Teoria della Complessità e Protocolli Zero Knowledge: un Approccio non Matematico

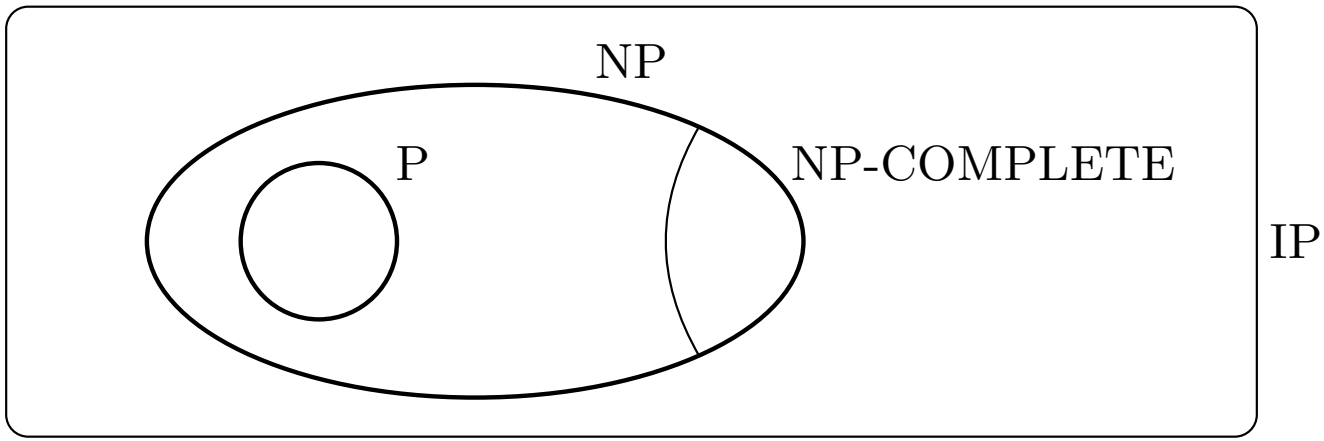
Giovanni Tognolini

Università di Trento

Aprile 2023

Cosa Facciamo Oggi?

1. Consideriamo diversi problemi.
   - Dato n, n è primo?
   - Dato un grafo G, G possiede proprietà particolari?
2. Studio della complessità.
   (problemi più facili, problemi più difficili)
3. Formalizzazione



4. Come la crittografia si relaziona con tutto ciò.

Concetti Base

problemi decisionali vs problemi computazionali

Problema 1 (Decisionale)

Dato un numero n, determinare se esso è un numero primo.
Input: n
Output: Sì/No

Istanza generica del problema 1: $n = 48$
Output: No

Problema 2 (Computazionale)

Dato un numero n, determinare la sua fattorizzazione in fattori primi.
Input: n
Output: Fattorizzazione di n

Istanza generica del problema 2: $n = 48$
Output: $2^4 \cdot 3$

Concetti Base

problemi decisionali vs problemi computazionali

Problema 1 (Decisionale)

Dato un numero n, determinare se esso è un numero primo.
Input: n
Output: Sì/No

Istanza generica del problema 1: $n = 48$
Output: No

Problema 2 (Computazionale)

Dato un numero n, determinare la sua fattorizzazione in fattori primi.
Input: n
Output: Fattorizzazione di n

Istanza generica del problema 2: $n = 48$
Output: $2^4 \cdot 3$

~~Problemi Computazionali~~
Problemi Decisionali

La Classe P

Alcuni Problemi Facili

la classe P

Esempio

Dato un numero intero n, decidere se n è dispari.

Esempio

Dato un array L di n numeri interi, dato un numero intero k, esiste un elemento di L maggiore o uguale a k?

$$L = \boxed{4}\boxed{12}\boxed{15}\boxed{3}\boxed{\cdots}\boxed{29}\boxed{37}\boxed{4}\boxed{52}\boxed{10} \qquad k = 48$$

Osservazione

Sembrano problemi “facili da risolvere”.

Alcuni Problemi Facili

la classe P

Esempio

Dato un numero intero n, decidere se n è dispari.

Esempio

Dato un array L di n numeri interi, dato un numero intero k, esiste un elemento di L maggiore o uguale a k?

$L =$ | 4 | 12 | 15 | 3 | $\cdots$ | 29 | 37 | 4 | 52 | 10 | $\qquad k = 48$

Osservazione

Sembrano problemi “facili da risolvere”.

Cosa Vuol Dire "Facili da Risolvere"?

proviamo a formalizzare questa intuizione...

Calcoleremo la complessità, ma cos'è e come si calcola?

Cos'è?

Numero di operazioni elementari

Come si calcola?

1. Dati (input) del problema.
2. Calcolo della grandezza dell'input.
3. Calcolo della complessità in funzione della grandezza dell'input.

Osservazione

Se la cresce come un polinomio, diremo che il problema è facile da risolvere.

Cosa Vuol Dire "Facili da Risolvere"?

proviamo a formalizzare questa intuizione...

Calcoleremo la complessità, ma cos'è e come si calcola?

Cos'è?

Numero di operazioni elementari

Come si calcola?

1. Dati (input) del problema.
2. Calcolo della grandezza dell'input.
3. Calcolo della complessità in funzione della grandezza dell'input.

Osservazione

Se la cresce come un polinomio, diremo che il problema è facile da risolvere.

Cosa Vuol Dire "Facili da Risolvere"?

proviamo a formalizzare questa intuizione...

Calcoleremo la complessità, ma cos'è e come si calcola?

Cos'è?

Numero di operazioni elementari

Come si calcola?

1. Dati (input) del problema.
2. Calcolo della grandezza dell'input.
3. Calcolo della complessità in funzione della grandezza dell'input.

Osservazione

Se la cresce come un polinomio, diremo che il problema è facile da risolvere.

Cosa Vuol Dire "Facili da Risolvere"?

proviamo a formalizzare questa intuizione...

Calcoleremo la complessità, ma cos'è e come si calcola?

Cos'è?

Numero di operazioni elementari

Come si calcola?

1. Dati (input) del problema.
2. Calcolo della grandezza dell'input.
3. Calcolo della complessità in funzione della grandezza dell'input.

Osservazione

Se la cresce come un polinomio, diremo che il problema è facile da risolvere.

Rivediamo l'Esempio di Prima

e calcoliamo la complessità del nostro algoritmo risolutivo

1. **Dati del problema:**

$$L = \boxed{4}\boxed{12}\boxed{15}\boxed{3}\boxed{\cdots}\boxed{29}\boxed{37}\boxed{4}\boxed{52}\boxed{10} \qquad k = 48$$

2. **Grandezza dei dati:**

- Per rappresentare L su un computer utilizziamo $8 \cdot n$ bit.

$$4 = 00000100 \qquad 12 = 00001100 \qquad \ldots \qquad 48 = 00110000$$

- Per rappresentare k usiamo 8 bit.
- In totale: $8n + 8$ bit
  (normalmente (per n grande) si eliminano le costanti $\longrightarrow n$)

3. **Complessità:**

- Tempo di lettura di k: 1 operazione elementare
- n confronti elementari.
- In totale: $1 + n \approx n$ (per n grande).

Rivediamo l'Esempio di Prima

e calcoliamo la complessità del nostro algoritmo risolutivo

1. **Dati del problema:**

$$L = \boxed{4}\boxed{12}\boxed{15}\boxed{3}\boxed{\cdots}\boxed{29}\boxed{37}\boxed{4}\boxed{52}\boxed{10} \qquad k = 48$$

2. **Grandezza dei dati:**

- Per rappresentare L su un computer utilizziamo $8 \cdot n$ bit.

$$4 = 00000100 \qquad 12 = 00001100 \qquad \ldots \qquad 48 = 00110000$$

- Per rappresentare k usiamo 8 bit.
- In totale: $8n + 8$ bit
  (normalmente (per n grande) si eliminano le costanti $\longrightarrow n$)

3. **Complessità:**

- Tempo di lettura di k: 1 operazione elementare
- n confronti elementari.
- In totale: $1 + n \approx n$ (per n grande).

Rivediamo l'Esempio di Prima

e calcoliamo la complessità del nostro algoritmo risolutivo

1. **Dati del problema:**

$$L = \boxed{4}\boxed{12}\boxed{15}\boxed{3}\boxed{\cdots}\boxed{29}\boxed{37}\boxed{4}\boxed{52}\boxed{10} \qquad k = 48$$

2. **Grandezza dei dati:**

- Per rappresentare L su un computer utilizziamo $8 \cdot n$ bit.

$$4 = 00000100 \qquad 12 = 00001100 \qquad \ldots \qquad 48 = 00110000$$

- Per rappresentare k usiamo 8 bit.
- In totale: $8n + 8$ bit
  (normalmente (per n grande) si eliminano le costanti $\longrightarrow n$)

3. **Complessità:**

- Tempo di lettura di k: 1 operazione elementare
- n confronti elementari.
- In totale: $1 + n \approx n$ (per n grande).

Rivediamo l'Esempio di Prima

e calcoliamo la complessità del nostro algoritmo risolutivo

1. **Dati del problema:**

$$L = \boxed{4}\boxed{12}\boxed{15}\boxed{3}\boxed{\cdots}\boxed{29}\boxed{37}\boxed{4}\boxed{52}\boxed{10} \qquad k = 48$$

2. **Grandezza dei dati:**

- Per rappresentare L su un computer utilizziamo $8 \cdot n$ bit.

$$4 = 00000100 \qquad 12 = 00001100 \qquad \ldots \qquad 48 = 00110000$$

- Per rappresentare k usiamo 8 bit.
- In totale: $8n + 8$ bit
  (normalmente (per n grande) si eliminano le costanti $\longrightarrow n$)

3. **Complessità:**

- Tempo di lettura di k: 1 operazione elementare
- n confronti elementari.
- In totale: $1 + n \approx n$ (per n grande).

Stesso Problema, Diverso Algoritmo

$L =$ | 4 | 12 | 15 | 3 | $\cdots$ | 29 | 37 | 4 | 52 | 10 | $\qquad k = 48$

1. Ordiniamo gli elementi di L in ordine crescente

3	4	4	10	12	$\cdots$	15	29	37	52

(può essere fatto con diversi algoritmi (Bubblesort, Quicksort,...): $n\log_2(n)$ operazioni)

2. Prendiamo l'ultimo elemento della lista e confrontiamolo con k. 1 operazione.

In totale: $n\log_2(n) + 1$ operazioni

Osservazione

$$n\log_2(n) + 1 \leq n^2 + 1$$

Un Altro Esempio

Dato un numero intero n, n è dispari?

1. *Input:* n.
2. *Grandezza input:* $\log_2(n)$.
3. *Complessità del nostro algoritmo:* 1.

Alcuni Problemi Comuni

non confondere input e grandezza dell'input

Consideriamo il problema di determinare la primalità di un dato numero n.
Proviamo a vedere se è divisibile per $2, 3, 4, 5, 6, 7, \ldots, \sqrt{n}$.

Qual è la complessità di questo algoritmo? $\sqrt{n}$

Questo significa che abbiamo trovato un algoritmo polinomiale per fattorizzare n?

Quanto è grande l'input? $\log(n)$.
Pertanto la complessità in funzione della grandezza dell'input è

$$\sqrt{n} = n^{\frac{1}{2}} = 2^{(\log(n))^{\frac{1}{2}}} = 2^{(\frac{1}{2}\log(n))}$$

(ESPONENZIALE)

Alcuni Problemi Comuni

non confondere input e grandezza dell'input

Consideriamo il problema di determinare la primalità di un dato numero n.
Proviamo a vedere se è divisibile per $2, 3, 4, 5, 6, 7, \ldots, \sqrt{n}$.

Qual è la complessità di questo algoritmo? $\sqrt{n}$

Questo significa che abbiamo trovato un algoritmo polinomiale per fattorizzare n?

Quanto è grande l'input? $\log(n)$.
Pertanto la complessità in funzione della grandezza dell'input è

$$\sqrt{n} = n^{\frac{1}{2}} = 2^{(\log(n))^{\frac{1}{2}}} = 2^{(\frac{1}{2}\log(n))}$$

(ESPONENZIALE)

Alcuni Problemi Comuni

non confondere input e grandezza dell'input

Consideriamo il problema di determinare la primalità di un dato numero n.
Proviamo a vedere se è divisibile per $2, 3, 4, 5, 6, 7, \ldots, \sqrt{n}$.

Qual è la complessità di questo algoritmo? $\sqrt{n}$

Questo significa che abbiamo trovato un algoritmo polinomiale per fattorizzare n?

Quanto è grande l'input? $\log(n)$.
Pertanto la complessità in funzione della grandezza dell'input è

$$\sqrt{n} = n^{\frac{1}{2}} = 2^{(\log(n))^{\frac{1}{2}}} = 2^{(\frac{1}{2}\log(n))}$$

(ESPONENZIALE)

Alcuni Problemi Comuni

non confondere input e grandezza dell'input

Consideriamo il problema di determinare la primalità di un dato numero n.
Proviamo a vedere se è divisibile per $2, 3, 4, 5, 6, 7, \ldots, \sqrt{n}$.

Qual è la complessità di questo algoritmo? $\sqrt{n}$

Questo significa che abbiamo trovato un algoritmo polinomiale per fattorizzare n?

Quanto è grande l'input? $\log(n)$.
Pertanto la complessità in funzione della grandezza dell'input è

$$\sqrt{n} = n^{\frac{1}{2}} = 2^{(\log(n))^{\frac{1}{2}}} = 2^{(\frac{1}{2}\log(n))}$$

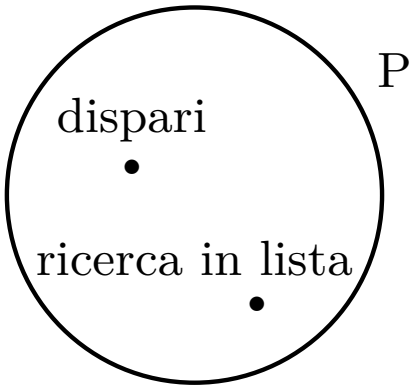
(ESPONENZIALE)

Piccolo Recap

per riassumere

Dato un problema con input grande n diremo che esso s vive in P se esiste almeno un algoritmo che:

1. Risolve il problema.
2. Lo risolve in tempo limitato da un polinomio in n (es: $n^0, n, n^3 + 2n + 1, \ldots$).



La Classe NP

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

SAT

SAT (Satisfaiability) è il problema di determinare se una formula booleana è soddisfacibile o insoddisfacibile.

Esempio (SAT)

- x_1
  $(x_1 = \text{True})$
- $\neg x_1$
  $(x_1 = \text{False})$
- $x_1 \wedge x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
- $x_1 \vee x_2$
  $(x_1 = \text{True})$ e $(x_2 = \text{True})$
  $(x_1 = \text{True})$ e $(x_2 = \text{False})$
  $(x_1 = \text{False})$ e $(x_2 = \text{True})$
- $(x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \neg x_6)$

Sapere e Saper Fare

Un problema per voi

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

Sapere e Saper Fare

Un problema per voi

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

Una soluzione è data (ad esempio) dalla terna $(x_1, \neg x_2, \neg x_3)$.

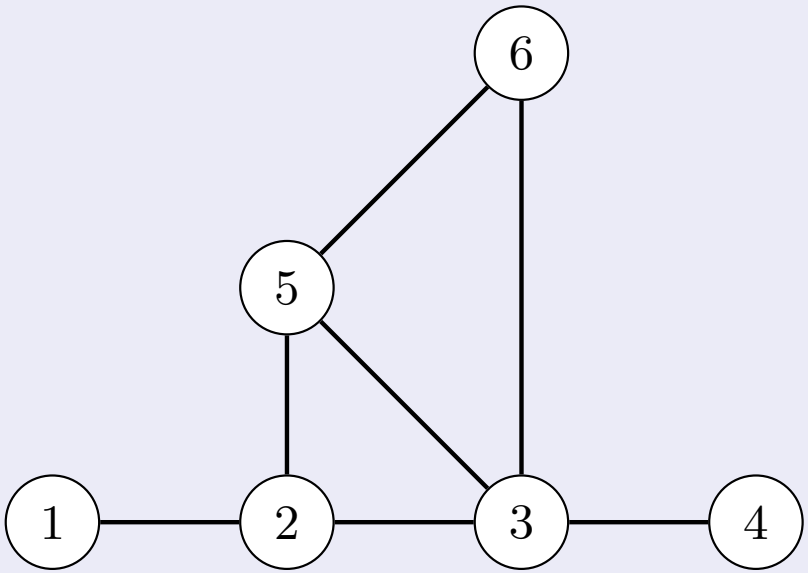
Vediamo un Altro Esempio

VertexCover

Definizione

Un *vertexCover* di un grafo è un insieme S di nodi tale per cui ogni arco del grafo entra o esce da S.
Input: Grafo G, intero k.
Esiste un vertex cover di G di dimensione k?

Esempio



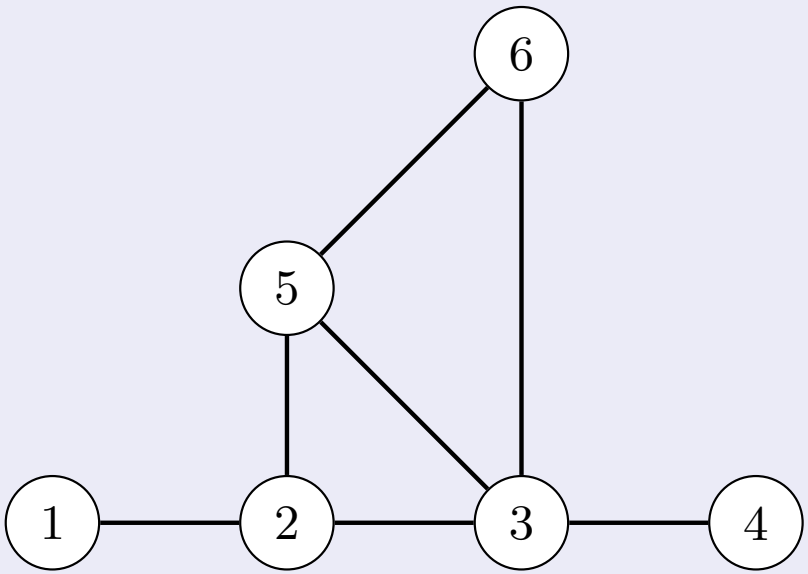
Vediamo un Altro Esempio

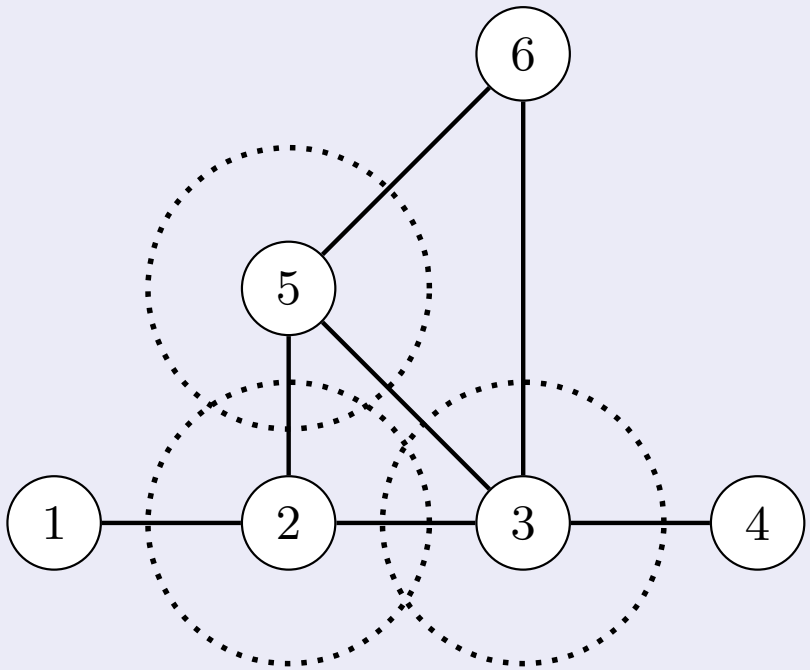
VertexCover

Definizione

Un *vertexCover* di un grafo è un insieme S di nodi tale per cui ogni arco del grafo entra o esce da S.
Input: Grafo G, intero k.
Esiste un vertex cover di G di dimensione k?

Esempio





SetCover

Definizione

Dato un insieme U di elementi e un insieme $\{S_1, \ldots, S_n\}$ di sottoinsiemi di U, esistono k sottoinsiemi che ricoprono U?

Esempio

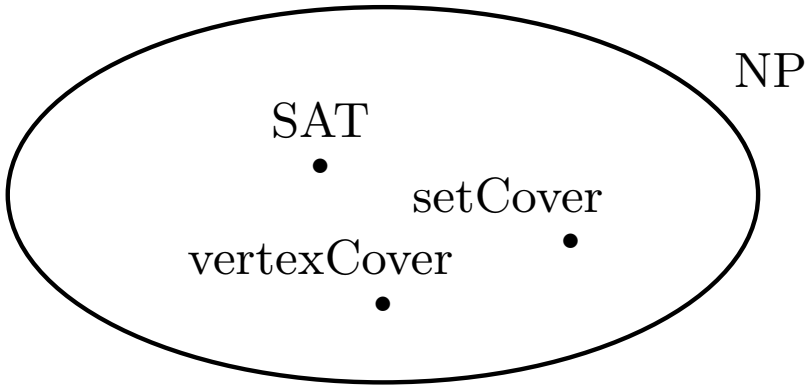
$$U = \Big\{1, 2, 3, 4, 5, 6, 7\Big\} \qquad k = 3$$

$$\begin{aligned} S_1 &:= \{1\} \\ S_2 &:= \{1, 2, 3\} \\ S_3 &:= \{3, 4, 5, 6\} \\ S_4 &:= \{5\} \\ S_5 &:= \{2, 4, 7\} \\ S_6 &:= \{6, 7\} \end{aligned}$$

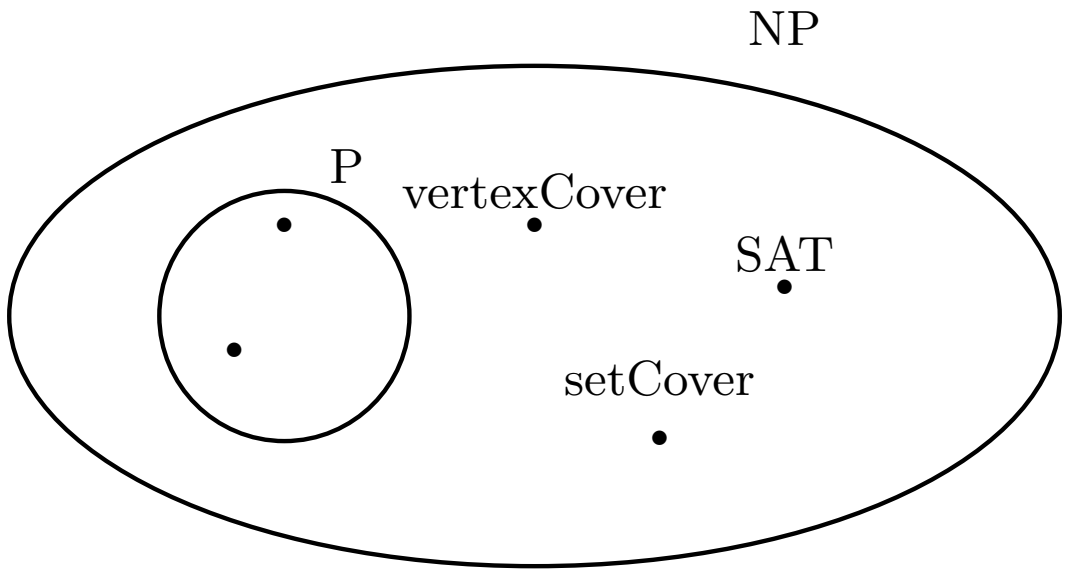
SAT, VertexCover e SetCover, per istanze "grandi", sono difficili da risolvere, ma facili da verificare.

Mettiamo Tutto Insieme

Diremo che un problema vive in NP se le sue soluzioni possono essere verificate in tempo polinomiale.



Relazione fra P ed NP



Una Domanda Lecita

Siamo sicuri che i problemi che abbiamo visto (SAT/vertexCover/setCover) non siamo in grado di risolverli in modo più efficiente?

detto diversamente

È possibile che anche questi problemi vivano in P, e siamo noi che non riusciamo a pensare ad un algoritmo migliore?

Il problema "Dato un numero n, stabilire se n è primo" vive in P.

Una Domanda Lecita

Siamo sicuri che i problemi che abbiamo visto (SAT/vertexCover/setCover) non siamo in grado di risolverli in modo più efficiente?

detto diversamente

È possibile che anche questi problemi vivano in P, e siamo noi che non riusciamo a pensare ad un algoritmo migliore?

Il problema "Dato un numero n, stabilire se n è primo" vive in P.

Annals of Mathematics, **160** (2004), 781–793

PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

Abstract

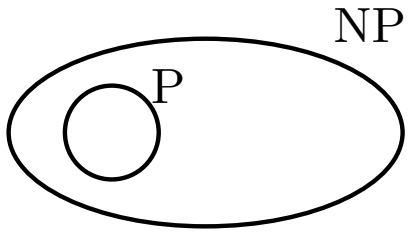
We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

1. Introduction

Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So it is of great interest to study different properties of prime numbers. Of special interest are those properties that allow one to determine efficiently if a number is prime. Such efficient tests are also useful in practice: a number of cryptographic protocols need large prime numbers.

Let PRIMES denote the set of all prime numbers. The definition of prime

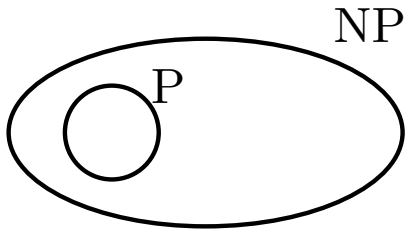
Quindi possiamo fattorizzare un numero in tempo polinomiale?

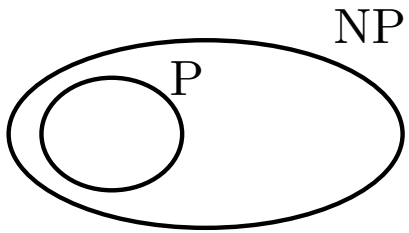


P = NP

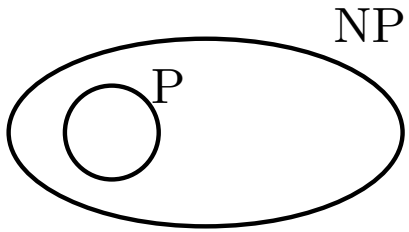
P ≠ NP

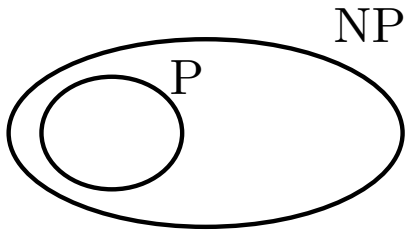
Domanda aperta:
P vs NP

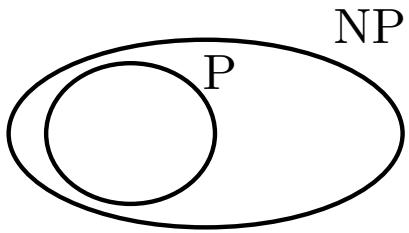




Domanda aperta:
P vs NP



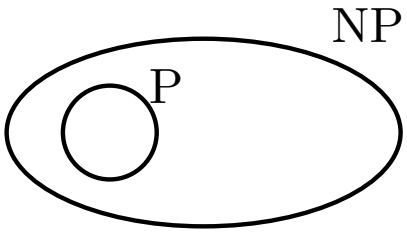


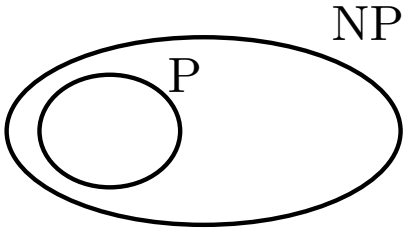


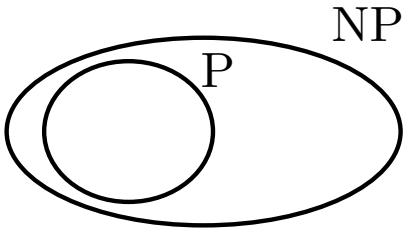
P = NP

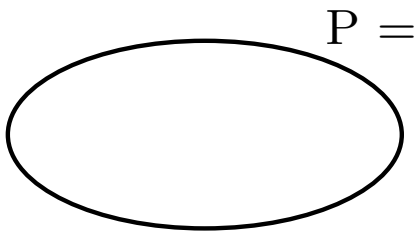
P ≠ NP

Domanda aperta:
P vs NP

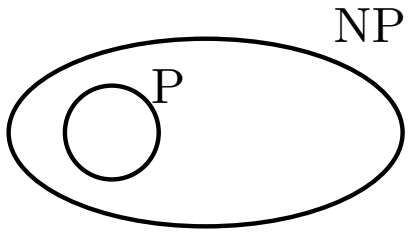


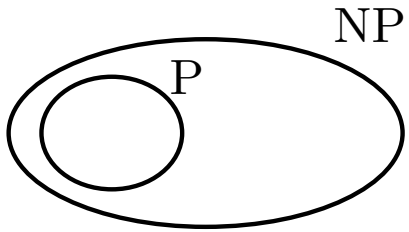


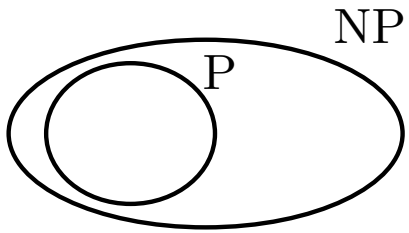


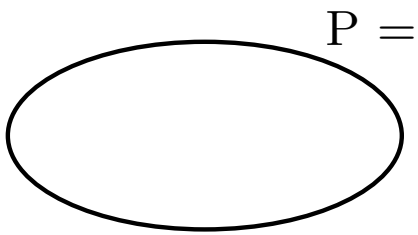


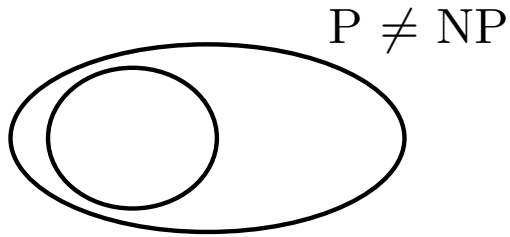
Domanda aperta:
P vs NP



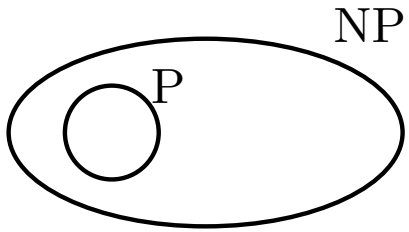


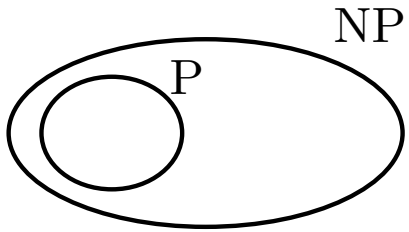


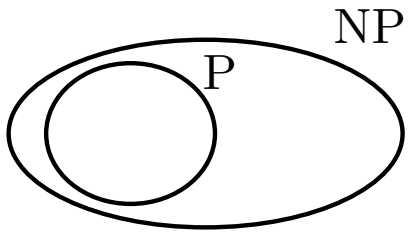


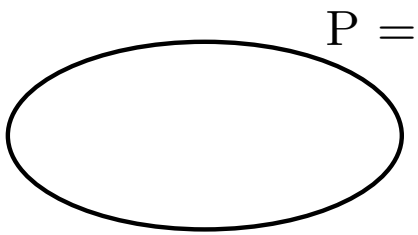


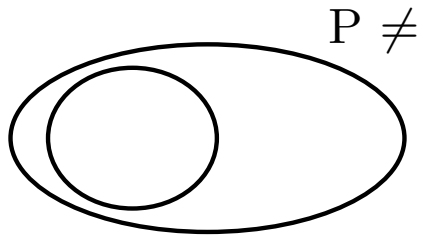
Domanda aperta:
P vs NP











Domanda aperta:
P vs NP

Un Problema non da Poco

i problemi del millennio

Studiamo più a fondo il rapporto fra P e NP

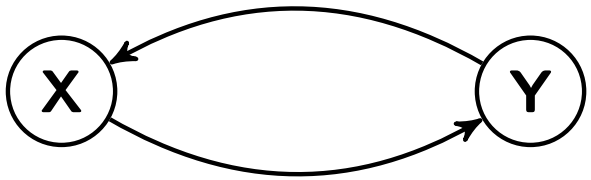
i problemi NP-Completi

Per studiare il rapporto fra P ed NP (negli anni) ci si è focalizzato a cercare i problemi "più difficili" della classe NP.

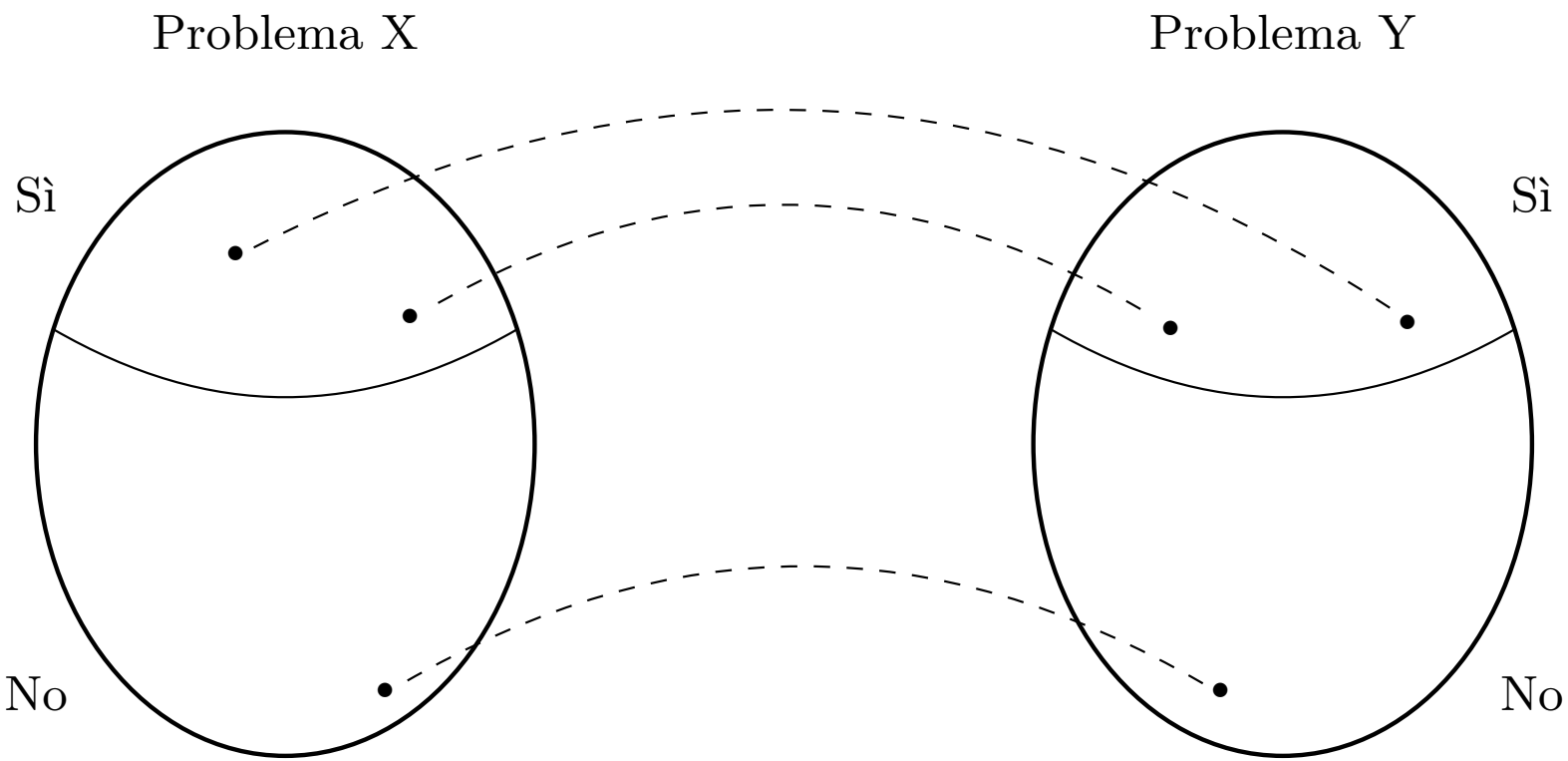
Ma cosa vuol dire che
"*un problema X è più facile da risolvere di un altro problema Y*"?

Una Prima Bozza di Definizione...

1. Prendo un'instanza di X.
2. In *tempo polinomiale* la trasformo in un'istanza di Y.
3. Risolvo Y.
4. Ri-trasformo la soluzione di Y in una soluzione di X.



La trasformazione non deve mischiare le istanze SÌ e le istanze NO dei due problemi.


Problema X
Problema Y
Sì
Sì
No
No

Esempio

VertexCover è più facile di SetCover

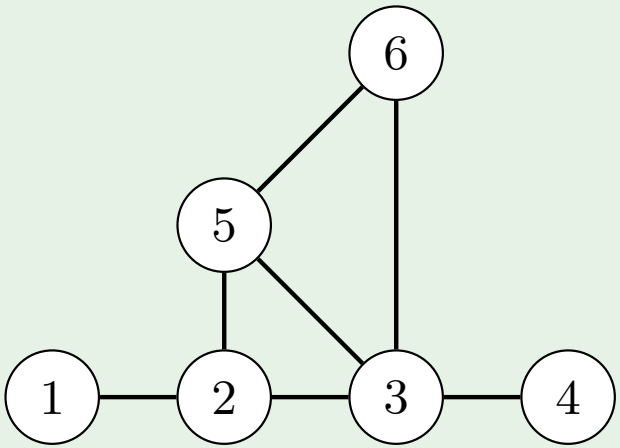


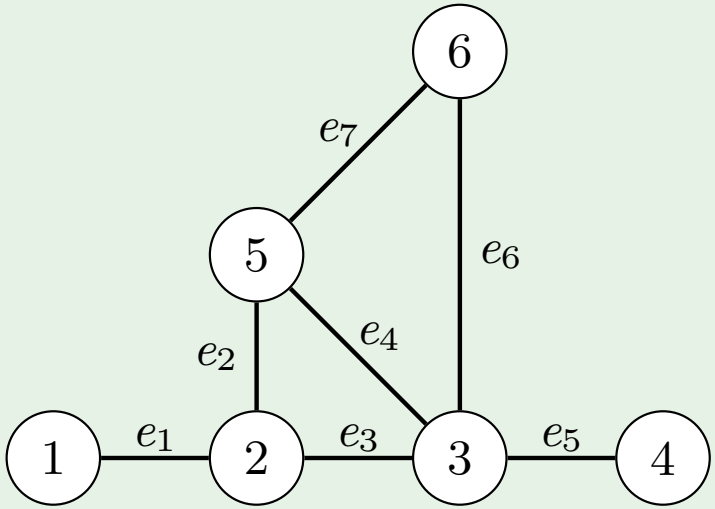
Figura: VertexCover

$$U = \Big\{1, 2, 3, 4, 5, 6, 7\Big\}, k = 3$$

$$S_1 := \{1\}$$
$$S_2 := \{1, 2, 3\}$$
$$S_3 := \{3, 4, 5, 6\}$$
$$S_4 := \{5\}$$
$$S_5 := \{2, 4, 7\}$$
$$S_6 := \{6, 7\}$$

Figura: SetCover

1. Consideriamo un'istanza (V, E), $k = 3$ di VertexCover.
2. Creiamo l'istanza associata di SetCover nel modo seguente: per ogni vertice $v \in V$, creiamo l'insieme S_v che contiene gli archi adiacenti a v.



$$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}.$$

$$S_1 := \{e_1\}$$
$$S_2 := \{e_1, e_2, e_3\}$$
$$S_3 := \{e_3, e_4, e_5, e_6\}$$
$$S_4 := \{e_5\}$$
$$S_5 := \{e_2, e_4, e_7\}$$
$$S_6 := \{e_6, e_7\}$$

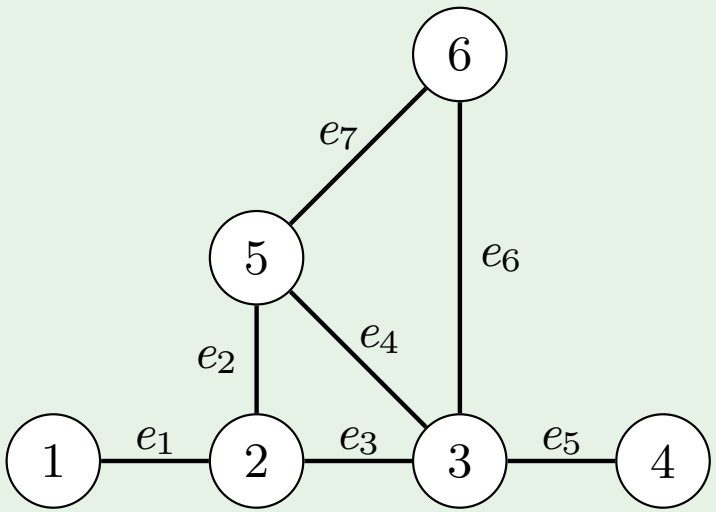
3. Risolviamo l'istanza di SetCover e trasformiamo la soluzione.

$$\{2, 3, 5\} \longleftarrow \{S_2, S_3, S_5\}$$
$$\{2, 3, 6\} \longleftarrow \{S_2, S_3, S_6\}$$

Osservazione

Le istanze "Sì" e "No" non si mischiano.

1. Consideriamo un'istanza (V, E), $k = 3$ di VertexCover.
2. Creiamo l'istanza associata di SetCover nel modo seguente: per ogni vertice $v \in V$, creiamo l'insieme S_v che contiene gli archi adiacenti a v.



$$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}.$$

$$S_1 := \{e_1\}$$
$$S_2 := \{e_1, e_2, e_3\}$$
$$S_3 := \{e_3, e_4, e_5, e_6\}$$
$$S_4 := \{e_5\}$$
$$S_5 := \{e_2, e_4, e_7\}$$
$$S_6 := \{e_6, e_7\}$$

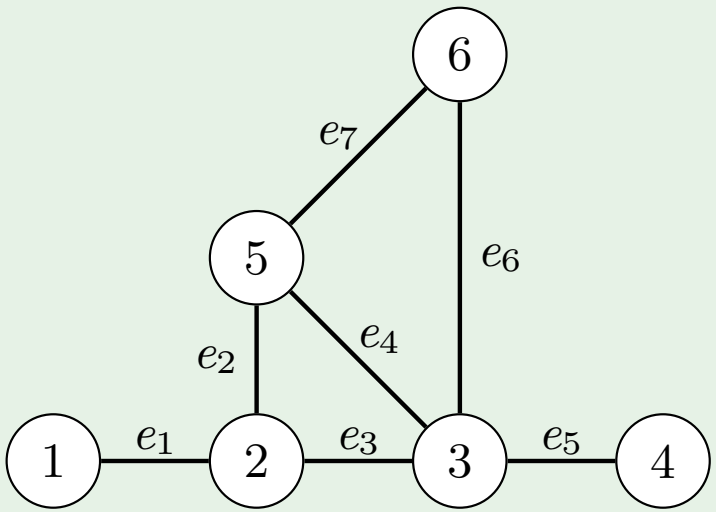
3. Risolviamo l'istanza di SetCover e trasformiamo la soluzione.

$$\{2, 3, 5\} \longleftarrow \{S_2, S_3, S_5\}$$
$$\{2, 3, 6\} \longleftarrow \{S_2, S_3, S_6\}$$

Osservazione

Le istanze "Sì" e "No" non si mischiano.

1. Consideriamo un'istanza (V, E), $k = 3$ di VertexCover.
2. Creiamo l'istanza associata di SetCover nel modo seguente: per ogni vertice $v \in V$, creiamo l'insieme S_v che contiene gli archi adiacenti a v.



$$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}.$$

$$S_1 := \{e_1\}$$
$$S_2 := \{e_1, e_2, e_3\}$$
$$S_3 := \{e_3, e_4, e_5, e_6\}$$
$$S_4 := \{e_5\}$$
$$S_5 := \{e_2, e_4, e_7\}$$
$$S_6 := \{e_6, e_7\}$$

3. Risolviamo l'istanza di SetCover e trasformiamo la soluzione.

$$\{2, 3, 5\} \longleftarrow \{S_2, S_3, S_5\}$$
$$\{2, 3, 6\} \longleftarrow \{S_2, S_3, S_6\}$$

Osservazione

Le istanze "Sì" e "No" non si mischiano.

Due Osservazioni Fondamentali

- La riduzione deve avvenire in tempo polinomiale
- La riduzione deve essere tale che non mischi le istanze SI e le istanze NO dei due problemi. Più formalmente:

In generale

Diremo che un problema X è più facile di un altro problema Y se possiamo ridurre *in tempo polinomiale* ogni istanza del problema X in un'istanza del problema Y, e la soluzione dell'istanza ridotta è *coerente* con l'istanza di partenza.

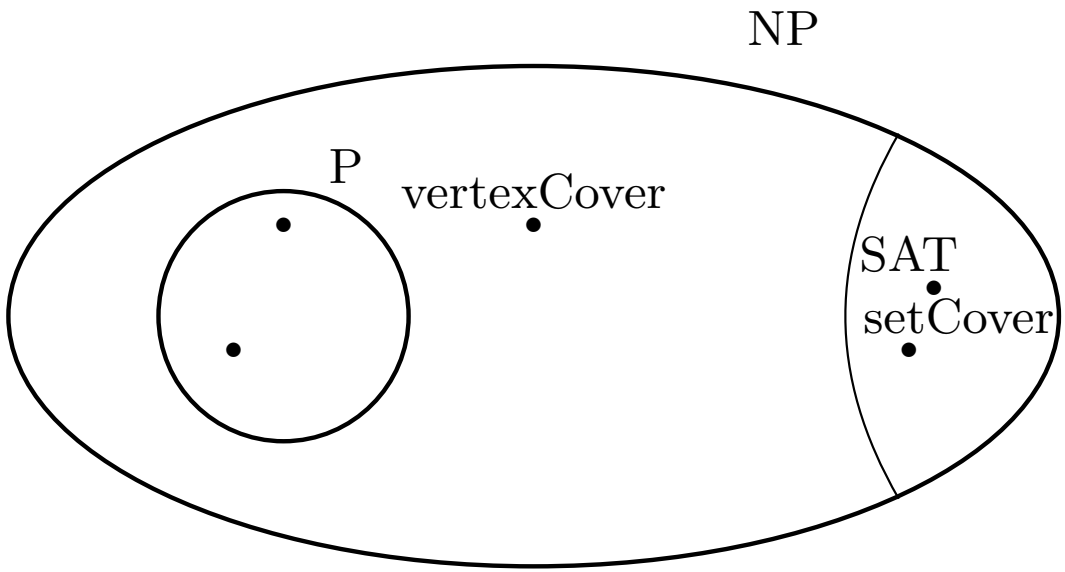
SetCover è più Difficile di VertexCover

e quindi?

Teorema

Ogni problema in NP è riducibile a SetCover

I problemi (di NP) con questa "proprietà" sono detti NP-Completi.



Se anche uno solo di questi problemi stesse in P ....

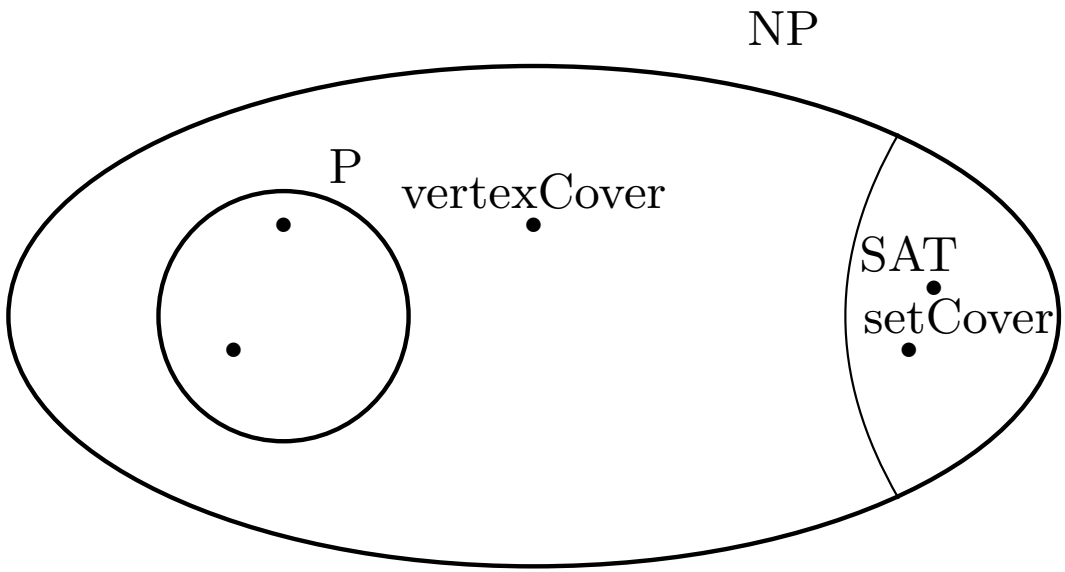
SetCover è più Difficile di VertexCover

e quindi?

Teorema

Ogni problema in NP è riducibile a SetCover

I problemi (di NP) con questa "proprietà" sono detti NP-Completi.

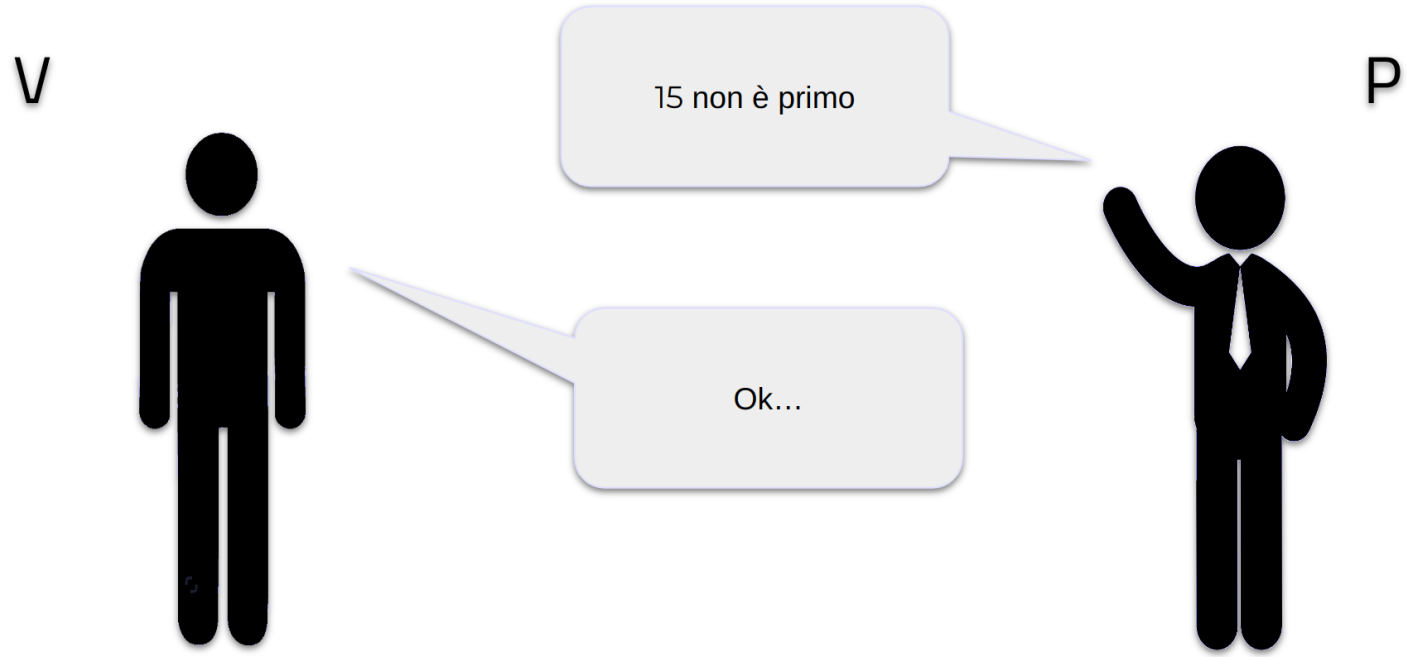


Se anche uno solo di questi problemi stesse in P .... P = NP.

La Classe IP

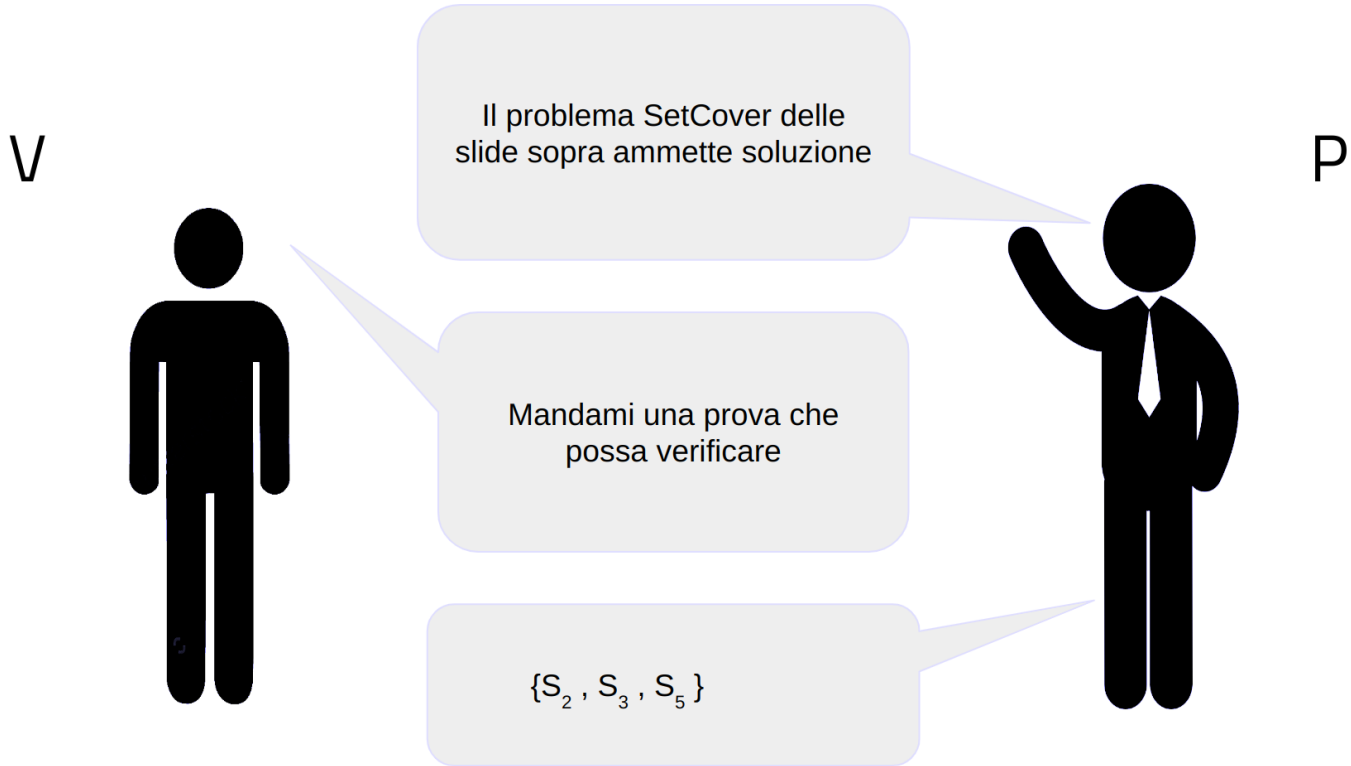
IP = Interactive Protocols

"tutti quei problemi che sono risolvibili facendo interagire due enti"

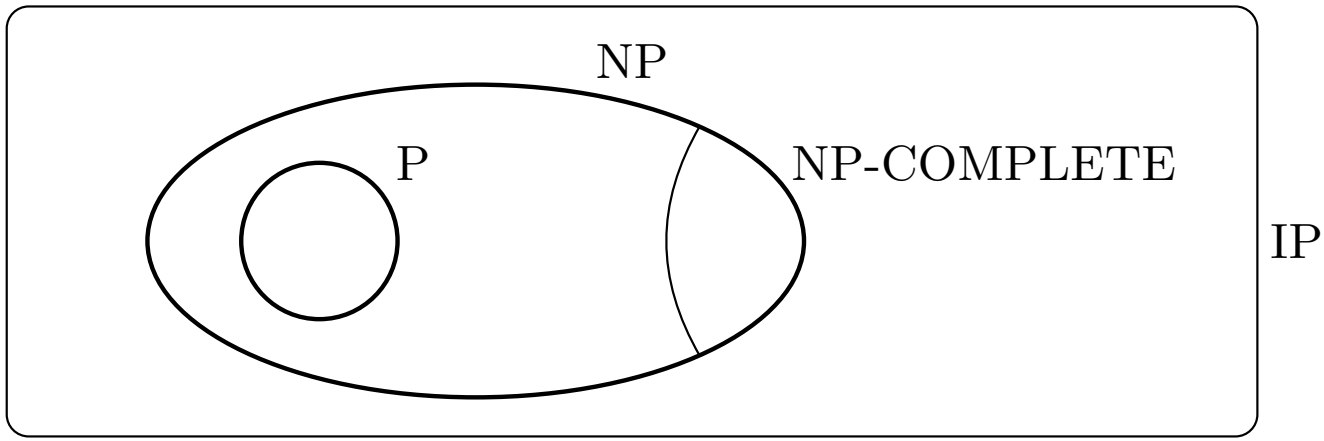


IP = Interactive Protocols

"Tutti quei problemi che sono risolvibili facendo interagire due enti"

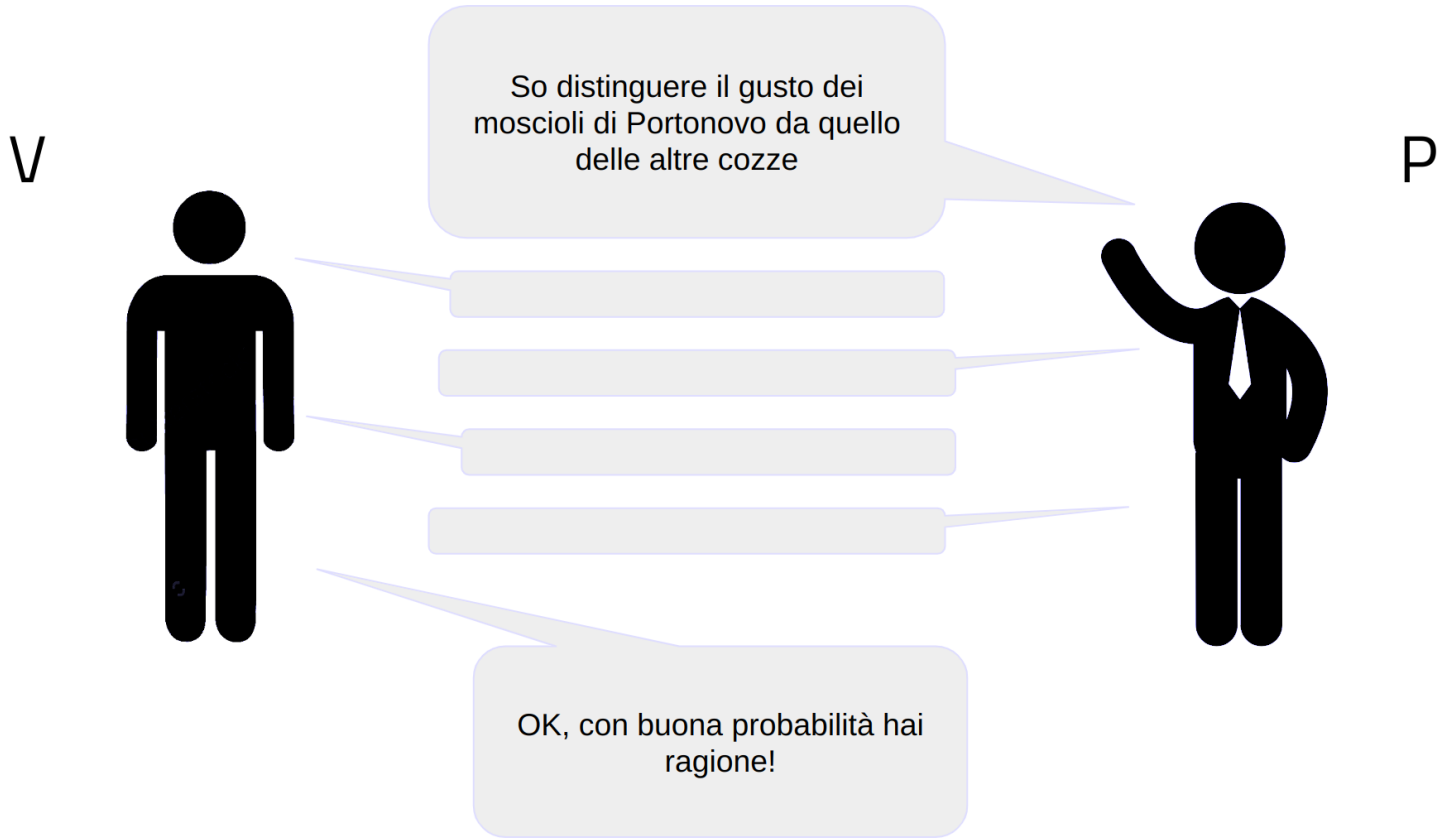


Cosa abbiamo appena visto?

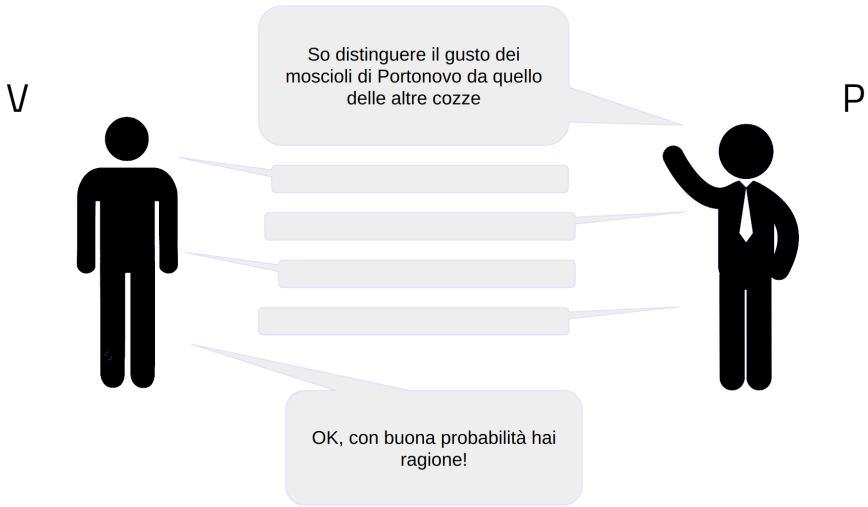


IP = Interactive Protocols

"Tutti quei problemi che sono risolvibili facendo interagire due enti"



Che Proprietà Richiediamo per Questa Classe?



Completezza: Se la risposta al problema è SI, il verifier accetta con probabilità alta.

Soundness: Se la risposta al problema è NO, il verifier accetta con probabilità molto bassa.

Seconda Parte:

Protocolli Zero-Knowledge

Cambiamo il Punto di Vista

Fino ad ora è il verifier a controllare l'onestà del prover...

... e se fosse il prover a non fidarsi del verifier?

Proof vs Proof of Knowledge

una differenza importante

Cosa possiamo dimostrare con uno scambio di messaggi interattivo?

Affermazione "su un fatto"

La porta ha una combinazione che la apre.
Il grafo G ammette un vertex cover di dimensione 3.
Il numero 149612341 non è primo.
...

Affermazione "sulla nostra conoscenza personale"

Conosco la combinazione che apre la porta.
Conosco un vertexCover di G di dimensione 3.
Conosco una fattorizzazione di 149612341.
...

Sono problemi molto diversi!

~~Zero Knowledge Proof~~
Zero Knowledge Proof of Knowledge

Per brevità si parla sempre di Zero Knowledge (proof of knowledge).

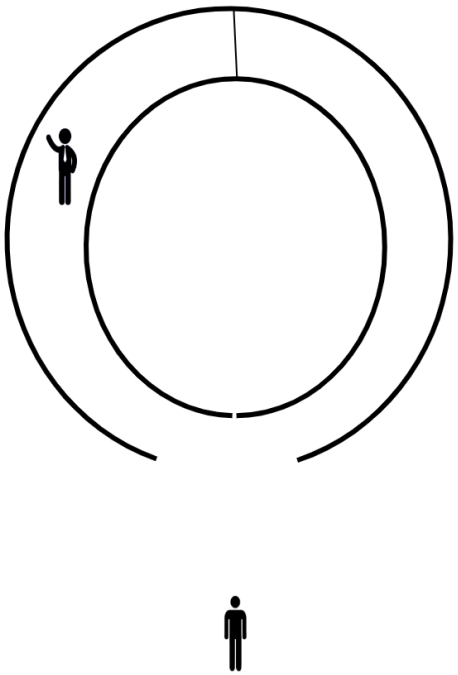
~~Zero Knowledge Proof~~
Zero Knowledge Proof of Knowledge

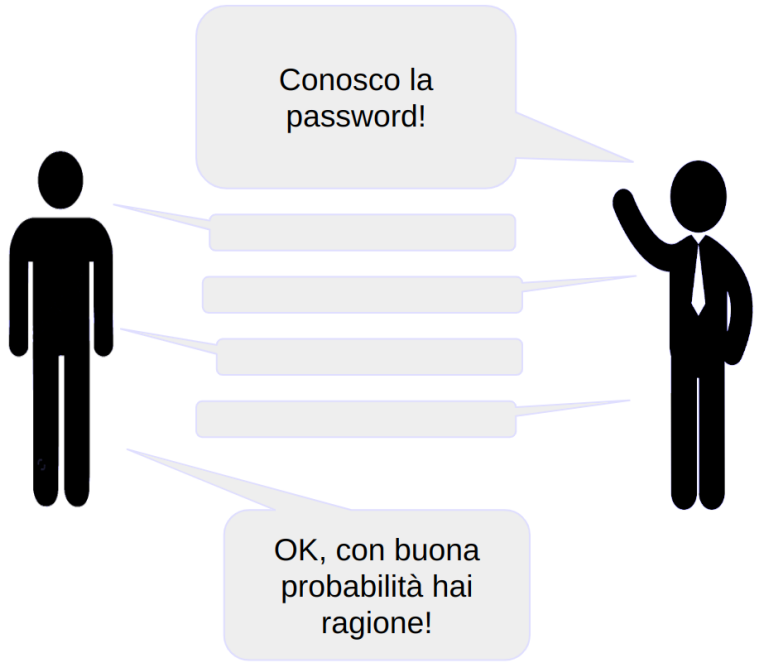
Per brevità si parla sempre di Zero Knowledge (proof of knowledge).

Obiettivi dei Protocolli Zero-Knowledge

- Convincere il verifier della conoscenza di un segreto (proof of knowledge).
- Non rilasciare nessun'altra informazione.

Un Esempio Classico: la Grotta



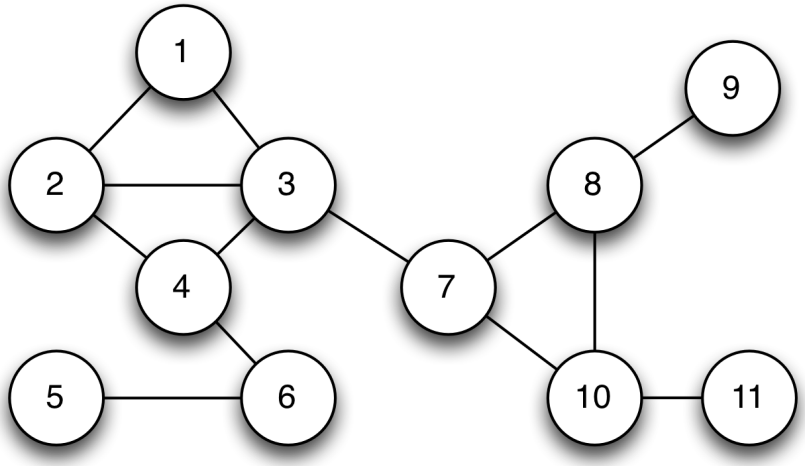


PRO: Rende bene l'idea di zero-knowledge.

CONTRO: Non ci permette di andare in profondità.

Un Esempio più Grande

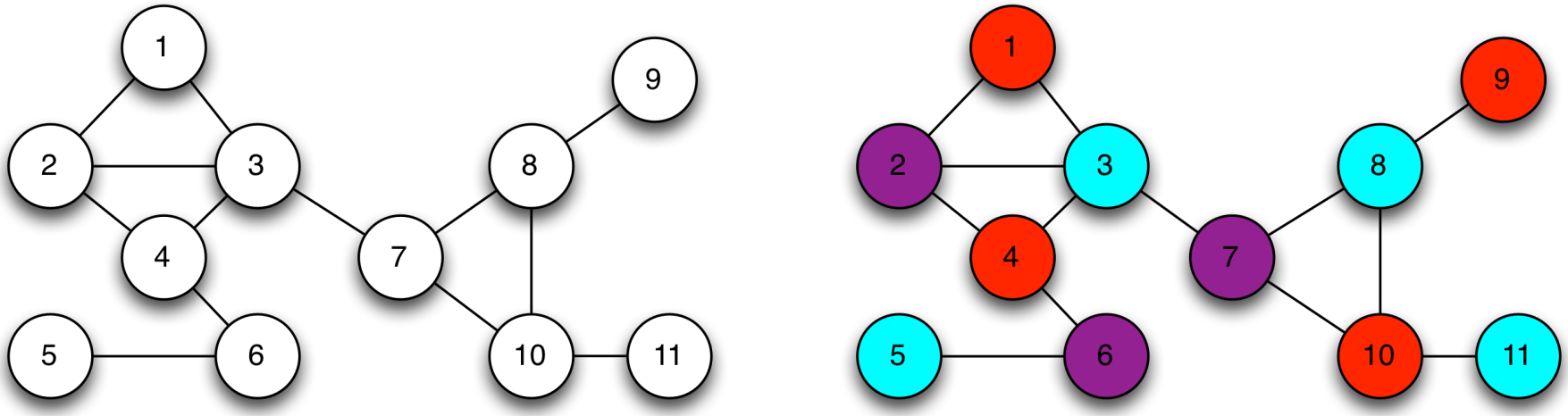
Supponiamo di essere una grande multinazionale delle telecomunicazioni e di voler progettare una nuova rete di comunicazione.



- I nodi rappresentano le torri radio.
- Gli archi rappresentano le zone di interferenza.

Un Esempio più Grande

Per fortuna il design di progettazione ci permette di configurare 3 frequenze di comunicazione differenti, in modo da evitare questi problemi.



Abbiamo ricondotto la progettazione della nostra rete alla 3-colorabilità di un grafo

Qual è il Punto?

Il problema di 3-colorare un grafo è NP-completo.

Istanze piccole: (computazionalmente) facili
Istanza grandi: (computazionalmente) difficili

L'esempio di prima lo abbiamo risolto perché era piccolo,
ma come fareste se aveste a che fare con una rete più grande?

La cosa più naturale è affidare il problema a qualcuno che abbia più potenza computazionale della vostra.

Io ho accesso al cluster dell'università (!)

Questo Porta a un Problema...

supponiamo che io mi prenda a carico i vostri conti...

Da un lato voi vorreste essere sicuri che io ho effettivamente trovato una soluzione al problema, prima di pagarmi.

Dall'altro lato, io vorrei essere pagato prima di darvi la soluzione.

...impasse...

Due Soluzioni

Passare per avvocati e vie legali

Ci inventiamo una soluzione “folle” a questo problema

Due Soluzioni

Passare per avvocati e vie legali

Ci inventiamo una soluzione “folle” a questo problema

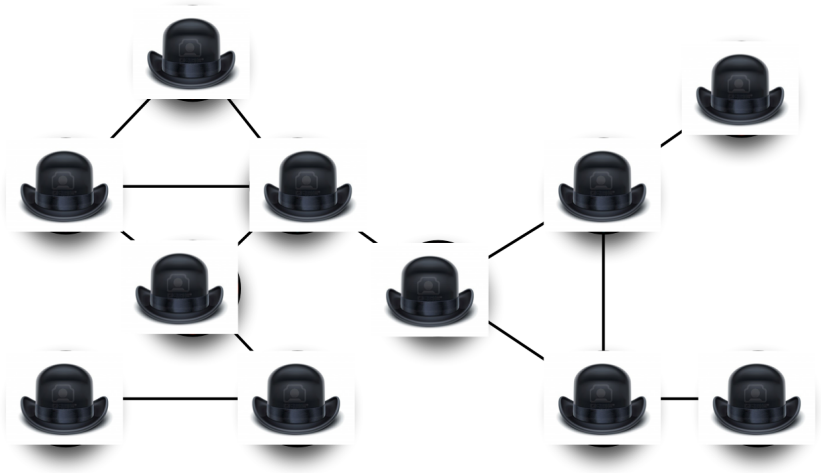
Una Soluzione un po' Folle

Per convincervi che ho risolto il problema
(senza rivelarvi nulla sulla soluzione)
ho bisogno di

1. Una stanza grande.
2. Tre pastelli colorati.
3. Un po' di fogli di carta.
4. Un po' di cappelli

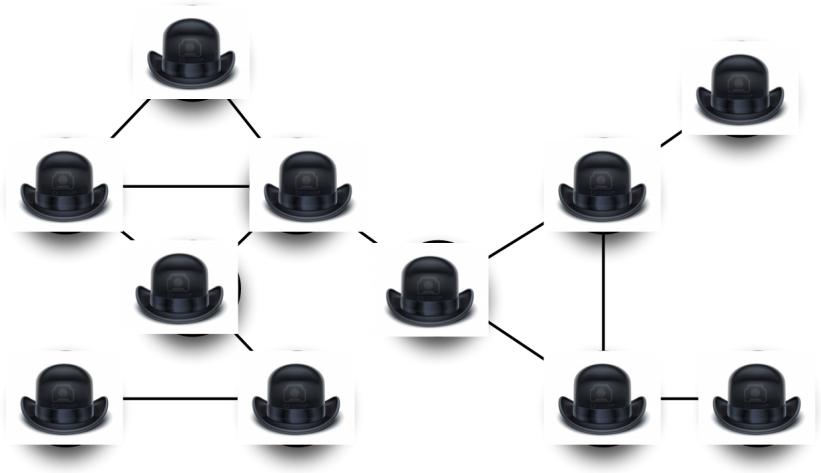
Come Funziona?

1. Voi entrate nella stanza, tappezzate il pavimento con la carta e ci disegnate sopra il grafo che rappresenta la vostra infrastruttura di rete.
2. Io entro nella stanza con 3 pastelli colorati e coloro il grafo seguendo la soluzione che mi ha dato il cluster. Copro tutti i nodi del grafo con dei cappelli.
3. Voi ri-entrate nella stanza, e vedete questo:



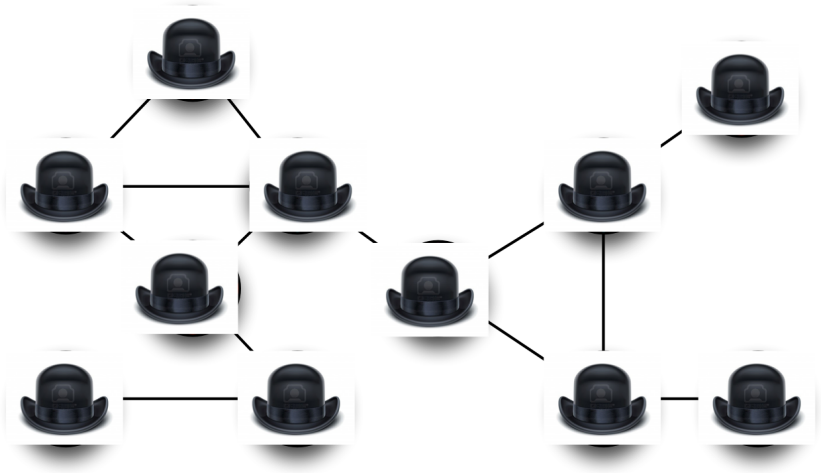
Come Funziona?

1. Voi entrate nella stanza, tappezzate il pavimento con la carta e ci disegnate sopra il grafo che rappresenta la vostra infrastruttura di rete.
2. Io entro nella stanza con 3 pastelli colorati e coloro il grafo seguendo la soluzione che mi ha dato il cluster. Copro tutti i nodi del grafo con dei cappelli.
3. Voi ri-entrate nella stanza, e vedete questo:



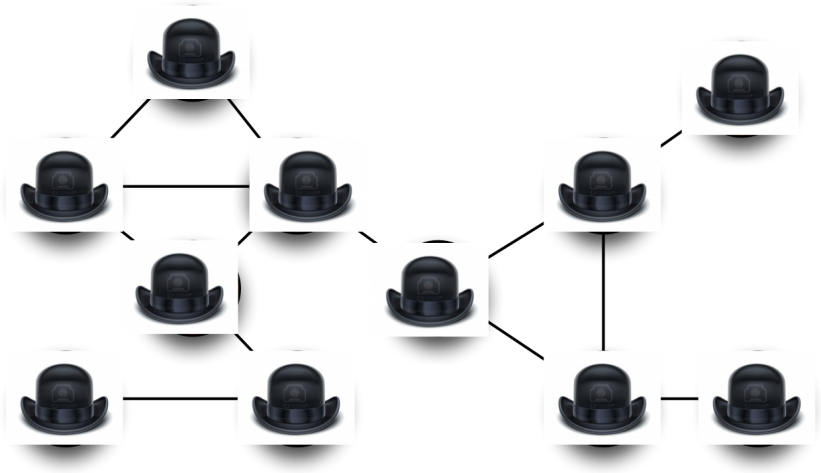
Come Funziona?

1. Voi entrate nella stanza, tappezzate il pavimento con la carta e ci disegnate sopra il grafo che rappresenta la vostra infrastruttura di rete.
2. Io entro nella stanza con 3 pastelli colorati e coloro il grafo seguendo la soluzione che mi ha dato il cluster. Copro tutti i nodi del grafo con dei cappelli.
3. Voi ri-entrate nella stanza, e vedete questo:

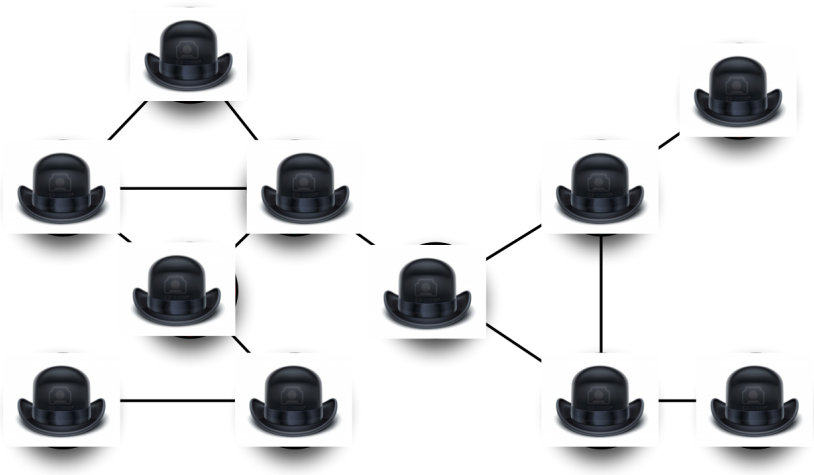


Come Funziona?

1. Voi entrate nella stanza, tappezzate il pavimento con la carta e ci disegnate sopra il grafo che rappresenta la vostra infrastruttura di rete.
2. Io entro nella stanza con 3 pastelli colorati e coloro il grafo seguendo la soluzione che mi ha dato il cluster. Copro tutti i nodi del grafo con dei cappelli.
3. Voi ri-entrate nella stanza, e vedete questo:



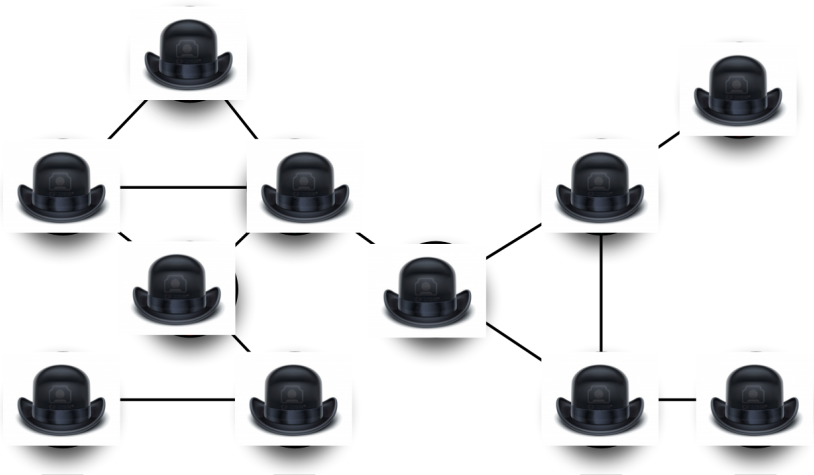
Alcune Osservazioni sul Grafo



- I cappelli nascondono perfettamente i colori (io sono sicuro che voi non vedete la soluzione)
- Per quanto ne sapete, potrei aver messo i colori a caso, magari non ho neanche colorato il grafo! (voi non siete sicuri che io possiedo la soluzione)

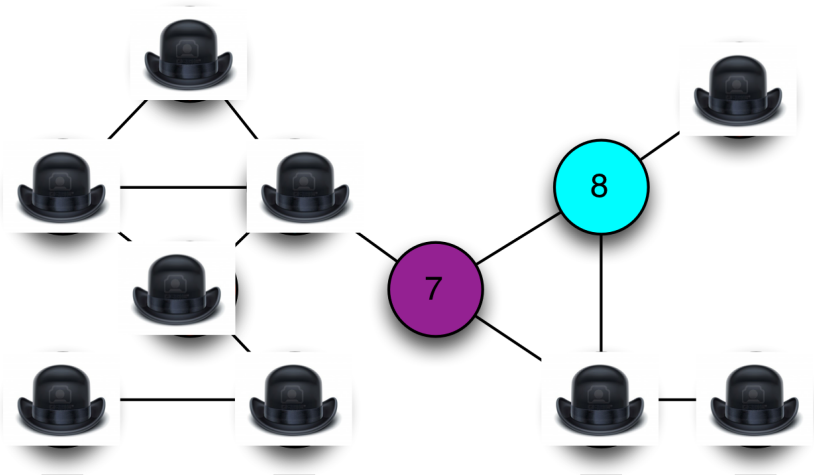
Cosa Posso fare per Convincervi?

Vi do la possibilità di controllare una piccolissima parte della mia colorazione.



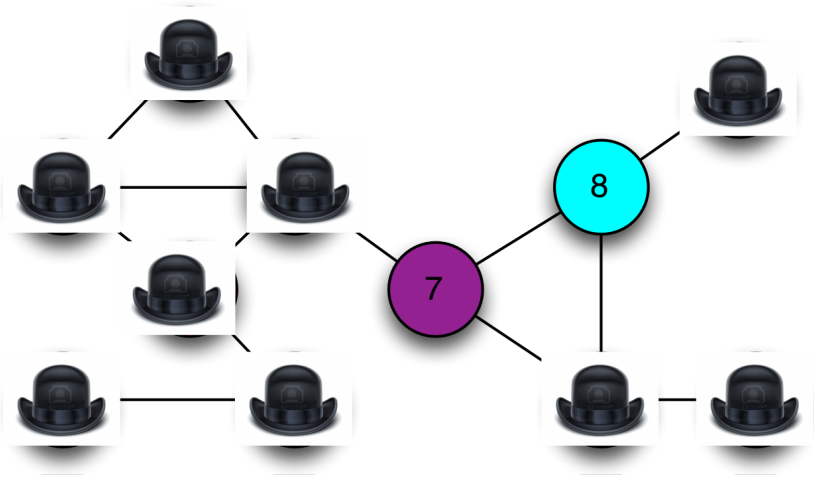
Vi lascio scegliere un arco qualsiasi del grafo.

Cosa Posso Fare per Convincervi?



A questo punto rimuoverò i due cappelli che mi avete detto, rivelando una piccola parte della mia soluzione.

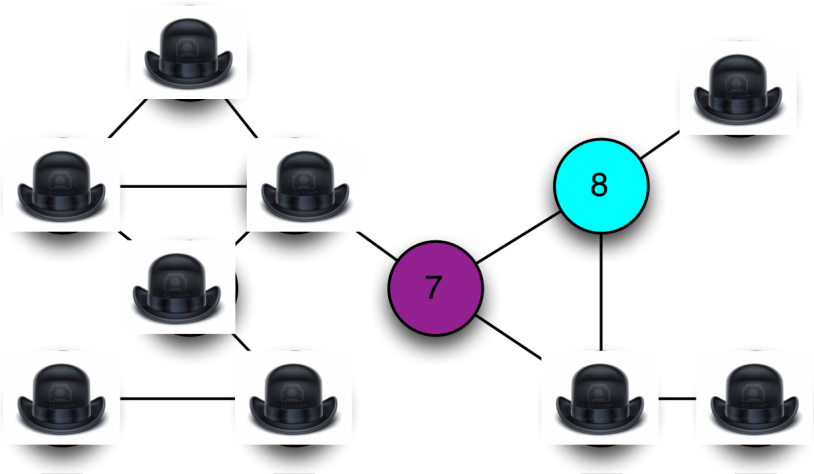
Quali Sono i Possibili Esiti?



- Se i vertici fossero dello stesso colore, sareste sicuri che sto mentendo e chiudereste la trattativa.
- Se i vertici fossero di colori diversi, c'è la possibilità che io sia onesto.

Quant'è questa probabilità?

Con che Probabilità vi ho Convinto?

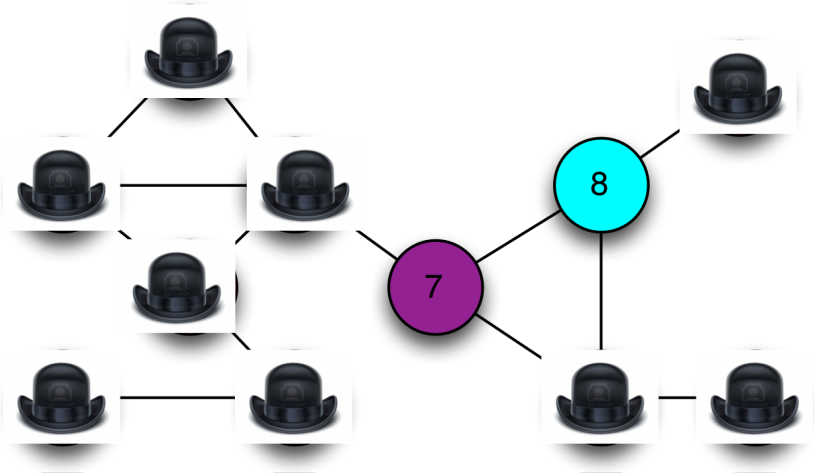


- Ammettiamo che io non abbia una 3-colorazione del grafo.
- Almeno un arco è colorato “sbagliato”.
- La probabilità con cui posso ingannarvi è minore o uguale a $(E-1)/E$.

Osservazione

Questo significa che per un grafo con 1000 archi io posso ingannarvi il 99,9% delle volte.

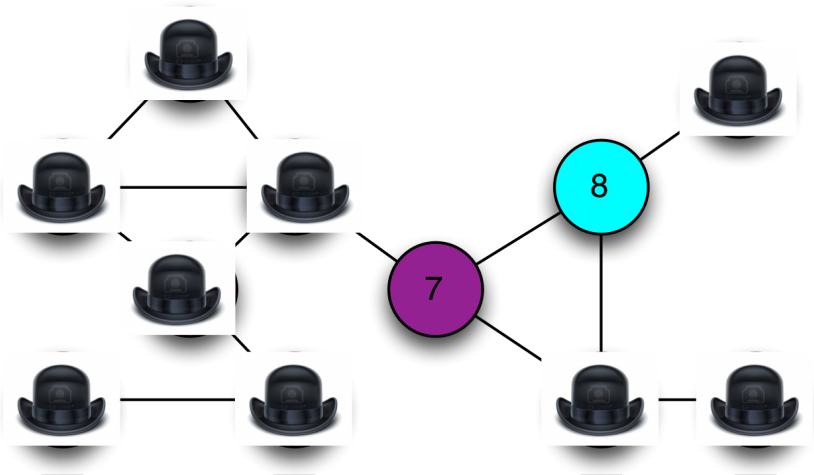
Il Problema Sembra Rimasto...



Se da un lato (nodi delle stesso colore) siete sicuri che io vi sto imbrogliando, nel secondo caso (colori diversi) siete comunque molto sospettosi della mia soluzione.

Come risolviamo il problema?

Ripetiamo il Protocollo di Nuovo!



- Puliamo tutto e usciamo tutti dalla stanza.
- Ri-entrate nella stanza, sistemate i fogli, uscite.
- Ri-entro nella stanza, con tre nuovi colori, coloro il grafo, metto i cappelli.
- Ri-entrate nella stanza, scegliete un nuovo arco.
- Togliamo i cappelli dai nodi associati all'arco scelto.

Qual è la Probabilità con cui vi Imbroglio Entrambe le Volte?

$$\frac{E-1}{E} \cdot \frac{E-1}{E} = \left(\frac{E-1}{E}\right)^2.$$

Osservazione

Quindi, per un grafo con 1000 archi, la probabilità con cui posso imbrogliare è del $99,8\%$.

Idea

Ripetere il protocollo finché non siete convinti che io abbia effettivamente la soluzione.

Domanda

Quante volte dovreste ripetere il procollo
affinchè io possa barare con probabilità al più del 10% ?

Domanda

Quante volte dovreste ripetere il procollo
affinchè io possa barare con probabilità al più del 10% ?

$$\left(\frac{E-1}{E}\right)^n = 0.1$$

$$n = \frac{\log(0.1)}{\log(\frac{E-1}{E})} \approx 2301$$

Ancora un Paio di Osservazioni

- Da un lato, voi non sarete mai certi che io sia onesto: c'è sempre una piccola possibilità che vi stia imbrogliando. Tuttavia potete ridurre questa probabilità a piacimento, finchè non sarete sufficientemente convinti che io sia onesto.
- Dall'altro lato io sono protetto, perchè ogni volta cambio i colori con cui coloro il grafo, pertanto non imparerete nulla della soluzione effettiva.

In altri termini:
Abbiamo appena descritto un protocollo zero-knowledge

Non Credetemi sulla Parola, Dimostriamolo

Un protocolli è detto “zero-knowledge” se rispetta le seguenti 3 proprietà:

1. **Completeness**: se io sono onesto, riuscirò a convincervi con alta probabilità.
2. **Soundness**: se io non sono onesto, riuscirò a convincervi solo con probabilità trascurabile.
3. **Zero-knowledgeness**: voi non imparerete nulla sulla mia soluzione dall’esecuzione del protocollo.

Le Prime due Proprietà Sono Facili da Mostrare

1. La completeness è immediata: se sono onesto allora sono sempre in grado di rispondere alla vostra domanda.
2. La soundness pure è immediata: potete fissare una probabilità oltre la quale ritenervi convinti della mia onestà, e derivare un numero di round sufficiente per raggiungere questa probabilità.
3. La zero-knowledgeness è difficile.

Proviamo a dimostrare che il protocollo non rilascia nessuna informazione riguardo la colorazione del grafo.

Uno Strano Esperimento Fantascientifico

- Supponiamo che io non sia riuscito a trovare la soluzione al vostro problema.
- Voglio comunque il vostro denaro.
- Decido di provare a imbrogliarvi, anche se non ho la soluzione.
- Vado nell'open space dei dottorandi di elettronica e prendo questo prototipo di macchina del tempo.

Uno Strano Esperimento Fantascientifico

- La mia idea iniziale è quella di usare la macchina per tornare indietro un paio di anni, e prendermi quel tempo per risolvere il problema, tuttavia la macchina è ancora un prototipo e può tornare indietro nel tempo solo di 5 minuti.
- Anche l'opzione di usare la macchina ripetutamente è fuori considerazione, perché la macchina ha bisogno di altri 5 minuti per ricaricarsi.

...anche con questo prototipo limitato posso comunque ingegnarmi...

Qual è il Mio Piano?

- Coloro il grafo a caso
- Metto i cappelli sopra i vertici.
- Se scegliete un arco per il quale i colori sono diversi. In questo caso tiro un sospiro di sollievo e scopro i cappelli.
- Inevitabilmente prima o poi sceglierete un arco per il quale i colori sono uguali.
  1. Mondo reale: fine.
  2. Mondo fantascientifico: uso la macchina del tempo: torno indietro 5 minuti, ricoloro il grafo e aspetto che voi rifacciate la vostra scelta.

Osservazione

Osservazione (Importante!)

Dal vostro punto di vista (non sapendo che io sto usando una macchina del tempo) lo scambio di messaggi è indistinguibile da quello che avremmo con una interazione onesta. Tuttavia, in questo modo fanstascientifico io non ho assolutamente idea di come 3-colorare il grafo.

Qual è il Punto di Tutto Questo?

Nel mondo reale nessuno può ingannarvi in questo modo, e il protocollo che abbiamo descritto rispetta sia la correctness che la soundness.

tradotto:

Dopo molti round voi sarete convinti (con probabilità quasi 1) che io so veramente 3-colorare il grafo in questione.

Quello che abbiamo mostrato è semplicemente questo: se io potessi fare un "rewind" del tempo, potrei falsificare un protocollo valido anche se non avessi alcuna informazione sulla colorazione effettiva del grafo.

Osservazione

Dal vostro punto di vista i due protocolli sono identici, e pertanto veicolano la stessa quantità di informazioni utili.

Qual è il Punto di Tutto Questo?

Figura: Interazione reale

$\longleftarrow$ Ind. $\longrightarrow$

Figura: Interazione fantastica

$\downarrow$

Informazioni (sul mio segreto)
che potete estrarre da
un'interazione reale

$\downarrow$

Informazioni (sul mio segreto)
che potete estrarre da
un'interazione fantascientifica

Qual è il Punto di Tutto Questo?

Figura: Interazione reale

$\longleftarrow$ Ind. $\longrightarrow$

Figura: Interazione fantastica

$\downarrow$

Informazioni (sul mio segreto)
che potete estrarre da
un'interazione reale

$\downarrow$

0

Qual è il Punto di Tutto Questo?

Figura: Interazione reale

$\downarrow$

0

$\longleftarrow$ Ind. $\longrightarrow$

Figura: Interazione fantastica

$\downarrow$

0

Rimane Solo da Mostrare che Esiste un Macchina del Tempo

Ovviamente no: trasportiamo questo esempio nel mondo digitale.

- Liberiamoci dei cappelli.
- Liberiamoci delle macchine del tempo.

Un Equivalente Digitale dei Cappelli

La funzione dei cappelli era quella di nascondere la nostra colorazione, ma allo stesso tempo vincolarmi e impedirmi di cambiarla in un secondo momento

Figura: Mondo fantascientifico: cappelli

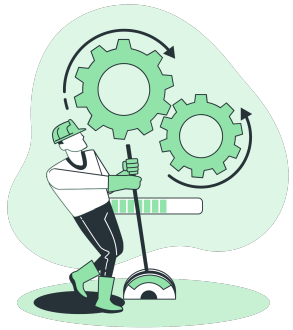


Figura: Mondo digitale: funzioni di hash

Osservazione

Le funzioni di hash permettono a un partecipante di fare un "commit" di un certo messaggio, tenendo segreto il suo contenuto, per poterlo rivelare in seguito. Uno scambio di messaggi di questo tipo definisce uno schema di commitment.

Costruiamo il Protocollo

indichiamo i partecipanti con i nomi di “prover” e “verifier”.

- Il prover codifica la sua colorazione con i valori 0,1,2, dopodichè (grazie alle funzioni di hash) genera un commitment per questa colorazione.
- Il commitment viene inviato al verifier
- Il verifier sceglie un arco
- Il prover rivela i valori associati al commitment dei vertici associati all’arco.

Un Equivalente Digitale Della Macchina del Tempo

(proviamo la zero-knowledgeness)

Nel mondo digitale gli enti che interagiscono sono programmi, e il rewinding non è una cosa così strana, dopotutto.

Figura: Mondo fantascientifico: macchina del tempo

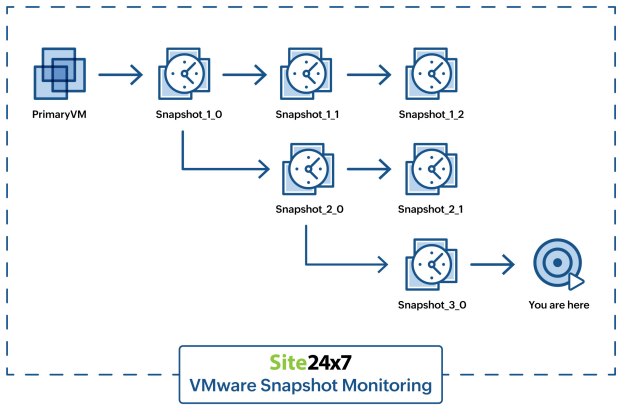


Figura: Mondo digitale: snapshot oppure restart

In Conclusione

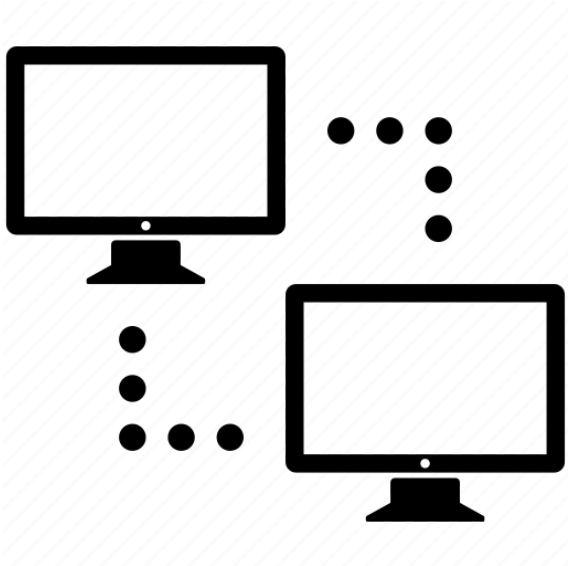


Figura: Interazione reale

$\longleftarrow$ Ind. $\longrightarrow$

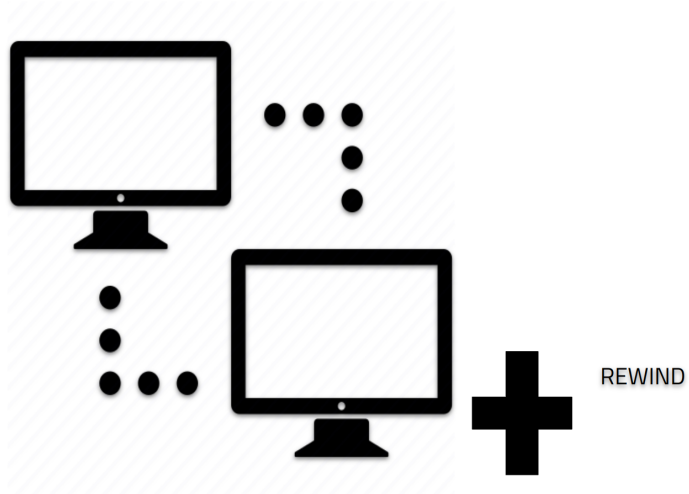


Figura: Interazione fantastica

$\downarrow$

Informazioni che un verifier può estrarre da un'interazione reale

$\downarrow$

Informazioni che il verifier può estrarre da un'interazione ideale

Recap

- Il protocollo è corretto.
- Il protocollo è sound: la soundness vale nel mondo reale/digitale, ovvero quando non c'è nessun trick del tipo macchine del tempo o rewind.
- Il protocollo è anche zero knowledge: abbiamo mostrato che l'informazione estratta da ogni "programma verifier" in un protocollo onesto è la stessa quantitià di informazione che è possibile estrarre da un'interazione ideale, in cui il prover non conosce nessun segreto. Il protocollo non rilascia pertanto nessuna informazione sensibile.

Prima e Seconda Parte

tutto insieme

Il problema della 3-colorazione di un grafo è NP-completo.

Ogni altro problema in NP può essere traslato in un'istanza del nostro problema!

Possiamo costruire protocolli zero-knowledge per qualsiasi problema in NP!

Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems

ODED GOLDREICH

Technion, Haifa, Israel

SILVIO MICALI

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

AVI WIGDERSON

Hebrew University, Jerusalem, Israel

Abstract. In this paper the generality and wide applicability of *zero-knowledge proofs*, a notion introduced by Goldwasser, Micali, and Rackoff is demonstrated. These are probabilistic and interactive proofs that, for the members of a language, efficiently demonstrate membership in the language without conveying any additional knowledge. All previously known zero-knowledge proofs were only for number-theoretic languages in NP ∩ CoNP.

Under the assumption that secure encryption functions exist or by using "physical means for hiding information," it is shown that all languages in NP have zero-knowledge proofs. Loosely speaking, it is possible to demonstrate that a CNF formula is satisfiable without revealing any other property of the formula, in particular, without yielding neither a satisfying assignment nor properties such as whether there is a satisfying assignment in which $x_1 = x_3$ etc.

It is also demonstrated that zero-knowledge proofs exist "outside the domain of cryptography and number theory." Using no assumptions, it is shown that both graph isomorphism and graph nonisomorphism have zero-knowledge interactive proofs. The mere existence of an interactive proof for graph nonisomorphism is interesting, since graph nonisomorphism is not known to be in NP and hence no

Fine?

Extra

Dalla Teoria alla Pratica

Ricondurre ogni problema in NP a 3-coloring è computazionalmente infattibile.

Quello che abbiamo mostrato prima è un risultato di "fattibilità". Una volta dimostrato che qualcosa è possibile, il passo successivo è renderlo efficiente.

Lo schema di Schnorr

Uno schema ZK efficiente

Osservazione

È la base di molti protocolli moderni.

Contesto:

Alice ha pubblicato la sua chiave pubblica al mondo, e in seguito vuole dimostrare di conoscere la chiave segreta corrispondente a quella chiave pubblica.

Nel Dettaglio

La coppia chiave pubblica/privata ha un formato molto specifico.

$$\left(p \text{ primo. } g \text{ generatore di un gruppo ciclico di ordine } q. \right)$$

Per generare una coppia di chiavi, Alice deve prima scegliere un numero intero casuale a compreso tra 1 e q, quindi calcolare la coppia di chiavi come:

$$SK_A = a \qquad PK_A = g^a \pmod{p}$$

Stesso tipo di chiave utilizzato per Diffie-Hellman e l'algoritmo di firma DSA.

Il Protocollo

Alice tiene la sua chiave segreta per sé, ma è libera di pubblicare la sua chiave pubblica per il mondo. In seguito, quando vuole dimostrare di conoscere la sua chiave segreta, conduce il seguente semplice protocollo interattivo con Bob.

PROVER

Random k in $\{1, \ldots, q\}$.

$$\xrightarrow{h = g^k \pmod p}$$

$$\xleftarrow{c}$$

$$\xrightarrow{s = ac + k \pmod p}$$

VERIFIER

Random $c \in \{1, \ldots, q\}$.

Check
$g^s \equiv PK_A^c \cdot h \pmod p$.

Completeness

iniziamo a vedere se il protocollo rispetta la completeness

La completezza è abbastanza facile da vedere semplicemente facendo un po' di sostituzioni:

$$g^s \equiv PK_A^c \cdot h \pmod{p}$$
$$g^{ac+k} \equiv (g^a)^c \cdot g^k \pmod{p}$$
$$g^{ac+k} \equiv g^{ac+k} \pmod{p}$$

Soundness

spunto di riflessione personale

Zero Knowledge(ness)

Dotiamoci della nostra "macchina del tempo" e proviamo a dimostrare la conoscenza di un segreto a per qualche chiave pubblica $g^a \pmod p$, anche se in realtà non conosciamo il valore a.

1. Per prima cosa, mandiamo g^{k_1} come primo messaggio del Prover e vediamo quale sfida c sceglie il verifier.
2. Prepariamo il commitment e la risposta alla challenge c. Dopodichè facciamo rewind del verifier finchè non ci chiede la challenge c:
   1. Prendiamo un intero casuale $z \in \{0, \dots, q-1\}$.
   2. Calcoliamo $g^z \cdot g^{a(-c)} = g^{k_2}$.
   3. Pubblichiamo k_2.
3. Rispondiamo con z.

Passiamo il check?

$$g^{\text{responce}} \equiv PK_A^{\text{challenge}} \cdot \text{commitment} \iff g^z \equiv PK_A^c \cdot k_2$$
$$\iff True$$

Per Chi Conoscesse Queste Dimostrazioni

Quella che abbiamo mostrato è la *honest verifier* zero knowledge

Da Interattivo a non Interattivo

Finora abbiamo mostrato come utilizzare il protocollo di Schnorr per provare interattivamente la conoscenza di una chiave segreta a che corrisponde a una chiave pubblica g^a. Questo è un protocollo incredibilmente utile, ma funziona solo se il nostro verifier è online ed è disposto a interagire con noi.

Possiamo renderlo non-interattivo?

Sembra difficile, poiché il protocollo si basa fondamentalmente sul verifier che sceglie una sfida casuale.

Un Trick
(Fiat-Shamir)

Se abbiamo una funzione hash, possiamo convertire un protocollo interattivo in uno non interattivo semplicemente usando la funzione hash per scegliere la challenge.

Il protocollo di Schnorr, trasformato, diventa

1. Il Prover sceglie g^k (proprio come nel protocollo interattivo).
2. Ora, il prover calcola la sfida come $c = H(g^k||M)$, dove M è una stringa aggiuntiva opzionale.
3. Calcola $ac + k \mod q$ (proprio come nel protocollo interattivo).

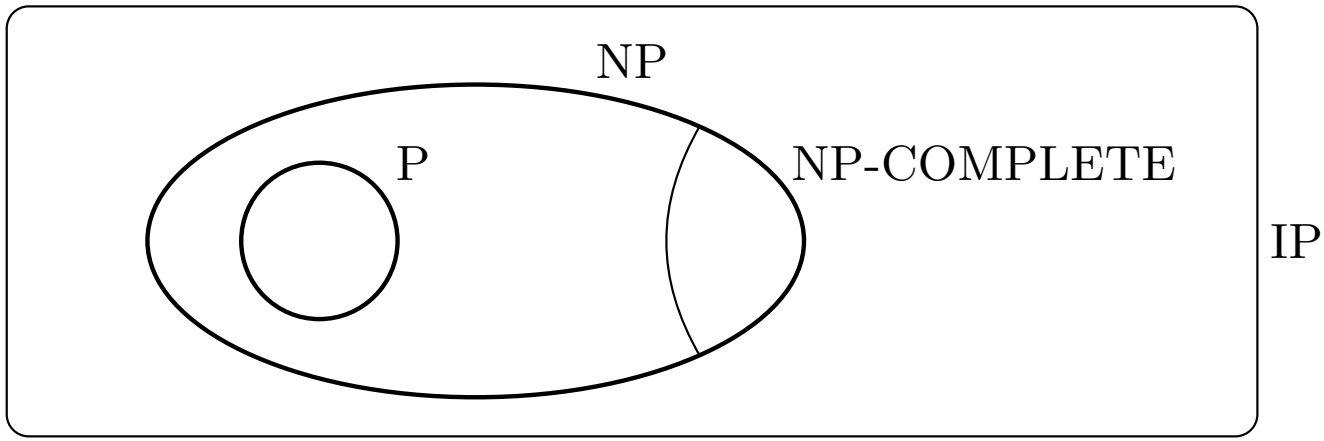
Se la funzione hash è "abbastanza forte", il risultato è una prova di conoscenza del valore a completamente non interattiva, che il Prover può inviare al Verifier.

Firme Digitali!

La cosa particolarmente interessante di questo protocollo è che non è solo una prova di conoscenza, è anche uno schema di firma.

Se si inserisce un messaggio nel valore (opzionale) M, si ottiene una firma su M, che può essere prodotta solo da qualcuno che conosce la chiave segreta a.

Recap Conclusivo



Grazie