Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica
Curriculum SR (Sistemi e Reti)

Algoritmi e Laboratorio a.a. 2006-07
Lezioni

prof. Elio Giovannetti

Parte 1 – Problemi e algoritmi

Che cosa è un algoritmo ?

- un algoritmo è un *procedimento di calcolo*, ossia un procedimento ben definito che può essere eseguito meccanicamente da un uomo o da una macchina per eseguire un dato compito;
- algoritmo NON è sinonimo di diagramma di flusso; anzi, i diagrammi di flusso non sono oggi, in generale, una buona tecnica di descrizione degli algoritmi;
- algoritmi sono stati inventati ben prima della nascita dei calcolatori:
  - uno dei più antichi è l'algoritmo di Euclide per il MCD;
  - la parola algoritmo deriva dal nome del matematico persiano al-Khwarismi خوارزمی (vissuto intorno all'anno 800), il cui libro "Calcoli con i numerali indiani" descriveva gli algoritmi di calcolo per le operazioni aritmetiche con il sistema di numerazione indiano, cioè quelli che ancora oggi studiamo nella scuola elementare;
- l'*algoritmica*, ovvero la scienza degli algoritmi, è fin dagl'inizi dell'informatica una delle sue discipline centrali, ancora oggi attivo campo di ricerca.

Cos'è un problema algoritmico (o computazionale) ?

Esempi (informali):

- Problema del massimo di una sequenza di interi: Data una sequenza (non vuota) di numeri interi, trovare il massimo di tale sequenza e l'indice (o posto) di un tale valore massimo nella sequenza.
- Problema della ricerca in una sequenza ordinata: Data una sequenza ordinata **S** di elementi e dato un elemento **e,** determinare se **e** compare nella sequenza o no, e se si l'indice del posto in cui compare.
- Problema del cammino minimo: Data una mappa stradale che riporti località, strade e lunghezze dei percorsi, e data una coppia di località (raggiungibili l'una dall'altra), trovare un percorso di lunghezza minima fra di esse.
- Problema della più lunga sottosequenza comune: Date due sequenze di DNA, trovare la più lunga sottosequenza comune alle due sequenze (avendo definito in modo preciso le nozioni di sequenza di DNA e di sottosequenza).
- …

Problema algoritmico o computazionale

Consiste di:

- precondizione: una specifica precisa dei tipi dei dati di partenza (parametri di ingresso) e delle condizioni cui devono soddisfare;
- postcondizione: una specifica precisa dei tipi di dati costituenti la risposta (risultato) e della relazione che vi deve essere fra l'input e l'output, ossia una specifica che definisca qual è, per ogni istanza dell'input, la risposta o l'insieme delle possibili risposte che si vogliono ottenere;

NOTA BENE:

L'enunciazione di un problema algoritmico NON specifica COME ottenere l'output dall'input;

COME ottenere l'output dall'input costituisce la SOLUZIONE del problema.

Soluzione di un problema algoritmico

È un algoritmo, cioè la descrizione precisa di un procedimento "meccanico" che permette, per ogni input (soddisfacente alla relativa specifica) di trovare un output (che soddisfi alla relativa specifica).

Nota terminologica

Nei corsi di Programmazione 1 e 2 vi si è insegnato a NON confondere l'input-output con il passaggio-parametri.

Nello studio astratto degli algoritmi si parla spesso di:

- parametri di input, o semplicemente **input**, per indicare i dati iniziali su cui opera l'algoritmo, in qualunque modo ricevuti;
- parametri di output, o semplicemente **output**, per indicare i dati finali, o risultati, dell'esecuzione dell'algoritmo, in qualunque modo restituiti;

Che nella realizzazione concreta si tratti di un effettivo input/output (su periferiche), oppure di passaggio-parametri e restituzione di risultati fra procedure, è ovviamente inessenziale.

Non si confondano quindi input e output di un algoritmo, in senso astratto, con l'input/output su periferiche operato da un programma.

Esempi rivisitati

Problema del massimo di una sequenza di interi.

dato di partenza: una sequenza (non vuota) di numeri interi $n_1, n_2, \ldots, n_m$;

risposta: un indice (cioè un numero naturale) **imax**, compreso fra 1 ed m, tale che $n_{imax} >= n_i$ per ogni naturale **i** compreso fra **1** ed **m** inclusi.

Problema della ricerca in una sequenza ordinata di tipo generico.

dato di partenza:

tipo:

1) una sequenza (eventualmente vuota) $e_1, e_2, \ldots, e_m$ di elementi appartenenti ad un dominio (insieme) E, di solito infinito, su cui è definita una relazione d'ordine totale <=;

2) un elemento **e** di tale insieme E;

precondizione: la sequenza è ordinata in modo crescente rispetto a tale ordine, cioè: $e_1 <= e_2 <= \ldots <= e_m$

risposta:

tipo: un valore di verità;

postcondizione: il valore è TRUE o FALSE a seconda che l'elemento **e** compaia o no nella sequenza.

Il “che cosa” e il “come”

- Problema: descrive “che cosa” si deve calcolare.
- Algoritmo: descrive “come” effettuare un calcolo.

NOTA:

Nella realtà produttiva, quando si deve realizzare un’applicazione software può essere non banale identificare quali sono i problemi.

Efficienza di un algoritmo

Una corretta implementazione dell’algoritmo quanto tempo ci mette, e di quanto spazio di memoria ha bisogno, per produrre la risposta per un input di una data dimensione ?

Oppure: qual è la massima dimensione dell’input per la quale l’algoritmo produce una risposta in un tempo accettabile e con un’occupazione di memoria accettabile ?

Definiremo con precisione le nozioni di complessità temporale e spaziale di un algoritmo.

Algoritmi e strutture dati

La fattibilità o efficienza di un algoritmo è spesso legata alle caratteristiche della struttura-dati su cui opera, cioè al modo in cui i dati sono rappresentati ed organizzati.

Esempio:
una soluzione del problema della ricerca in una sequenza ordinata è data dall’ algoritmo di ricerca binaria;
tale soluzione è sensata – perché molto più efficiente della ricerca sequenziale – solo se la sequenza è realizzata per mezzo di un array (oppure con un albero di ricerca), NON se la sequenza è realizzata come una lista concatenata !

Lo studio degli algoritmi è quindi inseparabile dallo studio delle strutture-dati.

Problemi algoritmici per i quali non si conoscono soluzioni efficienti.

Data una formula booleana composta di n variabili booleane, stabilire se esiste oppure no una assegnazione di valori di verità alle variabili che renda la formula vera:

A and (B or (not B and C) xor …

L’algoritmo che risolve il problema produce la risposta in un tempo ragionevole solo per formule “piccole”.

Si sospetta che non esista un algoritmo più efficiente, ma non lo si è potuto finora dimostrare.

Nota: è invece facile (cioè esiste un ovvio algoritmo efficiente per) controllare che una data assegnazione, fornita "miracolosamente" da qualcuno, renda effettivamente vera la formula.

Problemi “intrinsecamente” difficili
cioè problemi algoritmici per i quali si sa (= si è dimostrato) che non possono esistere soluzioni efficienti.

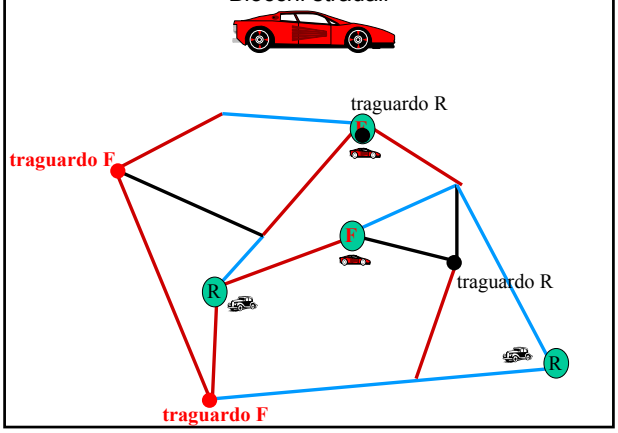
Esempi

Il problema delle Torri di Hanoi.
Il problema *dei blocchi stradali (roadblocks).*

Il problema dei blocchi stradali: il gioco.

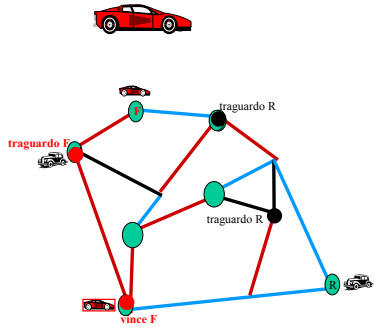
- È un gioco con due giocatori (Ferrari e Renault) su una rete di strade che si intersecano; ogni tratto di strada fra due intersezioni è colorato con uno fra tre colori possibili.
- Alcune intersezioni sono marcate “traguardo Ferrari”, altre sono marcate “traguardo Renault”, le rimanenti non sono marcate. Ognuno dei due giocatori possiede un certo numero di auto, ciascuna delle quali inizialmente occupa un’intersezione.
- A turno ogni giocatore sposta una delle sue auto da un’intersezione ad un’altra, lungo un percorso che deve essere tutto dello stesso colore, e su cui non si devono trovare altre auto.
- Vince il primo giocatore che raggiunge con un’auto un’intersezione marcata come proprio traguardo.

Blocchi stradali



Blocchi stradali

Se F muove per primo, può vincere quali che siano le mosse di R.



Il problema

- Determinare se, data un'arbitraria configurazione iniziale del gioco, esiste una strategia vincente per il giocatore che gioca per primo.
- L'algoritmo che risolve il problema impiega un tempo esponenziale nel numero delle intersezioni.
- Nota: occorrerebbe un tempo esponenziale anche solo per dimostrare che una data risposta è corretta.
- Si dimostra che non può esistere un algoritmo migliore.

Telecomunicazioni

- Problema di trovare un percorso non congestionato in una rete:
  - La congestione può essere definita in modo preciso per mezzo del ritardo nella trasmissione e della diminuzione della larghezza di banda.
  - La congestione è variabile nel tempo.
- La soluzione esatta del problema, formulato in modo preciso, è un algoritmo anch'esso esponenziale, e si dimostra che non può esistere un algoritmo migliore.

Problemi algoritmici non risolubili

Non tutti i problemi algoritmici sono risolubili: per alcuni problemi algoritmici si può dimostrare che non può esistere una soluzione, cioè che non può esistere un algoritmo che risolve il problema.

Esempi

Dati due programmi java arbitrari, determinare se essi sono equivalenti, cioè se per qualunque input producono lo stesso output.

Dato un programma java arbitrario, ed un input legale arbitrario per tale programma, determinare se il programma, per quell'input, termina.

Riassumendo

- Problema e algoritmo (il "che cosa" ed il "come")
- Specifica: pre e post-condizioni
- Problemi risolubili, problemi risolubili ma con soluzioni note inutilizzabili, problemi intrinsecamente difficili, problemi insolubili.

Contenuto del corso.

- Analisi di alcuni fondamentali algoritmi e strutture-dati che risolvono importanti problemi computazionali in modo efficiente (non si esaminano problemi "difficili"). Si richiede:
  - comprensione del problema algoritmico;
  - comprensione del perché un dato algoritmo risolve correttamente un dato problema (dimostrazione informale);
  - analisi della complessità degli algoritmi, confronto fra soluzioni diverse di uno stesso problema.
- Realizzazione di tali algoritmi e strutture dati nello stile della programmazione a oggetti.
- Approfondimento delle nozioni di Java necessarie o utili per la suddetta realizzazione. Eventuali cenni a C#.
- Progetto e realizzazione di algoritmi e strutture dati per risolvere problemi "simili" a quelli studiati.

Riferimenti e bibliografia

Demetrescu, Finocchi, Italiano. Algoritmi e strutture dati. McGraw-Hill, 2004.
sito web: http://www.ateneonline.it/demetrescu/

Crescenzi, Gambosi, Grossi. Strutture di dati e algoritmi. Pearson Addison-Wesley, 2006.

Bertossi. Algoritmi e strutture di dati. UTET, 2000.

Cormen, Leiserson, Rivest, Stein,
Introduzione agli algoritmi e strutture dati, Seconda edizione McGraw-Hill, 2005.

per il laboratorio:

Horstmann, Big Java, 2nd Edition. John Wiley & Sons, 2006.