Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2025/2026

Appello del 29 gennaio 2026

Esercizio 1 [6 punti]

Si consideri il seguente codice Python.

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

class B:
    def __init__ (self, n:int):
        self.n= n + 2

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

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

class D (A, B):
    pass

class E (D, B):
    pass

c = C(10)
d = D(12)
e = E(8)
print (e.n)
print (d==c)
```

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	
E	

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]

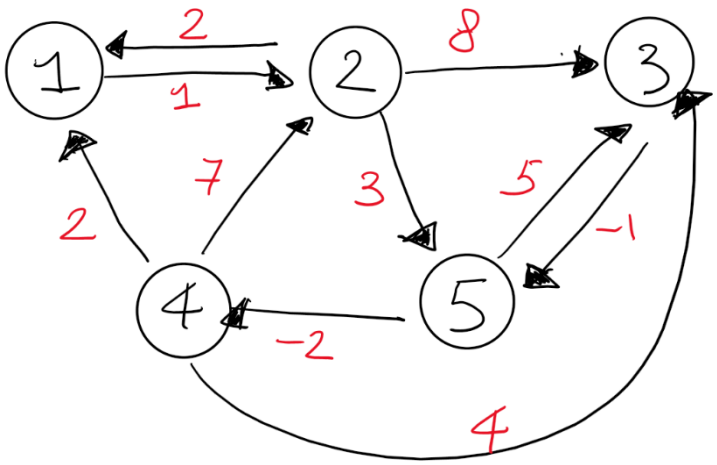
Si considerino le seguenti sequenze di numeri interi:

A = 7, 12, 22, 15, 16, 35, 29, 5, 18 **B = 22, 29** **C = 3, 9, 140**

1. **[3 punti]** Mostrare **passo-passo l'evoluzione della tabella Hash di dimensione 13 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 13 con hashing interno (hashing doppio)** (la seconda funzione Hash dell'hashing doppio deve essere modulo 12).

Esercizio 3 [11 punti]

Si consideri il grafo (diretto) in figura, per il quale si vuole risolvere il problema di calcolare i **cammini minimi tra tutte le coppie di nodi.**



[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.

[9 punti] Si applichi quindi tale algoritmo 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.

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.
- metodo **__str__ (self)** che restituisce la rappresentazione come stringa di un Premio, nella forma `Premio` **`x`** `con codice` **`y:`** `punti` **`z`**
  dove x, y e z rappresentano la descrizione, il codice e i punti del premio **self**, rispettivamente.

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**.

[3 punti] Scrivere (fuori dalle classi) un metodo
acquisizione_premi (premi:list[Premio], budget_prezzo:float, budget_punti:int) -> bool
che restituisce **true** se e solo se è possibile prendere **tutti insieme** i premi della lista **premi** utilizzando un numero di punti al più uguale a **budget_punti**, e pagando al più **budget_prezzo** Euro.

[3 punti] Si scriva la *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.

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.

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **GestionePremi** creando liste vuote per _clienti, _premi e _richieste_accettate e in dizionario vuoto per _saldi_punti.
- metodo **add_cliente (self, c:Cliente, punteggio_iniziale:int=0)** che aggiunge il cliente c alla lista _clienti, se non è già presente in essa, assegnandogli il punteggio iniziale ricevuto come parametro.
- metodo **add_premio (self, p:Premio)** che aggiunge il premio p alla lista _premi, se non è già presente in essa

[2 punti] Aggiungere alla classe **GestionePremi** un metodo di istanza
richiedi_premio (self, c:Cliente, p:Premio) -> None
Che, se il cliente **c** è un cliente della catena commerciale avente un saldo punti maggiore o uguale ai punti necessari per ottenere il premio **p**:

- aggiunge alla lista self._richieste_accettate una richiesta contenente **c** come cliente e **p** come premio;
- aggiorna opportunamente nel dizionario self._saldi_punti il saldo punti del cliente **c**, sottraendo i punti necessari per ottenere il premio **p**.

[3 punti] Aggiungere alla classe **GestionePremi** una proprietà in lettura
cliente_con_contributo_maggiore (self) -> Cliente|None
che restituisce uno dei clienti della catena commerciale che massimizza il contributo economico totale dovuto per i premi corrispondenti a sue richieste accettate presenti nella lista self._richieste_accettate (None se tale lista non contiene premi che richiedono un contributo economico). Se la lista self._clienti è lunga **nc** e la lista self._richieste_accettate è lunga **nr**, la complessità nel caso medio (con l'uso di dizionari) deve essere *O(**nc** + **nr**)*.
Suggerimento: si usi un dizionario per memorizzare e calcolare il contributo economico totale di ogni cliente.

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.