Problemi

- Non computabili (es. problema della fermata di Turing): non possono essere risolti da alcun calcolatore, indipendentemente dal tempo disponibile
- Computabili
  - Trattabili = risolvibili in tempo polinomiale (cioè in tempo $O(n^k)$ per qualche costante k) = classe P
  - Intrattabili = risolvibili in tempo superpolinomiale = (classe NP – classe P) e altro

classe NP – classe P: tempo al più esponenziale
altro: tempo superesponenziale

Problemi NP-completi = classe NPC

Classe contenuta in NP per la quale

- Non è stato trovato alcun algoritmo risolvente che operi in tempo polinomiale
- Non è stato fornito alcun limite inferiore con tempo superpolinomiale degli algoritmi risolventi

Tema di ricerca dell'informatica teorica formulato nel 1971: $P \neq NP$

Convinzione: i problemi NP-completi sono intrattabili
Motivazione: se un qualsiasi singolo problema NP-completo potesse essere risolto in tempo polinomiale, allora ogni problema NP-completo avrebbe un algoritmo risolvente in tempo polinomiale (equivalenza di tutti i problemi NP-completi), mentre sinora nessun algoritmo cosiffatto è stato trovato

Teoria della NP-completezza

- Permette di stabilire se un problema dato è NP-completo
- Sfrutta le correlazioni tra problemi: noto (dalla letteratura) un problema NP-completo (ad es. soddisfattibilità di circuiti), la metodologia della riduzione può dimostrare che altri problemi lo sono
- Limita lo studio ai problemi di decisione (perché consentono di sfruttare la teoria dei linguaggi formali)

Se un problema è NP-completo, si ha quasi la certezza della sua intrattabilità $\rightarrow$ sviluppo di un algoritmo approssimato (ricerca di una soluzione *quasi* ottima) anziché di uno esatto o utilizzo di altre tecniche (ad es. euristiche, randomizzate, semplice restrizione dello spazio di ricerca attribuendo valori fissi a certi parametri del problema così da cercare le soluzioni di casi trattabili, ecc.)

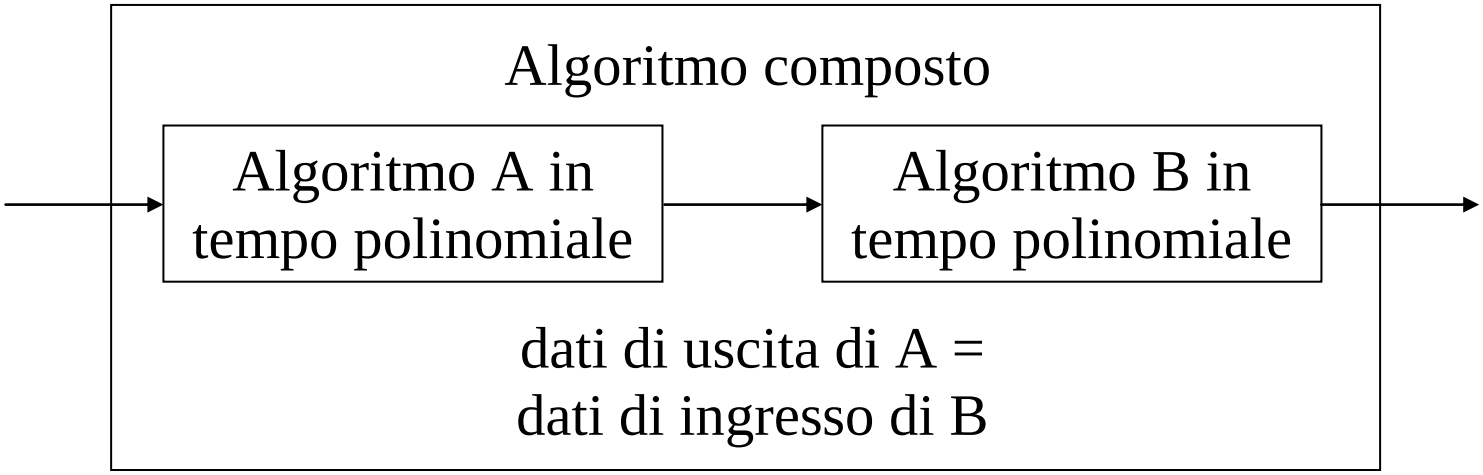
Problemi in tempo polinomiale = classe P

Formano una classe

- indipendente dal modello computazionale: un problema risolvibile in tempo polinomiale con un certo modello computazionale (ad es. macchina sequenziale RAM), è risolvibile in tempo polinomiale anche con un altro modello (ad es. calcolatore parallelo, anche se il numero di processori adottati crescesse in modo polinomiale in relazione alla dimensione dell'input)
- che gode di interessanti proprietà di chiusura, che derivano dalle proprietà di chiusura dei polinomi rispetto alle operazioni di addizione, moltiplicazione e composizione

Algoritmi in tempo polinomiale: chiusura

La composizione in pipeline di due algoritmi in tempo polinomiale è un algoritmo in tempo polinomiale



La composizione di un algoritmo in tempo polinomiale attraverso la chiamata di un numero costante di procedure in tempo polinomiale è ancora una algoritmo in tempo polinomiale

Un algoritmo che, oltre a compiere un numero polinomiale (dipendente dalle dimensioni dell'input) di passi di computazione, effettua un numero polinomiale di chiamate di un algoritmo in tempo polinomiale, opera globalmente in tempo polinomiale

Problemi astratti

Problema astratto Q = relazione fra l'insieme I delle istanze e l'insieme S delle soluzioni
N.B. a una istanza possono corrispondere più soluzioni (vedi esempio qui sotto)

Esempio: Problema CAMMINO-MINIMO: trovare il cammino più corto tra due vertici u e v di un grafo non orientato G = (V, E)

Istanza i del problema: <G, u, v>, dove u $\in$ V, v $\in$ V
Soluzione CAMMINO-MINIMO(i): sequenza di vertici di G (la sequenza vuota rappresenta il fatto che fra u e v non esiste alcun cammino)

Problemi di decisione

Problemi che hanno come insieme S delle soluzioni {0,1} (oppure {sì, no} oppure {vero, falso})

Esempio. Problema di decisione CAMMINO: stabilire se esiste fra due vertici u e v di un grafo non orientato G = (V, E) un cammino la cui lunghezza sia al più l'intero non negativo k

Istanza i del problema: <G, u, v, k>, dove u $\in$ V, v $\in$ V, k $\in N^+$
Soluzione CAMMINO(i): 1 (sì) se $\exists$ un cammino non nullo fra u e v che ha una lunghezza $\leq$ k, 0 (no) altrimenti

Problemi di ottimizzazione

CAMMINO-MINIMO è un problema di ottimizzazione perché chiede di minimizzare un valore.
Altri problemi di ottimizzazione possono chiedere di massimizzare un valore

La teoria della NP-completezza richiede che i problemi di ottimizzazione siano riformulati come problemi di decisione, imponendo una limitazione sul valore da ottimizzare

Il problema di ottimizzazione CAMMINO-MINIMO è stato riformulato nel problema di decisione CAMMINO, imponendo una limitazione sul valore da ottimizzare k

Riformulazione di problemi di ottimizzazione in problemi di decisione

Dato un metodo per risolvere il problema di ottimizzazione, automaticamente si risolve anche il problema di decisione (per ogni k)

Esempio. $\forall i \in I$ di CAMMINO-MINIMO, i = <G, u, v>,
se |CAMMINO-MINIMO(i)| = h ($h \geq 0$), allora

- CAMMINO(i') = 1 se $h \leq k$ e $h \neq 0$,
- CAMMINO(i') = 0 se $h > k$ o $h = 0$,

dove i' = <G, u, v, k>

Riformulazione di problemi di ottimizzazione in problemi di decisione (cont.)

Se un problema di ottimizzazione O può essere risolto velocemente, allora si può risolvere velocemente anche il problema di decisione D corrispondente, confrontando il valore della soluzione di O con il limite fornito in ingresso a D; D è arduo se anche O lo è

$$\text{difficoltà } D \leq \text{difficoltà } O$$

Problema concreto

Problema il cui insieme I di istanze è codificato in un insieme di stringhe su un alfabeto finito contenente almeno due simboli (ogni algoritmo eseguito su un calcolatore richiede la codifica binaria delle istanze)

Una istanza composta è codificata combinando le rappresentazioni delle sue componenti

e(Q) = problema concreto corrispondente al problema astratto Q secondo la codifica e: $I \rightarrow \{0,1\}^*$

L'insieme delle soluzioni S={0,1} di un problema di decisione concreto è lo stesso del problema astratto corrispondente, quindi Q(i) è la soluzione sia di i sia di e(i)

Difficoltà: se si assume $\{0,1\}^*$ come insieme delle istanze di e(Q), $\exists$ stringhe binarie che non corrispondono ad alcuna istanza di Q $\rightarrow$ ipotizziamo che per ciascuna di tali stringhe il valore d'uscita del problema concreto sia 0

È risolvibile in tempo polinomiale se esiste un algoritmo risolvente in tempo $O(n^k)$ per qualche costante k, dove n è la lunghezza della stringa in ingresso

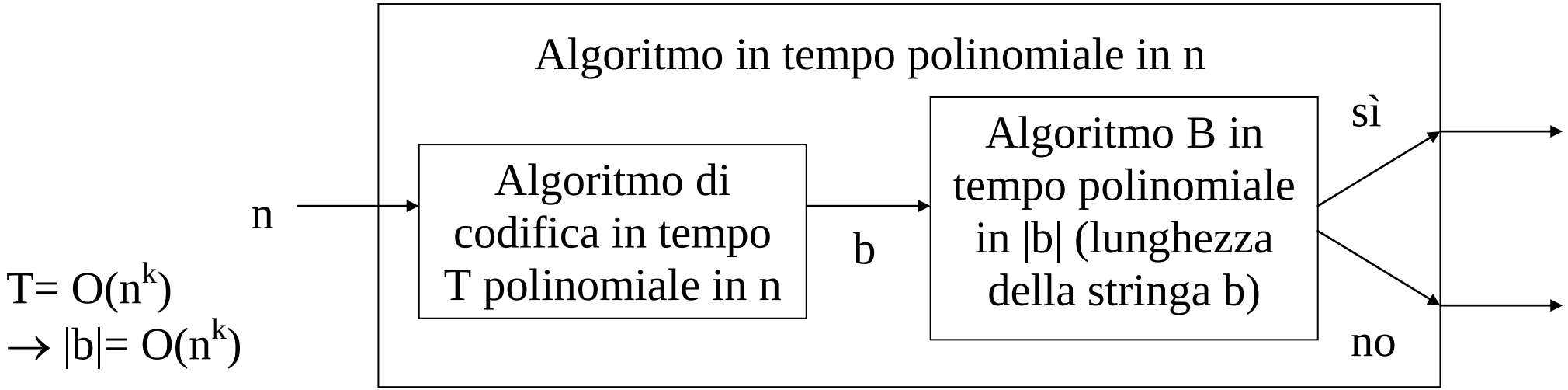
Classe di complessità P

Insieme dei problemi di decisione concreti che sono risolvibili in tempo polinomiale

Codifica

Si desidera estendere la definizione di problema risolvibile in tempo polinomiale dal problema concreto a quello astratto, in modo indipendente dalla codifica (cioè si considerano codifiche tali per cui l'efficienza della risoluzione di un problema non dipende da come il problema è codificato) $\rightarrow$ si considerano solo codifiche calcolabili in tempo polinomiale

Il cambiamento della base di rappresentazione (da base 2 a una qualsiasi base >2) non trasforma un problema risolvibile in tempo polinomiale in uno risolvibile in tempo superpolinomiale perché la conversione da una base all'altra avviene in tempo polinomiale (vedi Teorema successivo)



Funzioni calcolabili in tempo polinomiale

Se esiste un algoritmo con tempo polinomiale che, dato come input un qualsiasi $x \in \{0,1\}^*$, produce come output f(x), dove

$f: \{0,1\}^* \rightarrow \{0,1\}^*$,

si dice che f è una funzione calcolabile in tempo polinomiale

Codifiche correlate polinomialmente

Due codifiche e_1 ed e_2 dell'insieme I delle istanze di un problema astratto si dicono correlate polinomialmente se esistono due funzioni calcolabili in tempo polinomiale f_{12} e f_{21} che consentano di passare da una codifica all'altra, cioè tali che, $\forall i \in I$,

$f_{12}(e_1(i)) = e_2(i)$ e $f_{21}(e_2(i)) = e_1(i)$

Teorema. Sia Q un problema di decisione astratto su un insieme I di istanze; siano e_1 ed e_2 due codifiche di I correlate polinomialmente.
Allora $e_1(Q) \in P$ sse $e_2(Q) \in P$

↓

Si assume che la codifica di un intero sia correlata polinomialmente alla sua rappresentazione binaria, che la codifica di un insieme finito sia correlata polinomialmente alla sua codifica come lista di elementi separati da virgole, chiusa fra parentesi graffe, ecc.

↓

Da qui in poi si trascurerà la distinzione tra problema astratto e concreto

Linguaggi formali

Alfabeto Σ = insieme finito di simboli

Linguaggio L = insieme di stringhe costituite da simboli di Σ ($\rightarrow$ operazioni insiemistiche di unione e intersezione sui linguaggi)

Stringa vuota: ε

Linguaggio vuoto: Φ

Linguaggio di tutte le stringhe su Σ: Σ^* ($\rightarrow L \subseteq \Sigma^*$)

(Linguaggio) complemento di L: $\overline{L} = \Sigma^* - L$

(Linguaggio) concatenazione di due linguaggi L_1 e L_2: $L=\{x_1x_2: x_1 \in L_1 \text{ e } x_2 \in L_2\}$

(Linguaggio) chiusura (o stella di Kleene) di un linguaggio L:

$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots$

dove L^k è la concatenazione di L con se stesso k volte

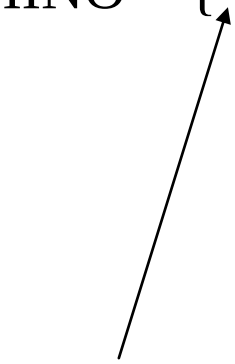
Problemi di decisione e linguaggi formali

Un qualunque problema di decisione Q si può considerare come un linguaggio L su $\Sigma = \{0,1\}$, esprimendo così in modo conciso la relazione fra problemi di decisione e algoritmi che li risolvono:

$L = \{x \in \Sigma^* : Q(x)=1\}$

Esempio. Il problema di decisione CAMMINO ha il corrispondente linguaggio (omonimo)

CAMMINO = {<G, u, v, k>: G = (V, E) è un grafo non orientato,
u, v $\in$ V,
k $\in N^+$ e
$\exists$ un cammino da u a v in G la cui lunghezza è al più k}



Linguaggi accettati

$L = \{x \in \Sigma^*: Q(x)=1\}$

è il linguaggio accettato (o riconosciuto) dall'algoritmo Q perché contiene tutte le stringhe x (istanze) accettate (o riconosciute) da Q, cioè quelle per cui Q(x)=1

L è riconosciuto in tempo polinomiale da Q se, $\forall x \in \Sigma^*$, $|x|=n$, Q accetta (cioè risolve) x in tempo $O(n^k)$, per qualche costante k, sse $x \in L$

↓

per riconoscere un linguaggio L, un algoritmo non deve rifiutare le stringhe di L

Stringhe rifiutate

Una stringa $x \in \Sigma^*$ si dice rifiutata dall'algoritmo Q se Q(x)=0

$x \notin L$ (linguaggio accettato da Q) $\not\rightarrow$ Q(x)=0

$x \notin L \rightarrow Q(x) \neq 1$ (ovvero Q(x)=0 $\vee$ Q(x) indefinito, ad es. l'esecuzione di Q in corrispondenza dell'istanza x può non terminare a causa di un loop infinito)

Linguaggi decisi

$L = \{x \in \Sigma^*: Q(x)=1\}$ si dice deciso dall'algoritmo Q sse $\forall x \notin L$, $Q(x)=0$

L è <u>deciso in tempo polinomiale</u> da Q se, $\forall x \in \Sigma^*$, $|x|=n$, Q decide circa x (cioè fornisce una risposta corretta sull'appartenenza di x a L) in tempo $O(n^k)$, per qualche costante k

↓

per decidere un linguaggio, un algoritmo deve accettare e rifiutare correttamente tutte le stringhe di $\{0,1\}^*$

Esempio: linguaggio CAMMINO

Sia dato un algoritmo Q di tempo polinomiale che calcola il cammino più corto tra i vertici u e v in G, usando una ricerca in ampiezza e confrontando poi la distanza ottenuta con k. Se la distanza è al più k, l'algoritmo restituisce 1 e si ferma, altrimenti continua la sua esecuzione all'infinito

Q riconosce il linguaggio CAMMINO (in tempo polinomiale) ma non lo decide (perché non restituisce 0 quando la lunghezza del cammino minimo è > k)

Per il linguaggio CAMMINO è semplice progettare un algoritmo che lo decide; per altri linguaggi (cioè problemi, ad es. la fermata di Turing) esistono algoritmi in grado di riconoscerli ma non di deciderli

Definizione alternativa di P

$P = \{L \subseteq \{0,1\}^* : L \text{ è deciso da un algoritmo in tempo polinomiale}\}$

Teorema.
$P = \{L \subseteq \{0,1\}^* : L \text{ è riconosciuto da un algoritmo in tempo polinomiale}\}$

$\downarrow$

$\text{CAMMINO} \in P$

Esempio: CICLO-HAMILTONIANO

Ciclo hamiltoniano di un grafo non orientato G = (V, E) = ciclo semplice che contiene tutti i vertici in V

Grafo hamiltoniano = grafo non orientato che contiene un ciclo hamiltoniano

Grafo non hamiltoniano = grafo non orientato che non contiene un ciclo hamiltoniano

Problema (di decisione) del ciclo hamiltoniano: “un grafo G ha un ciclo hamiltoniano?”

Linguaggio corrispondente:
CICLO-HAMILTONIANO = {<G>: G è un grafo hamiltoniano}

Esempio: CICLO-HAMILTONIANO (cont.)

Algoritmo (ingenuo) di decisione del linguaggio: elenca tutte le permutazioni dei vertici di G e poi verifica ciascuna di esse, fino a che non ne trova una che è un ciclo hamiltoniano o fino a quando ha controllato, senza successo, tutte le permutazioni

Codifica di un grafo: matrice di adiacenza
n = lunghezza della codifica di G = |<G>|

m = numero dei vertici del grafo = $\Omega(\sqrt{n})$
m! = numero permutazioni dei vertici

Tempo di esecuzione (superpolinomiale): $\Omega(m!) = \Omega(\sqrt{n}!) = \omega(2^{\sqrt{n}})$
(per l'approssimazione di Stirling)

Verifica

Sia A un algoritmo avente due stringhe (binarie) x e y come argomenti, dove x è un dato d'ingresso di un altro algoritmo e y è chiamata certificato

Si dice che

- A verifica la stringa x se esiste un certificato y tale che $A(x, y) = 1$
- il linguaggio verificato da A è
  $L = \{x \in \{0,1\}^* : \exists\, y \in \{0,1\}^* \text{ tale che } A(x, y) = 1\}$

Esempio: linguaggio/algoritmo CICLO-HAMILTONIANO

Certificato = sequenza dei vertici di un ciclo hamiltoniano

Algoritmo di verifica = controlla che il certificato sia una permutazione dei vertici di G e che ogni arco consecutivo del ciclo esista veramente in G

Tempo (polinomiale) di verifica: $O(n^2)$,
dove, come sopra, n = lunghezza della codifica di G = |<G>|

↓

Un algoritmo di decisione operante in tempo superpolinomiale può essere verificato in tempo polinomiale

Verifica in tempo polinomiale

L'operazione di verifica di un problema di decisione L avviene un tempo polinomiale se $\exists$ un algoritmo A di verifica operante in tempo polinomiale e una costante c tali che

$L = \{x \in \{0,1\}^*: \exists\ y \in \{0,1\}^* \text{ con } |y| = O(|x|^c) \text{ tale che } A(x, y) = 1\}$

(cioè il linguaggio del problema di decisione coincide col linguaggio verificato da A)

Un algoritmo di verifica può essere usato come il componente centrale di un algoritmo di forza bruta che risolva un problema L.
Ad esempio, per una stringa x, si provano tutte le stringe y, con $|y| \leq$ lunghezza massima dei certificati di x (dove tale lunghezza dipende polinomialmente da x), e si vede se $A(x, y) = 1$ per qualcuna di esse. Trovata un stringa y cosiffatta, è $L(x) = 1$

Classe di complessità NP

Insieme dei linguaggi che possono essere verificati in tempo polinomiale.

Più formalmente, $L \in NP$ se $\exists$ un algoritmo A di verifica operante in tempo polinomiale e una costante c tale che

$L = \{x \in \{0,1\}^*: \exists\ y \in \{0,1\}^* \text{ con } |y| = O(|x|^c) \text{ tale che } A(x, y) = 1\}$

$\downarrow$

CICLO-HAMILTONIANO $\in$ NP
(in particolare, CICLO-HAMILTONIANO è NP-completo)

$P \subseteq NP$
perché, se $\exists$ un algoritmo A di tempo polinomiale che decide il linguaggio L, allora A può essere trasformato in un algoritmo di verifica che semplicemente ignora qualsiasi certificato e accetta esattamente quelle stringhe che esso determina appartenere a L

Classe di complessità co-NP

Insieme dei linguaggi L tali che $\overline{L} \in NP$

P è chiuso rispetto all'operazione di complemento (cioè $L \in P \rightarrow \overline{L} \in P$; infatti un algoritmo $\overline{L}$ può essere realizzato invocando L e complementando la sua uscita); pertanto $P \subseteq NP \cap$ co-NP

Domanda (aperta): lo è anche NP (cioè $L \in NP \rightarrow \overline{L} \in NP$ ovvero NP = co-NP) ?

Teorema. Se NP ≠ co-NP, allora P ≠ NP

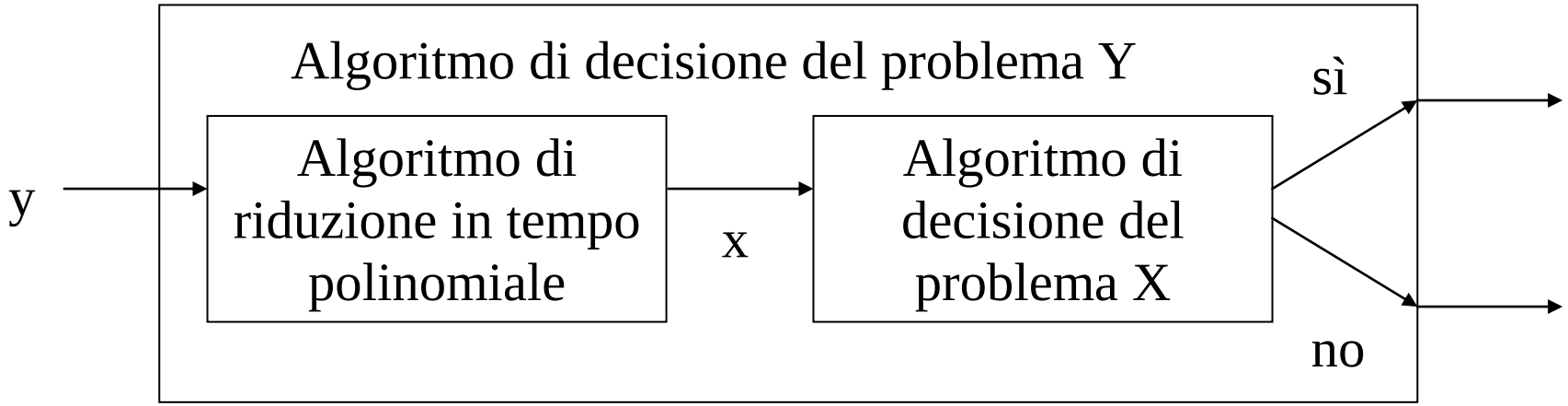
Riducibilità fra problemi

Siano X e Y siano due problemi.
Supponiamo che X possa essere risolto direttamente da un algoritmo in tempo polinomiale secondo il modello computazionale adottato.

Domanda: è possibile risolvere istanze arbitrarie del problema Y attraverso un numero polinomiale di passi computazionali più un numero polinomiale di chiamate all'algoritmo che risolve X?

Se la risposta è sì, allora si dice che "Y è riducibile in tempo polinomiale a X" (Y non è più difficile da risolvere di X rispetto al tempo polinomiale)" e si scrive

$$Y \leq_p X$$



Riducibilità fra problemi (cont.)

Teorema. Si supponga che $Y \leq_p X$. Se X può essere risolto in tempo polinomiale, allora Y può essere risolto in tempo polinomiale.
(Formulazione equivalente: Si supponga che $Y \leq_p X$. Se Y non può essere risolto in tempo polinomiale, allora X non può essere risolto in tempo polinomiale.)

Impiego (desiderato) del teorema: se Y è arduo e $Y \leq_p X$, allora la difficoltà di Y si propaga a X (X deve essere arduo altrimenti potrebbe essere usato per risolvere Y)

Impiego (effettivo) del teorema: dal momento che non possiamo sapere se Y è arduo (cioè se sia impossibile risolverlo in tempo polinomiale), si usa $Y \leq_p X$ per stabilire livelli relativi di difficoltà dei problemi, in particolare per stabilire quali sono i problemi più ardui entro NP

Riducibilità fra problemi (cont.)

In particolare, un problema Q è riducibile a un problema Q' se un'istanza x di Q può essere riformulata come istanza x' di Q' e la soluzione di x' fornisce una soluzione di x $\rightarrow$ Q non è più arduo da risolvere di Q'

Esempio. Il problema di risolvere equazioni lineari si riduce al problema di risolvere equazioni quadratiche
(un'istanza $ax + b = 0$ si trasforma in $0\ x^2 + ax + b = 0$)

N.B. Sebbene la definizione di $\leq_p$ consenta di effettuare più chiamate dell'algoritmo risolvente del problema a dx per risolvere un'istanza di quello a sinistra, in questo esempio e in tutta la trattazione successiva si effettua una singola chiamata. In un caso come questo a ciascuna istanza del problema a sx corrisponde un'istanza di quello a dx e il problema a dx è una *generalizzazione* di quello a sx

Transitività della riducibilità

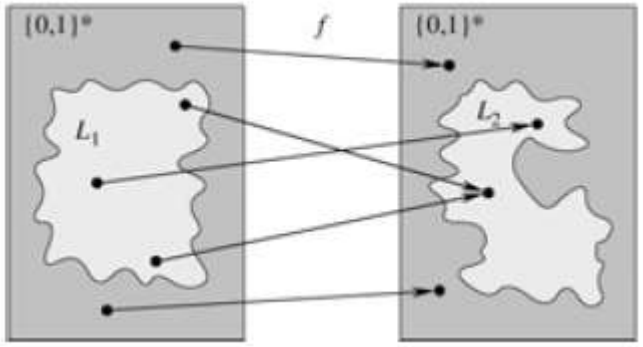
Teorema. Se $Z \leq_p Y$ e $Y \leq_p X$, allora $Z \leq_p X$.

Riducibilità fra linguaggi

Un linguaggio L_1, associato a un problema di decisione, è riducibile in tempo polinomiale a un linguaggio L_2, associato a un altro problema di decisione, se $\exists$ una funzione f: $\{0,1\}^* \rightarrow \{0,1\}^*$ calcolabile in tempo polinomiale tale che, $\forall x \in \{0,1\}^*$, $x \in L_1$ sse $f(x) \in L_2$.
Formalmente si scrive $L_1 \leq_p L_2$ perché L_1 non è più arduo di L_2 (e, dualmente, L_2 è arduo almeno quando L_1)

f: funzione di riduzione (calcolata in tempo polinomiale dall'algoritmo di riduzione)

Decidere se $f(x) \in L_2$ consente di decidere anche se $x \in L_1$



Teorema. Se $L_1 \leq_p L_2$, allora $L_2 \in P$ implica $L_1 \in P$

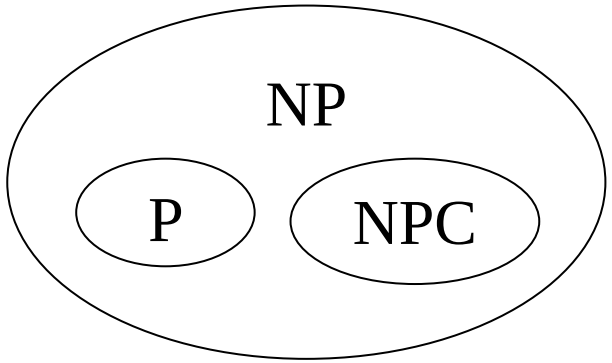
Linguaggi NP-completi – classe NPC

Un linguaggio $L \subseteq \{0,1\}^*$ è NP-completo se

1. $L \in NP$
2. $\forall L' \in NP,\ L' \leq_p L$ (in tal caso L è NP-hard)

Teorema. Se un qualsiasi problema NP-completo è risolvibile in tempo polinomiale, allora P = NP.
(Equivalentemente, se un qualsiasi problema in NP non è risolvibile in tempo polinomiale, allora nessun problema NP-completo è risolvibile in tempo polinomiale.)

Visione attuale



Dimostrazioni di NP-completezza

Teorema. Se $L' \leq_p L$ per qualche $L' \in NPC$, allora L è NP-hard

Dimostrazione.

a) $L' \in NPC$; quindi $\forall L'' \in NP$, $L'' \leq_p L'$
b) $L' \leq_p L$; quindi, per transitività, $L'' \leq_p L$

↓

Metodo per dimostrare che $L \in NPC$

1. Si dimostri che $L \in NP$
2. Si selezioni un linguaggio L' NP-completo noto
3. Si descriva un algoritmo che calcoli una funzione f che fa corrispondere a ogni istanza di L' un'istanza di L
4. Si dimostri che $x \in L'$ sse $f(x) \in L$, $\forall x \in \{0,1\}^*$
5. Si dimostri che l'algoritmo che calcola f è eseguito in tempo polinomiale

Ma come si è dimostrata la NP-completezza la prima volta nel 1971, quando non esisteva ancora alcun problema NP-completo noto?

Problema SODDISFATTIBILITÀ-DI-CIRCUITI

È il problema per cui si è effettuata la prima dimostrazione di NP-completezza

Istanza del problema: un circuito combinatorio c dotato di una sola uscita
Soluzione: SODDISFATTIBILITÀ-DI-CIRCUITI(c): 1 (sì) se c è soddisfattibile, 0 (no) altrimenti

c è soddisfattibile se ha (almeno) un assegnamento di soddisfabilità, cioè un assegnamento di valori (booleani) agli ingressi che comporta il valore 1 come uscita

Problema SODDISFATTIBILITÀ-DI-CIRCUITI (cont.)

Alla base della dimostrazione che il problema è NP-completo sta il fatto che ogni algoritmo A che acquisisce in ingresso un numero n di bit e produce una risposta 0/1 può essere rappresentato mediante un circuito c come sopra descritto
$\rightarrow$ c è equivalente ad A
Se A effettua un numero di passi polinomiale in n, allora il circuito ha dimensione polinomiale

Si vuole dimostrare che X $\leq_p$ SODDISFATTIBILITÀ-DI-CIRCUITI, dove X ha un tempo di verifica polinomiale; cioè, dato un input s (di lunghezza n), si vuole decidere se $s \in X$ risolvendo istanze di SODDISFATTIBILITÀ-DI-CIRCUITI; ciò richiede di rispondere alla domanda se esista un certificato t, di lunghezza polinomiale in n, p(n), tale che l'algoritmo di verifica di X, ricevendo in ingresso s e t, produca in uscita 1
Convertiamo tale algoritmo di verifica in un circuito c con n+p(n) bit di ingresso; i primi n bit assumeranno i valori dei bit di s, mentre i rimanenti p(n) saranno delle variabili $\rightarrow$ $s \in X$ sse c è soddisfattibile $\rightarrow$
X $\leq_p$ SODDISFATTIBILITÀ-DI-CIRCUITI