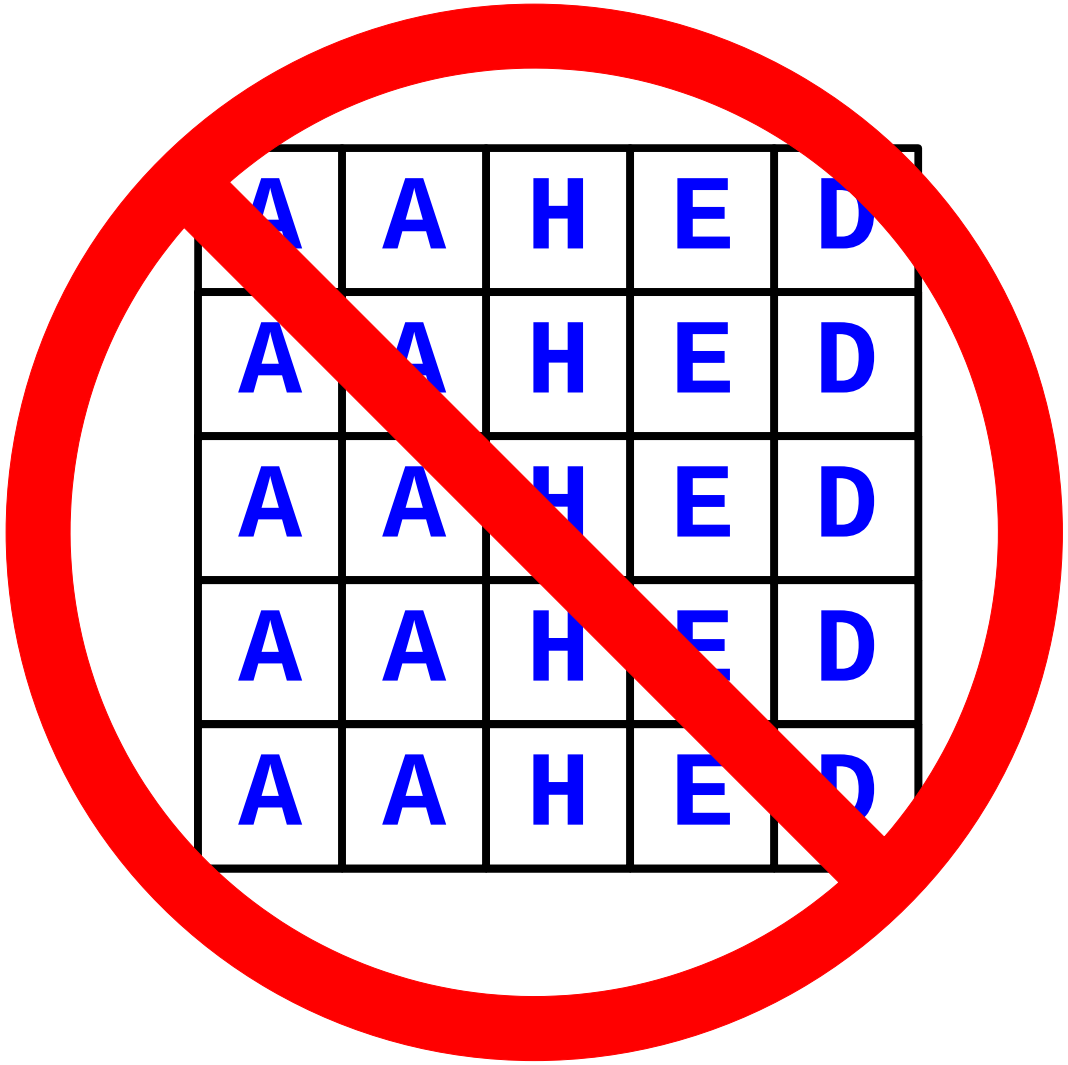
Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D

Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D

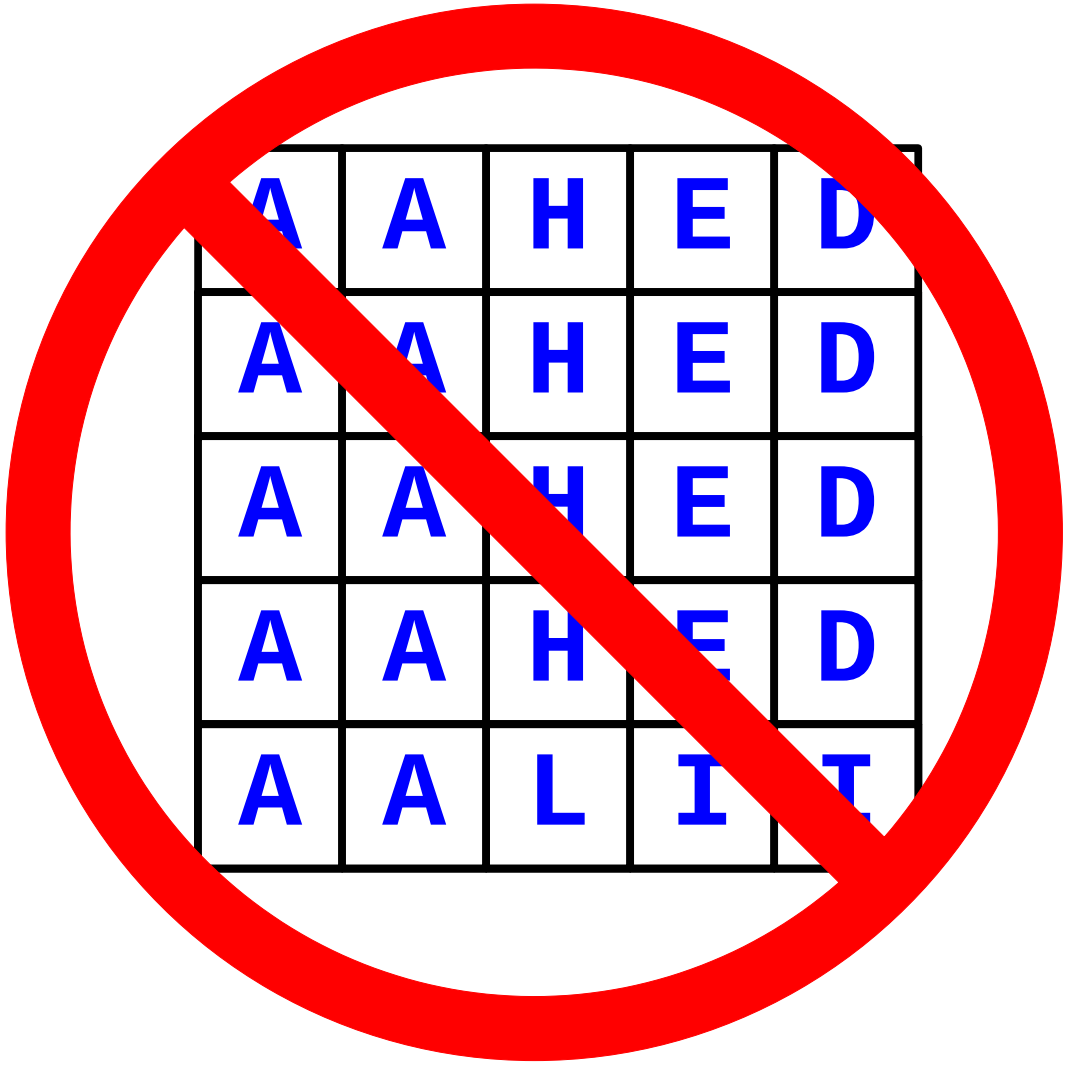
Generating Dense Crosswords



Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	L	I	I

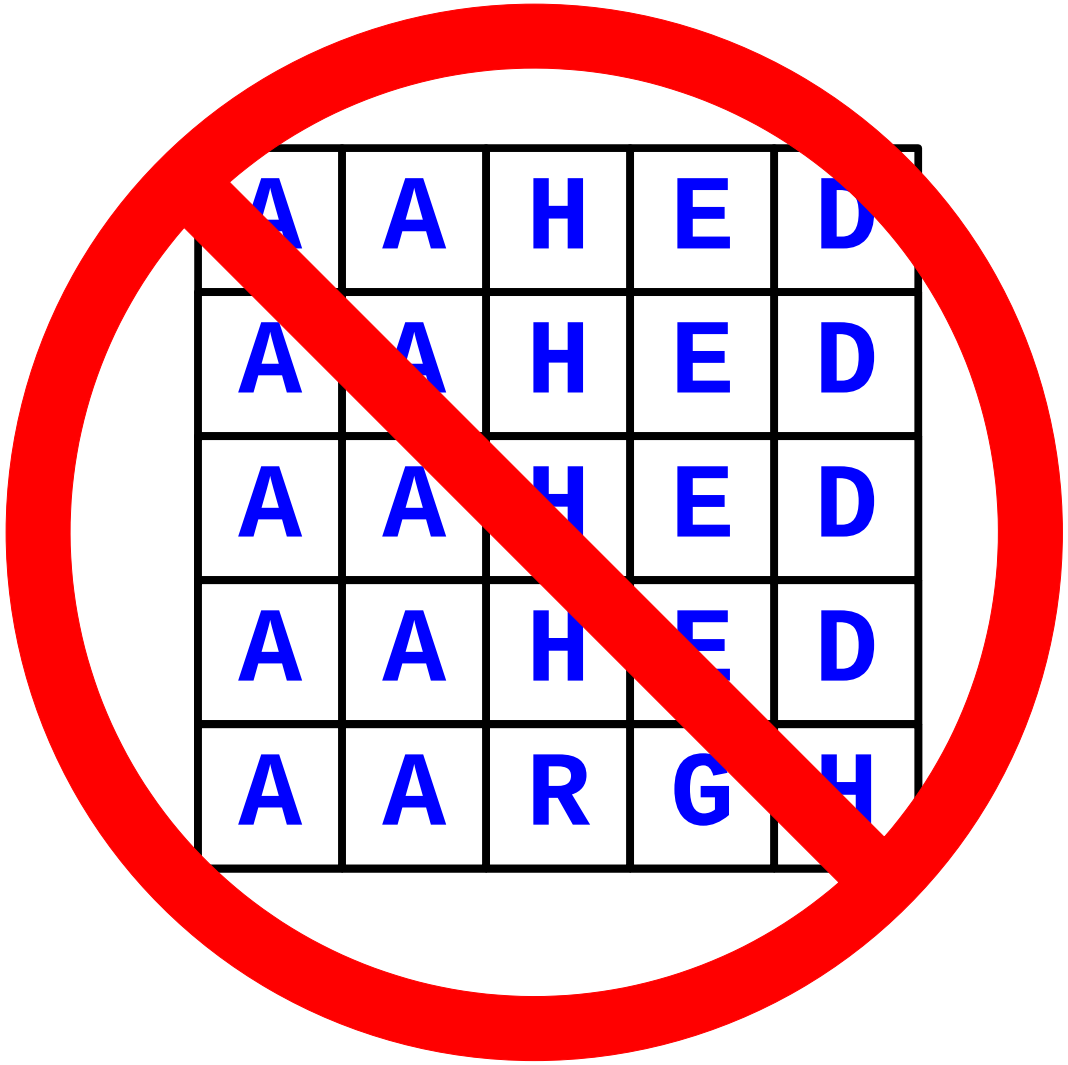
Generating Dense Crosswords



Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	R	G	H

Generating Dense Crosswords

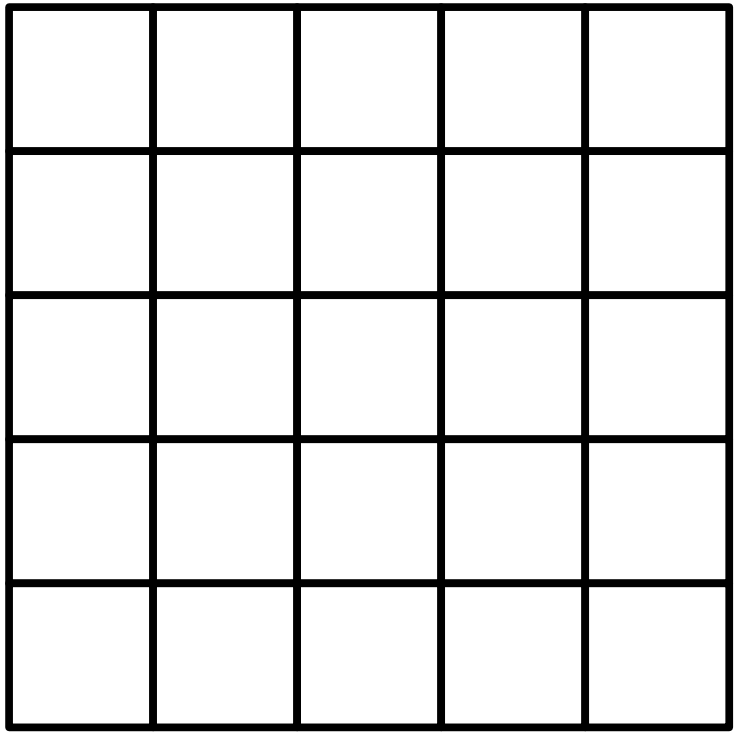


Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	R	G	H

These columns are silly. No words start with three A's, or three H's, etc.

Generating Dense Crosswords



Generating Dense Crosswords

A	A	H	E	D

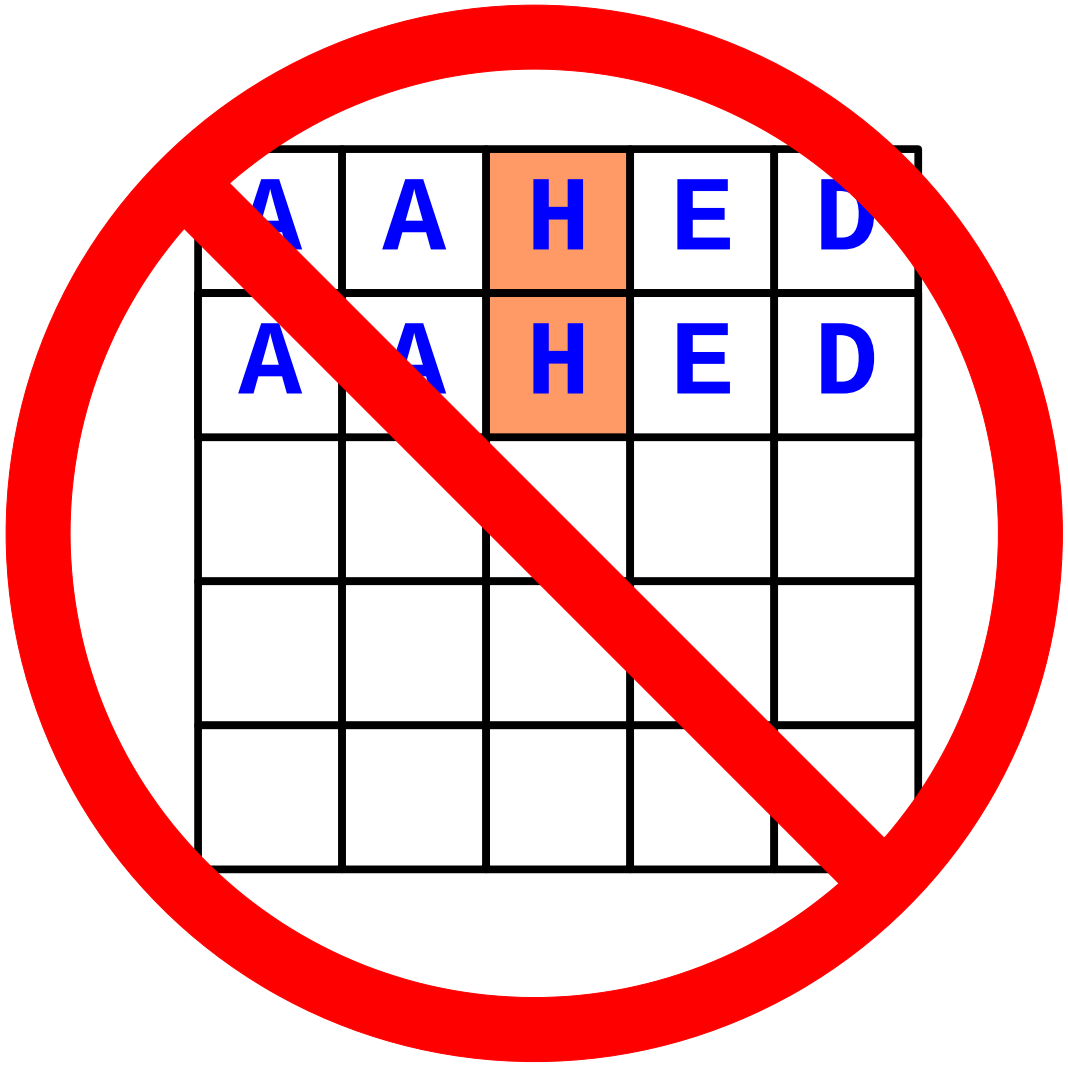
Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

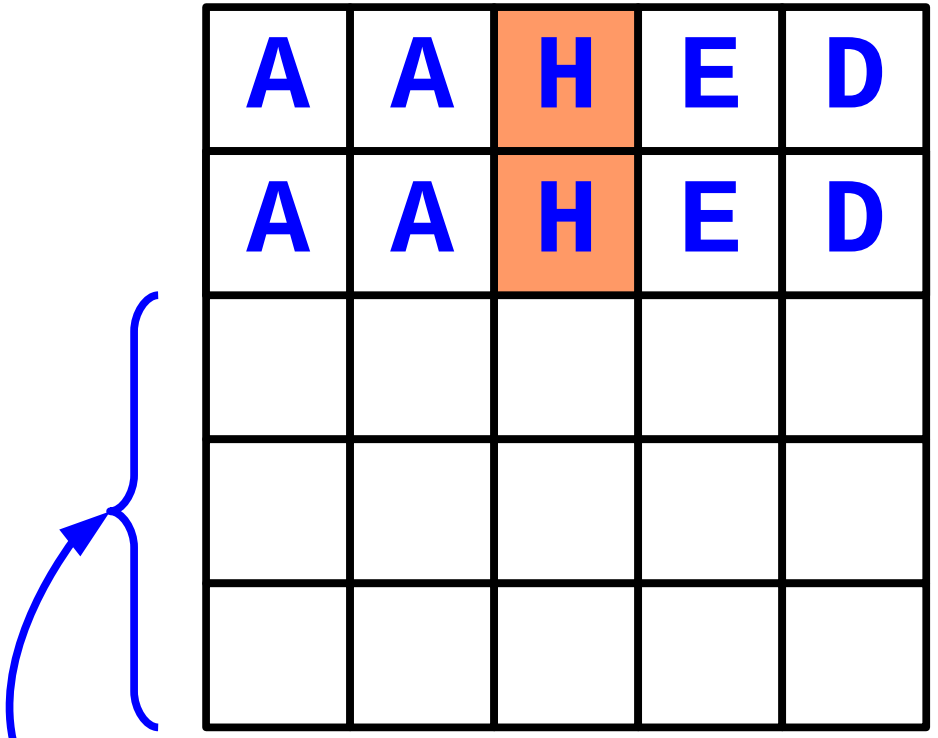
Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

Generating Dense Crosswords



Generating Dense Crosswords



We just skipped checking $8{,}636^3 = 644{,}077{,}163{,}456$ combinations of words.

Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

The `Lexicon` has a fast function `containsPrefix` that's perfect for this.

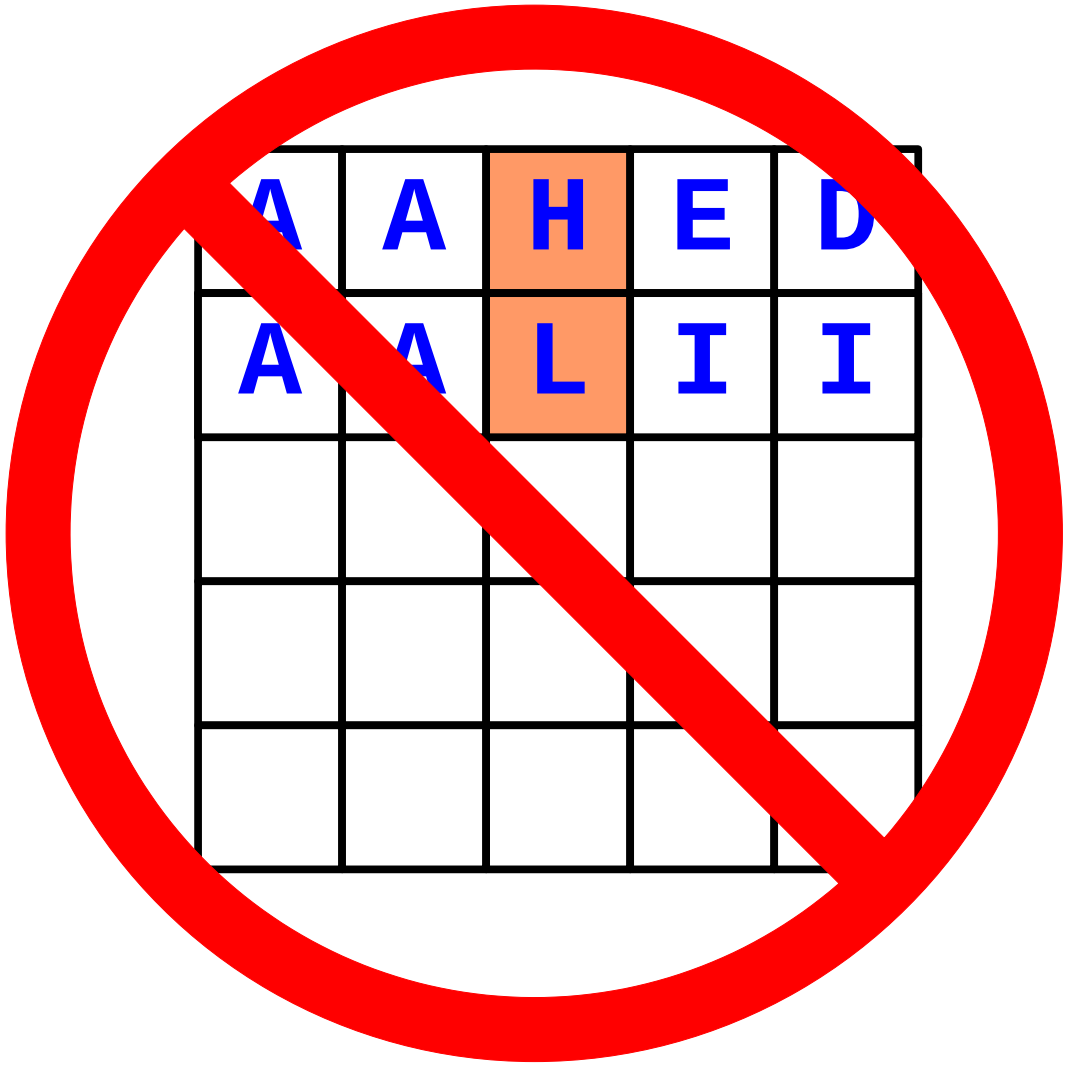
Generating Dense Crosswords

A	A	H	E	D
A	A	L	I	I

Generating Dense Crosswords

A	A	H	E	D
A	A	L	I	I

Generating Dense Crosswords



Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A

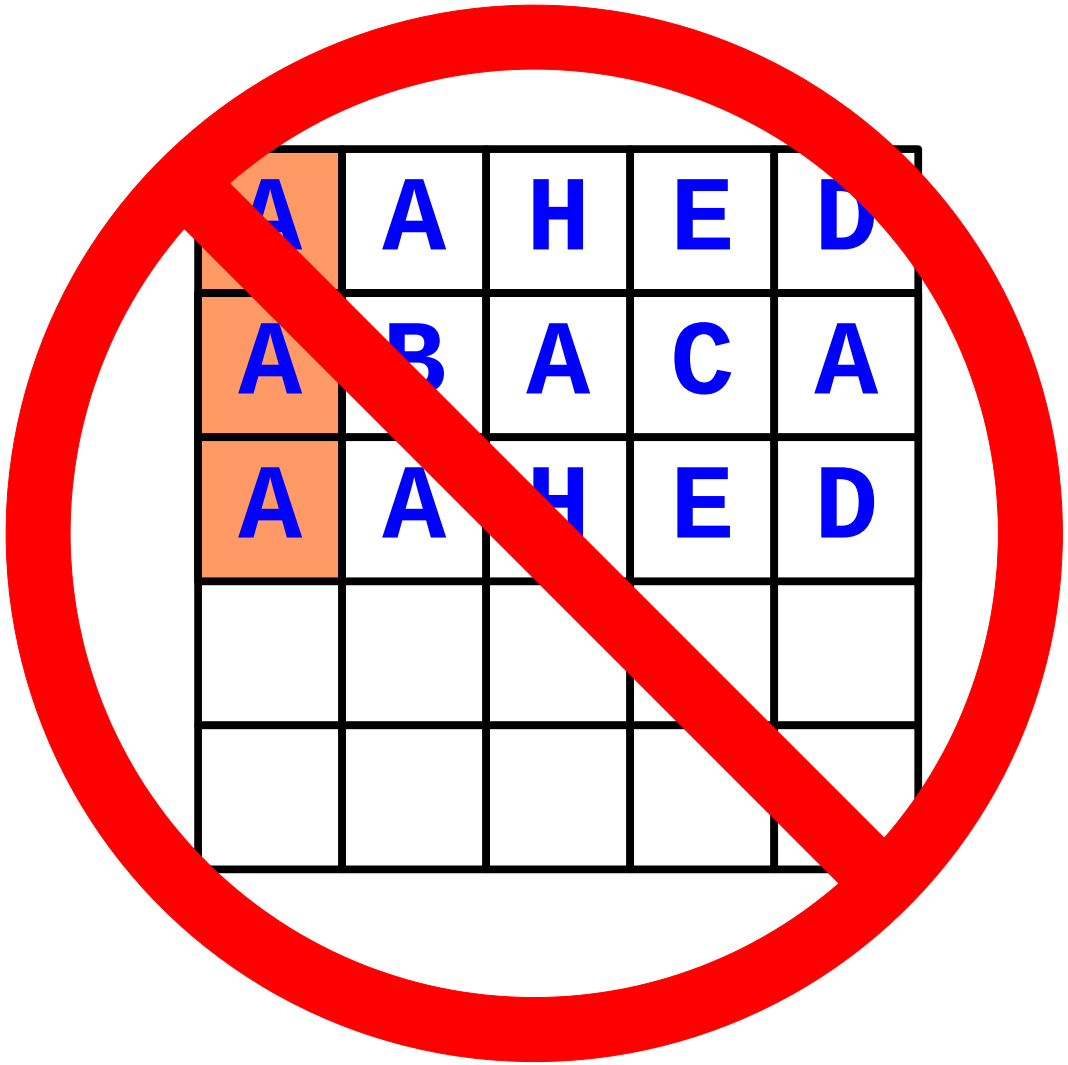
Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A
A	A	H	E	D

Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A
A	A	H	E	D

Generating Dense Crosswords



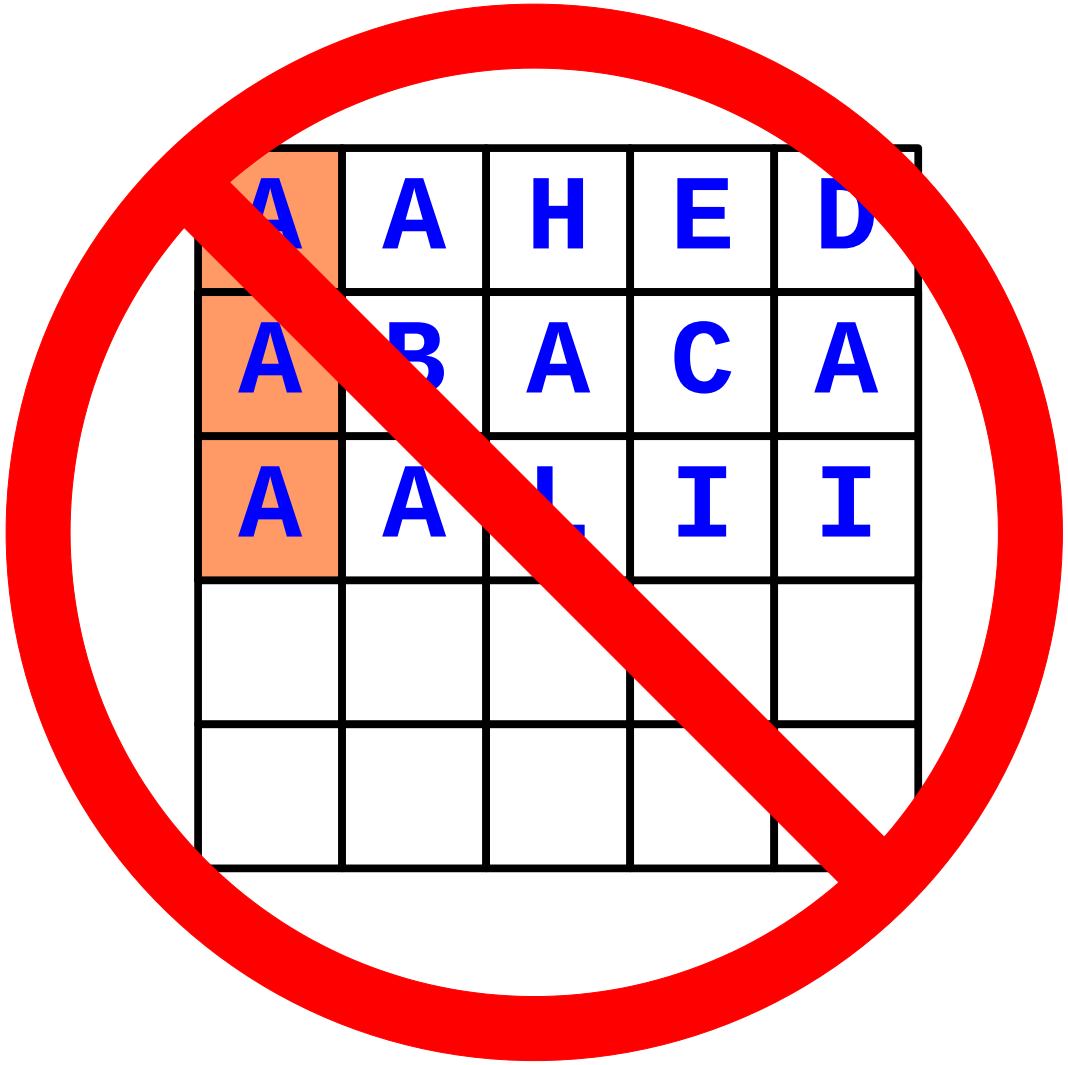
Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A
A	A	L	I	I

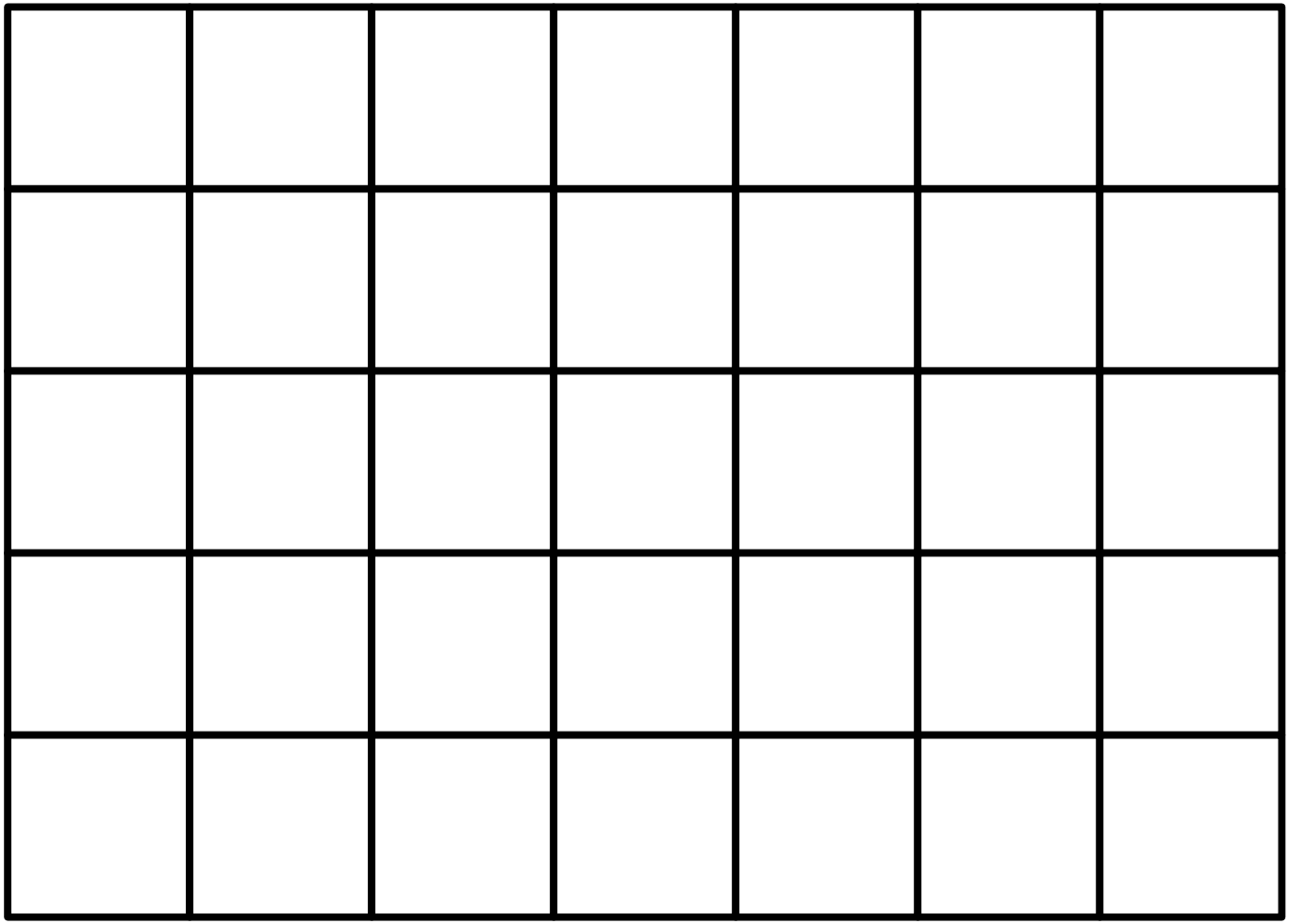
Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A
A	A	L	I	I

Generating Dense Crosswords



Let’s Code it Up!

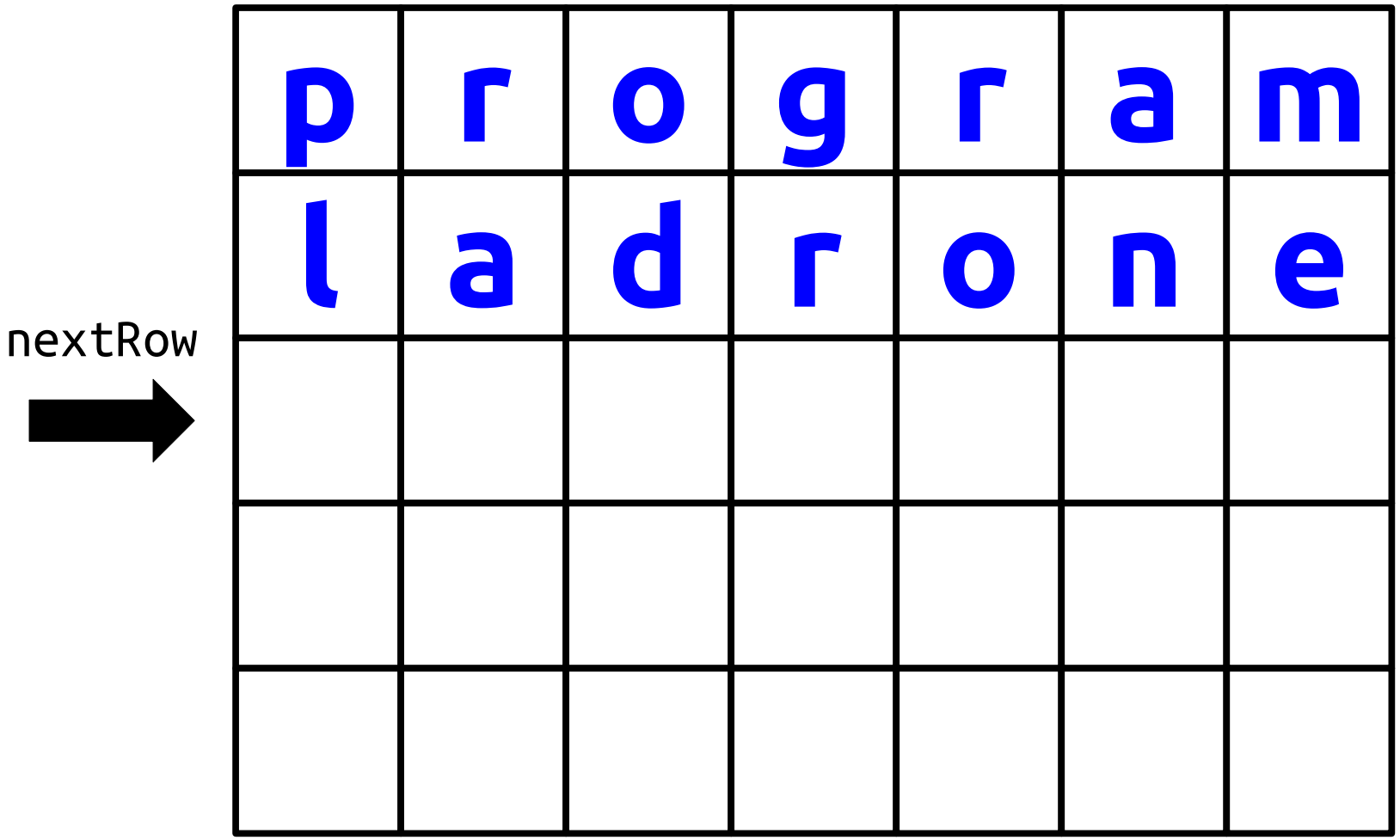


This word's length is the number of columns.

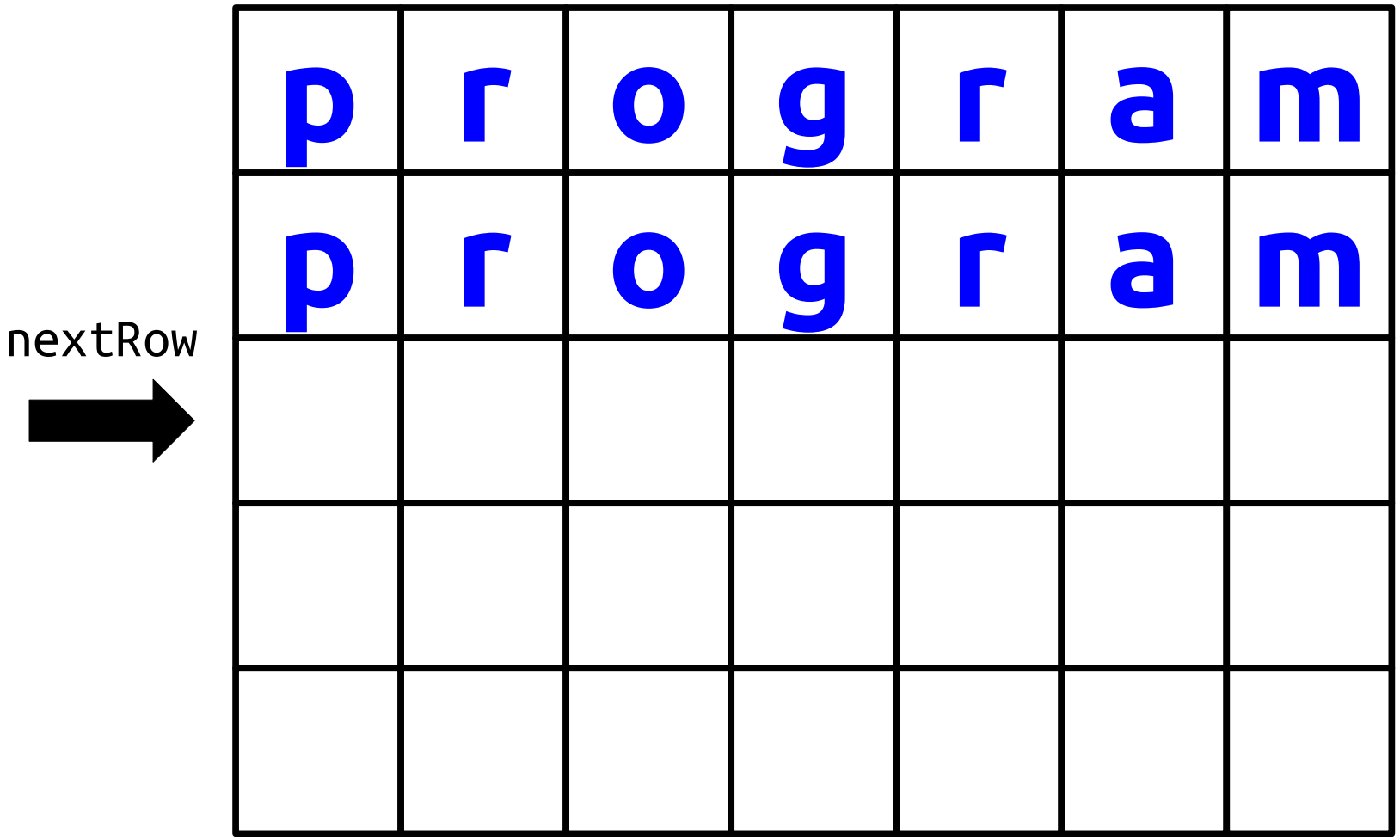
p	r	o	g	r	a	m

This word's length is the number of rows.

p						
l						
a						
c						
e						




```
bool canMakeCrosswordRec(Grid<char>& crossword,
                         int nextRow,
                         const Lexicon& rowWords,
                         const Lexicon& colWords);
```

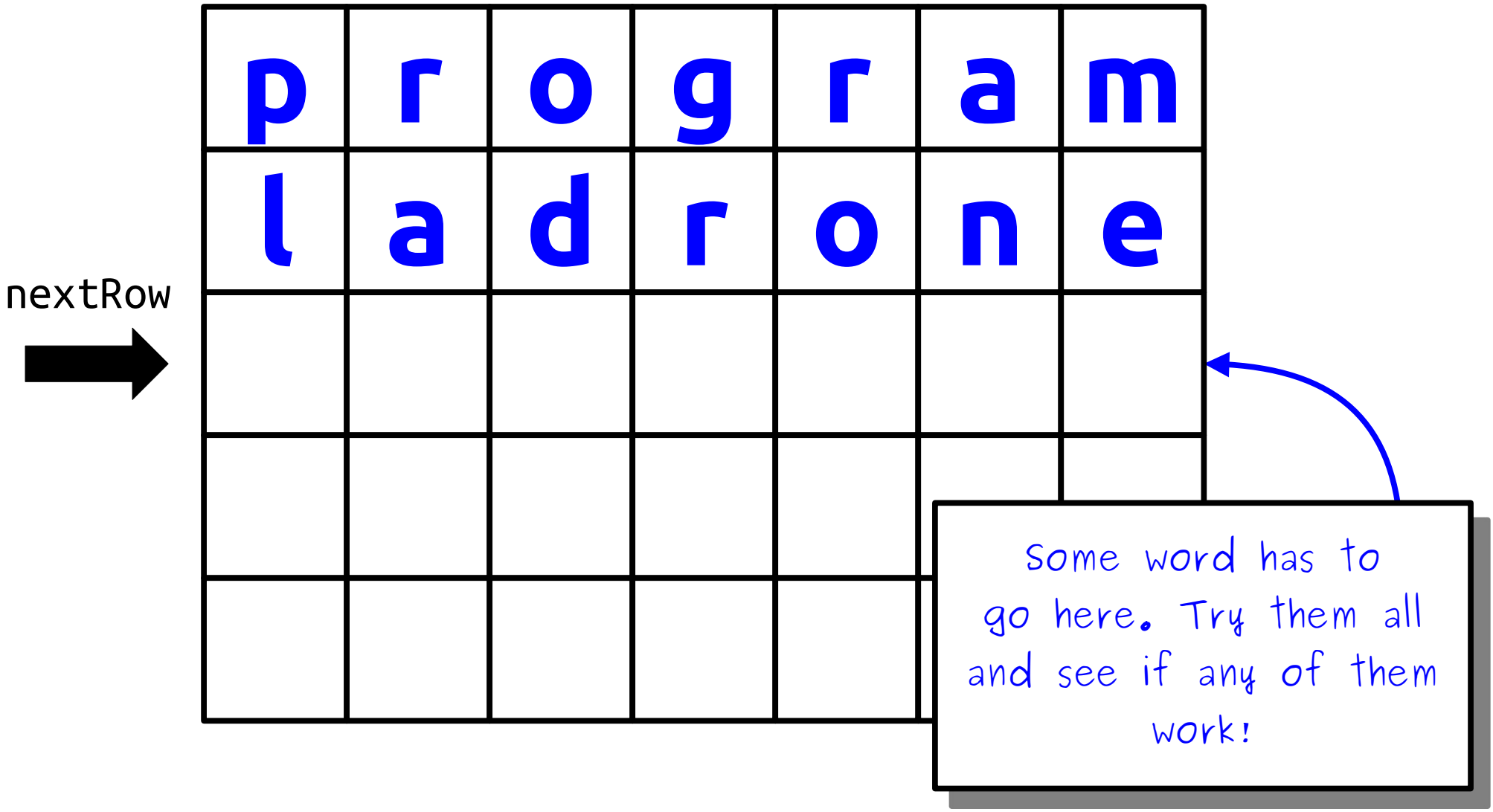



```
bool canMakeCrosswordRec(Grid<char>& crossword,
                         int nextRow,
                         const Lexicon& rowWords,
                         const Lexicon& colWords);
```

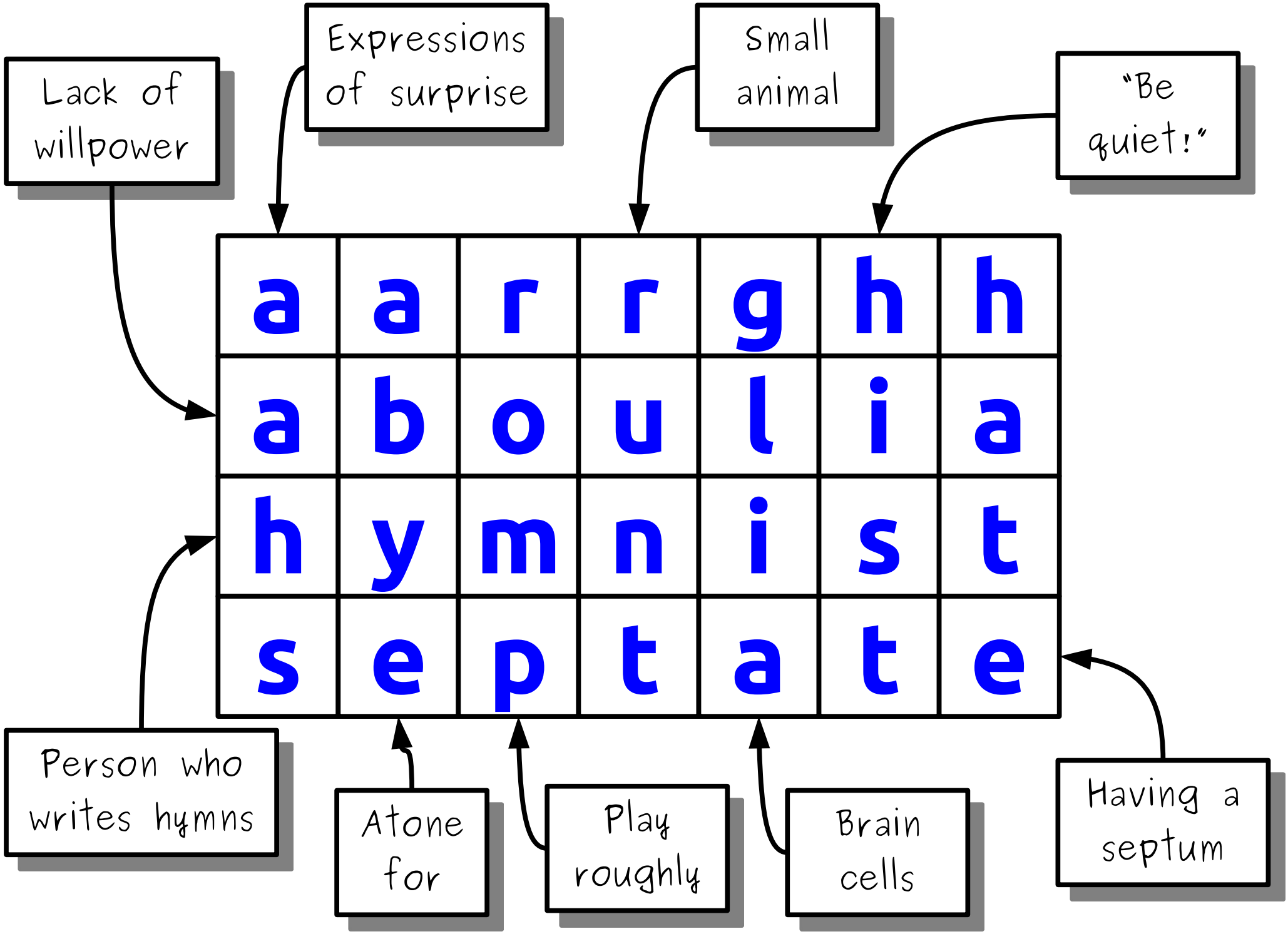
p	r	o	g	r	a	m
l	a	d	r	o	n	e
a	v	i	a	t	o	r
c	e	s	t	o	d	e
e	n	t	e	r	e	r

nextRow

```
bool canMakeCrosswordRec(Grid<char>& crossword,
                         int nextRow,
                         const Lexicon& rowWords,
                         const Lexicon& colWords);
```




```
bool canMakeCrosswordRec(Grid<char>& crossword,
                         int nextRow,
                         const Lexicon& rowWords,
                         const Lexicon& colWords);
```


Lack of willpower
Expressions of surprise
Small animal
"Be quiet!"
a a r r g h h
a b o u l i a
h y m n i s t
s e p t a t e
Person who writes hymns
Atone for
Play roughly
Brain cells
Having a septum

Going Deeper

- You can speed this up even more if you're more clever. Here are some thoughts to get you started:
  - Once you've placed a few rows down, the columns will be very constrained. Consider switching to going one *column* at a time versus one *row* at a time at that point.
  - Figure out which row or column is most constrained at each point, and only focus on that row/column.
- ***Completely optional challenge:*** Make this program run faster, and find a cool dense crossword. If you find something interesting (and PG-13), we'll share it with the rest of the class!

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!
- ***Start Assignment 4.***
  - Slow and steady progress is the name of the game here. Aim to complete the debugging exercise and to make good progress on Matchmaker over the weekend.

Next Time

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