lezione17

November 20, 2023

1 Fondamenti di Programmazione

Andrea Sterbini

lezione 17 - 20 novembre 2023

2 RECAP:

- Analisi OOP
- Ereditarietà
- Esempio di gerarchia di figure disegnate con Turtle
- Ricorsione e sue proprietà
- Esempi: albero frattale, fattoriale, conigli di Fibonacci

3 OGGI: altri esempi di ricorsione

[1]:
```
# carico il decoratore che stampa le chiamate ed uscite di una funzione␣
↪ricorsiva
from rtrace import trace
```

[59]:
```
# mi costruisco una lista di valori casuali
import random
lista = random.choices(range(-10000,10001), k=10)
```

4 Somma ricorsiva di una lista (svolta in uscita)

- **caso base**: se la lista è vuota la somma è 0
- **se la lista non è vuota**: sommiamo il primo elemento alla somma del resto della lista

[54]:
```
@trace(True)
def somma_ricorsiva(L):
    if L:           # se caso ricorsivo
        primo, *resto = L
        return primo + somma_ricorsiva(resto)
    else:
        return 0    # altrimenti caso base
```

```
somma = somma_ricorsiva.trace(lista)
somma2 = sum(lista)
somma, somma2
```

```
------------------- Starting recursion -------------------
 entering       somma_ricorsiva([-5609, -4883, -4645, -1475, 1886, -4765, -8477,
-7863, 5173, -7811],)
|-- entering    somma_ricorsiva([-4883, -4645, -1475, 1886, -4765, -8477, -7863,
5173, -7811],)
|--|-- entering somma_ricorsiva([-4645, -1475, 1886, -4765, -8477, -7863, 5173,
-7811],)
|--|--|-- entering      somma_ricorsiva([-1475, 1886, -4765, -8477, -7863, 5173,
-7811],)
|--|--|--|-- entering   somma_ricorsiva([1886, -4765, -8477, -7863, 5173,
-7811],)
|--|--|--|--|-- entering        somma_ricorsiva([-4765, -8477, -7863, 5173,
-7811],)
|--|--|--|--|--|-- entering     somma_ricorsiva([-8477, -7863, 5173, -7811],)
|--|--|--|--|--|--|-- entering  somma_ricorsiva([-7863, 5173, -7811],)
|--|--|--|--|--|--|--|-- entering       somma_ricorsiva([5173, -7811],)
|--|--|--|--|--|--|--|--|-- entering    somma_ricorsiva([-7811],)
|--|--|--|--|--|--|--|--|--|-- entering somma_ricorsiva([],)
|--|--|--|--|--|--|--|--|--|-- exiting  somma_ricorsiva([],)    returns 0
|--|--|--|--|--|--|--|--|--|-- exiting     somma_ricorsiva([-7811],)       returns
-7811
|--|--|--|--|--|--|--|--|-- exiting        somma_ricorsiva([5173, -7811],) returns
-2638
|--|--|--|--|--|--|--|-- exiting   somma_ricorsiva([-7863, 5173, -7811],)  returns
-10501
|--|--|--|--|--|-- exiting      somma_ricorsiva([-8477, -7863, 5173, -7811],)
returns -18978
|--|--|--|--|-- exiting somma_ricorsiva([-4765, -8477, -7863, 5173, -7811],)
returns -23743
|--|--|--|-- exiting    somma_ricorsiva([1886, -4765, -8477, -7863, 5173,
-7811],)      returns -21857
|--|--|-- exiting       somma_ricorsiva([-1475, 1886, -4765, -8477, -7863, 5173,
-7811],)       returns -23332
|--|-- exiting  somma_ricorsiva([-4645, -1475, 1886, -4765, -8477, -7863, 5173,
-7811],)        returns -27977
|-- exiting     somma_ricorsiva([-4883, -4645, -1475, 1886, -4765, -8477, -7863,
5173, -7811],) returns -32860
 exiting        somma_ricorsiva([-5609, -4883, -4645, -1475, 1886, -4765, -8477,
-7863, 5173, -7811],)  returns -38469
-------------------- Ending recursion --------------------
Num calls: 11
```

[54]: (-38469, -38469)

[55]:
```
@trace()
def somma_ricorsiva_distruttiva(L):
    if L:           # se caso ricorsivo
        ultimo = L.pop()
        return ultimo + somma_ricorsiva_distruttiva(L)
    else:
        return 0    # altrimenti caso base

somma3 = somma_ricorsiva_distruttiva.trace(lista.copy())
# controlliamo che venga il risultato giusto
somma2, somma3
```

```
------------------- Starting recursion -------------------
 entering       somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863, 5173, -7811],)
|-- entering    somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863, 5173],)
|--|-- entering somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863],)
|--|--|-- entering      somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475,
1886, -4765, -8477],)
|--|--|--|-- entering   somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475,
1886, -4765],)
|--|--|--|--|-- entering        somma_ricorsiva_distruttiva([-5609, -4883,
-4645, -1475, 1886],)
|--|--|--|--|--|-- entering     somma_ricorsiva_distruttiva([-5609, -4883,
-4645, -1475],)
|--|--|--|--|--|--|-- entering  somma_ricorsiva_distruttiva([-5609, -4883,
-4645],)
|--|--|--|--|--|--|--|-- entering       somma_ricorsiva_distruttiva([-5609,
-4883],)
|--|--|--|--|--|--|--|--|-- entering    somma_ricorsiva_distruttiva([-5609],)
|--|--|--|--|--|--|--|--|--|-- entering somma_ricorsiva_distruttiva([],)
|--|--|--|--|--|--|--|--|--|-- exiting  somma_ricorsiva_distruttiva([],)
returns 0
|--|--|--|--|--|--|--|--|-- exiting     somma_ricorsiva_distruttiva([-5609],)
returns -5609
|--|--|--|--|--|--|--|-- exiting        somma_ricorsiva_distruttiva([-5609,
-4883],)    returns -10492
|--|--|--|--|--|--|-- exiting   somma_ricorsiva_distruttiva([-5609, -4883,
-4645],)     returns -15137
|--|--|--|--|--|-- exiting      somma_ricorsiva_distruttiva([-5609, -4883,
-4645, -1475],)      returns -16612
|--|--|--|--|-- exiting         somma_ricorsiva_distruttiva([-5609, -4883,
-4645, -1475, 1886],)        returns -14726
|--|--|--|-- exiting    somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475,
```

```
1886, -4765],) returns -19491
|--|--|-- exiting       somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475,
1886, -4765, -8477],)  returns -27968
|--|-- exiting  somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863],)   returns -35831
|-- exiting     somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863, 5173],)     returns -30658
 exiting        somma_ricorsiva_distruttiva([-5609, -4883, -4645, -1475, 1886,
-4765, -8477, -7863, 5173, -7811],)      returns -38469
-------------------- Ending recursion --------------------
Num calls: 11
```

[55]: `(-38469, -38469)`

4.1 Somma iterativa di una lista

- accumuliamo la somma con un ciclo

[5]:
```python
def somma_iter(L : list[int]) -> int:
    somma = 0
    N = len(L)
    for i in range(N):
        somma += L[i]
    return somma

somma_iter(lista), sum(lista)
```

[5]: `(-38469, -38469)`

4.2 Somma ricorsiva in avanti (simulando il ciclo)

- le variabili di stato, **somma**, **i** ed **N**
  - diventano argomenti della funzione
  - ad ogni step le aggiorniamo nella chiamata ricorsiva

[60]:
```python
@trace()
def _somma_ric_avanti(L : list[int], i : int,
                      N : int, somma : int) -> int :
    if i==N:
        return somma
    else:
        return _somma_ric_avanti(L, i+1, N, somma + L[i]) # AGGIORNAMENTO

def somma_ric_avanti(L):
    return _somma_ric_avanti(L, 0, len(L), 0)

somma_ric_avanti(lista)
_somma_ric_avanti.trace(lista, 0, len(lista), 0)
```

```
------------------- Starting recursion -------------------
 entering       _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 0, 10, 0)
|-- entering    _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 1, 10, -3711)
|--|-- entering _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 2, 10, 3169)
|--|--|-- entering      _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784,
-6567, -5864, -3613, 2242, -3865], 3, 10, 225)
|--|--|--|-- entering   _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784,
-6567, -5864, -3613, 2242, -3865], 4, 10, 9158)
|--|--|--|--|-- entering        _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 5, 10, 17942)
|--|--|--|--|--|-- entering     _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 6, 10, 11375)
|--|--|--|--|--|--|-- entering  _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 7, 10, 5511)
|--|--|--|--|--|--|--|-- entering       _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 8, 10, 1898)
|--|--|--|--|--|--|--|--|-- entering    _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 9, 10, 4140)
|--|--|--|--|--|--|--|--|--|-- entering _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 10, 10, 275)
|--|--|--|--|--|--|--|--|--|-- exiting  _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 10, 10, 275)      returns 275
|--|--|--|--|--|--|--|--|-- exiting     _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 9, 10, 4140)      returns 275
|--|--|--|--|--|--|--|-- exiting        _somma_ric_avanti([-3711, 6880, -2944,
8933, 8784, -6567, -5864, -3613, 2242, -3865], 8, 10, 1898)      returns 275
|--|--|--|--|--|--|-- exiting   _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 7, 10, 5511)      returns 275
|--|--|--|--|--|-- exiting      _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 6, 10, 11375)     returns 275
|--|--|--|--|-- exiting         _somma_ric_avanti([-3711, 6880, -2944, 8933,
8784, -6567, -5864, -3613, 2242, -3865], 5, 10, 17942)     returns 275
|--|--|--|-- exiting    _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784,
-6567, -5864, -3613, 2242, -3865], 4, 10, 9158)      returns 275
|--|--|-- exiting       _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784,
-6567, -5864, -3613, 2242, -3865], 3, 10, 225)       returns 275
|--|-- exiting  _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 2, 10, 3169)      returns 275
|-- exiting     _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 1, 10, -3711)     returns 275
 exiting        _somma_ric_avanti([-3711, 6880, -2944, 8933, 8784, -6567, -5864,
-3613, 2242, -3865], 0, 10, 0) returns 275
------------------- Ending recursion -------------------
Num calls: 11
```

[60]: 275

4.3 Stampa di una lista

- **in avanti** (prima del passo ricorsivo)
- **indietro** (dopo il passo ricorsivo)
- **avanti poi indietro** (prima e dopo)

Analisi: - la lista vuota non stampa nulla (caso base) - una volta stampato il primo elemento va stampato il resto (passo ricorsivo)

[61]:
```
# per scandire e stampare una lista in avanti
def stampa_in_avanti(L):
    if L:
        primo, *resto = L
        print(primo, end=' ')
        stampa_in_avanti(resto)
    # else caso base non stampo nulla

stampa_in_avanti(lista)
print()
print(lista)
```

```
-3711 6880 -2944 8933 8784 -6567 -5864 -3613 2242 -3865
[-3711, 6880, -2944, 8933, 8784, -6567, -5864, -3613, 2242, -3865]
```

[62]:
```
# per scandire e stampare una lista a rovescio
def stampa_dalla_fine(L):
    if L:
        primo, *resto = L
        stampa_dalla_fine(resto)
        print(primo, end=' ')

stampa_dalla_fine(lista)
print()
print(lista)
# TODO per scandire una lista sia in avanti che a rovescio
#      basta stampare sia prima della ricorsione che dopo
```

```
-3865 2242 -3613 -5864 -6567 8784 8933 -2944 6880 -3711
[-3711, 6880, -2944, 8933, 8784, -6567, -5864, -3613, 2242, -3865]
```

4.4 Altro esempio: Massimo Comun Divisore di x, y interi positivi

Ovvero dobbiamo trovare quel'intero M tale che: - $x = M * k$ - $y = M * j$ - con $k, j >= 1$ (e senza fattori comuni)

4.4.1 Quali sono le proprietà di x ed y?

- se **x == y** allora esistono **k == j == 1** e **M == x** (ecco un buon **caso base**!)

- altrimenti proviamo a sottrarre il minore dal maggiore (assumiamo sia y)
  - $z = x - y = Mk - Mj = M(k - j)$ quindi **z** e **y** hanno lo stesso **MCD**, e inoltre z è più piccolo di x (ecco la nostra **riduzione**!)
    - se prendiamo come *dimensione del problema* la somma $x + y$
      - ad ogni passo si riduce la somma $x + y = M(k + j)$ di almeno $y = Mj$ (il più piccolo dei due)
      - quindi anche $k + j$ diminuisce mano a mano fino a diventare 1
      - perchè sottraendo un numero più piccolo diminuiamo la somma ma non si può andare nei negativi nè sullo 0
  - quindi a forza di sottrarre arriveremo per forza a $j = k = 1$ ovvero al caso base (ed ecco la **convergenza**)
  - una volta trovato **M** abbiamo già la soluzione di ciascun caso **più grande** (**ricomposizione**)

Ottimizzazione: invece di sottrarre y da x calcoliamone il resto -> algoritmo di **Euclide**

[64]:
```
# Sfruttiamo la definizione ricorsiva del problema
# per dare una implementazione ricorsiva
@trace()
def GCD(x : int, y : int) -> int :
    # FIXME: controllare che siano  interi E positivi
    if x == y:
        return x
    else:
        if x>y:
            return GCD(x-y, y) # sottraggo il minore dal maggiore e tengo␣
 ↪l'altro
        else:
            return GCD(y-x, x) # sottraggo il minore dal maggiore e tengo␣
 ↪l'altro

GCD.trace(32, 18)
```

```
------------------- Starting recursion -------------------
 entering       GCD(32, 18)
|-- entering    GCD(14, 18)
|--|-- entering GCD(4, 14)
|--|--|-- entering      GCD(10, 4)
|--|--|--|-- entering   GCD(6, 4)
|--|--|--|--|-- entering        GCD(2, 4)
|--|--|--|--|--|-- entering     GCD(2, 2)
|--|--|--|--|--|-- exiting      GCD(2, 2)       returns 2
|--|--|--|--|-- exiting         GCD(2, 4)       returns 2
|--|--|--|-- exiting    GCD(6, 4)       returns 2
|--|--|-- exiting       GCD(10, 4)      returns 2
|--|-- exiting  GCD(4, 14)      returns 2
|-- exiting     GCD(14, 18)     returns 2
 exiting        GCD(32, 18)     returns 2
```

```
-------------------- Ending recursion --------------------
Num calls: 7
```

[64]:
```
2
```

[66]:
```
## Non è difficile farne una versione iterativa

def GCD_iter(x : int, y : int) -> int :
    while x != y:
        print(x,y)
        if x > y:
            x -= y
        else:
            y -= x
    return x

print(GCD_iter(75,45))
```

```
75 45
30 45
30 15
15
```

4.5 Esempio: Check se una stringa/lista è palindroma (si legge uguale in senso inverso)

4.5.1 soluzioni iterative

1. rovescio e confronto
2. scandisco gli indici degli elementi partendo dagli estremi e li confronto
   - se trovo una differenza torno False
   - se arrivo a metà senza differenze torno True

[69]:
```
from typing import Sequence
## Rovescio e confronto
def palindromaP_iter1(sequenza : Sequence):
    rovesciata = sequenza[::-1]
    return rovesciata == sequenza

# leggermente inefficiente, costruisco una nuova sequenza e confronto N coppie␣
 ↪di elementi
# (potrei confrontarne solo N//2)
palindromaP_iter1('amoRoma')
```

[69]:
```
True
```

[71]:
```
## Versione iterativa

def palindromaP_iter2(sequenza : Sequence ) -> bool :
```

```
    for i in range(len(sequenza)//2):  # scandisco la prima metà degli indici
        print('comparing', sequenza[i], sequenza[-i-1], sequenza[i] ==␣
 ↪sequenza[-i-1])
        if sequenza[i] != sequenza[-i-1]:  # e confronto il primo con l'ultimo .
 ↪... eccetera ... usando indici negativi
            return False
    return True

def palindromaP_iter3(sequenza : Sequence ) -> bool :
    "con una list comprehension e 'all'"
    return all(sequenza[i] == sequenza[-i-1] for i in range(len(sequenza)//2))


#palindromaP_iter2('amoRoma')
palindromaP_iter2([1, 2, 3, 4, 5, 4, 3, 2, 1])
```

```
comparing 1 1 True
comparing 2 2 True
comparing 3 3 True
comparing 4 4 True
```

[71]: True

[74]:
```
def palindroma_iter2(sequenza):
    "lo stesso usando sue indici e spostandoli mano a mano verso il centro"
    inizio = 0
    fine   = len(sequenza)-1
    while inizio < fine:
        print('comparing', sequenza[inizio], sequenza[fine], sequenza[inizio]␣
 ↪== sequenza[fine])
        if sequenza[inizio] != sequenza[fine]:
            return False
        inizio += 1      # l'inizio avanza di un passo
        fine   -= 1      # la fine arretra di un passo
    return True

palindroma_iter2([1, 2, 3, 4, 5, 4, 3, 2, 1])
```

```
comparing 1 1 True
comparing 2 2 True
comparing 3 3 True
comparing 4 4 True
```

[74]: True

4.5.2 soluzione ricorsiva

- **casi base**: ogni sequenza di lunghezza 0 o 1 è già palindroma
- se è lunga 2 o più elementi il primo e l'ultimo devono essere uguali

- se non lo sono NON è palindroma (altro **caso base**)
- se lo sono deve essere palindromo anche il resto, tolto primo ed ultimo carattere
  - (togliere i 2 caratteri = **riduzione**)
  - a forza di togliere 2 caratteri si arrivarà sempre ad averne 1 o 0 (**convergenza**)
  - la stringa è palindroma se sono uguali primo e ultimo AND la sottostringa è palindroma (**costruzione della soluzione dalla sottosoluzione** + calcolo locale)

[77]:
```
@trace()
def palindromaP(sequenza : Sequence ) -> bool :
    "predicato che verifica se una sequenza è palindroma"
    if len(sequenza) < 2:
        return True
    if sequenza[0] != sequenza[-1]:
        return False
    else:
        return palindromaP(sequenza[1:-1])

palindromaP.trace('amoRoma')

# NOTA: è un po' inefficiente perchè crea tante sottosequenze
```

```
------------------- Starting recursion -------------------
 entering       palindromaP('amoRoma',)
|-- entering    palindromaP('moRom',)
|--|-- entering palindromaP('oRo',)
|--|--|-- entering      palindromaP('R',)
|--|--|-- exiting       palindromaP('R',)       returns True
|--|-- exiting  palindromaP('oRo',)     returns True
|-- exiting     palindromaP('moRom',)   returns True
 exiting        palindromaP('amoRoma',) returns True
-------------------- Ending recursion --------------------
Num calls: 4
```

[77]:
```
True
```

[80]:
```
# versione che non crea sottostringhe usando gli indici inizio e fine per␣
 ↪sapere quali caratteri confrontare
@trace()
def _palindromaP2(sequenza : Sequence, inizio : int, fine : int ) -> bool :
    if inizio >= fine:                          # se si sono raggiunti non ho␣
 ↪trovato differenze (caso base)
        return True                             # la stringa è palindroma
    if sequenza[inizio] != sequenza[fine]:      # se sono diversi (caso base)
        return False                            # # la stringa NON è palindroma
    return _palindromaP2(sequenza, inizio+1, fine-1)   # altrimenti gli estremi␣
 ↪sono OK, tutto dipende dal resto, sposto gli indici

def palindromaP2(sequenza):
```

```
    return palindromaP2(sequenza, 0, len(sequenza)-1)   # inizio da 0 e da N-1


_palindromaP2.trace('amoRoma', 0, 6)
```

```
------------------- Starting recursion -------------------
 entering       _palindromaP2('amoRoma', 0, 6)
|-- entering    _palindromaP2('amoRoma', 1, 5)
|--|-- entering _palindromaP2('amoRoma', 2, 4)
|--|--|-- entering      _palindromaP2('amoRoma', 3, 3)
|--|--|-- exiting       _palindromaP2('amoRoma', 3, 3)  returns True
|--|-- exiting  _palindromaP2('amoRoma', 2, 4)  returns True
|-- exiting     _palindromaP2('amoRoma', 1, 5)  returns True
 exiting        _palindromaP2('amoRoma', 0, 6)  returns True
------------------- Ending recursion -------------------
Num calls: 4
```

[80]: True

4.6 Esploriamo un albero di directory: cerchiamo tutti i file .txt e la loro dimensione

Torniamo un dizionario **path del file -> dimensioni** - una directory contiene files (caso base) e sottodirectory (caso ricorsivo) - ogni volta che esaminiamo una sottodirectory abbiamo un problema simile a quello iniziale, e più piccolo (riduzione) - a forza di scendere arriveremo in una sottodirectory che contiene solo file (convergenza) - i file trovati nelle sottodirectory vanno raccolti assieme a quelli della dir iniziale (composizione)

Per esaminare directory e files si usa la libreria **os** (os.listdir, os.path.isdir, os.path.getsize)

[83]:
```
import os

@trace()
def cerca_file_sizes(directory : str) -> dict[str, int] :
    "cerco tutti i file '.txt' e ne ritorno le dimensioni"
    risultato = {}
    for nome in os.listdir(directory):                  # esamino tutti i nomi␣
 ↪nella directory
        if nome[0] in '_.': continue                    # ignoro file e dir che␣
 ↪iniziano per '.' oppure '_'
        fullname = directory + '/' + nome               # costruisco il path␣
 ↪completo del file/dir
        if os.path.isdir(fullname):                     # se è una directory␣
 ↪devo cercarci dentro
            trovati = cerca_file_sizes(fullname)        # trovo tutti i file␣
 ↪della sottodirectory
            risultato.update(trovati)                   # aggiorno il dizionario␣
 ↪con ciò che ho trovato nella sottodirectory
```

```
        elif nome.endswith('.txt'):                    # altrimenti è un file e␣
 ↪se finisce con '.txt'
            size = os.path.getsize(fullname)           # ne trovo le dimensioni
            risultato[fullname] = size                 # e aggiorno il␣
 ↪dizionario    path -> dimensioni
    return risultato                                   # e alla fine torno il␣
 ↪risultato

cerca_file_sizes.trace('../lezione11')
```

```
------------------- Starting recursion -------------------
 entering       cerca_file_sizes('../lezione11',)
|-- entering    cerca_file_sizes('../lezione11/files',)
|-- exiting     cerca_file_sizes('../lezione11/files',) returns
{'../lezione11/files/prince.txt': 305864, '../lezione11/files/alice.txt':
167517, '../lezione11/files/results.txt': 272,
'../lezione11/files/frankenstein.txt': 448684, '../lezione11/files/holmes.txt':
594933, '../lezione11/files/alice_it.txt': 144377,
'../lezione11/files/testo2.txt': 19, '../lezione11/files/testo.txt': 146}
 exiting        cerca_file_sizes('../lezione11',)       returns
{'../lezione11/paperopoli.txt': 10, '../lezione11/files/prince.txt': 305864,
'../lezione11/files/alice.txt': 167517, '../lezione11/files/results.txt': 272,
'../lezione11/files/frankenstein.txt': 448684, '../lezione11/files/holmes.txt':
594933, '../lezione11/files/alice_it.txt': 144377,
'../lezione11/files/testo2.txt': 19, '../lezione11/files/testo.txt': 146}
-------------------- Ending recursion --------------------
Num calls: 2
```

[83]:
```
{'../lezione11/paperopoli.txt': 10,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/alice.txt': 167517,
 '../lezione11/files/results.txt': 272,
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/holmes.txt': 594933,
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/testo2.txt': 19,
 '../lezione11/files/testo.txt': 146}
```

[84]:
```
## soluzione ricorsiva che raccoglie i file
## in un dizionario fornito come argomento
## e aggiornato mano a mano che si esplora

def cerca_file_sizes2(directory : str, dizionario : dict[str, int]) -> None:
    "cerco tutti i file '.txt' e ne ritorno le dimensioni, ma aggiornando un␣
 ↪dizionario fornito dall'esterno"
    for nome in os.listdir(directory):
```

```
        if nome[0] in '_.': continue                          # ignoro file e dir␣
↪che iniziano per '.' oppure '_'
        fullname = directory + '/' + nome                     # costruisco il path␣
↪completo

        if os.path.isdir(fullname):                           # se sono nel caso␣
↪ricorsivo
            cerca_file_sizes2(fullname, dizionario)           # continuo a cercare␣
↪e passo il dizionario da aggiornare

        elif nome.endswith('.txt'):                           # altrimenti è un␣
↪file e se finisce con '.txt'
            size = os.path.getsize(fullname)                  # ne trovo le␣
↪dimensioni
            dizionario[fullname] = size                       # ed aggiorno il␣
↪dizionario
```

[85]:
```
# per usare cerca_file_sizes2 devo PRIMA creare il dizionario
D : dict[str,int] = {}
cerca_file_sizes2('../lezione11', D)     # e passarlo
D
```

[85]:
```
{'../lezione11/paperopoli.txt': 10,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/alice.txt': 167517,
 '../lezione11/files/results.txt': 272,
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/holmes.txt': 594933,
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/testo2.txt': 19,
 '../lezione11/files/testo.txt': 146}
```

4.7 come sbagliare usando gli argomenti con default

Potrebbe venire in mente di definire l'argomento del dizionario con un valore di default

```
def  cerca_file_sizes2( directory : str, dizionario : dict = {}):    # <-- notare questo valore
    ...
```

Ma questo crea effetti collaterali ed **errori difficili da riconoscere** - il valore di default **viene istanziato al momento della definizione** della funzione - viene quindi **condiviso tra tutte le chiamate** - questo non è un problema se il valore è immutabile - **diventa un casino se il valore di default è mutevole**

[86]:
```
# esempio sbagliato
def cerca_file_sizes_sbagliato(directory, dizionario = {} ):
    "cerco tutti i file '.txt' e ne ritorno le dimensioni, ma aggiornando un␣
↪dizionario CON DEFAULT!!!"
```

```
    for nome in os.listdir(directory):
        if nome[0] in '_.': continue                                # ignoro file e␣
↪dir che iniziano per '.' oppure '_'
        fullname = directory + '/' + nome                           # costruisco il␣
↪path completo
        if os.path.isdir(fullname):                                 # se sono nel␣
↪caso ricorsivo
            cerca_file_sizes_sbagliato(fullname, dizionario)  # continuo a␣
↪cercare e passo il dizionario da aggiornare
        elif nome.endswith('.txt'):                                 # altrimenti è un␣
↪file e se finisce con '.txt'
            size = os.path.getsize(fullname)                        # ne trovo le␣
↪dimensioni
            dizionario[fullname] = size                             # ed aggiorno il␣
↪dizionario
    return dizionario
```

[87]:
```
cerca_file_sizes_sbagliato('.')              # prima esecuzione, raccolgo i␣
↪file in questa dir
```

[87]:
```
{'./ita.lezione.17.txt': 72501, './eng.lezione.17.txt': 67840}
```

[88]:
```
cerca_file_sizes_sbagliato('../lezione11')   # seconda esecuzione, ritrovo␣
↪anche quelli di prima
```

[88]:
```
{'./ita.lezione.17.txt': 72501,
 './eng.lezione.17.txt': 67840,
 '../lezione11/paperopoli.txt': 10,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/alice.txt': 167517,
 '../lezione11/files/results.txt': 272,
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/holmes.txt': 594933,
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/testo2.txt': 19,
 '../lezione11/files/testo.txt': 146}
```

[22]:
```
cerca_file_sizes_sbagliato('../lezione10')   # terza esecuzione, ritrovo anche␣
↪quelli di prima!!!!
```

[22]:
```
{'./ita.lezione.17.txt': 72501,
 './eng.lezione.17.txt': 67840,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/alice.txt': 167517,
 '../lezione11/files/results.txt': 272,
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/holmes.txt': 594933,
```

```
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/testo2.txt': 19,
 '../lezione11/files/testo.txt': 146,
 '../lezione10/files/prince.txt': 305864,
 '../lezione10/files/alice.txt': 167517,
 '../lezione10/files/results.txt': 272,
 '../lezione10/files/frankenstein.txt': 448684,
 '../lezione10/files/holmes.txt': 594933,
 '../lezione10/files/alice_it.txt': 144377,
 '../lezione10/files/testo2.txt': 19,
 '../lezione10/files/testo.txt': 146,
 '../lezione10/topolino.txt': 71,
 '../lezione10/pippo.txt': 71}
```

[89]:
```
## conclusione: NON usate parametri di default mutevoli
def cerca_file_sizes_corretto(directory, dizionario = None ):
    "cerco tutti i file '.txt' e ne ritorno le dimensioni, ma aggiornando un␣
 ↪dizionario CON DEFAULT!!!"
    if dizionario is None: dizionario = {}                # ne creo uno nuovo␣
 ↪ad ogni chiamata
    for nome in os.listdir(directory):
        if nome[0] in '_.': continue                      # ignoro file e dir␣
 ↪che iniziano per '.' oppure '_'
        fullname = directory + '/' + nome                 # costruisco il path␣
 ↪completo
        if os.path.isdir(fullname):                       # se sono nel caso␣
 ↪ricorsivo
            cerca_file_sizes_corretto(fullname, dizionario)        # continuo a␣
 ↪cercare e passo il dizionario da aggiornare
        elif nome.endswith('.txt'):                       # altrimenti è un␣
 ↪file e se finisce con '.txt'
            size = os.path.getsize(fullname)              # ne trovo le␣
 ↪dimensioni
            dizionario[fullname] = size                   # ed aggiorno il␣
 ↪dizionario
    return dizionario
```

[90]:
```
cerca_file_sizes_corretto('.')
```

[90]:
```
{'./ita.lezione.17.txt': 72501, './eng.lezione.17.txt': 67840}
```

[91]:
```
cerca_file_sizes_corretto('../lezione11')    # ORA VA BENE
```

[91]:
```
{'../lezione11/paperopoli.txt': 10,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/alice.txt': 167517,
 '../lezione11/files/results.txt': 272,
```

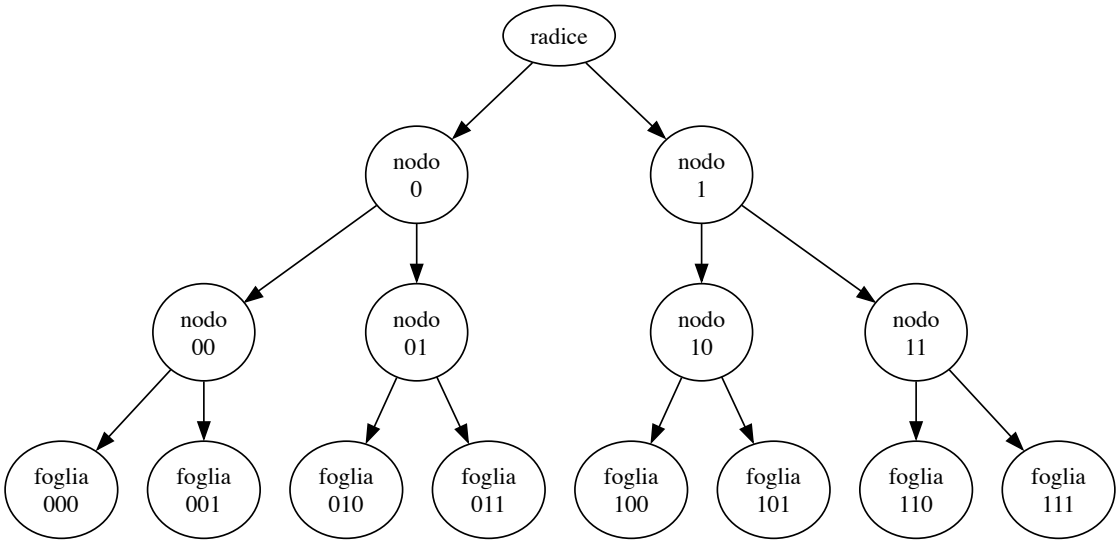
```
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/holmes.txt': 594933,
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/testo2.txt': 19,
 '../lezione11/files/testo.txt': 146}
```

5 Alberi binari

- si parte da un nodo "radice" che non ha "padri"
- ogni nodo piò avere fino a 2 "figli"
- i nodi senza figli sono le "foglie"
- ogni nodo ha un solo "padre"

[31]:
```
from pygraphviz import AGraph
G = AGraph(directed=True)
G.add_path(['radice', "nodo\n0", "nodo\n00", "foglia\n000"])
G.add_path(['radice', "nodo\n1", "nodo\n10", "foglia\n100"])
G.add_path([          "nodo\n0", "nodo\n01", "foglia\n010"])
G.add_path([          "nodo\n1", "nodo\n11", "foglia\n110"])
G.add_edge("nodo\n00", "foglia\n001")
G.add_edge("nodo\n10", "foglia\n101")
G.add_edge("nodo\n01", "foglia\n011")
G.add_edge("nodo\n11", "foglia\n111")
G.layout('dot')
G
```

[31]:



[92]:
```
# usiamo gli oggetti per rappresentare i nodi dell'albero
class NodoBinario:
```

```
    def __init__(self,   V : int, left  : 'NodoBinario' = None, right :␣
 ↪'NodoBinario' = None):
        self._value = V
        self._sx = left
        self._dx = right
    def __repr__(self):
        return f'NodoBinario({self._value})'

n11 = NodoBinario(11)
n10 = NodoBinario(10)
n1 = NodoBinario(1, right=n11)
n0 = NodoBinario(0, left= n10)
r = NodoBinario('radice', n0, n1)
r
```

[92]: NodoBinario(radice)

5.1 Stampa di un albero

5.1.1 con visita in PREordine (la radice prima dei sottoalberi)

[33]:
```
# radice è un nodo oppure None
# per stampare indentato passo un argomento 'livello'
# e lo incremento ogni volta che scendo in un sottoalbero
def stampa_PRE(radice : NodoBinario, livello : int = 0) -> None :
    print('|--'*livello, radice)             # PRIMA stampo la radice (lo␣
 ↪faccio in ogni caso, per mostrare i figli mancanti)
    if radice:
        stampa_PRE(radice._sx, livello+1)    # poi SE questo è un nodo stampo SX
        stampa_PRE(radice._dx, livello+1)    # e DX

stampa_PRE(r)
```

```
 NodoBinario(radice)
|-- NodoBinario(0)
|--|-- NodoBinario(10)
|--|--|-- None
|--|--|-- None
|--|-- None
|-- NodoBinario(1)
|--|-- None
|--|-- NodoBinario(11)
|--|--|-- None
|--|--|-- None
```

5.1.2 con visita in POSTordine (la radice DOPO i sottoalberi)

[34]:
```
def stampa_POST(radice, livello=0):
    if radice:                                  # se il nodo esiste
        stampa_POST(radice._sx, livello+1)      # scendo nei sottoalberi SX
        stampa_POST(radice._dx, livello+1)      # e DX
    print('|--'*livello, radice)                # e POI stampo il nodo (lo faccio␣
 ↪in ogni caso, per mostrare i figli mancanti)
    # altrimenti non ho nulla da fare
stampa_POST(r)
```

```
|--|--|-- None
|--|--|-- None
|--|-- NodoBinario(10)
|--|-- None
|-- NodoBinario(0)
|--|-- None
|--|--|-- None
|--|--|-- None
|--|-- NodoBinario(11)
|-- NodoBinario(1)
 NodoBinario(radice)
```

5.1.3 con visita INordine (la radice TRA i sottoalberi)

[35]:
```
def stampa_IN(radice, livello=0):
    if radice:                                  # se questo è un nodo
        stampa_IN(radice._sx, livello+1)        # stampo il sottoalbero SX
    print('|--'*livello, radice)                # poi la radice (lo faccio in ogni␣
 ↪caso, per mostrare i figli mancanti)
    if radice:
        stampa_IN(radice._dx, livello+1)        # poi il DX

stampa_IN(r)
```

```
|--|--|-- None
|--|-- NodoBinario(10)
|--|--|-- None
|-- NodoBinario(0)
|--|-- None
 NodoBinario(radice)
|--|-- None
|-- NodoBinario(1)
|--|--|-- None
|--|-- NodoBinario(11)
|--|--|-- None
```