Classe ListAvl

Per implementare il nostro progetto abbiamo usato le classi già definite LinkedListDictionary e AVLTree e le relative classi da loro utilizzate. La nostra classe, chiamata da noi ListAvl, prende come parametri nel costruttore tre valori come richiesto dalla traccia: il minimo, il massimo, r = 6 e b. Il testo richiede che questi parametri rispettino determinate condizioni, noi abbiamo scelto di lanciare delle eccezioni nel caso i parametri venissero inseriti erroneamente. A questo punto, tramite l'ausilio dei parametri inseriti, verrà calcolato il parametro d, che servirà a definire la lunghezza dell'array ausiliario. Per quanto riguarda il parametro d, esso veniva richiesto in modulo ma abbiamo notato che per qualsiasi parametro da noi scelto d sarebbe risultato sempre positivo, quindi abbiamo omesso l'utilizzo del modulo. L'array verrà riempito alla sua creazione di oggetti della classe LinkedListDictionary, come richiesto.

Per l'implementazione dei metodi richiesti Insert, Search e Delete abbiamo creato dei metodi privati ausiliari.

__findRIghtSet

Questo metodo permette, data una chiave, di capire il suo insieme di appartenenza, e quindi in quale cella dell'array verrà inserito l'elemento associato. Inizialmente il metodo era implementato utilizzando un for che controllava ogni singolo insieme. Abbiamo notato che questo approccio era molto lento perché richiedeva O(d) , e questo pesava sulla velocità della ricerca, soprattutto per d grandi. Abbiamo dimostrato che è possibile effettuare la ricerca dell'insieme in modo costante.

<u>Dimostrazione:</u>

$\text{min} + i * b \leq n < \text{min} + (i + 1) * b \quad \Longrightarrow \quad i * b \leq n - \text{min} < i * b + b \quad \Longrightarrow \quad i \leq \frac{n-min}{b} < i + 1$

Siccome $i \in \mathbb{N} \quad \Longrightarrow \quad i = \left\lfloor \frac{n-min}{b} \right\rfloor$

__checkListOrAvl

Questo metodo permette, dato in input un oggetto, di capire se esso appartiene alla classe LinkedListDictionary o a AVLTree e ritorna True o False rispettivamente. In caso non appartenga a nessuna delle due classi, lancia una eccezione.

__listToAvl

Il seguente metodo consente di trasformare una lista in un albero avl, viene richiamato all'interno del codice quando una lista arriva a contenere 6 elementi. Il cambio avviene richiamando il metodo delle LinkedListDictonary popFirst per poi aggiungere l'elemento ritornato all'interno di un albero avl, inizialmente vuoto, come radice, usando il rispettivo metodo insert. Questo avviene per ogni elemento della lista e alla fine la procedura ritorna l'albero avl.

__avlToList

Il seguente metodo consente di trasformare un albero avl in una lista, viene richiamato all'interno del codice quando un avl arriva a contenere 5 elementi. Il cambio avviene prendendo la coppia chiave elemento della radice dell'avl ed inserendola nella nuova lista con l'insert, per poi eleminare la radice con un delete. Questo avviene finché l'albero non si svuota e alla fine la procedura ritorna la lista.

print

Stampa il contenuto dell'oggetto ListAvl.

Metodi Insert, Search e Delete

Insert

Il metodo insert prende come parametri una chiave e un valore da inserire all'interno della struttura dati. Come prima cosa trova la posizione del vettore nella quale l'elemento deve essere inserito, con l'ausilio del metodo __findRightSet. Il metodo ora si suddivide in due casi a seconda se la struttura all'interno della posizione è una lista o un avl, questa suddivisione viene effettuata tramite il metodo __checkListOrAvl. Se si tratta di una lista l'elemento viene inserito e poi nel caso in cui la lunghezza sia diventata pari a 6 viene convertita in un albero avl con il metodo __listToAvl. Diversamente, se dovesse contenere un albero avl, l'elemento viene semplicemente inserito.

Search

Data una chiave restituisce il valore associato ad essa, se esiste, altrimenti None. Come nell'insert viene trovato l'insieme di appartenenza della key e viene diviso in due casi usando di conseguenza i metodi per il search peculiari delle classi LinkedListDictonary e AvlTree.

Delete

Il metodo delete prende come parametro una chiave e cancella l'elemento associato ad essa dalla struttura dati. Come prima cosa trova la posizione del vettore da dove l'elemento dovrà essere eliminato, con l'ausilio del metodo __findRightSet. Il metodo ora si suddivide in due casi a seconda se la struttura all'interno della posizione è una lista o un avl, questa suddivisione viene effettuata tramite il metodo __checkListOrAvl. Se si tratta di una lista l'elemento verrà eliminato. Diversamente, se dovesse contenere un albero avl, l'elemento verrà eliminato e nel caso in cui la lunghezza sia diventata pari a 5 verrà convertito in una lista con il metodo __avlToList.