Τσουράκης Ορφέας Α.Μ.: 1115201700175

Ταράτσας Ιωάννης Α.Μ.: 1115201700160

Project K23α-2ο Μέρος Εργασίας

Ενδεικτικές Εκτέλέσεις

Βασική εκτέλεση:

./project -x 2013_camera_specs -w sigmod_large_labelled_dataset.csv –s ./include/stopwords.txt

Unit Testing:

./all_tests

Το παραδοτέο στο δεύτερο μέρος της εργασίας συμπεριλαμβάνει:

- Στον βασικό φάκελο, τα αρχεία main.c, structs.c, text_support.c, logistic_regression.c και το σχετικό makefile που παράγει τα παραπάνω εκτελέσιμα.
- Στον φάκελο tests, τα αρχεία stack_test.c, queue_test.c και rbt_test.c, heap_test.c, ht_test.c, text_test.c, all_tests.c τα οποία χρησιμοποιούν την acutest.h, για το unit testing.
- Στον φάκελο include, έχουμε τα αρχεία structs.h, text_support.h, all_tests.h, logistic_regression.h, acutest.h, αλλά και ένα txt το οποίο συμπεριλαμβάνει τα stopwords τα οποία τα χρησιμοποιούμε για το καθάρισμα του κειμένου.

Αλλαγές από το πρώτο παραδοτέο:

- Στο makefile, σε όλες τις μεταγλωττίσεις έχει προστεθεί το –Wall και έχουν διορθωθεί τα warnings.
- Το all_test.c πλέον τρέχει όλα τα test από τα διαφορετικά αρχεία σε ένα εκτελέσιμο.
- Πραγματοποίηση στην text_test.c του unit testing της parse του json.
- Δημιουργία δομής Hash Table και ενσωμάτωση των Red-Black Trees ως δομή αποθήκευσης και εύκολης αναζήτησης στα buckets του.

Λεπτομέρειες υλοποίησης:

main.c

Στην main.c μετά την ολοκλήρωση του parsing ενός αρχείου json δημιουργούνται επιπλέον οι κλίκες, τα FilesStats και το λεξιλόγιο. Μετά το διάβασμα του csv δημιουργείται το Hash Table των κλικών από τα pairs, γίνεται η αλλαγή των μη συσχετιζόμενων και τυπώνονται τα αποτελέσματα σε δύο αρχεία, related.csv και unrelated.csv αντίστοιχα για κάθε περίπτωση. Τέλος, δημιουργούμε το TF-IDF, γίνεται αποκοπή του λεξιλογίου και εκτελούνται οι λειτουργίες του μοντέλου τυπώνοντας το accuracy που προκύπτει στο stdout.

structs.h

Έχουμε κάνει define τα buckets των Hash Tables.
Έχουν προστεθεί οι αντίστοιχες δομές για το heap και για το Hash Table.
Ακόμα υπάρχουν και άλλα structs τα οποία μας βοήθησαν στην οργάνωση των δεδομένων:
Clique: Περιλαμβάνει ένα id, έναν δείκτη σε ουρά για τα συσχετιζόμενα και ένα Hash Table για τα μη συσχετιζόμενα.
FileStats: Περιλαμβάνει ένα δείκτη σε αντικείμενο(item), το index στον sparse πίνακα, τις μοναδικές λέξεις και τον αριθμό όλων των λέξεων που περιλαμβάνει.
WordStats: Περιλαμβάνει μια λέξη, το index στον sparse πίνακα, και τα αρχεία στα οποία συναντάται η λέξη αυτή.
ModelStats: Περιλαμβάνει το WordStats μιας λέξης μαζί με τις τιμές BOW και TF-IDF.
Record: Περιλαμβάνει δύο δείκτες σε FileStats αντικειμένων και την τιμή value, για το αν ταιριάζουν ή όχι.

structs.c(Νέες συναρτήσεις και αλλαγές)

Heap(σωρός)

Έχει την ιδέα της heapsort αλλά έχει γίνει υλοποίηση με την μορφή δέντρου. Η δομή του σωρού αποθηκεύει τον αριθμό των κόμβων του δέντρου και το ύψος όπου αυξομειώνεται με βάση αυτόν τον αριθμό. Όλες σχεδόν οι συναρτήσεις κάνουν αναζήτηση κατά επίπεδο στο δέντρο που δημιουργείται. Κατά την εισαγωγή με αναδρομή ο νέος κόμβος τοποθετείται στο σωστό σημείο του τελευταίου επιπέδου ώστε να διατηρηθεί η ιδιότητα του πλήρους δέντρου και ανεβαίνοντας γίνονται οι "ανταλλαγές" για δημιουργηθεί ένας σωρός μεγίστου. Όταν δεν χρειάζεται η ρίζα η συνάρτηση findLastAndReplace βρίσκει τον τελευταίο κόμβο(από αριστερά προς τα δεξιά) του τελευταίου επιπέδου τον διαγράφει και βάζει το στοιχείο του στον κόμβο της ρίζας. Για να διατηρηθεί η ιδιότητα του σωρού μεγίστου υπάρχει η συνάρτηση Heapify όπου κατεβαίνοντας από την ρίζα κάνει τις σωστές "ανταλλαγές". Για την εξαγωγή της ρίζας HeapRevoveFirst συνδυάζονται οι δύο τελευταίες συναρτήσεις. Τέλος, η HeapifyWords χρησιμοποιεί το heap για να κάνει εισαγωγή της λέξης με βάσει το μέσο TFIDF τους .

*Ο κώδικας του σωρού έχει υλοποιηθεί από τον Ορφέα Τσουράκη στο μάθημα Προγραμματισμός Συστήματος.

Red-Black δέντρο

Τα Red-Black δέντρα τροποποιήθηκαν σε generic, κάνοντας και τις απαραίτητες αλλαγές στον κώδικα και έτσι ώστε να συμβαδίζουν με τα Hash Tables.

Hash Tables

Για την εισαγωγή(HTinsert) και την αναζήτηση(HTfind) στο Hash Table χρησιμοποιούμε μια συνάρτηση κατακερματισμού για συμβολοσειρές από την βιβλιογραφία, ώστε να καθορίσουμε σε ποίο bucket(δέντρο) θα γίνει ο έλεγχος. Υπάρχει συνάρτηση αρχικοποίησης(HTinit), που αρχικοποιεί τα buckets (τους δείκτες των δέντρων) και συνάρτηση καταστροφής(HTdestr) με βάση τις συναρτήσεις καταστροφής που έχουν οριστεί παραπάνω. Τέλος, έχει υλοποιηθεί μια συνάρτηση για την ένωση δύο hash tables σε ένα(HTmerge).

Επιπλέον συναρτήσεις

PairDestroy(): Καταστροφή Ζευγαριών

CliqueDestroy(): Καταστροφή Κλικών

WordsDestroy(): Καταστροφή των WordStats

FilesDestroy(): Καταστροφή των FileStats

CliqueConcat(): Η συνάρτηση αυτή δέχεται δύο ζευγάρια(pair1, pair2), και μια επιλογή. Αν τα δύο αυτά ζευγάρια ταιριάζουν(δηλαδή η επιλογή είναι 1), τότε ενώνονται τα μη συσχετιζόμενα(αφού δύο όμοια αντικείμενα έχουν τα ίδια μη συσχετιζόμενα), και επιπλέον ενώνονται τα συσχετιζόμενα. Αν όμως αυτά τα δύο ζευγάρια δεν ταιριάζουν τότε απλά το κάθε pair προστίθεται στα μη συσχετιζόμενα του άλλου.

printUnrelated(): In order διάσχιση του δέντρου και χρήση της συνάρτησης VisitUnrelated με την οποία γίνεται η διάσχιση του Hash Table και η εκτύπωση των μοναδικών ζευγαριών (χωρίς διπλότυπα) που δεν ταιριάζουν. Παράλληλα στην VisitUnrelated, γίνεται και η τυχαία εισαγωγή των εγγραφών σε Train, Test και Valid, όπως παρόμοια προστέθηκε και στην printRelated(πρώτο παραδοτέο).

MakeCliqueHT(): Δημιουργεί ένα Hash Table με κλίκες με βάσει τα pairs τα οποία έχουν διαμορφωθεί μετά το διάβασμα των αρχείων.

ChangeUnrelated(): Αποτυπώνει τα μη συσχετιζόμενα pairs των κλικών με τις κλίκες που τελικά ανήκουν, με την βοήθεια της MakeCliqueUnrelated, δημιουργώντας έτσι για κάθε κλίκα όλες τις αρνητικές συσχετίσεις.

Η αναπαράσταση των διανυσμάτων των λέξεων για όλα τα αρχεία έχει υλοποιηθεί, έτσι ώστε να γνωρίζουμε μόνο τις θετικές τιμές με την χρήση Hash Tables **(sparse υλοποίηση).** Για αυτό υπάρχουν βοηθητικές συναρτήσεις για την προσπέλαση αυτών των Hash Table εκτελώντας μια συγκεκριμένη λειτουργία.

CreateTFIDF(): Χρησιμοποιώντας την UpdateTFIDF δημιουργεί τις TF-IDF τιμές.

SumTFIDF(): Αθροίζει το TF-IDF.

AdjustMStats(): Χρησιμοποιεί την InsertMStats για να κρατήσει σε κάθε αρχείο τις λέξεις που δεν έχουν αποκοπεί.

text_support.h

Στο text_support.h, έχει γίνει define το limit των λέξεων που θα γίνει το CutOffDictionary, και το maxSplit.

text_support.c

Στο text_support.c έχουν υλοποιηθεί συναρτήσεις οι οποίες αφορούν την επεξεργασία των κειμένων.

CutOffDictionary(): Η CutOffDictionary, αφού τοποθετήσει τις λέξεις που υπάρχουν στα κείμενα σε ένα on-the-fly heap, αρχικοποιεί ένα καινούριο hash table με λέξεις στο οποίο τοποθετεί τις limit πιο χρήσιμες λέξεις, βάσει του μέσου όρου του TF-IDF κάθε λέξης. Το όριο των limit λέξεων το δίνουμε εμείς ως όρισμα. Στο τέλος, πριν την διαγραφή χρησιμοποιείται και η AdjustMStats.

CreateDictionary(): Η CreateDictionary, ανοίγει ένα FileStats(αρχείο), και για κάθε κείμενο που έχει μέσα, το καθαρίζει και το χωρίζει σε λέξεις με την χρήση των συναρτήσεων textCleaning και tokenize που θα εξηγήσουμε παρακάτω.

InsertWord (): Η InsertWord, πρώτα ελέγχει αν η λέξη η οποία πάει να εισαχθεί, υπάρχει στα stopwords. Αν υπάρχει, τότε την απορρίπτει. Διαφορετικά, στο επόμενο στάδιο γίνεται ο έλεγχος αν η λέξη αυτή υπάρχει ήδη στο HashTable όλων των λέξεων. Αν υπάρχει, ελέγχουμε αν υπάρχει και στις λέξεις του αρχείου για να αυξήσουμε τις ανάλογες μετρικές, είτε διαφορετικά να τις αρχικοποιήσουμε. Αν δεν υπάρχει εισάγει αυτήν την λέξη στα HashTables όλων των λέξεων και των λέξεων του κάθε αρχείου.

textCleaning(): Η textCleaning, δέχεται μια συμβολοσειρά και χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης ctype.h καθαρίζει το κείμενο από τα σημεία στίξης, κάνει τα κεφαλαία γράμματα μικρά, εξαφανίζει τους χαρακτήρες που δεν είναι εκτυπώσιμοι, και διαγράφει τις λέξεις που ξεκινάνε με \.

tokenize(): Η tokenize, χωρίζει μια συμβολοσειρά σε λέξεις(tokens) και τις τοποθετεί με την βοήθεια της InsertWord στα ανάλογα HashTables.

read_stopwords(): Η read_stopwords, ανοίγει το αρχείο των stopwords.txt, το οποίο το έχουμε συμπεριλάβει στον φάκελο include, και τοποθετεί τις λέξεις σε ένα Hash Table, για γρήγορη αναζήτηση τους.

DatasetSplit(): Η DatasetSplit χωρίζει το dataset τυχαία σε 60% train set, 20% test set και 20% validation set. Για να το καταφέρουμε αυτό κάνουμε χρήση της συνάρτησης rand(). Συγκεκριμένα, ανά είκοσι εγγραφές, τις χωρίζουμε τυχαία στα αντίστοιχα set, δηλαδή 12 στο train set, 4 στο test set, 4 στο validation set για να διατηρήσουμε την αναλογία του 60-20-20 που αναφέραμε παραπάνω. Έτσι, ανά είκοσι εγγραφές, εισάγουμε τυχαία σε μία από τις 3 ουρές την πρώτη εγγραφή. Στην συνέχεια, για τις υπόλοιπες 19 εγγραφές, εισάγουμε πάλι τυχαία τις εγγραφές, αλλά ελέγχουμε και αν έχει συμπληρωθεί το ποσοστό το οποίο αντιστοιχεί για κάθε ουρά στις 20 εγγραφές(12 train - 4 test - 4 validation). Αν δεν έχει συμπληρωθεί, τότε εισάγουμε κανονικά την εγγραφή αυτή στην συγκεκριμένη ουρά. Διαφορετικά ξαναχρησιμοποιούμε την rand() για να επιλέξει πάλι τυχαία μία από τις εναπομείναντες ουρά. Αφού, ακολουθεί παρόμοιος έλεγχος για το αν η ουρά η οποία επιλέχθηκε χωράει επιπλέον στοιχείο, τότε η εγγραφή εισάγεται σε μια από τις δύο, ανάλογα το αποτέλεσμα που λαμβάνουμε. Η παραπάνω διαδικασία πραγματοποιείται επαναληπτικά, για όλες τις εγγραφές(records).

logistic_regression.h

Περιλαμβάνει την δομή του μοντέλου και τα αντίστοιχα defines, δηλαδή τα maxIters που είναι ο αριθμός των επαναλήψεων, lrate που είναι το learning rate, epsilon που είναι η ανεκτικότητα της σύγκλισης και το dBoundary που είναι το σύνορο απόφασης.

logistic_regression.c

Έχουν υλοποιηθεί συναρτήσεις οι οποίες κάνουν το train, το test και το predict του μοντέλου με βάσει τους μαθηματικούς τύπους της εκφώνησης. Πιο συγκεκριμένα, η είσοδος των συναρτήσεων train και test είναι μια ουρά με τις αντίστοιχες εγγραφές records. Χρησιμοποιούν την υλοποίηση των sparse array, για τον υπολογισμό των τιμών.

Φάκελος Tests

Για το test των συναρτήσεων έχουμε χρησιμοποιήσει την συνάρτηση TEST_ASSERT της acutest.h.

heap_test.c

test_heapnodecreate: Στην test_heapnodecreate αρχικοποιούμε έναν node του Heap και ελέγχουμε αν οι τιμές έχουν αρχικοποιηθεί σωστά.

test_swap: Στην test_swap αρχικοποιούμε δύο nodes, και εκτελούμε την συνάρτηση της swap, και μετά την εκτέλεση ελέγχεται ότι έχει γίνει η αλλαγή των δύο κόμβων.

test_heapcreate: Στην test_heapcreate αρχικοποιούμε έναν σωρό και ελέγχουμε ότι όλες οι τιμές έχουν αρχικοποιηθεί σωστά.

test_heapinsert: Στην test_heapinsert, δημιουργούμε δύο αντικείμενα τύπου δείκτη σε Details και ελέγχουμε κάθε φορά αν το μέγεθος του σωρού αυξάνεται σωστά, αλλά και αν οι τιμές καταχωρούνται και αποθηκεύονται σωστά.

test_heapremovefirst: Στην test_heapremovefirst, αρχικοποιούμε πάλι δύο αντικείμενα τύπου δείκτη σε Details, τα τοποθετούμε στον σωρό και στην συνέχεια με την χρήση της συνάρτησης HeapRemoveFirst(), τα βγάζουμε από την κορυφή και ελέγχουμε αν το στοιχείο που απομακρύνθηκε είναι το σωστό και αν οι τιμές του σωρού μεταβάλλονται σωστά.

test_heapifywords: Στην test_heapifywords, τοποθετούμε σε ένα HashTable λέξεις και χρησιμοποιώντας την συνάρτηση HeapifyWords(), τοποθετούμε όλες τις λέξεις σε έναν σωρό. Ελέγχουμε αν ο σωρός αυτός έχει δημιουργηθεί σωστά και περιέχει όλες τις λέξεις

text_test.c

check_parse: Στην check_parse ανοίγουμε ένα αρχείο μορφής .json, το οποίο είναι από το dataset τα οποία έχουν δοθεί, και ελέγχουμε αν οι τιμές έχουν αρχικοποιηθεί σωστά βάσει των προδιαγραφών που έχουν δοθεί.

check_CutOffDictionary: Στην check_CutOffDictionary αρχικοποιούμε ένα λεξικό με τέσσερις λέξεις, και εκτελούμε την συνάρτηση της CutOffDictionary με όριο δύο λέξεων. Στην συνέχεια ελέγχουμε ότι όντως ο αριθμός των λέξεων που έχουν μείνει στο λεξικό είναι δύο.

check_textcleaning: Στην check_textcleaning ελέγχουμε με την χρήση διάφορων συμβολοσειρών ότι ο καθαρισμός των δεδομένων έχει πραγματοποιηθεί όπως επιθυμούμε.

check_InsertWord: Στην check_InsertWord αφού αρχικοποιήσουμε κατάλληλα τα απαραίτητα δεδομένα, εισάγουμε λέξεις με την χρήση της συνάρτησης InsertWord, στο HashTable, των words και στην συνέχεια ελέγχουμε ότι έχουν όντως οι λέξεις αυτές έχουν εισαχθεί.

check_read_stopwords: Στην check_read_stopwords, ανοίγουμε το αρχείο των stopwords, το οποίο βρίσκεται στον φάκελο include, και έχοντας πάρει από το αρχείο αυτό κάποιες λέξεις, ελέγχουμε ότι οι λέξεις αυτές έχουν αποθηκευτεί στο αντίστοιχο HashTable.

ht_test.c

test_htcreate: Στην test_htcreate, αρχικοποιούμε ένα Hash Table και ελέγχουμε αν έχουν αρχικοποιηθεί σωστά όλες οι τιμές, δηλαδή αν οι κουβάδες έχουν αρχικοποιηθεί σωστά και αν δείχνουν όλοι οι δείκτες στον dummy κόμβο.

test_htinsert: Στην test_htinsert, δημιουργούμε 100 τιμές τύπου συμβολοσειράς, και προσθέτουμε στο Hash Table κάθε φορά την τιμή που δημιουργείτε. Εκεί ελέγχουμε, κάθε φορά, αν υπάρχει το κλειδί στο Hash Table, αν μπήκε το ίδιο κλειδί, αν η τιμή αυτή μπήκε όντως στο Hash Table, και αν βρέθηκε η ίδια τιμή. Στην συνέχεια με την χρήση της HTFind ελέγχουμε αν κάθε τιμή έχει εισαχθεί στο Hash Table.

test_htmerge: Στην test_htmerge, αφού δημιουργούμε 100 τιμές και τις εισάγουμε σε ένα Hash Table και άλλες 100 και τις εισάγουμε σε ένα δεύτερο Hash Table. Στην συνέχεια, αφού χρησιμοποιήσουμε την συνάρτηση HTmerge, ελέγχουμε, αν ο τελικός πίνακας περιλαμβάνει όλα τα pairs τα οποία πριν είχαμε σε δύο Hash Tables

all_tests.c

Η all_tests.c περιέχει μια λίστα από όλα τα test, έτσι ώστε να μπορούν να εκτελεστούν όλα μαζί

Εξαιτίας της αλλαγής της υλοποίησης από σκέτο Red Black Tree σε Hash Table με Red Black Trees, αλλά και την προσθήκη στοιχείων όπως οι κλίκες, τα προηγούμενα test για τις δομές, έχουν προσαρμοστεί στα καινούρια δεδομένα.