

Thinking Recursively

Part V

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
---	---	---	---

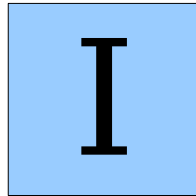
One Solution

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

One Solution

I	N
---	---

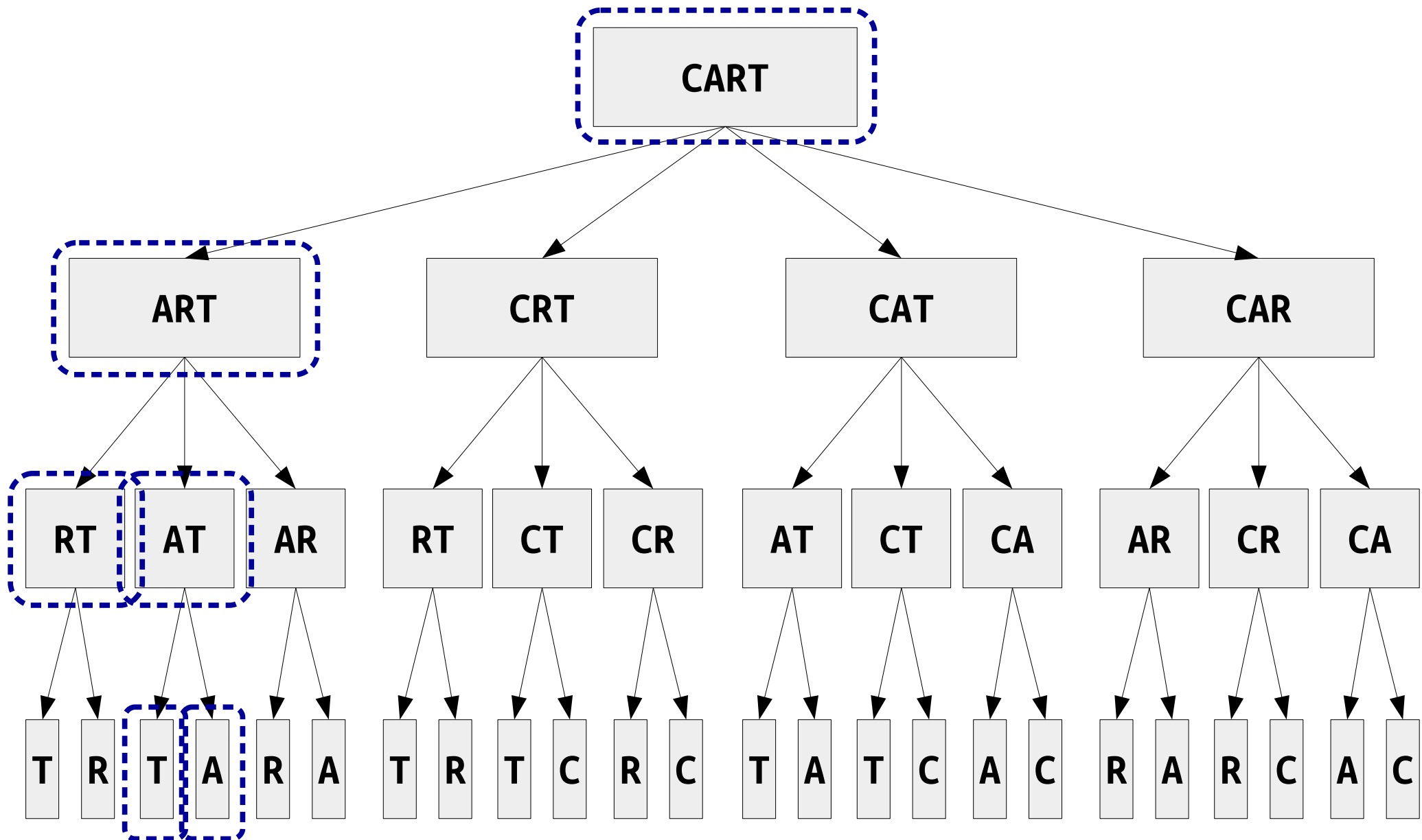
One Solution



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;
}
```

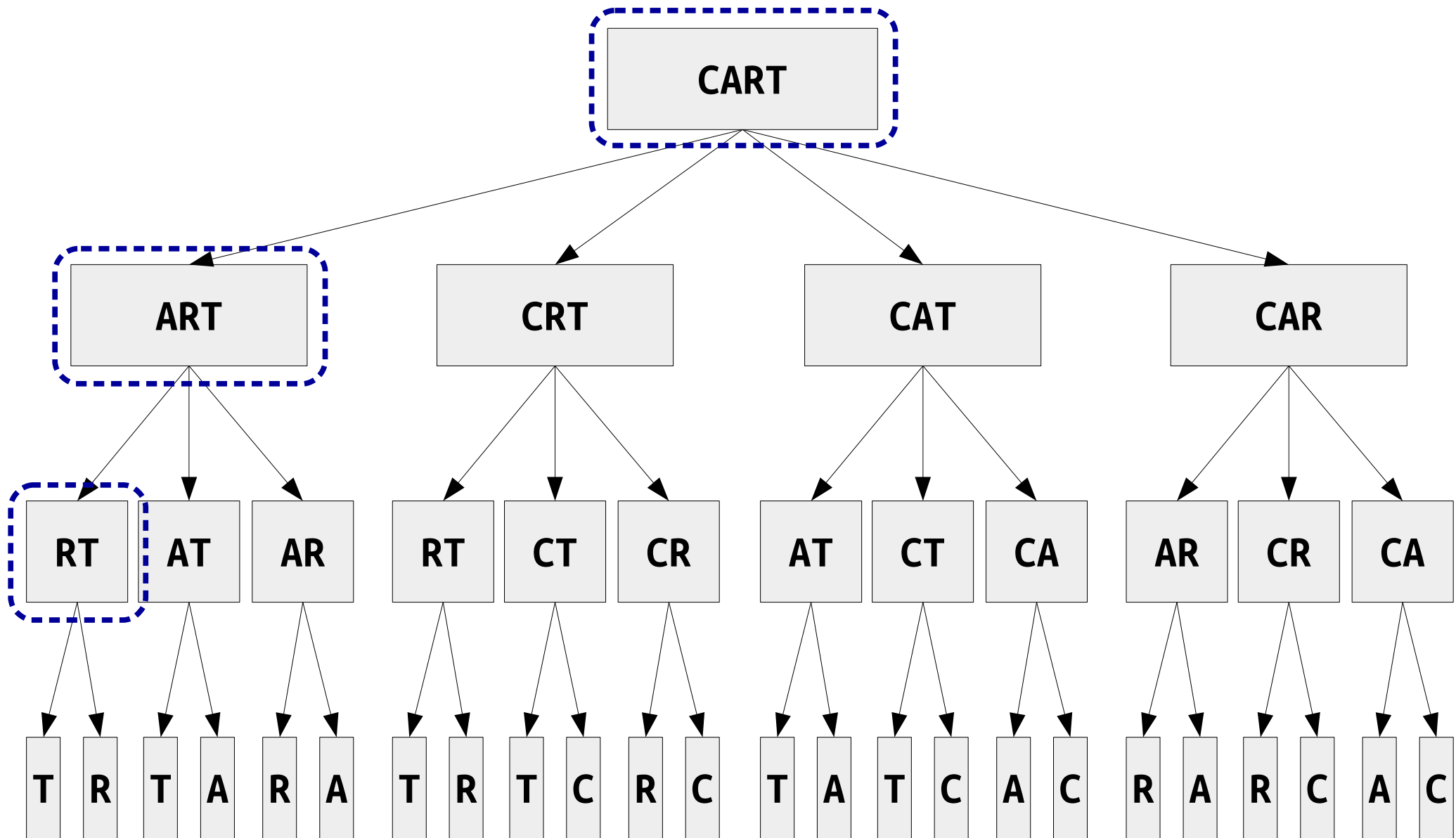
```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    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;
}
```

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;  
}
```

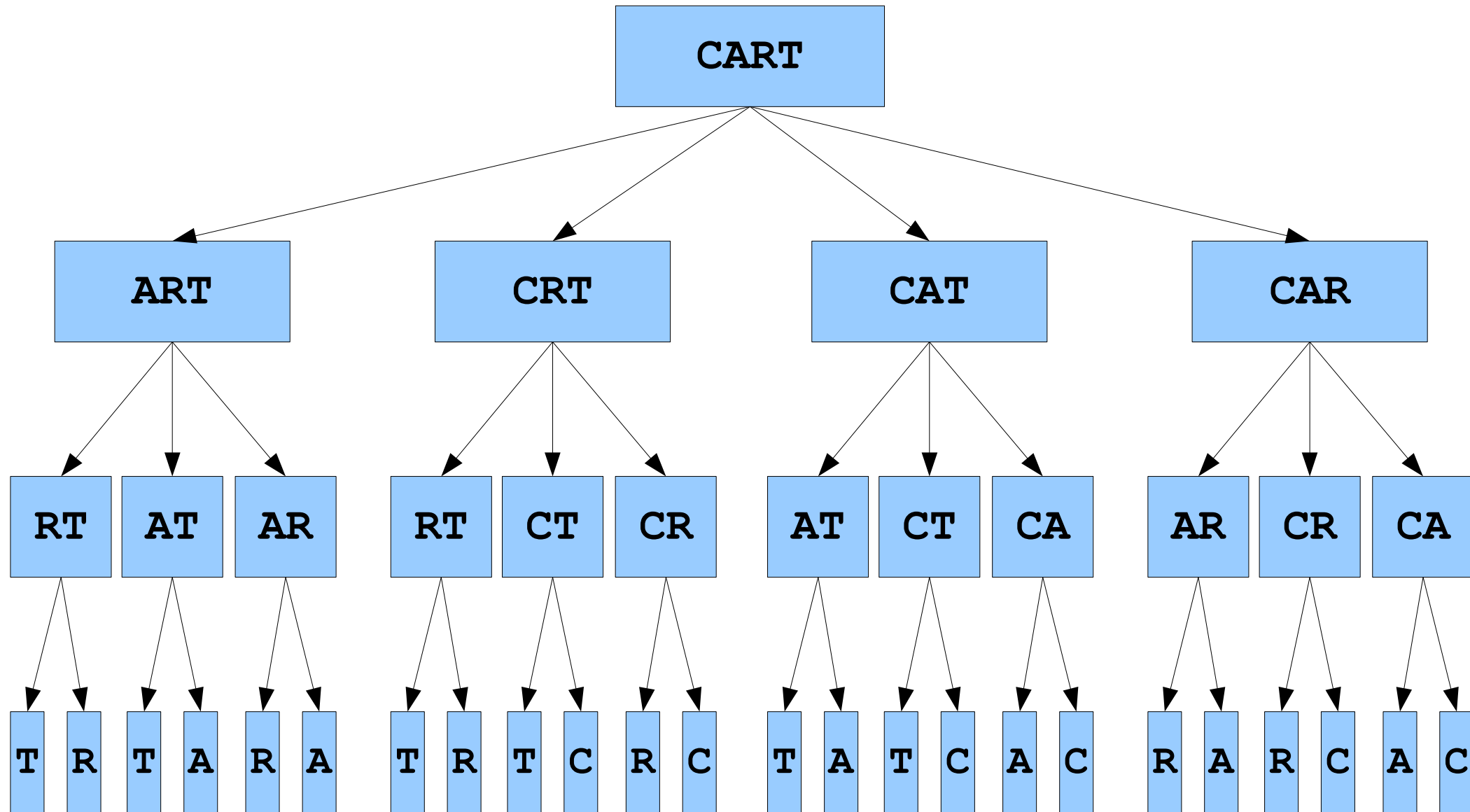
Note that if the recursive call succeeds, then we return success. If it doesn't succeed, that doesn't mean we've failed - it just means we need to try out the next option.

How do we know we're correct?

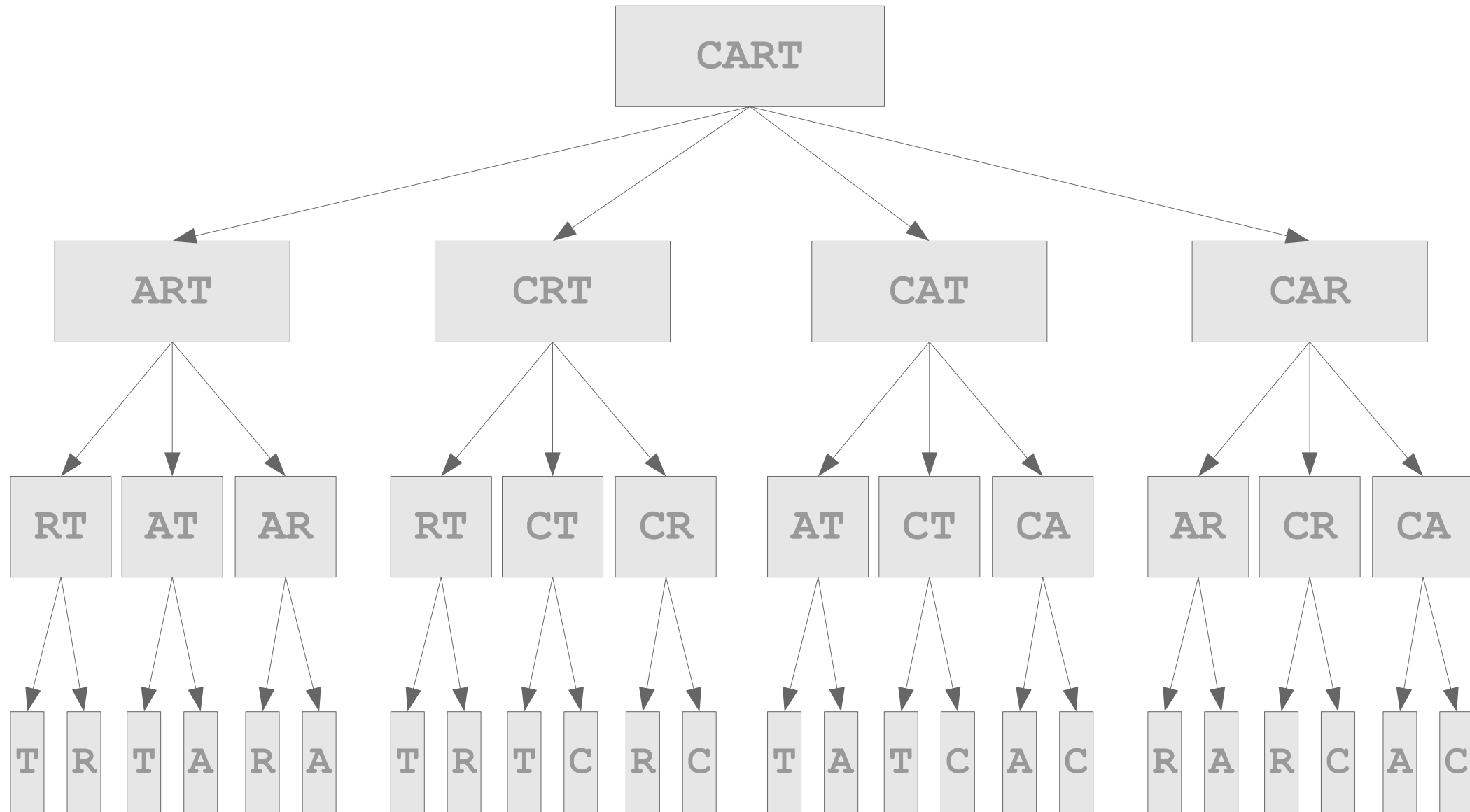
Output Parameters

- An ***output parameter*** (or ***outparam***) is a parameter to a function that stores the result of that function.
- Caller passes the parameter by reference, function overwrites the value.
- Often used with recursive backtracking:
 - The return value says whether a solution exists.
 - If one does, it's loaded into the outparameter.

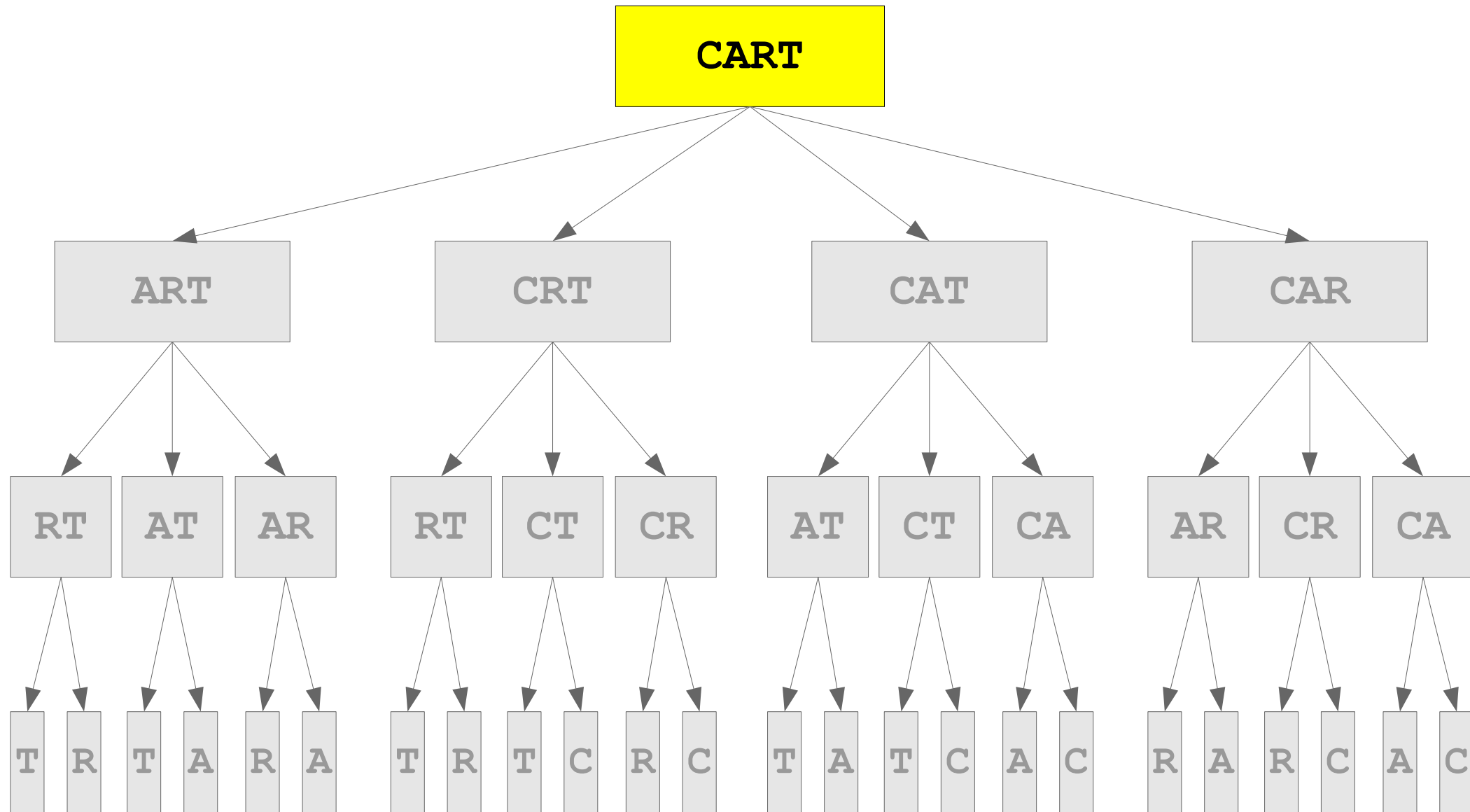
Generating the Answer



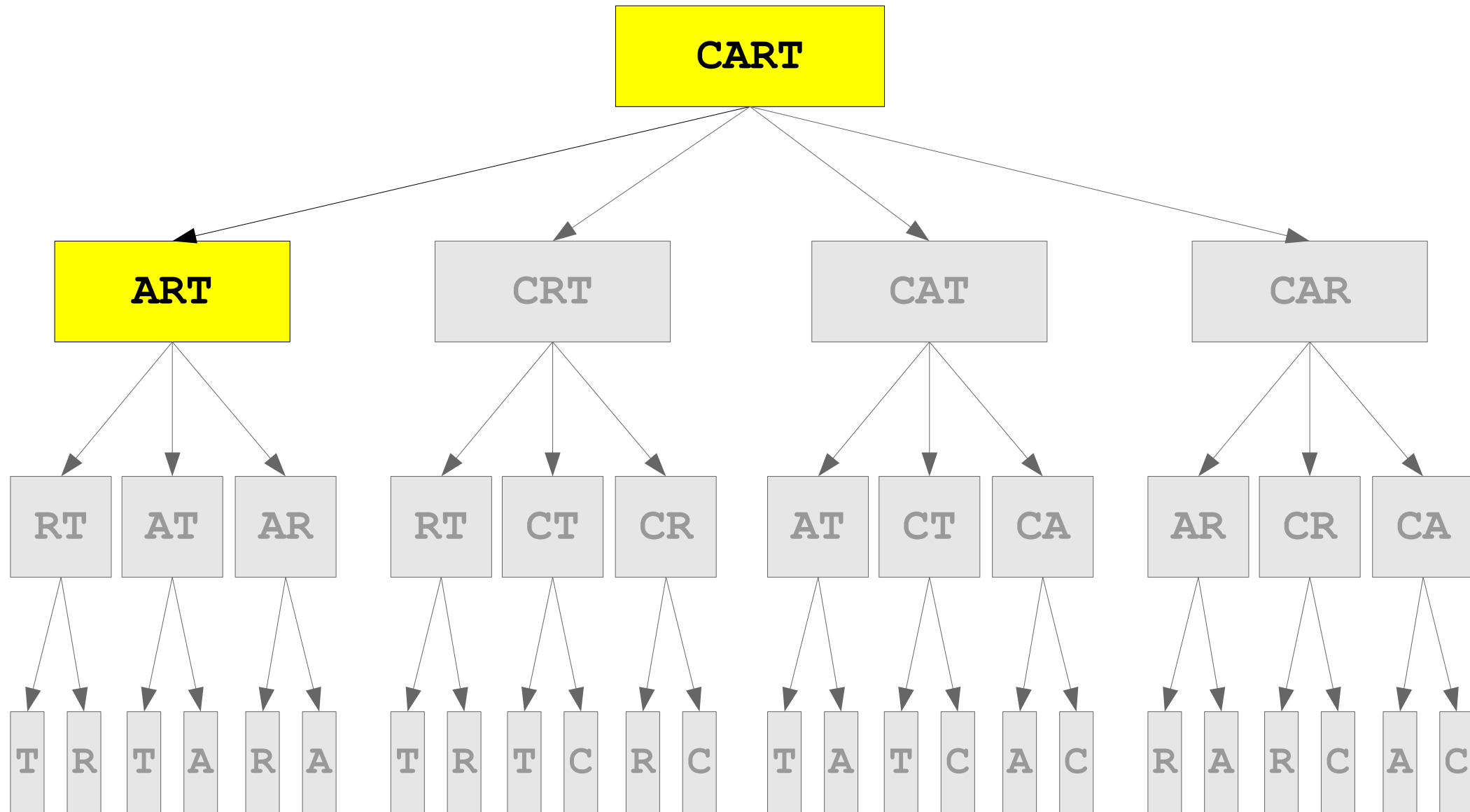
Generating the Answer



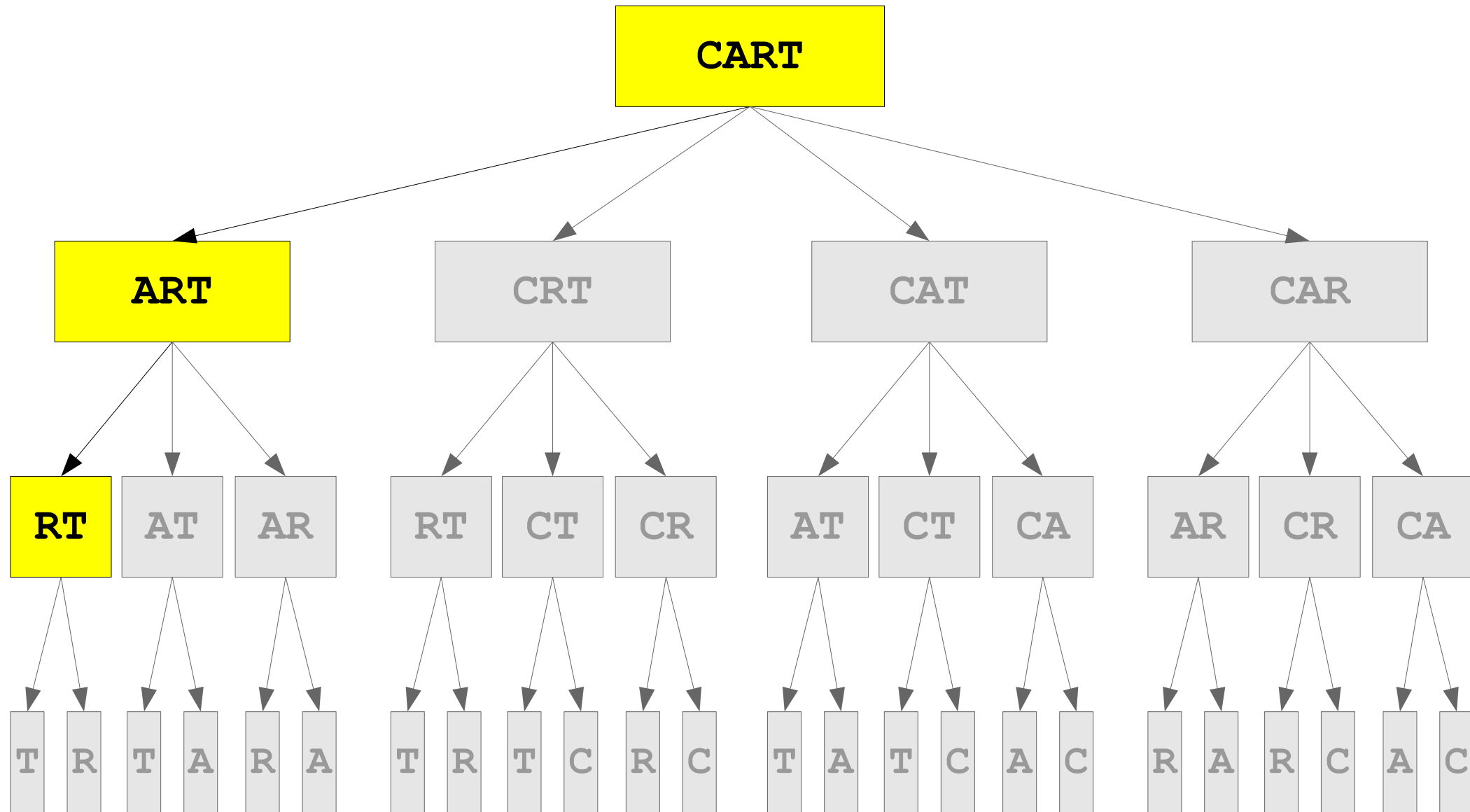
Generating the Answer



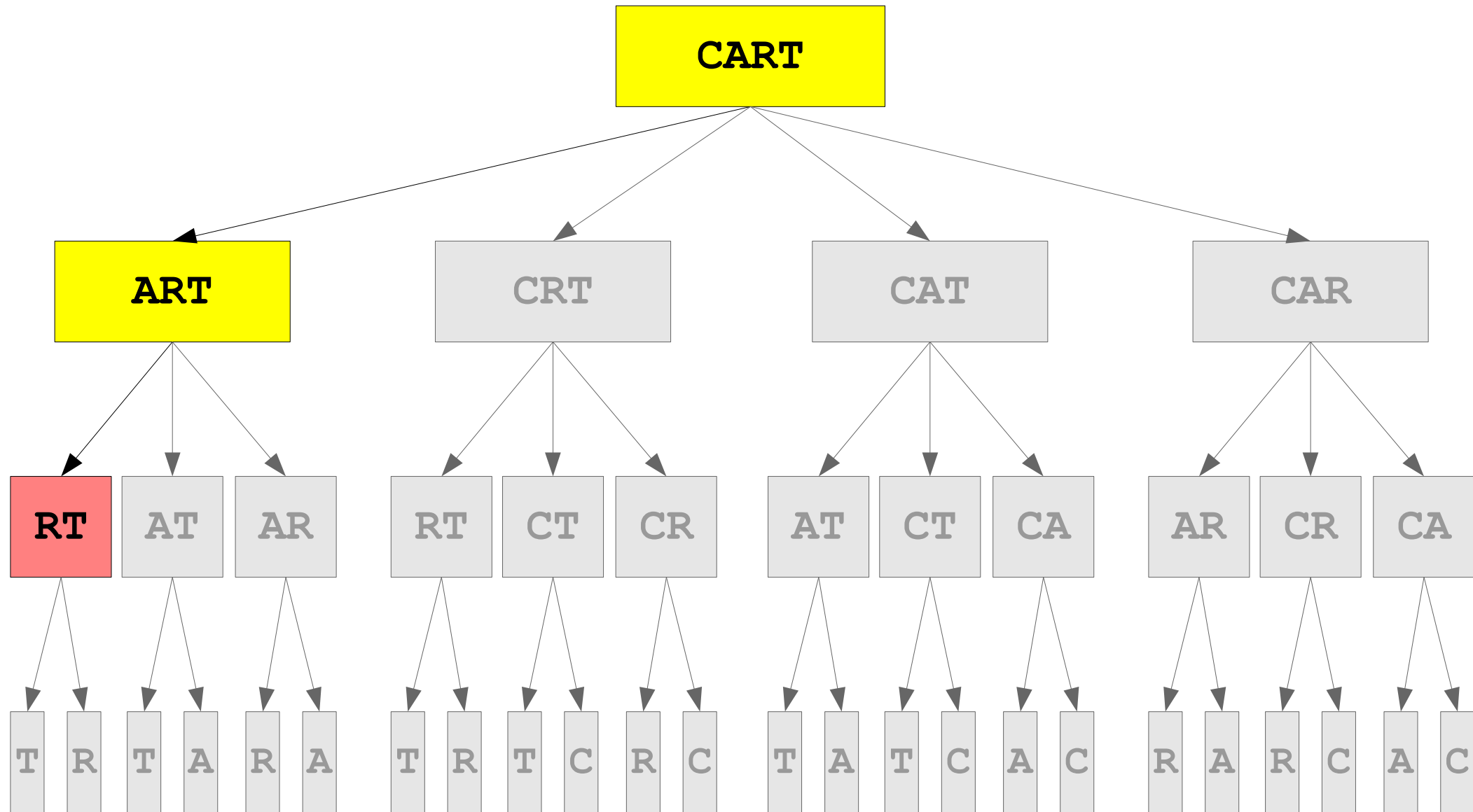
Generating the Answer



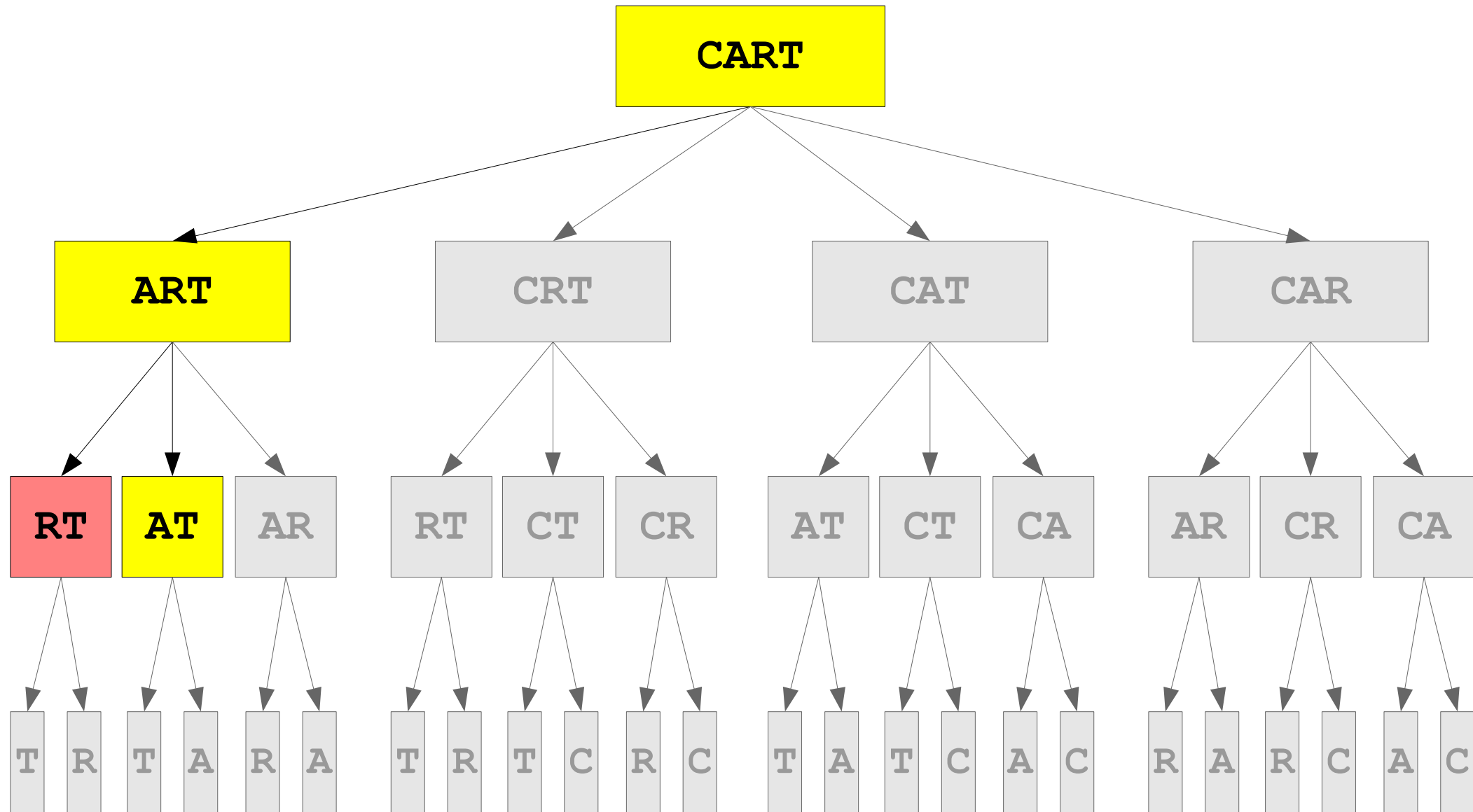
Generating the Answer



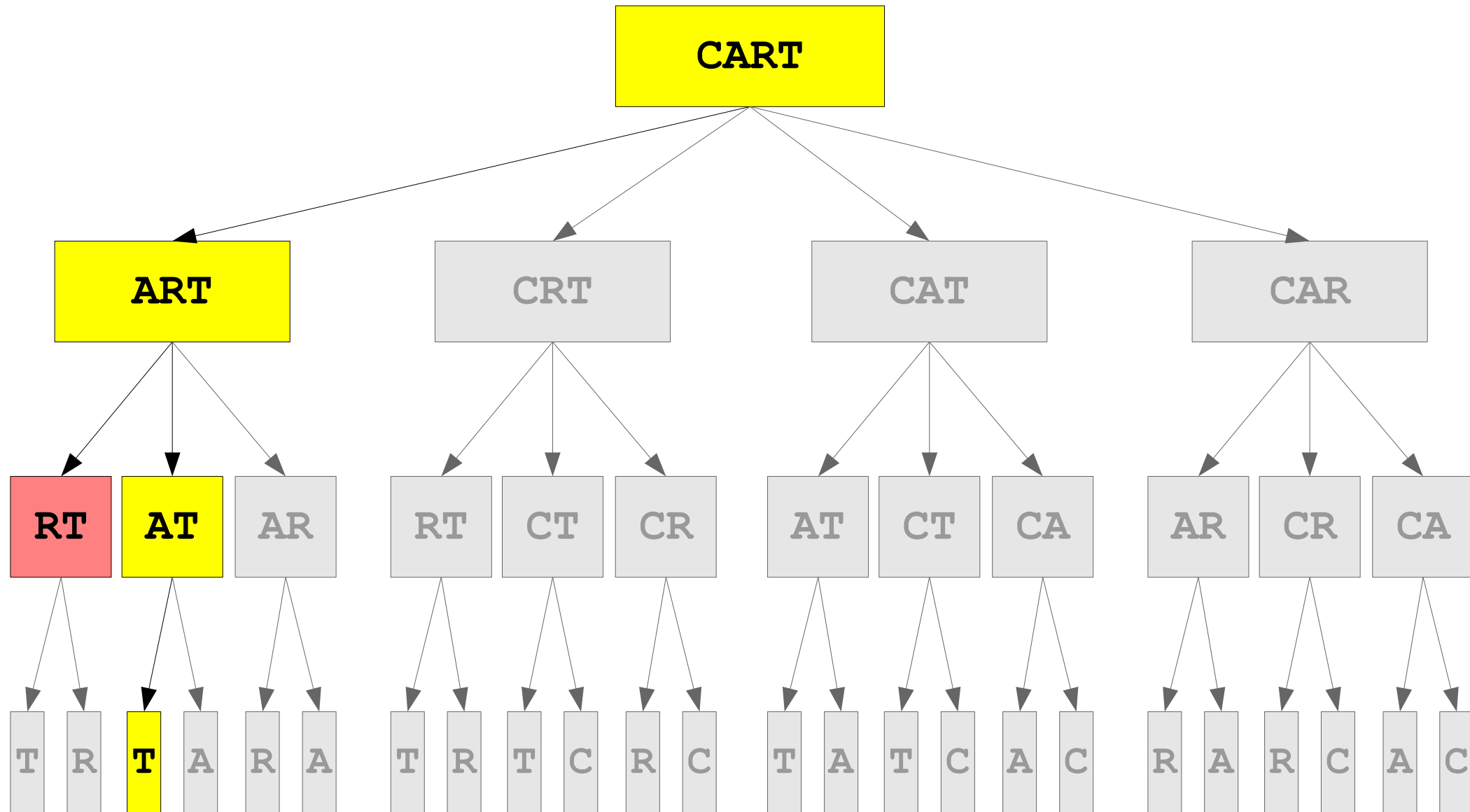
Generating the Answer



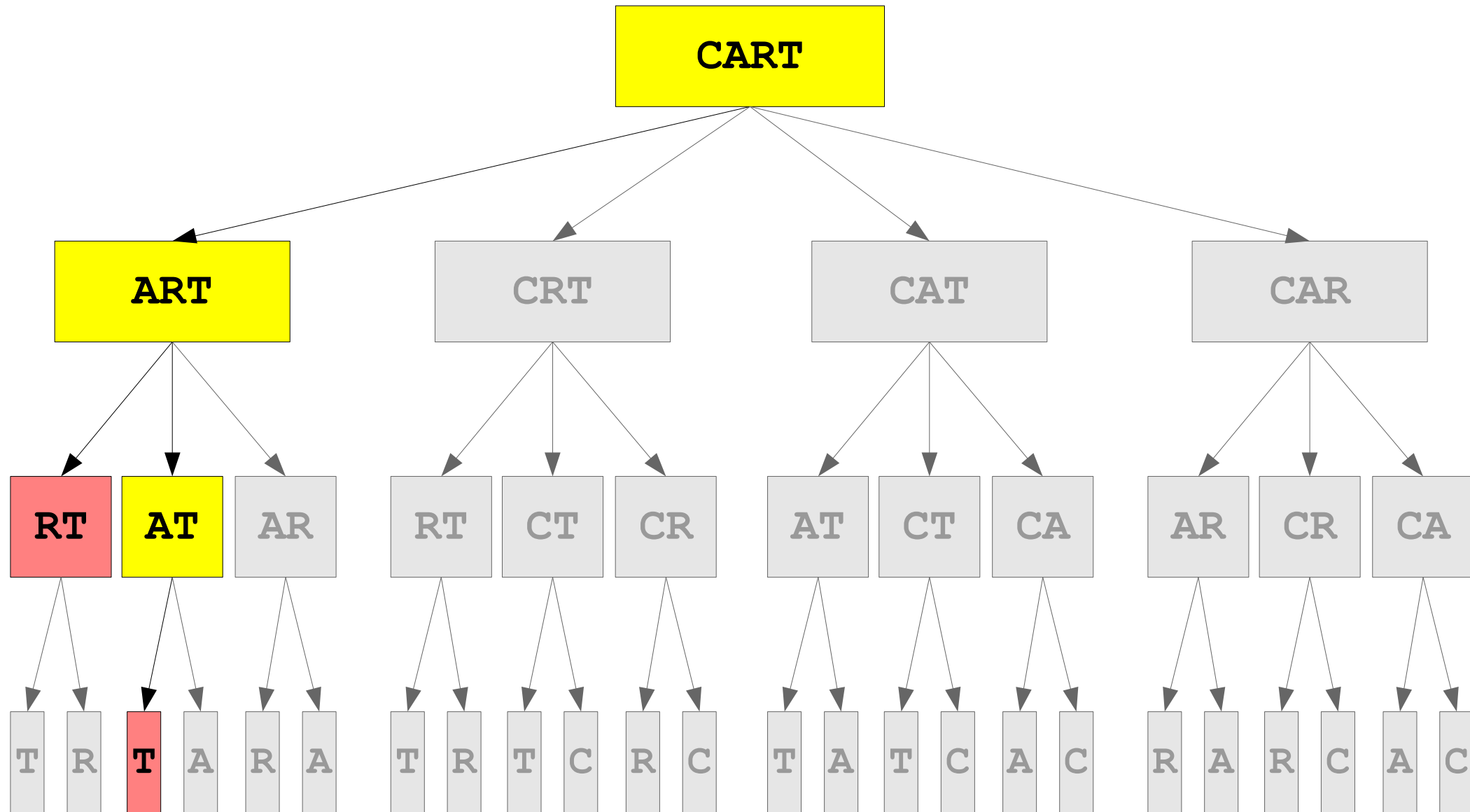
Generating the Answer



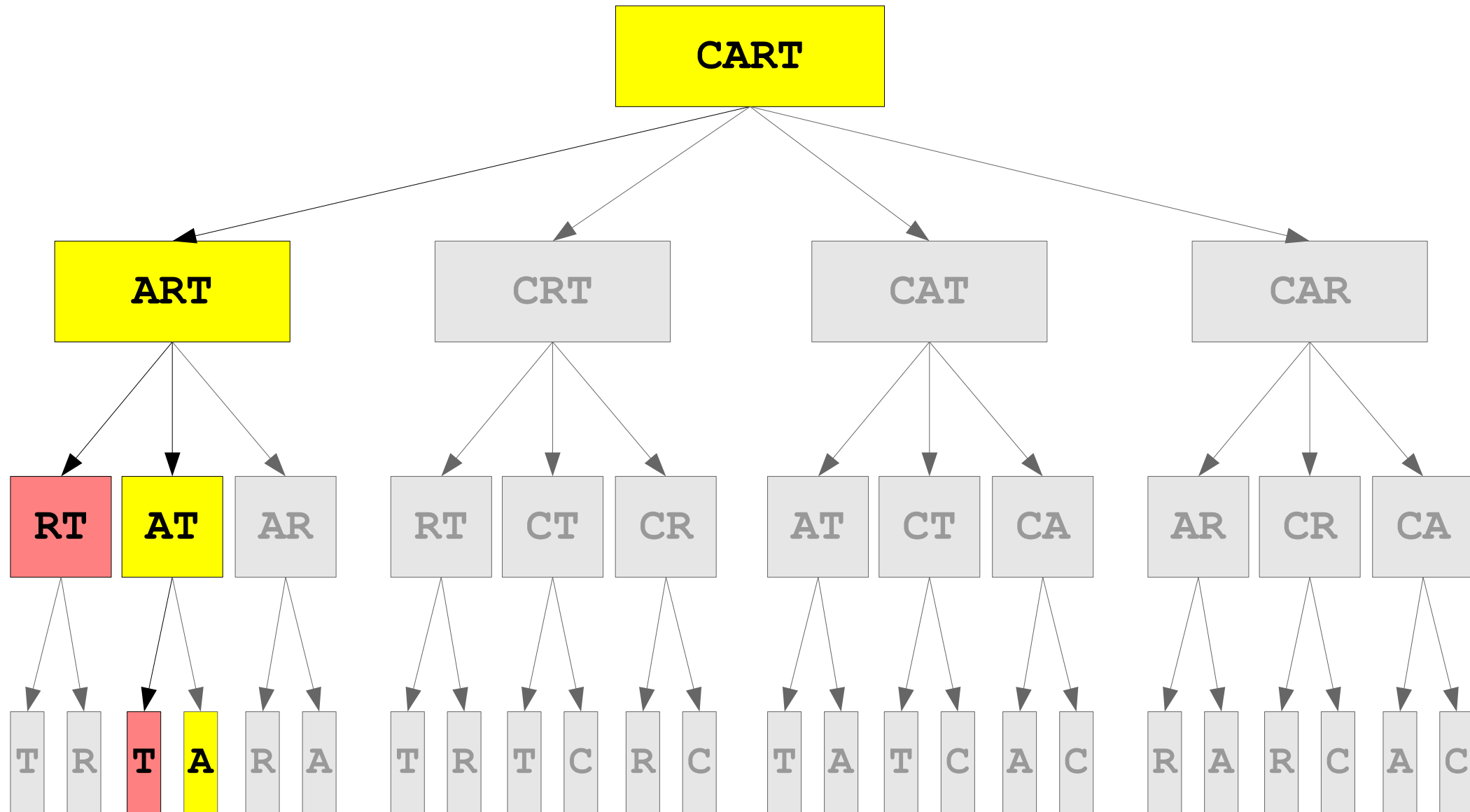
Generating the Answer



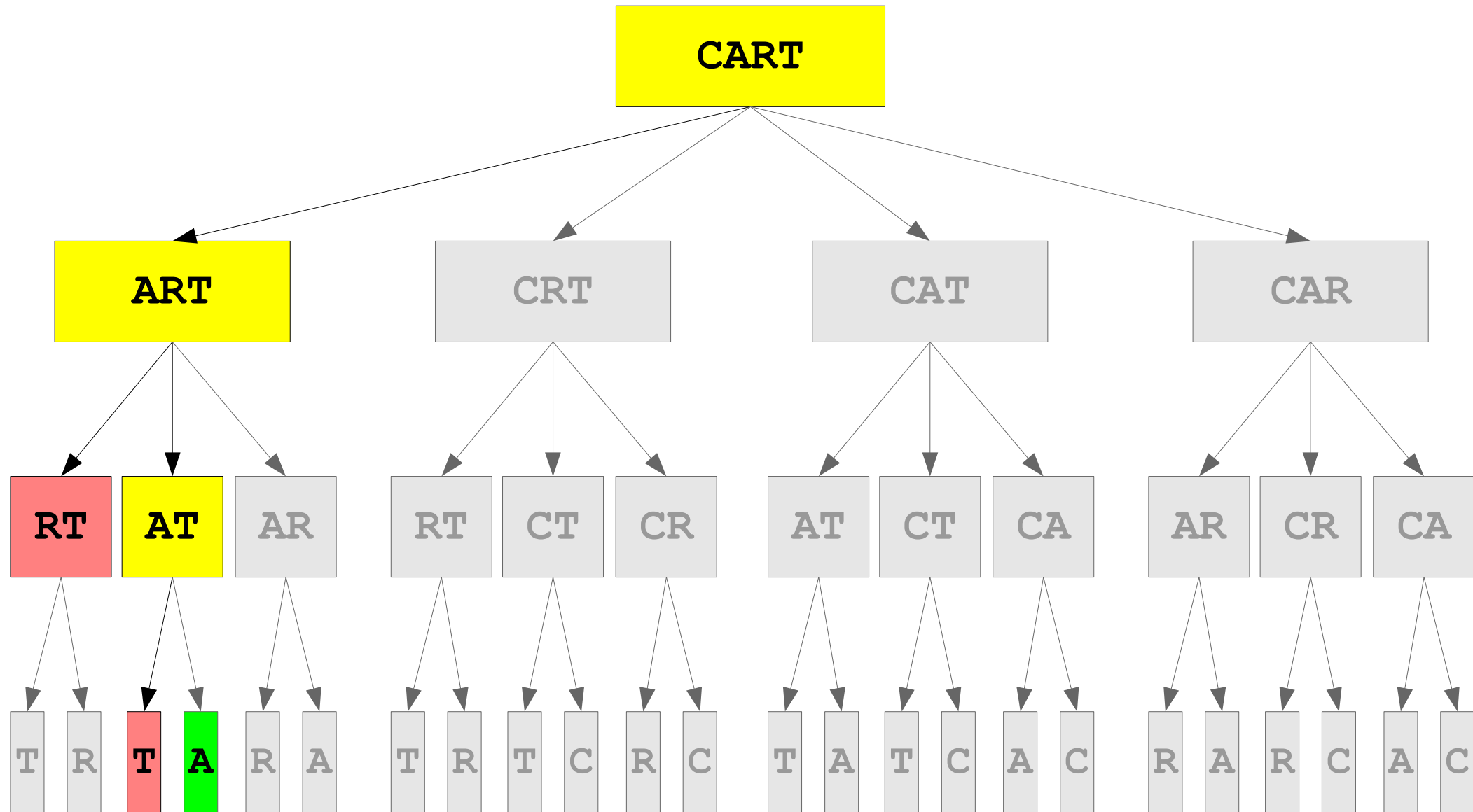
Generating the Answer



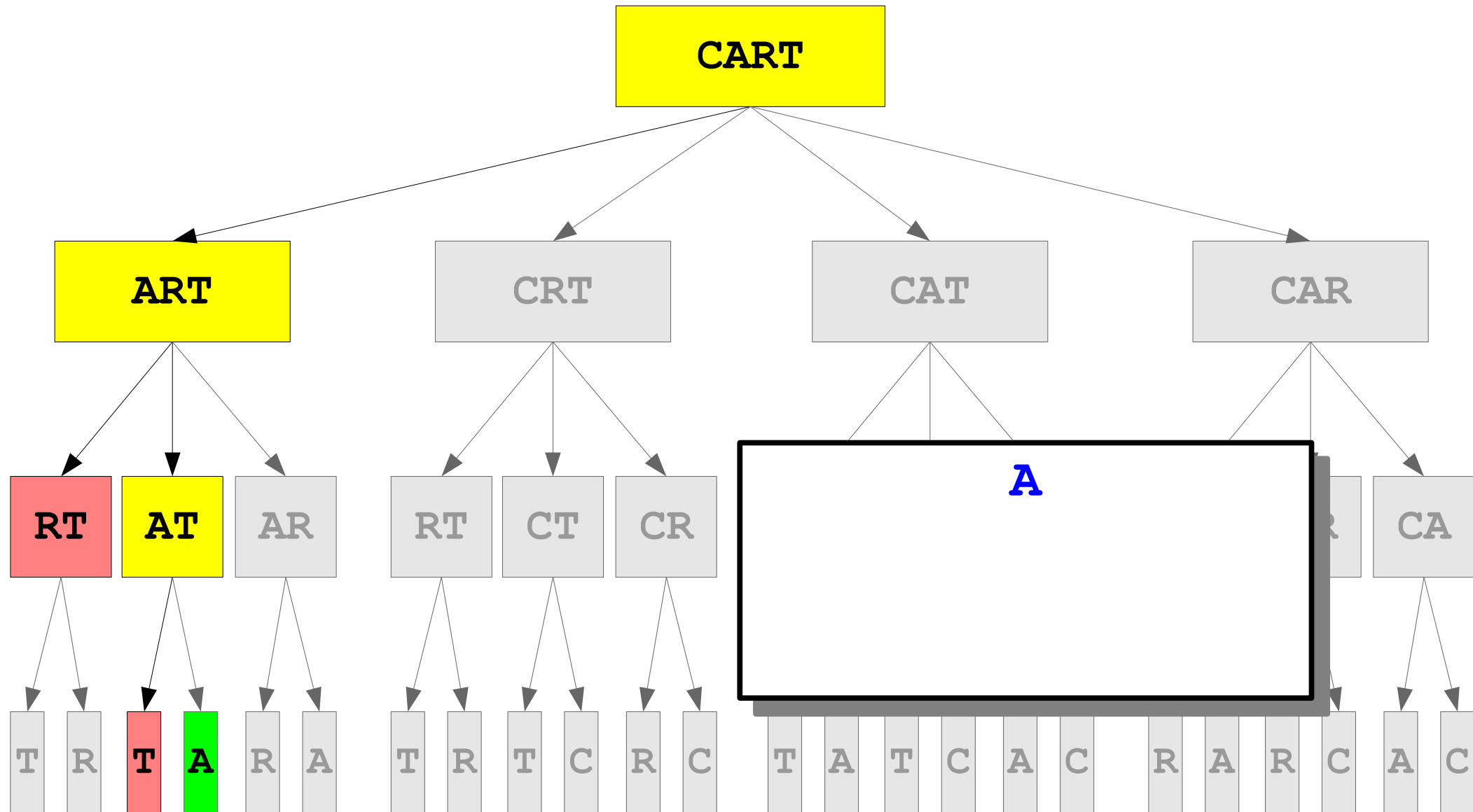
Generating the Answer



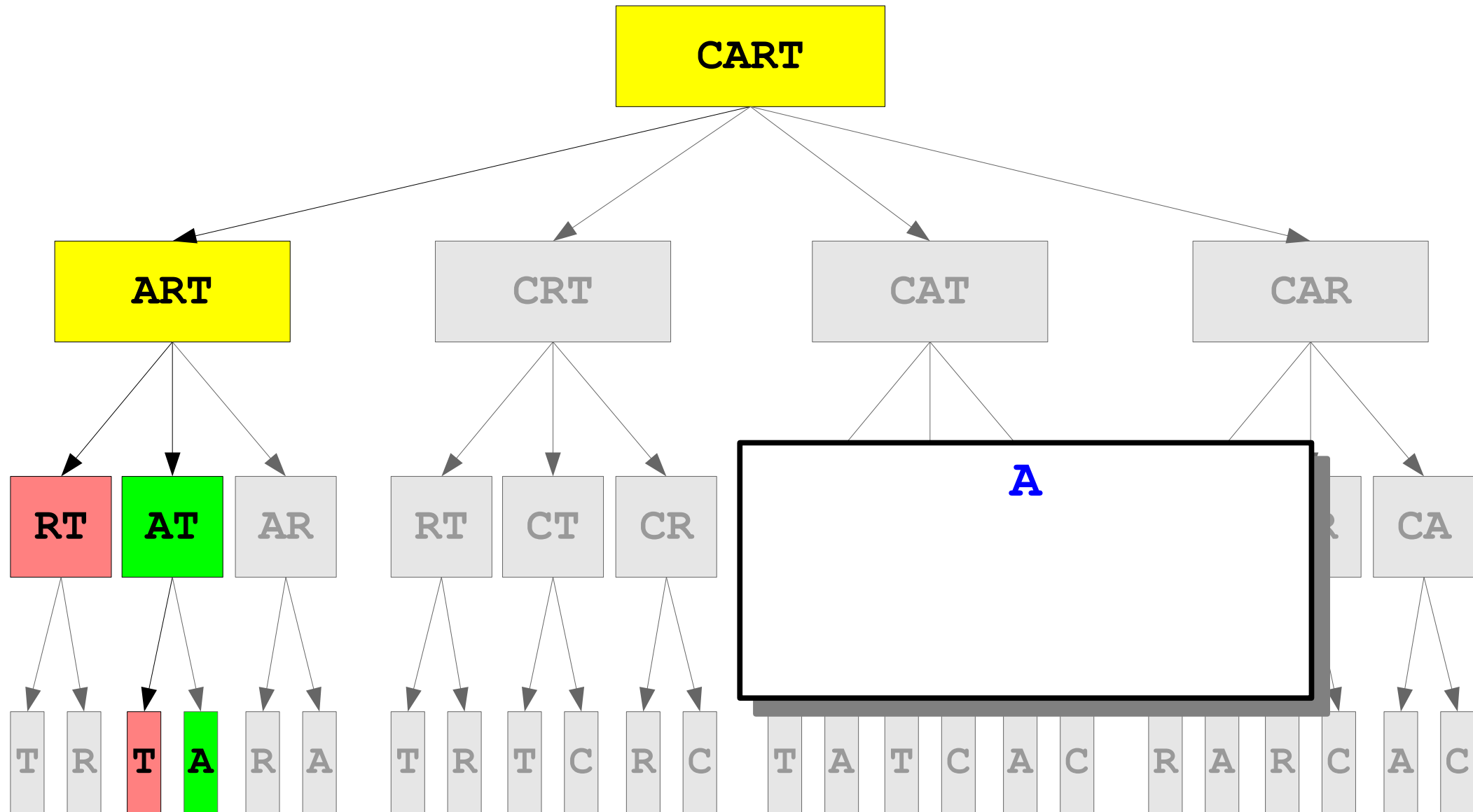
Generating the Answer



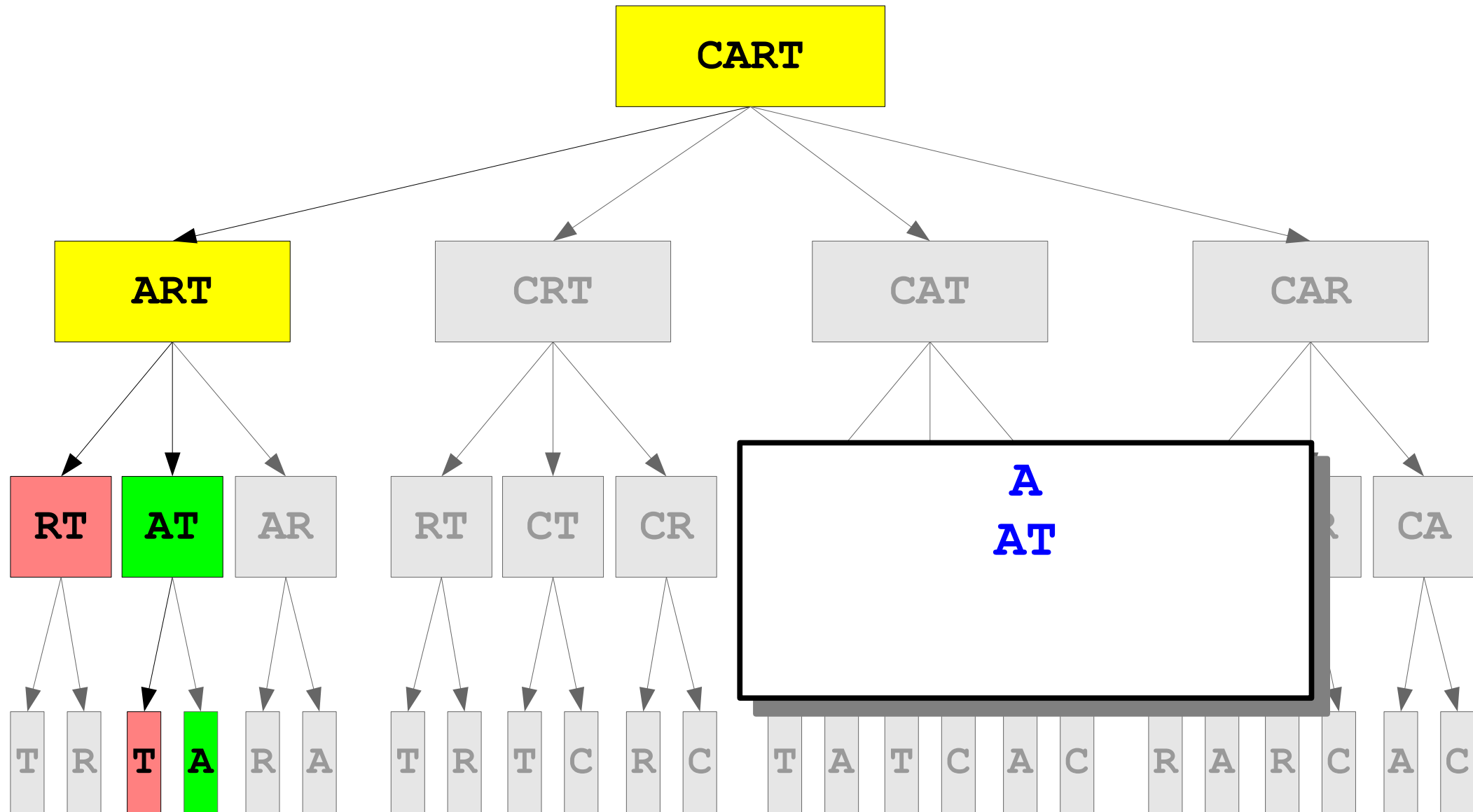
Generating the Answer



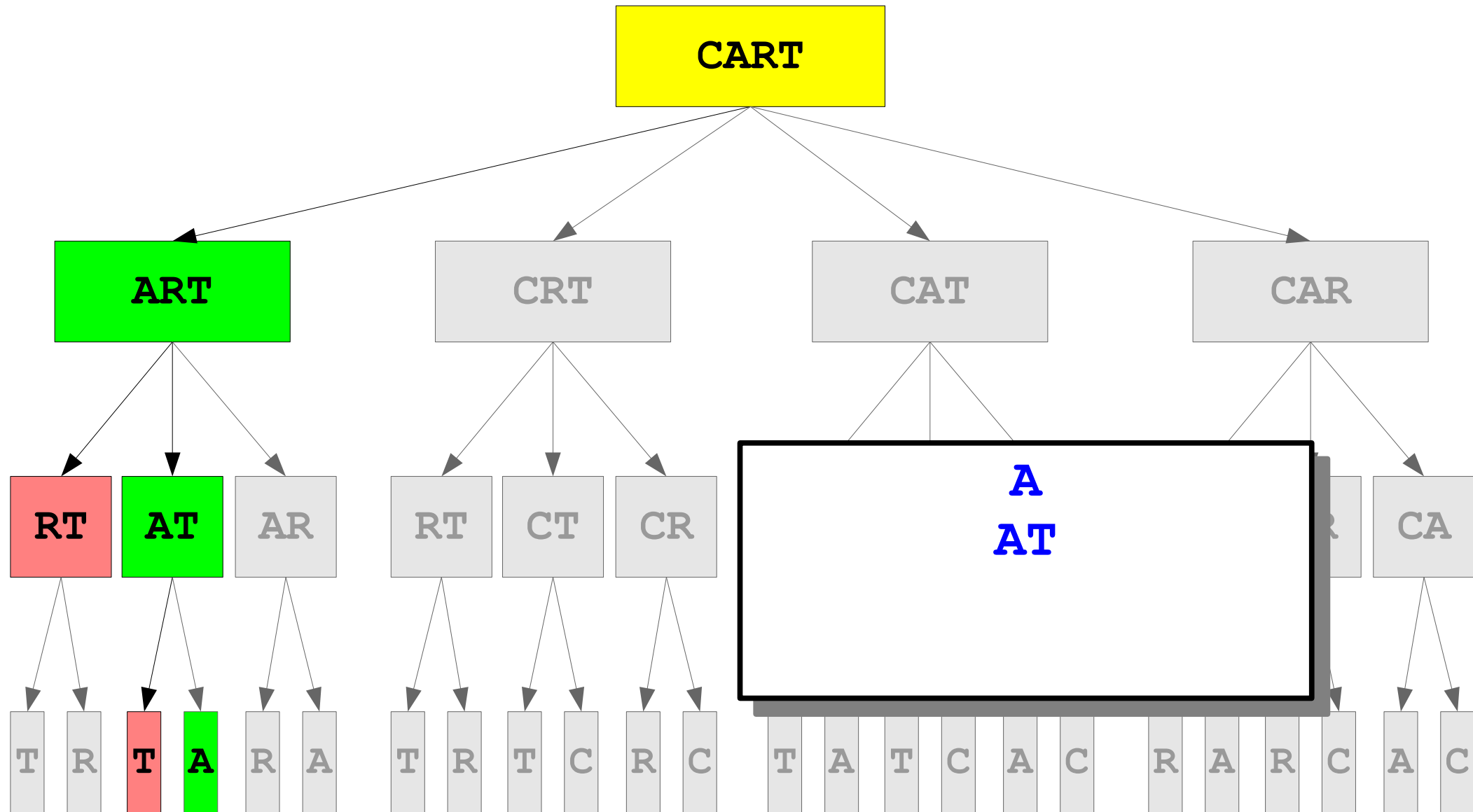
Generating the Answer



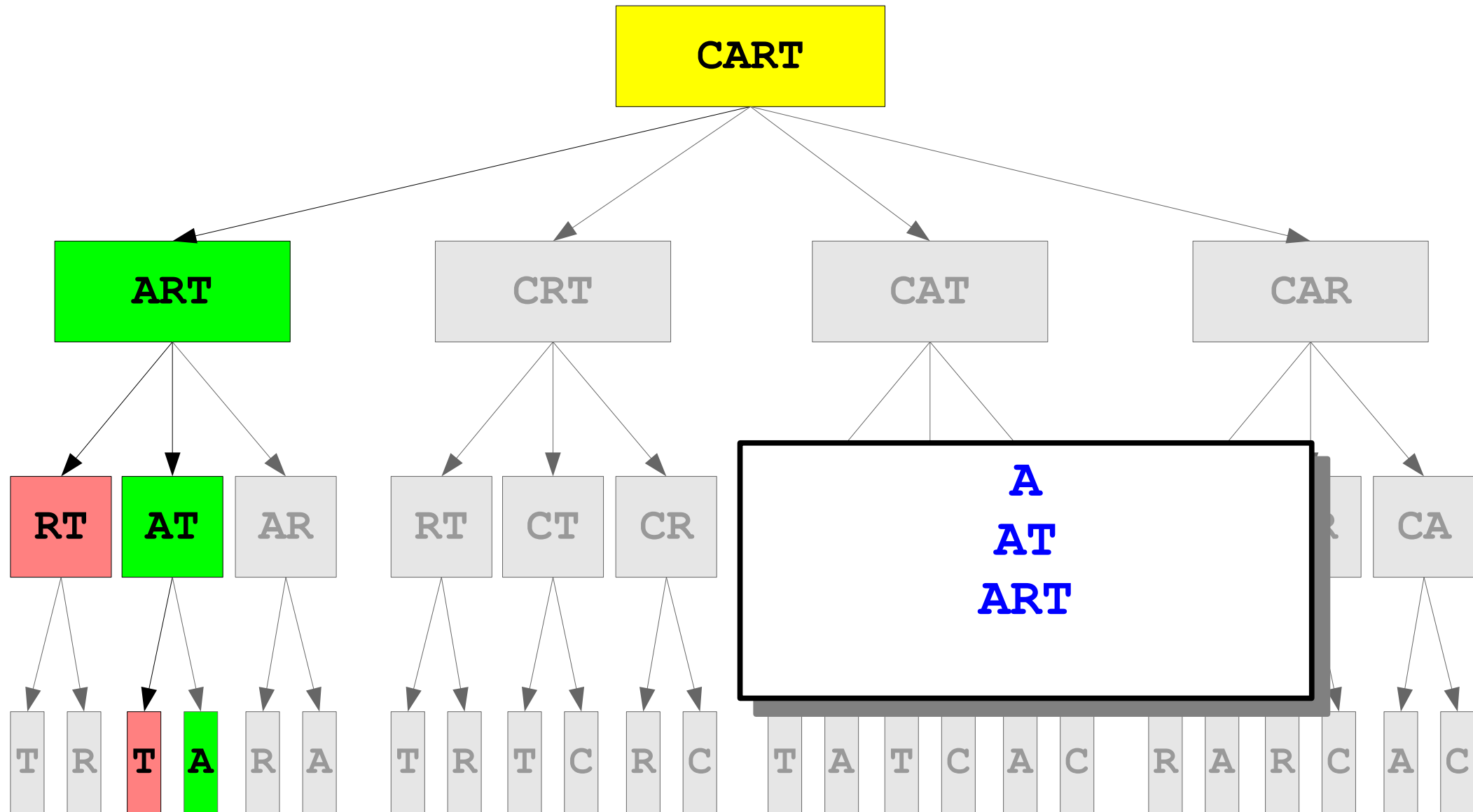
Generating the Answer



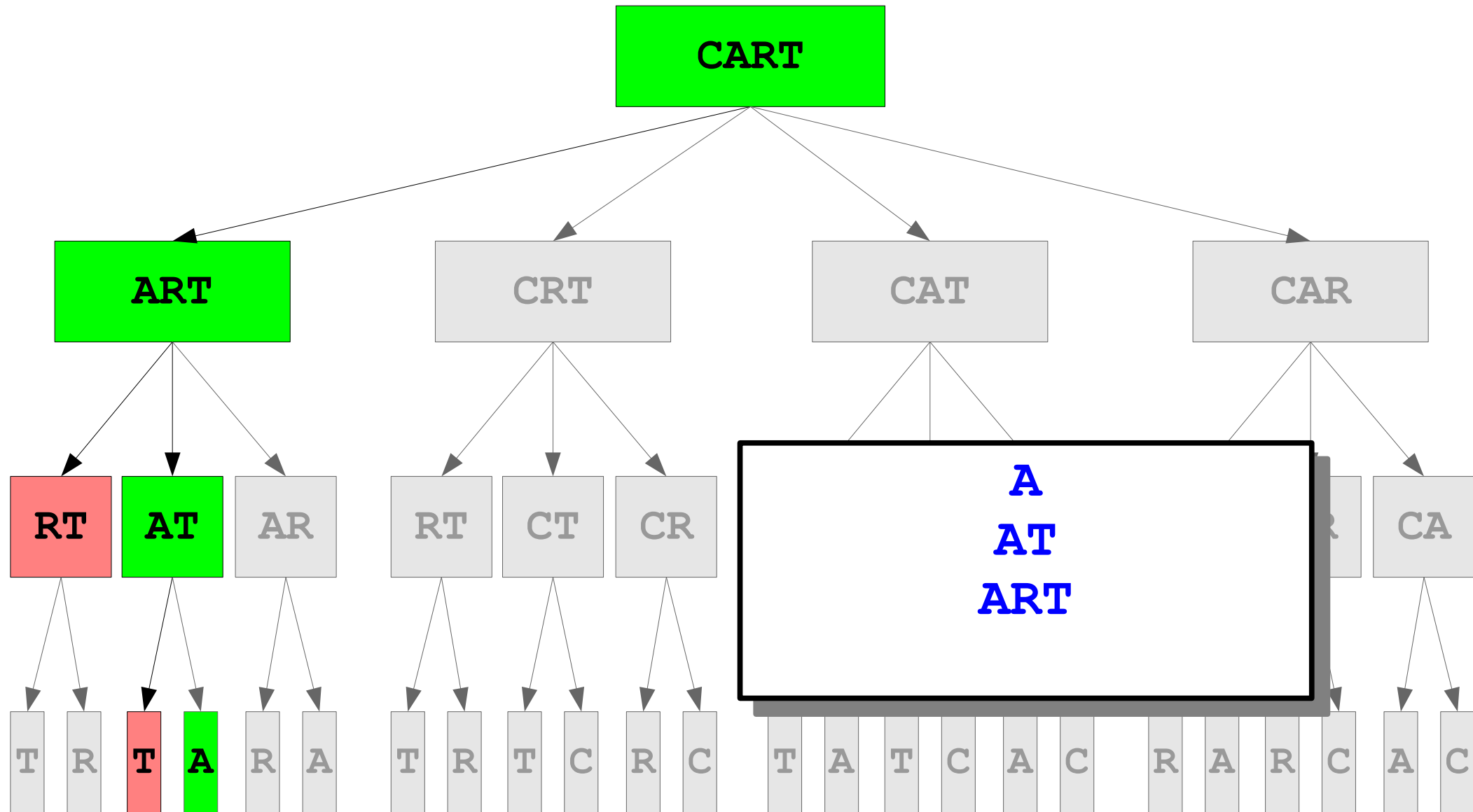
Generating the Answer



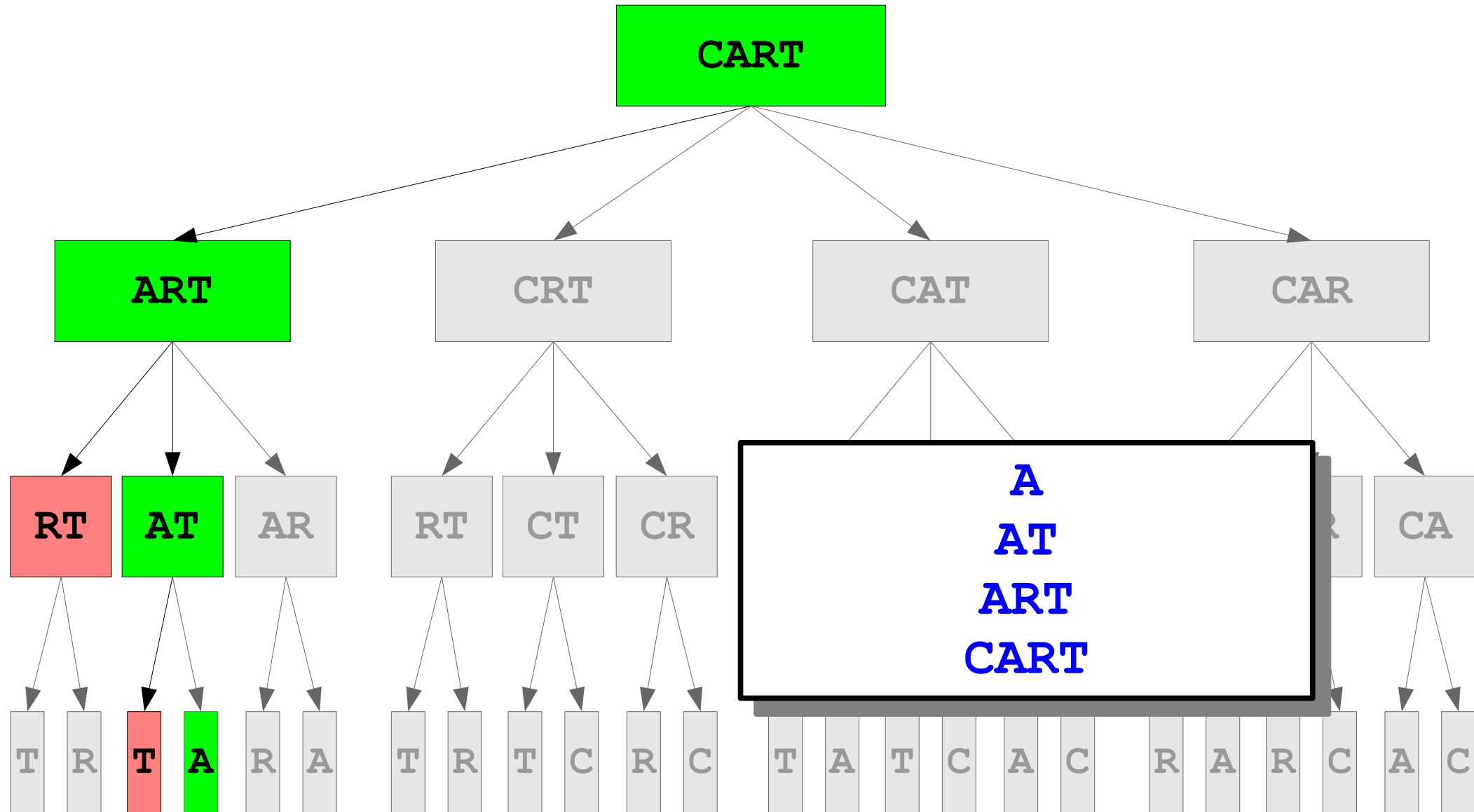
Generating the Answer



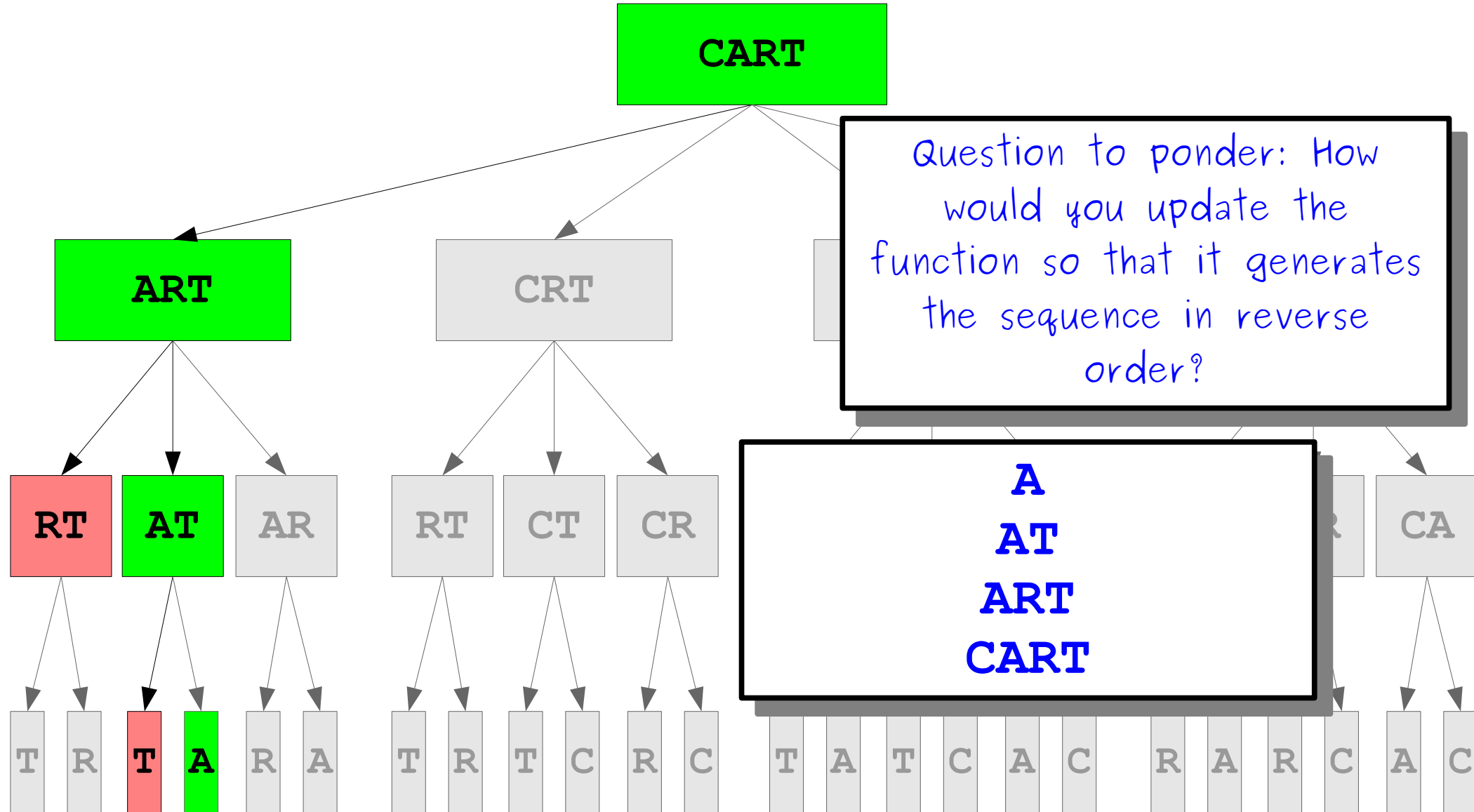
Generating the Answer



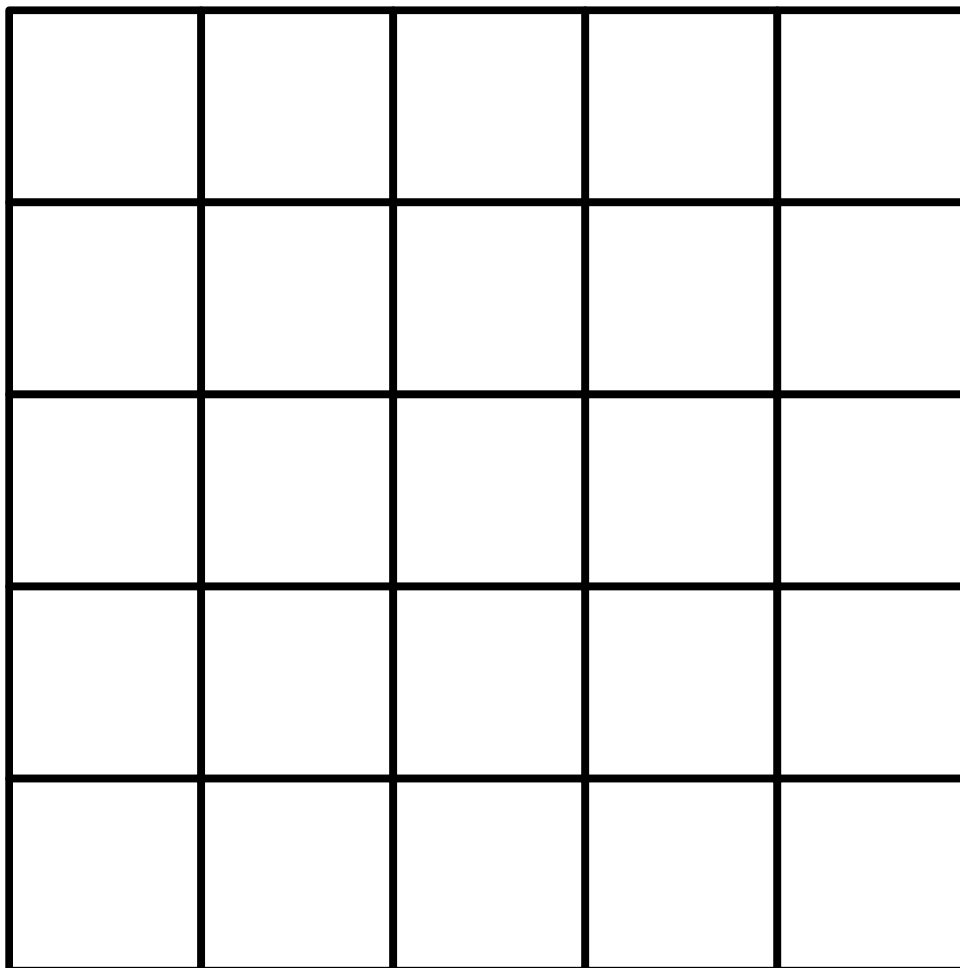
Generating the Answer



Generating the Answer



Dense Crosswords



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

Scoundrel

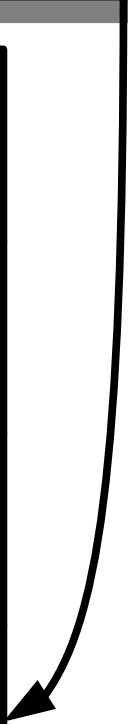
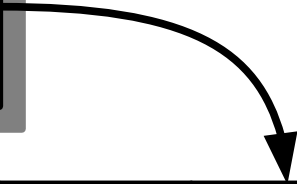
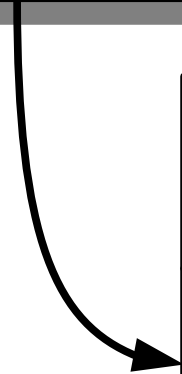
Where current
flows in

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

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

stuffed grape leaves

Hilly

Synonym for keyboard

Bind with lace

Thanks for former CS106B student 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"



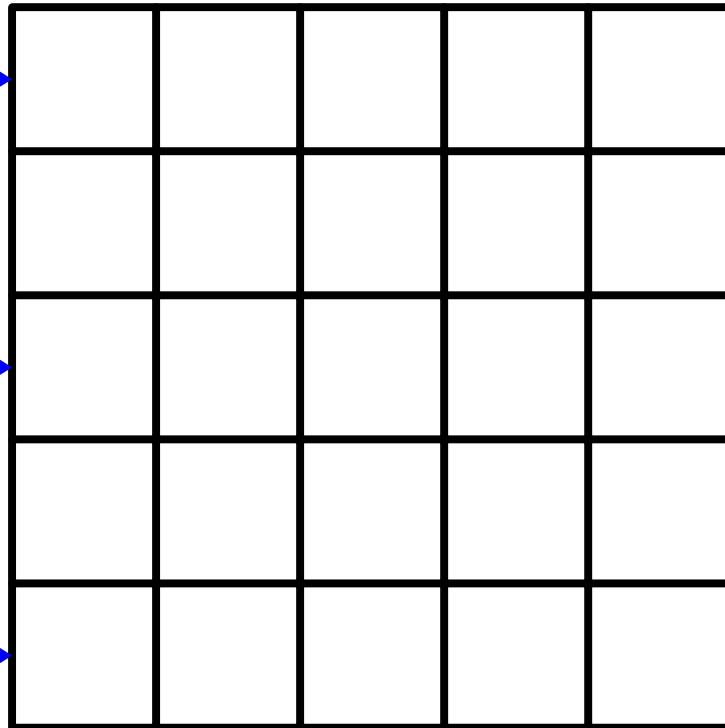
Type of
wheat

Thanks for former CS106B student Jose Francisco!

Try all words
that can go in
this row.

And here.

Same.



Same here.

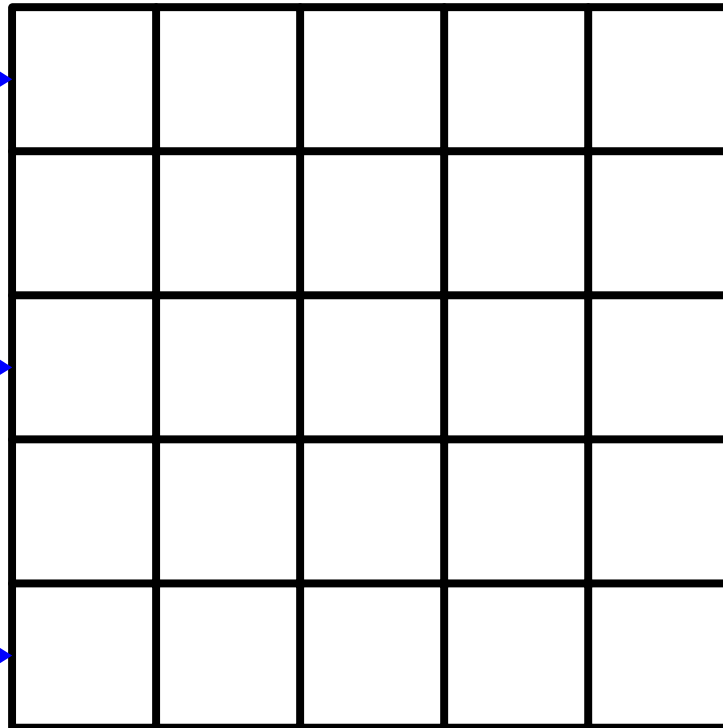
Here too.

Idea: Fill this in using recursive backtracking.

There are 8,636 words that can go in this row.

And here.

Same.



Same here.

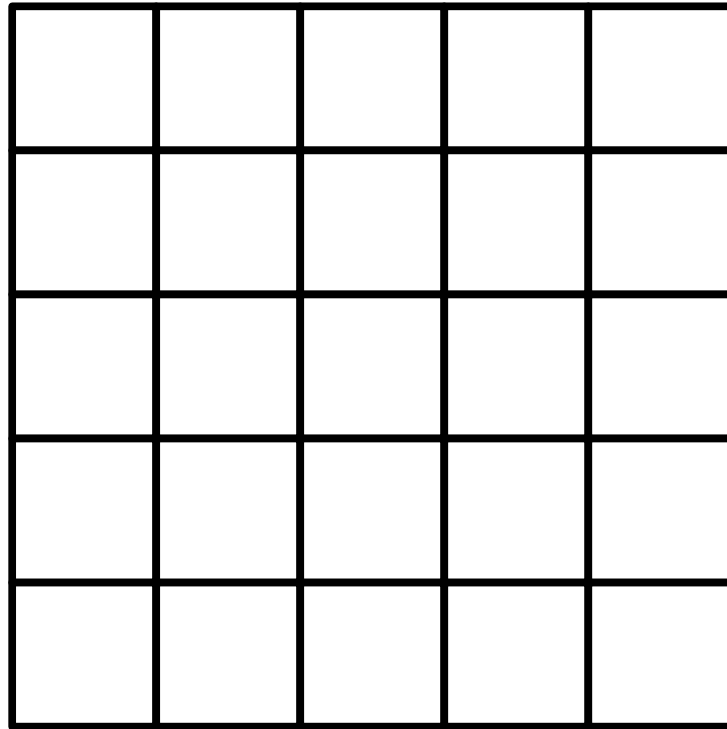
Here too.

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

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

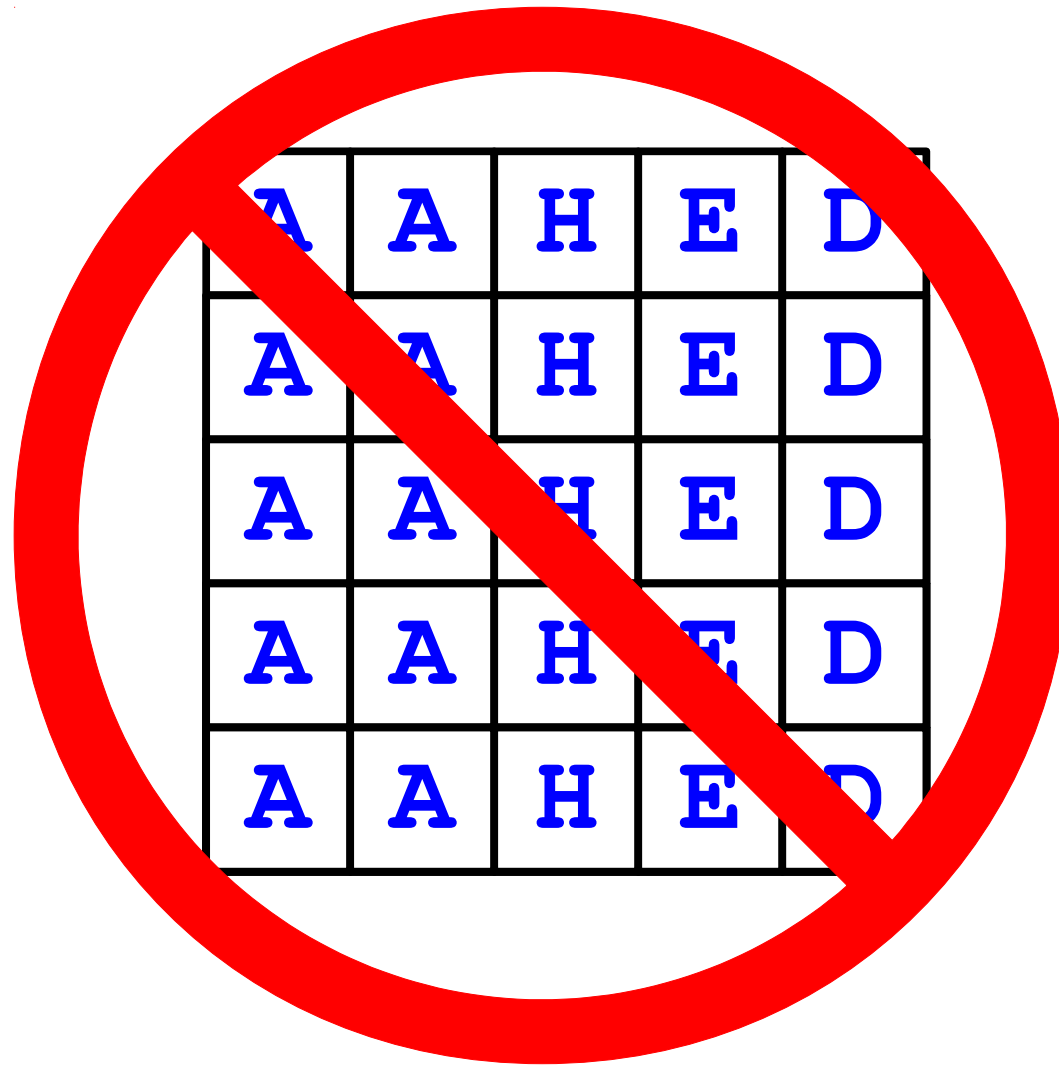
Generating Dense Crosswords

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

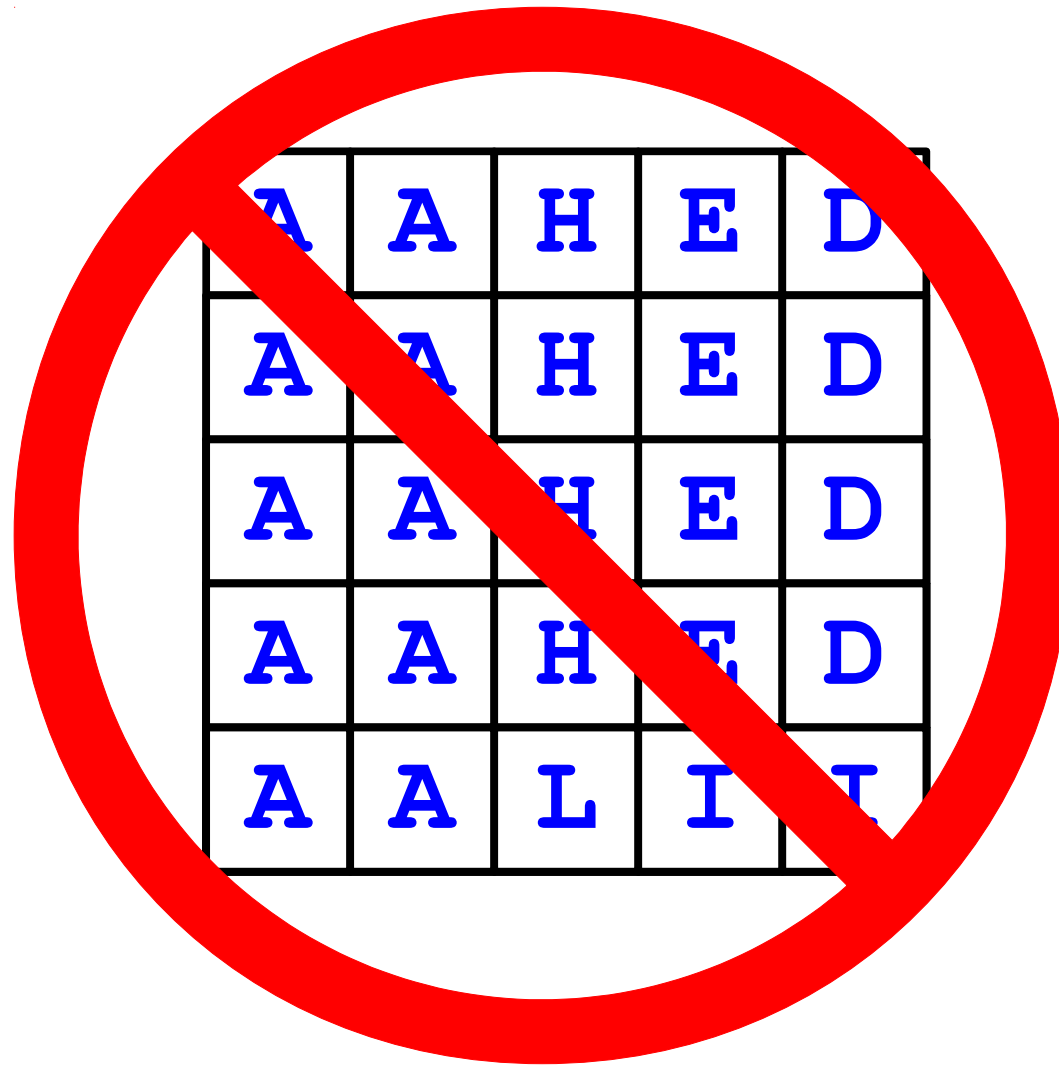
Generating Dense Crosswords



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

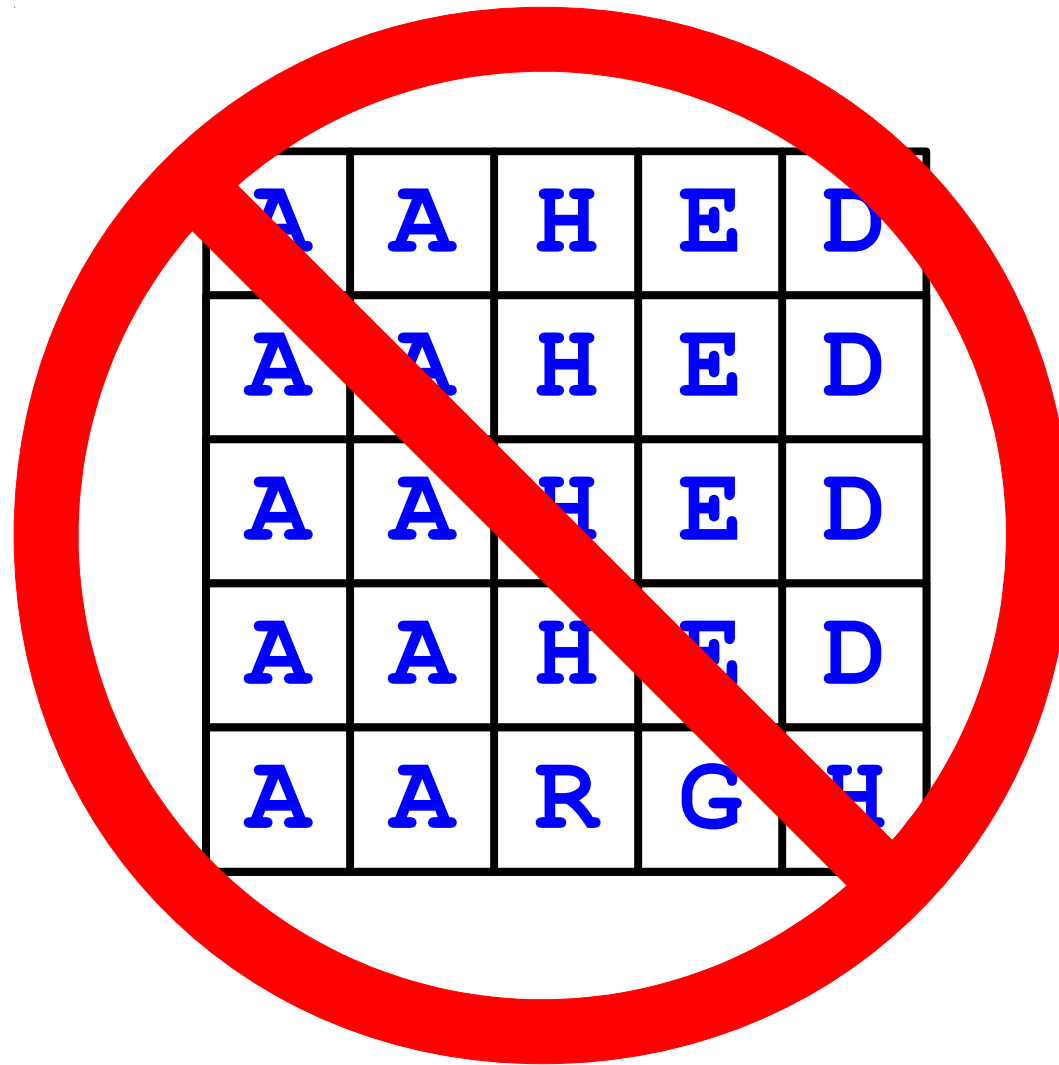
Generating Dense Crosswords



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

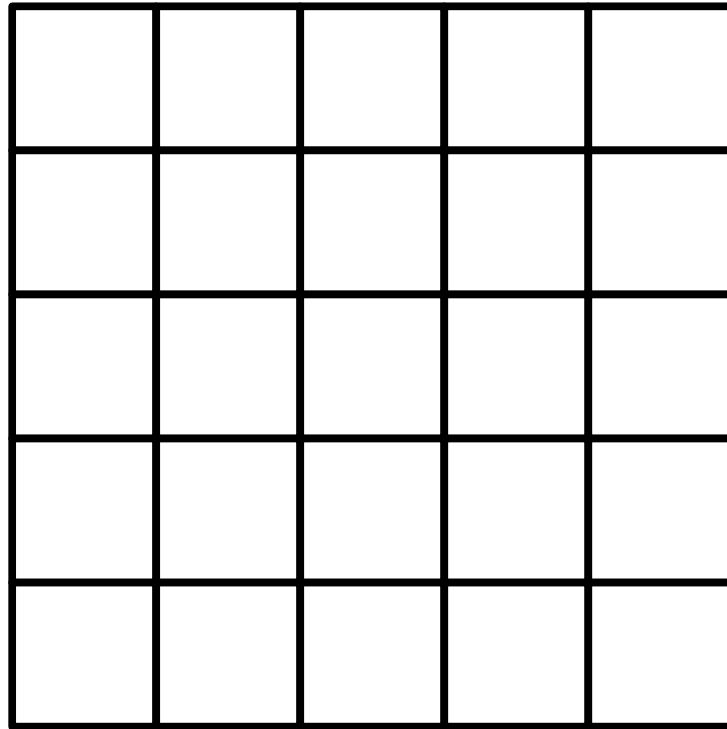


Generating Dense Crosswords

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

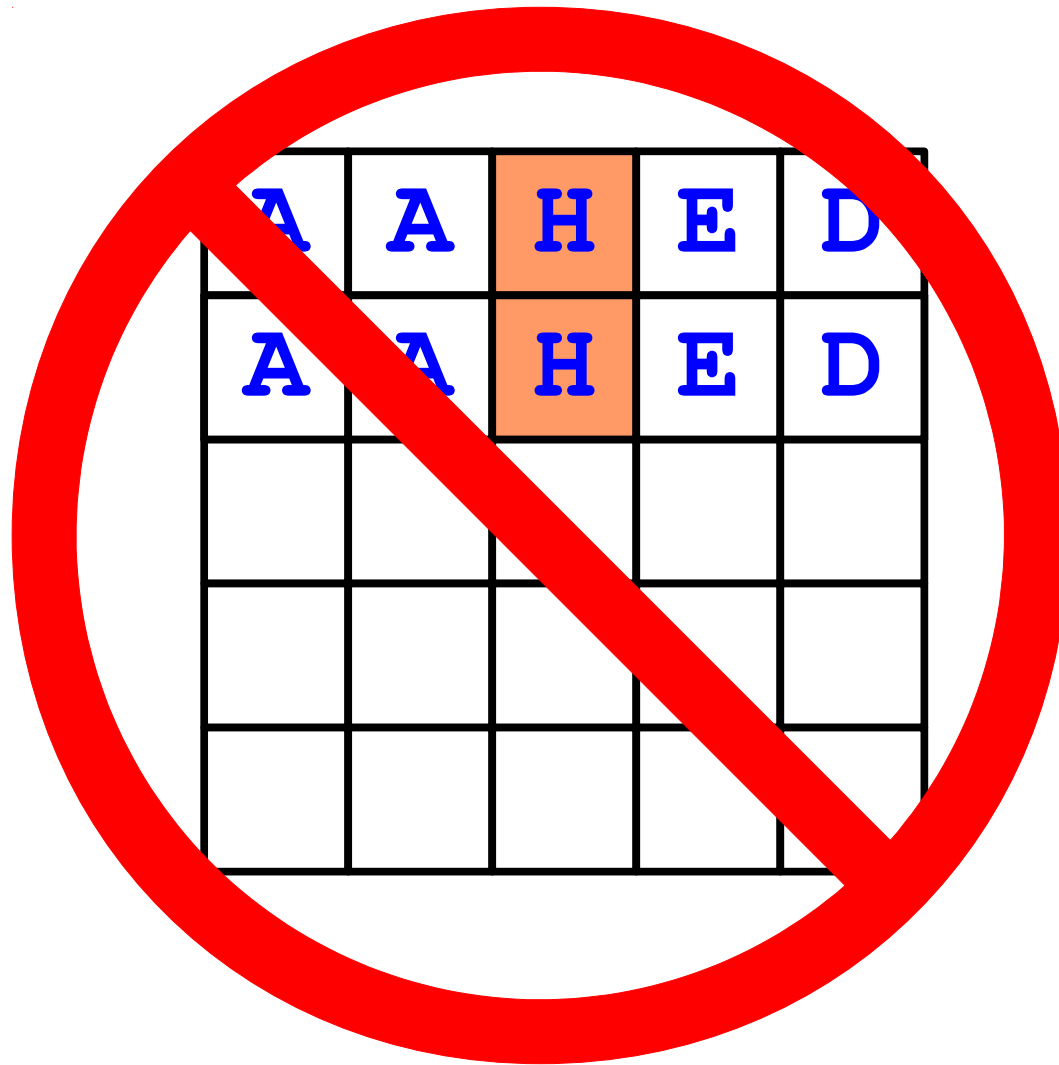
Generating Dense Crosswords

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

A	A	H	E	D
A	A	H	E	D

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.

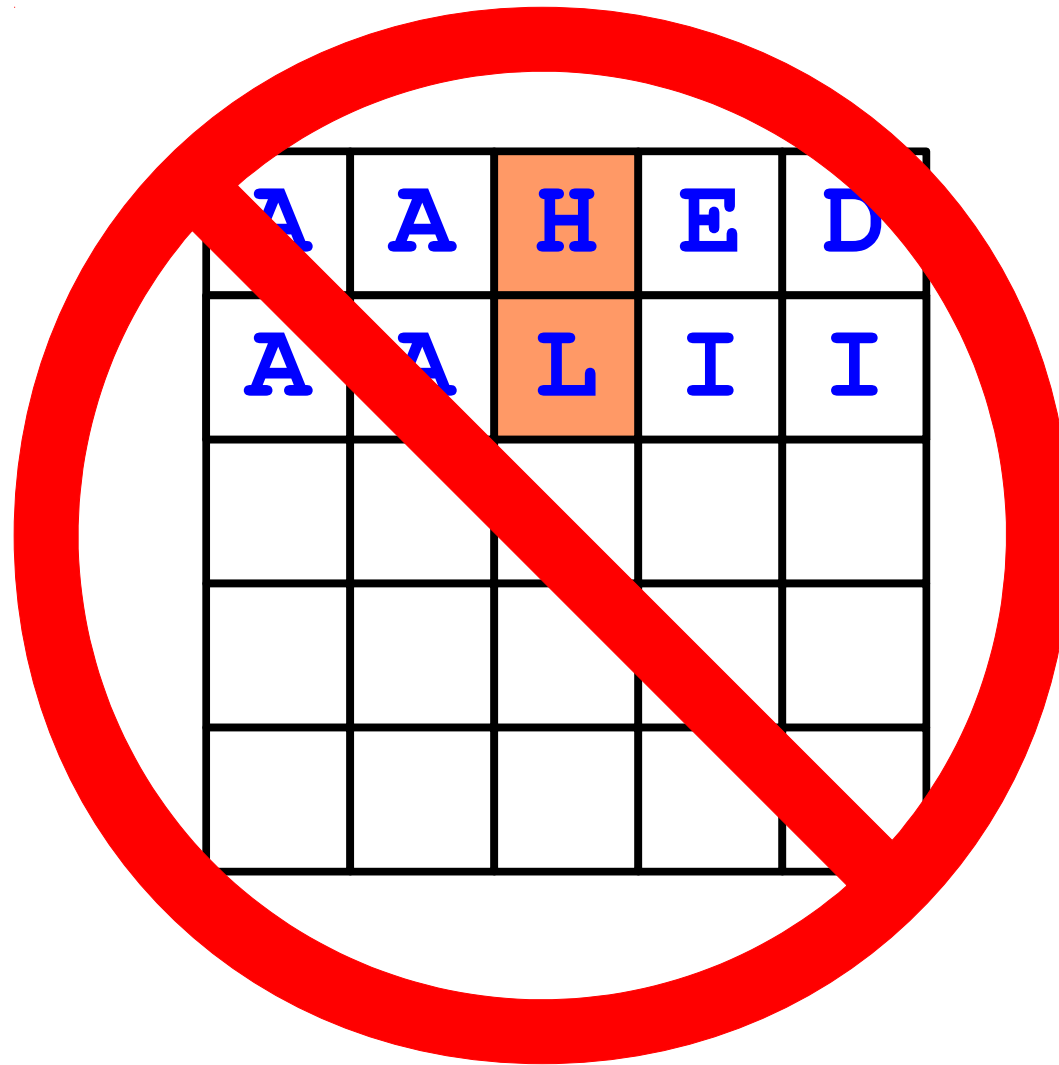
Generating Dense Crosswords

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

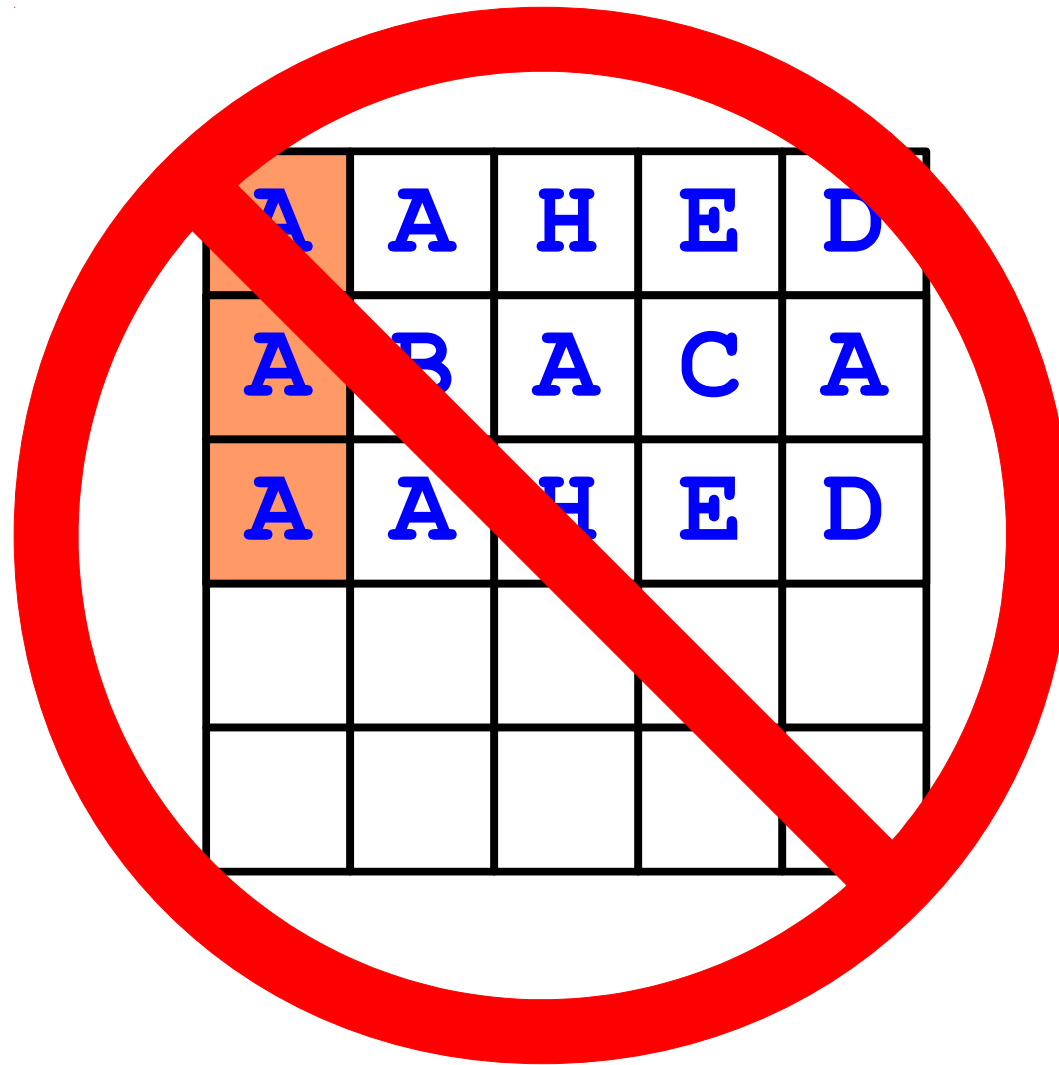
Generating Dense Crosswords

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



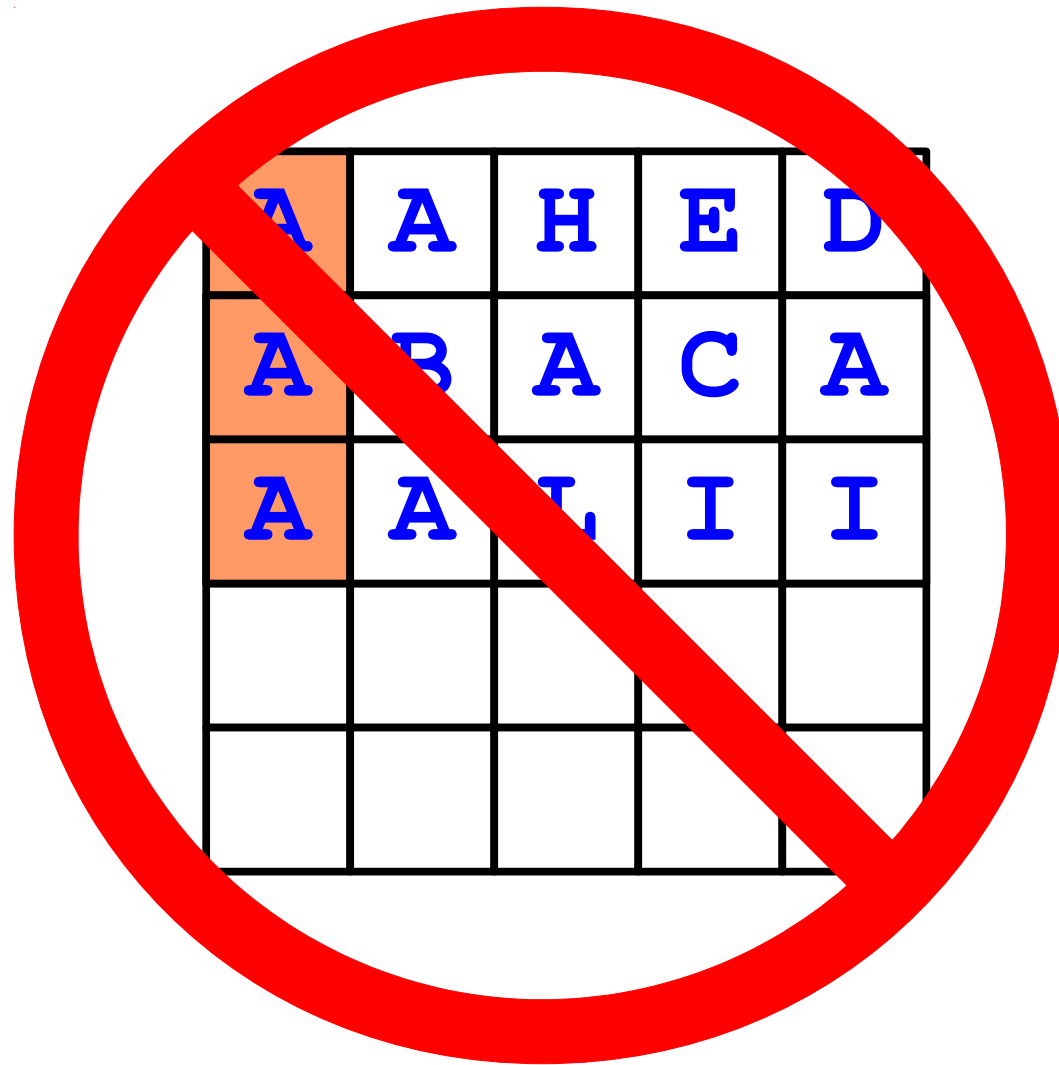
Generating Dense Crosswords

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

Generating Dense Crosswords

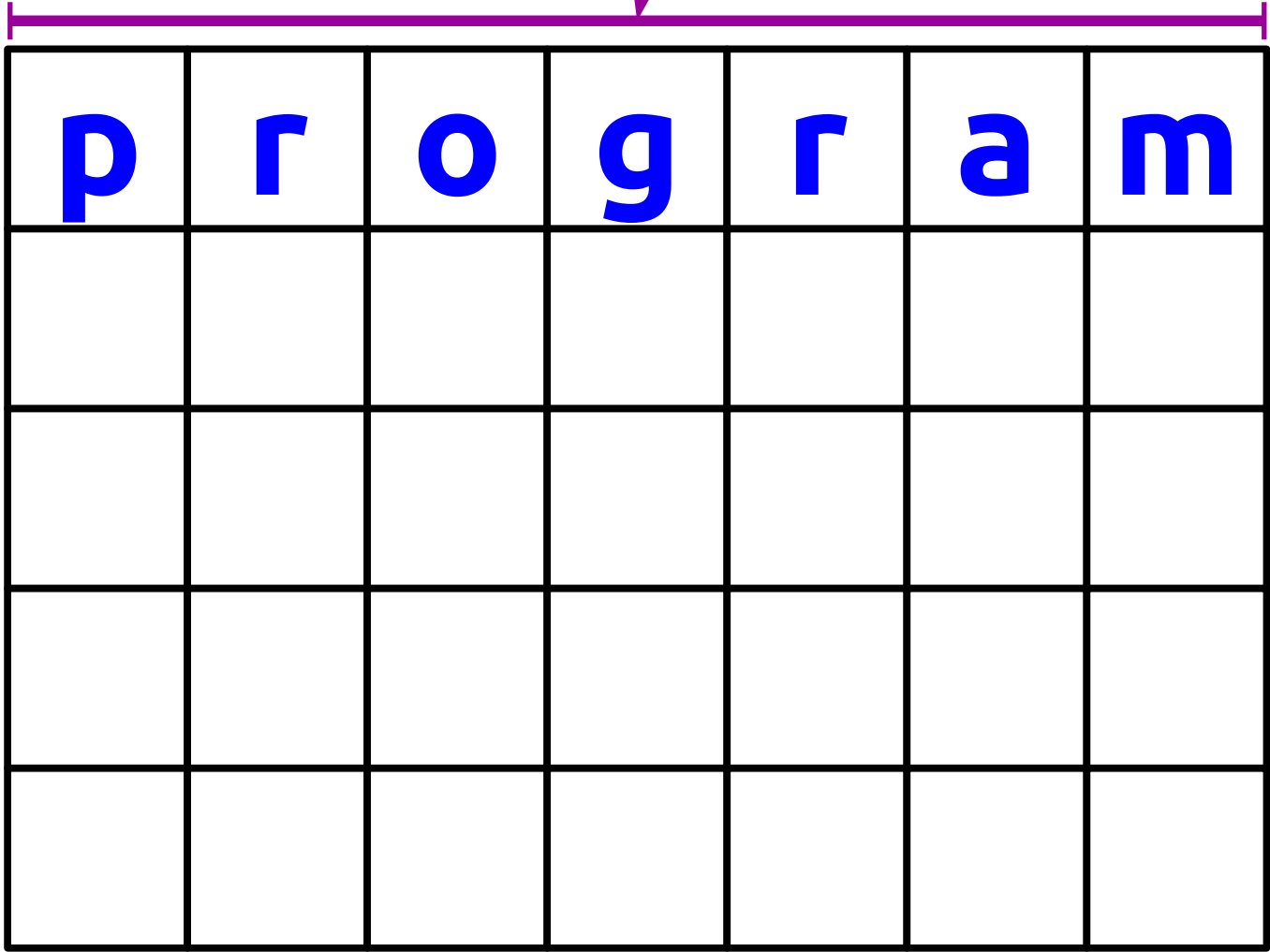
A	A	H	E	D
A	B	A	C	A
A	A	L	I	I

Generating Dense Crosswords



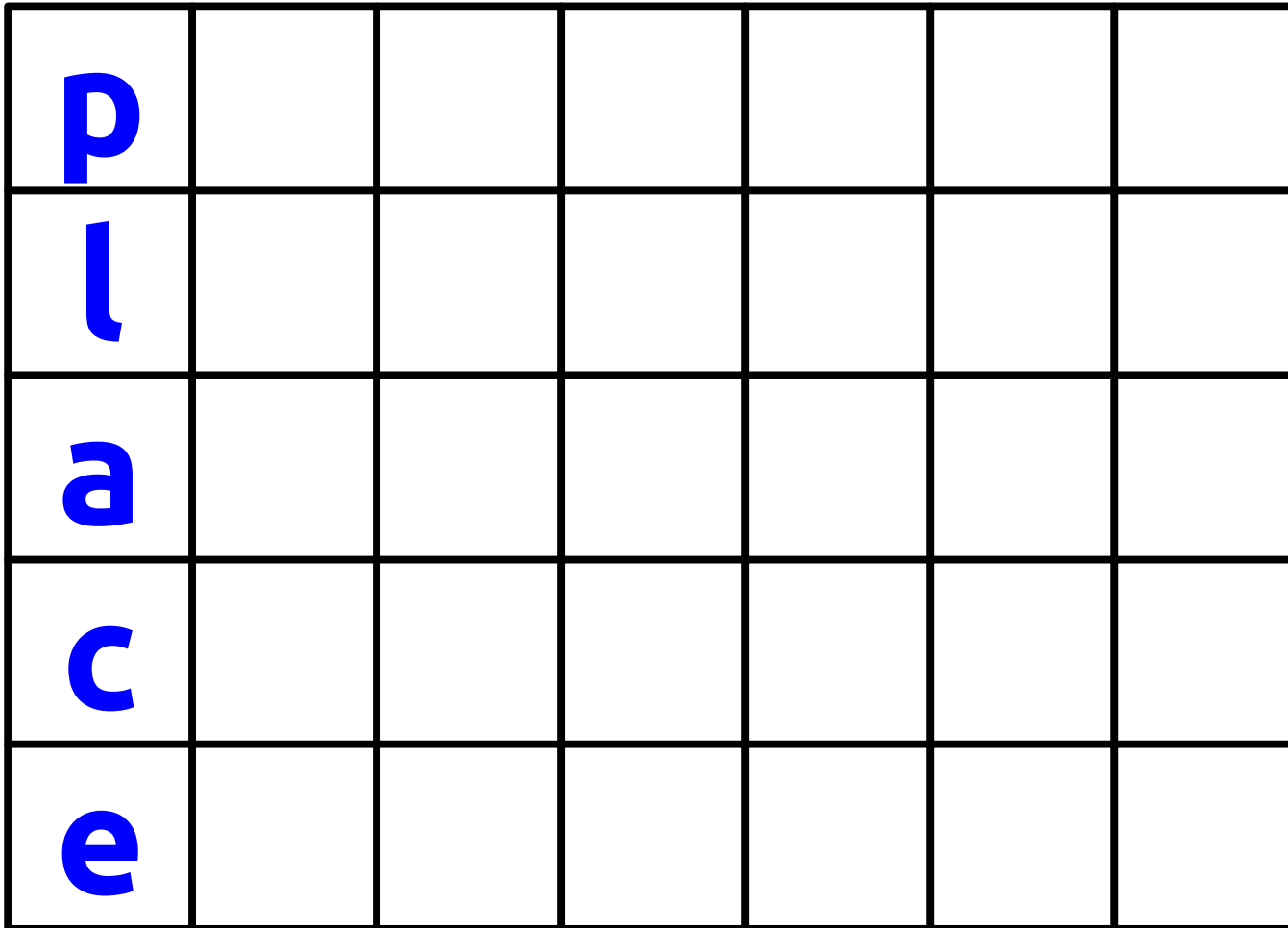
Let's Code it Up!

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						

Can we make a
dense crossword...

...that starts with
the first few rows
of this grid...

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

... given only these
words?

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;  
}
```

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.***
 - Aim to complete the debugging exercise and Doctors Without Orders by Monday.

Next Time

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