Informatica Grafica

Corso di Laurea in Ingegneria Edile – Architettura

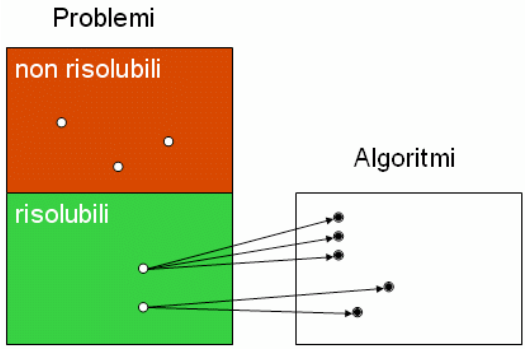
Fondamenti dell'Informatica

Paolo Torroni

Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
Università degli Studi di Bologna

Anno Accademico 2009/2010

Fondamenti dell'Informatica



- **Fondamenti dell'Informatica**
  - Procedimenti risolutivi (algoritmi)
  - Che linguaggi parlano i computer? Livelli di astrazione. Compilatori e interpreti.
  - Esistono problemi che nessun calcolatore è in grado di risolvere?
  - Esistono problemi risolubili in teoria ma non in pratica?

Parte I

Introduzione

Algoritmo per gli spaghetti all'arrabbiata

- **Ingredienti**: (per 6 persone)
  - spaghetti g 500
  - polpa di pomodoro fresco g 400
  - cipolla mondata g 30
  - olive nere
  - capperi
  - peperoncino piccante
  - basilico
  - olio d'oliva
  - sale

- Lessate gli spaghetti in abbondante acqua salata. Intanto, in una larga casseruola, fate riscaldare 3 cucchiaiate d'olio, insaporendolo con la cipolla tritata, un peperoncino intero, una decina di olive e una cucchiaiata di capperi. Unite quindi la polpa di pomodoro e fate cuocere il tutto per circa 5'. Scolate la pasta al dente e fatela saltare nel sugo, poi trasferitela in una terrina e servitela guarnita con basilico fresco.

Problemi, dati e risultati

- Tre elementi:
  - Scopo della ricetta: **problema** da risolvere
  - Ingredienti: **dati iniziali**
  - come trasformare gli ingredienti nel piatto finale: **istruzioni**
- Generalità:
  - generico cuoco, ingredienti
  - può essere ripetuta per ottenere lo stesso risultato
  - solo relazioni generali (es, 8 o 10 persone)

Proprietà degli algoritmi

- Generalità
- Comprensibilità da parte dell'esecutore
  - Uso di **operazioni elementari** alla portata dell'esecutore
- Finitezza
  - Numero limitato di istruzioni $\not\Rightarrow$ terminazione!
- Non ambiguità
  - Intanto fate riscaldare ...
  - Scolare la pasta al dente
    (Forse gli spaghetti all'arrabbiata non sono un buon esempio)
- Determinismo
  - Servire a piacere, con o senza parmigiano
- Efficienza (complessità)
- Semplicità

Algoritmo

Definizione (Algoritmo)

Successione non ambigua e ripetibile *di* ***istruzioni eseguibili*** *che permette di risolvere in modo* ***generale*** *un* ***problema****.*

- Soluzione costruita a partire da alcuni **dati iniziali**
- Istruzioni fanno riferimento a un'**insieme di operazioni elementari**, **eseguibili** da un certo **esecutore**.
- **Esecuzione** dell'algoritmo: seguire passo dopo passo le istruzioni

Un semplice problema: A+B

- Provare a scrivere **in linguaggio naturale** l'algoritmo per la somma di due numeri interi
  - Problema
  - Dati
  - Istruzioni
    1.
    2.
    3.

    ...
  - Quali sono le **operazioni elementari**?

Flusso del controllo

- Esecuzione "passo passo"
  1. Fate riscaldare 3 cucchiai d'olio ...
  2. Unite la polpa di pomodoro
  3. Fate cuocere il tutto per 5 min
- Istruzione condizionale
  - **Se** la pasta è cotta **allora** scolare la pasta
    altrimenti aspettate 1 altro minuto
- Istruzione iterativa
  - **Continuate a** cuocere **finché** il sugo non ha raggiunto la consistenza desiderata

Algoritmi, programmi, calcolatori

- **Calcolatore**: esecutore di algoritmi
- **Linguaggio di programmazione**: linguaggio per esprimere algoritmi in modo comprensibile al calcolatore
- **Programma**: algoritmo scritto in un linguaggio di programmazione
- A ciascun calcolatore è associato un **linguaggio macchina**, cioè un linguaggio di programmazione che può comprendere
  - Calcolatore $C \Rightarrow$ Linguaggio macchina L_C
  - Programma P scritto in $L_C \Rightarrow P$ eseguibile da C
  - Dati P e opportuni dati di ingresso a $C \Rightarrow$ soluzione

Calcolatore Rudimentale (CR)

Composizione. CR consiste di due parti:

- **controllo**: esecuzione del programma;
- **memoria**: un numero **illimitato** di celle:
  - capaci di memorizzare un numero naturale;
  - indicate con un numero: C_1, C_2, C_3, ...

Operazioni elementari. 5 operazioni:

- **somma** 1 alla cella C_n;
- **memorizza** 0 nella cella C_n;
- **leggi** dall'esterno un numero, memorizzalo in C_n;
- **mostra** il contenuto della cella C_n;
- **stop**.

Flusso del controllo. Salto condizionato

- se C_n e C_m contengono lo stesso numero, vai all'istruzione (X).
  ...salto *incondizionato*?

Un esercizio di analisi

- Cosa fa il seguente programma?
  1. **Leggi** un numero e memorizzalo in C_1.
  2. **Memorizza** zero nella cella C_2.
  3. **Se** C_1 e C_2 contengono lo stesso numero, vai all'istruzione (5).
  4. **Somma** uno alla cella C_1.
  5. **Mostra** la cella C_1.
  6. **Stop**.

Un altro esercizio di analisi

- Cosa fa il seguente programma?
  1. **Leggi** un numero e memorizzalo in C_1.
  2. **Memorizza** zero nella cella C_2.
  3. **Se** C_1 e C_2 contengono lo stesso numero, vai all'istruzione (3).
  4. **Somma** uno alla cella C_1.
  5. **Mostra** la cella C_1.
  6. **Stop**.

Problema della terminazione

- La possibilità di avere programmi CR che non terminano è intrinsecamente legata alla definizione di CR.
- Costituisce una caratteristica fondamentale di ogni linguaggio di programmazione.

Un terzo programma da analizzare

- Cosa fa il seguente programma?
  1. **Leggi** un numero e memorizzalo in C_1.
  2. **Leggi** un numero e memorizzalo in C_2.
  3. **Memorizza** zero in C_3.
  4. **Se** C_2 e C_3 contengono lo stesso numero, vai all'istruzione (8).
  5. **Somma** uno alla cella C_1.
  6. **Somma** uno alla cella C_3.
  7. **Se** C_2 e C_2 contengono lo stesso numero, vai all'istruzione (4).
  8. **Mostra** la cella C_1.
  9. **Stop**.

Problemi CR-risolubili

- La somma è un problema CR-risolubile

Definizione (Problema CR-risolubile)

Un problema è CR-risolubile se esiste un programma CR che lo risolve.

- I problemi dell'aritmetica elementare ($A + B$, $A - B$, $A \times B$, $\frac{A}{B}$, A^B, $MCD(A, B)$, ...) sono tutti CR-risolubili.
- Esattamente quali problemi sono CR-risolubili?

Parte II

Linguaggi di programmazione

Macchine e linguaggi più evoluti di CR

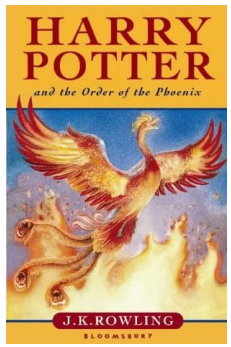
- Proviamo a esprimere algoritmi complessi nel linguaggio macchina di CR
  - migliaia di istruzioni
  - nessuna struttura che la colleghi alla struttura del problema originale
- Servono linguaggi per scrivere programmi concisi e strutturati (**linguaggi di alto livello**)
  - non ambigui
  - eseguibili
  - facilmente utilizzabili da un programmatore

Linguaggi di alto livello

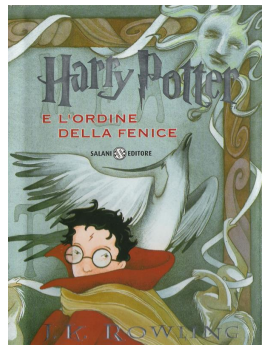
- Come rendere un linguaggio più semplice da utilizzare?
  - nomi simbolici per indicare le celle $\Rightarrow$ **variabili**
  - non solo numeri naturali $\Rightarrow$ **tipi di dato**
  - non solo somma $\Rightarrow$ **operatori**
  - non solo azzeramento $\Rightarrow$ **assegnamento**
  - non solo "Se $C_1 = C_2$ vai a (X)" $\Rightarrow$
    - **espressioni condizionali** (if, then, else) e
    - **iterazioni** (while, do) che usano
    - **espressioni logiche** $(=, <, \leq, \vee, \wedge, \ldots)$
  - possibilità di scomposizione in sottoproblemi $\Rightarrow$ **procedure**

Compilatori e interpreti

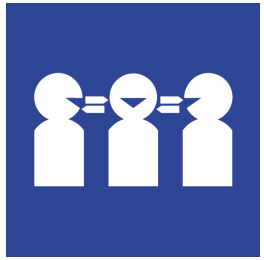
- Com'è possibile far eseguire a CR un programma scritto in un linguaggio di alto livello?
- **Traduzione**
  - **Compilazione**
    - In ingresso: il programma di alto livello P_L
    - In uscita: il programma per CR P_{CR}



$\Rightarrow$



  - **Interpretazione**
    - In ingresso: il programma di alto livello P_L e i dati per P_L
    - In uscita: il risultato dell'esecuzione di P_L



Problemi L-risolubili

Definizione (Problema L-risolubile)

Un problema è L-risolubile se esiste un programma scritto in L che lo risolve.

- Esistono problemi Java-risolubili, C-risolubili, ...
- Qual è la relazione tra la classe dei problemi L-risolubili e quella dei problemi CR-risolubili?
  - Classe più ampia/meno ampia/stessi elementi?

Tesi di Church-Turing

Teorema (Potenza espressiva dei linguaggi di programmazione)

Tutti i linguaggi di programmazione hanno la stessa potenza: risolvono gli stessi problemi, che possiamo identificare con quelli risolubili dal semplice calcolatore CR

- Tutti i linguaggi di programmazione progettati sinora hanno la stessa potenza
- Differenze tra linguaggi: ~~potenza espressiva~~, semplicità d'uso, facilità di apprendimento, eleganza, ...
  - Vero anche per i linguaggi che non sono stati ancora inventati?

Tesi (Tesi di Church-Turing)

Ogni problema che sia intuitivamente risolubile mediante un algoritmo è CR-risolubile

(principio empirico)

Parte III

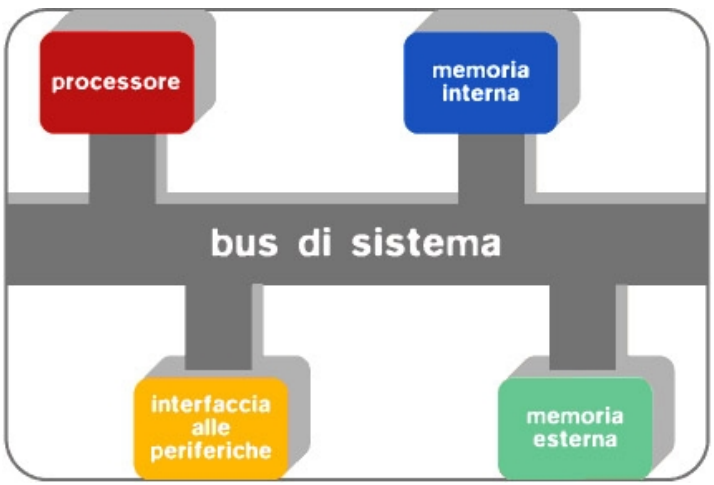
Problemi irresolubili

Macchina CR universale

- È possibilie compiere operazioni aritmetiche tramite manipolazioni testuali
  - ASCII/Unicode: *carattere* $\leftrightarrow$ *numero*
- Interpretazione: il programma P_L è il dato di input
- Programma universale **UNIV**
  - Può simulare l'esecuzione del programma fornito come dato
  - Dati: il **testo** di un programma P e dei dati per P (D_P)
  - Esecuzione di **UNIV**$(P, D_P) \equiv$ esecuzione di $P(D_P)$
- **UNIV** è una **macchina universale**, cioè può simulare qualsiasi altro programma per CR.
- Il nostro **computer** è una macchina universale.
  - Può essere usato per eseguire programmi diversi
  - Interprete per L_{CR}
- Quindi basta fare riferimento all'architettura di **UNIV** per descrivere l'archiettura di un generico computer

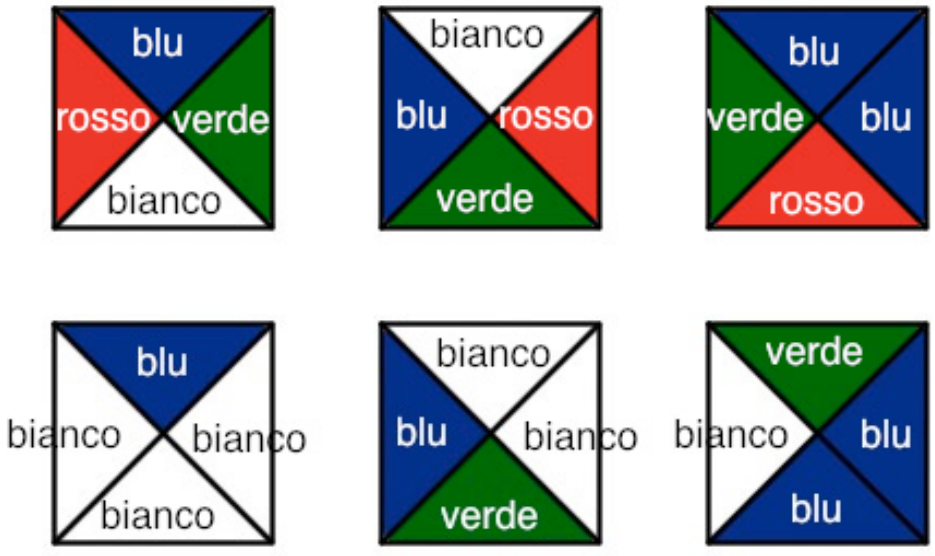
Architettura di Von Neumann

- Realizza l'universalità
  - Programma registrato in memoria
  - Dati registrati in memoria
  - Processore esegue un ciclo di interpretazione utilizzando programma e dati



Esistono problemi non CR-risolubili

- Tiling problem (Harel)
  - Dato: un insieme finito di tipi di piastrelle



  - Vincolo sul colore dei lati di piastrelle adiacenti
  - **Problema**: determinare se un tipo fornito in ingresso è in grado di piastrellare qualsiasi stanza, di qualsivoglia forma o dimensione.

Esistono problemi non CR-risolubili

Teorema (Irresolubilità assoluta del tiling problem)

*È **impossibile** scrivere un programma CR in grado di determinare se un tipo fornito in ingresso è in grado di piastrellare **qualsiasi** stanza, di qualsivoglia forma o dimensione.*

- Irresolubilità dipende dalla **struttura** del problema
- Motivazione intuitiva: non posso sapere se ho considerato **tutte** le possibili forme e dimensioni della stanza

Esistono problemi non CR-risolubili

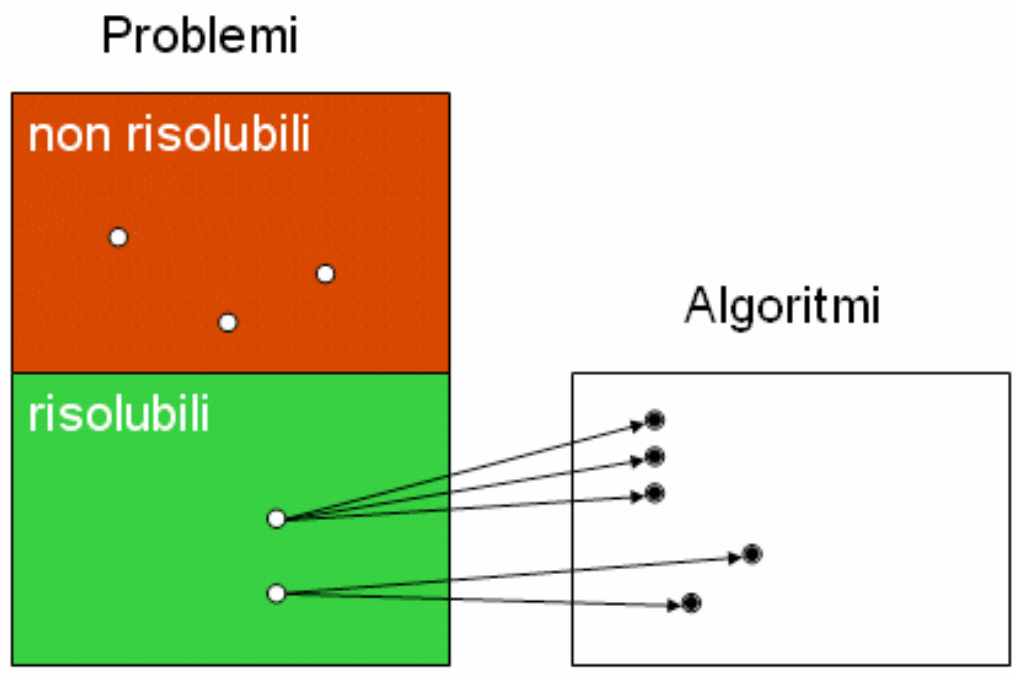
- Halting problem (Turing)
  - Esistono programmi che possono andare in ciclo.
  - Dato un programma, è possibile determinare che non andrà mai in ciclo per qualsiasi valore dei dati di ingresso?

Teorema (Irresolubilità assoluta dell'halting problem)

*È **impossibile** scrivere un programma CR che, preso come dato un altro (generico) programma P e un dato n per P, termina stampando SÌ se $P(n)$ termina; e termina stampando NO se $P(n)$ va in ciclo.*

Problemi non algoritmicamente risolubili

- Ogni problema algoritmicamente risolubile è CR-risolubile (Church-Turing)
- Esistenza di problemi non CR-risolubili: una limitazione assoluta, che **non dipende da CR** o dalla nostra capacità di inventare **linguaggi**
- Problemi irresolubili in senso assoluto: **indecidibili**



Parte IV

Complessità di calcolo

Costo del calcolo

- Consideriamo solo i problemi CR-risolubili
- Quanto costa risolvere un problema di questa classe?
  - Costo = *tempo di calcolo*
  - Per indipendenza dal tipo di calcolare: **passi di calcolo**
  - Una qualsiasi operazione elementare costa una unità di tempo
  - Per semplicità: si guarda solo alla **dimensione** dei dati di ingresso, non al loro specifico valore
    - Somma: costo dipende dalla dimensione degli addendi
    - Minimo di un insieme: dipende dal numero di elementi
    - ...

Un esempio

- Consideriamo di nuovo il programma in L_{CR} per la somma:
  1. **Leggi** un numero e memorizzalo in C_1.
  2. **Leggi** un numero e memorizzalo in C_2.
  3. **Memorizza** zero in C_3.
  4. **Se** C_2 e C_3 contengono lo stesso numero, vai all'istruzione (8).
  5. **Somma** uno alla cella C_1.
  6. **Somma** uno alla cella C_3.
  7. **Se** C_2 e C_2 contengono lo stesso numero, vai all'istruzione (4).
  8. **Mostra** la cella C_1.
  9. **Stop**.
- Quanto costa sommare due numeri?
  - istruzioni (1)–(3): 3 passi iniziali
  - istruzioni (4)–(7): 4 passi *per ogni valore da 1 a* C_2
  - istruzioni (4): 1 passo *quando* $C_2 = C_3$
  - istruzione (8): 1 passo
- Costo: tempo di calcolo (**complessità**) $T(n) = 4 \times n + 5$
- $n =$ valore di C_2

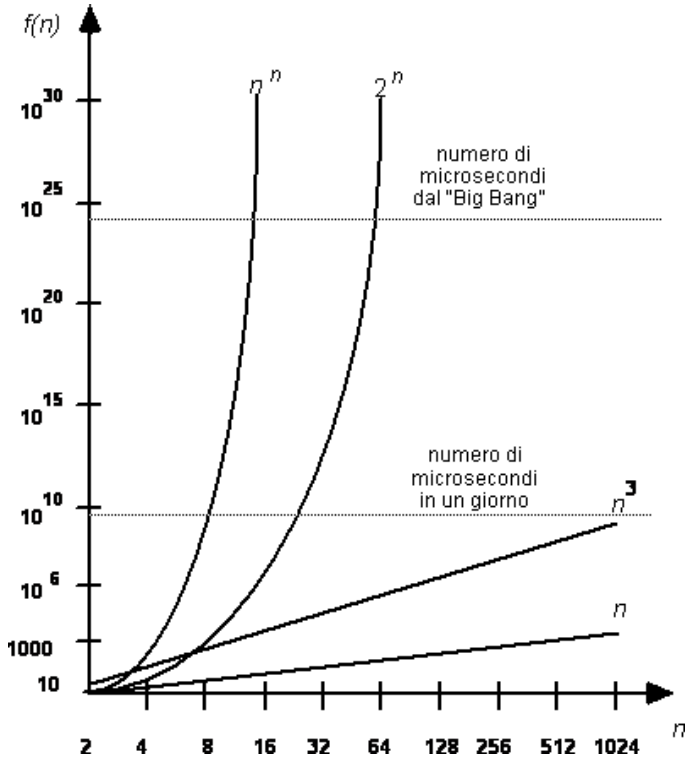
Tassi di crescita

- Interessa sapere l'andamento di $T(n)$ al crescere di n
- Esempio di prima: $T(n) \sim 4 \times n$ (complessità **lineare**)
- Anche: $T(n) = \mathcal{O}(n)$
- **Classi di complessità**

$\mathcal{O}(1)$	Complessità costante: $T(n) \sim K$
$\mathcal{O}(n)$	Complessità lineare: $T(n) \sim K \times n$
$\mathcal{O}(n^p)$	Complessità polinomiale: $T(n) \sim K \times n^p$
$\mathcal{O}(2^n)$	Complessità esponenziale: $T(n) \sim K \times 2^n$

Tempo esponenziale

- Cosa significa in pratica complessità esponenziale?
  - Ipotesi: un'operazione elementare $\Rightarrow 1\mu s = 10^{-6}s$
  - Complessità di P esponenziale: $T(n) = 2^n$
  - Dato di dimensione 50
  - Quanto tempo per ottenere la soluzione?



Trattabilità e intrattabilità

- Esiste una linea di confine tra algoritmi con complessità polinomiale ed esponenziale

Definizione (Problemi intrattabili)

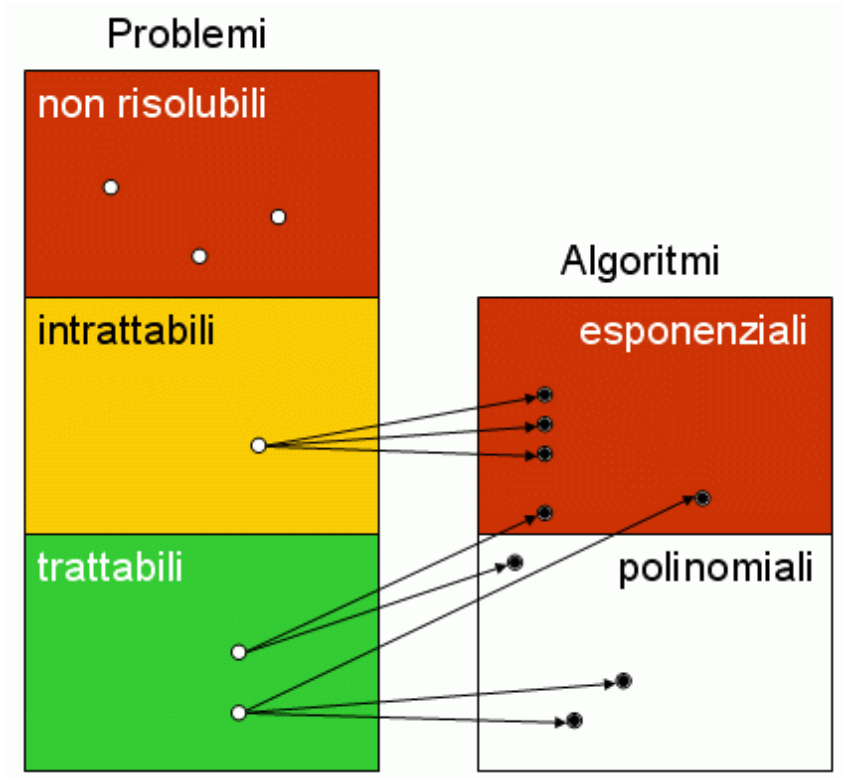
*Un problema è intrattabile se **tutti gli algoritmi** che lo risolvono hanno complessità per lo meno **esponenziale***

Definizione (Problemi trattabili)

*Un problema è trattabile se esiste per esso **almeno un algoritmo** risolutivo con complessità **polinomiale***

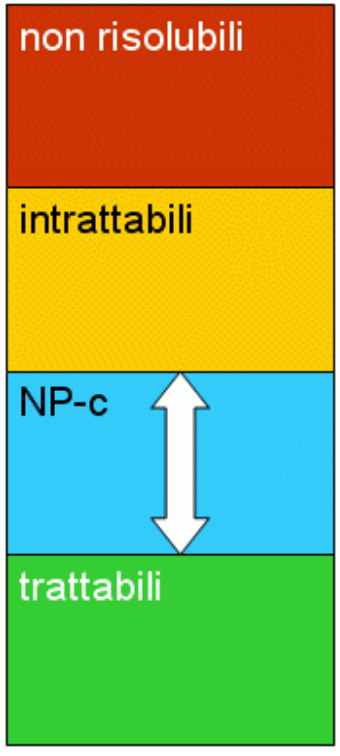
Trattabilità e intrattabilità

- Problemi intrattabili $\Rightarrow$ risolubili solo per dimensioni dei dati molto limitate.
- È sempre possibile inventare algoritmi inutilmente inefficienti



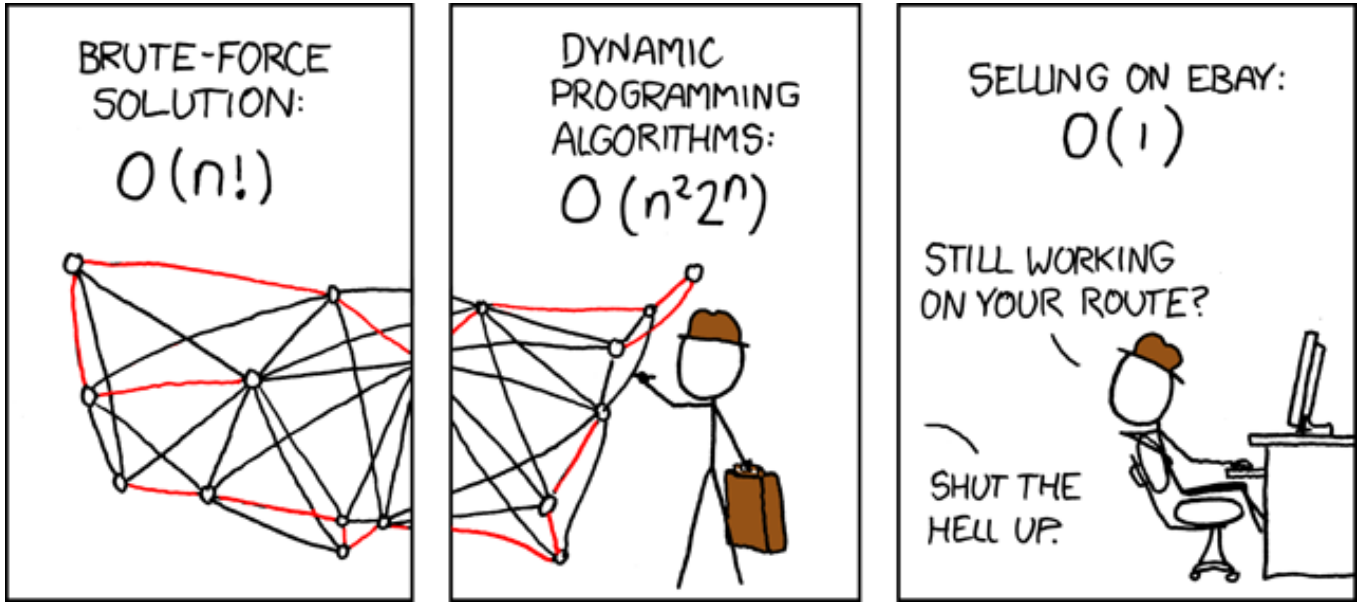
Alcuni esempi di problemi

- Problemi trattabili
  - Ordinamento ($nlogn$)
  - Cammino minimo (n^2)
  - Sottostringa
- Problemi non risolubili
  - Ricoprimento
  - Terminazione
- Problemi intrattabili
  - Ricoprimento finito (data una stanza specifica)



Problemi NP-completi

- Esistono problemi per cui
  - esiste una soluzione esponenziale
  - non è dimostrato che non esista una soluzione polinomiale
- Alcuni esempi:
  - Orario
  - Commesso viaggiatore



http://xkcd.com/399/