Esercizio 2: Classe Statistics

La classe elabora statistiche su un dataset di key-value usando un NewAVLTreeMap i cui nodi contengono gli elementi key, frequency e total che rappresentano rispettivamente la chiave dell’elemento, la frequenza con la quale si presenta nel dataset e la somma dei valori per ogni elemento contenuto.

Inoltre, la classe presenta gli attributi occur e total che permetto di memorizzare il numero di occorrenze e la somma dei valori degli elementi nell’AVL permettendo così di velocizzare le operazioni di occurrences() e average().

La complessità del costrutto è O(nlogk) dove n rappresenta il numero di righe nel dataset mentre k il numero di chiavi.

Metodi della classe:

* add(k,v): aggiunge la coppia (k,v) nella mappa. Se la chiave k è già presente all’interno della mappa si aggiornano i campi frequency e total associati a quel nodo. La complessità è O(logk) dove k è il numero di chiavi all’interno del dataset;
* len(): restituisce il numero di elementi della mappa. Complessità O(1);
* occurrences(): restituisce il numero di occorrenze degli elementi della mappa. Complessità O(1);
* average(): restituisce il valore medio degli elementi inseriti nella mappa. Complessità O(1);
* percentile(j): calcola il j-esimo percentile, per j = 0, …, 99 delle frequenze delle chiavi definito come la chiave k tale che il j% delle occorrenze nel dataset abbia la chiave minore o uguale a k. Complessità O(k) dove k è il numero di chiavi all’interno del dataset;
* median(): richiamando il metodo percentile, restituisce il j = 50 percentile con complessità O(k);
* most\_frequences(): restituisce una lista contenente le j chiavi più frequenti. Per l’implementazione di questo metodo è stata utilizzata unaHeapPriorityQueue in quanto consente l’inserimento e la cancellazione in O(logj) con j numero di elementi nell’heap. Complessità O(klogj).

Esercizio 3: find\_repetition()

Funzione utilizzata per individuare file duplicati (file con nomi diversi ma con medesimi contenuti) all’interno di una cartella. Il suo funzionamento è basato sull’utilizzo di un dizionario e della funzione hash() built-in di python. Per ogni file, viene calcolato l’hash del contenuto utilizzato come chiave in una struttura associativa chiave-valore della forma (hash calcolato, lista dei file collidenti). Se i file hanno stesso contenuto il loro hash sarà lo stesso causando una collisione che comporterà un’aggiunta alla lista dei file collidenti con quel contenuto.

La complessità di tale funzione è lineare rispetto al numero di file presente nella cartella.