Complessità e Teoria dell'Informazione

Prof.ssa Carla Piazza

Appunti di Claudio Desideri e Lara Vignotto – a.a. 2023/2024

Indice

Parte I

Complessità

Capitolo 1

Introduzione

Libro di Papadimitriou main reference.

In questa parte utilizzeremo come modello di computazione le **macchine di Turing** (TM). Esistono diversi modelli di TM: macchine di Turing multinastro, macchine di Turing input/output, macchine di Turing con oracolo, macchine di Turing nondeterministiche. Le TM verranno utilizzate per confrontare i diversi risultati di complessità che possiamo ottenere.

Ci concentreremo sia su **complessità temporale** (time complexity) che **spaziale** (space complexity). Il focus non sarà sulla complessità di un dato algoritmo, ma sulla complessità di un problema. I **problemi** possono essere classificati come di decisione (decision problems), di funzione (function problems), o di ottimizzazione (optimization problems).

- **Decision problem** P : inputs $\to \{yes, no\}$
- **Function problem** computare una data funzione, ad esempio l'ordinamento di una lista
- **Optimization problem** tra tutti i possibili output, si vuole trovare quello che minimizza o massimizza una funzione di costo.

Esempio Sia $G = (V, E)$ un grafo, e $u, v \in V$ due nodi.

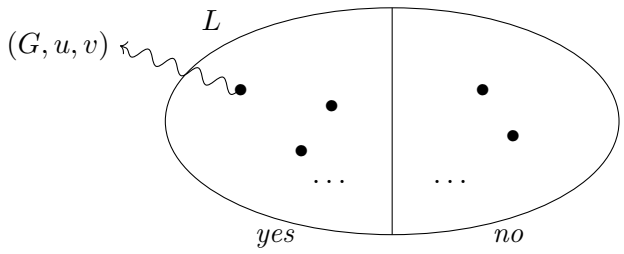
– decidere se esiste un cammino da u a v è un problema di decisione

– trovare un cammino da u a v è un problema di funzione

– trovare il cammino più corto da u a v è un problema di ottimizzazione

In questo corso ci concentreremo sui problemi di decisione. Se si ha una soluzione per un problema di funzione o di ottimizzazione, si possiede automaticamente una soluzione per il problema di decisione.

Immaginiamo tutti gli input possibili al problema dell'esempio precedente come ad un insieme infinito di tuple (G, u, v). Questo insieme si può dividere in due: il sottoinsieme dei *yes* di tutte le codifiche binarie di triple (G, u, v) tali che esiste un cammino in G da u a v, e, inversamente, il sottoinsieme *no*.



La codifica binaria di una tripla è una stringa del tipo $1011\ldots$. Più precisamente, è una stringa sull'alfabeto $\Sigma = \{0, 1\}$. L'insieme di tutte le possibili stringhe binarie è Σ^*. Questo insieme è quindi il linguaggio L sottoinsieme di Σ^*, ovvero $L \subseteq \Sigma^*$.

$$L = \{\text{bin}(G, u, v) | ln\ G\ u \to v\}$$

Esempio Consideriamo interi rappresentati in binario. Vogliamo decidere se un dato intero x è divisibile per 4.

$$\text{bin}(x) = 10\ldots11$$

in questo caso non è divisibile per 4. Un numero binario è divisibile per 4 se e solo se i due bit meno significativi sono 0.

$$\text{bin}(x) = x_n, x_{n-1}, \ldots, x_1, x_0 \quad \Leftrightarrow \quad x_0 = 0 \text{ and } x_1 = 0$$

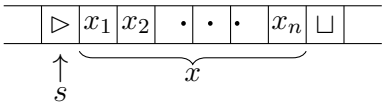
Il linguaggio indicato da questo problema di decisione è

$$L = \{x \in \{0,1\}^* | x = x_n, x_{n-1}, \ldots, x_1, x_0 \wedge x_0 = 0 \wedge x_1 = 0\}$$

Esempio: palindromo Decidere se una stringa è palindroma, con $\Sigma = \{0,1\}$.

$$x_1, x_2, x_3, \ldots, x_3, x_2, x_1$$

Ad esempio, $x = 101$ è palindroma, mentre $x = 1010$ non lo è. Cerchiamo il linguaggio $L = \{x | x \text{ è palindroma}\}$. Utilizziamo una macchina di Turing.



Si parte dallo stato s e si vuole finire nello stato p solo quando x è palindroma. Per decidere se x è palindroma, si può leggere x_1, ricordarne il valore nello stato del puntatore, e poi confrontarlo con x_n. Se sono uguali, si ripete lo stesso procedimento con x_2 e x_{n-1}, e così via. Se si arriva a x_n e x_1 senza aver trovato una discrepanza, allora x è palindroma. Se invece si trova una discrepanza, allora x non è palindroma. Le transizioni sono le seguenti:

$$\begin{aligned}\delta(s, \triangleright) &= (q, \triangleright, \rightarrow)\\ \delta(q, 1) &= (q_1, \triangleright, \rightarrow)\\ \delta(q, 0) &= (q_0, \triangleright, \rightarrow)\end{aligned}$$

TODO: finire di scrivere le transizioni Questa macchina eseguirà un numero quadratico di passi per controllare se la stringa x è palinfroma: $O(|x|^2)$.

Se si vuole controllare in C (o in un altro linguaggio) se una stringa è palindroma, si può scrivere un programma che confronta il primo e l'ultimo carattere, poi il secondo e il penultimo, e così via, eseguendo un numero lineare di passi. La complessità è $O(|x|)$. Questo è un esempio di come la complessità di un problema dipenda dal modello di computazione utilizzato.

1.1 Tesi di Church-Turing Estesa

La tesi di Church-Turing afferma che ogni cosa che può essere computata, può essere computata da una macchina di Turing.

La versione estesa afferma che tutti i modelli (ragionevoli) di calcolo sono correlati polinomialmente. Questo significa che se un problema è risolvibile in tempo polinomiale in un modello di computazione, allora è risolvibile in tempo polinomiale in ogni modello di computazione.

In altre parole, la tesi di Church-Turing estesa afferma che la complessità computazionale di un problema è indipendente dal modello di calcolo utilizzato per risolverlo.

$$\underset{\text{problema}}{P} \rightarrow \text{TMs } O(f(n)) \rightarrow \underset{\text{modello}}{M} \; O(p(f(n)))$$

Ma è vera anche la direzione contraria. TODO: ???

Capitolo 2

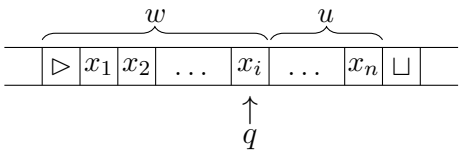
Macchine di Turing e Unlimited Register Machines

2.1 Definizioni

Definizione 2.1.1 (Configurazione) *Una configurazione è una tripla (q, w, u), con*

- $q \in K \cup \{yes, no, halt\}$
- $w, u \in \Sigma^*$

Ad esempio, graficamente, una configurazione è



Definizione 2.1.2 (Configurazione Iniziale) *La configurazione iniziale su una stringa x è una tripla*

$$(1, \triangleright, x)$$

Definizione 2.1.3 (Configurazioni Finali) *Le configurazioni finali su una stringa x sono una tripla*

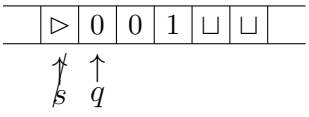
$$(H, w, u)$$

dove $H \in \{yes, no, halt\}$.

Definizione 2.1.4 (Passo di Computazione)

$$(q, w, u) \xrightarrow{\delta} (q', w', u')$$

Ad esempio, il passo di computazione è $(s, \triangleright, 001) \to (q, \triangleright 0, 01)$



Eseguito applicando $\delta(s, \triangleright) = (q, \triangleright, \to)$.

Definizione 2.1.5 (Time Complexity per una MdT $\mathcal{M}$ sull'input x) *$\mathcal{M}$ ha time complexity t su x se dopo esattamente t passi si raggiunge una configurazione finale.*

$$(s, \triangleright, x) \underbrace{\to \cdots \to}_{t \text{ passi}} (H, w, u)$$

Indicata in breve con $(s, \triangleright, x) \to^t (H, w, u)$.

$\mathcal{M}$ ha time complexity $f : \mathbb{N} \to \mathbb{N}$ se, $\forall x \in \Sigma^*$, $(s, \triangleright, x) \to^t (H, w, u)$ con $t \leq f(|x|)$.

La dimensione dell'input (bit length dell'input) è $|x|$. Questa è una complessità nel caso peggiore ($\leq$). Non stiamo utilizzando la notazione big-O.

2.2 Unlimited Register Machines

Una Unlimited Register Machine (URM) è una macchina di Turing con un numero illimitato di registri.

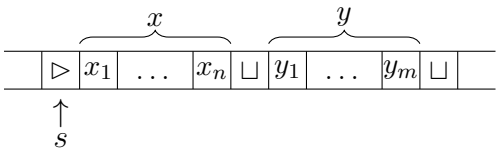
R_0	r_0
R_1	r_1
	$\dots$
R_m	r_m
	$\dots$

Ogni registro contiene un numero naturale. Quindi, il contenuto del registro R_m sarà $r_m \in \mathbb{N}$. Le operazioni possibili sono:

- **incremento** $S(i)$: $r_i := r_i + 1$
- **azzeramento** $Z(i)$: $r_i := 0$
- **trasferimento** $T(i, j)$: $r_j := r_i$, ovvero trasferisco il contenuto del registro R_i nel registro R_j
- **jump** $J(i, j, k)$: se $r_i = r_j$ allora salta all'istruzione k, altrimenti prosegue con l'istruzione successiva

Esempio Dati $x, y \in \mathbb{N}$, decidere se $x = y$.

MdT Si può utilizzare una macchina di Turing che contiene la rappresentazione binaria dei due interi, separati da un separatore.



Questa macchina richiede, nel caso peggiore, un numero quadratico di passi per terminare. La complessità è $\Theta(|x|^2)$.

URM Possiamo utilizzare una URM con x e y rispettivamente nei registri R_0 e R_1.

R_0	x
R_1	y

Alla fine, scriveremo 1 in R_0 se $x = y$, 0 altrimenti. Le istruzioni sono le seguenti:

1. $J(0,1,4)$
2. $Z(0)$
3. $J(0,0,100)$
4. $Z(0)$
5. $S(0)$

In questo caso, la complessità si può calcolare in due modi.

Definizione 2.2.1 (Time Complexity su URM)

- Uniform cost criterium *(criterio del costo uniforme): numero di istruzioni eseguite.*
- Logarithmic cost criterium *(criterio del costo logaritmico): ogni istruzione ha un costro proporzionale al numero di cifre coinvolte.*

Quindi, per questa macchina, la complessità è

- utilizzando il criterio del costo uniforme: $\Theta(1)$
- utilizzando il criterio del costo logaritmico: $\Theta(|x| + |y|)$

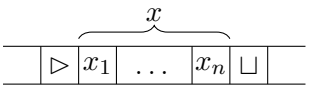
Nel secondo caso, ci si avvicina al costo per la macchina di Turing.

Mentre le macchine di Turing sono un modello di computazione sequenziale, nelle URM si ha l'istruzione *jump*. In altre parole:

- **MdT** 1 bit di informazione in ogni cella $\rightarrow$ tempo: numero di passi
- **URM** registri, un intero di lunghezza arbitraria (più bit) in ogni registro $\rightarrow$ tempo: numero di istruzioni (uniform time complexity)

Esempio Computare $x + 1$, $x \in \mathbb{N}$.

MdT Si ha una macchina di Turing che contiene x in binario.



Nel caso peggiore $x = 111\ldots1$, quindi la complessità è lineare $\Theta(n)$.

URM Si ha una URM con x nel registro R_0. È suffieciente una singola istruzione $S(0)$, quindi la complessità è $\Theta(1)$.

2.2.1 URM + Prodotto

Cambiamo il modello di computazione URM, considerando URM + prodotto. Oltre alle istruzioni $S(i)$, $Z(i)$, $T(i,j)$, e $J(i,j,k)$, aggiungiamo l'istruzione $P(i)$, che esegue l'operazione $r_i := r_i * r_i$.

Esempio di programma per URM + prodotto Dati $x, y \in \mathbb{N}$, decidere se $x = y$.

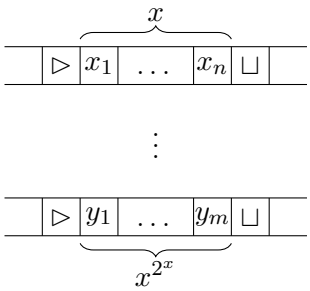
1. $J(1,2,5)$
2. $P(0)$
3. $S(2)$
4. $J(3,3,1)$

Pertendo da un input di x in R_0, x in R_1, e 0 in tutti gli altri registri. Si avrà:

R_0	x		R_0	x^2		R_0	$(x^2)^2$		R_0	$(x^4)^2$						R_0	x^{2^i}
R_1	x		R_1	x		R_1	x		R_1	x						R_1	x
R_2	0	$\rightarrow$	R_2	1	$\rightarrow$	R_2	2	$\rightarrow$	R_2	3	$\rightarrow$	$\ldots$	$\rightarrow$	R_2	2^i
R_3	0		R_3	0		R_3	0		R_3	0						R_3	0
R_4	$\vdots$		R_4	$\vdots$		R_4	$\vdots$		R_4	$\vdots$						R_4	$\vdots$

Il numero di istruzioni è lineare $\Theta(n)$.

MdT Se si eseguisse la stessa computazione su una macchina di Turing, si avrebbe



Quindi $\Omega(\log(x^{2^x})) = \Omega(2^x \log(x))$.

Questo risultato sembra contraddire la tesi di Church-Turing estesa, che afferma che tuttti i modelli **ragionevoli** di computazione sono correlati polinomialmente. Ma cosa significa *ragionevole*? Non si può avere una operazione che fa crescere "troppo" l'input (nell'esempio, il prodotto), si deve utilizzare il creiterio logaritmico.

In altre parole, se l'algoritmo utilizza operazioni che in un numero polinomiale di passi fanno crescere l'input esponenzialmente, e queste sono utilizzate un numero di volte che dipende dalla dimensione dell'input, allora si deve utilizzare un criterio logaritmico. Quando non si è sicuri della potenza delle operazioni della macchina, il costo di ogni singola operazione dev'essere proporzionale al numero di bit manipolati.

istruzione	uniform	logarithmic
$S(i)$	$\Theta(1)$	$\Theta(\log(r_i))$
$Z(i)$	$\Theta(1)$	$\Theta(1)$
$T(i,j)$	$\Theta(1)$	$\Theta(\log(r_i))$
$J(i,j,k)$	$\Theta(1)$	$\Theta(\min(\log(r_i), \log(r_j)))$
$P(i)$	$\Theta(1)$	$\Theta((\log(r_i))^2)$

Con r_i contenuto del registro i. In particolare per $P(i)$, nella moltiplicazione di un numero x per se stesso si ha $x_1, x_2, \ldots, x_n \times x_1, x_2, \ldots, x_n$. Si hanno x^n bit operazioni, quindi $O((\log(x))^2)$.

2.3 Ulteriori Definizioni

Come abbiamo visto, nei problemi di decisione si ha un input $x \in \Sigma^*$ e un output in $\{\text{yes}, \text{no}\}$. Possiamo definire un linguaggio L come l'insieme di tutte le stringhe che hanno output yes.

$$L \subseteq (\Sigma \backslash \{\sqcup\})^*$$

Un problema P è una funzione

$$P : \Sigma^* \rightarrow \{\text{yes}, \text{no}\}$$

Definizione 2.3.1 (Decidibilità di un Linguaggio da una MdT)

Una macchina di Turing $\mathcal{M}$ decide un linguaggio L

$$\Updownarrow$$

$$\forall x \in (\Sigma \backslash \{\sqcup\})^* \begin{cases} x \in L \to \mathcal{M}(x) = yes \\ x \notin L \to \mathcal{M}(x) = no \end{cases}$$

Il linguaggio L si dice ***ricorsivo****.*

Definizione 2.3.2 (Accettazione di un Linguaggio da una MdT)

Una macchina di Turing $\mathcal{M}$ accetta un linguaggio L

$$\Updownarrow$$

$$\forall x \in (\Sigma \backslash \{\sqcup\})^* \begin{cases} x \in L \to \mathcal{M}(x) = yes \\ x \notin L \to \mathcal{M}(x) \uparrow \ \textit{(non termina)} \end{cases}$$

Il linguaggio L si dice ***ricorsivamente enumerabile****.*

Teorema 2.3.1

$$L \text{ è ricorsivo } \Rightarrow \ L \text{ è ricorsivamente enumerabile}$$

Esempio Trovare un linguaggio L tale che L è ricorsivamente enumerabile ma non ricorsivo.
Nell'halting problem abbiamo

$$\mathcal{U}(\mathcal{M}; x) = \mathcal{M}(x)$$

L'halting language

$$H = \{(\text{bin}(\mathcal{M}); x) \mid \mathcal{M}(x) \downarrow\}$$

è ricorsivamente enumerabile ma non ricorsivo. Infatti, se $\mathcal{M}$ termina su x, allora $\mathcal{U}(\mathcal{M}; x) = \mathcal{M}(x) =$ yes, altrimenti $\mathcal{U}(\mathcal{M}; x) \uparrow$. Questo è un risultato qualitativo.

Esempio Sia

$$L = \{\text{bin}(\mathcal{M}) \mid \forall x \ \mathcal{M}(x) \downarrow \text{ in al massimo 100 passi}\}$$

L è ricorsivo. Infatti, la macchina $\mathcal{M}$ può eseguire al massimo 100 spostamenti a destra sul nastro. Quindi, tutte le macchine che terminano in al massimo 100 passi accettano input $\forall x \in |\Sigma|^n$ con $n \leq 100$.

Definizione 2.3.3 (Computazione di Funzioni) *Sia f una funzione $f : (\Sigma \backslash \sqcup)^* \to \Sigma^*$. Una macchina di Turing $\mathcal{M}$ computa f se*

$$\forall x \in (\Sigma \backslash \sqcup)^* \qquad \mathcal{M}(x) \downarrow \ \textit{e alla fine } f(x) \textit{ è sul nastro}$$

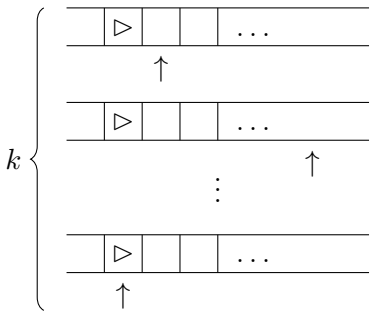
La funzione f è detta ***ricorsiva****, o* ***computabile****.*

2.4 Macchine di Turing a k-nastri

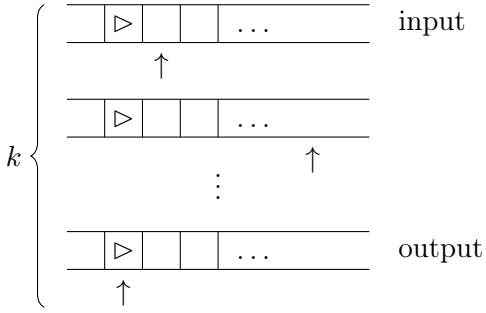
Definizione 2.4.1 (Macchina di Turing a k-nastri) *Una macchina di Turing a k-nastri è una tupla $\mathcal{M} = (K, \Sigma, \delta, s)$ con K, Σ, s definite come per una macchina di Turing, e*

$$\delta : K \times \Sigma \to (K \cup \{yes, no, halt\}) \times (\Sigma \times \{\leftarrow, \to, -\})^k$$

Una macchina di Turing a k-nastri è una macchina di Turing con un numero limitato di nastri, che possono essere utilizzati in parallelo. La funzione δ cambia perché si ha un puntatore per nastro.



Definizione 2.4.2 (Macchina di Turing a k-nastri con Input/Output) *Una macchina di Turing a k-nastri con I/O è una macchina di Turing a k-nastri con un nastro di input e un nastro di output. Il nastro di input è di sola lettura, il nastro di output è di sola scrittura.*



Definizione 2.4.3 (Configurazione e Configurazione Iniziale) *Siano $w_i, u_i \in \Sigma^*$ stringhe. Una configurazione è una tupla*

$$(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k) \to (q', w_1', u_1', w_2', u_2', \ldots, w_k', u_k')$$

Una configurazione iniziale su input x è una tupla

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$$

Capitolo 3

Complessità Spaziale

Definiamo la classe P come

$$P = \bigcup_{h \in \mathbb{N}} \text{TIME}(n^h)$$

ovvero l'unione di tutti i problemi che possono essere risolti in tempo polinomiale. La classe P ci piace così tanto perché abbiamo la tesi di Church-Turing estesa. Questa classe è **invariante** rispetto alla scelta del modello di computazione. Possiamo definire la classe EXP

$$\text{EXP} = \bigcup_{h \in \mathbb{N}} \text{TIME}(2^{n^h})$$

La classe $\mathbb{L}$, PSPACE, e EXPSPACE

$$\begin{aligned} \mathbb{L} &= \text{SPACE}(\log n) \\ \text{PSPACE} &= \bigcup_{h \in \mathbb{N}} \text{SPACE}(n^h) \\ \text{EXPSPACE} &= \bigcup_{h \in \mathbb{N}} \text{SPACE}(2^{n^h}) \end{aligned}$$

Proprietà 3.0.1

$$TIME(f(n)) \subseteq SPACE(f(n))$$

RAM, random access machine. (1)