Complessità

Calcolabilità: studia la frontiera tra problemi solubili e problemi insolubili,

Complessità: analizza problemi solubili.

Scopo: quantificare le risorse in tempo e/o spazio necessarie a risolvere un problema dato, in funzione della sua taglia.

Complessità

La teoria della calcolabilità si limita ad aspetti qualitativi della risolubilità dei problemi
(distingue ciò che è risolubile da ciò che non lo è).

Complessità

La teoria della calcolabilità si limita ad aspetti qualitativi della risolubilità dei problemi
(distingue ciò che è risolubile da ciò che non lo è).

La teoria della complessità si occupa di una caratterizzazione dei problemi **(risolubili per ipotesi)** 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).

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

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?

Complessità

- Quali problemi considereremo?

  Problemi di **decisione**

  (I problemi di decisione sono problemi che hanno come soluzione una risposta sì o no).

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**
  (cioè tali che il linguaggio associato sia decidibile).

Complessità

- Problema P decidibile $\Longleftrightarrow$ Linguaggio L_P decidibile

- Problema P decidibile $\Longleftrightarrow$ Linguaggio L_P decidibile

  Esempio: G è un grafo connesso se e solo se $\langle G \rangle$ appartiene al linguaggio associato.

Complessità

- Problema P decidibile $\Longleftrightarrow$ Linguaggio L_P decidibile

  Esempio: G è un grafo connesso se e solo se $\langle G \rangle$ appartiene al linguaggio associato.
- Taglia input x $\Longleftrightarrow$ $|\langle x \rangle|$.

Complessità temporale

Come misuriamo il tempo?

Complessità temporale

Come misuriamo il tempo?

- Si raggruppano input per dimensione (taglia n dell'input)
- **Complessità**: funzione f(n), con n=dimensione input, che rappresenta il numero di unità di tempo usate dall' algoritmo
  - Unità di tempo varia: es. numero di istruzioni semplici in linguaggio usato, numero di passi MdT,....
  - Tempo effettivo dipende da vari paramentri: velocit del processore usato, compilatore,.

Complessità temporale

Come misuriamo il tempo?

- Si raggruppano input per dimensione (taglia n dell'input)
- **Complessità**: funzione f(n), con n=dimensione input, che rappresenta il numero di unità di tempo usate dall' algoritmo
  - Unità di tempo varia: es. numero di istruzioni semplici in linguaggio usato, numero di passi MdT,....
  - Tempo effettivo dipende da vari paramentri: velocit del processore usato, compilatore,.

  nel **caso peggiore**
  (cioè relativo all' input di taglia n che richiede il maggior numero di passi).

Complessità temporale

Come misuriamo il tempo?

- Si raggruppano input per dimensione (taglia n dell'input)
- **Complessità**: funzione f(n), con n=dimensione input, che rappresenta il numero di unità di tempo usate dall' algoritmo
  - Unità di tempo varia: es. numero di istruzioni semplici in linguaggio usato, numero di passi MdT,....
  - Tempo effettivo dipende da vari paramentri: velocit del processore usato, compilatore,.

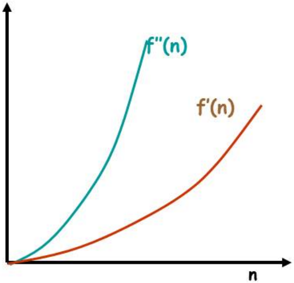
  nel **caso peggiore**

  (cioè relativo all' input di taglia n che richiede il maggior numero di passi).
- utilizzando la notazione O–grande **(analisi asintotica)**

Complessità temporale

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

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



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

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

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

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

Analisi asintotica

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

Definizione

Analisi asintotica

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

Definizione

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

Analisi asintotica

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

Definizione

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

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

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

Analisi asintotica

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

Definizione

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

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

Complessità temporale

Es. Data $f(n)=(n+1)(n+2)$, mostriamo che $f(n)=\mathcal{O}(n^2)$.

Prendiamo $n_0=1$, $c=6$:

$$
\begin{aligned}
f(n) &= (n+1)(n+2) = n^2+3n+2 \\
&\leq n^2+3n^2+2n^2 \quad (\text{per } n \geq 1,\ \ n_0 = 1 \leq n \leq n^2) \\
&= 6n^2 = c\, n^2
\end{aligned}
$$

Analisi asintotica

Costanti non hanno valore

$f(n)=\mathcal{O}(d\ f(n))$, per ogni costante d

Infatti: siano $n_0=0$, $c=1/d$. Si ha

$$f(n)=(1/d)\ d\ f(n)= c\ (d\ f(n))$$

Analisi asintotica

Low-order terms non hanno valore Dato il polinomio

$f(n)=a_k n^k+a_{k-1}n^{k-1}+\ldots+a_1 n+a_0$, con $a_k>0$

risulta

$$f(n)=\mathcal{O}(n^k)$$

Siano $n_0=1$, $c=\sum_{i=0,a_i>0}^{k} a_i$ (nota $a_i \le c$ per ogni $0 \le i \le k$)

Si ha

$$f(\mathrm{n}) = \sum_{i=0}^{k} a_i\, n^i \le \sum_{i=0,a_i>0}^{k} a_i\, n^i$$

$$\le \sum_{i=0,a_i>0}^{k} a_i\, n^k \quad (\text{essendo } 1 \le n)$$

$$= n^k \sum_{i=0,a_i>0}^{k} a_i = n^k\, c = cn^k .$$

Analisi asintotica

<u>Low-order terms non hanno valore</u> **Dato il polinomio**

$f(n)=a_k n^k+a_{k-1}n^{k-1}+\ldots+a_1 n+a_0$, con $a_k>0$

risulta

$$f(n)=\mathcal{O}(n^k)$$

$$\text{Es. } f(n) = 10\, n^5 + 3n^3 - 2n^2 + 1$$

$$n_0 = 1, \quad c = \sum_{i=0, a_i>0}^{k} a_i = 10 + 3 + 1 = 14$$

$$f(n) = 10\, n^5 + 3n^3 - 2n^2 + 1 \le 10\, n^5 + 3n^3 + 1$$
$$\le 10\, n^5 + 3n^5 + n^5 = 14\, n^5 = cn^5$$

Analisi asintotica

Transitività

Se $f(n)=\mathcal{O}(g(n))$ e $g(n)=\mathcal{O}(h(n))$ allora $f(n)=\mathcal{O}(h(n))$

$f(n)=\mathcal{O}(g(n))$ → Esistono c', n' tali che $f(n) \leq c' g(n)$ per ogni $n \geq n'$

$g(n)=\mathcal{O}(h(n))$ → Esistono c'', n'' tali che $g(n) \leq c'' h(n)$ per ogni $n \geq n''$

Quindi, prendiamo $n_0=\max\{n', n''\}$ e $c=c'c''$

Per $n \geq n_0$ $f(n) \leq c' g(n) \leq c' (c'' h(n)) = c\, h(n)$

Analisi asintotica

Si vuole come $\mathcal{O}$-grande la funzione con il minimo tasso di crescita!!!

Es. $f(n)=12n+3$, si ha $f(n)=\mathcal{O}(n)$

risulta anche $f(n)=\mathcal{O}(n^2)$, $f(n)=\mathcal{O}(n^3)$, $f(n)=\mathcal{O}(2^n)$, ...
ma non è quello che vogliamo.

Analisi asintotica

- $3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$
- È necessario specificare la base del logaritmo?

Analisi asintotica

- $3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$
- È necessario specificare la base del logaritmo?
- No, perchè $\log_a n = (\log_a b)(\log_b n)$.

Analisi asintotica

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

Complessità temporale

La quantità **precisa** di risorse utilizzate nella pratica dipende (anche) dalle caratteristiche strutturali e tecnologiche delle macchine usate.

Complessità temporale

La quantità **precisa** di risorse utilizzate nella pratica dipende (anche) dalle caratteristiche strutturali e tecnologiche delle macchine usate.
Per classificare i problemi in base alla loro intrinseca difficoltà faremo riferimento a un **modello astratto**.

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.*

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* $f : \mathbb{N} \to \mathbb{N}$

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

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.

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

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) $f(n)$

Complessità temporale: un esempio

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

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT M a nastro singolo che decide L.
Sull'input w, M:

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT M a nastro singolo che decide L.
Sull'input w, M:

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

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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 $\sqcup$.

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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 $\sqcup$.

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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 $\sqcup$:

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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.

Complessità temporale: un esempio

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

Abbiamo visto una MdT 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.

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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 temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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: l'azione 1 richiede $O(n)$ passi, le azioni 2 e 3 richiedono ciascuna $O(n)$ passi e sono eseguite al più $\frac{n}{2}$ volte.

Complessità temporale: un esempio

Esempio. $L = \{0^n 1^n \mid n \geq 0\}$
Abbiamo visto una MdT 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: l'azione 1 richiede $O(n)$ passi, le azioni 2 e 3 richiedono ciascuna $O(n)$ passi e sono eseguite al più $\frac{n}{2}$ volte.
- Quindi M decide L in tempo $O(n) + \frac{n}{2}O(n) = O(n) + O(n^2) = O(n^2)$.

Complessità temporale: un esempio

E' possibile decidere $L = \{0^n 1^n \mid n \geq 0\}$ in tempo $O(n \log n)$

$M =$ "Sulla stringa input w:

1. Verifica che $w = 0^t 1^q$ (altrimenti rifiuta w)
2. **While** qualche carattere 0 e 1 sul nastro:
   2.1 Verifica che la lunghezza della stringa sia pari (altrimenti rifiuta).
   2.2 Cancella ogni 0 di posizione dispari, a partire dal primo, e ogni 1 di posizione dispari, a partire dal primo.
3. Se nessun carattere uguale a 0 o a 1 resta sul nastro, accetta. Altrimenti rifiuta."

Per induzione su $t + q$ si può provare che l'algoritmo decide correttamente se $w = 0^t 1^q \in L$

Complessità temporale: un esempio

E' possibile decidere $L = \{0^n 1^n \mid n \geq 0\}$ in tempo $O(n \log n)$

$M =$ "Sulla stringa input w:

1. Verifica che $w = 0^t 1^q$ (altrimenti rifiuta w)
2. **While** qualche carattere 0 e 1 sul nastro:
   - 2.1 Verifica che la lunghezza della stringa sia pari (altrimenti rifiuta).
   - 2.2 Cancella ogni 0 di posizione dispari, a partire dal primo, e ogni 1 di posizione dispari, a partire dal primo.
3. Se nessun carattere uguale a 0 o a 1 resta sul nastro, accetta. Altrimenti rifiuta."

ciclo while dopo la prima esecuzione
– è applicato 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

Quindi il numero di 0 e 1 si dimezza (almeno) ad ogni passo e il ciclo viene eseguito $O(\log n)$ volte.

Ogni passo dell'algoritmo richiede tempo $O(|w|)$, quindi $O(n \log n)$

Classi di complessità

Definizione (Classe di complessità in tempo deterministico)

Sia $f : \mathbb{N} \to \mathbb{R}^+$ *una funzione, sia* $\mathcal{M}$ *l'insieme delle MdT deterministiche e a nastro singolo.*

Classi di complessità

Definizione (Classe di complessità in tempo deterministico)

Sia $f : \mathbb{N} \rightarrow \mathbb{R}^+$ *una funzione, sia* $\mathcal{M}$ *l'insieme delle MdT deterministiche e a nastro singolo.*

La classe di complessità in 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

Definizione (Classe di complessità in tempo deterministico)

Sia $f : \mathbb{N} \to \mathbb{R}^+$ *una funzione, sia* $\mathcal{M}$ *l'insieme delle MdT deterministiche e a nastro singolo.*

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

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

Una classe di complessità temporale è una famiglia di linguaggi. La proprietà che determina l'appartenza di un linguaggio L alla classe è la complessità temporale di un algoritmo per decidere se una stringa è in L.
Esempio $L = \{0^n 1^n \mid n \geq 0\} \in TIME(n \log n)$.

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?

Sì.

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

Non esiste MdT (con un nastro) che decide L in tempo $O(n)$

L può essere deciso da MdT con due nastri in tempo $O(n)$:

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

1. Scorre primo nastro verso destra fino a primo 1: per ogni 0, scrive un 1 sul secondo nastro

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

1. Scorre primo nastro verso destra fino a primo 1: per ogni 0, scrive un 1 sul secondo nastro
2. Scorre primo nastro verso destra e secondo nastro verso sinistra: se simboli letti non uguali, termina in q_{reject}

Due osservazioni sulla complessità temporale

La complessità temporale dipende dal modello di calcolo?
Sì.
Esempio: $L = \{0^n 1^n \mid n \geq 0\}$
Non esiste MdT (con un nastro) che decide L in tempo $O(n)$
L può essere deciso da MdT con due nastri in tempo $O(n)$:

1. Scorre primo nastro verso destra fino a primo 1: per ogni 0, scrive un 1 sul secondo nastro
2. Scorre primo nastro verso destra e secondo nastro verso sinistra: se simboli letti non uguali, termina in q_{reject}
3. Se legge $\sqcup$ su entrambi i nastri, termina in q_{accept}

Due osservazioni sulla complessità temporale

- Le varianti di macchine di Turing deterministiche introdotte possono simularsi tra di loro con un sovraccarico computazionale polinomiale.

Due osservazioni sulla complessità temporale

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

Due osservazioni sulla complessità temporale

- Le varianti di macchine di Turing deterministiche introdotte possono simularsi tra di loro con un sovraccarico computazionale polinomiale.
- Anche gli altri modelli di calcolo proposti in alternativa alle macchine di Turing possono simularsi vicendevolmente con un sovraccarico computazionale polinomiale.
- Unica eccezione: la macchina di Turing non deterministica.

Due osservazioni sulla complessità temporale

La complessità temporale dipende dalla codifica utilizzata?

Due osservazioni sulla complessità temporale

La complessità temporale dipende dalla codifica utilizzata?

Sì.

Due osservazioni sulla complessità temporale

La complessità temporale dipende dalla codifica utilizzata?

Sì.

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 ...

Due osservazioni sulla complessità temporale

La complessità temporale dipende dalla codifica utilizzata?

Sì.

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 ...

... esponenzialmente più di una codifica "ragionevole" in base k per un qualche $k \geq 2$

Due osservazioni sulla complessità temporale

Occorre considerare codifiche "ragionevoli": **non "prolisse"** cioè tali che non vi siano istanze la cui rappresentazione sia artificiosamente lunga.

- Esempio: considerare codifiche in base $k \geq 2$ dei numeri (cioè escludere la rappresentazione unaria dei numeri),

Due osservazioni sulla complessità temporale

Occorre considerare codifiche "ragionevoli": **non "prolisse"** cioè tali che non vi siano istanze la cui rappresentazione sia artificiosamente lunga.

- Esempio: considerare codifiche in base $k \geq 2$ dei numeri (cioè escludere la rappresentazione unaria dei numeri),
- grafi come coppie di insiemi (di nodi e archi) o mediante la matrice di adiacenza,
  ciò assicura dimensione input polinomiale nella dimensione (intesa come numero di nodi/archi) del grafo

Due osservazioni sulla complessità temporale

Occorre considerare codifiche "ragionevoli": **non "prolisse"** cioè tali che non vi siano istanze la cui rappresentazione sia artificiosamente lunga.

- Esempio: considerare codifiche in base $k \geq 2$ dei numeri (cioè escludere la rappresentazione unaria dei numeri),
- grafi come coppie di insiemi (di nodi e archi) o mediante la matrice di adiacenza,
  ciò assicura dimensione input polinomiale nella dimensione (intesa come numero di nodi/archi) del grafo

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.

Relazioni tra i modelli: MdT multinastro

Ricordiamo la definizione di MdT multinastro.

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 $O(t^2(n))$.

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 $O(t^2(n))$.

Ricordiamo la definizione della MdT M' che simula M a k nastri: Il contenuto del nastro di M' è la concatenazione di k blocchi separati da $\#$ (seguita da $\sqcup$) e ogni blocco corrisponde a un nastro di M. Un elemento $\dot{\gamma}$, con $\gamma \in \Gamma$, nel blocco $t-$esimo indica la posizione della testina del nastro $t-$esimo, $t \in \{1, \ldots, k\}$.

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 $O(t^2(n))$.

Ricordiamo la definizione della MdT M' che simula M a k nastri: Il contenuto del nastro di M' è la concatenazione di k blocchi separati da $\#$ (seguita da $\sqcup$) e ogni blocco corrisponde a un nastro di M. Un elemento $\dot{\gamma}$, con $\gamma \in \Gamma$, nel blocco $t-$esimo indica la posizione della testina del nastro $t-$esimo, $t \in \{1, \ldots, k\}$.
Quindi a una configurazione di M

$$(u_1 q a_1 v_1, \ldots u_k q a_k v_k)$$

corrisponde la configurazione di M'

MdT multinastro: Analisi della simulazione

Sia $w = w_1 \cdots w_n$ una stringa input, $w_t \in \Sigma$, $t \in \{1, \ldots, k\}$.

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

MdT multinastro: Analisi della simulazione

Sia $w = w_1 \cdots w_n$ una stringa input, $w_t \in \Sigma$, $t \in \{1, \ldots, k\}$.

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

2. Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.

MdT multinastro: Analisi della simulazione

Sia $w = w_1 \cdots w_n$ una stringa input, $w_t \in \Sigma$, $t \in \{1, \ldots, k\}$.

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

2. Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.
3. Nota: Se una testina di M si sposta sulla cella a destra del carattere diverso da $\sqcup$ più a destra del suo nastro allora la testina virtuale (puntino) sul segmento corrispondente del nastro di M' si sposta sul delimitatore $\#$. In questo caso, M' cambia $\dot{\#}$ con $\dot{\sqcup}$ e deve spostare di una posizione tutto il suffisso del nastro a partire da $\dot{\#}$ (cambiato con $\#$) fino all'ultimo $\#$ a destra.

MdT multinastro: Analisi della simulazione

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

MdT multinastro: Analisi della simulazione

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

- Analisi:
  M' inizia generando la configurazione che simula la configurazione iniziale di M. Questo richiede $O(n)$ passi, dove n è la lunghezza dell'input.

MdT multinastro: Analisi della simulazione

1. **(generazione della configurazione iniziale di M)** M' passa dalla configurazione iniziale $q_0 w_1 \cdots w_n$ alla configurazione

$$q' \# \dot{w}_1 \cdots w_n \# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$$

- Analisi:
  M' inizia generando la configurazione che simula la configurazione iniziale di M. Questo richiede $O(n)$ passi, dove n è la lunghezza dell'input.

  Infatti, abbiamo visto che per inserire $\#$, M' esegue due passi per ogni carattere di $w_1 \cdots w_n$, a partire dall'ultimo. Poi occorrono ancora $O(k) = O(1)$ passi per cambiare w_1 in $\dot{w}_1$ e scrivere la stringa $\# \dot{\sqcup} \# \ldots \# \dot{\sqcup} \#$ dopo w_n.

MdT multinastro: Analisi della simulazione

(2) Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.

MdT multinastro: Analisi della simulazione

(2) Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.

(3) Inoltre potrebbe dover spostare di una posizione un suffisso della stringa sul nastro a partire da $\dot{\#}$ (cambiato con $\#$) fino all'ultimo $\#$ a destra.

MdT multinastro: Analisi della simulazione

(2) Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.

(3) Inoltre potrebbe dover spostare di una posizione un suffisso della stringa sul nastro a partire da $\dot{\#}$ (cambiato con $\#$) fino all'ultimo $\#$ a destra.

- Nota: Una macchina non può toccare un numero di caselle maggiore al numero dei passi che compie. Di conseguenza, dopo $t(n)$ passi di M, la lunghezza di ogni blocco corrispondente a un nastro è al piú $t(n)$.

MdT multinastro: Analisi della simulazione

(2) Per simulare un passo di computazione di M, per effetto dell'applicazione di $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, d_1, ..., d_k)$, la macchina M' scorre il dato di ingresso dal primo $\#$ al $(k+1)-$esimo $\#$ da sinistra a destra e **viceversa** due volte.

(3) Inoltre potrebbe dover spostare di una posizione un suffisso della stringa sul nastro a partire da $\dot{\#}$ (cambiato con $\#$) fino all'ultimo $\#$ a destra.

- Nota: Una macchina non può toccare un numero di caselle maggiore al numero dei passi che compie. Di conseguenza, dopo $t(n)$ passi di M, la lunghezza di ogni blocco corrispondente a un nastro è al piú $t(n)$.

  Quindi la lunghezza totale del nastro scritto è al piú
  $K = k \times (t(n) + 1) + 1$
  (la presenza degli addendi 1 è dovuta ai simboli $\#$).

MdT multinastro: Analisi della simulazione

- Analisi:
  Per simulare ognuno dei $t(n)$ passi di M, la MdT M' scorre il dato da sinistra a destra e viceversa; e, nel caso peggiore, effettua k spostamenti di suffissi.

MdT multinastro: Analisi della simulazione

- Analisi:

  Per simulare ognuno dei $t(n)$ passi di M, la MdT M' scorre il dato da sinistra a destra e viceversa; e, nel caso peggiore, effettua k spostamenti di suffissi.

  Ognuna di queste operazioni richiede tempo $O(t(n))$, quindi M' effettua $O(t(n))$ passi per simulare uno dei passi di M.

MdT multinastro: Analisi della simulazione

- Analisi:
  Per simulare ognuno dei $t(n)$ passi di M, la MdT M' scorre il dato da sinistra a destra e viceversa; e, nel caso peggiore, effettua k spostamenti di suffissi.

  Ognuna di queste operazioni richiede tempo $O(t(n))$, quindi M' effettua $O(t(n))$ passi per simulare uno dei passi di M.
- **In conclusione:**
  M' effettua $O(n) + t(n) \times O(t(n)) = O(t^2(n))$ passi per simulare $t(n)$ passi di M.

  (Ricorda: per ipotesi $t(n) \geq n$).

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*

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:*

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

Relazioni tra i modelli: MdT non deterministica

Anche per le macchine di Turing non deterministiche abbiamo definito due famiglie di linguaggi:

Relazioni tra i modelli: MdT non deterministica

Anche per le macchine di Turing non deterministiche abbiamo definito due famiglie di linguaggi:

- I linguaggi L riconosciuti da macchine di Turing non deterministiche, cioè tali che esiste una MdT non deterministica M con $L = L(M)$

Relazioni tra i modelli: MdT non deterministica

Anche per le macchine di Turing non deterministiche abbiamo definito due famiglie di linguaggi:

- I linguaggi L riconosciuti da macchine di Turing non deterministiche, cioè tali che esiste una MdT non deterministica M con $L = L(M)$
- I linguaggi L decisi da macchine di Turing non deterministiche.

Relazioni tra i modelli: MdT non deterministica

Anche per le macchine di Turing non deterministiche abbiamo definito due famiglie di linguaggi:

- I linguaggi L riconosciuti da macchine di Turing non deterministiche, cioè tali che esiste una MdT non deterministica M con $L = L(M)$
- I linguaggi L decisi da macchine di Turing non deterministiche. Un linguaggio $L \subset \Sigma^*$ è deciso da una macchina di Turing non deterministica $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ se M è tale che:

Relazioni tra i modelli: MdT non deterministica

Anche per le macchine di Turing non deterministiche abbiamo definito due famiglie di linguaggi:

- I linguaggi L riconosciuti da macchine di Turing non deterministiche, cioè tali che esiste una MdT non deterministica M con $L = L(M)$
- I linguaggi L decisi da macchine di Turing non deterministiche.

  Un linguaggio $L \subset \Sigma^*$ è deciso da una macchina di Turing non deterministica $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ se M è tale che:

  1. per ogni $w \in \Sigma^*$, tutte le computazioni a partire da $q_0 w$ terminano in una configurazione di arresto
  2. M riconosce L (cioè esiste una computazione $q_0 w \to^* u q_{accept} v$ se e solo se $w \in L$).

Relazioni tra i modelli: Complessità in tempo non deterministico

Definizione (Complessità in tempo non deterministico)

Sia $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *una MdT non deterministica tale che per ogni* w *tutte le computazioni a partire da* $q_0 w$ *terminano in una configurazione di arresto.*

Relazioni tra i modelli: Complessità in tempo non deterministico

Definizione (Complessità in tempo non deterministico)

Sia $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ una MdT non deterministica tale che per ogni w tutte le computazioni a partire da $q_0 w$ terminano in una configurazione di arresto.

La **complessità temporale** *di N è la funzione $f : \mathbb{N} \to \mathbb{N}$ dove $f(n)$ è il massimo numero di passi eseguiti da N in ogni computazione di N su un input di lunghezza n, per ogni $n \in \mathbb{N}$.*

Relazioni tra i modelli: Complessità in tempo non deterministico

In altri termini, la complessità temporale di una MdT non deterministica N è la funzione $f : \mathbb{N} \rightarrow \mathbb{N}$ dove $f(n) =$ massimo $\{$altezza dell'albero che rappresenta le possibili computazioni su $w \mid w \in \Sigma^n\}$.

Relazioni tra i modelli: MdT non deterministica, Osservazioni preliminari

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 $2^{O(t(n))}$.

Relazioni tra i modelli: MdT non deterministica, Osservazioni preliminari

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 $2^{O(t(n))}$.

Ricordiamo la definizione della MdT D, a tre nastri, che simula N:

Relazioni tra i modelli: MdT non deterministica, Osservazioni preliminari

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 $2^{O(t(n))}$.

Ricordiamo la definizione della MdT D, a tre nastri, che simula N:
Sul **primo nastro** viene copiata la stringa input w e il contenuto del primo nastro non verrà alterato dalle computazioni di D.

Relazioni tra i modelli: MdT non deterministica, Osservazioni preliminari

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 $2^{O(t(n))}$.

Ricordiamo la definizione della MdT D, a tre nastri, che simula N:
Sul **primo nastro** viene copiata la stringa input w e il contenuto del primo nastro non verrà alterato dalle computazioni di D.
Sul **terzo nastro** vengono generate le codifiche delle possibili computazioni di N con input w.

Relazioni tra i modelli: MdT non deterministica, Osservazioni preliminari

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 $2^{O(t(n))}$.

Ricordiamo la definizione della MdT D, a tre nastri, che simula N:

Sul **primo nastro** viene copiata la stringa input w e il contenuto del primo nastro non verrà alterato dalle computazioni di D.

Sul **terzo nastro** vengono generate le codifiche delle possibili computazioni di N con input w.

Sul **secondo nastro** D simula la computazione $\mathcal{C}$ di N su w, per ogni codifica $\mathcal{C}$ generata sul terzo nastro.

Relazioni tra i modelli: MdT non deterministica

- Quante sono le codifiche delle computazioni di N sull'input w?

Relazioni tra i modelli: MdT non deterministica

- Quante sono le codifiche delle computazioni di N sull'input w?

  Se b è il massimo numero di alternative proposte dalla funzione di transizione δ di N, D genera sul nastro 3 tutte le stringhe sull'alfabeto $\Sigma_b = \{1, \ldots, b\}$, di lunghezza minore o uguale a $t(|w|)$, in ordine lessicografico.

Relazioni tra i modelli: MdT non deterministica

- Quante sono le codifiche delle computazioni di N sull'input w?

  Se b è il massimo numero di alternative proposte dalla funzione di transizione δ di N, D genera sul nastro 3 tutte le stringhe sull'alfabeto $\Sigma_b = \{1, \ldots, b\}$, di lunghezza minore o uguale a $t(|w|)$, in ordine lessicografico.

  Il numero di tali stringhe è

$$1 + b + b^2 + \ldots b^{t(|w|)} = \sum_{j=0}^{t(|w|)} b^j = \frac{b^{t(|w|)+1} - 1}{b - 1} = O(b^{t(|w|)}).$$

Relazioni tra i modelli: MdT non deterministica

Alternativamente, D deve effettuare una visita per livelli dell'albero delle computazioni di N su w.

Relazioni tra i modelli: MdT non deterministica

Alternativamente, D deve effettuare una visita per livelli dell'albero delle computazioni di N su w.

D genera sul nastro 3 la lista dei nodi al livello 1, poi quella dei nodi al livello 2, fino a quella dei nodi al livello $t(|w|)$.

Alternativamente, D deve effettuare una visita per livelli dell'albero delle computazioni di N su w.

D genera sul nastro 3 la lista dei nodi al livello 1, poi quella dei nodi al livello 2, fino a quella dei nodi al livello $t(|w|)$.

Il numero totale di nodi in tali liste è $O(b^{t(|w|)})$.

Alternativamente, D deve effettuare una visita per livelli dell'albero delle computazioni di N su w.

D genera sul nastro 3 la lista dei nodi al livello 1, poi quella dei nodi al livello 2, fino a quella dei nodi al livello $t(|w|)$.

Il numero totale di nodi in tali liste è $O(b^{t(|w|)})$.

Infatti, si può dimostrare, mediante induzione, che:

1. il numero di nodi al livello ℓ nell'albero è al più b^{ℓ}

Relazioni tra i modelli: MdT non deterministica

Alternativamente, D deve effettuare una visita per livelli dell'albero delle computazioni di N su w.

D genera sul nastro 3 la lista dei nodi al livello 1, poi quella dei nodi al livello 2, fino a quella dei nodi al livello $t(|w|)$.

Il numero totale di nodi in tali liste è $O(b^{t(|w|)})$.

Infatti, si può dimostrare, mediante induzione, che:

1. il numero di nodi al livello ℓ nell'albero è al più b^{ℓ}
2. quindi il numero dei nodi nell'albero è $\leq 1 + b + b^2 + \ldots b^{t(|w|)} = O(b^{t(|w|)})$.

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 $2^{O(t(n))}$.

Dimostrazione

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

- copia sul primo nastro la stringa input w
  Analisi: Questo richiede $O(|w|)$ passi.

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 $2^{O(t(n))}$.

Dimostrazione

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

- copia sul primo nastro la stringa input w
  Analisi: Questo richiede $O(|w|)$ passi.
- genera sul nastro 3 la stringa successiva a quella corrente in ordine lessicografico
  Analisi: Questo richiede un numero di passi proporzionale alla lunghezza della stringa, quindi $O(t(|w|))$ passi.

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 $2^{O(t(n))}$.

Dimostrazione

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

- copia sul primo nastro la stringa input w
  Analisi: Questo richiede $O(|w|)$ passi.
- genera sul nastro 3 la stringa successiva a quella corrente in ordine lessicografico
  Analisi: Questo richiede un numero di passi proporzionale alla lunghezza della stringa, quindi $O(t(|w|))$ passi.
- simula sul secondo nastro la computazione $\mathcal{C}$ di N su w, per ogni codifica $\mathcal{C}$ generata sul terzo nastro
  Analisi: Questo richiede $O(t(|w|))$ passi.

- **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.

- **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 $t(n) \geq n$).

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

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

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

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

La classe P

- In altri termini, $L \in P$ se e solo se esiste una MdT deterministica M che si arresta su ogni input, tale che

La classe P

- In altri termini, $L \in P$ se e solo se esiste una MdT deterministica M che si arresta su ogni input, tale che
  1. $L(M) = L \quad$ (M decide L).

La classe P

- In altri termini, $L \in P$ se e solo se esiste una MdT deterministica M che si arresta su ogni input, tale che
  1. $L(M) = L \quad (M \text{ decide } L)$.
  2. Esiste $k \geq 1$ tale che, per ogni $w \in \Sigma^*$, la computazione di M a partire da $q_0 w$ si arresta in $O(|w|^k)$ passi.

La classe P

- In altri termini, $L \in P$ se e solo se esiste una MdT deterministica M che si arresta su ogni input, tale che
    1. $L(M) = L \quad (M$ decide $L)$.
    2. Esiste $k \geq 1$ tale che, per ogni $w \in \Sigma^*$, la computazione di M a partire da $q_0 w$ si arresta in $O(|w|^k)$ passi.

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

La classe P

- È una classe:

La classe P

- È una classe:
  - Robusta rispetto al modello di calcolo
    (tutti i modelli "ragionevoli" - equivalenti alle macchine di Turing deterministiche possono simularsi tra di loro con un sovraccarico computazionale polinomiale).

La classe P

- È una classe:
  - Robusta rispetto al modello di calcolo (tutti i modelli "ragionevoli" - equivalenti alle macchine di Turing deterministiche possono simularsi tra di loro con un sovraccarico computazionale polinomiale).
- Viene comunemente identificata con l'insieme dei linguaggi **trattabili**, cioé praticamente risolubili su calcolatore.

Domanda: Quali algoritmi sono utili in pratica?

Complessità in tempo deterministico

Domanda: Quali algoritmi sono utili in pratica?

Una definizione operativa: (Jack Edmonds, 1962)

Domanda: Quali algoritmi sono utili in pratica?

Una definizione operativa: (Jack Edmonds, 1962)

- **Efficienti**: quelli che utilizzano tempo **polinomiale** per **TUTTI** gli input.

Complessità in tempo deterministico

Domanda: Quali algoritmi sono utili in pratica?

Una definizione operativa: (Jack Edmonds, 1962)

- **Efficienti**: quelli che utilizzano tempo **polinomiale** per **TUTTI** gli input.
- **Inefficienti**: quelli che utilizzano tempo **esponenziale** per **QUALCHE** input.

Complessità in tempo deterministico

Domanda: Quali algoritmi sono utili in pratica?

Una definizione operativa: (Jack Edmonds, 1962)

- **Efficienti**: quelli che utilizzano tempo **polinomiale** per **TUTTI** gli input.
- **Inefficienti**: quelli che utilizzano tempo **esponenziale** per **QUALCHE** input.

Nota: Vi sono delle eccezioni. Alcuni algoritmi utilizzano tempo polinomiale n^k con esponente k grande, e sono inutili in pratica. Alcuni algoritmi che utilizzano tempo esponenziale per qualche input sono utilizzati perchè tali input sono rari.

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni “ragionevoli” della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l’una con l’altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.
- Quindi, in base alla tesi di Cook, P è l’insieme dei problemi di decisione solubili in tempo polinomiale sui computer REALI.

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.
- Quindi, in base alla tesi di Cook, P è l'insieme dei problemi di decisione solubili in tempo polinomiale sui computer REALI.
- A supporto di tale tesi:

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.
- Quindi, in base alla tesi di Cook, P è l'insieme dei problemi di decisione solubili in tempo polinomiale sui computer REALI.
- A supporto di tale tesi:
  - Vera per tutti i computer attuali.

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.
- Quindi, in base alla tesi di Cook, P è l'insieme dei problemi di decisione solubili in tempo polinomiale sui computer REALI.
- A supporto di tale tesi:
  - Vera per tutti i computer attuali.
  - Posso progettare una TM deterministica che simula un qualsiasi computer.

Tesi di Cook

- Tesi di Cook (o Strong Church-Turing Thesis):
  1. Tutte le formalizzazioni "ragionevoli" della nozione intuitiva di computazione trattabile sono equivalenti (cioè simulabili l'una con l'altra con un costo in tempo limitato polinomialmente).
  2. La computazione in tempo polinomiale su una MdT deterministica è una formalizzazione ragionevole di computazione trattabile.
- Quindi, in base alla tesi di Cook, P è l'insieme dei problemi di decisione solubili in tempo polinomiale sui computer REALI.
- A supporto di tale tesi:
  - Vera per tutti i computer attuali.
  - Posso progettare una TM deterministica che simula un qualsiasi computer.
- Possibili eccezioni: Computazione quantistica e Computazione molecolare.

Esempi di problemi in P

- Nel seguito gli algoritmi saranno descritti come una lista (finita) di passi in linguaggio naturale.

Esempi di problemi in P

- Nel seguito gli algoritmi saranno descritti come una lista (finita) di passi in linguaggio naturale.
- Dobbiamo essere certi che questo insieme di passi possa essere eseguito da una macchina di Turing deterministica M che si ferma su ogni input.

Esempi di problemi in P

- Nel seguito gli algoritmi saranno descritti come una lista (finita) di passi in linguaggio naturale.
- Dobbiamo essere certi che questo insieme di passi possa essere eseguito da una macchina di Turing deterministica M che si ferma su ogni input.
- Inoltre, per concludere che M ha complessità temporale polinomiale dobbiamo:

Esempi di problemi in P

- Nel seguito gli algoritmi saranno descritti come una lista (finita) di passi in linguaggio naturale.
- Dobbiamo essere certi che questo insieme di passi possa essere eseguito da una macchina di Turing deterministica M che si ferma su ogni input.
- Inoltre, per concludere che M ha complessità temporale polinomiale dobbiamo:
  1. Fornire un limite superiore polinomiale al numero dei passi eseguiti dall'algoritmo,

Esempi di problemi in P

- Nel seguito gli algoritmi saranno descritti come una lista (finita) di passi in linguaggio naturale.
- Dobbiamo essere certi che questo insieme di passi possa essere eseguito da una macchina di Turing deterministica M che si ferma su ogni input.
- Inoltre, per concludere che M ha complessità temporale polinomiale dobbiamo:
  1. Fornire un limite superiore polinomiale al numero dei passi eseguiti dall'algoritmo,
  2. Mostrare che ogni passo può essere eseguito in tempo polinomiale da M (o da un modello di computazione equivalente).

Esempi di problemi 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

$PATH \in P$.

Esempi di problemi 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 (metodo "forza bruta" - algoritmo "brute-force") condurrebbe a un algoritmo di complessità esponenziale.

Esempi di problemi in P

Proof.

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 finchè nessun nuovo vertice viene marcato:
3. per ogni arco (a, b) in G se a è un vertice marcato, marca b se non lo è già.
4. Se t è marcato, accetta. Altrimenti rifiuta."

M decide *PATH* in tempo deterministico polinomiale. Infatti il numero dei passi è al più $1 + 1 + m$ dove m è il numero dei vertici di G (il passo 3 viene eseguito al più m volte). Quindi il numero dei passi è polinomiale nella lunghezza dell'input. Inoltre i passi 1, 3 e 4 possono essere implementati in tempo polinomiale nella lunghezza dell'input su una MdT deterministica. $\square$

Esempi di problemi 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).

Esempi di problemi 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).

$$RELPRIME = \{\langle x, y \rangle \mid x \text{ e } y \text{ sono relativamente primi } \}$$

Esempi di problemi 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).

$$RELPRIME = \{\langle x, y\rangle \mid x \text{ e } y \text{ sono relativamente primi } \}$$

Teorema

$RELPRIME \in P$.

Esempi di problemi in P

- Una ricerca esaustiva dei divisori di x e y (metodo “forza bruta” - algoritmo “brute-force”) condurrebbe a un algoritmo di complessità esponenziale.

Esempi di problemi in P

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

Esempi di problemi in P: Euclide

Teorema (teorema di ricorsione del MCD)

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

Esempi di problemi in P: Euclide

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 $MCD = MCD(b, a \pmod b)$

Esempi di problemi in P: Euclide

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.

Esempi di problemi in P

Algoritmo di Euclide

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

Algoritmo di Euclide

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

- Esempio:

$$
\begin{aligned}
MCD(30, 21) &= MCD(21, 30 \pmod{21}) \\
&= MCD(21, 9) \\
&= MCD(9, 3) \\
&= MCD(3, 0) \\
&= 3
\end{aligned}
$$

Analisi di complessità di Euclide

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod b)$

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod b)$

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod b)$

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).

Algoritmo di Euclide

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

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$

Analisi di complessità di Euclide

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod b)$

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$
  - $b_{k-1} \geq b_k + b_{k+1},\ b_k \geq b_{k+1} \Rightarrow b_{k-1} \geq 2b_{k+1}$

Analisi di complessità di Euclide

Algoritmo di Euclide

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

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$
  - $b_{k-1} \geq b_k + b_{k+1},\ b_k \geq b_{k+1} \Rightarrow b_{k-1} \geq 2b_{k+1}$
  - Per induzione: $b = b_0 \geq 2^{k/2} b_k$ per ogni $k \geq 2$

Analisi di complessità di Euclide

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod{b})$

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$
  - $b_{k-1} \geq b_k + b_{k+1},\ b_k \geq b_{k+1} \Rightarrow b_{k-1} \geq 2b_{k+1}$
  - Per induzione: $b = b_0 \geq 2^{k/2} b_k$ per ogni $k \geq 2$
  - Numero di chiamate al più $O(\log b)$

Analisi di complessità di Euclide

Algoritmo di Euclide

$MCD(a, b)$

if $b = 0$ then $MCD = 0$

else $MCD = MCD(b, a \pmod b)$

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$
  - $b_{k-1} \geq b_k + b_{k+1},\; b_k \geq b_{k+1} \Rightarrow b_{k-1} \geq 2b_{k+1}$
  - Per induzione: $b = b_0 \geq 2^{k/2} b_k$ per ogni $k \geq 2$
  - Numero di chiamate al più $O(\log b)$
- Complessità temporale di MCD logaritmica rispetto al *valore* dei due numeri

Analisi di complessità di Euclide

Algoritmo di Euclide

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

- Sono necessarie al più $O(\log b)$ chiamate ricorsive:
  Siano $(a_{k-1}, b_{k-1}), (a_k, b_k), (a_{k+1}, b_{k+1})$ tre coppie successive (la prima chiama MCD sulla seconda e la seconda chiama MCD sulla terza).
  - $b_{k-1} = a_k = qb_k + b_{k+1} \Rightarrow b_{k-1} = a_k \geq b_k + b_{k+1}$
  - $b_{k-1} \geq b_k + b_{k+1},\ b_k \geq b_{k+1} \Rightarrow b_{k-1} \geq 2b_{k+1}$
  - Per induzione: $b = b_0 \geq 2^{k/2} b_k$ per ogni $k \geq 2$
  - Numero di chiamate al più $O(\log b)$
- Complessità temporale 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)

Esempi di problemi in P

Teorema

$RELPRIME \in P$.

Proof.

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. $\square$