Πρώτο μέρος για το μάθημα Ανάπτυξη λογισμικού για πληροφοριακά Συστήματα.

Ονόματα Ομάδας

Νικολετόπουλος Παναγιώτης 1115201200126

Βιρβίλης Αντώνης 1115201200014

Παραδοτέα:

-**main.c** : Περιέχει τις κλήσεις των συναρτήσεων για να αρχίσει το πρόγραμμα

-**functions.**c: Περιέχει τις συναρτήσεις για διάβασμα από αρχείο και σπάσιμο σε λέξεις, εισαγωγή σε trie, διαγραφή από trie, ψάξιμο κειμένου με βάση το trie και όλες τις συναρτήσεις που χρησιμοποιούν .

-**functions.h**: Ο ορισμός των προτύπων των συναρτήσεων και ο ορισμός των δομών trie_node,paths καθώς και κάποια macros

-**stack.c:** Περιέχει τις συναρτήσεις με τις οποίες βάζω , βγάζω στοιχειά από την δομή stack , την αρχικοποιούν και την διαγράφουν. Πολλές φορές χρησιμοποιώ την δομή stack χωρίς pop και push.

-**stack.h**: Περιέχει τα πρότυπα των συναρτήσεων και τον ορισμό της δομής stack καθώς και μερικα macros

-**test_main**.c: Η main που χρησιμοποιουμε για το για το unit testing

-**test.c** :Οι συναρτήσεις για unit testing

-**test.h:** ο ορισμός των συναρτήσεων για unit testing

-**makefile**: Η εντολές για το compilation του προγράμματος:

make project: γίνεται compile το πρόγραμμα

make test_project: γίνεται compile το πρόγραμμα χρησιμοποιώντας την main για unit test

make run: τρέχει το πρόγραμμα με το μικρό dataset

make pipe: τρέχει το πρόγραμμα με το μικρό dataset και κάνει ανακατεύθυνση την έξοδο στο results.txt ώστε να συγκρίνουμε τα αποτελέσματα μετά

make testrun: τρέχει το πρόγραμμα με valgrind για να βρεθούν τυχόν memory leaks

-**το dataset**: που μας δώθηκε

-**Και μερικές συναρτήσεις** που μας βοήθησαν στο debugging

Αποτελέσματα : Το πρόγραμμα για το μικρό dataset τρέχει σε 1.0925 δευτερόλεπτα και τα αποτελέσματα είναι ιδία με τα αποτελέσματα που μας δώσατε (το τσεκάραμε με ένα δικό μας πρόγραμμα και μετά με online diff checker όπου αλλάξαμε μόνο το format της εξόδου μας για να είναι όμοιο)

Από άποψη μνήμης από το valgrind παίρνουμε: All heap blocks were freed – no leaks are possible

Οι μετρήσεις έγιναν από την αρχή πριν αρχικοποιήσουμε το trie και αφού καταστρέψουμε όλο το trie.

Σχεδιαστικές επιλογές:

Η εισαγωγή σε ngram γίνετε αναδρομικά αλλα εχει υλοποιηθη και σειριακα . Οι χρονοι δεν δειχνουν διαφορες (αισθητες) :

Σειριακα: 1.1335 δευτερολεπτα

Αναδρομικα: 1.1469 δευτερολεπτα

Παρολα αυτα ενω τα αποτελεσματα ειναι τα ιδια εχουμε αφηση την αναδρομικη υλοποιηση να τρεχει γιατι εχει δοκιμαστει πιο πολυ

Η διαγραφή ngram από trie γινετε αναδρομικα . Ο λόγος είναι για ευκολία κώδικα, αποφυγή δεύτερου stack και όπως αναφέρθηκε στο μάθημα το heap από την κλήση των συναρτήσεων δεν μπορεί να είναι πολύ μεγάλο. Παρόλα αυτά θα φτιάξουμε τις αντίστοιχες συναρτήσεις και θα συγκρίνουμε τα αποτελέσματα.

Η αναζήτηση για ngram σε κείμενο γίνεται επαναληπτικά. Ο λόγος είναι ότι εκεί θα γίνονται πολλές επαναλήψεις και πιστεύουμε ότι η συνεχής κλήση των συναρτήσεων αναδρομικά θα επιφέρει καθυστέρηση.

Για την μη επανάληψη ngrams που έχουν ήδη βρεθεί σε ένα κείμενο χρησιμοποιούμε μια δομή paths όπου κρατεί το μονοπάτι που έχει κάθε ngram στο trie. Το paths είναι ένας δισδιάστατος πίνακας. Αν γεμίσει τότε διπλασιάζεται. Όταν βρούμε μια λέξη στο κείμενο, πρώτα τσεκάρουμε αν υπάρχει το αντίστοιχο μονοπάτι στο paths, και μετά το εμφανίζουμε. Η αναζήτηση στο paths είναι σειριακή. Έγινε προσπάθεια να γίνει δυαδική αναζήτηση αλλά θα έπρεπε πάλι να

πληρώσουμε τις μετακινήσεις των paths αφού πρέπει κάθε φορά να είναι sorted. Η χρήση της memove δεν βοήθα σε αυτή την περίπτωση. Προς το παρόν έχουμε μια απλοποιημένη μορφή της paths και εμφανίζουμε τα αποτελέσματα της αναζήτησης όπως τα βρίσκουμε και όχι μόλις φτάνουμε στο F. Έχουμε βρει λύση και στην εμφάνιση ανά ριπές και στην αποτελεσματική αναζήτηση. Η υλοποίηση της θα γίνει στο δεύτερο κομμάτι.

Το μέγεθος μιας λέξης ξεκινάει απο 25 γράμματα. Με λιγη μελετη υπολογισαμε πως δεν θα πετυχουμε ευκολα λεξη μεγαλυτερου μηκους. Παρολα αυτά σε περίπτωση μεγαλύτερης λέξης στο διάβασμα του αρχείου, το μεγεθος επαναπροσδιοριζεται στο μεγεθος τις λέξης που προκειτε να εισάγουμε τον κόμβο. Αυτό γίνεται κάθε φορά που μια λέξη δεν χωράει στον buffer μας.

Μερικές δοκιμές που έγιναν σε αυτό το σημείο:

-Η πρώτη υλοποίηση ήταν με έναν δείκτη ο οποίος γινόταν malloc σε WORD_SIZE*sizeof(char) αν και θεωρητικά μη ασφαλής λύση σε περίπτωση που έρθει μια λέξη πάνω από 25 γράμματα .Στην πράξη δεν ήρθε ποτέ λεξη με πάνω από 25 χαραχτήρες. Ο χρόνος εκτέλεσης ήταν 1.100 δευτερόλεπτα περίπου.

-Στην δεύτερη υλοποίηση αποφασίσαμε να κλείσουμε αυτή την τρυπά και βάλαμε την επιλογή αν είναι πάνω από 25 χαραχτήρες να γίνετε malloc η αν είναι κάτω απο 25 χαραχτήρες να αποθηκεύετε σε στατικό πίνακα . Ο pointer θα δείχνει είτε στον στατικό πίνακα η στο εαυτό του που θα εχει γίνει malloc . Ο χρόνος εκτέλεσης ήταν 1.130 . Αν και δεν έκανε ποτέ malloc έδωσε μια παραπάνω καθυστέρηση.

Οι λέγεις διαβάζονται απο το αρχείο εισόδο γραμμή γραμμή, η οποία χωρίζεται σε λέξεις με την χρήση συναρτήσεων της string.h. Κάθε λέξη που διαβάζεται τοποθετείται σε μια δομή, μέχρι ότου ολοκληρωθεί η γραμμή, οπου και καταχωρείται σαν nrgam. Η συνάρτηση init_input διαβαζει το work.init και αποθηκευει τα ngrams, ενώ η test_input διαβάζει τις εντολές απο το work.test. Η τελευταία, ανεβάζει κάποια flags ανάλογα με το πρώτο γράμμα που θα διαβάσει, ωστε στο τέλος να γνωρίζει αν θα κάνει add, remove ή search.

Η συνάρτηση cleanup καθαριζει τη δομη που χρησιμοποιούμε για την μεταγορα ngrams απο συναρτηση σε συνάρτηση.

Για το κομμάτι του unit testing :

Για το unit test της add και της delete τσεκάρουμε αν μετά την add και την delete του ngram υπάρχει το ngram στο trie. Μετά την add, το ngram περιμένουμε να υπάρχει, ενώ, μετά την delete περιμένουμε να μην υπάρχει, ή να γυρνά ότι δεν μπορεί να την σβήσει, είτε ότι άλλαξε τον τελικό κόμβο σε μη τελικό.

Για την συνάρτηση αναζήτησης ελέγχουμε τις συναρτήσεις τις δυαδικής αναζήτησης με μερικά test cases και αν υπάρχει το μονόπατη στο paths.