ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Αναφορά εργασίας Εαρινού Εξαμήνου 2020

Φοιτητές: Χριστίδης Γεώργιος (ΑΕΜ: 3850)
Παπαστέργιος Βασίλειος (ΑΕΜ: 3651)
Μάθημα: Δομές Δεδομένων
Διδάσκων Καθηγητής: Παπαδόπουλος Απόστολος

Πίνακας περιεχομένων

Εισαγωγή

Το παρόν έγγραφο παραδίδεται συμπληρωματικά με τα αρχεία κώδικα σε γλώσσα προγραμματισμού C++ που δημιουργήθηκαν από τους γράφοντες, στο πλαίσιο της εργασίας του Εαρινού Εξαμήνου 2020 για το μάθημα των Δομών Δεδομένων. Ειδικότερα, στόχος του παρόντος εγγράφου είναι να παρουσιάσει, να επεξηγήσει και να αναλύσει τη λειτουργία του προγράμματος σε γλώσσα προγραμματισμού C++ που εκπονήθηκε για να ικανοποιήσει τα ζητούμενα της εργασίας αυτής.

Κεντρική ιδέα της εργασίας ήταν η τριβή με διάφορες δομές δεδομένων και η υλοποίηση μερικών εξ' αυτών σε γλώσσα προγραμματισμού C++, με σκοπό την αποθήκευση πληροφορίας κειμένου. Πιο συγκεκριμένα, ζητήθηκε η υλοποίηση των εξής τριών δομών δεδομένων:

1. απλό δυαδικό δένδρο αναζήτησης
2. δυαδικό δένδρο αναζήτησης τύπου AVL
3. πίνακας κατακερματισμού με ανοικτή διεύθυνση.

Αξιοποιώντας τις τρεις αυτές υλοποιήσεις, το πρόγραμμα εστιάζει στην αποθήκευση των διαφορετικών λέξεων ενός κειμένου αλλά και της συχνότητας εμφάνισης αυτών. Απώτερος στόχος είναι η επιτέλεση βασικών ενεργειών (όπως προσπέλαση, αναζήτηση, διαγραφή κ.α.) επί των δεδομένων αυτών, στα πρότυπα που η κάθε μία από τις δομές επιτάσσει. Ως είσοδος δίνεται ένα αρχείο απλού κειμένου (επέκτασης .txt), στο οποίο βρίσκεται η πληροφορία (λέξεις και συχνότητα) για την οποία ενδιαφερόμαστε.

Ξεκινώντας την εκτέλεσή του, το πρόγραμμα αρχικοποιεί τις τρεις δομές, δημιουργώντας από ένα στιγμιότυπο της κάθε μίας. Στα τρία αυτά στιγμιότυπα των δομών θα αποθηκευτεί η πληροφορία του κειμένου. Καθώς το αρχείο εισόδου διαβάζεται, οι λέξεις του διαχωρίζονται και αποθηκεύονται ταυτόχρονα και στις τρεις δομές. Στο τέλος, επομένως, των διαδικασιών της αρχικοποίησης και επεξεργασίας του αρχείου εισόδου, θα έχει δημιουργηθεί ένα απλό δυαδικό δέντρο αναζήτησης, ένα δέντρο αναζήτησης τύπου AVL και ένας πίνακας κατακερματισμού ανοικτής διεύθυνσης που θα περιέχουν ως στοιχεία τους τις μοναδικές λέξεις το αρχείου εισόδου και τη συχνότητα εμφάνισης αυτών μέσα στο κείμενο. Με άλλα λόγια, το ίδιο περιεχόμενο (μοναδικές λέξεις του αρχείου κειμένου) αποθηκεύεται σε τρεις διαφορετικές δομές δεδομένων.

Η διαπίστωση αυτή αποτελεί βασική πτυχή της εργασίας, καθώς δημιουργεί ευνοϊκό περιβάλλον για τη σύγκριση των τριών δομών μεταξύ τους, ως προς βασικές πράξεις επί των δεδομένων. Το γεγονός πως οι ίδιες λέξεις τροφοδοτούνται στις δομές, διαμορφώνει ένα ιδανικό πλαίσιο σύγκρισης των (σχετικών) χρόνων και κατ' επέκταση της αποδοτικότητας κάθε δομής αναφορικά με λειτουργίες, όπως η αναζήτηση, η διαγραφή κλπ.

Σε κάθε εκτέλεση, λοιπόν, του προγράμματος, ένα μέρος των λέξεων που αποθηκεύονται στις δομές, επιλέγονται με τυχαίο τρόπο και εναποτίθενται (πέρα από τις δομές) και σε ένα σύνολο ερωτημάτων (Queries' Set ή απλά Q). Όπως υποδηλώνει και το όνομά του, το σύνολο αυτό περιέχει τις λέξεις εκείνες του αρχικού κειμένου τις οποίες μπορούμε να αξιοποιήσουμε προκειμένου να επιτελέσουμε πράξεις των δομών, μετρώντας και συγκρίνοντας την αποδοτικότητά τους.

Η σύσταση του Q, επομένως, διαφέρει από εκτέλεση σε εκτέλεση, καθώς οι λέξεις επιλέγονται με τυχαίο τρόπο και το πλήθος τους μπορεί να καθοριστεί από το χρήστη. Στην παρούσα εργασία, οι παραπάνω αξιώσεις θεωρούνται επαρκείς ώστε να εξάγουμε με ασφάλεια, γενικής φύσης και εμπειρικής προσέγγισης συμπεράσματα για την αποδοτικότητα των δομών που υλοποιήθηκαν.

Εν κατακλείδι, δεδομένης της γενικότερης λειτουργικότητας του προγράμματος, η προσέγγιση που ακολουθήθηκε για την τελική υλοποίησή του ήταν ιεραρχική (top-down). Το τελικό ζητούμενο διασπάστηκε σε μικρότερα, επιμέρους υπο-προβλήματα, τα οποία μελετήθηκαν και υλοποιήθηκαν ανεξάρτητα. Στις σελίδες που ακολουθούν, παρουσιάζονται και αναλύονται οι υπο-ενότητες που οδήγησαν την πορεία εκπόνησης της εργασίας από τους γράφοντες.

Ενότητα Α΄: Χειρισμός αρχείου εισόδου – Τροφοδότηση Δομών

Παραδοτέα αρχεία που εμπλέκονται: ***main.cpp, TextFileHandling.h*** *&* ***TextFileHandling.cpp.***

Λαμβάνοντας υπ' όψη τις αξιώσεις της εκφώνησης, ρεύμα εισόδου του προγράμματος είναι ένα αρχείο απλού κειμένου (.txt) στο οποίο περιέχεται η πληροφορία για την οποία ενδιαφερόμαστε. Σημαντικό ρόλο στην οικονομία του προγράμματος διαδραματίζει ο χειρισμός του αρχείου αυτού. Βασικός στόχος είναι η επεξεργασία της αρχικής, ακατέργαστης πληροφορίας, όπως αυτή βρίσκεται στο αρχείο εισόδου, και η μετατροπή της σε μια μορφή αξιοποιήσιμη από το πρόγραμμα για τις περεταίρω διαδικασίες του. Ο χειρισμός αυτός, στο παρόν πρόγραμμα, επιτυγχάνεται με τη χρήση μιας σειράς συναρτήσεων που έχουν υλοποιηθεί από τους γράφοντες και αξιοποιούνται από τη main του προγράμματος.

A_1. Διαθέσιμες συναρτήσεις

Οι συναρτήσεις που υλοποιήθηκαν και αξιοποιούνται από το συγκεκριμένο τμήμα του προγράμματος είναι οι εξής:

[1] **readContentAndCreateWordsFile(char *sourceFileName, char *allWordsDestinationFileName, unsigned long int &nofAllWords)**

επιτελεί την ανάγνωση του περιεχομένου του αρχείου εισόδου, τον διαχωρισμό των λέξεων αυτού και την εγγραφή τους σε ένα αρχείο κειμένου με ομοιόμορφο τρόπο (μία ανά γραμμή). Επιστρέφει με αναφορά το πλήθος των λέξεων του αρχικού κειμένου που προσπελάστηκαν

[2] **erasePunctuationMarks(string &givenString)**

Δοθείσης μιας συμβολοσειράς, απαλείφει από αυτήν σημεία στίξης και ειδικούς χαρακτήρες (πλην παύλας)

[3] **isAlphanumeric(const char &givenCharacter)**

Δοθέντος ενός χαρακτήρα, αποφαίνεται αν αυτός είναι αλφαριθμητικός (γράμμα λατινικού αλφαβήτου ή αριθμός) ή όχι

[4] **isDash(const char &givenCharacter)**

Δοθέντος ενός χαρακτήρα, αποφαίνεται αν αυτός είναι το σύμβολο της παύλας (-) ή όχι

[5] **tossCoin(double givenProbabilityPerThousand = 500)**

Προσομοιώνει τη ρίψη ενός κέρματος με δύο πιθανά αποτελέσματα. Δέχεται ως όρισμα την πιθανότητα επί τοις χιλίοις του επιτυχημένου σεναρίου. Ως προεπιλογή, τα δύο ενδεχόμενα (αποτυχία – επιτυχία) είναι ισοπίθανα

A_2. Περιγραφή λειτουργίας σε φυσική γλώσσα

Προτού προχωρήσουμε στην ανάλυση των βημάτων της διαχείρισης του αρχείου εισόδου, κρίνεται σκόπιμο να θέσουμε τις υποθέσεις - παραδοχές της θεώρησής μας. Ειδικότερα, ως λέξη θεωρήθηκε οποιαδήποτε ακολουθία αλφαριθμητικών συμβόλων που βρίσκεται μεταξύ δύο κενών χαρακτήρων (space), σημείων στίξης ή και των δύο. Ο μόνος χαρακτήρας (εκτός των αλφαριθμητικών) που δύναται να διατηρηθεί στο εσωτερικό μιας λέξης είναι αυτός της παύλας (-), καθώς πολλές είναι οι λέξεις της αγγλικής γλώσσας που εμφανίζουν τον χαρακτήρα αυτό ως ενωτικό των συνθετικών τους. Οποιοδήποτε άλλο σύμβολο, σημείο στίξης ή ειδικός χαρακτήρας απαλείφεται από τη λέξη. Κύριος λόγος που αιτιολογεί την πρακτική αυτή είναι η διατήρηση της καθαρότητας των δομών δεδομένων, ώστε αυτές να περιλαμβάνουν όσο το δυνατόν αμιγείς λέξεις.

Στο σημείο αυτό, είμαστε σε θέση να παρουσιάσουμε τα βήματα διαχείρισης του αρχείου εισόδου, όπως αυτά εκτελούνται από το πρόγραμμα. Η πρώτη βασική διαδικασία (συνάρτηση [1]) που λαμβάνει χώρα είναι το άνοιγμα και η ανάγνωση των δεδομένων του αρχείου, τα οποία βρίσκονται ακόμη σε ακατέργαστη μορφή. Αξιοποιώντας αντικείμενα και υπερφορτωμένους τελεστές της C++ κλάσης ρευμάτων εισόδου, η ανάγνωση των δεδομένων από το αρχείο γίνεται επαναληπτικά, μέχρι να διαβαστεί όλο το περιεχόμενο του αρχείου.

Κάθε φορά, διαβάζεται από το αρχείο μια ακολουθία συμβόλων, έως ότου βρεθεί ο χαρακτήρας του κενού ή αυτός της αλλαγής γραμμής. Την ανάγνωση αυτή ακολουθεί η επεξεργασία της συμβολοσειράς και η εναπόθεσή της σε ένα καθαρό αρχείο, ώστε να είναι εύκολα αξιοποιήσιμη.

Αναλυτικότερα, αμέσως μετά την ανάγνωση, η συμβολοσειρά απαλλάσσεται από σημεία στίξης ή ειδικούς χαρακτήρες (τυχόν παύλες στο εσωτερικό διατηρούνται). Η διαδικασία αυτή υλοποιείται στη συνάρτηση [2], η οποία επιστρέφει τη λέξη, έχοντας εξαλείψει από αυτή μη επιθυμητούς χαρακτήρες. Επικουρικό ρόλο στη διαδικασία αυτή επιτελούν οι συναρτήσεις [3] και [4], οι οποίες, επιπλέον, προσδίδουν μεγαλύτερη αναγνωσιμότητα στον κώδικα.

Εκτελώντας τα βήματα αυτά, η στοίβα χρόνου εκτέλεσης επιστρέφει στη συνάρτηση [1] η οποία κλείνει τον κύκλο της μιας επανάληψης «καθαρογράφοντας» την επεξεργασμένη λέξη σε ένα βοηθητικό αρχείο απλού κειμένου. Τα παραπάνω βήματα (ανάγνωση συμβολοσειράς – επεξεργασία – εγγραφή τελικής λέξης σε ενδιάμεσο αρχείο) επαναλαμβάνονται, έως ότου διαβαστεί όλο το περιεχόμενο του αρχείου εισόδου.

Στο τέλος, επομένως, αυτής της ακολουθίας εντολών και συναρτήσεων έχει επιτευχθεί η ανάγνωση του περιεχομένου του αρχείου εισόδου και η δημιουργία ενός «καθαρού», ενδιάμεσου αρχείου κειμένου που περιέχει τις αμιγείς λέξεις του αρχικού κειμένου. Με τον τρόπο αυτό, η πληροφορία έχει λάβει μορφή εύκολα αξιοποιήσιμη, προκειμένου να τροφοδοτηθεί στις δομές.

Η αρχικοποίηση και τροφοδοσία των δομών γίνεται από τη συνάρτηση main. Η γραμμογράφηση του ενδιάμεσου αρχείου (περιέχει όλες τις λέξεις, μία ανά γραμμή) μας επιτρέπει να διαχειριστούμε πολύ εύκολα την πληροφορία και να εισάγουμε τις λέξεις στις τρεις δομές ταυτόχρονα. Παράλληλα με την εισαγωγή στις δομές, γίνεται και η δημιουργία του συνόλου Q. Στη διαδικασία αυτή αξιοποιείται η συνάρτηση [5], εξασφαλίζοντας την τυχαιότητα της σύστασης του συνόλου. Οι λέξεις του Q αποθηκεύονται επίσης σε ένα ενδιάμεσο, βοηθητικό αρχείο με ανάλογη γραμμογράφηση (μία λέξη ανά γραμμή). Το αρχείο αυτό θα αξιοποιηθεί σε μεταγενέστερο στάδιο, αυτό των ερωτημάτων επί των δομών.

Συνοπτικά, μετά το πέρας της εκτέλεσης του τμήματος αυτού, έχουν επιτευχθεί τα ακόλουθα:

- ανάγνωση της πληροφορίας του αρχείου εισόδου
- καθαρισμός των λέξεων του αρχείου εισόδου από ανεπιθύμητα σύμβολα
- εγγραφή των επεξεργασμένων πλέον λέξεων σε ένα ενδιάμεσο, βοηθητικό αρχείο (με όνομα Total Words.txt) με ευνοϊκή γραμμογράφηση
- αρχικοποίηση των τριών δομών
- τροφοδότηση των λέξεων από το ενδιάμεσο αρχείο στις δομές
- δημιουργία του συνόλου Q
- εγγραφή των λέξεων του Q σε ένα ενδιάμεσο, βοηθητικό αρχείο (με όνομα Random Words.txt) με ευνοϊκή γραμμογράφηση

Στο σημείο αυτό, ολοκληρώνεται το πρώτο τμήμα της εκτέλεσης του προγράμματος που αφορά το χειρισμό του αρχείου εισόδου και ορισμένες προπαρασκευαστικές ενέργειες για την οικονομία και τη λειτουργικότητα του προγράμματος. Τα τμήματα του προγράμματος που έπονται είναι αυτά της υλοποίησης των τριών δομών. Εντελώς αντίστοιχα με όσα αναλύθηκαν έως τώρα, τα τμήματα αυτά μελετήθηκαν και υλοποιήθηκαν ως υπο-προβλήματα του τελικού ζητούμενου. Οι δομές αυτές παρουσιάζονται ακολούθως, στις ενότητες Β’ και Γ’.

Ενότητα Β': Δομές Δένδρων

Παραδοτέα αρχεία που εμπλέκονται: ***bst.h, bst.cpp, avl.h*** *&* ***avl.cpp.***

Στην παρούσα ενότητα παρουσιάζεται η υλοποίηση και η λειτουργία των δομών δένδρων που υλοποιήθηκαν στο πλαίσιο της εργασίας. Πιο συγκεκριμένα, οι εν λόγω δομές είναι αυτή του απλού δυαδικού δέντρου αναζήτησης και αυτή του δυαδικού δέντρου αναζήτησης τύπου AVL. Ιδιαίτερη έμφαση δίνεται στις αξιώσεις και τις προϋποθέσεις που λήφθηκαν υπ' όψιν από τους γράφοντες, κατά την υλοποίηση των δομών.

Στην υλοποίηση των δύο δομών, όπως ορίζεται και από την εκφώνηση της εργασίας- συμπεριλήφθηκαν μέθοδοι για την εισαγωγή, αναζήτηση και διαγραφή κόμβων των δέντρων, δοθείσης μιας συμβολοσειράς. Συμπληρωματικά με αυτές, περιλαμβάνονται και μέθοδοι διάσχισης των κόμβων των δέντρων για την εκτύπωση των δεδομένων τους. Η εκτύπωση αυτή διακρίνεται σε υποκατηγορίες, με βάση τη σειρά επίσκεψης της ρίζας κάθε υποδένδρου της δομής. Έτσι, μπορεί να είναι προδιατεταγμένη, ενδοδιατεταγμένη ή μεταδιατεταγμένη (inorder, preorder και postorder traversal).

Σε μια αρχική, επιφανειακή ανάλυση. οι κόμβοι των δέντρων αναπαρίστανται με την βοήθεια μίας ένθετης κλάσης στο εσωτερικό των κλάσεων των δέντρων, στην οποία αποθηκεύονται τα δεδομένα για τα οποία ενδιαφερόμαστε. Σε κάθε κόμβο αποθηκεύεται, συνεπώς, μία συμβολοσειρά (λέξη) και η συχνότητα εμφάνισής της στο αρχικό κείμενο. Βασικά στοιχεία του κάθε κόμβου είναι, επίσης, οι θέσεις μνήμης των δύο θυγατρικών κόμβων, δηλαδή δείκτες στη ρίζα του δεξιού και αριστερού υποδένδρου του. Στην περίπτωση του δέντρου τύπου AVL, έχει προστεθεί ένα ακόμη μέλος (μεταβλητή), στην οποία αποθηκεύεται το «ύψος» του εκάστοτε κόμβου. Ακολούθως περιγράφονται λεπτομερώς οι δύο αυτές κλάσεις δέντρων:

B_1. Απλό Δυαδικό Δέντρο Αναζήτησης

Παραδοτέα αρχεία που εμπλέκονται: ***bst.h*** *&* ***bst.cpp***

Με αφόρμηση από μια ετυμολογική προσέγγιση της ονομασίας της δομής, διαπιστώνουμε πως αυτή συμπυκνώνει και καθορίζει, εν πολλοίς, τις βασικές αρχές που διέπουν τη λειτουργία της. Πιο συγκεκριμένα, ο όρος «δυαδικό» καθορίζει πως κάθε κόμβος – ρίζα ενός υποδέντρου μπορεί να έχει το πολύ δύο θυγατρικούς κόμβους (αριστερό και δεξί υποδένδρο).

Στο σημείο αυτό, επενεργεί ο όρος «δέντρο αναζήτησης», διασαφηνίζοντας την ακριβή διάρθρωση των θυγατρικών κόμβων σε σχέση με τη ρίζα. Ορίζει, λοιπόν, πως όσα στοιχεία-κόμβοι έχουν τιμή μικρότερη από αυτή της ρίζας τοποθετούνται στο αριστερό υποδένδρο αυτής, ενώ όσα στοιχεία-κόμβοι έχουν τιμή μεγαλύτερη από αυτή τοποθετούνται στο δεξί υποδένδρο της. Οι ανισοτικές αυτές σχέσεις ισχύουν αναδρομικά για τις ρίζες κάθε υποδένδρου της αρχικής δομής. Με τον τρόπο αυτό, κάθε υποδένδρο του αρχικού δέντρου είναι στην ουσία και αυτό ένα (μικρότερο) δυαδικό δέντρο αναζήτησης.

Λαμβάνοντας υπ' όψιν τα παραπάνω, είμαστε σε θέση να παρουσιάσουμε σε τεχνικό επίπεδο υλοποίησης την κλάση του δυαδικού δέντρου αναζήτησης. Προς χάριν ευκολότερης οπτικής αναπαράστασης, τα ονόματα των ιδιωτικών μελών της κλάσης παρατίθενται σε κόκκινο χρώμα, ενώ τα αντίστοιχα δημόσια σε πράσινο:

[1] **class node**

~node() : *Καταστροφέας των κόμβων. Αποδεσμεύει την μνήμη που δεσμεύτηκε δυναμικά.*

string data : *Αποθηκεύει την συμβολοσειρά που αποτελεί τα δεδομένα του κάθε κόμβου*

node* right & **node* left** *: Αποθηκεύουν τις διευθύνσεις μνήμης δεξιού και αριστερό θυγατρ. κόμβου*

unsigned int frequency : *Αποθηκεύει τον αριθμό των φορών που μία συμβολοσειρά εισήχθη στο δέντρο*

[2] **root** : *Αποθηκεύει την θέση μνήμης του κόμβου της ρίζας*

[3] **createNode(string key)** :

Δεσμεύει δυναμικά χώρο στην μνήμη για την δημιουργία ενός καινούργιου κόμβου (τύπου node) και εισάγει στο πεδίο των δεδομένων του κόμβου την συμβολοσειρά key. Η συχνότητα εμφάνισης της λέξης ορίζεται σε 1, καθώς η μέθοδος καλείται μόνο την πρώτη φορά που εισάγεται η κάθε μοναδική συμβολοσειρά. Οι δύο θυγατρικοί κόμβοι λαμβάνουν τιμή ίση με NULL, καθώς δεν τους έχει αποδοθεί ακόμη κάποιο περιεχόμενο. Η μέθοδος επιστρέφει την διεύθυνση μνήμης του καινούργιου κόμβου που μόλις δημιουργήθηκε.

[4] **insert(string key, node* current)** :

Με αναδρομικές κλήσεις, αναζητεί την θέση στην οποία ανήκει η συμβολοσειρά key στο δέντρο ξεκινώντας την αναζήτηση από τον κόμβο current. Όταν η θέση αυτή βρεθεί, καλεί την createNode, ώστε να δημιουργήσει ένα καινούργιο κόμβο και να εισάγει την key στη θέση αυτή. Σε περίπτωση που η συμβολοσειρά υπάρχει ήδη στο δέντρο, αυξάνει τον μετρητή frequency κατά 1.

[5] **findMinNode(node* n)** :

Δοθέντος ενός μη κενού (υπο)δέντρου με ρίζα τον κόμβο n, η μέθοδος αναζητάει και επιστρέφει τον αριστερότερο κόμβο αυτού του δέντρου, ο οποίος έχει και την μικρότερη τιμή.

[6] **remove(node* current, string key)** :

*Με αναδρομικές κλήσεις, αναζητεί την θέση στην οποία βρίσκεται η συμβολοσειρά **key** στο δέντρο ξεκινώντας την αναζήτηση από τον κόμβο current. Αν ο κόμβος που περιέχει την **key** βρεθεί, υπάρχουν 3 περιπτώσεις, τις οποίες η συνάρτηση διαχειρίζεται ως ακολούθως:.*

***ΠΕΡΙΠΤΩΣΗ 0: Ο κόμβος δεν έχει παιδιά.** Σε αυτή την περίπτωση, ο γονικός κόμβος ενημερώνεται ώστε να μην έχει το αντίστοιχο παιδί («δείχνει» σε **NULL**) και αποδεσμεύεται η μνήμη του προς διαγραφή κόμβου.*

***ΠΕΡΙΠΤΩΣΗ 1: Ο κόμβος έχει ένα παιδί.** Σε αυτή την περίπτωση, ενημερώνεται ο γονικός κόμβος, ώστε να δείχνει στο παιδί του προς διαγραφή κόμβου. Κατόπιν, αποδεσμεύεται η μνήμη του προς διαγραφή κόμβου.*

***ΠΕΡΙΠΤΩΣΗ 2: Ο κόμβος έχει δύο παιδιά.** Σε αυτή την περίπτωση, αναζητείται ο διάδοχος του προς διαγραφή κόμβου, ο οποίος είναι (κατά σύμβαση) ο μικρότερος κόμβος του δεξιού υποδένδρου του προς διαγραφή κόμβου. Όταν βρεθεί, αντικαθιστούμε τα δεδομένα του προς διαγραφή κόμβου με τα δεδομένα του «διαδόχου» και καλούμε αναδρομικά την **remove,** ώστε να αποδεσμεύσει την θέση μνήμης της παλιάς θέσης του «διαδόχου».*

[7] **bst()**:

*Μέθοδος δόμησης του δυαδικού δένδρου αναζήτησης. Θέτει τον δείκτη του κόμβου της ρίζας του δέντρου ισο με **NULL,** ώστε να μπορεί να χρησιμοποιηθεί ο κόμβος πριν την εισαγωγή δεδομένων.*

[8] **~bst ()** :

Μέθοδος απόδόμησης του δυαδικού δένδρου αναζήτησης. Αποδεσμεύει την μνήμη του κόμβου της ρίζας και ο μεταγλωττιστής διαχειρίζεται την αυτόματη αποδέσμευση και των υπόλοιπων κόμβων.

[9] **search(string key)** :

*Δοθείσης μίας συμβολοσειράς **key**, αναζητεί τη συμβολοσειρά αυτή στο δέντρο. Η αναζήτηση υλοποιείται τεχνικά με διαδοχικές συγκρίσεις του περιεχομένου των κόμβων με το key. Σε περίπτωση που η key βρεθεί, επιστρέφει τον αριθμό των εμφανίσεων της συμβολοσειράς (συχνότητα). Στην περίπτωση που η συμβολοσειρά **key** δεν υπάρχει στο δέντρο, επιστρέφει 0.*

[10] **insert(string key)** :

*Βοηθητική συνάρτηση η οποία, δοθείσης μιας συμβολοσειράς **key**, ξεκινάει την αναδρομική αναζήτηση για την θέση εισαγωγής της **key** από την ρίζα του δέντρου, μέσω της υπερφορτωμένης έκδοσης της. Στην περίπτωση που το δέντρο είναι άδειο, δημιουργεί έναν κόμβο ρίζας και εισάγει την **key** εκεί.*

[11] **remove(string key)** :

*Βοηθητική συνάρτηση η οποία, δοθείσας μιας συμβολοσειράς **key**, ξεκινάει την αναδρομική αναζήτηση της **key** μέσω της υπερφορτωμένης έκδοσης της, με σκοπό την διαγραφή της από το δέντρο.*

[12] **preOrder / inOrder / postOrder (node* current, ostream &output)**:

*Αναδρομικές συναρτήσεις οι οποίες εμφανίζουν ολόκληρο το δέντρο, τυπώνοντας διαδοχικά τα δεδομένα του κάθε κόμβου σύμφωνα με τις **preorder**, **inorder** και **postorder** διασχίσεις(traversals), αντιστοίχως. Η εκτύπωση των δεδομένων γίνεται σε αρχείο απλού κειμένου, το αντικείμενο ρεύματος του οποίου δέχονται με αναφορά.*

[13] **preOrder() / inOrder() / postOrder()** :

Βοηθητικές συναρτήσεις οι οποίες ξεκινούν την αναδρομική διαδικασία των υπερφορτωμένων εκδόσεων τους από τον κόμβο της ρίζας. Στις συναρτήσεις αυτές γίνεται σύνδεση ενός αρχείου εκτύπωσης με ένα αντικείμενο ρεύματος εξόδου. Το αντικείμενο αυτό περνά ως όρισμα στην υπερφορτωμένη εκδοχή της κάθε διάσχισης.

B_2. Δυαδικό Δέντρο Αναζήτησης τύπου AVL

*Παραδοτέα αρχεία που εμπλέκονται: **avl.h** & **avl.cpp.***

Το δυαδικό δέντρο αναζήτησης τύπου AVL αποτελεί, πρακτικά, μια υπο-κατηγορία του απλού δυαδικού δέντρου αναζήτησης, αν στην τελευταία προσθέσουμε το στοιχείο της ομοιόμορφης κατανομής υψών. Ειδικότερα, ένα δυαδικό δέντρο τύπου AVL διαθέτει όλα όσα περιεγράφηκαν στην προηγούμενη ενότητα, σε συνδυασμό με κάποιες επιπλέον μεθόδους.

Οι μέθοδοι αυτές επιτρέπουν στο δέντρο να ανακατανέμει τη διάρθρωση των κόμβων του, σε περίπτωση που διαταραχθεί η ομοιομορφία των υψών του. Πιο συγκεκριμένα, βασική αρχή του δέντρου AVL είναι πως για κάθε κόμβο θα πρέπει το ύψος του δεξιού και το αντίστοιχο ύψος του αριστερού υποδένδρου του να μην διαφέρουν περισσότερες από 1 μονάδα. Αυτή η διαφορά ονομάζεται και **παράγοντας ισορροπίας (balance factor)** του κόμβου.

Κατά τη διάρκεια, βέβαια, εισαγωγών ή διαγραφών από το δέντρο, ενδέχεται να διαταραχθεί αυτή η ομοιομορφία των υψών κάποιου υποδένδρου της δομής, δηλαδή ο παράγοντας ισορροπίας να γίνει 2 ή -2. Στην περίπτωση αυτή, θα χρειαστεί παρέμβαση προκειμένου να αποκατασταθεί η δομή του δέντρου. Η παρέμβαση αυτή λαμβάνει τη μορφή περιστροφών.

Όπως φαίνεται και στις συναρτήσεις **insert** και **remove**, υπάρχουν δύο είδη περιστροφών, οι **απλές** και οι **διπλές** περιστροφές. Οι **απλές** περιστροφές (LL/RR) κάνουν απλά μία περιστροφή στο υποδένδρο με ρίζα τον κόμβο του οποίου ο παράγοντας ισορροπίας είναι 2 ή -2. Οι **διπλές** περιστροφές εκτελούν πρώτα μία απλή περιστροφή στον θυγατρικό κόμβο (ο οποίος έχει παράγοντας ισορροπίας -1 ή 1) και έπειτα εκτελούν την αντίθετη περιστροφή στον κόμβο που εμφανίστηκε η ανισορροπία (παράγοντας ισορροπίας -2 ή 2).

Λαμβάνοντας υπ' όψιν τα παραπάνω, παρουσιάζεται η τεχνική ανάλυση της υλοποίησης την κλάση του δυαδικού δέντρου αναζήτησης. Σημειώνεται ότι οι περισσότερες μέθοδοι, καθώς και η ένθετη κλάση αναπαράστασης των κόμβων, είναι πανομοιότυπες με τις αντίστοιχες του απλού δυαδικού δέντρου αναζήτησης. Προς χάριν ευκολότερης οπτικής αναπαράστασης, τα ονόματα των ιδιωτικών μελών της κλάσης παρατίθενται σε κόκκινο χρώμα, ενώ τα αντίστοιχα δημόσια σε πράσινο:

[1] **class node**

~node() : *Καταστροφέας των κόμβων. Αποδεσμεύει την μνήμη που δεσμεύθηκε δυναμικά.*

string data : *Αποθηκεύει την συμβολοσειρά που αποτελεί τα δεδομένα του κάθε κόμβου*

node* right & **node* left** *: Αποθηκεύουν τις διευθύνσεις μνήμης δεξιού και αριστερό θυγατρ. κόμβου*

unsigned int frequency : *Αποθηκεύει τον αριθμό των φορών που μία συμβολοσειρά εισήχθη στο δέντρο*

unsigned int height : *Αποθηκεύει το ύψος του κόμβου στο δέντρο σύμφωνα με την αξίωση ότι τα φύλλα έχουν ύψος 1.*

[2] **root** : *Αποθηκεύει την θέση μνήμης του κόμβου της ρίζας*

[3] **height(node* n)**:

Βοηθητική μέθοδος η οποία επιστρέφει το ύψος του κόμβου στο δέντρο. Αν ο κόμβος δεν υπάρχει, επιστρέφει 0.

[4] **rRotate(node* &n)**:

Εκτελεί μία δεξιά περιστροφή -σύμφωνα με την θεωρία των δέντρων AVL- στο υποδένδρο με ρίζα τον κόμβο n. Στόχος είναι ο αριστερός θυγατρικός κόμβος του n να τεθεί ως η νέα ρίζα του υποδένδρου. Κατόπιν, γίνεται εκ-νέου υπολογισμός του ύψους των κόμβων και επιστρέφεται η θέση μνήμης της νέας ρίζας.

[5] **lRotate(node* &n)**:

Εκτελεί μία αριστερή περιστροφή -σύμφωνα με την θεωρία των δέντρων AVL- στο υποδένδρο με ρίζα τον κόμβο n. Στόχος είναι ο δεξιός θυγατρικός κόμβος του n να τεθεί ως η νέα ρίζα του υποδένδρου. Κατόπιν, γίνεται εκ-νέου υπολογισμός του ύψους των κόμβων και επιστρέφεται η θέση μνήμης της νέας ρίζας.

[6] **createNode(string key)** :

Δεσμεύει δυναμικά χώρο στην μνήμη για την δημιουργία ενός καινούργιου κόμβου (τύπου node) και εισάγει στο πεδίο των δεδομένων του κόμβου την συμβολοσειρά key. Η συχνότητα εμφάνισης της λέξης ορίζεται σε 1, καθώς η μέθοδος καλείται μόνο την πρώτη φορά που εισάγεται η κάθε μοναδική συμβολοσειρά. Οι δύο θυγατρικοί κόμβοι λαμβάνουν τιμή ίση με NULL, καθώς δεν τους έχει αποδοθεί ακόμη κάποιο περιεχόμενο. Η μέθοδος επιστρέφει την διεύθυνση μνήμης του καινούργιου κόμβου που μόλις δημιουργήθηκε.

[7] **insert(string key, node* current)** :

Με αναδρομικές κλήσεις, αναζητεί την θέση στην οποία ανήκει η συμβολοσειρά key στο δέντρο ξεκινώντας την αναζήτηση από τον κόμβο current. Όταν η θέση αυτή βρεθεί, καλεί την createNode, ώστε να δημιουργήσει ένα καινούργιο κόμβο και να εισάγει την key στη θέση αυτή. Σε περίπτωση που η συμβολοσειρά υπάρχει ήδη στο δέντρο, αυξάνει τον μετρητή frequency κατά 1. Μετά την εισαγωγή της συμβολοσειράς στο δέντρο, γίνεται έλεγχος της ισορροπίας του δέντρου -όπως ορίζει η θεωρία των δέντρων AVL. Αν διαπιστωθεί πως η ισορροπία των υψών έχει διαταραχθεί, τότε καλούνται οι συναρτήσεις **lRotate** και **rRotate**, *ώστε να εκτελέσουν τις αναγκαίες περιστροφές στους κόμβους και να επιλύσουν την ανισορροπία.*

[8] **findMinNode(node* n)** :

Δοθέντος ενός μη κενού (υπο)δέντρου με ρίζα τον κόμβο n, η μέθοδος αναζητεί και επιστρέφει τον αριστερότερο κόμβο αυτού του δέντρου, ο οποίος έχει και την μικρότερη τιμή.

[9] **remove(node* current, string key)** :

*Με αναδρομικές κλήσεις, αναζητεί την θέση στην οποία βρίσκεται η συμβολοσειρά **key** στο δέντρο ξεκινώντας την αναζήτηση από τον κόμβο current. Αν ο κόμβος που περιέχει την **key** βρεθεί, υπάρχουν 3 περιπτώσεις, τις οποίες η συνάρτηση διαχειρίζεται ως ακολούθως:.*

***ΠΕΡΙΠΤΩΣΗ 0: Ο κόμβος δεν έχει παιδιά.** Σε αυτή την περίπτωση, ο γονικός κόμβος ενημερώνεται ώστε να μην έχει το αντίστοιχο παιδί («δείχνει» σε **NULL**) και αποδεσμεύεται η μνήμη του προς διαγραφή κόμβου.*

***ΠΕΡΙΠΤΩΣΗ 1: Ο κόμβος έχει ένα παιδί.** Σε αυτή την περίπτωση, ενημερώνεται ο γονικός κόμβος, ώστε να δείχνει στο παιδί του προς διαγραφή κόμβου. Κατόπιν, αποδεσμεύεται η μνήμη του προς διαγραφή κόμβου.*

***ΠΕΡΙΠΤΩΣΗ 2: Ο κόμβος έχει δύο παιδιά.** Σε αυτή την περίπτωση, αναζητείται ο διάδοχος του προς διαγραφή κόμβου, ο οποίος είναι (κατά σύμβαση) ο μικρότερος κόμβος του δεξιού υποδένδρου του προς διαγραφή κόμβου. Όταν βρεθεί, αντικαθιστούμε τα δεδομένα του προς διαγραφή κόμβου με τα δεδομένα του «διαδόχου» και καλούμε αναδρομικά την **remove,** ώστε να αποδεσμεύσει την θέση μνήμης της παλιάς θέσης του «διαδόχου».*

Μετά από μία επιτυχή διαγραφή μιας συμβολοσειράς στο δέντρο, γίνεται έλεγχος της ισορροπίας του δε ντρου - όπως ορίζει η θεωρία των δέντρων AVL. Αν διαπιστωθεί πως η ισορροπία των υψών έχει διαταραχθεί, τότε καλούνται οι συναρτήσεις **lRotate** και **rRotate**, *ώστε να εκτελέσουν τις αναγκαίες περιστροφές στους κόμβους και να επιλύσουν την ανισορροπία.*

[10] **avl()**:

Μέθοδος δόμησης του δυαδικού δένδρου αναζήτησης AVL. Θέτει τον δείκτη του κόμβου της ρίζας του δέ-ντρου ίσο με ***NULL,*** *ώστε να μπορεί να χρησιμοποιηθεί ο κόμβος πριν την εισαγωγή δεδομένων.*

[11] **~avl()** :

Μέθοδος απόδόμησης του δυαδικού δένδρου αναζήτησης AVL. Αποδεσμεύει την μνήμη του κόμβου της ρίζας και ο μεταγλωττιστής διαχειρίζεται την αυτόματη αποδέσμευση και των υπόλοιπων κόμβων.

[12] **search(string key)** :

Δοθείσης μίας συμβολοσειράς ***key****, αναζητεί τη συμβολοσειρά αυτή στο δέντρο. Η αναζήτηση υλοποιείται τεχνικά με διαδοχικές συγκρίσεις του περιεχομένου των κόμβων με το key. Σε περίπτωση που η key βρεθεί, επιστρέφει τον αριθμό των εμφανίσεων της συμβολοσειράς (συχνότητα). Στην περίπτωση που η συμβολο-σειρά* ***key*** δεν υπάρχει στο δέντρο, επιστρέφει 0.

[13] **balanceFactor(node* n)** :

Βοηθητική συνάρτηση η οποία, επιστρέφει τον παράγοντα ισορροπίας ενός δοθέντος κόμβου n. Ως παράγο-ντας ισορροπίας ενός κόμβου ορίζεται η διαφορά του ύψους του αριστερού θυγατρικού κόμβου από το ύψος του δεξιού θυγατρικού κόμβου. Σε περίπτωση που ο κόμβος δεν υπάρχει, επιστρέφει 0.

[14] **insert(string key)** :

Βοηθητική συνάρτηση η οποία, δοθείσης μιας συμβολοσειράς ***key****, ξεκινάει την αναδρομική αναζήτηση για την θέση εισαγωγής της* ***key*** *από την ρίζα του δέντρου, μέσω της υπερφορτωμένης έκδοσης της. Στην περί-πτωση που το δέντρο είναι άδειο, δημιουργεί έναν κόμβο ρίζας και εισάγει την* ***key*** *εκεί.*

[15] **remove(string key)** :

Βοηθητική συνάρτηση η οποία, δοθείσας μιας συμβολοσειράς ***key****, ξεκινάει την αναδρομική αναζήτηση της* ***key*** *μέσω της υπερφορτωμένης έκδοσης της, με σκοπό την διαγραφή της από το δέντρο.*

[16] **preOrder / inOrder / postOrder (node* current, ostream &output)**:

Αναδρομικές συναρτήσεις οι οποίες εμφανίζουν ολόκληρο το δέντρο, τυπώνοντας διαδοχικά τα δεδομένα του κάθε κόμβου σύμφωνα με τις ***preorder****,* ***inorder*** *και* ***postorder*** *διασχίσεις(traversals), αντιστοίχως. Η εκτύ-πωση των δεδομένων γίνεται σε αρχείο απλού κειμένου, το αντικείμενο ρεύματος του οποίου δέχονται με αναφορά.*

[17] **preOrder() / inOrder() / postOrder()** :

Βοηθητικές συναρτήσεις οι οποίες ξεκινούν την αναδρομική διαδικασία των υπερφορτωμένων εκδόσεων τους από τον κόμβο της ρίζας. Στις συναρτήσεις αυτές γίνεται σύνδεση ενός αρχείου εκτύπωσης με ένα αντι-κείμενο ρεύματος εξόδου. Το αντικείμενο αυτό περνά ως όρισμα στην υπερφορτωμένη εκδοχή της κάθε διά-σχισης.

Ενότητα Γ΄: Πίνακας Κατακερματισμού Ανοικτής Διεύθυνσης

Παραδοτέα αρχεία που εμπλέκονται: ***HashTableEntry.h, HashTableEntry.cpp, openAddressHashTable.h*** *&* ***openAddressHashTable.cpp***

Πρόκειται για το τμήμα του προγράμματος που υλοποιεί την κλάση ενός πίνακα κατακερματισμού ανοικτής διεύθυνσης. Οι καταχωρήσεις στις θέσεις του πίνακα αποτελούν αντικείμενα μιας ανεξάρτητης κλάσης που υλοποιήθηκε από τους γράφοντες και έχει ως στόχο τη διευκόλυνση των διαδικασιών που υποστηρίζει η κλάση του πίνακα κατακερματισμού. Ακολούθως παρουσιάζονται οι δύο αυτές κλάσεις σε τμηματική απεικόνιση και λεκτική περιγραφή του τρόπου υλοποίησης και λειτουργίας τους.

$Γ_1$. Βοηθητική Κλάση: Στοιχείο του πίνακα κατακερματισμού

Παραδοτέα αρχεία που εμπλέκονται: ***HashTableEntry.h*** *&* ***HashTableEntry.cpp.***

Δεδομένων των αξιώσεων της εκφώνησης, η πληροφορία που ενδιαφερόμαστε να αποθηκεύσουμε στον πίνακα κατακερματισμού είναι όχι μόνο οι μοναδικές λέξεις που περιέχονται σε ένα αρχείο κειμένου, αλλά και η συχνότητα εμφάνισής τους σε αυτό.

Με άλλα λόγια, σε κάθε θέση του πίνακα, είναι χρήσιμο να αποθηκεύεται, εκτός από τη συμβολοσειρά που αναπαριστά μια λέξη, και ένας ακέραιος αριθμός που υποδηλώνει το πλήθος των εμφανίσεων της λέξης στο κείμενο από το οποίο αυτή προήλθε. Οι δύο αυτές τιμές, επομένως, μπορούν να «ομαδοποιηθούν» υπό την ευρύτερη έννοια μιας κλάσης. Η κλάση αυτή εννοιολογεί ένα στοιχείο, ή διαφορετικά μία καταχώρηση (entry), που αποθηκεύεται σε μία από τις διαδοχικές θέσεις μνήμης του πίνακα κατακερματισμού.

Στο πλαίσιο της θεώρησής μας, η κλάση αυτή περιλαμβάνει ως ιδιότητες μια συμβολοσειρά (word) και έναν ακέραιο αριθμό (frequency), αλλά παράλληλα έχει εμπλουτιστεί με μια σειρά μεθόδων που της προσδίδουν περεταίρω λειτουργικότητα. Οι μέθοδοι αυτές είναι σχετικά απλές ως προς την υλοποίησή τους, στηρίζονται στη γενική αρχή του τμηματικού προγραμματισμού ως πρακτική ανάπτυξης προγράμματος και προσδίδουν μεγαλύτερη αναγνωσιμότητα στα τμήματα κώδικα που αξιοποιούν την κλάση. Μια συνοπτική παρουσίαση των μεθόδων αυτών θα μπορούσε να είναι η ακόλουθη. Προς χάριν ευκολότερης οπτικής αναπαράστασης, τα ονόματα των ιδιωτικών μελών της κλάσης παρατίθενται σε κόκκινο χρώμα, ενώ τα αντίστοιχα δημόσια σε πράσινο:

[1] **HashTableEntry**() : *υπερφορτωμένος κατασκευαστής κλάσης για δυνατότητα κλήσης με και χωρίς ορίσματα*

[2] **getWord() - getFrequency() :** *Getters για τα ιδιωτικά μέλη της κλάσης*

[3] **setData(const string &givenString, unsigned long int givenFrequency = 1)**
Setter για τα ιδιωτικά μέλη της κλάσης. Η τοποθέτηση default ορίσματος στην συχνότητα εμφάνισης επιτρέπει την κλήση της μεθόδου με μόνο ένα όρισμα, αυτό της λέξης προς αποθήκευση. Στην περίπτωση αυτή, θεωρείται πως η λέξη έχει συχνότητα εμφάνισης ίση με 1.

[4] **increaseFrequencyBy (unsigned long int givenFrequency = 1)** και
decreaseFrequencyBy (unsigned long int givenFrequency = 1)
Βοηθητικές μέθοδοι που αυξομειώνουν τη συχνότητα εμφάνισης της λέξης κατά ένα επιθυμητό πλήθος εμφανίσεων. Το πλήθος αυτό είναι ως προεπιλογή ίσο με 1.

[5] **isOccupied()**
Βοηθητική μέθοδος που αποφαίνεται αν το συγκεκριμένο στοιχείο του πίνακα είναι κατειλημμένο ή όχι. Χρησιμοποιείται κυρίως για ευανάγνωστο κώδικα.

$Γ_2$. Κύρια κλάση: Πίνακας κατακερματισμού ανοικτής διεύθυνσης

Παραδοτέα αρχεία που εμπλέκονται: ***openAddressHashTable.h*** *&* ***openAddressHashTable.cpp***

Αξιοποιώντας τη βοηθητική κλάση που αναλύθηκε παραπάνω, είμαστε σε θέση να υλοποιήσουμε την δομή δεδομένων του πίνακα κατακερματισμού ανοικτής διεύθυνσης. Πρόκειται για μια δομή η οποία, ακολουθώντας τα ζητούμενα, υποστηρίζει όλες τις βασικές ενέργειες ενός πίνακα κατακερματισμού, εξαιρουμένης της διαγραφής στοιχείου.

$Γ_{2.1}$ Ανάλυση θεωρητικής προσέγγισης

Σε μια πιο ενδελεχή περιγραφή, η δομή δεσμεύει δυναμικά χώρο από το σωρό της μνήμης, προκειμένου να χειριστεί τα δεδομένα που ο χρήστης επιθυμεί να αποθηκεύσει. Όταν η δομή ολοκληρώσει τις ενέργειές της και χρειαστεί να «φύγει» από τη μνήμη, ο χώρος αυτός απελευθερώνεται, αποφεύγοντας διαρροή μνήμης.

Αξιοποιώντας θεωρητικό υπόβαθρο, γνωρίζουμε ότι σημαντικό ρόλο στη χρηστή λειτουργία αλλά και αποδοτικότητα ενός πίνακα κατακερματισμού διαδραματίζει η μέθοδος που χρησιμοποιεί, ώστε να διαχειρίζεται τις συγκρούσεις (collisions) που λαμβάνουν χώρα κατά την εισαγωγή των στοιχείων. Στην περίπτωση της παρούσης εργασίας, υιοθετήθηκε η μέθοδος του διπλού κατακερματισμού (double hashing).

Σύμφωνα με αυτή, η τιμή που πρόκειται να εισαχθεί (έστω key) περνά ως όρισμα σε μια συνάρτηση κατακερματισμού (έστω h_1(key)), από την οποία προκύπτει η θέση του πίνακα στην οποία προβλέπεται να εισαχθεί το key. Σε περίπτωση που η θέση αυτή είναι κατειλημμένη, συμβαίνει «σύγκρουση» και θα χρειαστεί περεταίρω διερεύνηση (probing) προκειμένου να βρεθεί ελεύθερη θέση που θα δεχθεί το key.

Η διερεύνηση αυτή, βέβαια, θα πρέπει να διενεργηθεί με συστηματικό τρόπο. Στην περίπτωση της μεθόδου του διπλού κατακερματισμού, ο συστηματικός αυτός τρόπος λαμβάνει τη μορφή μιας δεύτερης συνάρτησης κατακερματισμού (έστω h_2(key)). Τελικά, προκύπτει ότι η θέση που θα δεχθεί το key ορίζεται από τον τύπο:

$$index = (h_1(key) + \mathrm{i} \cdot h_2(key)) \ \% \ array\_size, \quad \mathrm{i} = 0,1,2,3, \dots$$

όπου array_size το μέγεθος του πίνακα κατακερματισμού και i η τιμή ενός πολλαπλασιαστή που ξεκινά από το 0 και αυξάνεται κατά ένα σε κάθε σύγκρουση, έως ότου βρεθεί ελεύθερη θέση που δύναται να φιλοξενήσει το key. Η πράξη της εύρεσης του υπολοίπου της ακέραιας διαίρεσης του αθροίσματος των συναρτήσεων με το μέγεθος του πίνακα είναι αυτή που εξασφαλίζει πως η τιμή που λαμβάνει το index είναι εντός των ορίων του πίνακα, δηλαδή στο εύρος [0, array_size-1].

Ωστόσο, ο καθορισμός της μεθόδου διαχείρισης των συγκρούσεων δεν είναι αρκετός για τον πλήρη προσδιορισμό της δομής του πίνακα κατακερματισμού. Ιδιαίτερο ενδιαφέρον για τους γράφοντες παρουσίασαν τα πιθανά ζητήματα που προκύπτουν από την επιλογή της μεθόδου του διπλού κατακερματισμού και ιδιαίτερα οι τρόποι θωράκισης του προγράμματος από αυτά.

Ειδικότερα, βασικό πρόβλημα που είναι πιθανό να προκύψει από τη χρήση διπλού κατακερματισμού είναι η ατέρμονη εκτέλεση των βημάτων εισαγωγής ενός κλειδιού. Υποθέτοντας ότι κατά την εισαγωγή ενός στοιχείου συμβαίνουν συγκρούσεις, ο πολλαπλασιαστής i αυξάνεται διαδοχικά παράγοντας τιμές για το index. Η χρήση της πράξης mod κανονικοποιεί τις τιμές αυτές στο εύρος του πίνακα και εκεί ενδέχεται να παραβιαστεί το κριτήριο της περατότητας του αλγορίθμου εισαγωγής. Με απλά λόγια, αν οι τιμές που παράγονται είναι μια πεπερασμένη ακολουθία αριθμών και όλες οι θέσεις της ακολουθίας είναι κατειλημμένες, τότε δημιουργείται κύκλος και ο αλγόριθμος της εισαγωγής εκτελείται επ' άπειρον. Ελέγχονται, δηλαδή, συνεχώς οι ίδιες (κατειλημμένες) θέσεις και ο αλγόριθμος δεν μπορεί να προσεγγίσει άλλες, τυχόν ελεύθερες!

Έχοντας αναγνωρίσει το πρόβλημα, η αναζήτηση μεθόδων αντιμετώπισής του προκύπτει ως λογικό επόμενο της σκέψης μας. Αξιοποιώντας σχετική, έντυπη και ψηφιακή βιβλιογραφία, υιοθετήθηκαν δύο μέθοδοι θωράκισης

του προγράμματος από τον εγκλωβισμό σε έναν τέτοιο ατέρμων βρόγχο διερεύνησης θέσεων. Οι μέθοδοι αυτές είναι οι ακόλουθες:

1. ***Ορισμός του μεγέθους του πίνακα ως ένας αρκούντως μεγάλος (συναρτήσει του πλήθους των στοιχείων προς εισαγωγή) πρώτος αριθμός***

   Για την επίτευξη αυτού γίνεται χρήση δυαδικού αρχείο δεδομένων (.dat) στο οποίο υπάρχουν καταγεγραμμένοι πρώτοι αριθμοί από το 2 έως και το 15’485’857. Οι αριθμοί αυτοί έχουν καταγραφεί και η ορθότητά τους έχει ελεγχθεί από τους γράφοντες σε χρόνο προγενέστερο της εκτέλεσης του προγράμματος από τον τελικό χρήστη. Ο έλεγχος της ορθότητας έγινε με χρήση αλγορίθμου που προσομοιώνει την κατασκευή «Κοσκίνου του Ερατοσθένη». Πηγή του αρχείου των πρώτων αριθμών είναι το Πανεπιστήμιο του Τενεσί των ΗΠΑ (University of Tennessee at Martin – UTM). Το αρχείο αυτό είναι αποτέλεσμα επιστημονικού και ερευνητικού έργου του Ιδρύματος αυτού. Η λήψη του πραγματοποιήθηκε από τον επίσημο διαδικτυακό τόπο του Ιδρύματος και συγκεκριμένα από την καταχώριση για «μικρούς» πρώτους αριθμούς. Πρόκειται για το αρχείο “Primes up to 15'485'863.dat” των παραδοτέων αρχείων της εργασίας. Το αρχείο αυτό απαιτείται να βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο αρχείο του προγράμματος, προκειμένου το τελευταίο να λειτουργήσει σωστά.

   Προς χάριν ακρίβειας, το αρχείο πηγής βρίσκεται σε μορφή απλού κειμένου (.txt) και όχι δυαδικό με επέκταση .dat, όπως το παραδοτέο. Αυτό συμβαίνει καθώς, σε χρόνο προγενέστερο της εκτέλεσης του προγράμματος από τον τελικό χρήστη, οι γράφοντες φρόντισαν να μετατραπεί το αρχείο αυτό σε δυαδικό, λαμβάνοντας, επίσης, την επέκταση .dat, η οποία υποδηλώνει πως πρόκειται για σημαντικά δεδομένα που αξιοποιεί το πρόγραμμα κατά τη λειτουργία του. Οι κυριότεροι λόγοι για τους οποίους οι γράφοντες αποφάσισαν τη μετατροπή του αρχείου αυτού δεδομένων σε δυαδικό είναι δύο.

   Αφενός, η αποθήκευση του αρχείου ως δυαδικό παρέχει τη δυνατότητα τα δεδομένα να αποθηκευτούν σε τύπο και μορφή άμεσα αξιοποιήσιμη από τον υπολογιστή. Αντιθέτως, στο αρχείο απλού κειμένου οι αριθμοί θα έπρεπε να αποθηκευτούν και να διαβαστούν, αρχικά, ως συμβολοσειρές και, κατόπιν, να μετατραπούν σε αριθμητικές τιμές, προκειμένου να αξιοποιηθούν από το πρόγραμμα.

   Αφετέρου, με τη χρήση δυαδικού αρχείου δεδομένων το πρόγραμμα παρουσιάζει μεγαλύτερο βαθμό θωράκισης από τυχόν ακούσια ή εκούσια δράση του χρήστη που μπορεί να επηρεάσει το αρχείο δεδομένων, με απρόβλεπτα αποτελέσματα. Ένα δυαδικό αρχείο δεν είναι τόσο εύκολα επεξεργάσιμο, συγκριτικά με ένα αρχείο απλού κειμένου, το οποίο ο χρήστης μπορεί να ανοίξει, να προσπελάσει σε κατανοητή για τον ίδιο μορφή και να προβεί σε οποιαδήποτε αλλαγή επί του περιεχομένου του, κάτι που προφανώς δεν είναι ενδεικτικό καλής προγραμματιστικής πρακτικής.

2. ***Χρήση κατάλληλων συναρτήσεων κατακερματισμού h_1 και h_2, οι οποίες περιλαμβάνουν πρώτους αριθμούς ως πολλαπλασιαστές στον τύπο τους***

   Οι συναρτήσεις κατακερματισμού κατέχουν κομβικής σημασίας ρόλο για την αποδοτικότητα της δομής, μιας και καθορίζουν την κατανομή των στοιχείων προς εισαγωγή στις θέσεις του πίνακα. Το ιδανικό σενάριο είναι οι συναρτήσεις αυτές να οδηγούν σε κανονική κατανομή των κλειδιών. Κατόπιν μελέτης σχετικής βιβλιογραφίας, επιλέχθηκαν οι εξής συναρτήσεις κατακερματισμού:

$$h_1(key) = (37^n \cdot c_1 \ + \ 37^{n-1} \cdot c_2 \ + \ \dots \ + \ 37 \cdot c_{n-1} \ + \ c_n) \ \% \ array\_size$$

$$h_2(key) = (13^n \cdot c_1 \ + \ 13^{n-1} \cdot c_2 \ + \ \dots \ + \ 13 \cdot c_{n-1} \ + \ c_n) \ \% \ array\_size$$

   όπου n το μήκος της λέξης προς εισαγωγή, C_0 ο πρώτος και C_n ο τελευταίος χαρακτήρας αυτής (από αριστερά προς τα δεξιά). Οι συναρτήσεις αυτές τοποθετούνται στον τύπο του διπλού κατακερματισμού, όπως αυτός αναλύθηκε παραπάνω, δίνοντας τελικά τη θέση του πίνακα στην οποία προβλέπεται να εισαχθεί το key.

Χωρίς να είμαστε σε θέση να τεκμηριώσουμε με μαθηματική απόδειξη την εγκυρότητα των παραπάνω μεθόδων, η εμπειρική προσέγγιση που ακολουθήθηκε δεν κατέρριψε τον ισχυρισμό ότι τα βήματα αυτά θωρακίζουν το

πρόγραμμα από επ' άπειρον επανάληψη εντολών. Πιο συγκεκριμένα, οι δοκιμές του κώδικα με πολλά διαφορετικά πλήθη λέξεων προς εισαγωγή (από 10 λέξεις έως και 5.5 εκατομμύρια λέξεις με πυκνές δειγματοληψίες στις ενδιάμεσες τιμές) δεν παρουσίασαν σε κάποια από τις εκτελέσεις ατέρμων βρόγχο.

Γ2.2 Παρουσίαση μεθόδων

Έχοντας αναλύσει την γενικότερη θεωρητική προσέγγιση της υλοποίησης του πίνακα κατακερματισμού ανοικτής διεύθυνσης, είμαστε σε θέση να παρουσιάσουμε και να επεξηγήσουμε τις λειτουργίες που η δομή υποστηρίζει. Οι λειτουργίες αυτές λαμβάνουν τη μορφή δημόσιων μεθόδων, οι οποίες είναι οι ακόλουθες:

[1] **openAddressHashTable(unsigned long int preferredSize)**

κατασκευαστής κλάσης που δέχεται ως όρισμα το επιθυμητό μέγεθος για τον πίνακα κατακερματισμού που πρόκειται να δημιουργηθεί. Σύμφωνα με όσα αναλύθηκαν, το μέγεθος του πίνακα θα πρέπει να είναι πρώτος αριθμός. Για το λόγο αυτό, ο κατασκευαστής δημιουργεί έναν καινούργιο πίνακα κατακερματισμού με μέγεθος, όχι απαραίτητα ίσο με το επιθυμητό, αλλά ίσο με τον πλησιέστερο προς τα επάνω πρώτο αριθμό. Για την εύρεση του κατάλληλου μεγέθους αξιοποιείται η (εξωτερική της κλάσης) συνάρτηση [6]. Για παράδειγμα, αν ο κατασκευαστής κληθεί με όρισμα το 9 (που δεν είναι πρώτος) θα δημιουργήσει έναν καινούργιο πίνακα κατακερματισμού 11 θέσεων, καθώς το 11 είναι ο πλησιέστερος προς τα επάνω πρώτος του 9. Η μνήμη δεσμεύεται δυναμικά.

[2] **~ openAddressHashTable()** : *τυπικός καταστροφέας κλάσης, αποδεσμεύει το χώρο που δεσμεύτηκε δυναμικά*

[3] **insert(const string &givenWord, unsigned long int givenFrequency = 1)**

Μέθοδος που υλοποιεί την εισαγωγή μιας λέξης στον πίνακα κατακερματισμού. Η εισαγωγή γίνεται με βάση τις συναρτήσεις κατακερματισμού h1 και h2 που αναλύθηκαν [5]. Σε περίπτωση που η λέξη υπάρχει ήδη στον πίνακα κατακερματισμού, η συχνότητά της αυξάνεται κατά ένα. Το default όρισμα επιτρέπει την κλήση της συνάρτησης για την εισαγωγή μιας λέξης αν είναι γνωστό ότι έχει ήδη μια συχνότητα εμφάνισης (όχι αναγκαστικά ίση με 1). Το default αυτό όρισμα υλοποιήθηκε, κυρίως, για λόγους πληρότητας. Δεν χρησιμοποιείται στο παρόν πρόγραμμα.

[4] **search (const string &givenWord)**

Μέθοδος που υλοποιεί την αναζήτηση μιας λέξης στον πίνακα κατακερματισμού. Η αναζήτηση γίνεται με βάση τις συναρτήσεις κατακερματισμού h1 και h2 που αναλύθηκαν [5]. Η συνάρτηση αποφαίνεται αν η λέξη που αναζητείται βρίσκεται στον πίνακα ή όχι. Σε περίπτωση που η λέξη βρεθεί, επιστρέφεται το πλήθος των εμφανίσεων (συχνότητα αυτής). Σε διαφορετική περίπτωση επιστρέφεται η τιμή 0.

[5] **hashFunction1(const string &givenWord)** και **hashFunction2(const string &givenWord)**

Οι συναρτήσεις κατακερματισμού h1 και h2 όπως παρουσιάστηκαν στην υπο-ενότητα της θεωρητικής προσέγγισης. Δέχονται ως όρισμα μια συμβολοσειρά και επιστρέφουν την τιμή κατακερματισμού, η οποία με τη σειρά της εισάγεται στον τύπο της τελικής θέσης εισαγωγής.

Συμπληρωματικά με τις μεθόδους της κλάσης, υπάρχει και μια εξωτερική συνάρτηση που αξιοποιείται στον κατασκευαστή της κλάσης. Η συνάρτηση αυτή είναι η εξής:

[6] **findMinPrimeGreaterThan(const unsigned long int &givenNumber)**

Είναι η συνάρτηση που εξασφαλίζει πως το μέγεθος του πίνακα κατακερματισμού είναι πρώτος αριθμός. Δέχεται ως όρισμα έναν θετικό ακέραιο αριθμό και επιστρέφει τον πλησιέστερο προς τα πάνω πρώτο αριθμό αυτού. Για την εύρεση του αριθμού αυτού αξιοποιείται ένα δυαδικό αρχείο που περιέχει πρώτους αριθμούς, όπως αυτό αναλύθηκε παραπάνω. Για παράδειγμα, αν δεχθεί ως όρισμα τον αριθμό 14, τότε θα επιστρέψει τον πλησιέστερο προς τα επάνω πρώτο αριθμό, δηλαδή τον αριθμό 17.

Ενότητα Δ': Χειρισμός και σύγκριση αποδοτικότητας δομών

Παραδοτέα αρχεία που εμπλέκονται: ***main.cpp*** *&* ***DataStructuresHandling.h***

Έχοντας παρουσιάσει τον τρόπο λειτουργίας και των τριών δομών που υλοποιήθηκαν στο πλαίσιο της εργασίας, είμαστε σε θέση να μελετήσουμε τον τρόπο με τον οποίο το πρόγραμμα χειρίζεται (ενδεικτικά) τις δομές αυτές. Ο χειρισμός αυτός λαμβάνει τη μορφή ερωτημάτων επί των δομών (με βάση τις λέξεις του συνόλου Q) και έχει ως στόχο τη σύγκριση της αποδοτικότητας των δομών μεταξύ τους. Ακολούθως παρουσιάζεται το τμήμα του προγράμματος που είναι υπεύθυνο για αυτές τις ενέργειες.

$Δ_1$. Αξιοποίηση του συνόλου Q – Υποβολή ερωτημάτων επί των δομών

Ως απόρροια του πρώτου τμήματος του προγράμματος (βλ. ενότητα Α΄ της παρούσης αναφοράς), στον ίδιο φάκελο με το εκτελέσιμο αρχείο του προγράμματος έχει δημιουργηθεί ένα ενδιάμεσο, βοηθητικό αρχείο το οποίο περιέχει καταγεγραμμένες τις λέξεις του συνόλου Q. Το αρχείο αυτό, μάλιστα, έχει ευνοϊκή γραμμογράφηση, καθώς περιέχει όλες τις λέξεις, μία ανά γραμμή. Το γεγονός αυτό καθιστά ιδιαίτερα απλό και γρήγορο να αξιοποιηθούν οι λέξεις του συνόλου Q και να μετατραπούν σε αντικείμενα ερωτήσεων (queries) προς τις δομές.

Η ενέργειες αυτές υλοποιούνται μέσω πρότυπης συνάρτησης που αξιοποιείται από τη συνάρτηση main του προγράμματος, ενώ τα ερωτήματα λαμβάνουν τη μορφή αναζητήσεων των λέξεων του Q επί των δομών. Η επιλογή της πράξης της αναζήτησης υπαγορεύθηκε από το γεγονός πως είναι κοινή μεταξύ των τριών δομών, βοηθώντας να εξαχθούν πιο ασφαλή συμπεράσματα. Αντί αυτής, πάντως, θα μπορούσε να έχει επιλεγεί οποιαδήποτε άλλη πράξη υποστηρίζουν οι δομές, χωρίς να υπάρξει αλλαγή στην προσέγγιση του προβλήματος.

Σε επίπεδο τεχνικής υλοποίησης, η χρήση πρότυπης συνάρτησης (με όνομα writeProgrammeReport) που μπορεί να διαχειριστεί αναζητήσεις σε αντικείμενα και των τριών δομών, επιτρέπει στη συνάρτηση main του προγράμματος να διαχειρίζεται με ενιαίο και ομοιόμορφο τρόπο τη διαδικασία των ερωτημάτων. Ειδικότερα, σε κάθε εκτέλεση της πρότυπης συνάρτησης, γίνεται ανάγνωση των λέξεων του Q από το βοηθητικό αρχείο και αναζήτηση αυτών στο αντικείμενο της δομής που έχει περαστεί ως όρισμα στην πρότυπη συνάρτηση. Η συνάρτηση αυτή καλείται τρεις φορές, διαδοχικά από τη main του προγράμματος, κάθε φορά με το αντίστοιχο αντικείμενο της κάθε δομής. Στο εσωτερικό της συνάρτησης παράγεται η γραπτή αναφορά της κάθε δομής, με τη μορφή αρχείου απλού κειμένου (.txt) με χαρακτηριστικό όνομα. Η συνάρτηση επιστρέφει το χρόνο σε δευτερόλεπτα που διήρκεσε η εκτέλεσή της, δηλαδή η διενέργεια των αναζητήσεων στην εκάστοτε δομή.

Δ2. Χρονομέτρηση απόκρισης και σύγκριση των δομών

Η υπο-ενότητα αυτή θα μπορούσε να χαρακτηριστεί ως το διακύβευμα όλων όσων αναλύθηκαν έως τώρα. Μια από τις κεντρικές ιδέες της εργασίας είναι η σύγκριση της αποδοτικότητας των τριών δομών μεταξύ τους. Η σύγκριση αυτή λαμβάνει χώρα σε αυτό ακριβώς το τμήμα του προγράμματος αξιοποιώντας πληροφορίες που έχουν συλλεχθεί από την εκτέλεση προηγούμενων τμημάτων.

Ως μέτρο σύγκρισης εκλαμβάνεται το χρονικό διάστημα που κάθε μία από τις δομές έχει να επιδείξει στην αντιμετώπιση των ερωτημάτων του συνόλου Q. Γίνεται, επομένως, σύγκριση των σχετικών χρόνων απόκρισης των δομών. Οι χρόνοι αυτοί είναι σχετικοί, καθώς μπορούν να διαφέρουν από τη μια εκτέλεση σε άλλη ή από τη υπολογιστική μηχανή σε άλλη. Ένα άλλο στοιχείο αποτελεί το γεγονός πως παράλληλα με την εκτέλεση του προγράμματος (εφαρμογή κονσόλας) η υπολογιστική μηχανή που χρησιμοποιείται εκτελεί πολλές ενέργειες μέσω του λειτουργικού συστήματος και επομένως ο χρόνος που υπολογίζεται δεν μπορεί να είναι πάντοτε αντιπροσωπευτικός της απόκρισης στα ερωτήματα και αποκλειστικά αυτής.

Λαμβάνοντας υπ’ όψιν αυτές τις παραμέτρους, στο πλαίσιο της παρούσης εργασίας θεωρήθηκε πως η σύγκριση της αποδοτικότητας των δομών δύναται να βασιστεί όχι στους χρόνους απόκρισης, αλλά στην απόλυτη διαφορά αυτών μεταξύ τους. Με άλλα λόγια, η απόλυτη διαφορά των χρόνων απόκρισης, στα μέτρα πάντα της θεώρησής μας, θεωρείται πως υπερκερνά τις ενστάσεις που προκύπτουν κατά τη χρονομέτρηση των αναζητήσεων και αποτυπώνουν σε απόλυτους χρόνους τη διαφορά των δομών μεταξύ τους.

Σε κάθε περίπτωση, τα συμπεράσματα αυτά είναι γενικής φύσης και παράγονται μέσω μιας εμπειρικής προσέγγισης, που θεωρείται επαρκώς αξιόπιστη στα πλαίσια της παρούσης εργασίας. Οι χρόνοι καταγράφονται σε ένα αρχείο απλού κειμένου (.txt) το οποίο αναπαριστά τη συνολική αναφορά του προγράμματος. Στην αναφορά αυτή εμπλέκονται δύο συναρτήσεις, οι οποίες, ωστόσο, συμβάλλουν αποκλειστικά στο οπτικό αποτέλεσμα του αρχείου αναφοράς και δεν εμφανίζουν κάποια άλλη λειτουργικότητα αναφορικά με την γενικότερη λειτουργία του προγράμματος.

Χρήσιμες Παρατηρήσεις

Λαμβάνοντας υπ' όψιν τις αξιώσεις της εκφώνησης, το πρόγραμμα δεν εμφανίζει κάτι στην οθόνη, αλλά παράγει ως αναφορές κάποια αρχεία απλού κειμένου, αναλόγως με τις λειτουργίες που επενεργούνται. Σε όλες αυτές τις περιπτώσεις που το πρόγραμμα παράγει κάποια γραπτή αναφορά, έχει οριστεί, σε χρόνο προγενέστερο της χρήσης του προγράμματος, ως προεπιλογή ένα χαρακτηριστικό όνομα αρχείου. Τα προκαθορισμένα αυτά ονόματα βρίσκονται στο αρχείο επικεφαλίδας "DefaultDefinitions.h" και δύνανται να τροποποιηθούν από έμπειρο και εξοικειωμένο χρήστη. Ο λόγος ύπαρξης του αρχείου είναι καθαρά ζήτημα καλής προγραμματιστικής πρακτικής. Πιο συγκεκριμένα, σε περίπτωση που κάποιο από αυτά τα ονόματα χρειαστεί να μεταβληθεί, αυτό μπορεί να γίνει εύκολα και γρήγορα από το εν λόγω αρχείο, χωρίς να χρειαστεί κάποιου είδους παρέμβαση στον κώδικα τον κλάσεων ή της main.

Το μόνο αρχείο για το οποίο παρέχεται η δυνατότητα να οριστεί απευθείας από το χρήστη είναι το αρχείο εισόδου. Στην περίπτωση αυτή, θα χρειαστεί ο χρήστης να περάσει (πριν την έναρξη εκτέλεσης του προγράμματος) ως ορίσματα μέσω του λειτουργικού συστήματος το όνομα του αρχείου εισόδου και το πλήθος της σύστασης του συνόλου Q. Σε περίπτωση που δεν δοθούν ορίσματα μέσω του λειτουργικού συστήματος (πιο συγκεκριμένα μέσω της γραμμής εντολών cmd), τότε ως αρχείου εισόδου θεωρείται (από προεπιλογή) το "input.txt" και το σύνολο Q ταυτίζεται με το σύνολο των λέξεων του αρχικού κειμένου.

Τέλος, όπως αναλύθηκε, κατά τη διάρκεια εκτέλεσης του προγράμματος παράγονται δύο βοηθητικά αρχεία με ευνοϊκή γραμμογράφηση για διευκόλυνση των λειτουργιών του προγράμματος. Τα αρχεία αυτά περιέχουν τις συνολικές λέξεις του αρχείου εισόδου και τις λέξεις του συνόλου Q και αξιοποιούνται στα κατάλληλα τμήματα του προγράμματος. Μετά το πέρας της εκτέλεσης των τμημάτων αυτών, τα αρχεία αυτά δεν παρουσιάζουν κάποια χρηστικότητα (ούτε ενδιαφέρουν τον τελικό χρήστη) και επομένως αφαιρούνται από τον φάκελο του εκτελέσιμου προγράμματος.

Εν κατακλείδι, η εκτέλεση του προγράμματος δεν παράγει κάποια έξοδο στην οθόνη. Μοναδική απαίτηση από το χρήστη είναι να φροντίσει ώστε να υπάρχουν στον ίδιο φάκελο με το εκτελέσιμο αρχείο το αρχείο εισόδου και το προεπιλεγμένο αρχείο δεδομένων, που περιέχει σε γλώσσα μηχανής πρώτους αριθμούς. Μετά το πέρας της εκτέλεσης, ο φάκελος θα έχει εμπλουτιστεί με αρχεία αναφορών τόσο για την κάθε δομή ξεχωριστά όσο και για το πρόγραμμα συνολικά.

Επίλογος - Δυνητικές Βελτιώσεις

Έχοντας αναλύσει διεξοδικά τα επιμέρους τμήματα του προγράμματος αλλά και τη συνολική λειτουργικότητα αυτού, η τεχνική έκθεση της παρούσης εργασίας βαίνει προς τη νοηματική της ολοκλήρωση. Κρίνοντας εκ του αποτελέσματος, η ενασχόληση με την παρούσα εργασία και η τριβή με το θεωρητικό αλλά και πρακτικό πλαίσιο των διαφόρων δομών που μελετήθηκαν πρόσφερε στους γράφοντες σημαντικές γνώσεις και συνέβαλλε στην βελτίωση των προγραμματιστικών τους δεξιοτήτων.

Βασική επιδίωξη των τελευταίων αποτέλεσε, καθ' όλη τη διάρκεια ενασχόλησης με την εργασία, η παραγωγή πηγαίου κώδικα με όσο το δυνατόν βέλτιστη λειτουργικότητα, κυρίως σε επίπεδο υλοποίησης των διάφορων ενεργειών των δομών και κατ' επέκταση του προγράμματος. Η παραδοχή αυτή οδήγησε αρκετές φορές τους γράφοντες σε αναθεώρηση των μεθόδων προσέγγισης ή υλοποίησης επιμέρους ενεργειών, στο βωμό της παρουσίασης ποιοτικότερου κώδικα.

Στις βάσεις αυτές, επομένως, δημιουργήθηκε η πλήρως λειτουργική, παρούσα εκδοχή του προγράμματος, όπως παρουσιάστηκε στις σελίδες που προηγήθηκαν. Αυτό, βέβαια, δεν σημαίνει πως το πρόγραμμα έχει εξαντλήσει τα περιθώρια βελτιστοποίησής του. Προς χάριν της αναγνώρισης αυτής, ακολούθως παρουσιάζονται ενδεικτικά ορισμένες ιδέες, σκέψεις ή προτάσεις, οι οποίες εκφράζουν τους γράφοντες και στηρίζονται στο πλαίσιο της διαρκούς προσπάθειας για βελτίωση. Οι προτάσεις αυτές θα μπορούσαν να παρουσιαστούν συνοπτικά, ως εξής:

- Εμπλουτισμός της φάσης του χειρισμού του αρχείου εισόδου με διεργασία εύρεσης των μοναδικών λέξεων του κειμένου. Η διεργασία αυτή θα μπορούσε να παρέχει χρήσιμες πληροφορίες για επόμενες λειτουργίες, κυρίως στον προσδιορισμό του χώρου μνήμης που απαιτείται για την αποθήκευση της πληροφορίας του κειμένου.
- Αλλαγή της κωδικοποίησης των ενδιάμεσων αρχείων λέξεων από απλού κειμένου (.txt) σε δυαδικά. Με τον τρόπο αυτό, το πρόγραμμα προστατεύεται ακόμη περισσότερο από τυχόν κακόβουλο χρήστη, ο οποίος μπορεί να παρέμβει σε αυτά στο χρονικό διάστημα που μεσολαβεί από τη δημιουργία μέχρι την αφαίρεσή τους από τον φάκελο του εκτελέσιμου αρχείου. Ο χρόνος αυτός, βέβαια, που έχει στη διάθεσή του ο τελευταίος δεν ξεπερνά, στη μέση εκτέλεση, τα το μισό δευτερόλεπτο.
- Προσθήκη ερευνητικής μελέτης, με στόχο τη βολιδοσκόπηση της διάρθρωση των κειμένων της αγγλικής γλώσσας. Η μελέτη αυτή θα μπορούσε να περιλαμβάνει την εξέταση μεγάλου αριθμού κειμένων της αγγλικής γλώσσας, με στόχο την σκιαγράφηση ενός μοντέλου σχετικά με το πλήθος των μοναδικών λέξεων ενός κειμένου, δεδομένων των συνολικών λέξεων αυτού. Μια ποιοτική μοντελοποίηση θα μπορούσε να εφαρμοστεί με τεχνικές της μηχανικής μάθησης, παρέχοντας χρήσιμες υποθέσεις-προβλέψεις για το χώρο που θα χρειαστεί η αποθήκευση της πληροφορίας του αρχείου εισόδου, αν είναι γνωστό το πλήθος των λέξεων που περιέχονται σε αυτό.
- Προσθήκη παράγοντα πληρότητας (load factor) ως ιδιότητα του πίνακα κατακερματισμού. Με τον τρόπο αυτό θα μπορούσε να βελτιστοποιηθεί η δέσμευση χώρου για τον πίνακα δυναμικά. Μια τέτοια προσθήκη, βέβαια, θα προϋπέθετε και προσθήκη λειτουργίας αυξομείωσης του μεγέθους του πίνακα και κατόπιν εκ νέου κατακερματισμό των υπαρχουσών τιμών από τον παλιό στον καινούργιο πίνακα.

Εν κατακλείδι, όλα τα παραπάνω αποτελούν απλώς ενδεικτικές προσεγγίσεις τις οποίες οι γράφοντες διαμόρφωσαν ως απόρροια της ενασχόλησής τους με την εκπόνηση της εργασίας και θέλησαν να μοιραστούν στην παρούσα αναφορά. Σε κάθε περίπτωση, η παραδοτέα έκδοση της εργασίας αποτελεί μια πλήρως λειτουργική πρόταση επίλυσης του προβλήματος της εργασίας, στα πλαίσια του μαθήματος των Δομών Δεδομένων.