

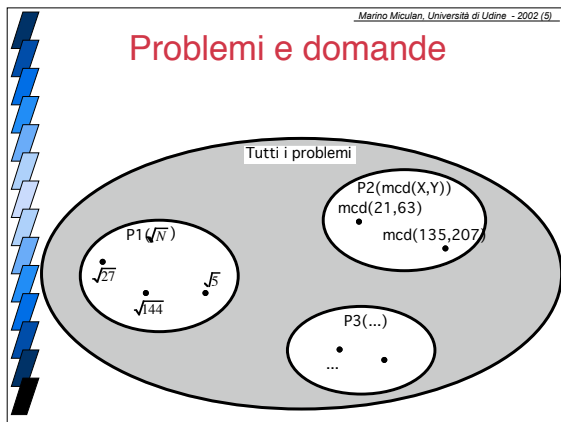
Ufficio e Direttore

- torniamo alla metafora iniziale...
 - l'impiegato conosce poche e semplici istruzioni
 - l'ufficio effettua operazioni più complesse
 - grazie alle istruzioni del direttore
 - per rispondere ad una domanda dell'utente
- il direttore come si comporta?
 - ascolta una domanda posta dall'utente, che gli pone una questione ben precisa
 - non dà una risposta alla domanda dell'utente, ma progetta un metodo per dare risposte anche a domande analoghe
 - infatti, quando l'utente chiede il valore di 3^4 , il direttore progetta la risposta generica per il problema M^N

Alcune definizioni

- **Problema:**
 - insieme di domande omogenee, ovvero domande cui è possibile rispondere utilizzando una procedura uniforme
 - Esempi:
 - "Quanto vale la radice quadrata intera del numero naturale X?"
 - "Calcolare il massimo comun divisore di due numeri naturali X e Y"
- **Istanza di problema:**
 - specifico esemplare di domanda facente parte dell'insieme che costituisce il problema
 - Esempi:
 - "Quanto vale la radice quadrata intera del numero naturale 27?"
 - "Calcolare il massimo comun divisore dei due numeri 21 e 63"
 - la chiamiamo **domanda**

Problemi e domande



Le variabili di ingresso

- affinché possa identificare un insieme di domande, la descrizione di un problema avrà necessariamente dei punti non specificati, la cui variabilità permette di ottenere tutte le domande dell'insieme
- questi punti non specificati si chiamano **variabili d'ingresso**: sono i termini variabili che caratterizzano la definizione del problema e, assumendo dei valori all'interno di opportuni domini, permettono di generare tutte le istanze
- i valori che le variabili assumono in corrispondenza di una specifica istanza si dicono **dati**
- Esempio: la X nella domanda "Quanto vale la radice quadrata intera del numero naturale X?"

Le variabili d'uscita

- è meno evidente, ma anche la risposta ad una domanda è un termine variabile, che assume come valori le possibili soluzioni alle istanze del problema
- Esempi:
 - "Quanto vale la radice quadrata intera Y del numero naturale X?"
 - "Calcolare il massimo comun divisore Z di due numeri naturali X e Y"
- chiamiamo **variabili d'uscita** i termini variabili che caratterizzano le soluzioni attese di un problema
- I valori assunti dalle variabili d'uscita si dicono **risultati**
- si dice **soluzione di un'istanza** di problema la risposta ad una domanda

Notazione

- $Z = P[X, Y]$: problema P a variabili d'ingresso X, Y e variabile d'uscita Z
- $P[x, y]$: istanza del problema $P[X, Y]$ quando le variabili d'ingresso sono i dati x e y
- $z = P[x, y]$: z è la soluzione dell'istanza $P[x, y]$
- $P[x, Y]$: sottoproblema di $P[X, Y]$ quando alla variabile X si sostituisce il dato x
- bisogna specificare anche i domini delle variabili
- Esempio:
 - $Z = P[X, Y]$: "Calcolare il massimo comun divisore Z di due numeri naturali X e Y"
 - domini: X e Y sono numeri naturali
 - $P[21, 63]$: "Calcolare il massimo comun divisore di 21 e 63"
 - $P[21, Y]$: "Calcolare il massimo comun divisore di 21 e Y"

Utilità delle variabili

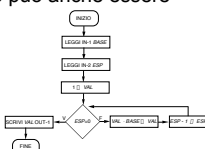
- semplificano la descrizione dei problemi
 - esempio:
"Dati 6 numeri interi N1, N2, N3, N4, N5 e N6, calcolare l'espressione $(4 * N1 + (N2^{N3})) / (N4 + N5 * N6 + N1)$ "
- esplicitano il legame tra problema e istanza
 - in quanto l'istanza deve specificare il valore delle variabili
 - se non tutte le variabili vengono specificate, non otteniamo un'istanza ma un **sottoproblema**, più semplice del problema originario
 - Esempio: "Calcolare il massimo comun divisore Z dei due numeri naturali 21 e Y"

Algoritmo risolvete

- è un metodo generale che permette di fornire in modo uniforme la risposta a tutte le istanze di un problema
- a volte viene chiamato, con imprecisione, "soluzione di un problema"
- Esempio:
 - "Quanto vale la radice quadrata intera Y di X?"
 - (Un) algoritmo risolvete:
 - 1: assegna a Y il valore 0
 - 2: incrementa Y di 1
 - 3: se $Y^2 < X$, allora torna all'istruzione 2
 - 4: se $Y^2 > X$, allora decrementa Y di 1
 - (con X e Y numeri naturali)
- per un problema ci sono più algoritmi risolvete

Algoritmo

- è un metodo, intangibile, di cui è possibile dare una rappresentazione tangibile, e di cui è possibile percepire l'esecuzione
- essendo un concetto astratto, affinché possa essere usato per ottenere le soluzioni di un problema, deve essere **rappresentato** in qualche modo
- Per esempio: il diagramma di flusso visto all'inizio del corso è una rappresentazione di un algoritmo per l'elevamento a potenza, che però può anche essere descritto così:
 - leggi i 2 numeri;
 - se il secondo numero è 0, stampa 1;
 - altrimenti moltiplica il primo numero per se stesso tante volte quanto è indicato dal secondo numero e stampa il risultato ottenuto

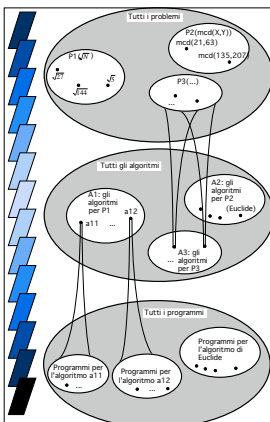


Rappresentazione degli algoritmi

- un algoritmo deve essere descritto in un opportuno linguaggio di descrizione che sia comprensibile ed "eseguibile" da parte dell'entità preposta alla risoluzione dei problemi (il "calcolatore" secondo la definizione informale data all'inizio del corso)
- chiamiamo **linguaggio di programmazione** questo linguaggio di descrizione
- chiamiamo **programma** la descrizione dell'algoritmo nel linguaggio di programmazione
- esistono più linguaggi di programmazione, da cui gli algoritmi sono *indipendenti*
- esistono più descrizioni dello stesso algoritmo nello stesso linguaggio

Esempio: algoritmo in linguaggio umano

- Per calcolare la potenza di due numeri fai così:
 - Leggi i due numeri
 - Se il secondo numero è zero, stampa 1
 - Altrimenti moltiplica il primo numero per se stesso tante volte quanto indicato dal secondo numero, e stampa il risultato
- Questa rappresentazione è molto diversa da quella dei diagrammi di flusso
 - Più astratta, vicina al nostro modo di parlare
 - Meno precisa, più ambigua
 - Non è espressa in istruzioni elementari eseguibili dall'impiegato



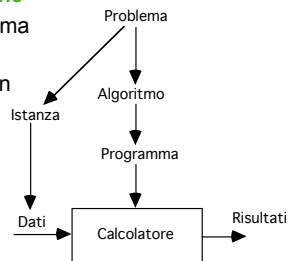
Rapporto tra
problemi,
domande,
algoritmi,
programmi

Calcolatori

- rivisitiamo la definizione data in passato...
- Un calcolatore è un sistema che:
 - riceve un programma $P[X,Y,...]$ e dei dati iniziali $x,y,...$, ove:
 - il programma $P[X,Y,...]$ è la rappresentazione di un algoritmo risolvibile $A[X,Y,...]$
 - in un linguaggio di programmazione comprensibile ed eseguibile dal calcolatore
 - i dati iniziali $x,y,...$ sono i dati che identificano una specifica istanza del problema
 - esegue il programma sui dati
 - fornisce la risposta all'istanza del problema, derivata istanziando il problema sui dati ($z = P[x,y,...]$)
- un calcolatore è quindi un **esecutore universale di algoritmi**, purché non abbia limitazioni di memoria

Attività del calcolatore

- l'esecuzione di un programma su un determinato calcolatore è detta **computazione**
- algoritmo e programma sono concetti *statici*
- la computazione è un concetto *dinamico*



Proprietà formali degli algoritmi

- poiché un algoritmo è un metodo per risolvere problemi, è necessario che sia effettivamente utilizzabile nella pratica da un esecutore
- questa utilizzabilità effettiva si traduce in alcune proprietà che l'algoritmo è bene possessa
- Un algoritmo deve essere:
 - **Finito**: un algoritmo deve essere composto da un numero finito di istruzioni
 - **Univoco**: ogni istruzione deve essere interpretabile in maniera non ambigua. Anche il modo di combinare assieme le istruzioni deve essere non ambiguo
 - **Effettivo**: deve esistere un calcolatore in grado di eseguire effettivamente ogni istruzione dell'algoritmo in un tempo finito

Finitezza

- se un algoritmo fosse composto da infinite istruzioni, non esisterebbe calcolatore in grado di eseguirlo:
- Perché?
 - un calcolatore reale esegue, in un tempo finito, un numero finito di istruzioni
 - un algoritmo composto da infinite istruzioni sarebbe rappresentato da un programma di lunghezza infinita (per esempio, un diagramma a blocchi infinito)
- Esempio di ~algoritmo "infinito":
 - calcolo del successore di un numero N:
 - 1. Se $N=0$ allora il risultato è 1
 - 2. Se $N=1$ allora il risultato è 2
 -

Univocità

- le istruzioni devono essere riconosciute in modo univoco dall'esecutore, affinché possano essere eseguite
- se una istruzione è ambigua, oppure più istruzioni sono combinate assieme ambiguamente, l'esecutore in realtà non ha indicazioni su come comportarsi
- Esempio:
 - blocco condizionale contenente la condizione "X è grande abbastanza?"
 - combinazione ambigua: "Ripeti le seguenti istruzioni per un po' di volte"

Effettività

- deve esistere un calcolatore in grado di eseguire effettivamente ogni istruzione dell'algoritmo in un tempo finito; questo garantisce che ogni algoritmo finito sia effettivamente eseguito
- Quattro sottoproprietà:
 - Deve esistere un limite finito ai tipi di istruzioni eseguibili dall'esecutore
 - Deve esistere un limite finito alla complessità delle istruzioni eseguibili
 - Deve essere disponibile all'esecutore una memoria illimitata, cioè finita ad ogni istante ma per la quale non esiste limite finito
 - L'esecutore deve operare per passi discreti

Cos'è un algoritmo?

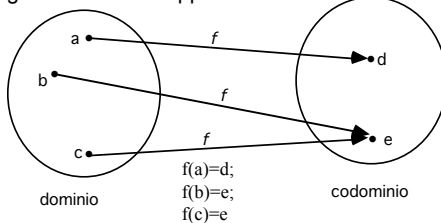
- di fatto, non l'abbiamo definito formalmente
- perché è un concetto astratto difficilmente catturabile,
- ed è più facile definire il programma corrispondente
- non si può dire che sia l'insieme dei programmi che risolvono lo stesso problema,
- si potrebbe dire -sempre informalmente- che è l'insieme dei programmi che risolvono lo stesso problema nello stesso modo...

Cosa si può fare con gli algoritmi?

- è una domanda legittima... si può fare tutto?
- altre domande:
 - se non si può fare tutto, cosa si può fare?
 - avendo più linguaggi di programmazione, esistono algoritmi rappresentabili in un linguaggio e non in un altro?
- scopo della **teoria della computabilità** (o calcolabilità) è rispondere a domande di questo tipo
- utilizzando il concetto di funzione calcolata da un algoritmo

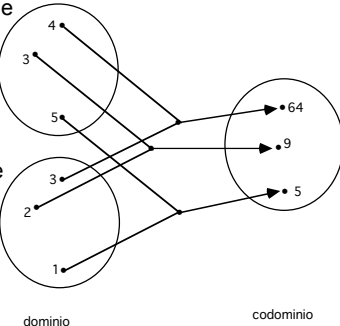
Funzioni

- una funzione f è una legge che associa ad un elemento di un insieme D detto **dominio** un elemento di un altro insieme C detto **codominio**
- notazione: $f: D \rightarrow C$
- graficamente è rappresentabile così:



Dominio e codominio

- possono assumere forme anche molto complesse
- per esempio, l'elevamento a potenza M^N è funzione di due numeri
- la notazione corrispondente sarà
 $f: D_1 \times D_2 \rightarrow C$



dominio

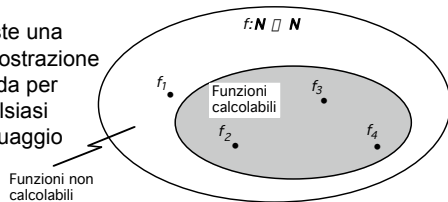
codominio

Funzione computata

- oltre che come risolutore di problemi, un algoritmo può essere visto come calcolatore di una funzione
- per esempio, l'algoritmo che risolve il problema dell'elevamento a potenza è anche l'algoritmo che calcola la funzione corrispondente (quella che associa a due numeri M, N il valore M^N)
- in teoria della computabilità, si considera un sottoinsieme delle funzioni possibili, poiché ci interessano domini e codomini particolari:
 - solo numeri naturali: il calcolatore memorizza solo numeri naturali (li usa anche per gli altri dati)
 - solo funzioni $f: \mathbf{N} \rightarrow \mathbf{N}$: domini multipli possono essere codificati (con apposita funzione...) in un unico dominio
 - anche funzioni non definite per qualche valore del dominio

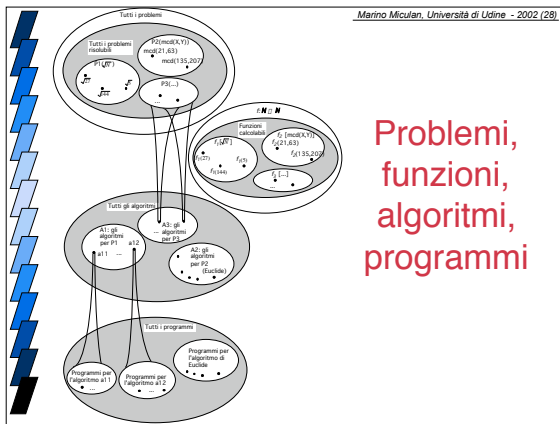
Funzioni computabili

- "con gli algoritmi si può fare tutto?" diventa:
- data una qualsiasi funzione, è possibile costruire un algoritmo che la computa?
- oppure: tutte le funzioni $f: \mathbf{N} \rightarrow \mathbf{N}$ sono computabili?
- **NO.**
- esiste una dimostrazione valida per qualsiasi linguaggio

Funzioni non
calcolabili

Esempi

- funzione non computabile:
decimo problema di Hilbert (stabilire se una generica equazione polinomiale a coefficienti interi ha o no una soluzione intera)
- funzione la cui non computabilità intuitiva non è ancora stata dimostrata:
 $f(n) = 1$ se esiste una sequenza di esattamente n "5" nell'espansione decimale di π ; 0 altrimenti
- funzione non computabile importante: problema della terminazione



Tesi di Church-Turing

- ora sappiamo che non possiamo calcolare tutto, e quindi che gli algoritmi non fanno tutto; cosa possono fare, allora?
- **Tesi di Church-Turing:**
Le funzioni computabili sono quelle *intuitivamente computabili*, cioè quelle che un essere umano è in grado di calcolare
- non è un teorema, ma è universalmente accettata
- come si applica: se di un problema riesco a dare una descrizione ragionevolmente algoritmica, allora la sua funzione è calcolabile ed il problema è solubile (...).



Equivalenza dei formalismi

- la nozione di computabilità non dipende dal formalismo
- buona parte dei formalismi finora studiati sono equivalenti (in particolare, se non lo sono sono meno espressivi):
- macchine di Turing, funzioni parziali ricorsive, sistemi di Post, lambda calcolo, ... calcolano tutti lo stesso insieme di funzioni, che è poi lo stesso calcolato dal nostro ufficio metaforico (a meno dei limiti di memoria)
- quindi in linea di massima possiamo utilizzare qualsiasi formalismo per computare le funzioni
