Complessità Computazionale

Analisi

- Algoritmi e pseudocodice
- Cosa significa analizzare un algoritmo
- Modello di calcolo
- Analisi del caso peggiore e del caso medio

Esempio di algoritmo in pseudocodice

```
INSERTION-SORT(A)
for j←2 to lenght[A]
    do    key←A[j]
    ▷si inserisce A[j] nella sequenza ordinata A[1..j-1]
          i ← j - 1
          while i>0 and A[i]>key
                do    A[i+1]←A[i]
                      i←i - 1
          A[i+1]←key
```

Esempio di algoritmo in C++

```
INSERTION-SORT(A)
int i,j;
for (j= 2; j<A.lenght;j++){
    //si inserisce A[j] nella sequenza ordinata A[1..j-1]
    int key=A[j]
    for (i=j-1; i>0 && key>A[j];i--)
          A[i+1]=A[i];
    A[i+1]=key;
}
```

Spiegazione intuitiva

- Supponiamo di avere i primi x elementi del vettore già ordinati
- Consideriamo l’elemento di posizione x+1 e chiamiamolo key
- L’idea è di scorrere gli elementi già ordinati e più grandi di key e di trovare la posizione giusta di key
- Mentre si scorrono gli elementi si fa spazio per l'eventuale inserzione spostando a destra di una posizione tutti gli elementi che sono maggiori di key
- Appena si trova un elemento più piccolo di key ci si ferma
- Dopo l'elemento piu' piccolo e prima di uno piu' grande si posiziona key

Cosa significa analizzare un algoritmo

- Analizzare un algoritmo significa determinare le risorse richieste per il completamento con successo dell'algoritmo stesso
- Le risorse di interesse possono essere quelle di:
  - memoria
  - tempo
  - numero di porte di comunicazione
  - numero di porte logiche
- Noi saremo interessati principalmente alla risorsa di *tempo computazionale*

Modello di calcolo

- Per poter indicare il tempo di calcolo è necessario specificare un modello di calcolo e di calcolatore
- Dovremo lavorare con un calcolatore astratto
- In questo modo possiamo confrontare fra di loro tempi di esecuzione di algoritmi senza il problema di avere macchine con architetture diverse:
  - singolo processore
  - piu' processori in parallelo
- ...e processori diversi:
  - con frequenze di clock diverse
  - con set di istruzioni diverse
  - con gerarchie di memoria diverse (cache di livello 1,2, RAM, memoria secondaria) e di diverse dimensioni

Il nostro modello di calcolo

- Noi faremo riferimento al modello di calcatore RAM (Random Access Machine) costituito da un mono-processore con accesso casuale ad un unico tipo di memoria di dimensioni illimitate
- In questo modello ogni istruzione è eseguita in successione (ovvero senza concorrenza)
- Ogni istruzione viene eseguita in tempo costante (anche se in generale diverso da istruzione a istruzione)
- Una cella di memoria puo' contenere un dato numerico di qualsiasi valore
- E' possibile usare il concetto di indirizzamento indiretto (puntatori)

Dimensione dell’input

- Per poter comparare l’efficienza di due algoritmi in modo generale si definisce una nozione di dimensione dell’input e si compara il tempo di calcolo dei due algoritmi in relazione ad esso

- Ad esempio per un algoritmo di ordinamento è ragionevole aspettarsi che al crescere del numero di dati da ordinare cresca il tempo necessario per completare l’algoritmo
- In questo caso la dimensione dell’input coincide con la numerosità dei dati in ingresso

Dimensione dell’input

- **Nota**: non sempre la dimensione dell’input coincide con il numero di elementi in ingresso
- **Ex**: un algoritmo di moltiplicazione fra due numeri naturali ha come dimensione il numero di bit necessari per rappresentare la codifica binaria dei numeri

- **Nota**: non sempre la dimensione dell’input è rappresentabile con una sola quantità
- **Ex**: un algoritmo che opera su grafi ha come dimensione il numero di nodi e di archi del grafo

Analisi del tempo computazionale

- Lo scopo dell'analisi del tempo computazionale è di dare una descrizione sintetica del tempo di calcolo dell'algoritmo al variare della dimensione dell'ingresso
- Inizieremo con un calcolo esatto del tempo
- Successivamente utilizzeremo un formalismo più sintetico e compatto che fa uso della notazione asintotica (ordini di grandezza)

Analisi dell'Insertion Sort

Sia n ← length[A]

N°	Costo	INSERTION-SORT(A)
n	c1	`for j ← 2 to lenght[A]`
n-1	c2	`do key ← A[j]`
n-1	0	`▸ si inserisce A[j] ...`
n-1	c4	`i ← j - 1`
$\Sigma_{j=2..n}\ t_j$	c5	`while i>0 e A[i]>key`
$\Sigma_{j=2..n}\ (t_j-1)$	c6	`do A[i+1]← A[i]`
$\Sigma_{j=2..n}\ (t_j-1)$	c7	`i ← i - 1`
n-1	c8	`A[i+1] ← key`

Dove t_j è il numero di volte che l'istruzione while è eseguita per un dato valore di j

Il tempo complessivo è dato da:

$$T(n)=c1.n + c2.(n-1)+c4.(n-1)+c5.(\Sigma_{j=2..n}\ t_j)+c6.(\Sigma_{j=2..n}\ (t_j-1)) +c7.(\Sigma_{j=2..n}\ (t_j-1))+c8.(n-1)$$

Caso migliore/peggiore

- Anche a parità di numerosità dei dati in ingresso il tempo di esecuzione può dipendere da qualche caratteristica complessiva sui dati, ad esempio da come sono ordinati inizialmente
- Si distinguono pertanto i casi migliore e peggiore a seconda che i dati abbiano (a parità di numerosità) le caratteristiche che rendono minimo o massimo il tempo di calcolo del dato algoritmo
- Nell’esempio dell’insertion sort
  - il caso migliore è che i dati siano già ordinati
  - il caso peggiore è che siano ordinati in senso inverso

Analisi del caso migliore

- Per ogni $j=2,3,\ldots,n$ in 5) si ha che $A[i]<key$ quando i ha il suo valore iniziale di $j-1$
- Quindi vale $t_j=1$ per ogni $j=2,3,\ldots,n$
- Il tempo di esecuzione diviene quindi:

  $T(n)=c1.n+c2(n-1)+c4.(n-1)+c5.(n-1)+c8.(n-1)$

  ovvero

  $T(n)=(c1+c2+c4+c5+c8).n -(c2+c4+c5+c8)$

  ovvero

  $T(n)=a.n+b$
- Diciamo che $T(n)$ è una funzione lineare di n

Analisi del caso peggiore

- Se l'array è ordinato in ordine decrescente allora si deve confrontare l'elemento key=A[j] con tutti gli elementi precedenti A[j-1], A[j-2],…,A[1]
- In questo caso si ha che t_j=j per j=2,3,4,…,n
- Si ha che:

  $\Sigma_{j=2..n}\ j = n(n+1)/2 -1$

  $\Sigma_{j=2..n}\ (j-1) = n(n-1)/2$

- Il tempo di esecuzione diviene quindi:

  T(n)=c1.n+c2(n-1)+c4.(n-1) +c5.(n(n+1)/2 -1) +c6.(n(n-1)/2) +c7.(n(n-1)/2) +c8.(n-1)

  T(n)=(c5/2+c6/2+c7/2).n^2+(c1+c2+c4+c5/2-c672-c7/2+c8).n-(c2+c4+c5+c8)

  T(n)=a.n^2+b.n+c

- Diciamo che T(n) è una funzione quadratica di n

Analisi del caso medio

- Se si assume che tutte le sequenze di una data numerosità siano equiprobabili allora mediamente per ogni elemento key=A[j] vi saranno metà elementi nei restanti A[1,..,j-1] che sono più piccoli e metà che sono più grandi
- Di conseguenza in media $t_j=j/2$ per $j=2,3,4,\dots,n$
- Si computa T(n) come nel caso peggiore
- Il tempo di calcolo risulta di nuovo quadratico in n a meno di costanti moltiplicative

Quale caso analizzare?

- Come è accaduto anche nel caso appena visto, spesso il *caso medio* è dello stesso ordine di grandezza del caso peggiore
- Inoltre la conoscenza delle prestazioni nel caso peggiore fornisce una limitazione superiore al tempo di calcolo, cioè siamo sicuri che mai per alcuna configurazione dell'ingresso l'algoritmo impiegherà più tempo
- Infine per alcune operazioni il caso peggiore si verifica abbastanza frequentemente (ad esempio il caso di ricerca con insuccesso)
- Pertanto si analizzerà spesso solo il caso peggiore

Ordine di grandezza

- Per facilitare l'analisi abbiamo fatto alcune astrazioni
- Si sono utilizzate delle costanti c_i per rappresentare i costi ignoti delle istruzioni
- Si è osservato che questi costi forniscono più dettagli del necessario, infatti abbiamo ricavato che il tempo di calcolo è nel caso peggiore $T(n)=a.n^2+b.n+c$ ignorando così anche i costi astratti c_i
- Si può fare una ulteriore astrazione considerando solo l'ordine di grandezza del tempo di esecuzione perché per input di grandi dimensioni è solo il termine principale che conta e dire che $T(n)=\Theta(n)$

Un algoritmo è tecnologia

- Si consideri il seguente caso:
  - si abbia un personal computer capace di eseguire 10^6 operazioni al secondo ed un supercomputer 100 volte più veloce
  - si abbia un codice di insertion sort che una volta ottimizzato sia in grado di ordinare un vettore di n numeri con $2n^2$ operazioni
  - si abbia un altro algoritmo (mergesort) in grado di fare la stessa cosa con 50 n log n operazioni
  - si esegua l'insertion sort su un milione di numeri sul supercomputer e il mergsort sul personal computer
- il risultato è che il supercomputer impiega:

  $2(10^6)^2/10^8 = 5.56$ ore
- mentre il personal computer impiega:

  $50\ 10^6 \log 10^6 / 10^6 = 16.67$ minuti

Efficienza asintotica

- L'ordine di grandezza del tempo di esecuzione di un algoritmo caratterizza in modo sintetico l'efficienza di un algoritmo e consente di confrontare fra loro algoritmi diversi per la soluzione del medesimo problema
- Quando si considerano input sufficientemente grandi si sta studiando *l'efficienza asintotica* dell'algoritmo
- Ciò che interessa è la crescita del tempo di esecuzione al tendere all'infinito della dimensione dell'input
- In genere un algoritmo asintoticamente migliore di un altro lo è in tutti i casi (a parte input molto piccoli)