

Collections, Part Four

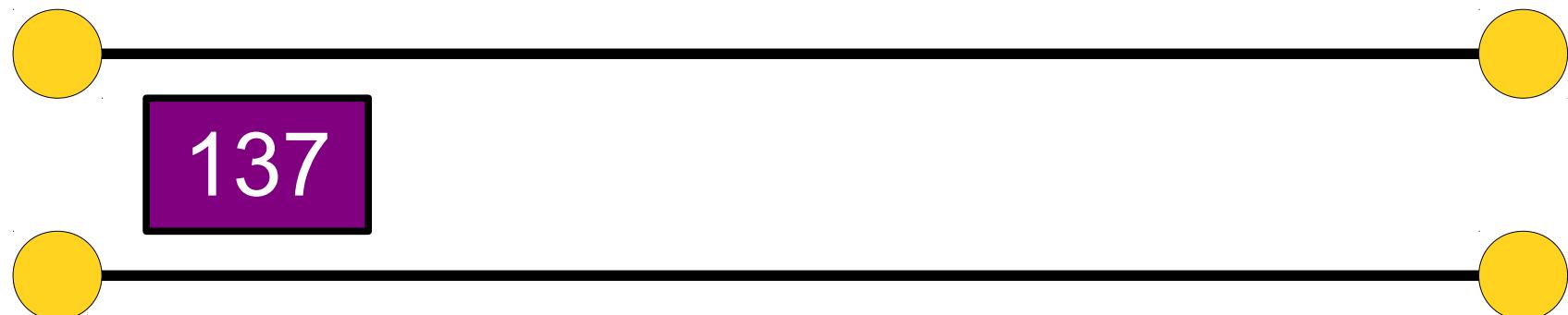
Announcements

- Assignment 1 due right now.
 - Due Wednesday with one late day and Friday with two late days.
- Assignment 2 (**Fun with Collections**) out, due **Wednesday, April 24** at 2:15PM.
 - Explore the different collections classes!
 - Teach the computer to write!

Queue

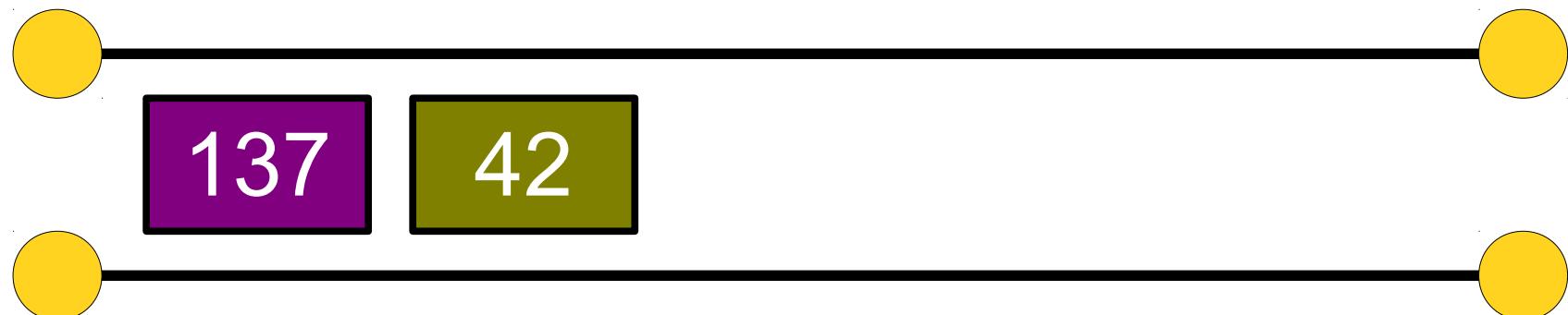
Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



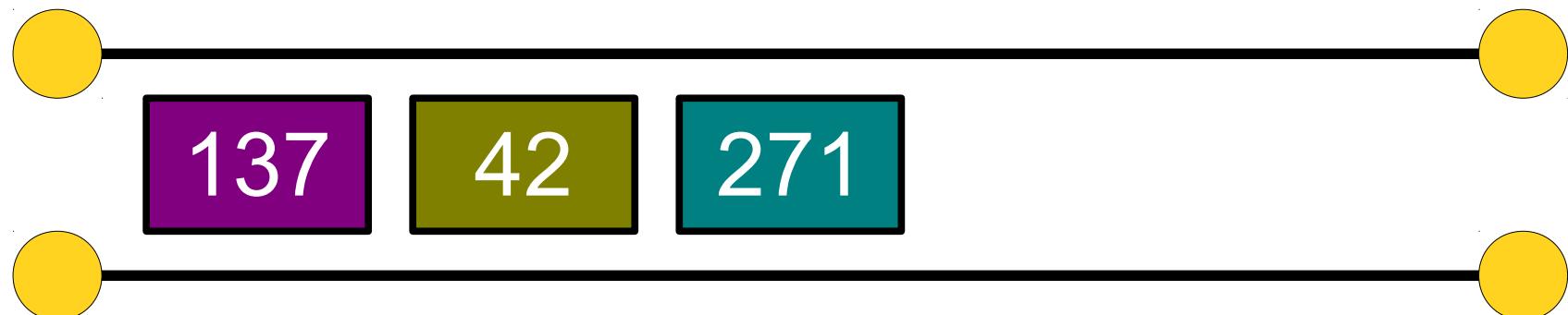
Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



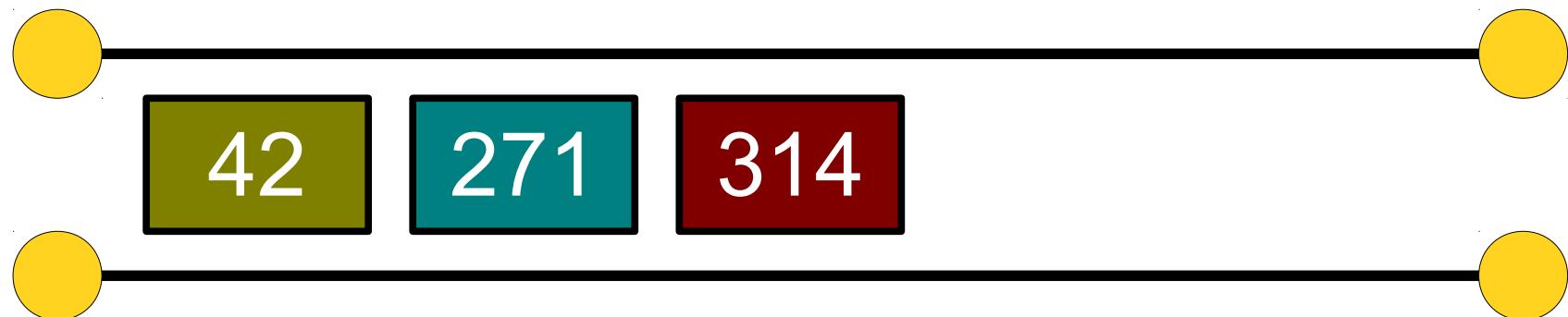
Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



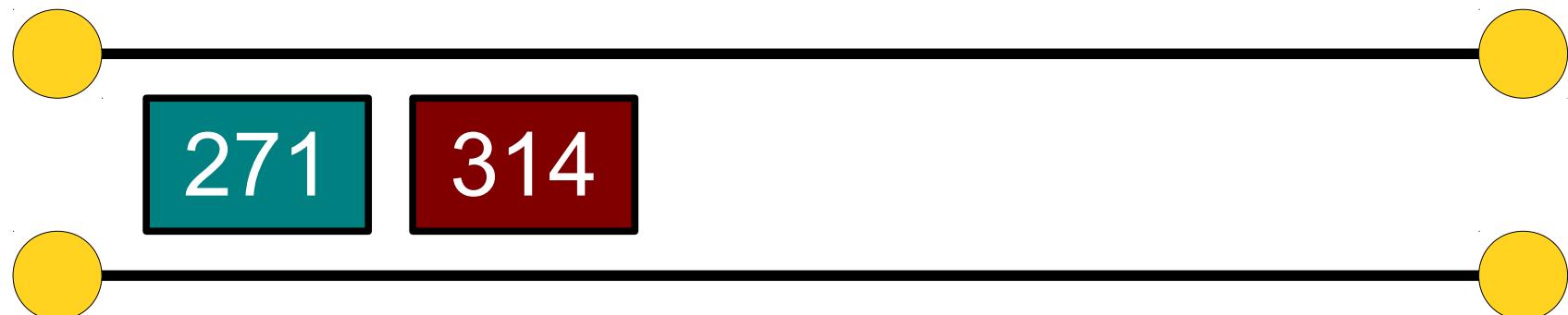
Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



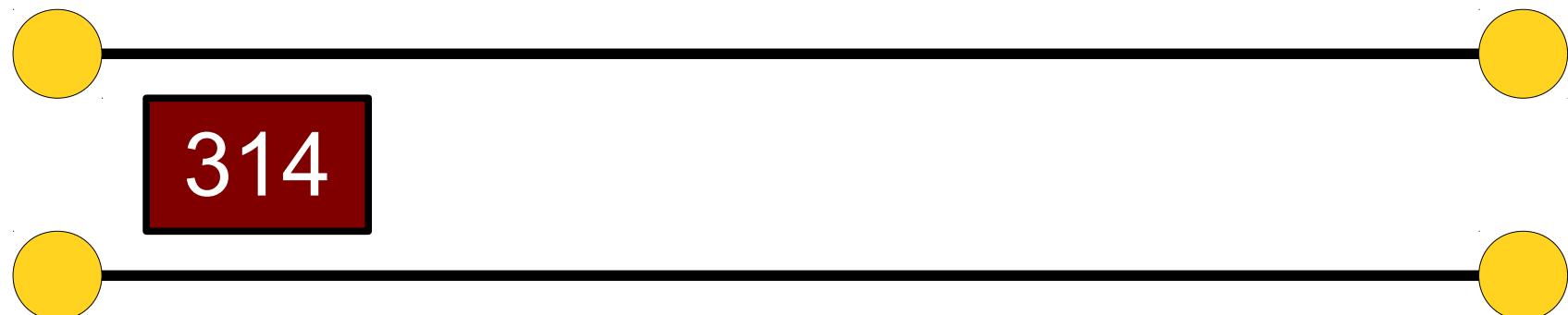
Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Listing All Strings

- Suppose we want to generate all strings of letters **A** and **B** of length at most three.
- How might we do this?

" "

"A"

"B"

"AA"

"AB"

"BA"

"BB"

" "

"A"

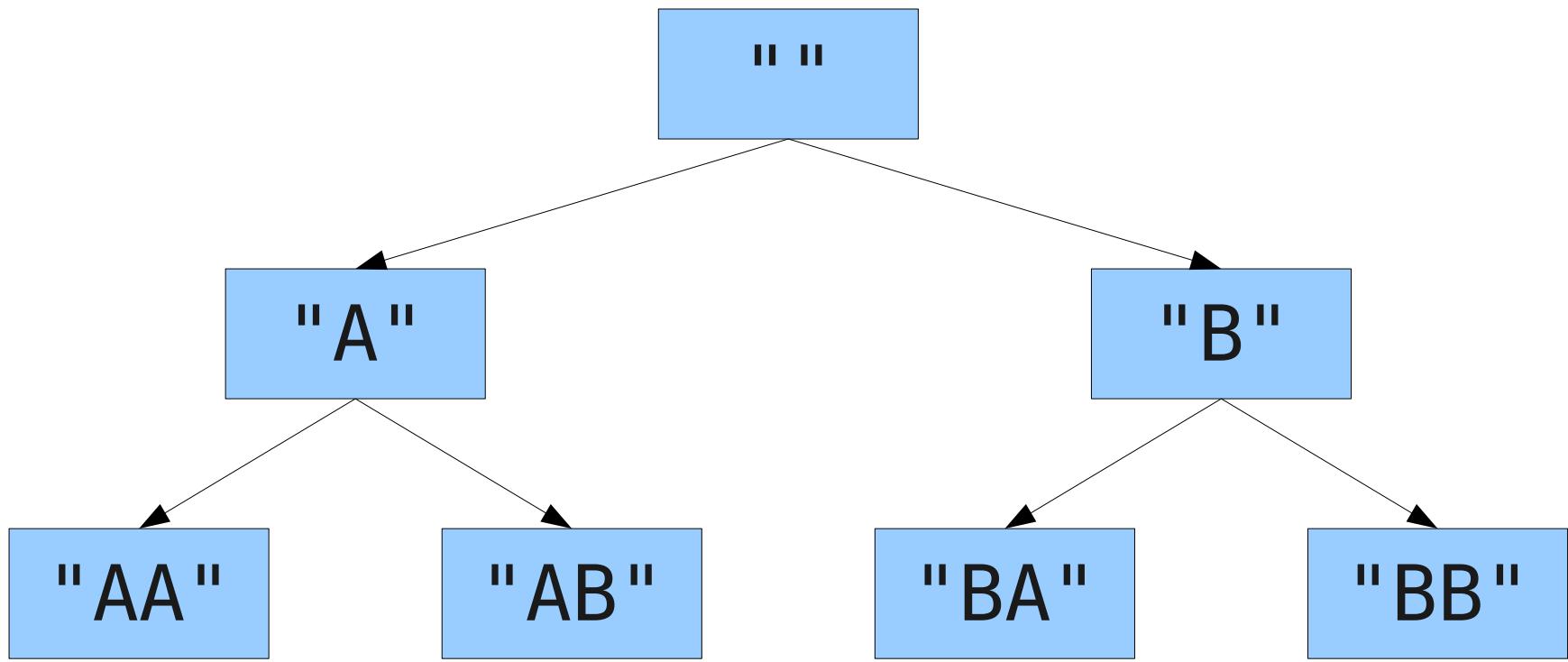
"B"

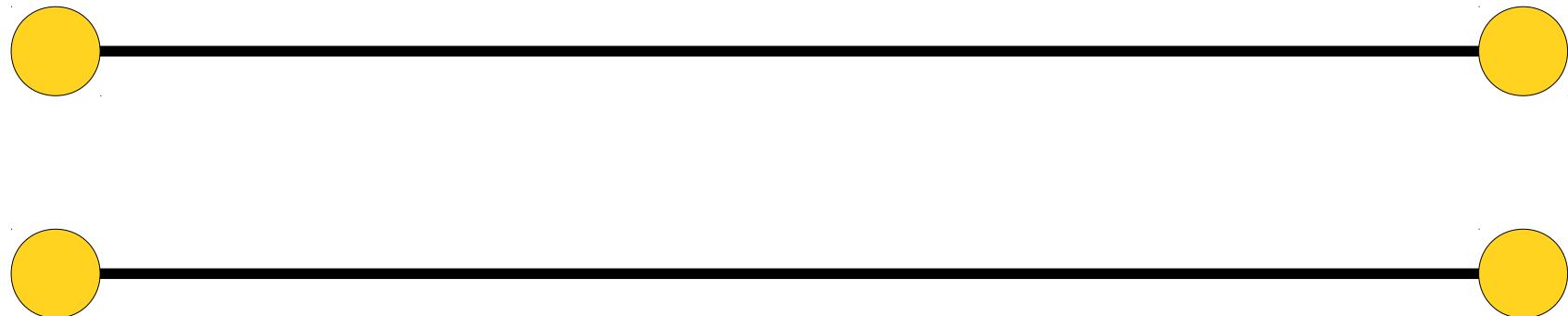
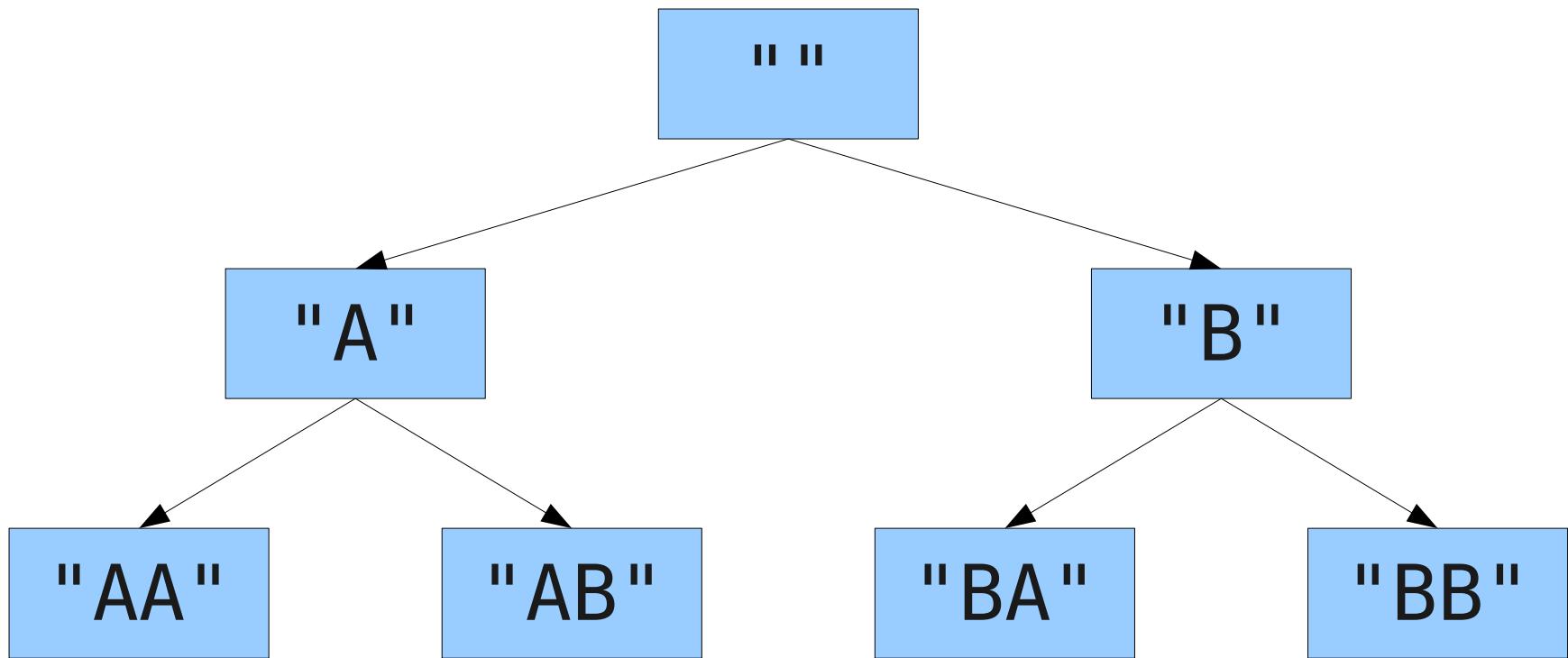
"AA"

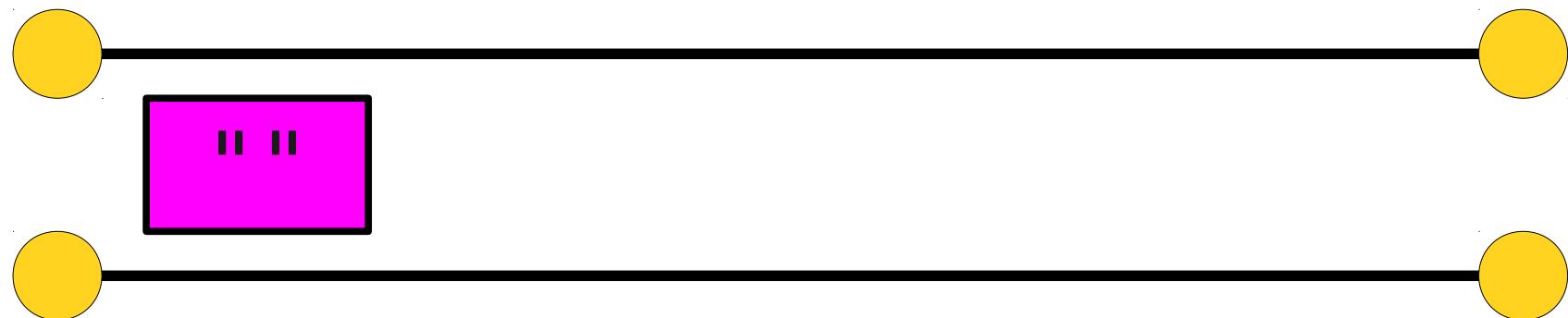
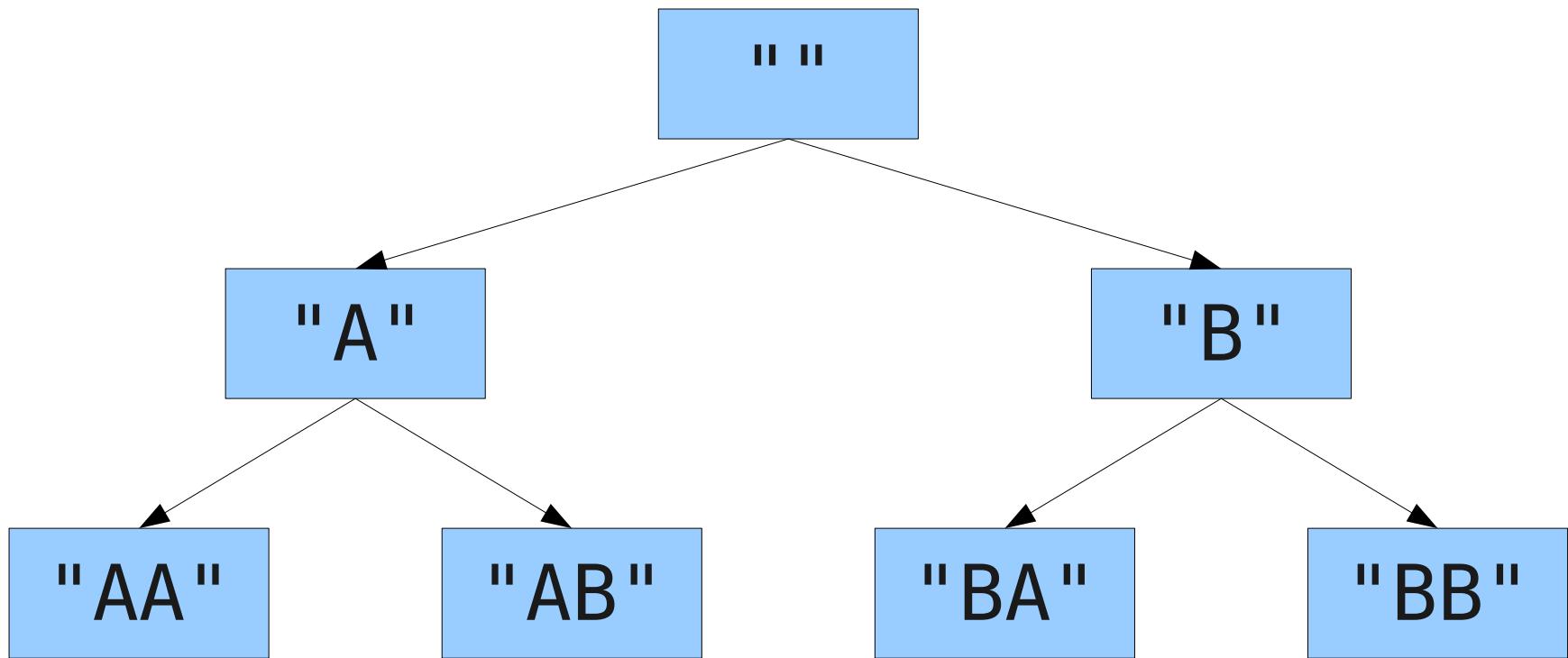
"AB"

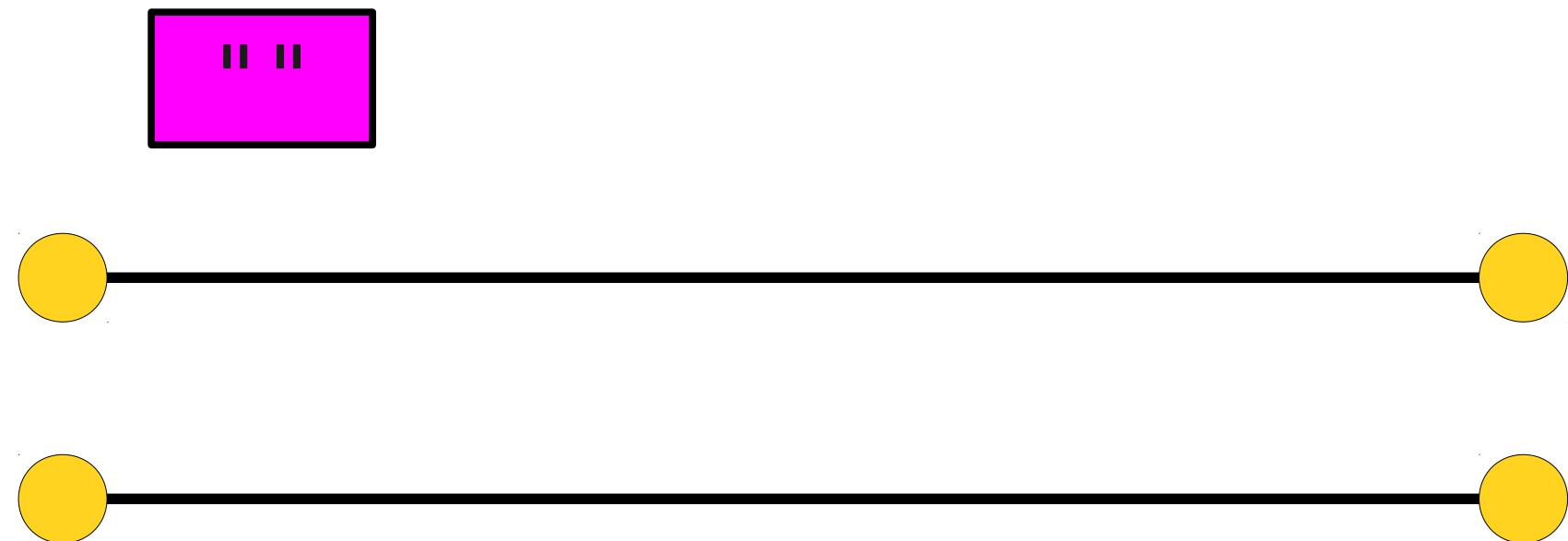
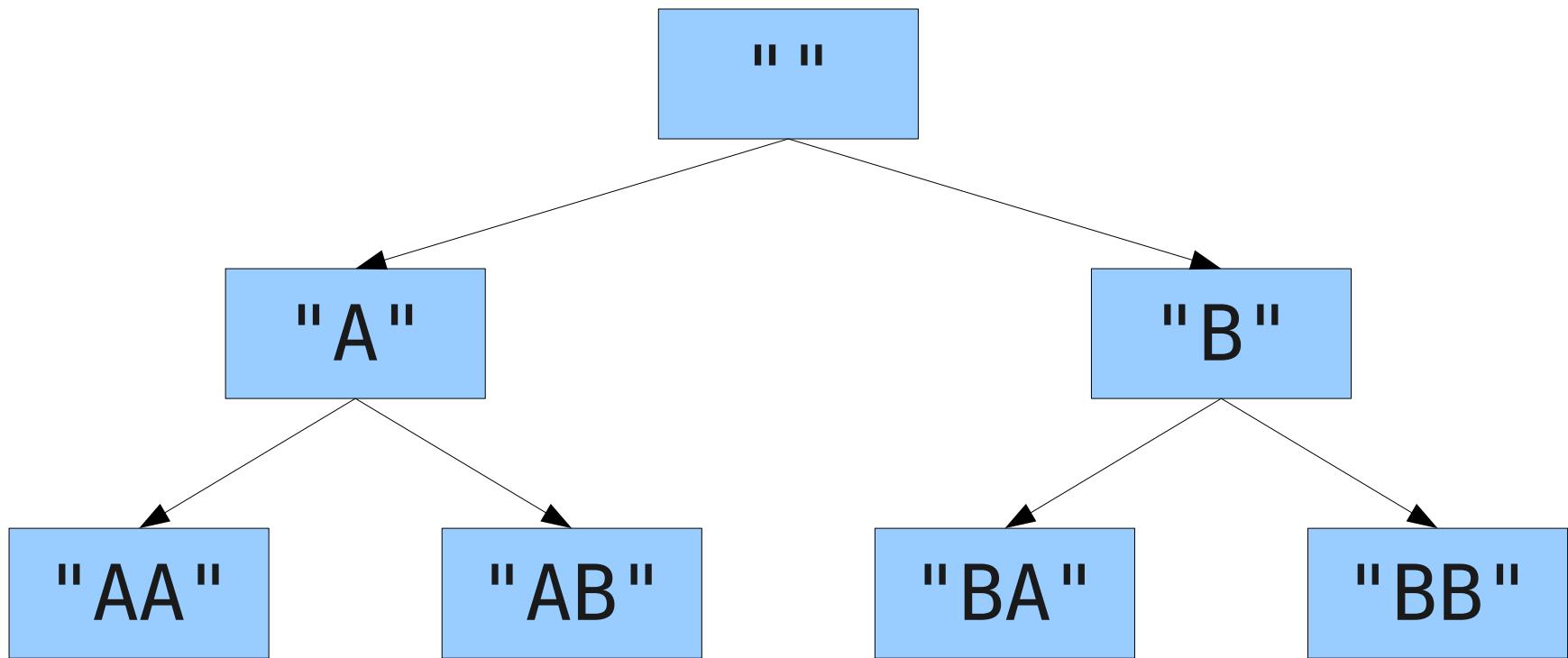
"BA"

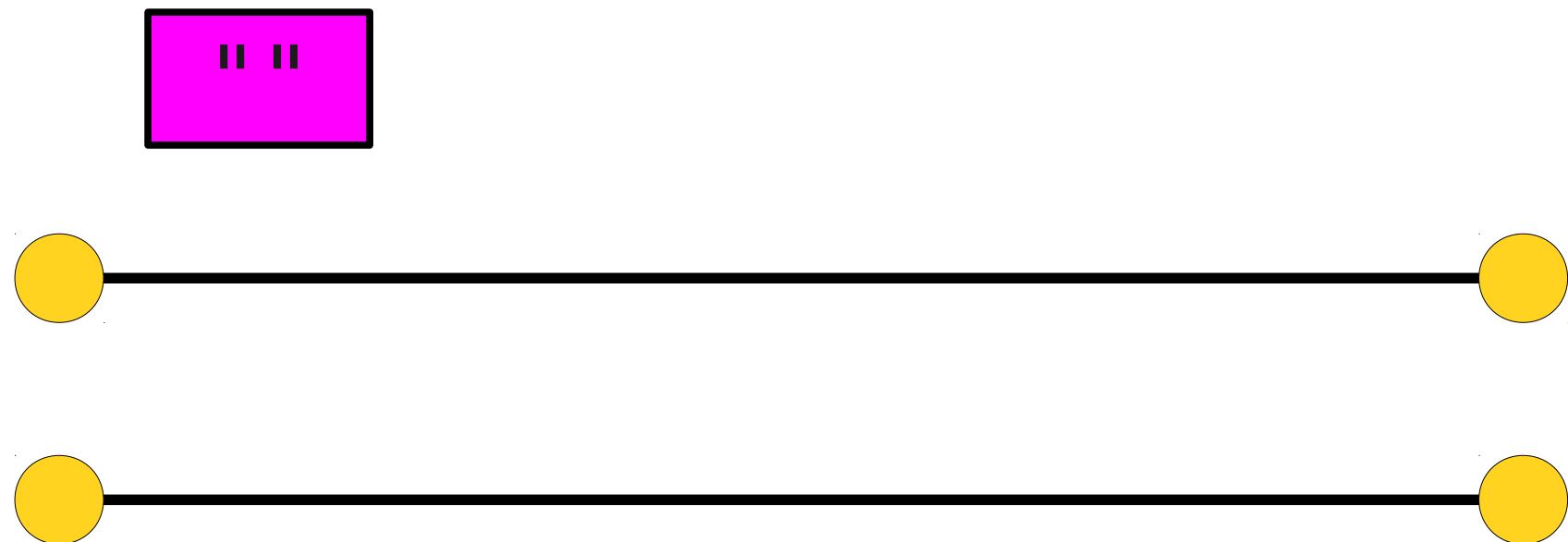
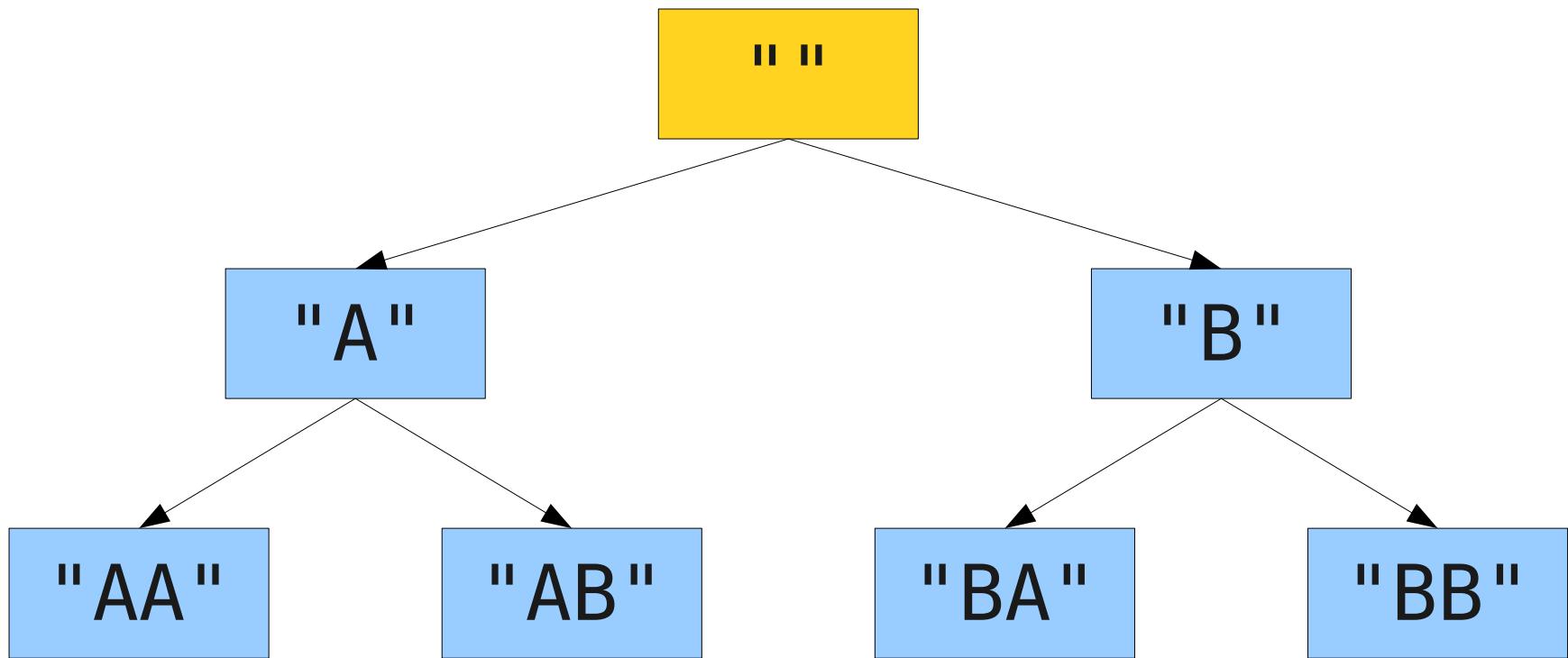
"BB"

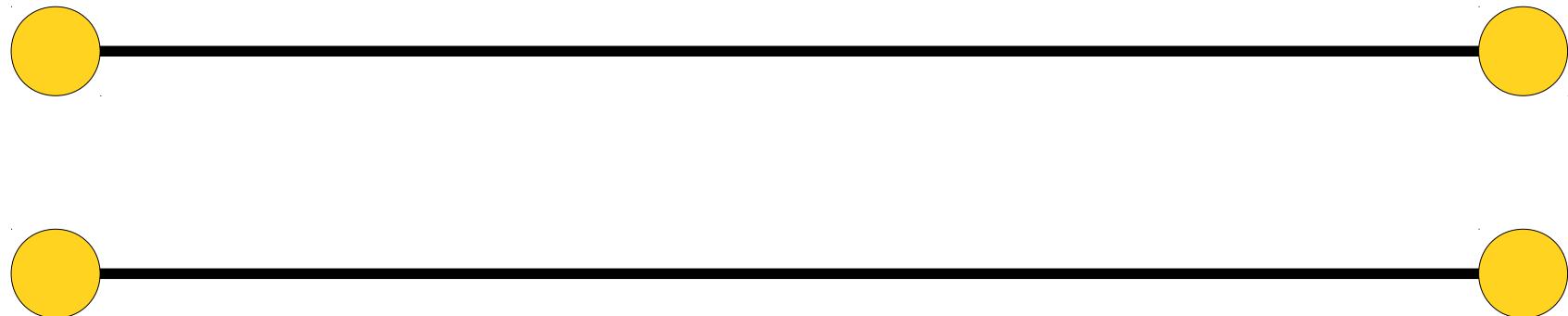
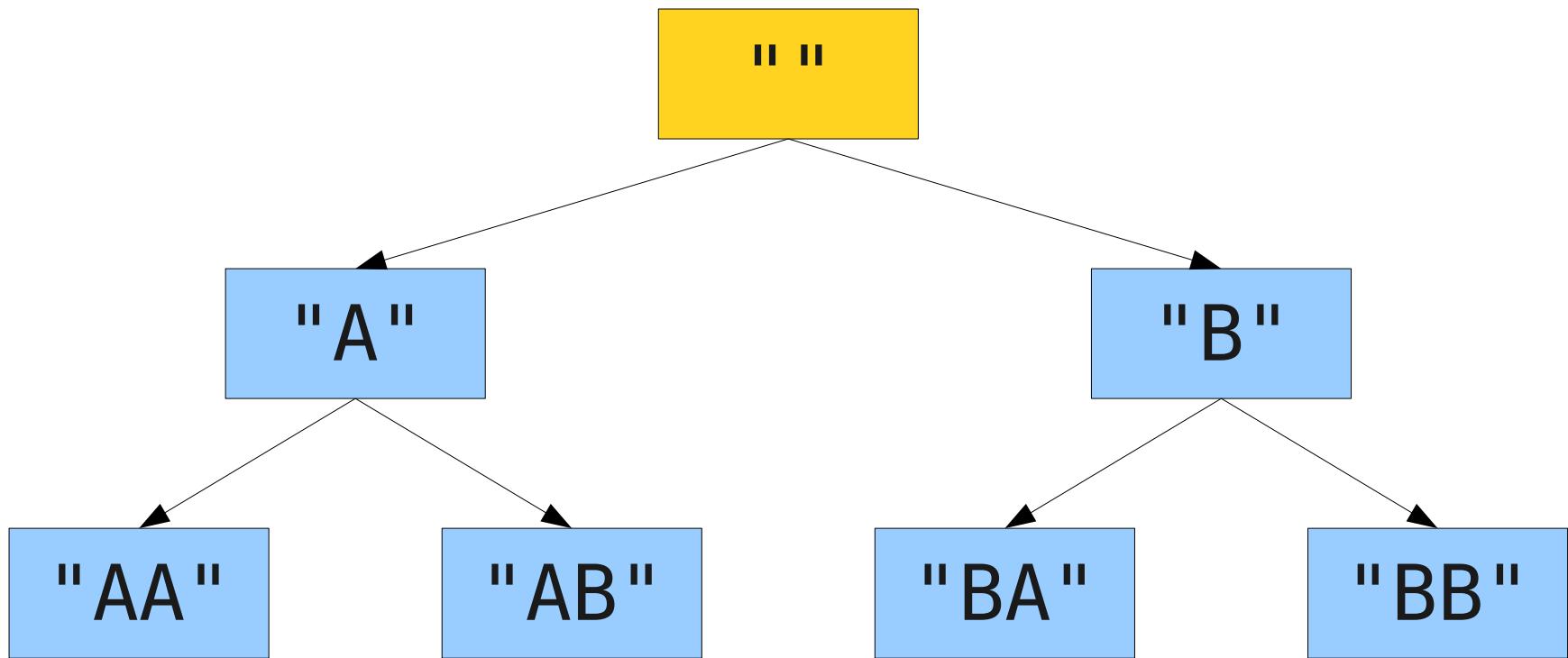


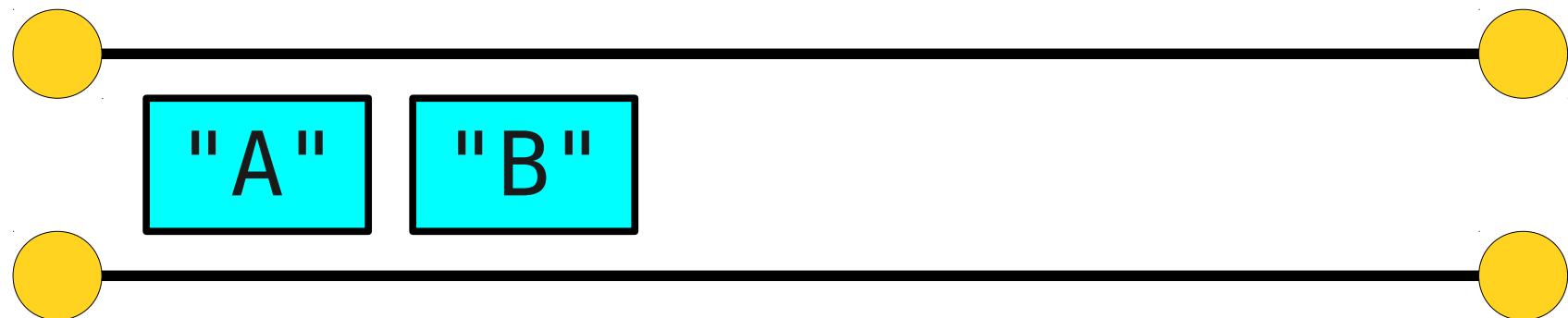
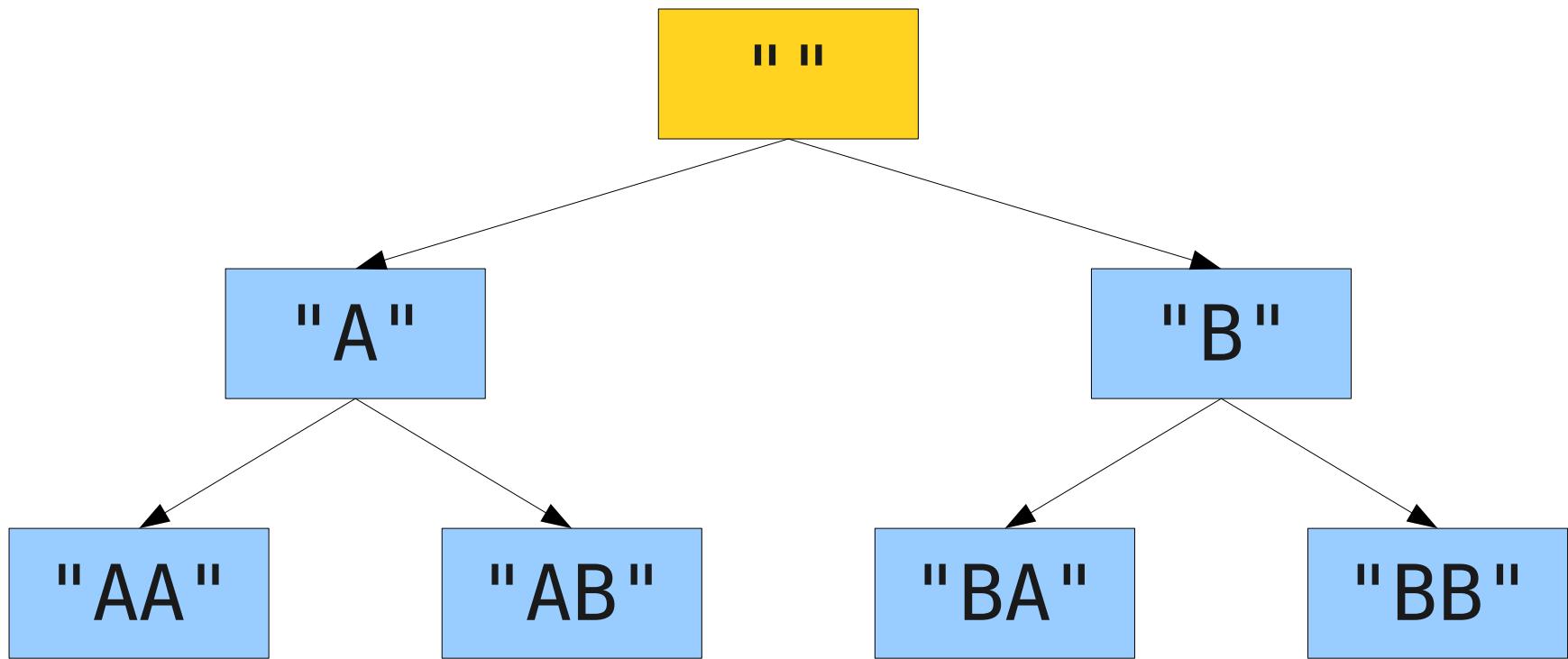


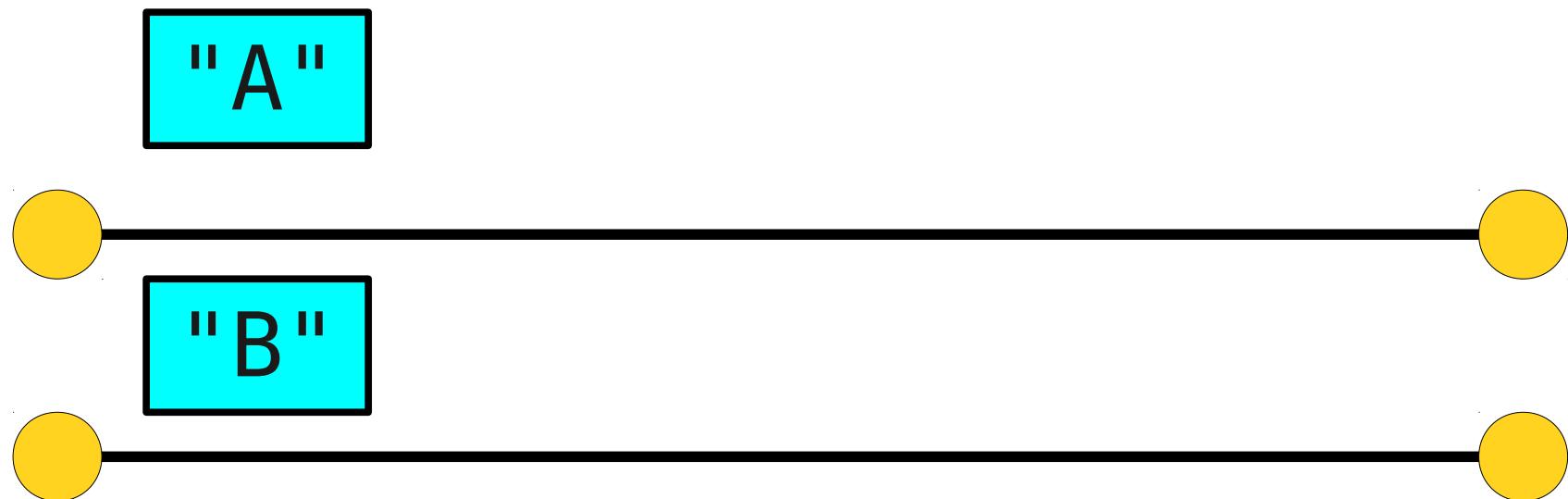
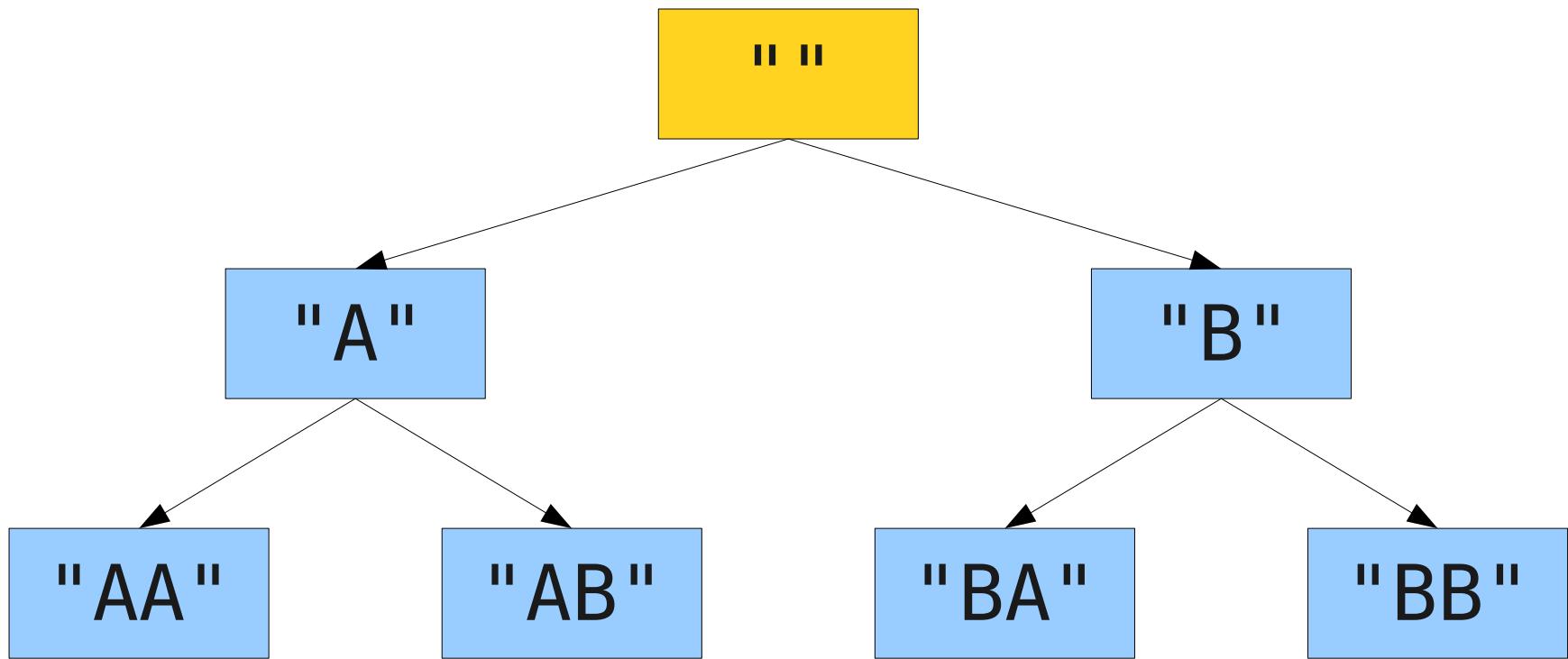


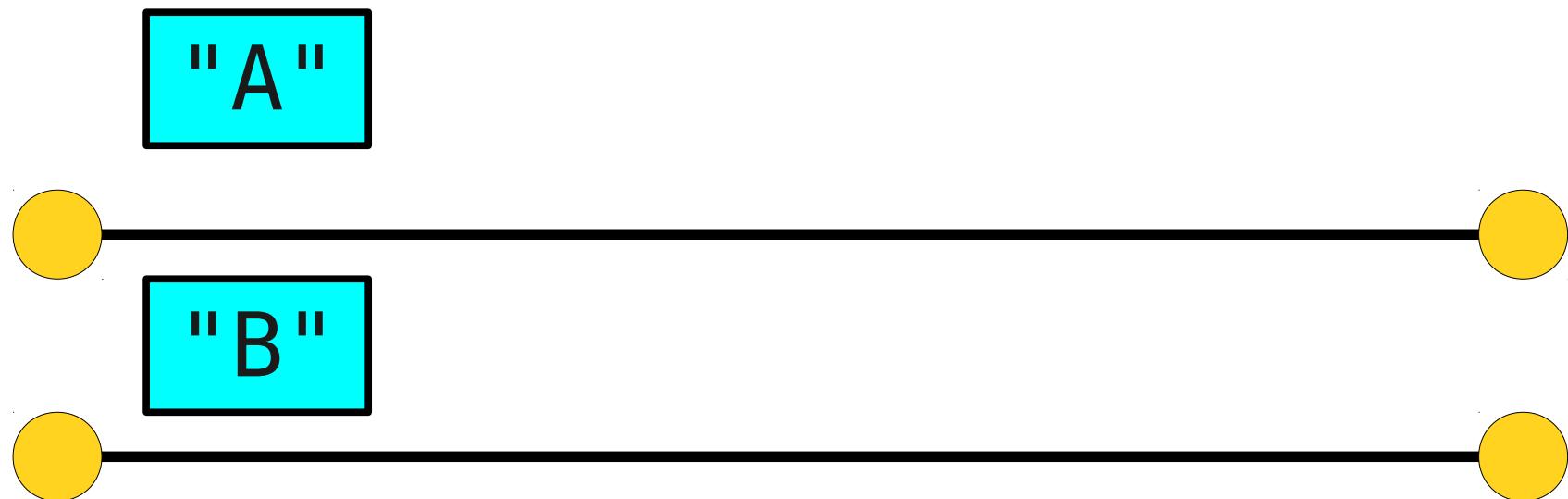
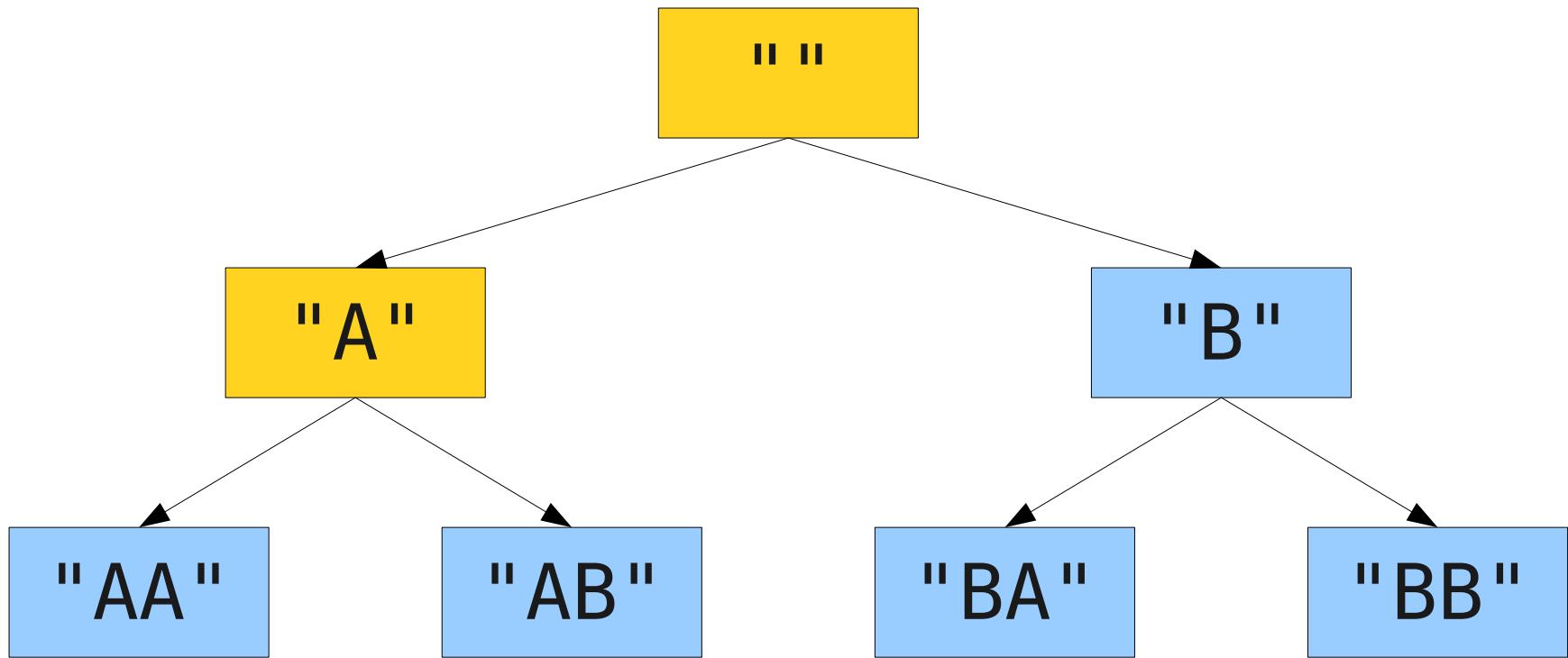


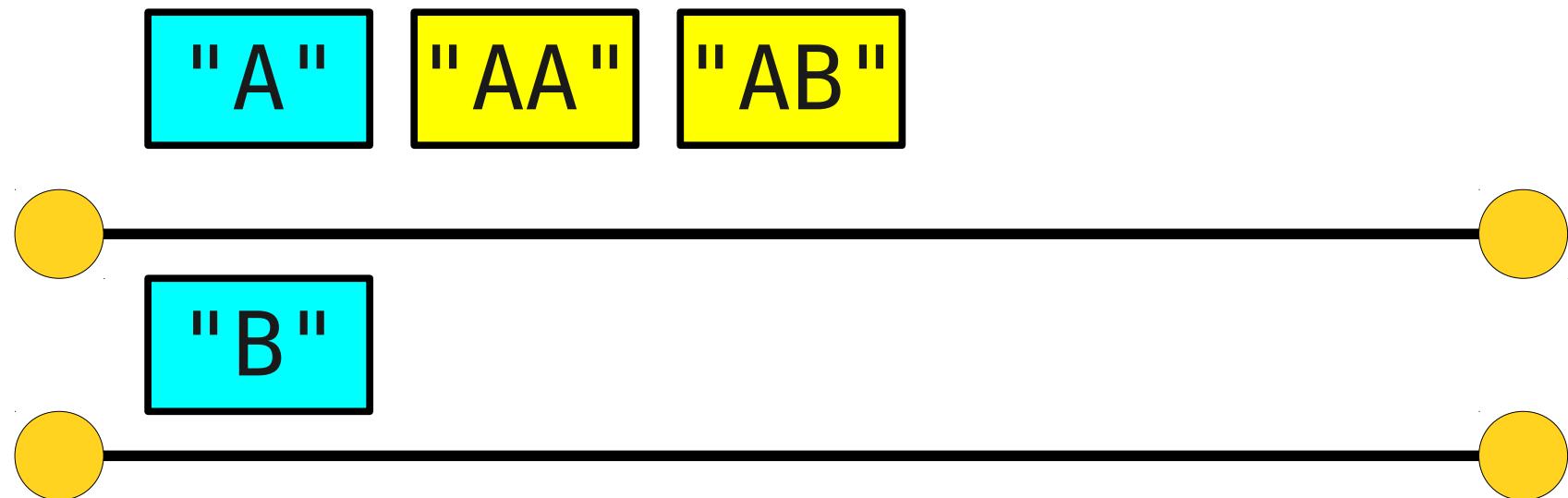
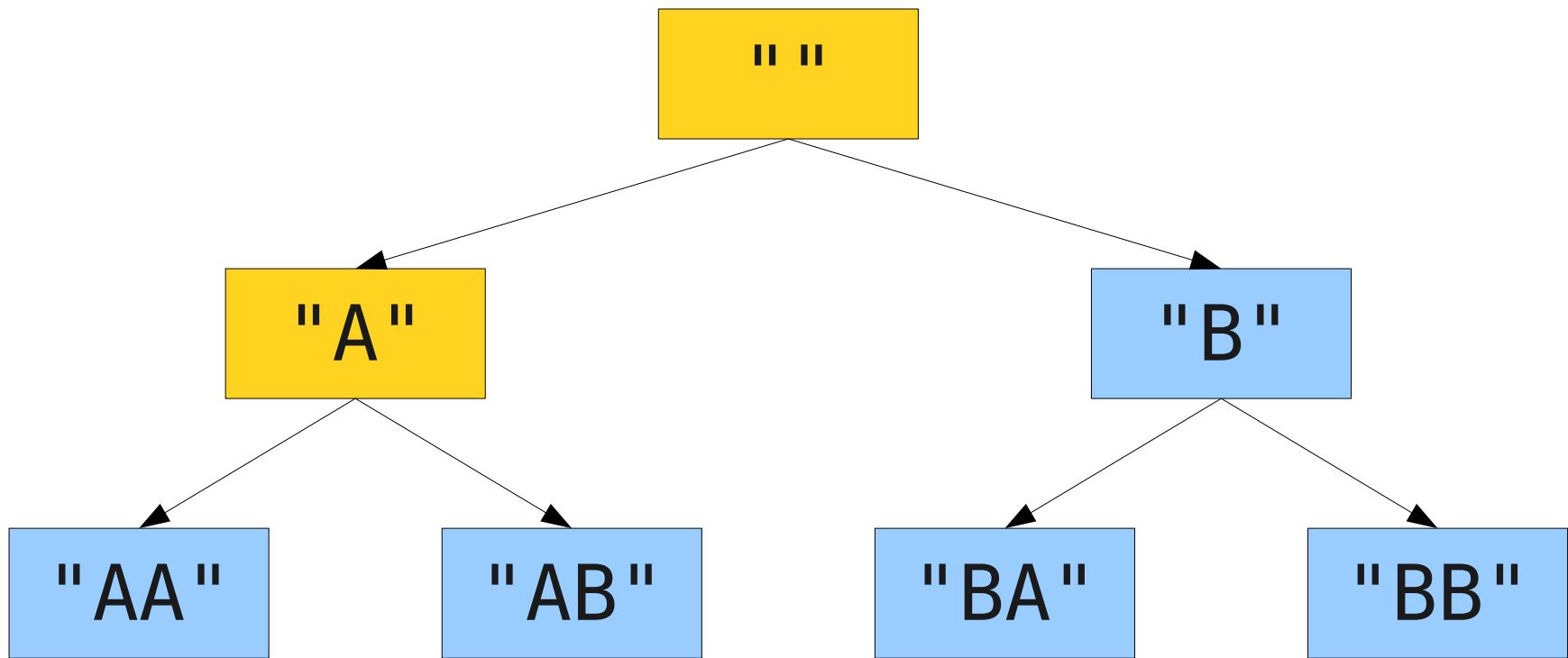


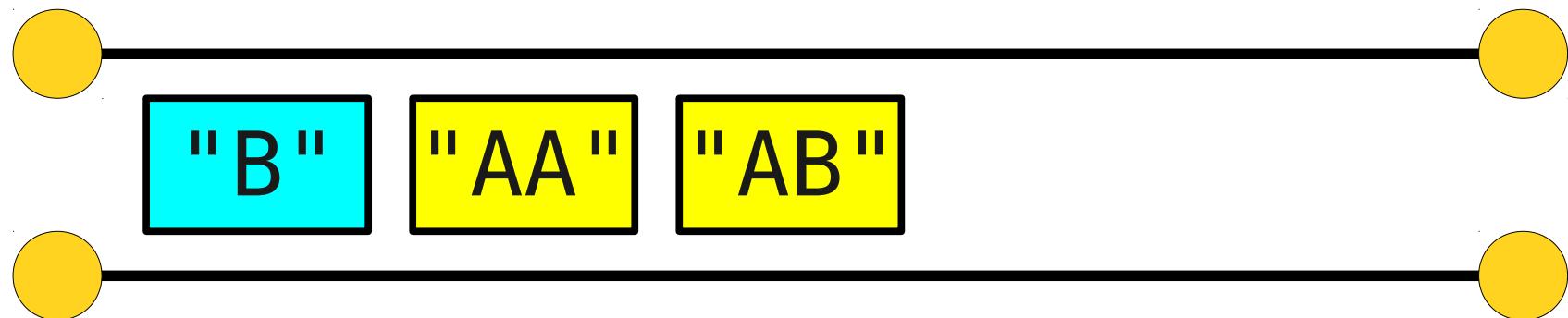
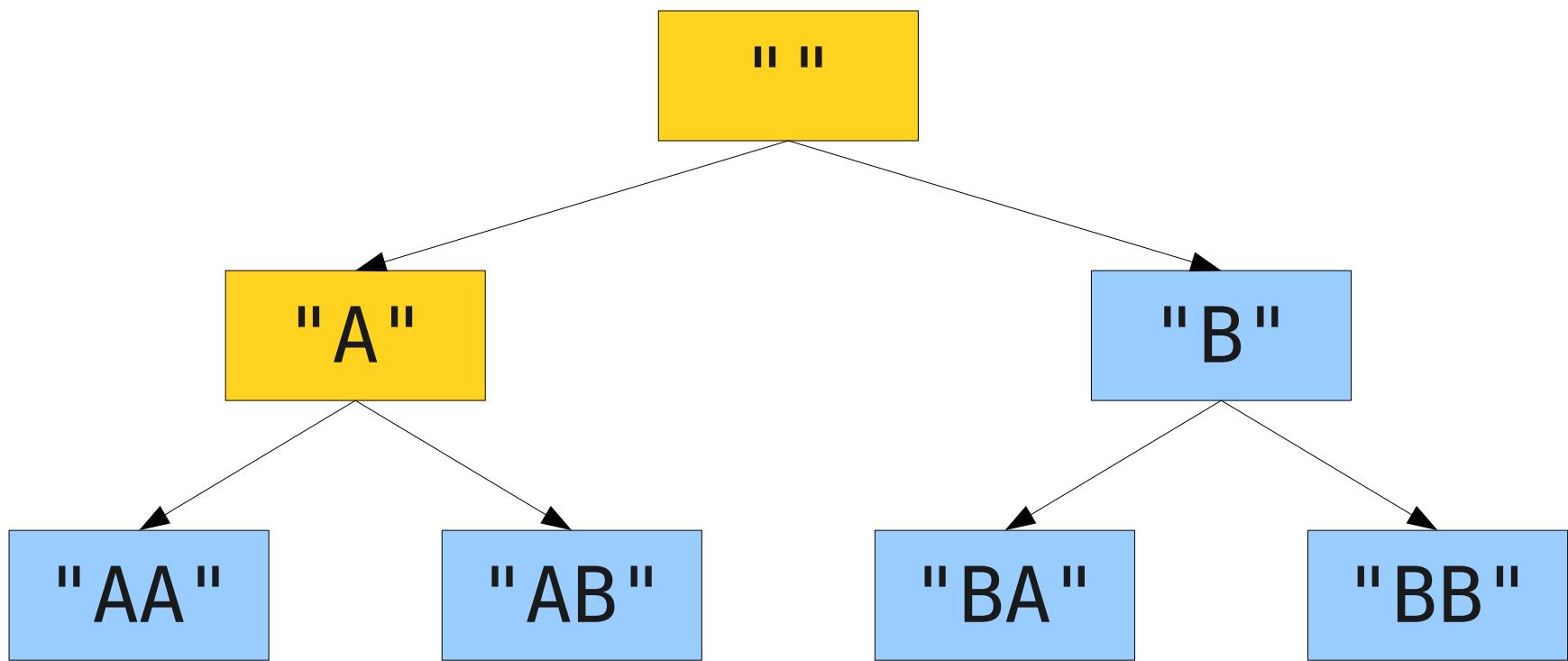


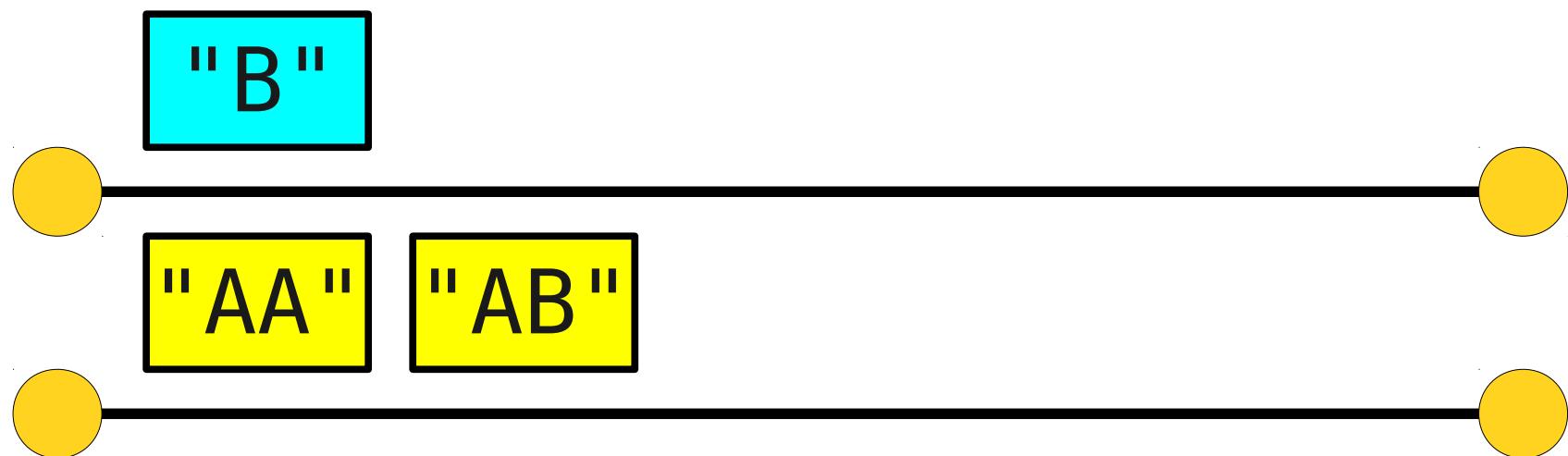
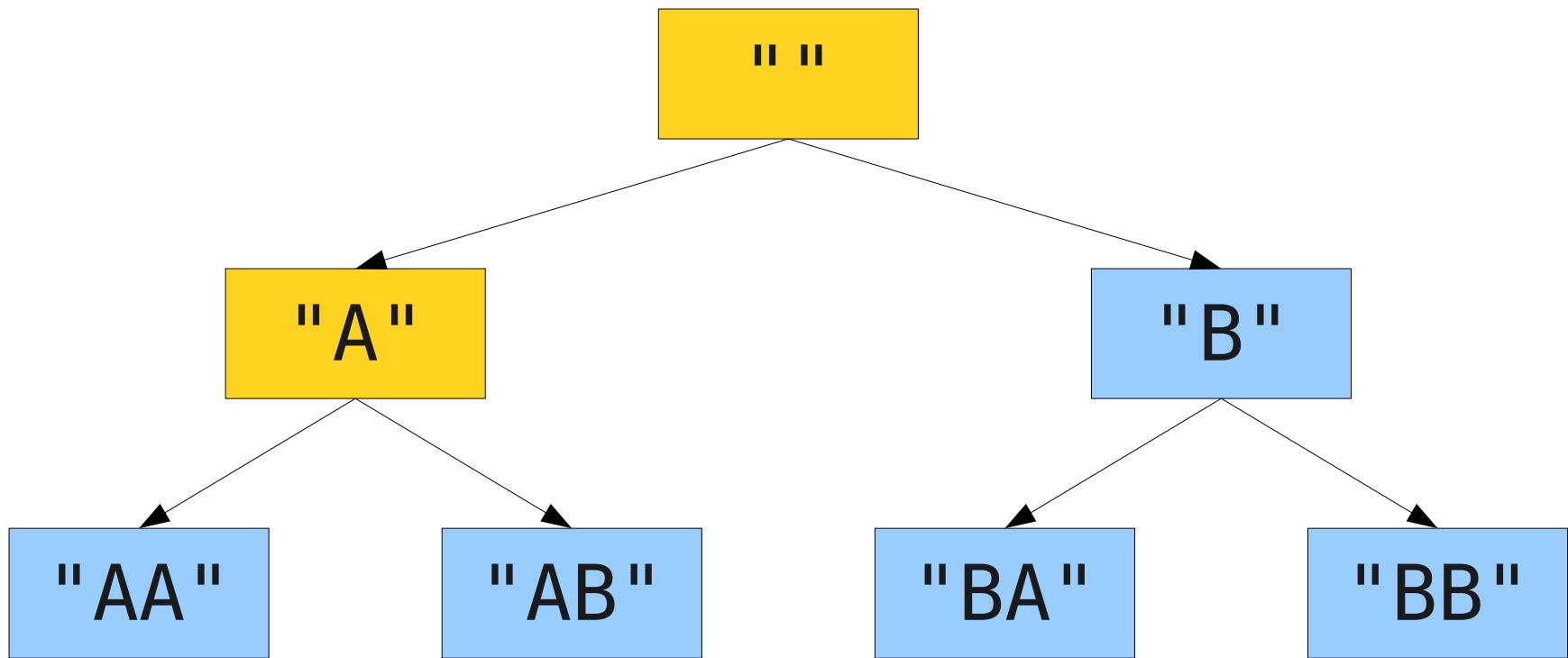


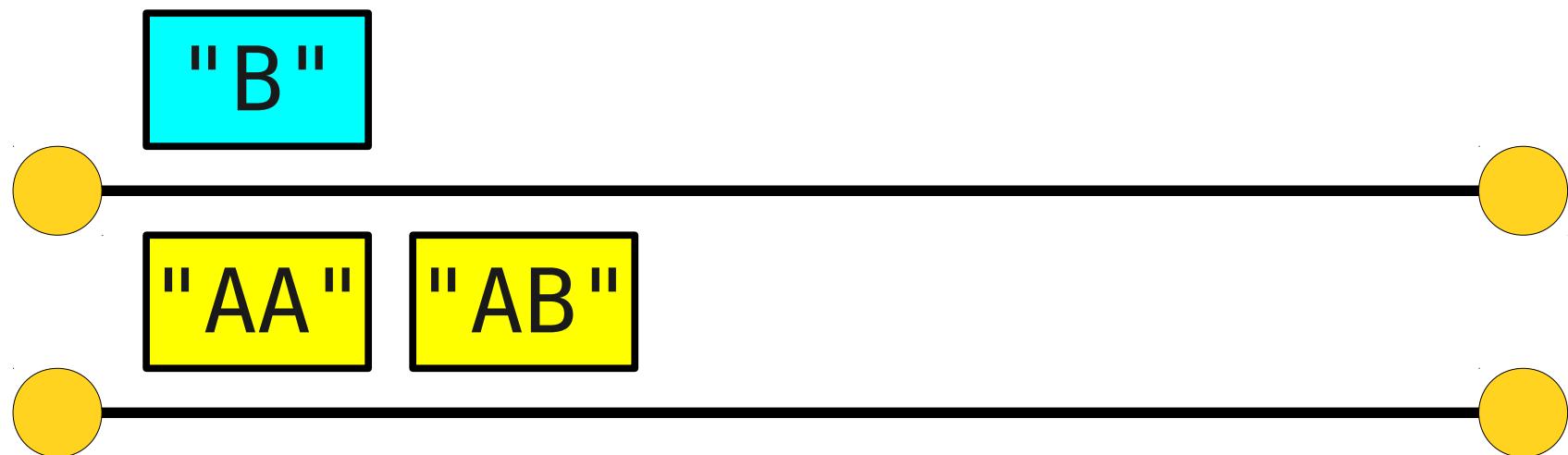
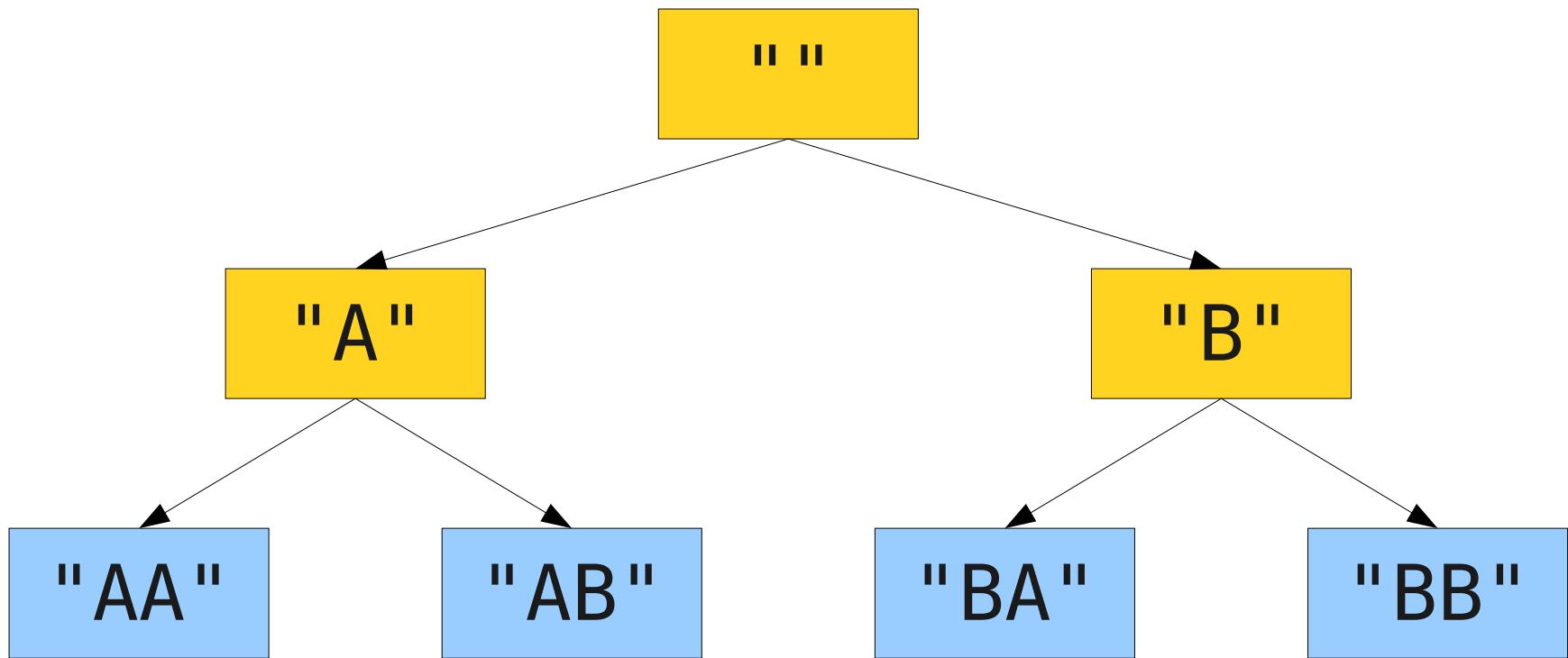


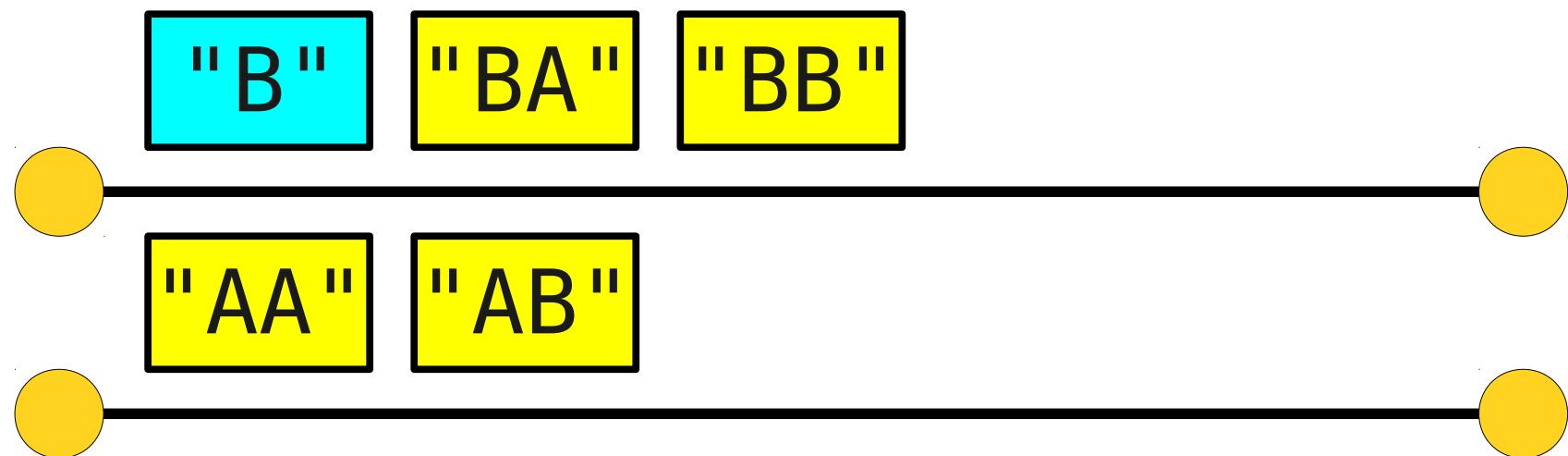
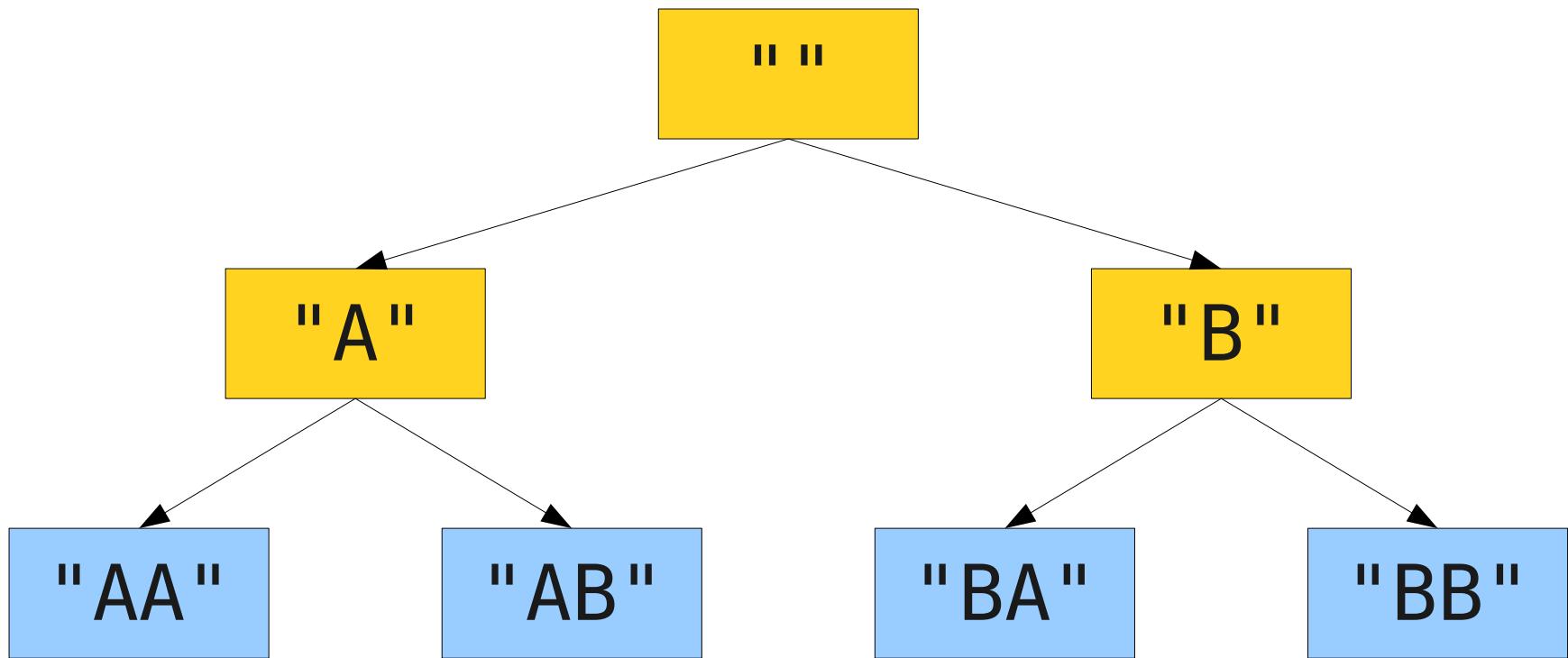


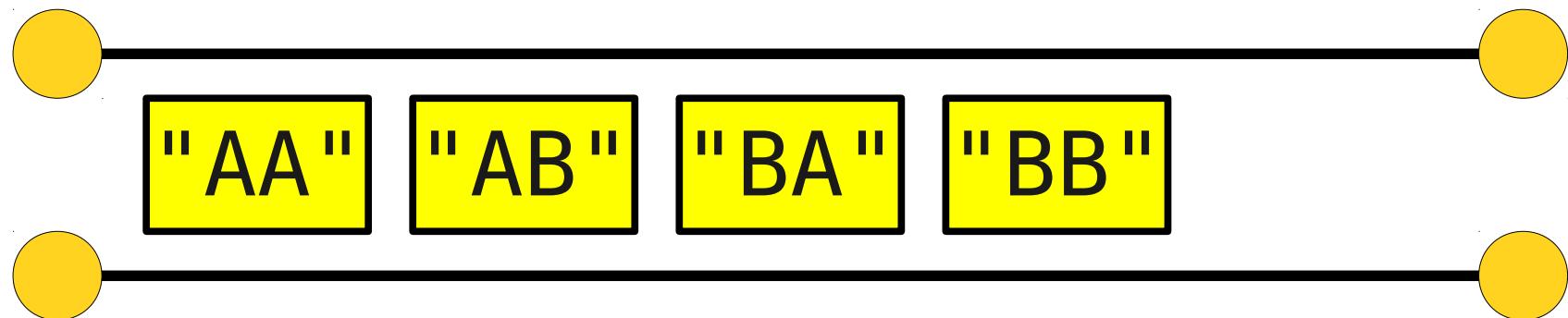
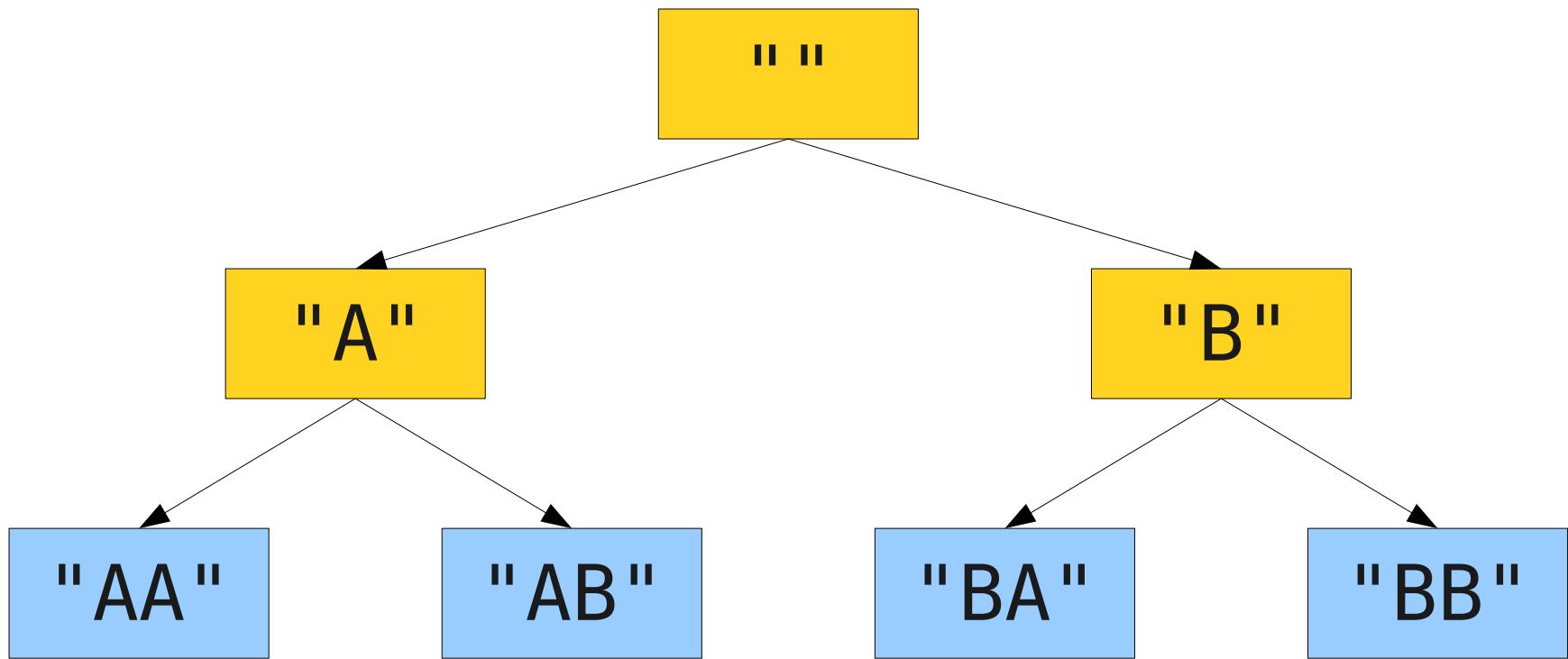


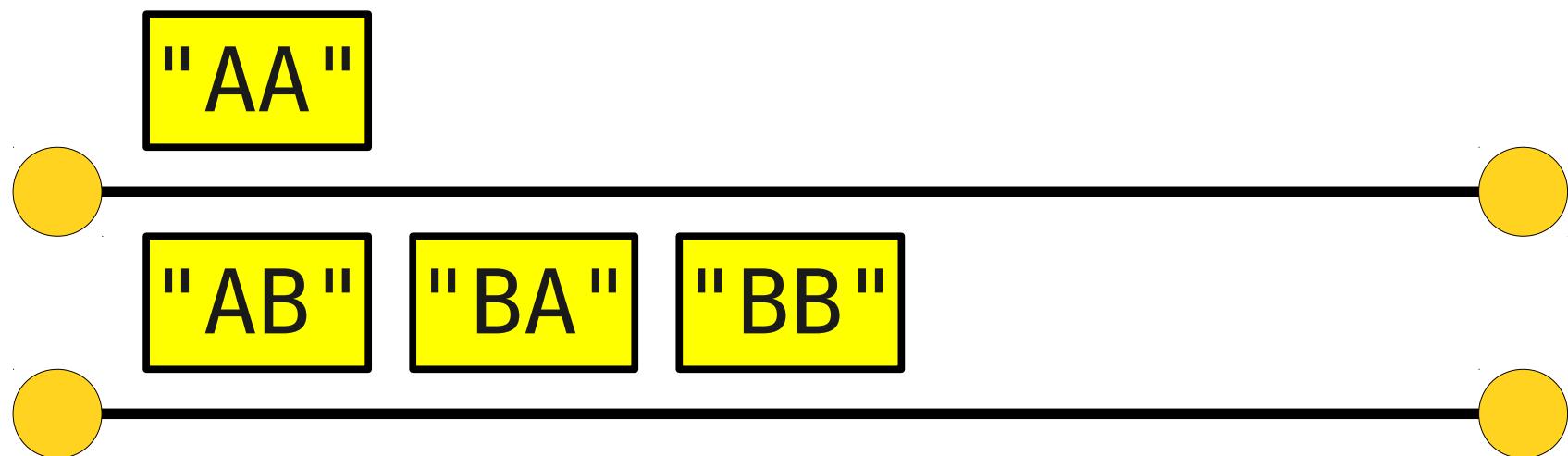
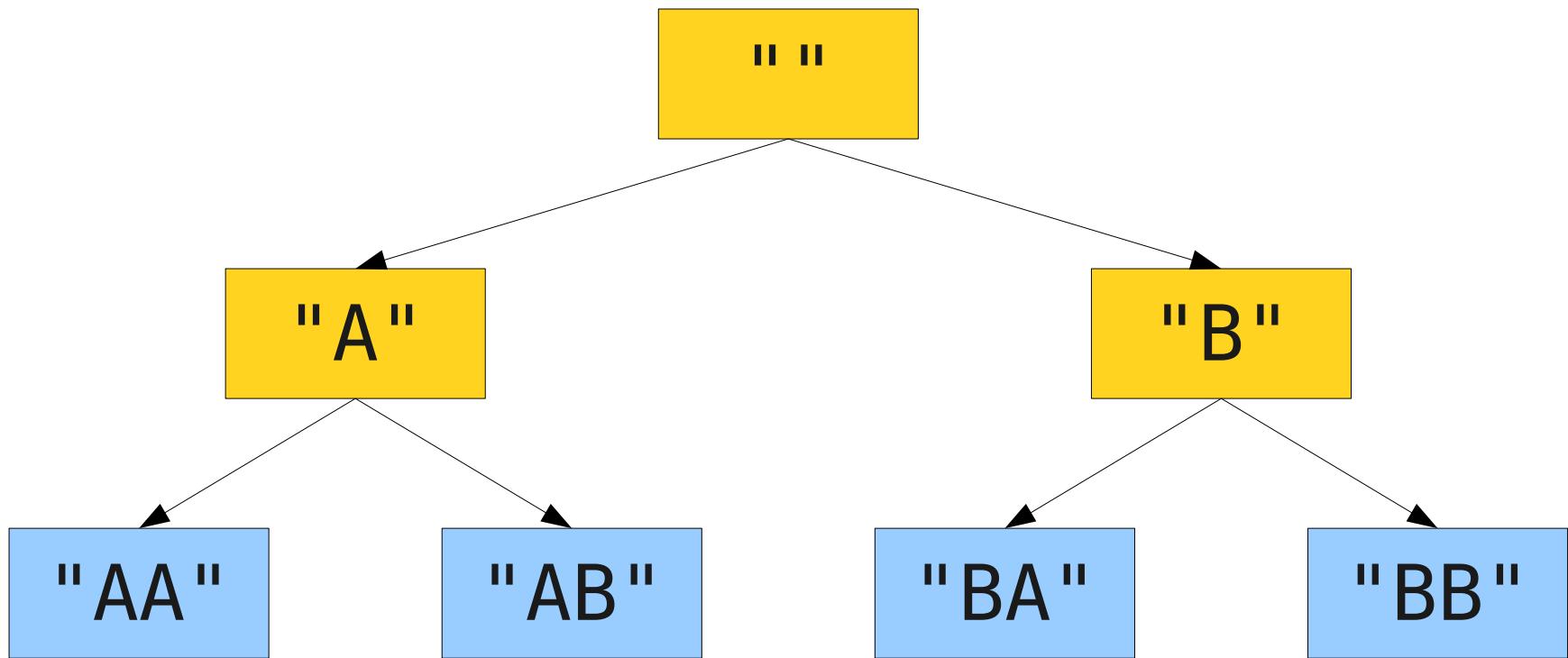


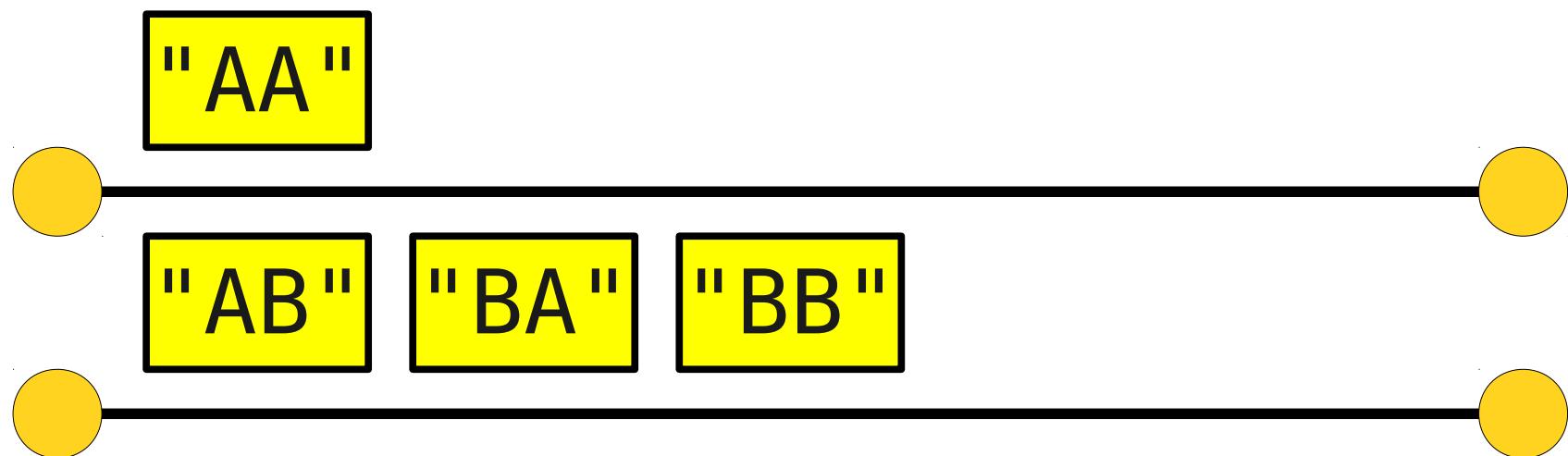
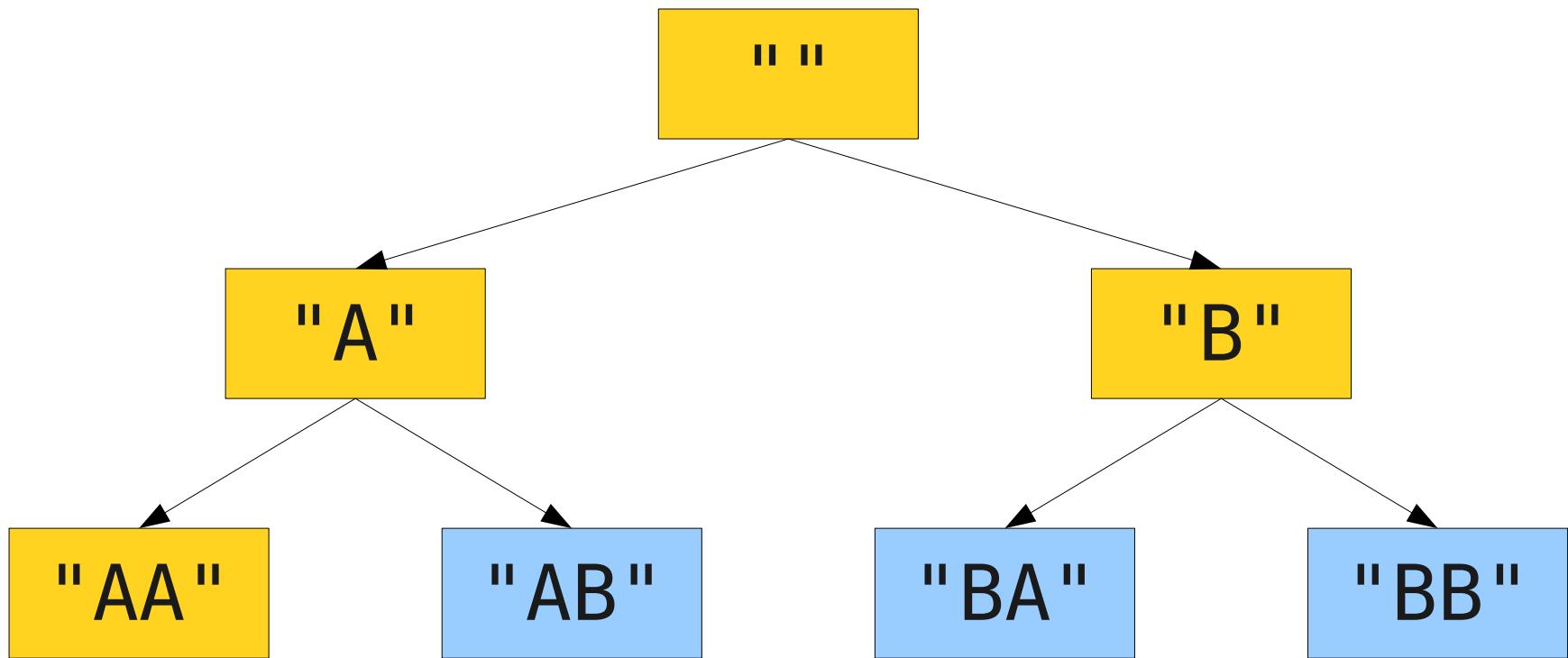


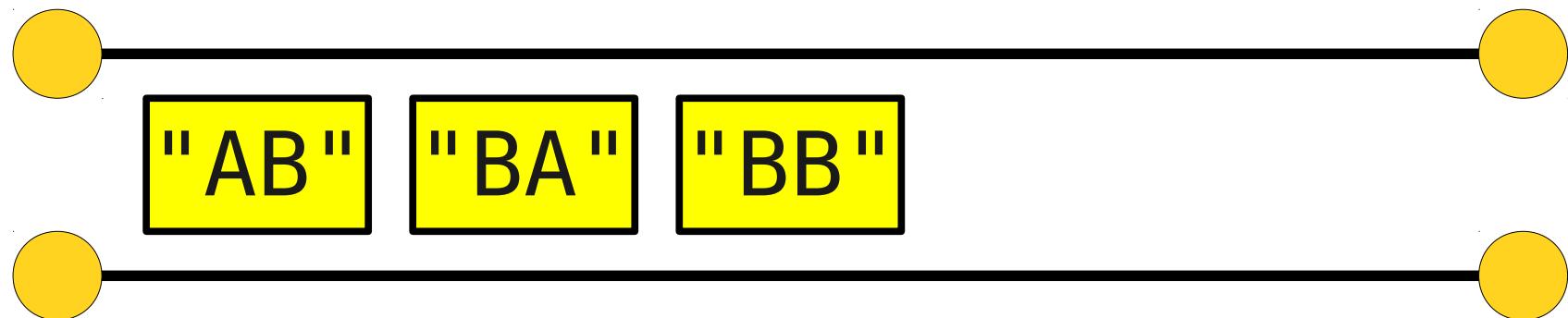
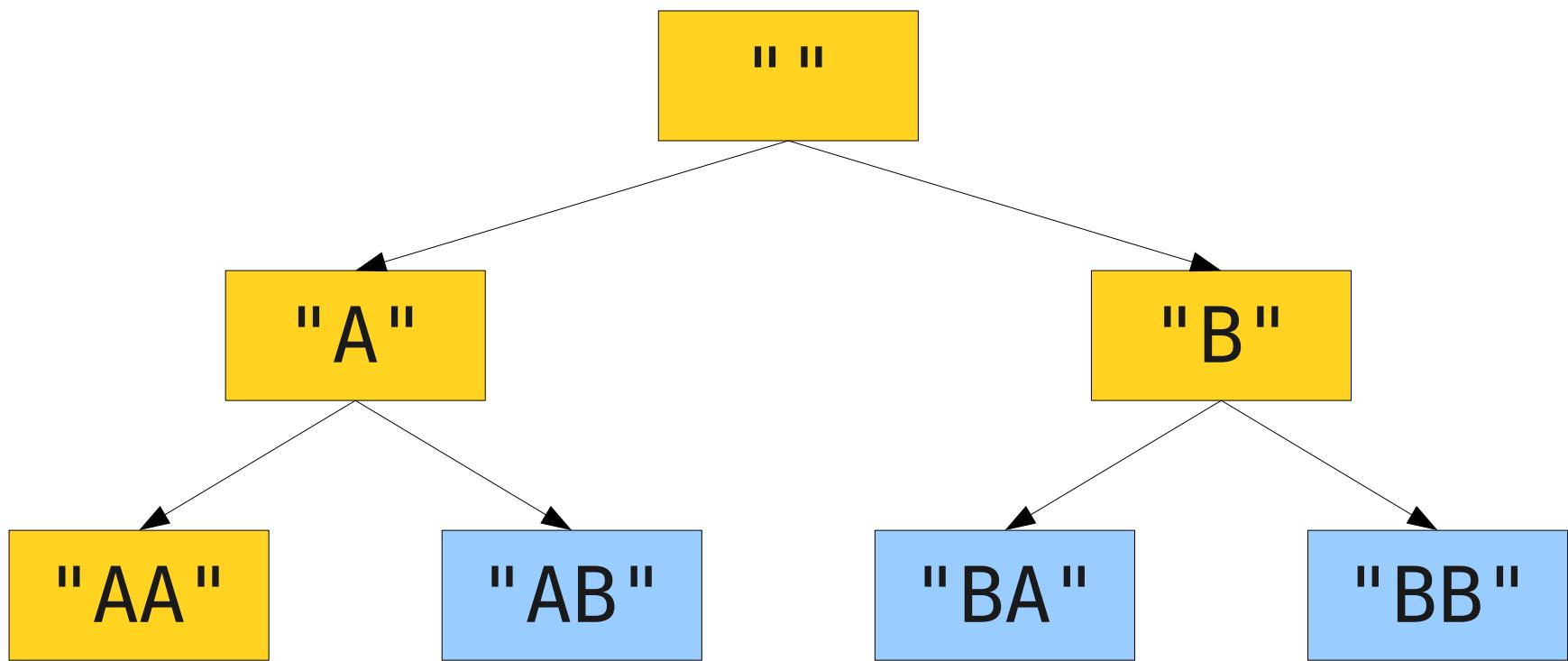


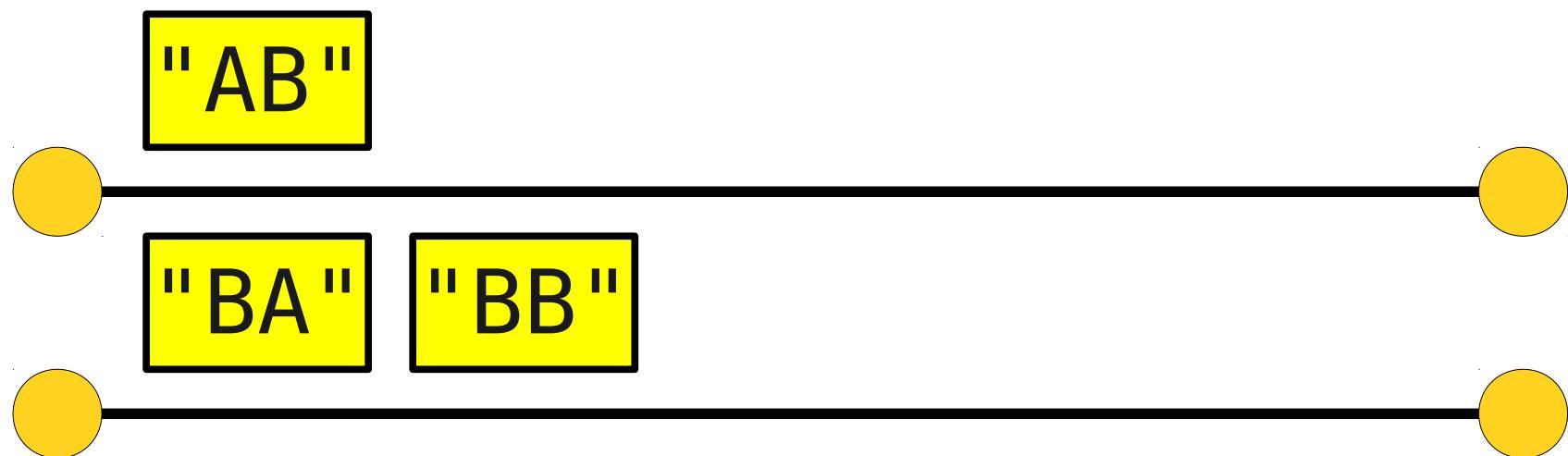
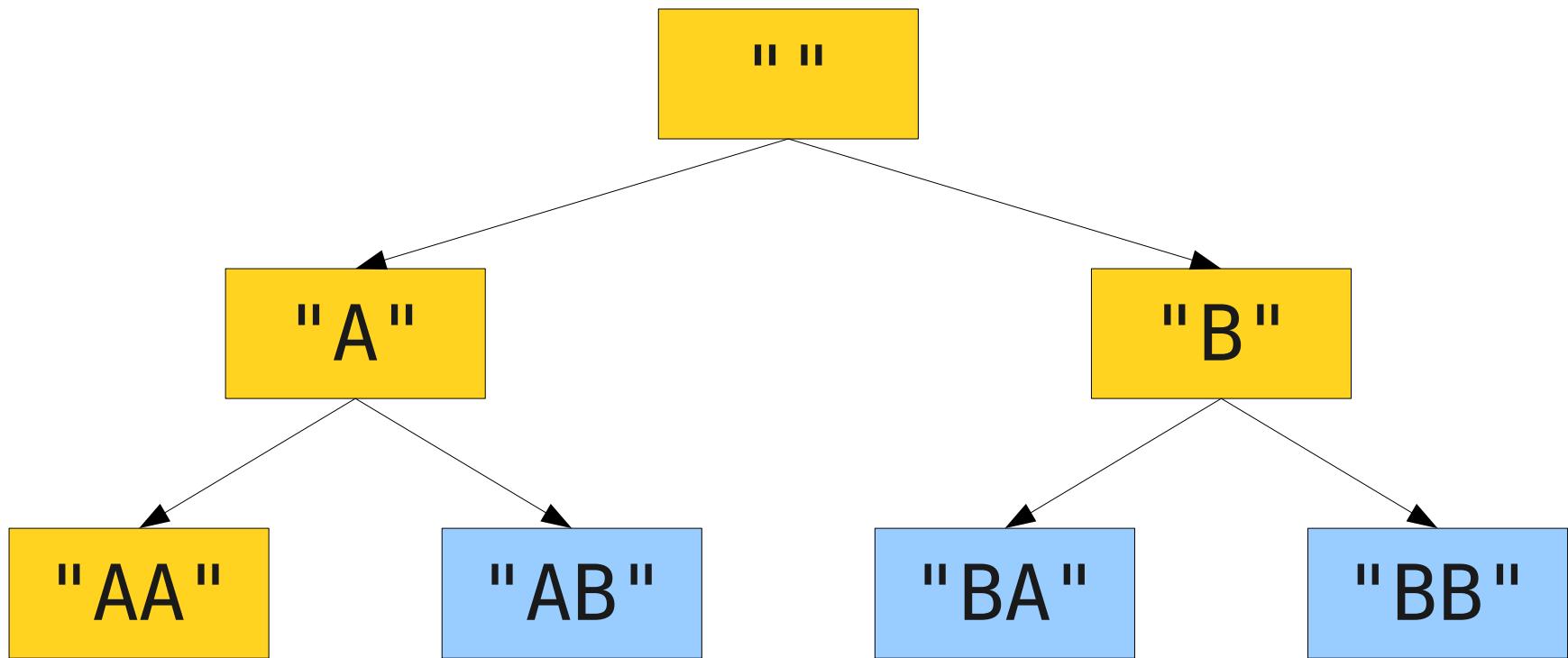


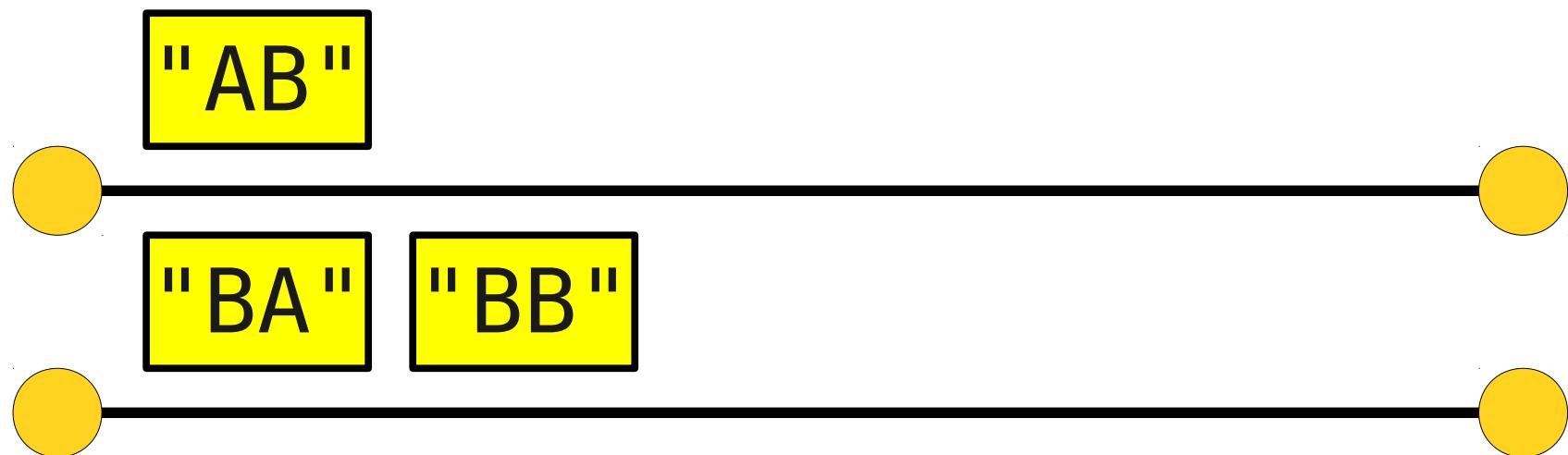
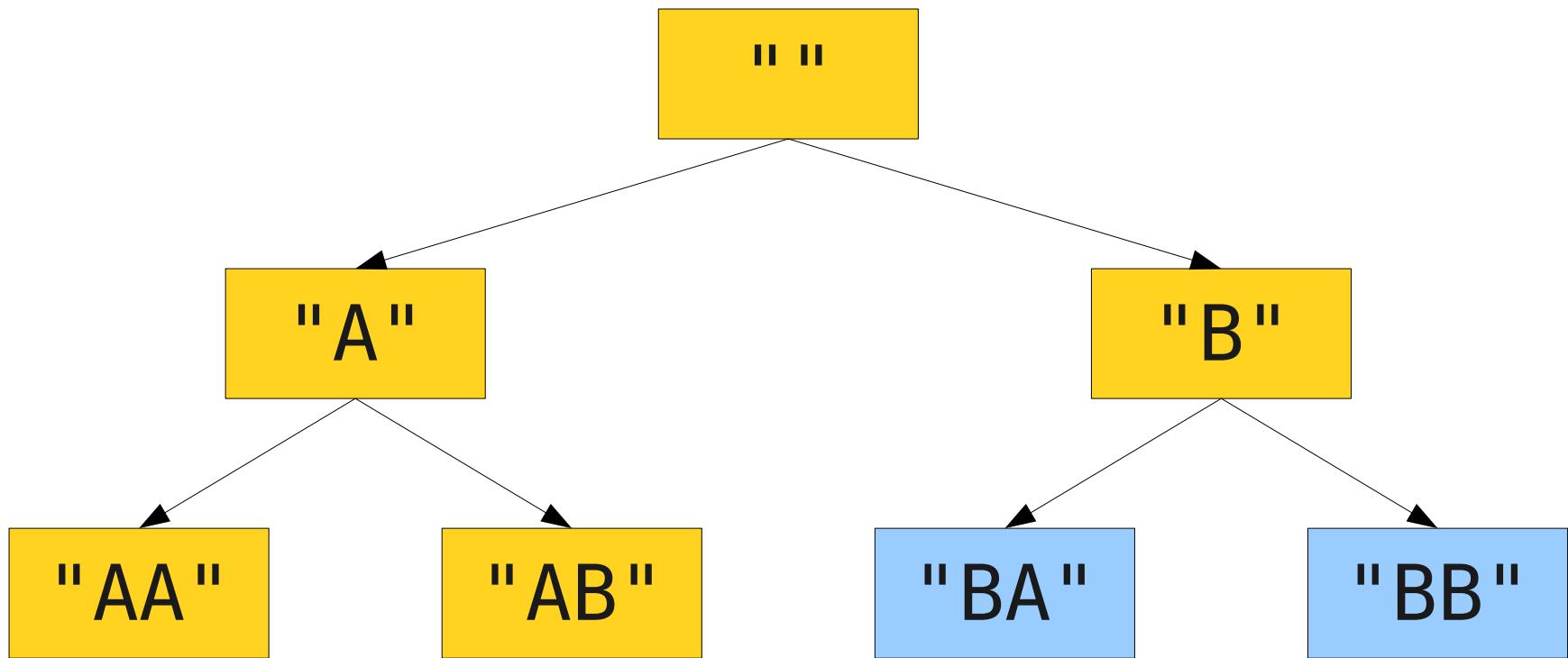


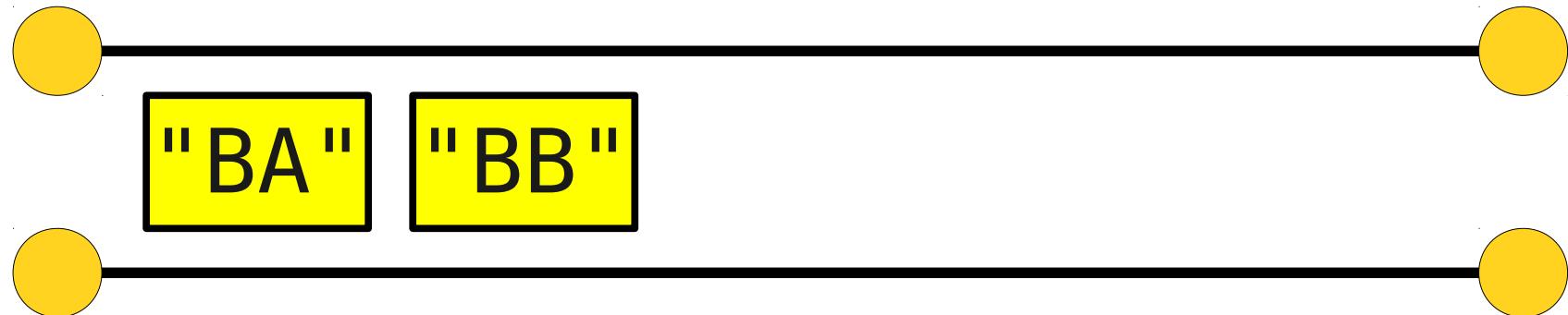
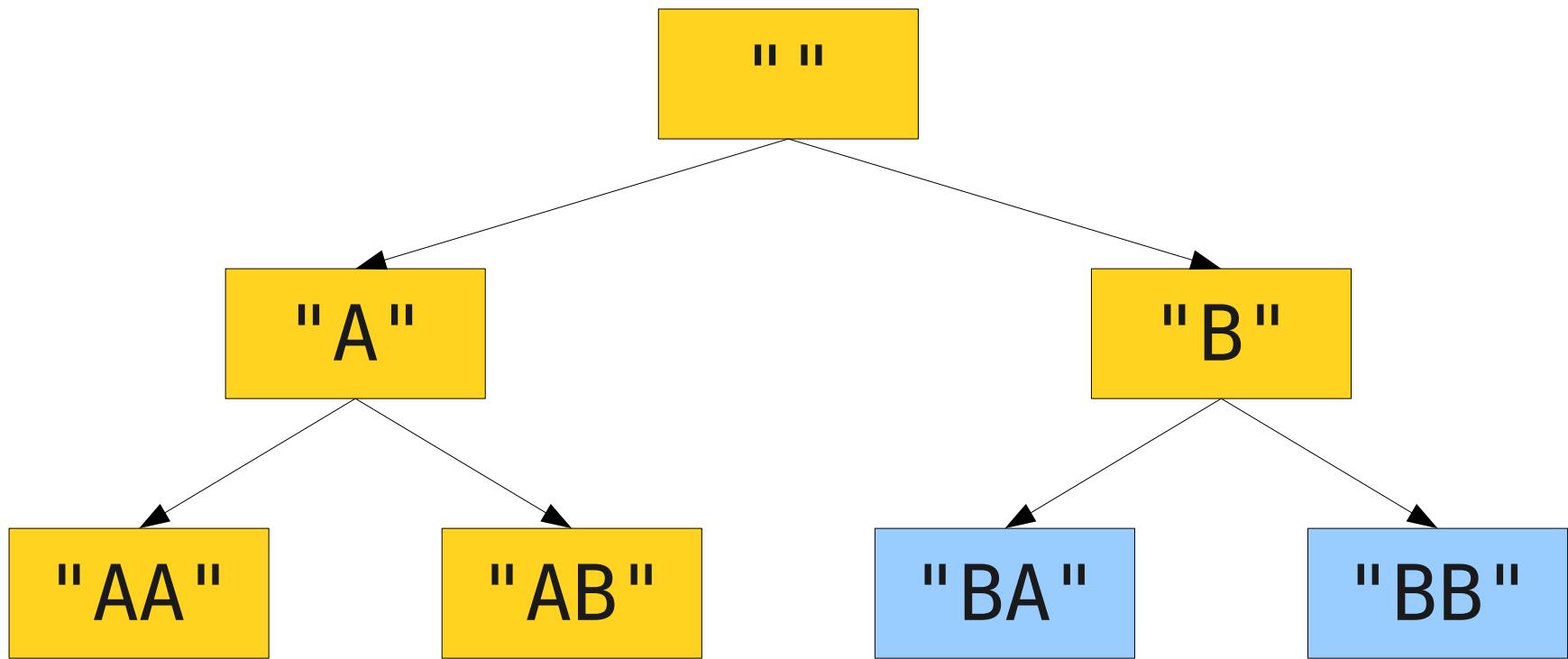


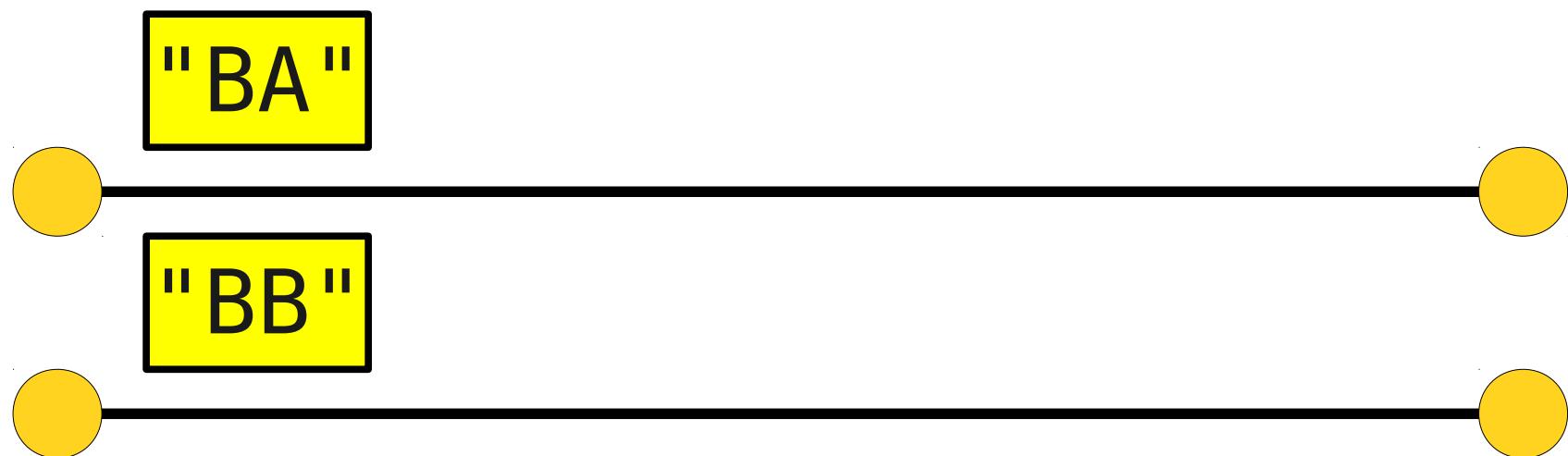
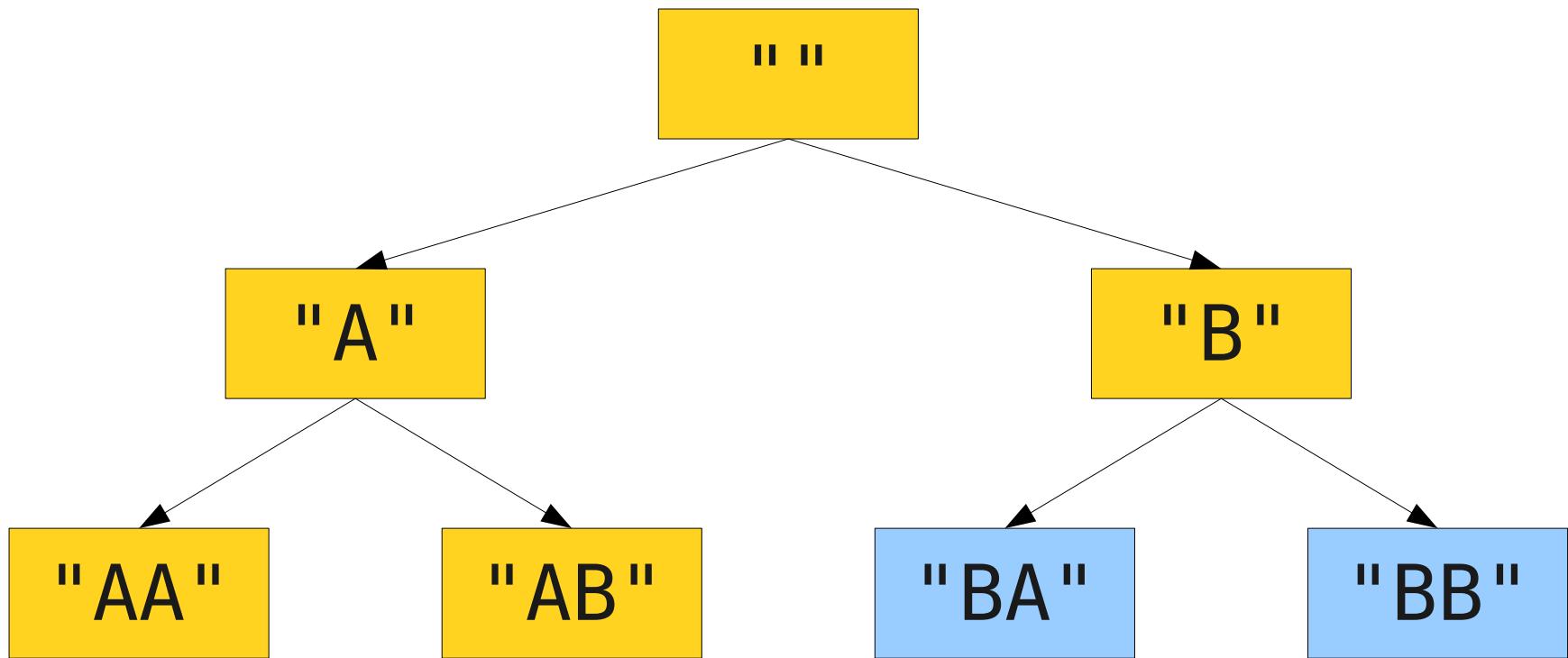


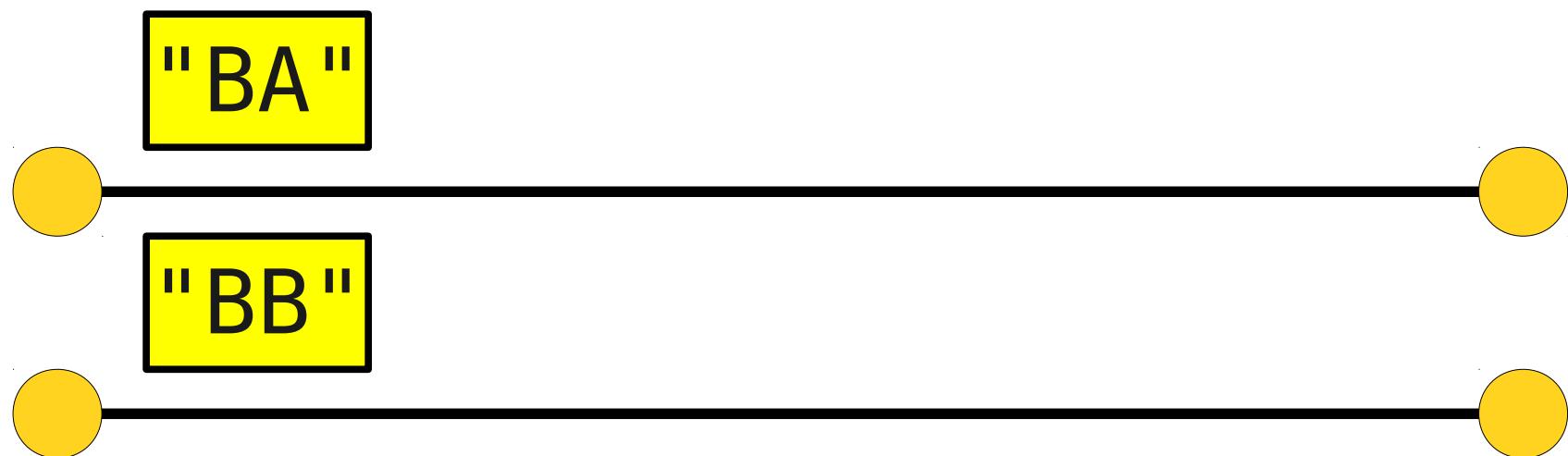
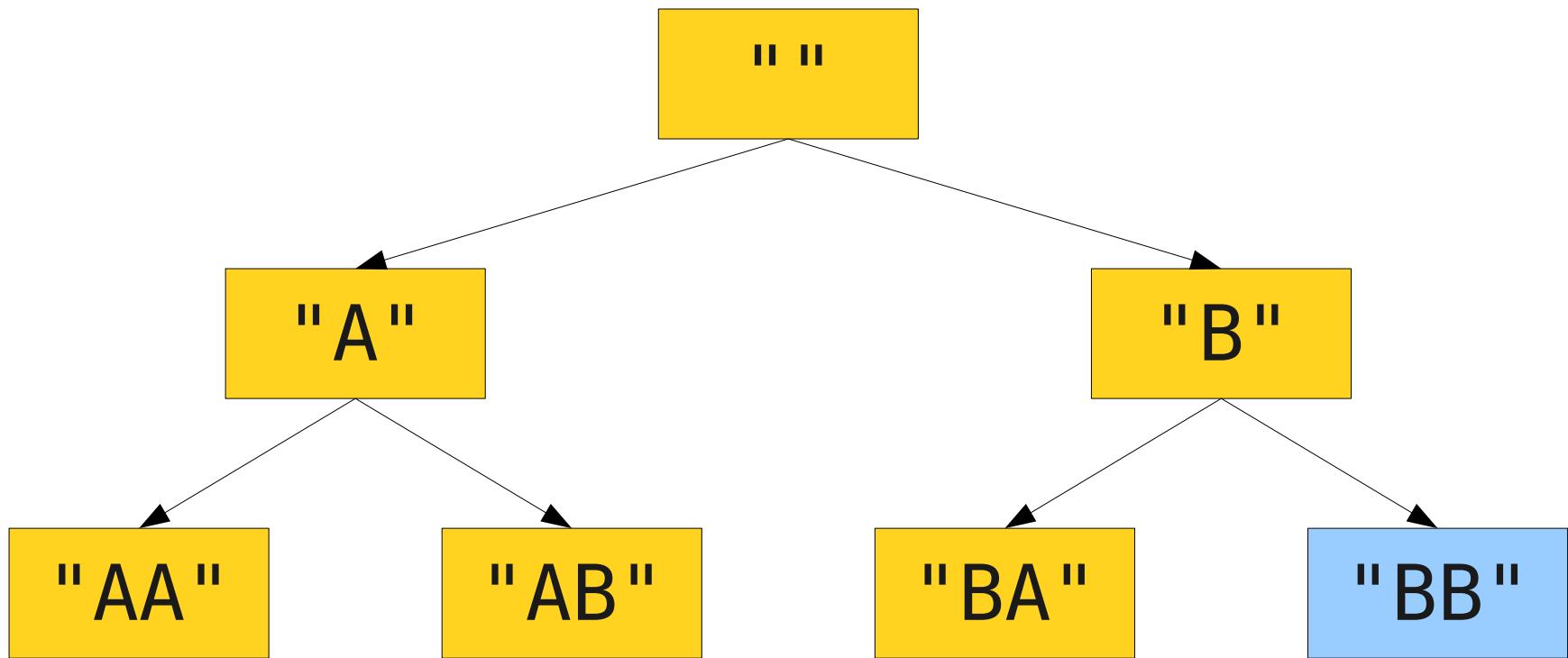


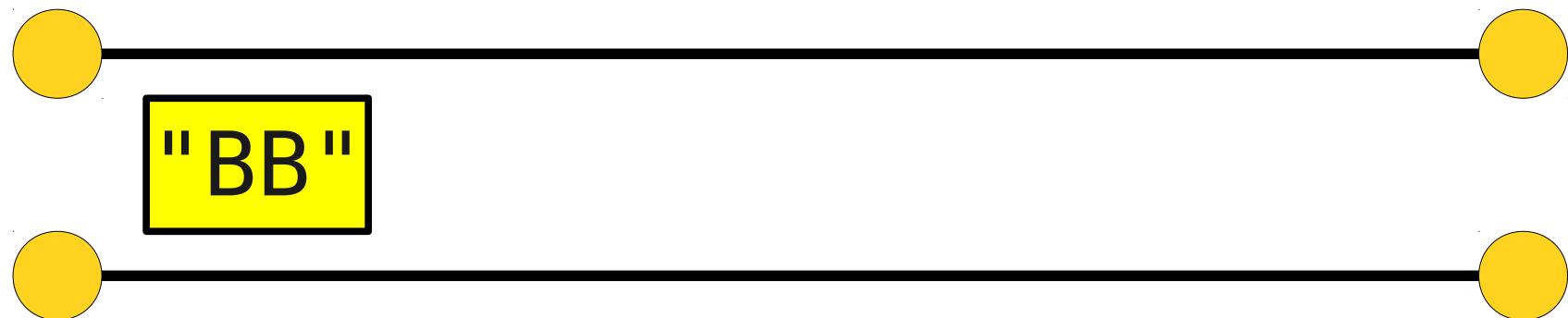
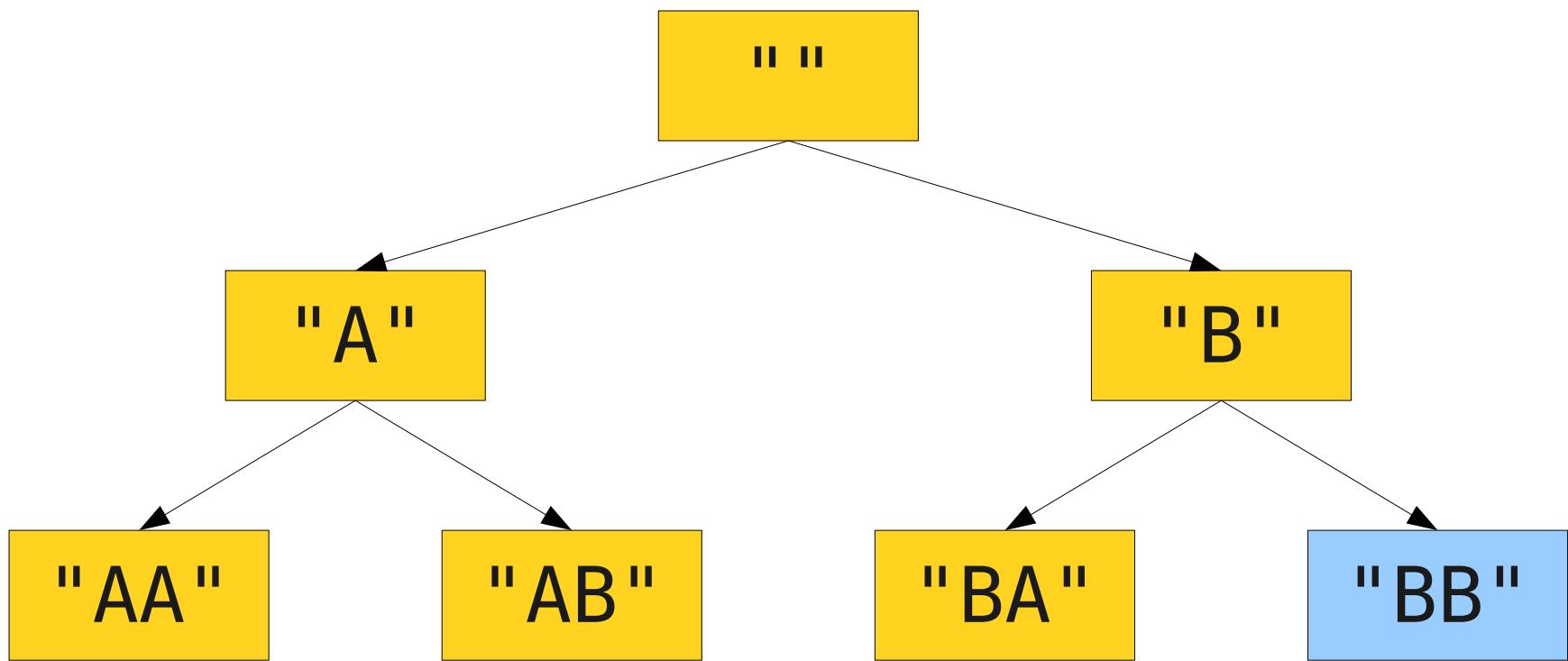


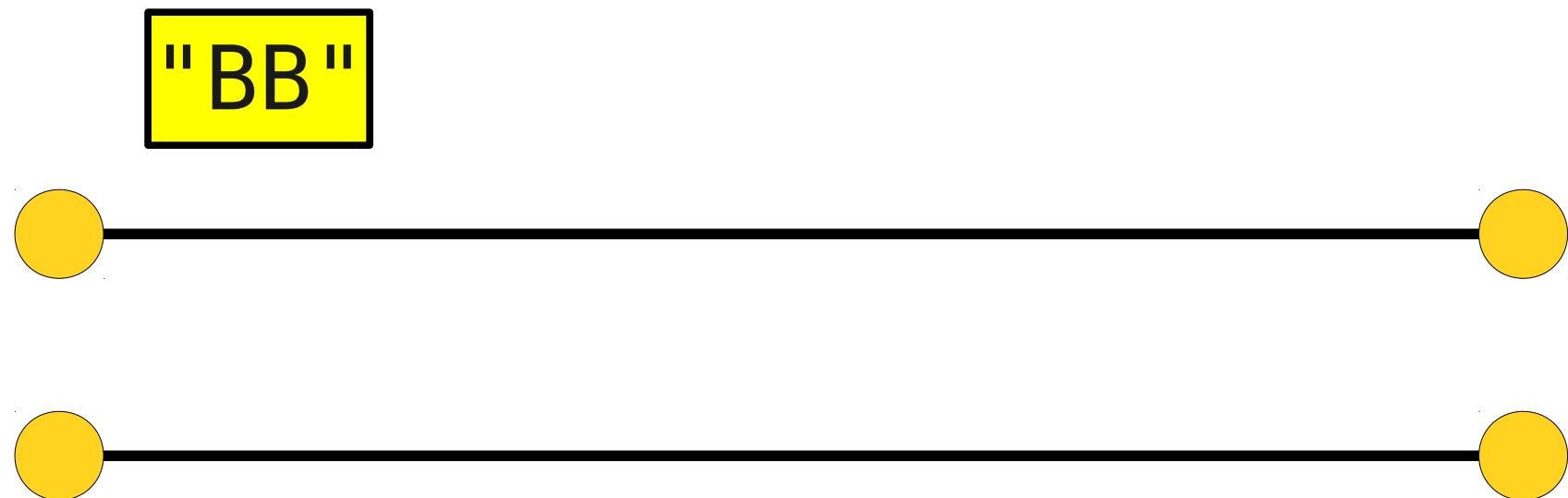
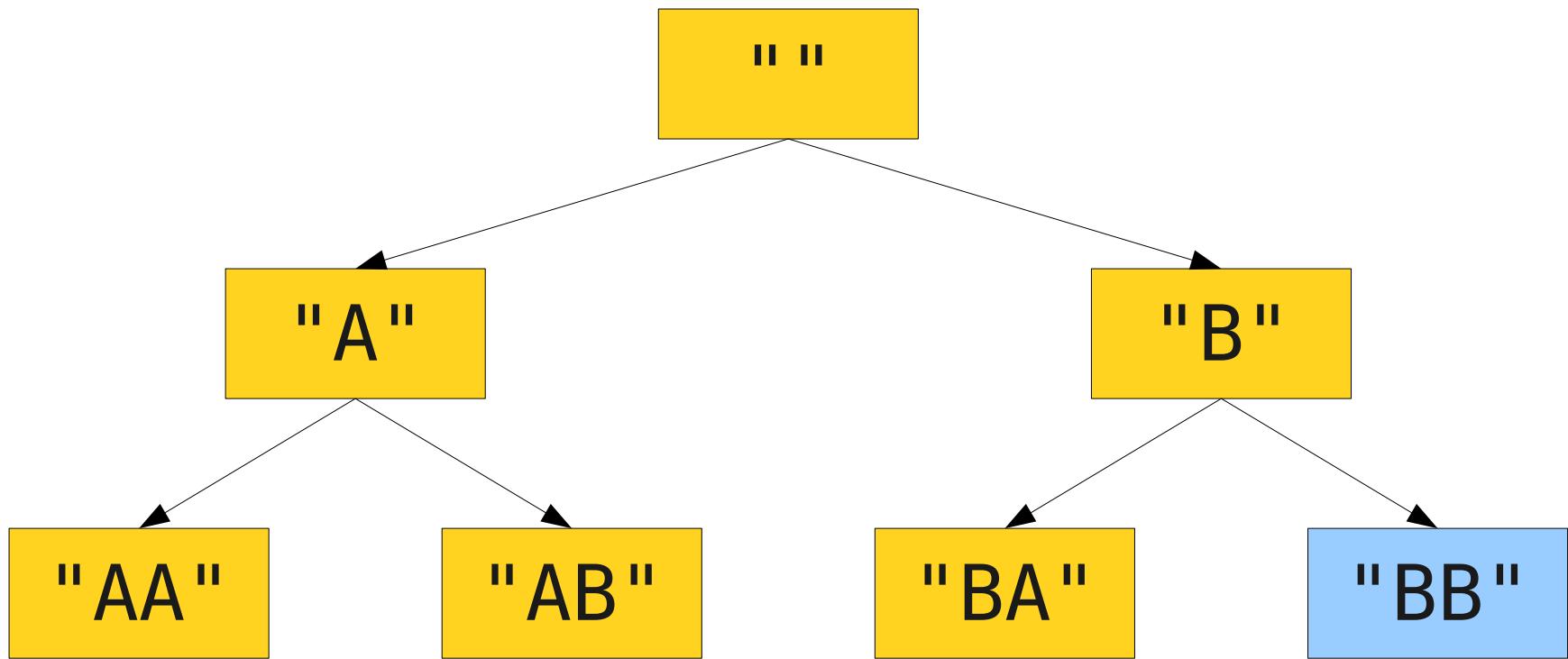


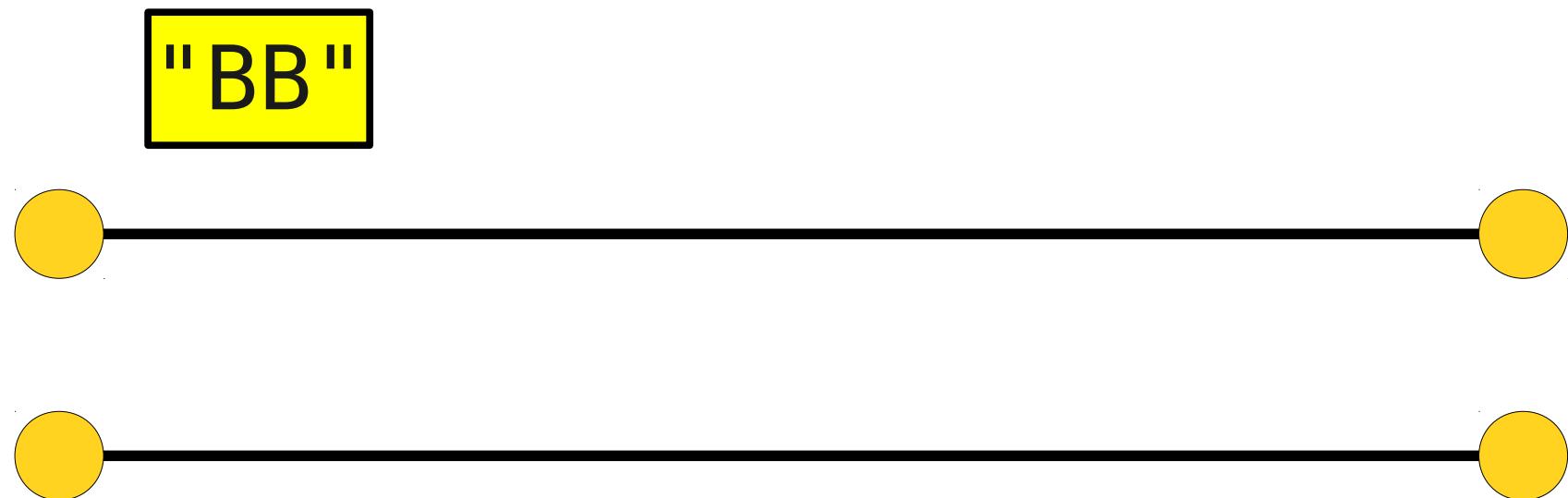
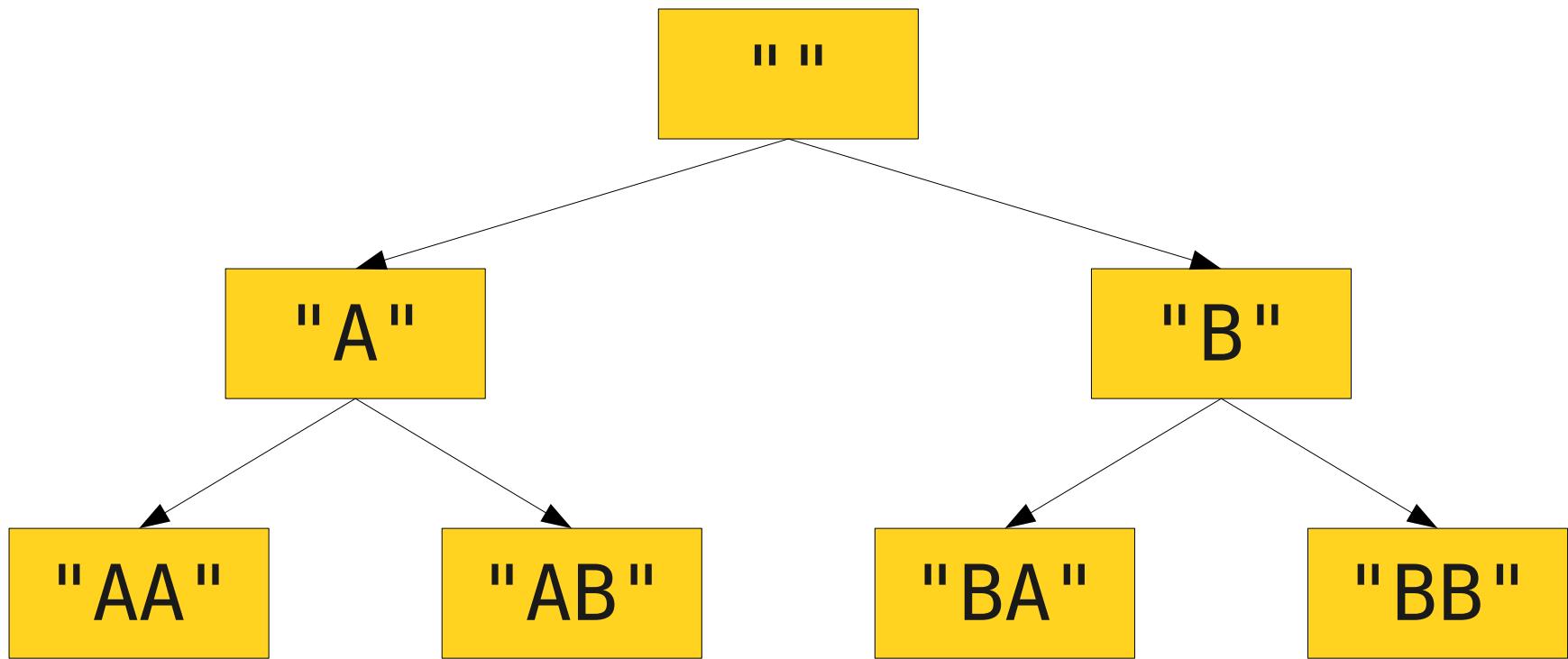


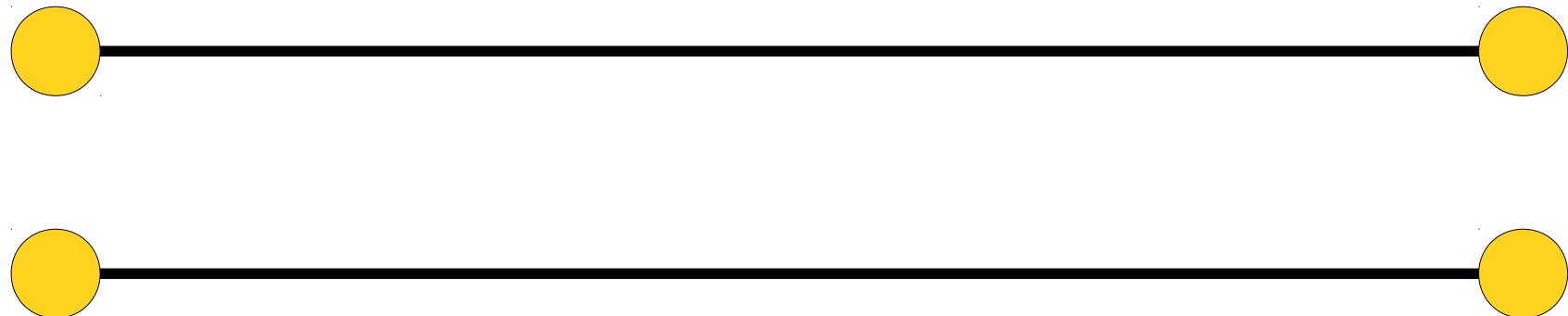
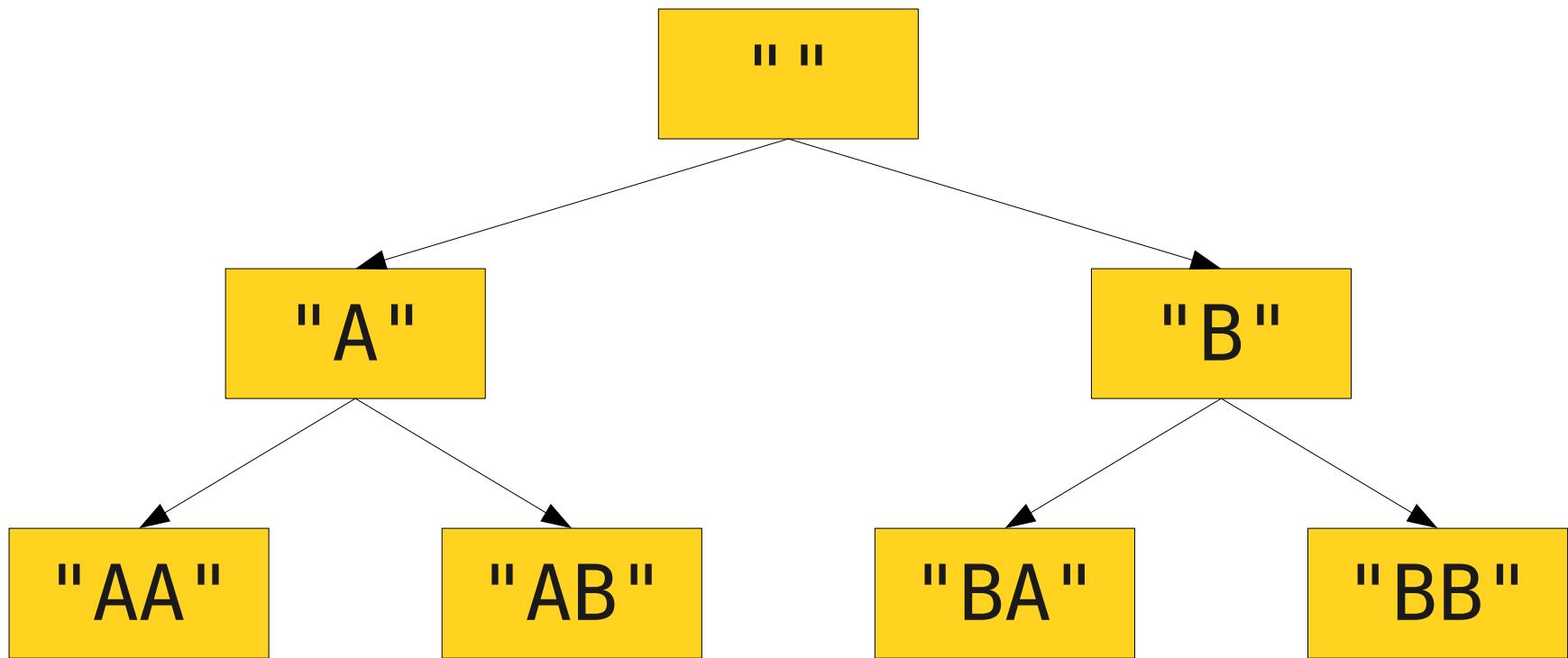












Application: Cracking Passwords

Disclaimer

- I am **not** trying to teach you how to hack into computer systems!
- This lecture (and the start of the next) explores techniques for attacking password systems and for preventing these attacks.
- My intent is to help you better understand computer security and to motivate the techniques from the next section of CS106B.
- Do **not** use these techniques offensively! It is **illegal**!

The Setup

- Suppose that you have a mystery function

```
bool login(string& username,  
          string& password);
```

- This function takes in a username and password, then tells you whether it is correct.
- Suppose that you know someone's username. How might you break in to their account?

A Brute-Force Attack

- **Idea:** Try logging in with all possible passwords!
 - Try logging in with all strings of length 0, then all strings of length 1, then all strings of length 2, ...
 - Eventually, this will indeed crack the password.
 - What would this look like in code?
 - How fast will it be?

Analyzing Efficiency

- How long will it take for our program to crack a password?
- Let's make the following assumptions:
 - The password consists purely of lowercase letters.
 - The password has length at most n .
- There are 26^k possible lowercase strings of length k .
- Might have to try all strings of length $0, 1, \dots, n$.
- To break the password, we need to try at most
$$1 + 26 + 26^2 + 26^3 + 26^4 + \dots + 26^n = \frac{26^{n+1} - 1}{25}$$
different strings.

How Many Strings?

- If $n = 0$, might have to try 1 string.
- If $n = 1$, might have to try 27 strings.
- If $n = 2$, might have to try 703 strings.
- If $n = 3$, might have to try 18,279 strings.
- ...
- If $n = 8$, might have to try 217,180,147,159 strings.
- ...
- If $n = 15$, might have to try
1,744,349,715,977,154,962,391 strings.

Dictionary Attacks

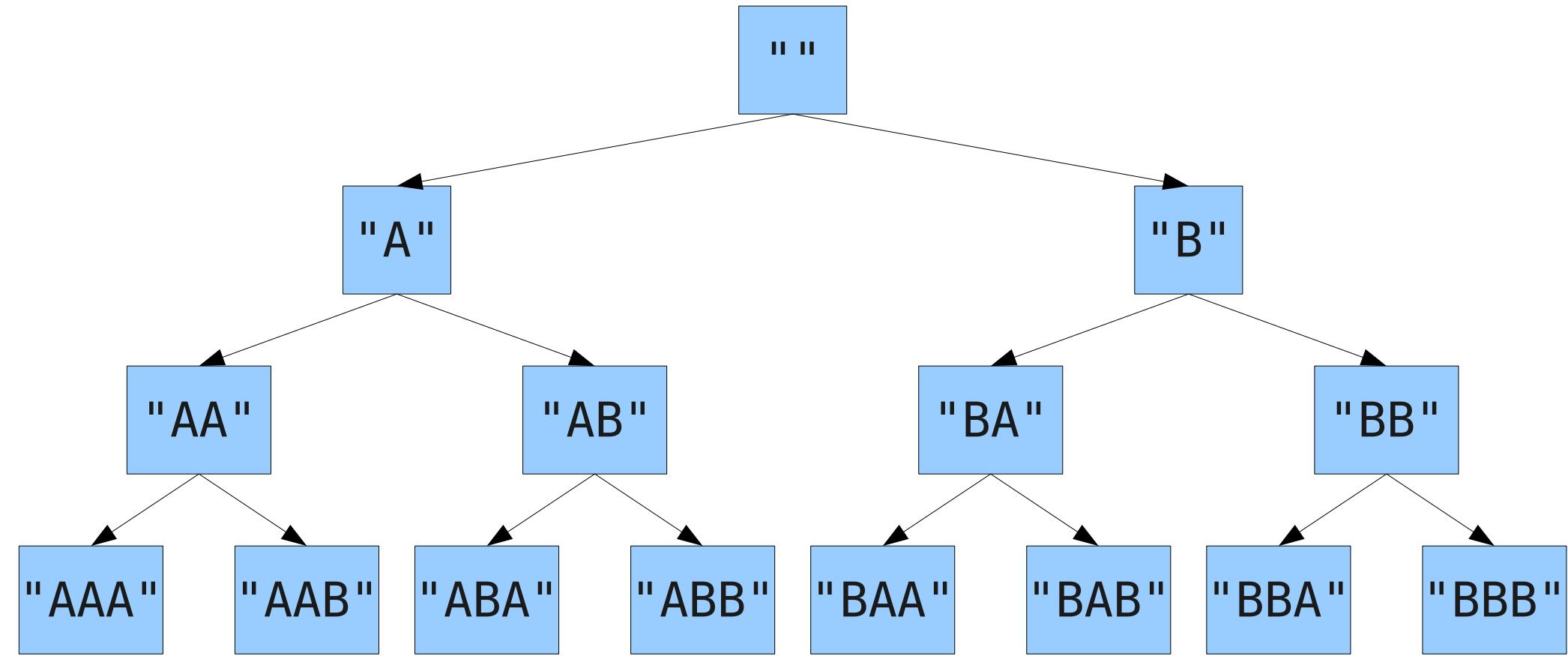
- Passwords are not random strings; people often use
 - words from the dictionary,
 - simple combinations of numbers,
 - their birthdays,
 - etc.
- A **dictionary attack** is an attack on a password system in which the attacker runs through a list of likely password candidates to try to break in.
- Watch how easy this is...

Preventing Attacks

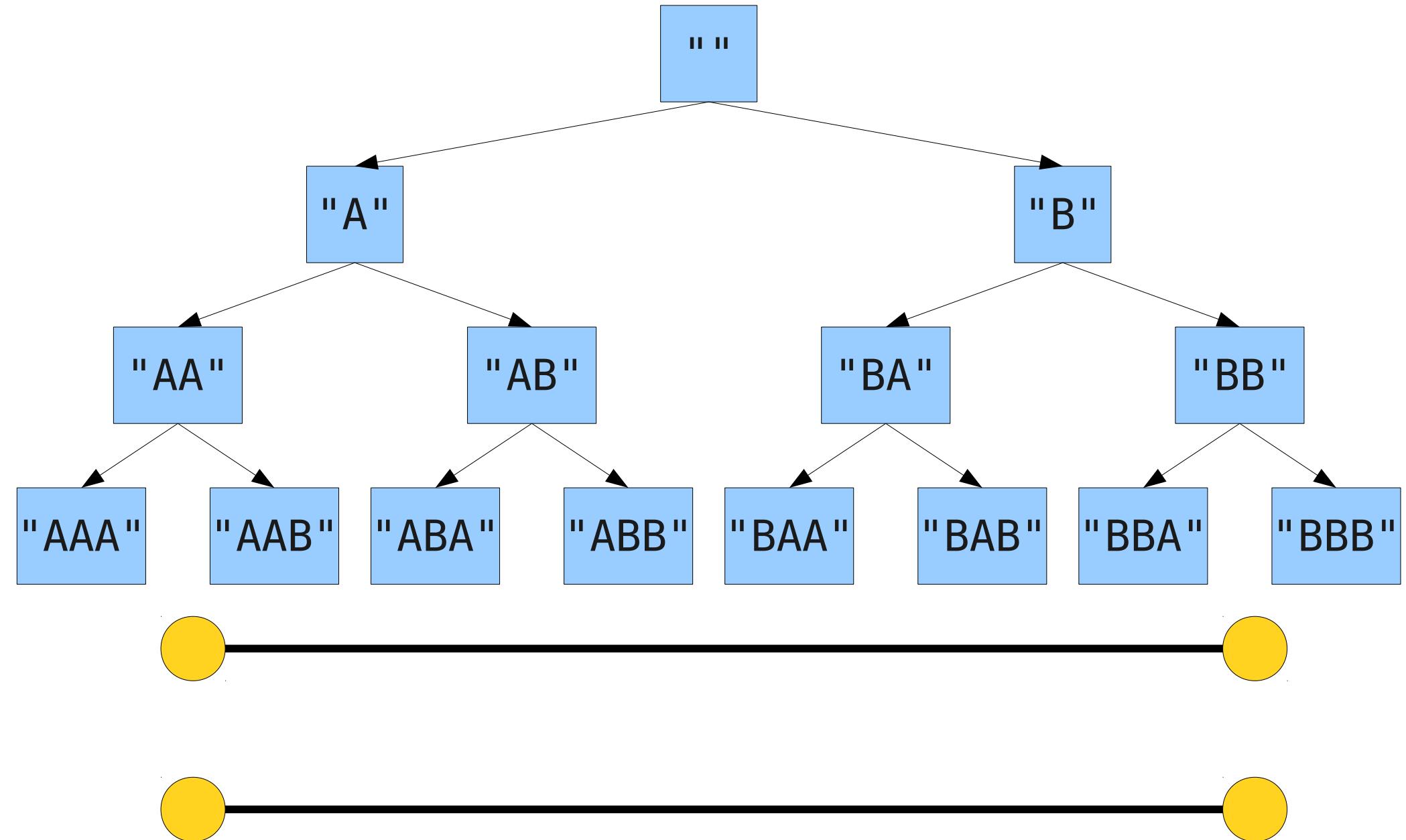
- To make it harder to crack a password, it's generally a good idea to
 - Pick a password that isn't known to be a very common password (to prevent dictionary attacks), and
 - Use a combination of lower case letters, UPPER CASE LETTERS, and numb3r5 (to increase the number of passwords that must be tried.)
- You've now seen exactly why this is!

Why Brute-Forcing is Hard

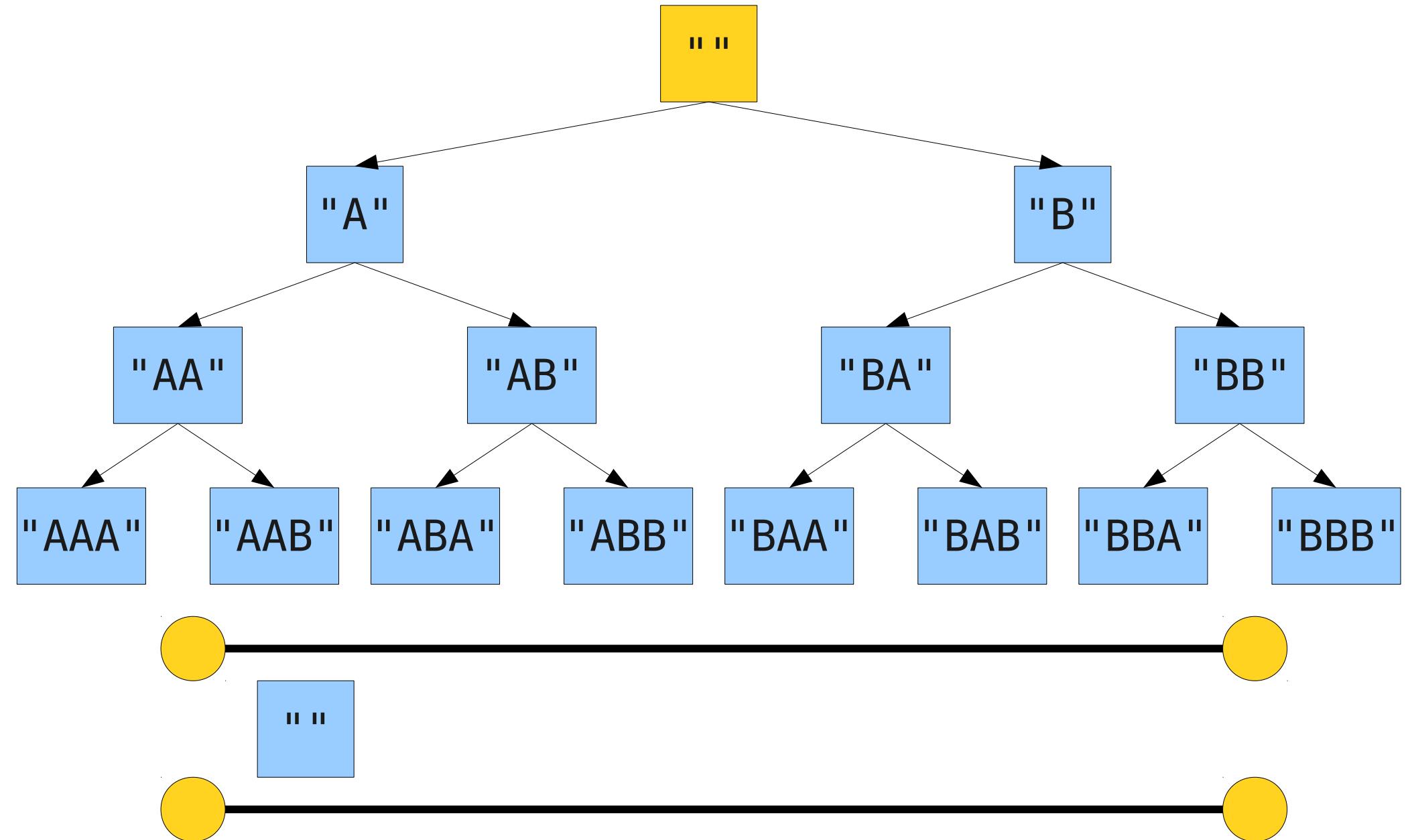
Memory Usage



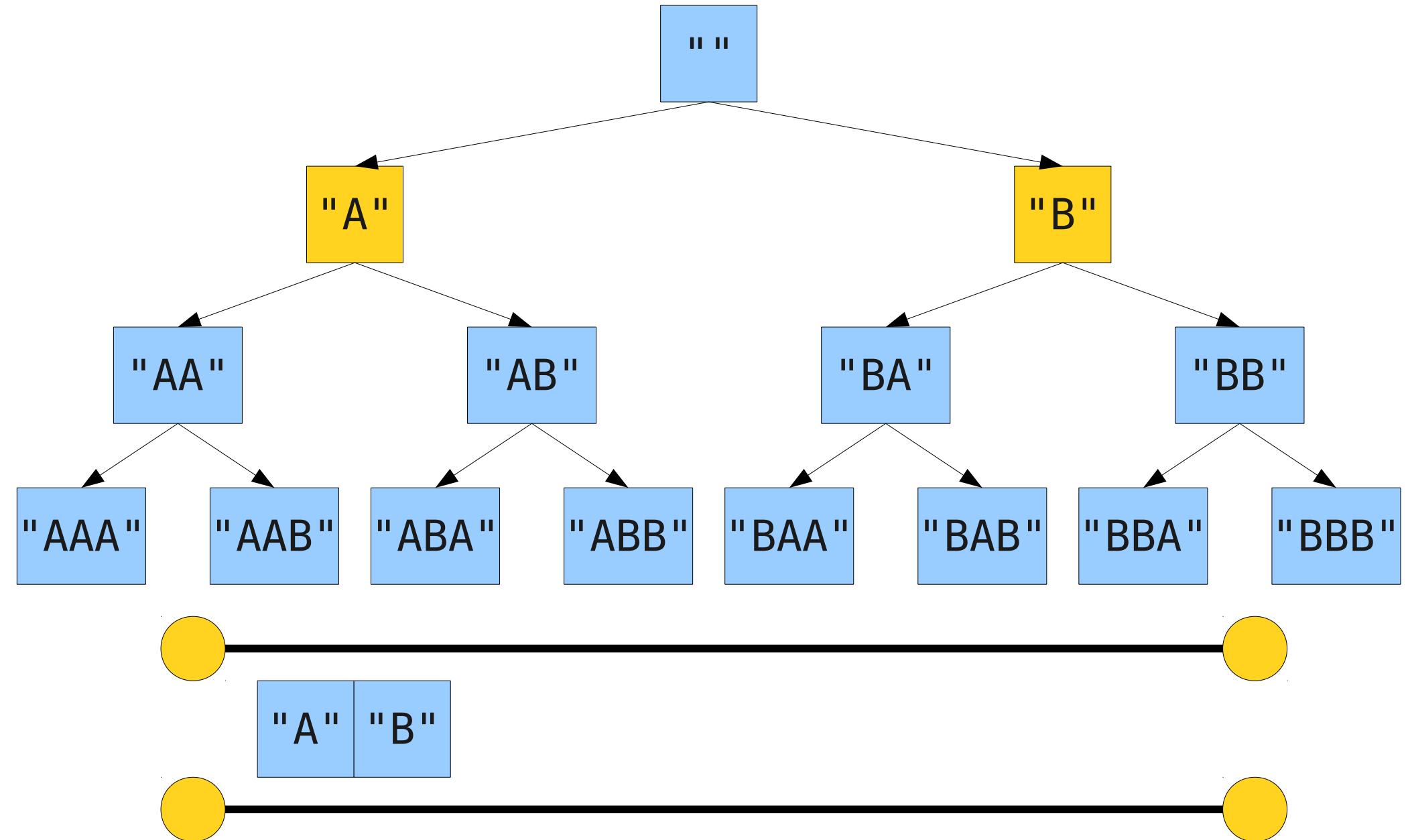
Memory Usage



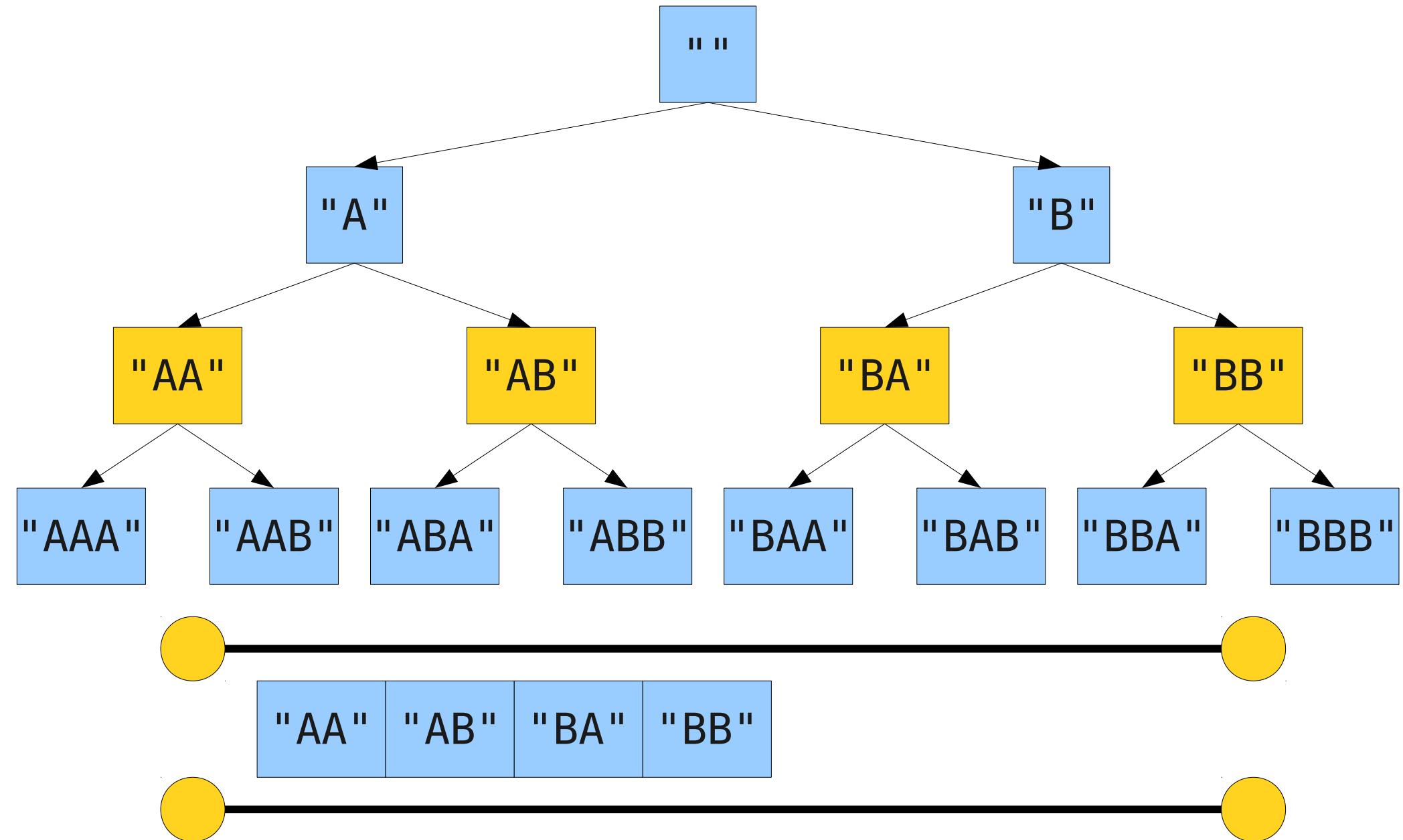
Memory Usage



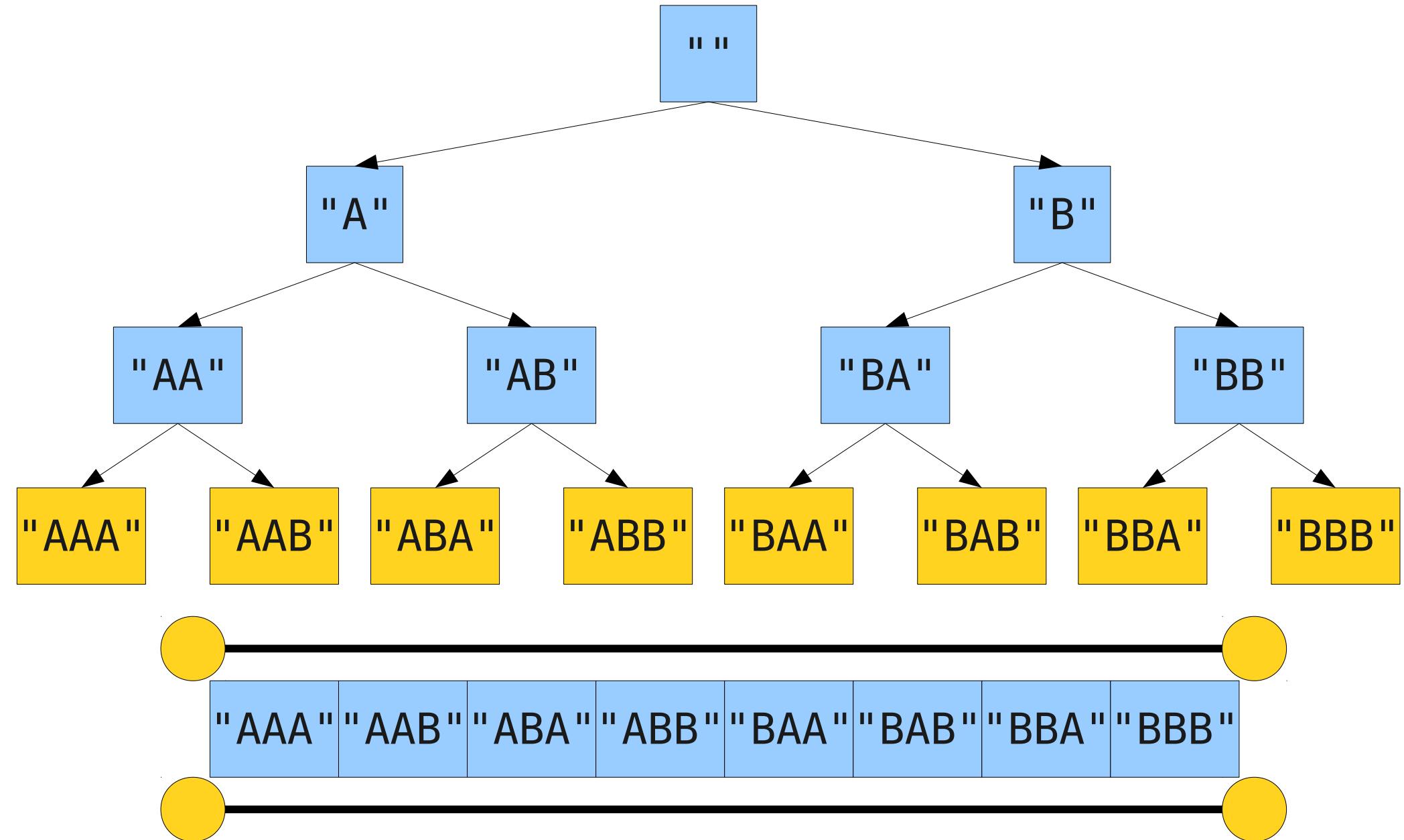
Memory Usage



Memory Usage



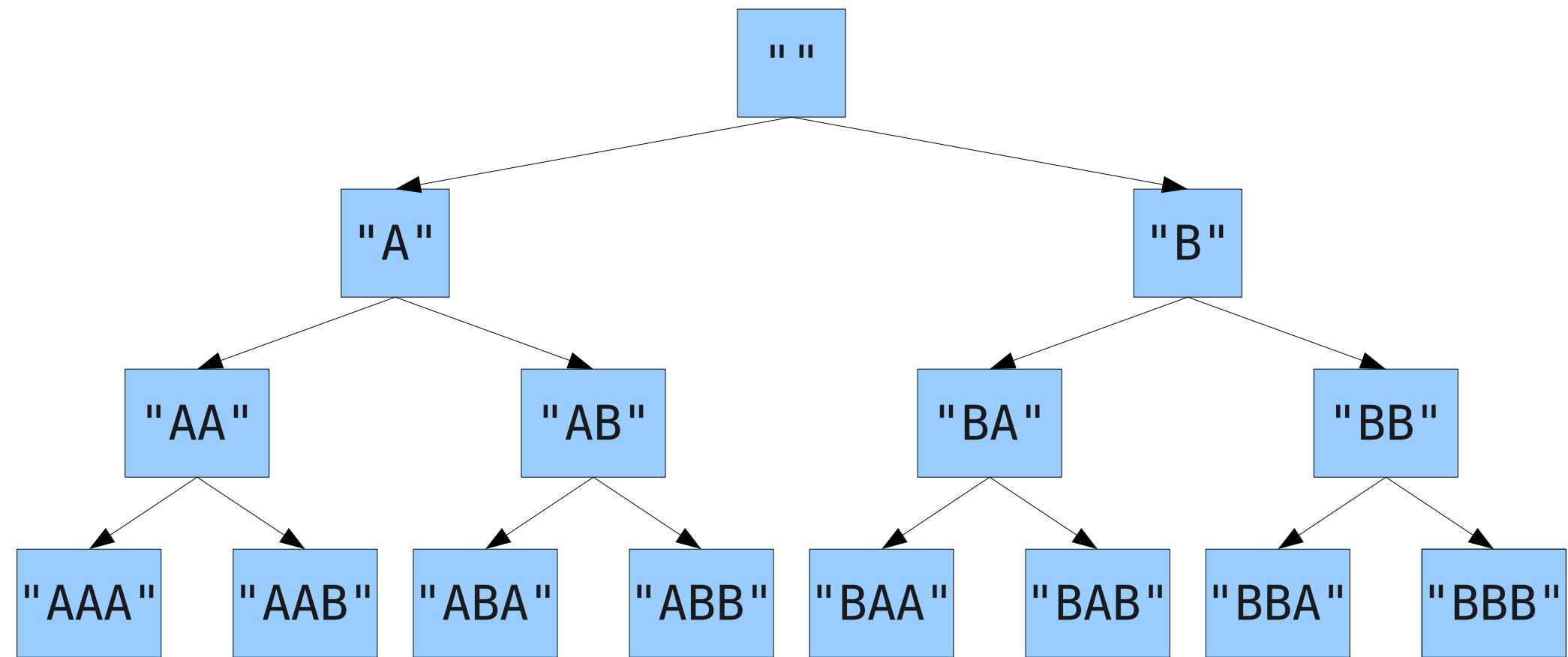
Memory Usage



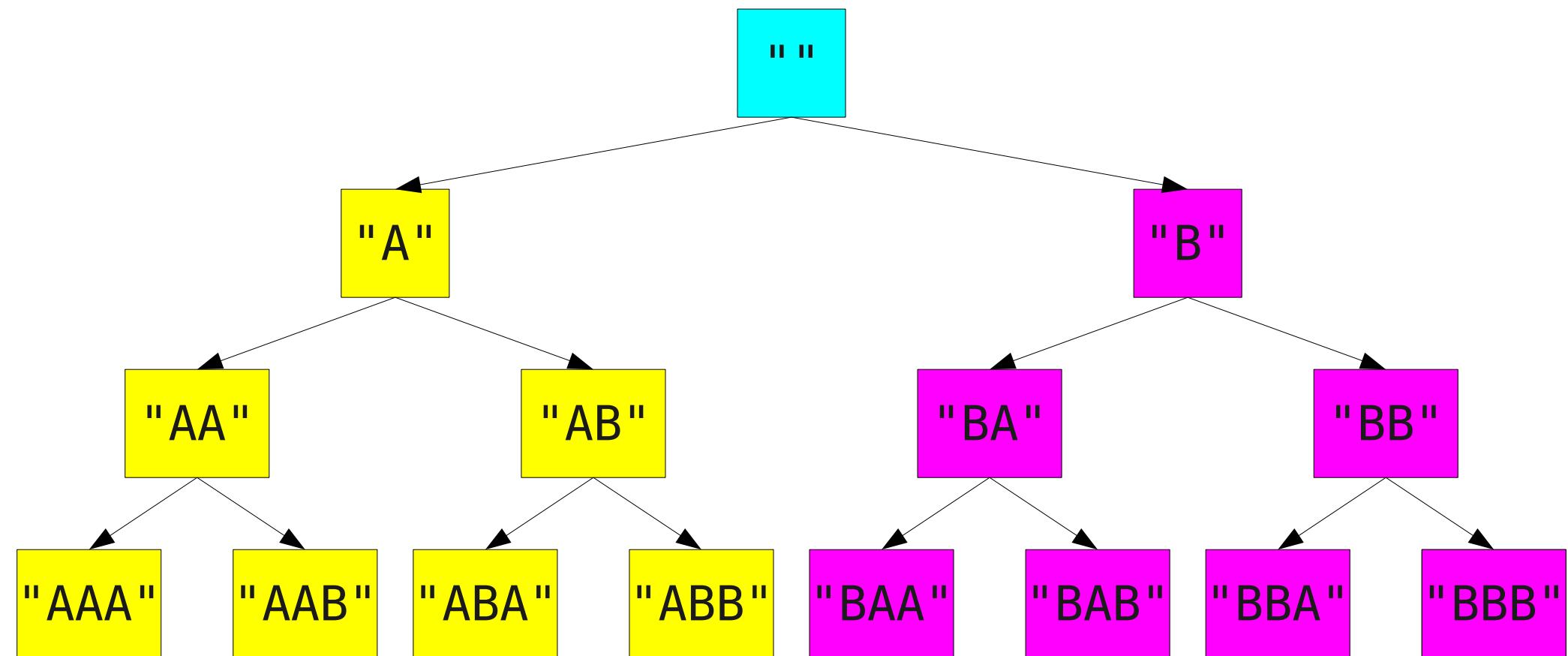
Memory Usage

- Our current brute-force attack requires us to have about 26^n strings in memory when attacking a password of length n .
- When n gets large, this is an *enormous* amount of memory.
- On a computer with only 4GB or 8GB of memory, we might run out of memory trying to do a brute-force attack!

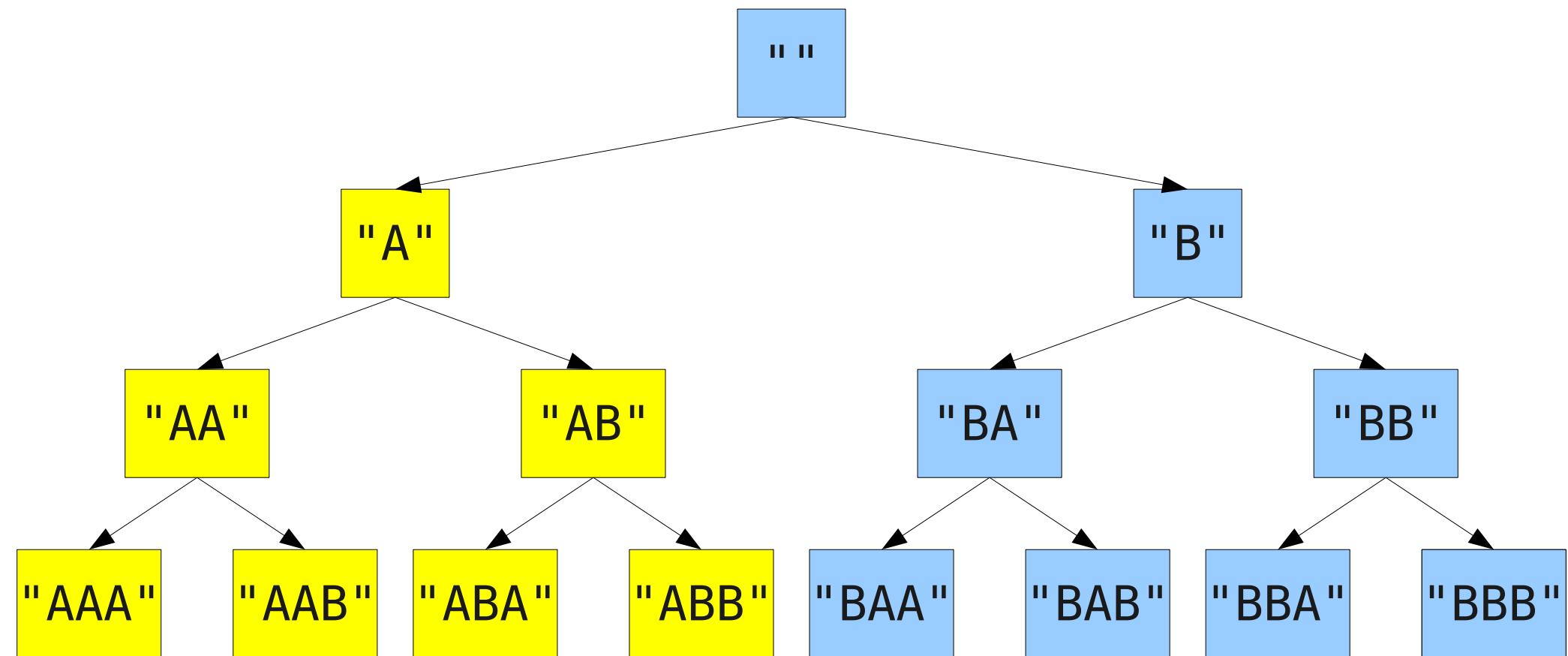
A Recursive Approach



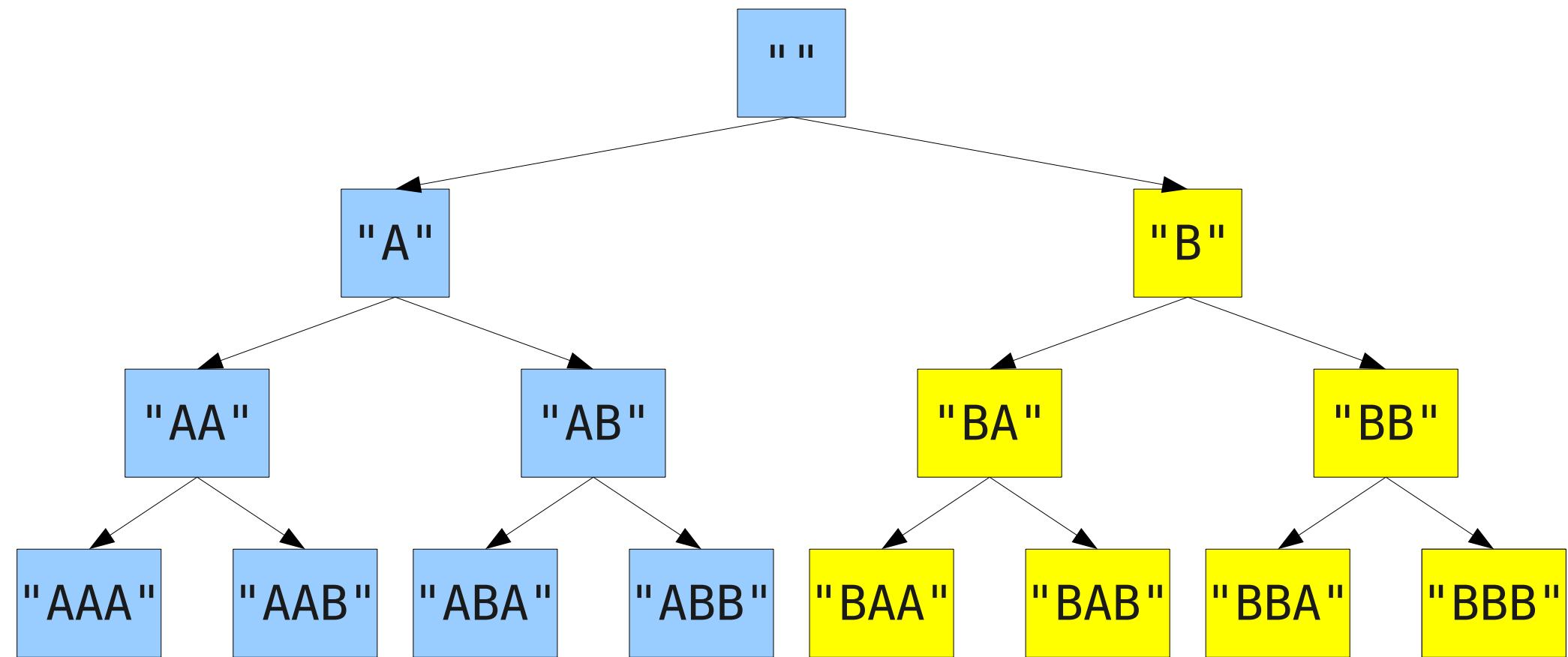
A Recursive Approach



A Recursive Approach



A Recursive Approach



A Recursive Approach

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

soFar "" maxLength 3

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

soFar "" maxLength 3

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

soFar "" maxLength 3

... ...

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```



""

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```



""

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

soFar "" maxLength 3

""

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar "A" maxLength 3

... ...

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar "A" maxLength 3

... ...

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar "A"

maxLength

3

..."

"A"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar "A"

maxLength

3

..."

"A"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar

"A"

maxLength

3

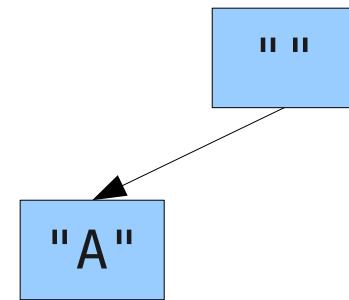
..."

"A"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

"A"

..."

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

"AA"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

"AA"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

"AA"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

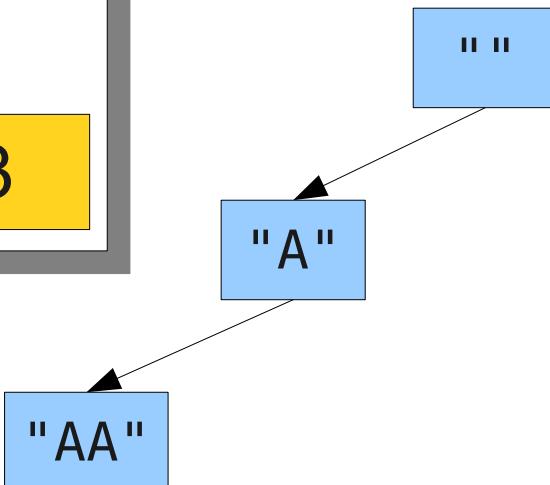
"AA"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

soFar "AAA"

maxLength 3

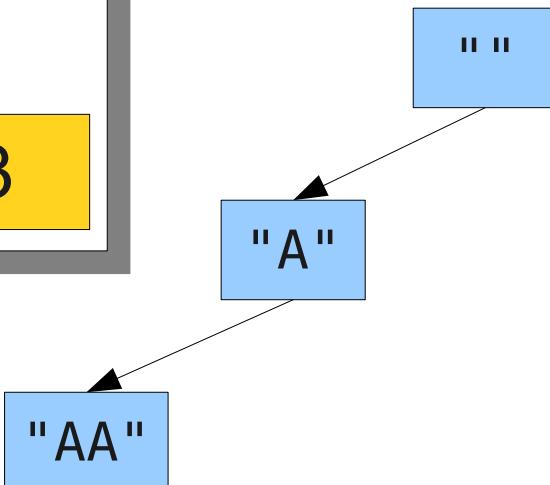


A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

soFar "AAA"

maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

soFar "AAA"

maxLength 3

"AAA"

"AA"

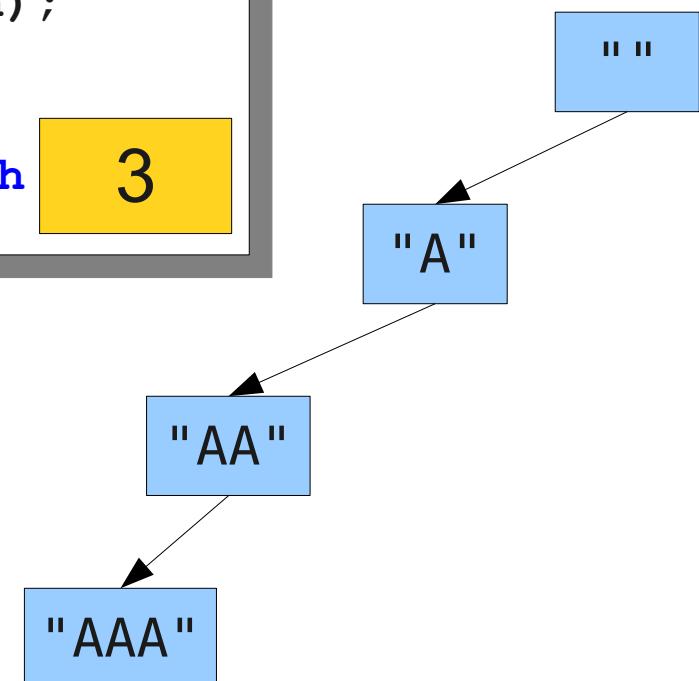
"A"

..."

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

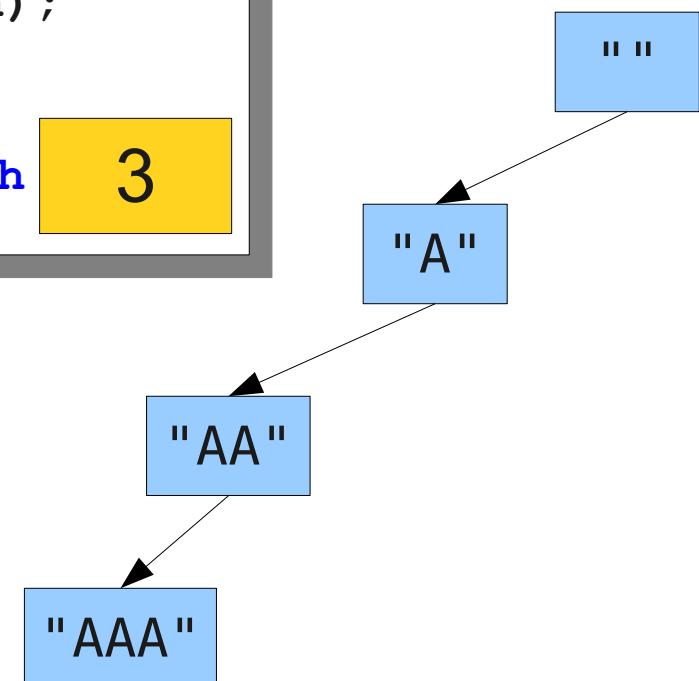
soFar "AAA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

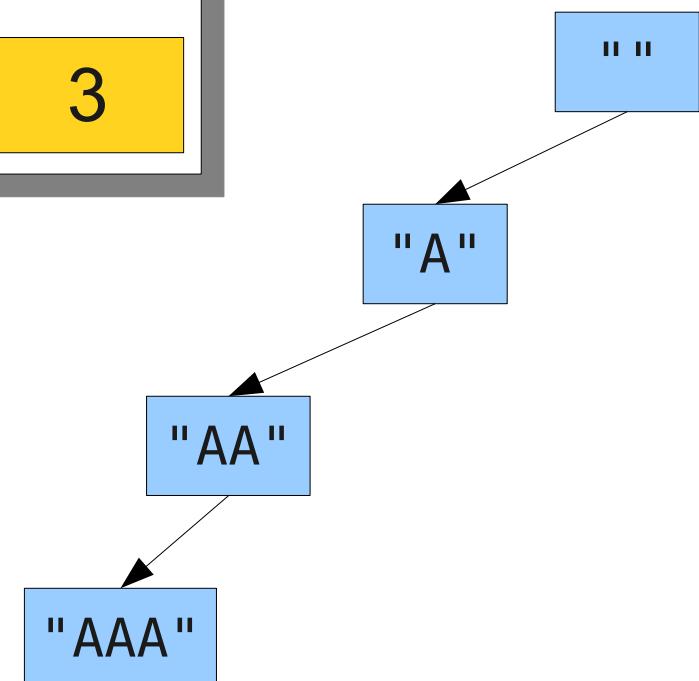
soFar "AAA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar "AA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar

"AA"

maxLength

3

..."

"A"

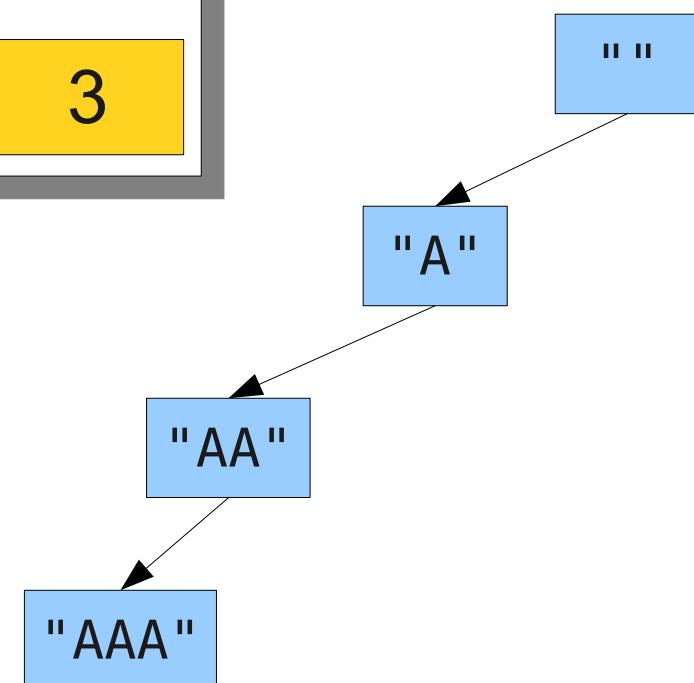
"AA"

"AAA"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

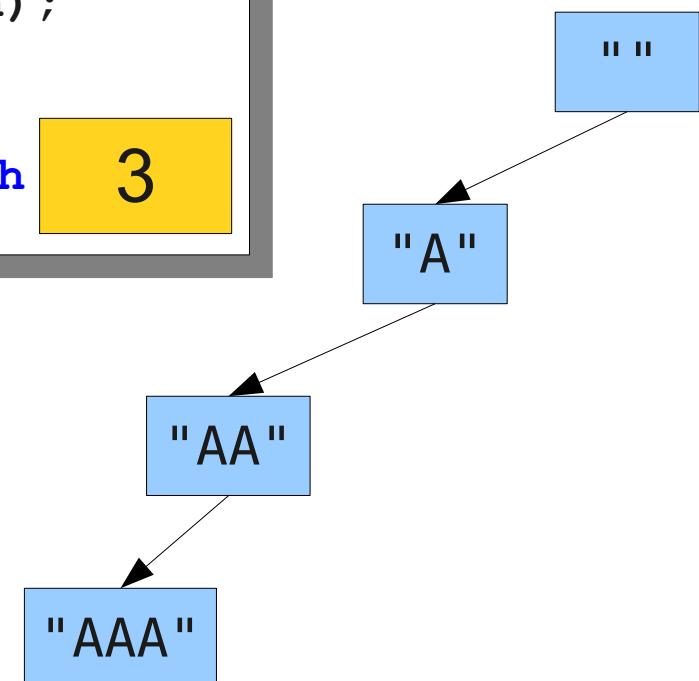
soFar "AA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

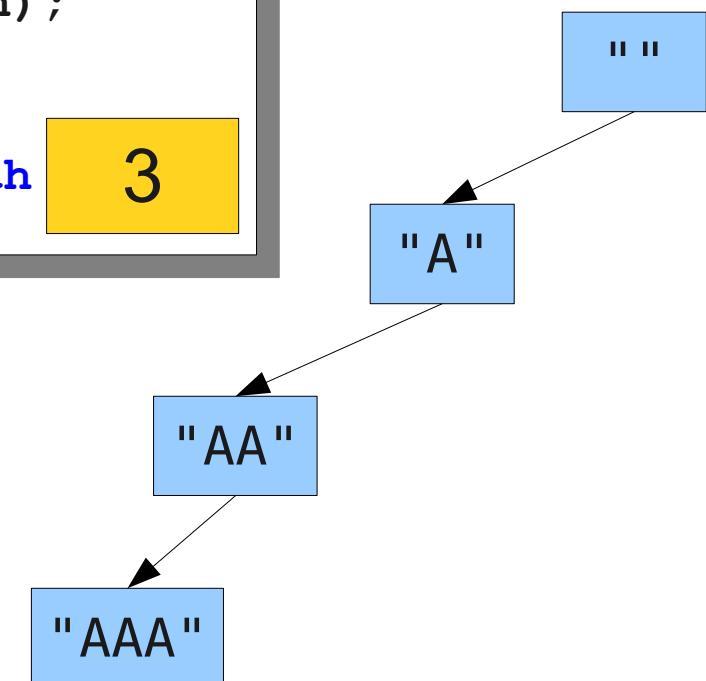
soFar "AAB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

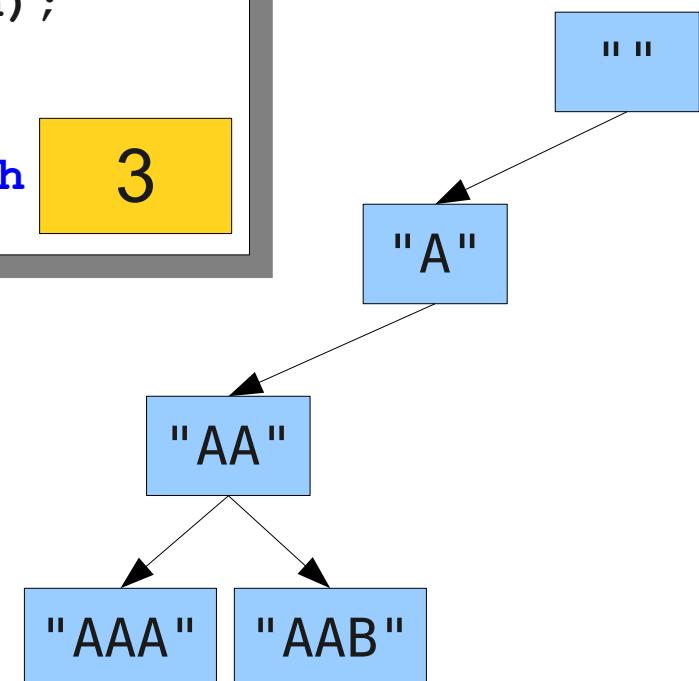
soFar "AAB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

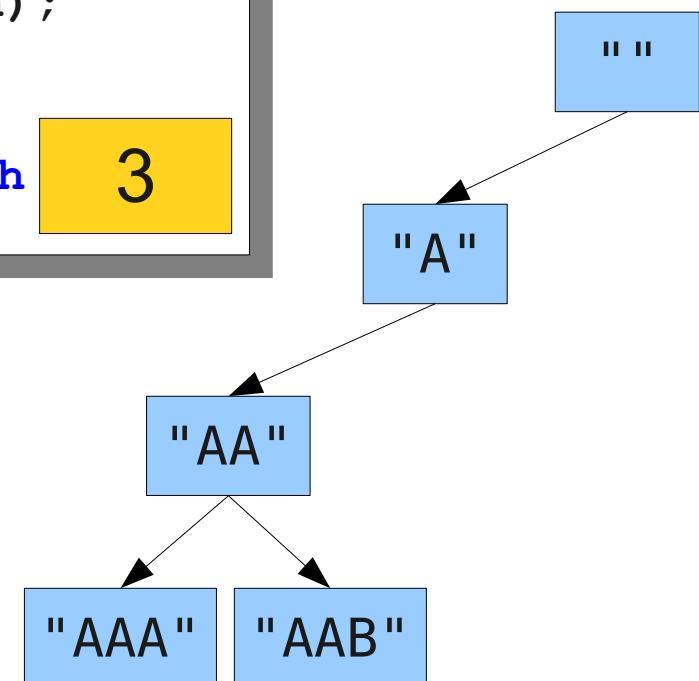
soFar "AAB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

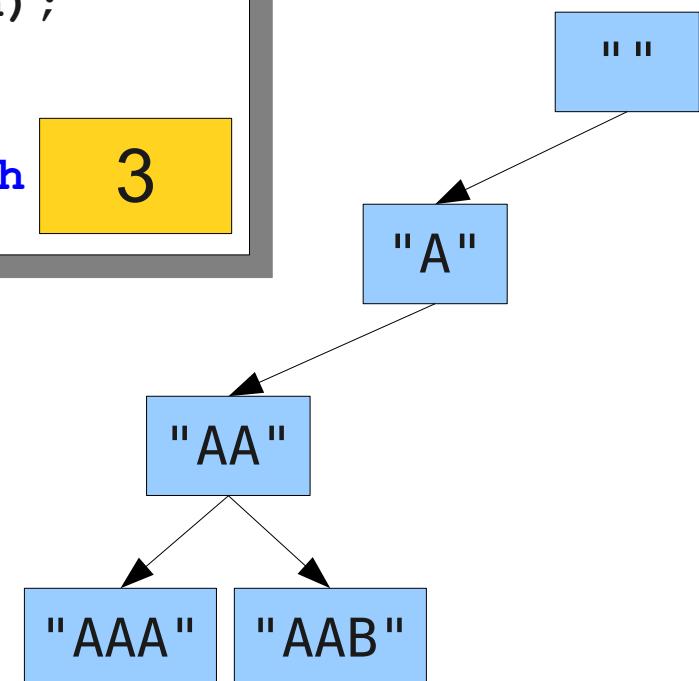
soFar "AAB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

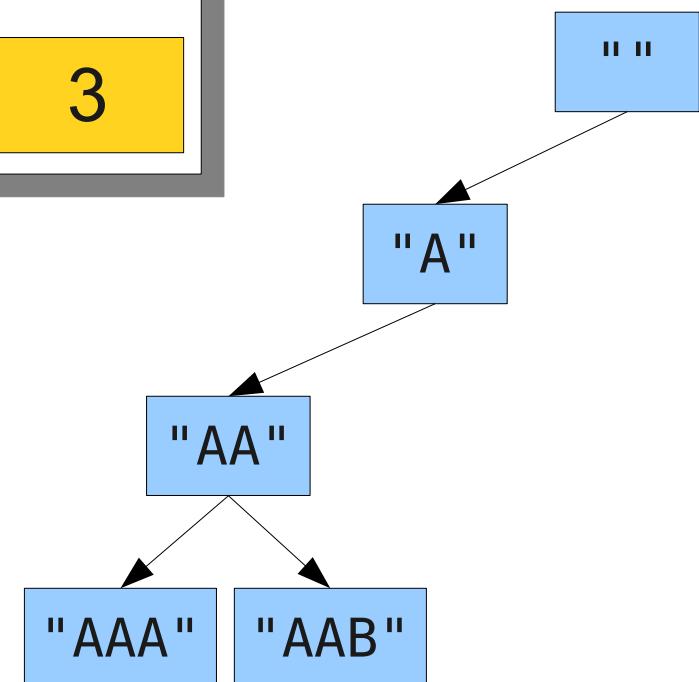
soFar "AAB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

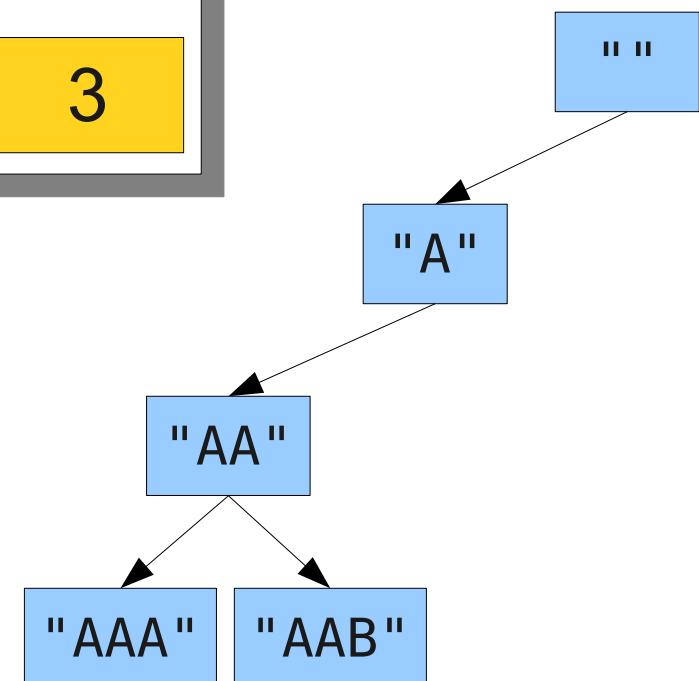
soFar "AA"
maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

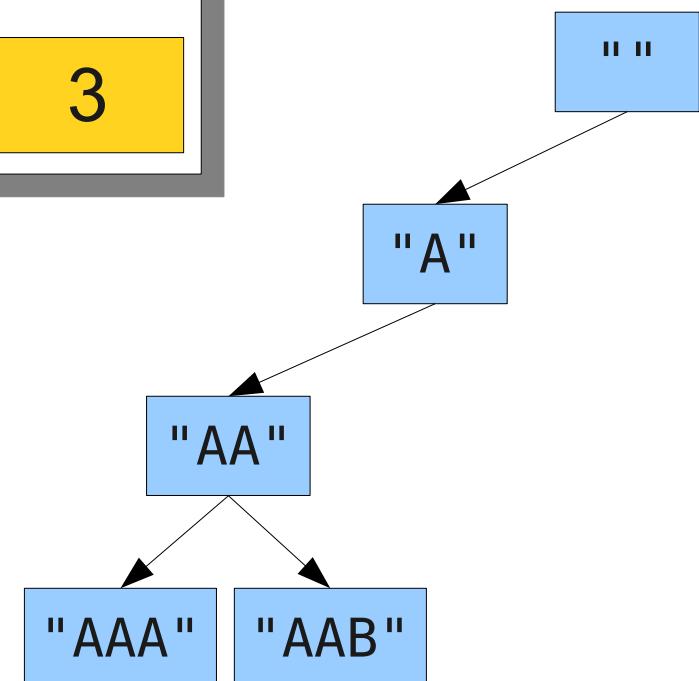
soFar "AA"
maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

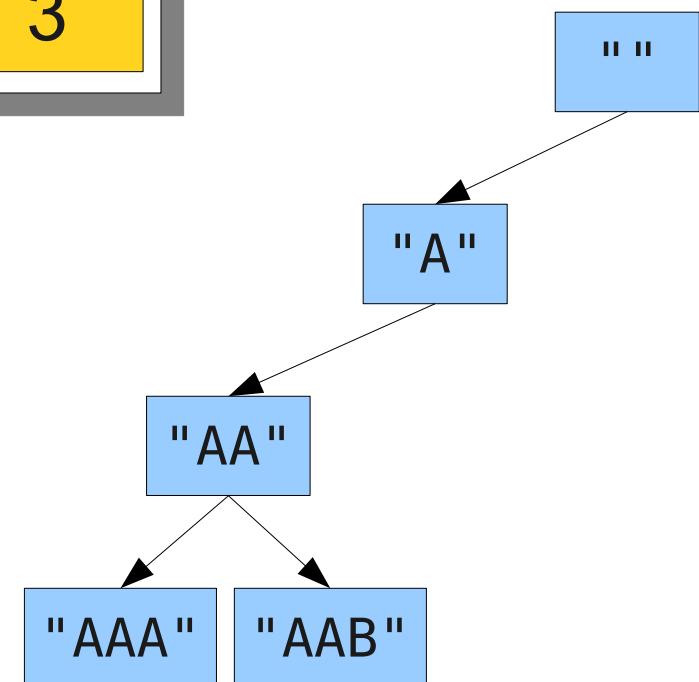
soFar "AA"
maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

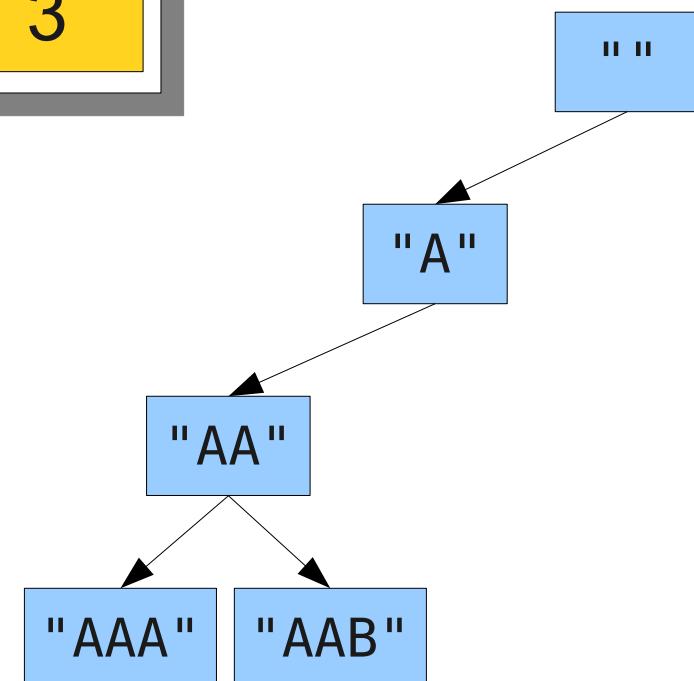
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

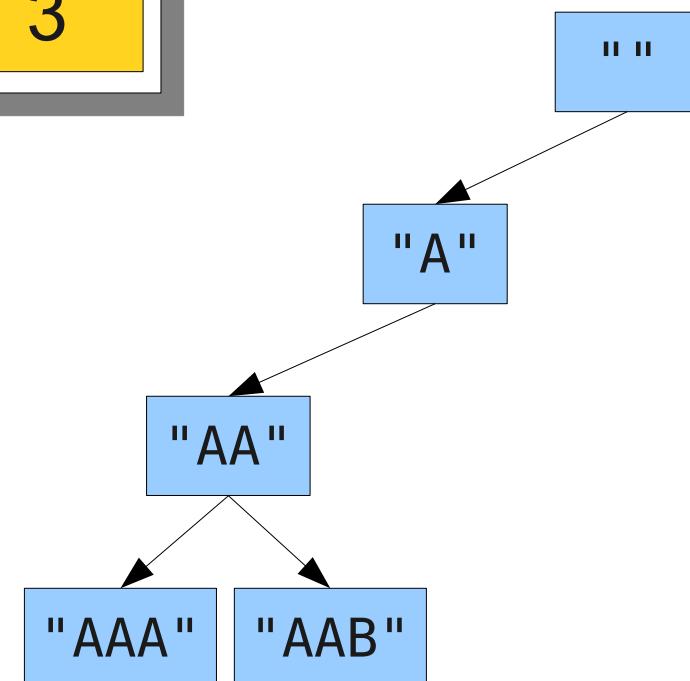
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

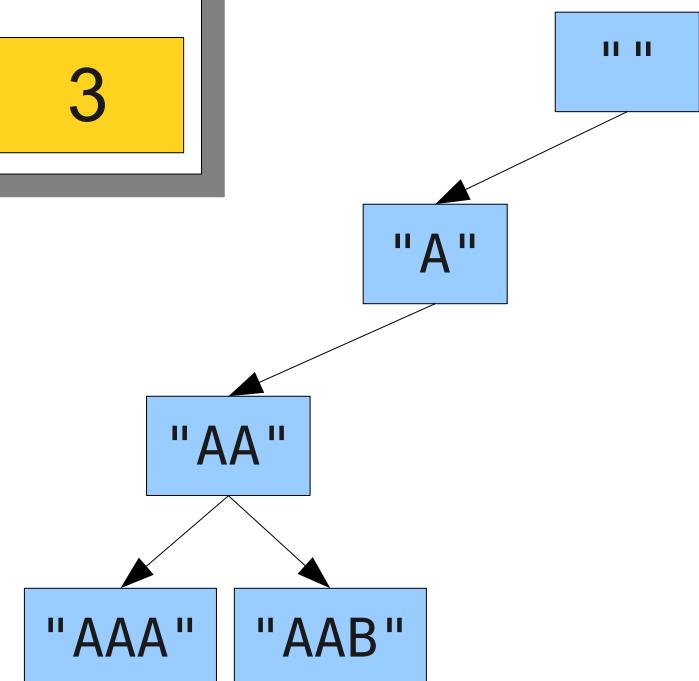
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

soFar "AB"

maxLength 3

" "

"A"

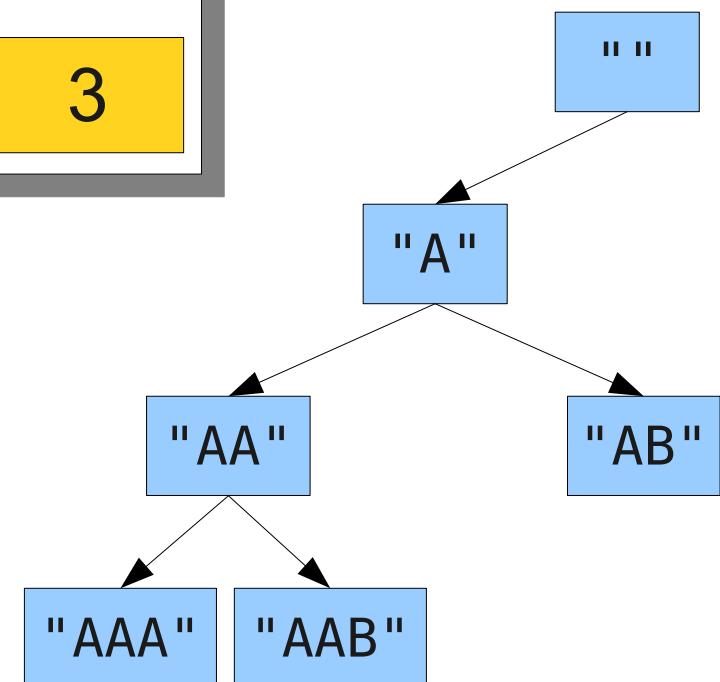
"AA"

"AAA" "AAB"

A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

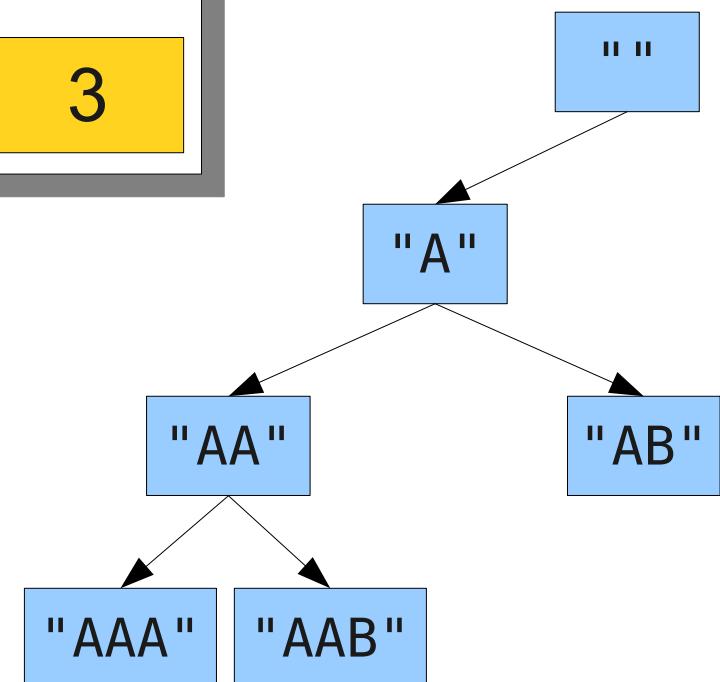
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

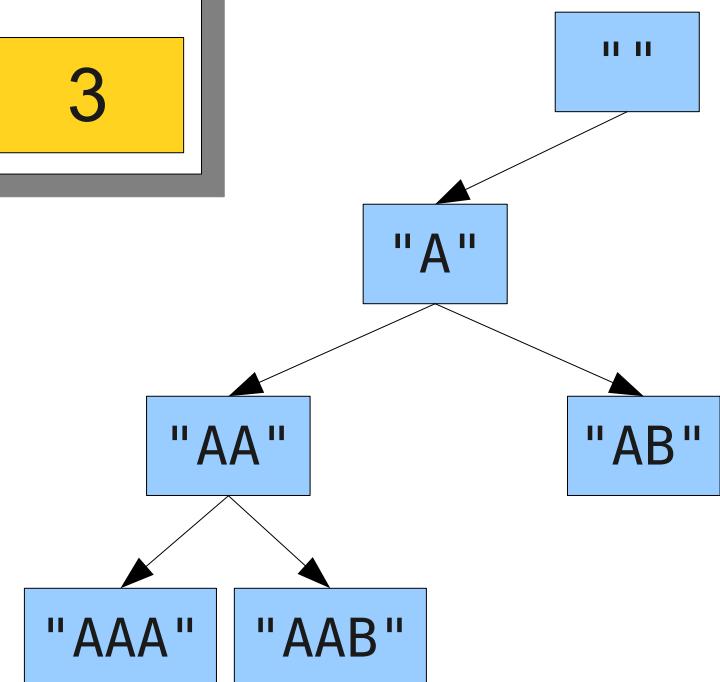
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

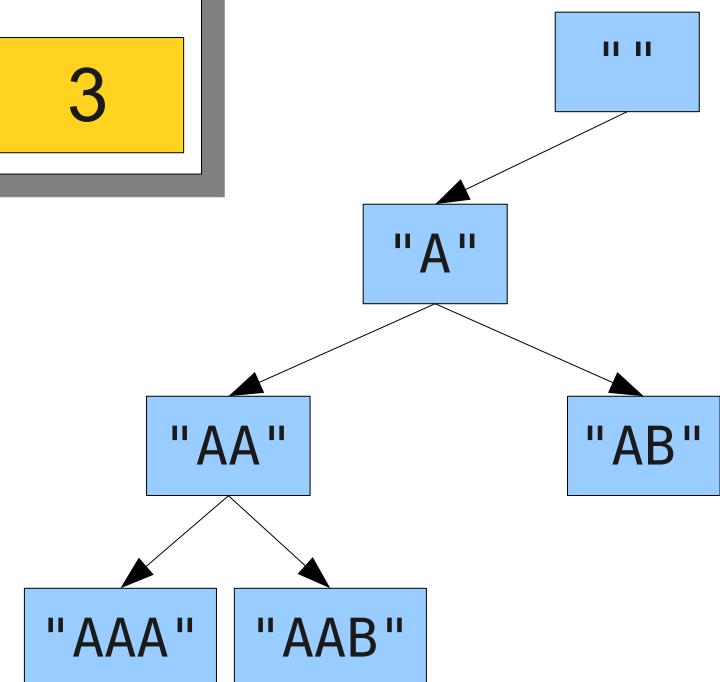
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

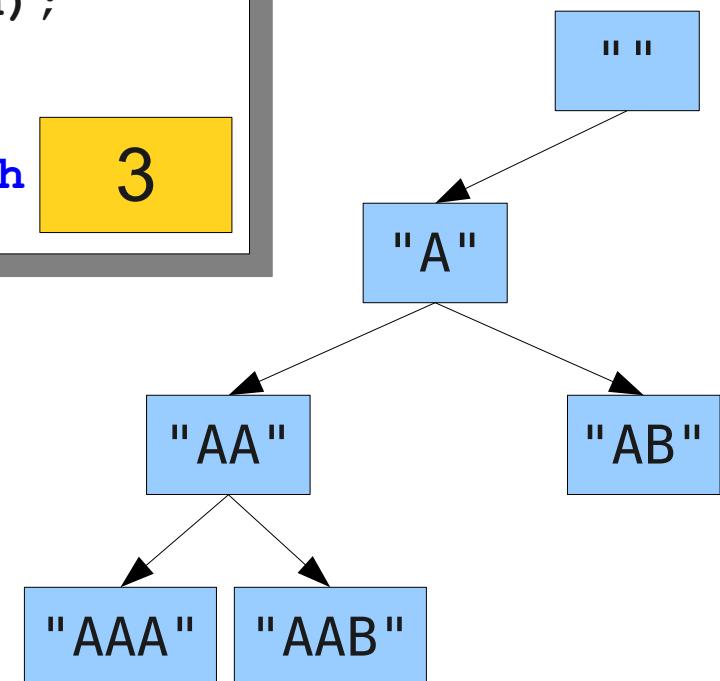
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

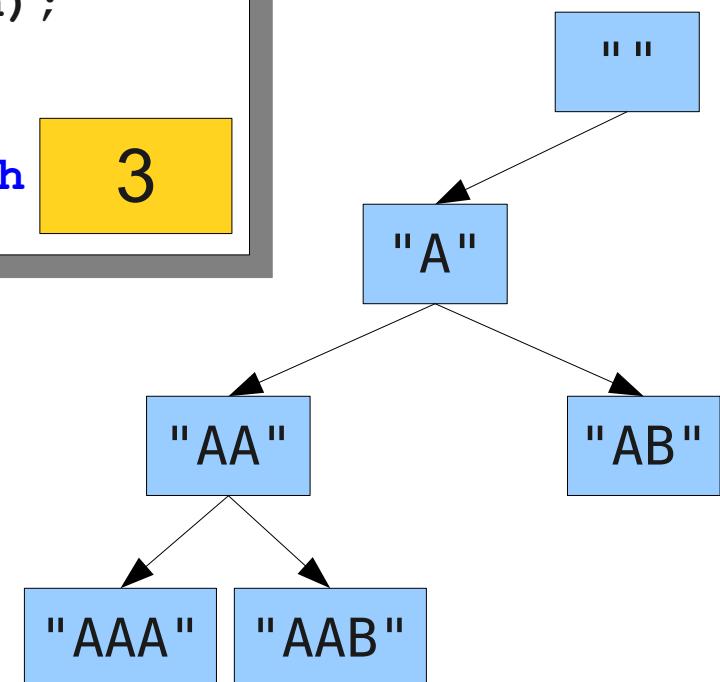
soFar "ABA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

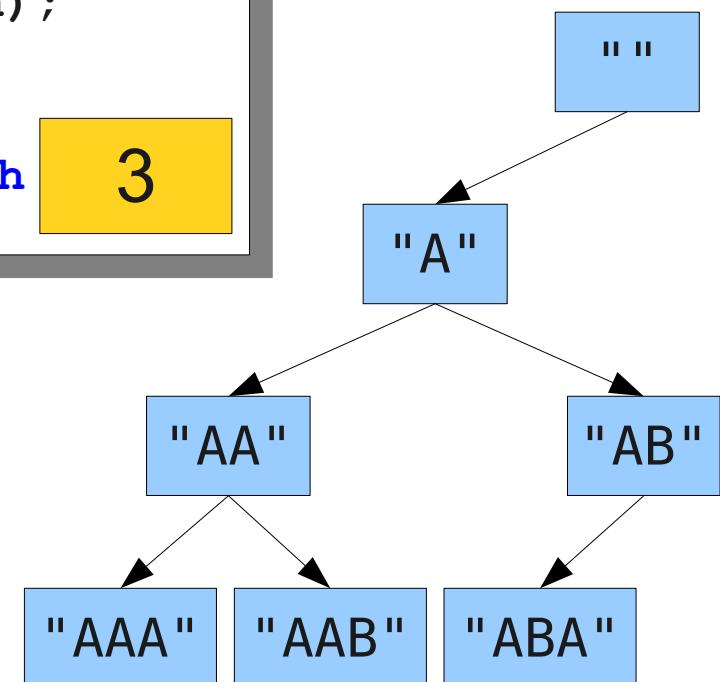
soFar "ABA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

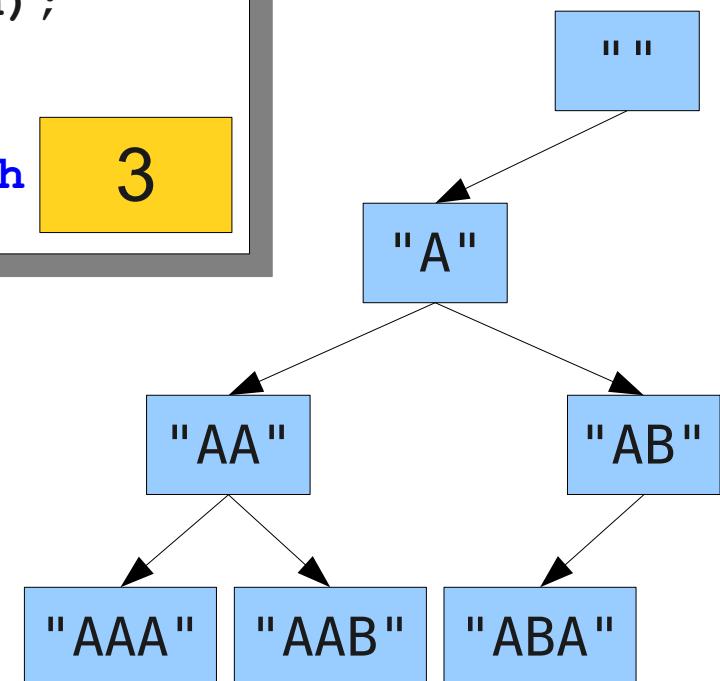
soFar "ABA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

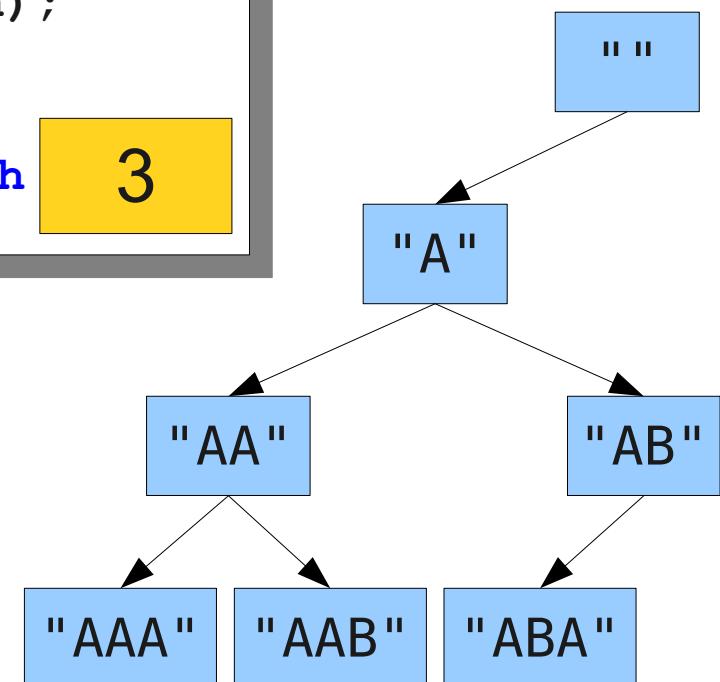
soFar "ABA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

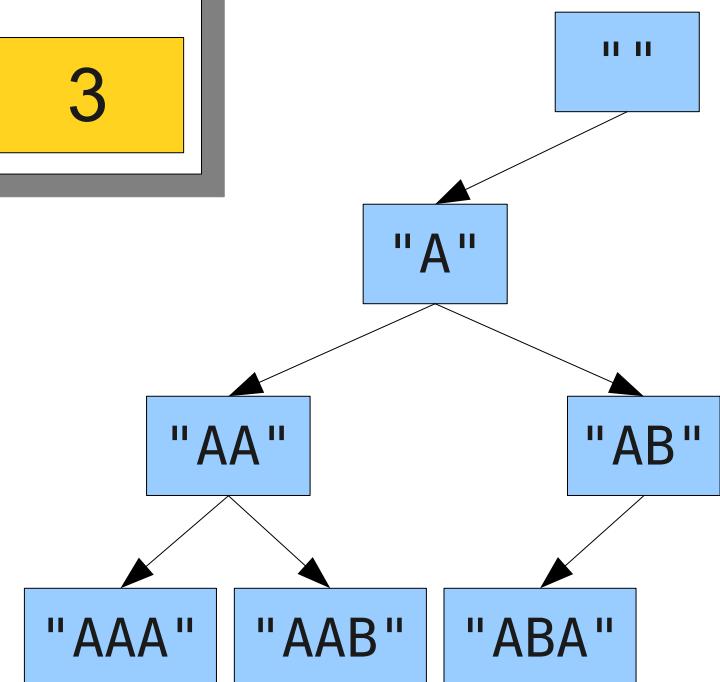
soFar "ABA" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

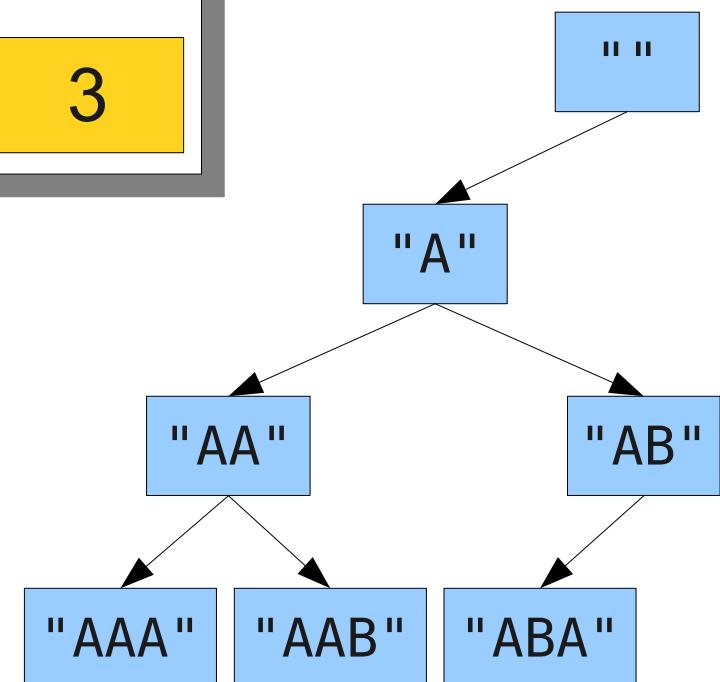
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

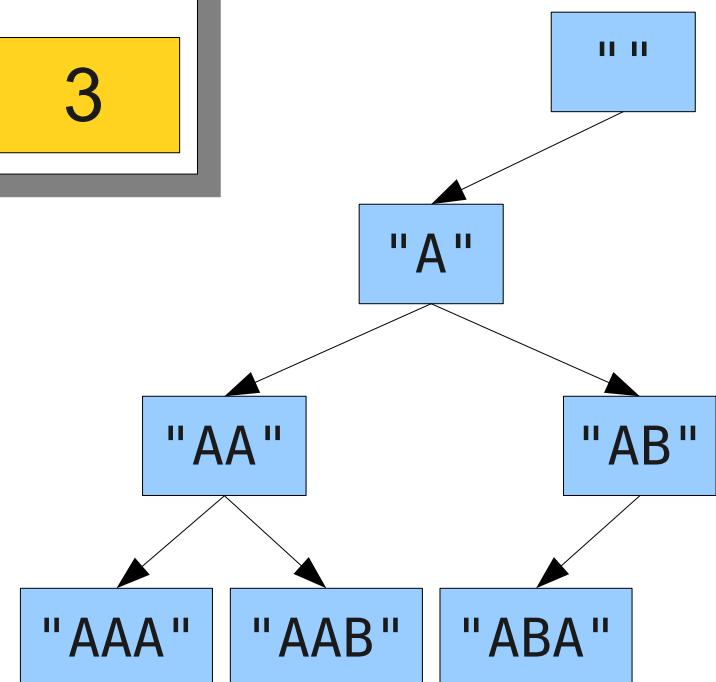
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

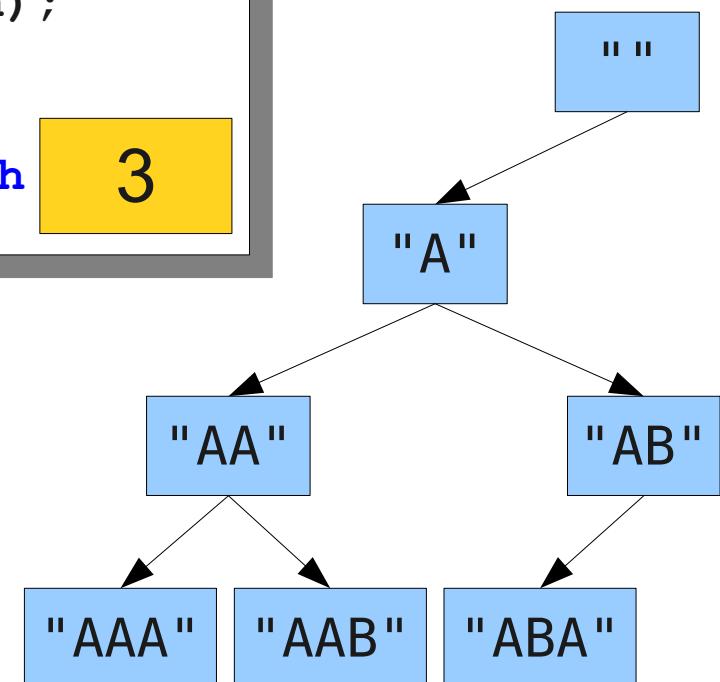
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

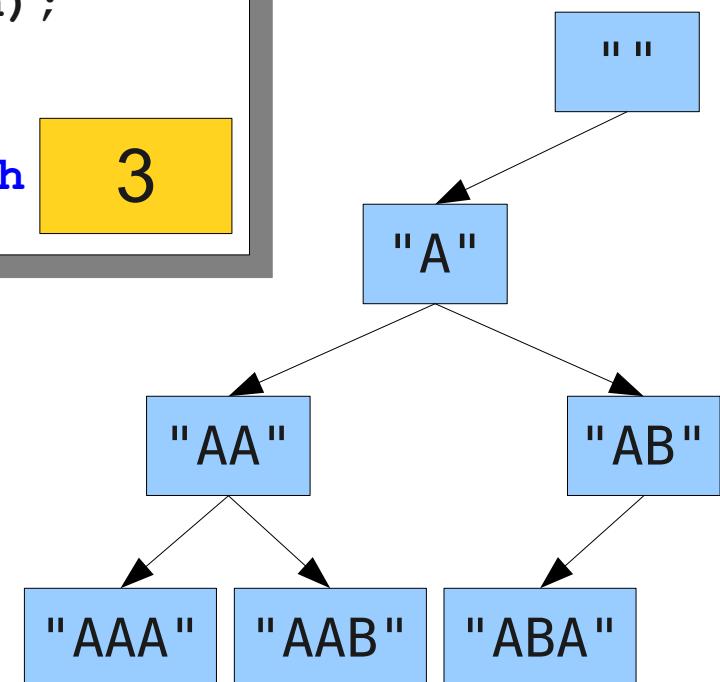
soFar "ABB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

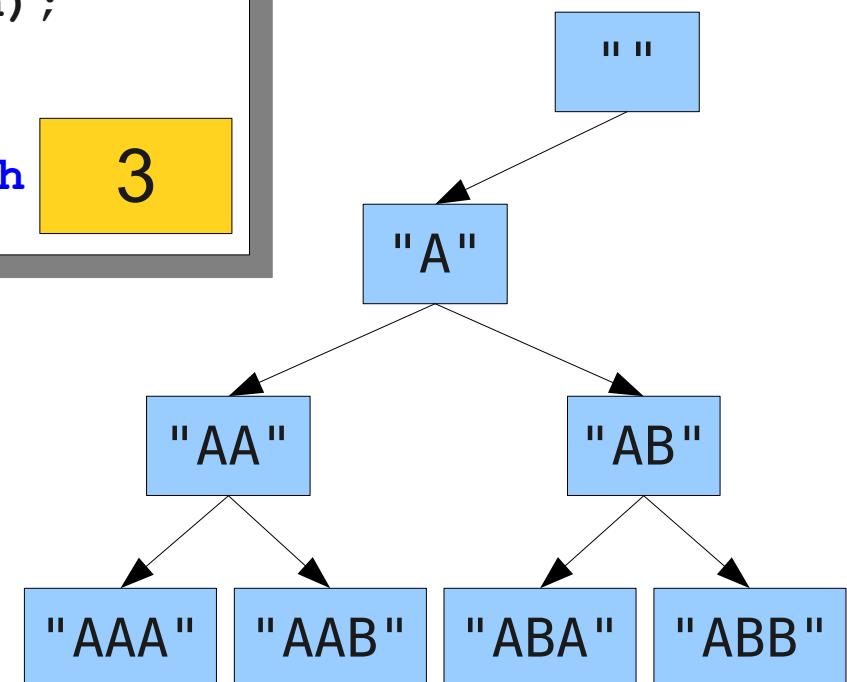
soFar "ABB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

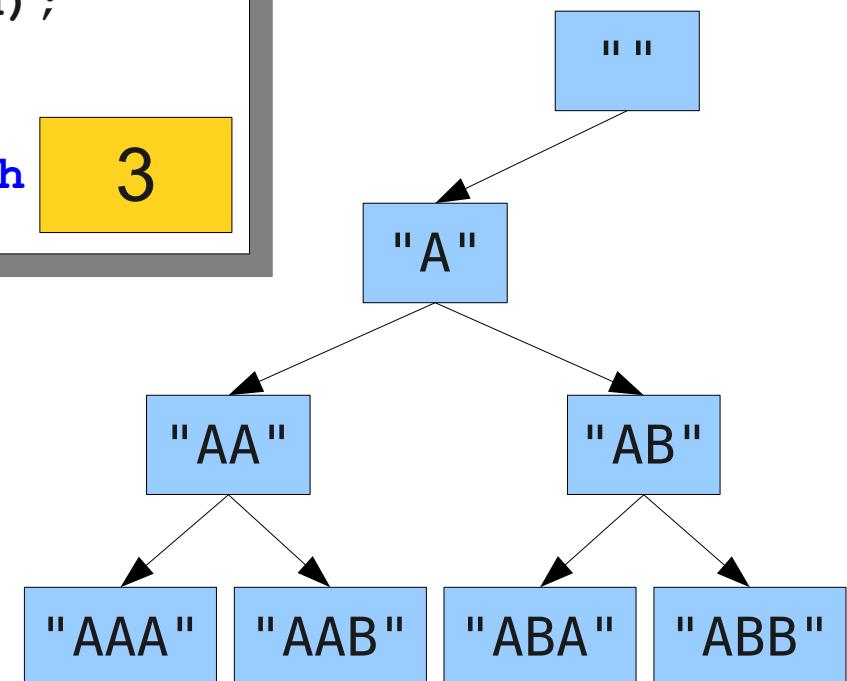
soFar "ABB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

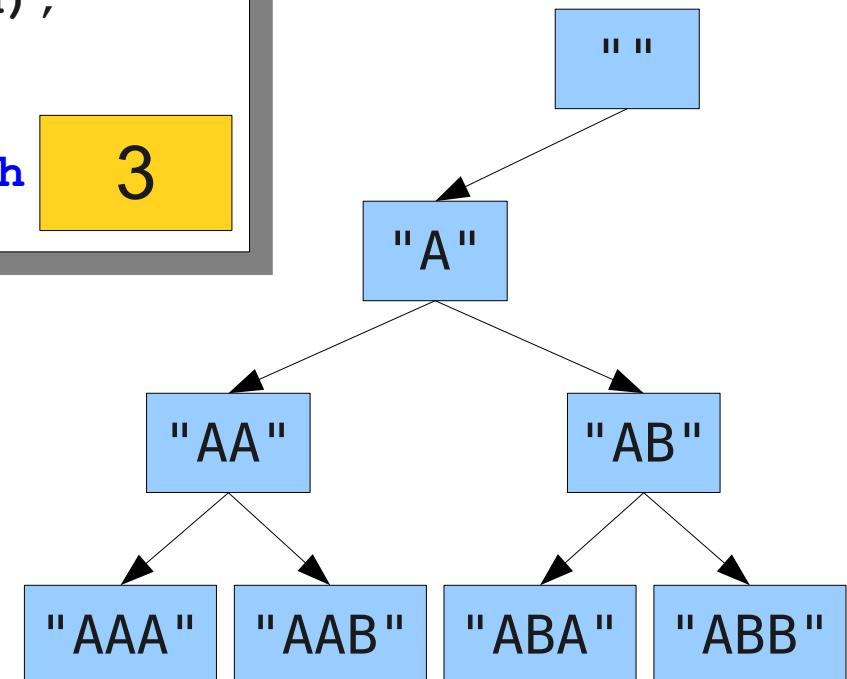
soFar "ABB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

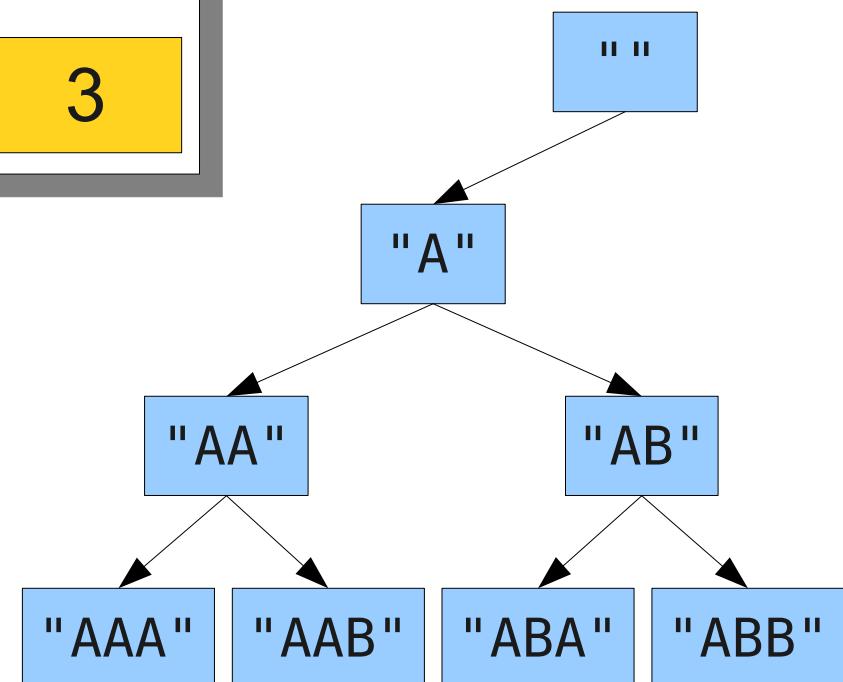
soFar "ABB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

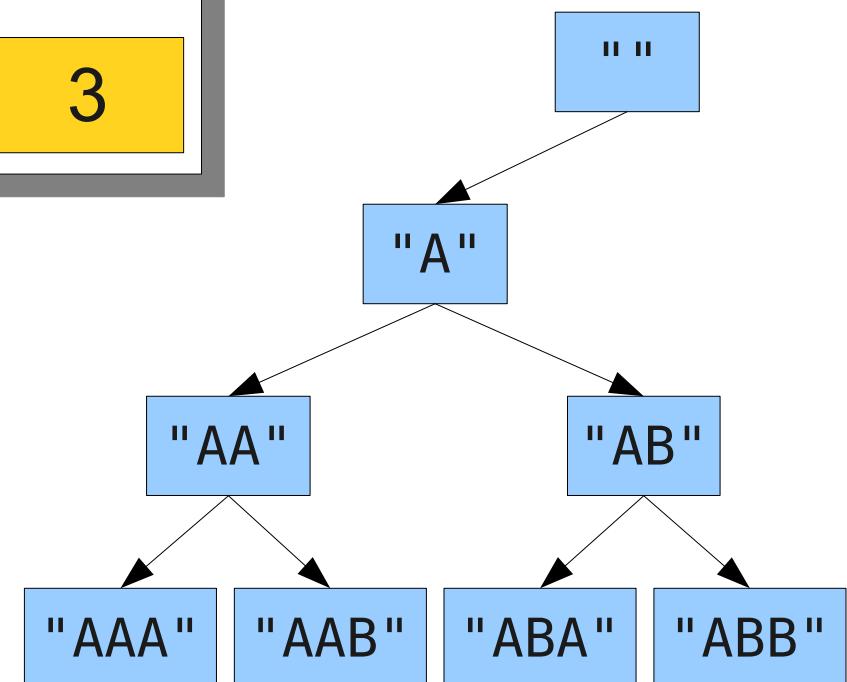
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

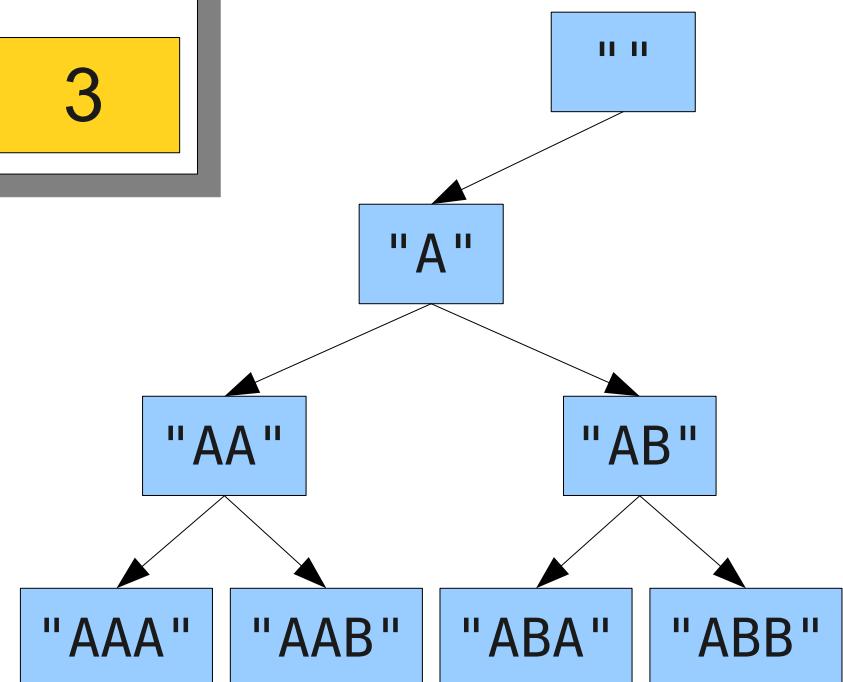
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            doSomethingWith(soFar);  
            if (soFar.length() < maxLength) {  
                for (char ch = 'A'; ch <= 'B'; ch++) {  
                    generate(soFar + ch, maxLength);  
                }  
            }  
        }  
    }  
}
```

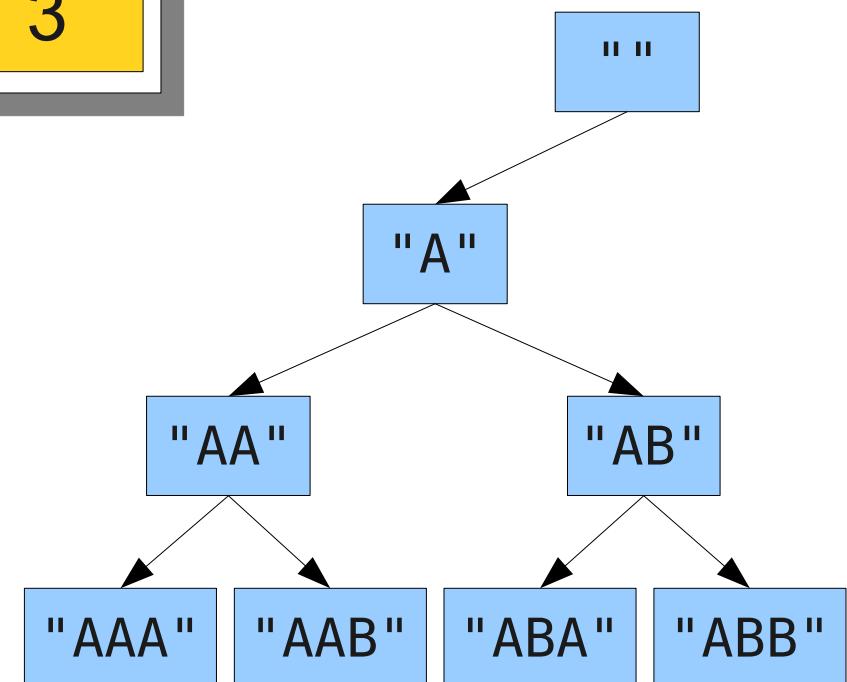
soFar "AB" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

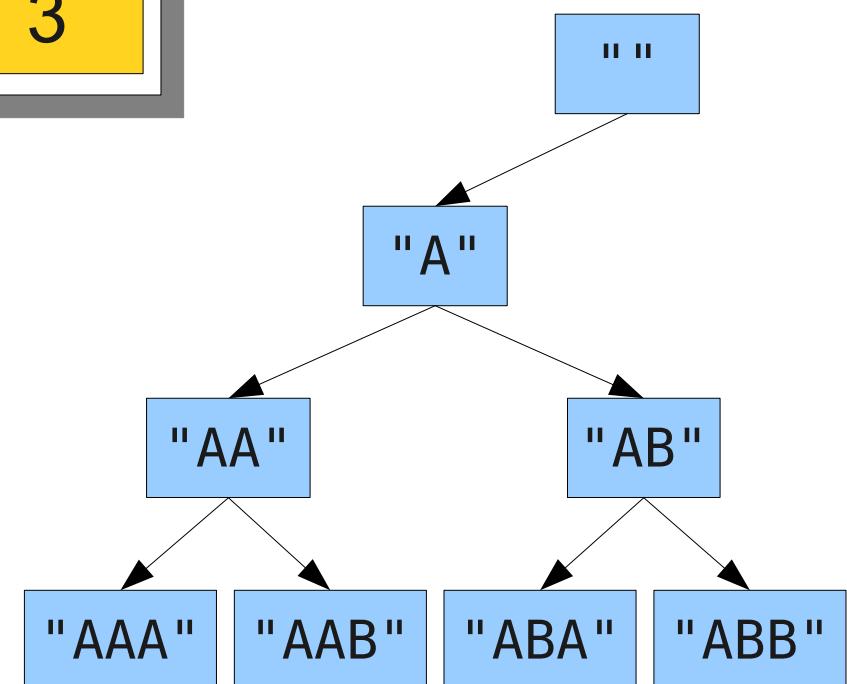
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

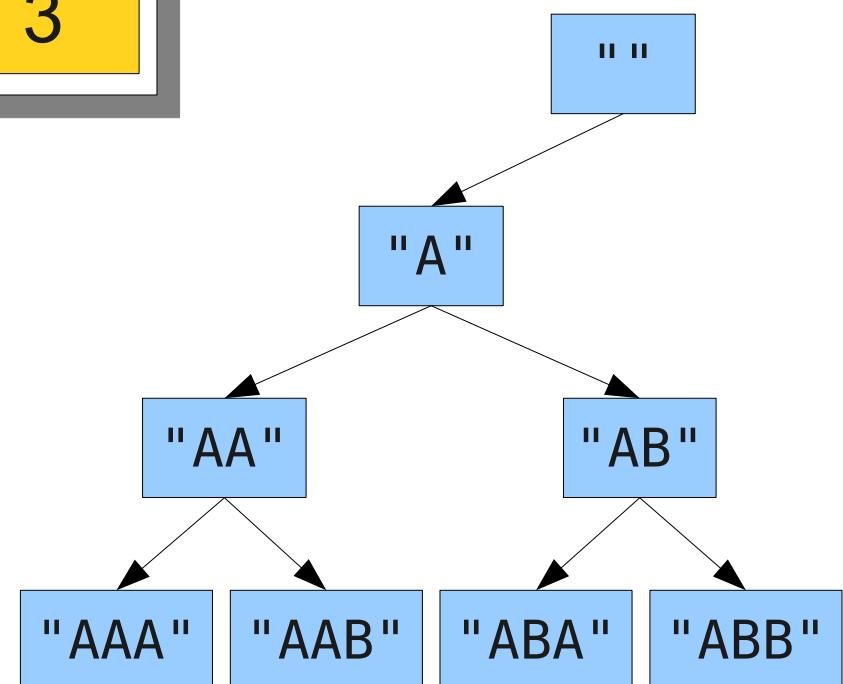
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        doSomethingWith(soFar);  
        if (soFar.length() < maxLength) {  
            for (char ch = 'A'; ch <= 'B'; ch++) {  
                generate(soFar + ch, maxLength);  
            }  
        }  
    }  
}
```

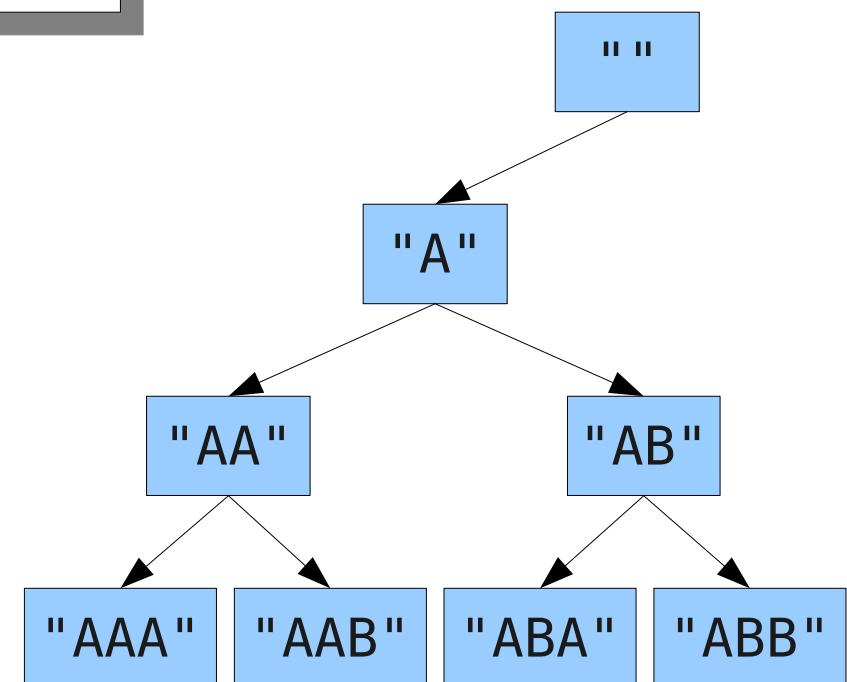
soFar "A" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

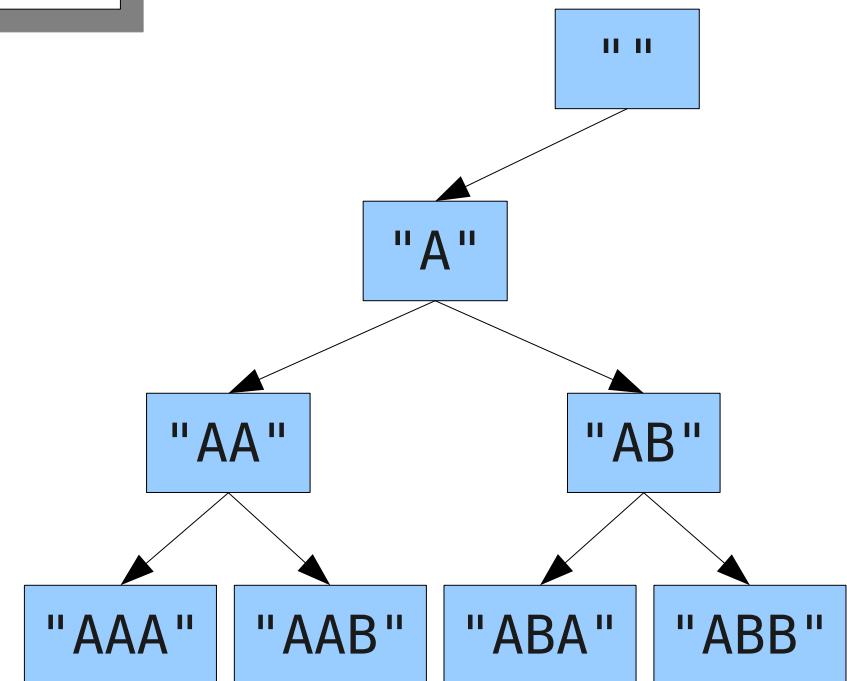
soFar "" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

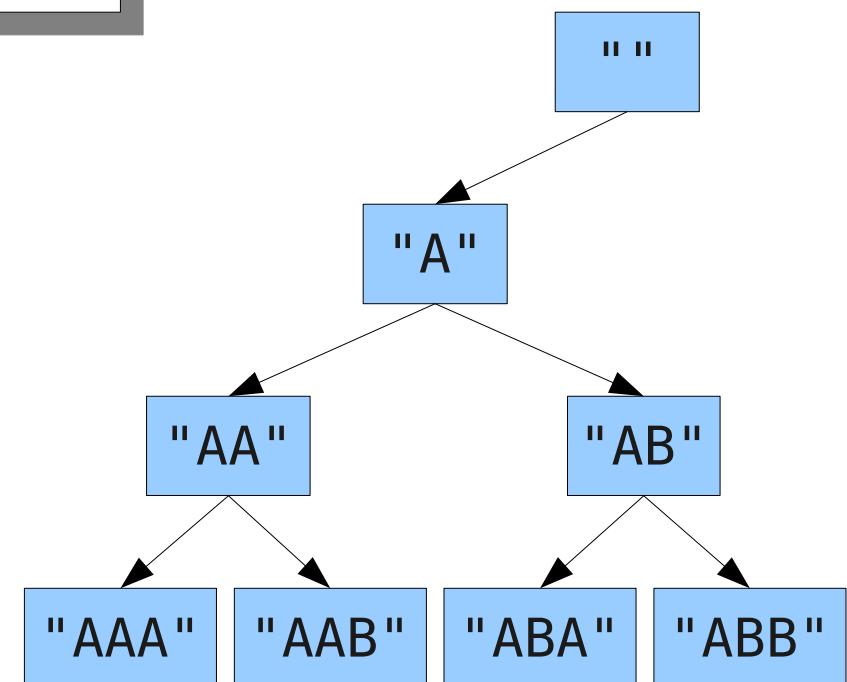
soFar "" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

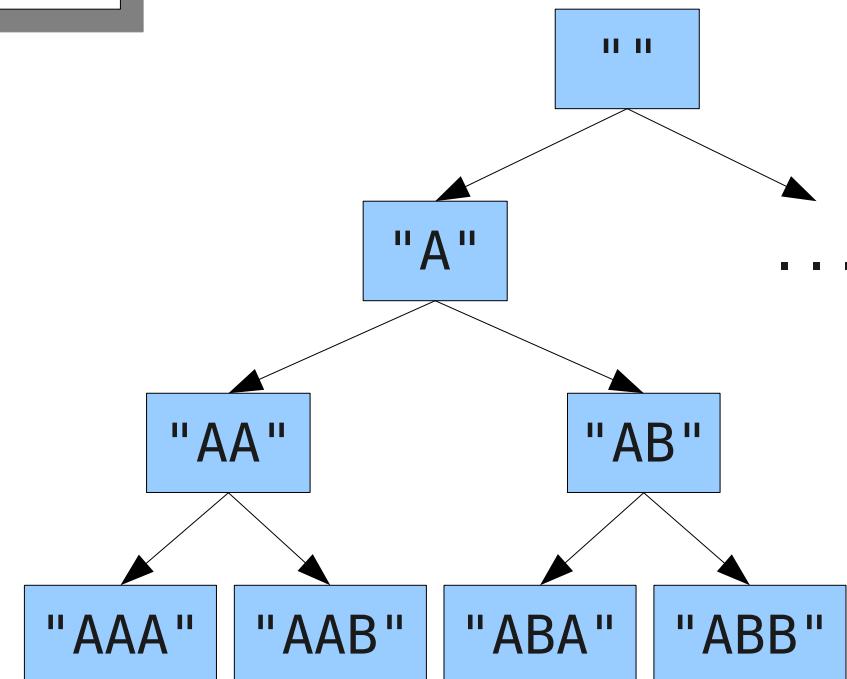
soFar "" maxLength 3



A Recursive Approach

```
void generate(string soFar, int maxLength) {  
    doSomethingWith(soFar);  
    if (soFar.length() < maxLength) {  
        for (char ch = 'A'; ch <= 'B'; ch++) {  
            generate(soFar + ch, maxLength);  
        }  
    }  
}
```

soFar "" maxLength 3

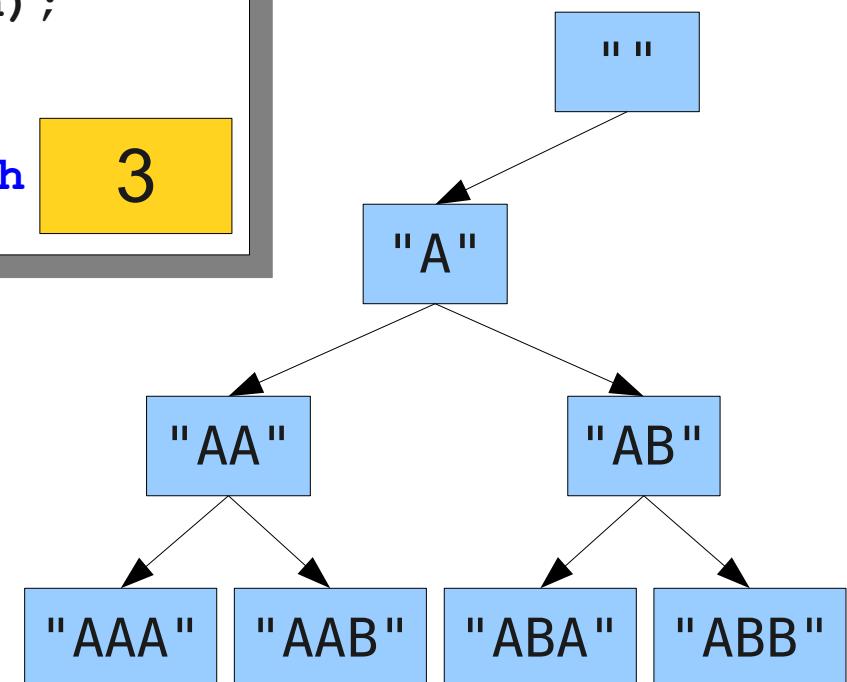


Why This Matters

Why This Matters

```
void generate(string soFar, int maxLength) {  
    void generate(string soFar, int maxLength) {  
        void generate(string soFar, int maxLength) {  
            void generate(string soFar, int maxLength) {  
                doSomethingWith(soFar);  
                if (soFar.length() < maxLength) {  
                    for (char ch = 'A'; ch <= 'B'; ch++) {  
                        generate(soFar + ch, maxLength);  
                    }  
                }  
            }  
        }  
    }  
}
```

soFar "ABB" maxLength 3



A Comparison

- Queue-based approach:
 - Time: $\approx 26^{n+1}$
 - Space: $\approx 26^{n+1}$
- This algorithm is called ***breadth-first search*** (or just **BFS**).
- Although less useful here, BFS has many applications; you'll see one in Assignment 2.
- Recursive approach:
 - Time: $\approx 26^{n+1}$
 - Space: $\approx n$
- This algorithm is called ***depth-first search*** (or just **DFS**).
- Also quite useful; we'll see more applications later on.

Next Time

- **Thinking Recursively**
 - How can you best solve problems using recursion?
 - What techniques are necessary to do so?
 - And what problems yield easily to a recursive solution?