Algoritmi I

Lezione introduttiva e motivazionale

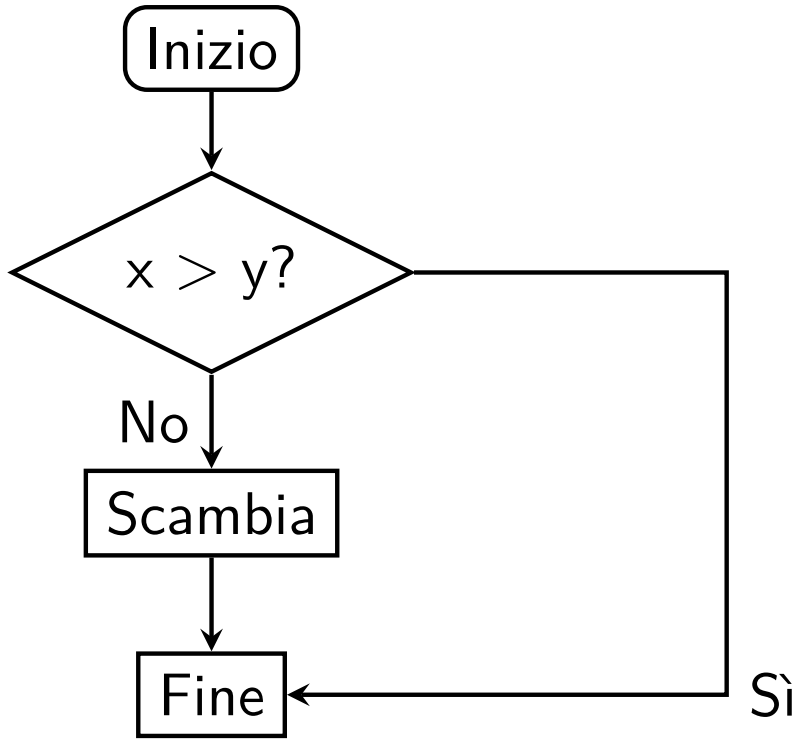
Angelo Monti

Perché studiare algoritmi?

- Google, Netflix, Amazon, Maps
- Compressione video, crittografia, intelligenza artificiale
- Gli algoritmi sono ovunque
- Non basta che un programma funzioni $\rightarrow$ deve funzionare **bene**

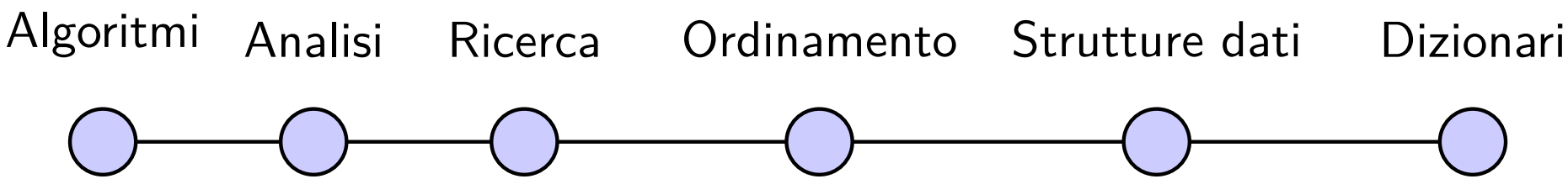
Cos'è un algoritmo?

- Sequenza finita di istruzioni
- Risolve un problema generale
- Deve essere: corretto, efficiente, comprensibile



Struttura del corso

- Analisi di algoritmi (notazione asintotica e costo)
- Algoritmi di ricerca
- Algoritmi di ordinamento
- Strutture dati fondamentali
- Dizionari e strutture avanzate (hash, AVL, B-alberi)



Analizzare gli algoritmi

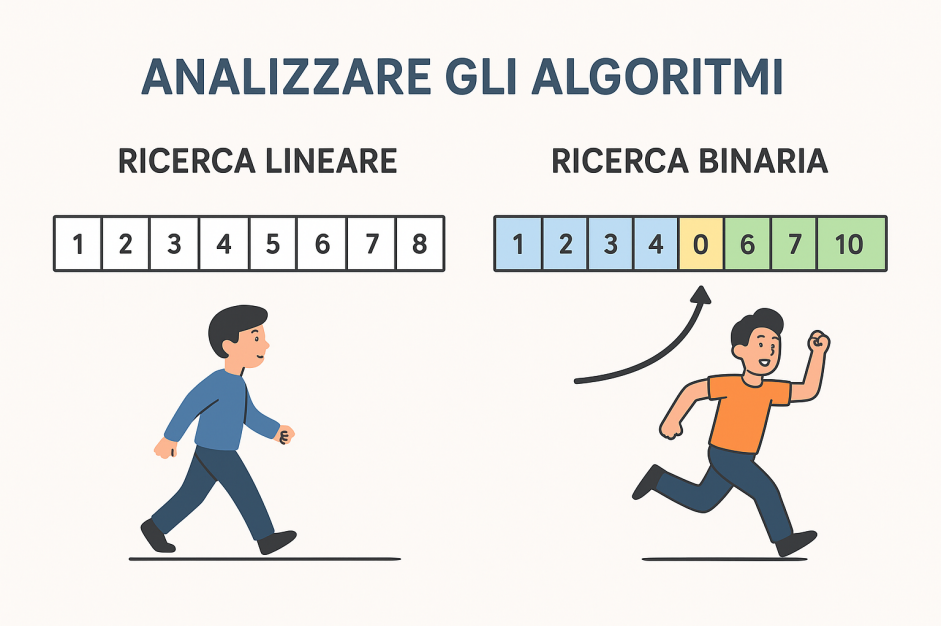
- Non basta il "funziona" $\rightarrow$ serve l'efficienza
- Notazione asintotica: O, Ω, Θ
- Iterazioni e ricorsione $\rightarrow$ costo computazionale

```
def funzione_iterativa():
    for i in range(10):
        print(i)
```

```
def funzione_ricorsiva(i=0):
    if i < 10:
        print(i)
        funzione_ricorsiva(i + 1)
```

Algoritmi di ricerca

- Esempio: ricerca lineare vs ricerca binaria
  - Ricerca sequenziale
  - Ricerca binaria



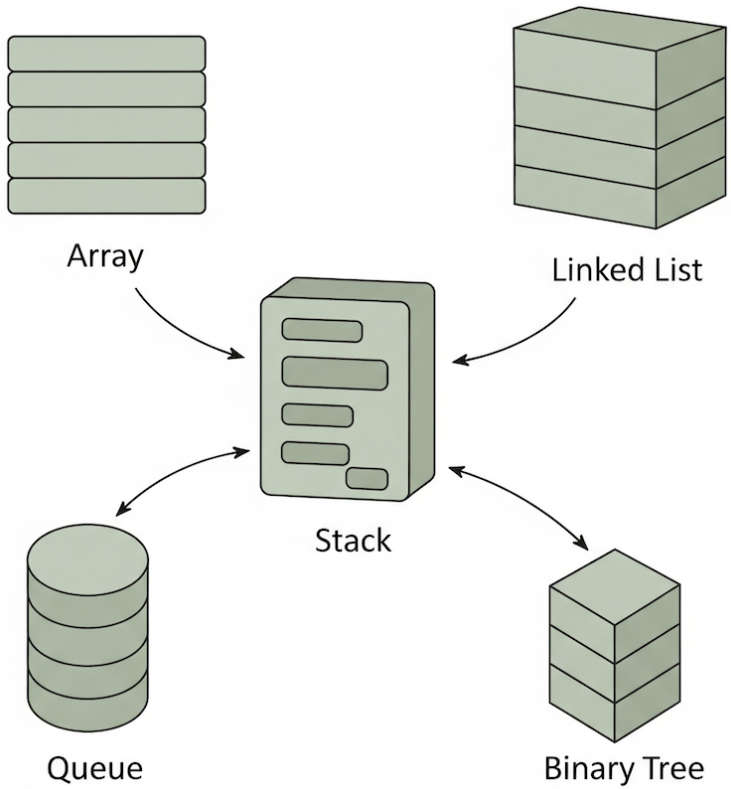
- Per la ricerca di un elemento in **un miliardo** di elementi:
- la ricerca sequenziale può richiedere anche un miliardo di passi
- la ricerca binaria richiede al più 30 passi

Ordinamento

- **Mantenere l'ordine è importante**
- Algoritmi naif: Insertion, Selection, Bubble
- Algoritmi efficienti: Merge, Quick, Heap Sort
- Limite inferiore ai tempi di calcolo per l'ordinamento
- Algoritmi lineari in casi speciali

Strutture dati

- Array, liste, pile, code
- Alberi binari e visite
- Perché contano: organizzare i dati $\rightarrow$ algoritmi più rapidi



Il “viaggio” del corso

- Dal concetto di algoritmo $\rightarrow$ analisi del costo
- Dai problemi base (ricerca, ordinamento) $\rightarrow$ strutture dati avanzate
- Obiettivo: strumenti per affrontare qualsiasi problema computazionale

Dizionari: operazioni fondamentali

- Inserisci
- Cerca
- Cancella

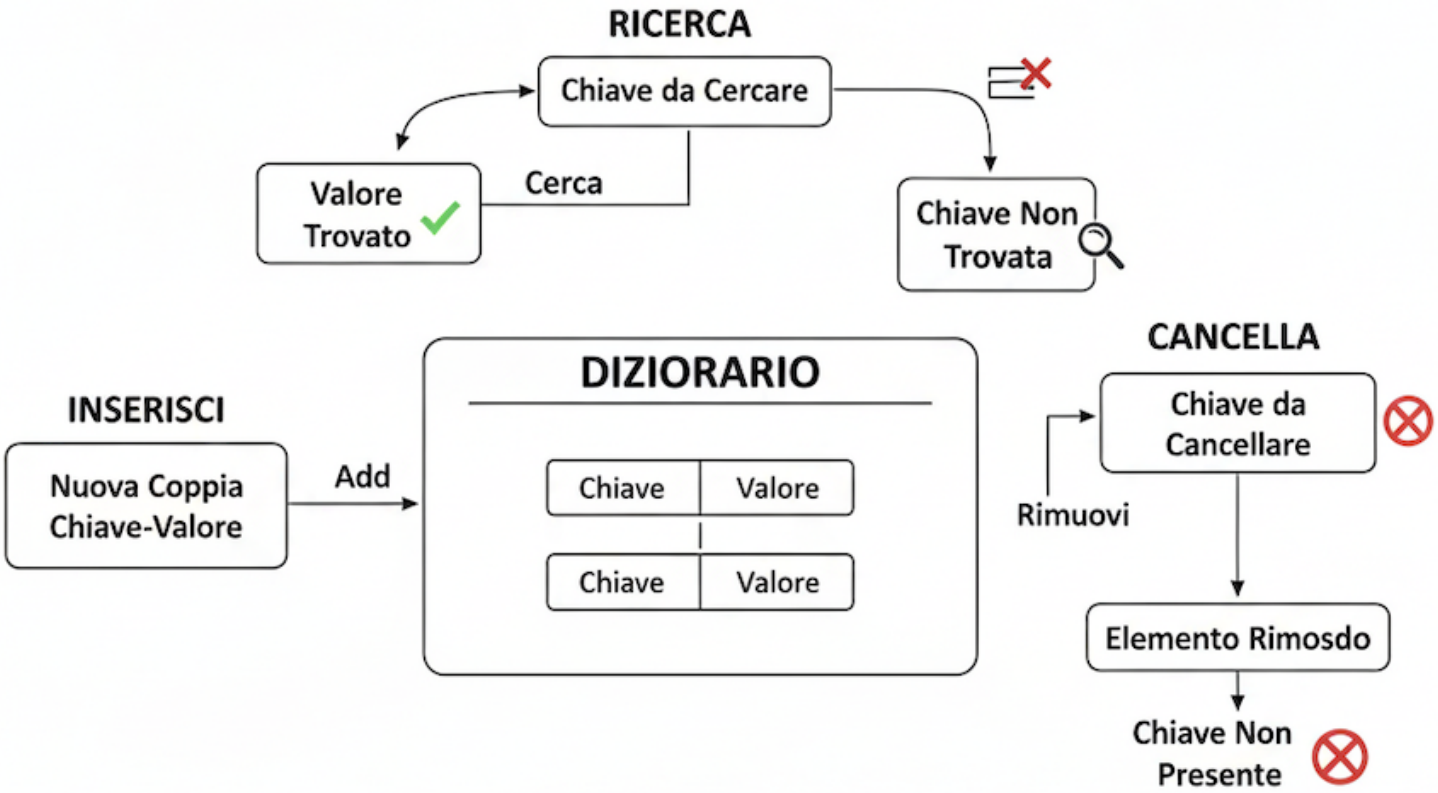


Tabelle hash

- Mappe chiave $\rightarrow$ posizione in array tramite funzione hash

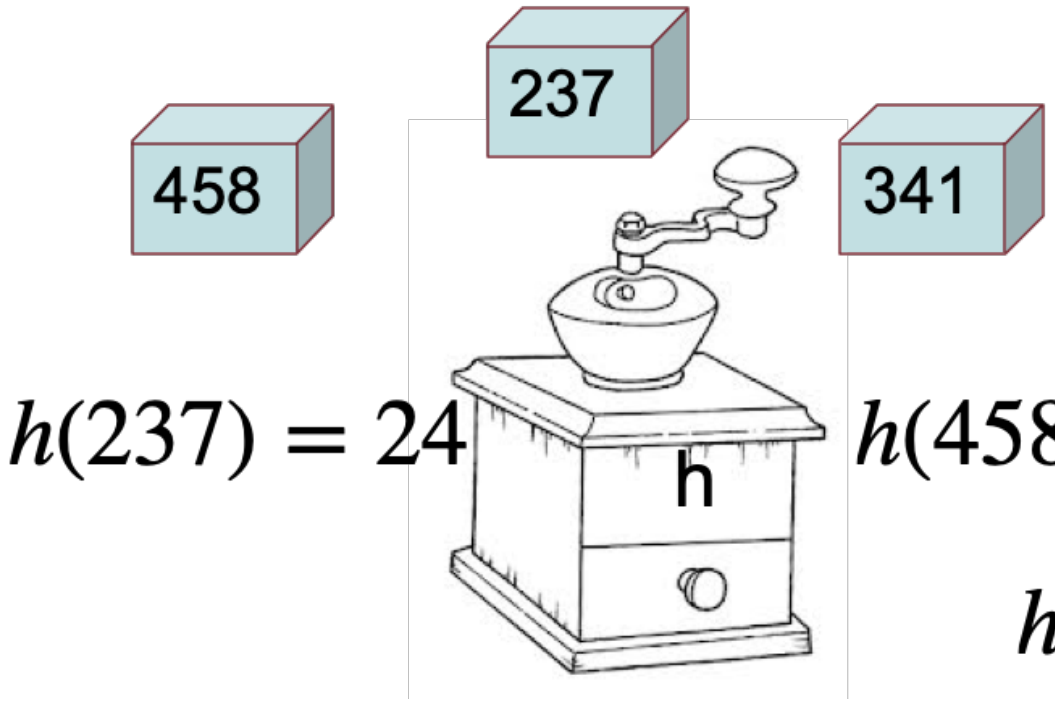


Tabelle hash

- Possibili collisioni da gestire

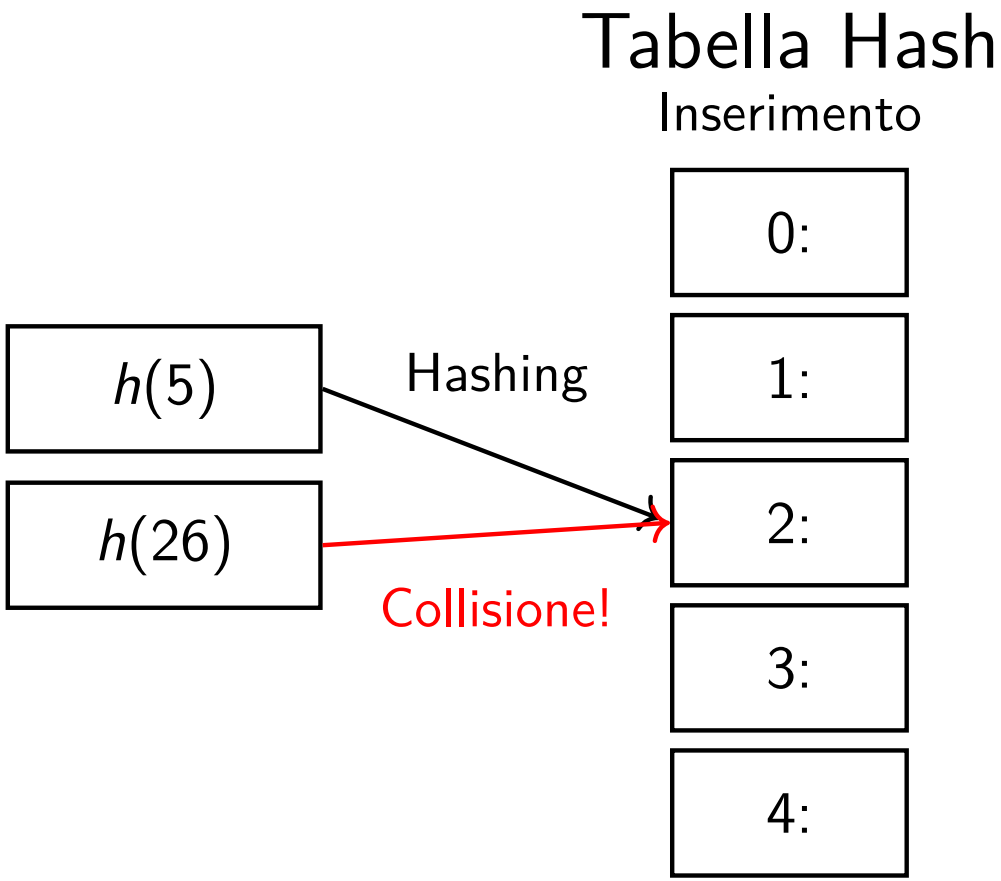


Tabelle hash

- Possibili collisioni gestite con **liste di trabocco**
- Accesso atteso: $O(1)$

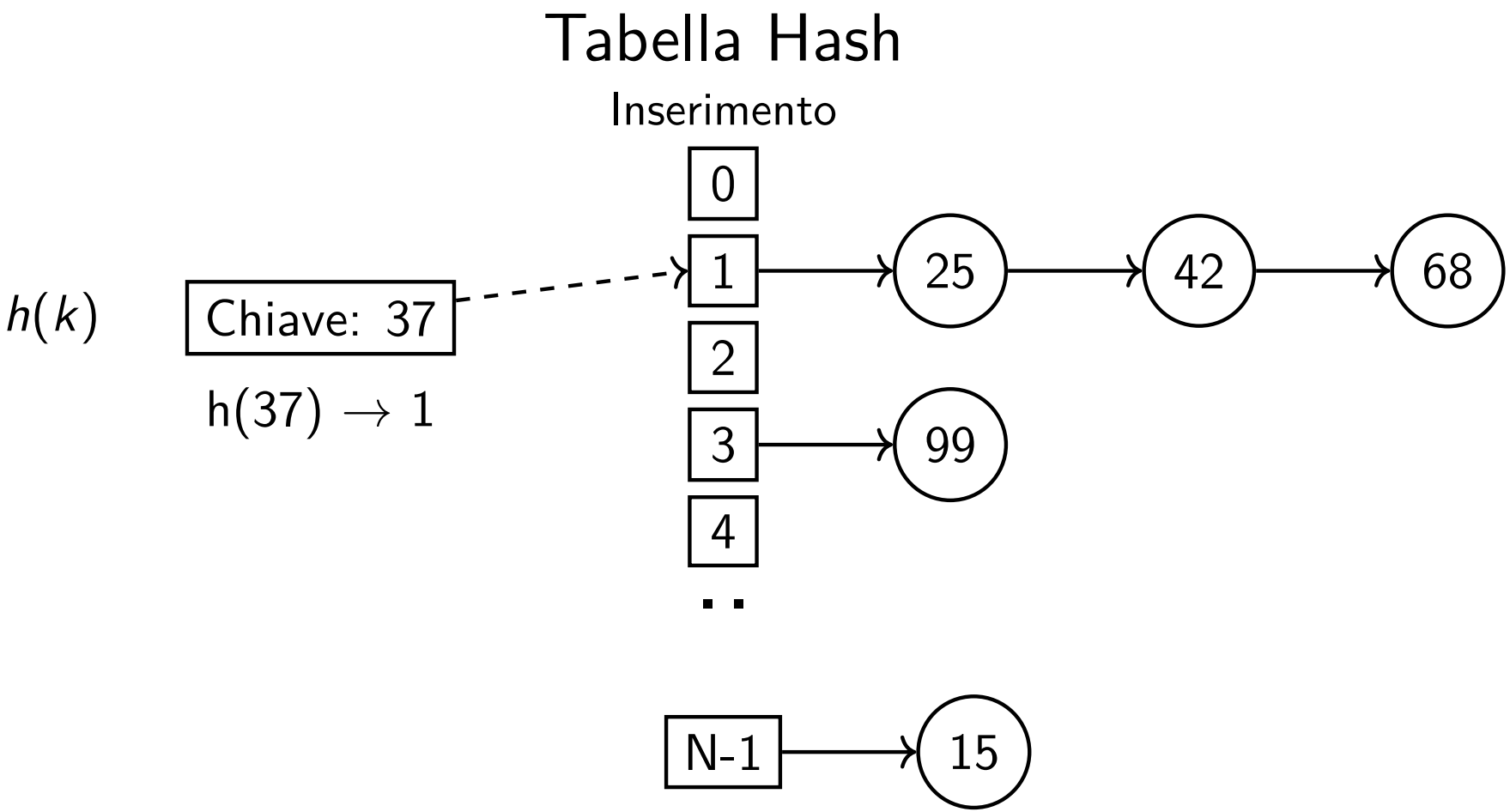
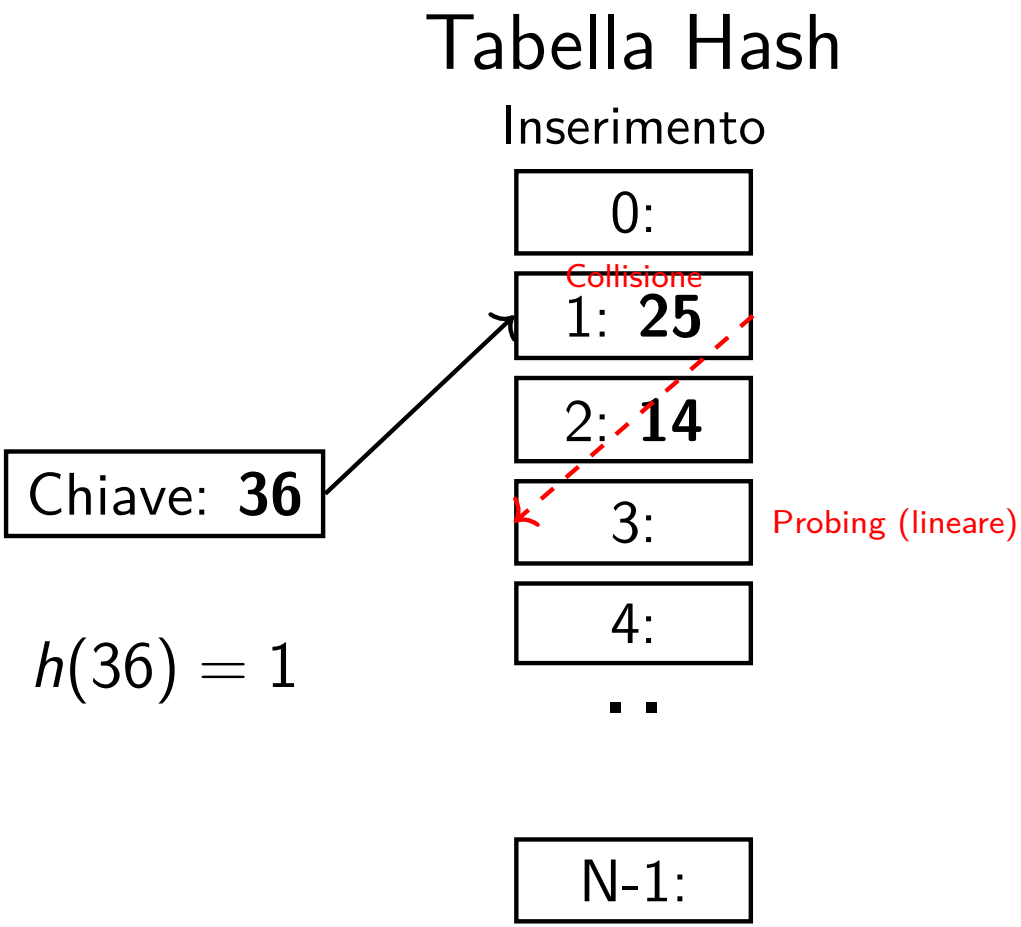


Tabelle hash

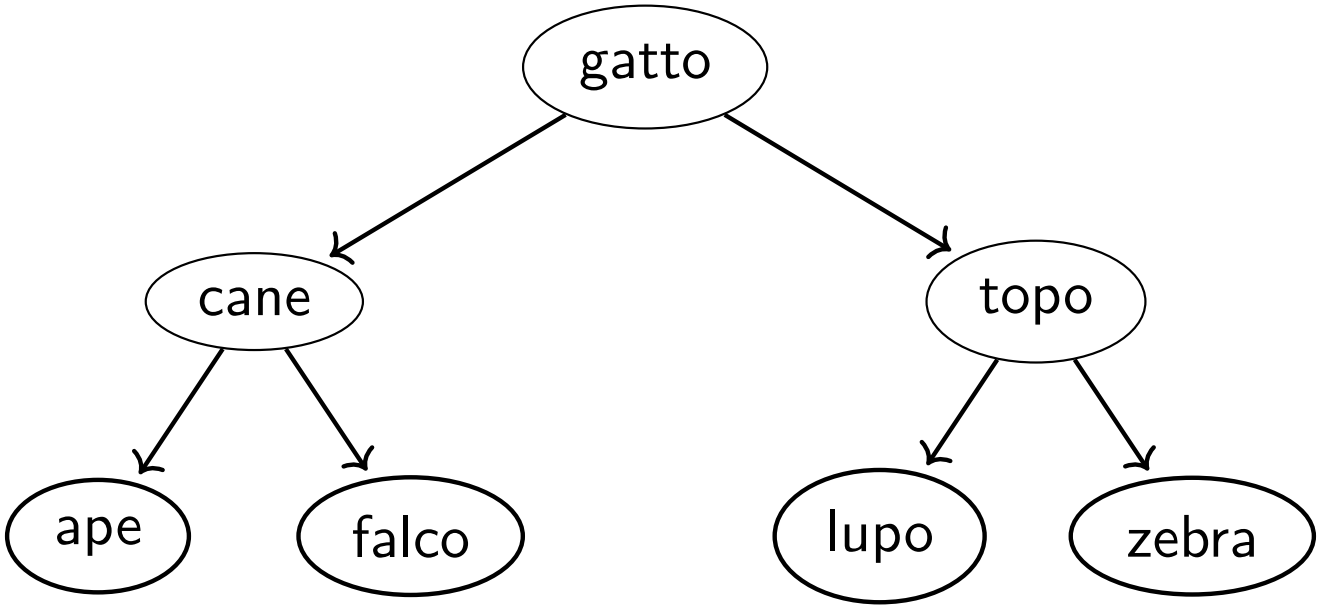
- Possibili collisioni gestite con **indirizzamento aperto** (vai alla prima locazione libera successiva)
- Accesso atteso: $O(1)$



chiave $= 25$, $h(25) = 1$

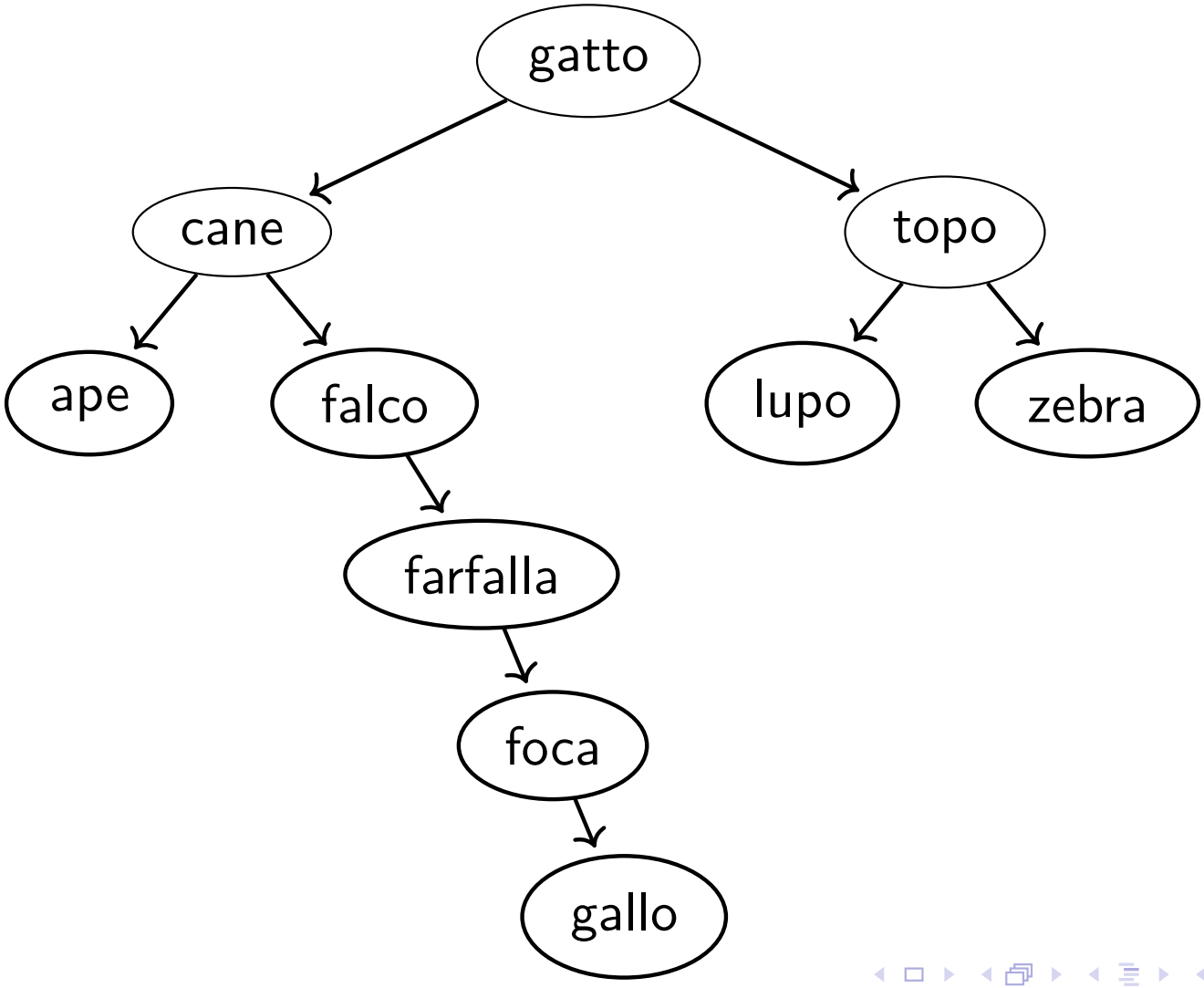
chiave $= 14$, $h(14) = 2$

Alberi di ricerca binari (BST)



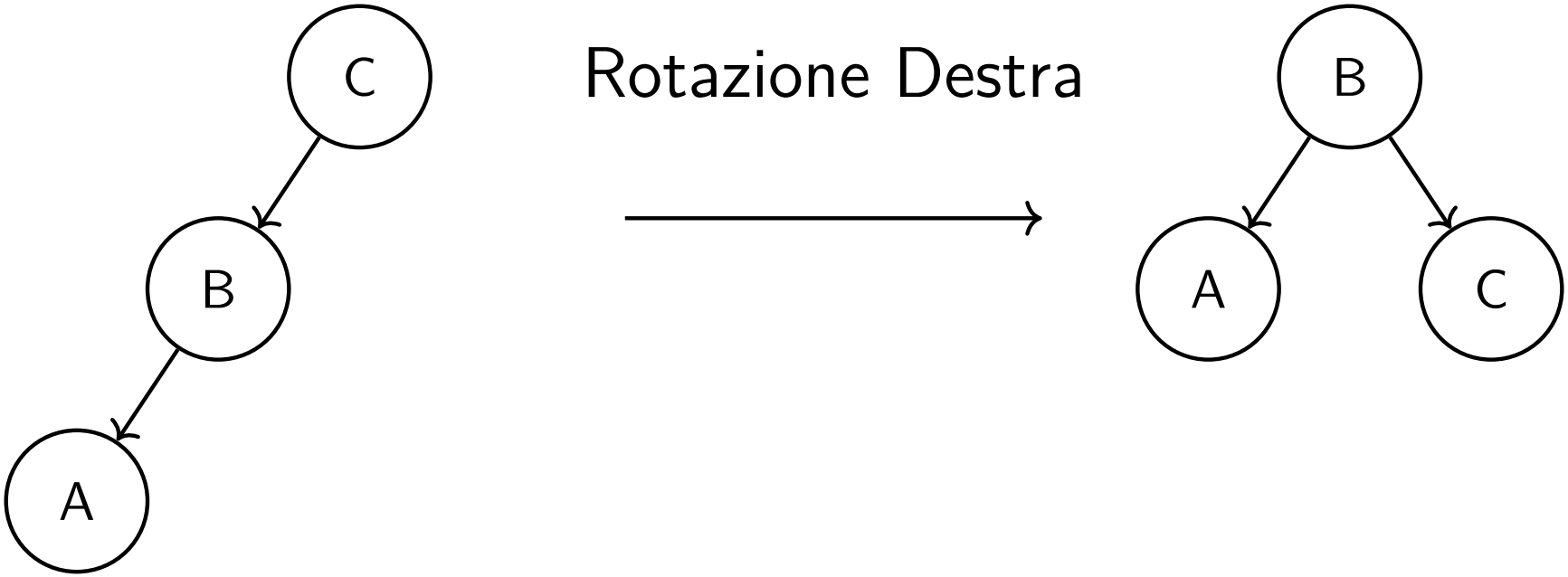
Alberi di ricerca binari (BST)

- Operazioni: inserimento, ricerca, cancellazione
- Problema: sbilanciamento $\rightarrow$ peggiora le prestazioni



Alberi di ricerca binari (BST)

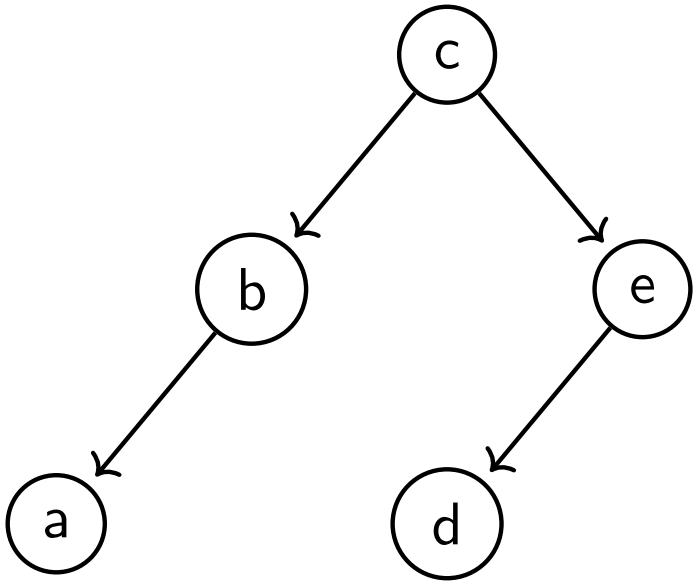
- Operazioni di ribilanciamento dopo inserimento e cancellazione
- Rotazioni



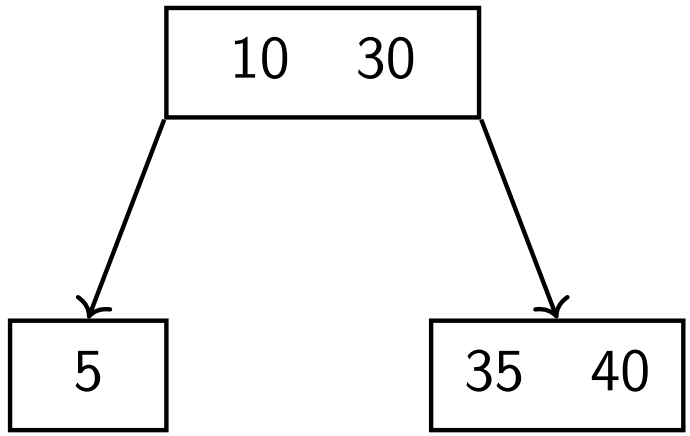
Alberi bilanciati: AVL e B-alberi

- Alberi che conservano il bilanciamento
  - AVL: rotazioni per bilanciare dopo inserimenti/cancellazioni
  - B-alberi: usati in database, ottimizzati per accesso su disco

Albero Binario di Ricerca



B-albero



Conclusione

- Gli algoritmi sono il cuore dell'informatica
- Ci insegnano a pensare in termini di efficienza
- Non solo programmi che funzionano, ma programmi intelligenti

ESERCIZI

Suggerimento: Prima di cercare le soluzioni in rete, prova a ragionarci da solo: gli esercizi che seguono sono pensati per aiutarti a mettere alla prova la tua comprensione, la tua capacità di sviluppare una prova di correttezza e costruire un algoritmo corretto.
Solo dopo averci riflettuto, confronta la tua soluzione con altre possibili versioni.