Ingegneria degli Algoritmi

Antonio D'Orazio
0242178

Prima prova pratica – Traccia 1 (Albero binario Con Lazy Deletion)

Introduzione

Ho modificato i file BinaryTree.py e dictBinaryTree.py, rinominati rispettivamente in LazyBinaryTree.py e LazyDictionary.py.

Ho organizzato le funzioni nelle varie classi nel seguente modo:

Ho scelto String come tipo di dato per la chiave in modo da consentire l'inserimento di stringhe oltre che di numeri. Tutte le chiavi sono convertite in stringhe in modo da rendere corretto l'ordinamento.

La classe LazyDictionary è quella al livello più alto per l'utente, le sue funzioni restituiscono immediatamente i valori del dizionarioLe classi LazyBinaryTree e BinaryNode sono ad un livello di implementazione inferiore: gestiscono i dati a livello di BinaryNode e non di output diretto.

Esempio: Dalla classe LazyDictionary chiamo il metodo search(self, key). Questo metodo chiamerà LazyBinaryTree.searchNode(self, key) che restituirà un BinaryNode. Da qui Search si occuperà di elaborare il BinaryNode e di restituire un risultato.

Poiché la logica è stata affidata completamente alla classe LazyBinaryTree, attraverso questa implementazione è possibile riciclare la classe LazyDictionary con alberi con implementazioni diverse.

Descrizione dei metodi ed analisi del tempo teorico

Classe BinaryNode

Costruttore	Assegna i valori info, father, leftSon e rightSon. In particolare, info è una lista di tipo [chiave, valore, attivo]. Attivo è un valore booleano per implementare la Lazy Deletion. Se attivo = true l'elemento è visibile. Se attivo = false l'elemento è cancellato. **Tempo:** O(1) poiché esegue delle semplici assegnazioni e la lista ha grandezza costante
toString(self)	Semplice metodo che restituisce una stringa contenente tutte le informazioni del nodo. **Tempo:** O(1) poiché costruisce semplicemente una stringa

Classe LazyBinaryTree

Costruttore	Stabilisce la radice root dell'albero. Di default: albero vuoto. **Tempo:** O(1) poiché l'assegnamento è su un singolo nodo
insert(self, key, value):	Inserisce coppia chiave-valore nell'albero secondo la seguente logica: Creo una tripla di valori [chiave, valore, True (valore di attivo)] Creo prima un Nodo e poi un Albero su questa tripla di valori Se l'albero su cui vogliamo effettuare l'inserimento è vuoto, allora la radice dell'albero diventa quella dell'albero appena creato Altrimenti scorro tutti i nodi dell'albero, sfruttando le proprietà dell'ordinamento, finché non trovo un nodo nullo o non attivo. Se il nodo è nullo: inserisco il nuovo nodo come figlio destro o sinistro del nodo nullo Se il nodo è disattivo: sostituisco le sue informazioni con quelle del nuovo nodo Insert restituisce True se il nodo è stato inserito da zero, False se ne è stato sovrascritto uno già presente (perché disattivo o perché con la stessa chiave) **Tempo:** O(1) nel caso migliore, quando la radice è nulla o viene sovrascritta. O(logn) nel caso peggiore, quando si inserisce una nuova foglia o ne viene sovrascritta un'altra, poiché corrisponde all'altezza di un albero binario [Ricontrolla]
InsertAsLeftSubTree(self, father, subtree):	inserisce la radice di un sottoalbero come figlio sinistro del nodo father. Semplicemente assegna all'attributo father.leftSon il sottoalbero. **Tempo:** : O(1) poiché si limita ad assegnare al valore leftSon un nodo
InsertAsRightSubTree(self, father, subtree):	inserisce la radice di un sottoalbero come figlio destro del nodo father. Semplicemente assegna all'attributo father.rightSon il sottoalbero. **Tempo:** O(1) poiché si limita ad assegnare al valore rightSon un nodo
delete(self, key):	metodo per la cancellazione. Cancella dall'albero il nodo con chiave key. Prima di tutto effettua una search per ottenere il nodo con chiave key. Dopodiché se il nodo ha 0 o 1 figli la funzione può chiamare il metodo oneSonDeletion, che imposta a False il campo attivo del nodo senza conseguenze per i figli. Se invece il nodo ha due figli, seguo il seguente algoritmo: Cerco il predecessore del nodo (il figlio con chiave piu grande) sfruttando maxKeySon; Scambio il contenuto dei due nodi; Elimino con oneSonDeletion il nodo da eliminare, poiché ora è in una posizione sicura Delete restituisce True se il nodo è stato eliminato, False se non era presente **Tempo:** O(logn) (dato da search) + tempo O(1) dato da oneSonDeletion + O(logn) dato da maxKeySon = O(logn) nel caso peggiore. [Ricontrolla + caso migliore]

oneSoneDeletion: [attenzione mancano parametri]	implementa la lazy deletion: si limita ad impostare a False il campo active del nodo **Tempo:** O(1) poiché si limita ad una assegnazione
search(self, key):	restituisce il nodo corrispondente alla chiave key in ingresso. Dopo aver verificato che l'albero non sia vuoto, esegue il seguente algoritmo: assegno alla variabile curr il nodo radice dell'albero; finché il nodo curr non è nullo, confronto la sua chiave secondo tre casi: se la chiave è la stessa inserita in input, restituisco curr se il nodo attivo, None se altrimenti se la chiave in input è minore o maggiore della chiave di curr, assegno a curr rispettivamente curr.leftSon o curr.rightSon se alla fine curr è un nodo nullo e non ho trovato nulla, restituisco None **Tempo:** O(1) nel caso peggiore, quando la radice è nulla o la radice corrisponde al nodo da cercare. Tempo O(logn) nel caso peggiore, che corrisponde all'altezza dell'albero e si verifica quando l'elemento si trova nelle foglie più in profondità
key(self, node):	metodo di appoggio, restituisco la chiave del nodo (None se nodo è nullo) **Tempo:** O(1) poiché si limitano a restituire uno o più campi di un nodo
value(self, node):	metodo di appoggio, restituisco il valore del nodo (None se nodo è nullo) **Tempo:** O(1) poiché si limitano a restituire uno o più campi di un nodo
isActive(self, node):	metodo di appoggio, restituisco lo stato del nodo (False se nodo è nullo) **Tempo:** O(1) poiché si limitano a restituire uno o più campi di un nodo
info(self, node):	metodo di appoggio, restituisco le informazioni [chiave, valore, attivo] (None se nodo è nullo) **Tempo:** O(1) poiché si limitano a restituire uno o più campi di un nodo
maxKeySon(self, root):	restituisco il nodo figlio con chiave più grande: scorro nei sottoalberi destri e restituisco il nodo più in profondità. **Tempo:** O(logn) nel caso peggiore, corrispondente all'altezza dell'albero.
DFS(self):	Restituisco una lista di BinaryNode.info ordinati secondo il criterio della visita in profondità. Per farlo prima di tutto inizializzo una pila inserendo la radice (se non nulla). A questo punto, finché lo stack non è vuoto, eseguo questi passaggi: estraggo dalla pila l'ultimo elemento in ordine di inserimento e, se marcato come attivo, lo inserisco nella lista da restituire; **Tempo:** visita l'albero in O(n) iterazioni occupando spazio O(n) (l'array da restituire)

BFS(self):	Restituisco una lista di BinaryNode.info ordinati secondo il criterio della visita in ampiezza. Per farlo prima di tutto inizializzo una coda inserendo la radice (se non nulla). A questo punto, finché la coda non è vuota, eseguo questi passaggi: estraggo dalla coda il primo elemento in ordine di inserimento a, se marcato come attivo, lo inserisco nella lista da restituire; inserisco nella coda, se non nulli, il figlio destro e sinistro del nodo estratto al punto uno **Tempo:** visita l'albero in O(n) iterazioni occupando spazio O(n) (l'array da restituire)
stampa(self):	Consente di stampare l'albero completo, compresi gli elementi disabilitati, al fine di analizzarne visivamente la gerarchia. Sfrutta la tecnica della visita in profondità **Tempo:** O(n), corrispondente al tempo della visita in profondità

Classe LazyDictionary

Costruttore	inizializza un Dizionario costruendo un albero binario di ricerca che implementa la lazy deletion, insieme alla lunghezza del dizionario. Supporta la creazione di un dizionario partendo da una lista precedentemente creata, con sintassi [[chiave1, valore1], [chiave2, valore2], [.. , ..], [chiaveN, valoreN]]. Per implementare questa funzionalità è stato sufficiente scorrere ogni elemento della lista, richiamando il metodo per l'inserimento nel dizionario per ogni coppia **Tempo:** Costruisce un Dizionario in tempo O(nlogn), dove n è la grandezza della lista in input. Questo perché corrisponde ad effettuare n volte la procedura Insert che ha tempo O(logn)
add(self, key, val):	Aggiunge una voce al dizionario. Prende in input chiave e valore da inserire e richiama il metodo insert dell'albero. Se insert ha inserito un nuovo nodo (restituendo True) incremento self.length, altrimenti non viene modificato **Tempo:** O(1) nel caso migliore ed O(logn) nel caso peggiore, poiché chiama insert ed eventualmente esegue un incremento
remove(self, key, val):	Aggiunge una voce al dizionario. Prende in input chiave e valore da inserire e richiama il metodo delete dell'albero. Se delete restituisce True decremento self.length **Tempo:** O(logn) nel caso peggiore e O(1) nel caso migliore, poiché effettua una Delete ed eventualmente esegue un decremento [Grammatica]
get(self, key):	Restituisce il valore del nodo con chiave key. Chiama il metodo search dell'albero e dal nodo che ottiene restituisce value(nodo) **Tempo:** O(1) nel caso migliore ed O(logn) nel caso peggiore, poiché effettua una Search ed una Value
size(self):	restituisce il numero di elementi nel dizionario, presenti nella variabile self.length **Tempo:** O(1) poiché restituisce il valore di un attributo

allPairs(self):	restituisce la lista di coppie [chiave, valore] degli elementi nel dizionario. Per farlo effettua una visita chiamando la funzione DFS, dopodiché ne restituisce il risultato. **Tempo:** O(n) poiché effettua una visita DFS
keys(self):	restituisce la lista di tutte le chiavi degli elementi del dizionario. Per farlo effettua una visita chiamando la funzione DFS, dopodiché scorrendo il risultato salva tutte le chiavi in una nuova lista, per poi restituirla **Tempo:** O(n) poiché ho O(n) + O(n), rispettivamente per la visita DFS e poi per estrarre ogni chiave
values(self):	restituisce la lista di tutte i valori degli elementi del dizionario. Per farlo effettua una visita chiamando la funzione DFS, dopodiché scorrendo il risultato salva tutti i valori in una nuova lista, per poi restituirla **Tempo:** O(n) poiché ho O(n) + O(n), rispettivamente per la visita DFS e poi per estrarre ogni valore

Contact Information

[If you want to add any important info about the contacts that follow, you can do that here. If not, just click this placeholder and press Delete to remove it.]

[Client Project Manager]
Office: [Office Phone]
Mobile: [Cell Phone]
Email: [Email address]

[Client Project Champion]
Office: [Office Phone]
Mobile: [Cell Phone]
Email: [Email address]