Relazione Progetto 1

Ingegneria degli Algoritmi 2018/2019

Matteo Lungo - 0253524

10 Dicembre 2018

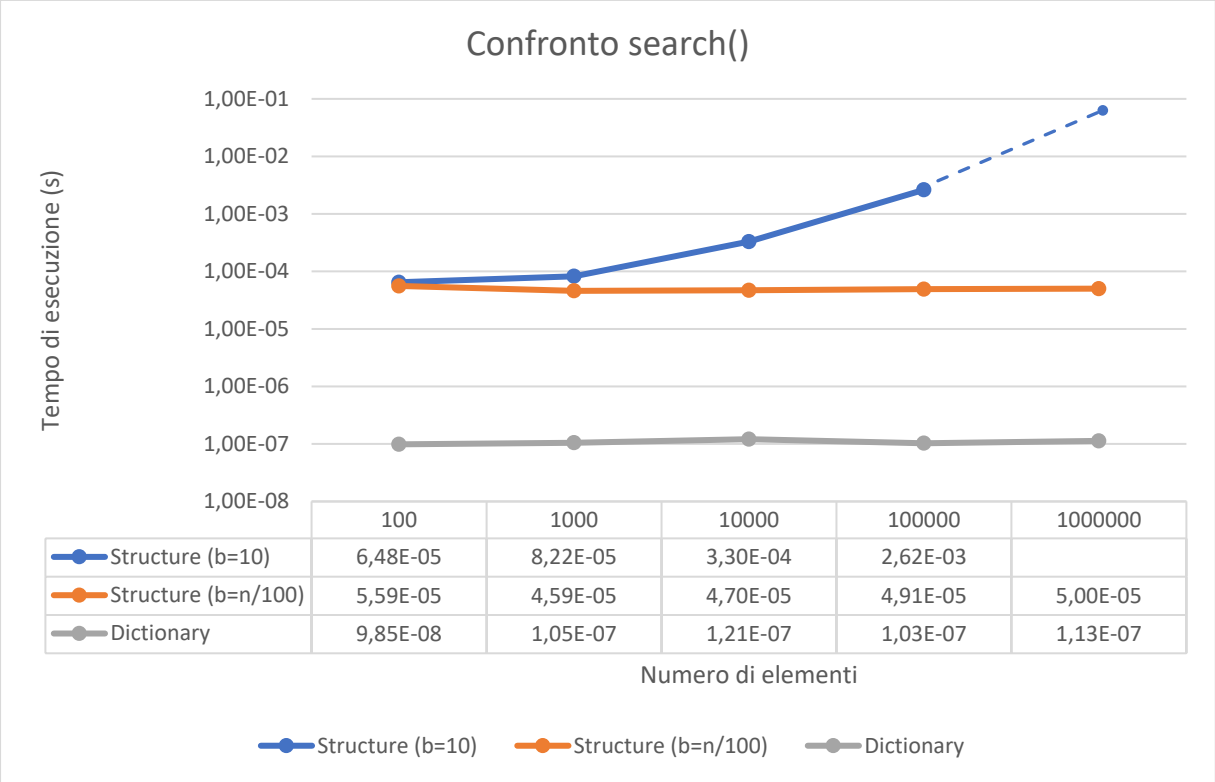
Scelte Implementative

Per la realizzazione del progetto, è stata definita una classe **Structure**, codificata nel file "Structure" e inizializzata secondo i parametri forniti dal testo. La struttura comprende una lista array contenente gli indici delle sottostrutture e una lista **structList** contenente le sottostrutture stesse come sottoliste. Le strutture "lista concatenata" e "albero AVL" sono state importate rispettivamente dai file "LinkedListDictionary" e "avlTree". All'interno della classe sono stati implementati i metodi **insert**, **search** e **delete** del **Dictionary**, un metodo **index** che verifichi l'indice corrispondente alla sottostruttura in cui andrà la chiave inserita e un metodo **control** che verifichi se necessario trasformare la struttura da lista concatenata ad albero AVL e viceversa. A tale scopo sono stati definiti ulteriori due metodi:

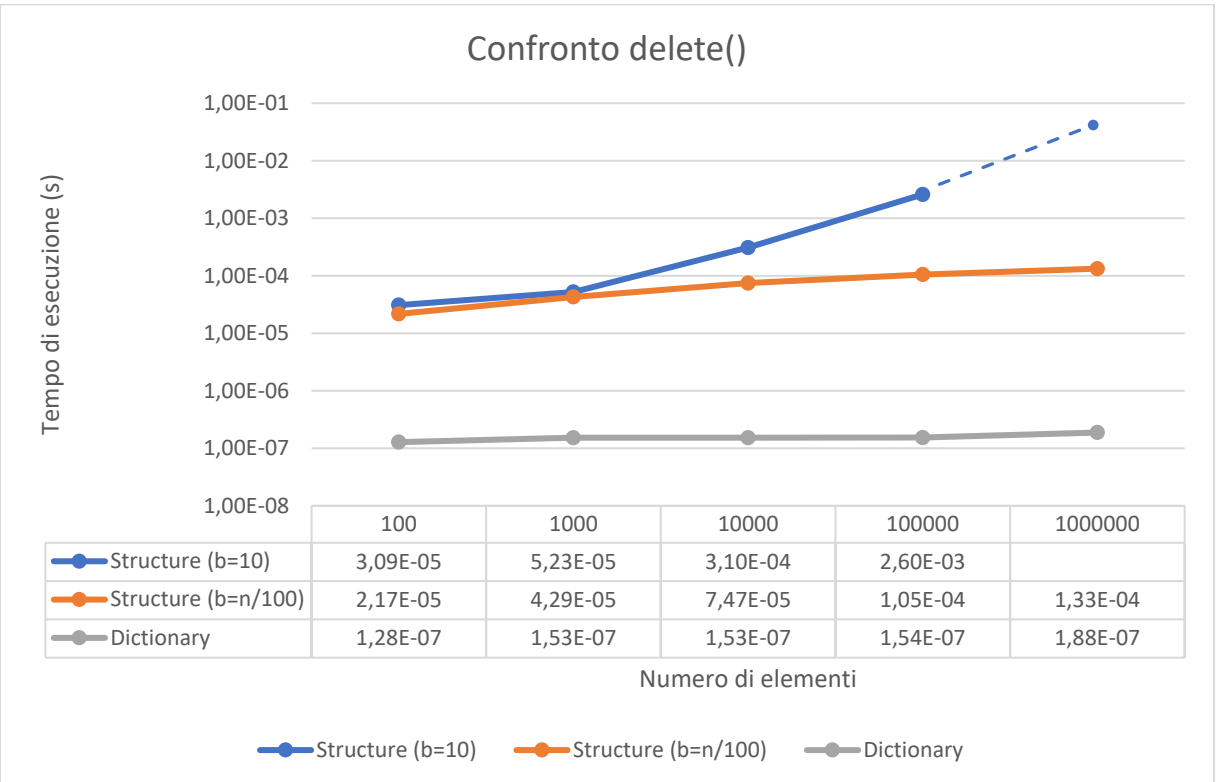
- **listToAVL** che, basandosi sul metodo **printOrdered** della **LinkedList**, copia gli elementi della lista concatenata in una lista temporanea per poi sostituirla con una struttura di tipo albero AVL e inserirli al suo interno.
- **AVLToList** che, al contrario, appoggiandosi su una pila come nel metodo **print** del **binaryTree**, copia gli elementi dell'albero all'interno di una lista temporanea e una volta sostituito l'albero con una lista concatenata li inserisce al suo interno.

Il codice dei file "LinkedListDictionary" e "avlTree" è stato modificato con l'aggiunta di un metodo **lenght** per il calcolo del numero di elementi presenti nelle strutture. Le stime dei tempi di esecuzione sono state effettuate mediante l'utilizzo della libreria timeit.

Risultati sperimentali e commenti



	100	1000	10000	100000	1000000
Structure (b=10)	6,48E-05	8,22E-05	3,30E-04	2,62E-03	
Structure (b=n/100)	5,59E-05	4,59E-05	4,70E-05	4,91E-05	5,00E-05
Dictionary	9,85E-08	1,05E-07	1,21E-07	1,03E-07	1,13E-07



	100	1000	10000	100000	1000000
Structure (b=10)	3,09E-05	5,23E-05	3,10E-04	2,60E-03	
Structure (b=n/100)	2,17E-05	4,29E-05	7,47E-05	1,05E-04	1,33E-04
Dictionary	1,28E-07	1,53E-07	1,53E-07	1,54E-07	1,88E-07

Negli esperimenti effettuati la struttura **Structure** è stata inizializzata con i seguenti parametri:

- $min = 10$
- $max = n$
- $b = 10, \frac{n}{100}$

I metodi **insert**, **search** e **delete** utilizzati sono stati effettuati con valori casuali su un intervallo [x,y] con $x, y \in \mathbb{Z}$.

Dall'analisi dei tempi di esecuzione medi dei metodi **search** e **delete,** il **Dictionary** di Python ha riportato valori costanti all'aumentare degli elementi n, secondo una stima asintotica $O(1)$.

L'analisi dei tempi di esecuzione medi sulla struttura **Structure** dimostra che, per valori di b grandi rispetto all'intervallo *min*, *max* (es. $b = 10$) , al crescere degli elementi presenti, i tempi crescono esponenzialmente. Per valori di b più bassi (es. $b = n/100$), la struttura (effettuando meno confronti) riporta tempi di esecuzione sempre minori, che in casi come quello analizzato rimangono pressoché costanti anche all'aumentare del numero di elementi presenti, pur restando valori notevolmente superiori a quelli restituiti dell'analisi sul **Dictionary**.

In conclusione, la struttura **Dictionary** di Python risulta molto più efficiente rispetto alla struttura **Structure** per entrambi i metodi **search** e **delete** anche nel caso migliore di quest'ultima (tra i 10^2 e 10^3 secondi in meno per operazione negli intervalli considerati).