1. Nell'implementazione degli alberi In our AVL vista al corso ogni nodo contiene l'altezza del suo sottoalbero e questo è valore è un intero arbitrariamente grande. Lo spazione necessario per rappresentare un albero AVL può essere ridotto memorizzando in ciascun nodo il suo fattore di bilanciamento, definito come la differenza tra le altezze del suo sottoalbero sinistro e sottoalbero destro. Si osservi che il fattore di bilanciamento di un nodo è un valore copreso in {-1, 0, 1} ma durante l'esecuzione di un inserimento o di una cancellazione può temporaneamente assumere anche i valori -2 o 2.
   Progettare la classe **NewAVLTreeMap**, che fornisce la stessa interfaccia di AVLTreeMap e memorizza nei nodi i fattori di bilanciamento invece che le altezze dei sottoalberi.

2. Progettare ed implementare la classe **Statistics** che permette di elaborare statistiche su di un dataset costituito da un insieme di coppie (***key, value***). La classe deve utilizzare un NewAVLTreeMap i cui nodi contengono elementi (***key, frequency, total***) dove ***key*** è la chiave presente nel dataset, ***frequency*** è uguale al numero di occorrenze della chiave nel dataset, e ***total*** è la somma dei valori associati a tutte le sue occorrenze. Il costruttore della classe deve prendere in input il nome del file che rappresenta il dataset.

   La classe deve fornire i seguenti metodi pubblici:
   a. **add(k, v)** – aggiunge la coppia ***(k, v)*** alla mappa; se la chiave ***k*** è già presente nella mappa deve aggiornare i campi ***frequency*** e ***total***;
   b. **len()** – restituisce il numero di chiavi presenti nella mappa;
   c. **occurrences()** – restituisce la somma delle frequenze di tutti gli elementi presenti nella mappa
   d. **average()** - restituisce la media dei valori di tutte le occorrenze presenti nel dataset;
   e. **median()** – restituisce la mediana delle key presenti nel dataset, definita come la key tale che la metà delle occorrenze del dataset hanno key minori o uguali della mediana (sugg. tener conto delle frequenze di ciascuna key)***;***
   f. **percentile(j = 20)** – restituisce il ***j***-imo percentile, per ***j*** = 1, … 99, delle lunghezze delle ***key,*** definito come la key ***k*** tale che il j per cento delle occorrenze del dataset hanno key minori o uguali di ***k***;
   g. **mostFrequent(j)** – restituisce la lista delle ***j key*** più frequenti;
   Analizzare la complessità di tempo delle operazioni fornite.

3. Supponete di avere la directory dir contenente ***n*** file, alcuni dei quali potrebbero essere copie con nomi diversi. Fornire la funzione **find_repetition**(dir) che, presa in input la directory dir, restituisce la lista dei file replicati. La funzione proposta deve avere complessità lineare nel numero dei file esaminati.

4. Diciamo che un pattern P di lunghezza ***m*** è una sottostringa circolare di un testo T di lunghezza ***n*** > ***m*** se P e una sottostringa (normal) di T oppure se P è uguale ad una concatenazione di un suffisso di T ed un prefisso di T, cioè se esiste un indice 0 <= ***k*** < ***m***, tale che P = T [***n*** − ***m*** + ***n*** : ***n***] + T [0 : ***k***].
   Implementare la funzione **circular_substring**(P, T) che restituisce True se P è una sottostringa circolare di T e False altrimenti. La funzione deve avere complessità O(***m*** + ***n***).

La soluzione deve essere caricata sul sito del corso **entro le ore 23.59 di domenica 25 novembre**.
Il materiale consegnato deve essere costituito da un unico file compresso contenente:

- Un file pdf con la documentazione del progetto

- Un progetto Python chiamato #gruppo_2_TdP contenente
  - il package Python TdP_collections (con tutte le implementazioni delle strutture dati di supporto, eventualmente modificate);
  - il package Python pkg_i contenente le classi e le funzioni per la soluzione dell'esercizio i (i = 1, 2, 3, 4);
  - script di testing per ciascun esercizio.

Per ogni gruppo deve essere caricato un solo file.