Πανεπιστήμιο Κρήτης, Τμήμα Επιστήμης Υπολογιστών
ΗΥ463 - Συστήματα Ανάκτησης Πληροφοριών
2005-2006 Εαρινό Εξάμηνο

3η Σειρά Ασκήσεων
Ανάθεση: 1 Απριλίου
Παράδοση: 14 Απριλίου

Άσκηση 1 *(3 βαθμοί) (Ενότητα: Ευρετηρίαση Κειμένου)*
Θεωρείστε ένα έγγραφο το οποίο περιέχει τα εξής λόγια του Βενιαμίν Φραγκλίνου:
«***Όποιος θυσιάζει την ελευθερία για την ασφάλεια δεν αξίζει ούτε την ελευθερία ούτε την ασφάλεια.***»
(α) Φτιάξτε το ανεστραμμένο ευρετήριο αυτού του εγγράφου.
(β) Φτιάξτε το δένδρο καταλήξεων του εγγράφου θεωρώντας ως σημεία ευρετηρίου (index points) τις αρχές των λέξεων (μπορείτε να δώσετε κατευθείαν το PATRICIA tree).
(γ) Σχολιάστε το μέγεθος των (α) και (β) ως προς το μέγεθος του εγγράφου.

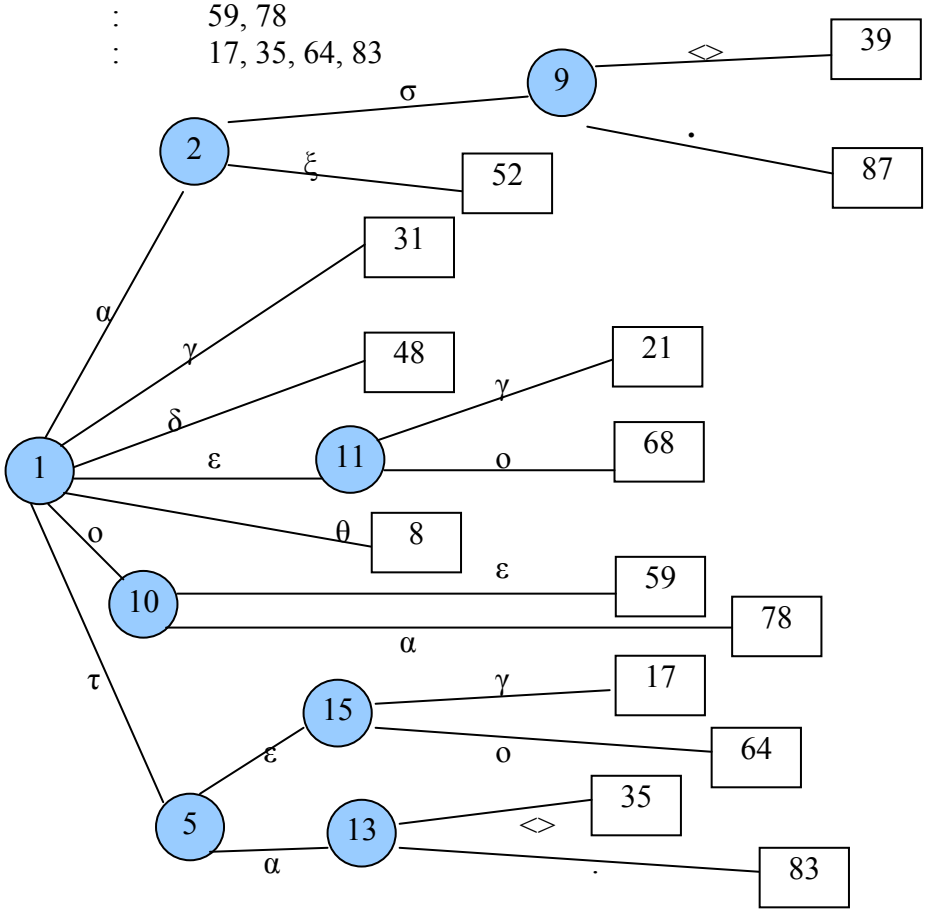
Λύση
α)

Αν θεωρήσουμε ότι δεν περνάμε το παραπάνω κείμενο από stoplist και stemming, το ανεστραμμένο ευρετήριο θα έχει ως εξής:

Ανεστραμμένο ευρετήριο :

ασφάλεια	:	39, 87
αξίζει	:	52
για	:	31
δεν	:	48
ελευθερία	:	21, 68
θυσιάζει	:	8
όποιος	:	1
ούτε	:	59, 78
την	:	17, 35, 64, 83

β)



Ασφαλώς θα μπορούσαμε να περάσουμε το κείμενο από stoplist, οπότε θα αγνοούσαμε λέξεις όπως το «για», «δεν», «την».

γ)

Το μέγεθος του ανεστραμμένου ευρετηρίου είναι μικρότερο από το μέγεθος του κανονικού αρχείου για ένα σχετικά μεγάλο κείμενο. Ο λόγος είναι ότι τα strings που επαναλαμβάνονται, αντί να καταλαμβάνουν χώρο όσο είναι το μέγεθος του string επί τον αριθμό των επαναλήψεων, καταλαμβάνουν τόσο χώρο όσο το μέγεθος του string συν τον αριθμό των επαναλήψεων επί το μέγεθος ενός ακέραιου αριθμού. Το δικό μας κείμενο όμως είναι σχετικά μικρό και επειδή δεν έχουμε βγάλει και τα stopwords, τα οποία είναι σχετικά μικρά (3 με 4 χαρακτήρες και θα μας βόλευε να τα κρατάμε σαν strings) ουσιαστικά από άποψη χώρου το ανεστραμμένο ευρετήριο είναι λίγο χειρότερο από το κανονικό text. Έτσι ενώ το κείμενο είναι 95 χαρακτήρες και αρά θα χρειαστούμε 95 bytes, για το ανεστραμμένο ευρετήριο χρειαζόμαστε 106 bytes.

Όσον αφορά το Patricia tree, είναι καταδικασμένο να καταλαμβάνει πολύ περισσότερο χώρο από το χώρο που θα καταλάμβανε το αρχείο. Ο λόγος είναι ότι για κάθε κόμβο χρειάζεται να καταλαμβάνουμε 4 bytes για να αποθηκεύουμε ένα ακέραιο, και για κάθε ακμή χρειαζόμαστε 5 bytes, αφού ουσιαστικά χρειάζεται να κρατάμε ένα δείκτη και ένα χαρακτήρα. Μετρώντας για το συγκεκριμένο κείμενο βρίσκουμε ότι χρειαζόμαστε 213 bytes. Αν το κείμενο μας ήταν ακόμα πιο μεγάλο η κατάσταση θα χειροτέρευε ακόμα περισσότερο μιας και θα έπρεπε για κάθε νέο πρόθεμα που θα εισάγαμε να δεσμεύσουμε 18 bytes επιπλέον (δύο ακμές, ένα επιπλέον occurrence και ένα node).

Άσκηση 2 *(5 βαθμοί) (Ενότητα: Ευρετηρίαση Κειμένου)*

Θέλετε να σχεδιάσετε ένα ΣΑΠ που να βασίζεται στο διανυσματικό μοντέλο για μια συλλογή κειμένων συνολικού μεγέθους στο δίσκο 1 Gigabyte. Έστω ότι το μέσο μέγεθος των λέξεων που εμφανίζονται στα κείμενα είναι 10 χαραχτήρες και ότι το πλήθος των διαφορετικών λέξεων της συλλογής είναι 10.000.

(α) Ποιο το αναμενόμενο (μέγιστο) μέγεθος του ανεστραμμένου ευρετηρίου για τη συλλογή αυτή;

(β) Ποιο το αναμενόμενο (μέγιστο) μέγεθος του ανεστραμμένου ευρετηρίου αν χρησιμοποιήσετε block addressing με μέγεθος block ίσο με 200 λέξεις;

Τι ποσοστό μείωσης έχουμε, σε σχέση με το (α);

(γ) Αν έπρεπε το ευρετήριο να καταλαμβάνει το πολύ 1 MB (π.χ. για να χωράει στην κύρια μνήμη ενός κινητού τηλεφώνου) πως θα σχεδιάζατε το ανεστραμμένο ευρετήριο;

(δ) Αν έπρεπε να καταλαμβάνει το πολύ 100 K τι θα κάνατε;

(ε) Αν έπρεπε να καταλαμβάνει το πολύ 10 K τι θα κάνατε;

(στ) Αν έπρεπε να υποστηρίξετε και phrase queries (μέγιστου μήκους 4 διαδοχικών λέξεων) και είχατε στη διάθεση σας μόνο 1 Mbyte μνήμης και επιλέγατε αντί για ανεστραμμένο ευρετήριο να είχατε αρχείο υπογραφών πως θα το σχεδιάζατε;

<u>Λύση</u>

Κατ' αρχήν θεωρούμε, είτε ότι στη γλώσσα μας δεν μας ενδιαφέρουν τα stopwords, είτε ότι ήδη στην συλλογή του 1Gbyte που έχουμε ήδη αφαιρέσει τα stopwords (αν τα είχαμε στη συλλογή μας το 10byte/λέξη θα φαινόταν λίγο υπερβολικό). Ακόμα θα κάνουμε τις εξής θεωρήσεις :

Σαν occurencies στους κόμβους των inverted lists θα κρατάμε σε ποιο document βρίσκεται ο όρος, σε ποιες θέσεις μέσα στο κείμενο εμφανίζεται και το tf-idf βάρος του όρου στο έγγραφο. Έτσι στην παρακάτω ανάλυση μας θεωρούμε ότι η συλλογή μας καθώς και το vocabulary αποτελούνται από μη-stopword λέξεις που γίνονται όλες indexing. Η συλλογή μας αποτελείται από 1 GByte = 1.000.000.000 bytes και με μέσο όρο 10bytes/λέξη η συλλογή μας περιέχει 100.000.000 λέξεις (δεν θεωρούμε κενά

μεταξύ λέξεων, αν θέλαμε να πάρουμε υπ' όψιν κενά η ανάλυση θα ήταν παρόμοια αφού μετά από κάθε λέξη θα παίρναμε υπ' όψιν μας ένα κενό). Γνωρίζουμε ότι η συλλογή μας περιέχει 10.000 διαφορετικές λέξεις και συνεπώς κάθε λέξη εμφανίζεται κατά μέσο όρο 100.000.000 / 10.000 = 10.000 φορές. Τέλος δεν αγνοούμε το μέγεθος των pointers (θα μπορούσαμε να λάβουμε υπ' όψιν 4Bytes επιβάρυνση για κάθε pointer με συνακόλουθες μικρές αλλαγές στα νούμερα παρακάτω).

α)

Για το vocabulary του ευρετηρίου θέλουμε να αποθηκεύσουμε τις 10.000 διαφορετικές λέξεις των 10byte και άρα θέλουμε χώρο 100.000 bytes = 100 KB. Αν για κάθε vocabulary entry κρατήσουμε και 4 bytes για το document frequency, το vocabulary θα καταλαμβάνει μόνο του χώρο 140K. Για κάθε μια entry από τις 10.000 θα αποθηκεύσουμε τόσα occurencies σε όλα τα κείμενα όσες φορές εμφανίζεται η λέξη και ήδη αναφέραμε ότι κάθε λέξη εμφανίζεται κατά μέσο όρο 10.000 φορές και άρα θέλει 10.000 integers για να αποθηκευτούν όλα τα occurencies μιας λέξης. Συνεπώς για ένα entry του vocabulary θέλουμε 10.000 * 4 bytes = 40.000 bytes = 40 KB. Αυτό μόνο για τις θέσεις των εμφανίσεων της ίδιας λέξης. Εκτός από αυτό πρέπει να αποθηκεύσουμε το σε ποια έγγραφα υπάρχουν αυτές οι εμφανίσεις καθώς και το tf-idf βάρος της (εμφάνισης της) λέξης στο κείμενο. Επειδή ψάχνουμε το μέγιστο μέγεθος ευρετηρίου που μπορεί να έχουμε θα κάνουμε την χειρότερη από άποψη χώρου υπόθεση, ότι κάθε εμφάνιση μιας λέξης βρίσκεται σε διαφορετικό κείμενο και άρα θέλουμε (για κάθε λέξη) 10.000 integers για να κρατάμε τα id του κειμένων που αυτή εμφανίζεται και άλλους 10.000 integers για τα tf-idf βάρη. Συνεπώς για ένα entry του vocabulary θέλουμε (μιλάμε πάντα κατά μέσο όρο) 40K για τις θέσεις των εμφανίσεων στα κείμενα, 40K για τα id των documents και άλλα 40K για τα βάρη. Σύνολο 120K για κάθε μια από τις entries του vocabulary. Συνεπώς για όλες τις inverted lists θέλουμε χώρο 120K * 10.000 = 1.200.000.000 bytes = 1,2 G.

Εν κατακλείδι για το ανεστραμένο αρχείο θέλουμε 1,2G + 140K = 1200,14 Mbytes.

β)

Το να χρησιμοποιήσουμε block addressing με μέγεθος block = 200 λέξεις * 10 byte/λέξη = 2000 bytes δεν έχει σαν αποτέλεσμα καμία βελτίωση στο μέγεθος του ευρετηρίου σε σχέση με το ερώτημα (α). Αυτό συμβαίνει διότι με μέγεθος block 2000 bytes , η 1G συλλογή μας χωρίζεται σε 500.000 blocks (περίπου), τα οποία είναι πάρα πολλά σε σχέση με το πλήθος των διαφορετικών λέξεων της συλλογής μας. Με την υπόθεση ότι οι εμφανίσεις μιας λέξης κατανέμονται ομοιόμορφα μέσα στο κείμενο, μια λέξη εμφανίζεται κάθε 10.000 άλλες άρα μια λέξη εμφανίζεται κάθε 10.000 / 200 = 50blocks.

Κατά συνέπεια το πλήθος των occurencies κάθε λέξης θα παραμείνει το ίδιο αφού ναι μεν πλέον ένας integer occurence θα δηλώνει θέση μπλόκ και όχι θέση στο κείμενο αλλά επειδή με μεγάλη πιθανότητα οι εμφανίσεις της ίδιας λέξης θα είναι σε διαφορετικά μπλοκ το πλήθος των μπλοκς που θα εμφανίζεται μια λέξη θα είναι 10.000 και θέλουμε τόσο χώρο όσο και στο (α). Κάνουμε πάλι την χείριστη υπόθεση ότι κάθε μπλοκ (και συνεπώς κάθε εμφάνιση λέξης) είναι σε διαφορετικό κείμενο, και άρα πάλι θέλουμε 3 ακέραιους (αριθμό μπλοκ, id εγγράφου, tf-idf βάρος) ανά κόμβο λίστας, έχοντας 10.000 κόμβους για κάθε μια από τις 10.000 λέξεις.

Συνεπώς και πάλι για το ανεστραμμένο αρχείο θέλουμε 1200,14 Mbytes.

γ)

Σε αυτό το ερώτημα θα χωρίσουμε τη συλλογή των εγγράφων μας σε μεγάλα μπλοκς, ώστε να χωρούν πολλές εμφανίσεις της ίδιας λέξης σε ένα block, και για κάθε λέξη στο ευρετήριο θα κρατούμε μόνο ακεραίους που θα είναι τα μπλοκ που βρέθηκε η λέξη. Το vocabulary του ευρετηρίου από μόνο του (και χωρίς document frequency πληροφορία) καταλαμβάνει 100K και άρα έχουμε ακόμα 900K στην διάθεσή μας. Αυτά τα 900K θα κατανεμηθούν σε 10000 εγγραφές (μια για κάθε λέξη) και έτσι κάθε inverted list θα έχει περίπου 90 bytes (μέσο όρο) χώρο για πληροφορίες σχετικές με την λέξη. Σε αυτά τα

90 bytes πρέπει να περιγραφούν και οι (κατά μέσο όρο) 10.000 εμφανίσεις της λέξης. Αν θεωρήσουμε, όπως παραπάνω, ότι για κάθε μπλοκ θα κρατούμε 4 bytes, τότε θέλουμε 22,5 μπλοκς (90=22,5*4) που θα έχουν τις 10000 εμφανίσεις της λέξης. Άρα σε ένα μπλοκ θέλουμε να εμφανίζεται η λέξη 10.000/ 22,5 = 444,44 φορές κατά μέσο όρο.Όπως αναφέραμε και στο ερώτημα (β) η ιδια λέξη εμφανίζεται κάθε 10.000 άλλες λέξεις και άρα για να περιέχει ένα μπλοκ 444,44 φορές την ίδια λέξη πρέπει να περιέχει συνολικά 10.000 * 444,44 λέξεις = 4.444.400 λέξεις. Η κάθε λέξη είναι 10 bytes και άρα ένα μπλοκ πρέπει να έχει μέγεθος 44.444.000 bytes = 44,44 Mbyte.

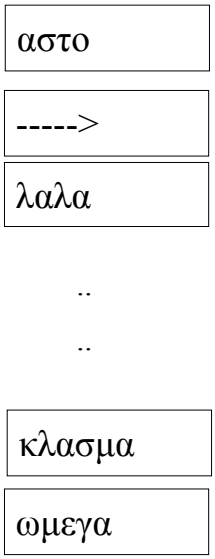
Συνεπώς με μπλοκ των 44,44 Mbyte, το ευρετήριο (κρατώντας πληροφορία μόνο 4 byte/block) έχει μέγεθος 90 bytes (ανά εγγραφή) * 10.000 διαφορετικές εγγραφές + 100K (το vocabulary) = 1Mbyte. Αν θέλαμε να κρατάμε πιο πολύ πληροφορία ανά μπλοκ, π.χ ένα ακόμη byte (έστω έναν pointer σε μια άλλη δομή στο δίσκο η οποία θα κρατά βάρη ή/και άλλη πληροφορία), τότε για παράδειγμα θα έπρεπε να έχουμε τα μισά μπλοκ σε πλήθος (αφού θα κρατούσαμε την διπλάσια πληροφορία) και συνεπώς διπλάσιο μέγεθος μπλοκ.

δ)

Το να καταλαμβάνει το ευρετηρίο 100K σημαίνει ότι το vocabulary ίσα που χωράει στη μνήμη αφού αυτό από μόνο του είναι 100K. Δεν έχουμε πολλές επιλογές παρά να κρατάμε στη μνήμη pointers για θέσεις στο δίσκο όπου βρίσκεται η υπόλοιπη πληροφορία του ευρετηρίου. Όμως το vocabulary καταλαμβάνει και τα 100K και δεν αφήνει χώρο για τίποτα άλλο. Μπορούμε να εφαρμόσουμε stemming στο vocabulary ώστε το μέγεθος της μέσης λέξης να μετατραπεί από 10 σε 6 bytes. Έτσι θα περισσεύουν 4 bytes ανά λέξη που θα μπορούσε να μπει ο pointer και πάλι να έχουμε 100K χώρο. Ακόμα και εάν έχει εφαρμοστεί stemming ήδη, το μεγεθος της μέσης λέξης (10 bytes) μας αφήνει περιθώρια να την κόψουμε και να εφαρμόσουμε την ιδέα μας. Έχουμε βέβαια το πρόβλημα ότι για μια λέξη που έχει κοινό πρόθεμα με κάποια του vocabulary μας πρέπει να φτάσουμε ώς το δίσκο για να καταλάβουμε ότι αποτύχαμε.

ε)

Όλες οι ιδέες που παρουσιάζονται (blocks,stemming και η παρούσα ιδέα) μπορούν βέβαια να εφαρμοστούν μόνες τους ή συνδυασμένες σε όλα τα ερωτήματα. Αυτό που μπορούμε να κάνουμε όταν έχουμε τόσο μικρή μνήμη όσο 10K είναι να «σπάσουμε» το ευρετήριο του προηγούμενου ερωτήματος σε δέκα κομμάτια και να έχουμε 1 κομμάτι κάθε φορά στη μνήμη. Αν δεν περιέχει τη λέξη που ψάχνουμε πάμε στο δίσκο και φορτώνουμε ένα άλλο κομμάτι. Η παραπάνω διαδικασία είναι πάρα πολύ αργή και μια βελτιστοποίηση είναι να χωρέσουμε κάποιες από τις λέξεις (π.χ. 900) αλφαβητικά επιλέγοντάς τες όσο περισσότερο ομοιόμορφα μπορούμε από όλο το σύνολο του vocabulary. Ανάμεσα σε λέξεις (π.χ κάθε 10 λέξεις) θα κρατούμε έναν pointer σε ένα τμήμα του vocabulary που είναι 10K και αλφαβητικά ανάμεσα στην λέξη πριν από τον pointer και στην επόμενη. Το παρακάτω σχήμα δείχνει τι εννοώ.



Αν ψάχνουμε μια λέξη λεξικογραφικά ανάμεσα σε «αστο» και «λαλα» ακολουθούμε τον pointer και φορτώνουμε εκείνο το κομμάτι του vocabulary που βρίσκεται λεξικογραφικά ανάμεσα σε «αστο» και «λαλα» και περιέχει ίσως την λέξη που θέλουμε. Αν χρειάζεται μπορούμε να προσθέσουμε και άλλα επίπεδα indexing (δηλαδή αυτό το τμήμα που θα φορτώσουμε να έχει και αυτό pointers σε λεξικογραφικά εμπεριέχοντα τμήματα). Όπως και να έχει το τελευταίο επίπεδο indexing του vocabulary πρεπει να περιέχει pointers στον δίσκο για περαιτέρω πληροφορία (pointers π.χ. σε inverted lists).

στ)

Το μέγεθος του αρχείου υπογραφών είναι τα bit masks των μπλοκ συν ένας pointer για κάθε μπλοκ. Για να μην έχουμε conflicted hashes δυο block σε ίδιο bit mask, θα χρησιμοποιήσουμε B = log(10.000)=10, όπου B το πλήθος των bits ανά mask. Για κάθε μπλοκ θα αποθηκέυουμε λοιπόν 10 bits = 1,25 bytes (κατά μέσο όρο) + 4bytes ο pointer στο block της μάσκας. Σε αυτό το σημείο μπορούμε να θεωρήσουμε είτε ότι θα δώσουμε εν τέλει 1byte = 8 bit ανά mask, με αναμενόμενη τετραπλασίαση των false drops (αφού 8 bits μπορουν να κωδικοποιήσουν έως $2^8 = 256$ masks = ¼ των διαφορετικών λέξεων), είτε ότι η αρχιτεκτονική της μηχανής μας, μας επιτρέπει να γράφουμε σειριακά bits μέσα σε ένα byte και οπότε μπορεί να υπάρξει ένα byte, στο οποίο μερικά bit ανήκουν σε ένα mask ενώ τα υπόλοιπα σε άλλο. Εμείς (λόγω και του περιορισμού χώρου) θα επιλέξουμε να δώσουμε 1 byte ανα mask.

Συνεπώς στο αρχείο υπογραφών για κάθε μπλοκ κειμένου μας χρειαζόμαστε 1+4 = 5 bytes. Το 1 MB = 1.000.000 bytes της μνήμης, μας αφήνει περιθώριο να έχουμε συνολικά 200.000 blocks. Άρα χωρίζουμε το 1G τις συλλογής μας σε 200.000 blocks των 5.000 bytes (= 5KB). Άρα κάθε μπλοκ θα έχει β = 500 λέξεις (αφού η λέξη είναι 10 bytes).

Κλείνοντας παρατηρούμε πως μπορούμε να εφαρμόσουμε και overlapping μεταξύ των blocks σε πλήθος λέξεων εξαρτώμενο από το proximity των proximity queries ή το μέσο μέγεθος των phrase queries.

Άσκηση 3 *(2 βαθμοί)*

Το Google θέλει να υποστηρίξει εξατομικευμένη διαβάθμιση των ιστοσελίδων στους χρήστες του. Κάθε χρήστης να μπορεί να ορίζει ένα ή περισσότερα προφίλ. Κάθε προφίλ (ενός χρήστη) θα έχει ένα όνομα (π.χ. ψυχαγωγία, my Master, MyMusic, computer science, κλπ) το οποίο ο χρήστης θα δίδει αρχικά. Η διαδικασία της αναζήτησης θα τροποποιηθεί ως εξής: Δίπλα στο πλαίσιο διατύπωσης επερωτήσεων του Google, θα υπάρχει ένα μενού το οποίο θα εμφανίζει όλα τα ονόματα προφίλ που έχει δηλώσει ο χρήστης καθώς και την προεπιλεγμένη επιλογή «ΧωρίςΠροφίλ». Η αναζήτηση με επιλογή «ΧωρίςΠροφίλ» θα διενεργείται όπως γίνεται σήμερα. Αν ο χρήστης έχει επιλέξει ένα προφίλ, τότε δίπλα σε κάθε στοιχείο της απάντησης (σύνδεσμο προς ιστοσελίδα) της κάθε απάντησης θα εμφανίζονται δυο κομβία: ένα good και ένα bad. Ανάλογα με το περιεχόμενο της κάθε σελίδα και τις προτιμήσεις του χρήστη (και για το συγκεκριμένο προφίλ), ο χρήστης θα μπορεί να πατήσει το good ή το bad για όποιες σελίδες το επιθυμεί. Βάσει αυτής της ανατροφοδότησης το Google πρέπει να παρέχει εξατομικευμένη διαβάθμιση σελίδων..

(α) Πως θα αξιοποιούσατε τα good και bad εισόδους του χρήστη ώστε να υποστηρίξετε εξατομικευμένη ανάκτηση; Μπορείτε να περιγράψετε παραπάνω από ένα τρόπους.

(β) Για κάθε προφίλ ενός χρήστη τι θα αποθηκεύατε; Λάβετε υπόψη ότι εκατομμύρια χρήστες χρησιμοποιούν το Google καθώς και το γεγονός ότι το Google ευρετηριάζει περίπου 9 δις σελίδες.

(γ) Ποιο θα ήταν το μεγαλύτερο τεχνικό πρόβλημα για την υποστήριξη αυτής της λειτουργικότητας;

Αν προτείνατε παραπάνω από μια μέθοδο απαντήστε αυτό το ερώτημα για κάθε μία μέθοδο και αναφέρετε τη μέθοδο που θα επιλέγατε (και δικαιολογείστε).

Λύση

Ακολουθούν 2 λύσεις που μπορούν να θεωρηθούν σαν συμπληρωματικές η μία ως προς την άλλη (χωρίς να σημαίνει πως δεν υπάρχουν κι άλλες λύσεις).

1η Λύση:

α)

Θα μπορούσαμε για να αξιοποιήσουμε την good/bad πληροφορία ως εξής: Για ένα προφίλ όπου ο χρήστης έχει δώσει προτιμήσεις θα μπορούσαμε να τροποποιούμε το ranking των απαντήσεων που δίνουμε λαμβάνοντας υπ' όψιν τις προηγούμενες απαντήσεις του χρήστη στο προφίλ αυτό. Σε μια απόλυτη περίπτωση θα μπορούσαμε να εμφανίζουμε πρωτα τα good αποτελέσματα στον χρήστη (ταξινομημένα ως προς το μεταξύ τους ranking), έπειτα τα ουδέτερα αποτελέσματα (αυτά που δεν απάντησε ούτε good είτε bad) και αυτά ταξινομημένα ως προς το μεταξύ τους ranking και στο τέλος τα bad αποτελέσματα (αν και αυτά μπορούμε και να τα παραλείψουμε). Αυτή η προσέγγιση όμως υστερεί από την άποψη ότι δεν εκμεταλλευόμαστε την επανεπιλογή ενός εγγράφου σαν good (αυτό θα σήμαινε ότι του χρήστη του αρέσει πολύ) και επίσης όταν στο ίδιο προφίλ για διαφορετικά queries ο χρήστης έχει δώσει το ίδιο έγγραφο σαν good και bad η παραπάνω προσέγγιση θα το κατατάξει σύμφωνα με την τελευταία αξιολόγηση του χρήστη (αφού π.χ. για ένα έγγραφο η παραπάνω προσέγγιση θα διατηρεί 2 λίστες με good και bad έγγραφα και θα προσθέτει/αφαιρεί από αυτές καθώς και θα μετακινεί έγγραφα από την μια λίστα στην άλλη).

Μια αλλη προσέγγιση θα ήταν τα good και bad να είχαν μια έννοια ψήφων. Το good θα ήταν θετική ψήφος για ένα έγγραφο (+1) ενώ το bad αρνητική (-1 και όταν ένα έγγραφο εχει <0 σύνολο ψήφων (ή π.χ. <-5) να μην εμφανίζεται καθόλου). Έτσι τώρα για κάθε προφίλ υπάρχει μια λίστα με έγγραφα κατεταγμένα ως προς σύνολο ψήφων και θα εμφανίζονται πρώτα τα έγγραφα που θα έχουν το ίδιο σύνολο ψήφων (κατεταγμένα ως προς το ranking του ΣΑΠ) πριν εμφανιστούν έγγραφα με μικρότερο σύνολο ψήφων κ.ο.κ.

Μια βελτιστοποίηση της πιο πάνω ιδέας θα ήταν να προσπαθήσουμε να ευνοήσουμε documents τα οποία μοιάζουν (π.χ. όσον αφορά στο cosSim μέτρο) με τα good διανύσματά μας άρα κατά την διαδικασία της απάντησης παρουσιάζουμε στον χρήστη τα έγγραφα με το υψηλότερο σύνολο ψήφων και αυτά (που μας επέστρεψε το ΣΑΠ και) που μοιάζουν πολύ με κάποιο έγγραφο από αυτά της υψηλής προτίμησης (τα παρουσιάζουμε όλα μαζί ταξινομημένα ως προς το ranking του ΣΑΠ). Έπειτα «κατεβαίνουμε» επίπεδο ψήφων και παρουσιάζουμε (κατεταγμένα ως προς το ranking) αυτά που εχουν αμέσως λιγότερους ψήφους και αυτά που μας επέστρεψε το ΣΑΠ και «μοιάζουν» με τα πρώτα κ.οκ.
Εναλλακτικά μια προσέγγιση που θα απορρίψω στο ερώτημα (γ) έχει να κάνει με το να διατηρούμε και πληροφορία όπως το σε ποιο query έδωσε το feedback ο χρήστης.Έτσι διατηρώντας για κάθε διαφορετικό query διαφορετική λίστα με good και bad έγγραφα και όταν ο χρήστης έκανε ένα query που μοιάζει με ένα(ή περισσότερα) query για τα οποία έχουμε good και bad έγγραφα του επιστρέφουμε τα έγγραφα που ξέρουμε ότι είναι good (και έγγραφα που μοιάζουν με αυτά) και όχι αυτά που είναι bad (ή έγγραφα που μοιάζουν με αυτά).
(Ακόμα μια προσέγγιση που είναι ουτοπική φαίνεται στο ερώτημα (β)).

β)

Σύμφωνα με την πιο πάνω προσέγγιση των ψήφων για κάθε προφίλ χρήστη θα αποθηκεύαμε μια λίστα με τα id's των εγγράφων για τα οποία έχει δώσει feedback ο χρήστης και έναν αριθμό που δηλώνει τις «ψήφους» κάθε εγγράφου.
(Σε ένα κόσμο όπου το αποθηκευτικό κόστός θα ήταν αμελητέο θα μπορούσαμε να διατηρούμε και πληροφορία όπως το σε ποιο query έδωσε το feedback ο χρήστης ή ακόμα και να έχουμε διαφορετικές συλλογές διανυσμάτων εγγράφων για κάθε προφίλ με τα βάρη των διανυσμάτων των εγγράφων κάθε

συλλογής να έχουν γίνει train από το feedback του χρήστη ώστε να εμφανίζονται ψηλά τα έγγραφα που θέλουμε στο συγκεκριμένο προφίλ)

γ)

Το κυριότερο πρόβλημα είναι το πρόβλημα χώρου (για κάθε προφίλ κάθε χρήστη μια λίστα με όλα τα id's των εγγράφων για τα οποία ο χρήστης έχει δώσει απάντηση). Επίσης υπάρχει και ένα επιπλέον υπολογιστικό κόστος στην αποτίμηση των επερωτήσεων (για να τροποποιηθούν τα αποτελέσματα σύμφωνα με τις λίστες όπως περιγράψαμε παραπάνω). Θα επέλεγα τον τρόπο που περιέγραψα πιο πάνω (με μια λίστα με Document_id's – ψήφους ανά προφίλ) γιατί φαίνεται ο πιο εφικτός σε σχέση με το να είχαμε διαφορετικές συλλογές διανυσμάτων εγγράφων για κάθε προφίλ, καθενός χρήστη το οποίο θα πολλαπλασίαζε τα 9 δις διανύσματα σελίδων που έχει το Google σήμερα * κάθε προφίλ * κάθε χρήστη. Το να κρατάμε πληροφορία όπως το σε ποιο query έδωσε το feedback ο χρήστης είναι επίσης πάρα πολύ σπάταλο αφού θέλει μια λίστα για κάθε query (!), που γίνεται σε κάθε προφίλ από κάθε χρήστη.

2η Λύση:

α)

Κάθε φορά που ο χρήστης εκτελεί μια επερώτηση, τα έγγραφα που επιστρέφονται εξετάζονται από το χρήστη και κρίνονται από αυτόν ως "good" ή "bad". Οι λέξεις εξάγονται αυτόματα από τα έγγραφα που αξιολογήθηκαν και χρησιμοποιούνται για να τροποποιήσουν την αρχική επερώτηση. Η επαναδιατύπωση των επερωτήσεων μπορεί να γίνει με πολλούς τρόπους. Μπορεί να προστεθούν ή να αφαιρεθούν όροι στην επερώτηση ή ρυθμιστούν τα βάρη των όρων του διανύσματος επερώτησης, ώστε τα βάρη των συναφών όρων να σταθμιστούν υψηλότερα. Σε συστήματα που χρησιμοποιείται το διανυσματικό μοντέλο για την αναπαράσταση των εγγράφων και των επερωτήσεων, η όλη διαδικασία της ανάδρασης συνάφειας μπορεί να θεωρηθεί ως μια προσπάθεια να μετακινήσουμε το διάνυσμα επερώτησης πλησιέστερα στα διανύσματα που αναπαριστάνουν τα αξιολογημένα ως συναφή έγγραφα. Πολλοί αλγόριθμοι έχουν χρησιμοποιηθεί για να κάνουν αυτήν την τροποποίηση, με την αρχική ιδέα να έχει δοθεί από το Roccio.

Η τροποποιημένη επερώτηση επαναδιατυπώνεται στην μηχανή αναζήτησης και ένα νέο σύνολο από έγγραφα επιστρέφεται και η διαδικασία προχωράει. Η όλη αυτή διαδικασία θα γίνει εξατομικευμένη ανάκτηση εάν αυτή η κρίση σχετικά με τη συνάφεια των εγγράφων αντιγραφεί σε ένα προφίλ χρήστη και χρησιμοποιηθούν στις επόμενες αναζητήσεις για να βελτιωθεί η ανάκτηση εγγράφων και οι διαβαθμίσεις των αποτελεσμάτων αναζήτησης. Η αξιολόγηση των μη-συναφών σελίδων μπορεί να χρησιμοποιηθεί ως φιλτράρισμα, ώστε η μηχανή αναζήτησης να μην τις επιστρέφει στα αποτελέσματα των επερωτήσεων του αντίστοιχου προφίλ.

β)

Το σύστημα πρέπει να ρυθμίζει την επερώτηση που δόθηκε από τον χρήστη σύμφωνα με το αντίστοιχο προφίλ χρήστη. Ο κλασικός τρόπος είναι ανάλογα με το προφίλ που έχει επιλέξει ο χρήστης (το έχει δημιουργήσει ο ίδιος) να αποθηκευτεί και το αντίστοιχο διάνυσμα που αντανακλά τις κρίσεις του χρήστη στα επιστρεφόμενα έγγραφα. Έτσι κάθε φορά που είναι να εκτελεστεί μία επερώτηση, θα διάνυσμα της τελικής επερώτησης θα είναι της μορφής $q' = q * (1+p)$, όπου q το αρχικό διάνυσμα της επερώτησης και p το διάνυσμα του συγκεκριμένου προφίλ. Αυτό το διάνυσμα θα περιέχει τις υψηλότερες τιμές βάρους για τους όρους εκείνους που κρίθηκαν συναφή με τη διαδικασία που περιγράψαμε προηγουμένως. Επίσης μπορούμε να χρησιμοποιήσουμε το προφίλ όχι ως μέθοδος επαναδιατύπωσης της επερώτησης αλλά για να προσδιορίσουμε καλύτερα την αναζήτηση. Δηλαδή μέσα από τους όρους που έχουν υψηλότερο βάρος στο διάνυσμα του προφίλ, η μηχανή αναζήτησης θα αντιστοιχεί την επερώτηση εισόδου σε ένα σύνολο από ενδιαφέρουσες κατηγορίες, βασισμένες στο προφίλ χρήστη, και περιορίζει την περιοχή αναζήτησης σε αυτές τις κατηγορίες.

γ)

Το μεγαλύτερο πρόβλημα για την υποστήριξη αυτής της λειτουργικότητας είναι η δυσκολία απόκτησης ανάδρασης συνάφειας με σαφή τρόπο από τους χρήστες σε εξατομικευμένες διαδικτυακές μηχανές αναζήτησης, λόγο του τεράστιου μεγέθους του ιστού, ο οποίος αποτελείται από δισεκατομμύρια έγγραφα. Είναι ιδιαίτερα χρονοβόρο και σχεδόν αδύνατο να συλλεχθούν κρίσεις συνάφειας από τον κάθε χρήστη, για κάθε σελίδα που επιστρέφεται ως αποτέλεσμα μίας επερώτησης. Οι χρήστες γενικά απεχθάνονται να έχουν να ξοδέψουν χρόνο και προσπάθεια υποβάλλοντας δεδομένα σε οποιοδήποτε σύστημα, ιδιαίτερα αν τα πλεονεκτήματα δεν είναι αμέσως εμφανή. Επίσης οι χρήστες αποφεύγουν να δώσουν προσωπικές πληροφορίες στους ανώνυμους servers για λόγους απορρήτου και ασφάλειας. Τέλος ένα ακόμα πρόβλημα βρίσκεται στην διατήρηση της αποδοτικότητας. Υπάρχει χρονική πολυπλοκότητα όσο μεγαλώνει το μέγεθος των προφίλ των χρηστών (π.χ αυξανόμενος αριθμός λέξεων, ενδιαφέρων κατηγοριών και περιοχών).