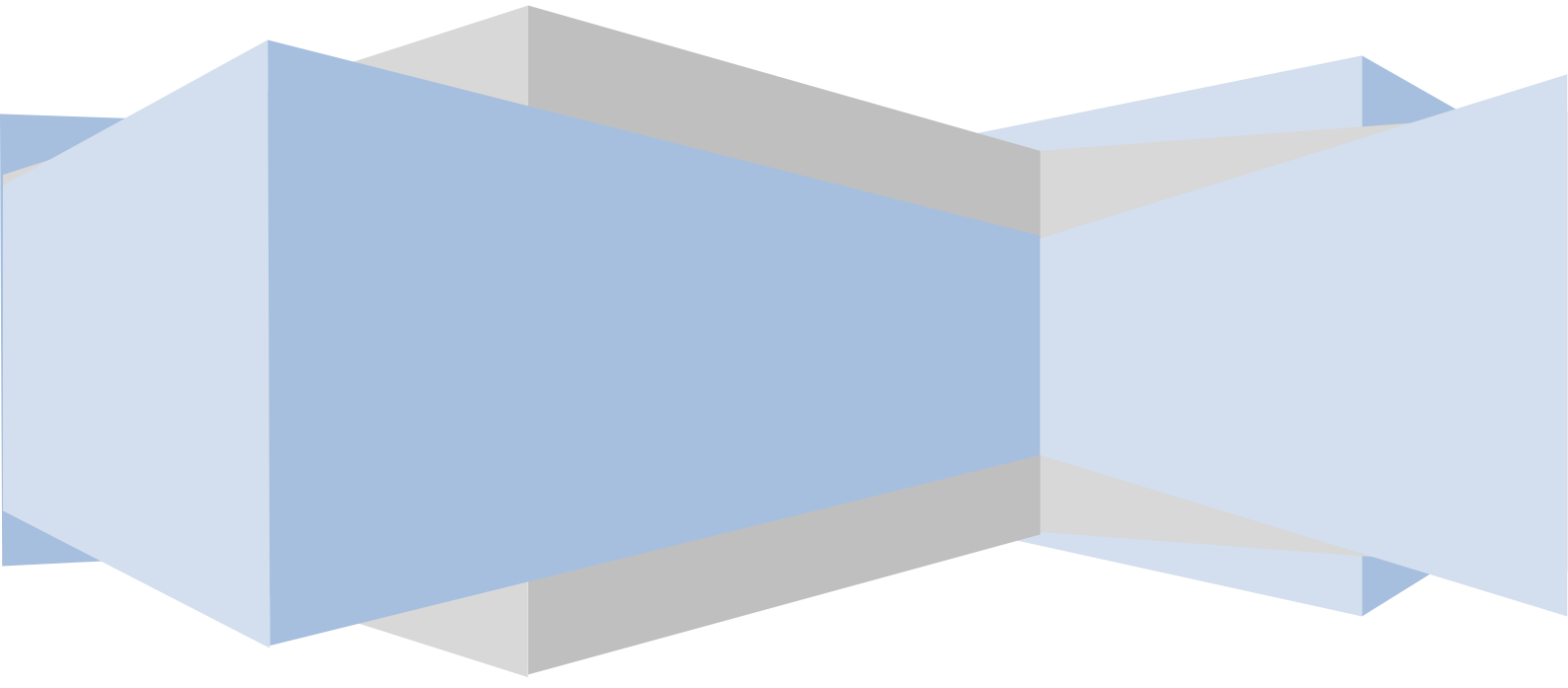
Δομές Δεδομένων

Project 2015



Πίνακας περιεχομένων

Ιεραρχία κώδικα

Για τον σχεδιασμό της άσκησης αυτής ακολουθήθηκε μία βασική δομή του Observer Pattern. Ο λόγος ήταν, η ευκολία επέκτασης ή και απομόνωσης λειτουργιών από τον κώδικα χωρίς να επηρεάζεται η δομή του. Παρακάτω θα δοθεί μία πιο λεπτομερής ανάλυση.

Κλάση Data Handler

Η κλάση αυτή επιφορτίζεται με τον ρόλο της εισαγωγής και εξαγωγής των εταιριών του προγράμματος μέσω των ειδικά διαμορφωμένων αρχείων csv καθώς και περεταίρω επεξεργασίας, εισαγωγές, διαγραφές μέσω του χρήστη. Η κλάση αυτή δρα ως το υποκείμενο (Subject) του pattern μας, ενημερώνοντας έτσι σε κάθε αλλαγή τους παρατηρητές (Observers) ώστε να αναβαθμίσουν τις δομές τους ή να προβούν σε οποιαδήποτε λειτουργία προβλέπεται.

Κλάση Observer

Η abstract κλάση αυτή, αναπαριστά την βασική δομή του κάθε αλγορίθμου αναζήτησης που θα υλοποιήσουμε ως προς την είσοδο και έξοδο των δεδομένων. Την κλάση αυτή κληρονομούν οι κλάσεις AVL Searches, Binary Searches, Linear Searches και TRIE Searches οι οποίες καλούνται να υλοποιήσουν τις μεθόδους της.

Κλάση Linear Searches

Η κλάση αυτή περιέχει τις γραμμικές αναζητήσεις βάση αναγνωριστικού ID καθώς και επωνύμου όπως ζητούνται στο Α μέρος της εργασίας. Κληρονομεί την κλάση Observer, για καλύτερη συνοχή του κώδικα, αλλά δεν υλοποιεί με κάποιο τρόπο τις μεθόδους αυτής, καθώς η δομή της δεν διαφοροποιείται από αυτή της Data Handler. Έτσι χρησιμοποιεί απευθείας τα δεδομένα στην σειριακή δομή, όπως και αυτά αποθηκεύονται στην τελευταία.

Κλάση Binary Searches

Η κλάση αυτή περιέχει την δυαδική αναζήτηση καθώς και την δυαδική αναζήτηση παρεμβολής βάση αναγνωριστικού ID, όπως και ζητείται στο μέρος Β της εργασίας. Κληρονομεί την κλάση Observer, υλοποιώντας τις μεθόδους της, διατηρώντας έτσι ξεχωριστή δομή η οποία ταξινομείται με βάση το ID, ώστε να δουλεύουν σωστά οι αλγόριθμοι μας. Η νέα αυτή δομή αναπαρίσταται με τη χρήση της βοηθητικής κλάσης BinCont.

Κλάση AVL Searches

Η κλάση αυτή περιέχει την δομή του AVL δένδρου υλοποιώντας αναζήτηση βάση αναγνωριστικού ID, όπως και ζητείται στο μέρος Γ της εργασίας. Κληρονομεί την κλάση Observer, υλοποιώντας τις μεθόδους της, δημιουργώντας έτσι την απαιτούμενη δομή δένδρου. Η νέα αυτή δομή αναπαρίσταται με τη χρήση της βοηθητικής κλάσης Node.

Κλάση TRIE Searches

Η κλάση αυτή περιέχει την δομή του digital tree υλοποιώντας αναζήτηση βάση επιθέτου εργαζομένου, όπως και ζητείται στο μέρος Δ της εργασίας. Κληρονομεί την κλάση Observer, υλοποιώντας τις μεθόδους της, διατηρώντας έτσι την ξεχωριστή δομή αυτή. Η νέα αυτή δομή αναπαρίσταται με την χρήση της δομής TRIECont.

Κλάση Testable

Η κλάση αυτή περιέχει βοηθητικές μεθόδους ώστε να προστεθεί κώδικας μέτρησης πράξεων σε κάθε κλάση αναζήτησης, όπως και ζητείται στο μέρος Ε της εργασίας. Η κλάση αυτή κληρονομείται από τις κλάσεις AVL Searches, Binary Searches, Linear Searches και TRIE Searches όπου και δημιουργούν μετρητές που μετράνε το πλήθος τον πράξεων για κάθε αναζήτηση όταν βρισκόμαστε σε Test Mode.

Κλάση Company-Employee

Οι κλάσεις αυτές αποτελούν την βασική αναπαράσταση των δεδομένων όπως και αυτές ζητήθηκαν στην εκφώνηση της άσκησης. Επιπρόσθετα διαθέτουν μερικές βοηθητικές μεθόδους εκτύπωσης των δεδομένων μας.

Λοιπά αρχεία

Επίσης έχουν δημιουργηθεί και τα αρχεία helper, main για του σκοπούς της άσκησης αυτής. Το αρχείο helper περιέχει κώδικα για χρονομέτρηση των λειτουργιών αναζήτησης, ενώ το main περιέχει την διεπαφή χρήστη.

Δομές Δεδομένων

Στο σημείο αυτό θα περιγραφούν οι δομές που δημιουργήθηκαν ή χρησιμοποιήθηκαν για το κάθε μέρος της εργασίας.

Α Μέρος

Η βασική δομή μίας εταιρίας απεικονίζεται μέσω των κλάσεων company και employee. Για την αποθήκευση όλης της πληροφορίας χρησιμοποιήθηκαν STL vectors ώστε να έχουμε δυνατότητα τυχαίας προσπέλασης σε O(1), αλλά και δυνατότητα επέκτασης της δομής (reallocate). Κατά την εκκίνηση της εφαρμογής και εισαγωγής από αρχείο, γίνεται δέσμευση χώρου ίσο με το πλήθος των εγγραφών συν μία σταθερά ALLOC_FACTOR, που ορίσαμε, ώστε να μην έχουμε πολλαπλά reallocate O(n) κατά την εισαγωγή, των εταιριών στο vector. Τέλος η δομή αυτή διαμοιράζεται σε όλες τις κλάσεις που υλοποιούν την κλάση Observer.

Β Μέρος

Για το μέρος αυτό, χρησιμοποιήθηκε η κλάση BinCont με περιεχόμενα το id της κάθε εταιρίας, δείκτη προς την εταιρία αναφοράς, ενώ η αποθήκευση των containers έγινε σε STL vector. Η χρήση του pointer εξοικονομεί χώρο στην μνήμη, ενώ η χρήση του επιπρόσθετου πεδίου id, ενισχύει την τοπικότητα των αναφορών κάνοντας γρηγορότερη την προσπέλαση των στοιχείων βάση του id τους.

Γ Μέρος

Για το μέρος αυτό, χρησιμοποιήθηκε η κλάση Node με περιεχόμενα το id της κάθε εταιρίας ως κλειδί, ένα δείκτη προς την εταιρία αναφοράς, καθώς και πληροφορία για τα παιδιά και το ύψος του κόμβου. Η δόμηση του δένδρου γίνεται μέσω της παραπάνω δομής.

Δ Μέρος

Για το μέρος αυτό, χρησιμοποιήθηκε η δομή TRIECont. Για την αναπαράσταση κάθε επιπέδου χρησιμοποιεί ένα STL Map, ενώ για την αποθήκευση των εταιριών μία STL List. Η χρήση του STL Map, σε αντίθεση με την κλασική υλοποίηση του πίνακα μας κοστίζει στην προσπέλαση από O(1) σε O(logk) , όπου k το μέγεθος του λεξικού μας καθώς υλοποιούνται με Red-Black trees. Έτσι για μεγάλο λεξικό και αραιά επίπεδα εξοικονομείται χώρος, δηλαδή σε αποθήκευση λίγων ονομάτων. Τέλος έγινε χρήση λίστας για την αποθήκευση των εταιριών, καθώς μπορεί να υπάρχει συνωνυμία άρα και το ίδιο όνομα να παρουσιάζεται σε πολλαπλές εταιρίες.

Όλες αναζητήσεις ονομάτων επιστρέφουν STL list(O(1) insert, O(n) προσπέλαση) με τους δείκτες των εταιριών, ενώ η αναζήτηση βάση αναγνωριστικού id, ένα δείκτη προς την εταιρία.

Πειραματικά αποτελέσματα

Για την εκτέλεση της πειραματικής διαδικασίας γίνεται χρήση της κλάσης Testable, καθώς και των βοηθητικών συναρτήσεων μέτρησης χρόνου στο helper namespace. Όταν βρισκόμαστε σε πειραματική λειτουργία τότε δηλώνουμε την σταθερά TEST_MODE στο Testable.h ως #define TEST_MODE_ON 1. Υπό την σταθερά αυτή μέσω συνθηκών του preprocessor εκτελούμε τον δοκιμαστικό κώδικα αναζητήσεων, μετρώντας συγκρίσεις και χρόνο εκτέλεσης. Με την χρήση των εντολών του preprocessor, δεν επιβαρύνουμε τον κώδικα μας με τον επιπρόσθετο δοκιμαστικό κώδικα στην κανονική του λειτουργία, διαχωρίζοντας έτσι τις λειτουργίες αυτές. Τέλος για τις πειραματικές μετρήσεις χρησιμοποιήθηκε το επισημασμένο αρχείο εισόδου data.csv[1]

Ενδεικτικά αποτελέσματα

Για κάθε αλγόριθμο αναζήτησης πραγματοποιείται ένα σύνολο από 10 μετρήσεις έτσι ώστε να πάρουμε μία καλή εκτίμηση περί της απόδοσης του κάθε αλγορίθμου. Παρακάτω παρουσιάζονται ομαδοποιημένα ανά τύπο αναζήτησης, τα αποτελέσματα για σύνολα αναζήτησης μεταβλητού μεγέθους. Σημειώνεται πως για αναζητήσεις με βάση το όνομα το μέγεθος των συνόλων αναζήτησης ήταν αρκετά μικρότερο από τα σύνολα αναζήτησης με βάση το ID λόγο των χρονοβόρων αναζητήσεων.

[1] Βλέπε Βιβλιογραφία

Αναζητήσεις αναγνωριστικού ID:

Search - Size	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Linear	168,40	345,50	510,70	691,80	878,60	1088,20	1236,10	1477,70	1536,00	1755,30
Binary	16,70	22,30	33,20	44,00	55,90	71,00	79,80	91,80	90,80	101,50
Interpolation	14,60	27,00	39,70	55,00	71,10	81,80	95,70	106,80	100,50	129,10
AVL	11,40	21,30	31,90	44,30	56,30	63,00	75,00	86,40	85,30	104,30

Πίνακας 1: Χρόνος εκτέλεσης σε ms συναρτήσει πλήθος αναζητήσεων

Search - Size	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Linear	2379147	4845753	7198431	9634683	12254553	14751390	17204331	19731562	22075439	24676470
Binary	6253	12563	18826	25092	31302	37455	43913	50237	56374	62655
Interpolation	3024	6293	9190	12544	15589	18562	22032	24835	28146	31453
AVL	6171	12475	18632	24816	30962	37140	43427	49700	55703	62038

Πίνακας 2: Πλήθος συγκρίσεων συναρτήσει πλήθος αναζητήσεων

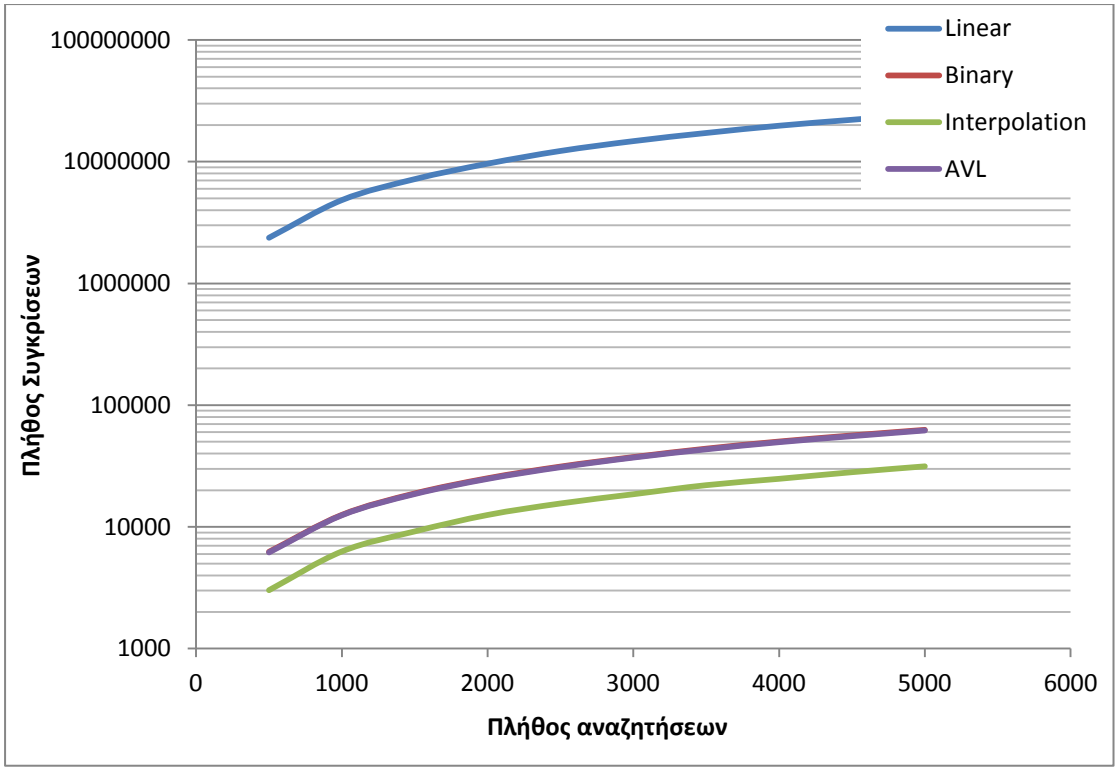
Αναζητήσεις ονόματος:

Search -Size	5	10	15	20	25	30	35	40	45	50
Trie	4,30	7,70	11,70	15,20	20,40	23,20	31,90	31,70	29,20	36,30
Linear	481,50	921,80	1389,40	1930,30	2320,90	2842,20	3222,50	3688,00	3588,30	4305,50

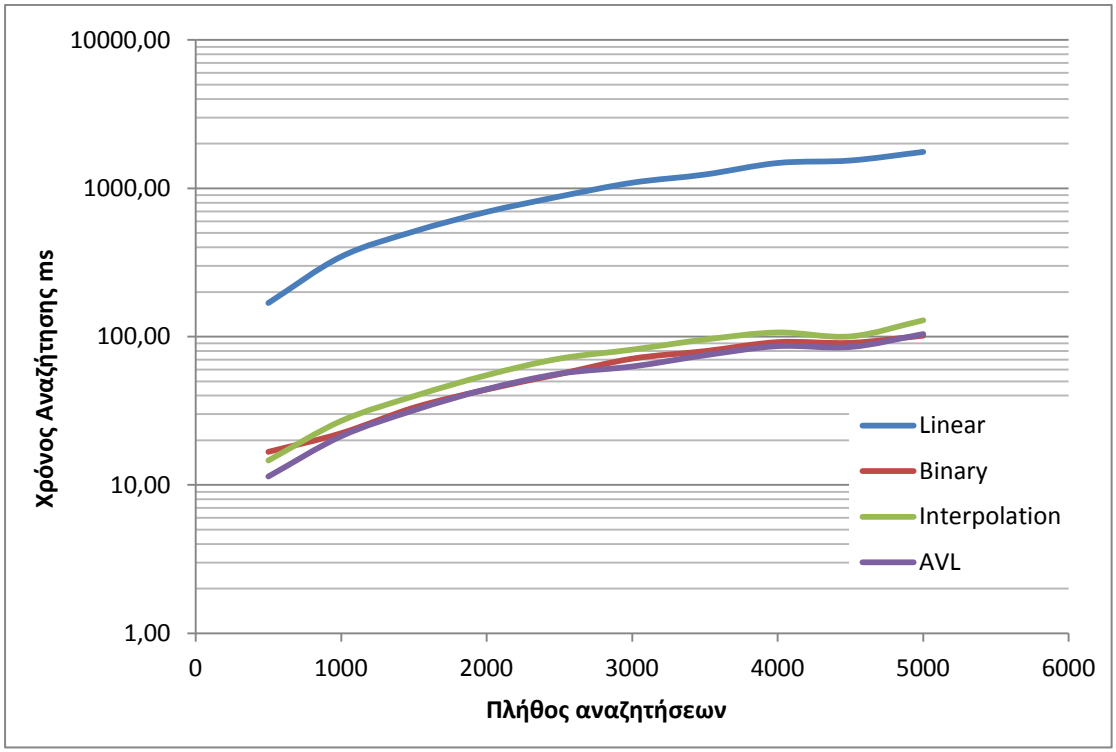
Πίνακας 3: Χρόνος εκτέλεσης σε ms συναρτήσει πλήθος αναζητήσεων

Search - Size	5	10	15	20	25	30	35	40	45	50
Trie	41	70	107	144	175	213	243	275	303	336
Linear	1600939	2665899	4087839	5508976	6664861	8132066	9241109	10442742	11464896	12709549

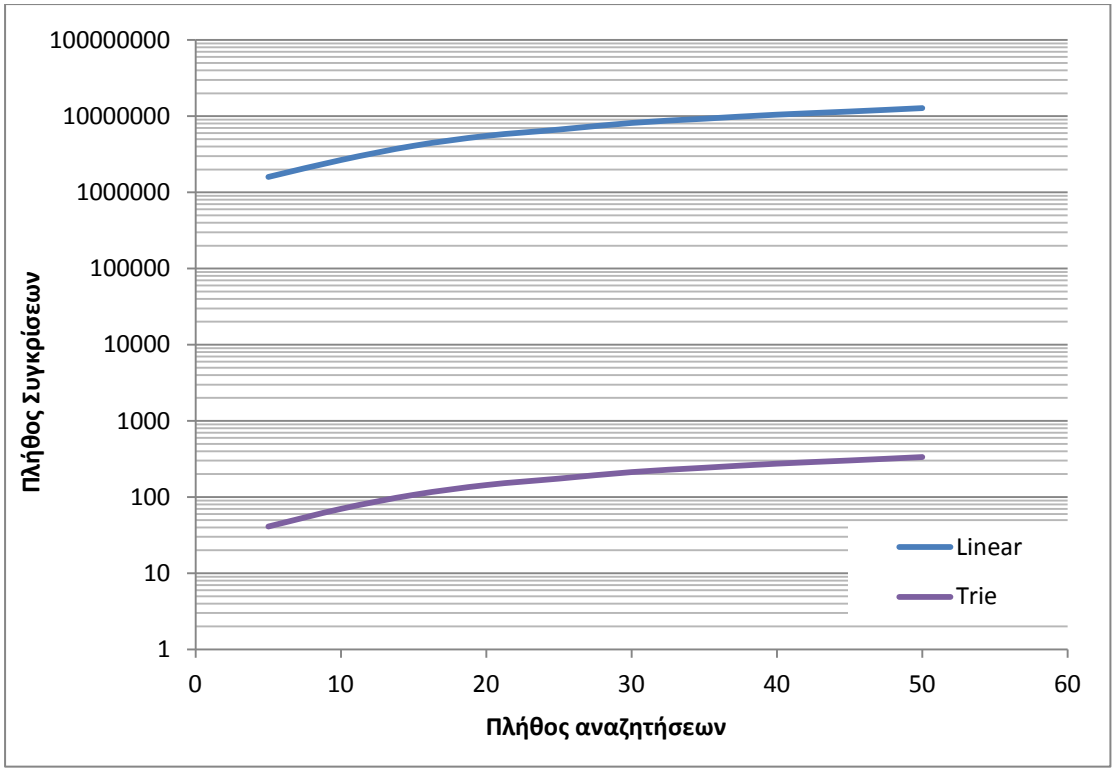
Πίνακας 4: Πλήθος συγκρίσεων συναρτήσει πλήθος αναζητήσεων



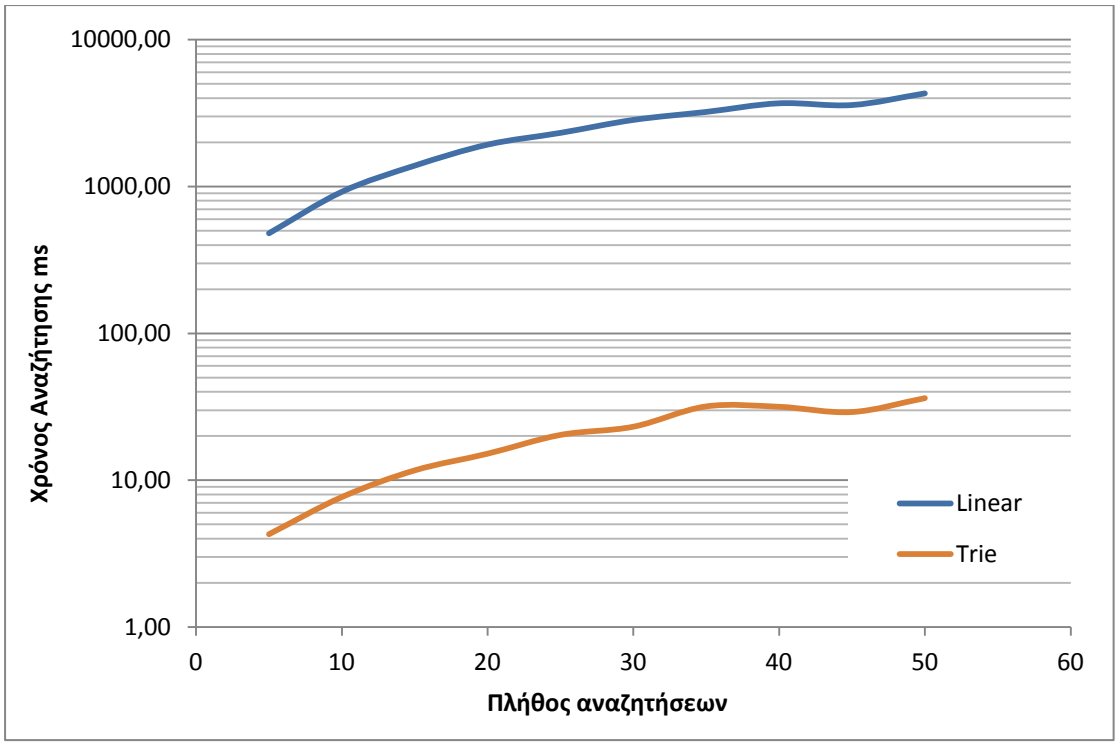
Γράφημα 1: Πλήθος συγκρίσεων συναρτήσει πλήθος αναζητήσεων



Γράφημα 2: Χρόνος εκτέλεσης σε ms συναρτήσει πλήθος αναζητήσεων



Γράφημα 3: Πλήθος συγκρίσεων συναρτήσει πλήθος αναζητήσεων



Γράφημα 4: Χρόνος εκτέλεσης σε ms συναρτήσει πλήθος αναζητήσεων

Ανάλυση Αποτελεσμάτων και Συμπεράσματα

Όλα τα παραπάνω γραφήματα βρίσκονται σε λογαριθμική κλίμακα, με βάση το 10 λόγο της μεγάλης αύξησης τιμών στις γραμμικές αναζητήσεις. Σημειώνεται πως στο γράφημα 1, οι πράξεις για AVL και Binary Search ταυτίζονται για αυτό και οι γραφικές αλληλεπικαλύπτονται. Παρακάτω θα γίνει μία θεωρητική ανάλυση της απόδοσης του κάθε αλγορίθμου και σύγκριση με τα πειραματικά μας αποτελέσματα.

Αναζητήσεις Βάση ID

Γραμμική Αναζήτηση

Η γραμμική αναζήτηση προσπελαύνει όλα τα στοιχεία ένα προς ένα μέχρις ότου να βρει το στοιχείο που αναζητείται. Στην χειρότερη περίπτωση που το στοιχείο δεν βρίσκεται στο σύνολο μας, επειδή δεν έχουμε καμία επίγνωση επί της δομής αυτού, θα προσπελάσουμε όλα τα στοιχεία του. Στην καλύτερη το στοιχείο μπορεί να βρεθεί κοντά στην αρχή και να μην χρειαστεί να προσπελάσουμε πολλά στοιχεία. Από τα παραπάνω είναι εύκολο να διαπιστώσουμε πως η πολυπλοκότητα του αλγορίθμου είναι γραμμική και ορίζεται ως $O(n)$, όπου n το πλήθος των στοιχείων.

Δυαδική Αναζήτηση

Στην δυαδική αναζήτηση, κάθε αναζήτηση γίνεται πάνω σε ένα ταξινομημένο σύνολο. Αρχίζοντας από την μέση του συνόλου υποδιπλασιάζουμε τον χώρο. Αν το στοιχείο δεν βρεθεί στην μέση, αναζητούμε το στοιχείο σε ένα από τα δύο μισά. Υποδιπλασιάζοντας έτσι τον χώρο σε κάθε αναζήτηση, είναι εύκολο να ορίσουμε την πολυπλοκότητα της δυαδικής αναζήτησης ως $O(log\, n)$, όπου n το πλήθος των στοιχείων.

Δυαδική Αναζήτηση Παρεμβολής

Στην δυαδική αναζήτηση παρεμβολής, η αναζήτηση γίνεται πάνω σε ένα ταξινομημένο σύνολο υπό την συνθήκη οι τιμές αυτού να είναι ισοκατανεμημένες. Ο μέσος χρόνος εκτέλεσης δίνεται ως $O(loglogn)$, ενώ ο χειρότερος $O(\sqrt{n})$, όπου n το πλήθος των στοιχείων.

Αναζήτηση AVL δένδρου

Το AVL δένδρο είναι ένα δυαδικό ισοζυγισμένο δένδρο. Αυτό μας εγγυάται πως κάθε κόμβος θα έχει το πολύ 2 παιδιά, αλλά και η διαφορά ύψους μεταξύ δύο παιδιών δεν ξεπερνάει το 1. Έτσι σε αντίθεση με το δυαδικό δένδρο αναζήτησης, μας εγγυάται μέσο χρόνο αναζήτησης ίσο με $O(log\ n)$, όπου n το πλήθος των στοιχείων.

Σύγκριση αποτελεσμάτων

Το επισυναπτόμενο αρχείο data.csv, στο οποίο και έγιναν όλες οι αναζητήσεις των διάφορων αλγορίθμων περιείχε n = 10.000 εγγραφές εταιριών. Από τον *πίνακα (2)* μπορούμε να επαληθεύσουμε τα αποτελέσματα μας. Έστω για size = 500 αναζητήσεις το μέσο πλήθος συγκρίσεων θα είναι:

- Γραμμική αναζήτηση: size * O(n) = 500 * 10000 = 5e6 > 2379147 άρα φράζεται από το θεωρητικό μέγιστο.

- Δυαδική αναζήτηση: size * O(logn) = 500 * 13,3 = 6650 > 6253 άρα φράζεται από το θεωρητικό μέγιστο.

- Δυαδική αναζήτηση παρεμβολής: size * O(loglogn) = 500 * 3.7 = 1850 < 3024 ενώ φράζεται από την χειρότερη περίπτωση της υλοποίησης μας, size * O($\sqrt{n}$) = 500*100 = 5e4. Στην παραπάνω μέτρηση αναμενόταν να προκύψουν καλύτερα αποτελέσματα, καθώς τα id στο αρχείο είναι καλά κατανεμημένα, χωρίς μεγάλες αποστάσεις μεταξύ τους. Βέβαια η πολυπλοκότητα παραπάνω περιέχει και την σταθερά 2.4 η οποία απλοποιείται από το big O notation. Αν συμπεριλάβουμε την σταθερά αυτή, ώστε να μετρήσουμε ακριβώς τις πράξεις της μέσης περίπτωσης προκύπτει: size * 2.4 * loglogn = 4440 > 3024 η οποία φράζει το αποτέλεσμα μας.

- Αναζήτηση σε AVL δένδρο: size * O(logn) = 500 * 13,3 = 6650 > 6171 άρα φράζεται από το θεωρητικό μέγιστο.

Ακόμη από τους χρόνους που παρουσιάζονται στον *πίνακα (1),* μπορούμε να διακρίνουμε την διαφορά τάξης μεγέθους της γραμμικής αναζήτησης από τις υπόλοιπες. Επίσης η δυαδική καθώς και το AVL δένδρο παρουσιάζουν πανομοιότυπο χρόνο με πολύ μικρή διαφορά όπως ακριβώς και στις πράξεις τους. Η αναζήτηση παρεμβολής αν και παρουσιάζει μικρότερο πλήθος συγκρίσεων, έχει μεγαλύτερο χρόνο αναζήτησης λόγο των πολλαπλών συνθηκών που εξετάζονται για την σωστή εκτέλεση του.

Αναζητήσεις ονόματος

Σύγκριση αλφαριθμητικών

Στο σημείο αυτό θα γίνει μία προσπάθεια να περιγραφεί η σύγκριση αλφαριθμητικών. Στην C++ που έγινε η υλοποίηση μας, τα αλφαριθμητικά αναπαριστώνται ως ένα πίνακα χαρακτήρων. Για την σύγκριση δύο αλφαριθμητικών στην καλύτερη περίπτωση που αναφέρονται στην ίδια θέση μνήμης τότε γνωρίζουμε πως είναι ίδια σε O(1) ή αν έχουν διαφορετικό μέγεθος, αν υποθέσουμε πως το διατηρούμε και δεν το υπολογίζουμε, πάλι ελέγχουμε αν είναι διαφορετικά σε O(1). Αν τώρα καμία από τις παραπάνω συνθήκες δεν ισχύει, δηλαδή έχουμε ίδιο μέγεθος και ανήκουν σε διαφορετικές θέσεις μνήμης, η σύγκριση απαιτεί στην χειρότερη γραμμικό χρόνο O(k) όπου k το μήκος του αλφαριθμητικού. Στις μετρήσεις μας υποθέσαμε τον απλοϊκό τρόπο σύγκρισης που απαιτεί γραμμικό χρόνο πάντα.

Γραμμική Αναζήτηση

Όπως και αναφέρθηκε και παραπάνω η γραμμική αναζήτηση στην χειρότερη περίπτωση θα διατρέξει όλα τα στοιχεία έτσι ώστε να ολοκληρωθεί η αναζήτηση. Πλέον η αναζήτηση γίνεται σε κάθε υπάλληλο της εταιρίας, κάθε εταιρία μπορεί να περιέχει από 1 έως 7 υπαλλήλους, κατά μέσω όρο 4 υπαλλήλους, πράγμα που μεγαλώνει τον χώρο μας από σε m = 4n, όπου n το πλήθος των εταιριών. Άρα συνολικά ο χρόνος αναζήτησης ενός ονόματος μεγέθους k θα ισούται με mk, δηλαδή O(mk).

Αναζήτηση Trie

Η δόμηση του κάθε επιπέδου Trie, στην υλοποίηση μας, έγινε με χρήση STL Maps. Η αναζήτηση ενός γράμματος σε κάθε επίπεδο δηλαδή, παίρνει χρόνο logx, όπου x το μέγεθος του λεξικού μας, και εφόσον αποθηκεύουμε αγγλικά ονόματα x = 26, άρα πράξεις εύρεσης γράμματος στο επίπεδο ίσες με log26 = 4,7. Εφόσον το αλφαριθμητικού έχει μεγέθους k οι συγκρίσεις εύρεσης του είναι 4,7k άρα τελικά μία αναζήτηση απαιτεί O(k) πράξεις.

Σύγκριση αποτελεσμάτων

Για τις αναζητήσεις χρησιμοποιήθηκε το ίδιο αρχείο εισόδου όπως και παραπάνω, αλλά σε μικρότερο σύνολο αναζήτησης, λόγο του πολύ μεγάλου χώρου που δημιουργείται. Επειδή οι αναζητήσεις γίνονται με ονόματα τυχαίου μεγέθους από το σύνολο μας όμως θα πάρουμε ως μέγεθος k = 25 το μέγιστο μεγέθους ονόματος, σε size = 5 αναζητήσεων.

- Γραμμική αναζήτηση: size * O(mk) = 5 * 40000 * 25 = 5e6 > 1600939 άρα φράζεται από το θεωρητικό μέγιστο.

- Αναζήτηση Trie: size * O(k) = 5 * 25 = 125 > 41 άρα φράζεται από το θεωρητικό μέγιστο.

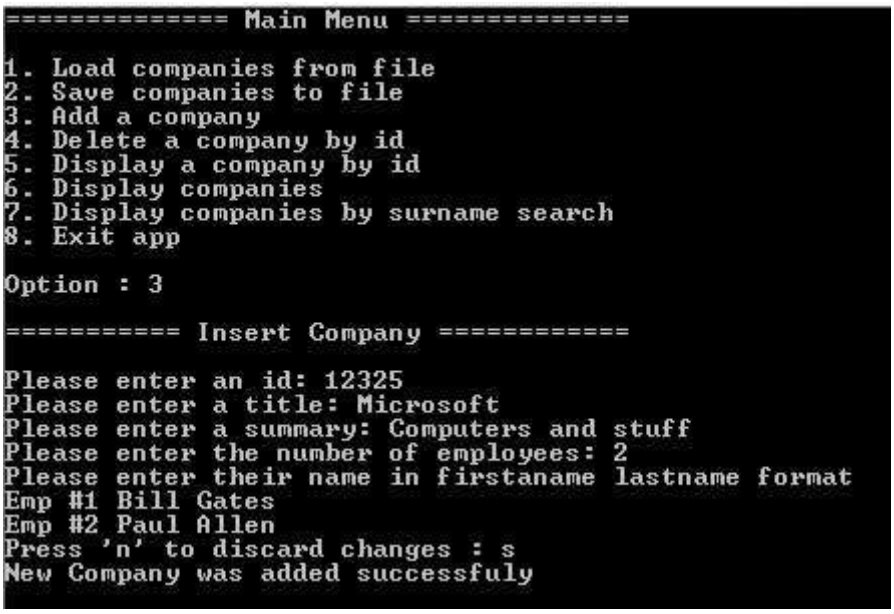
Ακόμη *από τον πίνακα (3)* παρατηρούμε την εξαιρετικά μεγάλη διαφορά των δύο υλοποιήσεων και σε θέμα χρόνου όπως και αναμενόταν.

Παραδείγματα χρήσης

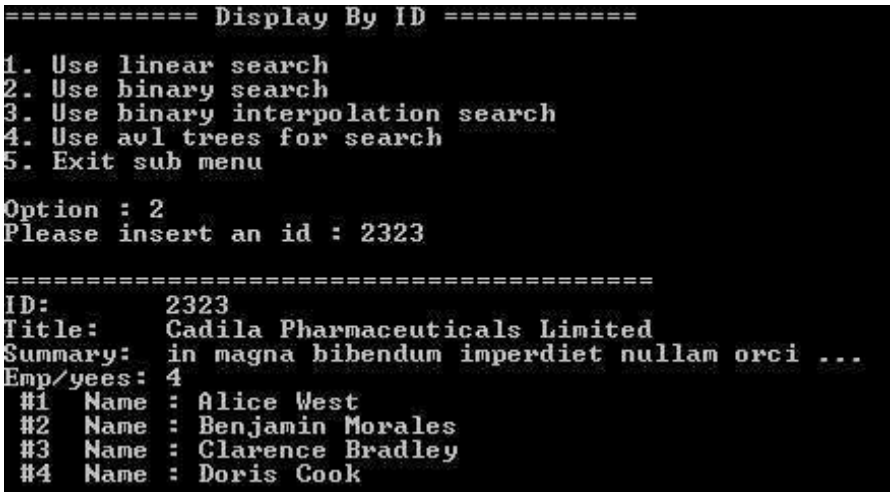
Στα αρχεία του παραδοτέου περιέχεται αρχείο small.csv, το οποίο περιέχει μερικές εγγραφές. Με την εκκίνηση του προγράμματος από το μενού λειτουργιών μπορούμε να εισάγουμε τις εγγραφές αυτές by default. Για να φορτώσουμε τις δικές μας εγγραφές παρέχουμε ως όρισμα στο εκτελέσιμο το νέο αρχείο, σύροντας το πάνω στο εκτελέσιμο ή με την χρήση του command line, όπως φαίνεται και παρακάτω:

Εικόνα 1: Φόρτωση αρχείου data.csv

Με την φόρτωση των εγγραφών από το αρχείο μπορούμε να εισάγουμε νέες, είτε να διαγράψουμε κάποιες επιλέγοντας τις αντίστοιχες επιλογές. Για την αποθήκευση των αλλαγών στο αρχείο πρέπει να επιλεχθεί η επιλογή 2, όπου και θα γίνει τελικά η αποθήκευση στο δίσκο.



Εικόνα 2: Εισαγωγή νέας εγγραφής



Εικόνα 3: Αναζήτηση εγγραφής

Τέλος μετά την σωστή χρήση είναι σημαντικό να τερματίζουμε κανονικά την εφαρμογή ώστε να αποδεσμεύεται η δυναμικά δεσμευμένη μνήμη από τις δομές μας.

Βιβλιογραφία

Αρχείο εισόδου data.csv : *http://mmlab.ceid.upatras.gr/attachments/article/95/data.csv*

Δομές Δεδομένων : *Αθανάσιος Τσακαλίδης*, Πάτρα 2012