Δομές Δεδομένων Project

Εαρινό Εξάμηνο 2022

Χαράλαμπος-Μάριος Χάλλας 1084589

Φώτιος Κίζιλης 1084588

Αχιλλέας Βιλλιώτης 1084567

Εισαγωγή:

Το αρχείο αυτό έχει ως περιεχόμενα το documentation των αρχείων πηγαίου κώδικα και την ενδεικτική επίδειξη της λειτουργίας αυτών. Έτσι κάτω από κάθε μέρος του project, βρίσκονται πρώτα το documentation και έπειτα screenshots από την λειτουργεία των προγραμμάτων.

Στο part 1 γίνεται χρήση πινάκων για την παρουσίαση των χρόνων εκτέλεσης, στο part 2 αναφέρονται ξεχωριστά τα προγράμματα αλλά στο παραδοτέο zip υπάρχουν ενοποιημένα σε ένα.

Part 1:

Documentation:

Για όλο το πρώτο μέρος του project γίνεται χρήση του projectshared.h header file. Εκεί βρίσκονται οι ορισμοί του oceanRecord struct, αλλά και του parseCSV όπως επίσης σταθερές:

- **MAGNITUDE_RUNS:** Ορίζει με πόση ακρίβεια γίνεται ο υπολογισμός των runtime.
- **MAXCHAR:** Μέγιστοι χαρακτήρες στην είσοδο.
- **BREAKPOINT:** Breakpoint χαρακτήρας για το CSV.
- **DATA_FILE_NAME:** Όνομα του αρχείου csv προς ανάγνωση.
- **oceanRecord:** Έγινε χρήση πολλών μεταβλητών αντί για array με σκοπό την αναγνωσιμότητα του κώδικα
- **parseCSV:** Χρησιμοποιώντας τις μεταβλητές buf_size και buf_used μπορούμε δυναμικά να διπλασιάζουμε την μνήμη που χρησιμοποιούμε ώστε να διαβαστεί όλο το input.csv ενώ στο τέλος μπορούμε να ξέρουμε και το μέγεθος του array για χρήση αργότερα στον κώδικα.

Για τα 1.3 και 1.4 τα oceanRecord και parseCSV δεν χρησιμοποιούνται λόγω της ανάγκης για int date.

1.1

Documentation: Το πρόγραμμα πρώτα διαβάζει το αρχείο “ocean.csv” και έπειτα οι αλγόριθμοι “Quick sort” και “Insertion sort” ταξινομούν τα δεδομένα και δημιουργούνται τα νέα csv αρχεία. Για την δημιουργία των csv αρχείων χρησιμοποιούνται διαφορετικές συναρτήσεις από αυτές που χρησιμοποιούνται για την χρονομέτρηση αλλά οι αλγόριθμοι διαφέρουν μόνο στην πρόσθετη ικανότητα να δημιουργείται ένα νέο αρχείο με τον ταξινομημένο πίνακα. Και στους 2 αλγόριθμους πριν αρχίσει να γίνεται η σύγκριση των στοιχείων , πρώτα τα μετατρέπουμε σε ακέραιους αριθμούς και όταν πραγματοποιηθεί η σύγκριση τα επαναφέρουμε σε δεκαδικούς.

insertionSort: Απλή εφαρμογή του αλγόριθμου.
quickSortWrapper: Καλείται όταν θέλουμε να χρησιμοποιήσουμε τον αλγόριθμο και είναι υπεύθυνη για την κλήση της quickSort.
quickSort: Είναι υπεύθυνη για τα αναδρομικά βήματα του αλγόριθμου, δηλαδή κάθε φορά που το array “χωρίζεται” σε καινούρια κομμάτια η quicksort καλείται πάλι με τις παραμέτρους του νέου array.
partition: Είναι υπεύθυνη για την σύγκριση των στοιχείων και την κλήση της swap. Προκειμένου να είναι πιο ευέλικτος ο αλγόριθμος η τιμή του pivot είναι μία τυχαία τιμή, διαφορετική της τελευταίας τιμής του πίνακα, με αποτέλεσμα ο αλγόριθμος σε πολλές περιπτώσεις να είναι πιο γρήγορος σε σχέση με όταν η τιμή του pivot είναι η τελευταία τιμή του array.
swap: Χρησιμοποιείται για να ανταλλάξει θέση 2 στοιχεία ενός array.

Χρονομέτρηση:
Για την χρονομέτρηση χρησιμοποιήσαμε τις συναρτήσεις insertionSortNoCSV και quickSortWrapperNoCSV οι οποίες δεν δημιουργούν νέο ταξινομημένο αρχείο κατά την διάρκεια του runtime τους.

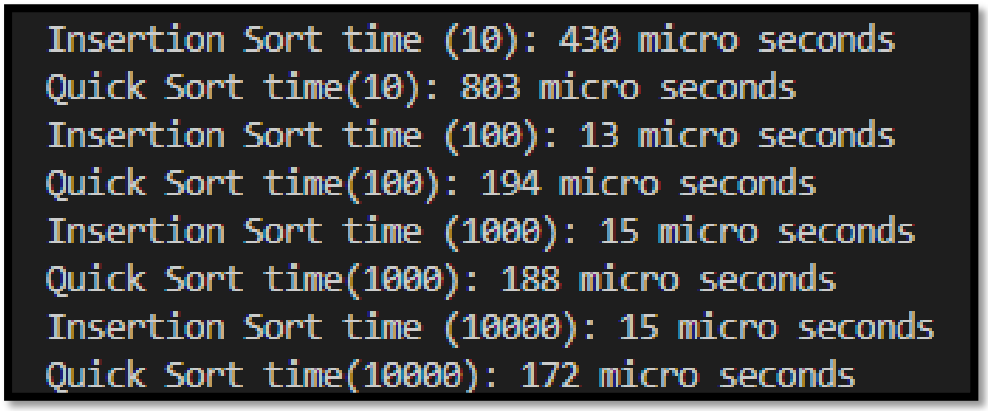
Για τον υπολογισμό των χρόνων των αλγόριθμων υπολογίζουμε το runtime τους μέσω του βασικού βρόχου της main, ο οποίος τρέχει τόσες φορές όσο η τιμή της σταθεράς **MAGNITUDE_RUNS** (ορίζεται στο **projectshared.h**) (1,2,3,… επαναλήψεις), και οι αλγόριθμοι κάθε φορά με μια μεγαλύτερη τάξη αριθμού (10, 100, 1000,…) ώστε να υπολογίζεται το πραγματικό runtime.

Παρατηρούμε:

Times Ran	Insertion Sort	Quick Sort
10	430 μs	803 μs
100	13 μs	194 μs
1000	15 μs	188 μs
10000	15 μs	172 μs

Οι χρόνοι είναι realtime και παρατηρούμε πως ο insertionSort είναι περίπου 10 φορές πιο γρήγορος από τον quickSort. (Ryzen 5 3600,16GB RAM).

Screenshots:



1.2

Documentation:

Η λογική του προγράμματος είναι αρχικά το διάβασμα του csv αρχείου και στην συνέχεια η αντιγραφή του array of struct αποτελέσματός με σκοπό την ταξινόμηση του αντίγραφου και την παραγωγή των csv αρχείων. Οι NoCSV συναρτήσεις είναι ακριβώς οι ίδιες χωρίς την δημιουργία csv.

heapSort: Απλή εφαρμογή του αλγόριθμου.

countingSort: Αρχικά γίνεται χρήση του findRangeOfPO4uMNumbers ώστε να βρεθούν οι μέγιστοι και ελάχιστοι αριθμοί των δεδομένων. Έπειτα γίνεται απλή εφαρμογή του αλγορίθμου. Η χρήση memset φαίνεται να λειτουργεί (βλ. https://stackoverflow.com/questions/17288859/using-memset-for-integer-array-in-c) επείδη έχουμε μηδενικά, όμως, υπάρχει commented out και η υλοποίηση με O(1) αρχικοποίηση πίνακα, η οποία δεν χρησιμοποιήθηκε διότι μεγάλο μέρος του πίνακα έπαιρνε τιμές στο τέλος και έτσι απλά χανόταν χρόνος. Στο τέλος, οι αριθμοί επαναφέρονται στην αρχική τους μορφή (PO4uM επαναφέρονται τα δεκαδικά).

findRangeOfPO4uMNumbers: Γίνεται προσπέλαση του array of oceanRecord και οι δείκτες-είσοδοι ενημερώνονται ώστε να κρατάνε την μέγιστη και ελάχιστη τιμή PO4uM που υπάρχει.

Χρονομέτρηση:

Για την χρονομέτρηση έγινε χρήση τροποποιημένων συναρτήσεων countingSortNoCSV και heapSortNoCSV, με σκοπό τον υπολογισμό των runtime τους. Οι συγκεκριμένες δεν παράγουν csv αρχείο και έτσι μπορούμε τρέχοντας τες πολλές φορές να υπολογίσουμε τον μέσο χρόνο εκτέλεσης του κάθε αλγόριθμου. Στην main υπάρχει μέσα σε σχόλια η συνάρτηση που χρησιμοποιήθηκε για τον υπολογισμό του copyTime, ο οποίος τελικά δεν χρησιμοποιήθηκε διότι ήταν πολύ μικρός (~2μs) σε σχέση με τους αλγόριθμους.

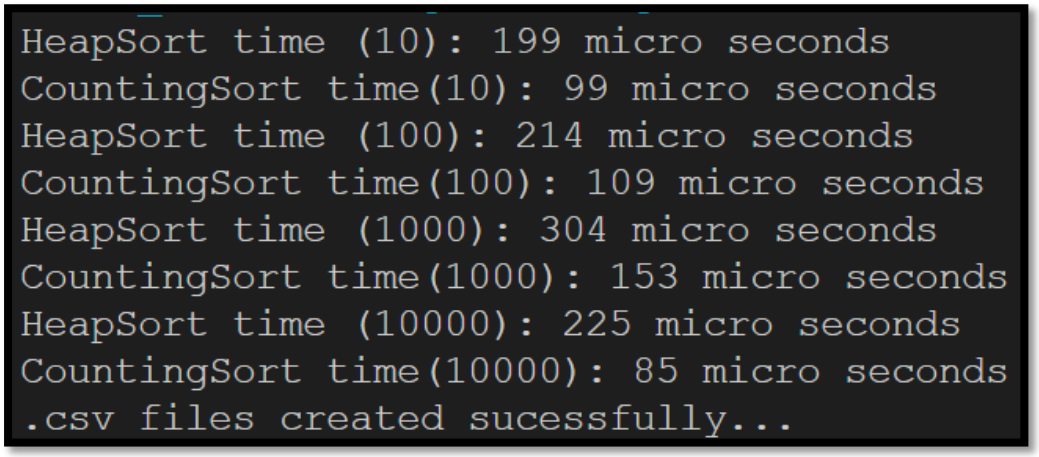
Για τον υπολογισμό των χρόνων των αλγόριθμων υπολογίζουμε το runtime των αλγορίθμων μέσω του βασικού βρόχου της main, ο οποίος σε συνδυασμό με την σταθέρα **MAGNITUDE_RUNS** (ορίζεται στο **projectshared.h**)τρέχει τόσες φορές (1,2,3,…), και οι αλγόριθμοι κάθε φορά με μια μεγαλύτερη τάξη αριθμού (10, 100, 1000,…) ώστε να υπολογίζεται το πραγματικό runtime.

Παρατηρούμε:

Times Ran	Heap Sort	Counting Sort
10	199μs	99μs
100	234μs	99μs
1000	249μs	114 μs
10000	249μs	98 μs

Οι χρόνοι είναι real-time και όχι runtime όμως φαίνεται πως ο Counting Sort είναι περισσότερο από 2 φορές πιο αποτελεσματικός απ’ ότι ο Heap Sort (i5-4460, 16GB RAM).

Screenshots:



1.3

Documentation:

Το πρόγραμμα αρχικά μέσω της **parseCSV_unsorted** αντιγράφει τα αναγκαία στοιχεία από το ocean.csv και τα αποθηκεύει στην μνήμη. Η αντιγραφή της ημερομηνίας γίνεται με σειρά Έτος-Μήνας-Ημέρα χωρίς ενδιάμεσα στοιχεία προκειμένου να είναι ένας ενιαίος ακέραιος αριθμός. Αυτό μας βοηθάει, όταν θα χρειαστεί να συγκρίνουμε δύο ημερομηνίες, να είναι έτοιμοι δύο αριθμοί που μπορούμε να χρησιμοποιήσουμε αμέσως καθώς η συγκεκριμένη διάταξη εξασφαλίζει πως οι αριθμοί είναι ήδη με σειρά φθίνουσας σημαντικότητας bit.

Έπειτα γίνεται χρήση της **insertionSort** (ίδια υλοποίηση με 1.1) η οποία ταξινομεί το array που είναι αποθηκευμένο από την **parseCSV_unsorted** (προκειμένου να μπορούμε να χρησιμοποιήσουμε τις **binarySearch** και **interpolationSearch** πρέπει ο πίνακας να είναι ταξινομημένος) και δημιουργεί ένα νέο αρχείο, το pt1_3_insertion.csv.

Στην συνέχεια χρησιμοποιείται η **parseCSV_sorted** η οποία αποθηκεύει στην μνήμη τα στοιχεία του πίνακα από το αρχείο pt1_3_insertion.csv, όπου μπορούμε πλέον να χρησιμοποιήσουμε τους αλγόριθμους **binarySearch** και **interpolationSearch**.

insertionSort: Ίδια υλοποίηση με 1.1

swap: Ίδια υλοποίηση με 1.1

binarySearch: Απλή υλοποίηση του αλγόριθμου.

InterpolationSearch: Απλή υλοποίηση του αλγόριθμου.

Χρονομέτρηση:

Οι συναρτήσεις που χρησιμοποιήθηκαν για την χρονομέτρηση είναι οι **binarySearchNoPrint** και **interpolationSearchNoPrint** οι οποίες είναι ίδιες με τις **binarySearch** και **interpolationSearch** , χωρίς τις εντολές printf , προκειμένου να μην επηρεάσουν τον υπολογισμό του χρόνου αλλά και να μην συσκοτίσουν το terminal με πολλαπλά μηνύματα λόγω των επαναλήψεων του **repeats** και **MAGNITUDE_RUNS** στην main.

Για τον υπολογισμό των χρόνων των αλγόριθμων υπολογίζουμε το runtime τους μέσω του βασικού βρόχου της main, ο οποίος τρέχει τόσες φορές όσο η τιμή της σταθεράς **MAGNITUDE_RUNS** (ορίζεται στο **projectshared.h**) (1,2,3,… επαναλήψεις), και οι αλγόριθμοι κάθε φορά με μια μεγαλύτερη τάξη αριθμού (10, 100, 1000,…) ώστε να υπολογίζεται το πραγματικό runtime.

Εφ’όσον το σετ δεδομένων μας είναι ομοιόμορφα κατανεμημένο τότε μπορούμε να υποθέσουμε πως ο χρόνος μέσης περήπτωσης για την δυαδική αναζήτηση και αναζήτηση παρεμβολής είναι αντίστοιχα **O(logn)** και **O(log(logn))** , και επομένως στις περισσότερες περιπτώσεις μπορούμε να υποθέσουμε πως σε σχέση με τον χρόνο , η αναζήτηση παρεμβολής είναι πιο αποτελεσματική.

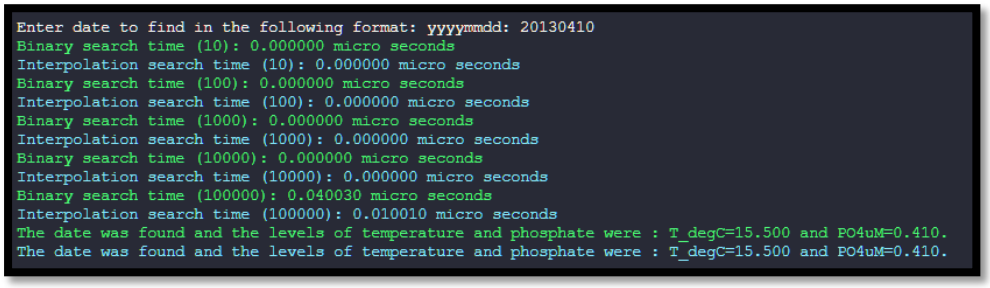
Πράγματι όπως φαίνεται από τα παρακάτω στιγμιότυπα , όταν επιλέγουμε μία τυχαία ημερομηνία , παρατηρούμε πως η αναζήτηση παρεμβολής είναι κατά μέσο όρο πιο αποτελεσματική από την δυαδική αναζήτηση.

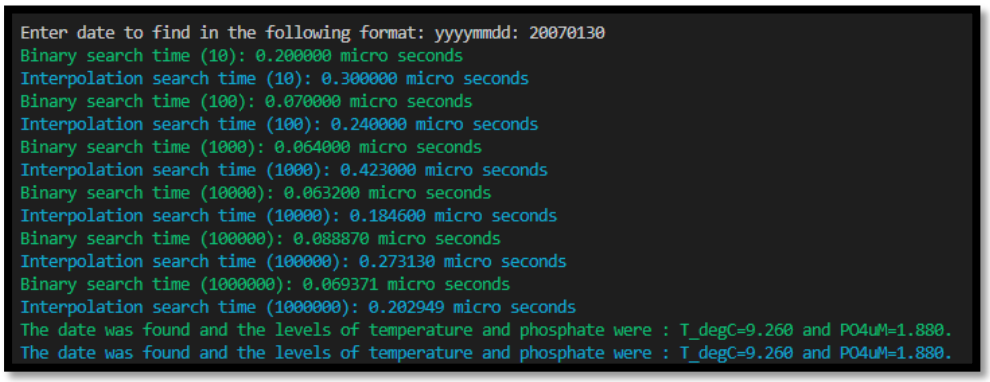
Παρατηρήσεις:

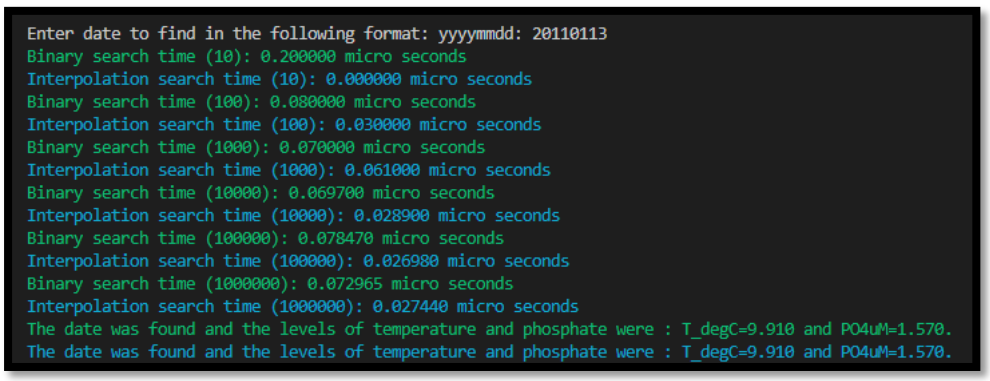
Η αναζήτηση με παρεμβολή λειτουργεί βέλτιστα όταν οι αριθμοί του σετ δεδομένων είναι ομοιόμορφα κατανεμημένοι (**uniform**). Όταν εκπληρώνεται αυτή η συνθήκη, η χρονική πολυπλοκότητα είναι **O(log(logn))**. Στην χείριστη περίπτωση η αναζήτηση με παρεμβολή χρείαζεται **O(n)** χρόνο.

Η δυαδική αναζήτηση δεν επηρεάζεται αισθητά απο την κατανομή των αριθμών και η χρονική πολυπλοκότητα παραμένει **O(logn)**.

Screenshots:







```
Enter date to find in the following format: yyyymmdd: 20120129
Binary search time (10): 0.100000 micro seconds
Interpolation search time (10): 0.100000 micro seconds
Binary search time (100): 0.620000 micro seconds
Interpolation search time (100): 0.040000 micro seconds
Binary search time (1000): 0.063000 micro seconds
Interpolation search time (1000): 0.027000 micro seconds
Binary search time (10000): 0.062800 micro seconds
Interpolation search time (10000): 0.026200 micro seconds
Binary search time (100000): 0.067550 micro seconds
Interpolation search time (100000): 0.031290 micro seconds
Binary search time (1000000): 0.082696 micro seconds
Interpolation search time (1000000): 0.035330 micro seconds
The date was found and the levels of temperature and phosphate were : T_degC=9.980 and PO4uM=1.740.
The date was found and the levels of temperature and phosphate were : T_degC=9.980 and PO4uM=1.740.
```

```
Enter date to find in the following format: yyyymmdd: 20170814
Binary search time (10): 0.100000 micro seconds
Interpolation search time (10): 0.100000 micro seconds
Binary search time (100): 0.060000 micro seconds
Interpolation search time (100): 0.040000 micro seconds
Binary search time (1000): 0.056000 micro seconds
Interpolation search time (1000): 0.026000 micro seconds
Binary search time (10000): 0.055300 micro seconds
Interpolation search time (10000): 0.025800 micro seconds
Binary search time (100000): 0.077760 micro seconds
Interpolation search time (100000): 0.053580 micro seconds
Binary search time (1000000): 0.069421 micro seconds
Interpolation search time (1000000): 0.039205 micro seconds
The date was found and the levels of temperature and phosphate were : T_degC=20.920 and PO4uM=0.130.
The date was found and the levels of temperature and phosphate were : T_degC=20.920 and PO4uM=0.130.
```

1.4

Documentation:

Η υλοποίηση της main είναι παρόμοια με την **1.3** .

BinarySearchInterpolation: Για τον αλγόριθμο χρησιμοποιήθηκε ο ψευδοκώδικας από τις διαφάνειες searching.pdf . Ο αλγόριθμος κάθε φορά μετά το πρώτο ωήμα του αλγόριθμου παρεμβολής πραγματοποιεί επιπλέον βήματα που αυξάνονται γραμμικά μέχρι να βρεί το διάστημα που περιέχει την ζητούμενη ημερομηνία οπού έπειτα θα αρχίσει να επαναλαμβάνει την ίδια διαδικασία στο νέο διάστημα.

binarySearchInterpolationImproved: Εδώ χρησιμοποιήθηκε η λογική απο το βιβλίο «Δομές Δεδομένων», Α.Κ. Τσακαλίδης. Τα ±1 στα left right είναι για να αποφύγουμε τυχόν αστοχίες με τα indexes, ενώ υπάρχουν πολλοί έξτρα έλεγχοι (στην αρχή εάν είναι μέσα στον array, έπειτα εάν η τιμή του next ξεφύγει) με σκοπό να αποφύγουμε λάθη κατά την εκτέλεση. Επίσης γίνεται χρήση του binarySearch από προηγούμενο ερώτημα για το τελευταίο βήμα του αλγορίθμου(η αναζήτηση της μεγαλύτερης σε θέση στον πίνακα ημερομηνίας δεν λειτουργεί).

Χρονομέτρηση:

Η υλοποίηση του τρόπου χρονομέτρησης είναι ίδια με την **1.3** .

Γνωρίζουμε πως η πολυπλοκότητα του χρόνου της μέσης περίπτωσης της δυικής αναζήτησης παρεμβολής είναι O(log(logn)) και ο χειροτερος O(√n) . Από τα παρακάτω στιγμιότυπα (Screenshot 1* και Screenshot 2*) παρατηρουμε πως ο χρόνος της χειρότερης περίπτωσης είναι περίπου 18 φορές μεγαλύτερος που επαληθεύει τα παραπάνω καθώς με βάση το μέγεθος του πίνακα ο χρόνος της μέσης περίπτωσης μπορεί να είναι έως και 70 φορές μεγαλύτερος(log(log1400))=0.5 και √(1400)=35).

Από το στιγμιότυπο 3 παρατηρούμε πως ο χρόνος χειρότερης περίπτωσης του αλγόριθμου έχει μειωθεί σημαντικά (10 φορές πιο γρήγορος) καθώς πλέον η πολυπλοκότητα της χειρότερης περίπτωσης του αλγόριθμου είναι O(logn) αντί για O(√n) και με βάση το μέγεθος του συγκεκριμένου πίνακα ο χρόνος μπορεί να βελτιωθεί και να γίνει έως και 11 φορές πιο γρήγορος(logn1400=3.14 και √(1400)=35).

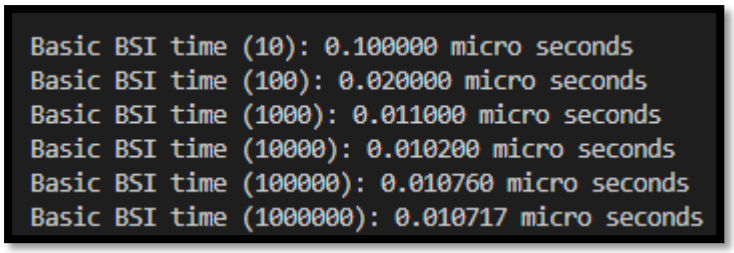
Παρατηρήσεις:

Για την improved BSI χρειάζεται ένα βήμα διότι από τον ορισμό του next προσεγγίζεται καλά η τιμή.

Screenshots:

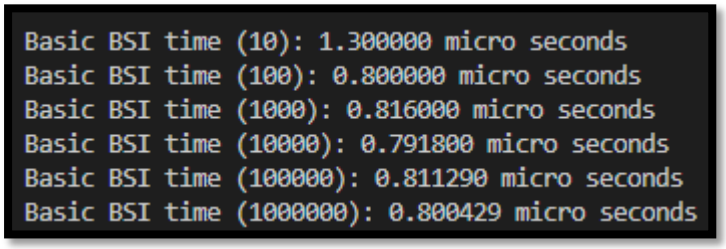
Average time case(Basic BSI):

Screenshot 1*(Date: 20000107)



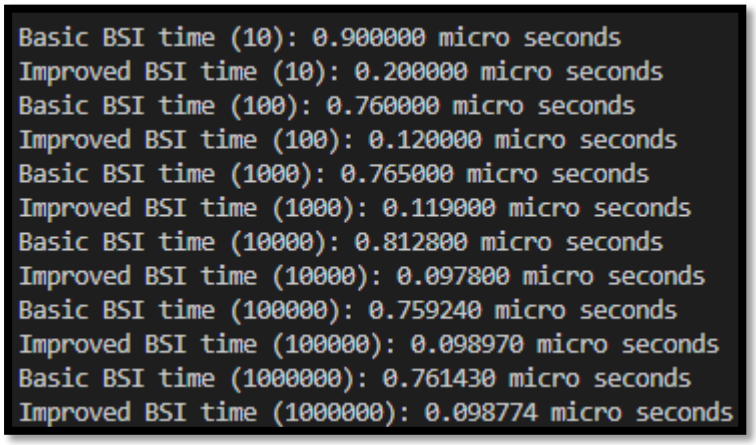
Worst time case (Basic BSI):

Screenshot 2* (Date: 20191117)



Wost time case (Basic BSI and Improved BSI):

Screenshot 3*(Date:20191117)



Part 2:

2.1

Documentation:

Το πρόγραμμα αφού πρώτα μετατρέψει το αρχείο csv σε δέντρο AVL, εισέρχεται σε βρόχο ο οποίος κάθε φορά αφήνει τον χρήστη να επιλέξει και εκτελέσει μία από τις προβλεπόμενες λειτουργίες. Η ημερομηνία εύκολα μετατράπηκε σε ακέραιο με την χρήση 2 breakpoint για την strtok και αριθμητικές πράξεις σε μορφή yyyymmdd. Επίσης να αναφερθεί πως δεν υπάρχει έλεγχος χαρακτήρων στην είσοδο επιλογής και έτσι το πρόγραμμα μπορεί να κολλήσει αν η είσοδος δεν είναι μεταξύ των ακεραίων 1-5.

Σύντομη περιγραφή απλών συναρτήσεων:

leftRotate: Αριστερή περιστροφή AVL.

rightRotate: Βλ. πάνω.

inOrderTree: Απεικόνιση δέντρου in order με σωστό formatting για την ημερομηνία.

accessTree: Απλό BST access που επιστρέφει τον κόμβο αν βρέθηκε.

modifyTree: Όπως πάνω αλλά αλλάζει την τιμή του κόμβου.

nextInOrder: Βρίσκεται ο επόμενος κόμβος κατά τιμή.

getBalance: Εύρεση ισορροπίας κόμβου.

getHeight: Εύρεση ύψους κόμβου.

newNode: Χρησιμοποιείται από την insert για δυναμική δημιουργία node.

Για την πρακτική υλοποίηση των insert και delete πέρα από θεωρητικές πήγες έγινε και χρήση του άρθρων του www.geeksforgeeks.org, σε κάθε περίπτωση με τροποποιήσεις ή και διαφορετικές υλοποιήσεις σε σημεία.

insertNode: Χρησιμοποιώντας αναδρομή μπορούμε να βρούμε τη θέση που πρέπει να προστεθεί ο κόμβος και ανάποδα να ενημερώνουμε ύψη, ισορροπίες και να εκτελούμε το rotate. Με σκοπό την αποφυγή αχρείαστων ελέγχων για τυχόν περιστροφές, εάν σε μία συνολική εκτέλεση του insertNode υπάρξει περιστροφή, η μεταβλητή εισόδου rotated γίνεται 1 (από 0 που τίθεται στην main) και έτσι αποφεύγονται επιπλέον έλεγχοι (Το τελευταίο εντέλει δεν υλοποιήθηκε, ούτε με τη χρήση βοηθητικής συνάρτησης και έτσι γίνονται οι επιπλέον έλεγχοι).

deleteNode: Όπως η insert αλλά εδώ δεν υπάρχει rotated μεταβλητή. Επίσης σε περίπτωση που βρεθεί ο κόμβος προς διαγραφή, ακολουθώντας την διαγραφή BST τον διαγράφουμε, απελευθερώνουμε την μνήμη δυναμικά και στην συνέχεια συνεχίζουμε με τον εκ νέου υπολογισμό για τα ύψη, τις ισορροπίες και τις περιστροφές.

Screenshots:

```
AVL tree was sucessfully created.
1.In order depiction of AVL tree.
2.Search for temperature given date.
3.Modify temperature for given date.
4.Delete record for given date.
5.Quit.
```

```
2019/11/15: 10.73
2019/11/16: 11.71
2019/11/17: 13.41
2019/11/18: 16.03


1.In order depiction of AVL tree.
```

```
Enter date to find in the following format: yyyymmdd: 20191114
2019/11/14: 10.43
```

```
Enter date to modify in the following format: yyyymmdd: 20191114

Enter new temperature for selected date: 111
2019/11/14: 111.00
```

```
2019/11/14: 111.00
2019/11/15: 10.73
2019/11/16: 11.71
2019/11/17: 13.41
2019/11/18: 16.03
```

```
Enter date to delete in the following format yyyymmdd: 20191114
Record was deleted, if it existed.
```

```
2019/11/12: 18.28
2019/11/13: 10.29
2019/11/15: 10.73
2019/11/16: 11.71
2019/11/17: 13.41
2019/11/18: 16.03
```

2.2

Documentation:

Μετατρέποντας το date σε array, μπορούμε να κρατάμε μέχρι **MAX_DUPLICATES** ημερομηνίες με την ίδια θερμοκρασία και να εκτυπώνονται όλες μαζί. Αυτή ήταν η υλοποίηση που επιλέχθηκε, καθώς η εναλλακτική του BST με πολλαπλά από το ίδιο κλειδί είναι περίπλοκη ενώ οι περιστροφές για το AVL περισσότερο (https://stackoverflow.com/questions/300935/are-duplicate-keys-allowed-in-the-definition-of-binary-search-trees).

Σύντομη περιγραφή απλών συναρτήσεων (σε σχέση με 2.1):

inOrderTree: Αφαιρέθηκε.

accessTree: Αφαιρέθηκε.

modifyTree: Αφαιρέθηκε.

nextInOrder: Αφαιρέθηκε.

getBalance: Εύρεση ισορροπίας κόμβου.

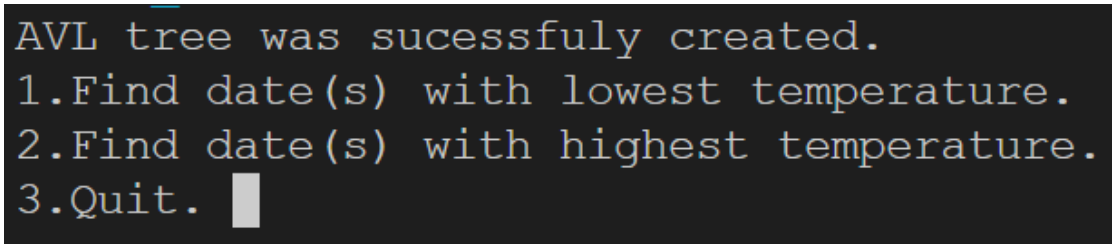
getHeight: Εύρεση ύψους κόμβου.

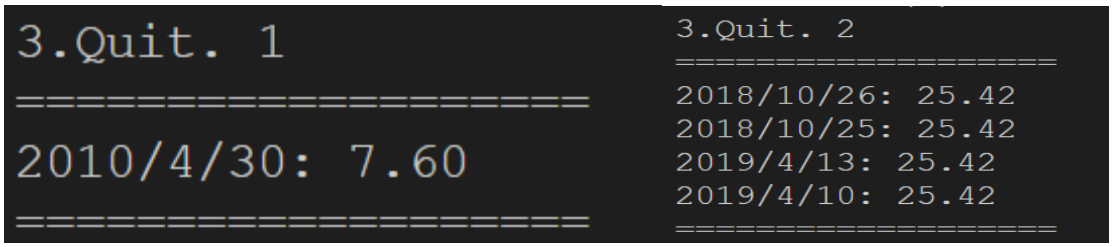
min/maxTemperature: Επιστρέφει τον κόμβο με μικρότερη/μεγαλύτερη τιμή (μπορεί να είναι πολλαπλές)

insertNode: Ίδιος με 2.1 αλλά υποστηρίζονται πλέον πολλαπλές τιμές για ίδιο κλειδί.

deleteNode: Αφαιρέθηκε

Screenshots:





2.3

Documentation:

Το πρόγραμμα χρησιμοποιεί την συνάρτηση **parseCSV** του projectshared.h για να μετατρέψει τις γραμμές του ocean.csv σε κόμβους λίστας recordsIn στοιχείων oceanRecord. Η λίστα στη συνέχεια χρησιμοποιείται σαν παράμετρος στην **hashOceanRecords**, η οποία δημιουργεί ένα **array of pointers** που δείχνουν σε struct τύπου recordNode.

Κάθε δείκτης του πίνακα είναι η αρχή ενός singly linked list.

Σύντομη περιγραφή συναρτήσεων:

hashFunction: Επιστρέφει το υπόλοιπο της διαίρεσης της συνολικής ASCII τιμής μιας ημερομηνίας δια του αριθμού γραμμών του hash table (HASH_SIZE. Ενδεικτικά, έχει οριστεί σαν 13).

hashOceanRecords: Προσπελάζει κάθε στοιχείο oceanRecord της recordsIn και το μετατρέπει σε recordNode του hash table. Νέα στοιχεία προστίθενται στην αρχή της γραμμής.

searchDate: Γραμμική αναζήτηση θερμοκρασίας μιας ημερομηνίας στο hash table χρησιμοποιώντας δείκτη (**linearProber**) που διατρέχει τα στοιχεία του.

deleteDate: Γραμμική αναζήτηση μιας ημερομηνίας στο hash table και διαγραφής της.

modifyDate: Γραμμική αναζήτηση μιας ημερομηνίας στο hash table και τροποποίησης του πεδίου θερμοκρασίας του αντίστοιχου recordNode.

Έχει υλοποιηθεί ένα απλό menu στην main μέσα σε ένα ατέρμονο while loop (condition !NULL), για την χρήση των προαναφερόμενων συναρτήσεων.

Ένα **πρόβλημα** που αντιμετωπίστηκε κατά την υλοποίηση του προγράμματος ήταν η εύρεση condition λήξης της γραμμικής αναζήτησης σε περίπτωση που ο δείκτης αναζήτησης linearProber βρεθεί στο τέλος της γραμμής. Η **λύση** που εφαρμόστηκε είναι η εξής: Κατά την δημιουργία του hash table, το πρώτο στοιχείο που εισάγεται σε κάθε γραμμή αποτελεί και το tailNode της γραμμής. Ο δείκτης tailNode.next δείχνει πάντα στο tailNode, και condition λήξης της γραμμικής αναζήτησης είναι linearProber == (*linearProber).next.

Στη περίπτωση διαγραφής του tailNode, το αμέσως επόμενο node μετατρέπεται στο tail node της λίστας, κάτι το οποίο διαχειρίζεται η deleteDate.

Κάθε δομή που δεν χρειάζεται πια από το πρόγραμμα διαγράφεται για να ελευθερωθεί μνήμη χρησιμοποιώντας την εντολή free(). Το ίδιο ισχύει για το πρόγραμμα, που μετά την επιλογή του exit από τον χρήστη, κάνει free κάθε γραμμή του πίνακα μέσω της freeHashTableMemory.

```
Greetings!
Load contents of file using AVL (1) or HASHING (2)? Type anything else to EXIT from program.
2
Hash table was successfully created.
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
1
Which date's temperature would you like to see?
02/01/2002
Temperature for 02/01/2002 was recorded to be 14.73 degrees celsius.
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
2
Which date's temperature would you like to modify?
02/01/2002
What would you like to modify the temperature for 02/01/2002 to?
35469.2
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
1
Which date's temperature would you like to see?
02/01/2002
Temperature for 02/01/2002 was recorded to be 35469.20 degrees celsius.
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
3
Which date's records would you like to delete?
02/01/2002
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
1
Which date's temperature would you like to see?
02/01/2002
Temperature for 02/01/2002 was not recorded.
1. See temperature based on date given.
2. Modify temperature of date given.
3. Delete record of date given.
4. Quit
4
Freeing memory...
Exiting...
```

Ενοποίηση του part 2:

Κατά την ενοποίηση του part 2, τα αρχεία μετατράπηκαν από .c σε .h ώστε να μπορούν να χρησιμοποιηθούν οι συναρτήσεις τους από το part2_menu.c. Στη main αυτού του προγράμματος, ο χρήστης ερωτάται αν θέλει το αρχείο να φορτωθεί σε μορφή AVL (και στη συνέχεια φόρτωση με βάση ημερομηνίας ή θερμοκρασίας) ή σε μια δομή hashing. Με βάση την επιλογή του, καλείται η αντίστοιχη συνάρτηση από τα header file που υπάρχουν (part2_1, part2_2, part2_3).

Για την αποφυγή **name conflicts** μεταξύ συναρτήσεων ή struct διαφορετικών header (των part2_1 και part2_2 συγκεκριμένα) ήταν αναγκαία η **μετονομασία** μερικών από αυτών. Παραδείγματος χάρη, το part2_1 και το part2_2 χρησιμοποιούν μια δομή κόμβου AVL (avlNode), όμως το κάθε part χρείαζεται διαφορετικά δεδομένα σε κάθε κόμβο του για να λειτουργήσει. Στο part2_2 λοιπόν, το avlNode μετονομάστηκε σε avlNodeTemp.