



Computer Engineering & Informatics
Department (CEID)

Δομές Δεδομένων

Project 2017-2018

Παναγιώτης Καββαδίας

AM:1054350

pkavvadias@ceid.upatras.gr

Το Project έχει υλοποιηθεί με την χρήση του Visual Studio 2017 σε γλώσσα C++.

Περιλαμβάνει 3 header files και 5 αρχεία cpp.

Τα αρχεία είναι ως εξής:

FileManagement.h

Στο συγκεκριμένο header file περιλαμβάνεται ο ορισμός της κλάσης File. Η κλάση αυτή χρησιμοποιείται για την διαχείριση των αρχείων integers.txt και words.txt.

Συγκεκριμένα οι public συναρτήσεις της loadIntegers και loadWords είναι υπεύθυνες για την φόρτωση των περιεχομένων των αρχείων ακεραίων και λέξεων αντίστοιχα.

Ως private έχουν οριστεί οι vectors στους οποίους φορτώνονται οι ακέραιοι και οι λέξεις(vector integers και words αντίστοιχα).

Εφόσον οι vectors είναι private χρειάζονται και getter συναρτήσεις ώστε το πρόγραμμα να έχει πρόσβαση στα περιεχόμενα τους οι οποίες έχουν οριστεί ως public(προφανώς).

Ορίζεται επίσης η κλάση MergeSort μέσω της οποίας υλοποιείται ο αλγόριθμος ταξινόμησης Mergesort.

Υπάρχει μια public συνάρτηση, η sort, και μια private, η merge.

Θα δούμε ότι η sort εσωτερικά καλεί την merge η οποία όμως δεν είναι απαραίτητο να είναι ορατή στο υπόλοιπο πρόγραμμα και για αυτό έχει οριστεί ως private.

Έπειτα υπάρχει η κλάση RBTree η οποία υλοποιεί ένα red black tree και η κλάση Node η οποία υλοποιεί έναν κόμβο του red black tree.

Αξίζει να σημειωθεί ότι μέσα στην κλάση RBTree έχει οριστεί ως friend η συνάρτηση RBSearch της κλάσης Search(θα αναφερθούμε αργότερα σε αυτήν). Αυτό είναι απαραίτητο καθώς η συνάρτηση RBSearch χρειάζεται να έχει πρόσβαση σε private μέλη της RBTree.

Ως private ορίζονται δυο κόμβοι, ο κόμβος q και κόμβος root τους οποίους ο constructor αρχικοποιεί σε null.

Ως private είναι επίσης ορισμένες και οι συναρτήσεις fix, rightrotate, leftrotate τις οποίες χρησιμοποιεί η public συνάρτηση insert.

Είναι απαραίτητες ώστε κατά την εισαγωγή ενός νέου στοιχείου στο δέντρο αυτό να εισαχθεί στην σωστή θέση.

Ακόμα ορίζεται η κλάση TrieNode η οποία υλοποιεί ένα digital Tree(Trie). Ως private ορίζεται ένας pointer σε πίνακα τύπου TrieNode των 128 χαρακτήρων. Επιλέχθηκε αυτός ο αριθμός χαρακτήρων καθώς ο κώδικας ASCII περιλαμβάνει τόσους κι έτσι θα μπορεί χωρίς πρόβλημα να παρασταθεί κάθε λέξη όποιους χαρακτήρες κι αν

περιλαμβάνει. Υπάρχει η συνάρτηση haveChildren η οποία ελέγχει αν ένας κόμβος έχει παιδιά. Χρησιμοποιείται κατά την διαγραφή ώστε να γίνεται έλεγχος αν ένας κόμβος μπορεί να διαγραφεί. Υπάρχει και η μεταβλητή τύπου boolean isEnd η οποία ορίζει αν ένας κόμβος είναι φύλλο.

Ως public υπάρχουν οι συναρτήσεις εισαγωγής στοιχείων στο Trie(insert) και διαγραφής(deletion) ενώ ορίζεται ως friend η συνάρτηση TrieSearch της κλάσης Search.

Ο constructor αρχικοποιεί την μεταβλητή isEnd σε τιμή false και κάθε θέση του πίνακα characters σε null.

Search.h

Στο συγκεκριμένο header file ορίζεται η κλάση search. Σε αυτή την κλάση υπάρχουν όλες οι ζητούμενες συναρτήσεις αναζήτησης. Συγκεκριμένα η συνάρτηση γραμμικής αναζήτησης(linearSearch),δυαδικής αναζήτησης(binarySearch),αναζήτησης παρεμβολής(interpolationSearch),αναζήτησης στο red black(RbSearch) και αναζήτησης στο Trie(TrieSearch).

Στο συγκεκριμένο header file γίνεται forward declaration των κλάσεων TrieNode και RBTree καθώς χρειάζονται ως ορίσματα(arguments) σε κάποιες από τις συναρτήσεις.

TimeCalc.h

Στο συγκεκριμένο header file ορίζεται η κλάση Calculate. Σε αυτήν περιλαμβάνονται συναρτήσεις για την μέτρηση του χρόνου που ζητείται στο Μέρος Δ.

Η linearCalc μετράει τον χρόνο που απαιτείται για γραμμική αναζήτηση, η binaryCalc τον χρόνο για δυαδική, η interpolCalc τον χρόνο για αναζήτηση παρεμβολής και η rbTreeCalc τον χρόνο για αναζήτηση στο red black tree.

Οι συναρτήσεις αυτές υπολογίζουν τον μέσο χρόνο και τον χρόνο χειρότερη περίπτωσης.

RedBlack.cpp

Στο συγκεκριμένο αρχείο υπάρχει η υλοποίηση των συναρτήσεων insert,fix,rightrotate και leftrotate του red black.

Η συνάρτηση insert δέχεται ως όρισμα τον ακέραιο που θέλουμε να εισάγουμε στο δέντρο ενώ οι υπόλοιπες τον κόμβο που θέλουμε να επεξεργαστούμε.

Η συνάρτηση fix καλείται από την συνάρτηση insert. Η insert μετά από έλεγχο των συνθηκών μπορεί να καλέσει την rightrotate ή την leftrotate ανάλογα αν απαιτείται δεξιά ή αριστερή περιστροφή στο δέντρο.

Search.cpp

Σε αυτό το αρχείο περιέχονται οι υλοποιήσεις όλων των αναζητήσεων. Οι συναρτήσεις `linearSearch`, `binarySearch` και `interpolationSearch` δέχονται ως ορίσματα αναφορά(reference) στον vector που βρίσκονται οι αριθμοί και τον προς αναζήτηση αριθμό. Αν ο αριθμός βρεθεί σε κάποια θέση του vector η μέθοδος επιστρέφει(return) την θέση που βρέθηκε. Διαφορετικά επιστρέφει -1.

Στην `interpolation search` αν δοθεί προς αναζήτηση ένας πολύ μεγάλος αριθμός υπάρχει ο κίνδυνος να βρεθούμε εκτός της εμβέλειας(out of range) του vector με αποτέλεσμα να προκληθεί crash του προγράμματος. Το ίδιο έχει εφαρμοστεί και στην `binarySearch` καθώς σε διαφορετική περίπτωση αν ο προς αναζήτηση αριθμός ήταν μεγαλύτερος από τον μέγιστο η αναζήτηση έδειχνε λανθασμένα ότι ο αριθμός βρέθηκε στην τελευταία θέση του vector.

Έτσι πριν την εκτέλεση της αναζήτησης γίνεται έλεγχος και αν ο προς αναζήτηση αριθμός είναι μεγαλύτερος από τον αριθμό στην μεγαλύτερη θέση του vector επιστρέφεται -1. Το συγκεκριμένο μπορεί να εφαρμοστεί χωρίς βλάβη της γενικότητας καθώς οι αριθμοί στον vector είναι ταξινομημένοι, δηλαδή ο αριθμός στην μεγαλύτερη θέση είναι πάντοτε ο μεγαλύτερος διαθέσιμος αριθμός.

Η συνάρτηση `RbSearch` δέχεται ως ορίσματα τον προς αναζήτηση αριθμό και αναφορά στο red black tree. Η διαφορά στην υλοποίηση της συγκεκριμένης συνάρτησης είναι ότι ενώ στις υπόλοιπες επιστρέφεται μια τιμή και ο χειρισμός(εκτύπωση αποτελέσματος) γίνεται στην `main` η συγκεκριμένη δεν επιστρέφει κάτι(void) και η εμφάνιση των αποτελεσμάτων γίνεται από την ίδια την συνάρτηση. Αυτό επελέγη καθώς στην συγκεκριμένη αναζήτηση θέλουμε να εμφανίζονται περισσότερες πληροφορίες(χρώμα κόμβου, γονιός, παιδιά) με αποτέλεσμα ο χειρισμός να είναι πιο εύκολος μέσα στην ίδια την συνάρτηση.

Η συνάρτηση `TrieSearch` δέχεται ως ορίσματα την λέξη(string) που θέλουμε να αναζητήσουμε και αναφορά στον pointer τύπου `TrieNode`(στο δέντρο δηλαδή).

Αν το η λέξη βρεθεί επιστρέφει true διαφορετικά επιστρέφει false και ο χειρισμός του αποτελέσματος γίνεται στην `main`.

Το ορίσματα δίνονται με αναφορά(pass by reference) αντί να δοθούν με την τιμή τους(pass by value) καθώς έτσι αντιγράφεται μόνο η διεύθυνση και όχι το αντικείμενο-στοιχείο. Αυτό είναι ιδιαίτερα χρήσιμο για εξοικονόμηση χώρου στην μνήμη καθώς αν αντιγράφαμε τον vector με τους αριθμούς θα είχαμε μεγάλη σπατάλη χώρου.

TimeCalc.cpp

Σε αυτό το αρχείο υπάρχουν οι υλοποιήσεις των συναρτήσεων που μετρούν τον χρόνο σε κάθε αναζήτηση. Εκτυπώνεται ο χρόνος χειρότερης περίπτωσης και ο μέσος χρόνος σε κάθε αναζήτηση σε microseconds. Για την μέτρηση χρόνου χρησιμοποιείται η βιβλιοθήκη `chrono` ενώ για την δημιουργία τυχαίων προς

αναζήτηση αριθμών χρησιμοποιείται η βιβλιοθήκη random. Οι τυχαίοι προς αναζήτηση αριθμοί βρίσκονται σε ένα range από 0 έως 600.000 το οποίο καλύπτει όλες τις πιθανές περιπτώσεις(ο αριθμός να βρεθεί,ο αριθμός να μην υπάρχει, ο αριθμός να είναι μεγαλύτερος από όλους τους αριθμούς που υπάρχουν διαθέσιμοι).

Trie.cpp

Στο συγκεκριμένο αρχείο βρίσκονται οι υλοποιήσεις των συναρτήσεων του Trie.

Management.cpp

Στο συγκεκριμένο αρχείο βρίσκονται υλοποιημένες όλες οι συναρτήσεις της κλάσης File καθώς και οι 2 συναρτήσεις που απαιτούνται για τον αλγόριθμο merge sort.

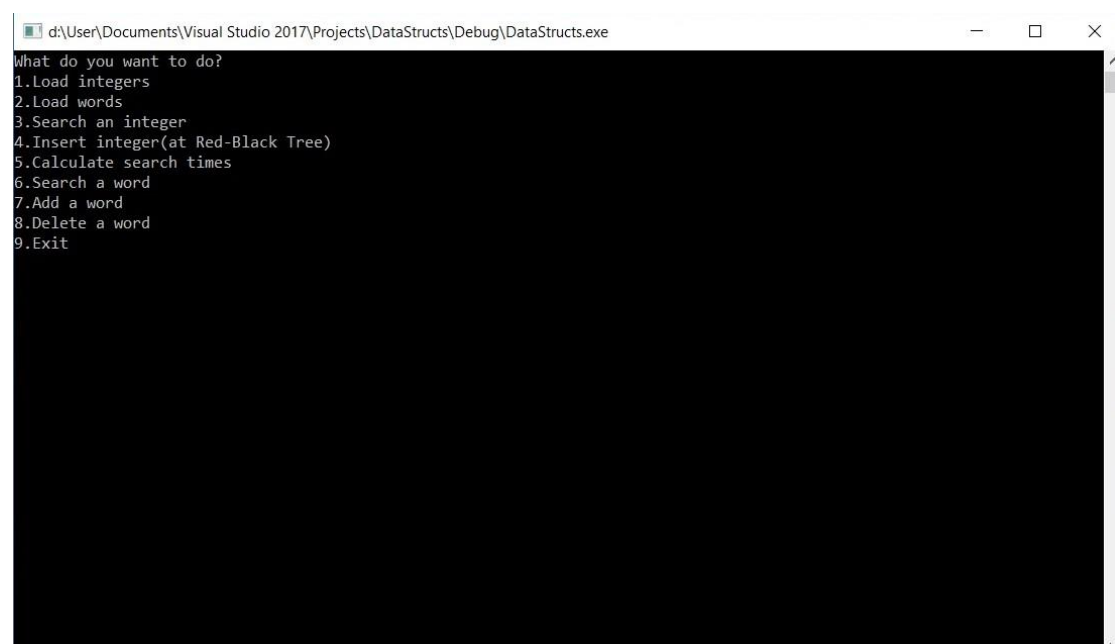
Η συνάρτηση sort είναι αναδρομική(recursive) καθώς στην υλοποίηση της περιλαμβάνονται κλήσεις προς τον εαυτό της.

Σε αυτό το αρχείο υπάρχει υλοποιημένη και η main η οποία είναι υπεύθυνη για το μενού σε περιβάλλον γραμμής εντολών του προγράμματος

Το project βρίσκεται αναρτημένο στο github στην διεύθυνση <https://github.com/pkavvadias/DataStructs>

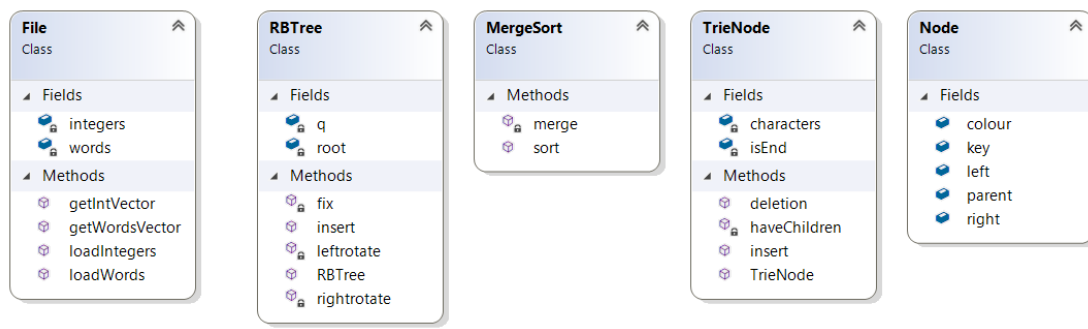
Το repository είναι private και θα γίνει public αμέσως μετά το περας της προθεσμίας υποβολής

Screenshot του αρχικού μενού του προγράμματος

A screenshot of a terminal window showing the main menu of a program. The window title is "d:\User\Documents\Visual Studio 2017\Projects\DataStructs\Debug\DataStructs.exe". The menu text is as follows:

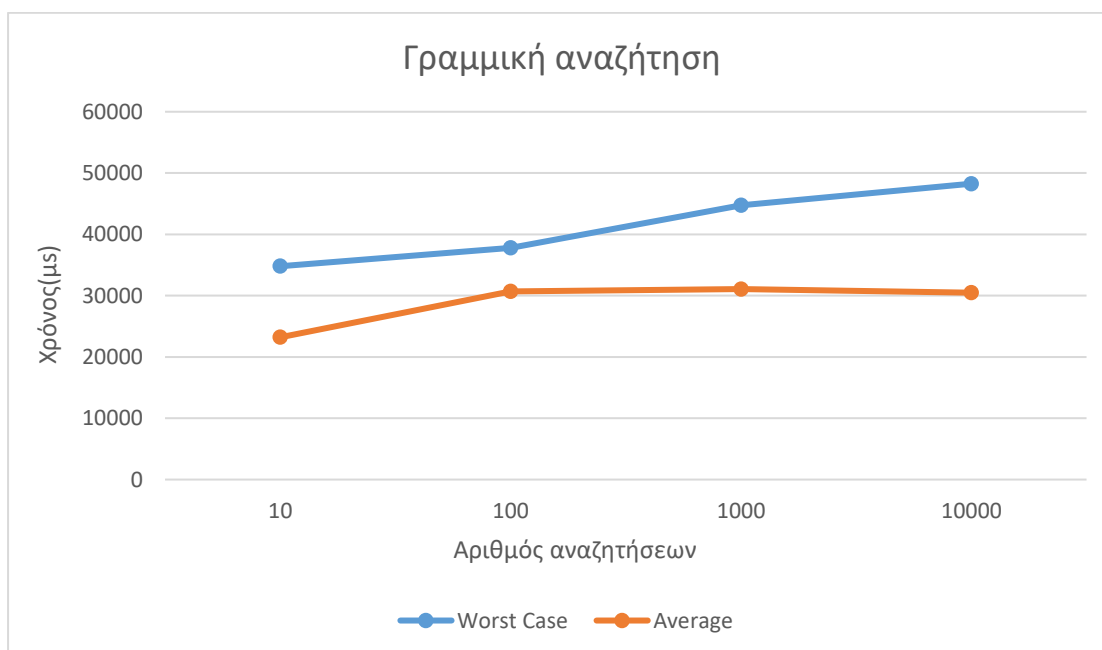
```
What do you want to do?
1.Load integers
2.Load words
3.Search an integer
4.Insert integer(at Red-Black Tree)
5.Calculate search times
6.Search a word
7.Add a word
8.Delete a word
9.Exit
```

UML διάγραμμα του προγράμματος

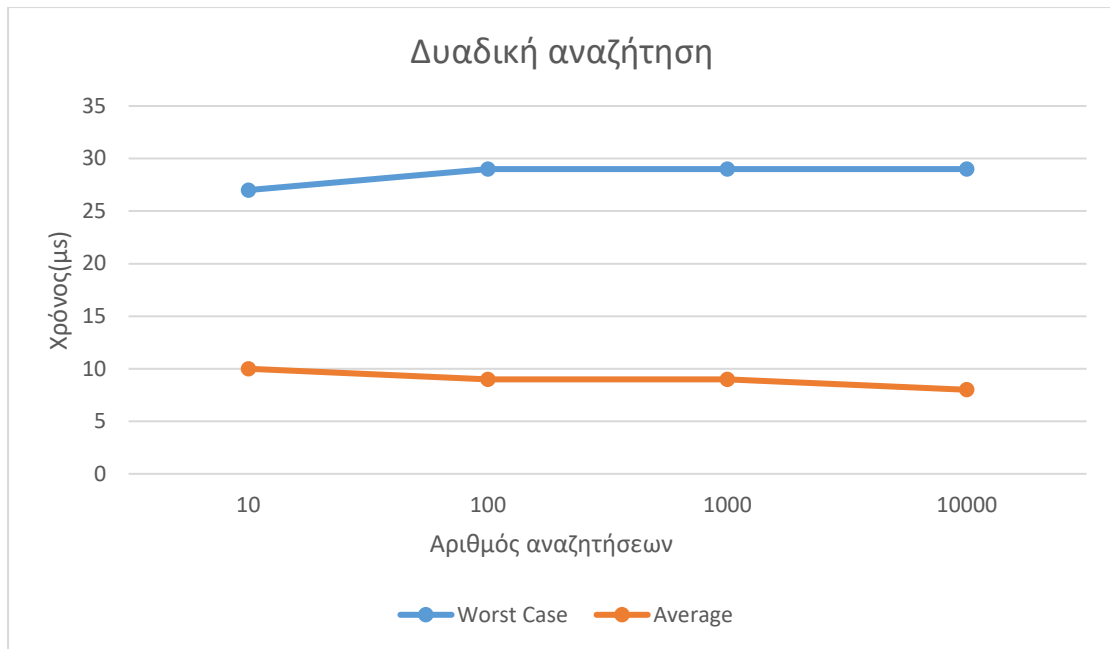


Γραφικές παραστάσεις χρόνων εκτέλεσης αναζητήσεων

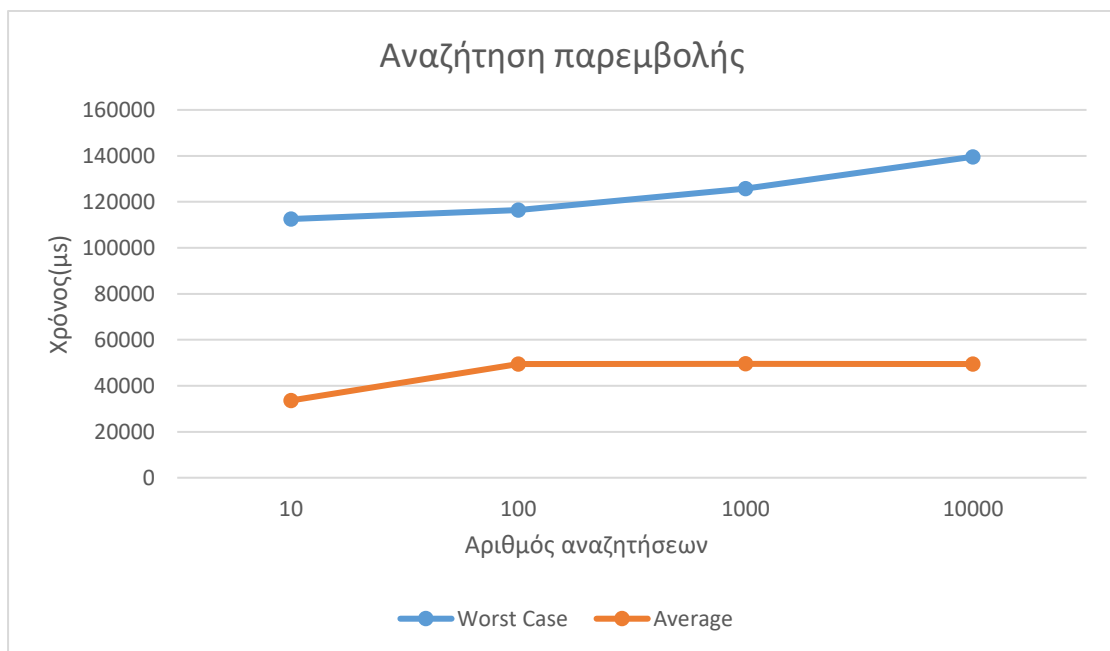
Για κάθε είδος αναζήτησης θα παρατίθεται γραφική παράσταση με τον χρόνο χειρότερης περίπτωσης και τον μέσο χρόνο αναζήτησης κατά την εκτέλεση 10,100, 1000 και 10000 αναζητήσεων. Οι χρόνοι δίνονται σε microseconds(μs)



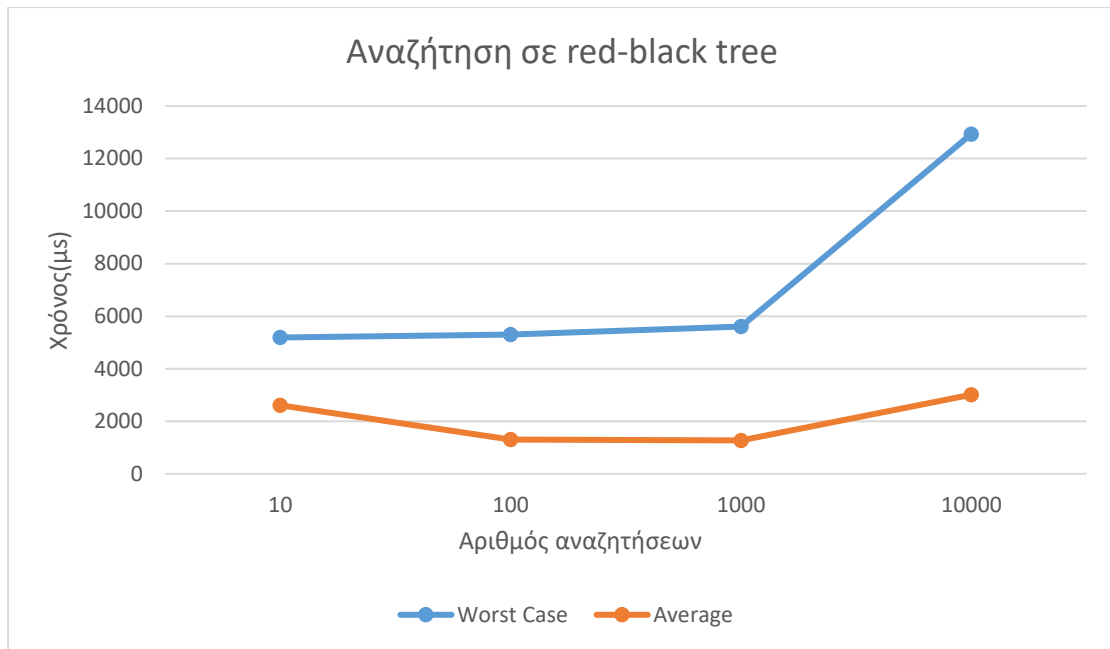
Είναι γνωστό από την θεωρία ότι η γραμμική αναζήτηση έχει χρόνο χειρότερης περίπτωσης $O(n)$ όπου n ο αριθμός των στοιχείων στον πίνακα.



Η διαδική αναζήτηση έχει χρόνο χειρότερης περίπτωσης $O(\log n)$. Παρατηρούμε ότι είναι η πιο γρήγορη από όσες έχουμε υλοποιήσει



Η αναζήτηση παρεμβολής έχει μέσο χρόνο $O(\log(\log n))$. Όμως έχει χρόνο χειρότερης περίπτωσης $O(n)$ το οποίο είναι αρκετά αργό καθώς την ίδια πολυπλοκότητα έχει και η γραμμική αναζήτηση. Σε πρακτικές εφαρμογές η αναζήτηση παρεμβολής χρησιμοποιείται συνδυαστικά με κάποια άλλη μέθοδο αναζήτησης.



Το red-black tree έχει χρόνο αναζήτησης χειρότερης περίπτωσης $O(\log n)$. Παρατηρούμε ότι ενώ η αναζήτηση είναι πιο γρήγορη από την δυαδική και την αναζήτηση παρεμβολής είναι σημαντικά πιο αργή από την δυαδική