



# Progetto Mastermind

di Simone Guardati  
Tutor: Andrea Galassi

---

# Obiettivi

- Sviluppare una intelligenza artificiale sub-simbolica capace di *giocare* al gioco da tavolo Mastermind

Per fare ciò dovremo affrontare due differenti sfide:

1. Creare un database di partite giocate
2. Creare ed *allenare* degli algoritmi di *machine learning*

---

# Mastermind chi ?!?

---

# Mastermind

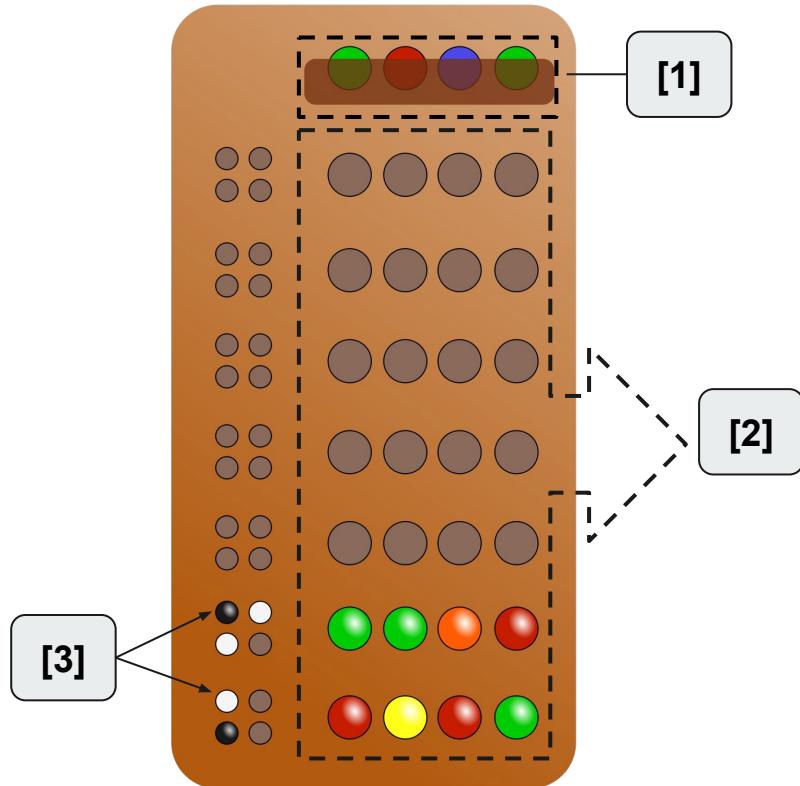
- Gioco da tavolo del 1970
- Gioco di decriptazione
- Due giocatori
- In figura una delle prime versioni messe in commercio



---

# Regole di mastermind

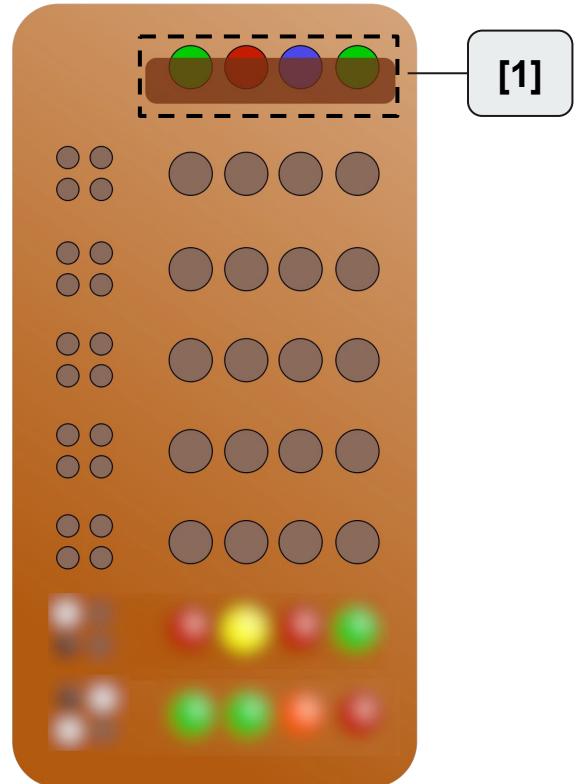
- [1] Il giocatore *codificatore* crea una password e la tiene nascosta al giocatore *decodificatore*.
- [2] Il *decodificatore* ha 10 tentativi per capire quale sia la password.
- [3] Ad ogni tentativo del *decodificatore* il *codificatore* assegna dei suggerimenti:
  - Pedine bianche
  - Pedine nere



---

# Esempio di partita

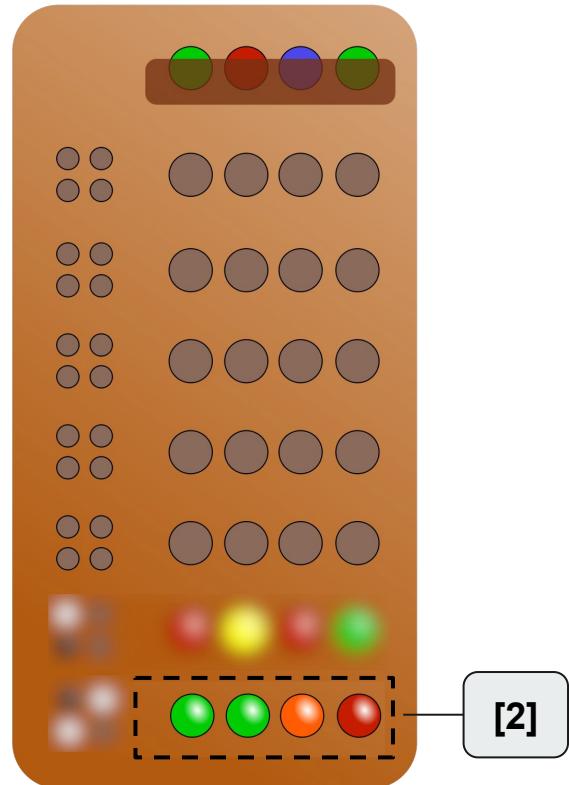
- Giocatore:  
**CODIFICATORE**
- Azione:  
**Inventa una password e la posiziona nella scacchiera dietro una barriera per non farla vedere al decodificatore. [1]**



---

# Esempio di partita

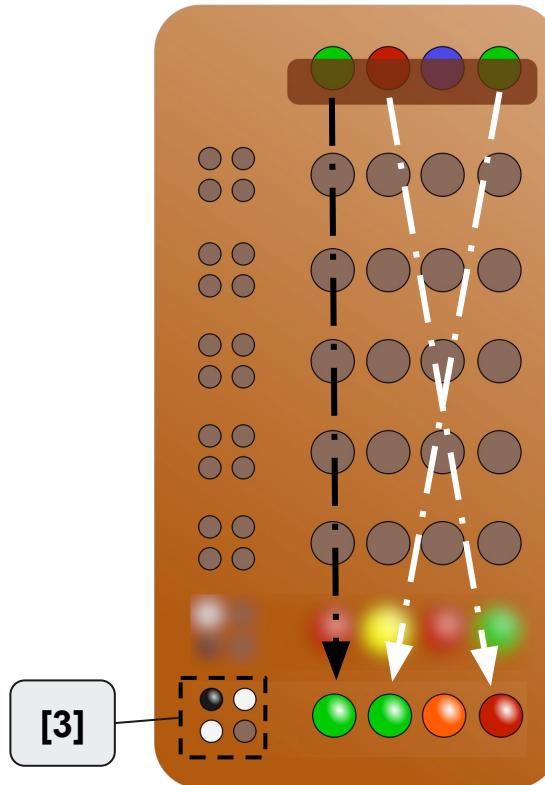
- Giocatore:  
**DE-CODIFICATORE**
- Azione:  
**Propone una password** posizionando le palline colorate nella prima posizione libera della scacchiera. **[2]**



---

# Esempio di partita

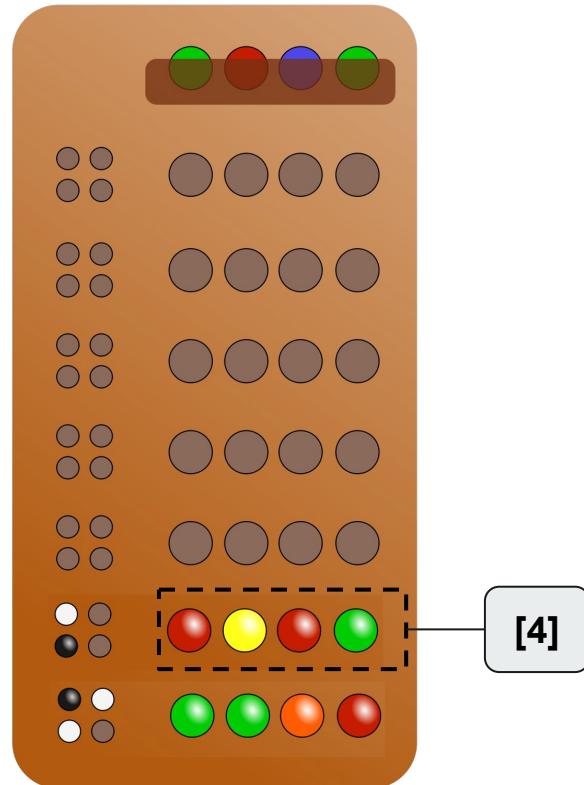
- Giocatore:  
**CODIFICATORE**
- Azione:  
**Assegna i suggerimenti. [3]**
  - Pedine Bianche:  
tanto quante sono le palline del giusto colore nella giusta posizione
  - Pedine Nere:  
tante quante sono le palline del giusto colore ma in una posizione sbagliata. In caso



---

# Esempio di partita

- Giocatore:  
**DE-CODIFICATORE**
- Azione:  
Gioca una ulteriore combinazione,  
cercando di trovare la soluzione. [4]
  - Possibilmente in meno di  $n$  tentativi.



Prima parte:  
*Matematica, algoritmi e  
dataset*

---



# Suggerimenti: l'algoritmo

- Pedine bianche:

$N\_BIANCHE = 0$

Per ogni possibile colore:

A = conta quante volte il colore è presente nel *tentativo*

B = conta quante volte il colore è presente nella *password*

C = minimo tra A e B

$N\_BIANCHE = N\_BIANCHE + C$

$N\_BIANCHE = N\_BIANCHE - N\_NERE$

***tentativo*** = codice proposto dal *de-codificatore*

***password*** = codice nascosto del *codificatore*

- Pedine nere:  
Lasciato per esercizio al lettore

---

# L'algoritmo aiuta in casi particolari

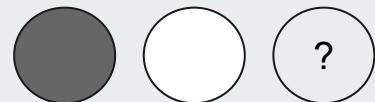
**TENTATIVO**



**PASSWORD**



**SUGGERIMENTI**



?

Una o due pedine bianche?

---

# L'algoritmo aiuta in casi particolari

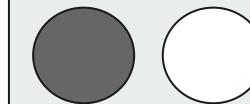
**TENTATIVO**



**PASSWORD**



**SUGGERIMENTI**



Una!

---

# L'algoritmo aiuta in casi particolari

**TENTATIVO**



**PASSWORD**



**SUGGERIMENTI**



Una o due pedine bianche?

---

# L'algoritmo aiuta in casi particolari

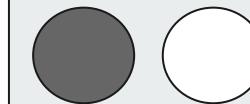
**TENTATIVO**



**PASSWORD**



**SUGGERIMENTI**



Sempre una!

---

# Possibili suggerimenti = 14

Table 1: The 14 possible grades in  $\text{MM}(4,7)$

b \ w	0	1	2	3	4
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	x
2	(2,0)	(2,1)	(2,2)	x	x
3	(3,0)	x	x	x	x
4	(4,0)	x	x	x	x

---

Paese che vai,  
*Mastermind* che trovi



# Esistono molteplici varianti

## BULLS & COWS

Dominio	len(psw)	Colori ripetuti	Tentativi max	Dominio(psw)
10	4	no	-	10.000

## SUPER MASTERMIND

Dominio	len(psw)	Colori ripetuti	Tentativi max	Dominio(psw)
8	5	si	12	32.768

---

# La nostra versione

## CLASSIC MASTERMIND

Dominio	len(psw)	Colori ripetuti	Tentativi max	Dominio(psw)
6 (A,B,C,D,E,F)	4 (ABDF)	si (ABBA)	10	1296 (AAAA-FFFF)

colori → lettere: A - B - C - D - E - F



---

# La matematica e Mastermind



# Un pò di numeri

Memo:

- 6 possibili lettere: [A → F]
- Password lunghe 4 (e.g. CFBD)
- Ammesse ripetizioni (e.g. AECE)

- **1.296** = numero di possibili password
- **1,3 e<sup>31</sup>** = numero di possibili match
- **1** = tentativo minimo
- **10** = tentativi massimi ammessi
- **5** = tentativi massimi *del miglior algoritmo*

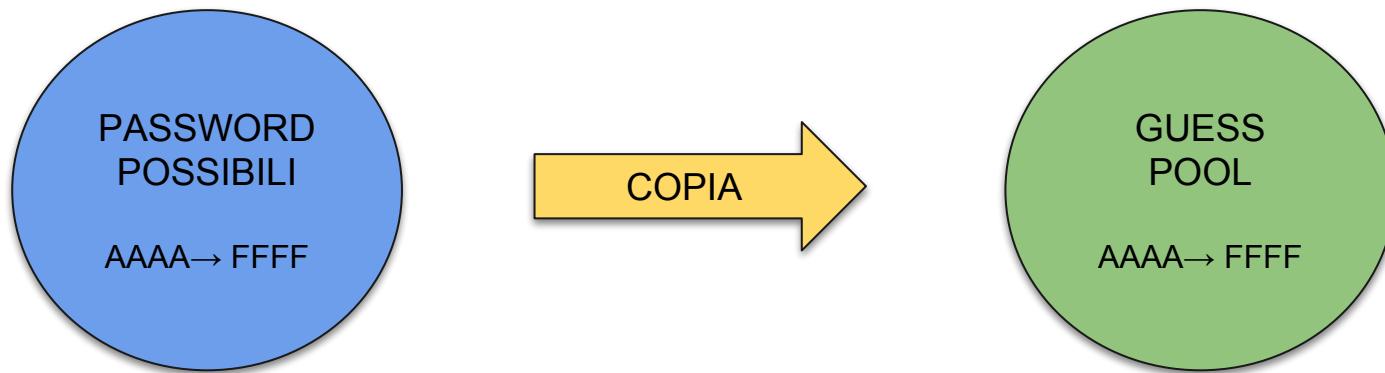
---

# Un semplice algoritmo: *Hopeful*

- Molto veloce.
  - ≈ 12 secondi per eseguire 1296 partite
- Media dei tentativi  
[ $N$  tentativi → % match vinti con  $N$  tentativi]
  - 4 → 32%
  - 5 → 44%
  - 6 → 13%
- Scritto da me, implementa una strategia *cut & hope...*

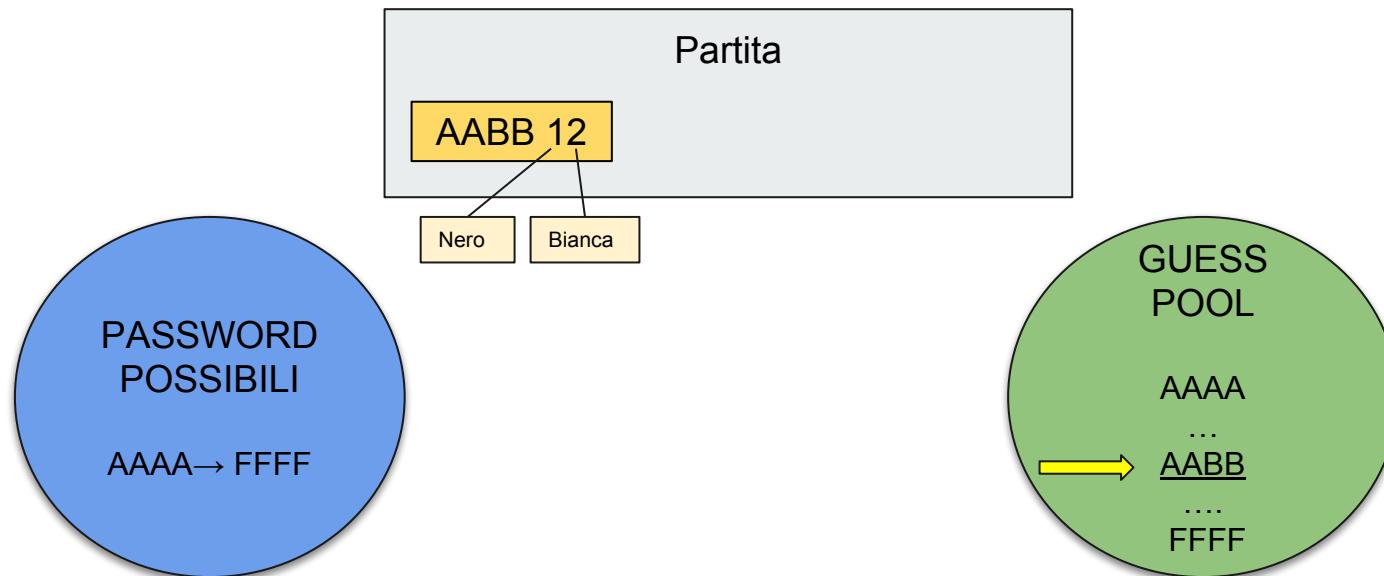
---

## Strategia *Hopeful*

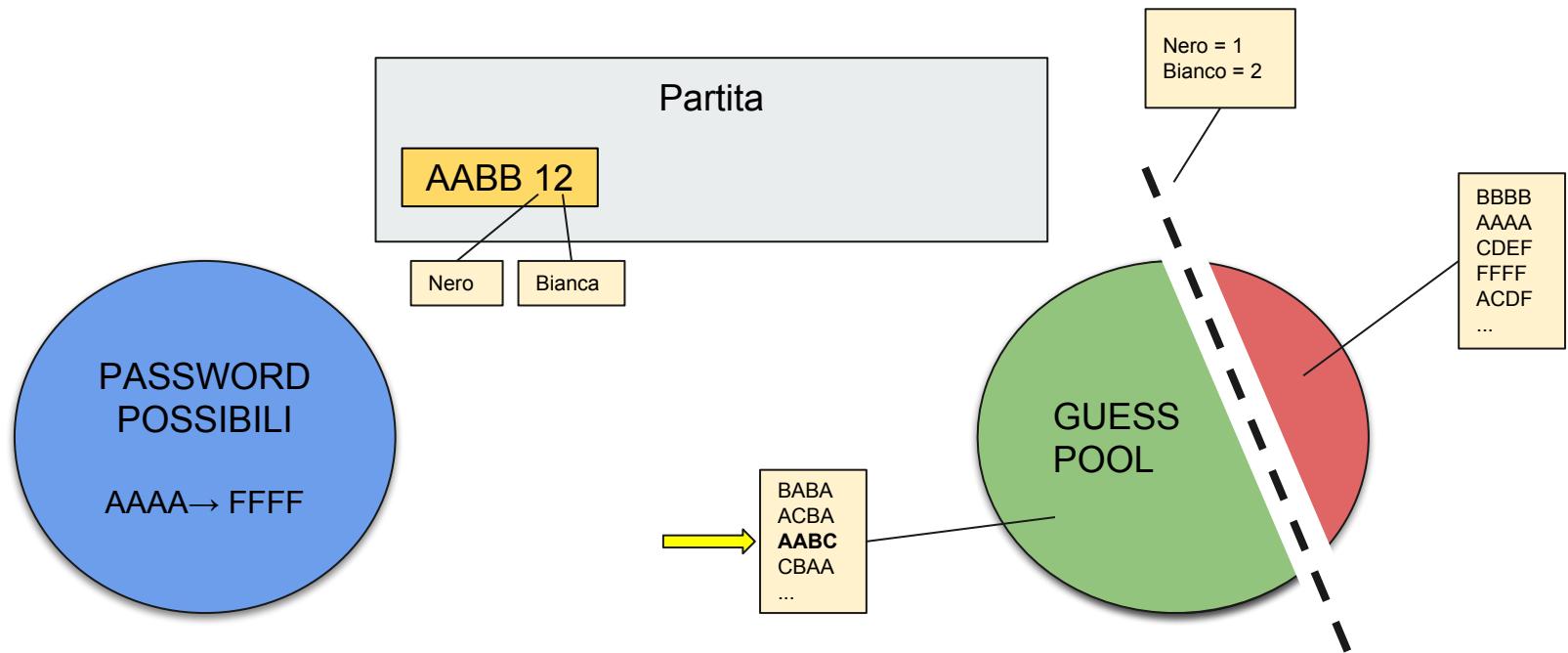


---

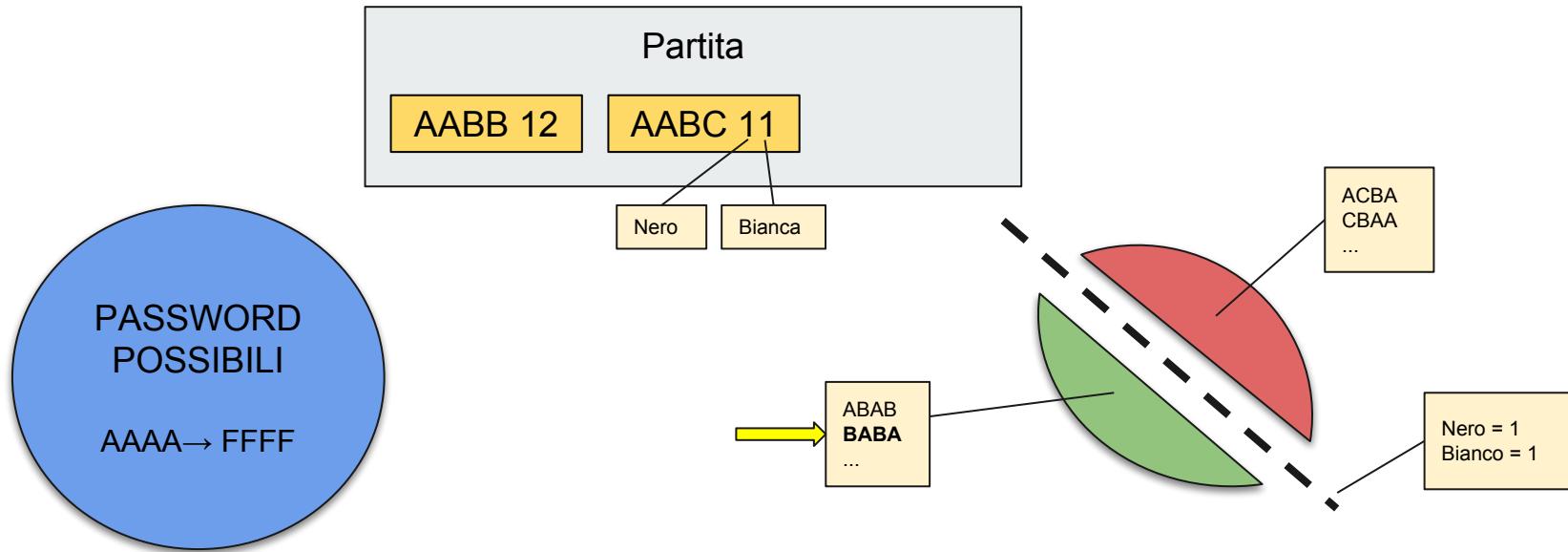
# Strategia *Hopeful*



# Strategia *Hopeful*

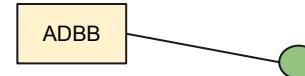
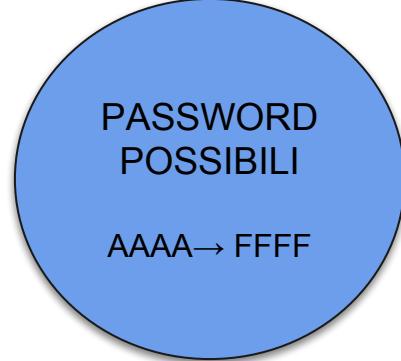


# Strategia *Hopeful*

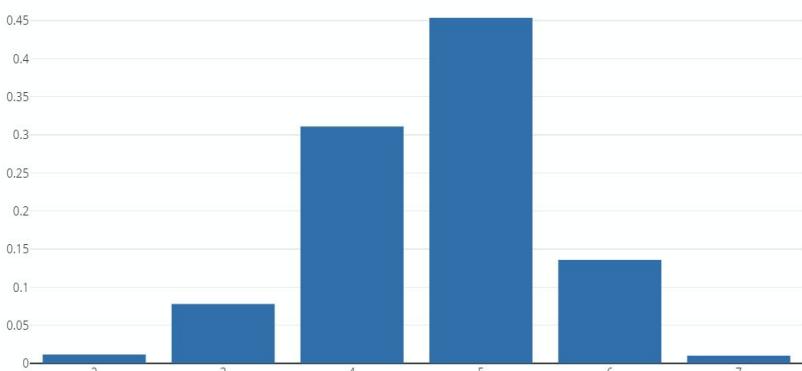


---

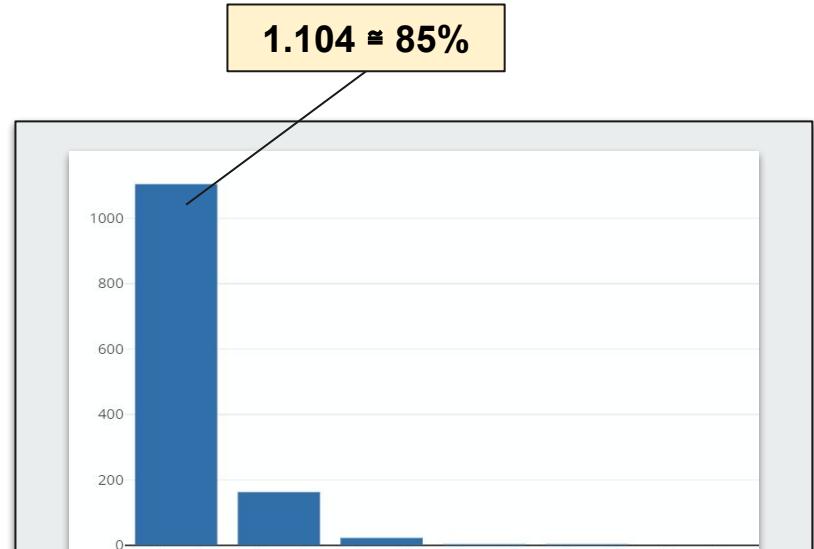
## Strategia *Hopeful*



# Percentuali di taglio



Lunghezza media di una partita



Password tagliate ad ogni tentativo

---

# Il miglior algoritmo: *Knuth*

- Proposto da *Donald Knuth* nel 1976 [[1](#)]
- Risolve una partita di *mastermind* in massimo 5 tentativi
- Non è il migliore per quanto riguarda *il numero medio* di tentativi.
  - *Numero medio di tentativi*: giocando N partite, con quanti tentativi in media hai vinto?
- Esistono algoritmi risolvono una partita con un numero medio di tentativi migliore.
  - Tuttavia potrebbero utilizzare anche 6 tentativi

---

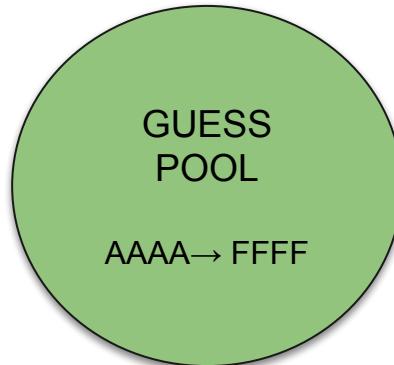
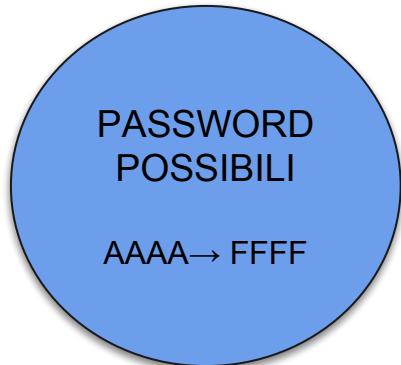
# Strategia Knuth



---

# Strategia Knuth

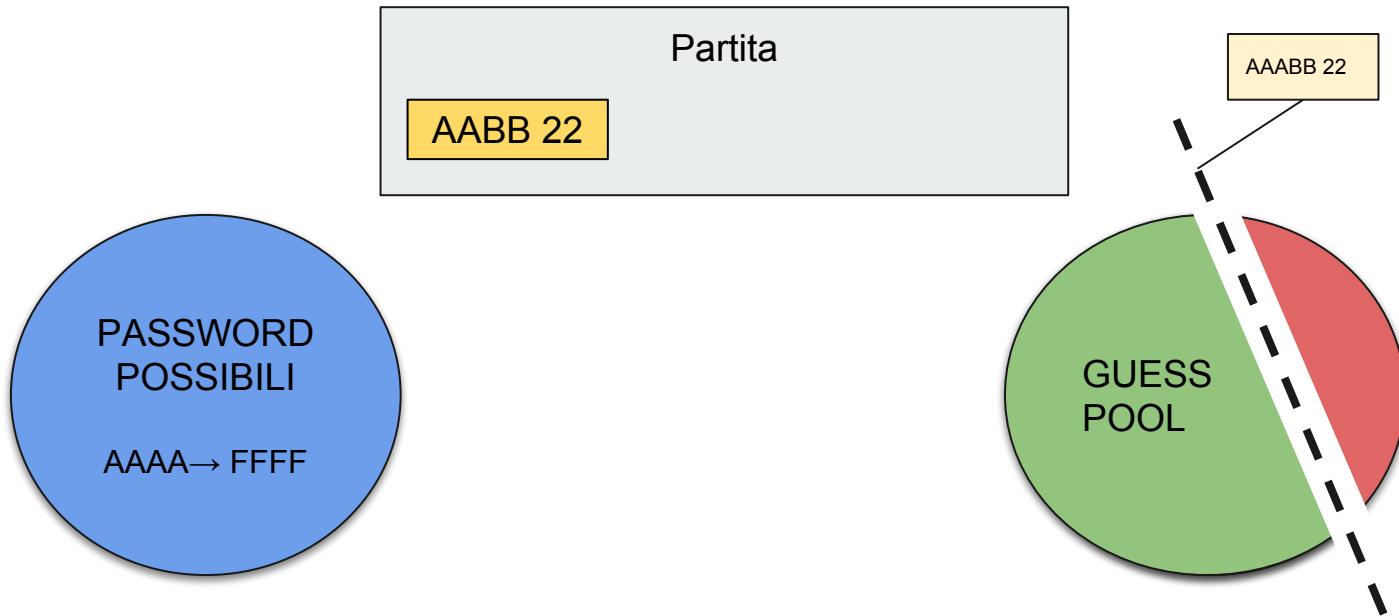
“Come primo tentativo usa **AABB**”



---

# Strategia Knuth

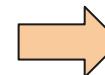
“Scegli il migliore  
tentativo da  
**Password Possibili”**



---

# Strategia Knuth

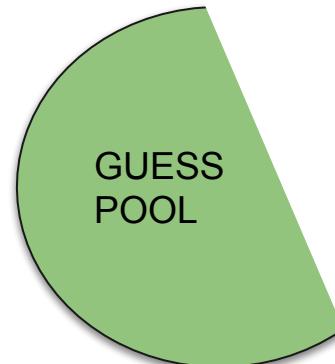
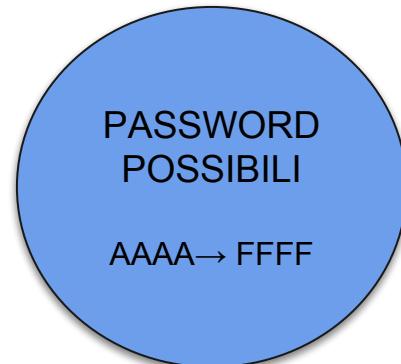
“Scegli il migliore tentativo da Password Possibili”



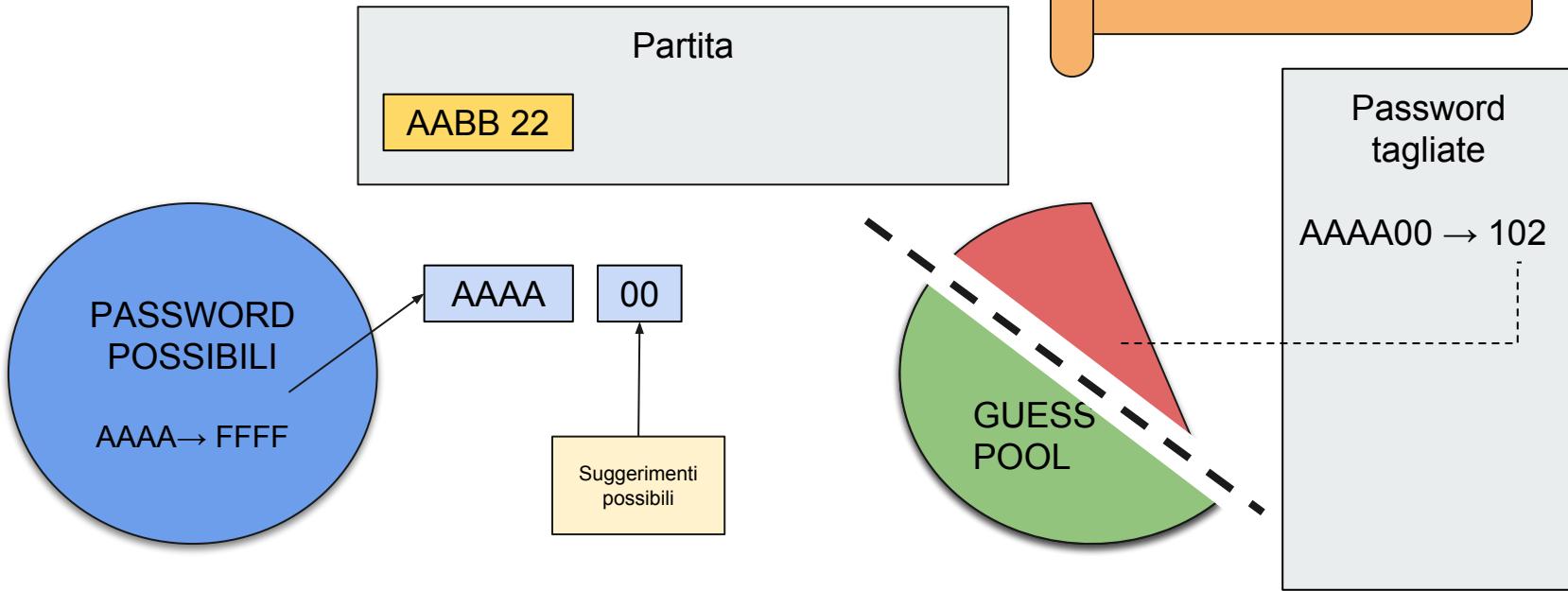
“Prendi il tentativo tra psw possibili che ti da il minimo taglio più grande”

Partita

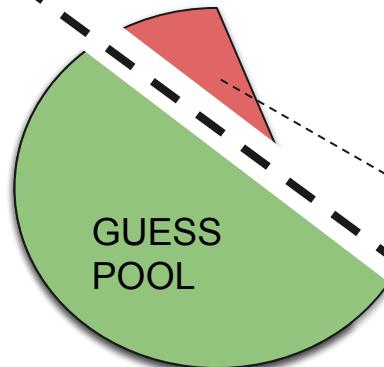
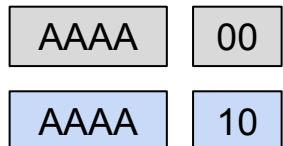
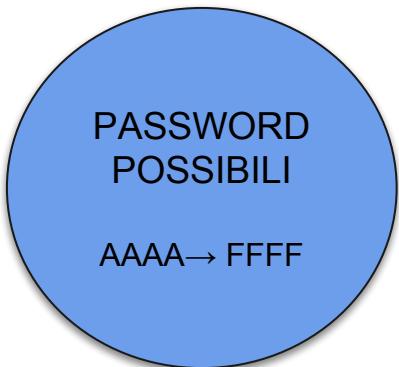
AABB 22



# Strategia Knuth



# Strategia Knuth

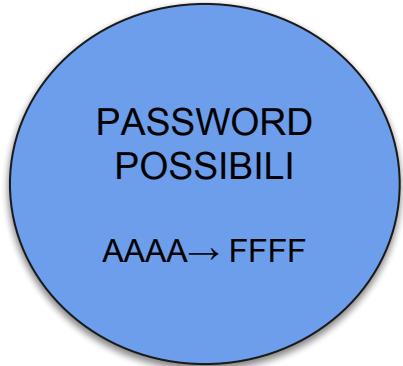


Password tagliate

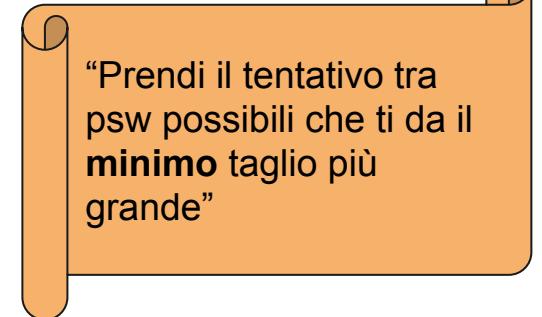
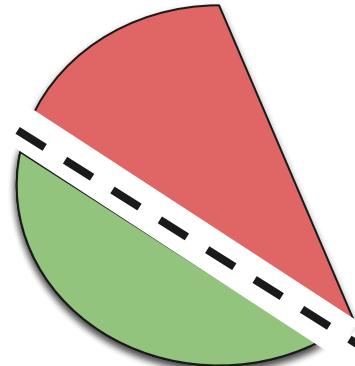
AAAA00 → 102  
AAAA10 → 56

“Prendi il tentativo tra psw possibili che ti da il minimo taglio più grande”

# Strategia Knuth



AAAA	00
AAAA	10
AAAA	...
AAAA	04



Password tagliate
AAAA00 → 102
AAAA10 → <u>56</u>
AAAA20 → 284
... → ...
AAAA03 → 501

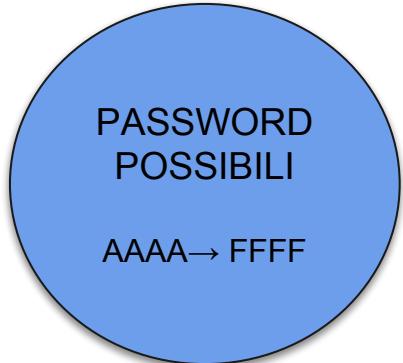
Storico
AAAA → 56

---

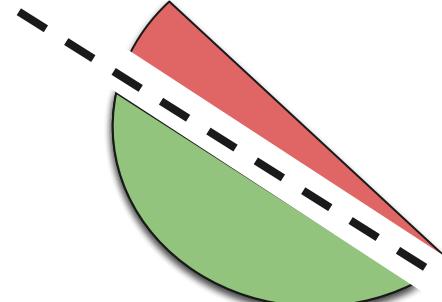
Finito con “AAAA”?

Iniziamo con “AAAB”

# Strategia Knuth



AAAB	00
AAAB	10
AAAB	...
AAAB	04



“Prendi il tentativo tra psw possibili che ti da il **minimo** taglio più grande”

Password tagliate

AAAB00 → **120**  
AAAB10 → 258  
...  
AAAB22 → 500  
AAAB04 → 326

Storico

AAAA → 56  
AAAB → **120**

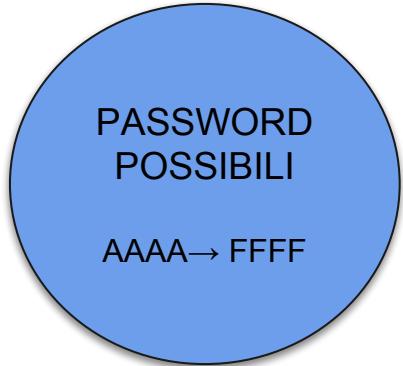
---

Finito con “AAAB” ?

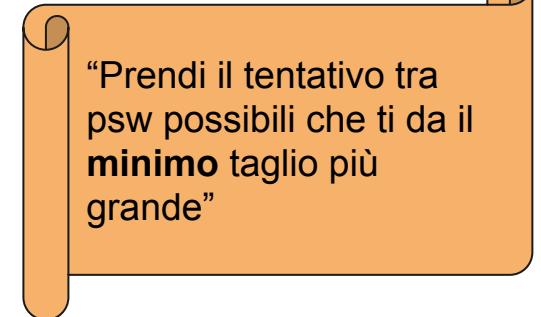
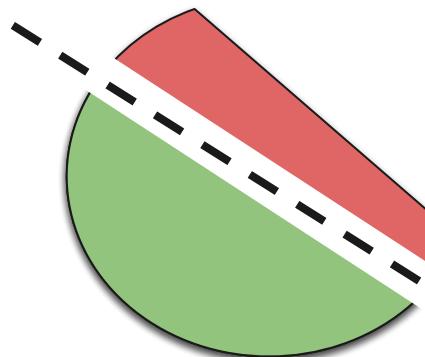
...

Passiamo direttamente a “FFFF”!

# Strategia Knuth



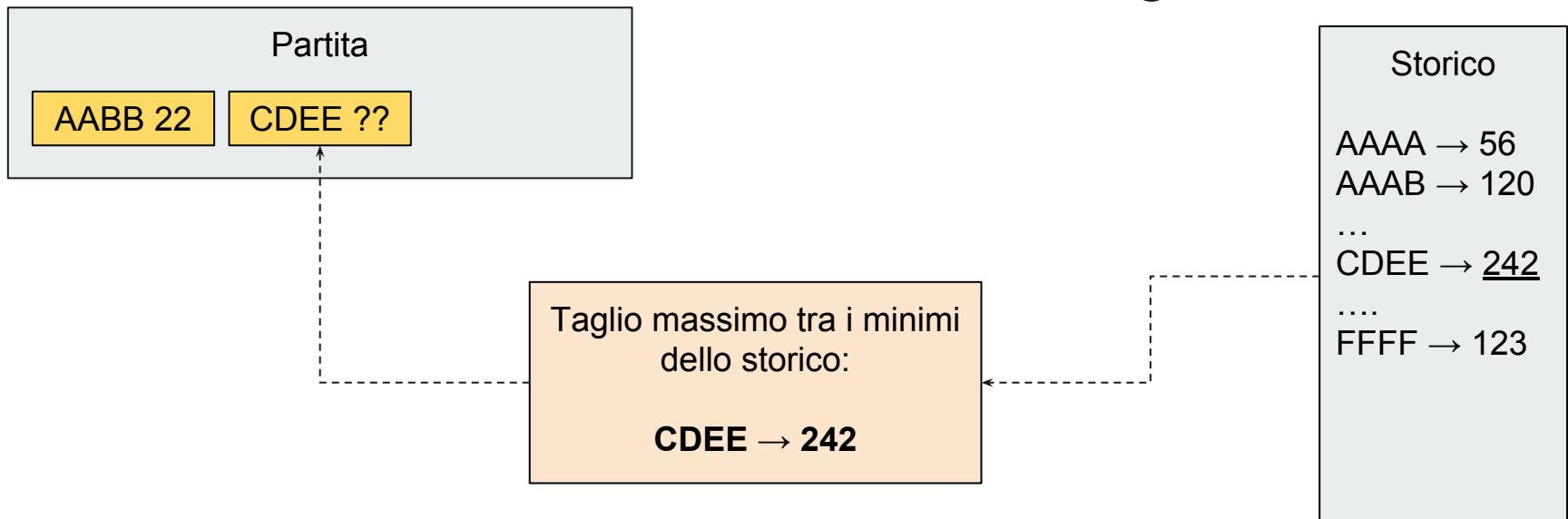
FFFF	00
FFFF	10
FFFF	...
FFFF	04



Password tagliate
FFFF00 → 800
FFFF10 → 150
...
FFFF22 → 300
FFFF04 → <u>123</u>

Storico
AAAA → 56
AAAB → 120
...
FFFF → <u>123</u>

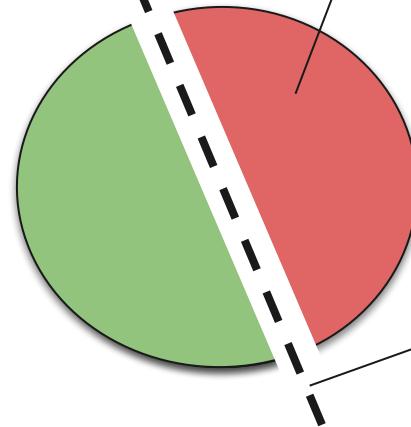
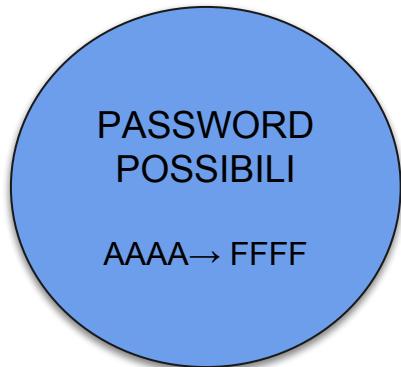
# Strategia Knuth



“Prendi il tentativo tra psw possibili che ti da il minimo taglio più grande”

---

# Strategia Knuth



Password tagliate effettivamente: **690**  
Peggior taglio calcolato: **242**



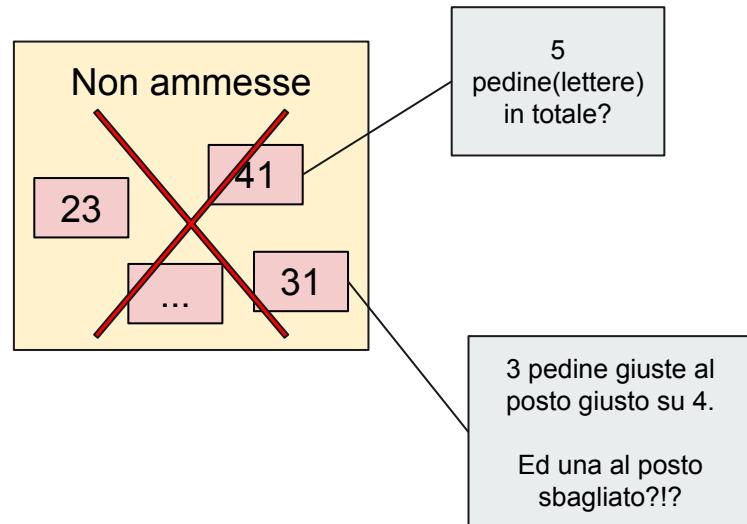
---

# Alcune precisazioni

---

# “Tutti i suggerimenti possibili”

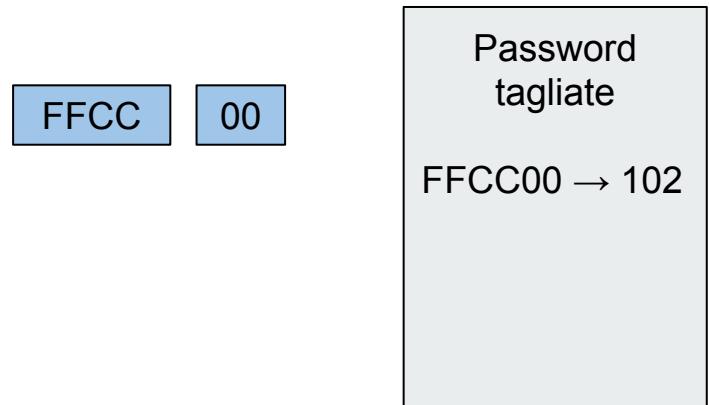
- Bisogna, per ogni password possibile, provare tutte le combinazioni *possibili* dei suggerimenti.
- Scartare quelle *non ammesse*.



---

# “Tutti i suggerimenti possibili”

- Bisogna, per ogni password possibile, provare tutte le combinazioni possibili dei suggerimenti.
- Scartare quelle *non ammesse*.
- Scartare quelle **incompatibili con la partita in corso**.



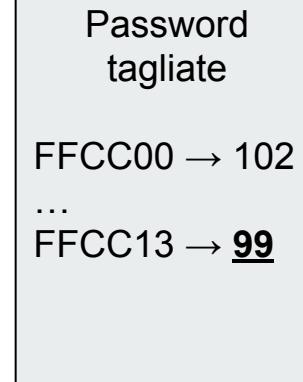
---

# “Tutti i suggerimenti possibili”

- Bisogna, per ogni password possibile, provare tutte le combinazioni possibili dei suggerimenti.
- Scartare quelle *non ammesse*.
- Scartare quelle **incompatibili con la partita in corso**.

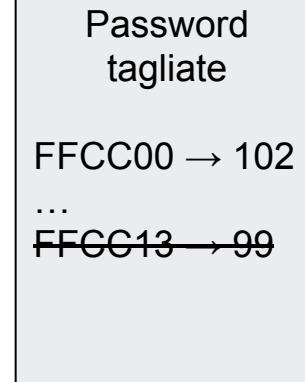
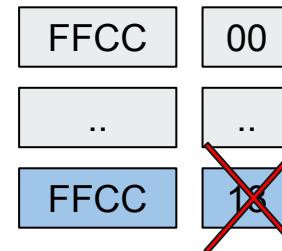
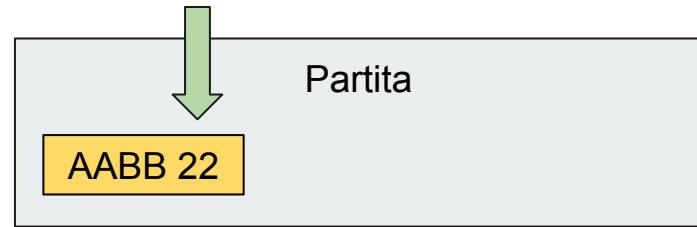


FFCC	00
..	..
FFCC	13



# “Tutti i suggerimenti possibili”

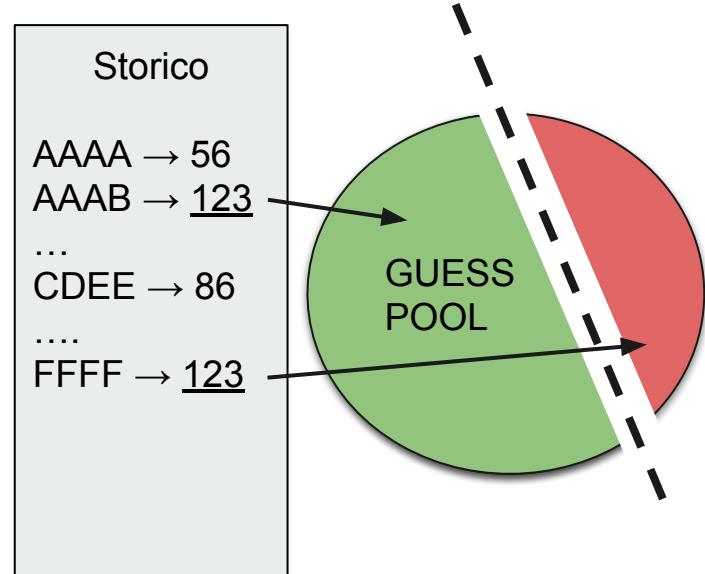
- Bisogna, per ogni password possibile, provare tutte le combinazioni possibili dei suggerimenti.
- Scartare quelle *non ammesse*.
- Scartare quelle **incompatibili con la partita** in corso.



---

## “A parità di taglio, scegli *guess pool*”

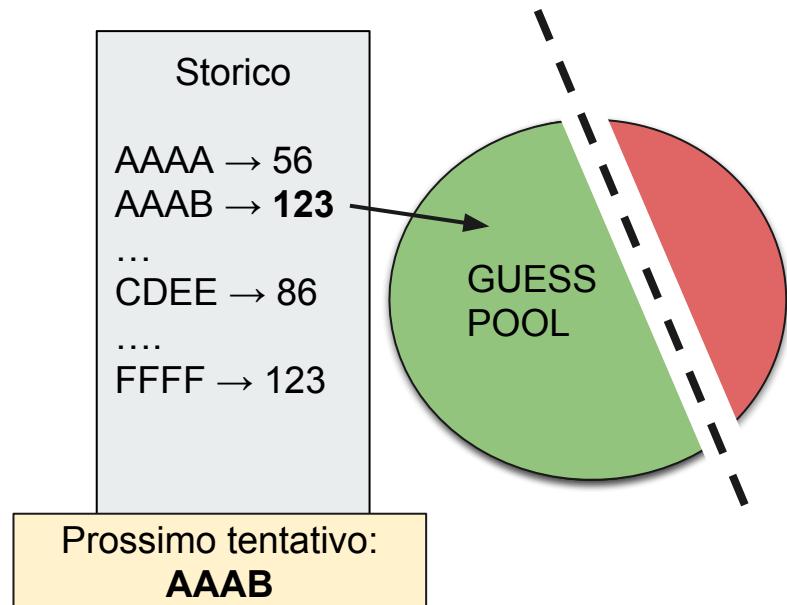
- Se due password *tagliano* la stessa quantità da *guess pool*:
  - Scegli, se presente, un che è all'interno di *guess pool*
  - Altrimenti è prendine una random



---

## “A parità di taglio, scegli *guess pool*”

- Se due password *tagliano* la stessa quantità da guess pool:
  - Scegli, se presente, un che è all'interno di *guess pool*
  - Altrimenti è prendine una random



---

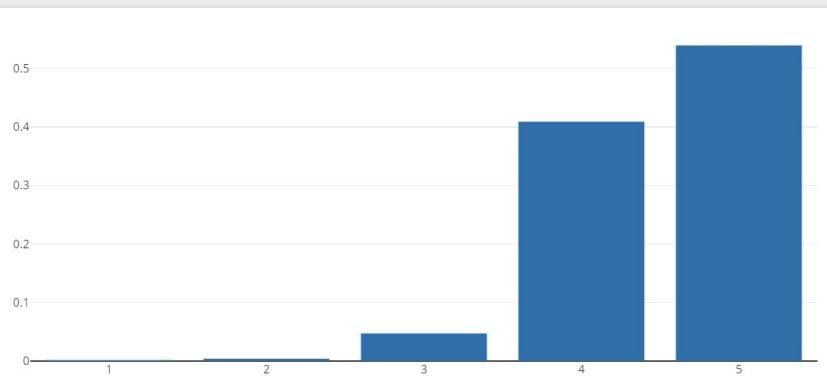
# Prestazioni algoritmo *Knuth*



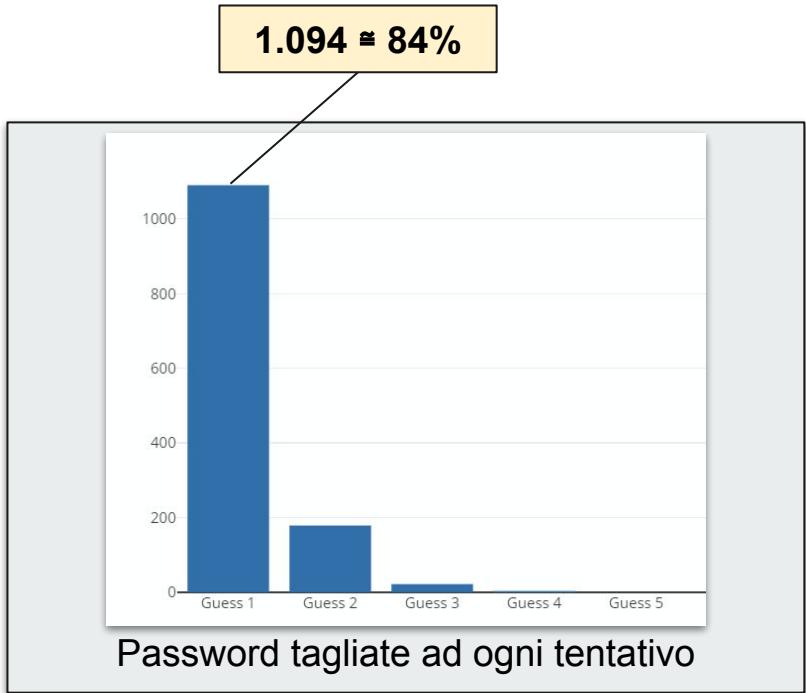
# Tentativi medi per risolvere una partita

Numero tentativi	Numero password	Numero password / 1296
1	1	$\approx 0$
2	6	$\approx 0.003$
3	25	0,047
4	239	<b>0,40</b>
5	1025	<b>0,53</b>

# Percentuali di taglio

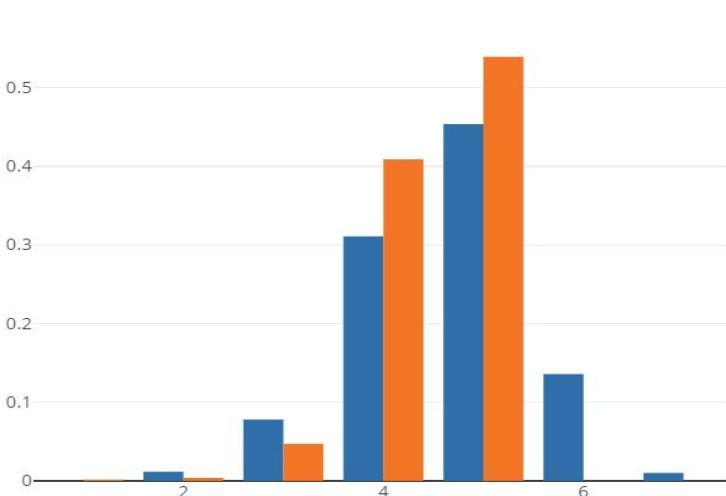


Lunghezza media di una partita

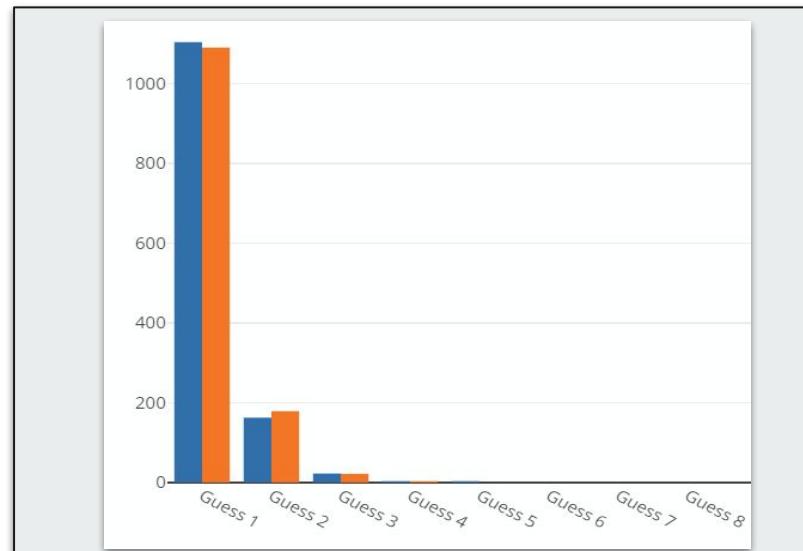


# Percentuali di taglio - Confronto

Arancio → Knuth  
Blu → Hopeful



Lunghezza media di una partita



Password tagliate ad ogni tentativo

---

# Tempo di esecuzione...

- Tempo totale: **≈5 giorni**

- Per giocare 1296 partite
    - Goal: AAAA → FFFF
  - Utilizzato il server di Galassi

- Processo non parallelizzabile
- Questo è un problema...

---

**Stiamo per mostrare del *codice***

```
1 def knuth_guess(psw_pool, hints_pool):
2     if len(psw_pool) == 1:
3         next_guess = psw_pool.pop()
4         return next_guess
5     else:
6         psw_pool_complete = GENERA TUTTE LE PASSWORD POSSIBILI
7         random.shuffle(psw_pool_complete)
8
9         max_cut_found = -1
10        ➔ for possible_next_guess in psw_pool_complete:
11            min_cut_found = None
12
13        ➔ for psw_survived in psw_pool:
14            cut_counter = 0
15            possible_hint = hints_calculator(psw_survived, possible_next_guess)
16
17        ➔ for psw_to_cut in psw_pool:
18            if hints_calculator(psw_to_cut, possible_next_guess) != possible_hint:
19                cut_counter += 1
20
21            if min_cut_found is None or cut_counter < min_cut_found:
22                min_cut_found = cut_counter
23
24            if (min_cut_found > max_cut_found) or ((min_cut_found == max_cut_found) and possible_next_guess in psw_pool):
25                max_cut_found = min_cut_found
26                next_guess = possible_next_guess
27
28    return next_guess
```

1296 cicli

Dipende  
dal turno

Dipende  
dal turno

```

1 def knuth_guess(psw_pool, hints_pool):
2     if len(psw_pool) == 1:
3         next_guess = psw_pool.pop()
4         return next_guess
5     else:
6         psw_pool_complete = GENERA TUTTE LE PASSWORD POSSIBILI
7         random.shuffle(psw_pool_complete)
8
9         max_cut_found = -1
10        ➔ for possible_next_guess in psw_pool_complete:
11            min_cut_found = None
12
13        ➔ for psw_survived in psw_pool:
14            cut_counter = 0
15            possible_hint = hints_calculator(psw_survived, possible_next_guess)
16
17        ➔ for psw_to_cut in psw_pool:
18            if hints_calculator(psw_to_cut, possible_next_guess) != possible_hint:
19                cut_counter += 1
20
21            if min_cut_found is None or cut_counter < min_cut_found:
22                min_cut_found = cut_counter
23
24            if (min_cut_found > max_cut_found) or ((min_cut_found == max_cut_found) and possible_next_guess in psw_pool):
25                max_cut_found = min_cut_found
26                next_guess = possible_next_guess
27
28    return next_guess

```

1296 cicli

$\approx 300$

$\approx 300$

$1296 \times 300 \times 300 = \underline{116.640.000}$

---

**Bisogna velocizzare**

---

## Un algoritmo diverso: *Knuth Fast*

- Proposto da me
- Risolve una partita di *mastermind* con mediamente 5 tentativi
- Tempo di esecuzione: **≈ 10 ore**
  - Tempo calcolato sempre per 1 ciclo: 1296 partite

```

1 def knuth_guess(psw_pool, hints_pool):
2     if len(psw_pool) == 1:
3         next_guess = psw_pool.pop()
4         return next_guess
5     else:
6         psw_pool_complete = GENERA TUTTE LE PASSWORD POSSIBILI
7         random.shuffle(psw_pool_complete)
8
9         max_cut_found = -1
10        → for possible_next_guess in psw_pool_complete:
11            min_cut_found = None
12
13        X → for psw_survived in psw_pool:
14            cut_counter = 0
15            possible_hint = hints_calculator(psw_survived, possible_next_guess)
16
17        → for psw_to_cut in psw_pool:
18            if hints_calculator(psw_to_cut, possible_next_guess) != possible_hint:
19                cut_counter += 1
20
21            if min_cut_found is None or cut_counter < min_cut_found:
22                min_cut_found = cut_counter
23
24            if (min_cut_found > max_cut_found) or ((min_cut_found == max_cut_found) and (min_cut_found < max_cut_found)):
25                max_cut_found = min_cut_found
26                next_guess = possible_next_guess
27
28    return next_guess

```

Utilizziamo tutte le combinazioni possibili (**14**) dei suggerimenti

1296 cicli

$\approx 300$

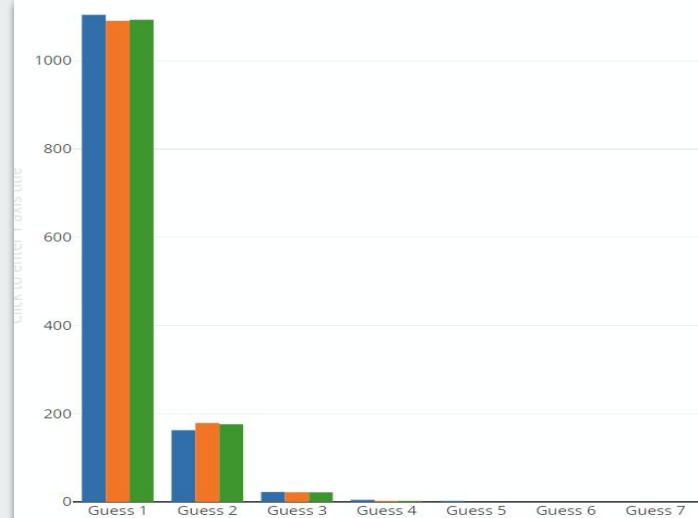
$\approx 300$

$$1296 \times 14 \times 300 = \underline{\underline{5.443.200}}$$

$$116.640.000 - 5.443.200 = \underline{\underline{111.196.800}}$$

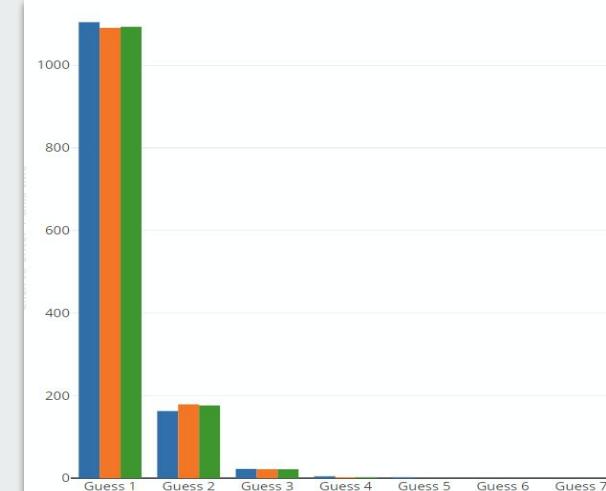


## Percentuali di taglio - Confronto



Lunghezza media di una partita

Arancio → Knuth  
Blu → Hopeful  
Verde → Knuth Fast



Password tagliate ad ogni tentativo



# Altre info

- Database usato per il training e l'allenamento delle NN:
  - 1.300.000 x 2 Hopeful (**Train** ed **Eval**)
  - 1.296 Knuth come **Test**
- Risultati ML
  - Rete singola Bi-LSTM AS
    - **29% Accuracy**
    - 1.9 Avg Peg matched
    - 0.85 White peg on psw
    - 1g e 15h di train
  - Rete BiLSTM AS goal scomposto
    - **Singola: 70% accuracy**
    - 4 serie: 18% accuracy
    - 3h 30 l'una di train (14h circa)

# Seconda parte: *Machine Learning*

---



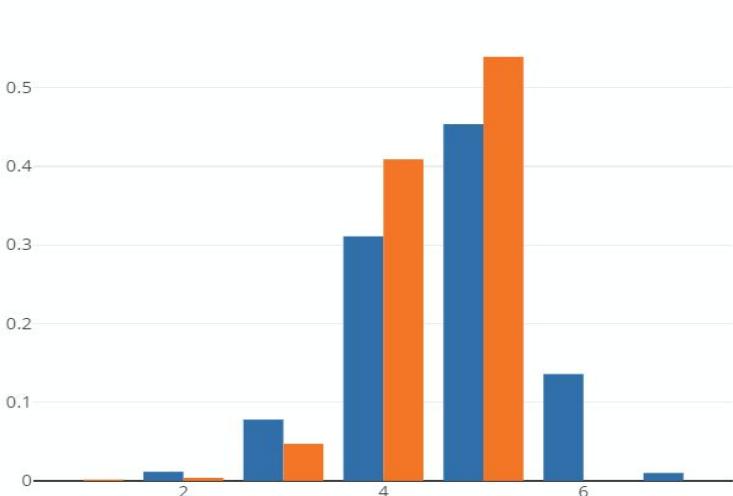
# Vediamo prima i dataset che useremo

- Info
  - Knuth risolve in massimo 5 tentativi qualsiasi partita
  - Percentuale di divisione Train/Eval = 0.5

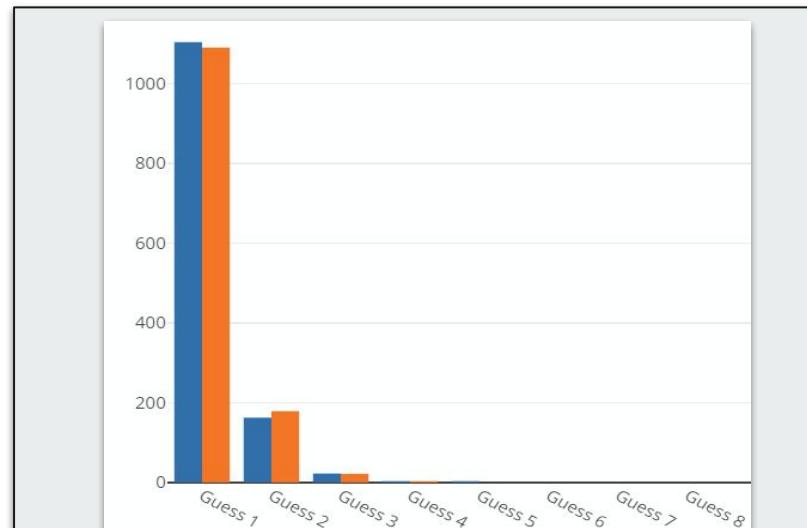
- *Hopeful Big*
  - Strategia *Hopeful*
  - Train + Eval ≈ 2M 600K partite
  - 24h con 4 CPU
    - ≈ 12 secondi per eseguire 1296 partite
- *Knuth optimal*
  - Strategia *Knuth* versione base
  - Test set = 1296 partite
    - [AAAA→FFFF]
  - ≈ 5 Giorni

# Percentuali di taglio - Confronto

Arancio → Knuth  
Blu → Hopeful



Lunghezza media di una partita



Password tagliate ad ogni tentativo

---

# Conversione dei dati

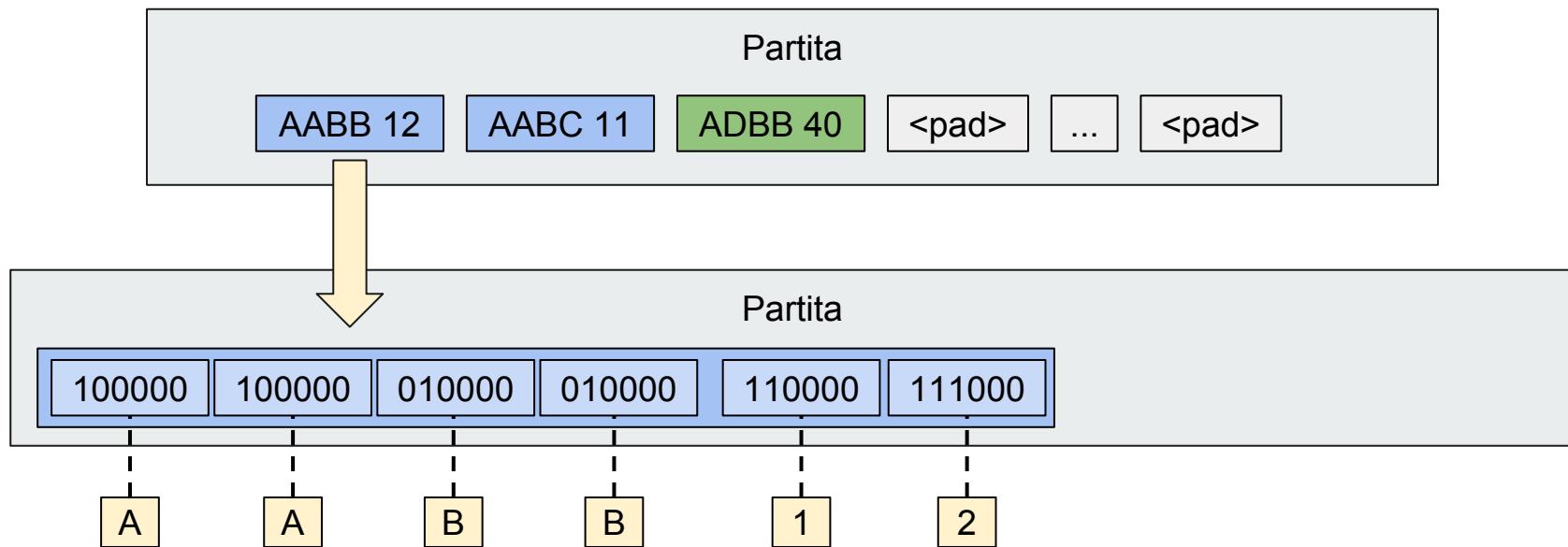
---

# Conversione dei dati per il M.L.

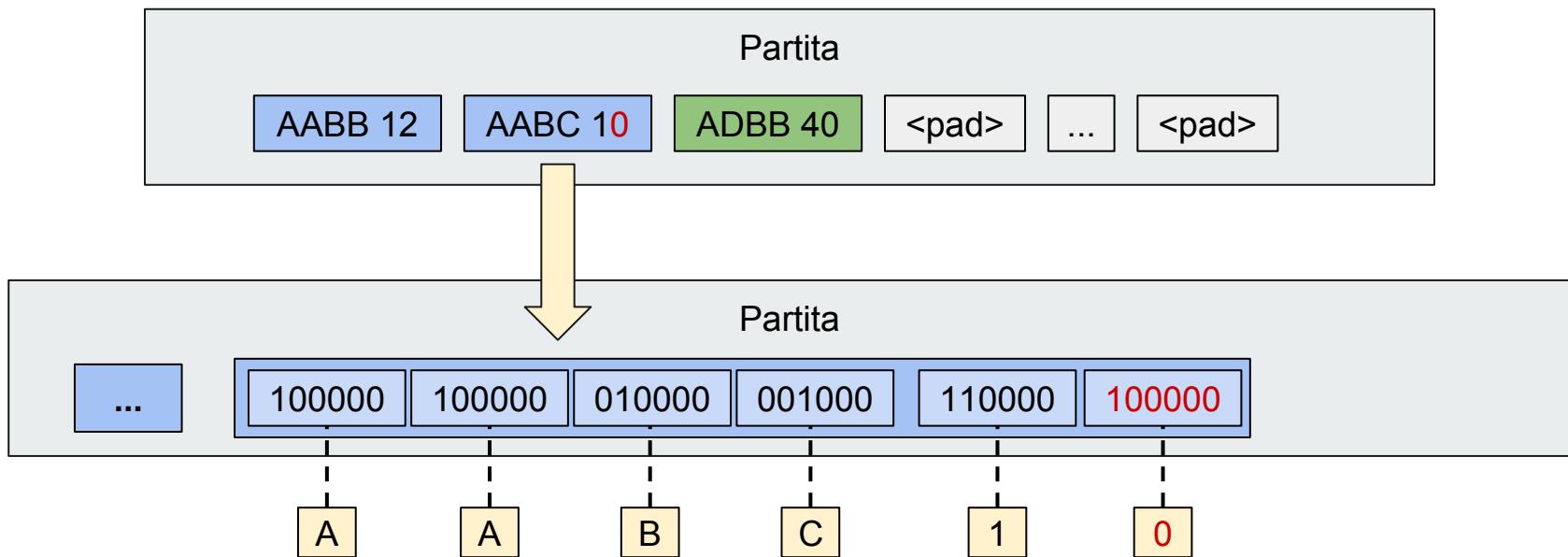
- Conversione lettere in *one-hot encoding*
  - $A \rightarrow 100000$
  - $B \rightarrow 010000$
  - $C \rightarrow 001000$
  - ...
- Eliminato l'ultimo tentativo da ogni partita
- Conversione suggerimenti in *simil-binary*
  - $0 \rightarrow 10000$
  - $1 \rightarrow 11000$
  - $2 \rightarrow 11100$
  - ...
- Conversione dei tentativi non usati
  - $\langle pad \rangle \rightarrow [0, 0, 0, \dots] \times 34$

---

## Esempio conversione



## Esempio conversione



---

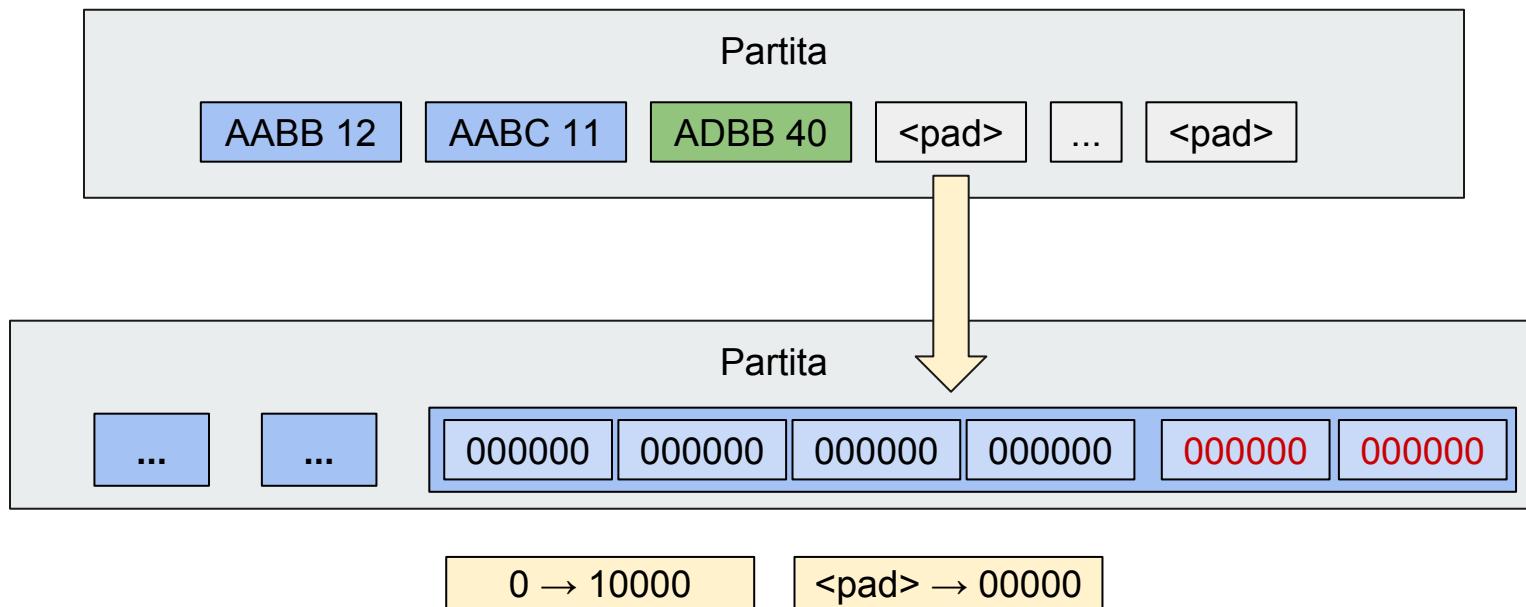
## Esempio conversione



Ultimo tentativo scartato in  
quanto contiene la  
password

---

# Esempio conversione

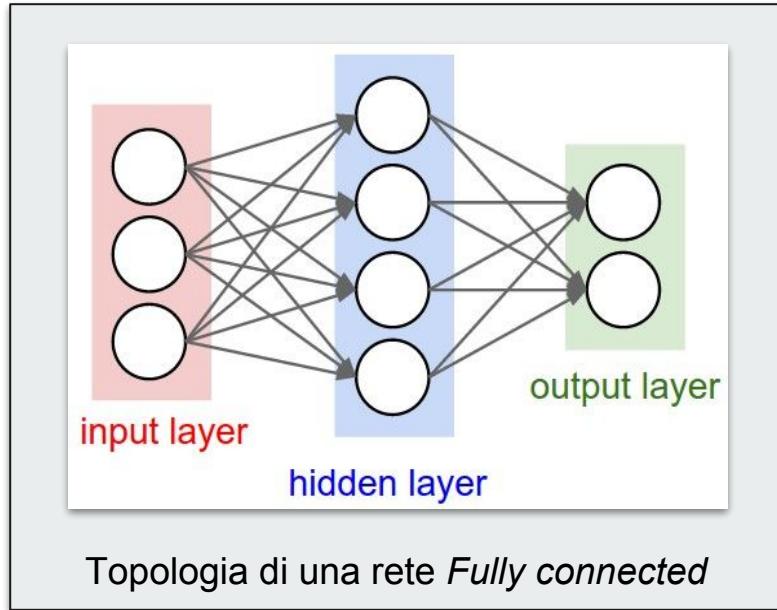


---

# Modelli di reti neurali

# Fully Connected

- Primo test “di prova”
  - Usate per testare l’infrastruttura
- Testate varie dimensioni (128, 256,..., 2048)
- Non si è mai superato il 12% di *accuracy of one* [1] all’interno dell’*Evaluation set*

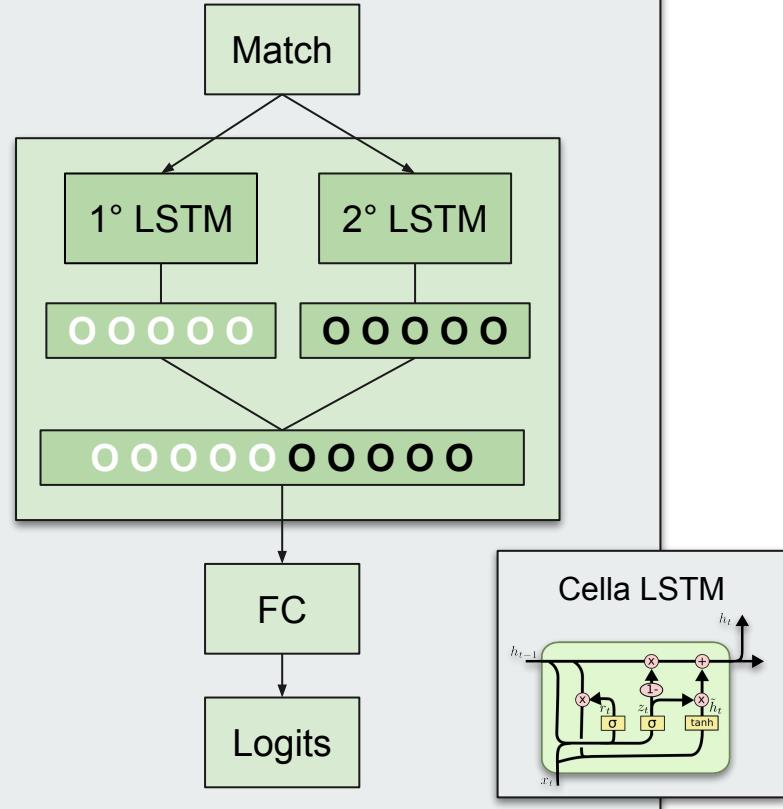


[1] Metrica creata da me per lo studio del numero di **black pegs** della predizione della NN

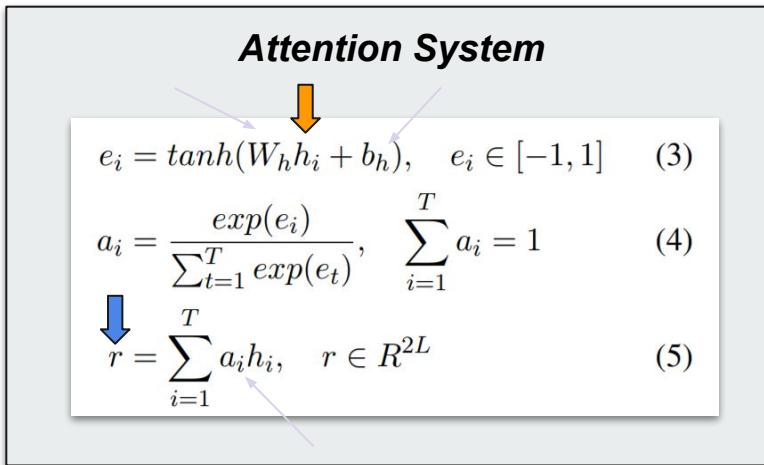
# Bi-LSTM

- Bi-LSTM:
  - Prima LSTM
    - legge il match dal primo all'ultimo tentativo
  - Seconda LSTM
    - legge il match dall'ultimo al primo tentativo
  - Si impilano i risultati delle due reti
- Fully connected aventi softmax come funzione di attivazione

Struttura di una *Bi-LSTM*

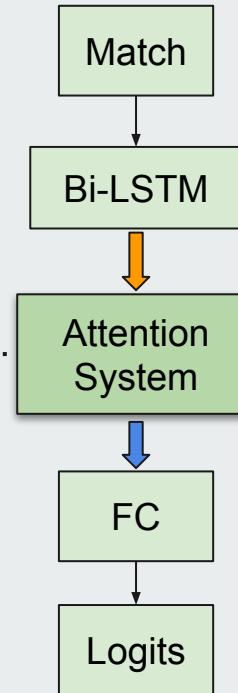


# Bi-LSTM con A.S.



Da ora in poi useremo solo questa

## Bi-LSTM con Attention System



---

# Modalità delle reti neurali

- Memo

- Tutte le *modalità* hanno all'interno una BiLSTM A.S.
- *Train hopeful* = 1M 300K
- *Eval hopeful* = 1M 300K

- **Semplice**

- *Input* → match
- *Predice* → psw
- Allentato su *train hopeful*

- **Ridotto**

- *Input* → match
- *Predice* → posizione N della psw
- Necessario allenare 4 reti
- Allentato su *train hopeful*

- **Ridotto “smart”**

- *Input* → match e N-1 posizioni della psw
- *Predice* → posizione N della psw
- Necessario allenare 4 reti
- Allentato su *train hopeful*

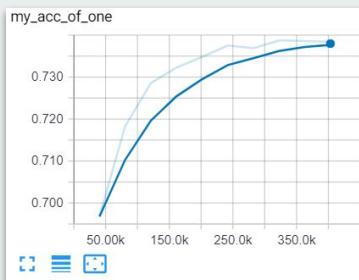
- **Ridotto “smart” Big train**

- Identico a ridotto smart
- Allenato con *train + eval hopeful* (1M 300K)

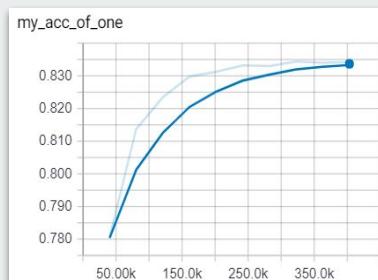
## Focus Ridotto “smart” - Eval set

**my\_acc\_of\_one** → Metrica interna, misura il numero di “1” corretti all’interno del *one hot encoding*

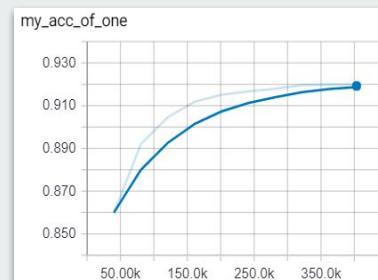
Prima rete



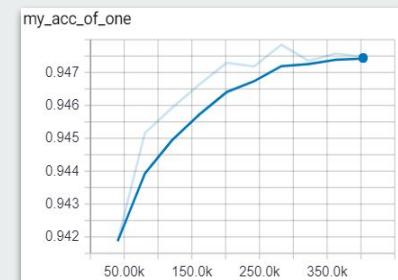
Seconda rete



Terza rete



Quarta rete



Notiamo come la rete più “parti” della password riceve, e maggiori probabilità ha di fare una predizione corretta



# **Modalità di valutazione**

- Memo

- Test set = **knuth optimal**
  - 1296 elementi
  - [AAAA → FFFF]

- **Non giocando**

- Fornita la partita del test set:
  - predire la password

- **Giocando**

- Fornita la partita del test set:
  - predire la password in meno di 10 tentativi
- Ogni tentativo errato viene aggiunto al match provvisto dei relativi suggerimenti

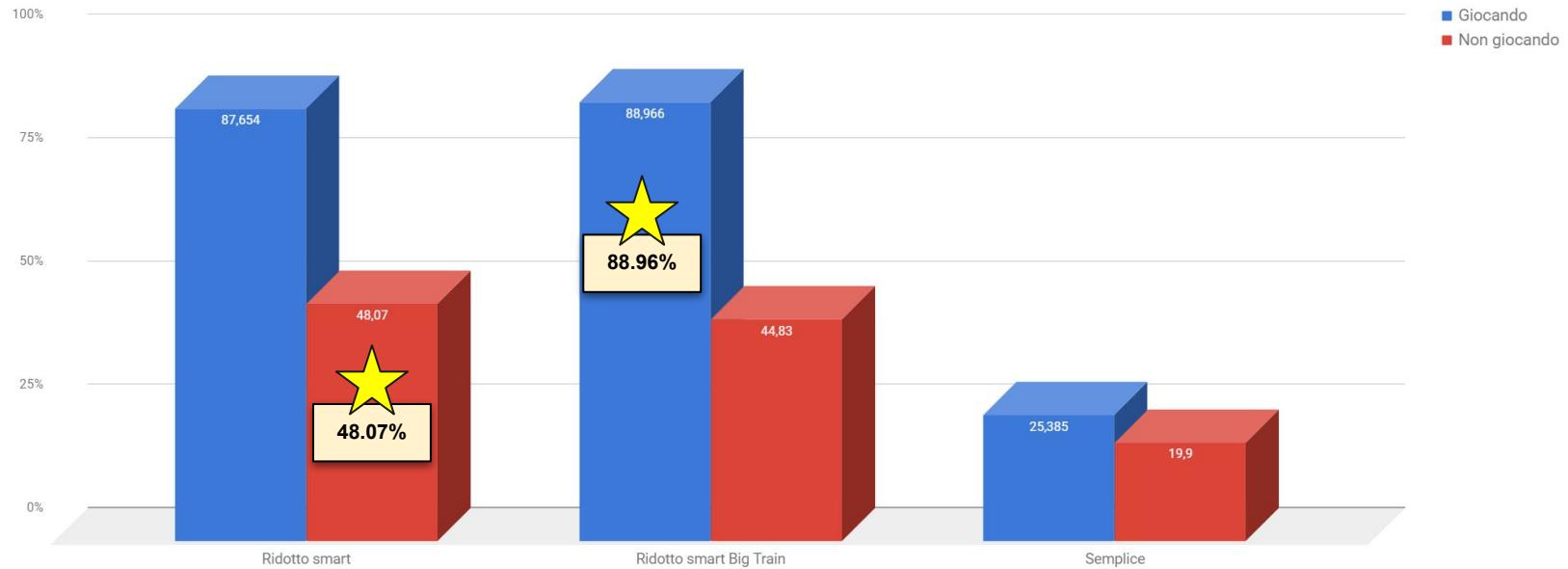
---

# Risultati ottenuti (\*)

(\*) Usato come *Test Set Knuth optimal*

## Percentuali di partite vinte dalle reti neurali

Giocate 1296 partite

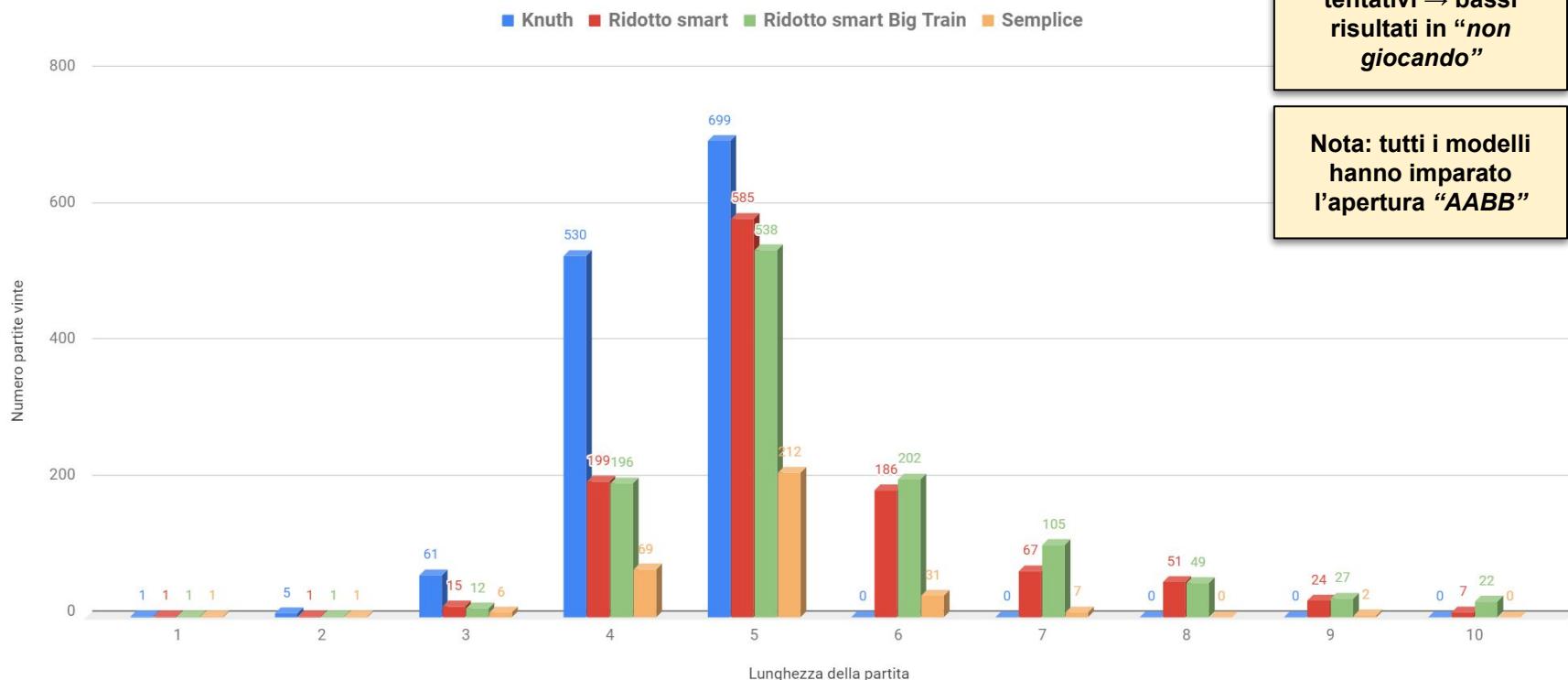


**Ridotto smart:**  
la rete  $N$  riceve match più  
 $N-1$  lettere della psw,  
predice la  $N$ -esima

**Semplice:**  
la rete riceve il match e  
predice la password

## Numero di tentativi delle vittorie su 1296 partite

Nota: ogni partita parte dalla storia di Knuth optimal e prosegue



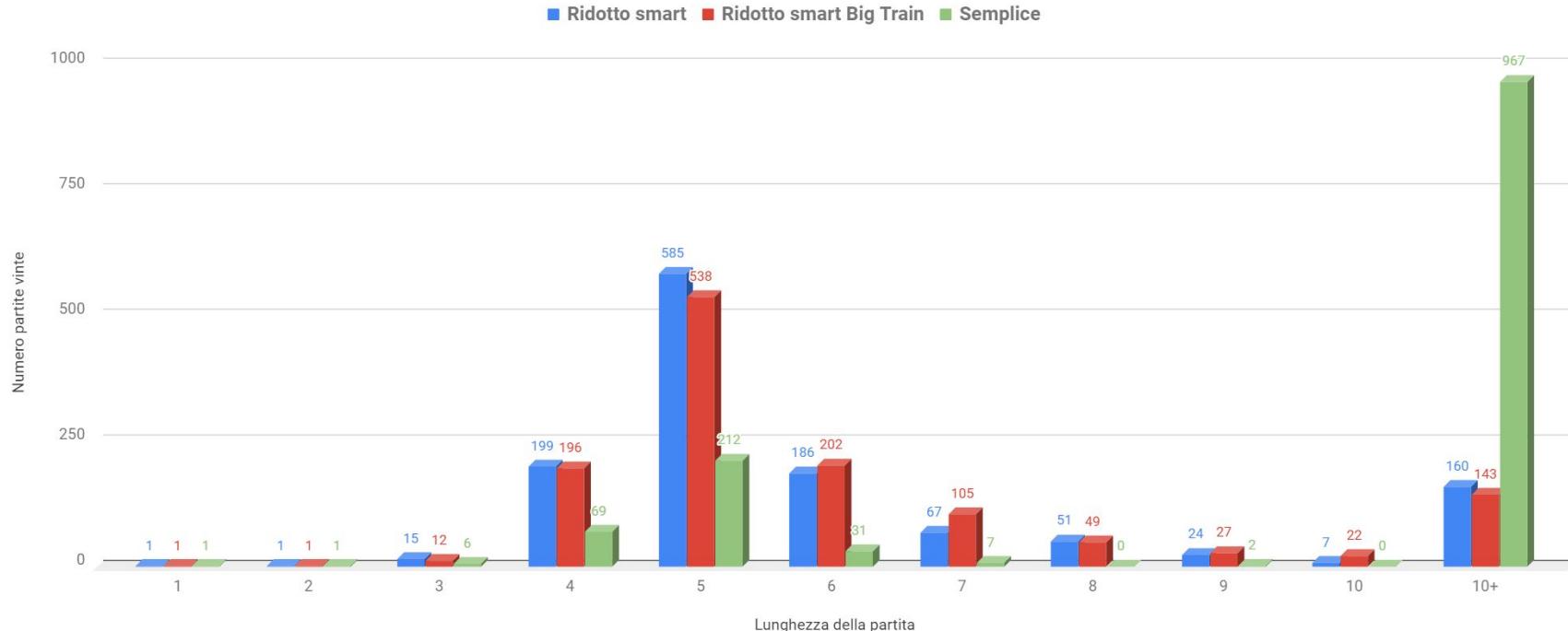
**Nota: Knuth a 4 tentativi → bassi risultati in “non giocando”**

**Nota: tutti i modelli hanno imparato l’apertura “AABB”**

## Numero di tentativi delle vittorie su 1296 partite

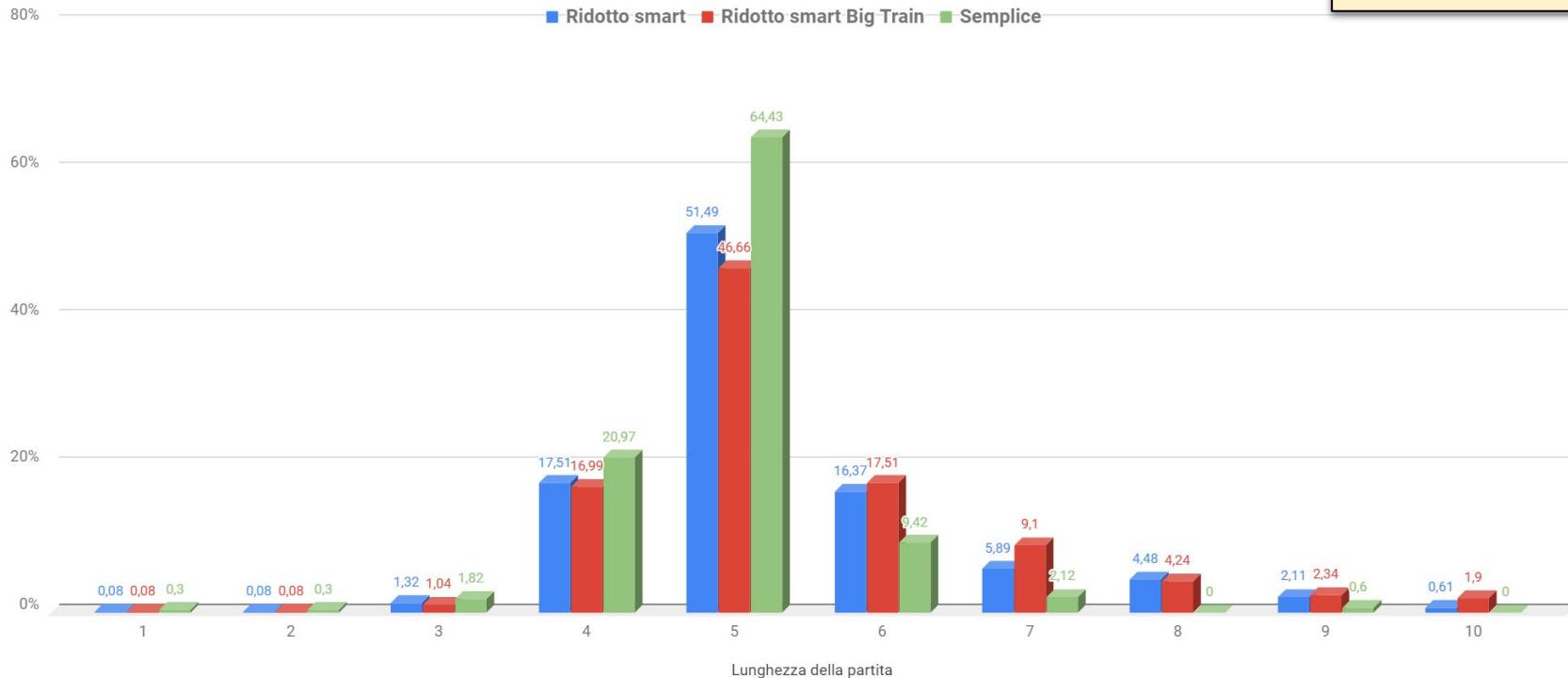
Nota: ogni partita parte dalla storia di Knuth optimal e prosegue

**Focus senza Knuth**



## Percentuale di tentativi usati all'interno delle vittorie

Nota: ogni partita parte dalla storia di Knuth



[!] La percentuale è calcolata sul **numero di vittorie**



# Tempistiche di allenamento

- Memo

- Perché strategia *hopeful*?
  - Knuth → 5g per 1296 match
  - Knuth fast → 2g per 30.000 match

- **Semplice**

- *Train set* → Hopeful 1M 300K
- *Eval set* → Hopeful 1M 300K (diversi)
- **Tempo** → 1 giorno e 15h
  - LSTM → 256; AS → 256
  - 100 epoche

- **Ridotto “smart”**

- *Train set* → Hopeful 1M 300K
- *Eval set* → Hopeful 1M 300K (diversi)
- **Tempo** → 16h (≈4h a rete)
  - LSTM → 256; AS → 256
  - 10 epoche

- **Ridotto “smart” Big train**

- *Train set* → Hopeful completo 2M 600K
- *Eval set* → /
- **Tempo** → 14h (≈3,5h a rete)
  - Stessi parametri ridotto smart



## Altre note

- Sviluppato in *TensorFlow* (python).
  - Tramite nuove strutture chiamate **Estimator**.
    - Supporto gcp
    - Gestione *train - eval - export* implementato e indipendente dalla rete
- Supporto a **Colab** degli script.
- [!] I modelli esportati sono *autosufficienti* nella conversione del match.
  - Input → dizionario
    - “Guess 1” → “ABCD03”
    - “Guess 10” → “<pad>”
    - “Guess N” → “~~AFCD40~~” → “<pad>”

---

# Grazie per l'attenzione

Domande?