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 <u>https://cs106b.stanford.edu/pollev</u>

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.

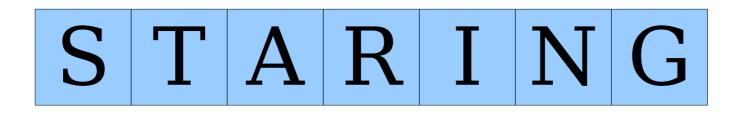
Recap from Last Time

A Little Word Puzzle

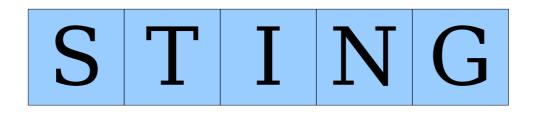
"What nine-letter word can be reduced to a single-letter word one letter at a time by removing letters, leaving it a legal word at each step?"

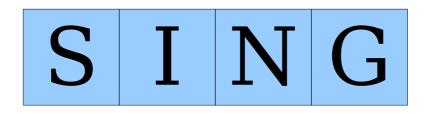
S T A R T L I N G

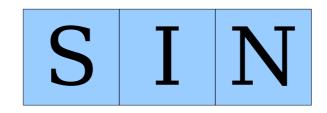
STARTING



S T R I N G











New Stuff!

Our Solution, In Action

```
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++) {</pre>
        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++) {</pre>
        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
                                                Specifically, this
      an unconditional return
                                              code makes its final
                                               decision based on
    statement in a for loop. This
                                              the first character it
     almost certainly indicates
       the presence of a bug.
                                                tries removing.
```

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;

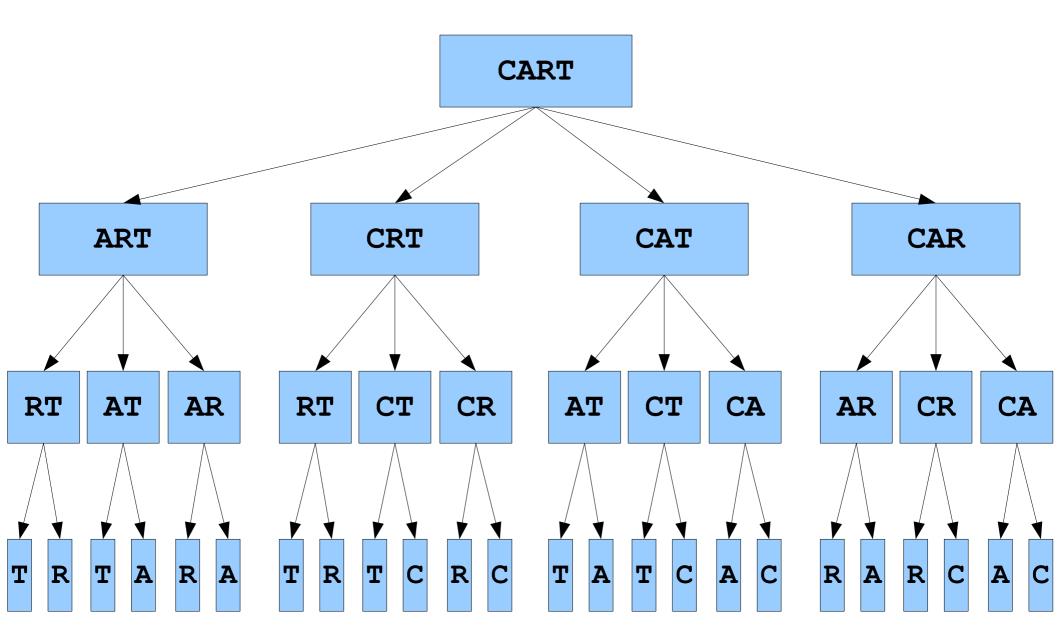
}

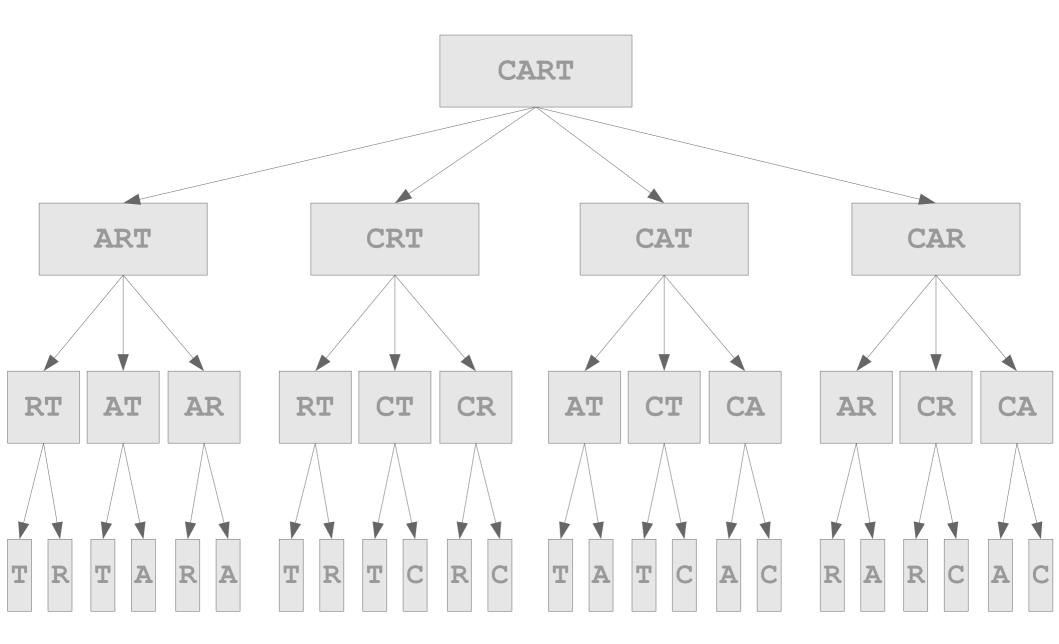
Note that <u>if</u> the recursive call succeeds, <u>then</u> we return success. If it doesn't succeed, that doesn't mean we've failed - it just means we need to try out the next option.

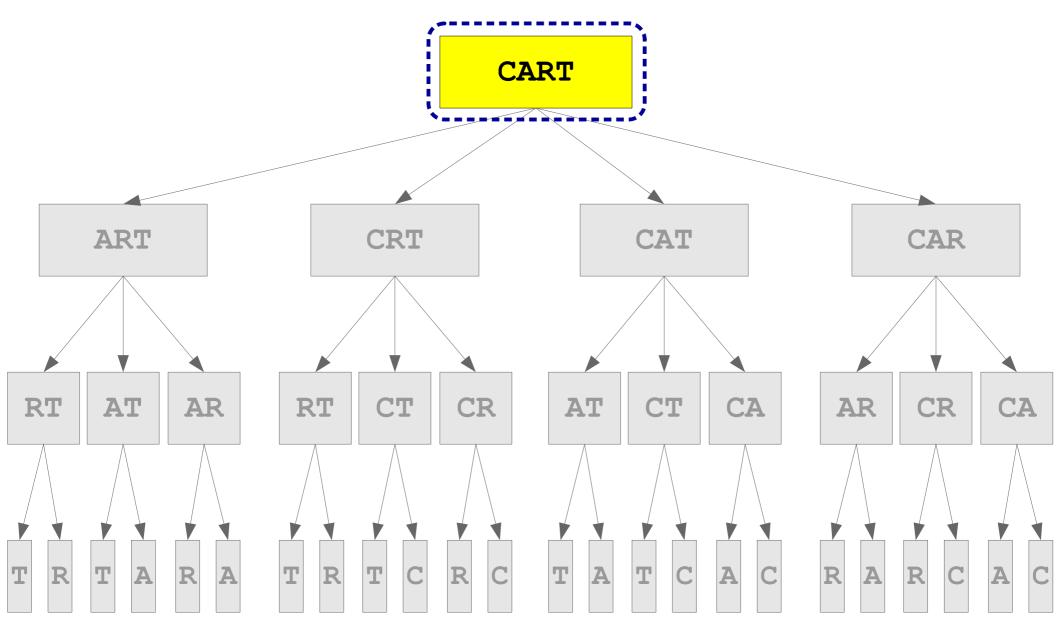
How do we know we're correct?

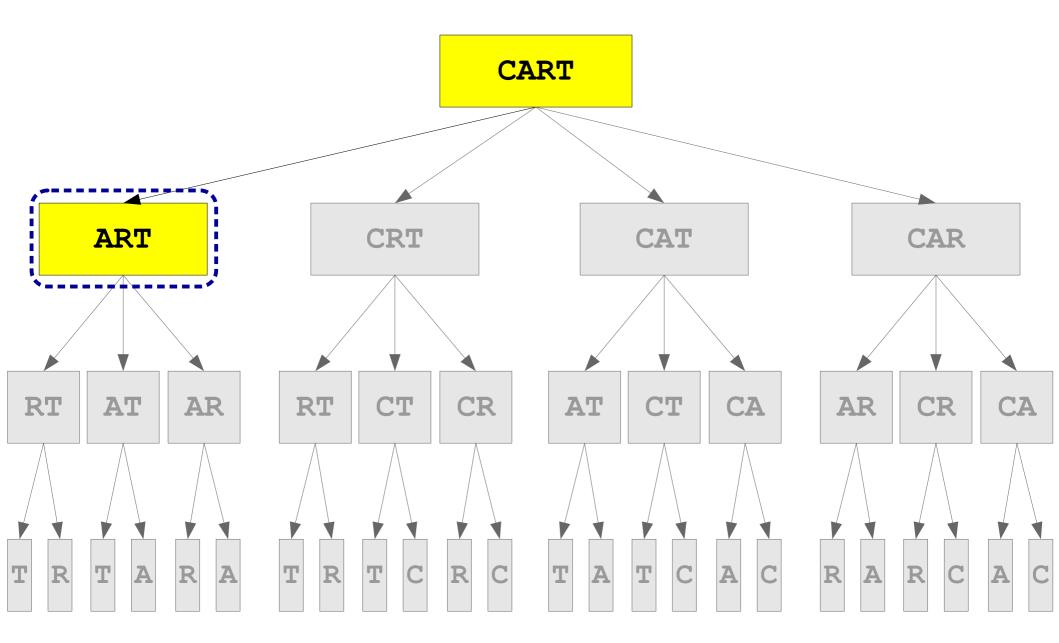
Optional<T>

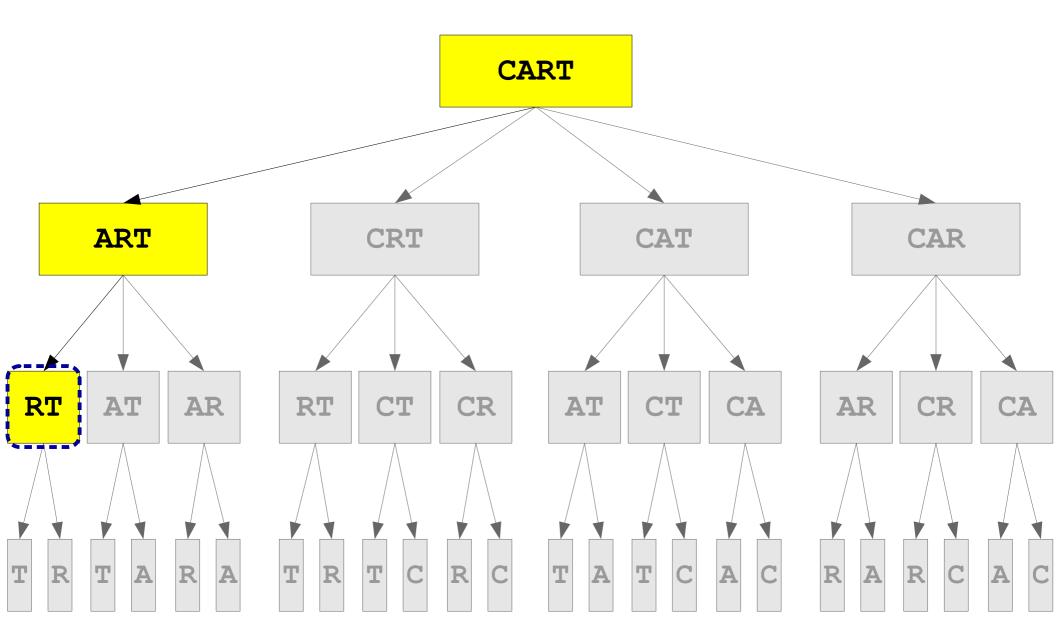
- The Optional<T> type represents either an object of type T or is Nothing at all.
- It's useful when working with recursive functions that look for something that may or may not exist.
 - If a solution exists, return it as usual.
 - Otherwise, return Nothing.
- If the Optional<T> is a value of type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

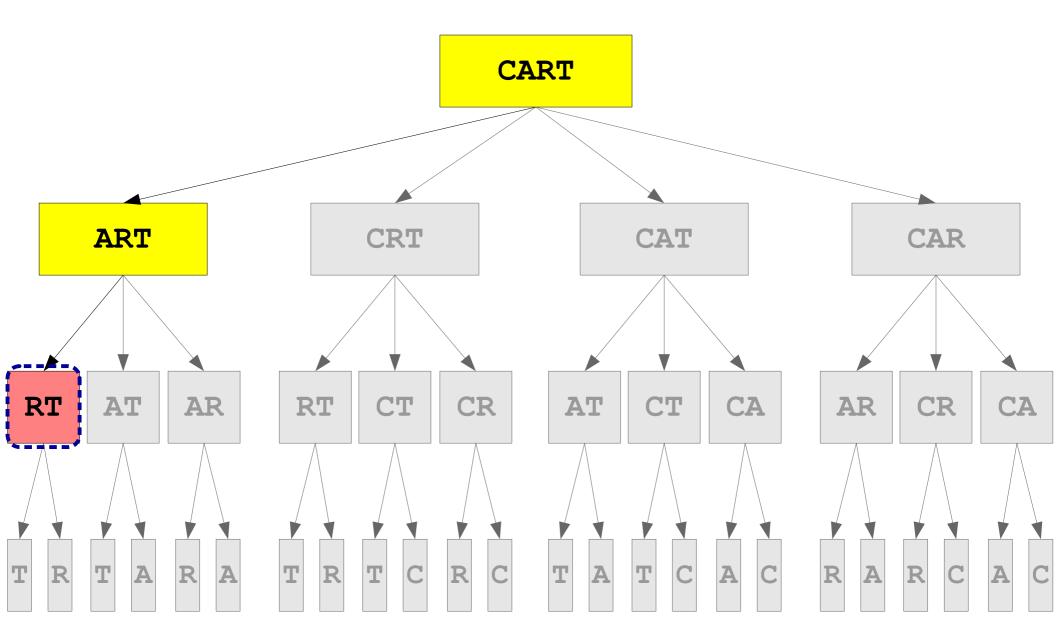


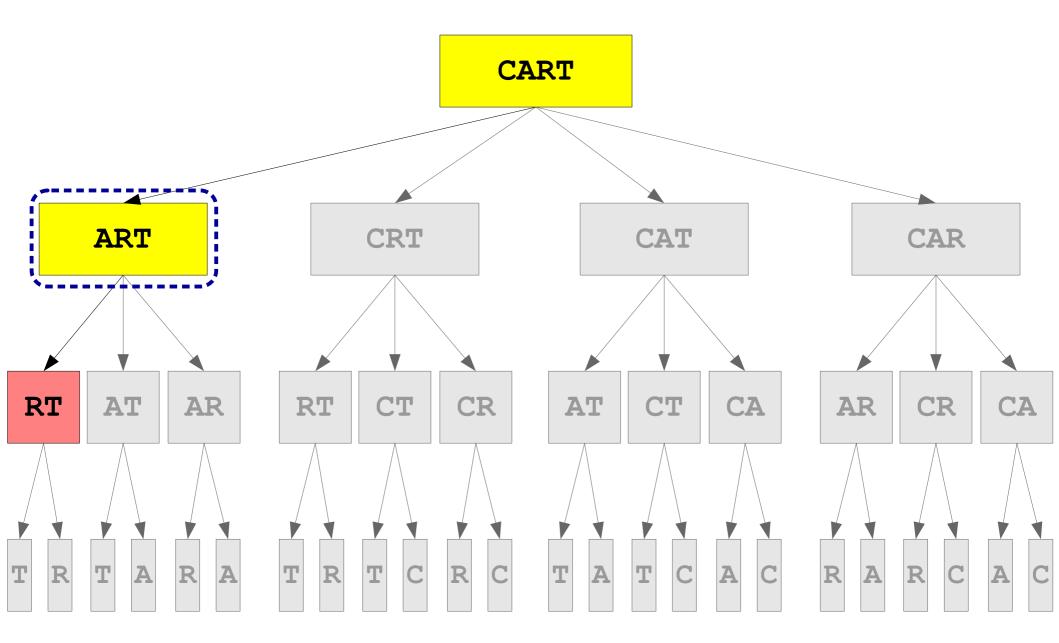


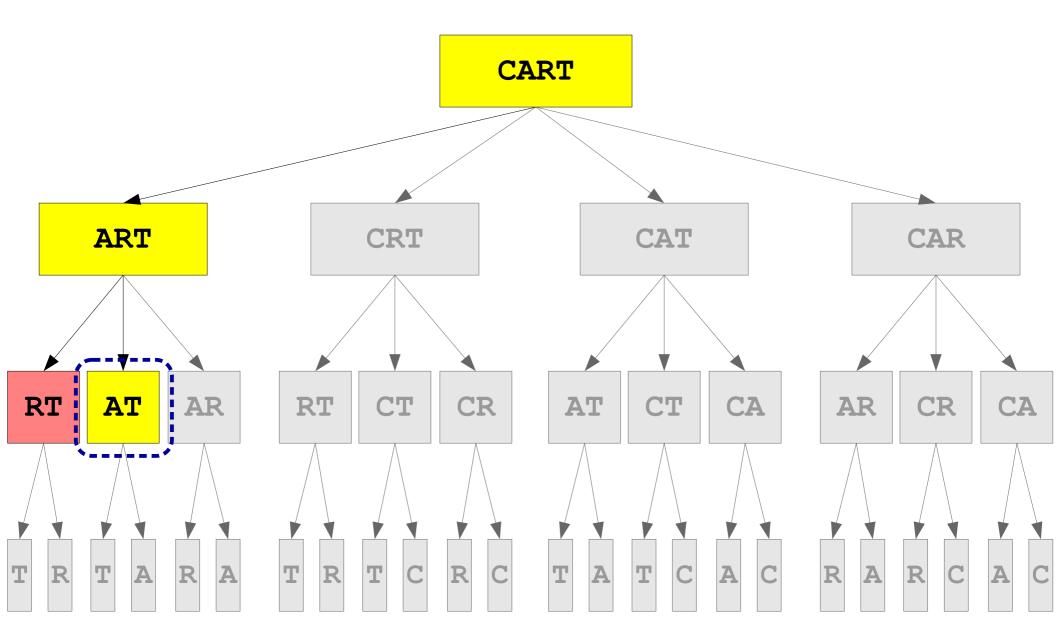


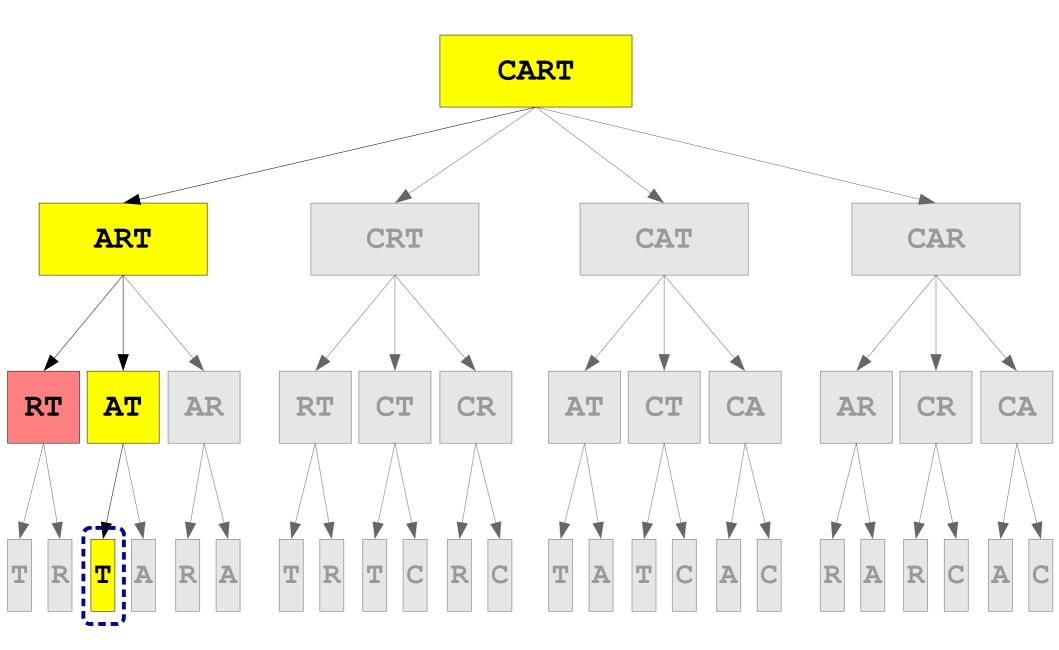


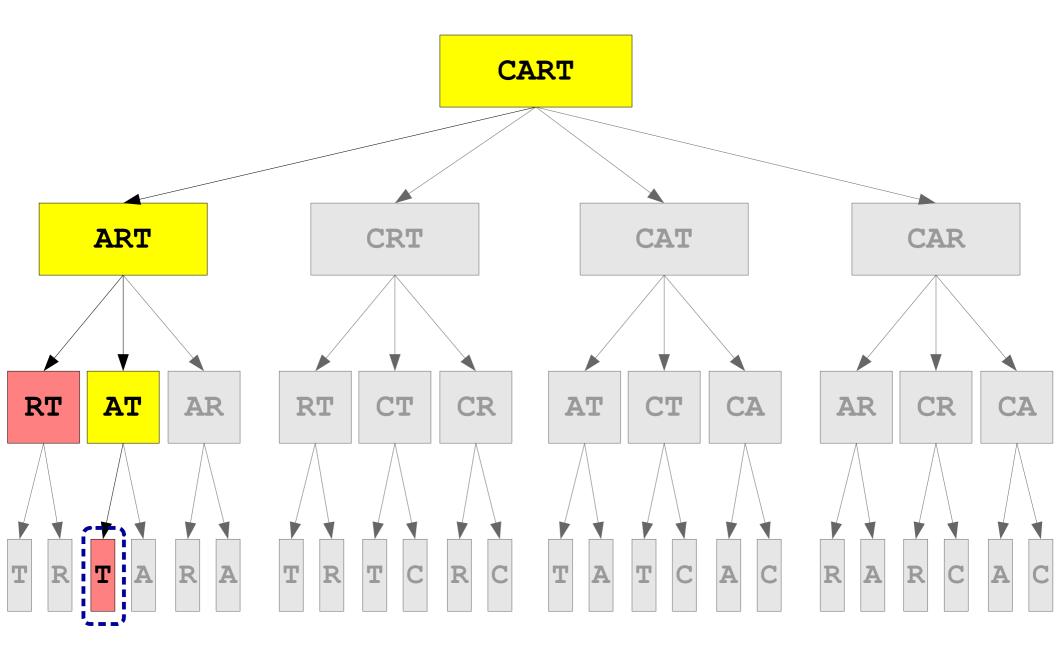


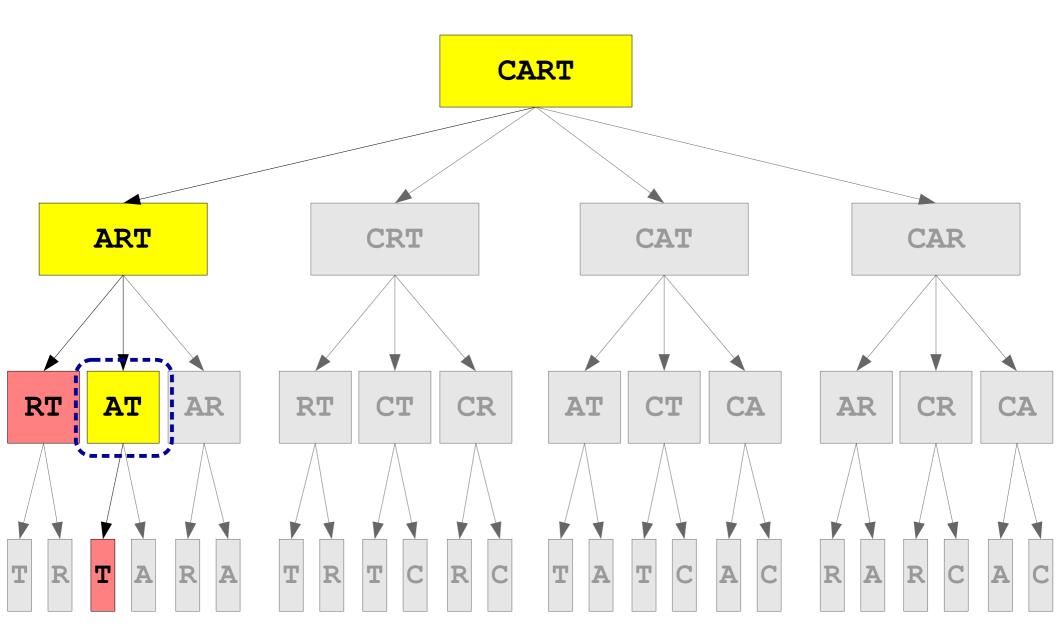


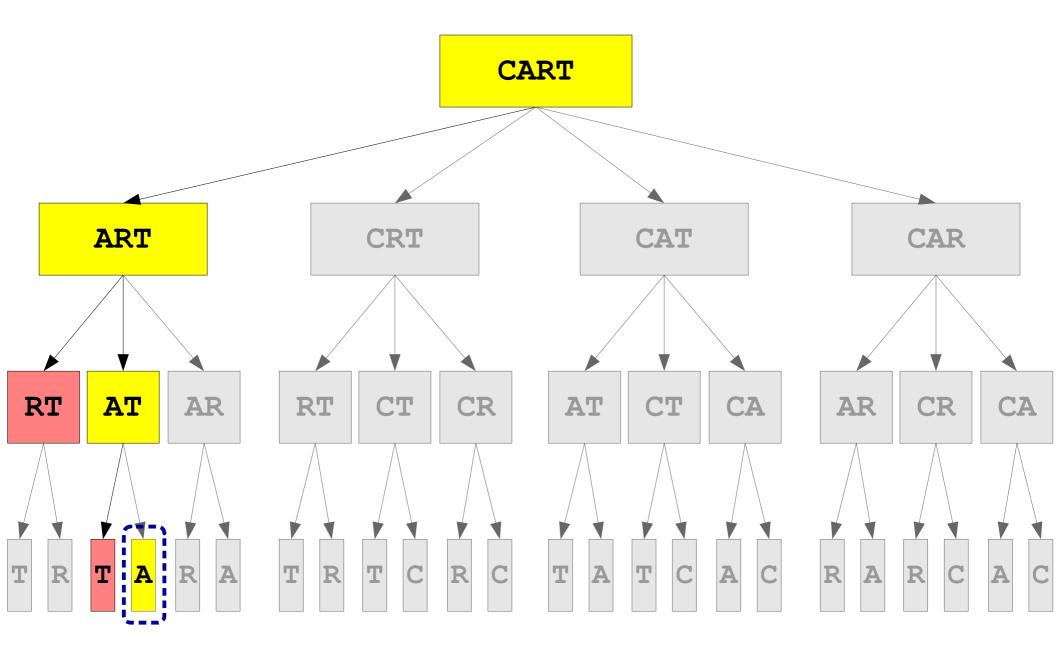


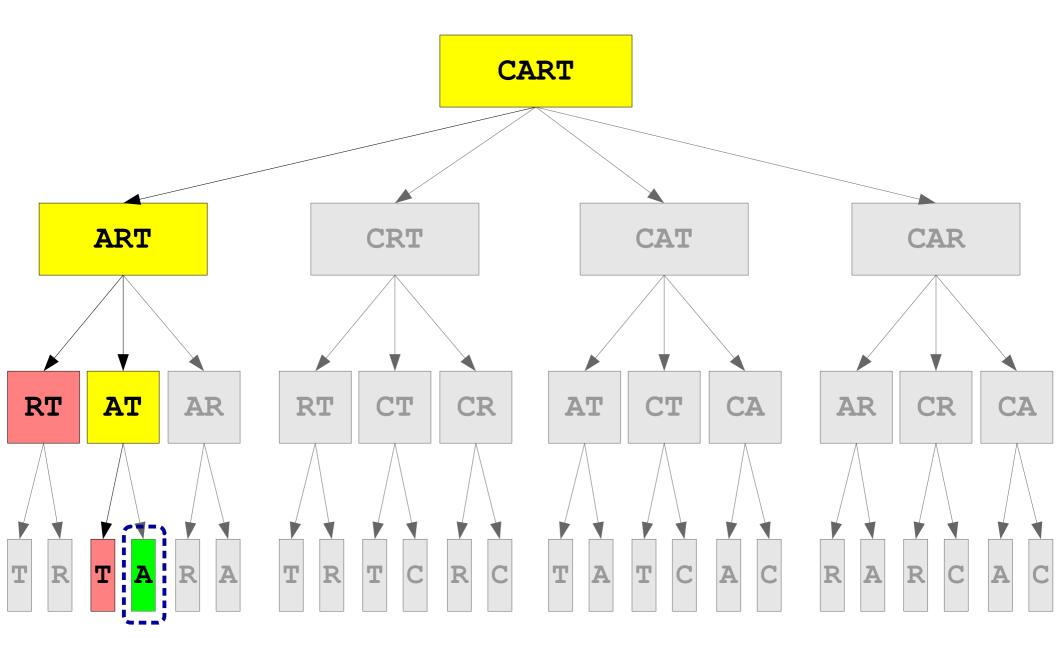


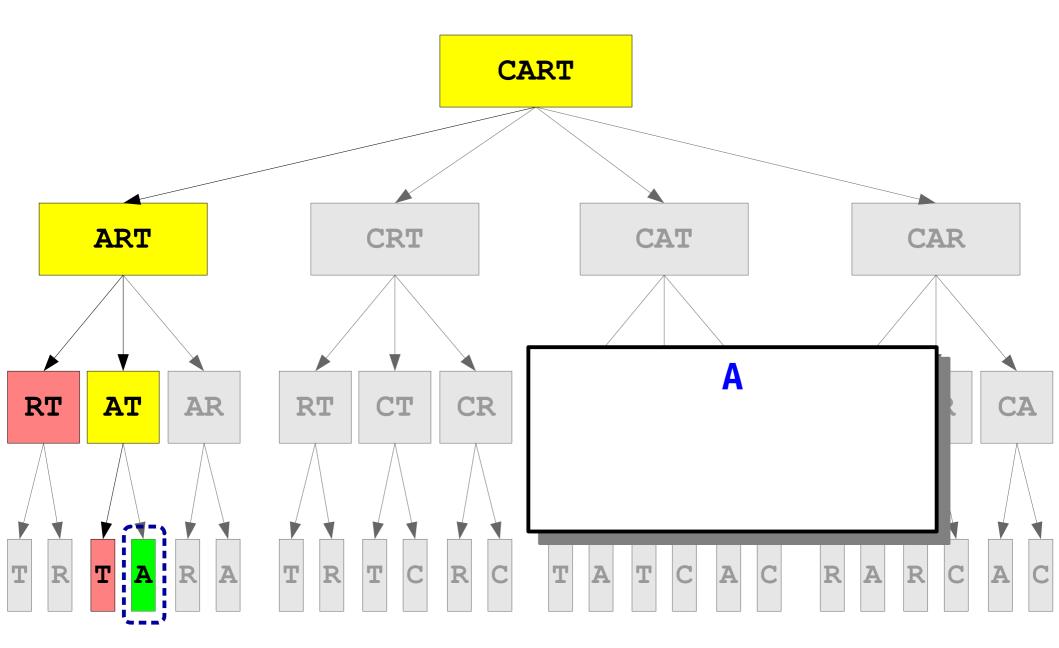


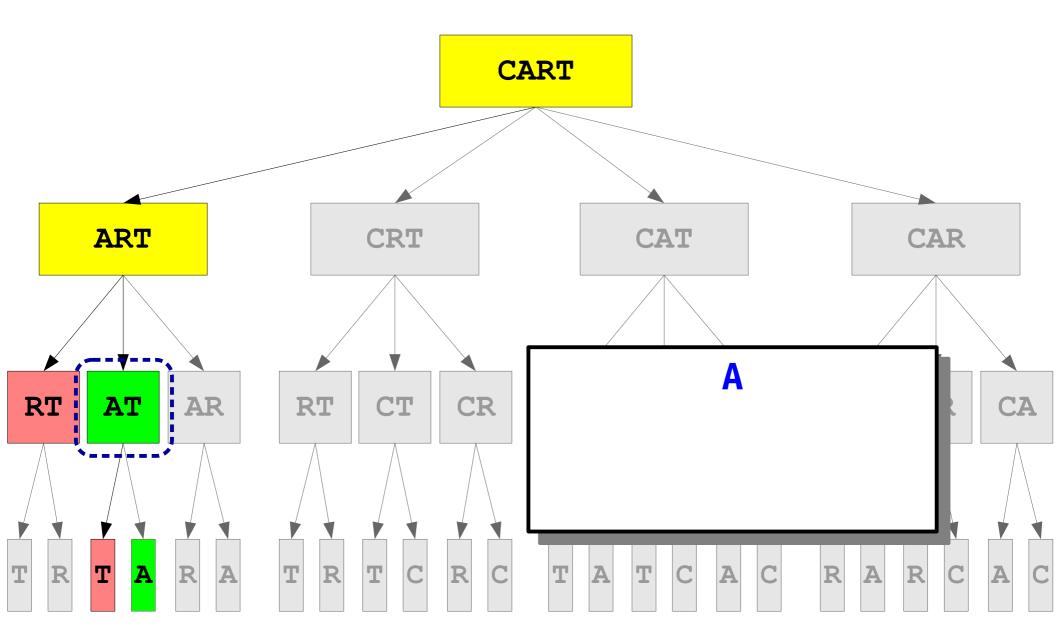


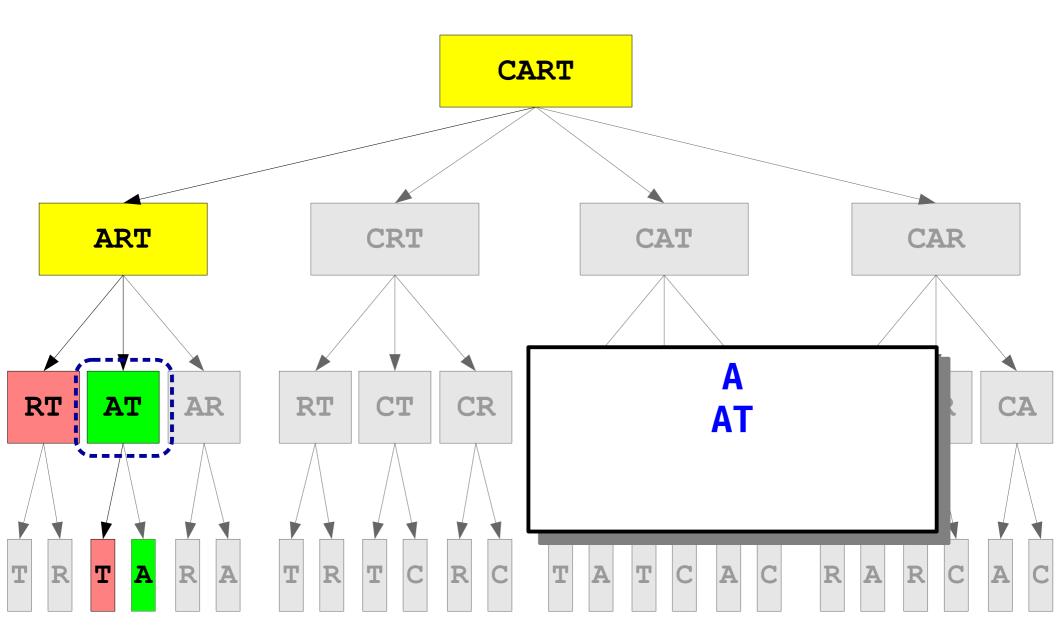


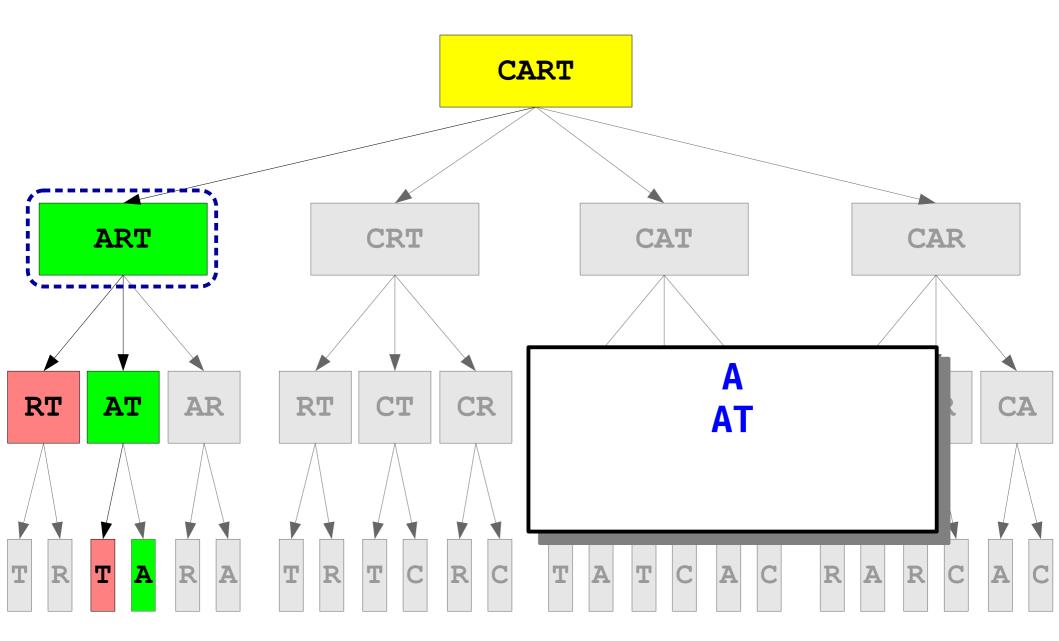


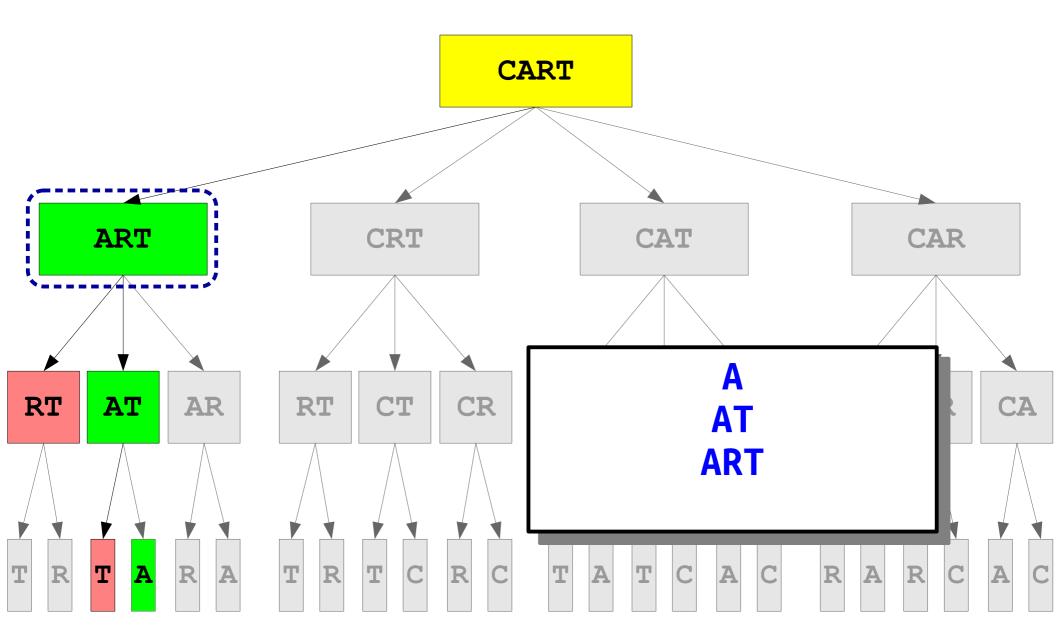


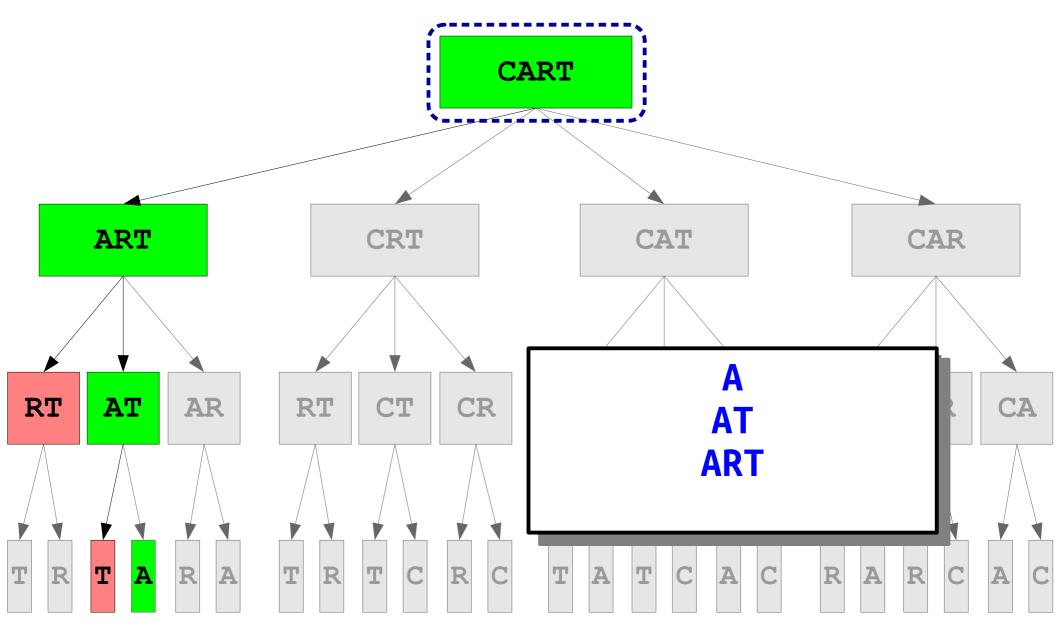


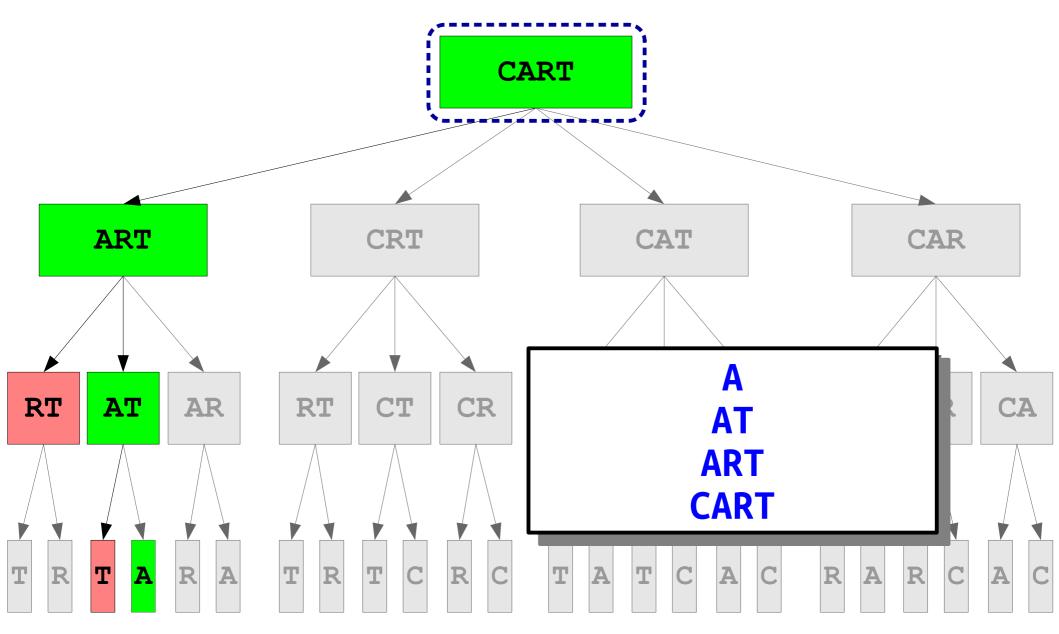












Time-Out for Announcement!

CoDa Move Continues

- Jonathan has now moved offices into the new CoDa building. Yay!
- His office hours will be in *CoDa B45*, starting today.
- Stop on by, and enjoy the CoDa building while you're there!

Back to CS106B!

Another Backtracking Example

A Great Tool of Science

Н Hydrogen 3 Be Lithium Beryllium 11 12 Mg Na Sodium Magnesi.. 19 20 21 22 23 24 25 26 27 28 29 Ti V Κ Sc Fe Co Ni Mn Ja Potassium Calcium Scandium Titanium Vanadium Chromium Mangan... Iron Cobalt Nickel 42 46 37 38 39 40 41 43 44 45 47 Rb Zr Nb Rh Pd Sr Тс Ru Mo Rubidium Strontium Yttrium Zirconium Niobium Molybde ... Techneti... Ruthenium Rhodium Palladium 55 56 57 72 73 74 75 76 77 78 79 Hf W Ba Та Re Os Pt Cs Ir La Barium Lanthan... Hafnium Tantalum Rhenium Osmium Iridium Platinum Caesium Tungsten 88 89 104 105 106 107 108 109 110 111 Sg Fr Rf Bh Mt Ra Hs Ac Db Ds Francium Radium Actinium

Aluminium Silicon Phosph.. Sulfur Chlorine Argon 30 31 32 33 34 35 36 Ga Se Br Kr Cu Zn Ge As Copper Zinc Gallium Germani... Arsenic Selenium Bromine Krypton 48 49 50 51 52 53 54 Ag Sn Cd Sb Te In Хе Silver Cadmium Indium Tin Antimony Tellurium lodine Xenon 80 81 82 83 84 85 86 Hg Pb ΤI Bi Au Po At Rn Gold Mercury Thallium Lead Bismuth Polonium Astatine Radon 112 113 114 115 116 117 118 Rg FI Nh Mc Ts Cn Lν Οq Dubnium Seaborg... Bohrium Hassium Meitneri... Darmsta... Roentge... Coperni... Nihonium Flerovium Moscovi... Livermor... Tenness... Oganes... Rutherfo...

5

13

В

Boron

A

Carbon

Si

14

2

10

18

8

16

Oxygen

S

Nitrogen

Ρ

15

9

17

Fluorine

С

He

Helium

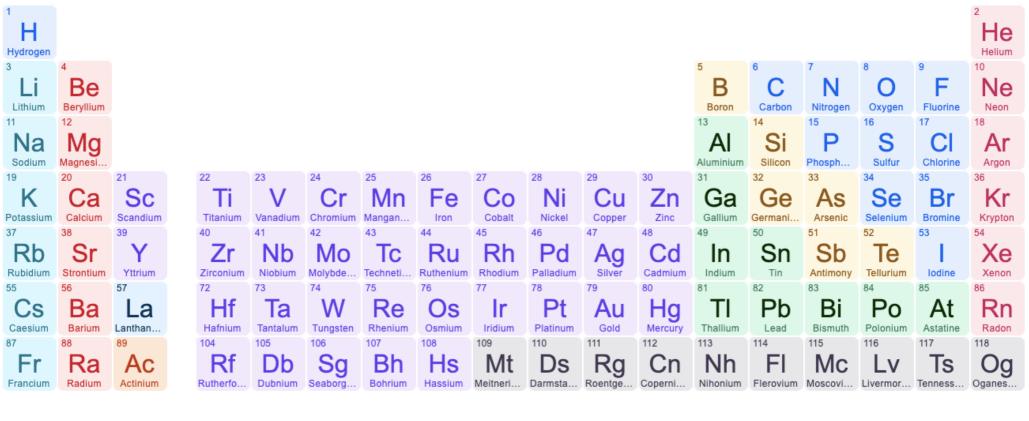
Ne

Neon

Ar

65 59 60 62 63 66 67 68 69 70 61 71 Pr Nd Pm Sm Eu Tb Dy Er Ce Gd Ho Tm Yb Lu Cerium Praseod... Neodym... Prometh... Samarium Europium Gadolini... Terbium Dysprosi... Holmium Erbium Thulium Ytterbium Lutetium 91 92 93 94 95 96 97 98 99 90 100 101 102 103 Th Pa Bk Es Fm No l r Thorium Protacti... Uranium Neptunium Plutonium Americium Curium Berkelium Californi... Einsteini. Fermium Mendele.. Nobelium Lawrenc...

Oooh! Letters!



59 62 65 66 67 68 58 60 61 63 64 69 70 71 Pr Nd Pm Sm Eu Tb Dy Ho Er Ce Gd Tm Yb Lu Praseod... Neodym... Prometh... Samarium Europium Gadolini... Terbium Dysprosi... Cerium Holmium Erbium Thulium Ytterbium Lutetium 91 92 93 94 95 96 97 98 99 100 90 101 102 103 Th Pa Bk Es Fm No l r Md Curium Thorium Protacti... Uranium Neptunium Plutonium Americium Berkelium Californi... Einsteini... Fermium Mendele... Nobelium Lawrenc...

Oooh! Letters!



Can We Do Better?



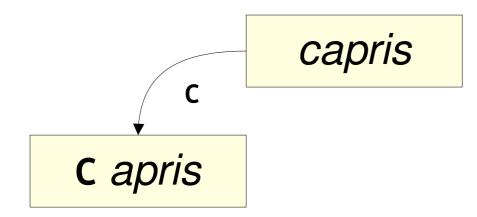
CHeMoWIZrDy

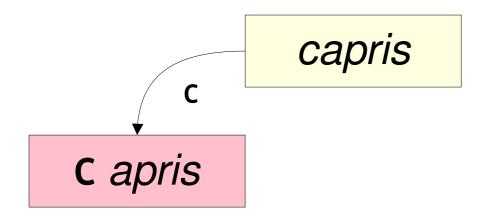
- Some words can be spelled using just element symbols from the periodic table.
- For example:

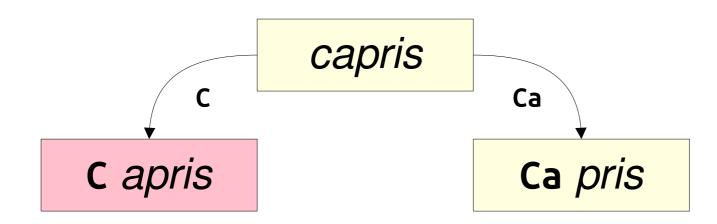
CaNiNe FeLiNe PHYSiCs UNIVErSITiEs HAlLuCINoGeNiCs

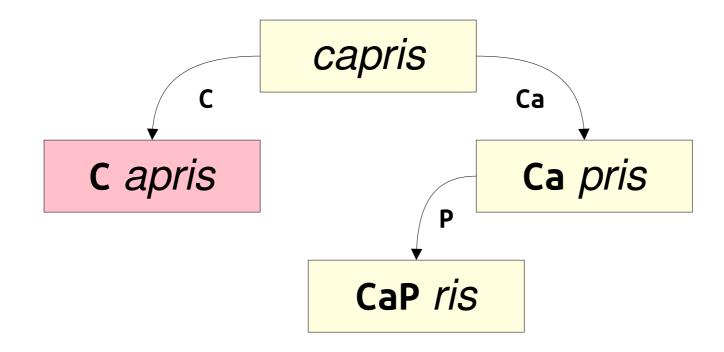
- Given a word, can we spell it using only symbols from the periodic table?
- And, if so, how?

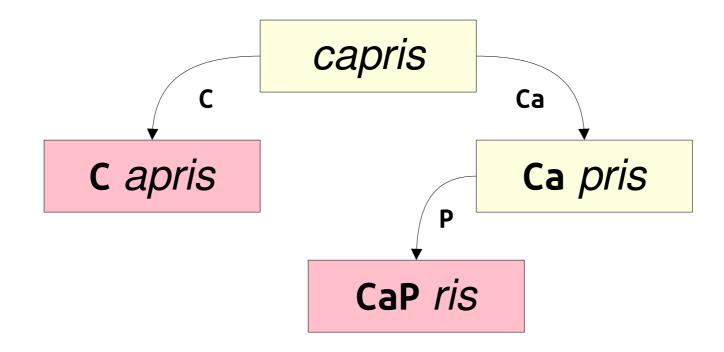
capris

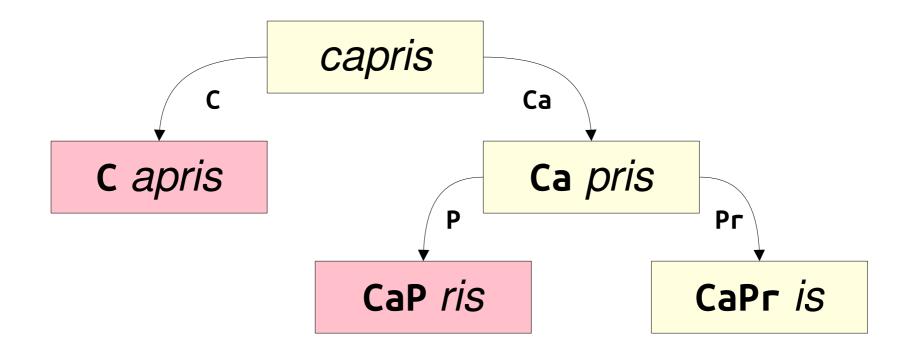


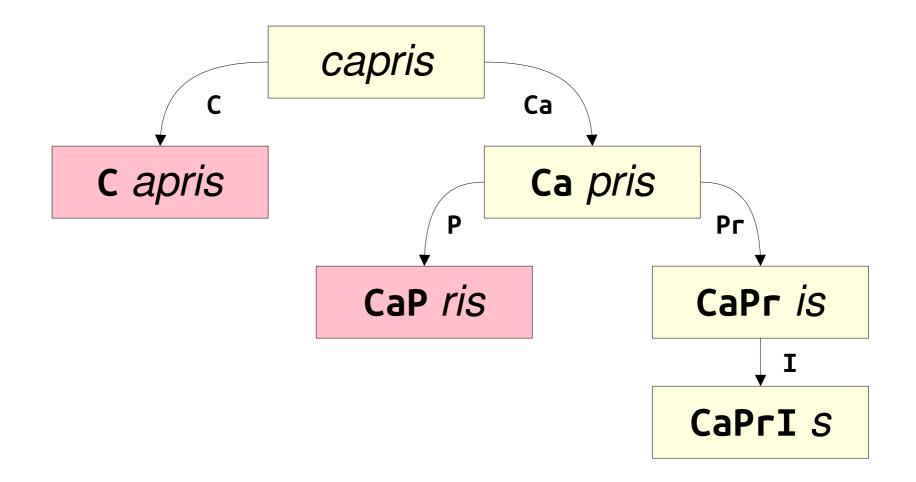


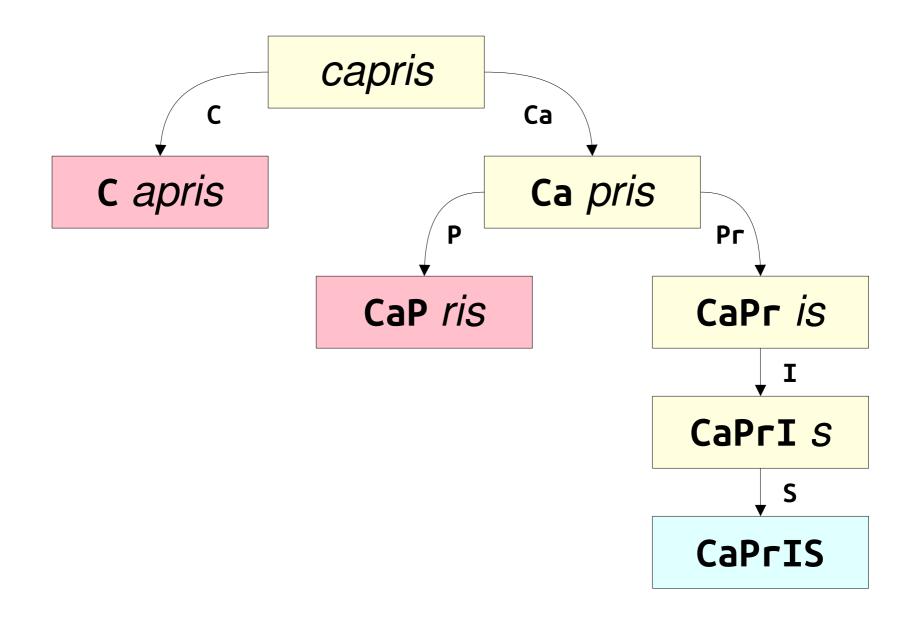












RhHeCuRhSiON

• 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!