Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα 2017-2018
Μέρος 1ο

Αργυρός Ιωάννης - 1115201200009
Γιαννούδης Αστέριος - 1115201200025
Κολυβάς Κωνσταντίνος - 1115201200066

Γλώσσα: C++11

Μεταγλώττιση:

```
$ cd build
$ cmake ..
$ make ngrams
```

Εκτέλεση:

```
$ ./ngrams -i <init_file> -q <query_file>
```

Μεταγλώττιση και εκτέλεση των Unit Tests:

```
$ make basic_tests
$ ./basic_tests
```

Άδειασμα του φακέλου build :

```
$ sh clean.sh
```

1. Δομή του Trie:

Αποτελείται από έναν κόμβο (root). Ο κάθε κόμβος, περιέχει μία λέξη ενός n-gram και ένα αντικείμενο mstd::vector (βλ. 3). Αυτό το vector περιέχει υπο-κόμβους, δηλαδή κάθε λέξη η οποία είναι συνέχεια του n-gram.

Επιλέχθηκε μια δομή σαν το vector ως ο τρόπος αποθήκευσης των παιδιών του κάθε κόμβου διότι:

Α. Ένας πίνακας έχει τα παιδιά του σε συνεχόμενες θέσεις μνήμης, οπότε αυξάνεται η πιθανότητα “επιτυχίας” στην μνήμη cache.

Β. Με την προϋπόθεση πως ο πίνακας παραμένει πάντοτε ταξινομημένος (εγγυημένο από τον τρόπο εισαγωγής νέου κόμβου), η πολυπλοκότητα αναζήτησης σε χρόνο γίνεται O(logn) με την χρήση της binary search.

2. Λειτουργίες του Trie:

2.1. Εισαγωγή:

Κατά την εισαγωγή νέου n-gram στο δένδρο, για κάθε λέξη του ελέγχεται αν ήδη υπάρχει στο δένδρο ή θα πρέπει να εισαχθεί. Σε αυτό τον έλεγχο ύπαρξης, χρησιμοποιείται η μέθοδος binary search στον πίνακα (mstd::vector) των παιδιών του κάθε κόμβου, η οποία είτε

επιστρέφει το στοιχείο του πίνακα το οποίο θα πρέπει να “ακολουθήσουμε”, είτε την θέση στην οποία θα πρέπει να εισαχθεί το νέο στοιχείο ώστε ο πίνακας να παραμείνει ταξινομημένος

2.2 Αναζήτηση:

Για τη μέθοδο της αναζήτησης στο trie υλοποιήθηκαν οι συναρτήσεις search και _bsearch_children (η οποία καλείται μέσα στην trie_node::get_child(...)). Η search δέχεται όλο το δωθέν κείμενο από το query και ψάχνει μέσα σε αυτό όλα τα φορτωμένα ngrams του trie. Η _bsearch_children εκτελεί δυαδική αναζήτηση μέσα στον ταξινομημένο πίνακα των παιδιών κάθε κόμβου για να βρει το ζητούμενο παιδί κάθε φορά. Στην μέθοδο search χρησιμοποιήσαμε τις δομές queue και hashtable για να πετύχουμε την αποθήκευση όλων των αποτελεσμάτων κάθε ριπής καθώς και να διαχειριστούμε τις διπλές εμφανίσεις των ζητούμενων ngrams. Κάθε φορά που βρίσκεται μία απάντηση, ελέγχεται εάν υπάρχει μέσα στο hashtable και αν υπάρχει σημαίνει πως έχει ξαναβρεθεί. Τέλος, η απάντηση μιας ζητούμενης αναζήτησης μπαίνει σαν κόμβος στην ουρά των απαντήσεων για να εξαχθεί αργότερα κατά την εντολή F στο αρχείο ενεργειών. Αν κανένα από τα αποθηκευμένα ngrams δεν υπάρχει στο κείμενο που δίνεται, τότε δημιουργείται κόμβος στην ουρά με το ειδικό string “$$END$$”. Αντίστοιχα, όταν το trie είναι τελείως άδειο, αποφεύγεται ολόκληρη η αναζήτηση και προστίθεται στην ουρά κόμβος με το ειδικό string “$$EMPTY$$”. Στην ακραία περίπτωση που το κείμενο του query περιέχει τα δύο ειδικά αυτά strings, τότε το αποτέλεσμα της αναζήτησης θα περιέχει λάθη. Κατά σύμβαση, μπορούμε να αλλάξουμε τις ειδικές φράσεις για να γίνουν ακόμα πιο ειδικές και να μην υπάρχουν μέσα στα κείμενα.

2.3 Διαγραφή:

Για τη διαγραφή του ζητούμενου n-gram, έχει υλοποιηθεί μια επαναληπτική μέθοδος. Η μέθοδος αυτή, αρχικά ψάχνει στο trie κάθε λέξη του n-gram, μέχρις ότου βρει την τελευταία. Ταυτόχρονα, χρησιμοποιούνται δύο παράλληλοι πίνακες για την αποθήκευση των δεικτών των κόμβων που έχουμε επισκεφθεί, καθώς και τη θέση στην οποία βρίσκεται το επόμενο κομμάτι του n-gram, δηλαδή το παιδί του κάθε κόμβου. Διατρέχοντας από κάτω προς τα πάνω τώρα το δέντρο, αποφασίζουμε εάν κάποιος κόμβος πρέπει να διαγραφεί και, αν αυτό είναι αναγκαίο, τότε ο πατέρας αυτού του κόμβου είναι αυτός που θα το διαγράψει τελικά.

Η επιλογή για την υλοποίηση της διαγραφής μέσω μίας επαναληπτικής μεθόδου, προέκυψε για την αποφυγή αποθήκευσης κλήσεων συναρτήσεων στην στοίβα χρόνου εκτέλεσης, που θα υπήρχαν με την υλοποίηση μιας αναδρομικής μεθόδου.
Τέλος, η επιλογή δυο πινάκων για την αποθήκευση των κόμβων πατέρων και παιδιών, προέκυψε καθώς η χρήση δεικτών προς κόμβους κατέστη αδύνατη λόγω της μετακίνησης κόμβων μέσα στον πίνακα παιδιών του κάθε κόμβου.

3.. mstd::vector (template):

Παρόμοιο με το std::vector της stl της C++ (με την διαφορά πως το μέγεθός του μπορεί να ελαττωθεί).

3.1.1. Εισαγωγή στο vector: Το νέο στοιχείο εισάγεται στο τέλος του πίνακα. Αν το νέο μέγεθος θα υπερέβαινε την χωρητικότητά του, τότε η χωρητικότητα διπλασιάζεται.

3.1.2. Εισαγωγή σε συγκεκριμένη θέση του vector: Το νέο στοιχείο εισάγεται στην θέση και όλα τα στοιχεία στα δεξιά του νέου αντικειμένου μετακινούνται κατά 1 θέση δεξιά. Αν το νέο μέγεθος θα υπερέβαινε την χωρητικότητα του vector, τότε η χωρητικότητα διπλασιάζεται.

3.1.3. Διαγραφή συγκεκριμένης θέσης του vector: Το στοιχείο στην θέση του vector διαγράφεται (όλα τα στοιχεία στα δεξιά του vector μετακινούνται 1 θέση αριστερά). Αν το νέο μέγεθος του vector είναι το ¼ της χωρητικότητάς του, τότε η χωρητικότητα υπο-διπλασιάζεται. Αυτό γίνεται για να απελευθερώνεται χώρος ο οποίος δεν χρησιμοποιείται, χωρίς όμως να σπαταλάται χρόνος σε αντιγραφές/μετακινήσεις αντικειμένων.

3. Unit Testing:

3.1. Για την υλοποίηση του unit testing χρησιμοποιήθηκε η βιβλιοθήκη Google Tests. Έχει προσαρμοστεί κατάλληλα το αρχείο CMakeLists.txt ώστε να κατεβάζει μέσα στον φάκελο build τα απαραίτητα αρχεία από το repository google/googletest στο github.

Επιλέχθηκε κάποιο έτοιμο framework λόγω του scalability που παρέχει για όταν θα φτάσουμε στις επόμενες φάσεις τις εργασίας. Δεν είχαμε ξαναδουλέψει με το unit testing και ξεκινήσαμε να ασχολούμαστε με αυτό μετά την υλοποίηση της υπόλοιπης εργασίας. Συνεπώς, όλα τα τεστ έγιναν με δεδομένο ότι όλα λειτουργούσαν ήδη σωστά.