Project DS - Part II

Μελιτσόπουλος Φώτιος	e-mail: up1084600@upnet.gr	AM: 1084600
Ζάγκλας Μάριος	e-mail: up1084686@upnet.gr	AM: 1084686
Ψαθογιαννάκης Χρήστος	e-mail: up1090052@upnet.gr	AM: 1090052

Επεξήγηση Υλοποίησης Κώδικα

current directory: project/part 2/basics

Αρχείο basics.cpp: *αρχείο κώδικα που περιέχει τα definitions των παρακάτω στοιχείων:*

int num (int num) {} → επιστρέφει την τιμή num – 48 ουσιαστικά μετατρέποντας το ascii value ενός χαρακτήρα ψηφίου στην αντίστοιχή του τιμή: πχ: αφού ascii [‘ 0 ’] = 48, τότε num (‘ 0 ’) = 48 – 48 = 0.

void space (void) {} → εκτυπώνει τον χαρακτήρα ‘ = ’ και χρησιμοποιείται ώστε να είναι το menu πιο ευανάγνωστο.

std :: string check (std :: string date) {} → ελέγχει εάν η μεταβλητή date τύπου std :: string είναι μορφοποιημένη κατάλληλα για να εισαχθεί στο σύστημα ως ημερομηνία. Αν αυτό ισχύει, απλά επιστρέφει την τιμή της μεταβλητής, διαφορετικά ζητά σε βρόγχο από τον χρήστη καινούρια ημερομηνία, έως ότου αυτή είναι μορφοποιημένη σωστά.

void exit (bool flag) {} → εμφανίζει κατάλληλο μήνυμα ενημερώνοντας τον χρήστη πως η εφαρμογή πρόκειται να τερματιστεί, και ανάλογα την τιμή της μεταβλητής bool flag, τερματίζει επιστρέφοντας είτε exit code 0 σε επιτυχή τερματισμό, είτε exit code 1 σε ανεπιτυχή τερματισμό. Exit code 0 επιστρέφεται μόνο κατά την επιλογή του χρήστη να τερματίσει την εφαρμογή. Διαφορετικά, σε σφάλμα που ενδεχομένως προκύψει κατά το runtime, ο κώδικας τερματίζει με τιμή exit code 1.

Αρχείο basics.hpp: *header file του αρχείου basics.cpp. Περιέχει τα declarations όλων των functions εντός αυτού. Επιπλέον, περιέχει:*

typedef std :: chrono :: steady_clock clk;

typedef std :: chrono :: milliseconds ms;

typedef std :: chrono :: microseconds us

→ αποτελούν δημιουργίες ψευδονύμων για ορισμένα στοιχεία της βιβλιοθήκης <chrono>. Χρησιμοποιούνται για την μέτρηση χρόνου εκτέλεσης διαφόρων λειτουργιών του Part 2.

current directory: project/part 2/avl/shared

Σε αυτό το σημείο αξίζει να σημειωθεί πως τα AVL trees απεικονίζονται μέσω του κώδικα ως κομβοπροσανατολισμένα. Η επιλογή αυτή στηρίζεται στην έλλειψη πολλαπλών instances του ίδιου κόμβου, συνεπώς και η υλοποίηση δεσμεύει λιγότερο χώρο στη μνήμη, καθώς και πολλές από τις διαδικασίες όπως εύρεση, εισαγωγή, τροποποίηση και αφαίρεση είναι ευκολότερο να υλοποιηθούν, καθώς αφορούν μόνον ένα κόμβο εντός του AVL tree.

Αρχείο shared.cpp: *αρχείο κώδικα που περιέχει τα definitions των παρακάτω στοιχείων:*

AVLNode * root = nullptr; → pointer σε στοιχείο AVLNode το οποίο αρχικοποιείται με την τιμή nullptr, δηλώνοντας πως αρχικά δεν είναι initialized το AVL tree. Αν αυτό αρχικοποιηθεί, τότε δείχνει στον κόμβο – ρίζα του AVL tree, εξού και το όνομα.

int height (AVLNode * node) {} → υπολογίζεται και ενημερώνεται το ύψος του κόμβου node, συγκρίνοντας μέσω recursion τα ύψη των υποδέντρων του, και αυξάνοντας κατά 1 το μεγαλύτερο από αυτά. Αν το recursion φτάσει σε κόμβο ο οποίος δεν υπάρχει (node == nullptr), τότε επιστρέφεται η τιμή 0. Έτσι τα leaf nodes έχουν ύψος 0 + 1 = 1 κοκ.

void update (void) {} → καλείται το function height στον κόμβο root, ενημερώνοντας έτσι τα ύψη όλων των κόμβων εντός του AVL tree.

int bal (AVLNode * node) {} → επιστρέφει την διαφορά των υψών των υποδέντρων του κόμβου node (height of right subtree – height of left subtree).

void RR (AVLNode * node) {} → πραγματοποιεί δεξιά περιστροφή ανάμεσα στον κόμβο – πατέρα node και στον αριστερό του κόμβο – παιδί, αν όντως αυτοί οι δύο κόμβοι υπάρχουν. Επιπλέον πραγματοποιείται έλεγχος για το αν ο κόμβος – πατέρας ταυτίζεται με την ρίζα, στην οποία περίπτωση πρέπει να αλλάξει και ο pointer της ρίζας, ώστε να δείχνει στην καινούρια ρίζα μετά την περιστρόφη. Ακόμα, υπάρχει διαχείριση πιθανών κόμβων – παππού και εγγονιού, ώστε να διατηρείται η δομή του AVL tree. Τέλος, γίνεται κλήση της update () ενημερώνοντας έτσι τα ύψη όλων των κόμβων εντός του AVL tree.

void LL (AVLNode * node) {} → ανάλογη του function void RR (AVLNode * node), αλλά εκτελεί την αριστερή περιστροφή, αντί της δεξιάς, ανάμεσα στον κόμβο – πατέρα node και στο δεξί του κόμβο – παιδί, αν όντως αυτοί υπάρχουν. Αυτά τα δύο functions διαφέρουν μόνο στο ποιοι κόμβοι συμμετέχουν.

bool rotation (AVLNode * node) {} → ελέγχει αν υπάρχει λόγος περιστροφής (παραβίαση της συνθήκης του balance factor ενός κόμβου ενός AVL tree) ανάμεσα στον κόμβο - πατέρα node, στους κόμβους – παιδιά του και στους κόμβους – εγγόνια του, ελέγχοντας πρώτα αν υπάρχουν οι κόμβοι οι οποίοι πρόκειται να συμμετάσχουν σε κάποια περιστροφή. Αν αυτοί δεν υπάρχουν το function επιστρέφει τιμή false δηλώνοντας ανεπιτυχή περιστροφή. Διαφορετικά, ανάλογα την τιμή του balance factor του κόμβου – πατέρα node και των τιμών των balance factors των κόμβων – παιδιών του πραγματοποιείται είτε δεξιά ή αριστερή απλή περιστροφή (RR rotation – LL rotation) είτε δεξιά ή αριστερή διπλή περιστροφή (LR rotation – RL rotation). Τέλος, εφόσον έχει πραγματοποιήσει κάποια από τις περιστροφές, επιστρέφει τιμή true δηλώνοντας επιτυχή περιστροφή.

void balPath (AVLNode * node) {} → πραγματοποιεί περιστροφές, όπου αυτό κρίνεται αναγκαία, στο path από τον κόμβο node εώς την ρίζα.

AVLNode * search (AVLNode * node, int key) {} → επιστρέφει είτε τον κόμβο του οποίου το key ταυτίζεται με το key που δίνεται ως όρισμα, αν αυτός υπάρχει, διαφορετικά επιστρέφει τον κόμβο φύλλο που προσπελάστηκε τελευταίος. Η αναζήτηση πραγματοποιείται χρησιμοποιώντας recursion στο αριστερό ή δεξί υπόδεντρο του κόμβου node, ανάλογα με το αποτέλεσμα της σύγκρισης του key του κόμβου node, σε σχέση με το key που δίνεται ως όρισμα.

AVLNode * search (int key) {} → επικαλύπτει το function AVLNode * search (AVLNode * node, int key) ώστε η αναζήτηση να ξεκινά πάντα από την ρίζα του AVL tree.

AVLNode * create (AVLNode * node, std :: string date, float temp, int key) {} → δημιουργεί έναν καινούριο κόμβο, αρχικοποιεί τον κόμβο αυτό με τις τιμές date, temp και key, ορίζει ως πατέρα αυτού του κόμβου τον κόμβο node, και ελέγχει που θα πρέπει να τοποθετηθεί αυτός ο κόμβος στο AVL tree, ανάλογα το αν υπάρχει ρίζα ή όχι, καθώς και με το αποτέλεσμα της σύγκρισης της τιμής key του καινούριου κόμβου σε σχέση με του πατέρα του. Τέλος, επιστρέφει pointer αυτό τον κόμβο.

Αρχείο shared.hpp: *header file του αρχείου shared.cpp. Περιέχει τα declarations όλων των functions εντός αυτού. Επιπλέον, περιέχει:*

struct AVLNode {}; → definition του struct AVLNode, το οποίο αντιπροσωπεύει ένα node ενός AVL tree. Περιέχει τα πεδία std :: vector <std :: string> date και std :: vector <float> temp στα οποία αποθηκεύουμε ανάλογα το είδος του AVL tree, μία ή πολλαπλές ημερομηνίες και θερμοκρασίες. Ακόμα, περιέχει και την μεταβλητή int key στην οποία αποθηκεύουμε είτε μια τροποποιημένη μορφή μιας ημερομηνίας (date sorted AVL tree), είτε την ίδια την θερμοκρασία πολλαπλασιασμένη με το 100 (temperature sorted AVL tree), επιτρέποντας έτσι την ταξινόμηση των nodes εντός του AVL tree. Επιπλέον, περιέχει ένα πεδίο int

height, στο οποίο αποθηκεύεται το ύψος του κόμβου εντός του AVL tree. Τέλος, περιέχει και 3 pointers AVLNode * r / l / p (right / left / parent) οι οποίοι αρχικοποιούνται με την τιμή nullptr, των οποίων η κατάλληλη διαχείριση έχει ως αποτέλεσμα την σωστή δόμηση του AVL tree.

extern AVLNode * root; → δηλώνει πως η μεταβλητή AVLNode * root εντός του αρχείου shared.cpp υπάρχει και είναι προσβάσιμη από κάθε αρχείο που κάνει #include το header file shared.hpp.

current directory: project/part 2/avl/date

Αρχείο date.cpp: *αρχείο κώδικα που περιέχει τα definitions των παρακάτω στοιχείων:*

bool dateflag = false; → καθορίζει αν έχει δημιουργηθεί το date sorted AVL tree ή όχι.

bool searchTemp (AVLNode * node, float temp) {} → ψάχνει το διάνυσμα node->temp, εάν ο κόμβος node υπάρχει (αν δεν υπάρχει επιστρέφει αυτόματα false), και αν βρει την τιμή temp εντός αυτού, τότε επιστρέφει true, δηλώνοντας επιτυχή εύρεση, διαφορετικά επιστρέφει false.

bool insertTemp (AVLNode * node, float temp) {} → ελέγχει καλώντας το function bool searchTemp (AVLNode * node, float temp), αν η τιμή temp υπάρχει εντός του διανύσματος node->temp, εάν ο κόμβος node υπάρχει (αν δεν υπάρχει επιστρέφει αυτόματα false), και αν αυτή επιστρέψει false, εισάγει στο διάνυσμα αυτό την τιμή temp και επιστρέφει true, δηλώνοντας επιτυχή εισαγωγή τιμής. Αν η τιμή, ωστόσο, βρεθεί, τότε επιστρέφει false, αποφεύγοντας έτσι πολλαπλά instances της ίδιας τιμής.

void printTemps (AVLNode * node) {} → εμφανίζει με σωστό format όλες τις τιμές εντός του διανύσματος node->temp, εάν ο κόμβος node υπάρχει.

float convert (std :: string date) {} → επιστρέφει την τροποποιημένη μορφή της τιμής date ως int. Συγκεκριμένα, αγνοεί τις καθέτους, και οργανώνει το τελικό αποτέλεσμα κατά αύξουσα σειρά ετών, μηνών και τέλος ημερών, π.χ. αφού date = 18/09/2453, τότε convert (date) = 24530918, όπου 2453 το έτος, 09 ο μήνας και 18 η ημέρα. Έτσι, η μορφή αυτή μπορεί να χρησιμοποιηθεί για την ταξινόμηση των κόμβων εντός του AVL tree.

bool insertNode (std :: string date, float temp) {} → αρχικά ελέγχει αν το AVL tree είναι αρχικοποιημένο σωστά (διαφορετικά επιστρέφει false), και στη συνέχεια, ελέγχει βάσει της ημερομηνίας, αν θα δημιουργηθεί καινούριος κόμβος (η ημερομηνία date δεν έχει ακόμα καταχωρηθεί), ή αν απλά θα εισαχθεί η καινούρια θερμοκρασία σε κάποιον υπάρχοντα (η ημερομηνία date έχει ήδη καταχωρηθεί). Στην περίπτωση δημιουργίας καινούριου κόμβου, τότε καλείται το function void balPath (AVLNode * node), επαναφέροντας την ζύγιση στο path από το σημείο εισαγωγής του καινούριου κόμβου, έως και τη ρίζα. Είτε εισαχθεί καινούρια θερμοκρασία, ή δημιουργηθεί καινούριος κόμβος (τον οποίο αναζητεί εντός του AVL tree, για να πιστοποιήσει πως εισάχθηκε σωστά), επιστρέφεται η τιμή true δηλώνοντας επιτυχή εισαγωγή των στοιχείων εντός του AVL tree.

bool createDate (void) {} → εξάγει τα δεδομένα από το αρχείο ocean.csv και τα εισάγει ανά γραμμή ως κόμβο στο AVL tree καλώντας το function bool insertNode (std :: string date, float temp). Αν η εισαγωγή αυτή αποτύχει, τότε εμφανίζεται ανάλογο μήνυμα και η εφαρμογή τερματίζεται με exit code 1.

Επιστρέφει true μόνο όταν τελείωσει πλήρως η διαδικασία αυτή, δηλώνοντας πως το AVL tree έχει δημιουργηθεί επιτυχώς.

int iotDate (AVLNode * node, int counter) {} → Εκτελεί ενδοδιατεταγμένη διάσχιση στο υπόδεντρο με ρίζα τον κόμβο node. Αν ο κόμβος node δεν υπάρχει (node = nullptr), τότε απλά επιστρέφει την τιμή counter. Διαφορετικά, πρώτα πραγματοποιεί recursion στο αριστερό υπόδεντρο του κόμβου node, έπειτα εμφανίζει τα περιεχόμενα των διανυσμάτων node->date, και node->temp, και στη συνέχεια πραγματοποιεί recursion στο δεξί υπόδεντρο του κόμβου node. Τέλος, επιστρέφει την μεταβλητή counter, η οποία χρησιμοποιείται για την αρίθμηση των κόμβων κατά την ενδοδιατεταγμένη διάσχιση.

int iotDate (void) {} → wrapper του function int iotDate (AVLNode * node, int counter), ώστε η ενδοδιατεταγμένη διάσχιση να ξεκινά πάντα από την ρίζα, και η αρίθμηση από το 1.

bool modifyTemp (int key, float oldTemp, float newTemp) {} → αντικαθιστά την τιμή oldTemp με την τιμή newTemp, αν αυτή υπάρχει, εντός διανύσματος node->temp, αν ο κόμβος node υπάρχει. Επιστρέφει true στην περίπτωση της επιτυχούς αντικατάστασης, διαφορετικά επιστρέφει false.

bool erase (int key) {} → διαγράφει τον κόμβο στον οποίο αντιστοιχεί η τιμή int key την οποία δέχεται ως όρισμα, αν αυτός υπάρχει. Η διαγραφή αυτή διακρίνεται σε περιπτώσεις ανάλογα με το τι είδους κόμβος είναι ο κόμβος διαγραφής. Οι περιπτώσεις αναλύονται στα σχόλια εντός του αρχείου κώδικα, αλλά η βασική ιδέα είναι κατάλληλη διαχείριση των pointers r / l / p και επαναζύγιση του AVL tree όπου χρειάζεται, καθώς και απελευθέρωση της μνήμης που δεσμεύει ο κόμβος που διαγράφεται. Τέλος, επιστρέφει την τιμή true στην περίπτωση επιτυχούς διαγραφής, διαφορετικά επιστρέφει την τιμή false.

*Τα functions **void dateOptions (void) {}**, **void dateOption1 (void) {}**, **void dateOption2 (void) {}**, **void dateOption3 (void) {}**, **void dateOption4 (void) {}**, **void dateOption5 (void) {}** και **void menuDate (void) {}** εξυπηρετούν στην κατάτμηση και ευκολότερη διαχείριση τόσο του submenu του date sorted AVL tree, καθώς και των επιλογών του. Αν ο χρήστης επιλέξει την έξοδο από αυτό το submenu η ρίζα λαμβάνει την τιμή nullptr, και η μεταβλητή dateflag λαμβάνει την τιμή false, πρακτικά διαγράφοντας το προηγούμενο AVL tree που δημιουργήθηκε, καθώς και όλων των τροποποιήσεων που είχαν πραγματοποιηθεί σε αυτό.*

Αρχείο date.hpp: *header file του αρχείου date.cpp. Περιέχει τα declarations όλων των functions εντός αυτού. Επιπλέον, περιέχει:*

extern bool dateflag; → δηλώνει πως η μεταβλητή bool dateflag εντός του αρχείου date.cpp υπάρχει και είναι προσβάσιμη από κάθε αρχείο που κάνει #include το header file date.hpp.

current directory: project/part 2/avl/temp

Αρχείο temp.cpp: *αρχείο κώδικα που περιέχει τα definitions των παρακάτω στοιχείων:*

bool tempflag = false; → καθορίζει αν έχει δημιουργηθεί το temperature sorted AVL tree ή όχι.

bool searchDate (AVLNode * node, std :: string date) {} → ψάχνει το διάνυσμα node->date, εάν ο κόμβος node υπάρχει (αν δεν υπάρχει επιστρέφει αυτόματα false), και αν βρει την τιμή date εντός αυτού, τότε επιστρέφει true, δηλώνοντας επιτυχή εύρεση, διαφορετικά επιστρέφει false.

bool insertDate (AVLNode * node, std :: string date) {} → ελέγχει καλώντας το function bool searchDate (AVLNode * node, std :: string date), αν η τιμή date υπάρχει εντός του διανύσματος node->date, εάν ο κόμβος node υπάρχει (αν δεν υπάρχει επιστρέφει αυτόματα false), και αν αυτή επιστρέψει false, εισάγει στο διάνυσμα αυτό την τιμή date και επιστρέφει true, δηλώνοντας επιτυχή εισαγωγή τιμής. Αν η τιμή, ωστόσο, βρεθεί, τότε επιστρέφει false, αποφεύγοντας έτσι πολλαπλά instances της ίδιας τιμής.

void printDates (AVLNode * node) {} → εμφανίζει όλες τις τιμές εντός του διανύσματος node->temp, εάν ο κόμβος node υπάρχει.

bool insertNode (float temp, std :: string date) {} → αρχικά ελέγχει αν το AVL tree είναι αρχικοποιημένο σωστά (διαφορετικά επιστρέφει false), και στη συνέχεια, ελέγχει βάσει της θερμοκρασίας, αν θα δημιουργηθεί καινούριος κόμβος (η θερμοκρασία temp δεν έχει ακόμα καταχωρηθεί), ή αν απλά θα εισαχθεί η καινούρια ημερομηνία σε κάποιον υπάρχοντα (η θερμοκρασία temp έχει ήδη καταχωρηθεί). Στην περίπτωση δημιουργίας καινούριου κόμβου, τότε καλείται το function void balPath (AVLNode * node), επαναφέροντας την ζύγιση στο path από το σημείο εισαγωγής του καινούριου κόμβου, έως και τη ρίζα. Είτε εισαχθεί καινούρια ημερομηνία, ή δημιουργηθεί καινούριος κόμβος (τον οποίο αναζητεί εντός του AVL tree, για να πιστοποιήσει πως εισάχθηκε σωστά), επιστρέφεται η τιμή true δηλώνοντας επιτυχή εισαγωγή των στοιχείων εντός του AVL tree.

bool createTemp (void) {} → εξάγει τα δεδομένα από το αρχείο ocean.csv και τα εισάγει ανά γραμμή ως κόμβο στο AVL tree καλώντας το function bool insertNode (float temp, std :: string date). Αν η εισαγωγή αυτή αποτύχει, τότε εμφανίζεται ανάλογο μήνυμα και η εφαρμογή τερματίζεται με exit code 1. Επιστρέφει true μόνο όταν τελείωσει πλήρως η διαδικασία αυτή, δηλώνοντας πως το AVL tree έχει δημιουργηθεί επιτυχώς.

int iotTemp (AVLNode * node, int counter) {} → Εκτελεί ενδοδιατεταγμένη διάσχιση στο υπόδεντρο με ρίζα τον κόμβο node. Αν ο κόμβος node δεν υπάρχει (node = nullptr), τότε απλά επιστρέφει την τιμή counter. Διαφορετικά, πρώτα πραγματοποιεί recursion στο αριστερό υπόδεντρο του κόμβου node,

έπειτα εμφανίζει τα περιεχόμενα των διανυσμάτων node->temp, και node->date, και στη συνέχεια πραγματοποιεί recursion στο δεξί υπόδεντρο του κόμβου node. Τέλος, επιστρέφει την μεταβλητή counter, η οποία χρησιμοποιείται για την αρίθμηση των κόμβων κατά την ενδοδιατεταγμένη διάσχιση.

int iotTemp (void) {} → wrapper του function int iotTemp (AVLNode * node, int counter), ώστε η ενδοδιατεταγμένη διάσχιση να ξεκινά πάντα από την ρίζα, και η αρίθμηση από το 1.

AVLNode * findMin (void) {} → επιστρέφει τον πιο αριστερό κόμβο φύλλο εντός του AVL tree, αφού σε αυτόν περιέχεται, σύμφωνα με την ταξινόμηση κατά θερμοκρασία, η ελάχιστη θερμοκρασία.

AVLNode * findMax (void) {} → επιστρέφει τον πιο δεξιό κόμβο φύλλο εντός του AVL tree, αφού σε αυτόν περιέχεται, σύμφωνα με την ταξινόμηση κατά θερμοκρασία, η μέγιστη θερμοκρασία.

*Τα functions **void tempOptions (void) {}**, **void tempOption1 (void) {}**, **void tempOption2 (void) {}**, **void tempOption3 (void) {}, void tempOption4 (void) {}** και **void menuTemp (void) {}** εξυπηρετούν στην κατάτμηση και ευκολότερη διαχείριση τόσο του submenu του temperature sorted AVL tree, καθώς και των επιλογών του. Αν ο χρήστης επιλέξει την έξοδο από αυτό το submenu η ρίζα λαμβάνει την τιμή nullptr, και η μεταβλητή dateflag λαμβάνει την τιμή false, πρακτικά διαγράφοντας το προηγούμενο AVL tree που δημιουργήθηκε.*

Αρχείο temp.hpp: *header file του αρχείου temp.cpp. Περιέχει τα declarations όλων των functions εντός αυτού. Επιπλέον, περιέχει:*

extern bool tempflag; → δηλώνει πως η μεταβλητή bool tempflag εντός του αρχείου temp.cpp υπάρχει και είναι προσβάσιμη από κάθε αρχείο που κάνει #include το header file temp.hpp.

current directory: project/part 2/hash

Σε αυτό το σημείο αξίζει να σημειωθεί πως αφού πραγματοποιήθηκε η υλοποίηση της δομής Chain Hashing, χρησιμοποιώντας τη μεταβλητή buckets ως μέγεθος του πίνακα, αποφασίσαμε να πειραματιστούμε σύντομα με την τιμή αυτής, καθώς το λάβαμε σαν πρόταση από το παράδειγμα εντός της εκφώνησης, όχι αυστηρή οδηγία. Αποφασίσαμε να δημιουργήσουμε δομές Chain Hashing για τιμές buckets = 3, 5, μικρότερες από την αρχική τιμή 11, καθώς και για τιμές buckets = 18, 23, μεγαλύτερες από αυτή. Καταλήξαμε όμως γρήγορα στο συμπέρασμα πως για μικρές τιμές της μεταβλητής buckets, η δομή Chain Hashing δεν έβγαζε νόημα αφού όσο μικρότερη η τιμή, τόσο έτεινε η αναζήτηση σε γραμμική, η οποία συνεπάγεται μεγαλύτερου χρόνου αναζήτησης, ενώ για μεγάλες τιμές, η κατανομή των στοιχείων εντός των αλυσίδων ήταν αρκετά κακή, με αποτέλεσμα να υπάρχει μεγάλη διαφορά αναφορικά με το μέγεθος κάθε αλυσίδας. Έτσι, διατηρήσαμε την τιμή buckets = 11, αφού αυτή φαίνεται να συνεισφέρει στη δημιουργία μιας δομής Chain Hashing, στην οποία τα δεδομένα είναι σχετικά καλά κατανεμημένα, καθώς και κάθε αλυσίδα έχει αρκετά μικρό μέγεθος, ώστε να αποκλίνει από την γραμμική αναζήτηση (στην χειρότερη περίπτωση, λόγω σχετικά καλής κατανομής η αναζήτηση εντός της δομής απαιτεί περίπου το 1 / 11 του χρόνου της γραμμικής αναζήτησης εντός ενός μονοδιάστατου πίνακα, αφού μοιράζονται σε 11 περίπου ισομεγέθη διανύσματα).

Αρχείο hash.cpp: *αρχείο κώδικα που περιέχει τα definitions των παρακάτω στοιχείων:*

bool hashflag = false; → καθορίζει αν έχει αρχικοποιηθεί το Hash Table tree ή όχι.

const int buckets = 11; → καθορίζει το πλήθος των buckets του Hash table.

std :: vector <HashNode * > hashTable[buckets]; → δημιουργεί τον πίνακα hashTable μεγέθους όσο καθορίζει η μεταβλητή buckets, τύπου διανυσμάτων στα οποία αντιστοιχούν pointers σε elements τύπου HashNode.

bool insert (std :: string date, float temp) {} → αν η τιμή date ενός κόμβου ήδη εισαγμένου εντός του hashTable ταυτίζεται με αυτή που λαμβάνει το function ως όρισμα, τότε η τιμή temp του κόμβου αυτού αντικαθίσταται από αυτή που λαμβάνει το function ως όρισμα. Διαφορετικά δημιουργείται καινούριος κόμβος HashNode, ο οποίος λαμβάνει τις τιμές date και temp, και τοποθετείται σε ένα από τα διανύσματα του πίνακα hashTable βάσει της τιμής που επιστρέφει το function calcHash (std :: string date), το οποίο εξηγείται παρακάτω. Τέλος ελέγχει αν ο καινούριος αυτός κόμβος εντάχθηκε σωστά εντός του hashTable. Αν ο έλεγχος πετύχει, επιστρέφεται η τιμή true δηλώνοντας επιτυχή εισαγωγή κόμβου εντός του πίνακα, διαφορετικά επιστρέφει false.

bool createHash (void) {} → εξάγει τα δεδομένα από το αρχείο ocean.csv και τα εισάγει ανά γραμμή ως κόμβο στο hashTable καλώντας το function bool insert (std :: string date, float temp). Αν η εισαγωγή αυτή αποτύχει, τότε εμφανίζεται ανάλογο μήνυμα και η εφαρμογή τερματίζεται με exit code 1. Επιστρέφει true μόνο όταν τελείωσει πλήρως η διαδικασία αυτή, δηλώνοντας πως το hashTable έχει συμπληρωθεί επιτυχώς.

int calcHash (std :: string date) {} → υπολογίζει το ascii sum όλων των χαρακτήρων εντός της τιμής date, και επιστρέφει το υπόλοιπο το sum αυτού διαιρεμένο με το πλήθος των buckets. Έτσι, μπορεί κάθε ημερομηνία να ταξινομηθεί σε ένα από τα 11 (εφόσον buckets = 11) διανύσματα εντός του hashTable.

void printBuckets (void) {} → εμφανίζει τα περιεχόμενα κάθε κόμβου εντός του hashTable, αναδεικνύοντας έτσι την δομή Chain Hashing.

HashNode * search (std :: string date) {} → αναζητεί εντός του διανύσματος του hashTable, στο οποίο αντιστοιχεί το hash value της τιμής date την οποία λαμβάνει ως όρισμα, εάν υπάρχει κόμβος, ο οποίος περιέχει την τιμή αυτή. Αν υπάρχει τότε επιστρέφεται pointer σε αυτόν, δηλώνοντας αυτόματα επιτυχή αναζήτηση, διαφορετικά επιστρέφει nullptr, δηλώνοντας ανεπιτυχή αναζήτηση.

bool modify (std :: string date, float temp) {} → καλεί το function HashNode * search (std :: string date), για να αναζητήσει αν υπάρχει εντός του hashTable κόμβος που περιέχει ήδη την τιμή date. Αν δεν υπάρχει τότε επιστρέφει αυτόματα false, δηλώνοντας ανεπιτυχή τροποποίηση. Διαφορετικά, αντικαθιστά την τιμή temp του κόμβου αυτού με την τιμή temp που λαμβάνει ως όρισμα, και επιστρέφει true δηλώνοντας επιτυχή τροποποίηση.

bool erase (std :: string date) {} → αναζητεί εντός του διανύσματος του hashTable, στο οποίο αντιστοιχεί το hash value της τιμής date την οποία λαμβάνει ως όρισμα, εάν υπάρχει κόμβος, ο οποίος περιέχει την τιμή αυτή. Αν υπάρχει, τότε διαγράφει τον κόμβο αυτό, και επιστρέφει true δηλώνοντας επιτυχή διαγραφή στοιχείου, διαφορετικά επιστρέφει false δηλώνοντας ανεπιτυχή διαγραφή.

*Τα functions **void hashOptions (void) {}, void hashOption1 (void) {}, void hashOption2 (void) {}, void hashOption3 (void) {}, hashOption4 (void) {}, hashOption5 (void) {}** και **void menuHash (void) {}** εξυπηρετούν στην κατάτμηση και ευκολότερη διαχείριση τόσο του submenu του Chain Hashing, καθώς και των επιλογών του. Αν ο χρήστης επιλέξει την έξοδο από αυτό το submenu η μεταβλητή dateflag λαμβάνει την τιμή false, και τα διανύσματα εντός του πίνακα διαγράφουν όλα τους τα περιεχόμενα, πρακτικά καταστρέφοντας την δομή Chain Hashing.*

Αρχείο hash.hpp: *header file του αρχείου hash.cpp. Περιέχει τα declarations όλων των functions εντός αυτού. Επιπλέον, περιέχει:*

struct HashNode {}; → definition του struct HashNode, το οποίο αντιπροσωπεύει ένα node μιας αλυσίδας της δομής Chain Hashing. Περιέχει τα πεδία std :: string date και float temp στα οποία αποθηκεύουμε την ημερομηνία και θερμοκρασία αντίστοιχα, ανά γραμμή data εντός του αρχείου ocean.csv.

extern bool hashflag; → δηλώνει πως η μεταβλητή bool hashflag εντός του αρχείου hash.cpp υπάρχει και είναι προσβάσιμη από κάθε αρχείο που κάνει #include το header file hash.hpp.

extern const int buckets; → δηλώνει πως η "σταθερά" buckets εντός του αρχείου hash.cpp υπάρχει και είναι προσβάσιμη από κάθε αρχείο που κάνει #include το header file hash.hpp.

extern std :: vector <HashNode * > hashTable[]; → δηλώνει πως ο πίνακας hashTable εντός του αρχείου hash.cpp υπάρχει και είναι προσβάσιμος από κάθε αρχείο που κάνει #include το header file hash.hpp.

current directory: project/part 2/menu

Αρχείο menu.cpp: *αρχείο κώδικα που περιέχει τα definitions των functions* ***void menuOptions1 (void) {}, void menuOptions2 (void) {} και void menu (void) {}****, οι οποίες εξυπηρετούν στην κατάτμηση και ευκολότερη διαχείριση του κυρίως menu, καθώς και των επιλογών του. Αν ο χρήστης επιλέξει την έξοδο από αυτό το menu, τότε εμφανίζεται ανάλογο μήνυμα ειδοποιώντας αυτόν πως το preview του κώδικα αυτού τελείωσε, και τερματίζει με exit code 0, δηλώνοντας πως όλα έτρεξαν όπως έπρεπε, και δεν παρουσιάστηκαν τυχόν σφάλματα.*

Επεξήγηση Δοκιμής Κώδικα

Προκειμένου να εξεταστεί ο κώδικας, αρκεί να εκτελεστεί οποιοδήποτε από τα εκτελέσιμα avldate, avltemp, hash, καθώς και project. Τα πρώτα τρία οδηγούν απευθείας στα αντίστοιχα submenus (date sorted AVL tree, temperature sorted AVL tree, Chain Hashing), ενώ το τελευταίο οδηγεί στο κεντρικό menu, απ' όπου ο χρήστης μπορεί να επιλέξει σε ποιο από τα τρία submenus θα οδηγηθεί, ενώ μπορεί να επιστρέψει και να επισκεφθεί κάποιο άλλο submenu. Σημειώνεται ξανά πως στην περίπτωση αυτή, οποιαδήποτε τροποποίηση έχει πραγματοποιηθεί τόσο στις δομές AVL trees, όσο και στην δομή Chain Hashing αναιρείται, αφού κατά την είσοδο στο εκάστοτε submenu για πρώτη φορά, δημιουργείται καινούριο instance των δομών αυτών βάσει των δεδομένων εντός του αρχείου ocean.csv. Η δημιουργία των εκτελέσιμων αυτών έγινε από command line, εφόσον ο χρήστης βρίσκεται ήδη στο part 2 directory εντός του directory project, χρησιμοποιώντας τις παρακάτω εντολές:

Δημιουργία εκτελέσιμων αρχείων:

g++ basics/basics.cpp avl/shared/shared.cpp avl/date/date.cpp -o avldate*

g++ basics/basics.cpp avl/shared/shared.cpp avl/temp/temp.cpp -o avltemp *

g++ basics/basics.cpp hash/hash.cpp -o hash *

g++ basics/basics.cpp avl/shared/shared.cpp avl/date/date.cpp avl/temp/temp.cpp hash/hash.cpp menu/menu.cpp -o project

*(*) έχουν αφαιρεθεί οι χαρακτήρες " // " από την γραμμή δήλωσης της main εντός του εκάστοτε αρχείου*