# Università Degli Studi dell'Aquila

I appello di Laboratorio di Algoritmi e Strutture Dati - A.A. 2022/2023

Martedì 24 gennaio 2023 – Dott.ssa Giovanna Melideo (Durata: 1:30 h)

Svolgere i seguenti esercizi avendo come riferimento il linguaggio JAVA.

**ESERCIZIO 1**

Un Ospedale gestisce un archivio elettronico per raccogliere i dati relativi ai pazienti ricoverati. Si consideri la seguente classe **`Paziente`** (**cartella "I appello 24-01-2023" del Team di classe**) che rappresenta un paziente ricoverato. Di ogni paziente vengono archiviate le seguenti informazioni: nome, cognome, CF, età.

Realizzare una classe **`Archivio`** come una mappa che associa ad ogni reparto (oggetto di tipo String, chiave univoca) l'insieme di pazienti (Set<Paziente>) in esso ricoverati, fornendo le seguenti funzionalità:

1. Inserimento di un nuovo paziente nell'insieme associato al dato reparto (se il paziente è già presente in archivio l'operazione deve fallire):
   ```
   public boolean insert(Paziente paziente, String reparto);
   ```
2. Cancellazione di un paziente dall'insieme associato al dato reparto (se il paziente non è presente nell'insieme l'operazione deve fallire):
   ```
   public boolean delete(Paziente paziente, String reparto);
   ```
3. Ricerca del reparto di ricovero di un dato paziente (null se assente):
   ```
   public String find (Paziente p);
   ```
4. Estrazione dell'insieme dei pazienti ricoverati in un dato reparto, in ordine crescente di CF:
   ```
   public Set<Paziente> elencoReparto (String reparto);
   ```
5. Creazione di un nuovo insieme contenente tutti i pazienti contenuti nell'archivio, in ordine crescente di età:
   ```
   public Set<Paziente> elenco ();
   ```

**ESERCIZIO 2**

Realizzare un metodo costruttore della classe LinkedBinaryTree<E>

```
public LinkedBinaryTree (List<E> list)
```

che prende in input una lista di oggetti di tipo E e costruisce un nuovo albero binario come una catena lineare tale che ogni nodo ha esattamente un figlio e l'oggetto in posizione i nella lista è collocato nel nodo di livello i dell'albero. Il nodo di livello $i \neq 0$ è inserito come figlio destro se i è pari, come figlio sinistro se i è dispari.

**ESERCIZIO 3**

Aggiungere alla classe **`UndirectedNetwork<>`** un metodo

```
public UndirectedNetwork<Vertex> complementary()
```

che restituisce una nuova istanza di tipo **`UndirectedNetwork<Vertex>`** che rappresenta il grafo "complementare" al grafo corrente (assegnare peso unitario a tutti gli archi).

*Dato un grafo G = (V, E), il uso grafo complementare ha gli stessi nodi di G, ma contiene solo gli archi che non sono presenti in G.*

```
class Paziente implements Comparable<Paziente>
{
  String nome, cognome;
  String cf; //valore univoco
  int eta;

  public Paziente (String nome, String cognome, String cf, int eta) {
    this.nome = nome.toUpperCase();
    this.cognome = cognome.toUpperCase();
          this.cf = cf.toUpperCase();
    this.eta = eta;
  } // constructor

  public Paziente (String cf) {
             this(null, null, cf, 0);
  } // constructor

  public String getNome(){ return nome; }
  public String getCognome(){ return cognome; }
  public String getCF(){ return cf; }
  public int getEta(){ return eta; }

  /**
   * confronta questo oggetto Paziente con l'oggetto Paziente specificato in base al CF
   **/
  public int compareTo (Paziente altroP) {
           return this.cf.compareTo(altroP.cf);
  } // method compareTo

  /**
   * determina se il cf di questo oggetto Paziente   uguale al cf
   * dell'oggetto Paziente specificato
   */
  public boolean equals (Object obj)
  {
           if (obj == null || ! (obj instanceof Paziente)) return false;
                    return this.cf.equals(((Paziente)obj).cf);
  } // method equals


  /**
   * Restituisce una rappresentazione String di questo oggetto Paziente
   */
  public String toString()
  {
           return nome + " " + cognome + ", CF: " + cf + ", et : " + eta;
  } // method toString

} // end-class
```