Can a Computer Solve a Word Puzzle?
- or -
Can You Change MAN to APE?

Computers can add; they can store numbers; they can solve equations.

But word puzzles ask for abilities we don't usually associate with computers.

Can we apply computational thinking to these unusual problems?

To do so, we have to understand how we solve these puzzles, and identify those parts of our thought processes that can be "explained" to a computer.

Computational Thinking

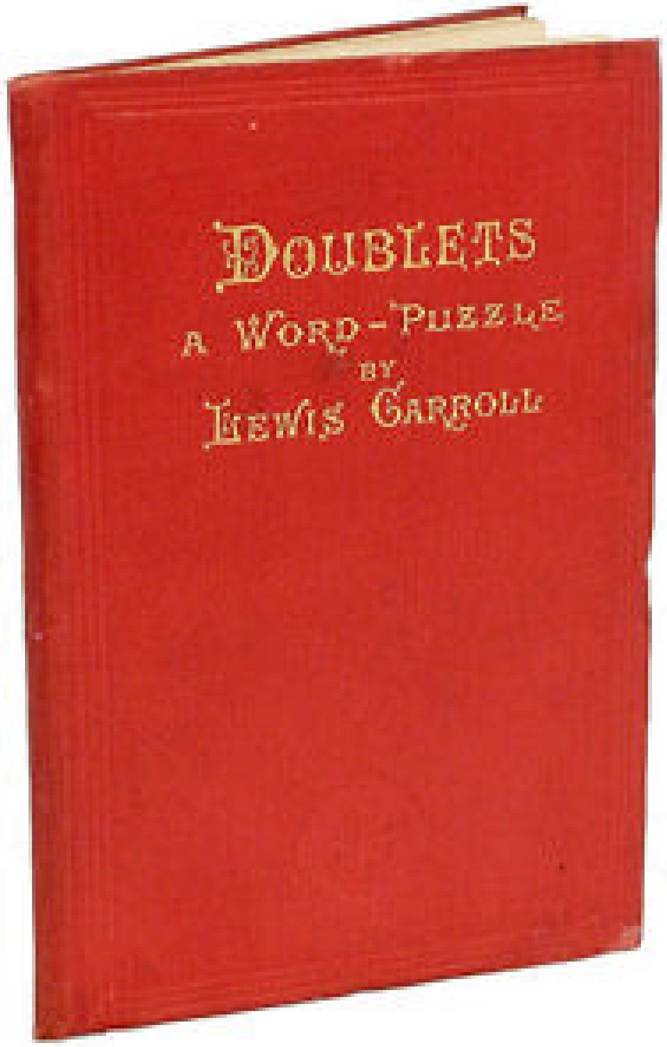
In this discussion, we will look at a simple word puzzle.

If we think about how we solve such puzzles, we can identify some mental processes (human thinking) that a computer would have to mimic:

- **memory**: we need to know a lot of words;
- **imagination**: we need to imagine possible changes to a word;
- **evaluation**: given several possible changes, we need to choose the one most likely to take us to our goal;
- **backtracking**: when a choice doesn't work out, we need to backtrack and search for an alternate choice;

If we want a computer to solve these puzzles, we have to understand how we do them first, and then try to translate our thinking into computational thinking.

DOUBLETS: Invented by Lewis Carroll



DOUBLETS: Invented by Lewis Carroll

Lewis Carroll, who wrote the children's book "Alice in Wonderland", was very fond of word games and puzzles.

He asked a riddle that no one has solved: *Why is a raven like a writing desk?*.

He wrote poems like Jabberwocky full of nonsense words, a few of which were absorbed into English: **burbled** and **gallumphing**.

And he invented a word game which he called *"Doublets"*. Lewis Carroll enjoyed asking friends to try it, saying "Do you know how to turn MAN into APE?"

After getting a puzzled look, he would say: "But it's so easy!"

DOUBLETS: Can you change MAN to APE?

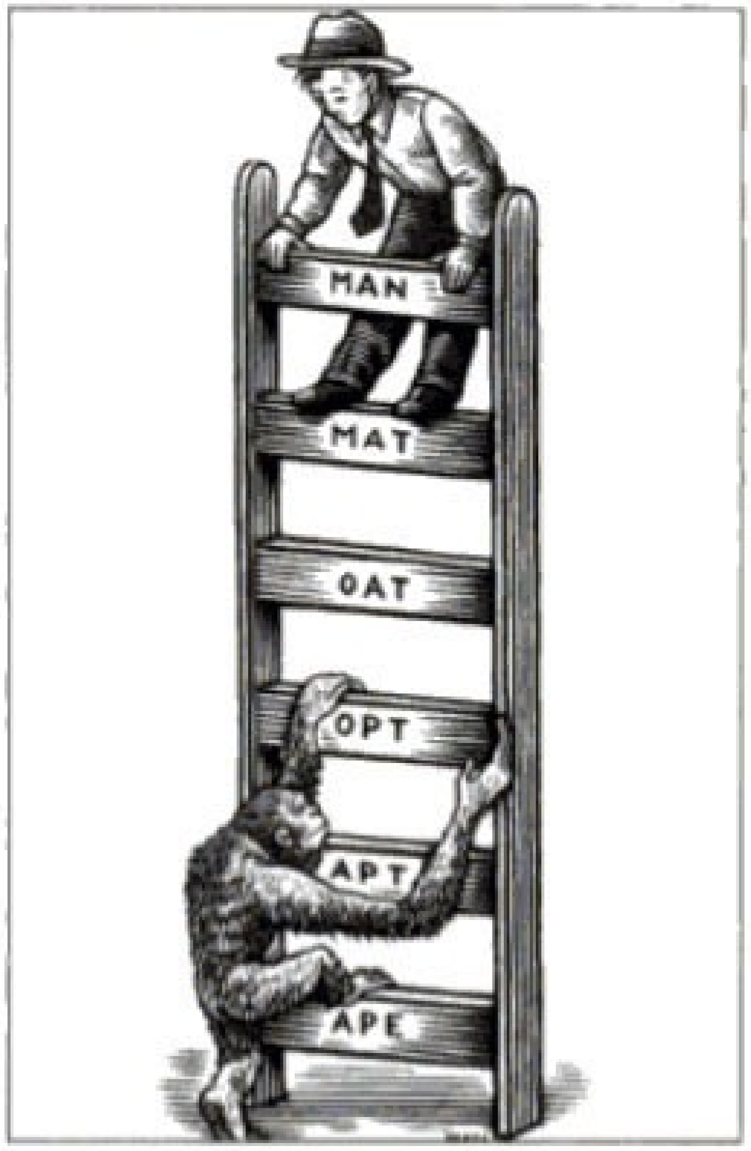


Illustration by Gregory Nemec

DOUBLETS: The Rules

The rules for doublets are:

1. The puzzle consists of a START word and an END word.
2. The solution is a sequence of legal English words connecting START to END.
3. Each word in the sequence must differ by just one letter from its neighbors.
4. The “distance” between the words is the number of steps we used.

```
MAN
MAT (change N to T)
OAT (change M to O)
APT (change A to P)
APE (change T to E)
```

Distance = 4, because we took 4 steps to transform MAN into APE.

DOUBLETS: The Rules

Here are some consequences of the doublet rules:

1. If the START and END words are the same, we're done (distance $= 0$)
2. The START and END words must have the same number of letters.
3. If the START and END words differ in each position, and they are N letters long, then we will need **at least** N steps to solve the puzzle.
4. For some START and END words, the puzzle might be impossible.
5. The puzzle would be a cinch if we could use "words" that weren't English!
6. If we swap the START and END words, the puzzle is just as hard (or easy)
7. If someone has a solution, but uses the same intermediate word more than once, we can make a shorter solution.
8. It's very tempting to seek a solution by trying to replace letters of START with letters of END.

DOUBLETS: We don't know how we solve them

The Doublets rules are simple, but the process of solving a puzzle is a little mysterious.

If you are given a puzzle, sometimes you must see an answer right away, and have no idea how you got it.

Other times, you might have to write down some guesses, or say "Is that a word?" or give up on one idea and try another.

After you've solved a puzzle, you could probably report most of the ideas you had, but you wouldn't be able to give someone much advice on how to solve a new puzzle.

So, unlike the tasks we know computers are good at, it's not obvious that a word game like Doublets would be suitable for a computer to solve.

DOUBLETS: Examples posed by Lewis Carroll

Drive PIG into STY.
Raise FOUR to FIVE.
Make WHEAT into BREAD.
Dip PEN into INK.
Touch CHIN with NOSE.
Change WET to DRY.
Make HARE into SOUP.
PITCH TENTS.
Cover EYE with LID.
Prove PITY to be GOOD.
STEAL COINS.
Make EEL into PIE.
Turn POOR into RICH.
Prove RAVEN to be a MISER.
Change OAT to RYE.
Get WOOD from TREE.
Prove GRASS to be GREEN.
Evolve MAN from APE.
Change CAIN into ABEL.

Make FLOUR into BREAD.
Make TEA HOT.
Run COMB into HAIR.
Prove a ROGUE to be a BEAST.
Change ELM into OAK.
Combine ARMY and NAVY.
Place BEANS on SHELF.
HOOK FISH.
QUELL a BRAVO.
Stow FURIES in a BARREL.
BUY an ASS.
Get COAL from MINE.
Pay COSTS in PENCE.
RAISE ONE to TWO.
Change BLUE to PINK.
Change BLACK to WHITE.
Change FISH to BIRD.
Sell SHOES for CRUST.
Make KETTLE HOLDER.

DOUBLETS: Class exercise

In order to get a feel how how your mind works on this kind of problem, let's try a few of these puzzles in class:

- Change WET to DRY.
- Raise FOUR to FIVE.
- Raise COLD to WARM.
- Make WHEAT into BREAD.

DOUBLETS: Class exercise remarks

Longer words are harder...because there are so many more possibilities.

Changing a consonant to a vowel is hard.

One of these puzzles had an easy solution; we could simply swap in the new letters, one at a time.

At the beginning, there was almost no way to tell which idea was going to work out; our search was very disorganized.

Once a chain of words got started, the problem got easier as we got closer to the end.

We probably have little idea how a computer could attack such problems.

DOUBLETS: Sometimes there can be many solutions

Here are three attempts to turn MAN into APE, taking 9, 6 and 4 steps:

```
MAN      MAN      MAN
MAY      MAR      MAT
PAY      EAR      OAT
PAT      ERR      APT
PIT      ERE      APE
PIE      ARE
DIE      APE
DYE
AYE
APE
```

DOUBLETS: Did we get the best (shortest) solution?

If we start with a 4 letter word, and we have to get to another four letter word, and no letters are the same in the two words, then we will need at least 4 steps.

So this gives us at least a limit on how low the “distance” can be.

It also suggests that one way to get started is just to see whether you can swap in some letter from the end word. If that works, we can try swapping in another, and so on. But that won’t work very often!

When we do find a solution, we often don’t know if there is a shorter one.

As we can see, in the MAN to APE example, it’s easy to drag out the solution, although the best solution is very short.

DOUBLETS: The “stay connected” rule

Without thinking about it, one strategy we used was to work from the START or END word. We didn’t simply pick a random word and make moves from it.

If someone had looked at the MAN to APE problem, and said “I’ll start at the word PIT and see where I can get!” we would think that person was crazy.

It turns out you can get a solution that goes through PIT, but that’s not the way any human would work on the problem. It just doesn’t seem right.

So although we can’t necessarily explain why we do it, we have a heuristic or “rule of thumb” that suggests that a good way to seek solutions is by working away from the START or END word, and trying to make one move at a time, getting closer to the other word if you can.

HEURISTIC: a simple guide for decision-making that usually identifies one of the best choices.

DOUBLETS: Consonant/Vowel Rule

Looking at our experiences with the puzzles, another thing we might notice is that, when taking a single step (changing one letter), there are many choices to make if we are changing one consonant to another, and usual a few choices for changing one vowel to another.

But if we want to change a consonant to a vowel, or the other way around, there are often no choices, or just one possibility.

So comparing the START and END words, we can see in advance how many of these difficult steps we are going to have to manage.

Changing MAN to APE involves the maximum number of such switches, a nightmare in which every vowel must become a consonant and vice versa.

DOUBLETS: Consonant/Vowel Rule

Consonant/Vowel Rule: switching consonants to vowels is hard. If you see an opportunity, always try it!

MAN
MAT
PAT
PIT
PIE $\leftarrow$ consonant "T" switches to vowel "E"
DIE
DYE $\leftarrow$ let's not worry about whether "Y" is a consonant or vowel!
AYE $\leftarrow$ consonant "D" switches to vowel "A"
APE $\leftarrow$ vowel "Y" switches to consonant "P"

DOUBLETS: Sometimes there can be no solution

Sometimes we can see there's no answer. Can you change SWEET to ALOOF? You could try and try for days getting nowhere on this example!

But if you look carefully at the word ALOOF, you will see there's a problem. There is no way to change just one letter in ALOOF to get another common word in the English language.

(BLOOF is not a word, nor is CLOOF, or DLOOF or ... or ALOOX or ALOOY or ALOOZ.)

That means there can't be any way to solve any doublets problem if ALOOF is the START or END word.

Here are some more impossible Doublets:

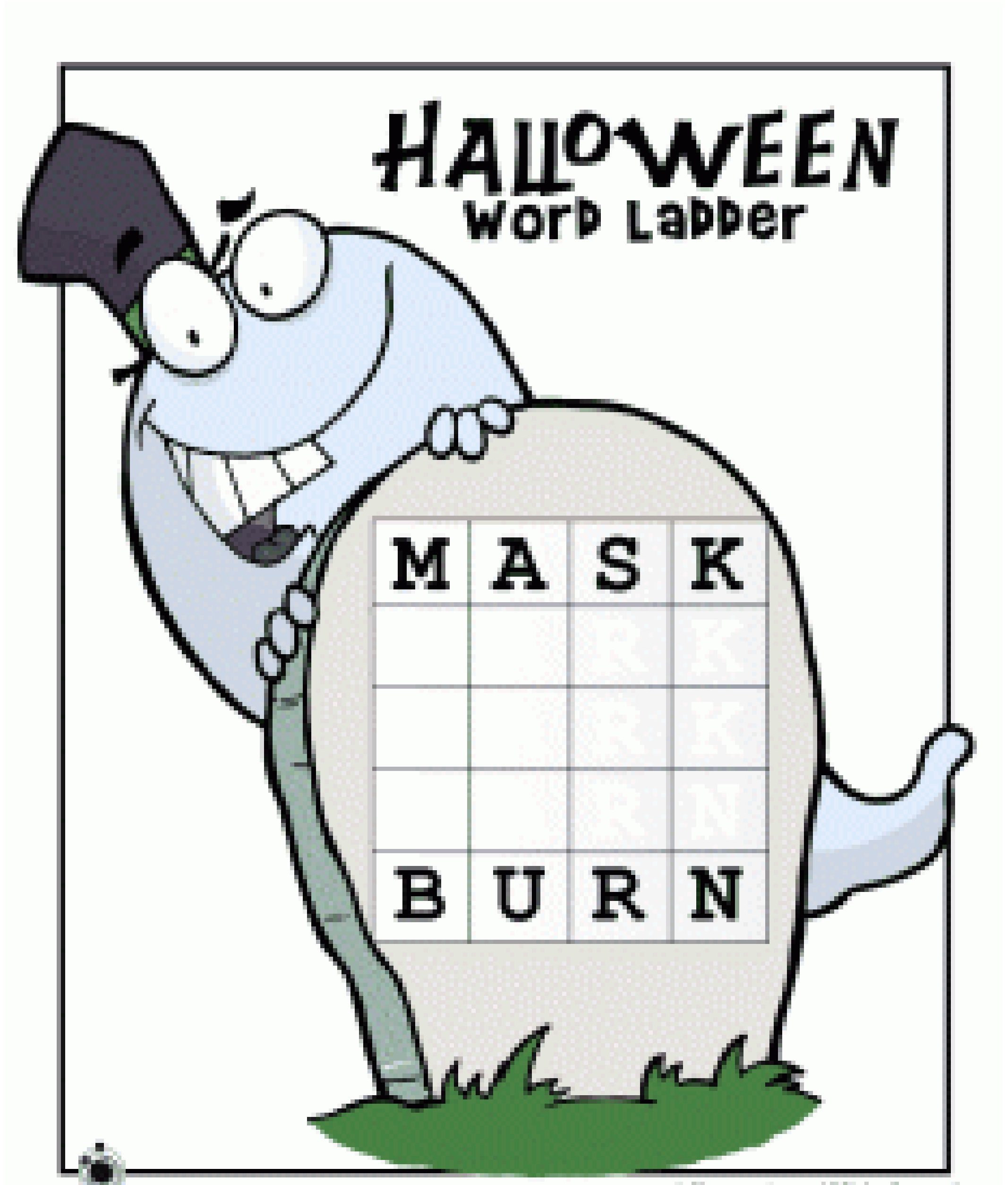
```
Transform IRON into LEAD.   (no!)
Move FIRST to THIRD.        (you can't!)
From BELOW go ABOVE.        (don't try it!)
```

DOUBLETS: Can a computer handle word puzzles?

So if we think about giving these kind of word problems to a computer, there are some issues:

- The computer doesn't know English.
- The computer can't look at the START and END words and simply make a good guess for a connecting word.
- We can feel when we're getting closer. The computer needs some way to estimate whether it's getting closer.
- Since the problem might have no solution, perhaps the computer will end up in an infinite loop. But we said an algorithm had to have a definite stopping point.

DOUBLETS: Let’s try another example



DOUBLETS: MASK to BURN uses a “Greedy” algorithm

In this version of a Doublet puzzle, we see that we need to transform one four letter word into another using 3 steps, which is the minimum possible when the words have no matching letters.

One solution is **MASK, BASK, BARK, BARN, BURN**.

This is another example of a solution in which we simply swap out one old letter for one new one on each step. So the only difficulty is figuring out which letter to swap, and this gets easier as we get closer to the final word.

This kind of solution method (which only works occasionally) is called a greedy algorithm; it’s like an impatient child, trying to go directly to the goal as fast as possible.

DOUBLETS: Sporcle includes hints

Sporcle at **www.sporcle.com** is a puzzle and game web site that offers many versions of Doublet puzzles.

Instead of giving a start and end word, it lists a sequence of clues.

The words that are clued form a doublet sequence.

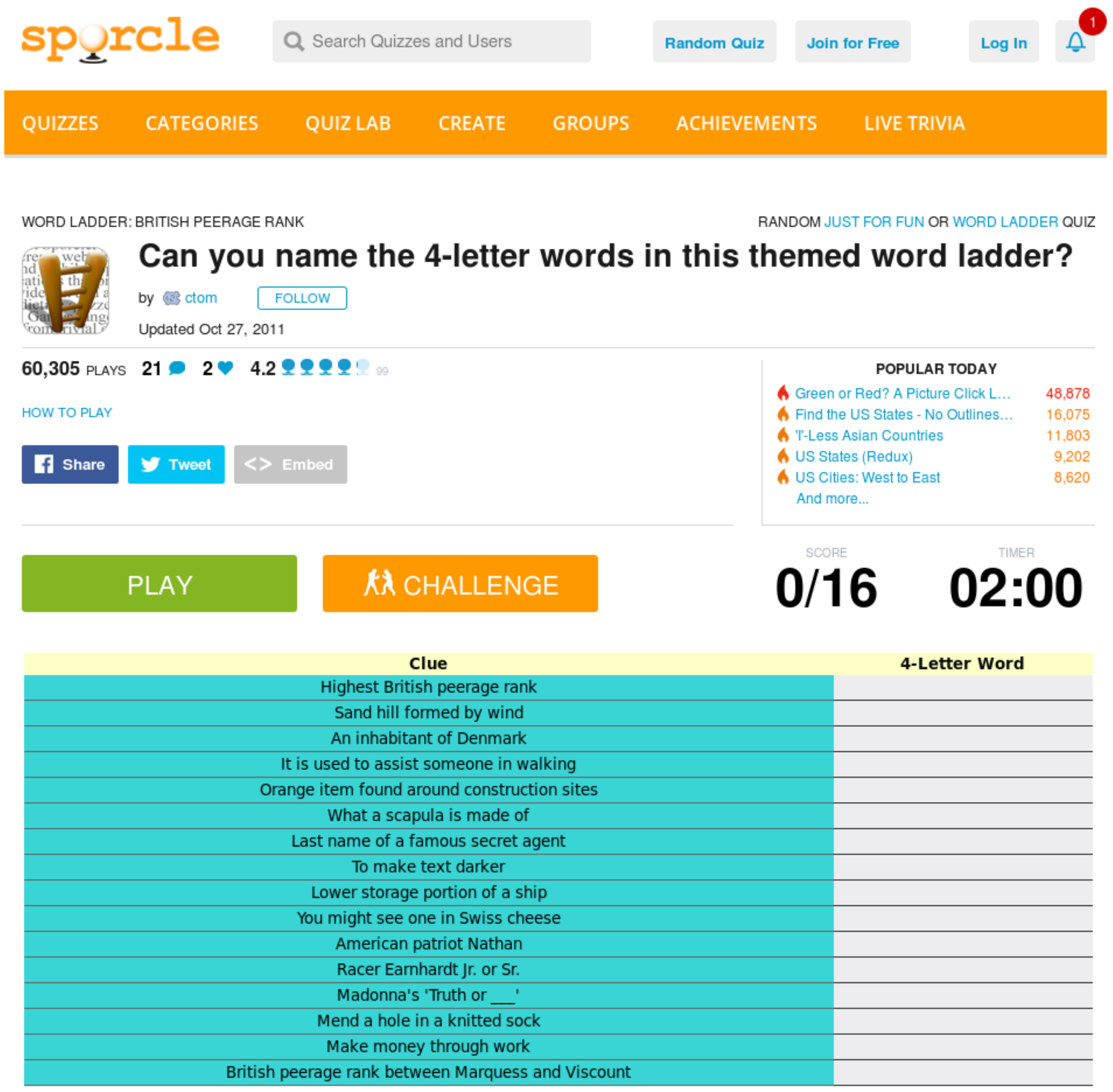
So you can sometimes fill in a word from the clue.

If the clue doesn't help, sometimes you can get a word just because you have the neighboring word in the double sequence.

Thus this puzzle doesn't require the "stay connected" rule; it can be much easier to solve.

Let's look at a typical example.

DOUBLETS: A typical Sporcle puzzle



sporcle   Search Quizzes and Users   Random Quiz   Join for Free   Log In   1

QUIZZES   CATEGORIES   QUIZ LAB   CREATE   GROUPS   ACHIEVEMENTS   LIVE TRIVIA

WORD LADDER: BRITISH PEERAGE RANK

RANDOM JUST FOR FUN OR WORD LADDER QUIZ

Can you name the 4-letter words in this themed word ladder?

by ctom   FOLLOW

Updated Oct 27, 2011

60,305 PLAYS   21   2   4.2   99

HOW TO PLAY

Share   Tweet   Embed

POPULAR TODAY

Green or Red? A Picture Click L...	48,878
Find the US States - No Outlines...	16,075
'I'-Less Asian Countries	11,803
US States (Redux)	9,202
US Cities: West to East	8,620
And more...	

PLAY   CHALLENGE

SCORE 0/16   TIMER 02:00

Clue	4-Letter Word
Highest British peerage rank	
Sand hill formed by wind	
An inhabitant of Denmark	
It is used to assist someone in walking	
Orange item found around construction sites	
What a scapula is made of	
Last name of a famous secret agent	
To make text darker	
Lower storage portion of a ship	
You might see one in Swiss cheese	
American patriot Nathan	
Racer Earnhardt Jr. or Sr.	
Madonna's 'Truth or ___'	
Mend a hole in a knitted sock	
Make money through work	
British peerage rank between Marquess and Viscount	

DOUBLETS: Sporcle class exercise

1	Highest British peerage rank	- - - -
2	Sand hill formed by wind	- - - -
3	An inhabitant of Denmark	- - - -
4	It is used to assist someone in walking	- - - -
5	Orange item found around construction sites	- - - -
6	What a scapula is made of	- - - -
7	Last name of a famous secret agent	- - - -
8	To make text darker	- - - -
9	Lower storage portion of a ship	- - - -
10	You might see one in Swiss cheese	- - - -
11	American patriot Nathan	- - - -
12	Racer Earnhardt Jr or Sr	- - - -
13	Madonna's Truth or	- - - -
14	Mend a hole in a knitted sock	- - - -
15	Make money through work	- - - -
16	British peerage rank between Marquess and Viscount	- - - -

DOUBLETS: Sporcle hints are a huge help

The hints that Sporcle provides make a huge difference in solving a doublet puzzle.

Imagine if, instead, you'd been asked to solve a doublet puzzle and simply told to start at DUKE and end at EARL.

We certainly wouldn't have raced through the puzzle like we did here.

We wouldn't have filled in some of the intermediate steps early.

We would have worked from the START or END, and made many false steps while searching for the answer.

I'm not even sure how long it would have taken to find the answer.

You can almost imagine that a computer, if it had a dictionary available, would have a chance to solve this puzzle as fast as we did.

DOUBLETS: Look at the greedy algorithm again

Remember two doublet puzzles that were very easy: Turn COLD to WARM, and MASK to BURN,

`COLD`	`MASK`
`WARM`	`BURN`

DOUBLETS: Look at the greedy algorithm again

These particular puzzles can be solved by the greedy method, where we simply look at the word we're trying to reach, and take the simplest possible step, by replacing one letter of the start word with a letter of the target word.

Starting from **COLD**, our first greedy step checks whether **WOLD**, **CALD**, **CORD** or **COLM** is a word. Just one choice works.

Since that gets us one letter closer, we risk it, and move to **CORD**.

Then we try changing another letter of **CORD** to match a letter in **WARM**, and we get three possiblities, **WORD**, **CARD**, or **CORM**.

Both **WORD** and **CARD** are legal words. So we choose one of these, but we remember the other choice. If we hit a dead end using **WORD**, then we can backtrack to **CARD** and try that.

Let's look at the complete process for both our puzzles.

DOUBLETS: Solutions by the greedy algorithm

COLD to WARM	MASK to BURN
COLD	MASK
CORD	BASK
WORD	BARK
WARD	BARN
WARM	BURN

DOUBLETS: Always check the greedy step

It is surprising to see that this puzzle can be done simply by swapping one letter at a time.

A greedy person might stumble on this strategy, saying "Let's go for the goal right away! The fastest way is to swap in a target letter on every move!"

The greedy algorithm strives for an immediate obvious payoff. In this example, it reaches the target word by taking a greedy step every time.

Most puzzles aren't so simple. However, remember that our biggest problem in solving a doublet is trying to find any move at all. The greedy algorithm always has a few suggestions for us. Since these choices, if they work, are guaranteed to move us closer to the target word, they are always worth investigating.

So even though the greedy algorithm isn't a guarantee of a solution, it is another heuristic, or rule of thumb, that gives us an idea of how to search efficiently for the next step.

DOUBLETS: Greedy algorithm class exercise

Try the greedy algorithm on these doublets.

The greedy algorithm says, “Always use the greedy step.”

```
LEAF    RICH    COME
. . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . . .
WORD    DUNE    SALT
```

DOUBLETS: The greedy algorithm sometimes works

LEAF	RICH	COME
LEAD	RICE	SOME
LOAD	DICE	SAME
LORD	DINE	SALE
WORD	DUNE	SALT

DOUBLETS: Greedy step class exercise

In these problems, the greedy step will usually work, but not always.

```
MAD     HEAD    HARD    RISE
...     ....    ....    ....
...     ....    ....    ....
...     ....    ....    ....
DOG     ....    ....    ....
        TAIL    EASY    ....
                        FALL
```

DOUBLETS: Greedy step class exercise

MAD	HEAD	HARD	RISE
LAD	HEAL	HARE	RITE
LAG	TEAL	BARE	MITE
LOG	TELL	BASE	MILE
DOG	TALL	EASE	FILE
	TAIL	EASY	FILL
			FALL

In these puzzles, the greedy step doesn't always work. We have to take more steps than usual, and occasionally swap in a letter that later we have to swap back out.

Nonetheless, it's worth it to try the greedy step before we just start making guesses.

Socrative Quiz PartIV_Practice_Quiz1

IUZGAZ34E

1. Is (**arms, rams, mars, maps**) a legal doublet? (Yes, No)
2. Is (**ears, earn, warn, worn, worm, warm**) a legal doublet? (Yes, No)
3. Is (**gray, bray, blay, bluy, blue**) a legal doublet? (Yes, No)
4. Is (**mats, bats, bars, barn, born**) an example in which every step is greedy? (Yes, No)
5. Can any four letter word be transformed into any other four letter word using doublets? (Yes, No)
6. The words **bat** and **tab** have the same letters but in reverse order. Does this mean it is impossible to convert **bat** to **tab**? (Yes, No)

SAGE $\rightarrow$ FOOL: A harder puzzle

So far, we've been able to solve a doublets puzzle pretty quickly, using our memory of all English words, by guessing, and by using the greedy algorithm, by backtracking when one possibility becomes a dead end, and by a kind of brute force approach where we just start coming up with every possible connecting word we can think of.

So we understand a little bit about how we think about solving this problem, which is important if we want to try to have a computer solve it for us.

But so far, our problems have been simple and short. To get a feeling for how complicated a solution can be, we're going to look at a problem that looks simple - just 4 letter words - but which requires a significantly longer chain of connecting words.

Once we solve the puzzle, we will see a way of organizing the problem, which will give us a kind of map that makes it clear how far apart the START and END words are.

SAGE $\rightarrow$ FOOL

One of Lewis Carroll's puzzles asks us to turn SAGE into FOOL.

Let's try to think about how we might solve such a puzzle.

Our first response is to try a greedy step, that is, swapping a letter of **SAGE** for one of **FOOL**...after all, we have to do that eventually.

However, we can see that **FAGE**, **SOGE**, **SAOE** and **SAGL** are not words, so we can't take a greedy step right away.

SAGE $\rightarrow$ FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & \text{FAGE} \ \textit{Not a word!} \\
& \text{SOGE} \ \textit{Not a word!} \\
& \text{SAOE} \ \textit{Not a word!} \\
& \text{SAGL} \ \textit{Not a word!}
\end{tikzcd}$$

$$\boxed{\text{FOOL}}$$

SAGE $\rightarrow$ FOOL

So the next thing to consider is ... what words **can** we jump to from **SAGE**?

Changing the first letter of **SAGE** gives us CAGE, MAGE, PAGE, RAGE, WAGE.

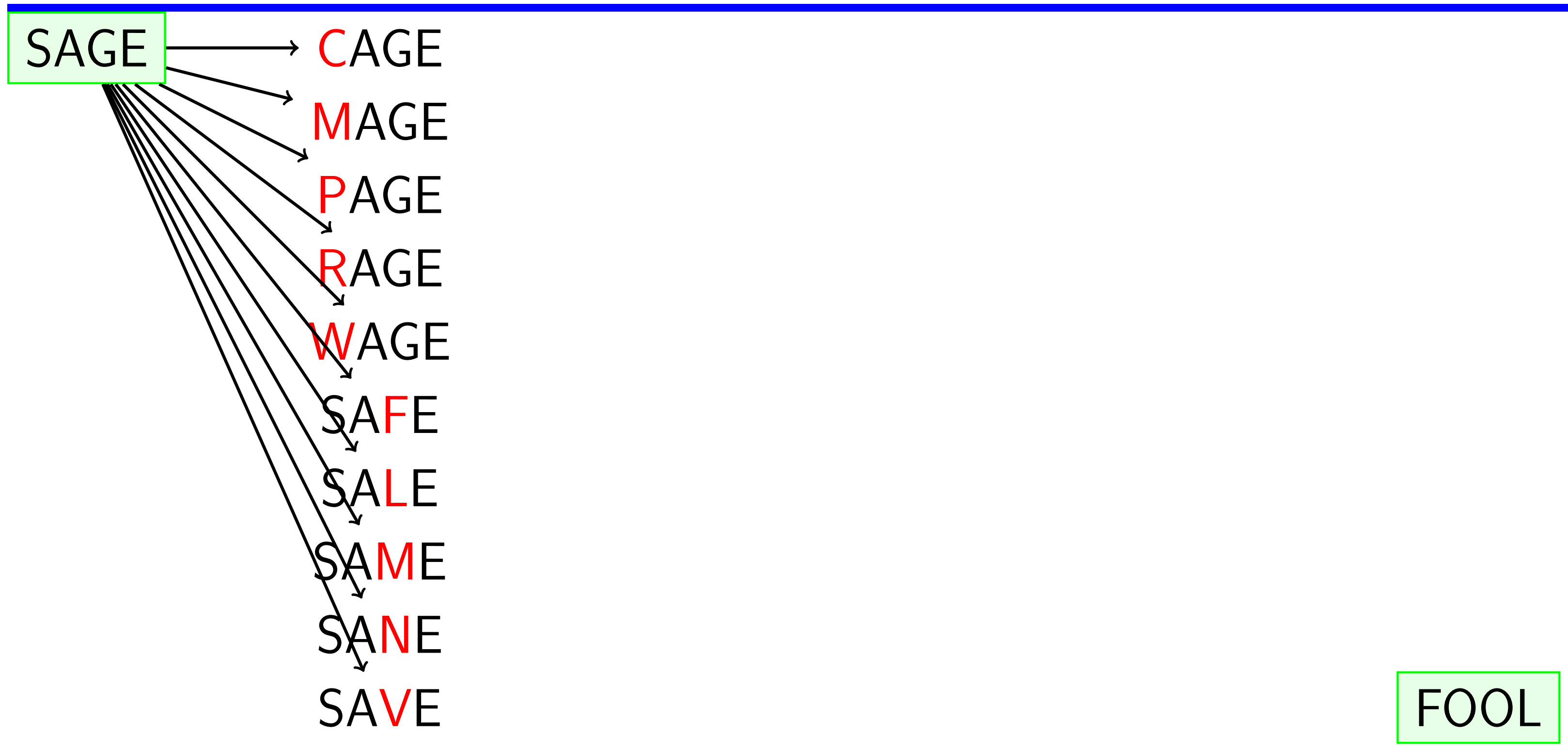
Changing the second letter of **SAGE** gives us no words.

Changing the third letter of **SAGE** gives us SAFE, SALE, SAME, SANE, SAVE.

Changing the fourth letter of **SAGE** gives us no words.

Now we know the "neighborhood" of **SAGE**, that is, all the legal English words that are just one letter-change away.

SAGE → FOOL



SAGE $\rightarrow$ FOOL

Given so many choices, let's focus on the very first one, and then move the others onto the back burner. If our first choice fizzles out, then we can backtrack, that is, come back to these unexplored choices and try them out.

The first word on our list, **CAGE** looks very useful, because there seem to be a lot of words we can get to next: CAFE, CAKE, CAME, CANE, CAPE, CARE, CASE, CAVE.

This suggests another heuristic or rule of thumb:

When exploring possibilities, open the door that leads to many more doors.

SAGE $\rightarrow$ FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] \arrow[ddddr] \arrow[dddddr] \arrow[ddddddr] \arrow[dddddddr] & \text{CAFE} & \\
 & & \text{CAKE} & \\
 & & \text{CAME} & \\
 & & \text{CANE} & \\
 & & \text{CAPE} & \\
 & & \text{CARE} & \\
 & & \text{CASE} & \\
 & & \text{CAVE} & \\
 & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

SAGE $\rightarrow$ FOOL

Now that we've stepped forward to **CAGE**, and we want to choose our next step to explore, we can again hope for a greedy step, that is, whether we can immediately swap in a letter of **FOOL**.

If we choose **CAFE**, then we can't take a greedy step from there.

But if we choose **CAKE**, then a greedy step is possible from there, getting closer to **FOOL** by transforming into **COKE** or **FAKE**.

Similarly, **CAME**, **CANE**, **CAPE**, **CARE** and **CAVE** all seem to offer a chance of stepping closer.

So now let's focus on the jump from **CAKE**, and put the other options also on the backburner.

SAGE → FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] \arrow[dddr] \arrow[ddddr] \arrow[dddddr] \arrow[dddddddr] \arrow[ddddddddr] & \text{CAKE} \arrow[r] \arrow[dr] & \text{COKE} & \\
 & & & \text{FAKE} & \\
 & & & \text{COME} & \\
 & & \text{CAME} \arrow[ur] \arrow[r] & \text{FAME} & \\
 & & \text{CANE} \arrow[r] & \text{CONE} & \\
 & & \text{CAPE} \arrow[r] & \text{COPE} & \\
 & & & \text{CORE} & \\
 & & \text{CARE} \arrow[ur] \arrow[r] & \text{FARE} & \\
 & & \text{CAVE} \arrow[r] & \text{COVE} & \\
 & & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

SAGE → FOOL

So starting from **SAGE**, and taking **CAGE** and then **CAKE**, we can see two greedy steps to choose from, to get us closer to **FOOL**.

Tentatively, we'll explore **COKE**, and put **FAKE** in reserve.

Now we imagine being at **COKE**, and look ahead to our next possible move. Another greedy step isn't possible, so let's just ask what other new words we can get.

It seems next steps are possible to at least JOKE, POKE, TOKE, WOKE, YOKE.

Notice now that, if we can't take a greedy step, our solution process is just blind. We're simply looking for any possible next step we can take.

SAGE → FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] & \text{CAKE} \arrow[r] \arrow[ddr] & \text{COKE} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] \arrow[ddddr] & \text{JOKE} \\
 & & & & \text{POKE} \\
 & & & \text{FAKE} & \text{TOKE} \\
 & & & & \text{WOKE} \\
 & & & & \text{YOKE} \\
 & & & & \\
 & & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

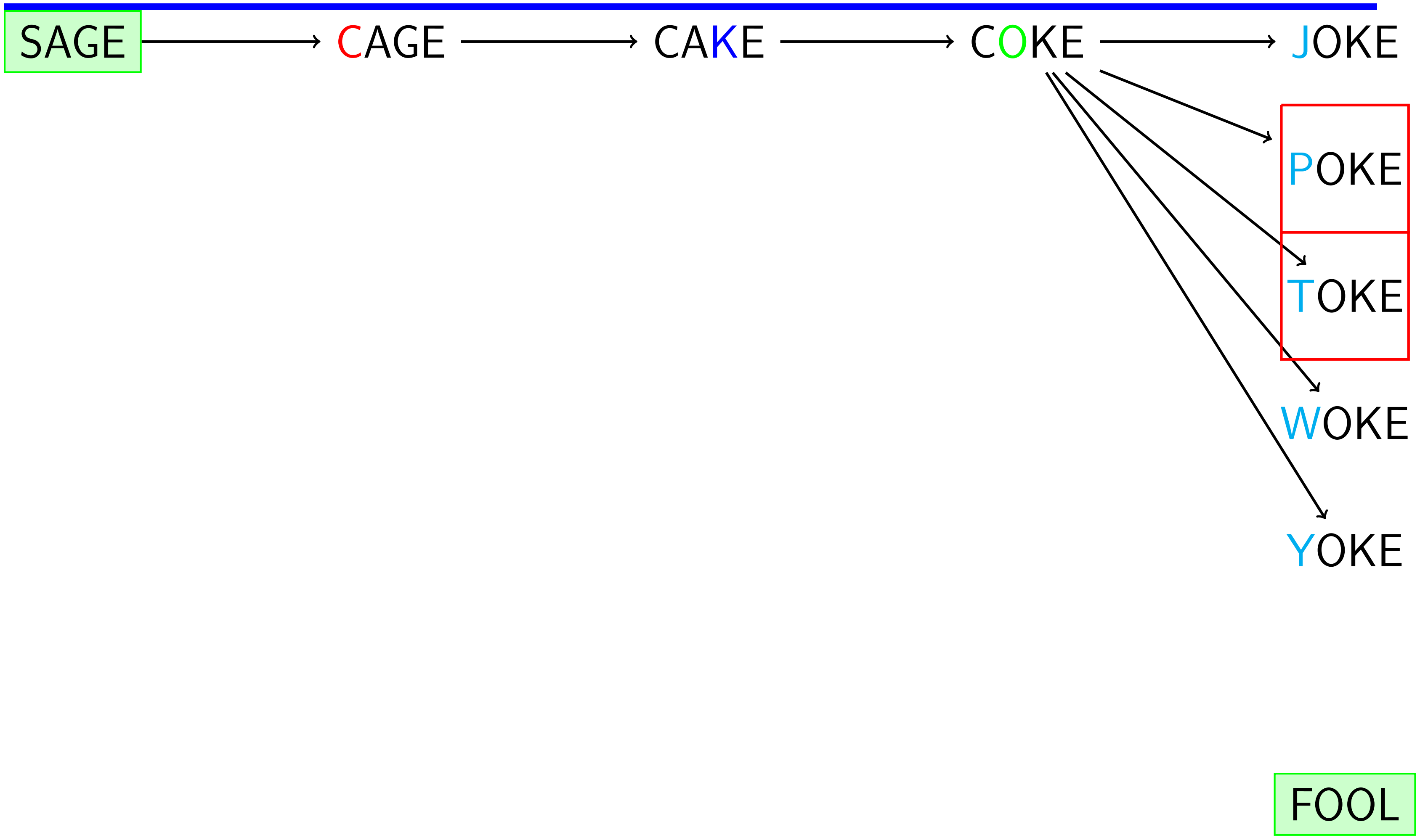
SAGE $\rightarrow$ FOOL

I really don't find **JOKE, WOKE, YOKE** attractive because words with the letters "J", "W" and "Y" don't seem very common. I'd much rather work with **POKE** or **TOKE**.

Let's make POKE our focus, with **TOKE** as our backup, and **JOKE, WOKE, YOKE** as backup backups...

Here's a new example of a heuristic, or rule of thumb. We are guessing that, if we have a choice of which letter to swap into our current word, it's much better to choose a common letter, since that will likely mean that on the next step, our new word will lead to many other choices.

SAGE → FOOL



SAGE $\rightarrow$ FOOL

What letter of **POKE** can we change for our next step?

We got here by changing the first letter, so let's leave that alone.

If we change the second letter, we can get PIKE, PILE, PINE, PIPE. But we'd like to keep the second letter, since that matches the **O** in **FOOL**.

(Heuristic: try not to throw away letters that already match your goal!)

Changing the third letter can get us POLE, POPE, PORE, POSE

Changing the fourth letter seems impossible.

SAGE → FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] & \text{CAKE} \arrow[r] & \text{COKE} \arrow[r] \arrow[dr] & \text{POKE} \\
\text{POKE} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] \arrow[ddddr] \arrow[dddddr] \arrow[ddddddr] \arrow[dddddddr] & \text{PIKE} & & & \text{TOKE} \\
& \text{PILE} & & & \\
& \text{PINE} & & & \\
& \text{PIPE} & & & \\
& \text{POLE} & & & \\
& \text{POPE} & & & \\
& \text{PORE} & & & \\
& \text{POSE} & & & \\
& & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

SAGE $\to$ FOOL

And now things start to get exciting!

If we choose **POLE**, then we can see that the greedy step can work next, giving us POLL.

If instead we choose **PORE**, then we can also take a greedy step next, getting FORE.

Thus we have another heuristic:

When deciding which step to choose, look ahead to see if that step can be followed by a greedy step.

In either case, we seem to be moving close to our solution by getting two matching letters!

SAGE → FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] & \text{CAKE} \arrow[r] & \text{COKE} \arrow[r] \arrow[dr] & \text{POKE} \\
\text{POKE} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & \boxed{\text{POLE}} \arrow[r] & \text{POLL} & & \text{TOKE} \\
& \text{POPE} & & & \\
& \boxed{\text{PORE}} \arrow[r] & \text{FORE} & & \\
& \text{POSE} & & & \\
& & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

SAGE → FOOL

I am really interested in choosing **POLE** followed by **POLL**, because it means that we have swapped out a vowel for a consonant in the fourth position, which is a difficult jump.

Remember, that was one of our heuristic rules:

Always choose a change if it replaces a mismatched vowel by a matching consonant (or the other way around.

SAGE → FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] & \text{CAKE} \arrow[r] & \text{COKE} \arrow[r] & \text{POKE} \\
\text{POKE} \arrow[r] \arrow[ddr] & \text{POLE} \arrow[r] & \text{POLL} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & \text{POOL} & \\
 & & & \text{DOLL} & \\
 & \text{PORE} \arrow[r] & \text{FORE} & \text{PILL} & \\
 & & & \text{POLO} & \\
 & & & \textit{many more!} & \\
 & & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

SAGE $\rightarrow$ FOOL

And once we have $\mathbf{POLL}$, things become very clear.

A greedy step is possible, taking us to POOL.

And immediately, a final greedy step takes us to $\mathbf{FOOL}$ and we're done!

And this reminds us of another heuristic:

The problem gets easier as you get close to the solution.

SAGE $\rightarrow$ FOOL

$$\begin{tikzcd}
\boxed{\text{SAGE}} \arrow[r] & \text{CAGE} \arrow[r] & \text{CAKE} \arrow[r] & \text{COKE} \arrow[r] & \text{POKE} \\
\text{POKE} \arrow[r] & \text{POLE} \arrow[r] & \text{POLL} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & \text{POOL} \arrow[dddddr] & \\
 & & & \text{DOLL} & \\
 & & & \text{PILL} & \\
 & & & \text{POLO} & \\
 & & & \textit{many more!} & \\
 & & & & \boxed{\text{FOOL}}
\end{tikzcd}$$

Our solution process summarized

Now we can display our solution, hiding all the work we did, and all the partial results we kept in backup in case our first guesses didn't work.

SAGE turned into **FOOL** using 7 intermediate words.

I think it is fair to say that, until near the end, we really had no idea whether or not we would reach the solution.

And that's because there were many times when we could not take a greedy step, and we simply had to blindly come up with as many words as possible that were connected to our current word.

The only thing that kept us from being completely random was a few heuristic rules we came up with.

Our solution process cleaned up

$$\begin{tikzcd}
\text{SAGE} \arrow[dr] & & & & & & & & \\
& \text{CAGE} \arrow[dr] & & & & & & & \\
& & \text{CAKE} \arrow[dr] & & & & & & \\
& & & \text{COKE} \arrow[dr] & & & & & \\
& & & & \text{POKE} \arrow[dr] & & & & \\
& & & & & \text{POLE} \arrow[dr] & & & \\
& & & & & & \text{POLL} \arrow[dr] & & \\
& & & & & & & \text{POOL} \arrow[dr] & \\
& & & & & & & & \text{FOOL}
\end{tikzcd}$$

We can measure distance

We can measure how close we are getting to the solution simply by counting the number of incorrect letters. This is a sort of distance measurement.

SAGE starts with four letters incorrect; our next two moves don't add a correct letter, they are just searching around for a good jump.

When we go from **CAKE** to **COKE**, though, our distance does drop to 3, since "O" is the right letter in the right place.

Just as in real life, being able to tell how far away you are is a huge help in solving a problem.

For this example, the distance always went down. We can imagine puzzles for which the distance might go up, where we have to temporarily lose a correct letter to reach a useful steppingstone word.

How the distance went down

$$\begin{tikzcd}
\text{SAGE : 4} \arrow[dr] & & & & & & & & \\
& \text{CAGE : 4} \arrow[dr] & & & & & & & \\
& & \text{CAKE : 4} \arrow[dr] & & & & & & \\
& & & \text{COKE : 3} \arrow[dr] & & & & & \\
& & & & \text{POKE : 3} \arrow[dr] & & & & \\
& & & & & \text{POLE : 3} \arrow[dr] & & & \\
& & & & & & \text{POLL : 2} \arrow[dr] & & \\
& & & & & & & \text{POOL : 1} \arrow[dr] & \\
& & & & & & & & \text{FOOL : 0}
\end{tikzcd}$$

Heuristics / Rules of Thumb we discovered

Closeness: Once you get close, the puzzle gets much easier.

Distance: Have a measurement of how close you are.

Backtrack: Remember unused choices in case you hit a dead end.

Greedy step: Can we swap in another letter of the target word?

Greedy step next: If we take this step, can we take a greedy step next?

Vowel/Consonants: Take steps matching vowel/consonant pattern of target.

Unusual letters: Avoid steps adding unusual letters: J, K, Q, W, X, Y or Z.

Look ahead: Prefer a step that leads to many next steps.

Forward!: Try not to sacrifice a target letter once you've gotten it.

No waste!: Don't change the letter you just changed on the previous step!

Socrative Quiz PartIV_Quiz1

IUZGAZ34E

1. If **cold** can be converted to **warm** using doublets, what is the smallest number of intermediate words necessary? (1, 2, 3, 4)
2. In the doublet (**head, heal, ????, tell, tall, tail**), what is the missing word? (hall, tall, teal, heat)
3. If there is a doublet that transforms **hit** to **cog**, must there be a doublet that transforms **cog** to **hit**? (Yes, No)
4. If the start word includes only one vowel, but the final word includes only one consonant, then is it impossible to connect them with a doublet? (Yes, No)
5. If we can transform **pale** to **fool** using 3 intermediate words, and we can convert **cope** into **pale** using 3 intermediate words. Knowing only this information, how many intermediate words could we use to transform **pale** into **fool**? (Not possible, 3, 5, 7, 9)

Enlighted Stumbling?

If we had to describe our solution procedure, we might call it enlightened stumbling, nothing to brag about!

We know where to start, and where we would like to end.

But in between, we only have some general suggestions for how to choose our next step.

It seems like we haven't completely understood how a human solves such puzzles.

It's a little like Captain Kirk from Star Trek, jumping into a new planet, knowing a goal without really being sure what obstacles will arise on the way.

Our solution method is not entirely satisfactory!

Doublets are just fun word problems, but if we seriously wanted a computer to solve a doublet puzzle, then our enlightened stumbling method is not satisfactory.

Even if the puzzle has a simple solution, the computer could go off in the wrong direction (so could a human). But the human would get tired after a while and look for a simpler solution. The computer might just keep on churning.

Along the way, the computer might get into a cycle, taking the steps

CARE => CAKE => COKE => CORE => CARE => CAKE => ...

Again, a human would spot this, but maybe not the computer.

We like to think that an algorithm is a calculating recipe that has a definite termination. Our stumbling approach doesn't!

Think about the problem as a trip, and look for a map

In doublets, we are trying to "travel" from one word to another, but we don't have a map.

A map would help us plan the route; it would tell us if there was a route from one word to another; and even how to take the shortest one.

Making such a map would require a lot of preliminary work.

Road maps may include in small print the distance between two cities that are directly connected by a stretch of road; but we will want to know the distance of the total journey.

So we are looking for a map that answers the questions:

Is there a path from the start word to the end word?
If so, what is the shortest path?

Shortest Path

It looks like, in order to efficiently use a computer to solve the doublets problem, we need to know how to set up and solve something called the shortest path problem.

The shortest path problem is a famous case in computing. Versions of this problem arise during many kinds of computation, and our doublets problem is one of them.

Let's consider two simple examples of the shortest path problem:

- a city-to-city driving map;
- a maze of connected rooms.

Solving these problems will suggest how to make a systematic solution of any doublets puzzle.

City-to-City Driving Map

For the driving map problem, suppose we have cities $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$ and $\mathbf{F}$, with a network of roads of varying lengths. We live in city B and want to know the shortest distance to all the other cities on the map.

We know that the shortest distance from B to itself is 0 miles, so we can fill that in before we start. We're not sure of the other distances yet, so we can temporarily set them to ∞, the mathematical symbol for infinity.

To fill in better estimates for the unknown distances, we look at all the cities that are directly connected to B, and pick the closest one. Let's say this is city A. It should be clear that there's shorter route to A. We can now add A to the $\mathbf{sure}$ set, that is, the set of cities whose distance from B is known for sure.

We are sure that there is no way to shorten the distance from B to A by going through another town, say C, because just getting to C takes longer than getting to A directly.

Update distances, then add the closest one

Now that we know the distance from B to A, we need to check whether we can lower the distances to some of the unsure cites.

We check the distances of trips that start at B, pass through A, and stop at an immediate neighbor of A. Any time such a trip is shorter than what we've already recorded, we put down the new shorter estimate.

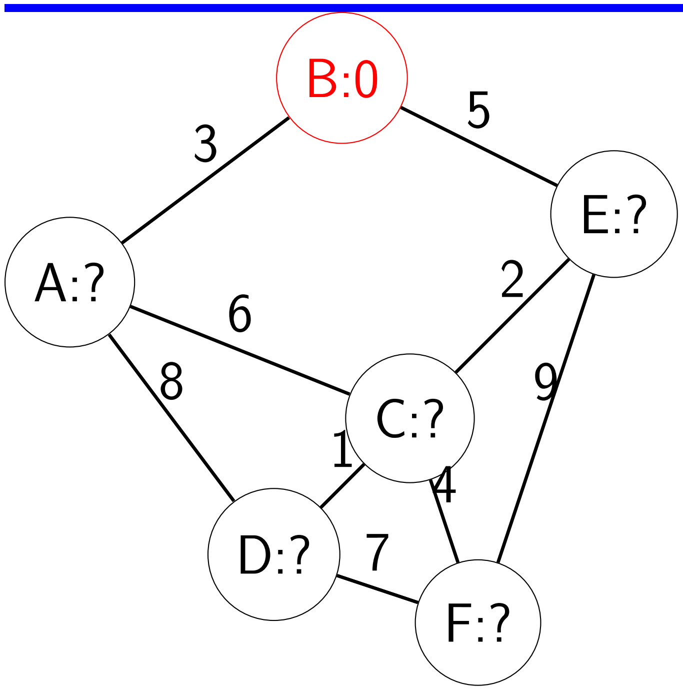
After this check, we look at the table for the city, say "F", with the lowest distance in the unsure set, and move it to the sure set.

Then we consider the distance of trips that start at B, pass through F, and continue to any one of F's immediate neighbors.

Our completed table lists the shortest distance from B to any city.

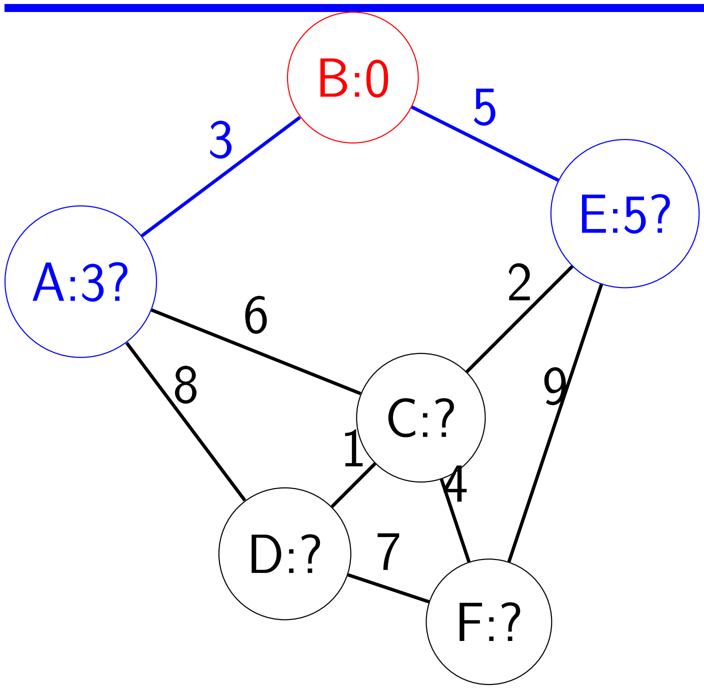
To do a table of shortest distances from any city to any city, we have to repeat the whole procedure, picking a new starting city each time.

Distance from B to itself is 0



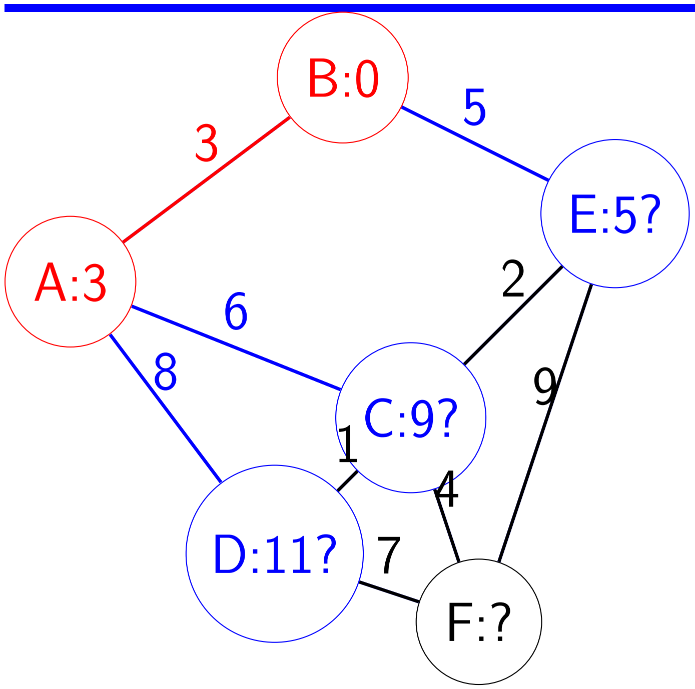
City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞

Immediate neighbors are no more than 3 and 5 away



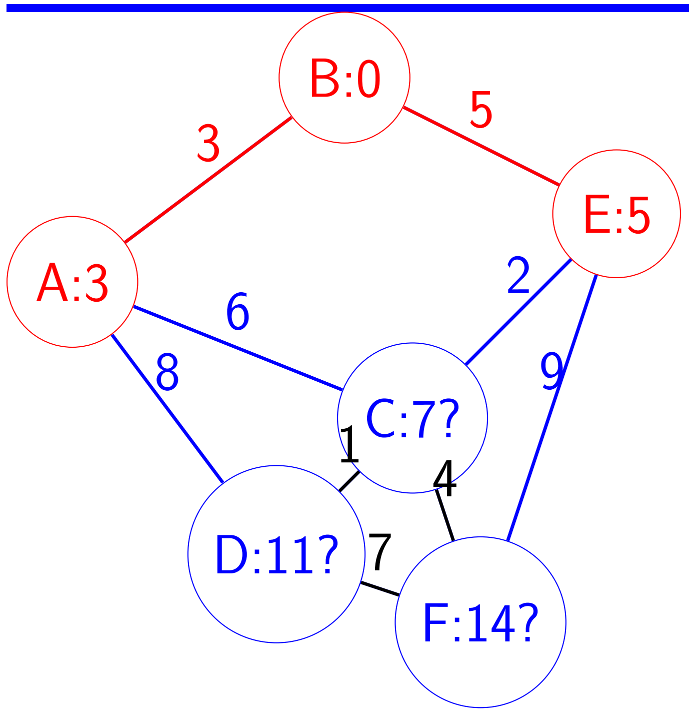
City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞

A is surely 3 away; C, D and E are updated



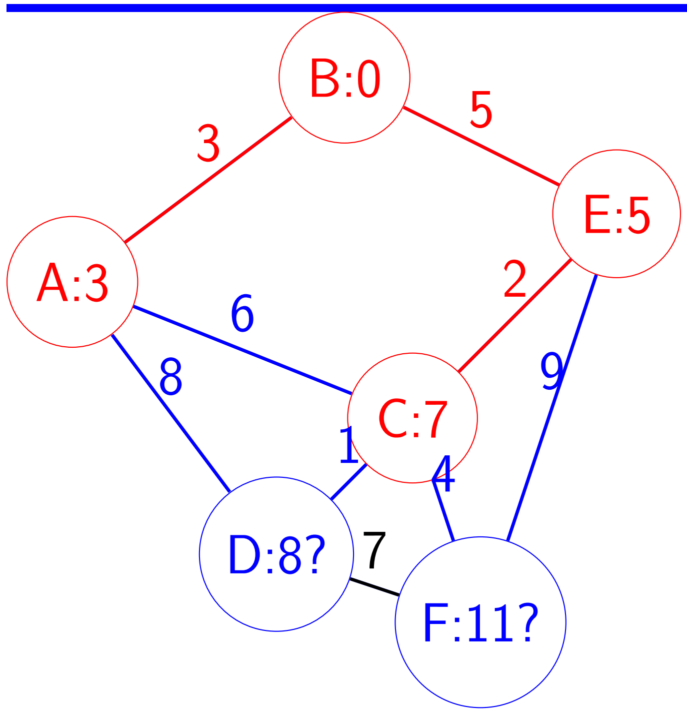
City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞
From A	3	0	9	11	5	∞

E is surely 5 away; C, D and F are updated



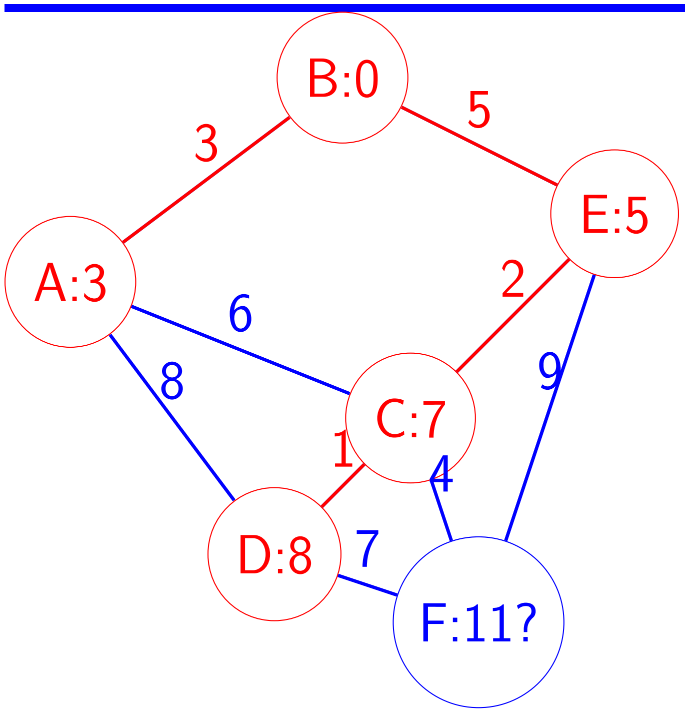
City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞
From A	3	0	9	11	5	∞
From E	3	0	7	11	5	14

C is surely 7 away; D and F are updated



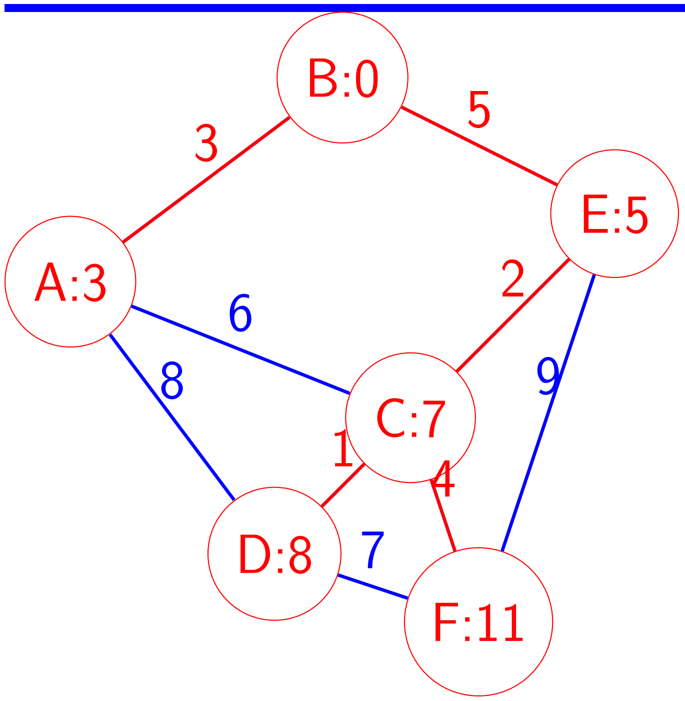
City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞
From A	3	0	9	11	5	∞
From E	3	0	7	11	5	14
From C	3	0	7	8	5	11

D is surely 8 away; F is updated



City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞
From A	3	0	9	11	5	∞
From E	3	0	7	11	5	14
From C	3	0	7	8	5	14
From D	3	0	7	8	5	11

F is surely 11 away; all distances computed



City	A	B	C	D	E	F
B	∞	0	∞	∞	∞	∞
From B	3	0	∞	∞	5	∞
From A	3	0	9	11	5	∞
From E	3	0	7	11	5	14
From C	3	0	7	8	5	14
From D	3	0	7	8	5	11
Done!	3	0	7	8	5	11

We can make a distance map from any city to any city

After all that work, we only know the shortest distances for trips that start at city B. To make a complete driving distance table, we need to repeat this process for each possible starting city.

Here's the result for our sample map, with our previous city B results highlighted in red:

	To A	To B	To C	To D	To E	To F
From A	0	3	6	7	8	10
From B	3	0	7	8	5	11
From C	6	7	0	1	2	4
From D	7	8	1	0	3	5
From E	8	5	2	3	0	6
From F	10	11	4	5	6	0

The distance table has some interesting properties

This distance table has some properties that correspond to our ideas of distance in the real world:

- The distance is never negative;
- The distance from a city to itself is always 0;
- The distance from A to B is the same as from B to A;
- The distance from A to B plus the distance from B to C can never be less than the distance from A to C.

Would these properties be true if some routes were one-way only?

Do airline flying times from city to city obey these patterns?

In a maze, all the connections are 1 unit in length

In the city problem, the length of the road between two cities is part of the problem.

In a maze, two rooms are simply connected or not.

A maze problem concentrates on taking the fewest steps

Doublets is somewhat like our city distance problem, because we do have a beginning word, an end word that we are trying to reach, and connections from one word to another that we could also think of as roads.

Having a map, and knowing the shortest distance between any pair of words, would be very helpful.

Both mazes and doublets are simpler than the city distance problem, however, because the connections we use don't have different lengths. We count the steps we take in transforming words, so each word we "visit" involves a trip of 1 unit in length.

Since we don't have to worry about the length of the connections, we can think of our doublets problem as more closely related to solving a maze.

A maze problem is like a doublets problem

Suppose we are placed in a "start" room of a maze and told to wander around seeking the "goal" room.

We could seek our goal by aimless wandering, of course. (That's what we did in our doublets solution.)

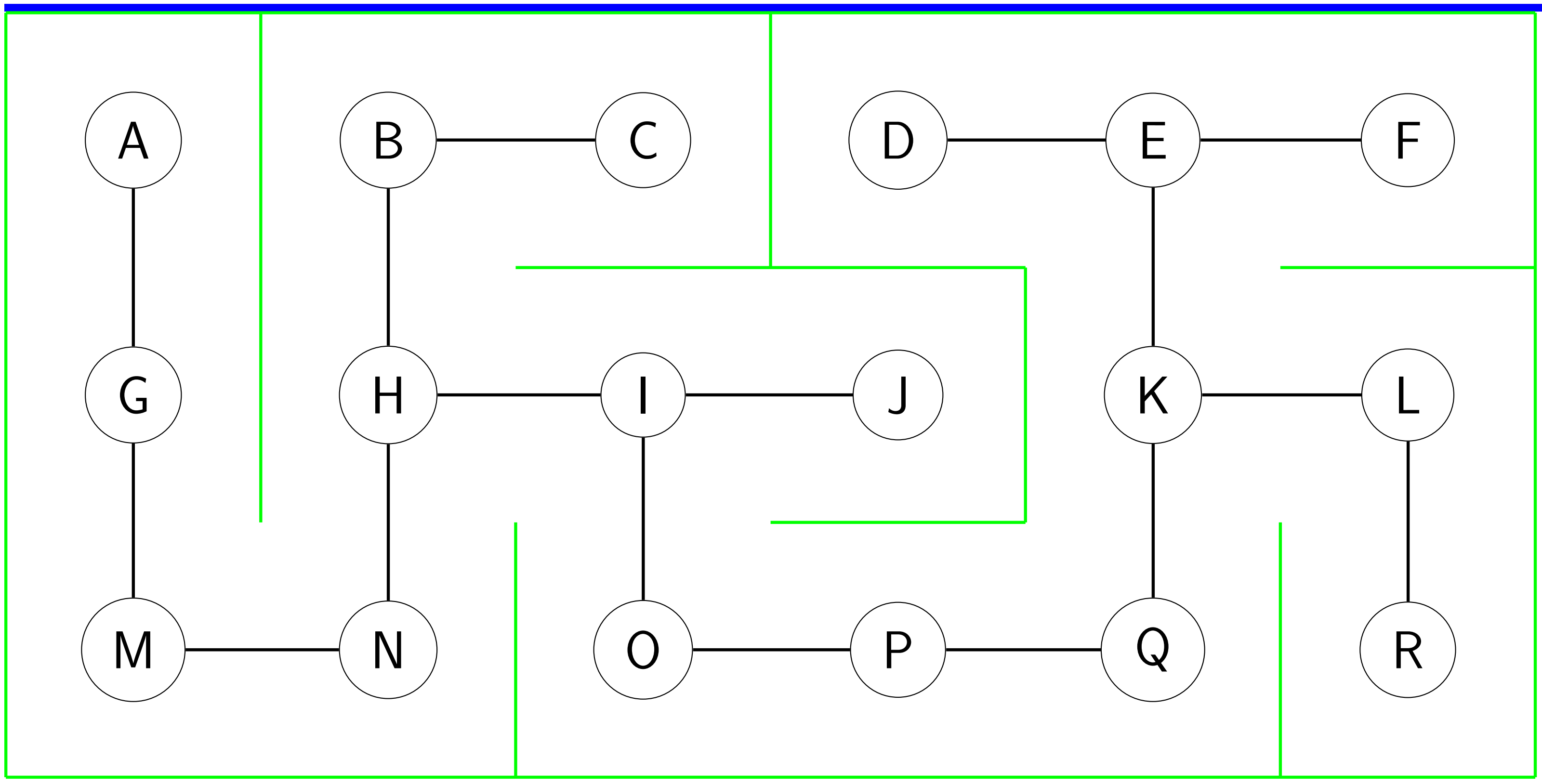
But if we have a map of the the maze, we could instead imagine a systematic approach, which involves measuring the distance (number of steps) from our starting room to every other room.

We know the starting room has distance 0, of course. Now reach into each room immediately connected to the starting room and paint a "1" on the floor.

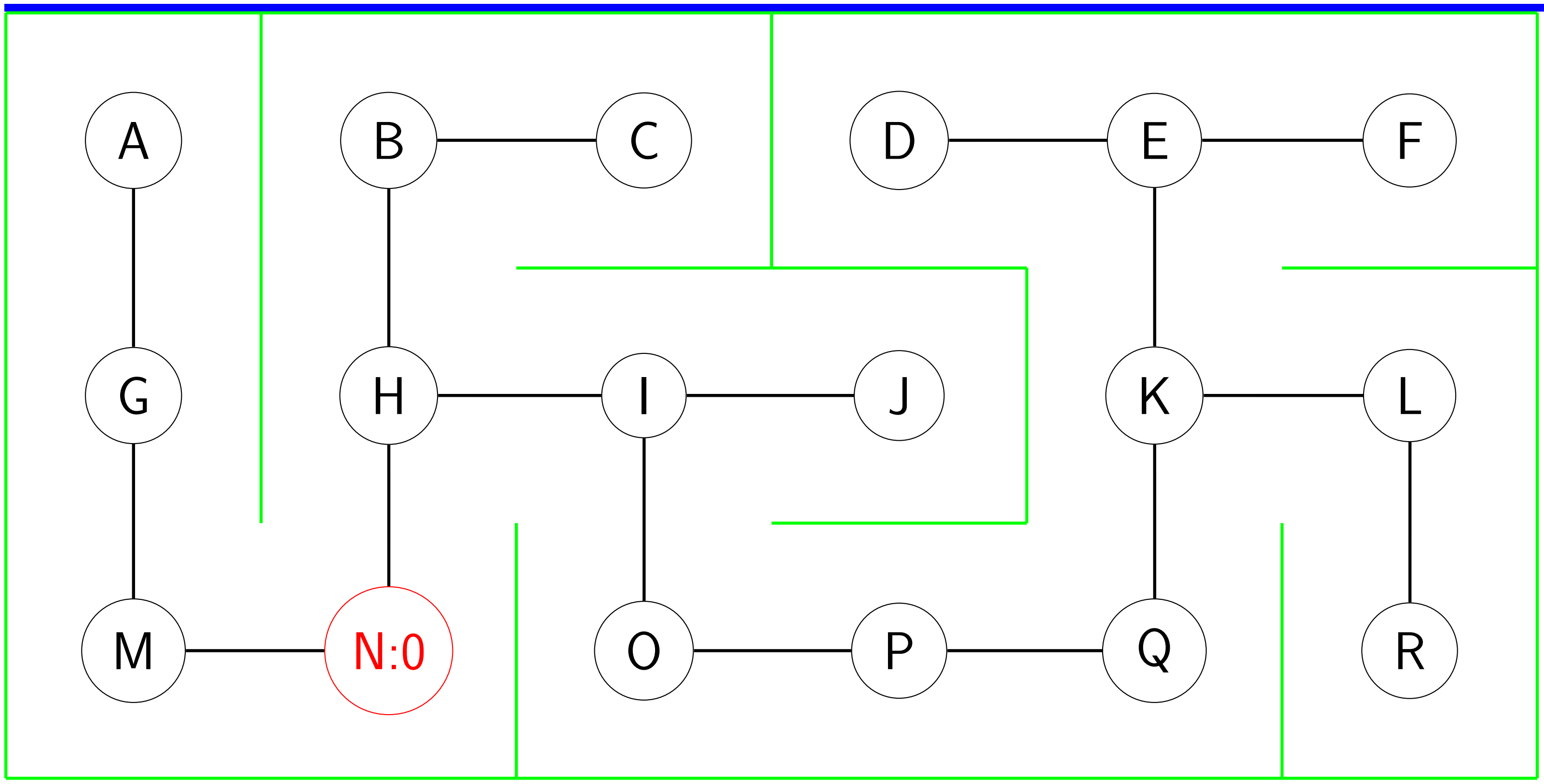
From every "1" room, reach into every unpainted neighbor rooms and mark them "2".

Repeating this process gets you to the goal room, tells you how far the goal room is from the start, and even gives you a trail to follow back to the starting room.

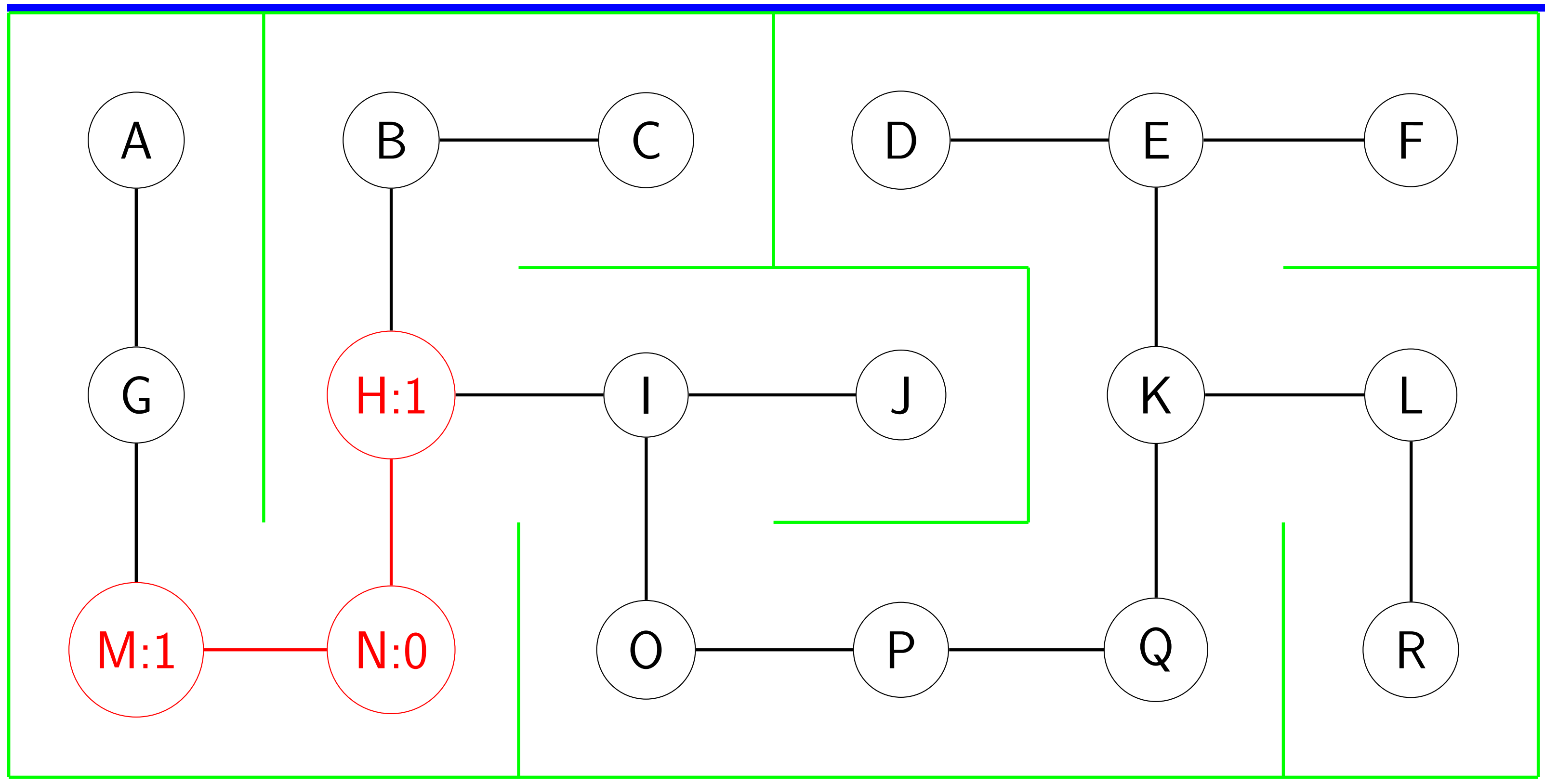
Here’s a simple maze



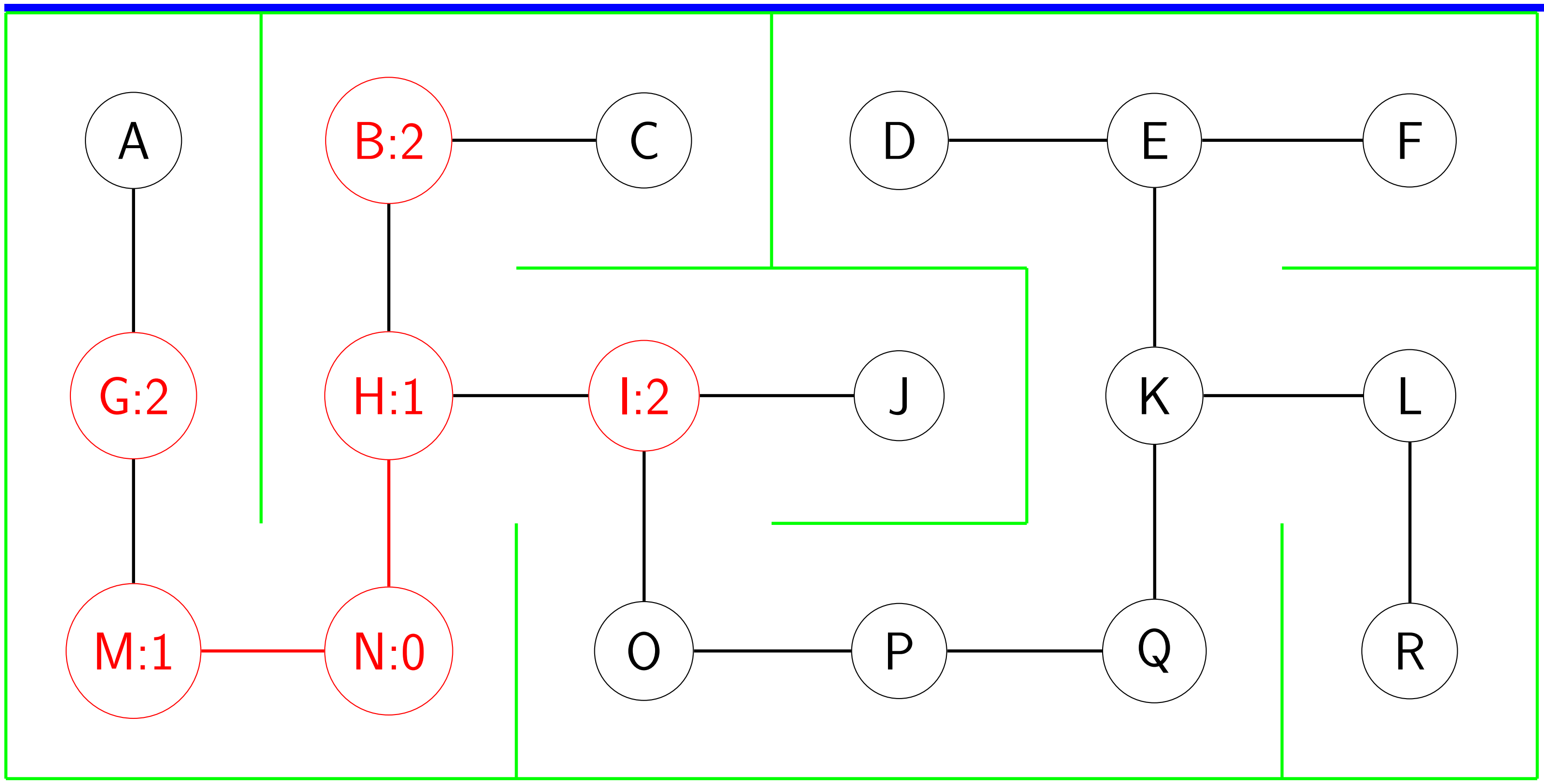
We start in room N, so mark it 0



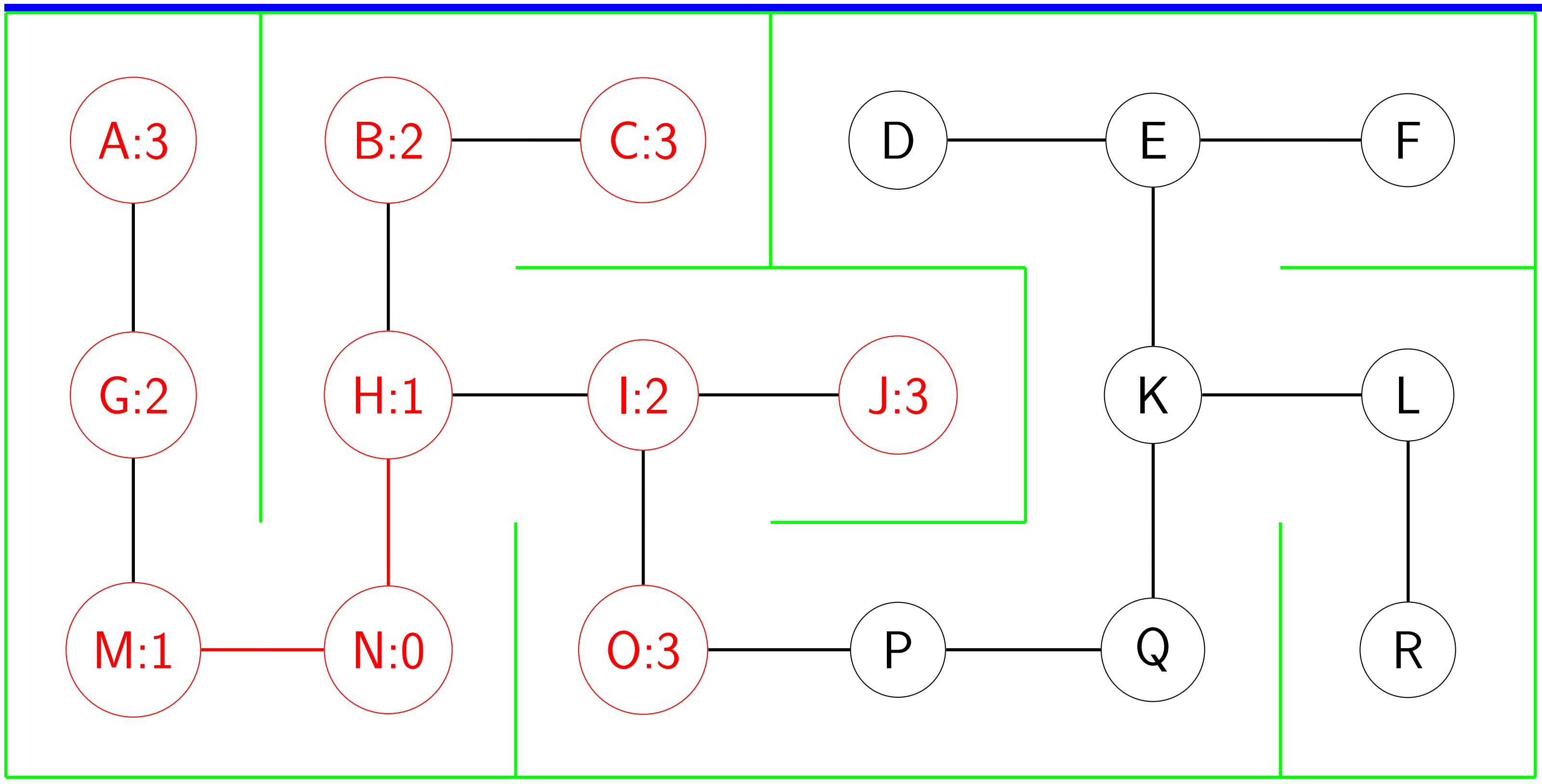
Mark the neighbors H and M with a “1”



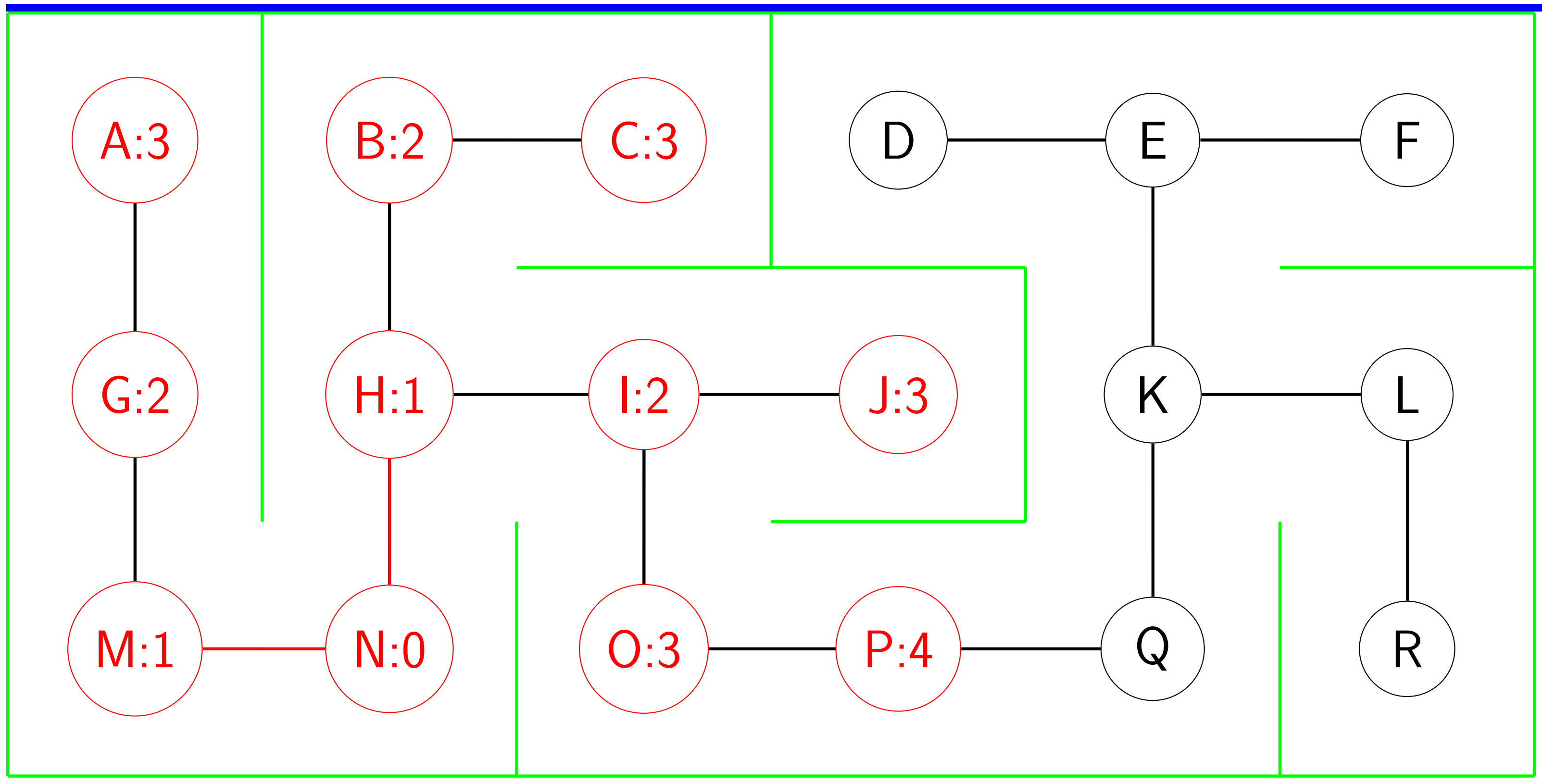
Mark the neighbors B, G, I with a “2”



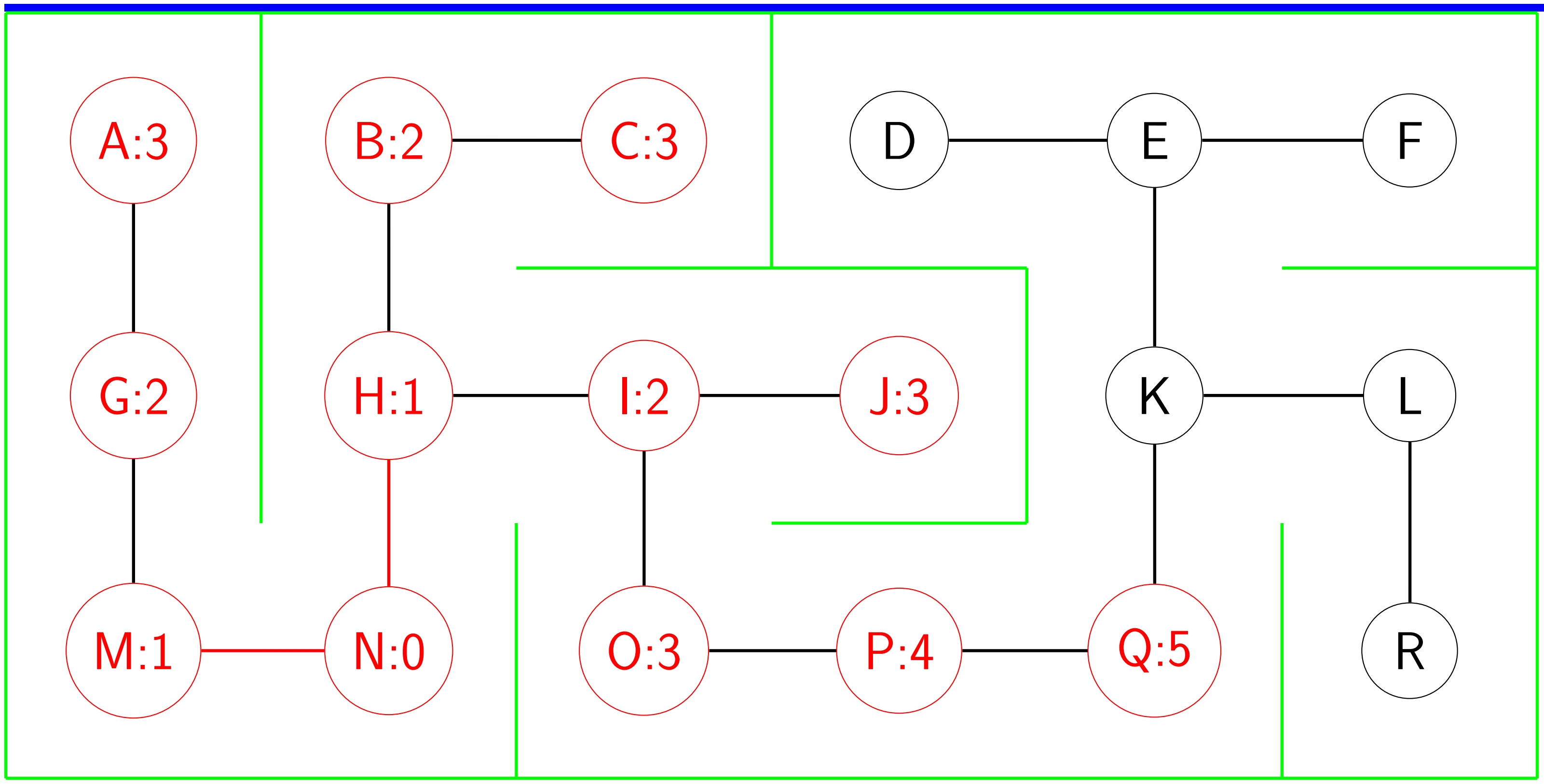
Mark the neighbors A, C, J, O with a “3”



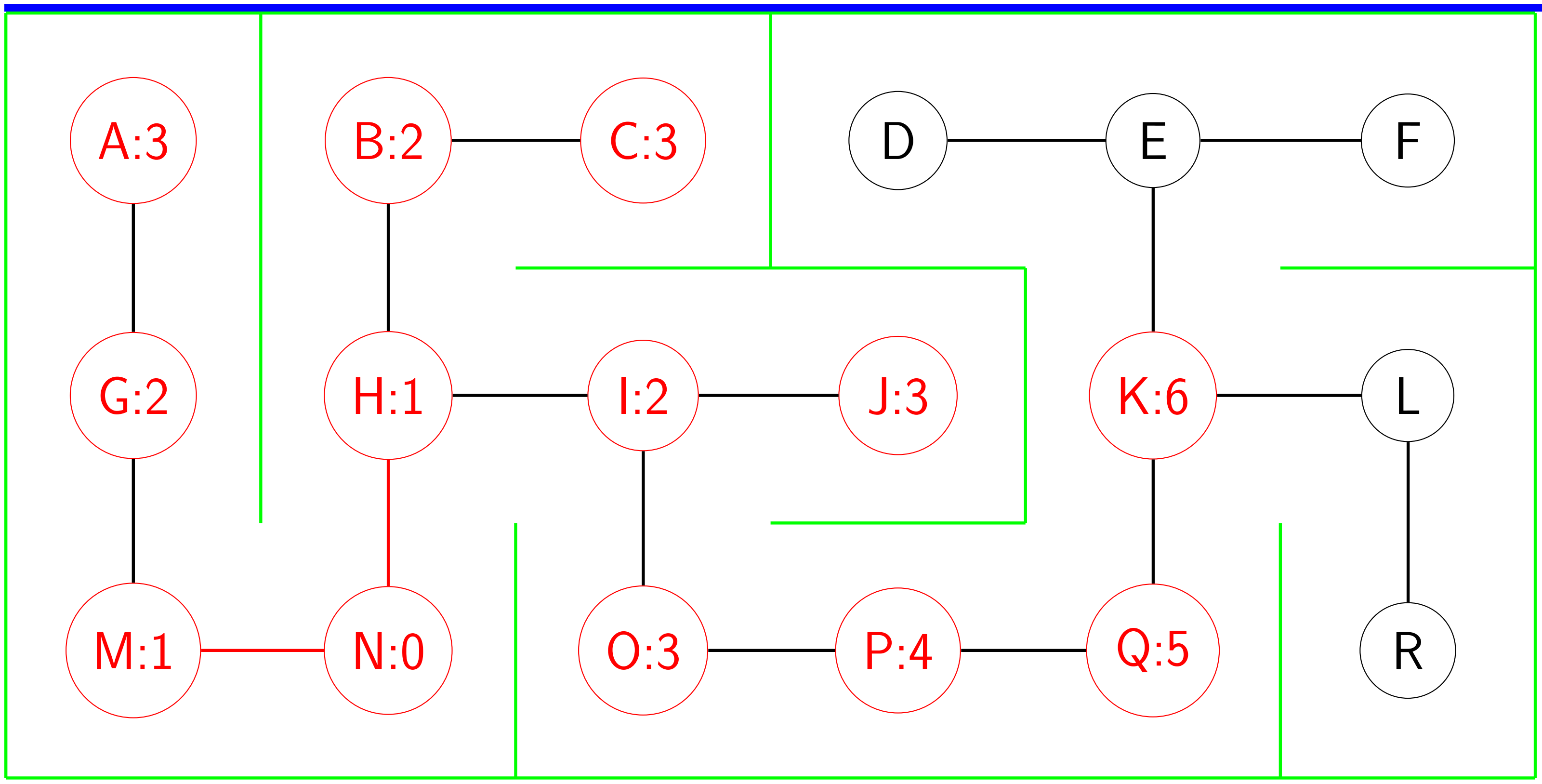
Mark P with a “4”



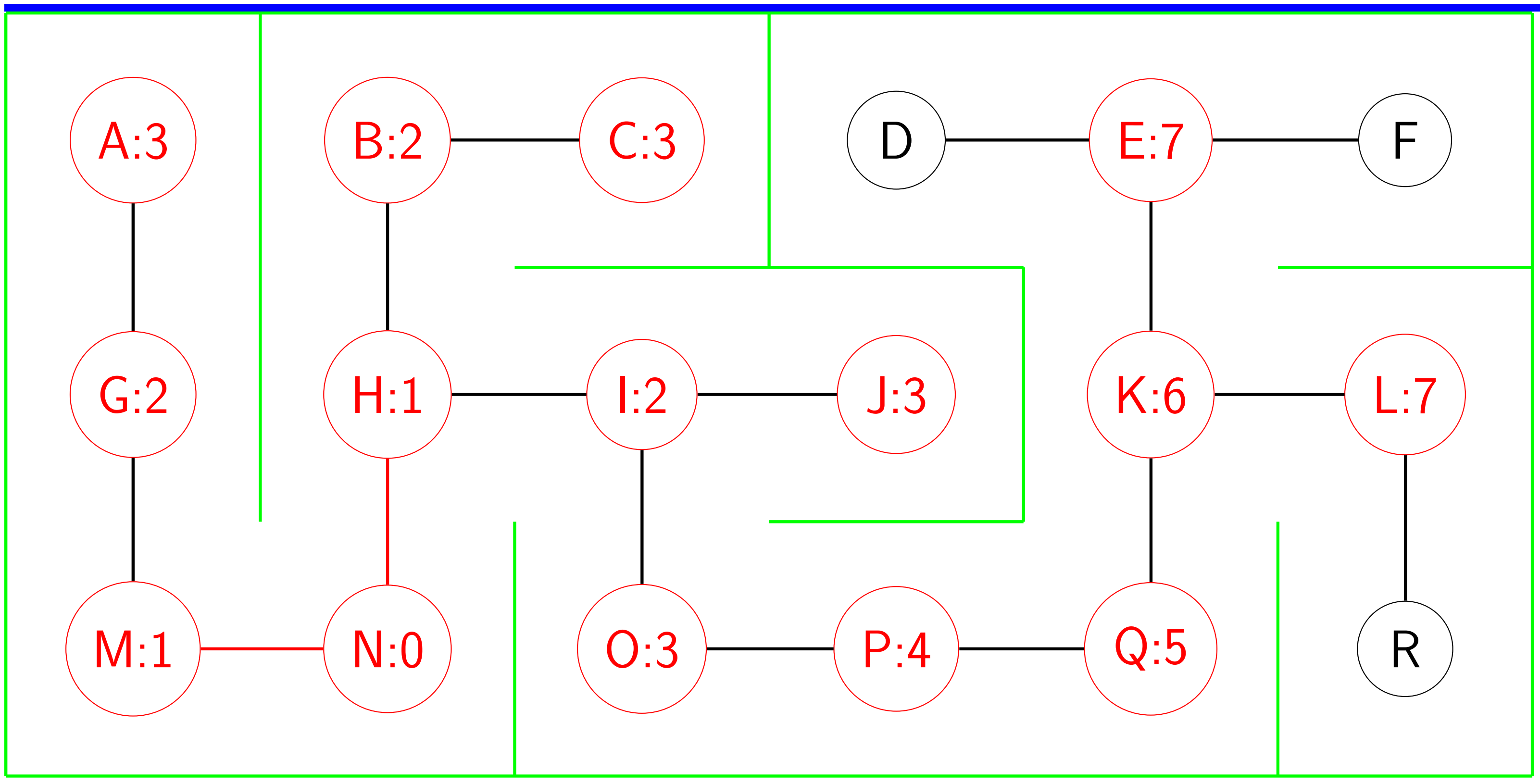
Mark Q with a “5”



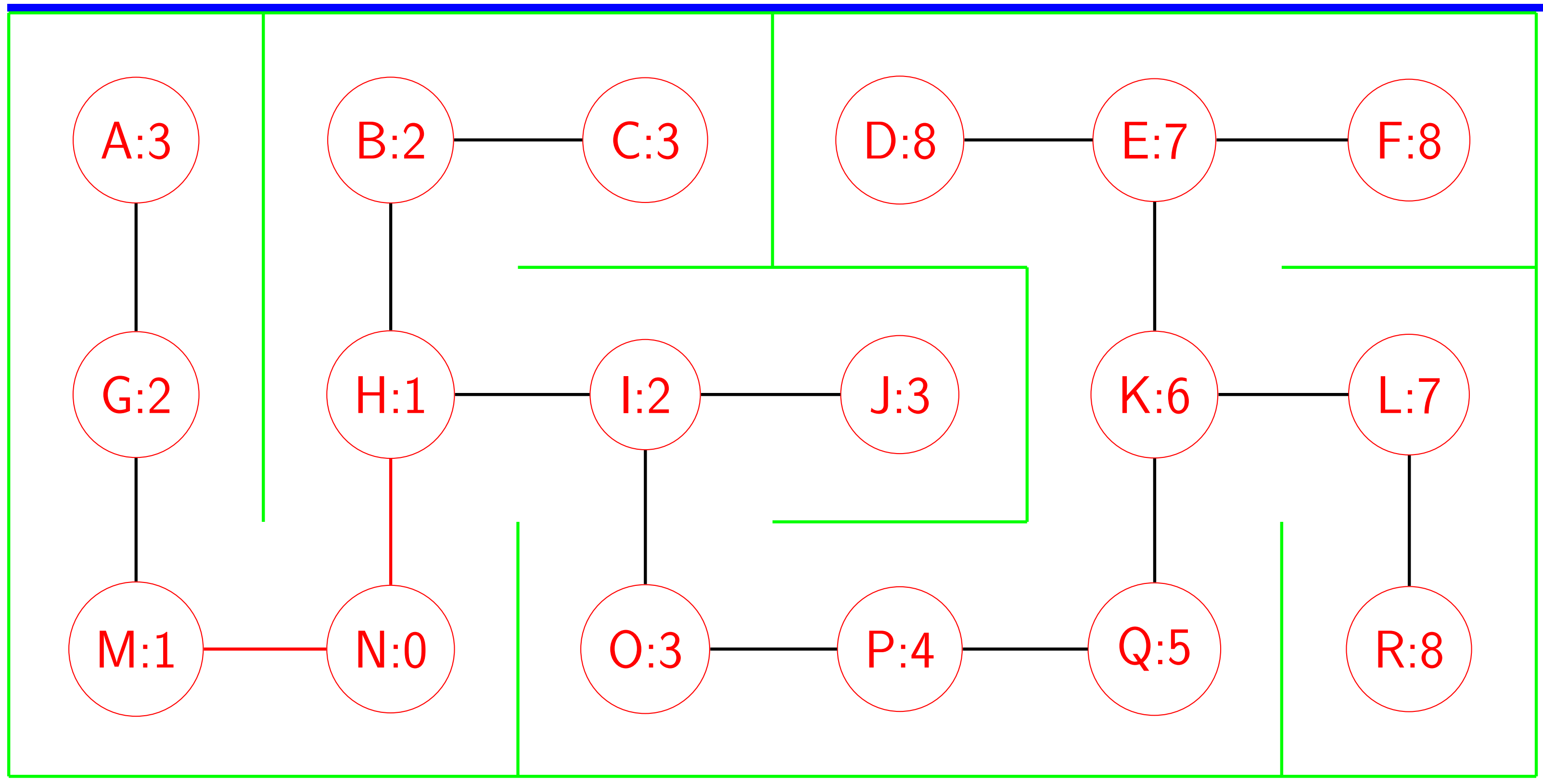
Mark K with a “6”



Mark L and E with a “7”



Mark D, F and R with an “8”



Our maze map solves the puzzle

Now our diagram of the maze has turned into a shortest distance table for trips that start at position $\mathbf{N}$.

This means, for example, that the shortest distance from $\mathbf{N}$ to $\mathbf{K}$ requires 6 steps, and you get there by starting at $\mathbf{K}$ and moving in the direction of decreasing distance.

So the shortest distance problem is simpler to work on when the connections or road all have length 1.

We simply pick our starting point, and then all the immediate neighbors are guaranteed to be one unit away.

All their neighbors (if we haven't already seen them) are 2 units away, and so on.

By marking each spot with its distance, we get a table of distances, and we can even work out the path back to our starting point.

Finding the path

Let's make sure we understand that our map tells us how to find the shortest path from our START point (room $\mathbf{N}$) to any END point.

Suppose our END point is room $\mathbf{K}$. To find the path, we actually have to work backwards, that is, we start by looking at room $\mathbf{K}$, and notice that its distance to room $\mathbf{N}$ is given as 6.

That means that at least one neighbor of room $\mathbf{K}$ must have a distance of 5. Looking at neighbors $\mathbf{E}$, $\mathbf{L}$ and $\mathbf{Q}$, we see that $\mathbf{Q}$ is the right choice.

Now we move to room $\mathbf{Q}$ (distance 5), and look for any neighbor that is a distance of 4 away from room $\mathbf{N}$, and we see that room $\mathbf{P}$ will do it.

Proceeding in this way, in 6 steps, we have found our way back to $\mathbf{N}$

You should see that we have to work backward to find the path, and that if we instead started at room $\mathbf{N}$, we wouldn't see any information that would allow us to choose the correct path to $\mathbf{Q}$.

Solving the maze problem gives us ideas for doublets

Now that we've thought about maps and shortest distances, let's return to our doublets problem and use these ideas.

The words in doublets are like the rooms in the maze.

Two words are connected if they differ by a single letter.

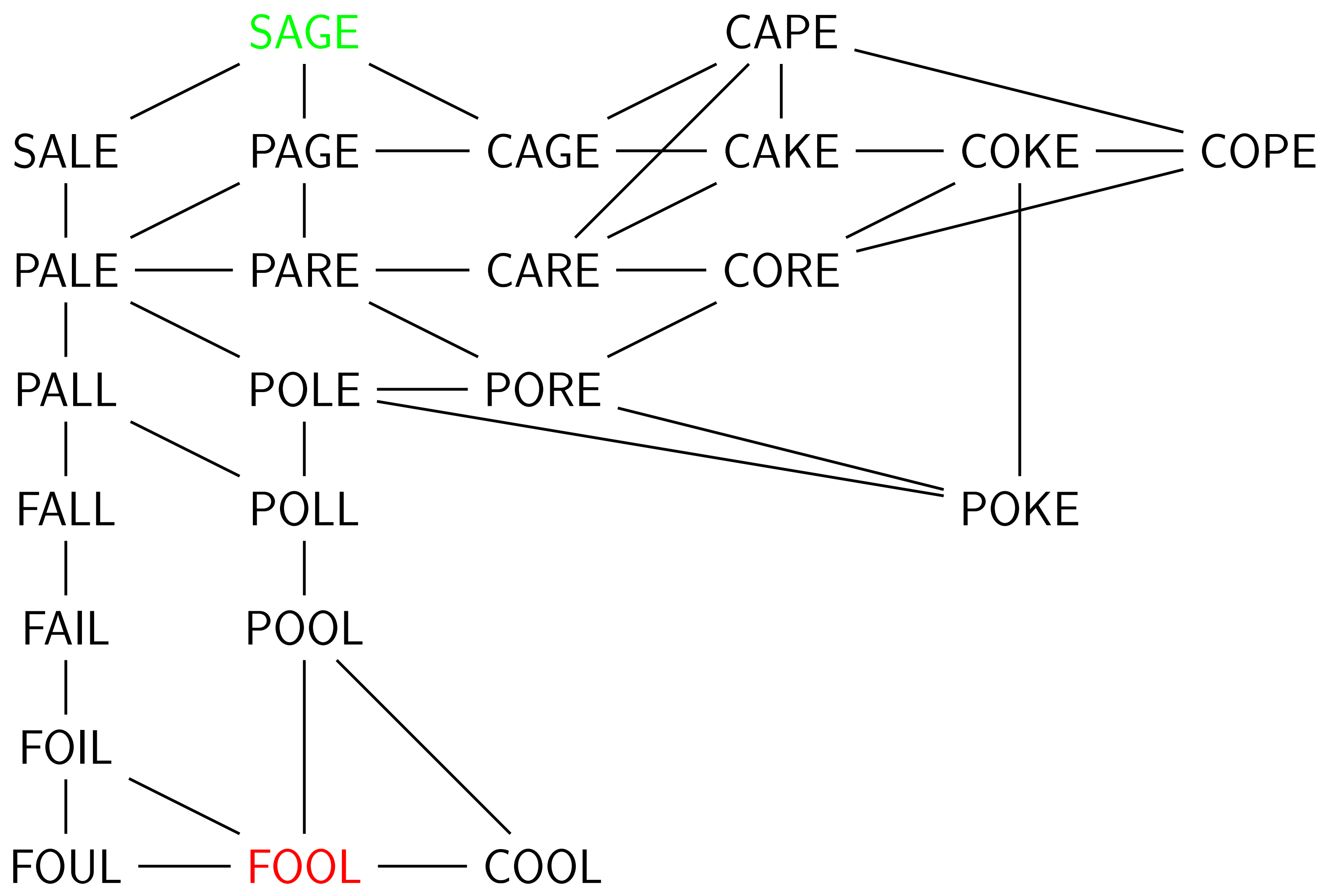
We begin at the START word (one room) and want to reach the END word (another room).

We can think about words as though they were rooms in a maze. There is a door between two words if they differ by just one letter.

If we have a map of the maze, we can see if a solution exists, and even what the shortest one is.

We can't afford to draw a map of all possible four-letter words, so let's draw a reduced map with a limited vocabulary.

A very simplified map of SAGE $\rightarrow$ FOOL



A very simplified map of SAGE $\rightarrow$ FOOL

Here is a sort of map of our word problem for transforming SAGE to FOOL.

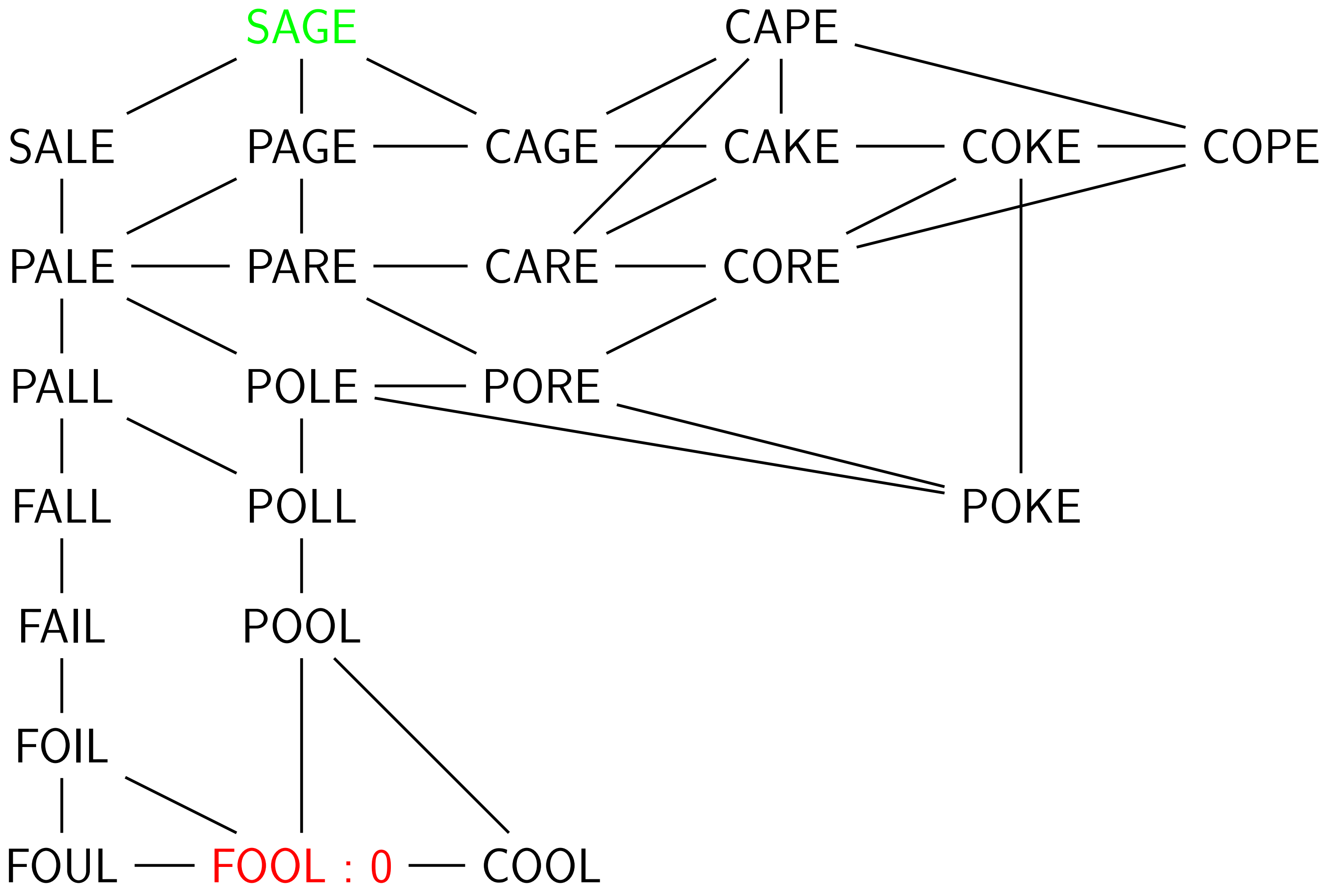
Of course, we have left out many many possible words, but this map gives us some very interesting information.

It shows us that there are many solutions to the problem.

It shows us that there are dead ends, and worthless steps that just lengthen our journey.

Moreover, if we gave a computer just the list of words, it could set up a similar map all by itself. Of course, it can't see the map the way we can, but from our maze investigation, we know the computer can find its way around the map very efficiently.

Let's start at FOOL: distance 0



Let’s start at FOOL: distance 0

We can even determine the number of steps necessary to transform ANY word into FOOL.

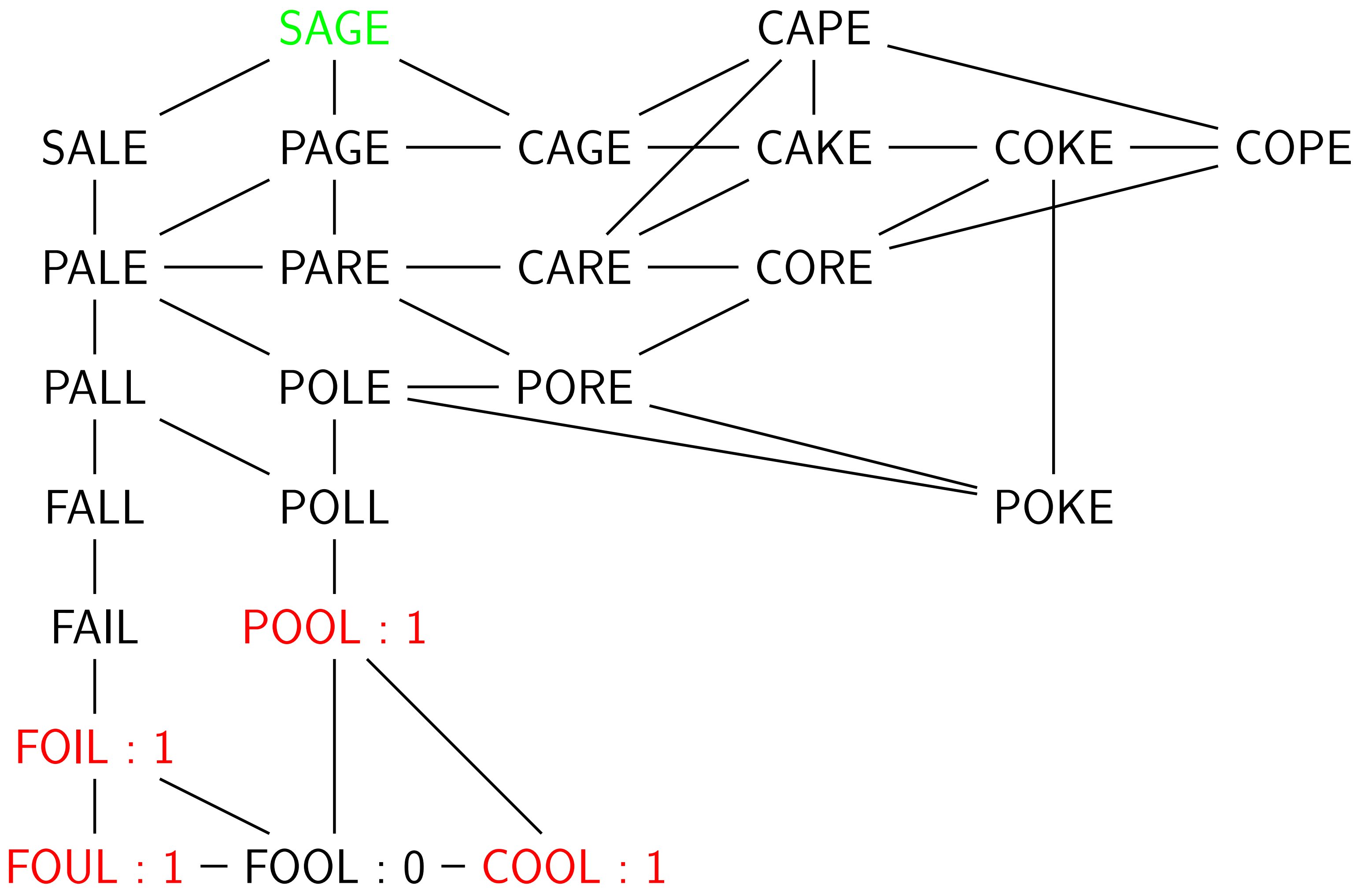
Mark FOOL’s distance as ”0”.

Every word in the map that touches FOOL now has distance 1.

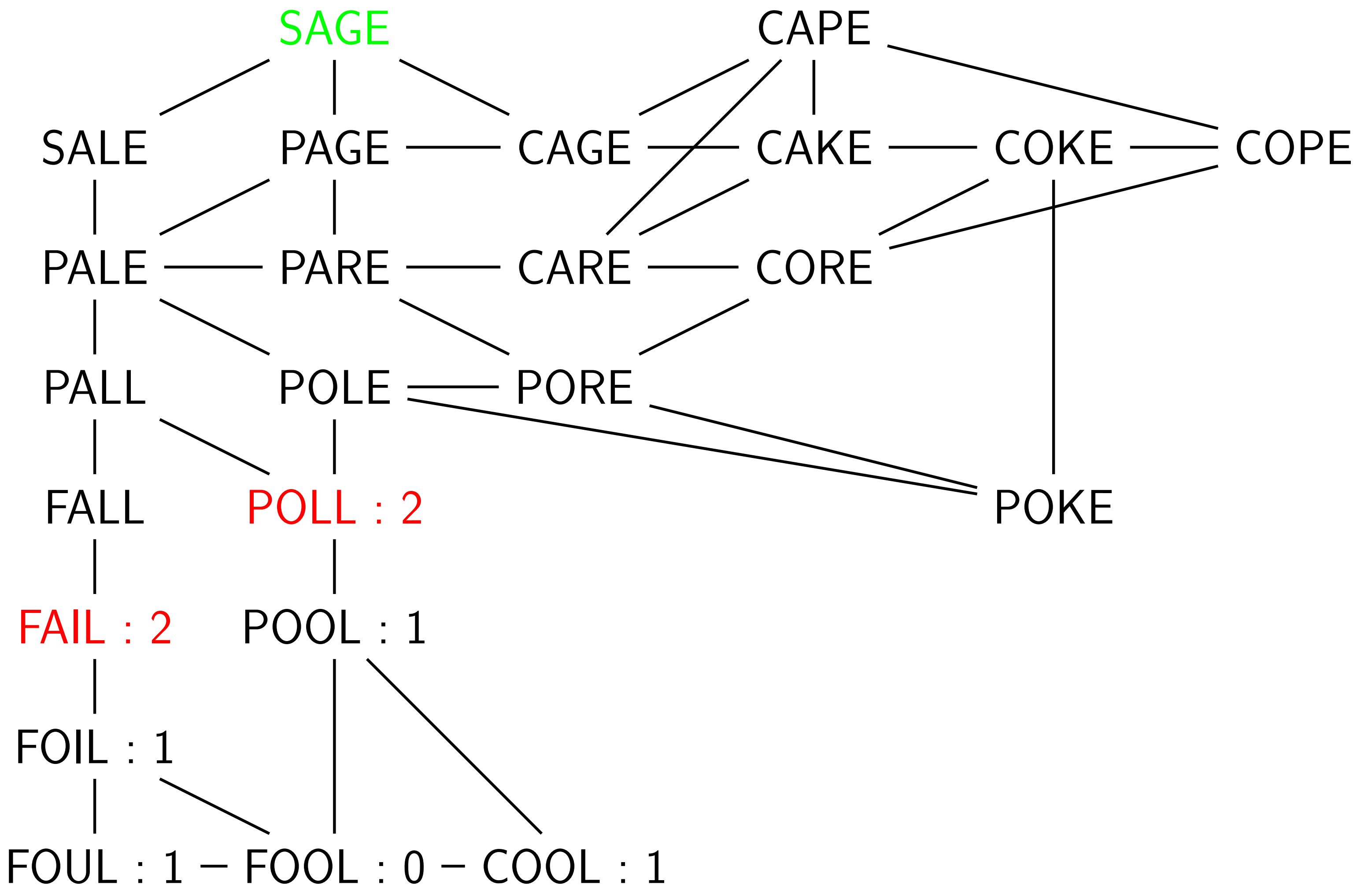
Any unmarked word that touches a word of distance 1 now has distance 2.

Keep going until you can reach no more words. If any words remain unmarked, you can’t transform them to FOOL!

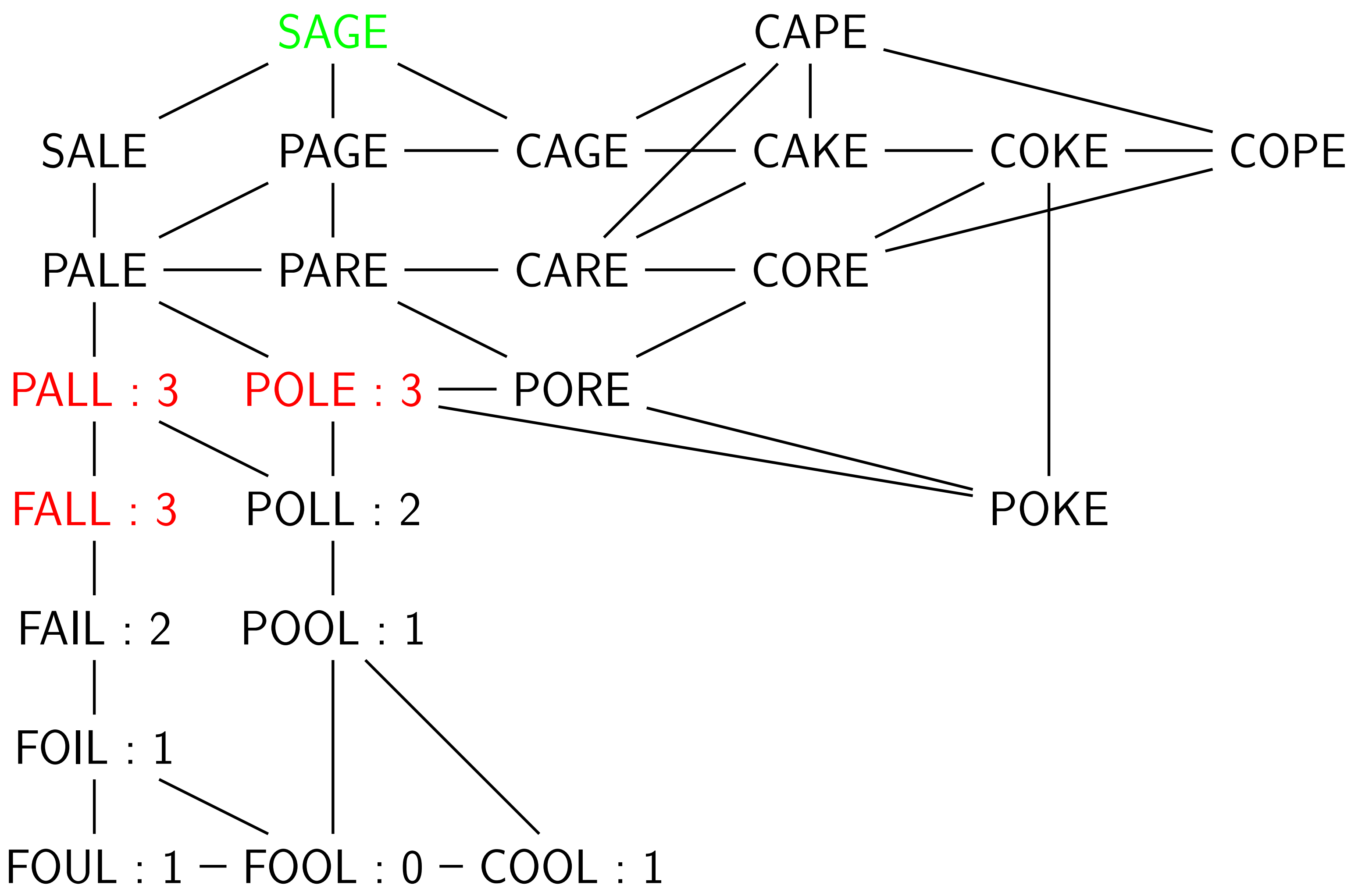
COOL, FOIL, FOUL, POOL: distance 1



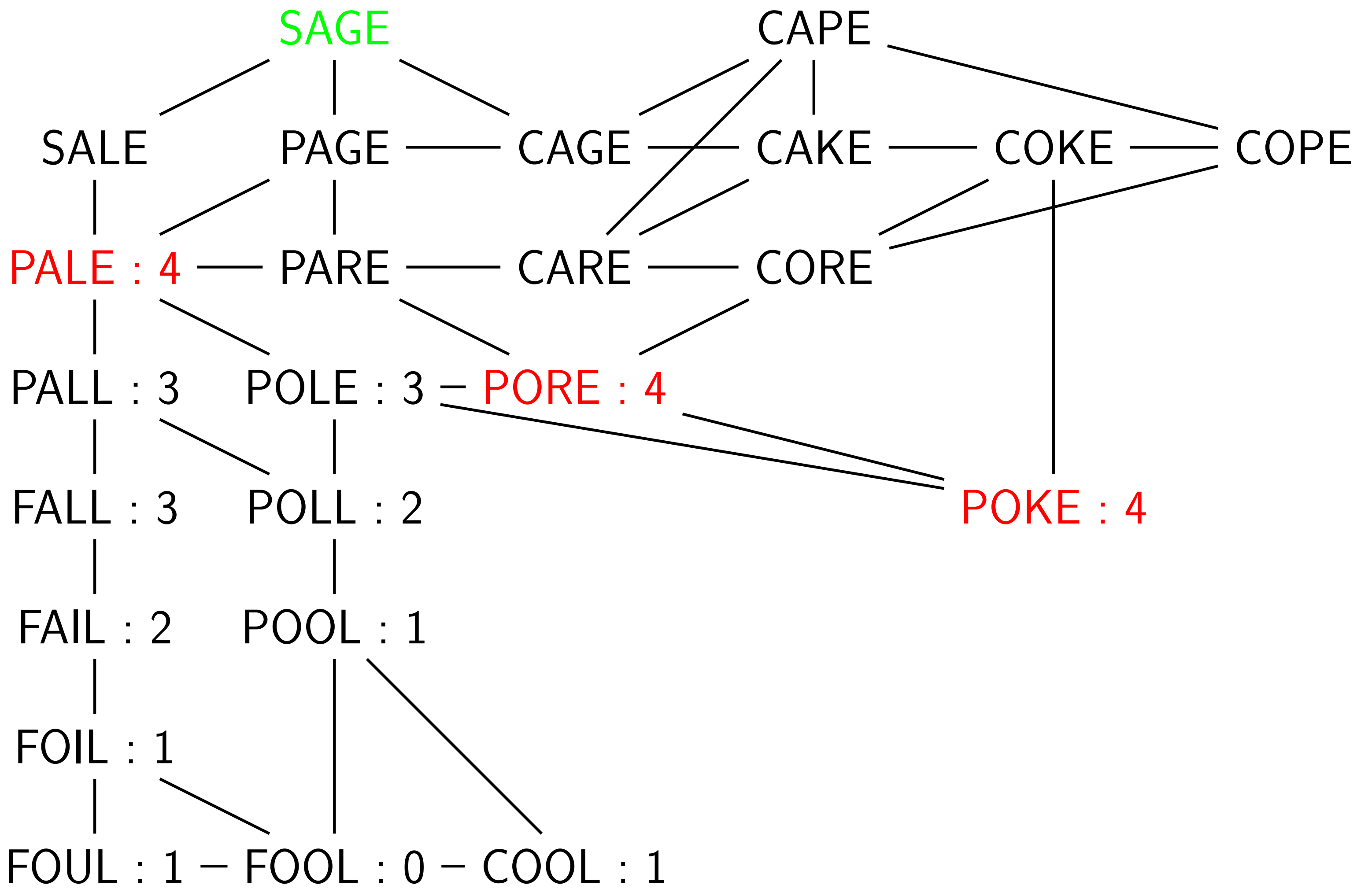
FAIL, POLL: distance 2



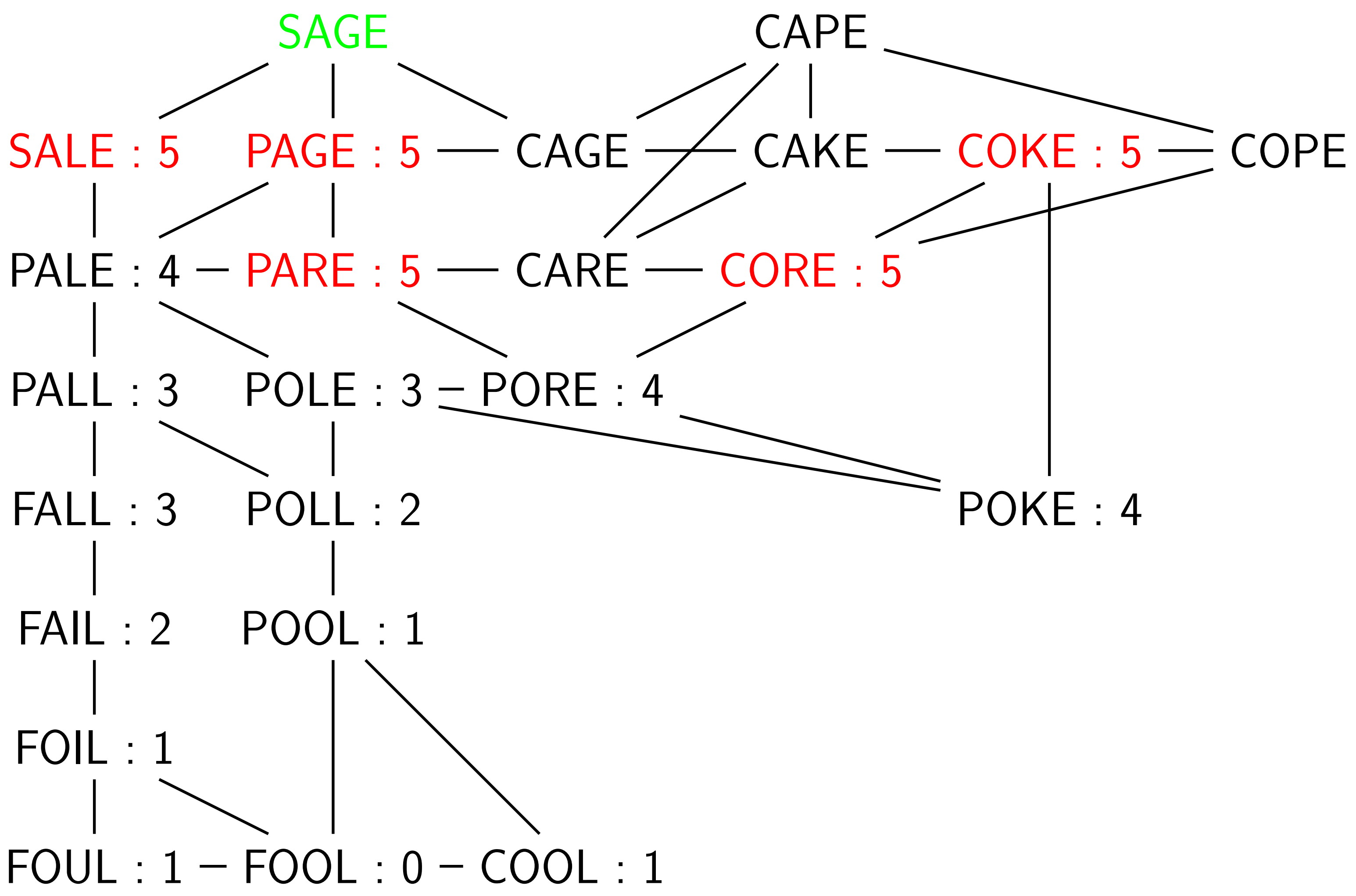
FALL, PALL, POLE: distance 3



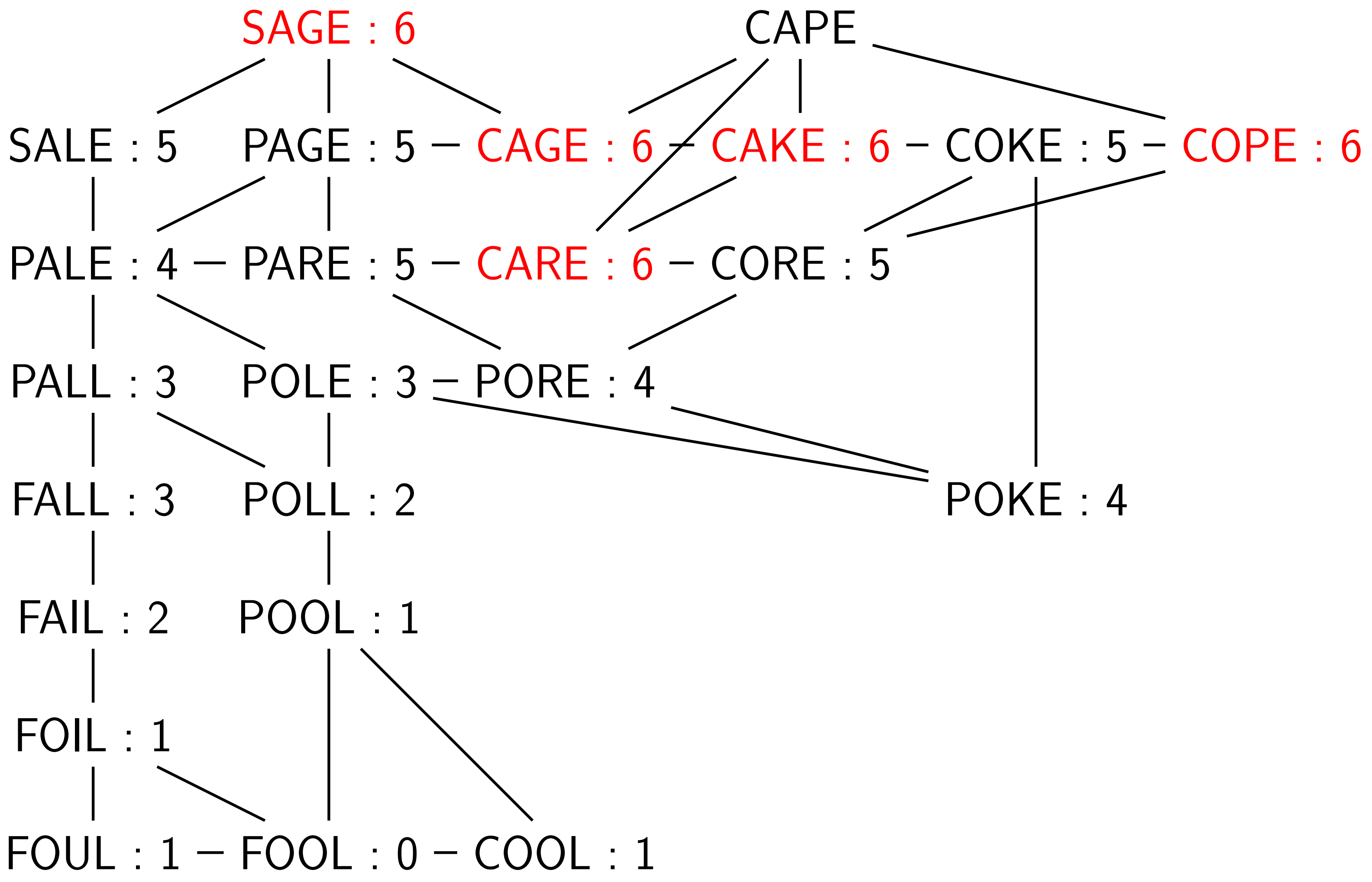
PALE, POKE, PORE: distance 4



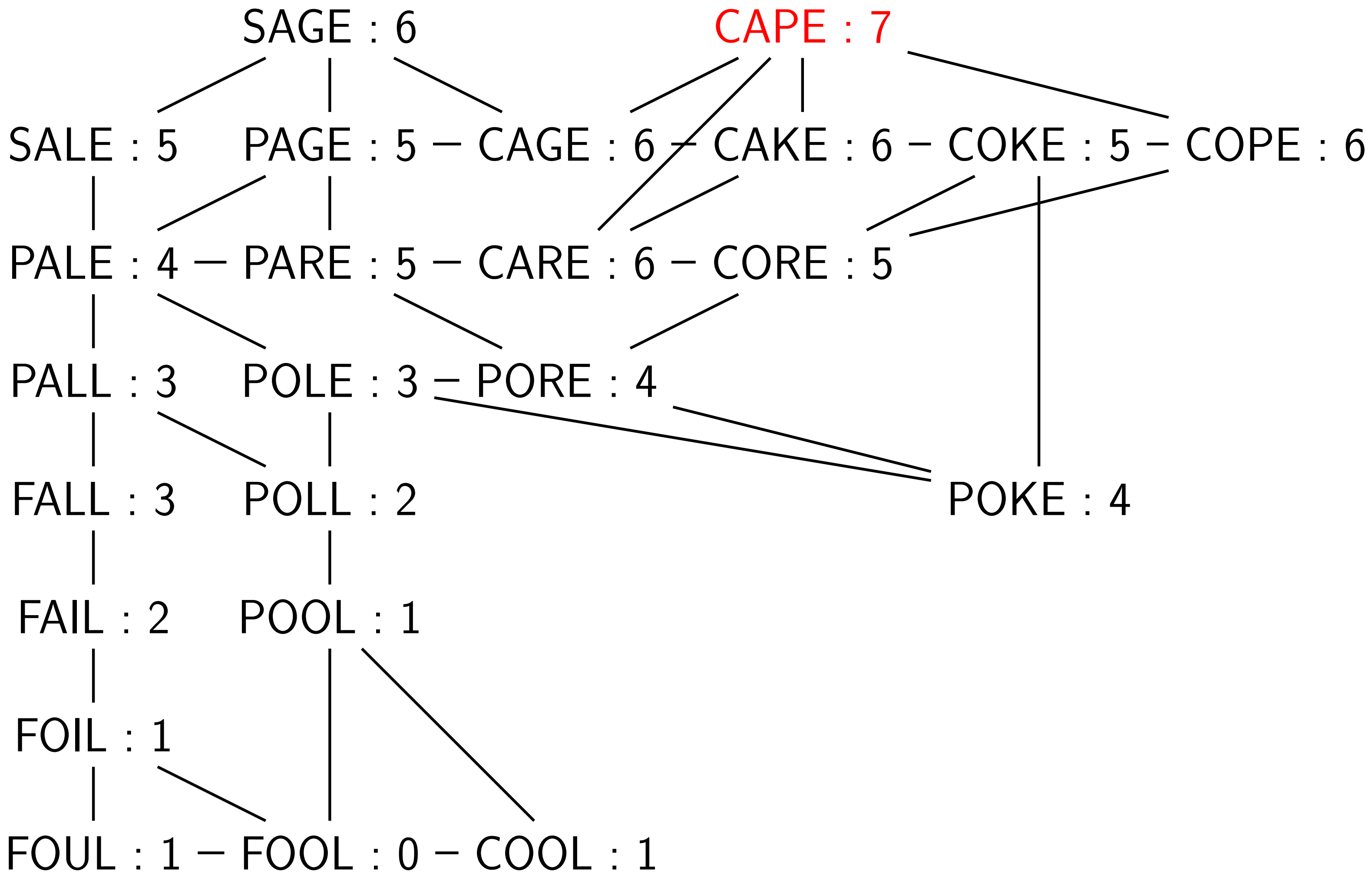
COKE, CORE, PAGE, PARE, SALE: distance 5



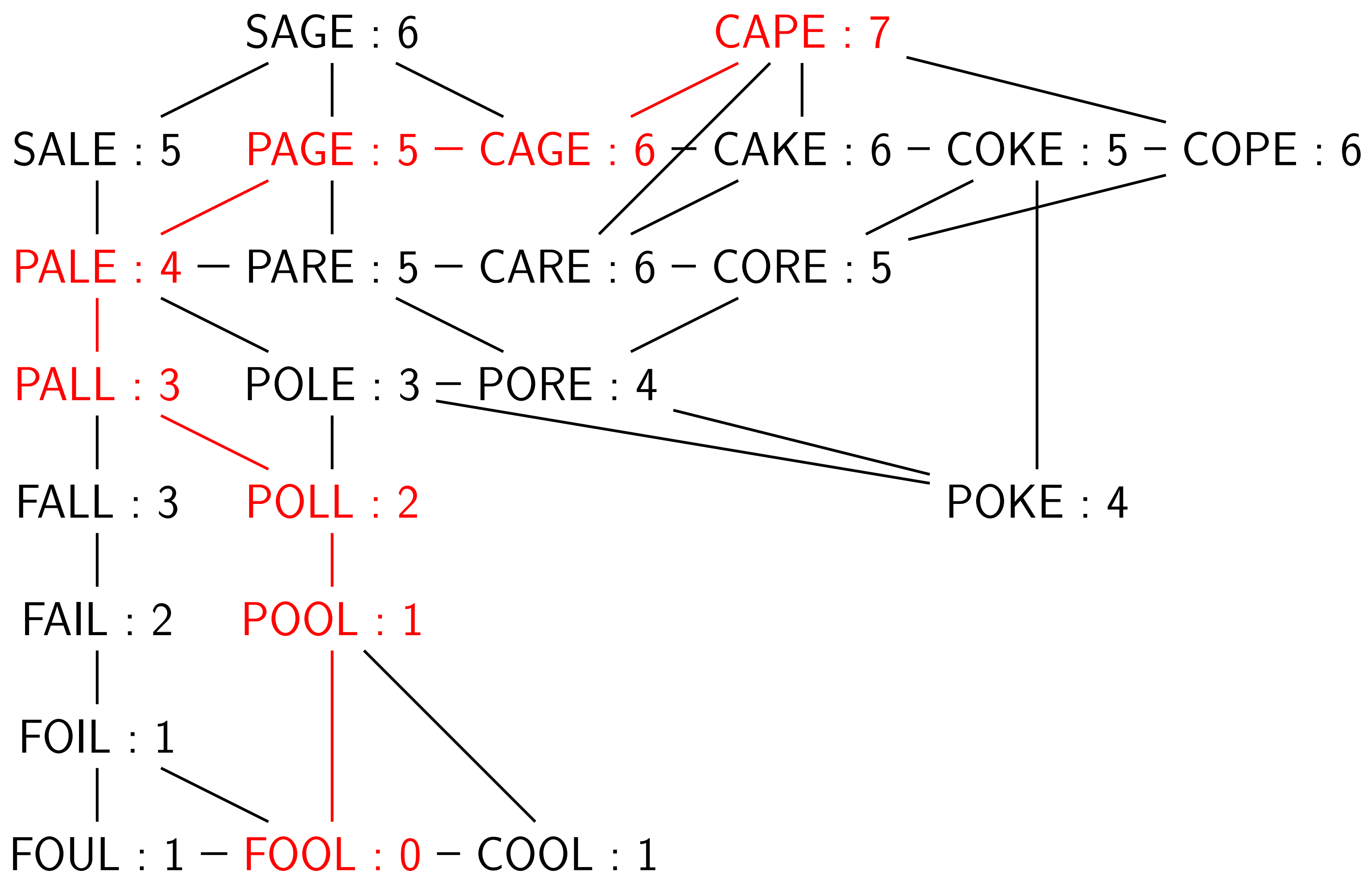
CAGE, CAKE, CARE, COPE, SAGE: distance 6



CAPE: distance 7



To connect FOOL to CAPE, go backwards from CAPE!



We changed the problem, and decided to get to CAPE

We can use our marked map to determine the transformation of any word into FOOL.

Pick a starting word, such as “CAPE”. It has a distance 7. To find the solution, move to any neighboring word that is one unit closer, and keep doing it til you reach FOOL.

One such path is CAPE, CAGE, PAGE, PALE, PALL, POLL, POOL, FOOL.

Of course, we can also easily solve our FOOL $\rightarrow$ SAGE problem now, and we even know that, limited to this set of words, we need exactly 5 intermediate words to make the trip.

How would a computer handle a doublets puzzle?

So if we give a computer the words **SAGE** and **FOOL**, what have we learned about our own solution methods that can be translated into computational thinking?

The first useful heuristic is that we should look at the START or END word and generate the immediate neighbors. We can do this automatically, because we know that **SAGE** has four letters, and we should try varying each letter to search for new words such as **CAGE** or **SANE**.

So the computer needs to know the alphabet (**A** through **Z**) and it needs a list of all words in English (or at least, in this case, all four letter words.)

That means the computer needs access to a database.

How would a computer handle a doublets puzzle?

By looking at the neighbors of $\mathbf{SAGE}$, the computer will either find no neighbors, in which case we have to stop the search, or it will find exactly one neighbor, in which case it should "move" to that new word, or it will find several neighbors.

If there are several neighbors, then the computer has to store all the neighbors in a list, so that if it makes a bad choice, it can backtrack and explore another choice.

That means the computer needs a memory in which to store temporary lists.

How would a computer handle a doublets puzzle?

If there are several neighbors, then the computer should try to pick the choice that looks most likely to help. We have several heuristics to guide us, including

- Try inserting a letter from the END word;
- Try matching the vowel and consonant pattern of the END word;
- Prefer words with more common letters;
- Prefer words that have many neighbors;

That means the computer needs the ability to make decisions by evaluating its choices and taking the best one.

How would a computer handle a doublets puzzle?

One thing we forgot to warn the computer about, a new heuristic:

While making a path, never add a word that is already part of your path!

Otherwise, if there are any loops in the word map, it would be possible for the computer to go around and around in circles forever!

```
  SAGE -> CAGE -> CAKE -> SAKE -> SAGE -> CAGE -> CAKE -> ...
```

So the computer needs some tests to reveal special problems like this that we didn't think about.

How would a computer handle a doublets puzzle?

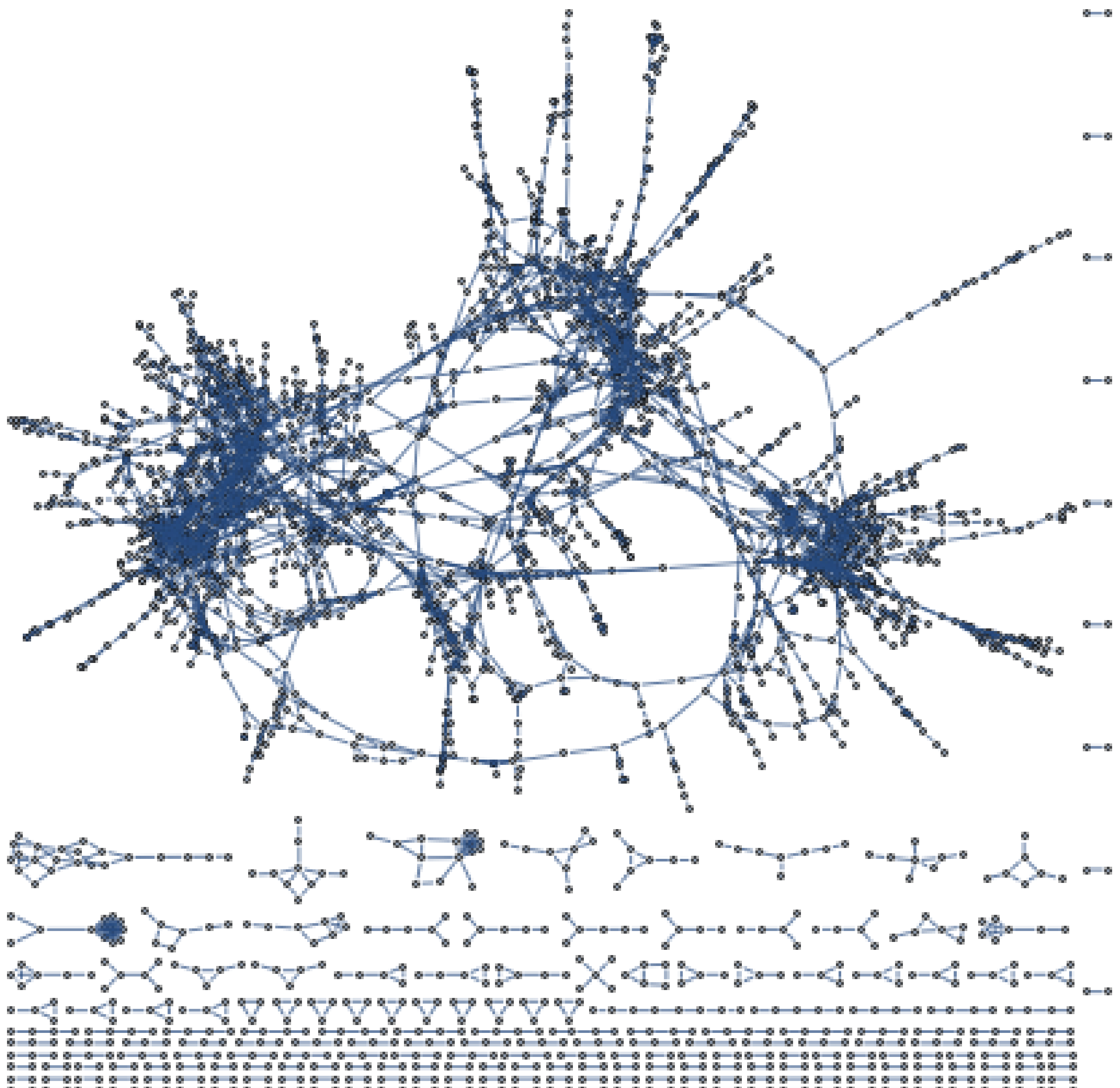
Using these simple ideas, the computer could efficiently check every possible path from **SAGE**, and if there is a path to **FOOL**, it will eventually find it.

We hope that the heuristic rules will help the computer find the right path sooner, without having to do a brute force check of every possible path.

Now we have a reasonable idea of how a computer could automatically search for solutions to a doublet puzzle.

A map of 5 letter words

Out[4]=



A map of 5 letter words

If we were playing Doublets using 5 letter words, and we had a computer, we could make a map of all the connections.

Here is such a map, using more than 5,000 five letter words. In this map, each word appears only as a dot, so we are just seeing the abstract connection pattern.

Most words are connected, although there are some disconnected sets, and even solitary, unconnected words. One of them is **ALOOF**.

You can see a few cases where words are connected but very far apart. One such pair is COMEDY and CHARGE which can be connected using a sequence of 48 words, some of them uncommon.

The fact that we can make such a map means that this is actually a fairly simple problem...for a computer.

Computational Thinking

In discussing how to deal with word puzzles and mazes and maps, we have seen several tools of computational thinking:

- **map**: seeing how things are connected;
- **lists**: gathering data into a single location;
- **indexing**: rapid access to data using an index;
- **distance**: begin able to measure how far we are from a solution;
- **heuristics**: rules of thumb for making small choices;
- **greedy algorithms**: taking a step that makes the biggest immediate improvement;
- **brute force algorithm**: just trying every possibility;
- **backtracking algorithm**: trying one choice, but remembering the alternatives:

Two styles of solving problems

Two styles of solving problems

Is an algorithm more like Captain Kirk or like Mr Spock?

We have seen two ways to turn SAGE into FOOL.

One way is haphazard - we check to see if we can make a greedy move, otherwise we look at our choices and evaluate them, taking the best and saving the rest.

The other method spends a great deal of time preparing a map, and then calmly says "Go here, then here, then here, and that's the fastest way."

The mathematical, organized method is nice if you can discover it, and have the time to set it up. The one-step at a time, rule-of-thumb approach may not always work, may not be the fastest, but it may be better at handling problems where the data changes, or it's really hard to see the big picture.

Socrative Quiz PartIV_Quiz2

IUZGAZ34E

1. If a pair of five-letter words differ in exactly two locations, then it is always possible to connect them using doublets. (True, False)
2. When working on the shortest path problem for a city-to-city road map, what can happen each time we update the table? (A: all distances must decrease; B: at least one distance must decrease; C: distances may decrease, or stay the same; D: distances may decrease, stay the same, or go up)
3. On a city-to-city road map, the shortest path is always the one that goes through the fewest number of intermediate cities. (True, False)
4. In a maze map, the shortest path is always the one that goes through the fewest number of rooms. (True, False)
5. If we have used a maze map to determine the distance from room A to all the other rooms, then to find the shortest path from room A to room W, we start at room W and work backwards. (True, False)
6. When solving a doublets problem, the first step is to choose a word that is halfway between the start and end words. (True, False)