LEC 11

CSE 123

Recursive Backtracking

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Valentine's Day: romantic or commercial?

Respond on sli.do!

Instructor: Brett Wortzman

TAs:

Arohan	Jonah	Kavya	Eeshani	Trien
Ashar	Brice	Misha	Aidan	Evan
Sean	Chris	Kieran	Cora	Rena
Chloe	Elden	Sahana	Dixon	Katharine
Jenny	Ishita	Anirudh	Nhan	Anya
Nate	Kuhu	Crystal		

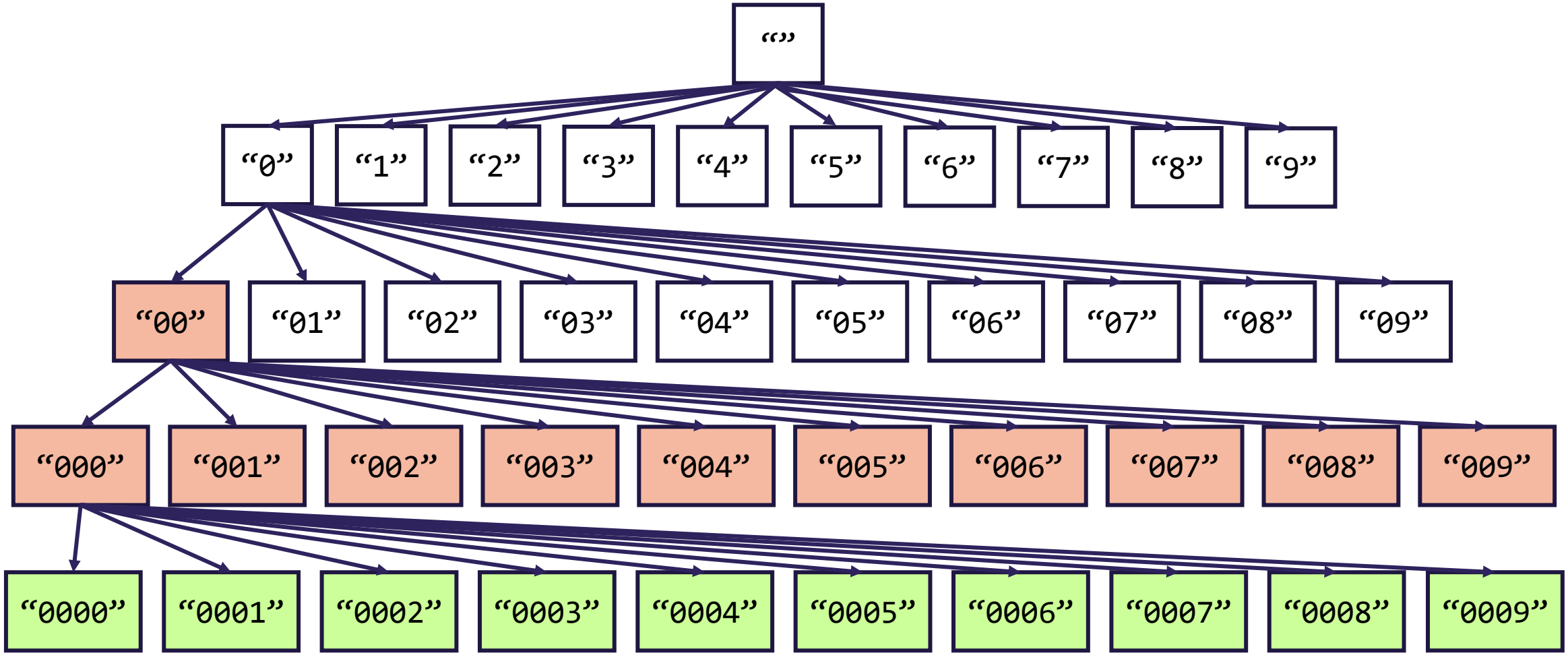
Now playing: CSE 123 26wi Lecture Tunes

Announcements

- Creative Project 2 is out, due Wednesday, February 18
  - Focused on recursion!
- Resubmission Cycle 3 closes tonight (Friday, February 13)
  - ***P0***, C1 eligible
- CSE 12X TA application is open!
- Brett travelling next week
  - No office hours, guest lecturers
- Monday is a holiday
  - IPL closed

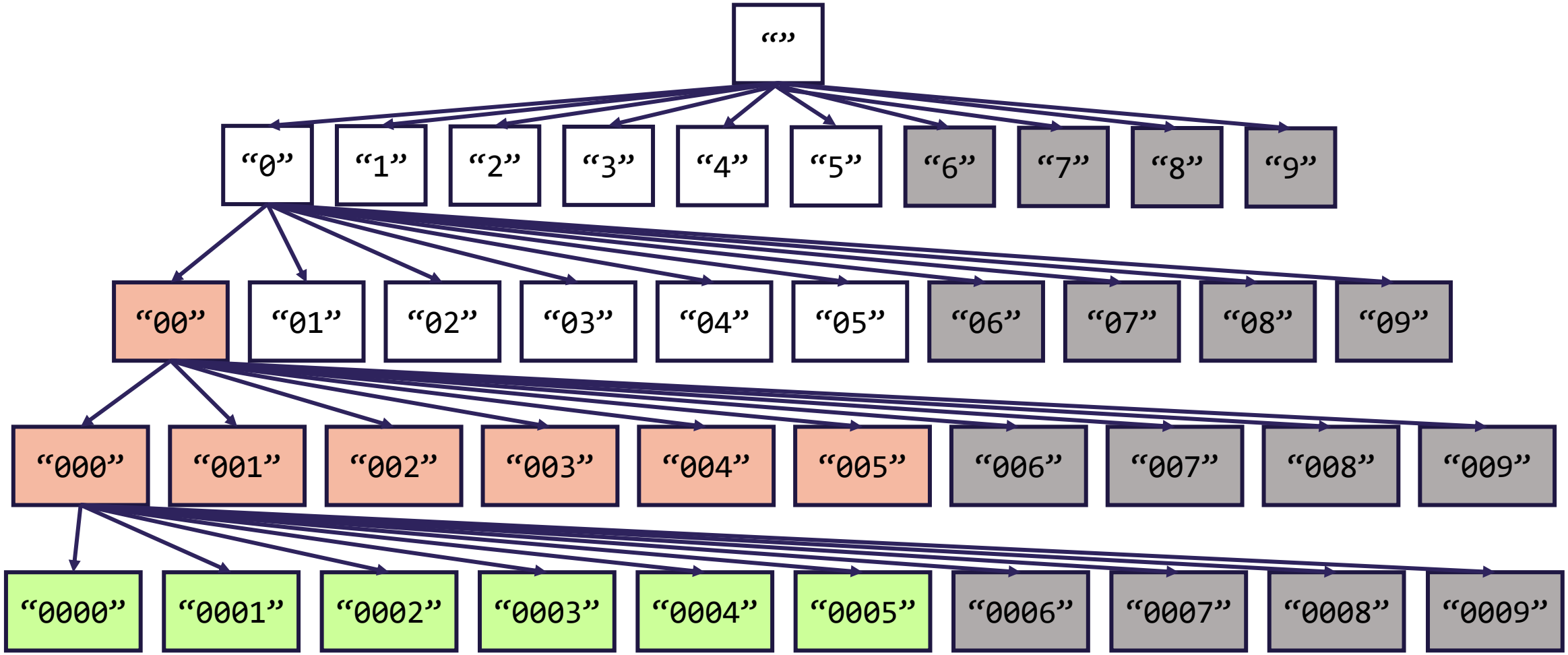
Password Cracker

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



Password Cracker: Pruning Dead Ends

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



Updated Exhaustive Search Pattern

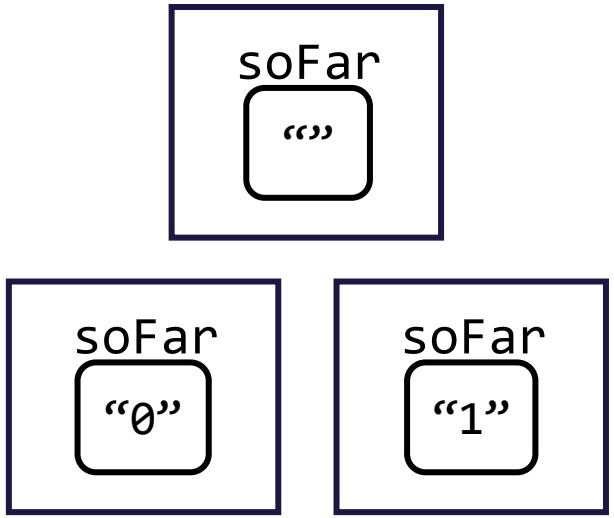
```
public static void search(input) {
    search(input, “”);
}

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);
}
```

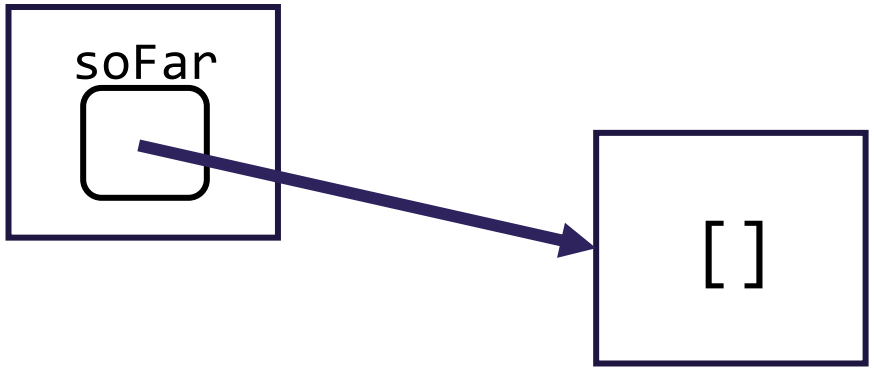


Recursive Backtracking

- 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);
}
```

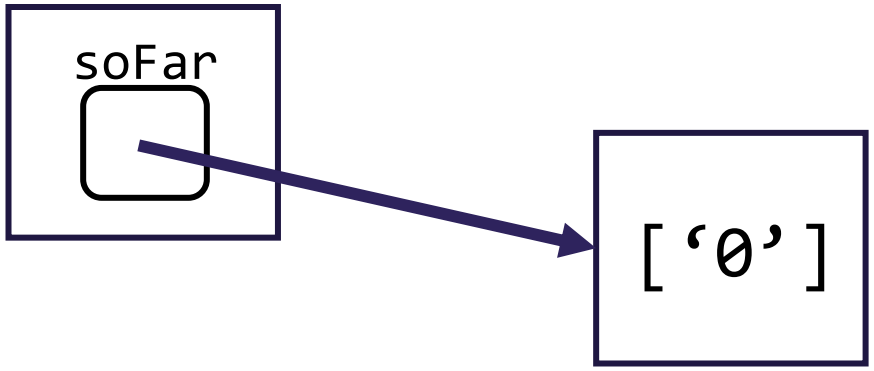


Recursive Backtracking

- 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);
}
```

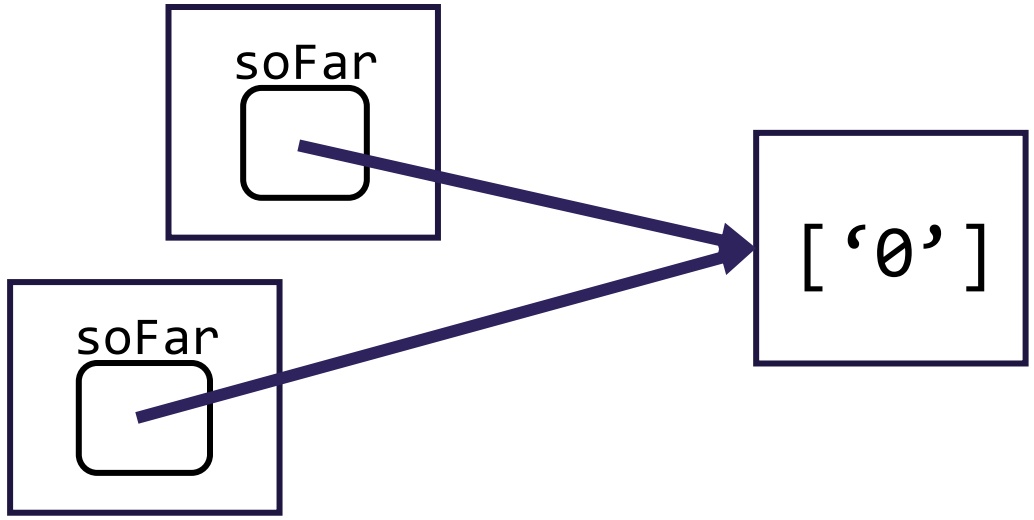


Recursive Backtracking

- 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);
}
```

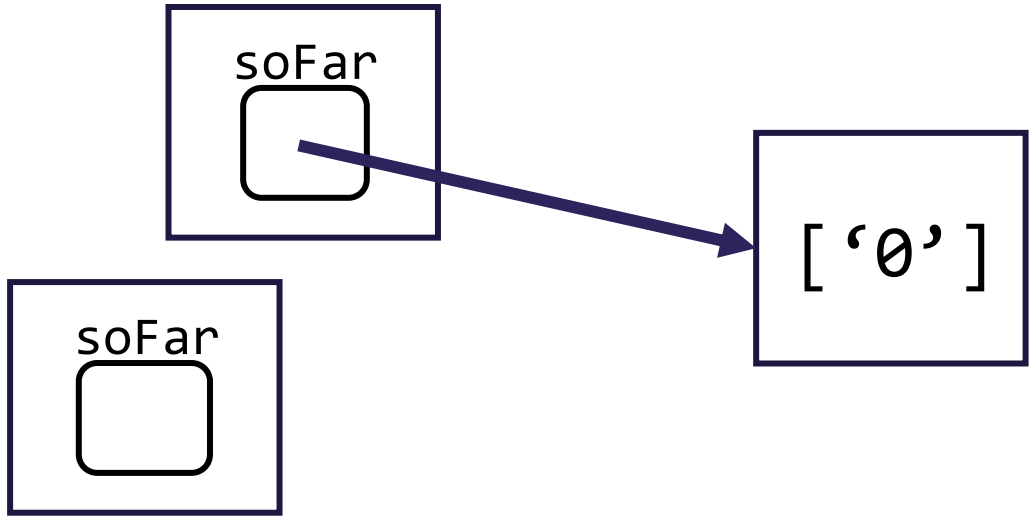


Recursive Backtracking

- 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);
}
```

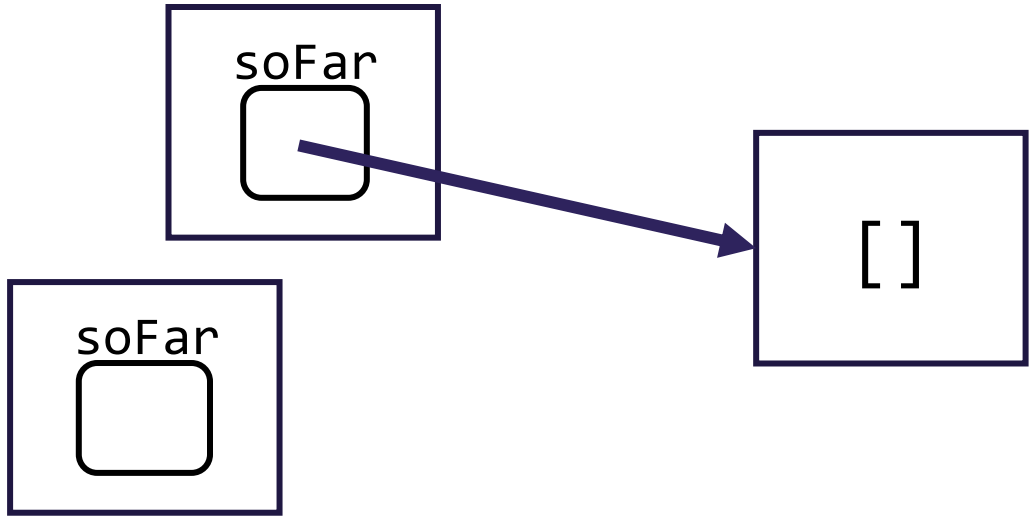


Recursive Backtracking

- 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);
}
```

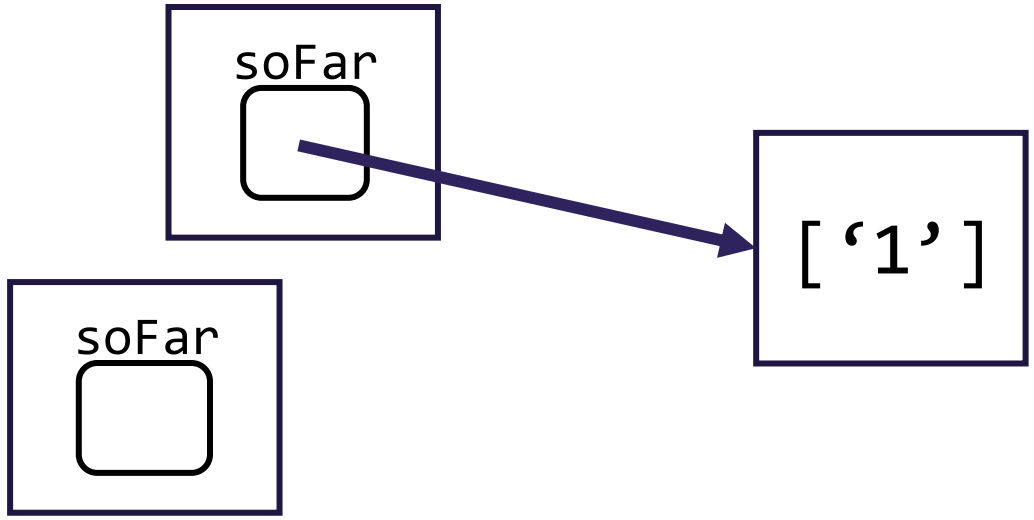


Recursive Backtracking

- 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);
}
```

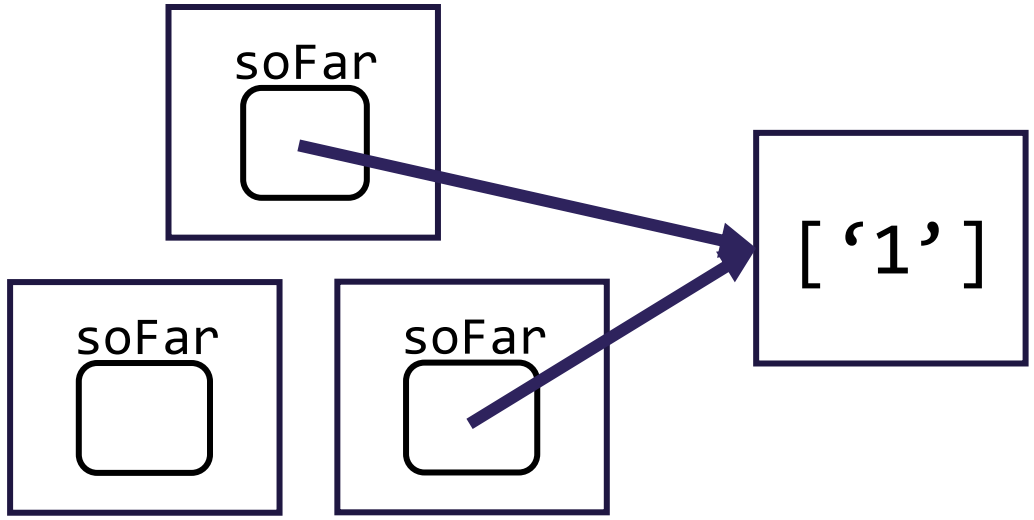


Recursive Backtracking

- 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);
}
```

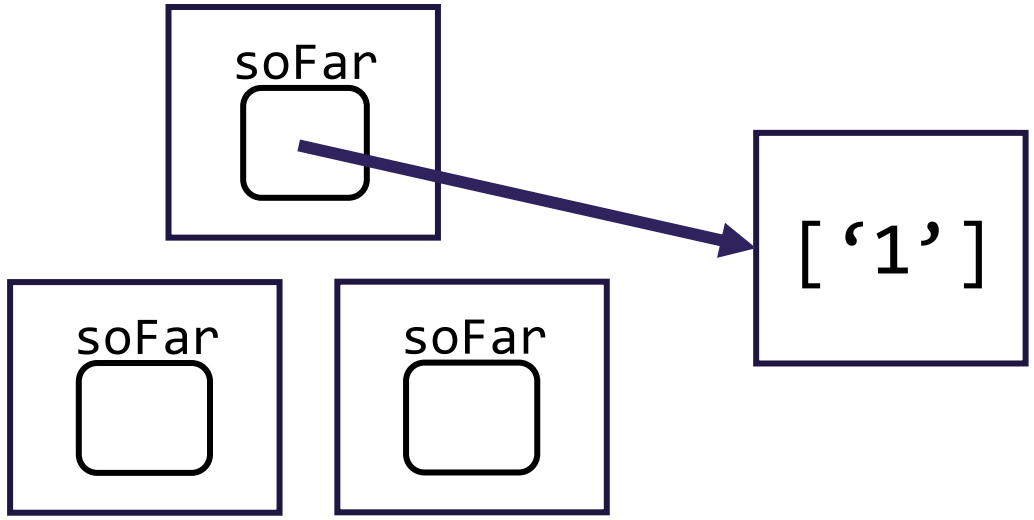


Recursive Backtracking

- 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);
}
```

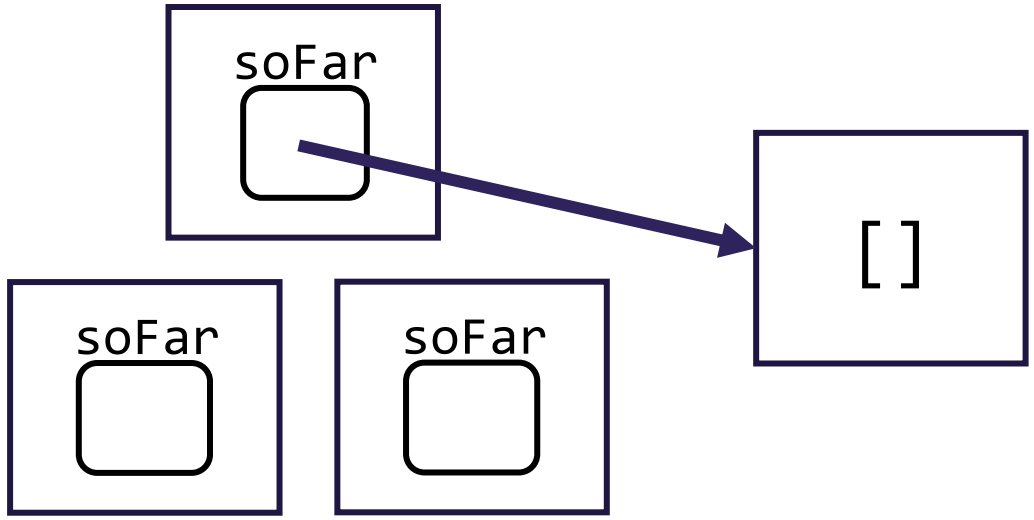


Recursive Backtracking

- 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
        }
    }
}
```