LEC 11

CSE 123

Recursive Backtracking

Questions during Class?

Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Do you play Sudoku?

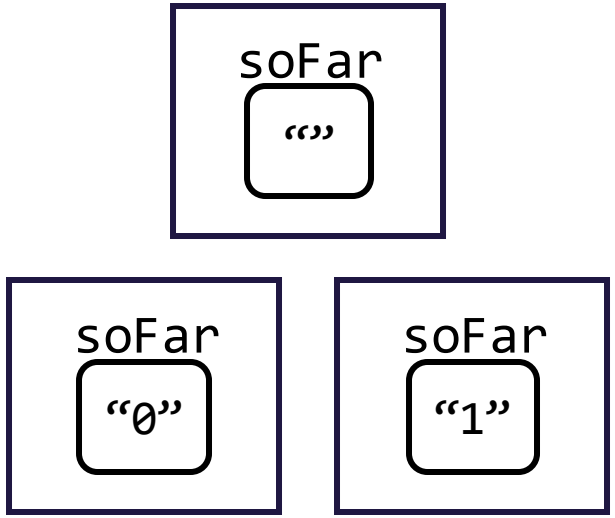
Instructor: James Wilcox

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);
}
```

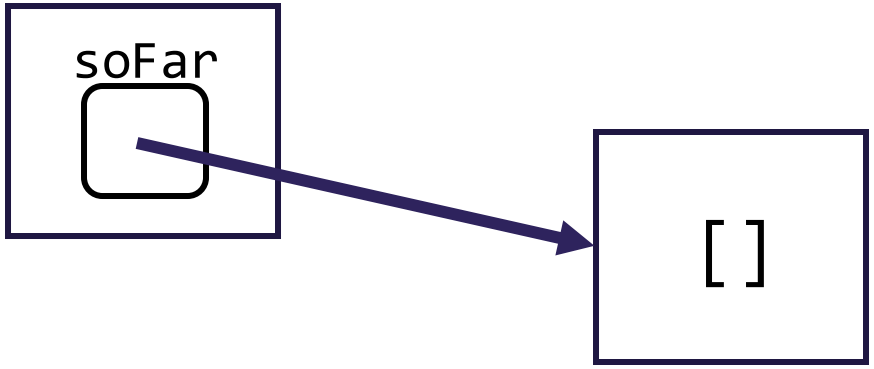


Recursive Backtracking

- 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);
}
```

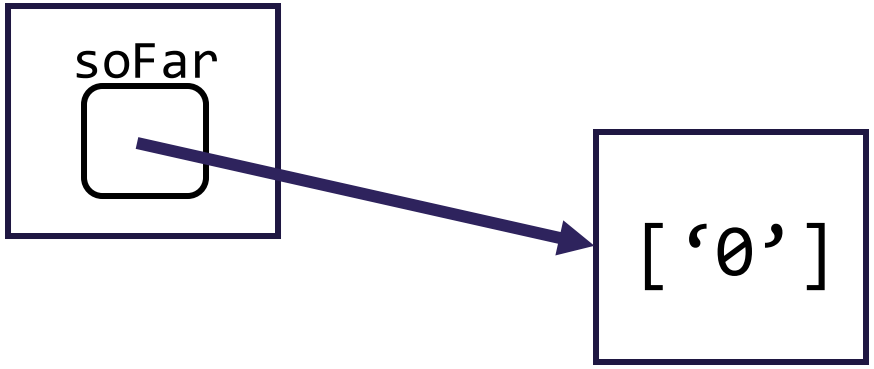


Recursive Backtracking

- 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);
}
```

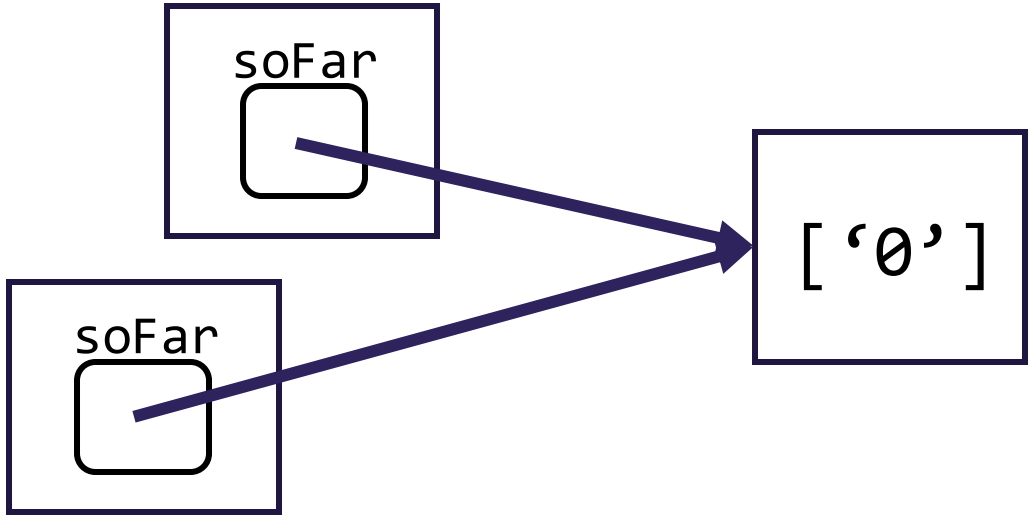


Recursive Backtracking

- 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);
}
```

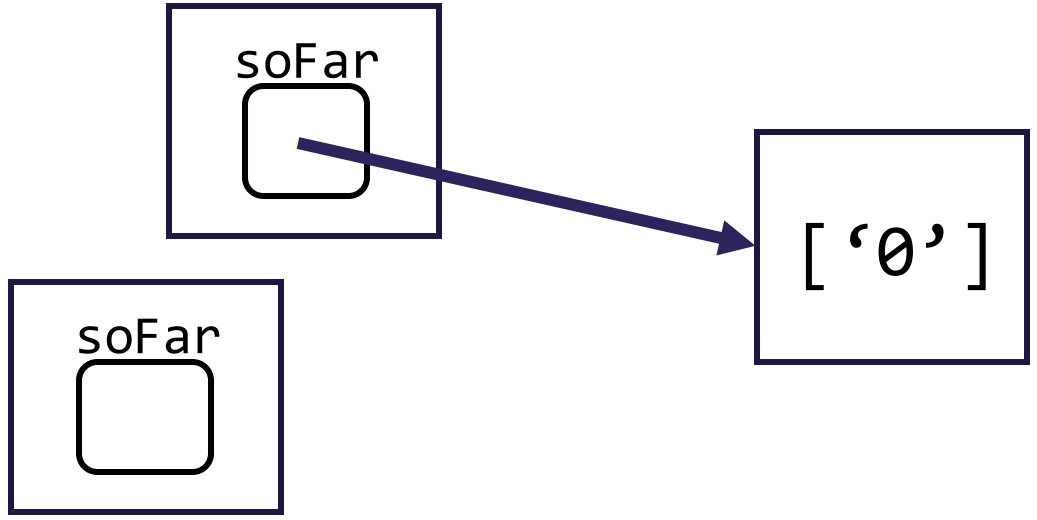


Recursive Backtracking

- 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);
}
```

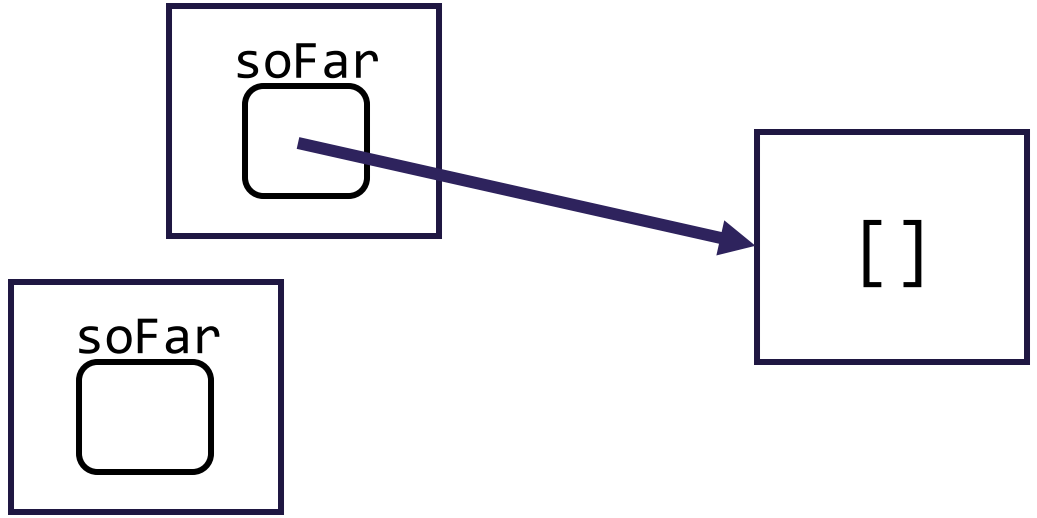


Recursive Backtracking

- 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);
}
```

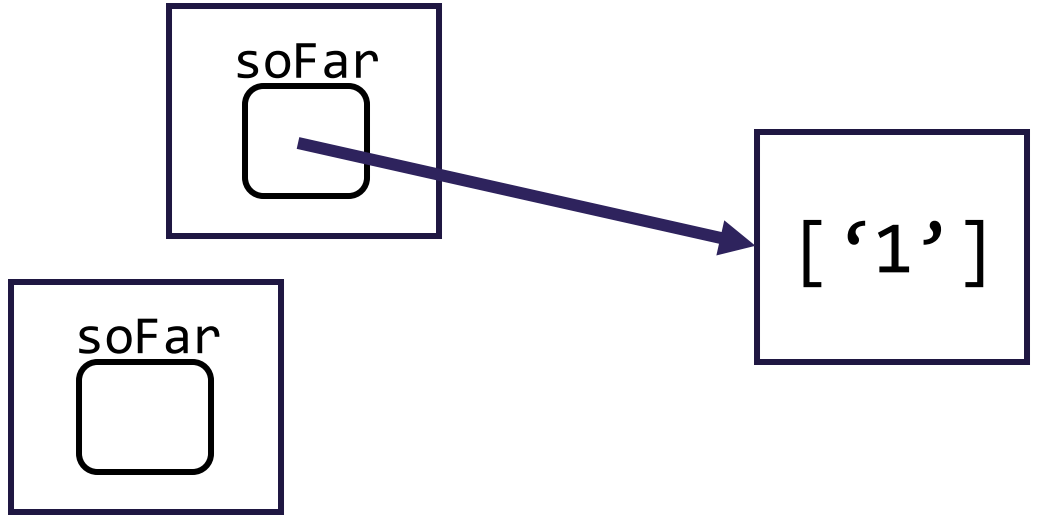


Recursive Backtracking

- 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);
}
```

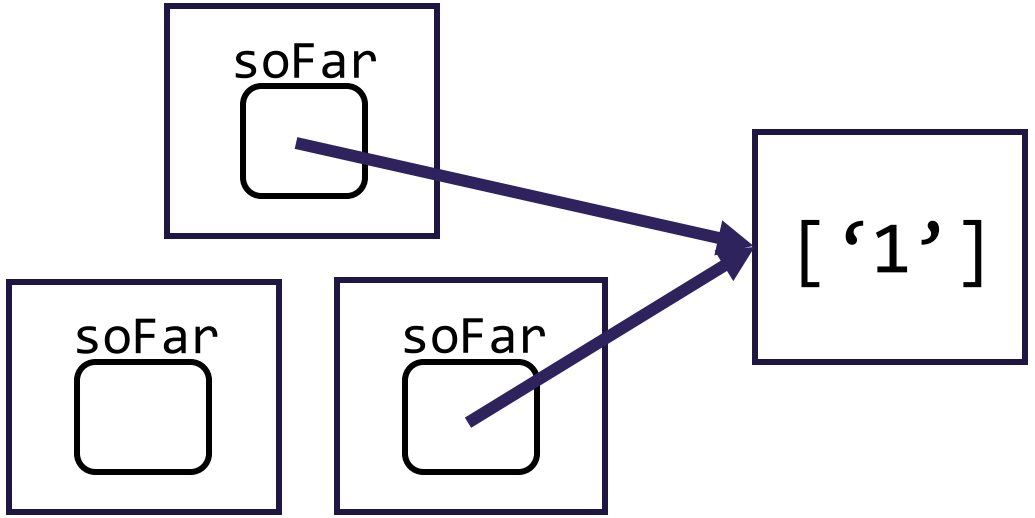


Recursive Backtracking

- 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);
}
```

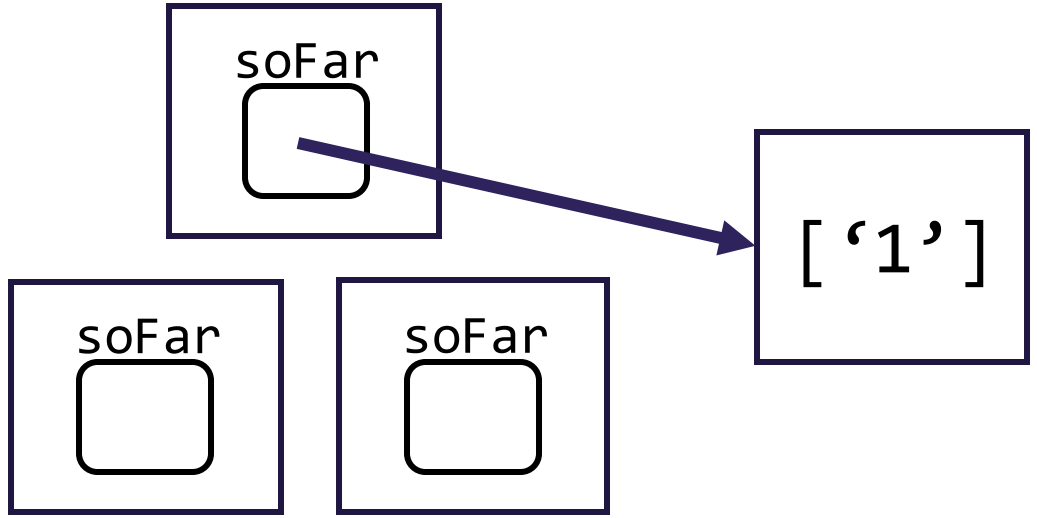


Recursive Backtracking

- 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);
}
```

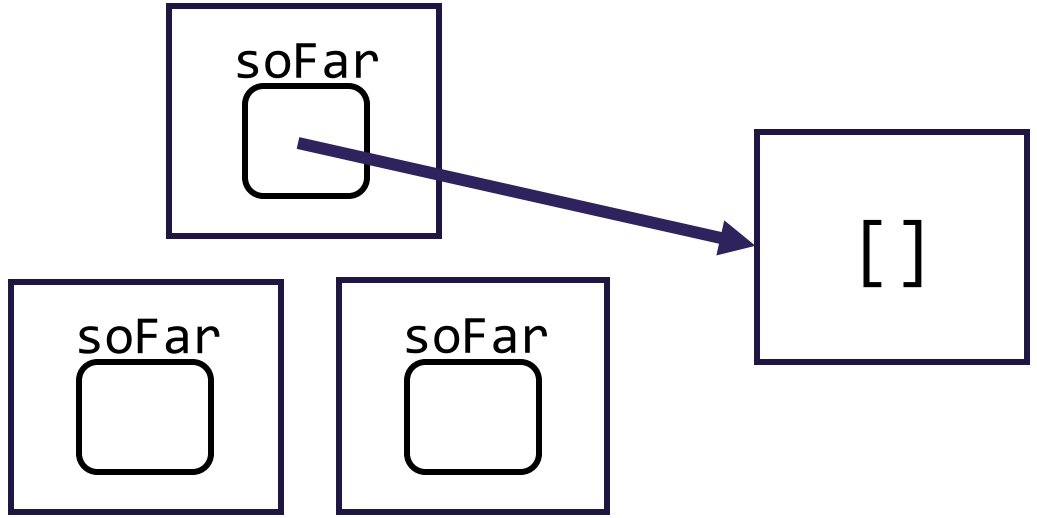


Recursive Backtracking

- 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
        }
    }
}
```