Relazione Progetto Ingegneria degli Algoritmi 2018:

Francesco Lasco
Giorgio Liberatore

December 14, 2018

Abstract

Analisi sull'implementazione di una struttura di dati in grado di gestire dizionari basati su alberi AVL o liste concatenate in maniera dinamica (come da traccia).

1 Introduzione

Lo svolgimento del progetto prevedeva l'implementazione di un dizionario, il quale, basandosi su strutture di dati predefinite di supporto (alberi AVL e liste collegate), inserite in un array, decide in maniera autonoma quando la struttura di supporto deve essere un albero AVL o una lista collegata, andandola a modificare di conseguenza.

2 Scelte Implementative

2.1 Classi importate

LinkedListDictionary; AVLTree; Dictionary.

2.2 Descrizione Del Codice

`__init__`: Il costruttore crea un'istanza della nostra classe con gli attributi di massimo, minimo, b, r, e d come definiti dalla consegna. L'attributo array è una lista inizializzata a None nella quale andremo a inserire i dizionari. L'attributo counterList serve a mantenere il numero di valori inseriti nelle strutture di dati puntate.

`__getIndex`: Il metodo ausiliario __getIndex serve alla nostra classe per trovare l'indice i del partizionamento di Z negli insiemi Bi.

`__linkedListToAvl:` Questo metodo è stato implementato per trasformare una lista collegata in albero avl: prende come parametro l'indice, restituito dal metodo __getIndex, della struttura da cambiare. Scorre tutti gli elementi della lista collegata e li inserisce, uno alla volta, in un albero avl temporaneo. Infine viene sostituita la lista collegata con l'albero avl appena creato.

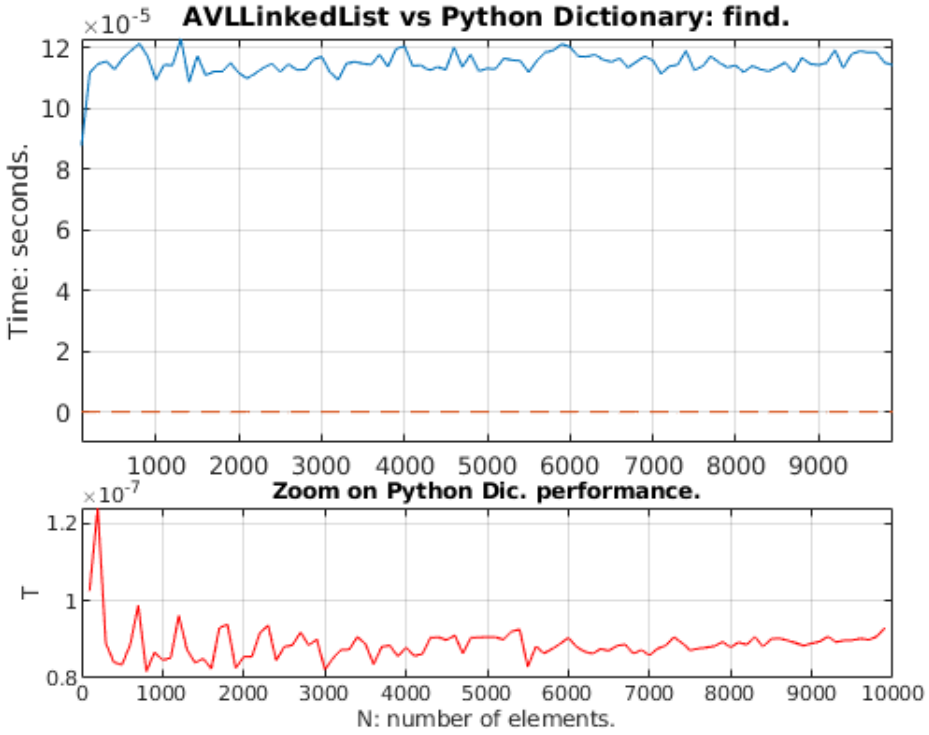
`__avlToLinkedList:` Questo metodo è l'inverso di __linkedListToAvl ma è stato implementato in maniera diversa: prende sempre come parametro l'indice restituito dal metodo __getIndex ma utilizza anche la lista counterList per scorrere tutti gli elementi dell'albero tramite un ciclo for, dove vengono estratti chiave e valore della radice. Successivamente le chiavi e i valori vengono inseriti ,uno ad uno, in una lista collegata temporanea di supporto ed infine viene cancellata la radice, poi si sostituisce l'albero avl con la lista temporanea.

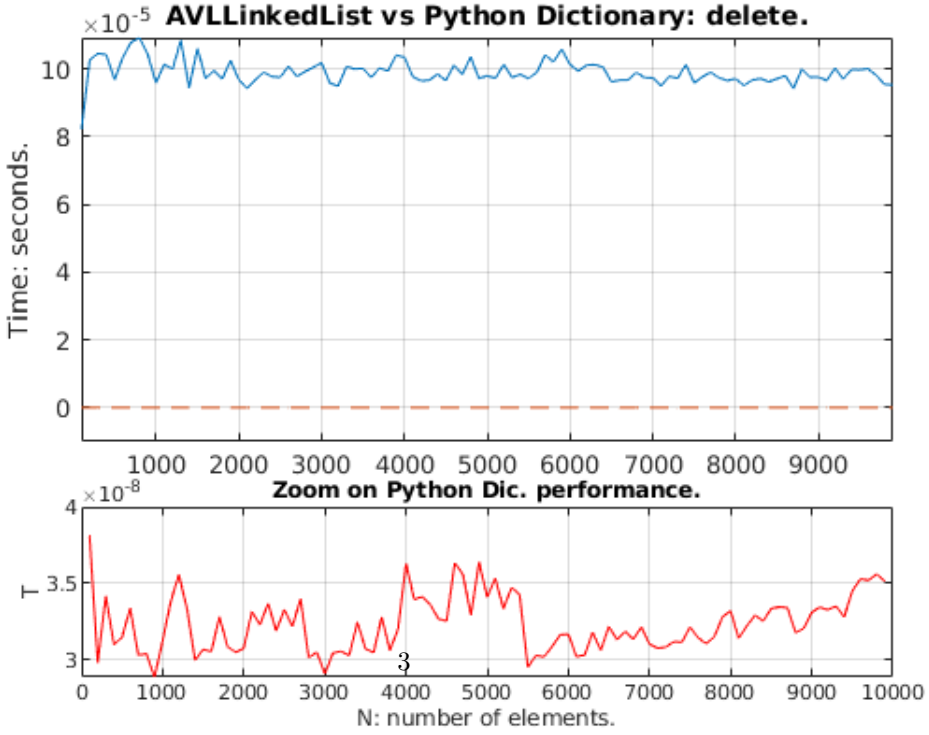
`insert:` Con questo metodo vengono inserite le coppie (key, value). Per prima cosa viene chiamato il metodo __getIndex per individuare a quale posizione dell'array verrà inserito il valore in base alla chiave. Viene effettuato un controllo per verificare che la struttura dati puntata dall'indice non sia vuota (in quel caso viene creata una lista collegata) e si richiama automaticamente il metodo insert della struttura dati corrispondente, incrementando di 1 anche il contatore della counterList. A questo punto se il contatore di elementi è uguale ad r (quindi se dopo un inserimento vengono raggiunti 6 elementi) viene richiamato il metodo __linkedListToAvl per trasformare la struttura dati in albero avl.

`delete:` Questo metodo permette di cancellare una chiave e il relativo valore dalla struttura dati. Per prima cosa si controlla che la struttura dati puntata dalla chiave non sia vuota, altrimenti si solleva un errore. Si fa un ulteriore controllo per vedere se nella struttura dati è presente solo l'elemento da cancellare, in caso affermativo si assegna al valore della struttra dati None e si resetta la counterList degli elementi. In caso contrario si elimina l'elemento e si riduce la counterList di 1. Nel caso in cui la counterList dopo una cancellazione sia uguale ad r-1 si richiama il metodo __avlToLinkedList per trasformare la struttra dati in una lista collegata.

`search:` Questo è il metodo che permette di ritornare un valore presente nel dizionario a partire da una chiave. Prende come parametro la chiave e calcola l'indice della struttura dati corrispondente attraverso la funzione __getIndex. Viene fatto un controllo per verificare che la struttura dati puntata non sia vuota. Poi si ritorna semplicemente il risultato della funzione search sulla struttura dati corrispondente.

3 Risultati Sperimentali





Per i test, i valori (minimo = 88; massimo = 12454; b = 9) sono stai scelti in maniera casuale tali che minimo e massimo fossero abbastanza distanti, in modo tale da diminuire la probabilità di capitare negli ultimi due insiemi Bi.
Le chiavi per gli inserimenti, usate durante i test, sono state selezionate comprese tra il minimo - 10000 e il massimo + 10000. con probabilità di scegliere una chiave minore del minimo o maggiore del massimo pari al 25%.

(I test sono stati effettuati con un numero crescente di insert, fino ad un massimo di 10000 elementi).

Search (average time):

N	Tempo A.L.L (sec.)	Tempo Dic. (sec.)
100	0.000165	0.000000050
1000	0.000108	0.000000030
5000	0.000097	0.000000034
10000	0.000095	0.000000033

Delete (average time):

N	Tempo A.L.L (sec.)	Tempo Dic. (sec.)
100	0.000120	0.000000100
1000	0.000123	0.000000085
5000	0.000113	0.000000091
10000	0.000113	0.000000089

4 Conclusione e commento

I risultati sperimentali ci confermano che il dizionario built-in di Python ha performance migliori nelle operazioni studiate (search, delete), perché basato su tabelle hash.
All'aumentare del numero di elementi inseriti, il tempo medio necessario allo svolgimento dei metodi testati rimane costante.