# 1 Elementi di complessità

## 1.1 Misure di complessità deterministiche

**Definizione 1.1** (Macchina *"acchiappanastri"*)
MdT a $k$ nastri: $M = (Q, \Sigma, \delta, q_0)$, con:

- $\#, \triangleright \in \Sigma,\ L, R, - \notin \Sigma$
- $\text{SI}, \text{NO} \notin Q$
- $\delta : Q \times \Sigma^k \to Q \cup \{\text{SI}, \text{NO}\} \times (\Sigma \times \{L, R, -\})^k$, funzione soggetta alle stesse condizioni della definizione ad un nastro.

La funzione di transizione cambia perciò lo stato globale e scrive/muove su tutti i nastri:

$$\delta(q, \sigma_1, \dots, \sigma_k) = (q', (\sigma'_1, D_1), \dots, (\sigma'_k, D_k))$$

Ed una configurazione della MdT ha la forma:

$$(q, u_1\sigma_1 v_1, \dots, u_k\sigma_k v_k)$$

**Definizione 1.2** (misura del tempo come numero di passi)
$t$ è il tempo richiesto da una MdT $M$ a $k$ nastri per decidere il caso $x \in I$ se

$$(q_0, \trianglerighteq x, \trianglerighteq, \dots \trianglerighteq) \to^t (H, w_1, \dots, w_k) \qquad H \in \text{SI}, \text{NO}$$

**Definizione 1.3** (misura del tempo *in funzione della taglia* di $x$)
$M$ **decide** $I$ **in tempo deterministico** $f$ se per ogni dato di ingresso $x$ il tempo $t$ richiesto da $M$ per decidere $x$ è minore di/è uguale a $f(|x|)$, dove con $|x|$ si indica la **taglia** del dato $x$.

**Definizione 1.4** (Classe $TIME(f)$)

$$TIME(f) = \{I \mid \exists M \text{ che decide } I \text{ in tempo deterministico } f\}$$

**Teorema 1.1.1** (Riduzione del numero di nastri)
Data una macchina a $k$ nastri $M$ che decide $I$ in tempo deterministico $f$, allora esiste una macchina $M'$ ad un solo nastro che decide $I$ in tempo deterministico $\mathcal{O}(f^2)$

*Traccia della dimostrazione.* Costruiamo $M'$ che simula $M$, rappresentando ogni configurazione come segue, usando le parentesi angolate per separare i vari nastri.

$$(q, \triangleright w_1\sigma_1 u_1, \dots, \triangleright w_k\sigma_k u_k) \mapsto (q, \triangleright\langle w_1\bar{\sigma}_1 u_1\rangle \dots \langle w_k\bar{\sigma}_k u_k\rangle)$$

I $\bar{\sigma}_i$ sono simboli speciali che sono sostituiti ai simboli $\sigma_i$ quando il cursore è su di essi.

- Anzitutto $M'$ genera la configurazione iniziale di $M$: $(q_0, \triangleright x, \triangleright, \dots, \triangleright) \mapsto (q_0, \triangleright\langle x\rangle\left[\langle\rangle\right]^{k-1})$
  Per fare ciò bastano $2k + \#\Sigma$ nuovi stati ed un numero di passi dell'ordine di $|x|$;
  **Idea** Si deve spostare $x$ di un posto, il che comporta al più $\#\Sigma$ nuovi stati e $\mathcal{O}(|x|)$ passi. Inoltre Si devono scrivere le parentesi, quindi ci vogliono altri $\sim 2k$ stati.
- Per simulare un passo la macchina deve scorrere il dato d'ingresso (da sinistra a destra e viceversa due volte) per determinare quali simboli sono da aggiornare e conseguentemente aggiornarli.
  **Osservazione cruciale**: non posso toccare un numero di caselle maggiore del numero di passi che compio.
  Di conseguenza, la lunghezza del nastro è al più $K = k(f(|x|) + 2) + 1$ (contando parentesi e respingente). Di conseguenza il numero di passi di $M'$ è $\mathcal{O}(f(|x|)^2)$

□

**Stabilità delle MdT** Il teorema appena dimostrato mostra che miglioramenti che siano accettabili *algoritmicamente* (e.g. aggiungere nastri) non solo non cambiano le funzioni calcolate, ma **non modificano il tempo richiesto se non polinomialmente**.

**Corollario:** Le macchine parallele sono polinomialmente più veloci di quelle sequenziali.

Abbiamo inoltre osservato che **non si può usare più spazio che tempo**. (Osservazione 1)

**Teorema 1.1.2** (Accelerazione lineare MdT)

$$I \in TIME(f) \implies \forall \epsilon \in \mathbb{R}^+.\ I \in TIME(\epsilon \cdot f(n) + n + 2)$$

.....

**Idea:** Se si considera una MdT $M'$ in cui codifichiamo $m$-ple di caratteri di $M$ con un solo carattere, allora la computazione sarà svolta più velocemente da $M'$ (e.g. esempio delle barrette [DEG 3.2.8]).

.....

*Idea della dimostrazione.* La dimostrazione è lunga e piena di dettagli insidiosi;

Ciò che invece dimostriamo è che, codificando come nell'**Idea**, possiamo determinare il numero $m$ di simboli di $M$ da "compattare" in un unico simbolo di $M'$ in funzione del solo $\epsilon$.

Gli stati di $M'$ hanno la forma $(q, \sigma_{i_1} \dots \sigma_{i_m}, k)$, in modo da rappresentare il fatto che $M$ è nello stato $q$, con il cursore sul $k$-esimo carattere della stringa $\sigma_{i_1} \dots \sigma_{i_m}$.

Allora $m$ passi di $M$ possono essere simulate da $M'$ in al più **sei passi**:

- 4 passi per esaminare i caratteri (i.e. blocchi di $m$ caratteri) adiacenti al corrente (`LRRL`)
- 2 passi per modificare il corrente ed uno tra i blocchi a destra e a sinistra.

Di conseguenza $M'$ farà $|x| + 2 + 6 \cdot \left\lfloor \frac{f(|x|)}{m} \right\rfloor$ passi ($|x| + 2$ per *condensare* il dato in ingresso)

Infine, basta sostituire $m = \left\lfloor \frac{6}{\epsilon} \right\rfloor$ e si ottiene $I \in TIME(\epsilon \cdot \left( \frac{f(n)}{\not{6}} \right) \cdot \not{6} + n + 2)$ □

**Definizione 1.5**

Definiamo la classe $\mathcal{P}$ come segue:

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

Consideriamo solo gli $n^k$ poiché

$$c_1 x^k + c_2 x^{k-1} + \ldots + c_k$$

Gli elementi della forma $c_i x^i$, per $i < k$ spariscono asintoticamente, mentre il $c_1$ può essere eliminato grazie al teorema di accelerazione lineare.

**Cambiare modello è operazione polinomiale** Ma bisogna stare attenti, e.g. se in una Random Access Machine conto le moltiplicazioni anziché somme ho sbagliato!

**Definizione 1.6** (MdT a $k$ nastri di tipo I/O)

$$\delta(q, \sigma_1, \ldots, \sigma_k) = (q', (\sigma'_1, D_1), \ldots, (\sigma'_k, D_k))$$

Valgono le seguenti:

- Il primo nastro è di **sola lettura**: $\sigma'_1 = \sigma_1$
- L'ultimo nastro è di **sola scrittura**: $D_k = R$, oppure $D_k = - \implies \sigma'_k = \sigma_k$
- La macchina **visita al massimo una cella bianca** a destra del dato d'ingresso (che ipotizziamo non contenere il carattere bianco): $\sigma_1 = \# \implies D_1 \in \{L, -\}$

**Proposizione 1.1.1**
Per ogni MdT a $k$ nastri $M$ che decide $I$ in tempo deterministico $f$ esiste una MdT a $k+2$ nastri $M'$ di tipo I/O che decide $I$ in tempo deterministico $c \cdot f$, per qualche costante $c$.

*Dimostrazione.* La macchina copia l'input dal primo nastro di sola lettura nel secondo, in $|x|+1$ passi; poi opera come $M$, e infine copia il risultato dal penultimo nastro all'ultimo (di sola scrittura) al più in $f(|x|)$ passi.

La macchina $M'$ ha impiegato quindi al più un numero di passi pari a $2f(|x|)+|x|+1$. Determinare ora la costante $c$ è immediato.

□

Ancora una volta abbiamo mostrato che modifiche algoritmicamente "ragionevoli" alle MdT non ne alterano il potere espressivo.

**Complessità in spazio deterministico** Definiamo lo spazio necessario ad una computazione come il numero totale delle caselle toccate **solamente** sui nastri di lavoro (non su input e output).

Modifichiamo la definizione di configurazione, in modo che tenga traccia di **tutte** le caselle toccate, i.e. anche quelle divenute bianche. Formalmente dovremmo aggiungere un simbolo $\triangleleft$ da usare come *delimitatore* della parte scritta.

**Definizione 1.7** (spazio deterministico)
Sia $M$ una MdT a $k$ nastri di tipo $I/O$ tale che

$$\forall x.(q_0, \trianglerighteq x, \trianglerighteq, \ldots, \trianglerighteq) \to (H, w_1, \ldots, w_k) \text{ con } H \in \{\text{SI}, \text{NO}\}$$

lo *spazio richiesto* da $M$ per decidere è

$$\sum_{i=2}^{k-1} |w_i|$$

Dove $|w_i|$ è la lunghezza della stringa $w_i$, ossia esattamente il numero di caratteri toccati dalla computazione sul nastro $i$-esimo.

Diciamo che, se $M$ decide $I$ in spazio deterministico $f(n)$, allora $I \in SPACE(f(n))$.

Alcuni autori definiscono lo spazio richiesto come $\max_{2 \le i \le k-1} |w_i|$, ma l'unica differenza dalla nostra definizione è un fattore $k$, che sparisce quando si considerano gli ordini di grandezza.

**Teorema 1.1.3** (Compressione lineare dello spazio)

$$I \in SPACE(f(n)) \implies \forall \epsilon \in \mathbb{R}^+. I \in SPACE(2 + \epsilon \cdot f(n))$$

**Definizione 1.8**

$$PSPACE = \bigcup_{k \ge 1} SPACE(n^k)$$

$$LOGSPACE = \bigcup_{k \ge 1} SPACE(k \cdot \log n)$$

**Teorema 1.1.4**

$$LOGSPACE \subsetneq PSPACE$$

Non dimostrato

**Teorema 1.1.5**

$$LOGSPACE \subseteq \mathcal{P}$$

. . . . . .

*Dimostrazione.* Svolgiamo la dimostrazione per macchine a 3 nastri.
Poiché $I \in LOGSPACE$, c'è una MdT $M$ che decide ogni sua istanza $x \in I$ in $\mathcal{O}(\log |x|)$ spazio deterministico, basta notare che $M$ può attraversare al più:

$$\mathcal{O}(|x| \cdot log|x| \cdot \#Q \cdot \#\Sigma^{\log |x|})$$

configurazioni non terminali diverse.

- $\#Q$ è il numero di stati
- $|n| \cdot \log |x|$ sono le posizioni possibili del cursore nei nastri di input e lavoro;
- $\#\Sigma^{\log |x|}$ è il numero di possibili stringhe che posso scrivere nel nastro di lavoro

$$\begin{aligned} &\mathcal{O}(|x| \cdot log|x| \cdot \#Q \cdot \#\Sigma^{\log |x|}) \\ =&\mathcal{O}(|x| \cdot log|x| \cdot \#\Sigma^{\log |x|}) \\ =&\mathcal{O}(|x|^k) \end{aligned}$$

$n = |x|$, un tale $k$ deve soddisfare le seguenti disuguaglianze:

$$n \cdot \log n \cdot \#\Sigma^{\log n} \leq n^k$$

$$\log(n) + \log(\log(n)) + \log(n) \cdot \log(\#\Sigma) \leq k \cdot \log n$$

$$1 + \frac{\log(\log(n))}{\log n} + \log(\#\Sigma) \leq k$$

$$1 + \log(\log(n) - n) + \log(\#\Sigma) \leq k$$

Una computazione terminante non può passare per la stessa configurazione due volte, se no entrerebbe in un ciclo, quindi una computazione ha almeno $\mathcal{O}(|x|^k)$ passi per qualche $k$. □

## 1.2 Misure di complessità non deterministiche

**Definizione 1.9** (MdT non deterministica)
$N = (Q, \Sigma, \Delta, q_0)$

- $Q, \sigma, q_0$ come prima
- $\Delta \subseteq (Q \times \Sigma) \times ((Q \cup \{\text{SI}, \text{NO}\}) \times \Sigma \times \{L, R, -\})$ relazione – non più funzione

**Definizione 1.10**
$N$ MdT *non deterministica* decide $I \iff N(x) \to^\star (\text{SI}, w)$

Per accettare basta mostrare una computazione che accetta, mentre per rifiutare bisogna mostrare che **tutte** le computazioni non accettano

**Definizione 1.11**
$N$ decide $I$ in tempo *non deterministico* $f$ se

$$\forall x \in I. \exists w : N(x) \to^t (\text{SI}, w), \quad t \leq f(|x|)$$

O, in una macchina n.d. a $k$ nastri (I/O): la somma degli $|w_i|$ deve essere minore di $f(|x|)$, come nella definizione 1.7.

NPSPACE e NPTIME $= \mathcal{NP}$ sono definiti come ovvio, a partire da NTIME e NSPACE, anch'essi definiti in modo analogo a TIME e SPACE.

**Teorema 1.2.1** (Di Savitch)

$$NPSPACE = PSPACE$$

**Teorema 1.2.2**

$$I \in NTIME(f) \implies \exists c \,.\, I \in TIME(c^f)$$

i.e. $NTIME(f(n)) \subseteq TIME(c^{f(n)})$

*Dimostrazione.* Sia $d$ il grado di non-determinismo di $N$, cioè:

$$d = \max\{\text{Grado}(q, \sigma), q \in Q, \sigma \in \Sigma\}$$

Dove $\text{Grado}(q, \sigma) = \#\{(q', \sigma', D) \mid ((q, \sigma), (q', \sigma', D)) \in \Delta\}$, ossia il numero di transizioni che partono da $(q, \sigma)$. Supponiamo, per semplicità di trattazione, che nel seguito la macchina abbia sempre $d$ scelte; Per ogni coppia $(q, \sigma)$ ordiniamo lessicograficamente $\Delta(q, \sigma)$. Ogni computazione di $N$ è una sequenza di scelte. Se tale sequenza è lunga $t$, possiamo vederla come una successione lunga $t$ di numeri naturali $\in [0..d-1]$, rappresentando con 0 la prima scelta. La macchina $M$ riproduce la sequenza di scelte $(c_1 \dots c_t)$ tenendo il numero $t'$ in base $d$ che la rappresenta sul nastro aggiuntivo; Se la simulazione arriva in uno stato SI termina accettando, se no la macchina $M$ genera la prossima successione $t' + 1$. Se tutte le computazioni terminano in stato NO, la macchina $M$ termina rifiutando. Il numero di scelte è $\mathcal{O}(d^{f(n)+1})$, quindi basta sostituire $c$ con $d$ □

**Commesso viaggiatore**
Trovare il cammino di costo minimo che attraversa tutti i nodi una ed una sola volta su grafi **pesati non orientati**

. . . . . .

Per ora si conoscono solo **algoritmi polinomiali n.d.** o **esponenziali deterministici**.

. . . . . .

Per trasformare questo problema in uno di decisione, lo possiamo riscrivere come:
**Esiste un cammino** [...] di costo $\leq B$ su G [...]?

- Soluzione bruteforce: genera tutte le permutazioni di interi da 1 ad $n$ ($= |N|$) — costo $\mathcal{O}(n!)$. Per ogni permutazione si deve controllare se il costo è minore di $B$, e questo comporta tempo $\mathcal{O}(n^3)$: si deve accedere $\mathcal{O}(n)$ volte ad ognuna delle $\mathcal{O}(n^2)$ coppie $(i, j)$ per ottenere la distanza $d(i, j)$
- Soluzione non deterministica: Genera le permutazioni **implicitamente** e scrive una stringa lunga $n$ in $O(n)$ passi. La macchina n.d. poi verifica se tale permutazione è un cammino accettabile, sempre in $\mathcal{O}(n^3)$

**Funzioni Appropriate** Una funzione $f$ è appropriata/costruibile/onesta se:

- $f$ è monotona crescente
- $\exists M$ a $k$ nastri tale che
$$M(x) \to^* \star^{f(|x|)} \qquad \star \notin \Sigma \text{ simbolo speciale}$$
in tempo $O(f(|x|) + |x|)$ e spazio $O(f(|x|))$

La definizione sembra mordersi la coda; seguono alcuni criteri:

- $k, n, n^k, k^n, n!, \lfloor \log n \rfloor, \lfloor \sqrt{n} \rfloor$ sono funzioni appropriate
- $f, g$ appropriate $\implies f \cdot g,\ f + g$ appropriate
- $f \circ g$ appropriata, $f^g$ appropriata.

**Teorema 1.2.3** (gerarchia di tempo e spazio)
Sia $f$ appropriata:

- $TIME(f(n)) \subsetneq TIME(f(2n+1)^3)$
- $SPACE(f(n)) \subsetneq SPACE\big(f(n) \cdot \log f(n)\big)$

. . . . . .

*Dimostrazione.* Omessa, ma notiamo che il primo punto si dimostra mostrando che:

$$\{x \mid \varphi_x(x) \text{ converge entro } f(|x|) \text{ passi}\}$$

Appartiene a $TIME(f^3)$ e non a $TIME(f)$. La dimostrazione del secondo punto è analoga. □

Naturalmente esiste un teorema analogo per le misure non deterministiche.

**Definizione 1.12**

$$EXP = \bigcup_{k \geq 1} TIME(2^{n^k})$$

**Proposizione 1.2.1**

$$\mathcal{P} \subsetneq EXP$$

.....

*Dimostrazione.* L'inclusione (1) è ovvia poiché $2^n$ cresce più velocemente di ogni polinomio.
Per mostrare che questa è propria, si noti:

$$\mathcal{P} \underset{(1)}{\subseteq} TIME(2^n) \underset{T.ger}{\subsetneq} TIME\left(2^{(2n+1)^3}\right) \underset{6n+3 \leq n^2}{\subseteq} TIME\left(2^{n^2}\right)$$

□

.....

Inoltre, dato che $NTIME(f(n)) \subseteq TIME\left(c^{f(n)}\right)$ per qualche $c \implies \mathcal{NP} \subseteq EXP$.

**Teorema 1.2.4**
Siano $f$ una funzione di misura appropriata e $k$ una costante; allora:

- $SPACE(f(n)) \subseteq NSPACE(f(n))$
- $TIME(f(n)) \subseteq NTIME(f(n))$
- $NSPACE(f(n)) \subseteq TIME(k^{\log n + f(n)})$

**Teorema 1.2.5** (Sempre peggio)
Per ogni funzione calcolabile totale $g$ esiste un problema $I \in TIME(f(n))$ e $I \notin TIME(g(n))$, con $f(n) \geq g(n)$ definitivamente

.....

I.e. **la gerarchia non è superiormente limitata**

**Teorema 1.2.6** (Di accelerazione, Blum)
Per ogni funzione calcolabile totale $h$ esiste un problema $I$ tale che, per ogni algoritmo $M$ che decide $I$ in tempo $f$ esiste una macchina $M'$ che decide $I$ in tempo $f'$, con

$$f(n) > h(f'(n)) \text{ definitivamente}$$

.....

**Intuitivamente** Ci sono programmi che sono più veloci su una macchina universale vecchia e lenta che su una macchina universale nuova e veloce.

**Intuizione: Blum** Esistono problemi per cui non esiste un algoritmo ottimo.

.....

Attenzione: Il problema che si costruisce a partire da $h$ è **artificiale** (non è un problema incontrato prima) e, sebbene si sappia che una successione di algoritmi sempre migliori esista, non sappiamo come si costruiscono l'uno dall'altro!

**Teorema 1.2.7** (Della lacuna, Borodin)
Esiste $f$ calcolabile tale che $TIME(f(n)) = TIME(2^{f(n)})$

. . . . . .
Una formulazione più fine di questo teorema richiede che $f$ sia monotona

**Definizione 1.13** (Assiomi di Blum)
Una funzione $\phi$ è una misura di complessità se prende come input una funzione $\psi$ ed il suo dato di ingresso $x$ e restituisce un naturale, e valgono i seguenti assiomi

A1. $\phi(\psi, x)$ è definita sse $\psi(x)$ lo è (ossia hanno lo stesso domino)

A2. Per ogni $\psi, x, k$, l'uguaglianza:
$$\phi(\psi, x) = k$$
è decidibile. In altre parole, l'insieme $\{(\psi, x, k) \in (\mathbb{N} \to \mathbb{N}) \times \mathbb{N}^2 \mid \phi(\psi, x) = k\}$ è ricorsivo.

## 1.3 $\mathcal{P}$, $\mathcal{N}P$

**Tesi di Cook-Karp** Un problema è trattabile esattamente quando ammette un algoritmo polinomiale

### 1.3.1 SAT

Il problema SAT consiste nel decidere se, data un'espressione booleana $B$ in **forma normale congiuntiva** (*and di or*), esiste un assegnamento $\mathcal{V}$ tale che $\mathcal{V} \models B$ (i.e. $\mathcal{V}$ soddisfa $B$).

SAT è in NP perché, banalmente, basta lasciar scegliere al non-determinismo l'assegnamento corretto e controllare se $B$ è soddisfatta, in tempo polinomiale.

### 1.3.2 HAM

HAM consiste nel decidere se un grafo **orientato** ha un cammino hamiltoniano, i.e. esiste una permutazione dei nodi tale che esista un arco tra ogni nodo ed il successivo nella permutazione.
(esiste un cammino che tocca tutti i nodi una ed una sola volta)

### 1.3.3 HAM $\leq_L$ SAT

*Dimostrazione.* Dobbiamo costruire una funzione $f \in LOGSPACE$ tale che $G$ ha un cammino hamiltoniano se e solo se $f(G)$ è soddisfacibile.

Prima scriviamo un'espressione booleana in forma congiuntiva che *rappresenta* i cammini hamiltoniani; poi mostriamo che tale espressione booleana è soddisfacibile se e solo se esiste un cammino hamiltoniano nel grafo $G$. Infine mostriamo che la funzione è logaritmica in spazio.

Sia $n = |V|$, allora $f(G)$ ha $n^2$ variabili booleane: $x_{ij}$, con $i, j$ comprese tra 1 ed $n$. Costruiamo i congiunti che rappresentano le proprietà dei cammini hamiltoniani:

1. Lo stesso nodo $j$ non può comparire due volte nella permutazione/cammino: $\neg(x_{ij} \wedge x_{kj})$, con $i \neq k$. Applicando De Morgan (dobbiamo portare in FNC):

$$(\neg x_{ij} \vee \neg x_{kj}) \quad i \neq k$$

2. Ogni nodo deve apparire in un cammino:

$$(x_{1j} \vee ... \vee x_{nj}) \quad 1 \leq j \leq n$$

3. Qualche nodo deve essere l'$i$-esimo del cammino:

$$(x_{i1} \vee ... \vee x_{in}) \quad 1 \leq i \leq n$$

4. Due nodi non possono essere contemporaneamente l'$i$-esimo:

$$(\neg x_{ij} \vee \neg x_{ik}) \quad j \neq k$$

5. Se $(i, j)$ non è un arco di $G$, $i$ e $j$ non possono apparire in sequenza nel cammino:

$$(\neg x_{ki} \vee \neg x_{k+1,j}) \quad \forall k.1 \leq k \leq n-1 \quad \forall (i,j) \notin A$$

Vediamo adesso che $G$ ha un cammino hamiltoniano se $\mathcal{V} \models f(G)$.
Si noti che se $\mathcal{V} \models f(G)$, allora per ogni $j$ esiste unico $i$ tale che $\mathcal{V}(x_{ij}) = tt$, altrimenti (1) e (2) non potrebbero essere soddisfatte entrambe. Allo stesso modo per ogni $i$ esiste un solo $j$ tale che $\mathcal{V}(x_{ij}) = tt$.
Quindi l'espressione booleana rappresenta una permutazione dei nodi, e per (5) abbiamo che tale permutazione è un cammino.

Vediamo ora l'inverso: sia $H = (\Pi(1), \ldots, \Pi(n))$ un cammino hamiltoniano; allora è immediato verificare che:

$$\mathcal{V}(x_{ij}) = \begin{cases} tt & \Pi(j) = i \\ ff & \Pi(j) \neq i \end{cases}$$

soddisfa $f(G)$.

Infine, per vedere che la trasformazione è logaritmica in spazio, basta notare che le uniche informazioni che dobbiamo salvare nei nastri di lavoro sono $n$ e tre contatori $i, j, k$ che vanno da 1 ad $n$, tutti rappresentati in binario, quindi il costo in spazio è $\mathcal{O}(\log n)$ □

### 1.3.4 Problema CRICCA

Consiste nel decidere se in un grafo (non orientato) esiste un sottografo completo (cricca) di ordine $k$.

### 1.3.5 SAT $\leq_L$ CRICCA

*Dimostrazione.* Data un'espressione booleana $B = \bigwedge_{1 \leq k \leq n} C_k$, costruisci il grafo $f(B) = (N, A)$ così:

i) $N$ è l'insieme delle occorrenze dei letterali in $B$

ii) $A$ è l'insieme $\{(i,j) \mid i \in C_k \implies (j \notin C_k \wedge i \neq \neg j)\}$
Ossia esiste un arco tra:

- ogni coppia di nodi che **non fanno parte dello stesso congiunto** e
- che **non sono l'uno il negato dell'altro**

Essendo $B$ in forma normale congiuntiva, è soddisfatta se e solo se almeno un letterale in ogni congiunto è vero. Quindi, $B$ è soddisfacibile se e solo se $f(B)$ ha una cricca di ordine pari al numero di congiunti. □

**Definizione 1.14** (Circuito booleano)
Un circuito booleano è un grafo diretto aciclico in cui i nodi sono detti porte, ed hanno una **sorta** $s(i) \in \{tt, ff, \neg, \vee, \wedge\} \cup X$, dove $X$ è un insieme di variabili. Alcune porte sono **ingressi**, ed hanno sorta $tt, ff$ o in $X$

Il valore di verità di una determinata porta $i$ è così definito:

$$\begin{aligned}
[\![i]\!]_{\mathcal{V}} &= tt && \text{se } s(i) = tt \\
&= ff && \text{se } s(i) = ff \\
&= \mathcal{V}(x) && \text{se } s(i) = x \\
&= \text{not } [\![j]\!]_{\mathcal{V}} && \text{se } s(i) = \neg \text{ e } (j,i) \in A \\
&= [\![j]\!]_{\mathcal{V}} \text{ or } [\![k]\!]_{\mathcal{V}} && \text{se } s(i) = \vee \text{ e } (j,i),(k,i) \in A \\
&= [\![j]\!]_{\mathcal{V}} \text{ and } [\![k]\!]_{\mathcal{V}} && \text{se } s(i) = \wedge \text{ e } (j,i),(k,i) \in A
\end{aligned}$$

Il valore del circuito è quello della **porta di uscita** (che assumiamo essere unica).

### 1.3.6 Circuit SAT

Il problema CIRCUIT SAT consiste nel decidere se esiste un assegnamento $\mathcal{V}$ tale che $\mathcal{V}(C) = tt$

**Proposizione 1.3.1**
Circuit SAT $\in \mathcal{NP}$: banale, dato l'assegnamento ci metto tempo polinomiale a certificare la soddisfacibilità.

### 1.3.7 Circuit Value

Consiste nel calcolare il valore di un circuito senza variabili, ossia in cui tutti gli ingressi sono della sorta *tt, ff*. Tale problema è ovviamente in $\mathcal{P}$.

**Proposizione 1.3.2**
CIRCUIT VALUE $\leq_L$ CIRCUIT SAT

Questo è vero poiché **ogni caso particolare di un problema si riduce al problema generale** attraverso la funzione identità.

## 1.4 Problemi completi per $\mathcal{P}$, $\mathcal{NP}$

**Proposizione 1.4.1**
CIRCUIT-SAT $\leq_L$ SAT

*Dimostrazione.* Dato il circuito $C = (N, A)$ con variabili in $X$, cerco una riduzione $f \in LOGSPACE$ tale che $f(C)$ sia soddisfacibile $\iff$ $C$ lo è.

- Le variabili $X'$ sono $X \cup \{x_g \mid g \in N\}$, ossia *si aggiunge una variabile per ogni porta*
- Per ogni porta di $g$ si costruiscono i congiunti di $f(C)$ come segue:
  - Se $g$ è porta di uscita, allora si genera il congiunto $x_g$
  - $s(g) = tt/ff$ si genera $x_g/\neg x_g$
  - $s(g) = x \in X$ si genera il congiunto equivalente a $x_g \iff x$:

$$\begin{aligned}(x_g \iff x) &\equiv (x_g \implies x) \wedge (x \implies x_g) \\ &\equiv (\neg x_g \vee x) \wedge (\neg x \vee x_g)\end{aligned}$$

  - Se $s(g) = \neg$ e $(h, g) \in A$, ossia per ogni arco da $h$ a $g$: si genera il congiunto equivalente a $x_g \iff \neg x_h$, ossia $(\neg x_g \vee \neg x_h) \wedge (x_h \vee x_g)$
  - Se $s(g) = \wedge$, e $(h, g), (k, g) \in A$ allora genera $x_g \iff (x_h \wedge x_k)$:

$$\begin{aligned}&(\neg x_g \vee (x_h \wedge x_k)) \wedge (\neg(x_h \wedge x_k) \vee x_g) \\ &(\neg x_g \vee x_h) \wedge (\neg x_g \vee x_k) \wedge (\neg x_h \vee \neg x_k \vee x_g)\end{aligned}$$

  - Se $s(g) = \vee$, e $(h, g), (k, g) \in A$ allora genera $x_g \iff (x_h \vee x_k)$:

$$\begin{aligned}&(\neg x_g \vee (x_h \vee x_k)) \wedge (\neg(x_h \vee x_k) \vee x_g) \\ &(\neg x_g \vee x_h \vee x_k) \wedge ((\neg x_h \wedge \neg x_k) \vee x_g) \\ &(\neg x_g \vee x_h \vee x_k) \wedge (\neg x_h \vee x_g) \wedge (\neg x_k \vee x_g)\end{aligned}$$

La trasformazione richiede spazio logaritmico perché mi servono solo tre contatori (per $x_g$, $x_h$, $x_k$), che rappresento in binario. □

. . . . . .

**Corollario**

- CIRCUIT VALUE $\leq_L$ SAT
- CIRCUIT VALUE $\leq_L$ CRICCA

### 1.4.1 $\mathcal{P}$-completezza di CIRCUIT-VALUE

**Definizione 1.15** (Tabella di computazione)
La *tabella di computazione* $T_M$ di una MdT $M$ deterministica ad un nastro è una matrice quadrata in cui $T(i,j)$ è il $j$-esimo carattere nel nastro all'$i$-esimo passo di computazione:

| | 1 | 2 | 3 | 4 | ... | |
|---|---|---|---|---|---|---|
| 1 | $\triangleright$ | $a_{q_0}$ | $b$ | $b$ | ... | # |
| 2 | $\triangleright$ | $\triangleright$ | $b_{q_a}$ | $b$ | ... | # |
| 3 | $\triangleright$ | $\triangleright$ | $b$ | $b_{q_a}$ | ... | # |
| 4 | $\triangleright$ | $\triangleright$ | $b$ | $b$ | ... | # |
| $\vdots$ | | | | | | |

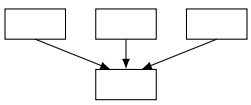
Con alcune "standardizzazioni":

1. $M$ si arresta prima di $\underbrace{|x|^k}_{n} - 2$ passi $\implies$ la tabella è $n \times n$
2. $M$ non tocca mai il $\triangleright$ in colonna 1
3. $M$ non tocca mai il # in colonna $n$
4. $T$ contiene tanti # a destra quanti ne servono
5. Lo stato e la posizione del cursore sono così codificati: Si aggiungono dei nuovi simboli che rappresentano gli stati, e nella riga:

   | i | | | | | | |
   |---|---|---|---|---|---|---|
   | | $\triangleright$ | $\sigma^1$ | $\sigma_q^2$ | $\sigma^3$ | ... | # |

   Lo stato è $q$ ed il cursore è nella posizione alla sinistra del simbolo arricchito, in questo caso su $\sigma^1$
6. Se $\sigma_{\text{SI}}$ o $\sigma_{\text{NO}}$ si trova su $T(i,j)$, con $i < n$ e $j > 2$, viene spostato in $T(k,2)$ con altri passi.
7. Si riempie con righe uguali alla $k$-esima finché serve.

**Proposizione 1.4.2**
$T(i,j)$ dipende solo dai valori di $T(i-1,j-1)$, $T(i-1,j)$, $T(i-1,j+1)$, in base alla funzione $\delta$.



Se il cursore è in posizione $i$, può spostarsi su $i-1$ o $i+1$, ed il carattere in $i$ può essere modificato.

**Teorema 1.4.1** (CIRCUIT VALUE è $\mathcal{P}$-completo)

. . . . . .

*Dimostrazione.* Sappiamo che CIRCUIT-VALUE $\in \mathcal{P}$. Prendiamo quindi qualunque $I \in \mathcal{P}$ e mostriamo che esiste una $f \in LOGSPACE$ che lo trasforma in CIRCUIT-VALUE. Sia $M$ una MdT che decide $I$ in $n^k$, e $T$ la sua tabella di computazione.

- Costruiamo un circuito a partire dalla tabella di computazione su $x$. Per far ciò **codifichiamo ogni simbolo** $\rho \in \Sigma'$ (dove $\Sigma'$ è l'alfabeto con i simboli arricchiti) **con una stringa di bit** $(S_1, \ldots, S_m) \in (tt, ff)^m$. In questo modo ogni riga della tabella è una stringa di bit lunga $n^k \cdot m$. Rappresentiamo quindi ognuno degli $m$ bit della codifica del simbolo $T(i,j)$ come $S_{i,j,m}$.
- Adesso sappiamo che il bit $S_{i,j,l}$ dipende solo dalle tre $m$-ple di bit che rappresentano i simboli come in figura 1.4.2, quindi ci sono $m$ funzioni booleane:

$$\begin{aligned} S_{i,j,l} = F_l(&S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}, \\ &S_{i-1,j,1}, \ldots, S_{i-1,j,m}, \\ &S_{i-1,j+1,1}, \ldots, S_{i-1,j+1,m}) \end{aligned}$$

  Per ogni funzione booleana esiste un circuito booleano $C_I$ composto da dei sottocircuiti $\overline{C}$ con $3m$ ingressi ed $m$ uscite, "incastrati" l'uno sull'altro come da figura 1.4.2.

  La dimostrazione di $C_I = f(x)$ è true se e solo se $x \in I$ procede per induzione sul numero dei passi della computazione:

  1. Caso base $i = 1$ banale
  2. Il valore per $i-1$ è ben calcolato e dipende solo dai tre precedenti. Il valore per $i$ dipende da tre valori calcolati in $i-1$, in modo che dipende esclusivamente dalla funzione di transizione, perciò è immediato dedurre che $\overline{C}_{n^k,2} = tt, \ldots, tt$ se e solo se $T(n^k, 2) = \sigma_{\text{SI}}$, ovvero $x \in I \iff f(x) = tt, \ldots, tt$.

  ◦ Adesso basta vedere che $f \in LOGSPACE$:

  – Vanno costruite le porte di ingresso: si esamina $x$ e si conta fino a $|x|^k$, ricordando $k$ in base 2, scrivendo la codifica di $x$ e di # finché serve
  – Gli elementi della prima ed ultima tabella sono circuiti costanti
  – Vengono create $(x^k - 1) \cdot (x^k - 2)$ copie del circuito $\overline{C}$, che dipende solo da M ed ha costo costante. A ciascuna copia associamo gli appropriati indici; tali indici sono tutti minori di $|x|^k$, ed averli rappresentati in binario ci consente di manipolarli in $\mathcal{O}(\log |x|^k) = \mathcal{O}(\log |x|)$

□

. . . . . .

**Corollario** MONOTONE CIRCUIT VALUE è $\mathcal{P}$-completo

. . . . . .

MONOTONE CV è una variante di CV che **non utilizza la negazione**. I circuiti senza il $\neg$ sono meno espressivi di quelli con, ma sono altrettanto difficili da valutare.

. . . . . .

*Dimostrazione.* Mostriamo che CV $\leq_L$ MONOTONE CV, cioè trasformiamo un circuito qualsiasi in uno monotono equisoddisfacibile. Questo si può fare applicando le regole di DeMorgan dall'alto verso il basso, trasformando alla fine le negazioni in dei *ff* e aggiungendo porte. Questo si fa in logspace perché basta visitare una sola volta le porte, rappresentate come $(i,j)$ dove $i,j$ sono gli indici di "livello e colonna", rappresentati in binario. □

### 1.4.2 $\mathcal{NP}$-completezza di SAT

**Teorema 1.4.2** (SAT è $\mathcal{NP}$-completo)

- Sappiamo già che $SAT \in \mathcal{NP}$.
- Devo mostrare che $\forall I \in \mathcal{NP}.\, I \leq_L SAT$.

  Dato che sappiamo CIRCUIT SAT $\leq_L SAT$, basta mostrare che

  $$\forall I \in \mathcal{NP}.\, I \leq_L \text{CIRCUIT SAT}$$

  Sia $I \in \mathcal{NP}$. Costruiamo $f \in LOGSPACE$ t.c. $x \in I \iff f(I)$ è soddisfacibile.

  Per ipotesi esiste una MdT non deterministica che decide $I$ in tempo $|x|^k$. Assumiamo che la macchina non deterministica faccia scelte binarie (possiamo sempre ricondurci a questo caso): allora possiamo definire una stringa binaria:

  $$B = b_0 b_1 \dots b_{|x|^k} \quad b_i \in \{tt, ff\}$$

  Ovviamente una macchina non deterministica non ha una tabella di computazione, ma fissata una serie di scelte se ne può costruire una. In questo caso però una casella non dipende solo dalle tre solite, ma anche dalla stringa $B$ delle scelte, quindi il circuito avrà $3m + 1$ ingressi ed $m$ uscite. $f \in LOGSPACE$ per la stessa ragione mostrata nella dimostrazione di CV $\mathcal{P}$-completo.