ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ ΓΙΑ ΤΟ ΜΑΘΗΜΑ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022

ΜΕΛΗ ΟΜΑΔΑΣ : ΚΑΛΛΙΟΠΗ ΜΑΛΛΑ 3979

ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΑΚΗΣ 3518

Στην συγκεκριμένη προγραμματιστική εργασία ασχοληθήκαμε με την υλοποίηση των εξής πέντε βασικών δομών δεδομένων:

1) *μη ταξινομημένος πίνακας,*
2*) ταξινομημένος πίνακας*,
3) *απλό δυαδικό δένδρο αναζήτησης*,
4) *δυαδικό δένδρο αναζήτησης τύπου AVL* και
5) *πίνακας κατακερματισμού με ανοικτή διεύθυνση*.

Μας ζητήθηκε οι δομές αυτές να αποθηκεύουν τα ζεύγη συνεχόμενων λέξεων του κειμένου Gutenberg Project που περιέχει γνωστά έργα παγκόσμιας λογοτεχνίας καθώς επίσης και το πλήθος εμφανίσεων αυτών.

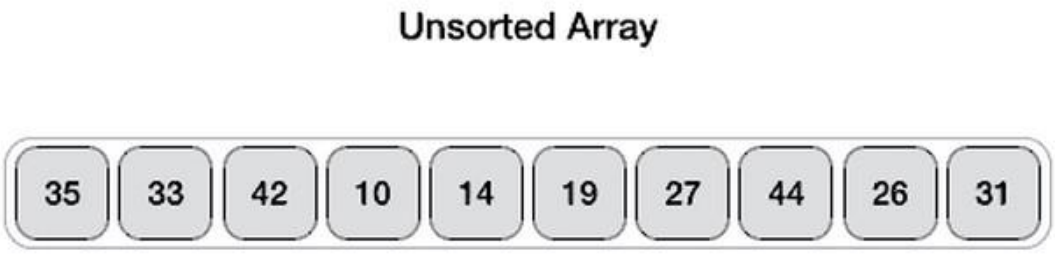
Πρώτη μας μέριμνα ήταν η προ επεξεργασία του κειμένου που θα χρησιμοποιηθεί. Αυτό σημαίνει ότι μετατρέψαμε όλους τους χαρακτήρες σε πεζούς κι τους λευκούς χαρακτήρες σε κενά (spaces). Η συνάρτηση αυτή *pre_processing* βρίσκονται στην βοηθητική κλάση που δημιουργήσαμε **Tester.h**

Έπειτα, δημιουργήσαμε το υποσύνολο Q με την συνάρτηση *createQ* που βρίσκεται και αυτή στην κλάση **Tester.h** καθώς και τις βοηθητικές συναρτήσεις benchmarkInsertClock και benchmarkSearchClock. Αυτές, φροντίζουν την εισαγωγή των ζευγών σε μία δομή και την αναζήτηση των ζευγών του υποσυνόλου Q σε μία δομή, αντίστοιχα.

Επόμενο βήμα μας ήταν να ξεκινήσουμε να υλοποιούμε τις δομές.

1. ΜΗ ΤΑΞΙΝΟΜΗΜΕΝΟΣ ΠΙΝΑΚΑΣ (UnsortedArray)

Η δομή του αταξινόμητου πίνακα έχει την εξής μορφή:



Γνωρίζουμε ότι η πολυπλοκότητα του είναι ίση με είναι O(n) για τον χώρο, O(1) για την εισαγωγή και την διαγραφή και O(n) για την αναζήτηση. Αυτό σημαίνει ότι για να βρούμε ένα στοιχείο στον αταξινόμητο πίνακα θα πρέπει να κάνουμε σειριακή αναζήτηση ώσπου να εντοπίσουμε αυτό που ψάχνουμε ή να φτάσουμε στο τέλος του πίνακα.
Οι λειτουργίες ενός αταξινόμητου πίνακα που θα πρέπει να συμπεριλάβουμε στον κώδικα είναι :

- Εισαγωγή
- Διαγραφή
- Αναζήτηση

Με αυτή την λογική πέρα από τις βασικές μεθόδους της κλάσης που χρησιμοποιούνται για την ορθή δομή κι λειτουργία του πίνακα ώστε να είναι σε θέση πλέον να επεξεργαστεί τα δεδομένα που της παρέχονται ως είσοδος(πίνακας pairs) :

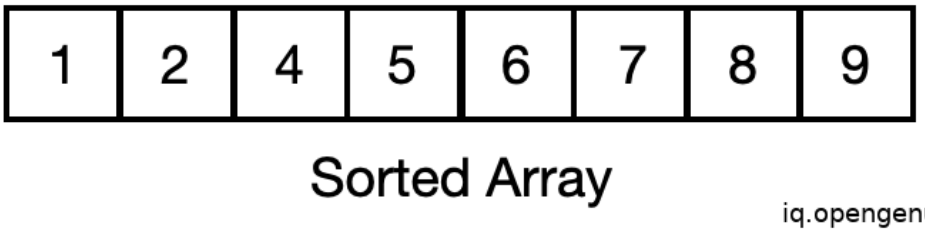
- UnsortedArray//Constructor
- UnsortedArray//Destructor
- getSize // getters
- getPairs
- getClass

Επιπλέον χρησιμοποιήσαμε τις παρακάτω συναρτήσεις :

- **Insert**->Υλοποιεί την λειτουργία της εισαγωγής στοιχείων του πίνακα . Αναλυτικότερα ψάχνει αν το στοιχείο που θέλει να προσθέσει υπάρχει ήδη μέσα στον πίνακα ή όχι με την βοήθεια σειριακής αναζήτησης . Αν δεν υπάρχει τότε το προσθέτει στο τέλος του πίνακα
- **Search**-> Υλοποιεί την λειτουργία της σειριακής αναζήτησης για να βρούμε το πλήθος εμφάνισης του κάθε ζεύγους .Αν το στοιχείο που ψάχνει υπάρχει επιστρέφει τις φορές που έχει εμφανιστεί αλλιώς -1
- **SearchPos->** Υλοποιεί την λειτουργία της σειριακής αναζήτησης για να βρούμε αν το στοιχείο υπάρχει στον πίνακα (επιστρέφει τη θέση) ή όχι (επιστρέφει -1)
- **Remove->** Υλοποιεί την λειτουργία της διαγραφής στοιχείου από τον πίνακα κι μετακίνησης των υπολοίπων για να μην υπάρχει κενή θέση
- **Reallocate->** Αλλάζει το μέγεθος του πίνακα

2. ΤΑΞΙΝΟΜΗΜΕΝΟΣ ΠΙΝΑΚΑΣ (SortedArray)

Η δομή του ταξινομημένου πίνακα έχει την εξής μορφή :



Γνωρίζουμε ότι η πολυπλοκότητα του είναι ίση με είναι O(n) για τον χώρο, O(n) για την εισαγωγή και την διαγραφή και O(log n) για την αναζήτηση. Αυτό σημαίνει ότι για να βρούμε ένα στοιχείο στον αταξινόμητο πίνακα θα πρέπει να κάνουμε δυαδική αναζήτηση ώσπου να εντοπίσουμε αυτό που ψάχνουμε. Οι λειτουργίες ενός ταξινομημένου πίνακα που θα πρέπει να συμπεριλάβουμε στον κώδικα είναι :

- Εισαγωγή
- Διαγραφή
- Αναζήτηση

Ο αταξινόμητος(**UnsortedArray.h)** κι ο ταξινομημένος πίνακας**(SortedArray.h)** δημιουργήθηκαν παράλληλα καθώς χρησιμοποιήσαμε κληρονομικότητα μεταξύ τους. Συγκεκριμένα ο SortedArray(παράγωγος κλάση) κληρονομεί τις public μεθόδους της UnsortedArray(κλάση βάση).Για αυτόν τον λόγο ο

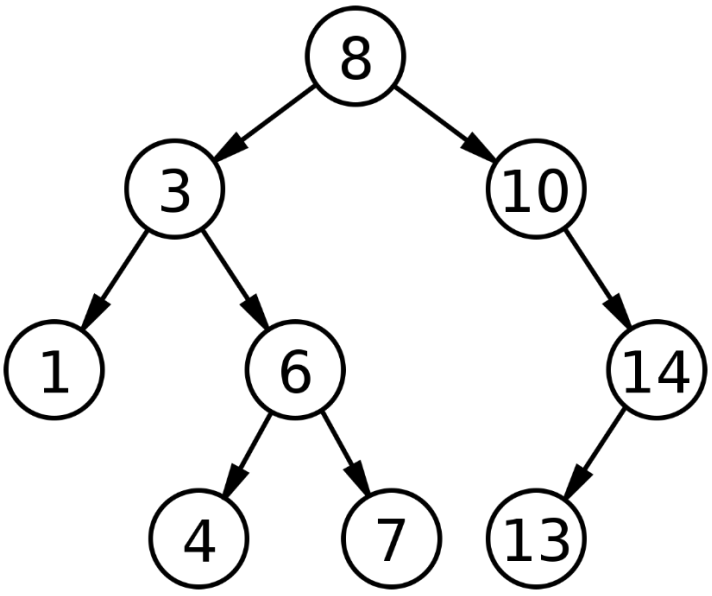
κατασκευαστής , ο καταστροφέας κι η συνάρτηση για διαγραφή στοιχείων (remove) παραλείπονται στην κλάση του ταξινομημένου πίνακα.

Οι επιπρόσθετες συναρτήσεις που χρησιμοποιούμε είναι :

- **Insert->** Υλοποιεί την λειτουργία της εισαγωγής στοιχείων στον πίνακα. Η διαφορά της με την συνάρτηση insert του αταξινόμητου πίνακα είναι ότι η αναζήτηση στοιχείου του πίνακα γίνεται με δυαδική αναζήτηση κι το στοιχείο τοποθετείτε με αλφαβητική σειρά στην κατάλληλη θέση καθώς υπάρχει μια προκαθορισμένη διάταξη η οποία πρέπει να τηρηθεί
- **SearchForInsertion->** Υλοποιεί την λειτουργία της αναζήτησης χρησιμοποιώντας τον αλγόριθμο της δυαδικής αναζήτησης. Στην ουσία αναζητά κι επιστρέφει την κατάλληλη θέση για να τοποθετηθεί το στοιχείο κι να συνεχίσει να ισχύει η προκαθορισμένη διάταξη (αλφαβητική σειρά)
- **Search->** Υλοποιεί την λειτουργία της αναζήτησης χρησιμοποιώντας τον αλγόριθμο της δυαδικής αναζήτησης. Επιστρέφει το πλήθος εμφάνισης του ζεύγους αν η αναζήτηση ήταν επιτυχής αλλιώς -1 αν ήταν ανεπιτυχής

3. ΑΠΛΟ ΔΥΑΔΙΚΟ ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ (BinaryTree)

Η δομή του απλού δυαδικού δένδρου αναζήτησης έχει την εξής μορφή :



Γνωρίζουμε ότι η πολυπλοκότητα, για την μέση περίπτωση του είναι O(n) για τον χώρο, O(log n) για την εισαγωγή, την διαγραφή και την αναζήτηση. Η πολυπλοκότητα στην χειρότερη περίπτωση, γίνεται O(n) για όλες τις

παραπάνω λειτουργίες. Το δυαδικό δένδρο είναι μια δενδρική δομή στην οποία κάθε κόμβος έχει το πολύ δύο παιδιά ,που αναφέρονται ως το αριστερό παιδί κι το δεξί παιδί. Η δομή αυτή χρησιμοποιείται συνήθως για αναζήτηση κι ταξινόμηση καθώς, λόγω της μορφής της, αποθηκεύει τα δεδομένα ιεραρχικά.
Μερικές βασικές λειτουργίες της δομής είναι :

- Εισαγωγή
- Διαγραφή
- Διαπέραση (inorder/postorder/preorder)

Με αυτή την λογική πέρα από τις βασικές μεθόδους της κλάσης που χρησιμοποιούνται για την ορθή δομή κι λειτουργία του binary tree ώστε να είναι σε θέση πλέον να επεξεργαστεί τα δεδομένα που του παρέχονται ως είσοδος(πίνακας pairs) :

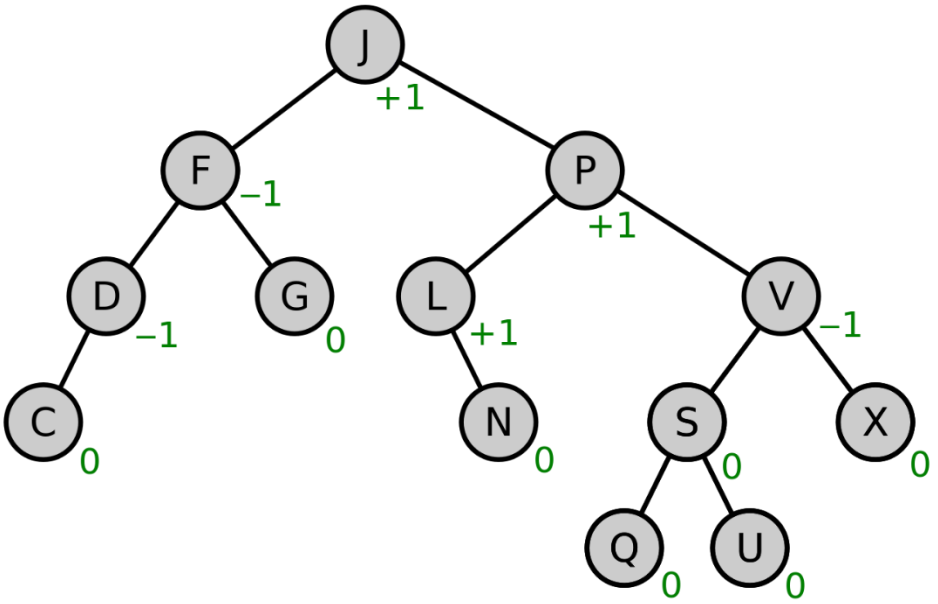
- BinaryTree // Constructor
- BinaryTree // Destructor
- getSize // getters
- getClass

Επιπλέον, χρησιμοποιήσαμε τις παρακάτω συναρτήσεις :

- **DestroyTree->** Υλοποιεί την λειτουργία της διαγραφής του δένδρου (αριστερού κι δεξιού κόμβου)
- **Insert->** Υλοποιεί τη λειτουργία της εισαγωγής στοιχείου στο δένδρο. Η συνάρτηση καλείται με αναδρομικό τρόπο ώστε να ξεκινώντας από την ρίζα του δένδρου και με την βοήθεια επαναλαμβανόμενων συγκρίσεων ψάχνει τη κατάλληλη θέση (στον αριστερό ή στον δεξί κόμβο) να τοποθετήσει το στοιχείο διατηρώντας την ιεραρχία που υπάρχει στο δέντρο
- **Search->**Υλοποιεί την λειτουργία της αναζήτησης ζεύγους στον πίνακα κι επιστρέφει το πλήθος εμφανίσεων αυτού
- **Inorder->** Υλοποιεί τη λειτουργία της διαπέρασης για την εκτύπωση των στοιχείων του δέντρου . Χρησιμοποιείται inorder δηλαδή ενδο-διατεταγμένη διεύλευση. Συγκεκριμένα, επισκεπτόμαστε αρχικά τον αριστερό κόμβο, ύστερα την ρίζα κι τέλος τον δεξί κόμβο.

4. ΔΥΑΔΙΚΟ ΔΕΝΔΡΟ ΑΝΑΖΗΤΗΣΗΣ AVL (AvlTree)

Η δομή ενός δυαδικού δένδρου αναζήτησης AVL έχει την εξής μορφή :



Γνωρίζουμε ότι η πολυπλοκότητα του AVL δέντρου είναι O(n) για τον χώρο, O(log n) για την εισαγωγή, την διαγραφή και την αναζήτηση. Αυτή είναι και για την χειρότερη περίπτωση. Το απλό δυαδικό δέντρο αναζήτησης(**BinaryTree.h)** και το δυαδικό δέντρο αναζήτησης AVL **(AvlTree.h)** δημιουργήθηκαν παράλληλα, καθώς χρησιμοποιήσαμε κληρονομικότητα μεταξύ τους. Συγκεκριμένα, το AvlTree(παράγωγος κλάση) κληρονομεί τις public μεθόδους του BinaryTree(κλάση βάση).

Γενικά στα AVL δένδρα :

- Η διαφορά των υψών του αριστερού κι του δεξιού υποδένδρου δεν πρέπει να ξεπερνάει το 1. Αυτό, πρέπει να ισχύει αναδρομικά για όλους τους κόμβους του δένδρου
- Αναζήτηση ίδια όπως κι στα απλό δυαδικό δένδρο αναζήτησης
- Εισαγωγή/Διαγραφή πρέπει να γίνεται έλεγχος για τον περιορισμό AVL κ να γίνουν οι απαραίτητες αλλαγές στην διαφορά των υψών των υποδένδρων αν χρειάζεται (περιστροφές)

Πέρα από τις βασικές μεθόδους της κλάσης που χρησιμοποιούνται για την ορθή δομή κι λειτουργία του πίνακα ώστε να είναι σε θέση πλέον να επεξεργαστεί τα δεδομένα που της παρέχονται ως είσοδος:

- AVLtree // Constructor
- ~AVLtree // Destructor

- getSize // getters
- getClass

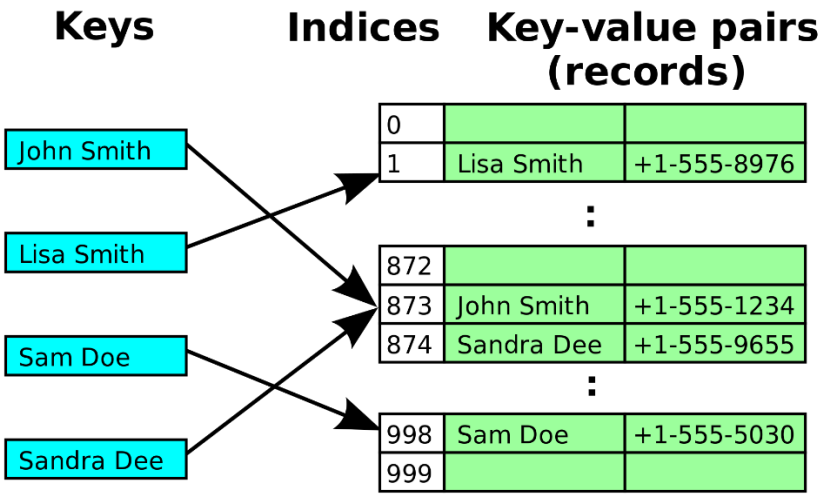
Επιπλέον, χρησιμοποιήσαμε τις παρακάτω συναρτήσεις :

- **DestroyTree->** Υλοποιεί την λειτουργία της διαγραφής του δένδρου (αριστερού κι δεξιού κόμβου)
- **Insert->** Υλοποιεί την λειτουργία της εισαγωγής στοιχείων στο δένδρο. Η συνάρτηση καλείται με αναδρομικό τρόπο ώστε να ξεκινήσουμε από την ρίζα του δένδρου και με την βοήθεια επαναλαμβανόμενων συγκρίσεων ψάχνει τη κατάλληλη θέση (στον αριστερό ή στον δεξί κόμβο) να τοποθετήσει το στοιχείο διατηρώντας την ιεραρχία που υπάρχει στο δέντρο
- **Searchword->** Ψάχνω αναδρομικά ένα συγκεκριμένο ζεύγος pair
- **Deleteword->** Συνάρτηση που διαγράφει ένα ζεύγος από το δέντρο
- **Min->** Συνάρτηση που βρίσκει το μικρότερο ζεύγος (αλφαβητικά)
- **Height->** Συνάρτηση που υπολογίζει το ύψος του δέντρου
- **Difference->** Συνάρτηση που υπολογίζει την διαφορά των δυο υψών μεταξύ του αριστερού κι το δεξιού υπόδενδρου(δεν πρέπει να ξεπερνάει την μονάδα)
- **Balanced->**Συνάρτηση που είναι υπεύθυνη να διατηρεί την ισορροπία του AVL δένδρου καλώντας συναρτήσεις ώστε να υλοποιήσουν περιστροφές
- **Rightright->** Συνάρτηση που υλοποιεί right right περιστροφή. Περίπτωση RR (Right-Right): Και οι δύο ακμές οδηγούν δεξιά. θα εκτελούμε μια αριστερή περιστροφή γύρω από τον κρίσιμο κόμβο.
- **Leftleft->** Συνάρτηση που υλοποιεί left left περιστροφή. Περίπτωση LL: Και οι δύο ακμές οδηγούν αριστερά. Είναι συμμετρική της περίπτωσης RR. Μία δεξιά περιστροφή γύρω από τον κρίσιμο κόμβο αρκεί για να επιλυθεί το πρόβλημα με το balance του
- **Rightleft->** Συνάρτηση που υλοποιεί right left περιστροφή. Περίπτωση RL: Η πρώτη ακμή οδηγεί δεξιά και η δεύτερη αριστερά. Απαιτούνται δύο περιστροφές, μια δεξιά περιστροφή γύρω από τον επόμενο του κρίσιμου κόμβου στο μονοπάτι που οδηγεί στον v και μια αριστερή περιστροφή γύρω από τον κρίσιμο κόμβο

- **Leftright->** Συνάρτηση που υλοποιεί left right περιστροφή. Περίπτωση LR: Η πρώτη ακμή οδηγεί αριστερά και η δεύτερη δεξιά. θ Είναι συμμετρική της περιπτώσεως RL. Απαιτούνται δύο περιστροφές, μια αριστερή περιστροφή γύρω από τον επόμενο του κρίσιμου κόμβου στο μονοπάτι που οδηγεί στον v και μια δεξιά περιστροφή γύρω από τον κρίσιμο κόμβο
- **Inorder->** Αναδρομική συνάρτηση που υλοποιεί ενδο-διατεταγμένη διαπέραση. Δηλαδή, για κάθε κόμβο επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τον ίδιο τον κόμβο και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου
- **Preorder->** Αναδρομική συνάρτηση που υλοποιεί προ-διατεταγμένη διαπέραση. Δηλαδή, για κάθε κόμβο, επισκεπτόμαστε πρώτα τον ίδιο τον κόμβο, έπειτα τους κόμβους του αριστερού του υποδένδρου και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου
- **Postorder->** Αναδρομική συνάρτηση που υλοποιεί μετα-διατεταγμένη διαπέραση. Δηλαδή, για κάθε κόμβο, επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τους κόμβους του δεξιού του υποδένδρου και στη συνέχεια τον ίδιο τον κόμβο.

5. ΠΙΝΑΚΑΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ ΜΕ ΑΝΟΙΚΤΗ ΔΙΕΥΘΥΝΣΗ (HashMap)

Η δομή του απλού δυαδικού δένδρου αναζήτησης έχει την εξής μορφή :



Γνωρίζουμε ότι η πολυπλοκότητα της δομής HashMap, στην μέση περίπτωση, είναι O(n) για τον χώρο, O(1) για την εισαγωγή, την διαγραφή και την

αναζήτηση. Η δομή αυτή, έχει μια συνάρτηση κατακερματισμού. Η συνάρτηση αποδίδει μία τιμή στα στοιχεία που θέλουμε να αποθηκεύσουμε. Αυτή η τιμή προσαρμόζεται στα όρια ενός πίνακα και έτσι βρίσκουμε την θέση στην οποία θα εισαχθεί ή θα υπάρχει ένα στοιχείο κατά την εισαγωγή και την αναζήτηση αντίστοιχα. Στο πρόγραμμα μας υλοποιήσαμε κατακερματισμό με ανοιχτή διεύθυνση. Έτσι, σε περίπτωση σύγκρουσης των τιμών δύο διαφορετικών στοιχείων πηγαίνουμε στην αμέσως επόμενη θέση και προσπαθούμε να εισάγουμε (ή να βρούμε) το στοιχείο εκεί.

Οι λειτουργίες ενός πίνακα κατακερματισμού που θα πρέπει να συμπεριλάβουμε στον κώδικα είναι :

- Εισαγωγή
- Αναζήτηση

Βασικές μέθοδοι της κλάσης είναι οι :

- HashMap // Constructor
- HashMap // Destructor
- getSize // getters
- getClass
- getComparisons

Παράλληλα, χρησιμοποιήσαμε τις παρακάτω συναρτήσεις :

- **Insert->** Υλοποιεί την λειτουργία της εισαγωγής στοιχείων στον πίνακα. Αρχικά, μετατρέπει το στοιχείο που θέλουμε να εισάγουμε σε μία θέση του πίνακα Αν είναι κενή εισάγει το στοιχείο, διαφορετικά, προσπαθεί στην επόμενη θέση μέχρι να βρεθεί η πρώτη κενή.
- **Search->** Υλοποιεί την αναζήτηση ενός στοιχείου στον πίνακα κατακερματισμού. Μετατρέπει το στοιχείο που θέλουμε να αναζητήσουμε σε μία θέση του πίνακα με την ίδια συνάρτηση κατακερματισμού που χρησιμοποιεί η insert. Αν υπάρχει άλλο στοιχείο στην θέση, αναζητά το στοιχείο στις επόμενες θέσεις μέχρι να το βρει (επιστρέφει το πλήθος εμφανίσεων) ή μέχρι να βρει την πρώτη κενή θέση (επιστρέφει 0, το στοιχείο δεν υπάρχει στον πίνακα)
- **Print->** Υλοποιεί την λειτουργία της εκτύπωσης των στοιχείων του πίνακα.

Τέλος, από τις υλοποιήσεις των παραπάνω δομών και του προγράμματος συνολικά, τρέχοντας το πρόγραμμα για το αρχείο (small-file) που μας δόθηκε παρατηρήσαμε μεγάλες διαφορές στους χρόνους που έκανε η κάθε δομή να ολοκληρώσει την διαδικασία εισαγωγής και αναζήτησης των ζευγών των λέξεων. Το υποσύνολο Q περιείχε 10000 ζεύγη λέξεων για τις αναζητήσεις. Παρακάτω, είναι κάποια ενδεικτικά αποτελέσματα, τα οποία είναι πολύ κοντά στο μέσο όρο που χρειαζόταν η κάθε δομή να εκτελέσει τις λειτουργίες.

```
Creating Unsorted Array
Duration is 49585 microseconds.

Creating Sorted Array
Duration is 68321 microseconds.

Creating Binary Tree
Duration is 1995 microseconds.

Creating AVL Tree
Duration is 397431 microseconds.

Creating Hash Map
Duration is 1321 microseconds.

Searching Q subset in Unsorted Array
Duration is 1108 microseconds.

Searching Q subset in Sorted Array
Duration is 67 microseconds.

Searching Q subset in Binary Tree
Duration is 84 microseconds.

Searching Q subset in AVL Tree
Duration is 77 microseconds.

Searching Q subset in Hash Map
Duration is 57 microseconds.
```

Κατά την εισαγωγή των δεδομένων παρατηρούμε μεγάλη διακύμανση στο χρόνο τον οποίο χρειάστηκε η κάθε δομή. Ο ταξινομημένος πίνακας πήρε περισσότερη ώρα να ολοκληρώσει την διαδικασία εισαγωγής σε σχέση με τον αταξινόμητο, λόγω της συνεχούς διατήρησης της ταξινόμησης. Περισσότερο χρόνο πήρε η δομή του δένδρου AVL, λόγω των περιστροφών. Αυτές όμως βελτιώνουν την ταχύτητα αναζήτησης του δένδρου σε σχέση με το απλό δυαδικό δένδρο. Η δομή που εκτέλεσε την λειτουργία εισαγωγής στον λιγότερο χρόνο είναι το “HashMap”.

Στην αναζήτηση, παρατηρούμε πως ο αταξινόμητος πίνακας έχει σημαντικά χαμηλότερη απόδοση και ο χρόνος που χρειάζεται για να αναζητήσουμε τα στοιχεία στον πίνακα είναι υπερπολλαπλάσιος του χρόνου που χρειάζονται οι υπόλοιπες δομές. Το δένδρο AVL είναι ταχύτερο από το απλό δυαδικό δένδρο. Η δομή HashMap είναι ταχύτερη από τις υπόλοιπες και στην αναζήτηση των στοιχείων. Για να επιτευχθεί αυτό βέβαια, καταλαμβάνει περισσότερο χώρο στην μνήμη από τις υπόλοιπες δομές.