Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2024/2025

Appello del 13 febbraio 2025

Esercizio 1 [6 punti]

Si consideri il seguente codice Python.

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

    @property
    def n (self) -> int:
        return self._n + 1

class B:
    _n:int = 0
    @property
    def n (self) -> int:
        return self._n * 2
```

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

class D (A, B):
    pass

class E (D):
    pass

c = C(5)
print (c.n)

e = E(3)
print (e.n)
```

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

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

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

Esercizio 2 [6 punti]

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array **[17, 30, 8, 1, 9, 7, 2]** in modo non decrescente (compresa la fase iniziale di **build-max-heap**).

Esercizio 3 [11 punti]

Tutto il codice richiesto deve essere in **Python** ed annotato con i tipi opportuni.
Si vogliono gestire gli spettacoli di un cinema/teatro con una sala avente capienza di 333 posti.
Si assuma, senza scrivere nulla, l'esistenza di una classe **Persona** con

- una variabile di istanza **_nome** (una stringa);
- una variabile di istanza **_cognome** (una stringa);

e i seguenti metodi di istanza:

- **`__init__`** che inizializza un oggetto della classe **Persona** assegnando nome e cognome
- Proprietà **`nome`** e **`cognome`** (in lettura)
- metodo **`__eq__ (self, o:object)`** che restituisce True se **o** è un oggetto della classe **Persona** con lo stesso nome e cognome di self, False altrimenti.

[4 punti] Si scriva una classe astratta **Spettacolo** con

- una variabile di istanza **_spettatori** (lista di Persona);
- una variabile di istanza **_titolo** (una stringa);
- una variabile di istanza **_giorno** (un intero), che rappresenta il giorno della data dello spettacolo contando i giorni trascorsi dal 1° gennaio 2000.

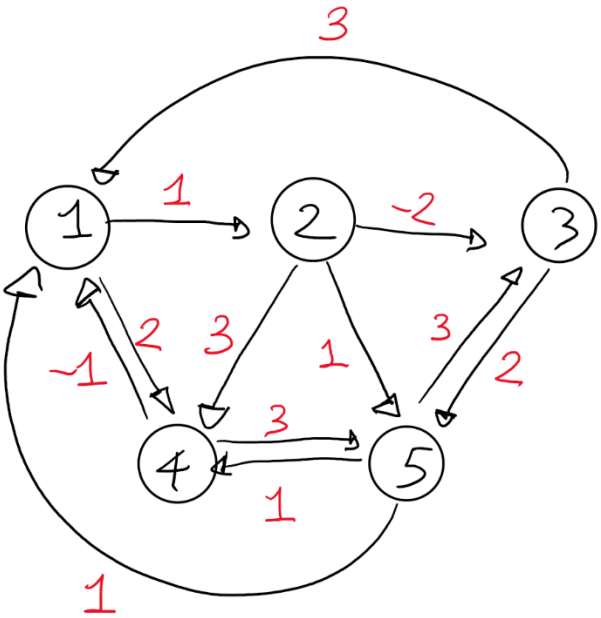
e i seguenti metodi di istanza:

- **`__init__`** che inizializza un oggetto della classe **Spettacolo** assegnando titolo e giorno, e creando una lista vuota di spettatori.
- Proprietà **`giorno`** e **`titolo`** (in lettura)
- metodo di istanza **`is_spettatore(self, p:Persona) -> bool`** che restituisce True se p è presente nella lista degli spettatori, False altrimenti.
- metodo di istanza **`add_spettatore(self, p:Persona) -> None`** che aggiunge p alla lista degli spettatori, qualora non sia già presente e non sia stata raggiunta la capienza massima di 333.
- metodo **astratto** di istanza **`get_tutte_persone(self) -> list[Persona]`** che restituisce una nuova lista contenente tutte le persone coinvolte nello spettacolo (tutti gli spettatori e, nel caso di rappresentazione teatrale, anche tutto il cast).

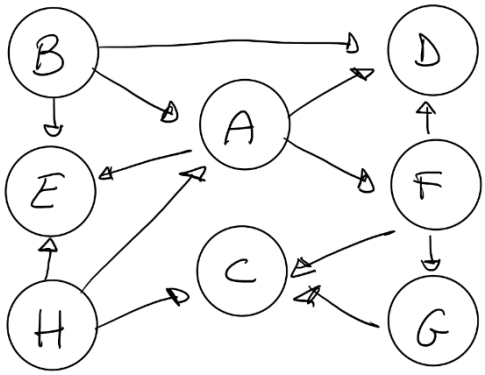
[4 punti] Si scrivano le sottoclassi **Proiezione** e **Rappresentazione,** che estendono **Spettacolo.** Si tenga conto del fatto che la classe **Proiezione** ha una variabile di istanza **_durata** (un intero) che contiene la durata in minuti della proiezione, mentre la classe **Rappresentazione** ha una variabile di istanza **_cast** (lista di Persona) che contiene la lista degli attori coinvolti nella rappresentazione. Per tutte e due le sottoclassi devono essere scritti i metodi __init__ (che prendono in input anche la durata e la lista del cast, rispettivamente) che richiamino opportunamente il metodo __init__ della superclasse **Spettacolo** e deve essere implementato il metodo **`get_tutte_persone(self)`**.

[3 punti] Si scriva una funzione
`segue (p:Persona, elenco:list[Spettacolo])->list[Spettacolo]` che restituisce una lista contenente tutti gli spettacoli della lista **elenco** che hanno la persona **p** tra il proprio pubblico.

Esercizio 4 [11 punti]



a. **[6 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Bellman-Ford** per calcolare i cammini minimi a partire dalla sorgente **2.** Mostrare, per ogni iterazione completa che si fa sugli archi, l'evoluzione del **vettore delle distanze** e del **vettore dei padri**, assumendo di visitare, in ogni iterazione, gli archi in ordine crescente rispetto alle etichette dei nodi da cui escono e, a parità di nodo di partenza, in ordine crescente rispetto all'etichetta del nodo di arrivo. Dire, giustificando la risposta, se il grafo possiede cicli di peso negativo.



b. **[5 punti]** Calcolare un ordinamento topologico del DAG in figura, usando il metodo della visita in profondità ed evidenziando per ogni nodo i timestamp di inizio e fine visita. Si assuma di iniziare e riprendere la visita sempre dal nodo (non ancora visitato) avente l'etichetta alfabeticamente più piccola.

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.