Problemi algoritmici e Complessità degli algoritmi

Università degli Studi di Salerno

Corso di Laurea in Scienze della comunicazione

Informatica generale

Docente: Angela Peduto

A.A. 2005/2006

Problemi algoritmici

- Ma quali sono i tipi di problemi che possono essere risolti tramite un algoritmo (“Problemi algoritmici”)?
- Un problema algoritmico è caratterizzato dalla conoscenza precisa di due elementi:
  - L’insieme degli input ammissibili
  - L’insieme degli output desiderati in funzione degli input
- Un problema algoritmico applicato ad un particolare input è detto istanza del problema

Algoritmi computazionali

- **Input**: prevalentemente numerici
- **Output**: numerici, come risultato di elaborazioni computazionali sugli stessi input o da loro determinate
- Esempio num. 1
  - Input: due numeri interi a e b
  - Output: il numero intero $a*b^2$
- Esempio num. 2
  - Input: due numeri interi J e K
  - Output: La somma di tutti gli interi da J e K
  - *Nota: il numero di somme da effettuare (e quindi la durata dell'algoritmo) non è nota a priori (come per il caso precedente), ma la si conosce solo al momento dell'esecuzione dell'algoritmo*

Algoritmi decisionali (1)

- **Input**: qualsiasi
- **Output**: Valore SI se una certa proprietà dei dati input è verificata, NO altrimenti
- Esempio num. 3
  - Input: un numero intero positivo, K
  - Output: SI se K è primo NO altrimenti
  - *Nota: l'algoritmo prevede certamente e l'esecuzione di calcoli, ma il risultato non è numerico*

Algoritmi decisionali (2)

- Esempio num. 4 (commesso viaggiatore – versione decisionale)
  - Input: Carta stradale con n città collegate da strade di cui si conosce la lunghezza e un numero intero K
  - Output: SI se esiste un itinerario di lunghezza massima pari a K che tocca tutte le città. NO altrimenti (cioè non esiste oppure tutti gli itinerari sono più lunghi di K chilometri)

Algoritmi di riorganizzazione dei dati

- **Input**: qualsiasi lista di dati omogenei (parole, numeri, date,…)
- **Output**: stessa lista con gli elementi riorganizzati in base a determinate priorità
- Esempio num. 5 (ordinamento)
  - Input: una lista di parole
  - Output: lista in ordine alfabetico
  - *Nota: il problema è di tipo non numerico e presenta un numero di elementi input che è noto solo al momento dell'esecuzione dell'algoritmo o addirittura solo al suo termine.*

Algoritmi di ricerca (1)

- **Input**: due liste di dati omogenei (parole, numeri, date,…). La seconda lista può consistere anche di un solo elemento
- **Output**: Elenco delle strutture comuni alle due liste
- Esempio num. 6 (ricerca)
  - Input: Un testo e una singola parola (detta pattern)
  - Output: Tutte le occorrenze del pattern
  - *Nota: Il problema è di tipo non numerico e presenta un numero di elementi input che è noto solo al momento dell'esecuzione dell'algoritmo o addirittura solo al suo termine*

Algoritmi di ricerca (2)

- Esempio num. 7 (String matching)
  - Input: Un testo e un dizionario
  - Output: Tutte le occorrenze delle parole del dizionario nel testo
  - *Note: il problema è di tipo non numerico e presenta un numero di elementi input che è noto solo al momento dell'esecuzione dell'algoritmo o addirittura solo al suo termine. Inoltre devono essere individuate anche parole parzialmente sovrapposte. Spesso si vuole conoscere anche la frequenza delle occorrenze delle singole parole.*

Algoritmi di ricerca (2)

- Esempio num. 8 (Calcolo del massimo o del minimo)
  - Input: Una lista di oggetti omogenei (parole, numeri, …)
  - Output: l'elemento più grande o quello più piccolo
  - *Nota: Il problema presenta un numero di elementi input che è noto solo al momento dell'esecuzione dell'algoritmo o addirittura solo al suo termine. Esso può riguardare un insieme di parole, come un insieme di numeri che, per esempio, rappresentano l'età di un gruppo di persone di cui si vuole conoscere la più anziana o la più giovane*

Algoritmi di ottimizzazione (1)

- **Input**: un insieme di elementi omogenei di tipo qualsiasi purché sia loro associata una funzione di costo (o di misura). Il problema può presentare dei vincoli sulle modalità di raggruppamento o di organizzazione dei dati
- **Output**: Riorganizzazione dei dati input in modo da ottimizzare la funzione di costo
- Esempio num. 9 (commesso viaggiatore – versione ottimizzazione)
  - Input: carta stradale con n città collegate da strade di cui si conosce la lunghezza e due città fissate, A e B
  - Output: il tragitto più breve che unisca A e B

Algoritmi di ottimizzazione (2)

- Esempio num. 10 (Bin Packing – versione monodimensionale)
- **Input**: una lista di oggetti omogenei (pacchetti) a cui è associata una misura, espressa da un numero reale compreso tra 0 e 1 e un insieme di contenitori (Bin) tutti della stessa capacità (pari ad 1).
- **Output**: Raggruppamento dei pacchetti in modo da utilizzare il minor numero di bins.
- ***Nota**: Il problema trova moltissime applicazioni*
  - *Distribuzione di spot pubblicitari all'interno di trasmissioni TV*
  - *Taglio di elementi da nastro trasportatore*
  - *Ottimizzazione del carico di una flotta di camion*
  - *Distribuzione di brani musicali su un insieme di CD*
  - *Distribuzione dei processi a un sistema multiprocessoe*
  - *Algoritmi di Bin Packing sono attualmente utilizzati per la allocazione della RAM in un sistema multiprogrammato*

Un problema due algoritmi

- Si vuole calcolare la **somma dei primi 100 numeri interi positivi**, cioè la somma 1+2+3+…+99+100
- Algoritmo diretto (detto di *forza bruta*)

```
Var x, somma integer;
begin
somma:=0;
x:=0;
while x < 100 do
begin
    x:= x+1;
    somma:=somma+x;
end;
write(somma);
end.
```

Un problema e due algoritmi (2)

- Oppure utiliziamo la **formula di Gauss** secondo la quale 1+2+3+…+99+100**=(100*101)/2** e diamone l'algoritmo

```
var somma: integer;
begin
    somma:=(100*101)/2;
    write(somma);
end
```

- Quindi questo è un problema computazionale che ammette più di un algoritmo

Un problema e due algoritmi (2)

- Generalizzando l'algoritmo per la somma dei primi N numeri interi positivi

```
var N, x, somma: integer;
begin
1. read(N);
2. Somma:=0;
3. x:=0;
4. while x < N do
   begin
5.     x:=x+1;
6.     somma:=somma+x;
   end;
7. Write(somma);
```

Un problema e due algoritmi (3)

- Formula di Gauss: 1+2+3+…+N=(N*(N+1))/2

Var N, somma: integer;
begin

1. Read (N);
2. Somma:=(N*(N+1))/2;
3. Write(somma);

End.

Quante operazioni sono richieste da ciascun algoritmo?

- Il primo algoritmo richiede:
  - 1 operazione di lettura dell'input (istruzione1),
  - 2 operazioni di inizializzazione (istruzioni 2 e 3),
  - N confronti (confronto nell'istruzione 4, ripetuto N volte),
  - 2*N somme (istruzioni 5 e 6 ripetute N volte l'una)
  - 1 operazione finale di stampa del risultato (istruzione 7)

Quante operazioni sono richieste da ciascun algoritmo? (2)

- Per cui il primo algoritmo
  - Per **N=100** richiede complessivamente 1+2+100+200+1=**304** operazioni
  - Per **N=200** richiede complessivamente 1+2+200+400+1=**604** operazioni, cioè quasi il doppio delle operazioni richieste per N=100
- Si noti che **il numero delle operazioni che esegue il primo algoritmo è di fatto proporzionale al valore N letto in input**

Quante operazioni sono richieste da ciascun algoritmo? (3)

- Il secondo algoritmo richiede:
  - 1 operazione iniziale di lettura dell'input (istruzione 1)
  - 3 operazioni aritmetiche: una somma, una moltiplicazione e una divisione (istruzione 2)
  - 1 operazione finale di stampa del risultato (istruzione 3)
- Si noti che il numero delle operazioni che esegue il primo algoritmo è di fatto proporzionale al valore N letto in input

Quante operazioni sono richieste da ciascun algoritmo? (3)

- Il secondo algoritmo richiede:
  - 1 operazione iniziale di lettura dell'input (istruzione 1)
  - 3 operazioni aritmetiche: una somma, una moltiplicazione ed una divisione (istruzione 2)
  - 1 operazione finale di stampa del risultato (istruzione 3)
- Si noti che il numero delle operazioni che esegue il secondo algoritmo è indipendente dal valore N letto in input
- Anzi esso esegue sempre lo stesso numero di operazioni

Confronto tra algoritmi per lo stesso problema

- Diremo allora che:
  - Il **primo algoritmo ha un tempo di esecuzione proporzionale al numero di addendi** da sommare
  - Il **secondo algoritmo ha un tempo di esecuzione costante**
- Il primo algoritmo è quindi più costoso in termini di numero di operazioni rispetto al secondo
- Inoltre il primo algoritmo è tanto più costoso quanto più è grande il numero N in input
- Il secondo algoritmo ha un tempo di esecuzione che è indipendente dal valore di N
- quindi **la differenza tra i due algoritmi è tanto più a vantaggio del secondo quanto più grande è N**

Criteri di valutazione degli algoritmi

- Valutazione ‘quantitativa’ di un algoritmo
  - Dipendente dalla dimensione dei dati di input
    - Intuitivamente ci aspettiamo che un algoritmo che elabora 1.000 dati input sia più lento dello stesso algoritmo quando elabora 1.000 .000 di dati input
  - Indipendentemente dalla velocità del singolo computer sul quale l’algoritmo verrà implementato
    - Vogliamo sapere quanto è veloce una soluzione, non quanti millisecondi impiega sul mio computer anche perché questo valore cambia da computer a computer e dipende, sullo stesso computer, da altri fattori contingenti quale ad esempio il carico globale del sistema.
    - Vogliamo una valutazione parametrica della velocità dell’algoritmo, legata solo al numero di operazioni che esso effettua

Criteri di valutazione degli algoritmi (2)

- La teoria degli algoritmi mette effettivamente a disposizione un insieme di strumenti per poter valutare un algoritmo in base ai seguenti criteri valutativi:
  - Il **tempo di esecuzione** espresso in termini del numero di operazioni elementari che esso fa sui dati input, mettendosi nella ipotesi del caso peggiore (Worst Case Analysis)
  - Lo **spazio di memoria** occupato come il numero massimo di dati elementari che esso ha bisogno di vedere contemporaneamente durante la computazione, mettendosi nell’ipotesi del caso peggiore (Worst Case Analysis)

Criteri di valutazione degli algoritmi (3)

- Il numero di operazioni elementari dipende dall'algoritmo e non dalla velocità del singolo computer quindi ci si svincola dai valori tecnologici prestazionali del singolo computer
- Il tempo effettivo di esecuzione dell'algoritmo dipende poi dal tempo che il singolo computer impiega per eseguire tali operazioni elementari

Il caso peggiore

- La legge di Muphy "Se c'è una possibilità che qualcosa possa andare male, allora è certo che andrà male"
- Il caso migliore spesso non è significativo
- In situazioni pratiche, dobbiamo garantirci che la computazione termini comunque entro un tempo massimo, e questo è possibile valutarlo solo considerando la situazione più sfortunata (e quindi più lenta) possibile.
- *(es. per andare ad A a B con e senza traffico…)*

Il caso peggiore (2)

- Nelle prossime lezioni vedremo il caso si due algoritmi per la ricerca in un o schedario con 1.000.000.000 di schede
  - Caso migliore entrambi: 1 operazione
  - Caso peggiore del primo: 1.000.000.000 di operazioni
  - Caso peggiore: 30 operazioni

Criteri qualitativi

- Sono requisiti “qualitativi” di un algoritmo (molto meno importanti di quelli “quantitativi”)
  - Adattabilità dell’algoritmo a soluzioni diverse
  - Semplicità
  - Modularità
  - Eleganza