ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Εργασία Εξαμήνου

|  |  |  |
| --- | --- | --- |
| Συμμετέχοντες |  |  |
| Όνομα | Αλέξανδρος | Χαράλαμπος |
| Επίθετο | Καραμουχτάρης | Δεληγιαννάκης |
| ΑΕΜ | 4369 | 4383 |
| Έτος | 1ο | 1ο |

Η συγκεκριμένη εργασία στοχεύει στην υλοποίηση διάφορων δομών δεδομένων στη γλώσσα C++ και την αξιολόγηση τους αναφορικά με:

* Την χρονική τους αποδοτικότητα στην αποθήκευση και την αναζήτηση ογκώδους πλήθους στοιχείων, καθώς και
* Την αξιοπιστία τους στη διαχείριση της μνήμης.

Οι δύο συμφοιτητές συνεργαστήκαμε με τη βοήθεια του git, το οποίο μας επέτρεψε να κρατάμε στιγμιότυπα του κώδικα μας και να συνεργαζόμαστε με ευκολία. Μάλιστα, το repository μας είναι πλέον δημόσιο και διαθέσιμο μέσω του ακόλουθου συνδέσμου: <https://github.com/AlexandrosKaram/DS-Project-2023.git> .

Οι δομές που υλοποιήθηκαν στην εργασία είναι οι εξής:

* Αταξινόμητος πίνακας
* Ταξινομημένος πίνακας
* Δυαδικό Δένδρο Αναζήτησης
* Δυαδικό Δένδρο AVL
* Πίνακας κατακερματισμού με ανοιχτή διεύθυνση

Περιεχόμενα

[ΚΩΔΙΚΑΣ 2](#_Toc139060904)

[MAIN 3](#_Toc139060905)

[FUNCS 4](#_Toc139060906)

[PAIR 4](#_Toc139060907)

[ΔΟΜΕΣ 5](#_Toc139060908)

[ΑΤΑΞΙΝΟΜΗΤΟΣ ΠΙΝΑΚΑΣ 5](#_Toc139060909)

[ΤΑΞΙΝΟΜΗΜΕΝΟΣ ΠΙΝΑΚΑΣ 6](#_Toc139060910)

[ΔΥΑΔΙΚΟ ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ 6](#_Toc139060911)

[ΔΕΝΔΡΟ AVL 6](#_Toc139060912)

[ΠΙΝΑΚΑΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ 7](#_Toc139060913)

[ΣΥΜΠΕΡΑΣΜΑΤΑ 7](#_Toc139060914)

## ΚΩΔΙΚΑΣ­­­­

Το συγκεκριμένο πρόγραμμα αποτελείται από τρία βασικά τμήματα:

1. Τις δηλώσεις των κλάσεων και των συναρτήσεων σε αρχεία “.h”
2. Τις αντίστοιχες υλοποιήσεις τους σε αρχεία “.cpp”
3. Τα αποτελέσματα σε αρχεία κειμένου “.txt”.

Συγκεκριμένα:

Στον φάκελο “header/” βρίσκονται όλες οι επικεφαλίδες των κλάσεων και των συναρτήσεων, ενώ στον “src/” οι υλοποιήσεις τους καθώς και το κυρίως πρόγραμμα (“main.cpp”). Τα αρχεία κειμένου που τίθενται υπό επεξεργασία λαμβάνονται από τον “Parent folder” ενώ τα αποτελέσματα των δομών βρίσκονται στον φάκελο “results/”. Παρακάτω, μπορείτε να δείτε αναλυτικότερα τις πληροφορίες σχετικά με κάθε τμήμα του κώδικα

## MAIN

H main.cpp αποτελεί το κυρίως πρόγραμμα και ανάλογα με τις εισόδους που θα τις δοθούν μέσω του τερματικού εκτελεί τις ανάλογες ενέργειες.

Για την μεταγλώττιση του κώδικα με gcc compiler είναι απαραίτητη η παρακάτω εντολή:

g++ src/\*.cpp –o src/main.

Με αυτόν τρόπο δημιουργείται και βρίσκεται στη σχετική διεύθυνση src/main.exe το εκτελέσιμο αρχείο. Για να εκτελεστεί το πρόγραμμα πρέπει να περάσουμε ως όρισμα το όνομα του αρχείου κειμένου και ακέραιους αριθμούς από το 1 μέχρι και το 5 που αντιπροσωπεύουν μία δομή ο καθένας.

1. Αταξινόμητος πίνακας
2. Ταξινομημένος πίνακας
3. Δυαδικό δένδρο αναζήτησης
4. Δυαδικό δένδρο AVL
5. Πίνακας κατακερματισμού ανοιχτής διεύθυνσης

Για παράδειγμα η παρακάτω εντολή θα τρέξει τις δομές του πίνακα κατακερματισμού και του ταξινομημένου πίνακα για το αρχείο “small-file.txt” με τη σειρά που ορίσαμε:

./src/main small-file.txt 5 2

Στην main.cpp δηλώνονται αρχικά οι απαραίτητες βιβλιοθήκες κι η επικεφαλίδα του αρχείου “funcs.h”- στο οποίο υλοποιούνται διάφορες συναρτήσεις που χρησιμοποιούμε. Αναλυτικότερα, γίνεται η τροποποίηση του αρχείου κειμένου που έχουμε επιλέξει μέσω ορισμάτων και δημιουργείται ένα σύνολο Qset το οποίο περιλαμβάνει τυχαίο πλήθος από 5000 έως 10000 ζεύγη τα οποία θα αναζητηθούν αργότερα στις δομές μας. Στη συνέχεια, καλεί τις συναρτήσεις που βρίσκονται στο αρχείο “funcs.h” οι οποίες υλοποιούν τις ανάλογες δομές που λήφθηκαν ως ορίσματα.

Η δομή της main είναι απλή ώστε να είναι εύκολα κατανοητή κι ευανάγνωστη.

## FUNCS

Οι διάφορες συναρτήσεις που καλούνται σε όλη την έκταση του προγράμματος βρίσκονται δηλωμένες στο funcs.h κι υλοποιημένες στο funcs.cpp.

Συγκεκριμένα:

* Η συνάρτηση “formatFile” αφαιρεί τη στίξη από το αρχείο κειμένου και μετατρέπει τα κεφαλαία γράμματα σε μικρά. Το τροποποιημένο αρχείο έχει όνομα “formatted.txt” κι είναι το κείμενο πάνω στο οποίο θα γίνουν όλες μας οι ενέργειες.
* Η συνάρτηση “createSet” δημιουργεί έναν πίνακα 5000-10000 θέσεων που περιλαμβάνει τυχαία ζευγάρια λέξεων τα οποία θα αναζητηθούν στις δομές αργότερα. Επιπλέον επιστρέφει με αναφορά το μέγεθος του πίνακα.
* Παρακάτω, έχουμε τις void συναρτήσεις array, sortedArray, binarySearchTree, avlTree, hashTable οι οποίες δημιουργούν ένα αντικείμενο της αντίστοιχης κλάσης-δομής, καλούν τις απαραίτητες μεθόδους και τις αποδεσμεύουν από τη μνήμη αφού ολοκληρωθεί η εκτέλεσή τους.

## PAIR

Πρόκειται για ένα struct το οποίο αντιπροσωπεύει ένα ζεύγος λέξεων και περιλαμβάνει δύο μεταβλητές τύπου std::string, τις word1-word2, καθώς και το πλήθος των εμφανίσεων του (int apps).

H δήλωση του, καθώς κι οι υπερφορτώσεις ορισμένων τελεστών γίνονται στο αρχείο Pair.h ενώ στο Pair.cpp αυτές οι υπερφορτώσεις υλοποιούνται.

## ΔΟΜΕΣ

Η κάθε μία από τις δομές που υλοποιούνται στο πρόγραμμά μας έχει το ξεχωριστό της αρχείο δήλωσης που βρίσκεται στον φάκελο header/ κι υλοποίησης που βρίσκεται στον φάκελο src/.

|  |  |  |
| --- | --- | --- |
| ΔΟΜΗ | ΚΕΦΑΛΙΔΑ | ΥΛΟΠΟΙΗΣΗ |
| Αταξινόμητος Πίνακας | Array.h | Array.cpp |
| Ταξινομημένος Πίνακας | SortedArray.h | SortedArray.cpp |
| Δυαδικό Δένδρο Αναζήτησης | BST.h | BST.cpp |
| Δυαδικό Δένδρο AVL | AVL.h | AVL.cpp |
| Πίνακας κατακερματισμού | Hashtable.h | Hashtable.cpp |

Όλες οι κλάσεις-δομές ακολουθούν μία κοινή λογική: εισάγουν δυναμικά όλα τα μοναδικά ζεύγη του αρχείου κειμένου στη δομή και αναζητούν τα ζευγάρια από το Qset. Επιπλέον, υπολογίζουν και τυπώνουν τον χρόνο δημιουργίας της δομής, τον χρόνο αναζήτησης των ζευγαριών και το πλήθος των εμφανίσεων κάθε ζεύγους. Κάθε κλάση περιέχει την κοινή μέθοδο showResults η οποία καλεί τις μεθόδους δημιουργίας και αναζήτησης ζευγαριών – createPairs, searchPairs. Εάν εκτελεστούν όλες οι δομές, περιμένουμε να έχουμε διαφορετικό χρόνο εκτέλεσης κι αναζήτησης για την κάθε μια, αλλά ίδιο πλήθος εμφανίσεων των ζευγαριών του τυχαίου συνόλου. Στις επόμενες ενότητες παρουσιάζεται αναλυτικά η λειτουργία της κάθε δομής.

## ΑΤΑΞΙΝΟΜΗΤΟΣ ΠΙΝΑΚΑΣ

Η λειτουργία του αταξινόμητου πίνακα έχει ως εξής:

Αρχικά δημιουργείται μέσω του constructor ένας πίνακας περιορισμένου μεγέθους που αποθηκεύεται στον σωρό.

Στη συνέχεια τα ζεύγη ελέγχονται ένα προς ένα για το αν είναι κατάλληλα για την εισαγωγή τους στον πίνακα. Δηλαδή, με σειριακή προσπέλαση των στοιχείων του πίνακα ελέγχεται αν το ζεύγος υπάρχει ήδη στον πίνακα. Αν ισχύει, τότε αυξάνεται το πλήθος των εμφανίσεων του, διαφορετικά προστίθεται στον πίνακα.

Πριν την εισαγωγή κάθε στοιχείου ελέγχουμε εάν υπάρχει διαθέσιμος χώρος στον πίνακα κι άμα δεν υπάρχει τότε δυναμικά διπλασιάζουμε το μέγεθός του.

Η αναζήτηση των ζευγαριών γίνεται σειριακά και τα αποτελέσματα τυπώνονται στο Array.txt

## ΤΑΞΙΝΟΜΗΜΕΝΟΣ ΠΙΝΑΚΑΣ

Ο ταξινομημένος πίνακας αποτελεί παράγωγη κλάση του αταξινόμητου και μοιράζεται μαζί του κοινές ιδιότητες και μεθόδους. Παρ ‘όλα αυτά παρουσιάζονται κάποιες διαφορές. Αναλυτικότερα:

Κατά την εισαγωγή κάθε στοιχείου ελέγχουμε με δυαδική αναζήτηση εάν το ζεύγος είναι μοναδικό. Εάν είναι, τότε αυξάνουμε το πλήθος των εμφανίσεων του. Αλλιώς, βρίσκουμε τη θέση στην οποία πρέπει να τοποθετηθεί, μετακινούμε όλα τα στοιχεία μία θέση δεξιά και τοποθετούμε το ζευγάρι στον πίνακα. Η συγκεκριμένη ενέργεια καθυστερεί πολύ τη δημιουργία του πίνακα.

Η αναζήτηση των στοιχείων του συνόλου Qset γίνεται δυαδικά.

## ΔΥΑΔΙΚΟ ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ

Το δυαδικό δένδρο είναι από τις δομές που έχουμε δει ως τώρα η πιο αποδοτική, καθώς υλοποιείται με δείκτες – καθαρά δυναμικά.

Προκειμένου να υλοποιήσουμε το ΔΔΑ, δηλώνουμε ένα βοηθητικό struct Node που αντιπροσωπεύει τους κόμβους του δένδρου.

Ως ιδιότητες έχουμε το μέγεθος του δένδρου και τον κόμβο - ρίζα του που αναδρομικά δείχνει στο υπόλοιπο υποδένδρο.

Η εισαγωγή των ζευγαριών γίνεται με ταχύτατο τρόπο, λόγω της ικανότητας του ΔΔΑ να έχει πολύ φθηνή πράξη αναζήτησης - O(logn)στη μέση περίπτωση για το κάθε στοιχείο και εισαγωγής, επίσης O(logn).

## ΔΕΝΔΡΟ AVL

**Η συγκεκριμένη δομή δεν υλοποιήθηκε με επιτυχία, καθώς δεν προλάβαμε να διορθώσουμε τα λάθη που κόστισαν τη λειτουργικότητά της. Ωστόσο, στο repository στο github θα υπάρχει το πλήρες πρόγραμμα που θα περιέχει κανονικά το AVL δένδρο και μια τροποποιημένη αναφορά. Θα το εκτιμούσαμε πολύ να λάβετε υπ’ όψην σας την ποιότητα της εργασίας μας και να σκεφτείτε να μας βαθμολογήσετε με βάση την έκδοση που θα είναι διαθέσιμη στον σύνδεσμο:** [**https://github.com/AlexandrosKaram/DS-Project-2023**](https://github.com/AlexandrosKaram/DS-Project-2023)

Το δένδρο AVL αποτελεί υποκλάση του απλού ΔΔΑ και μοιράζεται πολλά κοινά στοιχεία. Η κύρια διαφορά τους, είναι ότι σε κάθε εισαγωγή στοιχείου ελέγχεται εάν το δένδρο είναι ισορροπημένο – δηλαδή εάν κάθε αριστερό και δεξί υποδένδρο έχουν διαφορά ύψους <=1.

Εάν το δένδρο δεν είναι ισορροπημένο, τότε εκτελούνται περιστροφές για να το επαναφέρουμε σε ισορροπία. Αυτό έχει ως συνέπεια μια καθυστέρηση στη δημιουργία του δένδρου, άλλα του δίνει το πλεονέκτημα της γρηγορότερης αναζήτησης. (Κάνει τη χειρότερη περίπτωση ίση με τη μέση.)

## ΠΙΝΑΚΑΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ

Ο πίνακας κατακερματισμού ανοιχτής διεύθυνσης είναι ίσως η πιο ενδιαφέρουσα κι εντυπωσιακή δομή ως τώρα. Η λειτουργία του προϋποθέτει τη δημιουργία μίας βοηθητικής κλάσης – της hashEntry – η οποία έχει ως ιδιότητες: ένα ζευγάρι, τον κωδικό “hash” του και τη λογική τιμή occupied, που αναδεικνύει εάν το κελί είναι κατειλημμένο.

Ο πίνακας αποθηκεύει αντικείμενα της κλάσης hashEntry.

Το κύριο χαρακτηριστικό του πίνακα είναι η συνάρτηση κατακερματισμού, η οποία υπολογίζει τη θέση του πίνακα που αντιστοιχεί σε κάθε ζεύγος ως τον λόγο του hashCode προς την χωρητικότητά του.

Αυτό γίνεται για να έχουμε σχεδόν αυτόματη πρόσβαση σε κάθε στοιχείο του πίνακα με σταθερή πολυπλοκότητα αναζήτησης Ο(1) στην καλύτερη περίπτωση και Ο(n) στη χειρότερη (όταν ο πίνακας είναι σχεδόν γεμάτος).

Για να είμαστε κοντά στην καλύτερη περίπτωση αναζήτησης, διπλασιάζουμε τον πίνακα κάθε φορά που φτάνει στο 50% της χωρητικότητας του. Για να γίνει ο διπλασιασμός πρέπει να κάνουμε “rehash” στα στοιχεία του ώστε να τοποθετηθούν σε καινούργια θέση.

Αυτό, όμως, έχει ως μειονέκτημα την κακή διαχείριση μνήμης καθώς πάντα καταλαμβάνει τουλάχιστον τον διπλάσιο χώρο από το πλήθος των στοιχείων του.

## ΣΥΜΠΕΡΑΣΜΑΤΑ

*Καθώς δεν ολοκληρώθηκε με επιτυχία η υλοποίηση του AVL δένδρου τα συμπεράσματα μας θα αφορούν τις υπόλοιπες τέσσερις δομές.*

Η πιο αποτελεσματική δομή όσον αφορά τον χρόνο δημιουργίας καθώς και αναζήτησης ζευγαριών είναι ο Πίνακας Κατακερματισμού, λόγω του ταχύτατου εντοπισμού στοιχείων. Παράλληλα, στη διαχείριση της μνήμης είναι η πιο «σπάταλη» δομή καθώς δεσμεύει περισσότερο χώρο απ ‘ότι χρειάζεται.

Αρκετά γρήγορο στη δημιουργία και την αναζήτηση, είναι και το Δυαδικό Δένδρο Αναζήτησης, το οποίο όμως είναι εξαιρετικά αποτελεσματικό στη διαχείριση της μνήμης καθώς δηλώνεται δυναμικά και πάντα έχει μέγεθος ίσο με αυτό που χρειάζεται.

Από την άλλη, ο ταξινομημένος κι ο αταξινόμητος πίνακας, δηλαδή οι δομές που «παραδοσιακά» είναι στατικές δυσκολεύονται να αποθηκεύσουν τα ζευγάρια γρήγορα με δυναμικό τρόπο. Ωστόσο, παρά το γεγονός ότι ο ταξινομημένος πίνακας είναι ο πιο αργός στη δημιουργία της δομής, είναι επίσης αρκετά γρήγορος στην αναζήτηση στοιχείων.

Δείγμα: 8500 ζευγάρια για αναζήτηση.