Ingegneria degli algoritmi

Tutoraggio

Anno Accademico 2022/2023

Dott. Staccone Simone

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Contatti:

- simone.staccone@virgilio.it
- simone.staccone@students.uniroma2.eu
- Inserire oggetto [IA22-23]
- Microsoft Teams

  https://teams.microsoft.com/l/chat/0/0?users=simone.staccone@students.uniroma2.eu

Orario esercitazione:

Ogni mercoledì 16:00-17:30 aula A1

Complessità computazionale

- Per calcolare la complessità di un algoritmo dobbiamo prendere in considerazione due fattori principali:
  - Quanta memoria occupa
  - Quanto velocemente termina

- Nel corso ci occuperemo di affrontare problematiche legate alla complessità temporale di un algoritmo, ma un discorso analogo a quello che faremo vale anche per la complessità spaziale.

- Per cominciare abbiamo bisogno di un modello astratto che ci renda possibile studiare un algoritmo indipendentemente dal tipo di linguaggio utilizzato per la sua implementazione, dalla macchina fisica o virtuale nel quale viene eseguito, ecc..

Complessità computazionale - 2

- Come modello considereremo una Macchina di Touring di tipo RAM(Random Access Memory). Ovvero usiamo come astrazione un calcolatore che ha memoria infinita e scorribile in entrambi i sensi.

- Approfondimento: https://www.geeksforgeeks.org/turing-machine-in-toc

- Dobbiamo definire quindi un metodo qualitativo per misurare quanto veloce un algoritmo viene eseguito. Proprio per questo si è sviluppata la teoria dell'analisi della complessità.

Che linguaggio utilizziamo?

C

```
#include <stdio.h>

int main(int argc, char *argv){
	printf('Hello World');
	return 0;
}
```

C++

```
#include <iostream>

int main(int argc, char *argv){
	std::cout << 'Hello World';
	return 0;
}
```

Java

```
public class Main{
	public void main(String[] args){
		System.out.println('Hello World');
	}
}
```

Python

```
print('Hello World!')
```

Che linguaggio utilizziamo? - 2

- Dobbiamo utilizzare un linguaggio astratto che non tiene conto delle implementazioni pratiche e per cui esiste una corrispondenza univoca con qualsiasi altro tipo di linguaggio.

- Utilizzeremo quindi un linguaggio chiamato pseudocodice: utilizziamo il linguaggio comune per esprimere le operazioni di base di ogni linguaggio di programmazione (if, else, for, while, ecc…), considerando ogni operazione avente un costo unitario in lettura e in scrittura.

- Non esiste uno standard per lo pseudocodice, l'importante è che ogni operazione rappresenti un'operazione unitaria per poter effettuare delle stime quantificabili ma qualitative sul codice.

Che linguaggio utilizziamo? - 3

Compilatore pseudocodice online: https://pseudocode.deepjain.com

Pseudocodice (Hello World)

```
fun stampa
    output "Hello World!"
```

Pseudocodice (Esempio)

```
fun somma() : int
    read(a)
    read(b)
    sum = a + b
    return sum
```

Esercizio 1

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Pseudocodice (Esempio)

```
• fun somma() : int
    read(a)
    read(b)
    sum = a + b
    return sum
```

Esercizio 1 - 2

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Pseudocodice (Esempio)

- fun somma() : int

read(a)	1
read(b)	1
sum = a + b	1
return sum	1

Costo 4
Ma 4 cosa???
Questa è semplicemente un'analisi qualitativa!!!!

Esercizio 2

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Calcolo dei divisori di 100

```
• fun somma()
    i = 1
    loop while (i < 100)
        if 100 mod i == 0 then
            output i
        end if
        i = i + 1
    end loop
```

Esercizio 2 - 2

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Calcolo dei divisori di 100

- fun somma()

```
    i = 1                                   1
    loop while (i < 100)                    ???
        if 100 mod i == 0 then              ???
            output i                        1
        end if
        i = i + 1                           1
    end loop
```

Esercizio 2 - 2

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Calcolo dei divisori di 100

- fun somma()

```
    i = 1                                   1
    loop while (i < 100)
        if 100 mod i == 0 then
            output i                        1
        end if
        i = i + 1                           1
    end loop
```

Costo: 1 + 99*1 + ??? = 100 + ???

Esercizio 2 - 3

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Calcolo dei divisori di 100

- fun somma()

i = 1	1
loop while (i < 100)	1 (viene eseguito 99 volte)
if 100 mod i == 0 then	1 (viene eseguito 99 volte)
output i	1 (divisori di 100 sono 8)
end if	
i = i + 1	1 (viene eseguito 99 volte)
end loop	

Costo: 1 + 99*1 + 99*1 + 8 + 99*1= 1 + 99 + 99 + 8 + 99 = (costo i = 1) + (costo controllo while) + (costo controllo if) + (costo di output, non viene eseguito sempre ma solo 8 volte) + (costo i = i+1) = 306

Per conoscere il costo finale è necessario sapere il risultato dell'algoritmo, non uno dei modi migliori per calcolare la complessità di un algoritmo prima di lanciarlo.

Esercizio 3

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Minimo comune multiplo tra due numeri

```
fun gcd(a,b) : int
    loop while b != 0
        x = b
        b = a mod b
        a = x
        end loop
    return a

fun mcm(a,b)
    c=(a*b)/gcd(a,b)
    output c
```

Esercizio 3 - 2

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Minimo comune multiplo tra due numeri

```
fun gcd(a,b) : int
    loop while b != 0
        x = b                    1
        b = a mod b              1
        a = x                    1
        end loop
    return a

fun mcm(a,b)
    c=(a*b)/gcd(a,b)          1 + 3*?
    output c                     1
```

Costo: 1 + 1 + 3*?

Il costo dipende dal numero di cicli while che vengono eseguiti.

Sovrastimando sono sicuro che il termina dopo aver compiuto b passi.

Esercizio 3 - 3

Calcolare quante righe di codice vengono eseguite dal seguente algoritmo:

Minimo comune multiplo tra due numeri

```
fun gcd(a,b) : int                                  Costo: 1 + 1 + 3*b = 2 + 3*b
    loop while b != 0
        x = b                                   1   In generale il costo è O(b)
        b = a mod b                             1
        a = x                                   1   (Un'analisi più completa rivela
        end loop                                    una complessità di log(b))
    return a

fun mcm(a,b)
    c=(a*b)/gcd(a,b)                        1 + 3*?
    output c                                  1
```

Complessità computazionale - 3

- La misurazione della velocità di un algoritmo avviene quindi relativamente alla grandezza del suo input.
- Proprio per questo, l'analisi di un algoritmo va contestualizzata rispetto a quale tipo di input stiamo considerando.
- Alcuni algoritmi sono molto 'veloci' per input molto piccoli, ma disastrosi per input grandi. Altri possono essere più o meno 'veloci' con input diversi ma della stessa grandezza. (Vedi quicksort)
- In generale si tenta sempre di utilizzare il caso pessimo per valutare la complessità di un algoritmo.

Ricorrenze

Trovare la soluzione delle seguenti relazioni di ricorrenza, utilizzando il master theorem quando possibile, oppure il metodo di sostituzione o il metodo iterativo. (Assumere il costo per il caso $n \leq 1$ unitario)

(In classe)

$T(n) = 4T(n/2) + n$

$T(n) = 4T(n/2) + n^2$

$T(n) = 4T(n/2) + n^3$

$T(n) = T(n - 1) + n$

$T(n) = T(n - 1) + \log n$

$T(n) = 2T(n - 1) + 1$

$T(n) = 2T(\sqrt{n}) + \log n$

(Extra)

$T(n) = T(n/4) + T(3n/4) + n$

$T(n) = T(n - d) + T(d) + n^2$ $\quad d \geq 1$ costante

$T(n) = \sqrt{n}\, T(\sqrt{n}) + n \log \sqrt{n}$

$T(n) = \sqrt{n}\, T(\sqrt{n}) + O(n)$