Teoriasecondaparteetc

Elementi di Teoria della Computazione (Università degli Studi di Salerno)

PROBLEMI DI DECISIONE

"Ragionevole codifica": un algoritmo che produce una rappresentazione univoca dell'input mediante una stringa. Un problema di decisione è :

-Decidibile: se il linguaggio associato è decidibile;

-Semidecidibile: se il linguaggio associato è turing riconoscibile;

-indecidibile: se il linguaggio associato non è decidibile.

Nota: La macchina di turing verifica, come passo preliminare, che l'input corrisponda a una codifica dell'input del problema. Prosegue la computazione solo nel caso lo sia.

Funzioni : una funzione f: x $\rightarrow$ y è una relazione input/ output.

X è l'insieme dei possibili input (dominio);

y è l'insieme dei possibili output(codominio).

Per ogni x $\in$ X esiste un solo output y= f(x) $\in$ Y.

Funzione iniettiva: f: x $\rightarrow$ y è iniettiva se $\forall$ x, x' $\in$ X con x $\neq$ x' $\rightarrow$ f(x) $\neq$ f(x') .

Funzione suriettiva: f: x $\rightarrow$ y è suriettiva se $\forall$ y $\in$ Y $\rightarrow$ y= f(x) per qualche x $\in$ X.

Funzione biettiva: Una funzione f: x $\rightarrow$ y è biettiva di x su y, se f è sia iniettiva che suriettiva.

Cardinalità: Due insiemi x e y hanno la stessa cardinalità se esiste una funzione biettiva f: x $\rightarrow$ y di x su y.

Numerabile: Un insieme x è numerabile se è finito oppure ha la stessa cardinalità di N

Teorema: $\sum^*$ è numerabile .

Dim. Dimostriamo che $\sum^*$ è numerabile, poiché ha la stessa cardinalità di N . Supponiamo che $\sum=\{0,1\}$, possiamo avere una lista dove inseriamo come primo elemento la stringa vuota $w_0= \varepsilon$. Poi possiamo inserire tutti gli elementi di lunghezza 1, in un ordine lessicografico prestabilito, ad esempio $w_1=0$, $w_2=1$, poi quelli di lunghezza 2 e così via. Per conoscere data una stringa w di lunghezza i, quale sarà la sua posizione nella lista, posso trovarla

con : $\sum_{j=0}^{i-1} 2^j + ordine\ lessicografico$. Quindi le macchine di turing potendo esprimerle come stringhe, sono numerabili.

Nota: Un sottoinsieme di un insieme numerabile è anche esso numerabile.

Teorema: L'insieme dei numeri razionali Q è numerabile.

Dim. Dimostriamo che l'insieme $Q=\{\frac{m}{m} \vee m,n \in N\} e\, N$, hanno la stessa cardinalità.

Creiamo una matrice infinita contenente tutti i numeri razionali positivi. La riga i-esima contiene tutti i numeri con numeratore i e la colonna j-esima ha tutti i numeri con denominatore j. In tal modo il numero $\frac{i}{j}$ occupa la i-esima riga e la j-esima colonna. Ora trasformiamo questa matrice in una lista, possiamo farlo elencando gli elementi sulle diagonali a partire da un angolo. La prima diagonale include il singolo elemento 1/1, la seconda diagonale include gli elementi 2/1 e 1/2, la terza i tre elementi 3/1, 2/1 e 1/3.

1/1	1/2	1/3	1/4	...
2/1	2/2	2/3	2/4	...
3/1	3/2	3/3	3/4	...
4/1	4/2	4/3	4/4	...
...	...	...	...	...

Per trovare un elemento $\frac{i}{j}$ nella lista, possiamo utilizzare il seguente metodo: : $\sum_{n=1}^{(i+j)-1} (n-1)^{\square} + ordine\ lessicografico$

Nota: Tale formula è approssimata, poiché nella lista sono presenti numeri equivalenti, tipo 1/1 e 2/2 che non dovrebbero esserci (in una lista non abbiamo ripetizioni).

Teorema: L'insieme dei numeri reali R non è numerabile.

Dim. Per dimostrare che R non è numerabile, mostriamo che non esiste una biezione tra N e R . Supponiamo che esista una biezione f che associa tutti gli elementi di N con tutti gli elementi di R . Ciò significa che possiamo costruire la lista:

f(1), f(2), f(3) …

$\forall$ i $\geq$ 1, scriviamo f(i)= f_0 (i), f_1(i) f_2(i) f_3(i) …

Possiamo togliere la parte intera f_0(i) e considerare solo la parte decimale e quindi organizzarla in una matrice nel seguente modo:

$i \backslash f(i)$	f_1	f_2	f_3	…	…	…
1	$f_1(1)$	$f_2(1)$	$f_3(1)$	…	$f_i(1)$	…
2	$f_1(2)$	$f_2(2)$	$f_3(2)$	…	$f_i(2)$	…
3	$f_1(3)$	$f_2(3)$	$f_3(3)$	…	$f_i(3)$	…
⋮	⋮	⋮	⋮	⋱	⋮	⋮
i	$f_1(i)$	$f_2(i)$	$f_3(i)$	…	$f_i(i)$	…
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Costruiamo un numero x $\in$ (0,1), dove x=0, x_1 x_2 x_3 … x_i … , ottenuto scegliendo:

$x_i \neq f_i(i)$, $\forall$ i $\geq$ 1, chiaramente x $\in R$. Il numero x è formato da tutti i numeri diversi da quelli della diagonale. Se x=f(j), allora il suo j-mo simbolo è $x_j=f_j(j)$, ma per definizione di x, $x_j \neq f_j(j)$, assurdo.

Teorema: Alcuni linguaggi non sono turing-riconoscibili.

Dim. Posso costruire una matrice in cui le righe corrispondono a macchine di turing, che sono numerabili, essendo esprimibili come stringhe. E sulle colonne delle stringhe $\{w_1, w_2, w_3, \dots\} = \sum^*$, che è numerabile. La matrice sarà binaria, ovvero, avrà 1 se la stringa appartiene alla macchina di turing o 0 se non appartiene. Quindi

$x_{i,j}=1$ se $w_j \in L(M_i)$, $x_{i,j}=0$ altrimenti.

Costruiamo un linguaggio $L=\{ w_i \in \sum^* \mid w_i \notin L(M_i) \}$, cioè L è il complemento della diagonale.

	w_1	w_2	w_3	…	w_i	w_j
M_1	$x_{1,1}$	·	·	·	·	·
M_2	·	$x_{2,2}$	·	·	·	·
·	·	·	$x_{3,3}$	·	·	·
·	·	·	·	$x_{4,4}$	·	·
M_i	·	·	·	·	$x_{i,i}$	$x_{i,j}$
·	·	·	·	·	·	·
·	·	·	·	·	·	·

Supponiamo che $L=L(M_h)$, allora risulta:

- $w_h \in L \rightarrow x_{h,h}=0 \rightarrow w_h \notin L(M_h) = L$, assurdo ;
- $w_h \notin L \rightarrow x_{h,h}=1 \rightarrow w_h \in L(M_h) = L$, assurdo .

Abbiamo così dimostrato che l'insieme di tutti i linguaggi non può essere messo in corrispondenza biunivoca con tutte le macchine di turing. Quindi alcuni linguaggi non sono turing riconoscibili.

Macchina di turing universale: Una macchina di turing universale U simula la computazione di una qualsiasi macchina di turing M. U riceve in input una rappresentazione <M, w> di M e di un possibile input w di M.

<M, w> → Macchina universale U → $\begin{cases} accetta\, se\ M\ accetta\ w \\ rifiuta\, se\ M\, rifiuta\ w \end{cases}$

Teorema: Il linguaggio A_{TM}= { <M,w> | M è una mdt che accetta la stringa w. } è turing riconoscibile

Dim. Definiamo una macchina di turing U che riconosce A_{TM}:

U = sull'input <M,w> dove M è una macchina di turing e w una stringa:

1. Simula M sull'input w;
2. Se M accetta, accetta, se M rifiuta, rifiuta.

Teorema: Il linguaggio A_{TM} non è decidibile.

Dim. Assumiamo che A_{TM} sia decidibile , quindi esiste un decisore H che decide A_{TM}.

Su input <M,w>, dove M è una macchina di turing e w una stringa, H si ferma e accetta se M accetta w, inoltre H si ferma e rifiuta se M non accetta. Schematizzato:

<M, w> → H → $\begin{cases} accetta\, se\ M\ accetta\ w \\ rifiuta\, altrimenti \end{cases}$

Costruiamo una nuova macchina di turing D su input <M>:

1. Simula H su input <M, <M>> dove M è una macchina di turing e <M> una sua codifica.
2. Fornisce come output l'opposto di H, cioè se H accetta, rifiuta, se H accetta, rifiuta.

Schematizzato:

D(<M>) → ¿

Se diamo in input a D, la sua stessa codifica <D>, risulta che :

D(<D>) → ¿

Risulta un assurdo, poiché D accetta <D> se e solo se D rifiuta <D>.

Vediamola dal punto di vista della diagonalizzazione, consideriamo la tavola:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...
M_1	acc		acc		...
M_2	acc	acc	acc	acc	...
M_3					
M_2	acc	acc			...
⋮	⋮	⋮	⋮	⋮	⋮

Considerando che esiste un decisore H, non abbiamo caselle vuote, poiché rifiuta se la macchina M_i va in loop, quindi avremo:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...
M_1	acc	rej	acc	rej	...
M_2	acc	acc	acc	acc	...
M_3	rej	rej	rej	rej	...
M_2	acc	acc	rej	rej	...
⋮	⋮	⋮	⋮	⋮	⋮

Consideriamo ora la macchina D come costruita in precedenza e diamole in input <D>

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...	$\langle D \rangle$	...
M_1	acc	rej	acc	rej	...	...	...
M_2	acc	acc	acc	acc	...	...	...
M_3	rej	rej	rej	rej	...	...	...
M_2	acc	acc	rej	rej	...	...	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
D	rej	rej	acc	acc	...	???	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

L'autoreferenzialità è un problema da tener conto.

Co-turing riconoscibile: Un linguaggio L è co- turing riconoscibile se $\overline{L}$ è turing riconoscibile.

L decidibile: Un linguaggio L è decidibile se è sia turing riconoscibile che co-turing riconoscibile. Quindi , per qualsiasi

linguaggio non decidibile, o esso non è turing riconoscibile o il suo complemento non è turing riconoscibile.

Teorema: L è decidibile ↔ L e il suo complemento sono entrambi turing riconoscibili

Dim.

Dobbiamo dimostrare che se L è decidibile, sia L che il suo complemento sono turing riconoscibili. Se L è decidibile allora esiste una macchina di turing M con due possibili risultati di una computazione e tale che M accetta w se e solo se w ∈ L. Allora L è turing riconoscibile. Inoltre è facile costruire una macchina di turing $\acute{M}$ che accetta w se e solo se w ∉ L.

$\acute{M}$(w) → $\begin{cases} accetta\ se\ M\ rifiuta\ w \\ rifiuta\ se\ M\ accetta\ w \end{cases}$

Ora bisogna dimostrare che se L e il suo complemento sono entrambi turing riconoscibili, allora L è decidibile. Supponiamo quindi che L e il suo complemento siano entrambi turing riconoscibili. Sia

- M_1 una macchina di turing che riconosce L;
- M_2 una macchina di turing che riconosce $\overline{L}$.

Definiamo una macchina di Turing N a due nastri su input x:

1. Copia x sui nastri di M_1 e M_2 ;
2. Simula M_1 e M_2 in parallelo (usa un nastro per M_1, l'altro per M_2);
3. Se M_1 accetta, accetta, se M_2 accetta, rifiuta.

Quindi:

N(w) → $\begin{cases} accetta\ se\ M_1 accetta\ x \\ rifiuta\ se\ M_2 accetta\ x \end{cases}$

N decide L, infatti per goni stringa x, risulta:

- x∈ L. Ma x ∈ L se M_1 si arresta e accetta x;
- x∉ L. Ma x ∉ L se M_2 si arresta e accetta x.

Teorema: A'_{TM} non è turing riconoscibile

Dim.

Sappiamo che A_{TM} è turing riconoscibile . Se A'_{TM} lo fosse, allora A_{TM} sarebbe decidibile. Ma abbiamo dimostrato che non lo è , quindi A'_{TM} non è turing riconoscibile.

RIDUCIBILITÀ

Nota: Schema per dimostrare che un problema B è indecidibile, tramite una riduzione ad un problema noto A indecidibile:

1. Sappiamo che A risulta indecidibile, vogliamo provare che B è indecidibile;
2. Assumiamo per assurdo che B è decidibile ed usiamo questa assunzione per provare che A è decidibile;
3. La contraddizione ci fa assumere che B sia indecidibile.

Teorema: Il problema $HALT_{TM}$ ={ <M,w> | M è una Macchina di turing e M si ferma su input w } è indecidibile.

Dim. Assumiamo al fine di ottenere una contraddizione che esista una macchina di turing R che decide $HALT_{TM}$. Costruiamo la macchina di turing S che decide A_{TM} che opera come segue:

S= " Su input <M,w>, una codifica di una macchina di turing M ed una stringa w:

1. Esegue la macchina di turing R su input <M,w>, se R rifiuta, rifiuta, altrimenti va al passo 2;
2. Simula M su w, finché non si ferma (si deve fermare altrimenti sarebbe stato rifiutato da R);
3. Se M accetta, accetta, se M rifiuta, rifiuta.

Se esistesse R che decide $HALT_{TM}$, allora anche S decide A_{TM} , ma sappiamo che A_{TM} è indecidibile, allora R non può esistere e $HALT_{TM}$ è indecidibile.

Funzione calcolabile: Una funzione f: $\sum^* \rightarrow \sum^*$ è calcolabile se esiste una macchina di turing M, che su qualsiasi input w, si ferma avendo solo f(w) sul nastro.

Linguaggio riducibile: Un linguaggio A è riducibile a B ($A \leq_m B$) se esiste una funzione calcolabile f: $\sum^* \rightarrow \sum^*$ tale che $\forall$ w, risulta: $w \in A \leftrightarrow f(w) \in B$, la funzione f è chiamata riduzione da A a B.

Teorema: Se $A \leq_m B$ e B è turing decidibile , allora A è turing decidibile.

Dim. Siano M il decider per B ed f la riduzione da A a B.

Descriviamo un decider N per A, come segue:

N=”su input w”:

1. Computa f(w);
2. Esegue M su input f(w) e restituisce lo stesso output di M.

Se $w \in A$, allora $f(w) \in B$ perché f è una riduzione da A a B, per definizione $w \in A \leftrightarrow f(w) \in B$, quindi M accetta f(w) ogni volta che $w \in A$. Di conseguenza N decide A.

Teorema: Se $A \leq_m B$ e B è turing riconoscibile , allora A è turing riconoscibile.

Dim. Siamo M il riconoscitore per B ed f la riduzione da A a B .

Descriviamo un riconoscitore N per A come segue:

N=”Su input w “:

1. Computa f(w);
2. Esegue M su input f(w) e restituisce lo stesso output di M.

Se $w \in A$, allora $f(w) \in B$ perché f è una riduzione da A a B . Quindi, M accetta f(w) ogni volta che $w \in A$. Quindi N è un riconoscitore di A.

Corollario : Se $A \leq_m B$ e A è indecidibile, allora B è indecidibile. (Se B fosse decidibile, lo sarebbe anche A per il teorema precedente).

Corollario : Se $A \leq_m B$ e A non è turing riconoscibile, allora B non è turing riconoscibile. (Se B fosse turing riconoscibile, lo sarebbe anche A per il teorema precedente).

Teorema: E_{TM}={<M>| M è una macchina di turing e L(M)= $\varnothing$} è indecidibile.

Dim.

Posso mostrare che esiste una riduzione da A_{TM} a E_{TM} . Per farlo posso considerare il complemento di E_{TM}. Che definisco come :

E'_{TM} ={ <M> | M è una macchina di turing e L(M) $\neq \varnothing$ }

Vediamo una riduzione da A_{TM} a E'_{TM} , quindi costruiamo una funzione f: $\sum^* \rightarrow \sum^*$, tale che f(<M,w>)=<M_1>, dove M_1 su input x:

1. Se x $\neq$ w, allora M_1 si ferma e rifiuta;
2. Se x=w, allora M_1 simula M su w e accetta x se M accetta w.

f è una riduzione da A_{TM} a E'_{TM} , perché :

- La funzione f è calcolabile, perché modifichiamo solo la codifica dell'input;
- Se <M,w> $\in A_{TM}$, M accetta w , quindi se x=w, M_1 simula M, quindi M_1 accetta il linguaggio. Quindi il linguaggio non è vuoto, di conseguenza $M_1 \in E'_{TM}$;
- Se <M,w> $\notin A_{TM}$,M rifiuta w, quindi $\forall$ x $\neq$ w M_1 rifiuta, per x=w M_1 simula M su x, che rifiuta. Quindi non ci sono stringhe accettate e di conseguenza $M_1 \notin E'_{TM}$.

$A_{TM} \leq_m E'_{TM}$, A_{TM} indecidibile → E'_{TM} è indecidibile.

Nota: La decidibilità non risente della complementazione.

Teorema: $E_{TM}=\{<M>|$ M è una macchina di turing e $L(M)=\varnothing\} \leq_m E_{QTM}=\{$

$<M_1,M_2>$ M_1,M_2 sono macchine di turing e $L(M_1)=L(M_2)$ $\}$

Dim.

Sia M_1 una macchina di turing tale che $L(M_1)=\varnothing$. Creiamo una funzione di riduzione da E_{TM} a E_{QTM}, definita come $f(<M>) \rightarrow <M, M_1>$. Tale funzione è una riduzione, perché:

- La funzione è calcolabile, perché deve prendere la macchina di partenza ed una macchina predefinita ;
- $<M> \in E_{TM} \rightarrow L(M)=\varnothing \rightarrow L(M)=L(M_1)=\varnothing \rightarrow <M, M_1> \in E_{QTM}$;
- $<M> \notin E_{TM} \rightarrow L(M) \neq \varnothing \rightarrow L(M) \neq L(M_1)=\varnothing \rightarrow <M, M_1> \notin E_{QTM}$.

Quindi f è una riduzione da E_{TM} a E_{QTM}.

Teorema: $A_{TM} \leq_m E_{QTM}=\{<M_1,M_2>$ M_1,M_2 sono macchine di turing e $L(M_1)=L(M_2)$ $\}$

Dim.

Creiamo una funzione f di riduzione da A_{TM} a E_{QTM}, definiamola come

f: $<M,w> \rightarrow <M_1,M_w>$, dove per ogni input x:

- M_1 accetta x;
- Mw simula M su w, se accetta, accetta, se rifiuta, rifiuta.

È una riduzione da A_{TM} a E_{QTM}, perché :

- La funzione è calcolabile, perché abbiamo una macchina predefinita che accetta tutti gli input e la descrizione di M e il suo input.
- $<M,w> \in A_{TM} \rightarrow w \in L(M) \rightarrow \begin{cases} L(M_1)=\sum ¿ \\ L(Mw)=\sum ¿ \end{cases} \rightarrow f(<M,w>) \in E_{QTM}$;

- $<M,w> \notin A_{TM} \rightarrow w \notin L(M) \rightarrow \begin{cases} L(M_1) = \sum ¿ \\ L(Mw) = \emptyset \end{cases} \rightarrow f(<M,w>) \notin E_{QTM}$;

Quindi f è una riduzione da A_{TM} a E_{QTM} .

Teorema di Rice: Il teorema di Rice permette di dimostrare la non decidibilità di una serie di problemi. Sia:

P={<M> | M è una macchina di turing che verifica la proprietà P}

un linguaggio che soddisfa le seguenti due condizioni:

- L'appartenenza di M a P dipende solo dal linguaggio L(M), cioè:
  $\forall$ M_1, M_2 macchine di turing tali che $L(M_1) = L(M_2)$, risulta che $<M_1> \in P \leftrightarrow <M_2> \in P$;
- P è un problema non banale, cioè:
  $\exists$ M_1 , M_2 macchine di turing tali che $<M_1> \in P$ e $<M_2> \notin P$.

Allora P è indecidibile.

Teorema: P= {<M> | M è una macchina di turing che verifica la proprietà P} è indecidibile

Dim.

Supponiamo che $T_\emptyset$ sia una macchina di turing che rifiuta qualsiasi input, per cui $L(T_\emptyset)=\emptyset$ e che $T_\emptyset \notin P$, possiamo supporlo perché nel caso appartenga a P, possiamo considerare il complemento di P . Prendiamo una macchina T che appartenga a P, quindi $<T> \in P$, esiste perché P è non banale. Mostriamo che $A_{TM} \leq_m P$, prendiamo una funzione di riduzione f, tale che f: $<M,w> \rightarrow <Mw>$, dove :

Mw = " su input x":

1. Simula M su w. Se si ferma e rifiuta , rifiuta se accetta, va al passo 2;

2. Simula T su x. Se accetta, accetta.

Schematizzato :

Mw: x → w → M $\begin{cases} accetta\ se\ T\ con\ input\ x, accetta \\ rifiuta\ se\ M\ non\ accetta\ w\ o\ T\ non\ accetta\ x \end{cases}$

Abbiamo una riduzione, perché:

- La funzione è calcolabile, perché ho la macchina M che simula w e una macchina predefinita T che simula x.
- Se <M,w> $\in A_{TM}$ → M accetta w → L(Mw)= L(T), ma <T> ∈ P → <Mw> ∈ P;
- Se <M,w> $\notin A_{TM}$ → $\forall$ x , x $\notin$ L(Mw)→ L(Mw)= ∅= L($T_{\varnothing}$), ma <$T_{\varnothing}$> $\notin$ P → <Mw> $\notin$ P.

COMPLESSITÀ

Asintoticità: Siano f e g due funzioni, tali che :

f: $N \rightarrow R^{+¿¿}$, g: $N \rightarrow R^{+¿¿}$

f= O(g(n)) se esistono una costante c > 0 e una constante $n_0 \geq 0$, tali che $\forall$ n ≥ n_0 :

$$f(n) \leq c\ \ g(n)$$

Diremo che g(n) è un limite asintotico superiore per f(n).

Complessità temporale macchina di turing: Sia M=(Q, ∑, Γ, δ, q_0 , q_{accept}, q_{reject}) una macchina di turing deterministica a nastro singolo che si arresta su ogni input. La complessità temporale di M è la funzione f: $N \rightarrow N$, dove f(n) è il massimo numero di passi eseguiti da M su input di lunghezza n, per ogni n ∈ N. Cioè f(n) = massimo numero di passi in $q_o w \rightarrow {}^{*} uqv$, dove q ∈ { q_{accept}, q_{reject}}, al variare di w in $\sum^n$..

Nota : Se M ha complessità temporale f(n), diremo che M decide L(M) in tempo (deterministico) f(n).

Nota : Considerare un numero intero n in base 2 richiede $\log_2 n + 1$ cifre binarie, mentre codificarlo in base unaria, richiede n cifre unarie. Avere una complessità temporale O(n) rispetto alla codifica unaria, può voler dire che la complessità temporale rispetto alla codifica binaria è $O(2^n)$. Non ho problemi con i grafi, perché le rappresentazioni sono correlate polinomialmente, neanche per codifiche in base $k \geq 2$

Teorema: Sia t(n) una funzione tale che $t(n) \geq n$. Ogni macchina di turing multinastro M di tempo t(n) ammette una macchina di turing equivalente a nastro singolo M' con complessità temporale $O(t^2(n))$.

Dim.

1. Ogni mossa di M viene simulata da M' scorrendo il nastro che contiene i k nastri di M ;
2. Il numero di elementi su ogni nastro di M è al più t(n) (una macchina non può toccare un numero di caselle maggiore al numero di passi che compie, e poi per ipotesi $t(n) \geq n$);

Ogni mossa di M per essere simulata da M' impiega O(t(n)). In seguito, M' simula ciascuno dei t(n) passi di M, quindi risulta: O(t(n)) x $t(n) = O(t^2(n))$.

Teorema: Sia t(n) una funzione tale che $t(n) \geq n$. Ogni macchina di turing non deterministica a singolo nastro M di tempo t(n) ammette una macchina di turing equivalente a nastro singolo deterministica M' con complessità temporale $2^{O(t(n))}$.

Dim.

Una macchina di turing a tre nastri che simula M:

1. Copia sul primo nastro la stringa input w;

2. Genera sul nastro 3 (in ordine lessicografico) le stringhe corrispondenti alle possibili computazioni di M su w;
3. Per ogni possibile computazione di M su w, simula sul secondo nastro la computazione.

Ogni nodo dell'albero può avere al più b figli, così il numero totale di foglie per input n è al massimo $b^{t(n)}$. Il tempo per partire dalla radice ed arrivare ad un nodo è O(t(n)). Inoltre dobbiamo elevare al quadrato il tempo di esecuzione, per la conversione da macchina a tre nastri a singolo nastro. Quindi risulta :

$O(t(n) \times b^{t(n)})^2 = (2^{O(t(n))})^2 = 2^{O(t(n))}$.

Classe P: La classe P è l'insieme dei linguaggi L per i quali esiste una macchina di turing M con un solo nastro che decide L e per cui $t_M(n)=O(n^k)$ per qualche k>1.

Tesi di cook (o strong church-turing): La tesi di cook afferma che tutto quello che può essere computato da una macchina di turing in tempo polinomiale, può essere computato da un algoritmo in tempo polinomiale.

RIDUZIONE POLINOMIALE

Riduzione polinomiale: Un problema x si riduce in tempo polinomiale al problema y ($x \leq_p y$) , se arbitrarie istanze di x, possono essere risolte usando:

- Un numero polinomiale di passi di computazione;
- Un numero polinomiale di chiamate ad un oracolo che risolve y.

Le istanze di y date in input all'oracolo devono essere polinomiali nella dimensione rispetto a istanze x.

Nota : Se $x \leq_p y$ e y ammette soluzione in tempo polinomiale, allora anche x ammette soluzione in tempo polinomiale.

Nota : Se $x \leq_p y$ e x non ammette soluzione in tempo polinomiale, allora y non ammette soluzione in tempo polinomiale.

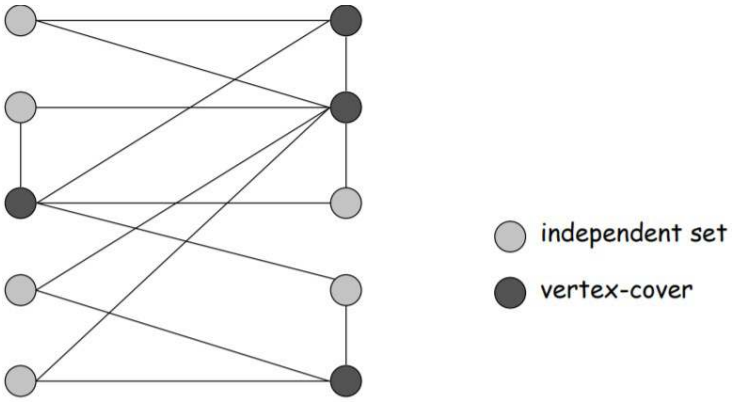
Nota : Se $x \leq_p y$ e $y \leq_p x$, allora $x \equiv_p y$.

RIDUZIONE MEDIANTE EQUIVALENZA SEMPLICE

Independent set: Dato un grafo G=(V,E) e un intero k, esiste un sottoinsieme di vertici

$S \subseteq V$, tale che $|S| \geq k$, e per ogni arco del grafo, al più uno dei suoi estremi è in S ?

Vertex cover: Dato un grafo G=(V,E) e un itero k, esiste un sottoinsieme di vertici $S \subseteq V$, tale che $|S| \leq k$, e per ogni arco del grafo, almeno uno dei suoi estremi è in S?



Teorema:

vertex cover $\equiv_p$ independent set

Dim.

Mostriamo che S è un insieme indipendente se e solo se V-S è un vertex cover.

(→) Sia S un qualsiasi insieme di vertici independent set. Consideriamo un arco arbitrario (u,v). Dato che S è un un independent set → u ∉ S o v ∉ S → u ∈ V-S o v∈ V-S .

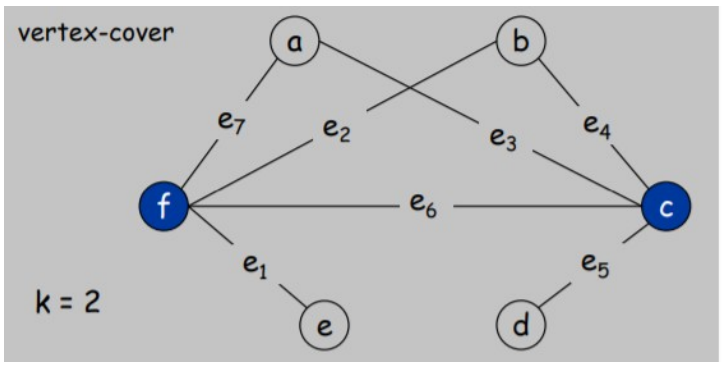
Quindi V-S copre qualsiasi arco del grafo (u,v).

(←) Sia V-S un qualsiasi vertex-cover. Consideriamo due nodi u ∈ S e v ∈ S. Notiamo che (u,v) ∉ E, perché V-S è un vertex cover e di conseguenza l'arco (u,v) sarebbe dovuto essere in V-S, dato che per definizione almeno uno dei vertici di tutti gli archi del grafo è in V-S. Quindi non esistono due nodi in S connessi da un arco → S independent set.

Abbiamo dimostrato che S è un independent set se e solo se il suo complemento V-S è un vertex cover. Se ci sono K nodi in S (independent set), ci saranno n-k nodi in V-S (vertex cover), e viceversa.

RIDUZIONE DA CASO SPECIALE A CASO GENERALE

Set cover: Dato un insieme U di elementi, una collezione di S_1, S_2, … , S_n di sottoinsiemi di U, e un intero k, esiste una collezione di sottoinsiemi ≤ k la cui unione è uguale a U?



SET-COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_a = {3, 7}
S_b = {2, 4}
S_c = {3, 4, 5, 6}
S_d = {5}
S_e = {1}
S_f = {1, 2, 6, 7}

Teorema:

vertex cover $\leq_p$ set-cover

Dim.

Data un'istanza di vertex- cover G=(V,E), un intero k, costruiamo un'istanza di set-cover la cui dimensione è uguale alla dimensione dell'istanza di vertex- cover. Tale istanza è costruita nel modo seguente:

$$k=k,\ U=E,\ S_v=\{e \in E : e \text{ incidente su } v\}.$$

Se nel grafo esiste un set-cover di dimensione ≤ k, esistono k vertici tali che ogni arco ha uno dei vertici in V → Tale insieme di sottoinsiemi , è proprio un insieme vertex cover di dimensione ≤ k. Quindi ho un set cover di dimensione k se e solo se ho un vertex cover di dimensione ≤ k. Inoltre la riduzione è polinomiale, perché ci sono al massimo |V| sottoinsiemi formati da al massimo |E| archi ciascuno. Posso concludere che è una riduzione polinomiale da vertex-cover a set-cover.

RIDUZIONE VIA "GADGETS"

Letterale: una variabile booleana o la sua negazione, x_i o $\acute{x}_i$.

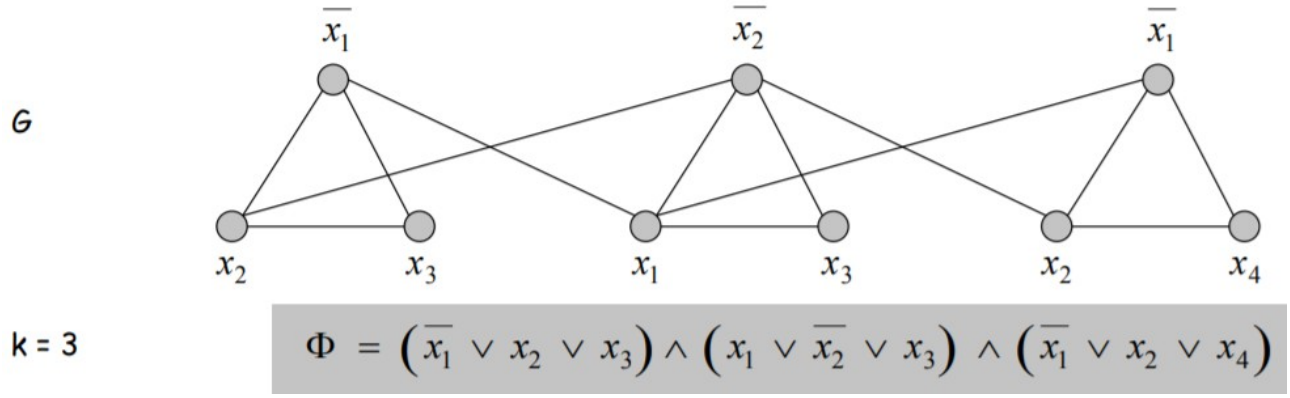
Clausola: un or (disgiunzione) di letterale, $C_j=x_1 \vee x_2 \vee x_3$.

Formula normale congiuntiva (CNF): una formula proposizionale ϕ che è un and (congiunzione) di clausole, $\phi= C_1 \wedge C_2 \wedge C_3$.

3-SAT: Data formula CNF ϕ, dove ogni clausola contiene esattamente 3 letterali, esiste un'assegnazione di verità che la soddisfa?

Nota : Viene chiamata riduzione via "gadgets" perché cerchiamo una struttura in un linguaggio che possa simulare variabili e

clausole in 3-sat.



Teorema:

3-SAT $\leq_p$ Independent-set

Dim.

Data istanza ϕ di 3-SAT, costruiamo un'istanza (G,k) di independent set che ha un insieme indipendente di nodi di dimensione k se e solo se ϕ è soddisfacibile. Costruiamo G come segue:

- G contiene 3 vertici per ogni clausola, uno per ogni variabile;
- Connettiamo i 3 letterali presenti in una clausola fra di loro, al fine di formare un triangolo con gli archi.
- Connettiamo ogni letterale al suo negato.

G contiene un insieme indipendente di dimensione k=|ϕ| (cioè pari al numero di clausole), se e solo se ϕ è soddisfacibile.
Dimostriamolo:

(→) Sia S l'insieme indipendente di dimensione K.

- S deve contenere esattamente un vertice in ogni triangolo(una variabile positiva in ogni clausola);

- Non può contenere un letterale ed un suo negato (non posso avere due variabili complementari con la stessa assegnazione di verità).

Poniamo le variabili rappresentate dai nodi in S a true ed avrò un'assegnazione di verità consistente per tutte le clausole, quindi ϕ è soddisfatta.

(←) Data un'assegnazione di verità, selezioniamo un letterale true da ogni triangolo . Per come è costruito G, sarà un insieme indipendente di dimensione k, perché i vertici sono indipendenti, dato che prendo un elemento in ogni triangolo non correlato con elementi di altri triangoli.

La riduzione è polinomiale, perché se ho una formula di dimensione k, il grafo ha (3*k) vertici e quindi è lineare rispetto l'input 3-sat . Posso concludere che ho una riduzione polinomiale da 3-sat a independent-set.

Teorema:

Se $x \leq_p y$ e $y \leq_p z$, allora $x \leq_p z$ (transitività)

Dim.

Se esiste $x \leq_p y$, sappiamo che se partiamo da un'istanza x di X, in tempo polinomiale riusciamo a trasformare x in un'istanza y di Y, tale che dalla di risposta di y, so quella di x. Se $y \leq_p z$, sappiamo che l'istanza y di Y può essere trasformata in tempo polinomiale in un'istanza z di Z, tale che dalla risposta di z, so quella di y. In un tempo polinomiale riesco, quindi, a trasformare x in un'istanza di y e poi questa stessa istanza in un'istanza di z, in modo tale che la risposta di z è valida per y e di conseguenza anche per x.

Self-reducibility: problema di ricerca $\leq_p$ problema versione di decisione.

Nota : Un esempio può essere:

trovare vertex-cover di cardinalità minima $\leq_p$ versione di decisione di vertex-cover

Consideriamo un grafo G=(V,E)

1. Trovo k*, ovvero il valore ottimale di k che minimizzi il numero di vertici vertex-cover. Lo posso trovare semplicemente in maniera iterativa partendo da k=V e decrementando k finché vertex cover non restituisce false. Il valore prima della restituzione di false è il nostro k ottimale;
2. While(G ha più di k* vertici){
   Scegli un v qualsiasi in G, calcola x= vertex cover per G- {v};
   Se x=k* →elimino V da G (perché non sarà il vertice che mi permette di ottenere un vertex cover ottimale);
   }.

   PROBLEMI NP COMPLETI

Classe P: La classe P contiene problemi di decisione per cui esiste un algoritmo polinomiale.

Certificatore: Un algoritmo C(s,t) è un certificatore per problema X se per ogni stringa s , risulta che : $s \in X$ (istanza sì del nostro problema) se e solo se esiste una stringa t, detto certificatore, tale che C(s,t)=yes.

Nota: Se $s \notin X$ non posso trovare un qualche t per cui l'algoritmo C(s,t)=yes.

Nota: Il certificatore C(s,t) per essere di tempo polinomiale deve:

- essere un algoritmo polinomiale;
- $|t| \leq p(|s|)$, per qualche polinomio p(.) .

Classe NP: Per una determinata classe NP di problemi, non è noto se esiste o non esiste un algoritmo polinomiale di soluzione. Tale classe rappresenta i problemi di decisione che ammettono un certificatore avente tempo polinomiale.

Classe EXP: Classe che rappresenta i problemi di decisione avente algoritmi di tempo esponenziale.

Teorema:

$P \subseteq NP$

Dim.

Consideriamo un qualsiasi problema x in P. Per definizione un algoritmo polinomiale A(s) che risolve x. Prendiamo un certificato t=ε, il certificatore sarà C(s,t)=A(s).

Teorema:

$NP \subseteq EXP$

Dim.

Consideriamo un qualsiasi problema x in NP. Per definizione, esiste un certificatore polinomiale C(s,t) per x. Per risolvere su input s, usiamo C(s,t) su tutte le stringhe t con $|t| \leq p(|s|)$. Restituisce yes, se C(s,t) restituisce yes per una qualsiasi stringa. Schematizzando, abbiamo:

$$\text{Algo}(s)=\begin{cases} yes\ se\ \exists t \in C(s,t)=yes, \\ no\ se\ \forall t \in C(s,t)=no \end{cases}$$

Possiamo ridurre i passi se sappiamo che $|t| \leq T$, dove T è un certo valore (non abbiamo bisogno di provare anche per t che sappiamo

essere non verificabili, quindi scegliamo un certo numero di t). Abbiamo:

$$\text{Algo(s)}=\begin{cases} yes\ se\ \exists t \in C(s,t)=yes, tale\ che \vee t \vee \leq T \\ no\ se\ \forall t \in C(s,t)=no, tale\ che \vee t \vee \leq T \end{cases}$$

Riduzione in tempo polinomiale (Cook): Il problema x si riduce in modo polinomiale al problema y, se istanze arbitrarie del problema x , possono essere risolte usando:

- Un numero polinomiale di passi di computazione;
- Un numero polinomiale di chiamate ad un oracolo che risolve il problema y.

Trasformazione in tempo polinomiale (Karp): Il problema x si trasforma in modo polinomiale al problema y se dato un qualsiasi input x di X, possiamo costruire un input y di Y tale che x è istanza yes di X se e solo se y è istanza yes di Y, con |y| di dimensione polinomiale in |x| .

NP Completo: Problema y in NP con la proprietà che per ogni problema x in NP, $x \leq_p y$.

Teorema:

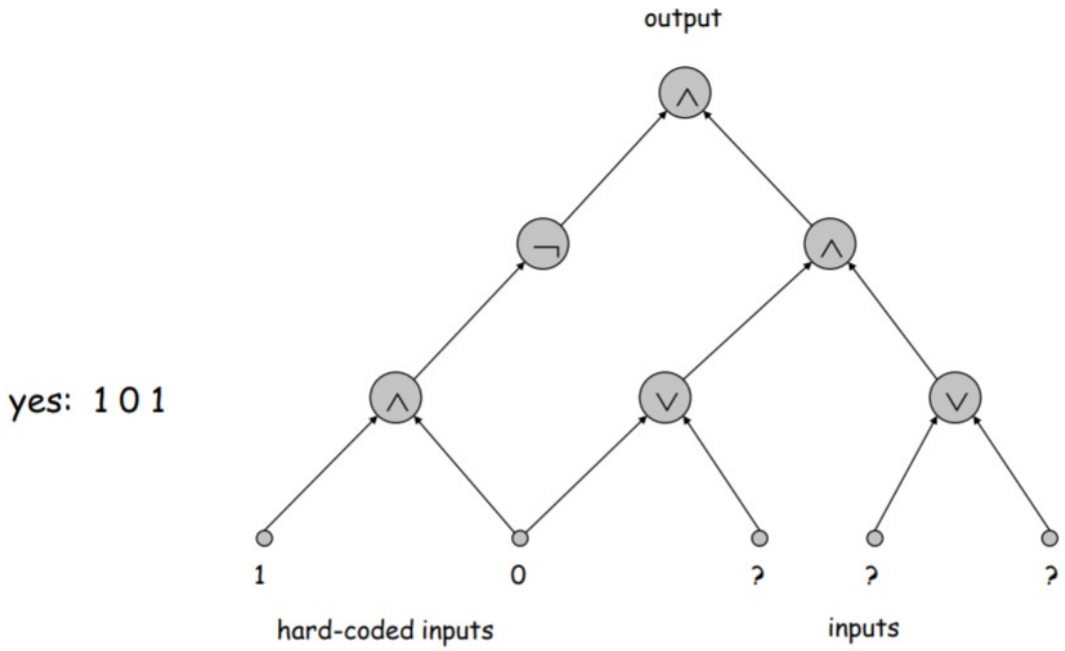
Assumiamo che y è un problema NP-Completo. Allora y è risolvibile in tempo polinomiale se e solo se P=NP

Dim.

(→) Assumiamo che y può essere risolto in tempo polinomiale. Sia x un qualsiasi problema in NP . Poiché $x \leq_p y$, possiamo risolvere $x \in$ NP in tempo polinomiale→ NP$\subseteq$ P, sappiamo per il teorema precedente che P$\subseteq$ NP → P=NP.

(←) Se P=NP allora y può essere risolto in tempo polinomiale, perché y è in NP.

Circuit-sat: Dato un circuito combinatoriale composto da porte AND,OR e NOT e degli input prefissati, c'è un modo di settare i restanti input del circuito in modo che l'output sia 1?



Teorema:

Circuit-sat è NP-Completo

Dim(idea).

Un qualsiasi algoritmo che prende in input un numero fissato n di bits e produce risposta yes/no può essere rappresentato con un circuito. Fissare il numero di bit è importante perché sancisce la differenza base tra algoritmo e circuito. Se l'algoritmo è di tempo polinomiale, allora il circuito è di dimensione polinomiale. Consideriamo un problema x in NP. Ha certificato polinomiale C(s,t). Per determinare se s è in x, dobbiamo sapere se esiste un certificato t di lunghezza p(|s|), tale che C(s,t)=yes. Consideriamo

C(s,t) come un algoritmo su |s|+p(|s|) bits (input s, certificato t) e convertiamolo in circuito di dimensione k, dove:

- I primi |s| bits sono hard-coded (con valori s prefissati);
- Restanti p(|s|) bits rappresentano i bits di t .

Il circuito k è soddisfacibile se e solo se C(s,t)=yes. È chiaro che è in NP, perché il certificato di circuit-sat è la soluzione stessa.

Nota: Passi per stabilire l'NP-Completezza di problemi y:

- Mostrare che y è in NP;
- Scegliere un problema NP completo x;
- Provare che $x \leq_p y$.

Teorema:
Se x è un problema NP-completo e y è un problema in NP tale che $x \leq_p y$, allora y è NP-completo.

Dim.

Supponiamo che x è un problema NP-completo e che y è un problema in NP, tale che $x \leq_p y$. Sia w un qualsiasi problema in NP. Allora $w \leq_p x \leq_p y$. Per la transitività $w \leq_p y$. Quindi y è NP-completo.

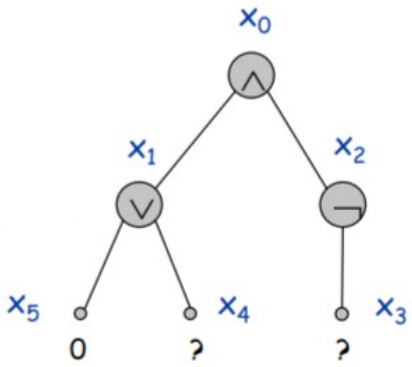
Teorema:
3-SAT è NP-Completo

Dim.

- Mostriamo che 3-SAT è in NP. Il certificato t può essere un'assegnazione di valori di verità alle n variabili booleane. Il

certificatore, controlla se l'istanza sì del problema 3-SAT, ovvero la formula ϕ, ha almeno un letterale true per ogni clausola, in base all'assegnazione del certificato t.

- Dimostriamo che CIRCUIT-SAT $\leq_p$ 3-SAT. Sia k un qualsiasi circuito. Creiamo una variabile 3-SAT, x_i per ogni elemento i del circuito, aggiungiamo le seguenti clausole, nelle seguente casistiche(prendiamo come riferimento la figura a lato):



  - $x_2 = \neg x_3$ → aggiungiamo 2 clausole : $(x_2 \vee x_3) \wedge (\acute{x}_2 \vee \acute{x}_3)$, ci serve per garantire che $x_2 \neq x_3$;
  - $x_1 = x_5 \vee x_4$ → aggiungiamo 3 clausole: ($\acute{x}_1 \vee x_4 \vee x_5 ¿ \wedge (x_1 \vee \acute{x}_4) \wedge (x_1 \vee \acute{x}_5)$, ci serve per garantire che x_1 sia 0 nel caso x_4 e x_5 siano uguali a 0, e per garantire che x_1 sia 1 nel caso x_4 o x_5 siano 1;
  - $x_0 = x_1 \vee x_2$ → aggiungiamo 3 clausole: ($x_0 \vee \acute{x}_1 \vee \acute{x}_2 ¿ \wedge (\acute{x}_0 \vee x_1) \wedge (\acute{x}_0 \vee x_2)$, ci serve per garantire che x_0 sia 0 nel caso x_1 o x_2 siano uguali a 0, e per garantire che x_0 sia 1 nel caso x_1 e x_2 siano 1.

Aggiungiamo gli hard-coded input e l'output desiderato:

  - $x_5 = 0$→ aggiungiamo 1 clausola $\acute{x}_5$;
  - $x_0 = 1$→aggiungiamo 1 clausola x_1.

Passo finale: trasformare clausole di lunghezza ≤ 3 in clausole di lunghezza 3.

CO-NP: CO-NP è la classe di problemi che rappresentano il complemento di un problema di decisione NP.

Teorema:

Se NP ≠ CO-NP, allora P≠NP

Dim.

Se P=NP, allora NP=CO-NP, perché P è chiusa per il complemento, se P=NP, allora NP è chiusa per il complemento, quindi NP=CO-NP.

NP ∩ CO-NP : Se un problema x è sia in NP che in CO-NP (NP ∩ CO-NP), allora:

- Per istanze yes, esiste un certificatore polinomiale;
- Per istanze no, esiste un "disqualifier" polinomiale.

Nota: P⊆ (NP ∩ CO-NP)

PROBLEMI DI SEQUENCING

Ham-cycle: Dato un grafo non orientato G=(V,E), esiste un ciclo semplice (che non tocca nodi già toccati in precedenza) Γ che contiene ogni nodo in V?

Dir-ham-cycle: Dato un grafo orientato G=(V,E), esiste un ciclo semplice (che non tocca nodi già toccati in precedenza) Γ che contiene ogni nodo in V?

Teorema:

Dir-ham-cycle $\leq_p$ ham-cycle

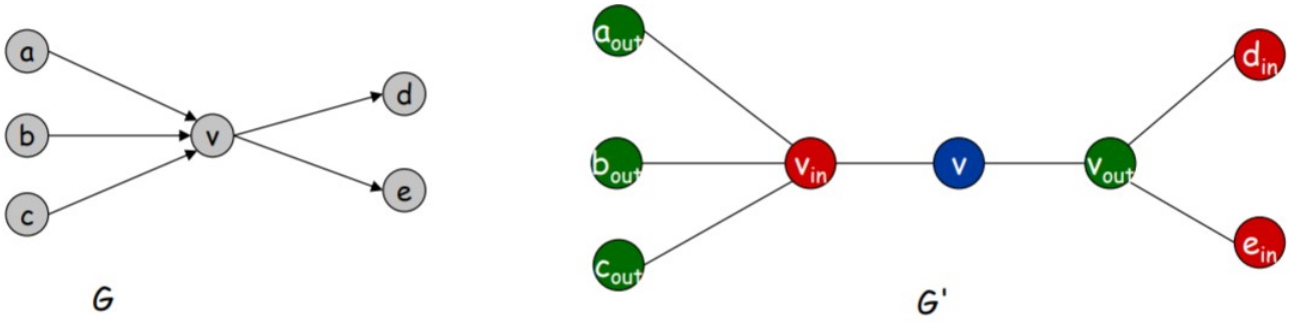
Dim.

Dato un grafo orientato G=(V,E), costruiamo un grafo non orientato G' con 3n nodi.

Per ogni vertice V creiamo 3 nodi fra loro collegati, v_{in}, v e v_{out} . Creiamo una corrispondenza fra G e G', nel seguente modo:

- Per ogni arco entrante in v in G, avremo in G' un arco entrante in v_{in};

- Per ogni arco uscente da v in G, avremo in G’ un arco uscente da v_{out}.



Dimostriamo che G ha ciclo hamiltoniano se e solo se G’ lo ha.

(→) Assumiamo che G abbia un ciclo hamiltoniano orientato Γ. Allora G’ ha un

ciclo hamiltoniano non orientato che attraversa le triple corrispondenti ai nodi esattamente nello stesso ordine.

(←) Assumiamo che G’ abbia un ciclo hamiltoniano non orientato Γ’. Assegniamo delle sigle per ogni tipologia di nodo:

- B=v;
- R=v_{in};
- G=v_{out}.

Γ’ deve visitare tutti i nodi in G’ usando uno dei due ordini:

- B, G, R, B, G, R, B;
- B, R, G, B , R, G, B.

I nodi B in Γ’, formano un circuito hamiltoniano orientato Γ in G, o l’inverso di un ciclo hamiltoniano orientato.

Teorema:

3-SAT $\leq_p$ Dir-ham-cycle

Dim.

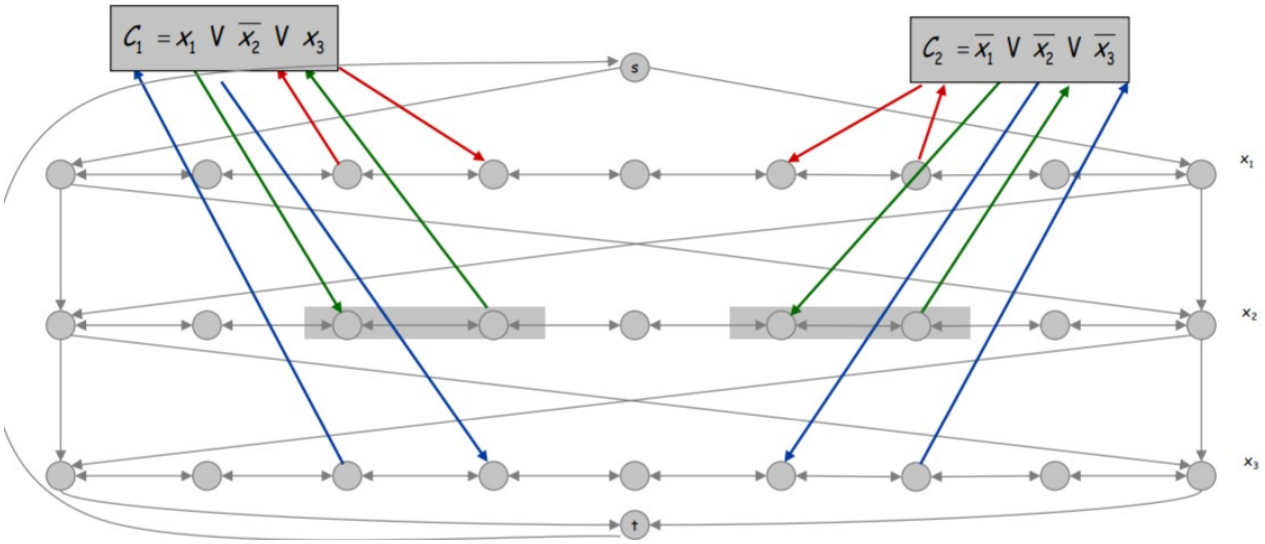
Data istanza ϕ di 3-SAT, costruiamo un'istanza di Dir-ham-cycle che ammette un circuito hamiltoniano se e solo se ϕ è soddisfacibile.

Costruzione

Supponiamo che ϕ abbia n variabili, le possibili assegnazioni di verità a tali variabili sono 2^n. Quindi creiamo un grafo che ha 2^n cicli hamiltoniani che corrispondono in modo naturale alle 2^n possibili assegnazioni di verità per ϕ. Creiamo il grafo G nel seguente modo:

1. In base al numero di variabili in ϕ, creiamo n righe, ogni riga rappresenta una variabile presente in ϕ;
2. Ogni riga ha 3k +3, dove k è il numero di clausole in ϕ, nodi fra loro connessi in ambedue le direzioni;
3. Abbiamo un nodo s, ovvero il nodo di partenza, che ha degli archi uscenti che vanno nelle due estremità della prima riga;
4. I nodi presenti nell'estremità di sinistra e destra di ogni riga, creata al passo 2, hanno un arco partente da tali nodi ed entrante nei nodi sottostanti (in verticale), tranne ovviamente per l'ultima riga, che viene trattata nel passo 5;
5. Aggiungiamo un nodo t, l'ultima riga ha un nodo partente dalle due estremità ed entrante in t. A sua volta, t ha un arco che parte da esso e va nell'arco di partenza s (ci serve per formare un ciclo);
6. Per ogni clausola identifichiamo due colonne successive di nodi nelle righe creati dalle variabili e lasciamo una colonna libera.
7. Infine, aggiungiamo un nodo per ogni clausola che ha due archi collegati a due nodi della riga associata alla variabile e presente nella colonna selezionata per la clausola. Se la variabile è negata, ha un arco che parte dal nodo clausola e

va a finire nel primo nodo della riga selezionata ed un arco che parte dal secondo nodo della suddetta riga e va nel nodo clausola. Nel caso di variabile non negato, la disposizione degli archi è il contrario, cioè primo nodo arco entrante nel nodo clausola e da nodo clausola a secondo nodo.



Dimostriamo che ϕ è soddisfacibile se e solo se G ha ciclo hamiltoniano

(→) Assumiamo istanza 3-SAT che ha un'assegnazione che la soddisfa, chiamiamola x*. Costruiamo il circuito hamiltoniano in G. Dove se attraverso la riga i, posso farlo:

- Da sinistra a destra e assegno x_i*=1;
- Da destra a sinistra e assegno x_i*=0;

Per ogni C_j esiste la riga i che l'attraversa nella direzione che rende l'intera clausola vera, altrimenti un nodo C_j non sarebbe attraversato e non ci sarebbe un ciclo in G.

(←) Assumiamo che G abbia un ciclo hamiltoniano Γ. Se Γ entra nel nodo C_j, deve uscire tramite l'arco uscente dalla clausola. I nodi immediatamente prima e dopo C_j, sono connessi da un arco in G. Se rimuoviamo C_j dal ciclo, sostituendoli con tale arco, abbiamo un

ciclo hamiltoniano su G-{C_j}. Iterando restiamo con ciclo hamiltoniano Γ' in G-{ C_1, C_2, …, C_k} . Poniamo x_i*=1 se e solo se Γ' attraversa la riga i da sinistra a destra, x_i*=0 se da destra a sinistra. Poiché Γ visita ogni C_j, almeno ogni clausola è attraversata e quindi soddisfatta.

Commesso viaggiatore(TSP): Dato un insieme di n città e distanze d(u,v), esiste un tour di lunghezza ≤ D? (per tour intendiamo un percorso che parta da un punto, tocchi tutte le città almeno una volta e torni al punto di partenza).

Teorema:

Ham-cycle $\leq_p$ TSP

Dim.

Data un'istanza G=(V,E) di Ham-cycle, creiamo n città che corrispondono ai nodi del grafo. Per ogni coppia stabiliamo la distanza fra i due nodi nel seguente modo:

$$d(u,v)=\begin{cases}1 \, if \, (u,v)\in E\\2 \, if \, (u,v)\notin E\end{cases}$$

Il bound D lo poniamo uguale ai vertici presenti nel grafo, D=V. Se il grafo di partenza ha un circuito hamiltoniano, seguendo gli archi del grafo, posso toccare i nodi(le città) esattamente una sola volta, quindi ogni arco avrà peso 1. Istanza di TSP ha tour di lunghezza ≤ D se e solo se G è hamiltoniano. Perché per passare da una città all'altra devo per forza usare un arco di peso 2 se il ciclo non è hamiltoniano, o passare più volte per lo stesso nodo, quindi > D.

PROBLEMI DI PARTITIONING

K-color: Dato un grafo non orientato G, esiste un modo di colorare i nodi usando k colori in modo che nodi adiacenti non abbiano lo stesso colore?

Allocazione di registri: Assegna le variabili di un programma ai registri della macchina, in modo tale che non più di k registri siano utilizzati e non esistano 2 variabili che siano utilizzate contemporaneamente e siano assegnate allo stesso registro.

Teorema:

K-color $\leq_p$ k-register-allocation per $k \geq 3$

Dim.

Data un'istanza di K-color G=(V,E), possiamo creare un grafo dei conflitti G'=(V',E'), dove:

- V'={variabili};
- (u,v) $\in$ E' se esiste un'operazione dove u è una variabile concorrente con v del problema.

Se il grafo di partenza è colorabile, posso dare colori diversi ai vertici collegati. Il fatto che esistano le colorazioni, posso vederli come registri, se due variabili sono in conflitto hanno un arco che li unisce, ma hanno registri diversi. Quindi k-register-allocation è risolvibile se e solo se 3-color lo è.

Teorema:

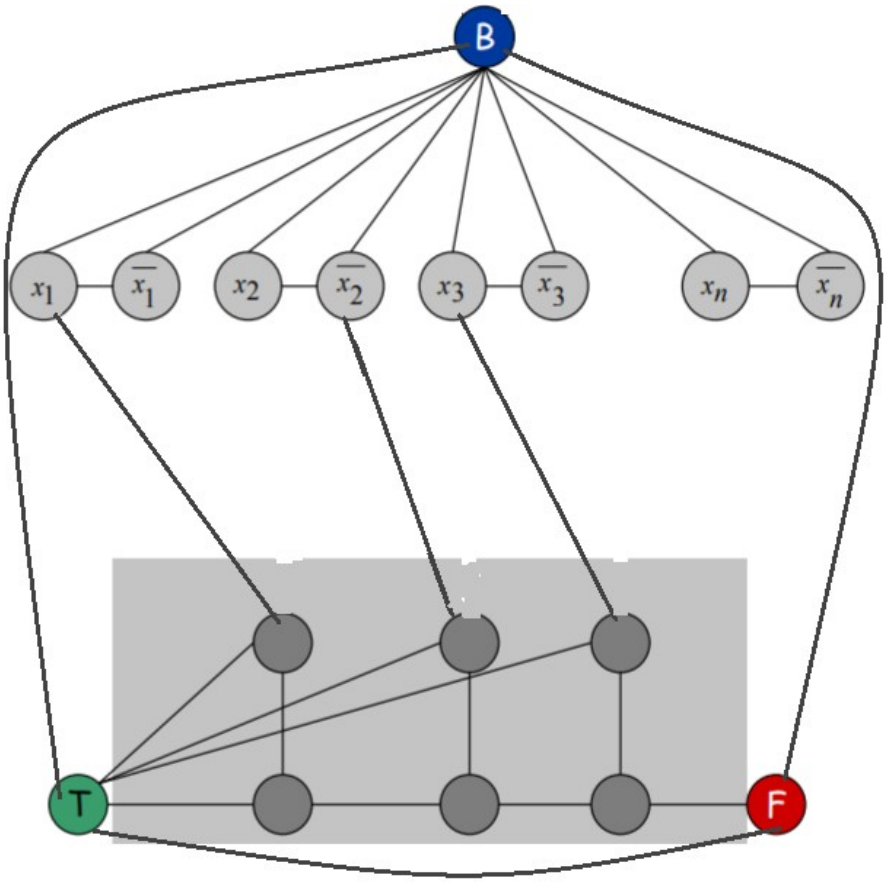
3-SAT $\leq_p$ 3-color

Dim.

Data istanza 3-SAT ϕ, costruiamo istanza di 3-color che è 3-colorabile se e solo se ϕ è soddisfacibile.

Costruzione:

1. Per ogni variabile creiamo 2 nodi, uno per il negato e l’altro per la variabile non negata;
2. Creiamo 3 nodi, T (true), F (false), B (base) e li colleghiamo fra di loro in modo da formare un triangolo (assicura che T, F, B, abbiano 3 colori diversi);
3. Connettiamo ogni letterale a B ;
4. Colleghiamo ogni variabile al suo negato (assicura che ogni variabile e il suo negato abbiano colori diversi);
5. Per ogni clausola, aggiungiamo un gadget di 6 nodi e 13 archi (assicura che almeno un letterale in ogni clausola è a T) e colleghiamo ogni variabile della clausola, al gadget (il rettangolo evidenziato in grigio nella figura sottostante).

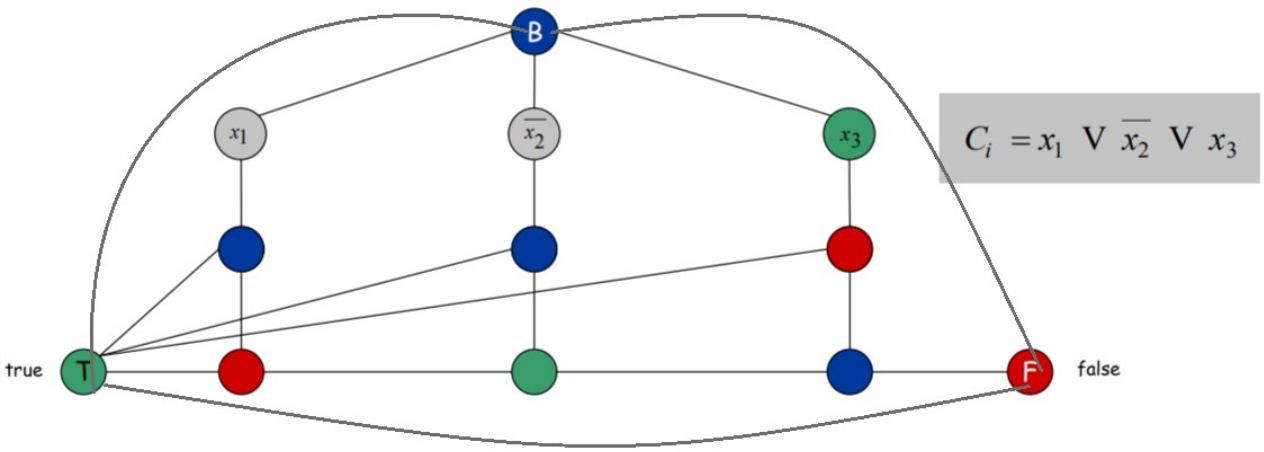


Dimostriamo che l’istanza 3-SAT ϕ, è soddisfacibile se e solo se il grafo creato è 3-colorabile.

(→) Assumiamo che il grafo sia soddisfacibile, ogni variabile e il suo negato hanno colori diversi, per via del collegamento descritto al punto 4. Ogni clausola ha almeno una variabile di valore true grazie al gadget (punto 5), quindi ϕ è soddisfacibile.

(←) Assumiamo che ϕ è soddisfacibile . Possiamo assegnare ai letterali il nodo T di un colore e F di un altro, in base alle assegnazioni di ϕ. Coloriamo i nodi sotto il letterale true con F e sotto true con B. Coloriamo i nodi restanti nel gadget in modo che non ci siano due colori vicini per come è costruito il gadget, se è presente almeno un letterale true nella clausola, sarà colorabile. Ciò sarà valido per tutte le clausole, quindi sarà colorabile l’intero grafo.

Quindi posso ridurre istanze 3-SAT a 3-color in tempo polinomiale perché il grafo passa da una formula con n variabili, ad un grafo con 2n+3 + 6c , ove c sono le clausole, nodi.



PROBLEMI NUMERICI

Subset sum: Dati numeri naturali w_1, …, w_n e intero W, esiste un sottoinsieme di w_i la cui somma è esattamente W?

Nota: Con problemi aritmetici input interi sono codificati in binario. La lunghezza di ognuno corrisponde al logaritmo dell'intero.

Teorema:

3-SAT $\leq_p$ Subset-sum

Dim.

Data istanza ϕ di 3-SAT, costruiamo istanza di subset-sum che ha soluzione se e solo se ϕ è soddisfacibile.

Costruzione:

Data istanza ϕ con n variabili e k clausole, formiamo 2n+2k interi, ognuno di questi interi ha lunghezza n+k digit. Creo una tabella come segue:

1. Una colonna per ogni variabile e clausola;
2. Le righe sono formate da variabile non negata e negata;
3. Nella matrice inseriamo 1 se la variabile della riga fa riferimento alla variabile della colonna (anche negata) o si trova all'interno della clausola della colonna;
4. Le k righe restanti associate alle colonne in corrispondenza delle variabili hanno tutti 0;
5. Le k righe restanti associate alle clausole, hanno una matrice identità formata da 1 e 2;
6. Aggiungiamo un ultima riga W, che è un numero formato di 1 in corrispondenza delle colonne associate alle variabili e di 4 in corrispondenza delle colonne associate alle clausole.

$$C_1 = \bar{x} \vee y \vee z$$
$$C_2 = x \vee \bar{y} \vee z$$
$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

	x	y	z	C_1	C_2	C_3	
x	1	0	0	0	1	0	100,010
¬x	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
¬y	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
¬z	0	0	1	0	0	1	1,001
	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444

(→)Se ϕ è soddisfacibile, prendiamo i numeri corrispondenti ai letterali corrispondenti ai valori true. In ogni clausola, avrò un letterale vero, essendo ϕ soddisfacibile, risulta che almeno uno dei tre letterali è vero. Se sommo la colonna ho al più 3 come valore e almeno 1. Essendo che ho creato una matrice identità in corrispondenza della clausole con 1 e 2 , posso sommare nel caso ci sia in ogni clausola almeno un letterale a 1 per raggiungere 4. Però nel caso non avessi una clausola con almeno un valore a true, non potrei farlo, perché il massimo che posso aggiungere è 2+1, quindi 3 e non raggiungo 4. Quindi se la formula di partenza è soddisfacibile, cioè almeno un letterale in ogni clausola è a true, lo sarà anche la costruzione subset-sum.

(←) Viceversa, se ho sottoinsieme di subset-sum, considero le righe corrispondenti ai letterali true, la somma 4 assicura che ogni clausola abbia un letterale vero ed 1 messo nella riga W in corrispondenza delle variabili, evita di prendere una variabile e il

suo negato (la somma sarebbe 2). Poniamo a true i letterali che formano W e di conseguenza ϕ sarà soddisfatta.

> Scheduling con relase time: Dato un insieme di n jobs con processing time t_i (tempo per processare job i), tempo di rilascio r_i (quando il job i è disponibile all'esecuzione) e deadline d_i (il tempo entro il quale il job deve essere processato). È possibile schedulare tutti i jobs su una sola macchina in modo che job i è processato in uno slot t_i contiguo nell'intervallo $[r_i, d_i]$?

Teorema:

Subset sum $\leq_p$ Schedule relase time

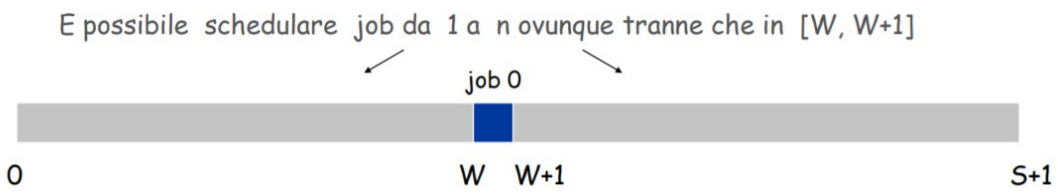
Dim.

Data istanza di subset sum, costruiamo istanza di schedule relase time che ha soluzione se e solo se subset sum è soddisfacibile.

Costruzione:

Data un'istanza di subset sum : $w_1, \ldots, w_n$, e target W.

1. Creiamo n jobs con processing time $t_i = w_i$, tempo di rilascio $r_i=0$ e nessuna deadline $d_i=(1+ \sum_{i=1}^{n} w_i)$.
2. Creiamo job 0 con $t_0=1$, $r_0=W$, $d_0=W+1$.



(→)Se l'istanza di SUBSET-SUM è sì, allora abbiamo una serie di job che possiamo pianificare senza spazi dal tempo 0 a W, e se poi pianifichiamo i lavori rimanenti uno dopo l'altro dopo il tempo W + 1, poiché la deadline di ognuno è $1+\sum_{i=1}^{n} w_i$ rispettiamo le scadenze. Quindi è istanza sì di schedule relase.

(←)Se l'istanza di schedule relase è sì, ci sono n jobs che hanno scadenza $1+\sum_{i=1}^{n} w_i$. Poiché tutte le loro scadenze sono al massimo $1+\sum_{i=1}^{n} w_i$. Ma il job 0 deve essere eseguito da W a W + 1, Se il subset sum sarà valido, ci saranno w_i elementi continui fino a W che posso prendere , poiché vorrà dire che avrò raggiungo con la somma dei t_i ovvero dei w_i il target W. Inoltre possono essere eseguiti solo senza spazi vuoti fino a W, altrimenti qualche job supera $1+\sum_{i=1}^{n} w_i$, poiché il job 0 deve essere eseguito da W a W+1 (il + 1 davanti alla sommatoria), e risulterebbe qualche w_i in ritardo rispetto la deadline. Quindi deve essere un'istanza sì di subset-sum.

Dick Karp
(1972)
1985 Turing
Award