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.pyHo 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

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.

Classe LazyBinaryTree

Costruttore	Stabilisce la radice root dell'albero. Di default: albero vuoto.
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)
InsertAsLeftSubTree(self, father, subtree):	inserisce la radice di un sottoalbero come figlio sinistro del nodo father. Semplicemente assegna all'attributo father.leftSon il sottoalbero.
InsertAsRightSubTree(self, father, subtree):	inserisce la radice di un sottoalbero come figlio destro del nodo father. Semplicemente assegna all'attributo father.rightSon il sottoalbero.
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
oneSoneDeletion: [attenzione mancano parametri]	implementa la lazy deletion: si limita ad impostare a False il campo active del nodo
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
key(self, node):	metodo di appoggio, restituisco la chiave del nodo (None se nodo è nullo)
value(self, node):	metodo di appoggio, restituisco il valore del nodo (None se nodo è nullo)
isActive(self, node):	metodo di appoggio, restituisco lo stato del nodo (False se nodo è nullo)

info(self, node):	metodo di appoggio, restituisco le informazioni [chiave, valore, attivo] (None se nodo è nullo)
maxKeySon(self, root):	restituisco il nodo figlio con chiave più grande: scorro nei sottoalberi destri e restituisco il nodo più in profondità.
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;

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.

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]