**Τμήμα Μηχανικών Η/Υ & Πληροφορικής**

**Πανεπιστήμιο**

**PROJECT:**

**DATA STRACTURES 2023**

**`**

**2. PART B**

Για το Part II της εργασίας υλοποιήθηκε κώδικας στον οποίο, με βάση την επιλογή του χρήστη, τα δεδομένα του αρχείου “dataset effects-of-covid-19-on-trade-at-15-december-2021-provisional.csv” φορτώνονται σε ένα δένδρο δυαδικής αναζήτησης (BST) ή σε ένα πίνακα κατακερματισμού με αλυσίδες. Κάθε κόμβος του δέντρου και του πίνακα κατακερματισμού είναι μια εγγραφή (struct), της οποίας τα μέλη είναι οι στήλες του csv αρχείου (Direction Year Date Weekday Country Commodity Transport\_Mode Measure Value Cumulative).

**A/Β. BST Δέντρο**

To **BST δέντρο** που επιλέχθηκε για την υλοποίηση αυτής της εργασίας είναι ένα AVL δένδρο, το οποίο ονομάστηκε έτσι από τα ονόματα των Adelson, Velskii και Landis.

Είναι ένα δυαδικό δέντρο στο οποίο σε κάθε κόμβο, το ύψος του αριστερού και του δεξιού υπο-δέντρου διαφέρει το πολύ κατά ένα. Μετά την εισαγωγή κάθε νέου κόμβου διατρέχεται το μονοπάτι του νέου κόμβου μέχρι την ρίζα αντίστροφα και γίνονται οι απαραίτητες διορθώσεις μέσω περιστροφών.

A picture containing black, darkness

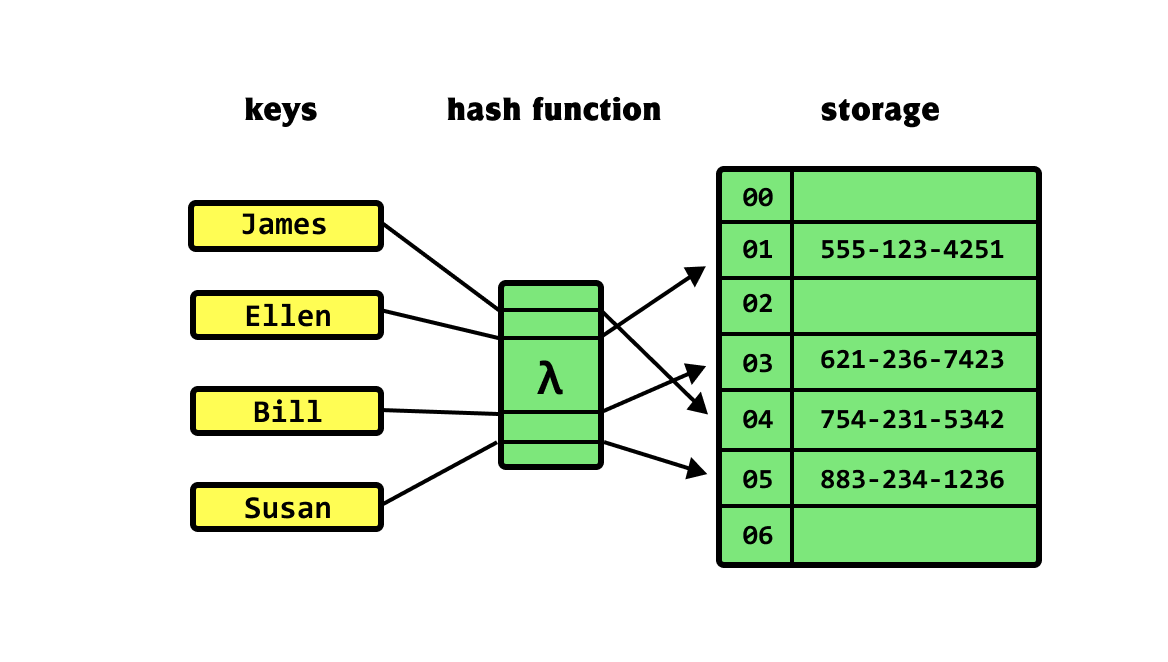
Description automatically generated

(παράδειγμα binary search tree)

Στην περίπτωση φόρτωσης με BST, ο χρήστης μπορεί να επιλέξει αν η φόρτωση (και αντίστοιχα η ταξινόμηση) θα γίνει με βάση την ΗΜΕΡΟΜΗΝΙΑ ή την τιμή VALUE, που αποτελούν μέλη κάθε εγγραφής.

**B. Hashing (Κατακερματισμός)**

Στον **πίνακα κατακερματισμού** (hashing) η ταξινόμηση γίνεται με βάση το αποτέλεσμα της συνάρτησης κατακερματισμού (hash function). Kάθε στοιχείο έχει ένα κλειδί το οποίο δίνεται από την συνάρτηση κατακερματισμού. Η συνάρτηση κατακερματισμού υπολογίζεται ως το υπόλοιπο (modulo) της διαίρεσης του αθροίσματος των κωδικών ASCII των επιμέρους χαρακτήρων που απαρτίζουν την ΗΜΕΡΟΜΗΝΙΑ με ένα περιττό αριθμό m=11 που συμβολίζει το πλήθος των κάδων (buckets). Ο πίνακας κατακερματισμού αποτελείται από 11 θέσεις, όσοι και οι κάδοι (buckets). Η συνάρτηση κατακερματισμού μπορεί να οδηγήσει σε συγκρούσεις που σημαίνει ότι δύο ή περισσότερα κλειδιά αντιστοιχίζονται στην ίδια τιμή. Για να αποφευχθούν οι συγκρούσεις αυτές χρησιμοποιείται η συνάρτηση κατακερματισμού με αλυσίδες. Η ιδέα είναι να κάνουμε κάθε κελί του πίνακα κατακερματισμού να οδηγεί σε μια συνδεδεμένη λίστα εγγραφών που έχουν την ίδια τιμή συνάρτησης κατακερματισμού.



(παράδειγμα διαδικασίας hashing)

**Επιλογές Χρήστη:**

Αφού ο χρήστης επιλέξει τον τρόπο που θα γίνει η φόρτωση, στην οθόνη του θα εμφανίζεται ένα μενού.

* Για την **πρώτη επιλογή**: Φόρτωση του αρχείου σε ένα BST με βάση την ΗΜΕΡΟΜΗΝΙΑ, θα είναι το εξής:

**1.** **Απεικόνιση** του BST με ενδο-διατεταγμένη διάσχιση. Κάθε απεικόνιση θα πρέπει να περιέχει μια επικεφαλίδα με τους τίτλους των στοιχείων των εγγραφών που απεικονίζονται.

**2.** **Αναζήτηση** της τιμής Value βάσει ΗΜΕΡΟΜΗΝΙΑΣ (Date) που θα δίνεται από το χρήστη.

**3.** **Τροποποίηση** του περιεχομένου του πεδίου value που αντιστοιχεί σε συγκεκριμένη

ΗΜΕΡΟΜΗΝΙΑ (Date).

**4.** **Διαγραφή** μιας εγγραφής που αντιστοιχεί σε συγκεκριμένη ΗΜΕΡΟΜΗΝΙΑ (Date).

**5.** **Έξοδος** από την εφαρμογή.

* Για την **δεύτερη επιλογή** με χρήση πίνακα κατακερματισμού: Φόρτωση του αρχείου σε μία δομή Hashing με αλυσίδες, θα είναι το εξής:

**1.** **Εύρεση** ΗΜΕΡΑΣ/ΗΜΕΡΩΝ με την ΕΛΑΧΙΣΤΗ ΤΙΜΗ Value.

**2.** **Εύρεση** ΗΜΕΡΑΣ/ΗΜΕΡΩΝ με τη ΜΕΓΙΣΤΗ ΤΙΜΗ Value

* Για την **τρίτη επιλογή**: Φόρτωση του αρχείου σε ένα BST με βάση την τιμή Value, θα είναι το εξής:

**1.** **Αναζήτηση** Τιμής Value βάσει της ΗΜΕΡΟΜΗΝΙΑΣ που θα δίνεται από το χρήστη.

**2.** **Τροποποίηση** των στοιχείων εγγραφής βάσει ΗΜΕΡΟΜΗΝΙΑΣ που θα δίνεται από το

χρήστη. Η τροποποίηση προφανώς αφορά ΜΟΝΟ το πεδίο Value.

**3.** **Διαγραφή** μιας εγγραφής από τον πίνακα κατακερματισμού βάσει ΗΜΕΡΟΜΗΝΙΑΣ που

θα δίνεται από το χρήστη.

**4.** **Έξοδος** από την εφαρμογή.

**ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ**

(με αναφορές σε γραμμές του κώδικα)

\*\*Ο κώδικας περιέχει ελάχιστα σχόλια διότι οι μεταβλητές ονομάστηκαν κατάλληλα ώστε να παραπέμπουν στην λειτουργία/χρήση τους και επειδή ο κώδικας και κάθε συνάρτηση εξηγείται αναλυτικά στην αναφορά\*\*

Αρχικά δηλώνονται oι 3 βιβλιοθήκες(γραμμές 1-3), stdio.h για την εκτέλεση συναρτήσεων όπως η scanf,sscnaf,printf και sprint, stdlib.h η οποία δηλώνει διάφορες βοηθητικές συναρτήσεις για μετατροπές τύπων, δυναμική διαχείριση μνήμης, αλγόριθμους και άλλες παρόμοιες περιπτώσεις χρήσης και string.h για τον χειρισμό ακολουθιών χαρακτήρων και συναρτήσεις όπως strcpy και strmcp. Έπειτα (γραμμές 4-10) γίνονται define τιμές ώστε ο κώδικας να είναι πιο ευανάγνωστος και να γίνονται πιο εύκολα αλλαγές ή/και διορθώσεις.

Στην συνέχεια δηλώνονται οι δομές(structs). Πρώτα το **struct record** το οποίο έχει ως μέλη του τα στοιχεία των στηλών του csv αρχείου. Έπειτα το **struct AVL\_NODE** που έχει ως μέλη το **data**, που είναι της μορφής RECORD γιατί κάθε κόμβος είναι μία εγγραφή, το **AVL\_NODE \*left** και το **AVL\_NODE \*right** που είναι δείκτης στα παιδιά κάθε κόμβου( αν υπάρχουν), **struct AVL\_LIST\_NODE \*list** που είναι δείκτης στην λίστα που αποθηκεύονται οι κόμβοι που επαναλαμβάνονται και το **height** όπου αποθηκεύεται το ύψος κάθε κόμβου. Ακολούθος δηλώνεται το **typedef struct AVL\_LIST\_NODE** είναι μέλος του **struct AVL\_NODE** για τον λόγω που προαναφέρθηκε και έχει ως μέλη τα **data**, που είναι της μορφής RECORD γιατί κάθε κόμβος είναι μία εγγραφή και **struct avl\_list\_node \*next\_list\_node** που είναι δείκτης στον επόμενο κόμβο της λίστας. Μετά δηλώνεται το **struct list\_node** και έχει για μέλη το record, αφού κάθε κόμβος της λίστας αποτελείται από μια εγγραφή του csv αρχείου και το **list\_node\*** nextπου είναι δείκτης στο επόμενο στοιχείο της λίστας. Τελευταίο δηλώνεται το **struct hash\_table** του οποίου μέλος είναι το **Node\* buckets[HASH\_TABLE\_SIZE]** δηλαδή ένας πίνακας που σε κάθε θέση του έχει δείκτη σε συνδεδεμένη λίστα list\_node.

Συνεχίζουμε με την δήλωση των συνωνύμων. Αρχικά το **typedef const char element[11]**. Όπου χρησιμοποιούμε const char είναι της μορφής πίνακα 11 χαρακτήρων, οπότε για ευκολία το ονομάζουμε element[11]. Ακολούθως δηλώνονται τα **typedef struct AVL\_NODE AVL\_NODE** και **typedef struct AVL\_NODE \*AVL\_NODE\_PTR** όπου το πρώτο είναι συνώνυμο του κόμβου δένδρου και το δεύτερο συνώνυμο του δείκτη κόμβου. Με αυτό τον τρόπο αποφεύγουμε την επανάληψη του struct AVL\_NODE, ενώ επιπρόσθετα κάνουμε τον κώδικα πιο ευανάγνωστο.

Ακολούθως δηλώνονται οι συναρτήσεις, των οποίων η λειτουργεία εξηγείται παρακάτω, μετά την περιγραφή της main:

Στην **main** ο χρήστης ερωτάται αν θέλει τα δεδομένα να φορτωθούν σε ένα BST ή σε δομή Hashing with chaining. Στην περίπτωση που ο χρήστης επιλέξει BST, έχει την επιλογή αν τα δεδομένα θα φορτωθούν με βάση την ημερομηνία ή την τιμή value. Με βάση την επιλογή του χρήστη καλούνται οι αντίστοιχες συναρτήσεις με τα menu. Στην περίπτωση των menu του BST, ο αριθμός που πληκτρολόγησε ο χρήστης για την επιλογή του τρόπου ταξινόμησης περνάει ως όρισμα στην συνάρτηση.

**void menu\_bst\_sorted\_by\_date(int users\_sort\_choice);**

**void menu\_bst\_sorted\_by\_value(int users\_sort\_choice);**

**void menu\_hashing();**

Τα 3 menu λειτουργούν με παρόμοιο τρόπο. Αρχικά δηλώνεται το δένδρο και ο πίνακας κατακερματισμού αντίστοιχα και αρχικοποιούνται. Στη συνέχεια καλείται η load\_data\_to\_avl\_tree( load\_data\_to\_hashTable αντίστοιχα), η οποία φορτώνει τα δεδομένα του αρχείου στο δένδρο/πίνακα. Ακολούθως τα menu δίνουν στον χρήστη τις επιλογές όπως ζητούνται στις οδηγίες της άσκησης και με βάση την επιλογή του χρήστη εκτελούνται οι κατάλληλες ενέργειες ή/και καλούνται οι κατάλληλες συναρτήσεις. Και τα 3 menu έχουν την κοινή επιλογή για έξοδο η οποία τερματίζει το πρόγραμμα.

**void** **load\_data\_to\_avl\_tree(const char\* filename, AVL\_NODE\_PTR \*root, int users\_sort\_choice);**

**void load\_data\_to\_hashTable(HashTable\* table, const char\* filename);**

Επίσης και αυτές οι συναρτήσεις λειτουργούν με παρόμοιο τρόπο. Ξεκινούν με τ άνοιγμα του αρχείου και ελέγχοντας αν υπάρχει. Ακολούθως δηλώνουν την μεταβλητή char line στην οποία αποθηκεύεται κάθε νέα γραμμή του csv αρχείου. Το αρχείο csv είναι το excel σε μορφή αρχείου κειμένου. Κάθε στήλη χωρίζεται με κόμμα(«,») και κάθε γραμμή είναι μία νέα γραμμή. Εκτελούμε μία φορά την fgets ώστε να αγνοηθεί η πρώτη γραμμή που περιέχει το όνομα της κάθε στήλης. Έπειτα διαβάζουμε όλες τις εγγραφές μέχρι το τέλος του αρχείου. Σε κάθε επανάληψη δηλώνεται εκ νέου η μεταβλητή record και στα μέλη της αποθηκεύονται τα στοιχεία κάθε στήλης του αρχείου. Αφού κάθε νέα γραμμή βρίσκεται στην μεταβλητή line, εκτελούμε την sscanf η οποία παίρνει σαν ορίσματα την γραμμή line, την μορφή που είναι γραμμένο το csv αρχείο και που θέλουμε να αποθηκεύει κάθε στοιχείο. Στο τέλος κάθε επανάληψης καλείται η συνάρτηση insert\_avl\_node\_by\_date(ή insert\_avl\_node\_by\_value, ανάλογα την επιλογή ταξινόμησης που είχε κάνει ο χρήστης στην main, η οποία επιλογή (int users\_sort\_choice) είναι όρισμα της load\_data\_to\_avl\_tree) και insert\_record\_to\_hashTable αντίστοιχα. Τέλος κλείνουμε το αρχείο.

**int insert\_avl\_root(AVL\_NODE\_PTR \*root, RECORD rec);**

Παίρνει σαν ορίσματα AVL\_NODE\_PTR \*root που είναι δείκτης στην θέση που θα μπει ο νέος κόμβος στο δένδρο και την RECORD rec που είναι η εγγραφή που εισάγεται στο δένδρο. Ξεκινάει με την δημιουργία της παραμέτρου new\_node. Ελέγχει αν η ο δείκτης σε κόμβο του δένδρου δείχνει στο κενό για να συνεχίσει με τις υπόλοιπες εντολές, διαφορετικά επιστρέφει “FALSE”. Αν ο δείκτης δεν δείχνει στο κενό σημαίνει υπάρχει ήδη κάποιο στοιχείο σε αυτόν τον κόμβο. Ακολούθως δεσμεύει μνήμη για το νέο κόμβο και κάνει τον κατάλληλο έλεγχο. Αποθηκεύει στο data του νέου κόμβου την νέα εγγραφή και αρχικοποιεί τα παιδιά της στο “NULL”. Βάζει τον δείκτη να δείχνει στο νέο κόμβο και επιστρέφει “TRUE”.

**int delete\_bst\_node(AVL\_NODE\_PTR \*root, element users\_given\_date);**

Η συνάρτηση διαγράφει ένα κόμβο του. Παίρνει ορίσματα την ρίζα του δένδρου AVL\_NODE\_PTR \*root και την ημερομηνία που έδωσε ο χρήστης element users\_given\_date, για να διαγραφή ο κόμβος με την ημερομηνία αυτή. Παράμετροι της συνάρτησης είναι οι εξής:

AVL\_NODE\_PTR current, parent, next\_in\_order; Σε μορφή δείκτη σε κόμβο του δένδρου ο current που είναι o τρέχον κόμβος με τον οποίο γίνονται η συγκρίσεις για να βρούμε τον κόμβο με την ημερομηνία που έδωσε ο χρήστης, ο parent που ο πατέρας του κόμβου current και ο next\_in\_order o επόμενος στην ενδοδιατεταγμένη διάταξη.

int plevra\_ipodentrou; Παίρνει τις τιμές 1 ή 2 η οποίες αντιστοιχούν στην πλευρά του υποδένδρου που βρίσκεται ο κόμβος που ψάχνουμε.

Διασχίζει το δένδρο με την while συγκρίνοντας την ημερομηνία που έδωσε ο χρήστης και αυτή που έχει κάθε κόμβος. Σε κάθε επανάληψη ανανεώνει τις τιμές των current, parent και plevra\_ipodentrou. Αν ο κόμβος είναι κενός επιστρέφει “FALSE”. Μετά ψάχνει αν ο κόμβος έχει παιδιά. Στο πρώτο if έχουμε την περίπτωση που ο κόμβος δεν έχει ούτε δεξί, ούτε αριστερό παιδί απλώς ελευθερώνεται και ο κατάλληλος θυγατρικός δείκτης του γονέα ενημερώνεται ανάλογα. Εάν ο τρέχων κόμβος έχει ένα αριστερό παιδί αλλά όχι το δεξί παιδί, το αριστερό παιδί γίνεται το νέο παιδί του γονέα και ο τρέχων κόμβος ελευθερώνεται. Εάν ο τρέχων κόμβος έχει δεξί παιδί αλλά όχι αριστερό παιδί, το δεξί παιδί γίνεται το νέο παιδί του γονέα και ο τρέχων κόμβος ελευθερώνεται. Εάν ο τρέχων κόμβος έχει και αριστερό και δεξί παιδί, ο κώδικας βρίσκει τον επόμενο κόμβο με τη σειρά (ο μικρότερος κόμβος μεγαλύτερος από τον τρέχοντα κόμβο) στο δεξιό υποδέντρο. Ενημερώνει τα δεδομένα του τρέχοντος κόμβου με τα δεδομένα του κόμβου next\_in\_order. Στη συνέχεια, ανάλογα με το αν ο κόμβος next\_in\_order είναι το αριστερό ή το δεξί παιδί του γονέα του, ενημερώνεται ο κατάλληλος θυγατρικός δείκτης και ο κόμβος next\_in\_order ελευθερώνεται. Τέλος, η συνάρτηση επιστρέφει “TRUE” για να υποδείξει την επιτυχή διαγραφή.

**AVL\_NODE\_PTR rotate\_avl\_right(AVL\_NODE\_PTR node);**

**AVL\_NODE\_PTR rotate\_avl\_left(AVL\_NODE\_PTR node);**

Οι συναρτήσεις αυτές εκτελούν περιστροφή του δένδρου. Στα AVL δένδρα τα ύψη του αριστερού και δεξιού υποδέντρου οποιουδήποτε κόμβου διαφέρουν κατά ένα το πολύ. Παίρνουν σαν όρισμα ένα δείκτη σε κόμβο και δημιουργούν την «προσωρινή-temp» παράμετρο ιδίας μορφής με το όρισμα, στην οποία αποθηκεύουν το παιδί του κόμβου της αντίθετης πλευράς από αυτή που θα γίνει η περιστροφή. Στη συνέχεια αυτός ο κόμβος ενημερώνεται με το παιδί της πλευράς της παραμέτρου της πλευράς στην οποία γίνεται η περιστροφή. Έπειτα το παιδί της παραμέτρου στην πλευρά που γίνεται η περιστροφή ενημερώνεται με τον δείκτη του κόμβου που αρχικά πήραμε σαν όρισμα. Τα ύψη της παραμέτρου και του ορίσματος ανανεώνονται βάση τα σχετικά δεξιά και αριστερά τους υποδένδρα. Τέλος επιστρέφεται η παράμετρος η οποία πλέον είναι η ρίζα του υποδένδρου που περιστράφηκε.

**AVL\_NODE\_PTR rotate\_avl\_left\_right(AVL\_NODE\_PTR node);**

παίρνουν σαν όρισμα δείκτη σε κόμβο, εκτελεί αριστερή περιστροφή για το αριστερό παιδί του δείκτη με την συνάρτηση αριστερής περιστροφής που περιγράψαμε παραπάνω στην οποία στέλνει σαν όρισμα το αριστερό παιδί του δείκτη. Έπειτα επιστρέφει το αποτέλεσμα της συνάρτησης δεξιάς περιστροφής στην οποία στέλνει σαν όρισμα το όρισμα της συνάρτησης(το δείκτη σε κόμβο). Αντίστοιχα γίνεται στην **AVL\_NODE\_PTR rotate\_avl\_right\_left(AVL\_NODE\_PTR node);** προφανώς χρησιμοποιώντας το αντίθετο παιδί στην κάθε περίπτωση.

**AVL\_NODE\_PTR insert\_avl\_node\_by\_date(AVL\_NODE\_PTR \*root, RECORD rec);**

Η συνάρτηση αυτή εισάγει κάθε νέο κόμβο στο δένδρο. Τα δυαδικά δένδρα αναζήτησης ταξινομούνται κατά την εισαγωγή κάθε νέου κόμβου. Στην συγκεκριμένη περίπτωση το δένδρο ταξινομείται με βάση την ημερομηνία κάθε εγγραφής. Παίρνει σαν ορίσματα δείκτη σε κόμβο του δένδρου και την εγγραφή rec που εισάγεται. Ξεκινάει ελέγχοντας αν το δένδρο είναι άδειο και αν είναι εισάγει το νέο κόμβο ως ρίζα του δένδρου με την χρήση της συνάρτησης insert\_avl\_root την οποία περιγράψαμε παραπάνω. Αν το δένδρο δεν είναι άδειο συγκρίνει την ημερομηνία της εγγραφής που εισάγεται με την με την ημερομηνία της εγγραφής του τρέχων κόμβου χρησιμοποιώντας την συνάρτηση compare\_dates. Αν η ημερομηνία της εγγραφής είναι μικρότερη(πιο παλιά) από την ημερομηνία της εγγραφής του κόμβου η συνάρτηση insert\_avl\_node\_by\_date καλείται αναδρομικά ώστε ο νέος κόμβος να προστεθεί στο αριστερό υποδένδρο το τρέχοντα κόμβου. Μετά την εισαγωγή του κόμβου γίνεται έλεγχος αν το ύψος του αριστερού και δεξιού υποδένδρου είναι ίσο με 2. Αν είναι εκτελείται περιστροφή ώστε να επανα-ισσοροπήησει το δένδρο. Αν η ημερομηνία της νέας εγγραφής είναι μικρότερη κάποιου εκ των παιδιών του κόμβου, τότε εκτελείται περιστροφή προς την αντίθετη πλευρά. Αντίθετα, αν η ημερομηνία της νέας εγγραφής είναι μεγαλύτερη(πιο πρόσφατη) από την ημερομηνία της εγγραφής του κόμβου η συνάρτηση insert\_avl\_node\_by\_date καλείται αναδρομικά ώστε ο νέος κόμβος να προστεθεί στο δεξιό υποδένδρο το τρέχοντα κόμβου. Τελευταία περίπτωση που ελέγχεται είναι όταν η ημερομηνία της εγγραφής που εισάγεται είναι ίση με την ημερομηνία της εγγραφής του κόμβου. Τότε η νέα εγγραφή εισάγεται σε μια συνδεδεμένη λίστα ώστε όταν επαναλμβάνεται μια ημερομηνία η εγγραφή να αποθηκεύεται. Τα AVL δένδρα δεν αποθηκεύουν κόμβους με ίδιο κελιδί. Μετά την εισαγωγή του κόμβου γίνεται έλεγχος αν το ύψος του αριστερού και δεξιού υποδένδρου είναι ίσο με -2. Αν είναι εκτελείται περιστροφή ώστε να επανα-ισσοροπήησει το δένδρο. Αν η ημερομηνία της νέας εγγραφής είναι μικρότερη κάποιου εκ των παιδιών του κόμβου, τότε εκτελείται περιστροφή προς την αντίθετη πλευρά. Τέλος, το ύψος του τρέχοντος κόμβου ενημερώνεται με βάση το μέγιστο ύψος των αριστερών και δεξιών υποδέντρων του και επιστρέφεται ο τρέχων κόμβος. Παρόμοια είναι η εκτέλεση της **AVL\_NODE\_PTR insert\_avl\_node\_by\_value(AVL\_NODE\_PTR \*root, RECORD rec);** με την διαφορά ότι οι συγκρίσεις για την ταξινόμηση του δένδρου γίνονται με βάση την τιμή value της κάθε εγγραφής αφού στην συγκεκριμένη περίπτωση το δένδρο ταξινομείται με βάση την value.

**int delete\_avl\_node(AVL\_NODE\_PTR \*root, element users\_given\_date);**

Η συνάρτηση αυτή με την χρήση της int delete\_bst\_node(AVL\_NODE\_PTR \*root, element users\_given\_date); διαγράφει ένα κόμβο του δένδρου και το επανισορροπεί. Παίρνει σαν ορίσματα δείκτη σε κόμβο του δένδρου και την ημερομηνία που έδωσε ο χρήστης ώστε να βρεθεί ο κόμβος που θα διαγραφεί. Ορίζει επίσης τις εξής παραμέτρους: new\_parent\_node ένα πίνακα για την αποθήκευση των γονικών κόμβων του τρέχοντος κόμβου κατά τη διέλευση, previous\_parent\_nodes μια συστοιχία για την αποθήκευση της κατεύθυνσης (αριστερά ή δεξιά) που λαμβάνεται σε κάθε επίπεδο κατά τη διάρκεια της διέλευσης, index μια μεταβλητή ευρετηρίου για την παρακολούθηση της τρέχουσας θέσης στους πίνακες γονικού κόμβου και new\_root ένα δείκτη για την αποθήκευση της νέας ρίζας του δέντρου AVL (εάν η ρίζα αλλάξει κατά την εξισορρόπηση). Ελέγχει εάν το δέντρο AVL είναι κενό. Εάν ναι, εκτυπώνει ένα μήνυμα και επιστρέφει την κατάλληλη τιμή. Αρχικοποιεί τον τρέχοντα δείκτη στη ρίζα του δέντρου AVL και ξεκινά έναν βρόχο για να βρει τον κόμβο με τη δεδομένη τιμή ημερομηνίας. Συγκρίνει τη δεδομένη τιμή ημερομηνίας με την τιμή ημερομηνίας του τρέχοντος κόμβου και αν είναι ίσα, βγαίνει από τον βρόχο ειδάλλως αν είναι ίσα, βγαίνει από τον βρόχο. Εάν η δεδομένη ημερομηνία είναι μικρότερη από την ημερομηνία του τρέχοντος κόμβου, μετακινείται στο αριστερό παιδί και αποθηκεύει την κατεύθυνση στον πίνακα previous\_parent\_nodes. Εάν η δεδομένη ημερομηνία είναι μεγαλύτερη από την ημερομηνία του τρέχοντος κόμβου, μετακινείται στο σωστό παιδί και αποθηκεύει την κατεύθυνση στον πίνακα previous\_parent\_nodes. Αποθηκεύει επίσης τον γονικό κόμβο στον πίνακα new\_parent\_node και αυξάνει το index. Μετά τον βρόχο, ελέγχει εάν ο τρέχων δείκτης είναι NULL. Αν ναι, σημαίνει ότι δεν βρέθηκε κανένας κόμβος με τη δεδομένη τιμή ημερομηνίας και εκτυπώνει κατάλληλο μήνυμα και επιστρέφει FALSE. Εάν ο τρέχων κόμβος έχει και αριστερό και δεξιό παιδί, βρίσκει τον διάδοχο σειράς του τρέχοντος κόμβου (ο μικρότερος κόμβος στο δεξί υποδέντρο) και ενημερώνει τον γονικό κόμβο και την κατεύθυνση στους αντίστοιχους πίνακες. Καλεί τη συνάρτηση delete\_bst\_node για να διαγράψει τον κόμβο από το υποκείμενο δυαδικό δέντρο αναζήτησης (BST). Στη συνέχεια, εκτελεί εξισορρόπηση δέντρου AVL και τέλος εκτυπώνει κατάλληλο και επιστρέφει TRUE.

**AVL\_NODE\_PTR search\_avl\_node(AVL\_NODE\_PTR root, element users\_given\_date);**

Η συνάρτηση αυτή αναζητά ένα κόμβο στο δένδρο με βάση την ημερομηνία που έδωσε ο χρήστης. Η συνάρτηση παίρνει σαν ορίσματα ένα δείκτη σε κόμβο του δένδρου και την ημερομηνία που έδωσε ο χρήστης. Έπειτα δηλώνει την παράμετρο AVL\_NODE\_PTR current και την αρχικοποιεί στον τρέχοντα δείκτη στη ρίζα του δέντρου AVL. Ξεκινά έναν βρόχο για να αναζητήσει τον κόμβο με τη δεδομένη τιμή ημερομηνίας. Συγκρίνει τη δεδομένη τιμή ημερομηνίας με την τιμή ημερομηνίας του τρέχοντος κόμβου. Εάν είναι ίσοι, σημαίνει ότι βρέθηκε ο κόμβος με τη δεδομένη τιμή ημερομηνίας, επομένως επιστρέφει τον τρέχοντα κόμβο. Εάν η δεδομένη ημερομηνία είναι μικρότερη από την ημερομηνία του τρέχοντος κόμβου, μετακινείται στο αριστερό παιδί. Εάν η δεδομένη ημερομηνία είναι μεγαλύτερη από την ημερομηνία του τρέχοντος κόμβου, μετακινείται στο σωστό παιδί. Εάν ο βρόχος τελειώσει χωρίς να βρεθεί ένας αντίστοιχος κόμβος (δηλαδή, ο τρέχων δείκτης γίνεται NULL), σημαίνει ότι ο κόμβος με τη δεδομένη τιμή ημερομηνίας δεν υπάρχει στο δέντρο AVL. Σε αυτήν την περίπτωση, επιστρέφει NULL.

**void print\_avl\_node(AVL\_NODE\_PTR root);**

Η συνάρτηση εκτυπώνει το δένδρο σε ενδοδιατεταγμένη μορφή. Η συνάρτηση παίρνει ως όρισμα μια ρίζα, η οποία είναι ένας δείκτης στην ρίζα του δέντρου AVL. Η συνάρτηση διασχίζει αναδρομικά το δέντρο AVL. Ελέγχει πρώτα αν το αριστερό παιδί του τρέχοντος κόμβου (root->left) δεν είναι NULL. Εάν δεν είναι NULL, καλεί αναδρομικά τη συνάρτηση print\_avl\_node στο αριστερό παιδί, η οποία εκτυπώνει όλους τους κόμβους στο αριστερό υποδέντρο. Μετά την εκτύπωση του αριστερού υποδέντρου, εκτυπώνει τα δεδομένα του τρέχοντος κόμβου(τα πεδία ημερομηνίας και την value). Στη συνέχεια εκτυπώνει έναν χαρακτήρα νέας γραμμής (\n) για να διαχωρίσει τους κόμβους. Στη συνέχεια, ελέγχει αν το δεξί παιδί του τρέχοντος κόμβου (root->right) δεν είναι NULL. Εάν δεν είναι NULL, καλεί αναδρομικά τη συνάρτηση print\_avl\_node στο δεξιό παιδί, η οποία εκτυπώνει όλους τους κόμβους στο δεξιό υποδέντρο.

**int height(AVL\_NODE\_PTR root);**

Παίρνει σαν όρισμα δείκτη σε κόμβο του δένδρου, ελέγχει αν είναι άδειο και αν δεν είναι επιστρέφει το ύψος του κόμβου.

**int max(int x, int y);**

Παίρνει σαν ορίσματα δύο μεταβλητές, τις συγκρίνει και επιστρέφει την μεγαλύτερη.

**AVL\_NODE\_PTR search\_max\_value(AVL\_NODE\_PTR tree);**

Η συνάρτηση βρίσκει την μέγιστη τιμή value και επιστρέφει στην main δείκτη στον κόμβο της τιμής αυτής ώστε να εκτυπωθούν η/οι ημερομηνία/ες στην/ις οποία/ες η τιμή value είναι μέγιστη. Παίρνει σαν όρισμα δείκτη σε κόμβο του δένδρου. Αρχικά ελέγχει εάν το δέντρο είναι κενό και εάν είναι, επιστρέφει NULL υποδεικνύοντας ότι δεν υπάρχουν κόμβοι στο δέντρο. Εάν το δέντρο δεν είναι κενό, αρχικοποιεί στον δείκτη current ώστε να δείχνει στην ρίζα του δέντρου. Στη συνέχεια, η συνάρτηση εισέρχεται σε έναν βρόχο που συνεχίζεται όσο το δεξί παιδί του τρέχοντος κόμβου δεν είναι NULL. Έπειτα εκτελείται ένας βρόχος ο οποίος επαναλαμβάνεται μέσω των δεξιών θυγατρικών δεικτών των κόμβων, διασχίζοντας το δέντρο προς τη δεξιά πλευρά. Μέσα στον βρόχο, ο current ενημερώνεται για να δείχνει στο δεξί παιδί του τρέχοντος κόμβου σε κάθε επανάληψη, κινούμενος προς τη δεξιά πλευρά του δέντρου. Μόλις ο βρόχος τελειώσει, σημαίνει ότι ο τρέχων κόμβος είναι ο πιο δεξιός κόμβος στο δέντρο, ο οποίος αντιστοιχεί στον κόμβο με τη μέγιστη τιμή. Τέλος, η συνάρτηση επιστρέφει τον current, ο οποίος δείχνει στον κόμβο με τη μέγιστη τιμή στο δέντρο AVL. Πανομοιότυπα η συνάρτηση **AVL\_NODE\_PTR search\_min\_value(AVL\_NODE\_PTR tree);** υλοποιείται με τον ίδιο τρόπο με την διαφορά ότι στον βρόχο το δένδρο διασχίζεται προς τα αριστερά ώστε να βρεθεί η ελάχιστη τιμή.

**void date\_input\_check(element\* date);**

Η συνάρτηση ελέγχει τις ημερομηνίες που δίνει ο χρήστης ως προς την δομή και την εγκυρότητα της. Παίρνει σαν όρισμα την μεταβλητή στην οποία θα επιστραφεί η ημερομηνία που ο χρήστης θα δώσει ο χρήστης αφού ελέγχει. Ορίζει τις παραμέτρους day, month, year και valid\_date και ξεκινάει ένα βρόχο do while. Γίνεται είσοδος της ημερομηνίας της μορφής "%d/%d/%d", δηλαδή την μορφή που θα γράψει ο χρήστης την ημερομηνία και κάθε πεδίο αποθηκεύεται στα day, month και year και ταυτόχρονα γίνεται έλεγχος ότι δόθηκαν και οι 3 τιμές. Στην συνέχεια μέσα στο βρόχο ελέγχεται ότι η ημερομηνία βρίσκεται ενδιάμεσα στο 2015-2021 όπως και τα στοιχεία του csv αρχείου, αν ο μήνας και η μέρα που έχουν δοθεί επίσης είναι σωστά και όταν είναι σωστά η μεταβλητή valid\_date παίρνει την τιμή TRUE ενώ όταν δεν είναι παίρνει την τιμή FALSE γίνεται break, ζητείτε από τον χρήστη να δώσει νέα ημερομηνία και επαναλαμβάνεται ο επαναληπτικός κόμβος. Όταν δοθεί σωστή ημερομηνία τα πεδία day, month, year αποθηκεύονται στην date με την χρήση της sprintf.

**int hash\_function(element date[11]);**

Στους πίνακες κατακερματισμού, οι κόμβοι κατανέμονται σε buckets με βάση το αποτέλεσμα της συνάρτησης κατακερματισμού. Η συνάρτηση παίρνει σαν όρισμα την εγγραφή date του κάθε record και προσθέτει τους κωδικούς ASCII κάθε χαρακτήρα και έπειτα επιστρέφει το modulo του αθροίσματος με τον πλήθος των buckets.

**void insert\_record\_to\_hashTable(HashTable\* table, RECORD record);**

Η συνάρτηση εισάγει κόμβους στον πίνακα κατακερματισμού. Υπολογίζει την τιμή κατακερματισμού για το δεδομένο record.date καλώντας το hash\_function με την ημερομηνία ως όρισμα. Το αποτέλεσμα που προκύπτει καθορίζει τον κάδο όπου θα εισαχθεί η εγγραφή. Δεσμεύει μνήμη σε έναν νέο κόμβο (Node) για την αποθήκευση της εγγραφής. Εκχωρεί την εγγραφή στο πεδίο εγγραφής του νέου κόμβου και ορίζει το επόμενο πεδίο σε NULL. Ελέγχει εάν το bucket στον υπολογισμένο δείκτη είναι κενός (NULL). Εάν είναι άδειο, σημαίνει ότι δεν έχουν εισαχθεί ξανά εγγραφές σε αυτόν τον κάδο. Σε αυτήν την περίπτωση, εκχωρεί τον νέο κόμβο απευθείας στον κάδο. Εάν το bucket δεν είναι άδειος, σημαίνει ότι υπάρχουν ήδη εγγραφές σε αυτόν τον κάδο. Έπειτα διασχίζει τη συνδεδεμένη λίστα στον κάδο μέχρι να φτάσει στον τελευταίο κόμβο (current->next == NULL). Μόλις βρεθεί ο τελευταίος κόμβος, εκχωρεί τον νέο κόμβο ως τον επόμενο κόμβο του τελευταίου κόμβου, εισάγοντας ουσιαστικά τον νέο κόμβο στο τέλος της συνδεδεμένης λίστας στο bucket.

**Node\* search\_value\_by\_date(HashTable\* table, element users\_given\_date);**

Η συνάρτηση βρίσκει τον κόμβο με ημερομηνία ίδια με αυτή που έδωσε ο χρήστης και επιστρέφει δείκτη στον κόμβο αυτό, ώστε στην main να εκτυπωθεί το μέλος value της εγγραφής record του κόμβου αυτού. Υπολογίζει την τιμή της συνάρτησης κατακερματισμού για το δεδομένο users\_given\_date καλώντας το hash\_function με την ημερομηνία αυτή ως όρισμα. Το αποτέλεσμα που προκύπτει καθορίζει τον κάδο για αναζήτηση της τιμής. Ανακτά τη συνδεδεμένη λίστα (Node) στον κάδο στον υπολογισμένο δείκτη και την εκχωρεί στην τρέχουσα μεταβλητή. Επαναλαμβάνει τη συνδεδεμένη λίστα (current) μέχρι να φτάσει στο τέλος (current == NULL) ή βρίσκει έναν κόμβο με ημερομηνία που ταιριάζει (strcmp(current->record.date, users\_given\_date) == 0). Μέσα στον βρόχο, συγκρίνει την ημερομηνία του τρέχοντος κόμβου (current->record.date) με το users\_given\_date χρησιμοποιώντας το strcmp. Εάν ταιριάζουν, σημαίνει ότι βρέθηκε η επιθυμητή τιμή. Σε αυτήν την περίπτωση, επιστρέφει τον τρέχοντα κόμβο. Εάν ο βρόχος τελειώσει χωρίς να βρεθεί αντιστοιχία, σημαίνει ότι η τιμή δεν βρέθηκε στον πίνακα κατακερματισμού. Σε αυτήν την περίπτωση, επιστρέφει NULL και στο menu του hash table τυπώνεται το κατάλληλο μήνυμα.

**void delete\_hashTable\_record(HashTable\* table, element users\_given\_date);**

Η συνάρτηση βρίσκει τον κόμβο του πίνακα κατακερματισμού με ημερομηνία αυτή που έδωσε ο χρήστης και τον διαγράφει. Υπολογίζει την τιμή της συνάρτησης κατακερματισμού για το δεδομένο users\_given\_date καλώντας το hash\_function με την ημερομηνία ως όρισμα. Το αποτέλεσμα που προκύπτει καθορίζει τον κάδο για αναζήτηση της εγγραφής. Ανακτά τη συνδεδεμένη λίστα (Node) στον κάδο στον υπολογισμένο δείκτη και την εκχωρεί στην τρέχουσα μεταβλητή. Αρχικοποιεί επίσης μια μεταβλητή prev για να παρακολουθεί τον προηγούμενο κόμβο. Επαναλαμβάνει τη συνδεδεμένη λίστα (current) μέχρι να φτάσει στο τέλος (current == NULL) ή βρίσκει έναν κόμβο με ημερομηνία που ταιριάζει (strcmp(current->record.date, users\_given\_date) == 0). Μέσα στον βρόχο, συγκρίνει την ημερομηνία του τρέχοντος κόμβου (current->record.date) με το users\_given\_date χρησιμοποιώντας το strcmp. Εάν ταιριάζουν, σημαίνει ότι βρέθηκε η επιθυμητή εγγραφή. Εάν βρεθεί η αντιστοίχιση, ελέγχει εάν ο τρέχων κόμβος είναι η κεφαλή της συνδεδεμένης λίστας (δηλαδή, prev == NULL). Εάν είναι, ενημερώνει την κεφαλή της λίστας εκχωρώντας το table->buckets[index]. Διαφορετικά, ενημερώνει τον επόμενο δείκτη του προηγούμενου κόμβου (prev->next) για να παραλείψει τον τρέχοντα κόμβο. Απελευθερώνει τη μνήμη που καταλαμβάνεται από τον τρέχοντα κόμβο χρησιμοποιώντας το free(current) για να τον διαγράψει από τον πίνακα κατακερματισμού. Εκτυπώνει ένα μήνυμα που υποδεικνύει ότι η εγγραφή με τη δεδομένη ημερομηνία έχει διαγραφεί επιτυχώς και στη συνέχεια επιστρέφει από τη συνάρτηση. Εάν ο βρόχος τελειώσει χωρίς να βρεθεί αντιστοιχία, σημαίνει ότι η εγγραφή με τη δεδομένη ημερομηνία δεν βρέθηκε στον πίνακα κατακερματισμού. Σε αυτήν την περίπτωση, εκτυπώνει ένα μήνυμα που υποδεικνύει ότι η εγγραφή δεν βρέθηκε.

**int compare\_dates(const char\* date1, const char\* date2);**

Η συνάρτηση χρησιμοποιείται για να γίνεται σωστός έλεγχος των ημερομηνιών, δηλαδή να συγκρίνονται πρώτα τα έτη, μετά οι μήνες και τελευταίες οι ημέρες δύο ημερομηνιών. Για να επιτευχθεί αυτό η συνάρτηση παίρνει σαν όρισμα δύο ημερομηνίες και με την χρήση της sscanf τις χωρίζει στα πεδία ημέρα, μήνες, χρόνος και αποθηκεύει κατάλληλα. Έπειτα γίνεται σύγκριση πρώτα μεταξύ των ετών έπειτα των μηνών και τέλος των ημερών. Αν σε κάποιο από τους τρεις ελέγχους βρεθεί ανισότητα επιστρέφεται η διαφορά των δύο. Στις συναρτήσεις που επιστρέφονται γίνεται έλεγχος αν η τιμή που επιστράφηκε είναι μικρότερη, μεγαλύτερη ή ίση του 0.