

LEC 11

CSE 123

Recursive Backtracking

Questions during Class?

Raise hand or send here

sli.do #cse123



BEFORE WE START

*Talk to your neighbors:**What's your favorite
board game?***Instructors:** Brett Wortzman
Miya Natsuhara**TAs:**

Arohan	Neha	Rushil	Johnathan	Nicholas
Sean	Hayden	Srihari	Benoit	Isayah
Audrey	Chris	Andras	Jessica	Kavya
Cynthia	Shreya	Kieran	Rohan	Eeshani
Amy	Packard	Cora	Dixon	Nichole
Trien	Lawrence	Liza	Helena	

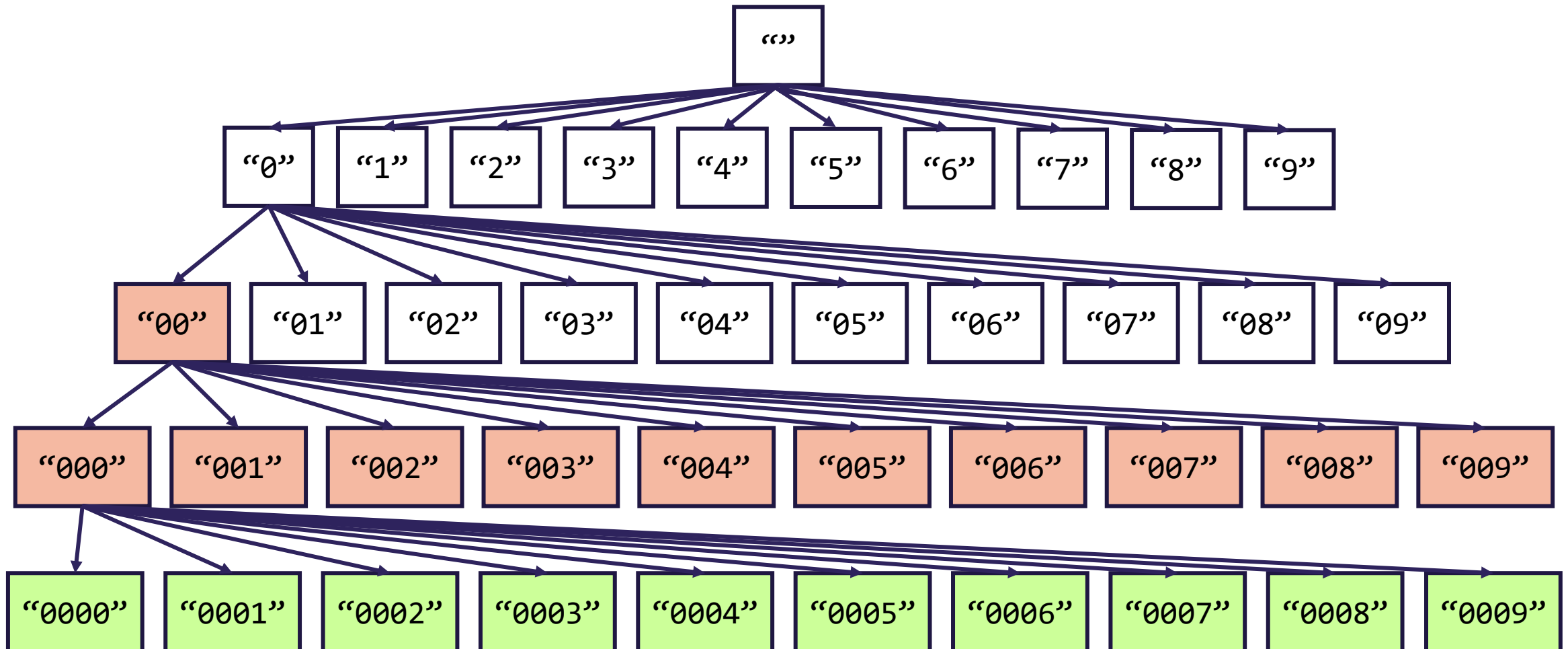
Music: [CSE 123 25wi Lecture Tunes](#)

Announcements

- Creative Project 2 is out, due Wednesday, Feb 19
 - Focused on recursion!
- Resubmission Cycle 3 closes tonight (Friday, Feb 14)
 - PO, C1 eligible
- The [CSE 12x/14x TA application](#) is now open for Spring 2025!
- University holiday on Monday
 - No IPL, no instructor OH, possibly longer message board response times

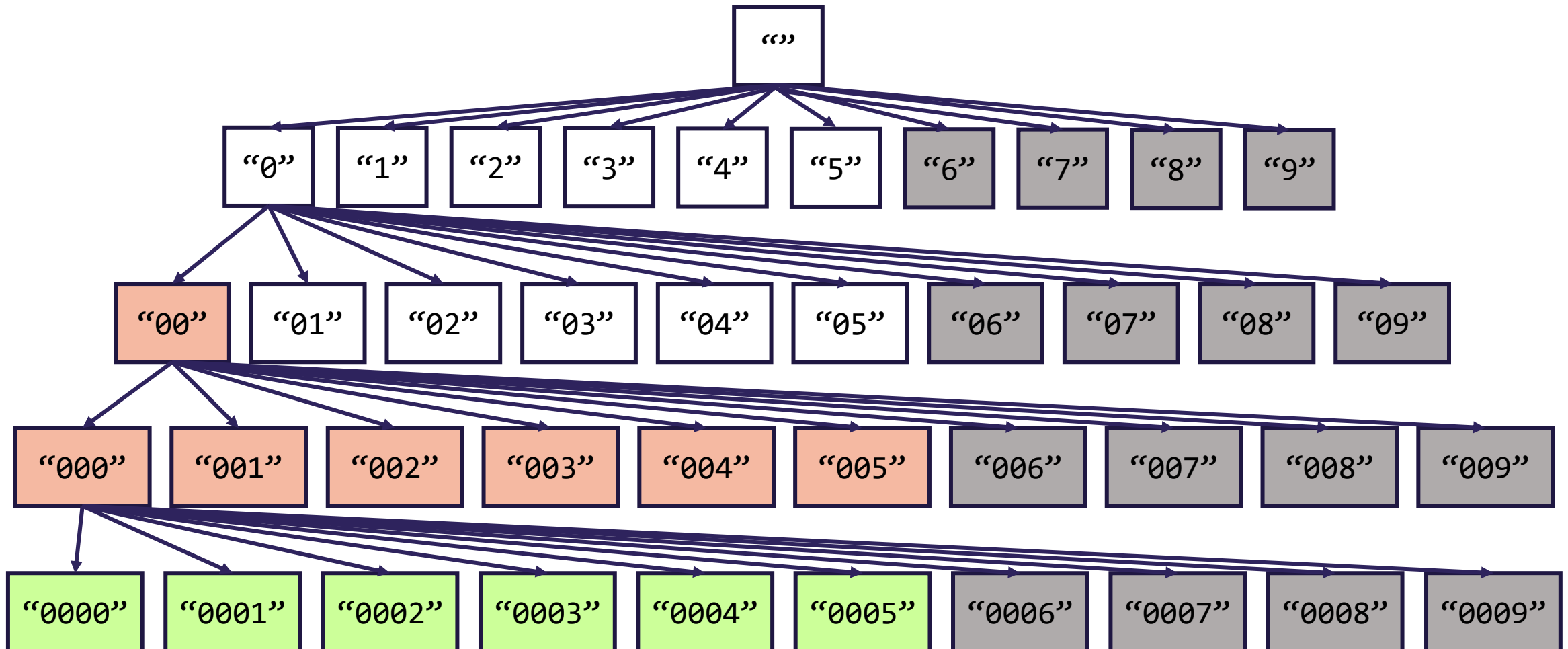
Password Cracker

- 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, "");
}

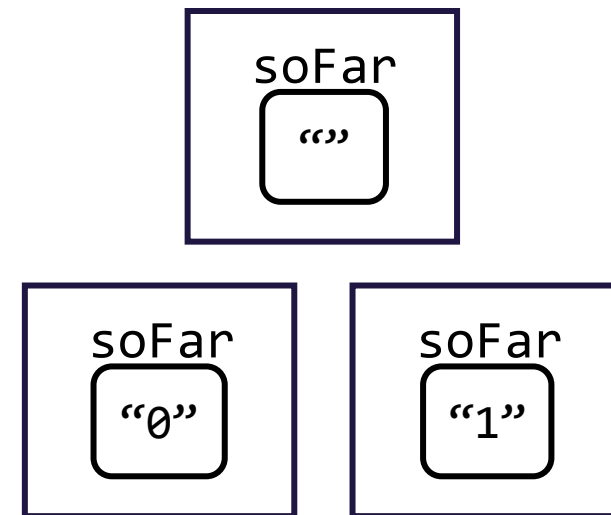
private static void search(input, String soFar) {
    if (base case) {
        // Do something with soFar (e.g. print it out)
        System.out.println(soFar);
    } else if (not dead end) {
        // Might not be a loop, but 1 recursive call for each option
        for (each option) {
           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

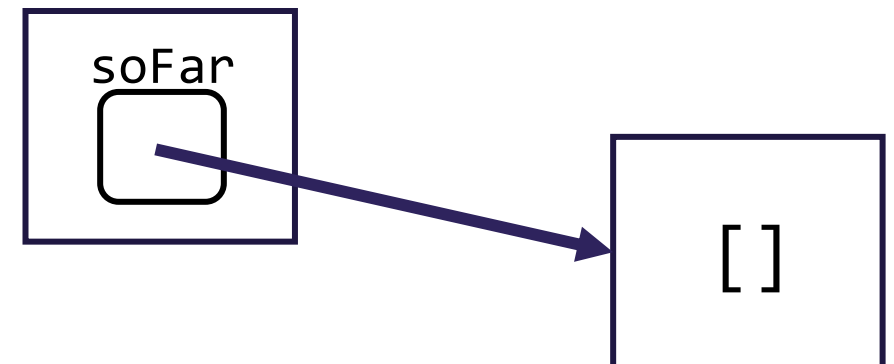


Recursive Backtracking

- 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);  
}
```

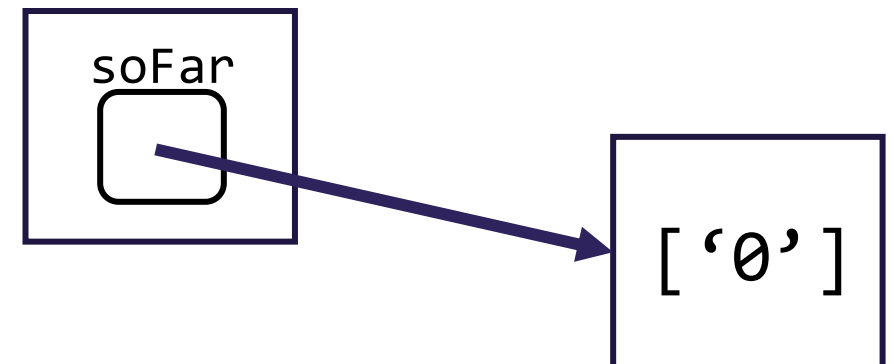


Recursive Backtracking

- 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);  
}
```

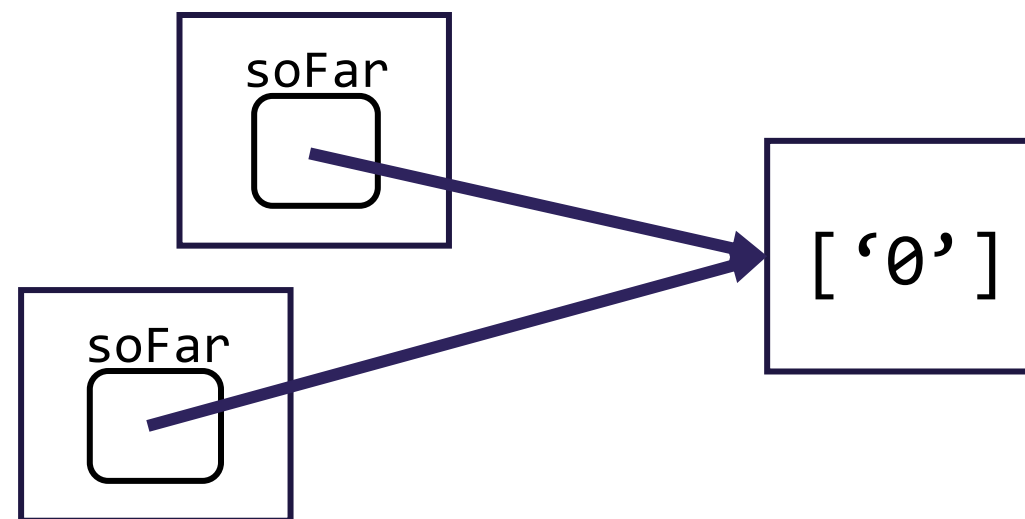


Recursive Backtracking

- 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);  
}
```

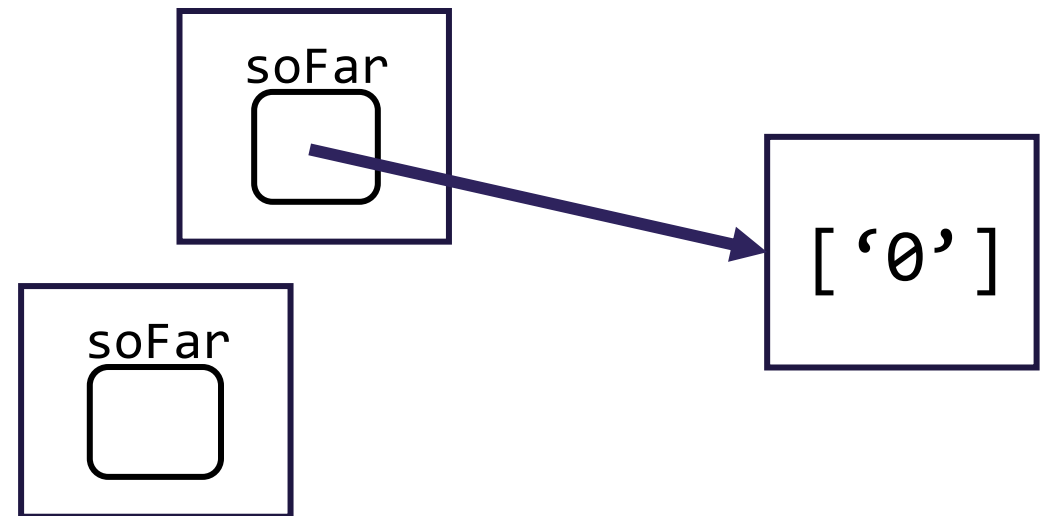


Recursive Backtracking

- 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);  
}
```

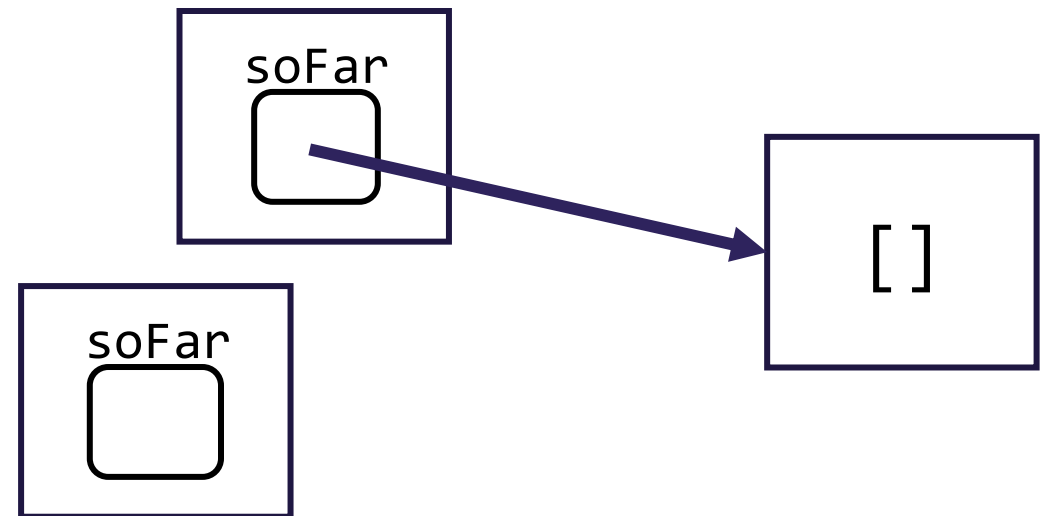


Recursive Backtracking

- 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);  
}
```

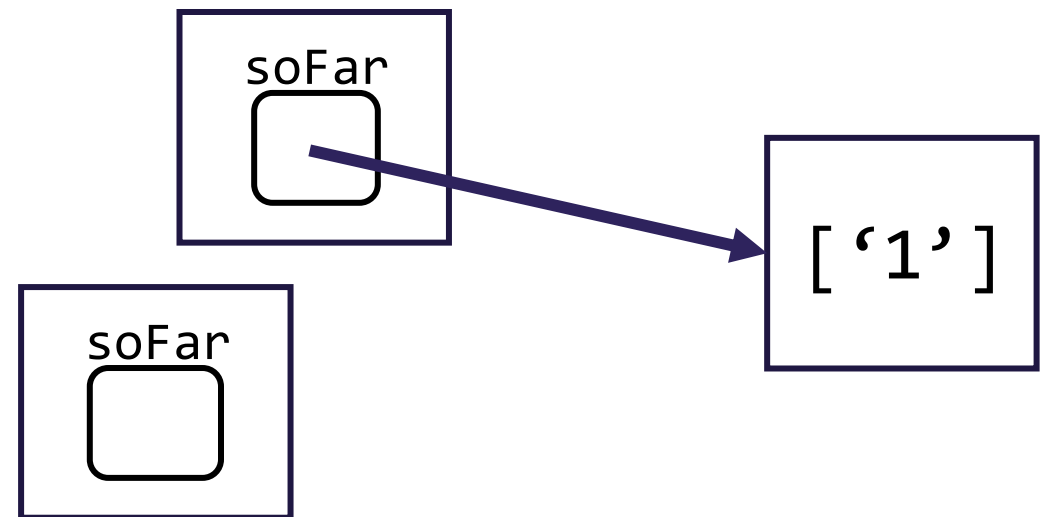


Recursive Backtracking

- 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);  
}
```

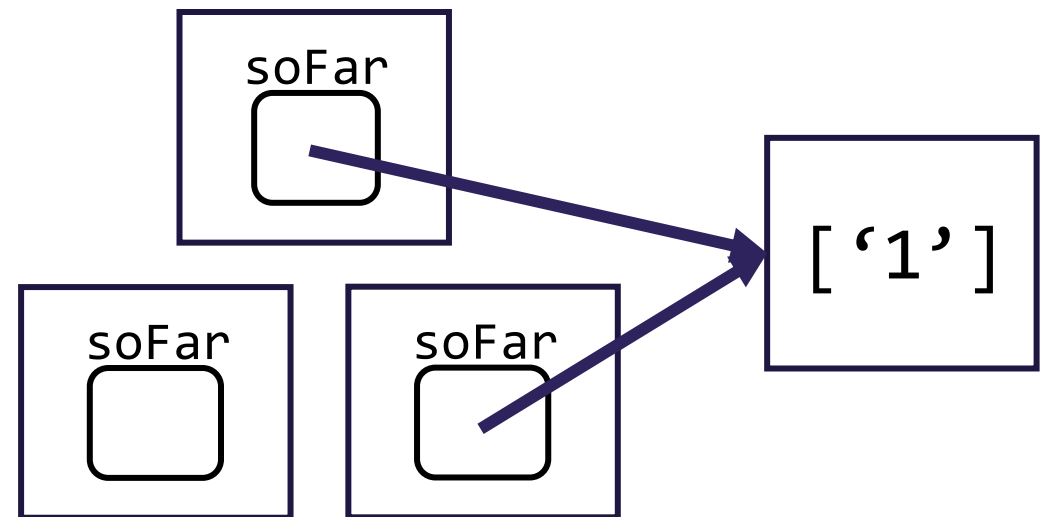


Recursive Backtracking

- 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);  
}
```

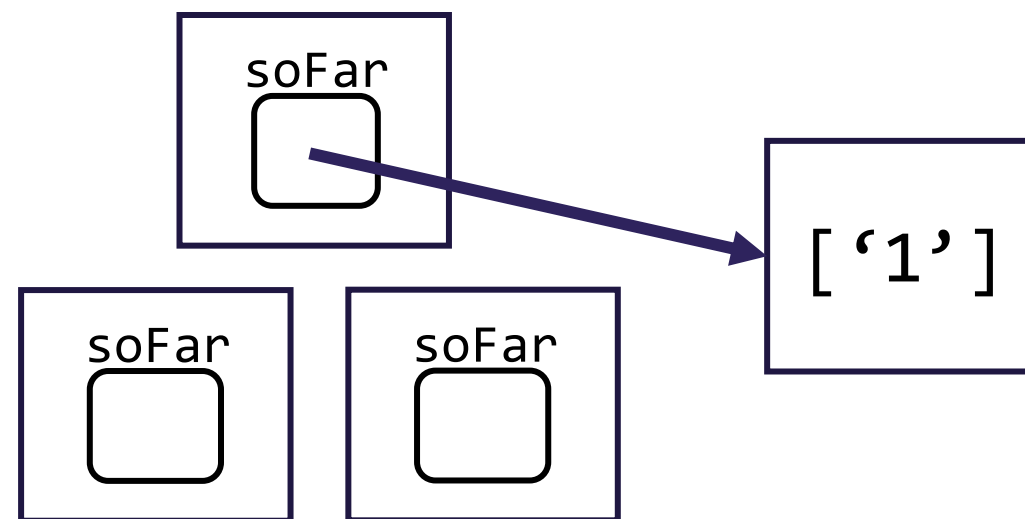


Recursive Backtracking

- 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);  
}
```

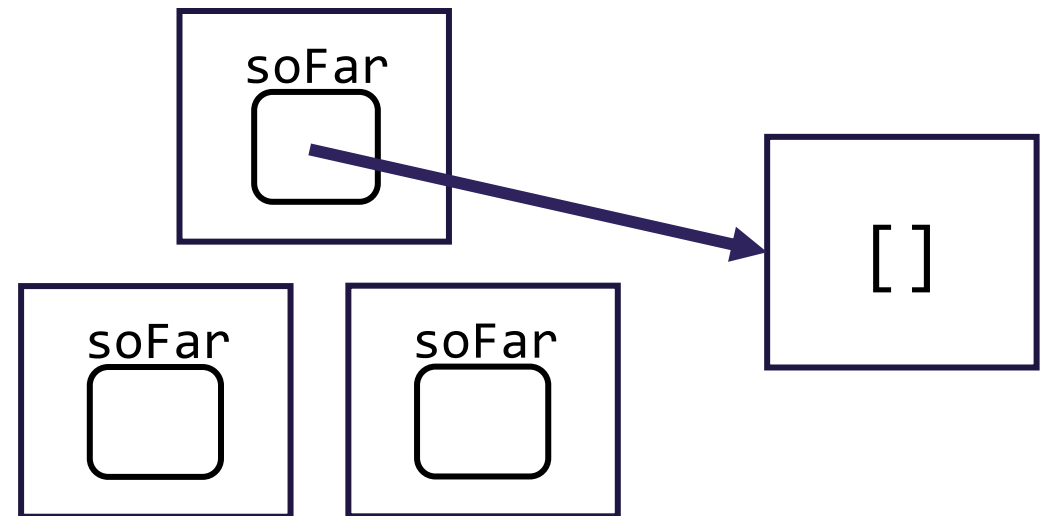


Recursive Backtracking

- 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  
        }  
    }  
}
```