Complessità

Calcolabilità: studia la frontiera tra problemi solubili e insolubili,

Non considera la difficoltà dei problemi
(distingue solo ciò che è risolubile da ciò che non lo è).

Complessità

Calcolabilità: studia la frontiera tra problemi solubili e insolubili,

Non considera la difficoltà dei problemi
(distingue solo ciò che è risolubile da ciò che non lo è).

Complessità:
considera solo problemi **solubili** allo scopo di fornirne una caratterizzazione dal punto di vista della quantità di risorse di calcolo necessarie a risolverli.

Complessità

Le risorse di cui si tiene principalmente conto quando si scrivono o si utilizzano programmi sono relative al **tempo** e allo **spazio** (ma queste non sono le uniche risorse critiche usate durante il calcolo).

Ci limiteremo a considerare il **tempo** utilizzato per la soluzione di un problema.

Complessità

- Quali problemi considereremo?
- Problemi di **decisione**
  (I problemi di decisione sono problemi che hanno come soluzione una risposta sì o no).
- che sono **decidibili**

Complessità

Ricorda:

- Problema P decidibile $\Longleftrightarrow$ Linguaggio L_P decidibile
- Dimensione dell' input x $\Longleftrightarrow$ $|\langle x \rangle|$.
- Esempio:
  G è un grafo connesso?$\Longleftrightarrow$$\{\langle G \rangle \mid G \text{ è un grafo connesso}\}$
  Dimensione di G (numero nodi) $\Longleftrightarrow$ $|\langle G \rangle|$

Complessità temporale

Come misuriamo il tempo?

- In funzione della taglia n dell'input, utilizzando la notazione O-grande **(analisi asintotica)**;
- nel **caso peggiore**,
  cioè relativo alla stringa di input di taglia n che richiede il maggior numero di passi.

Analisi asintotica

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

Definizione

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

$f = O(g(n))$ *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 asintotica

Esempi

- $f(n) = 1 + 10n + 7n^2 + \ldots + 22n^6$
  $f(n) = O(n^6)$
  Nota: risulta anche $f(n) = O(2^n)$, ma non significativo!
- In generale:

$$a_c n^c + \ldots + a_1 n + a_0 = O(n^c).$$

- $3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$

Analisi asintotica

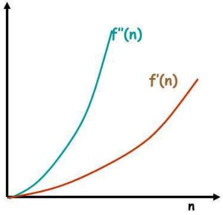
Espressione O	Nome informale
$O(1)$	costante
$O(\log n)$	logaritmico (base?)
$O(n)$	lineare
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k), k \geq 1$	polinomiale
$O(d^n), d \geq 2$	esponenziale

Analisi asintotica

Espressione O	Nome informale
$O(n^k), k \geq 1$	polinomiale
$O(d^n), d \geq 2$	esponenziale

Confronto di complessit\'a Dato un problema
consideriamo 2 algoritmi A e B con compl. f'(n) e f''(n)

$f'(n) = 10n^2$ $f''(n) = 2^n$



n<10, f''(n) $\leq$ f'(n)

n$\geq$10, f''(n) > f'(n)

n=20, f''(n) = 1000 f'(n)

n=30, f''(n) =100000 f'(n)

Complessità temporale

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 temporale** *di* M *è la funzione* $f : \mathbb{N} \to \mathbb{N}$

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

Cioè $f(n) =$ *massimo numero di passi in* $q_0 w \to^* uqv$*,* $q \in \{q_{accept}, q_{reject}\}$*, al variare di* w *in* Σ^n*.*

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

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

La complessità temporale dipende dalla codifica utilizzata

Codificare un numero intero n in base 2 richiede $\lceil \log n + 1 \rceil$ ($=$ più piccolo intero $\geq n + 1$) cifre binarie, mentre codificarlo in base unaria richiede n cifre unarie.

Essendo l'input più lungo, la macchina ha più tempo a disposizione: avere una complessità temporale $O(n)$ rispetto alla codifica unaria, può voler dire che la complessità temporale rispetto alla codifica binaria sia $O(2^n)$.

- Esempio. PRIMO: Dato un numero intero x, x è primo? Algoritmo semplice: dividi x per tutti gli interi $i < x$. Se tutti i resti di tali divisioni sono diversi da zero, x è primo.

  Richiede: ponendo $n = |\langle x \rangle|$
  $O(n)$ passi se x è rappresentato in unario,
  $O(2^n)$ passi se se x è rappresentato in base 2.

Due osservazioni sulla complessità temporale

Occorre considerare codifiche “ragionevoli”: **non “prolisse”** cioè tali che le istanze non abbiano una rappresentazione artificiosamente lunga.

Esempio:

- considerare codifiche in base $k \geq 2$ dei **numeri** (cioè escludere la rappresentazione unaria),
- rappresentare **grafi** come coppie di insiemi (di nodi e archi) o mediante la matrice di adiacenza,
- rappresentare **insiemi, relazioni, funzioni** mediante enumerazione delle codifiche dei relativi elementi.

Codifiche “ragionevoli” dei dati sono polinomialmente correlate: è possibile passare da una di esse a una qualunque altra codifica “ragionevole” delle istanze dello stesso problema in un tempo polinomiale rispetto alla rappresentazione originale.

La complessità temporale dipende dal modello di calcolo

Le varianti di macchine di Turing deterministiche introdotte possono simularsi tra di loro con un sovraccarico computazionale polinomiale.
Anche gli altri modelli di calcolo possono simularsi vicendevolmente con un sovraccarico computazionale polinomiale.

Eccezione: **non determinismo.**

Relazioni tra i modelli: MdT multinastro

Ricordiamo la definizione di MdT multinastro.

Definizione (MdT a k nastri)

Dato un numero naturale k, **una macchina di Turing con k nastri** *è una settupla*

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

dove $Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}$ sono definiti come in una MdT deterministica e la funzione di transizione δ è definita al modo seguente:

$$\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$$

Relazioni tra i modelli: MdT multinastro

Teorema

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

Ricordando la definizione della MdT M' che simula la macchina M a k nastri:

1. Ogni mossa di M viene simulata da M' scorrendo il nastro (che contiene i k nastri di M)
2. il numero di elementi sul ogni nastro di M è al piú $t(n)$ (Una macchina non può toccare un numero di caselle maggiore al numero dei passi che compie e per ipotesi $t(n) \geq n$)
3. Totale per ogni mossa di M': $O(t(n))$

Relazioni tra i modelli: MdT non deterministica

Definizione (Macchina di Turing non deterministica)

Una Macchina di Turing **non deterministica** *è una settupla* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *dove:*

- $Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}$ *sono definiti come in una MdT deterministica*
- *la funzione di transizione* δ *è definita al modo seguente:*

$$\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

MdT non deterministica: Analisi della simulazione

Teorema

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

Idea Dimostrazione

Una macchina di Turing a tre nastri D che simula N:

- copia sul primo nastro la stringa input w
- genera sul nastro 3 (in ordine lessicografico) le stringhe corrispondenti alle possibili computazioni di N su w
  per ognuna, simula sul secondo nastro tale computazione di N su w
  Analisi: Questo richiede $O(t(|w|))$ passi.

MdT non deterministica: Analisi della simulazione

- **In conclusione:**
  D effettua $O(|w|) + O(t(|w|)) \times O(b^{t(|w|)}) = 2^{O(t(|w|))}$ passi per simulare $t(|w|)$ passi di N su w.

  (Ricorda: per ipotesi $t(n) \geq n$).
- D ha tre nastri. Quindi N può essere simulata da una MdT T deterministica a nastro singolo con complessità temporale $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$.

$\square$

La classe P

Vogliamo definire classi chiuse rispetto al cambio del modello di calcolo utilizzato e al cambio di rappresentazione dei dati.

Definizione

La classe P è l'insieme dei linguaggi L per i quali esiste una macchina di Turing M con un solo nastro che decide L e per cui $t_M(n) = O(n^k)$ per qualche $k \geq 1$, cioè

Tesi di Cook

Tesi di Cook (o Strong Church-Turing Thesis):
Semplificando, la tesi di Church-Turing afferma che tutto quello che risulta computabile puó essere computato da una MdT deterministica.
La versione forte, afferma la correlazione polinomiale nel tempo tra algoritmi e computazione di una MdT deterministica.
A supporto di tale tesi:

- Vera per tutti i computer attuali.
- Posso progettare una TM deterministica che simula un qualsiasi computer.

Quindi, in base alla tesi di Cook, P è l'insieme dei problemi di decisione che ammettono un algoritmo polinomiale

Tesi di Cook

La classe P:
Quindi, possiamo definire P come
l'insieme dei problemi che ammettono un algoritmo polinomiale

La Classed P

La classe P è spesso considerata come la classe dei problemi computazionali che sono "efficientemente risolvibili" o "trattabili".

I problemi che sono risolvibili in teoria, ma non possono essere risolti nella pratica, sono chiamati intrattabili.

Esistono problemi in P che sono intrattabili in termini pratici, ad esempio alcuni richiedono almeno $n^1000000$ operazioni.

P è nota contenere molti problemi naturali, quali le versioni di decisione di: programmazione lineare, calcolo del massimo comune divisore, ricerca di un matching massimo.
Nel 2002, è stato dimostrato che il problema di determinare se un numero è primo è in P.