La classe NP, i problemi NP-completi e la congettura P≠NP

Alberto Marchetti Spaccamela

Sapienza Università di Roma

A.A.2020-2021

Richiami di complessità

Ricordiamo la notazione introdotta

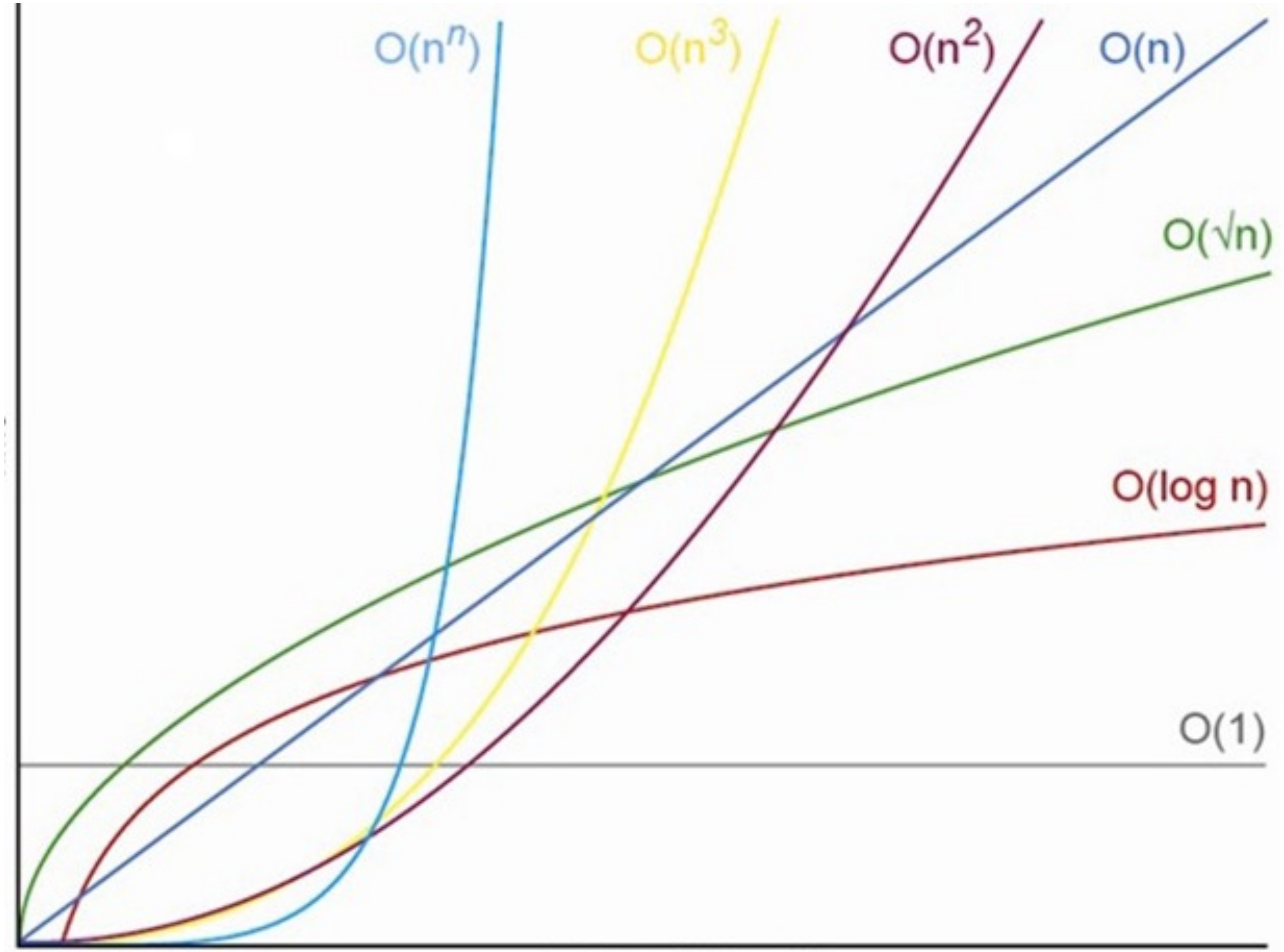
- $f(n)$ è $O(g(n))$ se esistono un intero n_0 e due costanti $c>0$ e b tali che per ogni $n > n_0$: $f(n) \leq c\, g(n) + b$
- $f(n)$ è $\Omega(g(n))$ se esistono un intero n_0 e una costante $c>0$ tali che per ogni $n > n_0$: $f(n) \geq c\, g(n)$
- $f(n)$ è $\theta(g(n))$ se $f(n)$ è sia $O(g(n))$ che $\Omega(g(n))$

Esempi:

1. $n^2+ 3n +2$ è $O(n^2)$ ($n_0=3$, $c=2$, $b= 2$)
2. $n^2+ 3n +2$ è $\Omega(n^2)$ ($n_0=1$, $c=1$)
3. Nota: 1. e 2. precedenti implicano che $n^2+ 3n +2$ è $\theta(n^2)$
4. n^2 è $O(2^n)$ ($n_0=4$, $c=1$) (ma n^2 NON è $\Omega(2^n)$)
5. $2^{(n+10)}$ è $O(2^n)$ ($n_0=1$, $c=1024$, $b=0$) e è $\Omega(2^n)$ ($n_0=1$, $c=1$)
6. $1000\, n^{100} + 2^n$ è $O(2^n)$ ($n_0=1$, $c=1000$, $b=0$)

Andamento asintotico

T
e
m
p
o



Dimensione input

Crescita esponenziale

Assumi di mettere un chicco di riso su una casella della scacchiera, poi 2 sulla successiva e così via: in ogni casella metti il doppio della casella precedente

- Sull'ultima casella si pongono 2^{63} chicchi di riso
- In totale sulla scacchiera ci sono 2^{64}-1 chicchi di riso

Si noti la crescita esponenziale del numero

2^{64}-1 = 18.446.744.073.709.551.615 (> 18 miliardi di miliardi) chicchi di riso

che corrispondono a circa 210 miliardi di tonnellate

(sufficienti a coprire l'intero territorio dell'Italia con uno strato di riso alto circa 11 metri)

Diversi tipi di problemi

Problemi decisionali: con risposta SI o NO

Problemi di ottimizzazione: con risposta un valore

- Ogni problema di ottimizzazione può essere riformulato come problema decisionale.
  - Problema ottimizzazione: qual è la distanza minima da casa a Palermo?
  - Problema di decisione: la distanza minima da casa a Palermo è minore di 750 Km?
- Il problema decisionale ha complessità minore o uguale al problema non decisionale corrispondente
  - Quindi se il problema decisionale è difficile, a maggior ragione lo è quello di ottimizzazione
- In molti casi i due problemi hanno la stessa complessità: possiamo risolvere il problema di ottimizzazione risolvendo un numero ridotto di problemi di decisione (usando ad esempio la ricerca binaria)

Soddisfacibilità - SAT (decisione)

SAT: soddisfacibilità di una formula nel calcolo proposizionale

Data una formula logica F con n variabili (che utilizza operatori di AND, OR, negazione e le parentesi), trovare, se esiste, una combinazioni di valori booleani che, assegnati alle variabili di F, la rendono vera

Esempi

F1: (x and not x) or (y and not y) non è soddisfacibile

F2: (x and not y) or (not x and y) è soddisfacibile (x=falso, y=vero)

F3: (x and not y) and (not x and y) non è soddisfacibile

F4: (x or y)and(not x or y)and(x or not y)and(not x or not y) non è soddisfacibile

Nota gli esempi precedenti evidenziano che:

- dimostrare che una formula sia soddisfacibile è più facile di
- dimostrare che una formula NON sia soddisfacibile

Soddisfacibilità - SAT (decisione)

Algoritmo di forza bruta: data una formula F con n variabili

1. Si provano tutte le combinazioni di valori delle variabili
2. Per ciascuna si verifica se soddisfa la formula; se soddisfa temina con successo (esiste assegnazioni di valori che rende la formula vera)
3. Se nessuna assegnazione soddisfa la formula allora la formula non è soddisfacibile

Nota: non è difficile trovare un algoritmo polinomiale per il passo 2

Analisi

- Ogni variabile può assumere due valori (vero e falso)
- Dato che le variabili sono n tutte le possibili assegnazioni dei valori delle variabili sono 2^n
- L'algoritmo ha costo $O(2^n)$ - o più precisamente $O(p(|F|)\ 2^n)$,dove $p(|F|)$ rappresenta il costo per verificare se una data assegnazione di valori alle variabili soddisfa F ($p(|F|)$ è una funzione polinomiale nelle dimensioni di F)

Problema Knapsack 0-1 (Zaino) (ottimizzazione)

Input: una lista di oggetti I0, I1,.. I(n-1)

Ogni oggetto ha due attributi:

Valore (beneficio): vi il valore dell'oggetto i

Peso: wi il peso dell'oggetto i

- Il beneficio e il valore complessivo di un insieme di oggetti è dato dalla somma dei benefici degli oggetti dell'insieme
- L'obbiettivo è massimizzare il valore di uno zaino che può contenere al massimo W chili di peso di elementi (oggetti) scelti da una lista data I0, I1, … In-1
- Un oggetto è preso interamente o per niente; da questo il nome del problema Knapsack 0-1 (Zaino 0-1).

Problema Knapsack 0-1 (ottimizzazione)

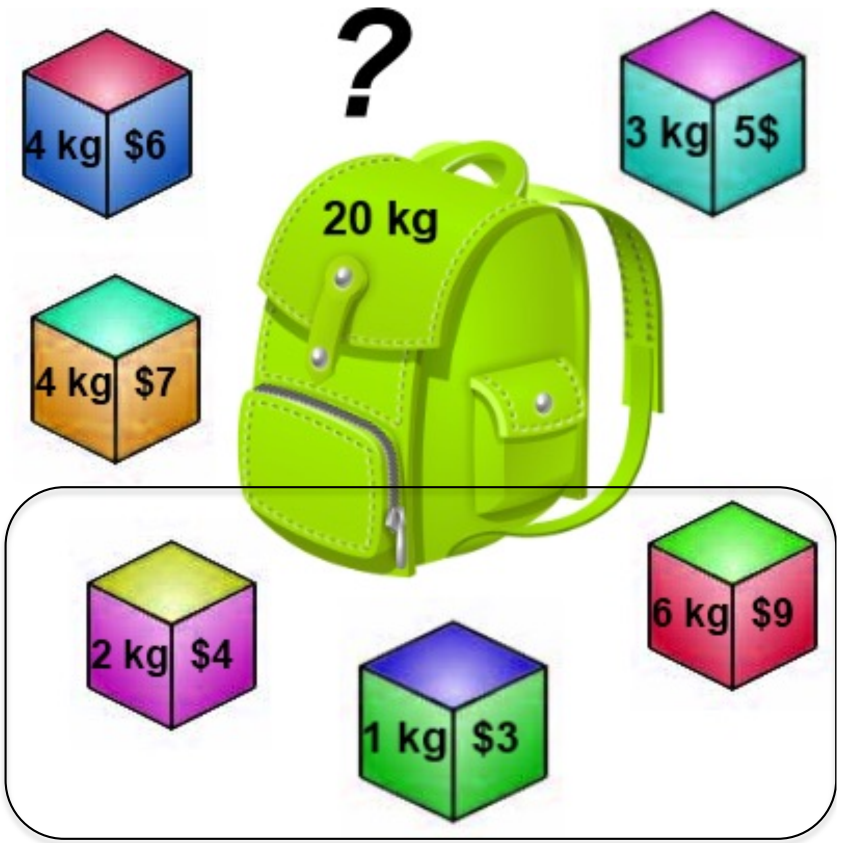
Input: una lista di oggetti I0, I1,.. In-1

Ogni oggetto ha due attributi:

Valore (beneficio): vi il valore dell'oggetto i

Peso: wi il peso dell'oggetto i

- Il beneficio e il valore complessivo di un insieme di oggetti è dato dalla somma dei benefici degli oggetti
- L'obbiettivo è massimizzare il valore di uno zaino che può contenere al massimo W chili di peso di elementi (oggetti) scelti da una lista data I0, I1, … In-1
- Un oggetto è preso interamente o per niente; da questo il nome del problema Knapsack 0-1 (Zaino 0-1).



Insieme di peso complessivo 9 kg
e beneficio complessivo 13

Problema Knapsack 0-1 (ottimizzazione)

Algoritmo Forza bruta

enumera tutti i possibii 2^n sottoinsiemi di n elementi e sceglie la migliore combinazione (massimo beneficio rispettando il vincolo del peso)

sia val(S) profitto di insieme di oggetti S

Max= 0

Per ogni insieme di oggetti S do

if S è soluzione ammissibile

then {if val(S)> Max then Max= val(S)}

- posso rappresentare con una stringa binaria lunga n un sottoinsieme di un insieme di n oggetti
  - Es. Supponi 5 oggetti: la stringa 00000 rappresenta l'insieme vuoto, la stringa 11000 l'insieme con i primi due oggetti

Problema commesso viaggiatore (ott.)

Sia data una mappa di una regione che indica la distanza fra le diverse città e un insieme di città S da visitare e una città di partenza

Trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città in S una e una sola volta per poi tornare alla città di partenza

Formulazione con grafi
Input: un grafo con archi etichettati con un peso (che rappresenta la distanza fra i due vertici)
Trova il percorso che parte dal vertice 1, visita tutti i vertici del grafo una sola volta e termina in 1

Problema commesso viaggiatore (ott.)

Algoritmo Forza bruta: supponi che il grafo contenga n città (nodi)
Idea: enumera tutte le possibii n! sequenze di città;
Data sequenza T ordinata di città sia Len(T) il costo di questa soluzione

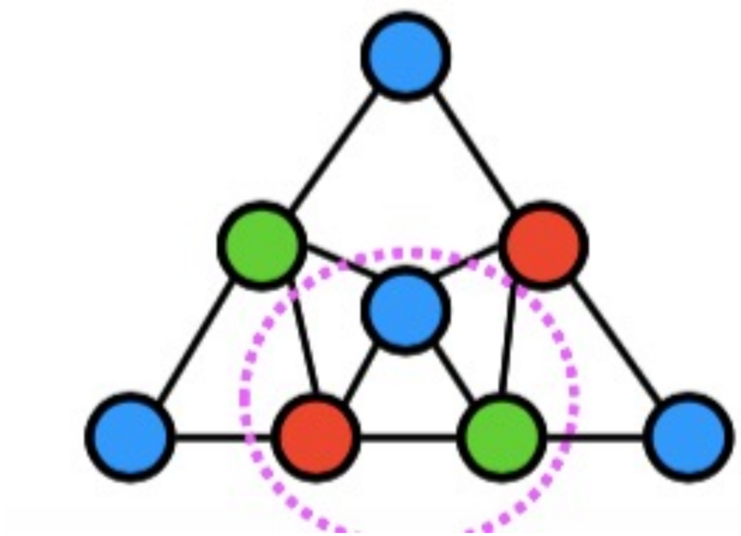
Sia T_0 percorso di visita delle città nell'ordine 1,2...n,1 (parto da 1 e ritorno in 1)
min = Len(T_0)
Per ogni sequenza di città T do
    calcola Len(T) (partendo da 1 e tornando in 1)
    if Len(T) < min then min= Len(T)

- Il costo di questo algoritmo è almeno pari a (n-1)!, numero delle sequenza considerate
- Ricorda che $n! > (n/e)^n$ dove e (=2,71…) è la costante di Eulero;

Colorazione di grafi (decisione)

Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

- Il numero cromatico di G, C(G) è il minimo numero di colori necessario per colorare tutti i suoi nodi
- Chiaramente se un grafo ha n nodi abbiamo che $1 \leq C(G) \leq n$
- Il problema di decisione associato è: dato un grafo G e intero k posso colorare G con k colori (k intero)?



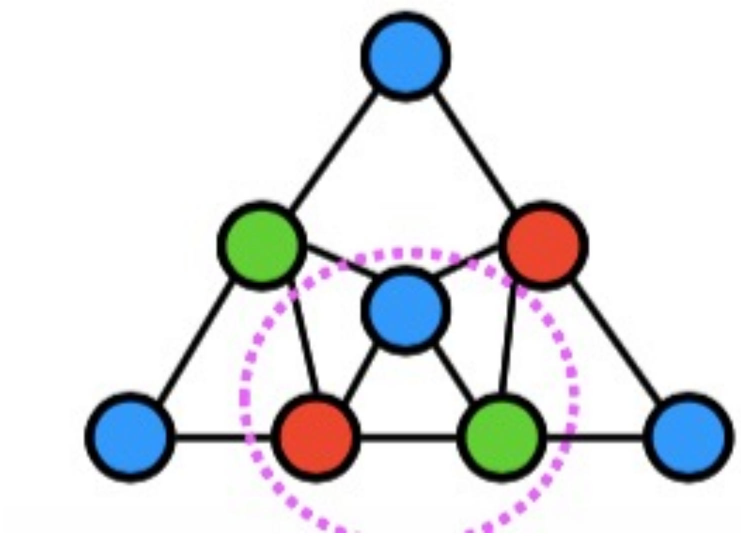
La figura mostra una colorazione con 3 colori

Si noti che se tre nodi sono mutuamente adacenti e formano una cricca allora richiedono tre colori diversi (vedi nodi cerchiati a sinistra)

Colorazione di grafi (decisione)

Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

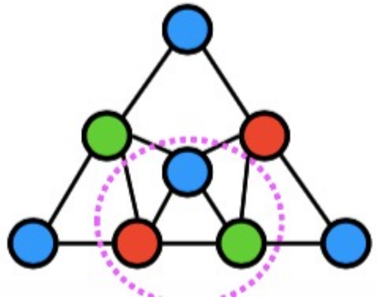
- Il problema di decisione associato è: dato un grafo G e un intero k il grafo G può essere colorato con k colori?



La figura mostra una colorazione con 3 colori

Si noti che se tre nodi sono mutuamente adacenti e formano una cricca allora richiedono tre colori diversi (vedi nodi cerchiati a sinistra)

Colorazione di grafi (ottimizzazione)



Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

- Il numero cromatico di G, C(G) è il minimo numero di colori necessario per colorare tutti i suoi nodi

Dato un algoritmo ALG(G,k) per il problema di decidere se possiamo colorare G con k colori posso risolvere il problema di ottimizzazione

- se un grafo ha n nodi abbiamo che $1 \leq C(G) \leq n$
- Quindi possiamo provare tutte le possibilità

Algoritmo, input un grafo G
Fine= falso
While fine= falso
    do if ALG(G,k) then fine= true else k=k+1

Numero di tentativi da fare con k colori su grafo di n nodi è $\binom{n}{k}$ -binomiale
Il binomiale cresce come $O(n^k)$ – esponenziale in k

Problema Knapsack 0-1 (decisione)

Input: una lista di oggetti I0, I1,.. In-1

Ogni oggetto ha due attributi:

Valore (beneficio): vi il valore dell'oggetto i

Peso: wi il peso dell'oggetto i

- Il beneficio e il valore complessivo di un insieme di oggetti è dato dalla somma dei benefici degli oggetti

Problema di decisione

Esiste un insieme di oggetti da inserire nello zaino di peso non superiore a W chili di peso e beneficio almeno pari a B?

- Il beneficio massimo è dato dalla somma dei valori di tutti gli oggetti Σ vi; il beneficio è compreso fra 0 e Σ vi
- Dato un algoritmo per il problema di decisione uso ricerca binaria per risolvere il problema di ottimizzazione

No. Problemi di decisione:
$\log_2 (\Sigma\, vi) \leq \log_2 (n\, v(max))$
$= \log_2 n + \log_2 v(max))$

v(max) = massimo beneficio

Diversi tipi di problemi

- Problemi decisionali: con risposta SI o NO
- Problemi di ottimizzazione: con risposta un valore

1. Ogni problema di ottimizzazione può essere riformulato come problema di decisione.
2. Applicare più volte un algoritmo per il problema di decisione permette in molti casi di risolvere il problema di ottimizzazione
3. Il numero di problemi decisionali da risolvere è una funzione polinomiale delle dimensioni dell’input

- se il problema decisionale è difficile, a maggior ragione lo è il problema di ottimizzazione
- Poiché il numero di problemi decisionali è polinomiale i due problemi di ottimizzazione e decisione sono correlati da un polinomio

Diversi tipi di problemi

Per risolvere il problema di ottimizzazione devo eseguire un numero polinomiale (nella grandezza dell'input) di problemi decisionali

Quindi

1. se il problema di decisione è facile (polinomiale) anche il problema di ottimizzazione è facile (polinomiale)
2. se il problema decisionale è difficile, a maggior ragione lo è il problema di ottimizzazione

Conclusione

possiamo limitarci a studiare la complessità dei problemi di decisione

SAT

Consideriamo problemi di decisione

Data una formula F del calcolo proposizionale bisogna decidere se la formula è soddisfacibile

Data un'assegnazione di valori di verità possiamo decidere se la formula è vera. Costo : polinomiale

Certificato di un problema P: data un'istanza di un problema decisionale P un certificato è dato da un insieme di informazioni che permette di verificare se P ha soluzione

SAT

Data una formula e un'assegnazione di valori di verità alle variabili possiamo decidere se la formula è soddisfacibile.

Costo : polinomiale nella dimensione dell'input

Pertanto per decidere se una formula F del calcolo proposizonale decidere se è soddisfacibile possiamo provare tutte le possibili di assegnazioni di valori di verità e verificare se uno soddisfa

- Se SI → la formula è soddisfacibile
- Se nessun valore soddisfa → la formula NON è soddisfacibile

Costo di questo algoritmo?

Costo polinomiale p(n) per ogni possibile assegnazione di valori

Il numero di possibili assegnazioni di una formula con n variabili è 2^n

Costo totale algoritmo ($p(n)\, 2^n$) esponenziale

Macchina di Turing non deterministiche

In una macchina di Turing deterministica dato uno stato q e un simbolo s letto dalla testina, f(q,s)=(q', s', m) determina la transizione da eseguire ed è univocamente determinata.

Una macchina di Turing non deterministica è una macchina di Turing a cui non viene imposto il vincolo che, dato uno stato della macchina e un simbolo letto dalla testina, la transizione da eseguire sia univocamente determinata.

Quindi, f(q,s) non è una funzione ma per un dato q e s possiamo avere più valori.

La macchina termina con successo se almeno una scelta porta a stato finale

Macchina di Turing non deterministiche

Una macchina di Turing non deterministica è una macchina di Turing a cui non viene imposto il vincolo che, dato uno stato della macchina e un simbolo letto dalla testina, la transizione da eseguire sia univocamente determinata.

La computazione di una macchina di Turing T non è più rappresentabile mediante una sequenza di configurazioni.

Tuttavia, essa può essere vista come un albero, detto **albero delle computazioni**, i cui nodi corrispondono alle configurazioni di T e i cui archi corrispondono alla produzione di una configurazione da parte di un'altra configurazione.

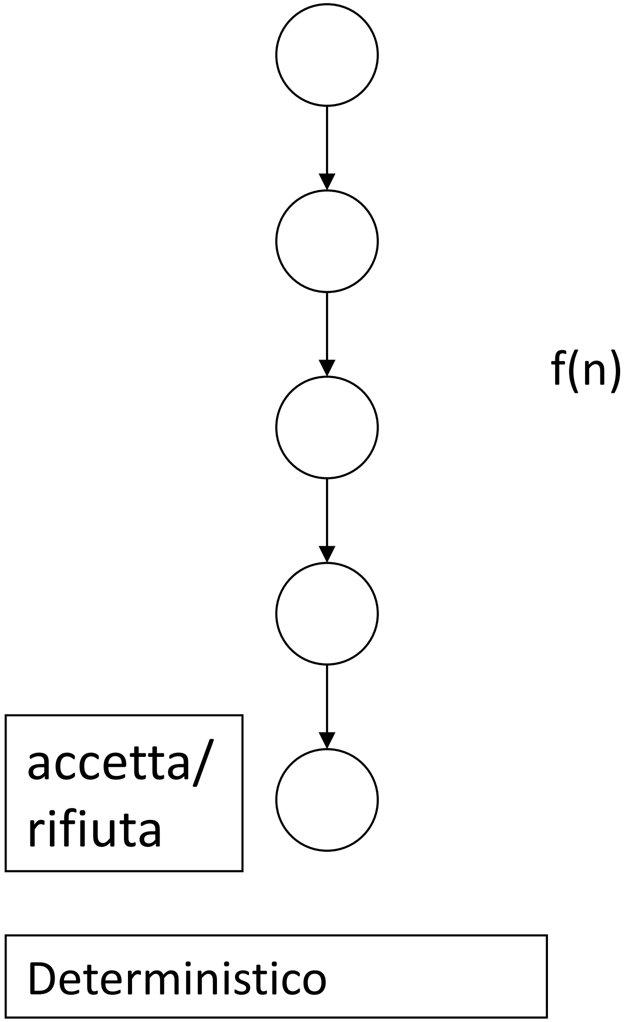
Macchina di Turing non deterministiche

La computazione di MdT può essere vista come un albero, detto **albero delle computazioni**, i cui nodi corrispondono alle configurazioni di T e i cui archi corrispondono alla produzione di una configurazione da parte di un'altra configurazione.

Un input x è accettato da una macchina di Turing non deterministica T se l'albero delle computazioni corrispondente a x include almeno un cammino di computazione accettante, ovvero un cammino di computazione che termini in una configurazione finale.

L'insieme delle stringhe accettate da T è detto essere il linguaggio accettato da T ed è indicato con L(T).

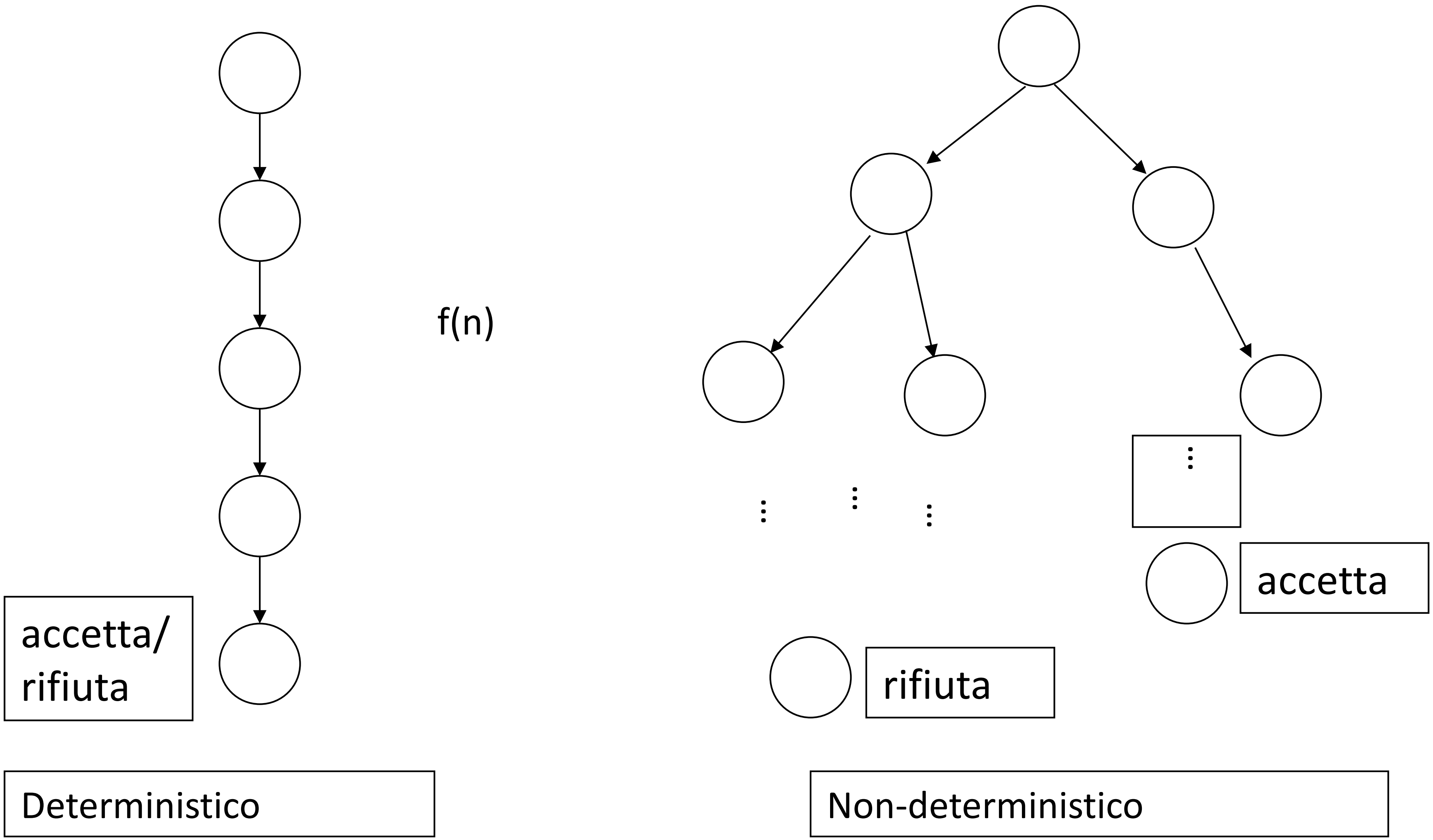
Albero di computazione Non-deterministico



Una macchina di Turing deterministica ha un solo cammino di esecuzione

La macchina accetta/rifiuta se il cammino accetta rifiuta

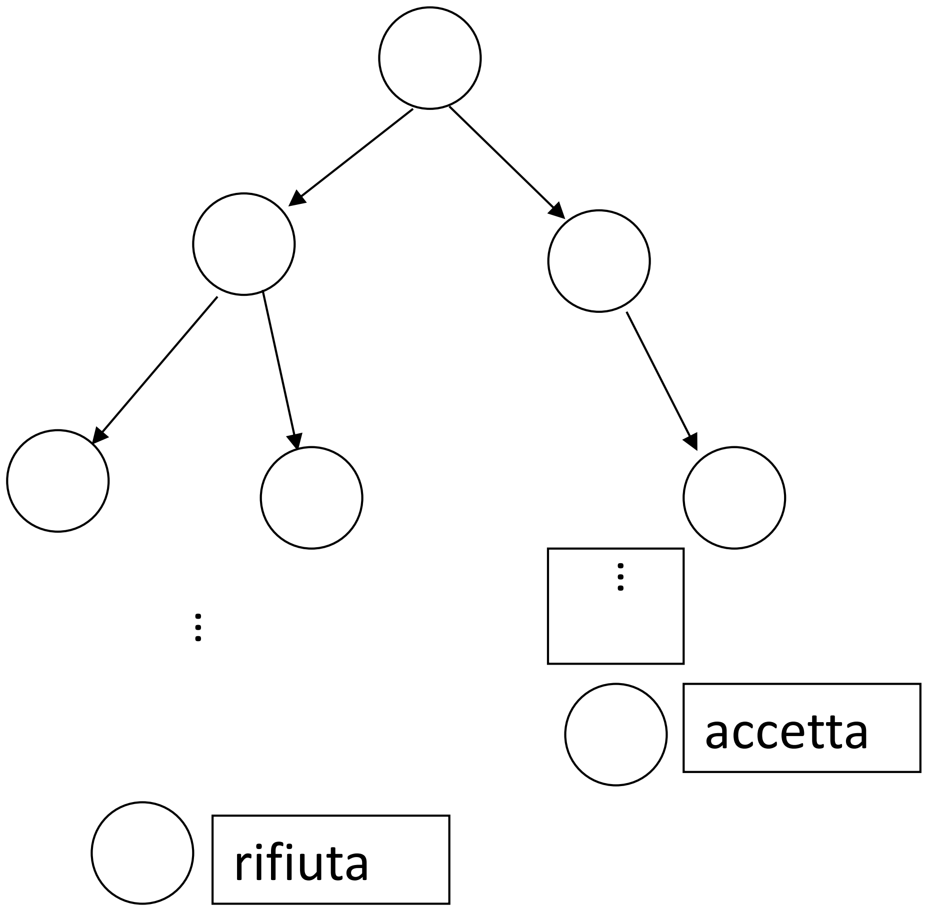
Albero di computazione Non-deterministico



Albero di computazione Non-deterministico

Una macchina di Turing Non deterministica ha molti cammini di esecuzione

La macchina accetta se almeno un cammino accetta
Rifiuta se tutti i cammini rifiutano



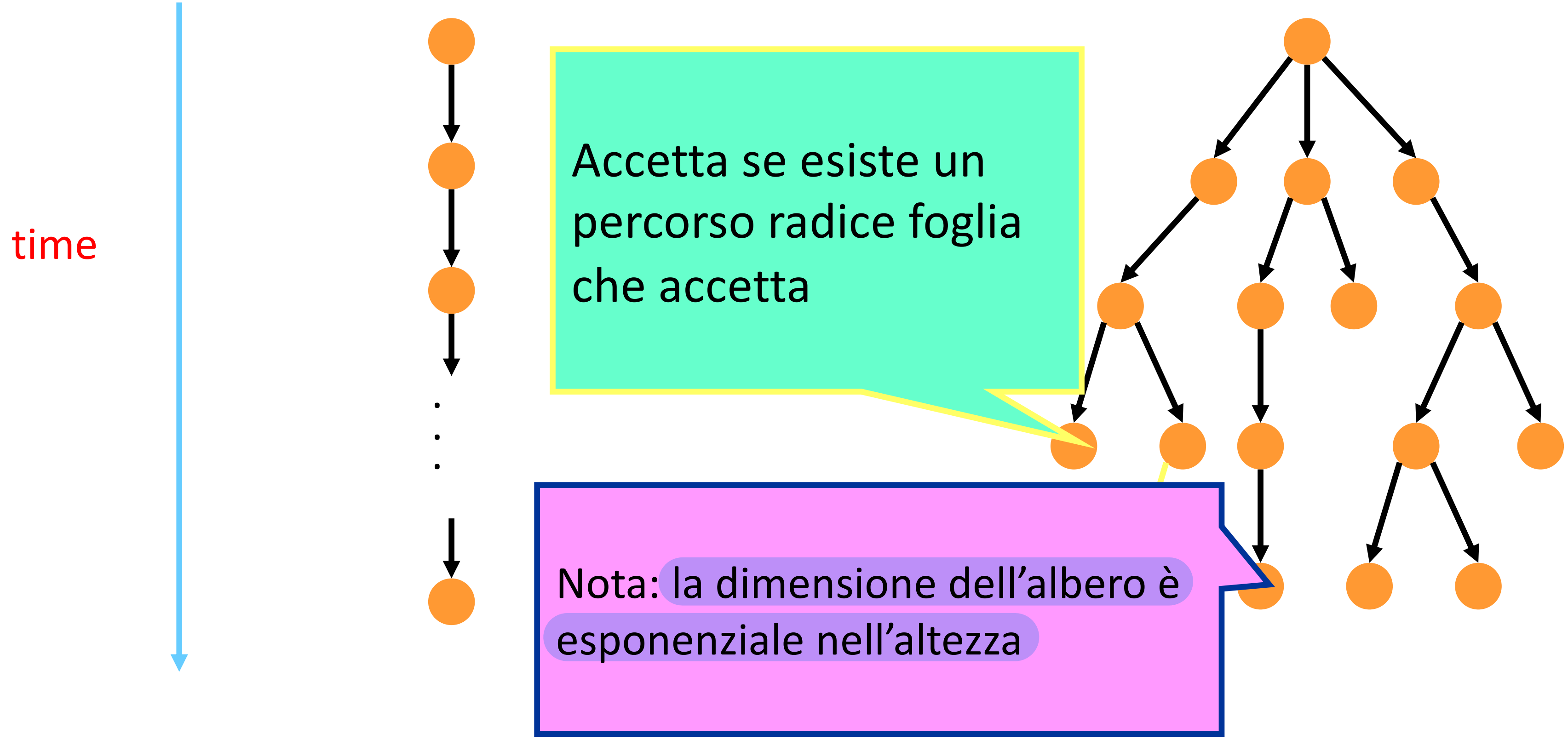
Deterministico

Non-deterministico

Albero di computazione Non-deterministico

Computazione di una
MdT deterministica

Albero di Computazione di
una
MdT non-deterministica



Descrizione alternativa

Una macchina di Turing nondeterministicaè come un indovino: indovina sempre la scelta giusta. In altre parole

- Una MdT non deterministica sceglie sempre il cammino corretto (se esiste)
- E poi verifica la correttezza del cammino

Esempio

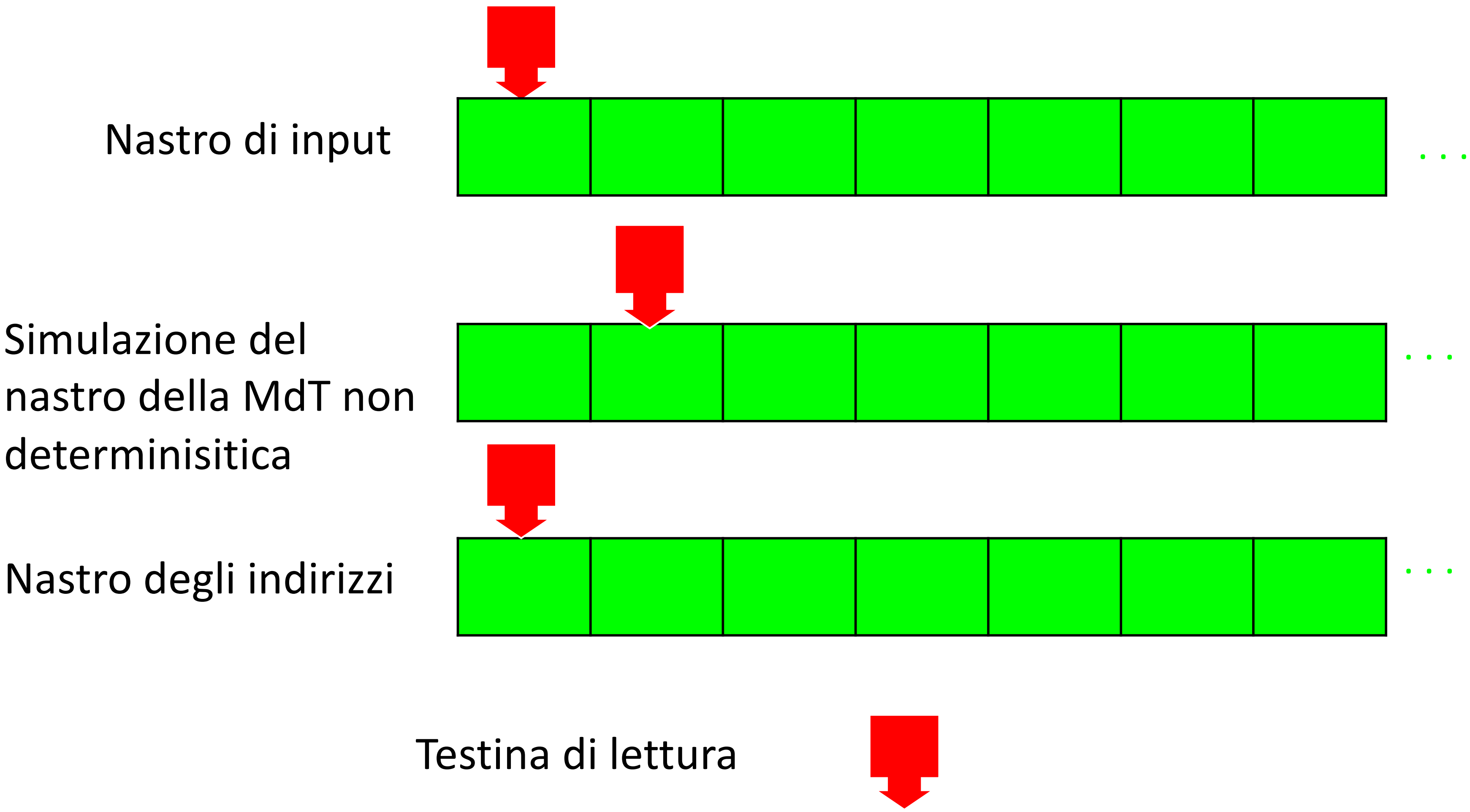
Una macchina di Turing non deterministica TM che verifica

- se due vertici di un grafo in input sono connessi;
  - 1) indovina un cammino fra i due vertici e
  - 2) verifica che sia un cammino valido
- Se esiste un cammino fra due vertici di un grafo pesato dato in input di lunghezza al più L è analogo:
  - 1) indovina un cammino fra i due vertici e
  - 2) verifica che sia un cammino valido di lunghezza al massimo L
- Se esiste un'assegnazione di valori di verità alle variabili di una formula logica che la rende vera;
  - 1) indovina un'assegnazione di valori vero/falso alle variabili e
  - 2) verifica che sia un assegnazione che rende vera la formula

Simulazione di una MdT Non deterministica con una MdT Deterministica

- Vediamo come una MdT deterministica con 3 nastri sia in grado di simulare una macchina di Turing non deterministica con un solo nastro
- Nota: usare 3 nastri non è limitativo: abbiamo visto che a sua volta una MdT deterministica con 1 solo nastro è in grado di simulare una MdT con 3 nastri
- La simulazione esplora TUTTO l'albero delle computazioni e quindi richiede tempo esponenziale nel caso peggiore
- MdT non deterministiche non esistono al momento ma gli studi sui calcolatori quantistici possono portare a simulazioni migliori

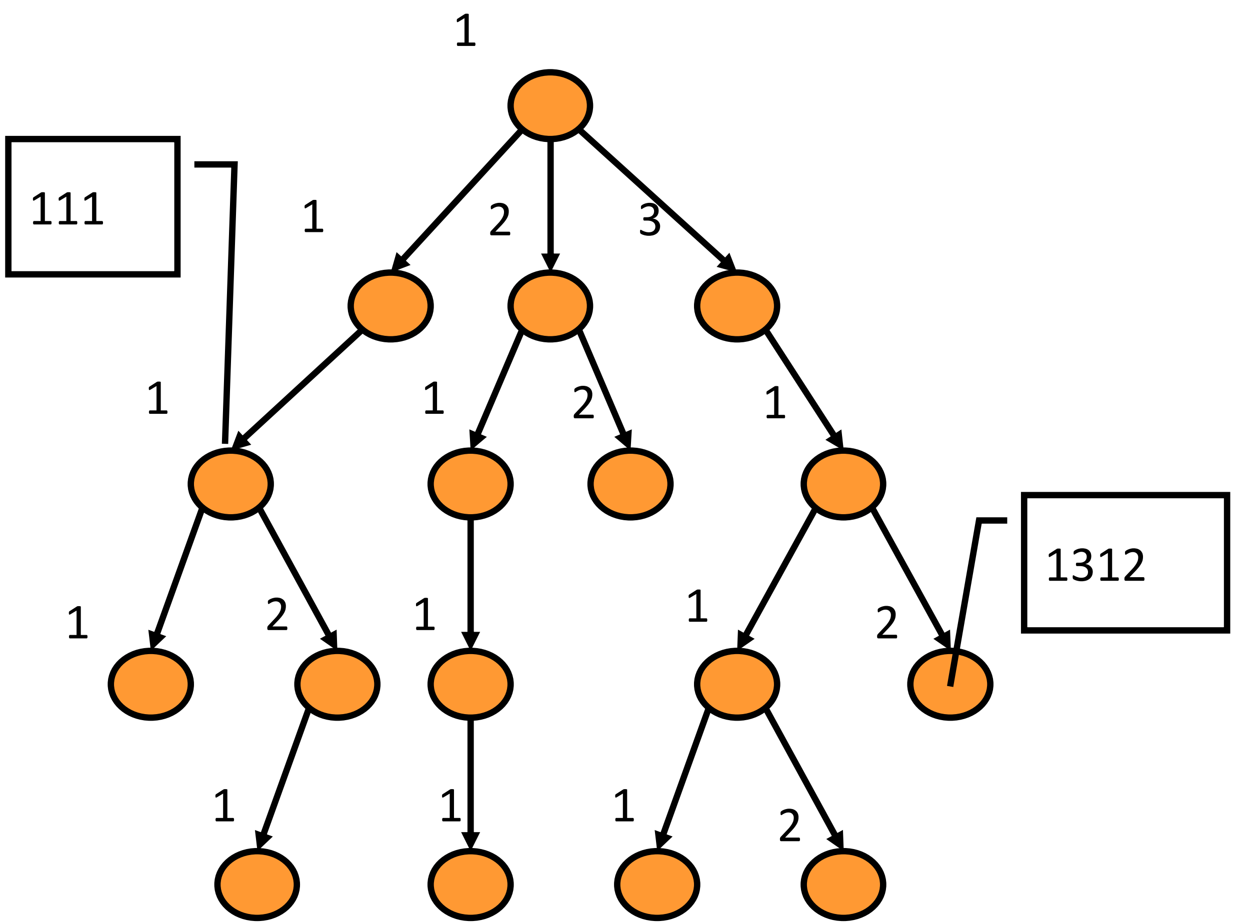
Simulazione di una MdT Non deterministica con una MdT Deterministica



Simulazione di una MdT Non deterministica con una MdT Deterministica: Indirizzi

Albero delle computazioni non determinisitico

- Ogni cammino dalla radice ad un nodo del grafo è rappresentato da una lista di indirizzi
- Ciascun elemento della stringa indica una direzione nell’albero



Simulazione di una MdT Non deterministica con una MdT Deterministica

1. Scrivi 111…1 sul nastro degli indirizzi (nastro 3)
2. Copia l'input sul nastro di simulazione (nastro 2)
3. Simula la MdT non determinisitica usando le scelte indicate nel nastro degli indirizzi
4. Se le scelte sono valide e arrivi in uno stato accetta allora – accetta e termina
5. Se le scelte non sono valide o arrivi in uno stato di rifiuto aggiorna il nastro degli indirizzi con la stringa successiva in ordine lessicografico; se tutte le scelte sono state provate – rifiuta
6. Vai a 2

Nota:

- il nastro 1 è di sola lettura (contiene la MdT non deterministica); il nastro 3 (degli indirizzi) memorizza un cammino dalla radice nell'albero delle computazioni della macchina non deterministica
- La MdT deterministica visita l'albero delle computazioni fino a quando trova un cammino accettante o quando ha visitato TUTTO l'albero e non ha trovato alcuna soluzione (rifiuta) – l'albero delle computazioni può avere un numero di nodi esponenziale rispetto alla sua profondità

Algorimi nondeterministici

Un modo più intuitivo di definire algoritmi nondeterministici

Si aggiunge il comando choice(I) (I è un insieme)

- choice(I) sceglie nondeterministicamente un elemento da I
- costo di choice(I) è 1

Esempio

Un algoritmo nondeterministico per la soddisfacibilità di una formula f

a è array di valori booleani: a[i]=0 (a[i]=1) implica variabile i è falsa (vera)

```
for (int i=0; i < n; i++)          (applico choice a ciascuna variabile)
      a[i]=choice({0,1});
if (value(f,a)) (verifico se assegnazioni a[] rende vera la formula)
      then return 1;  (la formula è soddisfacibile)
      else return 0; (la formula non è soddisfacibile) }
```

- L'algoritmo ritorna 1 se esiste almeno una scelta che soddisfa la formula
- costo algoritmo nondeterministico: costo di choice(I) pari a 1; costo algoritmo polinomiale è dato dalla verifica se la formula è soddisfacibile (polinomiale)

Algorimi nondeterministici

Si aggiunge il comando choice(I) (I è un insieme)

- choice(I) sceglie nondeterministicamente un elemento dell'insieme I

Esempio

Un algoritmo nondeterministico per la colorazione di un grafo G di n nodi con k colori: a è array di valori booleani: a[i]=j implica colore nodo i il j-esimo colore

```
for (int i=0; i < n; i++)          (applico choice a ciascun nodo)
    a[i]=choice({1,2,…, k});  (scelgo un colore per il nodo i)
if (value(G,a)) (verifico che ogni coppia di nodi uniti da arco abbiano colori diversi
)
then return 1;  (il grafo G è colorabile con k colori)
else return 0; (il grafo G non è colorabile con k colori) }
```

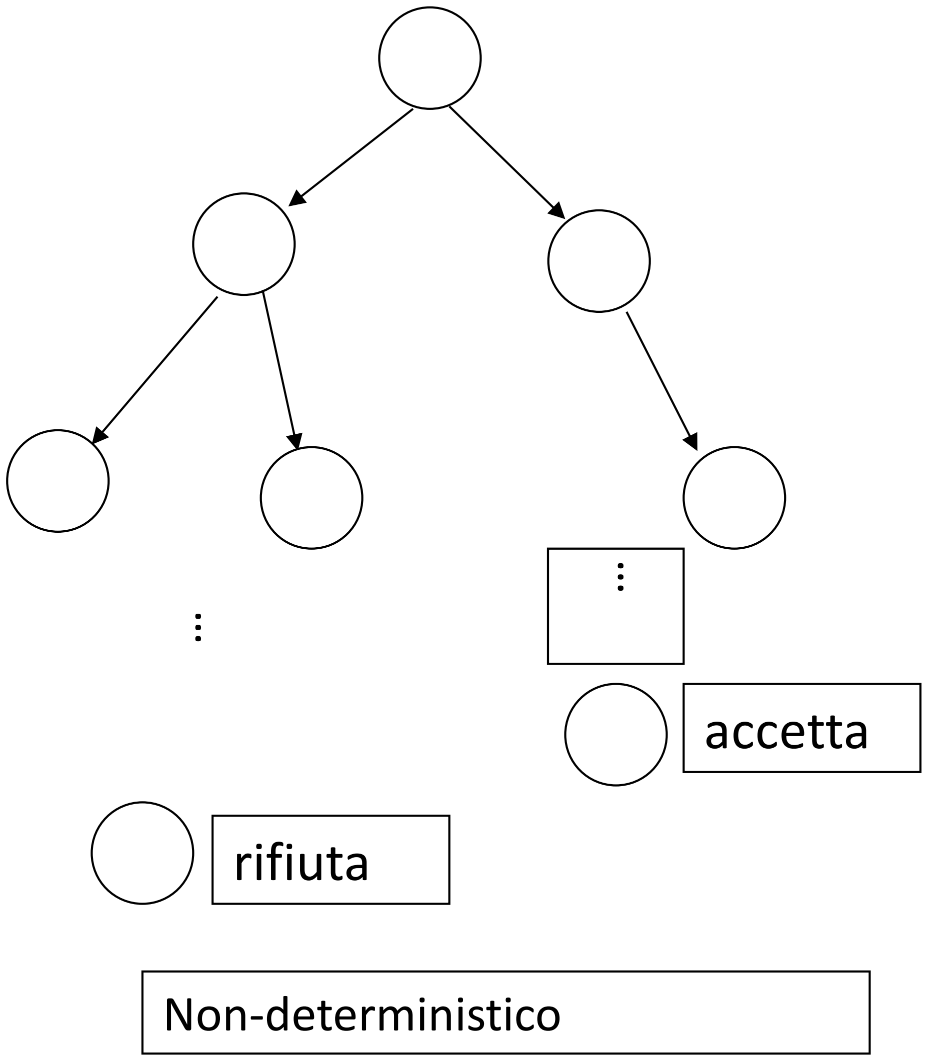
- L'algoritmo ritorna 1 se esiste almeno una scelta di colori ai nodi del grafo che colora propriamente il grafo (nodi adiacenti hanno colori diversi)
- Costo algoritmo nondeterministico :
  n (oper. choice + costo per verificare la colorazione)

Albero di computazione Non-deterministico

- Una macchina di Turing Non deterministica ha molti cammini di esecuzione
- Ciascun cammino corrisponde a possibili scelte di choice(I)

Le due condizioni seguenti sono equivalenti

- La macchina non deterministica accetta se almeno un cammino accetta
- Se una scelta di choice(I) porta a stato accetta



La classe NP

- La classe P è l'insieme di tutti i linguaggi L per cui esiste una Macchina di Turing deterministica che con input x in tempo polinomiale decide se x appartiene o meno a L
- La classe NP è l'insieme di tutti i linguaggi L per cui esiste una macchina di Turing nondeterministica che in tempo polinomiale decide se x appartiene o meno a L (cioè nell'albero delle computazioni della macchina esiste almeno un percorso dalla radice ad un nodo che termina e accetta)

La classe P è inclusa nella classe NP (una macchina di Turing deterministica è un caso particolare di una macchina di Turing non deterministica)

Esempio: Macchina di Turing non deterministica che risolve SAT

Nota: Il nome NP deriva da **N**ondeterministic **P**olynomial time.

La classe NP

Definizione alternativa

La classe NP è l'insieme di tutti i linguaggi L per cui - se x appartiene a L – allora esiste un certificato c(x) tale che

- c(x) ha lunghezza polinomiale in |x|
- Esiste una Macchina di Turing deterministica che con input x e c(x) verifica che x appartiene a L in tempo polinomiale in |x| e |c(x)|(lunghezza di x e di c(x))

Nota: non si richiede un certificato quando x NON appartiene a L

Nota: le due definizioni sono equivalenti:
Le decisioni nondeterministiche della macchina nondeterm. (def. 1) che '*indovina*' dove andare sono come il certificato (usato nella definizione 2) che "*informa*" la MdT deterministica che verifica la soluzione su quale scelta fare

Ma per quale c… di motivo devo imparare due definizioni equivalenti??

La classe NP

- La prima definizione è diretta facendo riferimento ad una macchina di Turing (o un algoritmo non deterministico)
- La seconda definizione evidenzia il fatto che per i problemi nella classe NP la verifica di una soluzione è facile (polinomiale). Il problema di trovare la soluzione (o dimostrare che una soluzione non esiste) è un compito più difficile

- Le scelte nondeterministiche della macchina di Turing corrispondono alle scelte della primitiva choice(I). Infatti
  - La MdT non deterministica accetta se esiste un cammino accettante
  - Il programma con choice(I) accetta se esiste una scelta di choice che accetta

La classe NP

La classe P è l'insieme di tutti i linguaggi L per cui esiste un algoritmo che in tempo polinomiale decide se x appartiene o meno a L

La classe NP è l'insieme di tutti i linguaggi L per cui – quando x appartiene a L – allora esiste un certificato c(x) tale che

- c(x) ha lunghezza polinomiale in |x|
- c(x) certifica che x appartiene a L

Nota: non si richiede un certificato che certifichi quando x NON appartiene a L

La classe P è inclusa nella classe NP (anche con la seconda definizione)

Ovviamente se L appartiene a P abbiamo un certificato

Infatti la sequenza di tutte le istruzioni dell'algoritmo polinomiale che verifica se x appartiene o no a L certifica

- *Se x appartiene a L*
- *Se x non appartiene a L*

La classe NP

Formalmente
La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- $|c(x)|$ (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e $|c(x)|$ certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in $|x|$ e $|c(x)|$ (lunghezza di x e di c(x))

In molti casi il certificato rappresenta la soluzione del problema stesso

Esempi di certificato

- SAT : certificato è assegnazione di valori Vero/Falso alle variabili
- Dato un grafo esiste un percorso da u a v lungo al massimo h archi : certificato sequenza di k archi ($k \leq h$) che collega u a v)
- Colorazione di un grafo con k colori: certificato è un'assegnazione di uno fra k colori a ciascun nodo del grafo

SAT appartiene a NP

Formalmente

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- $|c(x)|$ (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e $|c(x)|$ certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in $|x|$ e $|c(x)$

Esempi di certificato (problema x :: certificato c(x))

- SAT :: assegnazione di valori Vero / Falso alle variabili

Algoritmo per decidere se x è soddisfacibile

1. leggi sul certificato i valori delle variabili e determina un'assegnazione A(x) di valori vero/falso alle variabili
2. se A(x) soddisfa la formula
   allora x è soddisfacibile
   altrimenti x non è soddisfacibile

Colorazione di un grafo appartiene a NP

Formalmente

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- |c(x)| (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e |c(x)| certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in |x| e |c(x)

Esempi di certificato (problema x :: certificato c(x))

- Colorazione :: assegnazione di 1 fra k colori ai nodi del grafo

Algoritmo per decidere se un grafo G è colorabile con k colori

1. leggi sul certificato i colori dei nodi del grafo
2. Per ogni arco (a,b) del grafo verifica se a e b hanno colori diversi
3. Se la condizione 2 è verificata per ogni arco

   allora il grafo è k colorabile altrimenti non lo è

Il problema dello zaino appartiene a NP

Problema dello zaino (come problema di decisione9

- dato un insieme di oggetti ciascuno con un peso ed un valore voglio sapere se posso inserire in uno ziano un insieme di oggetti che pesano al max 20 kg e che mi danno un profitto pari almeno 30.

Esempi di certificato (problema x :: certificato c(x))

- Certificato per il problema dello zaino:: lista che indica per ogni oggetto se è da prendere o no

Algoritmo per decidere se -- con il certificato - il problema di decisione dello zaino è risolubile in tempo polinomiale

1. Leggo certificato
2. Costruisco insieme di oggetti S così come indicato nel certificato
3. Verifico se S ha peso compelssivo ≤20 e profitto ≥ 30

La classe NP

Formalmente

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione $c(x)$ e un polinomio p per cui x se x appartiene a L allora

- $|c(x)|$ (lunghezza di $c(x)$) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di $c(x)$ è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e $|c(x)|$ certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in $|x|$ e $|c(x)|$

Nota: l'esistenza di un certificato di appartenenza (x in L) non implica che esista un certificato che dimostra che x NON appartiene a L

Nota: nella definizione al posto di una macchina di Turing deterministica si può sostituire un algoritmo oppure un programma scritto in un qualunque linguaggio di programmazione

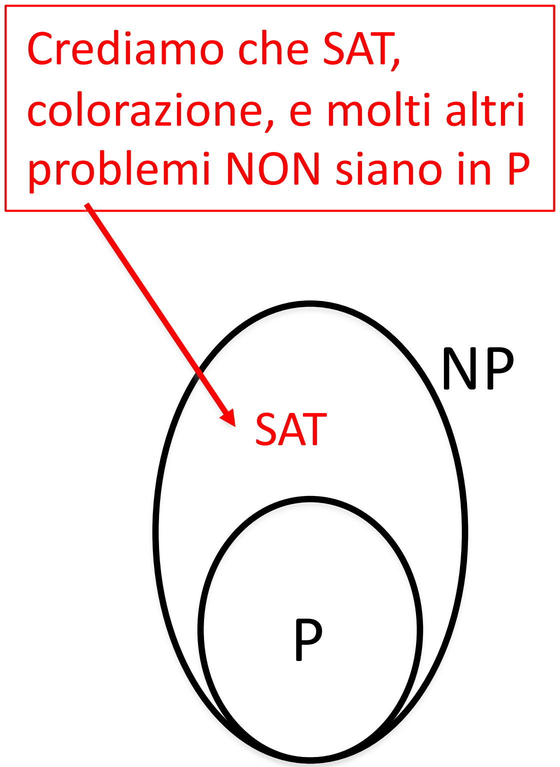
Le classi P e NP sono disgiunte?

Abbiamo visto che la classe P è contenuta nella classe NP

DOMANDA: La classe P è uguale alla classe NP? O la classe P è inclusa proriamente nella classe NP?

RISPOSTA: non lo sappiamo, ma congetturiamo che la risposta sia NO (problema da 1 milione di dollari)

- Informalmente, tale problema è equivalente a chiedersi se trovare una soluzione di un problema è, in generale, più difficile che verificare se una soluzione proposta è corretta.
- Chiunque si sia posto una simile domanda sa che la risposta più naturale è "la classe P è inclusa propriamente nella classe NP": per questo motivo, il problema è stato quasi sempre presentato come quello di trovare la dimostrazione della congettura $P \neq NP$ che afferma che la classe P è diversa dalla classe NP.

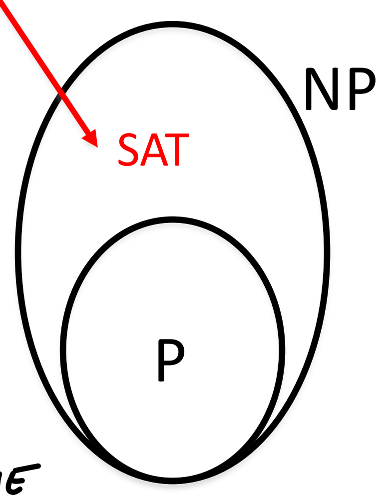


Le classsi P e NP sono disgiunte?

Abbiamo visto che la classe P è contenuta nella classe NP
DOMANDA: La classe P è uguale alla classe NP? O la classe P è inclusa proriamente nella classe NP?
RISPOSTA: non lo sappiamo, ma congetturiamo che la risposta sia NO (problema da 1 milione di dollari)

Vediamo ora una motivazione forte per cui crediamo (congetturiamo) che
la classe P sia inclusa propriamente nella classe NP (P≠ NP)

Crediamo che SAT, colorazione, e molti altri problemi NON siano in P



Per questo scopo introduciamo

- la classe dei problemi NP-completi

PER DIMOSTRARE CHE $P \neq NP$ BISOGNA TROVARE UN PROBLEMA CHE SI TROVA STRETTAMENTE IN NP.
PER FARE CIÒ SERVE TROVARE ALMENO UN PROBLEMA CON LOWER-BOUND PIÙ CHE POLINOMIALE PER UNA MACCHINA DETERMINISTICA
MA PER ORA NON È STATO TROVATO

Linguaggi NP-completi

Riduzione fra linguaggi (definizione vista nello studio di indecidibilità)

Intuitivamente, tale tecnica consiste nel dimostrare che,

- dati due linguaggi L1 e L2, L1 non è più difficile di L2 o, più precisamente, che
- se esiste una macchina di Turing che decide L2, allora esiste anche una macchina di Turing che decide L1.

Un linguaggio L1 è riducibile a un linguaggio L2 se esiste una funzione totale calcolabile f : $\{0,1\}^* \rightarrow \{0,1\}^*$, detta riduzione, tale che, per ogni stringa binaria x, x appartiene a L1 se e solo se f(x) appartiene a L2

Per studiare la classe P e NP ci limitiamo a riduzioni polinomiali

Un linguaggio L1 è **polinomialmente** riducibile a un linguaggio L2 se esiste una riduzione f da L1 a L2 che sia calcolabile da un algoritmo con complessità temporale polinomiale.

Linguaggi NP-completi

DOMANDA: La classe P è diversa da NP?

Definizione Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Intuitivamente, un linguaggio NP-completo è tra i più difficili della classe NP, nel senso che se appartenesse alla classe P, allora l'intera classe NP sarebbe inclusa nella classe P.

Teorema SAT è NP-completo.

Prova: sappiamo che SAT appartiene a NP.

La dimostrazione che ogni altro linguaggio in NP è riducibile a SAT è complessa.

La prova mostra che - data una stringa x - esiste una formula di SAT che è soddisfacibile se e solo se x appartiene a L

Linguaggi NP-completi

Teorema SAT è NP-completo.

Prova: la prova che ogni altro linguaggio in NP è riducibile a SAT è complessa (e non la vediamo in dettaglio).

Dato un linguaggio L in NP, sappiamo che esiste un algoritmo con complessità temporale polinomiale V e un polinomio p tali che, per ogni stringa x,

- se x è in L, allora esiste y (il certificato c(x) di lunghezza polinomiale in $|x|$) per cui V con x e y in ingresso termina e accetta
- se x non appartiene a L, allora, per ogni sequenza y, V con x e y in ingresso termina in uno stato non finale (non accetta)

L'idea della dimostrazione: costruire, per ogni x, in tempo polinomiale una formula booleana F(x,y) le cui uniche variabili da decidere sono $p(|x|)$ variabili $y_0, y_1, \ldots, y_{p(|x|)-1}$, il cui valore è calcolato usando il certificato e mostrando che la soddisfacibilità della formula dipende solo dai valori assegnati a tali variabili

Linguaggi NP-completi

DOMANDA: La classe P è diversa da NP?

Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.
Intuitivamente, un linguaggio NP-completo è tra i più difficili della classe NP, nel senso che se appartenesse alla classe P, allora l'intera classe NP sarebbe inclusa nella classe P.

Perché crediamo che P≠NP

- ci sono migliaia di problemi NP-completi in (Biologia, Fisica, Matematica, logistica, informatica, economia, …)
- se dimostriamo che un problema NP-completo sia risolubile in tempo polinomiale allora P=NP (quindi tutti i problemi in NP sono risolubili in tempo polinomaile)
- Per nessun problema NP completo conosciamo un algoritmo polinomiale

Linguaggi NP-completi

Un linguaggio L è **NP-completo** se

1) L appartiene a NP e
2) se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Se vale solo 1) allora il linguaggio appartiene a NP.

Se vale solo 2) allora ho la seguente definizione

Un linguaggio L è **NP-difficile (NP-hard)** se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Intuitivamente, un linguaggio NP-completo è tra i più difficili della classe NP, nel senso che se appartenesse alla classe P, allora l'intera classe NP sarebbe inclusa nella classe P.

Linguaggi NP-completi

Come provo che un linguaggio L che appartiene a NP è NP-completo?

- Mostrare che L può simulare una qualunque MdT non deterministica con tempo polinomiale (MOLTO DIFFICILE)
- Mostrare **una riduzione polinomiale** da un linguaggio NP-completo (ad esempio SAT) a L (PIÙ FACILE):

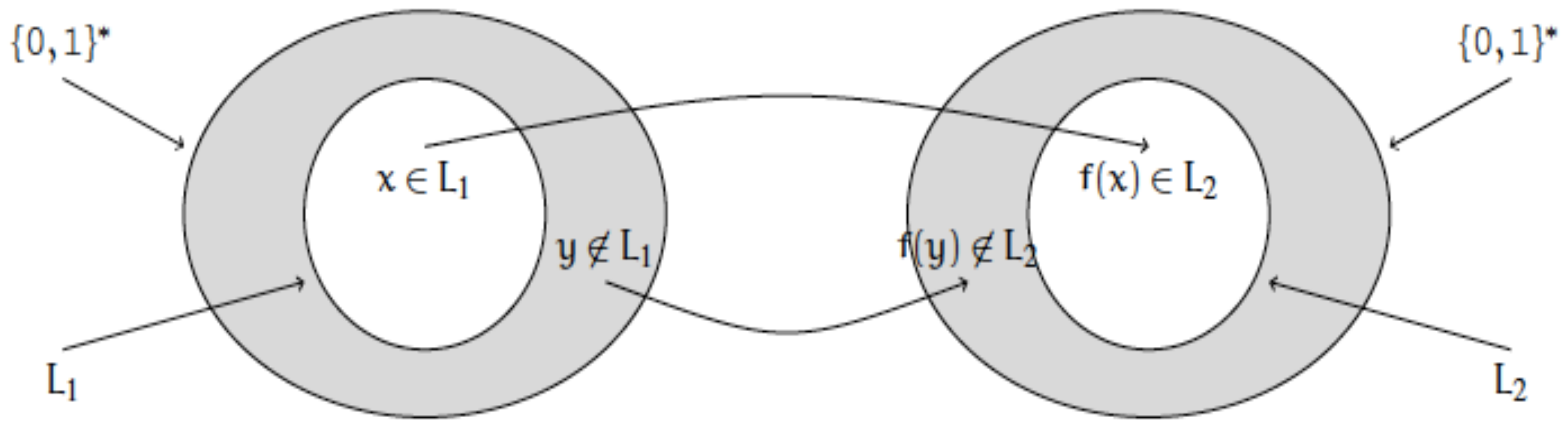
Il concetto di riduzione (visto nella calcolabilità) deve essere opportunamente modificato nel nuovo contesto.

Riduzione per mostrare indecidibilità (già viste)

Un linguaggio L1 è riducibile a un linguaggio L2 se

- esiste una funzione totale **calcolabile** f : {0,1}* →{0,1*, detta **riduzione**, tale che, per ogni stringa binaria x, x ε L1 se e solo se f(x) ε L2 (x in L1 se e solo se f(x) in L2)

Figura 3.4: riducibilità tra linguaggi



Siano L1 e L2 due linguaggi tali che L1 è riducibile a L2.

1- Se L2 è decidibile, allora L1 è decidibile.

2- Se L1 non è decidibile, allora L2 non è decidibile.

Intuizione se L1 riducibile a L2 allora

- L1 non è più difficile di L2 (vedi 1 sopra)
- L2 è almeno tanto difficile quanto L1 (2 sopra)

Nota Bene

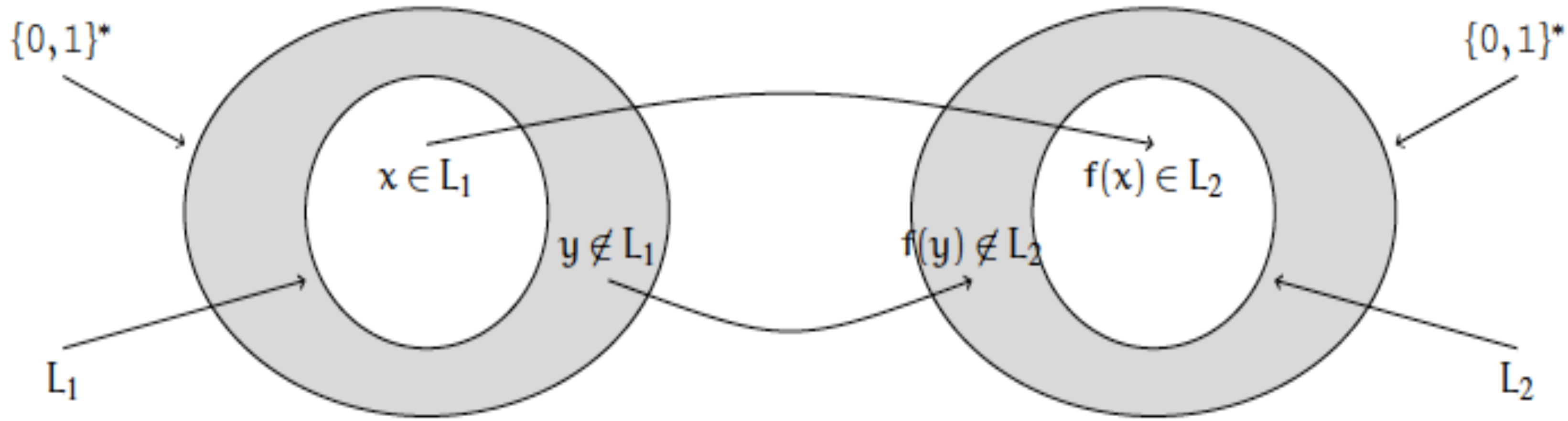
- la riduzione va da problema che so indecidibile a problema che voglio dimostrare indecidibile
- La riduzione nella direzione opposta non fornisce nulla (mi dice che A non è più facile di B ma non dice nulla su quanto sia difficile B)

Riduzione per mostrare NP-completezza (nuovo)

Un linguaggio L1 è riducibile a un linguaggio L2 se

- esiste una funzione totale calcolabile ***in tempo polinomiale*** f : {0,1}* →{0,1}*, detta **riduzione polinomiale**, tale che, per ogni stringa binaria x, x ε L1 se e solo se f(x) ε L2 (x in L1 se e solo se f(x) in L2)

Figura 3.4: riducibilità tra linguaggi



Siano L1 e L2 due linguaggi tali che L1 è riducibile a L2 con una riduzione polinomiale

1- Se L2 è risolubile in tempo polinomiale, allora L1 è risolubile in tempo polinomiale

2- Se L1 non è risolubile in tempo polinomiale, allora L2 non è risolubile in tempo polinomiale

se L1 riducibile a L2 allora L1 non è più difficile di L2 (vedi 1 sopra) e L2 è almeno tanto difficile quanto L1 (2 sopra)

Nota Bene

- la riduzione va da problema che so NP-completo a problema che voglio dimostrare NP-completo
- La riduzione nella direzione opposta non fornisce nulla (mi dice che A non è più facile di B ma non dice nulla su quanto sia difficile B)
- SAT clausole del tipo (x or y or z)

Riduzione polinomiale da SAT a 3SAT

Data una formula F con clausole con un numero di variabili fra 1 e k si definisce una formula F' con clausole di soli 3 variabili che è soddisfacibile se e solo se F è soddisfacibile

F è formata da un insieme di clausole e ciascuna clausola è formata da un insieme di letterali $l_0 \;\; l_1 \; l_2 \;\; \ldots. \; l_{k-1}$

F' modifica clausole e aggiunge nuove variabili diverse da $l_0 \;\; l_1 \; l_2 \;\; \ldots. \; l_{k-1}$, nel seguito $y_0 \;\; y_1 \; y_2 \;\; \ldots.$ *(sono variabili aggiuntive)*

- $k = 1$: in questo caso, (la clausola è del tipo l_0)

$$D_c = \{\{l_0, y_0^c, y_1^c\}, \{l_0, y_0^c, \neg y_1^c\}, \{l_0, \neg y_0^c, y_1^c\}, \{l_0, \neg y_0^c, \neg y_1^c\}\}$$

Osserviamo che le quattro clausole in D_c sono soddisfatte se e solo se l_0 è soddisfatto.

- $k = 2$: in questo caso, $D_c = \{\{l_0, l_1, y_0^c\}, \{l_0, l_1, \neg y_0^c\}\}$. Osserviamo che le due clausole in D_c sono soddisfatte se e solo se l_0 oppure l_1 è soddisfatto.

$$D_c = \{\{l_0, l_1, y_0^c\}, \{\neg y_0^c, l_2, y_1^c\}, \{\neg y_1^c, l_3, y_2^c\}, \ldots, \{\neg y_{k-4}^c, l_{k-2}, l_{k-1}\}\}$$

Riduzione polinomiale da SAT a 3SAT

Data una formula F con k clausole si definisce una formula F' che è soddisfacibile se e solo se F è soddisfacibile (modifica clausole e aggiunge variabili)

Una clausola con k letterali (k≠3) si trasforma nel seguente modo

- $k = 1$: in questo caso,

$$D_c = \{\{l_0, y_0^c, y_1^c\}, \{l_0, y_0^c, \neg y_1^c\}, \{l_0, \neg y_0^c, y_1^c\}, \{l_0, \neg y_0^c, \neg y_1^c\}\}$$

  Osserviamo che le quattro clausole in D_c sono soddisfatte se e solo se l_0 è soddisfatto.

- $k = 2$: in questo caso, $D_c = \{\{l_0, l_1, y_0^c\}, \{l_0, l_1, \neg y_0^c\}$. Osserviamo che le due clausole in D_c sono soddisfatte se e solo se l_0 oppure l_1 è soddisfatto.

- $k > 3$: in questo caso,

$$D_c = \{\{l_0, l_1, y_0^c\}, \{\neg y_0^c, l_2, y_1^c\}, \{\neg y_1^c, l_3, y_2^c\}, \ldots, \{\neg y_{k-4}^c, l_{k-2}, l_{k-1}\}\}$$

Riduzione polinomiale da SAT a 3SAT

Una clausola con k letterali (k≠3) si trasforma nel seguente modo

- $k > 3$: in questo caso,

$$D_c = \{\{l_0, l_1, y_0^c\}, \{\neg y_0^c, l_2, y_1^c\}, \{\neg y_1^c, l_3, y_2^c\}, \ldots, \{\neg y_{k-4}^c, l_{k-2}, l_{k-1}\}\}$$

- Supponiamo che esista un'assegnazione di τ verità alle variabili di X che soddisfa C. msotriamo che esiste assegnazione che soddisfa D
- Quindi, τ soddisfa c per ogni clausola di C: mostriamo che tale assegnazione può essere estesa alle nuove variabili di tipo y^c introdotte in modo che tutte le clausole in $D_c$siano soddisfatte
- Poiché c è soddisfatta da τ , deve esistere h tale che τ soddisfa l_h con $0 \leq i \leq h-1$
- Estendiamo τ assegnando il valore true a tutte le variabili $y^c{}_i$ con $0 \leq i \leq h-2$ e il valore false alle rimanenti variabili di tipo y^c
- In questo modo, siamo sicuri che la clausola di D_c contenente l_h è soddisfatta (da l_h stesso), le clausole che la precedono sono soddisfatte grazie al loro terzo letterale e quelle che la seguono lo sono grazie al loro primo letterale.

Riduzione polinomiale da SAT a 3SAT

Una clausola con k letterali (k≠3) si trasforma nel seguente modo

- $k > 3$: in questo caso,

$$D_c = \{\{l_0, l_1, y_0^c\}, \{\neg y_0^c, l_2, y_1^c\}, \{\neg y_1^c, l_3, y_2^c\}, \ldots, \{\neg y_{k-4}^c, l_{k-2}, l_{k-1}\}\}$$

- Viceversa, supponiamo che esista un'assegnazione di verità alle variabili di Z che soddisfi tutte le clausole in D e, per assurdo, che tale assegnazione ristretta alle sole variabili di X non soddisfi almeno una clausola c di C, ovvero che tutti i letterali contenuti in c non siano soddisfatti (di nuovo, ipotizziamo che $|c| > 3$).
- Ciò implica che tutte le variabili di tipo y_c devono essere vere, perché altrimenti una delle prime $|c| - 3$ clausole in D_c non è soddisfatta, contraddicendo l'ipotesi che soddisfa tutte le clausole in D.
- Quindi, $\tau(y^c_{|c|-4}) = \text{true}$, ovvero $\tau(\text{not } y^c_{|c|-4}) = \text{false}$: poiché abbiamo supposto
- che anche $l_{|c|-2}$ e $l_{|c|-1}$ non sono soddisfatti, l'ultima clausola in D_c non \`e soddisfatta,
- contraddicendo nuovamente l'ipotesi che soddisfa tutte le clausole in D

Programmazione a numeri interi

La programmazione a numeri interi (PI) è un fondamentale pro blema di ottimizzazione. Possiamo pensare a una programmazione lineare con variabili limitate ad assumere solo valori interi

Input: un insieme V di variabili intere, un insieme di disuguaglianze su V, una funzione obiettivo di massimizzazione f(V) e un intero B.

Output: esiste un attributo di interi a V tale che tutte le disuguaglianze siano vere e $f(V) \geq B$?

Esempio

due variabili v_1 e v_2 con valori interi e i vincoli

$v_1 \geq 1$, $v_2 \geq 1$, $v_1 + v_2 \leq 3$

Funzione obiettivo $f(V) = v_1 + 2\, v_2$ B=5

Una soluzione del problema è $v_1 = 1$, $v_2 = 2$

Se poniamo B=6 non abbiamo soluzione (max valore f(V) compatibile con i vincoli è 5)

Riduzione da 3-SAT a programmazione intera

Dimostriamo che la programmazione di interi è difficile usando una riduzione da 3-SAT. Per questa particolare riduzione, il problema generale funzionerebbe altrettanto bene, anche se in questo utilizzare 3-SAT facilita la riduzione.

In che direzione deve andare la riduzione?
Vogliamo dimostrare che la programmazione di interi è difficile, e sappiamo che 3-SAT è difficile.
Se potessi risolvere il 3-SAT usando la programmazione di interi e la programmazione di interi fosse facile, ciò significherebbe che 3-SAT sarebbe facile.

Quindi dobbiamo "tradurre" 3-SAT nella programmazione a numeri interi.

Riduzione da 3SAT a programmazione intera

Ogni istanza di 3-SAT contiene variabili booleane (vero / falso).
Ogni istanza di programmazione intera contiene variabili intere (valori ristretti a 0,1,2 ..) e vincoli.
Nel seguito le variabili intere corrispondono alle variabili booleane

Data una formula con n variabili per ogni variabile logica x_i (che può assumere solo i valori vero o falso) del problema 3SAT abbiamo due variabili v_i e w_i
Per limitare ogni variabile di programmazione intera a valori di 0 o 1, aggiungiamo i seguenti vincoli $0 \leq v_i \leq 1$ e $0 \leq w_i \leq 1$ per ogni i

Intuitivamente $v_i = 1$ e $w_i = 0$ rappresentano l'assegnazione x_i =vero mentre $w_i = 1$ e $v_i = 0$ rappresentano l'assegnazione x_i =falso

Aggiungiamo i vincoli per ogni i
$1 \leq v_i + w_i \leq 1$ che impone che al massimo una fra v_i e w_i sia 1 (equivalgono a $v_i + w_i = 1$)

Riduzione da 3SAT a programmazione intera

Data una formula con n variabili per ogni variabile x_i del problema 3SAT abbiamo due variabili v_i e w_i . $v_i = 1$ e $w_i = 0$ rappresentano l'assegnazione x_i =vero mentre $w_i = 1$ e $v_i = 0$ rappresentano l'assegnazione x_i =falso

Aggiungiamo i vincoli per ogni i
$1 \leq v_i + w_i \leq 1$ che impone che al massimo una fra v_i e w_i sia 1

Per ogni clausola nell'istanza 3-SAT definiamo un vincolo.
Ad esempio data la clausola C= (x_1 or x_2 or x_3) definiamo il vincolo
$v_1 + v_2 + v_3 \geq 1$ che impone che 1 fra le variabili x_1, x_2 , x_3 sia vera

Analogamente per la clausola C= (x_1 or (not x_2) or (not x_5)) definiamo il vincolo
$v_1 + w_2 + w_5 \geq 1$ che impone che o x_1 è vera oppure una fra le variabili x_2 , x_3 è falsa

Per soddisfare il vincolo, almeno uno dei letterali della clausola deve essere impostato su 1, quindi a un letterale vero.

Riduzione da 3SAT a programmazione intera

Per stabilire che questa riduzione sia corretta dobbiamo verificare due cose:

- Qualsiasi soluzione SAT fornisce una soluzione al problema IP

Infatti data una qualsiasi soluzione SAT, un valore letterale vero corrisponde ad una variabile posta a 1 nel programma intero, poiché la clausola è soddisfatta.
Pertanto, la somma in ogni vincolo della clausola è almeno 1

- Qualsiasi soluzione IP fornisce una soluzione SAT

In qualsiasi soluzione dell'istanza di programmazione intera, tutte le variabili devono essere impostate su 0 o su 1.
Se v_i =1, assumi x_i =vero; se w_i =1, assumi x_i =falso.
Dato che v_i e w_i non possono essere entrambi posti 1, quindi è un'assegnazione corretta alle variabili.
Inoltre per ogni clausola abbiamo che almeno un letterale deve essere vero.
La riduzione funziona in entrambi i modi, quindi la programmazione intera deve essere difficile.

Riduzione da 3SAT a programmazione intera

Per stabilire che questa riduzione sia corretta dobbiamo verificare due cose:

- Qualsiasi soluzione SAT fornisce una soluzione al problema IP

Infatti data una qualsiasi soluzione SAT, se x_i è vero allora $v_i = 1$ e $w_i = 0$ (se x_i è falso $v_i = 0$ e $w_i = 1$)

Questo soddisfa tutti i vincoli

- Facile verificare che $1 \leq v_i + w_i \leq 1$
- Poiché la formula è soddisfacibile ogni clausola è soddisfatta; supponi che una clausola sia soddisfatta se x_i è vera ; allorasi fissa $v_i = 1$ abbiamo esiste un vincolo del tipo $v_i + v_k + w_j \geq 1$ in (v_k w_j rappresentano gli altri letterali della formula)

Pertanto, la somma in ogni vincolo della clausola è almeno 1

Riduzione da SAT a PI: conclusioni

Notare le seguenti proprietà, che sono valide in generale per NP-complete:

- La riduzione ha preservato la struttura del problema. Ci ha permesso di formulare il problema SAT in un formato diverso.
- Le possibili istanze IP che possono risultare da questa trasformazione sono solo un piccolo sottoinsieme di tutte le possibili istanze IP. Tuttavia, poiché alcuni di essi sono difficili, il problema generale deve essere difficile.
- La trasformazione cattura l'essenza del perché IP è difficile. Non richiede grandi coefficienti e valori per le variabili 0/1 sono sufficiente. Non richiede un numero elevato di variabili o di vincoli.
- La programmazione a numeri interi è difficile perché soddisfare un insieme di vincoli è difficile quando richiediamo valori interi per le variabili. Infatti se eliminiamo il vincolo di interezza e permettiamo di assegnare valroi razionali alle variabili il problema diventa facile

Programmazione a numeri interi: conclusioni

Abbiamo mostrato una riduzione da 3-SAT a PI

Questa riduzione dimostra che PI è tanto difficile quanto SAT

Perché SAT è riducibile a 3-SAT e 3-SAR è riducibile a IP

Poiché le riduzioni sono del ‘se e solo se’

Questo implica che esiste una riduzoine da SAT a IP

Possiamo dire che PI sia NP-completo?

Programmazione a numeri interi: conclusioni

Abbiamo mostrato una riduzione da 3-SAT a PI

Questa riduzione dimostra che PI è tanto difficile quanto SAT

Possiamo dire con questo che PI sia NP-completo? **NO**

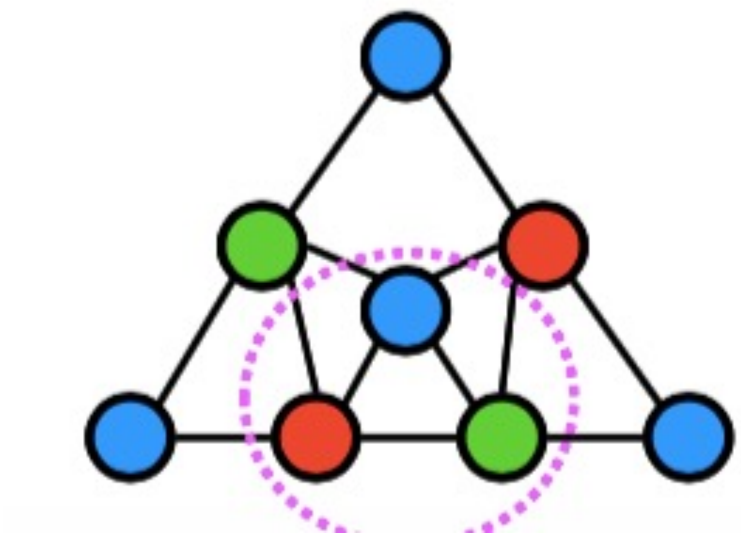
Dobbiamo ancora dimostrare che PI sia nella classe NP
Questa dimostrazioni non è semplice (non la vediamo) e permette di stabilire il seguente teorema

Teorema La programmazione a numeri interi è NP-completo

Colorazione di grafi

Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

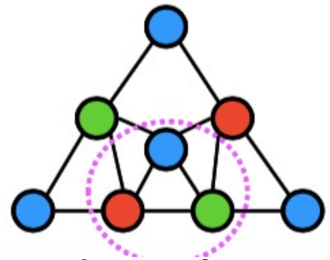
- Il numero cromatico di G, C(G) è il minimo numero di colori necessario per colorare tutti i suoi nodi
- Chiaramente se un grafo ha n nodi abbiamo che $1 \leq C(G) \leq n$
- Il problema di decisione associato è: dato un grafo G può essere colorato con k colori (k intero)?



La figura mostra una colorazione con 3 colori

Si noti che se tre nodi sono mutuamente adacenti e formano una cricca allora richiedono tre colori diversi (vedi nodi cerchiati a sinistra)

Colorazione di grafi



Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

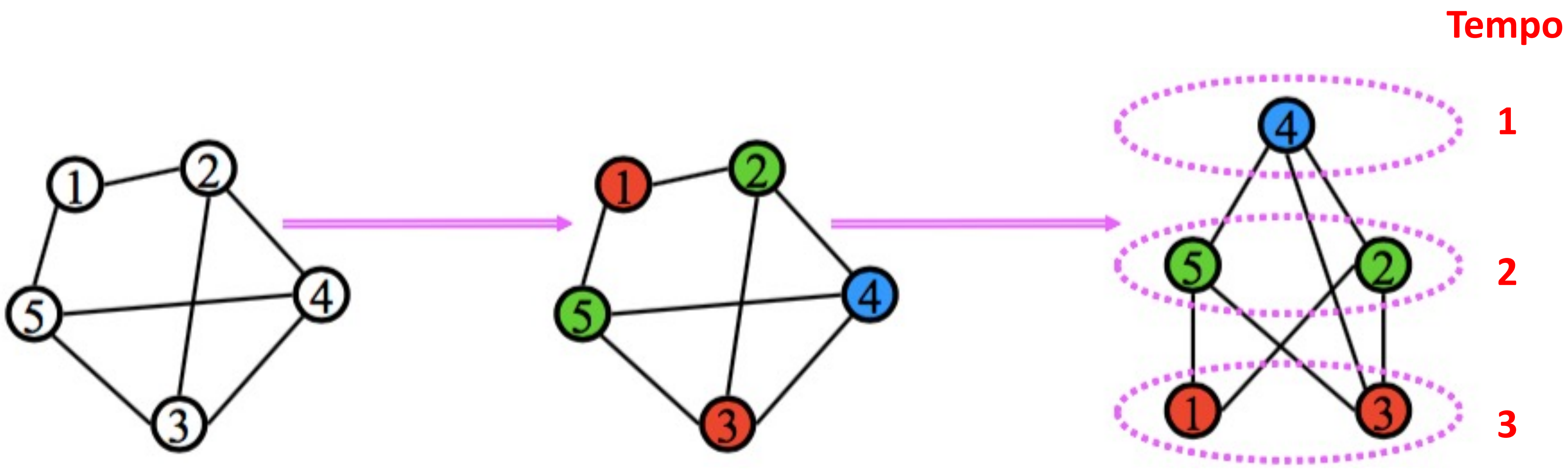
- Il problema di decisione associato è: dato un grafo G può essere colorato con k colori (k intero)?
- Il problema appartiene a NP
- Prova 1 utilizza come certificato C una colorazione dei nodi dato C è molto semplice (e realizzabile in tempo polinomiale) verificare che C rappresenti una colorazione corretta: Basta verificare che per ogni arco i nodi corrispondenti abbiano colori diversi
- Prova 2: fornire un algoritmo non deterministico

per ogni nodo del grafo v[i] (applico choice a ciascun nodo)
    v[i]=choice({1,…k});
per ogni arco del grafo (i,j) (verifica che i colori dei due nodi sono diversi)
    verifica che v[i] ≠ v[j]

Colorazione di grafi

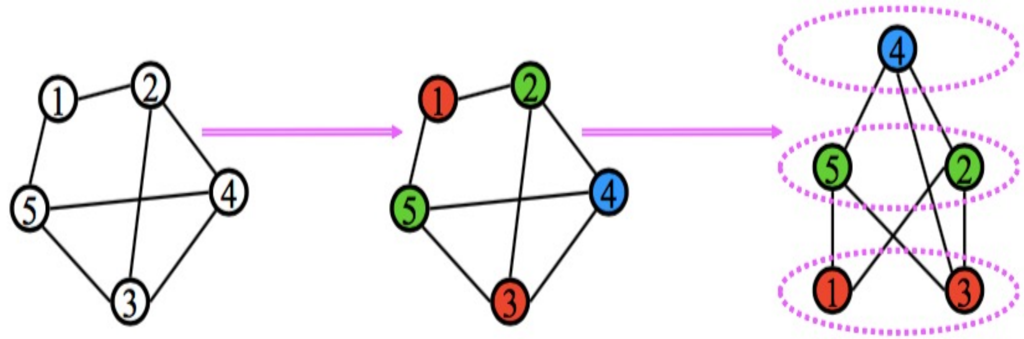
Esempio di applicazione: sequenziamento di lavori (scheduling)

- Dobbiamo assegnare lavori a intervalli di tempo
- Alcuni lavori sono in conflitto e non possono essere eseguiti insieme (ad esempio usano risorse condivise)
- Modella i lavori con i nodi di un grafo e i conflitti come archi
- Il minimo numero di colori usato rappresenta il minomo tempo di completamento (“makespan”) per finire i lavori



Colorazione di grafi

- Dato un grafo colorabile con tre colori esistono almeno 3! (=6) colorazioni del grafo. Infatti puoi permutare i 3 colori
- Dato un grafo colorabile con k colori esistono almeno k! colorazioni del grafo.
- Data una colorazione con k colori esiste una colorazione che
  - assegna a uno specifico nodo un colore prescelto
  - assegna a due nodi collegati uniti da un arco due colori specifici
- Dato un grafo completo di 3 nodi possiamo assumere che **due specifici nodi uniti da un arco** siano colorati uno verde e uno blu. L'altro necessariamente rosso. E' chiaro che anche se abbiamo 6 colorazione sono tutte equivalenti

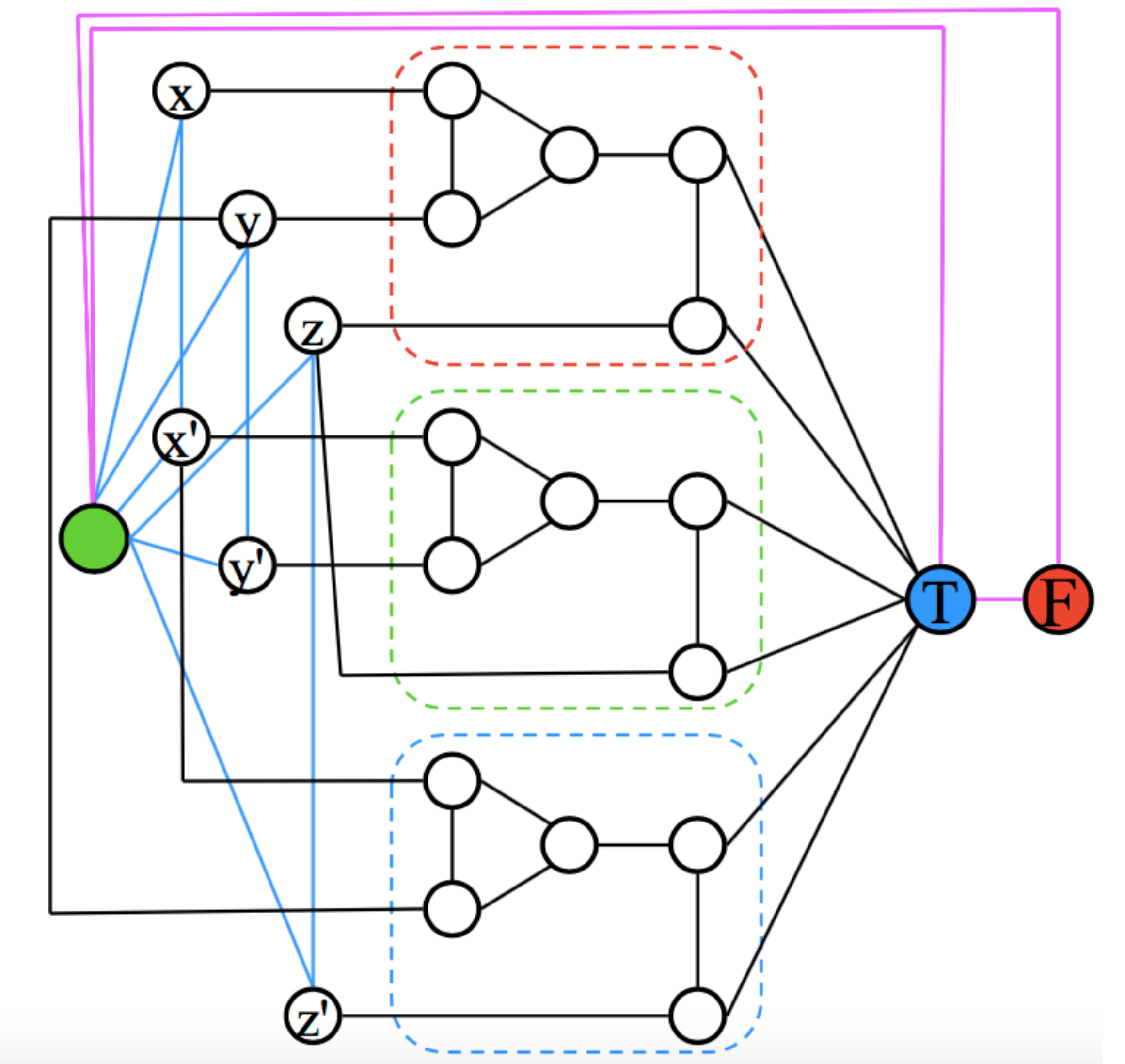


Riduzione da 3SAT a 3-colorazione di grafi

Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or not z)

Otteniamo il grafo grande a destra



NOTA

- I nodi colorati sono uniti da tre archi.
- Quindi possiamo assumere che siano colorati come in figura

Riduzione da 3SAT a 3-colorazione di grafi

Dimostriamo che decidere se un grafo è colorabile con 3 colori di interi è difficile usando una riduzione da 3-SAT.

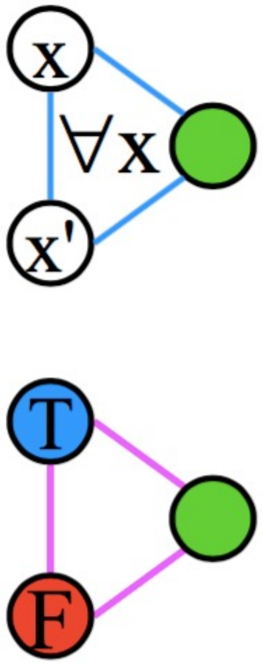
Idea costruire un sottografo che simula OR e realizza una clausola

- Tre colori: verde, blu (rappresenta T – vero), rosso (rapp. F- falso)

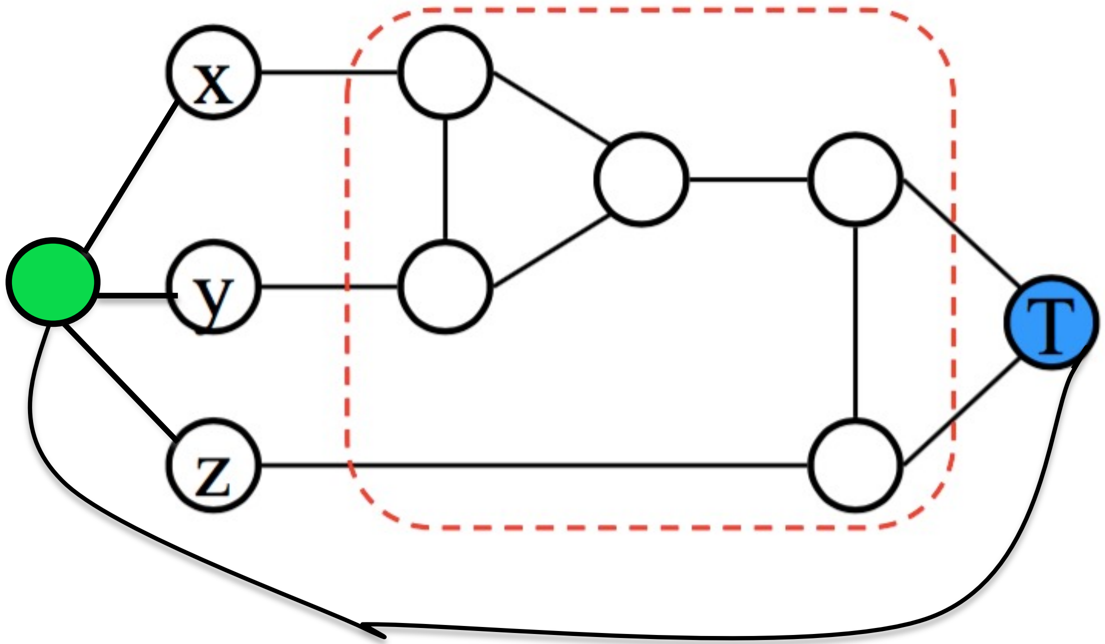
Per ogni variabile x abbiamo
(x' rappresenta not x)

Il grafo a destra mostra x e x' collegati a verde ➔
- x=T e x'=F

Oppure
- x=F e x'=T



Per ogni clausola del tipo (x or y or z) il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)



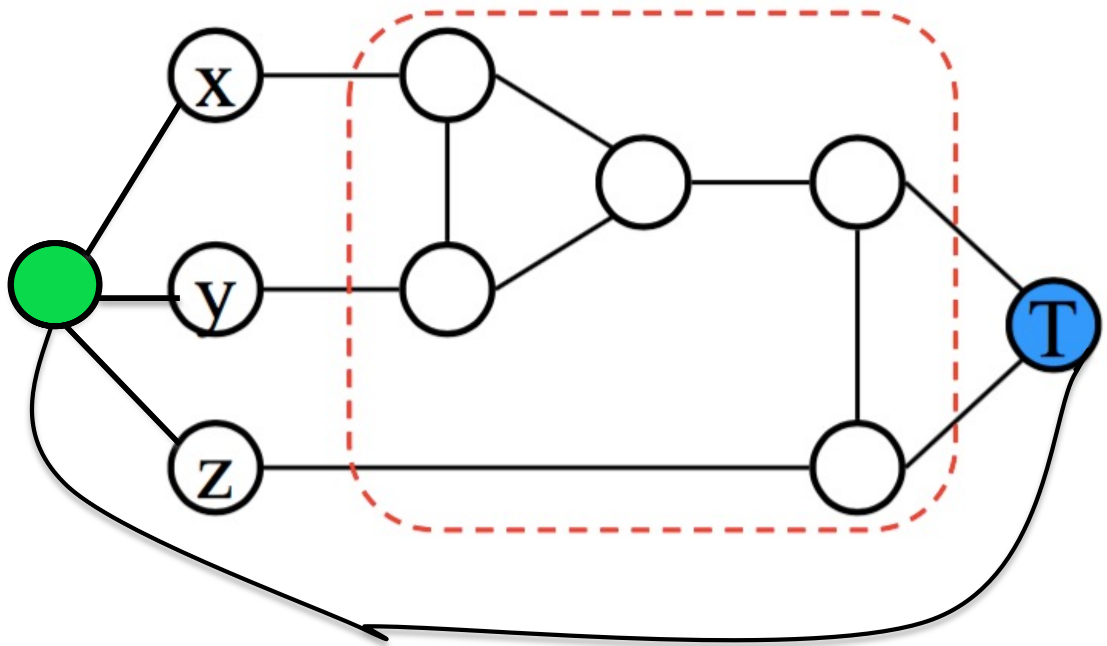
Riduzione da 3SAT a 3-colorazione di grafi

Dimostriamo che la decidere se un grafo è colorabile con 3 colori di interi è difficile usando una riduzione da 3-SAT.
Idea costruire un gadget che simula OR
Tre colori: verde, blu (rappresenta T – vero), rosso (rapp. F- falso)

Per ogni variabile x abbiamo
(x' rappresenta not x)

Nel seguito assumiamo che due nodi del grafo a destra siano colorati uno verde e l'altro blu
Vedremo alla fine come completare la prova verificando come il grafo finale costruito soddisfi questa ipotesi

Per ogni clausola del tipo (x or y or z) consideriamo il grafo seguente **NOTA** un nodo (ad es. X) compare tante volte quante volte la var. x compare in una clausola

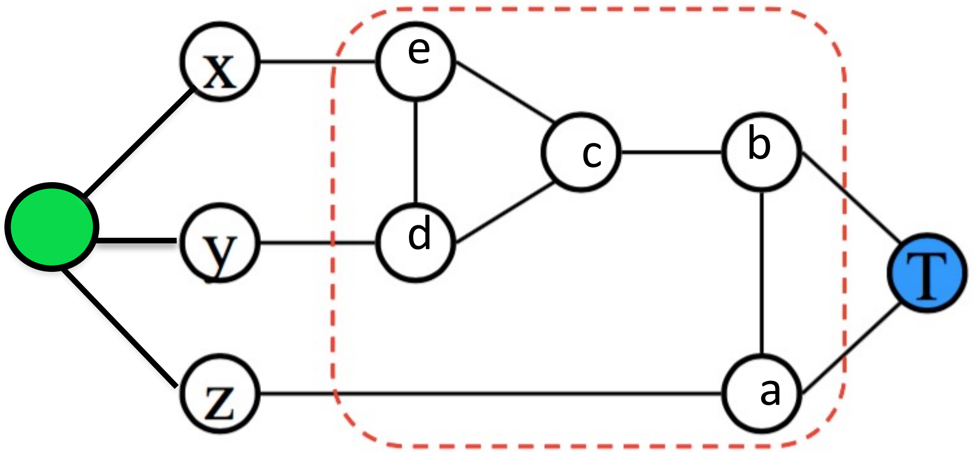


Riduzione da 3SAT a 3-colorazione di grafi

Tre colori: verde, blu (Rappresenta T - vero, rosso (rapp. F- falso)

Lemma Per ogni clausola (x or y or z) il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)

Prova: per contraddizione: esiste colorazione in cui x,y,z non sono blu; poiché sono adiacenti ad un nodo verde segue che x,y, z sono tutti colorati rosso



NOTA: la costruzione finale permette di assumere che i due nodi verde e blu sopra possano essere colorati con questi colori

1. z rosso implica le seguenti colorazioni:
- a= verde che quindi implica
- b= rosso

2. Ora nota che i nodi d,e,c

2.a Devono avere tre colori diversi

2.b Ognuno è adiacente ad un nodo rosso e quindi non possiamo colorarli rosso

2.c Pertanto d,e,c non possono essere colorati usando solo verde, rosso, blu

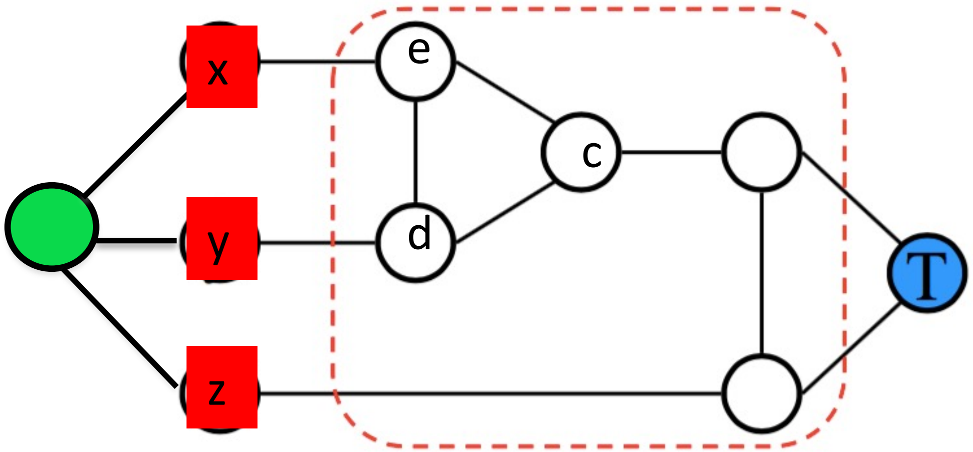
3. Quindi l’assunzione fatta che nessuno fra x,y,z sia blu è falsa

Riduzione da 3SAT a 3-colorazione di grafi

Tre colori: verde, blu (Rappresenta T - vero, rosso (rapp. F- falso)

Lemma Per ogni clausola (x or y or z) il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)

Prova: **per contraddizione: esiste colorazione in cui x,y,z non sono blu; poiché sono adiacenti ad un nodo verde segue che x,y, z sono tutti colorati rosso**



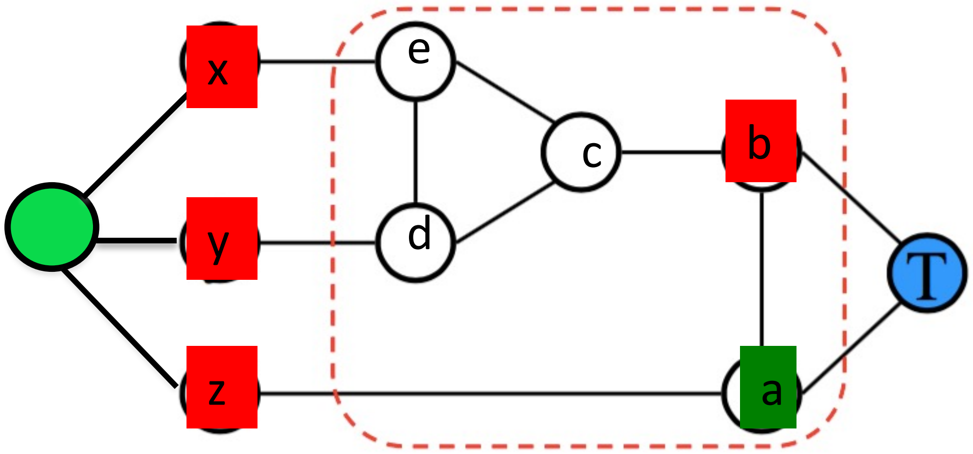
NOTA: la costruzione finale permette di assumere che i due nodi verde e blu sopra possano essere colorati con questi colori

Riduzione da 3SAT a 3-colorazione di grafi

Tre colori: verde, blu (Rappresenta T - vero, rosso (rapp. F- falso)

Lemma Per ogni clausola (x or y or z) il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)

Prova: per contraddizione: esiste colorazione in cui x,y,z non sono blu; poiché sono adiacenti ad un nodo verde segue che x,y, z sono tutti colorati rosso



NOTA: la costruzione finale permette di assumere che i due nodi verde e blu sopra possano essere colorati con questi colori

1. **z rosso implica le seguenti colorazioni:**
   - **a= verde che quindi implica**
   - **b= rosso**

2. Ora nota che i nodi d,e,c

2.a Devono avere tre colori diversi

2.b Ognuno è adiacente ad un nodo rosso e quindi non possiamo colorarli rosso

2.c Pertanto d,e,c non possono essere colorati usando solo verde, rosso, blu

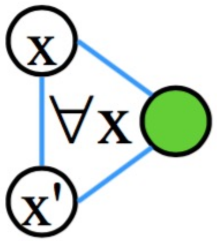
3. Quindi l'assunzione fatta che nessuno fra x,y,z sia blu è falsa

Riduzione da 3SAT a 3-colorazione di grafi

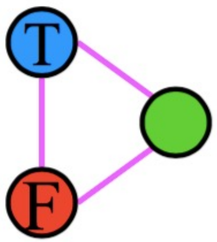
Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

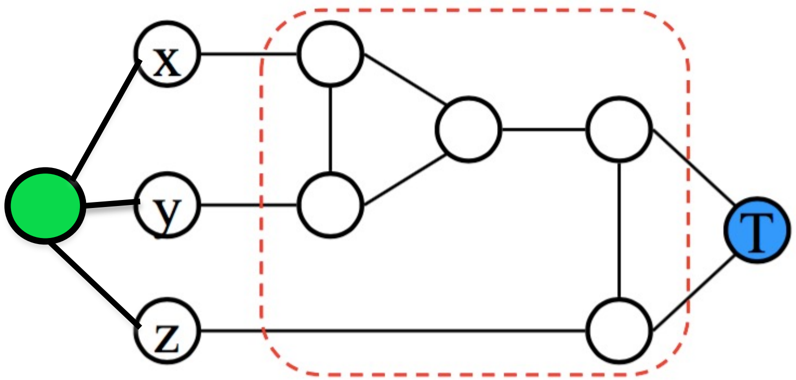
Otteniamo il grafo grande a destra

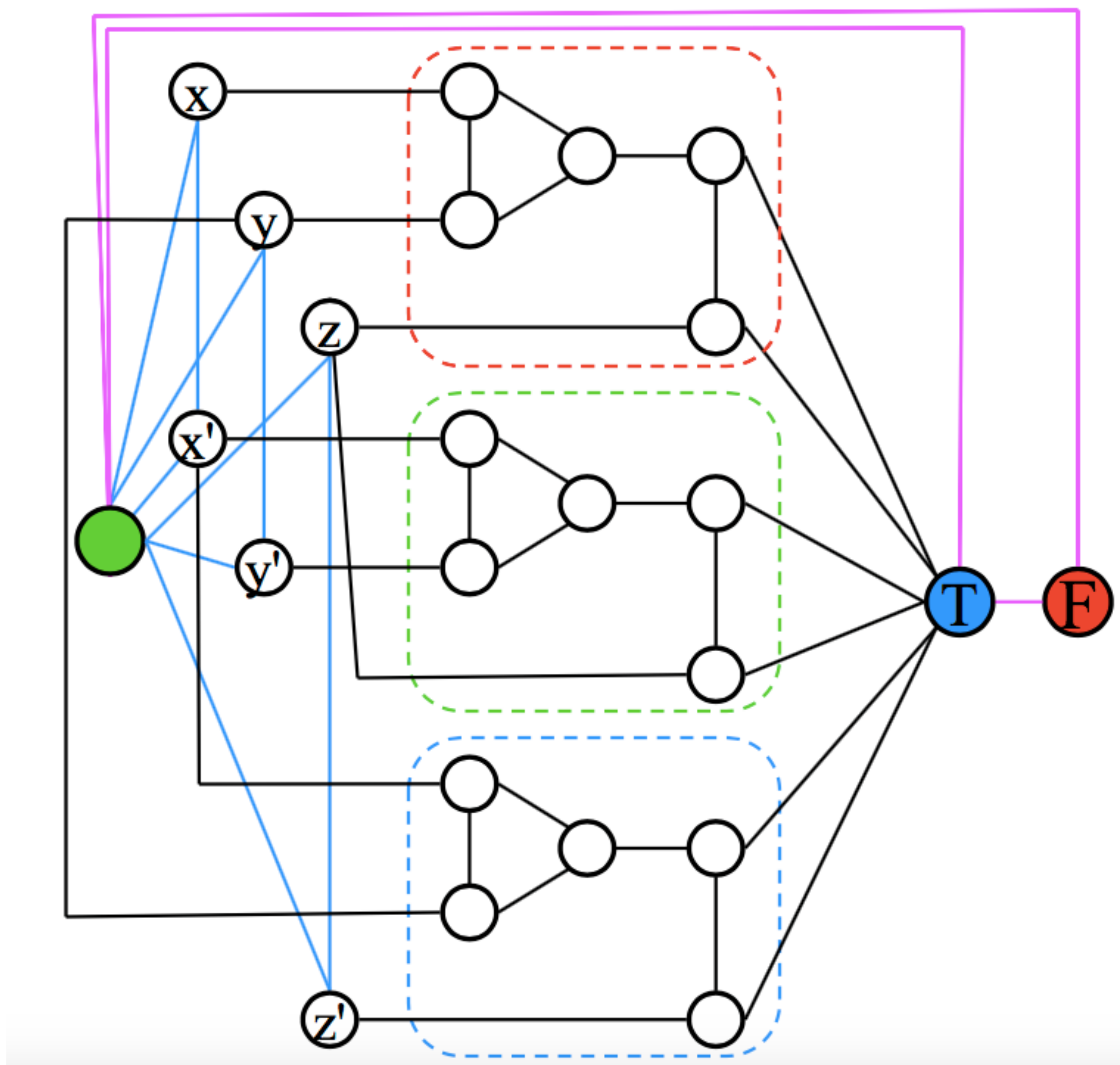


Impone che x e x’ abbiano rosso o blu valori diversi fra loro



Impone che almeno uno fra x,y,z sia blu (vero)



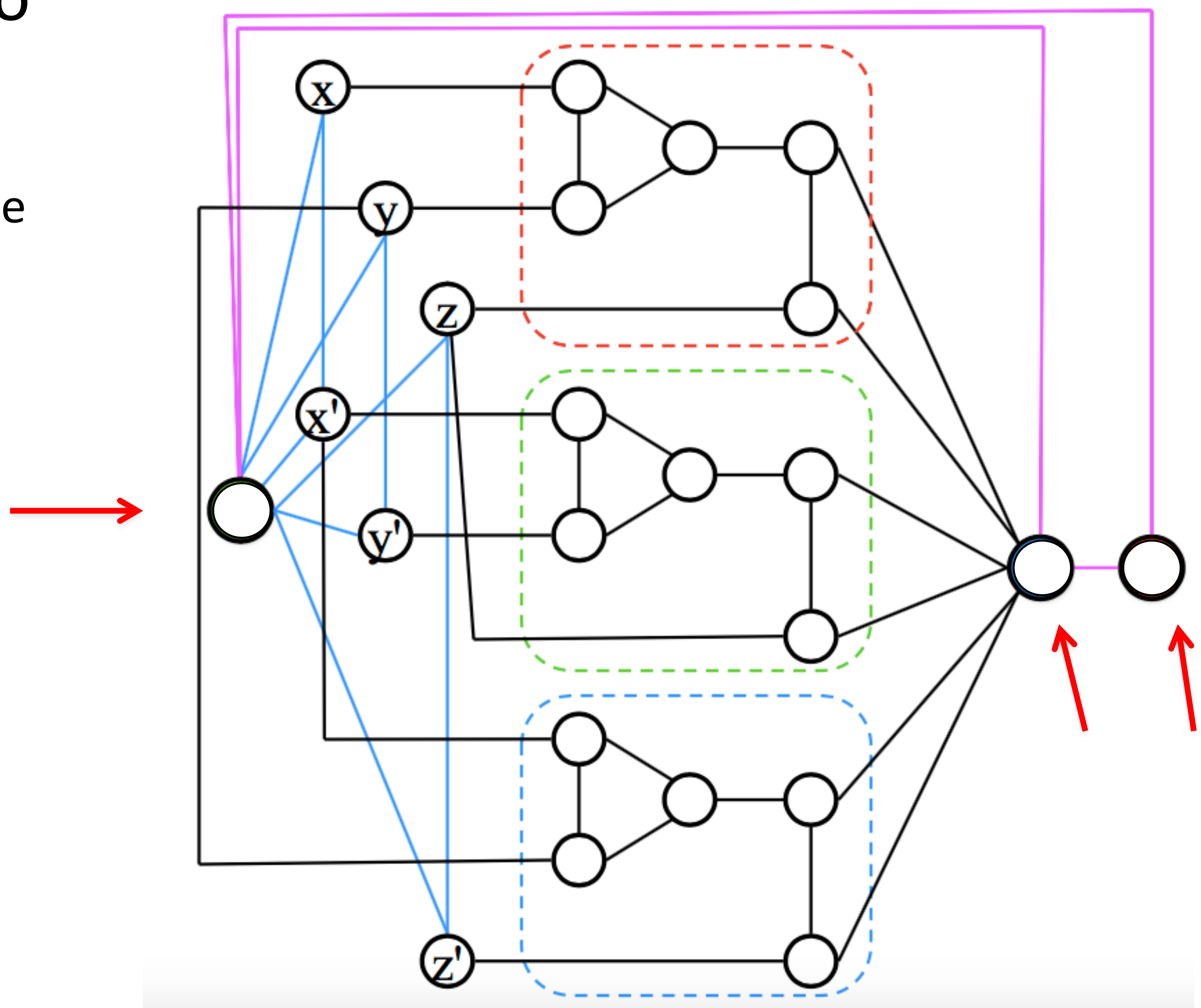


Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

Otteniamo il grafo

I tre nodi indicati da tre frecce sono mutuamente connessi quindi devono usare tre colori diversi



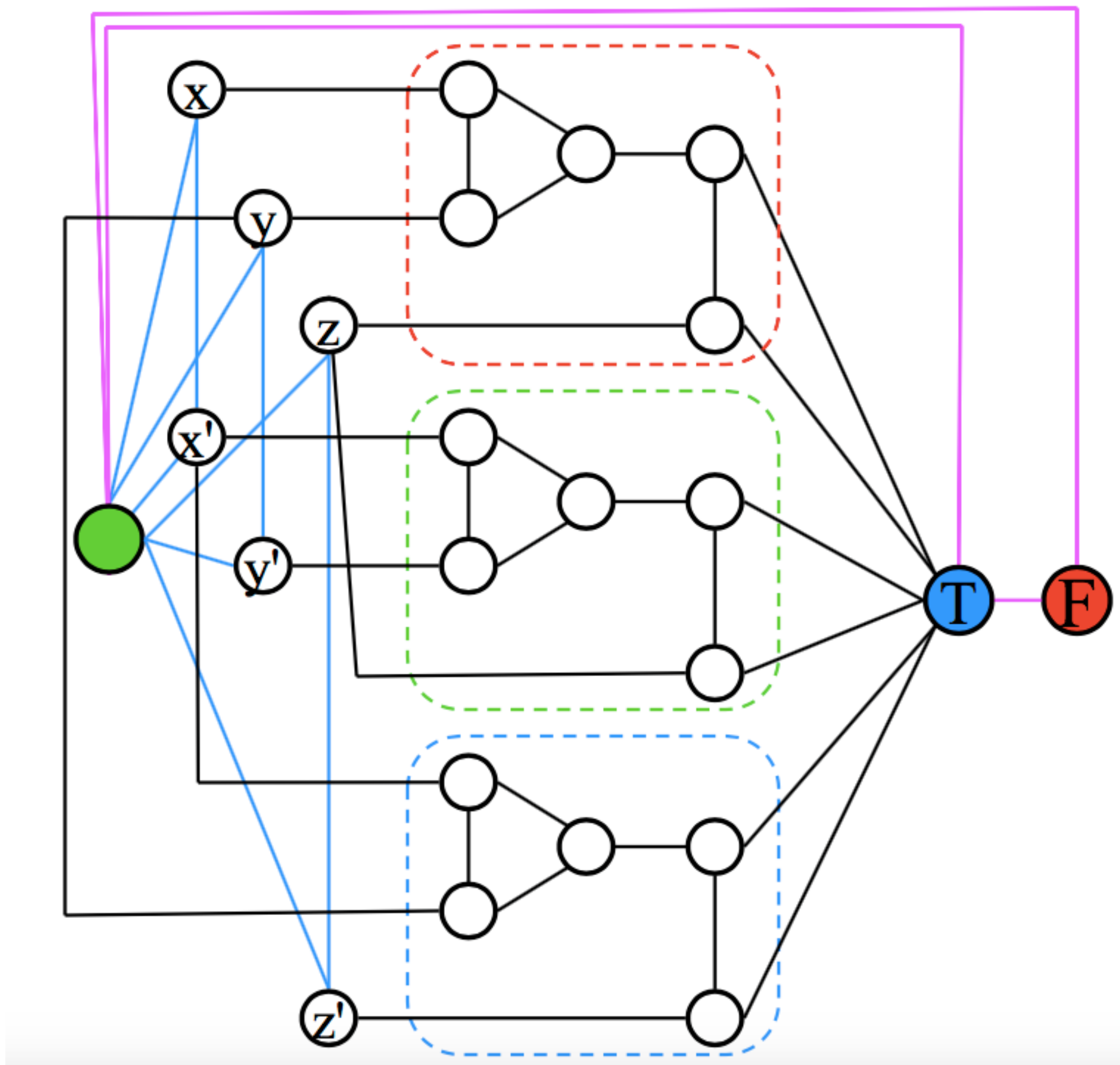
Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

Otteniamo il grafo

I tre nodi già colorati sono mutuamente connessi quindi devono usare tre colori diversi

Posso scegliere tre colori arbitrari come in figura



Riduzione da 3SAT a 3-colorazione di grafi

verde, blu (vero), rosso (falso)

Data la formula

(x or y or z) and (not x or not y or z) and (not x or y or not z)

(x or y or z) and (x’ or y’ or z) and (x’ or y or z’)

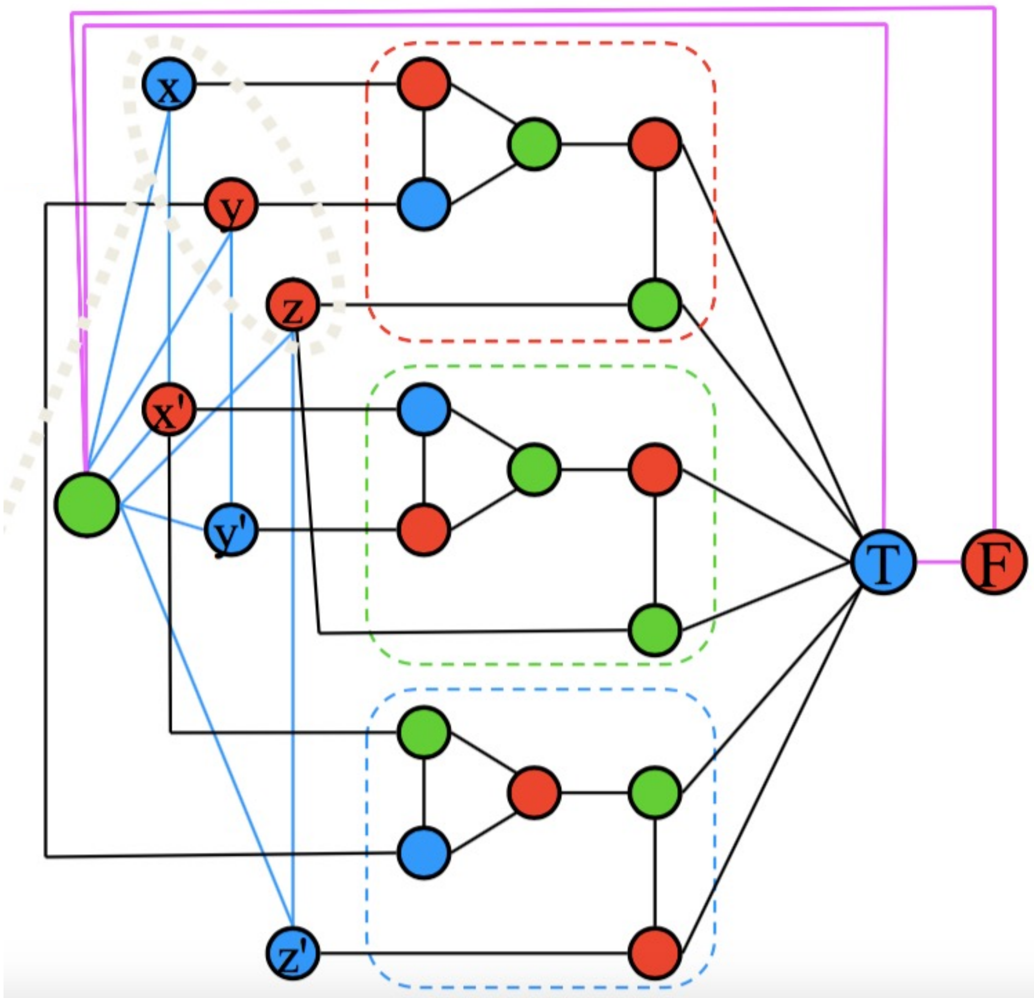
La figura mostra 3 colorabile con

x= blu, y= rosso, z= rosso

La figura verifica che

- x’=rosso, y’= blu, z= blu
- Per ogni gadgetdi clausola almeno uno dei nodi collegati a sinistra è blu (vero)

Quindi se assumiamo x=vero, y falso, z= falso otteniamo che 1 e 2 precedenti implicano che questa sia un’assegnazione di valori di verità che rende vera la formula



Riduzione da SAT a 3colorazione: conclusioni

1. Non è difficile modificare la prova precedente per dimostrare che decidere se un grafo è colorabile con k colori (k >3) è NP-completo

2. Cosa rende difficile il problema della colorazione: vertici di grado alto?
- NO: possiamo modificare la riduzione in modo tale che ogni nodo abbia massimo grado 3

3. I seguenti problemi sono facili (risolubili in tempo polinomiale)
- Decidere se un grafo può essere colorato con 1 colore (banale)
- Decidere se un grafo può essere colorato con 2 colori (buon esercizio)

4. Per alcune classi di grafi il problema è semplice
- Ogni albero può essere colorato con due colori

4. Ma non sempre restringere lo studio semplifica
- Grafo planare: un grafo che può essere disegnato su un foglio senza avere incroci fra gli archi
- Colorare un grafo planare con 3 colori è NP- completo (difficile)
- Colorare un grafo planare con 4 colori è facile (polinomiale)

Vertex Cover e Independent Set

Vertex Cover

Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che coprono tutti gli archi. (S copre i nodi se per ogni arco (u,v) abbiamo che almeno uno fra u e v appartiene a S)

Independent set

Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che non sono collegati fra loro - cioè se u e v appartengono a S allora non esiste arco (u,v)

E' facile vedere che i due problemi sono in NP

Usa come certificato un insieme S di k nodi; verifica la proprietà

Algoritmo (non deterministico)

Scegli un insieme S di k nodi

Verifica se S è un vertex cover (o un independent set)

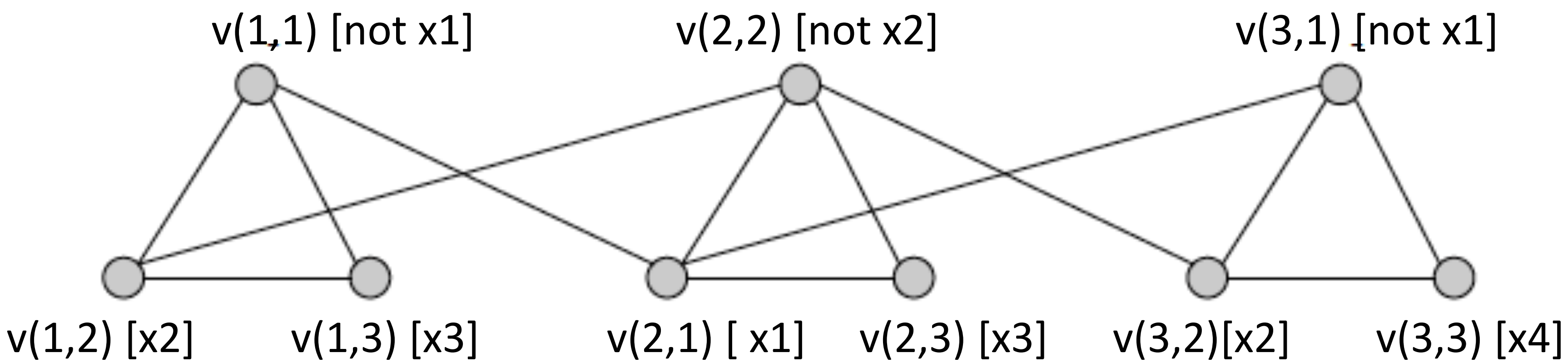
Riduzione da 3-SAT a Independent set

Data una formula f di 3SAT con m clausole si definisce un grafo con 3m nodi, 3 per ciascuna clausola e indichiamo con v(C,i) il nodo associato al letterale l_i della clausola C. Gli archi sono così definiti

- I tre nodi associati ad una medesima clausola sono mutuamente collegati da un arco
- Due nodi v(C,i) e v(C',j) associati a due letterali di due clausole diverse (C≠C') sono collegati da un arco se la variabile x corrisponde sia al letterale i della clausola C che al letterale j della clausola C' e in un caso è positiva e in un altro è negativa

Esempio. Data la formula:
((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)

Otteniamo il seguente grafo (in cui fra [..] è indicato il letterale della clausola

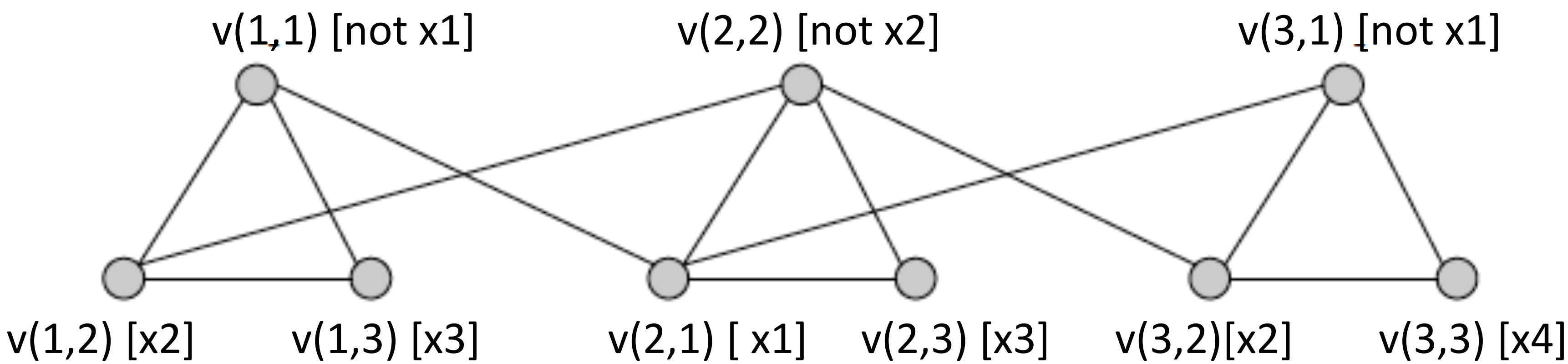


Riduzione da 3-SAT a Independent set

Data una formula f di 3SAT con m clausole si definisce un grafo con 3m nodi, 3 per ciascuna clausola e indichiamo con v(C,i) il nodo associato al letterale l_i della clausola C. Gli archi sono così definiti

- I tre nodi associati ad una medesima clausola sono mutuamente collegati da un arco
- Due nodi v(C,i) e v(C',j) associati a due letterali di due clasuole diverse (C≠C') sono collegati da un arco se la variabile x corrisponde sia al letterale i della clausola C che al letterale j della clausola C' e in un caso è positiva e in un altro è negativa

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile
Se e solo se implica che: Abbiamo una riduzione da 3-SAT a independent set se dimostriamo che
1 ⇒ dato insieme indipendente di taglia m f è soddisfacibile
2 ⇐ data soluzione di 3SAT a f nel grafo corrispondente esiste insieme indipendente di taglia m

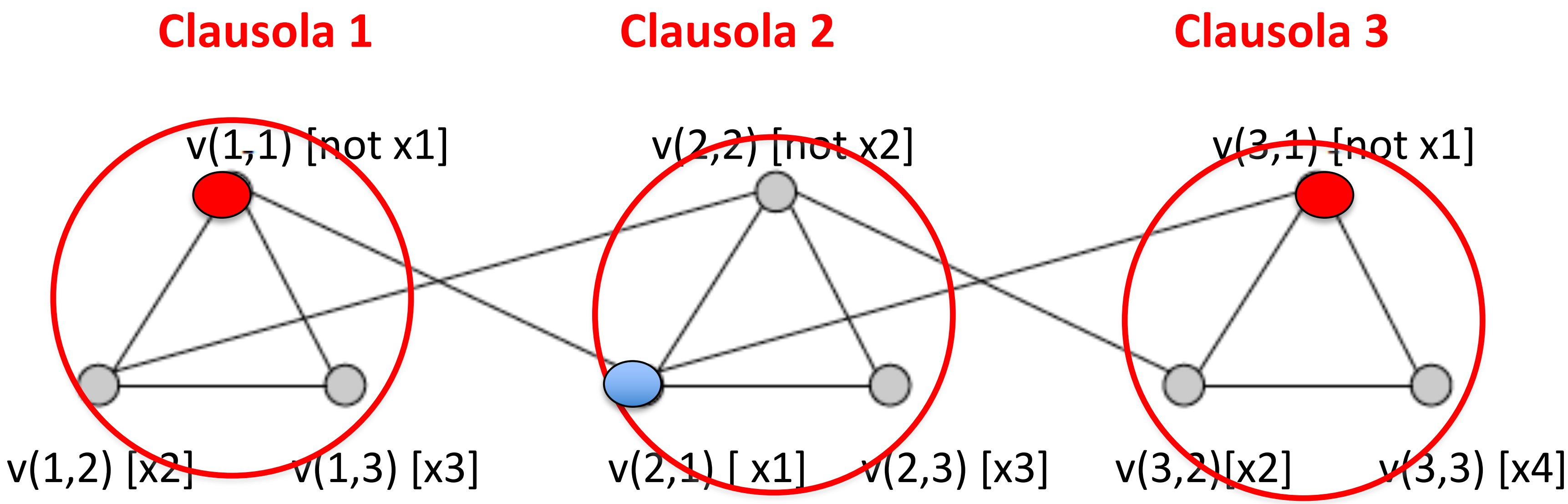


Riduzione da 3-SAT a Independent set

Data una formula F di 3SAT con m clausole si definisce un grafo con 3m nodi, 3 per ciascuna clausola e indichiamo con v(C,i) il nodo associato al letterale l_i della clausola C. Gli archi sono così definiti

- I tre nodi associati ad una medesima clausola sono mutuamente collegati da un arco
- Due nodi v(C,i) e v(C',j) associati a due letterali di due clasuole diverse (C≠C') sono collegati da un arco se la variabile x corrisponde sia al letterale i della clausola C che al letterale j della clausola C' e in un caso è positiva e in un altro è negativa

Esempio. Data la formula F:
((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)
Otteniamo il seguente grafo (in cui fra [..] è indicato il letterale della clausola
Nota che il nodo blu (v(2,1)) ha due archi con nodi rossi di altre clausole perché v(2,1) corrisponde al letterale x1 mentre i nodi rossi corrispondono al letterale not x1



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

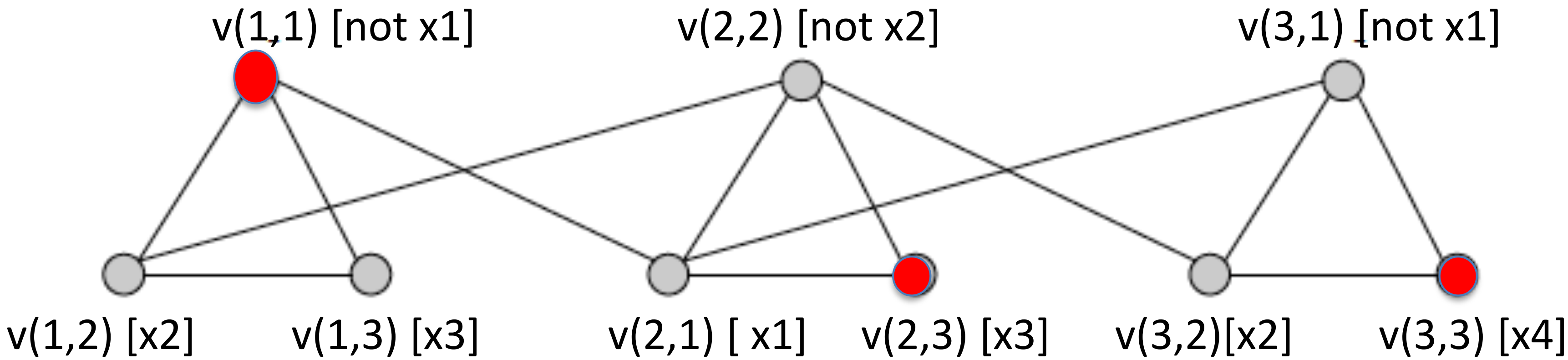
Prova 1 ⇒ (dato insieme indipendente S di dimensione m ottieni soluzioni a 3SAT)

Sia S ins. Indipendente di taglia m. E' facile vedere che

- S contiene esattamente un vertice in ogni triangolo (nodi triangolo sono adiacenti, m triangoli)
- Poni questi letterali a Vero
- Questa assegnazione è consistente (gli archi fra clausole impediscono di dare a x sia il valore vero che falso) e soddisfa F (ogni clausola ha un letterale vero)

Assegnazione corrispondente a nodi rossi (insieme indipendente) da assegnazione x1=Falso x3=x4= Vero (x2 può essere sia Vero che Falso e la formula è soddisfatta)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

Prova

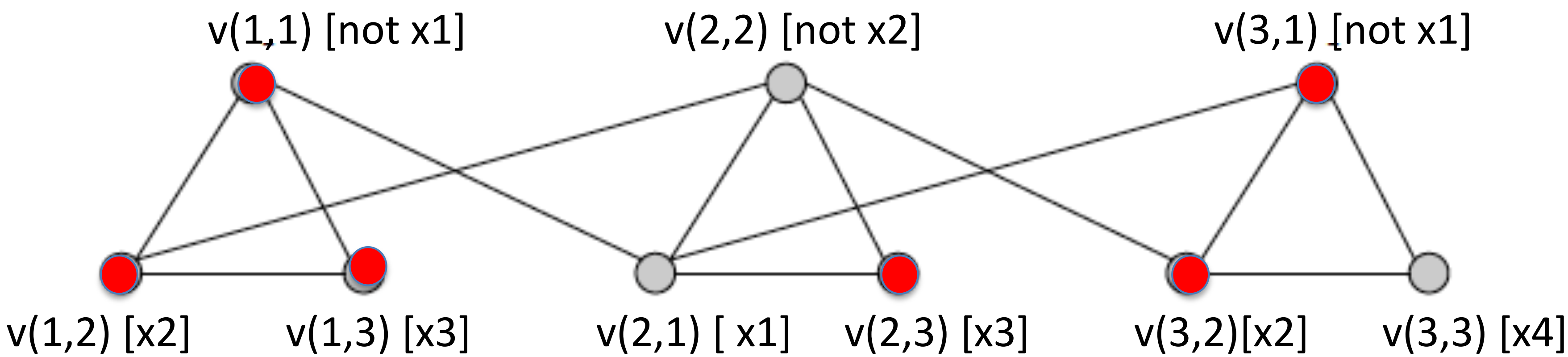
2 $\Leftarrow$ (data soluzione di 3SAT ottieni insieme indipendente)

Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo (ci deve essere almeno uno dato che ciascuna clausola è soddisfatta).

Questo insieme è un insieme indipendente di taglia m (ricorda m triangoli)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)

Assegnazione soddisfa F: x1=x4= Falso x2=x3=Vero

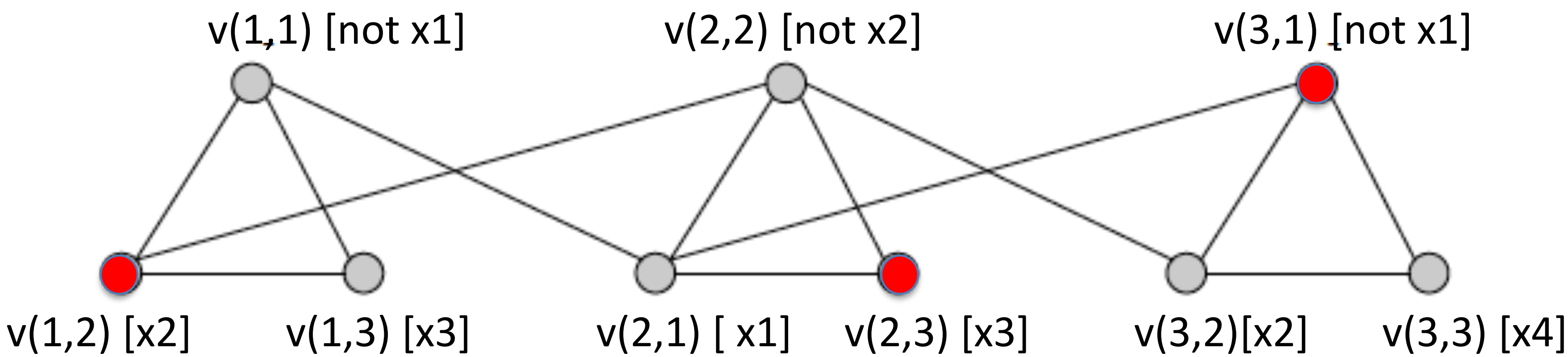


Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile
Prova
2 ⇐ (data soluzione di 3SAT ottieni insieme indipendente)
Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo (ci deve essere dato che ciascuna clausola è soddisfatta).
Questo insieme è un insieme indipendente di taglia m (ricorda m triangoli)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)
Assegnazione soddisfa F: x1=x4= Falso x2=x3=Vero

Nodi rossi in figura sono insieme indipendente



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

Prova

1 ⇒ (dato insieme indipendente S di dimensione k ottieni soluzioni a 3SAT)

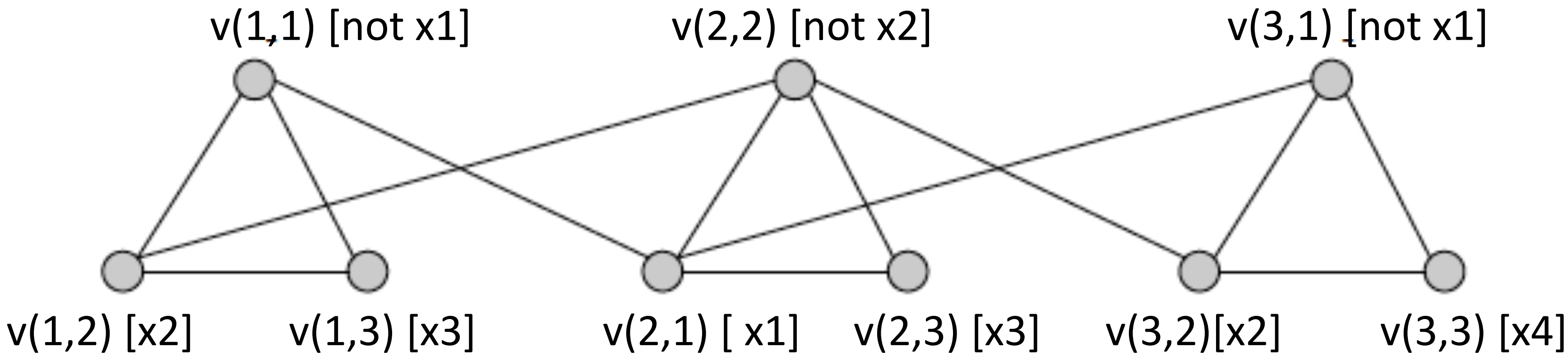
Sia S ins. Indipendente di taglia m. E' facile vedere che

- S deve contenere esattamente un vertice in ogni triangolo
- Poni questi letterali a Vero
- Questa assegnazione è consistente (gli archi fra clausole impediscono di dare a x sia il valore vero che falso) e soddisfa F (ogni clausola ha un letterale vero)

2 ⇐ (data soluzione di 3SAT ottieni insieme indipendente)

Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo. Questo insieme eè un insieme indipendente di taglia m

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)



Riduzione da Vertex Cover a Independent set

Vertex Cover (VC) Dato un grafo G ed un intero k esiste un insieme S di k nodi che coprono tutti gli archi?

Independent set (IS) Dato un grafo G ed un intero h esiste un insieme S di h nodi che non sono collegati fra loro (cioè se u e v appartengono a S non esiste arco (u,v)) ?

I due problemi sono molto simili.

Riduzione da Vertex Cover a Independent set

Data istanza di VC G e k costruiamo un'istanza di IS G (nota stesso grafo di VC) e poniamo h = n-k (n numero nodi G)

Questa è una riduzione

Dato un grafo G, S è insieme indipendente allora V − S è un vertex cover.

Riduzione da Vertex Cover a Independent set

Riduzione da Vertex Cover a Independent set
Data istanza di VC G e k costruiamo un'istanza di IS G (nota stesso grafo di VC) e poniamo h = n-k (n numero nodi G)

Prova che questa è una riduzione.
1) Dato un grafo G se S è insieme indipendente allora V − S è un vertex cover
se S è un insieme indipendente, ogni arco (x, y) non può avere entrambi gli estremi in S; quindi almeno uno dei due deve essere in V − S, quindi V − S è un vertex cover.
1) Dato un grafo G se T è vertex cover allora V − T è insieme indipendente
se T è un vertex cover, vogliamo dimostrare che ` S e un insieme indipendente. Supponiamo per assurdo T sia un vertex cover ma che (V-T) non sia insieme indipendente; quindi esiste un arco (x, y) che unisce due nodi in (V-T). Allora nessuno degli estremi di (x, y) sta in T, il che implica che T non è un vertex cover.

Questo conclude la riduzione che (Vertex Cover) è riducibile in tempo polinomiale a Independent Set
NOTA: i ragionamenti fatti evidenziano anche una riduzione da independent set a Vertex cover

Problemi in NP: certificato I

Nella definizione di NP il concetto di certificato può sembrare poco intuitivo

Consideriamo la classe dei problemi decisionali - data una qualunque istanza del problema - è possibile provare che
la risposta associata all'istanza è true (appartiene al linguaggio) fornendo una descrizione (certificato) polinomiale della possibile soluzione che soddisfa i requisiti del problema.

Ad esempio,

- nel caso del problema SAT, un certificato è dato da un assegnamento di verità alle variabili della formula;
- nel caso del problema Colorazione, è un'associazione nodo-colore
- nel caso del problema Insiemi indipendenti su un grafo in cui V è l'insieme dei vertici , è un sottoinsieme di V

In ciascuno dei casi precedenti

- il certificato è polinomiale nella dimensione dell'input
- Data una istanza del problema ed un certificato C è semplice (tempo polinomiale) verificare che sia una soluzione

Problemi in NP: certificato II

la risposta associata all'istanza è SI (true) (appartiene al linguaggio) fornendo una descrizione (certificato) della possibile soluzione che soddisfa i requisiti del problema.

In ciascuno dei casi precedenti (SAT, Vertex cover, colorazione di un grafo, ecc.) il certificato è una soluzione del problema e la l'algoritmo (macchina di Turing det.) della definizione di NP non fa altro che verificare la correttezza della soluzione

Domanda
perché abbiamo bisogno di una definizione di certificato; non potremmo dire che diamo direttamente una soluzione; ad esempio qualcosa del tipo

la risposta associata all'istanza è true (appartiene al linguaggio) fornendo una ~~descrizione (certificato) della possibile~~ soluzione che soddisfa i requisiti del problema.

Risposta: la definizione precedente ***va bene nella grandissima maggioranza dei casi ma non vale in tutti i casi.***

Problemi in NP: certificato III

la risposta associata all'istanza è true (appartiene al linguaggio) fornendo una ~~descrizione (certificato) della possibile~~ soluzione che soddisfa i requisiti del problema.

la definizione precedente va bene nella maggioranza dei casi ma non vale in tutti i casi.

Un esempio in cui è non è facile trovare un certificato polinomiale è l seguente

Test di primalità: dato un intero p dispari fornire un certificato polinomiale che stabilisce che p è primo (NOTA dimensione input in questo caso è (log p))

- un possibile certificato è dato dal resto della divisione di p con tutti i numeri dispari minori della radice quadrata di p
- questo certificato NON è polinomiale nella dimensione dell'input [i possibili resti sono $O(p^{1/2})$]
- Un certificato polinomiale esiste ma è complesso da dare (oltre i nostri scopi)

Conclusione
Questo (e altri esempi) giustificano la definizione data della classe NP usando certificati

Riduzioni NP: sommario

Dimostrare che A si riduce a B
Richiede trasformazione f: A→ B (istanze di A in istanze di B)
tale che

- Se istanza x di A appartiene al linguaggio anche f(x) appartiene a B
- Se istanza y di B appartiene al linguaggio anche $f^{-1}(y)$ appartiene a A

Data riduzione da A a B: B è almeno tanto difficile quanto A

- A difficile (NP-completo) → B deduco che anche B è difficile
- B facile (polinomiale) → A deduco che anche A è facile
- A facile (polinomiale)→ ???: la riduzione non mi aiuta a capire se B sia polinomiale o no

.

Problemi di ottimizzazione

Colorazione di grafi
Problema di decisione: Esiste una colorazione con k o meno colori?
Problema di ottimizzazione: Qual è il minimo numero di colori necessario per colorare un grafo?

Problemi di ottimizzazione

La definizione di NP si estende a problemi di ottimizzazione riducendo la ricerca dell'ottimo a più problemi di riconoscimento di linguaggi

Esempio determinare numero cromatico Crom(G) di un grafo G

Il problema richiede di trovare il più piccolo k tale che G è k colorabile

1. Il numero cromatico di un grafo con n nodi è compreso fra 1 e n
2. Per trovare k(G) effettuiamo una ricerca binaria iniziando con h= n/2 e ci chiediamo se G sia h colorabile
   - Se sì allora ci chiediamo se G sia n/4 colorabile
   - Se no ci chiediamo se G sia 3n/4 colorabile
3. Risolvendo (log n) problemi di riconoscimento risolviamo il problema di ottimizzazione

Si applica a tutti i problemi che consideriamo e giustifica l'attenzione su problemi di decisione

Problemi di ottimizzazione e problemi NP-hard

Posso avere problemi più difficili di NP

- Definizione vista*: Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.*
- Definizione nuova: Un linguaggio L è NP-hard (NP-difficile) ~~se L appartiene a NP e~~ se ogni altro linguaggio in NP è polinomialmente riducibile a L (ma L può non appartenere a NP)

I linguaggi NP-hard possono essere più difficili di NP

Esempi

- Formule logiche con quantificatori (per ogni, esiste)
  Data una formula booleana f(x,y,v,w,z) mi chiedo se
  per ogni x (x vero o x falso) [(esiste y per cui f e' vera) oppure (per ogni v f e' falsa)]

- gioco della dama generalizzato su una scacchiera n x n
  Esiste una strategia vincente per il bianco?
  In altre parole esiste una mossa del bianco tale che, per ogni mossa del nero, esiste una mossa del bianco tale che, per ogni mossa del nero…..

I problemi precedenti richiedono spazio polinomiale e sono nella classe PSPACE

Riduzioni NP: Robustezza definizione

1. Composizione di polinomi è un polinomio
quindi se riduco SAT a problema A e poi A a problema B e poi B a problema C la composizione delle riduzioni fornisce una riduzione polinomiale da SAT a C, quindi concludo C è NP-difficile (e se C è in NP allora è NP-completo)

2. La definizione data non dipende dal modello di calcolo
I modelli di calcolo noti (MdT, JAVA, Python, C ecc.) sono tutti polinomialmente equivalenti; infatti un algoritmo polinomiale in un modello A si traduce in un algoritmo polinomiale in un altro modello B e viceversa.

Teoria NP-completezza: conclusioni

Desiderata

Classificare i problemi distinguendo quelli che possono essere risolti in tempo polinomiale da quelli che non si possono risolvere in tempo polinomiale

Notizia frustrante

Molti problemi fondamentali non sono classificabili (anche se studiati da decenni)

Cosa sappiamo

Teoria NP-completezza mostra che molti di questi problemi sono "*computazionalmente equivalenti*" e appaiono essere formulazioni differenti dello stesso problema

Spesso il confine fra facile (polinomiale) e difficile (NP- completo) è noto ma in alcuni casi controintuitivo

Facile (polinomiale)	**Difficile (NP-hard)**
2-SAT	3-SAT
Colorare i nodi di un grafo con 2 colori	Colorare i nodi di un grafo con 3 colori
Programmazione lineare	Programmazione lineare a numeri interi
Cammino minimo in un grafo	Cammino più lungo in un grafo
Vertex Cover in grafi bipartiti	Vertex Cover in grafi generali
Test per stabilire se un numero è primo	Fattorizzazione di un numero intero

Teoria NP-completezza: classi di complessità

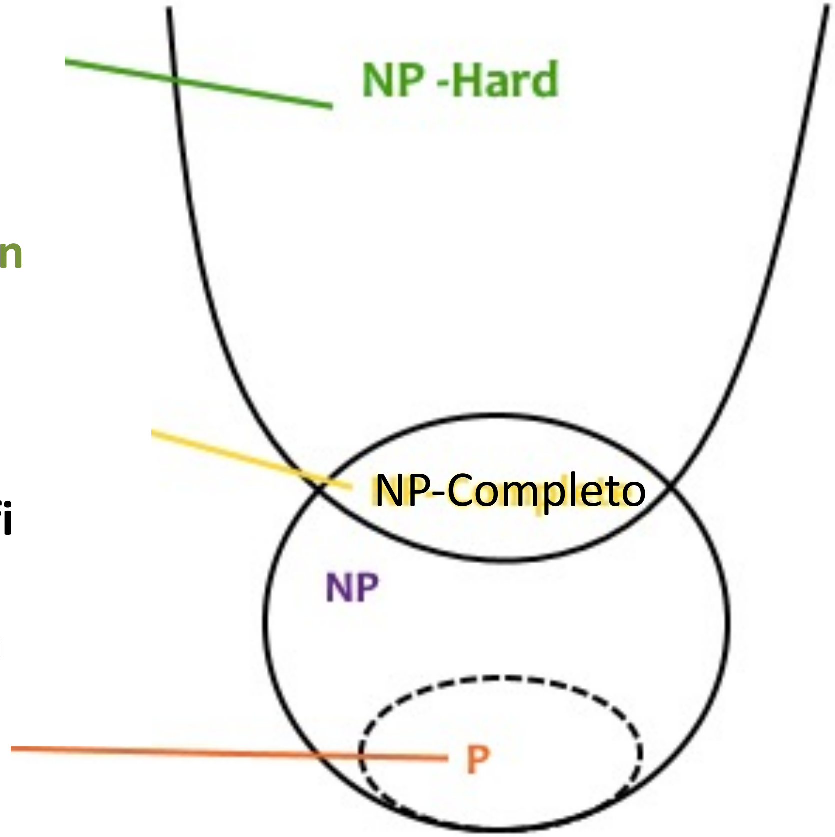
La figura riassume le relazioni fra i diversi problemi assumendo che la congettura P≠NP sia vera

NP-Hard:
L è un problema **NP hard** se esiste una riduzione da un problema NP completo a L (però L non appar-tiene necessa-riamente alla classe P)

- **Problema della fermata**
- **Soddisfacibilità di formule logiche con quantificatori**

- **SAT**
- **Knapsack**
- **Colorazione di grafi**
- **Independent set**
- **Programmazione a numeri interi**

Minimo albero ricoprente,
2SAT, Cammino più breve in un grafo,
Calcolo Massimo Comun Divisore,



Teoria NP-completezza: implicazioni pratiche

Il vostro capo vi chiede di trovare un algoritmo veloce (polinomiale) per un problema; avete due settimane di tempo

Non riuscite a trovare un algoritmo veloce e dopo due settimane il capo vi chiama e si lamenta; voi siete in difficoltà
Avete un’altra settimana di tempo ….

Teoria NP-completezza: implicazioni pratiche

Dopo una settimana voi riuscite a provare che il problema datovi è NP-completo

A questo punto potete dire al capo che: “*Io non sono riuscito a trovare un algoritmo polinomiale ma nessuno di questi ricercatori è riuscito a farlo. La congettura è che non esiste un algoritmo veloce; quindi… (a vostra scelta…)*”

Teoria NP-completezza: implicazioni pratiche

Dopo una settimana voi riuscite a provare che il problema datovi è NP-completo

A questo punto potete dire al capo che: "*Io non sono riuscito a trovare un algoritmo polinomiale ma nessuno di questi ricercatori è riuscito a farlo. La congettura è che non esiste un algoritmo veloce; quindi …*

quindi l'unica cosa possibile è trovare un algoritmo che sia in grado di risolvere efficentemente la maggior parte dei casi che si incontrano nella pratica

Domande su P=NP?

Assumi che P non sia uguale a NP. Sapendo che SAT è NP-completo, quale delle seguenti affermazioni sono vere? Motivare le risposte

Vero o falso?

(a) Non esiste un algoritmo che risolve istanze di SAT arbitrariamente grandi

(b) Non esiste un algoritmo efficiente (tempo polinomiale) che risolve tutte le istanze di SAT arbitrariamente grandi

(c) Esiste un algoritmo che risolve velocemente (tempo polinomiale) una arbitraria istanza di SAT ma nessuno è stato ancora in grado di trovarlo

(d) SAT non è in P

(e) Tutti gli algoritmi per SAT terminano in tempo esponenziale per tutti i possibili input

(f) Per ogni possibile input tutti gli algoritmi noti per SAT terminano in tempo polinomiale

Domande su P=NP?

Assumi che P non sia uguale a NP. Sapendo che SAT è NP-completo, quale delle seguenti affermazioni sono vere? Motivare le risposte

(a) Non esiste un algoritmo che risolve istanze di SAT arbitrariamente grandi (falso, ad esempio DPLL risolve SAT)

(b) Non esiste un algoritmo efficiente (tempo polinomiale) che risolve istanze di SAT arbitrariamente grandi (vero)

(c) Esiste un algoritmo che risolve velocemente (tempo polinomiale) una arbitraria istanza di SAT ma nessuno è stato ancora in grado di trovarlo (falso, se uno trova un algoritmo polinomiale per SAT allora P=NP)

(d) SAT non è in P (non lo sappiamo, congetturiamo di no)

(e) Tutti gli algoritmi per SAT terminano in tempo esponenziale per tutti i possibili input (falso)

(f) Per ogni possibile input tutti gli algoritmi per SAT non terminano in tempo polinomiale (falso)

Domande su riduzioni fra problemi

Assumi che X e Y siano due problemi di decisione; supponi che esista una riduzione polinomiale che riduce X a Y.

Quale delle seguenti affermazioni sono vere?

(a) Se Y è NP-completo anche X lo è.
(b) Se X è NP-completo anche Y lo è.
(c) Se Y è NP-completo e X è in NP allora X è NP-completo.
(d) Se X è NP-completo e Y è in NP allora Y è NP-completo .
(e) X e Y non possono essere entrambi NP-completi.
(f) Se X è in P, allora Y è in P.
(g) Se Y è in P, allora X è in P.

Domande su riduzioni fra problemi

Assumi che X e Y siano due problemi di decisione; supponi che esista una riduzione polinomiale che riduce X a Y.

Quale delle seguenti affermazioni sono vere?

(a) Se Y è NP-completo anche X lo è.

(b) Se X è NP-completo anche Y lo è.

(c) Se Y è NP-completo e X è in NP allora X è NP-completo.

(d) Se X è NP-completo e Y è in NP allora Y è NP-completo .

(e) X e Y non possono essere entrambi NP-completi.

(f) Se X è in P, allora Y è in P.

(g) Se Y è in P, allora X è in P.

Vere: d, g; le altre false

Riduzione: DOPPIO-SAT

DOPPIO-SAT : data una formula logica ψ ci chiediamo se esistono almeno due assegnazioni di valori di verità diverse che rendono ψ vera?

Esempio

- (x or y) and (x or not y)   appartiene a DOPPIO-SAT
- (x or y) and (x or not y) and (not x or y) non apppartiene a DOPPIO-SAT (infatti è soddisfatta solo se assegniamo x=y= vero)

Dimostrare che DOPPIO-SAT è NP-completo.

Risposta

1. DOPPIO-SAT ∈ NP:
2. Ogni problema in NP è riducibile a DOPPIO-SAT

Riduzione: DOPPIO-SAT

DOPPIO-SAT : data una formula logica ψ ci chiediamo se esistono almeno due assegnazioni di valori di verità diverse che rendono ψ vera?

Esempio

- (x or y) and (x or not y) appartiene a DOPPIO-SAT
- (x or y) and (x or not y) and (not x or y) non apppartiene a DOPPIO-SAT (infatti è soddisfatta solo se assegniamo x=y= vero)

Dimostrare che DOPPIO-SAT è NP-completo.

Risposta

1. DOPPIO-SAT ∈ NP:

- Un algoritmo non deterministico semplicemente indovina due assegnazioni di valori di verità diverse e verifica che ambedue soddisfino la formula.
- Oppure possiamo considerare un certificato C formato da due assegnazioni diverse di valori logici alla formula che ambedue verificano la formula; dato C è semplice e veloce (tempo polinomiale) verificare che C contenga due assengazioni diverse di valori di verità e che ambedue le assegnazioni soddisfano ψ

2. Ogni problema in NP è riducibile a DOPPIO-SAT

Dimostrare che DOPPIO-SAT è NP-completo.

2. Ogni problema in NP è riducibile a DOPPIO-SAT

È sufficiente fornire una riduzione polinomiale da SAT a DOPPIO-SAT.

Data ψ, crea una nuova formula ψ′ aggiungendo una nuova clausola (x or not x) a ψ, dove x è una nuova variabile non presente in ψ. Poi verifica se

⟨ψ′⟩ ∈ DOPPIO-SAT

- Questa riduzione chiaramente richiede tempo polinomiale
- Ora dimostriamo che la formula originale ⟨ψ⟩ ∈ 3SAT se e solo se la nuova formula ⟨ψ′⟩ ∈ DOPPIO-SAT. Se la formula originale non è soddisfacibile allora anche la nuova formula non lo è. Se invece ⟨ψ⟩ ∈ 3SAT, allora esiste almeno un'assegnazione di valori X alle variabili logiche che rende vera ψ.
- Considera l'assegnazione di valori di verità alle variabili di ψ′ che coincide con l'assegnazione dei valori delle variabili di ψ e assume x = 0 (falso). Ovviamente questa assegnazione rende vera ψ′ (infatti soddisfa tutte le clausole di) ψ′).

  Analogamente l'assegnazione di valori di verità alle variabili di ψ′ che coincide con l'assegnazione dei valori delle variabili di ψ e assume x = 1 (vero) soddisfa ψ′. Pertanto abbiamo mostrato che esistono due assegnazioni di valori di verità che soddisfano ψ′, so ⟨ψ′⟩ ∈ DOUBLE-SAT.

Ridurre vertex cover a Programmazione a numeri interi (nota non implica che vertex cover sia NP-completo)

Vertex cover: dato un grafo ed uniintero k esiste un insieme S di archi che coprono tutti gli archi del grafo? (un arco (xi, xj) e' coperto da S se almeno uno fra x e y appartiene a S)

Dato un grafo introduco una variabile x_i per ogni nodo i del grafo

Idea= se nodo i appartiene a S allora pongo $x_i=1$ $x_i=0$ altrimenti (i non appartiene a S)

Vincoli

Cardinalita' di S e' al max k: $\Sigma x_i = k$ (sommatoria per ogni i) (a)

Ogni arco è coperto da insieme S: arco (x_i, x_j) è coperto da S vuol dire

$x_i+x_j \geq 1$ per ogni arco (i, j) del grafo (b)

$0 \leq x_i \leq 1$ x_i intero per ogni i n (c)

(a) Implica che l'insieme dei nodi scelti ha cardinalità k
(b) Implica che ogni arco del grafo è coperto
(c) è un vincolo binario che indica che un nodo o fa parte di S o no (se si rilascia questo vincolo potremmo avere che un arco possa essere coperto anche ponendo $x_i = x_i = 0.5$)

Ridurre vertex cover a Programmazione a numeri interi (nota non implica che vertex cover sia NP-completo)

Vertex cover: dato un grafo ed uniintero k esiste un insieme S di archi che coprono tutti gli archi del grafo? (un arco (xi, xj) e' coperto da S se almeno uno fra x e y appartiene a S)

Dato un grafo introduco una variabile x_i per ogni nodo i del grafo

Idea= se nodo i appartiene a S allora pongo x_i=1 x_i=0 altrimenti (i non appartiene a S)

Vincoli

Cardinalita' di S e' al max k: $\Sigma x_i = k$ (sommatoria per ogni i) (a)

Ogni arco è coperto da insieme S: arco (x_i, x_j) è coperto da S vuol dire

$x_i+x_j \geq 1$ per ogni arco (i, j) del grafo (b)

$0 \leq x_i \leq 1$ x_i intero per ogni i n (c)

(a) Implica che l'insieme dei nodi scelti ha cardinalità k
(b) Implica che ogni arco del grafo è coperto
(c) è un vincolo binario che indica che un nodo o fa parte di S o no (se si rilascia questo vincolo potremmo avere che un arco possa essere coperto anche ponendo $x_i = x_i = 0.5$)

Esercizi

- Dimostrare che SAT, colorazione di un grafo, sono in NP
- Ridurre Independent set a vertex cover (e viceversa)
- Ridurre Independent set a Programmazione a numeri interi
- Ridurre colorazione di un grafo con 3 colori a Programmazione a numeri interi
- Il problema Cricca è così definito: dato un grafo G e un intero k ci si chiede se esiste un insieme di k nodi a due a due collegati da un arco. Dimostrare che Cricca appartiene a NP. Ridurre Independent set a Cricca mostrando pertanto che Cricca è NP-completo
- Dimostrare che il problema di soddisfacibilità per formule logiche in forma disgiuntiva è polinomiale (una formula logica disgiuntiva è del tipo

  (l1 and l2 and l3) or (l4 and … and...) or (... and … and...) ...

  esempio formula disgiuntiva

  (x1 and x2 and (not x4))or((not x1) and x2 and x3) or ((not x1) and x2 and x4)

Ridurre 3 colorazione di grafi a 4 colorazione

Abbiamo visto che colorare un grafo con 3 colori è NP-completo.

Ma il linguaggio C4 dei grafi G colorabili con al massimo 4 colori è lo stesso NP-completo? Risposta: SÌ

Prova: dobbiamo ridurre un linguaggio NP-completo a C4. Scegliamo il linguaggio C3 dei grafi G colorabili con 3 colori. Sappiamo che C3 è NP-completo

Una riduzione polinomiale da C3 a C4 è la seguente

Dato un grafo G con n nodi x1, x2, ….xn considera il grafo G' con n+1 nodi

x1, x2,… xn, y.

Gli archi di G sono anche archi di G' (cioè se (xi,xj) è un arco in G lo è anche in G')

Inoltre abbiamo n archi (x1,y) (x2,y) ... (xn,y) fra y e ogni altro nodo

- Chiaramente la riduzione è polinomiale
- inoltre è facile verificare che il nodo y di G' deve avere un colore diverso da tutti i colori dati ai nodi di G. Quindi G è colorabile con tre colori se e solo se G' è colorabile con 4 colori.

- La riduzione è facilmente modificabile per dimostrare che per ogni costante k il problema di colorare un grafo con al più k colori è NP-completo.
- Nota che si chiede k costante; se k è una funzione f() del numero di nodi la risposta dipende dalla funzione f(); ad esempio
- (a) colorare un grafo di n nodi con n colori (o anche n-1 colori) è facile;
- (b) colorare un grafo con $\sqrt{n}$ (radice quadrata del numero di nodi n) è NP-completo. (esercizio provare (a) e (b) precedent)

Esercizi

Provare che la classe NP dei linguaggi è chiusa rispetto alle seguenti operazioni:

(a) Unione di due linguaggi L1 e L2 (b) Intersezione di due linguaggi L1 e L2

(c) Concatenazione di due linguaggi L1 e L2

Risposta: Per dimostrare viene utile usare a definizione di NP che usa certificati.

Dato x ed un linguaggio L , c(x) certifica che x appartiene a L e un polinomio p per cui x se x appartiene a L allora

- verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- esiste un algoritmo A(x, c(x)) che con ingresso x e c(x) certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in |x| e |c(x)| (lunghezza di x e di c(x)). Nel seguito assumi che A(x,c(x))=1 implica x in L

(a) Siano L1, L2 due linguaggi in NP con certificati c1() e c2() con algoritmi di verifica A1 e A2; assumi che A(x,c(x))= 1 (0) se x appartiene a L (non appartiene a L)

Dato x un certificato c' per L1 U L2 è (c1(x), c2(x)) e l'algoritmo di verifica è:

```
A1(c1(x),x)
if A1(c1(x),x)=1
    then return 1
    else return A2(c2(x),x)
```

Esercizio: completare la prova e dimostrare che questo è algoritmo che verifica se x appartiene a L1 U L2

Domande (b) e (c) analoghe (nota per (c) è opportuno usare un ciclo

Esercizi

Prova che la riduzione polinomiale è una relazione transitivia. Cioè dati tre linguaggi L1, L2, L3, se L1 è riducibile in tempo polinomiale a L2 e L2 è riducibile in tempo polinomiale a L3 allora L1 è riducibile in tempo polinomiale a L3

Risposta : Siano f(x), g(x) le funzioni calcolabili in tempo polinomiale che riducono L1 a L2 e L2 a L3, rispettivamente. Sia h(x) = g(f(x)).

Per tutte le stringhe x abbiamo :

- x appartiene a L1 se e solo se f(x) appartiene a L2
- y = f(x) appartiene a L2 se e solo se g(y) = g(f(x)) appartiene a L3

Abbiamo che

- x appartiene a L1 se e solo se h(x) = g(f(x)) appartiene a L3
- Nota che h(x) = g(f(x)) è calcolabile in tempo polinomiale dato che è la composizione di due funzioni calcolabili in tempo polinomiale
- Quindi questo prova che L1è riducibile in tempo polinomiale a L3

Domande

Assumi nel seguito che P ≠ NP; quali delle seguenti affermazioni sono vere o false? Motivare le risposte.

(a) Il linguaggio L={0, 1}* appartiene a P?

(b) Esistono linguaggi NP-completi che sono regolari (sugg. Usare il fatto che linguaggi regolari sono accettati da automi a stati finiti deterministici)

(c) Se L contiene propriamente un linguaggio L1 NP-completo (cioè ogni stringa di L1 appartiene a L e esistono stringhe di L che non appartengono a L1) allora anche L è NP-completo

(d) Tutti i problemi NP-Completi possono essere risolti in tempo $O(2^{p(n)})$, per qualche polinomio p(n) (n rappresenta lunghezza input)

(e) Il problema della fermata è NP-completo

(f) Il problema della fermata è NP-difficile

Esercizi

Assumi nel seguito che P ≠ NP; quali delle seguenti affermazioni sono vere o false? Motivare le risposte.

(a) Il linguaggio {0, 1}?* appartiene a P? Vero

(b) Esistono linguaggi NP-completi che sono regolari (sugg. Usare il fatto che linguaggi regolari sono accettati da automi a stati finiti deterministici) Falso (un automa è un algoritmo con tempo di esecuzione lineare nella dimensione dell'input)

(c) Se L contiene propriamente un linguaggio L1 NP-completo (cioè ogni stringa di L1 appartiene a L e esistono stringhe di L che non appartengono a L1) allora anche L è NP-completo Falso (il linguaggio {0, 1}?* appartiene a P; inoltre possiamo codificare con una stringa binaria SAT (una stringa binaria x appartiene a SAT se la formula logica che codifica è soddisfacibile); chiaramente SAT è incluso nel linguaggio {0, 1}?*)

(d) Tutti i problemi NP-Completi possono essere risolti in tempo $O(2^{p(n)})$, per qualche polinomio p(n) (n rappresenta lunghezza input) Vero (abbiamo visto che una MdT determinsitica è in grado di simulare in tempo $O(2^{p(n)})$ una macchina di Turing non deterministica che ha tempo di calcolo O(p(n))

(e) Il problema della fermata è NP-completo Falso (per essere NP completo un problema deve appartenere a NP; il problema della fermata è indecidibile!)

Assumi nel seguito che P ≠ NP; quali delle seguenti affermazioni sono vere o false? Motivare le risposte.

(f) Il problema della fermata Halt è NP-difficile

Vero: bisogna dimostrare che dato un linguaggio L in NP esiste una riduzione da L a Halt.

Dato che L è in NP esiste MdT M(x) non determinsitica che decide in tempo polinomiale se x appartiene a L. Considera il seguente programma M1

```
if M(x) = 1  (M accetta x)
    then return 1 (accetta e si ferma)
    else while true do
        { cicla per sempre} (se M non accetta x allora cicla)
```

Sia Halt(y,x) il linguaggio delle stringhe per cui la MdT codificata da y si ferma con input x

Dati M1 e x considera la seguente funzione (riduzione) f(x)= (M1, x)

Proviamo che f() è una riduzione da L a problema della fermata. Chiaramente la funzione è calcolabile in tempo polinomiale. Inoltre abbiamo che

x appartiene a L	<==>	M(x) =1
	<==>	M1(x) termina
	<==>	(M1, x) appartiene a Halt(y,x)

Quindi abbiamo dimostrato che se L appartiene a NP allora L è riducibile a Halt

Esercizi

Supponi che qualcuno vi fornisca un algoritmo A() che - dato x - in tempo polinomiale decide se x codifica una formula logica soddisfacibile.   L'algoritmo fornisce come risposta solo SI o NO.

Descrivi come usare questo algoritmo per trovare in tempo polinomiale un'assegnazione di valori alle variabili che rende vera la formula se la formula è soddisfacibile

Ricorda: noi assumiamo che P≠NP e quindi che non esista un algoritmo che decide in tempo polinomiale se x codifica una formula soddisfaibile.

Sugg.:

Sia F(x1,x2,…xn) una formula nelle variabili x1, x2,…xn e sia F(s1, x2,...xn) la formula logica (con n-1 variabili) ottenuta da F fissando il valore di x1 pari a s1 in tutte le clausole di F (s1 rappresenta il valore vero o falso)

Chiaramente se F(x1,x2,…xn) è soddisfacibile almeno una fra le due formule logiche F(vero,x2,…xn) e F(falso,x2,…xn) deve essere soddisfacibile.

Supponi che qualcuno vi fornisca un algoritmo A() che dato x in tempo polinomiale decide se x codifica una formula soddisfacibile.

Descrivi come usare questo algoritmo per trovare in tempo polinomiale un'assegnazione di valori alle variabili che rende vera la formula se la formula è soddisfacibile)

Sia F(x1,x2,…xn) una formula nelle variabili x1, x2,… e sia F(s1, x2,...xn) la formula logica (con n-1 variabili) ottenuta da F fissando il valore di x1 pari a s1 in tutte le clausole di F (s1 rappresenta il valore vero o falso)

Chiaramente se F(x1,x2,…xn) è soddisfacibile almeno una fra le due formule logiche F(vero,x2,…xn) e F(falso,x2,…xn) deve essere soddisfacibile. Algoritmo è il seguente

```
Trova-assegnazione ((x1, x2, . . . , xn))
if A(x1, x2, . . . , xn)= "no"  (A algoritmo dato per decidere SAT)
    then return "formula non soddisfacibile"
for i  = 1 to n
    do s[i]  = false
    if A(s[1], . . . , s[i], xi+1, . . . , xn) = "non soddisfacibile"
        then s[i]  = true
return (s1, s2, ... sn)
```

L'algoritmo esegue nel caso peggiore n chiamate all'algoritmo A() che decide se una formula è soddisfacibile; quindi ha tempo di esecuzione polinomiale

Esercizi

Supponi che qualcuno vi fornisca un algoritmo A() che dato x in tempo polinomiale decide se x codifica una formula soddisfacibile.

Descrivi come usare questo algoritmo per trovare in tempo polinomiale un'assegnazione di valori alle variabili che rende vera la formula se la formula è soddisfacibile)

Ricorda: noi assumiamo che P≠NP e quindi che non esista un algoritmo che decide in tempo polinomiale se x codifica una formula soddisfacibile.

IMPORTANTE:

- Questo esercizio evidenzia come utilizzare i linguaggi per descrivere la complessità di problemi non è limitativo
- La tecnica di prova può essere estesa anche ad altri problemi. Ad esempio
- Supponi che qualcuno vi fornisca un algoritmo A() che dato x in tempo polinomiale decide se x codifica un grafo colorabile con 3 colori.

  Descrivi come usare questo algoritmo per trovare in tempo polinomiale una colorazione del grafo con tre colori (se una tale colorazione esiste)