Aniello Murano

Misurazione della Complessità e introduzione alla classe P

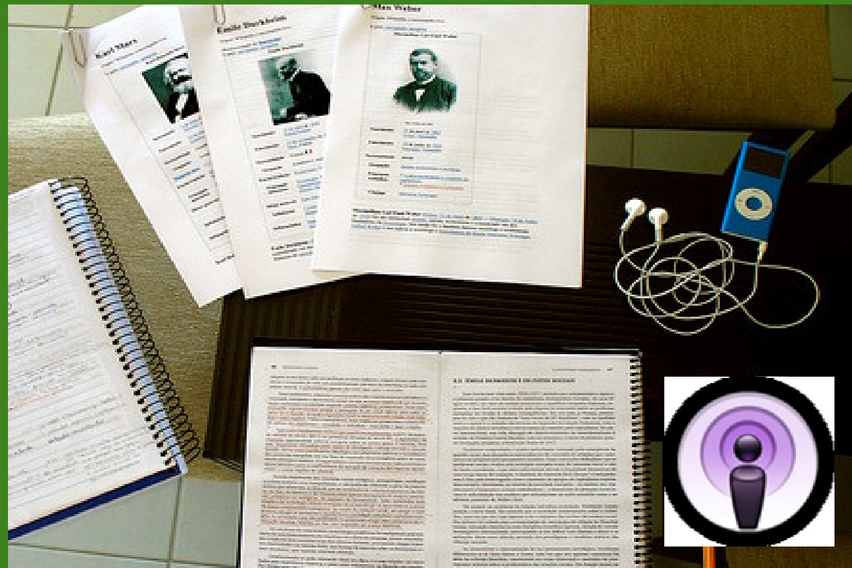
Lezione n.12
Parole chiave:
Complessità di Tempo

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Complessità di tempo

- DEFINIZIONE:

Sia **M** una MdT deterministica che si ferma su ogni possibile input.

La ***complessità di tempo*** (o ***tempo di esecuzione***) di **M** è la funzione **f : N → N**, dove **f(n)** è il numero massimo di passi che M compie su un input di lunghezza **n**.

Se **f(n)** è la complessità di tempo di **M**, allora si dice che **M** lavora in tempo **f(n)**, o che **M** è una macchina di tempo ***f(n)***.

Tempo di Esecuzione di un algoritmo

- Il tempo di esecuzione di un algoritmo è spesso dato una espressione, talvolta complessa, che dipende dalla particolare implementazione realizzata e dal particolare modello usato, la cui diversa scelta si concretizza nell'avere fattori moltiplicativi costanti diversi nel calcolo della complessità.
- Per questo (ed altri motivi), solitamente si preferisce stimare la complessità di tempo per valori molto grandi, trascurando i fattori costanti. Questa stima della complessità è chiamata ***analisi asintotica***.
- Se il tempo di esecuzione di un algoritmo è $f(n) = 3n^3+4n^2$ (dove n è la lunghezza dell'input), per valori di n molto grandi, il primo termine domina sul secondo. Inoltre, dovendo tralasciare i valori costanti, possiamo affermare che questo algoritmo cresce nell'ordine di n^3 e indichiamo questo con la notazione del grande ***O***, scrivendo $f(n) = O(n^3)$.
- La notazione asintotica è stata ampiamente studiata in corsi propedeutici a questo corso e non verrà trattata qui in dettaglio.
- Quello che è utile ricordare è che un algoritmo $O(n^c)$ è detto ***polinomiale,*** e un algoritmo $2^{O(n)}$ è detto esponenziale.

ANALISI DI ALGORITMI

- Analizziamo asintoticamente il seguente algoritmo M per decidere il linguaggio $A=\{0^k1^k|k\geq 0\}$.
- Su una stringa w in input, M si comporta come segue:

1. Scandisce il contenuto del nastro e rifiuta se trova uno 0 a destra di un 1;
2. Ripete se entrambi gli 0 e gli 1 rimangono sul nastro:
3. Scandisce il contenuto del nastro, cancellando un singolo 0 e un singolo 1.
4. Se gli 0 rimangono dopo tutti gli 1 che sono stati cancellati, o se gli 1 rimangono dopo che sono stati cancellati tutti gli 0 allora rifiuta. Altrimenti se sul nastro non rimangono né 1 e né 0 allora M accetta.

ANALISI DI ALGORITMI(2)

- Per analizzare M consideriamo ogni passo separatamente.
- Al passo 1:
  - la macchina scandisce il nastro per verificare che l'input è nella forma 0*1*. Questa elaborazione viene fatta in n passi, dove n rappresenta la lunghezza dell'input.
  - Per far tornare la testina all'inizio del nastro, la macchina esegue altri n passi. Cosi i passi totali in questo primo step sono *2n.*
  - Nella notazione O-grande diciamo che questo stage usa O(n) passi. Notiamo che nella descrizione della macchina non è menzionato il riposizionamento della testina. Usando una notazione asintotica, omettiamo alcuni dettagli nella descrizione della macchina che hanno l'effetto di al più un fattore costante nell'elaborazione.
- Nei passi 2 e 3:
  - la macchina ripete la scansione del nastro e cancella uno 0 e un 1 ad ogni scansione.
  - Ogni scansione utilizza O(n) passi. Poiché ogni scansione cancella due simboli, il numero di scansioni è al massimo n/2. Cosi il tempo totale dei passi 2 e 3 è $(n/2)O(n)=O(n^2)$.
- Al passo 4:
  - la macchina esegue una singola scansione per decidere se accettare o rifiutare. Il tempo di questo passo è al più O(n).
- Dunque, il tempo totale di M1 su input di lunghezza n è $O(n)+O(n^2)+O(n)$. Cioè, $O(n^2)$.
- Utilizziamo la stessa notazione per definire la classe di linguaggi per misurare il tempo.

Definizione di TIME(f(n))

- DEFINIZIONE:

  Per una funzione f: $N \rightarrow R^+$, si definisce classe di complessità di tempo, **TIME(f(n))**, l'insieme di tutti i linguaggi che sono decisi in tempo O(f(n)) da una Macchina di Turing.

- Quindi rifacendoci alla definizione di notazione che abbiamo definito nell'esempio precedente possiamo dire che il linguaggio A $\in$ TIME(n^2) poichè M1 decide A in tempo $O(n^2)$ e TIME(n^2) contiene tutti i linguaggi che possono essere decisi in tempo $O(n^2)$.

- Esiste una macchina che asintoticamente decide A più velocemente?

- Possiamo migliorare il tempo di elaborazione cancellando due 0 e due 1 ad ogni scansione in modo da ridurre il numero di scansioni di metà. Ma migliorare solo il tempo di elaborazione con un fattore di 2 non ha effetto asintoticamente sul tempo di elaborazione. La seguente macchina M_2, usa un metodo differente per decidere A che risulta asintoticamente migliore. Mostriamo che A $\in$ TIME(nlogn).

FUNZIONAMENTO DI M_2

- M_2=" Su input w stringa:
  1. Scandisce il contenuto del nastro e rifiuta se trova uno 0 a destra di un 1;
  2. Ripete fin quando sono presenti sul nastro degli 0 o degli 1;
  3. Scandisce il contenuto del nastro, verificando se il numero totale di 0 è pari o dispari. Se è dispari, rifiuta;
  4. Scandisce nuovamente il contenuto del nastro cancellando tutti gli altri 0 a partire dal primo 0, e in seguito cancellando tutti gli altri 1 partendo dal primo 1.
  5. Se non sono presenti ne 0 ne 1 sul nastro, accetta. Altrimenti rifiuta."

FUNZIONAMENTO DI M_2(2)

- Prima di analizzare il funzionamento di M_2, verifichiamo se A decide il linguaggio.
- Ad ogni scansione effettuata al passo 4, il numero totale di 0 rimanenti è ridotto di metà e qualsiasi altra cosa rimasta viene scartata. In questo modo se partiamo con tredici 0, dopo la singola esecuzione del passo 4 rimangono solo sei 0. Questo stage ha lo stesso effetto sul numero di 1.
- Ora esaminiamo la parità pari/dispari del numero di 0 e il numero di 1 ad ogni esecuzione dello stage 3. Supponiamo di partire sempre con tredici 0 e tredici 1. La prima esecuzione dello stage 3 trova un numero dispari di 0 e un numero dispari di 1. Dalla sottosequenza di esecuzioni, otteniamo un numero pari (6), poi un numero dispari (3), e in seguito un numero dispari (1). Non eseguiamo questo stage su 0 o su 1 perché la condizione di ripetizione del loop è specificata allo stage 2.
- Per la sequenza di parità trovata (dispari, pari, dispari, dispari) se sostituiamo le parità con gli 0 e le disparità con gli 1 e quindi invertiamo la sequenza, otteniamo 1101, che è la rappresentazione binaria di 13, o del numero di 0 e di 1 che avevamo inizialmente. La sequenza di parità prende sempre il contrario della rappresentazione binaria.

FUNZIONAMENTO DI M_2(3)

- Quando lo stage 3 effettua la verifica per determinare che il numero totale di 0 e 1 rimanenti è pari, verifica la corrispondenza tra la parità di 0 e la parità di 1. Se tutte le parità corrispondono, la rappresentazione binaria del numero di 0 e di 1 corrisponde, quindi i due numeri sono uguali.
- Per analizzare il tempo di elaborazione di M_2, prima osserviamo che ogni stage è eseguito in tempo O(n). In seguito determiniamo il numero di volte che viene fatta un esecuzione. Lo stage 1 e 5 sono eseguiti una solo volta, quindi con un tempo totale di O(n). Lo stage 4, saltando la metà degli 0 e degli 1 esegue al massimo $1+\log_2 n$ iterazioni di ripetizione del loop. Quindi il tempo totale dello stage 2,3,e 4 è $(1+\log_2 n)O(n)$ o anche O(nlogn).
- In conclusione il tempo di elaborazione di M_2 è:

$$O(n)+O(n\log n)=O(n\log n)$$

Complessità di tempo di MdT ad un solo nastro

- **Teorema**
  Sia M una MdT multinastro deterministica con tempo di esecuzione **O(t(n))**. La MdT a singolo nastro deterministica corrispondente ha un tempo di esecuzione **$O(t^2(n))$**.

- La prova si basa sulle costruzioni viste nelle lezioni precedenti. I dettagli sono lasciati per esercizio agli studenti.

Riassumiamo: Running time (1)

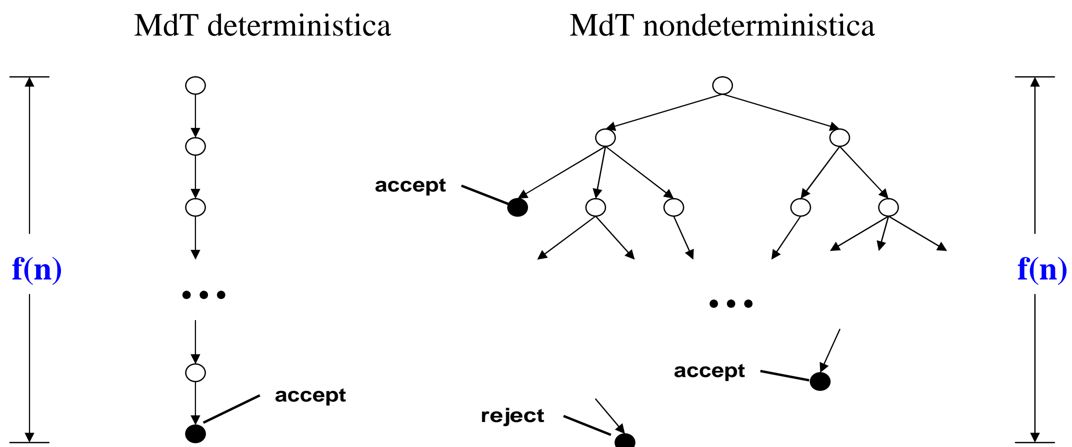
- Il "running time" o la complessità di tempo di una MdT M deterministica (che si ferma sempre) è la funzione f: N → N, dove f(n) è il massimo numero di operazioni che M esegue su ogni input di lunghezza n.
- Se f(n) è il running time di M, allora si dice che M lavora in tempo f(n)
- L'analisi asintotica di un algoritmo permette di valutarne la sua effettiva complessità pur tralasciando costanti e termini di basso ordine.
- Sia t: N → N una funzione. La classe di complessità di tempo TIME(t(n)) è la collezione dei linguaggi decidibili in tempo O(t(n)) da una MdT deterministica.
- Una complessità n^c per un algoritmo (con c>0) è chiamata **polinomiale**. In particolare con c=1 tale complessità è chiamata ***lineare.***
- Una complessità della forma $2^{O(n^c)}$ è chiamata **esponenziale**

Riassumiamo: Running time (2)

- Ritorniamo sull'esempio del linguaggio $L=\{0^n 1^n \mid n>0\}$
- Abbiamo visto che una MdT M può accettare L scandendo avanti e indietro il nastro, accorciando l'input di una unità ad ogni inversione. In questo caso, le operazioni da compiere sono n + (n-1) + (n-2) + ... Dunque M ha tempo $O(n^2)$ e si può concludere che L è in $TIME(n^2)$.
- La seguente macchina di Turing M_1 impiega invece tempo O(n log n)
  - Dapprima verifica che la stringa è del tipo 0*1*
  - Poi elimina gli 0 e gli 1 in posizione alterna.
  - Ad ogni scorrimento del nastro si verifica poi che il numero di 0 e di 1 e sempre pari.
- Si noti che ad ogni scorrimento del nastro, M_1 dimezza il numero di 0 e di 1.
- Dunque, L è in TIME(n log n) e questo migliora l'upper bound precedente
- É possibile mostrare che questo risultato non può essere ulteriormente migliorato su una macchina di Turing deterministica a nastro singolo.
- Si noti che una MdT M a due nastri può accettare L in tempo lineare
- Si noti anche che nella teoria della calcolabilità, MdT con numero differente di nastri e MdT deterministiche e non deterministiche sono tutte equivalenti. Nella teoria della complessità questo non è vero.
- Difatti, le MdT deterministiche possono richiedono tempo esponenziale rispetto a quelle nondeterministiche e la simulazione può richiedere tempo O(n log n).

Running Time di una MdT nondeterministica

- Sia N una MdT nondeterministica che è un decisore. La complessità di tempo di N è una fuzione f: N->N dove f(n) è il massimo numero di passi che N usa su ogni ramo della sua computazione su input di lunghezza n come mostrato nella seguente figura:



MdT deterministiche vs MdT nondeterministiche

- **Teorema**
  Sia M una MdT nondeterministica con tempo di esecuzione $O(t(n))$. La MdT deterministica corrispondente ha un tempo di esecuzione $2^{O(t(n))}$.

- La prova si basa sulle costruzioni viste nelle lezioni precedenti. I dettagli sono lasciati per esercizio agli studenti.

La classe P

- **P** è la classe dei linguaggi che sono decisi in tempo polinomiale da una **macchina di Turing deterministica a singolo nastro**. In altre parole

$$\mathbf{P} = \bigcup_k \mathrm{TIME}(n^k).$$

- Osservazioni su **P**:
  - È importante notare che **P** è invariante per modelli di computazione che sono *polinomialmente equivalenti* a macchine di turing deterministiche a singolo nastro.
  - Per "polinomialmente equivalenti" intendiamo una equivalenza con al più una differenza polinomiale in running time.
  - Dunque, P è una classe matematica molto robusta, non influenzata dal particolare modello deterministico utilizzato.
  - Si può ben affermare che P corrisponde alla classe dei problemi che sono realisticamente risolvibili da un computer. Questo fa di P una classe molto importante da un punto di vista pratico.

Osservazioni su P

- Quando ci interessiamo della polinomialità, gli algoritmi possono essere descritti ad alto livello senza riferirsi a particolari caratteristiche del modello utilizzato. In questo modo possiamo evitare dettagli tediosi sul natro o sul movimento della testina.
- Possiamo descrivere gli algoritmi con stage numerate, dove in genere uno stage è analogo a un passo della macchina di Turing anche se implementare uno stage in genere richiede molti passi della TM.
- Usando l'analisi asintotica possiamo trascurare queste differenze.
- Per vedere che un algoritmo può essere eseguito in tempo polinomiale è sufficiente mostrare che:
  1. Esiste un limite superiore polinomiale (upper bound), usando generalmente la O-notazione, sul numero di stage che l'algoritmo usa mentre gira su un input di lunghezza n, e
  2. Ogni stage utilizza un numero polinomiale di passi su un ragionevole modello deterministico (per esempio la macchina di Turing)
- La codifica degli oggetti deve essere "ragionevole", per esempio codificare 12 con 111111111111 non è ragionevole perché questa codifica è esponenziale rispetto alla codifica binaria.
- Tra le codifiche ragionevoli per i grafi c'è quella che usa la matrice di adiacenza. Visto che la matrice di adiacenza diferisce in modo polinomiale dal numero di nodi($O(N^2)$) si può vedere se l'algoritmo è polinomiale rispetto al numero dei nodi piuttosto che rispetto alla matrice.

Esempi di Problemi in P

- Il problema di verificare se un grafo ammette un percorso appartiene alla classe **P**. [Esercizio per gli studenti]
- Un altro problema in **P** è quello dei primi relativi. Due numeri sono primi relativi se il loro massimo comun divisore è 1.
- Un algoritmo brute force per il problema dei primi relativi fornisce una complessità esponenziale (si controlla ogni singolo numero da 2 al minimo dei due numeri per vedere che non divide entrambi)
- L'algoritmo di Euclide invece richiede tempo polinomiale

L'algoritmo di Euclide

- Sia RELPRIME{<x,y> : x e y sono primi relativi}
- L'algoritmo di Euclide può essere calcolato dalla seguente macchina E: *su input <x, y>, con x e y numeri naturali in binario E si comporta come segue:*

  1. ripeti finche y == 0:
  2. Assegna a x = x mod y.
  3. scambia x e y.
  4. restituisci x."

- *Mostriamo adesso una macchina R in grado di risolvere RELPRIME in tempo polinomiale, usando E come subroutine.*

  R = "su input <x, y>, con x e y naturali in binario:

  1. Se necessario, scambia x e y in modo che risulti x>y.
  2. esegui E su (x, y).
  2. Se il risultato è 1, accetta. Altrimenti rifiuta."

This document was created with Win2PDF available at http://www.win2pdf.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.