Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2025/2026

Appello del 15 luglio 2026

Esercizio 1 [6 punti]

Si consideri il seguente codice Python.

```
class A:
    def __init__ (self, n:int):
        self.n=n

    def __str__ (self) -> str:
        return f"A; n = {self.n}"

    def __eq__(self, o: object) -> bool:
        if isinstance(o, A):
            return o.n==self.n
        return NotImplemented

    def __hash__(self) -> int:
        return hash(self.n)

class B(A):
    def __init__ (self, n:int):
        self.n= n * 3
```

```
class C (B, A):
    def __str__ (self) -> str:
        return f"C; n = {self.n}"

class D (A, B):
    pass

a = A(12)
b = B(4)
c = C(8)
d = D(7)

z:dict[A,int]={c:c.n, b:b.n, a:a.n}

for k in z:
    print (f"{k} -> {z[k]}")
```

Denotando la classe `object` con **O**, si calcoli la linearizzazione di tutte le classi, e si scriva in tabella il procedimento e il risultato finale:

classe	linearizzazione (MRO)
A	L(A) = A + merge (L(O), O) = A + merge (O, O) = AO
B	
C	
D	

Si scriva nel riquadro a destra cosa **esclusivamente** ciò che viene stampato a video dal codice.

Nel caso in cui il codice dia errore in qualche punto (poiché ad esempio non è possibile calcolare qualche MRO), escludere dal codice il numero minimo di linee in modo da non ottenere errori, commentandole con **#** sul listato stampato in alto su questo foglio, e quindi stampare cosa risulta dal nuovo

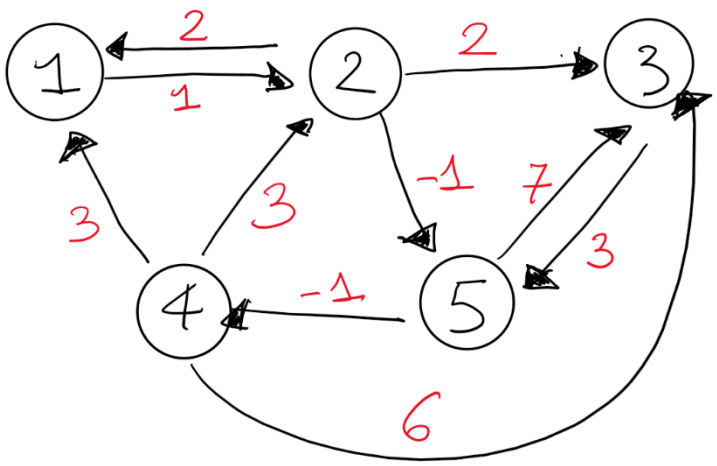
Esercizio 2 [6 punti]

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array **[70, 3, 2, 5, 9, 37, 25, 16]** in modo non crescente (compresa la fase iniziale di build-**min**-heap). *Si noti che, invece di procedere con un max-heap come visto a lezione, è necessario utilizzare un min-heap.*

Esercizio 3 [11 punti]

[2 punti] Si consideri il problema dei **cammini minimi tra tutte le coppie di nodi** in un grafo diretto pesato sugli archi, in cui compaiono anche pesi negativi. Si elenchino, tra quelli visti a lezione, i possibili algoritmi che possono essere impiegati per risolvere il problema, con associata la complessità computazionale nel caso peggiore. **Si individui pertanto l'algoritmo** che, in generale, assicura la migliore complessità computazionale nel caso peggiore.

[8 punti] Si applichi quindi tale algoritmo al grafo in figura mostrandone una possibile esecuzione **passo-passo**. Ad ogni passo, bisogna mostrare il contenuto di tutte le strutture dati utilizzate dall'algoritmo, compreso il/la vettore/matrice dei padri.



[1 punto] Si descriva il cammino minimo dal nodo **5** al nodo **3**.

Esercizio 4 [11 punti]

Tutto il codice richiesto deve essere in **Python** ed annotato con i tipi opportuni.

Nota: Se necessario, si aggiungano alle classi tutti i metodi necessari per far funzionare correttamente il codice richiesto, e si discuta quanto fatto.

Si vogliono implementare le classi per gestire diversi tipi di figurine per un album da collezione. Tutte le figurine hanno un codice identificativo e un nome, ma il loro valore dipende dal tipo di figurina.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **150** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Si assuma, senza scrivere nulla, l'esistenza di una classe astratta **Figurina** con i seguenti attributi di istanza:

- **_codice** (tipo **int, Final**)
- **_nome** (tipo **str**)

ed i seguenti metodi di istanza:

- **__init__** che inizializza un oggetto della *classe* **Figurina** assegnando codice e nome;
- proprietà (`@property`) in lettura per tutti gli attributi, con lo stesso nome (senza _);
- proprietà in scrittura per nome;
- metodo **__eq__ (self, o:object)** che restituisce **True** se **o** è una Figurina con lo stesso codice di self, (indipendentemente dal nome), **False** altrimenti;
- metodo **__hash__ (self)** che restituisce in modo opportuno l'hash di una Figurina.
- metodo astratto **calcola_valore(self)** che restituisce (come un intero) il valore in centesimi di euro della figurina.

Si assuma, senza scrivere nulla, l'esistenza di una *sottoclasse* **FigurinaNormale** che estende Figurina ed ha, inoltre, il seguente attributo:

- **_nuova** (tipo **bool**, vale True se la figurina è nuova e False se è usata)

La *classe* **FigurinaNormale** ha i seguenti metodi di istanza:

- **__init__** che inizializza un oggetto della *classe* **FigurinaNormale** prendendo in input gli opportuni parametri (incluso il booleano nuova) e richiamando opportunamente il metodo __init__ della superclasse
- proprietà (`@property`) **nuova** in lettura per l'attributo **_nuova**.
- metodo **calcola_valore(self)** che restituisce il valore della figurina, tenendo conto che una figurina nuova vale 2€ mentre una figurina usata vale 1€.

Si assuma, senza scrivere nulla, l'esistenza di una *sottoclasse* **FigurinaRara,** che estende Figurina, con i seguenti attributi di istanza:

- **_livello_rarità** (tipo **int**, valori tra 1 e 5)

ed i seguenti metodi di istanza:

- **__init__** che inizializza un oggetto della *classe* **FigurinaNormale** prendendo in input gli opportuni parametri (incluso il livello di rarità) e richiamando opportunamente il metodo __init__ della superclasse
- proprietà (`@property`) **livello_rarità** in lettura per l'attributo **_livello_rarità**.
- metodo **calcola_valore(self)** che restituisce il valore della figurina, tenendo conto che una figurina rara vale 10€ moltiplicato per il livello di rarità.

Si assuma, senza scrivere nulla, l'esistenza di una *classe* **Album** con i seguenti attributi di istanza:

- **_figurine** (**list** di **Figurina**), contenente tutte le figurine dell'album.

ed i seguenti metodi di istanza:

- **__init__** che inizializza un **Album** vuoto
- **add_figurina (self, f:Figurina)** che aggiunge la figurina f all'album corrente

[3 punti] Aggiungere alla classe **Album** un metodo di istanza
get_valore (self) -> float
che restituisce il valore in Euro dell'album. Il valore dell'album è calcolato sommando il valore di tutte le sue figurine.
Se la lista self._figurine è lunga **n**, la complessità nel caso medio (con l'uso di dizionari) deve essere ***O(n).***

[5 punti] Aggiungere alla classe **Album** un metodo di istanza
get_valore_senza_doppioni (self) -> float
che restituisce il valore in Euro dell'album, calcolato sommando il valore di tutte le figurine senza tener conto dei doppioni (in altre parole, una stesa figurina non deve essere conteggiata più volte).
Se la lista self._figurine è lunga **n**, la complessità nel caso medio (con l'uso di dizionari) deve essere ***O(n).***
Suggerimento: si usi un *set* per memorizzare le figurine presenti senza tener conto dei doppioni.

[3 punti] Aggiungere alla classe **Album** un metodo di istanza
get_occorrenze (self, f:Figurina) -> int
che restituisce il numero di occorrenze della figurina f nell'album corrente. Se la figurina non è presente, viene restituito 0.