Classi di complessità

A.A. 2015-2016

Classi di complessità

Per parlare di classi di complessità e trattare in modo omogeneo problemi estremamente eterogenei si fanno alcune ipotesi:

1. problemi di decisione.
2. modelli di calcolo (RAM).
3. codifiche dati “concisa”.
4. complessita` a costi logaritmici

1. Problemi di decisione

Un problema di decisione non può essere più difficile di uno di ricerca o di ottimizzazione. Se si dimostra l’intrattabilità computazionale di un problema in forma decisionale, si dimostra l’intrattabilità del problema formulato in uno degli altri modi.

Modello computazionali: RAM

La classe di funzioni calcolabili con programmi RAM coincide con la classe di funzioni calcolabili con vari formalismi: macchina di Turing, λ- calcolo, ... L'indipendenza dai formalismi rende questa classe di funzioni, dette **ricorsive parziali,** estremamente *robusta*, tanto che si è proposto di identificare il concetto di "problema risolubile per via automatica" con la classe delle funzioni ricorsive parziali (**Tesi di Church-Turing**).

Un secondo risultato può essere esteso dalla macchina RAM ad altri formalismi: la classe delle funzioni calcolabili *in tempo limitato da un polinomio.*
Questa proprietà di invarianza rende tale classe interessante ed è stato proposto di identificarla con la classe dei "problemi praticamente risolubili per via automatica" (**Tesi di Church estesa**).

Classi di complessità

Un altro modello: la Macchina di Turing.

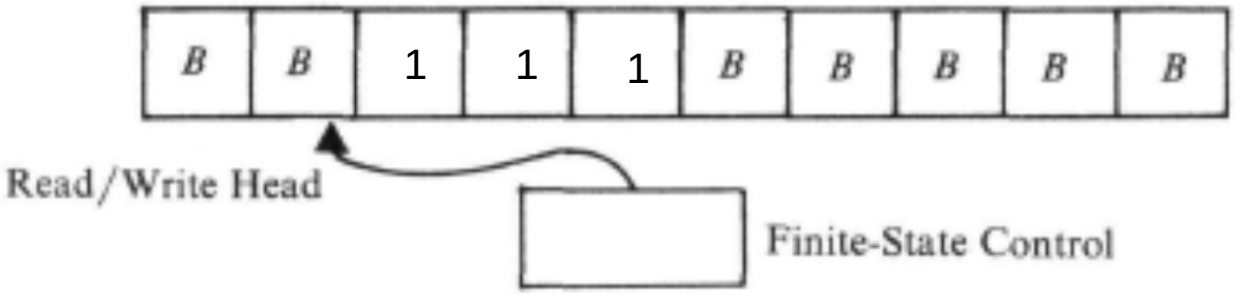


Figure 1 A Turing machine: The tape can be extended indefinitely by adding *B*'s at either end.

Aggiunge un "1" ad una sequenza di 1

$$
\begin{array}{llllll}
(q_0 & B & q_1 & B & R) \\
(q_1 & 1 & q_1 & 1 & R) \\
(q_1 & B & q_2 & 1 & L) \\
(q_2 & 1 & q_2 & 1 & L) \\
(q_0 & 1 & q_H & 1 & N) \\
(q_2 & B & q_H & B & N)
\end{array}
$$

Configurazione istantanea: $X_1 \ldots X_i \; q \; X_{i+1} \ldots X_n$

Una computazione è una successione di configurazioni istantanee

Esempio: q_1 B1 1 1 **|-** $q_1$1 1 1 **|-** 1 q_1 1 1 **|-** 1 1 q_1 1 **|-** 1 1 1 q_1 **|-** 1 1 1 q_2 1 **|-*** q_HB 1 1 1 1

Un altro esempio

"codice" simulare un'assegnazione in binario

q_0	1	B	R	q_0	(blanks the first block)
q_0	B	#	R	q_1	(leaves marker)
q_1	1	1	R	q_1	(moves to block 3)
q_1	B	B	R	q_2	
q_2	1	•	R	q_2	(changes 1's to •'s)
q_2	B	+	L	q_3	(leaves right marker)
q_3	•	1	L	q_4	(converts right-most • to 1)
q_4	•	•	L	q_4	(proceeds back to block 1)
q_4	1	1	L	q_4	
q_4	B	B	L	q_4	
q_4	#	#	L	q_5	(enters last block)
q_5	1	1	L	q_5	(moves to left end of block)
q_5	B	1	R	q_6	(marks a left-most 1)
q_6	1	1	R	q_6	(moves to right end of block 3)
q_6	B	B	R	q_6	
q_6	#	#	R	q_6	
q_6	•	•	R	q_6	
q_6	+	+	L	q_7	(turns around)
q_7	1	1	L	q_7	(finds right-most •, converts it to a 1, and repeats)
q_7	•	1	L	q_4	
q_7	B	B	R	q_8	(all block 3 •'s converted; head moves right)
q_8	1	1	R	q_8	
q_8	+	B	L	q_9	(right end marker erased)
q_9	1	1	L	q_9	(head moves left until # found)
q_9	B	B	L	q_9	
q_9	#	B	L	q_{10}	
q_{10}	1	1	L	q_{10}	(left-most block of 1's traversed)
q_{10}	B	1	L	q_H	(extra 1 inserted)

Modello computazionali

Per studiare in modo sistematico le classi di complessità è necessario chiarire su quali *modelli di calcolo* ci si deve basare.

E' importante che le classi di complessità che si vogliono confrontare siano, in qualche modo, indipendenti dal modello di calcolo considerato.

Per quanto molto differenti un programma su una macchina RAM può essere trasformato in modo *polinomiale* in un programma per una macchina di Turing (e vice-versa).

Un altro modello simile, sia pure un poco più informale, che gode delle stesse proprietà è rappresentato dallo pseudo-codice che usiamo per descrivere gli algoritmi. Certamente più leggibile e praticaible di un linguaggio di tipo assembler.

Per classi di complessità che non inferiori a quella polinomiale i risultati sono indipendenti dal modello ci calcolo scelto

3. Codifiche dati “concisa”

In un calcolatore tutti i dati sono rappresentati con sequenze di bit. Rappresentare i dati di un problema in base 2 o in base 5 non influenza la risolubilità polinomiale del problema. Infatti le rappresentazioni degli interi in qualsiasi base > 1 sono polinomialmente correlate, cioè si può passare dalla rappresentazione in una base a quella in un’altra con un algoritmo in tempo polinomiale. In altre parole, la dimensione dell’input rappresentata in una qualunque base b > 1, è polinomialmente correlata alla dimensione dello stesso input rappresentata in un’altra qualsiasi base b’ >1.
Non è così per la base unaria, in cui servono k simboli per rappresentare il numero k. Usando una base unaria un problema può diventare “difficile” solo a causa di una codifica inopportuna.

3. Costi logaritmici

Il criterio principale solitamente usato per valutare il comportamento di un algoritmo è basato sull'analisi asintotica della sua complessità in tempo; in particolare l'ordine di grandezza di tale quantità al tendere del parametro |x| ad infinito, fornisce una valutazione della rapidità di incremento del tempo di calcolo al crescere delle dimensioni del problema.

Tale valutazione è di solito sufficiente per stabilire se un algoritmo è utilizzabile e per confrontare le prestazioni di algoritmi diversi: una differenza anche piccola nell'ordine di grandezza della complessità di due procedure può comportare enormi differenze nelle prestazioni.

Classi di complessità

Gli algoritmi con complessità in tempo lineare o di poco superiore sono utilizzabili in maniera efficiente anche per elevate dimensioni dell'input.

Algoritmi con complessità dell'ordine di n^k , per $k \geq 2$, sono applicabili solo quando la dimensione dell'ingresso non è troppo elevata; in particolare per $2 \leq k < 3$, si possono processare in tempi ragionevoli istanze di dimensione media, mentre per $k \geq 3$ tale dimensione si riduce drasticamente e i tempi necessari per processare input di lunghezza elevata risultano inaccettabili.

Complessità computazionale dei programmi RAM

Vi sono essenzialmente *due criteri* usati per determinare le quantità di tempo e di spazio consumate dall'esecuzione di un programma RAM su un dato input.
Il primo è il ***criterio di costo uniforme*** secondo il quale l'esecuzione di ogni istruzione del programma richiede un'unità di tempo indipendentemente dalla grandezza degli operandi.
Analogamente lo spazio richiesto per l'utilizzo di un registro della memoria è di una unità indipendentemente dalla dimensione dell'intero contenuto.

Definizione. Un programma RAM P su input x richiede tempo di calcolo t e spazio di memoria s secondo il criterio di costo uniforme se la computazione di P su x esegue t istruzioni e utilizza s registri della macchina RAM, con la convenzione che $t = +\infty$ se la computazione non termina e $s = +\infty$ se vengono utilizzati un numero illimitato di registri.

Complessità computazionale dei programmi RAM

Se le dimensioni degli interi contenuti nei registri RAM diventano *molto grandi* rispetto alle dimensioni dell'input, risulta arbitrario considerare costante il costo di ciascuna istruzione.
Per questo motivo il criterio di costo uniforme è considerato un metodo di valutazione realistico solo per quegli algoritmi che non incrementano troppo la dimensione degli interi calcolati.
Ciò vale ad esempio per gli algoritmi di ordinamento e per quelli di ricerca.

Una misura più realistica di valutazione del tempo e dello spazio consumati da un programma RAM può essere ottenuta attribuendo ad ogni istruzione un costo di esecuzione che dipende dalla dimensione dell'operando.
Otteniamo in tal modo il ***criterio di costo logaritmico*** così chiamato perchè il tempo di calcolo richiesto da ogni istruzione dipende dal numero di bit necessari a rappresentare gli operandi.

Complessità computazionale dei programmi RAM

Per ogni intero n, la lunghezza della sua rappresentazione binaria è $\ell(n) = \lfloor \log_2 |n| \rfloor + 1$, dove |n| è il valore assoluto di n.

Definizione. Il tempo di calcolo *t* richiesto dal programma *P* su input *x* secondo il criterio di costo logaritmico è la somma dei costi logaritmici delle istruzioni eseguite nella computazione di *P* su *x*.

È evidente che per ogni programma *P* e per ogni input x, il costo misurato con il criterio del costo uniforme risulta uguale o inferiore al costo misurato secondo il criterio del costo logaritmico:

$$T(x) \leq T^L(x).$$

Per certi programmi i valori di $T(x)$ e $T^L(x)$ possono differire drasticamente portando a valutazioni diverse dell'efficienza di un algoritmo.

Complessità computazionale dei programmi RAM

Ad esempio consideriamo il seguente algoritmo che calcola la funzione $y = 3^{2^n}$, avendo in input un numero naturale n.

```
potenza (n)
    y ← 3
    x ← n
    while (x > 0)
            y ← y * y
            x ← x - 1
    return y
```

Si verifica immediatamente che $T(n) \in \Theta(n)$.

Poichè dopo la k-esima iterazione del ciclo while y contiene l'intero 3^{2^k}, il costo logaritmico delle istruzioni RAM è dell'ordine di $2^k \cdot \log 3$.
Di conseguenza $T^L(n) \in \Theta(2^n)$.

In modo analogo possiamo definire secondo il criterio logaritmico la quantità di memoria consumata da un programma su un dato input.
Lo spazio occupato in un certo stato della computazione è la somma delle lunghezze degli interi contenuti nei registri usati dal programma in quell'istante.
Lo spazio complessivo richiesto è quindi il massimo di questi valori al variare degli stati raggiunti durante la computazione.

Le classi P e NP

P: classe dei problemi di decisione per la cui soluzione esistono algoritmi ***deterministici*** di ***complessità polinomiale*** nella dimensione dei dati in input.

NP: classe dei problemi di decisione per la cui soluzione esistono algoritmi ***non deterministici*** di ***complessità polinomiale*** nella dimensione dei dati in input.

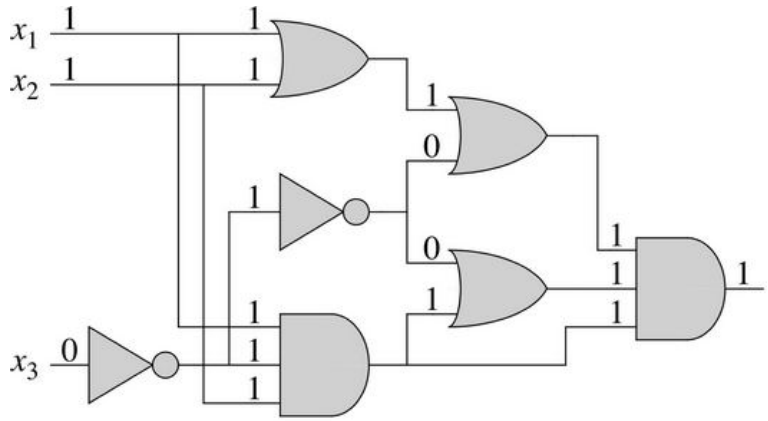
Per i problemi in **NP** non sono ancora stati trovati algoritmi deterministici di soluzione di complessità polinomiale, ma esistono algoritmi di certificazione deterministici di complessità polinomiale.

1. Soddisfattibilità di circuiti

ISTANZA: un circuito combinatorio con porte AND, OR e NOT, con un solo output;

DOMANDA: Esiste un'assegnazione di valori di verità per gli input che renda vera l'uscita y?"

Esempio:



2. Soddisfattibilità di formule

ISTANZA: una formula costruita mediante gli operatori **end**, **or** e **not** su un insieme U di variabili booleane;
DOMANDA: Esiste un'assegnazione di valori di verità su U che soddisfi la formula?

3. Soddisfattibilità di formule (3-SAT)

ISTANZA: una formula in forma normale congiuntiva (CNF) su un insieme U di variabili booleane, in cui ogni clausola sia formata da 3 letterali;
DOMANDA: Esiste un'assegnazione di valori di verità su U che soddisfi la formula?
NOTA: invece 2-SAT è decidibile in tempo polinomiale.

4. Copertura di vertici

ISTANZA: Un Grafo G = (V, E) e un intero positivo K ≤ |V|
DOMANDA: Esiste su G una copertura di dimensione k, cioè un sottoinsieme V' ⊆ V tale che |V'| ≤ K e per ogni arco u,v appartenente a E almeno u o v appartenga a V'?.

5. Insieme indipendente

ISTANZA: Un Grafo G = (V, E) non orientato e un intero positivo K ≤ |V|
DOMANDA: Esiste in G un sottoinsieme di vertici V' ⊆ V tale che |V'| = K e per ogni coppia u,v di vertici appartenenti a V', non esista l'arco (u,v) in E?

6. Copertura esatta di insiemi

ISTANZA: Un insieme X e una famiglia di suoi sottinsiemi Y = $\{Y_1, \ldots, Y_n\}$;
DOMANDA: Esiste una sottofamiglia F di Y che partizioni X?

7. Circuito Hamiltoniano

ISTANZA: Un grafo G non orientato;
DOMANDA: Esiste un circuito che attraversi ogni nodo una e una sola volta?

8. Programmazione lineare 0/1.

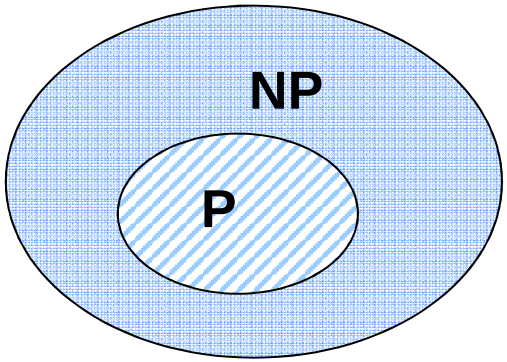
ISTANZA: un matrice A di interi di dimensione m×n, un vettore b di m elementi un
DOMANDA: Esiste un vettore x di n elementi in {0, 1) tali che A x ≤ b ?

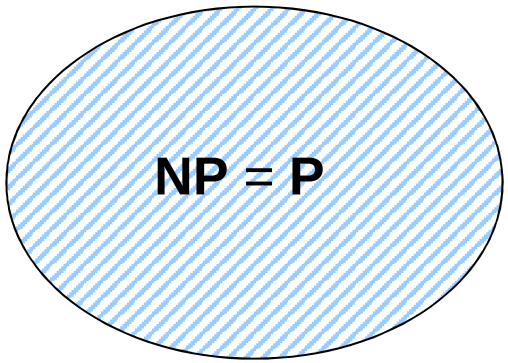
Le classi P e NP

Se un'istanza ammette risposta positiva, esiste una soluzione ammissibile che "testimonia" o "certifica" la validità della risposta mediante una facile verifica.

Si può allora definire **NP** come la classe dei problemi per cui esistono certificati polinomiali.

Ovviamente $\mathbf{P} \subseteq \mathbf{NP}$, ma non si sa se la contenenza è
propria: $\mathbf{P} \subset \mathbf{NP}$ oppure no: $\mathbf{P} = \mathbf{NP}$





Riducibilita` polinomiale

Un problema Q è polinomialmente riducibile ad un problema R: $Q \propto R$ se

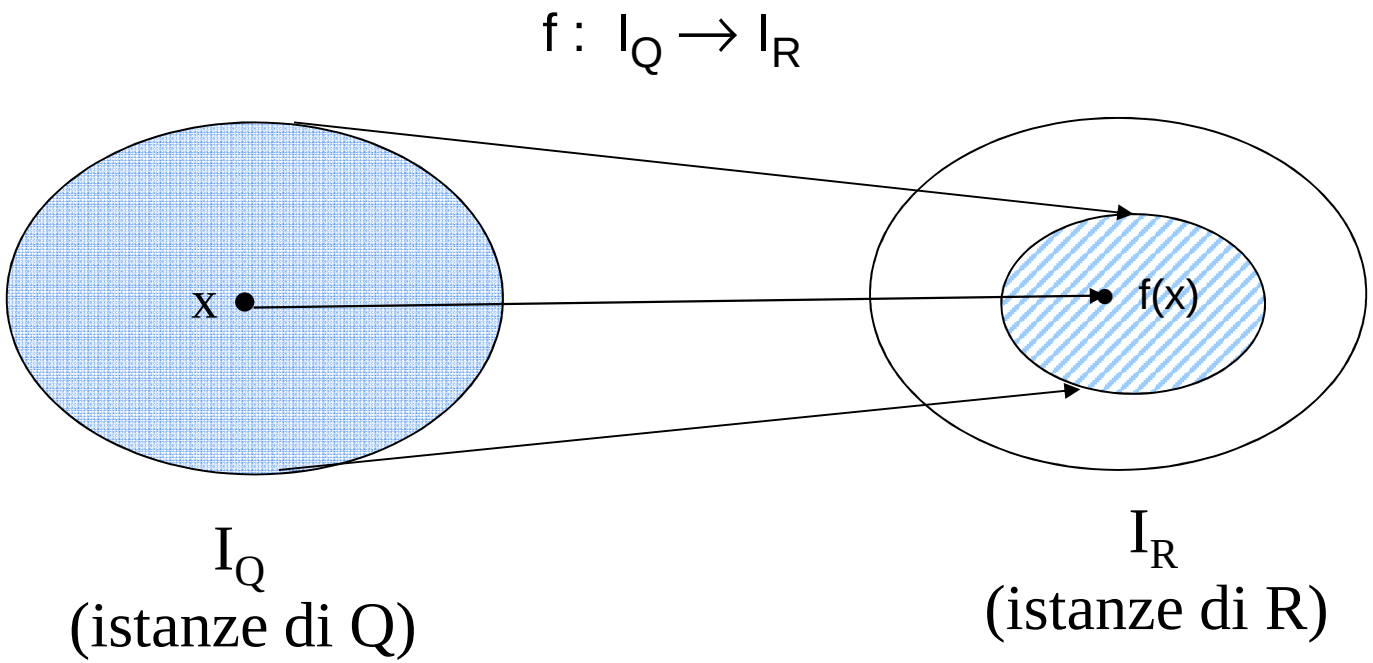
Esiste una funzione f che trasforma le istanze in input per Q in istanze in input per R: $f : I_Q \rightarrow I_R$ tale che

- f sia calcolabile con un algoritmo deterministico di complessità polinomiale.

- Per ogni $x \in I_Q$, Q risponde “vero” sull’input x **se e solo se** R risponde “vero” sull’input f(x)

  $Q(x) \Leftrightarrow R(f(x))$

Riducibilita` polinomiale



Riducibilita` polinomiale e NP-completezza

Teorema

1. $Q \propto R$ and $R \in \mathbf{NP} \Rightarrow Q \in \mathbf{NP}$

2. $Q \propto R$ and $R \in \mathbf{P} \Rightarrow Q \in \mathbf{P}$

Teorema di Cook-Levin

$\forall Q: Q \in \mathbf{NP} \Rightarrow Q \propto$ Domino-limitato

Corollario

$\mathbf{P} = \mathbf{NP} \Leftrightarrow$ Domino-limitato $\in \mathbf{P}$

Definizione

1. Q è **NP**-arduo se $\forall Q`: Q` \in \mathbf{NP} \Rightarrow Q` \propto Q$

2. Q è **NP**-completo se:
   a) Q è **NP**-arduo
   b) $Q \in \mathbf{NP}$

Riducibilita` polinomiale e NP-completezza

Per dimostrare che un problema R è **NP**-arduo basta dimostrare che un problema Q **NP**-arduo è polinomialmente riducibile a R.

Infatti:

a) La relazione $\propto$ è transitiva: $Q \propto R$ and $R \propto S \Rightarrow Q \propto S$

b) Se Q è **NP**-arduo e $Q \propto R$ si ha:

$$(\forall Q`: Q` \in \mathbf{NP} \Rightarrow Q` \propto Q) \text{ and } Q \propto R$$

$\forall Q`: Q` \in \mathbf{NP} \Rightarrow Q` \propto R$, cioè R è **NP**-arduo

Riducibilita` polinomiale

Esempio: Domino-limitato $\propto$ Cricca

Domino-limitato

Scacchiera n x n
m tessere

Cricca

Grafo G con:
- m x n x n vertici: v_{hij}
- archi (v_{hij} , $v_{h'i'j'}$) che indicano che le tessere d_h , $d_{h'}$ possono stare nelle posizioni i,j e i',j'

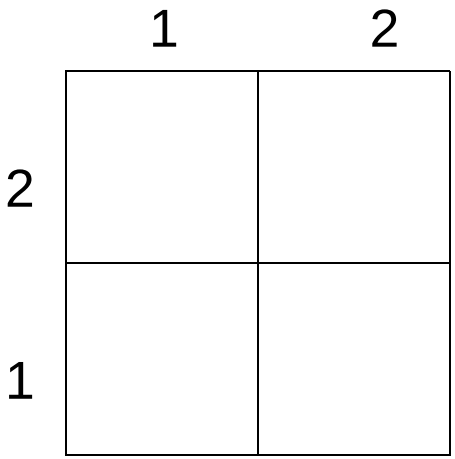
" esiste una copertura della scacchiera ?"

"esiste una cricca di dimensione n x n nel grafo G ?"

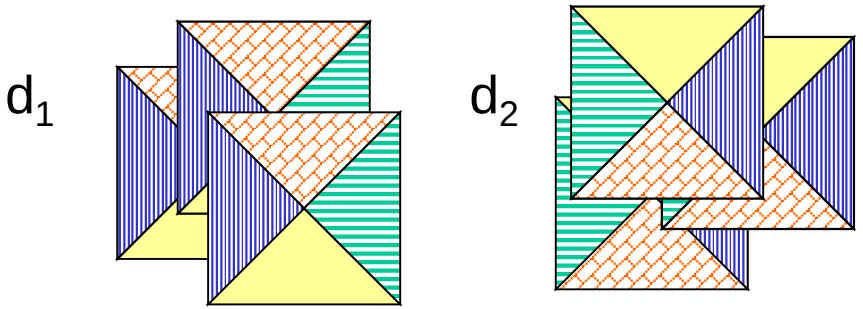
Esempio: Domino-limitato $\propto$ Cricca

Costruiamo il grafo per la seguente istanza di Domino-limitato

Scacchiera 2 x 2

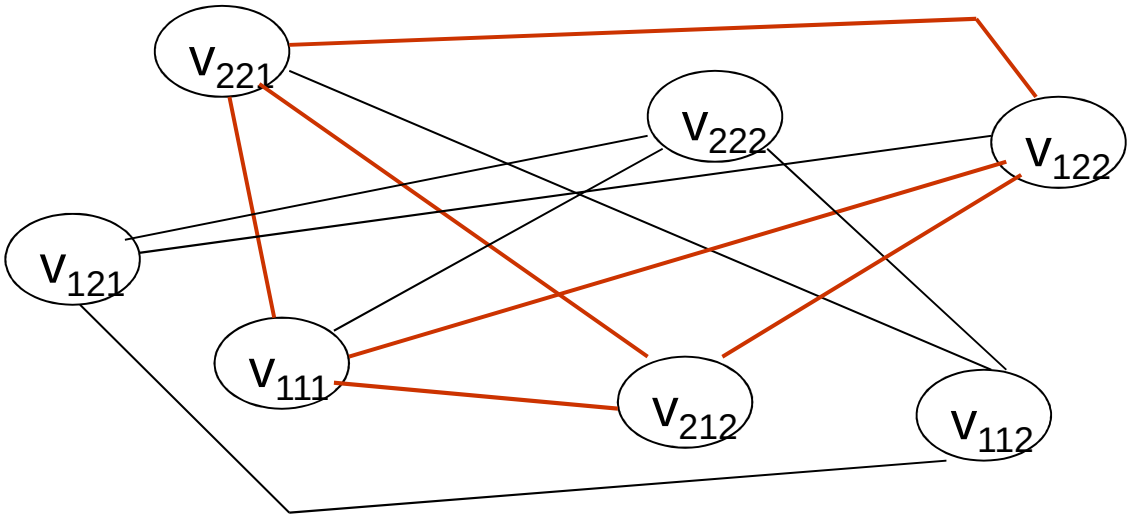


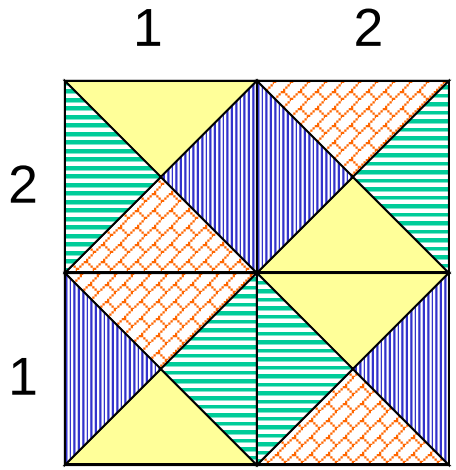
Tessere:



Esempio: Domino-limitato $\propto$ Cricca

Vertici del grafo: $v_{111}, v_{112}, v_{212}, v_{121}, v_{221}, v_{121}, v_{222}$





La cricca di dimensione 4 formata dai vertici $\{v_{111}, v_{212}, v_{122}, v_{221}\}$ fornisce una soluzione all'istanza data di domino limitato.

Teorema di Cook-Levin: $\forall Q: Q \in \mathbf{NP} \Rightarrow Q \propto$ Domino-limitato

Idea della prova (E' ovvio che domino limitato è in **NP**).

- Sia A un problema in NP e T una macchina di Turing nondeterministica che risolve T in tempo polinomiale.
- Una configurazione di una T è rappresentata mediante una stringa ''$\alpha q \beta$'' dove α e β sono i contenuti del nastro e q è lo stato.
- Un passo di calcolo è rappresentato dal passaggio da una configurazione a un'altra
- Poiché A e' in NP per accettare un'istanza di dimensione A il numero di passi necessari sarà dato da un polinomio $p_t(n)$.
- Poiché T non accede a più di una cella alla volta e si sposta solo di una posizione, la lunghezza massima del nastro sarà limitata da $p_t(n)$ celle.
- Si definisce un insieme opportuno di $p_t(n)^k$ tessere, con k costante, su una scacchiera $p_t(n) x p_t(n)$, tali che:
  - ogni riga corrisponde a una configurazione di T;
  - la prima riga è ricopribile solo se l'input x è corretto per A;
  - la riga i sia ricopribile se e solo T consente una transizione dalla configurazione rappresentata dalla riga i-1 a quella della riga i;
  - l'ultima riga sia ricopribile se e solo se la macchina accetta.

Riducibilita` polinomiale

Altro esempio: Domino-limitato $\propto$ soddisfacibilità

Domino-limitato

Scacchiera n x n
m tessere

3-sat

Si introducono le variabili logiche u_{hij} $(1 \leq h \leq m\ ,\ 1 \leq I,j \leq n)$ con il seguente significato

u_{hij} = *true*

se e solo se la tessera dh copre la casella I,j

Domino-limitato $\propto$ soddisfacibilità

Si introducono le seguenti clausole:

1. u_{111} la casella 1,1 è coperta con la tessera 1
2. $\mathbf{and}_{1\le i,j\le n}\ (\mathbf{or}_{1\le h\le m}\ u_{hij})$ almeno una tessera copre i,j
3. $\mathbf{and}_{1\le i,j\le n}\ (\mathbf{and}_{1\le h,h'\le m}\ (\mathbf{not}\ u_{hij}\ \mathbf{or}\ \mathbf{not}\ u_{h'ij})$
   al più una tessera occupa (i,j)
4. $\mathbf{and}_{1\le i\le n-1,\ 1\le j\le n}\ (\mathbf{and}_{1\le h,h'\le m}\ (\mathbf{not}\ u_{hij}\ \mathbf{or}\ \mathbf{not}\ u_{h'(i+1)j})$
   se d_h, $d_{h'}$ *non* possono essere adiacenti
5. $\mathbf{and}_{1\le i\le n,\ 1\le j\le n-1}\ (\mathbf{and}_{1\le h,h'\le m}\ (\mathbf{not}\ u_{hij}\ \mathbf{or}\ \mathbf{not}\ u_{h'i(j+1)})$
   se d_h, $d_{h'}$ *non* possono essere sovrapposte

Domino-limitato $\propto$ soddisfacibilità

Nel caso dell'esempio precedente otteniamo queste clausole:

1. u_{111}
2. (u_{111} **or** u_{211}) **and** (u_{112} **or** u_{212}) **and** (u_{121} **or** u_{221}) **and** (u_{122} **or** u_{222})
3. (**not** u_{111} **or not** u_{211}) **and** (**not** u_{112} **or not** u_{212}) **and** (**not** u_{212} **or not** u_{221}) **and** (**not** u_{122} **or not** u_{222})
4. (**not** u_{111} **or not** u_{121}) **and** (**not** u_{112} **or not** u_{122}) **and** (**not** u_{211} **or not** u_{121}) **and** (**not** u_{212} **or not** u_{122}) **and** (**not** u_{211} **or not** u_{221}) **and** (**not** u_{212} **or not** u_{222})
5. (**not** u_{111} **or not** u_{112}) **and** (**not** u_{121} **or not** u_{122}) **and** (**not** u_{211} **or not** u_{212}) **and** (**not** u_{221} **or not** u_{222})

“ esiste una copertura della scacchiera”

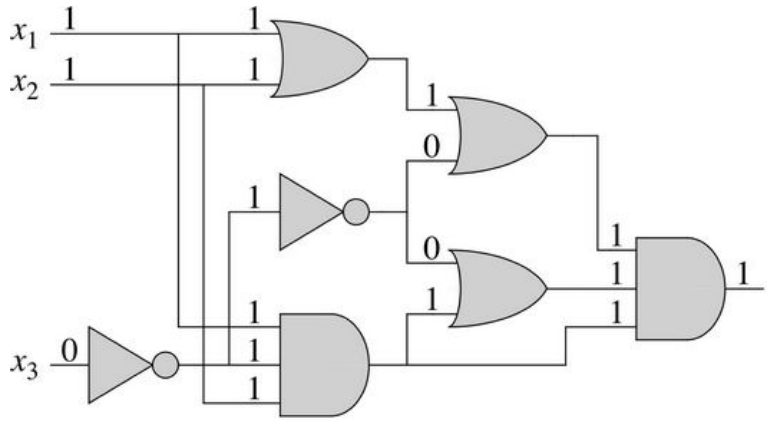
“esiste un'assegnazione di valori di verità che soddisfa tutte le clausole”

1. Soddisfattibilità di circuiti

ISTANZA: un circuito combinatorio con porte AND, OR e NOT, con un solo output;

DOMANDA: Esiste un'assegnazione di valori di verità per gli input che renda vera l'uscita y?"

Esempio:



2. Soddisfattibilità di formule

ISTANZA: una formula costruita mediante gli operatori **end**, **or** e **not** su un insieme U di variabili booleane;
DOMANDA: Esiste un'assegnazione di valori di verità su U che soddisfi la formula?

3. Soddisfattibilità di formule (3-SAT)

ISTANZA: una formula in forma normale congiuntiva (CNF) su un insieme U di variabili booleane, in cui ogni clausola sia formata da 3 letterali;
DOMANDA: Esiste un'assegnazione di valori di verità su U che soddisfi la formula?
NOTA: invece 2-SAT è decidibile in tempo polinomiale.

4. Copertura di vertici

ISTANZA: Un Grafo G = (V, E) e un intero positivo K ≤ |V|
DOMANDA: Esiste su G una copertura di dimensione k, cioè un sottoinsieme V' ⊆ V tale che |V'| ≤ K e per ogni arco u,v appartenente a E almeno u o v appartenga a V'?.

5. Insieme indipendente

ISTANZA: Un Grafo G = (V, E) non orientato e un intero positivo K ≤ |V|
DOMANDA: Esiste in G un sottoinsieme di vertici V' ⊆ V tale che |V'| = K e per ogni coppia u,v di vertici appartenenti a V', non esista l'arco (u,v) in E?

6. Copertura esatta di insiemi

ISTANZA: Un insieme X e una famiglia di suoi sottinsiemi Y = $\{Y_1, \ldots, Y_n\}$;
DOMANDA: Esiste una sottofamiglia F di Y che partizioni X?

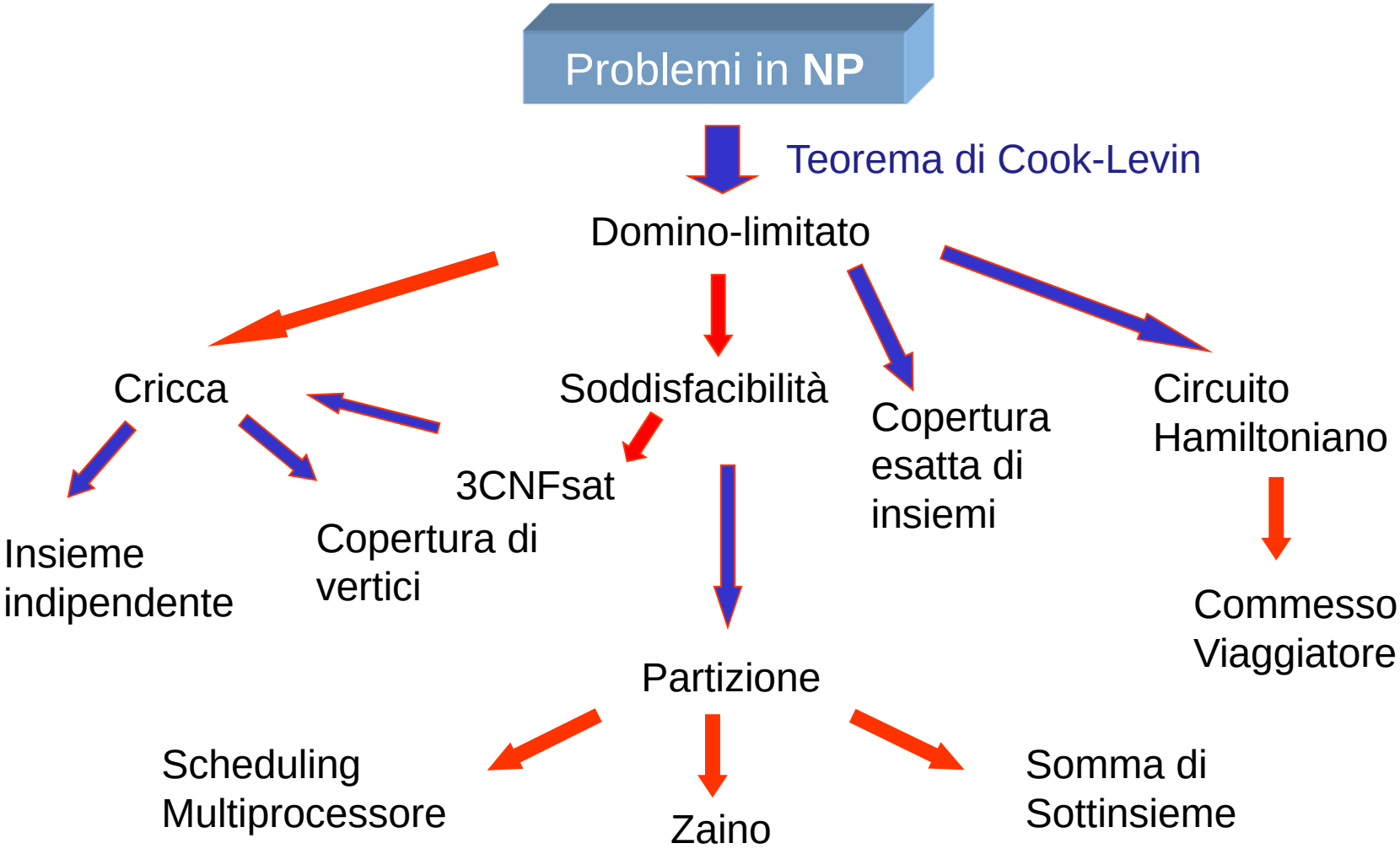
7. Circuito Hamiltoniano

ISTANZA: Un grafo G non orientato;
DOMANDA: Esiste un circuito che attraversi ogni nodo una e una sola volta?

8. Programmazione lineare 0/1.

ISTANZA: un matrice A di interi di dimensione m×n, un vettore b di m elementi un
DOMANDA: Esiste un vettore x di n elementi in {0, 1) tali che A x ≤ b ?

Problemi NP-completi



Riducibilita` polinomiale

Esempio: Circuito di Hamilton $\propto$ Commesso viaggiatore

Circuito Hamiltoniano

G non orientato

V

E

"Esiste in G un ciclo che attraversa ogni vertice una e una sola volta ?"

Commesso viaggiatore

G' non orientato

V' = V

E' = {(r, t) / $\forall$ r, t $\in$ V'}

W(u,v) = 1 se (u,v) $\in$ E
W(u,v) = 2 se (u,v) $\notin$ E

"Esiste in G' un percorso di costo uguale al numero dei vertici?"

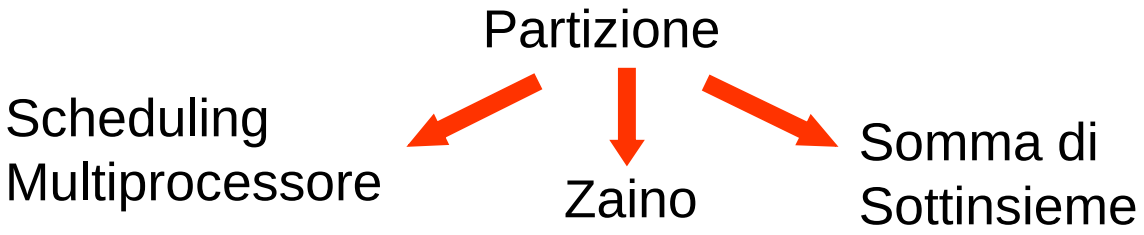
Riducibilita` polinomiale

Partizione: Dato un insieme di interi $A = \{a_1, a_2, \ldots, a_n\}$ "esiste un sottoinsieme S degli indici {1, 2,…, n} tale che $\Sigma_{i \in S} a_i = \Sigma_{i \notin S} a_i$?"

Scheduling multiprocessore: Dati n programmi $\{p_1, p_2, \ldots, p_n\}$, ognuno dei quali, p_i, richiede tempo d'esecuzione t_i, intero positivo, m processori identici e un intero d, "è possibile eseguire tutti i programmi in al più d unità di tempo ?"

Zaino: Dati un insieme $O = \{o_1, o_2, \ldots, o_n\}$ di oggetti, ad ognuno dei quali, o_i, sono associati un valore v_i e un peso p_i, interi positivi, due interi positivi c (capacità) e k (obiettivo) "esiste un sottoinsieme S degli indici {1, 2,…, n} tale che $\Sigma_{i \in S} p_i \leq c$ e $\Sigma_{i \in S} v_i \geq k$?"

Somma di sottoinsieme: Dati un insieme di interi $A = \{a_1, a_2, \ldots, a_n\}$ e un intero positivo k, "esiste un sottoinsieme S degli indici {1, 2,…, n} tale che $\Sigma_{i \in S} a_i = k$?"



Riducibilita` polinomiale

Partizione

$A = \{a_1, a_2, \ldots, a_n\}$

"esiste un sottoinsieme S degli indici $\{1, 2, \ldots, n\}$ tale che $\Sigma_{i \in S} a_i = \Sigma_{i \notin S} a_i$?"

Scheduling multiprocessore

$t_i = a_i \quad m = 2$

$d = (1/2) \Sigma_{i \in \{1,\ldots,n\}} a_i$

"è possibile eseguire tutti i programmi in al più d unità di tempo ?"

Somma di sottoinsieme

$A = \{a_1, a_2, \ldots, a_n\}$

$k = (1/2) \Sigma_{i \in \{1,\ldots,n\}} a_i$

"esiste un sottoinsieme S degli indici $\{1, 2, \ldots, n\}$ tale che $\Sigma_{i \in S} a_i = k$?"

Zaino

$p_i = v_i = a_i \quad c = k = (1/2) \Sigma_{i \in \{1,\ldots,n\}} a_i$

"esiste un sottoinsieme S degli indici $\{1, 2, \ldots, n\}$ tale che $\Sigma_{i \in S} p_i \leq c$ e $\Sigma_{i \in S} v_i \geq k$?"