Aniello Murano

Overview del corso

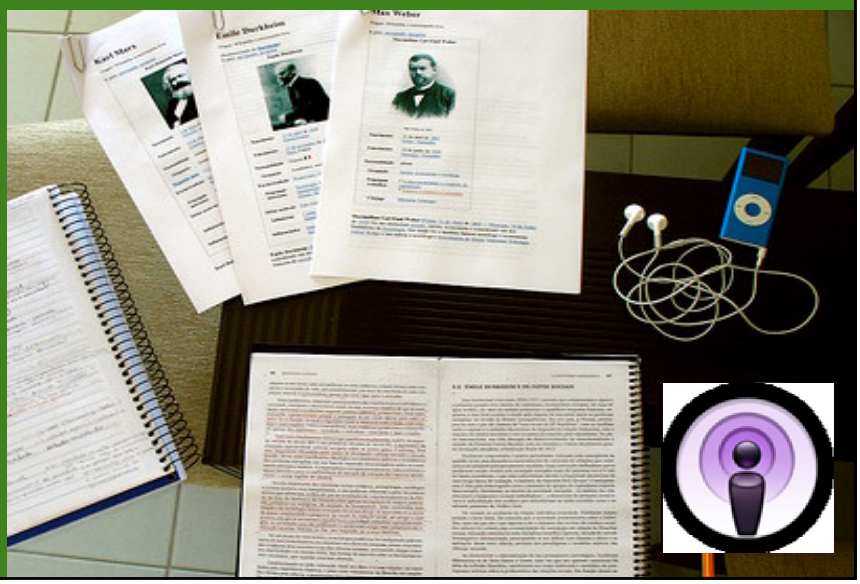
Lezione n.1
Parole chiave:
Overview

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Informazioni Generali sul Corso

- Esame: **Calcolabilità e Complessità**
- Libri di testo:
  - Michael Sipser. Introduction to the Theory of Computation, PWS, 1997
- Approfondimenti:
  - Christos H. Papadimitriou. *Computational Complexity*, Addison Wesley, 1994
  - J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
  - M. R. Garey and D. S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness. Freeman, 1979.
  - H. R. Lewis and C. H. Papadimitriou, Elements of the Theory of Computation. Prentice-Hall, 1981.
  - Michael R. Garey and David S. Johnson. Computers and Intractability, W. H. Freeman, 1979.
- Modalità d'esame: Una prova scritta seguita da una prova orale facoltativa. In particolare, si valuterà durante lo svolgimento del corso la possibilità di una prova di verifica intermedia e delle tesine di esonero di parti di esame.
- Modalità Lezioni: frontale/e-learning.
- Durata del corso frontale: 72 ore pari a 36 lezioni da 2 ore. 24 Lezioni si svolgeranno nel periodo Ottobre-Gennaio. Le rimanenti 12 lezioni saranno svolte nel periodo Marzo-Giugno

Informazioni sul Docente

- Prof. Dr. Aniello Murano, ricercatore universitario presso la Sezione di Informatica del Dipartimento di Scienze Fisiche – Università degli Studi di Napoli, Federico II – Abilitazione a professore di II fascia -
- Sito web: http://people.na.infn.it/~murano/
- Ricevimento: su appuntamento
- E-mail: murano@ na.infn.it
- Altre informazioni sul docente:
  - –Laurea in Scienze dell'informazione, Dottorato di ricerca e post-doc in Informatica presso l'Università degli Studi di Salerno. Researcher per un anno presso la Rice University di Houston (TX-USA). Post-doc per un anno presso la Hebrew University di Gerusalemme (Israele). Collabora con vari gruppi di ricerca in Italia e all'estero.
  - –Altri insegnamenti:
    - •Lab. di ASD - Laurea in Informatica, Facoltà di Scienze MM.FF.NN. - Università "Federico II".
    - •Fondamenti di Linguaggi di programmazione: Laurea Magistrale in Informatica, Facoltà di Scienze MM.FF.NN. - Università di Napoli "Federico II".
    - •Corsi al Dottorato di Ricerca in Informatica ed altro…
  - –Interessi principali di ricerca: Teoria degli Automi e dei Linguaggi Formali. Logiche Temporali discrete e real-time, Metodi Formali per la Specifica, la Verifica e la Sintesi di sistemi hardware e software, Model Checking, Teoria dei Giochi.

Obiettivi del Corso

- Il corso è costituito da un'introduzione alla **calcolabilità** dei problemi e alla loro **complessità** strutturale.
- **La teoria della calcolabilità** si occupa di stabilire se un problema è risolvibile o meno.
- **La teoria della complessità** è quella parte della teoria della calcolabilità che si occupa di stabilire le quantità di risorse necessarie durante la computazione per risolvere un dato problema. Solitamente, per risorse si intende:
  - **Tempo** (quante operazioni occorre fare per risolvere il problema
  - **Spazio** (quanta memoria ha bisogno una algoritmo per essere eseguito)
  - **Altre risorse** come ad esempio "quanti processi paralleli sono necessari per risolvere un dato problema".
- Scopo di questo corso è fornire dunque agli studenti gli strumenti necessari per comprendere e affrontare la difficoltà nel risolvere alcuni problemi comuni da un punto di vista computazionale.

Un esempio pratico(1)

Il problema degli ospiti al tavolo

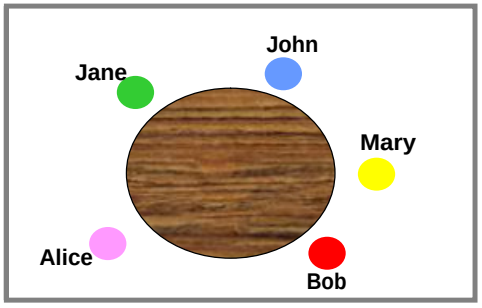
Problema: Dati n ospiti e una lista di preferenze (piace/non piace), posizionare tutti gli ospiti intorno al tavolo in modo che ognuno sia seduto vicino ad un suo ospite preferito

Un algoritmo "Naive": Per ogni possibile distribuzione degli ospiti verificare che ogni ospite è seduto vicino ad uno che gli piace

Domanda: Quanto tempo richiede questo algoritmo per sistemare correttamente n ospiti ?

	John	Mary	Bob	Jane	Alice
John		♥		♥	
Mary	♥		♥		♥
Bob		♥			♥
Jane	♥	♥	♥		♥
Alice			♥	♥	

Tavola delle preferenze



Una possibile distribuzione

Un esempio pratico(2)

- Nel caso peggiore, l'algoritmo Naive deve eseguire un numero di operazioni dell'ordine di n! (n fattoriale)
- Viene subito da chiedersi se questo tempo, in termini pratici, è un tempo ragionevole. Facciamo un po' di conti …
- Per **n=5** occorrono **120** operazioni.
- Per **n=20** occorrono **2.432.902.008.176.640.000 ($2*10^{19}$)** operazioni. Supponiamo che abbiamo una macchina in grado di eseguire **10^{10}** operazioni al secondo e che 3 anni sono circa **10^8 secondi**. Per n=20 ci vogliono circa **60 anni**!!!!
- Per **n=100** occorrono numero di operazioni dell'ordine di **10^{156}**, dunque un numero di anni dell'ordine di **10^{138}**!!!!!

Problemi trattabili vs Problemi non trattabili

- Per il problema appena visto, cosa è lecito chiedersi e cosa si può concludere:
- Il problema e' **risolvibile**? **Sì** (esiste un algoritmo che lo risolve)
- Esiste un algoritmo con una performance migliore? La risposta negativa si può provare formalmente con il concetto di **"riduzione"** (che vedremo in seguito). Per dare una intuizione, bisogna sapere che esistono problemi per i quali non è mai stato trovato un algoritmo migliore di una certa soglia. La "riduzione" della soluzione di uno di questi problemi a quello in esame, porta a concludere che anche quello in esame si deve comportare allo stesso modo.
- Se il miglior algoritmo ha un tempo di esecuzione che nel caso peggiore si comporta come quello Naive visto in precedenza, si conclude che il problema affrontato richiede un tempo esponenziale e dunque che si tratta di un problema "**intrattabile**". In termini asintotici, si dice che l'algoritmo ha una complessità di tempo $O(2^{n \log n})$.

Programma del corso (I parte)

- Concetto di modello di calcolo, risorsa computazionale, algoritmo efficiente e problema trattabile.
- Problemi computazionali: descrizione, istanze, codifica, relazione con i linguaggi: automi e PDA.
- Modelli di calcolo: Macchina di Turing (MdT). Introduzione della classe P
- Le classi di complessità P e NP. Relazione tra NP e P.
- Complessità in spazio
- Teorema di inclusione tra classi in tempo e in spazio.
- Concetto di Macchina di Turing Universale. Teorema di Savitch.
- Riduzioni e completezza! Riduzioni e completezza! Riduzioni e completezza!
- Riduzioni e completezza! Riduzioni e completezza! Riduzioni e completezza!

Per la seconda parte verranno trattati problemi specifici della logica temporale, teoria degli automi e della teoria dei giochi. Per tutti questi problemi verranno poi affrontate questioni di Calcolabilità e Complessità\

Alcuni concetti fondamentali

- Nella teoria della calcolabilità i seguenti tre concetti fondamentali si susseguono costantemente:
- **Problema**: **Cosa** si vuole computare
- **Modello**: **Chi** risolve il problema
- **Algoritmo**: **Come** si risolve il problema

- Il problema si può vedere come una funzione che dato un **input** (una **istanza** del problema) produce il corretto **output**.
- Un esempio di problema può essere la ricerca di un valore **x** in un **ABR T**, dove l'input è una coppia **(T,x)** e l'output è "**yes**" se x è in T.
- In questo corso ci occuperemo principalmente di problemi decisionali, di ricerca, di ottimizzazione, giochi, ecc.

- Un modello di computazione può essere un computer sequenziale (o una macchina di von Newman), un computer parallelo, un circuito booleano, ecc. In questo corso considereremo principalmente macchine di Turing (deterministiche, nondeterministiche, a più nastri, ecc.) nonché automi e PDA.

- Un algoritmo, è un **metodo** per risolvere il problema dato su un determinato modello computazionale.

Risorse di calcolo e complessità dei problemi

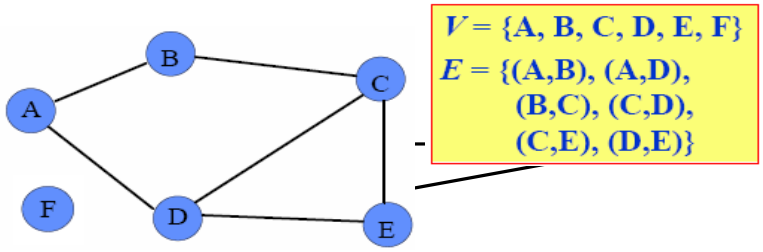
- L'esecuzione di un algoritmo comporta il dispendio di risorse di vario tipo: tempo, spazio, processori (nel caso di computazioni parallele), canali di comunicazioni (nel caso di computazione distribuita), ecc.
- La complessità di un algoritmo rispetto ad una data risorsa è l'ammontare di quella risorsa consumata per la sua esecuzione. Dunque, a seconda della risorsa di interesse si può parlare di complessità di tempo (**time-complexity**), di complessità di spazio (**space-complexity**), ecc. In questo corso ci occuperemo quasi esclusivamente di time- e space-complexity.
- **La complessità di un problema** rispetto ad una data risorsa è la minima quantità di quella risorsa necessaria affinché un algoritmo possa risolvere quel problema.
- L'obiettivo principale della teoria della complessità è

Caratterizzare la complessità dei problemi
e classificarli in classi di complessità

- Una **classe di complessità** è un insieme di problemi, ognuno dei quali ha una complessità in un range ben definito.

Raggiungibilità di un nodo in un grafo

- Un grafo è una coppia (V, E), dove
  - V è un insieme di nodi, chiamati vertici
  - E è un insieme di coppie di nodi, chiamati archi
  - Un arco è una coppia (v,w) di vertici in V
- Esempio:



- Un grafo (V,E) è non orientato se l'insieme degli archi E è un insieme di coppie non ordinate
- Un grafo (V,E) è orientato se l'insieme degli archi E è una relazione binaria tra vertici.

Raggiungibilità di un nodo in un grafo

- Input: Grafo orientato G = (N,E), nodo sorgente s, nodo target t
- Questione da risolvere: verificare se c'è un percorso in G da s a t?
- Problema decisionale: output = Si/No

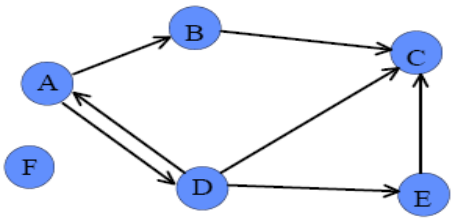
- **Graph Search**

```
for each v ∈ N-{s} do {mark[v]=0}
mark[s]=1 ;
Q = {s}
while Q ≠∅ do
    { select a node u and delete if from Q
      for each v ∈ N do
      if (A[u,v]=1 and mark[v]=0) then
      {mark[v]=1; Q=Q ∪ {v} }
    }
If mark[t]=1 then print "Yes" else print "No"
```

- Per calcolare la complessità di questo algoritmo si deve considerare come è implementato G

Matrice di adiacenza

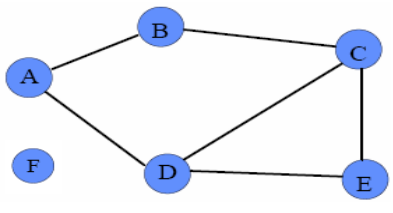
$$M(v, w) = \begin{cases} 1 & \text{se } (v, w) \in E \\ 0 & \text{altrimenti} \end{cases}$$

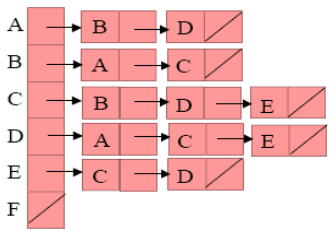


	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	1	0	1	0	0	0
E	0	0	0	1	0	0
F	0	0	0	0	0	0

Lista di adiacenza

$L(v)$ = lista di w, tale che $(v, w) \in E$,





Time Complexity

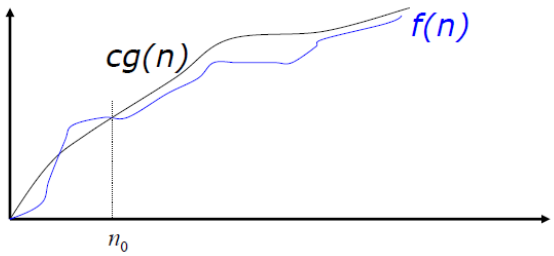
- Si assuma che G sia implementato con matrice di adiacenza.
- L'algoritmo visita ogni elemento della matrice solo una volta quando il vertice corrispondente alla sua riga è stato scelto.
- Inoltre, tutte le altre operazioni richieste dall'algoritmo come, scegliere un elemento di Q, marcare un vertice, o dire se esso e marcato o meno, possono essere eseguite in tempo costante
- Siccome la matrice ha n^2 posizioni dove n è il numero dei nodi nel grafo, si deduce che la time-complexity è proporzionale a n^2. Cioè la time-complexity è $O(n^2)$.
- La notazione O esprime la complessità dell'algoritmo nel caso peggiore. In pratica, la notazione O permette di sopprimere le costanti dal calcolo preciso della complessità.
- In realtà è possibile provare che la complessità è indipendente dalla implementazione di G e dalla implementazione di S. In particolare, anche nel caso in cui S sia una coda (in tal caso l'elemento si scegli in base ad una BFS) o nel caso di uno stack (in tal caso l'elemento si scegli in base ad una DFS).

Notazione Asintotica

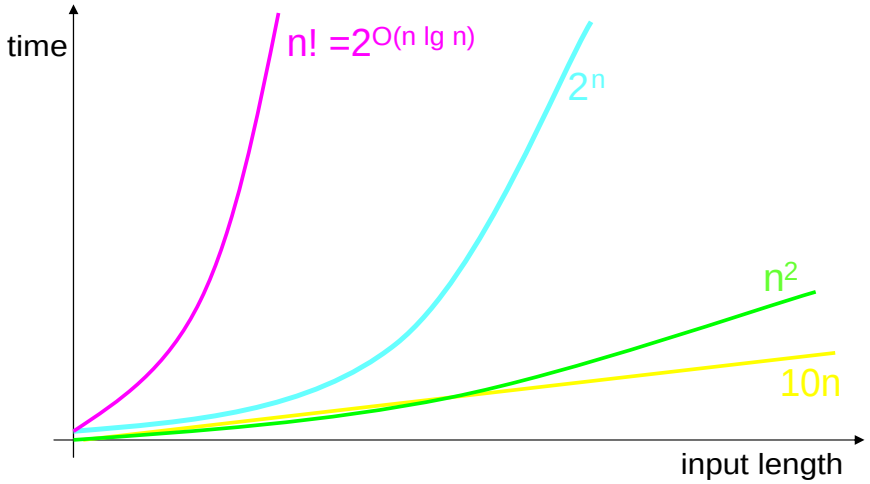
"Big O"

- $O(g(n)) = \{f(n) \mid$ per ogni $c>0$ e n_0 tale che per ogni $n>n_0$: $0 \leq f(n) \leq c\, g(n)\}$
- Per indicare che una certa funzione f(n) è nell'insieme appena definito, si usa scrive $f(n) = O(g(n))$. Si noti qui che = indica appartenenza.

- Esempi: $7n = O(n^2)$. $7n^2 = O(n^2)$. $7n^2$ non è $O(n)$.



Growth Rate: Sketch



I problemi rispetto alla complessità per risolverli

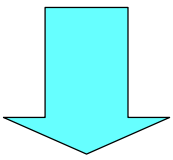
Trattabile	Intrattabile
polinomiale ≡ $n^{O(1)}$	esponenziale ≡ $_2n^{O(1)}$

Un altro problema

- Si supponga di avere una serie di città (rappresentate da un grafo orientato).
- Problema: E' possibile visitare tutte le città esattamente una volta.
- Soluzione (naive): Si cercano cioè tutte le città raggiungibile e che non sono state ancora visitate e per queste si considera una possibile visita delle città
- Domanda: Questo problema è trattabile?
- Possibili risposte:
  - Sì! Fornendo un algoritmo efficiente.
  - NO! Provandolo formalmente
- Per il No si può utilizzare il concetto di riduzione fra problemi
- Per esempio, uno si può chiedere se è possibile risolvere il problema degli ospiti a tavola una volta risolto il problema del tour

Relazione tra Problemi

Assumendo l'esistenza di un algoritmo efficiente per il "**problema del tour**", c'è allora un algoritmo efficiente per il "**problema degli ospiti a tavola**"



Il problema degli ospiti a tavola (OSPITI) non può essere "radicalmente" più difficile del problema del tour (TOUR)

Riduzione

OSPITI $\leq_p$ TOUR

OSPITI non può essere più difficile di TOUR

In altre parole

TOUR è almeno **tanto difficile quanto** OSPITI

Sebbene per alcuni problemi non siamo in grado di ottenere algoritmi efficienti, o provare che non ce ne sia uno, la tecnica della riduzione ci permette di definirne la sua difficoltà, mettendolo in relazione con altri problemi di cui si conosce la effettiva difficoltà

La classe dei Problemi NP-completi (NP-C)

Contiene una miriade di problemi

Ognuno riducible a tutti gli altri

☑ algoritmi esponenziali

? algoritmi efficienti