Αρχεία υλοποίησης: main.c, BulkTreeSimple.c, BulkTreeAVL.c, BulkTree.h, TStoixeioyTree.c, TStoixeioyTree.h
Ενδεικτικά αποτελέσματα : atoma-random-out.txt, atoma-sorted-out.txt, pinakas-xronometrisewn.txt, simple-AVL.txt
Όνομα/Επώνυμο : Νικόλαος Μπεγέτης 1115200700281
**********

Λειτουργικότητα:
**********

Το πρόγραμμα είναι η επέκταση-βελτίωση της προσομοίωση της ουράς πελατών μίας τράπεζας χρησιμοποιώντας λίστες με δείκτες για εξοικείωση με τις λίστες και τις εφαρμογές τους.
Υλοποίηση Ζητουμένων:

1. Το πρόγραμμα υλοποιείται με τον ΑΤΔ Δέντρο, και πιο συγκεκριμένα υλοποιείται με δύο τρόπους (με απλό ΔΔΑ και με AVL ΔΔΑ) και αφορά τη διαχείριση 2 αρχείων που περιέχουν 10.000 ονοματεπώνυμα(το ένα αρχείο ταξινομημένο και το άλλο random) και ενός αρχείου 1.000 ατόμων από όπου γίνεται αναζήτηση των ονομάτων στο δέντρο, στο οποίο κάθε κόμβος του περιέχει ένα όνομα και ένα επίθετο.
2. Τα αρχεία που αναπτύχθηκαν είναι:
    - main.c: όπου δημιουργείται ένα κενό BulkTree, έπειτα εισάγει στους κόμβους του τα ονοματεπώνυμα από τα αρχεία(πρώτα από το random αρχείο και μετά το πέρας όλων των διαδικασιών από το sorted), μετά διατρέχει το δέντρο και εκτυπώνει με αλφαβητική σειρά στο αρχείο atoma-random-out.txt τα ταξινομημένα πλέον ονόματα. Στη συνέχεια αναζητά τα άτομα από το αρχείο atoma-search.txt(δίνεται) στους κόμβους του δέντρου και τέλος μετά το πέρας όλων των παραπάνω καταστρέφει το δέντρο. Η ίδια διαδικασία επαναλαμβάνεται για άλλη μία φορά. Αυτή τη φορά όμως εισάγονται στους κόμβους του δέντρου τα στοιχεία από το atoma-sorted.txt και έπειτα οι εκτυπώσεις γίνονται στο atoma-sorted-out.txt. Τέλος να σημειωθεί ότι η συνάρτηση κατασκευής επιστρέφει ένα πίνακα με τους χρόνους που κάνει για να κατασκευάσει 2^n κόμβους, όπου 2^n: 2,4,8,...,8192 και όλο το δέντρο(10.000 άτομα) και οι συναρτήσεις εκτύπωσης και αναζήτησης επιστρέφουν τους συνολικούς χρόνους ολοκλήρωσης τους. Τα δεδομένα αυτά εκτυπώνονται στο τέλος στο stdout(οθόνη) και στο αρχείο pinakas-xronometrisewn.txt.
    - BulkTree.h: περιέχει τα πρωτότυπα των συναρτήσεων των αρχείων BulkTreeSimple.c και BulkTreeAVL.c που χρησιμοποιούνται άμεσα από την main(η οποία την κάνει include) και κάνει include το αρχείο κεφαλίδα TstoixeioyTree.h για τον τύπο στοιχείου και τις πράξεις του.
    - BulkTreeSimple.c και BulkTreeAVL.c: υλοποιούν τις 5 πράξεις(δημιουργία, καταστροφή, κατασκευή, εκτύπωση και αναζήτηση) που ορίζονται στο BulkTree.h και στην υλοποίηση τους χρησιμοποιούν τις συναρτήσεις που ορίζονται στα ch8_BSTpointer.h και ch8_AVLpointer.h(τα οποία θεωρούνται δεδομένα και δεν συμπεριλαμβάνω στο συμπιεσμένο φάκελο). Ακόμα μέσα στα αρχεία ορίζονται οι δομές της ρίζας του δέντρου και των κόμβων. Τέλος υπάρχουν άλλες 2 συναρτήσεις που βοηθάνε στην ταξινόμηση και εκτύπωση των στοιχείων σε αρχείο.
    - TStoixeioyTree.h, TStoixeioyTree.c: τα αρχεία περιλαμβάνουν αντίστοιχα τις δηλώσεις και τις υλοποιήσεις των πράξεων των στοιχείων του δέντρου καθώς και τα περιεχόμενα των στοιχείων(επώνυμο, όνομα).
    - Τα αρχεία ch8_BSTpointer2.h, ch8_BSTpointer2.c, ch8_AVLpointer.h, ch8_AVLpointer.c, atoma-random.txt, atoma-sorted.txt και atoma-search.txt θεωρούνται δεδομένα και δεν συμπεριλαμβάνονται στο συμπιεσμένο φάκελο.
        - Να σημειωθεί ότι η main πρέπει να μεταγλωττιστεί και εκτελεστεί 2 φορές(μία με τα αρχεία BulkTreeSimple.c και ch8_BSTpointer2 και άλλη μία με τα αρχεία BulkTreeAVL.c και ch8_AVLpointer.c)
3. Επίσης παραδίδονται ενδεικτικά αποτελέσματα τα οποία ούτως ή άλλως θα κατασκευάζονταν μέσα στη main.c. Αυτά είναι: atoma-random-out.txt, atoma-random-out.txt, atoma-sorted-out.txt και pinakas-xronometrisewn.txt. Τα 2 πρώτα είναι τα αρχεία που ζητούνται από την εκφώνηση για την ταξινομημένη εκτύπωση σε αρχείο και το τρίτο περιλαμβάνει τον πίνακα με τους χρόνους όπως διαμορφώθηκε μετά από 2 εκτελέσεις, η πρώτη με απλό δέντρο και η δεύτερη με AVL. Τα δεδομένα αυτά τα χρησιμοποίησα και στο αρχείο simple-AVL.txt στο οποίο συμπεριλαμβάνω και εκτενέστερα σχόλια για τα αποτελέσματα.
        - Να σημειωθεί ότι επειδή η κατασκευή γίνεται από το ίδιο πρόγραμμα και για AVL και για απλά δέντρα η συνάρτηση κατασκευής των αρχείων έχει mode=a ώστε να γίνεται επέκταση των αρχείων και όχι w(που διαγράφονται τα παλαιότερα δεδομένα του αρχείου για να γραφούν τα καινούργια). Ο λόγος είναι για να μπορεί ο χρήστης να κάνει συγκρίσεις μέσα στο ίδιο αρχείο και επειδή το mode=a είναι σαφώς πιο γρήγορο από το mode= w, αφού το μόνο που κάνει είναι να επεκτείνει και όχι να διαγράφει και να ξαναγράφει
        - Επίσης για να καταφέρω να δημιουργώ 2 διαφορετικά αρχεία χωρίς να χρειαστεί να αλλάξω τη συνάρτηση void TStree_writeValue(TStoixeioyTree Elem) προσθέτοντας της άλλο ένα όρισμα(που δεν επιτρέπεται γιατί επηρεάζει τα αρχεία που δεν πρέπει να αλλαχθούν) χρησιμοποίησα μία static μεταβλητή.

extras:
*********

1. Δημιουργία και χρησιμοποίηση της συνάρτησης "void inspectQbyOrder (BankOurasPtr oura);" η οποία εμφανίζει την κανονική ουρά των πελατών ή/και των ΑΜΕΑ(ανάλογα με την κλήση) όπως διαμορφώνεται μετά τις αλλαγές και εκτύπωση αυτής στην οθόνη κατά το τρέξιμο του εκτελέσιμου.

Οδηγίες χρήσης προγράμματος:
**********

Το πρόγραμμα μεταγλωττίζεται για το απλό δέντρο με την εντολή: *gcc -o tree main.c BulkTreeSimple.c ch8_BSTpointer2.c TStoixeioyTree.c -lm* και για το AVL: *gcc -o tree main.c BulkTreeAVL.c ch8_AVLpointer.c TStoixeioyTree.c -lm* και το εκτελέσιμο που δημιουργείται και στις 2 περιπτώσεις τρέχει με το ./tree. Να τονίσω ότι επειδή έκανα χρήση της βιβλιοθήκης <math.h> στον linker κατά την μεταγλώττιση πρέπει να μπαίνει και το επίθεμα *-lm*.
Περιβάλλον υλοποίησης και Δοκιμών:
**********

Το πρόγραμμα αναπτύχθηκε σε gcc σε linux και δοκιμάστηκε και στα linux και τα sun της σχολής

| Εργασία/Χρόνος | Απλό ΔΔΑ- random input | Απλό ΔΔΑ- sorted input | AVL ΔΔΑ- random input | AVL ΔΔΑ- sorted input |
|---|---|---|---|---|
| Κατασκευή μετά από | | | | |
| 2 | 0.093000 msec | 0.037000 msec | 0.097000 msec | 0.034000 msec |
| 4 | 0.097000 msec | 0.044000 msec | 0.101000 msec | 0.132000 msec |
| ++++ | ++++ | ++++ | ++++ | ++++ |
| 8196 | 16.530000 msec | 2609.682000 msec | 243.995000 msec | 323.097000 msec |
| όλα | 19.700000 msec | 3818.984000 msec | 286.483000 msec | 408.424000 msec |
| Εκτύπωση | 191.289000 msec | 192.082000 msec | 195.676000 msec | 226.478000 msec |
| Αναζήτηση | 21.907000 msec | 500.749000 msec | 35.366000 msec | 14.768000 msec |

Συμπεράσματα για την κατασκευή:

Για τα απλά ΔΔΑ παρατηρούμε ότι ενώ τα sorted αρχικά διαβάζονται και τοποθετούνται ταχύτερα στους κόμβους των δέντρων στη συνέχεια και όσο μεγαλώνει το δέντρο και συνεπώς το βάθος του επειδή το αρχείο είναι ήδη ταξινομημένο και άρα θα γίνονται κατά πολύ περισσότερες συγκρίσεις για να τοποθετηθεί τελικά το κάθε στοιχείο στην ουρά του δέντρου και όχι σε κάποιο ενδιάμεσο κόμβο όπως θα γινόταν με το random για αυτό και στο τέλος υπάρχουν τεράστιες αποκλίσεις. Με άλλα λόγια στα απλά ΔΔΑ δεν συμφέρει να δίνουμε για την κατασκευή sorted αρχεία γιατί αγγίζουμε σχεδόν τετραγωνική απόκλιση.Αυτό συνηθίζεται να λέγεται κόστος διαδρομής και εξαρτάται ως επί το πλήστον από το βάθος.

Για τα AVL ΔΔΑ η θεωρία μας λέει ότι το ύψος του AVL επηρεάζει πολύ στο κόστος κατασκευής και σύμφωνα με αυτήν αυτό είναι h~=1.44 lgn. Δηλαδή απαιτείται το πολύ 44% περισσότερο χρόνο από το βέλτιστο όταν τα αρχεία που δίνουμε προς εισαγωγή στο δέντρο είναι ταξινομημένα και αυτό γιατί όταν το αρχείο είναι ταξινομημένο το AVL κάνει περισσότερες περιστροφές οι οποίες μάλιστα δεν είναι και απαραίτητες αφού θα επιστρέψει πάλι στην αρχική του θέση. Παρόλα αυτά στην πράξη όπως και εδώ φαίνεται ότι τελικά είναι λίγο λιγότερος. Έτσι παρατηρούμε ότι αν πολλαπλασιάσουμε τον τελικό χρόνο κατασκευής από το random αρχείο(286.483000 msec) με 1.44 έχουμε αποτέλεσμα 411,84 που προσεγγίζει πολύ τον τελικό χρόνο κατασκευής AVL ΔΔΑ sorted(408.424000 msec).

Συμπεράσματα για την εκτύπωση:

Παρατηρούμε ότι επειδή στην εκτύπωση εκτυπώνουμε από ταξινομημένα με ενδοδιάταξη και τα απλά και τα AVL δέντρα, γι' αυτό οι τιμές ανάμεσα σε random input και sorted input διαφέρουν ελάχιστα(και αυτές κυρίως λόγω του συστήματος), ίσως αν η υλοποίηση ήταν διαφορετική, να βλέπαμε μεγαλύτερες διαφορές.

Συμπεράσματα για την αναζήτηση:

Όπως ο χρόνος κατασκευής,έτσι και ο χρόνος αναζήτησης εξαρτάται από το βάθος του δένδρου. Σύμφωνα με τη θεωρία σε ένα απλό ΔΔΑ ο μέσος αριθμός συγκρίσεων για μία επιτυχημένη αναζήτηση είναι περίπου ίδιος με εκείνος για μία αποτυχημένη. Γνωρίζοντας δηλαδή ότι ένα στοιχείο βρίσκεται στη λίστα, η αναζήτησή του με τη μέθοδο των συγκρίσεων των κλειδιών δεν προσφέρει μεγάλη βοήθεια. Η ελάχιστη τιμή του I επιτυγχάνεται για ένα πλήρες δυαδικό δέντρο όταν Imin = O(n lgn), δηλαδή εξαρτάται από το βάθος του δέντρου. Όσο μικρότερο το βάθος τόσο καλύτερα έτσι στα απλά random ΔΔΑ θα είναι O(n lgn) δηλαδή >= n lgn, ενώ στα ταξινομημένα Θ(n lgn) δηλαδή ~=n lgn. Ακόμα σύμφωνα με τη θεωρία το μέσο κόστος για τη μη ισοζύγιση ενός ΔΔΑ είναι περίπου 39% περισσότερες συγκρίσεις ( U(n) ~=1.39lgn ). Με βάση λοιπόν όλα αυτά παρατηρούμε ότι πράγματι επειδή στα απλά ΔΔΑ το βάθος είναι πολύ μεγάλος όταν αυτά είναι ταξινομημένα αυτό συνεπάγεται ότι το κόστος χρόνου θα είναι πολύ μεγαλύτερο και πράγματι έτσι είναι. Αντίθετα όμως παρατηρούμε ότι στην αναζήτηση σε ταξινομημένα αρχεία για AVL ΔΔΑ επειδή ακριβώς τα AVL δεν έχουν μεγάλο βάθος, και πόσο μάλλον όταν είναι και ταξινομημένα τότε το κόστος αναζήτησης είναι αισθητά μικρότερο από αυτό των random.

Για περισσότερες συγκρίσεις μπορείτε να δείτε και το αρχείο simple-AVL.txt