Recursion

09114319: Data Structures and Algorithms

Ratthaprom PROMKAM, Dr. rer. nat.

Department of Mathematics and Computer Science, RMUTT

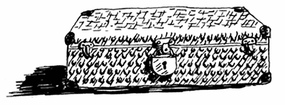
Outline

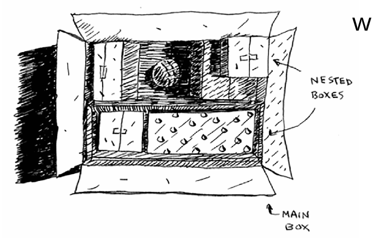
- Introduction to Recursion
- Base Case and Recursive Case
- Understanding the Call Stack
- Examples of Recursive Functions
- Recap and Key Takeaways

What is Recursion?

- Recursion is a method of solving problems where a function calls itself.
- A recursive solution typically has two parts:
  - **Base Case**: Stops the recursion.
  - **Recursive Case**: The function calls itself with a modified input.
- Recursion can simplify the code and make the solution more elegant.
- However, it is important to write it correctly to avoid infinite loops.

Searching for a Key in Boxes



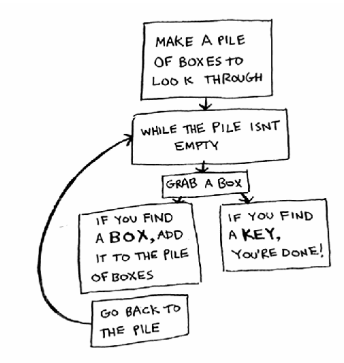


Imagine you have a large pile of boxes, some of which contain other boxes, and your goal is to find a key.

The problem can be approached in two ways:

- Iterative approach (manually checking every box one by one).
- Recursive approach (delegating the search to smaller subsets of boxes).

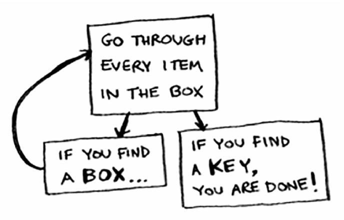
Searching for a Key in Boxes


MAKE A PILE OF BOXES TO LOOK THROUGH
WHILE THE PILE ISNT EMPTY
GRAB A BOX
IF YOU FIND A BOX, ADD IT TO THE PILE OF BOXES
IF YOU FIND A KEY, YOU'RE DONE!
GO BACK TO THE PILE

Example of iterative approach:

1. Make a pile of boxes to look through.
2. Grab a box, and look through it.
3. If you find a box, add it to the pile to look through later.
4. If you find a key, you're done!
5. Repeat.

Searching for a Key in Boxes


GO THROUGH EVERY ITEM IN THE BOX
IF YOU FIND A BOX...
IF YOU FIND A KEY, YOU ARE DONE!

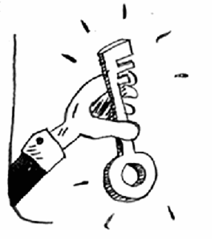
Example of recursive approach:

1. Look through the box.
2. If you find a box, go to step 1.
3. If you find a key, you're done!

Searching for a Key in Boxes

Pseudocode:

```python
def look_for_key(box):
    for item in box:
        if item.is_a_box():
            look_for_key(item) # Recursive case
        elif item.is_a_key():  # Base case
            print("Found the key!")
```

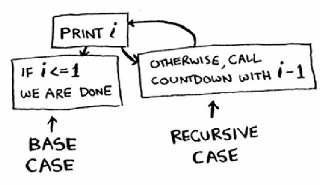


- This function checks a box for a key.
- If another box is found, it recursively searches inside it.
- If the key is found, it prints a success message.

Base Case and Recursive Case

Countdown Example:

```python
def countdown(i):
    print(i)
    if i <= 1:  # Base case
        return None
    else:
        countdown(i - 1)  # Recursive case
```



- The base case ensures the recursion stops when $i \leq 1$.
- The recursive case calls the function with $i - 1$.

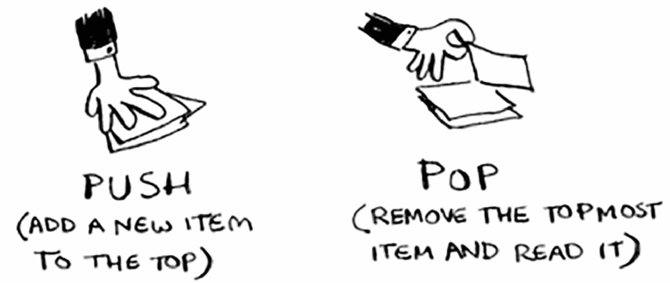
The Stack

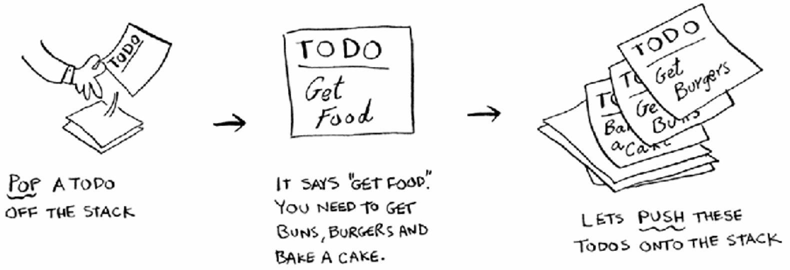
A stack is a linear data structure that follows the **LIFO** principle:

- ✎ **Last In, First Out**: The last element added to the stack is the first one to be removed.

- ✎ Think of a stack of sticky notes:
  - ✎ You can only add or remove sticky notes from the top of the stack.

The Stack


PUSH
(ADD A NEW ITEM TO THE TOP)
POP
(REMOVE THE TOPMOST ITEM AND READ IT)


TODO
POP A TODO OFF THE STACK
TODO
Get Food
IT SAYS "GET FOOD". YOU NEED TO GET BUNS, BURGERS AND BAKE A CAKE.
TODO
Bake a Cake
TODO
Get Buns
TODO
Get Burgers
LETS PUSH THESE TODOS ONTO THE STACK

Understanding the Call Stack

- ✎ Every function call is saved onto a stack called the **call stack**.
- ✎ The call stack helps the computer remember the state of each function call.
- ✎ When a function finishes, its data is removed from the stack.
- ✎ Recursive functions use the call stack heavily.

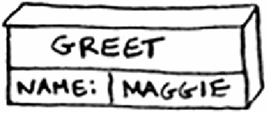
Understanding the Call Stack

```python
def greet(name):
    print(f'Hello {name}!')
    greet2(name)
    print("see you ...")
    bye()

def greet2(name):
    print(f'How are you, {name}?')

def bye():
    print("bye!")
```

greet('MAGGIE')



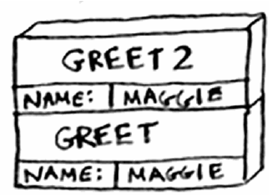
Understanding the Call Stack

```python
def greet(name):
    print(f'Hello {name}!')
    greet2(name)
    print("see you ...")
    bye()

def greet2(name):
    print(f'How are you, {name}?')

def bye():
    print("bye!")
```

greet('MAGGIE')



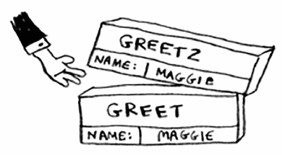
Understanding the Call Stack

```python
def greet(name):
    print(f'Hello {name}!')
    greet2(name)
    print("see you ...")
    bye()

def greet2(name):
    print(f'How are you, {name}?')

def bye():
    print("bye!")
```

greet('MAGGIE')



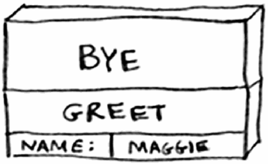
Understanding the Call Stack

```python
def greet(name):
    print(f'Hello {name}!')
    greet2(name)
    print("see you ...")
    bye()

def greet2(name):
    print(f'How are you, {name}?')

def bye():
    print("bye!")
```

greet('MAGGIE')



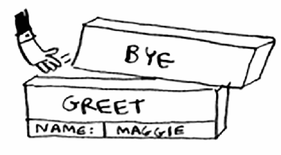
Understanding the Call Stack

```python
def greet(name):
    print(f'Hello {name}!')
    greet2(name)
    print("see you ...")
    bye()

def greet2(name):
    print(f'How are you, {name}?')

def bye():
    print("bye!")
```

```python
greet('MAGGIE')
```



Example: Factorial Function

Recursive Function:

```python
def fact(x):
    if x == 1:
        return 1  # Base case
    else:
        return x * fact(x - 1)  # Recursive case
```

- Computes the factorial of a number using recursion.
- Example: `fact(3)` computes $3 \times 2 \times 1 = 6$.

Recap and Key Takeaways

- ✎ Recursion is a function calling itself.
- ✎ Every recursive function has:
  - ✎ A **base case** to stop recursion.
  - ✎ A **recursive case** to continue solving the problem.
- ✎ The call stack plays a crucial role in recursion.
- ✎ Be cautious of infinite recursion—it can lead to stack overflow.
- ✎ Use recursion when it makes the problem easier to understand.