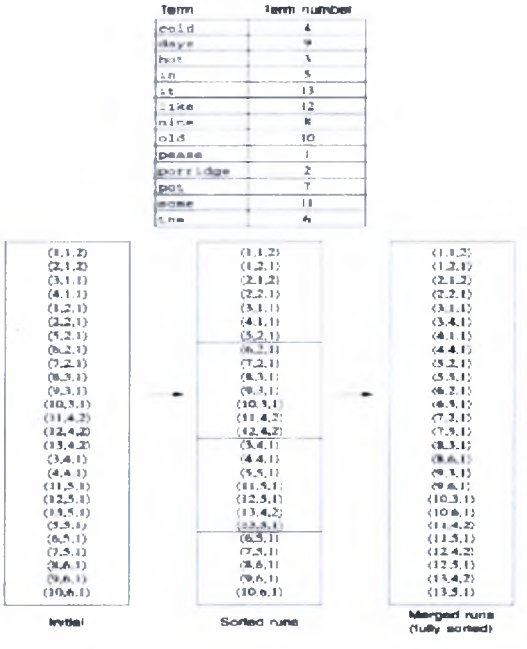
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΑΝΙΔΟΥ ΜΑΡΙΑ

Διπλωματική εργασία

Θέμα: ΤΕΧΝΙΚΕΣ ΔΟΜΗΣΗΣ ΑΝΤΕΣΤΡΑΜΜΕΝΩΝ ΛΙΣΤΩΝ

Επιβλέπων καθηγητής : ΜΠΟΖΑΝΗΣ ΠΑΝΑΓΙΩΤΗΣ



Οκτώβριος 2008

ΠΕΡΙΕΧΟΜΕΝΑ

Στην οικογένεια μου

Εισαγωγή

Τα τελευταία χρόνια έχει εκτοξευθεί η χρήση των υπολογιστικών συστημάτων. Κατά συνέπεια αυξάνεται εκθετικά ο όγκος των πληροφοριών που διακινείται μέσω δικτύων. Για την ανάκτηση και επεξεργασία αυτού του τεράστιου όγκου πληροφοριών έχουν αναπτυχθεί πολλές και διάφορες δομές που αποσκοπούν στην βελτιστοποίηση της επιτάχυνσης των διαδικασιών επεξεργασίας τους.
Ο όγκος αυτών των πληροφοριών διαχωρίζεται σε συλλογές. Οι διαδικασίες που μας ενδιαφέρον επί το πλείστον είναι οι διαδικασίες ανάκτησης πληροφορίας. Οπότε υπάρχουν μοντέλα που ασχολούνται με τον τρόπο ανάκτησης πληροφοριών από μια συλλογή.
Ένα ευρετήριο αποθηκεύει πληροφορίες που εξάγει από μια συλλογή κειμένων. Στόχος είναι η βελτίωση των μοντέλων που αποσκοπούν στην εξαγωγή πληροφοριών. Για να επιτευχθεί αυτό θα πρέπει επαγωγικά να βελτιστοποιηθεί η επίδοση των ευρετηρίων.
Στο κεφάλαιο 1 εξετάζονται τα βασικά συστατικά μιας μηχανής αναζήτησης καθώς και τα αντεστραμμένα αρχεία, δηλαδή τα inverted files. Επιπλέον γίνεται μια σύντομη αναφορά στα bitmaps και στα signature files.
Στο κεφάλαιο 2 δίδεται ο ορισμός των inverted files καθώς και οι αλγόριθμοι κατασκευής αυτών.
Στο κεφάλαιο 3 γίνεται αναλυτική επισκόπηση επί της συμπίεσης των αντεστραμμένων αρχείων. Αναλύονται τα μοντέλα κατανομής πιθανοτήτων και οι αλγόριθμοι συμπίεσης.
Τέλος στο κεφάλαιο 4 γίνεται αναφορά στον τρόπο κατασκευής των inverted files, signature files και bitmaps. Επιπλέον δίνεται ο τρόπος για την επεξεργασία ερωτήματος ώστε να ανακτηθεί η ζητούμενη πληροφορία.

1. Text Search Engines

1.1. Ορισμός των text search engines

Μία search engine (*μηχανή αναζήτησης*) είναι ένα σύστημα ανάκτησης πληροφοριών που αποσκοπεί στο να βοηθήσει τον χρήστη να εντοπίσει και να ανακτήσει αποθηκευμένες πληροφορίες από το σύστημα. Παρέχουν μια διεπαφή σε μια ομάδα στοιχείων που επιτρέπει στους χρήστες να διευκρινήσουν τα στοιχεία που τους ενδιαφέρουν και να αντιστοιχήσει τα στοιχεία αυτά με τα ανάλογα δεδομενα. Τα κριτήρια αυτά αναφέρονται ως ερώτηση αναζήτησης. Στην περίπτωση των μηχανών αναζήτησης κειμένων (*text search engines*) η ερώτηση αναζήτησης εκφράζεται ως ένα σύνολο λέξεων που προσδιορίζουν την επιθυμητή έννοια που ένα ή περισσότερα έγγραφα μπορεί να περιέχουν. Τα αντικείμενα που αποθηκεύονται σε μια συλλογή δεδομένων ενδέχεται να εμπεριέχουν, όχι μόνο αποκλειστικά κείμενα, αλλά και εικόνες, αρχεία ήχου και εικόνας κ.τ.λ. Θα ασχοληθούμε όμως μόνο με το κείμενο.

Τα text search engines ελαχιστοποιούν τον χρόνο αναζήτησης πληροφοριών και την ποσότητα πληροφοριών που πρέπει να χρησιμοποιηθούν για να εντοπιστεί η απάντηση του χρήστη στο ερώτημα του.

1.2. Τρόπος Λειτουργίας και Εφαρμογής των Text Search Engines

Η πιο ορατή και διαδεδομένη μηχανή αναζήτησης είναι ο Ιστοχώρος όπου γίνεται έρευνα για πληροφορίες για το World Wide Web. Στην συγκεκριμένη μηχανή αναζήτησης χρησιμοποιείται μια εφαρμογή που επιτρέπει την αναζήτηση κειμένων και αρχείων στο Διαδίκτυο. Αποτελείται από ένα *πρόγραμμα υπολογιστή,* που βρίσκεται σε έναν ή περισσότερους υπολογιστές στους οποίους δημιουργείται και διατηρείται μια *βάση δεδομένων* με τις πληροφορίες που συλλέγονται από το διαδίκτυο, και από το *διαδραστικό περιβάλλον* που εμφανίζεται στον τελικό χρήστη ο οποίος είναι συνδεδεμένος στο διαδίκτυο μέσω ενός υπολογιστή όπου «τρέχει» η εφαρμογή. Οι Βάσεις Δεδομένων τους είναι για την ακρίβεια *Βάσεις Δεδομένων ευρετηρίων*. Δηλαδή περιέχουν τον τίτλο, το μέγεθος, το πλήρες κείμενο και των URL των ιστοσελίδων. Σαν αποτέλεσμα η αναζήτηση γίνεται σε αυτές τις Βάσεις Δεδομένων και όχι σε ολόκληρο τον ιστοχώρο ώστε να έχουμε ταχύτερη αναζήτηση.

1.2.1 Τα μέρη μιας Μηχανής Αναζήτησης

1.2.1.1 Ειδικά προγράμματα

Η δημιουργία αυτών των Βάσεων Δεδομένων γίνεται με ειδικά προγράμματα, όπως είναι τα spiders, robots και crawlers, τα οποία συλλέγουν το πληροφοριακό υλικό και το τοποθετούν σε αυτές τις Βάσεις Δεδομένων.

1.2.1.2 Μηχανισμός Ευρετηρίου

Ο Μηχανισμός Ευρετηρίου αποθηκεύει τα κείμενα των ιστοσελίδων στην πλήρη τους μορφή με τρόπο ανεστραμμένου ευρετηρίου στις Βάσεις Δεδομένων μιας μηχανής αναζήτησης.

1.2.1.3 Μηχανισμός Αναζήτησης

Αποτελεί το πιο πολύπλοκο τμήμα μιας μηχανής αναζήτησης. Ένας Μηχανισμός Αναζήτησης έχει τα εξής επιμέρους τμήματα:

a) Φόρμα αναζήτησης και διασύνδεσης με τον τελικό χρήστη
b) Μηχανισμό αξιολόγησης ερωτήματος στην Βάση Δεδομένων και εντοπισμού σχετκών ιστοσελίδων στα ευρετήρια της μηχανής
c) Μορφοποιητής αποτελεσμάτων

1.3 Αλγόριθμοι των Text Search Engines

Κάθε μηχανή αναζήτησης χρησιμοποιεί διαφορετικό αλγόριθμο. Ένας αλγόριθμος ορίζεται ως ένα σύνολο από κανόνες που χρησιμοποιούν οι μηχανές αναζήτησης προκειμένου να αξιολογήσουν τις π.χ. ιστοσελίδες που περιλαμβάνουν στις βάσεις δεδομένων τους, σε σχέση με μια συγκεκριμένη αναζήτηση. Το αποτέλεσμα της αναζήτησης, δηλαδή η ποιότητα και σχετικότητα, εξαρτάται από τον αλγόριθμο που χρησιμοποιείται. Υπάρχουν τριες κύριες επιλογές για τον τρόπο που γίνεται η αναζήτηση δεδομένων στην βάση κατά την αξιολόγηση του ερωτήματος (*query evaluation*). Αυτές είναι:

a) Αναζήτηση και ανίχνευση όλης της συλλογής. Μία μέθοδος που είναι χαρακτηριστική στα πρόωρα συστήματα ανάκτησης δεδομένω. Λόγω του γεγονότος ότι οι υπολογιστικές και I/O δαπάνες μονάδων χρόνου είναι απαγορευτικές για μεγάλες συλλογές, χρησιμοποιείται μόνο σε μικρές συλλογές κειμένων και σε συστήματα μεγάλης μνήμης.
b) Δείκτες για άμεση πρόσβαση. Μέθοδος περισσότερο πρακτική για μεγάλες συλλογές και με πολλές δυνατότητες βελτιστοποίησης. Η αναζήτηση σε text search engines γίνεται κυρίως βάσει περιεχομένου του κειμένου έτσι ώστε το αποτέλεσμα της αναζήτησης να ικανοποιεί τον χρήστη. Ο πιο συνήθης τρόπος αναζήτησης και εύρεσης είναι αρχική η χρήση του ερωτήματος (*query*), η δημιουργία μιας λίστας προτεινόμενων αποτελεσμάτων με βάση την λέξη κλειδί, ή το σύνολο των λέξεων κλειδιά, και η εύρεση του ζητούμενου εγγράφου μέσω των pointers που αντιστοιχούν στα συγκεκριμένα αρχεία. Για την δημιουργία αυτής της λίστας χρησιμοποιούμε το index. Για παράδειγμα το index ενός βιβλίου περιέχει λέξεις, όρους τα οποία αντιστοιχούν στις σελίδες του βιβλίου. Η πιο αποτελεσματική δομή ενός index για την αξιολόγηση του ερωτήματος είναι το inverted file. Αξίζει να σημειωθεί ότι σε συλλογές εικόνων όπου χρησιμοποιείται η αναζήτηση με την βοήθεια των 'πλησιέστερων γειτόνων' (*Nearest Neighbor*) η χρήση δεικτών δεν ενδείκνυται.

c) Hybrids. Χρησιμοποιούνται δείκτες για υποσύνολα της συλλογής. Σαν αποτέλεσμα η αναζήτηση γίνεται στα υποσύνολα.

1.4 Το πρόβλημα του document retrieval

Τα δύο κύρια προβλήματα που αντιμετωπίζει κάποιος στην ανάκτηση δεδομένων είναι:

- Ο χώρος που απαιτείται απο το σύστημα για την αποθήκευση των τεράστιων αριθμών κειμένων
- Και ο χρόνος που χρειάζεται για τον έλεγχο αυτών ώστε να εντοπιστεί η ζητούμενη πληροφορία.

Η λύση του προβλήματος είναι ο συνδυασμός των τεχνικών συμπίεσης των κειμένων (*compression techniques),* ώστε να ελαχιστοποιηθεί στο μέγιστο ο χώρος που καταλαμβάνουν, με τις τεχνικές δεικτιοδότητης (*indexing techniques*).

1.5 Textual images

Ένα σύστημα Βάσεων Δεδομένων θα πρέπει να δύναται να αποθηκεύει εκτός από κείμενα και εικόνες. Οι εικόνες, που συνήθως συναντιούνται σε μορφή διαγραμμάτων ή φωτογραφιών, αποτελούν σημαντικό μέρος πολλών κειμένων. Το λάθος που γίνεται κάποιες φορές είναι ότι δεν υπολογίζεται στο σύστημα ο χώρος που καταλαμβάνουν οι εικόνες. Κάτι που είναι πολύ απαραίτητο, αν και δύσκολο, διότι δεν είναι τόσο εύκολος ο υπολογισμός των όγκων τους σε σύγκριση με τα απλά κείμενα.
Κάποιες φορές υπάρχει και η ανάγκη το κείμενο να αποθηκευθεί με μορφή εικόνας. Αυτό το είδος εικόνας ονομάζεται textual images. Για παράδειγμα η σάρωση εγγράφων και αποθήκευση τους ως εικόνα. Για αυτόν τον λόγο πρέπει το σύστημα να μπορεί να αποθηκεύσει και να ευρετηριοποιήσει τις εικόνες που περιέχονται σε κάποιο κείμενο.

1.6 Indexes

Όπως αναφέρθηκε το index χρησιμοποιείται για την δημιουργία λίστας όρων. Βοηθάει στην οργάνωση των πληροφοριών έτσι ώστε οι όροι της αναζήτησης να εντοπιστούν μέσα στην Βάση Δεδομένων και να εξαχθούν. Για γρηγορότερη αναζήτηση άρα και ανάκτηση δεδομένων ένας index μπορεί κα περιλαμβάνει πρωτεύον αλλά και δευτερεύον λέξη κλειδί. Η δημιουργία υποσυνόλων, υποδιαιρεί την συλλογή και η αναζήτηση δεν γίνεται πλέον σε ολόκληρη την συλλογή αλλά σε επιμέρους τμήματα αυτής. Λόγω του ότι το πρόβλημα των συστημάτων ανάκτησης (*IR*) είναι η μη δυνατή η πρόβλεψη των όρων που θα χρησιμοποιηθούν στην ερώτηση αναζήτησης, η μόνη βέβαια λύση είναι να γίνει ευρετηριοποίηση όλων των όρων. Σε κάποια συστήματα οι πολλοί κοινοί όροι, όπως 'και', 'το', 'of', 'the', 'and', επειδή χρειάζονται αρκετό χώρο αποθήκευσης και δεν θεωρούνται απαραίτητοι, δεν περιλαμβάνονται. Οι όροι αυτοί αποκαλούνται *stop words*.

Κοινές εφαρμογές των indexes είναι τα signature files, τα bitmaps και τα inverted files. Υπό συγκεκριμένες προυποθέσεις τόσο τα signature files όσο και τα bitmaps προσφέρουν γρηγορότερα αποτελέσματα αλλά υπό τις ίδιες συνθήκες απαιτούν πολύ περισσότερο χώρο αποθήκευση από τα inverted files. Πιο αναλυτικά θα εξετάσουμε κάποια βασικά στοιχεία των signature files και bitmaps, ενώ με τα inverted files θα επεκτείνουμε την εξέταση παρακάτω:

1.6.1 Bitmaps

Τα bitmaps είναι μια πολύ απλή δομή δεικτιοδότησης. Για κάθε όρο του συνόλου των λέξεων που αρχειοθετούνται (*lexicon*), αποθηκεύεται ένα bit που αντιπροσωπεύει την ύπαρξη ή μη του όρου στο κείμενο. Το bit που αντιστοιχεί στο 1 σημαίνει ότι σε αυτήν την γραμμή του κειμένου εμφανίζεται αυτός ο όρος, ενώ το 0 δηλώνει την απουσία του. Ένα παράδειγμα είναι το ακόλουθο:

Bitmap index

Πίνακας 1

Number	Term	Bitvector
1	cold	001101
2	days	110111
3	hot	000100
4	in	010000
5	nine	111000
6	the	111110

Στον πίνακα βλέπουμε ότι για τον όρο 'nine' έχουμε 111000 που σημαίνει ότι συναντάμε αυτόν τον όρο στις γραμμές 1,2 και 3 του κειμένου.
Τα bitmaps είναι γρήγορα και εύκολα στην χρήση αλλά καταλαμβάνουν τεράστιο χώρο αποθήκευσης. Για ένα κείμενο που αποτελείται από N γραμμές και έχει n διαφορετικούς όρους χρειαζόμαστε (N x n) bits. Το μέγεθος μπορεί να ελαττωθεί εάν δεν περιλαμβάνονται τα stop words, δίχως όμως να παρατηρούμε δραματικές αλλαγές.

1.6.2 Signature files

Για τα signature files είναι απαραίτητο καταρχάς η δημιουργία ενός *hash table* όπου θα δηλώνεται η εμφάνιση ή όχι του όρου. Για την κατασκευή αρχικά κάθε διακριτός όρος της συλλογής κειμένων υπογράφετε με μήκος F ψηφίων μέσω κατάλληλης διαδικασίας κατακερματισμού. Εν συνεχεία, για κάθε κείμενο της συλλογής υπολογίζεται μια ξεχωριστή υπογραφή του ίδιου μήκους F ψηφίων, με την υπέρθεση

των ψηφίων των υπογραφών των διακριτών όρων που περιέχονται. Το τελικό αρχείο υπογραφών προκύπτει από την ένωση των παραπάνω κειμένων.
Ο ψευδοκώδικας για την επεξεργασία ερωτήματος με signature files είναι ο εξής (οι όροι που περιέχονται στο ερώτημα περιέχονται στον πίνακα terms[]):

Πίνακας 2

```
FUNCTION create_signature(terms[])
BEGIN
        query_signature = or.signatures(terms);
        doc_id_set = [];
        FOR doc_id =  1 the total number of documents
        BEGIN
                doc_signature = get_signature_for_doc (doc_id, signature_file);
                bits_ok = check_corresponding_1_bits (query_signature,
doc_signature);
                IF bits_ok = TRUE THEN doc_id_set = doc_id_set + doc_id;
        END
        doc_id_set = remove_false_matches (terms, doc_id_set);
        return doc_id_set;
END
```

Αν το μήκος της υπογραφής του κειμένου είναι N ψηφία και η συλλογή αποτελείται από T συνολικά κείμενα, τότε το μέγεθος του signature file θα είναι (NxT) bits. Με βάση αυτό διαπιστώνεται ότι τα signature files δεν μπορούν να χρησιμοποιηθούν για την ευρετηριοποίηση του συνόλου του λεξιλογίου και των κειμένων μιας πραγματικής συλλογής, εξαιτίας των υπερβολικών αυξημένων απαιτήσεων σε αποθηκευτικό χώρο.

1.7 Μέτρα Ομοιότητας

Τα μέτρα ομοιότητας (*heuristic or similarity measures*) χρησιμοποιούνται σε όλες τις μηχανές αναζήτησης για να προσδιορίζουν την σχετικότητα του κάθε κειμένου με την ερώτηση αναζήτησης. Όσο μεγαλύτερο είναι το *similarity score* τόσο πλησιέστερο θεωρείται ότι είνια το κείμενο στην αναζήτηση του χρήστη. Οπότε τα αρχεία με την μεγαλύτερη ομοιότητα παρουσιάζονται ως οι προτεινόμενες απαντήσεις από το σύστημα προς τον χρήστη. Για τα μέτρα ομοιότητας υπάρχουν κάποιες αξίες, σύμβολα που ο συνδυασμός τους χρησιμοποιείται για το ranking. Κάποια από τα βασικότερα είναι τα ακόλουθα:

$\mathbf{f_{d,t}}$ → η συχνότητα εμφάνισης του όρου t στο κείμενο d
$\mathbf{f_{q,t}}$ → η συχνότητα εμφάνισης του όρου t στο ερώτημα αναζήτησης
$\mathbf{f_t}$ → ο αριθμός των κειμένων που περιέχουν παραπάνω από μία φορά τον όρο t
$\mathbf{F_t}$ → ο αριθμός των όρων t σε όλη την συλλογή κειμένων

N → ο αριθμός των αρχείων στην συλλογή
n → ο αριθμός των όρων που αρχειοθετούνται στην συλλογή
F → ο συνολικός αριθμός των όρων
f → ο αριθμός των indexes pointers
I → το τελικό μέγεθος του compressed inverted file

1.8. Βιβλιογραφία κεφαλαίου

1. **Inverted files for text search engines, Justin Zobel and Alistair Moffat**
2. **Managing gigabytes: Compressing and Indexing Documents and Images, Ian H.Witten, Alistair Moffat, Timothy C.Bell, Second Edition**
3. **Wikipedia, the free encyclopedia**
4. **www.google.com**

2. Inverted files

2.1. Ορισμός των Inverted files

Inverted file είναι ένα αρχείο που αναδιοργανώνει την δομή ενός υπάρχοντος αρχείου στοιχείων ('γραμμικό αρχείο') για να επιτρέψει την γρήγορη αναζήτηση. Η αναδιοργάνωση γίνεται με την βοήθεια κυρίως ενός B- tree όπου αποθηκεύονται οι όροι. Δύο βασικές παραλλαγές του inverted file υπάρχουν:

- **record level inverted index** (ή **inverted file index** ή απλά **inverted file**)
- **word level inverted index** (ή **full inverted index** ή **inverted list**)

Το inverted file index περιέχει μια λίστα από δείκτες στα αρχεία που εμπεριέχουν τους όρους, ενώ το full inverted index επιπλέον περιλαμβάνει και την θέση του κάθε όρου στο αρχείο.

Δύο ειναι τα βασικά στοιχεία ενός inverted file.

- Η δομή αναζητησης, η αποθήκευση όρων για κάθε διακριτό όρο που χαρακτηρίζεται από:
  - Τον αριθμό των κειμένων που περιέχουν τον όρο
  - Τον δείκτη που προσδιορίζει το κείμενο που εμπεριέχει τον όρο
- Ένα σύνολο από inverted lists όπου κάθε λίστα αποθηκεύει για τον προαναφερόμενο όρο
  - Το αναγνωριστικό των αρχείων που συμπεριέχουν τον όρο
  - Και τον αριθμό συχνότητας εμφάνισης του όρου στο κάθε κείμενο της λίστας

Για παράδειγμα ένα αρχείο που χρησιμοποιείται σε μια βιβλιογραφική βάση δεδομένων μπορεί να αποθηκεύσει τα ονόματα των συγγραφέων, τους τίτλους των βιβλίων, τα δημοσιευμένα άρθρα κτλ σε ένα inverted file. Όταν ένας όρος προσδιορίζεται σε ένα inverted file, τότε επιστρέφεται ο αριθμός αρχείων που περιέχουν τον όρο της αναζήτησης και ένα σύνολο αρχείων που αντιστοιχεί στα κριτήρια αναζήτησης δημιουργείται.

Ένα παράδειγμα που δείχνει πως χρησιμοποιείται ένα inverted file είναι το εξής:

Παρατίθονται κάποια αριθμημένα κείμενα

Πίνακας 3

Document	Text
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
5	Some like it in the pot,
6	Nine days old.

Έπειτα δημιουργούνται δύο indexes που περιέχουν το Lexicon (το σύνολο των όρων που εμφανίζονται στα κείμενα) και για κάθε όρο του lexicon δημιουργείται ένα inverted file. Βλέπουμε ότι ένα inverted file είναι απλά μία λίστα που δείχνει σε ποια κείμενα εμφανίζεται ο όρος, με ποια συχνότητα εμφάνισης. Ενώ το inverted list περιέχει επιπλέον την πληροφορά που δείχνει σε ποιο σημείο του κειμένου εντοπίζεται κάθε όρος.

Πίνακας 4

Number	Term	Documents
1	cold	⟨2;1,4⟩
2	days	⟨2;3,6⟩
3	hot	⟨2;1,4⟩
4	in	⟨2;2,5⟩
5	it	⟨2;4,5⟩
6	like	⟨2;4,5⟩
7	nine	⟨2;3,6⟩
8	old	⟨2;3,6⟩
9	pease	⟨2;1,2⟩
10	porridge	⟨2;1,2⟩
11	pot	⟨2;2,5⟩
12	some	⟨2;4,5⟩
13	the	⟨2;2,5⟩

Number	Term	Documents;Words
1	cold	⟨2;(1;6),(4;8)⟩
2	days	⟨2;(3;2),(6;2)⟩
3	hot	⟨2;(1;3),(4;4)⟩
4	in	⟨2;(2;3),(5;4)⟩
5	it	⟨2;(4;3,7),(5;3)⟩
6	like	⟨2;(4;2,6),(5;2)⟩
7	nine	⟨2;(3;1),(6;1)⟩
8	old	⟨2;(3;3),(6;3)⟩
9	pease	⟨2;(1;1,4),(2;1)⟩
10	porridge	⟨2;(1;2,5),(2;2)⟩
11	pot	⟨2;(2;5),(5;6)⟩
12	some	⟨2;(4;1,5),(5;1)⟩
13	the	⟨2;(2;4),(5;5)⟩

2.2. Index Construction

Η διαδικασία κατασκευής ενός index είναι γνωστή ως *inversion* ενός κειμένου. Το inversion αποτελεί το βασικότερο μέρος για να μπορέσει να χτιστεί μια βάση δεδομένων. Για να περιγράψουμε την διαδικασία αυτή θα χρησιμοποιήσουμε μια αρκετά προφανή μέθοδο με βάση το παράδειγμα του πίνακα 3. Αρχικά κατασκευάζεται ένας πίνακας συχνοτήτων για το συγκεκριμένο παράδειγμα.

Πίνακας 5

	Όροι												
	cold	*days*	*hot*	*in*	*it*	*like*	*nine*	*old*	*pease*	*porridge*	*pot*	*some*	*the*
1	1	-	1	-	-	-	-	-	2	2	-	-	-
2	-	-	-	1	-	-	-	-	1	1	1	-	1
3	-	1	-	-	-	-	1	1	-	-	-	-	-
4	1	-	1	-	2	2	-	-	-	-	-	2	-
5	-	-	-	1	1	1	-	-	-	-	1	1	1
6	-	1	-	-	-	-	1	1	-	-	-	-	-

Ο παραπάνω πίνακας εμφανίζει τον αριθμό συχνοτήτων του κάθε όρου της συλλογής του περιγράφεται στον πίνακα 3.

Κάθε γραμμή του κειμένου που περιγράφεται στον πίνακα 3 περιέχει κάποιους όρους που στον πίνακα 5 παρουσιάζεται κάθε όρος από το σύνολο σε ποια γραμμή και με πιοα συχνότητα εμφανίζεται. Με βάση αυτόν τον πίνακα μπορεί να δημιουργηθεί το index, αφού πρώτα όμως μετατεθεί σε έναν νέο πίνακα, ο οποίος έχει στις στήλες του τον αριθμό των κειμένων και στις γραμμές του κάθε όρο. Ο νέος πίνακας έχει ως εξής:

Πίνακας 6

Number	Term	Document					
		1	*2*	*3*	*4*	*5*	*6*
1	**Cold**	**1**	**-**	**-**	**1**	**-**	**-**
2	**Days**	**-**	**-**	**1**	**-**	**-**	**1**
3	**Hot**	**1**	**-**	**-**	**1**	**-**	**-**
4	**In**	**-**	**1**	**-**	**-**	**1**	**-**
5	**It**	**-**	**-**	**-**	**2**	**1**	**-**
6	**Like**	**-**	**-**	**-**	**2**	**1**	**-**
7	**Nine**	**-**	**-**	**1**	**-**	**-**	**1**
8	**Old**	**-**	**-**	**1**	**-**	**-**	**1**
9	**Pease**	**2**	**1**	**-**	**-**	**-**	**-**
10	**Porridge**	**2**	**1**	**-**	**-**	**-**	**-**
11	**Pot**	**-**	**1**	**-**	**-**	**1**	**-**
12	**Some**	**-**	**-**	**-**	**2**	**1**	**-**
13	**The**	**-**	**1**	**-**	**-**	**1**	**-**

Ο αλγόριθμος που μπορεί να χρησιμοποιηθεί για να μπορέσει να δημιουργηθεί ένα inverted file είναι ο ακόλουθος:
Κατασκευάζεται ο μετατιθέμμενος πίνακας συχνοτήτων, διαβάζεται το κάθε κείμενο στην σειρά με την βοήθεια της κάθε στήλης του πίνακα και εγγράφεται ο πίνακας στην μνήμη κατά σειρά. Ένα πρόβλημα που δυσκολεύει την inversion είναι το μέγεθος του πίνακα συχνοτήτων. Διότι έστω ότι έχουμε μία συλλογή που εμπεριέχει N διακριτούς όρους και M κείμενα. Εάν το μέγεθος του δείκτη είναι L τότε το μέγεθος της μνήμης θα είναι (L x N x M) bytes.

2.2.2. Αλγόριθμοι κατασκευής inverted files – index construction

Ένας index μπορεί να κατασκευαστεί επίσης και με τους εξής τρεις αλγορίθμους οι οποίοι θα παρουσιαστούν παρακάτω αναλυτικά.

1. Memory – based inversion
2. Sort – based inversion
3. Merge – based inversion

2.2.1. Memory – based inversion

Ο αλγόριθμος που χρησιμοποιείται στο memory – based inversion είναι ο εξής:

Πίνακας 7

Βήματα κατασκευής inverted index για μια συλλογή δεδομένων

1. /*Αρχικοποίηση*/
   Δημιουργία μια κενής δομής S
2. /* Δημιουργία συλλογής από τους όρους εμφάνισης*/
   Για κάθε κείμενο της συλλογής D_d. όπου $1 \subseteq d \subseteq N$
   A) Ανάγνωση του κειμένου D_d, εισαγωγή των όρων του κειμένου
   B) Για κάθε όρο t του κειμένου
      1) έχουμε f την συχνότητα εμφάνισης του όρου t
      2) αναζήτηση του όρου t στην S
      3) εάν δεν υπάρχει αυτός ο όρος, τον εισάγουμε
      4) επισύναψη ενός κόμβου (d, $f_{d,t}$) στην λίστα αντιστοιχώντας τον στον όρο t.

3. /*Δημιουργία του inverted file*/
   Για κάθε όρο $1 \subseteq t \subseteq n$
   A) δημιουργούμε ένα νέο κόμβο στο inverted file
   B) για κάθε (d, $f_{d,t}$) της λίστας που αντιστοιχεί στο όρο t, επισυνάπτουμε (d, $f_{d,t}$) σε αυτόν τον κόμβο του inverted file
   Γ) αν απαιτείται, συμπιέζουμε τον κόμβο του inverted file
   Δ) επισυνάπτουμε αυτόν τον νέο κόμβο στο inverted file

Ο παραπάνω αλγόριθμος περιγράφει το πως οι διακριτοί όροι μίας συλλογής εγγράφονται σε έναν πίνακα, συνοδευόμενοι από μια λίστα κόμβων όπου αποθηκεύονται οι γραμμές του κειμένου όπου εμφανίζεται ο όρος. Η δομή που δημιουργείται που αντιπροσωπεύει το inverted file του πίνακα 3 είναι :

Πίνακας 8

cold		→	1,1	→	4,1
days		→	3,1	→	6,1
hot		→	1,1	→	4,1
in		→	2,1	→	5,1
it		→	4,2	→	5,1
like		→	4,2	→	5,1
nine		→	3,1	→	6,1
old		→	3,1	→	6,1
pease		→	1,2	→	2,1
porridge		→	1,2	→	2,1
pot		→	2,1	→	5,1
some		→	4,2	→	5,1
the		→	2,1	→	5,1

Δομή αναζήτησης Συνδεδεμένες λίστες αποθήκευσης (d, $f_{d,t}$)

Ο συγκεκριμένος αλγόριθμος χωρίζεται σε δύο στάδια.

- Ανάλυση και ανάγνωση
- Εγγραφή.

Στο πρώτο στάδιο έχουμε την αναγνώριση των διακριτών όρων στο κάθε κείμενο και την ανανέωση της λίστας των όρων. Ενώ στο δεύτερο στάδιο έχουμε την εγγραφή του inverted file στο δίσκο για κάθε διακριτό όρο.

Ο συνολικός χρόνος που απαιτείται για την διαδικασία αυτή είναι ο εξής :

$T = Bt_r + Ft_p + I\ (t_d + t_r)$,

όπου Bt_r → ο συνολικός χρόνος για την μεταφορά όλων των δεδομένων

Ft_p → ο χρόνος που χρειάζεται για να αναλύσει, να ανατρέξει για όλους τους όρους

It_d → ο χρόνος για την δημιουργία του inverted file

It_r → ο χρόνος που χρειάζεται για την μεταφορά των δεδομένων για όλο το σύνολο του συμπιεσμένου inverted file

Το πλεονέκτημα αυτού του αλγορίθμου είναι ότι σχεδόν καθόλου μνήμη δεν σπαταλάτε σε σύγκριση με το τελικό μέγεθος του inverted file εφόσον υπάρχει

αμελητέος τεμαχισμός. Επιπλέον, εάν χρησιμοποιηθεί συμπίεση, τότε το index θα έχει συμπαγή τελική μορφή. Αυτή η τεχνική είναι εφαρμόσιμη όταν η διαθέσιμη κύρια μνήμη είναι περίπου 10% -20% μεγαλύτερη από τον συνδυασμό του index και των όρων που θα παραχθούν.
Έτσι για μια συλλογή μεγέθους 800.000.000 λέξεων χρειάζονται 400.000.000 λίστες εάν δεν υπάρχουν stop words και επαναλαμβανόμενοι όροι. Το τελικό μέγεθος μνήμης που απαιτείται είναι 4GB εφόσον χρειάζονται 10 bytes για κάθε λίστα.

2.2.2. Sort – based inversion

Το κύριο πρόβλημα με τις δύο μεθόδους που είδαμε μέχρι τώρα είναι η αρκετά μεγάλη μνήμη που απαιτείται και το γεγονός ότι χρησιμοποιούν μια ακολουθία πρόσβασης στοιχείων η οποία ειναι ουσιαστικά τυχαία. Η διαδοχική πρόσβαση είναι ο μόνος αποδοτικός τρόπος επεξεργασίας για αρχεία που καταλαμβάνουν μεγάλο μέγεθος μνήμης. Η λύση σε αυτά τα προβλήματα οδηγούν στον αλγόριθμο που θα δούμε τώρα είναι:

Πίνακας 9

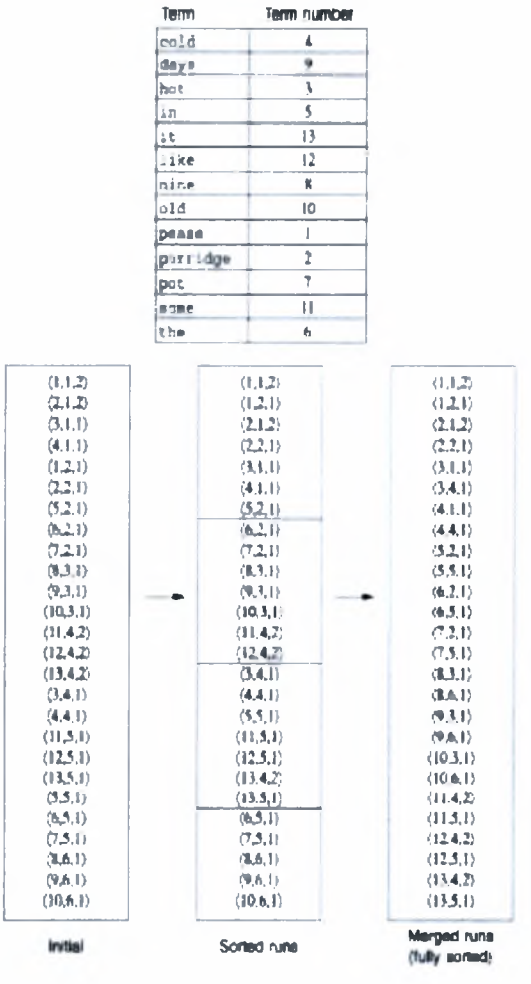
Βήματα για την παραγωγή ενός Inverted file για μια συλλογή κειμένων:

1) /*Αρχικοποίηση*/
   Δημιουργία μιας κενής δομής λεξικών S
   Δημιουργία ενός κενού προσωρινού αρχείου στο δίσκο
2) /*Κείμενο διαδικασίας και εγγραφή ενός προσωρινού αρχείου*/
   Για κάθε κείμενο D_d της συλλογής, όπου $1 \subseteq d \subseteq N$,
   a. Ανάγνωση D_d και ανάλυση του στους όρους δεικτών
   b. Για κάθε όρο δεικτών $t \in D_d$.
      i. Η συχνότητα εμφάνισης του όρου t στο κείμενο d είναι $f_{d,t}$
      ii. Γίνεται αναζήτηση του όρου t στην δομή λεξικών S
      iii. Εάν δεν υπάρχει ο όρος t στο S τότε τον εισάγουμε
      iv. Εισάγεται η εγγραφή <t, d, $f_{d,t}$> στο προσωρινό αρχείο, όπου το t αντιπροσωπεύεται στο S από τον αριθμό όρου του
3) /*Εσωτερική ταξινόμηση*/
   Έστω ότι k είναι ο αριθμός των εγγραφών που μπορούν να εισαχθούν στην μνήμη.
   a. Διαβάζουμε k εγγραφές από το προσωρινό αρχείο
   b. Ταξινομούνται οι εγγραφές σε φθίνουσα σειρά με βάση τον αριθμό εμφάνισης τους των όρων t, και για ίσες τιμές των όρων t, χρησιμοποιείται ο όρος d
   c. Εγγράφεται η ταξινόμηση στο προσωρινό αρχείο
   d. Επαναλαμβάνονται τα βήματα έως ότου δεν υπάρχουν άλλα στοχεία προς ταξινόμηση
4) /*Συγχώνευση*/
   Συγχωνεύονται ανα ζευγάρια στο προσωρινό αρχείο έως ότου μείνει ένα
5) /*Παραγωγή inverted file*/
   Για κάθε όρο $1 \subseteq t \subseteq n$,

a. Ξεκινάει μια νέα προσθήκη στο inverted file
b. Διαβάζονται τα <t, d, $f_{d,t}$> από το προσωρινό αρχείο και διαμορφώνεται η προσθήκη του inverted file για τον όρο t
c. Εάν απαιτείται, συμπιέζεται η προσθήκη του inverted file
d. Επισυνάπτεται αυτή η προσθήκη στο inverted file

Ο αλγόριθμος του sort – based inversion αρχικά διαβάζει τους όρους και τους περνάει σε ένα προσωρινό αρχείο. Στο προσωρινό αρχείο εγγράφεται η τριάδα <t, d, $f_{d,t}$> για κάθε συνδυασμό όρου t και κειμένου d. Το inverted file λαμβάνεται έπειτα από την ταξινόμηση αυτής της τριάδας ως προς τον όρο t. Έτσι συγκεντρώνονται όλες οι είσοδοι για κάθε όρο ώστε το inverted list για αυτόν τον όρο να μπορεί να δημιουργηθεί σκανάρωντας την ταξινομημένη λίστα διαδοχικά. Η ταξινόμηση ολοκληρώνεται από μια εξωτερική συγχώνευση. Κατά την διάρκεια της ταξινόμησης του προσωρινού αρχείου, γίνεται ανάγνωση των εγγραφών k. Ο αριθμός k συμβολίζει το μέγιστο αριθμό εγγραφών που μπορούν να αποθηκευτούν στην κύρια μνήμη. Αυτές οι εγγραφές έχουν όλες το ίδιο μέγεθος, περίπου 10 bytes, 4 bytes για τον κάθε όρο, 4 bytes για τον αριθμό του κειμένου, 2 bytes για να αποθηκευτεί η συχνότητα εμφάνισης του όρου στο αρχείο. Για παράδειγμα έαν η διαθέσιμη μνήμη είναι 1 Mbyte τότε ο αριθμός k θα είναι 100.000. Κάθε block που έχει μέχρι στιγμής αναγνωστεί, ταξινομείται με φθίνουσα σειρά. Το πρωτεύον κλειδί για την ταξινόμηση είναι ο όρος t και το δευτερεύον το d. Έπειτα αυτό το block εγγράφεται στο προσωρινό αρχείο. Στη επόμενη φάση, γίνεται συγχώνευση. Αυτή η συγχώνευση επιτυγχάνεται κυκλικώς, δηλαδή συγχωνεύεται το πρώτο με το δεύτερο, το τρίτο με το τέταρτο, κ.τ.λ. έως ότου απομείνει μόνο ένα. Σαν αποτέλεσμα εάν υπάρχουν R αρχικά περάσματα από το προσωρινό αρχείο, έπειτα από logR περάσματα θα απομείνει μόνο ένα. Το τελικό βήμα για αυτόν τον αλγόριθμο είναι η ανάγνωση του προσωρινού ταξινομημένου αρχείου, από όπου παράγεται το συμπιεσμένο inverted file εξόδου. Αφού εγγραφεί το inverted file, το προσωρινό αρχείο διαγράφεται.

Πίνακας 10



Στον πίνακα 10 φαίνεται το λεξικό και το προσωρινό αρχείο που θα προκύψει από το sort – based inversion του παραδείγματος που χρησιμοποιούμε. Αυτό που παρατηρούμε είναι ότι σε αυτόν τον πίνακα είναι διαφορετικό το inverted file που προκύπτει. Αυτό οφείλεται στο γεγονός οι αριθμοί των όρων ορίζονται όχι αλφαβητικά αλλά με βάση την σειρά που εμφανίζονται πρώτοι.
Ο χρόνος που απαιτείται για τις τέσσερις φάσεις αυτής της μεθόδου είναι:

$T = Bt_r + Ft_p + 10ft_r +$ (ανάγνωση και πέρασμα, εγγραφή αρχείου)
$20ft_r + R(1.2\ klogk)t_c$ (ταξινόμηση)
$[logR]\ (20ft_r + ft_c)$ (συγχώνευση)
$10ft_r + I(t_d + t_r)$ (εγγραφή συμπιεσμένου inverted file)

Ο μόνος πόρος που απέμεινε να εξετάσουμε είναι ο χώρος που απαιτείται στον δίσκο. Ο αλγόριθμος που περιγράψαμε απαιτεί δύο αντίγραφα των δεδομένων ανά πάσα στιγμή. Διότι όπως είδαμε στην περίπτωση της συγχώνευσης, υπάρχουν δύο αλγόριθμοι που τρέχουν, το κάθε ένα χρησιμοποιεί περίπου το μισό από το μέγεθος του αρχικού αρχείου. Εξαιτίας αυτού είναι αδύνατον τα δεδομένα από τα αποτελέσματα της συγχώνευσης να εγγράφονται στον ίδιο αρχείο εφόσον υπάρχουν δεδομένα ακόμα που πρέπει να παραχθούν. Αυτό πρακτικά σημαίνει ότι θα πρέπει να υπάρχουν δύο προσωρινά αρχεία. Οπότε θα χρειαστεί χώρος αρκετά μεγαλύτερος από το κείμενο που θέλουμε να αντιστρέψουμε, που είναι διπλάσιος από το μέγεθος

του αρχικού αρχείου. Αυτή η απαίτηση του αρκετά επιπλέον χώρου σημαίνει ότι παρότι ο συγκεκριμένος αλγόριθμος είναι ο βέλτιστος από αυτούς που έχουμε μέχρι τώρα εξετάσει, δεν είναι ο κατάλληλος για συλλογές που κατέχουν μέγαλο μέγεθος.

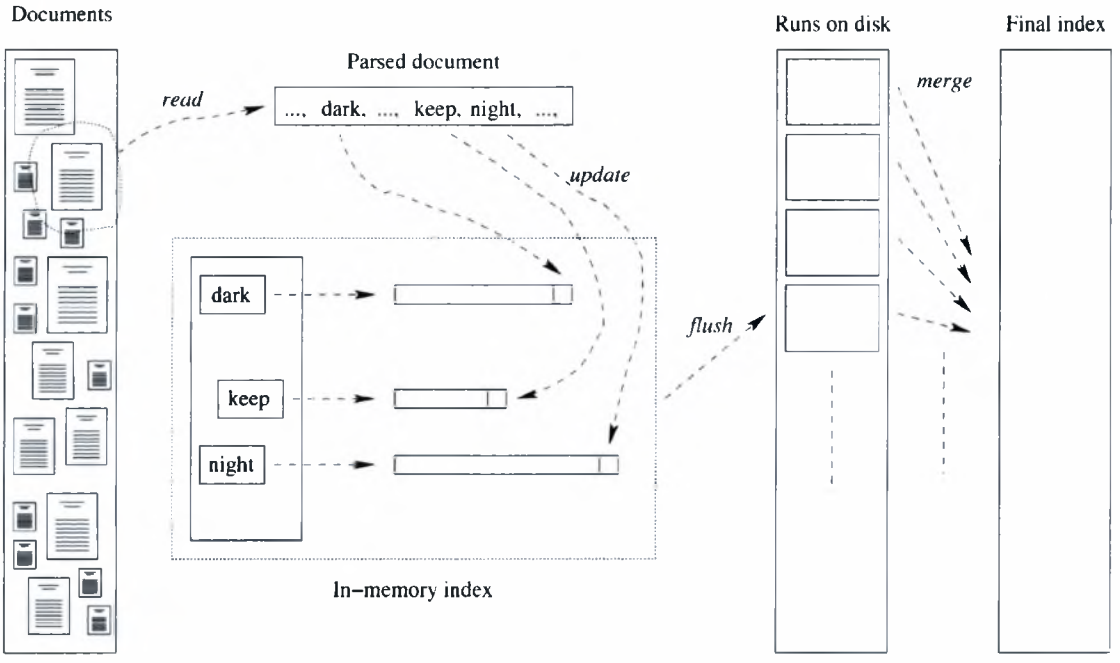
2.2.3.Merge – based inversion

Όσο οι απαιτήσεις σε χώρο αποθήκευσης και μέγεθος δεδομένων αυξάνονται, τόσο αυξάνενται δραματικά το κόστος διατήρησης λεξικού στην μνήμη. Τελικά, το αρχείο θα πρέπει να δημιουργηθεί από μικρότερα κομμάτια. Το κάθε ένα από αυτά θα δομηθεί με βάση έναν από τους προηγούμενος αλγόρθμους που εξετάσαμε.

Πίνακας 11

I. Μέχρι όλα τα δεδομένα να παραχθούν
   a. Αρχικοποίηση ενός αρχείου μνήμης, χρησιμοποιώντας μια δυναμική δομή και στατική κωδικοποίηση σχημάτων για τις αντεστραμμένες λίστες. Αποθήκευση των λιστών είτε σε δυναμική επαναταξινομημένη σειρά είτε σε συνδεδεμένες λίστες.
   b. Ανάγνωση δεδομένων και εισαγωγή <d, $f_{d,t}$> δεικτών σε αρχείο μνήμης, έως ότου ολόκληρη η διαθέσιμη μνήμη καταναλωθεί.
   c. Εκροή του προσωρινού αρχείου στον δίσκο, περιλαμβανομένου και του λεξικού.
II. Συγχώνευση των set των ξεχωριστών αρχείων σε ένα αρχείο, συμπίεση των αντεστραμμένων λιστών εάν απαιτείται

Στην τεχνική της merge – based inversion τα δεδομένα διαβάζονται και αντιστρέφονται στην μνήμη. Κάθε αντεστραμμένη λίστα χρειάζεται να αντιπροσωπηθεί σε μια δομή που μπορεί να διαθέσει όσο το δυνατόν περισσότερες πληροφορίες για τους όρους και να αναδιαμορφώσει δυναμικά τους πίνακες με βάση την καλύτερη επιλογή. Όταν γεμίζει η μνήμη, το αρχείο (συμπεριλαμβανομένου και το λεξικό) αποθηκεύεται στον δίσκο μαζί με τις αντεστραμμένες λίστες, ώστε να διευκολύνει την επόμενη συγχώνευση. Εφόσον δεν απαιτείται η αποθήκευση του *run* αλλά το αποτέλεσμα, δεν καταναλώνεται επιπλέον χώρος στην μνήμη. Όταν εγγραφεί το αποτέλεσμα του *run* διαγράφεται πλέον από την μνήμη για να ξεκινήσει το επόμενο *run* σε άδειο λεξικό.
Όταν παραχθούν όλα τα δεδομένα, τα *runs* συγχωνεύονται για το τελικό αρχείο. Εάν η διαθέσιμη μνήμη είναι λιγοστή, τότε το τελικό αρχείο μπορεί να εγγραφεί στον χώρο που καταλάμβαναν τα *runs* χωρίς πρόβλημα διότι το μέγεθος είναι λίγο μικρότερο από αυτό του *run*.
Η τεχνική merge – based inversion είναι πρακτική για συλλογές κάθε μεγέθους. Πιο συγκεκριμένα, λειτουργεί περισσότερο αποτελεσματικά όταν η διαθέσιμη μνήμη είναι έως 100MB. Πρόσθετα, οι τεχνικές συμπίεσης μπορούν να ελαττώσουν το κόστος κατασκευής ελαττώνοντας των αριθμό των *runs* που απαιτούνται.



Merge – built διαδικασία

2.3. Βιβλιογραφία κεφαλαίου

1. **Inverted files for text search engines, Justin Zobel and Alistair Moffat**
2. **Managing gigabytes: Compressing and Indexing Documents and Images, Ian H.Witten, Alistair Moffat, Timothy C.Bell, Second Edition**
3. **www.ir.iit.edu Information retrieval at the Illinois Institute of Technology**

3. Inverted file compression

Συμπίεση inverted file

Το μέγεθος ενός inverted file μπορεί να μειωθεί συμπιέζοντας το. Η συμπίεση ανεστραμμένων αρχείων είναι κρίσιμη για το αποτελεσματικό χειρισμό μεγάλου

όγκου πληροφορίας. Το κλειδί για την συμπίεση είναι η παρατήρηση ότι κάθε αντεστραμμένη λίστα δύναται να αποθηκευθεί ως μια ακολουθία ακεραίων αριθμών χωρίς να διαστραβλωθεί καθόλου το αποτέλεσμα. Ένα παράδειγμα είναι το εξής :

Τποθέτουμε ότι κάποιοι όροι εμφανίζονται σε 8 από τα κείμενα της συλλογής. Τα κείμενα αυτά είναι τα 3, 5, 20, 21, 23, 76, 77, 78. Ο κάθε ένας από αυτούς τους όρους περιγράφεται στην αντεστραμμένη λίστα με αυτή την λίστα

<8; 3, 5, 20, 21, 23, 76, 77, 78>

της οποίας η διεύθυνση περιέχεται στο λεξικό. Γενικεύοντας το παράδειγμα λέμε ότι η λίστα για τον όρο t αποθηκεύει τους αριθμούς των κειμένων f_i στα οποία εμφανίζεται ο όρος και η λίστα αυή έχει την μορφή

$< f_i; d_1, d_2, \ldots , df_i>$, όπου $d_k < d_{k+1}$.

Επειδή η λίστα των αριθμών των κειμένων μέσα σε κάθε αντεστραμμένη λίστα είναι με αύξουσα σειρά, και όλη η διαδικασία ξεκινάει από την αρχή της λίστας, η λίστα μπορεί να αποθηκευθεί ως μια αρχική θέση που ακολουθείται από μια λίστα d-gaps των $d_{k+1} - d_k$. Οπότε η λίστα του παραδείγματος αποθηκεύεται ως

<8; 3, 2, 15, 1, 2, 53, 1, 1>.

Δεν χάνεται καμία πληροφορία εφόσον οι αριθμοί των κειμένων που εμφανίζεται ο όρος μπορεί να ανακτηθεί από τον υπολογισμό της πρόσθεσης των d-gaps.
Και οι δύο μορφές απεικόνισης λιστών είναι ισοδύναμες. Και οι δύο μέθοδοι, εάν χρησιμοποιηθούν N κείμενα, θα χρειαστούν logN bits για κάθε δείκτη αποθήκευσης. Παρόλα αυτά η μέθοδος των d-gaps επιτρέπει πιο βελτιωμένη αντιπροσώπευση και είναι πιθανόν να κωδικοποιήσει τις αντεστραμμένες λίστες χρησιμοποιώντας κατά μέσο όρο λιγότερα από logN bits ανά δείκτη.
Αρκετές συγκεκριμένες μέθοδοι έχουν προταθεί για την περιγραφή διανομή πιθανότητας των d-gaps μεγεθών. Διαχωρίζονται σε δύο κατηγορίες :

i. global methods
ii. local methods.

Στις global methods κάθε λίστα συμπιέζεται χρησιμοποιώντας το ίδιο μοντέλο συμπίεσης και στις local methods κάθε μοντέλο συμπίεσης για κάθε όρο είναι ρυθμισμένο σύμφωνα με κάποιες παραμέτρους, που συνήθως η παράμετροι αυτοί είναι η συχνότητα εμφάνισης του όρου. Οι global methods χωρίζονται στις parameterized και nonparameterized. Οι local methods είναι πάντοτε parameterized.

Method	Reference
Global methods	
Nonparameterized	
Unary	
Binary	
γ	Elias (1975); Bentley and Yao (1976)
δ	Elias (1975); Bentley and Yao (1976)
Parameterized	
Bernoulli	Golomb (1966); Gallager and Van Voorhis (1975)
Observed frequency	
Local methods	
Bernoulli	Witten, Bell, and Nevill (1992); Bookstein, Klein, and Raita (1992)
Skewed Bernoulli	Teuhola (1978); Moffat and Zobel (1992)
Hyperbolic	Schuegraf (1976)
Observed frequency	
Batched frequency	Moffat and Zobel (1992)
Interpolative	Moffat and Stuiver (1996)

Αλγόριθμοι συμπίεσης inverted file

Έχει αποδειχθεί ότι η μείωση του χώρου που καταλαμβάνει ένα inverted file στο δίσκο έχει ως αποτέλεσμα την βελτίωση της αποδοτικότητας του στην επεξεργασία των ερωτημάτων, καθώς λιγότερα δεδομένα μεταφέρονται από τον δίσκο στη κυρία μνήμη. Η συμπίεση τους οφείλεται στην μείωση του μήκους της αναπαράστασης των ακεραίων που αποθηκεύονται στις αντεστραμμένες λίστες.

Οι περισσότερες από τις μεθόδους συμπίεσης βασίζονται στον Υπολογισμό της Μεταβολής των Δεικτών. Δηλαδή αντί να αποθηκεύοται οι απόλυτες τιμές των κειμένων που περιέχουν τους όρους, αποθηκεύονται οι διαφορές τους. Σε γενικές γραμμές αυτό έχει ως αποτέλεσμα την αποθήκευση μικρότερων και πιο συχνά εμφανιζόμενων αριθμών. Οι αρχικές λίστες μπορούν εύκολα να επανακτηθούν απλά προσθέτοντας τις διαφορές.

Η κωδικοποίηση είναι μια αντιστρεπτή διαδικασία υπολογισμού μιας ακολουθίας ψηφίων bits για μια ακολουθία στοιχείων. Στη συμπίεση το ζητούμενο είναι η κωδικοποίηση μιας ομάδας στοιχείων με το μικρότερο δυνατό πλήθος ψηφίων. Έχουν αναπτυχθεί διάφορες γενικές μέθοδοι συμπίεσης δεδομένων. Τα αποτελέσματα τους εξαρτώνται από το πόσο ικανοποιητικά μπορούν να υλοποιήσουν μοντέλα που περιγράφουν ή προβλέπουν την κατανομή των δεδομένων που πρόκειται να κωδικοποιηθούν. Τέτοιες μέθοδοι είναι η κωδικοποίηση Huffman και η αριθμητική (*arithmetic*) κωδικοποίηση.

Οι κώδικες συμπίεσης διαιρούνται σε δύο μεγάλες οικογένειες: στους κώδικες συμπίεσης σε επίπεδο ψηφίου (bit) και στους κώδικες συμπίεσης σε επίπεδο ψηφιολέξης.

Οι κώδικες συμπίεσης σε επίπεδο ψηφίου παράγουν τα μικρότερα δυνατά ευρετήρια και διακρίνονται σε παραμετροποιημένους και μη – παραμετροποιημένους που αξιοποιούν τον υπολογισμό της μεταβολής των δεικτών, αλλά και σε κώδικες οι οποίοι χρησιμοποιούν διαφορετικές τεχνικές. Οι πρώτοι εξαρτώνται από παραμέτρους οι οποίες αντικατοπτρίζουν την στατιστική κατανομή των ακεραίων μέσα στο inverted file. Οι δεύτεροι δεν εξαρτώνται από παραμέτρους και δίνουν σταθερές αναπαραστάσεις για τους ακεραίους, οι οποίες εφαρμόζονται αυτούσια σε κάθε συλλογή κειμένων. Έχουν μειωμένη ικανότητα συμπίεσης σε σχέση με τους

παραμετροποιημένους, αλλά είναι πιο κατάλληλοι για εφαρμογή σε δυναμικές συλλογές κειμένων. Οι υπόλοιποι κώδικες δεν έχουν ως βάση τον υπολογισμό μεταβολής δεικτών και αξιοποιούν το περιεχόμενο για να επιτύχουν μεγαλύτερη συμπίεση.
Οι κώδικες συμπίεσης σε επίπεδο ψηφιολέξης παράγουν μεγαλύτερα ευρετήρια και όταν το επιπρόσθετο κόστος μεταφοράς των δεδομένων από το δίσκο είναι αμελητέο, επιταχύνουν την διαδικασία αναζήτησης. Στη περίπτωση που ολόκληρο το ευρετήριο παραμένει στην κυρία μνήμη, η ταχύτητα αναζήτησης διπλασιάζεται, ενώ η ανάκτηση των δεδομένων γίνεται αποκλειστικά από το δίσκο. Τότε μπορεί να εφαρμοστεί συνδυασμός κωδίκων σε επίπεδο ψηφίου και ψηφιολέξης για την βελτιστοποίηση της αποδοτικότητας στη επεξεργασία των ερωτημάτων.

Μοντέλα κατανομής πιθανοτήτων

Για την επίτευξη μεγάλων ποσοστών συμπίεσης είναι στοιχειώδες συστατικό ένα μοντέλο που μπορεί να περιγράψει αναλυτικά ή να προβλέψει με ακρίβεια τις πιθανότητες των ως προς κωδικοποίηση στοιχείων και επιπλέον μια μέθοδος κωδικοποίησης που μπορεί να αποδώσει πιστά αυτές τις πιθανότητες. Επομένως η λειτουργία ενός μοντέλου κατανομής είναι η περιγραφή ή πρόβλεψη της εμφάνισης κάθε στοιχείου. Για τον σκοπό αυτό υπάρχουν τα στατικά μοντέλα, τα ημιστατικά και τα μοντέλα προσαρμογής.

Στατικά μοντέλα

Στατικά μοντέλα είναι τα μοντέλα που εφαρμόζονται σε οποιαδήποτε ακολουθία στοιχείων χρησιμοποιώντας την ίδια κατανομή πιθανοτήτων. Δεν ενδιαφέρονται για τα στατιστικά στοιχεία που σχετίζονται με την ακολουθία. Ένα παράδειγμα είναι το μοντέλο gamma του Elias που θα περιγραφθεί παρακάτω. Η κατανομή που εφαρμόζεται είναι η $Pr[x] = 1/2x^2$ δηλαδή ότι η πιθανότητα εμφάνισης διαφοράς με τιμή x είναι $1/2x^2$ για όλες τις αντεστραμμένες λίστες.

Ημιστατικά μοντέλα

Τα ημιστατικά μοντέλα χρησιμοποιούν στατιστικά στοιχεία της ακολουθίας των στοιχείων που πρόκειται να κωδικοποιηθεί. Για να είναι αυτό εφικτό θα πρέπει να γνωρίζουμε εκ των προτέρων τη ακολουθία και η οποία δεν πρόκειται να τροποποιηθεί. Τέτοιο μοντέλο είναι το global μοντέλο Bernoulli. Σύμφωνα με αυτό το μοντέλο η πιθανότητα εμφάνισης διαφοράς με τιμή x είναι

$$Pr[x] = (1 - p)^{x-1}p \text{ όπου } p = f/(N*n).$$

N είναι το πλήθος των κειμένων της συλλογής, n είναι το πλήθος των διακριτών όρων της συλλογής και f είναι το συνολικό πλήθος των δεικτών του inverted file.

Μοντέλα προσαρμογής

Ένα μοντέλο προσαρμογής αρχίζει με μια επιλεγμένη κατανομή για τα στοιχεία που θα κωδικοποιηθούν και τα οποία μεταβάλλει όταν λαμβάνονται τα πραγματικά στοιχεία προς κωδικοποίηση. Ένα παράδειγμα μοντέλου προσαρμογής είναι το Μηδενικό Μοντέλο Προσαρμογής (Adaptive Zero Model) που χρησιμοποιεί ως επιλεγμένη κατανομή το συνολικό τμήμα των στοιχείων που έχουν ήδη κωδικοποιηθεί. Επειδή πριν ξεκινήσει η κωδικοποίηση, η κατανομή των στοιχείων δίνει σε κάθε στοιχείο μηδενική πιθανότητα εμφάνισης, προκύπτει το πρόβλημα της μηδενικής συχνότητας. Η λύση στο πρόβλημα αυτό είναι η υπόθεση ότι όλα τα στοιχεία έχουν εμφανιστεί από μία φορά.

Τεχνικές κωδικοποίησης

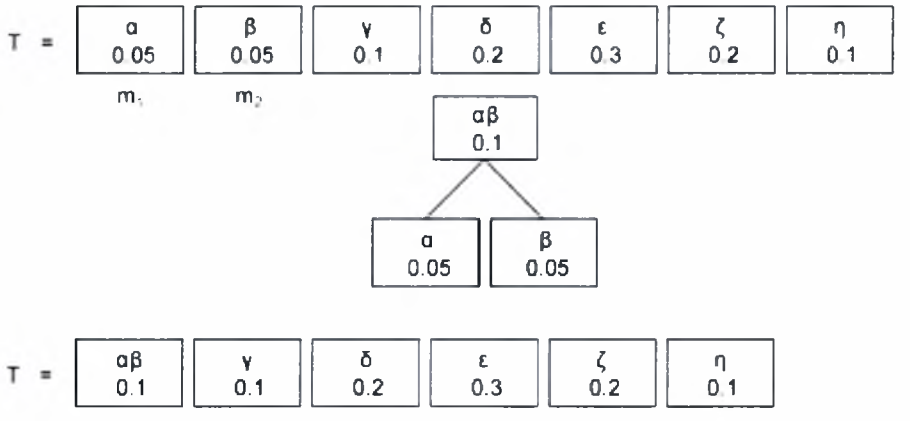
Κωδικοποίηση Huffman

Η κωδικοποίηση Huffman είναι μία από τις δύο γενικές τεχνικές κωδικοποίησης που μπορούν να υλοποιήσουν διάφορα μοντέλα κατανομής των προς κωδικοποίηση στοιχείων. Βασίζεται στη δημιουργία ενός δέντρου Huffman, με το οποίο υπολογίζεται ένας μοναδικός κώδικας για κάθε διακριτό στοιχείο. Το δένδρο κατασκευάζεται από τα φύλλα προς την κορυφή ως εξής :
Αρχικά δημιουργείται ένα σύνολο αποτελούμενο από κόμβους. Κάθε κόμβος περιέχει ένα διακριτό στοιχείο και την πιθανότητα εμφάνισης του. Στη συνέχεια εντοπίζονται οι κόμβοι με τις μικρότερες πιθανότητες και τοποθετούνται κάτω από τον ίδιο, νέο κόμβο πατέρα. Ο νέος αυτός κόμβος περιέχει το άθροισμα των πιθανοτήτων των κόμβων παιδιών του και και τα αντικαθιστά στο αρχικό σύνολο των κόμβων. Η διαδικασία της επιλογής των κόμβων με τις μικρότερες πιθανότητες, της δημιουργίας του κόμβου πατέρα τους και και της αντικατάστασης τους συνεχίζεται μέχρι να απομείνει στο τέλος ένος κόμβος, που γίνεται η ρίζα του δένδρου. Ο κώδικας για κάθε στοιχείο του αρχικού συνόλου προκύπτει ακολουθώντας την μοναδική διαδρομή από την ρίζα του δένδρου έως τον κόμβο φύλλο του στοιχείου. Κάθε αριστερή μετακίνηση σε κόμβο παράγει ένα ψηφίο '0' ενώ κάθε δεξιά μετακίνηση παράγει ένα ψηφίο '1'. Για την κωδικοποίηση οποιασδήποτε ακολουθίας στοιχείων χρησιμοποιούνται οι υπολογισμένοι κώδικες Huffman, οι οποίοι έχουν την ιδιότητα ότι κανείς δεν αποτελεί πρόθεμα κάποιου.
Κατά την αποκωδικοποίηση μιας ακολουθίας ψηφίων είναι απαραίτητο να χρησιμοποιηθεί το δένδρο Huffman όπου βασίστηκε η κωδικοποίηση. Ξεκινώντας από την ρίζα του δένδρου κάθε ψηφίο που λαμβάνεται από την ακολουθία υποδεικνύει το είδος της μετακίνησης μέσα στο δένδρο. Τα ψηφία με τιμή '0' επιβάλλουν αριστερές μετακινήσεις ενώ τα ψηφία με τιμή '1' δεξιές μετακινήσεις. Όταν συναντάμε κόμβο φύλλο, επιστρέφεται το στοιχείο του κόμβου και και η διαδικασία ξεκινάει πάλι από την ρίζα του δένδρου.
Ακολουθεί ο ψευδοκώδικας για την δημιουργία του δένδρου Huffman.

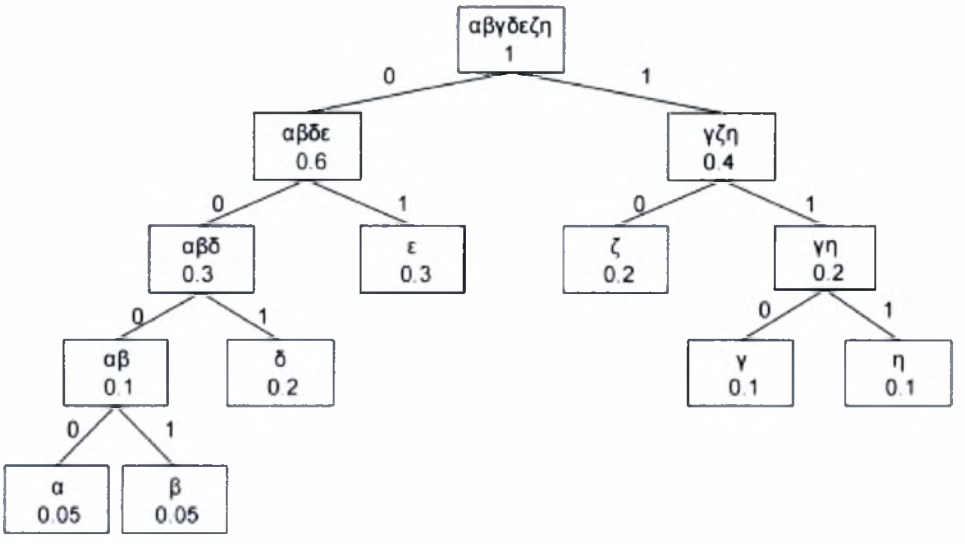
Πίνακας 12

```
FUNCTION Huffman_Tree(object[], Pr[], n)
BEGIN
  T = Set of nodes containing each distinct object
      and its probability;
  FOR k = 1 TO n - 1
  BEGIN
    m1, m2 = two nodes of least probability ∈ T;
    Create new node m_p whose probability is the sum
      of that of m1 and m2;
    Make m_p parent node for m1 and m2;
    Replace m1 and m2 with m_p in T;
  END
  The last node ∈ T becomes root for the Huffman Tree;
END
```

Ένα παράδειγμα όπου φαίνεται ο τρόπος λειτουργίας των διαδικασιών που περιγράψαμε παραπάνω είναι το ακόλουθο. Έστω ότι το σύνολο των στοιχείων που πρόκειται να κωδικοποιηθούν είναι οι χαρακτήρες του λεξικολογίου {α, β, γ, δ, ε, ζ, η} με πιθανότητες εμφάνισης {0.05, 0.05, 0.1, 0.2, 0.3, 0.2, 0.1}. Θα εξετάσουμε την κωδικοποίηση και αποκωδικοποίηση του αλφαριθμητικού 'βγγβ'.
Πριν ξεκινήσει η διαδικασία κωδικοποίησης οποιουδήποτε χαρακτήρα του αλφαριθμητικού πρέπει να κατασκευαστεί το δένδρο Huffman.



Αρχικά ορίζεται το σύνολο Τ με τους 7 κόμβους. Στη συνέχεια λαμβάνονται οι κόμβοι με τις μικρότερες πιθανότητες εμφάνισης που είναι 2 τα m1 και m2, δηλαδή οι κόμβοι α και β. Δημιουργείται ο νέος κόμβος αβ και ως πιθανότητα του νέου κόμβου ορίζεται το άθροισμα των πιθανοτήτων των α και β. Το σύνολο Τ μεταβάλλεται και πλέον έχει 6 κόμβους. Τα α και β έχουν αντικατασταθεί από τον κόμβο πατέρα αβ. Η διαδικασία επαναλαμβάνεται έως ότου απομείνει ένας κόμβος στο σύνολο Τ ο οποίος μετατρέπεται σε ρίζα του δένδρου. Το τελικό δένδρο είναι :



Το παραγόμενο δένδρο χρησιμοποιείται τόσο στην κωδικοποίηση χαρακτήρων όσο και στην αποκωδικοποίηση παραστάσεων. Εύκολα προκύπτει ότι η αναπαράσταση του αλφαριθμητικού 'βγγβ'είναι 00011101100001

Πίνακας 13

Χαρακτήρας	Πιθανότητα	Κώδικας	Χαρακτήρας	Πιθανότητα	Κώδικας
α	0.05	0000	ε	0.3	01
β	0.05	0001	ζ	0.2	10
γ	0.1	110	η	0.1	111
δ	0.2	001			

Arithmetic (αριθμητική) κωδικοποίηση

Η αριθμητική κωδικοποίηση είναι μια τεχνική η οποία μπορεί να υλοποιήσει πολυσύνθετα μοντέλα κατανομής των στοιχείων που κωδικοποιούνται ώστε να επιτύχει μεγαλύτερα ποσοστά συμπίεσης, καλύτερα από αυτά που επιτυγχάνονται μέσω Huffman. Αυτό συμβαίνει διότι με την βοήθεια της αριθμητικής κωδικοποίησης είνα εφικτή η απόδοση κωδίκων στα στοιχεία με μέσο μήκος μικρότερο του ενός ψηφίου. Το μειονέκτημα της είναι οι επιδόσεις στην ταχύτητα της κωδικοποίησης και αποκωδικοποίησης. Επιπλέον είναι δύσκολο να ξεκινήσει η διαδικασία της αποκωδικοποίησης από το μέσο της αναπαράστασης για να ληφθεί μέρος των στοιχείων που έχουν κωδικοποιηθεί. Παρόλο που αποφεύγεται η χρήση της αριθμητικής κωδικοποίησης πρέπει να την αναφέρουμε διότι χρησιμοποιείται σε μοντέλα όπως η κατανομή υπερβολής, διότι εκεί δεν μπορεί να χρησιμοποιηθεί ο Huffman.

Στην αριθμητική κωδικοποίηση αποθηκεύονται δυο μεταβλητές, η low και η high, οι οποίες θα ορίσουν το διάστημα [0, ...1]. Οι αρχικές τιμές αυτών των μεταβλητών είναι low = 0 και high = 1. Το διάστημα μεταξύ τους διαιρείται ανάλογα με το μοντέλο κατανομής πιθανοτήτων που θα εφαρμοστεί. Κάθε υποδιάστημα που προκύπτει, το οποίο υπολογίζεται από τις δύο αθροιστικές πιθανότητες, αντιστοιχεί σε ένα διακριτό στοιχείο. Σε κάθε βήμα της αριθμητικής κωδικοποίησης το διάστημα που ορίζεται από τις μεταβλητές εξισώνεται με το υποδιάστημα που αντιστοιχεί στο

υπό κωδικοποίηση στοιχείο. Η διαδικασία της διαίρεσης και της εξίσωσης επαναλαμβάνεται έως να εξαντληθούν τα στοιχεία. Η αναπαράσταση που θα χρησιμοποιηθεί τελικά για να τα κωδικοποιήσει είναι το τμήμα μετά την υποδιαστολή, του δυαδικού αριθμού με το ελάχιστο μήκος που εξασφαλίζει τιμή στο πεδίο [low … high].
Κατά την αποκωδικοποίηση η διαδικασία που εφαρμόζεται είνα παρόμοια. Οι αρχικές τιμές των low και high είναι αντίστοιχα 0 και 1. Η δεκαδική τιμή του τελικού αριθμού της διαδικασίας της κωδικοποίησης (*value*) υπολογίζεται με την βοήθεια της δοσμένης αναπαράστασης. Το διάστημα μεταξύ των low και high διαιρείται ανάλογα με το ίδιο μοντέλο κατανομής των πιθανοτήτων . Τα σημεία διαίρεσης καθορίζονται από τις αθροιστικές πιθανότητες των στοιχείων. Κάθε υποδιάστημα που προκύπτει αντιστοιχεί σε ένα διακριτό στοιχείο. Σε κάθε βήμα της αποκωδικοποίησης το διάστημα που ορίζεται από τις μεταβλητές low και high εξισώνεται με το υποδιάστημα που περιέχει την τιμή value και επιστρέφεται ο χαρακτήρας που αντιστοιχεί στο διάστημα αυτό. Η διαδικασία της διαίρεσης και της εξίσωσης επαναλαμβάνεται μέχρι να αποκωδικοποιηθούν όλα τα στοιχεία.
Οι ψευδοκώδικες της κωδικοποίησης και αποκωδικοποίησης παρουσιάζονται παρακάτω. Η διαδικασία στον ψευδοκώδικα recalculate έχει σημασία μόνο όταν υλοποιούνται μοντέλα προσαρμογής.

Πίνακας 14

```
FUNCTION Arithmetic_Encode(word[], alphabet[], Pr[], n)
BEGIN
  low = 0; high = 1; initialize(Pr);
  FOR k = 1 TO n
  BEGIN
    // s stores the position of symbol word[k] in the alphabet
    s = find_pos(word[k], alphabet);
    low_bound = Σ_{i=1}^{s-1} Pr[i]; high_bound = Σ_{i=1}^{s} Pr[i];
    range = high − low;
    high = low + range × high_bound; low = low + range × low_bound;
    recalculate(Pr, word[k]);
  END
  bitstr = find the smallest binary representation with
       value ∈ [low … high], omit '0.';
  return bitstr;
END
```

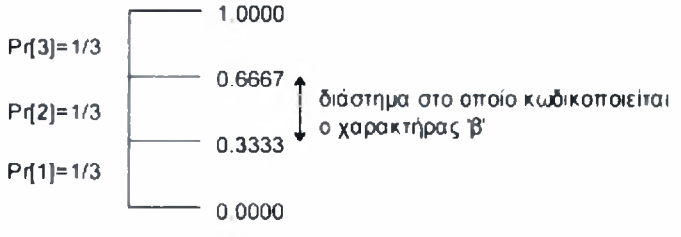
Αριθμητική κωδικοποίηση. Κωδικοποιείται η ακολουθία συμβόλων word[]. Τα n σύμβολα της ακολουθίας ανήκουν στο αλφάβητο alphabet ενώ ο πίνακας Pr αποθηκεύει τις πιθανότητες εμφάνισης των στοιχείων του αλφαβήτου.

Πίνακας 15

```
FUNCTION Arithmetic_Decode(bitstr, alphabet[], Pr[], n)
BEGIN
  value = calculate('0.' + bitstr);
  low = 0; high = 1; initialize(Pr);
  FOR k = 1 TO n
  BEGIN
    Find s such that
      Σ_{i=1}^{s-1} Pr[i] ≤ (value − low)/(high − low) < Σ_{i=1}^{s} Pr[i];
    word[k] = alphabet[s];
    low_bound = Σ_{i=1}^{s-1} Pr[i]; high_bound = Σ_{i=1}^{s} Pr[i];
    range = high − low;
    high = low + range × high_bound; low = low + range × low_bound;
    recalculate(Pr, word[k]);
  END
  return word;
END
```

Αριθμητική κωδικοποίηση. Αποκωδικοποιείται η ακολουθία ψηφίων bitstr[]. Τα n κωδικοποιημένα σύμβολα ανήκουν στο αλφάβητο alphabet ενώ ο πίνακας Pr αποθηκεύει τις πιθανότητες εμφάνισης των στοιχείων του αλφαβήτου.

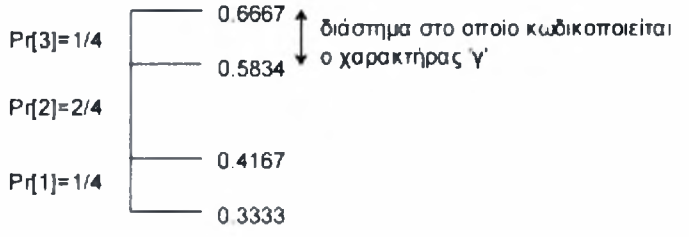
Ένα παράδειγμα όπου μπορεί να γίνει πλήρως κατανοητός ο τρόπος λειτουργίας των διαδικασιών που περιγράφονται παραπάνω είναι ο ακόλουθος: Έστω ότι το σύνολο των στοιχείων που πρόκειται να κωδικοποιηθούν είναι οι χαρακτηρες του λεξικολογίου {α, β, γ}. Θα εξετάσουμε την κωδικοποίηση και αποκωδικοποίηση του αλφαριθμητικού 'βγγβ'. Το μοντέλο κατανομής χαρακτήρων που θα εφαρμοστεί είναι το μηδενικό μοντέλο προσαρμογής, όπου η πιθανότητα εμφάνισης του χαρακτήρα θα μεταβάλλεται κάθε φορά που θα λαμβάνεται ο επόμενος χαρακτήρας του αλφαριθμητικού για κωδικοποίηση. Πριν από την κωδικοποίηση του πρώτου χαρακτήρα, θα ορίσουμε την πιθανότητα εμφάνισης σε όλους τους χαρακτήρες ως 1/3 αντί για μηδέν.
Στο πρώτο βήμα της κωδικοποίησης οι τιμές των low και high είναι '0' και '1' αντίστοιχα. Το διάστημα μεταξύ τους διαιρείται ως εξής :



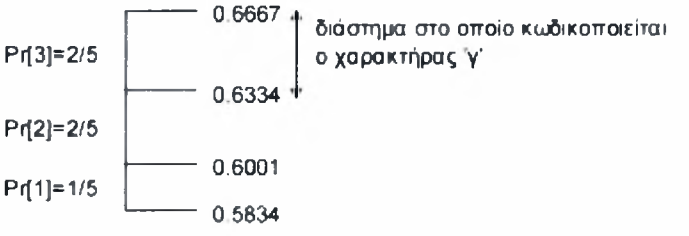
Το βήμα της κωδικοποίησης για τον πρώτο χαρακτήρα

Σαν αποτέλεσμα οι αθροιστικές πιθανότητες είναι 0.000, 0.333, 0.667, 1.000. Οπότε ο χαρακτήρας 'α' θα κωδικοποιηθεί στο διάστημα [0.000...0.333], ο 'β' στο [0.333...0.667] κ.τ.λ. Ο πρώτος χαρακτήρας που λαμβάνεται για κωδικοποίηση είναι ο 'β', έτσι οι τιμές των low και high μεταβάλλονται σε 0.333 και 0.667. Επειδή το μοντέλο που θα εφαρμοστεί είναι το μηδενικό μοντέλο προσαρμογής, χρησιμοποιείται η συνάρτηση recalculate για την μεταβολή των πιθανοτήτων των

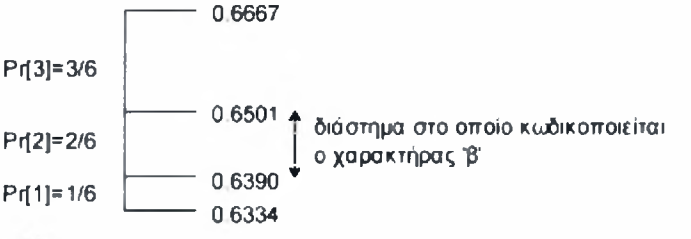
χαρακτήρων. Επειδή ο χαρακτήρας που χρησιμοποιήθηκε είναι ο 'β', αυξάνεται η συχνότητα εμφάνισης του κατά 1 και οι νέες πιθανότητες των α, β, γ είναι Pr[1] = ¼, Pr[2] = 2/4 και Pr[3] = ¼. Αφού εφαρμοστούν και τα υπόλοιπα βήματα θα έχουν στο τέλος low = 0.6390 και high = 0.6501, οπότε ψάχνουμε την τιμή value σε αυτό το διάστημα με το μικρότερο δυαδικό μήκος. Η τιμή είναι η 0.6396484375 που έχει ως δυαδική αναπαράσταση το 0.1010001111. Η αναπαράσταση που χρησιμοποιείται για να κωδικοποιήσει το 'βγγβ' είναι αυτή που ακολουθεί της υποδιαστολής.
Ακολουθούν σχηματικά τα άλλα τρία βήματα της κωδικοποίησης.



Το βήμα της κωδικοποίησης για τον δεύτερο χαρακτήρα



Το βήμα της κωδικοποίησης για τον τρίτο χαρακτήρα



Το βήμα της κωδικοποίησης για τον τέταρτο χαρακτήρα

Η διαδικασία αποκωδικοποίησης αρχίζει με την αναπαράσταση 1010001111 από όπου προκύπτει η τιμή value 0.6396484375. Ακολουθούν τα βήματα τα οποία είναι ανάλογα με αυτά της κωδικοποίησης. Εξετάζουμε σε ποιο υποδιάστημα ανήκει η τιμή value και αναλόγως βλέπουμε ποιος χαρακτήρας κωδικοποιήθηκε. Οπότε με αυτό το διάστημα συνεχίζεται η αποκωδικοποίηση. Τα βήματα είναι παρόμοια με της κωδικοποίησης.

3.5. Κώδικες συμπίεσης σε επίπεδο ψηφίου

Οι περισότεροι κώδικες συμπίεσης επιπέδου ψηφίου βασίζονται στον υπολογισμό της μεταβολής των δεικτών και σε μοντέλα που προσπαθούν να περιγράψουν την στατική κατανομή. Ανάλογα με το εάν οι κώδικες και τα μοντέλα τους χρησιμοποιούν αποτελέσματα στατιστικής επεξεργασίας της ευρετηριοποιημένης συλλογής κειμένων ή δεν κάνουν χρήση καμίας παραμέτρου, χωρίζονται στους παραμετροποιημένους και μη παραμετροποιημένους αντίστοιχα.
Οι παραμετροποιημένοι κώδικες είναι ικανότεροι στην συμπίεση και ταχύτεροι κατά την αποκωδικοποίηση, αλλά είναι προβληματικοί στην περίπτωση δυναμικών συλλογών κειμένων, διότι έχουμε συχνές διαδικασίες εισαγωγής, διαγραφής και μεταβολής κειμένων. Αυτό συμβαίνει διότι απαιτείται αρκετά συχνή ανακατασκευή των inverted files, η οποία απαιτεί τον επαναπροσδιορισμό των παραμέτρων και την ανανέωση της κωδικοποίησης των λιστών αυτών.
Οι μη παραμετροποιημένοι κώδικες έχουν μικρότερο βαθμό συμπίεσης αλλά είναι πιο κατάλληλοι για δυναμικές συλλογές κειμένων διότι μπορούν να εφαρμοστούν αποκλειστικά στο τμήμα κάποιας λίστας, η οποία επηρρεάζεται από πιθανές αλλαγές της συλλογής. Οι αναπαραστάσεις των ακεραίων που ανήκουν στα υπόλοιπα τμήματα της λίστα απλά αντιγράφονται για να επιτύχουμε πιο γρήγορα την ολοκλήρωση της νέας κωδικοποιημένης λίστας.
Πρέπει να σημειωθεί ότι υπάρχουν κώδικες οι οποίοι ακολουθούν διαφορετικό μονοπάτι όσον αφορά τον υπολογισμό των διαφορών και τα μοντέλα κατανομής, οι οποίοι όμως εξακολουθούν να έχουν τα ίδια μειονεκτήματα με τους παραμετροποιημένους που βασίζονται στον υπολογισμό της μεταβολής των δεικτών. Ένας τέτοιος κώδικας είναι αυτός του κώδικα παρεμβολής που επιτυγχάνει τα μεγαλύτερα ποσοστά συμπίεσης από όλους τους γνωστούς κώδικες συμπίεσης.

3.5.1. Μη παραμετροποιημένοι

3.5.1.1. Μοναδιαίος κώδικας

Οι απλούστεροι global κώδικες αντιπροσωπεύονται σταθερά από θετικούς ακέραιους. Όπως προείπαμε εάν υπάρχουν N κείμενα σε μια συλλογή, τότε απαιτούνται logN bits για κάθε δείκτη. Ένα nonparameterized model είναι το unary (μοναδιαίο). Σε αυτόν τον κώδικα ένας ακέραιος x $\geq$1 κωδικοποιείται ως x-1 bits με τιμή '1' ακολουθούμενο από ένα μηδενικό bit, για παράδειγμα το 3 γράφεται ως 110. Μια αντεστραμμένη λίστα που χρησιμοποιεί το unary θα χρειαστεί d_{fi} bits, εφόσον ο κώδικας για τα κενά των x απαιτεί x bits και σε κάθε αντεστραμμένη λίστα το σύνολο των κενών μεγεθών είναι το d_{fi} της τελευταίας εμφάνισης της συγκεκριμένης λέξης. Σε γενικές γραμμές, ένα αντεστραμμένο αρχείο κωδικοποιημένο με unary ενδέχεται κατά συνέπεια να καταναλώσει N*n bits και η ποσότητα θα είναι τελικά πολύ μεγάλη. Ο μοναδιαίος κώδικας ισοδυναμεί με απόδοση πιθανότητας 2^{-x} σε διαφορές μεγέθους x.

Πίνακας 16

```
FUNCTION Calc_Unary_Code(x)
BEGIN
  bitstr = NULL;
  FOR i = 1 TO x - 1
    bitstr = bitstr + 1;
  bitstr = bitstr + 0;
  return bitstr;
END

FUNCTION Calc_Unary_Number(bitstr, bit_pos)
BEGIN
  bit_value = get_bit_in_bitstr(bitstr, bit_pos);
  x = 1; bit_pos = bit_pos + 1;
  WHILE bit_value = 1
  BEGIN
    x = x + 1; bit_pos = bit_pos + 1;
    bit_value = get_bit_in_bitstr(bitstr, bit_pos);
  END
  return x;
END
```

Μοναδιαίος κώδικας. Ο ψευδοκώδικας κωδικοποίησης και αποκωδικοποίηση ενός ακεραίου. Ο αριθμός αυμβολίζεται με x, η αναπαράσταση του με bitstr και bit_pos είναι το ψηφίο μέσα στην αναπαράσταση από το οποίο ξεκινάει η διαδικασία της αποκωδικοποίησης.

3.5.1.2. Κώδικας gamma

Ο κώδικας Elias' gamma (γ) είναι ένας γνωστός μη – παραμετροποιημένος κώδικας με μεγάλο βαθμό συμπίεσης των ακεραίων ενός inverted file. Ο κώδικας αυτός χωρίζει την αναπαράσταση κάθε θετικού ακεραίου x σε δύο σκέλη με τον ακόλουθο τρόπο :

- Το πρώτο μέρος είναι ο μοναδιαίος κώδικας του αριθμού $k = 1 + \lfloor \log_2 x \rfloor$
- Το δεύτερο μέρος είναι η δυαδική αναπαράσταση του αριθμού $r = x - 2^{\lfloor \log_2 x \rfloor}$ χρησιμοποιώντας $\lfloor \log_2 x \rfloor$ ψηφία.

Ο κώδικας ισοδυναμεί με απόδοση πιθανότητας $Pr[x] = 1/2x^2$ σε διαφορές μεγέθους x. Τους αλγόριθμους κωδικοποίησης και αποκωδικοποίησης που χρησιμοποιεί τους βλέπουμε παρακάτω.

Πίνακας 17

```
FUNCTION Calc_Gamma_Code(x)
BEGIN
  bitstr1 = Calc_Unary_Code(1 + ⌊log2x⌋);
  bitstr2 = Calc_Binary_Code(x − 2^⌊log2x⌋, ⌊log2x⌋);
  return bitstr1 + bitstr2;
END

FUNCTION Calc_Gamma_Number(bitstr, bit_pos)
BEGIN
  k = Calc_Unary_Number(bitstr, bit_pos)
  r = Calc_Binary_Number(bitstr, bit_pos, k − 1)
  x = 2^(k−1) + r;
  return x;
END
```

Κώδικας gamma. Ο ψευδοκώδικας κωδικοποίησης και αποκωδικοποίηση ενός ακεραίου. Ο αριθμός αυμβολίζεται με x, η αναπαράσταση του με bitstr και bit_pos είναι το ψηφίο μέσα στην αναπαράσταση από το οποίο ξεκινάει η διαδικασία της αποκωδικοποίησης.

Έστω ότι έχουμε την ακολουθία 111000111011. Η διαδικασία αποκωδικοποίησης της είναι ως εξής : Αρχικά εξετάζονται μία προς μία όλες οι τιμές των ψηφίων της ακολουθίας και μετράμε το πλήθος των άσσων ώσπου να εντοπιστεί το πρώτο ψηφίο με τιμή '0'. Το πρώτο μηδενικό ψηφίο αγνοείται και στη συνέχεια λαμβάνεται η δυαδική τιμή της αναπαράστασης που ακολουθεί και αποτελείται από πλήθος ψηφίων ίσο με το πλήθος των άσσων της προηγούμενης διαδικασίας. Στο παράδειγμα μας η τιμή αυτή είναι '1' και αντιστοιχεί στην αναπαράσταση 001. Εδώ έχει ολοκληρωθεί η ανάγνωση των ψηφίων του πρώτου κωδικοποιημένου αριθμού, του οποίου η πραγματική τιμή προκύπτει από το $2^3 + 1 = 9$. Γενικότερα οι k – 1 διαδοχικοί άσσοι που προηγούνται του πρώτου μηδενικού ψηφίου, αποτελούν μαζί με το μηδενικό ψηφίο το πρώτο μέρος του κωδικοποιημένου αριθμού, δηλαδή το μοναδιαίο κώδικα του αριθμού k. Η δυαδική αναπαράσταση με τιμή r των k – 1 ψηφίων που ακολουθούν του πρώτου μηδενικού, είναι το δεύτερο μέρος του κωδικοποιημένου αριθμού. Ο πραγματικός αριθμός υπολογίζεται ως $x = 2^{k-1} + r$.

3.5.1.3. Κώδικας delta

Και ο κώδικας delta (δ) είναι γνωστός και έχει ικανοποιητικό βαθμό συμπίεσης. Ο κώδικας αυτός χωρίζει την αναπαράσταση κάθε θετικού ακεραίου x σε δύο σκέλη με τον ακόλουθο τρόπο :

- Το πρώτο μέρος είναι ο κώδικας γ του αριθμού $1 + \lfloor \log_2 x \rfloor$
- Το δεύτερο μέρος είναι η δυαδική αναπαράσταση του αριθμού $x - 2^{\lfloor \log_2 x \rfloor}$ χρησιμοποιώντας $\lfloor \log_2 x \rfloor$ ψηφία.

Ο κώδικας ισοδυναμεί με απόδοση πιθανότητας

$$\frac{1}{2x\log x^2}$$

σε διαφορές μεγέθους x. Τους αλγόριθμους κωδικοποίησης και αποκωδικοποίησης που χρησιμοποιεί τους βλέπουμε παρακάτω.

Στην αποκωδικοποίηση μιας ακολουθίας ακεραίων αρχικά χρησιμοποιείται ο αλγόριθμος αποκωδικοποίησης του κώδικα gamma για την ανάκτηση του αριθμού $k = 1 + \lfloor \log x \rfloor$. Η δυαδική αναπαράσταση με τιμή r των επόμενων k – 1 ψηφίων αποτελεί το δεύτερο μέρος του κωδικοποιημένου αριθμού. Η τελική μορφή του είναι $x = 2^{k-1} + r$. Η διαδικασία επαναλαμβάνεται έως ότου αποκωδικοποιηθούν όλα τα ψηφία της ακολουθίας.

Πίνακας 18

x	Μοναδιαίος	Κώδικας γ	Κώδικας δ	x	Μοναδιαίος	Κώδικας γ	Κώδικας δ
1	0	0	0	6	111110	11010	01100
2	10	1000	10000	7	1111110	11011	01101
3	1110	1001	10001	8	11111110	1110000	11000000
4	11110	11000	10100	9	111111110	11001	11000001
5	111110	11001	10101	10	1111111110	1110010	11000010

Και στους δύο κώδικες το codeword αποτελείται από δύο μέρη, το πρόθεμα και το επίθημα. Το πρόθεμα δείχνει το δυαδικό μέγεθος της αξίας και δείχνει ατον αποκωδικοποιητή πόσα bits υπάρχουν στο επίθημα. Το επίθημα δείχνει την αξία του αριθμού μέσα στην αντίστοιχη δυαδική σειρά.

Ποιος από τους δύο κώδικες θα προτιμηθεί εξαρτάται από την διανομή πιθανότητας της αξίας x που πρόκειται να κωδικοποιηθεί.

Όταν τα μισά από όλες τις αξίες είναι ο αριθμός 1, το ένα τέταρτο είναι 2, το ένα όγδοο είναι 3, και ούτω καθεξής, τότε προτιμάται ο unary κώδικας. Όταν οι μισές από τις αξίες είναι 1, το ένα τέταρτο είναι είτε 2 είτε 3, το ένα όγδοο είναι είτε 4 είτε 5 είτε 6 ή 7, και ούτω καθεξής, τοτε προτιμότερος κώδικας είναι ο gamma.

Πίνακας 19

value	unary	gamma	delta
1	0	0:	0::
2	10	10:0	10:0:0
3	110	10:1	10:0:1
4	1110	110:00	10:1:00
10	1111111110	1110:010	110:00:010
100		1111110:100100	110:11:100100
1,000		1111111110:111101000	1110:010:111101000

Colons are used as a guide to show the prefix and suffix components in each codeword. All three codes can represent arbitrarily large numbers; the unary codewords for 100 and 1,000 are omitted only because of their length.

3.5.1.4. Κώδικας Golomb

Σύμφωνα με τον κώδικα Golomb, έχοντας την παράμετρο b, κάθε ακέραιος x θα αποτελείται από δύο μέρη τα οποία προκύπτουν ως εξής :

- Το πρώτο μέρος είναι ο μοναδιαίος κώδικας του αριθμού q + 1, όπου $q = \lfloor (x-1)/b \rfloor$

- Το δεύτερο μέρος είναι η δυαδική αναπαράσταση του αριθμού r = x – q*b – 1 χρησιμοποιώντας $\lfloor \log_2 b \rfloor$ ή $\lceil \log_2 b \rceil$ ψηφία.

Οι πιθανότητες εμφάνισης που αποδίδονται σε αυτούς τους αριθμούς δεν μπορούν να υπολογιστούν καθώς εξαρτώνται και από την παράμετρο b. Το σύνολο των αναπαραστάσεων που προκύπτει είναι ένας μη παραμετροποιημένος κώδικας και μπορεί να παραμετροποιηθεί εάν συνδεθεί η τιμή του b με παραμέτρους που προκύπτουν από στατιστική επεξεργασία της ευρετηριοποιούμενης συλλογής κειμένων.

Πίνακας 20

	Golomb							
x	$b=2$	$b=3$	$b=4$	$b=6$	$b=7$	$b=11$	$b=12$	$b=13$
1	100	101	1000	1000	1000	10000	10000	10000
2	101	1010	1001	1001	1001 0	1001	1001	1001
3	1100	1011	1010	1010 0	1001 1	1001 0	1001 0	1001 0
4	1101	1100	1011	1011 0	1010 0	1001 1	1001 1	1001 10
5	11100	11010	11000	1011 10	1010 1	1010 0	1010 0	1001 11
6	11101	11011	11001	1100 0	1011 0	1010 10	1010 1	1010 0
7	111100	11100	11010	11000	1011 1	1010 11	1010 10	1010 1
8	111101	11010	11011	11001	11000	1011 0	1010 11	1010 10
9	1111100	11011	11000	11010 0	11001 0	1011 10	1011 0	1010 11
10	1111101	11100	11001	11010 1	11001 1	1011 11	1011 10	1011 0
11	11111100	11101 0	11010	11011 0	11010 0	1011 11	1011 11	1011 10
12	11111101	11101 1	11011	11011 1	11010 1	11000	1011 11	1011 11
13	111111100	11110 0	11100 0	11100 0	11011 0	11001	11000	11000
14	111111101	11110 1	11100 1	11100 1	11011 1	11001 0	11001	11001
15	1111111100	11111 0	11101 0	11101 00	11100 0	11001 1	11001 0	11001 0
16	1111111101	11111 10	11101 1	11101 01	11100 1	11010 0	11001 1	11001 10
17	11111111100	11111 110	11110 00	11101 10	11101 0	11010 1	11010 0	11001 11

Παραδείγματα κωδίκων Golomb για ακέραιους αριθμούς

Όσον αφορά το μήκος της δυαδικής αναπαράστασης του αριθμού r υπάρχουν δύο επιλογές,

$$\lfloor \log_2 b \rfloor \text{ ή } \lceil \log_2 b \rceil .$$

Ο αριθμός r μπορεί να πάρει τιμές από 0 έως b – 1. Χρησιμοποιώντας σταθερό μήκος δυαδικής αναπαράστασης, το ελάχιστο πλήθος που απαιτούνται για την κωδικοποίηση b διαφορετικών τιμών είναι

$$\lceil \log_2 b \rceil .$$

Αυτά μπορούν να κωδικοποιήσουν $2^{\lceil \log_2 b \rceil}$ τιμές. Οπότε όταν έχουμε b διακριτές τιμές υπάρχουν $2^{\lceil \log_2 b \rceil} - b$ αχρησιμοποίητοι συνδυασμοί.

Στην αποκωδικοποίηση ανακτώνται $\lfloor \log_2 b \rfloor$ ψηφία και υπολογίζεται η δεκαδική τους τιμή. Αν η τιμή αυτή είναι μικρότερη του $2^{\lceil \log_2 b \rceil} - b$ τότα αντιστοιχεί στην πραγματική τιμή του r. Αλλιώς λαμβάνεται και το επόμενο ψηφίο και υπολογίζεται και η δεκαδική τιμή k, ενώ για τον υπολογισμό του r, η τιμή του k μειώνεται κατά την ποσότητα $2^{\lceil \log_2 b \rceil} - b$.

Πίνακας 21

```
FUNCTION Calc_Golomb_Code(x, b)
BEGIN
  q = ⌊(x − 1)/b⌋;
  bitstr1 = Calc_Unary_Code(q + 1);
  r = x − q × b − 1;
  IF r ≥ 2^⌈log2b⌉ − b THEN
    bitstr2 = Calc_Binary_Code(r + 2^⌈log2b⌉ − b, ⌈log2b⌉);
  ELSE
    bitstr2 = Calc_Binary_Code(r, ⌊log2b⌋);
  return bitstr1 + bitstr2;
END

FUNCTION Calc_Golomb_Number(bitstr, b, bit_pos)
BEGIN
  q = Calc_Unary_Number(bitstr, bit_pos) − 1;
  r = Calc_Binary_Number(bitstr, ⌊log2b⌋, bit_pos);
  IF r ≥ 2^⌈log2b⌉ − b
  BEGIN
    bit_pos = bit_pos − ⌊log2b⌋;
    k = Calc_Binary_Number(bitstr, ⌈log2b⌉, bit_pos);
    r = k − (2^⌈log2b⌉ − b);
  END
  x = r + q × b + 1;
  return x;
END
```

Κώδικας Golomb. Ο ψευδοκώδικας κωδικοποίησης και αποκωδικοποίηση ενός ακεραίου. Ο αριθμός αυμβολίζεται με x, η αναπαράσταση του με bitstr και bit_pos είναι το ψηφίο μέσα στην αναπαράσταση από το οποίο ξεκινάει η διαδικασία της αποκωδικοποίησης.

Ένα παράδειγμα κατά το οποίο θα φανεί η διαδικασία είναι μια περίπτωση κατά την οποία το b παίρνει την τιμή '6'. Οπότε το r μπορεί να λάβει τιμές μεταξύ 0 και 5, δηλαδή 6 διακριτές τιμές για τις οποίες ο δυαδικός κώδικας απαιτεί τουλάχιστον

$$\lceil \log_2 6 \rceil = 3 \text{ ψηφία}.$$

Έτσι με βάση τα παραπάνω $2^{\lceil \log_2 6 \rceil} - 6 = 2$ τιμές του r και συγκεκριμένα οι 0 και 1 θα αποθηκευτούν με $\lfloor \log_2 6 \rfloor = 2$ ψηφία. Όταν x = 1 τότε r = 0 που κωδικοποιείται ως 00. Όταν x = 12 τότε r = 5 και αποθηκεύεται σε 3 ψηφία η ποσότητα

$$5 + (2^{\lceil \log_2 6 \rceil} - 6) = 7.$$

Κατά την αποκωδικοποίηση της τιμής r = 0, λαμβάνονται αρχικά τα πρώτα $\lfloor \log_2 6 \rfloor = 2$ ψηφία της αναπαράστασης του r. Υπολογίζεται η δεκαδική τιμή της ακολουθίας οπότε παίρνουμε την τιμή 0 και επειδή

$$0 < 2^{\lceil \log_2 6 \rceil} - 6 \text{ τελικά } r = 0$$

Για τη τιμή r = 5 η διαδικασία διαφέρει. Παίρνουμε τα $\lfloor \log 6 \rfloor = 2$ ψηφία της αναπαράστασης του r, δηλαδή η ακολουθία '11' με δεκαδική τιμή 3. Επειδή $3 \geq 2^{\lfloor \log 6 \rfloor} - 6$ παίρνουμε και το επόμενο ψηφίο. Τελικά η ακολουθία μετατρέπεται σε '111' με δεκαδική τιμή 7. Έτσι $r = 7 - (2^{\lfloor \log 6 \rfloor} - 6) = 5$.
Στο στάδιο της αποκωδικοποίησης της συνολικής αναπαράστασης αρχικά λαμβάνεται πρώτα το πρώτο μέρος που είναι ο μοναδιαίος κώδικας του αριθμού q. Στη συνέχεια λαμβάνεται το δεύτερο μέρος, η τιμή του r. Ο αρχικός αριθμός x είναι

$$x = r + q \times b + 1$$

3.5.2. Παραμετροποιημένοι

3.5.2.1. Αριθμητικός κώδικας ή κώδικας Huffman για το καθολικό μοντέλο των καταγεγραμμένων συχνοτήτων

Σύμφωνα με την συγκεκριμένη τεχνική κωδικοποίησης είναι απαραίτητη η καταγραφή των ακεραίων που εμφανίζονται ως διαφορές μέσα στις ανταστραμμένες λίστες. Η γνώση της κατανομής των καταγεγραμμένων συχνοτήτων μπορεί να αξιοποιηθεί, ώστε να πραγματοποιηθεί η συμπίεση των αντεστραμμένων λιστών με την βοήθεια είτε της κωδικοποίησης Huffman είτε με της αριθμητικής κωδικοποίησης. Ενώ θεωρητικά θα έπρεπε η συμπίεση να είναι καλύτερη, δεν συμβαίνει αυτό. Συμπεραίνουμε λοιπόν ότι τα πιθανολογικά μοντέλα των κωδικών του Elias (gamma, delta) προσεγγίζουν τα μοντέλα πραγματικών συλλογών κειμένων.

3.5.2.2. Κώδικας Golomb για το καθολικό μοντέλο Bernoulli

Έχουμε μια συλλογή N κειμένων που περιέχει n διακριτούς όρους και στα inverted files αποθηκεύονται f δείκτες σε κείμενα. Το καθολικό μοντέλο Bernoulli υποθέτει πως είναι ομοιογενείς οι μεγάλες συλλογές κειμένων, δηλαδή ότι οι όροι τους κατανέμονται ομοιόμορφα στα κείμενα. Άρα η πιθανότητα εμφάνισης ενός τυχαίου όρου σε ένα τυχαίο κείμενο είναι

$$p = f/ (N*n).$$

Έτσι η πιθανότητα εμφάνισης διαφοράς με τιμή x οποιοδήποτε όρου ενός inverted file θα δίνεται από την σχέση

$$Pr[x] = (1-p)^{x-1} p$$

καθώς θα πρέπει να υπάρχουν x – 1 διαδοχικά κείμενα στα οποία δεν εμφανίζεται ο όρος και ένα x- οστό όπου εμφανίζεται.

Η συγκεκριμένη κατανομή πιθανοτήτων μπορεί να υλοποιηθεί και με αριθμητική κωδικοποίηση. Οι δύο αθροιστικές πιθανότητες που απαιτούνται για τον τελικό υπολογισμό του υποδιαστήματος, το οποίο αντιστοιχεί σε κάθε διαφορά x για κάθε

βήμα των αλγορίθμων κωδικοποίησης και αποκωδικοποίησης θα δίνονται από τις σχέσεις:

$$low\_bound = \sum_{i=1}^{l-1}[(1-p)^{i-1}p] = 1-(1-p)^{l-1}$$

$$high\_bound = \sum_{i=1}^{l}[(1-p)^{i-1}p] = 1-(1-p)^{l}$$

Το δένδρο Huffman δεν μπορεί να κατασκευαστεί διότι όπως βλέπουμε στην πρώτη σχέση ο αλγόριθμος απαιτεί πεπερασμένο αριθμό στοιχείων εκτός της γνώσεως της κατανομής των πιθανοτήτων τους. Ο κώδικας που είναι κατάλληλος για την συμπίεση των διαφορών είναι ο κώδικας Golomb. Ο κώδικας Golomb που περιγράφεται για πρώτη φορά το 1966 από τον Solomon Golomb του Πανεπιστημίου της Νότιας Καλιφόρνιας περιγράφεται ως εξής:

Πίνακας 22

To encode integer $x \geq 1$ using parameter b:

[1] Factor $x-1$ into $q \cdot b + r + 1$ where $0 \leq r < b$.
[2] Code $q+1$ in unary.
[3] Set $c = \lceil \log_2 b \rceil$ and $g = 2^c - b$.
[4] If $0 \leq r < g$ then code r in binary using $c-1$ bits; otherwise, if $g \leq r < b$, then code $r+g$ in binary using c bits.

Οι Gallager και Van Voorhis το 1975 έδειξαν ότι εάν το b επιλεγεί ώστε να ικανοποιεί την παρακάτω συνθήκη:

$$(1-p)^b + (1-p)^{b+1} \leq 1 < (1-p)^{b-1} + (1-p)^b$$

τότε αυτός ο κώδικας παράγει ένα βέλτιστο πρόθεμα την γεωμετρική διανομή που ανταποκρίνεται στον Bernoulli με πιθανότητα επιτυχίας p. Λύνοντας την παραπάνω συνθήκη θα έχουμε για το b

$$b = \left\lceil \frac{\log_2(2-p)}{-\log_2(1-p)} \right\rceil$$

Ο κώδικας Golomb είναι στενά συνδεδεμένος με το unary και ορίζει τις πιθανότητες εκθετικά φθίνουσες. Παρόλα αυτά, στον Golomb κώδικα η εκθετική μείωση βασίζεται στην συνάρτηση του b . Πρέπει να σημειωθεί ότι εάν το p υπερβαίνει το 0.5 τότε ο κώδικας είναι περισσότερο αποτελεσματικός εάν το inverted file είναι ολοκληρωμένο προτού συμπιεστεί.

3.5.2.3. Κώδικας Golomb για το τοπικό μοντέλο Bernoulli

Η διαφορά ανάμεσα στο καθολικό και στο τοπικό μοντέλο είναι ότι στο τοπικό μοντέλο Bernoulli μια ξεχωριστή παράμετρος b συσχετίζεται με κάθε αντεστραμμένη λίστα. Η τιμή της παραμέτρου b για την λίστα διαφορών του όρου t υπολογίζεται όπως και στην περίπτωση του καθολικού μοντέλου Bernoulli μόνο που εδώ η τιμή του p προκύπτει από την σχέση

$$p = f_t / N$$

όπου f_t είναι το πλήθος των κειμένων της συλλογής όπου εμφανίζεται ο όρος. Κατά την διαδικασία αποσυμπίεσης της αντεστραμμένης λίστας υπολογίζουμε τη παράμετρο b χρησιμοποιώντας την ίδια τιμή f_t και ακολουθεί η γνωστή διαδικασία αποκωδικοποίησης της λίστας. Το τοπικό μοντέλο Bernoulli μπορεί να εφαρμοστεί και στις διαφορές που υπάρχουν στην αντεστραμμένη λίστα του όρου t. Η τιμή του p στην προκειμένη περίπτωση υπολογίζεται από την σχέση

$$p = f_{d,t} / N_d$$

όπου N_d είναι το συνολικό πλήθος όρων που εμφανίζονται στο κείμενο d και $f_{d,t}$ είναι το πλήθος των επαναλήψεων του όρου μέσα στο κείμενο.
.Χρησιμοποιώντας αυτήν την μέθοδο, οι λέξεις που είναι συχνές κωδικοποιούνται με μικρές τιμές του b, ενώ οι λιγότερο συχνές κωδικοποιούνται με μεγάλες τιμές. Οι πολύ κοινές λέξεις έχουν b = 1. Όταν b = 1 ο κώδικας εκφυλίζεται σε ένα σετ από unary codes για τα gap μεγέθη χωρίς δυαδικά στοιχεία. Αυτό είναι ισοδύναμο με το να αποθηκευθεί η αντεστραμμένη λίστα ως Bitvector. Αυτό είναι ένα δυαδικό διάνυσμα με ένα bit για κάθε δείμενο, όπου αυτό το bit δηλώνει την εμφάνιση του όρου στο κείμενο. Τα inverted file που συμπιέζονται με τον κώδικα του Huffman στο μοντέλο του Bernoulli δεν θα είναι ποτέ χειρότερα από το να συμπιεστούν με Bitvector ανά όρο.
Για να εκμεταλλευτούμε αυτό το local model, είναι απαραίτητο να αποθηκευτεί η παράμετρος f_t με κάθε αντεστραμμένη λίστα, ώστε η σωστή τιμή του b να μπορεί να χρησιμοποιηθεί κατά την διάρκεια της αποκωδικοποίησης

3.5.2.4. Αριθμητικός κώδικας για το τοπικό μοντέλο με κατανομή υπερβολής

Μια επίσης γνωστή επιλογή κατανομής των τιμών των διαφορών κάθε διακριτού όρου μιας συλλογής κειμένων είναι η κατανομή της υπερβολής. Σύμφωνα με την κατανομή της υπερβολής η πιθανότητα εμφάνισης διαφοράς μεγέθους x δίνεται από την σχέση

$$Pr[x] = p/x$$

Η τιμή του μ υπολογίζεται ως εξής :

$$p = \frac{1}{\ln(m+1) + 0.5772}$$

όπου 0.5772 είναι η σταθερά του Euler. Ως m μπορεί να ληφθεί η τιμή της μεγαλύτερης διαφοράς κάθε όρου.
Η μέθοδος που βασίζεται στο τοπικό μοντέλο με κατανομή υπερβολής έχει μαγαλύτερο βαθμό συμπίεσης από τον κώδικα Golomb για το τοπικό μοντέλο Bernoulli. Όμως το γεγονός ότι υλοποιείται μόνο με αριθμητική κωδικοποίηση αποτελεί μειονέκτημα διότι επιβραδύνεται η διαδικασία αποκωδικοποίησης.

3.5.3. Κώδικες συμπίεσης που δεν χρησιμοποιούν διαφορές

Η χρησιμοποίηση του υπολογισμού της μεταβολής των δεικτών δεν αποτελεί μοναδική λύση στην συμπίεση των αντεστραμμένων λιστών. Υπάρχουν μέθοδοι οι οποίες κωδικοποιούν το περιεχόμενο των αντεστραμμένων λιστών χωρίς να βασίζονται στον υπολογισμό των διαφορών. Η πιο ενδιαφέρουσα μέθοδος κωδικοποίησης αντεστραμμένων λιστών, που δεν χρησιμοποιεί την μεταβολή των δεικτών, είναι ο Κώδικας Παρεμβολής.

3.5.3.1. Κώδικες παρεμβολής

Για να γίνει περισσότερο κατανοητός ο τρόπος λειτουργίας του κώδικα παρεμβολής θα χρησιμοποιήσουμε ένα παράδειγμα. Έστω ότι η αντεστραμμένη λίστα ενός όρου t, ο οποίος εμφανίζεται σε $f_t = 7$ κείμενα μιας συλλογής N = 20 κειμένων είναι η {3, 8, 9, 11, 12, 13, 17}. Έστω επιπλέον ότι η τιμή του δεύτερου δείκτη είναι γνωστή πριν κωδικοποιηθεί ο πρώτος δείκτης. Στο παράδειγμα μας εάν ο δεύτερος δείκτης είναι το κείμενο 8 τότε είναι προφανές ότι ο πρώτος περιορίζεται σε κάποιες από τις τιμές του συνόλου [1...7]. Χρησιμοποιώντας το δυαδικό σύστημα μας αρκούν τρία ψηφία για την αναπαράσταση οποιουδήποτε συνόλου. Έστω ακόμα ότι είναι γνωστός και ο τέταρτος δείκτης που είναι το κείμενο 11. Οπότε ο τρίτος κυμαίνεται μεταξύ [9...10]. Ένα ψηφίο είναι αρκετό για την αναπαράσταση δύο διακριτών τιμών, χρησιμοποιώντας το δυαδικό κώδικα. Εάν είναι γνωστό ότι ο έκτος δείκτης είναι το κείμενο 13 τότε δεν χρειάζεται κανένα ψηφίο για την αναπαράσταση του ο πέμπτος δείκτης. Όσον αφορά τον έβδομο δείκτη, θα είναι μεταξύ [14...20] δεδομένου ότι έχουμε 20 κείμενα. Ο δυαδικός κώδικας απαιτεί τρία ψηφία για την κωδικοποίηση των τιμών του συγκεκριμένου πεδίου.
Οι αναπαραστάσεις των δεικτών στις θέσεις 1, 3, 5, 7 απαιτούν την γνώση των δεικτών στις θέσεις 2, 4, 6. Οπότε θα πρέπει να έχει προηγηθεί η κωδικοποίηση της λίστας. {8, 11, 13}. Η τεχνική κωδικοποίησης που περιγράψαμε προηγουμένως μπορεί να εφαρμοστεί και σε αυτήν την λίστα. Αν ο δεύτερος δείκτης ως προς το κείμενο 11 είναι ήδη γνωστός τότε ο πρώτος έχει πιθανές τιμές [1...10] και απαιτεί 4 ψηφία. Η διαφοροποίηση αυτής της περίπτωσης είναι ότι επειδή η δεύτερη λίστα χρησιμοποιείται για την κωδικοποίηση της πρώτης, αξιοποιείται ένα επιπλέον στοιχείο : ο πιθανός δείκτης της πρώτης θέσης της δεύτερης λίστας εννοείται ότι έχει δύο επιπλέον δείκτες, έναν αριστερό και έναν δεξί. Έτσι το πεδίο τιμών του μπορεί να περιοριστεί περισσότερο σε [2…9], με αποτέλεσμα για την κωδικοποίηση του να χρειάζονται 3 ψηφία.
Το μόνο πρόβλημα που απομείνει είναι η κωδικοποίηση του δείκτη ως προς το κείμενο 11. Ένας πενταψήφιος αριθμός είναι αρκετός. Όμως μπορεί να αξιοποιηθεί η γνώση ότι δεξιά και αριστερά υπάρχουν από 3 δείκτες στην αρχική λίστα και μπορεί το εύρος του να γίνει [4…17].

Πίνακας 23

```
FUNCTION Interpolative_Encode(x[], f, lo, hi, bitstr)
BEGIN
  IF f = 0 THEN return;
  IF f = 1 THEN
  BEGIN
    // binary representation for x[0] ∈ [lo...hi]
    bitstr = bitstr +
      + Calc_Binary_Code(x[0] − lo + 1, ⌈log2(hi − lo + 1)⌉);
    return;
  END
  ELSE
  BEGIN
    h = ⌊f/2⌋; m = x[h]; f1 = h; f2 = f − h − 1;
    x1 = x[0...h − 1]; x2 = x[h + 1...f − 1];
    // binary representation for m = x[h] ∈ [lo + f1...hi − f2]
    bitstr = bitstr +
      + Calc_Binary_Code(m − (lo + f1) + 1, ⌈log2(hi − f2 − (lo + f1) + 1)⌉);
    Interpolative_Encode(x1, f1, lo, m − 1, bitstr);
    Interpolative_Encode(x2, f2, m + 1, hi, bitstr);
    return;
  END
END
```

Κώδικας παρεμβολής

5.6. Κώδικες συμπίεσης σε επίπεδο ψηφιολέξης

Οι κώδικες συμπίεσης σε επίπεδο ψηφιολέξης μπορούν επίσης να χρησιμοποιηθούν γαι την συμπίεση των inverted files. Στην περίπτωση αυτή κάθε ακέραιος δεν αποθηκεύεται ως ακολουθία ψηφίων αλλά ως ακολουθία ψηφιολέξεων. Σε κάθε ψηφιολέξη δεσμεύεται το πιο σημαντικό ψηφίο και τα υπόλοιπα 7 χρησιμοποιούνται για την κωδικοποίηση τμήματος της συνολικής δυαδικής αναπαράστασης του ακεραίου. Στην τελευταία ψηφιολέξη το δεσμευμένο ψηφίο έχει την τιμή '1' ώστε να σηματοδοτείται το τέλος της ακολουθίας. Στις υπόλοιπες έχει τιμή '0'. Οι κώδικες συμπίεσης επιπέδου ψηφιολέξης παράγουν ευρετήρια με αυξημένο μέγεθος. Είναι όμως ταχύτερες διότι δεν απαιτούνται πράξεις μεμονομένων ψηφίων.

Ο κώδικας συμπίεσης σε επίπεδο ψηφιολέξης βελτιώνει αισθητά την ταχύτητα επεξεργασίας των ερωτημάτων. Αυτό εξηγείται διότι η αύξηση στο μέγεθος του ευρετηρίου από την χρήση κωδίκων σε επίπεδο ψηφιολέξης είναι μκρή. Επομένως δε επιβραδύνεται η επεξεργασία των ερωτημάτων. Όταν τα δεδομένα βρίσκονται στην κυρία μνήμη κατά την αποκωδικοποίηση δεν απαιτούνται χρονοβόρες συνθήκες ελέγχου και πράξεων με τα ψηφία. Τέλος εάν το ζητούμενο είναι η ελαχιστοποίηση του χώρου που καταλαμβάνει το ευρετήριο στον δίσκο, τότε προτείνεται η εφαρμογή κωδίκων σε επίπεδο ψηφίου γιατί έχουν μεγαλύτερο βαθμό συμπίεσης και δημιουργούν μικρότερα ευρετήρια.

5.7. Απόδοση των μεθόδων συμπίεσης των inverted files

Το πρόβλημα της συμπίεσης - κωδικοποίησης δεδομένων απαιτεί την συνδυασμένη χρήση διαφορετικών συστατικών. Συγκεκριμένα απαιτεί την ανάπτυξη ενός μοντέλου κατανομής πιθανοτήτων για την πρόβλεψη ή την περιγραφή των δεδομένων που πρόκειται να συμπιεστούν. Επιπλέον απαιτεί μια μέθοδο κωδικοποίησης η οποία θα δεχτεί ως είσοδο της το παραγόμενο μοντέλο και τα δεδομένα ώστε να υπολογίσει μια ακολουθία ψηφίων περιορισμένου μεγέθους για την αναπαράσταση τους.
Τα μοντέλα κατανομής διακρίνονται στα στατικά, στα ημιστατικά και στα μοντέλα προσαρμογής.
Οι κώδικες που χρησιμοποιούνται στην συμπίεση των ακεραίων που αποθηκεύονται στις αντεστραμμένες λίστες συνδυάζουν την ικανότητα στη συμπίεση με την ταχύτητα στην αποκωδικοποίηση. Διακρίνονται σε κώδικες συμπίεσης σε επίπεδο ψηφίου και σε επίπεδο ψηφιολέξης. Οι κώδικες συμπίεσης σε επίπεδο ψηφίου διαχωρίζονται στους παραμετροποιημένους και στους μη παραμετροποιημένους, που αξιοποιούν τον υπολογισμό της μεταβολής των δεικτών και σε κώδικες που χρησιμοποιούν άλλες τεχνικές για να επιτύχουν μεγάλα ποσοστά συμπίεσης. Οι παραμετροποιημένοι κώδικες και οι τελευταίοι ενδείκνονται για την ελαχιστοποίηση του μεγέθους των παραγόμενων ευρετηρίων, ενώ οι μη παραμετροποιημένοι προτείνονται για συλλογές με συχνές εισαγωγές, διαγραφές και μεταβολές κειμένων. Όλοι οι κώδικες συμπίεσης σε επίπεδο ψηφίου έχουν ως αποτέλεσμα ευρετήρια που ειναι πιο συμπαγή σε σχέση με αυτά που παράγονται από κώδικες σε επίπεδο ψηφιολέξης. Οι κώδικες σε επίπεδο ψηφιολέξης όμως έχουν ταχύτερη αποκωδικοποίηση και προτιμώνται σε περιπτώσεις που θέλουμε να έχουμε την μέγιστη επιτάχυνση των διαδικασιών αποκωδικοποίησης των αντεστραμμένων λιστών.

5.8. Βιβλιογραφία

1. **Using d-gaps patterns for index compression, Jilnin Clen**
2. **Διακριτά Μαθηματικά, Παναγιώτης Μποζάνης**
3. **Managing gigabytes, Compressing and indexing documents and images**
4. **Inverted files for text search engines**
5. **www.google.com**

4. Querying

Inverted files

Κατασκευή ευρετηρίου

Σύμφωνα με την δομή ενός αντεστραμμένου αρχείου – ευρετηρίου κειμένων δημιουργείται μια αντεστραμμένη λίστα κειμένου για κάθε διακριτό όρο της συλλογής. Η αντεστραμμένη λίστα περιέχει τους δείκτες όλων των κειμένων, όπου εμφανίζεται ο όρος, με αύξουσα σειρά. Οι αντεστραμμένες λίστες όλων των όρων

συγκεντρώνονται σε ένα μοναδικό αρχείο, που ονομάζεται αντεστραμμένο αρχείο – ευρετήριο κειμένου.
Κάθε αντεστραμμένο αρχείο – ευρετήριο κειμένου πρέπει να συνοδεύεται από ένα λεξικό (*Lexicon*). Το λεξικό αυτό είναι μια δομή (B+ δένδρο ή αρχείο κατακερματισμού) η οποία αποθηκεύει κάθε διακριτό όρο της συλλογής κειμένων συνοδευόμενο με ένα δείκτη που δειχνει την αντεστραμμένη λίστα του όρου μέσα στο inverted file. Επίσης μπορεί να περιέχεται και οποιαδήποτε άλλη χρήσιμη πληροφορία για τον όρο.

Επεξεργασία ερωτήματος

Για να ανακτήσουμε τα κείμενα που ικανοποιούν τα κριτήρια ενός ερωτήματος της μορφής (t1 &t2&…tn) ακολουθούμε την εξής διαδικασία :
Αρχικά εντοπίζονται οι όροι στο λεξικό, μέσω του αντίστοιχου δείκτη κάθε όρου, και ακολούθως εντοπίζονται οι αντεστραμμένες λίστες από το inverted file. Θέλουμε να υπολογίσουμε την τομή αυτών των λιστών. Ακολουθεί ο ψευδοκώδικας.

Πίνακας 24

```
FUNCTION if_conjunction(terms[])
BEGIN
  // Sort the terms contained in the query according
  // to their document frequency (ascending order)
  increasing_frequency_order(terms);
  doc_id_set = [];
  last_doc_id = the total number of documents in the collection;
  FOR i = 1 TO sizeof(terms)
  BEGIN
    doc_id = get_first_doc_id_from_inverted_list(terms[i]);
    temp_set = [];
    WHILE doc_id <> NIL AND doc_id < last_doc_id
    BEGIN
      temp_set = temp_set + doc_id;
      doc_id = get_next_doc_id_from_inverted_list(terms[i]);
    END
    doc_id_set = conjunct(doc_id_set, temp_set);
    last_doc_id = get_largest_value(doc_id_set);
  END
  return doc_id_set;
END
```

Ο ψευδοκώδικας για τον υπολογισμό της τομής αντεστραμμένων λιστών σε ερώτημα σύζευξης, οι όροι του οποίου εμπεριέχονται στο term[].

Πρέπει να αποθηκευθεί στο λεξικό η συχνότητα εμφάνισης κάθε όρου μέσα στην συλλογή κειμένων. Με τον όρο συχνότητα εμφάνισης όρου στην συλλογή εννοείται ο αριθμός των κειμένων που περιέχουν των όρο και όχι το συνολικό πλήθος των εμφανίσεων του όρου στα κείμενα της συλλογής. Η επιστρεφόμενη τομή περιέχει τους αύξοντες αριθμούς τω κειμένων στα οποία εμπεριέχεται ο όρος. Θα πρέπει να υπάρχει κάποια δομή η οποία συσχετίζει α/α με διευθύνσεις στο δίσκο, ώστε να είναι δυνατή η ανάκτηση των κειμένων.
Η επεξεργασία ενός ερωτήματος σύζευξης αρχίζει με την αντεστραμμένη λίστα που έχει τη μικρότερη συχνότητα εμφάνισης. Η λίστα αυτή περιέχει το ελάχιστο πλήθος

στοιχείων σε σχέση με τις λίστες των υπόλοιπων όρων και τοποθετείται στο αρχικά άδειο σύνολο των υποψήφιων απαντήσεων. Στη συνέχεια λαμβάνονται τα στοιχεία της αντεστραμμένης λίστας του όρου με την αμέσως μικρότερη συχνότητα εμφάνισης και αφαιρούνται από το σύνολο των υποψηφίων απαντήσεω όσοι αύξοντες αριθμοί δεν εμφανίζονται στην νέα λίστα. Στην ουσία από την νέα αντεστραμμένη λίστα ανακτώνται μόνο όσοι αύξοντες αριθμοί είναι μικρότεροι από τον μεγαλύτερο αύξοντα αριθμό του συνόλου των υποψηφίων απαντήσεων. Η διαδικασία επαναλαμβάνεται έως ότου να τελειώσουν οι όροι που αναφέρονται στο ερώτημα. Με αυτόν τον τρόπο εξασφαλίζουμε την ελαχιστοποίηση των πράξεων στο σύνολο των υποψηφίων απαντήσεων, καθώς και του κόστους μεταφοράς δεδομένων, διότι αρκετά από τα στοιχεία των αντεστραμμένων λιστών εκτός του πρώτου δεν ανακτώνται.

Signature files

Κατασκευή ευρετηρίου

Για την κατασκευή ενός signature file αρχικά αντιστοιχίζεται κάθε διακριτός όρος της συλλογής με μια υπογραφή μήκους F ψηφίων, μέσω κατάλληλης διαδικασίας κατακερματισμού. Στη συνέχεια, για κάθε κείμενο της συλλογής υπολογίζεται μια ξεχωριστή υπογραφή του ίδιου μήκους των F ψηφίων, με την υπέρθεση (λογικό or) των ψηφίων των υπογραφών των διακριτών όρων που εμπεριέχει. Το τελικό αρχείο υπογραφών προκύπτει από ένωση όλων των υπογραφών των κειμένων.

Επεξεργασία ερωτήματος

Η επεξεργασία ενός ερωτήματος χωρίζεται σε δύο στάδια. Στο πρώτο στάδιο επεξεργασίας υπολογίζεται η υπογραφή του ερωτήματος με υπέρθεση των υπογραφών των όρων που περιέχει. Εάν η υπογραφή ενός κειμένου έχει την τιμή '0' σε θέσεις ψηφίων που η υπογραφή του ερωτήματος έχει την τιμή '1' τότε αποκλείεται. Αυτό συμβαίνει διότι εάν οι υπό αναζήτηση όροι περιέχονταν στο κειμενο, τότε λόγω υπέρθεσης, η τελική τιμή του κειμένου θα έπρεπε να έχει την τιμή '1' σε όλα τα ψηφία που η υπογραφή του ερωτήματος έχει τιμή '1'.
Στο δευτερο στάδιο τα κείμενα που δεν έχουν αποκλειστεί ανιχνεύονται ένα προς ένα ώστε να απομακρυνθούν όσα έχουν συμπτωματικά κατάλληλες τιμές στις επίμαχες θέσεις των υπογραφών τους. Ακολουθεί ο ψευδοκώδικας.

Πίνακας 25

Ο ψευδοκώδικας για την επεξεργασία ερωτήματος σύζευξης με signature files. Οι όροι του ερωτήματος περιέχονται στο term[].

```
FUNCTION scsf_conjunction(terms[])
BEGIN
  query_signature = or_signatures(terms);
  doc_id_set = [];
  FOR doc_id = 1 TO the total number of documents in the collection
  BEGIN
    doc_signature = get_signature_for_doc(doc_id, signature_file);
    bits_ok = check_corresponding_1_bits(query_signature, doc_signature);
    IF bits_ok = TRUE THEN doc_id_set = doc_id_set + doc_id;
  END
  doc_id_set = remove_false_matches(terms, doc_id_set);
  return doc_id_set;
END
```

Bitmaps

Κατασκευή ευρετηρίου

Έχουμε μια συλλογή N κειμένων με V διακριτούς όρους. Το bitmap είναι η συνένωση πολλών υπογραφών, μια για κάθε κείμενο της συλλογής που ευρετηριοποιείται. Το bitmap ενός κειμένου αποτελείται από V διαδοχικά ψηφία, τα οποία αντιστοιχίζονται με ένα διακριτό όρο του λεξιλογίου μέσω B+ δένδρου ή ελαχιστοποιημένης συνάρτησης τέλειου κατακερματισμού. Η υπογραφή έχει ενεργά όλα τα ψηφία που αντιστοιχούν σε όρους που περιέχονται στο κείμενο της, και ανενεργά τα υπόλοιπα. Με αυτόν τον τρόπο αποφεύγεται το σφάλμα ταυτιποίησης, διότι δεν υπάρχουν επικαλύψεις ψηφίων με τιμή '1'.
Τα bitmaps δεν μπορούν να χρησιμοποιηθούν για την ευρετηριοποίηση του συνόλου του λεξιλογίου και των κειμένων μιας πραγματικής συλλογής, εξαιτίας των υπερβολικά αυξημένων απαιτήσεων σε αποθηκευτικό χώρο. Αν το μήκος της υπογραφής κάθε κειμένου είναι V ψηφία και η συλλογή αποτελείται από N κείμενα, τότε το μέγεθος του bitmap θα είναι N*V bits.

Επεξεργασία ερωτήματος

Η επεξεργασία ενός ερωτήματος που χρησιμοποιεί bitmap είναι πολύ απλή σε σχέση με τα υπόλοιπα. Αρχικά ελέγχονται ένα ένα όλα τα ψηφία των υπογραφών του ευρετηρίου που αντιστοιχούν στους όρους που περιέχονται στο ερώτημα. Ένα κείμενο που έχει ως απάντηση εάν έχει όλα τα επίμαχα ψηφία της υπογραφής ενεργά επιστρέφεται στον χρήστη. Η διαδικασία του ελέγχου σε κάθε υπογραφή σταματάει όταν όλα τα επίμαχα ωηφία βρεθούν ενεργά ή όταν βρεθεί έστω και ένα ανενεργό. Ο αλγόριθμος είναι πιο αποδοτικός όταν ελαχιστοποιείται το πλήθος των συγκρίσεων μεταξύ των ψηφίων. Για να γίνει αυτό εφικτό θα πρέπει η επεξεργασία των ψηφίων κάθε υπογραφής να γίνει με αύξουσα σειρά συχνότητας εμφάνισης των όρων του ερωτήματος στα κείμενα της συλλογής. Ακολουθεί ο ψευδοκώδικας.

Πίνακας 26

```
FUNCTION bf_conjunction(terms[])
BEGIN
  increasing_frequency_order(terms);
  doc_id_set = [];
  FOR doc_id = 1 TO the total number of documents in the collection
  BEGIN
    bits_ok = TRUE;
    bitmap = get_document_bitmap(doc_id, bitmap_file);
    FOR i = 1 TO sizeof(terms)
    BEGIN
      // h stores the corresponding bit for terms[i] (B+tree or MPHF)
      h = get_corresponding_bit(terms[i]);
      IF bitmap[h] = 0 THEN
      BEGIN
        bits_ok = FALSE;
        break; // break internal loop
      END
    END
    IF bits_ok = TRUE THEN doc_id_set = doc_id_set + doc_id;
  END
  return doc_id_set;
END
```

Βιβλιογραφία

1. **Inverted files for text search engines, Justin Zobel and Alistair Moffat**
2. **Managing gigabytes: Compressing and Indexing Documents and Images, Ian H.Witten, Alistair Moffat, Timothy C.Bell, Second Edition**
3. **www.google.com**