LEETCODATAAAAA
DEL 17/06/2025

Bacaro
Tech

CODE AND FUN

Bacaro Tech

CODE AND FUN

ANTI SPOILER

Oggi vi proponiamo degli esercizi a tema, se non volete spoiler sul tema e sulle tecniche di risoluzione e volete risolverli senza suggerimenti vi mettiamo i link e ci sentiamo alla fine per il confronto

https://leetcode.com/problems/counting-bits/

https://leetcode.com/problems/min-cost-climbing-stairs/

https://leetcode.com/problems/unique-paths/

LET’S START

Bacaro Tech

CODE AND FUN

PRESENTAZIONE DYNAMIC PROGRAMMING

Il tema di oggi è la programmazione dinamica (Dynamic Programming - DP) che è una tecnica per risolvere un problema difficile scomponendolo in sotto problemi: risolviamo i sotto problemi e troveremo la soluzione.

PROGRAMMAZIONE DINAMICA
COME DISTIGNUERLI

Come distinguerli?
Non tutti i problemi si possono risolvere con la DP.
Capire come distinguerli è un po' complicato e per questo aspetto proveremo a trarre delle conclusioni alla fine del primo esercizio

Attenzione che ricorsione e DP non sono la stessa cosa, anche se possono assomigliare(tanto che la ricorsione viene usata nella programmazione dinamica).

PROGRAMMAZIONE DINAMICA
COME RISOLVERLI

Bisogna trovare i sottoproblemi e costruire una tabella(o un vettore) con i casi dei sottoproblemi

Per fare questo bisogna individuare questi tre elementi:
1) **stato** → Quale sotto-problema sto risolvendo in questa cella?
2) **caso base** → Dove inizia la tabella?
3) **relazione** → Come ottengo dp[i][j] dai sotto-problemi più piccoli?
...non sempre vengono usate matrici

SUGGERIMENTO PER LA SOLUZIONE

STATO

Rende il problema suddivisibile in parti **sovrapponibili**

CASO BASE

Fissa i valori noti da cui propagare le soluzioni: senza, la ricorrenza non avrebbe termino.

RELAZIONE DI TRANSIZIONE

Descrive l’aggancio fra stati; è il cuore dell’algoritmo e trasforma la ricorsione in tabulazione

PRACTICE
TIME

Bacaro Tech

CODE AND FUN

ESERCIZIO 1
MIN COST CLIMBING STAIRS

https://leetcode.com/problems/min-cost-climbing-stairs/

Link esercizio: min-cost-climbing-stairs
Difficoltà: easy
Tempo standard: 25 minuti
Tempo di recupero: 5 minuti
Area di interesse: Dynamic Programming - D1

Bacaro Tech
CODE AND FUN

CAPIAMO IL PROBLEMA
MIN COST CLIMBING STAIRS

Facciamo un esempio dove la scala è cost = [10,15,20]

STATO	dp[i] = minimo costo per salire la scala allo scalino i
CASO BASE	dp[0] = cost[0](minimo costo per salire al gradino 0) = 10 dp[1] = cost[1] = 15
RELAZIONE DI TRANSIZIONE	Per ogni **i ≥ 2: dp[i] = cost[i] + min(dp[i-1], dp[i-2])** Al gradino **i** puoi arrivarci solo da i-1 o da i-2, perché in questo esercizio o sali un gradino o sali due gradini per volta. Quando arrivi al gradino i devi comunque pagare cost[i] quando ci atterri

IT’S TIME TO CODING!

Bacaro Tech
CODE AND FUN

SOLUZIONI PROPOSTE
CONSIGLI

Vi proponiamo prima la soluzione in pseudo codice, così se non ci siete arrivati e volete provare almeno a trascrivere lo pseudo codice nel vostro linguaggio preferito potete farlo.

Poi vi proponiamo due soluzioni: una in Java e una in Python.

Ricordiamo che le soluzioni si possono trovare con moltissimi linguaggi e data una soluzione nel nostro linguaggio preferito, capire come farla in un altro linguaggio può aiutarci a conoscerli in modo pratico

Bacaro
Tech
CODE AND FUN

PSEUDO CODICE

```
INPUT:  array cost[0 … n-1]          # n ≥ 2
OUTPUT: minimo costo per raggiungere il pianerottolo (indice n)
# 1.  Casi base
rev2 ← cost[0]                       # = dp[i-2]   (gradino 0)
prev1 ← cost[1]                      # = dp[i-1]   (gradino 1)

FOR i FROM 2 TO n-1 DO               # per ogni gradino successivo
    cur ← cost[i] + MIN(prev1, prev2)   # dp[i] = cost[i] + min( dp[i-1], dp[i-2] )
    prev2 ← prev1                    # shift delle finestre
    prev1 ← cur
END FOR

RETURN MIN(prev1, prev2)             # min( dp[n-1], dp[n-2] )
```

Bacaro
Tech

CODE AND FUN

JAVA

```
public int minCostClimbingStairs(int[] cost) {
    int n = cost.length;
    if (n == 0) return 0;                  // scala vuota
    if (n == 1) return cost[0];            // un solo gradino
    int prev2 = cost[0];                   // dp[i-2]  (gradino 0)
    int prev1 = cost[1];                   // dp[i-1]  (gradino 1)

    for (int i = 2; i < n; i++) {
        int cur = cost[i] + Math.min(prev1, prev2); // dp[i]
        prev2 = prev1;
        prev1 = cur;
    }
    return Math.min(prev1, prev2);         // min(dp[n-1], dp[n-2])
}
```

Bacaro
Tech

CODE AND FUN

PYTHON

```
def min_cost_climbing_stairs(cost: list[int]) -> int:
    prev_two, prev_one = cost[0], cost[1]
    for i in range(2, len(cost)):
        current = cost[i] + min(prev_one, prev_two)
        prev_two, prev_one = prev_one, current
    return min(prev_one, prev_two)
```

LESSON LEARN
QUANDO USARE DP?

POSSO SCRIVERE UNA SOLUZIONE RICORSIVA SEMPLICE?

I SOTTO-PROBLEMI SI RIPETONO IDENTICI?

C'È UN MODO UNIVOCO DI COMBINARE LE RISPOSTE DEI SOTTO-PROBLEMI?

Bacaro Tech

CODE AND FUN

TECNICHE PER LA DP

BOTTOM UP

Parti dal problema originale e lo scomponi ricorsivamente nei sottoproblemi.

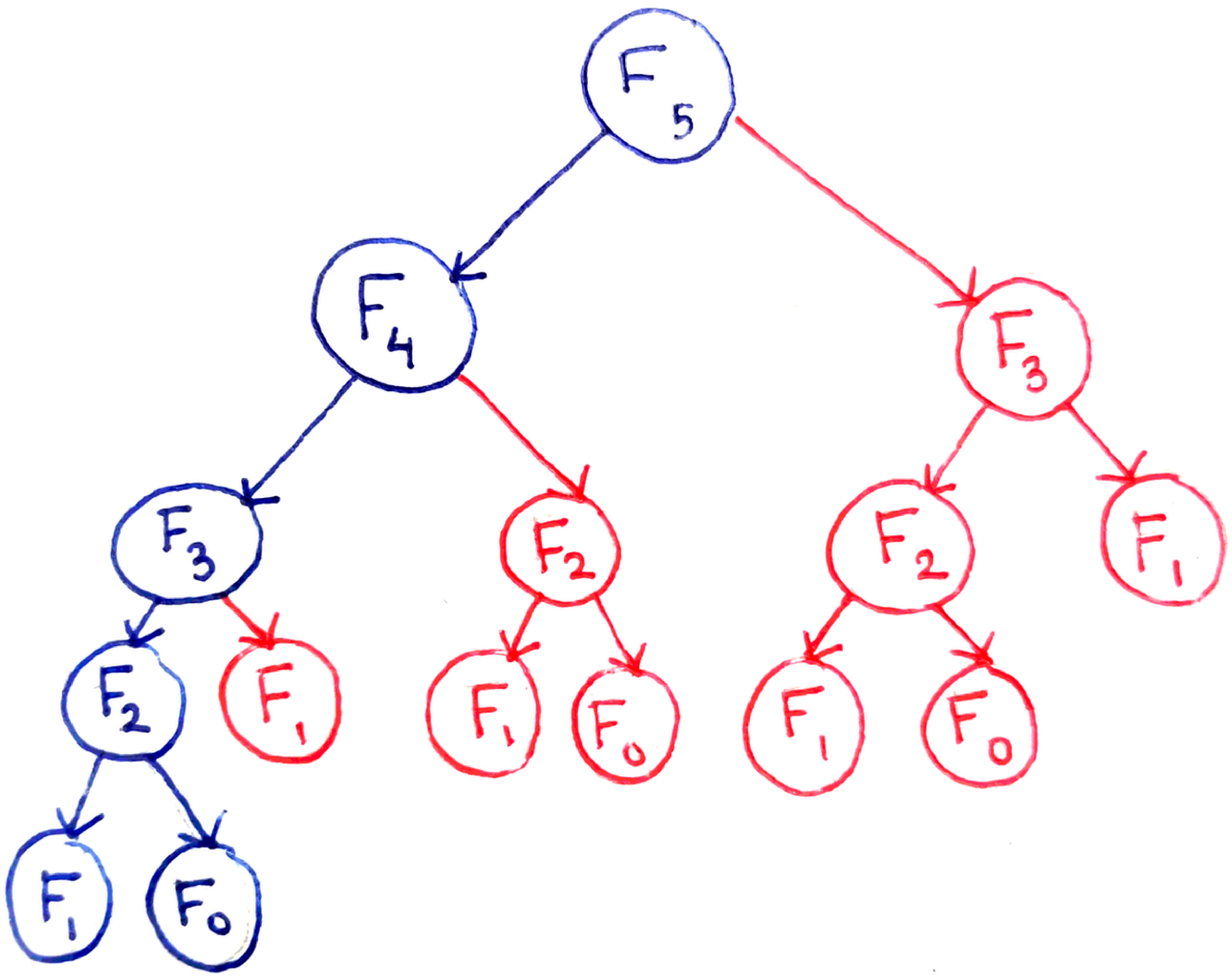
Memorizzi i risultati dei sottoproblemi già calcolati per evitare calcoli ripetuti.

Ricorsivo

TOP DOWN

Parti dai casi base e costruisci la soluzione finale passo dopo passo, salvando tutti i sottorisultati necessari in una tabella.

Iterativo


F_5
F_4
F_3
F_3
F_2
F_2
F_1
F_2
F_1
F_1
F_0
F_1
F_0
F_1
F_0

Bacaro
Tech
CODE AND FUN

ESERCIZIO 2
COUNTING BITS

https://leetcode.com/problems/counting-bits/

Link esercizio: counting-bits
Difficoltà: easy
Tempo standard: 25 minuti
Tempo di recupero: 5 minuti
Area di interesse: Dynamic Programming - D1

Bacaro Tech

CODE AND FUN

CAPIAMO IL PROBLEMA COUNTING BIT

Dato un numero, dobbiamo tornare un array che contiene per ogni cella il conteggio degli ‘1’ che ha il corrispettivo numero binario.
Leggiamo il testo e definiamo i tre passi per capire il problema.
Facciamo riferimento a un esempio n = 5

STATO dp[i] = numero di 1 del numero binario i

CASO BASE dp[000b] = 0, dp[001b] = 1, dp[2] = dp[010b]=1, dp[011]=2, dp[100b]=1 dp[101b]=2

RELAZIONE DI TRANSIZIONE
Opzione A: dp[i] = dp[i & (i-1)] + 1
Opzione B: dp[i] = dp[i >> 1] + (i & 1)

IT'S TIME TO CODING!

PSEUDO CODICE

```
funzione countBits(n):
    bits ← array di (n + 1) interi
    bits[0] ← 0
    per i da 1 a n:
        bits[i] ← bits[i div 2] + (i mod 2)
    restituisci bits
```

JAVA

```
public static int[] countBits(int n) {
        int[] bits = new int[n + 1];
        bits[0] = 0;
        for (int i = 1; i <= n; i++) {
            bits[i] = bits[i >> 1] + (i & 1);
        }
        return bits;
    }
```

PYTHON

```
def count_bits(n):
    bits = [0] * (n + 1)
    for i in range(1, n + 1):
        bits[i] = bits[i // 2] + (i % 2)
    return bits
```

Bacaro
Tech
CODE AND FUN

ESERCIZIO 3
UNIQUE PATHS

https://leetcode.com/problems/unique-paths/

Link esercizio: unique-paths
Difficoltà: medium
Tempo standard: 25 minuti
Tempo di recupero: 5 minuti
Area di interesse: Dynamic Programming - D1

CAPIAMO IL PROBLEMA UNIQUE PATHS

Dobbiamo tornare il numero di possibili percorsi che possiamo fare in una griglia n x m.

STATO

p[i][j] = numero di percorsi distinti per raggiungere la cella (i, j) partendo dall’angolo in alto a sinistra (0, 0) e muovendosi solo verso destra o in basso.

CAPIAMO IL PROBLEMA UNIQUE PATHS

CASO BASE

Prima riga: dp[0][j] = 1 per ogni j (puoi solo avanzare a destra).
Prima colonna: dp[i][0] = 1 per ogni i (puoi solo scendere).

RELAZIONE DI TRANSIZIONE

Per ogni cella interna $i \geq 1$, $j \geq 1$:
dp[i][j] = dp[i-1][j] + dp[i][j-1]
(somma dei percorsi che arrivano da sopra e da sinistra)

IT'S TIME TO CODING!

PSEUDO CODICE

```
Funzione uniquePaths(m, n):
    Crea una matrice dp di dimensione m x n, inizializzata
a 1 nella prima riga e nella prima colonna
    Per i da 1 a m-1:
        Per j da 1 a n-1:
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    Ritorna dp[m-1][n-1]
```

Bacaro
Tech
CODE AND FUN

JAVA

```java
public static int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
        for (int i = 0; i < m; i++)
            dp[i][0] = 1;
        for (int j = 0; j < n; j++)
            dp[0][j] = 1;
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
            }
        }
        return dp[m - 1][n - 1];
    }
```

PYTHON

```
def uniquePaths(m, n):
    dp = [[1] * n for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[m-1][n-1]
```

NON TI BASTA?
ECCO ALTRI ESERCIZI

https://leetcode.com/problems/longest-unequal-adjacent-groups-subsequence-i/?envType=problem-list-v2&envId=dynamic-programming

https://leetcode.com/problems/pascals-triangle/description/?envType=problem-list-v2&envId=dynamic-programming

https://leetcode.com/problems/all-possible-full-binary-trees/description/?envType=problem-list-v2&envId=dynamic-programming

SPOILER ALERT

Bacaro
Tech

CODE AND FUN

VI RINGRAZIA TUTTI PER AVER PARTECIPATO!