

● Problema 1 – Multipli di 3 o 5

Euler

Testo

Somma tutti i naturali < 1000 multipli di 3 o 5.

Idea (dal PDF)

Scorro da 1 a 999 e sommo solo quelli divisibili per 3 o 5.

```
p1 = sum(n for n in range(1000) if n % 3 == 0 or n % 5 == 0)
print(p1) # 233168
```

Risultato: 233168

Spiegazione per l'esame (approfondita)

Concetto chiave

È un problema di base su **condizioni aritmetiche** (divisibilità) e **iterazione**. Serve a verificare che sai:

- usare il modulo (%) per testare la divisibilità;
- filtrare elementi in un intervallo;
- accumulare una somma.

Divisibilità

Un numero n è multiplo di 3 se $n \% 3 == 0$, multiplo di 5 se $n \% 5 == 0$.

La condizione `if n % 3 == 0 or n % 5 == 0` include:

- multipli di 3,
- multipli di 5,
- quelli che sono multipli di entrambi (es. 15, 30, ...), ma vengono comunque contati una sola volta perché ogni n è considerato al massimo una volta nel ciclo.

Aspetto algoritmico

- Complessità temporale: $O(N)$, con $N=1000 \rightarrow$ trascurabile.
- Complessità spaziale: $O(1)$, usa solo accumulatori.

Possibili varianti “da orale”

- Trovare una **formula chiusa** usando progressioni aritmetiche (somma multipli di 3 + somma multipli di 5 – somma multipli di 15).
- Confrontare soluzione iterativa vs soluzione matematica: perché quella con la formula è $O(1)$?

Domande che possono farti

- Spiega cosa fa `n % 3 == 0`.
- Cosa cambia tra `or` e `and` qui?

- Come lo generalizzeresti a “multipli di 3 o 5 sotto 1 miliardo” (discorso su efficienza)?
-

● Problema 2 – Somma dei Fibonacci pari

Euler

Testo

Sequenza di Fibonacci, somma dei termini pari < 4.000.000.

Idea

Genero Fibonacci iterativamente, sommo solo i pari fino al limite.

```
a, b = 1, 2 # primi 2 termini
p2 = 0
while a < 4_000_000:
    if a % 2 == 0:
        p2 += a
    a, b = b, a + b # passo successivo
print(p2) # 4613732
```

Risultato: 4613732

Spiegazione per l'esame

Concetto chiave

La sequenza di Fibonacci è definita ricorsivamente:

- $F_1 = 1, F_2 = 2$ (in questa variante),
- $F_n = F_{n-1} + F_{n-2}$ per $n \geq 3$.

Qui viene testata:

- la capacità di **generare una sequenza ricorsiva in modo iterativo**;
- la gestione di un **vincolo di soglia** (valore < 4 milioni);
- il filtraggio per **parità**.

Perché iterativo e non ricorsivo?

- La versione ricorsiva “naïve” ha complessità esponenziale $O(\varphi^n)$.
- La versione iterativa è $O(k)$, con $k =$ numero di termini generati (molto più efficiente).

Osservazione teorica interessante (che puoi citare)

Nella normale sequenza di Fibonacci, i termini pari compaiono con periodicità: **ogni 3 numeri** uno è pari. Si può sfruttare per ottimizzare ancora (generando direttamente solo i pari), ma qui non è necessario per il limite dato.

Domande che possono farti

- Definisci la sequenza di Fibonacci formalmente.

- Perché la ricorsione basica su Fibonacci è inefficiente?
 - Complessità della soluzione proposta?
 - Come modificheresti il codice se la soglia fosse in numero di termini e non in valore (es: primi 100 numeri)?
-

● Problema 3 – Massimo fattore primo

Euler

Testo

Massimo fattore primo di 600851475143.

Idea

Divido progressivamente per i piccoli fattori primi, riducendo il numero.

```
def largest_prime_factor(n: int) -> int:  
    d = 2  
    while d * d <= n:           # mi fermo alla radice  
        if n % d == 0:  
            n //= d             # divido via tutti i fattori d  
        else:  
            d += 1 if d == 2 else 2 # dopo 2, salto ai dispari  
    return n                     # quello che resta è primo  
  
p3 = largest_prime_factor(600851475143)  
print(p3) # 6857
```

Risultato: 6857

Spiegazione per l'esame

Concetto chiave

Questo problema testa la **fattorizzazione** di un numero intero e l'uso di una proprietà fondamentale:

Se n non ha divisori primi $\leq \sqrt{n}$, allora n è primo.

Idea algoritmica

1. Parti da $d = 2$ (primo fattore possibile).
2. Finché $d \cdot d \leq n$:
 - Se d divide n , “togli” il fattore: $n // d$.
 - Altrimenti incrementa d (saltando i pari dopo 2).
3. Quando esci dal ciclo, il valore residuo di n è il **massimo fattore primo**.

Perché ci si ferma a \sqrt{n} ?

Se n fosse composto, potrebbe essere scritto come $n = a \cdot b$.

Se entrambi a e b fossero $> \sqrt{n}$, il prodotto sarebbe $> n$, impossibile.

Quindi almeno uno dei due è $\leq \sqrt{n}$: se non trovi nessun fattore $\leq \sqrt{n}$, n è primo.

Ottimizzazione

- Dopo aver gestito $d = 2$, i fattori pari sono esauriti → si può passare solo ai dispari ($d \neq 2$).
- Complessità $\sim O(\sqrt{n})$ nel caso peggiore.

Domande da orale

- Spiega perché è sufficiente cercare divisori fino a \sqrt{n} .
 - Differenza tra fattorizzazione completa e ricerca del massimo fattore primo.
 - Come cambierebbe l'algoritmo per restituire *tutti* i fattori primi?
-

● Problema 4 – Palindromo con due numeri a 3 cifre

Euler

Testo

Maggiore palindromo come prodotto di due numeri a 3 cifre.

Idea

Provo tutti i prodotti 100–999 e tengo il massimo palindromo.

```
def is_pal(n: int) -> bool:  
    s = str(n)  
    return s == s[::-1]  
  
p4 = 0  
for a in range(100, 1000):  
    for b in range(100, 1000):  
        prod = a * b  
        if is_pal(prod) and prod > p4:  
            p4 = prod  
print(p4) # 906609
```

Risultato: 906609

Spiegazione per l'esame

Concetto chiave

- Rappresentazione dei numeri come stringhe.
- Definizione di **palindromo**.
- Ricerca esaustiva (brute force) nello spazio delle soluzioni.

Verifica di palindromicità

- Converte il numero in stringa.
- Usa slicing $s[::-1]$ per ottenere la stringa inversa.
- Confronta.

Spazio di ricerca

- Coppie (a, b) con $100 \leq a, b \leq 999 \rightarrow$ circa $900 \times 900 \approx 810.000$ combinazioni.
- Complessità $O(N^2)$ sul range (qui accettabile).

Possibili ottimizzazioni

- Iterare da 999 verso 100 e usare “early stopping” in alcuni casi.
- Sfruttare simmetrie ($ab = ba$) e limitare il range interno $a < b \geq a$.

Domande da orale

- Cos’è un palindromo? Fai esempi numerici e di stringhe.
 - Complessità dell’algoritmo? È ottimale?
 - Come limiteresti il numero di moltiplicazioni?
-

● Problema 5 – Minimo comune multiplo 1..20

Euler

Testo

Minimo numero positivo divisibile per tutti i numeri da 1 a 20.

Idea

Calcolo l’LCM (least common multiple) iterato da 1 a 20.

```
import math
from functools import reduce

def lcm(a: int, b: int) -> int:
    return abs(a * b) // math.gcd(a, b)

p5 = reduce(lcm, range(1, 21))
print(p5) # 232792560
```

Risultato: 232792560

Spiegazione per l’esame

Concetto chiave

- Definizione di **MCD** (GCD) e **mcm** (LCM).
- Relazione fondamentale:

$$\text{LCM}(a, b) = \frac{|a \cdot b|}{\text{GCD}(a, b)}$$

Perché LCM iterato?

LCM è associativo:

$$\text{LCM}(1, \dots, 20) = \text{LCM}(\dots (\text{LCM}(\text{LCM}(1,2),3), \dots, 20)$$

Quindi puoi applicare la funzione due-argomenti iterativamente con `reduce`.

Uso di `math.gcd`

- Implementa l'algoritmo di Euclide (molto efficiente, $O(\log(\min(a,b)))$).
- L'approccio evita fattorizzazioni esplicite.

Domande da orale

- Dimostra la formula $\text{LCM}(a, b) = |ab| / \text{GCD}(a, b)$.
 - Spiega come funziona l'algoritmo di Euclide.
 - Perché è più efficiente usare GCD rispetto a scomporre ogni numero in fattori primi?
-

● Problema 6 – Somma quadrati vs quadrato somma

Euler

Testo

$$(1^2 + \dots + 100^2) - (1 + \dots + 100)^2$$

Idea

Calcolo le due somme e faccio la differenza.

```
sum_ = sum(range(1, 101))
sum_sq = sum(n*n for n in range(1, 101))
p6 = sum_*2 - sum_sq
print(p6) # 25164150
```

Risultato: 25164150

Spiegazione per l'esame

Concetto chiave

Confronto tra:

- **Quadrato della somma:** $(\sum_{k=1}^n k)^2$
- **Somma dei quadrati:** $\sum_{k=1}^n k^2$

Per $n = 100$.

Formule note (puoi citarle a memoria)

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$$

Per l'esame, potresti anche risolverlo **senza** programmazione usando queste formule.

Interpretazione

La differenza rappresenta una misura di quanto il “peso” dei termini grandi influisce quando li sommi prima e poi elevi al quadrato, rispetto a sommare i quadrati uno a uno.

Domande da orale

- Scrivi e giustifica la formula della somma dei primi n interi.
 - Idem per la somma dei quadrati.
 - Perché la differenza non è zero? C’è un’interpretazione geometrica?
-

● Problema 7 – 10001º numero primo

Euler

Testo / Idea (riassunto dal PDF)

Usare una funzione `is_prime` e contare quanti numeri primi troviamo finché non arrivi al 10001-esimo.

```
def is_prime(n: int) -> bool:
    if n < 2:
        return False
    if n % 2 == 0:
        return n == 2
    r = int(n**0.5)
    for d in range(3, r+1, 2):
        if n % d == 0:
            return False
    return True

count = 0
n = 1
while count < 10001:
    n += 1
    if is_prime(n):
        count += 1
p7 = n
print(p7) # 104743
```

Risultato: 104743

Spiegazione per l'esame

Concetto chiave

- Definizione di numero primo.
- Test di primalità efficiente (fino a \sqrt{n} , salti i pari).
- Iterazione fino al k-esimo primo.

Dettaglio sul test di primalità

- Escludi rapidamente i casi banali ($n < 2$, pari > 2).
- Controlli solo i divisori dispari fino a \sqrt{n} .
- Complessità: $O(\sqrt{n})$ per ogni test.

Complessità complessiva

Non esiste formula esatta per il k-esimo primo, ma approssimativamente:

$$p_k \sim k \log k$$

Quindi il numero massimo testato è circa dell'ordine di 10^5 per il 10001-esimo primo.

Domande da orale

- Differenza tra test di primalità deterministico e probabilistico (anche solo concettuale, tipo Miller–Rabin).
- Perché è sufficiente arrivare fino a \sqrt{n} per testare se un numero è primo?
- Come velocizzeresti la ricerca di molti primi (es. Sieve di Eratostene) rispetto a testare ogni numero singolarmente?

● Problema 8 – Prodotto massimo di 13 cifre adiacenti

Euler

Testo

Nel famoso numero a 1000 cifre, prodotto massimo di 13 cifre consecutive.

Idea

Scorro tutte le finestre da 13 cifre e calcolo il prodotto.

```
num_str = (
    "73167176531330624919225119674426574742355349194934"
    ...
    "71636269561882670428252483600823257530420752963450"
)
max_prod = 0
for i in range(len(num_str) - 13 + 1):
    prod = 1
    for c in num_str[i:i+13]:
        prod *= int(c)
    if prod > max_prod:
        max_prod = prod
```

```
p8 = max_prod
print(p8) # 23514624000
```

Risultato: 23514624000

Spiegazione per l'esame

Concetto chiave

- Tecnica della **finestra scorrevole** (“sliding window”).
- Conversione carattere → cifra (`int(c)`).
- Computazione di un **prodotto** su un sottoinsieme di valori.

Aspetto algoritmico

- Lunghezza numero: 1000 cifre.
- Numero di finestre di lunghezza 13: $1000 - 13 + 1 = 988$.
- Ogni finestra richiede un prodotto di 13 numeri → $O(13 \times 988) \approx O(10^4)$, trascurabile.

Possibili ottimizzazioni “da teoria”

- Se una finestra contiene uno zero, il prodotto è zero → puoi saltare alcune operazioni.
- Sliding window “vera”: invece di ricalcolare da zero, dividi per la cifra che esce e moltipichi quella che entra, ma qui la presenza di zeri la rende più complicata.

Domande da orale

- Spiega la tecnica della sliding window con un esempio semplice (somma di 3 elementi consecutivi).
 - Come tratteresti gli zeri per velocizzare l'algoritmo?
 - Complessità in funzione della lunghezza N e della finestra k.
-

● Problema 9 – Tripla pitagorica speciale

Euler

Testo

Trova a, b, c con $a^2+b^2=c^2$ e $a+b+c=1000$. Restituisci $a \cdot b \cdot c$.

Idea

Brute force vincolato da $a+b+c=1000$.

```
p9 = None
for a in range(1, 1000):
    for b in range(a+1, 1000):
        c = 1000 - a - b
        if a*a + b*b == c*c:
            p9 = a * b * c
            break
```

```

if p9 is not None:
    break
print(p9) # 31875000

```

Risultato: 31875000

Spiegazione per l'esame

Concetto chiave

- Applicazione del **teorema di Pitagora** a terne (a,b,c).
- Uso di un **vincolo lineare** $a + b + c = 1000$ per ridurre lo spazio di ricerca.

Riduzione dello spazio di ricerca

- Senza il vincolo, la ricerca sarebbe su tutte le terne $(a,b,c) \leq 1000 \rightarrow O(N^3)$.
- Con $c = 1000 - a - b$, riduci a due cicli annidati (a, b) $\rightarrow O(N^2)$.
- Inoltre, imponendo $b > a$ riduci ulteriormente le combinazioni simmetriche.

Osservazione teorica (se vuoi fare scena all'esame)

Le terne pitagoriche primitive possono essere generate con formule parametriche:

$$a = m^2 - n^2, b = 2mn, c = m^2 + n^2$$

per interi $m > n$.

Si potrebbe tentare un approccio più matematico per trovare (a,b,c) tali che la somma sia 1000, ma il brute force è già sufficiente per un computer.

Domande da orale

- Cos'è una tripla pitagorica?
- Spiega perché il vincolo $a + b + c = 1000$ riduce la complessità.
- Potresti evitare del tutto i cicli annidati usando le formule parametriche?

● Problema 10 – Somma dei primi < 2.000.000

Euler

Idea

Sieve di Eratostene fino a 2 milioni, poi sommo.

```

limit = 2_000_000
sieve = [True] * limit
sieve[0] = sieve[1] = False
for i in range(2, int(limit**0.5) + 1):
    if sieve[i]:
        for j in range(i*i, limit, i):
            sieve[j] = False

```

```
p10 = sum(i for i, is_p in enumerate(sieve) if is_p)
print(p10) # 142913828922
```

Risultato: 142913828922

Spiegazione per l'esame

Concetto chiave

- Implementazione della **Sieve di Eratostene**.
- Identificazione di tutti i numeri primi $\leq N$ in **tempo quasi lineare**.

Come funziona la sieve

1. Crea un array booleano `sieve[0..limit-1]`, inizialmente tutti `True`.
2. Imposta 0 e 1 a `False` (non primi).
3. Per ogni i da 2 a \sqrt{N} :
 - Se `sieve[i]` è `True`, marca tutti i multipli di i (a partire da i^2) come `False`.
4. Alla fine, gli indici con `True` sono i numeri primi.

Perché partire da i^2 ?

- I multipli $< i^2$ sono già stati segnati usando fattori più piccoli.

Complessità

- Tempo: $O(N \log \log N)$, molto più efficiente di testare ogni numero con \sqrt{n} .
- Spazio: $O(N)$ per l'array booleano.

Domande da orale

- Spiega la differenza tra `sieve` e test di primalità individuale.
- Perché la complessità è $O(N \log \log N)$? (almeno a livello qualitativo).
- Come modificheresti il codice per restituire la lista dei primi invece della loro somma?

● PROBLEMA 11 — Prodotto massimo in griglia 20×20

Euler

Testo

Nel grid 20×20 dato, massimo prodotto di 4 numeri adiacenti (orizz, vert, diagonali).

Idea

Scorro ogni cella e controllo 4 direzioni (dove possibile).

(Segue il lungo blocco di codice riportato nel PDF; non lo modifco)

```
# [CODICE INTEGRALE DA PDF – mantenuto identico]
```

```

grid = [
[8,2,22,97,38,15,0,40,0,75,4,5,7,78,52,12,50,77,91,8],
[49,49,99,40,17,81,18,57,60,87,17,40,98,43,69,48,4,56,62,0],
...
[1,70,54,71,83,51,54,69,16,92,33,48,61,43,52,1,89,19,67,48]
]
max_prod = 0
rows, cols = 20, 20
for r in range(rows):
    for c in range(cols):
        # destra
        ...
p11 = max_prod
print(p11) # 70600674

```

Risultato: 70600674

Spiegazione approfondita (stile universitario)

Questo problema valuta:

- capacità di lavorare su **matrici bidimensionali**,
- gestione di **indirizzamento righe/colonne**,
- manipolazione di offset direzionali,
- attenzione ai **limiti dell'indice**.

Struttura concettuale

La griglia 20×20 è rappresentata come una lista di liste. Ogni cella (r, c) può essere l'origine di 4 sequenze di 4 elementi:

1. **orizzontale a destra:** $(r, c), (r, c+1), \dots$
2. **verticale verso il basso:** $(r, c), (r+1, c), \dots$
3. **diagonale basso-destra**
4. **diagonale alto-destra**

Il punto chiave è **controllare che gli indici non escano dai limiti**.

Complessità e valutazione algoritmica

- Scorri tutte le 400 celle.
- Per ognuna test 4 direzioni \rightarrow complessità $O(400 \times 4) = O(1)$ per il problema dato.
- In generale: $O(N^2)$ per una griglia NxN.

Osservazione teorica

Il prodotto può essere visto come una forma elementare di **convoluzione direzionale**: ogni direzione definisce un “kernel” 1D applicato alla griglia.

Domande che possono uscire all’orale

- Come eviti l'accesso fuori dai limiti?
 - Perché parti solo da 4 direzioni e non 8?
 - La soluzione è scalabile per matrici molto più grandi?
 - Come generalizzi per una finestra di lunghezza k?
-

● PROBLEMA 12 — Numero triangolare con >500 divisori

Euler

Idea

Genero numeri triangolari $n(n+1)/2$ e per ognuno conto i divisori (tramite fattorizzazione).

(Codice completo come nel PDF)

Risultato: 76576500

■ Spiegazione approfondita

🔍 Numeri triangolari

Un numero triangolare T_n è:

$$T_n = \frac{n(n + 1)}{2}$$

Sono numeri con struttura moltiplicativa interessante perché **n e n+1 sono coprimi** (non condividono fattori).

🔍 Conteggio dei divisori

Per contare i divisori si usa la proprietà fondamentale:

Se

$$n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$$

allora il numero totale di divisori è:

$$d(n) = (a_1 + 1)(a_2 + 1) \dots (a_k + 1)$$

Il codice esegue una **fattorizzazione tramite enumerazione di divisori fino a \sqrt{n}** .

Osservazione interessante per l'esame

Il fatto che $T_n = n(n+1)/2$ implica che la fattorizzazione può essere ottenuta fattorizzando separatamente n e $n+1$.

Domande possibili

- Spiega perché n e $n+1$ sono coprimi.
 - Quanti divisori ha un numero come $2^4 \cdot 3^2$?
 - Complessità dell'algoritmo di fattorizzazione usato?
 - Come ottimizzeresti la ricerca del primo triangolare > 500 divisori?
-

PROBLEMA 13 — Somma di 100 numeri da 50 cifre

Euler

Testo

Somma 100 numeri di 50 cifre e prendi le prime 10 cifre del risultato.

Idea

Leggo le stringhe, converto in int e sommo.

Risultato: 5537376230

Spiegazione approfondita

Perché questo problema è significativo?

Perché richiede manipolazione di **interi arbitrariamente grandi**, non gestibili con i tipi primitivi del C o del Java standard.

Python, invece:

- implementa gli interi come **big integers**,
- usa una rappresentazione interna basata su word a 30–60 bit,
- permette somme di numeri enormi in $O(n)$ sulla lunghezza delle cifre.

Perché basta prendere le prime 10 cifre?

La somma di 100 numeri da 50 cifre dà un numero al massimo da 52 cifre (per via dei riporto a cascata).

Domande possibili

- Come Python rappresenta i big integers?
 - Qual è la complessità della somma di due numeri molto grandi?
 - Perché è più semplice lavorare con stringhe in input e non in file binari?
-



PROBLEMA 14 — Sequenza di Collatz

Euler

Testo

Per $n < 1.000.000$, trova n che genera la sequenza di Collatz più lunga.

Idea

Uso memoization: salvo la lunghezza già calcolata delle catene.

Risultato: 837799



Spiegazione approfondita

Collatz: definizione

$$C(n) = \begin{cases} n/2 & \text{se } n \text{ è pari} \\ 3n + 1 & \text{se } n \text{ è dispari} \end{cases}$$

Iteri finché non arrivi a 1.

La congettura afferma che **ogni intero positivo** alla fine arriva a 1, ma non è stato dimostrato.

Aspetto computazionale

Calcolare la lunghezza di tutte le sequenze fino a 1 milione senza ottimizzazioni è molto lento.

La memoization riduce drasticamente il tempo:

- Se durante la catena arrivi a un valore già noto, puoi “saltare” tutto il resto.



Nota teorica

Questo è un esempio di **programmazione dinamica top-down**.

🎤 Domande da esame

- Cos'è una memoization?
 - Spiega la differenza tra top-down e bottom-up.
 - Qual è il rischio dell'overflow in linguaggi senza big integers?
-

● PROBLEMA 15 — Percorsi in griglia 20×20

Euler

Idea

È un binomiale: seleziono 20 passi giù tra 40 → $C(40,20)$.

Risultato: 137846528820

📘 Spiegazione approfondita

🔍 Interpretazione combinatoria

Per andare da (0,0) a (20,20) servono **40 passi**:

- 20 verso destra,
- 20 verso il basso.

La domanda diventa:

In quante sequenze di 40 simboli posso inserire 20 “D” (down) e 20 “R” (right)?

Risposta:

$$\binom{40}{20}$$

🔍 Differenza tra soluzione combinatoria e DP

Soluzione alternativa:

$$dp[r][c] = dp[r-1][c] + dp[r][c-1]$$

Grandezza della matrice: 21×21 .

Domande possibili

- Deriva combinatoriamente il coefficiente binomiale.
 - Spiega perché DP e combinatoria danno lo stesso risultato.
 - Complessità delle due soluzioni.
-



PROBLEMA 16 — Somma cifre di 2^{1000}

Euler

Idea

Calcolo 2^{1000} e sommo le cifre.

Risultato: 1366



Spiegazione approfondita

Perché 2^{1000} è interessante?

2^{1000} ha circa:

$$1000 \cdot \log_{10}(2) \approx 301 \text{ cifre}$$

È un numero enorme ma gestibile con big integers.

Algoritmi rilevanti

- l'operazione di **esponenziazione veloce** in Python usa “exponentiation by squaring”.
- la conversione a stringa costa $O(n)$ sulle cifre.

Domande possibili

- Complessità dell'esponenziazione iterativa vs per esponenti grandi.
 - Perché è necessario convertire in stringa per ottenere le cifre?
-

● PROBLEMA 17 — Numeri scritti in inglese

Euler

Idea

Funzione che scrive il numero in inglese (“onehundredand...”), poi si sommano le lunghezze.

Risultato: 21124

📘 Spiegazione approfondita

🔍 Perché è un problema linguistico + algoritmico?

Bisogna implementare:

- la regola inglese di “hundred and”,
- mapping lessicale dei numeri.

🔍 Difficoltà concettuali

- trattamento delle decine irregolari (11...19);
- concatenazione senza spazi né trattini;
- struttura grammaticale:
 - 342 → "threehundredandfortytwo".

🧠 Osservazione

È un esempio di **trasposizione di regole linguistiche in funzione deterministica**.

🎤 Domande possibili

- Spiega come hai gestito le eccezioni linguistiche.
 - Perché la soluzione è $O(N)$ sul numero di parole generate?
-

● PROBLEMA 18 — Percorso massimo in triangolo

Euler

Idea

Programmazione dinamica bottom-up.

Risultato: 1074

Spiegazione approfondita

Algoritmo

Si parte dall'ultima riga e si “propaga” il massimo verso l'alto:

$$tri[r][c] = tri[r][c] + \max (tri[r + 1][c], tri[r + 1][c + 1])$$

Perché bottom-up?

- ogni cella dipende solo dai due figli,
- la struttura è un DAG (Directed Acyclic Graph),
- nessun bisogno di memoization.

Domande possibili

- Differenza tra top-down e bottom-up.
 - Complessità: $O(N^2)$ dove N è lato del triangolo.
-

PROBLEMA 19 — Quante domeniche cadono il primo del mese?

Euler

Idea

Usare una formula di giorno della settimana.

Risultato: 171

Spiegazione approfondita

Algoritmo

La formula usata è una variante del **Sakamoto's algorithm**, che calcola il giorno della settimana tramite aritmetica modulare.

🔍 Perché funziona?

Viene utilizzata una sequenza di offset mensili che trasforma la data (anno, mese, giorno) in un indice modulo 7.

🎤 Domande possibili

- Spiega cos'è l'aritmetica modulare.
 - Qual è il significato di $y + y//4 - y//100 + y//400$?
 - Differenze tra calendario gregoriano e giuliano?
-

● PROBLEMA 20 — Somma cifre di 100!

Euler

Idea

Calcolo $100!$ e sommo le cifre.

Risultato: 648

📘 Spiegazione approfondita

🔍 Caratteristica interessante

$100!$ è un numero enorme (158 cifre).

Il problema valuta:

- capacità di usare big integers,
- manipolazione delle cifre,
- assenza di overflow.

🔍 Complessità

Il calcolo di `factorial(n)` è $O(n^2 \log n)$ con big integers.

🎤 Domande possibili

- Complessità di factorial con numeri arbitrari?
- Perché la conversione a stringa è $O(k)$?

● PROBLEMA 21 — Numeri amici sotto 10000

Euler

Definizione (dal PDF)

$d(n)$ = somma divisori propri.

a e b formano una coppia amica se $d(a) = b$, $d(b) = a$ e $a \neq b$.

Idea

Calcolo $d(n)$ per 1..9999, cerco coppie amiche e sommo a.

```
def d(n: int) -> int:
    return sum(i for i in range(1, n//2 + 1) if n % i == 0)

p21 = 0
for a in range(1, 10000):
    b = d(a)
    if b != a and d(b) == a:
        p21 += a
print(p21) # 31626
```

Risultato: 31626



Spiegazione approfondita (stile universitario)



Concetto matematico

Due numeri a e b sono **amici** se:

- la somma dei divisori propri di a restituisce b ,
- la somma dei divisori propri di b restituisce a ,
- ma $a \neq b$.

Esempio classico: **220** e **284**.

Questo problema esplora:

- **funzione somma dei divisori** → aritmetica moltiplicativa
- **numeri abbondanti/perfetti/amici** → teoria classica dei numeri
- caratteristiche del dominio 1–10000.



Complessità

Il calcolo di $d(n)$ è $O(n)$ nella versione naïve, perché scandisce tutti i possibili divisori.

Totale → $O(n^2) \approx 10^8$ operazioni → Python riesce comunque a gestirlo.

Ottimizzazioni possibili:

- fermarsi a \sqrt{n} ,
- memorizzare $d(n)$ per tutti gli n (memoization),
- utilizzare proprietà moltiplicative dei divisori.

Domande da orale

- Definisci divisore proprio.
 - Spiega il concetto di funzione aritmetica.
 - Come ridurresti la complessità dell'algoritmo?
 - Differenza tra numeri perfetti e numeri amici.
-

PROBLEMA 22 — Name Scores

Euler

Testo (dal PDF)

Ogni nome ha uno “score”: (posizione nella lista) \times (somma valori lettere A=1, B=2...). Somma di tutti gli score.

Idea

Leggo il file, ordino i nomi, calcolo punteggio.

```
def name_value(name: str) -> int:  
    return sum(ord(c) - ord('A') + 1 for c in name)  
  
total = 0  
for i, name in enumerate(names, start=1):  
    total += i * name_value(name)  
p22 = total  
print(p22) # 871198282
```

Risultato: 871198282

Spiegazione approfondita

Aspetto algoritmico

- Il file ufficiale contiene migliaia di nomi.
- Ordinare alfabeticamente $\rightarrow O(n \log n)$.
- Calcolo dei punteggi $\rightarrow O(n \times \text{lunghezza_nome})$.

Uso dell'encoding ASCII

La formula:

$$\text{valore}(c) = \text{ord}(c) - \text{ord}('A') + 1$$

trasforma 'A'→1, 'B'→2 ... 'Z'→26.

Potenziali trabocchetti

- Presenza di virgolette nel file originale.
- Case sensitivity (tutto maiuscolo).
- Differenza tra posizione nella lista (1-based) e indice Python (0-based).

Domande da orale

- Complessità totale dell'algoritmo?
 - Perché l'ordinamento è $O(n \log n)$?
 - Qual è la differenza tra ASCII e Unicode?
 - Come gestiresti nomi con caratteri accentati?
-

PROBLEMA 23 — Numeri NON rappresentabili come somma di due abbondanti

Euler

Definizione

Numeri abbondanti: somma divisori propri > n.

Idea

1. Trovo tutti gli abbondanti ≤ 28123 .
2. Genero tutte le somme tra due abbondanti.
3. Sommo i numeri che NON appaiono come tali somme.

Risultato: 4179871

(Codice identico nel PDF)

Spiegazione approfondita

Concetto matematico

Un numero è:

- **deficiente** se $d(n) < n$
- **perfetto** se $d(n) = n$
- **abbondante** se $d(n) > n$

La congettura nota:

Ogni numero > 28123 dovrebbe essere rappresentabile come somma di due abbondanti.

Non è dimostrata, ma è empiricamente verificata per il range considerato.

Aspetto algoritmico

1. Calcolo della lista abbondanti → richiede molte chiamate a “somma divisori”.
2. Combinazioni abbondante+abbondante → numero quadratico di coppie.
3. Uso di un **set** → membership test $O(1)$.

Osservazione importante

Il limite **28123** deriva da studi analitici sulle proprietà dei numeri abbondanti.

Domande da orale

- Qual è la differenza concettuale tra numeri perfetti/abbondanti/deficienti?
- Perché viene usato un set e non una lista?
- Spiega la complessità $O(k^2)$ della combinazione abbondanti+abbondanti.
- Esiste un modo più efficiente per verificare la condizione?

PROBLEMA 24 — Milionesima permutazione lessicografica di 0–9

Euler

Idea

Uso `itertools.permutations` e prendo la permutazione numero 1.000.000.

```
perm = next(itertools.islice(itertools.permutations("0123456789"), 999999,
None))
print(p24) # 2783915460
```

Risultato: 2783915460

Spiegazione approfondita

Concetto matematico

Le permutazioni lessicografiche sono ordinamenti delle cifre secondo l'ordine alfabetico dei simboli.

Numero totale permutazioni $\rightarrow 10! = 3.628.800$.

La milionesima è quindi ben interna.

Osservazione matematica importante

Si può calcolare la milionesima permutazione **senza generarle tutte**, usando la decomposizione fattoriale:

$$n\text{-esima permutazione} = \lfloor \frac{n}{9!} \rfloor, \lfloor \frac{n \bmod 9!}{8!} \rfloor, \dots$$

Questo riduce la complessità da $O(n!)$ a $O(n^2)$.

Domande da orale

- Spiega la formula per ottenere direttamente la n-esima permutazione.
 - Qual è la complessità di `itertools.permutations`?
 - Quante permutazioni ha una stringa di lunghezza k con caratteri distinti?
-

PROBLEMA 25 — Primo Fibonacci con 1000 cifre

Euler

Idea

Genero Fibonacci finché un termine non ha 1000 cifre.

Risultato: 4782

(Codice identico al PDF)

Spiegazione approfondita

Concetto matematico

La lunghezza in cifre di un numero n è:

$$\lfloor \log_{10}(n) \rfloor + 1$$

Il numero F_n di Fibonacci cresce asintoticamente come:

$$F_n \approx \frac{\varphi^n}{\sqrt{5}}$$

dove

$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

Da cui:

$$n \approx \frac{(\text{cifre} - 1) + \log_{10} \sqrt{5}}{\log_{10} \varphi}$$

Per 1000 cifre, il valore teorico ≈ 4782 , confermato dal codice.

Aspetto computazionale

Ogni somma di Fibonacci su big integers costa $O(d)$, dove d è il numero delle cifre.

Domande da orale

- Deriva la formula asintotica di Fibonacci con il metodo di Binet.
- Perché Python è adatto a manipolare numeri enormi?
- Complessità totale dell'algoritmo?

PROBLEMA 26 — Periodo decimale più lungo per $1/d$

Euler

Idea

Simulo la divisione lunga tracciando i resti; il periodo è quando un resto si ripete.

Risultato: 983

(Codice integrale presente nel PDF)

Spiegazione approfondita

Matematica alla base

Per una frazione:

$$\frac{1}{d}$$

il periodo decimale è correlato all'ordine di 10 modulo d:

$$\text{periodo} = \min \{k \mid 10^k \equiv 1 \pmod{d}\}$$

se **d** è **primo** e non divide 10.

Per d non primo, la situazione è più complessa.

Cosa fa il codice

- Simula la divisione lunga: ogni passo genera un resto.
- Quando un resto si ripete → ciclo trovato → lunghezza = posizione2 – posizione1.

Perché 983?

983 è un primo che genera un periodo molto lungo vicino al massimo teorico di $d-1$.

Domande da orale

- Spiega perché i periodi decimali dipendono dai resti della divisione modulare.
- Cos'è l'ordine moltiplicativo?
- Quando una frazione ha espansione finita e quando periodica?

PROBLEMA 27 — Quadratica che genera più primi

Euler

Testo

Formula:

$$f(n) = n^2 + an + b, |a| < 1000, \quad |b| \leq 1000$$

Trovare i valori di a e b che generano la più lunga sequenza di valori primi consecutivi per n = 0,1,2,...

Risultato: -59231

Spiegazione approfondita

Concetto matematico

Per n = 0:

$$f(0) = b$$

→ quindi **b deve essere primo**, altrimenti la sequenza non inizia nemmeno.

Per n = 1:

$$f(1) = 1 + a + b$$

→ spesso usato per filtrare i valori di a.

Ricerca brute-force

Si provano tutte le coppie (a,b).

Per ciascuna si calcola la lunghezza della sequenza di primi generata.

Relazione con il polinomio di Euler

Il più celebre della storia:

$$n^2 + n + 41$$

produce 40 primi consecutivi (n = 0...39).

Il problema generalizza questo concetto.

Domande da orale

- Perché b deve essere primo?
 - Dimostra perché un polinomio non può generare solo numeri primi.
 - Come puoi ridurre lo spazio di ricerca per a e b?
-

● PROBLEMA 28 — Somma diagonali spirale 1001×1001

Euler

Idea

Per ogni “anello” k, gli angoli sono:

$$\begin{array}{ll} (2k+1)^2, & (2k+1)^2 \\ -2k, & (2k+1)^2 \\ -4k, & (2k+1)^2 - 6k \end{array}$$

Risultato: 669171001

■ Spiegazione approfondita

🔍 Costruzione concettuale

La spirale parte dal centro (1) e cresce a strati concentrici.

Per ogni strato k:

- la dimensione del lato è $(2k+1)$
- l'angolo in alto a destra è $(2k+1)^2$

Gli altri angoli si ottengono sottraendo multipli di $2k$.

🔍 Somma totale

Si sommano gli angoli per ogni k fino a k=500 (per la dimensione 1001×1001).

Non serve costruire la matrice → O(n).

🎤 Domande da orale

- Spiega perché il numero nell'angolo in alto a destra dello strato k è $(2k+1)^2$.
 - Da dove proviene la sequenza delle sottrazioni?
 - Complessità della soluzione?
-

● PROBLEMA 29 — Potenze distinte a^b ($2 \leq a,b \leq 100$)

Euler

Idea

Uso un set per registrare i valori unici di a^b .

Risultato: 9183

📘 Spiegazione approfondita

🔍 Concetto chiave

Si analizzano tutte le combinazioni (a,b) con $2 \leq a,b \leq 100$.

Numero teorico combinazioni $\rightarrow 99 \times 99 = 9801$.

Molte però generano duplicati (es. $4^3 = 2^6$).

🔍 Perché usare un set?

- per rimuovere automaticamente duplicati,
- per sfruttare membership test $O(1)$.

🔍 Osservazione matematica

I duplicati derivano da diverse rappresentazioni:

$$(a^x)^y = a^{xy}$$

es.:

$$4 = 2^2 \rightarrow 4^3 = 2^6.$$

🎤 Domande da orale

- Perché i duplicati sono inevitabili?
 - Come potresti prevedere in anticipo le coppie equivalenti?
 - Complessità tempo e spazio della soluzione.
-

● PROBLEMA 30 — Somma numeri uguali alla somma delle quinte potenze delle cifre

Euler

Testo

Trovare tutti i numeri tali che:

$$n = \sum(\text{cifre}(n)^5)$$

Risultato: 443839

Spiegazione approfondita

Concetto matematico

Per un numero con d cifre, la somma massima ottenibile è:

$$d \cdot 9^5$$

Calcolando:

- $6 \times 9^5 \approx 354.294 \rightarrow$ limite superiore ragionevole.

Aspetto algoritmico

- Si iterano i numeri 2...200.000.
- Per ognuno si:
 - converte in stringa,
 - eleva ogni cifra alla quinta,
 - somma e confronta.

Perché non considerare 1?

Per convenzione, 1 non è considerato numero valido nei problemi di questo tipo (somma banale).

Domande da orale

- Perché il limite è circa 354.294?
- Che differenza c'è tra potenza delle cifre e potenza del numero?
- Complessità in funzione del numero di cifre?