

ASD Laboratorio 07

The A(SD)-Team

UniTN

2025-02-27

ESERCITATORI

- Cristian Consonni (`cristian.consonni@unitn.it`)
- Quintino Francesco Lotito (`quintino.lotito@unitn.it`)

TUTOR

- Luca Mosetti (`luca.mosetti-1@studenti.unitn.it`)
- Riccardo Benevelli
(`riccardo.benevelli@studenti.unitn.it`)

CALENDARIO

27/02	Programmazione dinamica
03/04	Programmazione dinamica
06/05	Algoritmi approssimati 1
15/05	Algoritmi approssimati 2
22/05	Presentazione Progetto 2 (no lab)
27/05	Progetto alg approssimati
29/05	Progetto alg approssimati

PROGETTO ALGORITMI APPROSSIMATI

- Il secondo progetto verrà assegnato il **22/05/2025** e avrete circa una settimana di tempo. Il progetto riguarderà gli algoritmi approssimati (ultima parte del corso);
- Per iscriversi (se non vi siete iscritti al primo progetto):
https://bit.ly/ASDprog_2024-2025 (dovete essere loggati con l'account UniTN)
- Assumiamo gli stessi gruppi del primo semestre, in caso di cambiamenti, avvisare **entro il 19/05/2025**;

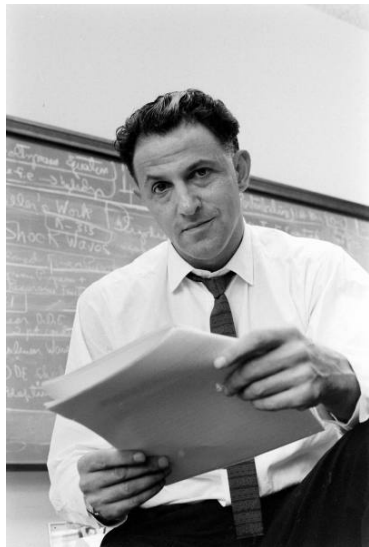
PROGRAMMAZIONE DINAMICA

Termine definito da Richard Bellman agli inizi degli anni '50, mentre lavorava con l'aeronautica militare statunitense.

Si riferiva al processo di risolvere un problema compiendo le migliori decisioni una dopo l'altra.

Programmazione → pianificazione
logistica

Dinamica → multilivello, variabile nel
tempo



DEFINIZIONE

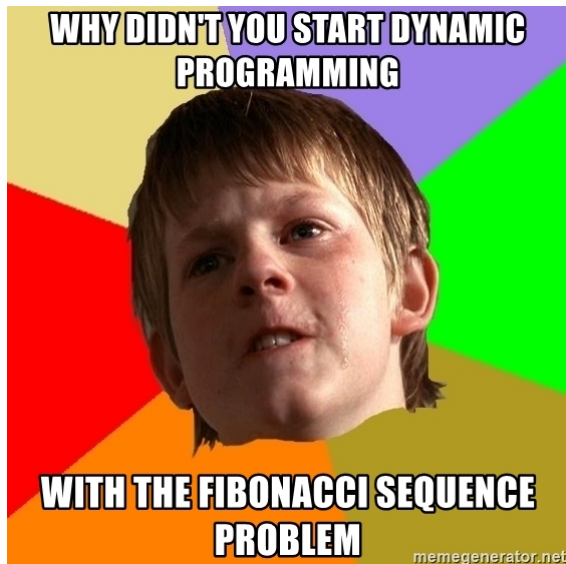
Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

SIGNIFICATO

Decidiamo il passo corrente in base allo stato corrente, senza riguardare le decisioni prese in precedenza.

Il tipico ragionamento è il seguente:

- 1 Definire i sottoproblemi/stati
- 2 Trovare l'equazione di ricorrenza che li collega
- 3 Utilizzare la memoization
- 4 Ridefinire l'algoritmo in maniera iterativa (opzionale)



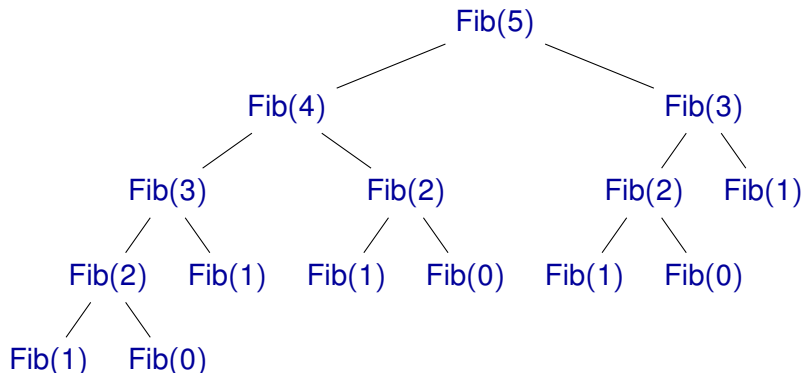
COMINCIAMO DA UN ESEMPIO

$$\text{Fib}(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{altrimenti} \end{cases}$$

problema “grande” **sotto-problemi**

The diagram shows the Fibonacci function definition. A teal arrow points from the text 'problema "grande"' to the 'Fib(n)' term in the equation. Another teal arrow points from the text 'sotto-problemi' to the 'Fib(n-1) + Fib(n-2)' term in the equation, with two smaller teal arrows branching out from this point to the individual 'Fib(n-1)' and 'Fib(n-2)' terms, illustrating the recursive decomposition of the problem into smaller sub-problems.

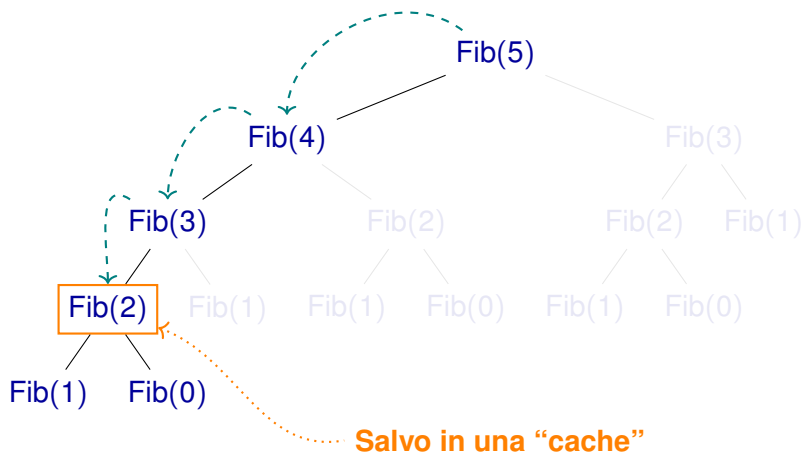
COMINCIAMO DA UN ESEMPIO



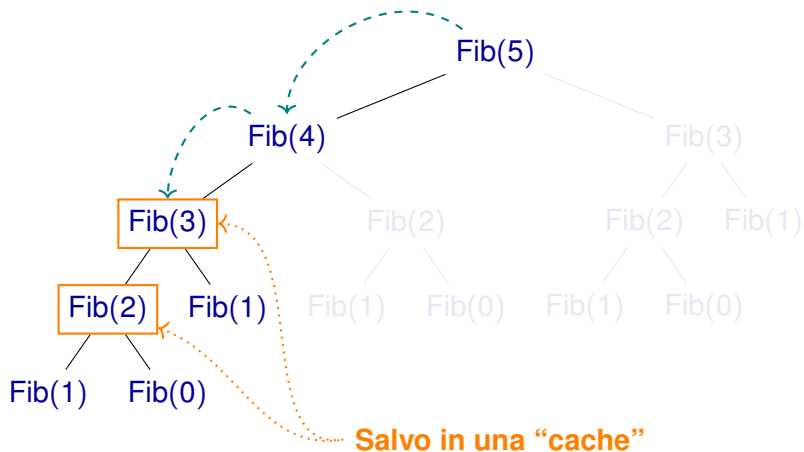
RICORSIONE

Fib(40) \approx 38s (in Python)

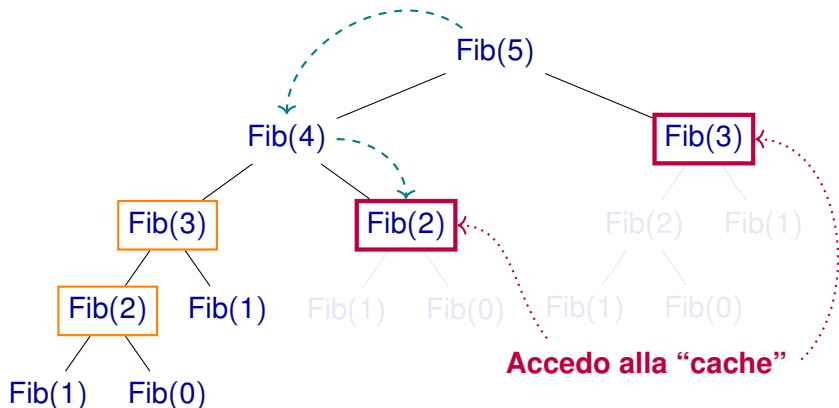
COMINCIAMO DA UN ESEMPIO



COMINCIAMO DA UN ESEMPIO



COMINCIAMO DA UN ESEMPIO



RICORSIONE CON MEMOIZATION

$Fib(500) \approx 1\text{ s}$ (in Python)

PROGRAMMAZIONE DINAMICA



imgflip.com

Il tipico ragionamento è il seguente:

- 1 Definire i sottoproblemi/stati
- 2 Trovare l'equazione di ricorrenza che li collega
- 3 Utilizzare la memoization
- 4 Ridefinire l'algoritmo in maniera iterativa (opzionale)

PROBLEMA DELLO ZAINO (SEMPLIFICATO)

Dato uno zaino di capacità C , ed un insieme di N elementi con peso P_i , vogliamo riempire al massimo lo zaino.

SOTTOPROBLEMI

Un sottoproblema è conseguenza di una serie di decisioni precedenti.

PROPRIETÀ DEI SOTTOPROBLEMI

- Devono essere autosufficienti: lo stato del sottoproblema deve contenere abbastanza informazioni da permettere la costruzione di una strategia ottima.
- Devono essere meno possibili.
- Siamo felici se si sovrappongono.

ESEMPI DI SOTTOPROBLEMI

- Sottoarray da i a N
- Sottoarray da i a j
- Sottoalbero radicato in v

Nel **problema dello zaino** può essere scelto come sottoproblema: riempire capacità c utilizzando gli oggetti dall' i -esimo in poi.

EQUAZIONE DI RICORRENZA

Ridefinire il costo della soluzione del problema corrente in base alle soluzioni di altri sottoproblemi.

DEFINIZIONE DELL'EQUAZIONE

- Consideriamo un ordinamento sui sottoproblemi, per evitare cicli infiniti
- Individuiamo le possibili scelte
- Le proviamo e prendiamo la migliore
- ... Facendo attenzione ai casi base

Nel **problema dello zaino**: possiamo scegliere se inserire l' i -esimo elemento o no.

$$S(c, i) = \begin{cases} \max \begin{cases} P_i + S(c - P_i, i + 1) \\ S(c, i + 1) \end{cases} & \text{if } i < N \\ -\infty & \text{if } c < 0 \\ 0 & \text{if } i == N \end{cases}$$

Evitiamo di ricalcolare più volte la stessa funzione. Memorizziamo in una tabella T le soluzioni dei vari sottoproblemi.

CONDIZIONE NECESSARIA

La funzione di ricorrenza deve dipendere solo da:

- Strutture costanti nell'esecuzione (es: la lista dei pesi)
- I parametri della funzione (c e i)

MEMOIZATION

Per ogni stato il cuore della funzione ricorsiva (il calcolo vero e proprio della soluzione) verrà eseguito soltanto una volta.

Algorithm 1: `solve(State s)` :

// casi base;

if $T[s] \neq null$ **then**

return $T[s]$;

end

// chiamate ricorsive per calcolare sol;

$T[s] = sol$;

return $T[s]$;

Nel **problema dello zaino**: uno stato è dato da capacità e oggetto correnti (c, i) . La soluzione del problema iniziale è data da $S(C, 0)$.

TABELLA DI MEMOIZATION

Matrice statica: `int T[MAXN][MAXM];`

Vettore dinamico:

`T=vector<vector<int>>(N,vector<int>(M,-1));`

Map: `map<pair<int,int>,int> T;`

State attenti alla memoria usata! Con la map spendete solo quello che utilizzate, ma pagate $O(\log n)$ per accesso.

PSEUDO-POLYNOMIAL TIME ALGORITHMS

Se vi sembra magico il fatto di poter risolvere problemi $NP-HARD$, come il problema dello zaino, con algoritmi di complessità (apparentemente) polinomiale, date un'occhiata qui:

https://en.wikipedia.org/wiki/Pseudo-polynomial_time

Testi completi su judge

ZAINO

Dato uno zaino di capacità C ed un insieme di elementi di peso P_i e di valore V_i , trovare il massimo valore che è possibile trasportare.

SOTTOINSIEME CRESCENTE DI SOMMA MASSIMA

Vi viene dato un array di interi. Per ogni elemento potete scegliere se includerlo nell'insieme soluzione, o se ignorarlo. Se un elemento X fa parte dell'insieme, tutti gli elementi che lo succedono nell'array e che hanno valore minore di X non possono essere inclusi nell'insieme. Bisogna massimizzare la somma dell'insieme.

PILLOLE

La zia ha N pillole in una bottiglia. Ogni giorno, pesca una pillola dalla bottiglia:

- 1 se è intera, la spezza in due metà, ne mangia una metà e rimette il resto nella bottiglia. (Caso I)
- 2 se è mezza pillola, la mangia. (Caso M)

Con 3 pillole, sono possibili queste sequenze:

IIIMMM IIMIMM IIMMIM IMIIMM IMIMIM

Quante sequenze sono possibili con N pillole?