# Exhaustive Search

Hitesh Boinpally
Summer 2023

# Agenda

- Exhaustive Search
- Gambling
- Phone Numbers

# Agenda

- Exhaustive Search 
- Gambling
- Phone Numbers

# Exhaustive Search

# Exhaustive Search

- Brute force algorithm!
  - Not **smart**
- Simply trying all possibilities, find the ones that work
- Often solved recursively

# `fourAB`

- Write a method to print out all `String`s four letters long made up of only `'a'`s and `'b'`s

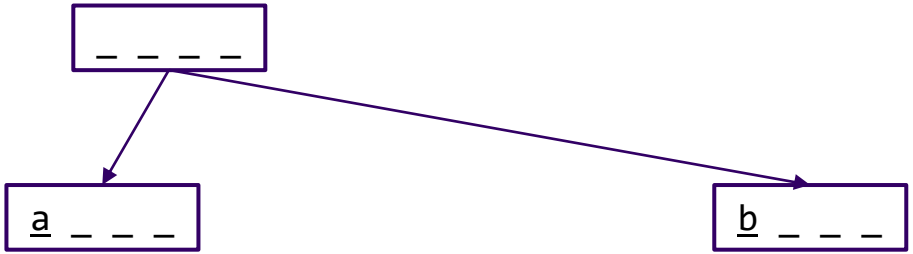
# fourAB

- Write a method to print out all `String`s four letters long made up of only `'a'`s and `'b'`s

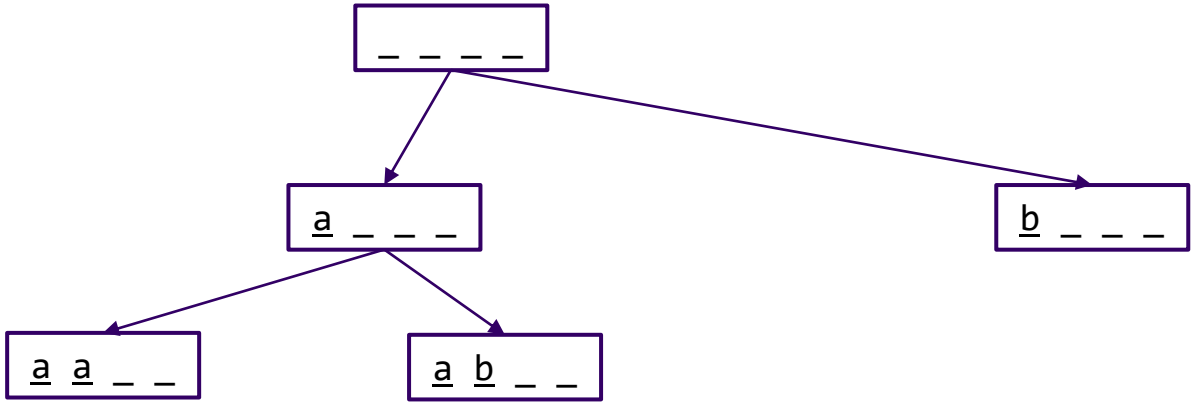
```
aaaa
aaab
aaba
aabb
abaa
abab
abba
abbb
baaa
baab
baba
babb
bbaa
bbab
bbba
bbbb
```

# fourAB Visualized

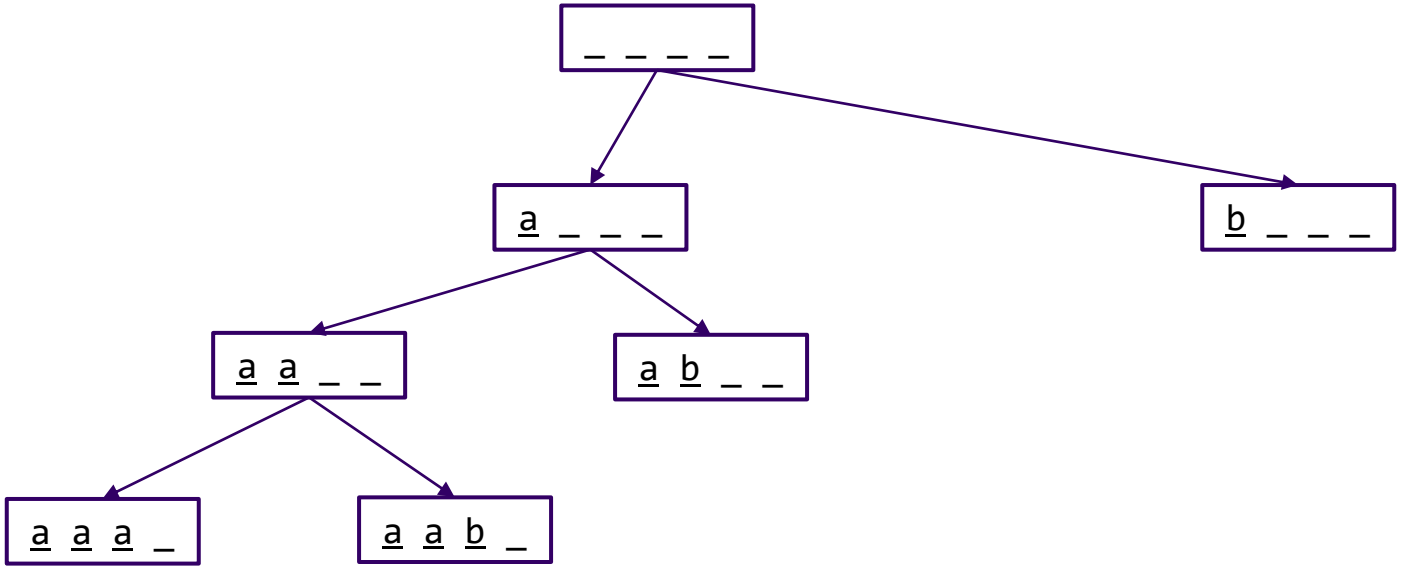
# fourAB Visualized



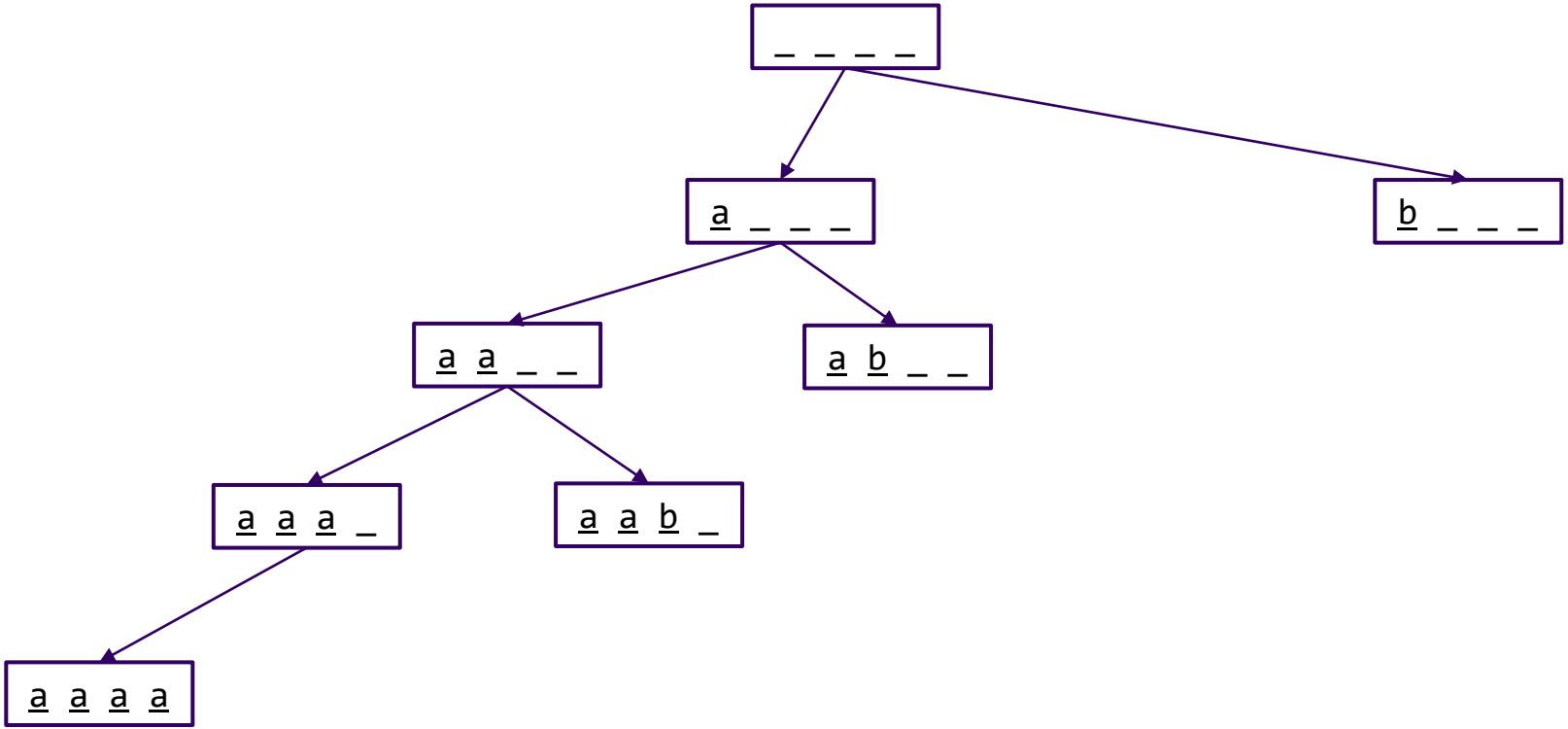
# fourAB Visualized



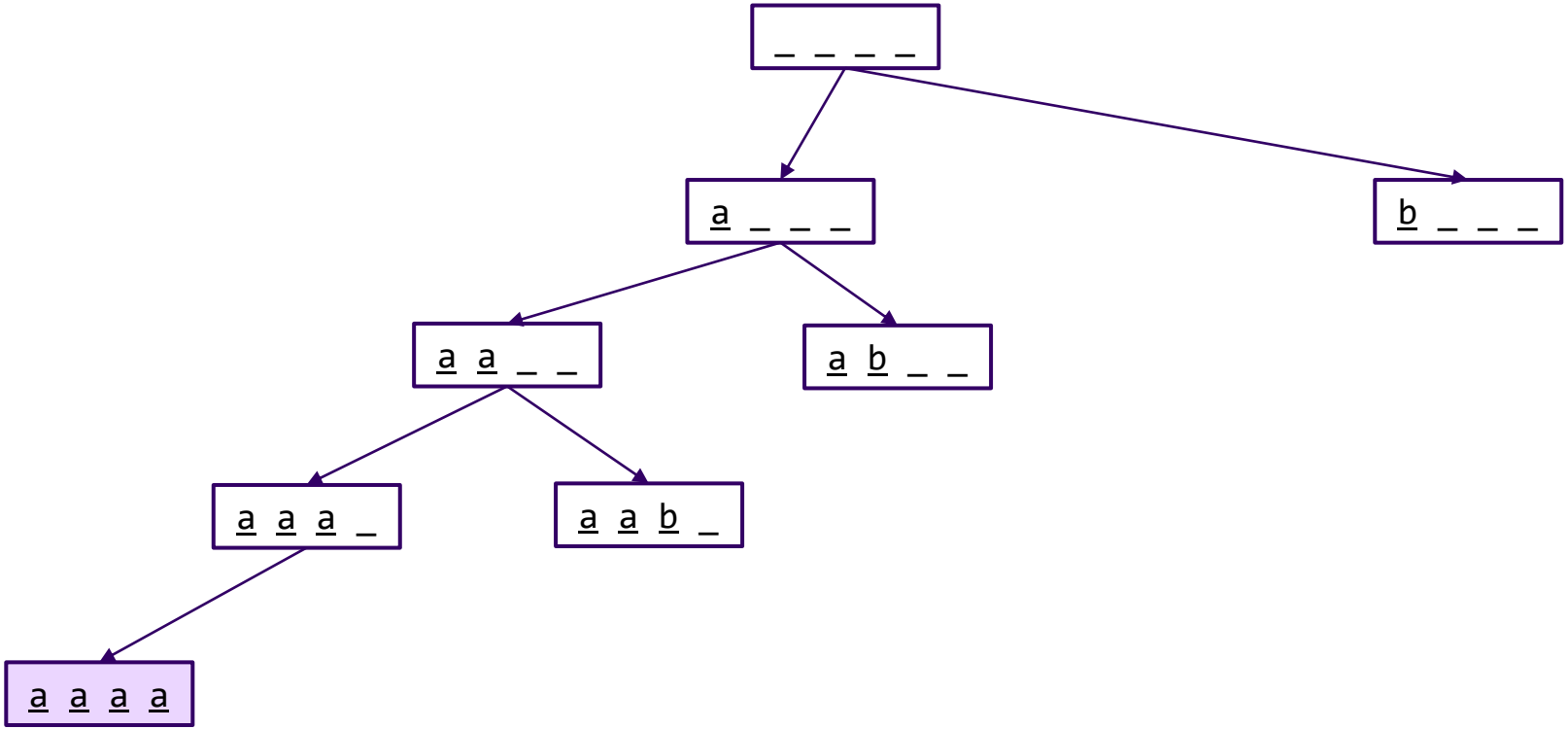
# fourAB Visualized



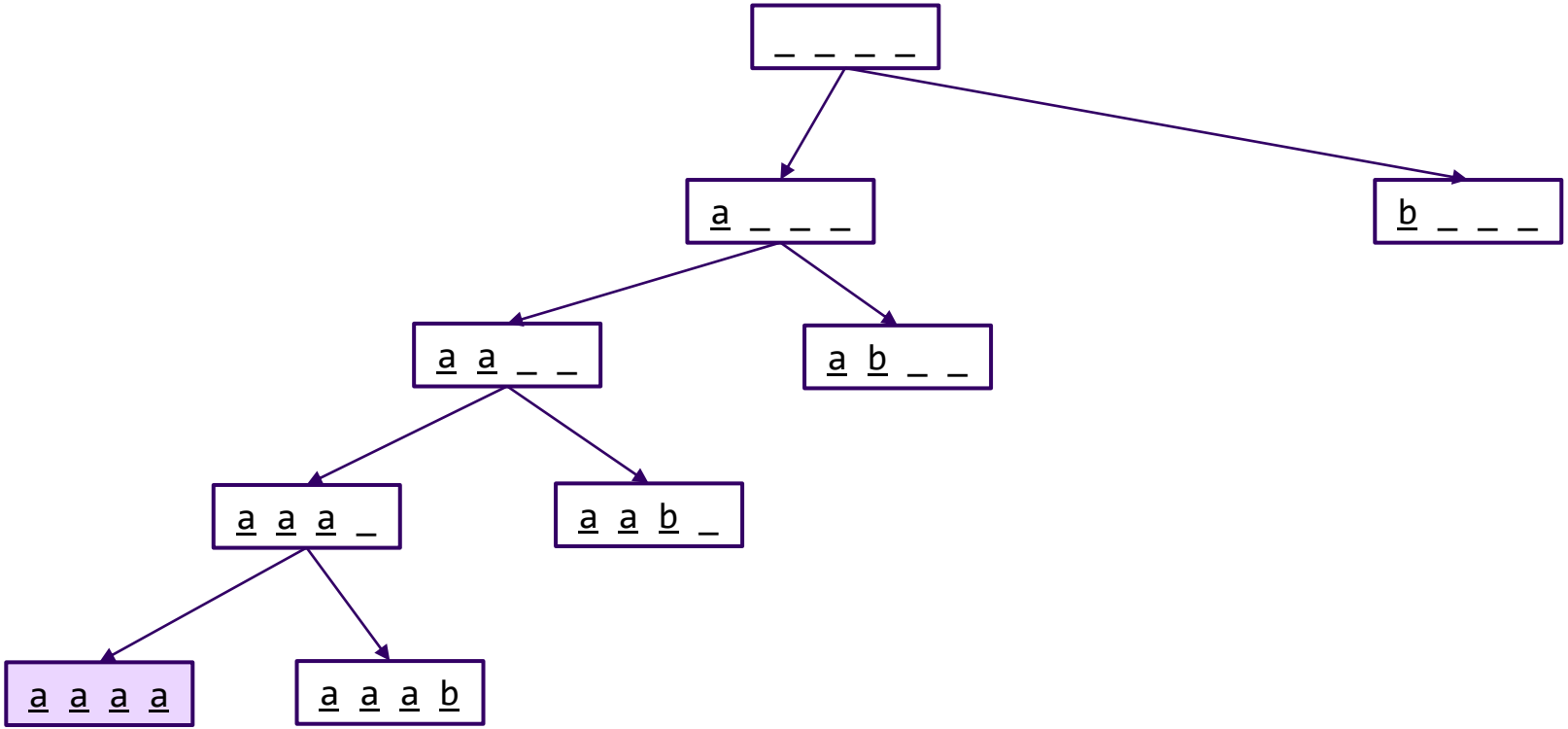
# fourAB Visualized



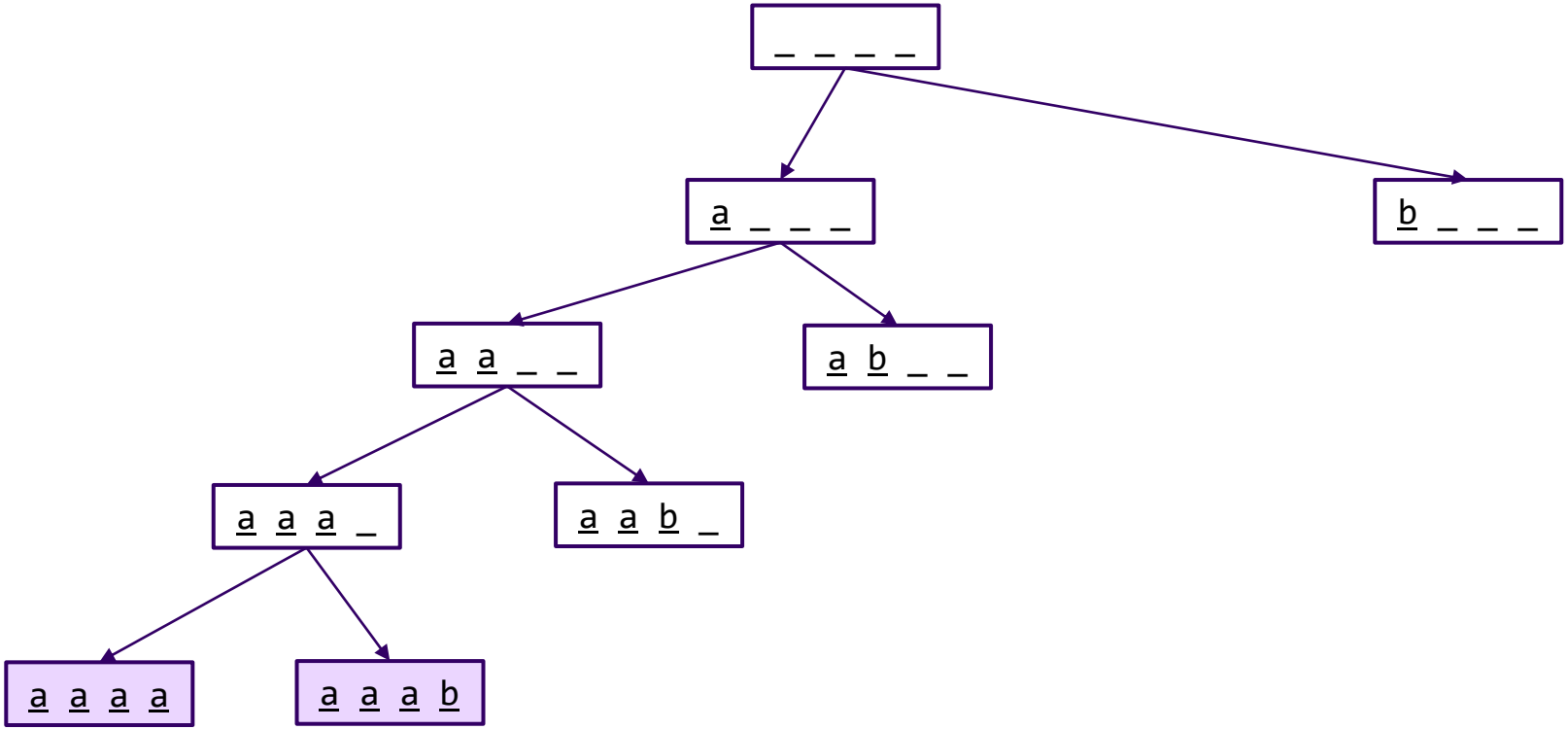
# fourAB Visualized



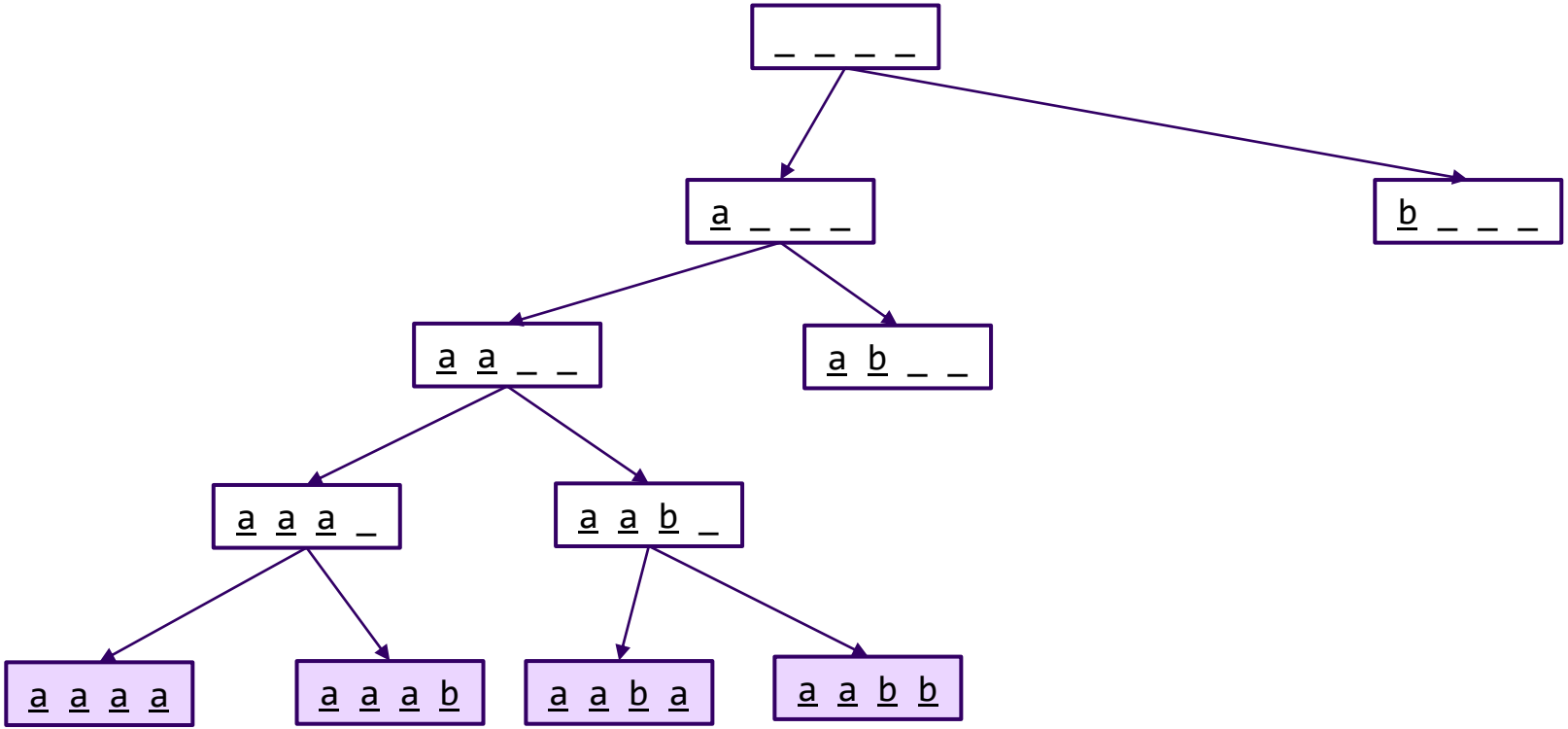
# fourAB Visualized



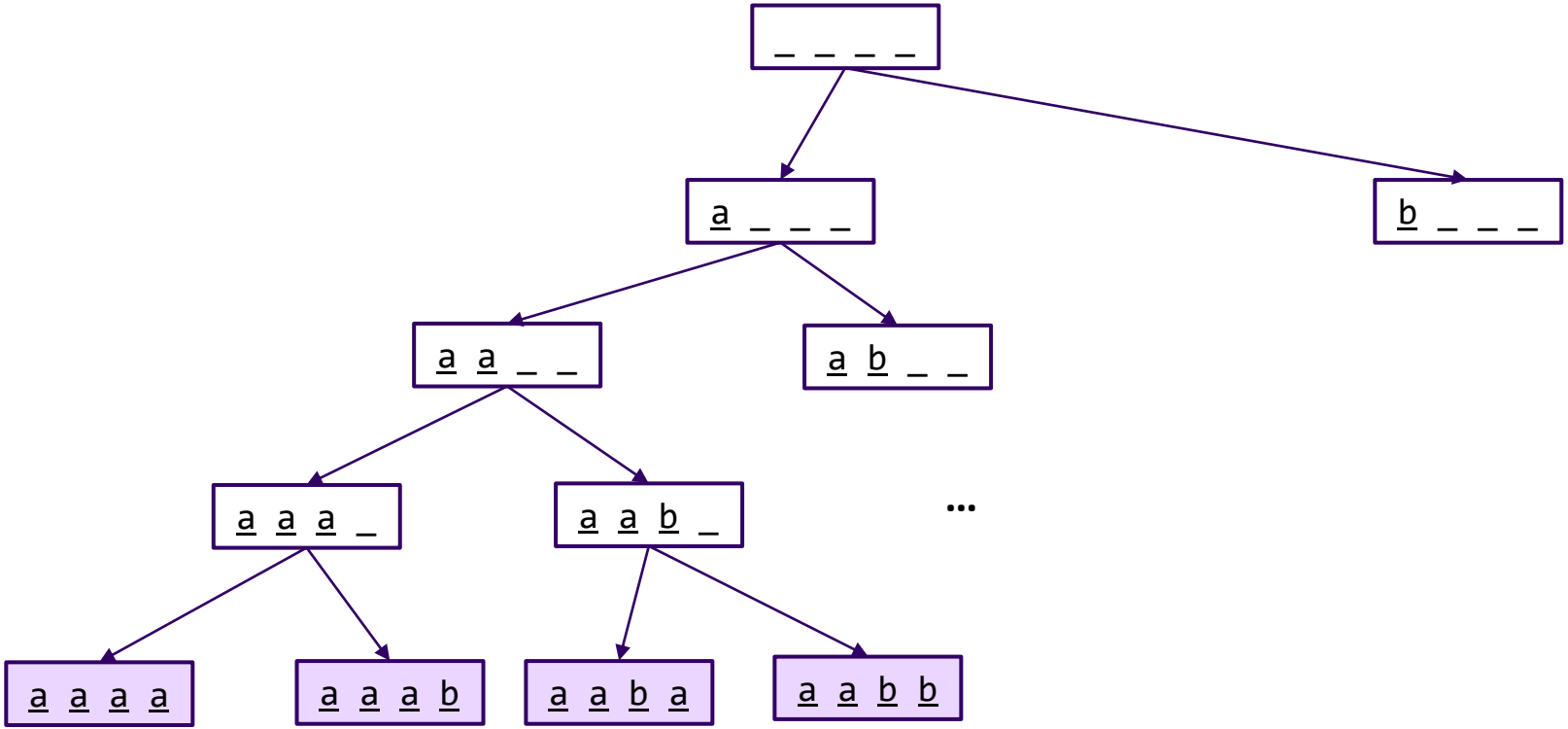
# fourAB Visualized



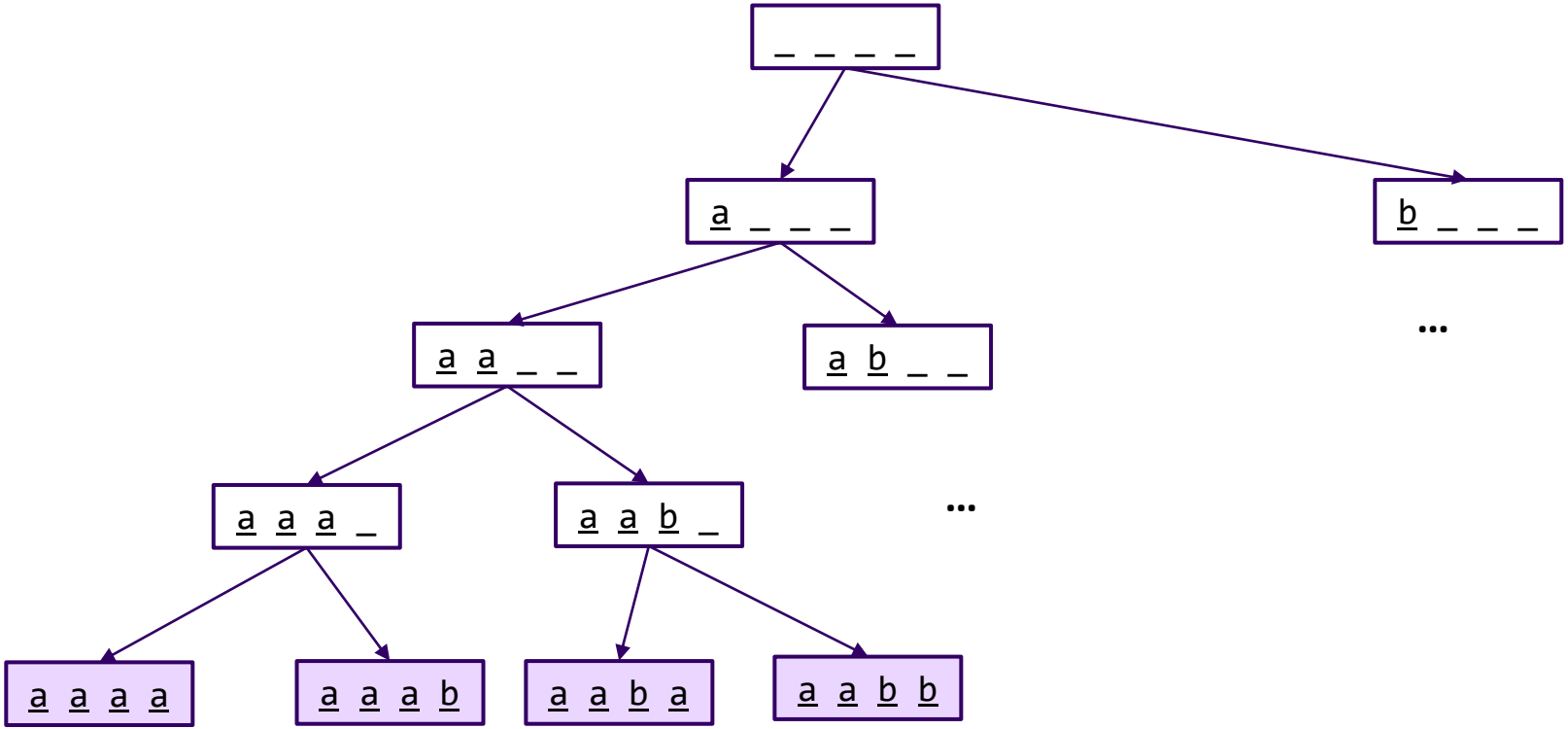
# fourAB Visualized



# fourAB Visualized



# fourAB Visualized



# fourAB Execution Visualized

```
private static void fourAB(String soFar) {
    if (soFar.length() == 4) {
        System.out.println(soFar);
    } else {
        fourAB(soFar + "a");
        fourAB(soFar + "b");
    }
}
```

| soFar | |
|---|---|

Console output:

# fourAB Execution Visualized

```
private static void fourAB(String soFar) {
    if (soFar.length() == 4) {
        System.out.println(soFar);
    } else {
        fourAB(soFar + "a");
        fourAB(soFar + "b");
    }
}
```

| soFar | |
|---|---|

Console output:

# fourAB Execution Visualized

```
private static void fourAB(String soFar) {
    if (soFar.length() == 4) {
        System.out.println(soFar);
    } else {
        fourAB(soFar + "a");
        fourAB(soFar + "b");
    }
}
```

| soFar | a |
|---|---|

Console output:

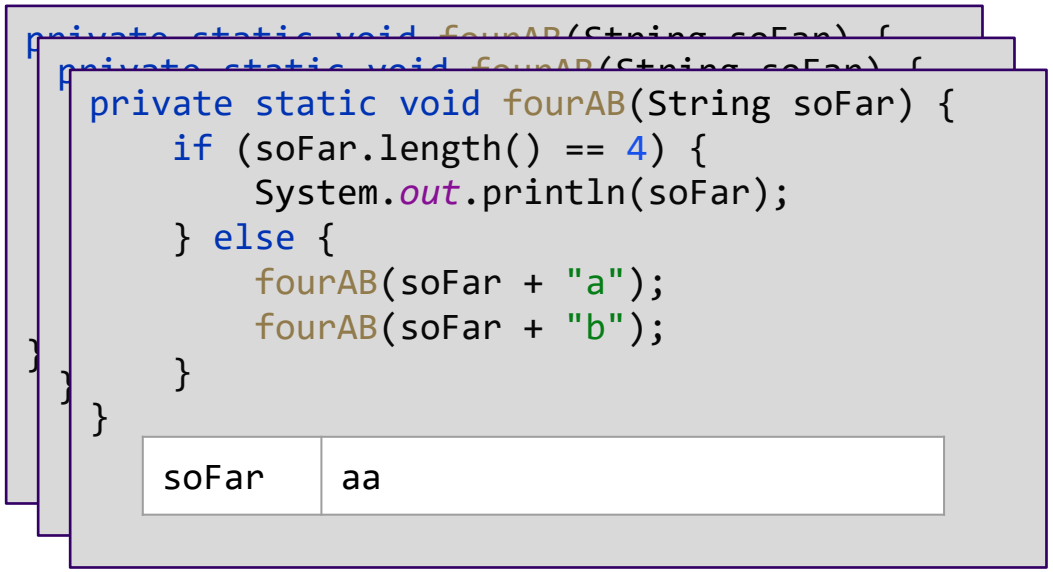
# fourAB Execution Visualized

```
private static void fourAB(String soFar) {
    if (soFar.length() == 4) {
        System.out.println(soFar);
    } else {
        fourAB(soFar + "a");
        fourAB(soFar + "b");
    }
}
```

| soFar | a |
|---|---|

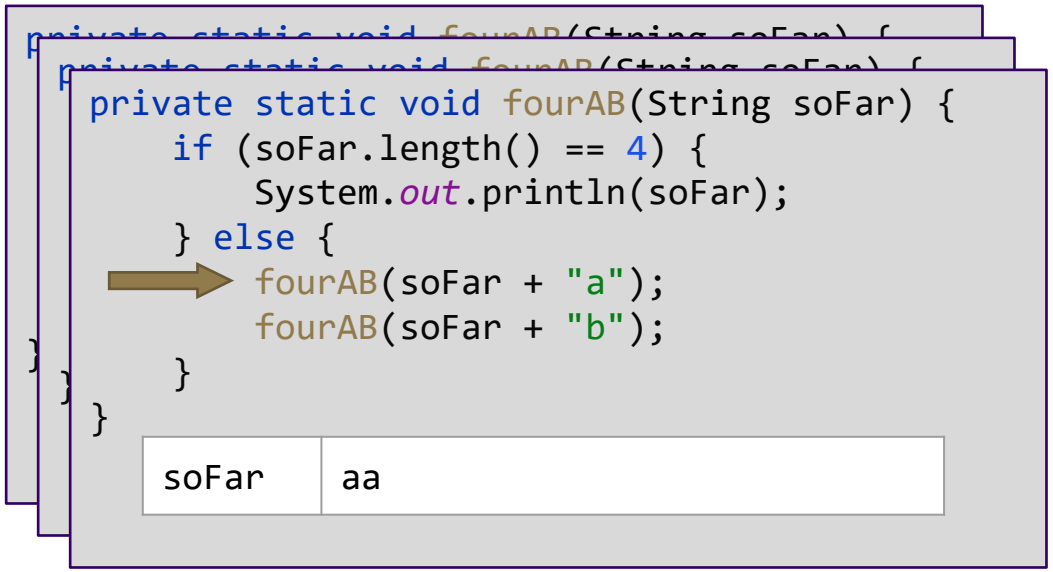
Console output:

# fourAB Execution Visualized



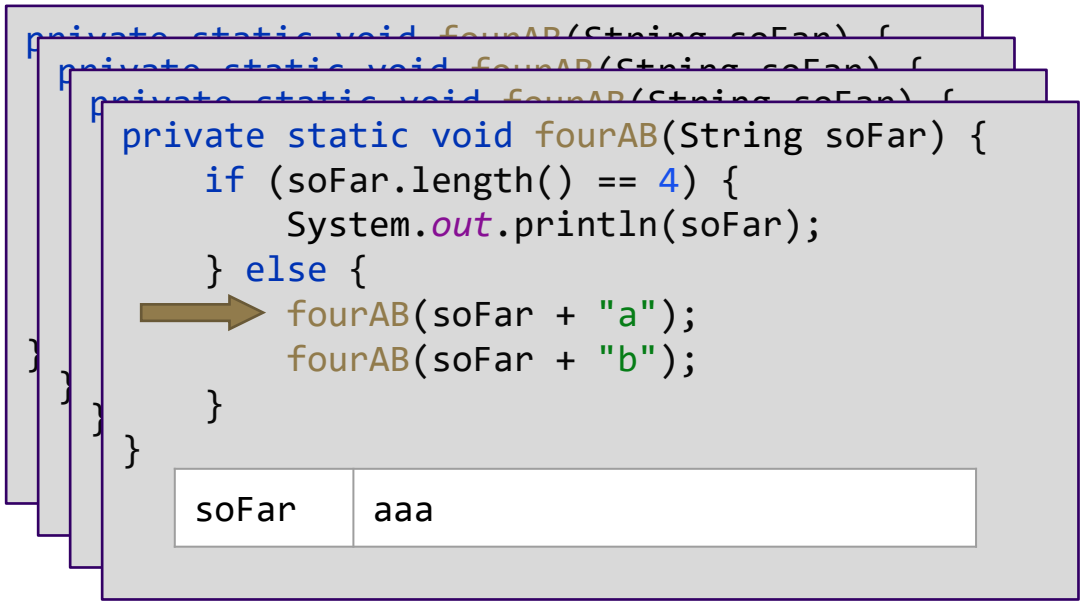

Console output:

# fourAB Execution Visualized



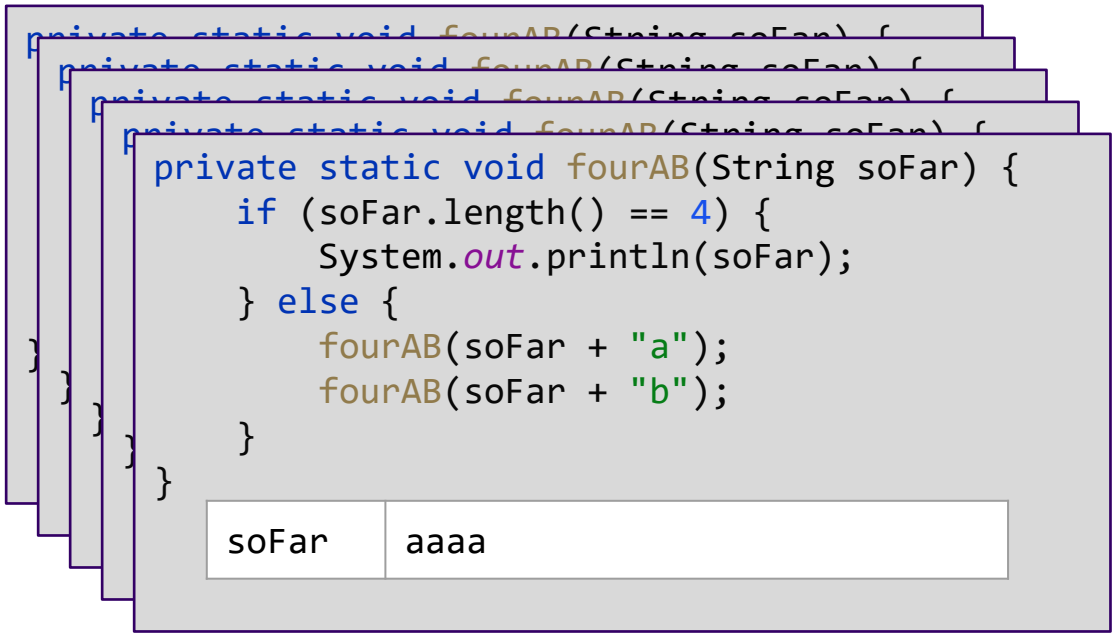

Console output:

# fourAB Execution Visualized



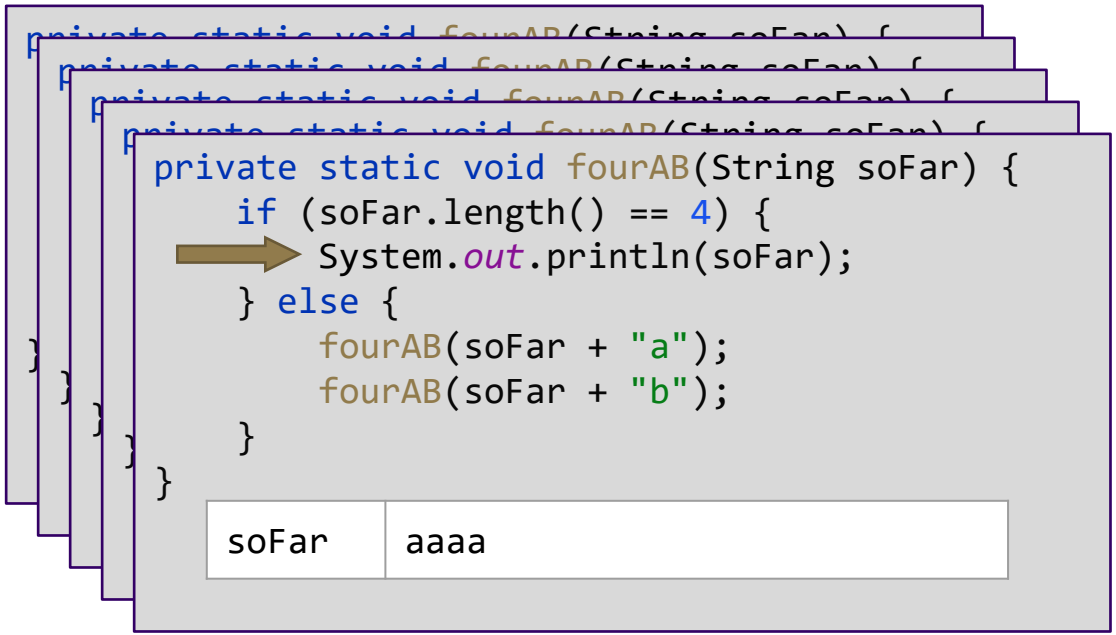

Console output:

# fourAB Execution Visualized



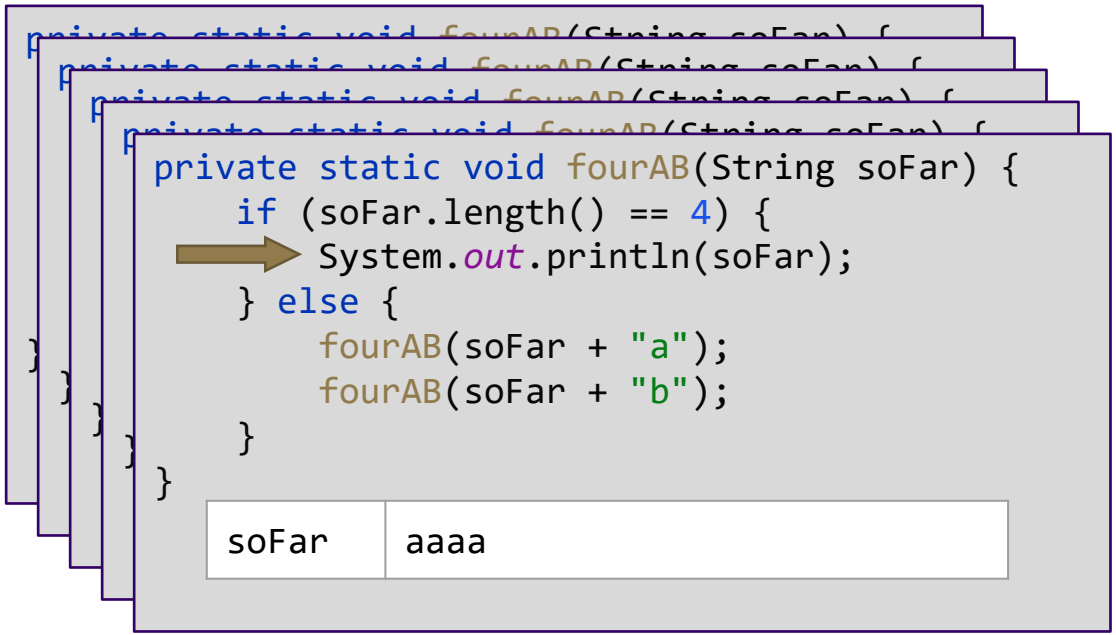

Console output:

# fourAB Execution Visualized



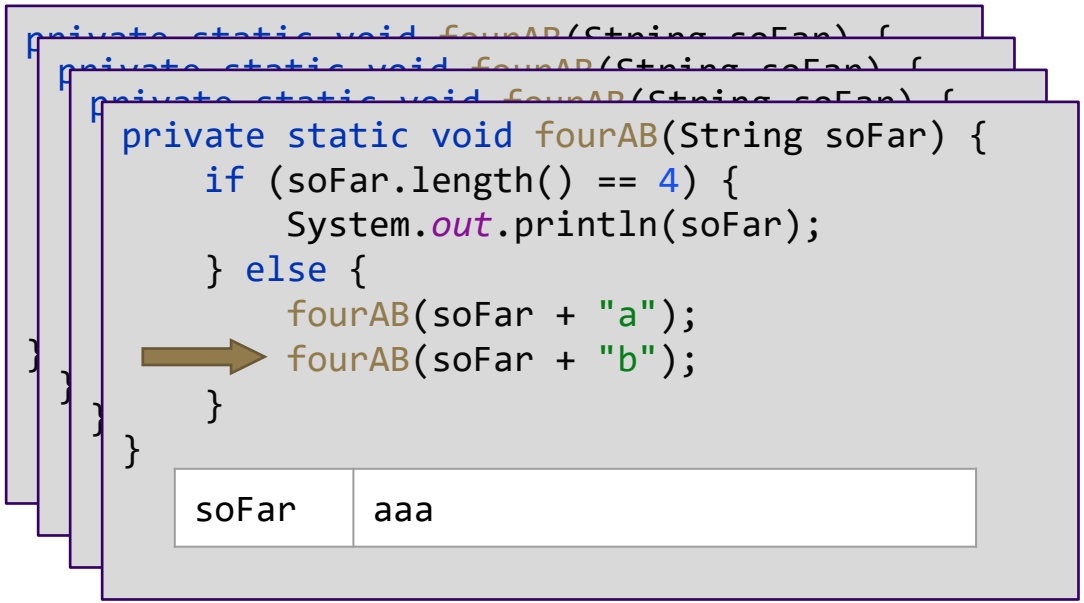

Console output:

# fourAB Execution Visualized



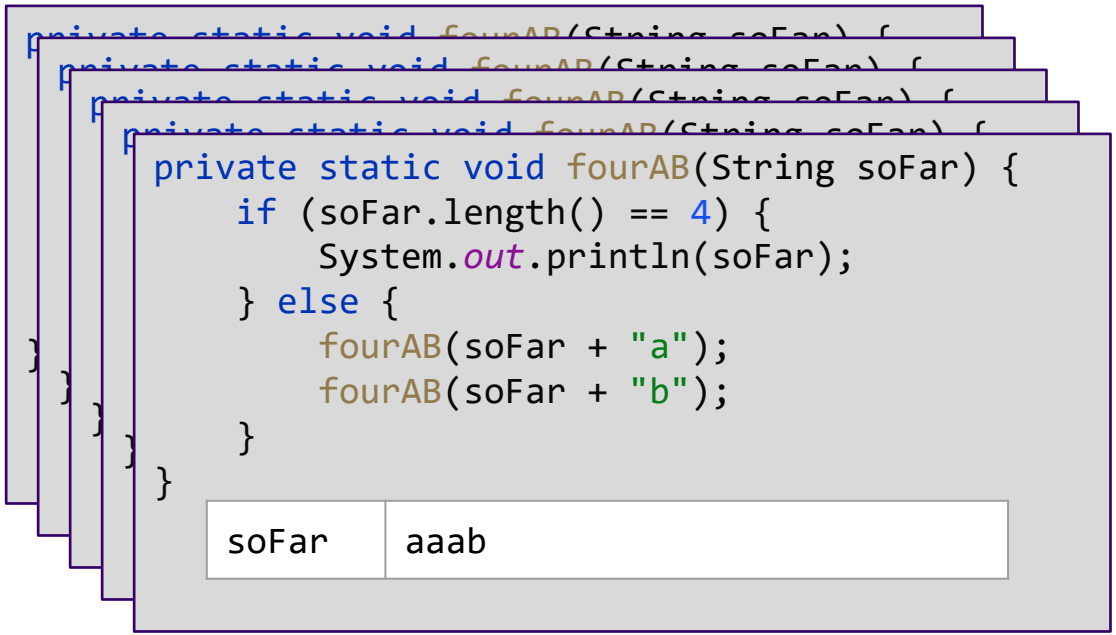

Console output:
aaaa

# fourAB Execution Visualized



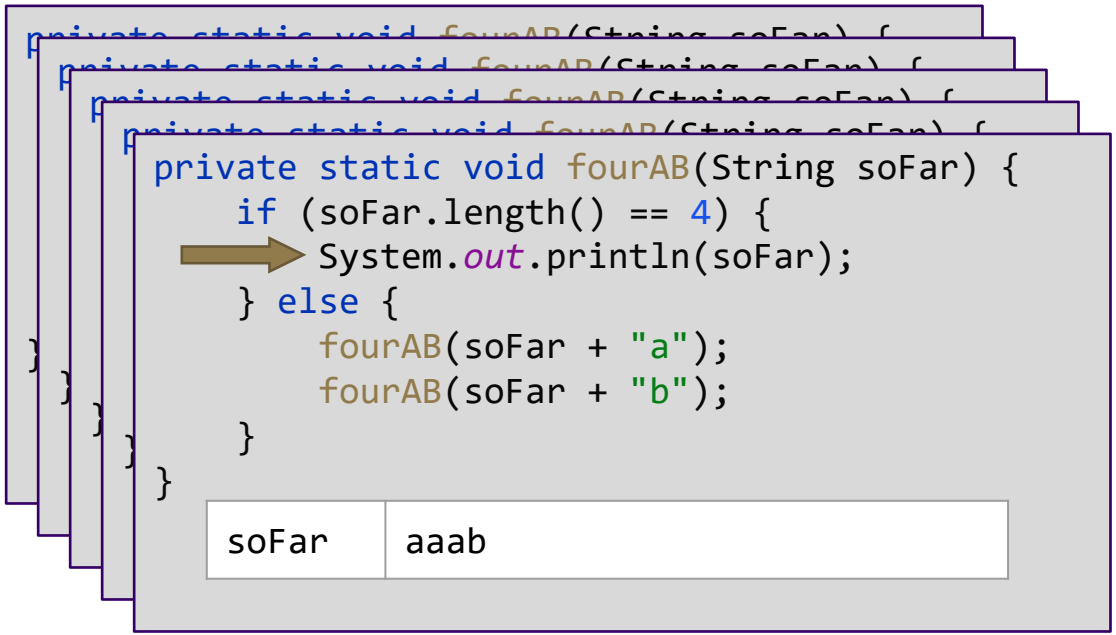

Console output:
aaaa

# fourAB Execution Visualized



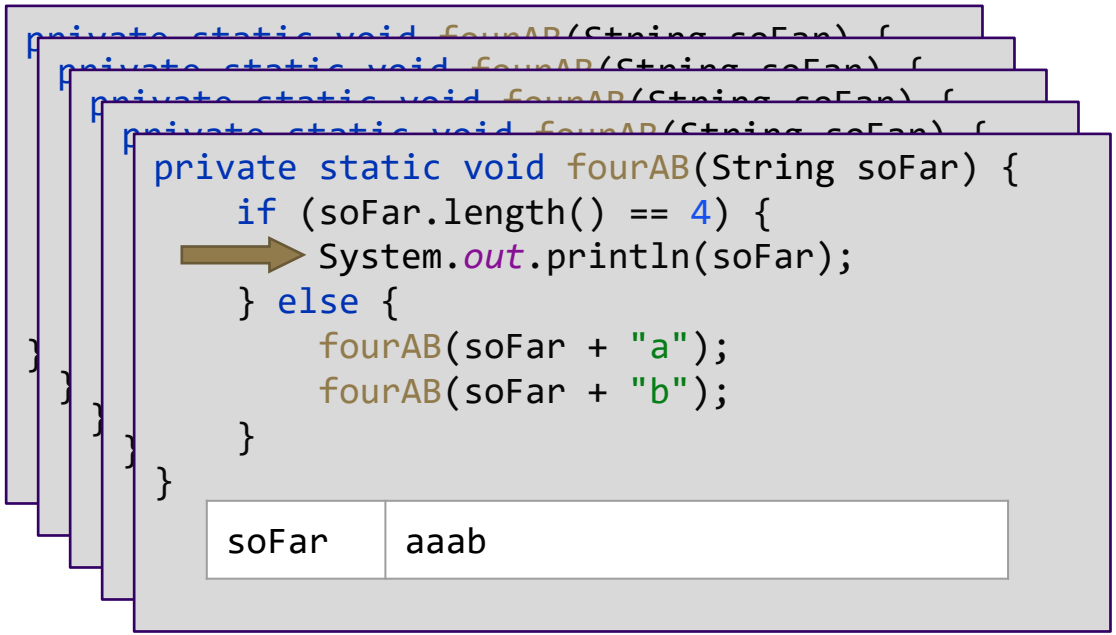

Console output:
aaaa

# fourAB Execution Visualized



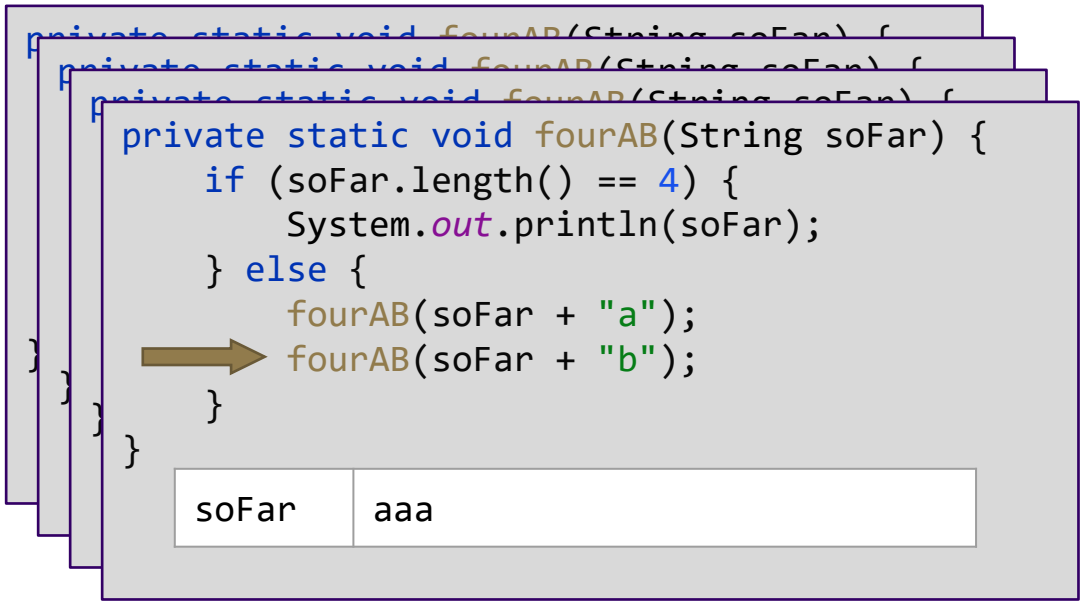

Console output:
aaaa

# fourAB Execution Visualized



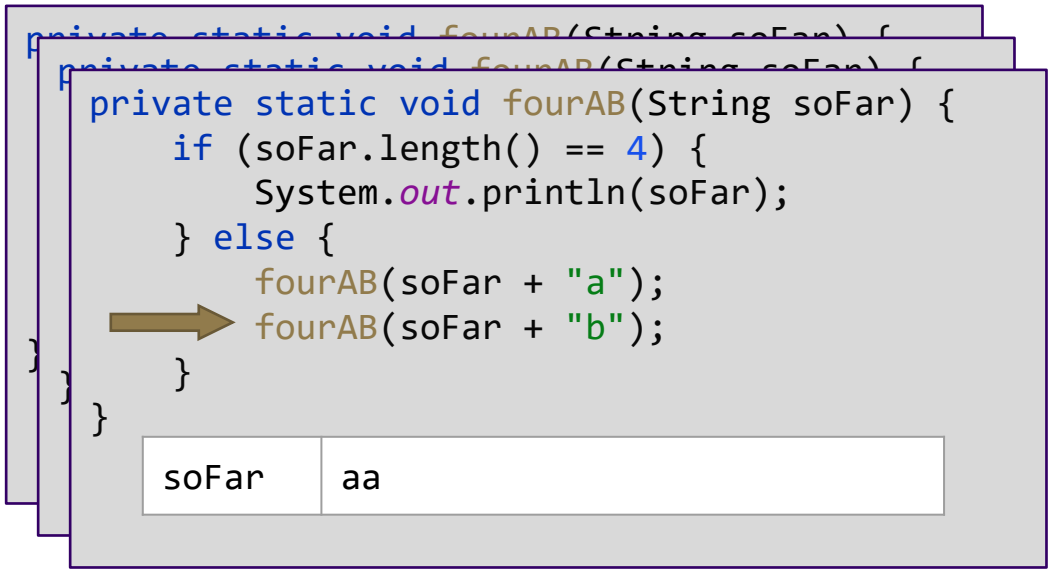

Console output:
aaaa
aaab

# fourAB Execution Visualized



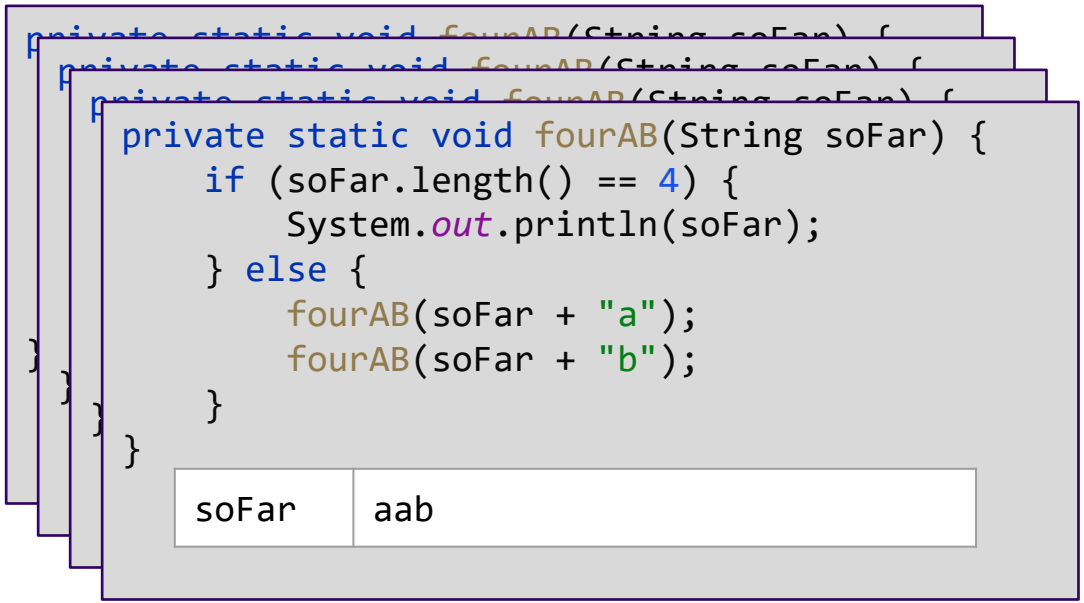

Console output:
aaaa
aaab

# fourAB Execution Visualized



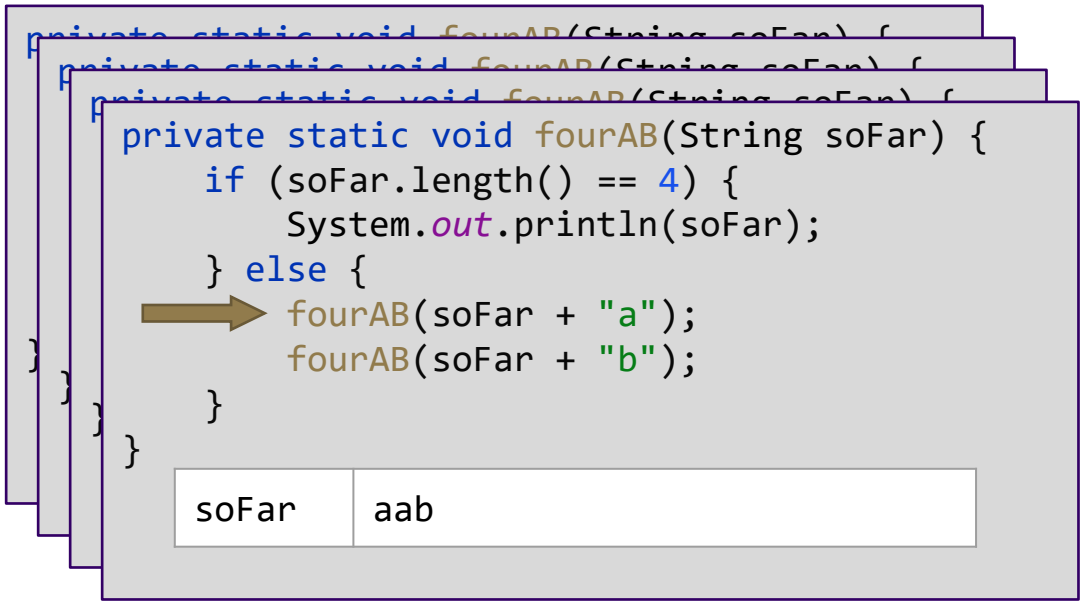

Console output:
aaaa
aaab

# fourAB Execution Visualized



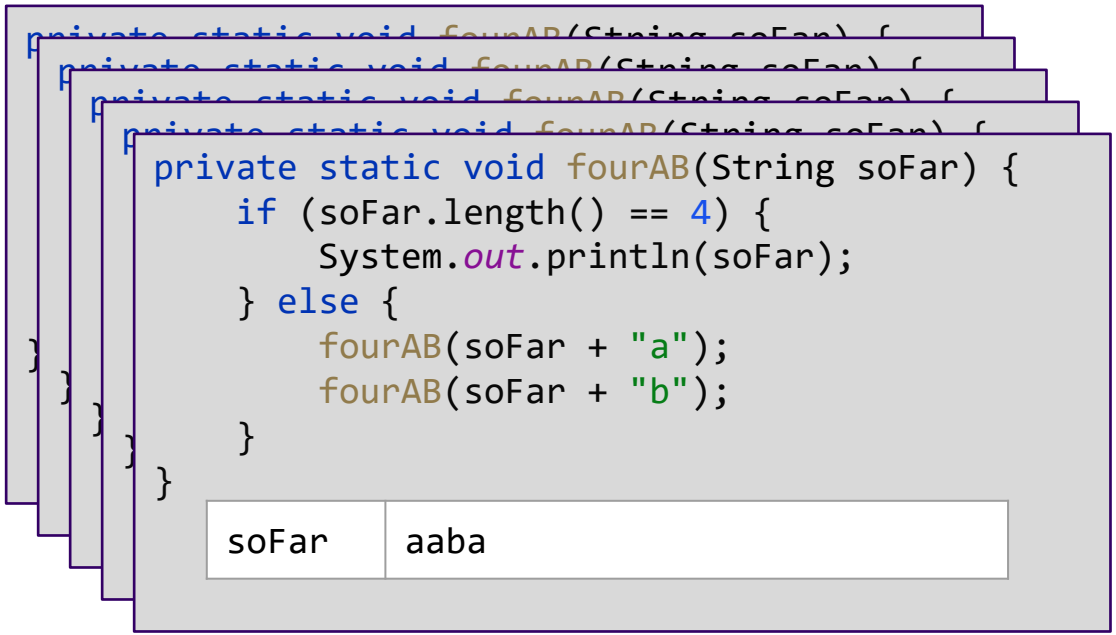

Console output:
aaaa
aaab

# fourAB Execution Visualized



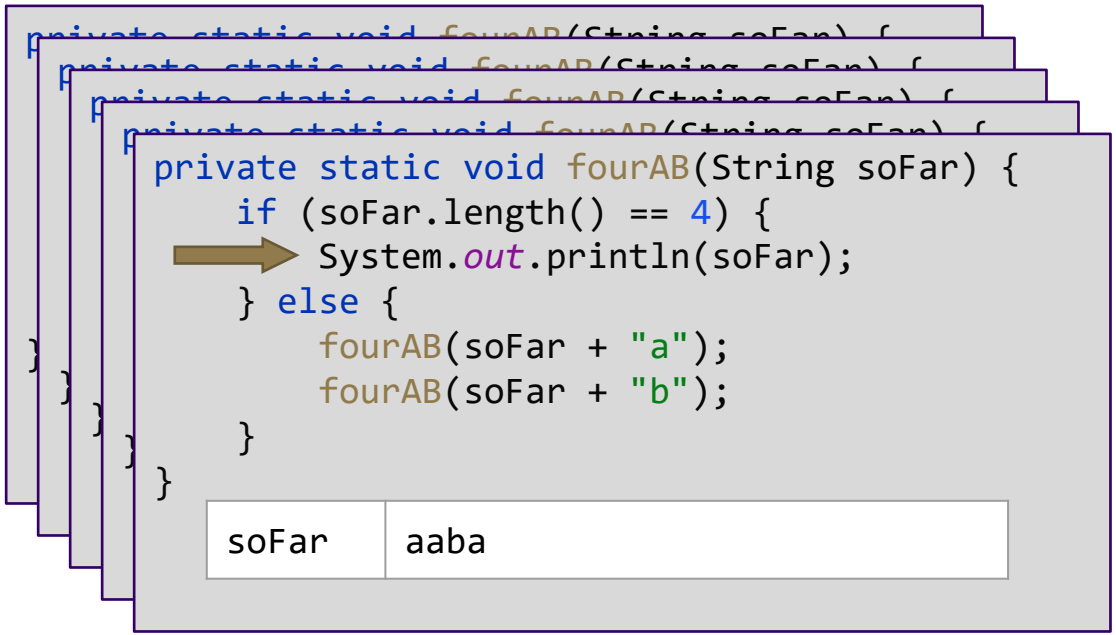

Console output:
aaaa
aaab

# fourAB Execution Visualized



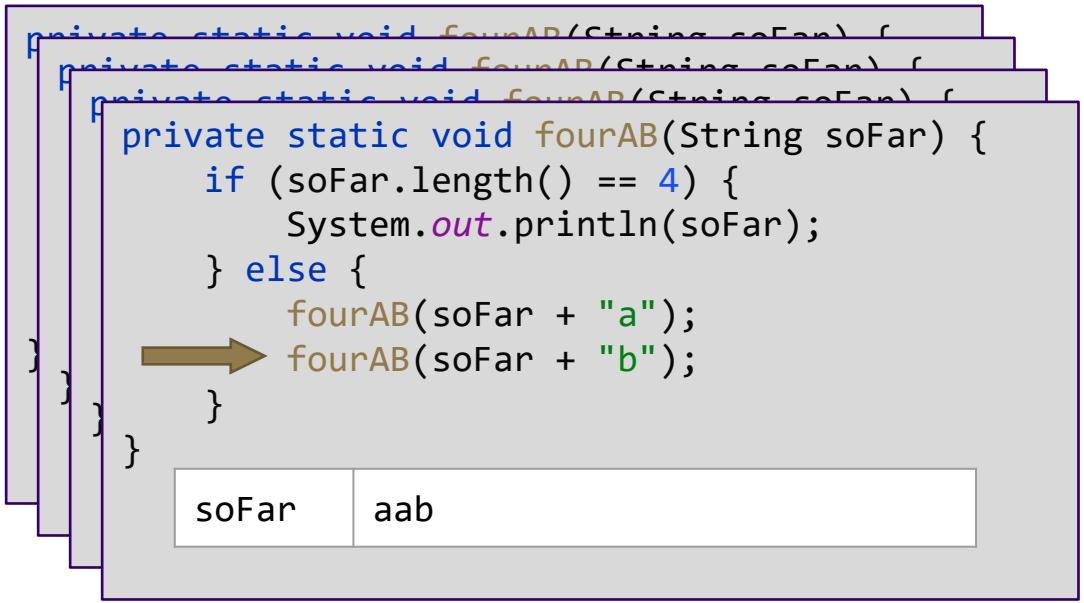

Console output:
aaaa
aaab

# fourAB Execution Visualized



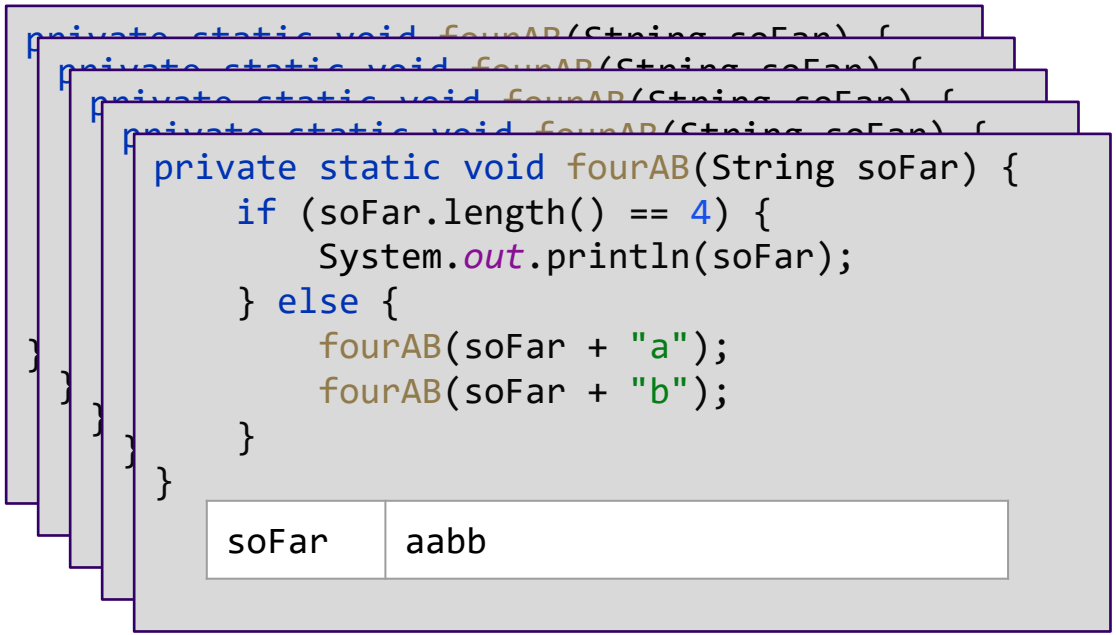

Console output:
aaaa
aaab
aaba

# fourAB Execution Visualized



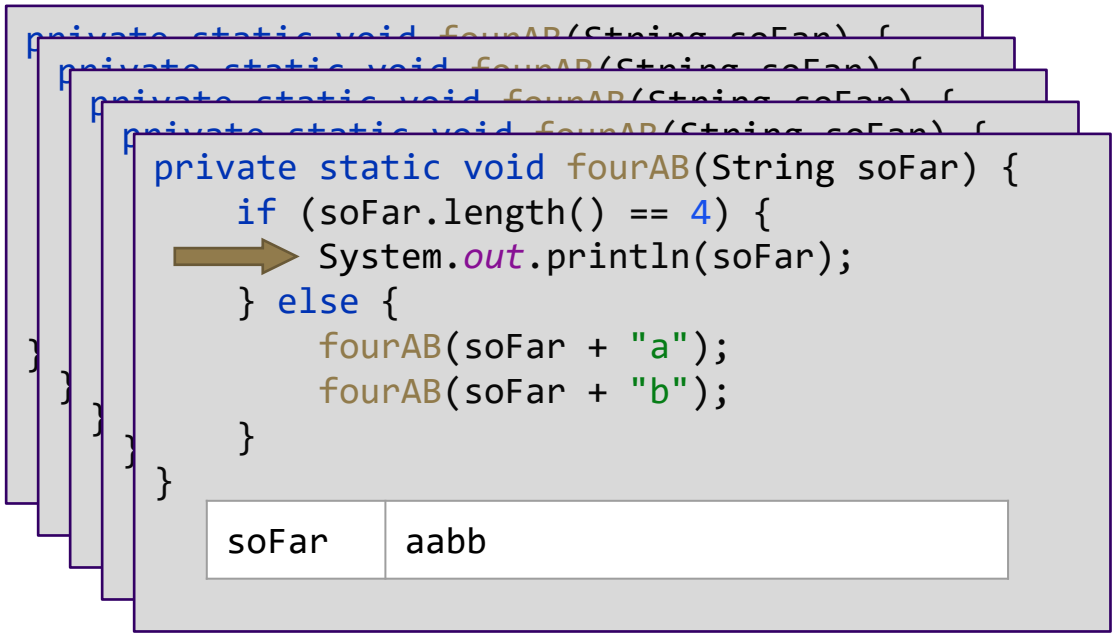

Console output:
aaaa
aaab
aaba

# fourAB Execution Visualized



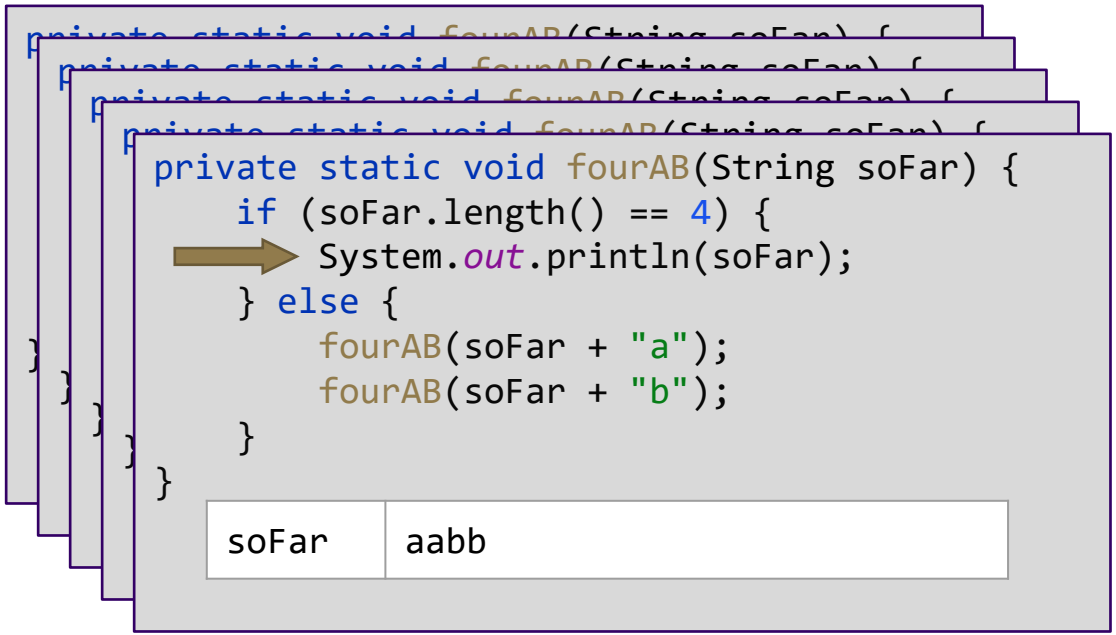

Console output:
aaaa
aaab
aaba

# fourAB Execution Visualized



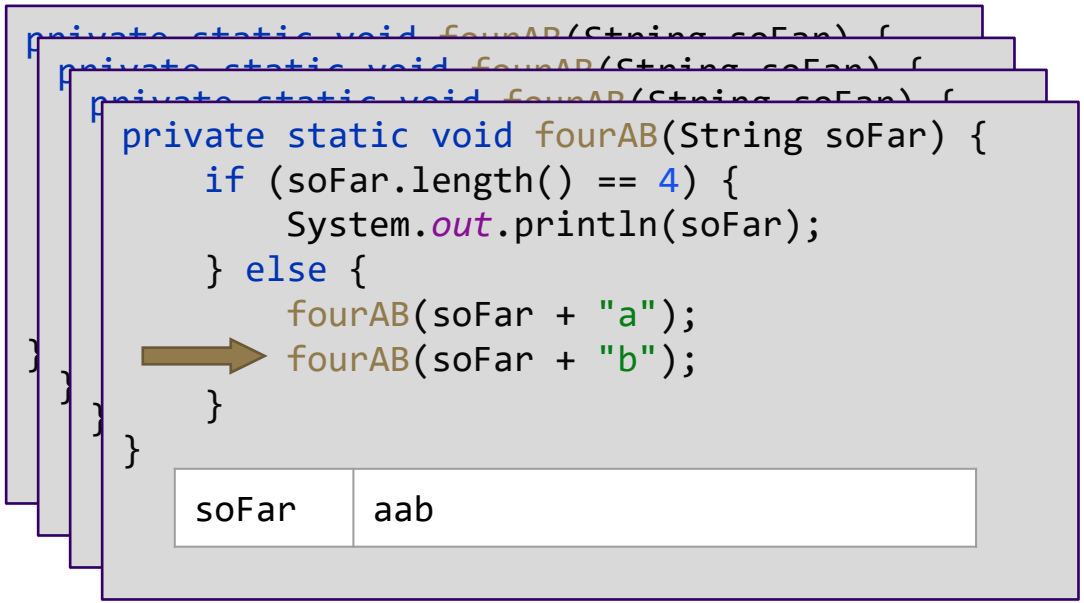

Console output:
aaaa
aaab
aaba

# fourAB Execution Visualized



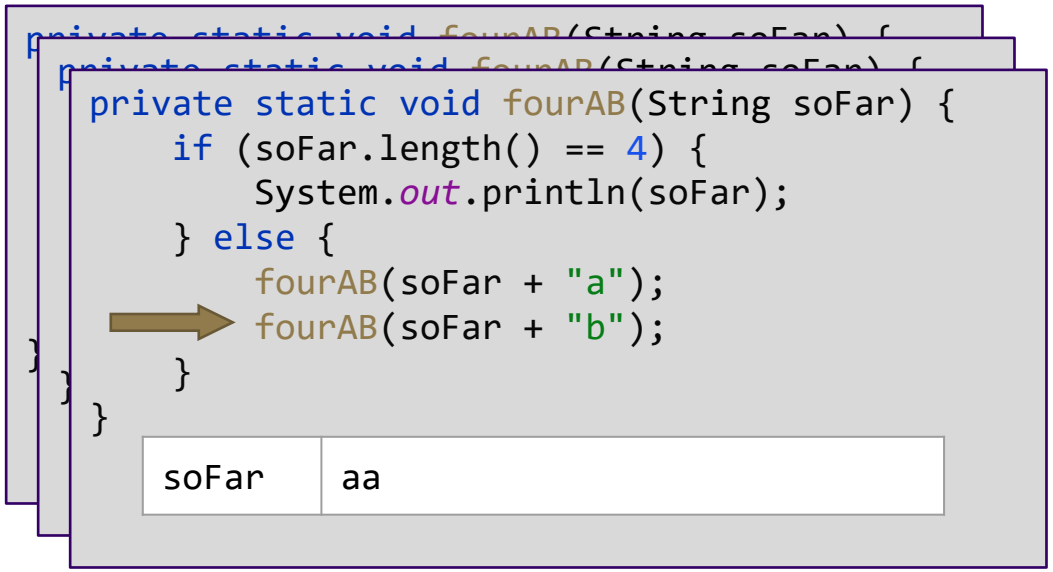

Console output:
aaaa
aaab
aaba
aabb

# fourAB Execution Visualized




Console output:
aaaa
aaab
aaba
aabb

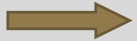
# fourAB Execution Visualized




Console output:
aaaa
aaab
aaba
aabb

# fourAB Execution Visualized

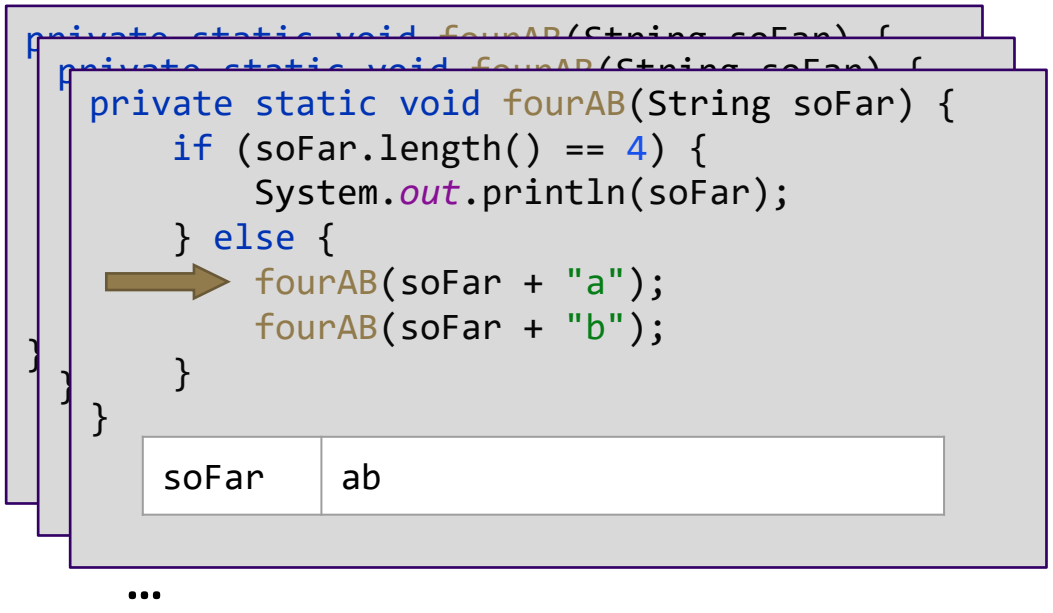
```
private static void fourAB(String soFar) {
    if (soFar.length() == 4) {
        System.out.println(soFar);
    } else {
        fourAB(soFar + "a");
        fourAB(soFar + "b");
    }
}
```



| soFar | a |
|---|---|

Console output:
aaaa
aaab
aaba
aabb

# fourAB Execution Visualized




Console output:
aaaa
aaab
aaba
aabb

# Exhaustive Search

- Brute force algorithm!
  - Not **smart**
- Simply trying all possibilities, find the ones that work
- Often solved recursively

# Exhaustive Search

- Brute force algorithm!
  - Not **smart**
- Simply trying all possibilities, find the ones that work
- Often solved recursively
- Two main cases
  - **Found Solution:** base case
  - **Continue Exploring**: recursive case
- Multiple recursive calls
  - One per option
- Public/Private pairs are essential

# Practice

slido.com
code: #su_cse123

```
public static void method() {
    method( soFar: "");
}
private static void method(String soFar) {
    if (soFar.length() == 3) {
        System.out.println(soFar);
    } else {
        method( soFar: soFar + "s");
        method( soFar: soFar + "o");
        method( soFar: soFar + "d");
    }
}
```

What's the fourth line of output produced by `method()`?

# Agenda

- Exhaustive Search
- Gambling 
- Phone Numbers

# Dice Rolls

- Write a method that prints out all possible outcomes when rolling some n 6-sided dice.

# Dice Rolls

- Write a method that prints out all possible outcomes when rolling some n 6-sided dice.

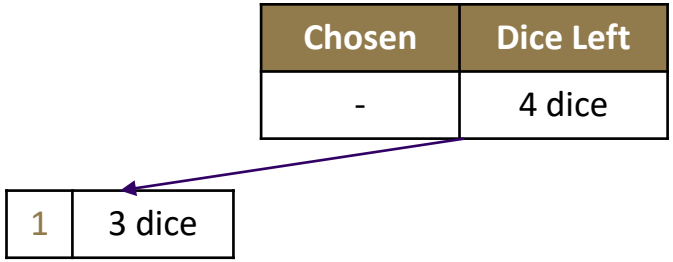
Example: `diceRoll(2)`

| | | |
|---|---|---|
| [1, 1] | [3, 1] | [5, 1] |
| [1, 2] | [3, 2] | [5, 2] |
| [1, 3] | [3, 3] | [5, 3] |
| [1, 4] | [3, 4] | [5, 4] |
| [1, 5] | [3, 5] | [5, 5] |
| [1, 6] | [3, 6] | [5, 6] |
| [2, 1] | [4, 1] | [6, 1] |
| [2, 2] | [4, 2] | [6, 2] |
| [2, 3] | [4, 3] | [6, 3] |
| [2, 4] | [4, 4] | [6, 4] |
| [2, 5] | [4, 5] | [6, 5] |
| [2, 6] | [4, 6] | [6, 6] |

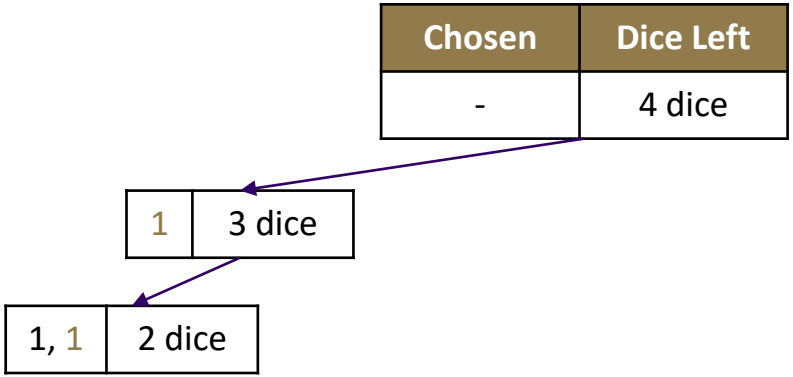
# `diceRolls` Decision Tree

| Chosen | Dice Left |
| --- | --- |
| - | 4 dice |

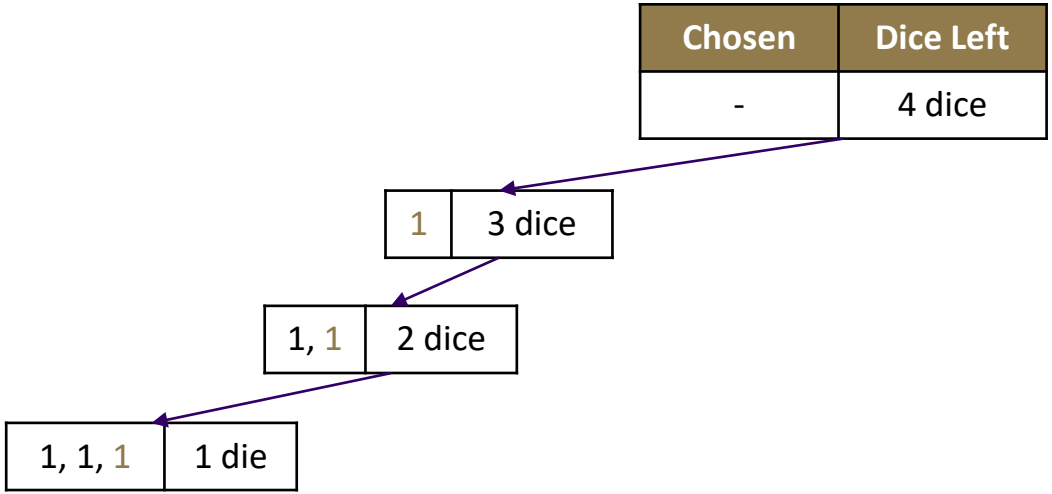
# `diceRolls` Decision Tree



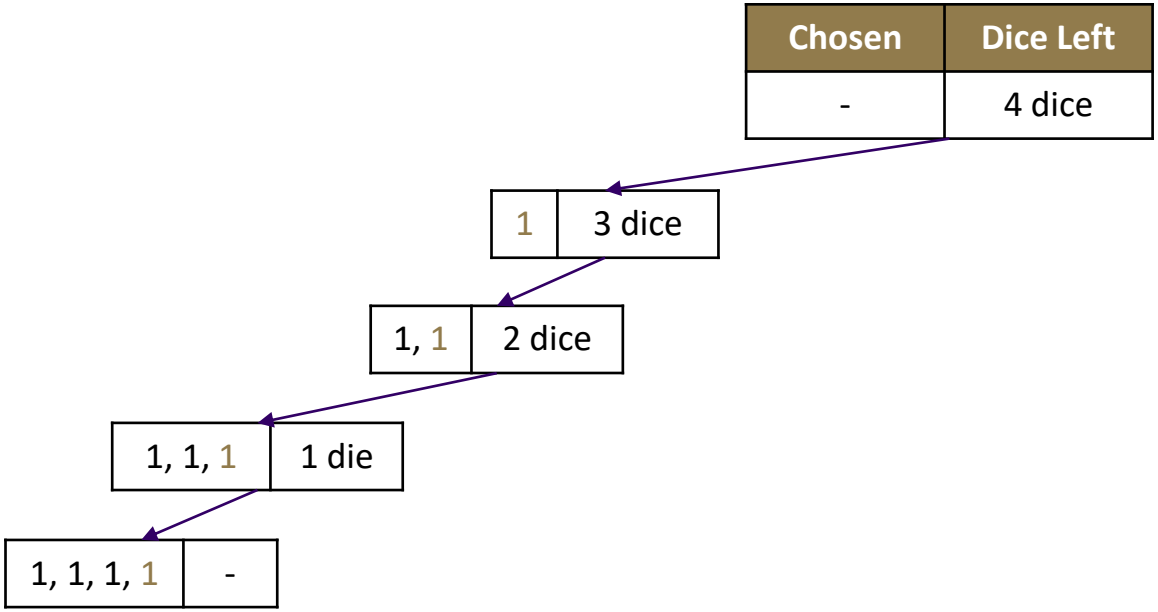
# `diceRolls` Decision Tree



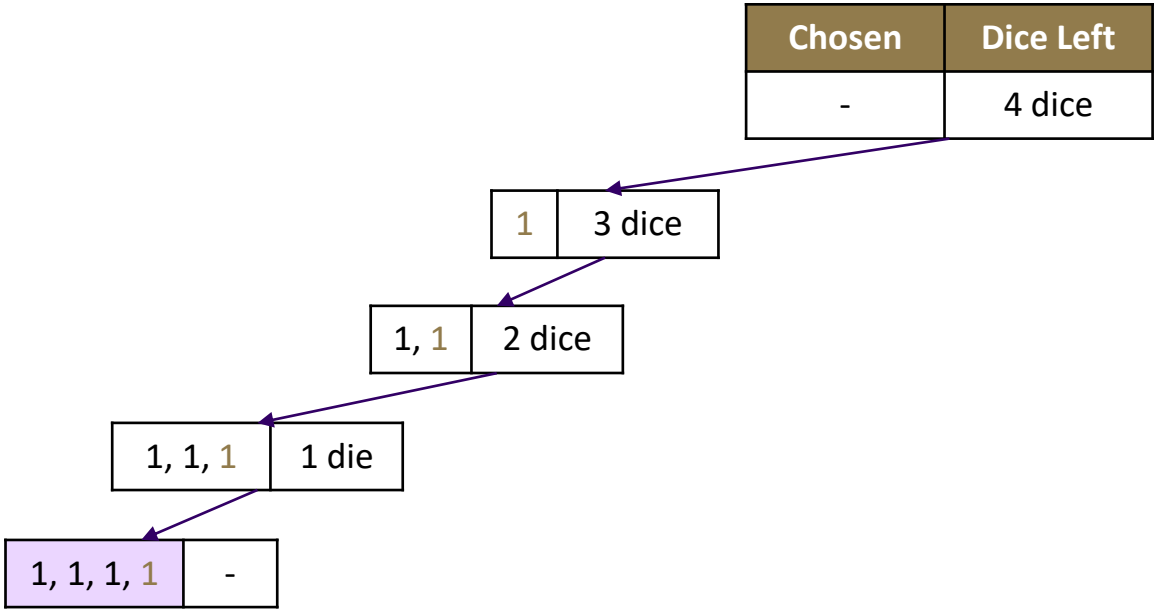
# `diceRolls` Decision Tree



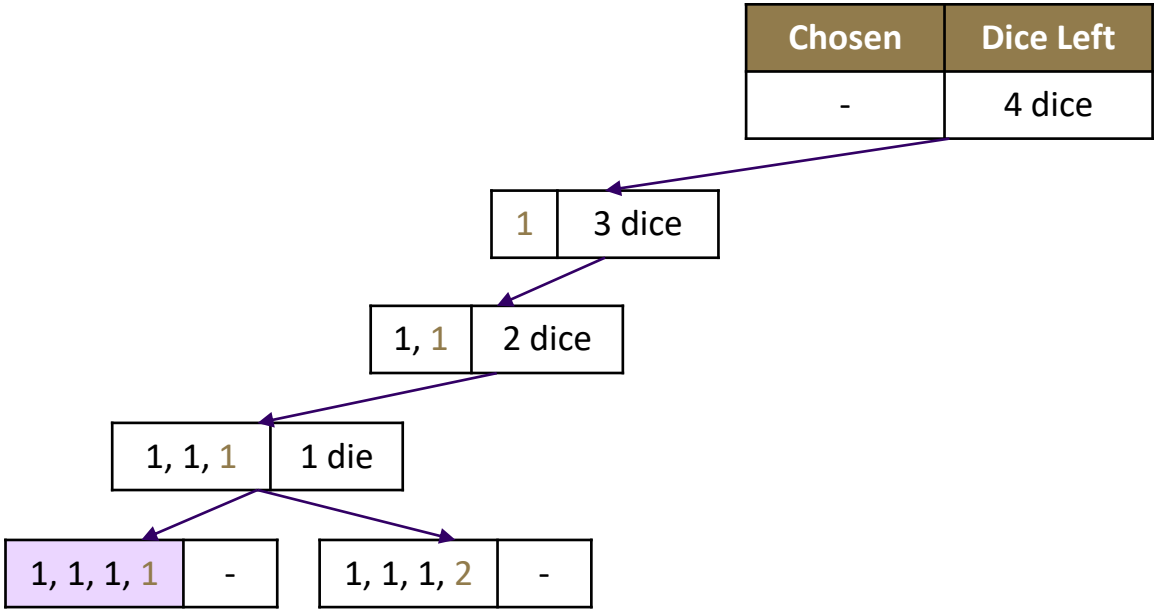
# `diceRolls` Decision Tree



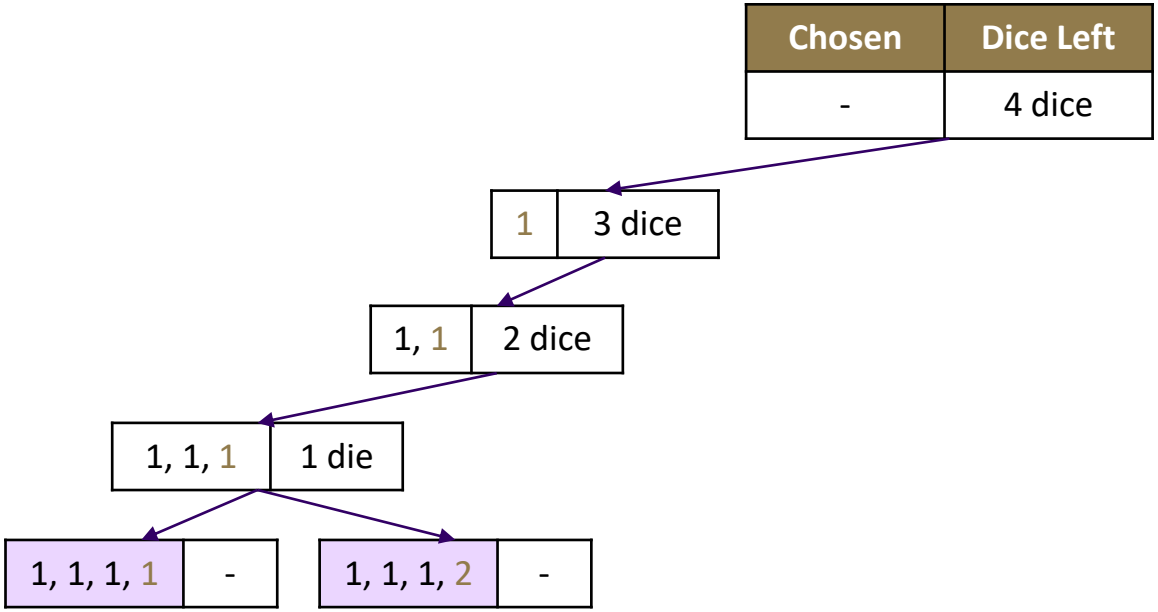
# `diceRolls` Decision Tree



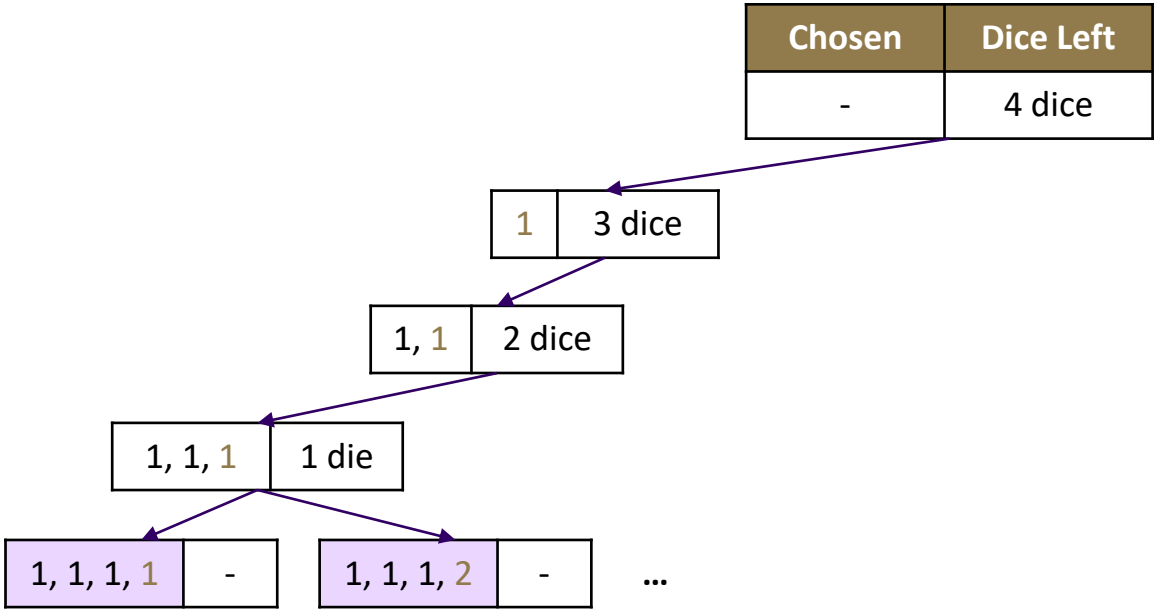
# `diceRolls` Decision Tree



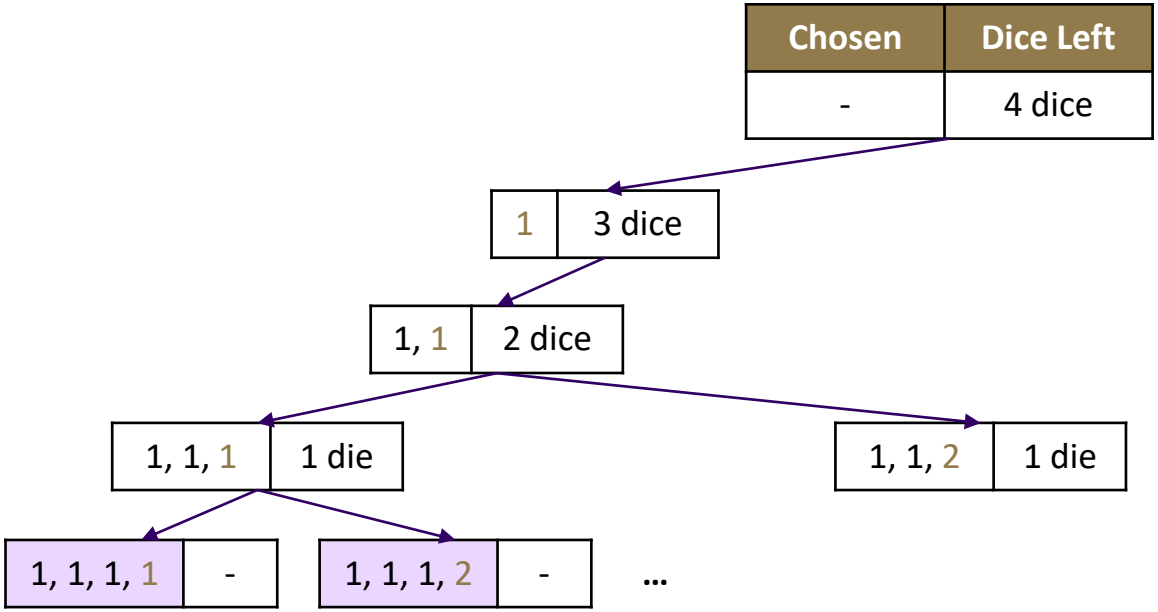
# `diceRolls` Decision Tree



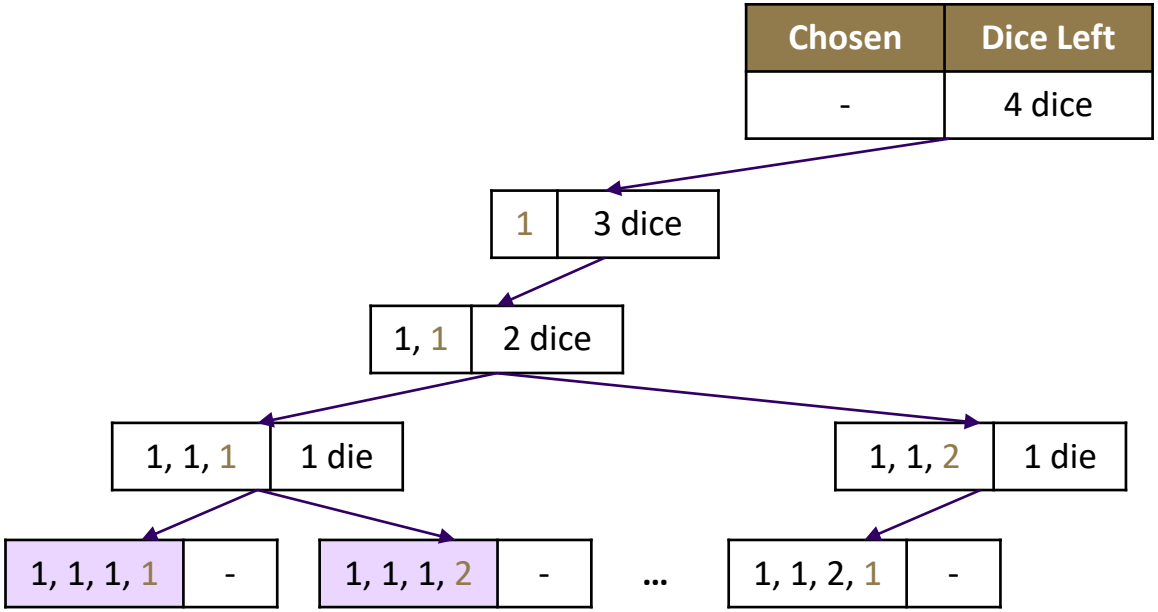
# `diceRolls` Decision Tree



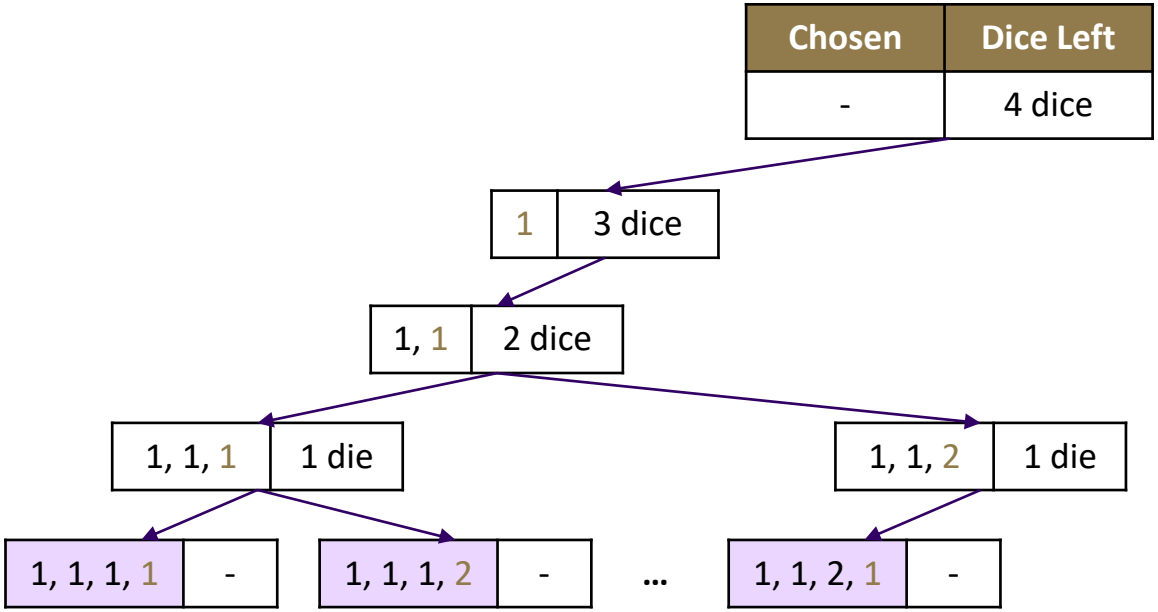
# `diceRolls` Decision Tree



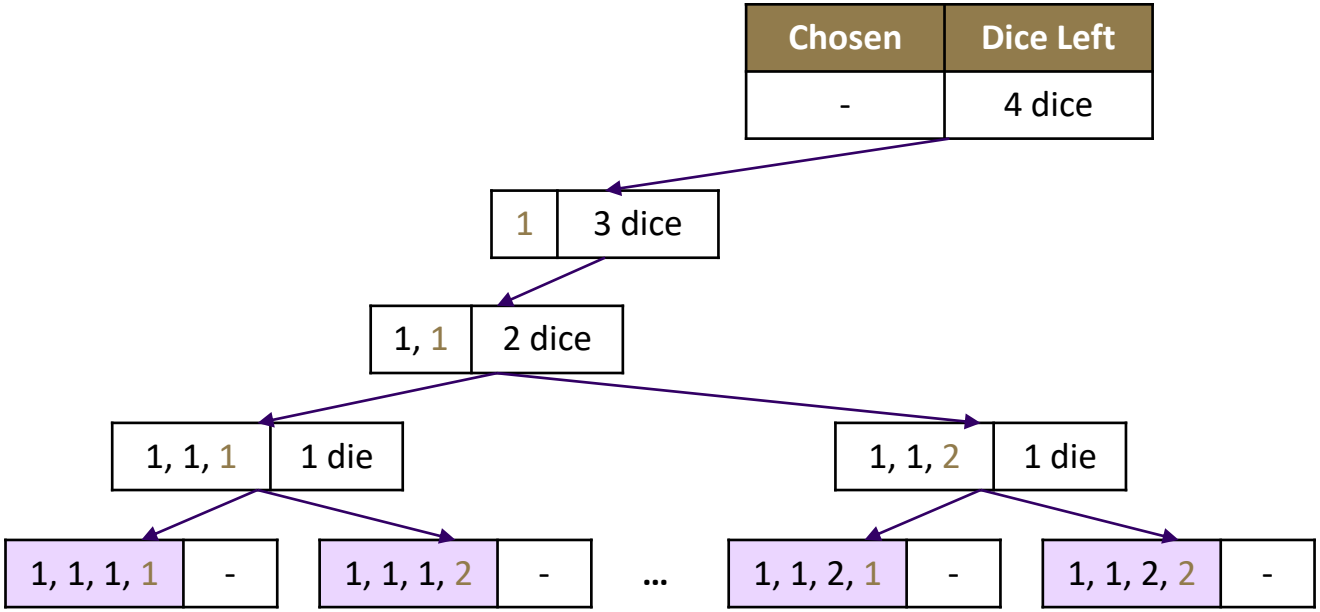
# `diceRolls` Decision Tree



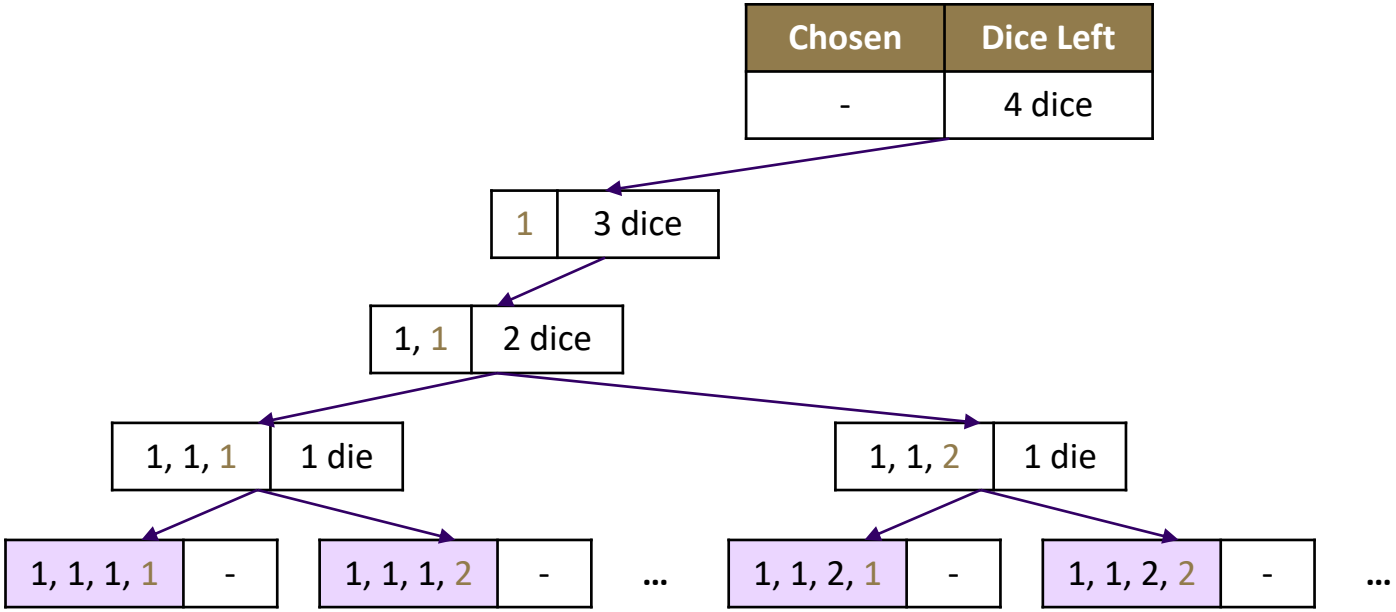
# `diceRolls` Decision Tree



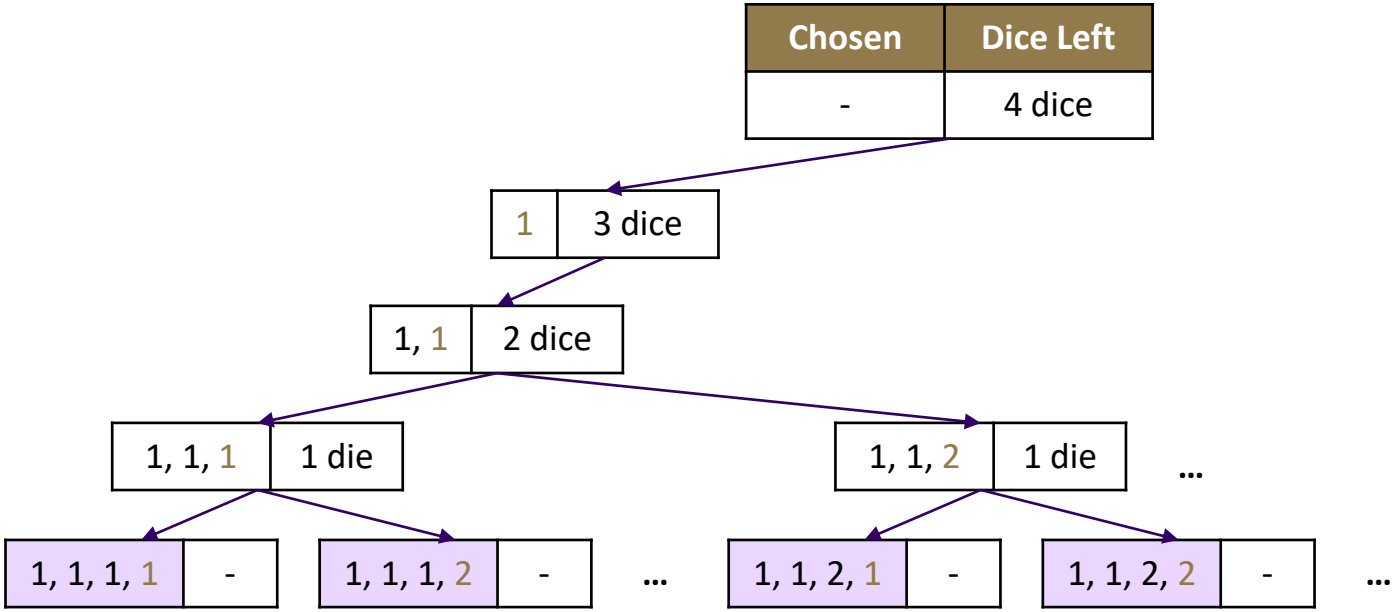
# `diceRolls` Decision Tree



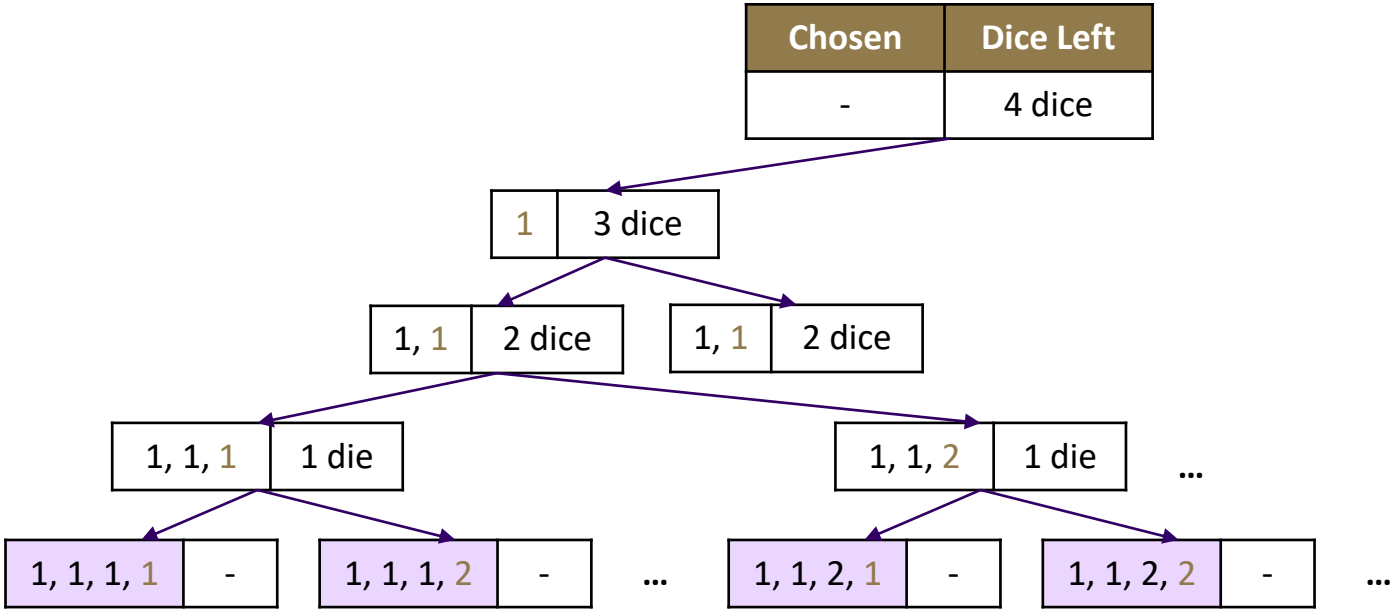
# `diceRolls` Decision Tree



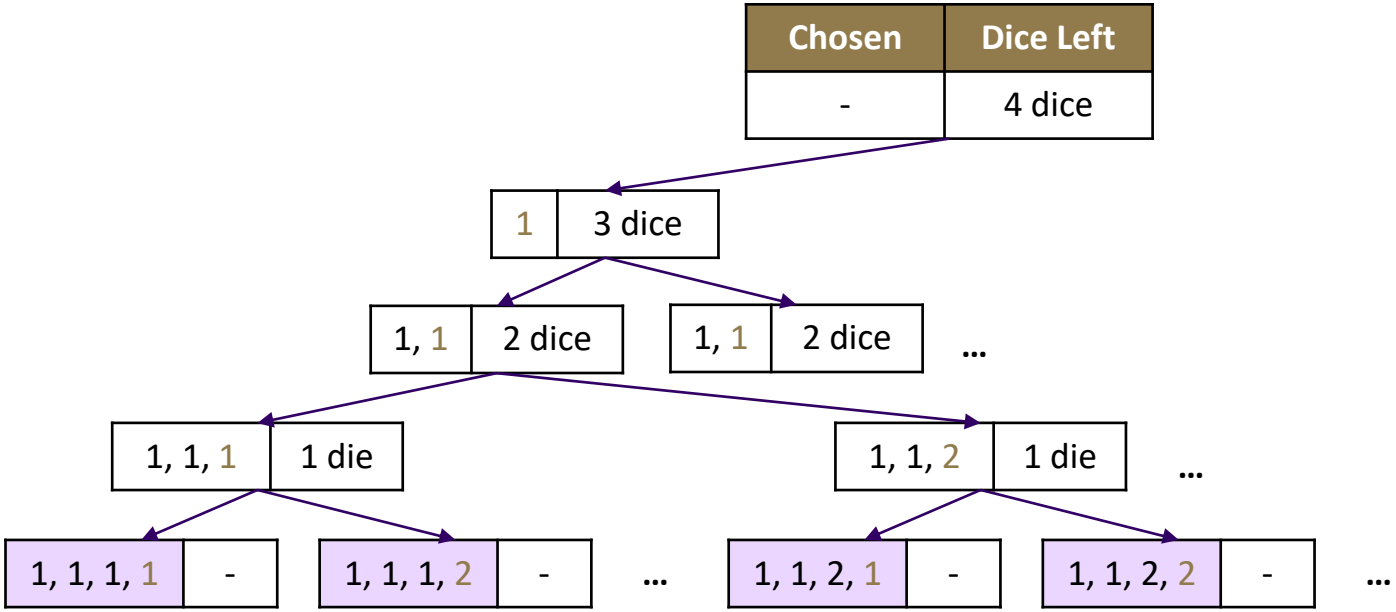
# `diceRolls` Decision Tree



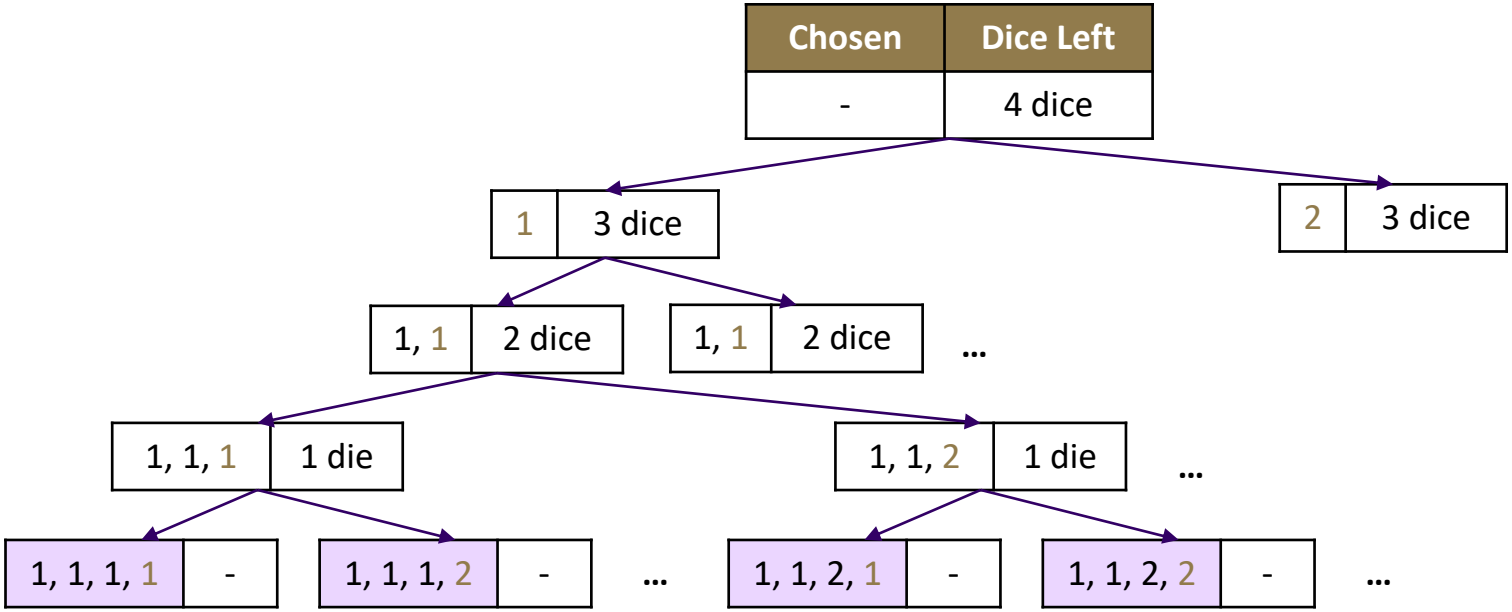
# `diceRolls` Decision Tree



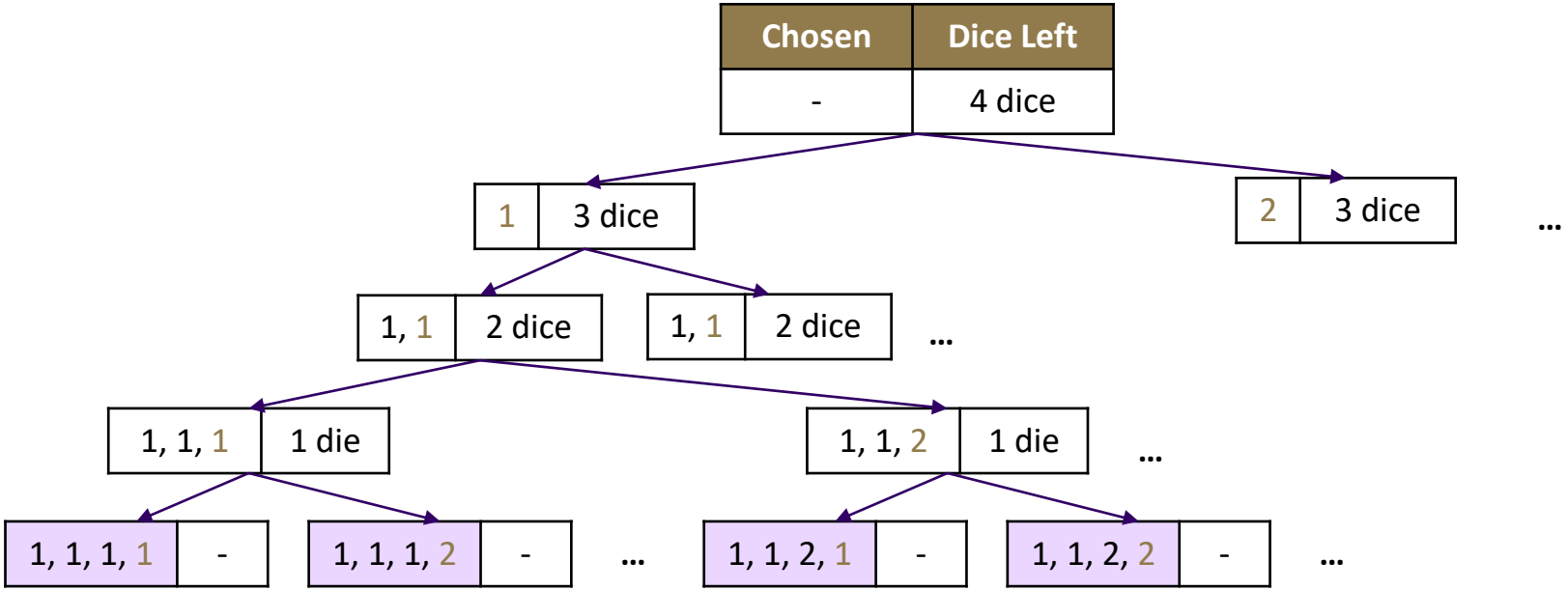
# `diceRolls` Decision Tree



# `diceRolls` Decision Tree



# `diceRolls` Decision Tree



# Agenda

- Exhaustive Search
- Gambling
- Phone Numbers 

# Phone Numbers

- Write a program that will produce all possible combinations of a phone number that is missing 2 numbers.