Capitolo 8: Teoria della complessitá

Un problema $\mathcal{P}$, qualunque sia la sua natura, include in primo luogo un insieme finito di *elementi* $\{e_1, \ldots, e_n\}$ ognuno dei quali è una struttura di dati costituita da un numero finito di *parametri* $\{x_{i_1}, \ldots, x_{i_{m_i}}\}$. Ogni parametro può assumere uno dei valori inclusi nell'insieme dei suoi *valori possibili*. I valori possibili di un elemento sono quindi inclusi nel prodotto cartesiano degli insiemi dei valori possibili dei parametri che lo costituiscono.

Esempio 1 *Il problema dell'ordinamento ha due elementi, n la cardinalità dell'insieme dei dati da ordinare, e S, l'insieme dei dati da ordinare. Il primo è un intero, mentre il secondo può essere un array di n interi. In tal caso, n ha un solo parametro, mentre S è caratterizzato da n parametri che assumono tutti valori interi.*

Un'*istanza* di $\mathcal{P}$ è un'assegnazione di valori a tutti i parametri di tutti gli elementi del problema.

Esempio 2 *Un'istanza del problema dell'ordinamento assegna 3 a n e l'array di elementi*
13 11 76
a S.

Dato un problema $\mathcal{P}$ indicheremo con $\mathcal{I}_{\mathcal{P}}$ l'insieme delle possibili istanze. Inoltre indicheremo genericamente con x i valori assegnati ai parametri di tutti gli elementi del problema per una istanza di $\mathcal{P}$.

Detto A un metodo risolutivo di $\mathcal{P}$, cioè un *algoritmo*, diremo che x è l'*input* di A. Indicheremo con $|x|$ la *dimensione* di x, cioè la lunghezza della descrizione dell'istanza stessa.

Valutazioni di complessitá
Denoteremo con

- $\widehat{t}_A(x)$ il tempo di esecuzione (il numero di passi effettuati) dell'algoritmo A per input x,

- $\widehat{s}_A(x)$ lo spazio di memoria utilizzato dall'algoritmo A per input x.

Come già osservato, per istanze di medesima lunghezza, le complessità sia in spazio e sia in tempo di un algoritmo possono essere differenti. In tal caso si dice che esse dipendono dalla *configurazione dei dati in ingresso*. Per questo siamo interessati a determinare la complessità in tempo e in spazio nel caso *peggiore* a parità di lunghezza dell'input, n.

Il *tempo di esecuzione di A nel caso peggiore* (worst case) è il tempo usato da A sull'istanza piú difficile, cioè
$t_A(n) = \max\{\hat{t}_A(x) | \forall x : |x| \leq n\}$.

Quindi $t_A(n)$ rappresenta il tempo di esecuzione di A sull'istanza di lunghezza al più n (secondo una qualche codifica) per il quale tale tempo di esecuzione è massimo.

Lo *spazio di esecuzione di A nel caso peggiore* è lo spazio usato da A sull'istanza piú difficile, cioè
$s_A(n) = \max\{\hat{s}_A(x) | \forall x : |x| \leq n\}$.

Nel caso di complessità spaziale si farà sempre riferimento allo spazio di lavoro addizionale, cioè alla quantità di celle di nastro (in una macchina di Turing) o di locazioni di memoria utilizzate per eseguire la computazione, non considerando lo spazio richiesto per la descrizione iniziale dellistanza in questione.

Data una istanza di un problema, la lunghezza cui si fa riferimento in Teoria della Complessità è definita come il *numero di caratteri* necessari per descrivere tale istanza nellambito di un qualche metodo di codifica predefinito.

Quindi, sia la lunghezza associata ad una istanza che la complessità di un algoritmo operante sull'istanza stessa saranno dipendenti dal metodo di codifica adottato.

Definizione 3 *Dato un problema $\mathcal{P}$ con insieme $\mathcal{I}_\mathcal{P}$ delle possibili istanze, e dato un alfabeto Σ, definiamo* schema di codifica *di $\mathcal{P}$ una funzione $e : \mathcal{I}_\mathcal{P} \longmapsto \Sigma^*$ che mappa ogni istanza di $\mathcal{P}$ in una corrispondente stringa di simboli in Σ^*.*

Uno schema di codifica è *ragionevole* se non ha istanza la cui descrizione è artificiosamente lunga.

L'uso di schemi di codifica non ragionevoli può portare a valutazioni di complessità paradossali.

Esempio 4 *Si consideri il problema di decidere, dato un naturale x, se esso è primo.*

Il semplice algoritmo consistente nel tentare di dividere x per tutti i naturali $i < x$ (e dedurre che x è primo se per tutti questi valori la divisione fornisce resto non nullo) esegue $O(x)$ passi (assumendo, per semplicità, che una divisione sia eseguita in tempo costante rispetto al valore dei relativi operandi) e risulta quindi esponenziale per una codifica ragionevole, quale ad esempio quella più naturale consistente nel rappresentare x per mezzo di $\lceil log_2(x+1) \rceil$ bit.

Al contrario, l'uso di una codifica non ragionevole, quale ad esempio quella unaria, *che utilizza un alfabeto di un solo carattere | e rappresenta l'intero n per mezzo di una stringa di n caratteri |, porterebbe a concludere che l'algoritmo precedente risulta avere complessità lineare.*

Uno schema di codifica ragionevole è uno che:
- considera codifiche in base $k \geq 2$ dei valori numerici interessati;
- rappresenta insiemi mediante enumerazione delle codifiche dei loro elementi;
- rappresenta relazioni e funzioni mediante enumerazione delle codifiche dei relativi elementi (coppie e n-ple);
- rappresenta grafi come coppie di insiemi (di nodi ed archi).

Per gli schemi di codifica ragionevole vale la seguente condizione di correlazione polinomiale:

Definizione 5 *Due schemi di codifica e ed e' per un problema $\mathcal{P}$ sono* polinomialmente correlati *se esistono due polinomi p e p' tali che, per ogni $x \in \mathcal{I}_{\mathcal{P}}$:*
✓$|e(x)| \leq p'(|e'(x)|)$
✓$|e'(x)| \leq p(|e(x)|)$

Pertanto la complessità di un algoritmo risulta invariante (a meno di un polinomio) al variare dello schema utilizzato.

Definizione 6 *Dato un problema $\mathcal{P}$ diciamo che:*
1. $\mathcal{P}$ ha limite superiore di complessitá temporale (upper bound) $O(g(n))$ se esiste un algoritmo A che risolve $\mathcal{P}$ tale che $t_A(n) = O(g(n))$;
2. $\mathcal{P}$ ha limite inferiore di complessitá temporale (lower bound) $\Omega(g(n))$ se per ogni algoritmo A che risolve $\mathcal{P}$ si ha $t_A(n) = \Omega(g(n))$;
3. $\mathcal{P}$ ha complessitá temporale esatta $\Theta(g(n))$ se ha upper bound $O(g(n))$ e lower bound $\Omega(g(n))$;

Tipi di problemi

Da un punto di vista strettamente matematico, ad ogni problema può essere associata una funzione f (eventualmente multivalore) che è definita su un opportuno spazio dei dati o delle istanze $\mathcal{I}$ (codificate mediante un qualche schema) e a valori in uno spazio dei risultati $\mathcal{O}$ ($2^{\mathcal{O}}$ se f è multivalore).

Per ogni funzione $f : \mathcal{I} \longmapsto 2^{\mathcal{O}}$, si possono porre quattro problemi:
1. dato $x \in \mathcal{I}$, si vuole determinare se esiste $y \in f(x)$ (*decisione*);
2. dato $x \in \mathcal{I}$, si vuole costruire un $y \in f(x)$ (*ricerca*);
3. dato $x \in \mathcal{I}$, si vuole conoscere $|f(x)|$ (*enumerazione*);
4. dato $x \in \mathcal{I}$, si vuole la $y \in f(x)$ "migliore" secondo qualche misura data (*ottimizzazione*).

Da tali considerazioni deriva una caratterizzazione dei problemi in termini di problemi di decisione, ricerca, enumerazione ed ottimizzazione, rispettivamente. I problemi di decisione rappresentano la tipologia di problemi più "semplici", nel senso che la loro caratterizzazione richiede il minor numero di concetti introdotti.

Definiamo un **problema di decisione** $\mathcal{P}_D$ per mezzo di:

1. un insieme di istanze $I_{\mathcal{P}_D}$;

2. un predicato $\pi : I_{\mathcal{P}_D} \longmapsto \{\text{VERO,FALSO}\}$ che determina una partizione di $I_{\mathcal{P}_D}$ in un insieme $Y_{\mathcal{P}_D} \cup N_{\mathcal{P}_D}$, dove $Y_{\mathcal{P}_D}$ (istanze positive, $x \in Y_{\mathcal{P}_D}$ sse $\pi(x) = \text{VERO}$), e in un insieme $N_{\mathcal{P}_D}$ (istanze negative, $x \in N_{\mathcal{P}_D}$ sse $\pi(x) = \text{FALSO}$).

Esempio 7 *I problemi di riconoscimento di un linguaggio L sono problemi di decisione, con $I_{\mathcal{P}_D} = \Sigma^*$ e $Y_{\mathcal{P}_D} = L$;*

Se consideriamo una qualsiasi codifica e delle istanze di $\mathcal{P}_D$ mediante stringhe su un qualche alfabeto Σ, possiamo associare a $\mathcal{P}_D$ il problema equivalente di riconoscere il linguaggio $L = \{a \in \Sigma^* | \exists x \in Y_{\mathcal{P}_D}, a = e(x)\}$ costituito da tutte le stringhe che descrivono istanze positive di $\mathcal{P}_D$.

L'insieme di stringhe $\Sigma^* - L$ comprende sia le descrizioni di istanze negative di $\mathcal{P}_D$ che stringhe che non rappresentano descrizioni di istanze $x \in I_{\mathcal{P}_D}$. Comunque, assumeremo di poter decidere in modo efficiente (in un numero di passi polinomiale nella lunghezza della stringa) se una stringa $a \in \Sigma^* - L$ descrive un'istanza di $\mathcal{P}_D$ o meno.

Un algoritmo per un problema di decisione $\mathcal{P}_D$ calcola il predicato π prendendo in input la descrizione di un'istanza $x \in I_{\mathcal{P}_D}$ secondo uno schema di codifica prefissato, e restituendo in output un valore $VERO$, $FALSO$ corrispondente a $\pi(x)$.

Esempio 8 *Il problema di decisione della* **CRICCA** *è definito come segue:*
ISTANZA: Grafo $G = (V, E)$, intero $K > 0$
PREDICATO: $\exists V' \subseteq V$, con $|V'| \geq K$ e tale che $\forall (u, v) \in V'$ si ha $(u, v) \in E$?

Dato un grafo G ed un intero positivo K, un algoritmo per CRICCA restituirà quindi il valore vero se esistono almeno K nodi in G tutti mutuamente collegati tra loro, altrimenti l'algoritmo restituirà il valore falso.

Definiamo un **problema di ricerca** $\mathcal{P}_R$ per mezzo di:

1. un insieme di istanze $I_{\mathcal{P}_R}$;

2. un insieme di soluzioni $S_{\mathcal{P}_R}$;

3. una proprietá di ammissibilitá che vale per tutte le soluzioni di una istanza; essa induce una relazione di ammissibilitá $R \subseteq I_{\mathcal{P}_R} \times S_{\mathcal{P}_R}$. Data una istanza x , $Sol(x) = \{y \in S_{\mathcal{P}_R} | \langle x, y \rangle \in R\}$ è detto insieme delle soluzioni ammissibili di x.

Il problema di ricerca $\mathcal{P}_R$ ha un problema di decisione associato $\mathcal{P}_D$ con $I_{\mathcal{P}_D} = I_{\mathcal{P}_R}$, e con predicato π definito come "esiste $y \in S_{\mathcal{P}_R}$ tale che $\langle x, y \rangle \in R$?"

Un algoritmo per un problema di ricerca $\mathcal{P}_R$ calcola una funzione (parziale) $\phi : I_{\mathcal{P}_R} \mapsto S_{\mathcal{P}_R}$ tale che $\phi(x)$ è definita se e solo se esiste almeno una soluzione $y = \phi(x)$ per la quale $\langle x, y \rangle \in R$.

Esempio 9 *Il problema di ricerca* **RICERCA CRICCA** *è definito come segue:*
ISTANZA: Grafo $G = (V, E)$*, intero* $K > 0$
SOLUZIONE: Insieme di nodi V'
AMMISSIBILITÀ: $V' \subseteq V$, $|V'| \geq K$, $\forall (u,v) \in V'$ $(u,v) \in E$*?*

Dato un grafo G *ed un intero positivo* K*, un algoritmo per RICERCA CRICCA restituirà quindi, se esiste, un sottoinsieme* V' *di* V *che soddisfa la proprietà di ammissibilità.*

Definiamo un **problema di enumerazione** $\mathcal{P}_E$ per mezzo di:

1. un insieme di istanze $I_{\mathcal{P}_E}$;
2. un insieme di soluzioni $S_{\mathcal{P}_E}$;
3. una proprietá di ammissibilitá che induce la relazione di ammissibilitá $R \subseteq I_{\mathcal{P}_E} \times S_{\mathcal{P}_E}$.

Un algoritmo per un problema di enumerazione $\mathcal{P}_E$ calcola una funzione $\psi : I_{\mathcal{P}_E} \mapsto \mathbb{N}$ tale che per ogni $x \in I_{\mathcal{P}_E}$ $\psi(x) = |Sol(x)|$.

Esempio 10 *Il problema di enumerazione* **ENUMERA CRICCHE** *viene definito allo stesso modo di RICERCA CRICCA, ma quel che si vuole in questo caso è restituire un valore intero n pari al numero di soluzioni ammissibili distinte.*

Definiamo un **problema di ottimizzazione** $\mathcal{P}_O$ per mezzo di:

1. un insieme di istanze $I_{\mathcal{P}_O}$;

2. un insieme di soluzioni $S_{\mathcal{P}_O}$;

3. una proprietá di ammissibilitá rappresentata dalla relazione di ammissibilitá $R \subseteq I_{\mathcal{P}_O} \times S_{\mathcal{P}_O}$;

4. una funzione di misura $\mu : I_{\mathcal{P}_O} \times S_{\mathcal{P}_O} \longmapsto \mathbb{N}$ tale che per ogni $x \in I_{\mathcal{P}_O}$, $y \in S_{\mathcal{P}_O}$, $\mu(x,y)$ è definita se e solo se $y \in Sol(x)$;

Un algoritmo per un problema di ottimizzazione $\mathcal{P}_O$ calcola una funzione (parziale) $\chi : I_{\mathcal{P}_O} \mapsto S_{\mathcal{P}_O}$ tale che $\chi(x)$ è definita se e solo se esiste almeno una soluzione $y = \chi(x)$ per la quale $\langle x, y \rangle \in R$, e inoltre per ogni $z \in Sol(x)$, $\mu(x,y) \leq \mu(x,z)$ se $c = MIN$ e $\mu(x,y) \geq \mu(x,z)$ se $c = MAX$.

Esempio 11 *Il problema di ottimizzazione* **MASSIMA CRICCA** *è definito come segue:*
ISTANZA: Grafo $G = (V, E)$
SOLUZIONE: Insieme di nodi V'
AMMISSIBILITÀ: $V' \subseteq V$, $|V'| \geq K$, $\forall (u,v) \in V'$ $(u,v) \in E$
MISURA: $|V'|$
CRITERIO: MAX

Dato un grafo G*, un algoritmo per MASSIMA CRICCA restituirà quindi, se esiste, il sottoinsieme* V' *di* V *che soddisfa la proprietà di ammissibilità e che massimizza la cardinalità di* V'*.*

Complessitá temporale e spaziale

Gran parte della Teoria della complessitá fa riferimento ai problemi di decisione ed ai linguaggi associati. Ogni problema di decisione è il problema di discriminare fra:
✓le stringhe che sono codifiche di istanze in $Y_{\mathcal{P}}$ e
✓le stringhe che sono codifiche di istanze in $N_{\mathcal{P}}$ piú quelle che non codficano istanze del problema.

Definizione 12 *Dato un alfabeto* Σ*, una macchina di Turing (deterministica o non deterministica)* $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ *(con* $\Sigma \subseteq \Gamma$*) ed un intero* $\tau > 0$*, una stringa* $x \in \Sigma^*$ è accettata *da* $\mathcal{M}$ *in* τ *passi se esiste una computazione* $q_0 x \vdash_{\mathcal{M}} c$ *composta da* r *passi, con* c *configurazione finale e* $r \leq \tau$*.*

Definizione 13 *Dato un alfabeto* Σ*, una macchina di Turing deterministica* $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ *(con* $\Sigma \subseteq \Gamma$*) ed un intero* $\tau > 0$*, una stringa* $x \in \Sigma^*$ è rifiutata *da* $\mathcal{M}$ *in* τ *passi se esiste una computazione* $q_0 x \vdash_{\mathcal{M}} c$ *composta da* r *passi, con* c *configurazione massimale e non finale, e* $r \leq \tau$*.*

Definizione 14 *Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ (con $\Sigma \subseteq \Gamma$) ed una funzione $t : \mathbb{N} \longmapsto \mathbb{N}$, diciamo che $\mathcal{M}$ accetta $L \subseteq \Sigma^*$ in tempo $t(n)$ se, per ogni $x \in \Sigma^*$, $\mathcal{M}$ accetta x in tempo al piú $t(|x|)$ se $x \in L$, mentre se $x \notin L$, allora $\mathcal{M}$ non la accetta.*

Definizione 15 *Dato un alfabeto Σ, una macchina di Turing (deterministica) $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$(con $\Sigma \subseteq \Gamma$) ed una funzione $t : \mathbb{N} \longmapsto \mathbb{N}$, diciamo che $\mathcal{M}$ riconosce $L \subseteq \Sigma^*$ in tempo $t(n)$ se, per ogni $x \in \Sigma^*$, $\mathcal{M}$ accetta x in tempo al piú $t(|x|)$ se $x \in L$, e rifiuta x in tempo al piú $t(|x|)$ se $x \notin L$.*

1. Un linguaggio L è accettabile in tempo deterministico $t(n)$ se esiste una mdt deterministica che lo accetta in tempo $t(n)$;
2. Un linguaggio L è accettabile in tempo non deterministico $t(n)$ se esiste una mdt non deterministica che lo accetta in tempo $t(n)$;
3. Un linguaggio L è decidibile in tempo $t(n)$ se esiste una mdt deterministica che decide L in tempo $t(n)$.

Definizione 16 *Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ (con $\Sigma \subseteq \Gamma$) ed un intero $\sigma > 0$, una stringa $x \in \Sigma^*$ è* accettata *da $\mathcal{M}$ in spazio σ se esiste una computazione $q_0 \sharp x \vdash_{\mathcal{M}} c$ nel corso della quale sono accedute v celle di nastro, con c configurazione finale e $v \leq \sigma$.*

Definizione 17 *Dato un alfabeto Σ, una macchina di Turing deterministica $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$(con $\Sigma \subseteq \Gamma$) ed un intero $\sigma > 0$, una stringa $x \in \Sigma^*$ è* rifiutata *da $\mathcal{M}$ in spazio σ se esiste una computazione $q_0 \sharp x \vdash_{\mathcal{M}} c$ composta da r passi, nel corso della quale sono accedute v celle di nastro, con c configurazione massimale e non finale, e $v \leq \sigma$.*

Definizione 18 *Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ (con $\Sigma \subseteq \Gamma$) ed una funzione $s : \mathbb{N} \longmapsto \mathbb{N}$, diciamo che $\mathcal{M}$ accetta $L \subseteq \Sigma^*$ in spazio $s(n)$ se, per ogni $x \in \Sigma^*$, $\mathcal{M}$ accetta x in spazio $s(|x|)$ se $x \in L$, mentre se $x \notin L$, allora $\mathcal{M}$ non la accetta.*

Definizione 19 *Dato un alfabeto Σ, una macchina di Turing (deterministica) $\mathcal{M} = \langle \Gamma, \not{b}, Q, q_0, F, \delta \rangle$ (con $\Sigma \subseteq \Gamma$) ed una funzione $s : \mathbb{N} \longmapsto \mathbb{N}$, diciamo che $\mathcal{M}$ riconosce $L \subseteq \Sigma^*$ in spazio $s(n)$ se, per ogni $x \in \Sigma^*$, $\mathcal{M}$ accetta x in spazio $s(|x|)$ se $x \in L$, e rifiuta x in spazio $s(|x|)$ se $x \notin L$.*

1. Un linguaggio L è accettabile in spazio deterministico $s(n)$ se esiste una mdt deterministica che lo accetta in spazio $s(n)$;
2. Un linguaggio L è accettabile in spazio non deterministico $s(n)$ se esiste una mdt non deterministica che lo accetta in spazio $s(n)$;
3. Un linguaggio L è decidibile in spazio $s(n)$ se esiste una mdt deterministica che decide L in spazio $s(n)$.

Classe di complessitá: l'insieme dei problemi risolubili da un modello di calcolo in un determinato limite delle risorse di calcolo.

1. $DTIME(f(n))$ è l'insieme dei linguaggi decisi da una mdt deterministica a 1 nastro in tempo al piú $f(n)$;
   $DSPACE(f(n))$ è l'insieme dei linguaggi decisi da una mdt deterministica a 1 nastro (con un nastro one-way di input) in spazio al piú $f(n)$;

2. $NTIME(f(n))$ è l'insieme dei linguaggi decisi da una mdt non deterministica a 1 nastro in tempo al piú $f(n)$;
   $NSPACE(f(n))$ è l'insieme dei linguaggi decisi da una mdt non deterministica a 1 nastro (con un nastro one-way di input) in spazio al piú $f(n)$;

Teoremi di compressione e accelerazione

Teorema 20 *(compressione lineare) Sia $\mathcal{M}$ una macchina di Turing a 1 nastro (deterministico o non deterministica) che accetta un linguaggio L in spazio $s(n)$, e sia $c > 0$ una costante. E' possibile costruire una macchina $\mathcal{M}'$ a 1 nastro che accetta L in spazio $s'(n) \leq cs(n)$.*

Dimostrazione. Sia $k = \lceil 1/c \rceil$. Mostriamo come a ogni configuraizone di lunghezza m del nastro di $\mathcal{M}$ possa essere associata una configurazione *compressa* di lunghezza $k = \lceil m/k \rceil$ del nastro di $\mathcal{M}'$.
Ogni cella del nastro di $\mathcal{M}'$ corrisponderà a k celle adiacenti sul nastro di $\mathcal{M}$.
Ciò è ottenuto utilizzando per $\mathcal{M}'$ un alfabeto di nastro Γ' tale che $|\bar{\Gamma}'| = |\bar{\Gamma}|^k$.

Oltre a codificare in modo più compatto i simboli di nastro, occorre rappresentare l'informazione relativa alla posizione della testina, ovvero su quale dei k simboli codificati in uno solo è posizionata la testina.
Ciò è ottenuto considerando gli stati di $\mathcal{M}'$ come coppie (q, j), con $q \in Q$, $1 \leq j \leq k$. Ad ogni istante, $\mathcal{M}'$ si trova nello stato (q, j) se e solo se $\mathcal{M}$ si trova nello staot q e la sua testina è posta sulla j-esima tra le k celle corrispondenti alla posizione su cui si trova la testina di $\mathcal{M}'$.

Osservazione: Il risparmio di spazio deriva esclusivamente da una opportuna modifica del modello di calcolo e non da miglioramenti nella struttura concettuale dell'algoritmo (rappresentata dalla funzione di transizione).

Teorema 21 *(accelerazione lineare) Sia $\mathcal{M}$ una macchina di Turing (deterministico o non deterministica) che accetta un linguaggio L in tempo $t(n)$, e sia $\varepsilon > 0$ una costante. E' possibile costruire una macchina $\mathcal{M}'$ a 2 nastri che accetta L in tempo $t'(n) \leq \lceil \varepsilon t(n) \rceil + O(n)$.*

Essenzialmente i teoremi di compressione e di accelerazione lineare esprimono che aumentando opportunamente la capacità della singola cella di memoria e quindi, in sostanza, la potenza di calcolo delle operazioni elementari definite nel modello, è possibile rendere più efficiente (in termini di operazioni elementari eseguite o di celle di memoria utilizzate) un algoritmo definito su tale modello.

Classi di complessità

In molti casi non siamo interessati a caratterizzare la complessità di un problema in modo molto preciso:

Per quale funzione t(n) il problema appartiene alla classe DTIME(t(n))?

In generale, è invece sufficiente disporre di una classificazione più grossolana, stabilendo ad esempio se il problema è risolubile in tempo polinomiale o esponenziale.

A tale scopo si definiscono le seguenti classi di complessità:

1. $P = \cup_{k=0}^{\infty} DTIME(n^k)$ è la classe dei linguaggi decidibili da una macchina di Turing deterministica in tempo polinomiale nella dimensione dell'input;

2. $PSPACE = \cup_{k=0}^{\infty} DSPACE(n^k)$ è la classe dei linguaggi decidibili da una macchina di Turing deterministica in spazio polinomiale nella dimensione dell'input;

3. $LOGSPACE = \cup_{k=0}^{\infty} DSPACE(\log n)$ è la classe dei linguaggi decidibili da una macchina di Turing deterministica in spazio logaritmico nella dimensione dell'input;

4. $EXPTIME = \cup_{k=0}^{\infty} DTIME(2^{n^k})$ è la classe dei linguaggi decidibili da una macchina di Turing deterministica in tempo proporzionale ad un esponenziale nella dimensione dell'input;

5. $NP = \cup_{k=0}^{\infty} NTIME(n^k)$ è la classe dei linguaggi decidibili da una macchina di Turing non deterministica in tempo polinomiale nella dimensione dell'input;

6. $NPSPACE = \cup_{k=0}^{\infty} NSPACE(n^k)$ è la classe dei linguaggi decidibili da una macchina di Turing non deterministica in spazio polinomiale nella dimensione dell'input;

7. $NEXPTIME = \cup_{k=0}^{\infty} NTIME(2^{n^k})$ è la classe dei linguaggi decidibili da una macchina di Turing non deterministica in tempo proporzionale ad un esponenziale nella dimensione dell'input;

Relazioni di contenimento tra classi di complessità

Teorema 22 *Per ogni funzione* $f : \mathbb{N} \longmapsto \mathbb{N}$ *valgono le due proprietá:*
$DTIME(f(n)) \subseteq NTIME(f(n))$;
$DSPACE(f(n)) \subseteq NSPACE(f(n))$.

Dimostrazione. Deriva semplicemente dall'osservazione che ogni macchina di Turing deterministica è un caso particolare di macchina di Turing non deterministica.

Corollario 23 $P \subseteq NP$; $PSPACE \subseteq NPSPACE$.

Teoremi di gerarchia In molti casi non è difficile individuare relazioni di contenimento (non stretto) tra due classi di complessità (vedi teoremi precedenti).

È più complicato mostrare che tali relazioni sono di contenimento stretto.

Date due classi di complessità $\mathcal{C}_1$, $\mathcal{C}_2$ tali che $\mathcal{C}_1 \subseteq \mathcal{C}_2$, al fine di *separare* $\mathcal{C}_1$ da $\mathcal{C}_2$ (e quindi mostrare che in effetti $\mathcal{C}_1 \subset \mathcal{C}_2$) è necessario mostrare l'esistenza di un linguaggio $L \in \mathcal{C}_2 - \mathcal{C}_1$.

Due approcci:
1. individuare tale linguaggio con modalità "ad hoc"
2. applicazione della tecnica di diagonalizzazione.

La tecnica di diagonalizzazione dà luogo a due classici teoremi, detti *di gerarchia*, che consentono, rispettivamente, la separazione di classi di complessità temporale e spaziale.

Tali teoremi si applicano a classi di complessità definite su funzioni aventi particolari proprietà di "*costruttività*" per mezzo di macchine di Turing.

Definizione 24 *Una funzione non decrescente* $s : \mathbb{N} \mapsto \mathbb{N}$ *è detta* space constructible *se esiste una mdt* $\mathcal{M}$ *che, per ogni* n*, utilizza esattamente* $s(n)$ *celle per ogni istanza di lunghezza* n*.*

Quindi una funzione $s(n)$ è space constructible se esiste una macchina di Turing che per ogni n *marca*, ad esempio scrivendoci dentro un determinato simbolo predefinito, esattamente $s(n)$ celle di nastro, senza utilizzarne altre.

Definizione 25 *Una funzione non decrescente* $t : \mathbb{N} \mapsto \mathbb{N}$ *è detta* time constructible *se esiste una mdt* $\mathcal{M}$ *che, per ogni* n*, si ferma dopo avere eseguito esattamente* $t(n)$ *passi su ogni istanza di lunghezza* n*.*

Quindi una funzione $t(n)$ è time constructible se esiste una macchina di Turing che, per ogni n, "*conta*" esattamente fino a $t(n)$, in quanto entra in un certo stato (il suo stato finale) dopo esattamente $t(n)$ passi di computazione.

Tutte le funzioni pi familiari, e che crescono *sufficientemente*, (e.g., n^k, k^n, $n!$) sono sia space che time constructible.
Non sono space constructible funzioni $s(n) = o(logn)$, mentre non sono time constructible funzioni $t(n) = o(n)$.

Teorema 26 *(di gerarchia spaziale)*
Siano date due funzioni $s_1, s_2 : \mathbb{N} \mapsto \mathbb{N}$ *tali che:*

✓ $s_2(n) > log(n+1)$ *per ogni* n
✓ $s_1(n) = o(s_2(n))$
✓ $s_2(n)$ *è space constructible.*

Allora esiste un linguaggio L *separatore delle due classi di complessità* $DSPACE(s_1(n))$ *e* $DSPACE(s_2(n))$, *cioè* $\exists L \in DSPACE(s_2(n))$ *e* $L \notin DSPACE(s_1(n))$

Corollario 27 $LOGSPACE \subset PSPACE$

Teorema 28 *(di gerarchia temporale)*
Siano date due funzioni $t_1, t_2 : \mathbb{N} \mapsto \mathbb{N}$ *tali che:*

✓ $t_2(n) > n$ *per ogni* n
✓ $t_2(n) = \omega(t_1(n)log(t_1(n)))$
✓ $t_2(n)$ *è time constructible.*

Allora esiste un linguaggio L *separatore delle due classi di complessità* $DTIME(t_1(n))$ *e* $DTIME(t_2(n))$, *cioè* $\exists L \in DTIME(t_2(n))$ *e* $L \notin DTIME(t_1(n))$

Corollario 29 $P \subset EXPTIME$

I teoremi di gerarchia stabiliscono essenzialmente che, se si considerano le classi di complessità definite da funzioni "sufficientemente diverse" in termini di rapidità di crescita asintotica, allora tali classi sono diverse, nel senso che l'insieme dei linguaggi associati alle due classi è diverso.

Relazioni fra misure di complessità

Teorema 30 *Per ogni funzione* $f : \mathbb{N} \longmapsto \mathbb{N}$ *valgono le due proprietá:*
$DTIME(f(n)) \subseteq DSPACE(f(n))$;
$NTIME(f(n)) \subseteq NSPACE(f(n))$.

Dimostrazione.È sufficiente notare che in t passi di computazione si possono utilizzare al più t celle di nastro distinte.

Corollario 31 $P \subseteq PSPACE$; $NP \subseteq NPSPACE$.

Teorema 32 *Per ogni coppia di funzioni* $f, g : \mathbb{N} \longmapsto \mathbb{N}$ *per le quali si ha* $g(n) = O(f(n))$ *valgono le due proprietá:*
$DSPACE(g(n)) \subseteq DSPACE(f(n))$;
$NSPACE(g(n)) \subseteq NSPACE(f(n))$.

Teorema 33 *Per ogni coppia di funzioni* $f, g : \mathbb{N} \longmapsto \mathbb{N}$ *tali che* $g(n) = O(f(n))$ *e* $f(n) = \omega(n)$ *si ha:*
$DTIME(g(n)) \subseteq DTIME(f(n))$;
$NTIME(g(n)) \subseteq NTIME(f(n))$.

Teorema 34 *Per ogni funzione space constructible* $f : \mathbb{N} \longmapsto \mathbb{N}$ *tale che* $f(n) = \Omega(\log n)$ *si ha che* $NSPACE(f(n)) \subseteq DTIME(2^{f(n)})$.

Teorema 35 *Per ogni funzione time constructible* $f : \mathbb{N} \longmapsto \mathbb{N}$ *si ha che* $NTIME(f(n)) \subseteq DSPACE(f(n))$.

Corollario 36 $NP \subseteq PSPACE$.

Teorema 37 *Per ogni funzione space constructible* $f : \mathbb{N} \longmapsto \mathbb{N}$ *tale che* $f(n) = \Omega(\log n)$ *si ha che* $DSPACE(f(n)) \subseteq DTIME(2^{f(n)})$.

Corollario 38 $LOGSPACE \subseteq P$.

Corollario 39 $PSPACE \subseteq EXPTIME$.

Si dimostra inoltre che se si considera il tempo di computazione, il passaggio da modello non deterministico a modello deterministico comporta un aumento al più esponenziale del costo di esecuzione di un algoritmo.

Teorema 40 *Per ogni funzione time constructible* $f : \mathbb{N} \longmapsto \mathbb{N}$ *si ha che* $NTIME(f(n)) \subseteq DTIME(2^{f(n)})$.

Corollario 41 $NP \subseteq EXPTIME$.

Finora nessuna simulazione in tempo sub esponenziale di una macchina di Turing non deterministica mediante macchine di Turing deterministiche è stata individuata, e si congettura che in effetti il passaggio da modello non deterministico a modello deterministico richieda necessariamente un aumento esponenziale del tempo di computazione.

Al contrario, nel caso dello spazio l'ipotesi di utilizzare un modello deterministico comporta un maggior costo limitato (dell'ordine del quadrato) rispetto al caso non deterministico.

Teorema 42 *(teorema di Savitch) Sia data una funzione space constructible* $f : \mathbb{N} \longmapsto \mathbb{N}$ *tale che* $f(n) \geq \log n$, *per ogni* n; *allora* $NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$.

Corollario 43 $NPSPACE \subseteq PSPACE$.

Avendo già dimostrato che $PSPACE \subseteq NPSPACE$, ne consegue che $NPSPACE = PSPACE$.

Quindi si ha l'equivalenza tra modello deterministico e modello non deterministico quando si considerano computazioni a spazio limitato polinomialmente.

Relazioni tra le classi di complessità introdotte.