ABR LIBRARY

Libreria per il progetto sugli alberi binari di ricerca sviluppata dal gruppo 11 composto da Francesco Borrelli, Alessandro Grieco e Camilla Zampella durante il corso di Laboratorio di Algoritmi e Strutture Dati dell’ a.a. 2017/2018.

Lo svolgimento della traccia è suddiviso in capitoli, ognuno dei quali rappresenta una funzionalità della traccia.

Sommario

[2 Struttura dati Tree 2](#_Toc482274329)

[3 Gestione dei duplicati in un ABR 2](#_Toc482274330)

[3.1 Strategia di risoluzione del problema 2](#_Toc482274331)

[3.2 Dettagli implementativi 3](#_Toc482274332)

[3.2.1 Funzione di inserimento 3](#_Toc482274333)

[3.2.2 Funzione di cancellazione 3](#_Toc482274334)

[4 Altezza media di alberi generati casualmente 4](#_Toc482274335)

[4.1 Dettagli implementativi 4](#_Toc482274336)

[4.1.1 Funzione di creazione albero casuale con n nodi 4](#_Toc482274337)

[4.2 Risultati dell’esperimento 4](#_Toc482274338)

[4.2.1 Valori risultanti dalle prove ripetute dell’esperimento 4](#_Toc482274339)

[4.2.2 Ricerca della funzione matematica dell’altezza media 5](#_Toc482274340)

[4.2.3 Ricerca del massimo valore di α 5](#_Toc482274341)

[4.3 Stima asintotica dell’altezza media 6](#_Toc482274342)

[5 Funzione Merge 7](#_Toc482274343)

[5.1 Dettagli implementativi 7](#_Toc482274344)

[5.1.1 Descrizione delle situazioni 8](#_Toc482274345)

[5.2 Descrizione della complessità asintotica 8](#_Toc482274346)

[6 Funzioni offerte dalla libreria 8](#_Toc482274347)

[6.1 Funzioni semplici 8](#_Toc482274348)

[6.1.1 Descrizione funzioni semplici 9](#_Toc482274349)

[6.2 Funzioni complesse 9](#_Toc482274350)

[6.2.1 Descrizione funzioni complesse 9](#_Toc482274351)

[6.3 HeapSort 9](#_Toc482274352)

[6.3.1 Descrizione di heapSort 9](#_Toc482274353)

[7 Esempio d’utilizzo della libreria 10](#_Toc482274354)

# Struttura dati Tree



La struttura dati Tree rappresenta un albero binario di ricerca e contiene al suo interno i seguenti campi

* info : contiene la chiave del nodo
* h : altezza del sottoalbero radicato in quel nodo
* right : figlio destro
* left : figlio sinistro

È stata modificata la struttura “di base” di un ABR aggiungendo il campo che tiene traccia dell’altezza, in quanto per alcuni algoritmi è necessaria questa informazione (si veda la funzione di stampa o il bilanciamento) ed è più opportuno accedervi in tempo costante piuttosto che visitare un intero albero per questa informazione.

# Gestione dei duplicati in un ABR

La prima funzionalità richiesta dalla traccia prevede la modifica degli algoritmi di gestione di un albero binario di ricerca al fine di poter gestire duplicati all’interno dell’albero.

## Strategia di risoluzione del problema

Dato che in un ABR il sottoalbero sinistro contiene i nodi più piccoli della radice e il sottoalbero destro quelli più grandi, l’inserimento del nodo già esistente avviene nel sottoalbero destro dell’albero che ha come radice il valore da inserire modificando così l’invariante di un ABR in modo tale da avere tutti i nodi maggiori o uguali alla radice nel sottoalbero destro.

## Dettagli implementativi

Di seguito l’implementazione che si adatta alla strategia di risoluzione del problema.

### Funzione di inserimento



È una funzione ricorsiva che permette l’inserimento di un nuovo nodo all’interno di un ABR o di creare un nuovo albero nel caso in cui quello preso in esame sia vuoto. Viene ricercata la posizione del valore da inserire sfruttando l’ordinamento delle chiavi all’interno di un ABR e, non appena si arriva ad una foglia, costruisce un nuovo nodo attaccandolo all’albero. La funzione updateH permette l’aggiornamento dell’altezza dell’albero nella chiamata locale.

Elenco dei parametri:

1. head : riferimento all’albero nel quale inserire il nuovo valore
2. val : il valore da inserire

Ritorna il riferimento alla radice della chiamata locale.

Post condizione : Il valore è inserito all’interno dell’albero

Complessità in termini di tempo : Ο(h) con h altezza dell’albero

Complessità in termini di spazio : richiede lo spazio aggiuntivo per la creazione di un nuovo nodo.

### Funzione di cancellazione

Sfruttando la strategia di risoluzione del problema su descritta non è stato necessario implementare una nuova funzione di cancellazione. Infatti questa funzione modifica l’albero che ha come radice il valore da eliminare, sostituendo questo con un nodo del sottoalbero destro che permette di rispettare ancora l’invariante di un albero binario di ricerca e, essendo i duplicati i candidati più adatti, ottenere così lo stesso risultato anche con gli alberi aventi chiavi duplicate.

# Altezza media di alberi generati casualmente

La traccia richiede di dimostrare in via sperimentale la relazione tra il numero di nodi e l’altezza media di alberi generati casualmente.

## Dettagli implementativi

Per ottenere l’altezza media è stata implementata una funzione di creazione di un albero binario di ricerca avente un numero fissato di nodi generati in maniera casuale. Creando una quantità casuale di alberi in un array è stata poi fatta la media delle loro altezze.

### Funzione di creazione albero casuale con n nodi



Funzione iterativa che genera un numero casuale e lo inserisce in un nuovo albero tante volte quanto il numero indicato in input. La funzione di inserimento è quella classica di un ABR.

Elenco dei parametri:

* nNodes : il numero di nodi da creare

Ritorna un riferimento all’albero creato

Post condizione: l’albero ha al massimo nNodes elementi. (dipende da quante volte la funzione rand() genera un duplicato).

## Risultati dell’esperimento

Dall’esperimento si evincono i seguenti risultati che tentano di esprimere l’altezza come funzione matematica del numero di nodi fissato.

### Valori risultanti dalle prove ripetute dell’esperimento

La tabella di seguito mostra l’altezza media della serie di alberi in base al numero di nodi generati casualmente.

|  |  |
| --- | --- |
| Numero di nodi generati | Altezza media della serie di alberi |
| 1 | 0 |
| 2 | 2 |
| 3 | 1,61 |
| 6 | 3,048 |
| 7 | 3,34 |
| 14 | 4,96 |
| 30 | 6,552 |
| 31 | 6,638 |
| 61 | 7,61 |
| 123 | 8,074 |
| 155 | 8 |
| 246 | 8 |
| 495 | 8 |
| 775 | 8,073 |
| 3876 | 8,061 |
| 19380 | 8,07 |
| 96901 | 8,067 |
| 480456 | 8,08 |
| 969010 | 8,068 |

### Ricerca della funzione matematica dell’altezza media

Dai risultati che si evincono dalla tabella si può notare che l’altezza rimane più o meno invariata con il crescere dei nodi assumendo valori simili alla funzione del logaritmo sul numero di nodi moltiplicata per una certa costante α. Data questa relazione è stato necessario calcolare il valore di α per ogni esperimento effettuato al fine di individuarne il valore massimo assunto per limitare superiormente l’altezza media delle serie di alberi generati casualmente.

### Ricerca del massimo valore di α

Di seguito la tabella che calcola α attraverso la formula dove *n* è il numero di nodi e *h* è l’altezza media della serie di alberi

|  |  |  |
| --- | --- | --- |
| n | h | α |
| 1 | 0 | 0 |
| 2 | 2 | 2 |
| 3 | 1,61 | 1,015797 |
| 6 | 3,048 | 1,179127 |
| 7 | 3,34 | 1,189732 |
| 14 | 4,96 | 1,302742 |
| 30 | 6,552 | 1,335265 |
| 31 | 6,638 | 1,339874 |
| 61 | 7,61 | 1,283146 |
| 123 | 8,074 | 1,162979 |
| 155 | 8 | 1,106358 |
| 246 | 8 | 1,017058 |
| 495 | 8 | 0,897078 |
| 775 | 8,073 | 0,841108 |
| 3876 | 8,061 | 0,676238 |
| 19380 | 8,07 | 0,566623 |
| 96901 | 8,067 | 0,487013 |
| 480456 | 8,08 | 0,428101 |
| 969010 | 8,068 | 0,405709 |

Il massimo valore assunto da α è 2 perché, al crescere di n, α assume valori sempre più piccoli. Attraverso il grafico che segue, è ancora più evidente l’andamento di *h* simile alla funzione .

## Stima asintotica dell’altezza media

Dai presupposti dei paragrafi precedenti possiamo descrivere l’altezza attraverso la seguente espressione asintotica

# Funzione Merge

È la funzione dedicata alla risoluzione del terzo punto della traccia. Ha lo scopo di effettuare l’unione insiemistica tra due alberi binari di ricerca dati in ingresso senza utilizzare memoria aggiuntiva.

## Dettagli implementativi

La funzione è implementata dal seguente codice



È una funzione ricorsiva che, dati due alberi in ingresso, unisce il primo con il secondo.  
Si basa su una ricerca della radice del primo albero nel secondo distinguendo così 5 casi innestati:

1. L’albero radicato in t1 è vuoto
2. L’albero radicato in t2 è vuoto
3. La chiave della radice del primo albero è maggiore di quella del secondo
4. La chiave della radice del primo albero è minore di quella del secondo
5. Le due radici hanno chiavi uguali

Elenco dei parametri:

* t1 : riferimento all’albero da unire
* t2 : riferimento all’albero in cui effettuare l’unione

Ritorna t2 aggiornato.

Pre condizione : i due alberi non devono avere chiavi duplicate.

Post condizione : l’albero ritornato contiene l’unione insiemistica dei due alberi e il riferimento al primo albero è inutilizzabile poiché potrebbe perdere dei collegamenti.

### Descrizione delle situazioni

Presi in esame i casi in cui si possono trovare i due alberi, di seguito sono descritte le azioni che svolge l’algoritmo su di essi:

* Se l’albero radicato in t1 è vuoto, banalmente si ritorna t2
* Se l’albero radicato in t2 è vuoto, si ritorna t1 che verrà attaccato ad un nodo che permette all’albero binario di ricerca di rispettarne le proprietà
* Se la chiave presente in t1 è maggiore di quella presente in t2, allora viene rotto il collegamento con il figlio sinistro di t1 e verranno effettuate due chiamate ricorsive:
  1. La prima sul sottoalbero destro di t2 e su t1
  2. La seconda su t2 e sul vecchio figlio sinistro di t1 (perché potrebbe avere chiave minore di quella di t2)
* Se la chiave presente in t1 è minore di quella presente in t2 , la situazione è speculare a quella descritta nel punto precedente
* Se le chiavi sono uguali, vengono effettuate le chiamate ricorsive sui rispettivi sottoalberi destri e sinistri ed infine viene deallocato il nodo t1.

## Descrizione della complessità asintotica

Dato che l’algoritmo in esame effettua una ricerca della chiave della radice t1 nell’albero t2 e, se non la trova, attacca tutto l’albero radicato in t1 è facile notare che si esegue una discesa lungo un percorso di t2 per ogni chiamata ricorsiva.

L’algoritmo di merge quindi costa un dove *h* è l’altezza dell’albero t2 in ingresso. Non occupa spazio aggiuntivo rendendo però l’albero radicato in t1 inutilizzabile.

# Funzione Rotation

È la funzione che risolve il quarto punto della traccia. Permette di effettuare un numero di volte dato in input una rotazione destra o sinistra.

## Dettagli implementativi

Le funzioni di rotazione destra e sinistra sono le stesse utilizzate per la gestione di un albero AVL e quindi rispettano tutte le proprietà di ordinamento di un albero binario di ricerca. La funzione “master” rotation è implementata come segue



È una funzione iterativa che decrementa il valore n passato in input fino a che non assume il valore 0 effettuando la rotazione dettata dal parametro direction.

Elenco dei parametri:

* t : riferimento all’albero da ruotare
* n : è il numero di volte che si deve ruotare l’albero
* direction : verso della rotazione codificato come intero ( 0 equivale a rotazione sinistra e 1 a quella destra)

Ritorna l’albero aggiornato in quanto le rotazioni modificano la radice.

Pre condizione : il parametro direction deve essere 0 o 1 in base alla direzione scelta. L’albero deve avere figlio destro/sinistro se si vuole effettuare una rotazione destra/sinistra.

Post condizione : la rotazione desiderata è stata effettuata n volte sull’albero. L’altezza dell’albero è aggiornata.

Errori generabili (codice presente nella variabile ABRERROR):

* -1 : il valore del parametro direction è diverso da 0 o 1
* -2 : impossibile applicare rotazione su di un albero vuoto
* -3 : impossibile effettuare rotazione destra perché il sottoalbero destro è vuoto
* -4 : impossibile effettuare rotazione sinistra perché il sottoalbero sinistro è vuoto

Costo in termini di tempo : dove n è il numero di rotazioni da effettuare dato che le rotazioni singole hanno un costo asintotico costante.

# Bilanciamento tramite rotazioni