Parte 3 - Teoria della Complessità

Si basa sulle risorse di TIME and SPACE. Il modello di riferimento è sempre la TM, e il tempo coincide col numero di passi.
Ma il numero di passi dipende dalla dimensione e tipologia dell'input!

- Sia M una DTM che termina sempre su ogni input. Il "RUNNING-TIME" o "TIME COMPLEXITY" di M è la funzione $f: \mathbb{N} \longmapsto \mathbb{N}$, tale che $f(n)$ è il numero massimo di passi che M impiega su qualunque input di dimensione pari ad n (CASO PEGGIORE). $\rightarrow$ "M runs in time $f(n)$".

- **Analisi Asintotica:**

Siano $f, g : \mathbb{N} \longmapsto \mathbb{R}^+$

- $f(n) = O(g(n))$ se $\exists\, c>0, n_0>0$ tali che: $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$

* $O(\log_2 n) = O\left(\frac{\log_{10} n}{\log_{10} 2}\right) = O(\log_{10} n)$ $\longrightarrow$ NON IMPORTA LA BASE DEL LOGARITMO!

* $2^{O(\log n)} = 2^{c \cdot \log_b n} = 2^{c \cdot \log_2 n / \log_2 b} = 2^{\log_2 n^d} = n^d$

$\Rightarrow$ $2^{O(\log n)} = O(n^d)$, for some d.

- Una funzione è POLINOMIALMENTE LIMITATA se $f(n) = O(n^d) = 2^{O(\log n)}$, per qualche $d>0$;

- Una funzione è ESPONENZIALMENTE LIMITATA se $f(n) \leq 2^{(n^d)}$, per qualche $d>0$.

- $f(n) = o(g(n))$ se $\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$

* se $f(n) = o(g(n)) \Rightarrow \forall c>0, \exists n_0 > 0$ tali che $\forall n \geq n_0 : f(n) \leq c \cdot g(n)$

* se $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$

• CLASSE DI COMPLESSITA' TEMPORALE:

Sia $t: \mathbb{N} \longmapsto \mathbb{R}^+$. La classe di complessita' temporale $\underline{TIME(t(n))}$ è la collezione di tutti i linguaggi DECIDIBILI da una DTM con 1 solo nastro in $O(t(n))$ passi; cioè con tempo d'esecuzione $O(t(n))$.

$TIME(1) \subseteq TIME(n) \subseteq TIME(n^2) \subseteq \dots \longrightarrow$ numero ∞ di classi temporali.

• Esempio:

$A = \{0^k 1^k \mid k \geq 0\}$ è decidibile. Sia M_1 una singletape DTM:

M_1 = "On input w:
1. Scan the tape and reject if a 0 is found to the right of a 1;
2. Repeat if both 0 and 1 remain on tape:
   3. Scan the tape and cross a single 0 and a single 1;
4. If neither a 0 or 1 remain on tape, then accept; oth. reject."

Sia $|w| = n \Rightarrow$ 1. richiede $2n$ passi
2. $\frac{n}{2}$ iterazioni $\Rightarrow$ 3. $\frac{n}{2} \cdot 2n$ passi
4. n passi

$$\Rightarrow O(2n) + O\left(\frac{n}{2} \cdot 2n\right) + O(n) = O(n^2 + 3n) = \underline{\underline{O(n^2)}} \rightarrow A \in TIME(n^2)$$

*Si può fare meglio? Sì $\rightarrow$ M_2 singletape DTM:

M_2 = "On input w:
- $O(n)$ 1. Scan the tape and reject if a 0 is found to the right of a 1;
- $O(\log n)$ 2. Repeat if both 0 and 1 remain on tape:
- $O(n)$ 3. Scan the tape and check the total number of 0's and 1's on tape; if it is odd, then reject;
- $O(n)$ 4. Scan the tape and cross every other 0 starting from the first 0;
  5. Continue the scanning and cross every other 1 starting from the first one;
- $O(n)$ 6. If neither 0 or 1 remain on tape, then accept; oth. reject."

$$Time = O(n) + O(\log n) \cdot (O(n)) \cdot 2 + O(n) = \underline{O(n \log n)}$$

$$\Rightarrow \underline{\underline{A \in TIME(n \log n)}}$$

• **TEOREMA**: Ogni linguaggio deciso da una singletape DTM in tempo $O(n \log n)$ è un LINGUAGGIO REGOLARE.

Infatti, A non è regolare. Tuttavia, usando una DTM con 2 nastri, si può' decidere in tempo lineare $O(n)$.

• **TEOREMA**: Sia $t(n) > n$. Allora, qualunque $t(n)$-time multitape DTM ha una singletape DTM equivalente, che esegue in $O(t^2(n))$ passi.

Sia M con K nastri ed S la singletape da costruire:

1) S modifica il proprio nastro, così da codificare il 1° nastro di M con l'input w e K-1 nastri vuoti:

$$w_1 \dots w_n \longrightarrow \hat{w}_1 \underbrace{\hat{\sqcup} \dots \hat{\sqcup}}_{k-1} \; w_2 \underbrace{\sqcup \dots \sqcup}_{k-1} \dots w_n \underbrace{\sqcup \dots \sqcup}_{k-1}$$

Deve aggiungere n volte K passi $\rightarrow O(Kn) = O(n)$;

2) Per ogni passo di M ($t(n)$ passi), S:

2.1) Scan the tape to discover the head position ($\hat{w}_i$) $\longrightarrow O(K \cdot t(n))$
↳ una volta per ogni step di M

2.2) Scan the tape to update all M tapes $\rightarrow O(K \cdot t(n))$

2.3) If any of the K tapes of M must be expanded, shift the contents of the tape to the right $\rightarrow K \cdot O(K \cdot t(n)) = O(K^2 t(n))$

$$\Rightarrow \text{time} = O(n) + t(n)\left(O(K \cdot t(n)) + O(K^2 t(n))\right) = O(n) + O(K^2 t^2(n))$$

Per ipotesi: $t(n) > n \Rightarrow \underline{O(t^2(n))}$ ■ ok!!!

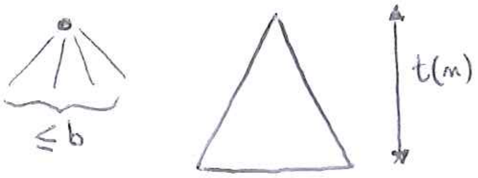
• **Def**:
Il tempo d'esecuzione di una NONDETERMINISTIC DECIDER TM, cioè che termina su ogni ramo di computazione, è la funzione:
$f: \mathbb{N} \longmapsto \mathbb{N}$, che il massimo numero di passi eseguiti $f(n)$ in ogni percorso, dalla radice ad una foglia dell'albero di computazione.

• **TEOREMA:**

> Sia $t(n) \geq n$. Allora, ogni $t(n)$-time nondeterministic singletape TM ha una singletape DTM equivalente, che esegue in tempo $2^{O(t(n))}$.

VISITA IN PROFONDITÀ (DFS): poiché so che ogni ramo termina.

Sia $b :=$ max grado di un nodo dell'albero computazionale:



$\Rightarrow$ # max di foglie $= b^{t(n)}$

$\Rightarrow$ 3-tape DTM: $O\left(b^{t(n)} \cdot t(n)\right)$

1-tape DTM: $O\left(b^{2t(n)} \cdot t^2(n)\right) = O\left(b^{2t(n) + 2\log_b t(n)}\right) = O\left(b^{t(n)}\right)$

Ma, $O\left(b^{t(n)}\right) = 2^{O(t(n))}$ $\longrightarrow$ TEMPO ESPONENZIALE! ■ ok!!!

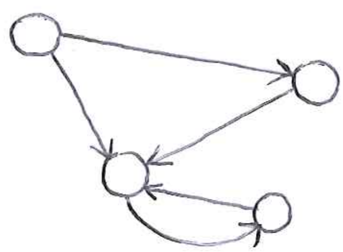
• **CLASSE POLINOMIAL-TIME P:**

La classe di linguaggi decidibili in TEMPO POLINOMIALE su DTM's è:

$$\boxed{P := \bigcup_{k \geq 0} \text{TIME}\left(n^k\right)}$$

* P è invariante rispetto a tutti i modelli di calcolo computazionali che sono polinomialmente equivalenti.

• **GRAFI:**

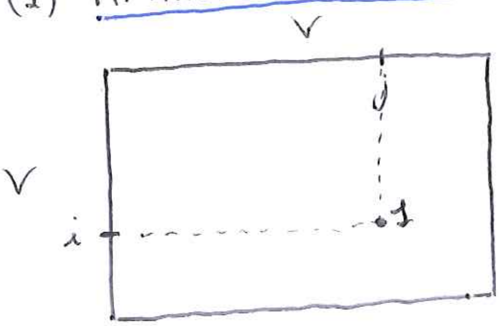


$G = (V, E)$, con $E \subseteq (V \times V)$

$|V| = n$; $|E| = m$

Rappresentabili tramite:

(1) MATRICI DI ADIACENZA:



$\forall i,j \in V : \begin{cases} M[i,j] = 1, & \text{se } (i,j) \in E \\ M[i,j] = 0, & \text{se } (i,j) \notin E \end{cases}$

$\rightarrow$ Peso della matrice $n \times n$ è $O(n^2)$

(2) LISTE DI ARCHI :

$|V|, \underbrace{(i,j) \ldots}_{|E|}$ $\rightarrow$ Costo d'occupazione $O(\log n + m \cdot 2 \cdot \log n)$

ma: $m \leq n^2$ $\Rightarrow$ $O(n^2 \log n)$

Quindi: $|<G>| = O(n^3)$

- PATH $\in P$:

$$\boxed{PATH = \{ <G,s,t> \mid G \text{ è un grafo diretto con un percorso diretto da } s \text{ a } t \} \in P}$$

wrong idea : brute-force search of the path $\rightarrow |V| = n$

$\Rightarrow$ # possibili percorsi : $O(n^n) = O(2^{n \log n}) \rightarrow$ EXPONENTIAL !

Idea : VISITA IN AMPIEZZA (BFS) :

M = "On input <G,s,t> :

$O(1)$ 1. Mark the node s;

$O(n)$ 2. Repeat until no new node can be marked :

$O(n) \cdot O(<G>)$ 3. Scan the edges and mark any unmarked node having an incoming edge from a marked node;

$O(1)$ 4. If node t si marked, then accept ; oth. reject."

Time = $O(n \cdot n \cdot <G>) = O(n \cdot n^2 \cdot n^2) = O(n^3) \Rightarrow$ PATH $\in P$! ole!!!

- RELPRIME $\in P$:

$$\boxed{RELPRIME = \{ <x,y> \mid x,y \in \mathbb{N} \text{ and } x \text{ and } y \text{ are relatively prime} \} \in P}$$

Anche qui, cercare i divisori brute-force è sbagliato, poiché il numero di divisori è EXP nella lunghezza della codifica.

Idea : $<x,y> \in RELPRIME \iff gcd(x,y) = 1$ $\rightarrow$ Algoritmo di Euclide "E"

E = "On input <x,y> :

1. Repeat until $y = 0$:
   2. $x \leftarrow x \bmod y$;
   3. Swap x and y $(x \longleftrightarrow y)$;
4. Output x."

R = "On input $<x,y>$:

1. Run E on input $<x,y>$;
2. If $E(<x,y>) = 1$, then accept; oth. reject."

$x \bmod y < y$; in realtà: $x \bmod y < \frac{x}{2}$;

- se $\frac{x}{2} \geq y \Rightarrow x \bmod y < y \leq \frac{x}{2}$
- se $\frac{x}{2} < y \Rightarrow x \bmod y = x - y < x - \frac{x}{2} = \frac{x}{2}$

L'operazione $x \longleftrightarrow x \bmod y$, mi fa diventare $x < \frac{x}{2}$

$\Rightarrow$ Ogni 2 cicli, si dimezza sia x che y $\Rightarrow$ # cicli $= \min(2\log_2 x, 2\log_2 y)$

$\Rightarrow$ E è polinomiale $\Rightarrow$ R è polinomiale $\Rightarrow$ RELPRIME $\in$ P ■ ok!!!

- **TEOREMA:** $\boxed{\text{Ogni linguaggio CFL è in P}}$

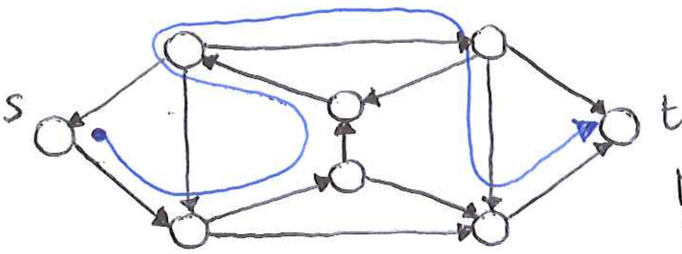
Il Parser di Earley per le CFG girare in
$\begin{cases} O(n^3) & \text{se CFG ambigue} \\ O(n^2) & \text{se CFG non ambigue} \\ O(n) & \text{se DCFG} \end{cases}$

$\Rightarrow$ gira in tempo polinomiale! $\Rightarrow$ Ogni CFL $\in$ P !!! ■ ok!!!

- **HAMPATH:**

HAMPATH = $\{<G,s,t>$ | G è un grafo diretto ed esiste un cammino Hamiltoniano tra i nodi $s,t \in V\}$

"Cammino Hamiltoniano" := percorso diretto che tocca ogni nodo del grafo esattamente 1 volta;



Algoritmo Brute-Force:

M = "On input $<G,s,t>$:

1. For all possible direct path from s to t:
   2. Check if the path goes through every node of G exactly once; if yes, then accept;
3. Reject."

HAMPAT è POLINOMIALMENTE VERIFICABILE !

- **POLINOMIALMENTE VERIFICABILE:**

Un problema è "POLINOMIALMENTE VERIFICABILE" se un CERTIFICATO di una qualunque istanza-sì del problema può essere verificato da una DTM in tempo polinomiale nella dimensione dell'istanza-sì.

$\Rightarrow$ HAMPATH è verificabile polinomialmente, poiché dato un percorso che lo risolve è facile verificarlo!

- **COMPOSITE:**

$$\boxed{\text{COMPOSITE} = \{ \langle x \rangle \mid x \in \mathbb{N} \text{ tale che } \exists\, p,q \in \mathbb{N},\ p,q > 1 : p \cdot q = x \}}$$

è polinomialmente verificabile.

Un CERTIFICATO per $\langle x \rangle$ è un divisore p per x.

Calcolare la divisione $\frac{x}{p}$ si può fare in tempo polinomiale nella dimensione di $\langle x \rangle$: $|\langle x \rangle| = O(\log_2 x)$

$\Rightarrow$ COMPOSITE è polinomialmente verificabile.

* **NOTA**: In realtà: COMPOSITE $\in P$, ma è complicato dimostrarlo.

- Consideriamo $\text{HAMPATH}^c := \overline{\text{HAMPATH}}$ (il complementare)

Per mostrare un certificato per $\overline{\text{HAMPATH}}$ dovrei far vedere che nessun cammino possibile soddisfa HAMPATH $\rightarrow$ Ma # cammini possibili è ESPONENZIALE!

$\Rightarrow$ HAMPATH^c non è polinomialmente verificabile.

Quindi, in generale:

$$\boxed{A \text{ polinomialmente verificabile} \not\Rightarrow A^c \text{ polinomialmente verificabile}}$$

- **Verificatore:**

Un verificatore di un linguaggio A è un ALGORITMO V tale che:

$A = \{ w \mid V \text{ accetta } \langle w, c \rangle \text{ per qualche stringa } c \}$

- POLYNOMIAL-TIME VERIFIER

È un VERIFICATORE che gira in tempo polinomiale nella lunghezza di w!
(NON in $|\langle w, c\rangle|$!)

* Definiamo $NTIME(t(n)) := \{ L \mid L$ is decided by a $O(t(n))$-time NTM $\}$

• CLASSE NP :

NP è la classe dei linguaggi che hanno dei polynomial-time verifier:

$$\boxed{NP := \bigcup_{k \geq 0} NTIME(n^k)}$$

"NP" sta per NONDETERMINISTICALLY POLINOMIAL-TIME.

HAMPATH $\in$ NP e COMPOSITE $\in$ NP

• Esempio: HAMPATH :

Vediamo una NTM per HAMPATH:

N_1 = "On input $\langle G, s, t\rangle$:

1. Guess and write on tape a sequence of n nodes $p_1, \dots, p_n$ $(n = |V|)$;
2. If any node on list is repeated, then reject;
3. If $p_1 \neq s$ or $p_n \neq t$, then reject;
4. For each $i = 1, \dots, n-1$:
   5. Check if $(p_i, p_{i+1}) \in G(E)$; if not, then reject;
6. Accept."

* Il NONDETERMINISMO nell'indovinare il cammino hamiltoniano è fondamentale!

ok!!!

• **TEOREMA**: $\boxed{\text{Un linguaggio } A \in NP \iff A \text{ è deciso da una NTM che gira in tempo polinomiale.}}$

Idea: convertire un verificatore in un decisore, e viceversa.

$\Rightarrow$) $A \in NP \Rightarrow \exists$ verificatore V in tempo polinomiale per A

$A = \{ w \mid V(\langle w,c\rangle) \text{ accepts in time } < n^k \}$

NTM

N = "On input w:
1. Nondeterministically guess a string c, with $|c| < n^k$;
2. Run V on $\langle w,c\rangle$;
3. If $V(\langle w,c\rangle)$ accepts, then accept; oth. reject."

Indovina nondeterministicamente il CERTIFICATO c! Lo verifica in tempo polinomiale $\Rightarrow$ N decide A in polinomial time! ok!

$\Leftarrow$) A ha una NTM che lo decide in tempo polinomiale.
Costruiamo il verifier V:

V = "On input $\langle w,c\rangle$:
1. Simulate N on w, using the symbols in c to choose among the nondeterministic steps;
2. If N(w) forced by c accepts, then accept; oth. reject."

E' giusto perché ogni NTM ha una DTM con 3 nastri equivalente:

1° nastro: input
2° nastro: passi dell'albero di computazione
3° nastro: nastro di lavoro

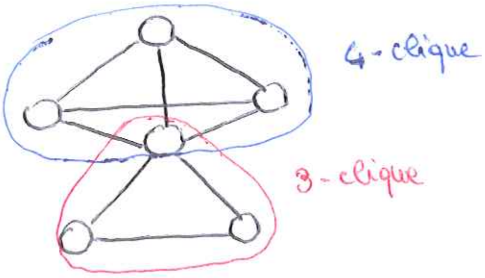
$\Rightarrow$ V è un polynomial-time verifier per A $\Rightarrow A \in NP$ ■ ok!!!

* **NOTA**: gli ALGORITMI dei VERIFICATORI V sono sempre DETERMINISTICI !!!

• CLIQUE $\in$ NP :

$$\boxed{\text{CLIQUE} = \{ <G, k> \mid G \text{ is a non direct graph with a } K\text{-clique} \} \in NP}$$

"K-clique" := Sottografo COMPLETO (esiste un arco tra ogni coppia di nodi) con K nodi.



Idea: un CERTIFICATO può essere una lista di K nodi;

Costruiamo il verificatore V:

V = "On input <G, K, c>:

1. Verify whether c encodes a subgraph of G with K nodes;
2. Verify whether G contains all edges between the nodes in c;
3. If both tests are satisfied, then accept; otherwise reject."

* ALTRIMENTI: costruiamo un NTM N che decide CLIQUE:

N = "On input <G, K>:

NTM

1. Nondeterministically guess a subset c of K nodes of G;
2. Verify whether G contains all edges between the nodes in c;
3. If the test is satisfied, then accept; otherwise reject."

* NOTA: accept or reject è riferito al singolo RAMO DI COMPUTAZIONE!

$\Rightarrow$ [CLIQUE $\in$ NP] !!!

■

ok!!!

• **TEOREMA:**

$$\boxed{\text{SUBSET-SUM} = \left\{ \langle S,t \rangle \;\middle|\; S = \{\{x_1, \dots, x_k\}\},\ x_i \in \mathbb{N} \text{ and for some } Y = \{\{y_1, \dots, y_e\}\} \subseteq S,\ \sum_{i=1}^{e} y_i = t \right\} \in NP}$$

S è un "MULTINSIEME": gli elementi possono anche essere ripetuti.
di MULTINSIEMI.

Un CERTIFICATO per S può essere il sottoinsieme di elementi di S la cui somma dia t.

V = "On input $\langle S,t,c \rangle$:
1. Test if c encodes a subset of S;
2. Test if the sum of the elements in c is t;
3. If both tests are satisfied, then accept; oth. reject."

* IN ALTERNATIVA:

NTM N = "On input $\langle S,t \rangle$:
1. Nondeterministically guess a subset c of S;
2. Test if the sum of the elements of c is t;
3. If the test is satisfied, then accept; oth. reject."

$\Rightarrow$ SUBSET-SUM $\in NP$ ■ ok!!!

• **OSSERVAZIONE:**

Consideriamo $\overline{\text{CLIQUE}}$ e $\overline{\text{SUBSET-SUM}}$: un certificato c per uno dei 2 dovrebbe contenere tutti i possibili sottoinsiemi → c è ESPONENZIALE!

$\Rightarrow$ Non possiamo dire se $\in$ o $\notin$ NP!

• **CLASSE CoNP:**

$$\boxed{CoNP := \{ L \mid \overline{L} \in NP \}}$$

$\Rightarrow$ $\overline{\text{CLIQUE}} \in CoNP$; $\overline{\text{SUBSET-SUM}} \in CoNP$

* Open Question: $NP \stackrel{?}{=} CoNP$

- Homework: $\boxed{NP \cap CoNP \neq \emptyset}$

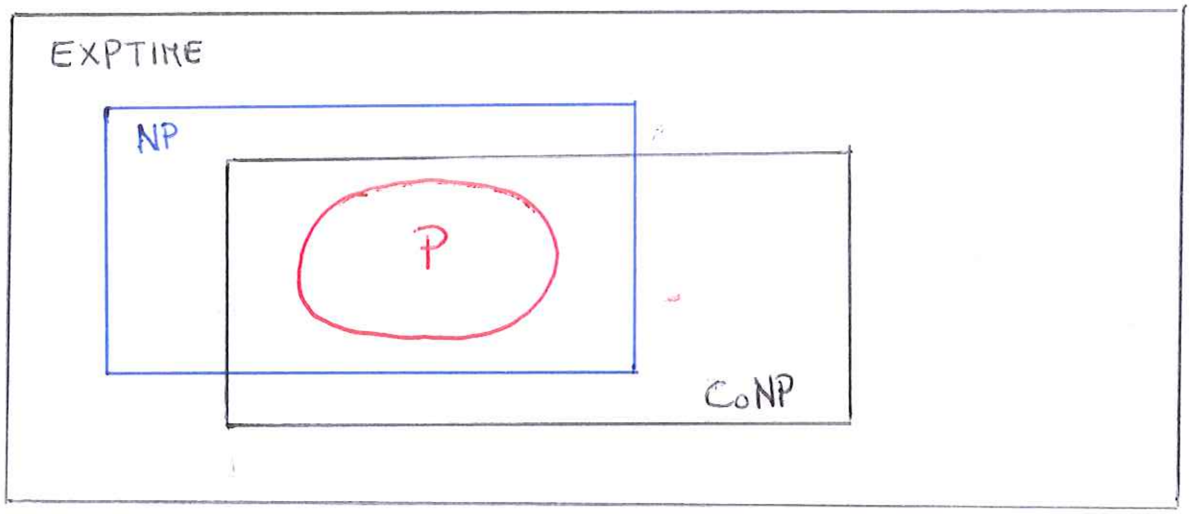
Basta ricordare che $P \subseteq NP$.

- **CLASSE EXPTIME:**

$$\boxed{EXPTIME := \bigcup_{k \geq 0} TIME(2^{n^k})}$$

* Abbiamo già dimostrato che una $O(t(n))$-time NTM ha un'equivalente $2^{O(t(n))}$-time DTM $\Rightarrow$ $\boxed{NP \subseteq EXPTIME}$

Dunque, considerato il dilemma $P \overset{?}{=} NP$, abbiamo la seguente situazione:



- **RIDUZIONE IN TEMPO POLINOMIALE:**

$f: \Sigma^* \longmapsto \Sigma^*$ è una POLYNOMIAL TIME COMPUTABLE FUNCTION se $\exists$ DTM che termina lasciando sul nastro $f(w)$ con w in input.

Un linguaggio A è POLYNOMIAL TIME REDUCIBLE ad un linguaggio B, se $\exists$ una POLYNOMIAL TIME COMPUTABLE FUNCTION f tale che:

$$\boxed{w \in A \iff f(w) \in B}$$

In tal caso, A è "RIDUCIBILE IN TEMPO POLINOMIALE" a B e si indica:

$$A \leq_p B$$

TEOREMA: $\boxed{\text{Se } A \leq_p B \ , \ B \in P \Rightarrow A \in P}$

Sia M che decide B in tempo polinomiale; sia T che calcola la riduzione $A \leq_p B$.
Costruiamo un decisore N per A in tempo polinomiale, che è composizione di M e T:

N = "On input w:
1. Run T on w and leave $f(w)$ on tape;
2. Run M on $f(w)$;
3. If $M(f(w))$ accepts, then accept, oth. reject."

Nel passo 2. M esegue in tempo polinomiale in $|f(w)|$.
Ma $|f(w)|$ è polinomiale, via T.
La composizione di polinomi è un polinomio $\Rightarrow$ $A \in P$ ■ ok!!!

SAT ∈ NP:

$\boxed{SAT = \{ \langle F \rangle \mid F \text{ is a satisfiable boolean formula} \} \in NP.}$

Un CERTIFICATO per SAT è una lista di asserzioni di verità sulle variabili di F, che soddisfano F.

V = "On input $\langle F, c \rangle$:
1. If c is not a list of k ~~truth~~ values, then reject;
2. If the number of variables in F is not k, then reject;
3. If c makes F true, then accept, otherwise reject."

Essendo solo controlli, gira in tempo polinomiale $\Rightarrow$ SAT ∈ NP ■ ok!!!

FORMA NORMALE CONGIUNTIVA (CNF):

- LITERAL: una variabile oppure la sua negazione: $\bar{a}$ ed a sono letterali
- CLAUSE: letterali in OR: $x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \dots \vee x_k$
- CNF: clausole in AND, quindi AND di letterali in OR:

$$CNF = (a \vee \bar{b} \vee c) \wedge (\bar{a} \vee b)$$

- **3CNF**: CNF in cui ogni clausola ha ESATTAMENTE 3 letterali

$$(a \vee \bar{b} \vee c) \wedge (\bar{c} \vee \bar{a} \vee b)$$

- **COROLLARIO**: $\boxed{3SAT = \{\langle F \rangle \mid F \text{ is a satisfiable formula in 3CNF}\} \in NP}$

- **Homework**: $\boxed{SAT \leq_p 3SAT}$

- **TEOREMA**: $\boxed{3SAT \leq_p CLIQUE}$

Idea: trasformare un'istanza di 3-SAT in una di CLIQUE. Sia F una formula di K clausole in 3CNF:

$$F = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_K \vee b_K \vee c_K)$$

Costruiamo un grafo G che ha una clique di ordine K.

- $V(G)$: for each clause of F, exist 3 nodes in G labeled as the corrispondent literals of the clause;
- $E(G)$: for each couple of nodes, exists an edge between them UNLESS:
  (1) the nodes belongs to the same clause;
  (2) the nodes have opposite labels.

NOT ALLOWED!

Dobbiamo ora dimostrare che $F \in 3SAT \iff \exists$ K-clique in G.

$\Rightarrow$) Se F è soddisfacibile, ci sarà un'assegnazione opportuna per le variabili e, per costruzione, esisterà 1 letterale VERO in OGNI CLAUSOLA.

$\rightarrow$ In G, la CLIQUE è formata dai nodi corrispondenti a tali letterali.

Vediamo perché è una clique:

- I letterali appartengono a clausole differenti $\Rightarrow \exists$ un arco tra tutti loro!
- Non possono esserci nodi opposti, perché o è vero un letterale o il suo opposto $\Rightarrow$ Hanno per forza tutti gli archi tra loro!
- Ci sono K clausole, 1 nodo per clausola $\Rightarrow$ È una K-clique!

ok!

$\Leftarrow$) Se esiste una K-clique in G, deve selezionare per forza 1 nodo in ogni clausola, poiché non ci sono archi tra nodi delle stesse triplette.

Poiché non si hanno archi tra nodi con variabili opposte, si può tranquillamente fare la seguente assegnazione:

le variabili che figurano nei nodi formanti la K-clique sono poste ad 1.

$\Rightarrow$ Ogni tripletta e, quindi, ogni clausola, ha almeno 1 variabile, anzi esattamente 1 variabile settata ad 1; ma, gli elementi della clausola sono in OR!

$\Rightarrow$ Ogni clausola è vera $\Rightarrow$ le clausole sono in AND $\Rightarrow$ F è vera!

Quindi:

$$\langle G,K\rangle \in \text{CLIQUE} \iff F \in 3SAT$$

e cioè: $\boxed{3SAT \leq_p \text{CLIQUE}}$ ∎. ok!!!

- **Linguaggio "NP-completo":**

Un linguaggio B si dice "NP-completo" se:

(1) $B \in NP$;

(2) $\forall A \in NP : A \leq_p B$.

- **Teorema:** Se B è NP-completo e $B \in P \Rightarrow P = NP$

Sia $A \in NP \Rightarrow A \leq_p B$, poiché B è NP-completo.

Ma $B \in P \Rightarrow$ se $B \in P$ e $A \leq_p B \Rightarrow A \in P$

Quindi: $NP \subseteq P$; già sappiamo che $P \subseteq NP$.

In conclusione $P = NP$!

• **Teorema:** $\boxed{\text{Se } B \text{ è NP-completo, } C \in NP \text{ e } B \leq_p C \Rightarrow C \text{ è NP-completo.}}$

Sappiamo che "$\leq_p$" è TRANSITIVA.

Poiché B è NP-completo $\Rightarrow \forall A \in NP : A \leq_p B$

$$\Rightarrow A \leq_p B \leq_p C \Rightarrow \left.\begin{array}{l} \forall A \in NP : A \leq_p C \\ C \in NP \end{array}\right\} \Rightarrow C \text{ è NP-completo} \quad \blacksquare$$

ok!!

• **Linguaggio NP-Hard:**

Un linguaggio B è detto "NP-hard" se:

$$\forall A \in NP, \; A \leq_p B$$

• **Teorema:** $\boxed{\text{Se } B \text{ è NP-hard e } B \in P \Rightarrow P \equiv NP}$

• **Teorema:** $\boxed{\text{Se } B \text{ è NP-hard, } B \leq_p C \Rightarrow C \text{ è NP-hard}}$

TEOREMA DI COOK-LEVIN

SAT è NP-completo!

IDEA: sappiamo già che SAT $\in$ NP, dobbiamo dimostrare che:

$$\forall A \in NP, \quad A \leq_p SAT$$

Ma, se $A \in NP \Rightarrow \exists$ NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ tale che ogni ramo di computazione termina in un numero di passi $< n^k$.

Da $\langle M, w \rangle \rightarrow$ costruiamo una FORMULA BOOLEANA ϕ tale che:

$$M(w) \text{ accetta} \iff \phi \in SAT$$, cioè se ϕ è soddisfacibile

PROOF: n^k è il limite massimo di lunghezza di un ramo di computazione, così come la configurazione di M nei vari passi del ramo.
Quindi, rappresentiamo 1 RAMO DI COMPUTAZIONE con 1 TABLEAU:

#	q_0	$w_1 \dots w_m \sqcup \dots \sqcup$	#
#	w_1	$q_1\, w_2 \dots w_m \sqcup \dots \sqcup$	#
		- - -	
#			#

n^k righe × n^k colonne

- Ogni RIGA è 1 CONFIGURAZIONE di computazione
- La 1ª riga è la START CONFIGURATION di M su w.
- cell $[i, j] \in C = \Gamma \cup Q \cup \{\#\}$
  - i: # row: step
  - j: # column: position on tape

* Un TABLEAU è ACCETTANTE se almeno 1 riga contiene lo stato q_A e tutte le righe sono correlate lecitamente: da una riga devo poter passare alla successiva se la transizione è lecita per la NTM M.

Ora, dobbiamo costruire la formula ϕ tale che:

$$\phi \text{ è soddisfacibile} \iff \exists \text{ un tableau legale e accettante per } M(w) \iff w \in A.$$

$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{MOVE} \wedge \phi_{ACCEPT}$$

* le VARIABILI di ϕ sono : $x_{i,j,s}$, dove : $i,j \in [1, n^k]$
$s \in C = \Gamma \cup Q \cup \{\#\}$

$x_{i,j,s} = 1$ → vuol dire cell $[i,j] = s$ nel TABLEAU !

- ϕ_{CELL} := "Ogni cella del tableau contiene ESATTAMENTE 1 simbolo di C"

$$\phi_{CELL} := \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

per ogni cella — deve esserci almeno 1 simbolo — e — non più di 1 simbolo

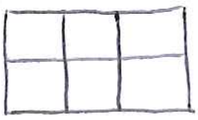
- ϕ_{START} := "La prima riga del tableau è la start configuration di M(w)"

$$\phi_{START} := x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \dots \wedge x_{1,n+2,w_n} \bigwedge_{n+3 \le j \le n^k - 1} x_{i,j,\sqcup} \wedge x_{i,n^k,\#}$$

- ϕ_{ACCEPT} := "Almeno 1 riga deve contenere q_A"

$$\phi_{ACCEPT} := \bigvee_{1 \le i,j \le n^k} x_{i,j,q_A}$$

- ϕ_{MOVE} := ["Ogni riga del tableau è seguita da una riga che corrisponde ad una configurazione che può essere raggiunta in un SINGOLO PASSO da M dalla riga attuale".]

Possiamo considerare delle FINESTRE 3×2  e usarle per verificare la correttezza delle mosse lungo tutto il tableau ; se tutte le finestre sono legali ⇒ il TABLEAU è legale.

• Esempio:

$$\delta(q_1, a) = \{(q_1, b, R)\}$$

$$\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$$

✓

a	q_1	b
q_2	a	c

✓

a	q_1	b
a	a	q_2

✓

a	a	q_1
a	a	b

✓

#	b	a
#	b	a

✓

a	b	a
a	b	q_2

✓

b	b	b
c	b	b

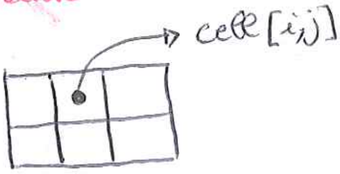
a	b	a
a	a	a

✗ non valida

a	q_1	b
q_2	a	a

✗ non valida

Dunque: se con (i,j)-window indichiamo la finestra



abbiamo:

$$\phi_{MOVE} := \bigwedge_{1 < i,j < n^k} \{(i,j)\text{-window is legal}\}$$

dove:

$$(i,j)\text{-window is legal} := \bigvee_{\text{all legal } \begin{array}{|c|c|c|}\hline a_1 & a_2 & a_3 \\ \hline a_4 & a_5 & a_6 \\ \hline\end{array}} \left\{ \begin{array}{l} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge \\ \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \end{array} \right\}$$

* Ma la costruzione di ϕ è polinomiale?

$$|\phi_{CELL}| \to O\big((n^k)^2 \cdot |C^2| \cdot \underbrace{\log n}_{\text{per codificare i valori di } i \text{ e } j}\big) = O(n^{2k+1})$$

$$|\phi_{START}| \to O(n^k \log n) = O(n^{k+1})$$

$$|\phi_{ACCEPT}| \to O((n^k)^2 \cdot \log n) = O(n^{2k+1})$$

$$|\phi_{MOVE}| \to O((n^k)^2 \cdot \log n) = O(n^{2k+1})$$

$\Rightarrow$ $|\phi| = O(n^{2k+1})$ $\to$ yes, è POLINOMIALE!

$\Rightarrow$ SAT è NP-completo!

■ ok!!!

- **COROLLARIO**: $\boxed{SAT \in P \iff P = NP}$

- **Lemma**: $\boxed{CNF\text{-}SAT \leq_p 3SAT}$

Sia F una formula in CNF $\Rightarrow F = \bigwedge (e_1 \vee e_2 \vee \ldots \vee e_k)$

- se $(e_1 \vee e_2)$ $\Rightarrow$ la riscrivo come $(e_1 \vee e_2 \vee e_2)$
- se $K > 3$: $(e_1 \vee e_2 \vee e_3 \vee e_4 \vee e_5) \rightarrow (e_1 \vee e_2 \vee z_1) \wedge (\overline{z}_1 \vee e_3 \vee z_2) \wedge$
$\wedge (\overline{z}_2 \vee e_4 \vee e_5)$

$\Rightarrow$ $CNF\text{-}SAT \leq_p SAT$ ∎ ok!!!

- **COROLLARIO**: $\boxed{3SAT \text{ è NP-completo}}$

Trasformiamo ϕ in CNF (Forma Normale Congiuntiva)

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

- ϕ_{cell} è già in CNF: 2 grandi AND
  - 1 AND di OR
  - 1 AND di AND di OR
- ϕ_{start} è in CNF: AND di clausole lunghe 1
- ϕ_{accept} è in CNF: tutti OR $\rightarrow$ è una formula di 1 UNICA CLAUSOLA
- ϕ_{move} non è CNF, ma possiamo trasformarla:

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R) \rightarrow \text{in CNF}$$

Ma la formula potrebbe esplodere ESPONENZIALMENTE nella dimensione!

$$(a \wedge b \wedge c) \vee (d \wedge e \wedge f) \equiv$$
$$\equiv [(a \wedge b \wedge c) \vee d] \wedge [(a \wedge b \wedge c) \vee e] \wedge [(a \wedge b \wedge c) \vee f] \equiv$$
$$\equiv [(d \vee a) \wedge (d \vee b) \wedge (d \vee c)] \wedge [(e \vee a) \wedge (e \vee b) \wedge (e \vee c)] \wedge [(a \vee f) \wedge (b \vee f) \wedge (c \vee f)]$$

è in CNF, ma ESPONENZIALMENTE più lungo!

$\rightarrow$ * MA, non è un problema! Poiché:

$$\phi_{move} = \underbrace{\bigwedge}_{O(n^{2k})} \Big[\underset{O(1)}{\bigvee} \; \underset{O(1)}{\bigwedge} \Big] = \underbrace{\bigwedge}_{O(n^{2k})} \Big[\underset{O(1)}{\bigwedge} \; \underset{O(1)}{\bigvee} \Big] \longrightarrow \underline{\underline{O(n^{2k+1})}}$$

finestre 3x2 legali della macchina → COSTANTI

6 elementi per finestra → COSTANTI

$\Rightarrow \phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$ è in CNF

$\Rightarrow$ $\boxed{\text{CNF-SAT è NP-completo}}$!

Poiché CNF-SAT $\leq_p$ 3SAT e 3SAT è NP-completo. ■ ok!!!

• **COROLLARIO:** $\boxed{\text{CLIQUE è NP-completo}}$

Abbiamo visto che 3SAT $\leq_p$ CLIQUE, inoltre, 3SAT è NP-completo e CLIQUE $\in$ NP $\Rightarrow$ [CLIQUE è NP-completo] ■ ok!!!

• Sia $X \in NP$ un problema "naturale"; abbiamo 2 possibilità:

$$\begin{cases} X \in P \longrightarrow \text{algoritmo polinomiale ed efficiente} \\ X \text{ è NP-completo} \end{cases}$$

* In pochissimi casi c'è una 3ª possibilità: non si riesce a dimostrare né che $X \in P$, né che X è NP-completo.

Uno di questi casi è il PROBLEMA DI ISOMORFISMO TRA GRAFI !