Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα
Χειμερινό εξάμηνο 2017-2018

"N-grams detection part 2"

Ημερομηνία Παράδοσης Α' Μέρους: **01 Δεκεμβρίου 2017**

Υπεύθυνος Καθηγητής	**Συνεργάτες μαθήματος**
Ιωαννίδης Ιωάννης	Γαλούνη Κωνσταντίνα
	Θεοδωρακόπουλος Ευθύμιος
	Λιβισιανός Τσαμπίκος
	Πασκαλής Σαράντης

Γενική περιγραφή

Αυτό το επίπεδο της εργασίας αποτελείται από τέσσερα τμήματα. Στο πρώτο τμήμα θα υλοποιήσετε μια ειδική δομή για την αποφυγή διπλότυπων n-grams σε ένα Q (bloom filter). Στο δεύτερο τμήμα θα φτιάξετε έναν πίνακα γραμμικού κατακερματισμού στο root του trie για την αποδοτική εύρεση των πρώτων λέξεων ενός n-gram. Στο τρίτο τμήμα, θα πρέπει να σχεδιάσετε και να θέσετε σε λειτουργία έναν μηχανισμό για να βρίσκει τα top-K n-grams μιας ριπής. Στο τέταρτο τμήμα, θα υλοποιήσετε ένα στατικό trie (χωρίς εισαγωγές και διαγραφές στην ριπή), στο οποίο θα γίνεται συμπίεση κόμβων σύμφωνα με την περιγραφή που θα δοθεί.

Για κάποιες λειτουργίες θα είναι απαραίτητο να φτιάξετε και δικές σας επιπλέον δομές. Όπως έχει ήδη ειπωθεί, αν κατά τη διάρκεια της υλοποίησης αυτού του επιπέδου διαπιστώσετε ανεπάρκειες στον κώδικα που ήδη έχετε παραδώσει, μπορείτε να τον αλλάξετε για να λειτουργήσει σωστά σε αυτό το επίπεδο.

Στατική και Δυναμική δομή

Στο 2ο επίπεδο της εργασίας το πρόγραμμα θα δέχεται ως είσοδο τόσο δυναμικά trie, δηλαδή έχουν προσθαφαιρέσεις n-gram, όσο και στατικά τα οποία δε μεταβάλλονται κατά τη διάρκεια εκτέλεσης των ερωτημάτων. Στην πρώτη γραμμή του init αρχείου θα υπάρχει το αναγνωριστικό DYNAMIC ή STATIC, το οποίο θα δηλώνει αν το trie είναι δυναμικό ή στατικό αντίστοιχα.
Τα τρία πρώτα τμήματα που περιγράφονται παρακάτω εφαρμόζονται και στα δύο είδη trie, ενώ το τελευταίο μόνο στα στατικά.

Β' μέρος της Άσκησης

1. Αποφυγή διπλότυπων n-grams σε ερώτημα Q

Στο πρώτο μέρος της εργασίας, σας είχε ζητηθεί να εκτυπώνετε μόνο την πρώτη εμφάνιση ενός n-gram σε ένα ερώτημα Q. Σε αυτό το μέρος καλείστε να υλοποιήσετε την δομή bloom filter https://en.wikipedia.org/wiki/Bloom_filter για την αποφυγή διπλότυπων.

Bloom filter

Είναι μια πιθανοτική δομή δεδομένων με κύρια έμφαση τη μείωση του χώρου. Έχει σχεδιαστεί για να απαντάει γρήγορα και αποδοτικά όσον αφορά τη μνήμη, εάν ένα στοιχείο εμφανίζεται στο σύνολο. Το τίμημα είναι ότι μπορεί να μας απαντήσει είτε εάν ένα στοιχείο **δεν είναι σίγουρα** στο σύνολο είτε **εάν πιθανότατα είναι (false positive)**.

Για να υλοποιηθεί ένα bloom filter στα πλαίσια της εργασίας χρειάζεται:

- Ένα bit vector μεγέθους ***m***, αρχικοποιημένο στο 0
- Ένα σύνολο ***k*** ανεξάρτητων συναρτήσεων κατακερματισμού (hash functions) h(x). Κάθε μια συνάρτηση θα παίρνει ως είσοδο ένα n-gram και θα επιστρέφει μία τιμή i, όπου i < m και αντιπροσωπεύει μια θέση του bit vector

Εισαγωγή

Ένα νέο n-gram για να εισαχθεί επιτυχώς στο φίλτρο θα πρέπει να εφαρμοστεί σε κάθε μια από τις k συναρτήσεις κατακερματισμού. Θα παραχθούν k τιμές (indices) και για κάθε μια τιμή που υποδεικνύει μια θέση στο bit vector, θα αλλάζει η τιμή της θέσης αυτής σε 1 (`array[i] = 1`).

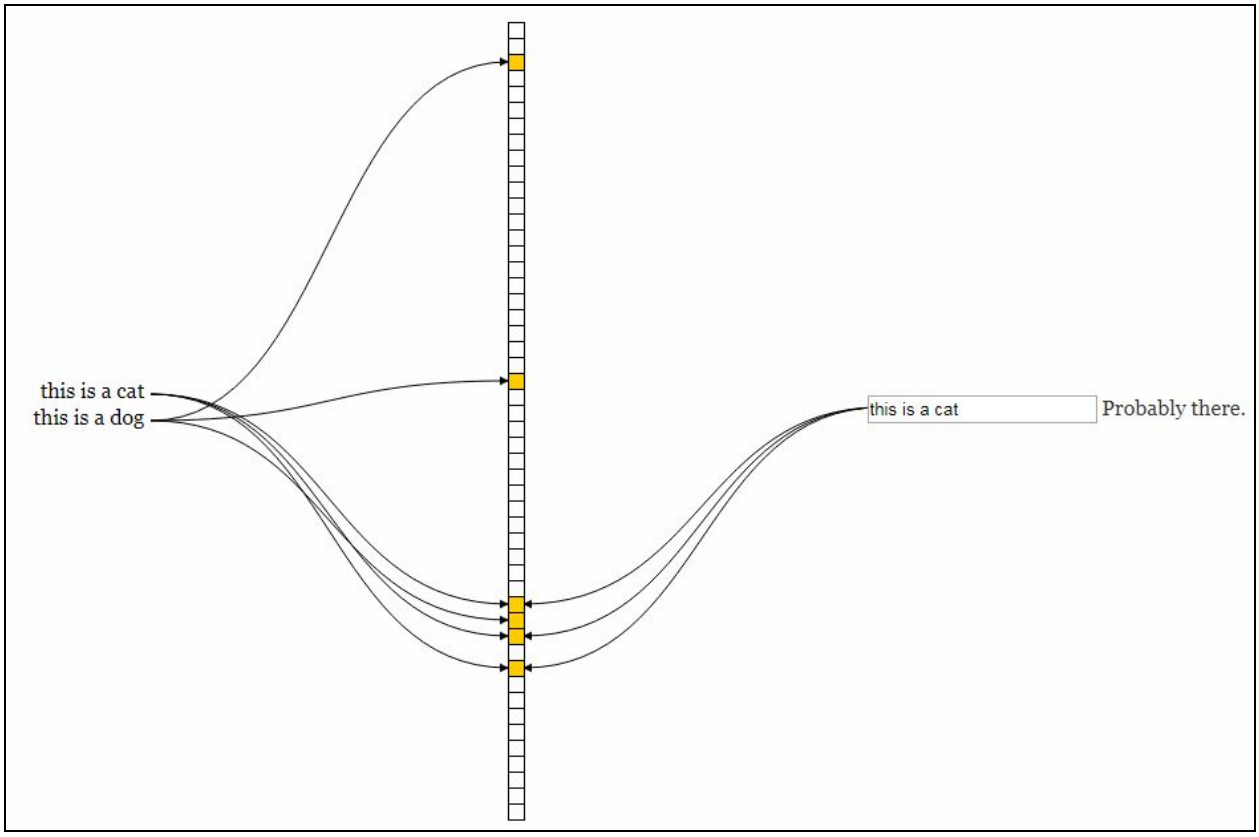
Έλεγχος εμφάνισης

Με παρόμοια διαδικασία, ένα n-gram προς έλεγχο θα πρέπει να εφαρμοστεί στις k συναρτήσεις κατακερματισμού. Εάν έστω και μια θέση του bit vector (που προκύπτουν από τις τιμές εξόδου των συναρτήσεων) είναι 0, τότε το n-gram σίγουρα δεν είναι στο φίλτρο, αλλιώς πιθανότατα είναι (όντως είναι ή αποτελεί false positive).

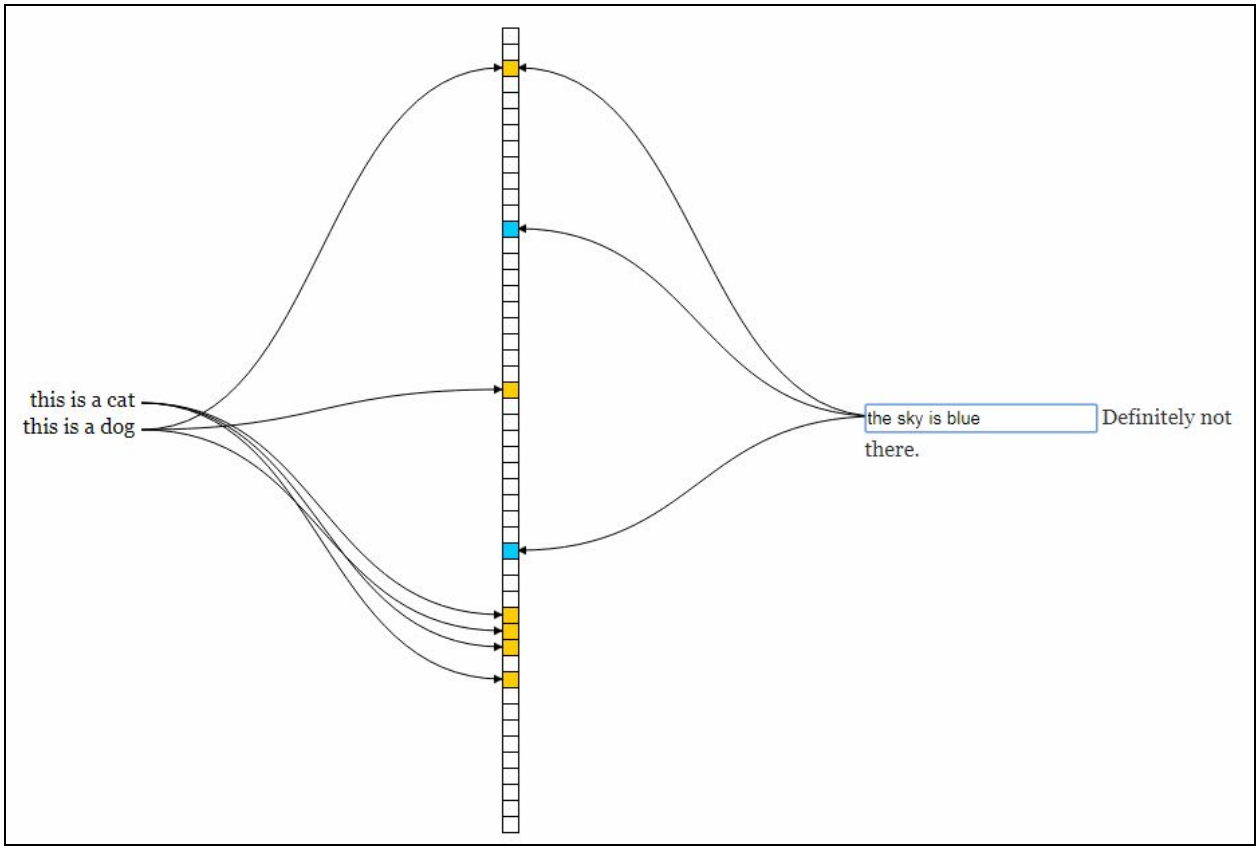
Παράδειγμα

Έστω ότι έχουμε 3 συναρτήσεις κατακερματισμού, ένα bit vector 50 θέσεων και έχουμε εισάγει τα n-grams “this is a cat” και “this is a dog”.

Παρατηρούμε ότι κατά τον έλεγχο του “this is a cat” η απάντηση είναι ότι πιθανότατα υπάρχει.



Ενώ κατά τον έλεγχο του “the sky is blue”, η απάντηση είναι ότι σίγουρα δεν υπάρχει.



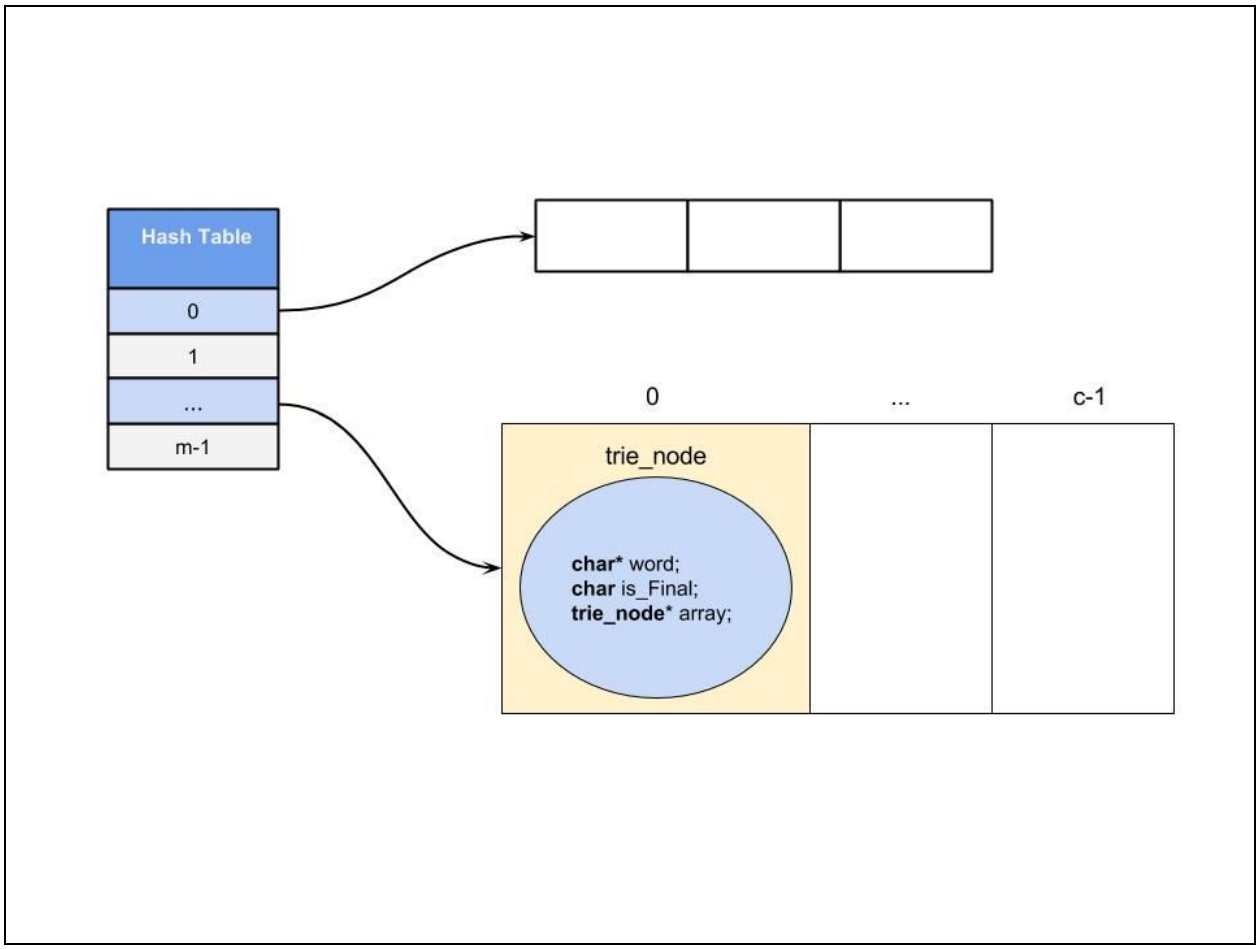
Στόχος

Στόχος σας είναι να μελετήσετε την βιβλιογραφία για τη συγκεκριμένη δομή. Στη συνέχεια σας παρέχεται μεγάλη ελευθερία, για τον τρόπο που θα την υλοποιήσετε, έχοντας λάβει περιπτώσεις και παραδοχές, με τις οποίες θα ελαχιστοποιείτε τα false positives, κρατώντας συγχρόνως την δομή σας αποδοτική. Σε επίπεδο συναρτήσεων κατακερματισμού πρέπει πάλι να μελετήσετε και να βρείτε αυτές που θεωρείτε κατάλληλες για την αντιμετώπιση του προβλήματος.

2. Υλοποίηση πίνακα κατακερματισμού

Στο πρώτο μέρος της εργασίας, σας είχε ζητηθεί να υλοποιήσετε μια δομή trie, όπου σε κάθε κόμβο της αποθηκευόταν ένας πίνακας Ν θέσεων με τα παιδιά του. Σαν επέκταση αυτής της λογικής, θα πρέπει στο root κόμβο του trie να αντικαταστήσετε τον πίνακα με έναν πίνακα γραμμικού κατακερματισμού (linear hashing). Την περιγραφή του αλγορίθμου μπορείτε να την βρείτε στο σύνδεσμο: http://cgi.di.uoa.gr/~ad/MDE515/e_ds_linearhashing.pdf. Σύμφωνα με τον παραπάνω αλγόριθμο, αν θεωρήσουμε ότι είμαστε στο i-οστό επίπεδο δημιουργίας split, έχουμε τα παρακάτω δεδομένα και τη γενικότερη σχηματική αναπαράστασή τους.

- hashtable αρχικού μεγέθους m
- c πλήθος κελιών σε κάθε κουβά
- hash function στο i-οστό επίπεδο δημιουργίας split $h_i(k)$



Στα πλαίσια αυτής της περιγραφής θα υλοποίησετε **ενδεικτικά** τις μεθόδους:

- ★ Δημιουργία πίνακα κατακερματισμού (**createLinearHash**).
  Η συνάρτηση αυτή δημιουργεί ένα άδειο πίνακα κατακερματισμού..
  Input: ένας ακέραιος m που είναι το αρχικό μέγεθος του hashtable, ένας ακέραιος c που είναι το πλήθος των κελιών κάθε κουβά.
  Output: δείκτης στον άδειο γράφο.
- ★ Καταστροφή του πίνακα κατακερματισμού (**destroyLinearHash**).
  Η συνάρτηση αυτή καταστρέφει τον πίνακα κατακερματισμού..
  Input: δείκτης στον πίνακα.
  Output: success/fail
- ★ Εισαγωγή trie_node στον πίνακα (**insertTrieNode**)** 1
  Η συνάρτηση αυτή εισάγει ένα trie_node με τα χαρακτηριστικά του στον πίνακα κατακερματισμού.
  Input: δείκτης σε δομή με τον κόμβο και τις ιδιότητες του, δείκτη στον γράφο.
  Output: success/fail.
- ★ Αναζήτηση ενός κόμβου (**lookupTrieNode**)
  Η συνάρτηση αυτή θα επιστρέφει ένα δείκτη στο ζητούμενο κόμβο, δίνοντας πρόσβαση σε ό,τι τον αφορά (ιδιότητες κόμβου).
  Input: το name του ζητούμενου κόμβου, δείκτης στον πίνακα κατακερματισμού.
  Output: δείκτης στον κόμβο.

Σημειώσεις:

(** 1) Οι κόμβοι μέσα σε κάθε κουβά πρέπει να είναι ταξινομημένοι για να μπορέσετε να χρησιμοποιήσετε αλγόριθμο αναζήτησης μικρότερης πολυπλοκότητας από τη σειριακή.

3. Εύρεση top-k n-grams σε μία ριπή

Σε αυτό το τμήμα της εργασίας ενδιαφερόμαστε για τη δημοφιλία των n-grams στα Queries μιας ριπής. Με την είσοδο μίας ριπής, δίπλα από το F, το οποίο σηματοδοτεί το τέλος της, κάποιες φορές θα υπάρχει ένας ακέραιος (μη αρνητικός) αριθμός που θα αντιστοιχεί στο k. Συνεπώς, σε κάθε ριπή μπορεί να ζητείται διαφορετικός αριθμός δημοφιλέστερων n-grams εντός της.
Αφού εκτελεστούν τα ερωτήματα και οι προσθαφαιρέσεις μίας ριπής, θα πρέπει να εξεταστούν τα αποτελέσματα των Q και να βρεθούν τα k πιο συχνά εμφανιζόμενα n-grams στα αποτελέσματα αυτά. Τα top-k n-grams θα πρέπει να εμφανίζονται με φθίνουσα σειρά, δηλαδή το πιο δημοφιλές να είναι πρώτο. Στην περίπτωση που δύο ή περισσότερα n-grams εμφανίζονται ίδιο αριθμό φορών στα αποτελέσματα, σειρά προτεραιότητας θα έχει το αλφαριθμητικά μικρότερο.
Όπως και στο πρώτο μέρος της εργασίας, δε λαμβάνονται υπόψη διπλότυπες εμφανίσεις n-grams σε ένα Q. Ο τρόπος με τον οποίο θα εμφανίζονται τα αποτελέσματα των top-k θα είναι όμοιος με τον τρόπο που εμφανίζεται το αποτέλεσμα ενός Q με τη διαφορά πως στην αρχή της απάντησης θα υπάρχει η ετικέτα “Top: ”.

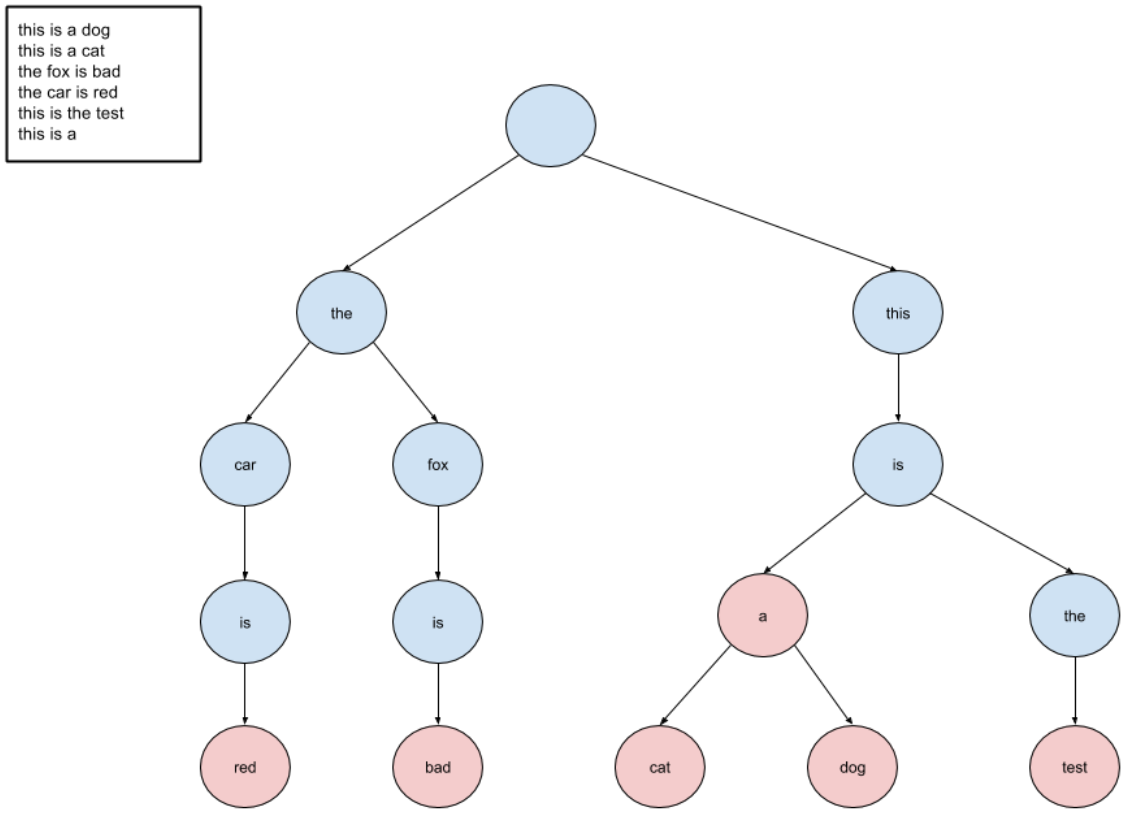
Παρακάτω ακολουθεί παράδειγμα εισόδων και εξόδου για την εύρεση top-k n-grams στις ριπές.

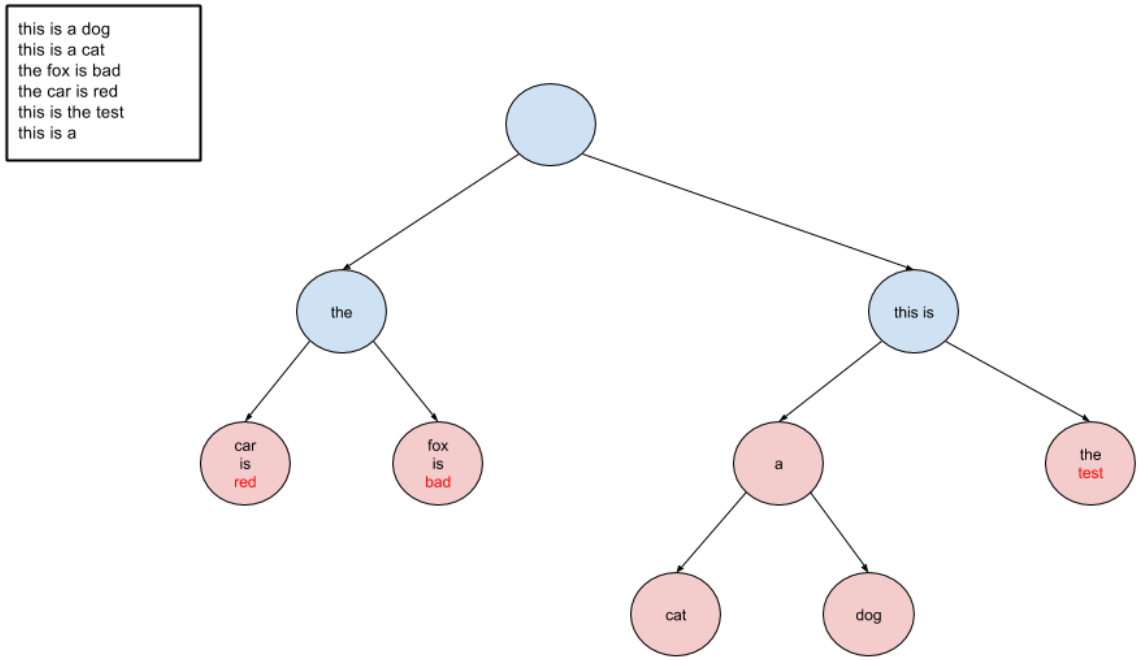
n-grams	Ριπές	Αποτελέσματα
this is a dog this is a cat car the car	A this is a Q this is a cat Q this is a dog Q that is a cat D the car Q car car car car Q this car Q this is a F 3	this is a|this is a cat this is a|this is a dog -1 car car this is a Top: this is a|car|this is a cat fox fox|this is a cat Top: fox
	D this is a A fox Q the fox Q this is a fox and this is a cat F 1	

Top-k

4. Στατικό trie

Για τα στατικά trie θα πρέπει να υλοποιηθεί η συνάρτηση **compress** η οποία θα δέχεται ένα trie και θα συμπιέζει τους κόμβους. Στις παρακάτω 2 εικόνες φαίνεται ποια θα πρέπει να είναι η τελική δομή μετά από την συμπίεση.





Για την αποθήκευση των νέων κόμβων (υπερκόμβων) θα πρέπει να χρησιμοποιηθούν τα εξής:

- Οι λέξεις θα αποθηκεύονται σε ένα συνεχόμενο char*.
- Θα υπάρχει ένας πίνακας από προσημασμένους μικρούς ακεραίους (signed shorts), ο οποίος θα κρατάει το μέγεθος της κάθε λέξης και το αν είναι τελική ή όχι χρησιμοποιώντας το πρόσημο.

Για παράδειγμα, ο κόμβος που περιέχει το “car is red” θα αναπαριστάται από char* words = “carisred” και από έναν πίνακα με 3 shorts τα οποία θα έχουν τις τιμές -3, -2, +3.

Προφανώς, θα πρέπει να υλοποιηθεί και η κατάλληλη search η οποία να λαμβάνει υπόψη της τους υπερκόμβους.

Στο ερώτημα αυτό θα πρέπει να κάνετε μετρήσεις και να συγκρίνετε το συμπιεσμένο trie με το ίδιο trie που δεν έχει συμπιεστεί.

Παρατηρήσεις

- Μπορείτε να χρησιμοποιήσετε είτε c είτε c++. Στην περίπτωση της c++ απαγορεύεται η χρήση των βιβλιοθηκών της stl.
- Θα πρέπει να δοθεί makefile
- Η εκτέλεση του προγράμματος θα γίνεται: ./ngrams -i <init_file> -q <query_file> και η εκτύπωση θα γίνεται στο stdout
- Μαζί θα πρέπει να υπάρχει εκτενής αναφορά που να εξηγεί τις επιλογές σας
- Στο εβδομαδιαίο μάθημα, κρίνεται υποχρεωτική η παρουσία τουλάχιστον ενός μέλους της ομάδας