Efficienza di funzioni

1. Scrivere funzioni efficienti è difficile
2. Le soluzioni più dirette difficilmente sono le più efficienti
3. Per scrivere funzioni più efficienti, dobbiamo scrivere funzioni più complesse

In pratica, uno sviluppattore spesso deve aumentare la **complessità concettuale** del suo codice, per diminuire la **complessità computazionale** (di calcolo) del suo codice.

Esempio: calcolo della radice quadrata o di Fibonacci.

Efficienza?

Ma come possiamo misurare l'efficienza di un programma?

Riferimento: Capitolo 9 del testo di Python consigliato
"A simplistic introduction to algorithmic complexity"

Micro-benchmark

Quanto tempo ci mette questa funzione ad eseguire?

```
def G(i):
    def Helper(i, a):
        if i <= 1:
            return a
        return Helper(i-1, a*i)
    return Helper(i, 1)
```

All'interno di un notebook possiamo provare a fare un micro-benchmark con il commando:

```
%prun G(10)
%prun G(50)
```

Il ciclo **while**

Quanto tempo ci mette questa funzione ad eseguire?

```
def G(i):
    a = 1
    while i >= 1:
        a *= i
        i -= 1
    return a
```

All'interno di un notebook possiamo provare a fare un micro-benchmark con il commando:

```
%prun G(1000)
%prun G(5000)
```

Micro-benchmark

Quanto è significativo questo micro-benchmark?

Il risultato ottenuto dipende da:

1. La velocità del computer su cui viene eseguito
2. L'efficienza dell'implementazione di Python installato sul computer (Python è un programma scritto in C)
3. Il valore del dato di input

In pratica abbiamo bisogno di un modello per la nostra macchina da calcolo, il nostro computer.

Random Access Machine

Usiamo, per semplicità, un modello di calcolo basato sulla **random access machine**.

In questo modello i **passi** di calcolo sono eseguiti in modo sequenziale, una alla volta. Un passo è un'operazione che richiede un numero fissato di tempo, come ad esempio:

- l'assegnamento di un oggetto ad una variabile
- l'esecuzione di un confronto tra due oggetti
- l'esecuzione di un'operazione aritmetica
- l'accesso di un oggetto in memoria (*subscripting?*)

Invece del tempo, possiamo cercare di stimare il numero di passi elementari che un programma deve eseguire.

Random Access Machine

Anche contanto le operazioni di base, il tempo di esecuzione dipende dalla dimensione dell'input e dai sui valori. **Esempio**:

```
def Contains(As, x):
    for e in As:
        if e == x:
            return True
    return False
```

Quanto tempo ci mette ad eseguire? Quando?

Running time

In generale, ci sono tre casi da considerare:

1. Il “**best-case**” dei tempi di esecuzione, che è il tempo di esecuzione quando l’input è il più favorevole possibile. In altri termini, il “best-case” corrisponde al minor tempo possibile di esecuzione a parità della dimensione dell’input.
2. Il “**worst-case**” dei tempi di esecuzione, dato dal massimo tempo di esecuzione a parità della dimensione dell’input.
3. Il “**average-case**” che è il tempo medio di esecuzione a parità della dimensione dell’input.

Fattoriale rivisto

Si consideri la funzione seguente:

```
def Fact(n):
    a = 1
    while n >= 1:
        a *= n
        n -= 1
    return a
```

DOMANDA: Quanti passi sono eseguiti dalla funzione?

Sono importanti i fattori costanti?

Notazione asintotica

Qualsiasi algoritmo eseguito su input sufficientemente piccolo resulta essere molto veloce. Dobbiamo quindi considerare cosa succede quando i dati di input diventano molto grandi. Si consideri la funzione:

```
def F(x):
  a = 0
  for i in range(1000):
    a += 1
  for i in range(x):
    a += 1
  for i in range(x):
    for j in range(x):
      a += 1
      a += 1
  return a
```

DOMANDA: Quanti passi sono eseguiti dalla funzione?

Notazione Big-O

Nella pratica quella che viene fatto è di descrivere il comportamento asintotico di un algoritmo con le seguenti due regole:

1. Se il tempo di esecuzione è dato dalla somma di molteplici termini, considera solo quello con l'ordine maggiore e non considerare gli altri
2. Se il termine rimanente è un prodotto, non considerare le costanti.

Nell'esempio precedente passiamo da 1000+x+x^2 a semplicemente x^2.

Si usa la notazione Big-O per indicare un upper bound (stima per eccesso) alla crescita asintotica (**ordine di crescita**) della complessità di una funzione. Nel caso precedente, avremo che la funzione è O(x^2), ovvero che la funzione non cresce più velocemente di quanto cresce il quadrato di x^2. In altri termini, nel caso peggiore (**worst-case**) sono necessary x^2 passi di esecuzione.

Classi fondamentali di complessità

I casi più comuni di complessità per delle funzioni (algoritmi) sono le seguenti:

- O(1) indica un **tempo costante** di esecuzione
- O(log(n)) indica un **tempo logaritmo** di esecuzione (n è la dimensione più importante dei dati di input).
- O(n) indica un **tempo lineare**
- O(n log(n)) indica un **tempo lognormale**
- O(n^k) indica un **tempo polinomiale** (k è una costante)
- O(c^n) indica un **tempo esponenziale** (c è una costante)

Esempio: tempo logaritmico

Si consideri la funzione:

```
def Int2Str(n):
    digits = "0123456789"
    if n == 0:
        return '0'
    result = ""
    while n > 0:
        result = digists[n%10] + result
        n = n // 10
    return result
```

DOMANDA: qual'è il Big-O di questa funzione?

Esempio: tempo lineare

Siete in grado di darmi una funzione di complessità lineare? Quale?

Esercizio: quale complessità ?

Siete in grado di darmi una funzione di complessità lineare? Quale?

```
def isSubset(As, Bs):
    for a in As:
        if a not in Bs:
            return False
    return True
```

Complessità esponenziale

Si consider la funzione seguente:

```
def getBinary(n, numBits):
    result = ''
    while n > 0:
        result = str(n%2) + result
        n = n // 2
    for i in range(numBits - len(result)):
        result = '0'+result
    return result
```

Complessità esponenziale

Si consider la funzione seguente:

```
def getPowerSet(As):
  powerset = list()
  n = len(As)
  for i in range(0, 2**n):
    binStr = getBinary(i, n)
    subset = []
    for j in range(n):
      if binStr[j] == '1':
        subset.append(L[j])
    powerset.append(subset)
  return powerset
```

Confronti