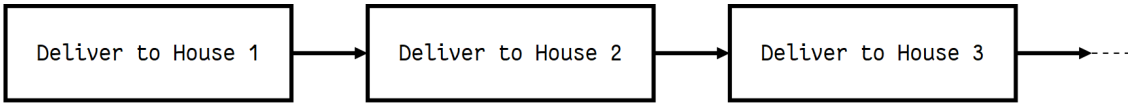
Recursion

February 18, 2021

Big problems broken into smaller chunks -that are easier to solve.



Iterative Delivery

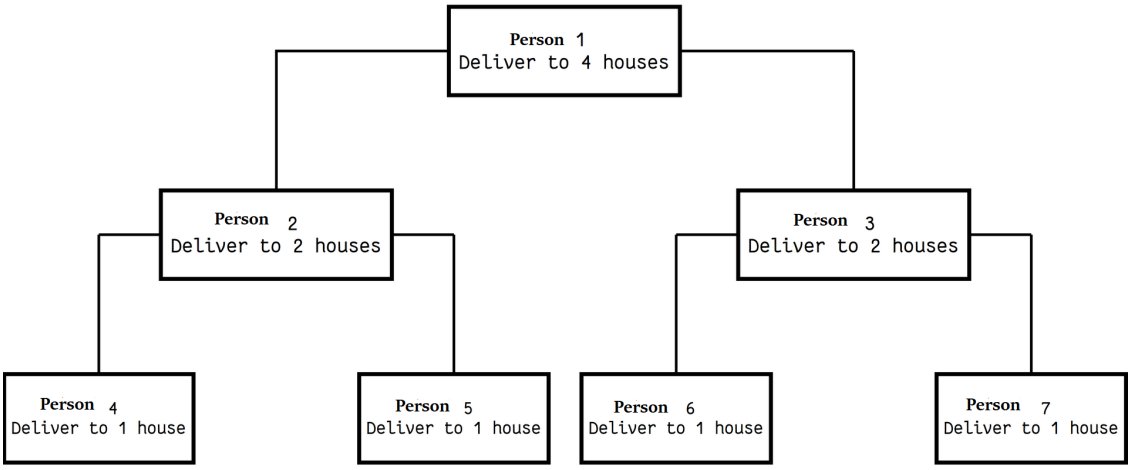
Essence of thinking recursively

```
[ ]: houses = ["Eric's house", "Kenny's house", "Kyle's house", "Stan's house"]

def deliver_presents_iteratively():
    for house in houses:
        print("Delivering presents to", house)
```

```
[ ]: deliver_presents_iteratively()
```

an algorithm with which he can divide the work of delivering presents among his assistant



Recursive Delivery

the typical structure of a recursive algorithm. If the current problem represents a simple case, solve it. If not, divide it into subproblems and apply the same strategy to them.

```
[ ]: houses = ["Eric's house", "Kenny's house", "Kyle's house", "Stan's house"]

# Each function call represents an person doing his work
def deliver_presents_recursively(houses):
    # Worker  doing his work
    if len(houses) == 1:
        house = houses[0]
        print("Delivering presents to", house)

    # Manager  doing his work
    else:
        mid = len(houses) // 2
        first_half = houses[:mid]
        second_half = houses[mid:]

        # Divides his work among two person
        deliver_presents_recursively(first_half)
        deliver_presents_recursively(second_half)
```

```
[ ]: deliver_presents_recursively(houses)
```

A recursive function is a function defined in terms of itself via self-referential expressions. two parts: base case and recursive case.

Decompose the original problem into simpler instances of the same problem. This is the recursive case:

n! = n x (n−1) x (n−2) x (n−3) x 3 x 2 x 1

n! = n x (n−1)!

As the large problem is broken down into successively less complex ones, those subproblems must eventually become so simple that they can be solved without further subdivision. This is the base case

n! = n x (n−1)!

n! = n x (n−1) x (n−2)!

n! = n x (n−1) x (n−2) x (n−3)!

n! = n x (n−1) x (n−2) x (n−3) x 3!

n! = n x (n−1) x (n−2) x (n−3) x 3 x 2!

n! = n x (n−1) x (n−2) x (n−3) x 3 x 2 x 1!

Here, 1! is our base case, and it equals 1.

[ ]:
```python
def factorial_recursive(n):
    # Base case: 1! = 1
    if n == 1:
        return 1

    # Recursive case: n! = n * (n-1)!
    else:
        return n * factorial_recursive(n-1)
```

[ ]:
```python
factorial_recursive(5)
```

Few more examples of Recursion

[ ]:
```python
def sum_recursive(nums):
    if len(nums) == 0:
        return 0

    last_num = nums.pop()
    return last_num + sum_recursive(nums)
```

[ ]:
```python
sum_recursive([1,2,3,4,5])
```

```
Integers:   0, 1, 2, 3, 4, 5, 6, 7
Fibonacci:  0, 1, 1, 2, 3, 5, 8, 13
```

[ ]:
```python
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)
```

[ ]:
```python
fibonacci(3)
```

Recursion allows us to break a large task down to smaller tasks by repeatedly calling itself. A recursive function requires a base case to stop execution, and the call to itself which gradually leads to the function to the base case

0.1 Additional video links

https://www.youtube.com/watch?v=B0NtAFf4bvU

https://www.youtube.com/watch?v=hNP72JdOIgY

https://www.youtube.com/watch?v=ygK0YON10sQ