Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2025/2026

Appello del 3 settembre 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 + 1
```

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

class D (A, C, B):
    pass

a = A(5)
b = B(4)
c = C(8)
d = D(7)

z:set[A]={a, b, c}

for k in z:
    print(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]

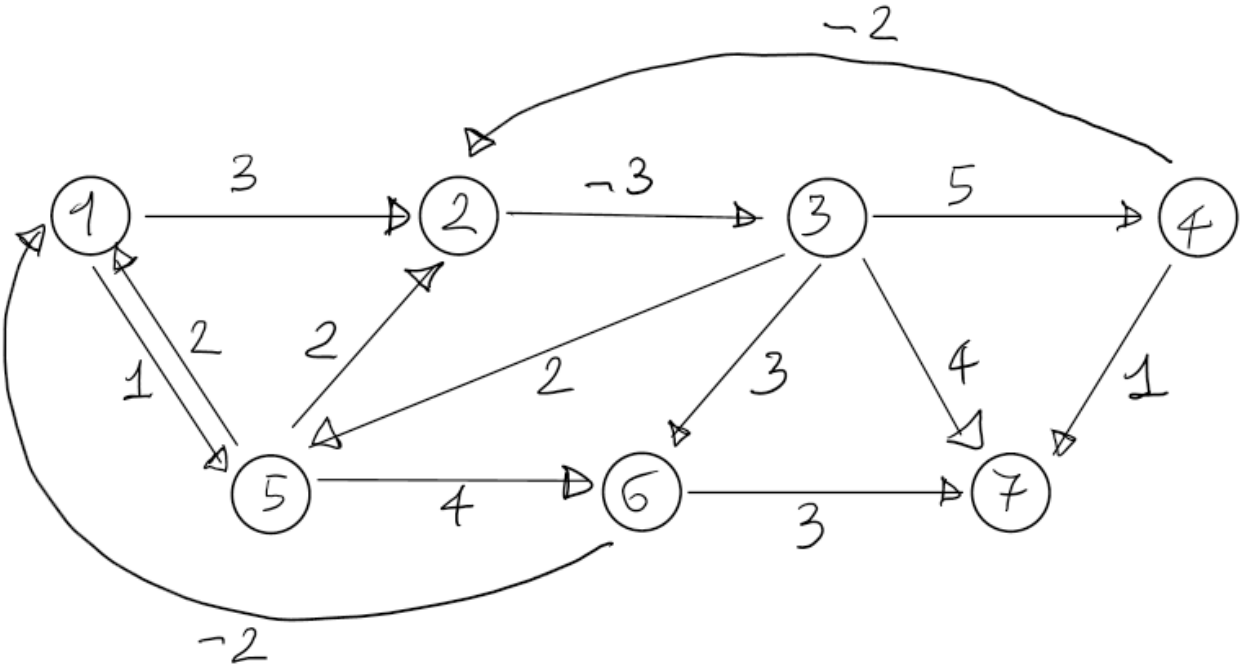
Si considerino le seguenti sequenze di numeri interi:

A = 24, 12, 22, 15, 16, 18, 29, 5 **B = 22, 29** **C = 20, 9, 138**

1. **[3 punti]** Mostrare **passo-passo l'evoluzione della tabella Hash di dimensione 17 con liste di trabocco** che si ottiene, a partire dalla tabella vuota, inserendo le chiavi della sequenza **A** nell'ordine indicato, rimuovendo quindi le chiavi della sequenza **B** nell'ordine indicato ed inserendo infine le chiavi della sequenza **C** sempre nell'ordine indicato.
2. **[3 punti]** Ripetere l'esercizio utilizzando una **tabella Hash di dimensione 17 con hashing interno (hashing doppio)** (la seconda funzione Hash dell'hashing doppio deve essere modulo 16).

Esercizio 3 [11 punti]

Si consideri il grafo (diretto) in figura, per il quale si vuole risolvere il problema di calcolare i **cammini minimi a partire dalla sorgente 5.**



[2 punti] 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 mostrandone una possibile esecuzione **passo-passo**, rendendo chiaro le scelte fatte quando l'algoritmo permette di effettuare le operazioni in più di un possibile modo/ordine. 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 dalla sorgente **5** alla destinazione **1**.

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.

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.

Si assuma, senza scrivere nulla, l'esistenza di una classe **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;

Si assuma, senza scrivere nulla, che esistano delle classi che estendono la classe astratta Figurina.

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

[4 punti] Aggiungere alla classe Figurina i seguenti metodi (e fare in modo che diventi una classe astratta):

- 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.

[4 punti] Aggiungere alla classe **Album** un metodo di istanza
get_valore (self) -> float
che restituisce il **valore** in Euro dell'album **self**, definito come la somma dei valori di tutte le figurine senza tener conto dei doppioni (in altre parole, una stessa figurina non deve essere conteggiata più volte).
Se la lista self._figurine è lunga **n**, la complessità nel caso medio 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_valore_marginale (self, a:Album) -> float
che restituisce il **valore marginale** dell'album **a**, definito come l'incremento di valore dell'album **self** qualora venisse fuso con l'album a.
Se la lista self._figurine è lunga **n** e la lista a._figurine è lunga **m**, la complessità nel caso medio deve essere *O(**n+m**).*
Suggerimento: si sfrutti il metodo get_valore.