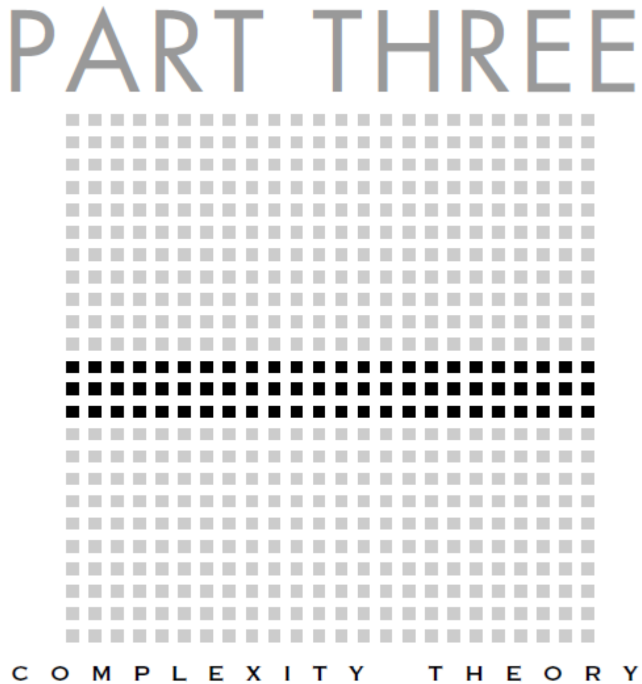

TEORIA DELLA COMPLESSITA’

12 maggio 2022

Contenuto del corso

Il corso è un'introduzione alle tre aree centrali della teoria della computazione:

- Teoria degli Automi (Linguaggi formali e modelli di calcolo)
- Teoria della Calcolabilità /Computabilità
- Teoria della Complessità

Le tre aree sono legate dalla domanda:

Quali sono le capacità e i limiti dei computer?

Contenuto del corso

- MODELLI DI COMPUTAZIONE:

  AUTOMI FINITI DETERMINISTICI E NON DETERMINISTICI.
  ESPRESSIONI REGOLARI. PROPRIETÀ DI CHIUSURA DEI LINGUAGGI REGOLARI. TEOREMA DI KLEENE. PUMPING LEMMA PER I LINGUAGGI REGOLARI.

  MACCHINA DI TURING DETERMINISTICA A NASTRO SINGOLO. IL LINGUAGGIO RICONOSCIUTO DA UNA MACCHINA DI TURING. VARIANTI DI MACCHINE DI TURING E LORO EQUIVALENZA.

- IL CONCETTO DI COMPUTABILITÀ: FUNZIONI CALCOLABILI, LINGUAGGI DECIDIBILI E LINGUAGGI TURING RICONOSCIBILI. LINGUAGGI DECIDIBILI E LINGUAGGI INDECIDIBILI. IL PROBLEMA DELLA FERMATA. RIDUZIONI. TEOREMA DI RICE.

- IL CONCETTO DI COMPLESSITÀ: MISURE DI COMPLESSITÀ: COMPLESSITÀ IN TEMPO DETERMINISTICO E NON DETERMINISTICO. RELAZIONI DI COMPLESSITÀ TRA VARIANTI DI MACCHINE DI TURING. LA CLASSE P. LA CLASSE NP. RIDUCIBILITÀ IN TEMPO POLINOMIALE. DEFINIZIONE DI NP-COMPLETEZZA. RIDUZIONI POLINOMIALI. ESEMPI DI LINGUAGGI NP-COMPLETI.

- Abbiamo introdotto, come modello di calcolo, la Macchina di Turing e alcune sue varianti.
- Prima di introdurre la Macchina di Turing, è stato introdotto un modello più semplice, l'automa a stati finiti.
- E' stato fornito un metodo di prova, la riducibilità mediante funzione.
- E' stata dimostrata l'indecidibilità di alcuni problemi, tra cui quello della fermata per le macchine di Turing.
- Per alcuni di essi, è stata dimostrata l'indecidibilità utilizzando il metodo della riducibilità mediante funzione.

Vi sono moltissimi esempi di problemi che sono risolvibili mediante algoritmi.

Per esempio: ordinare n numeri, raggiungibilità nei grafi, MST e altri studiati in PA

Però, alcuni problemi risolvibili sono (più) ‘’facili" e altri (più) ‘’difficili", a seconda che gli algoritmi finora noti per risolverli siano utili o meno in pratica.

Nel primo caso si parla di algoritmi efficienti, nel secondo caso di algoritmi inefficienti.

Ad esempio, ordinare n numeri è facile: $O(n \lg n)$

Calcolabilità e complessità

Calcolabilità: si occupa di problemi risolvibili algoritmicamente in linea di principio.

Domande che affronta:

Quali problemi sono risolvibili?

Cosa significa procedura effettiva di calcolo?

Complessità: si occupa di problemi risolvibili algoritmicamente in pratica.

La teoria della Complessità analizza problemi risolvibili.

Domande che affronta:

Quali sono le risorse minime necessarie (es. tempo di calcolo e memoria) per la risoluzione di un problema?

Come si misura il consumo delle risorse?

Problemi trattabili

Problema trattabile in pratica se può essere risolto da un algoritmo in tempo polinomiale.

Possibili risposte alla domanda:

Questo tale problema è trattabile?

1. Sì, ecco un algoritmo efficiente che lo risolve!
2. No, ed è anche possibile dimostrarlo
3. Nessuna delle due!

Problemi trattabili e complessità

Sono considerati efficienti gli algoritmi di complessità polinomiale.

La complessità esponenziale invece è spesso proibitiva, per dimensioni di istanze anche ragionevolmente piccole, tanto da rendere intrattabili i problemi di complessità esponenziale, come se l'algoritmo che li risolve non ci fosse.

La teoria della complessità cerca di capire le ragioni di questa complessità.

Raggrupperemo in una classe tutti i problemi che possono essere risolti con algoritmi della stessa complessità (di tempo o di spazio) per poi studiarli.

Classi di complessità

- Studieremo le classi di complessità più note.
- La classe P, che corrisponde alla classe dei problemi risolubili con un algoritmo di complessità di tempo polinomiale e la classe NP.
- Introdurremo il concetto di riduzione polinomiale tra linguaggi/problemi come strumento per dimostrare l'appartenenza o meno a una classe di complessità.
- Esporremo uno dei più grandi problemi aperti dell'informatica teorica: il limite fra problemi trattabili e intrattabili non è chiaro

$$P = NP?$$

Tempo di esecuzione di un algoritmo

Durante il corso di Progettazione di Algoritmi abbiamo detto che:

- Il tempo di esecuzione di un algoritmo è una funzione T(n) della taglia n dell'input
- T(n) lo calcoliamo (è proporzionale a) come il numero di operazioni elementari, cioè operazioni che impiegano un certo tempo costante, che non dipende da n
- Abbiamo usato analisi asintotica
- Interessati principalmente al caso peggiore.

Adesso possiamo essere più precisi:

- algoritmo → Decisore
- taglia dell'input → lunghezza di una codifica dell'istanza
- operazione elementare → passo di computazione

Complessità di tempo di un decisore

Definizione

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *una MdT deterministica, a nastro singolo, che si arresta su ogni input.*

La **complessità di tempo** *di* M *è la funzione* $f : \mathbb{N} \to \mathbb{N}$

dove $f(n)$ *è il massimo numero di passi di computazione eseguiti da* M *su un input di lunghezza* n, $n \in \mathbb{N}$.

Se M ha complessità di tempo $f(n)$, diremo che

M decide $L(M)$ in tempo (deterministico) $f(n)$

Passo di computazione

Siano C_1, C_2 due configurazioni di una MdT M.

Se C_1 produce C_2, scriveremo

$$C_1 \rightarrow C_2$$

La trasformazione $\rightarrow$ di C_1 in C_2 prende il nome di **passo di computazione**.

Corrisponde a un'applicazione della funzione di transizione di M.

Numero di passi

Sia M un decisore.

Una computazione di M su w è una sequenza di configurazioni $C_1, C_2, \ldots, C_k$ di M, tali che:

1) $C_1 = q_0w$ è la configurazione iniziale di M con input w
2) $C_i \longrightarrow C_{i+1}$ per ogni i in $\{1, \ldots, k-1\}$
3) C_k è una configurazione di arresto

Diremo che il numero di passi di questa computazione di *M* su *w* è la lunghezza k-1 della computazione.

Quindi, se f è la complessità di tempo di M, $f(n) =$ massimo numero di passi in $q_0 w \rightarrow^* uqv$, $q \in \{q_{accept}, q_{reject}\}$, al variare di w in Σ^n.

Alcune astrazioni

- identifichiamo gli input che hanno la stessa lunghezza,
- valuteremo la complessità nel **caso peggiore** (cioè relativo alla stringa di input di lunghezza n che richiede il maggior numero di passi),
- **non** valuteremo esattamente la complessità di tempo $f(n)$ di M ma piuttosto stabiliremo un limite asintotico superiore per $f(n)$, usando la notazione O-grande **(analisi asintotica)**.
- Quindi, dire che M ha complessità di tempo $O(g(n))$ vuol dire che M ha complessità di tempo $f(n)$ ed $f(n)$ è $O(g(n))$.

Definizione

Siano f e g due funzioni
$f : \mathbb{N} \to \mathbb{R}^+$, $g : \mathbb{N} \to \mathbb{R}^+$.

Diremo che $f(n)$ *è* $O(g(n))$ *oppure* $f(n) = O(g(n))$ *se esistono una costante* $c > 0$ *e una costante* $n_0 \geq 0$ *tali che, per ogni* $n \geq n_0$,

$$f(n) \leq cg(n).$$

Diremo che $g(n)$ *è un limite superiore (asintotico) per* $f(n)$.

Analisi della complessità di tempo

L'analisi della complessità di tempo di una MdT, dovrà essere fatta a partire da una descrizione, spesso ad alto livello, della MdT, analogamente all'analisi che facciamo di un algoritmo descritto tramite pseudocodice.

Esempio. Qual è la complessità di tempo della macchina di Turing M che:

1. rifiuta se l'input è la parola vuota
2. cancella il primo carattere dell'input e accetta, se l'input è una stringa non vuota.

Risposta: la complessità di tempo della macchina di Turing M è $O(1)$ (tempo costante).

Complessità di tempo: un esempio

Esempio. $L = \{0^k 1^k \mid k \geq 0\}$

Abbiamo visto una MT M a nastro singolo che decide L.
Sull'input w, M:

1. Verifica che $w \in L(0^*1^*)$.
2. Partendo dallo 0 più a sinistra, sostituisce 0 con $\sqcup$ poi va a destra, ignorando 0 e 1, fino a incontrare $\sqcup$.
3. Verifica che immediatamente a sinistra di $\sqcup$ ci sia 1: se così non è, rifiuta w. Altrimenti, sostituisce 1 con $\sqcup$ e va a sinistra fino a incontrare $\sqcup$.
4. Guarda il simbolo a destra di $\sqcup$:
    - Se è un altro $\sqcup$, allora accetta w.
    - Se è 1, allora rifiuta w.
    - Se è 0 ripete a partire dal passo 2.

Analisi della complessità di tempo

Sia $|w| = n$, cioè utilizziamo n per rappresentare la lunghezza dell'input.

Per analizzare M, consideriamo ciascuna delle sue quattro fasi separatamente.

1. Verifica che $w \in L(0^*1^*)$.

   Nella prima fase la macchina scansiona il nastro per verificare che l'input è in $L(0^*1^*)$, cioè del tipo 0^t1^q, $t, q \in \mathbb{N}$. Tale operazione di scansione usa n passi, dove $n = |w|$. Per riposizionare la testina all'estremità sinistra del nastro utilizza ulteriori n passi. Per cui il totale di passi utilizzati in questa fase è $2n$ passi. Nella notazione O-grande diciamo che questa fase usa $O(n)$ passi.

Analisi della complessità di tempo

2. Partendo dallo 0 più a sinistra, sostituisce 0 con $\sqcup$ poi va a destra, ignorando 0 e 1, fino a incontrare $\sqcup$.
3. Verifica che immediatamente a sinistra di $\sqcup$ ci sia 1: se così non è, rifiuta w. Altrimenti, sostituisce 1 con $\sqcup$ e va a sinistra fino a incontrare $\sqcup$.

Le operazioni 2 e 3 richiedono ciascuna O(n) passi e sono eseguite al più n/2 volte.
Infatti, nel caso peggiore, M esegue ripetutamente la scansione del nastro cancellando due simboli, il primo 0 e l'ultimo 1, ad ogni scansione.

4. Guarda il simbolo a destra di $\sqcup$:
   - Se è un altro $\sqcup$, allora accetta w.
   - Se è 1, allora rifiuta w.
   - Se è 0 ripete a partire dal passo 2.

L'operazione 4 ha tempo O(1).

Quindi M decide L in tempo $T(n) = O(n) + n/2\ O(n) + O(1) = O(n^2)$.

Classi di complessità

Obiettivo:

Classificare i linguaggi in base alla complessità di tempo di un algoritmo che li decide.

Raggruppare in una stessa classe linguaggi i cui corrispondenti decider hanno complessità di tempo che sia un O-grande della stessa funzione.

Definizione (Classe di complessità di tempo deterministico)

Sia $f : \mathbb{N} \rightarrow \mathbb{R}^+$ una funzione, sia $\mathcal{M}$ l'insieme delle MT deterministiche, a nastro singolo e che si arrestano su ogni input.

La classe di complessità di tempo deterministico $TIME(f(n))$ è

$$TIME(f(n)) = \{L \mid \exists M \in \mathcal{M} \text{ che decide } L \text{ in tempo } O(f(n))\}$$

Classi di complessità

TIME(1): insieme dei linguaggi per i quali esiste un decider che li decide in tempo O(1) (tempo costante).

TIME(n): insieme dei linguaggi per i quali esiste un decider che li decide in tempo O(n) (tempo lineare).

TIME(n^k): insieme dei linguaggi per i quali esiste un decider che li decide in tempo O(n^k) (tempo polinomiale).

TIME(2^n): insieme dei linguaggi per i quali esiste un decider che li decide in tempo O(2^n) (tempo esponenziale).

Complessità di tempo: un esempio

Torniamo al linguaggio $L = \{0^k1^k \mid k \geq 0\}$.

L'analisi precedente mostra che
$L = \{0^k1^k \mid k \geq 0\} \in TIME(n^2)$.
Infatti abbiamo fornito una macchina di Turing M che decide L 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 decide L in modo asintoticamente più veloce?

Complessità di tempo: un esempio

M_2 = ''Sulla stringa input w:

1. Verifica che w sia della forma 0*1* scandendo il nastro e rifiutando se trova uno 0 a destra di un 1
2. Ripete le due operazioni seguenti finché il nastro contiene almeno uno 0 e un 1 :
    3. Verifica che il numero di simboli rimasti sul nastro sia pari; se è dispari rifiuta
    4. Scandisce nuovamente l'input, cancellando uno 0 sì e il successivo no, a partire dal primo 0, poi cancellando un 1 sì e il successivo no, a partire dal primo 1
5. Se nessuno 0 e nessun 1 resta sul nastro, accetta. Altrimenti rifiuta."

Complessità di tempo: un esempio

Analizziamo il tempo di esecuzione $T(n)$ di M_2, dove $n = |w|$.

Le fasi 1 e 5 vengono eseguite una volta, impiegando un tempo totale $O(n)$.

Le fasi 3 e 4 fanno parte di un ciclo, l'iterazione è evidenziata al passo 2.
Ogni fase del ciclo richiede tempo $O(n)$.
Determiniamo poi il numero di volte in cui ognuna viene eseguita.
La fase 4 scarta almeno metà dei simboli 0 e 1 ogni volta che viene eseguita, quindi si verificano al massimo $O(\log n)$ iterazioni del ciclo prima di averli cancellati tutti.

Il tempo totale delle fasi 2, 3 e 4 è $O(n \log n)$.

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

Complessità di tempo: un esempio

Esempio.
$L = \{0^k 1^k \mid k \geq 0\} \in TIME(n^2)$, $L \in TIME(n \log n)$.

Possiamo decidere L in tempo lineare?

Sì, ma avendo 2 nastri!

Complessità di tempo: un esempio

$M_3 =$ "Su input w:

1. Scorre il primo nastro verso destra e rifiuta se trova uno 0 a destra di un 1.
2. Scorre il primo nastro verso destra fino al primo 1: per ogni 0, scrive un 1 sul secondo nastro.
3. Scorre primo nastro verso destra leggendo i simboli 1 e scorre il secondo nastro verso sinistra. Per ogni 1 letto sui due nastri li cancella. Se i simboli letti non sono uguali, rifiuta.
4. Se legge $\sqcup$ su entrambi i nastri, accetta.

Sia n = |w|.
Ciascuna delle quattro fasi utilizza O(n) passi.
Quindi il tempo di esecuzione complessivo risulta O(n), cioè lineare.

Si noti che questo tempo di esecuzione è il migliore possibile perché sono necessari n passi solo per leggere un input di lunghezza n.

Modello di calcolo

La complessità di tempo dipende dal modello di calcolo?

Sì

Esempio: $L = \{0^k 1^k \mid k \geq 0\}.$

- Esiste una MdT (a singolo nastro) che decide L in tempo O(n log n)
- Esiste una MdT con due nastri che decide L in tempo O(n)
- Si può dimostrare che non esiste una MdT (a singolo nastro) che decide L in tempo O(n)

Computabilità e complessità

Teoria della computabilità

La tesi di Church-Turing implica che tutti i modelli computazionali sono equivalenti, ossia che tutti decidono la stessa classe di linguaggi.

Teoria della complessità

la scelta del modello influisce sulla complessità di tempo.

Con quale modello misureremo la complessità di tempo?

Quale modello di calcolo?

Con quale modello misureremo la complessità di tempo?

Useremo decider (macchine di Turing deterministiche che si fermano su ogni input) e varianti polinomialmente equivalenti, cioè che possono simularsi tra di loro con un sovraccarico computazionale polinomiale.

Decisori, MdT multinastro, MdT che possono lasciare ferma la testina sono polinomialmente equivalenti.

Macchina di Turing non deterministica non è polinomialmente equivalente ai decisori perché il sovraccarico computazionale non è polinomiale, ma esponenziale.

Sovraccarico computazionale

Teorema

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni macchina di Turing deterministica multinastro M con complessità di tempo $t(n)$ esiste una macchina di Turing deterministica a nastro singolo M' con complessità di tempo $O(t^2(n))$, equivalente a M.

Teorema

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni macchina di Turing a nastro singolo, non deterministica N avente tempo di esecuzione $t(n)$ esiste una macchina di Turing a nastro singolo, deterministica e di complessità di tempo $2^{O(t(n))}$, equivalente ad N.

Dovremo però definire tempo di calcolo per MdT non deterministica

Codifiche

La complessità di tempo dipende dalla codifica utilizzata? Sì

Per esempio, un intero x può essere rappresentato in unario con x cifre, mentre in binario con $\lfloor \log_2 x \rfloor + 1$ bit. Quindi:

- in unario $n = |\langle x \rangle|$ e $x = n$
- in binario $n = |\langle x \rangle| = \lfloor \log_2 x \rfloor + 1$ e $x = O(2^n)$.

Esempio: un algoritmo che decide PRIMO su input x è

1. Dividi x per tutti gli interi i, con $1 < i < x$
2. Se tutti i resti sono diversi da 0 accetta, altrimenti rifiuta.

L'algoritmo eseguirà $x - 2$ divisioni. Quindi:

- $O(n)$ se x è codificato in unario
- $O(2^n)$ se x è codificato in binario

Quale codifica?

Quali codifiche usare?

Occorre considerare codifiche ''ragionevoli": non ''prolisse"
cioè tali che non vi siano istanze la cui rappresentazione sia artificiosamente lunga.
In particolare, scartare la rappresentazione unaria degli interi positivi.

Codifiche ''ragionevoli" dei dati sono quelle polinomialmente correlate, cioè quelle che consentono di passare da una di esse a una qualunque altra codifica ''ragionevole" delle istanze dello stesso problema in un tempo polinomiale rispetto alla rappresentazione originale.

Useremo la codifica binaria e le altre polinomialmente correlate.

MdT non deterministiche

La macchina di Turing non deterministica non corrisponde a nessun meccanismo di computazione reale.
Ma la definizione di tempo di esecuzione di una macchina di Turing non deterministica è utile per caratterizzare un'importante classe di linguaggi.

Ricordiamo che una macchina di Turing non deterministica è un decisore se, per ogni stringa input w, tutte le computazioni a partire da q_0w terminano in una configurazione di arresto.

MdT non deterministiche: tempo di esecuzione

Definizione (Tempo di esecuzione di una MdT non deterministica)

Sia $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *una macchina di Turing non deterministica che sia un decisore (ovvero tutte le computazioni, per ogni input* w*, terminano in una configurazione di arresto).*

Il **tempo di esecuzione** *di* N *è la funzione* $f : \mathbb{N} \to \mathbb{N}$ *dove* $f(n)$ *è il massimo numero di passi eseguiti da* N **in ognuna delle computazioni** *su ogni input di lunghezza* n*,* $n \in \mathbb{N}$*.*

MdT deterministiche e non: tempo di esecuzione

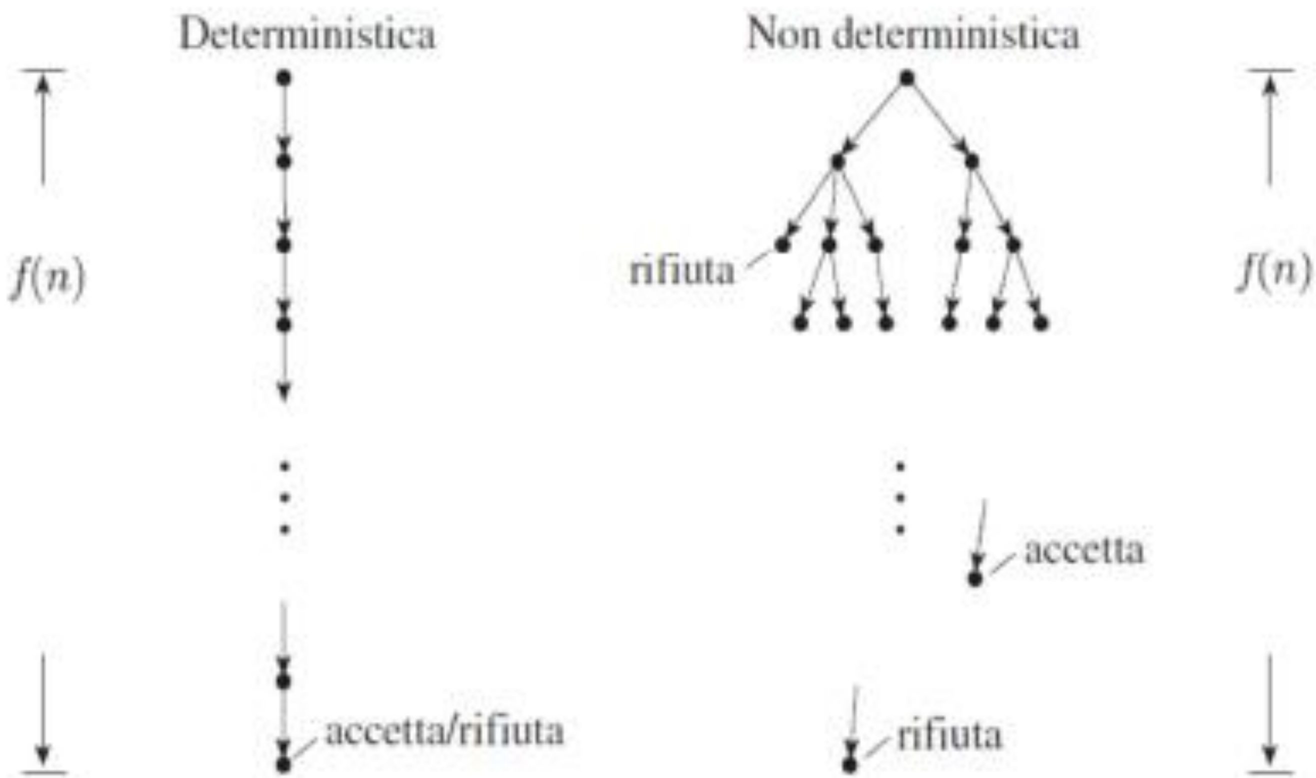

FIGURA 7.10
Misurazione del tempo nei casi deterministico e non deterministico

MdT non deterministiche: tempo di esecuzione

Il tempo di esecuzione di una MdT non deterministica N è la funzione $f : \mathbb{N} \to \mathbb{N}$ dove
$f(n) =$ massimo delle altezze degli alberi, ognuno dei quali rappresenta le possibili computazioni su input w, al variare di $w \in \Sigma^n$.

Teorema

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni macchina di Turing a nastro singolo, non deterministica N avente tempo di esecuzione $t(n)$ esiste una macchina di Turing a nastro singolo, deterministica e di complessità di tempo $2^{O(t(n))}$, equivalente ad N.

Tempo polinomiale

Differenze polinomiali nel tempo di esecuzione sono considerate piccole, mentre differenze esponenziali sono considerate grandi.
Perchè?

Esempio: Dato un elenco di n citta, stabilire se esiste un giro turistico che visiti ogni citta esattamente una sola volta.

L'algoritmo basato sulla ricerca esaustiva (algoritmo di forza bruta) richiede tempo esponenziale.
Nota: finora è l'unico algoritmo noto per risolvere il problema.
Se eseguo tale algoritmo su un computer che ha m volte la potenza del migliore calcolatore attuale (m = numero stimato degli elettroni nell'universo), il tempo richiesto per ottenere la soluzione per n = 1000 e maggiore di $10^{79+13+9+12}$.

Tempo polinomiale

La decisione di non tener conto delle differenze polinomiali non significa che tali differenze non siano considerate importanti.
Ma significa esaminare le soluzioni algoritmiche da una diversa prospettiva.

Tutti i modelli computazionali deterministici ‘’ragionevoli" sono polinomialmente equivalenti.

Cioè, uno di essi può simularne un altro con aumento solo polinomiale del tempo di esecuzione.

La classe P

Definizione

La classe P è l'insieme dei linguaggi L per i quali esiste una macchina di Turing deterministica M con un solo nastro che decide L in tempo $O(n^k)$ per qualche $k \geq 0$, cioè

$$P = \bigcup_{k \geq 0} TIME(n^k).$$

Esempio: $L = \{0^k 1^k \mid k \geq 0\} \in P$.

La classe P

La classe P gioca un ruolo centrale nella teoria della complessità ed e importante perché:

- P corrisponde, con una certa approssimazione, alla classe di problemi che sono realisticamente risolubili mediante programmi su computer reali.
- P è una classe «robusta» dal punto di vista matematico

- P è invariante per tutti i modelli di computazione che sono polinomialmente equivalenti alla macchina di Turing deterministica a nastro singolo.

- La classe P è invariante rispetto alla scelta di una codifica ‘’ragionevole" dell'input.

Teorema

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni macchina di Turing multinastro M con complessità di tempo $t(n)$ esiste una macchina di Turing a nastro singolo M' con complessità di tempo $O(t^2(n))$, equivalente a M.

Quindi, se L è deciso in tempo polinomiale su una macchina di Turing multinastro, allora L è deciso in tempo polinomiale su una macchina di Turing a nastro singolo.

La classe P

Inoltre: la composizione di (un numero finito) di polinomi è un polinomio.

$$N : w \rightarrow \boxed{M_1} \rightarrow \boxed{M_2}$$

- Il numero di passi di M_1 su input di lunghezza n è $O(n^k)$.
- Il numero di passi di M_2 sull'output di M_1, di lunghezza $O(n^k)$, è $O((n^k)^t) = O(n^{kt})$.
- Quindi il numero di passi di N su un input di lunghezza n è $O(n^k) + O(n^{kt})$ (polinomiale).

La lunghezza dell'output di M_1 non è necessariamente n, ma è certamente limitata dalla complessità in tempo di M_1, ovvero $O(n^k)$.

Esempi di linguaggi in P

$PATH =$

$\{\langle G, s, t \rangle \mid G$ è un grafo orientato in cui c'è un cammino da s a $t\}$

Teorema

$PATH \in P$.

E' un problema di raggiungibilità nei grafi.
Una visita BFS o DFS da s ha tempo lineare nella codifica di una rappresentazione dell'istanza.

Esempi di linguaggi in P

Due numeri interi positivi x, y sono relativamente primi (o coprimi) se il loro massimo comun divisore è 1.

RELPRIME = { $\langle$ x, y$\rangle$ | x e y sono interi positivi relativamente primi }

Teorema
RELPRIME $\in$ P.

L'algoritmo che permette di provare il teorema è basato sull'algoritmo di Euclide (intorno al 300 a.C.) per calcolare il massimo comune divisore MCD(x, y) di due numeri interi non negativi x, y.

Algoritmo di Euclide

```
MCD(a, b)
    If b = 0 then MCD = a
        else MCD = MCD(b, a (mod b))
```

Sono necessarie O(log b) chiamate ricorsive.
La complessità di tempo di MCD è logaritmica rispetto al valore dei due numeri.
Quindi, lineare rispetto alla loro codifica binaria.

Conclusioni

La complessità di tempo dipende:

- dal modello di calcolo e
- dalla codifica utilizzata

La classe P