“Lose Weight” project

Libreria per il progetto sui grafi 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.

Sommario

[2 Struttura dati Tree 2](#_Toc482277408)

[3 Gestione dei duplicati in un ABR 2](#_Toc482277409)

[3.1 Strategia di risoluzione del problema 2](#_Toc482277410)

[3.2 Dettagli implementativi 2](#_Toc482277411)

[3.2.1 Funzione di inserimento 3](#_Toc482277412)

[3.2.2 Funzione di cancellazione 3](#_Toc482277413)

[4 Altezza media di alberi generati casualmente 4](#_Toc482277414)

[4.1 Dettagli implementativi 4](#_Toc482277415)

[4.1.1 Funzione di creazione albero casuale con n nodi 4](#_Toc482277416)

[4.2 Risultati dell’esperimento 4](#_Toc482277417)

[4.2.1 Valori risultanti dalle prove ripetute dell’esperimento 4](#_Toc482277418)

[4.2.2 Ricerca della funzione matematica dell’altezza media 5](#_Toc482277419)

[4.2.3 Ricerca del massimo valore di α 5](#_Toc482277420)

[4.3 Stima asintotica dell’altezza media 6](#_Toc482277421)

[5 Funzione Merge 6](#_Toc482277422)

[5.1 Dettagli implementativi 7](#_Toc482277423)

[5.1.1 Descrizione delle situazioni 8](#_Toc482277424)

[5.2 Descrizione della complessità asintotica 8](#_Toc482277425)

[6 Funzione Rotation 8](#_Toc482277426)

[6.1 Dettagli implementativi 8](#_Toc482277427)

[7 Bilanciamento tramite rotazioni 10](#_Toc482277428)

# Strutture dati utilizzate

## Per la creazione del grafo

Per creare il grafo sono state utilizzate le seguenti tre strutture:

* La struttura Graph , la quale contiene al suo interno una variabile intera che indica il numero di nodi da cui è formato il grafo e un puntatore alla struttura Node.
* La struttura Node, la quale contiene una variabile float che indica l’altezza del nodo e un puntatore alla struttura edge.
* La struttura Edge, la quale contiene un intero che indica il nodo di destinazione dell’arco, una variabile float che indica il peso dell’arco e un puntatore al successivo elemento della lista di adiacenza.

## per la visita del grafo

Si è utilizzata la struttura Visit, la quale contiene un array per la memorizzazione dei colori, un array per la memorizzazione dei predecessori e un array di float per la memorizzazione delle distanze.

## strutture di supporto

Sono state inoltre utilizzate due strutture di supporto:

* Una lista puntata al fine di rappresentare una serie di nodi ordinati.
* Una coda a priorità di supporto all’algoritmo di Dijkstra per i percorsi minimi in caso in cui vi siano due nodi aventi stessa altezza.

# ricerca del percorso minimo

La traccia richiedeva di trovare il percorso minimo in un grafo tale da permettere a Matteo di percorrere la prima parte di strada in salita e la seconda parte in discesa, prima in un grafo in cui non potessero esservi due punti alla stessa altezza e poi in un grafo in cui possano esservi nodi alla stessa altezza. Per risolvere i punti del problema sono state proposte le seguenti soluzioni.

## percorso minimo in un grafo privo di nodi alla stessa altezza

Per risolvere il primo punto della traccia si è suddiviso il problema in due sottoproblemi: inizialmente si cerca il percorso minimo in salita partendo dalla sorgente, poi si cerca il percorso minimo in salita partendo dalla destinazione e infine si uniscono le due visite.

Poiché ogni visita genera un sottografo orientato aciclico è possibile generare un ordinamento topologico per ogni sottografo e settare la distanza minima dalla sorgente procedendo per topologia.

## Dettagli implementativi

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

### Funzione uphillVisit

È la funzione che sfrutta l’ordinamento topologico per settare le distanze tra i nodi all’interno del vettore dist della struct visit.

Elenco dei parametri:

1. g: grafo su cui devo eseguire la visita
2. s: nodo sorgente da cui deve partire la visita

Ritorna un riferimento alla struttura visit.

Pre condizione: la sorgente sia un nodo contenuto all’interno del grafo.

Post condizione : nel vettore dist di Visit sarà contenuto il percorso minimo

Complessità in termini di tempo : O(v+e)

Complessità in termini di spazio :

### Funzione DFSVisitUphillList

Funzione che crea l’ordinamento topologico del sottografo al fine di ottenere il percorso minimo

Elenco dei parametri:

1. g: grafo di cui si deve fare l’ordinamento topologico
2. v: riferimento della struttura visit
3. l: riferimento ad una lista puntata
4. s: nodo sorgente da cui parte la visita

Ritorna una lista contenente l’ordinamento topologico del sottografo.

Pre condizione: la sorgente deve essere un nodo contenuto all’interno del grafo

Post condizione: all’interno di l sarà contenuto l’ordinamento topologico.

Complessità in termini di tempo: O(v+e)

Complessità in termini di spazio:

### Funzione printPath

Funzione che serve a stampare un percorso dopo una visita

Elenco dei parametri:

1. v: riferimento della struttura visit
2. s: nodo sorgente

Complessità in termini di tempo:  
Complessità in termini di spazio:

### Funzione pathGenerator

### Funzione pathExtender

### Funzione initializeVisit:

Funzione che si occupa di allocare e inizializzare gli elementi all’interno della struttura visit prima di effettuare una visita del grafo.

Elenco dei parametri:

1. g: grafo su cui verrà eseguita la visita

Post condizione: restituisce un riferimento alla struttura visit

### Funzione loseWeightPathPrinter

### Funzione hasDuplicates

Funzione che controlla se nel grafo vi siano due nodi aventi la stessa altezza.

Elenco dei parametri:

1. g: grafo sui cui bisogna controllare che non vi siano nodi aventi la stessa altezza

Post condizione: restituisce un intero che sarà uguale a 0 se la funzione non contiene duplicati e 1 altrimenti.

Complessità in termini di tempo: O(V)

# 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

Funzione che permette il bilanciamento tramite rotazioni di un albero binario di ricerca avente nodi di tipo Tree.

## Dettagli implementativi

L’implementazione della funzione è la seguente



È una funzione iterativa che, dato un albero in ingresso, esegue gli algoritmi di ribilanciamento a destra o a sinistra in base a quale sottoalbero viola la condizione di bilanciamento. A differenza degli AVL per il quale è necessario effettuare al più una rotazione, è stato necessario effettuare un ciclo che termina solo quando l’albero d’ingresso è bilanciato.

Gli algoritmi di ribilanciamento destro e sinistro utilizzano la funzione rotation, descritta nel capitolo 6, con numero di rotazioni da effettuare pari ad uno per ogni rotazione. Di seguito il codice sia del bilanciamento destro che quello sinistro



Il costo asintotico in termini di tempo è un *Ο(h)* dove h è l’altezza dell’albero (caso dell’albero degenere).

# Stampa grafica di un albero

Le funzioni seguenti sono state implementate in modo tale da poter stampare un albero graficamente come richiesto dal punto opzionale della traccia.

## Dettagli implementativi

L’idea implementativa è quella di calcolare inizialmente il numero di spazi da dover stampare per ottenere una stampa della radice e, successivamente stampare per livelli i nodi dell’albero



La funzione printBst utilizza la funzione print che serve appunto a stampare i vari livelli.

La funzione ricorsiva di appoggio print verifica prima se il nodo preso in analisi è all’altezza desiderata e, se rispetta questo canone, stampa spazi e il valore della chiave.

La stampa non è ottimizzata per essere vista in console in quanto essa non è sufficientemente grande per poter contenere tutti i caratteri utili alla rappresentazione dell’albero.