ΑΝΑΦΟΡΑ 2ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Δομές δεδομένων και αρχείων

Μιχάλης Γαλάνης
2016030036

ΣΚΟΠΟΣ

Σκοπός της δεύτερης εργαστηριακής άσκησης είναι η εξοικείωσή μας με τις λειτουργίες του δυαδικού δέντρου και η σύγκριση της ταχύτητάς των αλγορίθμων τους στη κύρια μνήμη (πρώτο μέρος) και στη δευτερεύουσα μνήμη (δεύτερο μέρος).

ΕΙΣΑΓΩΓΗ

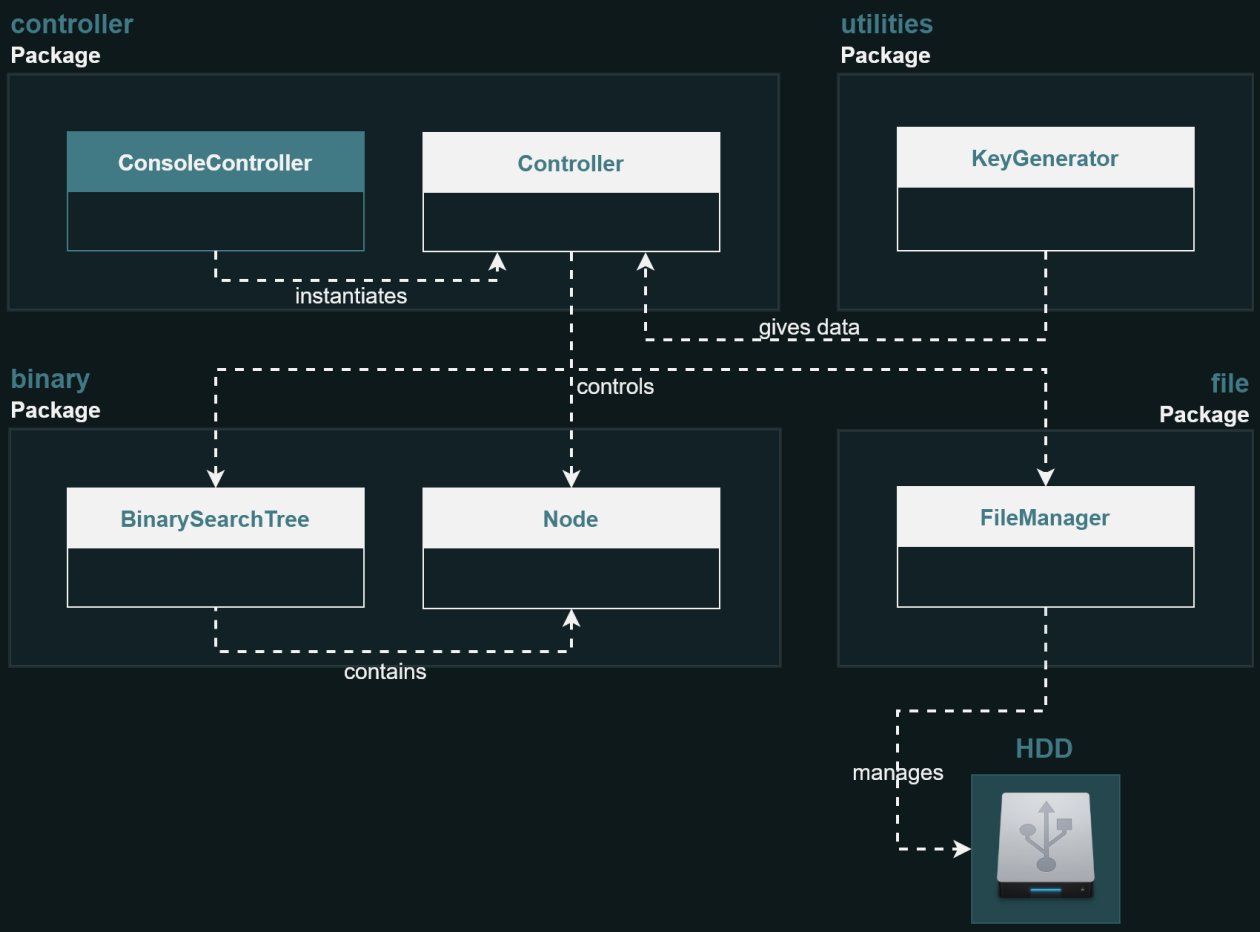
Τα Δυαδικά δέντρα αναζήτησης φημίζονται για την απόδοσή τους στις περισσότερες λειτουργίες τους. Η χρονική πολυπλοκότητά τους για προσπέλαση, αναζήτηση, εισαγωγή και διαγραφή στοιχείων είναι **O(log (n))** για αυτό και χρησιμοποιούνται ευρέως σε εφαρμογές με μεγάλες δομές δεδομένων.

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα αποτελείται απο 4 πακέτα και 6 κλάσεις οργανωμένα με τον τρόπο που φαίνεται στο παρακάτω σχήμα και αναδεικνύεται η σχέση και ο ρόλος των κλάσεων/πακέτων:



ΕΠΕΞΗΓΗΣΗ ΕΡΩΤΗΜΑΤΩΝ

ΜΕΡΟΣ Α – Δυαδικό δέντρο έρευνας

Η κλάση **BinarySearchTree** βασίζεται στα **nodes** (όπου το καθένα περιέχει **leftChild** και **rightChild**) και υποστηρίζει τις παρακάτω λειτουργίες:

- **Εισαγωγή τυχαίου κλειδιού**
  Η μέθοδος εισαγωγής στοιχείου στο δυαδικό δέντρο αναζήτησης δέχεται ως όρισμα το κλειδί, κατασκευάζει ένα νέο node **newNode** και συγκρίνοντας επαναληπτικά την τιμή του κλειδιού με την τιμή ενός **focusNode (currentNode)**, θα μετακινηθεί ο **currentNode** στο κατάλληλο παιδί. Μόλις ο currentNode πάρει την τιμή **null**, στο σημείο εκείνο θα τοποθετηθεί το **newNode.**

- **Αναζήτηση τυχαίου κλειδιού**
  Όπως και με την εισαγωγή στοιχείου, κι αυτή η μέθοδος δέχεται ένα κλειδί το οποίο ψάχνει με τη βοήθεια του **currentNode** συγκρίνοντας το key με την τιμή του currentNode και μετακινώντας το ανάλογα μέχρι να βρεθεί το κλειδί ή να τελειώσουν τα στοιχεία του δέντρου.

Οι δύο παραπάνω μέθοδοι επιστρέφουν τον αριθμό των συγκρίσεων που προέκυψαν για την εισαγωγή ή αναζήτηση κλειδιού αντίστοιχα.

- **Διάσχιση inorder**
  Με χρήση αναδρομής το δυαδικό δέντρο αναζήτησης μετατρέπεται εύκολα σε ένα array κλειδιών. Πιο συγκεκριμένα: δέχεται αρχικά ένα node το οποίο ελεγχεται για το αν είναι κενό, στη συνέχεια αυτοκαλούμε τη συνάρτηση με όρισμα το αριστερό παιδί του node, αντιγράφουμε στο array το στοιχείο και ξανακαλούμε τη συνάρτηση με όρισμα το δεξί παιδί του node.

- **Αναζήτηση εύρους τιμών**
  Η αναζήτηση εύρους τιμών εκτελείται επίσης με αναδρομή χρησιμοποιώντας διάσχιση inorder.

Με τα παραπάνω δεδομένα, ζητήθηκε λοιπόν να μετρηθεί ο **μέσος αριθμός συγκρίσεων** εκτελώντας τις παρακάτω λειτουργίες:

- Εισαγωγή **N = 10^4** τυχαίων κλειδιών
- Αναζήτηση **N = 100** τυχαίων κλειδιών
- **N = 100** τυχαίες αναζητήσεις έυρους τιμών για **offset = 100** και **1000** αντίστοιχα **(x2)**.

Ζητήθηκε επίσης η διάσχιση inorder και η αντιγραφή του δέντρου σε ένα ταξινομημένο αρχείο χρησιμοποιώντας μεθόδους της προηγούμενης εργαστηριακής άσκησης.

Εκτελώντας το πρόγραμμα, εκτυπώνονται τα παρακάτω αποτελέσματα:

```
CALCULATION 1 - Insertion of 10^4 keys into the binary tree.
-----------------------------------------------------------
10000 keys have been inserted into the binary tree.
Total comparisons: 153160
Average comparisons per insertion: 15

CALCULATION 2 - Search of 100 keys in the binary tree.
-----------------------------------------------------------
100 keys have been searched in the binary tree.
Total comparisons: 1592
Average comparisons per search: 15

CALCULATION 3A - Search of 100 keys with offset (+100) in the binary tree.
-----------------------------------------------------------
100 keys have been searched in the binary tree with offset (+ 100).
Total comparisons: 522810
Average comparisons per search: 5228

CALCULATION 3B - Search of 100 keys with offset (+1000) in the binary tree.
-----------------------------------------------------------
100 keys have been searched in the binary tree with offset (+ 1000).
Total comparisons: 5021776
Average comparisons per search: 50217
```

ΜΕΡΟΣ Β – Ταξινομημένο αρχείο

Το δεύτερο μέρος επικεντρώνεται στη λειτουργία της **αναζήτησης** κλειδιών.

Για την ανάλυση του δεύτερου μέρους χρησιμοποιήθηκε η κλάση **"FileManager"** που κατασκευάστηκε στην προηγούμενη άσκηση. Αυτή περιέχει όλες τις μεθόδους δημιουργίας/εγγραφής/ανάγνωσης που χρειάζονται για τη διαχείρηση αρχείων. Όλα γίνονται με βάση τις μεταβλητές **byte[512] page , int[128] buffer** όπου πληροφορία απτο αρχείο (δευτερεύουσα μνήμη) γίνεται extacted στο page (κύρια μνήμη) και ύστερα μετατρέπεται σε integer array (buffer) και αντίστροφα (περισσότερες πληροφορίες στη προηγούμενη εργαστηριακή αναφορά).

Αυτή τη φορά ζητήθηκε να μετρηθεί ο μέσος αριθμών προσβάσεων στο δίσκο για τις τρεις παρακάτω αναζητήσεις:

- **Ερώτημα 1ο – Σειριακή αναζήτηση**

  Η μέθοδος **serialSearchFile** της κλάσης **Controller** ήταν υπεύθυνη για την αναζήτηση στο αρχείο RandomGenFile.txt για τυχαία κλειδιά. Πιο συγκεκριμένα, για μια επανάληψη 100 φορών (αριθμός αναζητήσεων): παραγόταν ένας αριθμός από τη μέθοδο **generateRandomKey** της **KeyGenerator**. Αμέσως μετά, μια άλλη επανάληψη (while) έλεγχε για κάθε block του αρχείου για τη ταυτοποίηση του κλειδιού με τη βοήθεια των μεθόδων readNextBlock και isInBuffer. Σε κάθε περίπτωση μια μεταβλητή μετρούσε τα disk accesses (ισχύει 1 disk access ανά 1 block) και ύστερα υπολογιζόταν η μέση τιμή των disk accesses για όλες τις αναζητήσεις.

- **Ερώτημα 2ο – Δυαδική αναζήτηση**

  Αυτή τη φορά έχουμε τη μέθοδο **binarySearchFile** η οποία αναζητεί το κλειδί στο αρχείο με έναν πολύ πιο αποτελεσματικό τρόπο: διαβάζεται αρχικά η μεσαία σελίδα του αρχείου (μεταφέρεται στο **page**), μετατρέπεται ύστερα σε **buffer** και ελεγχεται εάν το κλειδί βρίσκεται εκεί. Εάν βρεθεί, τερματίζεται ο αλγόριθμος, διαφορετικά ανάλογα αν το κλειδί είναι μεγαλύτερο/μικρότερο από τον αριθμό της σελίδας, αναζητείται ξανά για το κατάλληλο μισό του αρχείου.

- **Ερώτημα 3ο – Δυαδική αναζήτηση με έυρος τιμών**

  Η αναζήτηση έυρους τιμών στο αρχείο χρησιμοποιεί αντίστοιχα τις μεθόδους της **FileManager.** Σε αυτή τη μέθοδο, αναζητούνται αριθμοί που είναι μεταξύ του κλειδιού και του K. Η αναζήτηση του πρώτου γίνεται δυαδικά ενώ η αναζήτηση των υπολοίπων γίνεται σειριακά. Η μέθοδος που αφορά αυτό το ερώτημα έχει ένα πρόβλημα και δεν εμφανίζει την ορθή τιμή του μέσου αριθμού προσβάσεων δίσκου.

Τα αποτελέσματα του δεύτερου μέρους διαφαίνονται παραδίπλα.

```
PART 2
=========================================================

CALCULATION 1 - Serial Search of 100 keys from File.
---------------------------------------------------------
This might take a while, please wait. . .
Average number of disk accesses for 100 searches: 55475


CALCULATION 2 - Binary Search of 100 keys from File.
---------------------------------------------------------
Average number of disk accesses for 100 searches: 17


CALCULATION 3 - Binary Search of 100 keys from File.
K = 100
---------------------------------------------------------
Average number of disk accesses for 100 searches: 0


K = 1000
---------------------------------------------------------
Average number of disk accesses for 100 searches: 0
```

Τεκμηρίωση αποτελεσμάτων

Συμπεραίνουμε με βάση όλα τα παραπάνω ότι το δέντρο δυαδικής αναζήτησης είναι πολύ πιο αποδοτικό σε σχέσια με τη σειριακή λίστα/array.

Στην περίπτωση του 3ου ερωτήματος, περιμένουμε θεωρητικά η αναζήτηση εύρους τιμών με K = 1000 να έχει πολύ λιγότερες προσβάσεις δίσκου σε σχέση με την περίπτωση του K=100.