**Project Δομών Δεδομένων Documentation**

|  |  |  |
| --- | --- | --- |
| Καραγιάννης Λεωνίδας Ιωάννης | 1064036 | [st1064036@ceid.upatras.gr](mailto:st1064036@ceid.upatras.gr) |
| Ακίνο Ελλούλ Παύλος | 1059630 | st1059630@ceid.upatras.gr |
| Αλ-Μπαμπούλ Γιώργος | 1059631 | St1059631@ceid.upatras.gr |

Το Project υλοποιήθηκε σε C++. Κάθε Part της εκφώνησης είναι σε διαφορετικό αρχείο.

**Part 1**

* Κομμάτια του Part 1 που λειτουργούν:

**Sorting**  
1. Bubble Sort

2. Insertion Sort

3. Selection Sort

4. Quick Sort

5. Merge Sort

6. Heap Sort

**Searching**

1. Linear Search
2. Binary Search
3. Interpolation Search
4. Binary/Interpolation Search

* Κομμάτια του Part 1 που δεν λειτουργούν:  
    
  1. Ο Binary/Interpolation Search αλγόριθμος δεν υλοποιήθηκε να κάνει εκθετικά μεγάλα βήματα μετά από ~3 βήματα. Υλοποιήθηκε με απλά βήματα μεγέθους sqrt(n) . (δηλαδή τρέχει κανονικά)

Λειτουργία κώδικα

1. Ανοίγει το αρχείο του οποίου το όνομα δηλώνεται στην main
2. Εμφανίζεται μενού επιλογής αλγορίθμου για sort και επιλογή εξόδου από το μενού αυτό. Όταν ολοκληρώνεται η εκτέλεση κάποιου sorting αλγορίθμου, εμφανίζεται και ο χρόνος εκτέλεσης σε millisecond και μετά ρωτάται ο χρήστης αν θέλει να εμφανιστεί το ταξινομημένο vector.
3. Αν επιλεχθεί έξοδος από αυτό το μενού, εμφανίζεται μενού για την επιλογή αλγορίθμου searching για την αναζήτηση κάποιου αριθμού στην ταξινομημένο πίνακα (vector στην προκειμένη), καθώς και επιλογή για έξοδο από το πρόγραμμα
4. Όταν ολοκληρώνεται η αναζήτηση αριθμού μέσω κάποιου searching αλγορίθμου, εμφανίζεται και ο χρόνος εκτέλεσης του.

Η δυναμική φόρτωση αρχείου λυνόταν με dynamic array και με vector. Επιλέξαμε vector γιατί είναι πιο εύκολο, απλό και ανά πάσα στιγμή, χρησιμοποιεί την λιγότερη δυνατή μνήμη.

Οι βοηθητικές συναρτήσεις πάνω από την main, καθώς και όλος ο κώδικας, έχουν αρκετά self-explanatory σχόλια.

**Παρατηρήσεις:**

Ο Merge Sort και ο Quick Sort, βγαίνουν πιο αργοί από τον Heap Sort (ο οποιός βγαίνει να είναι ο γρηγορότερος από όλους), λογικά λόγω αναδρομικής υλοποίησης.

**Part 2**

* Κομμάτια του Part 2 που λειτουργούν:

Όλα.

Το Part 2 υλοποιήθηκε με κλάση Student, BinarySearchTree και struct tree\_node.

**Η κλάση Student** έχει private χαρακτηρηστικά τα στοιχεία του φοιτητή (AM, first name, last name, grade) και public τις συναρτήσεις set και get των χαρακτηρηστικών αυτών.

Η κλάση BinarySearchTree έχει private χαρακτηρηστικά το struct tree\_node το οποίο έχει pointers για left και right, καθώς και το root.

Οι public συναρτήσεις είναι:

* για τον έλεγχο αν το δέντρο είναι άδειο
* για τα 3 traversals (in-pre-post order)
* τις αντίστοιχες print συναρτήσεις που καλούν τις συναρτήσεις των traversals από το root του δέντρου
* εισαγωγή node βάσει ΑΜ και βάσει βαθμού
* αφαίρεσης κάποιου node
* αναζήτησης κάποιου node
* αλλαγής στοιχείων κάποιου node

Για την επιλογή δέντρου grade-wise, υπάρχουν και οι συναρτήσεις για:

* εύρεση μικρότερου/υψηλότερου βαθμού
* εύρεση φοιτητών με τους αντίστοιχους βαθμούς
* αντίστοιχες print συναρτήσεις που καλούν τις από πάνω από το root του δέντρου.

Κάθε input του δέντρου, είναι instance της κλάσης Student.

Κάθε input των αλυσιδών hash είναι επίσης instance της κλάσης Student.

1. Ανοίγει το αρχείο του οποίου το όνομα δηλώνεται στην main
2. Εμφανίζεται μενού που ζητάει από τον χρήστη να διαλέξει μεταξύ:
   1. Φόρτωση του μενού BST το οποίο ταξινομείται βάσει ΑΜ
   2. Φόρτωση του μενού BST το οποίο ταξινομείται βάσει βαθμού
   3. Φόρτωση του μενού αλυσιδών hash

**Στο AM-wise BST, ανοίγεται το αρχείο, φορτώνεται δίνονται οι επιλογές:**

Binary Search Tree Operations

-----------------------------

0. Search

1. Insertion/Creation **(extra)**

2. In-Order Traversal

3. Pre-Order Traversal **(extra)**

4. Post-Order Traversal **(extra)**

5. Removal

6. Change a student's info

7. Return to main menu

8. Exit

**Στο μενού του grade-wise BST**, ανοίγεται το αρχείο, δημιουργείται το δέντρο βάσει βαθμού (φοιτητές με βαθμό ίσο αυτού που ελέγχεται κάθε φορά στην συνάρτηση insert, πάνε αριστερά). Το μενού που εμφανίζεται δίνει τις εξής επιλογές:

Grade-wise Binary Search Tree Operations

-----------------------------

0. Search for the students with the LOWEST grade

1. Search for the students with the HIGHEST grade

2. Return to main menu

3. Exit

**Στο μενού των αλυσιδών hashing**, ανοίγεται το αρχείο, φορτώνεται σε ένα δισδιάστατο vector, όπου κάθε γραμμή είναι ένα bucket, με τα chained inputs του.

Η Hash συνάρτηση έχει μία global constant μετάβλητη στην αρχή του κώδικα, την PRIME, την οποία μεταβάλλουμε όπως θέλουμε.

Το μενού που εμφανίζεται είναι το εξής:  
-----------------------------

Hash Table Operations

-----------------------------

0. Search

1. Edit

2. Delete

3. Insert student **(extra)**

4. Return to main menu

5. Exit

Όλος ο κώδικας έχει πολλά σχόλια που τον κάνουν ευανάγνωστο και κατανοητό.