Corso di laurea in Informatica
Introduzione agli Algoritmi

Syllabus e Introduzione

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Di cosa tratta questo corso

- **Problem solving di problemi computazionali**

- **Algoritmi**

descriveremo alcuni algoritmi “classici” per risolvere problemi di base (ricerca e ordinamento)

- **Efficienza**

Valuteremo l’efficienza, calcolandone il costo computazionale.

- **Strutture dati**

Esploreremo le strutture dati più adatte da usare

P.S. gli algoritmi verranno descritti tramite pseudocodice, per poterne dare una versione compatta senza perdersi in dettagli implementativi…

Quali problemi?

In questo corso restringiamo la nostra attenzione ai ***problemi computazionali***, problemi cioè che richiedono di descrivere in modo automatico una specifica relazione tra un insieme di valori in ***input*** e il corrispondente insieme di valori in ***output***.

Un algoritmo è ***corretto*** se, ***per ogni istanza di un problema computazionale, termina producendo l'output corretto***.

In tal caso diremo che ***l'algoritmo risolve il problema***.

Esempio di problema computazionale:

Ordinare n numeri dal più piccolo al più grande

Input: (anche detto istanza del problema)
sequenza di n numeri $a_1, a_2, \ldots, a_n$

Output:
permutazione $a'_1, a'_2, \ldots, a'_n$ della sequenza di input con $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Problem solving

Il problem solving è un'attività che ha lo scopo di raggiungere una soluzione a partire da una situazione iniziale.

E' quindi un'attività creativa, di natura essenzialmente progettuale, ed in questo risiede la sua difficoltà.

Approccio al problem solving:

- ***analisi del problema:*** lettura approfondita della situazione iniziale, comprensione ed identificazione del problema;

- ***esplorazione degli approcci possibili:*** identificazione delle metodologie di soluzione tra i metodi noti;

- ***selezione di un approccio:*** scelta dell'approccio migliore;

- ***definizione dell'algoritmo risolutivo:*** identificazione dei dati e progettazione della sequenza di passi elementari da applicare su di essi;

- ***riflessione critica:*** a problema risolto, ripensamento delle fasi della soluzione proposta per identificare eventuali criticità e possibili migliorie.

Efficienza

Un algoritmo è una sequenza di comandi "elementari" ed "univoci" che terminano in un tempo finito ed operano su strutture dati.

Affinché un algoritmo sia utilizzabile, deve concludersi e produrre il suo output entro un tempo "ragionevole".

Un aspetto fondamentale che va affrontato nello studio degli algoritmi è la loro **efficienza**, cioè la ***quantificazione delle loro esigenze in termini di tempo e di spazio***, ossia tempo di esecuzione e quantità di memoria richiesta.

Efficienza

quantificazione delle esigenze dell'algoritmo in termini di tempo e di spazio

Questo perché:

- I calcolatori sono molto veloci, ma non infinitamente veloci;
- La memoria è economica e abbondante, ma non è né gratuita né illimitata.

Nel corso illustreremo il concetto di **costo computazionale** degli algoritmi, in termini di numero di operazioni elementari e quantità di spazio di memoria necessari in funzione della dimensione dell'input.

Random Access Machine

- Per studiare l'efficienza di un algoritmo dobbiamo ANALIZZARLO, cioè prevedere le risorse che esso richiede per la sua esecuzione, **senza che tale analisi sia influenzata da una specifica tecnologia** che, inevitabilmente, col tempo diviene superata.
- Per fare questo, dobbiamo avere un modello astratto di calcolatore, che sia indipendente dalla tecnologia usata.
- In questo corso consideriamo un modello astratto di calcolo detto RANDOM ACCESS MACHINE (RAM).
- La RAM è quindi una ***macchina astratta***, la cui validità e potenza concettuale risiede nel fatto che non diventa obsoleta con il progredire della tecnologia.

Nel modello RAM:

- esiste un **singolo processore**, che esegue le operazioni sequenzialmente.

- esistono delle **operazioni elementari**, l'esecuzione di ciascuna delle quali richiede per definizione un ***tempo costante***. (Es.: operazioni aritmetiche, letture, scritture, salto condizionato, ecc.)

Misure di costo

- Faremo l'ipotesi che ogni operazione aritmetica o di confronto viene eseguita in tempo costante, indipendentemente dalla dimensione degli operandi.
- Questo criterio è detto MISURA DI COSTO UNIFORME.
- Sebbene utile, questo è un modello di costo non sempre realistico perché i dati potrebbero non essere rappresentabili in una sola parola di memoria, e quindi anche le operazioni aritmetiche non richiederebbero, in realtà, tempo costante.

Criterio della misura di costo uniforme

```
def esempio(n):
    x = 1
    for i in range(n):
        x = 2*x
    print(x)
```

(ciclo, reiterato *n* volte, che calcola il valore 2^n)

Il tempo di esecuzione totale (numero di passi elementari) è ***proporzionale ad*** n:

- si tratta di un **ciclo eseguito *n* volte**;
- ad ogni iterazione del ciclo si compiono **due operazioni**, ciascuna delle quali ha costo unitario:
  1. l'incremento del contatore
  2. il calcolo del nuovo valore di x.

Pseudocodice

Affinché ogni esecutore sia in grado di comprendere un algoritmo, è necessario utilizzare una descrizione:

- il più formale possibile
- indipendente dal linguaggio che si intende usare

Pseudocodice

L'importanza di progettare algoritmi di complessità temporale bassa.

- Consideriamo un algoritmo che su input n richiede un numero di passi di circa 2^n e assumiamo che un passo di calcolo richieda circa un secondo.

- Il numero di operazioni per $n = 60$:
  - 2^{60} è approssimativamente il numero di secondi dal Big-Bang che si stima avvenuto circa 13.8 miliardi di anni fa.

- Anche per valori di n relativamente piccoli il tempo di calcolo può rivelarsi proibitivo.

2^{60} è approssimativamente il numero di secondi dal Big-Bang.

- Se un passo viene eseguito in **millisecondi** (10^3 in un secondo) allora avremmo potuto in questo lasso di tempo gestire istanze di dimensione $n \approx 70$ ($2^{60} \times 2^{10} = 2^{70}$)

- Se un passo viene eseguito in **microsecondi** (10^6 in un secondo) allora avremmo potuto in questo lasso di tempo gestire istanze di dimensione $n \approx 80$ ($2^{60} \times 2^{20} = 2^{80}$)

Con l'avvento di macchine più veloci la situazione sostanzialmente non cambia. Un algoritmo con tempi di calcolo esponenziali al crescere di n è inutilizzabile e lo resterà.

Strutture dati

Un algoritmo è una sequenza di comandi "elementari" ed "univoci" che terminano in un tempo finito ed operano su strutture dati.

Per risolvere problemi avremo bisogno di usare dati e quindi useremo delle STRUTTURE DATI, strumenti per memorizzare e organizzare i dati e semplificarne l'accesso e la modifica.

Non esiste una struttura dati che vada bene per ogni problema, quindi dobbiamo conoscere proprietà, vantaggi e svantaggi delle principali strutture dati.

Il progetto o la scelta della struttura dati da adottare nella soluzione di un problema è un aspetto fondamentale per la risoluzione del problema stesso, al pari del progetto dell'algoritmo.

Perciò, gli algoritmi e le strutture dati fondamentali vengono sempre **studiati e illustrati assieme**.

Esempi: vettore, lista, coda, pila…

Dettagli per studiare

Pagina web del corso

Alla pagina:
https://twiki.di.uniroma1.it/twiki/view/Intro_algo/PZ/WebHome
troverete:

- Il programma del corso
- Il diario delle lezioni
- Le modalità d'esame
- Un elenco di libri di testo utili
- Delle dispense e degli esercizi svolti

Libri di testo

Un qualunque manuale di algoritmi e strutture dati fondamentali va bene, ma se dovete comprarne uno…

- Cormen, Leiserson, Rivest, Stein: Introduzione agli algoritmi e strutture dati, Mc Graw Hill

oppure

- Demetrescu, Finocchi, Italiano: Algoritmi e strutture dati, Mc Graw Hill

oppure

- R. Kleinberg ed E. Tardos. Algorithm Design. Addison Wesley.

Suggerimenti

Non basta essere solo “spettatori”:

- Alla fine di (quasi) ogni lezione troverete degli esercizi che vengono proposti.
- Provate a risolverli e, se non ci riuscite, approfondite ulteriormente l’argomento trattato.