Όνομα: Ανδρέας Σπανόπουλος
ΑΜ: 1115201700146

1. Σύντομη Περιγραφή

Η εργασία έχει υλοποιηθεί σε C++. Τα αρχεία κώδικα έχουν οργανωθεί σε modules για να διευκολύνεται η ανάγνωση τους, καθώς και για να γίνει χρήση separate compilation.
Οι προδιαγραφές και οι περιορισμοί που επιβάλλει η εκφώνηση έχουν τηρηθεί.
Το πρόγραμμα έχει δοκιμαστεί στα linux workstations (με g++) του τμήματος και δουλεύει κανονικά.
Το εκτελέσιμο ονομάζεται runelection.
Ο κώδικας έχει αναπτυχθεί στον κειμενογράφο Atom. Κατά συνέπεια ο κώδικας και τα σχόλια έχουν ευθυγραμμιστεί βάσει των default ρυθμίσεων του Atom. Θα πρότεινα να χρησιμοποιηθεί ο συγκεκριμένος κειμενογράφος για την ανάγνωση του κώδικα.

2. Επεξήγηση Αρχείων

Το αρχείο runelection.cpp είναι το πηγαίο που περιέχει τη main().
Στα αρχεία bloom_filter.cpp, red_black_tree.cpp και catalogue.cpp (και κατά συνέπεια list.cpp) περιέχονται οι αντίστοιχες δομές δεδομένων που ζητούνται στην εκφώνηση.
Στα αρχεία main_funcs.cpp και string_funcs.cpp περιέχονται διάφορες βοηθητικές συναρτήσεις που έχω υλοποιήσει (πχ. συναρτήσεις για ανάγνωση input, συναρτήσεις για διαχείριση συμβολοσειρών, κλπ).
Στο αρχείο voter.cpp υπάρχει η δομή voter και όλες οι συναρτήσεις που σχετίζονται με αυτήν.
Ο φάκελος Test_Files περιέχει τα registries που μας δίνονται στο site K22 του μαθήματος.
Και τέλος υπάρχει το αρχείο makefile, το οποίο όπως ήδη γνωρίζετε χρησιμεύει στη σύντομη μεταγλώττιση του προγράμματος, και σύντομα διαγραφή του εκτελέσιμου και των αντικείμενων αρχείων.

3. Επεξήγηση του runelection.cpp

Το αρχείο αυτό περιέχει τη main(). Οπότε θα εξηγήσω τη λειτουργία της main(). Στην αρχή ελέγχεται η ορθότητα των παραμέτρων. Έπειτα γίνεται η αποτίμησή τους με τη συνάρτηση
get_parameters(). Η συνάρτηση αυτή αναθέτει το όνομα του αρχείου registry στη μεταβλητή char* inputfile, το όνομα του αρχείου εξόδου στη μεταβλητή char* outfile, και τη τιμή
numofupdates (αν αυτή υπάρχει) στη μεταβλητή unsigned int numofupdates.
Έπειτα με τη συνάρτηση get_voters() χτίζουμε τη σύνθεση δομών, διαβάζοντας τα recods από το inputfile.
Στη συνέχεια μπαίνουμε σε βρόχο do {} while (), για να εκτελέσουμε τα commands που θα δίνει ο χρήστης. Αν δοθεί command με μη-γνωστό format ή αν δοθεί το command exit, τότε

βγαίνουμε από το βρόχο επανάληψης, αποδεσμεύεται σταδιακά η μνήμη και το πρόγραμμα τερματίζει. Η αποδέσμευση γίνεται με τη χρήση του destructor της κάθε δομής.

4. Επεξήγηση σύνθεσης δομών

4.1 Bloom Filter

Το Bloom Filter έχει υλοποιηθεί ακριβώς όπως το ζητάει η εκφώνηση:

α) Ένα array από Bits, όπου το μέγεθος του array είναι ο πρώτος πρώτος αριθμός που είναι μεγαλύτερος από το τριπλάσιο αριθμό των records που έχουμε εισαγάγει στο bloom filter.

β) Τρεις συναρτήσεις κατακερματισμού (MurmurHash3_x86_32(), FNV32(), hash_func_3()) που είναι ομοιόμορφα κατανεμημένες (δυστυχώς δεν είναι ανεξάρτητες, αφού για το τρίτο hash function έχει χρησιμοποιηθεί ότι hi(x) = h1(x) + i * h2(x), με i = 3).

Παραθέτω από κάτω τα links που με βοήθησαν στο να υλοποιήσω το Bloom Filter:

1) http://www.mathcs.emory.edu/~cheung/Courses/255/Syllabus/1-C-intro/bit-array.html
2) http://llimllib.github.io/bloomfilter-tutorial/
3) https://github.com/PeterScott/murmur3
4) http://isthe.com/chongo/tech/comp/fnv/
5) Το pdf που μας δίνεται στην ιστοσελίδα του μαθήματος

Καταρχάς, το Bloom Filter δημιουργείται πάντα από το Red Black Tree, καθώς αυτό περιέχει όλες τις εγγραφές ανά πάσα στιγμή.

Μετά τη δημιουργία (ή αναδημιουργία) του Bloom Filter, όταν κάνουμε numofupdates εισαγωγές/διαγραφές, το bloom filter καταστρέφεται, και αναδημιουργείται με τη βοήθεια του Red Black Tree.

Το array of bits, έχει υλοποιηθεί με ένα πίνακα από int, όπου για κάθε int διαχειριζόμαστε ξεχωριστά το καθένα από τα 32 του bits. Ακριβώς όπως προτείνει το πρώτο link που παρέθεσα. Έχω βάλει σχόλια για να φανεί πως καταννοώ τον κώδικα που "πήρα".

Η πρώτη συνάρτηση κατακερματισμού είναι η MurmurHash3_x86_32(). Θα είμαι ειλικρινής. Ουσιαστικά η συνάρτηση δεν έχει κάτι

δύσκολο για να καταλάβεις. Αυτό που την διαφοροποιεί από τις υπόλοιπες συναρτήσεις κατακερματισμού είναι οι επιλογές των
σταθερών. Απ' όσο καταλαβαίνω, οι σταθερές αυτές έχουν προκύψει μετά από εκτενές testing. Οπότε απλά τις πήρα και τις έβαλα
στην υλοποίηση που έκανα.
Η δεύτερη συνάρτηση κατακερματισμού είναι η FNV32(). Αυτή ξεκινάει με ένα offset, στη συνέχεια το κάνει xor (^) με κάθε
χαρακτήρα του string key, και μετά το πολλαπλασιάζει με έναν πρώτο που έχει φανεί να βελτιστοποιεί τη διαδικασία αυτή. Αρκετά απλή.
Για την τρίτη συνάρτηση κατακερματισμού έχω επιλέξει h3(x) = h1(x) + 3 * h2(x), αφού στο pdf που μας έχει δοθεί στο site του μαθήματος
αναφέρει πως αυτή η επιλογή φέρνει καλά αποτελέσματα.
Γενικά, ο λόγος για τον οποίο επέλεγα τις συγκεκριμένες συναρτήσεις κατακερματισμού είναι επειδή διάβασα σε πολλά άρθρα (κυρίως στο
wikipedia) πως αυτές είναι βέλτιστες. Έκανα και δικά μου test, στα οποία έφτιαχνα πίνακες και εκτύπωνα τα περιεχόμενά τους. Όντως, τα
collisions ελαχιστοποιούνταν όταν χρησιμοποιούσα αυτές τις συναρτήσεις.
Σχετικό άρθρο: https://en.wikipedia.org/wiki/Hash_function

4.2 Red Black Tree

Καταρχάς, η υλοποίησή μου του μελανέρυθρου δένδρου έχει "εμπνευστεί" από το βιβλίο "Εισαγωγή στους αλγορίθμους" των Cormen, Rivest, κλπ.
Δεν έχω κάνει σε καμμία περίπτωση αντιγράψει κώδικα, καθώς αναγκάστηκα να καταλάβω ενδελεχώς πως λειτουργούν οι αλγόριθμοι που παρουσιάζει
το βιβλίο για να μπορέσω να τους υλοποιήσω. Έχω κάνει αρκετές τροποποιήσεις και μια σημαντική πατέντα στη διαγραφή κόμβου από το δένδρο
για να δουλέψει ορθά η δομή. Επομένως, παραθέτω τις ιδιότητες ενός μελανέρυθρου δένδρου σύμφωνα με το βιβλίο:

1) Κάθε κόμβος είναι είτε ερυθρός είτε μελανός.
2) Ο ριζικός κόμβος είναι μελανός.
3) Κάθε φύλλο (ΚΕΝΟ) είναι μελανό.
4) Εάν ένας κόμβος είναι ερυθρός, τότε και οι δύο θυγατρικοί του κόμβοι είναι μελανοί.
5) Για κάθε κόμβο, όλες οι απλές διαδρομές από αυτόν προς φύλλα-απογόνους του περιέχουν ισάριθμους μελανούς κόμβους.

Η συνάρτηση αναζήτησης search() είναι η κλασσική αναζήτηση που θα κάναμε σε ένα οποιοδήποτε δυαδικό δένδρο, οπότε πιστεύω δεν υπάρχει κάτι να εξηγήσω.

Κατά τη διάρκεια μιας εισαγωγής ή διαγραφής ενός στοιχείου σε ένα μελανέρυθρο δένδρο, ενδέχεται μια ή και παραπάνω ιδιότητες να παραβιαστούν.

Γι' αυτό το λόγο μετά την εισαγωγή insert(), καλούμε τη συνάρτηση fix_insertion() η οποία αποκαθιστά τις ιδιότητες που μπορεί να έχουν παραβιαστεί.

Ίδια λογική ακολουθούμε και με τη διαγραφή remove(), κάνοντας χρήση της fix_deletion(). Ουσιαστικά αυτές οι δύο συναρτήσεις δουλεύουν με έναν

bottom-up τρόπο, καθώς παίρνουν ως όρισμα τον πρώτο κόμβο που μπορεί να έχει παραβιάσει κάποια από τις ιδιότητες του μελανέρυθρου δένδρου, αποκαθιστούν

την ιδιότητα που έχει παραβιαστεί, και προχωράνε προς το πάνω στο δένδρο για να εντοπίσουν κι άλλες παραβιάσεις (αν αυτές υπάρχουν) μέχρι να φτάσουν

στη ρίζα. Πρακτικά η επεξήγηση των λειτουργιών αυτών στο README χωρίς να έχω κάποιο σχήμα είναι κάτι αδύνατο να γίνει. Έχω βάλει πολλά σχόλια δίπλα στο

κώδικα σε κάθε σημείο που δεν είναι προφανές.

Το μόνο αξιοσημείωτο κομμάτι είναι μια πατέντα που έχω κάνει στη συνάρτηση fix_deletion(). Επειδή στο βιβλίο δεν υπάρχουν NULL δείκτες, θεωρείται πως

τα κένα φύλλα δείχνουν σε έναν κόμβο που ονομάζεται T.κενό, και λειτουργεί ως φρουρός. Κατά τη κλήση της fix_deletion(), υπάρχει περίπτωση ο κόμβος x

που δίνεται ως όρισμα, να είναι ο T.κενό. Συνεπώς, αν δεν γινόταν κάποια τροποποίηση στο πρόγραμμα, ο x θα είχε τιμή NULL. Σε αυτή τη περίπτωση δεν θα

ήταν δυνατόν να αποθηκευτούν πληροφορίες για αυτόν, όπως ο προκάτοχος. Για να αντιμετωπίσουμε αυτό το πρόβλημα, δεσμεύουμε χώρο και δημιουργούμε έναν

προσωρινό κόμβο τον temp_x, ο οποίος λειτουργεί σαν κενός κόμβος που αποθηκεύει τον προκάτοχό του. Αν καλέσουμε την fix_deletion() με όρισμα τον temp_x

και έπειτα τον αποδεσμεύσουμε, το πρόβλημα αυτό αντιμετωπίζεται.

4.3 Catalogue

Η δομή Catalogue είναι ουσιαστικά μια λίστα από λίστες. Κάθε κόμβος της Catalogue περιέχει μια λίστα. Κάθε λίστα έχει ως κλειδί έναν ταχυδρομικό κώδικα.

Μια λίστα περιέχει τους αριθμούς αστυνομικής ταυτότητας των ψηφοφόρων που έχουν ψηφίσει και που ο ταχυδρομικός τους κώδικας είναι ίδιος με της λίστας.

Μια λίστα περιέχει επίσης των αριθμό των ψηφοφόρων που έχουν τον ταχυδρομικό κώδικα της λίστας ανεξαρτήτως από το αν έχουν ψηφίσει ή όχι. Δηλαδή, κάθε

φορά που προσθέτουμε/διαγράφουμε ένα record στο Red Black Tree, αυξάνουμε/μειώνουμε τον μετρητή unsigned int total_people_in_postal_code. Αυτή η πληροφορία χρησιμεύει για το γρήγορο υπολογισμό του ποσοστού ψηφοφόρων σε κάθε ταχυδρομικό κώδικα (ερώτημα 10). Τόσο η λίστα Catalogue όσο και η List είναι διπλά συνδεδεμένες λίστες με δείκτες αρχή και τέλους. Πιστεύω δεν υπάρχει ανάγκη για περαιτέρω επεξήγηση.

5. Επεξήγηση αρχείων main_funcs.cpp και string_funcs.cpp

Τα αρχεία αυτά περιέχουν τα πηγαία αρχεία για συναρτήσεις διαχείρισης string (strlen(), strcpy(), strcmp()), συναρτήσεις ανάγνωσης εισόδου (get_option()) και αποτίμησης των παραμέτρων γραμμής εντολής (get_parameters()), συνάρτηση εύρεσης κατάλληλου μεγέθους για το Bloom Filter, και συνάρτηση για διαχείριση του inputfile και αρχικοποίηση της σύνθεσης δομών (get_voters()). Έχω βάλει σχόλια σε κάθε ορισμό συνάρτησης για να φανεί ο σκοπός της. Οι υλοποιήσεις είναι σχετικά απλές, οπότε πιστεύω δεν υπάρχει τίποτα άλλο σημαντικό να αναφέρω.

6. Επεξήγηση αρχείου voter.cpp

Στο αρχείο αυτό υπάρχει ο κώδικας της δομής voter και των σχετικών συναρτήσεων. Η δομή έχει σχεδιαστεί ακριβώς όπως ορίζεται στην εκφώνηση.

7. Σχεδιαστικές Επιλογές

Ο λόγος που χρησιμοποίησα C++ αντί για C, είναι κυρίως επειδή στη C++ υπάρχει έτοιμος τύπος δεδομένων bool, κάτι το οποίο είναι πολύ χρήσιμο κατά τη κατασκευή του Bloom Filter, αλλά και σε άλλες περιπτώσεις όπως στο μελανέρυθρο δένδρο (χρησιμοποιείται false για μαύρο χρώμα και true για κόκκινο στη μεταβλητή bool colour), στο πεδίο hasvoted, πεδίο στο φύλο ενός ψηφοφόρου, στο αν μια συνάρτηση λειτούργησε σωστά, κλπ.
Επίσης στη C++ υπάρχουν constructors και destructors, κατι που βολεύει πολύ στην αρχικοποίηση πεδίων στις δομές, και στη σωστή διαγραφή (με αποδέσμευση μνήμης) αντίστοιχα.

UPDATE: Τελικά η μεταβλητή bool χρησιμοποιεί ένα byte και όχι ένα bit όπως πίστευα αρχικά. Οπότε τελικά θα φτιάξω το bit array με ένα πίνακα από int όπως ανέφερα στο 4.1.

8. Χρήσιμες πληροφορίες

Για τη μεταγλώττιση του προγράμματος, απλά δώστε την εντολή make στο terminal.
Για την εκτέλεση δώστε την εντολή: ./runelection -i inputfile -outfile -n numofupdates,
όπου το inputfile είναι το αρχείο registry με τα records, το outfile είναι το αρχείο όπου θα καταγράφεται η έξοδος του προγράμματος και το numofupdates είναι ο αριθμός
των τροποποιήσεων που απαιτούνται για να γίνει αναδιοργάνωση του Bloom Filter. Η σειρά των παραμέτρων μπορεί να είναι οποιαδήποτε.
Για να τσεκάρετε για memory leaks (που προφανώς δεν υπάρχουν) μπορείται να τρέξετε: valgrind ./runelection -i inputfile -outfile -n numofupdates
Για την εκκαθάριση του εκτελέσιμου και των αντικείμενων αρχείων, αρκεί να δώσετε την εντολή make clean στο terminal.
Η είσοδος και η έξοδος ακολουθούν το πρότυπο που δίνεται στη σελίδα του μαθήματος στις οδηγίες μορφοποίησης.
Σε κάποια σημεία του κώδικα, ίσως έχω διάφορα cout που εκτυπώνουν μηνύματα λάθους. Τα χρησιμοποιούσα στο debugging. Θα σβήσω τα περισσότερα. Ωστόσο ίσως αφήσω κάποια
για μελοντική χρήση.
Όταν εισάγετε μια εντολή που δεν αντιστοιχεί σε μία από τις 11 εντολές της εκφώνησης, τότε απλά προχωράμε στο while() loop και ζητάμε νέα εντολή από το χρήστη.