**Ingegneria degli algoritmi – AA 2017 2018 – Prima Prova in Itinere**

**Studente:** Foderaro Salvatore

**Numero di Matricola:** 0214381

**Problema - Binary search tree con lazy deletion**

### Descrizione algoritmo

Per l’implementazione del dizionario, ho dapprima definito la funzione **lazyInsert**, simile alla funzione **insert** già presente nel dizionario degli alberi binari. L’unica differenza sta nell’aggiunta del terzo campo nella lista delle informazioni del nodo, impostato di default a **0**. Ho definito questo terzo elemento della lista come **Valore di eliminazione del nodo**: se uguale a **1**, il nodo risulta eliminato, se uguale a **0**, no.

Ho inoltre definito le seguenti funzioni aggiuntive:

* **deletionValue(nodo):** restituisce il valore di eliminazione del nodo;
* **setDeleted(nodo):** segna il nodo come eliminato, andando ad impostare ad **1** il valore di eliminazione;
* **setUnDeleted(nodo):** segna il nodo come non eliminato, andando ad impostare a **0** il valore di eliminazione;
* **isDeletedOr(nodo):** restituisce True se il nodo risulta eliminato, altrimenti False;
* **setValue(nodo,valore):** permette di modificare il valore del nodo.

Per la risoluzione del problema ho definito le funzioni richieste:

1. **lazySearch(chiave):** la funzione cerca il nodo associato alla chiave e, se presente, controlla il suo valore di eliminazione; se uguale a **0** restituisce il nodo, altrimenti restituisce **None**;
2. **boolDeleteLazy(chiave):** tramite **lazySearch**, la fuzione effettua il controllo sul nodo associato alla chiave. Se la funzione chiamata restituisce il nodo, **boolDeleteLazy** lo segna come eliminato e restituisce **True**, altrimenti, restituisce **False**;
3. **insertLazyControl(chiave,valore):** prima di inserire la coppia **(chiave, valore)**nell’albero, la funzione controlla se esiste già un nodo associato alla chiave da inserire ed eventualmente il suo valore di eliminazione. Se il nodo è già presente ed il suo valore di eliminazione è **1**, imposta il valore passato alla funzione come nuovo valore del nodo ed il valore di eliminazione a **0**. Al contrario se nell'albero non è presente nessun nodo associato alla chiave, o presente ma con valore di eliminazione **0**, effettua il normale inserimento tramite **lazyInsert**.

### Utilizzo Lazy Deletion

L’uso della **Lazy Deletion** è preferibile nei casi in cui si vuole indicare un nodo come disabilitato, mantenendo tutte le informazioni (valore) ad esso associati. Tramite le funzioni **boolDeleteLazy** e **InsertLazyControl** è possibile segnare il nodo come eliminato e successivamente riabilitarlo. In questo modo le operazioni sul nodo hanno il solo costo di aggiornamento dell'informazione relativa all'eliminazione, ed è possibile evitare eliminazioni ed inserimenti continui dei nodi.

### Tempo di esecuzione teorico

Trattandosi di un **Albero binario di ricerca**, tutte le operazioni hanno costo **O(h)**, dove h rappresenta l'altezza dell'albero. Le funzioni implementate includono le operazioni di confronto del valore di un array ed eventualmente la sua modifica, il tutto al costo di **O(1)**.

Dunque, le operazioni di un **Binary Search Tree con Lazy Deletion**, implementata come illustrato in questa breve relazione, hanno costo **O(h)**.

### Tempo di esecuzione sperimentale

* # di esecuzioni della funzione (sorgente disponibile in **demoProgetto.py**).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  | **#10.000** | **#100.000** | **#1.000.000** | | **boolDeleteLazy** | 0.58s | 7.93s | 100.24s | | **lazySearch** | 0.54s | 7.50s | 94.59s | | **insertLazyControl** | 0.96s | 12.58s | 161.88s | |