Proof theory

Introduzione alle zero knowledge proofs

Luca Donno - @donnoh - 02/03/2023

Argomenti discussi
brace yourselves

- Fondamenti di calcolabilità
- Basi di decidibilità
- Basi di complessità
- P vs NP
- NP proofs
- Interactive proofs
- Intuizione per ZKPOK

Che cos'è una proof

Wikipedia: A proof is sufficient evidence or a sufficient argument for the truth of a proposition.

Problemi: cosa vuol dire sufficient? cosa vuol dire truth? Argomenti di interesse dell'epistemologia, out of scope per questa presentazione.

...intuitivamente: Teorema di Euclide (*Elements*, 300 BC)

Claim: i numeri primi sono infiniti.

Proof: Sia $p_1, \ldots, p_n$ un insieme finito di numeri primi, dimostro che ne esiste almeno un altro. Sia $P = p_1 \cdot \ldots \cdot p_n$. Sia $q = P + 1$. Due casi:

1) q è primo.

2) q non è primo, quindi c'è un fattore primo p che lo divide. Se p appartiene alla lista $p_1, \ldots, p_n$, allora p divide anche P, e di conseguenza divide anche $q - P = P + 1 - P = 1$. Ciò non è possibile quindi p non appartiene alla lista.

Proof checking in Coq

```
From Coq Require Import ssreflect ssrfun ssrbool.
From mathcomp Require Import eqtype ssrnat div prime.
(* A nice proof of the infinitude of primes, by Georges Gonthier *)
Lemma prime_above m : {p | m < p & prime p}.
Proof.
have /pdivP[p pr_p p_dv_m1]: 1 < m`! + 1
  by rewrite addn1 ltnS fact_gt0.
exists p => //; rewrite ltnNge; apply: contraL p_dv_m1 => p_le_m.
by rewrite dvdn_addr ?dvdn_fact ?prime_gt0 // gtnNdvd ?prime_gt1.
Qed.
```

Proof checker

Un proof checker implementa una **funzione di decisione**. Formalmente:

$$f : \mathbb{N} \to \{0,1\}$$

Esempio: x è un numero pari $\implies f(x) = x \,\%\, 2$.

Data una funzione di decisione, posso prendere l'insieme degli elementi per cui essa restituisce 1:

$$A = \{x \mid f(x) = 1\} \subseteq \mathbb{N}$$

Ogni insieme A così definito è detto proprietà.

ATTENZIONE: nell'esempio dei numeri pari, a $f(x)$ non passo la dimostrazione che x sia pari oppure no, ma è la funzione stessa che lo calcola (i.e. che lo dimostra e verifica). La distinzione e la motivazione diventeranno più chiare avanti. <u>Intuizione</u>: è spesso più facile verificare una dimostrazione data che inventarne una da zero.

Tutte le proprietà sono decidibili?

Hilbert: probabilmente sì.

Turing: no: Halting problem, non esiste una funzione di decisione per la convergenza dei programmi.

Teorema di Rice: nessuna proprietà interessante (estensionale) dei programmi è decidibile.

Come fare? Le proprietà si possono approssimare con sistemi di tipo.

Basi di complessità computazionale

Alcune funzioni crescono più velocemente di altre nonostante costanti.

Esempio: $a_1x^2 + a_2 > b_1x + b_1$ per ogni $x > k$ con k opportuno, per ogni scelta di a_1, a_2, b_1, b_2. Formalmente, si dice che $O(x) \subset O(x^2)$.

Formalmente: $O(f) := \{g : \mathbb{N} \to \mathbb{N} \mid \exists c\,.\,\forall n\,.\,g(n) \leq cf(n) + c\}$.

$O(1) \subset O(\log n) \subset O(n) \subset (n \log n) \subset O(n^2) \subset O(n^3) \subset \ldots \subset O(2^n) \subset O(3^n) \subset \ldots \subset O(n^n)$

Inoltre: $O(n^n) = O(n!)$ e $O(n \log n) = O(\log(n!))$ (da relazione di Sterling).

Classi di complessità

Una **classe di complessità** è un insieme di problemi con una certa complessità. Tipicamente questi sono problemi di decisione definiti su **macchine di Turing** M.

— $time_M(x)$ è il tempo esecuzione di M su input x, ovvero il numero di passi della computazione.

— $t_M(n) := \max\{time_M(x) : |x| = n\}$ ovvero il tempo massimo impiegato da M su input di dimensione n.

— $DTIME(f) := \{L \subseteq \Sigma^* : \exists M . L = L_M \wedge t_M \in O(f)\}$.

— $P := \cup_{c \in \mathbb{N}} DTIME(n^c)$.

— NP (informalmente) è la classe di problemi tali per cui esiste una proof verificabile in tempo polinomiale da una macchina di Turing (Teorema di proiezione).

...intuitivamente

I problemi nella classe P sono problemi risolvibili in tempo polinomiale, che è molto meglio di esponenziale o fattoriale, quindi diciamo che questi problemi sono facili da risolvere.

I problemi nella classe NP sono problemi per cui esiste una proof verificabile in tempo polinomiale, quindi diciamo che sono problemi di facile verifica.

P vs NP

Problemi in P: ricerca di un elemento in un array disordinato, bubble sort, simplesso, game of life, type inference.

Problemi in NP: soddisfacibilità booleana, problema del commesso viaggiatore, sudoku.

Banalmente, tutti i problemi in P sono anche in NP: posso passare anche una proof vuota, il problema si può già risolvere in tempo polinomiale.

> **Domanda da letteralmente un milione di dollari** (Millenium Prize Problems): tutti i problemi in NP sono anche in P? Ovvero, tutti i problemi di facile verifica sono anche facili da risolvere?

THE PATH TO ZK PROOFS 😎😎😎

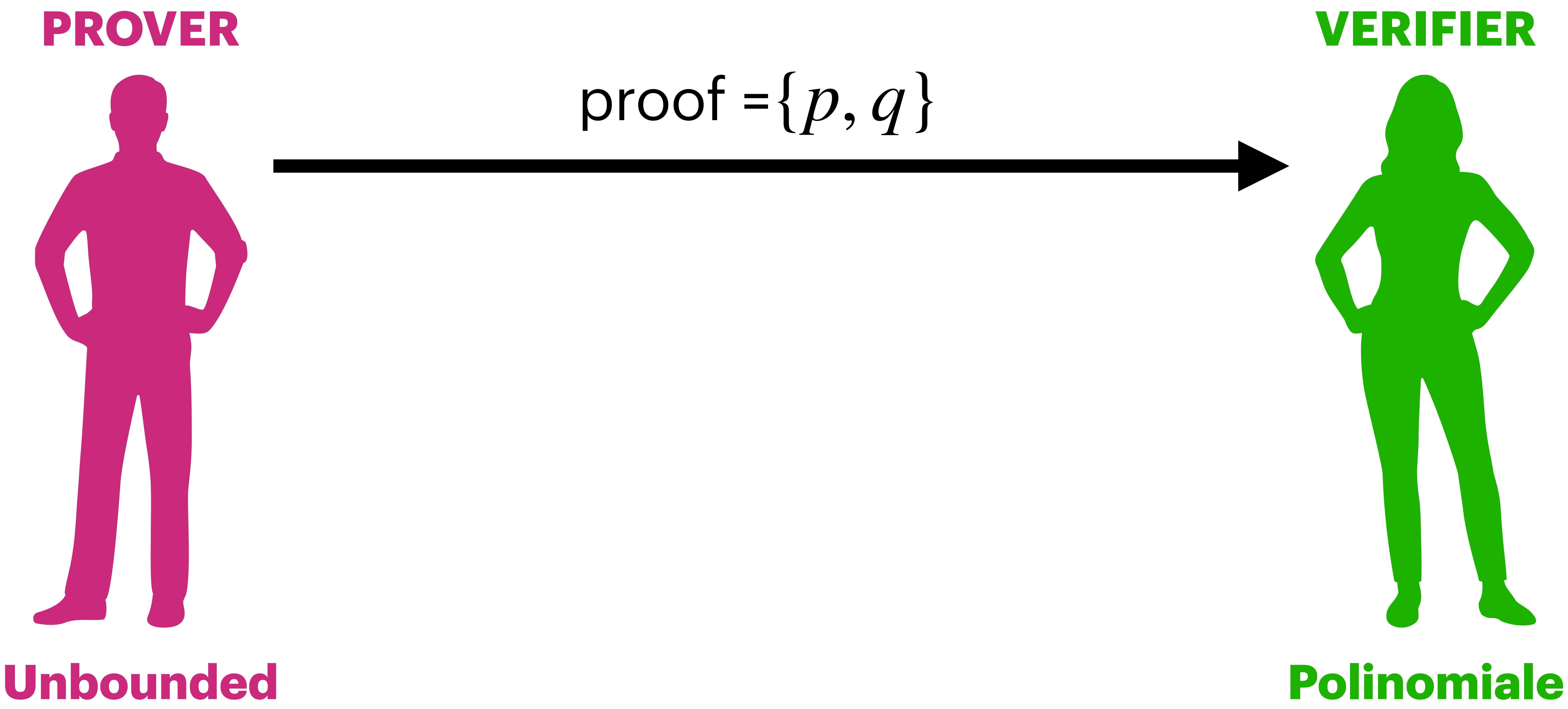
Esempio 1 di NP proof

con Prover illimitato e Verifier Polinomiale

Dopo l'interazione, il Verifier sa che:

1) N è il prodotto di due numeri primi.
2) I due numeri primi p e q !

Claim: N è il prodotto di due numeri primi.



V verifica che p e q siano primi (polinomiale, AKS primality test, 2002), li moltiplica insieme (polinomiale) e verifica che $N = pq$. Se vero accetta, altrimenti rifiuta.

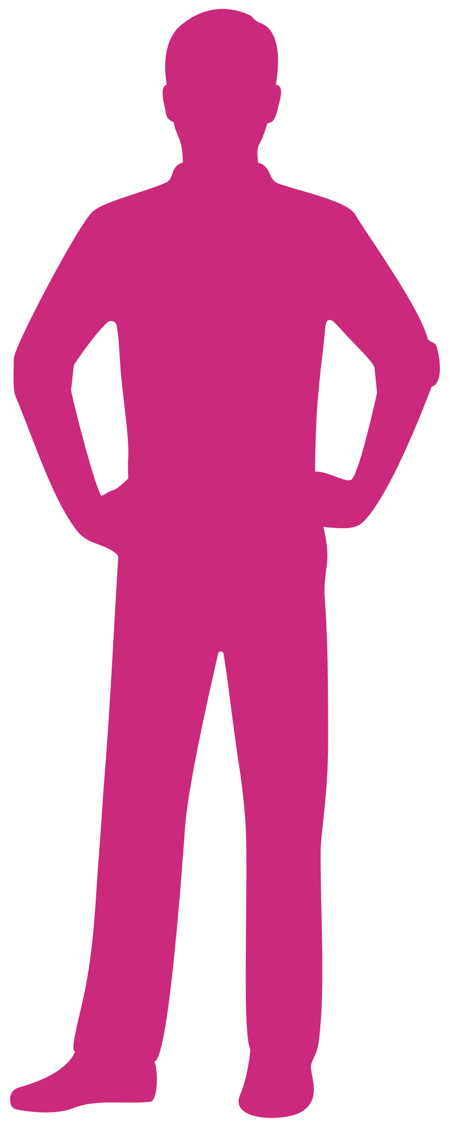
Esempio 2: residui quadratici

Dopo l'interazione, il Verifier sa che:

1) y è un residuo quadratico mod N.
2) $\sqrt{y}$ **!**

Claim: y è un residuo quadratico mod N, i.e. $\exists x \in \mathbb{Z}_N^*$ t.c. $y = x^2 \mod N$.

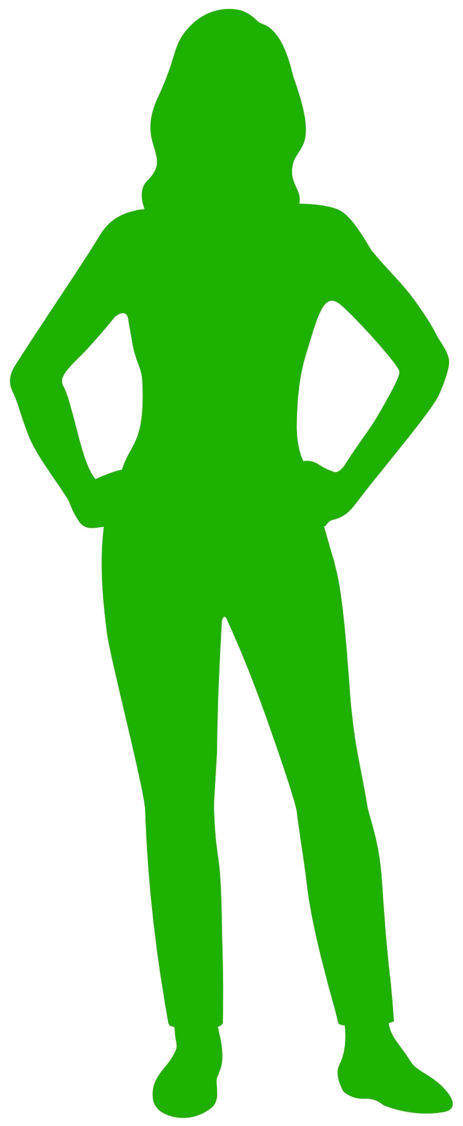
PROVER



Unbounded

proof = x $\longrightarrow$

VERIFIER



Polinomiale

Non si può calcolare $\sqrt{y}$ in tempo polinomiale, quindi **V** ha bisogno di un **Prover**.

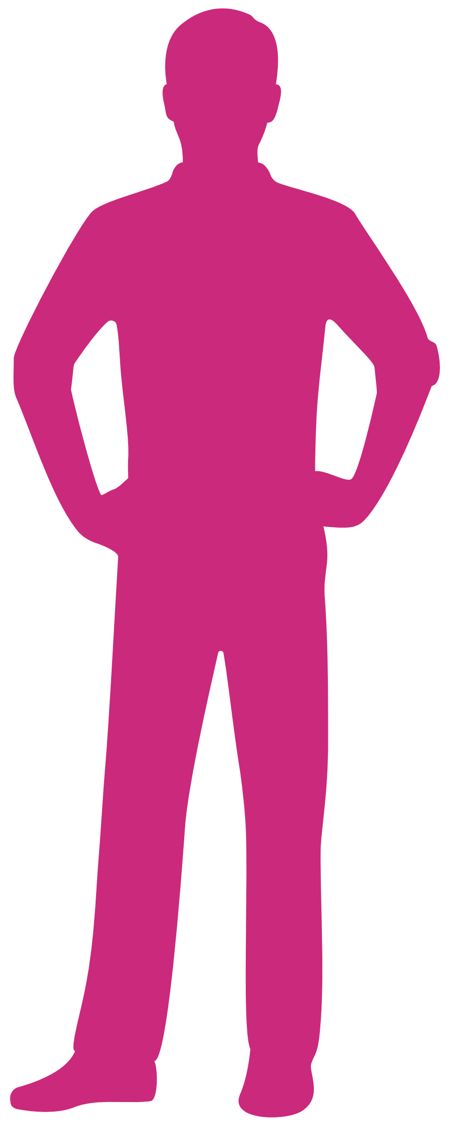
V prende x, calcola x^2 e verifica che sia uguale a y. Se vero accetta, altrimenti rifiuta.

Esempio 3: isomorfismo tra grafi

Dopo l'interazione, il Verifier sa che:
1) I due grafi sono isomorfi.
2) L'isomorfismo π !

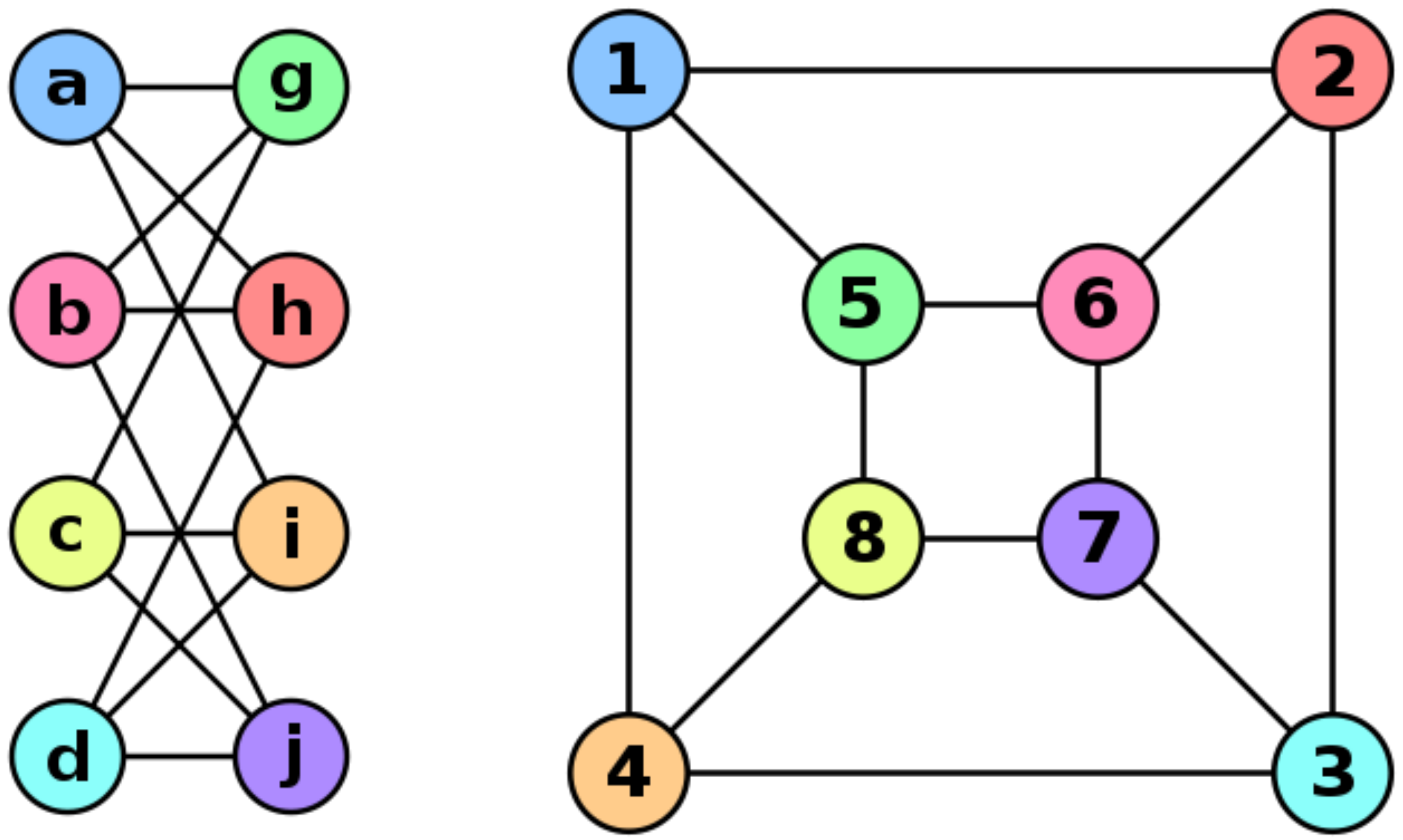
Claim: i due grafi sono isomorfi.

PROVER

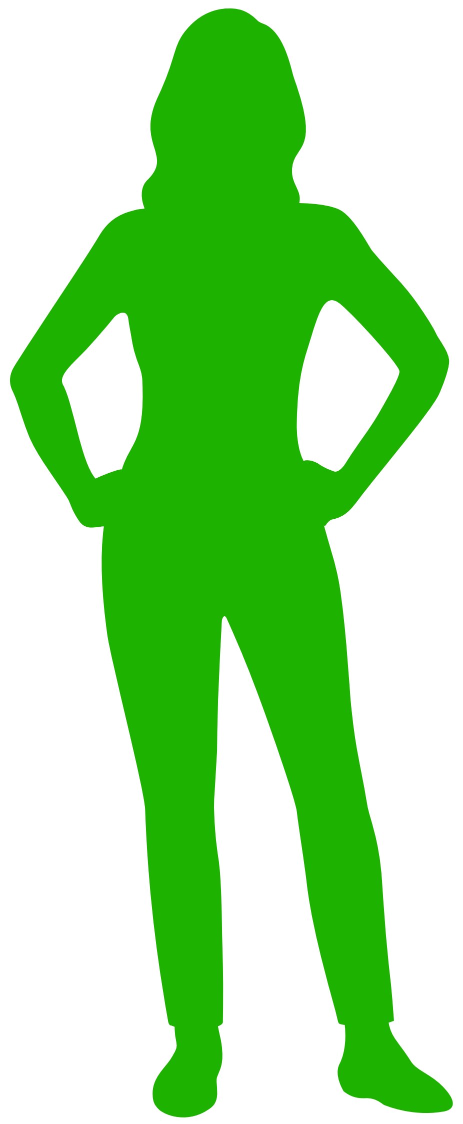


Unbounded

$$\pi : [\mathbb{N}] \rightarrow [\mathbb{N}]$$



VERIFIER



Polinomiale

Non si conosce un algoritmo che trova un isomorfismo in tempo polinomiale, quindi **V** ha bisogno di un **Prover**.

V: se $\forall i,j\,.\,(\pi(i),\pi(j)) \in E_1 \iff (i,j) \in E_0$ accetta altrimenti rifiuta.

Proprietà dei problemi NP

- **Completeness**: claim veri hanno una (short) proof.

$$x \in L \implies \exists w \in \mathbb{B}^* \,.\, |w| \text{ è } poly(|x|) \text{ e } \mathbf{V}(x, w) = 1.$$

- **Soundness**: claim falsi non hanno proof.

$$x \notin L \implies \forall w \in \mathbb{B}^* \,.\, \mathbf{V}(x, w) = 0.$$

C'è un altro modo per fare queste dimostrazioni?

Idea: invece di dimostrarti il claim, ti dimostro che potrei dimostrartelo se volessi

Implicazione: il **Verifier** è convinto che il claim è vero senza venire a conoscenza della witness!

Micali, Goldwasser, Rackoff ('82 - '85)

Zero Knowledge Interactive Proofs (ZKIP)

Due nuovi ingredienti rispetto alle NP proof:

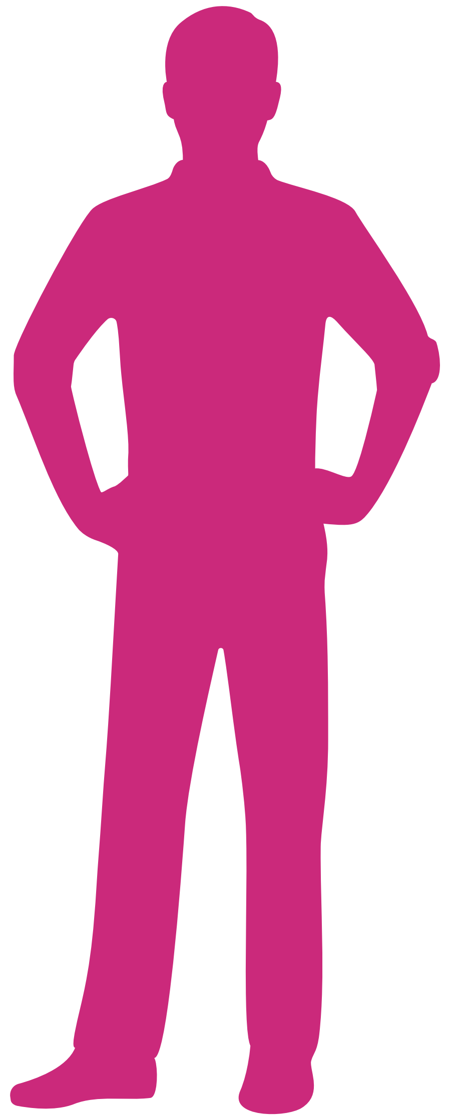
- **Interazione**: il **Prover** e il **Verifier** interagiscono non trivialmente (con al massimo un numero polinomiale di passi).
- **Randomness**: il **Verifier** è randomizzato, ovvero può lanciare monete. Ciò implica che esso può avere più possibili esecuzioni e si ammette che possa sbagliare ad accettare o rifiutare una prova con una piccola probabilità.

$\Longrightarrow$ **Probabilistic Polynomial Time (PPT)**

Esempio 1 di ZKIP

Claim: questa pagina contiene due colori.

PROVER



Può distinguere i colori

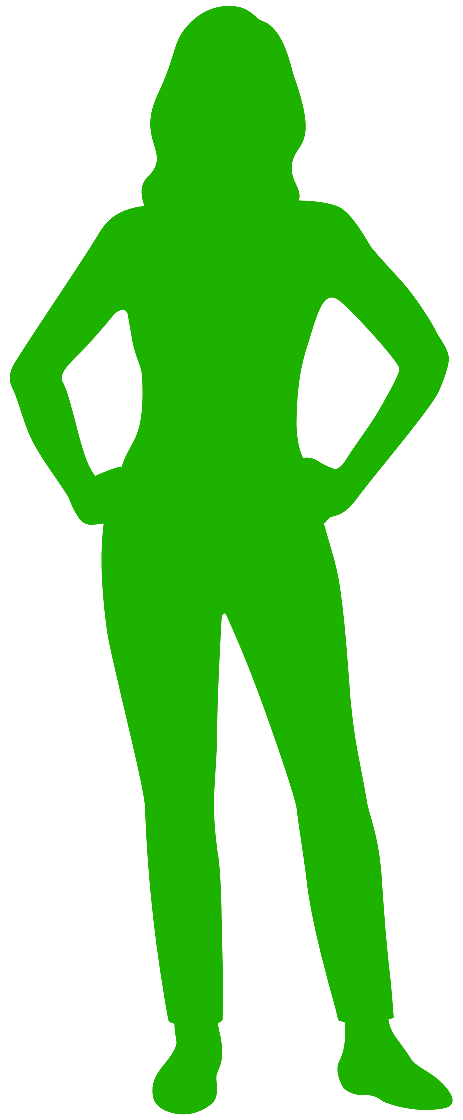
1) Lancia una moneta, se esce testa ruota la pagina 180° altrimenti non fare nulla.

pagina

2) Se la pagina è stata ruotata, setta $coin' = heads$ altrimenti setta $coin' = tails$

$coin'$

VERIFIER



Colorblind

3) Se **P** ha ragione, forse ci sono due colori, altrimenti rifiuta.

Analisi dell'esempio 1 di ZKIP

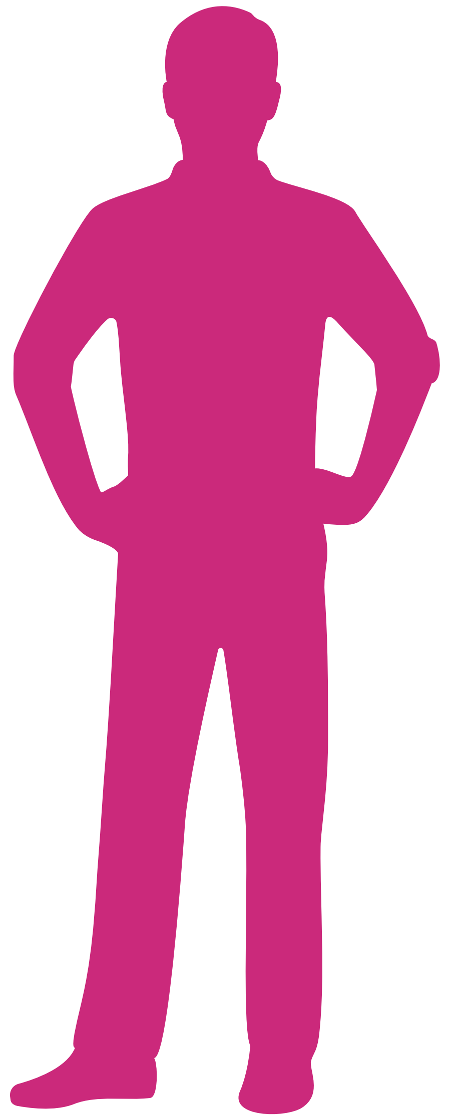
- **Completeness**: se ci sono effettivamente due colori distinti e il **Prover** sa distinguere i colori, allora il **Verifier** accetta.
- **Soundness**: se c'è un solo colore oppure il **Prover** non sa distinguerli, allora il **Verifier** accetta con probabilità $1/2$ (o inferiore).

Se la procedura viene ripetuta k volte, allora la probabilità di sbagliare scende a $1/2^k$!

Esempio 2 di ZKIP: residui quadratici

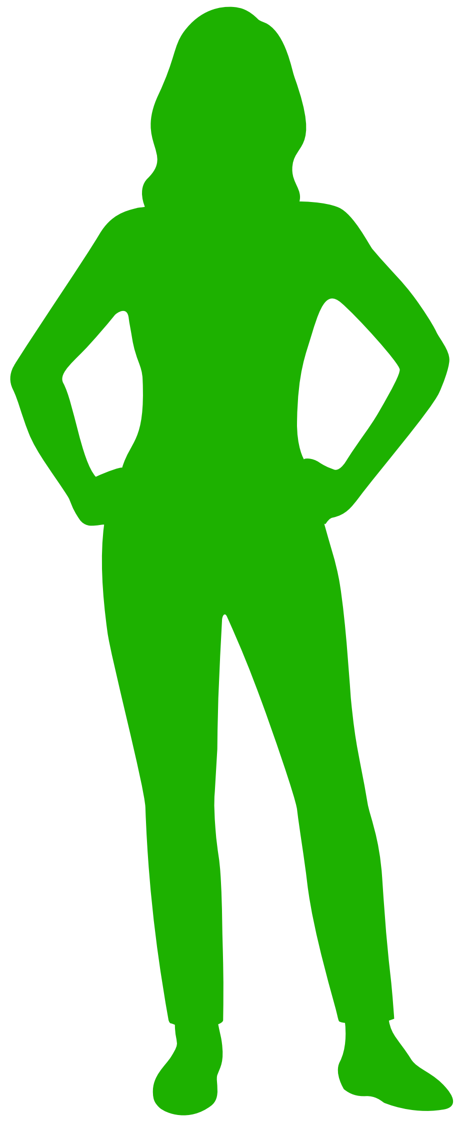
Claim: $QR = \{(N, y) \,|\, y = x^2 \mod N\}$

PROVER



Unbounded

VERIFIER



Polinomiale

1) Scegli $1 \leq r \leq N$
t.c. $gcd(r, N) = 1$.

invia $s = r^2 \mod N$ →

e di': se ti inviassi sia $\sqrt{s}$ che $\sqrt{sy} \mod N$, tu saresti convinto che il claim è vero (ma non voglio darti entrambi!). Quindi te ne darò solo uno, ma scegli tu quale.

2) Lancia una moneta b e scegli uno dei due.

Se **P** sta mentendo, non può averli entrambi (altrimenti non starebbe mentendo)

← b

3) Se $b = 1$ allora
$z = r$ altrimenti $z = r\sqrt{y}$

z →

4) Accetta solo se
$z^2 = sy^{1-b} \mod N$

Analisi dell'esempio 2 di ZKIP

- **Completeness**: se il claim è vero, **V** accetta.
- **Soundness**: se il claim è falso, **V** accetta con probabilità $1/2$.

Se la procedura viene ripetuta k volte, accetta con probabilità $1/2^k$.

Che cosa ha reso possibile questa proof?

- Il claim da dimostrare ha molte possibili proof e il **Prover** ne sceglie una a caso.
- Ognuna di queste proof è formata esattamente da due parti: vedere solo una delle due non dà al **Verifier** nessuna knowledge; mentre vederle implica una soundness del 100%.
- Il **Verifier** sceglie a caso quale delle due parte della prova ricevere dal **Prover**. L'abilità del **Prover** di fornirne una o l'altra è ciò che convince il **Verifier**.

Spoiler prossima puntata

- Definizione formale di interactive proof
- Definizione formale di zero knowledge
- The simulation paradigm
- The extractor paradigm
- Zero Knowledge Proof of Knowledge
- Complementi di complessità

thx <333