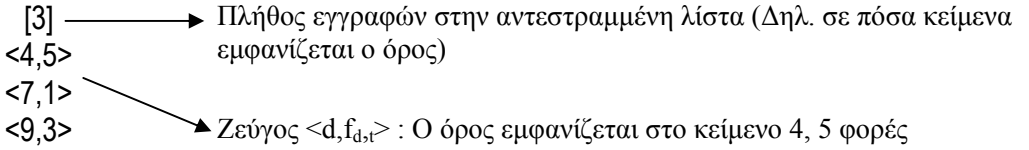
ΠΑΡΑΡΤΗΜΑ Α

Μέθοδοι Δημιουργίας Αντεστραμμένων Αρχείων

Μία από τις βασικότερες δομές ευρετηριοποίησης σε συλλογές εγγράφων είναι τα αντεστραμμένα αρχεία (inverted files). Ένα αντεστραμμένο αρχείο είναι ένα σύνολο αντεστραμμένων λιστών οι οποίες βρίσκονται αποθηκευμένες μέσα σε ένα αρχείο στο δίσκο, όπου μία αντεστραμμένη λίστα περιλαμβάνει όλες τις εμφανίσεις ενός όρου μέσα στα κείμενα της συλλογής.
Η δομή μίας αντεστραμμένης λίστας φαίνεται στο ακόλουθο σχήμα:

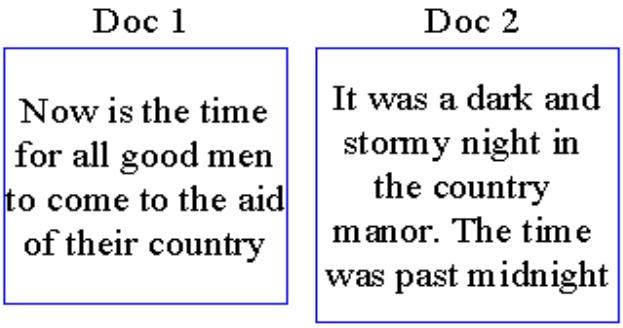


Ανάλογα με τις απαιτήσεις της εφαρμογής μια εγγραφή αντεστραμμένης λίστας μπορεί να περιέχει περισσότερη ή λιγότερη πληροφορία (π.χ. α/α παραγράφου στην οποία εμφανίζεται ο όρος κ.λ.π.). Οι όροι της συλλογής οργανώνονται (για διευκόλυνση του ψαξίματος) σε ένα λεξικό (συνήθως στην κύρια μνήμη) ενώ οι αντεστραμμένες λίστες αποθηκεύονται στον δίσκο.
Υπάρχουν πολλοί τρόποι για να αποθηκεύσει κανείς το λεξικό, οι βασικότεροι από τους οποίους είναι:
-*Ταξινομημένο*: απλά αποθηκεύονται οι όροι ο ένας μετά τον άλλο σε ένα ταξινομημένο πίνακα,
-*Tries* (Ψηφιακά Δέντρα) : οι όροι αποθηκεύονται σε ένα ψηφιακό δέντρο,
-*Perfect hashing*: οι όροι αποθηκεύονται με χρησιμοποίηση μίας συνάρτησης τέλειου κατακερματισμού (perfect hashing), αυτή η επιλογή προτιμάται για λεξικά σταθερού μεγέθους που δεν ανανεώνονται,
-B-trees: οι όροι αποθηκεύονται σε μία δενδρική δομή αποθήκευσης στη δευτερεύουσα μνήμη όπως είναι το B-δέντρο,
-*Front-coding*: αποθηκεύει τους όρους ταξινομημένους αλλά δεν επαναλαμβάνει το πρώτο μέρος τους. Απαιτεί πολύ μικρότερο χώρο από ένα απλό ταξινομημένο πίνακα.

Ένας πρόχειρος αλγόριθμος για τη δημιουργία ενός αντεστραμμένου αρχείου είναι ο ακόλουθος: κατασκευάζεται στη μνήμη ένας αντεστραμμένος πίνακας συχνοτήτων, διαβάζοντας το κείμενο με τη σειρά των κειμένων, μια στήλη του πίνακα, δηλαδή, κάθε φορά. Έπειτα, ο πίνακας εγγράφεται στο δίσκο με τη σειρά των όρων, γραμμή προς γραμμή. Παρά την προφανή ευκολία αυτής της προσέγγισης, στην πραγματικότητα η αντιστροφή είναι μια επίπονη διαδικασία και το πρόβλημα εντοπίζεται στο μέγεθος του πίνακα συχνοτήτων. Ας υποθέσουμε μια συλλογή με 8.965 όρους και 31.101 έγγραφα. Αν ένας 4-byte ακέραιος επιτρέπεται για κάθε είσοδο του πίνακα συχνοτήτων, τότε ο πίνακας θα καταλάμβανε 4*8.965*31.101 bytes στο δίσκο. Αυτό το μέγεθος είναι περίπου 1 GB, σχεδόν υποστηριζόμενο από ένα καλό μηχάνημα. Για μεγαλύτερες συλλογές όμως το κόστος σε χώρο φτάνει την τάξη των Terabytes.

Το παρακάτω παράδειγμα (από την αναφορά [WMB99]) δίνει τις βασικές φάσεις δημιουργίας ενός αντεστραμμένου αρχείου:

Φάση1:



Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Φάση2:

Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

Φάση3:

Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
is	2
is	1
it	2
major	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
is	2	1
is	1	1
it	2	1
major	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

Φάση4:

file

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
is	2	1
is	1	1
it	2	1
major	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

index

Term	N docs	Tot Freq
a	1	1
aid	1	1
all	1	1
and	1	1
come	1	1
country	2	2
dark	1	1
for	1	1
good	1	1
is	1	1
is	1	1
it	1	1
major	1	1
men	1	1
midnight	1	1
night	1	1
now	1	1
of	1	1
past	1	1
stormy	1	1
the	2	4
their	1	1
time	2	2
to	1	2
was	1	2

posting list

Doc #	Freq
2	1
1	1
1	1
2	1
1	1
1	1
2	1
2	1
1	1
1	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
1	1
2	1
2	1
1	2
2	2
1	1
1	1
2	1
1	2
2	2

Επίσης, οι περιορισμοί για το μέγεθος της κύριας μνήμης, σε συνδυασμό με το μέγεθος μιας συλλογής, μπορούν να οδηγήσουν σε πολύ μεγάλους χρόνους κατασκευής ενός αντεστραμμένου

αρχείου. Για αυτούς τους λόγους, πιο οικονομικές μέθοδοι κατασκευής και αντιστροφής του πίνακα συχνοτήτων πρέπει να χρησιμοποιηθούν και είναι αυτές που θα παρουσιαστούν στη συνέχεια.

ΠΑΡΑΜΕΤΡΟΣ	**ΣΥΜΒΟΛΟ**
Συνολικό μέγεθος κειμένου	B
Αριθμός εγγράφων	N
Αριθμός διακριτών λέξεων	n
Συνολικός αριθμός λέξεων	F
Αριθμός δεικτών ευρετηρίου	f
Τελικό μέγεθος συμπιεσμένου αντεστραμμένου αρχείου	l
Μέγεθος δομής δεδομένων δυναμικού ευρετηρίου	L
Χρόνος μετακίνησης (seek time) στο δίσκο	t_s
Χρόνος μεταφοράς στο δίσκο	t_r
Κωδικοποίηση αντεστραμμένου αρχείου/ byte	t_d
Χρόνος σύγκρισης και εναλλαγής για εγγραφές 10 byte	t_c
Χρόνος parsing, stemming και αναζήτησης για κάθε όρο	t_p
Διαθέσιμη κύρια μνήμη	M

Πίνακας 1

7.1 Τεχνικές Αντιστροφής Αρχείων

Το παρακάτω σχήμα παρουσιάζει εποπτικά τις διάφορες τεχνικές δημιουργίας αντεστραμμένων αρχείων:

	Χωρίς Συμπίεση	Με Συμπίεση
Χρήση μόνο **Κύριας Μνήμη**	Αντιστροφή στην Κύρια Μνήμη με χρήση **Συνδεδεμένων Λιστών** (Memory Based Linked List Inversion)	Αντιστροφή Ευρείας Μνήμης (Large Memory Inversion)
Χρήση **Κύριας Μνήμης** και Δίσκου	Αντιστροφή στο Δίσκο με χρήση **Συνδεδεμένων Λιστών** (Disk Based Linked List Inversion) Αντιστροφή μέσω **Διάταξης** (Sort Based Inversion)	Αντιστροφή με **Κατάτμηση Λεξικού** (Lexicon Based Partitioning) Αντιστροφή με **Κατάτμηση Κειμένου** (Text Based Partitioning) Διάταξη και Συμπίεση Πολλαπλή **Συγχώνευση** Πολλαπλή **In-place Συγχώνευση**

7.1.1 Αντιστροφή με χρήση Συνδεδεμένων Λιστών

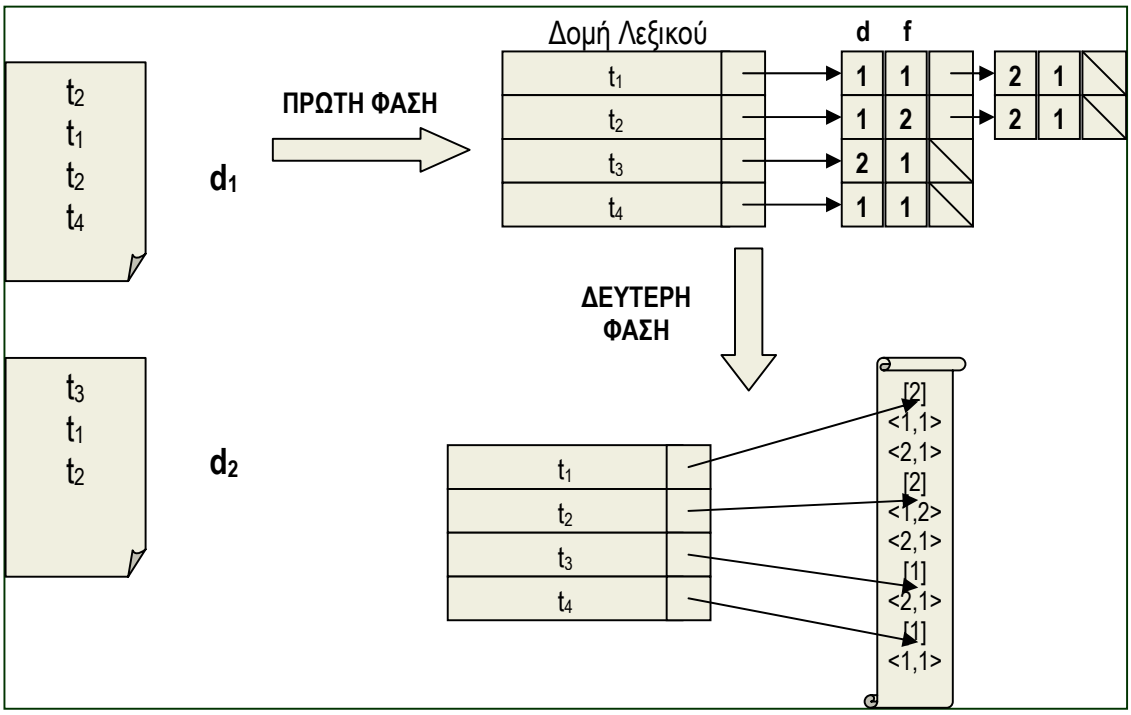
7.1.1.1 Αντιστροφή βασισμένη στη μνήμη (Memory-based Inversion)

Η διαδικασία της memory-based αντιστροφής έχει ως εξής: Αρχικά δημιουργείται η δομή δεδομένων για το λεξικό. Στη συνέχεια, για κάθε έγγραφο, αναζητείται κάθε όρος του εγγράφου στο λεξικό και προστίθεται ένας κόμβος, που περιέχει την πληροφορία για το έγγραφο και τη συχνότητα εμφάνισης, για τον κάθε όρο (μια προφανής υλοποίηση που εξασφαλίζει οικονομία χώρου και χρόνου είναι ένα hash table). Τελικά, δημιουργείται το αντεστραμμένο αρχείο, διασχίζοντας τη δομή του λεξικού και κατασκευάζοντας τη λίστα των όρων και των αντίστοιχων αριθμών των γραμμών. Η διαδικασία παρουσιάζεται λεπτομερώς στο σχήμα 1.

```
1. /* Αρχικοποίηση */
      Δημιούργησε μια άδεια δομή λεξικού S.
2. /* Φάση 1 – συλλογή των εμφανίσεων των όρων */
      Για κάθε κείμενο D_d στη συλλογή, 1≤d≤N
      (a) Διάβασε το D_d, αναλύοντας το σε όρους
      (b) Για κάθε όρο t∈D_d,
            i. Έστω f_{d,t} η συχνότητα εμφάνισης του t στο D_d
            ii. Ψάξε στο λεξικό S για το t
            iii. Αν δεν υπάρχει, εισήγαγέ το
            iv. Πρόσθεσε ένα κόμβο αποθηκεύοντας το <d, f_{d,t}> στη
λίστα που αντιστοιχεί στον t
```

```
3. /* Φάση 2 – έξοδος του αντεστραμμένου αρχείου */
      Για κάθε όρο 1≤t≤n
(a)   Άρχισε μια νέα καταχώρηση αντεστραμμένου αρχείου
(b)   Για κάθε <d, f_{d,t}> στη λίστα που αντιστοιχεί στο t, πρόσθεσε
το ζεύγος <d, f_{d,t}> στην καταχώρηση του αντεστραμμένου αρχείου
(c)   Αν χρειάζεται, συμπίεσε την καταχώρηση του αρχείου
(d)   Πρόσθεσε την καταχώρηση στο αντεστραμμένο αρχείο
```

Σχήμα 1 – Αντιστροφή βασισμένη στη Μνήμη



Σχήμα 2 – Παράδειγμα αντιστροφής βασισμένης στη Μνήμη

Ο χρόνος αντιστροφής μπορεί να υπολογιστεί ως εξής:

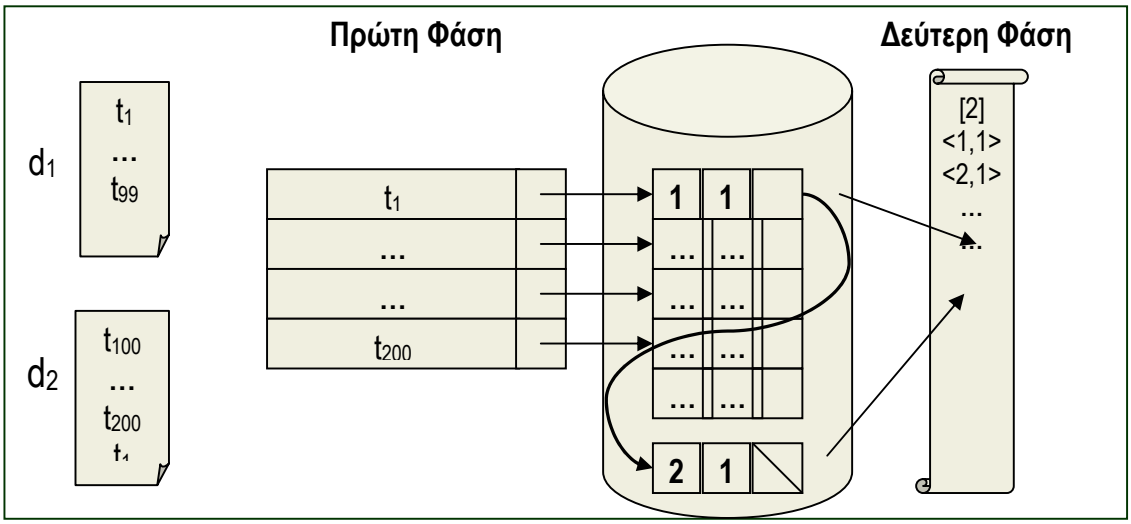
$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(διάβασμα και ανάλυση κειμένων)}$$

$$I \cdot (t_d + t_r) \quad \textit{(γράψιμο αντεστραμμένου αρχείου)}$$

Διαισθητικά, ο αλγόριθμος είναι βέλτιστος από άποψη χρόνου. Είναι αδύνατο να αντιστραφεί μία συλλογή χωρίς να χρειαστούν κάποιο από τα παραπάνω κόστη. Το μόνο πρόβλημα έγκειται στις μεγάλες απαιτήσεις σε μνήμη καθώς μετά το τέλος της πρώτης φάσης υπάρχει στην Κύρια Μνήμη ένα πλήρες ασυμπίεστο αντεστραμμένο αρχείο.

7.1.1.2. Αντιστροφή βασισμένη στο δίσκο (disk-based inversion)

Η τεχνική προσπαθεί να αντιμετωπίσει το πρόβλημα που ενδεχομένως να υπάρξει λόγω της μη ύπαρξης αρκετής μνήμης στο σύστημα και βασίζεται στη μεταφορά των συνδεδεμένων λιστών των αριθμών των εγγράφων από τη μνήμη στο δίσκο – ή ισοδύναμα στο τρέξιμο του προγράμματος σε μηχάνημα με τεράστια ιδεατή (virtual) μνήμη και λιγότερη φυσική μνήμη.



Σχήμα 3 – Παράδειγμα αντιστροφής βασισμένης στο δίσκο

. Ο χρόνος αντιστροφής είναι ίσος με

$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(διάβασμα, ανάλυση, γράψιμο αρχείου)}$$

$$f \cdot t_s + 10 f \cdot t_r + \quad \textit{(διάσχιση λιστών στο δίσκο)}$$

$$I \cdot (t_d + t_r) \quad \textit{(γράψιμο αντεστραμμένου αρχείου)}$$

Συμπερασματικά η τεχνική αντιστροφής με χρήση συνδεδεμένων λιστών απαιτεί πολύ χρόνο, πολύ χώρο, και δεν αξιοποιεί την κύρια μνήμη, συνεπώς για μεγάλες συλλογές η χρήσή της είναι ανεπαρκής γιατί είτε απαιτεί πολύ χρόνο είτε πολλή μνήμη. Αποτελεί όμως ιδανική λύση για μικρές συλλογές.

7.1.2. Αντιστροφή βασισμένη στην ταξινόμηση (sort based inversion)

Το κύριο πρόβλημα των παραπάνω μεθόδων είναι ότι απαιτούν πολλή κύρια μνήμη, και χρησιμοποιούν πρόσβαση στα δεδομένα που είναι ουσιαστικά τυχαία, αποτρέποντας μια αντιστοίχιση από τη μνήμη στο δίσκο. Η σειριακή προσπέλαση στο δίσκο είναι η μόνη αποτελεσματική υπολογιστική μέθοδος για μεγάλα αρχεία δίσκου αφού οι ρυθμοί μετάδοσης είναι συνήθως υψηλοί ενώ τα τυχαία ψαξίματα χρονοβόρα. Ακόμα η χρήση του δίσκου είναι αναπόφευκτη για μεγάλες συλλογές αρχείων, και έτσι ο αλγόριθμος αντιστροφής πρέπει να εκτελεί σειριακή επεξεργασία σε οποιαδήποτε αρχεία στο δίσκο. Όλα τα παραπάνω οδηγούν σε

ένα άλλο αλγόριθμο αντιστροφής: μια αντιστροφή βασισμένη στην ταξινόμηση (sort based), που αντιστοιχεί στην τρίτη γραμμή του συγκεντρωτικού πίνακα.

Ο αλγόριθμος περιγράφεται στο σχήμα 2 και λειτουργεί ως εξής:

Στο βήμα 2, το κείμενο αναλύεται σε όρους και προσωρινό αρχείο δημιουργείται. Το προσωρινό αρχείο αποθηκεύει μια τριπλέτα $< t,d,f_{d,t} >$ για κάθε συνδυασμό του αριθμού όρου t και του εγγράφου d όπου εμφανίζεται ο όρος. Το αντεστραμμένο αρχείο κατασκευάζεται ταξινομώντας αυτές τις τριπλές σε αύξουσα σειρά ως προς t. Συλλέγονται έτσι όλες οι καταχωρήσεις από κάθε όρο μαζί και συνεπώς η αντεστραμμένη λίστα για αυτό τον όρο μπορεί να δημιουργηθεί σαρώνοντας σειριακά την (ταξινομημένη) λίστα.

1. /* Αρχικοποίηση */

Δημιούργησε μια άδεια δομή λεξικού S.

Δημιούργησε ένα άδειο προσωρινό αρχείο στο δίσκο.

2. /* Επεξεργασία του κειμένου και γράψιμο του προσωρινού αρχείου */

Για κάθε έγγραφο D_d στη συλλογή, $1 \leq d \leq N$

(a) Διάβασε το D_d, αναλύοντας το σε όρους

(b) Για κάθε όρο $t \in D_d$,

i. Έστω $f_{d,t}$ η συχνότητα εμφάνισης του t στο D_d

ii. Ψάξε στο λεξικό S για τον όρο t

iii. Αν δεν υπάρχει, εισήγαγε τον όρο

iv. Γράψε μία τριπλέτα $<t, d, f_{d,t}>$ στο προσωρινό αρχείο, όπου ο όρος t αναπαρίσταται από τον αριθμό του όρου του στο S.

3. /* Εσωτερική ταξινόμηση για τη δημιουργία ομάδων */

Έστω k ο αριθμός των εγγραφών που χωρούν στη μνήμη

(a) Διάβασε k εγγραφές από το αντεστραμμένο αρχείο

(b) Ταξινόμησε σε αύξουσα σειρά ως προς t, και για ίδιο t, σε αύξουσα σειρά ως προς d

(c) Γράψε την ταξινομημένη ομάδα ξανά στο προσωρινό αρχείο

(d) Επανέλαβε μέχρι να μην υπάρχουν άλλες ομάδες για ταξινόμηση

4. /* Συγχώνευση */

Ανά δυο συγχώνευσε τις ομάδες στο προσωρινό αρχείο μέχρι να υπάρχει μια ταξινομημένη ομάδα

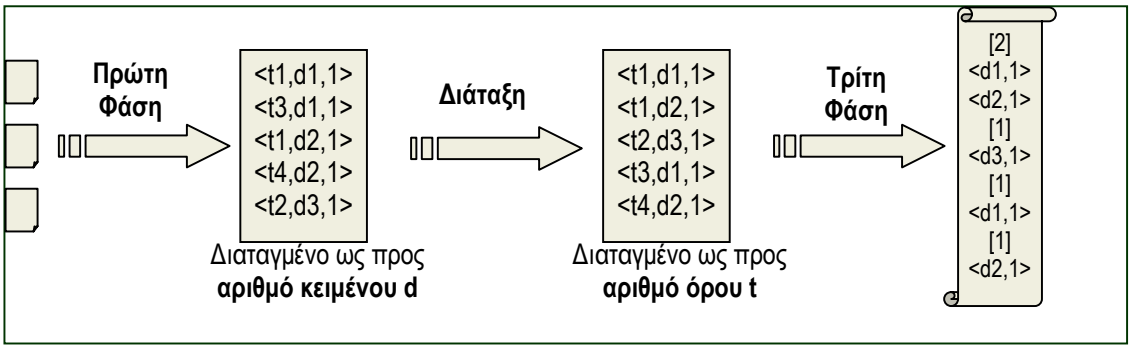
5. */ Έξοδος του τελικού αρχείου */

Για κάθε όρο $1 \leq t \leq n$

(a) Άρχισε μια νέα καταχώρηση αντεστραμμένου αρχείου

```
(b)  Διάβασε όλες τις τριπλέτες <t, d, f_{d,t}> από το προσωρινό
αρχείο και κατασκεύασε την καταχώρηση του αντεστραμμένου αρχείο για το
όρο t
(c)  Αν χρειάζεται, συμπίεσε την καταχώρηση του αρχείου
(d) Πρόσθεσε την καταχώρηση στο αντεστραμμένο αρχείο
```

Σχήμα 4 – Αλγόριθμος Αντιστροφής Βασισμένης Στην Ταξινόμηση



Σχήμα 5 – Αλγόριθμος Αντιστροφής Βασισμένης Στην Ταξινόμηση

Η ταξινόμηση, στο βήμα 3 του σχήματος 2, συμπληρώνεται με ένα εξωτερικό merge sort. Κατά τη διάρκεια της ταξινόμησης το προσωρινό αρχείο διαβάζεται σε μπλοκ των *k* εγγραφών, όπου *k* είναι ο μέγιστος αριθμός εγγραφών που μπορεί η κύρια μνήμη να διαχειριστεί. Αυτές οι εγγραφές είναι όλες του ίδιου μεγέθους, συνήθως περίπου 10 bytes, όπου 4 bytes δεσμεύονται για τον αριθμό του όρου, 4 bytes για τον αριθμό του εγγράφου και 2 bytes για την αποθήκευση της συχνότητας εμφάνισης του όρου στο έγγραφο. Για παράδειγμα, αν η διαθέσιμη μνήμη περιορίζεται στο 1 MB, το *k* θα είναι 100.000. Κάθε μπλοκ που διαβάζεται ταξινομείται σε αύξουσα σειρά ως προς *t*, και για ίδιο *t*, σε αύξουσα σειρά ως προς *d*. Μια in-memory τεχνική ταξινόμησης όπως ο QuickSort χρησιμοποιείται για αυτό το βήμα. Το μπλοκ στη συνέχεια γράφεται πίσω στο προσωρινό αρχείο ως μια ταξινομημένη «ακολουθία».

Η επόμενη φάση είναι να συγχωνευτεί ένας μεγάλος αριθμός ταξινομημένων ακολουθιών. Αυτό επιτυγχάνεται μετακινούμενοι κυκλικά μέσα στις ταξινομημένες ακολουθίες, συγχωνεύοντας την πρώτη με τη δεύτερη, την τρίτη με την τέταρτη, κ.ο.κ μέχρι ότου κάθε ακολουθία να έχει συγχωνευθεί μια φορά. Κάθε φάση συγχώνευσης μειώνει στο μισό τον αριθμό των ακολουθιών στο προσωρινό αρχείο και διπλασιάζει το μέγεθός τους. Αν υπάρχουν αρχικά *R* αρχικά ακολουθίες, μετά από $\lceil \log R \rceil$ φάσεις θα υπάρχει μία μόνο ακολουθία, και το αρχείο θα είναι ταξινομημένο. Αυτός είναι ο τυπικός τρόπος πραγματοποίησης μιας εξωτερικής συγχώνευσης – μέγιστου μεγέθους ομάδες κατασκευάζονται με εσωτερική ταξινόμηση και μετά συγχωνεύονται με σειρά γραμμικών περασμάτων– αυτός είναι ο τρόπος ταξινόμησης που καλείται όταν ένα μεγάλο αρχείο ταξινομείται.

Το τελικό βήμα, στην διαδικασία αντιστροφής, βήμα 5 στην εικόνα 2, είναι να διαβαστεί το ταξινομημένο προσωρινό αρχείο, δημιουργώντας το τελικό συμπιεσμένο αντεστραμμένο αρχείο. Ταξινομώντας το προσωρινό αρχείο δίνει την αναγκαία σειρά στο αντεστραμμένο αρχείο:

εγγραφές ταξινομημένες σε αύξουσα σειρά ως προς *t*, και για ίδιο t, σε αύξουσα σειρά ως προς *d*. Όταν το αντεστραμμένο αρχείο γραφτεί, ο χώρος του προσωρινού αρχείο αποδεσμεύεται.

Μελετώντας τώρα τη συνολική χρονική πολυπλοκότητα, μπορούμε να υπολογίσουμε το κόστος της φάσης στην οποία το αρχείο διαβάζεται, αναλύεται και φιλτράρεται ως:

$$B \cdot t_r + F \cdot t_p + 10f \cdot t_r$$

Στη συνέχεια, το προσωρινό αρχείο ταξινομείται. Ο Quicksort που είναι ο πιο διαδεδομένος γρήγορος εσωτερικός αλγόριθμος ταξινόμησης απαιτεί περίπου 1.2klogk συγκρίσεις και μεταθέσεις για να ταξινομήσει k αντικείμενα . Συνεπώς, με τους όρους του μοντέλου που εισαγάγαμε ο χρόνος της φάσης είναι:

$$20f \cdot t_r + R(1.2\log k)\, t_c$$

όπου $\kappa = M/10$ είναι ο αριθμός των εγγραφών που μπορούν να αποθηκευτούν στην κύρια μνήμη και $R = \lceil f/k \rceil$ είναι ο αριθμός των ακολουθιών.

Στο τέλος πραγματοποιείται το τμήμα της ταξινόμησης που αφορά τη δευτερεύουσα μνήμη. Οι ομάδες ταξινομούνται σε ζευγάρια και σε κάθε πέρασμα όλο το προσωρινό αρχείο διαβάζεται και γράφεται. Χρόνος επεξεργασίας απαιτείται επίσης και κατά της διάρκεια της συγχώνευσης – μια σύγκριση ανά τριπλέτα- αλλά αυτός ο χρόνος είναι μια τάξη μεγέθους μικρότερος από το χρόνο γραψίματος-διαβάσματος του αρχείου. Ο ακριβής χρόνος ταξινόμησης εξαρτάται από την ακριβή διάταξη των προσωρινών αρχείων στους δίσκους και το βαθμό στον οποίο η μέθοδος συγχώνευσης παρέχει επικάλυψη της εισόδου, της εξόδου και της διαδικασίας συγχώνευσης. Γενικά, ο χρόνος για αυτή τη φάση είναι:

$$\lceil \log R \rceil (20f \cdot t_r + f \cdot t_c)$$

όπου *R* είναι ο αριθμός των ομάδων.

Τελικά, το προσωρινό αρχείο διαβάζεται ξανά, συμπιέζεται και γράφεται στο δίσκο και το κόστος για τη φάση αυτή είναι:

$$10f \cdot t_r + I(\cdot t_d + t_r)$$

Συνολικά, με βάση το υπολογιστικό μοντέλο που χρησιμοποιούμε ,ο χρόνος είναι:

$$\begin{aligned} T = {} & B \cdot t_r + F \cdot t_p + 10f \cdot t_r + && \textit{(διάβασμα και ανάλυση, γράψιμο αρχείου)} \\ & 20f \cdot t_r + R(1.2\log k)\, t_c + && \textit{(ταξινόμηση ακολουθιών)} \\ & \lceil \log R \rceil (20f \cdot t_r + f \cdot t_c) + && \textit{(συγχώνευση ομάδων)} \\ & 10f \cdot t_r + I(\cdot t_d + t_r) && \textit{(γράψιμο συμπιεσμένου αντεστραμμένου αρχείου)} \end{aligned}$$

Τέλος πρέπει να λάβουμε υπ' όψιν μας τον χώρο στον δίσκο Δυστυχώς, ο αλγόριθμος ταξινόμησης που περιγράψαμε απαιτεί δυο αντίγραφα των δεδομένων κάθε στιγμή. Λόγου χάρη αν θεωρήσουμε την περίπτωση στα μισά της τελευταίας συγχώνευσης υπάρχουν δυο ακολουθίες για συγχώνευση, κάθε μία με μέγεθος το μισό του αρχικού αρχείου. Ακόμα, στα μισά της τελευταίας συγχώνευσης, και οι δυο ομάδες έχουν καταναλωθεί μερικώς. Εξαιτίας αυτού, είναι απίθανο για τη συγχωνευμένη έξοδο να γραφτεί σειριακά πίσω στο ίδιο αρχείο αφού θα μπορούσε να γράψει πάνω σε δεδομένα που δεν έχουν επεξεργαστεί ακόμα. Στο τελευταίο σημείο, ακριβώς πριν τελειώσει η συγχώνευση, το αρχείο εξόδου περιέχει όλες του τις εγγραφές ταξινομημένες όπως επίσης και τα δυο αρχεία εισόδου.

Αυτό σημαίνει πως χρειάζονται δυο προσωρινά αρχεία. Η μεγάλη απαίτηση σε χώρο δίσκου έχει ως συνέπεια αυτή η αντιστροφή η βασισμένη σε ταξινόμηση να είναι η καλύτερη μέθοδος για μετρίου μεγέθους συλλογές (10 έως 100MB), να μην είναι όμως κατάλληλη για πραγματικά μεγάλες συλλογές της τάξης των GB.

7.1.3. Αντιστροφή βασισμένη στην ταξινόμηση (sort based inversion) με χρήση Συμπίεσης

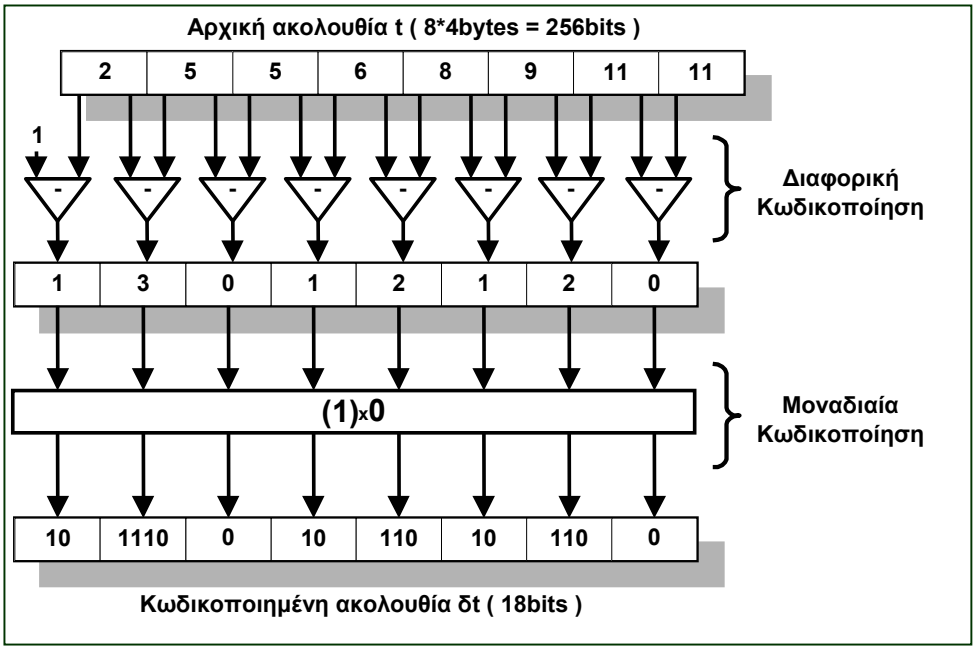
Υπάρχουν μία άμεση μεθοδολογία με την οποία αλγόριθμοι συμπίεσης μπορούν να μειώσουν τις απαιτήσεις της διαδικασίας αντιστροφής, και να οδηγήσουν σε ικανοποιητικές λύσεις αντιστροφής πολύ μεγάλων συλλογών. Πολύ απλά, το προσωρινό αρχείο που απαιτείται για την αντιστροφή με χρήση ταξινόμησης αποθηκεύεται συμπιεσμένο, κι έτσι κερδίζουμε χώρο στο δίσκο. Αυτή η προσέγγιση μελετήθηκε στην αναφορά [WMB99] όπου και παρουσιάστηκαν τρεις παραλλαγές.
Πριν προχωρήσουμε στην περιγραφή των τεχνικών αυτών είναι απαραίτητο να δώσουμε ορισμένα στοιχεία σχετικά με τη συμπίεση αντεστραμμένων λιστών.

Η βασική παρατήρηση που επιτρέπει τη συμπίεση των αντεστραμμένων λιστών είναι ότι κάθε αντεστραμμένη λίστα μπορεί, χωρίς βλάβη της γενικότητας, να θεωρηθεί ότι αποτελεί μία αύξουσα ακολουθία ακεραίων. Πιο γενικά η λίστα για τον όρο t αποθηκεύει τον αριθμό ft των κειμένων στα οποία ο όρος εμφανίζεται και μία λίστα από f_t αριθμούς κειμένων $<f_t; d_1, d_2, \ldots, d_{ft}>$, όπου $d_k<d_{k+1}$. Εφόσον η λίστα των αριθμών των κειμένων μέσα σε κάθε αντεστραμμένη λίστα είναι σε αύξουσα διάταξη και όλη η επεξεργασία είναι ακολουθιακή από την αρχή της λίστας η λίστα μπορεί να αποθηκευτεί ως μία αρχική θέση ακολουθούμενη από μία λίστα από d-χάσματα, τις διαφορές $d_{k+1}-d_k$.
Η κατάλληλη συμπίεση της αναφερόμενης ακολουθίας αριθμών απαιτεί την ύπαρξη μίας κατανομής πιθανοτήτων για τη μοντελοποίηση των μεγεθών των d-χασμάτων. Αυτές οι τεχνικές χωρίζονται δε δύο γενικές κατηγορίες: *global* (*ολικές*) και *local* (*τοπικές*) τεχνικές: στις ολικές τεχνικές κάθε αντεστραμμένη λίστα συμπιέζεται χρησιμοποιώντας το ίδιο κοινό μοντέλο, ενώ στις τοπικές μεθόδους το μοντέλο συμπίεσης προσαρμόζεται με βάση αποθηκευμένες παραμέτρους, με πιο συχνά χρησιμοποιούμενη την συχνότητα εμφάνισης ενός όρου.
Παράδειγμα τεχνικής κωδικοποίησης που χρησιμοποιείται και στα 2 μοντέλα είναι ο μοναδιαίος (unary) κώδικας. Σε αυτόν τον κώδικα ένας ακέραιος x>=1 κωδικοποιείται με x-1 bits με τιμή 1 ακολουθούμενο από bit με τιμή 0. Επίσης άλλη περίπτωση τέτοιων κωδικών είναι οι γ και δ κώδικες. Ο γ κώδικας αναπαριστά τον αριθμό x ως ένα μοναδιαίο κώδικα για την τιμή $1+\lfloor \log x \rfloor$ ακολουθούμενο από ένα κώδικα $\lfloor \log x \rfloor$ bits που αναπαριστά την τιμή $x-2^{\lfloor \log x \rfloor}$ στο δυαδικό σύστημα αρίθμησης. Ο δ κώδικας είναι παρόμοιος με τον γ κώδικα με την δαφορά ότι η τιμή $1+\lfloor \log x \rfloor$ αναπαρίσταται στο γ κώδικα και όχι στο μοναδιαίο κώδικα. Τέλος μία από τις πιο δημοφιλείς κατηγορίες μοντέλων συμπίεσης είναι τα Bernoulli μοντέλα εκ των οποίων ο πιο σημαντικός αντιπρόσωπος είναι ο κώδικας Golomb. Σύμφωνα με τον κώδικα αυτό για κάποια παράμετρο b, κάθε αριθμός $x>0$, κωδικοποιείται σε δύο μέρη: πρώτα το q+1 σε μοναδιαίο κώδικα, όπου $q=\lfloor (x-1)/b \rfloor$ και στη συνέχεια το υπόλοιπο r=x-qb-1, στο δυαδικό σύστημα αρίθμησης.

7.1.3.1. Συμπίεση Προσωρινών Αρχείων

Η βασισμένη στην ταξινόμηση αντιστροφή δεν είναι πρακτική γιατί απαιτεί τεράστιο χώρο στο δίσκο. Αλλά με σχετικά απλές μεθόδους συμπίεσης, όπως οι κώδικες γ και δ, το αντεστραμμένο αρχείο απαιτεί περίπου 1 byte για κάθε ζευγάρι $<d, f_t>$. Αφού το τελικό αρχείο μπορεί να συμπιεστεί κατά 6 φορές, αντίστοιχη συμπίεση μπορεί να γίνει και για το προσωρινό αρχείο με τις τριπλές $<t, d, f_t>$.

Ας δούμε πιο λεπτομερώς μια τέτοια λύση. Υπάρχουν τρεις τιμές που συνιστούν κάθε εγγραφή του προσωρινού αρχείου. Για τη συμπίεση των $d, f_{d,t}$ στοιχείων, οι χωρίς-παραμέτρους κωδικοποιήσεις (π.χ. unary, binary, γ, δ) έχουν το πλεονέκτημα της αποφυγής των δυο περασμάτων στο κείμενο. Υπάρχει μια πλεοναστικότητα στο όλο σχήμα, αλλά αυτή είναι μικρή. Μια οικονομική προσέγγιση για το t είναι η καλύτερη επιλογή και επιτυγχάνεται ως εξής:
Μέσα σε κάθε ταξινομημένη ακολουθία οι τιμές του t είναι αύξουσες. Η κωδικοποίηση με διαφορές είναι μια λογική επιλογή για την αναπαράσταση των τιμών του t. Κάθε t-χάσμα, δηλαδή η διαφορά μεταξύ των τιμών του t δυο διαδοχικών τριπλετών, είναι 0 ή παραπάνω. Ας υποθέσουμε ότι κάθε t-χάσμα του x αποθηκεύεται σαν η unary κωδικοποίηση του x+1, απαιτώντας x+1 bits. Εάν μία τριπλέτα έχει την ίδια τιμή του t με την προηγούμενη τριπλέτα, τότε το χάσμα είναι 0, και η κωδικοποίηση είναι 0. Αν το χάσμα είναι 1, τότε η κωδικοποίηση είναι το 10. Για χάσμα 2, η κωδικοποίηση είναι 110, κ.ο.κ. . Φυσικά στο τελικό αρχείο δε χρειάζεται η αποθήκευση των t-χασμάτων, αφού όλα είναι 0, εκτός από το πρώτο στοιχείο κάθε αντεστραμμένης λίστας, το οποίο είναι 1, και η δομή του λεξικού δείχνει πάντα που ξεκινά κάθε αντεστραμμένη λίστα. Τα στοιχεία t των τριπλών απαιτούνται μόνο στο προσωρινό αρχείο όπου κάθε ομάδα μπορεί να μην περιέχει όλους τους όρους.



Σχήμα 6– Αλγόριθμος Αντιστροφής μέσω Διάταξης και Συμπίεσης

Το επιπλέον κόστος για την αποθήκευση του t μ' αυτόν τον τρόπο είναι μικρό, αν λάβουμε υπ' όψιν ότι οι αρχικές ομάδες είναι σχετικά μεγάλες. Σε μια ταξινομημένη λίστα k τριπλετών, ο συνολικός αριθμός των bit που απαιτούνται για την αποθήκευση των χασμάτων του t, έχει όριο

το $k+n$, όπου n είναι ο αριθμός των διαφορετικών όρων. Αυτό προκύπτει από την παρατήρηση ότι το άθροισμα όλων των «t- χασμάτων +1» δεν υπερβαίνει το $k+n$. Παρόλα αυτά, αυτός ο υπολογισμός δεν είναι πλήρης. Επειδή όλες οι ακολουθίες είναι αποθηκευμένες και συμπιεσμένες, το μη ταξινομημένο προσωρινό αρχείο, δε μπορεί πραγματικά να αποθηκευτεί στο δίσκο, αφού η αναπαράσταση των χασμάτων του t, εξυπηρετεί μόνο τριπλέτες που είναι ήδη ταξινομημένες. Συνεπώς, η εσωτερική ταξινόμηση των ομάδων πρέπει να συνδυαστεί με την επεξεργασία του κειμένου. Μια συνέπεια της εκτέλεσης της εσωτερικής ταξινόμησης των ακολουθιών ταυτόχρονα με την ανάλυση του κειμένου και τη δημιουργία του λεξικού είναι ότι λιγότερος χώρος αποθήκευσης είναι διαθέσιμος για τη δημιουργία των ταξινομημένων ομάδων αφού κάποιος χώρος καταλαμβάνεται από την δομή δεδομένων S. Η στρωματοποιημένη δημιουργία των ακολουθιών λόγω της χρήσης της συμπίεσης, κάνει θεμιτή την αρίθμηση των όρων t με βάση την πρώτη τους εμφάνιση. Η πρώτη ακολουθία πρέπει να γραφτεί κατευθείαν στο δίσκο πριν επεξεργαστεί οσοδήποτε άλλο κείμενο, και αυτό δεν μπορεί να γίνει με ένα μόνο πέρασμα αν οι όροι ανατίθενται με λεξικογραφική σειρά για όλο το κείμενο.

Η χρήση συμπίεσης μειώνει τις απαιτήσεις σε χώρο σε σημαντικό βαθμό. Ο επιπλέον υπολογιστικός χρόνος που θα περίμενε κανείς είναι σχεδόν αμελητέος σε σχέση με τη μείωση των μεταφορών στο δίσκο. Έτσι, ο συνολικός χρόνος που οφείλεται στη συμπίεση είναι μικρός. Για την συγκεκριμένη τεχνική υπολογίζεται σε:

$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(διάβασμα και ανάλυση)}$$

$$R(1.2\log k)\, t_c + I'(t_d + t_r) + \quad \textit{(ταξινόμηση, συμπίεση και γράψιμο)}$$

$$\lceil \log R \rceil (2 I'(t_d + t_r) + f t_c) + \quad \textit{(συγχώνευση ομάδων)}$$

$$(I' + I)\ (t_d + t_r) \quad \textit{(γράψιμο συμπιεσμένου αντεστραμμένου αρχείου)}$$

Εδώ, $k=(M-L)/10$ είναι ο αριθμός των εγγραφών που υποστηρίζονται από τη μνήμη μετά το χώρο που απαιτείται για τη δομή S, $R=\lceil f/k \rceil$ είναι ο αριθμός των ακολουθιών και I' είναι το μέγεθος του προσωρινού αρχείου, συμπεριλαμβανομένου του επιπλέον κόστους αποθήκευσης των unary – κωδικοποιημένων χασμάτων του t και της διαρροής συμπίεσης.

7.1.3.2. Πολλαπλή συγχώνευση – Multiway merging

Η διαδικασία της συγχώνευσης, εξαρτάται κυρίως από υπολογισμούς στον επεξεργαστή και όχι από το δίσκο και συνεπώς μεγαλύτερη μείωση του χρόνου μπορεί να γίνει με τη χρήση πολλαπλής συγχώνευσης. Αυτή η προσέγγιση μπορεί να επεκταθεί και περισσότερο. Ας υποθέσουμε ότι όλες οι R ακολουθίες είναι καταγεγραμμένες σε ένα προσωρινό αρχείο, και στη συνέχεια λαμβάνει χώρα μια R-πλή συγχώνευση. Για την εκτέλεση μίας R-συγχώνευσης απαιτείται η χρήση μιας ουράς προτεραιότητας όπως ο σωρός που επιτρέπει στο μικρότερο στοιχείο ενός συνόλου από στοιχεία που αλλάζουν σπάνια, να βρίσκεται εύκολα. Κάθε μια από τις f τριπλέτες που αποτελούν το τελικό αρχείο περνά από μια τέτοια δομή δεδομένων με ένα κόστος $\lceil \log R \rceil$ συγκρίσεων και μετακινήσεων. Έτσι, ο χρόνος εκτέλεσης μειώνεται σε

$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(διάβασμα και ανάλυση, αρχείου)}$$

$$R(1.2\log k)\, t_c + I'(t_d + t_r) + \quad \textit{(ταξινόμηση, συμπίεση, γράψιμο)}$$

$$f \cdot \lceil \log R \rceil \cdot t_c + I'(t_s / b + t_r + t_d) + \quad (\textit{συγχώνευση})$$

$$I \ (t_d + t_r) \qquad (\textit{γράψιμο συμπιεσμένου αντεστραμμένου αρχείου})$$

δευτερόλεπτα, όπου b=M/R είναι ο χώρος στη μνήμη που δεσμεύεται για κάθε ακολουθία, και τα υπόλοιπα όπως πριν.

7.1.3.3. Πολλαπλή συγχώνευση επί τόπου (in place)

Κατά τη διάρκεια μιας *R*-συγχώνευσης, ένα μπλοκ των *b* bytes από κάθε ακολουθία βρίσκεται στη μνήμη, τροφοδοτώντας υποψηφίους για το σωρό. Στο ξεκίνημα της συγχώνευσης διαβάζεται το πρώτο μπλοκ κάθε ακολουθίας. Όταν η τελευταία τριπλέτα ενός μπλοκ πηγαίνει στο σωρό, ένα άλλο μπλοκ αντικαθιστά το προηγούμενο. Ας υποθέσουμε ότι το τελευταίο μπλοκ κάθε ακολουθίας είναι φτιαγμένο έτσι ώστε να είναι ακριβώς μέγεθος b bytes. Η προσέγγιση αυτή μεγαλώνει ελάχιστα το μέγεθος του προσωρινού αρχείου, αλλά έχει το σημαντικό πλεονέκτημα ότι κάθε συμπιεσμένη ομάδα καταλαμβάνει ένα ολοκληρωμένο αριθμό από μπλοκ.

Η συγχώνευση πραγματοποιείται λαμβάνοντας το επόμενο μικρότερο $<t, d, f_{d,t}>$ υποψήφιο από το σωρό και προσθέτοντάς το σε ένα μπλοκ εξόδου. Τα μπλοκ εξόδου μπορούν επίσης να περιοριστούν στα *b* bytes. Συνεπώς, δεδομένου ότι οι σελίδες του προσωρινού αρχείου μπορούν να γραφτούν σε τυχαία σειρά., κάθε μπλοκ εξόδου μπορεί να γραφτεί πίσω στο προσωρινό αρχείο σε οποιαδήποτε κενή θέση- σαν να ήταν το προσωρινό αρχείο μια σελιδοποιημένη μνήμη. Αυτό που χρειάζονται τα μπλοκ εξόδου είναι ένας πίνακας μπλοκ, που δείχνει τη θέση κάθε ταξινομημένου μπλοκ εξόδου. Αυτό απαιτεί μόνο μερικά byte για κάθε μπλοκ και μόνο ένα μικρό κομμάτι μνήμης απαιτείται για αυτό το σκοπό.

Στο ξεκίνημα της συγχώνευσης, υπάρχουν *R* κενά μπλοκ στο δίσκο που μπορούν να χρησιμοποιηθούν για την έξοδο, αφού αυτά τα μπλοκ έχουν ήδη αντιγραφεί στη μνήμη και παρέχουν τους πρώτους υποψήφιους για το σωρό. Αυτό σημαίνει ότι τα πρώτα *R* μπλοκ εξόδου μπορούν να γραφτούν πριν πιθανώς εξαντληθεί η ποσότητα των κενών μπλοκ. Αλλά αν έχουν γραφτεί *R* μπλοκ, τότε τουλάχιστον μια από τις ομάδες θα πρέπει να έχει ήδη αντικαταστήσει το πρώτο μπλοκ εισόδου της, και τότε θα υπάρχει τουλάχιστο ένα παραπάνω κενό μπλοκ διαθέσιμο. Αυτή η διαδικασία συνεχίζεται κατά τη διάρκεια της συγχώνευσης, μέχρι το τελευταίο μπλοκ κάθε ομάδας να το έχει επεξεργαστεί, και το το τελευταίο μπλοκ εξόδου να έχει γραφτεί τακτοποιημένα στην τελευταία κενή θέση. Σε κάθε στιγμή, για το χώρο που απαιτείται για ένα μπλοκ εξόδου, πρέπει να υπάρχει μια τουλάχιστον κενή θέση για να το κρατήσει.

Η χρησιμοποίηση unary κωδικοποίησης για τα *t*-χάσματα έχει το πλεονέκτημα της εξασφάλισης ότι δεν υπάρχει πιθανότητα οι συμπιεσμένες ακολουθίες να γίνουν μεγαλύτερες και στη διαδικασία εξόδου μπορούν να επανακωδικοποιηθούν με ασφάλεια σαν γ-κωδικοποιημένες f_t τιμές. Η μετατροπή κατά τη διάρκεια της συγχώνευσης μιας ακολουθίας κωδικοποιήσεων των *t*-χασμάτων σε μόνο μια *γ*-κωδικοποιήση δείχνοντας πόσα ζευγάρια αποθηκεύονται στην αντεστραμμένη λίστα είναι ασφαλές, επειδή η τελική κωδικοποίηση δε μπορεί να είναι μεγαλύτερη από την αρχική. Για παράδειγμα, ένας όρος που εμφανίζεται μια φορά, έχει τουλάχιστο 2 bit για το χάσμα του t, και απαιτεί μόνο ένα bit για γ-κωδικοποίηση. Ένας όρος που εμφανίζεται δυο φορές απαιτεί 3 bit για την κωδικοποίηση των t-χασμάτων, όσα και για τη γ-κωδικοποίηση, κ.ο.κ. . Όσο διαρκεί η αποκωδικοποίηση, η διαδικασία εξόδου καθυστερεί , αφού οι κωδικοποιήσεις των t-χασμάτων μικραίνουν και το «γέμισμα» στο τέλος των ομάδων αφαιρείται. Ένα αυξανόμενο χάσμα δημιουργείται μεταξύ της διαδικασίας διαβάσματος και γραψίματος των bit.

Είναι επίσης επιθυμητή σε αυτό το στάδιο η αποκωδικοποίηση των *d*-χασμάτων και των $f_{d,t}$ τιμών χρησιμοποιώντας τη μέθοδο που είναι πιο κατάλληλη για το τελικό συμπιεσμένο αντεστραμμένο αρχείο. Όλες οι παράμετροι ή οι ακριβείς κατανομές συχνοτήτων που

ενδεχομένως να χρειαστούν είναι ήδη διαθέσιμες αφού όλο το κείμενο έχει διαβαστεί ολόκληρο για μια φορά. Εντούτοις, αν οι αντεστραμμένες λίστες αποκωδικοποιηθούν, υπάρχει κίνδυνος για επέκταση του αρχείου – δηλαδή να υπάρχει μπλοκ προς εγγραφή αλλά όχι χώρος για να γραφτεί επειδή το αντίστοιχο μπλοκ εισόδου βρίσκεται ακόμα σε φάση επεξεργασίας. Αυτό το πρόβλημα προκύπτει επειδή δεν υπάρχει εγγυημένο μήκος, όταν π.χ. d-χάσματα κωδικοποιημένα σε δ-κώδικες αντικαθιστώνται από ισοδύναμες Golomb κωδικοποιήσεις. Ευτυχώς, η λύση είναι απλή. Όταν χρειαστεί, το αρχείο επεκτείνεται κατά ενα μπλοκ, και το μλοκ που είναι εκείνη την ώρα στη μνήμη γράφεται εκεί.

Στο τέλος της συγχώνευσης, το συμπιεσμένο αντεστραμμένο αρχείο είναι ψευδο-ταξινομημένο, αφού κάθε μπλοκ είναι ταξινομημένο και υπάρχει και ο πίνακας των μπλοκ, αλλά τα μπλοκ τα ίδια είναι σε λάθος σειρά. Για την αντιμετάθεση των μπλοκ έτσι ώστε το αρχείο να είναι ταξινομημένο, χρησιμοποιείται ο πίνακας των μπλοκ για διάταξη επιτόπου έτσι ώστε κάθε block να μετακινηθεί στη σωστή θέση. Αυτό μπορεί να γίνει σε γραμμικό χρόνο με την προυπόθεση ότι υπάρχουν δυο buffers των b bytes διαθέσιμοι, με κάθε μπλοκ να γράφεται και να διαβάζεται μια φορά το πολύ. Ο κώδικας για αυτό παρουσιάζεται στο σχήμα 7.
Αφού γίνει αυτή η διαδικασία, το αντεστραμμένο αρχείο μπορεί να περικοπεί από το προσωρινό στο τελικό μέγεθος, απελευθερώνοντας το μη χρησιμοποιημένο χώρο στο τέλος του αρχείου.

Ο πλήρης αλγόριθμος της αντιστροφής περιγράφεται στο σχήμα 8. .Οι τιμές $L=|S|$, k και R υπολογίζονται και ανανεώνονται συνεχώς, κι έτσι δεν υπάρχει λόγος για εκ των προτέρων ανάθεση τιμής σε αυτές. Το μόνο πρόβλημα είναι ότι είναι δύσκολη η επιλογή μιας τιμής για το μέγεθος του μπλοκ. Αυτό μπορεί να λυθεί αν δοθεί σε αυτό μια αρχική τιμή που να είναι πολλαπλάσια μιας μεγάλης δύναμης του 2, έτσι ώστε να μειώνεται στο μισό αν απαιτείται ενώ ακόμα να εξασφαλίζεται ότι οι ομάδες που έχουν ήδη γραφτεί παραμένουν ένα ακέραιο πολλαπλάσιο του νέου μεγέθους του μπλοκ.

```
      Αντιμετάθεση των μπλοκ έτσι ώστε η σελίδα i να περιέχει το μπλοκ
i, υποθέτωντας ότι αρχικά η i-οστή σελίδα περιέχει το μπλοκ
block_table[i]

      1. For  i=1 to  nblocks, if  i≠block_table[i]
      (a)         Διάβασε τη σελίδα i στη μνήμη
      (b)         Θέσε holding = block_table[i]
      (c)         Θέσε vacant=i
      (d)         While holding≠vacant do
                  i.       Ψάξε για j τέτοιο ώστε block_table[j]=vacant
                  ii.      Αντίγραψε τη σελίδα j στη σελίδα vacant
                  iii.     Θέσε block_table[vacant]=vacant
                  iv.      Θέσε vacant=j
      (e)         Γράψε το μπλοκ της μνήμης στη σελίδα vacant
      (f)         Θέσε το block_table[vacant]=holding
```

Σχήμα 7

Αυτή η διαίρεση στο μισό είναι αναγκαία για να εξασφαλιστεί ότι $R+1$ μπλοκ μπορούν να κρατηθούν στη μνήμη κατά τη διάρκεια της συγχώνευσης

1. /* Αρχικοποίηση */
   Δημιούργησε μια άδεια δομή λεξικού S.
   Δημιούργησεένα άδειο προσωρινό αρχείο στο δίσκο.
   Θέσε L=|S|
   Θέσε k=(M-L)/w, όπου w είναι ο αριθμός των bytes που απαιτούνται για την αποθήκευση μιας <t, d, $f_{d,t}$> εγγραφής
   Θέσε b=50 KB
   Θέσε R=0
2. /* Επεξεργασία του κειμένου και γράψιμο του προσωρινού αρχείου */
   Για κάθε έγγραφο D_d στη συλλογή, 1≤d≤N
   (a) Διάβασε το D_d, αναλύοντας το σε όρους
   (b) Για κάθε όρο t∈D_d,
      i. Έστω $f_{d,t}$ η συχνότητα εμφάνισης του t στο D_d
      ii. Ψάξε στην S για το t
      iii. Αν δεν υπάρχει,
   εισάγε το
   θέσε L=|S|

   θέσε k=(M-L)/w
   iv. Πρόσθεσε μια εγγραφή <t, d, $f_{d,t}$> στον πίνακα των τριπλετών.
   (c) Αν σε κάποιο σημείο, ο πίνακας των τριπλετών περιέχει k στοιχεία,
   i . Ταξινόμησε τον πίνακα των τριπλετών με Quick Sort
   ii. Γράψε τον πίνακα των τριπλετών, κωδικοποιώντας τα t-χάσματα σε unary, τα d-χάσματα με δ και τις $f_{d,t}$ τιμές σε unary.
   iii. Πρόσθεσε κάποιο γέμισμα για την συμπλήρωση του μπλοκ με b bytes
   iv. Θέσε R=R+1
   v. /* Μείωσε το μέγεθος του μπλοκ αν απειλείατι η κύρια μνήμη */
      Εάν b*(R+1)>M
         Θέσε b=b/2
3. /* Συγχώνευση */

```
(a) Διάβασε το πρώτο μπλοκ από κάθε ακολουθία και πρόσθεσε τον
αριθμό κάθε μπλοκ στην ελεύθερη λίστα
(b) Κατασκεύασε σωρό με R υποψηφίους, έναν από κάθε ομάδα
(c) Όσο ο σωρός δεν είναι κενός,
i. Διώξε τη ρίζα του σωρού
ii. Πρόσθεσέ τη στο μπλοκ εξόδου, επανακωδικοποιώντας τα d-
χάσματα και τις f_{d,t} τιμές
iii. Αντικατάστησε την από τον επόμενο υποψήφιο της ίδιας
ακολουθίας
(d) Κάθε φορά που γεμίζει το μπλοκ εξόδου
i. Χρησιμοποίησε την ελεύθερη λίστα για να βρεις ένα κενό μπλοκ
στο δίσκο. Αν δεν υπάρχει κενό μπλοκ στο δίσκο, πρόσθεσε ένα στο αρχείο
ii. Γράψε το μπλοκ εξόδου
iii. Ανανέωσε την ελεύθερη λίστα και τον πίνακα των μπλοκ
(e) Κάθε φορά που αδειάζει ένα μπλοκ εισόδου
i. Διάβασε το επόμενο μπλοκ της ίδιας ομάδας
ii. Ανανέωσε την ελεύθερη λίστα
4. /* Τοποθέτηση στις σωστές θέσεις */
Αναδιάταξε τα μπλοκ έτσι ώστε η φυσική σειρά να αντιστοιχεί
στη λογική χρησιμοποιώντας τον αλγόριθμο που γράψαμε παραπάνω
5. Μείωσε το μέγεθος του αρχείο στο τελικό του μέγεθος
```

Σχήμα 8

Ας υπολογίσουμε πόσος χώρος είναι διαθέσιμος για αυτά τα μπλοκ. Η μόνη επιπλέον απαίτηση στη μνήμη είναι η ανάγκη για τον πίνακα των μπλοκ. Αυτό απαιτεί μερικές λέξεις για κάθε μπλοκ αφού τα κενά μπλοκ πρέπει να συνδεθούν σε μια «ελεύθερη» λίστα. Παρόλα αυτά, οι συνολικές απαιτήσεις είναι χαμηλές.
Αναλυτικά, ο χρόνος που απαιτείται είναι

$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(διάβασμα και ανάλυση, αρχείου)}$$

$$R(1.2\log k)\, t_c + I'(t_d + t_r) + \quad \textit{(ταξινόμηση, συμπίεση, γράψιμο)}$$

$$f \cdot \lceil \log R \rceil \cdot t_c + I'(t_s / b + t_r + t_d) + \quad \textit{(συγχώνευση και επανακωδικοπίηση)}$$

$$2I \cdot (t_s / b + t_r) \quad \textit{(αντιμεταθέσεις μπλοκ)}$$

όπου k=(M-L)/10, R=⌈f/k⌉, b<M/(R+1) και I',είναι το μέγιστο μέγεθος του αντεστραμμένου αρχείου, με $I' \approx 1.35I$. Η τεχνική αυτή αντιστοιχεί στην έκτη σειρά του συγκεντρωτικού πίνακα.

7.1.4. Συμπιεσμένη – μέσα στη μνήμη – αντιστροφή

Στη συνέχεια παρουσιάζουμε τρεις ακόμα μεθόδους αντιστροφής. Τη –μέσα στη μνήμη- μέθοδο συμπίεσης και δύο ακόμα μεθόδους βασισμένες σε λεξικό. Η πρώτη μέθοδος, μπαίνει στην ίδια κατηγορία με την απλή τεχνική της αντιστροφής βασισμένης στην ταξινόμηση (sort-based), αποδεκτή για συλλογές μεσαίου μεγέθους αλλά όχι για μεγάλες συλλογές. Η τεχνική αυτή τροποποιείται για την περίπτωση όπου η μνήμη είναι περιορισμένη. Τότε, προκύπτουν δυο ακόμα τεχνικές τα χαρακτηριστικά των οποίων θα δούμε στη συνέχεια.

7.1.4.1. Αντιστροφή Ευρείας Μνήμης

Έστω ότι ένα μηχάνημα έχει πολύ μεγάλη μνήμη. Αν για κάθε όρο t είναι γνωστή η συχνότητα εμφάνισης f_t (δηλ. σε πόσα έγγραφα εμφανίζεται) όταν ξεκινά η αντιστροφή, ένας μεγάλος πίνακας στη μνήμη θα μπορούσε να δεσμευτεί ακριβώς στο σωστό μέγεθος, για την αποθήκευση της λίστας των αριθμών των εγγράφων d και των συχνοτήτων $f_{d,t}$. Αν τη συγκρίνουμε με την in-memory αντιστροφή που περιγράψαμε στην αρχή, έχουμε κέρδος σε χώρο με δυο τρόπους. Πρώτα, δεν υπάρχει λόγος για την ύπαρξη του «δείκτη επόμενου κόμβου» σε κάθε κόμβο στη λίστα. Δεύτερον, αν υποθέσουμε ότι ο αριθμός των εγγράφων N είναι γνωστός, τότε $f_t\lceil \log N \rceil$ bits μπορούν να δεσμευτούν για τον όρο t για τα d έγγραφα της αντεστραμένης λίστας, αντί του «ασφαλούς» $32f_t$ bits. Αν, για τον όρο t, η μέγιστη συχνότητα μέσα σε κάθε έγγραφο m_t, είναι γνωστή, τότε ο πίνακας των $f_{d,t}$ τιμών για τον όρο t, μπορεί να κωδικοποιηθεί με $f_t\lceil \log m_t \rceil$ bits Φυσικά, ο υπολογισμός των τιμών N, f_t, m_t απαιτεί ένα προκαταρκτικό πέρασμα σε όλη τη συλλογή κάτι που αποτελεί μία σημαντική επιβάρυνση κόστους.

Ας υποθέσουμε ότι μια τέτοια προσέγγιση είναι εφικτή. Ένας μοναδικός μεγάλος πίνακας του κατάλληλου μεγέθους δεσμεύεται για την αποθήκευση των ζευγαριών $<d,f_{d,t}>$, με μια καταχώρηση στο λεξικό για κάθε όρο έχοντας ένα δείκτη στον πίνακα δείχνοντας που πρέπει να αποθηκευτεί ο επόμενος αριθμός εγγράφου για αυτή τη λέξη. Στο δεύτερο πέρασμα έχουμε το εξής. Όλοι οι όροι έχουν αναγνωριστεί, έτσι μια μικρή μέθοδος κατακερματισμού μπορεί να σχεδιαστεί για το σύνολο των όρων, ελαχιστοποιώντας την ανάγκη για την αποθήκευση τους.

Αυτό που συμβαίνει ουσιαστικά σε αυτό το πέρασμα, είναι η δημιουργία ενός αντεστραμμένου αρχείου που κατασκευάζεται στην κύρια μνήμη με τυχαία προσπέλαση. Το αρχείο χρησιμοποιεί δυαδική κωδικοποίηση, χωρίς συμπίεση. Αυτό δίνει τη δυνατότητα της εφαρμογής μιας μεθόδου συμπίεσης (π.χ. κωδικοποίηση γ, δ). Το πρώτο πέρασμα που υπολογίζει τις τιμές f_t και m_t θα μπορούσε ακόμα να υπολογίσει για κάθε όρο t, τον συνολικό αριθμό των bit που απαιτούνται για την αποθήκευση της αντίστοιχης λίστας χρησιμοποιώντας μια άλλη κωδικοποίηση, π.χ. δ κωδικοποίηση για τα d-χάσματα και κωδικοποίηση γ ή unary για τα $f_{d,t}$ στοιχεία.

Ο χρόνος που απαιτεί αυτή η υλοποίηση υπολογίζεται παρακάτω:

$$T = B \cdot t_r + F \cdot t_p + \quad \textit{(πρώτο πέρασμα, διάβασμα και ανάλυση)}$$

$$B \cdot t_r + F \cdot t_p + 2I' \cdot t_d + I \cdot (t_d + t_r) \quad \textit{(δεύτερο πέρασμα, αντιστροφή)}$$

Στην περίπτωση αυτή $I' \leftarrow 1.05I$, λόγω της ακόμα υπάρχουσας διαρροής συμπίεσης.

Επίσης υπάρχει μια σειρά κωδικοποιήσεων που μειώνουν και σχεδόν εξαλείφουν τη διαρροή συμπίεσης. Η χρησιμοποίηση μιας τεχνικής ολικής (global) συμπίεσης απαιτεί τρία περάσματα της συλλογής εισόδου: ένα για τον υπολογισμό των συχνοτήτων και την κατασκευή του κώδικα Huffman, ένα δεύτερο για τον υπολογισμό των bit που χρειάζεται κάθε αντεστραμμένη λίστα και ένα τρίτο για την πραγματοποίηση της αντιστροφής μέσα στον πίνακα στη μνήμη. Αυτό

εξοικονομεί μερικά MB χώρο αλλά προσθέτει επιπλέον 5 ώρες χρόνο. Γενικά, το ίδιο ισχύει και για τις local μεθόδους κωδικοποίησης -απαιτούν τρία περάσματα της συλλογής εισόδου για την εξοικονόμηση μερικών MB. Παρόλα αυτά η local Bernoulli μέθοδος απαιτεί δυο περάσματα, γιατί είναι πιθανό σε ένα μόνο προκαταρκτικό πέρασμα να να συλλέξουμε τις αναγκαίες πληροφορίες για το χώρο που πρέπει να δεσμευτεί για κάθε αντεστραμμένη, συμπιεσμένη λίστα.

Ας δούμε λίγο την Golomb κωδικοποίηση. Η τιμή $b^A_t \approx 0.69(N/f_t)$ ελαχιστοποιεί το μέσο αριθμό των bit εξόδου για κάθε όρο, αλλά για την αντιστροφή μέσα στη μνήμη, πρέπει να υπολογιστεί ένα πάνω όριο χειρότερης περίπτωσης για τις τιμές του b αφού ο χώρος που δεσμεύεται δε μπορεί εύκολα να επεκταθεί αν η συμπιεσμένη αναπαράσταση είναι μεγαλύτερη απ' όσο περιμέναμε. Δεδομένων των τιμών N, f_t, b ο μέγιστος αριθμός bit που χρειάζεται για την Golomb κωδικοποίηση είναι

$$f_t(1+\lceil \log b \rceil) + \frac{N - f_t(2^{\lceil \log b \rceil} - b + 1)}{b}$$

Η τιμή του b που ελαχιστοποιεί τη χειρότερη περίπτωση» του αριθμού των bit είναι

$$b_t^W = 2^{\lfloor \log(N - f_t)/f_t \rfloor}$$

΄Ετσι, για να μειωθεί ο μέγιστος αριθμός των bit που μπορούν να καταναλωθούν, το b πρέπει να ληφθεί στη μέγιστη τιμή του 2 μικρότερη ή ίση με $(N-f_t)/f_t$.

Για αντιστροφή μέσα στη μνήμη κυρίαρχη σημασία έχει το μέγιστο όριο του αριθμού των bits, και για τη μείωση της χρήσης της μνήμης στο δεύτερο πέρασμα, χρησιμοποιείται η τιμή $b=b^W_t$. Φυσικά, αν ο μέγιστος αριθμός bit δεσμεύεται σε κάθε λίστα, κάθε συμπιεσμένη αναπαράσταση πιθανώς να είναι λίγο μικρότερη από το να γεμίσει το χώρο που δέσμευσε, κι έτσι θα έχουμε κάποια διαρροή. Αυτό είναι ανεκτό, με την προϋπόθεση ότι δε σπαταλιέται πολύς χώρος στο τέλος κάθε λίστας. Αυτό που δεν επιτρέπεται είναι κάποια λίστα να ξεχειλίζει.

Στη συνέχεια περιγράφεται η τεχνική που χρησιμοποιεί τα δύο περάσματα με κωδικοποίηση Golomb μέσα στη μνήμη. Το πρώτο πέρασμα υπολογίζει για κάθε όρο, τη συχνότητα εγγράφων f_t, την απόλυτη συχνότητα F_t και τις γράφει σε ένα αρχείο λεξικού. Για να ελαχιστοποιηθεί ο χώρος που απαιτείται για τη δομή των δεδομένων και για μεγιστοποίηση του χώρου που απαιτείται για το δεύτερο πέρασμα για τις αντεστραμμένες λίστες, η πρώτη φάση υπολογίζει μια ελάχιστη-ιδανική συνάρτηση κατακερματισμού για τους όρους της συλλογής.

Το δεύτερο πέρασμα διαβάζει το λεξικό και υπολογίζει για κάθε όρο την τιμή b^W_t, για να καθορίσει το μέγιστο αριθμό bits που απαιτούνται για τη Golomb κωδικοποίηση της αντερστραμμένης λίστας του όρου. Ακόμα, F_t bits προστίθενται για τη unary κωδικοποίηση των $f_{d,t}$ τιμών. Η πληροφορία αυτή χρησιμοποιείται για να δεσμεύσει τμήματα που σίγουρα θα είναι τέτοιου μεγέθους για κάθε συμπιεσμένη αντεστραμμένη λίστα. Στη συνέχεια, το κείμενο επεξεργάζεται ξανά, κατασκευάζοντας με τρόπο τυχαίας προσπέλασης ένα συμπιεσμένο αντεστραμμένο αρχείο μέσα στη μνήμη, χρησιμοποιώντας την Golomb κωδικοποίηση χειρότερης περίπτωσης και για κάθε όρο την αντίστοιχη τιμή b^W_t , με τις συχνότητες κωδικοποιημένες σε unary. Τελικά, στο τέλος του δεύτερου περάσματος, το αντεστραμμένο αρχείο μέσα στη μνήμη, αποσυμπιέζεται και επανασυμπιέζεται ξανά, για την κατασκευή του τελικού αρχείου. Αυτή η διαδικασία παρουσιάζεται λεπτομερώς στο σχήμα 9. Όπως και η βασισμένη στη ταξινόμηση αντιστροφή αυτή η τεχνική είναι κατάλληλη για μεσαίου μεγέθους συλλογές.

```
1.    /* Πρώτο πέρασμα - υπολογισμός των συχνοτήτων */
Δημιούργησε μια άδεια δομή λεξικού S.
```

Επεξεργάσου το κείμενο, υπολογίζοντας για κάθε όρο t

f_t: τον αριθμό των εγγράφων που περιέχουν τον t

F_t: τον συνολικό αριθμό των εμφανίσεων του όρου t.

Γράψε αυτές τις τιμές σε ένα αρχείο λεξικού.

2. */ Δεύτερο πέρασμα – κωδικοποίηση στη μνήμη */

Θέσε total=0

Διάβασε το αρχείο λεξικού

For t=1 to n do

(a) Θέσε $b_t = 2^{\lfloor \log(N-f_t)/f_t \rfloor}$

(b) Θέσε $B_t = f_t(1+\lceil \log b \rceil) + \frac{N - f_t(2^{\lceil \log b \rceil} - b + 1)}{b}$

(c) Θέσε $start_t = total$

(d) Θέσε $current_t = start_t$

(e) Θέσε $total = total + B_t + F_t$

3. Δέσμευσε ένα δείκτη των total bits.

4. */ Επεξεργάσου το κείμενο */

Για κάθε έγγραφο D_d στη συλλογή, για $1 \le d \le N$:

(a) Διάβασε το D_d, ανάλυσε το σε όρους

(b) Για κάθε όρο $t \in D_d$,

i. Έστω $f_{d,t}$ η συχνότητα του όρου t στο D_d

ii. Πρόσθεσε ένα d-χάσμα χρησιμοποιώντας κωδικοποίηση Golomb και μια $f_{d,t}$ τιμή κωδικοποιημένη σε unary, στη θέση $current_t$, ανανεώνοντας το $current_t$

5. /* Αποκωδικοποίηση, και επανακωδικοποίηση της εξόδου */

For t=1 to n do

(α) Χρησιμοποίησε το b_t για την αποκωδικοποίηση των f_t ζευγαριών στην καταχώρηση του αντεστραμμένου αρχείου για τον όρο t, αρχίζοντας από τη θέση $start_t$.

(b) Επανασυμπίεσε την καταχώρηση του αντεστραμμένου αρχείου χρησιμοποιώντας τον κώδικα συμπίεσης της εξόδου

(c) Πρόσθεσε αυτή την καταχώρηση στο αντεστραμμένο αρχείο.

Σχήμα 9

7.1.4.2. Τμηματοποίηση βασισμένη στο λεξικό

Για την επίτευξη αυτού του στόχου η αντιστροφή μοιράζεται σε μικρότερες εργασίες, κάθε μια από τις οποίες μπορεί να εκτελεστεί χρησιμοποιώντας ένα δείκτη μέσα στη μνήμη. Μια πιθανή διαίρεση είναι να κάνουμε πολλαπλά «δεύτερα» περάσματα στο κείμενο, που το καθένα θα επεξεργάζεται ένα φόρτωμα. Π.χ. αν οι υπολογισμοί με βάση το $b^{W}{}_{t}$ δείχνουν ότι η μνήμη που απαιτείται είναι τριπλάσια από τη διαθέσιμη, πραγματοποιούνται τρία «δεύτερα» περάσματα. Το πρώτο φόρτωμα επεξεργάζεται το πρώτο 1/3 του λεξικού, παραβλέποντας κάθε λέξη εκτός αυτού του εύρους, και γράφει το πρώτο 1/3 του αντεστραμμένου αρχείου. Έπειτα, ένα δεύτερο «δεύτερο» πέρασμα επεξεργάζεται το κείμενο, εξάγοντας το επόμενο 1/3 των λέξεων. Τελικά ένα ακόμα πέρασμα δημιουργεί το τελευταίο 1/3 του αντεστραμμένου αρχείου. Η ακριβής τμηματοποίηση του λεξικού μπορεί να εκτελεστεί στο τέλος του πρώτου περάσματος όταν οι τιμές f_t και F_t γίνουν γνωστές. Σ' αυτό το στάδιο, ο αριθμός των φορτωμάτων που απαιτούνται μπορεί να καθοριστεί και μπορούν να βρεθούν οι λέξεις-όρια. Στη συνέχεια παρουσιάζουμε δυο εκδοχές.

Αναλυτικά ο χρόνος που απαιτείται είναι

$T=B\ t_r + F\ t_p +$ (*διάβασμα και ανάλυση*)

$l(B\ t_r + F\ t_p) + 2\ I'\ t_d + I'(t_d + t_r)$ (*επεξεργασία των φορτωμάτων*)

όπου l είναι ο αριθμός των φορτωμάτων και $I'=0.5I$. Υποθέτωντας ότι το στατικό λεξικό δεύτερου περάσματος παίρνει το 1/3 του χώρου του δυναμικού λεξικού πρώτου περάσματος, τότε $I=\lceil I'/(M-L/3) \rceil$.
Ο χρόνος μπορεί να μειωθεί σημαντικά αν χρησιμοποιηθεί επιπλέον χώρος στο δίσκο. Τότε το κείμενο μπορεί να αναλυθεί μόνο δυο φορές, αφού ένα σύνολο από προσωρινά αρχεία μπορεί να γραφτεί από το δεύτερο πέρασμα, ένα για κάθε φόρτωμα. Οι εγγραφές σε κάθε προσωρινό αρχείο περιέχουν τον αριθμό ενός όρου t, ένα αριθμό εγγράφου d, μια συχνότητα $f_{d,t,}$ όπου όλες οι t τιμές είναι μέσα στο εύρος κάθε φορτώματος. Επιπλέον, αφού το πρώτο πέρασμα έχει ήδη καθορίσει τον αριθμό των εγγραφών σε κάθε προσωρινό αρχείο, μπορούν να ληφθούν σαν τμήματα του ίδιου μεγάλου αρχείου. Αυτό μας εξυπηρετεί σε περιπτώσεις συστημάτων όπου μόνο κάποιος αριθμός αρχείων μπορεί να επεξεργαστεί ταυτόχρονα.

Όταν οι εγγραφές έχουν μοιραστεί στα l προσωρινά αρχεία, κάθε φόρτωμα μπορεί να επεξεργαστεί, παίρνοντας τα δεδομένα από το αντίστοιχο προσωρινό αρχείο. Αυτό μειώνει το χρόνο εκτέλεσης σε

$T = B\ t_r + F\ t_p +$ (διάβασμα και ανάλυση)

$B\ t_r + F\ t_p + 10f\ (t_s/b + t_r) +$ (διάβασμα και γράψιμο προσωρινών αρχείων)

$10f + 2\ I't_d + I\ (t_d + t_r)$ (διάβασμα και επεξεργασία ομάδων)

δευτερόλεπτα, υποθέτοντας ότι κάθε $<t,d,f_{t,d}>$ απαιτεί 10 bytes και το b είναι ένα βολικό μέγεθος μπλοκ.

7.1.4.3 Τμηματοποίηση με βάση το κείμενο

Το κόστος σε χώρο της τμηματοποίησης βασισμένης στο λεξικό είναι μικρό υποθέτοντας ότι δε χρησιμοποιούνται επιπλέον προσωρινά αρχεία στο δίσκο, αλλά ο χρόνος είναι πολύ μεγαλύτερος από την «πολλαπλή συγχώνευση στη σωστή θέση». Παρόλα αυτά το μεγαλύτερο

τμήμα του χρόνου είναι για ανάλυση και επεξεργασία των δεδομένων. Ας υποθέσουμε ότι το κείμενο τμηματοποιείται από μόνο του και όχι από το λεξικό. Πρώτα, κατασκευάζεται ένα αντεστραμμένο αρχείο, για ένα σύνολο από έγγραφα, μετά για ένα δεύτερο και έτσι συγχωνεύουμε αυτά τα μερικά αντεστραμμένα αρχεία σε ένα τελικό αντεστραμμένο αρχείο.

Όταν κατασκευαστεί ένα αντεστραμμένο αρχείο μέσα στη μνήμη για ένα σύνολο εγγράφων, συγχωνεύεται με το πλήρες αντεστραμμένο αρχείο που κατασκευάζεται στο δίσκο. Για να γίνει αυτό, ολόκληρο το αρχείο διαβάζεται από το δίσκο, μπλοκ-μπλοκ, και ξαναγράφεται έχοντας προσθέσει τις νέες καταχωρήσεις. Αφού ο χώρος για τη λίστα κάθε όρου είναι δεσμευμένος από πριν στο δίσκο, το συγχωνευμένο αρχείο στο δίσκο μπορεί να ξαναγραφτεί στην ίδια θέση. Η μόνη επιπλέον απαίτηση είναι ένας δείκτης μεγέθους disk_current bit που πρέπει να μπορεί για κάθε όρο, για να δείχνει που πρέπει να προστεθούν οι εγγραφές των νέων «συνόλων των εγγράφων». Στο τέλος κάθε κειμένου, όταν όλα τα «σύνολα» θα έχουν αντιστραφεί και συγχωνευτεί, εκτελείται ένα τελικό πέρασμα στο αντεστραμμένο αρχείο, συμπιέζοντας τα αχρησιμοποίητα bit στο τέλος κάθε αντεστραμμένης λίστας, και πιθανώς επανακωδικοποιώντας και χρησιμοποιώντας μια άλλη κωδικοποίηση. Όταν τελειώσει αυτή η διαδικασία, μπορεί να απελευθερωθεί ο ελεύθερος χώρος στο τέλος.

Το πρώτο πέρασμα είναι τώρα υπεύθυνο για να αποφασίσει πόσα έγγραφα μπορούν να τοποθετηθούν σε κάθε «σύνολο εγγράφων». Για κάθε σύνολο γράφει ένα αρχείο πληροφοριών που καταγράφει τη συχνότητα κάθε όρου μέσα σε κάθε σύνολο έτσι οι αντιστροφές των «συνόλων» να χρησιμοποιούν τις ακριβείς συχνότητες όρων για την τμηματοποίηση των περιοχών τους στις αντεστραμμένες λίστες.

Αναλυτικά, ο συνολικός χρόνος είναι

$$
\begin{aligned}
T = \quad & B \cdot t_r + F \cdot t_p + && \textit{(διάβασμα και ανάλυση)} \\
& B \cdot t_r + F \cdot t_p + 3I' \cdot t_d + 2cI'(t_s / b + t_r) + && \textit{(αναστροφή επί τόπου)} \\
& (I' + I)(t_s / b + t_d + t_r) && \textit{(συμπίεση)}
\end{aligned}
$$

όπου $c=1'(M-L/3)$ είναι ο αριθμός των συνόλων εγγράφων και όπως πριν $I' \approx 1.05I$.

7.2. Συμπεράσματα

Οι αλγόριθμοι που είναι πιο κατάλληλοι για μεγάλες συλλογές είναι οι βασισμένοι στη ταξινόμηση (sort-based), οι «πολλαπλής συγχώνευσης επί τόπου» και η τεχνική «τμηματοποίησης βασισμένης στο λεξικό». Η τεχνική «τμηματοποίησης βασισμένης στο κείμενο» έχει το πλεονέκτημα ότι απαιτεί λιγότερο χώρο για τα προσωρινά αρχεία, και μπορεί αν χρειαστεί να λειτουργήσει με λιγότερη κύρια μνήμη, προσθέτοντας όμως επιπλέον χρόνο.

Η sort-based τεχνική έχει το πλεονέκτημα ότι απαιτεί μόνο ένα πέρασμα πάνω στο κείμενο. Το μοντέλο που χρησιμοποιήθηκε για να πάρουμε τα αποτελέσματα του συγκεντρωτικού πίνακα, περιέχει το κόστος διαβάσματος και ανάλυσης του κειμένου αλλά δεν περιέχει ένα σημαντικό δεδομένο – το χρόνο που απαιτείται για ην αποσυμπίεση του αρχικού κειμένου αν αποθηκεύεται συμπιεσμένο. Τυπικές υψηλής ταχύτητας μέθοδοι συμπίεσης αποσυμπιέζουν με ρυθμό 100 MB/min, έτσι κάθε πέρασμα πέρασμα κατά τη διάρκει της υποτιθέμενης αντιστροφής απαιτεί μια ακόμα ώρα χρόνο CPU για αποσυμπίεση. Έτσι η διαφορά μεταξύ των μεθόδων ενός περάσματος και των μεθόδων δυο περασμάτων θα είναι ακόμα μεγαλύτερη.

Αν και οι παραπάνω τεχνικές είναι οι μόνες κατάλληλες για μεγάλες συλλογές, οι υπόλοιπες έχουν και αυτές σημασία. Πολλές εφαρμογές απαιτούν την ευρετηριοποίηση 100 MB, και τότε, χρησιμοποιούνται οι προηγούμενοι αλγόριθμοι.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[BR99] R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley Longman Inc., 1999

[WMB99] I. Witten, A.Moffat, T. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishing, San Francisco, May 1999.

Καλογεράκης Παναγιώτης, Κρέτσης Αριστοτέλης, Νερατζίνης Αναστάσιος, κείμενα εργασιών στη Ανάκτηση Πληροφορίας

ΠΑΡΑΡΤΗΜΑ Β

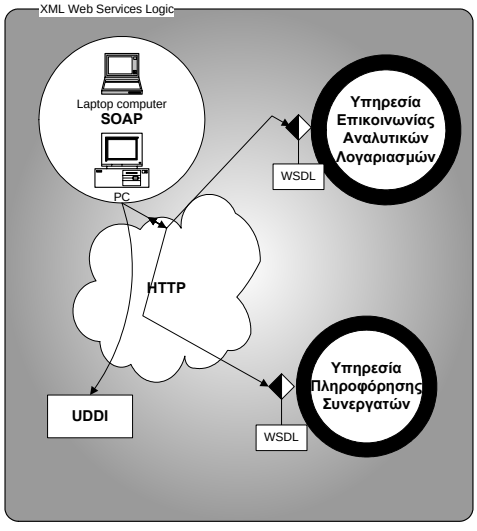
XML ΚΑΙ WEB SERVICES

Εισαγωγικά

Ο παγκόσμιος ιστός (World Wide Web) επέτρεψε την πρόσβαση σε ένα πλήθος από πηγές πληροφόρησης που παλαιότερα ήταν απροσπέλαστες και απομονωμένες. Κατά συνέπεια έχουν οικοδομηθεί τεχνολογίες για να επιτρέπουν την παροχή υπηρεσιών πληροφόρησης από διάφορους φορείς της οικονομίας που τις υιοθετούν, τόσο για εμπορικούς όσο και για σκοπούς προβολής και ενημέρωσης. Πρώτιστα η εφαρμογή αυτών των τεχνολογιών είναι αναγκαίο να επιδιώκει την αποδοτική παροχή των υπηρεσιών, ώστε να μην υπάρχουν καθυστερήσεις (bottlenecks) – λαμβανομένου υπόψη του συνεχώς αυξανόμενου όγκου πληροφορίας που καταγράφεται σε πληροφοριακά συστήματα. Επιπλέον θα πρέπει να διασφαλίζει μια ικανοποιητική ποιότητα στις προσφερόμενες υπηρεσίες (QoS), δηλαδή να εξασφαλίζει τη διάθεση των υπηρεσιών σε όλους τους χρήστες-συνδρομητές-πελάτες – και να εξαλείφει λειτουργικά σημεία που διαθέτουν «αχίλλειο πτέρνα» (single point failure).

Οι υπηρεσίες web (XML-Web Services) [WS] – ομάδες από σχετικές λειτουργίες και διευκολύνσεις που οι χρήστες μπορούν να προσπελάσουν και να διαχειριστούν μέσω του web – παρέχουν το κύριο μέσο για την ενεργοποίηση αυτής της ιδέας. Η αποδοτική διαμοίραση και δυνατότητα προσπέλασης υπηρεσιών web είναι ένα από τα σημαντικότερα βήματα για την εξάπλωση των επιχειρησιακών υπηρεσιών μέσω παγκόσμιου ιστού. Υπηρεσία web είναι ουσιαστικά κάθε μια υπηρεσία που είναι διαθέσιμη σε ένα εσωτερικό δίκτυο (intranet) ή στο Διαδίκτυο (Internet), η οποία χρησιμοποιεί ένα προτυποποιημένο XML σύστημα μηνυμάτων για την επικοινωνία μεταξύ πελάτη και υπηρεσίας. Οι υπηρεσίες web έχουν σχεδιαστεί ώστε να είναι ανεξάρτητες πλατφόρμας και γλώσσας προγραμματισμού. [Cerami02]. Οι υπηρεσίες web μπορούν να συνδεθούν με συγκεκριμένες πηγές πληροφορίας ή να είναι γενικής χρήσης (generic), ώστε να συνεργάζονται με ένα φάσμα από αυτές. Μπορούν να συναλλάσσονται με βάσεις δεδομένων ή άλλες υπηρεσίες. Στην πραγματικότητα πίσω από τις υπηρεσίες web «κρύβεται» μια σειρά από επιχειρησιακές και λειτουργικές διαδικασίες και πληροφορίες, ώστε με τρόπο διάφανο να διευκολύνεται η αποτελεσματική χρήση τους από ενδεχόμενους πελάτες.

Παρόλ' αυτά σε πολύπλοκα πληροφοριακά περιβάλλοντα, όπως για παράδειγμα τα συστήματα πληροφόρησης σύγχρονων τηλεπικοινωνιακών υπηρεσιών, απαιτείται η πρόσβαση σε πολλές διαφορετικές υπηρεσίες web που βρίσκονται σε πολλούς διαφορετικούς κόμβους ενός διαδικτυωμένου περιβάλλοντος. Υπάρχει η ανάγκη για ολοκληρωμένες λύσεις και «έξυπνους» μηχανισμούς που θα διαχειρίζονται και θα λειτουργούν απρόσκοπτα υπηρεσίες web. Ένα τέτοιο περιβάλλον απαιτεί αποδοτικές λύσεις για την περιγραφή, ανακάλυψη, επερώτηση, καταλογογράφηση, παρακολούθηση και διασφάλιση μιας σειράς από αυτόνομες και ετερογενείς υπηρεσίες web.



Σχήμα 1: Παράδειγμα Λογικής Υπηρεσιών Web – XML Web Services Logic

Οι τρόποι με τους οποίους οι οργανισμοί κοινής ωφέλειας, οι τράπεζες και οι επιχειρήσεις χρησιμοποιούν τον παγκόσμιο ιστό ως επιχειρησιακό εργαλείο ποικίλλει και διαφέρει. Πολλοί έχουν αρχίσει να ορίζουν τρόπους και μεθόδους, με τη βοήθεια των οποίων θα επιτραπεί στις εσωτερικές εφαρμογές τους να αλληλεπιδρούν με επιχειρηματικά περιβάλλοντα τρίτων, με χρήση της συνεχώς αναπτυσσόμενης υποδομής του δικτύου (Web). Κάθε εταιρεία επινοεί διαφορετική προσέγγιση, ανάλογα με την εμπειρία των προγραμματιστών (software architects), τις διαθέσιμες τεχνολογίες και το διαθέσιμο προϋπολογισμό. Η εξάπλωση των ολοκληρωμένων προσεγγίσεων και οι μοναδικές λύσεις έδωσαν ώθηση για τη δημιουργία και γέννηση μιας ολόκληρης βιομηχανίας, η οποία εστιάζεται στη γεφύρωση των μη συμβατών επίπεδων υπηρεσιών μεταξύ των ορίων των διαφόρων εταιρειών.

Πρόσφατη δουλειά στο επίπεδο περιγραφής πληροφορίας από το WWW Consortium (http://www.w3c.org), δίνει ελπίδες ότι η eXtensible Markup Language (XML) [XML] θα παίξει σημαντικό ρόλο στην απλοποίηση της ανταλλαγής επιχειρησιακών δεδομένων (business data) ανάμεσα στις εταιρείες. Επίσης, στα πλαίσια του W3C, η συνεργασία των βασικών εταιρειών της βιομηχανίας υπολογιστών με μικρές εταιρείες έντασης γνώσης έχουν δημιουργήσει ένα πρωτόκολλο που ονομάζεται SOAP [SOAP], το οποίο επιτρέπει σε ένα πρόγραμμα να ενεργοποιήσει προγραμματιστικές διεπαφές (interfaces) υπηρεσιών διαμέσου του διαδικτύου χωρίς να χρειάζεται μία κοινή γλώσσα προγραμματισμού ή κατανεμημένη υποδομή αντικειμένων. Χάρη σε αυτές τις τεχνολογίες και τα αναπτυσσόμενα πρότυπα, έχει γίνει ευκολότερη η προσέγγιση κάποιων από τα δυσεπίλυτα προβλήματα του παρελθόντος.

Η χρήση όμως των τεχνολογιών της XML και του SOAP δεν αρκεί για να υλοποιήσουν δύο οργανισμοί ή επιχειρήσεις μία υποδομή πληροφοριακής επικοινωνίας τόσο μεταξύ τους όσο και με τους πολίτες (communication infrastructure). Αυτό που απαιτείται είναι μία πλήρης από άκρο σε άκρο λύση, βασισμένη σε πρότυπα τα οποία υποστηρίζονται διεθνώς σε οποιαδήποτε υπολογιστική πλατφόρμα. Συνολικά χρειάζεται ακόμα αρκετή δουλειά για να επιτευχθεί ο στόχος αυτός. Οι UDDI [UDDI] προδιαγραφές έχουν κάνει ένα βήμα για να ορίσουν ένα επόμενο προς τα άνω επίπεδο το οποίο επιτρέπει σε δύο επιχειρήσεις να διαμοιράσουν (share) έναν τρόπο να

επερωτούν τις υπηρεσίες πληροφόρησης η μία της άλλης και να περιγράψουν τις δικές τους.

Υπάρχουσα Κατάσταση

Οι υπηρεσίες web αποτελούν ένα μοντέλο λειτουργίας που τα περισσότερα επιχειρησιακά περιβάλλοντα εφαρμόζουν ολοένα και περισσότερο. Οι υπηρεσίες web είναι πολύ λιγότερο αλληλο-εξαρτημένες από ένα παραδοσιακό κατανεμημένο σύστημα. Δύο υπηρεσίες web για να αλληλεπιδράσουν δυναμικά χρειάζεται να μπορούν να ικανοποιούν δύο τουλάχιστον βασικές απαιτήσεις. Πρώτα, μια υπηρεσία θα πρέπει να μπορεί να περιγράψει αφαιρετικά (abstract) τις διεπαφές που εξυπηρετεί και τις συνδέσεις που δέχεται, ώστε να μπορούν οι χρήστες της να μάθουν με ποιο τρόπο μπορούν να τη χρησιμοποιήσουν. Δεύτερο, οι χρήστες θα πρέπει να είναι δυνατό να ανακαλύψουν τις υπηρεσίες που τους ενδιαφέρουν και που τους καλύπτουν βέλτιστα σύμφωνα με κάποια κριτήρια αναζήτησης. Το πρώτο βήμα μπορεί να επιλυθεί με ικανοποιητικό τρόπο με τη χρήση της WSDL [WSDL].

Ωστόσο η τρέχουσα δυνατότητα για ανακάλυψη και συνδιαλλαγή με υπηρεσίες web πραγματοποιείται μόνο χρησιμοποιώντας συγκεντρωτικά - κεντρικοποιημένα σχήματα καταλόγων, όπως το UDDI [UDDI]. Η συγκεντρωτική αρχιτεκτονική του οδηγεί σήμερα σε ασταθή και μη αποδοτικά περιβάλλοντα (εμφανίζονται προβλήματα όπως: single point failure, performance bottleneck).

Συγκεκριμένα το Universal Description Discovery and Integration (UDDI) είναι μια πρωτοβουλία που ξεκίνησε από τη βιομηχανία της πληροφορικής και αποτελεί το μοναδικό διαθέσιμο πρότυπο (standard) στο είδος του. Επιτρέπει στις επιχειρήσεις να ανακαλύπτουν και να διαμοιράζονται πληροφορία σχετικά με υπηρεσίες web που είναι καταχωρημένες σε ένα

κατάλογο. Η λειτουργία του καταλόγου άρχισε ουσιαστικά στα μέσα του 2002 μαζί με την εμφάνιση των πρώτων υπηρεσιών web. Σήμερα, για να βρει κανείς μια υπηρεσία web απαιτείται η χρήση του UDDI και ο χρήστης χρειάζεται αρκετή πληροφόρηση σχετικά με την απαιτούμενη υπηρεσία προς ανακάλυψη, όπως λέξεις κλειδιά, μέρος του ονόματος της υπηρεσίας και υπομονή για να επιλέξει την κατάλληλη μέσα από τα αποτελέσματα του καταλόγου. Τα εργαλεία αναζήτησης και εντοπισμού που διατίθενται είναι πολύ απλοϊκά και δε λαμβάνουν υπόψη τυχόν συσχετίσεις μεταξύ υπηρεσιών web και τα ποιοτικά χαρακτηριστικά της κάθε μιας. Οι χρήστες σήμερα χρειάζεται να διατρέχουν συχνά εκατοντάδες εγγραφές για να εντοπίσουν την υπηρεσία web την οποία αναζητούν. Η διαδικασία αυτή είναι χρονοβόρα και σε πολλές περιπτώσεις χωρίς άμεσο αποτέλεσμα, αναγκάζοντας το χρήστη να επαναλάβει την αναζήτηση από την αρχή με νέες λέξεις κλειδιά. Στην επισκόπηση του ερευνητικού πεδίου (5.2.1) αναφέρεται αναλυτικότερα ο εντοπισμός του προβλήματος και οι περιορισμένες προσπάθειες που έχουν γίνει έως σήμερα για την αντιμετώπιση αυτού του νέου επιστημονικού προβλήματος.

Η τελική χρήση των υπηρεσιών web θα πρέπει να περιλαμβάνει επεκτάσιμους, ευέλικτους και εύρωστους μηχανισμούς αναζήτησης. Καθώς οι υπηρεσίες web βρίσκονται και λειτουργούν σε ένα πλήθος από διαφορετικά διαδικτυωμένα μηχανήματα (interconnected computers), απαιτείται να βρεθούν μηχανισμοί για την αποδοτικότερη διαχείρισή τους.

Η σημερινή παροχή προηγμένων υπηρεσιών στον τομέα της άμεσης ενημέρωσης σχετικά με υπηρεσίες κοινής ωφελείας, βασίζεται στην πλατφόρμα XML υπηρεσιών web (XML Web Services) [WS], που αποτελεί την κυρίαρχη τεχνολογία στον τομέα του web engineering. Στην περίπτωση της παροχής πληροφόρησης για υπηρεσίες που απαιτούν ανάκτηση πολύ μεγάλου όγκου πληροφορίας, ανήκουν οι πληροφορίες για τηλεπικοινωνιακές καταγραφές. Μπορεί εύκολα να αναλογιστεί κανείς ότι οι τηλεπικοινωνιακές κλήσεις είναι, με διαφορά, οι μεγαλύτερες σε αριθμό συναλλαγές μεταξύ πολίτη και υπηρεσίας κοινής ωφέλειας. Εξαιτίας του όγκου και της ευαισθησίας των δεδομένων της τηλεπικοινωνιακής πληροφορίας που καταγράφεται, τα δεδομένα αυτά αποτελούν ιδανική πλατφόρμα δοκιμών «έξυπνων» μηχανισμών για κάθε πληροφοριακό περιβάλλον παροχής υπηρεσιών κοινής ωφέλειας με υπηρεσίες web. Οι απαιτήσεις για τηλεπικοινωνιακές υπηρεσίες που απευθύνονται στον πολίτη/συνδρομητή/πελάτη, κυμαίνονται στην ανάλυση κίνησης λογαριασμών, την ενημέρωση χρέωσης, στην ενημέρωση για νέα πακέτα συνδρομών και συνολικότερα στην έγκαιρη και πληρέστερη πληροφόρηση τους για τις διαθέσιμες υπηρεσίες. Η πολυπλοκότητα αντιμετώπισης των επιχειρησιακών αναγκών, αναμένεται επίσης να αυξηθεί σημαντικά λαμβάνοντας υπόψη τα εξής:

- Εισαγωγή νέων υπηρεσιών και επέκταση του εύρους των παρεχομένων υπηρεσιών από τους επιχειρηματικούς φορείς και οργανισμούς, που μεταφράζεται σε ταυτόχρονη εισαγωγή πιο σύνθετων κριτηρίων στην αναζήτηση υπηρεσιών web στα επιχειρησιακά τους περιβάλλοντα.
- Άνοιγμα των παροχέων υπηρεσιών σε νέες αγορές - εξυπηρέτηση νέων χωρών, συνεργασία με νέους φορείς, εισαγωγή περισσότερων πληροφοριακών συστημάτων-, που αναμένεται να πολλαπλασιάσει τον όγκο των καταγραφόμενων δεδομένων.
- Εισαγωγή κριτηρίων ποιότητας στην παρεχόμενη υπηρεσία, (Quality of Service) η οποία αναφορικά μπορεί να σημαίνει γρηγορότερη έκδοση αποτελεσμάτων ή μεγαλύτερη πιστότητα στην αναζήτηση υπηρεσιών.

Πρότυπα

Οι υπηρεσίες web (XML Web Services) είναι μια τεχνολογία που αναπτύχθηκε πολύ πρόσφατα. Ορισμένες τεχνολογίες από τις πιο γνωστές προ – υπηρεσιών web εποχή είναι το EDI –

Electronic Data Interchange, CORBA - Common Object Request Broker Architecture, DCOM - Distributed Component Object Model, URPC - Unix Remote Procedure Call, and Java RMI - Java Remote Method Invocation. Το EDI ήταν δύσκολο να υλοποιηθεί εξαιτίας της πολυπλοκότητας και του κόστους. Το πρόβλημα με την CORBA και τα DCOM είναι ότι εμφανίστηκαν παράλληλα και ανταγωνιστικά και επιπλέον ήταν σχετικά δύσκολα για τους προγραμματιστές. Αυτό οδήγησε και τα δύο σε μικρή υποστήριξη από τη βιομηχανία. Το Unix RPC δε χρησιμοποιήθηκε ποτέ ευρέως σε περιβάλλοντα μη Unix και για το λόγο αυτό δε μπόρεσε να αποκτήσει ευρύτερο μερίδιο στην αγορά. Η τεχνολογία του Java RMI ξεκίνησε θετικά αλλά το μέλλον είναι αβέβαιο μιας και έχασε την υποστήριξη από ορισμένες εταιρείες πληροφορικής όπως η Microsoft. Οι τεχνολογίες αυτές υπάρχουν και σήμερα αλλά δε μπόρεσαν να κερδίσουν σημαντικό μερίδιο εξαιτίας πολυπλοκότητας, ακαμψίας, έλλειψης βιομηχανικής υποστήριξης και θεμάτων συμβατότητας. Αντίθετα οι υπηρεσίες web επιτρέπουν στους προγραμματιστές να χρησιμοποιούν τέσσερα ανοικτά πρότυπα – standards του web: HTTP, SOAP, XML, και WSDL. Μια σύντομη και περιεκτική τους περιγραφή ακολουθεί:

1. HTTP - Hypertext Transfer Protocol – το standard πρωτόκολλο χρησιμοποιείται στο Port 80 ακόμη και πάνω από συσκευές ασφαλείας και είναι υπεύθυνο για αναμετάδοση και ανταλλαγή δεδομένων μέσω Διαδικτύου - Internet.
2. SOAP - Simple Object Access Protocol – είναι ένα πρωτόκολλο που προέρχεται από την XML και περιλαμβάνει ένα σύνολο κανόνων για περιγραφή δεδομένων και διαδικασιών.
3. XML - Extensible Markup Language – αποτελεί τη βασική γλώσσα ετικετών που υποστηρίζει την αυστηρή συγγραφή και αποθήκευση πληροφοριών.
4. WSDL - Web Services Description Language – μια μέθοδος βασισμένη σε XML που χρησιμοποιείται για την ταυτοποίηση υπηρεσιών web καθώς και για την προσπέλασή τους κατά το χρόνο εκτέλεσης.

Επιπλέον το πρότυπο UDDI (Universal Description, Discovery and Integration), προδιαγράφει τα χαρακτηριστικά που πρέπει να πληροί ένας κατάλογος (registry), ο οποίος έχει καταχωρημένες Web Services, περιέχοντας πληροφορία για την λειτουργικότητά του, για τον παροχέα τους και τον τρόπο κλήσης τους. Η τρέχουσα έκδοση του προτύπου είναι η 3.0. Το πρότυπο WSDL περιγράφει τα χαρακτηριστικά μιας Web Service και χρησιμοποιείται επιπλέον στην καταχώρηση της Web Service αυτής στον κατάλογο UDDI.

Αποθήκευση XML πληροφορίας

Σε ότι αφορά την αποδοτική αποθήκευση XML αντικειμένων κειμένων, οι εργασίες που έχουν δημοσιευθεί, κατά κύριο λόγο συνδυάζουν γνωστές δομές δεδομένων για αποθήκευση συμβολοσειρών με δυναμικές δομές δεδομένων για δευτερεύουσα μνήμη. Οι Cooper et. al. [CSF+01] χρησιμοποιούν το λεγόμενο Index Fabric, μια δομή δεδομένων που ουσιαστικά αποτελεί συνδυασμός πολλών Patricia Tries [Morisson68] δομημένων κατά επίπεδα. Στο [KHY01] παρουσιάζεται μια δομή δυναμικής αποθήκευσης που αντιμετωπίζει αρκετά αποτελεσματικά, το πρόβλημα της δυναμικής αποθήκευσης XML κειμένων, όπου κατά τη διαδικασία ενημέρωσης, μπορεί να αλλάζουν οι συντεταγμένες της XML ετικέττας. Στην εργασία αυτή παρουσιάζεται η έννοια των σχετικών συντεταγμένων και συνδυασμός τους με τα B-δέντρα. Οι Li και Moon [LM01] παρουσιάζουν το σύστημα XISS, το οποίο χρησιμοποιείται στην αποθήκευση XML κειμένων και υποστηρίζει αποδοτικά την υποβολή ερωτήσεων με τη χρήση κανονικών εκφράσεων (regular expressions). Για την υποστήριξη αυτή, χρησιμοποιείται ένα καινοτόμο σχήμα αριθμοδότησης των εσωτερικών κόμβων ενός δέντρου XML ιεραρχίας και γίνεται συνδυασμός της αριθμοδότησης με ένα B-δέντρο. Για την αποδοτική υποστήριξη XML ερωτημάτων, στην εργασία των Chien et. al.[CVZ+02] χρησιμοποιούνται δύο αλγόριθμοι για την εκτέλεση structural joins που προκύπτουν κατά τη διάρκεια εκτέλεσης των ερωτημάτων, χρησιμοποιώντας B-δέντρα και R-δέντρα. Οι Wang et. al. [WSFY03], προτείνουν μια ακόμα μέθοδο, αποθήκευσης που υποστηρίζει ερωτήματα, τόσο στη δομή όσο και στο περιεχόμενο. Η

ιεραρχία που αντιπροσωπεύει ένα XML κείμενο, μετατρέπεται σε μια συμβολοσειρά βάσει της προδιάταξης (preorder) που παράγεται κατά τη διαπέραση της ιεραρχίας. Στη συνέχεια συνδυάζουν το B-δέντρο, με ένα Suffix tree [McCreight76] και με ένα δυναμικό σχήμα αρίθμησης των XML κόμβων και χτίζουν το Virtual Suffix Tree (ViST). Στα πειράματα που διεξήγαγαν, το ViST αποδείχθηκε πιο γρήγορο στην απάντηση ερωτημάτων, από τα XISS και Index Fabric

Ανακάλυψη XML πληροφορίας Υπηρεσιών Web (XML Web Services)

Η κύρια μέχρι τώρα χρήση των Web Services, αποτελείτο από την εμπλοκή υπηρεσιών από απόσταση με την αποστολή και τη λήψη μηνυμάτων. Η κατάσταση αυτή όμως δεν είναι αποδοτική και ο λόγος είναι οι ανάγκες που προτάσσονται από τις πολύπλοκες εφαρμογές, οι οποίες χρειάζονται αφενός να αποκτούν πρόσβαση σε πολύπλοκες Web Services και αφετέρου να μπορούν να επιλέξουν μέσα από μια πληθώρα παρεχόμενων Web Services με την ίδια λειτουργικότητα. Αυτή η ανάγκη έχει αναγνωριστεί πρόσφατα και στην επιστημονική κοινότητα.

Στην εργασία των Ouzzani και Bouguettaya [OB04], παρουσιάζεται μια αρχιτεκτονική τριών επιπέδων που σύμφωνα με τους συγγραφείς, είναι αρκετά ευέλικτη για να καλύψει τις ανάγκες για υποβολή πολύπλοκων ερωτημάτων αλλά και την παροχή ποιότητας των Web Services.

Στην εργασία των Schmidt και Parashar [SP04] παρουσιάζεται ένα σύστημα το οποίο παρέχει κατανεμημένη πρόσβαση σε Web Services μέσω αποθήκευσης των Web Services πάνω από ένα Peer-to-peer σύστημα όπως είναι το Chord. Επιπλέον το σύστημα αναθέτει μοναδικούς αριθμούς σε κάθε Web Service και την αντιστοιχίζει σε κάποιον κόμβο του P2P δικτύου. Το προτεινόμενο σύστημα έχει το πλεονέκτημα ότι υποστηρίζει γρήγορη αναζήτηση, με κόστος O(log2 N) μηνύματα και είναι δυναμικό γιατί υποστηρίζει την εισαγωγή και διαγραφή Web Services με κόστος O(log2 N) αριθμό μηνυμάτων, όπου N ο αριθμός των P2P κόμβων. Επιπλέον παρέχεται η δυνατότητα υποστήριξης πολύπλοκων ερωτημάτων με την ανάθεση των πεδίων ερώτησης σε διαστάσεις του χώρου και την περαιτέρω μεταφορά του προβλήματος σε μια διάσταση με τη χρήση καμπύλης Hilbert. Παρόμοια λογική αλλά για κάπως πιο περιορισμένη εκδοχή του προβλήματος, ακολουθείται και στην εργασία των Yi et. al. [LZW+04]

Το πρόβλημα της ανακάλυψης Web Services εξετάζεται και από τους Sajjanhar et. al.[SHZ04]. Συγκεκριμένα προτείνεται μια μέθοδος ευρετηριοποιήσης των Web Services βάσει των λέξεων κλειδιών που περιλαμβάνονται στην περιγραφή τους στο UDDI. Στη συνέχεια κάθε Web Service θεωρείται σαν κείμενο και χρησιμοποιείται η κλασσική μέθοδος του LSI για την ανάκτηση των πιο σχετικών Web Services με ένα ερώτημα, με εντυπωσιακά αποτελέσματα.

Σχεδιασμός, μοντελοποίηση & αρχιτεκτονική Υπηρεσιών Web

Σε ότι αφορά θέματα υλοποίησης της κατάλληλης αρχιτεκτονικής για Web Services, πάλι συναντάμε την εργασία των Ouzzani και Bouguettaya [OB04], οι στην οποία προτείνονται τα αρχιτεκτονικά στοιχεία που πρέπει να ακολουθούνται κατά το σχεδιασμό ενός συστήματος που θα επιτρέπει την αποτελεσματική πρόσβαση σε Web Services. Τα τρια επίπεδα που προτείνονται αποτελούνται από ένα επίπεδο το οποίο παρέχει τις δυνατότητες για την διατύπωση ερωτημάτων, ένα δεύτερο επίπεδο το οποίο μετατρέπει τα ερωτήματα σε υπαρκτές Web Services και επεξεργάζεται διαφορετικά σενάρια εκτέλεσης τους και το τρίτο επίπεδο που αντιστοιχεί στις οντότητες του πραγματικού κόσμου.

Αναγνωρίζοντας την ανάγκη για αποτελεσματικότερη οργάνωση των UDDI καταλόγων, προτείνεται στο [RS03] μια αρχιτεκτονική που προωθεί τη δημιουργία μιας ομοσπονδίας

(federation) από UDDI καταλόγους. Τα ερωτήματα προωθούνται στην ομοσπονδία όπως θα γινόταν σε έναν μόνο κατάλογο και υπάρχει ένα πρωτόκολλο που μπορεί να αποφασίσει για την περαιτέρω δρομολόγηση και αντιστοίχιση του ερωτήματος σε Web Service.

Στην εργασία των Vilas et. al. [VAV2004], προτείνεται μια αρχιτεκτονική για την υποστήριξη υψηλής διαθεσιμότητας και συσταδοποίησης (clustering) των παρεχόμενων υπηρεσιών, η οποία βασίζεται στην ιδέα του virtualization. Η τελευταία αντιστοιχεί στην δημοσίευση πολλών Web Services με το περιτύλιγμα μιας. Για την υλοποίηση αυτής της τεχνικής προτείνεται μια παραλλαγή της WSDL που ονομάζεται Virtual WSDL (VWSDL).

Ποιότητα υπηρεσίας Web (QoS)

Ζητήματα παροχής ποιότητας υπηρεσιών Web Services (Quality of Web Service – QoWS) έχουν μόλις πρόσφατα προκύψει σε ερευνητικές εργασίες. Αυτό που απασχολεί σε πρώτο στάδιο είναι η εκπόνηση κατάλληλων μετρικών που εγκλωβίζουν αυτό που πρακτικά θεωρείται καλή Web Service. Στο [Ran03] προτείνονται κάποιες μετρικές που θεωρούνται ότι σχετίζονται με την παροχή QoWS. Ενδεικτικά αναφέρονται παράμετροι όπως η Διαθεσιμότητα, ο Χρόνος Απόκρισης, η Ρυθμαπόδοση (Throughput), η Ακρίβεια σε ότι αφορά τα λάθη, η Ασφάλεια που παρέχεται κ.ο.κ. Στην [OB04] δίνονται επιπλέον κάποιες μετρικές και τρόποι υπολογισμού τους, ενώ επίσης επισημαίνεται ότι το προτεινόμενο μοντέλο, είναι ικανό για την επιλογή της κατάλληλης Web Service με βάση τα τεθέντα κριτήρια.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[Cerami02] Cerami, Ethan. (2002). Web Services Essentials. California: O'Reilly & Associates, Inc.

[CVZ+02] Shu-Yao Chien, Z.Vagena, D. Zhang, V. Tsotras and C. Zaniolo. " Efficient Stuctural Joins on Indexed XML documents". Proceedings of 28th VLDB Conference, Hong Kong, 2002.

[GSS+03] John Garofalakis, Evangelos Sakkopoulos, Spiros Sirmakessis, Athanasios Tsakalidis "Integrating Adaptive Techniques with Web Services", in the proceedings of IEEE International Conference on Information Technology: Coding & Computing, pp 415 419, April 28-30, 2003, Las Vegas, Nevada, USA.

[KHY01] D. Kha, M. Yokoshikava and S. Uemura." An XML indexing structure with relative region coordinate". The 17th IEEE International Conference on Data Engineering (ICDE2001), pp. 313-320, 2001.

[LM01] Q. Li and B.Moon. "Indexing and Querying XML data for Regular Path Expressions". of 27th VLDB Conference, pp. 361 – 370, Italy, 2001.

[LZW+04] Yin Li , Futai Zou, Zengde Wu, and Fanyuan Ma, PWSD: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network, In Proc. APWeb 2004, pp. 291-300, Lecture notes in Computer Science 3007, Springer Verlag, 2004.

[Morisson68] Morrison, D. R., Patricia: Practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM 15, 514–534, 1968.

[OB04] M. Ouzzani, A. Bouguettaya, Efficient Access to Web Services, IEEE Internet Computing, March/April 2004, pp. 34-44, 2004

[Ran03] Shuping Ran, A Model for Web Services Discovery with QoS, ACM, ACM SIGecom Exchanges, Volume 4 , Issue 1 Spring, 2003 pp. 1 - 10 .

[RS03] P. Rompothong, T. Senivongse, A Query Federation of UDDI Registries, In Proc. of the 1st international symposium on Information and communication technologies, Dublin, Ireland, SESSION: Communication technology II - Internet, services, and architectures pp. 561 - 566, 2003

[RST04] Maria Rigou, Spiros Sirmakessis, Athanasios Tsakalidis, "Integrating Personalization in E-learning Communities", to appear in the International Journal of Distance Education Technologies 2004.

[SHZ04] Atul Sajjanhar, Jingyu Hou, and Yanchun Zhang, Algorithm for Web Services Matching, In Proc. APWeb 2004, pp. 665-670, Lecture notes in Computer Science 3007, Springer Verlag, 2004.

[SK02] Evangelos Sakkopoulos, Athanasios Tsakalidis "Utilizing Complementary Know-How: Advanced Fine Arts meeting Information Technology to provide a Virtual University for artists - students and alumni", ACM SIGUCCS User Services conference 2002, Full Paper, November 22, 2002, Providence, Rhode Island, USA.

[SOAP] Simple Object Access Protocol Reference Site: http://www.w3.org/TR/SOAP

[SP04] Cr. Schmidt, M. Parashar, A Peer-to-Peer Approach to Web Service Discovery, World Wide Web: Internet and Web Information Systems, 7, pp. 211-229, 2004

[UDDI] UDDI Specifications http://www.uddi.org

[VAV2004] Julio Fernández Vilas, José Pazos Arias, and Ana Fernández Vilas, High Availability with Clusters of Web Services, In Proc. APWeb 2004, pp. 644-653, Lecture notes in Computer Science 3007, Springer Verlag, 2004.

[WSDL] Web Services Description Language Reference Site http://www.w3.org/TR/wsdl

[WSFY03] Haixun Wang, Sanghyun Park, Wei Fan and Philip S. Yu, "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures". In SIGMOD 2003, 12, 2003, San Diego, CA.

[WS] Web Services Activity Statement, WWW Consortium http://www.w3.org/2002/ws/Activity

[XML] Extensible Markup Language (XML) http://www.w3.org/XML/

[XMLS] XML Schema http://www.w3.org/XML/Schema

[URI] Naming and Addressing: URIs http://www.w3.org/Addressing