LEC 11

# CSE 123

# Recursive Backtracking

Questions during Class?

Raise hand or send here

sli.do #cse123A

BEFORE WE START

***Talk to your neighbors:***

*What's your favorite summer activity?*

**Instructors:** Nathan Brunelle

**TAs:**

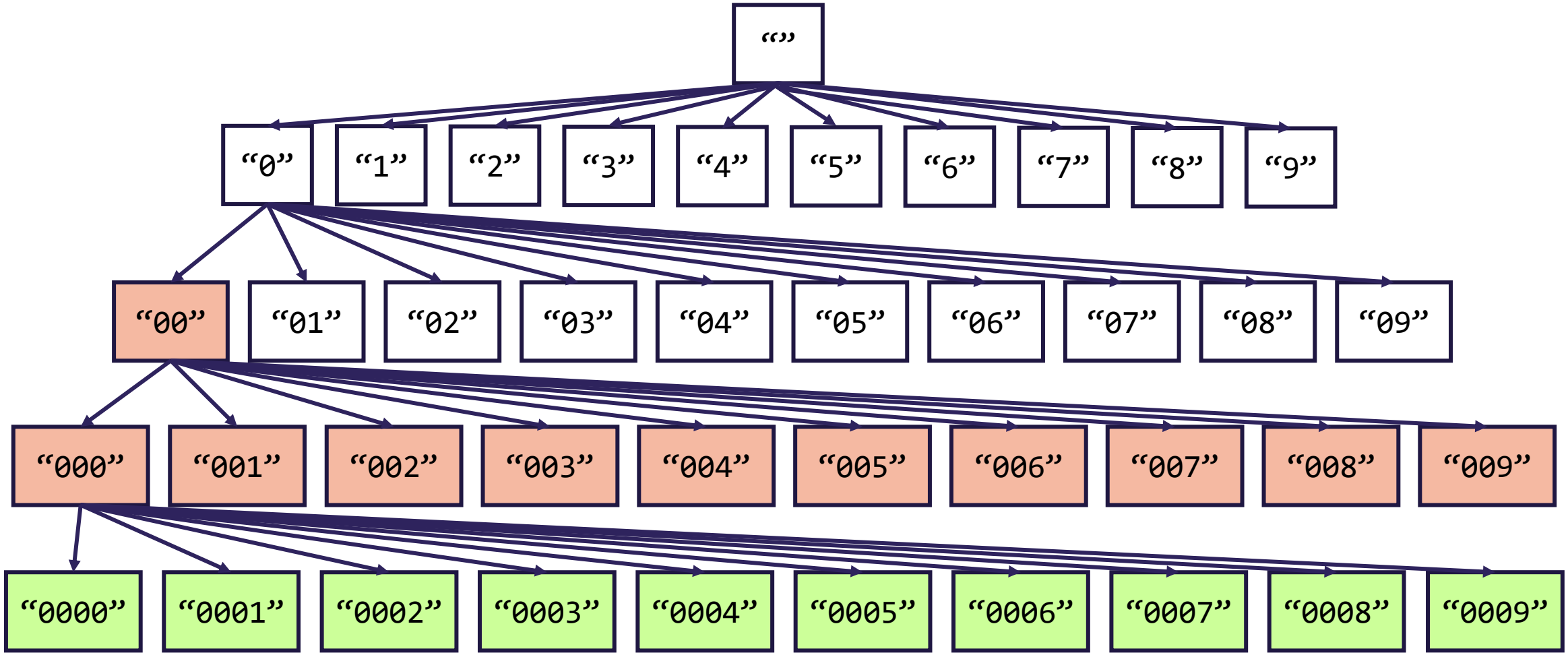
| | | | | |
|---|---|---|---|---|
| Arohan | Ashar | Neha | Rohini | Rushil |
| Ido | Zachary | Sebastian | Joshua | Sean |
| Hayden | Caleb | Justin | Heon | Rashad |
| Srihari | Benoit | Derek | Chris | Bhaumik |
| Kuhu | Kavya | Cynthia | Shreya | Ashley |
| Ziao | Kieran | Marcus | Crystal | Eeshani |
| Prakshi | Packard | Cora | Dixon | Nichole |
| Niyati | Trien | Lawrence | Evan | Cady |

# Announcements

- Creative Project 2 is out, due Wednesday, May 14
  - Focused on recursion!
- Resubmission Cycle 3 closes tonight (Friday, May 9)
  - ***P0***, C1 eligible
- The CSE 12x/14x TA application is now open for Spring 2025!

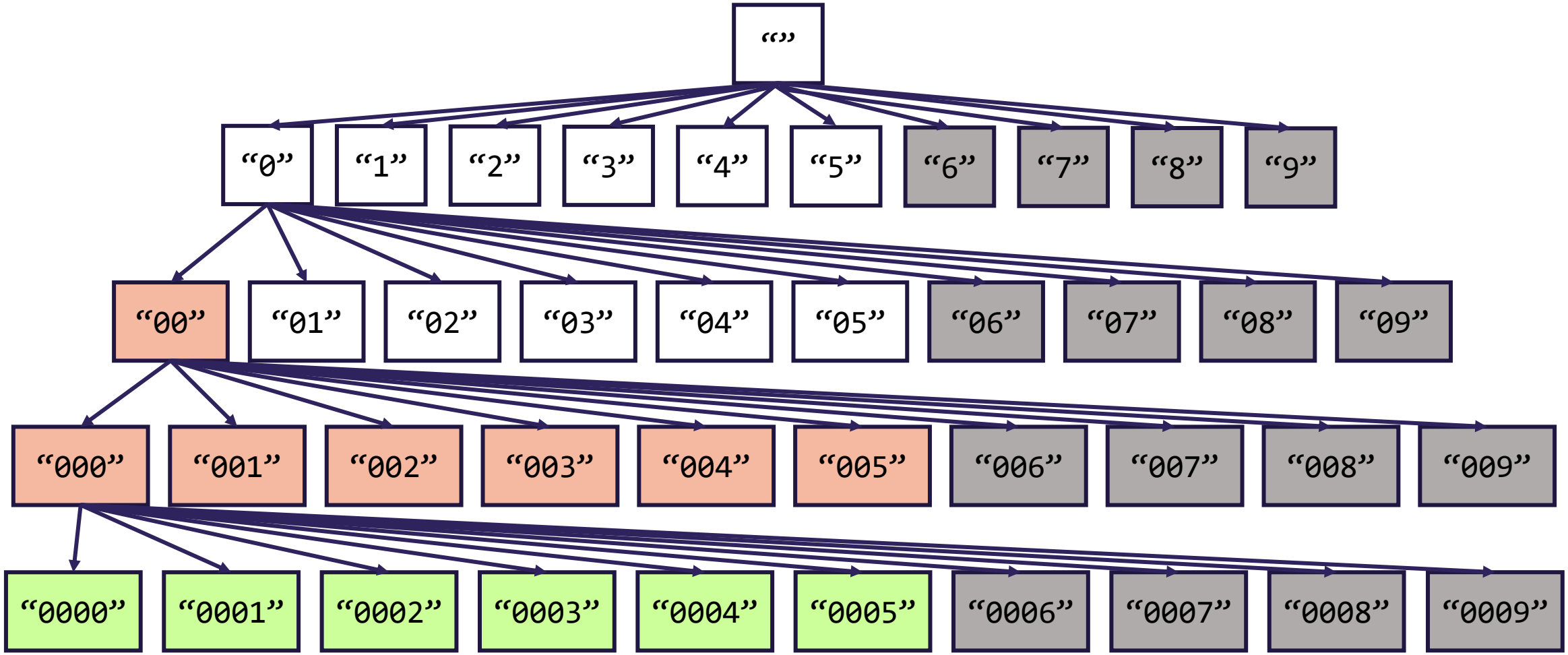
# Password Cracker

- Now, what if we knew the sum of all digits was 5?



# Password Cracker

- Now, what if we knew the sum of all digits was 5?



# Updated Exhaustive Search Pattern

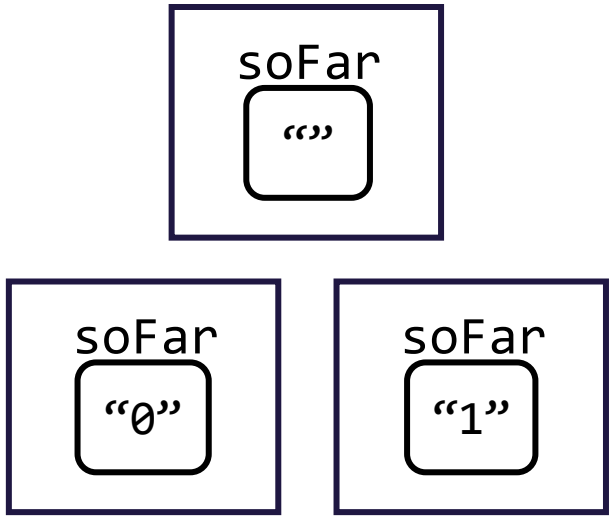
```
public static void search(input) {
    search(input, “”);
}

private static void search(input, String soFar) {
    if (base case) {
        // Do something with soFar (e.g. print it out)
        System.out.println(soFar);
    } else if (not dead end) {
        // Might not be a loop, but 1 recursive call for each option
        for (each option) {
            search(input, soFar + option);
        }
    }
}
```

# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

String soFar:

```
for (each option) {
    search(input, soFar + option);
}
```

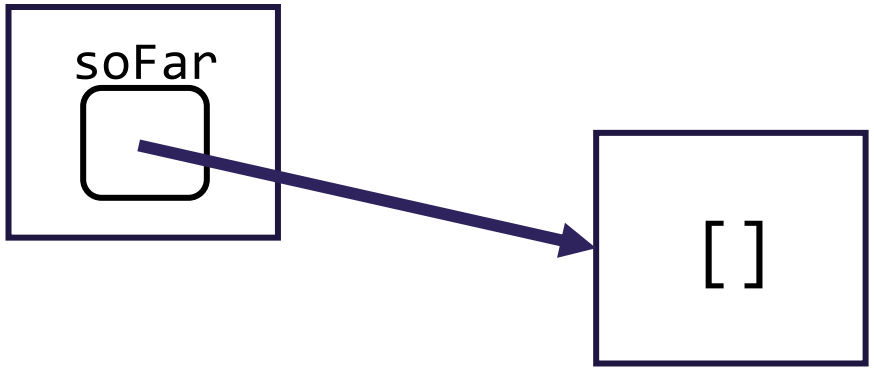


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() - 1);
}
```

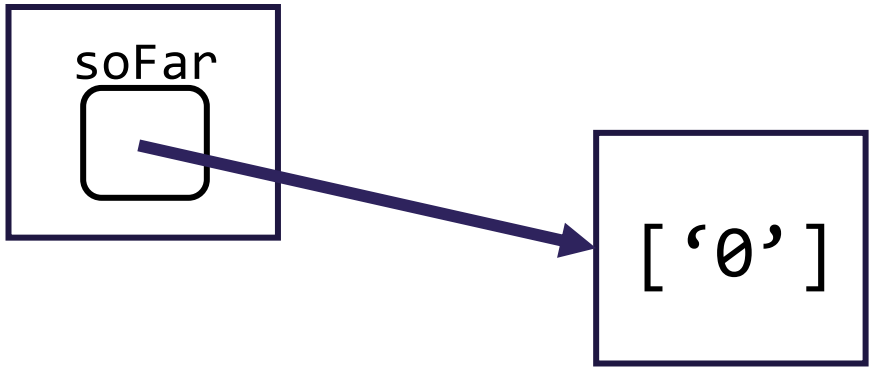


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() – 1);
}
```

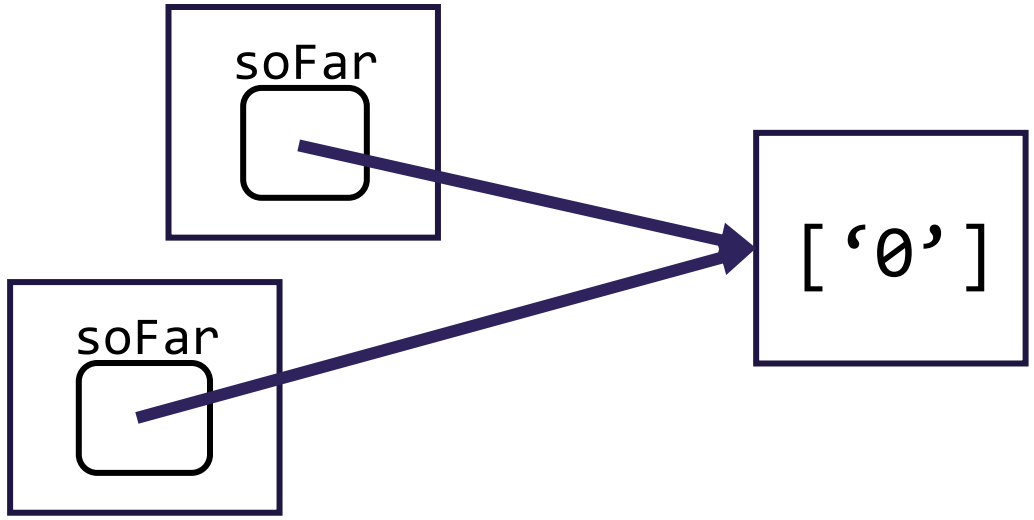


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() – 1);
}
```

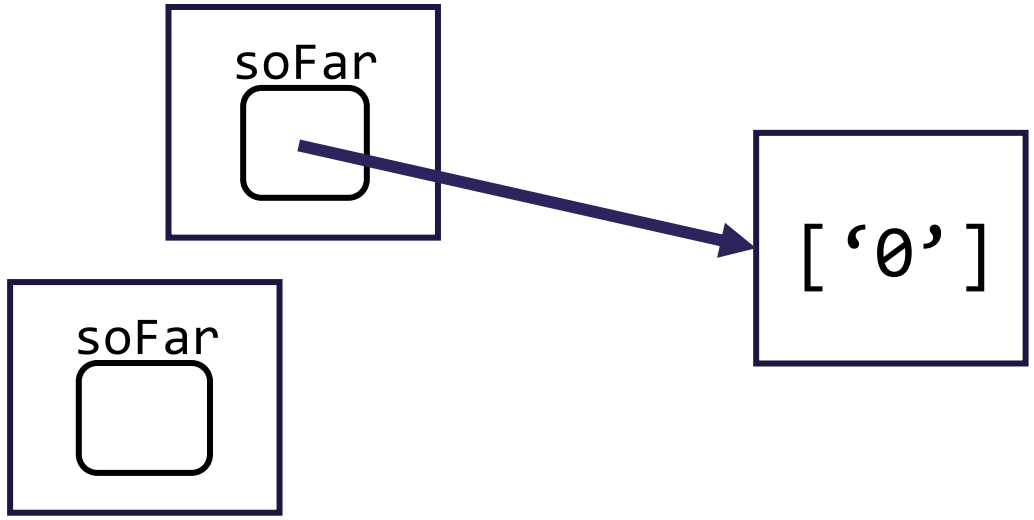


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() - 1);
}
```

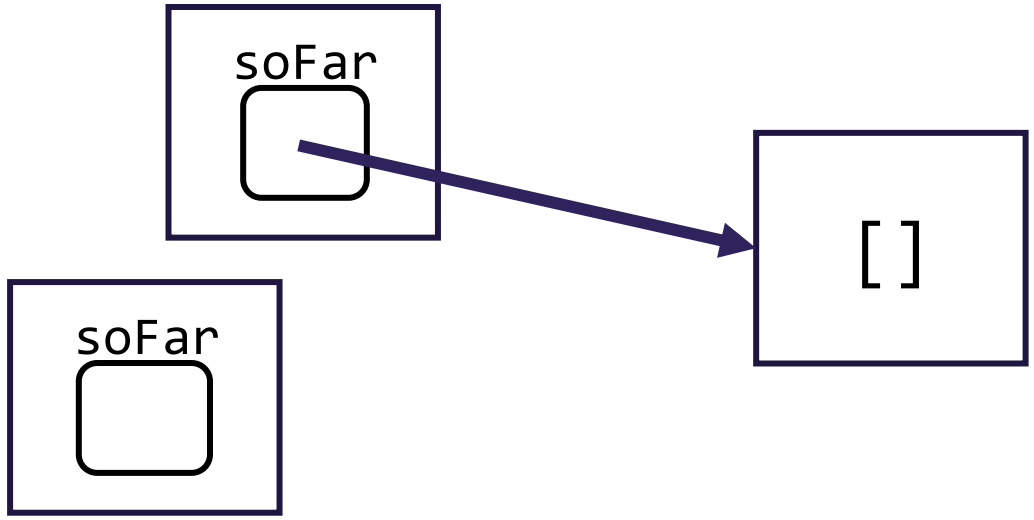


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() - 1);
}
```

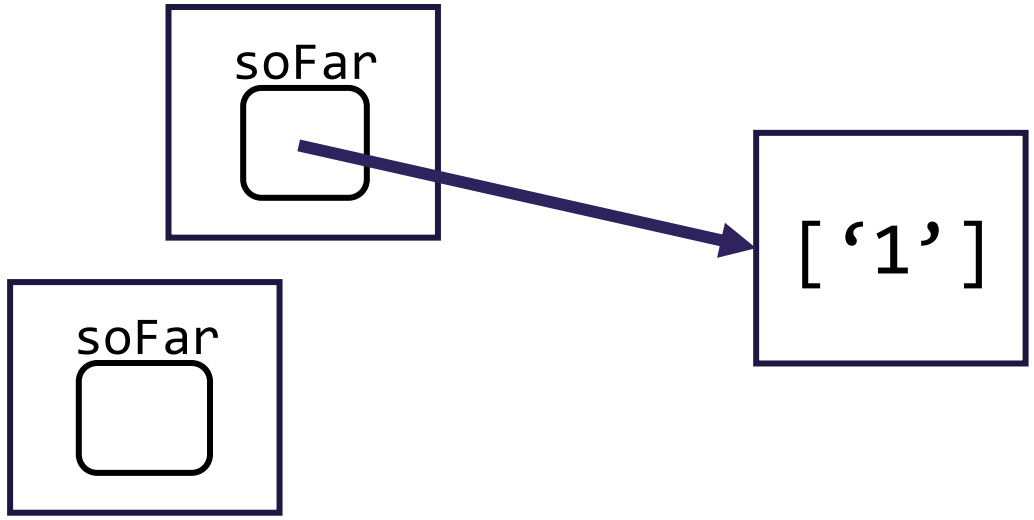


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() – 1);
}
```

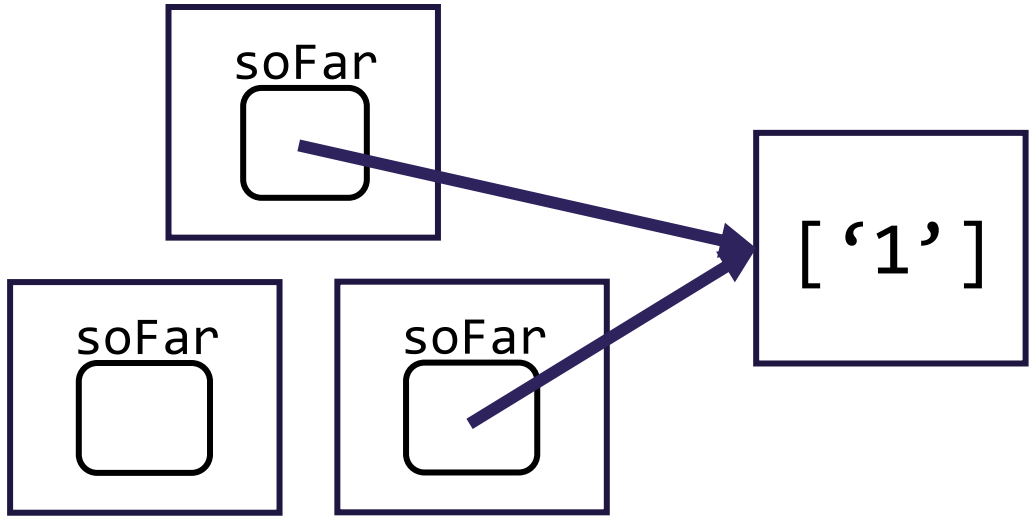


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() – 1);
}
```

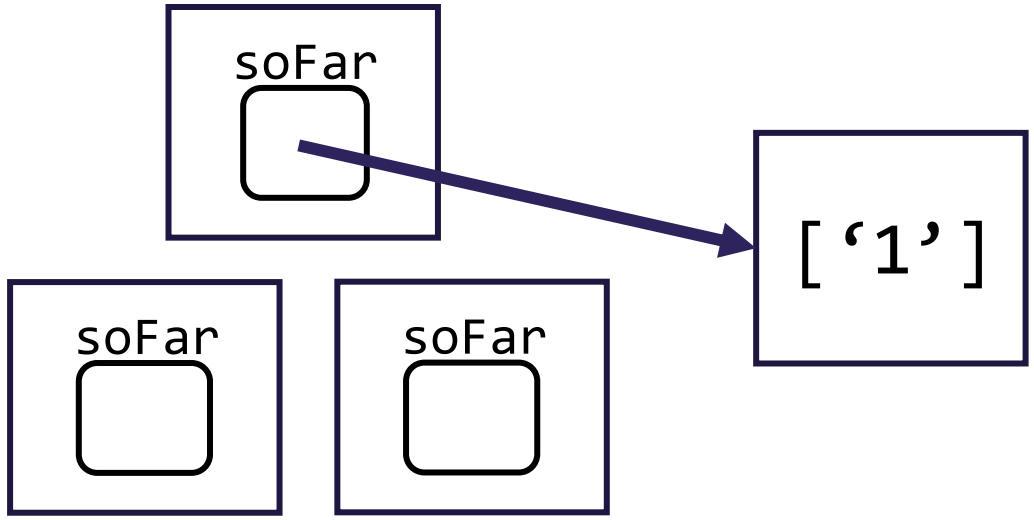


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() – 1);
}
```

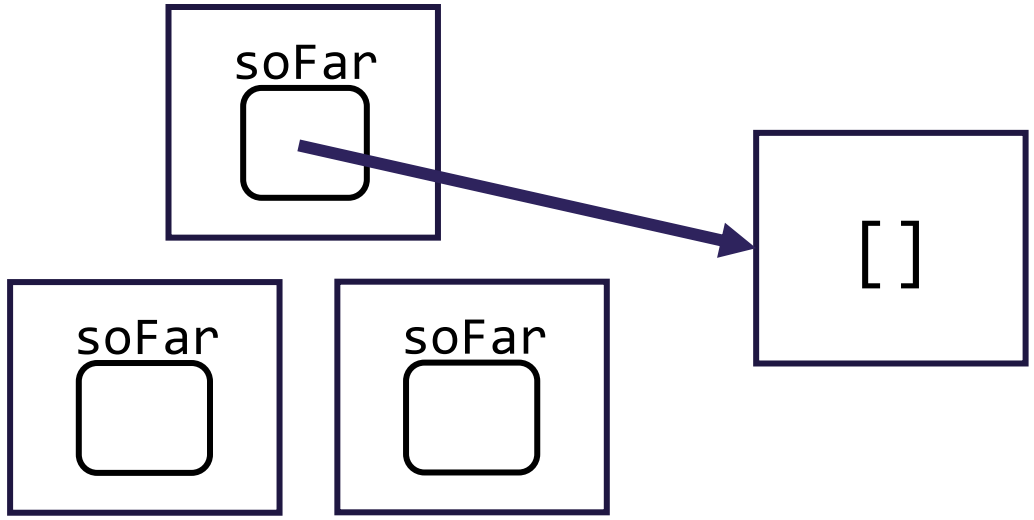


# Recursive Backtracking

- Exhaustive search with a data structure accumulator(s)
  - Now we have to deal with reference semantics…
- Major pattern: **Choose, Explore, Un-choose**
  - All of the stack frames share the same *one* data structure
  - Need to explicitly un-choose it so it's not remembered in other frames

List<Character> soFar:

```
for (each option) {
    soFar.add(option);
    search(input, soFar);
    soFar.remove(soFar.size() - 1);
}
```



# Recursive Backtracking Pattern

```
private static void search(input, List<Character> soFar) {
    if (base case) {
        // Do something with soFar (e.g. print it out)
        System.out.println(soFar);
    } else if (not dead end) {
        // Might not be a loop, but 1 recursive call for each option
        for (each option) {
            soFar.add(option);                  // Choose
            search(input, soFar);               // Explore
            soFar.remove(soFar.size() - 1);     // Unchoose
        }
    }
}
```