Δομές Δεδομένων

2022

Εργαλεία που χρησιμοποιήθηκαν:

1. visual studio code
2. discord
3. google document
4. github

Παραδοχές:

1. Ως πίνακας εισόδου στον αλγόριθμο quicksort δόθηκε ένας πίνακας με 2 επιπλέον στοιχεία (το πρώτο -inf και το τελευταίο +inf) όπως ζητάει ο κώδικας στο βιβλίο.
2. Τα AVL δέντρα υλοποιήθηκαν ως κομβοπροσανατολισμένη δομή υστέρα από σχετικό μειλ που στάλθηκε στον κ. Βονιτσανο.
3. Κατά την χρονομέτρηση των αλγορίθμων προσθέτουμε ένα δευτερόλεπτο καθυστέρηση αμέσως μετά την εκτέλεση του αλγορίθμου το οποίο αφαιρείται από -το τελικό αποτέλεσμα της χρονομέτρησης. Αυτό έγινε διότι χωρίς αυτήν την προσθήκη το αποτέλεσμα συχνά ήταν μηδενικό δηλαδή υπήρχε σφάλμα στην χρονομέτρηση.
4. Το αρχείο sort\_dates.c εκτελείται πριν την εκτέλεση των υπολοίπων αρχείων του part 1 και του part 2 ώστε να παραχθεί το αρχείο ocean\_s.csv το οποίο χρησιμοποιεί τον quicksort για να ταξινομήσει το ocean.csv με βάση την ημερομηνία.
5. Το αρχείο sorted\_unique.c εκτελείται μετά την εκτέλεση του sort\_dates.c και πριν την εκτέλεση των υπολοίπων αρχείων του part 2 ώστε να παραχθεί το αρχείο unique.csv το οποίο χρησιμοποιείται για την αφαίρεση ημερομηνιών που εμφανίζονται περισσότερες από μια φορά στο ocean\_s.csv.

**Part I**

**0) Αρχείο: basic\_functions.h**

Συναρτήσεις για την επεξεργασία του αρχείου ocean\_s.csv

0.1) char\*\*\* csv\_import()

Συνάρτηση για την μετατροπή του αρχείου ocean.csv σε δυναμικό 3d πίνακα χαρακτήρων.

Δημιουργία stream εισόδου του αρχείου ώστε να διαβάσουμε το αρχείο γραμμή-γραμμή μέσω της συνάρτησης fgets(). Έτσι μεταφέρουμε κάθε γραμμή του .csv αναθέτοντας σε κάθε στήλη τις τιμές που διαχωρίζονται με κόμματα μέσω της συνάρτησης strtok().

0.2) void export\_csv(char\*\*\* arr, char\* fname)

Συνάρτηση για την μετατροπή και εξαγωγή του πίνακα χαρακτήρων σε αρχείο .csv.

Δημιουργία stream εξόδου του αρχείου ώστε να γράψουμε το αρχείο γραμμή-γραμμή μέσω της συνάρτησης fprintf(). Έτσι μεταφέρουμε κάθε γραμμή του πίνακα στο αρχείο εξόδου χωρίζοντας με κόμματα τις τιμές των στηλών του.

0.3) double\*\* converter2double(char\*\*\* arr2d)

Συνάρτηση για την μετατροπή ενός δυναμικού 3d πίνακα χαρακτήρων σε 2d πίνακα αριθμών κινητής υποδιαστολής διπλής ακρίβειας.

Δημιουργία κενού double 2d πίνακα ώστε να αντιγράψουμε τον πίνακα χαρακτήρων σε αυτόν. Δημιουργούμε τρεις 2d πίνακες χαρακτήρων (month, day, year) ώστε να χωρίσουμε τις ημερομηνίες σε κομμάτια και να αγνοήσουμε τις καθέτους ώστε να μπορούν να μετατραπούν σε double. Ύστερα, συνενώνουμε τους πίνακες που δημιουργήσαμε στον αρχικό πίνακα αλλάζοντας το format των ημερομηνιών σε yyyymmdd ώστε να είναι δυνατή η σύγκριση μεταξύ τους. Τέλος, μετατρέπουμε τις συμβολοσειρές του αρχικού πίνακα σε double και τις βάζουμε στις αντίστοιχες θέσεις του double πίνακα.

0.4) char\*\*\* converter2str(double\*\* arr2d)

Συνάρτηση για την μετατροπή ενός δυναμικού double 2d πίνακα σε 3d πίνακα χαρακτήρων.

Δημιουργία κενού δυναμικού 3d πίνακα χαρακτήρων στον οποίο αντιγράφουμε τον πίνακα double αφού έχουμε μετατρέψει τις τιμές του σε συμβολοσειρές με την χρήση της sprintf().

Έπειτα για την σωστή αποθήκευση της ημερομηνίας σε format mm/dd/yyyy δημιουργούμε 2d πίνακα χαρακτήρων στον οποίο αποθηκεύουμε με τη σωστή σειρά τους χαρακτήρες των ημερομηνιών του 3d πίνακα χαρακτήρων που δημιουργήσαμε προσθέτοντας τις καθέτους στις κατάλληλες θέσεις.

Την τελική μορφή του 2d πίνακα χαρακτήρων που περιέχει τις ημερομηνίες στην μορφή mm/dd/yyyy τις αντιγράφουμε και τις αντικαθιστούμε με τις προηγούμενες που υπήρχαν στο 3d πίνακα.

0.5) void date\_sorting(double\*\* arr, int column)

Συνάρτηση για την ταξινόμηση των ημερομηνιών που έχουν ίδια θερμοκρασία ή φώσφορο κατά αύξουσα σειρά.

Δημιουργία 1d int πίνακα μεγέθους όσο ο αριθμός των γραμμών και αρχικοποίηση όλων των θέσεων με την τιμή 0. Στον πίνακα καταχωρείται στην κάθε θέση του είτε ο αριθμός 1 που δηλώνει την αρχική θέση εμφάνισης της επαναλαμβανόμενης τιμής μιας θερμοκρασίας/φωσφόρου, είτε ο αριθμός 2 που δηλώνει την τελευταία θέση εμφάνισης της τιμής αυτής. Σε κάθε άλλη περίπτωση παραμένει η τιμή 0.

π.χ. (έστω ότι οι θερμοκρασίες έχουν μονοσήμαντη απεικόνιση σε ακεραίους)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| temperatures[ ] | 2 | 6 | 4 | 5 | 5 | 5 | 5 | 2 | 7 | 7 | 8 | 8 |
| pos[ ] | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 2 | 1 | 0 |

Αφού έχουμε καταχωρήσει τις αρχικές και τελικές θέσεις εμφάνισης των επαναλαμβανόμενων τιμών της θερμοκρασίας/φωσφόρου ταξινομούμε τις αντίστοιχες γραμμές του πίνακα βάσει των ημερομηνιών που βρίσκονται ανάμεσα σε αυτές τις θέσεις τους κατά αύξουσα σειρά. Ο αλγόριθμος που χρησιμοποιούμε για την ταξινόμηση αυτή χρησιμοποιείται για μικρό πλήθος στοιχείων και η πολυπλοκότητα του δεν είναι σημαντική.

0.6) int get\_date()

Συνάρτηση για την εισαγωγή ημερομηνίας από τον χρήστη σε αυστηρό format mm/dd/yyyy.

Η ημερομηνία που δόθηκε από τον χρήστη καταχωρείται σε τρεις πίνακες χαρακτήρων (day, month, year) όπου στον καθένα αποθηκεύεται το αντίστοιχο κομμάτι της ημερομηνίας. Έπειτα συνενώνουμε τους πίνακες σε μία συμβολοσειρά και την αναθέτουμε σε μεταβλητή int αγνοώντας τις καθέτους.

**1) Αρχείο: 1.c**

a) double\*\* insertionsort(double\*\* arr, int column):

ορίσματα συνάρτησης:

* πίνακας προς ταξινόμηση τύπου double
* ακέραια μεταβλητή column η οποία δηλώνει την στήλη του πίνακα βάσει της οποίας θα γίνει η ταξινόμηση

υλοποίηση αλγορίθμου insertion sort:

Αρχικά αντιγράφουμε τον πίνακα ώστε ο αρχικός πίνακας να μην αλλοιωθεί. Έπειτα δημιουργούμε 1d πίνακα που λειτουργεί ως κλειδί και αποθηκεύει ολόκληρη τη γραμμή από τον πίνακα του στοιχείου που διαχωρίζει το ταξινομημένο από το αταξινόμητο μέρος και συγκρίνεται με τα στοιχεία του αταξινόμητου.

Έπειτα συγκρίνουμε το κλειδί με το προηγούμενο στοιχείο του και αν το κλειδί είναι μικρότερό του το τοποθετεί στο ταξινομημένο μέρος και κάνει διαδοχικές συγκρίσεις με τα προηγούμενα στοιχεία για να βρει την κατάλληλη θέση του.

Όταν τελειώσει αυτή η διαδικασία γίνεται κλειδί το επόμενο στοιχείο και ακολουθείται η ίδια διαδικασία. Στο τέλος επιστρέφεται το αντίγραφο του πίνακα που δόθηκε ως όρισμα ταξινομημένο στη στήλη που επιλέχθηκε.

b) void quicksort(double\*\* arr, int first,int last):

ορίσματα συνάρτησης:

* πίνακας προς ταξινόμηση τύπου double (έχει 2 επιπλέον γραμμές: την πρώτη arr[0] που παίρνει την ελάχιστη τιμή του πίνακα και την τελευταία arr[ROWS+1] που παίρνει την μέγιστη τιμή του πίνακα
* first->το σημείο του πίνακα από το οποίο αρχίζει η ταξινόμηση
* last->το σημείο του πίνακα στο οποίο τελειώνει η ταξινόμηση

υλοποίηση αλγορίθμου quick sort:

Καταρχάς, θέτουμε το πρώτο στοιχείο της περιοχής που ταξινομούμε ως pivot και ύστερα αρχικοποιούμε τους δείκτες right και left ώστε να δείχνουν στο τέλος και στην αρχή αντιστοίχως.

Μετά, αυξάνουμε το left μέχρι το στοιχείο στο οποίο δείχνει να είναι μεγαλύτερο του pivot και μειώνουμε το right μέχρι το στοιχείο στο οποίο δείχνει να είναι μικρότερο ή ίσο του pivot και στο τέλος αυτής της διαδικασίας εναλλάσσουμε τα left και right.

Αυτή η διαδικασία επαναλαμβάνεται μέχρι το right να γίνει μικρότερο του left. Ύστερα, εναλλάσσουμε το pivot με το right ώστε το pivot να τοποθετηθεί στην σωστή τελική θέση του στον πίνακα. Τέλος, καλούμε αναδρομικά τον αλγόριθμο στα δύο μισά του πίνακα που χωρίζονται από το pivot.

c) int main():

insertion sort:

Αρχικά εισάγουμε το αρχείο ocean\_s.csv με την csv\_import η οποία επιστρέφει έναν τρισδιάστατο πίνακα χαρακτήρων. Ύστερα τον μετατρέπουμε σε μορφή double με την converter2double. Μετράμε την χρονική στιγμή πριν την έναρξη του αλγορίθμου, καλούμε τον αλγόριθμο και τέλος εκτυπώνουμε τον χρόνο που χρειάστηκε για να εκτελεστεί.

Για την αποφυγή σφάλματος από την clock προσθέτουμε την εντολή sleep(1) η οποία προσθέτει μια χρονική μονάδα και έπειτα την αφαιρούμε.

Τέλος, μετατρέπουμε τον ταξινομημένο πίνακα double σε πίνακα συμβολοσειρών και τον εξάγουμε ως αρχείο σε μορφή csv.

quick sort:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort με μόνη διαφορά ότι στον πίνακα που παίρνουμε μέσω της csv\_import και έχει μετατραπεί σε double απο την converter2double ολισθαίνουμε κατά ένα τις γραμμές του πίνακα ώστε να προσθέσουμε στην αρχική θέση μία τιμή που αντιπροσωπεύει το -inf και αντίστοιχα στο τέλος που αντιπροσωπεύει το +inf για τις ανάγκες του αλγορίθμου όπως υποδεικνύεται στο βιβλίο. Αυτές οι γραμμές αφαιρούνται μετά την ταξινόμηση.



Συμπέρασμα: ο insertion sort είναι πιο αργός από τον quicksort για αυτό το μέγεθος του αρχείου.

**2. Αρχείο: 2.c**

a) void heapify(double\*\* arr, int length, int parent):

ορίσματα συνάρτησης:

* πίνακας προς ταξινόμηση τύπου double
* ακέραια μεταβλητή length που καθορίζει το πλήθος στοιχείων του σωρού
* ακέραια μεταβλητή parent που δηλώνει το index του στοιχείου που θα συγκριθεί με τα παιδιά του.

υλοποίηση αλγορίθμου heapify:

Για την υλοποίησή του αναθέτουμε σε τρεις int μεταβλητές τα index του πατέρα και των δύο παιδιών. Αν η θέση του αριστερού παιδιού είναι μικρότερη από το μέγεθος του σωρού και το αριστερό παιδί έχει μεγαλύτερη τιμή από τον πατέρα τότε κανει ανάθεση στην τιμή που έχει το index του μεγαλύτερου στοιχείου. Αντίστοιχα λειτουργεί και για το δεξιό παιδί.

Αν μετά από αυτό τον έλεγχο των τιμών των δύο παιδιών το index του parent είναι διαφορετικό από αυτό του big τότε κάνει swap. Τέλος κάνει αναδρομική κλήση θέτοντας ως parent τη μεταβλητή big.

b) void heapSort(double\*\* arr, int length):

ορίσματα συνάρτησης:

* πίνακας προς ταξινόμηση τύπου double
* ακέραια μεταβλητή length που καθορίζει το πλήθος στοιχείων του σωρού

υλοποίηση αλγορίθμου heap sort:

Αρχικά με ένα βρόγχο for του οποίου ο μετρητής αρχίζει από τη μέση του σωρού και σταδιακά μειώνεται κατά 1 μέχρι να γίνει μικρότερο από 0 καλεί την συνάρτηση heapify ώστε να δημιουργήσει τη δομή σωρού.

Μετά προσπαθούμε με διαδοχικές αφαιρέσεις του μέγιστου στοιχείου του σωρού να χωρίσουμε τον πίνακα σε δυο κομμάτια: το πρώτο με την δομή του σωρού και το δεύτερο με τον ταξινομημένο πίνακα που περιέχει όσα στοιχεία έχουμε αφαιρέσει από τον σωρό με κατάλληλη σειρά.

Έτσι, με ένα βρόγχο for μεταφέρουμε το μέγιστο στοιχείο του σωρού στην αρχή της ταξινομημένης λίστας η οποία βρίσκεται στο δεύτερο μισό του πίνακα μειώνοντας το μήκος του σωρού και καλούμε την συνάρτηση heapify για την διόρθωση της δομής του σωρού. Η διαδικασία αυτή τελειώνει όταν εξαντληθούν τα στοιχεία της δομής του σωρού και έχουν μεταφερθεί με την σωστή σειρά στην ταξινομημένη λίστα.

c) double\*\* countingSort(double\*\* arr):

ορίσματα συνάρτησης:

* πίνακας προς ταξινόμηση τύπου double

υλοποίηση αλγορίθμου counting sort:

Δημιουργούμε έναν πίνακα double (arr2) στον οποίο αρχικά αντιγράφουμε όλα τα στοιχεία της δεύτερης στήλης του πίνακα που δόθηκε ως όρισμα της συνάρτησης πολλαπλασιασμένα με το 100 διότι ο αλγόριθμος κάνει ταξινόμηση ακεραίων και μετά αντιγράφουμε όλες τις υπόλοιπες στήλες.

Έπειτα δημιουργούμε ένα ακόμα πίνακα (arr\_b) που θα είναι και ο τελικός ταξινομημένος πίνακας και αντιγράφουμε σε αυτόν τον πίνακα που δόθηκε ως όρισμα. Ο τρίτος 1d πίνακας (arr\_c) αρχικοποιείται με 0 και έχει μέγεθος ίσο με τιμή (#define RANGE 350) η οποία είναι μεγαλύτερη από τις τιμές phosphate πολλαπλασιασμένες με το 100.

Μετά αποθηκεύεται στις θέσεις του arr\_c ο αριθμός των εμφανίσεων κάθε τιμής φωσφόρου στο αντίστοιχο index. Ύστερα, αποθηκεύουμε σε κάθε θέση του arr\_c τον αριθμό εμφανίσεων τιμών φωσφόρου μικρότερων ή ίσων από τον αριθμό της θέσης αυτής.

Η ταξινόμηση τελειώνει με βρόγχο for όπου κατασκευάζουμε τον πίνακα arr\_b βάζοντας τα στοιχεία του arr2 (ξεκινώντας από το τελευταίο) στην τελική τους θέση στον arr\_b με βάση πόσα στοιχεία είναι μικρότερα ή ίσα από αυτά αξιοποιώντας τον πίνακα arr\_c. Ύστερα αφαιρούμε ένα από την κατάλληλη θέση του arr\_c για να φανεί ότι το στοιχείο ταξινομήθηκε και πήρε την τελική του θέση στον arr\_b.

Ο πίνακας επιστρέφεται στην main αφού διαιρέσουμε τα στοιχεία της δεύτερης στήλης με 100 για να τα επαναφέρουμε στις αρχικές τους τιμές.

d) int main():

heap sort:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.

counting sort:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.



Συμπέρασμα: ο counting sort είναι πιο αργός από τον heapsort για αυτό το μέγεθος του αρχείου.

**3. Αρχείο: 3.c**

a) int binarySearch(int\* arr, int key)

ορίσματα συνάρτησης:

* πίνακας ακέραιων προς ταξινόμηση
* η ημερομηνία που ψάχνουμε

υλοποίηση αλγορίθμου binary search:

Αρχικά ορίζουμε το μέρος του πίνακα στο οποίο ψάχνουμε το οποίο αρχικά είναι όλος ο πίνακας. Μετά αποθηκεύουμε στην μεταβλητή next την θέση του μεσαίου στοιχείου.

Ύστερα όσο το στοιχείο που βρίσκεται στη θέση next είναι διαφορετικό από το στοιχείο που ψάχνουμε και το αριστερό μέρος του μέρους που ψάχνουμε δεν έχει ξεπεράσει το δεξί μέρος τότε υποδιπλασιάζουμε το κομμάτι του πίνακα στο οποίο ψάχνουμε βάσει του αν το στοιχείο που βρίσκεται στη θέση next είναι μεγαλύτερο ή μικρότερο από το στοιχείο που ψάχνουμε και ενημερώνουμε το next.

Στο τέλος αυτής της επαναληπτικής διαδικασίας αν έχουμε βρει το στοιχείο το επιστρέφουμε αλλιώς επιστρέφουμε -1.

b) int interpolationSearch(int\* arr, int key):

ορίσματα συνάρτησης:

* πίνακας ακέραιων προς ταξινόμηση
* η ημερομηνία που ψάχνουμε

υλοποίηση αλγορίθμου interpolation search:

Αρχικά ορίζουμε το μέρος του πίνακα στο οποίο ψάχνουμε το οποίο αρχικά είναι όλος ο πίνακας.

Ύστερα όσο το στοιχείο που βρίσκεται στη θέση next είναι διαφορετικό από το στοιχείο που ψάχνουμε και το αριστερό μέρος του μέρους που ψάχνουμε δεν έχει ξεπεράσει το δεξί τότε

1) αποθηκεύουμε στην μεταβλητή next την θέση ενός ειδικά επιλεγμένου στοιχείου,

2) μικραίνουμε το κομμάτι του πίνακα στο οποίο ψάχνουμε βάσει του αν το στοιχείο που βρίσκεται στη θέση next είναι μεγαλύτερο ή μικρότερο από το στοιχείο που ψάχνουμε και ενημερώνουμε το next.

Στο τέλος αυτής της επαναληπτικής διαδικασίας αν έχουμε βρει το στοιχείο το επιστρέφουμε αλλιώς επιστρέφουμε -1.

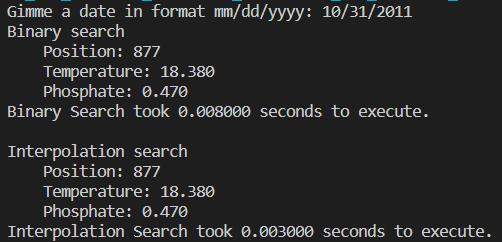
c) int main():

binary search:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.

interpolation search:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.



Συμπέρασμα: Παρατηρούμε ότι ο interpolation search είναι πιο γρήγορος από τον binary search. Ο binary search δουλεύει πιο αποδοτικά σε κατανομές uniform ενώ ο interpolation search δουλεύει πιο αποδοτικά σε κατανομές smooth. Τότε οι αλγόριθμοι έχουν πολυπλοκότητα μέσης περίπτωσης Ο(log(logN)).

Έτσι επειδή η κατανομή των ημερομηνιών δεν είναι uniform αν και είναι uniform κατά διαστήματα, ο interpolation search είναι πιο αποδοτικός έχοντας πολυπλοκότητα χειρότερης περίπτωσης Ο(Ν^½) αντί Ο(Ν) που έχει ο binary search.

Σημείωση: Λόγω του μεγέθους του δείγματος μερικές φορές ο binary search εμφανίζεται να είναι πιο γρήγορος. Ίσως αυτό το αποτέλεσμα όμως να οφείλεται στην μερική ομοιότητα της κατανομής των ημερομηνιών με την κατανομή uniform.

**4. Αρχείο: 4.c**

a) int BIS(int\* arr, int x)

ορίσματα συνάρτησης:

* πίνακας ακέραιων προς ταξινόμηση
* η ημερομηνία που ψάχνουμε

υλοποίηση αλγορίθμου binary interpolation search:

Αρχικά ορίζουμε το μέρος του πίνακα στο οποίο ψάχνουμε το οποίο αρχικά είναι όλος ο πίνακας ορίζοντας το size και επίσης υπολογίζουμε καταλληλά το next.

Ύστερα με ένα βρόχο while:

1) υπολογίζουμε το size και αν αυτό είναι μικρότερο ή ίσο από 3 τότε εκτελούμε γραμμική αναζήτηση στο τέλος της οποίας αν το στοιχείο δεν βρεθεί επιστρέφουμε -1 αλλιώς επιστρέφουμε το στοιχείο.

2) βρίσκουμε το αριστερό και το δεξί μέρος του κομματιού στο οποίο θα περιορίσουμε καταλληλά την αναζήτηση μας βασιζόμενοι στο next και τη σχετική του θέση με το στοιχείο που ψάχνουμε. Για την εύρεση των άκρων του νέου διαστήματος χρησιμοποιούμε γραμμική αύξηση του μήκους του βήματος το οποίο χρησιμοποιούμε για να προσδιορίσουμε το διάστημα.

3) υπολογίζουμε το next λαμβάνοντας υπόψιν τις καινούργιες τιμές του αριστερού και του δεξιού κομματιού του πίνακα στο οποίο ψάχνουμε.

Στο τέλος αυτής της επαναληπτικής διαδικασίας αν έχουμε βρει το στοιχείο το επιστρέφουμε αλλιώς επιστρέφουμε -1.

b) int BISalt(int\* arr, int x)

ορίσματα συνάρτησης:

* πίνακας ακέραιων προς ταξινόμηση
* η ημερομηνία που ψάχνουμε

υλοποίηση αλγορίθμου alternative binary interpolation search:

Αρχικά ορίζουμε το μέρος του πίνακα στο οποίο ψάχνουμε το οποίο αρχικά είναι όλος ο πίνακας ορίζοντας το size και επίσης υπολογίζουμε καταλληλά το next.

Ύστερα με ένα βρόχο while:

1) υπολογίζουμε το size και αν αυτό είναι μικρότερο ή ίσο από 3 τότε εκτελούμε γραμμική αναζήτηση στο τέλος της οποίας αν το στοιχείο δεν βρεθεί επιστρέφουμε -1 αλλιώς επιστρέφουμε το στοιχείο.

2) βρίσκουμε το αριστερό και το δεξί μέρος του κομματιού στο οποίο θα περιορίσουμε καταλληλά την αναζήτηση μας βασιζόμενοι στο next και τη σχετική του θέση με το στοιχείο που ψάχνουμε. Για την εύρεση των ακρών του νέου διαστήματος χρησιμοποιούμε εκθετική αύξηση του μήκους του βήματος το οποίο χρησιμοποιούμε για να προσδιορίσουμε το διάστημα.

3) υπολογίζουμε το next λαμβάνοντας υπόψιν τις καινούργιες τιμές του αριστερού και του δεξιού κομματιού του πίνακα στο οποίο ψάχνουμε.

Στο τέλος αυτής της επαναληπτικής διαδικασίας αν έχουμε βρει το στοιχείο το επιστρέφουμε αλλιώς επιστρέφουμε -1.

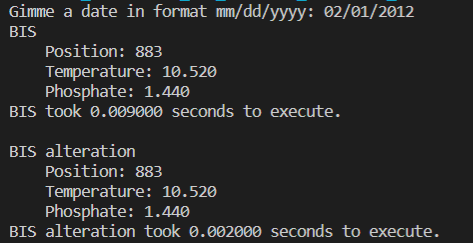
c) int main():

binary search:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.

interpolation search:

Ακολουθείται η ίδια διαδικασία όπως και στον insertion sort.



Συμπέρασμα: Παρατηρούμε οτι ο BIS alternative ειναι πιο γρήγορος απο τον BIS λόγω του ότι έχει καλύτερο χρόνο χειρότερης περίπτωσης.

**5. Αρχείο: sort\_dates.c**

Πρόγραμμα με το οποίο ταξινομούμε τις ημερομηνίες κατά αύξουσα σειρά. Η ταξινόμηση γίνεται με τη χρήση του αλγορίθμου insertionsort. Παράγεται το αρχείο ocean\_s.csv με ταξινομημένες ημερομηνίες. Το αρχείο αυτό (sort\_dates.c) πρέπει να εκτελείται πριν την εκτέλεση των υπόλοιπων αρχείων του part 1 στην περιπτωση που λείπει το αρχείο ocean\_s.csv από το directory του part 1.

**Part II**

**0. Αρχείο: basic\_functions.h**

Το αρχείο αυτό είναι το ίδιο με το αντίστοιχου του Part I και έχει χρησιμοποιηθεί για την επεξεργασία του αρχείου unique.csv.

**1. Αρχείο: 1.h**

Στο αρχείο αυτό γίνεται η υλοποίηση του AVL δέντρου.

Δημιουργείται εγγραφή για κάθε στην οποία καταχωρούνται τα εξής:

* η ημερομηνία
* η θερμοκρασία
* ο δεξιός κόμβος
* ο αριστερός κόμβος
* το ύψος του κόμβου

struct Komvos\* newKomvos(int date, double temperature)

Συνάρτηση για την δημιουργία και νέου κόμβου. Ορίζεται ως εξής:

* ημερομηνία: date
* η θερμοκρασία: temperature
* ο δεξιός κόμβος: null
* ο αριστερός κόμβος: null
* το ύψος του κόμβου: 1

struct Komvos\* minDate(struct Komvos \*komvos)

Αναδρομική συνάρτηση για την εύρεση της ελάχιστης ημερομηνίας η οποία θα βρίσκεται σε αριστερό παιδί κόμβου το οποίο θα είναι φύλλο.

int ypsos(struct Komvos \*komv)

Συνάρτηση που επιστρέφει το ύψος του κόμβου, αν δεν υπάρχει επιστρέφει 0.

int balance(struct Komvos \*komv)

Συνάρτηση που επιστρέφει την ισορροπία του κόμβου.

(ισορροπία=ύψος αριστερού κόμβου-ύψος δεξιού)

struct Komvos\* lRotate(struct Komvos \*u)

Συνάρτηση για αριστερή απλή περιστροφή. O κόμβος γίνεται αριστερό παιδί του δεξιού του παιδιού και το αριστερό υποδέντρο του πρώην δεξιού του παιδιού γίνεται δεξιό του παιδί. Το ύψος του κόμβου στον οποίο έγινε η περιστροφή παίρνει τη μέγιστη τιμή ανάμεσα στο ύψος του δεξιού ή του αριστερού παιδιού του +1.

Αντίστοιχα υπολογίζεται και το ύψος του κόμβου που πλέον είναι πατέρας του κόμβου που δόθηκε ως όρισμα στην συνάρτηση.

struct Komvos\* rRotate(struct Komvos \*u)

Συνάρτηση για δεξιά απλή περιστροφή. Η περιστροφή γίνεται με την ίδια λογική όπως στην αριστερή, στην περίπτωση αυτή το παιδί του κόμβου που δόθηκε ως όρισμα είναι αριστερά και το υποδεντρο δεξιά του παιδιού.

struct Komvos\* search(struct Komvos\* riza, int date)

Η αναδρομική αυτή συνάρτηση κάνει αναζήτηση βάση ημερομηνίας και επιστρέφει τον κόμβο στον οποίο βρέθηκε ή τον κενό κόμβο. Για να κάνουμε την αναζήτηση εκμεταλλευόμαστε την ιδιότητα του δέντρου στο δεξί παιδί να βρίσκεται η μεγαλύτερη τιμή και στο αριστερό η μικρότερη.

Ξεκινώντας από τη ρίζα κάνουμε συγκρίσεις και ανάλογα αν η ημερομηνία που ψάχνουμε είναι μικρότερη ή μεγαλύτερη του κόμβου που βρισκόμαστε καλούμε αναδρομικά τη συνάρτηση με κόμβο το μικρότερο ή μεγαλύτερο παιδί αντίστοιχα.

Η συνάρτηση επιστρέφει τον κόμβο αν βρεθεί τιμή ίση με αυτή που αναζητούμε αλλιώς επιστρέφεται ο κόμβος που είναι κενός.

struct Komvos\* insertKomvos(struct Komvos \*komvos, int date, double temperature)

Με τη συνάρτηση αυτή εισάγουμε νέο κόμβο. Για να γίνει αυτό βρίσκουμε τη σωστή θέση που θα τοποθετηθεί με βάση την ημερομηνία.

Αν δεν υπάρχει άλλος κόμβος τότε αυτός γίνεται ρίζα αλλιώς καλείται αναδρομικά η συνάρτηση αφού πρώτα κάνουμε σύγκριση των ημερομηνιών ώστε να ξέρουμε αν θα συνεχίσει δεξιά ή αριστερά του τρέχοντος κόμβου μέχρι να βρεθεί η θέση που θα τοποθετηθεί.

Έπειτα καθορίζουμε το ύψος του κάθε κόμβου και καλούμε την συνάρτηση που δίνει την ισορροπία του κάθε κόμβου. Ανάλογα του αποτελέσματος κάνουμε τις κατάλληλες περιστροφές ώστε να ισορροπήσει το δέντρο.

struct Komvos\* deleteKomvos(struct Komvos \*riza, int date)

Συνάρτηση για διαγραφή κόμβου σύμφωνα με την ημερομηνία που δίνεται ως όρισμα.

Ξεκινώντας από την ρίζα κάνουμε συγκρίσεις με τα παιδιά των κόμβων μέχρι να βρεθεί η ημερομηνία που δόθηκε. Όταν ο κόμβος βρεθεί αν δεν έχει παιδιά διαγράφεται και αν έχει μόνο ένα γίνεται αλλαγή και μετά διαγραφή. Αν ο κόμβος έχει παιδία τότε γίνεται εναλλαγή με την ελάχιστη ημερομηνία και καλείται ξανά η συνάρτηση για το δεξί παιδί του κόμβου.

Στο τέλος αφού γίνει η διαγραφή κάνουμε ενημέρωση της ισορροπίας του δέντρου.

char\* date\_str\_from\_int(int date)

Η συνάρτηση αυτή μετατρέπει μια ημερομηνία από format ddmmyyyy σε format yyyy/dd/mm. H ημερομηνία δίνεται ως όρισμα σε μορφή ακεραίου και αντιγράφεται σε ένα 1d πίνακα χαρακτήρων (datastr) με κατάλληλη συνάρτηση.

Δημιουργείται επίσης ένας πίνακας ιδίων προδιαγραφών final\_date και αποθηκεύουμε σε αυτόν τα περιεχόμενα του datastr με τη σειρά που πρέπει συν κάποιες καθέτους ώστε να υπακούν στο format yyyy/dd/mm.

void pr\_tree(struct Komvos \*riza)

Συνάρτηση για την εκτύπωση του AVL δέντρου με ενδο-διατεταγμένη διάσχιση και τις κεφαλίδες.

void menu1(struct Komvos\* riza)

Η συνάρτηση menu1 δίνει τις επιλογές:

1) presentation:

Η λειτουργία αυτή γίνεται με την συνάρτηση pr\_tree.

2) search:

Δίνεται ημερομηνία από τον χρήστη με την συνάρτηση get\_date και καλείται η συνάρτηση search με όρισμα την ημερομηνία που δόθηκε.

3) modification:

Με την επιλογή αυτή δίνεται ημερομηνία από τον χρήστη με την συνάρτηση get\_date και καλείται η συνάρτηση search με όρισμα την ημερομηνία που δόθηκε. Όταν βρεθεί η ημερομηνία κάνουμε αλλαγή στο πεδίο temperature της εγγραφής.

4) deletion:

Δίνεται ημερομηνία απο τον χρήστη με την συνάρτηση get\_date και καλείται η συνάρτηση deleteKomvos με όρισμα την ημερομηνία που δόθηκε.

5) exit:

Έξοδος από το πρόγραμμα.

int main1()

Χρησιμοποιείται η συνάρτηση csv\_import() για εισαγωγή του αρχείου. Ακολούθως η συνάρτηση converter2double(arr\_imp) για μετατροπή του πίνακα χαρακτήρων σε double και εισαγωγή όλων των τιμών του πίνακα από την πρώτη και τη δεύτερη στήλη με την συνάρτηση insertKomvos στο AVL δέντρο.

Τέλος καλείται η menu1(riza) με όρισμα τη ρίζα του δέντρου για να εμφανιστεί το menu για το δέντρο.

**2. Αρχείο: 2.h**

Το αρχείο αυτό είναι παραλλαγή του αρχείου 1.h. Στην περίπτωση αυτή υλοποιείται το AVL δέντρο με βάση το πεδίο θερμοκρασία ενώ προηγουμένως με βάση την ημερομηνία. Για αυτό θα αναφερθούμε μόνο στην συνάρτηση menu2() που διαφοροποιείται.

void menu2(struct Komvos2\* riza)

Η συνάρτηση menu2 δίνει τις επιλογές:

1) Find date with min temperature:

Με την κλήση της συνάρτησης minTemperature βρίσκουμε την ελάχιστη θερμοκρασία.

Εκμεταλλευόμαστε την ιδιότητα του δέντρου όπου η ελάχιστη τιμή βρίσκεται στο πιο αριστερό φύλλο.

Επομένως, βρίσκουμε την ελάχιστη θερμοκρασία καλώντας την minTemperature και την διαγράφουμε από το δέντρο αποθηκεύοντας την τιμή της.

Στην συνέχεια εκτελείται ένας βρόγχος με τον οποίο βρίσκουμε αν υπάρχουν άλλοι κόμβοι με ίδια θερμοκρασία.

Πρέπει να αναφέρουμε ότι αν υπάρχουν ημερομηνίες με ίδια θερμοκρασία τότε ξεκινώντας από τον κόμβο που βρίσκεται πιο αριστερά από όλους η επόμενη θα είναι πατέρας αυτού του κόμβου και το ίδιο ισχύει αν υπάρχει και τρίτη ίδια θερμοκρασία και ακολούθως το ίδιο.

2) Find date with max temperature:

Η εύρεση της μέγιστης τιμής γίνεται ακριβώς όπως της ελάχιστης με τη διαφοροποίηση ότι η μέγιστη βρίσκεται στον πιο δεξιό κόμβο του δέντρου.

3) exit:

Έξοδος από το πρόγραμμα.

int main2()

Η main σε αυτό το αρχείο είναι ίδια με του αρχείου 1.h και έχει προστεθεί η συνάρτηση pr\_tree με την οποία εκτυπώνεται το δέντρο.

**3. Αρχείο: 3.h**

Στο αρχείο αυτό γίνεται η μετατροπή του αρχείου unique.csv σε υλοποίηση hashing με αλυσίδες.

Δημιουργείται εγγραφή για την κάθε γραμμή του αρχείου unique.csv στην οποία καταχωρούνται τα εξής:

* η ημερομηνία
* η θερμοκρασία
* ο επόμενος κόμβος

int int\_from\_str(char\* date)

Με τη συνάρτηση αυτή βρίσκουμε το άθροισμα των ascii αναπαραστάσεων του κάθε χαρακτήρα της ημερομηνία που δόθηκε σε τύπο char και αγνοούμε το ascii των 2 καθέτων.

Komvos3\* insert(Komvos3\* array, int indx, int date, double temp)

Η συνάρτηση insert δέχεται ως ορίσματα τον πίνακα που έχει αποθηκευμένες τις εγγραφές, την κατάλληλη θέση εισαγωγής της νέας εγγραφής στο hashing table, την ημερομηνία και τη θερμοκρασία.

Δημιουργούμε δύο δείκτες σε κόμβους και αναθέτουμε στον πρώτο έναν κόμβο με τις τιμές που δώσαμε ως ορίσματα. Ελέγχουμε αν η θέση του πίνακα στην οποία θέλουμε να εισάγουμε τον κόμβο έχει και άλλη τιμή και αν δεν έχει καταχωρείται αλλιώς εισάγεται στο τέλος της λίστας.

Komvos3\* init(Komvos3\* array, int array\_size)

Εισάγουμε το αρχείο unique.csv με την csv\_import και ακολούθως το μετατρέπουμε σε πίνακα τύπου double. Κάθε ζευγάρι τιμών της πρώτης και της δεύτερης στήλης αποθηκεύεται ως κόμβος μέσω της συνάρτησης insert στον πίνακα που δημιουργείται μέσω της συνάρτησης κατακερματισμού. Η θέση κάθε εγγραφής στο hashing table υποδεικνύεται από το αποτέλεσμα που δίνει το υπόλοιπο της διαίρεσης του αθροίσματος των κωδικών ascii της ημερομηνίες με τον αριθμό 51.

int deletion(Komvos3\* array, int array\_size, int date)

Η συνάρτηση υλοποιεί τη διαγραφή ενός κόμβου αφού πρώτα δοθεί από τον χρήστη η ημερομηνία της εγγραφής.

Βρίσκουμε την θέση του hash table στην οποία υπάρχει ο κόμβος με την ζητούμενη ημερομηνία υπολογίζοντας το υπόλοιπο της διαίρεσης του αθροίσματος των ascii αναπαραστάσεων των αριθμών της ημερομηνίας με το μέγεθος του hash table.

Αν στη θέση βρίσκεται μόνο ένας κόμβος κάνουμε απλά την διαγραφή, αλλιώς κάνουμε ψάξιμο στη λίστα ώστε να τη διαγράψουμε.

Η συνάρτηση επιστρέφει έναν αριθμό μοναδικό για κάθε περίπτωση και η menu3 εκτυπώνει διαφορετικό μήνυμα για κάθε περίπτωση.

0 ->Date was found and Deleted!

-1 ->Date not found cause Table is emtpy!

-2 ->Date was not found in Hasharray!

void pr\_chain(Komvos3\* array, int array\_size)

Η συνάρτηση εκτυπώνει τον πίνακα με τις αλυσίδες. Διατρέχει όλες τις θέσεις του hash table και αν κάποια είναι άδεια εμφανίζει κατάλληλο μήνυμα. Στις υπόλοιπες θέσεις εκτυπώνει με τη βοήθεια δείκτη τύπου Komvos3 τις εγγραφές που υπάρχουν στη λίστα.

Επίσης για την καλύτερη εμφάνιση του πίνακα στην εκτύπωση υπάρχει μετρητής που αυξάνεται όσο σε κάθε θέση του πίνακα υπάρχουν άλλες εγγραφές και όταν το υπόλοιπο της διαίρεσης του με το 5 δώσει 0 τότε αλλάζει γραμμή στην εκτύπωση.

double search\_array(Komvos3\* array, int array\_size, int date)

Η συνάρτηση υλοποιεί την αναζήτηση ενός κόμβου αφού πρώτα δοθεί από τον χρήστη η ημερομηνία.

Βρίσκουμε την θέση του hash table στην οποία υπάρχει ο κόμβος με την ζητούμενη ημερομηνία υπολογίζοντας το υπόλοιπο της διαίρεσης του αθροίσματος των ascii αναπαραστάσεων των αριθμών της ημερομηνίας με το μέγεθος του hash table.

Αν στη θέση βρίσκεται μόνο ένας κόμβος τότε είναι αυτός που ψάχνουμε, αλλιώς διαπερνούμε την λίστα. Η συνάρτηση επιστρέφει έναν αριθμό μοναδικό για κάθε περίπτωση και η menu3 εκτυπώνει μήνυμα για κάθε περίπτωση.

0 ->Date found!

-1 ->Date not found!

-2 ->Date not found because Table is emtpy!

double modify\_array(Komvos3\* array, int array\_size, int date, double temp)

Η συνάρτηση υλοποιεί τη αναζήτηση ενός κόμβου για την τροποποίηση της τιμής της θερμοκρασίας αφού πρώτα δοθεί από τον χρήστη η ημερομηνία και η καινούργια θερμοκρασία.

Βρίσκουμε την θέση του hash table στην οποία υπάρχει ο κόμβος με την ζητούμενη ημερομηνία υπολογίζοντας το υπόλοιπο της διαίρεσης του αθροίσματος των ascii αναπαραστάσεων των αριθμών της ημερομηνίας με το μέγεθος του hash table.

Αν στη θέση βρίσκεται μόνο ένας κόμβος τότε είναι αυτός που ψάχνουμε και τροποποιούμε την τιμή της θερμοκρασίας του, αλλιώς διατρέχουμε την λίστα μέχρι να βρούμε τον κόμβο.

Η συνάρτηση επιστρέφει έναν αριθμό μοναδικό για κάθε περίπτωση και η menu3 εκτυπώνει μήνυμα για κάθε περίπτωση.

0 ->Date found and modified !

-1 ->Date not found!

-2 ->Date not found because Table is emtpy!

int menu3(Komvos3\* array)

Η συνάρτηση menu3 δίνει τις επιλογές:

1) presentation:

Εκτύπωση του πίνακα με την pr\_chain(array, N)

2) search:

Αναζήτηση θερμοκρασίας με βάση ημερομηνία που δίνει ο χρήστης με την συνάρτηση get\_date. Για την αναζήτηση καλούμε την συνάρτηση search\_array.

3) modification:

Τροποποίηση της θερμοκρασίας που δίνει ο χρήστης με βάση την ημερομηνία που δίνει ο χρήστης. Για την τροποποίηση καλούμε την συνάρτηση modify\_array.

4) deletion:

Διαγραφή κόμβου με βάση ημερομηνία που δίνει ο χρήστης με την συνάρτηση get\_date. Για την διαγραφή καλούμε την συνάρτηση deletion.

5) exit:

Έξοδος από το πρόγραμμα.

int main3()

Η συνάρτηση main δημιουργεί τον πίνακα κατακερματισμού με αλυσίδες μεγέθους Ν=51 οι θέσεις του οποίου αρχικοποιούνται στην τιμή null. Καλώντας την συνάρτηση init(array, Ν) βάζουμε τις δύο πρώτες στήλες του αρχείου ocen.csv στον πίνακα.

Τέλος καλείται η menu3 για να εμφανιστούν οι επιλογές που έχουμε για τον πίνακα κατακερματισμού με αλυσίδες.

**4. Αρχείο: final.c**

Τα τρία παραπάνω προγράμματα ενώνονται σε αυτό το πρόγραμμα κάνοντάς τα include. Για να επιλέξουμε μια από τις τρεις αναπαραστάσεις μέσα στη συνάρτηση main του προγράμματος υπάρχει μία μεταβλητή όπου ανάλογα με την επιλογή που κάνουμε καλείται το menu του της αναπαράστασης που επιλέξαμε.

1->AVL tree with date: Press 1

2->AVL tree with temperature: Press 2

3->HASHING CHAIN: Press 3

4->EXIT: Press 4

**5. Αρχείο: sorted\_unique.c**

Σκοπός του προγράμματος είναι να διαγραφούν οι ημερομηνίες οι οποίες εμφανίζονται στο αρχείο ocean\_s.csv(το ταξινομημένο αρχείο του ocean.csν όπου οι ημερομηνίες βρίσκονται σε αύξουσα σειρά) πάνω από μια φορά ώστε να μην υπάρχουν δυσλειτουργίες στα τρία προγράμματα που ζητούνται.

Οι συναρτήσεις που περιλαμβάνονται είναι ίδιες με αυτές που υπάρχουν στο αρχείο basic\_functions.h (εκτός από το αρχείο που χρησιμοποιεί η csv\_import() ) και για αυτό θα αναφερθούμε μόνο στην συνάρτηση main στην οποία γίνεται η διαδικασία εύρεσης των ημερομηνιών που υπάρχουν περισσότερες από μια φορά στο αρχείο ocean\_s.csv.

Στη main δημιουργούμε μια μεταβλητή last που θα κρατάει κάθε φορά την ημερομηνία και θα τη συγκρίνει με την επόμενη και άλλη k μια όπου θα δείχνει το μέγεθος του τελικού πίνακα που θα φτιάξουμε και θα αυξάνεται κάθε φορά που η τρέχουσα ημερομηνία είναι διαφορετική από την επόμενη.

Έτσι καταλήγουμε σε ένα τελικό πίνακα στον οποίο κάθε ημερομηνία εμφανίζεται μία φορά και είναι αυτός τον οποίο θα χρησιμοποιήσουμε για να υλοποιήσουμε τα τρία προγράμματα που ζητούνται.

Πηγές:

1. Βιβλίο Δομών Δεδομένων Α. Τσακαλίδη
2. <https://www.geeksforgeeks.org/heap-sort/>
3. <https://www.geeksforgeeks.org/quick-sort/?ref=lbp>
4. <https://www.geeksforgeeks.org/insertion-sort/>
5. <https://steemit.com/programming/@drifter1/programming-c-hashtables-with-chaining>
6. <https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/>
7. <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>
8. <https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm>
9. <https://www.programiz.com/dsa/avl-tree>
10. <https://www.geeksforgeeks.org/interpolation-search/>
11. <https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/>
12. <https://www.geeksforgeeks.org/interpolation-search/?ref=lbp>
13. <https://www.geeksforgeeks.org/counting-sort/>
14. <https://www.geeksforgeeks.org/relational-database-from-csv-files-in-c/>
15. <https://www.techiedelight.com/find-execution-time-c-program/>
16. <https://www.tutorialspoint.com/how-to-measure-time-taken-by-a-function-in-c>