Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

Τελική Έκθεση
για την εργασία του μαθήματος
Αλγόριθμοι και Δομές Δεδομένων

Κατασκευή μιας απλής μηχανής αναζήτησης.

Θωμάς Ρεπαντής
Έτος: Δ′
Α.Μ.: 4218
E-mail: darkzero@otenet.gr

Πάτρα, 17.1.2001

Εισαγωγή. Σκοπός της εργασίας είναι η κατασκευή μιας απλής μηχανής αναζήτησης (search engine). Αυτή περιλαμβάνει το σχεδιασμό και την υλοποίησή της. Απαραίτητη προϋπόθεση είναι προφανώς η ανάλυση, μελέτη και κατανόηση των εμπλεκόμενων διαδικασιών, προκειμένου να επιλεγούν οι κατάλληλες δομές δεδομένων και αλγόριθμοι και κατόπιν των μεθόδων υλοποιήσης στη γλώσσα προγραμματισμού που απαιτήθηκε (C). Ακολουθεί ο σχεδιασμός της μηχανής αναζήτησης και τέλος η υλοποίησή της.

Περιγραφή των ενεργειών. Αρχικά κατανοήθηκε και μελετήθηκε το θέμα. Κατόπιν αναλύθηκαν οι απαιτούμενες διαδικασίες. Στη συνέχεια μελετήθηκαν οι μέθοδοι ανάπτυξης (δομές δεδομένων και αλγορίθμοι) και οι προσφερόμενοι τρόποι υλοποίησης από τη γλώσσα. Ακολούθησε ο σχεδιασμός της εφαρμογής και έπειτα η υλοποίησή της στη C. Πιο συγκεκριμένα:

Κατανόηση θέματος. Η απλή μηχανή αναζήτησης που κληθήκαμε να κατασκευάσουμε θα λαμβάνει ως είσοδο αρχεία κειμένου, θα βρίσκει τις λέξεις που περιέχουν και θα τις διατηρεί στη μνήμη του υπολογιστή. Κατόπιν ο χρήστης θα εισάγει στην εφαρμογή μια λέξη ή ένα λογικό συνδυασμό λέξεων και η εφαρμογή θα τον πληροφορεί σε ποια αρχεία βρίσκεται η λέξη ή ο συνδυασμός αυτός.

Απαιτούμενες διαδικασίες. Ουσιαστικά οι βασικές απαιτούμενες διαδικασίες είναι οι εξής:

1. Ανάγνωση από τα αρχεία.
2. Σάρωση των λέξεων.
3. Αποθήκευση των λέξεων καθώς και λοιπών πληροφοριών (π.χ. αρχείο στο οποίο ανήκουν) σε μια δομή δεδομένων στη μνήμη του υπολογιστή.
4. Ανάγνωση της επιθυμίας του χρήστη και της λέξης ή του συνδυασμού λέξεων που αναζητά.
5. Αναζήτηση στη δομή για λέξη ή συνδυασμό λέξεων και ενημέρωση του χρήστη.

Μέθοδοι υλοποίησης. Βασική απαίτηση από τη δομή δεδομένων που θα επιλέξουμε είναι βέβαια η ταχύτητα κατασκευής της (αποθήκευσης των λέξεων σε αυτή), αλλά κυρίως η ταχύτητα αναζήτησης. Με βάση μια εκτεταμένη μελέτη στη βιβλιογραφία, έγινε φανερό ότι υπάρχουν αρκετές κατηγορίες δομών δεδομένων που θα μπορούσαν να καλύψουν την απαίτηση αποθήκευσης των στοιχείων στη μνήμη, η καθεμία με ξεχωριστά πλεονεκτήματα και μειονεκτήματα. Ξεχωρίσαμε μερικές βασικές:

1. Δένδρα (trees). Απλά δυαδικά δένδρα αναζήτησης, όπως και άλλες απλές δομές (λίστες, ουρές προτεραιότητας, σωροί) φάνηκε να μην επαρκούν, μιας και ο χρόνος εισαγωγής στοιχείου και αναζήτησης είναι μεγάλος (O(h), όπου h το ύψος του δένδρου, το οποίο στην περίπτωση μας είναι μεγάλο, λόγω του πλήθους των λέξεων και ούτως ή άλλως δεν μπορεί να είναι καλύτερο από το $O(log_2 n)$ του ισορροπημένου δυαδικού δένδρου αναζήτησης). Μια βελτίωση στην ταχύτητα θα μπορούσαν να προσφέρουν ειδικές μορφές ισορ- ροπημένων δένδρων, όπως τα AVL-trees, τα 2-3 trees ή τα red-black trees. Ωστόσο η πολυπλοκότητα στην υλοποίηση των απαιτούμενων διαδικασιών (π.χ. της επανεξισορρόπησης στα δένδρα AVL) συνεπάγεται πιθανώς και καθυστέρηση στην κατασκευή και έτσι αποφύγαμε και αυτές τις εξελιγμένες μορφές.

2. Tries. Η γενίκευση αυτή των δυαδικών δένδρων, κατά την οποία η διακλαδώσεις σε κάθε επίπεδο καθορίζονται όχι από ολόκληρη την τιμή του κλειδιού, αλλά μόνο από ένα μέρος της (π.χ. στην περίπτωσή μας από τα πρώτα γράμματα και όχι απ' όλη τη λέξη) παρουσιάζει σχετικά απλή υλοποίηση και πιθανά θα μπορούσε να χρησιμοποιηθεί. Η σπανιότητα των εφαρμογών και η ελλειπής βιβλιογραφία όμως μας απέτρεψαν. Επιπλέον για μεγάλο πλήθος λέξεων και ανάλογα με την ακριβή υλοποίηση (π.χ. αλφαβητική ή μη ταξινόμηση και έως ποιο βάθος) πιθανά οι χρόνοι να μην ήταν οι πλέον ικανοποιητικοί. Για παράδειγμα ο χρόνος αναζήτησης είναι γενικά O(l), όπου l ο αριθμός των επιπέδων του trie.

3. Κατακερματισμός (Hashing) Η τεχνική του κατακερματισμού μας επιτρέπει να κατανείμουμε τα στοιχεία που θέλουμε να αποθηκεύσουμε στις θέσεις ενός πίνακα, χρησιμοποιώντας μια συνάρτηση κατακερματισμού, η οποία μετασηματίζει τα κλειδιά σε δείκτες του πίνακα. (Προφανώς η άμεση αποθήκευση σε απλό πίνακα με θέσεις όσες και τα κλειδιά, δίχως συνάρτηση κατακερματισμού, απορρίπτεται μιας και θα οδηγούσε σε μεγάλο πίνακα, όπου η αναζήτηση θα έπρεπε να γίνεται σειριακά.) Βέβαια χρειάζεται μια μέθοδος χειρισμού περιπτώσεων όπου στην ίδια θέση

του πίνακα κατατάσσονται περισσότερα από ένα κλειδιά (overflows/ collisions - που υπάρχουν όπως αποδεικνύει το Birthday Paradox). Με χρήση πάντως της τεχνικής του hashing μπορούμε να επιτύχουμε στην καλύτερη περίπτωση (όταν δεν υπάρχει σύγκρουση) άμεση αποθήκευση και ανάκτηση (τάξης 1). Αυτά τα πλεονεκτήματα στην ταχύτητα, σε συνδυασμό με το γεγονός ότι ο κατακερματισμός είναι μια διαδεδομένη και δοκιμασμένη λύση σε προβλήματα όπως και το δικό μας, μας οδήγησαν στην επιλογή του ως μέθοδο δόμησης των δεδομένων μας. Αναλύουμε παρακάτω την επιλογή του είδους του κατακερματισμού (άρα και του αντίστοιχου πίνακα), της κατάλληλης συνάρτησης κατακερματισμού και της κατάλληλης μεθόδου χειρισμού των συγκρούσεων.

Είδος κατακερματισμού. Θα λέγαμε πως γενικά έχουμε να επιλέξουμε ανάμεσα σε:

1. Στατικό κατακερματισμό (static hashing). Στην περίπτωση αυτή ο πίνακας που χρησιμοποιείται είναι σταθερού μεγέθους.
2. Δυναμικό κατακερματισμό (dynamic or extendible hashing). Είτε χρησιμοποιώντας direcotories (αντίστοιχη περίπτωση με τα tries), είτε όχι έχουμε στην περίπτωση αυτή μεταβαλλόμενο αποθηκευτικό χώρο, ανάλογα με τα δεδομένα.

Προκειμένου να αποφύγουμε μια υπερβολικά πολύπλοκη υλοποίηση, καθώς και καθυστέρηση κατά την κατασκευή της δομής και μιας και ο δεσμευόμενος χώρος δεν είναι απαγορευτικός, προτιμήσαμε το **στατικό κατακερματισμό**.

Συνάρτηση κατακερματισμού (hash function). Μια ιδανική συνάρτηση κατακερματισμού καταναλώνει λίγο χρόνο από τον επεξεργαστή και διασπείρει ομοιόμορφα τα κλειδιά. Πολλές προτάσεις υπάρχουν, ανάμεσα στις οποίες και οι εξής:

1. Mid-square. Η διεύθυνση προορισμού υπολογίζεται παίρνοντας ορισμένα bits από το μέσο του τετραγώνου του προσδιοριστή του κλειδιού. Απαιτεί μέγεθος του πίνακα δυνάμεως του 2.
2. Division. Χρησιμοποιούμε τον τελεστή του υπολοίπου modulus στη διαίρεση του προσδιοριστή του κλειδιού με έναν πρώτο αριθμό για να υπολογίσουμε τη διεύθυνση προορισμού.

3. Folding. Χωρίζουμε τον προσδιοριστή του κλειδιού σε μέρη τα οποία προσθέτουμε για να πάρουμε τη διεύθυνση προορισμού.

4. Digit Analysis. Χρησιμοποιείται σε περιπτώσεις όπου -αντίθετα με το δικό μας πρόβλημα- τα κλειδιά είναι όλα εκ των πρωτέρων γνωστά (perfect hashing). Η διεύθυνση παίρνεται από τα ψηφία που προκύπτουν από το μετασχηματισμό των προσδιοριστών σε αριθμούς χρησιμοποιώντας κάποια βάση r.

Δοκιμάζοντας πολύπολοκες συναρτήσεις κατακερματισμού βασιζόμενες σε λογικές πράξεις πάνω σε bits, οι οποίες είναι ταχύτερες από τον τελεστή του υπολοίπου (λόγω υπερβολικού αριθμού συγκρούσεων- ίσως και από κακή επιλογή του μεγέθους του πίνακα κατακερματισμού. Καταφύγαμε έτσι στη συνάρτηση που βασίζεται στη **διαίρεση**, που είναι και πολύ διαδεδομένη για γενικού σκοπού εφαρμογές, βελτιωμένη, αφού προσθέσαμε **ολίσθηση**. Η ταχύτητα υπολογισμού των διευθύνσεων και ο αριθμός των συγκρούσεων που προέκυψαν ήταν για την εφαρμογή μας ικανοποιητικός. Επιλέξαμε φυσικά πρώτο αριθμό για μέγεθος του πίνακα και διαιρέτη.

Μέθοδος χειρισμού των συγκρούσεων. Έχουμε -ανάμεσα σε άλλεσδύο βασικές μεθόδους για να αποθηκεύσουμε τα στοιχεία τα οποία βρίσκουν τη θέση τους κατειλημμένη από άλλα:

1. Linear open addressing or linear probing. Αποθηκεύουμε το συγκρουσθέν στοιχείο σε μια επόμενη ελεύθερη θέση στον πίνακα.

2. Chaining. Σε κάθε θέση στον πίνακα ξεκινούν διασυνδεδεμένες λίστες, έτοιμες να δεχθούν τα συγκρουόμενα στοιχεία.

Αν και η πρώτη μέθοδος είναι πιο εύκολα υλοποιήσιμη δεν έχει τόσο καλά αποτελέσματα, μιας και η εισαγωγή μπορεί να καθυστερήσει πολύ αναζητώντας την επόμενη ελεύθερη θέση, ενώ και η αναζήτηση καταλήγει σε περισσότερες συγκρίσεις. Επιλέξαμε έτσι τη δεύτερη μέθοδο (**chaining**).

Προσφερόμενα εργαλεία υλοποίησης. Χρησιμοποιώντας τη γλώσσα C και τις συναρτήσεις που προσφέρει υλοποιήσαμε τα βασικά σημεία του προγράμματος. Πιο συγκεκριμένα: Ανοίξαμε και κλείσαμε τα αρχεία χρησιμοποιώντας τις fopen και fclose, η οποίες αν και ίσως λίγο αργότερες από τις open και close συνίστανται για απλές εργασίες όπως η ανάγνωση. Η σάρωση

των λέξεων έγινε όχι διαβάζοντας χαρακτήρα - χαρακτήρα, αλλά διαβάζοντας ολόκληρα strings (γραμμές) και διαχωρίζοντας τις λέξεις με τη strsep (reentrant BSD απόγονο της strtok). Η λύση αυτή αν και μας στέρησε τη φορητότητα προτιμήθηκε προκειμένου να έχουμε λιγότερες προσβάσεις στα αρχεία και άρα λιγότερη καθυστέρηση. Η αποθήκευση των λέξεων έγινε σε δομές, που περιλάμβαναν τη λέξη και το αρχείο στο οποίο ανήκει. Κάθε λέξη αποθηκεύτηκε μία φορά για κάθε αρχείο, για εξοικονόμηση χώρου και χρόνου, ενώ οι λέξεις απαλλάχθηκαν από κόμματα, τελείες κτλ. αν και διατηρήσαμε πεζά και κεφαλαία. Οι δομές αποθηκεύτηκαν σε διασυνδεδεμένες λίστες που ξεκινούσαν από πίνακα σταθερού μεγέθους, στον οποίο διευθύνσεις έδινε η συνάρτηση κατακερματισμού. Πρώτο αριθμό κοντινό στο μέγεθος του πίνακα που θέλαμε επιλέξαμε με το πρόγραμμα nextprime.c που επισυνάπτουμε. Η αναζήτηση γίνεται όπως και η εισαγωγή με βάση τη συνάρτηση κατακερματισμού, που μας οδηγεί στην κατάλληλη θέση στον πίνακα bucket και κατόπιν σειριακά στη λίστα, αν έχουμε σύγκρουση. Δώσαμε στο χρήστη τη δυνατότητα εισαγωγής λογικών εκφράσεων αναζήτησης τύπου λέξη1 ΚΑΙ λέξη2. Η λογική έκφραση Ή υποστηρίζεται έμμεσα με την αναζήτηση δύο λέξεων μαζί.

Σχεδιασμός της εφαρμογής. Με βάση τις παραπάνω βασικές διεργασίες που η μηχανή αναζήτησης καλείται να εκτελεί διακρίναμε τις παρακάτω συναρτήσεις και τμήματα του προγράμματος που ομαδοποιήσαμε σε αρχεία:

seng.h
Δηλώσεις συναρτήσεων, ονομάτων τύπων δεδομένων και καθολικών μεταβλητών.

main.c
Συνάρτηση:
main

engine.c
Συναρτήσεις:
ReadFile
SearchFor

hashtab.c
Συναρτήσεις:
InitializeTable
Find

Insert
PrintTable
DestroyTable
Hash

Υλοποίηση της εφαρμογής. Ο κώδικας ελέγχθηκε με το μεταγλωττιστή gcc (GNU Compiler Collection) version 2.95.2 στα λειτουργικά συστήματα: FreeBSD 4.2-STABLE και Linux 2.2.17. Δυστυχώς η χρήση της strsep, προκειμένου να αυξηθεί η ταχύτητα όπως εξηγήσαμε, δε μας επέτρεψε τη δοκιμή σε Microsoft Windows. Η άδεια κάτω από την οποία τέθηκε ο κώδικας επιλέχθηκε να είναι η GNU General Public License. Ακολουθούν σύντομες οδηγίες χρήσης του εκτελέσιμου κώδικα, παρμένες από τη βοήθεια που δίνει η εφαρμογή στο χρήστη:

Usage: seng [ASCII text files] seng takes a list of ASCII text files as an input, stores them in the memory, prompts for a word or a logical combination of words to search for and returns the name(s) of the file(s) containing this or those words (commas etc. are ignored and the search is case sensitive). You can enter a word to search for, or .AND to search for word1 AND word2, or .p to print Hash Table status, or .h for help, or .q to quit.

Συμπεράσματα. Ξεπερνώντας τα προβλήματα επιλογής της κατάλληλης δομής δεδομένων και αλγορίθμου, κατανόησης του τρόπου υλοποίησης τους στη C και οργάνωσης και δόμησης του κώδικα σε συναρτήσεις, τμήματα και αρχεία, καταφέραμε να υλοποιήσουμε μια μηχανή αναζήτησης με συγκριτικά πολύ καλές επιδόσεις στην ταχύτητα. (Ενδεικτικά αναφέρουμε φόρτωση 15MB κειμένου σε λιγότερο από 10 δευτερόλεπτα και αναζήτηση ακαριαία σε Pentium II - 266MHz.) Προφανώς οι εμπειρίες που αποκτήθηκαν είναι πολλές και σημαντικότατες. Ενδεικτικά αναφέρουμε την εξοικείωση με τις διάφορες δομές δεδομένων και αλγορίθμους, τη σε βάθος κατανόηση αυτών που επιλέχθησαν, την πείρα που αποκτήθηκε στον ορισμό προβλημάτων, στην ανάλυση και το σχεδιασμό ολοκληρωμένων -έστω και μικρού μεγέθουσ- εφαρμογών, στην προγραμματιστική αντιμετώπιση πραγματικών προβλημάτων, καθώς βέβαια και την καλύτερη κατανόηση και την πρακτική εξοικείωση με τις κατασκευές της γλώσσας C, ειδικά αυτές που αφορούν τρόπους υλοποίησης δομών δεδομένων και αλγορίθμων.

Αναφορές .

1. N. Wirth, ¨Αλγόριθμοι και Δομές Δεδομένων¨, 1990
2. B. Kernighan - D. Ritchie, ¨Η γλώσσα προγραμματισμού C (Δεύτερη Έκδοση)¨, 1990
3. E. Horowitz - S. Sahni - S. Anderson-Freed, "Fundamentals of Data Sructures in C", 1993
4. M. A. Weiss, "Data Structures and Algorithm Analysis in C (Second Edition)", 1997
5. R. Sedgewick, "Algorithms in C: Software Engineers, and Software Reuse", 1990
6. D. Knuth, "The Art Of Computer Programming: Sorting and Searching", 1973
7. Κ. Θραμπουλίδης, ¨Από τη C στην Java - Από τον διαδικαστικό στον object-oriented προγραμματισμό¨, 1999
8. H.M. Deitel - P.J. Deitel, "C: How to Program (Second Edition)", 1994
9. A. Friedman - L. Klander - M. Michaelis - H. Schildt, "C/C++ Annotated Archives", 1999
10. A. Binstock, "Hashing Rehashed. Is RAM speed making your hashing less efficient?", Dr. Dobb's Journal, 2001
11. R. J. Jenkins Jr., "Hash Functions for Hash Table Lookup", Dr. Dobb's Journal, 2000

Συνημμένα .

Επισυνάπτεται ο πηγαίος κώδικας σε μορφή κειμένου για τις εφαρμογές της εργασίας αυτής.

Επίσης σε δισκέττα επισυνάπτονται το παρόν κείμενο σε ηλεκτρονική μορφή, καθώς και ο πηγαίος και ο εκτελέσιμος (FreeBSD και Linux) κώδικας για τις εφαρμογές της εργασίας αυτής.