

Cognome e Nome _____

Matricola _____

Appello dell'11 giugno 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"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*2
```

```
class C (B, A):
    pass

class D (A, B):
    pass

a = A(10)
b = B(5)
c = C(8)
d = D(12)

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

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	$L(A) = A + \text{merge}(L(O), O) = A + \text{merge}(O, O) = AO$
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]

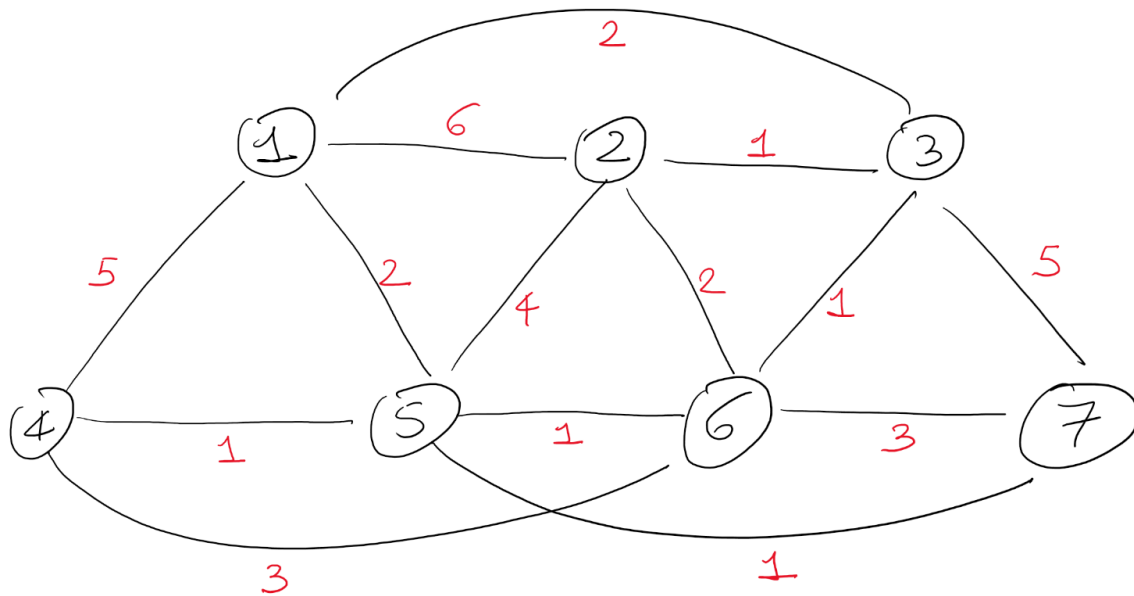
Si consideri il seguente array di numeri interi:

[12, 2, 18, 15, 29, 3, 25, 13, 32]

1. [3 punti] Mostrare **passo-passo** l'esecuzione di **build-max-heap**.
2. [3 punti] Mostrare **passo-passo** il **min-Heap** che si ottiene inserendo i valori nell'ordine in cui compaiono nell'array, a partire dall'heap vuoto.

Esercizio 3 [10 punti]

Si consideri il grafo diretto in figura.



- a. [5 punti] Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **7**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo e del vettore dei padri.
Si descriva il cammino minimo dal nodo 7 al nodo 3. Che peso ha?
- b. [5 punti] Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente, facendo partire l'algoritmo dal nodo **4**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo e del vettore dei padri.
Che peso complessivo ha il minimo albero ricoprente?

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 [12 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.

[4 punti] Si scriva 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.

[4 punti] Si scriva la 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 classe **FigurinaRara** 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

[4 punti] Aggiungere alla classe **Album** un metodo di istanza

`get_doppioni` (**self**) -> `list[Tuple[Figurina, int]]`

che restituisce una lista contenente tutte le figurine che sono presenti almeno due volte nella lista `self._figurine`, con associato il numero di occorrenze meno uno (se ad esempio una figurina è presente 4 volte, deve essere restituita associata all'intero 3, poiché ci sono 3 "doppioni" per lei). 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 dizionario per memorizzare e calcolare il numero di occorrenze di ogni figurina.