ASD Laboratorio 07

The A(SD)-Team

UniTN

2023-03-07

Contatti

Esercitatori

- Cristian Consonni (cristian.consonni@unitn.it)
- Quintino Francesco Lotito (quintino.lotito@unitn.it)
- Gabriele Masina (gabriele.masina@unitn.it)

Tutor

- Daniele Cabassi (daniele.cabassi@studenti.unitn.it)
- Alessio Faieta (alessio.faieta@studenti.unitn.it)
- Filippo Momesso (filippo.momesso@studenti.unitn.it)
- Luca Mosetti (luca.mosetti-1@studenti.unitn.it)
- Elisa Trento (elisa.trento@studenti.unitn.it)

Calendario

07/03	Programmazione dinamica
28/03	Programmazione dinamica
09/05	Algoritmi approssimati 1
16/05	Algoritmi approssimati 2
25/05	Progetto alg approssimati
30/05	Progetto alg approssimati

Progetto algoritmi approssimati

- Il progetto verrà assegnato il **24/05/2023** e avrete circa una settimana di tempo;
- Algoritmi approssimati (ultima parte del corso);
- Assumiamo gli stessi gruppi del primo semestre, in caso di cambiamenti, avvisare **entro il 22/05/2023**;

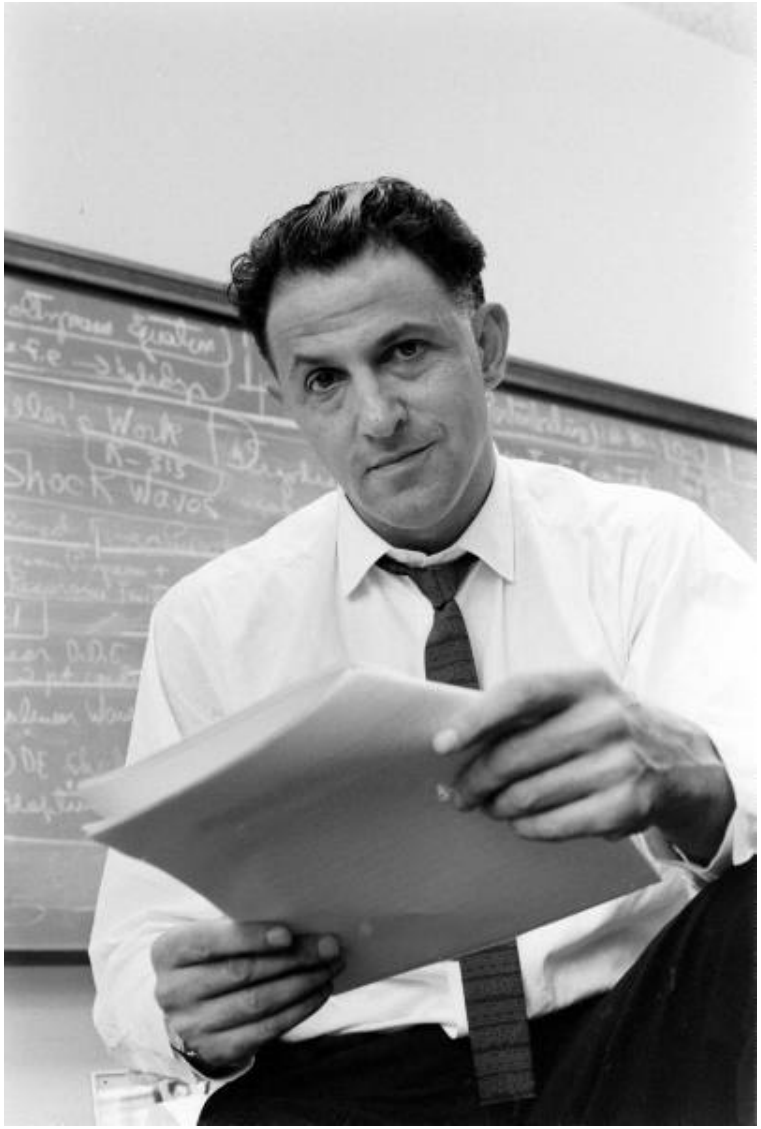
Programmazione dinamica

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 $\rightarrow$ pianificazione logistica

Dinamica $\rightarrow$ multilivello, variabile nel tempo



Principio di ottimalità di Bellman

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

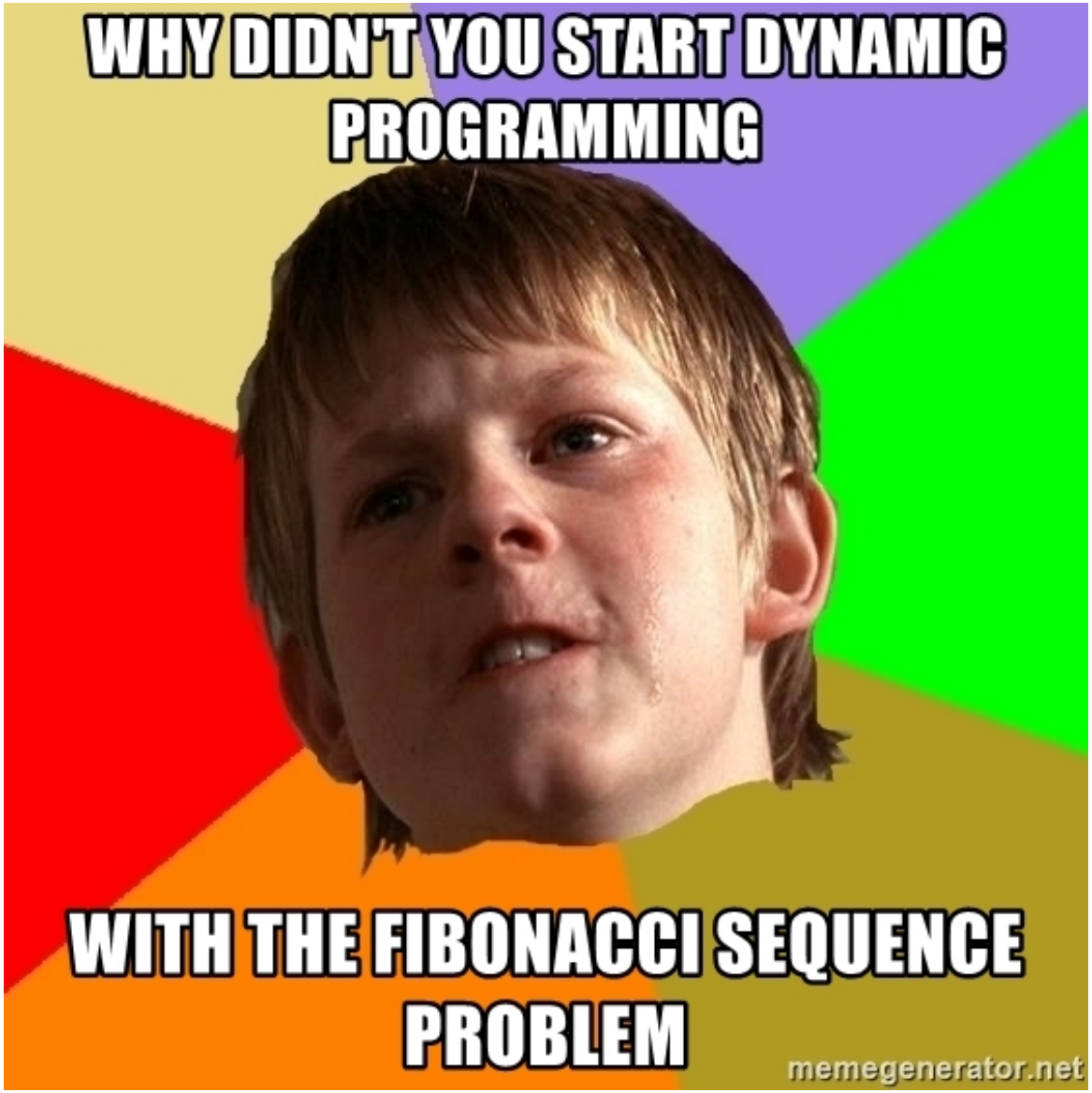
Decidiamo il passo corrente in base allo stato corrente, senza riguardare le decisioni prese in precedenza.

Procedura tipica

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)

Programmazione dinamica

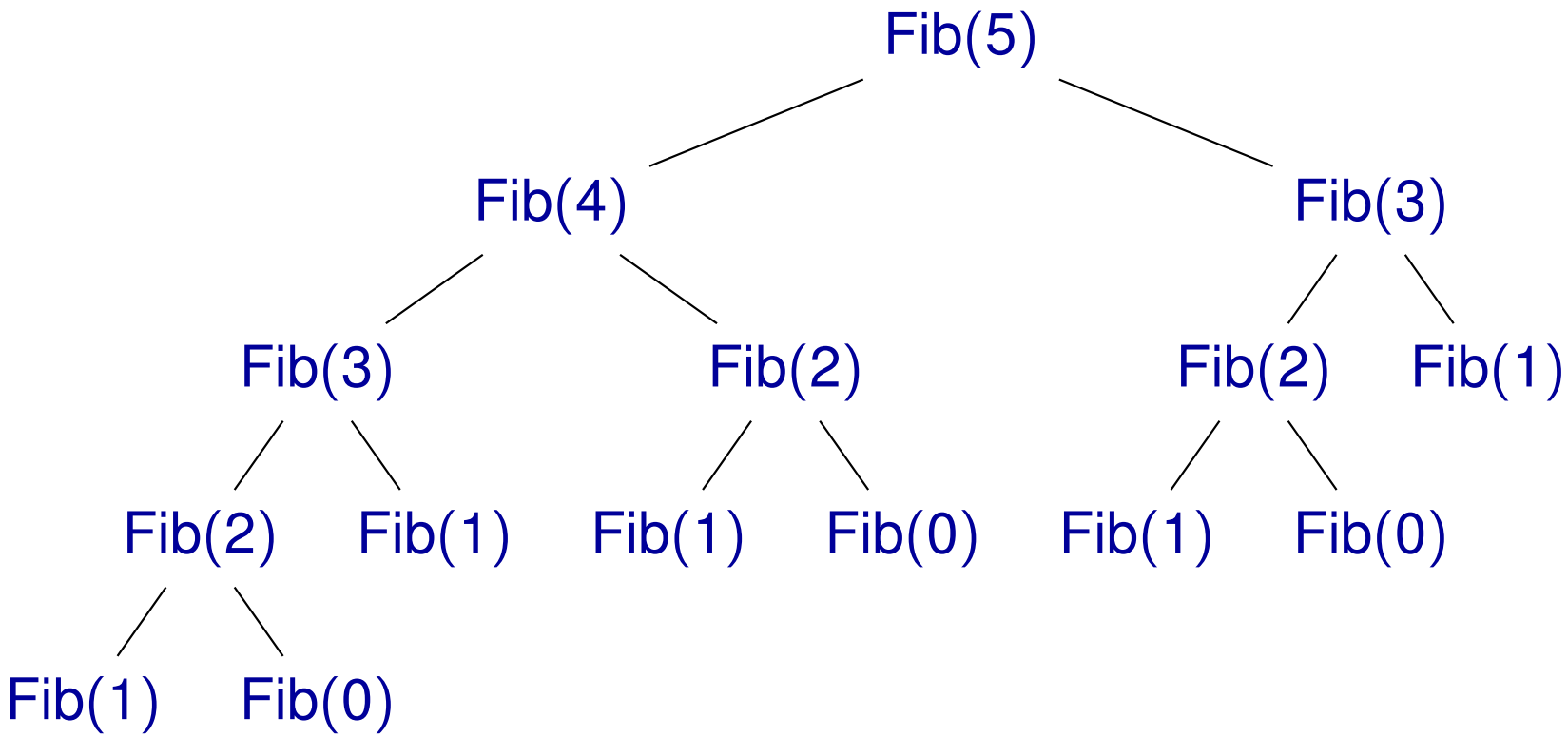


Cominciamo da un esempio

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

problema "grande" **sotto-problemi**

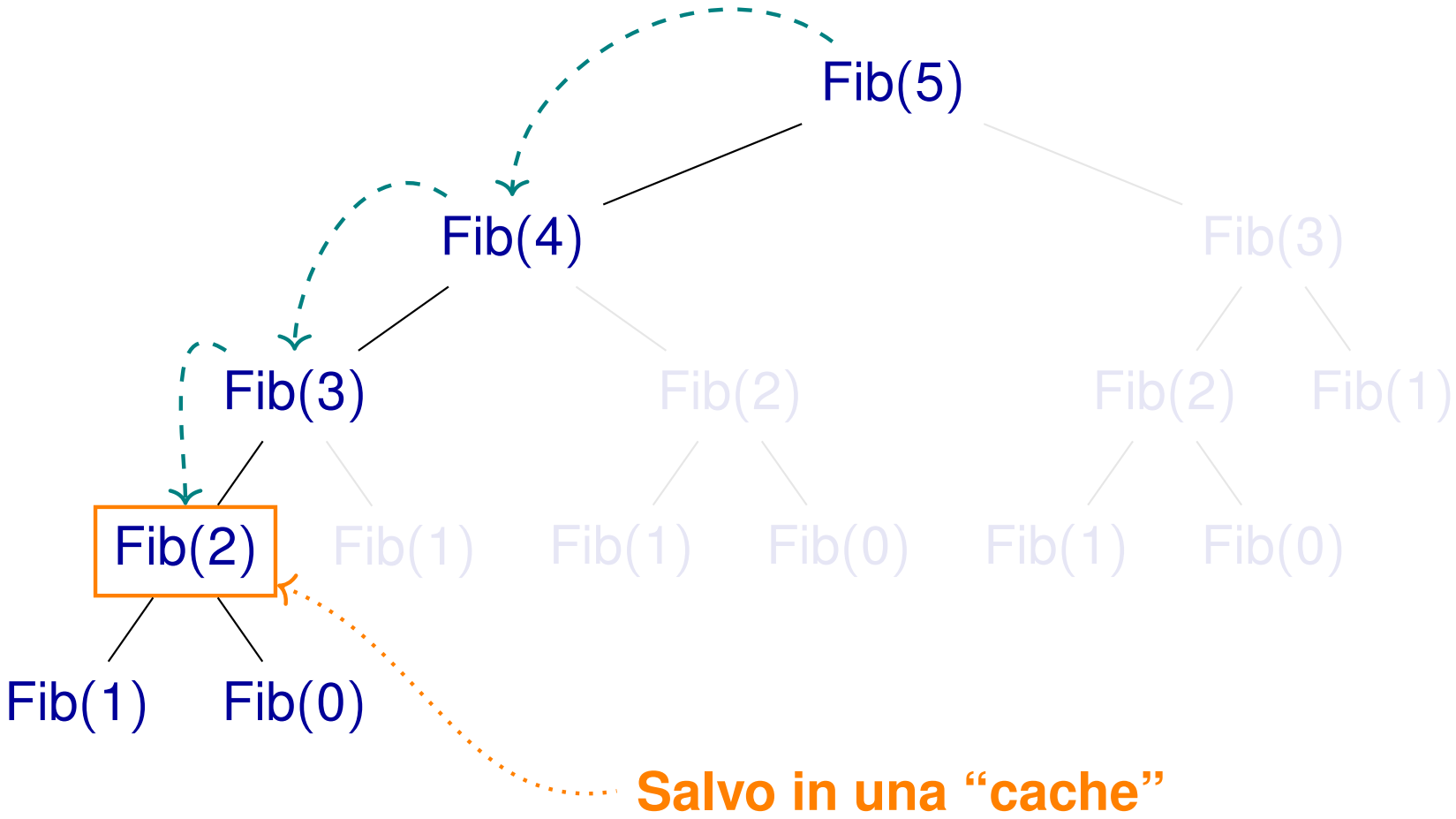
Cominciamo da un esempio



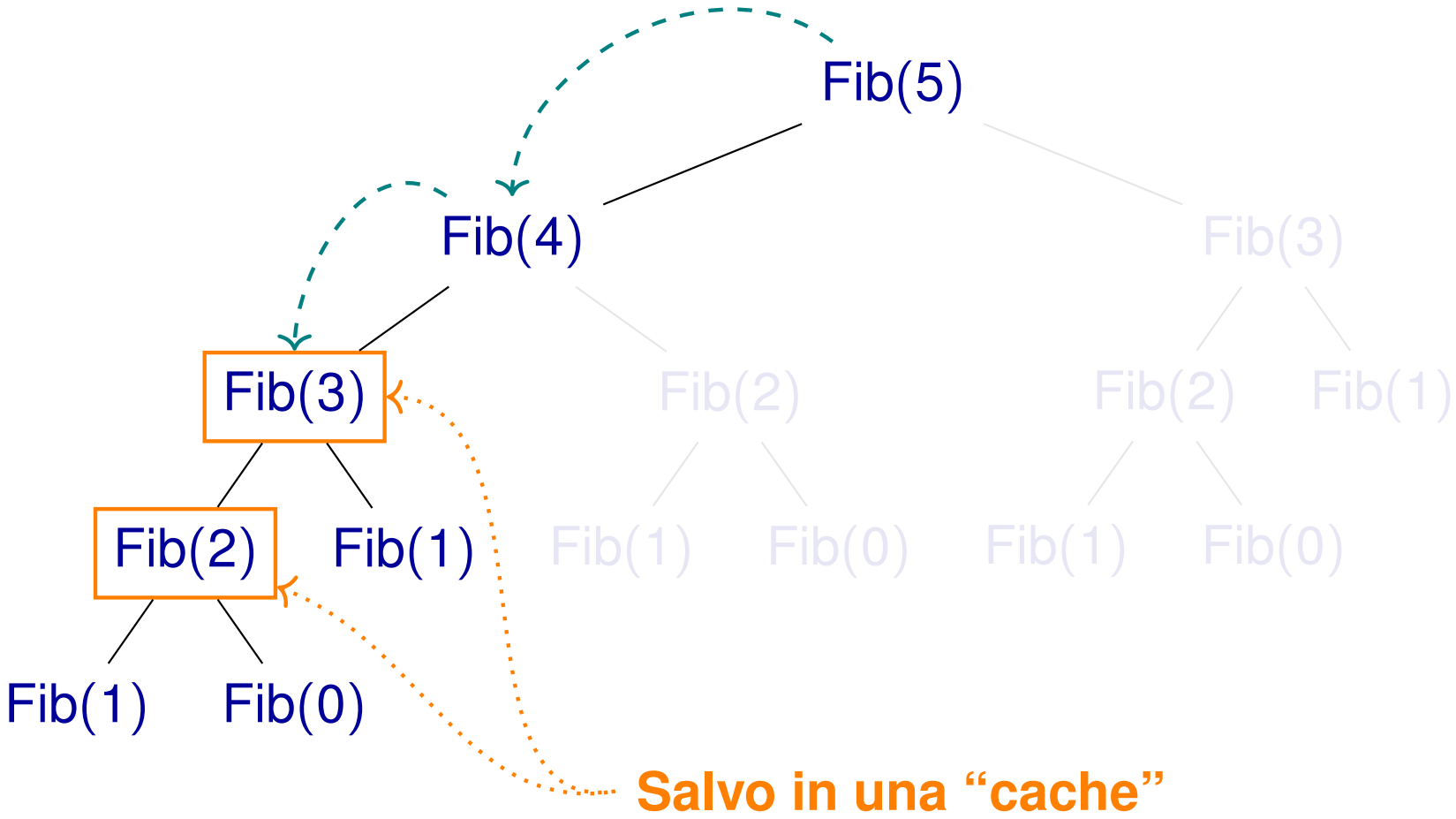
Ricorsione

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

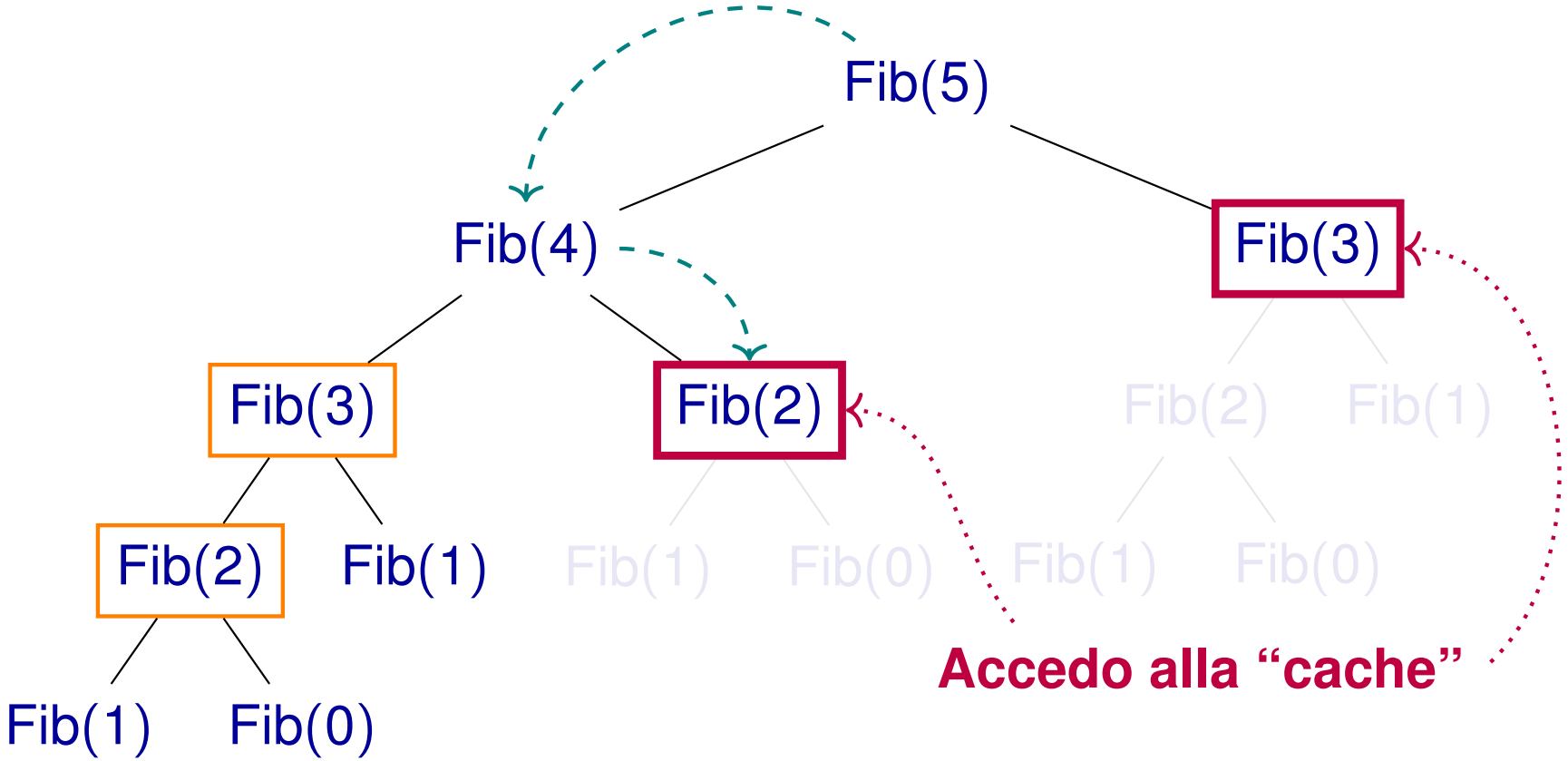
Cominciamo da un esempio



Cominciamo da un esempio



Cominciamo da un esempio



Ricorsione con memoization

Fib(500) $\approx$ 1s (in Python)

Programmazione dinamica

Procedura tipica

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.

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

Memoization

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] ≠ 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 $S(C, 0)$.

Note di programmazione

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.

Nota teorica

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`

Problemi

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 possible 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.

Problemi

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?