LA TEORIA della COMPLESSITÀ

Ramo dell'IT focalizzato sulla classificazione dei problemi computazionali in funzione della loro inerente difficoltà, ovvero delle risorse necessarie alla loro risoluzione.

RAPPRESENTAZIONE BINARIA

- $bin(n)$ rappresentazione binaria del naturale n

$$bin(n) = a_k a_{k-1} \dots a_1 a_0 \iff n = \sum_{i=0}^{k} a_i \cdot 2^i$$

- $log(n) := |bin(n)| - 1 = \log_2(\max(1,n))$ necessario per evitare $\log(0)$

  $n°$ di bit necessari per rappresentare n.

  ↳ ! capire che è rispetto allo spazio occupato in memoria che si considera la complessità

NOTAZIONI d'ORDINE

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

- $O(f) := \{ g : \mathbb{N} \to \mathbb{N} \mid \exists c\ \forall n,\ g(n) \leq c f(n) + c \}$ → a meno di costanti

  ↳ classe delle funz. che crescono al più come f

- $o(f) := \{ g : \mathbb{N} \to \mathbb{N} \mid \forall c\ \exists n_0\ \forall n \geq n_0,\ c g(n) + c \leq f(n) \}$

  ↳ 〃 〃 〃 〃 meno 〃 〃

- $\Omega(f) := \{ g : \mathbb{N} \to \mathbb{N} \mid f \in O(g) \}$

  ↳ 〃 〃 〃 〃 almeno 〃 〃

- $\Theta(f) := O(f) \cap \Omega(f)$

  ↳ 〃 〃 〃 〃 come f

PROPRIETÀ

- $\forall c > 0,\ O(cf) = O(f)$
- se $f_1 \in O(g_1) \wedge f_2 \in O(g_2)$, allora $f_1 + f_2 \in O(g_1 + g_2)$

  allora $f_1 \cdot f_2 \in O(g_1 g_2)$
- supposto $g(n) > 0\ \forall n$
  - se $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = l \neq 0$ allora $f \in O(g) \wedge g \in O(f)$
  - se $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0$ o $\lim\limits_{n \to \infty} \frac{g(n)}{f(n)} = \infty$ $f \in O(g) \wedge g \notin O(f)$
  - se $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0$ $f \in o(g)$

! $f \in o(g) \Rightarrow f \in O(g)$ ma non viceversa

OSSERVAZIONI

- $5n^4 + 3n^3 + n^2 \in O(n^4)$ → polinomio di grado max
- la base del logaritmo è ininfluente
- la base dell'esponente è influente → non sono dello stesso odg (ordine di grandezza)
- $O(n \log n) = O(\log(n!))$, $O(n^n) = O(n!)$

formula di Stirling → $e\left(\frac{n}{e}\right)^n \leq n! \leq ne\left(\frac{n}{e}\right)^n$

$$\lim_{n \to +\infty} \frac{3^n}{2^n} = \lim_{n \to +\infty} \left(\frac{3}{2}\right)^n = +\infty$$

ORDINE

- $O(1)$ costante
- $O(\log n)$ logaritmico
- $O(n)$ lineare → difficile avere complessità reali minori, spesso necessario almeno scorrere l'input
- $O(n \log n)$ pseudo-lineare $= O(\log n!)$
- $O(n^2)$ quadratico → prodotto di due interi → le operazioni $O(1)$ sono quelle presenti nativamente in assembly / instruction set
- $O(n^c)$ polinomiale
- $O(c^n)$ esponenziale
- $O(n!) = O(n^n)$ fattoriale

COME COMPORRE COMPLESSITÀ

es.

$f(g(x))$ $\qquad t_g \in O(n^2)$ $\qquad |l| \xrightarrow{g} |l|^2$ crea tutte le coppie

$\qquad\qquad t_f \in O(n \log n)$ ordina le coppie

$n \to n^2 \to n^2 \log n^2 \to n^2 \log|n|$

$t_f \circ t_g$

RICERCA vs. VERIFICA

Cercare è ~ alla dimensione dello spazio, quindi, anche se polinomiale, può essere lo stesso considerato costoso e.g. $(O(n^{11}))$

Cercare un insieme indipendente (ricoprimento/cricca) di card. K richiede l'esame di un n° di casi pari a: ↳ tutti i possibili sottoinsiemi di card. K.

→ n° elementi

$$\frac{n!}{K!(n-K)!}$$

per valori di $K \approx n/2$, questa q.tà cresce esponenzialmente in K.

NON è noto se sia possibile avere algoritmi polinomiali per risolvere questi problemi.

$\frac{n!}{(n-K)!}$ tutte possibili stringhe di lunghezza K $\rightarrow \underbrace{n(n-1)\dots(n-K)}_{K}$

$/K!$ diviso per tutte le permutazioni possibili (non siamo interessati all'ordine)

Al contrario, verificare se un sottoinsieme è ⊛ richiede un tempo polinomiale nel numero dei vertici $O(K^2) \leq O(n^2) \Leftarrow K \leq n$

I problemi che ammettono soluzioni di dimensione polinomiale rispetto all'input e algoritmi polinomiali di verifica della correttezza sono detti problemi **NP**.

L'algoritmo di raggiungibilità ha una complessità lineare su $|E|$.

Esiste un certificato che mi dimostri se una formula è tautologica? Sì, ma è 2^n, quindi esponenziale rispetto ai dati in input.
→ tabella di verità. Una dimostrazione logica sarebbe diversa.

CERTIFICATI

- $x \in A$? → sì
- dimostrazione? → c_x

$\langle x, c_x \rangle$ corretto $\Leftrightarrow x \in A$

NP: problemi di facile verifica

NP come PROIEZIONE di P

Un insieme A è in NP sse $\exists B$ (insieme dei certificati per A) decidibile in tempo polinomiale e un polinomio p t.c. $\forall x$

$$x \in A \Leftrightarrow \exists c_x, \underbrace{|c_x| \le p(|x|)} \wedge \langle x, c_x \rangle \in B$$

OSSERVAZIONI:

- il certificato deve avere dimensione polinomiale in x (se c_x enorme, il fatto che la sua verifica sia semplice è irrilev.)

- algoritmo rapido di verifica $\rightarrow$ NP   r.e.
- algoritmo rapido di ricerca $\rightarrow$ P   ricorsivi

$\updownarrow$

A è r.e. sse può essere visto come proiezione esistenziale di un insieme ricorsivo.

$P \subseteq NP$

Spazio di ricerca delle soluzioni (cammini)

Ogni nodo si prova a collegare gli altri

$n!$ → se limito a lunghezza l → n^l

$>$

$E \in O(n^2)$

$>$

n

Decidere se $\exists$ cammino tra due nodi > di una determinata lunghezza è NP-completo

caso peggiore → cammino Hamiltoniano

Verificare se V_0 è una cricca è economico.
Trovarla, però, ha come spazio di ricerca 2^n (insieme delle parti)

Il problema della cricca massima è NP-completo

Spazio di ricerca	2^n	$n!$
	insieme delle parti (tutti possibili sottinsiemi)	tutti i possibili cammini

GRAFI

Grafo finito → coppia (V, E)
- V insieme finito di vertici
- $E \subseteq V \times V$ relazione che definisce l'insieme degli archi

Un grafo si dice non orientato quando la relazione E è simmetrica e irriflessiva.

Sia $G = (V, E)$ un grafo
- $u, v \in V$ sono adiacenti se $(u, v) \in E$
- un cammino è una sequenza di vertici dove tutte le coppie di vertici consecutivi sono adiacenti
- un cammino è semplice se tutti i vertici sono distinti
- un ciclo è un cammino semplice il cui ultimo vertice è adiacente al primo.

DEFINIZIONI
- un cammino (ciclo) Hamiltoniano comprende tutti i vertici
- un ricoprimento di vertici per G è $V_0 \subseteq V$ t.c. $\forall e \in E$, ha almeno un'estremità in V_0
- G è n-colorabile se esiste una funzione di colorazione $col: V \rightarrow c_1, \dots, c_n$ t.c. vertici adiacenti abbiano colori diversi

$$(u, v) \in E \Rightarrow col(u) \neq col(v)$$

- G è completo se ogni coppia di nodi distinti è connessa da un arco.
- una cricca (clique) di G è un suo sottografo completo $G' = (V', E')$ ovvero $V' \subseteq V \wedge E' : V' \times V' \subseteq E$
- un insieme indipendente in G è un sottoinsieme di vertici $V' \subseteq V$ t.c. $\forall u, v \in V' \Rightarrow (u, v) \notin E$

OSSERVAZIONI
→ $E = \emptyset$
- V è un caso degenere di ricoprimento. Se R è un ricoprimento, ogni suo sovrainsieme lo è → ricoprimenti minimi
- $\forall v \in V$, $\{v\}$ caso degenere di insieme indipendente. $\forall$ sottoinsieme di un insieme indipendente lo è → insiemi indipendenti massimi
- $\forall v \in V$, $\{v\}$ " " " cricca (come anche $\emptyset$). $\forall (u, v) \in E$ sono una cricca. $\forall$ sottoinsieme è una cricca → cricche massime

MATRICE di ADIACENZA   ! $m \in O(n^2) \longrightarrow$ caso ~~pessimo~~

$M_G(u,v) = 1 \Leftrightarrow (u,v) \in E$ se $|V| = n$ la rappresentazione è $O(n^2)$

! se la matrice è sparsa, possono essere più convenienti altre rappresentazioni (e.g. lista di archi)

RAGGIUNGIBILITÀ

Dati due vertici, u e v, determinare se $\exists$ cammino da u a v

Partizioniamo V in 3: D (done), C (current), T (todo)

1. Inizialmente $D = \emptyset$, $C = \{u\}$, $T = V \setminus \{u\}$
2. Se $v \in C$ terminiamo con successo
3. Se $C = \emptyset$ terminiamo con fallimento
4. Selezioniamo un vertice $x \in C$ e consideriamo gli $Ad(x)$.
   poniamo $D = D \cup \{x\}$
   $C = (C \cup (Ad(x) \cap T)) \setminus \{x\}$
   $T = T \setminus Ad(x)$
5. Ripetiamo dal passo 2.   [$\forall$ arco visitato 1 volta $\rightarrow O(n^2)$]

2-COLORABILITÀ $\longrightarrow$ spazio di ricerca $\rightarrow 2^n$

Siano $\{0,1\}$ i due colori. $D \cup C$ sono già colorati, T da colorare.

1. Inizialmente $\equiv$.
2. Se $T = \emptyset$ l'algoritmo termina con successo.
3. Altrimenti selezioniamo un vertice $x \in C$ (se $C = \emptyset \rightarrow G$ non connesso $\rightarrow$ si prende un nuovo elemento da T e lo si colora arbitrariamente), consideriamo $\forall v \in Ad(x)$
   - se $v \in D \cup C$, verifico $col(v) = \overline{col(x)}$ e fallisco altrimenti
   - se $v \in T$, pongo $col(v) = \overline{col(x)}$ e $T = T \setminus \{v\}$ $C = C \cup \{v\}$
4. Rimuoviamo x da C. $C = C \setminus \{x\}$ $D = D \cup \{x\}$
5. Ripetiamo dal passo 2.

FLUSSO MASSIMO

Una rete N è un grafo orientato con una sorgente s (source), un pozzo t (target) e una capacità $c(u,v)$ associata ad $\forall$ arco.

Un flusso in N è una funz. $f(u,v)$ che ad ogni arco associa un intero t.c.

- $f(u,v) \leq c(u,v)$
- $\Sigma f_{in} = \Sigma f_{out}$ $\forall$ nodo $\setminus \{s,t\}$

Il problema consiste nel determinare il flusso massimo.

SOTTRARRE UN FLUSSO da UNA RETE

Data rete N e flusso f, la nuova rete $N \setminus f$:

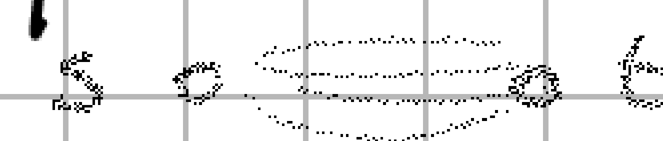
- $c'(i,j) = c(i,j) - f(i,j)$ cancellando gli archi nulli
- $c'(j,i) = c(j,i) + f(i,j)$

ALGORITMO

1. Si parte con flusso maxf inizialmente nullo
2. Si cerca un cammino da s a t nella rete N e si considera il flusso determinato dalla capacità massima del cammino (= capacità_{max} minore tra tutti gli archi → bottleneck). Se ∄ si termina con maxf
3. Si pone maxf += f ; $N = N \setminus f$ e si ripete da 2.

COMPLESSITÀ

Se C capacità max degli archi, maxf $\in O(nC)$ in quanto ci sono meno di n archi che partono dalla sorgente.



- la ricerca del cammino è $O(n^2)$
- ci sono al massimo $\underbrace{nC \text{ iterazioni}}_{(i)}$, dato che il flusso aumenta ad ogni ciclo.

Quindi è $O(n^3 C)$, ma attenzione! Non è cubico, infatti
$O(n^2) \cdot O(nC)$

‼ L'algoritmo dipende in modo lineare da C, quindi in maniera esponenziale rispetto alla rappresentazione della capacità ($\log_2 C$) (esponenziale nella dimensione dell'input)

Prendendo sempre il cammino più corto, possibile utilizzare un arco come collo di bottiglia in un numero limitato di casi, ottenendo (i) n^3 come maggiorazione del numero di iterazioni

al max n

Dunque è $O(n^5)$ $\quad O(n \cdot |E|)$ → n° max di flussi

$O(n^2) \cdot O(n^3)$ $\quad O(n^3)$

PROBLEMI DECISIONALI

Spesso è possibile trovare una versione decisionale di un problema di ottimizzazione che abbia una complessità simile (comparabile).
La raggiungibilità e la 2-colorabilità ne sono un esempio: richiedono una risposta booleana (definiscono un linguaggio)

Dato un problema di ottimizzazione è sempre possibile trovare una versione decisionale

MATCHING BIPARTITO

Un grafo bipartito è una tripla $B=(U,V,E)$ dove U e V sono due insiemi di nodi, $|U|=|V|$ e $E \subseteq U \times V$ l'insieme degli archi

Un matching è un insieme $M \subseteq E$ che associa ad ogni $u \in U$ uno e un solo $v \in V$

Il problema consiste nel determinare l'esistenza di un matching.

Può essere ridotto al problema del flusso massimo
(all'esistenza di un flusso di una capacità nota e prefissata $= n$)

ALTRI ESEMPI di RIDUCIBILITÀ

Dati $G=(V,E)$ e $K \in \mathbb{N}$, determinare se:

1) $\exists$ insieme indipendente $V' \subseteq V$ t.c. $|V'| \geq K$ (max)
2) $\exists$ ricoprimento " " $|V'| \leq K$ (min)
3) $\exists$ cricca " " $|V'| \geq K$ (max)

Sono riducibili l'uno all'altro (è possibile trasformare il problema, trovare una funzione).

Il complementare di un insieme indipendente è un ricoprimento e viceversa.
(se tra i nodi esclusi fosse esistito un arco (ragionamento per assurdo), allora almeno uno dei due nodi sarebbe dovuto appartenere al ricoprimento)

Per ridurre la cricca agli altri, necessario complementare gli archi. (o la matrice di adiacenza). Nel nuovo grafo la cricca sarà un insieme indipendente.

$V' \subseteq V$ indipendente sse $V \setminus V'$ è un ricoprimento
$V' \subseteq V$ " sse V' è una cricca nel $G=(V,\overline{E})$

CLASSI di COMPLESSITÀ

MACCHINA di TURING

- automa di controllo a stati finiti (~ microprocessore)
- testina di lettura mobile ∀ nastro
- nastri di memoria illimitati

input tape → M → output tape
working tapes ↔ M

↳ diff. con Von Neumann → accesso sequenziale ≠ diretto

configurazione istantanea (≠ stato interno automa)
↳ tutto ciò che serve per riprendere esecuzione in un secondo momento.

la computazione avviene per passi discreti, tramite transizione → $\vdash$
tra due configurazioni: char in read + stato interno →
char in write + " " + sx/dx

→ stella di Kleene

$\vdash^*$ chiusura transitiva e riflessiva di $\vdash$ $(q_0 \dots \vdash^* q_f)$

δ → funzione di transizione ≡ programma
$\vdash$ → passaggio tra 2 config ≡ effetto del programma

CLASSI DETERMINISTICHE di COMPLESSITÀ

- $time_M(x)$: tempo di esecuzione di M su input x
  i.e. n° di passi per la computazione
- $space_M(x)$: n° max di celle visitate (nastri di lavoro)

Ma non siamo interessati all'input quanto alle sue dimensioni ≡ lunghezza n

- $t_M(n) := \max\{time_M(x) : |x| = n\}$ caso pessimo con input di dim.?
- $s_M(n) := \max\{space_M(x) : |x| = n\}$

Data $f : \mathbb{N} \to \mathbb{N}$ introduco le classi di complessità:

- $DTIME(f) := \{L \subseteq \Sigma^* : \exists M, L = L_M \wedge t_M \in O(f)\}$
  determin. insieme di tutti quei linguaggi, riconosciuti da M, la quale lavora in tempo $O(f)$
- $DSPACE(f) := \{L \subseteq \Sigma^* : \exists M, L = L_M \wedge s_M \in O(f)\}$

classi di complessità ≈ insieme di problemi che posso riconoscere in tempo/spazio $O(f)$ ≈ insieme dei ling. che riconosco con complessità $O(f)$
≠
$t_M(n) / s_M(n)$ ≈ funzioni che calcolano effettivamente

TEMPO e SPAZIO

T.
$\forall f : \mathbb{N} \to \mathbb{N}$ si ha:

costa $f \sim O(f)$

$$DTIME(f) \subseteq DSPACE(f) \subseteq \bigcup_{c \in \mathbb{N}} DTIME(2^{c(\log + f)})$$

- $DTIME(f) \subseteq DSPACE(f)$

  per visitare una nuova cella ho bisogno di almeno un passo
  ↓
  il consumo in tempo è un upper-bound al consumo in spazio (se ci metto f tempo, ho occupato al massimo f spazio)
  l'inclusione sembrerebbe rovesciata, ma se si pensa a DTIME(f) maggiori si può capire la relazione

- $DSPACE(f) \subseteq \bigcup_{c \in \mathbb{N}} DTIME(2^{c(\log + f)})$

  se immagino di lavorare con memoria limitata K, una volta ottenute tutte le possibili configurazioni della memoria, per evitare cicli, devo terminare (altrimenti la macchina dovrebbe ripassare per una configurazione già trovata)
  Essendo la macchina deterministica, potrei in questo caso determinare la divergenza. (!!! configurazione → mem. + stati + pos)

  Quindi, se occupa spazio f, deve terminare in max quel bound di tempo, esponenziale su f.

  Lo spazio fornisce quindi un upper bound al tempo

  2^c → dimensione delle parole di memoria
  ex. 32 bit → 2^{32}

DIM.

una configurazione è una tupla: $q, (\sigma_1, \tau_1), \dots, (\sigma_K, \tau_K)$

stato interno ≈ registri

$\underbrace{(\sigma_1, \tau_1), \dots, (\sigma_K, \tau_K)}$ K nastri → divisi in 2 seminastri

grazie alla funz. $s_M(n)$ so lo spazio max occupato e posso andare a calcolare tutte le possibili configurazioni:

$$t_M(n) \leq |Q| \cdot K \cdot |\Gamma|^{s_M(n)}$$

$|Q|$: cardinalità stati interni
K: n° nastri
$|\Gamma|$: cardinalità alfabeto
$s_M(n)$: spazio occupato

posso cambiare la base da Γ a 2 utilizzando c per indicare la parola

- $t_M(n) \le |Q| \cdot k \cdot |\Gamma|^{s_M(n)} \le 2^{c(\log(n)+f(n))+c}$

!! capire perché ho una dipendenza esponenziale → il logaritmo è un dettaglio (non importante)

funzioni compatibili complessità computazionale ~ operazione
↓
log → non può essere calcolato in $O(\log f)$ ad es. $\log_2 \underbrace{2^{10}}$
→ mi serve contare la lunghezza dell'input → lineare nella dimensione dell'input, almeno (e non sublineare) "caso fortunato"

problema decisionale
takes $n \in \mathbb{N}$ as input → returns bool

il prendere una stringa ($\equiv n \in \mathbb{N}$) e restituire un bool è isomorfo ai sottoinsiemi di Σ^* e quindi ai linguaggi [classi di complessità]

RIDUZIONE dei NASTRI da K a 1

T.
$\forall f: \mathbb{N} \to \mathbb{N}$ si ha:

$$DSPACE(f) \subseteq DSPACE_1(f) \wedge DTIME(f) \subseteq DTIME_1(f^2)$$

L'idea è quella di dover scrivere un interprete per trasformare K nastri in 1, arricchendo l'alfabeto oppure utilizzando una porzione di nastro ridondante per marcare la posizione della testina.

$\forall$op $O(1)$ della M_3, su M_1 ho $O(2n)$ ($O(n)$), in quanto devo fare almeno due scansioni del tape, una per salvare la pos delle testine, l'altra per aggiornare i valori (di tape e testina).

Dato che la dim del nastro può al più crescere linearmente, il che vuol dire che la Σ finale di questo valore crescente, porterà a una dipendenza quadratica in tempo $\left(\frac{n(n+1)}{2} = \sum_{i=0}^{n} i\right)$

L'altro modo sarebbe quello di tenere la testina ferma e spostare i nastri, riducendo quindi il rw a $O(1)$, alzando però il costo della singola mossa a $O(n)$, in quanto bisogna riscrivere il nastro.

RIDUZIONE dei NASTRI da K a 2

$$DTIME(f) \subseteq DTIME_2(f \log f)$$

SEQUENZA di ATTRAVERSAMENTO

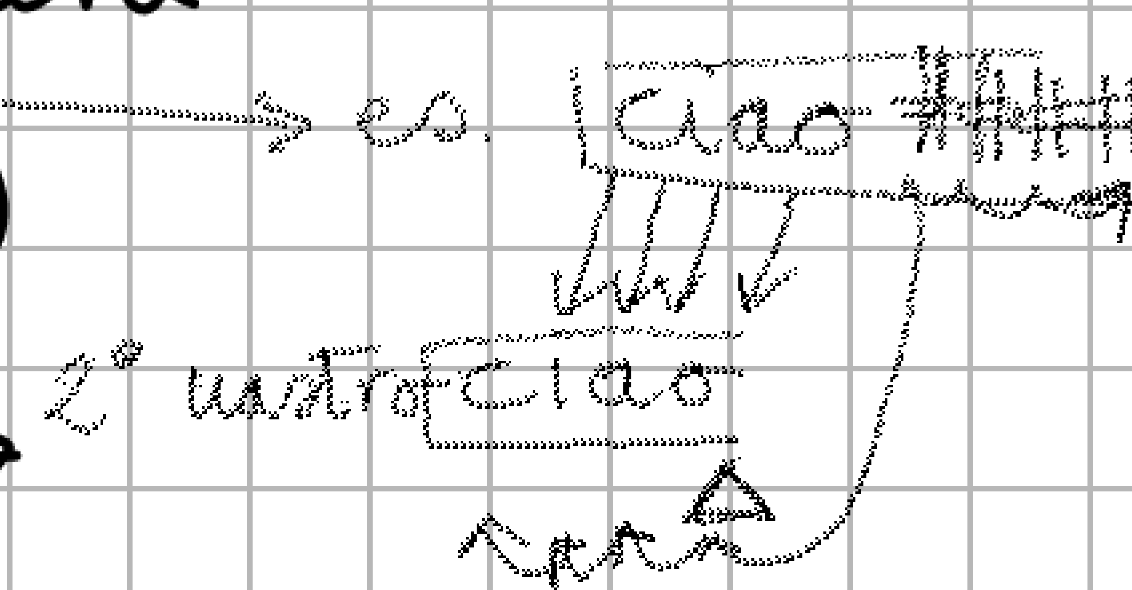
In una M_1 (MdT a 1 nastro) il costo di copiare / spostare una |stringa| = ℓ in memoria è lineare rispetto a ℓ e proporzionale alla distanza a cui si vuole portare ($> \ell$).
Non si può infatti pensare di sfruttare l'automa (in quanto a stati finiti). È necessario copiare un carattere alla volta.

UN LIMITE INFERIORE per la RIDUZIONE dei NASTRI

Sia $L := \{ w \#^{|w|} w \mid w \in \{0,1\}^* \}$. Allora

1) $L \in DTIME_2(n) \subseteq DTIME_1(n^2)$
2) $L \notin DTIME_1(t)$ per nessun $t \in o(n^2)$

linguaggio con padding $\longrightarrow$



MACCHINE ad ACCESSO CASUALE

RAM (Random Access Machine), ha le seguenti istruzioni:

istruzione	operando	semantica
– READ	j	$r_0 := i_j$
	$\uparrow j$	$r_0 := i_{r_j}$
– STORE	j	$r_j := r_0$
	$\uparrow j$	$r_{r_j} := r_0$
– LOAD	x	$r_0 := x$
– ADD	x	$r_0 := r_0 + x$
– SUB	x	$r_0 := r_0 - x$
– HALF		$r_0 := \text{div } r_0\ 2$
– JUMP	j	$K := j$
– JPOS	j	if $r_0 > 0$ then $K := j$
– JZERO	j	if $r_0 = 0$ then $K := j$
– HALT		$K := 0$

$K \to$ program counter
$r_i \to$ i-esimo registro
$i_j \to$ j-esimo input

$r_0 \to$ accumulatore

Queste op. le consideriamo con costo unitario $O(1)$, ma in realtà hanno costo lineare rispetto alla dimensione degli operandi (ℓ).
MODELLO di COSTO con input $I = (i_1, \dots, i_k)$

$$\ell(I) = \sum_{j=1}^{k} |bin(i_j)|$$

SIMULAZIONE di una RAM con una MdT

M_6:

- 1° nastro → input
- 2° nastro → registri → lista di coppie → enumerabile (→ limitati)
  $i : r_i$
  i registri hanno dim variabile → aggiornarlo? → nel caso pessimo necessario cancellarlo e appenderlo
- 3° nastro → PC (non si può usare lo stato interno dell'automa)
- 4°-6° " → operandi + risultato per istruzioni aritmetiche

CRESCITA del CONTENUTO dei REGISTRI

ex.

La crescita è dovuta alle op. aritm. (somma di 2 n' a x bit → x+1 bit) ed è quindi legata linearmente al tempo: a ogni passo posso al max crescere di 1 unità di dimensione.

Al passo t di una computazione su input I, il contenuto di ∀ reg ha una dim ≤

$$t + \ell(I) + c$$ → max costante del programma

!! N.B. Il lemma non varrebbe se si comprendesse la moltiplicazione → sarebbe esponenziale

COSTO della SIMULAZIONE RAM vs MdT

∀ programma RAM P con complessità in tempo $t_P(n)$ ($|\text{input}| = n$) può essere simulato da una MdT_6 in $O(t_P(n)^3)$

DIM.

- ∀ operando necessarie max 2 scan del nastro (riferimento indiretto)
- il nastro è composto da max $O(t_P(n))$ coppie di dim $O(t_P(n))$, quindi ha dim $O(t_P(n)^2)$
- quindi la simulazione di ogni op è $O(t_P(n)^2)$

La computazione complessiva è Σ op., quindi $O(t_P(n)) \cdot O(t_P(n)^2)$ (n° di passi sulla RAMachine)

⇒ $O(t_P(n)^3)$

↳ slowdown cubico (polinomiale)

LA COMPLESSITÀ COMPUTAZIONALE — DIPENDE → dal modello di calcolo !!!

Definizioni "ragionevoli" delle funzioni di costo permettono mutua simulazione in tempi polinomiali

MACCHINA di TURING UNIVERSALE

Per semplicità usiamo una MdT a 5 nastri (M_5).
Dato l'input $\langle x, y \rangle$, la macchina li ricopia sui nastri di lavoro
nastro del programma — nastro della computazione
Un opportuno algoritmo (parser) verifica la correttezza di x e passa alla simulazione di M_x su input y utilizzando un nastro per lo stato interno di M_x. Il risultato della computazione viene ricopiato sul nastro di output. (funzione)
La sim. richiede la scansione della tabella di transizione $\forall$ passo, per determinare la mossa da effettuare sul nastro della computazione.

Costi:
- $O(|x|) + O(|y|)$ — copiare l'input
- $O(|x|^2)$ — parsing
- $O(|x| \cdot time_{M_x}(y))$ — intera computazione, considerando $O(|x|)$
- $O(time_{M_x}(y))$ — ricopiare l'output — costo unitario

Lo slowdown dipende quindi dalla dimensione del programma $|x|$ e non dall'input y. $\rightarrow$ non cambio la complessità del prog. di partenza

Se $time_{M_x}(y) \in O(y^2)$, pure M lo è $\rightarrow$ con op non più $O(1)$ ma $O(|x|)$

Per il teorema della riduzione dei nastri, simularla con una M_1 porta a uno slowdown quadratico ma comunque polinomiale.

Tutti i nastri, tranne quello di lavoro, hanno dim. fissa dato x.
(motivo per cui un nastro non è così irragionevole

IL TEOREMA della GERARCHIA (SPAZIO)

I teoremi della gerarchia riflettono l'idea intuitiva che, disponendo di una maggiore quantità di risorse, sia effettivamente possibile affrontare problemi più complessi (e.g. esistono problemi con funz. calc. in tempo quadratico ma non in tempo lineare).

T.

Sia s una funz. costruibile in spazio e $\log \in O(s)$ → più che logaritm.

Allora:

$$O(s) \subseteq O(s') \Longleftrightarrow DSPACE(s) \subseteq DSPACE(s')$$

↑ sse !!!

DIM.

- $\Longrightarrow$ è ovvio: $O(s) \subseteq O(s') \Rightarrow s \in O(s')$ e quindi è anche risolubile con complessità s' (DSPACE)
  ovvero, aumentando le risorse disponibili riesco a calcolare almeno quello che calcolavo prima.
  → la negazione dell'ipotesi porta a una $\nsubseteq$ nella 2ª parte
- $\Longleftarrow$ non è ovvio:
  voglio dimostrare che aumentando l'ordine di grandezza delle risorse riesco effettivamente a risolvere problemi nuovi ovvero

$$O(s) \nsubseteq O(s') \Longrightarrow DSPACE(s) \nsubseteq DSPACE(s')$$

(nella dim.) ! → x_0

$\equiv$

$\circledast = s \notin O(s') \Leftrightarrow \forall f \in O(s')$ $\underline{\exists \text{ infiniti } n}$ t.c. $s(n) > f(n)$

$\equiv s' \in o(s)$

$n \in o(n \log n) \Rightarrow DSPACE(n \log n) \nsubseteq DSPACE(n)$

sotto questa ipotesi devo dimostrare $DSPACE(s) \nsubseteq DSPACE(s')$

↳ cioè → devo esibire un linguaggio L t.c. $L \in DSPACE(s)$ ma $L \notin DSPACE(s')$

LINGUAGGIO L

Definito come insieme di coppie $\langle \ulcorner M \urcorner, x \rangle$ → codice di una MdT, → stringa di padding

$$L := \{ \langle \ulcorner M \urcorner, x \rangle \mid \underbrace{space_M(|\langle \ulcorner M \urcorner, x \rangle|) \leq s(|\langle \ulcorner M \urcorner, x \rangle|)}_{\text{complessità}} \wedge \underbrace{\langle \ulcorner M \urcorner, x \rangle \notin L_M}_{\text{diagonalia.}} \}$$

complessità ← vincoli → diagonalia.

Voglio dim. che $L \notin DSPACE(s')$. Suppongo $\exists M_0$ che riconosce L, t.c. $space_{M_0} \in O(s')$. Per $\circledast$ $\exists$ infiniti n t.c. $s(n) > space_{M_0}(n)$.

Scelgo x_0 t.c. $space_{M_0}(|\langle \ulcorner M_0 \urcorner, x_0 \rangle|) \leq s(|\langle \ulcorner M_0 \urcorner, x \rangle|)$ e quindi

$\langle \ulcorner M_0 \urcorner, x_0 \rangle \in L \Leftrightarrow \langle \ulcorner M_0 \urcorner, x_0 \rangle \notin L_{M_0}$

→ contraddizione

supp. errata (per assurdo) su $L \notin DSPACE(s')$

È stato necessario aggiungere una stringa di padding, in modo da catturare un comportamento asintotico (creazione di infinite stringhe di dimensione crescente)
Infatti, il fatto che $space_{M_0} \in O(s')$ non sarebbe bastato da solo per rispettare il vincolo:

$$space_{M_0}(\ulcorner M_0 \urcorner) \leq s(\ulcorner M_0 \urcorner)$$

poiché la conoscenza della complessità asintotica nulla dice su quella puntuale (ovvero su $\ulcorner M_0 \urcorner$).

Dobbiamo ancora dimostrare che $L \in DSPACE(s)$, per farlo descrivo l'algoritmo riconoscitore.
Innanzitutto faccio partire la macchina M su input x e verifico che le risorse siano rispettate ($space_M(|\langle\ulcorner M \urcorner, x\rangle|) \leq s(|\langle\ulcorner M \urcorner, x\rangle|)$), se sì, controllo che la coppia $\langle\ulcorner M \urcorner, x\rangle$ non appartenga a L_M, per considerarla $\in L$.
È una simulazione di tipo bound, in quanto le risorse sono limitate.

1. Se l'input y è della forma $\langle\ulcorner M \urcorner, x\rangle$ allora calcola $0^{s(n)} \rightarrow$ spazio di lavoro con $n = |y| = |\langle\ulcorner M \urcorner, x\rangle|$ e fallisce altrimenti
2. calcola un timeout $t := |Q| \cdot 2 \cdot |\Sigma|^{s(n)} \rightarrow$ tutte le possibili config.
   stati, nastri, alfabeto
3. simula passo passo il comportamento di M su y fino a terminaz. o fino a quando sfora spazio ($s(n)$) o tempo t
4. la macchina accetta l'input y se la sim. di M termina rifiutandolo o perché sfora il tempo (ma non lo spazio)

! Il vincolo in spazio mi fornisce anche un limite superiore al tempo, creando effettivamente un timeout.

OSSERVAZIONE: il teorema non vale senza l'assunzione di costruibilità.
È possibile dimostrare che $\forall g$ ricorsiva non decrescente $\exists f$ t.c.
$DSPACE(f) = DSPACE(g \circ f)$

$s' \in o(s) \Longleftrightarrow O(s) \nsubseteq O(s')$
s' sicuramente crescita minore di s $\Longleftrightarrow$ O(s) non può essere contenuto in O(s')

IL TEOREMA della GERARCHIA (TEMPO)

t funzione costruibile in tempo e $n \in O(t)$

$$O(t) \subseteq O(t') \Rightarrow DTIME(t) \subseteq DTIME(t') \Rightarrow O\left(\frac{t}{\log t}\right) \subseteq O(t')$$

La prima implicazione è conseguenza delle definizioni, la seconda necessita di una dimostrazione simile a quella per lo spazio, ma ho bisogno di ipotesi più forti

[Lez 08 ~ min 30]

Di base dimostrazione simile a quello in spazio, ma ho bisogno di ipotesi più forti.

!! capire l'enunciato, il problema tecnico, ma non serve la dimostrazione (in tempo, in spazio invece sì)

ALCUNE CLASSI di COMPLESSITÀ DETERMINISTICA

- $P := \bigcup_{c \in \mathbb{N}} DTIME(n^c)$ insieme dei ling. riconoscibili in tempo polinomiale da una MdT
  $\hookrightarrow DTIME(n) \cup DTIME(n^2) \cup \dots$

- $EXP := \bigcup_{c \in \mathbb{N}} DTIME(2^{cn})$ oppure $\bigcup_{c \in \mathbb{N}} DTIME(2^{n^c})$ $\rightarrow$ classe più ampia con proprietà $\neq$

  $DTIME(2^n) \subset DTIME(2^{2n})$ strettamente contenuta

  strettamente più piccolo di ad es.

  $\subseteq$ PSPACE

  $\not\subseteq$ non è contenuto

- $LOGSPACE := DSPACE(\log)$

- $PSPACE := \bigcup_{c \in \mathbb{N}} DSPACE(n^c)$

- $EXPSPACE := \bigcup_{c \in \mathbb{N}} DSPACE(2^{cn})$

Come corollario:

- $LOGSPACE \subseteq P \subseteq PSPACE \subset EXPSPACE$ dato che $n^c \in \underline{o}(2^{nc})$ $\hookrightarrow \subset$ inclusione stretta
- $LOGSPACE \subset PSPACE$
- $P \subset EXP \subseteq EXPSPACE$

Per dimostrarli ho a disposizione 2 teoremi:
- $\subseteq$ tempo-spazio $\rightarrow$ per risorse diverse (e.g. $LOGSPACE \subseteq P$)
- $\subset, \subseteq$ gerarchia $\rightarrow$ stessa risorsa ($LOGSPACE \subset PSPACE$)

$\log \in o(n^c) \Rightarrow$
$DSPACE(\log) \subset DSPACE(n^c)$

$DSPACE(\log) \subseteq DTIME(2^{c(\log + f)})$
$DSPACE(\log) \subseteq \underbrace{DTIME(n^c)}_{\simeq P}$

Per nessuna inclusione $\subseteq$ si è riuscito a dimostrare che sia stretta

PADDING

EXP vs. PSPACE $\longrightarrow$ EXP $\neq$ PSPACE

Partendo da L costruisco un nuovo ling. L' con la tecnica del padding.

DIM.

Per il teorema della gerarchia in tempo

$$EXP \subseteq DTIME(2^{n^{1.5}}) \subset DTIME(2^{n^2}) \Rightarrow EXP \subset DTIME(2^{n^2})$$

! $EXP := \bigcup_{c \in \mathbb{N}} DTIME(2^{cn}) \Rightarrow EXP = DTIME(2^{n^c})$

ma $L \notin EXP$

Suppongo $EXP = PSPACE$ e sia $L \in DTIME(2^{n^2})$

Creo L' con padding (quadratico) $\rightarrow$ aggiungo info ridondanti i.e. '#'

$$L' := \{x\#^t : x \in L \wedge |x| + t = |x|^2\} \in DTIME(2^n) \subseteq EXP$$

La complessità del riconoscimento cambia e sembra essere logaritm. rispetto all'input, ma questo è solo perché faccio "esplodere" la dimensione dell'input, infatti la lunghezza è il quadrato della stringa di partenza. $t = |x|^2 - |x|$

In particolare, contare il n° di caratteri di x e quante gratelline ci sono è lineare, la parte più costosa è verificare che $x \in L$, che infatti costa $O(2^{n^2})$. N.B. $x \in L$ ha $|x| = \sqrt{|x\#^t|}$

Quindi $O(2^{\sqrt{n}^2}) = O(2^n)$ dove $n = |x\#^t|$

Per ipotesi $L' \in PSPACE$, ovvero $\exists k > 0 \mid L' \in DSPACE(n^k)$

Dunque anche $L \in PSPACE$.

Ma questo porterebbe a $DTIME(2^{n^2}) \subseteq PSPACE = EXP$

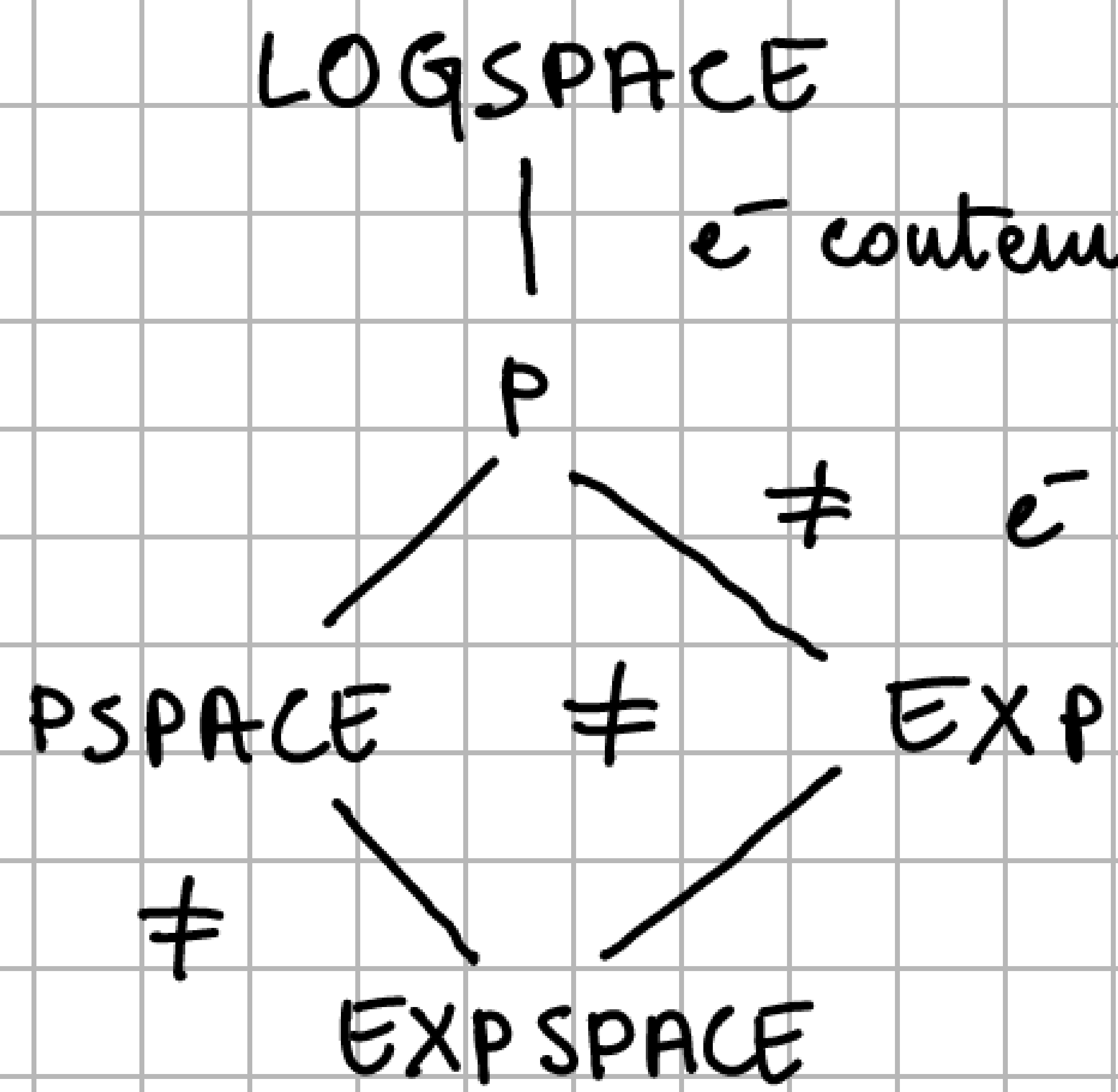
in contraddizione

$EXP \subset DTIME(2^{n^2})$

ASSURDO!

Il padding mi porta in EXP, mentre il de-padding mi tiene in PSPACE!

I due linguaggi sono isomorfi.



| è contenuto o uguale a

≠ è strettamente contenuto in

MdT NON DETERMINISTICHE

MdTN è definita in modo analogo alla MdT, con eccezione della funzione di transizione, che diventa una relazione multi-valore:

$$\delta \subseteq Q \times \Sigma^K \times Q \times (\Sigma \times \{L,R\})^K$$

- la nozione di configurazione è invariata
- la " di passo è adattata: le computazioni non sono più sequenze lineari ma alberi.
- la macchina si arresta su input x se $\exists$ almeno un ramo che termina (cammino nell'albero)
- la macchina riconosce un input x se $\exists$ " " " " si arresta in uno stato di riconoscimento.

CLASSI di COMPLESSITÀ N

Sia data MdTN M.

- $time_M(x)$ è il minimo n° di passi richiesto da una qualche computazione di M su x.
- $space_M(x)$ è il " spazio richiesto " " " " " " "

- $t_M(n) := \max(\{n\} \cup \{time_M(x) : |x| = n\})$
- $s_M(n) := \max(\{1\} \cup \{space_M(x) : |x| = n\})$

- $NTIME(f) := \{L \subseteq \Sigma^* : \exists \text{ MdTN } M,\ L = L_M \wedge t_M \in O(f)\}$
- $NSPACE(f) := \{L \subseteq \Sigma^* : \exists \text{ MdTN } M,\ L = L_M \wedge s_M \in O(f)\}$

— — —

- $NP := \bigcup_{c \in \mathbb{N}} NTIME(n^c)$ unione degli insiemi dei L riconoscibili it tempo polinom.
- $NEXP := \bigcup_{c \in \mathbb{N}} NTIME(2^{cn})$
- $NLOGSPACE := NSPACE(\log)$
- $NPSPACE := \bigcup_{c \in \mathbb{N}} NSPACE(n^c)$

<u>T.</u> $D \subseteq N$

$\forall f : \mathbb{N} \to \mathbb{N}$ $\quad DTIME(f) \subseteq NTIME(f) \wedge DSPACE(f) \subseteq NSPACE(f)$

Ovvio, le MdT sono un caso particolare di MdTN.

COROLLARIO

- $\underline{\underline{P \subseteq NP}}$!!!
- $LOGSPACE \subseteq NLOGSPACE$
- $PSPACE \subseteq NPSPACE$
- $EXP \subseteq NEXP$

SIMULAZIONE del NONDETERMINISMO

NTIME vs. DSPACE

$$NTIME(f) \subseteq DSPACE(\overbrace{id+f})$$ → come dire f, ovvero almeno lineare rispetto all'input

ex. Tautologicità di una proposizione, una MdTN la può dimostrare in tempo $O(f)$ e, considerando una "riga" (istanza di valori) per volta, una MdT la simula in spazio lineare all'input. Detto questo, il tempo sulla MdT per verificare tutti i casi è realisticamente esponenziale rispetto all'input.
Questo fa capire quanto sia grande la classe DSPACE.

Per simulare una MdTN uso un nastro supplementare con un nuovo alfabeto (con tanti char quante le scelte per δ) che sarà lungo $t_M(n)$.

Dato che NTIME(f) implica che c'è almeno un cammino che ci mette $O(f)$, nella simulazione sono interessato solo a quello.

Mi rendo conto che riesco a ottimizzare lo spazio a discapito del tempo (devo ripetere la verifica partendo dalla radice)

NSPACE vs. DTIME

$$NSPACE(f) \subseteq DTIME(2^{c(\log + f)})$$

Il modo più semplice per dimostrarlo è tramite il grafo della computazione. Devo di fatto esplorare tutto il grafo (nodi = configurazioni istantanee, archi = transizioni).
Trovare un cammino è quadratico rispetto al n° dei nodi, le configurazioni (nodi) crescono in modo esponenziale

MAGGIORAZIONI (corollario)

→ stessa dimensione a confronto

- $NTIME(f) \subseteq \bigcup_{c \in \mathbb{N}} DTIME(2^{c(id+f)})$
- $NSPACE(f) \subseteq \bigcup_{c \in \mathbb{N}} DSPACE(2^{c(\log+f)})$

OSSERVAZIONI

Lo spazio, a differenza del tempo, può essere riutilizzato. Questo dà l'intuizione che ci sia "spazio" per un miglioramento della complessità.

Se il grafo da analizzare (per la simulazione) è molto grande, tengo una funzione come sua descrizione (nodi adiacenti ⇒ archi).

∘ Costo in spazio per trovare un cammino ?

▷ Se uso un labeling (e.g. enumerazione) per i nodi, devo considerare un $\underset{\#(bit)}{\log n}$ aggiuntivo (perché cresce come info).

Se f (in $NSPACE(f)$) è la dimensione di una config., il numero totale di config. è quadratico rispetto al n° di nodi.

∃ un algoritmo che riesce a fare un ottimo lavoro in spazio, ma a discapito del tempo (sono risorse che si contrastano)

RAGGIUNGIBILITÀ

Sia $G=(V,E)$ grafo con $|V|=n$. È possibile determinare se $u,v \in V$ sono connessi da un cammino di lunghezza $<$ a 2^i con consumo in spazio $O(i \cdot \log n)$

inizialmente $\log n \longrightarrow 2^{\log n} = n$ (lunghezza cammino max)

computaz. sequenz. + and logico

```
let rec recheable(i,u,v) ::=
   if i=0 then u=v v <u,v> ∈ E
   else ∃z. reacheable(i-1,u,z) ∧ reachable(i-1,z,v)
```

→ for z   basta che uno dei due termini

(progr. ricorsivo)

Dato che lo spazio occupato è dato dai record di attivazione, quando calcolo il 2° ramo posso riutilizzare interamente lo spazio del primo.

I record di attivazione quanto sono grandi? Le var di I/O sono $O(\log n)$. Dato che ho al massimo $O(\log n)$ chiamate

→ $O(\log(n)^2)$

La complessità in tempo è invece $O(\underbrace{(2n)^i}_{n \cdot n^{\log n}})$ $i = \log n$ → esponenziale

2 computazioni con n passi, i volte

Ovviamente peggio che $O(n^2)$ → polinomiale

IL TEOREMA di SAVITCH

Come corollario del teorema precedente, riesco a trovare un'occupazione in spazio ancora più ridotta, ma con identico caso pessimo in tempo.

Faccio la ricerca del cammino su uno spazio 2^{cf}

$$O(\log(2^{cf})^2) = O((cf)^2) = O(f^2)$$

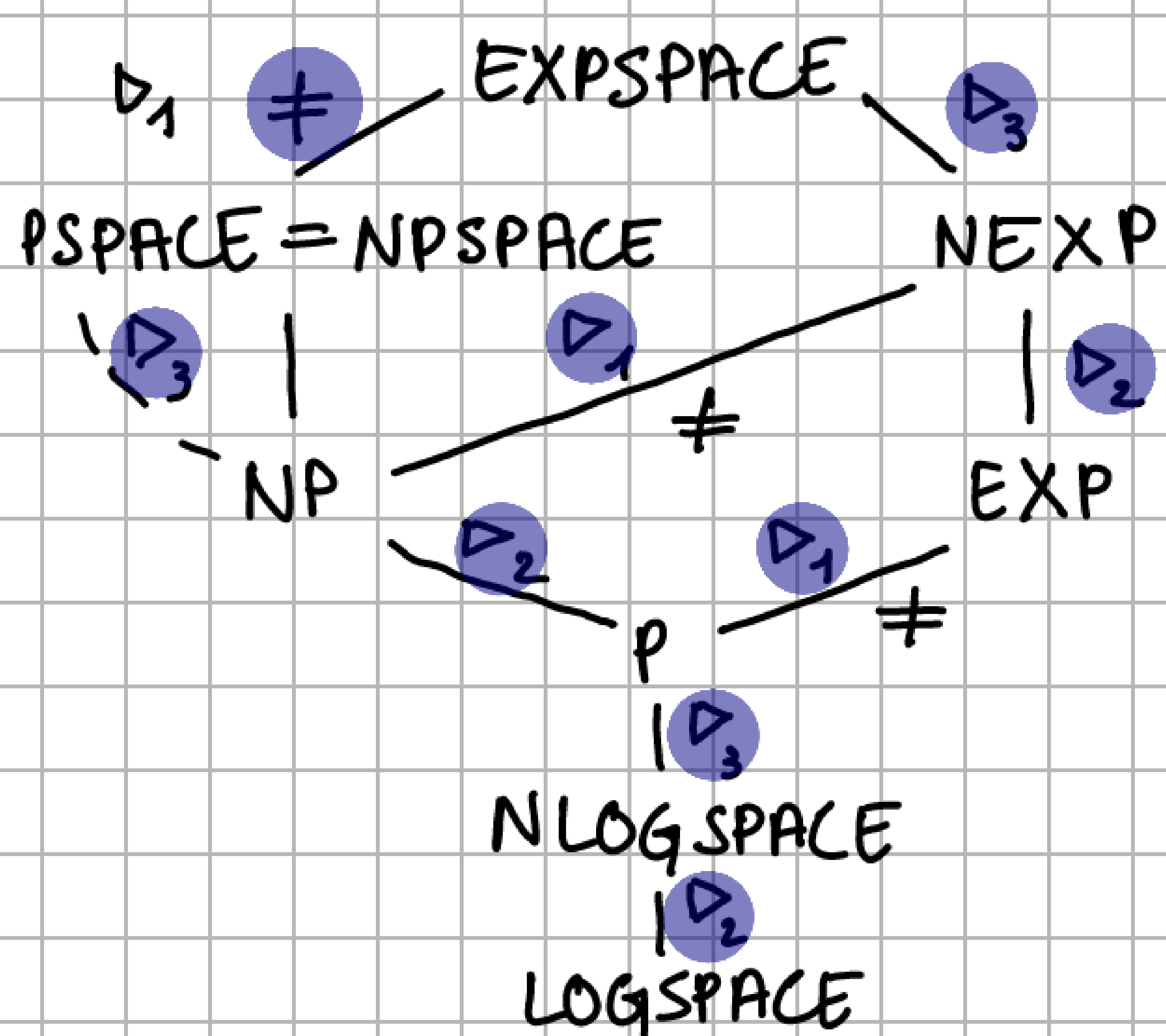
$$NSPACE(s) \subseteq DSPACE(s^2)$$

Come ulteriore risultato ottengo quindi

NPSPACE = DSPACE

~~IL TEOREMA di IMMERMAN e SZELEPSCÈNYI~~

INCLUSIONI tra CLASSI di COMP.



- stiamo considerando la stessa risorsa?
  - ▷1 teorema della gerarchia
  - ▷2 (da MdT a MdTN) si considera come caso speciale di una
- risorse ≠
  - ▷3 teorema della simulazione del nondeterminismo (esplosione esponenziale)

IL TEOREMA della PROIEZIONE

Dato un linguaggio $A \subseteq \Sigma^* \in NP$ sse $\exists$ ling. $B \in P$ e un polinomio p t.c. $\forall x \in \Sigma^*$

$$x \in A \iff \exists y,\ |y| \leq p(|x|) \wedge \langle x, y \rangle \in B$$

Un ling. A sta in NP sse E un ling. B verificabile in tempo polinomiale, formato da coppie, x e il suo certificato y.
Verificare che y certifichi correttamente che $x \in A$ deve essere efficiente (in tempo polinomiale) e anche la sua dimensione sia al più polinomiale nella dimensione del dato x.

CALCOLABILITÀ
Un problema è r.e. sse può essere visto come proiezione esistenziale di un problema ricorsivo.

Gli insiemi che stanno in NP prevedono una ricerca ($\exists$) all'interno di un problema polinomiale
(implementazione dell'algoritmo di semidecisione)
e.g. l'insieme è una cricca? → cerco il certificato

Il certificato è l'insieme delle scelte fatte dalla MdTN.

CLASSI di COMPLESSITÀ per FUNZIONI

Data una funzione $t : \mathbb{N} \to \mathbb{N}$

- $FTIME(t) := \{ f : \Sigma^* \to \Sigma^* \mid \exists$ MdT M, $f = f_M \wedge t_M \in O(t) \}$
- $FSPACE(t) :=$ 〃 〃 〃 $\wedge\, s_M \in O(t) \}$

- $FP := \bigcup_{c \in \mathbb{N}} FTIME(n^c)$
- $FLOGSPACE := FSPACE(\log)$

Lemma $FLOGSPACE \subseteq FP$

RIDUCIBILITÀ

Siano $A, B \in \Sigma^*$ due linguaggi.

- Diremo che A è riducibile in tempo polinomiale a B,

  $A \leq_p B$

  se $\exists f \in FP$ t.c. $\forall x \in \Sigma^*$, $x \in A \iff f(x) \in B$

- Analogamente, A è riducibile in spazio logaritmico a B

  $A \leq_L B$

  se $\exists f \in FLOGSPACE$ che realizza l'equazione precedente.

CHIUSURA per COMPOSIZIONE

Le classi FP e FLOGSPACE sono chiuse per composizione.

Lemma

Le relazioni $\leq_p$ e $\leq_L$ sono dei preordini

Indicheremo con $\equiv_p$ e $\equiv_L$ le equivalenze indotte.

Ovvero

$$A \leq_p B \wedge B \leq_p A \iff A \equiv_p B$$

CHIUSURA per RIDUCIBILITÀ

Sia C una classe di linguaggi e $\leq$ un preordine.

Diremo che C è chiusa rispetto a $\leq$ se

$$A \leq B \wedge B \in C \implies A \in C$$

Lemma
- le classi
  P, NP, PSPACE
  sono chiuse rispetto a $\leq_p$
- le classi
  LOGSPACE, NLOGSPACE, P, NP, PSPACE
  sono chiuse rispetto a $\leq_L$

PROBLEMI ARDUI e COMPLETI
- B è C-arduo rispetto a $\leq$, se $\forall A \in C \Rightarrow A \leq B$
- B è C-completo " , se $B \in C$ e B è C-arduo

e.g. se un linguaggio è NP-completo, tutti i linguaggi $\in$ NP sono riducibili a lui (e lui stesso $\in$ NP)
e.g. $\forall$ problema $\in$ P è NP-arduo, ma non NP-completo

IL PROBLEMA LIMITATO della FERMATA (Bounded Halting Problem)

$$BHP := \{ \langle \underbrace{x}_{\text{codice della macchina}}, \underbrace{y}_{\text{input}}, \underbrace{0^t}_{\text{stringa di 0 lunga } t \to \text{timeout}} \rangle \mid \text{MdTN } M_x \text{ accetta } y \text{ con } time_{M_x}(y) \leq t \}$$

Questo problema è NP-completo

P vs. NP

sembra che non sia possibile, ma ancora si cerca

Lemma

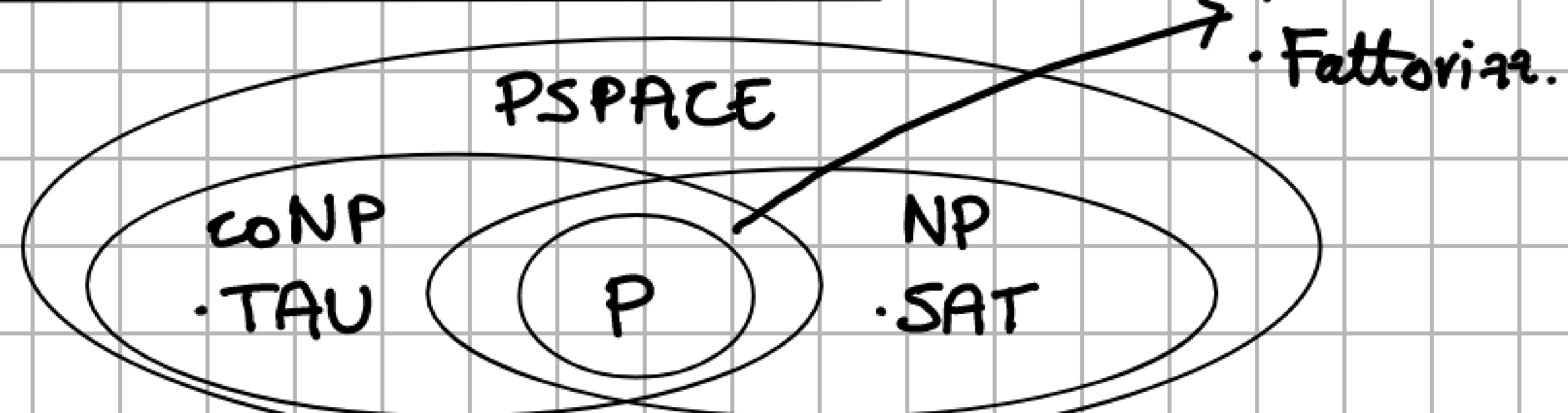
Se per qualche problema NP-completo A si ha $A \in P$, allora $P = NP$.
$P \subseteq NP$, quindi dobbiamo dimostrare l'inclusione opposta.
Se $B \in NP$, per la NP-completezza di A si ha $B \leq_p A$.
Siccome P è chiusa per la riducibilità in tempo polinomiale,
per ipotesi $A \in P$, anche $B \in P$.

Lemma

Se $P = NP$ allora ogni problema non banale $A \in NP$ è NP-completo.

ANALOGIE tra CALCOLABILITÀ e COMPLESSITÀ

R.E.	NP
Ricorsivo	P
$\leq_M$	$\leq_p$
K	SAT



Problema della soddisfacibilità

SAT → insieme delle formule soddisfacibili (hanno almeno 1 sol)
TAU → insieme delle formule tautologiche (sempre vere)
(non posso certificare tutte le soluzioni, ma solo quelle sbagliate)
il complementare di una formula non soddisfacibile è TAU
(mai soddisfacibile ⟷ sempre soddisfacibile)

! DIFFERENZE

- si sa che R.E. ≠ Ricorsivo ($NP \overset{?}{\neq} P$)
- RE ∩ coRE = Ricorsivo ⟶ NP ∩ coNP sembra ≠ P (leggermente + grande)
- si conoscono insiemi R.E. non completi

Fattorizzazione

- sfrutto l'unicità della fattorizzazione
- posso certificare anche la risposta negativa → sta in coNP

coNP → problemi per cui posso dare una certificazione del negato ($\notin$)

IL PROBLEMA della SODDISFACIBILITÁ

Sia $\Sigma := \{0, 1, \neg, \wedge, \vee, \rightarrow, (,), [,]\}$, $SAT \subseteq \Sigma^*$ è l'insieme delle formule booleane soddisfacibili, ovvero le formule vere rispetto ad un qualche assegnamento dei simboli proposizionali.

e.g. $[0] \rightarrow [1] \in SAT$; $[0] \wedge \neg [0] \notin SAT$

SAT è NP-completo !

Data una formula, $\exists$ un certificato che mi consente di verificare la soddisfacibilità in tempo polinomiale ($\in NP$).

DIM.

- è arduo ($\forall$ problema $\in NP$ è riducibile a SAT)

  Devo trovare una funzione ϕ che dato x, $x \in L \Leftrightarrow \phi(x) \in SAT$

  Sia $L \in NP$ ($\exists$ MdTN M che in $time_M(x) \leq p(|x|)$ decide se $x \in L_M$)

  $x \in L \equiv x$ è riconosciuto dalla macchina

  $\Updownarrow$

  $\exists$ una computazione non deterministica che arriva ad accett. in tempo $\leq p(|x|)$ (riconoscimento)

  La $\phi(x)$ deve codificare nel ling. della logica proposizionale la computazione non-det. di M su x.

  $\downarrow$

  sequenza di config. della macchina

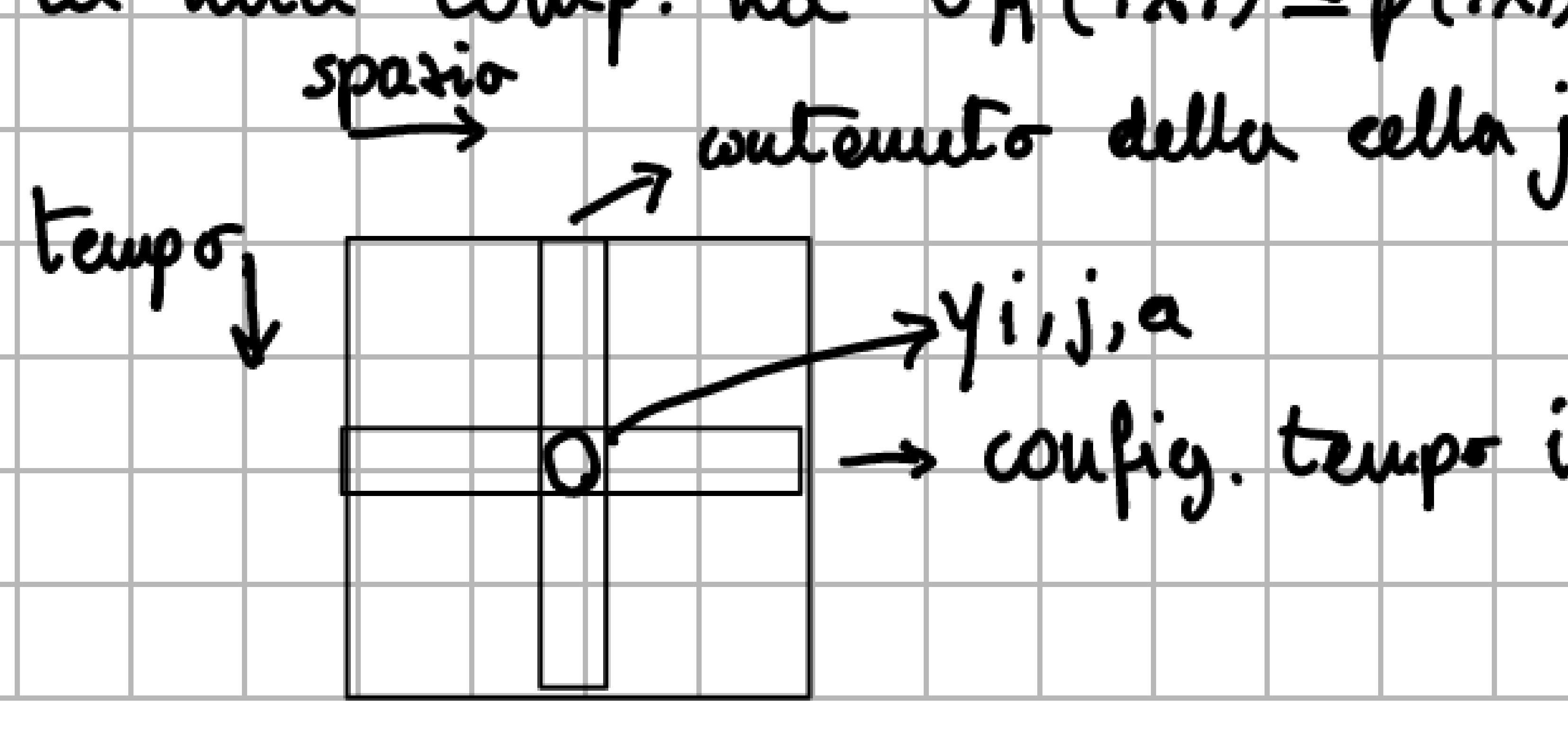
TEOREMA di COOK

La config. viene salvata come il nastro di lavoro con in più la posizione della testina salvata come q (stato interno) sulla cella puntata $\rightarrow$ (coppia : ⟨stato, char in lettura⟩)

Una computazione è una sequenza di config. che porta dalla config. iniziale a quella finale.

Con il T. tempo-spazio so che se la mia comp. ha $t_M(|x|) \leq p(|x|)$ allora anche $s_M(|x|) \leq p(|x|)$

La matrice ha dimensione $p(n) \times p(n)$

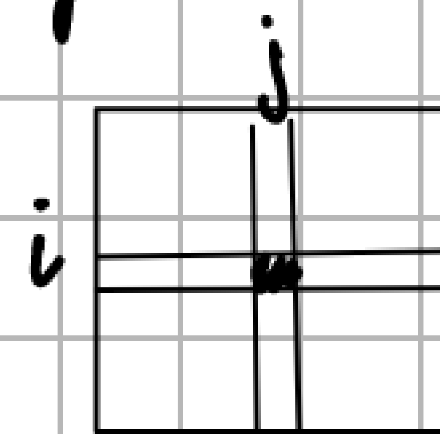


$y_{i,j,a}$ = true $\iff$ il carattere j della i-esima configurazione contiene il carattere a.

↳ questo è il modo di codificare x

VINCOLI

- Un carattere per cella

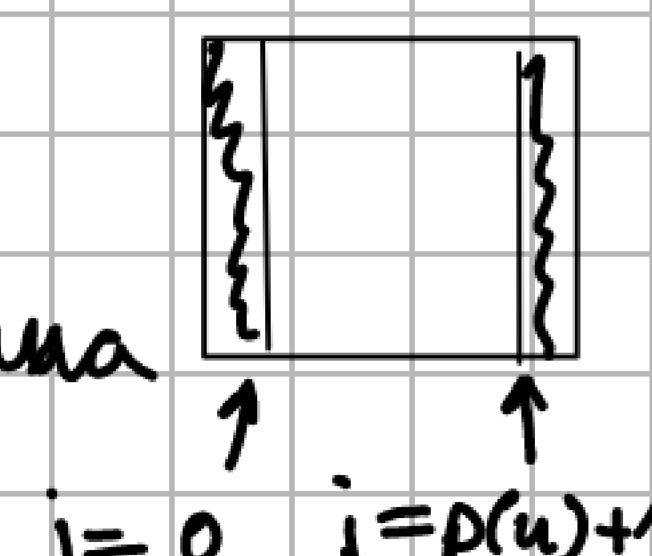


che solo uno dei due, non entr.

$$\Phi_0 := \bigwedge_{i=0}^{p(n)} \bigwedge_{j=1}^{p(n)} \left(\bigvee_{a\in\Gamma'} y_{i,j,a} \wedge \bigwedge_{a,b\in\Gamma' \wedge a\neq b} (\neg y_{i,j,a} \vee \neg y_{i,j,b}) \right)$$

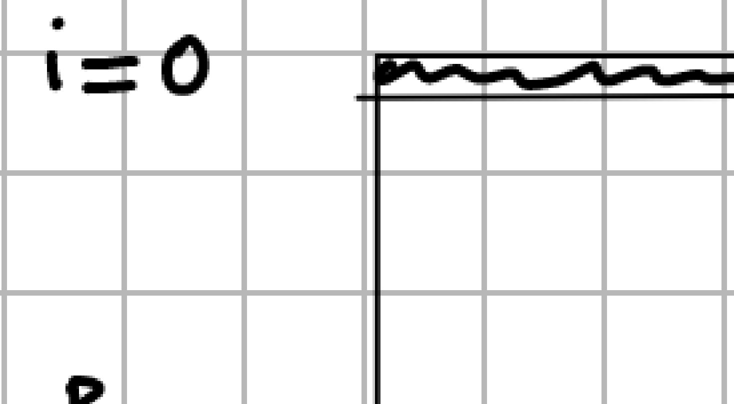
$\forall i$ $\forall j$ ↳ che almeno per 1 a valga

- Bordi invalicabili:
forziamo il carattere blank su prima e ultima colonna



$$\Phi_1 := \bigwedge_{i=0}^{p(n)} (y_{i,0,B} \wedge y_{i,p(n)+1,B})$$

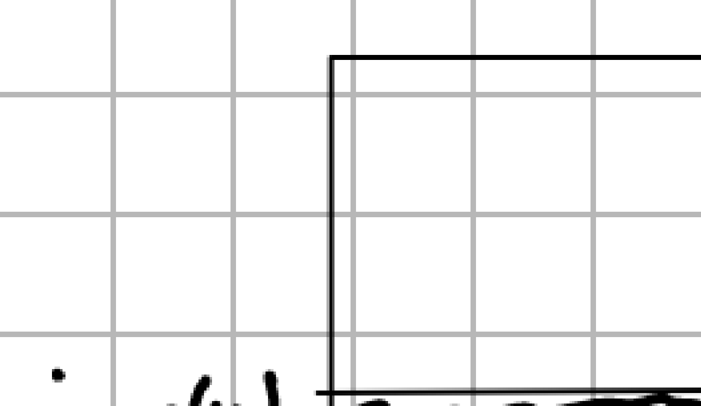
- Configurazione iniziale
  - il nastro deve contenere l'input $x = x_1 \dots x_n$, con la porzione restante B
  - la testina è sul carattere x_1



$$\Phi_2 := y_{0,1,(q_0,x_1)} \wedge \bigwedge_{j=2}^{n} y_{0,j,x_j} \wedge \bigwedge_{j=n+1}^{p(n)+1} y_{0,j,B}$$

i, j

- Configurazione finale
lo stato interno della macchina all'istante $t = p(n)$ deve essere uno degli stati di accettazione finali F



$$\Phi_3 := \bigvee_{j=1}^{p(n)} \bigvee_{\underbrace{a\in F\times\Gamma}_{\text{è una coppia } \langle \text{stato, char} \rangle}} y_{p(n),j,a}$$

almeno un j

- PASSAGGIO tra CONFIGURAZIONI
La porzione del nastro non adiacente alla testina al tempo $t \equiv t+1$ con quella

$$\Phi_4 := \bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=1}^{p(n)} \bigwedge_{a,b,c\in\Gamma} (y_{i,j-1,a} \wedge y_{i,j,b} \wedge y_{i,j+1,c} \rightarrow y_{i+1,j,b})$$

• EVOLUZIONE attorno alla TESTINA

$$\Phi_5 := \bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=1}^{p(n)} \bigwedge_{(q,a)\in Q\times\Gamma} \Delta_{q,a,i,j} \qquad \begin{matrix}(q',a',R)\\(q'',a'',L)\end{matrix}$$

$\Delta_{q,a,i,j}$ descrive le possibili evoluzioni della configurazione al passo i con una mossa non deterministica

SUMMING UP
La congiunzione:

$$\Phi_X := \Phi_0 \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4 \wedge \Phi_5$$

porta a

$$x \in L \iff \Phi_x \in SAT$$

Resta da appurare la complessità computazionale di $f: x \mapsto \Phi_x$
È evidente che tutte le formule $\Phi_i \in O(p)$ in tempo e quindi

$f \in FP$, si può dimostrare che $f \in LOGSPACE$

$NP \cap coNP \supseteq P$ ma si congettura che l'intersezione sia più grande

Mancano però tecniche per dimostrare l'inclusione stretta

TECNICHE per DIMOSTRARE NP-ARDUO
Due tecniche generali per dimostrare che A è NP-arduo:

- dimostrazione diretta → uso la dimostrazione e cerco di dimostrare $\underbrace{\forall B \in NP}$ DIFFICILE
  $B \leq_p A$
  ↳ Teorema di Cook → creare MdTN
- dimostrazione indiretta → scelgo A' che so essere NP-arduo
  $A' \leq_p A$
  dimostro quindi che A è altrettanto complicato quanto A' (a meno di polinomi)
  FACILE si può scegliere (tra gli innumerevoli problemi NP-completi) un problema conveniente

SPECIALIZZAZIONI di SAT

Come ottenere formule con forma canonica? Una di queste è la:

- forma normale congiuntiva (cnf) ($\exists$ anche la disgiuntiva)
  l'idea è di propagare la negazione verso l'intero e arrivare a una congiunzione ($\wedge$) di disgiunzioni ($\vee$) di letterali (simbolo atomico con eventuale negazione).
  Utilizzando De Morgan è sempre possibile spostare la disgiunzione dalla radice dell'albero a un livello inferiore

  una formula cnf è quindi un insieme di clausole
  e.g.
  $$\{\{\neg A, \neg B\}, \{A, \neg B\}, \{B\}\} = (\neg A \vee \neg B) \wedge (A \vee \neg B) \wedge B$$

SAT a CLAUSOLE è NP-COMPLETO

Con i nuovi vincoli mantengo la NP-completezza?
Ci basta ridurre SAT verso SAT a clausole, ovvero serve definire una trasformazione $\varphi \rightsquigarrow \varphi_c$ tale che:

per RIDURRE
! definire f che fa il job

- la trasformata φ_c sia a clausole
- φ_c sia soddisfacibile sse φ lo era
- la trasformazione prenda tempo polinomiale

in generale bisogna stare attenti

N.B. La forma normale congiuntiva potrebbe esplodere in spazio, per via di De Morgan.

$\exists$ una trasformazione più astuta che preserva il vincolo polinom.

↳ quindi è ancora NP-completo (non viene semplificato dai $\vee$)

Possiamo aggiungere ulteriori vincoli riguardo al numero di letterali in ogni clausola: nSAT ha n letterali per clausola.
Casi interessanti sono 2SAT (polinomiale!) e 3SAT.

TEOREMA di COOK per 3SAT

3SAT è NP-completo → riducendo SAT a clausole verso 3SAT

non preservo l'equivalenza
$\exists$ ma la soddisfacibilità

esattamente 3 clausole !!

RIASSUMENDO:
- SAT: NP-completo
- SAT a clausole: "
- 3SAT: "
- 2SAT: Polinomiale

IL PROBLEMA del RICOPRIMENTO

Dato $G=(V,E)$ un ricoprimento $V' \subseteq V$ è un sottoinsieme t.c.:

$$\forall (u,v) \in E,\ u \in V' \vee v \in V'$$

Il problema decisionale del ricoprimento consiste nel decidere se G ammette V' con cardinalità $|V'| \leq K$

COMPLESSITÀ del RICOPRIMENTO (Vertex Cover)

$(G,K) \in VC$ è NP-completo (dato che è di facile verifica, ovvio $\in NP$)

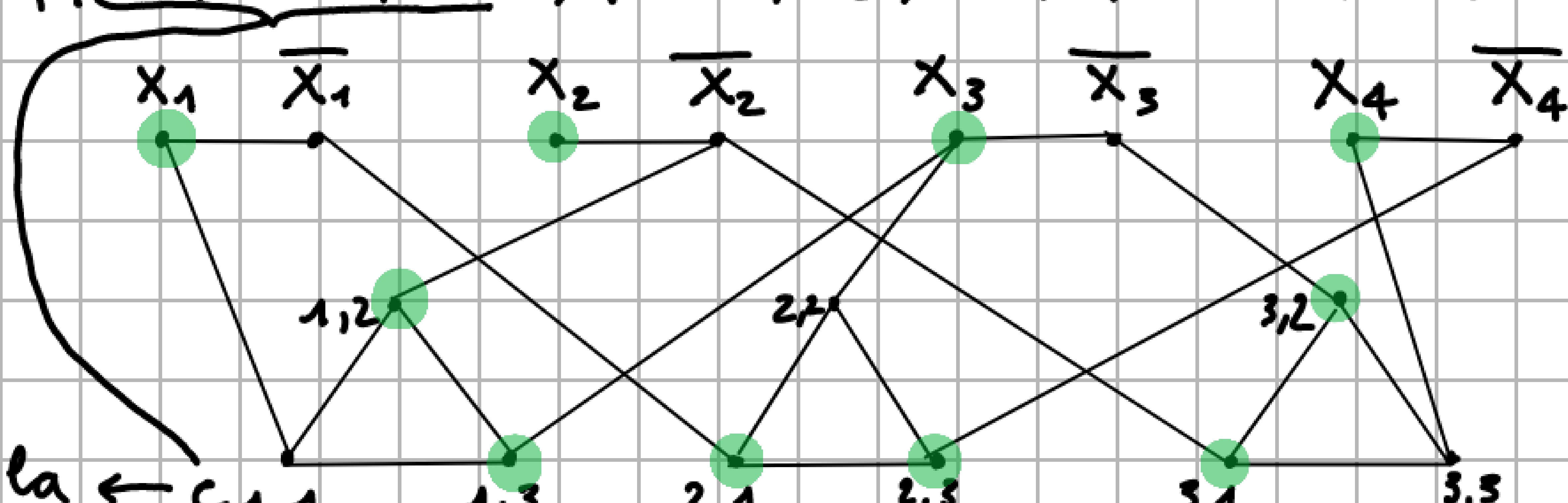
rimane da dimostrare sia NP-arduo → $3SAT \leq_p VC$

facendo vedere che

dobbiamo definire una trasformazione che prenda una formula e la traduca in una coppia (G, K), considerando che se la formula iniziale era soddisfacibile, allora la coppia $\in VC$; (e viceversa)
la seconda cosa che devo dimostrare è che la mia trasformazione sia al più polinomiale in tempo rispetto alla dim. dei dati di partenza (formula)

3SAT vs RICOPRIMENTO (e.g.) 1° es. importante di riduzione

$F = \{\{X_1, \neg X_2, X_3\}, \{\neg X_1, X_3, \neg X_4\}, \{\neg X_2, \neg X_3, X_4\}\}$



clausola ← $c_{1,1}$

$K = n + 2m$

n° delle variabili

n° delle clausole

F è soddisfacibile sse $\exists$ ricoprimento $S \mid |S| \leq n+2m$
(devo per forza prenderne 2 $\forall c$ e 1 $\forall X_i$)
↳ altrimenti non avrei il ricoprimento

trovo quindi una trasformazione di un input per un problema nell'input di un altro (assicurandomi la coerenza dell'appartenenza)

INSIEME INDIPENDENTE e CRICCA sono NP-completi in quanto equivalenti e riducibili al problema del ricoprimento

RIDUZIONE da SAT a cammino HAMILTONIANO

Chiedersi se un grafo ammette un cammino semplice che tocca tutti i nodi del grafo (che deve essere connesso)

- qual è lo spazio in cui vado a cercare il mio problema?
- ho un modo per certificare la correttezza?

] se sì sta in NP

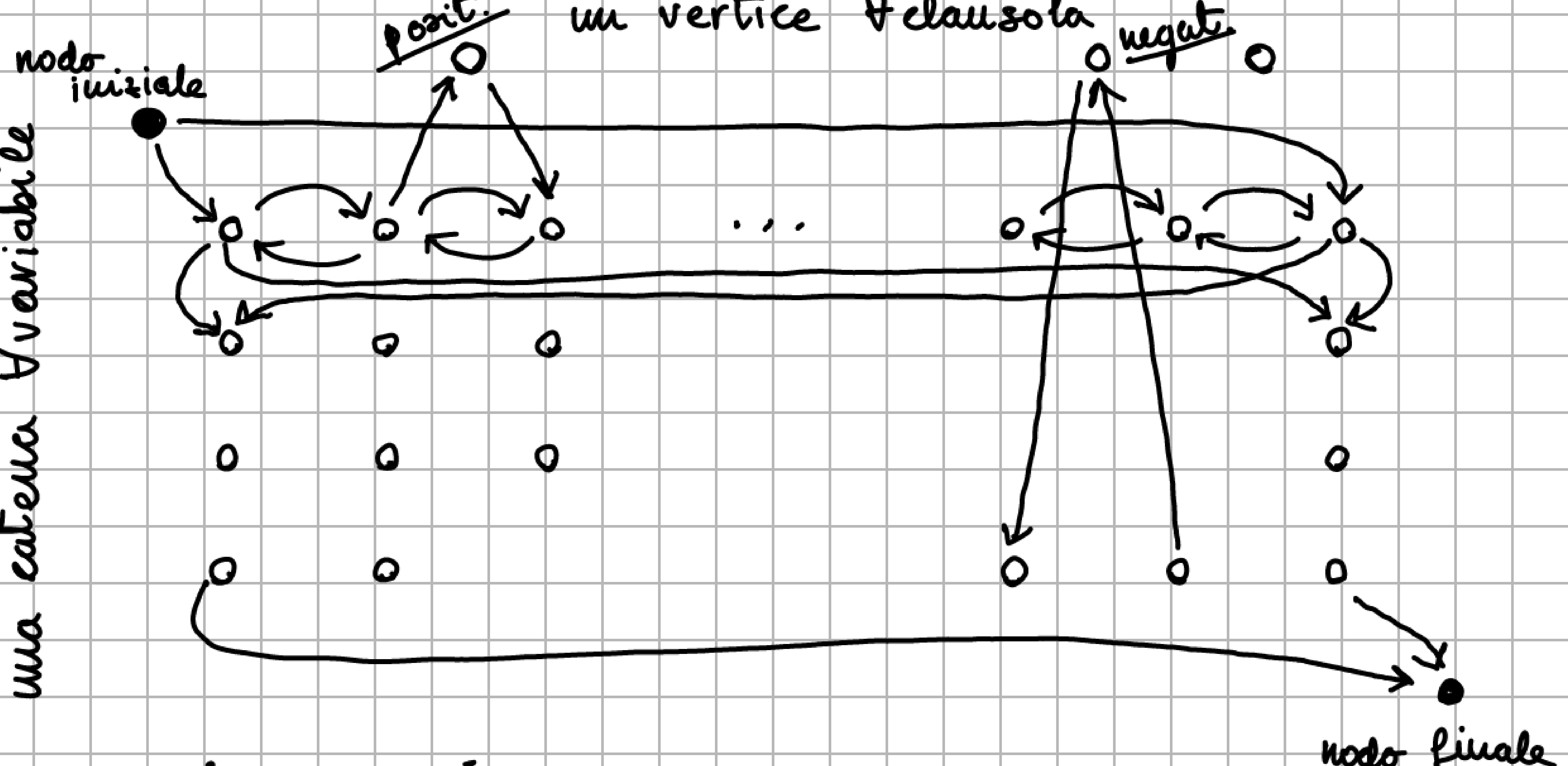
Suppongo che il problema sia in forma di clausole $C_1 \dots C_m$
$\forall X_i$ associo una catena di lunghezza $2m+2$ $[0; 2m+1]$ → nodi inizio/fine

$$X_0 \rightleftarrows X_1 \rightleftarrows \dots \rightleftarrows X_{2m} \rightleftarrows X_{2m+1} \qquad X = \underset{\text{false}}{\overset{\text{true}}{\rightleftarrows}}$$

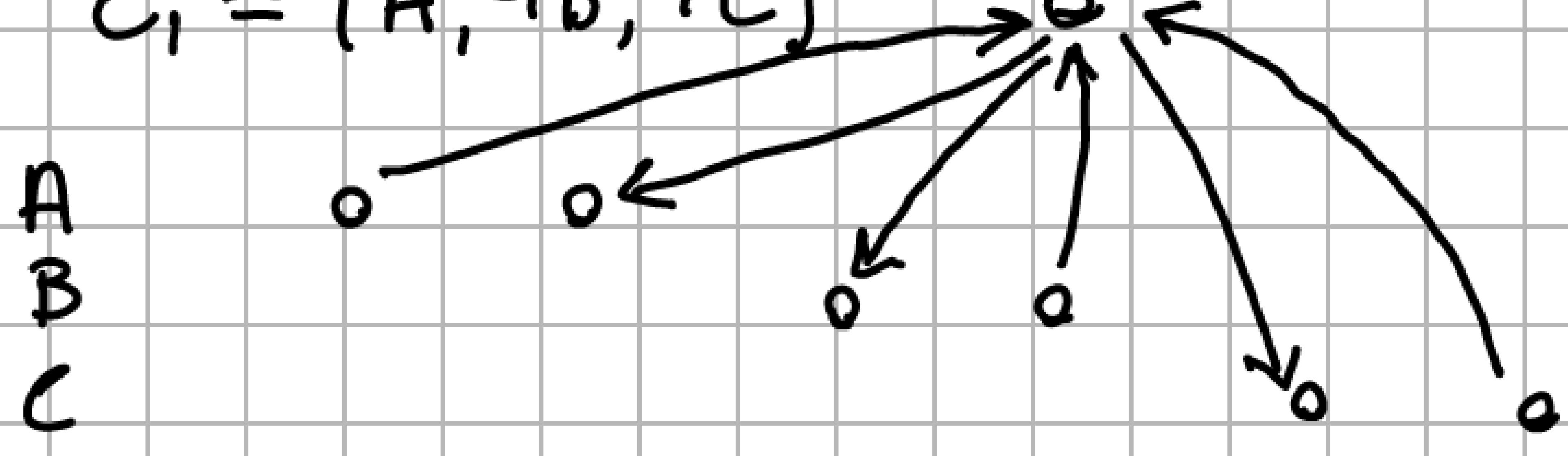
Devo trasformare una formula in un grafo e far sì che:
- la form. $\in$ SAT sse G ottenuto ammette cammino Hamiltoniano

Come collegare le variabili?

(colonna)
un vertice $\forall$ clausola

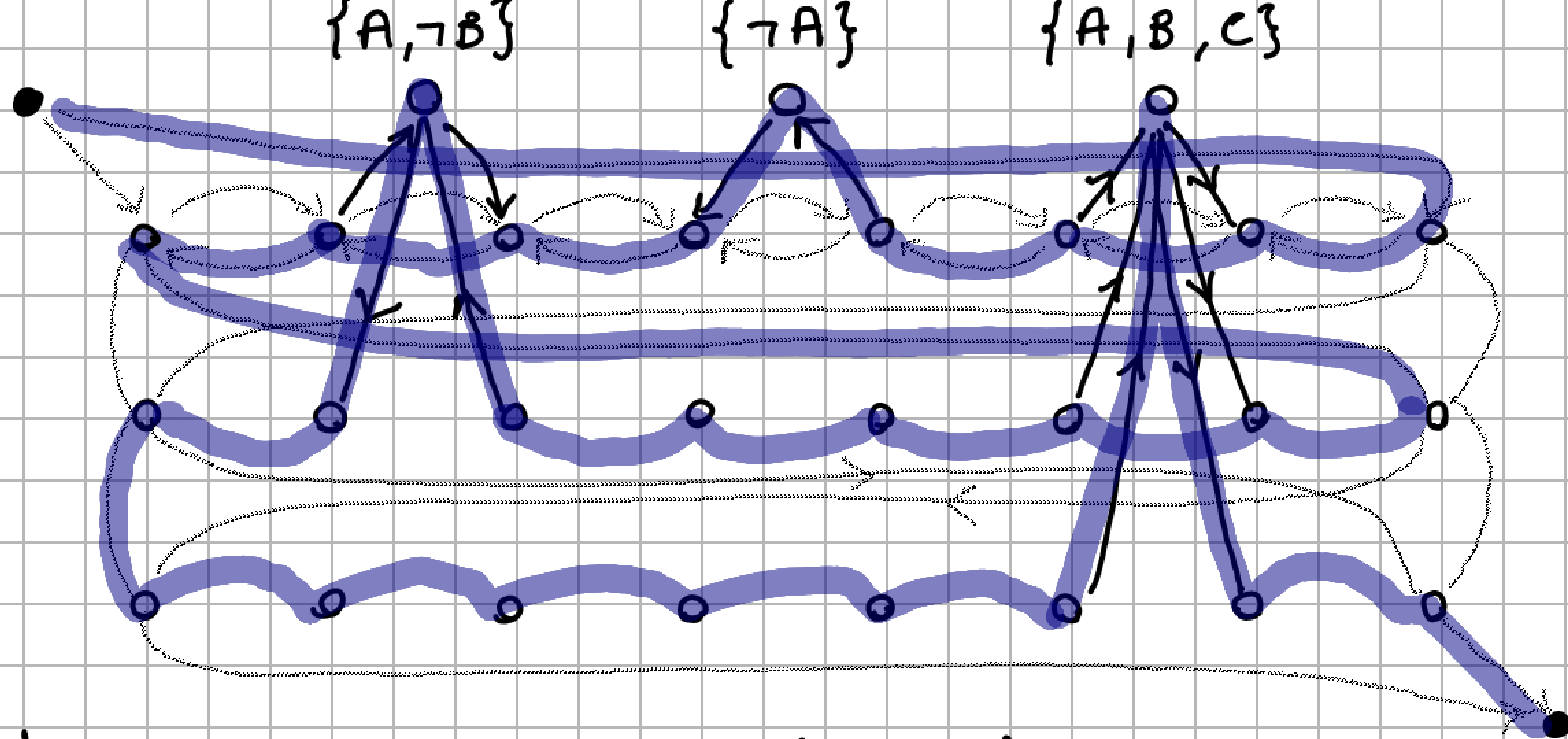


$C_i = \{A, \neg B, \neg C\}$



quindi Ci può essere attraversato o se A è att. pos., o B neg., o C neg.

es.



$|C_i| = 3 \Rightarrow$ catene lunghe 8 $(2m+2)$
↳ n° di catene

← sx ← sx → dx

$F = \{\{A, \neg B\}, \{\neg A\}, \{A, B, C\}\}$ $F \in SAT \Rightarrow \{\neg A, \neg B, C\}$

Ho trovato un cammino Hamiltoniano (ho toccato ∀ nodo)
Su un G di questo tipo devo per forza attraversare una catena interamente, altrimenti non avrei più modo di passare per i nodi lasciati indietro.

non può essere l'identità
↑!!

CAMMINO vs. CICLO (Path vs. Cycle)
Per ridurre l'uno all'altro devo immaginare una trasformazione di grafi $G \rightsquigarrow G'$ tale che:

(path) (cycle)

- Ham-P $\leq_p$ Ham-C : G ammette P sse G' ammette C
- Ham-C $\leq_p$ Ham-P : G ammette C sse G' ammette P

La trasformazione non può dipendere dal fatto ci sia o meno un cammino in Ham-P perché deve essere polinomiale (e sappiamo essere NP-completo).
SOLUZIONE → si aggiunge un nodo collegato a tutti i nodi
[Ham-P ≤ Ham-C] (cammino ⟶ ciclo)
SOLUZIONE → clonare un nodo arbitrario i, collegare i a start e
[Ham-C ≤ Ham-P] i' a end $(adj(i') \equiv adj(i))$
(ciclo → cammino)

! IMPORTANTE

- cercare cammini $\geq K$: NP-completo, generalizza Ham-P
- cercare cammini $\leq K$: $\in P$, visita in larghezza

HITTING SET (generalizzazione di VC)

Sia S un insieme e $C = S_1, \dots, S_n$ una collezione di sottoinsiemi di S. $H \subseteq S$ è un Hitting Set di C se, $\forall i$

$$H \cap S_i \neq \emptyset$$

Dato C, determinare se $\exists H$ con $|H| \leq K$ è NP-completo
Per dimostrarlo verifico che sia una generalizzazione di VC, quindi

$$VC \leq_p \text{Hitting Set}$$

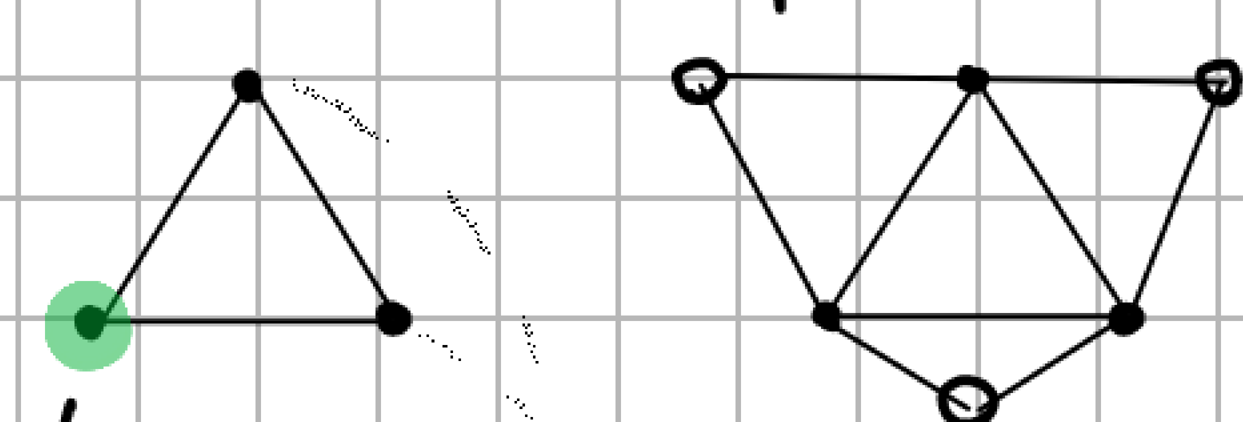
caso particolare di HS con sottoinsiemi formati da coppie $\equiv$ archi

Si può vedere S come un insieme di nodi e $C = S_1, \dots, S_n$ come un insieme di iperarchi su questi nodi.

INSIEME DOMINANTE (DOM)

Dato $G=(V,E)$, $V' \subseteq V$ è un insieme dominante se $\forall v \in V$ è possibile raggiungerlo da V' in al più un passo.
$\forall$ ricoprimento è un insieme dominante ma non viceversa.
È NP-completo, lo dim. riducendo il problema del ricoprimento ad esso.

$$VC \leq_p DOM$$



Dato $G=(V,E)$, $\forall (u,v) \in E$ aggiungo n_{uv} e lo connetto a u e v.

insieme dom $\neq$ ricoprimento

Se dall'insieme dominante voglio tornare al ricoprimento, ma se uno dei nodi che considero è un n_{uv}, posso sostituirlo con u o v

IL COMMESSO VIAGGIATORE (Travelling Salesman Problem)

Determinare l'∃ o meno di un ciclo di lunghezza K in un grafo di n città con distanze $d_{ij} \in \mathbb{N}^+$.
Il problema è in NP (spazio di ricerca grande → permutazioni)
(ho un modo per verificare la correttezza)

Ho delle euristiche (accorgimenti / ottimizzazioni) che posso mettere in atto. Questo è tipico dei problemi in NP.
Possiamo dimostrarne la completezza riducendo ad esso il problema del HAM-C. Dato G, genero G' totalmente connesso con:

$$d_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 2 & \sim \notin E \end{cases}$$

∃ un cammino hamiltoniano in G sse un cammino $\leq n+1$ in G'

PROGRAMMAZIONE INTERA (ILP – Integer Linear Programming)

Problema del stabilire se un insieme di equazioni e disequazioni a coeff. interi ammette soluzione intera.
Data la soluzione è banale verificarne la correttezza.
Lo spazio di ricerca è esponenziale al n° delle var
$\Big\} \in NP$
Per dimostrarne la completezza ho l'imbarazzo della scelta, in quanto ILP è molto duttile.

- SAT a clausole $\leq_p$ ILP
- TSP $\leq_p$ ILP

$DSPACE(\log n) \leq$

$DTIME(2^{c \log n})$

$$DSPACE(\log n) \leq DTIME(2^{c \log n}) = DTIME(2^{\log n^c}) = DTIME(\underset{P}{\underbrace{n^c}})$$

MEZZA CRICCA

Spesso, casi particolari di problemi NP-completi continuano ad esserlo. Un esempio tipico è quello di trovare una cricca di dimensione pari alla metà del n° di nodi.
Il problema generale della cricca può sempre essere ridotto a quest'ultimo. Infatti se è < della metà aggiungo densamente conn
è > n n 2 nodi isolati.
(e diventa metà)

KNAPSACK

Un caso particolare, noto come Subset Sum:
- dati n int. pos. $w_1, \dots, w_n$ e una somma tot W, decidere se $\exists\ I \subseteq \{1, \dots, n\}$ t.c.

$$(*) \quad \sum_{i \in I} w_i = W \qquad \text{è chiaro} \in NP$$

Per dimostrarne la completezza, riduco 3SAT ad esso.
Quindi la $F \in SAT$ sse $\exists I$ che rispetta $(*)$

3SAT $\leq_p$ KNAPSACK

Supponiamo di avere m clausole $c_0, \dots, c_{m-1}$ e n var. propos. $A_0, \dots, A_{n-1}$. $\forall A_i$ gli associo due numeri n_{A_i} e $n_{\overline{A}_i}$ corrispond. ai due valori di verità per A.
In particolare:
- le prime n cifre (meno significative) identificano via 1-hot encoding la variabile in questione
- le succ. m cifre indicano quali clausole sono soddisfatte dai rispettivi valori di verità della variabile. Nel numero n_L, la cifra $j+n$, per $0 \leq j < m$, è 1 se $L \in c_j$ e 0 altrimenti

es. $c_3 = \{A, \overline{B}, C\}$, $c_2 = \{\overline{A}, B, \overline{C}\}$, $c_1 = \{\overline{A}, \overline{B}, C\}$, $c_0 = \{A, B, \overline{C}\}$

numero		clausole				variabili		
n_A	=	1	0	0	1	0	0	1
$n_{\overline{A}}$	=	0	1	1	0	0	0	1
n_B	=	0	1	0	1	0	1	0
$n_{\overline{B}}$	=	1	0	1	0	0	1	0
n_C	=	1	0	1	0	1	0	0
$n_{\overline{C}}$	=	0	1	0	1	1	0	0
		c_3	c_2	c_1	c_0	C	B	A

N.B. non lavoriamo in base binaria
↓
devo sommare i vari n_i e devo assicurarmi che non ci sia riporto

- numeri RIEMPITIVI

  $\forall c_j$ aggiungiamo due numeri che hanno cifra 1 nella pos. della clausola j e 0 per il resto. Avendo 4 c_i dobbiamo aggiungere 8 numeri

$n_{c_0'}$	=	0	0	0	1	0	0	0
$n_{c_0''}$	=	0	0	0	1	0	0	0
$n_{c_1'}$	=	0	0	1	0	0	0	0
$n_{c_1''}$	=	0	0	1	0	0	0	0
$n_{c_2'}$	=	0	1	0	0	0	0	0
$n_{c_2''}$	=	0	1	0	0	0	0	0
$n_{c_3'}$	=	1	0	0	0	0	0	0
$n_{c_3''}$	=	1	0	0	0	0	0	0

Abbiamo un totale di 6+8 numeri

Il numero target è $W = 3333111$ $\quad F \in SAT$ sse la somma dei numeri scelti dà questa somma.

Quindi, con $\{A, \overline{B}, \overline{C}\}$ avrei 2112111

mi manca quindi 1221000

$n_{c_3'}$

$n_{c_2'} + n_{c_2''}$

$n_{c_1'} + n_{c_1''}$

$n_{c_0'}$

- abbiamo appena dimostrato il "se" $\Rightarrow$
  ovvero, se $F \in SAT$, è possibile scegliere dei numeri per arrivare a W

- il contrario, $\Leftarrow$, è ancora vero: se la somma mi dà W, allora $F \in SAT$.

COMPLESSITÀ RELATIVIZZATA
MACCHINE ad ORACOLO
Una MdTN con Oracolo è definita come una MdTN:

stati finali, blank, n° nastri

$$M = (Q, q_0, q?, q+, q-, F, \Sigma, \Gamma, B, k, \delta)$$

- è dotata di un nastro aggiuntivo, detto di interrogazione
- ha 3 stati speciali: $q?, \underbrace{q+, q-}_{\text{stati di risposta}} \in Q \setminus F$
  - $q?$ → stato di interrogazione
- δ non è definita sullo stato di interrogazione

Per interrogare l'oracolo si scrive la query (y) sul nastro di interr. e si entra nello stato $q?$. L'oracolo in $O(1)$ mi risponde facendomi entrare in uno dei due stati di risposta $q+ \vee q-$
$y \in O$ ← $q+$, $q-$ → $y \notin O$

L'Oracolo può essere visto come un linguaggio
Denotiamo con $L_O (f_M)$ il ling. accettato (funz. calcolata) dalla macchina M con oracolo O.

CLASSI di COMPLESSITÀ con ORACOLO
Sia C classe di complessità e $O \subset \Sigma^*$ un oracolo.
C^O è definita in modo simile a C, con la differenza che si considerano macchine ad oracolo O.
Ad es. P^{SAT} è l'insieme dei linguaggi che ammettono un algo di decisione polinomiale, ammesso di avere un oracolo per SAT.
Se inoltre C' è una c. di compl., allora

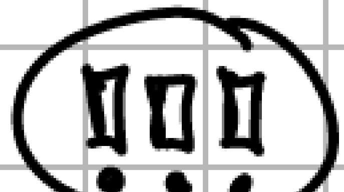
$$C^{C'} := \bigcup_{O \in C'} C^O$$ è la classe dei problemi che hanno complessità C ammesso di avere un oracolo per i problemi di complessità C'.

Ad esempio NP^{PSPACE} è l'insieme dei ling. riconosciuti da una MdTN mediante un oracolo relativo ad un problema in PSPACE.

ALCUNE PROPRIETÁ delle CLASSI con ORACOLO

Per tutti i linguaggi $A, B, C \in \Sigma^*$

- $A \in P^A$ ovvio, perché è l'oracolo a fare tutto il lavoro in $O(1)$
- $A \in P^B \Rightarrow A \in NP^B$ le MdT sono un caso particolare di MdTN
- $A \in NP^B \Rightarrow A \in NP^{\overline{B}}$ basta complementare invertendo $q+$ e $q-$
- $A \in P^B, B \in P^C \Rightarrow A \in P^C$ idea della composizione
  rimpiazzo la chiamata all'oracolo B con l'algoritmo polinomiale che interroga C. Dato che la <u>composizione</u> di polinomi continua a darmi un polinomio, so che continuerò a lavorare in tempo polinomiale (anche se ha grado maggiore)
- $A \in NP^B, B \in P^C \Rightarrow A \in NP^C$ dato che l'algoritmo per l'oracolo è polinomiale, posso rimpiazzarlo ottenendo ancora una computazione in NP



probabilmente: ho sempre il dubbio che $P = NP$ e tutto collasserebbe

- $A \in P^B, B \in NP^C \not\Rightarrow A \in P^C$ (né $A \in NP^C$)
- $A \in NP^B, B \in NP^C \not\Rightarrow A \in NP^C$

Ad esempio $TAU \in P^{SAT}$, $SAT \in NP^{\emptyset} \equiv NP$, ma $TAU \notin NP$ (probabilmente)

Lemma
- $P^P = P$ composizione di polinomi resta polinomiale
- $NP^P = NP$
- $NP^{PSPACE} = PSPACE$ da MdTN a MdT

Lemma

$NP^{P} = NP \subseteq P^{NP} \subseteq NP^{NP}$

segue dal fatto che $A \in P^{B} \Rightarrow A \in NP^{B}$

segue dal fatto che $A \in P^{A}$

NP e coNP

insieme dei problemi che hanno il complementare in NP

T.

$NP^{NP} = NP$ sse $NP = coNP$

DIM.

$\Leftarrow)$