Thinking Recursively

Part V

Recap from Last Time

Recursive Backtracking

- In a recursive ***enumeration*** problem, we list all solutions to a problem.
- In a recursive ***optimization*** problem, we find the best solution to a problem.
- In a recursive ***backtracking*** problem, we see whether there even is a solution.

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

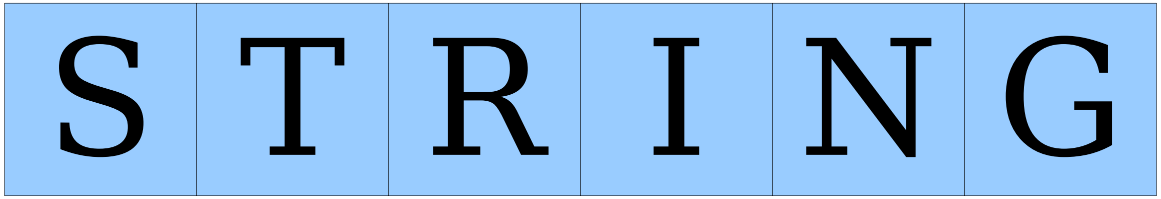
One Solution

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

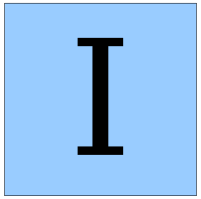
One Solution

S	I	N

One Solution

I	N

One Solution

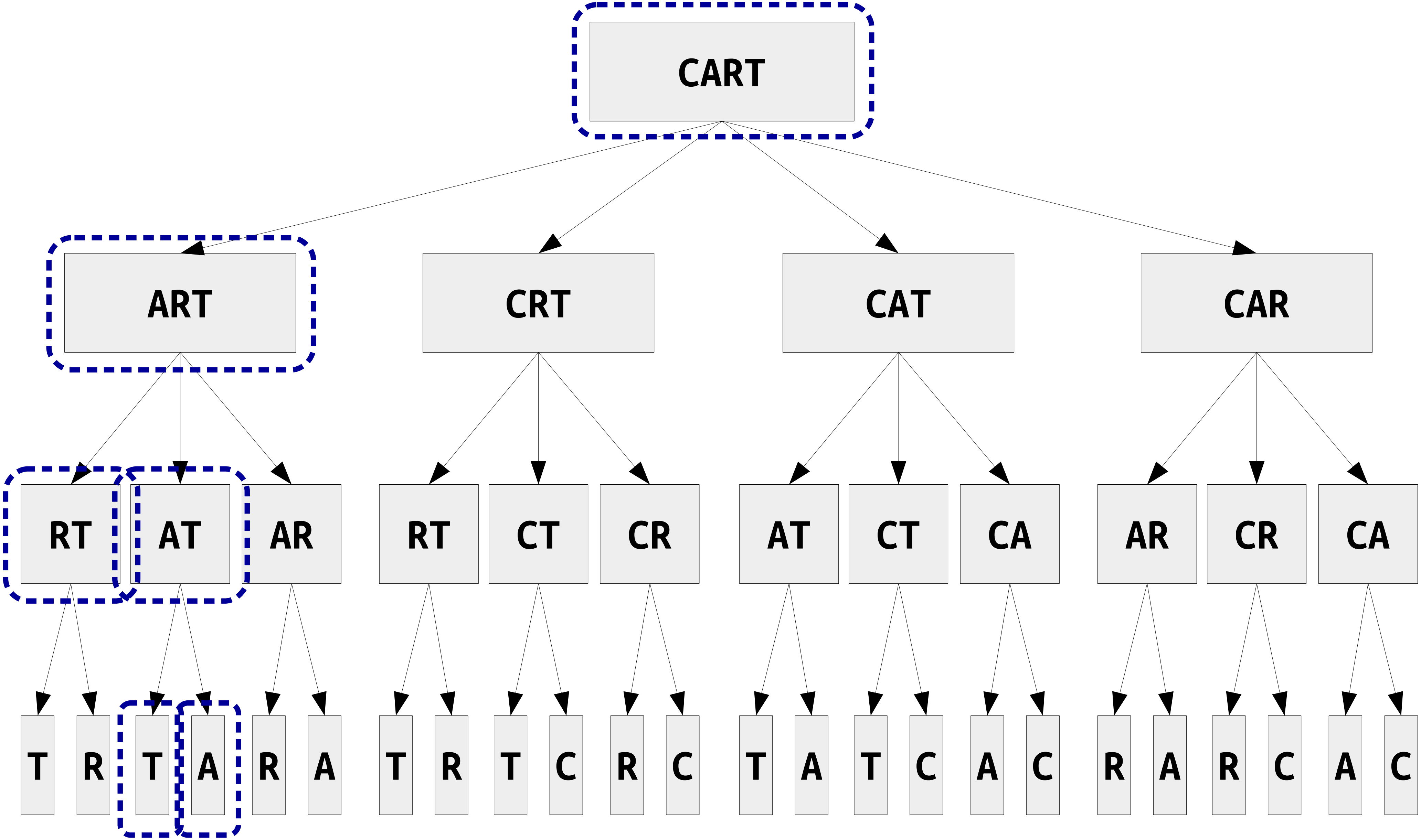


I

New Stuff!

Our Solution, In Action

The Incredible Shrinking Word



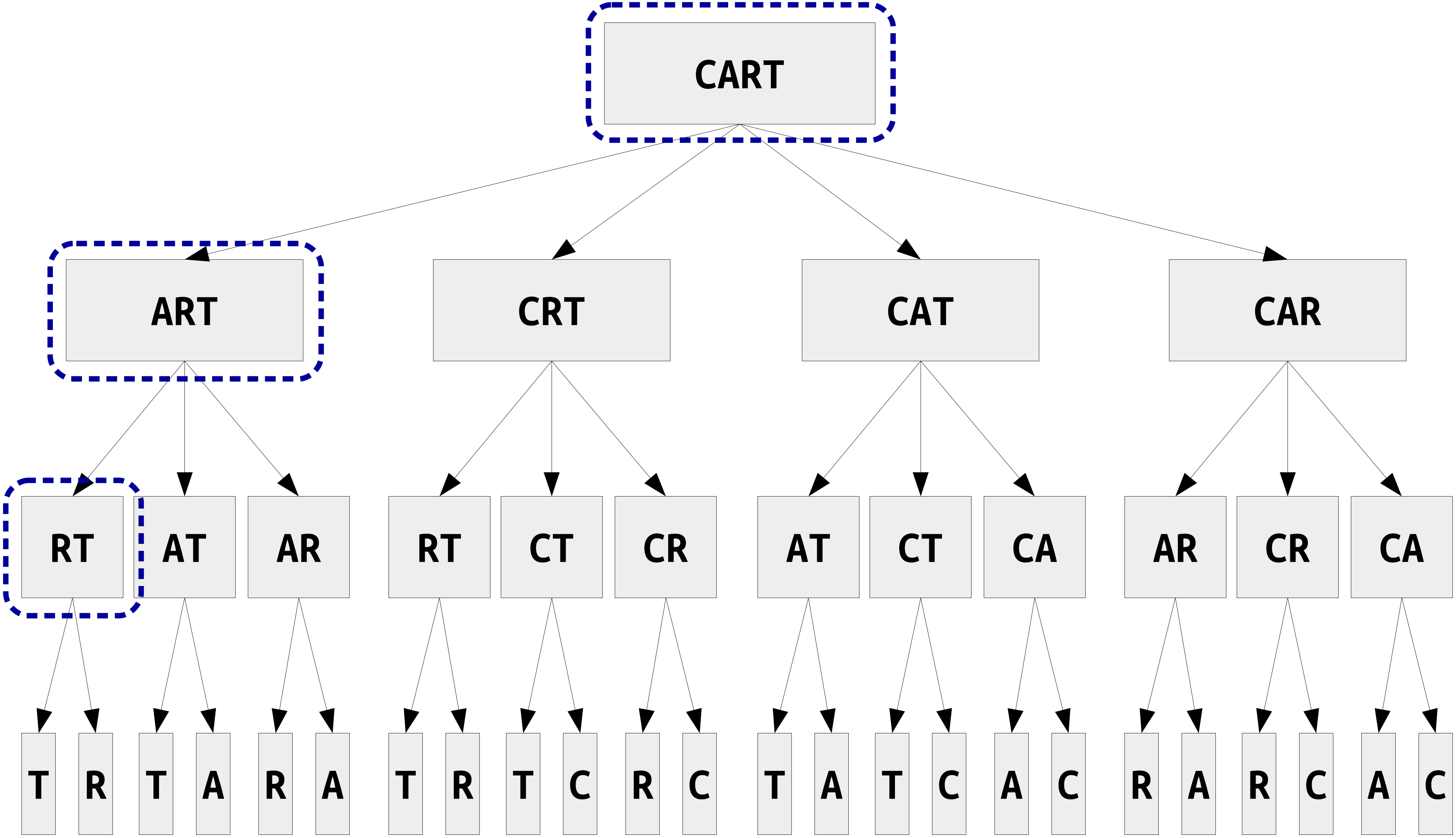
```
bool isShrinkable(const string& word, const Lexicon& english) {
    if (!english.contains(word)) return false;
    if (word.length() == 1) return true;

    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 isShrinkable(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 isShrinkable(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 isShrinkable(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



CART

ART CRT CAT CAR

RT AT AR RT CT CR AT CT CA AR CR CA

T R T A R A T R T C R C T A T C A C R A R C A C

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;
}
```

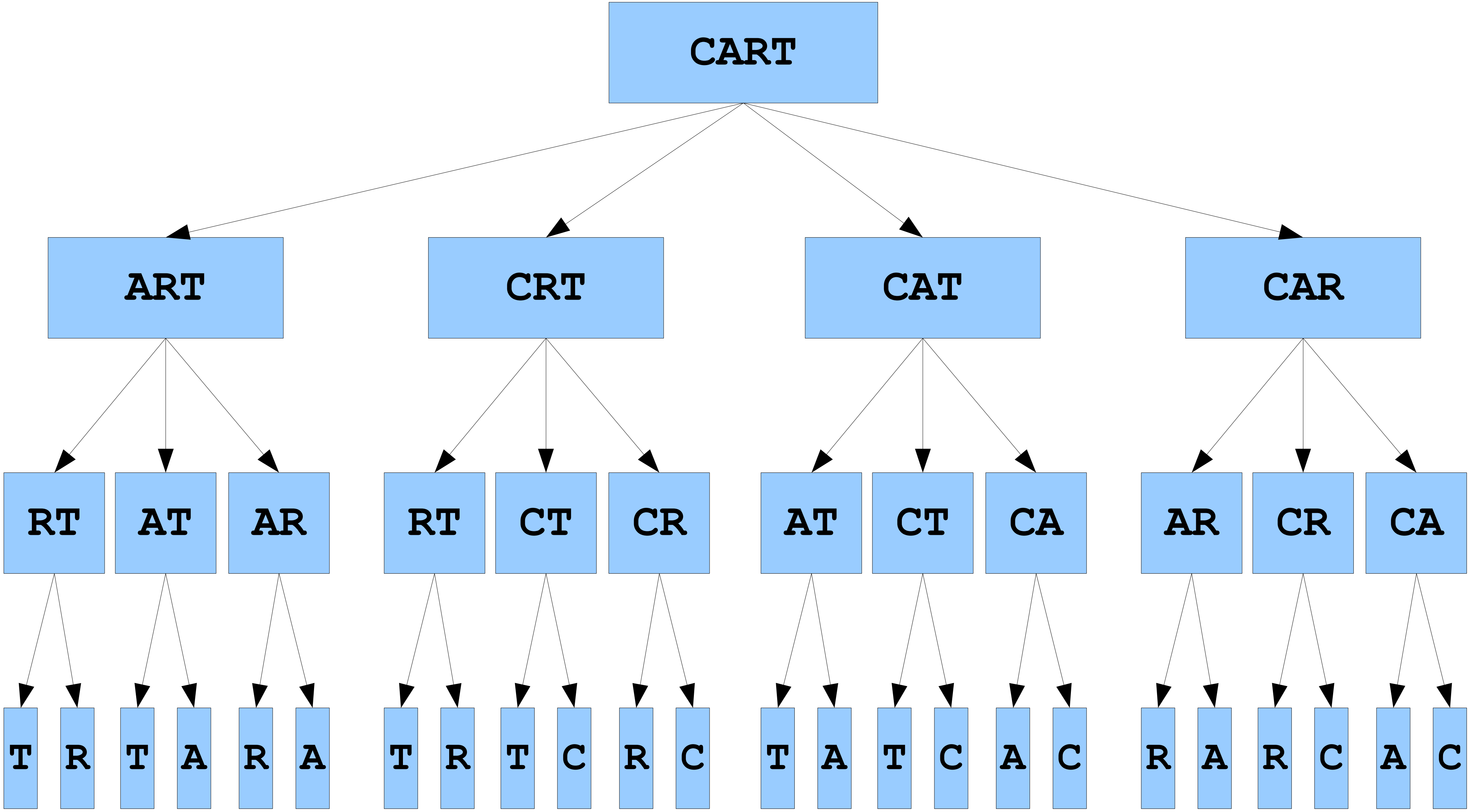
Note that *if* the recursive call succeeds, *then* we return success. If it doesn't succeed, that doesn't mean we've failed – it just means we need to try out the next option.

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



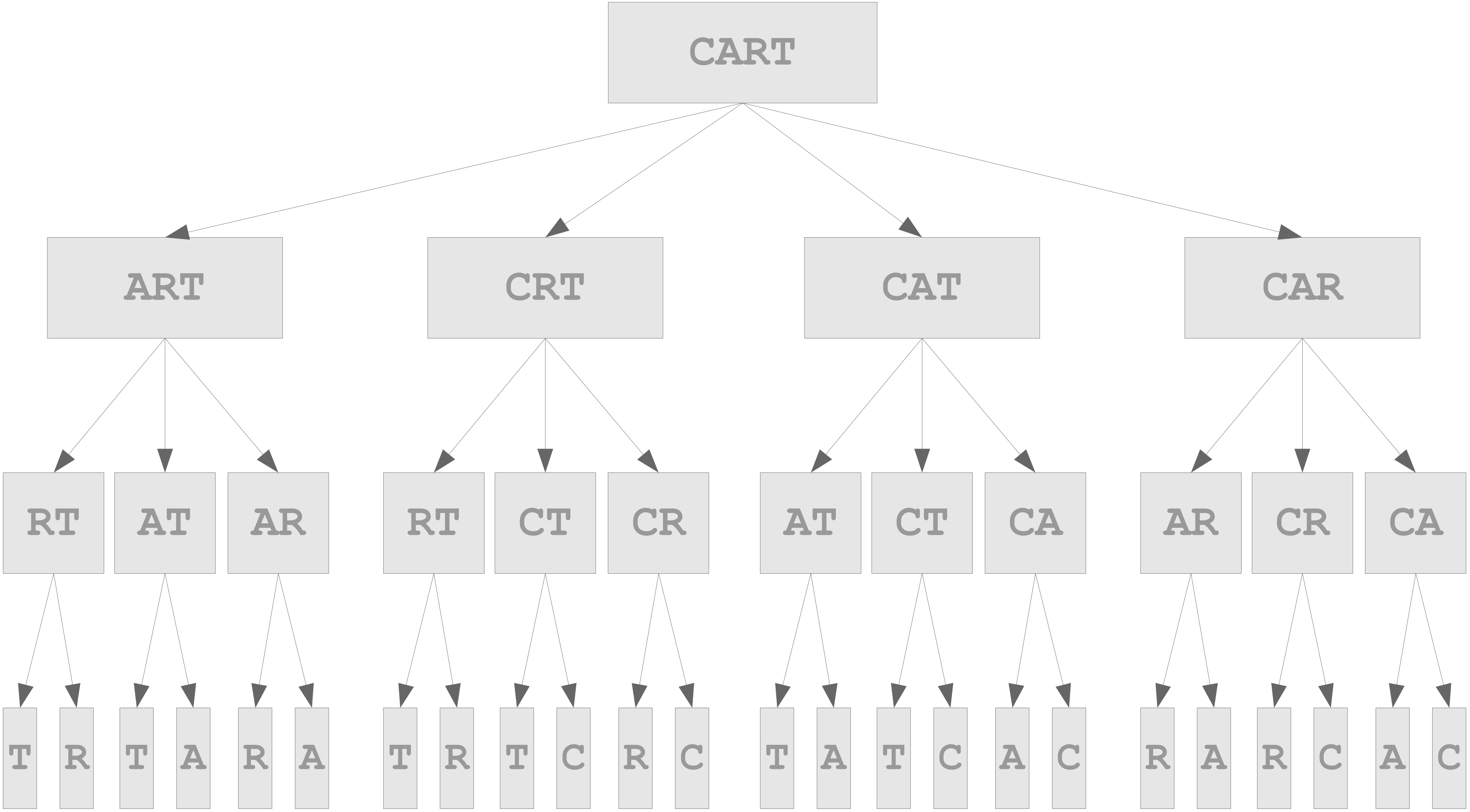
CART

ART CRT CAT CAR

RT AT AR RT CT CR AT CT CA AR CR CA

T R T A R A T R T C R C T A T C A C R A R C A C

Generating the Answer



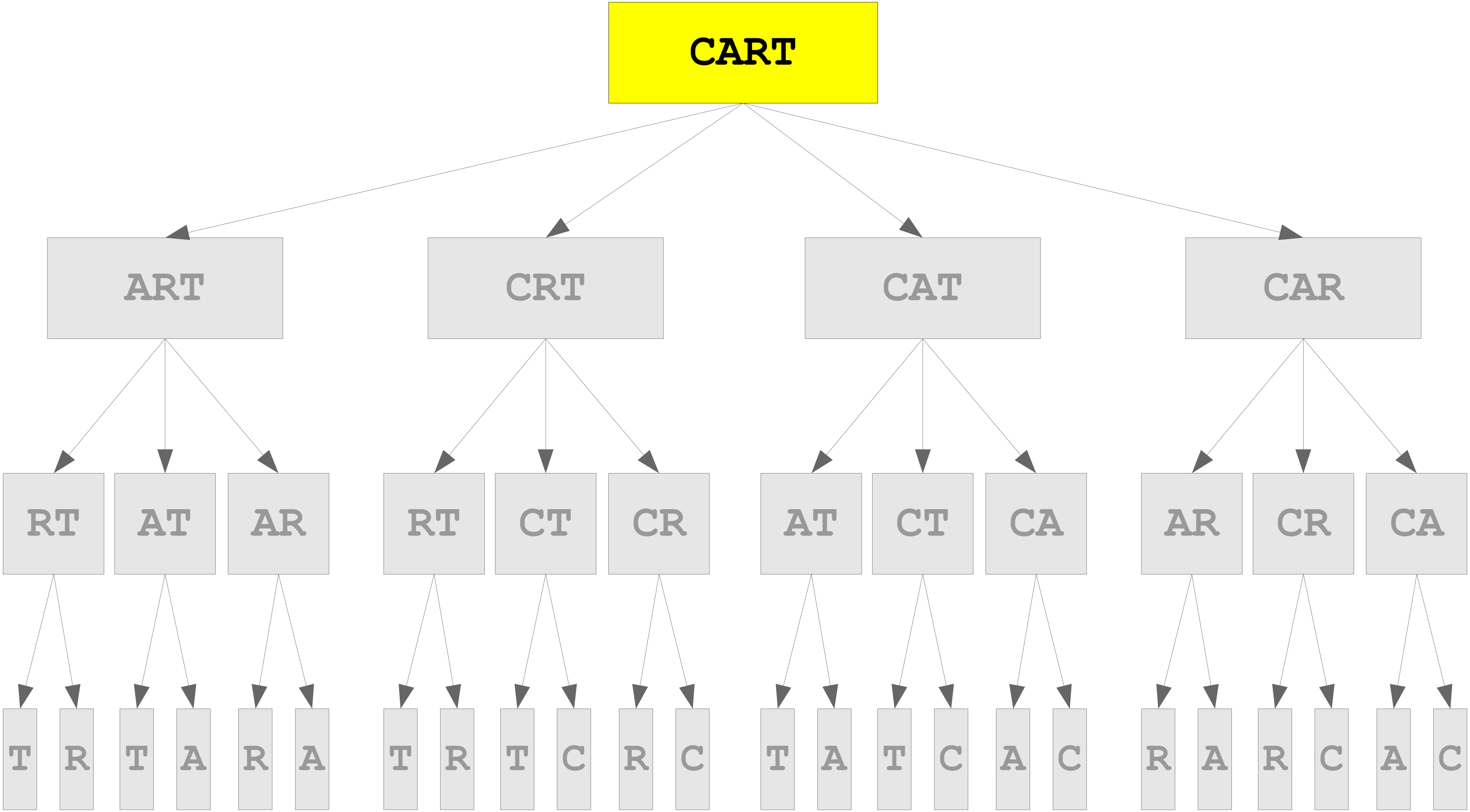
CART

ART CRT CAT CAR

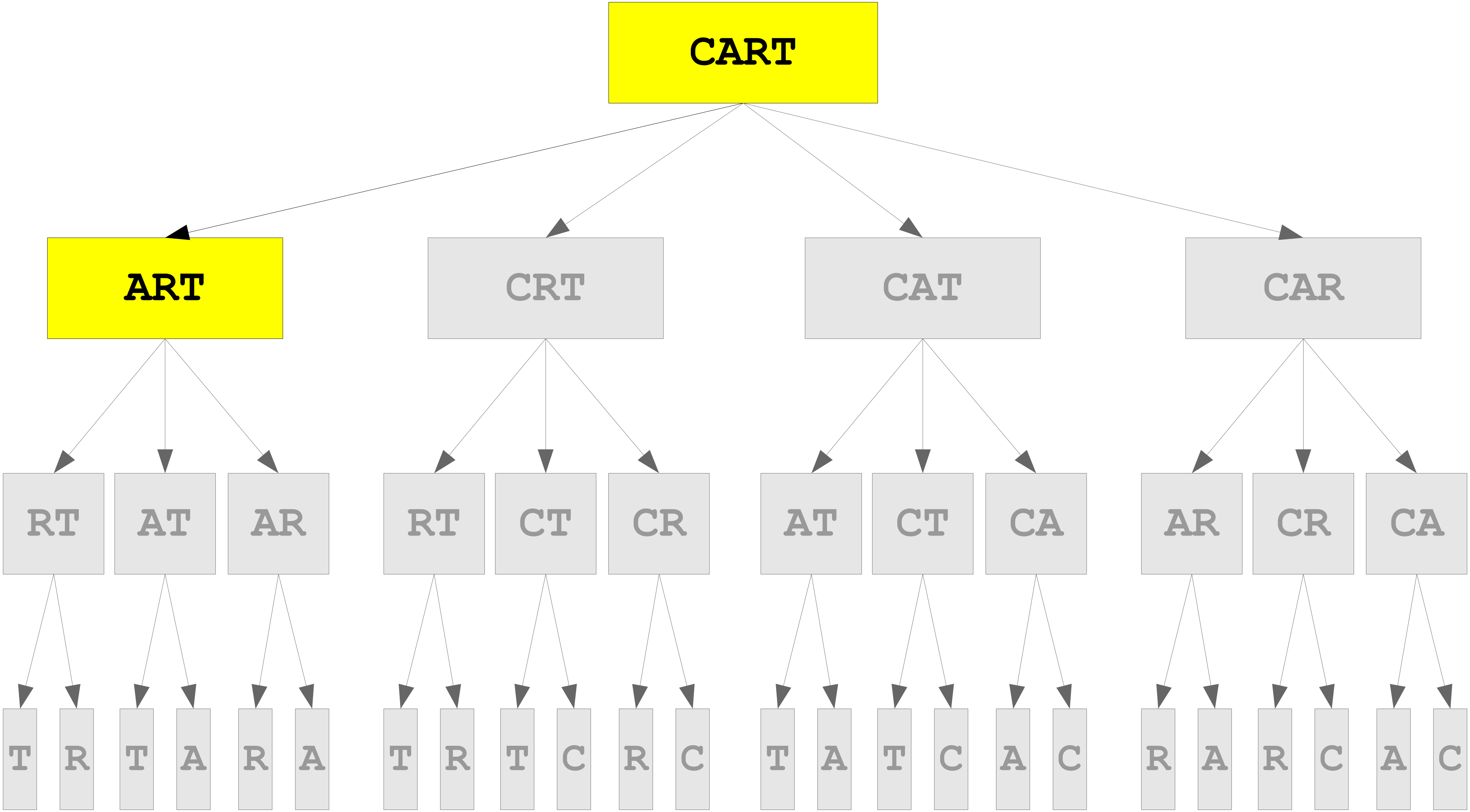
RT AT AR RT CT CR AT CT CA AR CR CA

T R T A R A T R T C R C T A T C A C R A R C A C

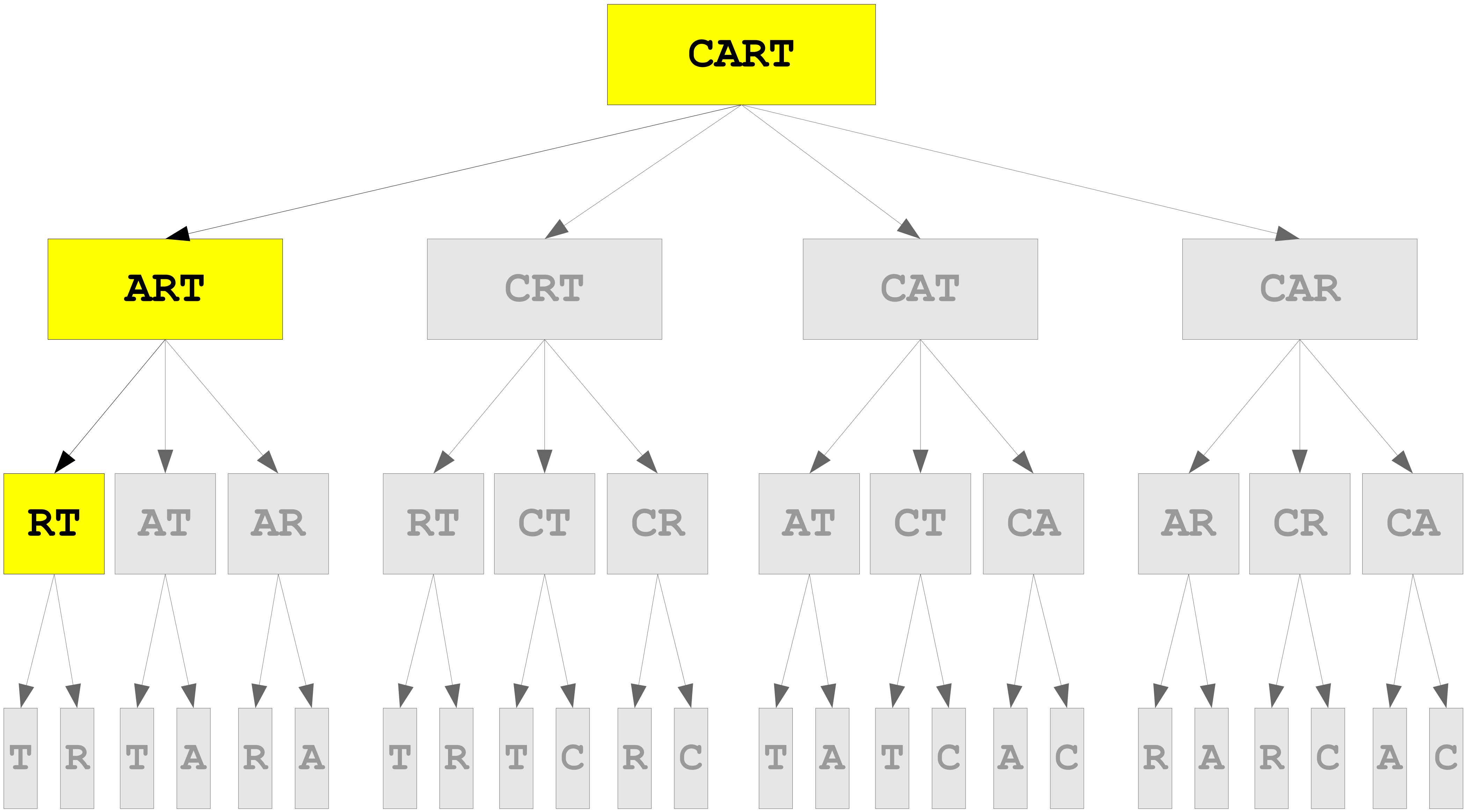
Generating the Answer



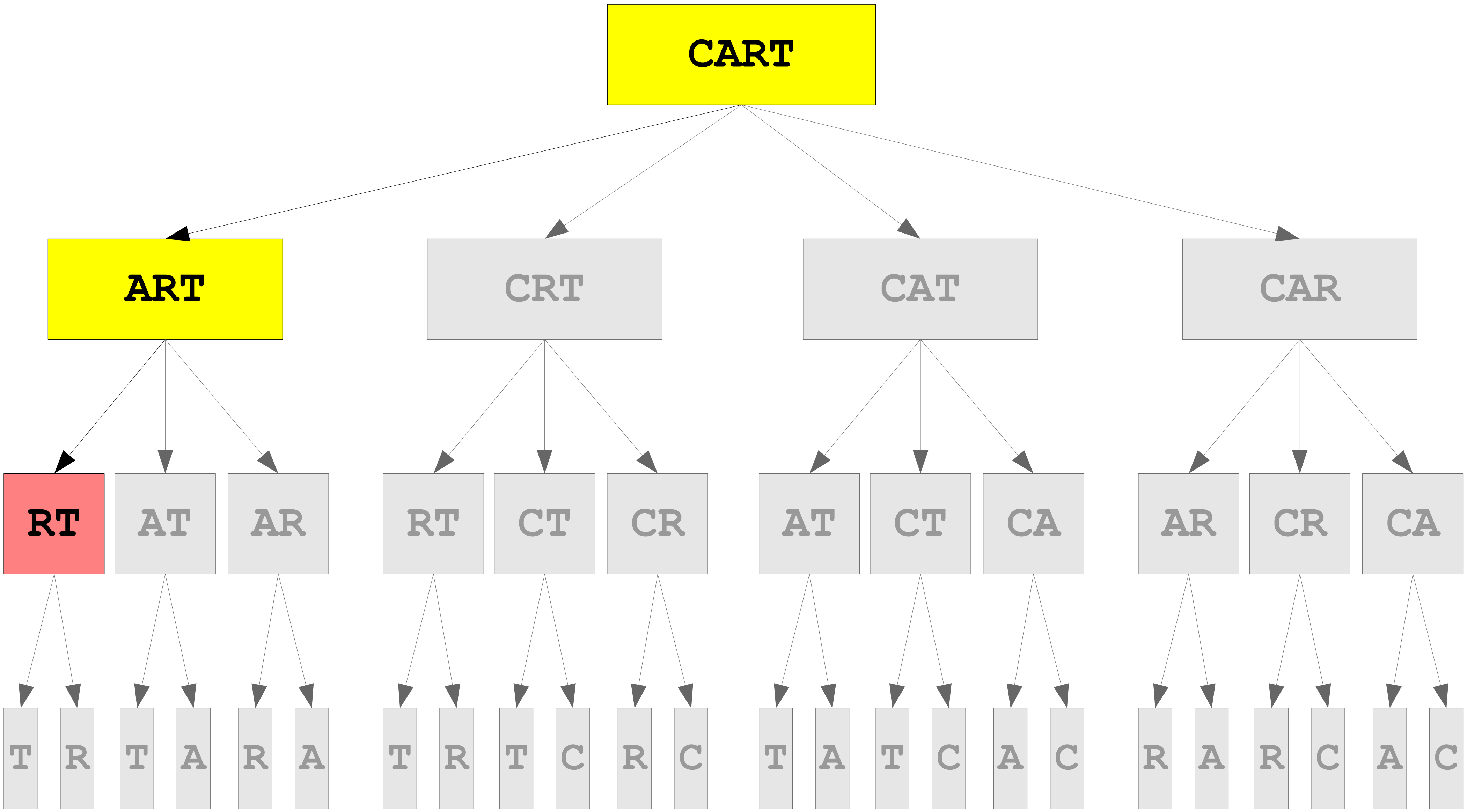
Generating the Answer



Generating the Answer



Generating the Answer



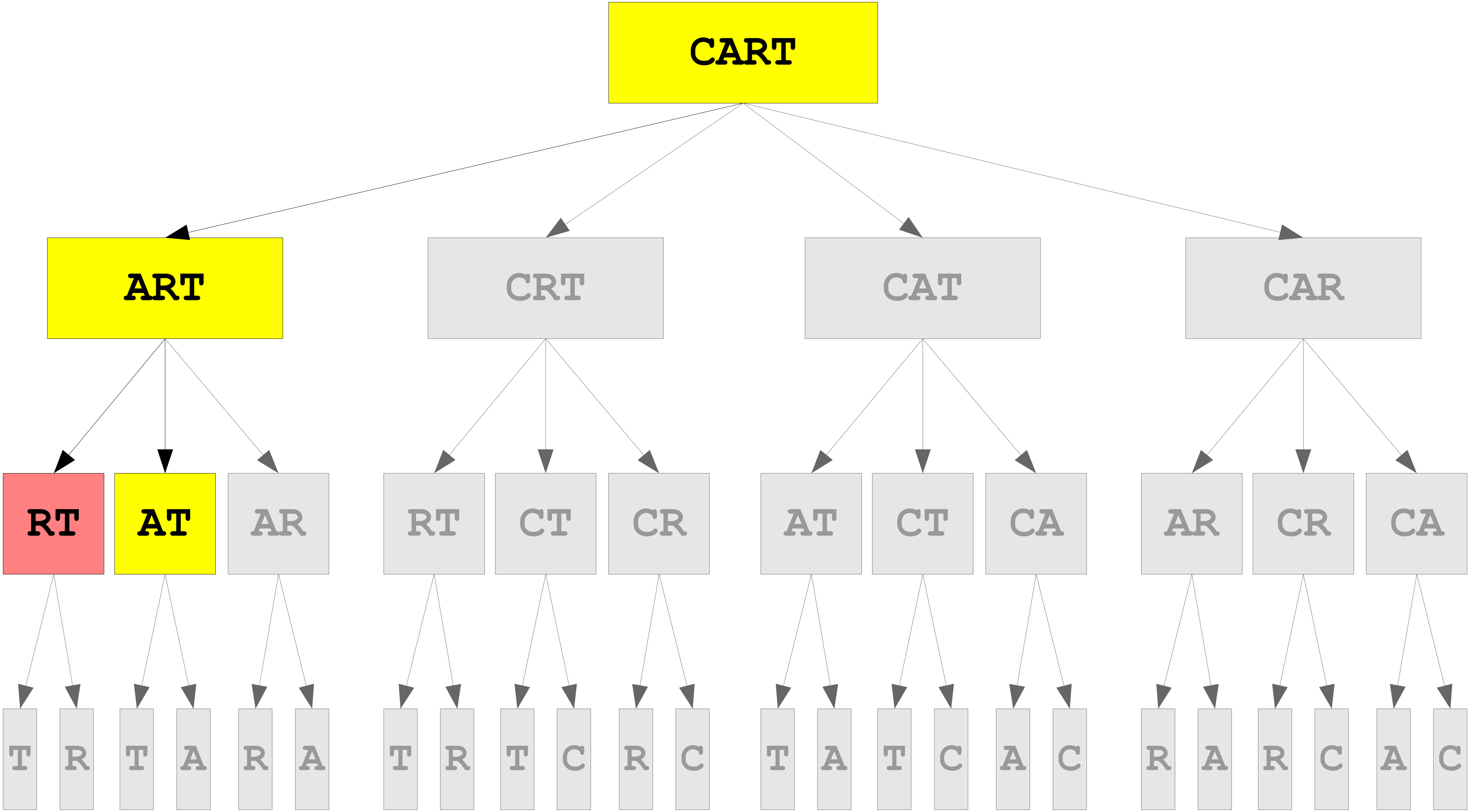
CART

ART CRT CAT CAR

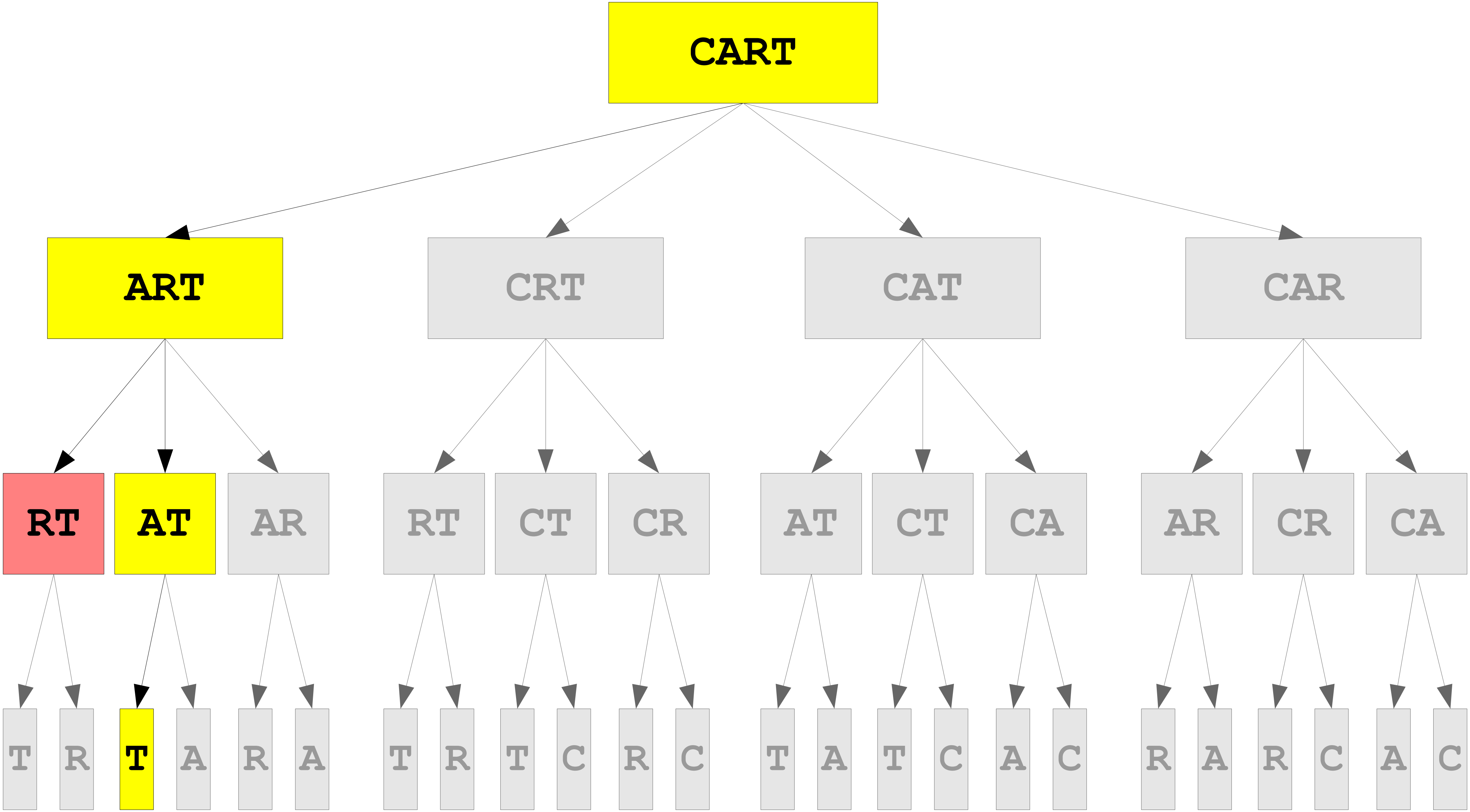
RT AT AR RT CT CR AT CT CA AR CR CA

T R T A R A T R T C R C T A T C A C R A R C A C

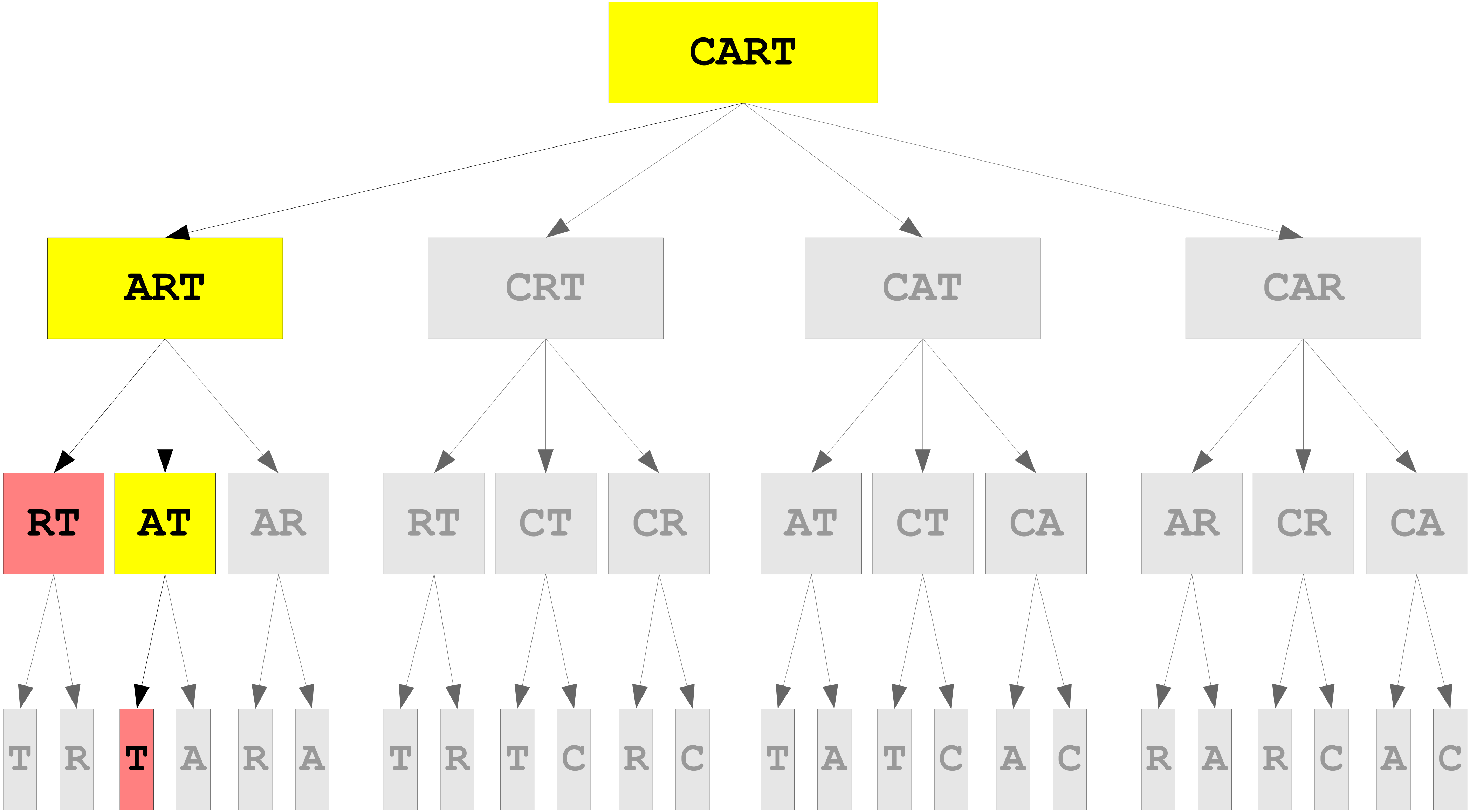
Generating the Answer



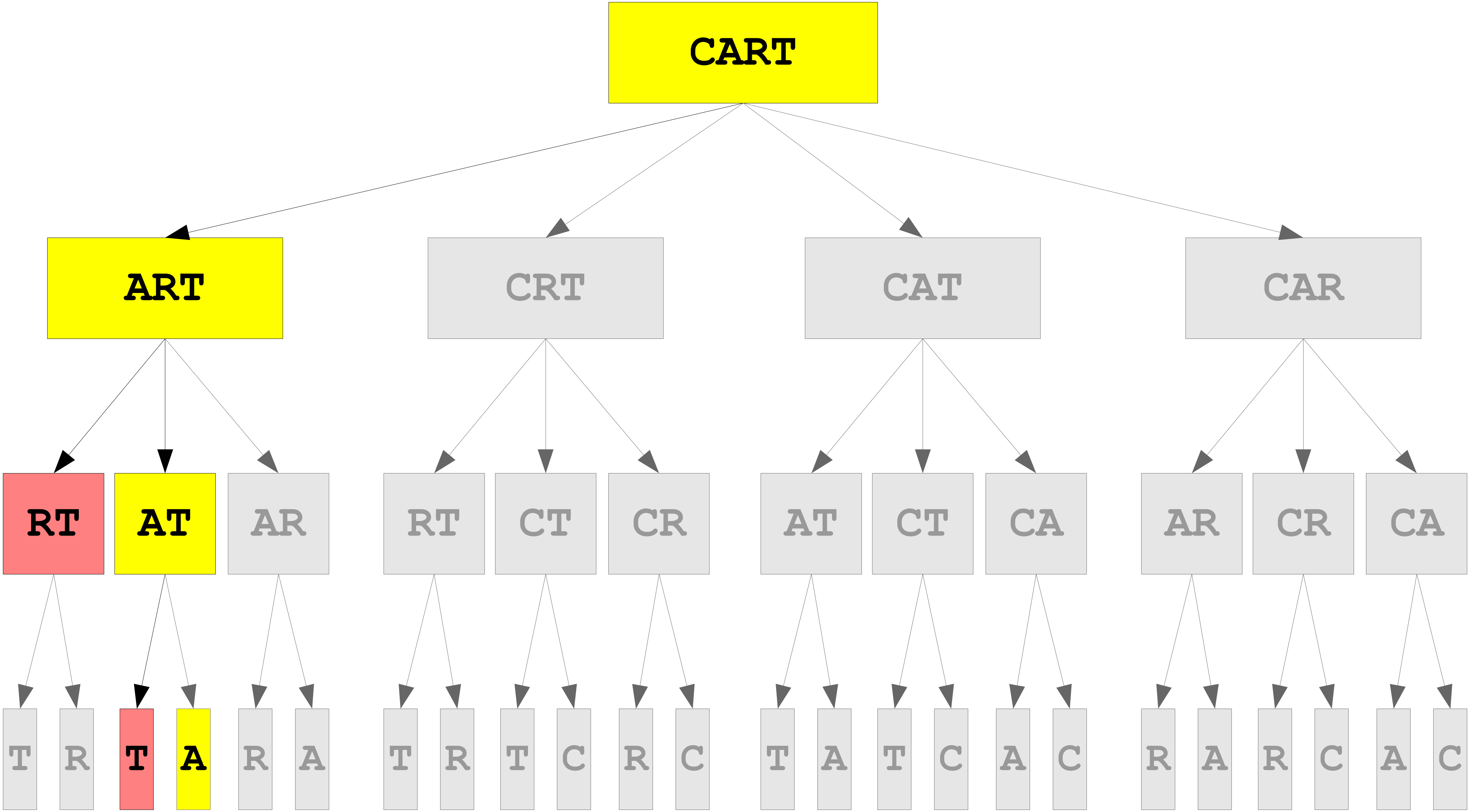
Generating the Answer



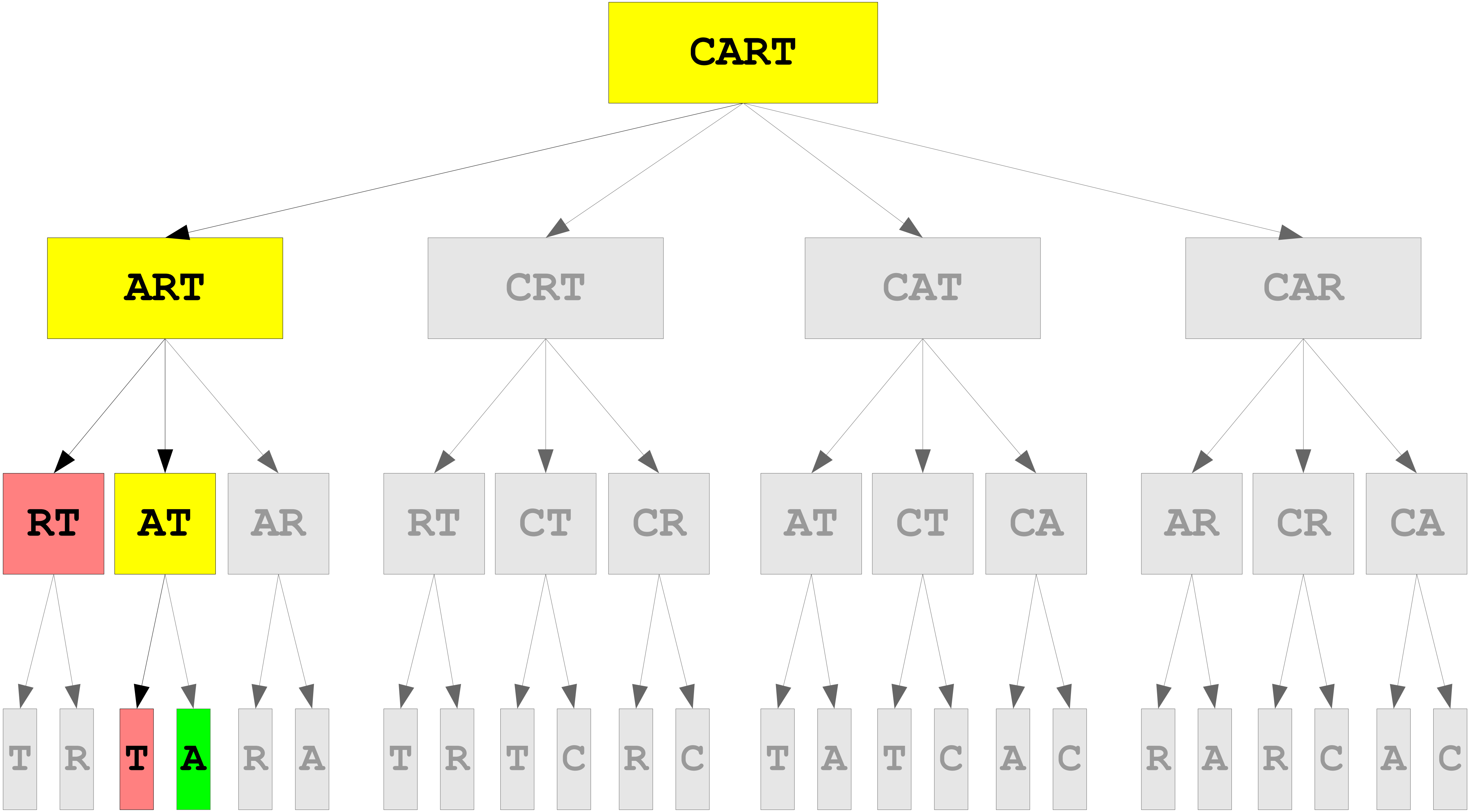
Generating the Answer



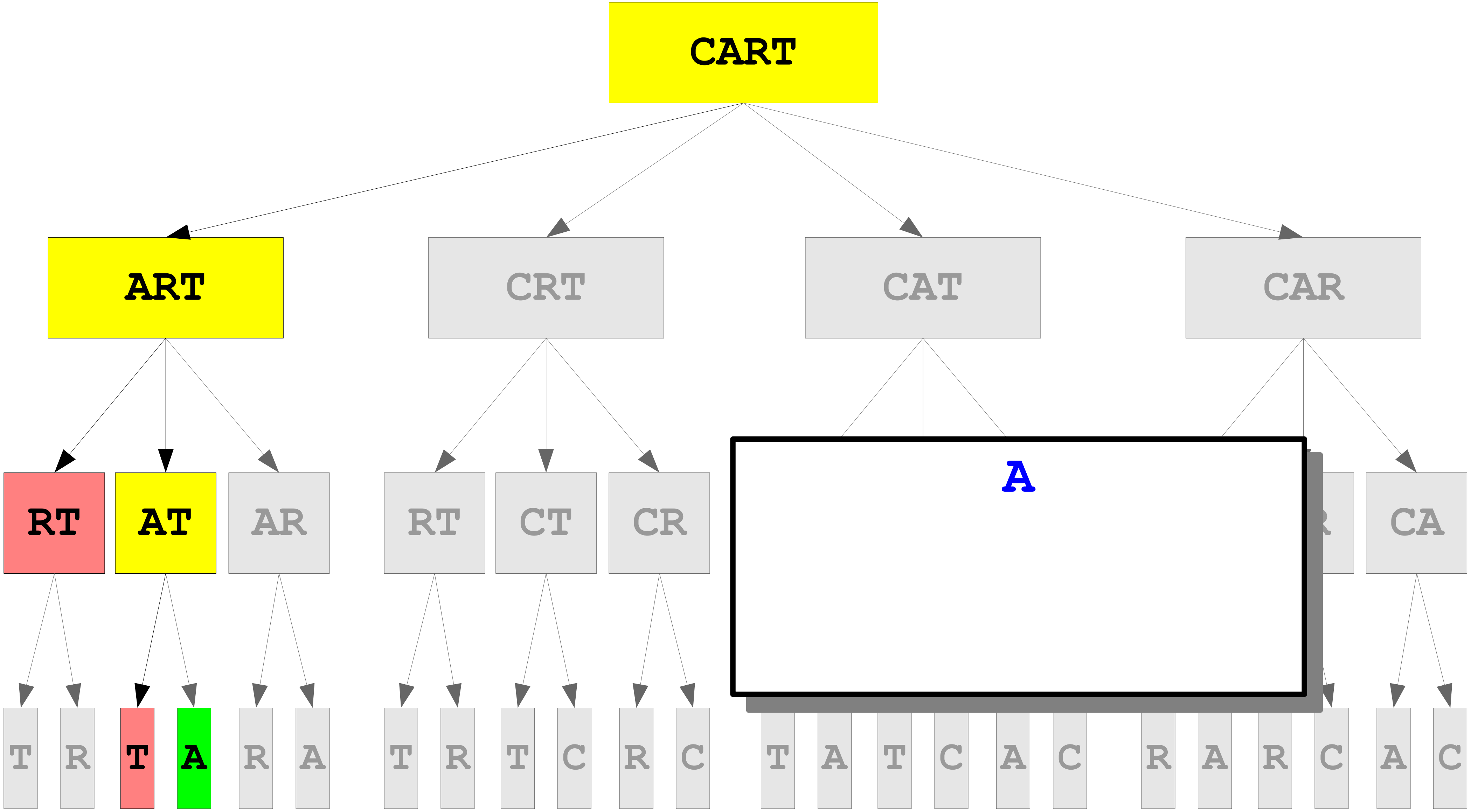
Generating the Answer



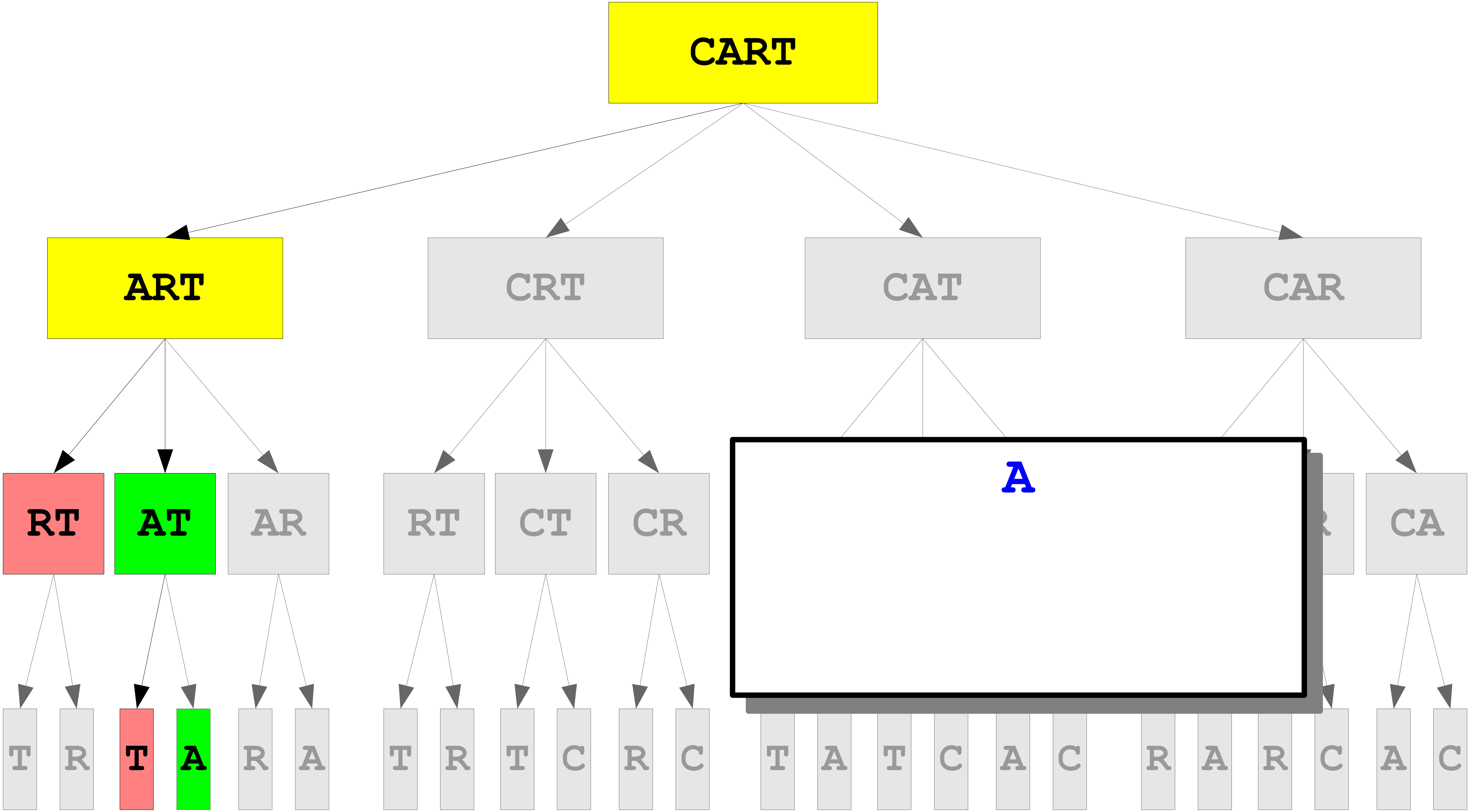
Generating the Answer



Generating the Answer



Generating the Answer



CART

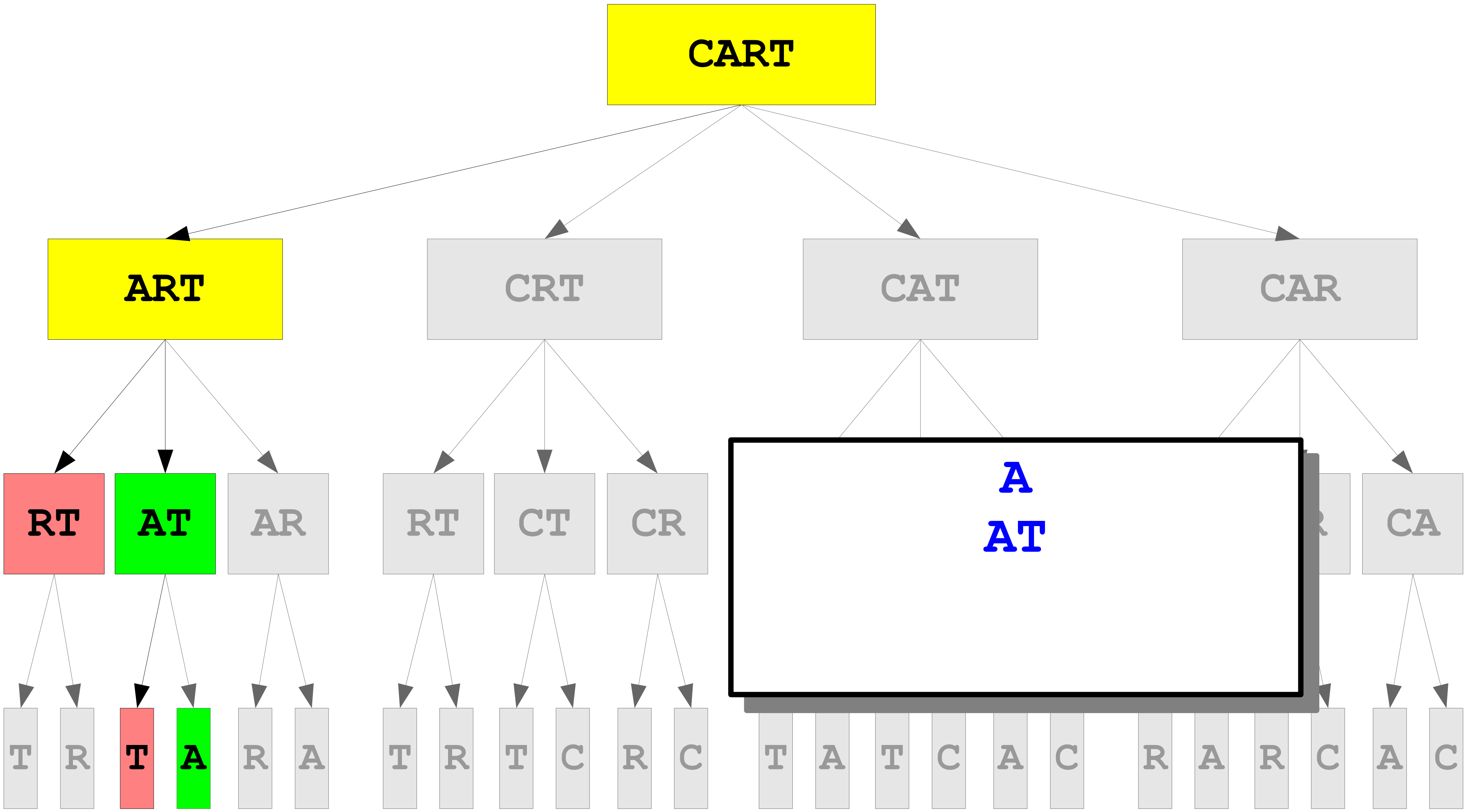
ART CRT CAT CAR

RT AT AR RT CT CR R CA

T R T A R A T R T C R C T A T C A C R A R C A C

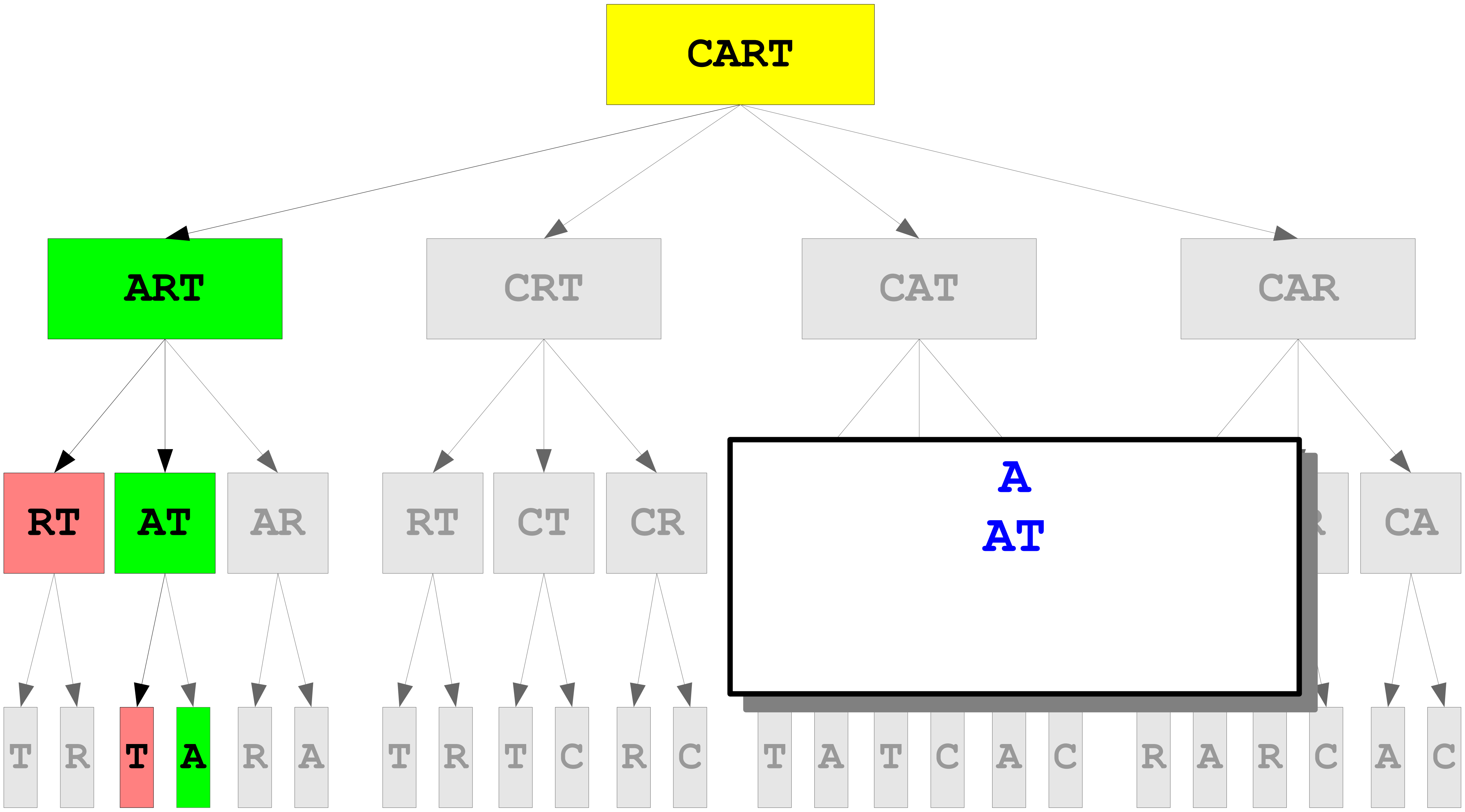
A

Generating the Answer



A
AT

Generating the Answer



CART

ART CRT CAT CAR

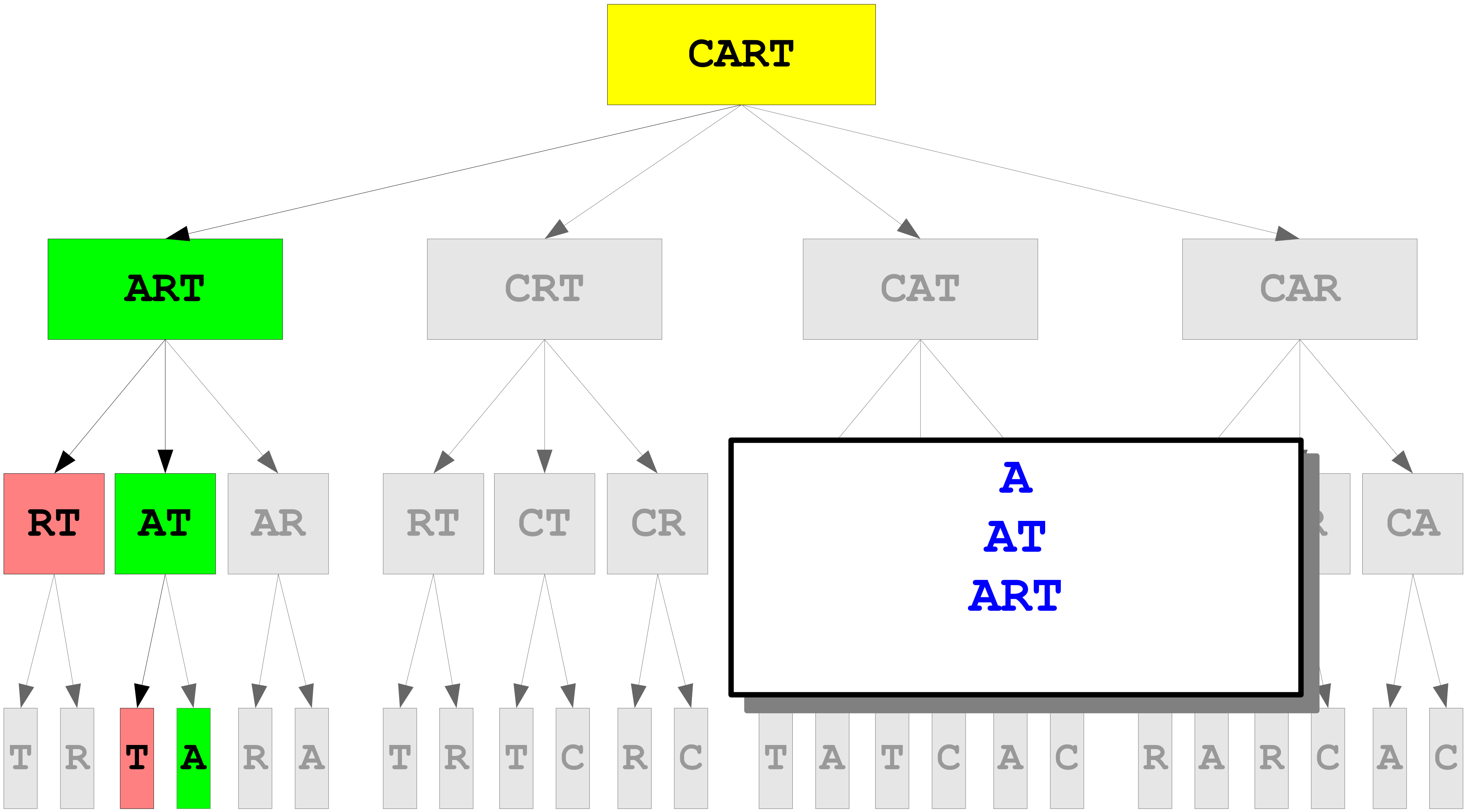
RT AT AR RT CT CR R CA

T R T A R A T R T C R C T A T C A C R A R C A C

A

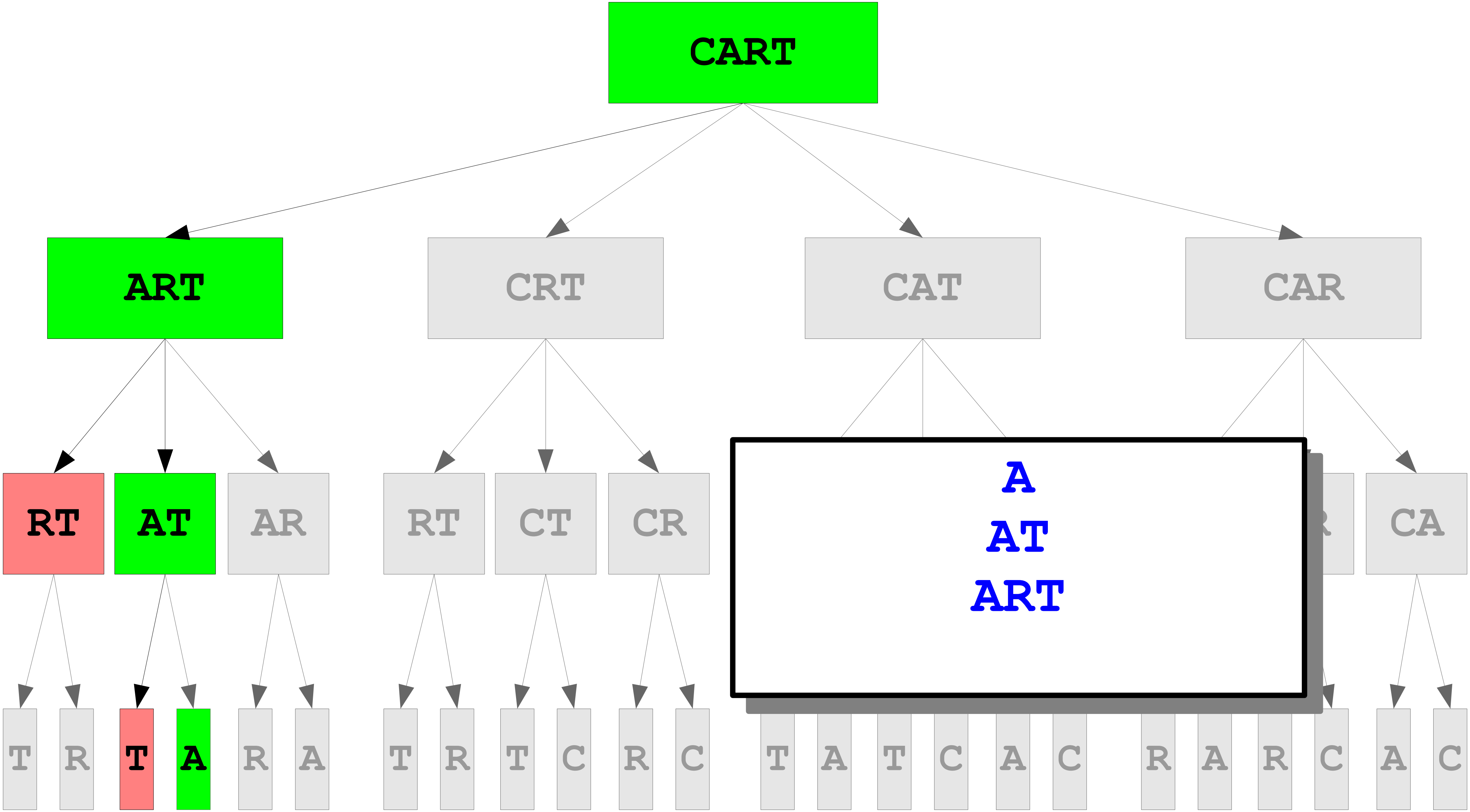
AT

Generating the Answer



A
AT
ART

Generating the Answer

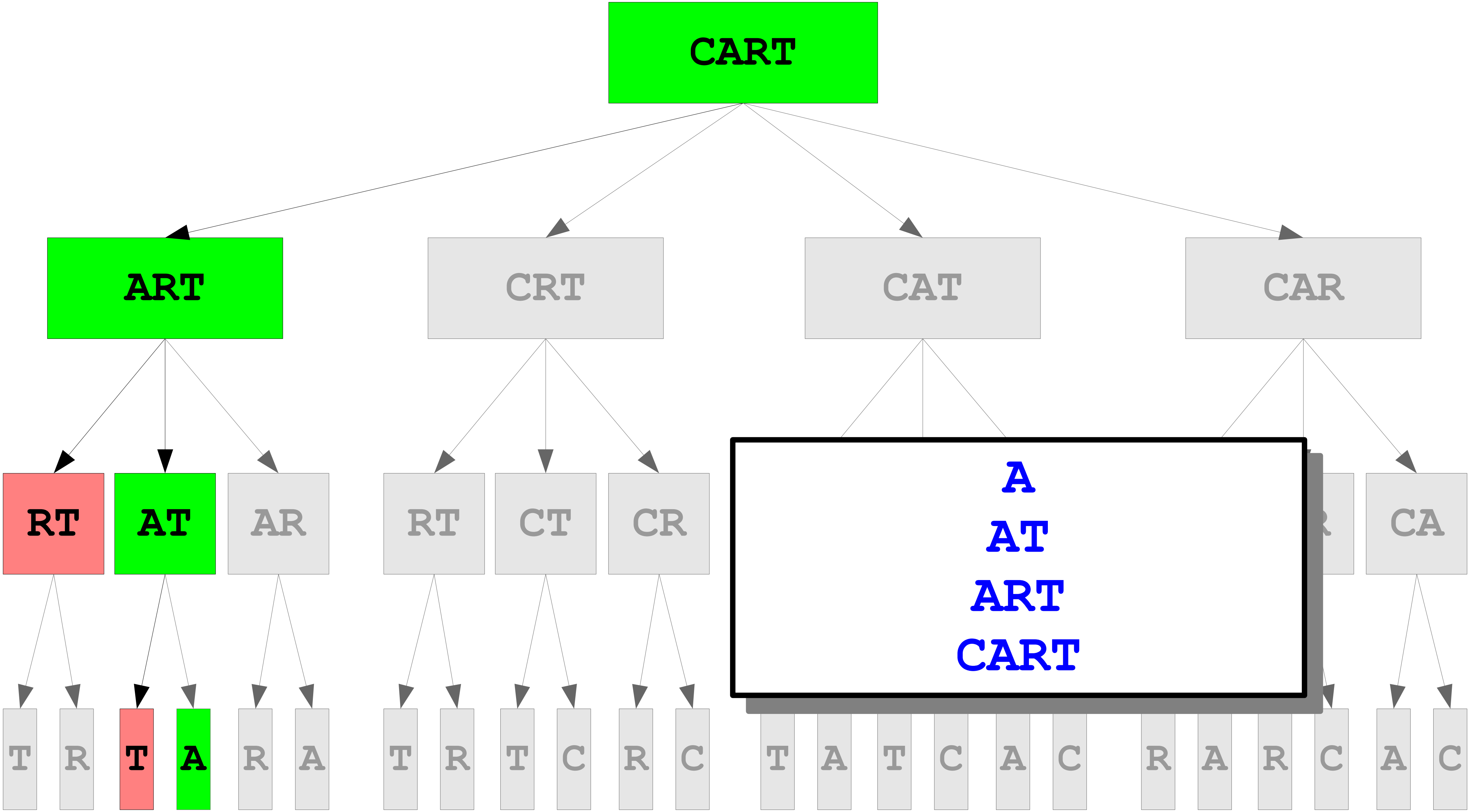


A

AT

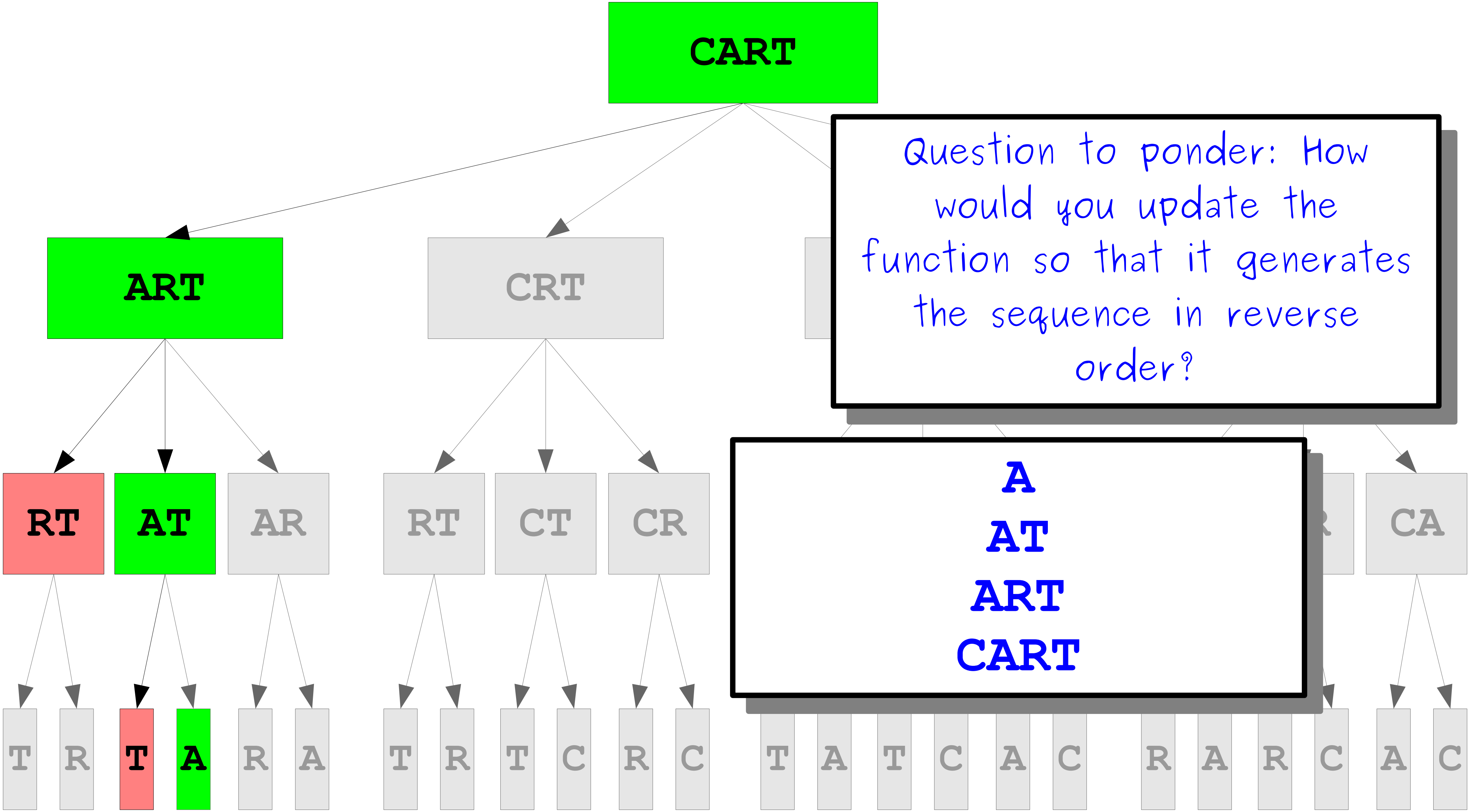
ART

Generating the Answer



A
AT
ART
CART

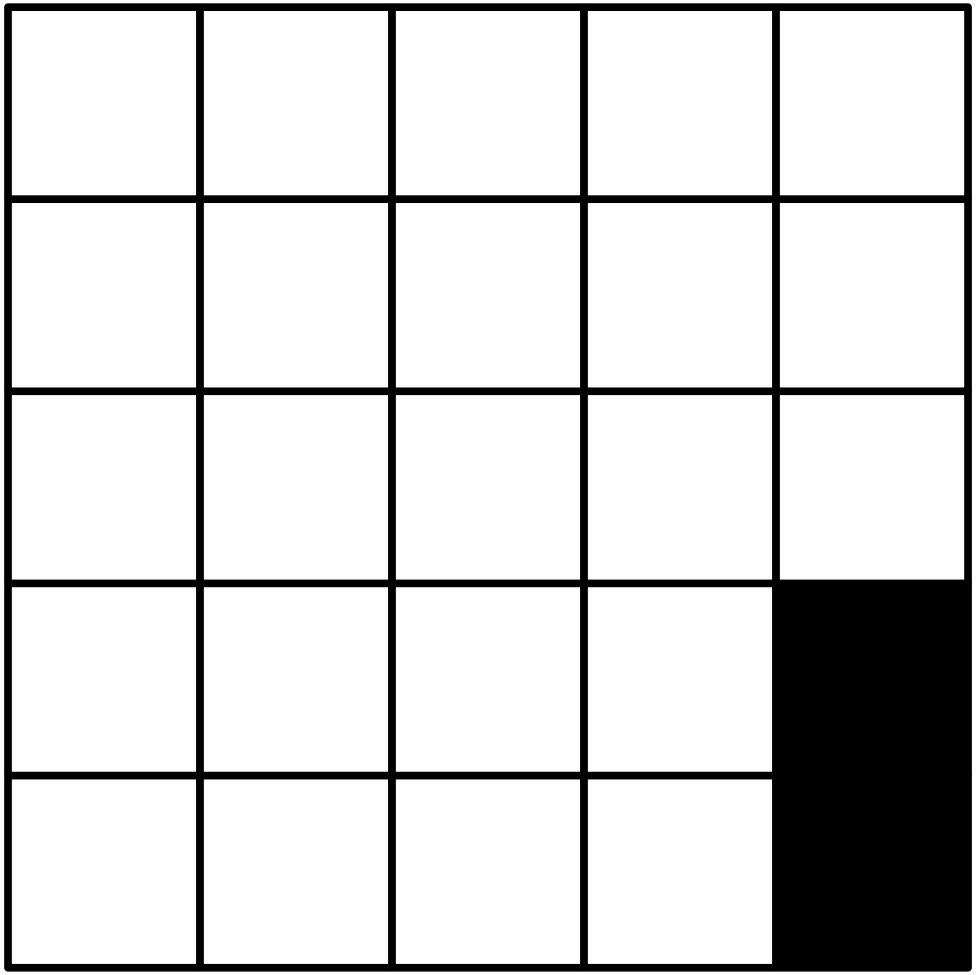
Generating the Answer



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

A
AT
ART
CART

Dense Crosswords

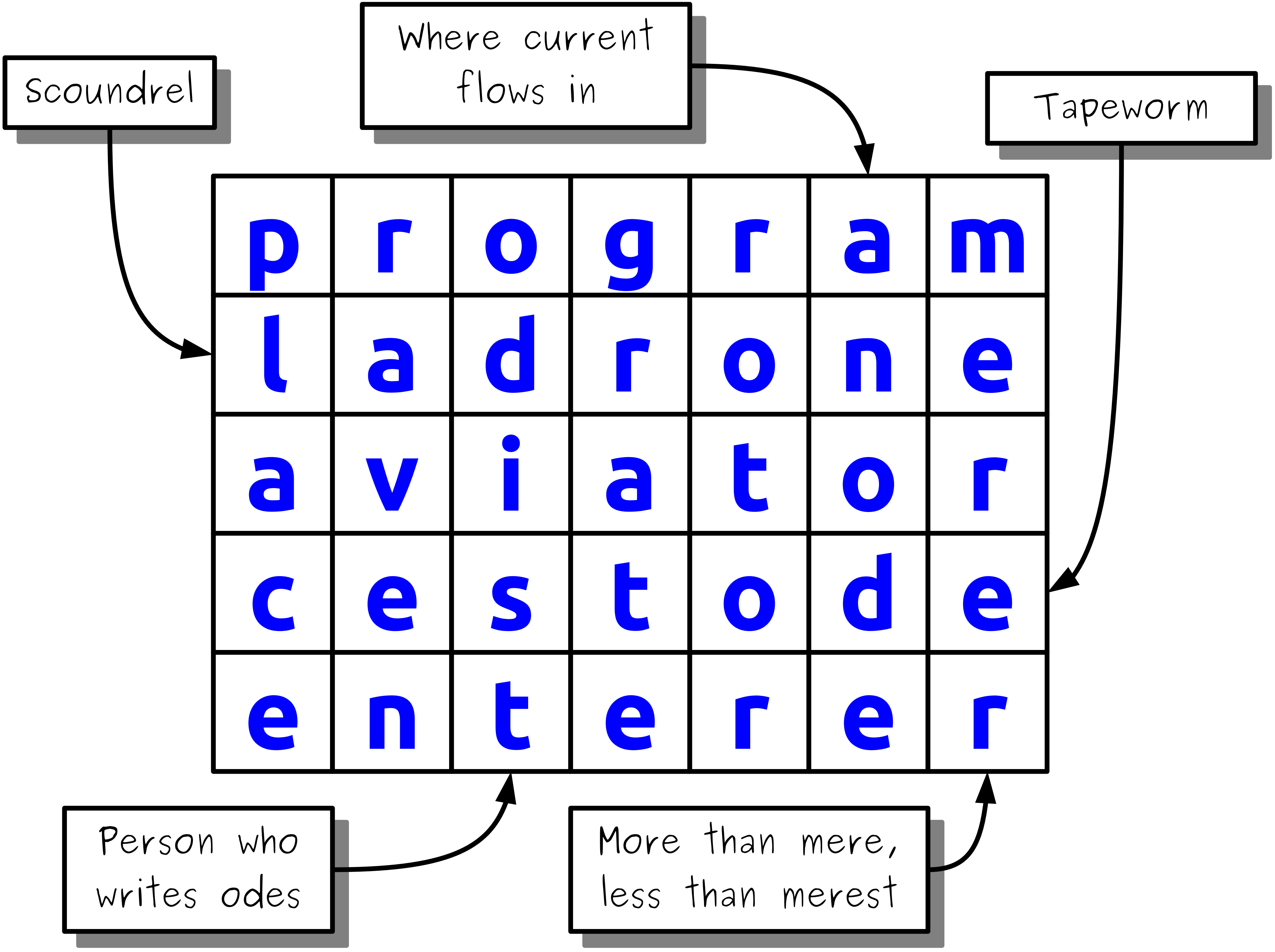


New York Times Mini Crossword, February 1, 2019

s	h	a	d	e
h	a	i	r	s
i	n	d	u	s
l	o	a	n	■
l	i	n	k	■

New York Times Mini Crossword, February 1, 2019

Can we design a crossword puzzle
where *every square* must be filled in?



Scoundrel

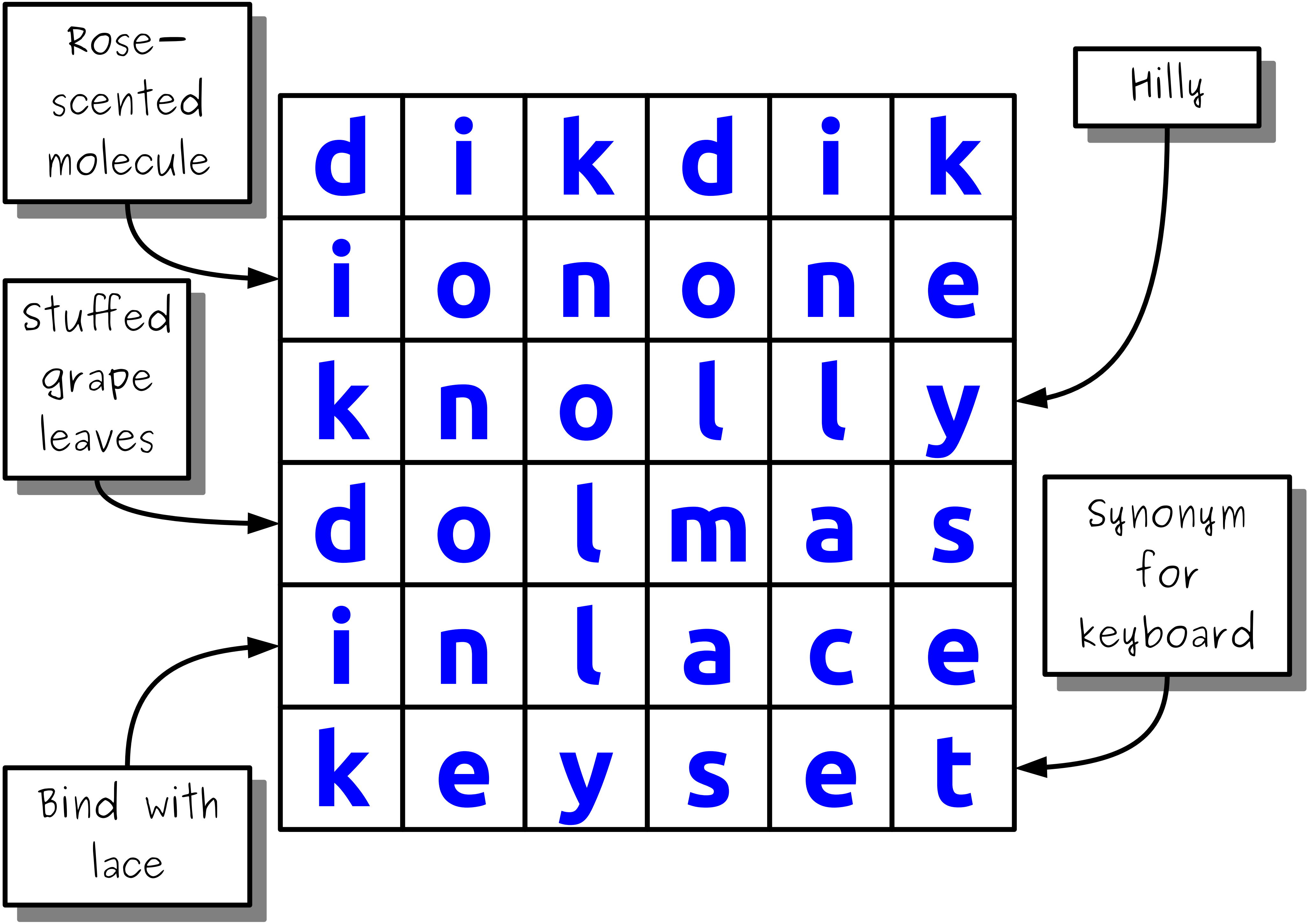
Where current flows in

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

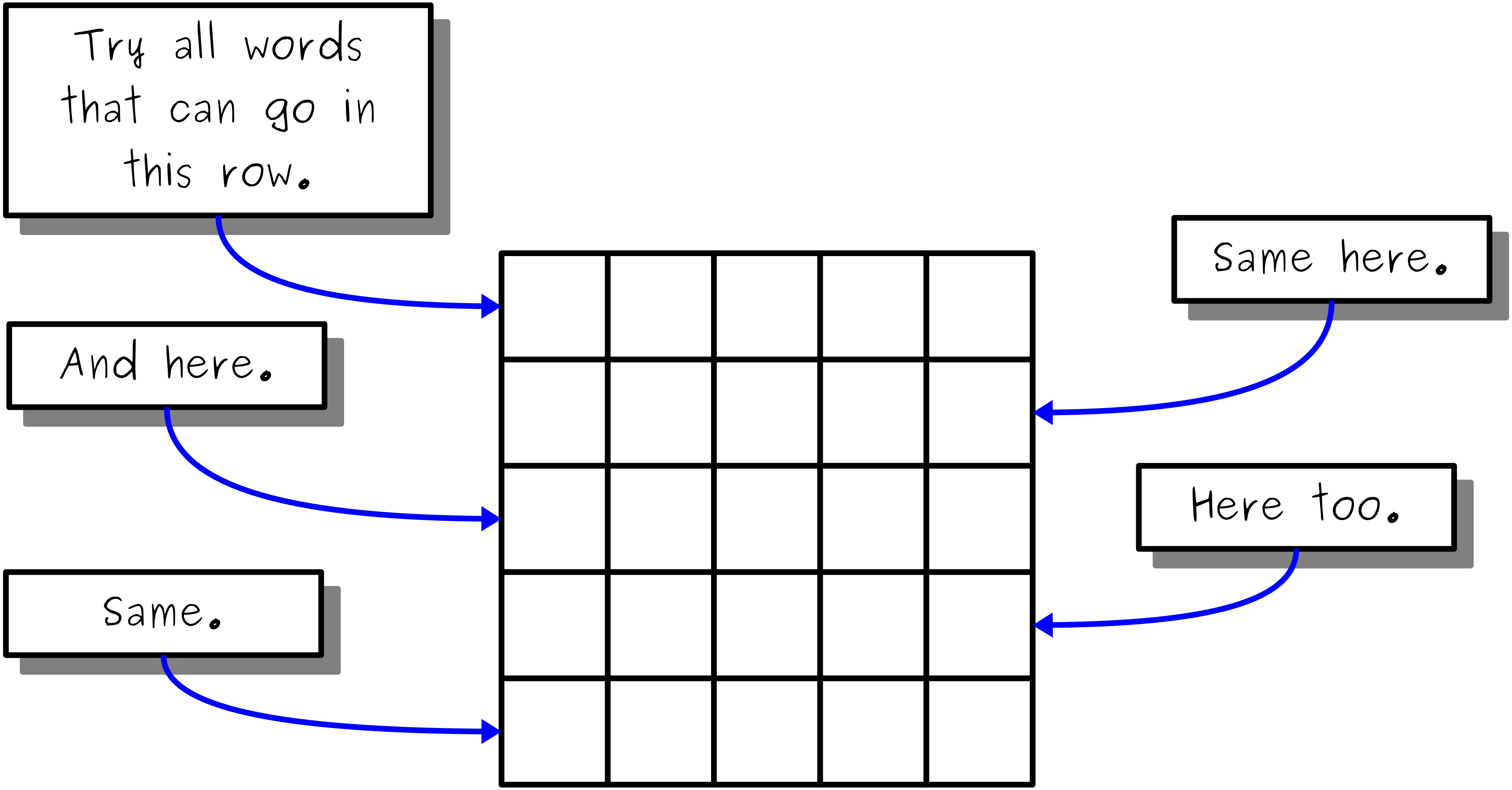
Stuffed grape leaves

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



Try all words that can go in this row.

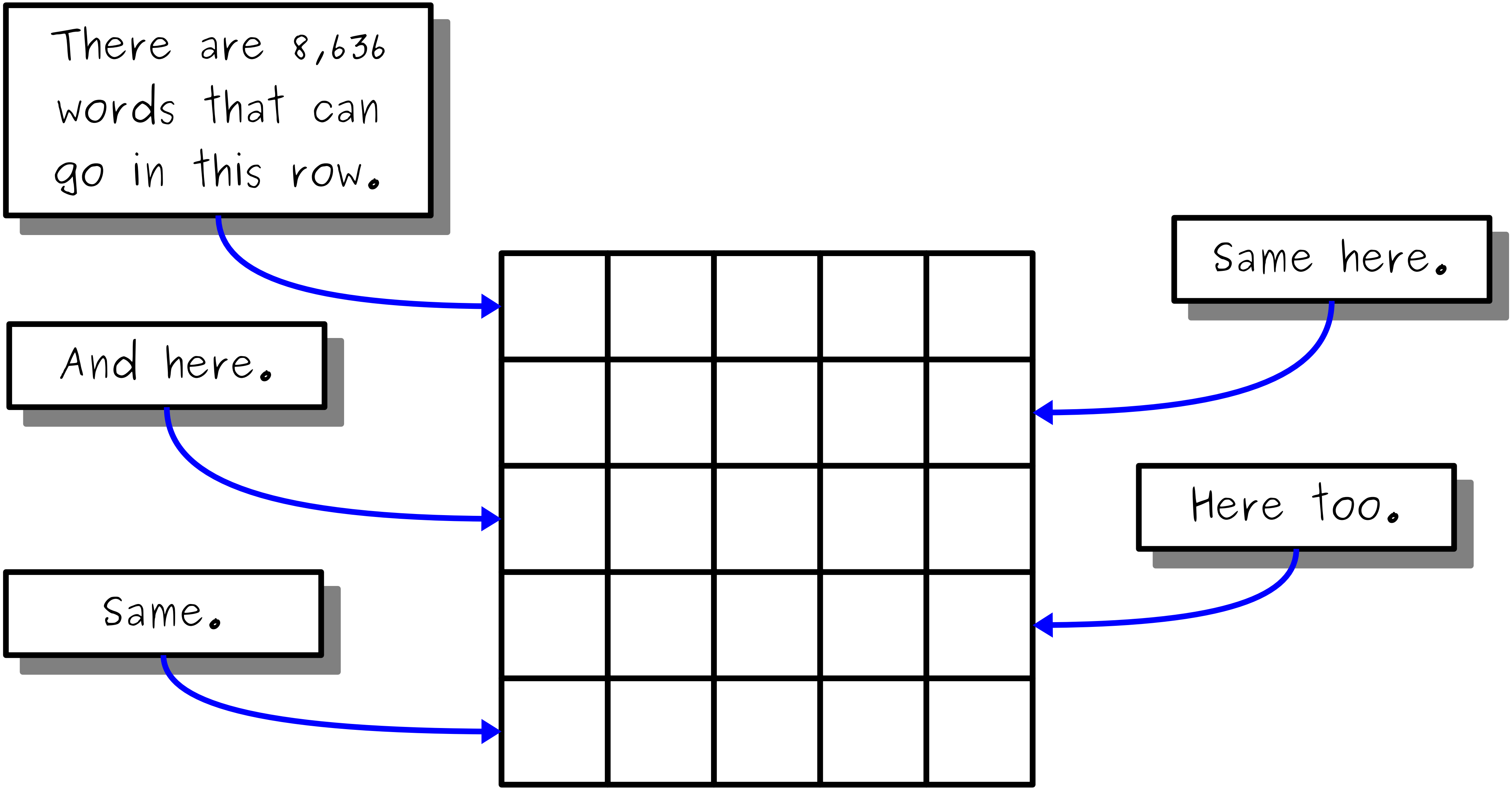
Same here.

And here.

Here too.

Same.

Idea: Fill this in using recursive backtracking.



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

Same here.

And here.

Here too.

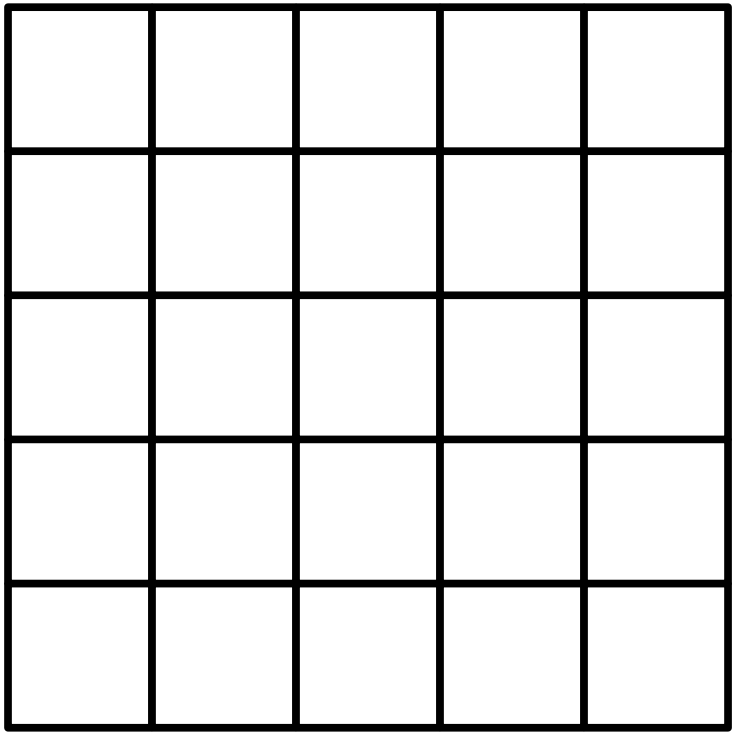
Same.

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

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

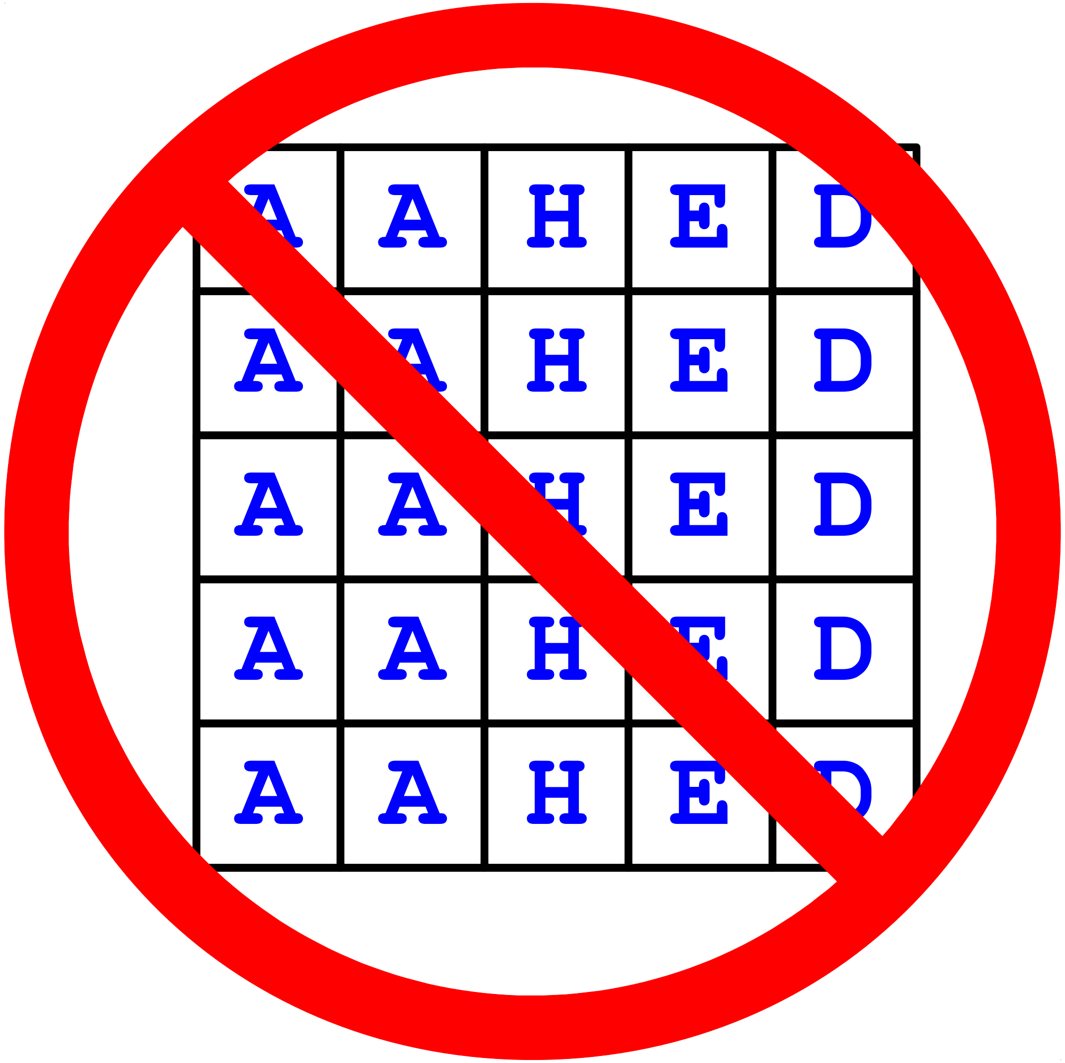
Generating Dense Crosswords

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

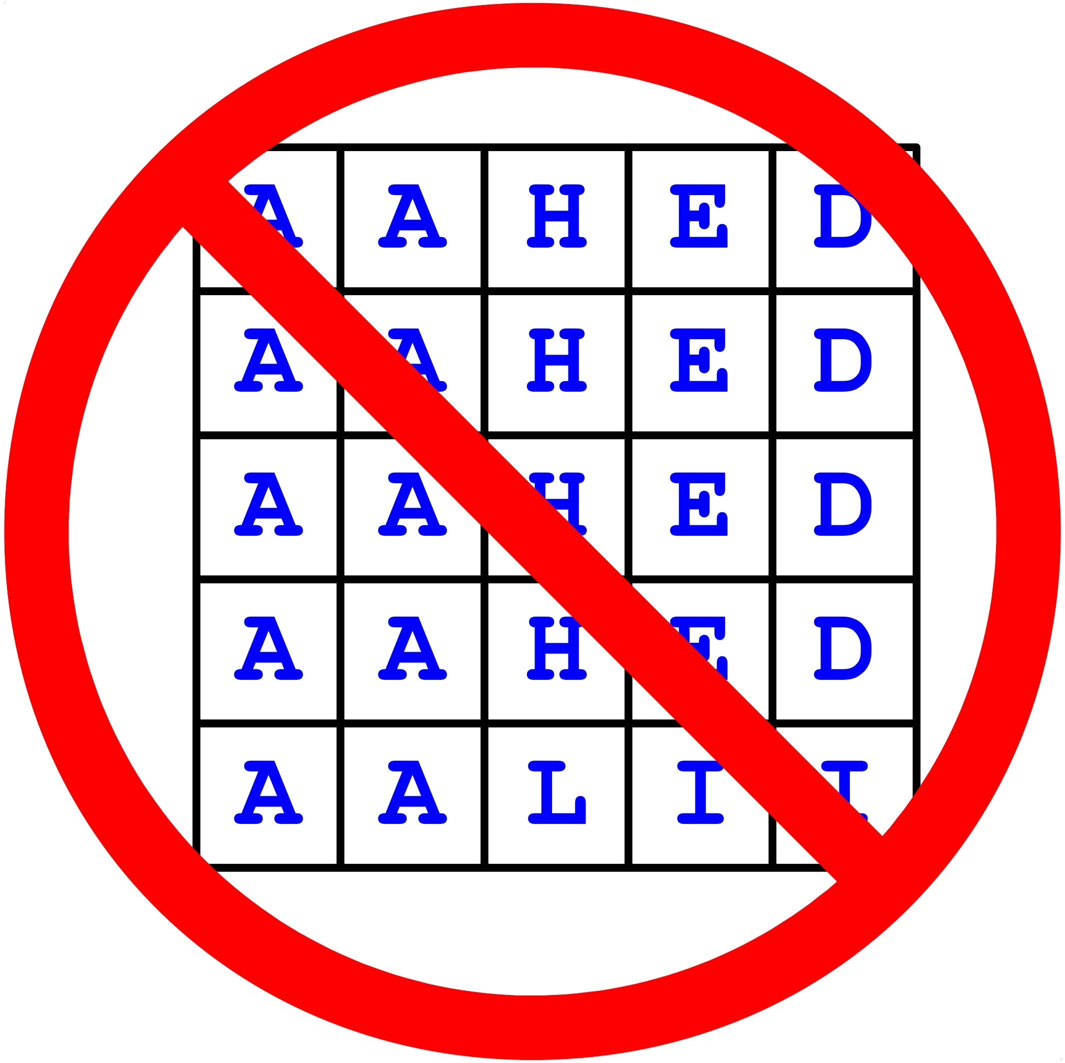


A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D

Generating Dense Crosswords

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

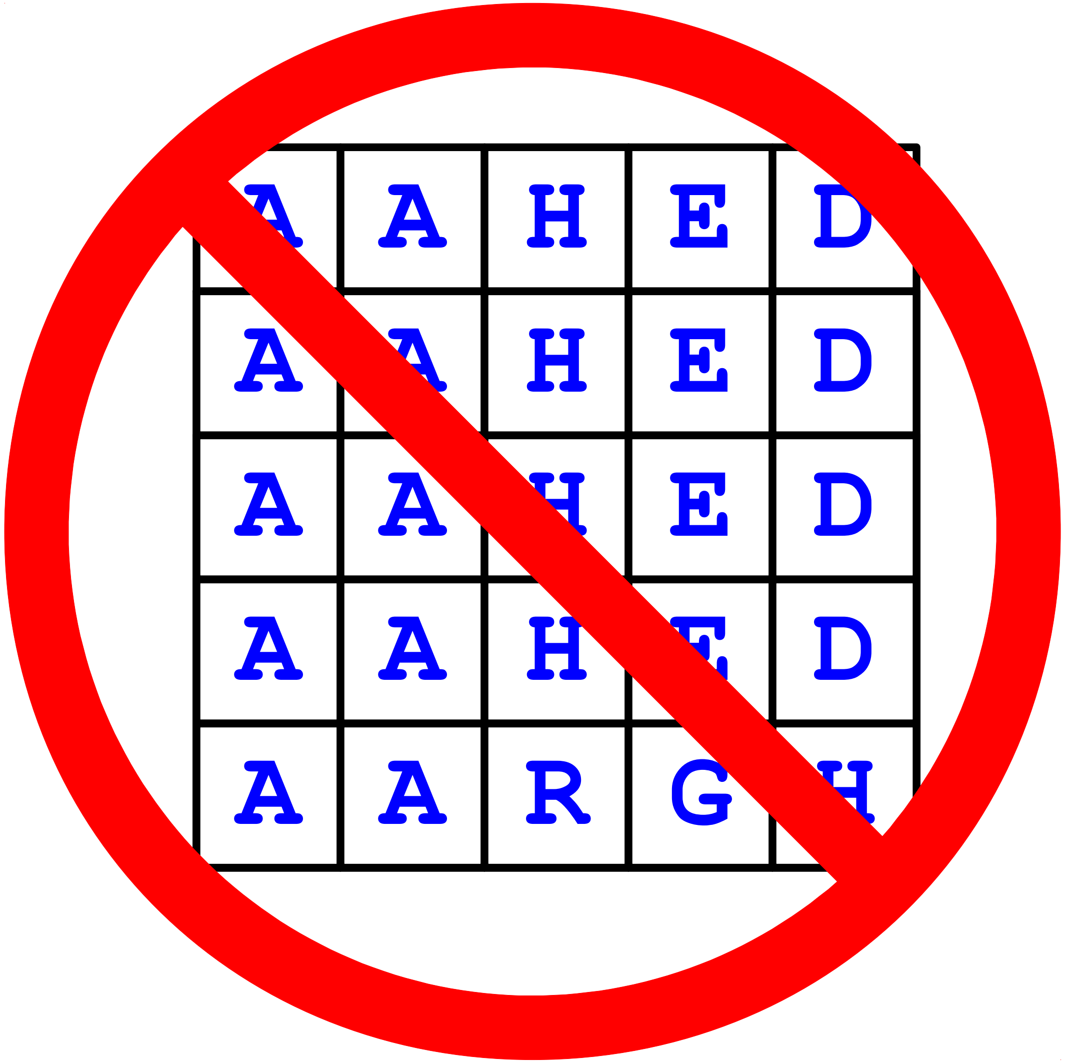


A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	L	I	I

Generating Dense Crosswords

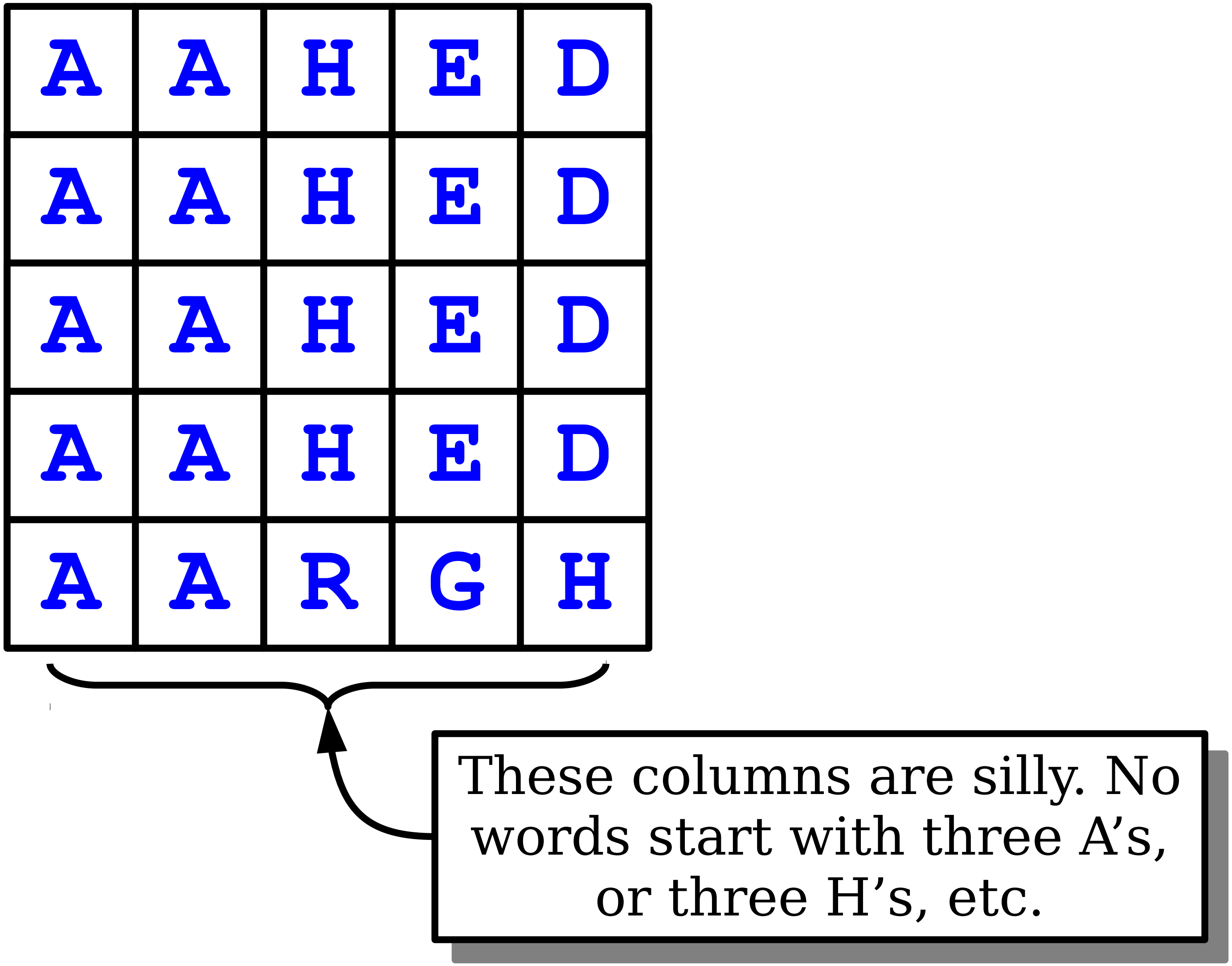
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	H	E	D
A	A	R	G	H

Generating Dense Crosswords



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

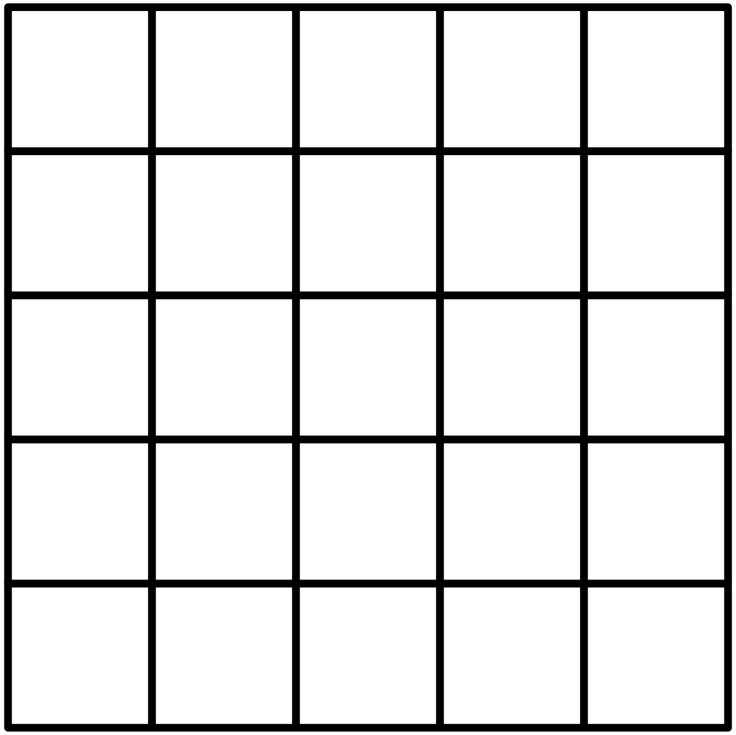
Generating Dense Crosswords



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

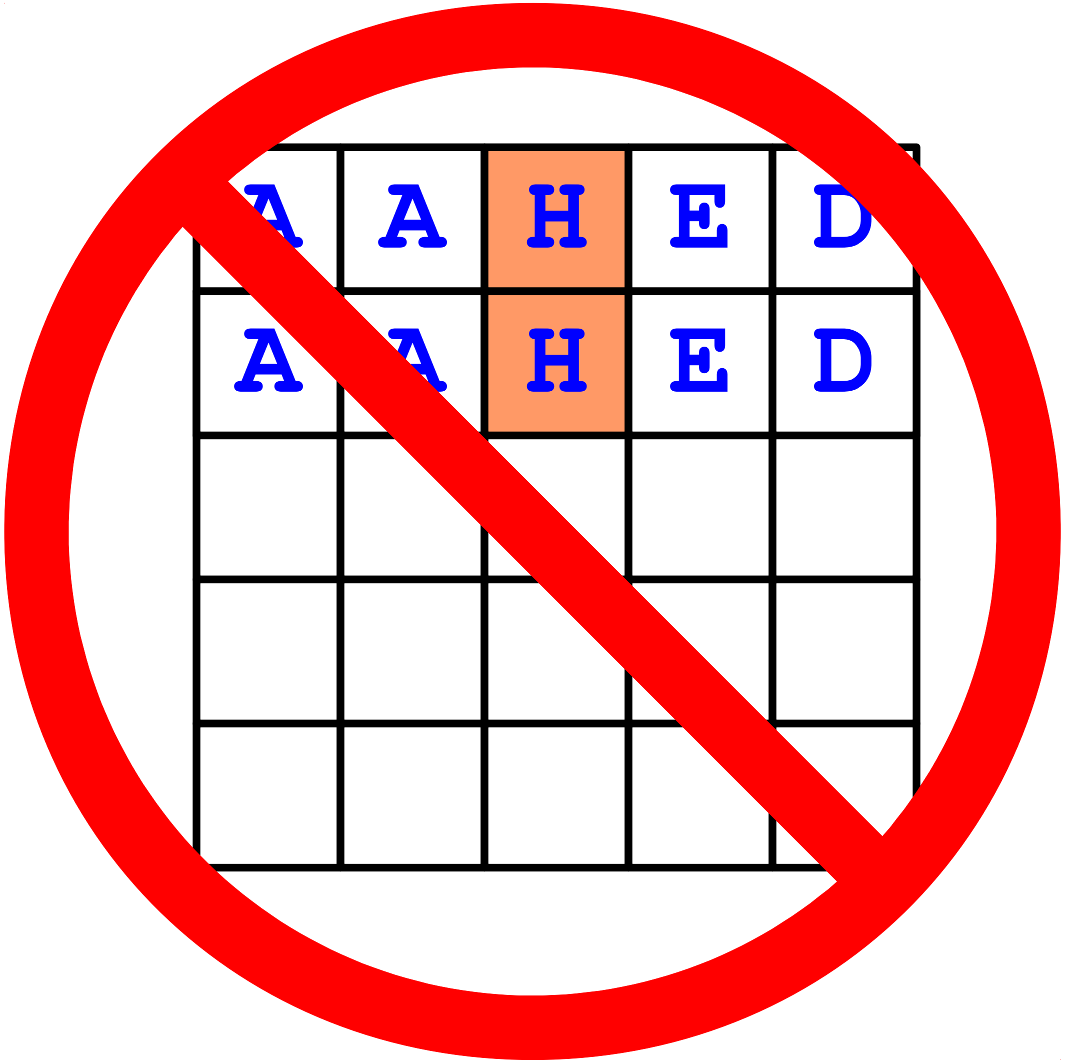
Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

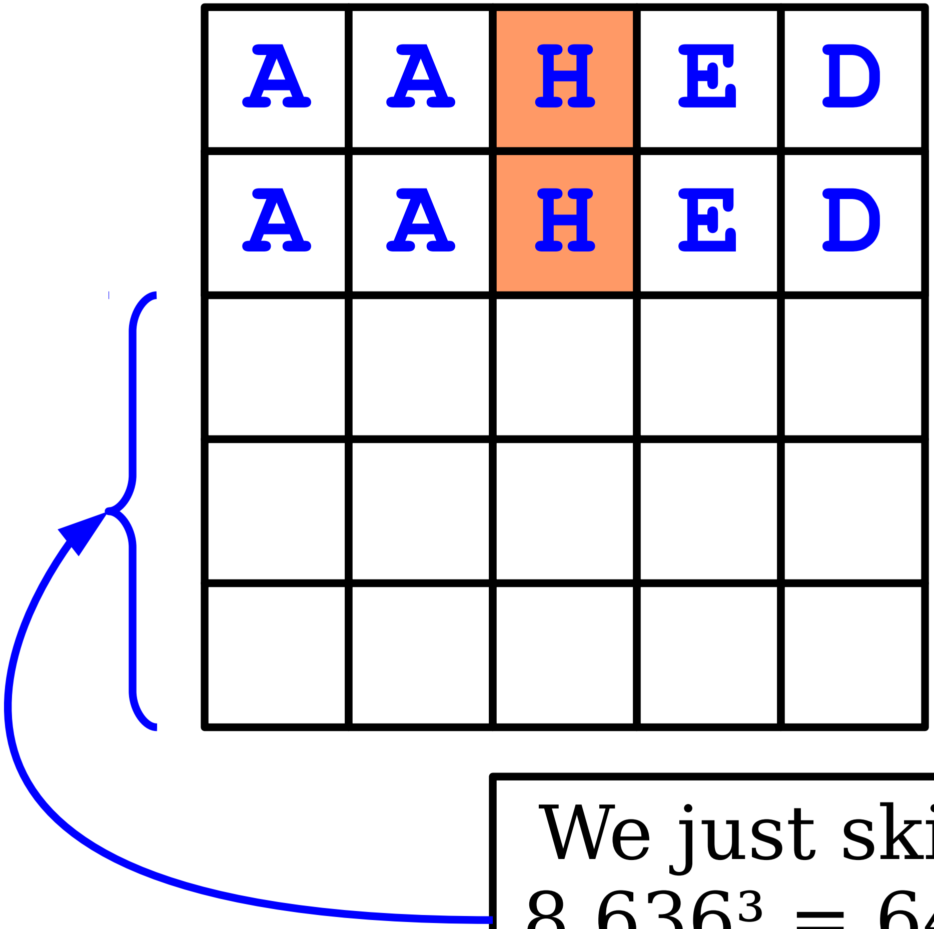
Generating Dense Crosswords

A	A	H	E	D
A	A	H	E	D

Generating Dense Crosswords



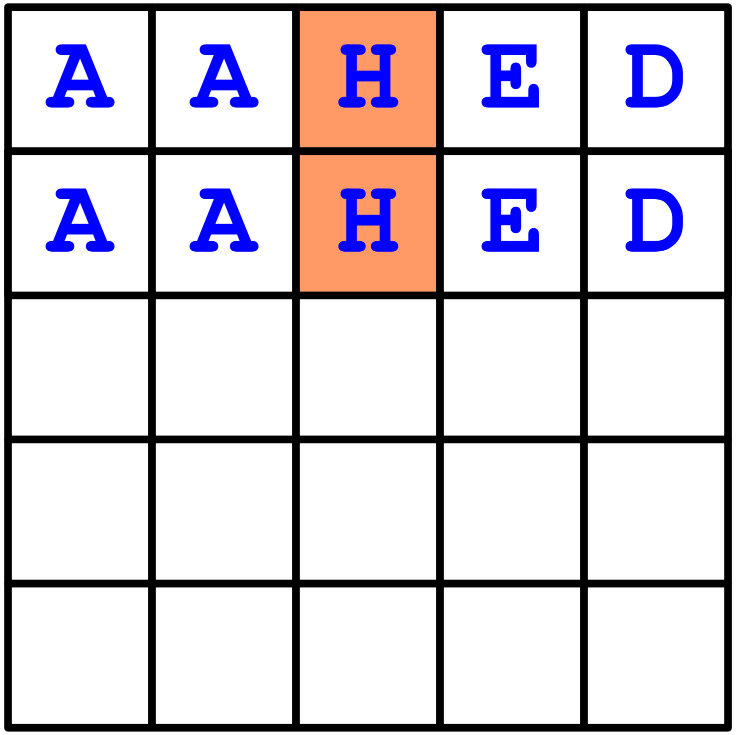
Generating Dense Crosswords



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



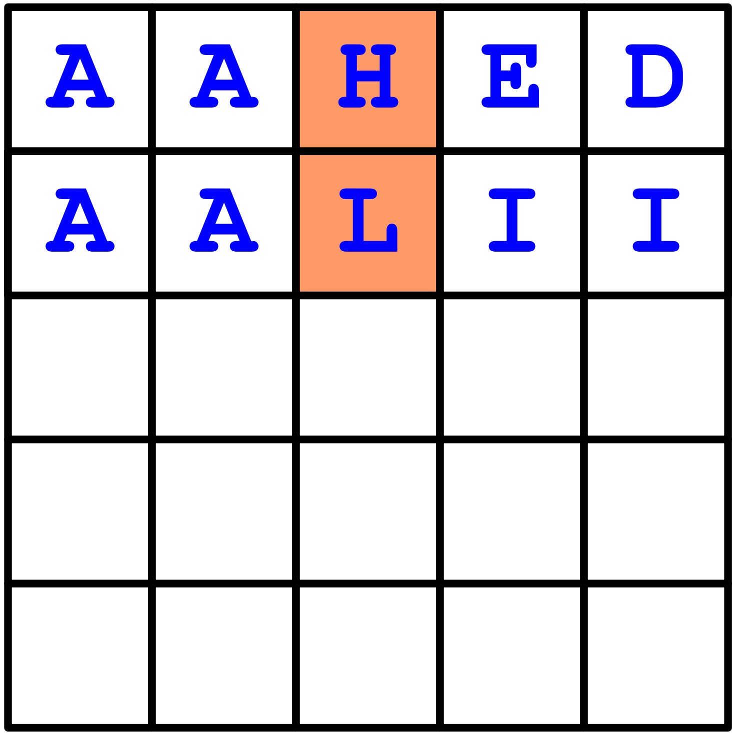
A	A	H	E	D
A	A	H	E	D

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

Generating Dense Crosswords

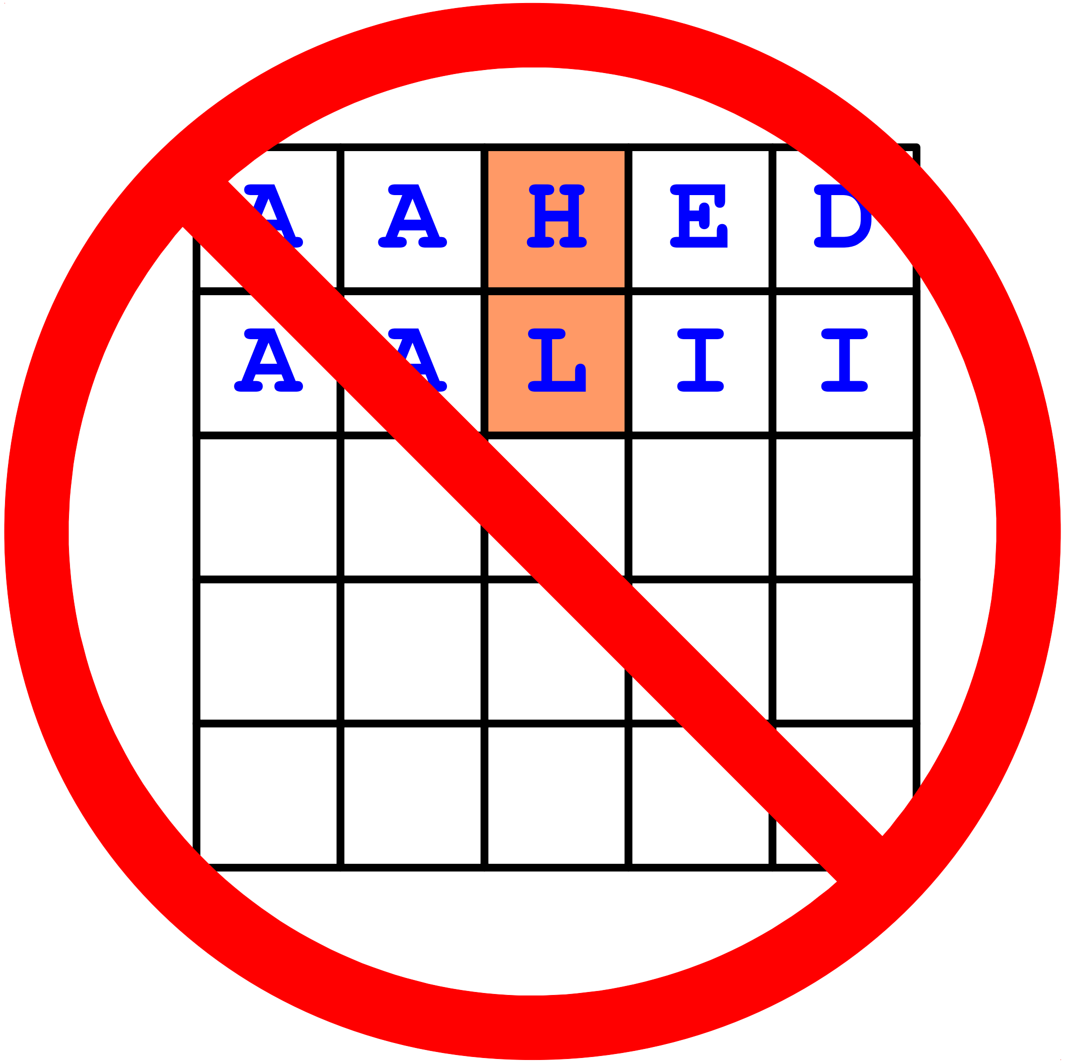
A	A	H	E	D
A	A	L	I	I

Generating Dense Crosswords



A	A	H	E	D
A	A	L	I	I

Generating Dense Crosswords



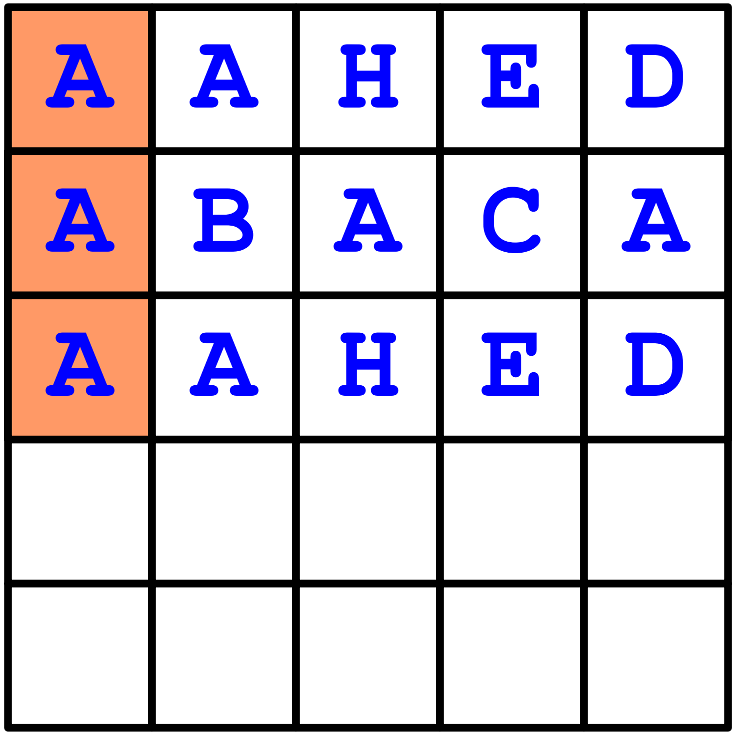
Generating Dense Crosswords

A	A	H	E	D
A	B	A	C	A

Generating Dense Crosswords

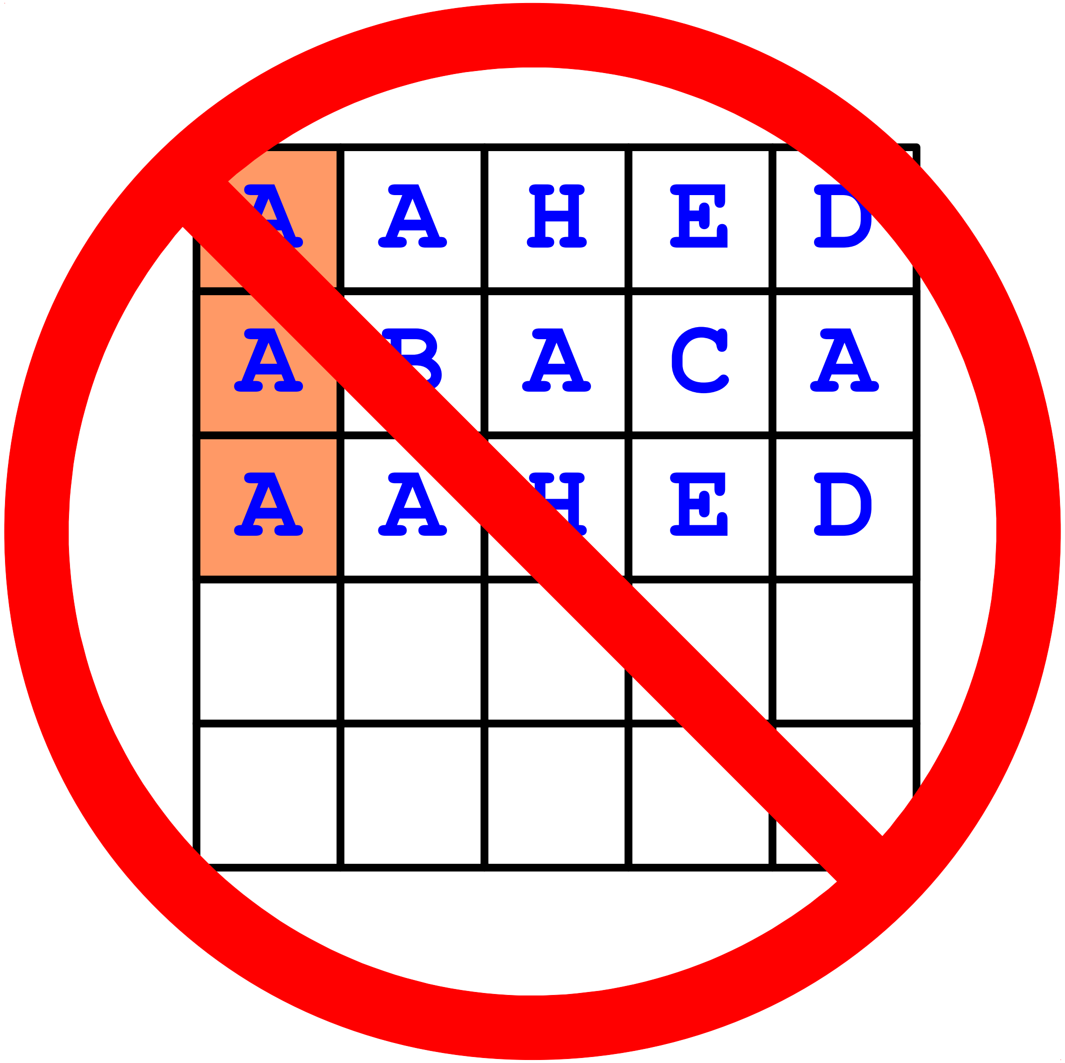
A	A	H	E	D
A	B	A	C	A
A	A	H	E	D

Generating Dense Crosswords



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

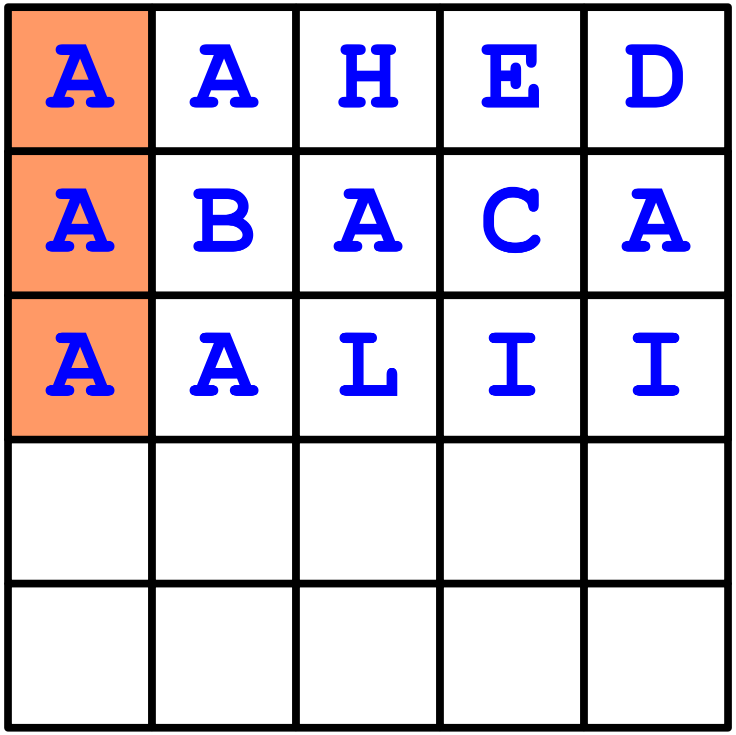
Generating Dense Crosswords



Generating Dense Crosswords

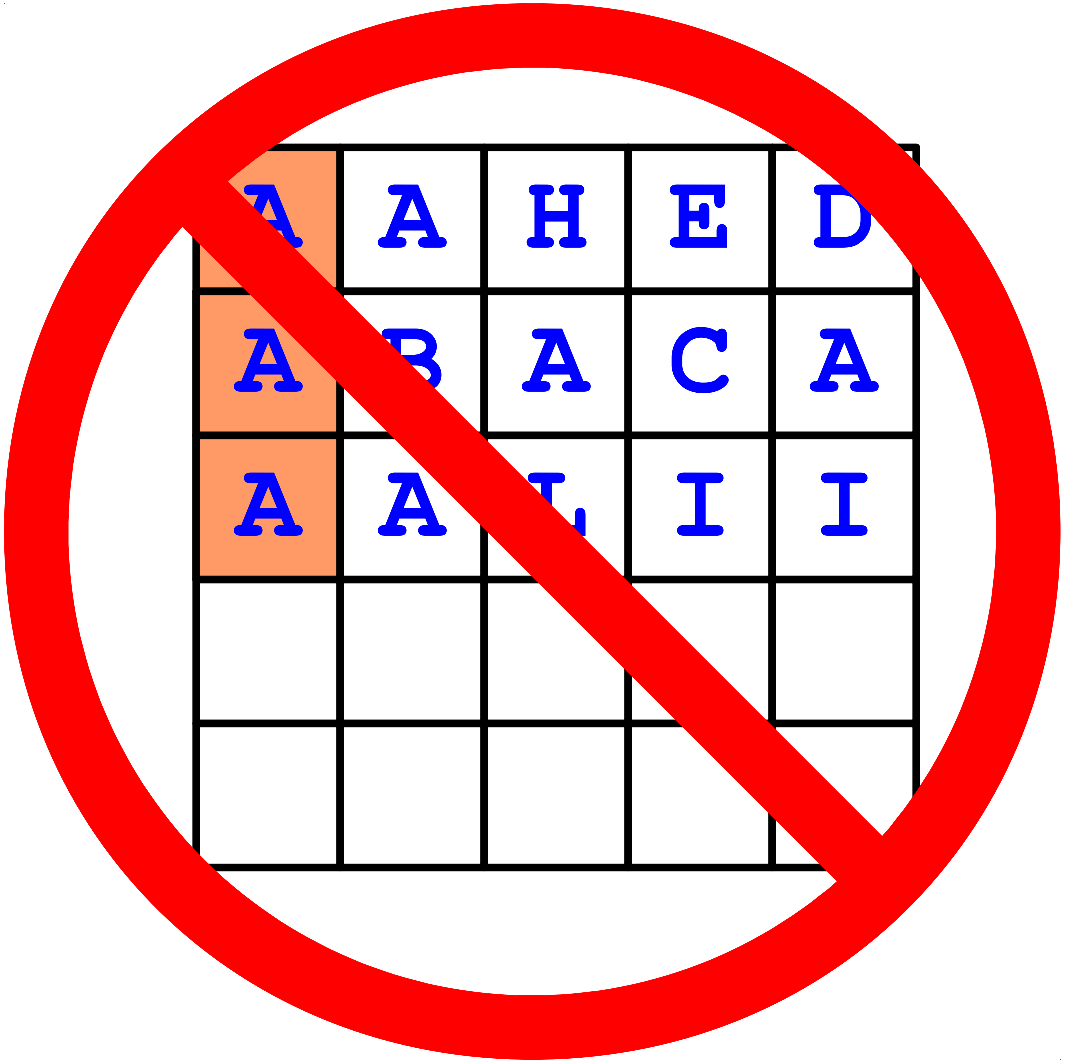
A	A	H	E	D
A	B	A	C	A
A	A	L	I	I

Generating Dense Crosswords



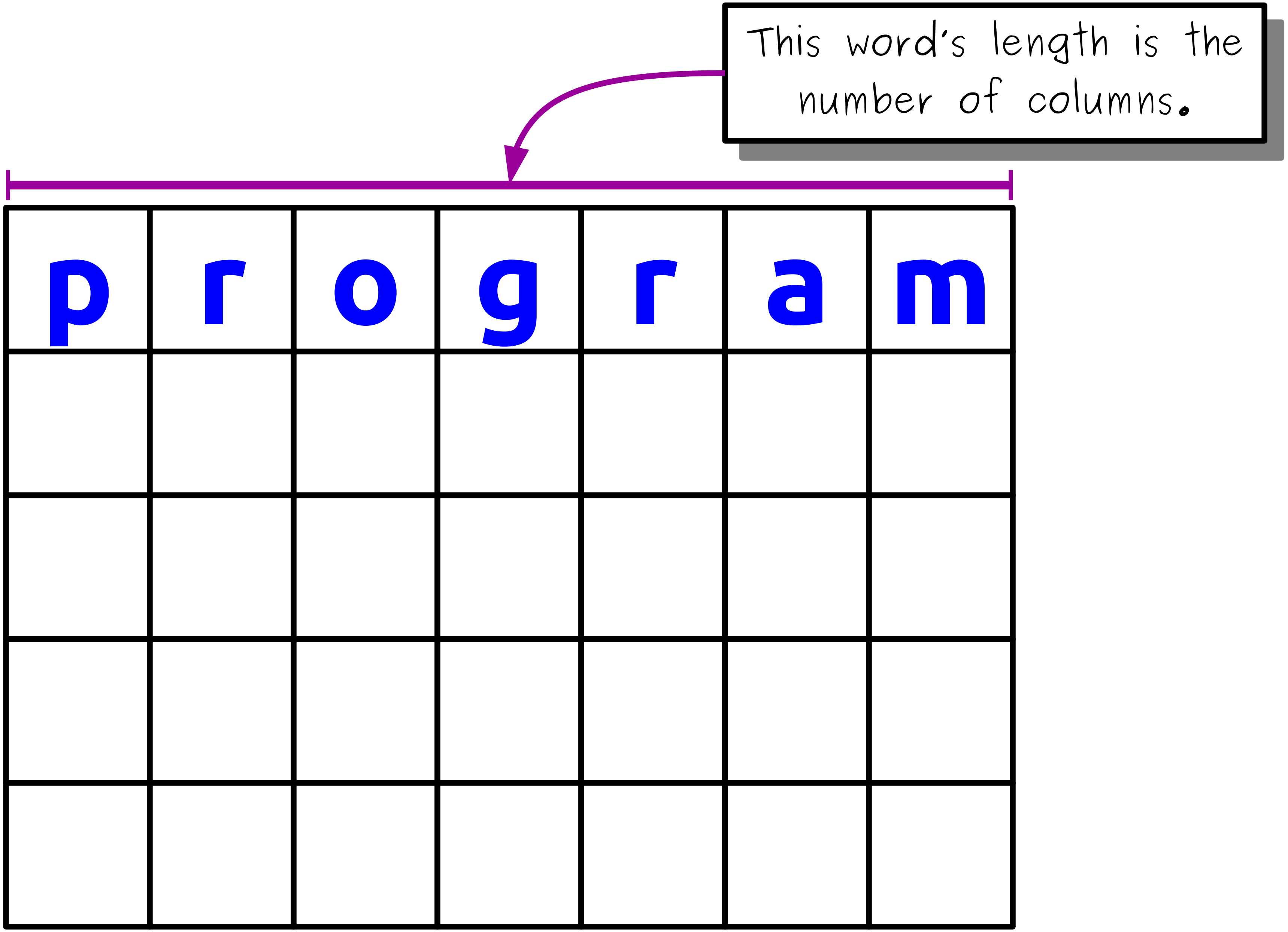
A	A	H	E	D
A	B	A	C	A
A	A	L	I	I

Generating Dense Crosswords



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

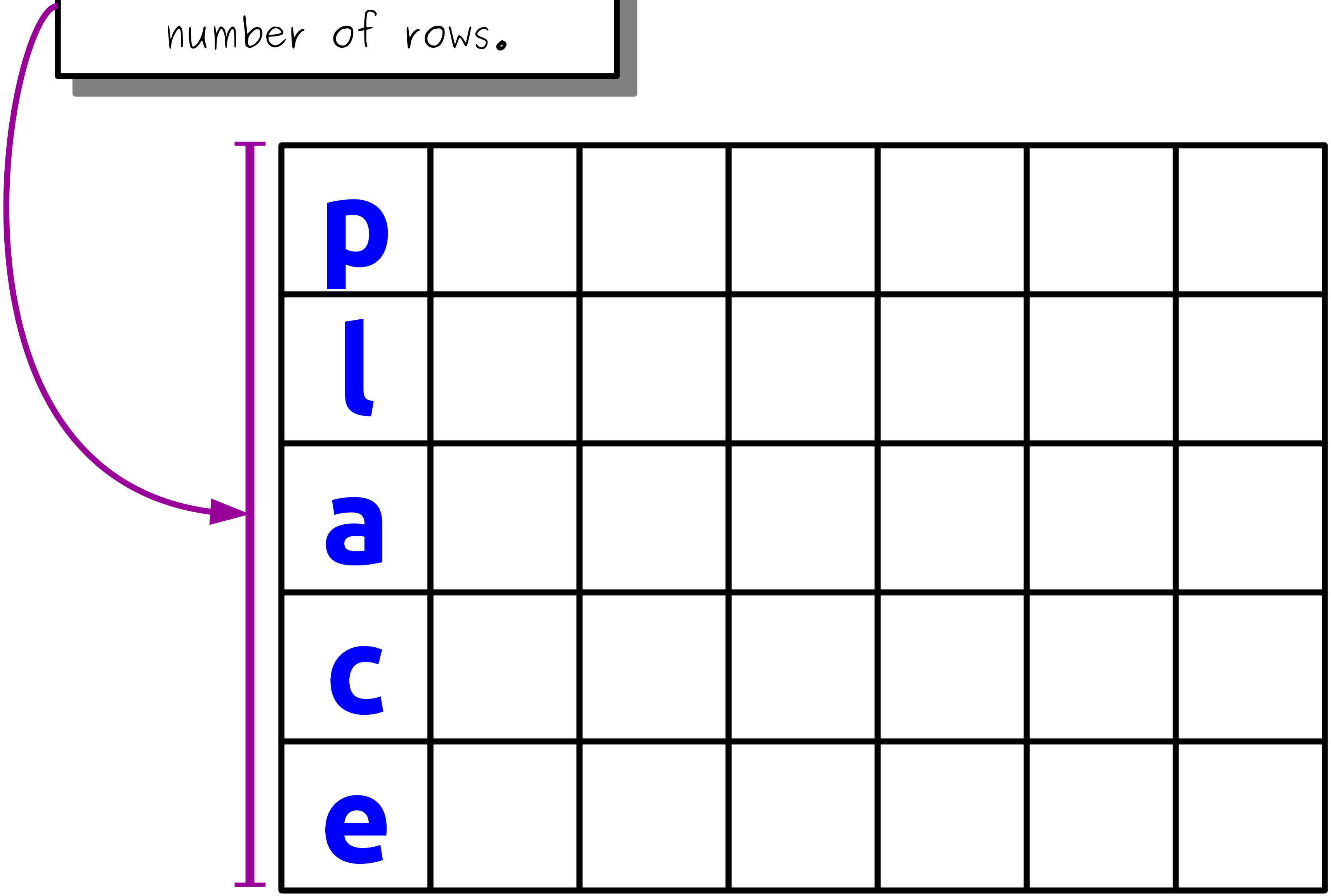
Let’s Code it Up!



This word's length is the number of columns.

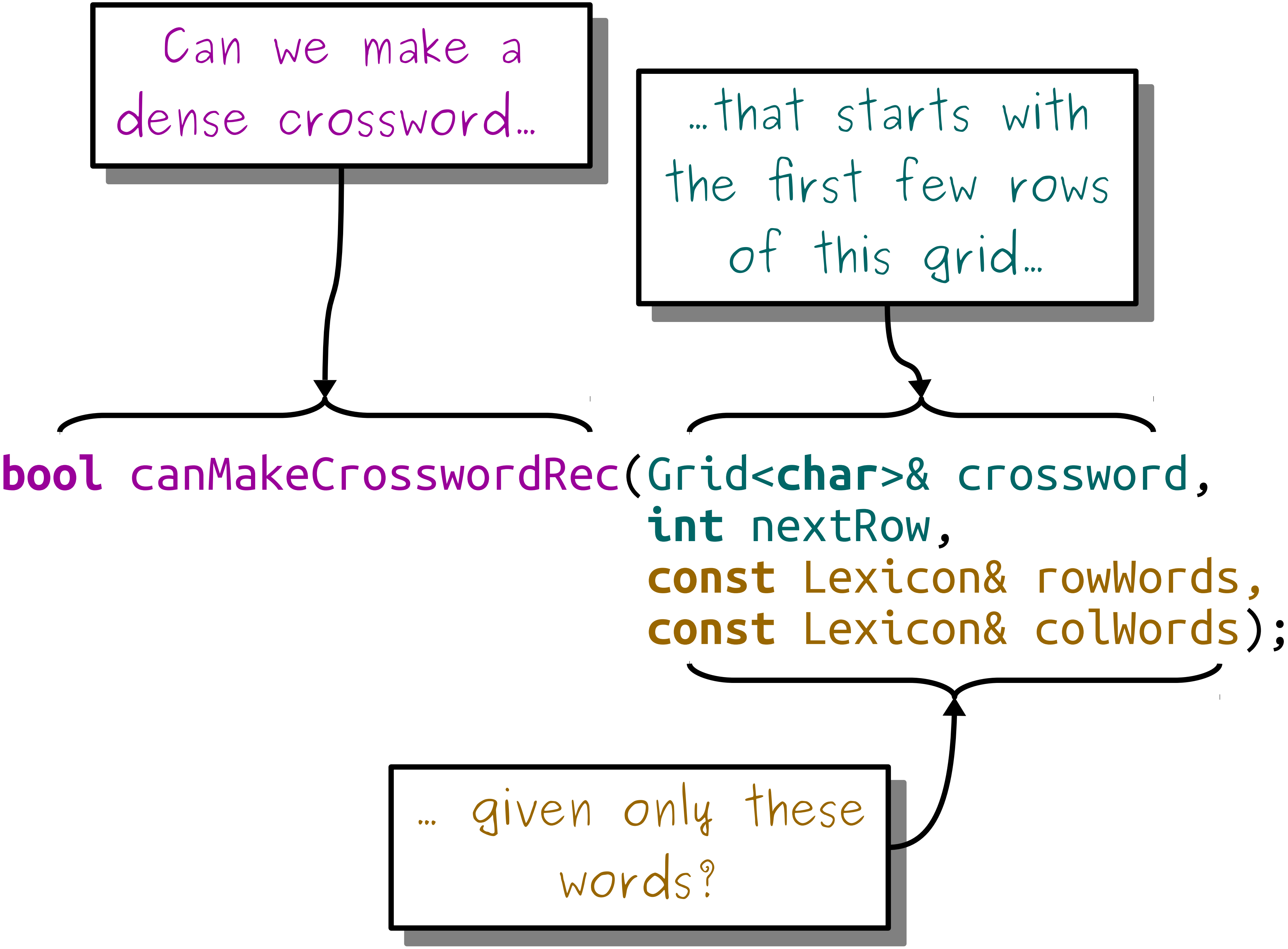
p	r	o	g	r	a	m

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);
```



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;
}
```

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***.

You’ve seen that ***optimizing code*** is more about ***changing strategy*** than writing less code.

Congratulations on making it this far!

Your Action Items

- ***Finish Chapter 9 of the textbook.***
  - It’s all about backtracking, and there are some great examples in there!
- ***Keep working on Assignment 3.***
  - You should be done with the Sierpinski Triangle and Human Pyramids, and be making good progress on Shift Scheduling.
  - Aim to complete Shift Scheduling and to have started Riding Circuit by Monday.

Next Time

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