Relazione sul primo progetto per l'esame di Ingegneria degli Algoritmi

Fabio Buracchi, 0253822

Danilo D'Amico, 0252956

PREFAZIONE

Seguendo le indicazioni riportate nella traccia, è stato svolto il Progetto 1.

Il codice è disponibile su GitHub al seguente link https://github.com/Zaimokuza/EX1IA18.

Nella realizzazione del progetto è stato utilizzato esclusivamente codice scritto dagli autori o i moduli forniti sul sito GitHub del corso, con l'eccezione dei package $matplotlib$ e $pandas$ richiesti dal modulo $plotter.py$ per la generazione dei grafici presenti in questa relazione.

Per riferirsi agli elementi richiesti dalla traccia durante questa relazione si utilizzerà la stessa nomenclatura presente nella traccia del progetto.

Il file $testing.py$ costituisce un variegato insieme di esempi di casi di utilizzo del codice scritto.

SCELTE IMPLEMENTATIVE

Il progetto si articola in due parti: i moduli $customDictionary$ e $linkedListAVLDictionaryHybrid$ presenti nella $root$ della repository ed i moduli presenti nella directory $Librerie$, dai quali viene importato tutto il necessario per l'inizializzazione di liste collegate capaci di gestire dizionari ed alberi AVL.

Il progetto è stato finalizzato alla creazione della classe $CustomDictionary$ che esegue in dettaglio le operazioni richieste dalla traccia. Le istanze $CustomDictionary$ ricevono come parametri min, max e b ed inizializzano un vettore v contenente $d+1$ oggetti di tipo $LinkedListAVLDictionaryHybrid$ che costituiscono delle speciali strutture dati.

Le istanze $LinkedListAVLDictionaryHybrid$ ricevono un parametro len ed inizializzano una lista collegata. Una volta che la lista raggiunge il numero len di elementi questa viene trasformata in un albero AVL (avente almeno len elementi ovviamente). Nel caso in cui attraverso un'operazione $delete$ il numero di elementi dell'albero AVL diventi inferiore a len l'albero AVL viene ritrasformato in lista collegata.

Tra le classi contenute nella cartella $Librerie$, sono state apportate modifiche solo in $LinkedListDictionary$, alla quale è stato aggiunto il metodo $size$ che restituisce la lunghezza della lista collegata. Tale aggiunta semplifica ed aumenta la leggibilità del codice che costituisce $LinkedListAVLDictionaryHybrid$, in quanto consente di astrarre l'implementazione dei metodi di $insert$ e $delete$ dalla struttura dati sulla quale stanno operando.

Per comprendere i costi asintotici delle operazioni $insert$, $search$ e $delete$ della classe $LinkedListAVLDictionaryHybrid$ dobbiamo prima comprendere i costi asintotici delle altre operazioni.

La trasformazione tra albero AVL e lista collegata avviene quando un'operazione di tipo $insert$ o $delete$ porta il numero di elementi contenuti nella struttura a len come abbiamo spiegato.

Il costo di $switchToAVL$, che trasforma una lista collegata in un albero AVL, consiste in len inserimenti di costo $O(\log(len))$, di conseguenza il costo asintotico sarà un $O(1)$

Il costo di $switchToLinkedList$, che trasforma un albero AVL in una lista collegata, consiste nel visitare ciascun nodo dell'albero AVL, contenente $len - 1$ nodi, mediante $fillLinkedList$ e copiarne il contenuto all'interno della nuova lista collegata. Il costo asintotico è chiaramente ancora $O(1)$

Per quanto riguarda i metodi $size$, mentre nell'albero AVL questa operazione ha costo $O(1)$, in $LinkedListDictionary$, l'implementazione di $size$ prevede una singola visita ad ogni nodo della lista collegata e l'aumento di un contatore $length$, rendendo di conseguenza il costo lineare: $O(n)$

Poiché nella $LinkedListDictionary$ il massimo numero di elementi di una lista è len come spiegato in precedenza, il costo del metodo size si riduce così ad $O(len)$, ovvero un $O(1)$

Possiamo ora determinare i costi dei metodi di $insert$, $delete$ e $search$.

Notiamo innanzi tutto che se $LinkedListAVLDictionaryHybrid$ è composto da una lista collegata questa avrà un numero massimo di elementi pari a len, dunque le operazioni di $search$ e $delete$ avranno costo $O(1)$. L'operazione di $insert$ oltre a richiamare quella di $size$ potrebbe richiamare anche un'operazione $switchToAVL$. Poiché tutte queste operazioni hanno costo $O(1)$ il costo di $insert$ sarà proprio $O(1)$.

Nell'altro caso, ovvero che $LinkedListAVLDictionaryHybrid$ sia composto da un albero AVL, le operazioni $search$ ed $insert$ avranno costo $O(\log(n))$. L'operazione di $delete$ oltre a richiamare quella di $size$ potrebbe richiamare anche un'operazione $switchToLinkedList$. Poiché queste due operazioni hanno costo $O(1)$ il costo di $delete$ sarà $O(\log(n))$.

Dunque, il costo asintotico delle tre operazioni $insert$, $delete$ e $search$ del $LinkedListAVLDictionaryHybrid$ nel caso peggiore corrispondono a quelle di un albero AVL.

Si è scelto di mantenere disordinati gli elementi all'interno della lista collegata in quanto un loro inserimento ordinato avrebbe aumentato il tempo di $insert$ da $O(1)$ a $O(\log(n))$ senza un miglioramento dei metodi di $delete$ e $search$, data l'impossibilità di eseguire una ricerca binaria.

Analizziamo ora i costi asintotici delle operazioni $insert$, $search$ e $delete$ della classe $CustomDictionary$.

Per analizzare il costo delle singole operazioni all'interno di $customDictionary$, è utile trovare la cardinalità massima degli insiemi B_i:

Riscrivendo la condizione d'appartenenza all'insieme come

$$B_i = \left\{n \in \mathbb{Z} : i \leq \frac{n - min}{b} < i + 1\right\} = \{n \in \mathbb{Z} : 0 \leq n - min - bi < b\}$$

E notando che

$$n, min \in \mathbb{Z} \wedge b, i \in \mathbb{N} \Rightarrow n - min - bi \in \mathbb{Z}$$

Possiamo facilmente ricavare la cardinalità degli insiemi,

$$B_i = \{x \in \mathbb{Z} : 0 \leq x < b\}$$

$$\#B_i = b$$

Partendo inoltre dall'assunzione

$$key \in B_i \Leftrightarrow 0 \leq key - min - bi < b \Rightarrow \frac{key - min}{b} - 1 < i \leq \frac{key - min}{b} \Rightarrow i = \left\lfloor \frac{key - min}{b} \right\rfloor$$

E poiché i rappresenta proprio l'i-esimo elemento di v, ricaviamo un'utile formula per estrarre l'istanza di tipo $linkedListAVLDictionaryHybrid$ su cui effettuare un'operazione in tempo $O(1)$

Ricordiamo poi che il parametro len delle $linkedListAVLDictionaryHybrid$ coinciderà con r.

Data la struttura del problema, possiamo individuare un caso migliore ed uno peggiore di scelta dei parametri della classe $customDictionary$.

Identifichiamo come caso peggiore una scelta di max e min tali che tutti i valori key inseriti in input siano maggiori del massimo e/o minori del minimo. In tal caso i primi d elementi dell'array non verranno utilizzati, portando l'analisi del costo asintotico dei metodi del $customDictionary$ a ricalcare quelli di $linkedListAVLDictionaryHybrid$.

Identifichiamo come caso migliore una scelta di max e min tali che tutti i valori key inseriti in input siano compresi nell'intervallo (min, max). In tale situazione sappiamo, avendolo dimostrato in precedenza, che il massimo numero di elementi dell'i-esima $LinkedListAVLDictionaryHybrid$ è proprio b. Il costo asintotico dei metodi $insert$, $delete$ e $search$, sarà dunque $O(log(b))$, ovvero $O(1)$.

È possibile ottimizzare l'andamento asintotico della classe $CustomDictionary$ mediante una scelta opportuna del parametro b.

Come visto in precedenza il parametro b determina il massimo numero di elementi che le prime d liste del vettore possono contenere e, di conseguenza, ha un grande peso sull'efficienza del codice.

È necessario notare tuttavia che la creazione di un'istanza $CustomDictionary$ richiede l'inizializzazione immediata di $d + 2$ liste, in modo da non gravare con continui controlli sulle operazioni di $insert$ e $delete$ durante il tempo di esecuzione.

Scegliere un valore per b troppo piccolo dunque, seppur aumentando l'efficienza della struttura dati, aumenterebbe il tempo di caricamento del $CustomDictionary$.

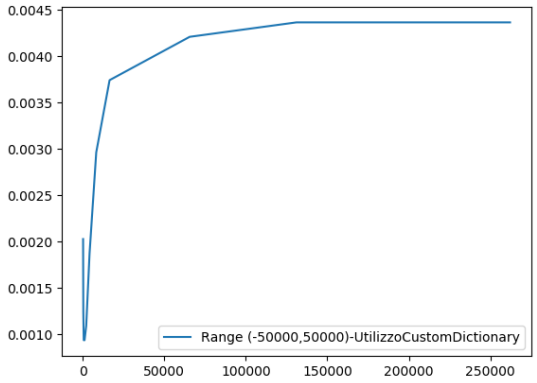
Per trovare un compromesso ottimale sono stati utilizzati test sperimentali.

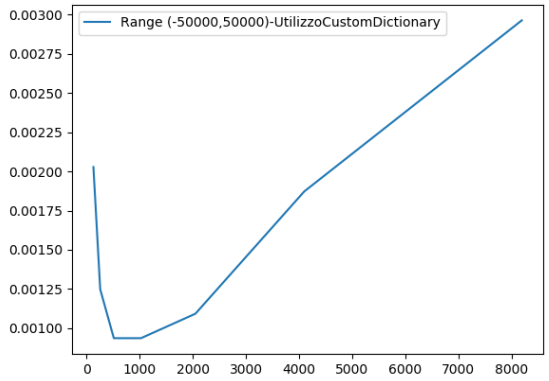
Da questi è emerso che per $|max - min| \ll 1'000'000$ la scelta di un qualsiasi b non compromette in maniera rilevabile l'efficienza del dizionario.

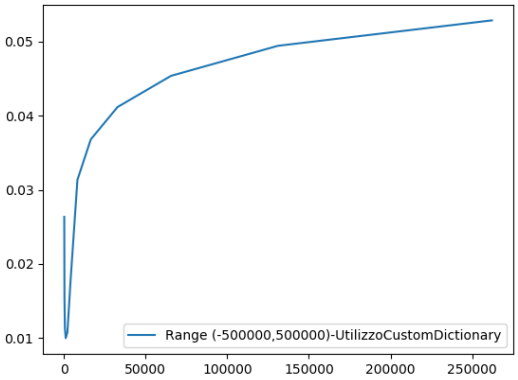
I seguenti grafici rappresentano i risultati dei test eseguiti di $10'000$ operazioni effettuate sul dizionario con b differenti.

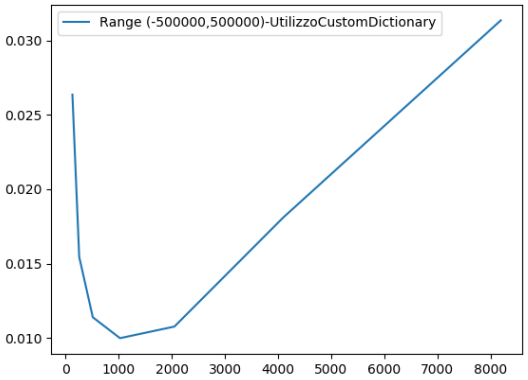
I grafici si alternano mostrando una versione (a destra) comprendente tutti i dati ottenuti dal test e da una versione (a sinistra), contenente in dettaglio solo i più rilevanti.

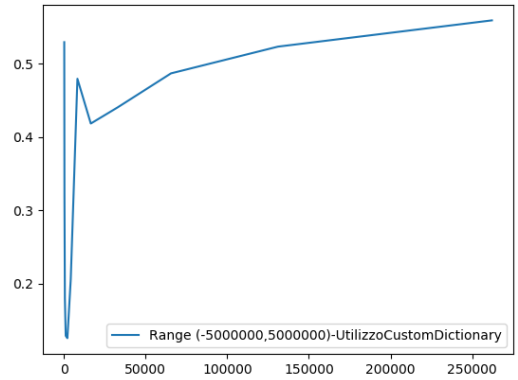
Sulle ordinate viene rappresentato il tempo in secondi mentre sulle ascisse i valori di b.

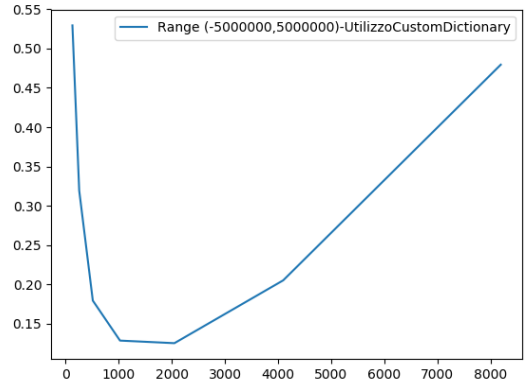












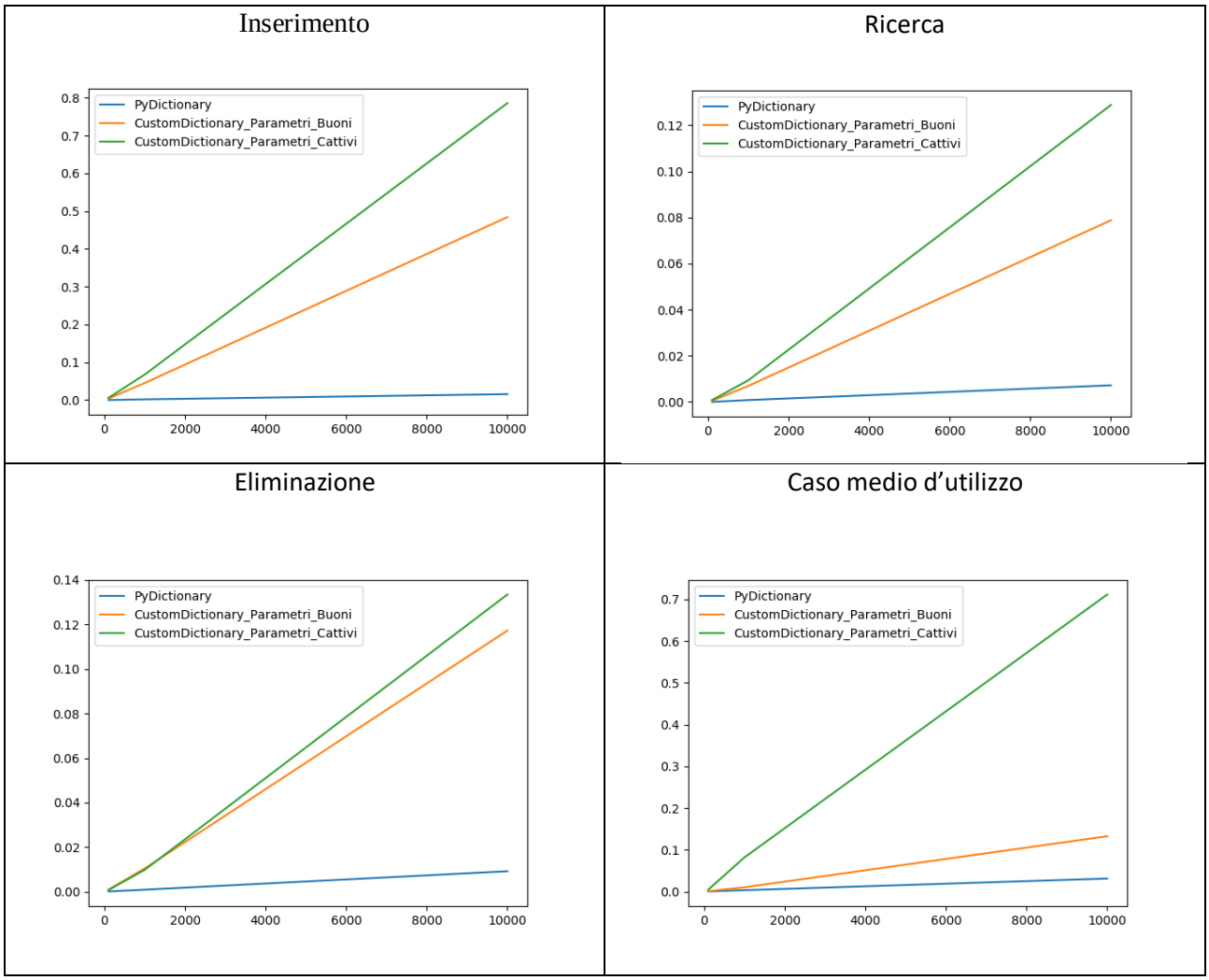
Da un'analisi sperimentale emerge che impostare $b \approx 1000$ contribuisca a rendere più efficiente la struttura dati.

Prendendo in considerazione la struttura degli alberi AVL che tendono a comporre un albero binario completo, un parametro adeguato sarebbe $b = 2^{10} = 1024$

RISULTATI SPERIMENTALI

Gli esperimenti sono stati condotti utilizzando le ultime versioni disponibili di *PyCharm* (versione 2018.3.1) e dell'interprete di *Python* (versione 3.7.1)

I test sono il risultato della media di 100 prove utilizzanti ognuna una delle 100 liste di input randomici differenti utilizzate per ciascun test per avere dati il più possibilmente affidabili.



A livello asintotico i grafici assumono un andamento rispecchiante le considerazioni di carattere teorico effettuate nella sezione precedente della relazione.

Dai test di utilizzo medio emerge che, utilizzando opportunamente i parametri, i risultati siano in media 5 o 6 volte peggiori rispetto a quelli della struttura dati di tipo Dizionario di Python, rappresentando un risultato soddisfacente.

Si può inoltre constatare che la scelta corretta dei parametri è fondamentale per avere un'efficienza adeguata del dizionario.

Per concludere ecco riportati i parametri di utilizzo consigliati per l'implementazione di un'istanza $CustomDictionary$.

Sia L_{key} la lista di tutte le chiavi che s'intendono utilizzare durante l'esecuzione del dizionario.

$$min = 1024 \left\lfloor \frac{\min(L_{key})}{1024} \right\rfloor$$

$$max = 1024 + 1024 \left\lfloor \frac{\max(L_{key})}{1024} \right\rfloor$$

$$b = 1024$$