

Strutture Dati e Algoritmi

La ricorsione

Luca Di Gaspero, Università degli Studi di Udine

Informazioni generali



Luca Di Gaspero

Dipartimento Politecnico di Ingegneria e
Architettura


🕒 Su appuntamento: bit.ly/bookldg

✉ luca.digaspero@uniud.it

☎ 0432 55 8242

In questo modulo

Obiettivi

- fornire degli approfondimenti riguardo alle metodologie di progetto e sviluppo di programmi
- concetto di algoritmo e di complessità computazionale
- metodologie per lo sviluppo di algoritmi
- metodologie per l'analisi degli algoritmi
- selezione di strutture dati e algoritmi
-  implementazioni in linguaggio C



P. Crescenzi, G. Gambosi, R. Grossi, G. Rossi. Strutture di dati e Algoritmi, 2a Edizione. Pearson Italia. 2012.

- Lezioni registrate
- Lucidi delle lezioni
- Codice ed esempi
- VPL

Prova Finale

Integrata con Fondamenti di Programmazione (unica prova)

Scritta e orale

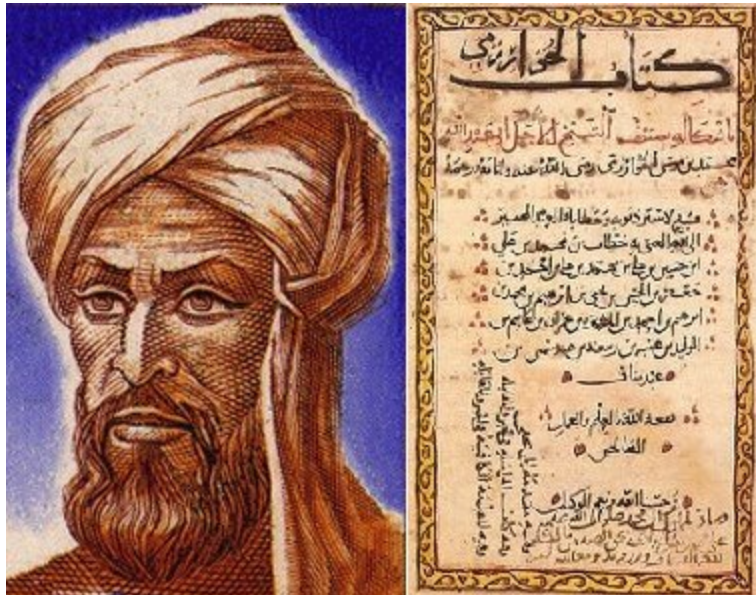
- scritto con esercizi relativi allo sviluppo di algoritmi e alla loro **implementazione in linguaggio C**
- orale di teoria / pratica / analisi degli algoritmi

Parziali che verranno resi noti in futuro

Algoritmi e computabilità

Algoritmo

- Essenza computazionale di un programma che ne descrive i passi fondamentali
- **Definizione:** un *algoritmo* è un procedimento che consente di ottenere un risultato eseguendo, in un determinato ordine, un insieme di passi elementari corrispondenti ad azioni scelte solitamente da un insieme finito
- il termine deriva dal nome del matematico persiano **Abū Ja'far Muhammad ibn Mūsā al-Khwārizmī** (ابو جعفر محمد خوارزمی, 780–850 d.C.)



Caratteristiche Fondamentali

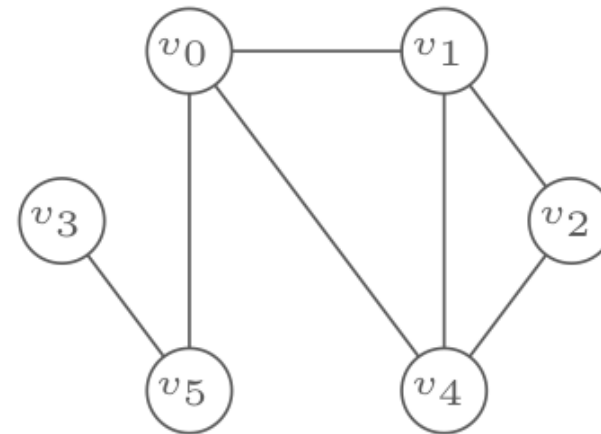
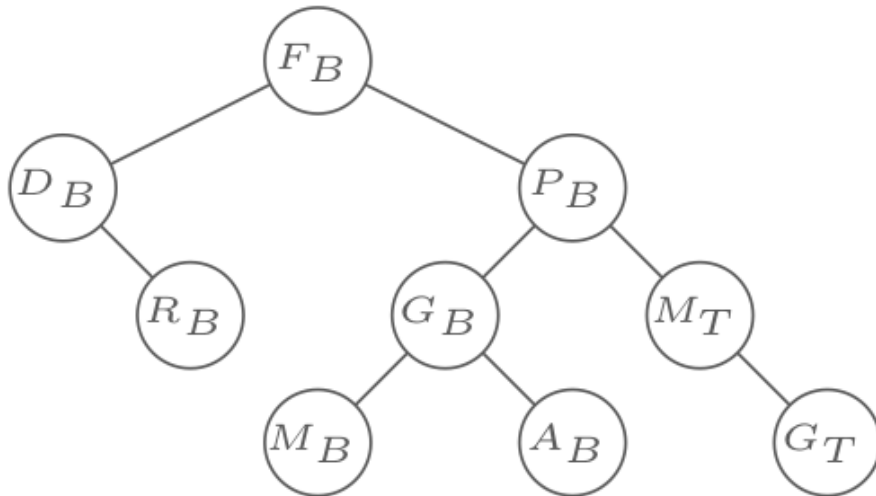
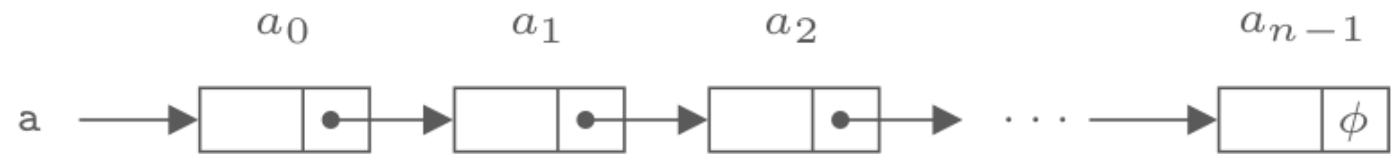
- la sequenza di istruzioni deve essere finita (finitezza della descrizione, finitezza del procedimento, finitezza dei singoli passi)
- le istruzioni devono essere eseguibili materialmente (realizzabilità)
- le istruzioni devono essere espresse in modo non ambiguo (non ambiguità)
- il procedimento deve portare ad un risultato (effettività)

Altre proprietà desiderabili

- un algoritmo dovrebbe funzionare correttamente anche variando alcuni aspetti del problema (generalità)
- il numero di azioni eseguite per la soluzione del problema dovrebbe essere minimo (efficienza)
- la sequenza di azioni che verranno eseguite per ogni insieme di dati dovrebbe essere prevedibile (determinismo)

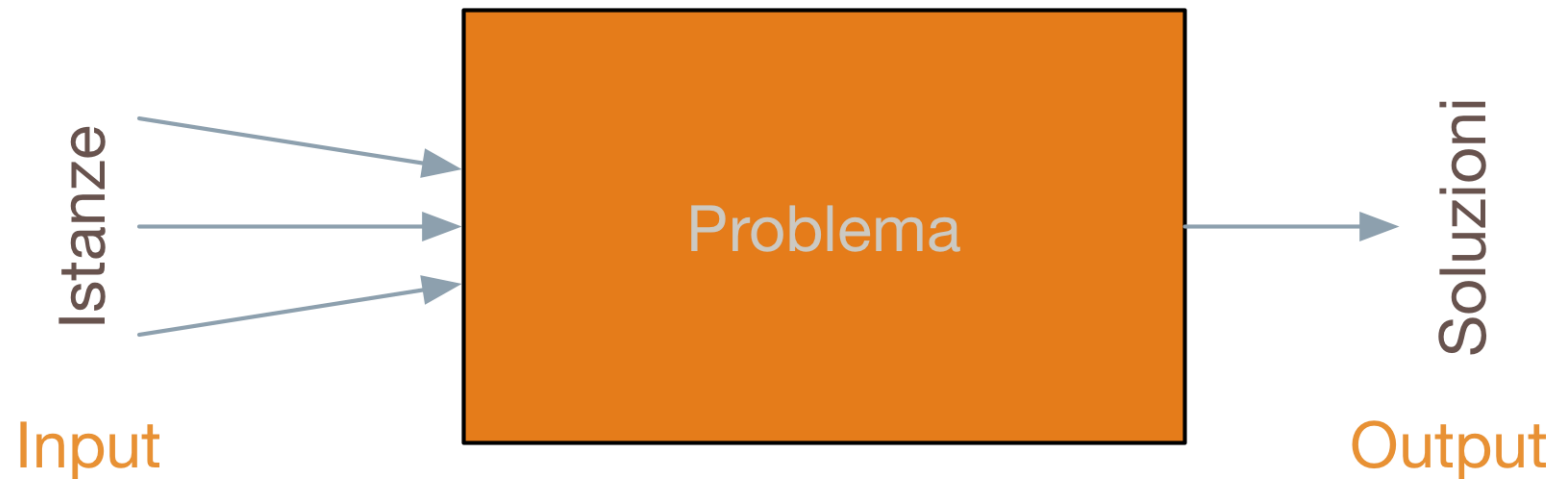
Strutture dati

- Uno degli “ingredienti” degli algoritmi: *sequenze, alberi e grafi*
- Servono a rappresentare i dati di problemi complessi strutturando i dati elementari (**bit, caratteri, interi, reali e stringhe**)
- Rappresentano istanze di problemi computazionali reali e concreti



Algoritmi e Problemi

- Un algoritmo risolve un *problema*
 - dal punto di vista matematico è una *corrispondenza univoca* fra *spazio delle istanze* e *spazio delle soluzioni*: $y = f(x) : x \in \Pi$
 - l'algoritmo descrive il procedimento che, una volta affidato ad un "esecutore", consente di "calcolare" la corrispondenza f



Algoritmi e Problemi, un esempio

- Calcolare il massimo comune divisore fra due numeri interi
 - *Istanze*: coppie di numeri interi $(a, b) \in \mathbb{Z} \times \mathbb{Z}$
 - *Soluzioni*: g , tale che, $a \bmod g = 0, b \bmod g = 0, \forall g' > g \quad a \bmod g' \neq 0 \vee b \bmod g' \neq 0$

Problemi decidibili e indecidibili

Classificazione dei problemi

Non tutti i problemi computazionali ammettono algoritmi di risoluzione: ad esempio il *problema della fermata (halting problem)* (Turing, 1937)

Dato un generico algoritmo (o programma) A in ingresso, esso termina o va in ciclo infinito?

Non c'è una soluzione al problema nel caso generale (ossia una dimostrazione che consenta di rispondere per un *qualunque algoritmo*)

In alcuni casi specifici tuttavia è possibile dare delle risposte

Problemi decidibili

- Un problema è detto **decidibile** (o **calcolabile**) se esiste un algoritmo che **per ogni sua istanza** sia in grado di **terminare** la sua esecuzione pervenendo ad una soluzione
- Ad esempio, *determinare se un numero è primo*

```
primo(n) {  
    fattore <- 2;  
    if (n < 1)  
        return false; /* l'istruzione return termina l'algoritmo e restituisce un valore */  
    while (n % fattore != 0)  
        fattore <- fattore + 1;  
    if (n == fattore)  
        return true;  
    else  
        return false;  
}
```


Problemi decidibili

L'algoritmo `primo()` termina su ogni sua istanza (ovvero per qualunque valore di `n`)?

- **Si**, sempre: perché se `n` è minore di 1 si risponde immediatamente, altrimenti la variabile `fattore` viene sempre incrementata di 1 e a un certo punto deve verificare la condizione (almeno quando `fattore` è esattamente uguale a `n`)

Problemi decidibili, algoritmi corretti

L'algoritmo `primo()` è sempre corretto (ovvero restituisce il valore corretto per qualunque valore di `n`)?

- **Si**, o meglio l'ipotesi per cui è corretto è che i numeri negativi non siano considerati come primi, cosa che è coerente con la definizione:

I numeri primi sono tutti e soli i *numeri naturali* che sono divisibili solamente per 1 e per se stessi.

Problemi indecidibili

- Un problema è **indecidibile** (o **non calcolabile**) se nessun algoritmo sia in grado di terminare la sua esecuzione pervenendo ad una soluzione su **ogni sua istanza**

Dato un programma **A**, non esiste un algoritmo per stabilire se esso terminerà o meno la sua esecuzione su un qualunque input

Problema della fermata (Turing, 1937)

Indecidibilità

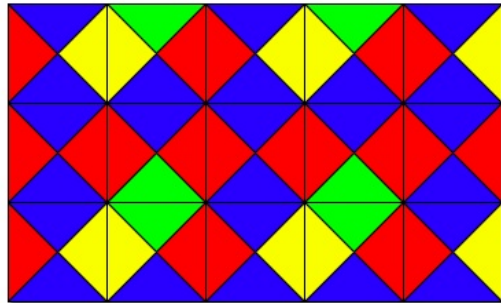
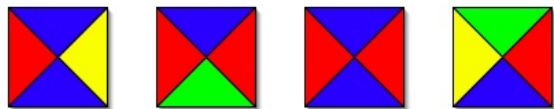
Il problema della fermata è **indecidibile**, ossia non esiste alcun algoritmo di risoluzione che termina su ogni suo input restituendo un risultato

- In realtà il problema è **semidecidibile**, ovvero è facile decidere se termina (quando termina): basta eseguirlo; è impossibile decidere se non termina

Altri problemi indecidibili

- stabilire l'*equivalenza tra due programmi* f, g : per ogni possibile input, producono lo stesso output?
- *tiling problem*: data una scacchiera $n \times m$ e delle tessere di dimensione unitaria, ognuna con quattro triangoli colorati in un certo modo, è possibile o meno coprire la scacchiera rispettando il vincolo che piastrelle adiacenti siano dello stesso colore?

Tiling problem



Ovviamente, in casi particolari, come quello riportato a lato (con $n = 3$ e $m = 5$), è possibile risolverlo ma non è possibile risolverlo nel caso generale con n e m arbitrari