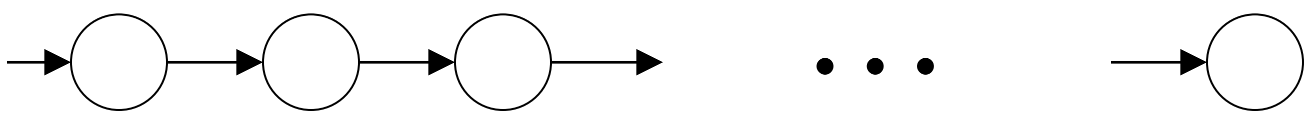
Complessità temporale

Considera una Turing Machine deterministica
M che decide un linguaggio L

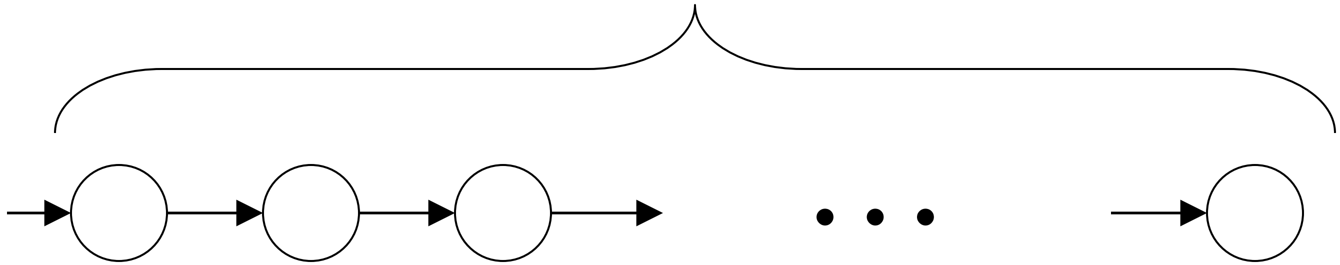
Per ogni stringa w la computazione di M
termina usando una quantità finita
di transizioni



Iniziale
stato

Accetta
o rifiuta w

Tempo di Decisione = #transizioni



stato
Iniziale

accettare
or rigettare w

Consideriamo ora, tutte le stringhe di
Lunghezza n

$T_M(n)$ = massimo tempo richiesto per decidere (calcolare) Una qualsiasi stringa di lunghezza n

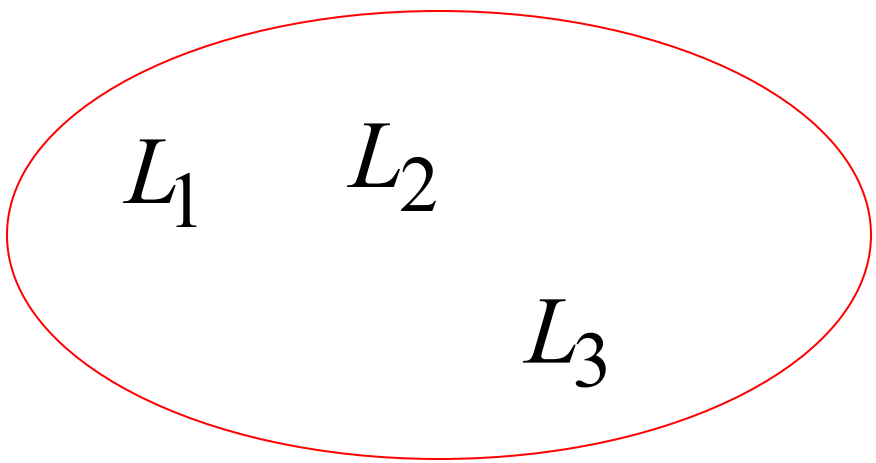
Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

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

When $f(n) = O(g(n))$ we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Classe Complessità temporale : $TIME(T(n))$

Tutti i linguaggi decidibili da una Turing Machine deterministica in tempo $O(T(n))$



Esempio: $L_1 = \{a^n b : n \geq 0\}$

Può essere deciso in tempo $O(n)$

$TIME(n)$

$L_1 = \{a^n b : n \geq 0\}$

Altri esempi nella stessa classe

$$TIME(n)$$

$$L_1 = \{a^n b : \ n \geq 0\}$$

$$\{ab^n aba : \ n,k \geq 0\}$$

$$\{b^n : \ n \text{ è pari}\}$$

$$\{b^n : \ n = 3k\}$$

Esempi nella classe

$$TIME(n^2)$$

$$\{a^n b^n : n \geq 0\}$$

$$\{ww^R : w \in \{a,b\}\}$$

$$\{ww : w \in \{a,b\}\}$$

M_1 = "On input string w:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

In stages 2 and 3, the machine repeatedly scans the tape and crosses off a 0 and 1 on each scan. Each scan uses $O(n)$ steps. Because each scan crosses off two symbols, at most $n/2$ scans can occur. So the total time taken by stages 2 and 3 is $(n/2)O(n) = O(n^2)$ steps.

Esempi nella classe: $TIME(n^3)$

Anche su sipster

CYK algorithm

$L_2 = \{\langle G, w\rangle : w$ è generata da una grammatica context - free $G\}$

Moltiplicazione tra matrici

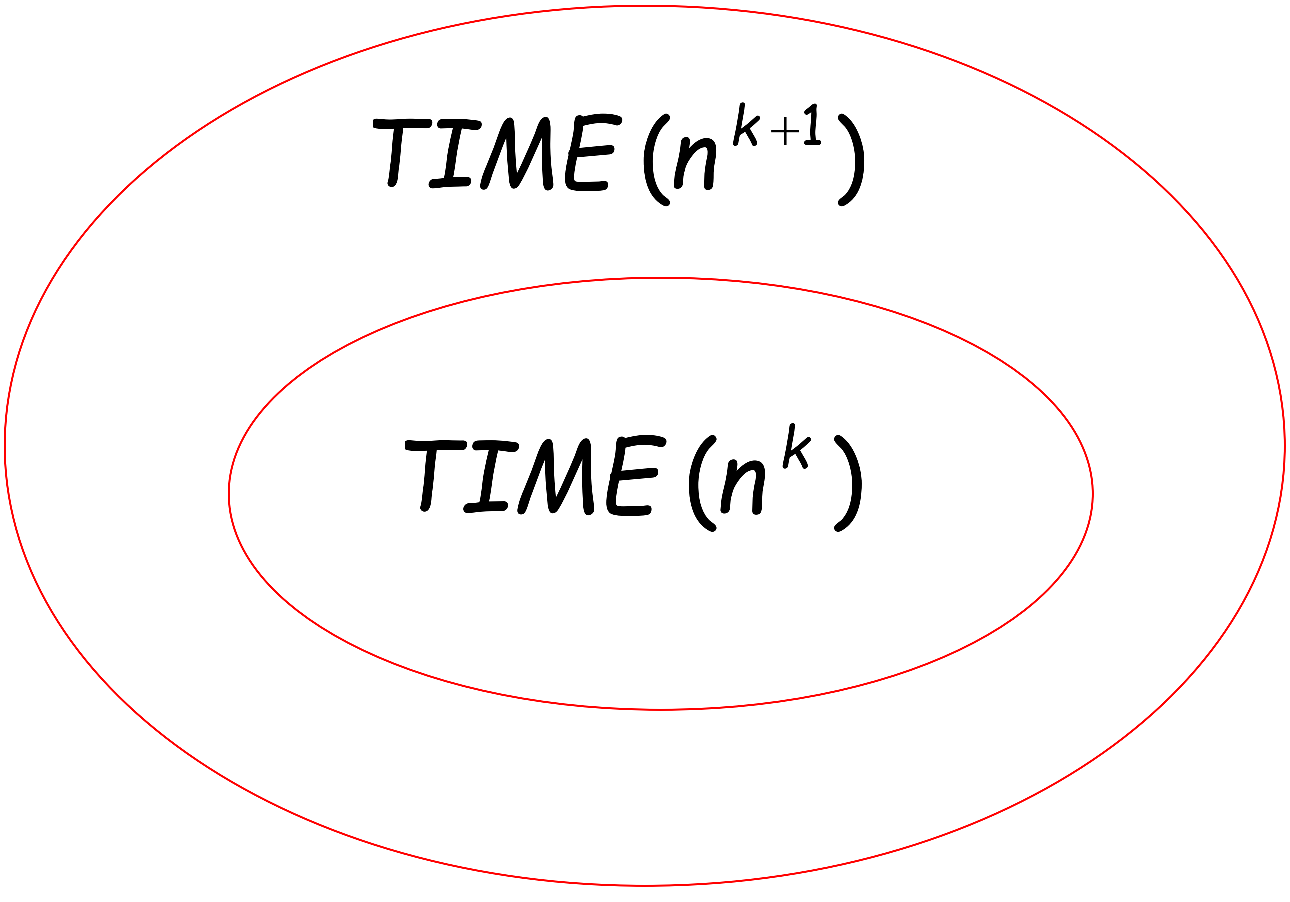
$L_3 = \{\langle M_1, M_2, M_3\rangle : n \times n$ matrices and $M_1 \times M_2 = M_3\}$

algoritmi tempo Polinomiale : $TIME(n^k)$

costante $k > 0$

Rappresentano gli algoritmi trattabili:

Per piccoli k possiamo decidere il risultato velocemente chiaramente: $TIME(n^{k+1}) \supset TIME(n^k)$



architetture

Multitape = singolo tape in tempo quadratico

Anche sul sipster

La Classe di Complessità temporale P

$$P = \bigcup_{k>0} TIME(n^k)$$

Representa:

- Algoritmi che usano tempo polinomiale

- Problemi "trattabili"

Class P

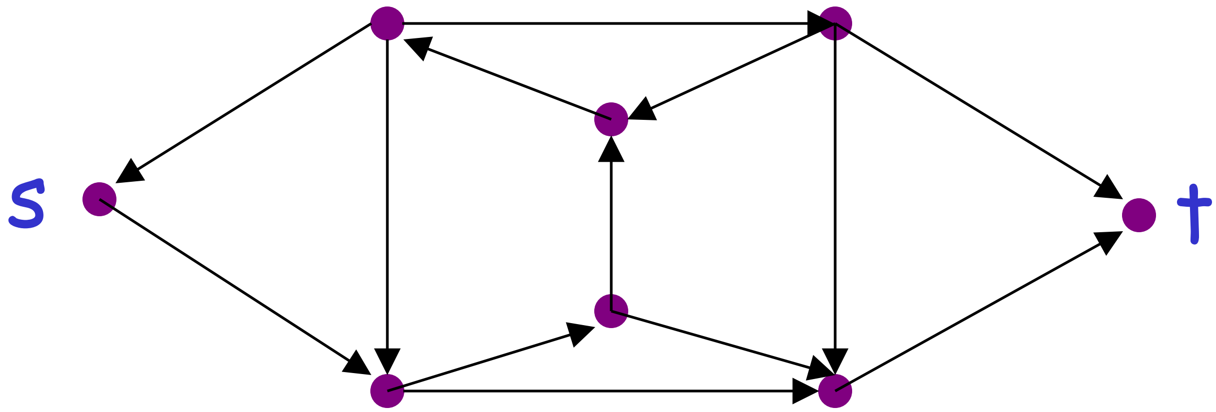
$\{a^n b\}$

$\{a^n b^n\}$

$\{ww\}$

CYK-algorithm

Dato un grafo e due nodi: esiste un cammino da un nodo all'altro?



Dato un grafo e due nodi: esiste un cammino da un nodo all'altro?

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = "On input $\langle G, s, t\rangle$ where G is a directed graph with nodes s and t:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
4. If t is marked, *accept*. Otherwise, *reject*."

Step 1 : 1
Step 4: 1
Step 2,3 : m (numero dei nodi)

Qualche problema non in P

algoritmi calcolabili
in tempo esponenziale

$$TIME(2^{poly(k)})$$

$$TIME(2^{n^k})$$

Rappresentano algoritmi intrattabili

Per alcuni input ci possono volere
secoli per trovare la soluzione

Ricordiamo:

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

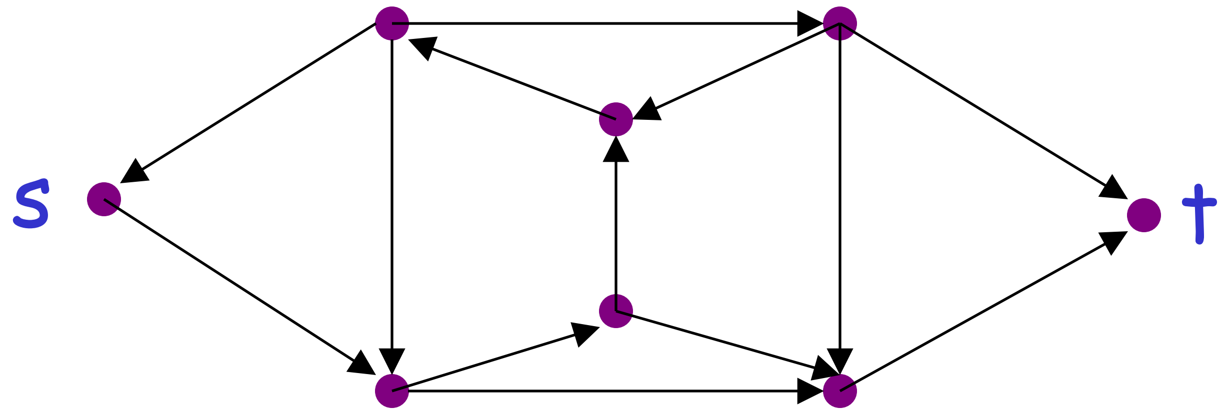
NonDeterminismo = Determinismo in tempo esponenziale (dim anche sipster)

uno

Problema: Cammino Hamiltonian

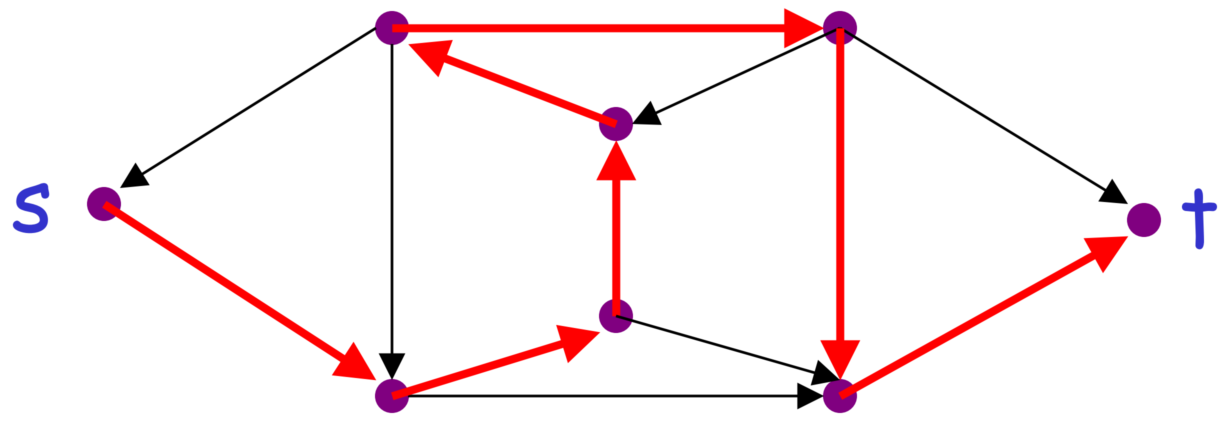
un cammino in un grafo
(orientato o non orientato)
è detto **hamiltoniano** se
esso tocca tutti i vertici
del grafo
una e una sola volta.

Problema del Cammino Hamiltonian



domanda: vi è un Cammino Hamiltoniano da s a t?

tocca tutti i vertici del grafo una e una sola volta.



si

tocca tutti i vertici del grafo una e una sola volta.

Non esistono algoritmi efficienti per la risoluzione del «problema del cammino hamiltoniano», l'unico metodo di risoluzione è rappresentato dall'enumerazione totale, ovvero nell'enumerazione di tutti i possibili cammini sul grafo fino a che non si trovi il cammino voluto.

ab, ba   Permutazione di n elementi

Arriva c

Possibili posizioni: Avanti, Dietro, mezzo(posizione 2);tre nuovi elementi per ogni elemento di partenza

cab, acb,abc, cba, bca, bac (3!)

Arriva d prendiamo per esempio abc

Avanti, Dietro, posizione 2, posizione 3 (Aa2b3cD); sono 4 nuovi elementi per ogni elemento di partenza

Per ogni tripla

6*4= 24=4!

$$TIME(2^{poly(k)})$$

Una soluzione :

Lavorare su tutti i cammini

L = {<G,s,t>: vi è un Cammino Hamiltonian in G da s a t}

n!=nx(n-1)x....

nxnxn...

$$L \in TIME(n!) \approx TIME(2^{n^k})$$

k=2

Permutazione di n elementi

tempo Esponenzale

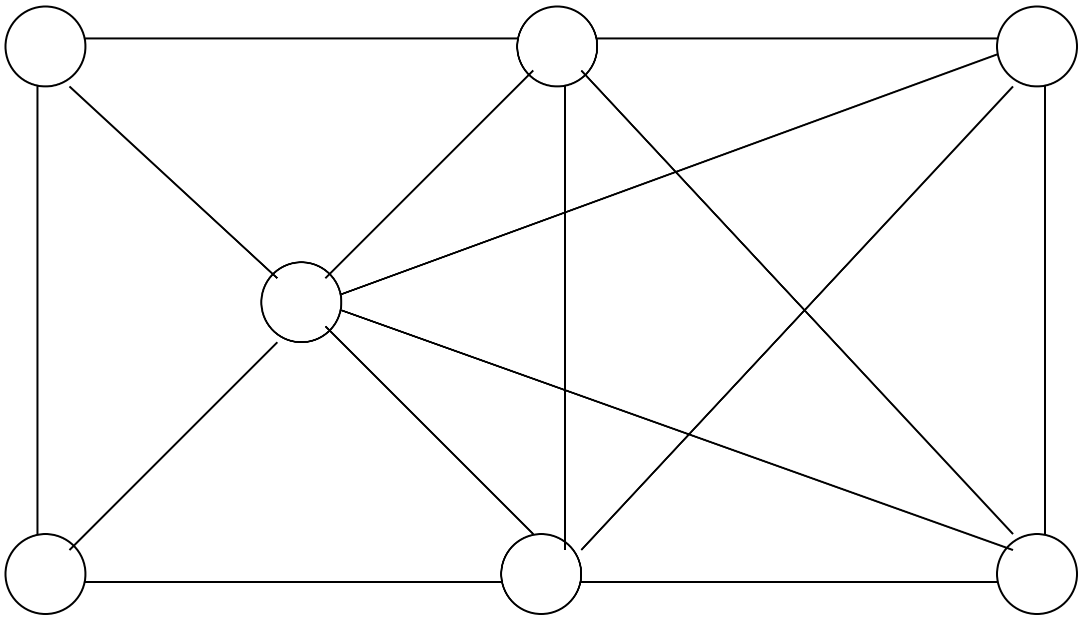
problema Intrattabile

due

problema della cricca
dato un grafo e dato un k

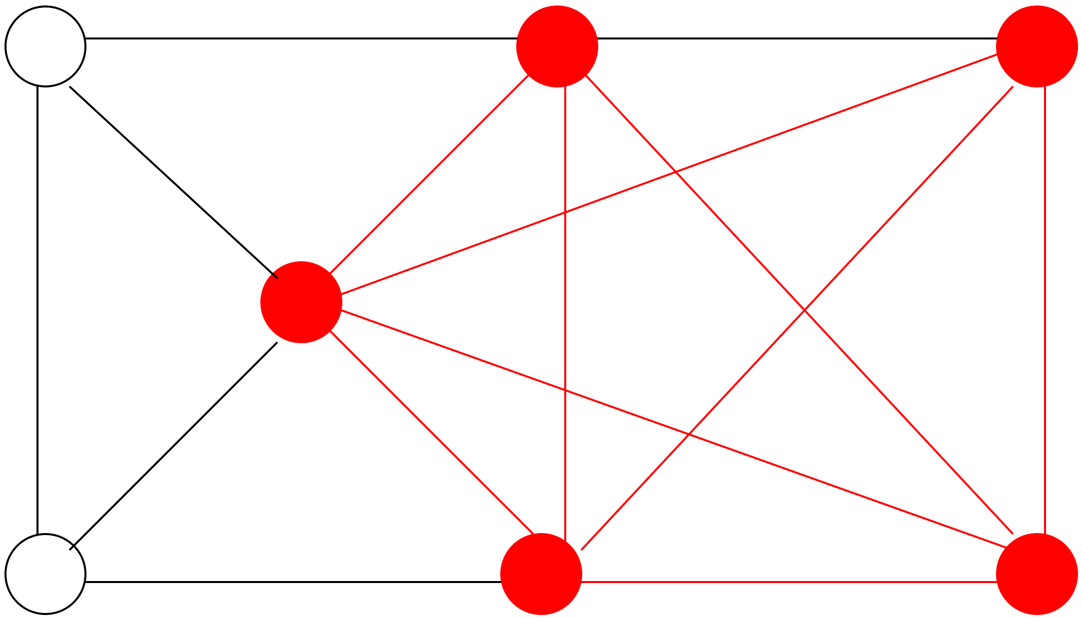
trovare un insieme di k nodi dove ciascun elemento è connesso con tutti gli altri.

Esempio: Il problema della cricca



Esiste una cricca di grado 5?

Il problema della cricca



Esiste una cricca di grado 5.

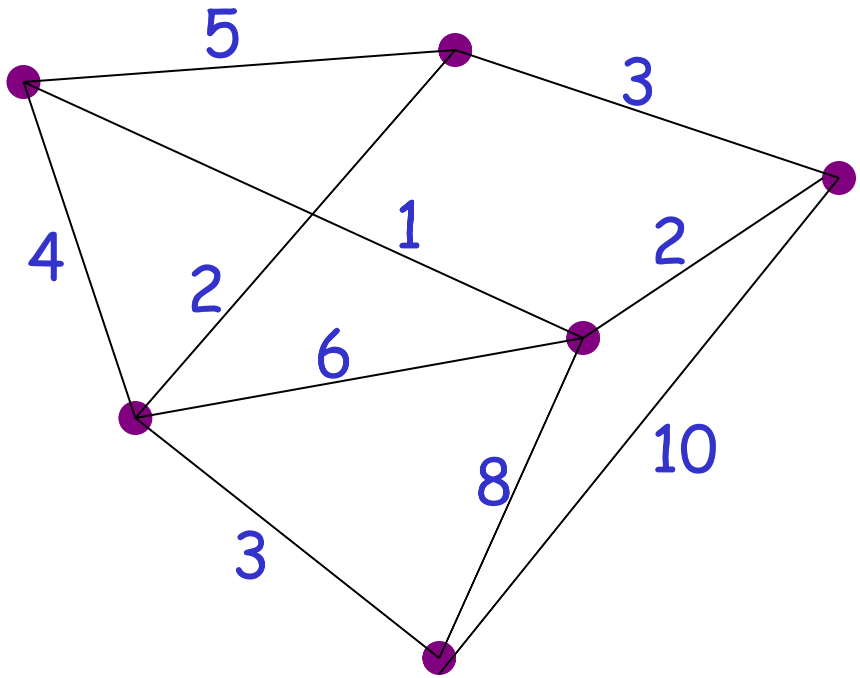
Tutte le permutazioni di 5 elementi e verificare che ogni elemento è connesso con tutti gli altri.

5! X 5!

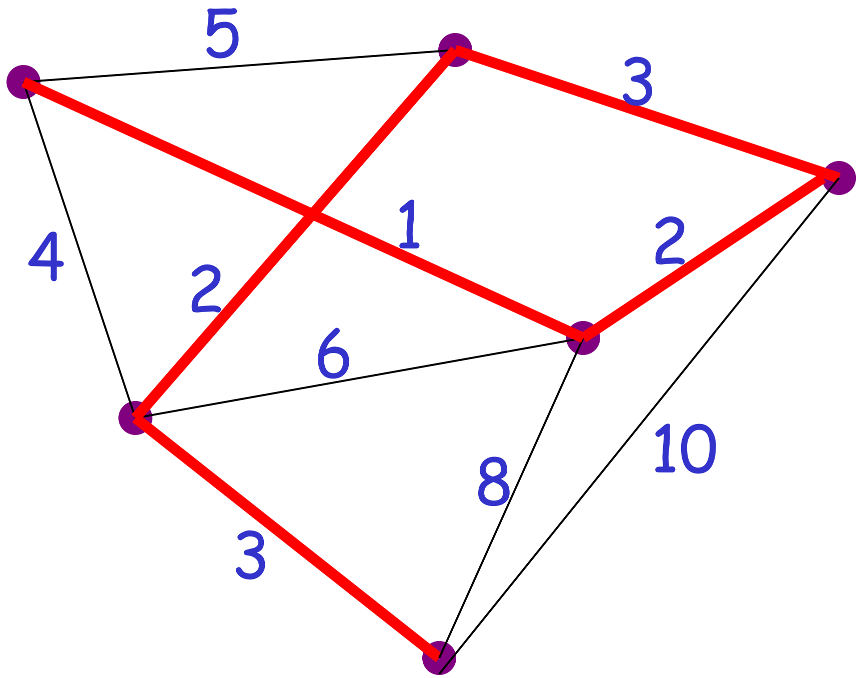
Per precisione 5!x(5-1)x(4-1)…x1

tre

Esempio: commesso viaggiatore



domanda: quale è la via più veloce
Per connettere tutte le città?



quale è la via più veloce
Per connettere tutte le città?

Una soluzione : ricerca tutti i cammini, Hamiltoniani, ovvero tutti i cammini che toccano tutti i vertici una sola volta.

ricorda $L \in TIME(n!) \approx TIME(2^{n^k})$

quindi

L = {shortest hamiltonian paths}

quattro: il Problema della soddisfacibilità

espressioni Booleani in
Conjunctive Normal Form:

$$t_1 \wedge t_2 \wedge t_3 \wedge \cdots \wedge t_k$$ clausole

$$t_i = x_1 \vee \bar{x}_2 \vee x_3 \vee \cdots \vee \bar{x}_p$$

Variabili

domanda: è l'espressione soddisfacibile?

Esempio: $(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_3)$

soddisfacibile: $x_1 = 0, \;\; x_2 = 1, \;\; x_3 = 1$

$$(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_3) = 1$$

Esempio: $(x_1 \vee x_2) \wedge \bar{x}_1 \wedge \bar{x}_2$

Non soddisfacibile

$$(x_1 \vee x_2) \wedge \bar{x}_1$$

soddisfacibile

$$L = \{w : \ w \ \text{soddisfacibile}\}$$

$$L \in TIME(2^{n^k})$$

Algoritmo:

ricerca , in modo esaustivo,
su tutti i possibili valori delle variabili
Tavole di verità, n variabili , 2 ^ n

Non-determinismo: prima definizione

La classe dei linguaggi: $NTIME(T(n))$

Turing Machine Non-Deterministica:
i rami di computazione sono limitati
da un T(n)

Linguaggi decidibili da una mdTuring
non deterministica in tempo $O(T(n))$

Non-determinismo: seconda definizione

A ***verifier*** for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

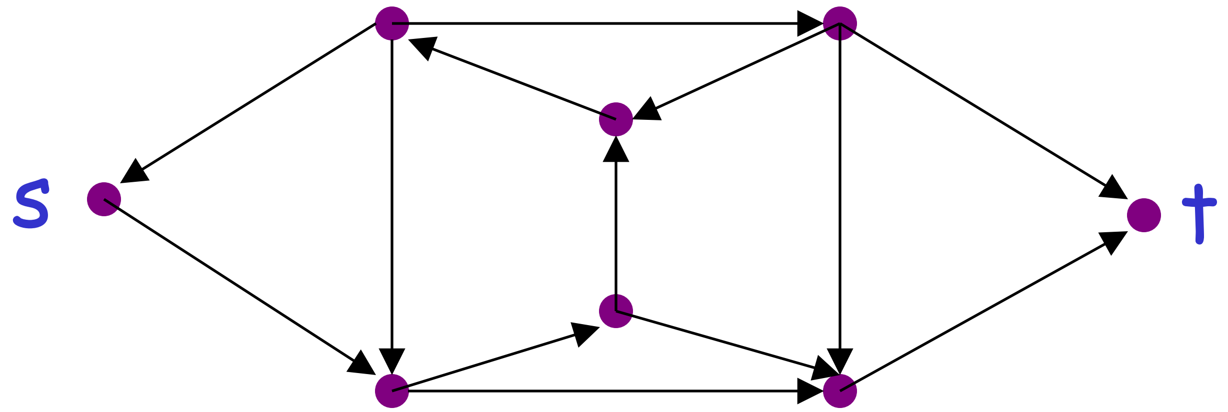
Decide per ogni stringa (w) di lunghezza n con l'aiuto di una stringa c in tempo O(T(n))

uno

Problema: Cammino Hamiltonian

un cammino in un grafo
(orientato o non orientato)
è detto **hamiltoniano** se
esso tocca tutti i vertici
del grafo
una e una sola volta.

Problema del Cammino Hamiltonian



domanda: vi è un Cammino Hamiltoniano da s a t?

tocca tutti i vertici del grafo una e una sola volta.

Problema del Cammino Hamiltoniano

Esempio

Stringa giusta che ci dà il cammino.

due

problema della cricca

dato un grafo e dato un k

trovare un insieme di k nodi dove ciascun elemento è connesso con tutti gli altri.

Esempio

.

problema della cricca, dato un k trovare un insieme di k elementi dove ciascun elemento è connesso con tutti gli altri.

Stringa giusta che ci dà la cricca.

il Problema della soddisfacibilità

Valori di verità giusti

quattro: il Problema della soddisfacibilità

espressioni Booleani in
Conjunctive Normal Form:

$$t_1 \wedge t_2 \wedge t_3 \wedge \cdots \wedge t_k$$ clausole

$$t_i = x_1 \vee \bar{x}_2 \vee x_3 \vee \cdots \vee \bar{x}_p$$

Variabili

domanda: è l'espressione soddisfacibile?

Esempio

•

il Problema della soddisfacibilità

Valori di verità giusti

Due definizioni sono uguali?

Verificatore $\Longrightarrow$ NdT

Assumiamo che V è limitata da n^k.
Prendiamo tutte le stringhe di lunghezza n^k :
NdT: Su input w di lunghezza n:

1. Non deterministicamente seleziona stringa c di lng al massimo n^k;
2. Calcola V su (w,c);
3. Se V accetta allora *accetta*
   altrimenti *rifiuta*

NdT $\Rightarrow$ Verificatore

Trovare la stringa

Ricorda che se NdT accetta s allora esiste un cammino, c, che ci porta all'accettazione.

Sia Ndt N costruiamo un verificatore V

V: input (w,c)

1. Simula NdT sull'input w scegliendo il cammino indicato da c.
2. Se V accetta allora accetta,
   altrimenti rigetta

Esempio: $L = \{ww\}$

algoritmo Non-Deterministico
per accettare una stringa ww:

- Usiamo una two-tape Turing machine

- Congetturiamo la metà della stringa e la copiamo w sul secondo nastro

- Compariamo i due nastri

$$L = \{ww\}$$

tempo necessario

Usiamo una two-tape Turing machine

Congetturiamo la metà della stringa e la copiamo w sul secondo nastro $O(|w|)$

Compariamo i due nastri $O(|w|)$

tempo totale: $O(|w|)$

$$NTIME(n)$$

$$L = \{ww\}$$

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

$$L \in NP$$

Il problema della soddisfacibilità è un NP - Problem

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

Tempo necessario per n variabili:

•Congetturiamo un assegnazione di valore alle variabili $O(n)$

•Verifichiamo che questo assegnamento sia soddisfacibie $O(n)$

Total tempo: $O(n)$

In modo simile possiamo definire

$$NTIME(T(n))$$

Per ogni funzione temporale : $T(n)$

Esempi: $NTIME(n^2), NTIME(n^3), \ldots$

La classe NP
Non-Deterministic Polynomial time

$$NP = \cup NTIME(n^k)$$

Per ogni k

$$P = \cup TIME(n^k)$$

osservazione:

$$P \subseteq NP$$

Deterministic
Polynomial

Non-Deterministic
Polynomial

Problema aperto: $P = NP\,?$

Non conosciamo la risposta

qui

Problema aperto: $P = NP\,?$

Esempio: il problema della sodisfacibilità
ha un algoritmo deterministico
che lo risolva in tempo polinomiale?

Non conosciamo la risposta

Fine cap.

NP-Completezza

Un problema è NP-complete se:

- E' in NP
- Ogni NP problema si può ridurre
- al problema di partenza

(in tempo polinomiale)

Osservazione:

Se possiamo risolvere un problema
NP-complete
in tempo Deterministic Polynomial (P tempo)
Allora avremo che :

$$P = NP$$

Osservazione :

Se proviamo che
non possiamo risolvere un problema
NP-complete
in tempo Deterministic Polynomial (P tempo)
Allora abbiamo:

$$P \neq NP$$

Teorema diCook':

Il problema della soddisfacibilità è NP-complete

Dimostrazione:

Convertire una Non-Deterministic Turing Machine

In una espressione booleana

in (congiuntiva) conjunctive normal form

Altri problemi NP-Complete :

- Il commesso viaggiatore
- Vertex cover
- Hamiltonian Path

Tutti questi problemi si possono ridurre al problema della soddisfacibilità

Osservazione:

sarebbe molto strano che NP-complete problemi sono in P

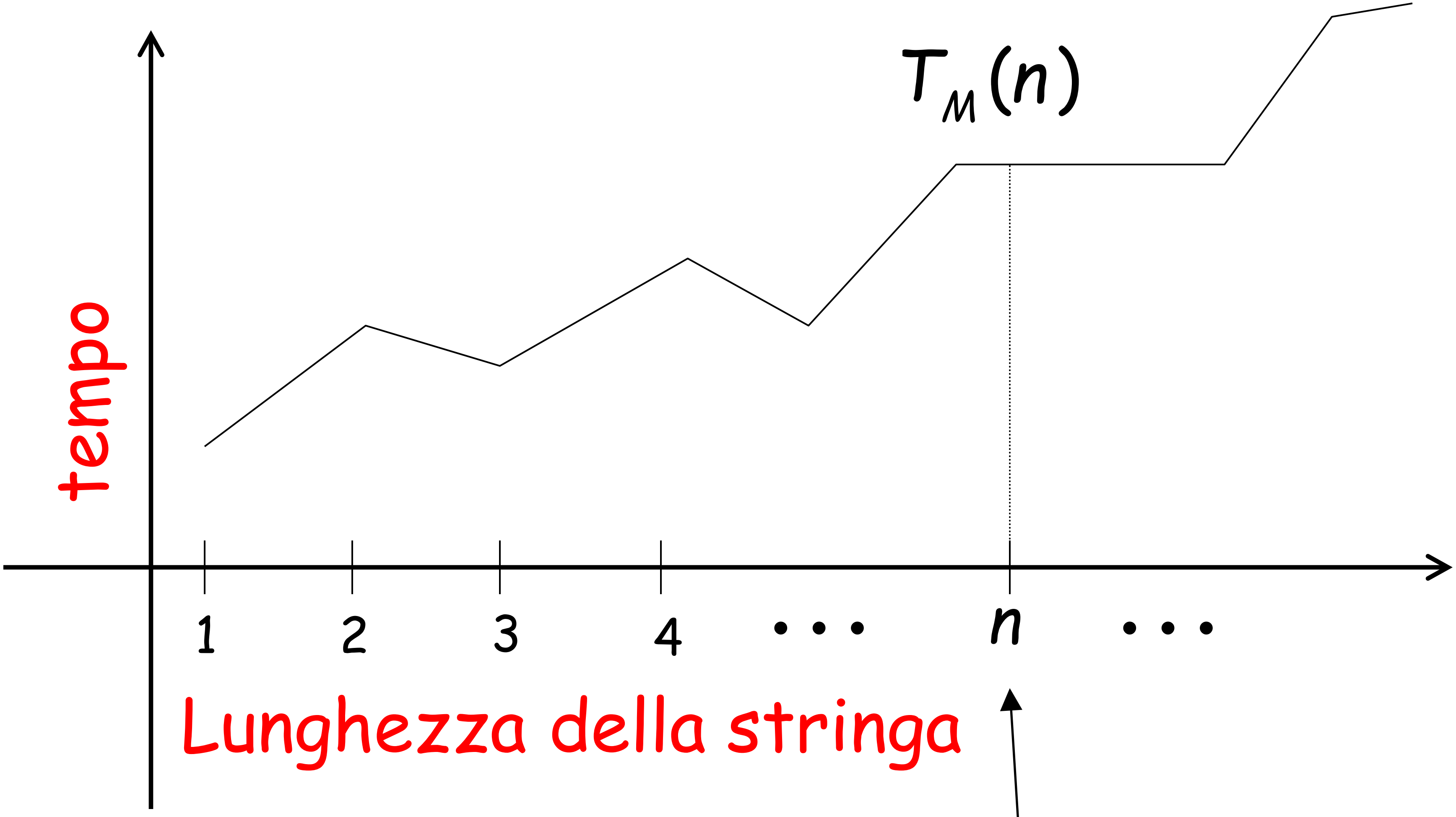
I problemi NP-complete hanno algoritmi in tempo esponenziale

Approssimazioni di questi problemi Sono in P tempo complessità:

Il numero di passi (step)
durante una computazione

spazio complessità:

spazio usato
durante una computazione



Massimo tempo per accettare una stringa di lunghezza n

algoritmi calcolabili
in tempo esponenziale

$$TIME(2^{n^k})$$

$$TIME(2^{2^{poly(k)}})$$

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Rappresentano algoritmi intrattabili

Per alcuni input ci possono volere secoli per trovare la soluzione

tempo algoritmi Non-Deterministic Polynomial :

$$L \in NTIME(n^k)$$

algoritmi

Tempo Polinomiale Non-Deterministico

$$L \in NTIME(n^k)$$

La classe NP

Non-Deterministic Polynomial time

Per ogni k

$$NP = \cup NTIME(n^k)$$

La classe P

Deterministic Polynomial time

Per ogni k

$$P = \cup TIME(n^k)$$

Linguaggi NP-completi

Una funzione $f:A \to B$

è calcolabile in tempo polinomiale se esiste una macchina M che calcola la funzione f in tempo polinomiale

$$w \Longrightarrow w0f(w)$$

Un linguaggio A è riducibile in tempo polinomiale a un linguaggio B , $A \leq B$, se esiste una funzione polinomiale f tale che

per ogni x : $x \in A$ implica $f(x) \in B$

Nota che se $A \leq B$ e B è polinomiale allora anche A è polinomiale.
Sia M che decide B,
Sia f che riduce A a B

Sia x elemento di A

1. Calcola f(x)
2. Calcola M(f(x))

A<=B

Pari<=Dispari
Dispari<=Pari

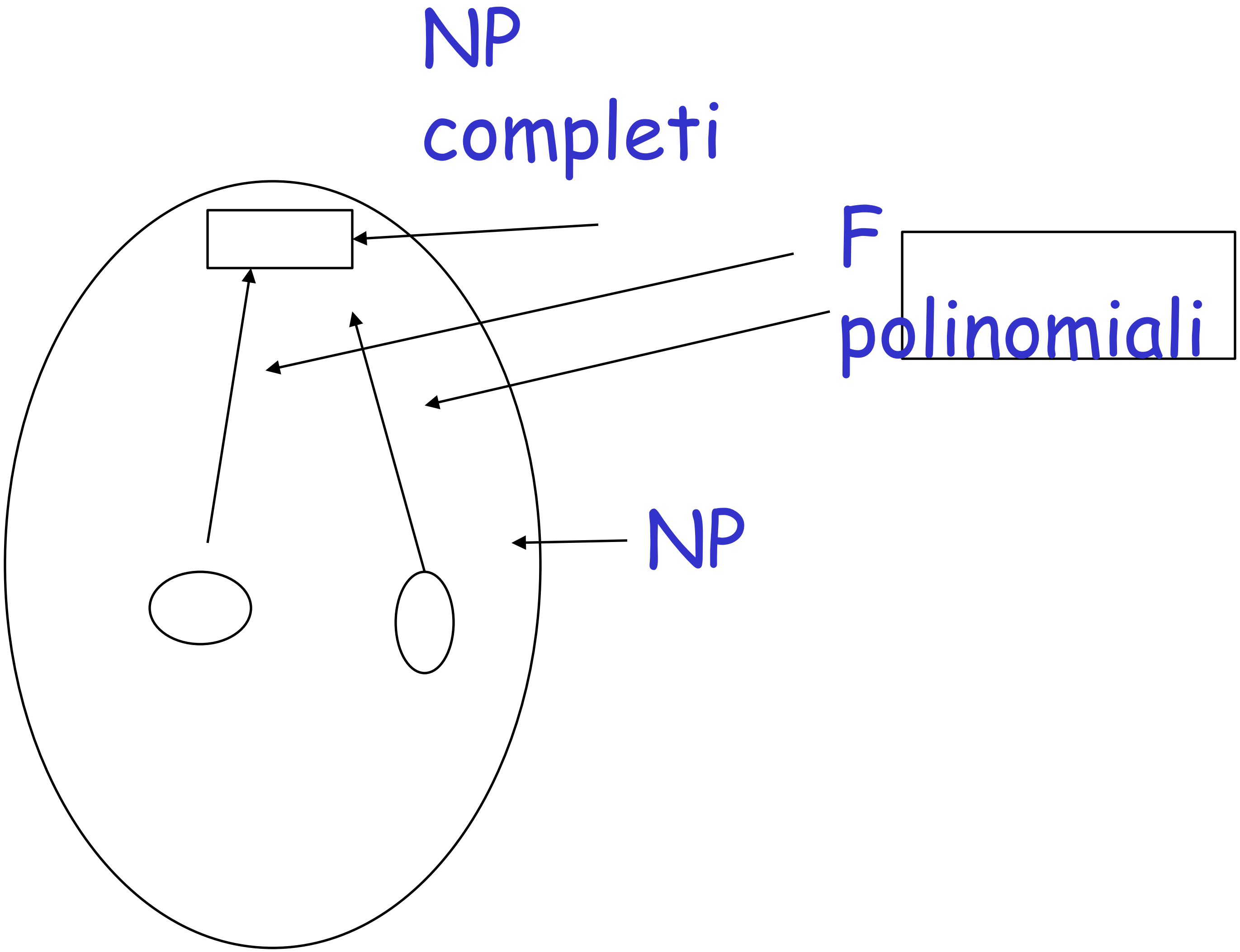
X è elemento di Pari
(div 2 (resto=0))
Dispari
X è elemento di dispari
Non (div 2)

NP-Completezza

Un problema *A* è NP-completo se:

- *A* è in NP
- Ogni problema NP è riducibile ad *A*

(in tempo polinomiale)


NP
completi
F
polinomiali
NP

Osservazione:

Se possiamo risolvere un problema
NP-complete
in tempo Deterministico Polinomiale
allora :

$$P = NP$$

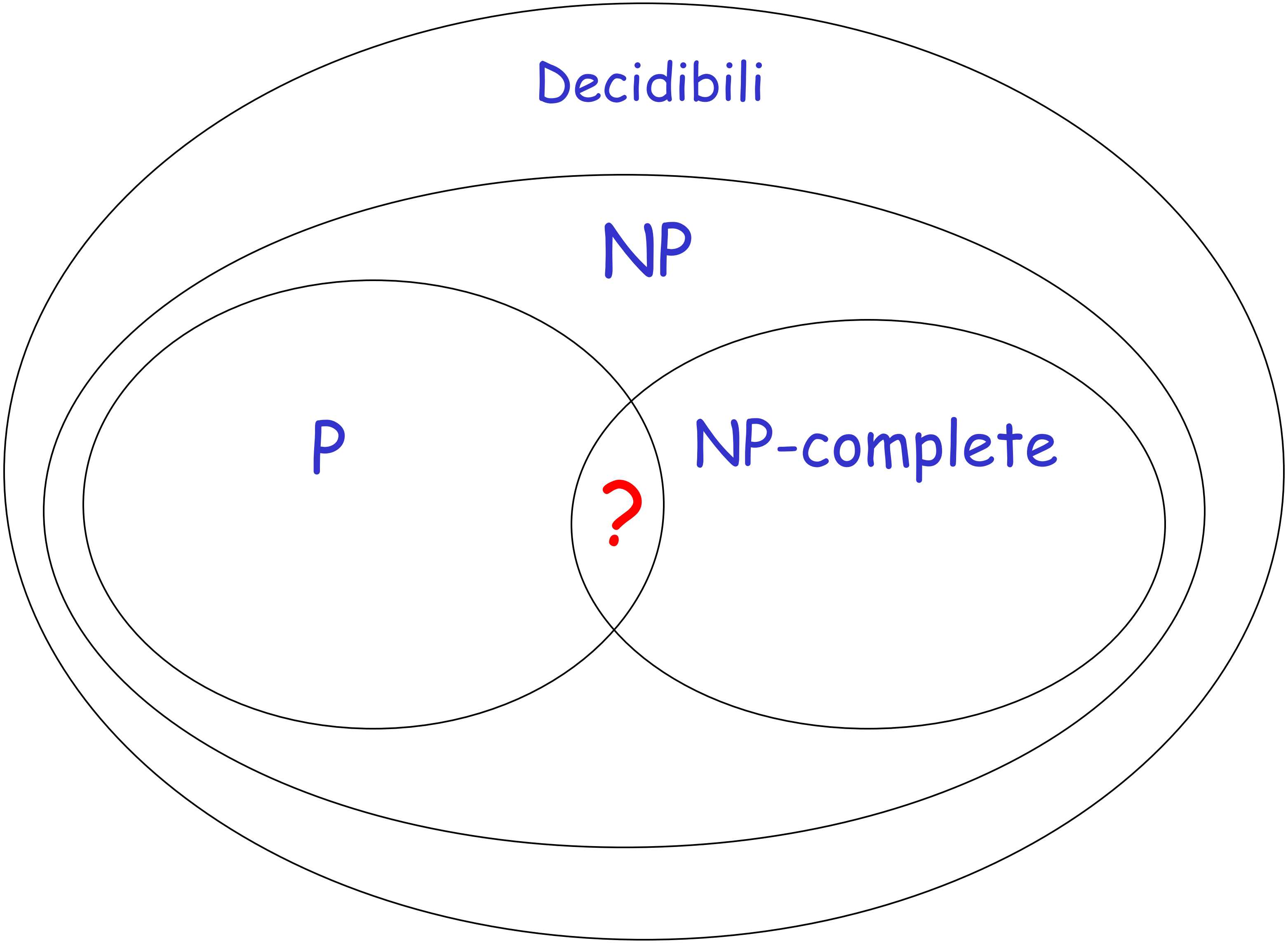
Osservazione :

Se proviamo che
non possiamo risolvere un problema
NP-complete
in tempo Deterministico Polinomiale
Allora :

$$P \neq NP$$

Un Linguaggio L è NP-completo se:

- L è in NP, e
- Ogni Linguaggio in NP può essere ridotto a L in Tempo Polinomiale


Decidibili
NP
P
NP-complete
?

Formule SAT:

literal

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

L: letterale o letterale negato

O: Gruppi di L collegati da $\vee$

Sat: Gruppi di O collegati da $\wedge$

Un Linguaggio NP-completo

Teorema di Cook-Levin :

Linguaggio SAT (satisfiability problem)
è NP-complete

Dim:

Part1: SAT è in NP
(gia provato)

Part2: ridurre tutti i Linguaggi NP
al problema SAT
in Tempo Polinomiale

Sia $L \in NP$, un arbitrario linguaggio

Definiamo una riduzione Polinomiale of L to SAT

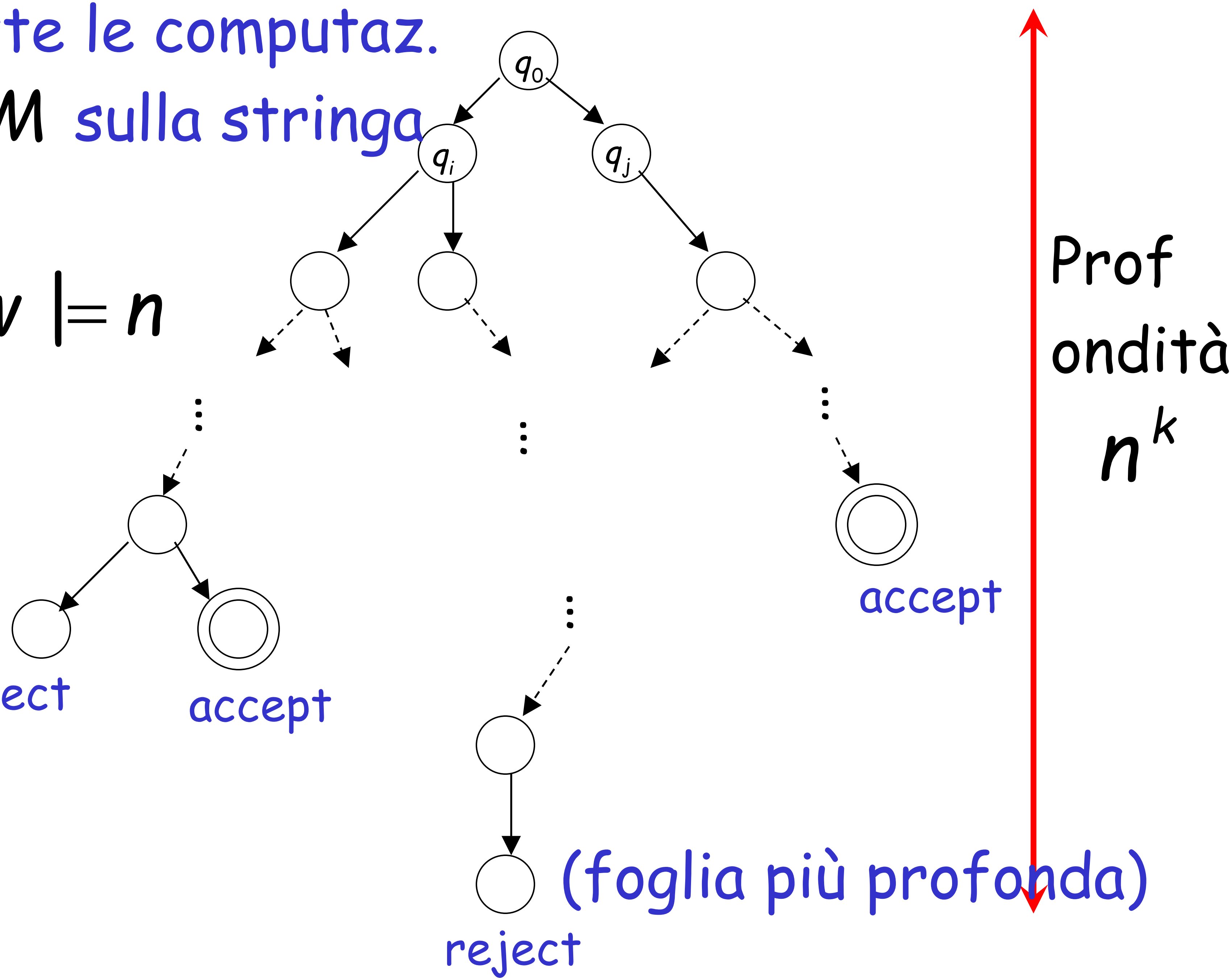
Sia M Macchina di Turing Nondeterministica che decide L in Tempo polinomiale

Per ogni stringa w costruiamo in Tempo Polinomiale una espressione Booleana $\varphi(M,w)$

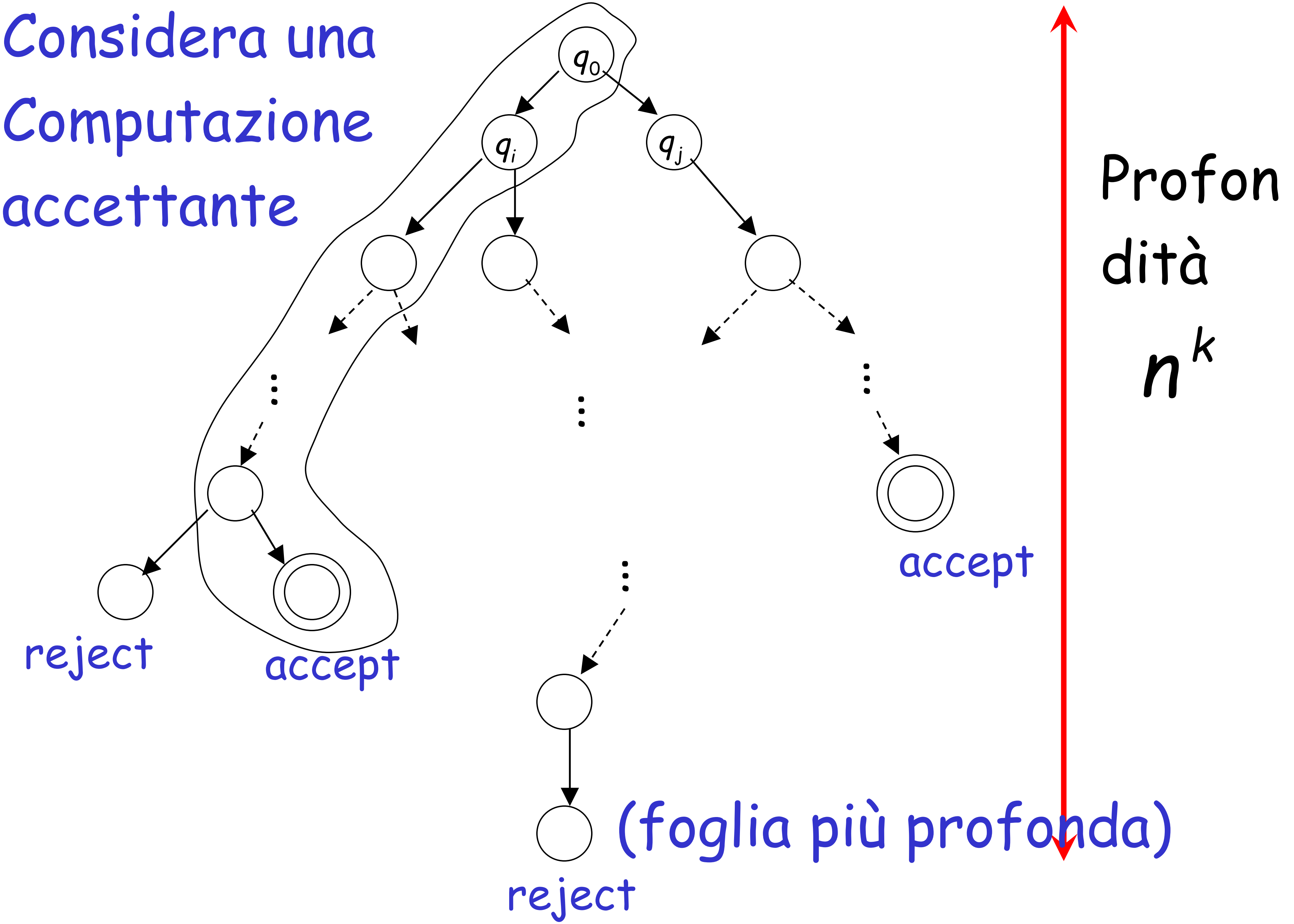
tale che: $w \in L \iff \varphi(M,w)$ è soddisfacibile

Tutte le computaz. of M sulla stringa w

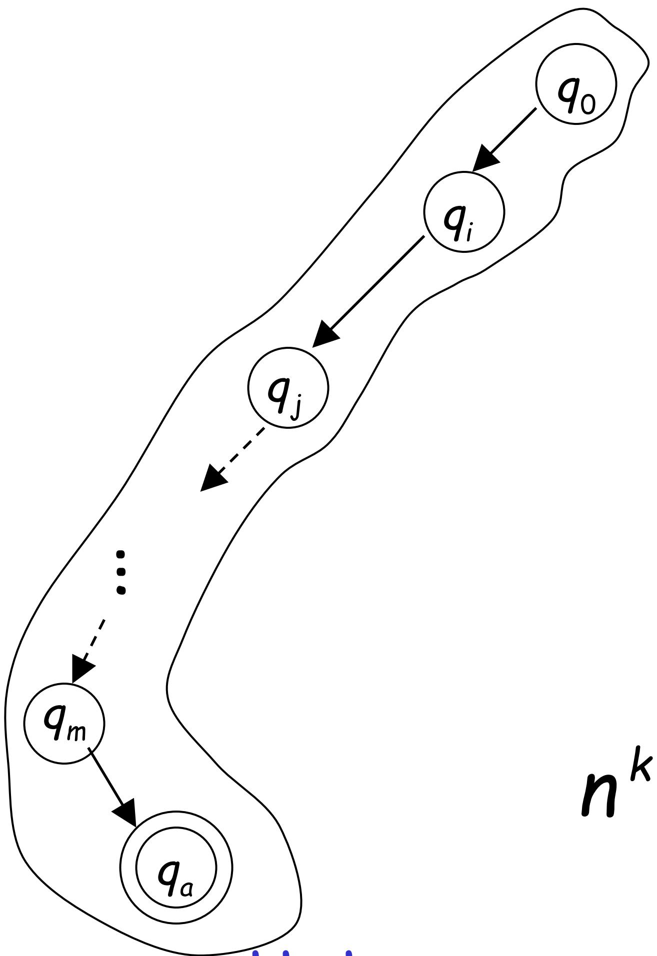
$|w| = n$



Considera una Computazione accettante



Cammino di computazione



Sequenze di configurazioni

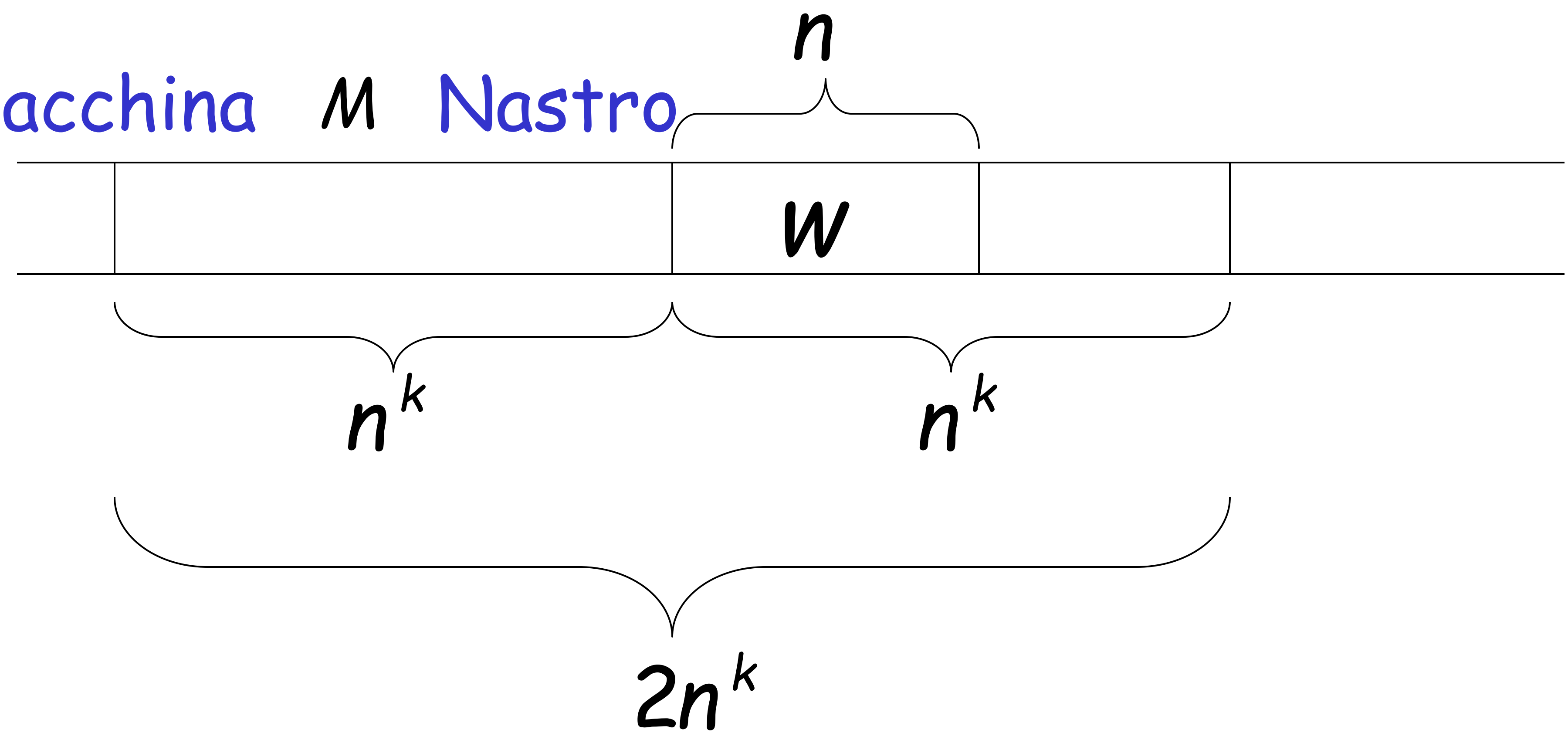
Stato iniziale

$$
\begin{array}{ll}
1: & q_0\ \sigma_1\sigma_2\cdots\sigma_n \\
2: & \succ \sigma'_1\ q_i\ \sigma_2\cdots\sigma_n \\
 & \vdots \\
n^k \geq x: & \succ \sigma'_1\cdots\sigma'_l\ q_a\ \sigma'_{l+1}\cdots\sigma'_{n^k}
\end{array}
$$

Stato accettazione

$$w = \sigma_1\sigma_2\cdots\sigma_n$$

Macchina M Nastro



Massima area di calcolo sul nastro

durante n^k steps (passi) temporali

Tableau delle configurazioni

1 :	#	◊	…	◊	q_0	σ_1	σ_2	…	σ_n	◊	…	◊	#
2 :	#	◊	…	◊	σ_1'	q_i	σ_2	…	σ_n	◊	…	◊	#
⋮													
x :	#	◊	…	σ''	σ_1'	σ_2'	σ_3'	…	q_a	σ_{l+1}'	…	σ_{n^k}	#
n^k :	#	◊	…	σ''	σ_1'	σ_2'	σ_3'	…	q_a	σ_{l+1}'	…	σ_{n^k}	#

Configurazione accettante

Righe identiche

$\longleftarrow n^k \longrightarrow$ $\qquad$ $\longleftarrow n^k \longrightarrow$

$\longleftarrow 2n^k + 3 \longrightarrow$

Alfabeto del Tableau

$$C = \{\#\} \cup \{\text{alfabeta nastro}\} \cup \{\text{insieme degli stati}\}$$
$$= \{\#\} \cup \{\alpha_1, \ldots, \alpha_r\} \cup \{q_1, \ldots, q_t\}$$

Dimensione finita (costante)

Per ogni cella con posizione i,j

E per ogni simbolo nell'alfabeto del tableau $s \in C$

Definiamo la variabile $x_{i,j,s}$

tale che se la cella i,j contiene il simbolo s allora $x_{i,j,s} = 1$ altrimenti $x_{i,j,s} = 0$

Esempio:

n^k+3

1:	$\#$	$\Diamond$	$\cdots$	$\Diamond$	q_0	σ_1	σ_2	$\cdots$	σ_n	$\Diamond$	$\cdots$	$\Diamond$	$\#$
2:	$\#$	$\Diamond$	$\cdots$	$\Diamond$	σ_1'	q_i	σ_2	$\cdots$	σ_n	$\Diamond$	$\cdots$	$\Diamond$	$\#$

$$x_{1,1,\#} = 1 \qquad x_{2,n^k+3,q_i} = 1$$

$$x_{1,1,\Diamond} = 0 \qquad x_{2,n^k+3,\#} = 0$$

$\varphi(M,w)$ è costruito con le variabili $x_{i,j,s}$

$$\varphi(M,w) = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{move}$$

Quando la formula è sodisfatta allora descrive una computazione di accettazione nel tableau della macchina M su input w

φ_{cell} Ci rende sicuri che ogni cella nel tableau contiene esattamente un simbolo

$$\varphi_{\text{cell}} = \bigwedge_{\text{all } i,j} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

Ogni cella contiene almeno un simbolo

Ogni cella contiene al massimo un simbolo

Dimensione di φ_{cell} :

$$\varphi_{\text{cell}} = \bigwedge_{\text{all } i,j} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$$(2n^k + 3)^2 \times (|C| + |C|^2)$$

$$= O(n^{2k})$$

φ_{start} il tableau parte con la configurazione iniziale

$$\varphi_{start} = x_{1,1,\#} \wedge x_{1,2,\Diamond} \wedge \cdots \wedge x_{1,n^k+1,\Diamond}$$
$$\wedge\, x_{1,n^k+2,q_0} \wedge x_{1,n^k+3,\sigma_1} \wedge \cdots \wedge x_{1,n^k+n+2,\sigma_n}$$
$$\wedge\, x_{1,n^k+n+3,\Diamond} \wedge x_{1,2n^k+2,\Diamond} \wedge \cdots \wedge x_{1,2n^k+2,\#}$$

Descrive la configurazione iniziale

Nella riga 1 del tableau

Dimensione di φ_{start} :

$$\varphi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,\Diamond} \wedge \cdots$$

$$\wedge\, x_{1,n^k+1,\Diamond} \wedge x_{1,n^k+2,q_0} \wedge x_{1,n^k+3,\sigma_1} \wedge \cdots$$

$$\wedge\, x_{1,2n^k+2,\Diamond} \wedge x_{1,2n^k+3,\#}$$

$$2n^k + 3 = O(n^k)$$

φ_{accept} Ci da la sicurezza che La computazione raggiunge stato di Accettazione

$$\varphi_{accept} = \bigvee_{\substack{\text{all } i,j \\ \text{all } q \in F}} x_{i,j,q}$$

Stati di accettazione

Uno stato di accettazione deve apparire
Da qualche parte nel tableau

Size of φ_{accept} :

$$\varphi_{accept} = \bigvee_{\substack{\text{all } i,j \\ \text{all } q \in F}} x_{i,j,q}$$

$$\downarrow$$

$$(2n^k + 3)^2 = O(n^{2k})$$

φ_{move} Ci rende sicuri che il tableau ci da una sequenza valida di configurazioni

φ_{move} è espresso in termini di windows legali

Tableau

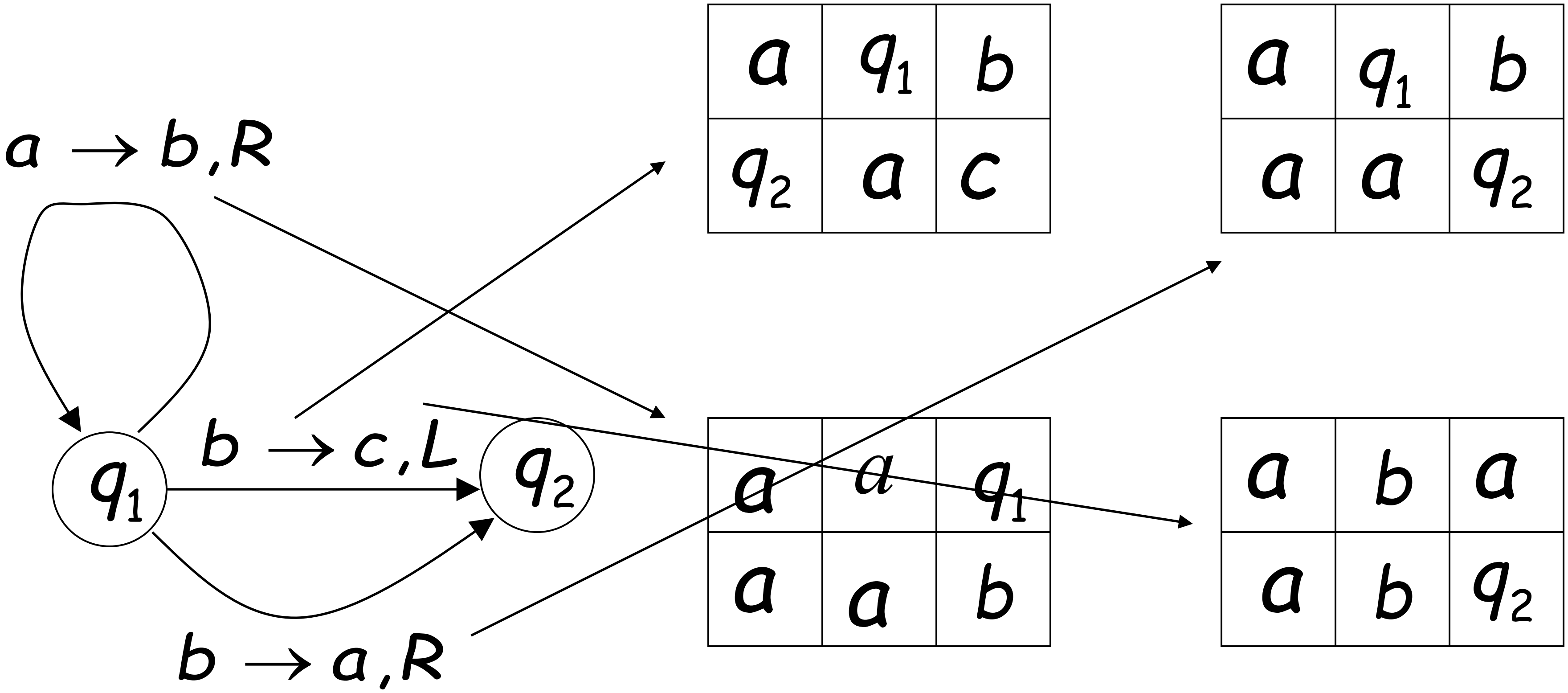
Per ogni i,j, elemento centrale superiore

Window

a	q_1	b
q_2	a	c

2x6 area di celle

Possibili windows Legali



a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

a	b	a
a	b	q_2

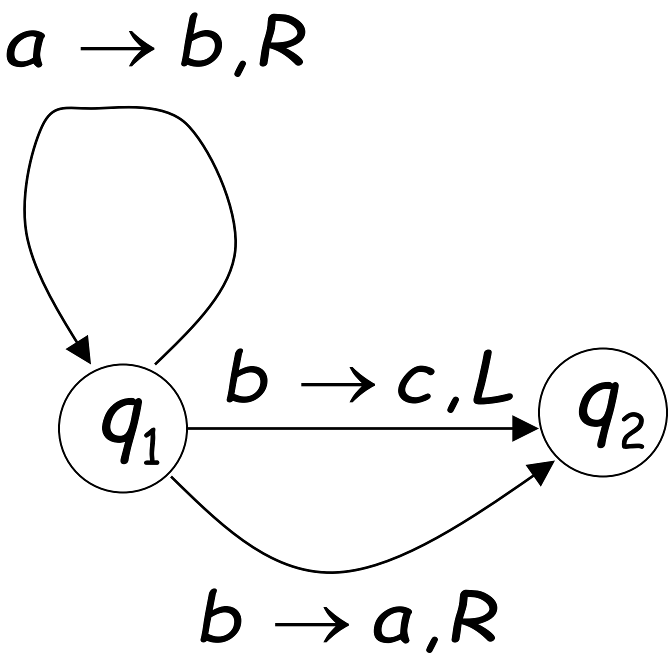
window Legali obbediscono alle transizioni

Altre finestre legali

#	b	a
#	b	a

b	b	b
c	b	b

Possibili window illegali



a	b	a
a	a	a

La b in a?

a	q_1	b
q_1	a	a

Se $\delta(q_1,b) = (q_1,c,R)$

b	q_1	b
q_2	b	q_2

Due stati

$$\begin{array}{c|c|c|c|}
 & j & & \\
\hline
i & a & q_1 & b \\
\hline
 & q_2 & a & c \\
\hline
\end{array}$$

(è legale)

Formula:

$$x_{i,j,a} \wedge x_{i,j+1,q_1} \wedge x_{i,j+2,b}$$

$$\wedge\, x_{i+1,j,q_2} \wedge x_{i+1,j+1,a} \wedge x_{i+1,j+2,c}$$

$$\varphi_{move} = \bigwedge_{\text{all } i,j} (\text{window } (i,j) \text{ is legal})$$

window (i,j) è legale:

$$\begin{array}{c|c|c|c|} & j & & \\ \hline i & a & q_1 & b \\ \hline & q_2 & a & c \\ \hline \end{array} \qquad \begin{array}{c|c|c|c|} & j & & \\ \hline i & a & q_1 & b \\ \hline & a & a & q_2 \\ \hline \end{array} \qquad \begin{array}{c|c|c|c|} & j & & \\ \hline i & a & a & q_1 \\ \hline & a & a & b \\ \hline \end{array} \qquad \cdots$$

((is legal) e (is legal) e (is legal))

Tutte le possibili celle legali nella posizione (i, j)

Dimensione di φ_{move} :

Dimensione di una formula per una window Legale in una cella i,j: 6

$\times$ Numero di possibili legal window in una cella i,j: al max $|C|^6$

$\times$ Numero di possibili celle: $(2n^k+3)n^k$

$$= \quad 6 \cdot |C|^6 \cdot (2n^k+3)n^k = O(n^{2k})$$

Dimensione di $\varphi(M,w)$:

$$\varphi(M,w) = \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}}$$

$$O(n^{2k}) + O(n^{k}) + O(n^{2k}) + O(n^{2k})$$

$$= O(n^{2k})$$

Costruita in Tempo $O(n^{2k})$

Quindi Polinomiale in n

$$\varphi(M,w) = \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}}$$

Abbiamo che:

$$w \in L \iff \varphi(M,w) \text{ is satisfiable}$$

poichè,

$$w \in L \Leftrightarrow \varphi(M,w) \text{ is satisfiable}$$

e

$\varphi(M,w)$ è costruita in Tempo Polinomiale

$$\Downarrow$$

L è riducibile in tempo Polinomiale a SAT

END OF Dim

Osservazione 1:

La $\varphi(M,w)$ formula può essere convertita in CNF (conjunctive normal form) formula in Tempo Polinomiale

$$\varphi(M,w) = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{move}$$

gia CNF

NOT CNF

Ma può essere convertita in CNF (distribuitività)

leggi: Distributività

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$$

$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

Osservazione 2:

La $\varphi(M,w)$ formula può essere Convertita in una formula 3CNF in Tempo Polinomiale

$$(a_1 \vee a_2 \vee \cdots \vee a_l)$$

Conv
erti

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee z_l)$$

Nuove z

3CNF formula:

literal

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \underbrace{\vee \overline{x_5} \vee}_{\text{clause}} x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

Ogni clausola ha tre letterali

Linguaggio:

3CNF-SAT $= \{ \; w \; : \; w$ è una formula 3CNF soddisfacibile $\}$

CNF-SAT e
3CNF-SAT sono
Linguaggi NP-completi

(sappiamo che sono NP Linguaggi)

Esempio di riduzione Tempo-Polinomiale:

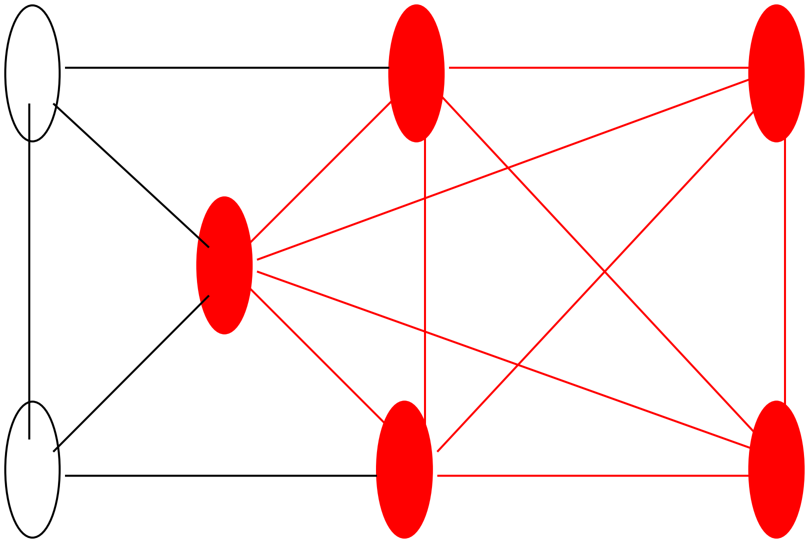
Ridurremo il problema

3CNF-satisfiability

al

problema CLIQUE

Una 5-cricca (Clique) nel grafo G



Linguaggio:

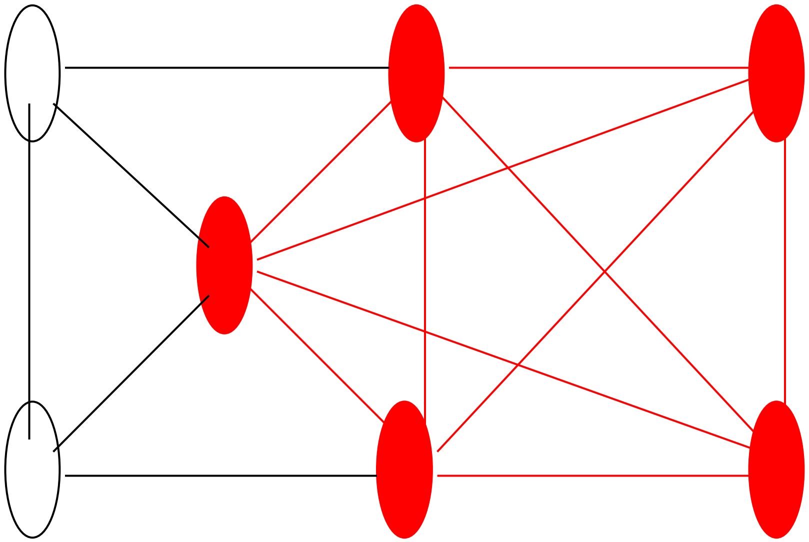
CLIQUE $= \{< G, k >:$ grafo G contiene un k -clique$\}$

Teorema: 3CNF-SAT è riducibile
In Tempo Polinomiale
a CLIQUE

Dim: Una riduzione in Tempo Polinomiale
da un problema all'altro

Transformare una formula in un grafo

Una 5-cricca (Clique) nel grafo G



Linguaggio:

CLIQUE $= \{< G, k >:$ grafo G contiene un k -clique$\}$

Teorema: 3CNF-SAT è riducibile
In Tempo Polinomiale
a CLIQUE

Dim: Una riduzione in Tempo Polinomiale
da un problema all'altro

Transformare una formula in un grafo

3CNF formula:

literal

$$(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\underbrace{x_3 \lor \overline{x_5} \lor x_6}_{\text{clause}}) \land (x_3 \lor \overline{x_6} \lor x_4) \land (x_4 \lor x_5 \lor x_6)$$

Ogni clausola ha tre letterali

Linguaggio:

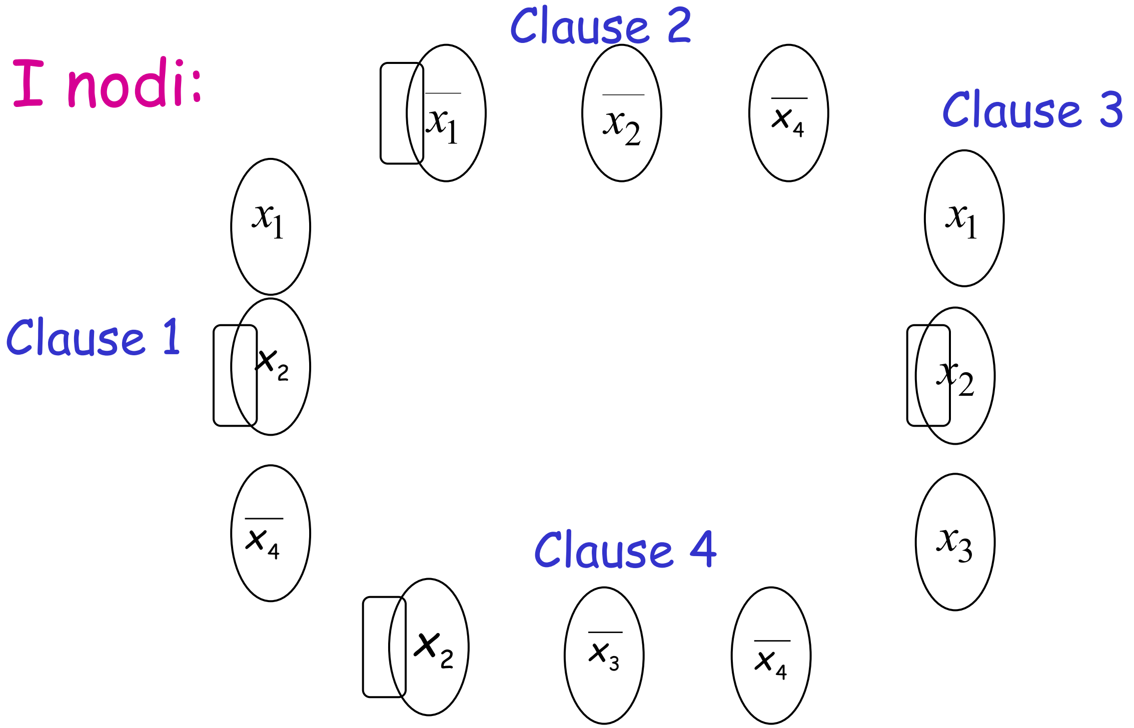
3CNF-SAT $= \{\, w : w$ è una formula 3CNF soddisfacibile $\}$

Transformare una formula in un grafo

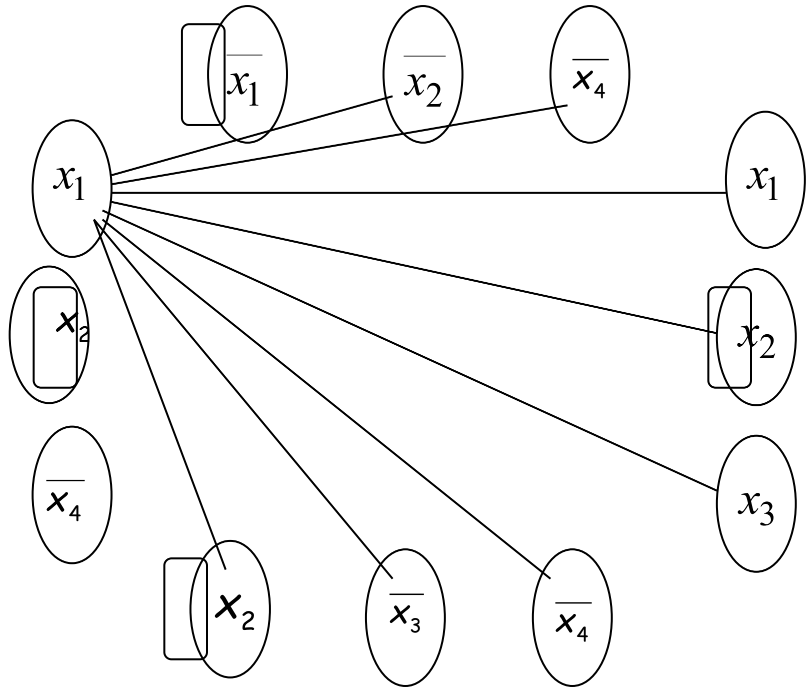
Esempio:

$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$

Creare I nodi:



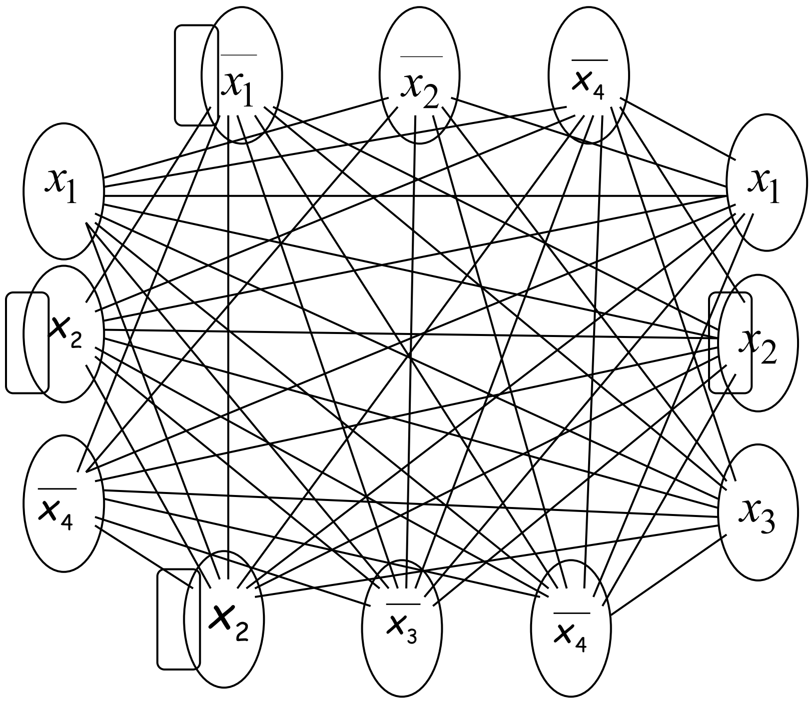
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Addizionare un arco da un letterale ξ

A ogni altro letterale in ogni clausola salvo a $\overline{\xi}$

$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Grafo risultante

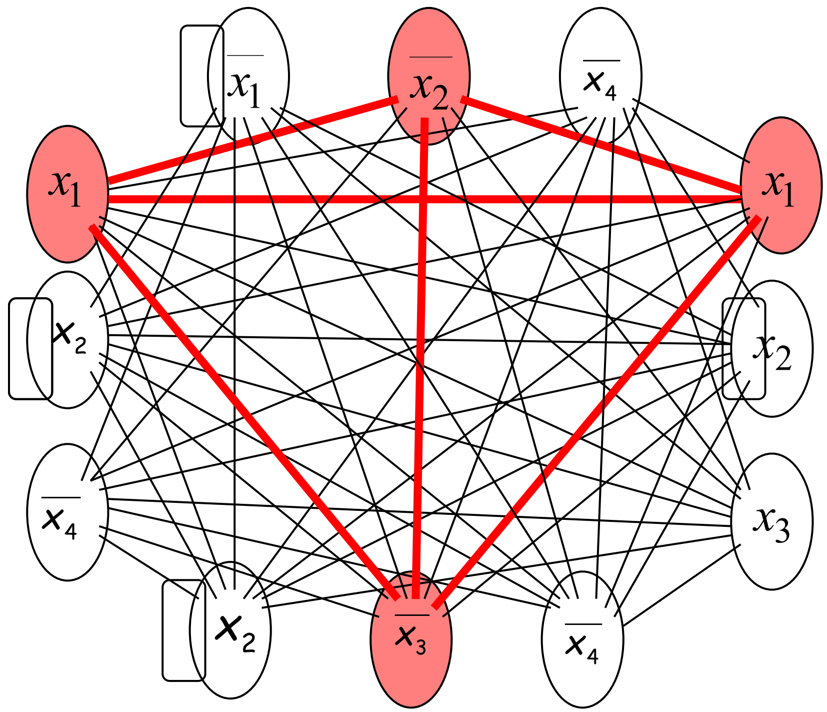
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 1$



La formula è sodisfacibile se e solo se il grafo ha un 4-clique

End of Dim

Teorema:

If: a. Linguaggio A is NP-complete

b. Linguaggio B is in NP

c. A è riducibile Tempo Polinomiale a B

Then: B is NP-complete

Dim:

Ogni linguaggio L in NP

È riducibile in Tempo Polinomiale a A

allora, L è riducibile in Tempo Polinomiale a B

(somma di due riducibilità Polinomiali dà una riduzione Polinomiale)

Corollario: CLIQUE è NP-complete

Dim:

a. 3CNF-SAT è NP-complete

b. CLIQUE è in NP (gia visto)

c. 3CNF-SAT è riducibile Polinomiale a CLIQUE (gia visto)

Applichiamo il teorema precedente

A=3CNF-SAT e B=CLIQUE

Fine chap

Osservazione:

SE si dimostra che un Linguaggio
NP-complete
è in P allora:

$$P = NP$$

Complessità spaziale
o meglio
Complessità nello spazio

- Considera una Turing Machine <u>deterministica</u>
- M che <u>decide</u> un linguaggio L

La complessità spaziale di M (deterministica) è una funzione f:N->N, dove f(n) è il numero massimo di celle che M usa su ogni input di lunghezza n.

Per ogni stringa w la computazione di M(w) termina usando una quantità finita di spazio f(n), (senza limitazione di tempo)

- Considera una Turing Machine non deterministica
- M che <u>decide</u> un linguaggio L

La complessità spaziale di M è una funzione f:N->N, dove f(n) è il numero massimo di celle che per ogni ramo della computazione M usa su ogni input di lunghezza n.

Consideriamo tutte le stringhe di
Lunghezza n e T una funzione da N->N

$Space_M(T(n))$=Linguaggi decidibili (calcolabili) in spazio O(T(n)) deterministicamente per una qualsiasi stringa di lunghezza n

$NSpace_M T((n))$ =Linguaggi decidibili (calcolabili) in spazio O(T(n)) non deterministicamente per una qualsiasi stringa di lunghezza n

Esempio: $L_1 = \{a^n b : n \geq 0\}$

Può essere deciso in spazio $O(n)$

Altri esempi nella stessa classe

$$Space(n)$$

$$L_1 = \{a^n b : \ n \geq 0\}$$

$$\{ab^n aba : \ n,k \geq 0\}$$

$$\{b^n : \ n \text{ is even}\}$$

$$\{b^n : \ n = 3k\}$$

Esempi nella classe

$$Space(n\ )$$

$$\{a^n b^n : n \geq 0\}$$

$$\{ww^R : w \in \{a,b\}\}$$

$$\{ww : w \in \{a,b\}\}$$

Macchine calcolabili
in spazio Polinomiale
deterministico .

$$Space(O(n^k))$$

costante $k > 0$

Macchine calcolabili
in spazio Polinomiale
Non deterministico.

$$NSpace(O(n^k))$$

costante $k > 0$

chiaramente:

$$Space(n^{k+1}) \supset Space(n^k)$$

$$NSpace(n^{k+1}) \supset NSpace(n^k)$$

La Classi di Complessità spaziale

$$PSpace = \bigcup_{k>0} PSpace(n^k)$$

Rapresenta:

- Algoritmi deterministici che usano spazio polinomiale
  - Problemi "trattabili nello spazio"

$$NPSpace = \bigcup_{k>0} NPSpace(n^k) \qquad NP$$

Rapresenta:

- Algoritmi non deterministici che usano spazio polinomiale tempo complessità:

Il numero di passi (step)
durante una computazione

spazio complessità:

spazio usato
durante una computazione

Ricorda che una macchina di Turing deterministica M simula una macchina di Turing M' non deterministica con una complessità di tempo esponenziale rispetto alla complessità di M.

.

Teorema di Savitch: Per ogni funzione **f: N→R⁺**, dove **f(n)≥n**, abbiamo
NSPACE(f(n)) ⊆SPACE(f^2 (n)).

una MdT deterministica può simulare una TM non deterministica usando soltanto una piccola parte di spazio aggiuntivo.

Questo è dovuto al fatto che lo spazio può essere riutilizzato, mentre il tempo no.
Se potessimo utilizzare lo stesso concetto con il tempo avremmo una prova che P=NP

Teorema di Savitch:

Per ogni funzione **f: N→R⁺**, dove **f(n)≥n**, abbiamo

$$\mathbf{NSPACE(O(f(n))) \subseteq SPACE(O(f^2(n)))}.$$

teorema di Savitch dimostra che nello spazio la complessità di questa simulazione, non determinismo -> determinismo, aumenta in modo al più quadratico.

Si consideri una NMdT (non deterministica) calcolata in **spazio f(n).**

Cerchiamo di simularla il comportamento con una MdT (deterministica) con spazio polinomiale f^2**(n).**

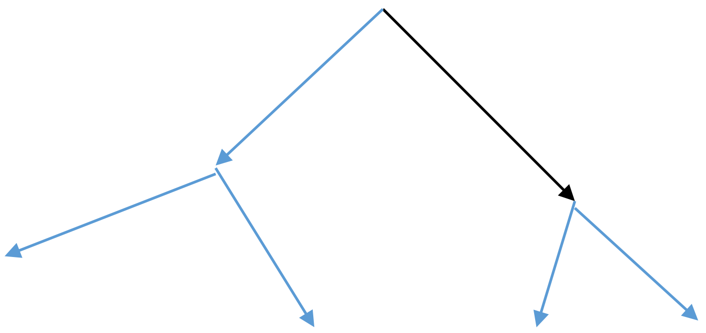
Un primo tentativo potrebbe essere quello di simulare tutti i possibili rami di computazione (di lunghezza finita) di NMdT, ma questo non permette di sovrascrivere l'area di memoria utilizzata, perché occorre tenere traccia delle scelte nondeterministiche fatte su un ramo specifico per scegliere un prossimo ramo.

1, 2 11,12, 111,112,121, 122 …….

Ogni passo deve tenere traccia del cammino percorso. Nota che se usiamo uno spazio f(n) il tempo che possiamo usare su quello spazio è $2^{O(f(n))}$ step (perché?) ed ogni step può essere una scelta non deterministica e quindi può accadere di dover memorizzare stringhe di lunghezza $2^{O(f(n))}$

Le scelte fatte per percorrere tutti i possibili cammini sono esponenziali in numero, anche se lo spazio necessario per calcolarli è polinomiale.
Dunque, questo tentativo richiede spazio esponenziale.

Metodo Non determinismo Verso determinismo Perché non funziona



1,2,11,12,21,22, …

Memoria che ci serve per percorrere tutti i cammini. Esponenziale

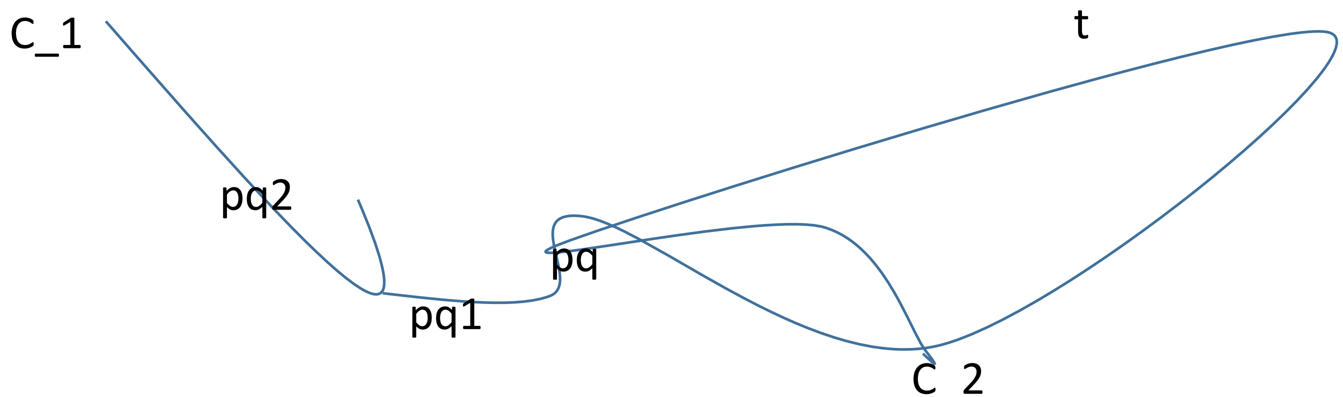
Memoria che si può riutilizzare

YELDABILITY PROBLEM

Verificare se una MNdT con input w può passare da una configurazione iniziale c_1 a quella finale c_2 in un numero di passi minore o uguale a un numero dato t (time). (dove t verrà scelto come il numero massimo di operazioni che MNdT necessita per accettare w).

Si consideri una MdT non deterministica arbitraria.

Preso un intero t positivo, e due configurazioni c_1 e c_2 di NMdT Diremo che c1 produce c2 in un numero minore o uguali di passi t se NMdT può produrre c_2 a partire da c_1 in t (o meno passi).



input c_1,c_2 e t, dove **t≥0.**
c_1,c_2 sono configurazioni che usano al più uno spazio f(n) (se lo spazio occupato è minore possiamo raggiungere spazio f(n) aggiungendo caratteri blank);

sia t una potenza del 2 ($t=2^p$ per qualche p maggiore o uguale a 0), possibile?.

CANYELD ha come input anche tutti gli stati e i collegamenti tra di loro ovvero la delta

CANYIELD(c_1, c_2, 2^t) =

1. Se p=0, allora "test (se c_1=c_2" oppure "c_1 produce c_2 in un solo step)" *(in base alle delta di MNdT).*
Accept se il test ha successo, altrimenti Reject.

2. Se t>0, allora per ciascuna (tutte) configurazione c_m di N (che usano spazio ≤f(n)):
3. Run CANYIELD(c_1, c_m, 2^{t-1}).
4. Run CANYIELD(c_m, c_2, 2^{t-1}).
5. Se lo step 3 e 4 entrambi accept, allora accept.
6. Se non hanno entrambi accettato allora, *reject*

Dimostrazione del teorema di Savitch

Passiamo a definire la MdT deterministica che simula MNdT. Prima abbiamo bisogno di fare una assunzione semplificativa:

1. Quando MNdT accetta, prima di fermarsi pulisce il nastro e ritorna all'inizio del nastro, dove entrerà in una (fissata) configurazione chiamata c_{accept}.

Notazione: Con w indichiamo l'input di N, n è la lunghezza di w e c_{start} è la configurazione iniziale di MNdT.

Denotiamo con d una costante tale che MNdT non usa più di d*f(n) configurazioni per computare w.

In pratica, $2^{d*f(n)}$ fornisce un upper bound per il tempo di esecuzione di MNdT su w (perché).

Con queste assunzioni, abbiamo che MNdT accetta w se e solo se si può andare da c_{start} a c_{accept} in al massimo $2^{d*f(n)}$ passi.

Analizziamo la seguente MdT deterministica:

M : = “”Su input **w**:
L’output è il risultato di
CANYIELD(c_{start}, c_{accept}, $2^{d*f(n)}$) .”

Quindi M simula NMdT

Analizziamo la complessità di spazio di M.

- Ogni volta che CANYIELD si autochiama (ricorsione) memorizza lo stato corrente (i valori c_1,c_2 e t) così da poter essere richiamati (usati) al ritorno dalla ricorsione.
- Ciascun livello della ricorsione quindi usa spazio O(f(n)) aggiuntivo.
- Il numero delle chiamate ricorsive è invece pari a d*f(n).

Dunque, lo spazio totale usato da N: **$d*f(n)*O(f(n))=O(f^2(n))$**

Ritorniamo per un momento sull'affermazione

M = "Su input **w**: L'output è il risultato di **CANYIELD**($\mathbf{c_{start}}$, $\mathbf{c_{accept}}$, $\mathbf{2^{d*f(n)}}$**)**"

M deve sapere il valore di f(n) quando invoca CANYIELD, questo lo dobbiamo calcolare. Quindi la complessità aumenta.

Un modo semplice per risolvere questo problema è quello di modificare M (chiamata M') in modo che provi per f(n) = 1,2,3, … Per ogni valore f(n) = i

,

M’ usa CANYIELD per accettare e determinare se la configurazione è raggiungibile.

Per garantire che M’ non continui ad aumentare i (e quindi a consumare spazio) nei casi in cui NMdT non accetta w, si effettua il seguente controllo prima di passare da i a i+1 (nuovo valore di f(n)):
Utilizzando CANYIELD, controlla che MNdT può raggiungere una configurazione che utilizza più di i celle del nastro (da c_{start}). Se questo non accade: non ha alcun senso cercare per i+1 quindi M’ va in reject.

Per memorizzare il valore i (f(n)): occorre spazio O(log f(n)). Inoltre, questo spazio può essere riciclato ad ogni iterazione.
Dunque, la complessità dello spazio di M’ rimane $O(f^2(n))$.

Questo è quello che conosciamo:

$$\mathbf{P \subseteq NP \subseteq PSPACE=NPSPACE \subseteq EXPTIME.}$$

Tuttavia non conosciamo se qualcuna delle inclusioni possa essere sostituita con un uguaglianza.

Sappiamo che

$$\mathbf{P \neq EXPTIME.}$$

Quindi sappiamo che l'ultima delle tre inclusioni deve essere necessariamente un inclusione (non una uguaglianza).

:

P ⊆ NP ⊆ PSPACE=NPSPACE ⊆ EXPTIME.

P≠EXPTIME.

Questo corso è dedicato alla città e agli abitanti che mi hanno ospitato in questo periodo.