

Cognome e Nome _____

Matricola _____

Appello del 12 febbraio 2026

Esercizio 1 [6 punti]

Si consideri il seguente codice Python.

```
class A:
    def __init__(self, n:int):
        self.n=n + 1

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

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

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

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

```
class C (B):
    pass

class D (A, C):
    pass

a = A(5)
b = B(3)
c = C(10)
d = D(12)

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

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	$L(A) = A + \text{merge}(L(O), O) = A + \text{merge}(O, O) = AO$
B	
C	
D	

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

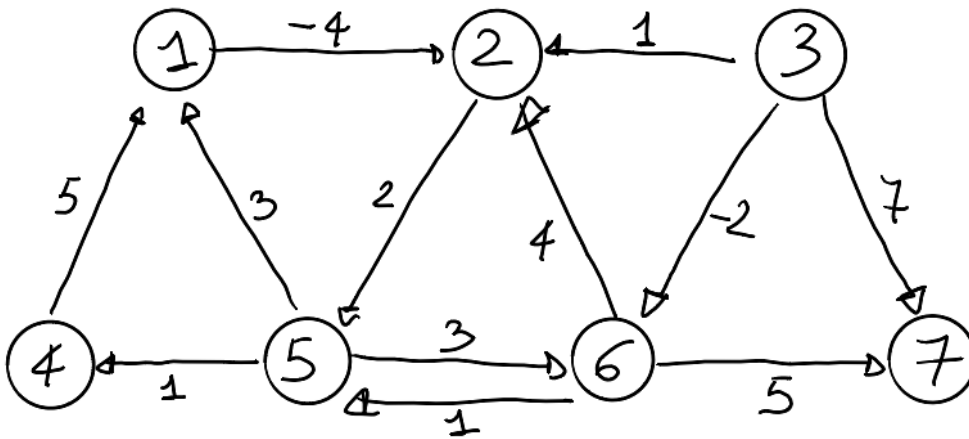
Nel caso in cui il codice dia errore in qualche punto (poiché ad esempio non è possibile calcolare qualche MRO), escludere dal codice il numero minimo di linee in modo da non ottenere errori, commentandole con `#` sul listato stampato in alto su questo foglio, e quindi stampare cosa risulta dal nuovo

Esercizio 2 [6 punti]

1. **[3 punti]** Mostrare l'albero binario di ricerca che si ottiene a partire dall'albero vuoto inserendo nell'ordine indicato i seguenti elementi: **13, 2, 1, 18, 60, 24, 59, 100, 3, 34**
Che altezza ha l'albero ottenuto?
2. **[3 punti]** Si consideri una *tabella hash con lista di trabocco* contenente numeri interi. Data la funzione hash **modulo**, si mostri un esempio in cui l'inserimento di **n chiavi intere** $k_1, k_2, k_3, \dots, k_n$, in una tabella con **31** posizioni risulti in una tabella in cui la ricerca di una chiave **k** risulta avere il peggior tempo possibile di esecuzione. In particolare, si fornisca per ogni i da 1 a n il valore di k_i , (in funzione dell'indice i), ed il valore **k** della chiave da cercare.

Esercizio 3 [11 punti]

Si consideri il grafo (diretto) in figura, per il quale si vuole risolvere il problema di calcolare i **cammini minimi dalla sorgente 3**.



[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.

[9 punti] Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Bellman-Ford** per risolvere il problema. 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.

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 le raccolte punti con premi organizzate da una catena commerciale.

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

- **_codice** (tipo **int**, **Final**)
- **_descrizione** (tipo **str**)
- **_punti** (tipo **int**, rappresenta il numero di punti necessari per ottenere il premio)

ed i seguenti metodi di istanza:

- **__init__** che inizializza un oggetto della *classe* **Premio** assegnando codice, descrizione e punti
- proprietà (**@property**) in lettura per tutti gli attributi, con lo stesso nome (senza **_**);
- proprietà in scrittura per descrizione e punti.
- metodo **__eq__** (**self, o:object**) che restituisce **True** se **o** è un **Premio** con lo stesso codice di **self**, (indipendentemente dalla descrizione e dai punti), **False** altrimenti.

Si assuma, senza scrivere nulla, l'esistenza di una *classe* **PremioConContributo** che estende **Premio** ed ha, inoltre, il seguente attributo:

- **_prezzo** (tipo **int**, esprime in centesimi di euro il contributo da pagare per ottenere il premio)

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

- **__init__** che inizializza un oggetto della *classe* **PremioConContributo** prendendo in input gli opportuni parametri (incluso il prezzo) e richiamando opportunamente il metodo **__init__** della superclasse
- proprietà (**@property**) **prezzo** in lettura e scrittura per l'attributo **_prezzo**.

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

- **_numero_tessera** (tipo **int**, **Final**)
- **_nome** (tipo **str**)
- **_cognome** (tipo **str**)

ed i seguenti metodi di istanza:

- **__init__** che inizializza un oggetto della *classe* **Cliente** assegnando numero_tessera, nome e cognome.
- proprietà (**@property**) in lettura per tutti gli attributi, con lo stesso nome (senza **_**);
- metodo **__eq__** (**self, o:object**) che restituisce **True** se **o** è un **Cliente** con lo stesso numero_tessera di **self**, (indipendentemente dagli altri attributi), **False** altrimenti.
- metodo **__hash__** (**self, o:object**) che restituisce l'hash di un **Cliente**.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Richiesta**, che rappresenta la richiesta di un premio da parte di un cliente, con i seguenti attributi di istanza:

- **_cliente** (di tipo **Cliente**)
- **_premio** (di tipo **Premio**)

ed i seguenti metodi di istanza:

- **__init__** con tutti i parametri necessari, che inizializza un oggetto della *classe* **Richiesta** assegnando **_cliente** e **_premio**
- Proprietà in lettura (senza **_**) per ciascun attributo

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

- **_clienti** (**list** di **Cliente**), contenente tutti i clienti della catena commerciale
- **_premi** (**list** di **Premio**), contenente tutti i premi della raccolta
- **_saldi_punti** (**dict** con chiave **Cliente** e valori **int** che rappresentano il saldo del punteggio)
- **_richieste_accettate** (**list** di **Richieste**), contenente le richieste di premio effettuate dai Clienti.

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

trasferisci_punti (self, punti:int, mittente:Cliente, destinatario:Cliente) -> None

Che, se i clienti **mittente** e **destinatario** sono clienti della catena commerciale e il saldo punti di mittente è maggiore o uguale a **punti**, aggiorna il saldo punti dei clienti **mittente** e **destinatario** in modo da trasferire **punti** punti dal cliente **mittente** al cliente **destinatario**.

Se le condizioni sopra illustrate non sono soddisfatte, viene sollevata un'opportuna eccezione.

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

aggiungi_richiesta_combinata (self, c:Cliente, premi:list[Premio]) -> float

Che, se **c** è un cliente della catena commerciale avente saldo punti sufficiente per richiedere contemporaneamente **tutti i premi** della lista **premi**, aggiunge alla `self._richieste_accettate` tutte le richieste aventi **c** come cliente e ciascun premio della lista **premi** come premio.

Il metodo deve restituire il contributo totale in euro che il cliente deve corrispondere per tutte le richieste aggiunte alla lista `self._richieste_accettate` (zero se non viene aggiunta nessuna richiesta).

[3 punti] Aggiungere alla classe **GestionePremi** una proprietà in lettura

premi_più_richiesti (self) -> list[Premio]

che restituisce una lista contenente tutti i premi richiesti, nella lista `self._richieste_accettate`, il maggior numero di volte (viene restituita una lista vuota se la lista `self._richieste_accettate` è vuota). Se la lista `self._premi` è lunga **np** e la lista `self._richieste_accettate` è lunga **nr**, la complessità nel caso medio (con l'uso di dizionari) deve essere $O(np + nr)$.

Suggerimento: si usi un dizionario per memorizzare e calcolare il numero di richieste di ogni premio.

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.