Università degli Studi di Salerno

Corso di Laurea Magistrale in Ingegneria Informatica

Tecniche di Programmazione

Documentazione Progretto n.2
Gruppo 14

Autori:
Giovanni Iodice
Francesco Liguori
Christian Rapido
Marco Ruggiero

Supervisori:
Prof. Vincenzo Auletta
Prof. Diodato Ferraioli

November 25, 2018

Indice

1 Introduzione

Scopo del progetto Lo spazio necessario per rappresentare un albero AVL può essere ridotto memorizzando in ciascun nodo il suo fattore di bilanciamento , definito come la differenza tra le altezze del suo sottoalbero destro e sinistro. Obiettivo della prima parte del progetto è stato realizzare la classe **NewAVLTreeMap** che fornisca la stessa interfaccia di AVLTreeMap e memorizzi nei nodi i fattori di bilanciamento invece che l'altezza. E' stata innanzitutto riprogettata la classe innestata Node, ora caratterizzata dall'attributo privato **_balance_factor**. Successivamente sono state implementate le utility private **_rebalance_insert(p)** e **_rebalance_delete(p)**, che consentiranno, una volta effettuato un inserimento o una cancellazione di individuare e compensare eventuali sbilanciamenti. Il fattore di bilanciamento di un nodo, infatti, in caso di albero bilanciato appartiene sempre al range -1, 0, 1. Esso diventa diventa temporaneamente pari a 2 o -2 nel caso in cui un'operazione di modifica dell'albero causi uno sbilanciamento. In tal caso, si procede a ribilanciare l'albero.

Successivamente, la classe NewAVLTreeMap è stata utilizzata per implementare la classe Statistics. Quest'ultima consente di elaborare statistiche su un dataset costituito da un insieme di coppie (key,value). In particolare, la key di un elemento del dataset corrisponde alla chiave di un elemento dell'avl. Il value di un nodo dell'albero è invece costituto dal numero di volte con cui la key compare nel dataset e dalla somma dei valori associati alle occorrenze, sarà allora implementato come una lista di due valori.

La terza parte del progetto ha riguardato l'analisi dei file contenuti in una directory. In particolare, lo scopo da perseguire è stato quello di trovare i file che risultino essere copie di altri file presenti nella stessa directory e restituirli in una lista. L'idea alla base della risoluzione di questo problema è quella di realizzare una tabella hash in cui a file dal contenuto uguale viene associato lo stesso hash e sono quindi posti all'interno dello stesso bucket. Pertanto, se un bucket non ha lunghezza unitaria, gli elementi dal secondo all'ultimo sono copie del primo elemento nel bucket e vengono posti all'interno della lista da restituire. Al fine di non appesantire eccessivamente la tabella hash, si è deciso di non usare il contenuto del file come chiave in quanto questo avrebbe causato un aumento della complessità spaziale qualora i file all'interno della directory fossero stati di grandi dimensioni. Ciascun file viene sottoposto quindi ad un hashing preventivo per cui file uguali avranno hash uguali. Questi hash così ottenuti sono stati utilizzati come chiavi della tabella, in modo tale che la dimensione di una chiave non è pari alla dimensione del file ma molto minore.

L'ultima parte del progetto ha riguardato il problema della ricerca di una sottostringa all'interno di un testo che non fosse soltanto normale ma, che potesse anche essere circolare, ovvero, che fosse composta da un suffisso e da un prefisso del testo. Si richiedeva inoltre, che la complessità dell'algoritmo fosse

pari a O(n + m), dove n è la lunghezza del testo ed m la lunghezza del pattern da ricercare.

2 Esercizio n.1

Nell'implementazione degli AVL che abbiamo trattato, ogni nodo conteneva un attributo altezza che veniva utilizzato per ottenere un albero bilanciato. La nuova implementazione, invece, contiene al suo posto il fattore di bilanciamento, inizialmente posto a 0. I metodi implementati, risultano essere tutti privati poichè vengono richiamati dai metodi della classe TreeMap.
In particolare, sono stati implementati i seguenti algoritmi:

- _is_balanced(p): data una position, restituisce True se il fattore di bilanciamento è compreso tra -1 e 1, false altrimenti. Dovendo soltanto accedere all'attributo, la sua complessità è pari ad O(1).

- _rebalance_insert: metodo privato di NewAVLTree che viene invocato all'aggiunta di un nodo all'interno dell'albero AVL. Questo metodo si occupa di ribilanciare l'albero e di reimpostare i fattori di bilanciamento dei nodi coinvolti. Il metodo _rebalance_insert(p) viene invocato ogni volta che nell'albero è eseguita un inserimento e riceve in ingresso la position p da aggiungere nell'albero. Innanzitutto si verifica se il nodo appena inserito ha un fratello. In tal caso, il fratello prima dell'inserimento non poteva avere un figlio a causa delle proprietà dell'avl. Pertanto, il padre del nodo inserito avrà fattore di bilanciamento pari a 0 e il metodo può terminare. In caso contrario si distinguono alcuni casi:

  - effettuato l'inserimento, si aggiorna il fattore di bilanciamento del padre che aumenterà o diminuirà di 1 a seconda che p risulti essere figlio destro o sinistro. Se il fattore di bilanciamento del padre diventa pari a zero allora anche in questo caso il metodo si conclude. In caso contrario si risale l'albero;
  - quando si risale l'albero nel caso in cui un nodo risulti avere un fattore di bilanciamento pari a -2 allora si analizza il bilanciamento del figlio sinistro così da individuare quale nipote di p passare come argomento alla funzione di ristrutturazione. Se il nodo p invece è sbilanciato a destra si ripete l'analisi appena citata ma scendendo nel sottoalbero destro. In entrambi i casi, una volta effettuata la ristrutturazione si aggiornano i fattori di bilanciamento. Questo porta la radice del sottoalbero ristrutturato ad avere fattore di bilanciamento pari a 0 e consente allora di smettere di risalire l'albero e alla funzione di terminare.

  Nel peggiore dei casi questa funzione comporta il dover risalire l'albero in tutta la sua altezza e quindi la complessità è O(log n).

- _rebalance_delete: metodo privato di NewAVLTree che viene invocato all'eliminazione di un nodo dell'albero AVL. Questo metodo si occupa di ribilanciare l'albero e di reimpostare i fattori di bilanciamento dei nodi coinvolti. Il metodo _rebalance_delete(p) viene invocato ogni volta che nell'albero è eseguita una cancellazione e riceve in ingresso la position p del padre del nodo eliminato. Quindi nel caso in cui p non sia None, cioè il nodo cancellato era la radice si distinguono vari casi:
  - se p è una foglia, allora suo fattore di bilanciamento viene posto a zero. Inoltre nel caso in cui il padre di p esista, se la sua chiave è maggiore di quella di allora il suo fattore di bilanciamento aumenta di 1. Viceversa, diminuisce di 1;
  - se p ha entrambi i figli ma non ha nipoti solo a destra allora il suo fattore di bilanciamento diminuisce di 1. Nel caso duale, entrambi i figli e mancano nipoti solo a sinistra, allora il fattore di bilanciamento di p aumenta di 1;
  - se p ha almeno il figlio destro, cioè il sinistro può essere assente, allora il suo balance factor viene incrementato di 1. Nel caso duale, il balance factor viene decrementato di 1.

  A questo punto, occorre verificare se, in seguito agli aggiornamenti del balance factor si siano verificati degli sbilanciamenti. Pertanto, inizia un ciclo che potrebbe potenzialmente scorrere tutto l'albero in cui:
  - se p è sbilanciato a sinistra, fattore di p pari a -2, si analizza il fattore di bilanciamento del figlio sinistro, così da individuare il nipote di p da inviare per argomento alla funzione di ristrutturazione;
  - se p è sbilanciato a destra, fattore di p pari a 2, si ripete il procedimento di prima andando però ad analizzare il fattore di bilanciamento del figlio destro. In entrambi i casi, una volta effettuate le rotazioni si aggiornano i fattori di bilanciamento e si risale l'albero;
  - se p non è sbilanciato ed ha fattore di bilanciamento uguale a 1 o a -1 allora il ciclo si interrompe e la funzione può terminare in quanto questo corrisponde ad una cancellazione che non comporta modifiche del fattore di bilanciamento che possono ripercuotersi sugli altri nodi dell'albero. Nel caso in cui il ciclo non si interrompa allora l'iterazione continua, risalendo l'albero da p al padre.

  La complessità di quest'algoritmo è nel caso peggiore uguale O(log n) in quanto occorre risalire l'albero in tutta la sua altezza.

3 Esercizio n.2

La classe Statistics permette di elaborare statistiche su di un dataset costituito da un insieme di coppie (key, value) presenti all'interno di un file. Per salvare tali valori, viene utilizzato un albero AVL implementato nell'esercizio precedente

che ha come valore una lista contenente i parametri frequency e total.Frequency è uguale al numero di occorrenze della chiave nel dataset, e total è la somma dei valori associati a tutte le sue occorrenze.
Nel costruttore, viene creata un'istanza di NewAVLTreeMap e viene richiamato il metodo privato _popola_albero che riceve in ingresso il nome del file. Tale metodo, dunque procede ad aprire il file e ad inserire le coppie (key, value) richiamando il metodo pubblico add. Se il file risulta essere corrotto o il path/nome non è corretto, verrà visualizzato un messaggio a video.
Come scelta implementativa, si da la possibilità di usare un dataset con chiavi di tipo int o str. La classe Statistics prevede un ulteriore attributo, entries, che tiene conto del numero di entries del dataset, infatti viene incrementato di uno ogni volta che un elemento viene aggiunto tramite il metodo add, e dunque rappresenta la somma delle frequenze dei nodi presenti.

- add(k,v): innanzitutto viene effettuato un controllo su v, valore della entry affinchè sia di tipo int. Dopodichè, si controlla se la chiave è già presente e nel caso non lo sia viene aggiunta altrimenti, si aggiornano i valori frequency e total. La sua complessità è pari a O(log n) dovuta alla ricerca della posizione in cui andare ad inserire la chiave.

- len() : restituisce il numero di chiavi presenti nell'albero. La sua complessità è O(1).

- _get_total(): restituisce il campo totale del nodo p. La sua complessità è 0(1).

- occurences(): restituisce la somma delle frequenze di tutti gli elementi presenti nell'albero. Se l'albero risulta essere vuoto, viene lanciata un'eccezione. La sua complessità è O(1) dovendo accedere all'attributo entries.

- average(): restituisce la media dei valori di tutte le occorrenze presenti nell'albero. Se l'albero risulta essere vuoto, viene lanciata un'eccezione. La sua complessità è O(n) dovendo attraversare tutti i nodi dell'albero.

- percentile(j=20): restituisce il j-imo percentile, per j = 1, ... 99, delle key, definito come la key k tale che il j per cento delle occorrenze del dataset hanno key minori o uguali di k. Se l'albero risulta essere vuoto o j non è un intero e non è compreso tra 1 e 99, viene lanciata un'eccezione. La sua complessità è O(n) dovendo attraversare tutti i nodi dell'albero, nel peggiore dei casi.

- median(): restituisce la mediana delle key presenti nel dataset, definita come la key tale che la metà delle occorrenze del dataset hanno key minori o

uguali della mediana. Essendo la mediana, un percentile con j=50, viene richiamato il metodo percentile e dunque ha la medesima complessità.

- mostFrequent(j): restituisce la lista delle j key più frequenti. Se l'albero risulta essere vuoto o j non è un intero e non è compreso tra 0 e il numero di nodi dell'albero, viene lanciata un'eccezione. La sua complessità è O(nlog(n)) perchè viene utilizzato il metodo sorted di Python che ha tale complessità.

4 Esercizio n.3

Si è deciso di utilizzare come funzione hash sha3_256. Si è adottata tale funzione per evitare che file differenti abbiamo lo stesso hash code e dunque vengano considerati come duplicati, visto che tale funzione hash prevede l'utilizzo di una chiave a 256 cifre e che ne utilizza 128 per controllare le collisioni.
Come prima operazione si controlla che il path del file, che si tenta di aprire, sia corretto, in caso non lo fosse viene lanciata un'eccezione. Siccome l'operazione di lettura file potrebbe essere onerosa per la memoria, in quanto bisogna leggerne tutto il testo, si preferisce usare un generatore, chunk_reader() che si occupa di prelevare il contenuto in chunk di 1024 bytes, valore di default. Ottenuti tutti i chunk di un file se ne calcola il valore hash corrispondente, infine, tale valore viene inserito come chiave in una ChainHashMap e nel bucket corrispondente il nome del file.
Alla fine dell'analisi in tutta la directory indicata, si otterrà una tabella hash in cui le chiavi sono gli hash values ottenuti da sha3_256() e in ogni bucket tutti i nomi dei file corrispondenti.
Nell'implementazione di ChainHashMap è già presente una funzione di hashing, in particolare una di tipo MAD. Per il nostro caso d'uso ChainHashMap avrebbe salvato come key l'intero contenuto di un file, usando il corrispondente hash value come discriminante per file dal diverso contenuto. Proprio per evitare tale carico della memoria è stata utilizzata la funzione di hashing sopra citata, presente nella libreria di Python, hahslib.
Il costo è di $O(n) * O(H)$, dove O(H) è il costo della funzione di hash, e n il numero dei file presenti nella directory.

5 Esercizio n.4

Per poter soddisfare la specifica della complessità, si è adottato l'algoritmo di Knuth-Morris-Pratt.
L'algoritmo si basa sulla pre-elaborazione del pattern per cercare corrispondenze dei prefissi del pattern nel pattern stesso attraverso la funzione di fallimento.In

particolare, la funzione di fallimento f (k) è definita come la lunghezza del prefisso più lungo di P che è un suffisso di P [1: k + 1]. Questo calcolo risulta molto importante per evitare di andare ad effettuare dei confronti che sappiamo già essere soddisfatti a differenza dell'algoritmo di forza bruta.

Il nostro algoritmo si occupa semplicemente di andare a modificare la stringa in cui ricercare, andando a concatenare ad essa, il suo prefisso di lunghezza m, ottenendo una stringa di lunghezza complessiva n+m. Ottenuto il risultato dall'algoritmo KMP, se esso restituisce -1 allora la sottostringa non è stata trovata e verrà restituito False altrimenti verrà restituito True.