COMPLESSITÀ DI TEMPO

Se un linguaggio associato a un problema di decisione è decidibile, è sempre possibile in teoria scrivere un programma che, data in input una istanza del problema, dia in output la risposta al problema. E in pratica?

Anche quando un problema è decidibile, e quindi in linea di principio computazionalmente risolvibile, può non essere risolvibile in pratica se la soluzione richiede una quantità eccessiva di tempo o di memoria.

Iniziamo dal tempo.

Come primo passo, introduciamo un metodo per misurare il tempo usato per risolvere un problema.

Poi mostreremo come classificare i problemi in base alla quantità di tempo che essi richiedono.

Infine, consideriamo la possibilità che determinati problemi decidibili richiedano enormi quantità di tempo e di come sia possibile determinare quando ci troviamo di fronte a un problema di questo tipo.

Ci riferiremo principalmente a problemi di decisione e quindi ai linguaggi associati.

Da questo momento in poi:

- solo linguaggi decidibili
- solo decider

Li studieremo dal punto di vista della "quantità di risorse di calcolo utilizzate" nella computazione.

Considereremo il **"tempo"** utilizzato nella computazione.

UNA MISURA DEL TEMPO

Complessità di tempo

Il numero di passi che utilizza un algoritmo su un particolare input può dipendere da diversi parametri.

Ad esempio, se l'input è un grafo, il numero di passi può dipendere dal numero di nodi, dal numero di archi e dal grado massimo del grafo, o da una combinazione di questi e/o altri parametri.

Il tempo di esecuzione di un algoritmo sarà calcolato in funzione della lunghezza della stringa che rappresenta l'input senza considerare eventuali altri parametri.

Ad esempio, se l'input è un grafo G, il tempo di esecuzione di un algoritmo sui grafi sarà calcolato in funzione della lunghezza della codifica $\langle G \rangle$ di G.

Considereremo l'analisi del **caso peggiore**, quindi valuteremo il tempo di esecuzione massimo tra tutti gli input di una determinata lunghezza.

(Non considereremo l'analisi del caso medio.)

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) $f(n)$

Se $C_1, C_2, \ldots, C_{k+1}$, $k \geq 1$, sono configurazioni di M, tali che

1. $C_1 = q_0 w$ è la configurazione iniziale di M con input w
2. $C_i \to C_{i+1}$ per ogni $i \in \{1, \ldots, k\}$
3. C_{k+1} è una configurazione di arresto,

il numero di passi eseguiti da M su w è k.

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

Una misura della complessità di tempo

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 $O(g(n))$.

$\mathbb{R}^+ =$ insieme dei numeri reali positivi.

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 $f(n)$.

Una misura della complessità di tempo

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).

Esempio. La domanda seguente è corretta?

"Qual è la complessità di tempo di una macchina di Turing M che copia il suo input?"

No, la domanda non è corretta perché possiamo avere diverse MdT che copiano il proprio input, con diversa complessità di tempo.

È semplice progettare una macchina di Turing M di complessità polinomiale che calcoli la funzione

$$f(w) = w\#w$$

dove w è una stringa su un alfabeto Σ.

COME CLASSIFICARE I LINGUAGGI

Complessità di tempo: un esempio

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

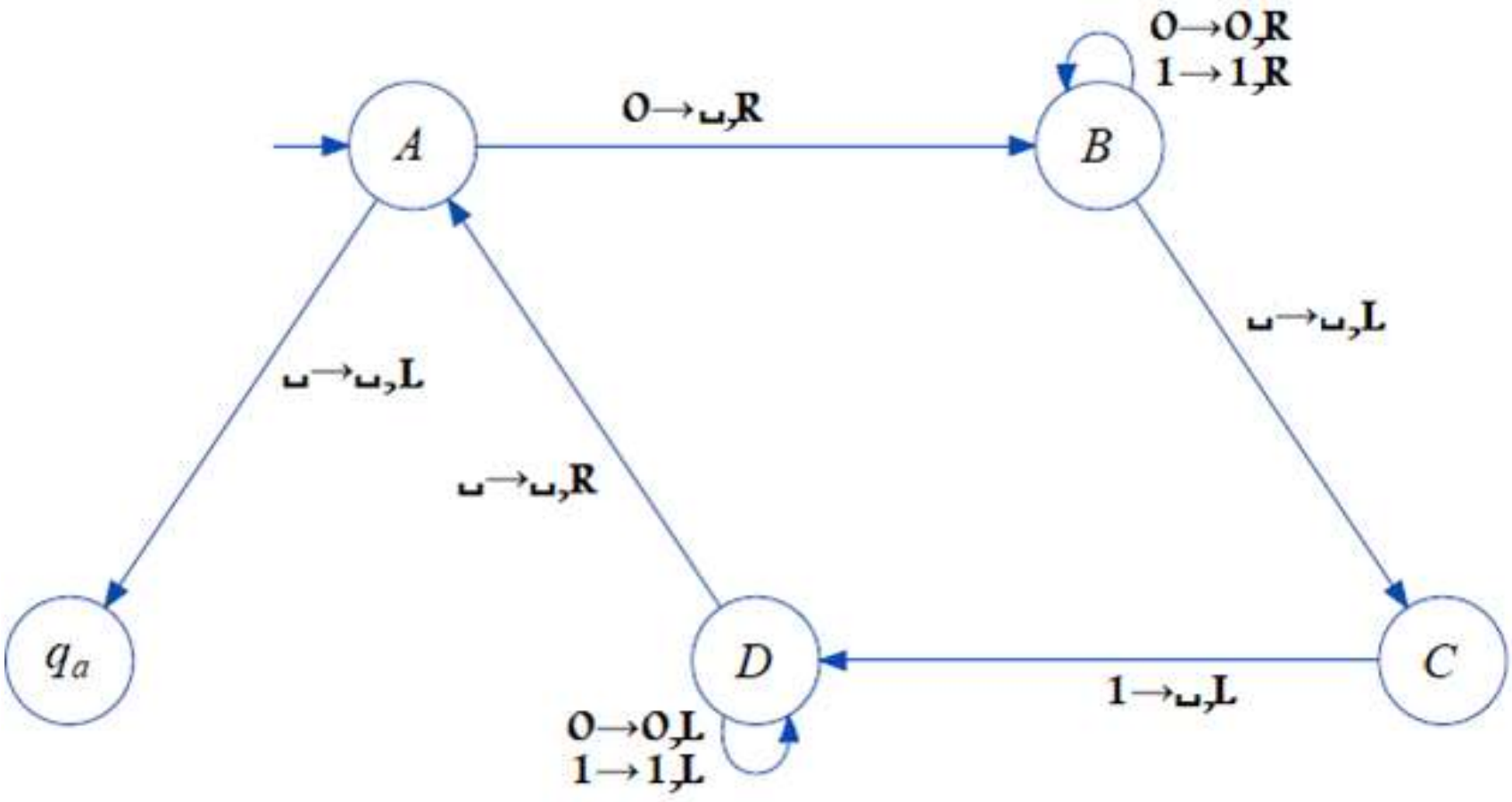

Figura: TM che riconosce $\{0^k 1^k \mid k \geq 0\}$

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.

Complessità di tempo: un esempio

Analisi dell'algoritmo.

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

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

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.

Complessità di tempo: un esempio

Si noti che non abbiamo fatto menzione del riposizionamento della testina del nastro nella descrizione della macchina. L'utilizzo della notazione asintotica ci permette di omettere quei dettagli della descrizione della macchina che influenzano il tempo di esecuzione al più di un fattore costante.

Complessità di tempo: un esempio

Le azioni 2 e 3 richiedono ciascuna $O(n)$ passi e sono eseguite al più $\frac{n}{2}$ volte.

Infatti, la macchina esegue ripetutamente la scansione del nastro e cancella uno 0 e un 1 ad ogni scansione.

Ogni scansione utilizza $O(n)$ passi.

Poichè ogni scansione elimina due simboli, possono verificarsi al più $n/2$ scansioni. Poi la macchina accetta o rifiuta.

Quindi M decide L in tempo
$O(n) + \frac{n}{2}O(n) = O(n) + O(n^2) = O(n^2)$.

Complessità di tempo: obiettivo

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 $f : \mathbb{N} \to \mathbb{R}^+$ 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 - Esempi

$TIME(1)$: 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).

Classi di complessità

Una classe di complessità di tempo è una famiglia di linguaggi. La proprietà che determina l'appartenza di un linguaggio L alla classe è la complessità di tempo di un algoritmo per decidere L.

Complessità di tempo: un esempio

Torniamo al linguaggio $L = \{0^k 1^k \mid k \geq 0\}$.

L'analisi precedente mostra che
$L = \{0^k 1^k \mid k \geq 0\} \in TIME(n^2)$.

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

Potremmo migliorare il tempo di esecuzione, cancellando due simboli 0 e due simboli 1 ad ogni scansione invece di uno solamente, perché questo ridurrebbe il numero di scansioni della metà.

Ma questo migliorerebbe il tempo di esecuzione solo per un fattore 2 e non influenzerebbe il tempo di esecuzione asintotico.

La seguente macchina M_2, utilizza un metodo differente per decidere L asintoticamente più velocemente.

Essa mostra che $L \in TIME(n \log n)$.

Complessità di tempo: un esempio

Il seguente algoritmo M_2 (ovvero la MdT equivalente ad M_2) decide $L = \{0^k 1^k \mid k \geq 0\}$ in tempo $O(n \log n)$.

M_2 = "Sulla stringa input w:

1. Verifica che $w = 0^t 1^q$ (altrimenti rifiuta w)
2. Ripete le due operazioni seguenti finché almeno un carattere 0 e almeno un carattere 1 resta sul nastro:
3. Verifica che la lunghezza della stringa sia pari (altrimenti rifiuta).
4. Scandisce nuovamente l'input, cancellando prima ogni secondo zero, a partire dal primo zero, poi cancellando ogni secondo 1 a partire dal primo 1.
5. Se nessun carattere uguale a 0 e a 1 resta sul nastro, accetta. Altrimenti rifiuta."

Complessità di tempo: un esempio

(Correttezza dell'algoritmo.) L'algoritmo in primo luogo verifica se w è una stringa di caratteri uguali a 0 seguiti da una stringa di caratteri uguali a 1, cioè $w = 0^t 1^q$.

In ogni scansione eseguita nella fase 4, il numero totale di 0 rimanenti è ridotto della metà e ogni eventuale resto viene scartato.

Ad esempio, se abbiamo iniziato con 13 simboli uguali a 0, dopo una prima esecuzione della fase 4 rimangono solo 6 simboli uguali a zero.

In generale, dopo la prima esecuzione del passo 4 sulla stringa $w = 0^t 1^q$, le fasi 3 e 4 verranno applicate alla stringa $0^{t/2} 1^{q/2}$ se t e q sono pari, alla stringa $0^{\frac{t-1}{2}} 1^{\frac{q-1}{2}}$ se t e q sono dispari.

Complessità di tempo: un esempio

Nella fase 3 l'algoritmo controlla che il numero totale di 0 e 1 rimanenti sia pari. Vediamo prima su un esempio a cosa serve questa fase.

Esempio: Sia $0^{13}1^{13}$ l'input.

La fase 3 è applicata nell'ordine a $0^{13}1^{13}$, $0^{6}1^{6}$, $0^{3}1^{3}$, $0^{1}1^{1}$ (la fase 4, applicata a $0^{1}1^{1}$, cancella 01 e si passa alla fase 5).

Il numero totale di 0 e 1 è pari se il numero di zeri e quello di 1 hanno la stessa parità (pari/dispari).

Quando la fase 3 controlla che il numero totale di 0 e 1 rimanenti sia pari, in realtà sta controllando la coerenza della parità del numero di 0 con la parità del numero di 1. Cioè controlla che siano entrambi pari o entrambi dispari.

Complessità di tempo: un esempio

Esempio: Sia $0^{13}1^{13}$ l'input.

La fase 3 è applicata nell'ordine a $0^{13}1^{13}$, $0^{6}1^{6}$, $0^{3}1^{3}$, $0^{1}1^{1}$.

La sequenza delle parità è uguale per 0 e 1 ed è (dispari, pari, dispari, dispari).

Se le parità corrispondono sempre, le rappresentazioni binarie dei numeri t di 0 e q di 1 coincidono, e quindi i due numeri sono uguali.
(Si confrontano sempre i bit meno significativi delle successive divisioni per 2).

Complessità di tempo: un esempio

Esempio: Sia $0^{13}1^{13}$ l'input.

La fase 3 è applicata nell'ordine a $0^{13}1^{13}$, $0^{6}1^{6}$, $0^{3}1^{3}$, $0^{1}1^{1}$.

La sequenza delle parità è (dispari, pari, dispari, dispari).

La rappresentazione binaria di 13 è 1101.

La sua inversa (o reverse) è

$$1011$$

e corrisponde a

(dispari, pari, dispari, dispari)

se sostituiamo "dispari" a 1 e "pari" a 0.

Complessità di tempo: un esempio

Quando la fase 3 controlla che il numero totale di 0 e 1 rimanenti sia pari, in realtà sta controllando la coerenza della parità (pari/dispari) del numero di 0 con la parità del numero di 1.

Cioè controlla che siano entrambi pari o entrambi dispari.

Se le parità corrispondono sempre, le rappresentazioni binarie dei numeri t di 0 e q di 1 coincidono, e quindi i due numeri sono uguali.

Questo perché la sequenza delle parità (ottenuta dall'algoritmo) fornisce sempre l'inversa (o reverse) della rappresentazione binaria.

Complessità di tempo: un esempio

Passiamo a esaminare il tempo di esecuzione.

$M_2 =$ "Sulla stringa input w:

1. Verifica che $w = 0^t1^q$ (altrimenti rifiuta w)
2. Ripete le due operazioni seguenti finché almeno un carattere 0 e almeno un carattere 1 resta sul nastro:
3. Verifica che la lunghezza della stringa sia pari (altrimenti rifiuta).
4. Scandisce nuovamente l'input, cancellando prima ogni secondo zero, a partire dal primo zero, poi cancellando ogni secondo 1 a partire dal primo 1.
5. Se nessun carattere uguale a 0 e a 1 resta sul nastro, accetta. Altrimenti rifiuta."

Complessità di tempo: un esempio

Sia $n = |w|$. Per analizzare il tempo di esecuzione di M_2, per prima cosa osserviamo che 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 $O(n) = O(|w|)$.

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 $O(\log |w|)$ iterazioni del ciclo prima di averli cancellati tutti.

Il tempo totale delle fasi 2,3, e 4 è $O(n \log n)$. Il tempo di esecuzione di M_2 è $O(n) + O(n \log n) = O(n \log n)$.

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

UNA DIFFERENZA TRA TEORIA DELLA COMPUTABILITÀ E TEORIA DELLA COMPLESSITÀ

Abbiamo parlato di complessità di tempo di **una MdT deterministica, a nastro singolo,** che si arresta su ogni input.

Anche nella definizione di classe di complessità abbiamo fatto riferimento al modello a nastro singolo.

Potremmo definire più in generale e in maniera analoga la complessità di tempo di una MdT deterministica multinastro. Cambierebbe qualcosa?

Due osservazioni sulla complessità di tempo

La complessità di tempo dipende dal modello di calcolo?

Sì.

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

- Esiste una macchina di Turing (con un nastro) che decide L in tempo $O(n \log n)$.
- Si potrebbe dimostrare che **non** esiste una macchina di Turing (con un nastro) che decide L in tempo $O(n)$.

Due osservazioni sulla complessità di tempo

$L = \{0^k 1^k \mid k \geq 0\}$ può essere deciso da una macchina di Turing M_3 con due nastri in tempo $O(n)$.

$M_3 =$ "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.

Due osservazioni sulla complessità di tempo

Questa macchina è semplice da analizzare. 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.

Due osservazioni sulla complessità di tempo

Questa discussione evidenzia una differenza importante tra la teoria della computabilità e la teoria della complessità.

Nella 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.

Nella teoria della complessità, la scelta del modello influisce sulla complessità di tempo. Per esempio, linguaggi decidibili in tempo lineare da un modello non sono necessariamente decidibili in tempo lineare da un altro modello.

Due osservazioni sulla complessità di tempo

La teoria della complessità classifica i linguaggi (o problemi) decidibili L in base alla complessità di tempo di un algoritmo che decide L.

Ma con quale modello misureremo la complessità di tempo?

Due osservazioni sulla complessità di tempo

Vedremo che la nostra classificazione non sarà molto sensibile a relativamente piccole differenze di complessità, come quelle che si verificano passando da un modello deterministico a un altro modello deterministico.

Quindi la scelta del modello **deterministico** non sarà cruciale.

Due osservazioni sulla complessità di tempo

Le varianti di macchine di Turing deterministiche introdotte (MdT multinastro, MdT che possono lasciare ferma la testina) sono **polinomialmente equivalenti**, cioè possono simularsi tra di loro con un sovraccarico computazionale polinomiale.

Anche altri modelli di calcolo deterministici proposti in alternativa alle macchine di Turing deterministiche possono simularsi tra di loro con un sovraccarico computazionale polinomiale.

Eccezione: la macchina di Turing non deterministica.

Relazioni tra i modelli: MdT multinastro

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.

Due osservazioni sulla complessità di tempo

La complessità di tempo dipende dalla codifica utilizzata?

Sì.

- Esempio: Consideriamo la funzione (calcolabile) f definita da $f(\langle m \rangle) = \langle m \rangle \# 1^m$, dove:

  $m \in \mathbb{N}$,
  $\langle m \rangle$ è la rappresentazione in base $b \geq 1$ di m, è la codifica che scegliamo per l'input,
  1^m è la rappresentazione in base 1 (unaria) di m.

Due osservazioni sulla complessità di tempo

Sia f definita da $f(\langle m \rangle) = \langle m \rangle \# 1^m$,
dove $m \in \mathbb{N}$,
$\langle m \rangle$ è la rappresentazione in base $b \geq 1$ di m (la codifica che scegliamo per l'input),
1^m è la rappresentazione in base 1 (unaria) di m.

Il valore della funzione dipende dalla base che scegliamo per la rappresentazione, cioè dalla codifica dell'input.

A seconda della codifica che scegliamo, il valore della funzione cambia.

In particolare cambia la lunghezza del valore della funzione rispetto alla lunghezza dell'input.

Due osservazioni sulla complessità di tempo

Ad esempio, per $m = 7$

se scegliamo per l'input la rappresentazione in base 2: $f(111) = 111\#1111111$, l'input ha lunghezza 3, l'output ha lunghezza 11;

se scegliamo per l'input la rappresentazione in base 10: $f(7) = 7\#1111111$, l'input ha lunghezza 1, l'output ha lunghezza 9;

se scegliamo per l'input la rappresentazione unaria: $f(1111111) = 1111111\#1111111$, l'input ha lunghezza 7, l'output ha lunghezza 15.

Due osservazioni sulla complessità di tempo

Se scegliamo per l'input la rappresentazione unaria,
$\langle m \rangle$ = rappresentazione unaria di m,

la macchina M che calcola f è la macchina che copia:
$\langle m \rangle \to \langle m \rangle \# \langle m \rangle$

M ha complessità di tempo $O(|\langle m^2 \rangle|)$ (**polinomiale**).

Due osservazioni sulla complessità di tempo

Supponiamo di scegliere per l'input m la rappresentazione in base 2.

Se $m = a_0 2^0 + \ldots + a_k 2^k$, l'input $\langle m \rangle$ è $a_k \cdots a_0$ e ha lunghezza $k+1$. (È noto che $k+1 = |\langle m \rangle|$ è $O(\log m)$).

La macchina M che calcola f deve scrivere (dopo l'ultimo carattere a destra dell'input $\langle m \rangle$) una stringa di lunghezza $m+1 = t$.

Siccome $2^k \leq m \leq 2^{k+1} - 1$ allora

$$2^{|\langle m \rangle|-1} + 1 \leq t \leq 2^{|\langle m \rangle|}$$

Poiché M deve eseguire almeno t passi, la complessità di tempo di M è $O(2^{|\langle m \rangle|})$ (e anche $\Omega(2^{|\langle m \rangle|})$).

Due osservazioni sulla complessità di tempo

Un altro esempio.

Consideriamo il problema di decisione:

Dato un numero x, x è primo?

e il linguaggio associato

$$PRIMO = \{\langle x \rangle \mid x \in \mathbb{N},\ x \text{ è primo}\}$$

Due osservazioni sulla complessità di tempo

Un algoritmo semplice che decide *PRIMO*:

1. Dividi x per tutti gli interi i, con $1 < i < x$.
2. Se tutti i resti delle divisioni sono diversi da zero, x è primo.

Quante divisioni?

$x - 2$

Quindi, se $n = |\langle x \rangle|$,

$O(n)$, se $\langle x \rangle$ è la rappresentazione unaria,
$O(2^n)$, se $\langle x \rangle$ è la rappresentazione binaria.

Due osservazioni sulla complessità di tempo

- 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.

Due osservazioni sulla complessità di tempo

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.

Metodi di codifica familiari per grafi, automi e oggetti simili sono tutti ragionevoli.

Relazioni tra i modelli: macchina di Turing non deterministica

Un discorso a parte va fatto per il modello non deterministico di macchina di Turing.

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.

Relazioni tra i modelli: MdT non deterministica

Ricordiamo che una macchina di Turing non deterministica è un decisore se, per ogni stringa input w, tutte le computazioni a partire da $q_0 w$ terminano in una configurazione di arresto.

Relazioni tra i modelli: MdT non deterministica

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*,* $n \in \mathbb{N}$*.*

Relazioni tra i modelli: MdT non deterministica

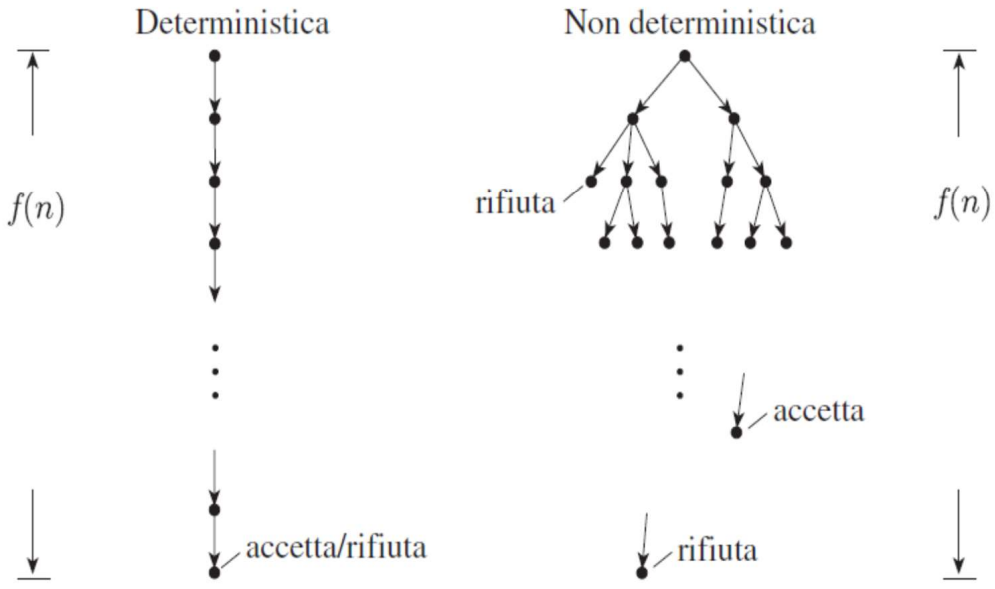

FIGURA 7.10
Misurazione del tempo nei casi deterministico e non deterministico

Figura tratta da M. Sipser, Introduzione alla teoria della computazione.

Relazioni tra i modelli: MdT non deterministica

Il tempo di esecuzione di una macchina non deterministica su input w viene definito come il tempo usato dalla computazione corrispondente alla ramificazione più lunga (nell'albero delle computazioni su w).

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 $w \in \Sigma^n$.

Relazioni tra i modelli: MdT non deterministica

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.

Perché?

Tempo polinomiale

- Esempio: Dato un elenco di n città, stabilire se esiste un giro turistico che visiti ogni città 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 $n = 1000$ è maggiore di $10^{79+13+9+12}$.

Tempo polinomiale

Algoritmi aventi tempo polinomiale sono abbastanza veloci per molti scopi mentre algoritmi aventi tempo esponenziale sono raramente utili.

Algoritmi aventi tempo esponenziale si presentano in genere quando risolviamo problemi mediante una ricerca esaustiva nello spazio delle soluzioni.

A volte, la ricerca mediante forza bruta può essere evitata attraverso una comprensione più approfondita del problema, che può suggerire un algoritmo polinomiale di maggiore utilità.

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.

Tempo polinomiale

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: tempo polinomiale

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 $k \geq 1$, cioè

$$P = \bigcup_{k \geq 1} TIME(n^k).$$

- *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 è importante perché:

1. P corrisponde, con una certa approssimazione, alla classe di problemi che sono realisticamente risolubili mediante programmi su computer reali. Quindi P è una classe rilevante dal punto di vista pratico.
2. P è una classe "robusta dal punto di vista matematico".

Che significa "robusta dal punto di vista matematico"?

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.

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.

Nota. Nel seguito

Algoritmo $=$ *Decider*
$=$ *Macchina di Turing deterministica (a un nastro) che si arresta su ogni input.*

- Nota: per mostrare che un algoritmo può essere eseguito in tempo $O(n^k)$, su un input di lunghezza n, da M dobbiamo:

  1. Fornire un limite superiore al numero dei passi eseguiti dall'algoritmo che sia **polinomiale in** n ,
  2. Mostrare che ogni passo può essere eseguito in tempo **polinomiale in** n da M (o da un qualsiasi "ragionevole" modello di computazione deterministico).

  Infatti allora esiste $h \in \mathbb{N}$ tale che ogni passo può essere eseguito in tempo $O(n^h)$. Se il numero dei passi eseguiti dall'algoritmo è $O(n^c)$, l'algoritmo potrà essere eseguito in tempo $O(n^{h+c})$, su un input di lunghezza n.

La classe P, una precisazione

Nel testo figura spesso la frase: "la composizione di (un numero finito) di polinomi è un polinomio."

In generale questa frase si riferisce alla seguente situazione: N è un algoritmo che, su un input di lunghezza n, simula una MdT M_1 di complessità in tempo polinomiale $O(n^k)$ e poi sull'output, simula una MdT M_2 di complessità in tempo polinomiale $O(m^t)$.

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

Si conclude che N è un algoritmo di complessità in tempo polinomiale. **Perché?**

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

Infatti, il numero di passi di N su un input di lunghezza n è uguale al numero di passi di M_1 su tale input più il numero di passi di M_2 **sull'output di** M_1.

La lunghezza dell'output di M_1 non è necessariamente n ma è certamente limitata dalla complessità in tempo di M_1.

Per $t(n) \geq n$, qualsiasi macchina che opera in tempo $t(n)$ può visitare al più $t(n)$ celle, perché una macchina può esplorare al più una nuova cella ad ogni passo della propria esecuzione.

Se l'input di M_1 (e di N) ha lunghezza n, il corrispondente output ha lunghezza $O(n^k)$.

La classe P, una precisazione

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

Se M_1 ha complessità $O(n^k)$ ed M_2 ha complessità $O(m^t)$ allora N ha complessità $O(n^{kt})$.

La classe P, una precisazione

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

- 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).

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

$PATH \in P$.

- Una generazione esaustiva dei cammini di G (algoritmo di "forza bruta") condurrebbe a un algoritmo di complessità esponenziale.
- Se V è l'insieme dei nodi in G, bisogna generare tutti i sottoinsiemi di V di cardinalità maggiore o uguale di 2. Tale numero è $O(2^{|\langle G,s,t\rangle|})$.

Esempi di linguaggi in P

Il seguente algoritmo M (ovvero la MdT equivalente ad M) decide *PATH* in tempo deterministico polinomiale.

$M =$ "Sull'input $\langle G, s, t \rangle$, dove G è un grafo con nodi s e t:

1. Marca il nodo s.
2. Ripete questa operazione finché nessun nuovo vertice viene marcato:
3. Scansiona tutti gli archi di G. Se trova un arco (a, b) che va da un nodo a marcato ad un nodo b non marcato, marca il nodo b.
4. Se t è marcato, accetta. Altrimenti rifiuta."

M decide $PATH$ in tempo deterministico polinomiale.

Ovviamente, le fasi 1 e 4 sono eseguite una sola volta.

Sia m il numero dei vertici di G. Il passo 3 viene eseguito al più m volte perché viene eseguito ogni volta che viene marcato un nuovo nodo in G e ne posso marcare al più m.

Quindi il numero totale dei passi è al più $1 + 1 + m$, che è polinomiale nella lunghezza dell'input.

Infine i passi 1, 3 e 4 possono essere implementati in tempo polinomiale nella lunghezza dell'input su una MdT deterministica.

Esempi di linguaggi in P

- Due numeri interi positivi x, y sono **relativamente primi** (o coprimi) se il loro massimo comun divisore è 1 (cioè 1 è il più grande intero che li divide entrambi).

$$RELPRIME =$$

$$\{\langle x, y\rangle \mid x \text{ e } y \text{ sono interi positivi relativamente primi}\}$$

Teorema
$RELPRIME \in P$.

Esempi di linguaggi in P

- Una ricerca esaustiva dei divisori non banali di x e y (metodo "forza bruta") condurrebbe a un algoritmo di complessità esponenziale.

  Ricerca esaustiva: dividere x per tutti i naturali i, con $1 < i < x$, e y per tutti i naturali j, con $1 < j < y$.
  Ci sono $x - 2$ naturali i, con $1 < i < x$.

  Se $\langle x \rangle = a_k \cdots a_0$ è la rappresentazione binaria di x allora $x - 2$ è $O(2^k) = O(2^{|\langle x \rangle|})$.

- L'algoritmo che permette di provare che $RELPRIME \in P$ è basato sull'algoritmo di Euclide (circa 300 a.C.) per calcolare il massimo comune divisore $MCD(x, y)$ di due numeri interi non negativi x, y.

Teorema (teorema di ricorsione del MCD)

Per qualsiasi numero intero a non negativo e qualunque intero b positivo, $MCD(a, b) = MCD(b, a \pmod b)$.

Algoritmo di Euclide

$MCD(a, b)$
if $b = 0$ then $MCD = a$
else $MCD = MCD(b, a \pmod b)$

- Nota: si può provare che la procedura è corretta per induzione: se $b = 0$ la procedura restituisce un valore corretto, se $b \neq 0$ usiamo il teorema di ricorsione del MCD e l'ipotesi induttiva.

Algoritmo di Euclide

$MCD(a, b)$
if $b = 0$ then $MCD = a$
else $MCD = MCD(b, a \pmod b)$

- Sono necessarie $O(\log b)$ chiamate ricorsive.
- Complessità di tempo di *MCD* logaritmica rispetto al *valore* dei due numeri.

  Quindi, *lineare* rispetto alla loro codifica (polinomiale, considerando anche il costo - logaritmico rispetto al valore dei due numeri - di ogni chiamata).

Teorema
$RELPRIME \in P$.

Dimostrazione.
Consideriamo l'algoritmo R:

> $R =$ "Sull'input $\langle x, y \rangle$, dove x e y sono numeri naturali in binario:
>
> 1. Simula MCD su $\langle x, y \rangle$
> 2. Se il risultato è 1 accetta. Altrimenti rifiuta."

R (ovvero la MdT equivalente ad R) decide $RELPRIME$ in tempo deterministico polinomiale. □