Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2023/2024

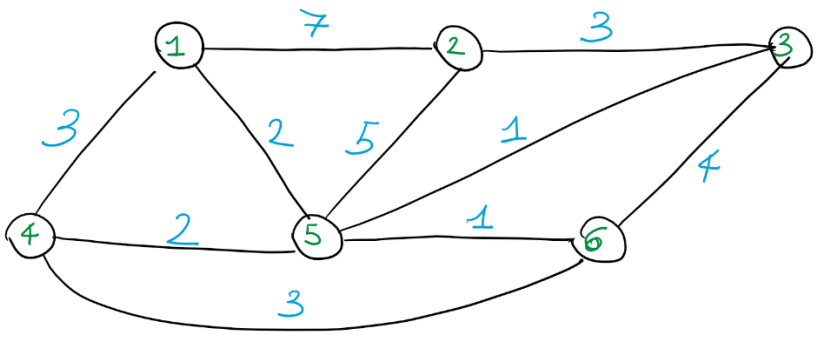
Appello del 25 gennaio 2024

COMPITO N° 1

SECONDA PARTE

Esercizio 1 [20 punti]

Si consideri il grafo diretto in figura.



a. **[10 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **1**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo e del vettore dei padri.
   Che peso complessivo ha il cammino minimo dal nodo 1 al nodo 2?

b. **[10 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente, a partire dal nodo sorgente **1**. 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 **100** minuti per chi svolge solo la seconda parte e **180** minuti per chi svolge il compito totale (chi non consegna dopo 100 minuti automaticamente rinuncia al voto dell'esonero).
- 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 lontano dalla vostra postazione.**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 2 [14 punti]

Si vogliono gestire, in Java, i movimenti associati al credito di una SIM di telefonia mobile.
Si scriva una classe astratta **Movimento** con

- una variabile di istanza **idSim** (tipo String, private), che contiene il numero telefonico associato alla SIM a cui il movimento si riferisce;
- una variabile di istanza **tempo** (tipo long, private), che rappresenta l'istante del movimento espresso in secondi trascorsi dalle ore 00:00 del 1° gennaio 2000.

e i seguenti metodi di istanza:

- un costruttore che crea un movimento dati campi **idSim** e **tempo**;
- metodo di accesso **`public String getIdSim()`**.
- metodo di accesso **`public long getTempo()`**.
- metodo pubblico astratto **`public abstract double getImporto()`** che l'importo in Euro del movimento. In particolare, se il movimento è un movimento di spesa viene restituito un valore minore o uguale a zero, mentre se è un movimento di ricarica credito viene restituito un valore positivo.
- metodo **`public boolean isPositivo()`** che restituisce true se e solo se il movimento è un movimento di ricarica (questo metodo non deve sfruttare l'operatore instanceof).

La classe astratta **Movimento** è estesa dalle sottoclassi (non astratte) **Chiamata**, **Sms** e **Ricarica.**
Si tenga conto del fatto che la classe **Chiamata** deve avere:

- una variabile di istanza **durata** (tipo int, private) che contiene la durata in secondi della chiamata;
- una variabile di istanza **destinatario** (tipo String, private), che contiene il numero chiamato.

La classe **Sms** deve avere:

- una variabile di istanza **destinatario** (tipo String, private), che contiene il destinatario dell'SMS.

La classe **Ricarica** deve avere:

- una variabile di istanza **importo** (tipo double, private), che contiene l'importo della ricarica e deve valere almeno 0,01.

Per tutte e tre le sottoclassi vanno scritti i costruttori (che prendono in input gli opportuni parametri e che richiamino opportunamente il costruttore della classe **Movimento**) e va implementato il metodo **`double getImporto()`** tenendo conto del fatto che:

- una chiamata ha un costo di 0,001 euro al secondo;
- un SMS ha un costo di 0,20 euro.

PRIMA PARTE

Esercizio 3 [14 punti]

Si progetti infine una classe **Gestore** con

- una variabile di istanza **movimenti** (tipo ArrayList<Movimento>, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea un ArrayList **movimenti** vuoto.
- un metodo **`public ArrayList<Movimento> estratto (String idSim, long tempoInizio, long tempoFine)`** che restituisce i movimenti (tra quelli nell'arraylist **movimenti**) relativi alla Sim **`idSim`** il cui campo tempo sia compreso tra **`tempoInizio`** $\in$ **`tempoFine`**.
- un metodo **`public double getCredito (String idSim)`** che restituisce il credito residuo attualmente presente sulla Sim **`idSim`.**
- un metodo **`public double getTotaleRicariche (String idSim)`** che restituisce il totale delle ricariche effettuate sulla Sim **`idSim`.**

Esercizio 4 [10 punti]

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array **[7, 3, 80, 12, 20, 9, 17, 4, 2]** in modo non decrescente (compresa la fase iniziale di **build-max-heap**).

Esercizio 5 [10 punti]

Si considerino le seguenti classi.

```
class A {
      protected int n;

      public A(int n) {
            this.n = n;
      }

      public int metodo(int i) {
            n = n + i;
            return n;
      }

      public double metodo(double x) {
            n = n + (int)x;
            return n+x;
      }
}
```

1

2

```
class B extends A {
      public B(int x, int y) {
            super(x + y);
      }

      public int metodo(int i) {
            n = n - 3*i;
            return n;
      }
}
```

3

Si scriva nel riquadro a destra cosa viene stampato a video dal seguente codice:

```
A a = new A(3);
B b = new B(5, 1);
A ab = b;
System.out.println(ab.metodo(3.0));
System.out.println(ab.metodo(1));
System.out.println(ab.metodo(b.metodo(1.3)));
```

Si giustifichi la risposta mostrando in particolare:

- le firme associate a tempo di compilazione e a tempo di esecuzione ad ogni chiamata di metodo:

	Tempo di compilazione	Tempo di esecuzione
`ab.metodo(3.0)`		
`ab.metodo(1)`		
`b.metodo(1.3)`		
`ab.metodo(b.metodo(1.3))`		

- l'evoluzione della memoria nelle parti *stack* ed *heap* (sul foglio protocollo).

Cognome e Nome _____

Matricola _____

COMPITO N° 1

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2023/2024

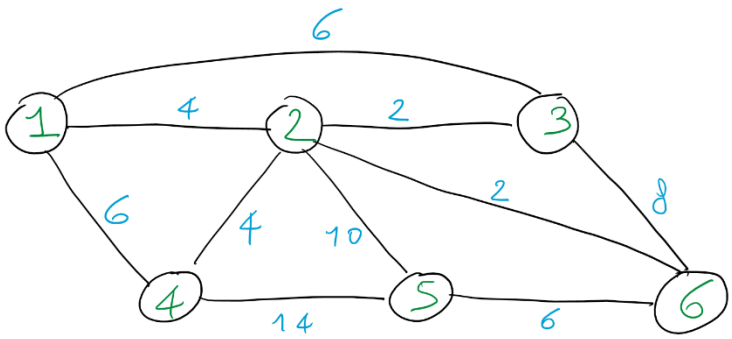
Appello del 25 gennaio 2024

COMPITO N° 2

SECONDA PARTE

Esercizio 1 [20 punti]

Si consideri il grafo diretto in figura.



c. **[10 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **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 cammino minimo dal nodo 4 al nodo 5?

d. **[10 punti]** Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Prim** per il minimo albero ricoprente, a partire dal nodo sorgente **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 **100** minuti per chi svolge solo la seconda parte e **180** minuti per chi svolge il compito totale (chi non consegna dopo 100 minuti automaticamente rinuncia al voto dell'esonero).
- 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 lontano dalla vostra postazione.**
- Mettere in vista sul banco un valido documento di identità.

Esercizio 2 [14 punti]

Si vogliono gestire, in Java, i movimenti associati al credito di una SIM di telefonia mobile.
Si scriva una classe astratta **Movimento** con

- una variabile di istanza **idSim** (tipo String, private), che contiene il numero telefonico associato alla SIM a cui il movimento si riferisce;
- una variabile di istanza **tempo** (tipo long, private), che rappresenta l'istante del movimento espresso in secondi trascorsi dalle ore 00:00 del 1° gennaio 2000.

e i seguenti metodi di istanza:

- un costruttore che crea un movimento dati campi **idSim** e **tempo**;
- metodo di accesso **`public String getIdSim()`**.
- metodo di accesso **`public long getTempo()`**.
- metodo pubblico astratto **`public abstract double getImporto()`** che l'importo in Euro del movimento. In particolare, se il movimento è un movimento di spesa viene restituito un valore minore o uguale a zero, mentre se è un movimento di ricarica credito viene restituito un valore positivo.
- metodo **`public boolean isPositivo()`** che restituisce true se e solo se il movimento è un movimento di ricarica (questo metodo non deve sfruttare l'operatore instanceof).

La classe astratta **Movimento** è estesa dalle sottoclassi (non astratte) **Chiamata**, **Sms** e **Ricarica.**
Si tenga conto del fatto che la classe **Chiamata** deve avere:

- una variabile di istanza **durata** (tipo int, private) che contiene la durata in secondi della chiamata;
- una variabile di istanza **destinatario** (tipo String, private), che contiene il numero chiamato.

La classe **Sms** deve avere:

- una variabile di istanza **destinatario** (tipo String, private), che contiene il destinatario dell'SMS.

La classe **Ricarica** deve avere:

- una variabile di istanza **importo** (tipo double, private), che contiene l'importo della ricarica e deve valere almeno 0,01.

Per tutte e tre le sottoclassi vanno scritti i costruttori (che prendono in input gli opportuni parametri e che richiamino opportunamente il costruttore della classe **Movimento**) e va implementato il metodo **`double getImporto()`** tenendo conto del fatto che:

- una chiamata ha un costo di 0,001 euro al secondo;
- un SMS ha un costo di 0,20 euro.

PRIMA PARTE

Esercizio 3 [14 punti]

Si progetti infine una classe **Gestore** con

- una variabile di istanza **movimenti** (tipo ArrayList<Movimento>, private);

Implementare i seguenti metodi di istanza:

- un costruttore senza parametri che crea un ArrayList **movimenti** vuoto.
- un metodo **`public ArrayList<Movimento> estratto (String idSim, long tempoInizio, long tempoFine)`** che restituisce i movimenti (tra quelli nell'arraylist **movimenti**) relativi alla Sim **`idSim`** il cui campo tempo sia compreso tra **`tempoInizio`** ∈ **`tempoFine`**.
- un metodo **`public double getCredito (String idSim)`** che restituisce il credito residuo attualmente presente sulla Sim **`idSim`.**
- un metodo **`public double getTotaleRicariche (String idSim)`** che restituisce il totale delle ricariche effettuate sulla Sim **`idSim`.**

Esercizio 4 [10 punti]

Mostrare **passo-passo l'esecuzione dell'heap-sort** per ordinare l'array **[6, 2, 11, 81, 9, 21, 2, 4, 16]** in modo non decrescente (compresa la fase iniziale di **build-max-heap**).

Esercizio 5 [10 punti]

Si considerino le seguenti classi.

```
class A {
      protected int n;

      public A(int n) {
            this.n = n;
      }

      public int metodo(int i) {
            n = n + i;
            return n;
      }

      public double metodo(double x) {
            n = n + 2*(int)x;
            return n+x;
      }
}
```

1

2

```
class B extends A {
      public B(int x, int y) {
            super(x + y);
      }

      public int metodo(int i) {
            n = n - i;
            return n;
      }
}
```

3

Si scriva nel riquadro a destra cosa viene stampato a video dal seguente codice:

```
A a = new A(2);
B b = new B(4, 3);
A ab = b;
System.out.println(ab.metodo(4.0));
System.out.println(ab.metodo(2));
System.out.println(ab.metodo(b.metodo(0.3)));
```

Si giustifichi la risposta mostrando in particolare:

- le firme associate a tempo di compilazione e a tempo di esecuzione ad ogni chiamata di metodo:

	Tempo di compilazione	Tempo di esecuzione
`ab.metodo(4.0)`		
`ab.metodo(2)`		
`b.metodo(0.3)`		
`ab.metodo(b.metodo(0.3))`		

- l'evoluzione della memoria nelle parti *stack* ed *heap* (sul foglio protocollo).

Cognome e Nome _____

Matricola _____

COMPITO N° 2