Lezione 2 – I Dizionari

Informatica

21 Aprile 2015

Esempio: Invertiamo un dizionario

- Invertire un dizionario significa creare un dizionario che mappa i valori nelle chiavi.
- Poiché un valore può essere associato a più chiavi, il dizionario inverso ha delle *liste* come valori

```
>>> ist = istogramma("pappagallo")
>>> print(ist)
{'g': 1, 'o': 1, 'p': 3, 'l': 2, 'a': 3}
>>> inv = inverti_diz(ist)
>>> print(inv)
{1: ['g', 'o'], 2: ['l'], 3: ['p', 'a']}
```

```
def inverti_diz(d):
    inverso = dict()
    for chiave in d:
        val = d[chiave]
        if val not in inverso:
            inverso[val] = [chiave]
        else:
            inverso[val].append(chiave)
    return inverso
```

Un esempio di codice inefficiente

Prendiamo come esempio il codice per calcolare la serie di Fibonacci:

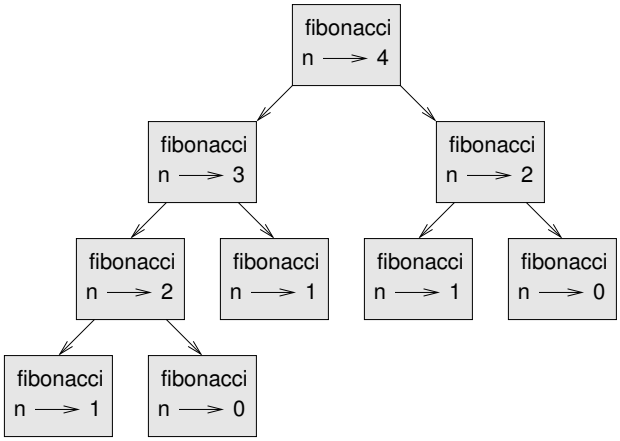
```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

Questo codice non è molto efficiente: il tempo impiegato per calcolare il risultato cresce molto in fretta

- provate `fibonacci(40)`

Perché `fibonacci` impiega così tanto tempo?

La spiegazione si trova nel *grafo delle chiamate* di `fibonacci`:



La funzione viene eseguita molte volte con lo stesso valore di input!

Usiamo un dizionario per velocizzare `fibonacci`

Per ottenere una soluzione più efficiente possiamo fare come segue:

- teniamo traccia dei valori già calcolati in un dizionario
- i valori memorizzati nel dizionario sono detti *memo*
- ad ogni chiamata, `fibonacci` controlla se l'argomento è presente nel dizionario
- se è già presente, ritorna immediatamente il risultato
- altrimenti, calcola il nuovo valore e aggiunge un memo al dizionario

```
known = {0:0, 1:1}

def fibonacci_memo(n):
    if n in known:
        return known[n]
    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res

print (" --- FIBONACCI MEMO --- ")
f = fibonacci_memo(4)
print(f)
f = fibonacci_memo(10)
print(f)
```

Memo: esercizi

Esercizio

Confrontare le prestazioni di `fibonacci` e di `fibonacci_memo`

Esercizio

Scrivere una versione con memo della *funzione di Ackermann*:

```
def ackermann(m, n):
    if m == 0:
        return n+1
    if n == 0:
        return ackermann(m-1, 1)
    return ackermann(m-1, ackermann(m, n-1))
```

Il tempo di esecuzione migliora in questo caso?

Variabili globali

- In `fibonacci_memo` il dizionario `known` viene creato al di fuori della funzione
  - appartiene al frame speciale chiamato `__main__`
  - le variabili in `__main__` sono dette *globali*
  - sono visibili da qualsiasi funzione
  - permangono tra una chiamata di funzione e la successiva
- le variabili dichiarate all'interno di funzioni sono *locali*
  - vengono distrutte quando la funzione termina

Immutabili globali ed assegnazioni

- Le variabili globali possono essere usate come *flag*:
  - variabili booleane che indicano se una certa condizione è vera
- Per poter riassegnare il valore ad una variabile globale all'interno di una funzione bisogna prima *dichiarare* la variabile globale:

```
def fun():
  global flag
  flag = True
```

- Senza la dichiarazione di `global`, *viene creata una nuova variabile locale con lo stesso nome*

Mutabili globali

- Le variabili mutabili (liste, dizionari) possono essere modificate anche senza dichiarazione:
  - aggiungere, togliere, modificare elementi
- Per essere riassegnate devono invece essere dichiarate `global`:

```
def globale()
  global known
  known = dict()
```

Esempi di uso (corretto e scorretto) di variabili globali:

```
verb = True

def flag():
    if verb:
        print("Flag attivo")

print(" --- verboso --- ")
flag()
verb = False
flag()
```

```
been_called = False

def flag2():
    been_called = True

print(" --- flag 2 --- ")
print(been_called)
flag2()
print(been_called)

def flag3():
    global been_called
    been_called = True

print(" --- flag 3 --- ")
been_called = False
print(been_called)
flag3()
print(been_called)
```

Dizionari e tuple

Le *tuple* possono essere utilizzate nei dizionari sia come *chiavi* che come *valori*:

- esempio: un elenco telefonico è un dizionario che associa tuple `(nome, cognome)` a tuple `(prefisso, numero_tel)`

Scriviamo una serie di funzioni per gestire un elenco telefonico:

1. una funzione per aggiungere un nuovo elemento
2. una funzione per stampare l'elenco telefonico
3. una funzione per cercare un numero telefonico
4. una funzione per cercare il nome associato ad un numero

```
def inserisci(e, cognome, nome, prefisso, tel):
    if (cognome, nome) in e:
        raise ValueError
    else:
        e[cognome, nome] = (prefisso, tel)

def cerca_tel(e, cognome, nome):
    if (cognome, nome) in e:
        return e[cognome, nome]
    else:
        raise ValueError

def cerca_num(e, prefisso, tel):
    for (cognome, nome) in e:
        if e[cognome,nome] == (prefisso, tel):
            return e[cognome, nome]
    raise ValueError

def stampa(e):
    for (cognome, nome) in e:
        (pref, tel) = e[cognome,nome]
        print(cognome, nome, pref, tel)
```

L’elenco telefonico: esempio di esecuzione

```
>>> elenco = dict()
>>> inserisci(elenco, "Rossi", "Mario", 555, 123456)
>>> inserisci(elenco, "Bianchi", "Luigi", 555, 666666)
>>> stampa(elenco)
Rossi Mario 555 123456
Bianchi Luigi 555 666666
>>> (pref,tel) = cerca_tel(elenco, "Rossi", "Mario")
>>> print(pref, tel)
555 123456
>>> (cognome, nome) = cerca_num(elenco, 555, 666666)
>>> print(cognome, nome)
555 666666
```