Universidad Carlos III de Madrid

Unit 4
Recursion

Data Structures and Algorithms

Agenda

- **What is recursion?**
- **Some examples of recursion.**
  - Factorial function
  - Multiply 2 numbers using addition
  - Calculating the sum of an array of integers
  - Power
  - Binary Search

What is recursion?

- Recursion is a **stratigy for solving problems**.
- Recursion is an **alternative to iteration**. Any recursive function can be solved using iteration.
- A recursive method **requires fewer lines** than using a loop.
- **Loops are more efficient** (recursion consumes more memory).
- Some problems are **very difficult to solve by iteration.**

```
for (i=0; i < cond; ++i){
Statements
.........
}
```

more efficient

```
myFunction():
if (cond)
    ..............
else:
    myFunction()
```

fewer lines

What is recursion?

- Recursion divides a complex problem **into smaller sub-problems that can be solved directly**.

divide

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1!$

$1! = 1 \times 0!$

$0!$

smaller sub-problems

Base Case vs Recursive Case

- The Case base is a **smaller sub-problems that can be solved directly**.
- The recursive case is where the **function calls itself again and again** until it reaches the base case.

$$\left.\begin{array}{l} 4! = 4 \times 3! \\ 3! = 3 \times 2! \\ 2! = 2 \times 1! \\ 1! = 1 \times 0! \end{array}\right\} \textbf{recursive case}$$

$$0! \longleftarrow \begin{array}{l} \mathbf{0! = 1} \\ \textbf{smaller sub-problems} \end{array}$$

Example: Factorial function

Find the factorial of 4! ?

divide

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1 \times 0!$$

$$0!$$

Example: Factorial function

Find the factorial of 4! ?

divide

$4! = 4 \times 3! \Rightarrow 24$

$3! = 3 \times 2! \Rightarrow 6$

$2! = 2 \times 1! \Rightarrow 2$

$1! = 1 \times 0! \Rightarrow 1$

$0! \Rightarrow 1$

Combine

Example: Factorial function

Find the factorial of 4! ?

divide

$4! = 4 \times 3! \Rightarrow 24$

$3! = 3 \times 2! \Rightarrow 6$

$2! = 2 \times 1! \Rightarrow 2$

$1! = 1 \times 0! \Rightarrow 1$

RECURSIVE CASES

$0! \Rightarrow 1$

BASE CASE
solved directly

Example: Factorial function

Recursive definition

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases}$$

```
def factorial(n):
  if n==0:#base case
    return 1
  else: #recursive case
    return n*factorial(n-1)
```

Big-O for factorial function?

O(n): There is n+1 recursive calls (each one counts as 1 operation).

Example: Multiply 2 numbers using addition

$$5 \times 3 = 5 + 5 + 5 = 15$$

Example: Multiply 2 numbers using addition

divide

$$5 \times 3 = 5 + (5 \times 2)$$

$$5 \times 2 = 5 + (5 \times 1)$$

$$5 \times 1 = 5$$

Example: Multiply 2 numbers using addition

divide

$5 \times 3 = 5 + (5 \times 2) = 15$

$5 \times 2 = 5 + (5 \times 1) = 10$

$5 \times 1 = 5$

Combine

Base Case
n=1 return x

Recursive case
x+(x, n-1)

Example: Multiply 2 numbers using addition

Recursive definition

$$x * n = \begin{cases} x, & if\ n = 1 \\ x + (x, n - 1) & if\ n \geq 1 \end{cases}$$

```
def multiplyRec(x, n):
        if (n==1):
                return x
        else:
        return x + multiplyRec(x, n-1)
```

O(?)

Example: Calculating the sum of an array of integers.

Find sum of [1,3,5,7,9]?

sum [1,3,5,7,9]	= 1 + sum [3,5,7,9]
sum [3,5,7,9]	= 3 + sum [5,7,9]
sum [5,7,9]	= 5 + sum [7,9]
sum [7,9]	= 7 + sum [9]
sum [9]	= 9

Example: Calculating the sum of an array of integers.

Find sum of [1,3,5,7,9]?

sum [1,3,5,7,9]	= 1 + sum [3,5,7,9]	= 25
sum [3,5,7,9]	= 3 + sum [5,7,9]	= 24
sum [5,7,9]	= 5 + sum [7,9]	= 21
sum [7,9]	= 7 + sum [9]	= 16
sum [9]		= 9

Example: Calculating the sum of an array of integers.

Recursive definition

$$\sum_{i=0}^{n} a_i = \begin{cases} a_0, & if\ n = 1 \\ a_0 + (a_{1,n-1}) & if\ n > 1 \end{cases}$$

```
def sumArray (a):
 if len(a)==1:
  return a[0]
 else:
  return a[0] + sumArray(a[1:])
```

Time complexity: O(n)

Example: power (base, exponente)

- Power function: = 3^4, power(x,n)=x^n

power (3, 4)	= 3 * power (3, 3)
power (3, 3)	= 3 * power (3, 2)
power (3, 2)	= 3 * power (3, 1)
power (3, 1)	= 3 * power (3, 0)
power (3, 0)	= 1

Example: power (base, exponente)

- Power function: = 3^4, power(x,n)=x^n

power (3, 4)	= 3 * power (3, 3)	= 81
power (3, 3)	= 3 * power (3, 2)	= 27
power (3, 2)	= 3 * power (3, 1)	= 9
power (3, 1)	= 3 * power (3, 0)	= 3
power (3, 0)		= 1

Example: power (base, exponente)

Recursive definition

$$x^n = \begin{cases} 1, & if\ n = 0 \\ x * (x, n-1) & if\ n > 1 \end{cases}$$

```
def power(x, n):
 if n==0:
  return 1
 else:
  return x*power(x,n-1)
```

Time complexity: O(n)

Example: Binary search

- Input: a <u>sorted</u> array of integers and a number

x = 23

	0	1	2	3	4	5	6	7	8
A	**2**	**5**	**8**	**10**	**13**	**20**	**23**	**50**	**90**

start =0

end = 8

mid = (0 + 8) / 2 = 4

1st Iteration

1) If x == A[4], Found!!!
2) If x < A[4], search from start to mid-1
3) **If x > A[4], search from mid+1 to end**

Example: Binary search

- Input: a sorted array of integers and a number

x = 23

	0	1	2	3	4	5	6	7	8
A	2	5	8	10	13	**20**	**23**	**50**	**90**

start=5 end=8

2st Iteration

mid = (5 + 8) / 2= 6

1) **If x == A[6], Found!!!**
2) If x < A[6], search from start to mid-1
3) If x > A[6], search from mid+1 to end

Example: Binary search

- Input: a <u>sorted</u> array of integers and a number

x = 7 (which does not exist in the list)

	0	1	2	3	4	5	6	7	8
A	**2**	**5**	**8**	**10**	**13**	**20**	**23**	**50**	**90**

start=0 end=8

mid = (0 + 8) / 2 = 4

1st Iteration

1) If x == A[4], Found!!!
2) **If x < A[4], search from start to mid-1**
3) If x > A[4], search from mid+1 to end

Example: Binary search

- Input: a sorted array of integers and a number

x = 7 (which does not exist in the list)

	0	1	2	3	4	5	6	7	8
A	**2**	**5**	**8**	**10**	**13**	**20**	**23**	**50**	**90**

start=0 end=3

mid=(0+3)/2=1

2^{nd} Iteration

1) If x == A[1], Found!!!
2) If x < A[1], search from start to mid-1
3) **If x > A[1], search from mid+1 to end**

Example: Binary search

- Input: a sorted array of integers and a number

x = 7 (which does not exist in the list)

	0	1	2	3	4	5	6	7	8
A	2	5	8	10	13	20	23	50	90

start=2 end=3

mid=(2+3)/2=2

3rd Iteration

1) If x == A[2], Found!!!
2) **If x < A[2], search from start to mid-1**
3) If x > A[2], search from mid+1 to end

Example: Binary search

- Input: a sorted array of integers and a number

x = 7 (which does not exist in the list)

	0	1	2	3	4	5	6	7	8
A	**2**	**5**	**8**	**10**	**13**	**20**	**23**	**50**	**90**
		end=1	start=2						

4th Iteration

start > end!!! : the array does not contain it!!!

Example: implementation of binary search

1) If x == A[mid], Found!!! Base case!!!
2) If x < A[mid], search from start to mid-1
3) If x > A[mid], search from mid+1 to end

Recursive cases!!!

```
def binary_search(data,x):
  if len(data)==0:
    return False

  #integer division
  mid=len(data)//2

  if x==data[mid]:   #base case
    return True #found!!!

  elif x<data[mid]:  #recursive case,
    #search at the first half of the array
    return binary_search(data[0:mid],x)

  else:#x>data[mid], recursive case
    #search at the second half of the array
    return binary_search(data[mid+1:],x)
```

Example: Analysis of binary search function

➢ Big-Oh for binary search function?

At each iteration, the array is divided by half.

- At **Iteration 1**, Length of array = $\mathbf{n/2^1}$
- At **Iteration 2**, Length of array = $\mathbf{n/2^2}$
- At **Iteration 3**, Length of array = $\mathbf{n/2^3}$

  ....

- After **k** divisions, the **length of array becomes 1**

  Length of array = $\mathbf{n/2^k = 1}$

  $\mathbf{n = 2^k}$

  $\mathbf{k = \log_2 (n)}$

Types of recursion

- **Linear Recursion**: a recursive call could produce at most one new recursive call. Example: **Factorial, Power, Binary Search, etc.**
- **Binary recursion**: a recursive call can generate two new recursive calls. Example, **Fibonacci Numbers.**
- **Multiple Recursion**: a recursive call can generate three or more recursive calls. Example: **exploring a file system.**

Binary Recursion:

- **Fibonacci Numbers**

Binary Recursion: Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$$Fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ Fib(n-1) + Fib(n-2) & \text{if } n>1 \end{cases}$$

```
def fib(n):
    if n<=1:
        return n;
    else:
        return fib(n-1) + fib(n-2)
```

Is it an efficient way to compute fib(50)?

Binary Recursion: Fibonacci Numbers

No, no, no! This code is inefficient.

$O(2^n)$