TEORIA DELLA COMPLESSITÀ

MATERIALE CONSIGLIATO:

TESTO DI RIFERIMENTO SULLA TEORIA DELLA COMPLESSITÀ:

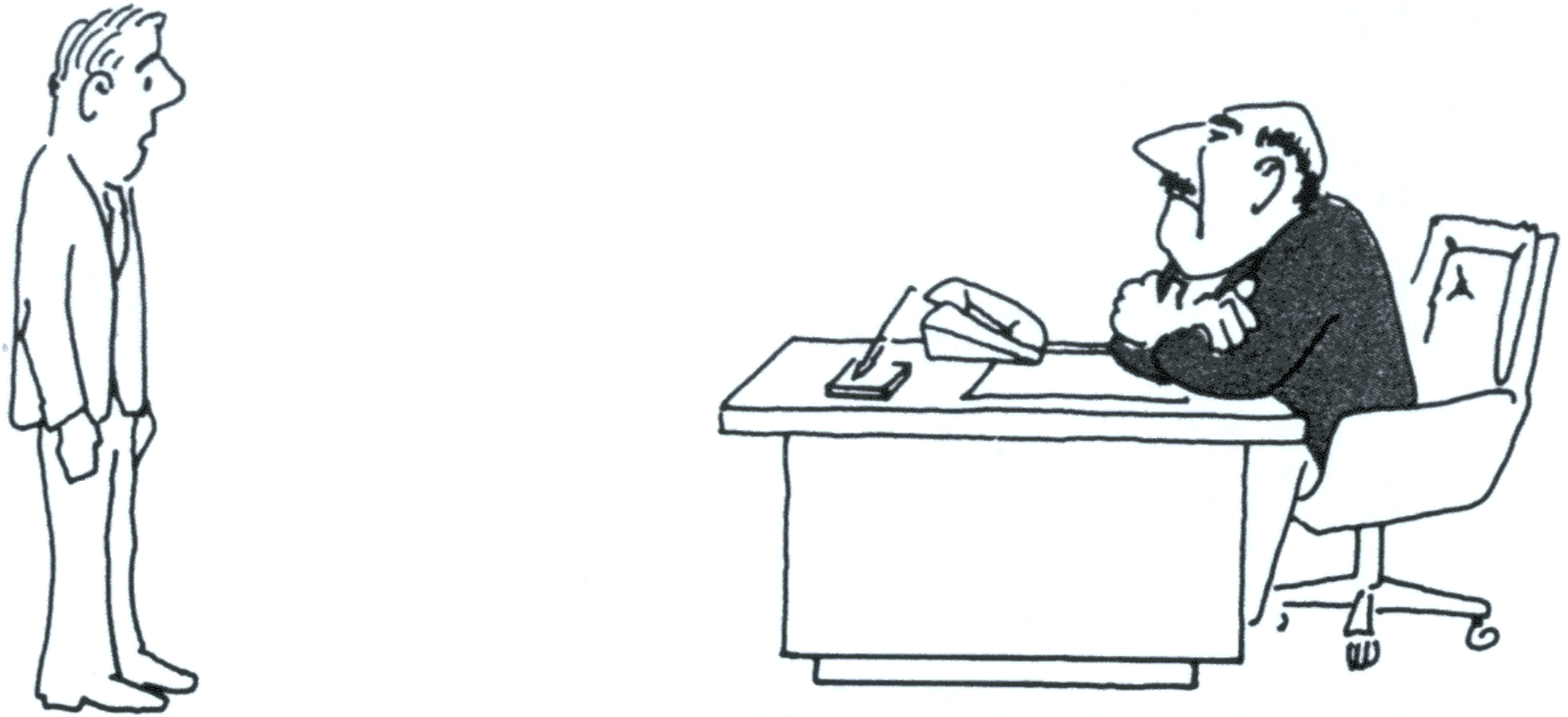
Computers and Intractibility – A Guide to the Theory of NP-Completeness
M. R. Garey, D. S. Johnson
Freeman and Company, 1979.

TESTO DEL CORSO (CAPITOLO 36)

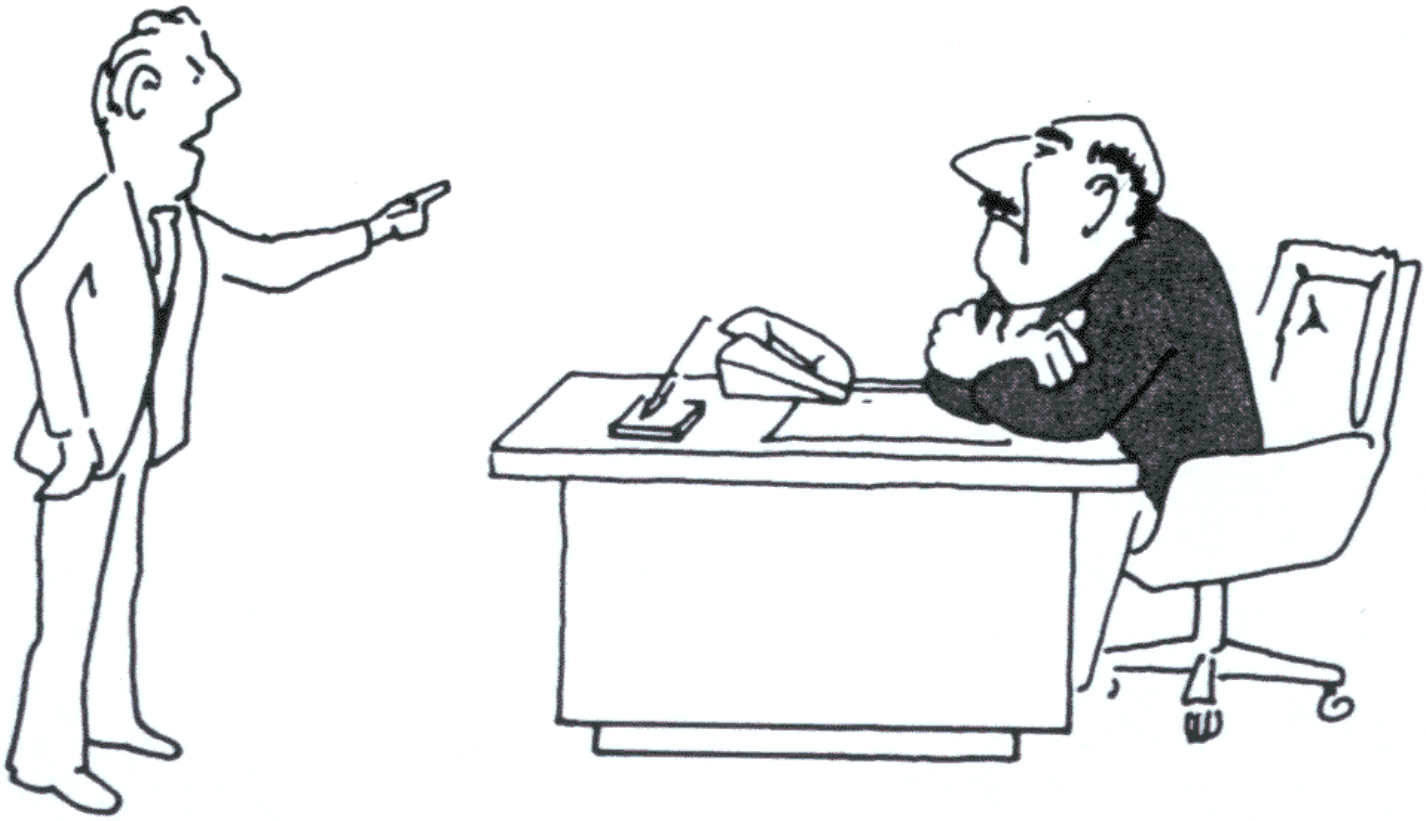
T.H. Cormen, C.E. Leiserson, R. L. Rivest
Introduzione agli algoritmi, 1999.

BIBLIOGRAFIA LUCIDI

Linguaggi Modelli Complessità, G. Ausiello, F. d'Amore, G. Gamosi, Univer. di Roma
Esercizi di Informatica Teorica, L. Cabibbo e W. Didimo, Università di Roma
Teoria della complessità computazionale, D. V., Università di Bologna



"I can't find an efficient algorithm, I guess I'm just too dumb."



"I can't find an efficient algorithm, because no such algorithm is possible!"

"I can't find an efficient algorithm, but neither can all these famous people."

Concetti fondamentali

- Risorse di calcolo considerate:
  - tempo utilizzato
  - memoria impiegata

- La teoria della complessità si occupa di classificare i problemi dal punto di vista delle risorse di calcolo necessarie e/o sufficienti per risolverlo.

Stabilire se un problema sia semplice o difficile.

PROBLEMA FACILE $\Rightarrow \exists$ un algoritmo **efficiente** in grado di risolverlo.
PROBLEMA DIFFICILE $\Rightarrow$ non esiste un algoritmo **efficiente** in grado di risolverlo .

Per esaminare se un algoritmo è efficiente si esamina la complessità computazionale dell'algoritmo.

Cosa è un Problema?

È la richiesta di rispondere a una domanda circa le proprietà di una struttura matematica in funzione di parametri costanti e di valori non specificati di variabili.

Esempio

Dato un numero intero, si vuole sapere se il numero primo.
Struttura matematica = numero intero;
Domanda = il numero è primo?
Parametri e Variabili = il numero intero dato (input del problema).

Esempio

Dato un grafo G = (V, E), si vuole determinare se esiste un percorso hamiltoniano di lunghezza inferiore a k.
Struttura matematica = grafo;
Domanda = il grafo contiene un ciclo hamiltoniano?
Parametri e Variabili = i nodi del grafo G, gli archi e i pesi degli archi.

Def. di Grafo **Hamiltoniano** Un grafo hamiltoniano è un grafo contenente almeno un ciclo Hamiltoniano, ovvero un ciclo che tocca tutti i vertici del grafo una ed una sola volta.

Cosa è un'istanza?

È un problema in cui i valori delle variabili sono specificati.

un problema $\Rightarrow$ molte istanze

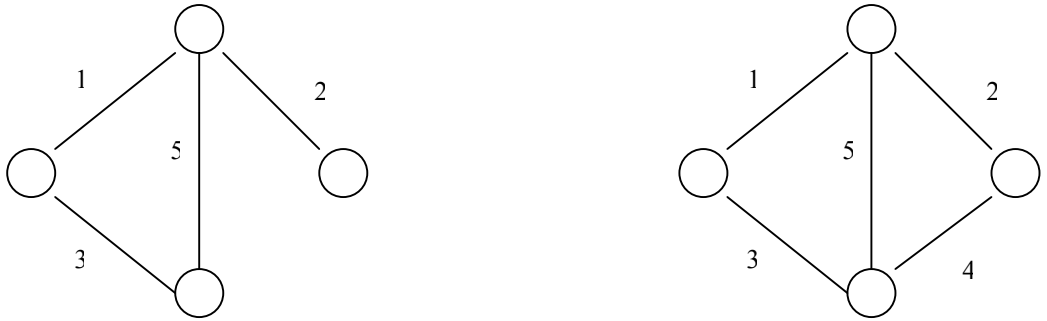
Un problema è l'insieme delle sue istanze.

Esempio

P: Dato un numero intero, si vuole sapere se il numero è primo.
Istanze di P: I_1=5, I_2= 10, I_3=452, ….. ,I_n=1836352183027

Esempio

P: Dato un grafo G, esiste un percorso hamiltoniano di lunghezza inferiore a 11?
Istanze di P:



Tipi di Problemi

Problemi di DECISIONE

- un insieme di istanze I_{PD}.

L'algoritmo per un problema di decisione calcola un predicato π: $I_{PD} \rightarrow \{V, F\}$ che determina una partizione $I_{PD}=Y_{PD} \cup N_{PD}$, $Y_{PD} \cap N_{PD}=\varnothing$ in un insieme Y_{PD} di istanze positive, tali cioè che $x \in Y_{PD}$ se e solo se $\pi(x)=V$, ed un insieme N_{PD} di istanze negative, tali che $x \in N_{PD}$ se e solo se $\pi(x)=F$.

L'algoritmo per un problema di decisione prende in input un'istanza $x \in I_{PD}$ e restituisce in output un valore di verità VERO o FALSO.

Esempio
Nel grafo dato, esiste un ciclo Hamiltoniano di lunghezza inferiore a k?

Tipi di Problemi (cont.)

Problemi di RICERCA

- un insieme di istanze I_{PR};
- un insieme di soluzioni S_{PR};
- una proprietà di ammissibilità $R \subseteq I_{PR} \times S_{PR}$ che deve valere per tutte le soluzioni di un'istanza.

L'algoritmo per un problema di ricerca calcola una funzione $\phi: I_{PR} \rightarrow S_{PR}$ tale che $y=\phi(x)$ è una soluzione per la quale $\langle x, y \rangle \in R$.

Data un'istanza $x \in I_{PR}$, si definisce insieme delle soluzioni ammissibili di x, sol(x):
$sol(x)= \{y \in S_{PR} \mid (x, y) \in R \}$

L'algoritmo per un problema di ricerca prende in input un'istanza $x \in I_{PR}$ e restituisce in output la codifica di una soluzione $y \in sol(x)$.

Esempio
Dato un grafo, trovare un ciclo Hamiltoniano di lunghezza inferiore a k.

Tipi di Problemi (cont.)

Problemi di ENUMERAZIONE

- un insieme di istanze I_{PE};
- un insieme di soluzioni S_{PE};
- una proprietà di ammissibilità $R \subseteq I_{PE} \times S_{PE}$.

L'algoritmo per un problema di enumerazione calcola una funzione $\psi: I_{PE} \rightarrow N$ tale che per ogni $\psi(x) = |sol(x)|$.

L'algoritmo per un problema di decisione prende in input un'istanza $x \in I_{PE}$ e restituisce in output il numero di soluzioni ammissibili dell'istanza.

Esempio
Dato un grafo, trovare il numero di cicli hamiltoniani di lunghezza inferiore a k.

Tipi di Problemi (cont.)

Problemi di OTTIMIZZAZIONE

- un insieme di istanze I_{PO};
- un insieme di soluzioni S_{PO};
- una proprietà di ammissibilità $R \subseteq I_{PO} \times S_{PO}$;
- una funzione di misura $\mu: I_{PO} \times S_{PO} \rightarrow N$;
- un criterio di scelta $c \in \{MIN, MAX\}$.

L'algoritmo per un problema di ottimizzazione prende in input un'istanza $x \in I_{PO}$ e restituisce in output la soluzione $y \in sol(x)$ t.c. per ogni $z \in sol(x)$, $\mu(x, y) \leq \mu(x, z)$ se $c = MIN$ e $\mu(x, y) \geq \mu(x, z)$ se $c = MAX$.

Esempio

Dato un grafo, trovare il ciclo hamiltoniano di lunghezza inferiore.

Codifica

Dato un problema P con insieme delle possibili istanze I_P, e dato un alfabeto Σ, definiamo come schema di codifica di P una funzione e: $I_P \rightarrow \Sigma^*$ che mappa ogni istanza di P in una corrispondente stringa di simboli in Σ^*.

La codifica di un'informazione richiede un numero diverso di simboli a seconda dell'alfabeto usato.

Codifica Ragionevole
- utilizza almeno due simboli;
- non introduce dati irrilevanti, né richiede una generazione esponenziale di dati per la rappresentazione di un'istanza del problema.

Proprietà

Se la codifica è ragionevole la complessità del problema è indipendente dalla codifica scelta per risolvere il problema.

Codifica (cont.)

Esempio

Il numero p è primo?

Codifica decimale: (alfabeto con 10 elementi)	Codifica unaria (alfabeto con un singolo elemento)
1 cifra ⇒ 10 simboli 2 cifre ⇒ 100 simboli 3 cifre ⇒ 1000 simboli 4 cifre ⇒ 10000 simboli ⇒ il numero di simboli cresce di un fattore 10 mentre il numero di cifre cresce come il logaritmo.	1 cifra ⇒ 1 simbolo 10 cifre ⇒ 10 simboli 100 cifre ⇒ 100 simboli 1000 cifre ⇒ 1000 simboli ⇒ il numero di simboli cresce linearmente con il numero di cifre.
CODIFICA RAGIONEVOLE!!!	CODIFICA IRRAGIONEVOLE!!!

Cifra = simbolo dell'alfabeto Σ.

Numero = stringa (sequenza finita di simboli).

Problemi come linguaggio

La teoria della complessità si concentra sui problemi di DECISIONE (sono quelli per cui è necessaria l'introduzione del minor numero di concetti).

La codifica di un problema definisce l'insieme Σ dei simboli sul quale il problema opera.

Problema di Decisione (RIFORMULAZIONE)
Un problema di decisione sull'alfabeto Σ è un linguaggio $L \subseteq \Sigma^*$

Un problema di decisione specifica quali stringhe (istanze I_{PD} del problema) appartengono a un determinato linguaggio L (soddisfano il predicato π).

Le istanze che fanno parte di L corrispondono a istanze positive Y_{PD}.
Le istanze che non fanno parte di L corrispondono a istanze negative N_{PD}.

Complessità di un Problema

Dato un problema P diciamo che:

- P ha complessità temporale delimitata superiormente da O(g(n)) se esiste un algoritmo A che risolve P tale che $t_A(n)=O(g(n))$;
- P ha complessità temporale delimitata inferiormente da $\Omega(g(n))$ se per ogni algoritmo A che risolve P si ha che $t_A(n)= \Omega(g(n))$;
- P ha complessità temporale $\Theta(g(n))$ se ha come upper bound O(g(n)) e come lower bound $\Omega(g(n))$.

Macchina di Turing Deterministica

Una macchina di Turing deterministica (MTD) è una sestupla $M = (\Gamma, b, Q, q_0, F, \delta)$ t.c.:

- Γ è l'alfabeto dei simboli del nastro;
- b è un carattere speciale denominato BLANK;
- Q è un insieme finito e non vuoto di stati;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali;
- δ è la fne. di transizione. δ: $(Q - F) \times (\Gamma \cup \{b\}) \rightarrow Q \times (\Gamma \cup \{b\}) \times \{d,s,i\}$, dove d, s e i indicano rispettivamente lo spostamento a destra, a sinistra e l'assenza di spostamento della testina.

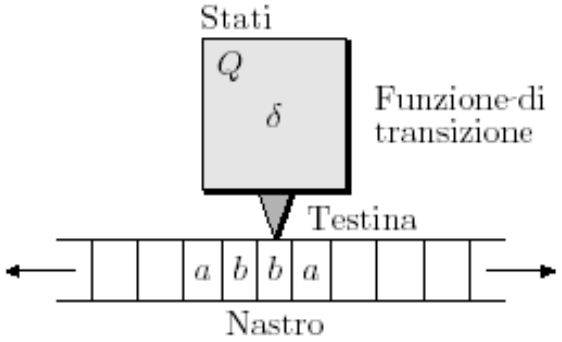


FIGURA 5.1 Macchina di Turing.

Macchina di Turing NON Deterministica

Una macchina di Turing non deterministica (MTND) è una sestupla $M=(\Gamma, b, Q, q_0, F, \delta_N)$ tale che:

- Γ è l'alfabeto dei simboli del nastro;
- b è un carattere speciale denominato BLANK;
- Q è un insieme finito e non vuoto di stati;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali;
- δ_N è la fne. di transizione δ_N: $Q \times (\Gamma \cup \{b\}) \rightarrow \{P(Q \times (\Gamma \cup \{b\}) \times \{d,s,i\})\}$ (con P insieme potenza), dove d, s e i indicano rispettivamente lo spostamento a destra, a sinistra e l'assenza di spostamento della testina.

In corrispondenza di uno stato q e di un simbolo letto possono esistere in generale più passi di computazione possibili (in numero finito).

Sequenza di computazione $\Rightarrow$ Albero di computazione

Il risultato della computazione è quindi l'insieme dei risultati ottenuti lungo tutti i rami dell'albero di computazione.

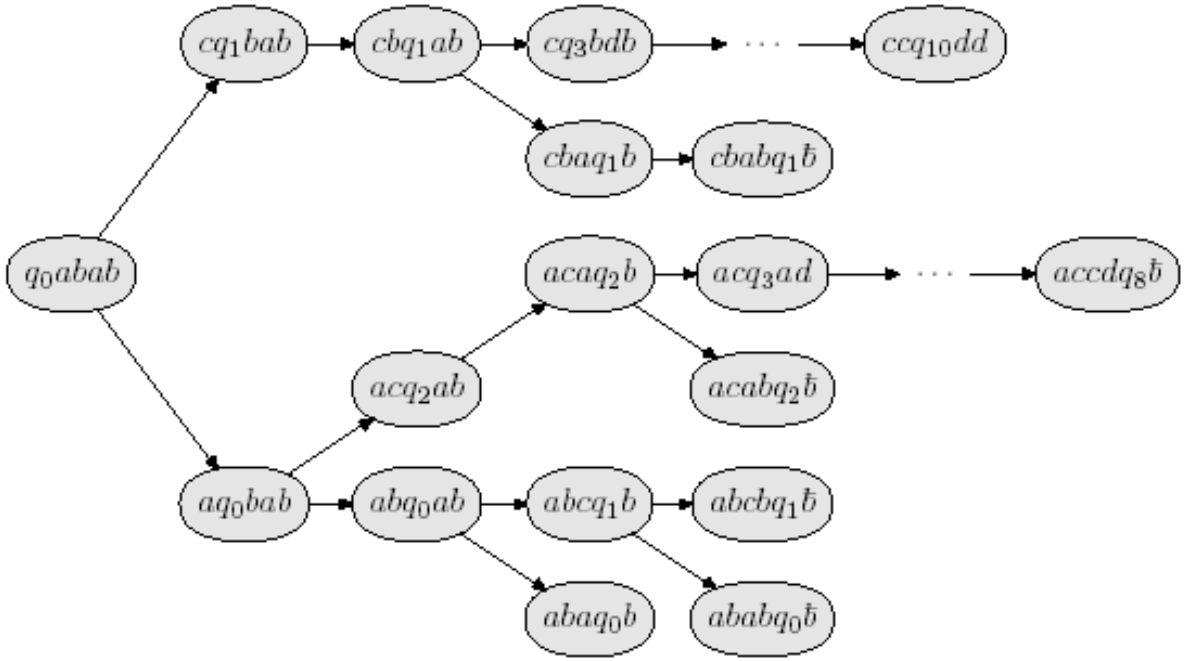
Esempio

Si consideri una macchina di Turing non deterministica con $\Gamma = \{a, b, c, d\}$, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$, $F = \{q_{10}\}$ e la funzione δ_N definita come segue:

	a	b	c	d	$\flat$
q_0	$\{(q_0, a, d), (q_1, c, d)\}$	$\{(q_0, b, d), (q_2, c, d)\}$	–	–	–
q_1	$\{(q_1, a, d), (q_3, d, s)\}$	$\{(q_1, b, d)\}$	–	–	–
q_2	$\{(q_2, a, d)\}$	$\{(q_2, b, d), (q_3, d, s)\}$	–	–	–
q_3	$\{(q_3, a, s)\}$	$\{(q_3, b, s)\}$	$\{(q_4, c, d)\}$	–	–
q_4	$\{(q_6, c, d)\}$	$\{(q_7, c, d)\}$	–	–	–
q_5	$\{(q_5, a, d)\}$	$\{(q_5, b, d)\}$	–	$\{(q_7, d, d)\}$	–
q_6	$\{(q_6, a, d)\}$	$\{(q_6, b, d)\}$	–	$\{(q_8, d, d)\}$	–
q_7	–	$\{(q_9, d, s)\}$	–	$\{(q_7, d, d)\}$	–
q_8	$\{(q_9, d, s)\}$	–	–	$\{(q_8, d, d)\}$	–
q_9	$\{(q_3, a, s)\}$	$\{(q_3, b, s)\}$	$\{(q_{10}, x, i)\}$	$\{(q_9, d, s)\}$	–
q_{10}	–	–	–	–	–

Esempio (cont.)

Comportamento della macchina di Turino su input “abab”.



NOTA: Si verifica un’accettazione dell’input se almeno uno dei cammini di computazione si arresta perché si è raggiunto uno stato finale. Si verifica un rifiuto altrimenti, cioè se tutti i cammini dell’albero di computazione terminano perché non è più possibile applicare δ_N senza avere però raggiunto uno stato finale.

Linguaggi: Accettazione (Rifiuto) Stringa

Dato un alfabeto Σ, una macchina di Turing (MTD deterministica) $M = (\Gamma, b, Q, q_0, F, \delta)$ ed un intero $\tau > 0$, una stringa $x \in \Sigma^*$ è accettata (rifiutata) da M in tempo τ se esiste una computazione $q_0x \vdash\!\!\!- c$ tale che la sua esecuzione impiega un tempo r con $r \leq \tau$ e c è una configurazione massimale di accettazione (di non accettazione).

Dato un alfabeto Σ, una macchina di Turing (MTD deterministica) $M = (\Gamma, b, Q, q_0, F, \delta)$ ed un intero $\sigma > 0$, una stringa $x \in \Sigma^*$ è accettata (rifiutata) da M in uno spazio σ se esiste una computazione $q_0x \vdash\!\!\!- c$ nel corso della quale si accede a v celle di nastro con $v \leq \sigma$ e c è una configurazione massimale di accettazione (di non accettazione).

Configurazione massimale:
Una configurazione c è massimale (di accettazione o di non) quando non è possibile applicare la funzione di transizione.

Accettazione di Linguaggi

Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) $M = (\Gamma, b, Q, q_0, F, \delta)$ ed una funzione t: N→N, diciamo che M accetta $L \subseteq \Sigma^*$ in tempo t(n) se, per ogni $x \in \Sigma^*$, M accetta x in tempo al più t(|x|) se $x \in L$, mentre, se $x \notin L$, allora M non lo accetta.

Dato un alfabeto Σ, una macchina di Turing (deterministica o non deterministica) $M = (\Gamma, b, Q, q_0, F, \delta)$ ed una funzione s: N→N, diciamo che M accetta $L \subseteq \Sigma^*$ in spazio s(n) se, per ogni $x \in \Sigma^*$, M accetta x in uno spazio al più s(|x|) se $x \in L$, mentre, se $x \notin L$, allora M non lo accetta.

Diciamo che un linguaggio L è accettabile in tempo deterministico (non deterministico) t(n) se esiste una macchina di Turing deterministica (non deterministica) che accetta L in tempo t(n).

Riconoscimento di Linguaggi

Dato un alfabeto Σ, una macchina di Turing deterministica $M = (\Gamma, b, Q, q_0, F, \delta)$ ed una funzione t: N→N, diciamo che M riconosce $L \subseteq \Sigma^*$ in tempo t(n) se, per ogni $x \in \Sigma^*$, M accetta x in tempo al più t(|x|) se $x \in L$, mentre se $x \notin L$ lo rifiuta (ancora in tempo al più t(|x|)).

Dato un alfabeto Σ, una macchina di Turing deterministica $M = (\Gamma, b, Q, q_0, F, \delta)$ ed una funzione s: N→N, diciamo che M riconosce $L \subseteq \Sigma^*$ in spazio s(n) se, per ogni $x \in \Sigma^*$, M accetta x in spazio s(|x|) se $x \in L$, mentre se $x \notin L$ lo rifiuta (ancora in spazio s(|x|)).

Diciamo che un linguaggio L è decidibile in tempo deterministico (non deterministico) t(n) se esiste una macchina di Turing deterministica (non deterministica) che riconosce L in tempo t(n).

Classi di Complessità

L'obiettivo della teoria della complessità è quello di catalogare i problemi in funzione della quantità di risorse di calcolo necessarie per risolverli.

Data una funzione f: N→N, indicata con n la dimensione dell'input definiamo le seguenti classi:

- DTIME(f(n)) è l'insieme dei linguaggi decisi da una macchina di Turing deterministica ad un nastro in tempo al più f(n);
- DSPACE(f(n)) è l'insieme dei linguaggi decisi da una macchina di Turing deterministica ad un nastro in spazio al più f(n);
- NTIME(f(n)) è l'insieme dei linguaggi accettati da una macchina di Turing NON deterministica ad un nastro in tempo al più f(n);
- NSPACE(f(n)) è l'insieme dei linguaggi accettati da una macchina di Turing NON deterministica ad un nastro in spazio al più f(n).

Classi di Complessità notevoli

In molti casi non si vuole caratterizzare la complessità di un problema in modo preciso. È sufficiente disporre di una classificazione più grossolana stabilendo ad esempio se il problema è risolubile in tempo proporzionale o esponenziale.
Definiamo le seguenti classi di complessità notevoli:

Problemi **P**:

$$P = \bigcup_{k=0}^{\infty} DTIME(n^k)$$

È la classe dei linguaggi decidibili da una macchina di Turing deterministica in tempo proporzionale ad un polinomio nella dimensione dell'input.

Problemi **NP**:

$$NP = \bigcup_{k=0}^{\infty} NTIME(n^k)$$

È la classe dei linguaggi accettati da una macchina di Turing NON deterministica in tempo proporzionale ad un polinomio nella dimensione dell'input.

Problemi **EXTTIME**:

$$EXPTIME = \bigcup_{k=0}^{\infty} DTIME(2^{n^k})$$

È la classe dei linguaggi decidibili da una macchina di Turing deterministica in tempo proporzionale ad un esponenziale nella dimensione dell'input.

Problemi **NEXPTIME**:

$$NEXPTIME = \bigcup_{k=0}^{\infty} NTIME(2^{n^k})$$

È la classe dei linguaggi accettati da una macchina di Turing NON deterministica in tempo proporzionale ad un esponenziale nella dimensione dell'input.

Classi di Complessità notevoli (cont.)

Problemi P-SPACE: $PSPACE = \bigcup_{k=0}^{\infty} DSPACE(n^k)$

È la classe dei linguaggi decidibili da una macchina di Turing deterministica in uno spazio proporzionale ad un polinomio nella dimensione dell'input.

Problemi NP-SPACE: $NPSPACE = \bigcup_{k=0}^{\infty} NSPACE(n^k)$

È la classe dei linguaggi accettati da una macchina di Turing NON deterministica in uno spazio proporzionale ad un polinomio nella dimensione dell'input.

Classi di Complessità notevoli (cont.)

Problemi **LOG-SPACE**:

$$LOGSPACE = \bigcup_{k=0}^{\infty} DSPACE(\log(n))$$

È la classe dei linguaggi decidibili da una macchina di Turing deterministica in uno spazio proporzionale al logaritmo nella dimensione dell'input.

Problemi **NLOG-SPACE**:

$$LOGSPACE = \bigcup_{k=0}^{\infty} NSPACE(\log(n))$$

È la classe dei linguaggi accettati da una macchina di Turing NON deterministica in uno spazio proporzionale al logaritmo nella dimensione dell'input.

Classe P

TESI DI CHURCH

I problemi di decisione trattabili sono quelli risolubili in tempo polinomiale (nella lunghezza dell'input) da una MTD.

$\Rightarrow$ La classe P è la classe dei problemi trattabili!

I problemi appartenenti a tale classe sono risolvibili tramite un algoritmo polinomiale e quindi sono risolvibili in modo efficiente.

Classe NP - definizione basata su **certificati**

Una macchina di Turing *M* con un nastro addizionale di input (nastro dei certificati) a senso unico verifica il linguaggio L se:

- qualunque sia il contenuto del nastro dei certificati, se $w \notin L$ allora *M* non accetta w;
- per ogni $w \in L$ esiste una stringa $w' \in \Sigma^*$ (detta certificato o prova di appartenenza di w) tale che *M* accetta w se trova w' sul nastro dei certificati.

La classe NTIME(*f(n)*) è la classe dei linguaggi verificabili da una macchina di Turing in un tempo pari al più a *f(n)* passi $\Leftrightarrow \forall$ w di lunghezza n, $\exists$ un certificato w' tale che dati w e w' come input alla macchina di Turing, la macchina termina al più in f(n) passi.

I problemi NP sono linguaggi per le cui stringhe è possibile produrre un certificato rapidamente verificabile.

Classe NP (cont.)

Intuitivamente la classe P è costituita dai problemi che possono essere risolti efficientemente, mentre la classe NP è costituita dai problemi per i quali una soluzione può essere verificata efficientemente.

Verificatore emula MTND

La macchina di Turing non deterministica è una macchina di Turing che a ogni passo di esecuzione può compiere più transizioni successive.
Il certificato è la descrizione di uno dei possibili cammini nell'albero di computazione della MTND. La macchina di Turing deterministica controlla, dato il cammino nell'albero, se esso conduce in effetti a una configurazione accettante.

MTND emula un verificatore

Ogni volta che il verificatore legge un simbolo dal nastro dei certificati la MTND biforca la computazione in $|\Sigma|$ rami.
Esiste un cammino accettante se e solo se esiste un certificato.

Classe NP (cont.)

I problemi NP sono quelli che sono risolvibili da un algoritmo di tipo polinomiale non deterministico.

ALGORITMO NP:
Guessing stage) Un “oracolo” indovina la soluzione (CERTIFICATO)
Checking stage) Verifica in tempo polinomiale della soluzione

$\Rightarrow$ CLASSE NP:
Appartengono alla classe NP tutti i problemi di decisione per cui, essendo nota la soluzione, è possibile verificare in tempo polinomiale che la risposta è affermativa (Non deterministic Polynomial Time Algorithm)

Un tipico problema NP è un problema in cui è difficile determinare l’esistenza della soluzione ma è facile verificare se una soluzione candidata è effettivamente tale.

Classe NP (cont.)

Esempio

Problema
Dato il grafo G, esiste un circuito Hamiltoniano di lunghezza inferiore a k?

Risposta
Sì

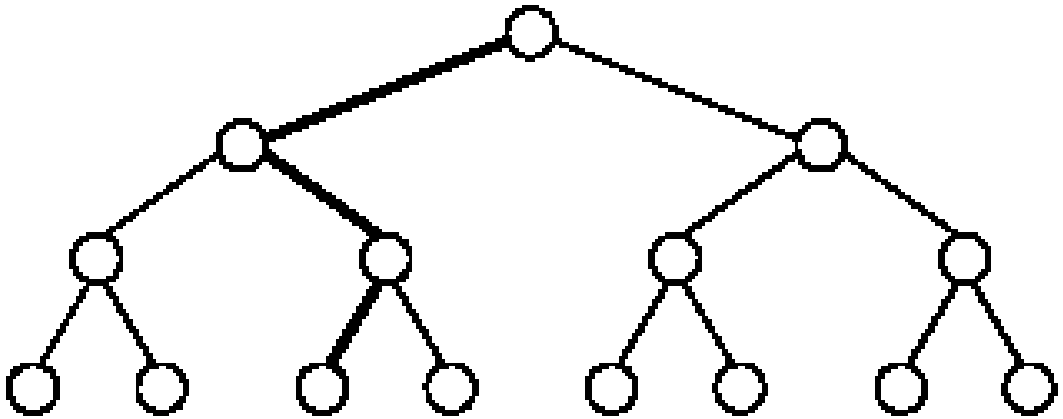
Certificato di Ammissibilità
Una sequenza di nodi del grafo.

Verifica
Seguire la sequenza di nodi verificando che corrisponde ad un circuito Hamiltoniano di lunghezza inferiore a k (verificabile in tempo polinomiale).

Il problema CICLO HAMILTONIANO è NP!

Classe NP (cont.)

Un problema è NP
⇔ dato un certificato è verificabile polinomialmente;
⇔ è risolubile con un albero decisionale, dove il più lungo percorso dalla radice alle foglie è visitabile polinomialmente;



⇔ è risolubile in tempo polinomiale da un calcolatore non deterministico.

Gerarchia

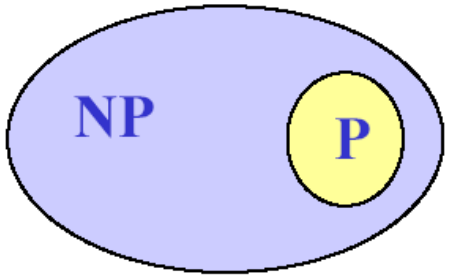
$P \subset EXPTIME$
$NP \subset NEXPTIME$

Teorema:
Per ogni funzione f: N→N valgono le seguenti proprietà:

- $DTIME(f(n)) \subseteq NTIME(f(n))$;
- $DSPACE(f(n)) \subseteq NSPACE(f(n))$.

Le relazioni sono dimostrate semplicemente osservando che ogni macchina di Turing deterministica è un caso particolare di macchina di Turing NON deterministica.

$P \subseteq NP$



Gerarchia (cont.)

LOGSPACE $\subset$ PSPACE

LOGSPACE $\subset$ NLOGSPACE

Teorema:
Per ogni funzione f: N$\rightarrow$N valgono le seguenti proprietà:

- DTIME(f(n)) $\subseteq$ DSPACE(f(n));
- NTIME(f(n)) $\subseteq$ NSPACE(f(n)).

Le relazioni sono dimostrate semplicemente osservando che in t passi di computazione una macchina di Turing utilizza al più t celle di nastro distinte.

Gerarchia (cont.)

$$P \subseteq PSPACE$$
$$NP \subseteq NPSPACE$$

$$P = NP \ ?$$

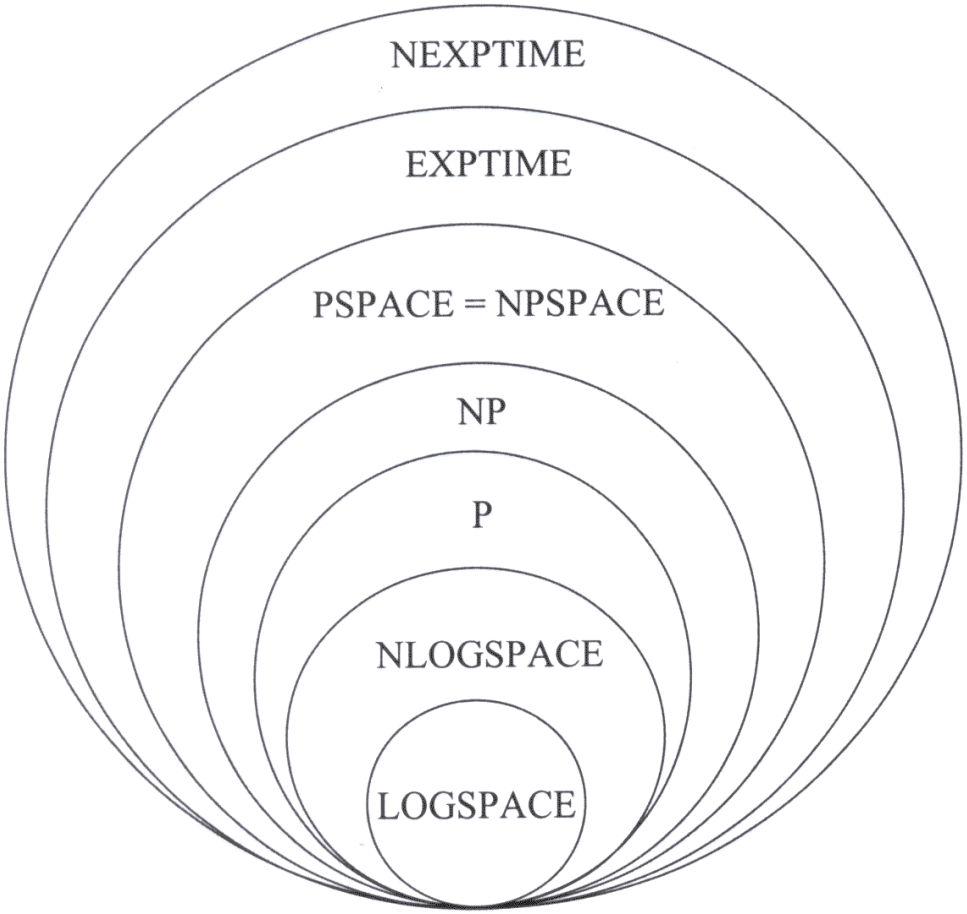
Molti matematici pensano che $P \neq NP$ ma nessuno finora è riuscito a dimostrarlo.

Dal teorema di Savitch:

$$PSPACE = NSPACE$$

$$P \subseteq NP \subseteq PSPACE = NPSPACE$$

Gerarchia (cont.)

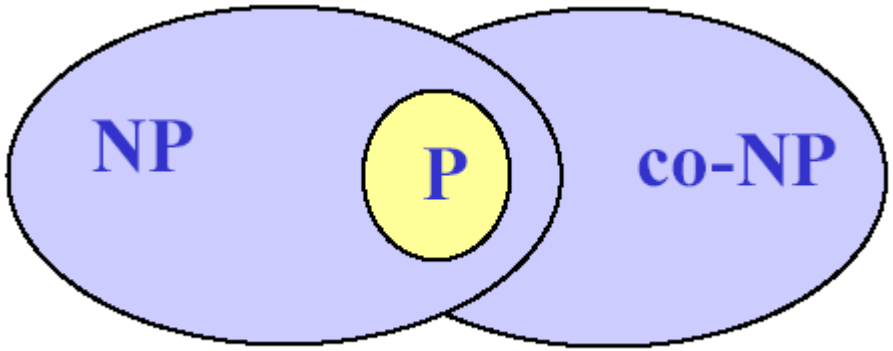


Problema complementare

I **problemi complementari** sono quelli per cui viene formulata una domanda in termini negati.

I problemi complementari dei problemi in P sono ancora in P $\Rightarrow$ **P = co-P**

I problemi complementari dei problemi in NP non sono in NP $\Rightarrow$ **NP ≠ co-NP**



$P \subseteq NP$, co-$P \subseteq$ co-NP, P=co-P, NP ≠ co-NP $\Rightarrow P \subseteq NP \cap$ co-NP

Problema complementare (cont.)

Esempio: NP≠co-NP
Problema del ciclo Hamiltoniano:
Dato il grafo G, esiste un ciclo hamiltoniano in G di lunghezza inferiore a k?

Problema del ciclo Hamiltoniano complementare:
Dato un grafo Hamiltoniano, è vero che in esso non esistono cicli di lunghezza inferiore a k?

CERTIFICATO di AMMISSIBILITA': la lista di tutti i possibili cicli Hamiltoniani.

Per un grafo completo di m nodi il certificato conterrebbe m! cicli ⇒ esponenziale
Quindi non verificabile in tempo polinomiale!!!

Esempio – Classi di Complessità

Problema del grafo BIPARTITO

Dato un grafo G connesso, si vuole sapere se G è bipartito.
Assumendo che la grandezza dell'input sia il numero di vertici di G, dimostrare che:

- il problema "bipartito" appartiene alla classe NP;
- il problema "bipartito" appartiene alla classe P.

Def. di **Grafo Bipartito**

Un grafo G=(V,E) è bipartito se esiste una partizione {A, B} di V tale che due nodi nello stesso insieme (A o B) non sono mai adiacenti (cioè non esiste un arco che li collega).

Esempio (cont.)

Dimostriamo che “Bipartito” appartiene alla classe NP

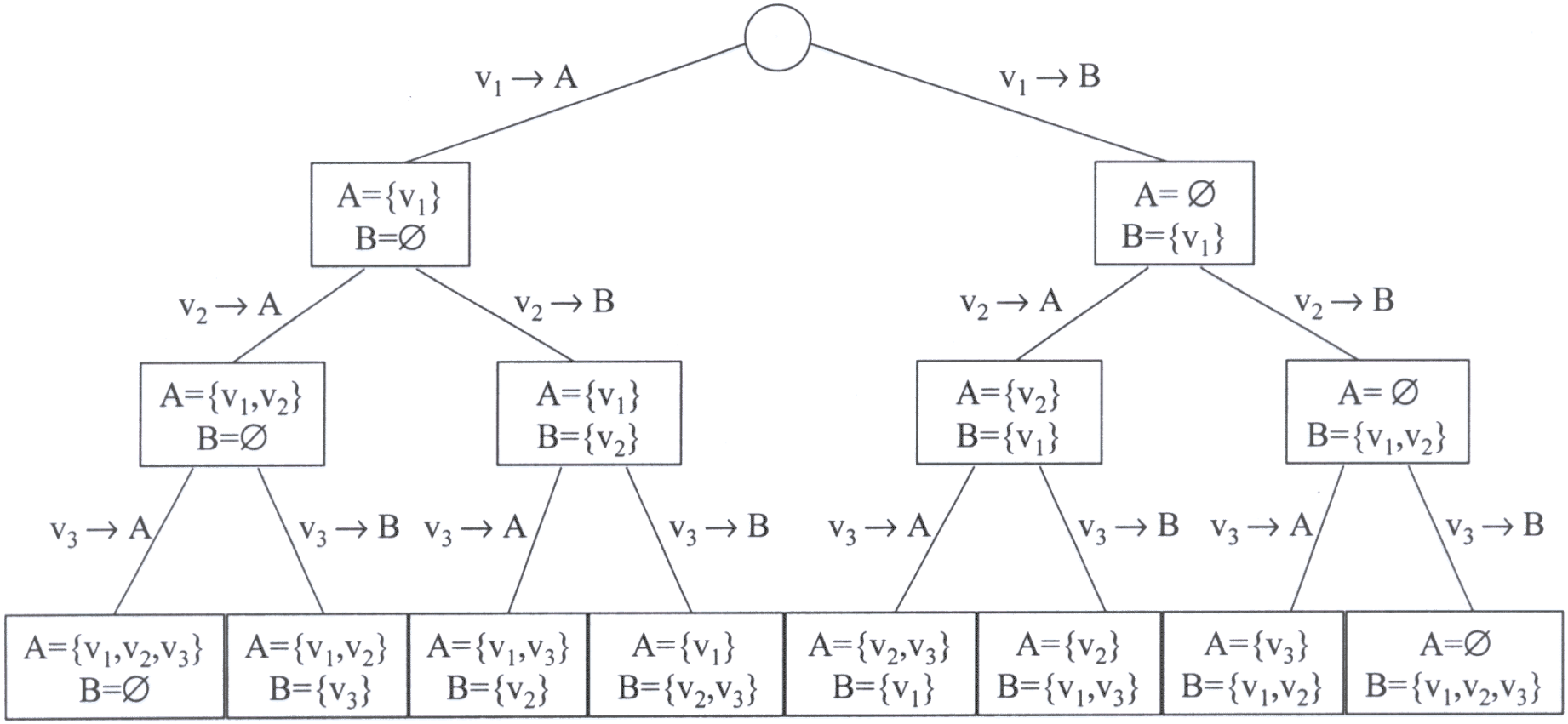
$\Rightarrow$ Descriviamo un algoritmo non deterministico che in tempo polinomiale decide se G è bipartito.

INPUT: G = (V, E) connesso;

OUTPUT: {V, F}.

1. ordina arbitrariamente i nodi di V = $\{v_1, v_2, \ldots, v_n\}$ e poni A=B=$\varnothing$
2. while (tutti i nodi non sono stati esaminati)
3. per ogni passo i-esimo, su ogni ramo della computazione esegui non deterministicamente le seguenti due operazioni:
   - metti v_i in A
   - metti v_i in B ▷ la computazione si divide in due rami
4. per ogni ramo verifica che in A ed in B non ci siano due nodi adiacenti;
   - se esiste un ramo che verifica la proprietà ritorna vero
   - altrimenti ritorna falso

Esempio (cont.)



albero di computazione per G con tre vertici

Esempio (cont.)

Osservazione: ogni ramo della computazione di “Bipartito” può essere eseguito contemporaneamente agli altri!

Analisi della complessità:
Su ogni ramo della computazione l’algoritmo descritto esegue n passi distinti, in ognuno dei quali esegue l’inserimento di un nodo in uno dei due insiemi A e B.
$\Rightarrow O(n)$

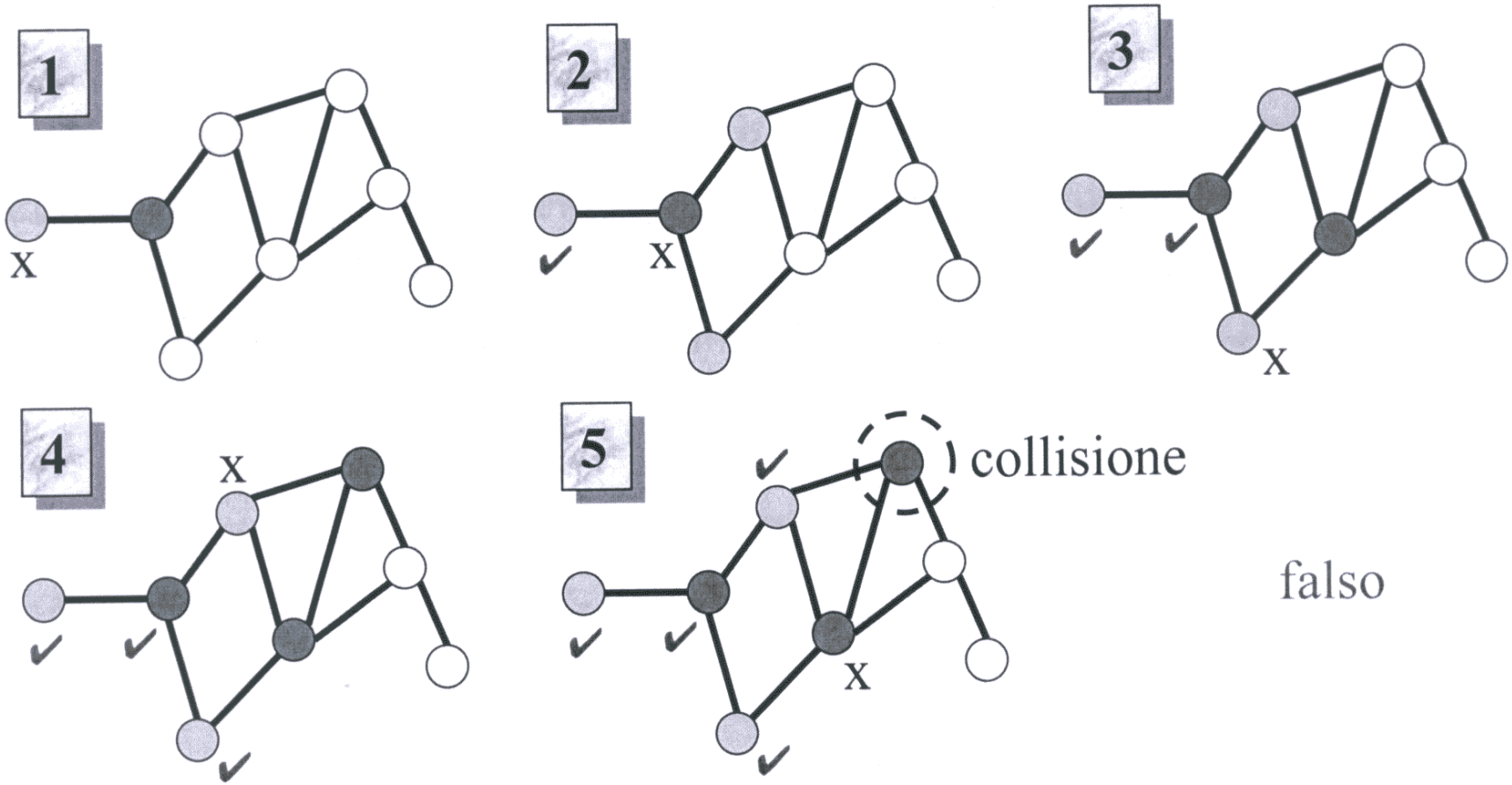
Alla fine degli n passi esegue una verifica di correttezza della bipartizione costruita confrontando un numero $O(n^2)$ di coppie di nodi.
$\Rightarrow O(n^2)$

Esempio (cont.)

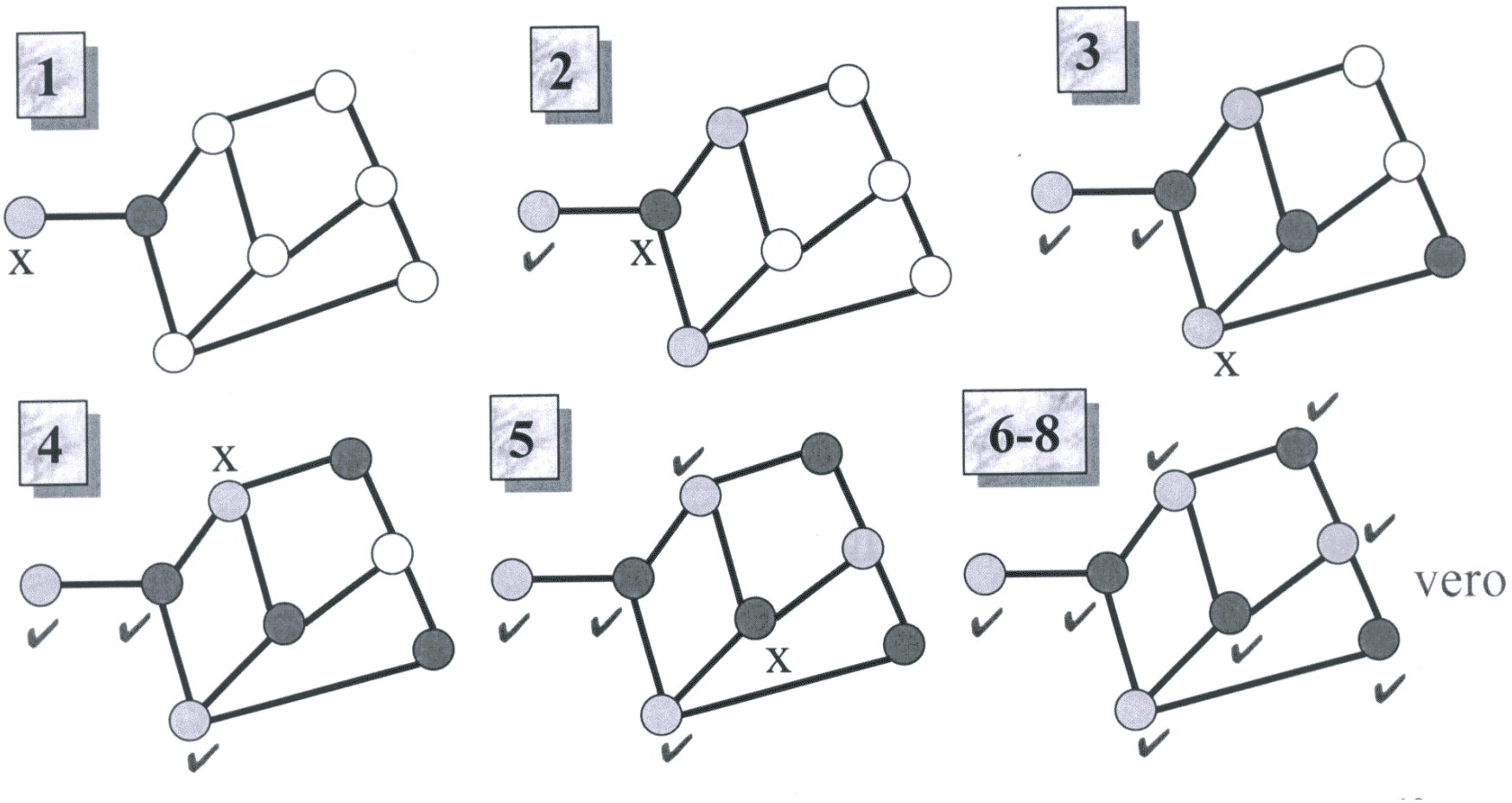
Dimostriamo che “Bipartito” appartiene alla classe P: si descrive un algoritmo deterministico che in tempo polinomiale decide se G è bipartito.

1. poni A=B=∅ e marca tutti i nodi come non esaminati
2. scegli un nodo qualunque x e mettilo in A
3. metti tutti i nodi adiacenti a x in B
4. marca x come esaminato
5. while(tutti i nodi sono esaminati) ———— O(n)
6. do scegli un nodo x non ancora esaminato tra quelli in A o in B
7. se x sta in A
8. controlla tutti i nodi adiacenti a x ———— O(n)
9. se tra questi ce ne è uno già inserito in A allora ritorna(falso)
10. altrimenti inserisci tutti gli adiacenti in B
11. marca x come esaminato
12. se x sta in B
13. controlla tutti i nodi adiacenti a x ———— O(n)
14. se tra questi ce ne è uno già inserito in B allora ritorna(falso)
15. altrimenti inserisci tutti gli adiacenti in A
16. marca x come esaminato
17. se x non esiste
18. ritorna vero

Esempio (cont.)



Esempio (cont.)



Riducibilità

- Introdotta da Cook negli anni ’70;
- per la trasformazione polinomiale tra problemi.

Una riduzione polinomiale da un problema di decisione $A \subseteq \Sigma^*$ ad un problema di decisione $B \subseteq \Theta^*$ è una funzione $f: \Sigma^* \rightarrow \Theta^*$ tale che:
- f è computabile in tempo polinomiale su una macchina di Turing deterministica
- $\forall x \in \Sigma^*$, $x \in A$ se e solo se $f(x) \in B$.

Si dice che un problema A è karp-riducibile ($A \leq_{m,p} B$) ad un problema B in tempo polinomiale se esiste una riduzione polinomiale da A a B.

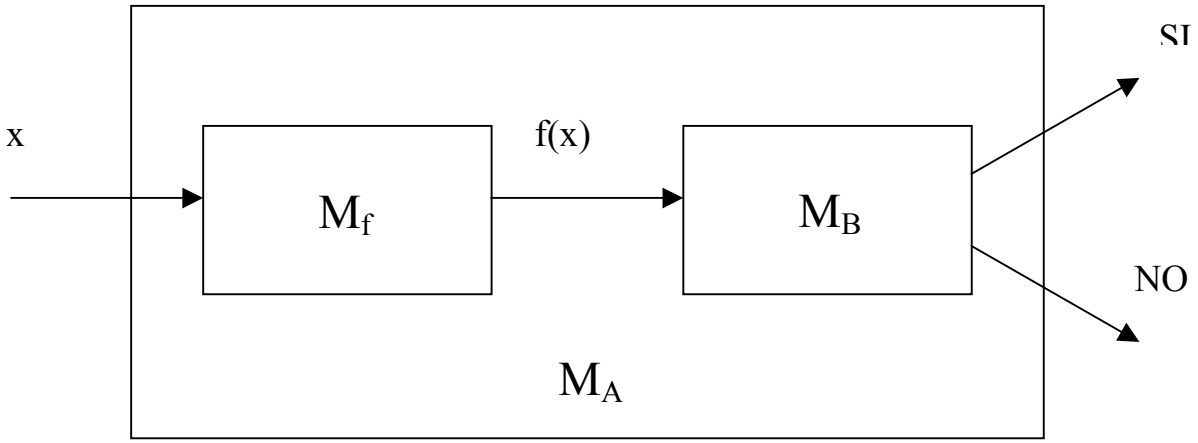
In altre parole, la karp-riducibilità mappa un’istanza x di A in un’istanza y di B in modo tale che $x \in Y_A$ se e solo se $f(x) \in Y_B$.

Riducibilità (cont.)

La karp-riducibilità $A \leq_{m,p} B$ dà l'idea che B sia più difficile di A.

Teorema:
Se $B \in P$ e $A \leq_{m,p} B$, allora $A \in P$.

Si dimostra semplicemente osservando che, avendo un algoritmo polinomiale per risolvere B, è possibile risolvere in tempo polinomiale A, applicando f all'input e risolvendo B.



Riducibilità (cont.)

Una riduzione polinomiale f NON è invertibile!

Teorema:
Se $A \leq_{m,p} B$, allora $A \in P$ non implica $B \in P$.

Teorema:
Se $A \leq_{m,p} B$ e $B \leq_{m,p} C$, allora $A \leq_{m,p} C$.

Problemi Polinomialmente equivalenti

Due problemi A e B sono polinomialmente equivalenti se esistono due funzioni f e g tali che:

- f è una riduzione polinomiale da A a B;
- g è una riduzione polinomiale da B a A.

Se A e B sono Polinomialmente equivalenti, $A \in P$ se e solo se $B \in P$.

Problema SAT

Problema SAT:
Istanza: una formula booleana congiunta nelle variabili $x_1, \ldots, x_n$
(es. $F= (x1 \vee x2) \wedge (x3 \vee x4 \vee x5) \wedge \ldots \wedge (x_{n-1} \vee x_n)$
Domanda: esiste un assegnamento di valori alle variabili che soddisfa F?

Casi particolari:

2-SAT
Il problema 2-SAT è un caso particolare di problema SAT in cui ogni clausola ha esattamente due letterali.

3-SAT
Il problema 3-SAT è un caso particolare di problema SAT in cui ogni clausola ha esattamente tre letterali.

Esempio Riducibilità

Dimostrare che il problema 2-SAT è P.

Problema CAMMINO: consiste nel decidere se in un grafo orientato G=(V,E) esiste un cammino tra due nodi fissati.
Questo problema è risolvibile in tempo polinomiale tramite una visita in ampiezza del grafo.

Problema CAMMINO-L:
Dato un insieme O(|V|) di coppie di nodi in G, esiste un cammino tra i nodi di ciascuna coppia?
CAMMINO-L è un insieme di O(|V|) problemi di tipo CAMMINO e quindi è ancora risolvibile in tempo polinomiale.

Esempio Riducibilità (cont.)

Le clausole della formula di un problema 2-SAT sono formate da due soli letterali, cioè sono del tipo $(\alpha \vee \beta)$.
Osserviamo che $(\alpha \vee \beta)$ può essere riscritto come: $(\neg\alpha \Rightarrow \beta) \wedge (\neg\beta \Rightarrow \alpha)$

Quindi è possibile riscrivere una formula 2-SAT come un insieme di implicazioni

Esempio
$F = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$
Si riscrive nel seguente modo:
$F = (\neg x_1 \Rightarrow x_2) \wedge (\neg x_2 \Rightarrow x_1) \wedge (x_1 \Rightarrow x_2) \wedge (\neg x_2 \Rightarrow \neg x_1)$

Si dimostra che una formula 2-SAT è NON soddisfacibile sse esiste una variabile x tale che "x implica ¬x" e "¬x implica x".

Esempio Riducibilità (cont.)

Esempio

$F = \underbrace{(\neg x_1 \vee x_2)}_{1} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{2} \wedge \underbrace{(\neg x_1 \vee x_3)}_{3} \wedge \underbrace{(x_1 \vee x_2)}_{4} \wedge \underbrace{(x_1 \vee \neg x_2)}_{5}$

è non soddisfacibile perché:

$(x_1 \overset{1}{\Rightarrow} x_2 \overset{2}{\Rightarrow} \neg x_3 \overset{3}{\Rightarrow} \neg x_1)$ e $(\neg x_1 \overset{1}{\Rightarrow} x_2 \overset{5}{\Rightarrow} x_1)$

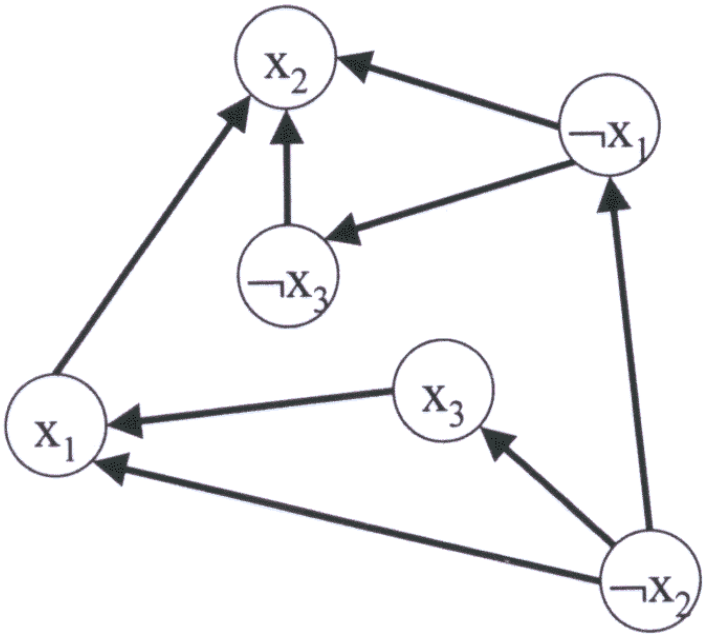
Riduzione 2-SAT$\leq_{m,p}$CAMMINO-L è definita nel seguente modo:

- il grafo G del problema CAMMINO-L ha per nodi tutti i letterali del problema 2-SAT, cioè per ogni variabile x ci sono i nodi x e ¬x;
- per ogni clausola del tipo $(\alpha \vee \beta)$ inseriamo gli archi tra i nodi $\neg\alpha$ e β e tra i nodi $\neg\beta$ e α
- se esiste un cammino per ogni coppia di nodi $(x_i, \neg x_i)$ e $(\neg x_i, x_i)$ allora il problema 2-SAT è insoddisfacibile

Esempio Riducibilità (cont.)

Esempio di Riduzione:

$F = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3)$



Non esiste un cammino tra x_1 e $\neg x_1$, quindi la formula è soddisfacibile.
Ad es. F è soddisfatta con x_1=VERO, x_2=VERO, x_3=FALSO.

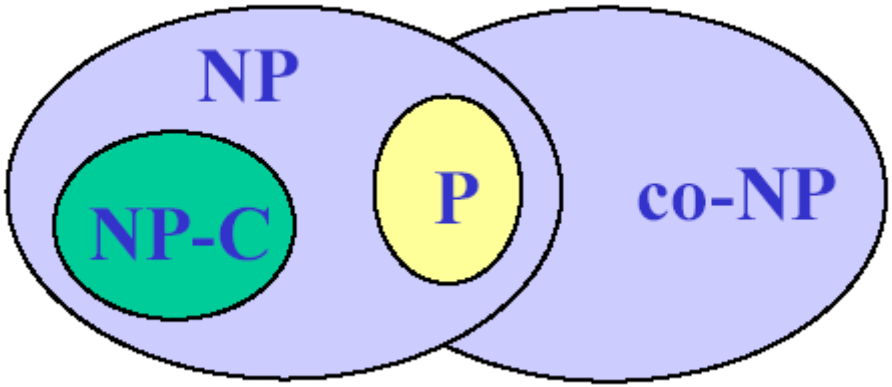
Classe NP-C

Si definiscono i problemi più difficili di una classe quei problemi appartenenti alla classe che risultano almeno difficili quanto gli altri.

Un problema A è NP-Completo se:

1. A $\in$ NP;
2. per ogni problema B $\in$ NP esiste una riduzione polinomiale da B ad A.

Definiamo NP-C come la classe dei problemi NP-completi.



Classe NP-C (cont.)

Perché la classe NP-C è così importante?
L'individuazione di un algoritmo polinomiale per un problema NP-completo consentirebbe di risolvere ogni problema NP in tempo polinomiale.
$\Rightarrow$ tutti i problemi considerati tutt'oggi intrattabili potrebbero essere risolti efficientemente.

Teorema:
Se esiste un problema NP-completo C tale che $C \in P$ allora P = NP.

Classe NP-C (cont.)

Teorema:
Se C è un problema NP-completo, A è in NP e $C \leq_{m,p} A$ allora A è NP-completo.

Si dimostra prendendo un generico problema $B \in NP$. Componendo la riduzione che mappa B in C (tale riduzione esiste per ipotesi perché C è NP-C) con la riduzione che mappa C in A (che esiste per ipotesi) otteniamo che un qualunque problema $B \in NP$ è karp-riducibile ad A.

Se si dimostra che almeno un problema è NP-completo è possibile utilizzare le riduzioni per classificare altri problemi come NP-C.

Teorema di Cook:
$SAT \in NP\text{-}C$.

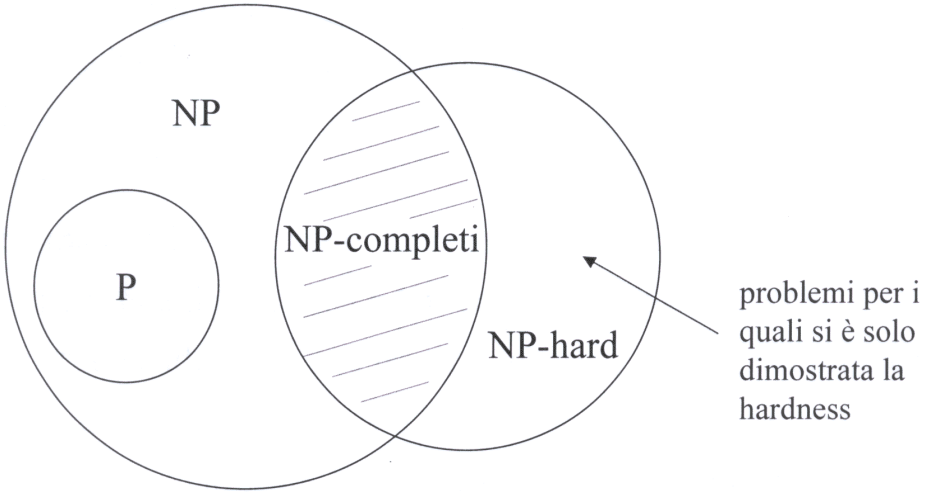
Classe NP-Hard

Un problema A è NP-Hard se:

- per ogni problema $B \in NP$, $B \leq_{m,p} A$.

Gli NP-Hard sono problemi per cui si è dimostrato il punto 2 (proprietà Hardness) della def. di NP-completezza, ma non il punto 1 (proprietà di Membership).

Osservazioni:
I problemi NP-hard possono essere problemi anche NON-NP.
I problemi NP-Hard sono difficili almeno quanto ogni problema NP-C.



Esempio NP-Completo

Dimostrare che il problema 3-SAT è NP-completo.

Il problema è NP perché è un caso particolare di SAT.
Per dimostrare che è NP-completo si sfrutta la riducibilità ($SAT \leq_{m,p}$ 3-SAT).

Consideriamo una generica clausola C del problema SAT e costruiamo a partire da essa una istanza C' del problema 3-SAT.

Per ogni clausola C di F distinguiamo tre possibilità:

- se C ha esattamente 3 letterali $\Rightarrow$ aggiungiamo C ad F';
- se C ha meno di tre letterali $\Rightarrow$ si ottiene C' ripetendo un qualunque letterale di C tante volte quanto basta affinché C' abbia tre letterali.
  Es. $C=(x_1 \vee x_2) \Rightarrow C'=(x_1 \vee x_2 \vee x_2)$
- se C ha più di 3 letterali; $C=(\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n) \Rightarrow$ aggiungiamo ad F l'insieme di clausole: $C'=(\alpha_1 \vee \alpha_2 \vee z_1) \wedge (\neg z_1 \vee \alpha_3 \vee z_2) \wedge \ldots \wedge (\neg z_{n-3} \vee \alpha_{n-1} \vee \alpha_n)$

Esempio NP-Completo (cont.)

Verifichiamo che C è soddisfatta $\Leftrightarrow$ C' è soddisfatta.

- se C è vera $\Rightarrow$ esiste almeno un letterale α_i vero; allora per fare in modo che C' sia vera basta assegnare falso ai nuovi letterali di C' che stanno nella stessa clausola di a'. Poi si propaga l'assegnamento di valori ai nuovi letterali facendo in modo che in ogni clausola ce ne sia almeno uno vero
  esempio:
  $C' = (\alpha_1 \vee \alpha_2 \vee z_1) \wedge (\neg z_1 \vee \alpha_3 \vee z_2) \wedge (\neg z_2 \vee \alpha_4 \vee \alpha_5)$
  V F V F V

- se C è falsa $\Rightarrow$ ogni letterale α_i è falso. Quindi per fare in modo che C' sia vera occorre assegnare z_1=vero; ciò implica $\neg z_1$=falso e quindi occorre assegnare z_2=vero; iterando il ragionamento risulta che z_{n-3} è falso e quindi l'ultima clausola di C' non è VERA $\Rightarrow$ C' è falsa!