Relazione sul Secondo Esercizio

Modalità di lettura dati: Esattamente come per i gli algoritmi di sorting e per i grafi, anche per gli alberi la lettura da file avviene una sola volta e i dati vengono immagazinati in un array di struct. Questa feature mi è tornata particolarmente utile nello sviluppo di questo secondo esercizio perché:

- Il value di ogni nodo semplicemente conterrà il puntatore alla rispettiva struct;
- Per accedere a una riga random del file mi basta generare un indice accettabile dall' array di struct e non lanciare ulteriori letture da file;
- Una volta concluso il lavoro con una tipologia di chiave mi basta distruggere la struttura interna dell'albero e ricrearne uno nuovo da capo, ordinato secondo una tipologia di chiave diversa;
- Per liberare le risorse, una volta distrutto l'albero, mi basta lanciare un ciclo di free() sull' array di struct.

Struttura fisica del' esercizio: Il motivo per cui l'esercizio è diviso in 2 sottocartelle separate, è che un'implementazione comune sarebbe stata troppo confusionaria. Si sarebbe potuto implementarli in un'unica cartella ma ciò avrebbe comportato un raddoppio di tutte le funzioni ausiliarie o addirittura una triplicazione se si fosse scelto di creare una funzione che, in base a un parametro passato, avesse scelto di lanciare la stessa funzione adattata per i gli alberi rosso neri o gli alberi binari. (Tale necessità di differenziazione è dovuto al fatto che le struct dei nodi sono diverse, e quindi lo saranno anche i cast e le dereferenziazioni).

Nodo sentinella: nell' implementazione degli alberi rosso neri il nodo sentinella è una variabile globale esterna alla struttura dell'albero. Ho scelto di rappresentarla cosi per potervi accedere in qualunque momento, anche senza la necessità di dover richiedere un puntatore all' albero come parametro.

Confronto sulle performance dei due algoritmi: qui si seguito sono riportati i tempi di esecuzione delle operazioni richieste sia per mezzo di alberi binari che per mezzo di alberi rosso-neri. I tempi riportati sono quelli raccolti sul mio pc, seguiti da quelli raccolti sul pc del' università.

BST: (*) BinaryTree (STRING Key)
(+) Insert data: Completed in 7.24700 seconds **VS** 5.33000 seconds
(+) Accessing to the tree: Completed in 0.07446 seconds **VS** 0.03000

(*) BinaryTree (INT Key)
(+) Insert data: Completed in 97.68912 seconds **VS** 42.57000 seconds
(+) Accessing to the tree: Completed in 5.54550 seconds **VS** 2.39000

(*) BinaryTree (Float Key)
(+) Insert data: Completed in 115.85930 seconds **VS** 50.67000 seconds
(+) Accessing to the tree: Completed in 6.30734 seconds **VS** 2.75000 seconds

RBT: (*) RedBalckTree (STRING Key)

(+) Insert data: Completed in 6.55095 seconds **VS** 5.62000 seconds
(+) Accessing to the tree: Completed in 0.15492 seconds **VS** 0.05000

(*) RedBalckTree (INT Key)
(+) Insert data: Completed in 39.21778 seconds **VS** 19.11000 seconds
(+) Accessing to the tree: Completed in 0.15311 **VS** 0.06000 seconds

(*) RedBalckTree (Float Key)
(+) Insert data: Completed in 55.28119 seconds **VS** 27.86000 seconds
(+) Accessing to the tree: Completed in 0.14008 seconds **VS** 0.06000 seconds