Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών & Δικτύων

Μέθοδος συμπίεσης συμβολοσειρών με σχεδόν βέλτιστο χώρο και βέλτιστο χρόνο ανάκτησης υποσυμβολοσειράς

Διπλωματική εργασία

Καλλίκης Μικές

Βόλος 2012

Επιβλέπων Καθηγητής: Κ. Παναγιώτης Μποζάνης
Αναπληρωτής Καθηγητής

Δεύτερος επιβλέπων: Κ. Δημήτριος Κατσαρός
Λέκτορας Τ.Μ.Η.Υ.Τ.Δ

Στην οικογένεια μου…

Περιεχόμενα

Περίληψη(Abstract)....................7

ΚΕΦΑΛΑΙΟ 1 – ΕΙΣΑΓΩΓΗ - ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

1.1 Αντικείμενο Αλγορίθμων....................8
1.1.1 Εισαγωγικά για τους Αλγορίθμους....................8
1.1.2 Γενική περιγραφή αλγορίθμου....................9

1.2 Ανάλυση αλγορίθμων....................9
1.2.1 Ασυμπτωτικός συμβολισμός....................10
1.2.1.1 Ο συμβολισμός Ο....................11
1.2.1.2 Ο συμβολισμός ο....................11
1.2.2 Μοντέλα υπολογισμού....................12
1.2.2.1Μηχανή Turing....................12
1.2.2.2 Μοντέλο Μηχανής τυχαίας προσπέλασης (RAM) 13
1.2.2.3 Ισοδυναμία RAM και μηχανής Turing....................14

1.3 Ανάλυση χειρότερης περιπτώσεως....................15

1.4 Ανάλυση μέσης περίπτωσης....................15

1.5 Ανάλυση Επιμερισμένης ή Κατανεμημένης Περίπτωσης ... 15
1.5.1 Μέθοδος Δυναμικού ... 16
1.5.2 Λογαριασμός Τραπεζίτη ή Λογιστική Μέθοδος ... 17

1.6 Ανάγκη για αποδοτικούς αλγόριθμος ... 18
Κεφάλαιο 2 - Συμπίεση δεδομένων – Αλγόριθμος Huffman ... 19

Κεφάλαιο 2 - Συμπίεση δεδομένων – Αλγόριθμος Huffman

2.1 Συμπίεση δεδομένων ... 19
2.1.1 Κατηγορίες συμπίεσης δεδομένων ... 20
2.1.1.1 Μη απωλεστική συμπίεση ... 20
2.1.1.2 Απωλεστική συμπίεση ... 20

2.2 Παραδείγματα μεθόδων συμπίεσης ... 21
2.2.1 Zip ... 21
2.2.2 Rar ... 21
2.2.3 JPEG ... 22
2.2.4 MPEG ... 22
2.2.5 MP3 ... 23

2.3 Αποσυμπίεση αρχείων ... 23

2.4 Αλγόριθμος συμπίεσης Huffman ... 24

2.4.1 Δυαδικό δέντρο T του αλγορίθμου Huffman 25
2.4.2 Αλγόριθμος Huffman σε ψευδοκώδικα 25
2.4.3 Σημερινή χρήση του αλγόριθμου 26

2.5 Αποσυμπίεση Huffman 26

Κεφάλαιο 3 - Αλγόριθμοι Ταξινόμησης – Burrow - Wheeler Transform

3.1 Ταξινόμηση δεδομένων 28
3.1.1 Αλγόριθμοι διάταξης δεδομένων 29

3.2 Burrow-Wheeler Transform 30
3.2.1 Εισαγωγή 30
3.2.2 Χρήση του BWT 31
3.2.3 Περιγραφή του BWT 31
3.2.4 Γιατί ο ταξινομημένος πίνακας συμπιέζεται καλύτερα .. 37
3.2.5 Βελτιώσεις του BWT 37
3.2.6 Μειονέκτημα του BWT 38

3.3 Θεωρήματα που προκύπτουν από την σύμπραξη των αλγορίθμων BWT και Huffman 38
3.3.1 Θεώρημα 1 39
3.3.2 Θεώρημα 2 39
3.3.3 Θεώρημα 3 39
3.3.4 Θεώρημα 4 40

3.4 Παράδειγμα της σύμπραξης των δύο αλγορίθμων BWT και Huffman……………………….…………………………..40

Συμπεράσματα (Conclusions)…………………………………………………..42

ΚΕΦΑΛΑΙΟ 4 – ΠΕΙΡΑΜΑΤΙΚΟ ΜΕΡΟΣ - ΜΕΤΡΗΣΕΙΣ

Περίληψη(Abstract)

Το αντικείμενο της παρούσης διπλωματικής εργασίας είναι η συμπίεση αρχείων με χρήση του αλγορίθμου Huffman μέσω του εργαλείου ταξινόμησης Burrow-Wheeler transform .Μελετάται ένα σχέδιο αποθήκευσης μιας συμβολοσειράς S[1,n] που προέρχεται από ένα αλφάβητο Σ που επιτρέπει σε οποιονδήποτε να ανακτήσει οποιαδήποτε υποσυμβολοσειρά του S με μήκος ℓ στο βέλτιστο $O(1 + \ell/\log_{|\Sigma|} n)$ χρόνο. Εφαρμόζουμε επίσης τον αλγόριθμο Burrow-Wheeler transform (BWT) στο πλάνο αποθήκευσης που μελετάμε και επιτυγχάνουμε την ταξινόμηση κατά τέτοιο τρόπο ώστε το τυχαίο κ-οστό στοιχείο να εξαρτάται και από την εντροπία της συμβολοσειράς S κι από την εντροπία της BW-transformed συμβολοσειράς δηλαδή της bwt(S).

Λέξεις Κλειδιά: <<Συμπίεση δεδομένων; Σχέδιο αποθήκευσης; Huffman;BWT>>

ΚΕΦΑΛΑΙΟ 1 – ΕΙΣΑΓΩΓΗ - ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

1.1 Αντικείμενο Αλγορίθμων

1.1.1 Εισαγωγικά για τους Αλγορίθμους

Το λεξικό της Οξφόρδης(Concise Oxford Dictionary)ορίζει ως *αλγόριθμο* μία διαδικασία ή ένα σύνολο κανόνων με σκοπό τον υπολογισμό (ιδιαίτερα τον μηχανιστικό υπολογισμό). Με τον όρο **«αλγόριθμος»** συνεπώς, χαρακτηρίζεται κάθε καλώς ορισμένη υπολογιστική διαδικασία, η οποία καλείται να επιλύσει ένα συγκεκριμένο πρόβλημα, εντός πεπερασμένου χρόνου. Τυπικότερα, κάθε αλγόριθμος συνιστά μια ακολουθία υπολογιστικών βημάτων, η οποία απεικονίζει την **είσοδο (input)** του προβλήματος –ήτοι, τα δεδομένα του- στην **έξοδο (output)** –δηλαδή, στην λύση του προβλήματος. Κάθε είσοδος που ικανοποιεί τις προδιαγραφές του προβλήματος καλείται **νόμιμη (legal)** και λέμε πως ορίζει ένα συγκεκριμένο **στιγμιότυπο (instance)** του προβλήματος. Ένας αλγόριθμος επιλύει ένα πρόβλημα, όταν, για κάθε στιγμιότυπο (instance) του εν λόγω προβλήματος, τερματίζει μετά από πεπερασμένο χρόνο, παράγοντας σωστή έξοδο. Η επιλογή ενός αλγορίθμου για κάποιο πρόβλημα εξαρτάται από πολλούς παράγοντες: το μέγεθος του στιγμιότυπου (δηλ. της συγκεκριμένης μορφής) του προβλήματος, τον τρόπο αναπαράστασης του προβλήματος, την ταχύτητα και το μέγεθος της μνήμης, καθώς και τη διαθεσιμότητα άλλων υπολογιστικών πόρων (αριθμό επεξεργαστών, δίκτυο κλπ.). Ο πλέον γνωστός ιστορικά αλγόριθμος είναι ο αλγόριθμος του Ευκλείδη για τον υπολογισμό του Μέγιστου

Κοινού Διαιρέτη δύο ακεραίων.

1.1.2 Γενική περιγραφή αλγορίθμου

Η φυσιολογική κατάληξη κάθε αλγορίθμου είναι η τελική έκφραση του σε κάποια γλώσσα προγραμματισμού, ώστε, πλέον, να είναι δυνατή η χρήση ενός υπολογιστή για την επίλυση του αντίστοιχου προβλήματος. Συνήθως όμως κατά το στάδιο σχεδιασμού ενός αλγορίθμου προτιμάται η περιγραφή του είτε σε ***φυσική γλώσσα*** είτε σε ψευδογλώσσα προγραμματισμού (pseudo-programming language) ή ψευδοκώδικα (pseudocode). Η τελευταία αποτελείται από μία μίξη εντολών πραγματικών γλωσσών προγραμματισμού με προτάσεις σε φυσική γλώσσα (Ελληνικά Αγγλικά κ.ο.κ.). Κατά αυτό τον τρόπο είμαστε σε θέση να εστιάσουμε την προσοχή μας στον προσδιορισμό του αλγορίθμου και στις ιδιαιτερότητες του προβλήματος. Η μεταγραφή ενός αλγορίθμου σε μία γλώσσα προγραμματισμού αποτελεί από μόνη της μία τέχνη. Ως αποτέλεσμα, τα τελευταία χρόνια έχει εισαχθεί ως γνωστικό αντικείμενο η ***μηχανική των αλγορίθμων(algorithm engineering)***.

1.2 Ανάλυση αλγορίθμων

Ανάλυση Αλγορίθμου (algorithm analysis) αποκαλείται η εύρεση των **πόρων (resources)** που αυτός απαιτεί να τρέξει. Με άλλα λόγια, ο **χρόνος (time)** περάτωσης του και ο αναγκαίος –για

τους υπολογισμούς **χώρος (space)**, μετρούμενος σε **αποθηκευτικές θέσεις (memory locations)**. Οι δυο αυτοί δείκτες μετρήσεων της αποτελεσματικότητας του εκάστοτε αλγορίθμου συνιστούν την **πολυπλοκότητα χρόνου και χώρου** του (**time and space complexity**)(βλ. 1.2.1).

Η ανάλυση των αλγορίθμων μας δίδει τα αναγκαία μέτρα συγκρίσεως για την αξιολόγηση και την επιλογή του καλύτερου, βάσει, βέβαια, των απαιτήσεών μας. Για παράδειγμα, εάν κάποιος ενδιαφέρεται για την γρήγορη περάτωση ενός αλγορίθμου, αδιαφορώντας για το κόστος σε χώρο που πρέπει να καταβάλει ,τότε ο χρόνος είναι το μόνο κριτήριο. Άλλος, πάλι που ενδιαφέρεται να δεσμεύσει για έναν αλγόριθμο όσο το δυνατόν λιγότερο χώρο, πρέπει να ψάξει σε ανάλογες κατευθύνσεις.

Η πιο απλή ανάλυση που μπορεί να γίνει είναι να υλοποιηθεί ο εκάστοτε αλγόριθμος και να μετρηθεί ο πραγματικός χρόνος τρεξίματος του και οι πραγματικές του απαιτήσεις σε μνήμη. Συνήθως όμως προτού κανείς υλοποιήσει τον αλγόριθμο κάνει μία εκτίμηση κατά την φάση του σχεδιασμού. Η εκτίμηση αυτή πρέπει να γίνει, υιοθετώντας ένα κατάλληλο μοντέλο υπολογισμού, το οποίο αποτελεί την αφαιρετική θεώρηση του **υλικού(hardware)** που έχει στην διάθεση του.

Η πιο συνηθισμένη κατηγορία ανάλυσης ενός αλγορίθμου αφορά το χρόνο εκτέλεσης. Ο χρόνος εκτέλεσης ενός αλγορίθμου συνήθως υπολογίζεται σαν συνάρτηση του αριθμού των στοιχειωδών βημάτων που εκτελούνται. Το είδος των στοιχειωδών βημάτων τα οποία προσμετρώνται στο χρόνο εκτέλεσης του αλγορίθμου ποικίλουν ανάλογα με το πρόβλημα.

1.2.1. Ασυμπτωτικός συμβολισμός

1.2.1.1 Ο συμβολισμός Ο

Στην περίπτωση που χρειαζόμαστε μόνο ένα ασυμπτωτικό άνω φράγμα στην τάξη μεγέθους μιας συνάρτησης, χρησιμοποιούμε το συμβολισμό O. Δεδομένης μιας συνάρτησης $g(n)$, συμβολίζουμε με $O(g(n))$ το σύνολο των συναρτήσεων όπου:
$O(g(n)) = \{f(n)$: υπάρχουν θετικές σταθερές c και $n0$ τέτοιες, ώστε:
$0 \leq f(n) \leq cg(n)$ για κάθε $n \geq n_0\}$.
Ο συμβολισμός O, πολλές φορές, χρησιμοποιείται για τη διατύπωση κατά προσέγγιση εκτιμήσεων σχετικά με το χρόνο εκτέλεσης ενός αλγορίθμου. Για παράδειγμα, έστω ένας αλγόριθμος ο οποίος περιέχει το ακόλουθο διπλό φωλιασμένο βρόγχο:

```
for (i =0; i <n; i++){

for j =i; j < n; j++){
< Code >
}
}
```

Αν το κόστος εκτέλεσης της <Code > είναι $O(1)$, δηλαδή σταθερό, τότε το συνολικό κόστος για την εκτέλεση του βρόγχου είναι $O(n^2)$, γιατί οι δείκτες *i* και *j* παίρνουν τιμές από 0 μέχρι *n-1*, και για κάθε τιμή του *i*, η <Code> εκτελείται το πολύ *n* φορές (για την ακρίβεια εκτελείται $n - i + 1$ φορές).

1.2.1.2 Ο συμβολισμός ο

Το ασυμπτωτικό άνω φράγμα που παρέχεται από το συμβολισμό O μερικές φορές είναι ακριβές και άλλες όχι. Ο συμβολισμός *ο* (μικρό

όμικρον) δηλώνει ένα άνω φράγμα το οποίο δεν είναι ακριβές. Οι ορισμοί των συμβολισμών O και ο είναι παρόμοιοι. Η ουσιαστική διαφορά είναι ότι στον ορισμό του O, ισχύει $0 \leq f(n) \leq cg(n)$ για κάποια σταθερά $c > 0$, ενώ στο ορισμό του o, ισχύει $0 \leq f(n) \leq cg(n)$ για κάθε σταθερά $c > 0$.

1.2.2 Μοντέλα υπολογισμού

Η θεωρητική μελέτη των αλγορίθμων και η κατάταξή σε ***κλάσεις πολυπλοκότητας (complexity classes)*** χρησιμοποιεί το μοντέλο Μηχανής Turing, ενώ για την περιγραφή και ανάλυση τους εφαρμόζεται το μοντέλο RAM.

1.2.2.1Μηχανή Turing

Το μοντέλο αυτό πήρε το όνομα του από τον εισηγητή Alan Turing. Σε αδρές γραμμές, αποτελείται από μία συσκευή πεπερασμένου ελέγχου, μία ταινία με αριστερό όριο, αλλά εκτεινόμενη απείρως προς τα δεξιά, και μία κεφαλή ανάγνωσης/εγγραφής της ταινίας. Τυπικότερα μία ***ντετερμινιστική μηχανή Turing*** M (***deterministic Turing Machine***) ορίζεται ως πεντάδα:

$M = (S, \Sigma, \delta, s_0, H)$.

Το S εμφαίνει ένα πεπερασμένο σύνολο ***καταστάσεων (states),*** εκ των οποίων, μια $s_0 \in S$ λογίζεται ως ***αρχική κατάστασή (initial states)***, ενώ ένα υποσύνολο του $H \subseteq S$ αποτελεί το σύνολο των ***τερματικών καταστάσεων (halting states).*** Το σύνολο Σ απαρτίζεται από τα πεπερασμένα στο πλήθος, ***σύμβολα (symbols)***

που χρησιμοποιεί η μηχανή, συμπεριλαμβανομένων των ⊳ και ⊔: το πρώτο δηλοί το ειδικό σύμβολο που τοποθετείται στο αριστερότερο κελί της ταινίας, ενώ το δεύτερο το κενό σύμβολο. Το Σ δεν πρέπει να περιέχει τα σύμβολα ← , →, τα οποία δηλώνουν την κίνηση, κατά μία θέση, δεξιά ή αριστερά, αντίστοιχα, της κεφαλής. Η συνάρτηση:

$$\delta : (S - H) \times \Sigma \rightarrow S \times (\Sigma \cup \{\leftarrow , \rightarrow\})$$

αποτελεί την ***συνάρτηση μεταβάσεως (transition function***), η οποία τηρεί, αναγκαστικά του ακόλουθους δύο περιορισμούς:

(α) Εάν $\delta(q,\rhd) = (p,b)$, $q \in (S - H)$ τότε $b = \rightarrow$, και

(β) εάν $\delta(q,a) = (p,b)$, $q \in (S - H)$ $a \in \Sigma$ τότε $b \neq \rhd$.

1.2.2.2 Μοντέλο Μηχανής τυχαίας προσπέλασης(RAM)

Το **Μοντέλο Μηχανής Τυχαίας Προσπελάσεως RAM (Random Access Machine Model)** αναφέρεται σε συστήματα του ενός επεξεργαστή γενικού σκοπού δίχως την δυνατότητα τελέσεως ταυτόχρονων ενεργειών (concurrent operations). Με απλά λόγια, σε ένα σύστημα που

(α) διαθέτει τους αναγκαίους καταχωρητές (registers), έναν συσσωρευτή (accumulator) και μια ακολουθία αποθηκευτικών

θέσεων με διευθύνσεις 0, 1, 2, …., οι οποίες συνιστούν την κύρια μνήμη του (main memory), και

(β) είναι σε θέση να εκτελεί τις αριθμητικές πράξεις {+, -, *, /, mod }, να παίρνει αποφάσεις διακλάδωσης τύπου **if,** βάσει των τελεστών {=, <, >, ≤, ≥, ≠} και να διαβάζει και να γράφει από και προς τις θέσεις μνήμης.

Οι **στοιχειώδεις πράξεις (primitive operations)** που αναφέρονται στο (β) πρέπει να χρεωθούν κάποιο χρόνο. Υπάρχουν δύο θεωρήσεις:

Η πρώτη, η λεγόμενη και **μέτρηση μοναδιαίου κόστους (unit cost measure)**, χρεώνει κάθε πράξη σταθερό (πεπερασμένο) κόστος, ανεξαρτήτως του μήκους της δυαδικής αναπαραστάσεως των τελεστών (operands). Η δεύτερη, η αποκαλούμενη **μέτρηση λογαριθμικού κόστους (logarithmic cost measure)**, θεωρεί πως η πράξη παίρνει χρόνο ανάλογο με το μήκος της δυαδικής αναπαραστάσεως των τελεστών. Λόγου χάριν, η μετακίνηση του αριθμού n από την κύρια μνήμη προς έναν καταχωρητή χρεώνεται $\lfloor \log n \rfloor + 1$ μονάδες χρόνου. Συνήθως, χρησιμοποιείται το μοναδιαίο μέτρο, εκτός και εάν γίνεται εκτεταμένη χρήση πράξεων επί συμβολοσειρών μπιτ.

1.2.2.3 Ισοδυναμία RAM και μηχανής Turing

Η μηχανή RAM είναι ισοδύναμη ,από απόψεως υπολογιστικής ισχύος με την μηχανή Turing. Κάθε υπολογιστική διαδικασία μίας μηχανής Turing με κόστος πολυωνυμικό στο μέγεθος της εισόδου, δύναται να εξομοιωθεί από μία υπολογιστική, πολυωνυμική στο μέγεθος της εισόδου, διαδικασία μίας μηχανής RAM και αντιστρόφως.

1.3 Ανάλυση χειρότερης περιπτώσεως

Με τον όρο **ανάλυση χειρότερης περίπτωσης (worst case analysis)** ονομάζουμε την μέγιστη τιμή που μπορεί να πάρει ο χρόνος τρεξίματος ή ο χώρος ενός αλγορίθμου για οποιοδήποτε είσοδο με συγκεκριμένο αριθμό n. Επομένως, η ανάλυση χειρότερης περίπτωσης βρίσκει το άνω όριο στην συμπεριφορά του αλγορίθμου, όταν του δοθεί μια νόμιμη είσοδος μεγέθους n. Είναι προφανής η αξία της ανάλυσης χειρότερης περίπτωσης (οτιδήποτε και να συμβεί σε μια ορθή εκτέλεση του αλγορίθμου, ο χρόνος απόκρισης θα είναι το πολύ τόσο άσχημος). Ωστόσο, οι αλγόριθμοι πολλές φορές χρησιμοποιούνται κατ' επανάληψη, και για πολλές διαφορετικές εισόδους.

1.4 Ανάλυση μέσης περίπτωσης

Σε περίπτωση που είναι γνωστή η κατανομή πιθανότητας επί του συνόλου των στιγμιότυπων του εν λόγω προβλήματος, τότε είναι δυνατή η **ανάλυση μέσης** ή **αναμενόμενης περίπτωσης (average/expected case analysis)**. Αυτή μας δίδει την μέση ή την αναμενόμενη συμπεριφορά του αλγορίθμου, όταν του δοθεί μια νόμιμη είσοδος με συγκεκριμένο μέγεθος n.

1.5 Ανάλυση Επιμερισμένης ή Κατανεμημένης Περίπτωσης

Η πολυπλοκότητα μιας δομής καθορίζει και την συμπεριφορά του αλγορίθμου που την μεταχειρίζεται. Υπάρχουν περιπτώσεις, στις οποίες μας ενδιαφέρει ο συνολικός χρόνος *ακολουθίας* πράξεων να είναι φραγμένος. Αυτό συνεπάγεται μεγαλύτερη ευελιξία στο θέμα σχεδιασμού της δομής· επιτρέπεται, πλέον, ο χρόνος μιας πράξεως να μεταβάλλεται, αρκεί «ακριβές» πράξεις να ακολουθούνται από πολλές «φθηνές». Αυτός ο τρόπος αναλύσεως της επιδόσεως μιας δομής, ως μέσος όρος επιδόσεως επί ακολουθίας πράξεων, είναι γνωστός **ως ανάλυση κατανεμημένης ή επιμερισμένης περίπτωσης (amortized-case analysis**).

Τυπικότερα, έστω $T(n)$ ο μέγιστος χρόνος εκτελέσεως μιας οποιασδήποτε ακολουθίας n πράξεων επί μιας δομής. Ως επιμερισμένος ή κατανεμημένος χρόνος για μια πράξη ορίζεται το πηλίκο $T(n)/n$. Μερικές φορές στην βιβλιογραφία χρησιμοποιείται και ο όρος **μέσος χειρότερης περίπτωσης (worst-case average**). Αυτό σημαίνει πως, εάν η επιμερισμένη επίδοση μιας δομής είναι $f(n)$, τότε μια οποιαδήποτε ακολουθία n πράξεων κοστίζει το πολύ $nf(n)$. Στη συνέχεια, θα εξετάσουμε δύο ισοδύναμες τεχνικές επιμερισμένης αναλύσεως: την **μέθοδο λογαριασμού τραπεζίτη (banker account method)** και την **μέθοδο συναρτήσεως δυναμικού (potential function method**).

1.5.1 Μέθοδος Δυναμικού

Η ***μέθοδος δυναμικού (potential method)*** στηρίζεται στην ιδέα της απεικονίσεως της καταστάσεως μιας δομής ή ενός αλγορίθμου Α μέσω μιας συνάρτησης δυναμικού:

$$\mathbf{\Phi} : A \rightarrow \mathbb{R}$$

Αρχικά, αποδίδεται μια αρχική τιμή $\Phi(A_0)$. Μετά την i-στη πράξη o_i, πραγματικού κόστους c_i, έχουμε μετάβαση από την κατάσταση A_{i-1} στην A_i και μεταβολή του δυναμικού κατά:

$$\mathbf{\Delta\Phi_i} = \mathbf{\Phi}(A_i) - \mathbf{\Phi}(A_{i-1})$$

Το κατανεμημένο κόστος c_i' της o_i ορίζεται ως:

$$c_i' = c_i + \boldsymbol{\Delta\Phi_i}$$

Δηλαδή, το πραγματικό κόστος συν την μεταβολή που επήλθε στο δυναμικό εξ αιτίας της o_i.

Από τα παραπάνω καθίσταται φανερό, πως κεντρικό ρόλο στην ανάλυση δυναμικού, παίζει η εκλογή της κατάλληλης συναρτήσεως δυναμικού **Φ**. Χάρις στην τελευταία, κάποιες πράξεις χρεώνονται περισσότερο (όταν $\boldsymbol{\Delta\Phi_i} > 0$) και κάποιες λιγότερο (όταν $\boldsymbol{\Delta\Phi_i} < 0$), συνολικά, όμως, επιτυγχάνεται ορθή ερμηνεία της πολυπλοκότητας της Α.

1.5.2 Λογαριασμός Τραπεζίτη ή Λογιστική Μέθοδος

Κατά τη **μέθοδο λογαριασμού τραπεζίτη (banker account method)** ή την **λογιστική μέθοδο (accounting method)**, κάθε πράξη χρεώνεται ένα **κατανεμημένο** ή **επιμερισμένο κόστος (amortized cost)**, το οποίο, ενδεχομένως να είναι μικρότερο ή και μεγαλύτερο από το αντίστοιχο πραγματικό. Η επιλογή του κατανεμημένου κόστους πρέπει να γίνει κατάλληλα, ούτως ώστε:

(α) να προσεγγίζεται το μέσο κόστος της πράξεως σε μια οποιαδήποτε ακολουθία πράξεων, και

(β) αθροιζόμενα τα επιμέρους κατανεμημένα κόστη όλων των πράξεων να φράσσουν, από πάνω, το πραγματικά παρατηρούμενο χειρότερο της ακολουθίας.

Συνήθως, η διαφορά μεταξύ πραγματικού και κατανεμημένου κόστους χαρακτηρίζεται ως **πίστωση (credit)** και δηλώνει είτε *πλεόνασμα*, που κατατίθεται προς μελλοντική χρήση, κατά την εξυπηρέτηση των επόμενων πράξεων, είτε το *δάνειο*, που λαμβάνεται από τα αποθεματικά, για την κάλυψη των τρεχουσών αναγκών μιας πράξεως. Η πρακτική της πιστώσεως, με τραπεζικούς όρους, χρεώνει τις «φθηνές» πράξεις κάτι παραπάνω, ώστε να καλυφθεί το επιπλέον, από το μέσο παρατηρούμενο, κόστος των «ακριβών» πράξεων.

1.6 Ανάγκη για αποδοτικούς αλγόριθμος

Καθώς η ταχύτητα των υπολογιστών αυξάνει με ταχύτατους ρυθμούς, κάποιος θα μπορούσε να ισχυριστεί ότι μοιάζει άσκοπη η συνεχής αναζήτηση για αποδοτικότερους αλγορίθμους. Στην περίπτωση που η αποδοτικότητα ενός αλγορίθμου δεν μας ικανοποιεί, θα μπορούσαμε να περιμένουμε την επόμενη, ταχύτερη γενιά υπολογιστών, αντί να επενδύσουμε χρόνο και χρήματα στην αναζήτηση ενός αποδοτικότερου αλγορίθμου. Στην πραγματικότητα υπάρχουν πολλή διαφορετική μεταξύ τους αλγόριθμοι που αποδεικνύουν ότι η αποδοτικότητα διαφορετικών αλγορίθμων που επιλύουν το ίδιο πρόβλημα μπορεί να διαφέρει πολύ περισσότερο από την ταχύτητα ενός προσωπικού υπολογιστή και ενός υπέρ – υπολογιστή.

Κεφάλαιο 2 - Συμπίεση δεδομένων – Αλγόριθμος Huffman

2.1 Συμπίεση δεδομένων

Με τον όρο ***συμπίεση δεδομένων (data compression)*** χαρακτηρίζεται η διαδικασία αλλαγής της αναπαραστάσεως ενός αρχείου (κειμένου, εικόνας, ήχου) έτσι ώστε να καταλαμβάνει μικρότερο χώρο στο αποθηκευτικό μέσο ή να χρειάζεται λιγότερο χρόνο μεταδόσεως, ενώ, ταυτόχρονα, να είναι δυνατή η αποκατάσταση της αρχικής μορφής από την συμπιεσμένη (αποσυμπίεση). Γενικότερα η έννοια της συμπίεσης δεδομένων είναι πολύ σημαντική για τους υπολογιστές αφού χωρίς αυτήν δεν θα υπήρχαν υπολογιστές και υπολογιστικά συστήματα. Για το σκοπό αυτό χρησιμοποιούνται πολλές μέθοδοι, οι οποίες χωρίζονται σε δύο μεγάλες κατηγορίες: τις μη απωλεστικές και τις απωλεστικές.

2.1.1 Κατηγορίες συμπίεσης δεδομένων

2.1.1.1 Μη απωλεστική συμπίεση

Στην μη απωλεστική συμπίεση (lossless compression) διατηρείται η ακεραιότητα των δεδομένων. Τα αρχικά δεδομένα και τα δεδομένα μετά τη συμπίεση και την αποσυμπίεση είναι ακριβώς τα ίδια, επειδή σε αυτές τις μεθόδους ο αλγόριθμος συμπίεσης και ο αλγόριθμος αποσυμπίεσης είναι ακριβώς αντίστροφοι. Κατά τη διαδικασία δε χάνεται κανένα μέρος των δεδομένων. Τα πλεονάζοντα δεδομένα κωδικοποιούνται κατά τη συμπίεση και αποκωδικοποιούνται κατά την αποσυμπίεση. Αυτοί οι μέθοδοι χρησιμοποιούνται όταν δεν πρέπει να χαθεί ούτε ένα μπιτ δεδομένων όπως στην περίπτωση ενός αρχείου κειμένου ή ενός προγράμματος.
Τα γνωστά προγράμματα 7z, bz, Zip και Rar χρησιμοποιούν μη απωλεστική συμπίεση.

2.1.1.2 Απωλεστική συμπίεση

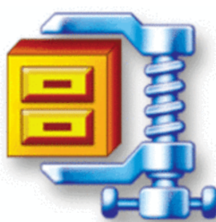
Η απώλεια δεδομένων μπορεί να μην είναι αποδεκτή σε αρχεία κειμένου ή ενός προγράμματος, είναι όμως αποδεκτή σε εικόνες και ταινίες. Ο λόγος είναι ότι τα μάτια μας και τα αφτιά μας δεν μπορούν να διακρίνουν πολύ μικρές αλλαγές. Για τέτοιες περιπτώσεις είναι κατάλληλες οι απωλεστικές μέθοδοι συμπίεσης (lossy data compression). Οι μέθοδοι αυτές είναι οικονομικότερες και απαιτούν λιγότερο χρόνο και χώρο όταν πρέπει να σταλούν εκατομμύρια μπιτ εικόνων και βίντεο το δευτερόλεπτο.

Χαρακτηριστικό παράδειγμα απωλεστικής συμπίεσης εικόνας είναι η μέθοδος JPEG (Joint Photografic Experts Group) για βίντεο η μέθοδος MPEG (Moving Pictures Experts Group) και για ήχο το πρότυπο MP3

2.2 Παραδείγματα μεθόδων συμπίεσης

2.2.1. Zip

Μέθοδος που χρησιμοποιείται στα format PDF and TIFF. Είναι αποτελεσματική για τη συμπίεση εικόνων με μεγάλες περιοχές του ίδιου χρώματος. Η διαφορά μεταξύ των αρχείων Rar που αναφέρονται παρακάτω είναι ότι μπορούν να συμπιεστούν και να αποσυμπιεστούν από τα Windows. Το εργαλείο που χρησιμοποιείται για την συμπίεση και αποσυμπίεση των αρχείων Zip είναι το WinZip.



Εικονίδιο εργαλείου WinZip

2.2.2 Rar

Ένα αρχείο RAR είναι μια συλλογή συμπιεσμένων αρχείων, τα οποία έχουν υποστεί επεξεργασία προκειμένου να καταλαμβάνουν λιγότερο χώρο. Συνήθως αναφέρεται με αυτό τον τρόπο, επειδή τα

αρχεία RAR συνήθως χρησιμοποιούν μια επέκταση ονόματος αρχείου τύπου .rar. Προκειμένου να γίνει εξαγωγή των αρχείων από ένα συμπιεσμένο αρχείο RAR, θα πρέπει να χρησιμοποιηθεί ένα ειδικό πρόγραμμα το οποίο έχει σχεδιαστεί για να λειτουργεί με αρχεία RAR το WinRAR.

Εικονίδιο εργαλείου WinRAR

2.2.3 JPEG

Οι εικόνες ψηφιοποιούνται και αποθηκεύονται με βάση τον αλγόριθμο συμπίεσης **JPEG** .Ο αλγόριθμος βασίζεται πάνω σε μερικές παρατηρήσεις που έχουν σχέση με τις ιδιαιτερότητες του ανθρώπινου ματιού και διάφορες μαθηματικές και φυσικές αναλύσεις. Το ανθρώπινο μάτι είναι περισσότερο ευαίσθητο στη φωτεινότητα μιας εικόνας και λιγότερο στις χρωματικές αποχρώσεις που περιέχει. Ο JPEG επιτυγχάνει αναλογία συμπίεσης 10 : 1.

2.2.4 MPEG

Η φιλοσοφία του αλγορίθμου MPEG στηρίζεται στη παρατήρηση, ότι όταν τα διαδοχικά καρέ δεν διαφέρουν σημαντικά μεταξύ τους δεν παράγεται σημαντικός αριθμός νέων δεδομένων και το

αντίθετο. Το αποτέλεσμα αυτής της κωδικοποίησης είναι μια μέση συμπίεση των δεδομένων σε αναλογία 25 : 1.

2.2.5 MP3

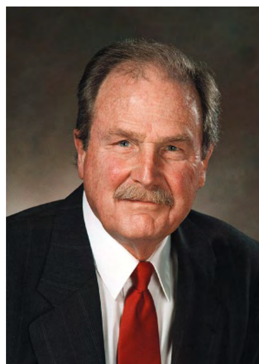
Το MP3 είναι ένα πρότυπο ψηφιακής κωδικοποίησης ήχου, το οποίο βασίζεται στην απωλεστική συμπίεση αρχείων μέσω ενός αλγορίθμου σχεδιασμένου να μειώνει δραστικά το πλήθος των ψηφιακών δεδομένων που απαιτούνται για την αποθήκευση και ορθή αναπαραγωγή του ήχου, ο οποίος ωστόσο συνεχίζει να ακούγεται σαν πιστή αναπαραγωγή του αρχικού ασυμπίεστου περιεχομένου από τους περισσότερους ακροατές.

2.3 Αποσυμπίεση αρχείων

Φυσικά και για να υπάρχει η συμπίεση πρέπει να υπάρχει και ***αποσυμπίεση αρχείων (uncompress files)*** . Θα πρέπει δηλαδή όλα τα αρχεία που συμπιέζονται να έχουν την δυνατότητα να επανέλθουν στην αρχική πρωτότυπη μορφή τους χωρίς ελλείψεις και "ζημιές". Αν δεν υπάρχει κώδικας αποσυμπίεσης τότε όλα αυτά τα εργαλεία συμπίεσης που αναφέρθηκαν πρωτύτερα δεν έχουν καμία χρήση.

2.4 Αλγόριθμος συμπίεσης Huffman

Το 1952, ο David Huffman παρουσίασε έναν άπληστο αλγόριθμο για τον υπολογισμό ενός βέλτιστου κώδικα προθεμάτων, ο οποίος ονομάζεται κώδικας Huffman. Στηρίζεται στην αρχή πως κοινοί χαρακτήρες (δηλαδή χαρακτήρες με μεγάλη συχνότητα) κωδικοποιούνται με λιγότερα μπιτ από ότι οι σπανιότεροι. Η μέθοδος Huffman στηρίζεται στην έννοια των ***κωδικών προθεμάτων (prefix codes)***, οι οποίοι χαρακτηρίζονται από την ιδιότητα πως κανένας τους δεν αποτελεί πρόθεμα οποιουδήποτε άλλου. Η ιδιότητα αυτή καθιστά δυνατή την περιγραφή των κωδικών με ένα δυαδικό δένδρο T , όπου κάθε αριστερή διακλάδωση αντιστοιχεί στο ψηφίο ‘0’ , κάθε δεξιά στο ψηφίο ‘1’, ενώ έκαστο φύλλο απεικονίζει και έναν διακριτό κωδικό. Σε κάθε εσωτερικό κόμβο αντιστοιχούμε συχνότητα ίση με το άθροισμα των συχνοτήτων όλων των χαρακτήρων/φύλλων του υποδέντρου. Σε κάθε βήμα, τα δύο υποδέντρα με τις μικρότερες συχνότητες ενώνονται και αντικαθιστώνται από ένα υποδέντρο που δημιουργείται με την προσθήκη ενός νέου εσωτερικού κόμβου. Στο νέο κόμβο αντιστοιχίζουμε συχνότητα ίση με το άθροισμα των συχνοτήτων των ριζών των δύο υποδέντρων. Ο αλγόριθμος ολοκληρώνεται όταν απομένει μόνο ένα δέντρο που περιέχει όλους τους χαρακτήρες που εμφανίζονται σαν φύλλα.

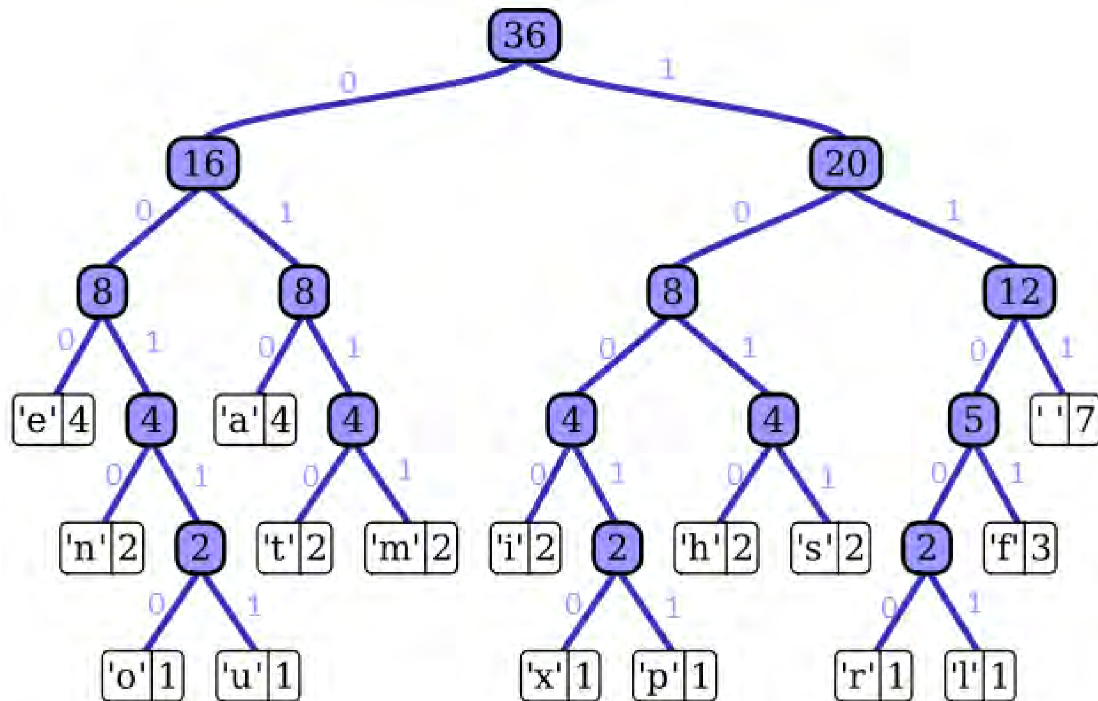


Dr. David Huffman

2.4.1 Δυαδικό δέντρο T του αλγορίθμου Huffman

Η πρόταση είναι: this is an example of a huffman tree

Στα φύλλα του δέντρου φαίνεται ο χαρακτήρας στα αριστερά και η συχνότητα του κάθε χαρακτήρα στα δεξιά του φύλλου.



2.4.2 Αλγόριθμος Huffman σε ψευδοκώδικα

Algorithm Huffman Coding (char [] C , int [] F)
Input: Δυο πίνακες C, F με τους χαρακτήρες και την συχνότητα τους, αντίστοιχα
Output: Η ρίζα του δέντρου κωδικοποιήσεως Τα

1. pqueue pq = new pqueue(); // ουρά προτεραιότητας
2. for (i = 0; i< C.length; i++) // αρχικοποίηση με τα φύλλα
3. pq.insKey(new bNode(new Item(F[i],C[i],F[i])));

```
4.  for (i = 1; i< C.length; i++){
5.  bNode t = new bNode();              //νέος εσωτερικός κόμβος
6.  x = pq.delMin();
7.  y = pq.delMin();
8.  t.setLeftSon(x);
9.  t.setRightSon(y);
10.     t.setItem(new Item(O,x.Key+y.Key));        //με εσωτερικό
    //στοιχείο τον τεχνητό χαρακτήρα O και συχνοτ. το άθροισμα τους
11.     pq.insKey(t,t.Key);
12.     }
13.     return  pq.delMin();        //ρίζα του δέντρου T ο εναπομείνας
    // κόμβος
```

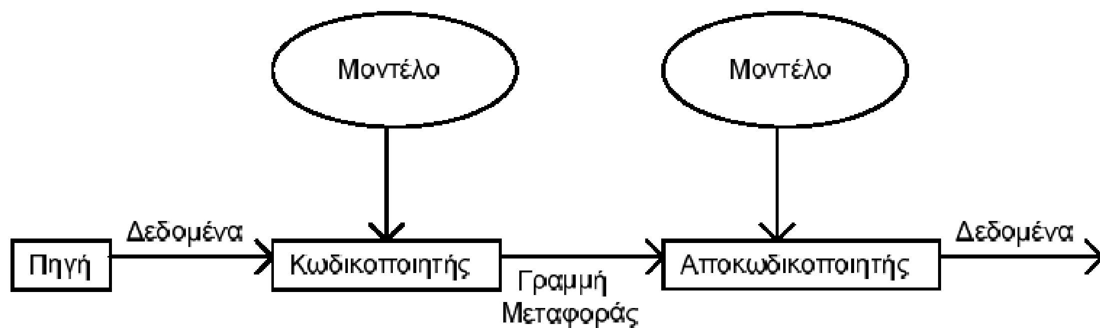
2.4.3 Σημερινή χρήση του αλγόριθμου

Η κωδικοποίηση Huffman είναι ακόμα σε ευρεία χρήση λόγω της απλότητας, της υψηλής ταχύτητας και της έλλειψης επιβάρυνσής του συστήματος που χρησιμοποιεί. Η κωδικοποίηση Huffman σήμερα χρησιμοποιείται συχνά ως back-end σε κάποια άλλη μέθοδο συμπίεσης όπως στην JPEG και MP3.

2.5 Αποσυμπίεση Huffman

Η διαδικασία της αποσυμπίεσης είναι απλά θέμα μετάφρασης των μεμονωμένων τιμών κάθε byte, διασχίζοντας από κόμβο σε κόμβο του δυαδικού δέντρου T, φτάνοντας σε ένα κόμβο που

τερματίζει η διαδικασία (η κορυφή του δέντρου). Υπάρχουν πολλές διαφορετικές διαδικασίες για την αποκωδικοποίηση Huffman. Σε κάθε περίπτωση, δεδομένου ότι τα συμπιεσμένα δεδομένα μπορεί να περιλαμβάνουν αχρησιμοποίητα bits η αποσυμπίεση πρέπει να είναι σε θέση να προσδιορίσει πότε θα σταματήσει να παράγει αντίγραφα. Αυτό μπορεί να επιτευχθεί είτε με τη διαβίβαση του μήκους των δεδομένων που θα αποσυμπιεστούν ή ορίζοντας ένα ειδικό κωδικό για να σημάνει το τέλος των εισροών.



Κεφάλαιο 3 - Αλγόριθμοι Ταξινόμησης – Burrow -Wheeler Transform

3.1 Ταξινόμηση δεδομένων

Με τον όρο ταξινόμηση ή διάταξη (sorting) καλούμε την επιβολή μίας δοθείσης ολικής διατάξεως (U,R) σε κάθε (πολύ)σύνολο S ⊆ U, |S| = n. Ένας αλγόριθμος καλείται αλγόριθμος ταξινομήσεως ή διατάξεως, όταν, δοθέντων μίας (U,R) και ενός S ⊆ U, είναι σε θέση πάντοτε να επιστρέφει το (πολύ)σύνολο S, διατεταγμένο βάσει της R. Ένας αλγόριθμος διατάξεως καλείται ***σταθερός (stable),*** όταν, μετά το πέρας του αλγορίθμου, τα όμοια στοιχεία διατηρούν την, προ της εφαρμογής μεταξύ τους σχετική διάταξη. Διαφορετικά χαρακτηρίζεται ως ***ασταθής (unstable)***. Μία άλλη διάκριση στους εν λόγω αλγορίθμους μπορεί να γίνει βάσει του εάν χρησιμοποιούν μόνο την ολική διάταξη (U,R) κατά την λήψη αποφάσεων, οπότε χαρακτηρίζοντας ***συγκριτικοί (comparison-based)*** ή ***γενικοί (general)*** ή εκμεταλλεύονται και την επιπλέον πληροφορία που δίνουν τα ποιοτικά χαρακτηριστικά του σύμπαντος U –λόγου χάριν, ότι μας δίδονται προς ταξινόμηση πενταψήφιοι δεκαδικοί αριθμοί. Οι δεύτεροι δρουν σωστά, μόνον όταν το σύνολο εισόδου ικανοποιεί τις προδιαγραφές του σύμπαντος και είναι γνωστοί ως ***αλγόριθμοι κατανομής (distribution-based).***

Ένα ζήτημα που ενδιαφέρει τους αλγορίθμους διάταξης, όπως και οποιονδήποτε άλλο αλγόριθμο, είναι η πολυπλοκότητα χρόνου ως συνάρτηση του αριθμού των προς διάταξη δεδομένων (στοιχείων).

Στους αλγορίθμους διάταξης συναντάμε τις ακόλουθες βασικές υπολογιστικές πράξεις:

(α*) συγκρίσεις* (*comparisons*), για τη σύγκριση δύο στοιχείων του πίνακα,

(β) *εναλλαγές* ή *αντιμεταθέσεις* (*interchanges*), για την αμοιβαία αλλαγή των τιμών δύο στοιχείων του πίνακα,

(γ) *καταχωρίσεις* (*assignments*), για καταχώριση τιμών σε μεταβλητές ή στοιχεία του πίνακα

3.1.1 Αλγόριθμοι διάταξης δεδομένων

Υπάρχουν αρκετοί μέθοδοι ή αλγόριθμοι διάταξης (sorting algorithms) που δίνουν λύση στο παραπάνω πρόβλημα. Τέτοιοι είναι οι εξής:

- διάταξη φυσαλίδας (bubble sort)
- διάταξη εισαγωγής (insertion sort)
- διάταξη επιλογής (selection sort)
- γρήγορη διάταξη (quicksort)
- διάταξη συγχώνευσης (merge sort)
- διάταξη σωρού (heapsort)
- διάταξη ρίζας (radix sort)

Επίσης υπάρχουν και παραλλαγές τους. Οι αλγόριθμοι αυτοί χαρακτηρίζονται ως *αλγόριθμοι εσωτερικής διάταξης* (*internal*

sorting algorithms), σε αντιδιαστολή με τους *αλγορίθμους εξωτερικής διάταξης* (*external sorting algorithms*). Οι πρώτοι αναφέρονται στη διάταξη στοιχείων αποθηκευμένων (σε πίνακα) στην κύρια μνήμη του Η/Υ, ενώ οι δεύτεροι στη διάταξη στοιχείων αποθηκευμένων (σε αρχεία) στη δευτερεύουσα μνήμη του Η/Υ.

3.2 Burrow-Wheeler Transform

3.2.1 Εισαγωγή

Φαίνεται ότι δεν υπάρχει όριο στην ποσότητα των δεδομένων που βρίσκονται αποθηκευμένα στους υπολογιστές μας ή στα δεδομένα που θέλουμε να αποστείλουμε. Παρόλο που η τεχνολογία μας παρέχει μεγαλύτερους δίσκους αποθήκευσης και ταχύτερα δίκτυα επικοινωνίας, η ανάγκη των ταχύτερων και πιο αποδοτικών αλγόριθμων συμπίεσης δεδομένων είναι πάντα στην πρώτη γραμμή της τεχνολογικής έρευνας.

Η πρόοδος όσον αφορά τη συμπίεση των δεδομένων αποτελείται συνήθως από μια μακρά σειρά από μικρές βελτιώσεις και μικροσυντονισμό των υπαρχόντων αλγορίθμων. Ωστόσο κάποιες φόρες η έρευνα βιώνει γιγαντιαία άλματα, όπως συνέβη στην περίπτωση της εισαγωγής του Burrow – Wheeler Transform στον τομέα της συμπίεσης χωρίς απώλειες δεδομένων (*Μη απωλεστική συμπίεση*). Ουσιαστικά ο BWT ταξινομεί κατά τέτοιο τρόπο μία συμβολοσειρά S που κάνει την συμπίεση πιο εύκολη και πιο γρήγορη. Επινοήθηκε από τον Michael Burrows και David Wheeler το 1994 ενώ εργαζόταν στο Systems Research Center στο Πάλο Άλτο της Καλιφόρνια. Βασίζεται σε αδημοσίευτο μέχρι τότε μετασχηματισμό που ανακαλύφθηκε από Wheeler το 1983. Ο David Wheeler έφυγε από την ζωή στις 13.12.2004.

(a) (b)

(a) David Wheeler (b) Michael Burrows

3.2.2 Χρήση του BWT

Ο BWT (συνήθως έτσι συναντάτε) είναι ένα πολύ ισχυρό εργαλείο ακόμη και για απλούστερους αλγόριθμους που όταν τον χρησιμοποιούν έχουν εκπληκτικά καλές επιδόσεις. Χρησιμοποιείται σε τεχνικές συμπίεσης δεδομένων, όπως το bzip2 και szip που αποτελούν δύο από τις καλύτερες μεθόδους συμπίεσης που διατίθενται σήμερα. Η επιστημονική κοινότητα θεωρεί πως είναι πιθανό σε μερικά χρόνια να γίνει το νέο πρότυπο στη συμπίεση χωρίς απώλειες δεδομένων.

3.2.3 Περιγραφή του BWT

Ο αλγόριθμος BWT αποτελείται από μία αναστρέψιμη μεταμόρφωση μίας συμβολοσειράς εισόδου s δηλαδή μπορεί να προκύψει από την παραλλαγμένη συμβολοσειρά η αρχική διάταξη των στοιχείων της. Η μετατραπείσα συμβολοσειρά που συμβολίζεται σαν BWT(s) είναι κατά τέτοιο τρόπο ταξινομημένη που γίνεται εξαιρετικά κατάλληλη για συμπίεση. Εάν η συμβολοσειρά περιέχει

πολλούς όμοιους χαρακτήρες τότε η μετατρεμμένη συμβολοσειρά θα έχει αυτούς χαρακτήρες μαζί στην σειρά. Αυτό είναι πολύ χρήσιμο για τη συμπίεση, δεδομένου ότι τείνει να είναι εύκολο να συμπιεστεί μια συμβολοσειρά που έχει τρεξίματα με επαναλαμβανόμενους χαρακτήρες στην σειρά παρά μια συμβολοσειρά που έχει μεν κοινούς χαρακτήρες αλλά διασκορπισμένους κατά μήκος.

Η παραλλαγμένη συμβολοσειρά φαίνεται παρακάτω όπου έχουμε s=mississippi. Αρχικά προστίθεται το τερματικό στοιχείο #(end-of-file) στο τέλος της λέξης. Εν συνεχεία σχηματίζεται ένας πίνακας με όλες τις κυκλικές εναλλαγές προς τα αριστερά του #(left shifts) όπως φαίνεται στον πρώτο πίνακα του σχήματος παρακάτω. Ταξινομούνται οι γραμμές του πίνακα από τα δεξιά προς τα αριστερά με αλφαβητική σειρά θεωρώντας το # το στοιχείο με την υψηλότερη προτεραιότητα. Μετά από αυτήν την διαδικασία τίθεται ως έξοδο του BWT(s) η τελευταία στήλη του δεύτερου πίνακα, δηλαδή η L, χωρίς όμως το τερματικό στοιχείο #. Άρα BWT(s)=L= ipssmpissii .Επίσης υπάρχει και το στοιχείο I το οποίο ισούται με την γραμμή του ταξινομημένου πίνακα που έχει σαν τελευταίο στοιχείο το #(στο συγκεκριμένο παράδειγμα I=5) .

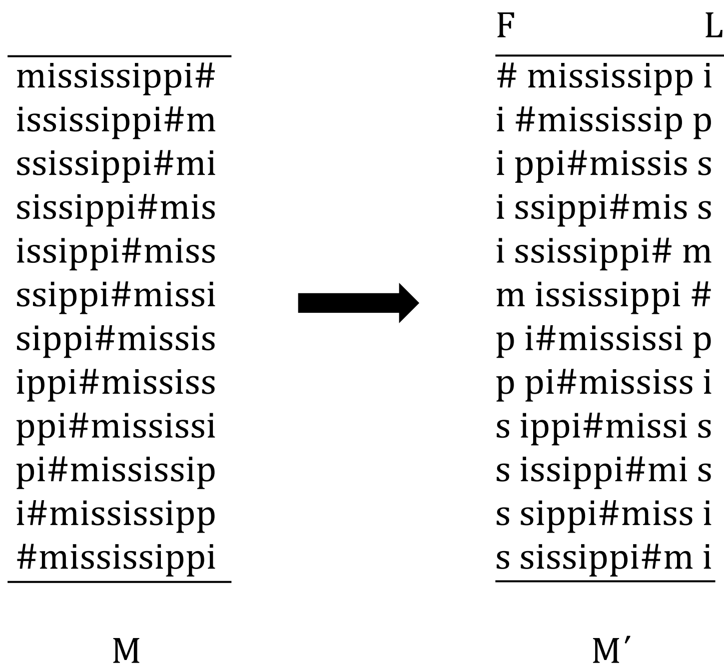
Παρότι μπορεί να φαίνεται περίεργο, έχοντας μόνο το BWT(s) και το I μπορούμε να ξαναγυρίσουμε και πάλι στο s. Προσθέτοντας και πάλι το # στην I-οστή θέση του BWT(s) έχουμε και πάλι την τελευταία στήλη L του ταξινομημένου πίνακα. Εάν τώρα ταξινομήσουμε αλφαβητικά την L μας προκύπτει η πρώτη στήλη του ταξινομημένου πίνακα F. Έστω s_i (αντίστοιχα F_i L_i) το i-οστό στοιχείο του s. Η θεμελιώδης παρατήρηση είναι ότι για $i = 2,\ldots, |S|+1$, το F_i είναι το σύμβολο που ακολουθεί μετά το L_i,μέσα στο s. Αυτή η ιδιότητα επιτρέπει στους χρήστες να ξαναβρούν την s, χαρακτήρα χαρακτήρα.

Ο λόγος που μας ενδιαφέρει τόσο η συμβολοσειρά BWT(s) είναι ότι έχει την εξής αξιοσημείωτη ιδιότητα: για κάθε υποσυμβολοσειρά w του s τα στοιχεία μετά την w είναι όλα συγκεντρωμένα μαζί στο

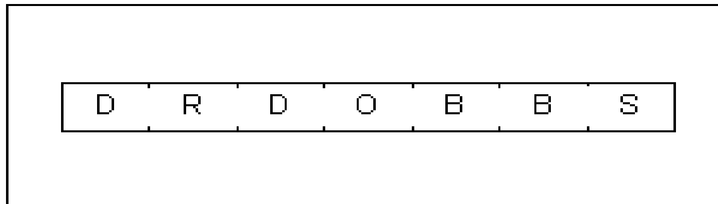
BWT(s). Δεδομένου ότι μετά από ένα (αρκετά μεγάλο) w λίγα θα είναι τα στοιχεία που θα ταιριάζουν με αυτήν την υποσυμβολοσειρά Χαρακτηριστικό παράδειγμα της αντίστροφης διαδικασίας είναι αυτό που ακολουθεί όπου το BWT(s)=BNN^AA@A και s=^BANANA@.

Δύο από τα πιο γνωστά παραδείγματα του BWT είναι:

1.Η συμβολοσειρά εισόδου είναι s=mississippi



2.Το s σε αυτό το παράδειγμα είναι:



και ο ταξινομημένος πίνακας γίνεται ως εξής:

	F						L
S 4	B	B	S	D	R	D	O
S 5	B	S	D	R	D	O	B
S 2	D	O	B	B	S	D	R
S 0	D	R	D	O	B	B	S
S 3	O	B	B	S	D	R	D
S 1	R	D	O	B	B	S	D
S 6	S	D	R	D	O	B	B

Αντίστροφη διαδικασία του αλγορίθμου:

Inverse Transformation

Input
BNN^AA@A

Add 1	Sort 1	Add 2	Sort 2
B	A	BA	AN
N	A	NA	AN
N	A	NA	A@
^	B	^B	BA
A	N	AN	NA
A	N	AN	NA
@	^	@^	^B
A	@	A@	@^

Add 3	Sort 3	Add 4	Sort 4
BAN	ANA	BANA	ANAN
NAN	ANA	NANA	ANA@
NA@	A@^	NA@^	A@^B
^BA	BAN	^BAN	BANA
ANA	NAN	ANAN	NANA
ANA	NA@	ANA@	NA@^
@^B	^BA	@^BA	^BAN
A@^	@^B	A@^B	@^BA

Add 5	Sort 5	Add 6	Sort 6
BANAN	ANANA	BANANA	ANANA@
NANA@	ANA@^	NANA@^	ANA@^B
NA@^B	A@^BA	NA@^BA	A@^BAN
^BANA	BANAN	^BANAN	BANANA
ANANA	NANA@	ANANA@	NANA@^
ANA@^	NA@^B	ANA@^B	NA@^BA
@^BAN	^BANA	@^BANA	^BANAN

A@^BA	@^BAN	A@^BAN	@^BANA
Add 7	**Sort 7**	**Add 8**	**Sort 8**
BANANA@	ANANA@^	BANANA@^	ANANA@^B
NANA@^B	ANA@^BA	NANA@^BA	ANA@^BAN
NA@^BAN	A@^BANA	NA@^BANA	A@^BANAN
^BANANA	BANANA@	^BANANA@	BANANA@^
ANANA@^	NANA@^B	ANANA@^B	NANA@^BA
ANA@^BA	NA@^BAN	ANA@^BAN	NA@^BANA
@^BANAN	^BANANA	@^BANANA	^BANANA@
A@^BANA	@^BANAN	A@^BANAN	@^BANANA
Output			
^BANANA@			

Ο ακόλουθος ψευδοκώδικας δείχνει με απλό, αλλά αποτελεσματικό τρόπο την διαδικασία που ακολουθεί ο BWT καθώς και ο αντίστροφος. Υποθέτει ότι η είσοδος είναι το s και περιέχει έναν ειδικό χαρακτήρα “EOF”(end-of-file)που είναι ο τελευταίος χαρακτήρας, και δεν εμφανίζεται πουθενά αλλού στο κείμενο.

```
function BWT (string s)
   create a table, rows are all possible rotations of s
   sort rows alphabetically
   return (last column of the table)
```

```
 function inverseBWT (string s)
   create empty table

   repeat length(s) times
       insert s as a column of table before first column of the table
// first insert creates first column
       sort rows of the table alphabetically
   return (row that ends with the 'EOF' character)
```

3.2.4 Γιατί ο ταξινομημένος πίνακας συμπιέζεται καλύτερα

Ο BWT ταξινομεί τις σειρές του πίνακα M και δίνει σαν έξοδο L το οποίο αποτελείται από τους τελευταίους χαρακτήρες της κάθε περιστροφής Γιατί λοιπόν αυτή η καινούργια συμβολοσειρά δίνει καλύτερα αποτελέσματα από την παλιά; Για να δούμε το γιατί μπορούμε να πάρουμε τα αποτελέσματα της ταξινόμησης σε ένα γράμμα μίας αγγλικής λέξης. Για παράδειγμα μπορούμε να πάρουμε το ‘t’ του ‘the’. O Burrows Wheeler Transform περιστρέφει όλες τις συμβολοσειρές του ‘the’ όπως το και στο δικό μας παράδειγμα. Αν το ‘he’ βρίσκεται στην αρχή τότε το ‘t’ βρίσκεται στο τέλος της συμβολοσειράς. Και εκτός από το t του s δεν υπάρχουν πολλές επιλογές για το τέλος της συμβολοσειράς (η οποία ξεκινά με ‘he’).

3.2.5 Βελτιώσεις του BWT

Υπάρχει ένας αριθμός αλγορίθμων που βελτιώνουν την λειτουργία του BWT είτε χωρίς την αλλαγή εξόδου (αλλάζοντας την πολυπλοκότητα) είτε αλλάζοντας και την έξοδο και την πολυπλοκότητα του αρχικού αλγορίθμου. Οι περισσότεροι αλγόριθμοι συμπίεσης βασισμένοι στον BWT (BWT-based

compressors) παίρνουν σαν είσοδο τους το αρχείο σε κομμάτια(blocks). Ένας από αυτούς είναι η μέθοδος συμπίεσης bzip2. Ένας άλλος αλγόριθμος προέρχεται από τον Schindler που δημιούργησε το BWS. Αυτός αλλάζει την έξοδο και την μετατρέπει ως εξής: BWS(mississippi) = msspipisisi . Επίσης ένας άλλος αλγόριθμος που προέρχεται από επιστημονική έρευνα από τον Sadakane θεωρεί πως με την έξοδο του BWT μπορούμε να χτίσουμε την συμβολοσειρά εισόδου με έναν πολύ αποτελεσματικό τρόπο. Πρότεινε λοιπόν έναν τροποποιημένο μετασχηματισμό bwu όπου η ταξινόμηση του πίνακα γίνετε χωρίς να λαμβάνετε υπόψιν η αλφαβητική σειρά. Αυτή η τεχνική ονομάζεται ενοποίηση *(unification)*.

3.2.6 Μειονέκτημα του BWT

Το μειονέκτημα του BWT αλγορίθμου είναι ότι δεν είναι on-line, δηλαδή, θα πρέπει να έχει την δυνατότητα να επεξεργαστεί ένα μεγάλο μέρος της εισόδου(input) πριν παραχθεί ένα ενιαίο κομμάτι εξόδου(output). Το ζήτημα της δημιουργίας on-line BWT έχει αντιμετωπιστεί ερευνητικά, αλλά εξακολουθεί να είναι αναγκαία και περαιτέρω εργασία σε αυτή την κατεύθυνση.

3.3 Θεωρήματα που προκύπτουν από την σύμπραξη των αλγορίθμων BWT και Huffman

Έστω S[1,n] μία συμβολοσειρά σχεδιασμένη από ένα αλφάβητο Σ, και να υποθέσουμε ότι το n είναι πολλαπλάσιο του b = $\lfloor \frac{1}{2}\log_{|\Sigma|} n \rfloor$. Κάθε μπλοκ S_i που αποτελείται από μια δυαδική συμβολοσειρά, συμβολίζεται με την κωδική ονομασία enc(i). Θέτουμε ως V το εξής: V=enc(1)...enc($\frac{n}{b}$).

3.3.1 Θεώρημα 1

Στο συγκεκριμένο σύστημα αποθήκευσης η συμβολοσειρά S[1,n] κωδικοποιείται με |V| + O($\frac{n log \log n}{\log_{|\Sigma|} n}$) bits (Η απόδειξη του στο [1] βιβλιογραφία).

3.3.2 Θεώρημα 2

Στο συγκεκριμένο σύστημα αποθήκευσης ανακτά μια υποσυμβολοσειρά του S με μέγεθος ℓ σε βέλτιστο O(1 + $\frac{l}{\log_{|\Sigma|} n}$) χρόνο. (Η απόδειξη του στο [1] βιβλιογραφία).

3.3.3 Θεώρημα 3

Είναι γνωστό ότι ο αλγόριθμος συμπίεσης Huffman δεν μπορεί να αναπαραστήσει ένα χαρακτήρα με λιγότερο από ένα bit. Για να παρακαμφθεί αυτό, η συμβολοσειρά S συνήθως χωρίζεται σε $\frac{n}{l}$ κομμάτια μήκους l το καθένα και στην συνέχεια ο Huffman εφαρμόζεται στο αλφάβητο Σ^l. Είναι φυσικό να αναρωτηθεί ποια είναι η αναλογία συμπίεσης αυτής της τεχνικής. Το παρακάτω θεώρημα ορίζει την εντροπία του 0-στου χαρακτήρα του S_l σε σχέση με την εμπειρική εντροπία του k-στο χαρακτήρα της αρχικής συμβολοσειράς S. (Η απόδειξη του στο [1] βιβλιογραφία).

$H_0(S_l) \leq l\, H_k(S) + O(k \log |\Sigma|)$, όπου $k \leq l$.

3.3.4 Θεώρημα 4

Στο συγκεκριμένο σύστημα αποθήκευσης όπου εφαρμόστηκε το L=BWT(S) δεν παίρνει περισσότερο από $\min\{nH_k(L), nH_k(S)\} + o(n \log|\Sigma|)$ bits, με $k = o(\log_{|\Sigma|} n)$. Κάθε ℓ μήκους υποσυμβολοσειρά του L μπορεί να ανακτηθεί σε βέλτιστο $O(1 + \frac{l}{\log_{|\Sigma|} n})$ χρόνο. (Η απόδειξη του στο [1] βιβλιογραφία).

3.4 Παράδειγμα της σύμπραξης των δύο αλγορίθμων BWT και Huffman

Στο παράδειγμα μας χρησιμοποιούμε την συμβολοσειρά που χρησιμοποιήσαμε και παραπάνω, s=mississippi.

Η διαδικασία που θα ακολουθηθεί φαίνεται στο παρακάτω διάγραμμα εμφανώς:

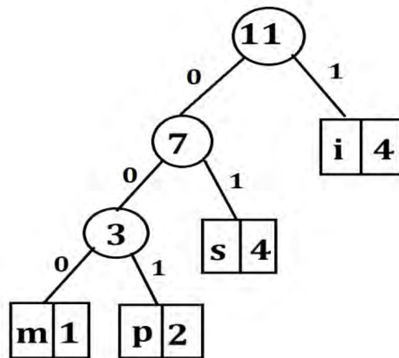
Με άλλα λόγια η διαδικασία που ακολουθείται είναι ότι η συμβολοσειρά s αποτελεί είσοδο του αλγορίθμου BWT. Η έξοδος του θα αποτελεί είσοδο για τον αλγόριθμο συμπίεσης Huffman. Για την αντίστροφη διαδικασία τώρα, η έξοδος του Huffman χρησιμοποιείται σαν είσοδος στον αλγόριθμο αποσυμπίεσης Huffman και η τελική μορφή της αρχικής συμβολοσειράς s δίνεται από την έξοδο του Un.BWT της αντίστροφής δηλαδή διαδικασίας του BWT.

Όπως αναφέραμε και προηγουμένως η είσοδος στο παράδειγμα μας είναι η s=mississippi .

1ο Βήμα: Η s δίνεται σαν είσοδος στον BWT.
Η αντίστοιχη έξοδος είναι η ταξινομημένη μορφή της εισόδου s: BWT(s)= ipssmpissii και I =5(θέση του στοιχείου # ‘EOF’) (παράδειγμα/Σελ. 33).Το BWT(s) μαζί με το I μας δίνουν το L. Το L ταξινομημένο αλφαβητικά δίνει σαν αποτέλεσμα το F.

2ο Βήμα: Το L μαζί με την πληροφορία του I δίνεται σαν είσοδος στον αλγόριθμο συμπίεσης Huffman. Αυτός με την σειρά του θα μετατρέψει την είσοδο, μέσω του δυαδικού δένδρου που φαίνεται παρακάτω δίνοντας τις αντίστοιχες εξόδους.

<u>Binary Tree Huffman</u>



Άρα η συμβολοσειρά μας μετατρέπεται σε:
s=000 1 10 10 1 10 10 1 100 100 1
Αυτή θα είναι η είσοδος για το επόμενο βήμα που είναι η διαδικασία αποσυμπίεσης του αλγορίθμου Huffman.

3ο Βήμα: Η αποσυμπίεση Huffman χρησιμοποιεί και αυτή με την σειρά της το δυαδικό δένδρο με την ακριβώς αντίθετη διαδικασία. Ξεκινάει από την κορυφή του δένδρου και το διατρέχει ολόκληρο μέχρι να βρει κάποιο τερματικό φύλο όπου αυτό θα είναι και το ζητούμενο στοιχείο.

4ο Βήμα: Το τελευταίο βήμα της διαδικασίας είναι η αντίστροφη διαδικασία του 1ου βήματος δηλαδή του αλγορίθμου BWT. Εκεί δίνεται σαν είσοδος η έξοδος της αποσυμπίεσης Huffman και η έξοδος που δίνει είναι η αρχική μορφή της συμβολοσειράς s= mississippi χωρίς καμία απολύτως αλλαγή. Με αυτό το βήμα ολοκληρώνεται η διαδικασία.

Συμπεράσματα (Conclusions)

Οι μέθοδοι που προτάθηκαν στην παρούσα διπλωματική εργασία αποτελούν μια απλούστευση και βελτίωση των ήδη υπαρχόντων αλγορίθμων. Αυτά οδήγησαν τους ερευνητές σε δύο πιθανές εκτιμήσεις.

Η πρώτη είναι ότι τώρα έχουμε μια λύση του προβλήματος αποθήκευσης των συμβολοσειρών, δηλαδή μπορούν να βρεθούν επιτυχημένες εφαρμογές σε πολλά ενδιαφέροντα περιβάλλοντα και καταστάσεις.

Η δεύτερη παρατήρηση αφορά την μελλοντική έρευνα που τώρα είναι σε θέση να πραγματοποιηθεί. Όλες οι γνωστές λύσεις απέχουν πολύ από το να χρησιμοποιηθούν στην πράξη, λόγω του επιπρόσθετου όρου που συνήθως κυριαρχεί το κ-στο στοιχείο της εντροπίας. Περισσότερη έρευνα χρειάζεται ακόμη για να επιτευχθεί ένα άνω όριο που θα είναι εφικτό στην συμπίεση της οικογένειας των Bzip ,όπου ο επιπρόσθετος όρος είναι O($|\Sigma|^k$ log n) και όχι o(nlog |Σ|)bits.

Παρόλο που υπάρχουν στην επιστημονική κοινότητα πολλές έρευνες και διατριβές πάνω στο συγκεκριμένο τρόπο συμπίεσης η χρήση του παραμένει περιορισμένη. Η συγκεκριμένη έρευνα των Ferragina Venturini καταλήγουν στο ότι oBWT έχει νόημα μόνο όταν έχουμε μεγάλα αρχεία και χρησιμοποιούμε αλγόριθμους συμπίεσης που χρησιμοποιούν blocks χαρακτήρων και όχι μεμονωμένους χαρακτήρες. Ο Huffman είναι ένας από αυτούς, εντούτοις πρέπει να μετασχηματίζουμε πολύ μεγάλα blocks τα οποία χρειάζονται πολύ χρόνο για να μετασχηματιστούν. Αυτό ουσιαστικά είναι και ο λόγος μη χρησιμοποίησης του BWT αφού ο χρόνος μετασχηματισμού είναι πολύ μεγαλύτερος του χρόνου που κερδίζουμε από την διαφορά μεγέθους που έχουμε χρησιμοποιώντας τον ίδιο αλγόριθμο συμπίεσης χωρίς το BWT.

ΚΕΦΑΛΑΙΟ 4 – ΠΕΙΡΑΜΑΤΙΚΟ ΜΕΡΟΣ - ΜΕΤΡΗΣΕΙΣ

// Ο αλγόριθμος BWT

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if ( INT_MAX == 32767 )
#define BLOCK_SIZE 20000
#else
#define BLOCK_SIZE 200000
#endif
int indices[ BLOCK_SIZE + 1 ];
long size,length,counter=0;
unsigned char buffer[ BLOCK_SIZE ];
unsigned int T[ BLOCK_SIZE + 1 ];
int compare( const unsigned int *i1, const unsigned int *i2 );
void bwt();
int main(int argc, char *argv[])
{
 int c;
freopen( argv[ 1 ], "rb", stdin ); //diavazei to arxeio pou tou dothike san
                                    //parametros
```

```
char *output;
for (c=0;c<strlen(argv[ 1 ]);c++)
{
 if (argv[ 1 ][c]=='.') break;                //kratame to onoma tou arxeiou
}
output=(char*) malloc((c+5)*sizeof(char));
memcpy(output,argv[1],c);
output[c]='\0';
strcat(output,".bwt");
freopen( output, "wb", stdout );
fseek (stdin, 0, SEEK_END);
size=ftell (stdin);
rewind(stdin);

for ( ; ; )
{
   length = fread( (unsigned char *) buffer, 1, BLOCK_SIZE, stdin );

   if ( length == 0 )
   break;
   bwt();

}

fclose(stdout);
fclose(stdin);
fflush(stdout);
fflush(stdin);
fprintf(stderr,"length=%ld \n time=%ld",size,counter );
```

```
return 0;
}

// i synarthsi sygkrishs poy xrhsimopoihtai sthn qsort

int compare( const unsigned int *i1, const unsigned int *i2 )
{
    unsigned int l1 = (unsigned int) ( length - *i1 );
    unsigned int l2 = (unsigned int) ( length - *i2 );
    int result;
    result = memcmp( buffer + *i1,buffer + *i2,l1 < l2 ? l1 : l2 );
    if ( result == 0 )
        return l2 - l1;
    else
        return result;
};
//i vasiki sinartisi BWT
void bwt()
{
 long  first,last,i;
 fwrite( (char *) &length, 1, sizeof( long ), stdout );
 for ( i = 0 ; i <= length ; i++ )
     indices[ i ] = i; //orizoume enan pinaka deiktwn gia tous xaraktires tou
                       //buffer oi opoioi otan taksinomithoun tha mas deixnoun thn
                      //thesi tou taksinomimenou xarakthra ston buffer
 qsort( indices,(int)( length + 1 ),sizeof( int ),(int (*)(const void *, const void
*) )compare ); //sinartisi taksinomisis
//egrafi sto arxeio ths L stilis
for ( i = 0 ; i <= length ; i++ ) {
```

```
            if ( indices[ i ] == 1 )
               first = i;
            if ( indices[ i ] == 0 ) {
               last = i;
               fputc( '?', stdout );
            } else
              fputc( buffer[ indices[ i ] - 1 ], stdout );
              }
         // egrafi sto arxeio tis thesis tou prwtou xarakthra
         fwrite( (char *) &first, 1, sizeof( long ), stdout );
         // egrafi sto arxeio tis thesis tou teleutaiou xarakthra
         fwrite( (char *) &last, 1, sizeof( long ), stdout );

}
```

// Ο αντίστροφος αλγόριθμος BWT

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if ( INT_MAX == 32767 )
#define BLOCK_SIZE 20000
#else
#define BLOCK_SIZE 200000
#endif
long length;
 unsigned long first,last,l,counter=0;
unsigned char buffer[ BLOCK_SIZE ];
unsigned int T[ BLOCK_SIZE + 1 ]; //pinakas apokwdikopoiisis
```

```
unsigned int Count[ 257 ]; //pinakas metritwn sixnotitas xaraktirwn
unsigned int RunningTotal[ 257 ];  //pinakas atristikis sixnotitas xaraktirwn
void ubwt();
int main(int argc, char *argv[])
{
  int c;
freopen( argv[ 1 ], "rb", stdin );
char *output;
for (c=0;c<strlen(argv[ 1 ]);c++)
{
 if (argv[ 1 ][c]=='.') break;
}
output=(char*) malloc((c+5)*sizeof(char));
memcpy(output,argv[1],c);
output[c]='\0';
strcat(output,".bwu");
freopen( output, "wb", stdout );
for ( ; ; )
{    fread( (char *) &length, sizeof( long ), 1, stdin ); //anagnwsi tou mhkous
                                                          //tou buffer
    fread( (unsigned char *) buffer, 1, length+1, stdin ); //anagnwsi tou buffer
    //anagnwsi tou prwtoy xaraktira
    fread( (char *) &first, sizeof( long ), 1, stdin );
   //anagnwsi tou teleutaiou xaraktira
   fread( (char *)&last, sizeof( long ), 1, stdin );
    if ( length == 0 )
    break;
    ubwt(); //klish tis sinartisis
    fclose(stdout);
```

```
   fclose(stdin);
}


return 0;
}
void ubwt()
{
    unsigned int i,j;


      for ( i = 0 ; i < 257 ; i++ )
          Count[ i ] = 0;
//ipologismos sixnotitas xaraktirwn
 for ( i = 0 ; i <=  length ; i++ )
          if ( i == last )
             Count[ 256 ]++;
          else
             Count[ buffer[ i ] ]++;
        int sum = 0;
//ipologismos athristikis sixnotitas xaraktirwn
for ( i = 0 ; i < 257 ; i++ ) {
          RunningTotal[ i ] = sum;
          sum += Count[ i ];
          Count[ i ] = 0;
      }
    //ipologismos pinaka apokwdikopoiishs
  for ( i = 0 ; i <=length ; i++ ) {
          int index;
```

```
            if ( i == last )
               index = 256;
            else
               index = buffer[ i ];
            T[ Count[ index ] + RunningTotal[ index ] ] = i;
            Count[ index ]++;
         }
//egrafi  prototypou arxeiou
 i =first;
         for ( j = 0 ; j <  length   ; j++ ) {
             putc( buffer[ i ], stdout );
            i = T[ i ];
         }
      }
```

// Ο αλγόριθμος συμπίεσης Huffman

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>



#if ( INT_MAX == 32767 )
#define BLOCK_SIZE 20000
#else
#define BLOCK_SIZE 200000
#endif
// domh gia twn pinaka pou periexei tis suxnotites twn xaraktirwn
```

```
typedef struct hfreq_type{
                unsigned char c;
                unsigned long f;
};
//domi kombou tis listas pou periexei thn kwdikopoiish kathe xarakthra
typedef struct huffcode{
       unsigned char c;
       huffcode *next;
       };
//domh komboy tou Huffman dentrou
typedef struct huflist{
                unsigned char c;
                unsigned long f;
                huflist *next;
                huflist *r;
                huflist *l;
                huflist *parent;
                huffcode *code;

};




hfreq_type hfreq[256]={0};
huflist *huffman,*head,*list,*temp,*prev,*codetable[256];
huffcode *listcode,*templ;
long length,counter=0,size,sizecomp;
unsigned char buffer[ BLOCK_SIZE ];
```

```
char huffmanbuffer=0;
int huffmancounter=0;
int huftreecount=0;
void read_stats ( );
void sortchars();
void insertnode();
void fixcodetable( );
void compress(unsigned char ch);

int main(int argc, char *argv[])
{
int c = 0;
unsigned int i;
freopen( argv[ 1 ], "rb", stdin );
fseek (stdin, 0, SEEK_END);
size=ftell (stdin);
rewind(stdin);
char *output;
for (c=0;c<strlen(argv[ 1 ]);c++)
{
 if (argv[ 1 ][c]=='.') break;
}
output=(char*) malloc((c+5)*sizeof(char));
memcpy(output,argv[1],c);
output[c]='\0';
strcat(output,".huf");
freopen( output, "wb", stdout );
read_stats(  );
sortchars();
```

```
fixcodetable();
rewind(stdin);
for ( ; ; )
{
   length = fread( (unsigned char *) buffer, 1, BLOCK_SIZE, stdin );
   if ( length == 0 )break;
   for (i=0;i<length; i++)
   compress(buffer[i]);
}
if  (huffmancounter>0)
 fwrite(&huffmanbuffer,sizeof(huffmanbuffer),1,stdout);
fclose(stdin);
fseek (stdout, 0, SEEK_END);
sizecomp=ftell (stdout);
fclose(stdout);
fprintf(stderr,"length=%ld \ntime=%ld\nO=%lf\ncomp
length=%ld\n",size,counter,size*log(size)/log(2),sizecomp);
return 0;
}

//anagnwsi suxnotitwn twn xaraaktirwn tou arxeiou
void read_stats(  )
{
            int j,i;
            for (i=0;i<256;i++)
    {
     hfreq[ i ].c=i;
     hfreq[ i ].f=0;
    }
```

```
                rewind(stdin);
                while ( (j=getc(stdin)) != EOF ) {
                       hfreq[ j ].f++;
                }
}
//dhmiourgia dentrou Huffman
void sortchars()
{ int i;

     head=(huflist*)malloc(sizeof(huflist));
     head->next=NULL;



     for (i=0;i<256;i++)
      if (hfreq[ i ].f>0 )
       {
         prev=head;
         temp=head->next;
         while (temp!=NULL && hfreq[i].f>temp->f )
         {
          prev=temp;
          temp=temp->next;
         }
         huftreecount++;
         list=(huflist*)malloc(sizeof(huflist));
          list->f=hfreq[i].f;
         list->c=hfreq[i].c;
         list->r=NULL;
         list->l=NULL;
```

```
    prev->next=list;
    list->next=temp;
  }
fwrite( &huftreecount,sizeof(int),1,stdout);
 for (i=0;i<256;i++)
 {
     if (hfreq[ i ].f>0)
      fwrite(&hfreq[ i ] ,sizeof(hfreq_type),1,stdout);
     codetable[i]=NULL;
     }
temp=head->next;

 while (temp!=NULL)
   { codetable[temp->c]=temp;
     temp=temp->next; }



huffman=head->next;
while (huftreecount>1)
{
      list=(huflist*)malloc(sizeof(huflist));
      list->c=0;
      list->parent=NULL;
      list->f=head->next->f+head->next->next->f;
      if (head->next->f<head->next->next->f)
       {
        list->l=head->next;
        list->r=head->next->next;
       }
```

```
            else
            {
             list->r=head->next;
             list->l=head->next->next;
            }
            head->next->parent=list;
            head->next->next->parent=list;
            head->next=head->next->next->next;
           huftreecount--;
           prev=head;
           temp=head->next;
           while (temp!=NULL && list->f>temp->f )
           {
            prev=temp;
            temp=temp->next;
           }
           prev->next=list;
           list->next=temp;
      }
     }
//kwdikopoiish xarakthrwn
void fixcodetable()
{   huflist *temp1;
    temp1=huffman;
    while (temp1)
    {temp=temp1;
    temp1->code=(huffcode*)malloc(sizeof(huffcode));
    temp1->code->next=NULL;
      while (temp->parent)
```

```
  {    listcode=(huffcode*)malloc(sizeof(huffcode));
       if (temp->parent->l==temp)
       listcode->c=0;
        else
         listcode->c=1;


        listcode->next=temp1->code->next;
        temp1->code->next=listcode;
        temp=temp->parent;
    }
   temp1=temp1->next;
  }
 }
//sumpiesh arxeiou
void compress(unsigned char ch)
{
   templ=codetable[ch]->code->next;
   while (templ)
   { if (templ->c==1)
       huffmanbuffer |= 1 << huffmancounter;
       huffmancounter++;
       templ=templ->next;
       if (templ) counter++;
       if (huffmancounter>7)
        {
         fwrite(&huffmanbuffer,sizeof(huffmanbuffer),1,stdout);
         huffmanbuffer=0;
         huffmancounter=0;
         }
```

```
    }
}
```

// Ο αλγόριθμος αποσυμπίεσης Huffman

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if ( INT_MAX == 32767 )
#define BLOCK_SIZE 20000
#else
#define BLOCK_SIZE 200000
#endif

typedef struct hfreq_type{
        unsigned char c;
        unsigned long f;
};

typedef struct huflist{
        unsigned char c;
        unsigned long f;
        huflist *next;
        huflist *r;
        huflist *l;
        huflist *parent;
};
```

```
hfreq_type hfreq[256]={0},freq;
huflist *huffman,*head,*list,*temp,*prev,;

long length;
unsigned char buffer[ BLOCK_SIZE ];
int huftreecount=0;
void read_freq ( );
void sortchars();
void fixcodetable( );
void decompress(unsigned char ch);

int main(int argc, char *argv[])
{
int c = 0;
unsigned int i;
freopen( argv[ 1 ], "rb", stdin );
char *output;
for (c=0;c<strlen(argv[ 1 ]);c++)
{
 if (argv[ 1 ][c]=='.') break;
}
output=(char*) malloc((c+5)*sizeof(char));
memcpy(output,argv[1],c);
output[c]='\0';
strcat(output,".ori");
freopen( output, "wb", stdout );
```

```
read_freq(  );
sortchars();
temp=head->next;
for ( ; ; )
{
   length = fread( (unsigned char *) buffer, 1, BLOCK_SIZE, stdin );
   if ( length == 0 )break;
   for (i=0;i<length; i++)
   decompress(buffer[i]);
}

fclose(stdin);
fclose(stdout);
return 0;
}

//diavazei tis suxnothtes twn xarakthrwn apo to sumpiesmeno arxeio
void read_freq(  )
{
            int j,i;
            for (i=0;i<255;i++)
    {
      hfreq[ i ].c=i;
      hfreq[ i ].f=0;
    }
            rewind(stdin);
            fread( &huftreecount, sizeof(int), 1, stdin );
            for ( j = 0; j < huftreecount; j++ ){
                 fread( &freq, sizeof(hfreq_type), 1, stdin );
```

```
                        hfreq[ (unsigned char) freq.c ] = freq;
                }

        }
//dimiourgia dentrou Huffman
void sortchars()
{ int i;

     head=(huflist*)malloc(sizeof(huflist));
     head->next=NULL;



     for (i=0;i<256;i++)
      if (hfreq[ i ].f>0 )
       {
         prev=head;
         temp=head->next;
         while (temp!=NULL && hfreq[i].f>temp->f )
         {
          prev=temp;
          temp=temp->next;
         }
         list=(huflist*)malloc(sizeof(huflist));
          list->f=hfreq[i].f;
         list->c=hfreq[i].c;
         list->r=NULL;
         list->l=NULL;
         prev->next=list;
         list->next=temp;
```

```
  }
huffman=head->next;
while (huftreecount>1)
{
       list=(huflist*)malloc(sizeof(huflist));
       list->c=0;
       list->parent=NULL;
       list->f=head->next->f+head->next->next->f;
       if (head->next->f<head->next->next->f)
        {
         list->l=head->next;
         list->r=head->next->next;
        }
        else
        {
         list->r=head->next;
         list->l=head->next->next;
        }
        head->next->parent=list;
        head->next->next->parent=list;
        head->next=head->next->next->next;
       huftreecount--;
       prev=head;
       temp=head->next;
       while (temp!=NULL && list->f>temp->f )
       {
        prev=temp;
        temp=temp->next;
       }
```

```
            prev->next=list;
            list->next=temp;
      }
     }
//apokwdikopoiisi xarakthrwn
void decompress(unsigned char ch)
{ int i;
 for (i=0;i<8;i++)
 {
    if (ch & (1 << i))
     temp=temp->r;
     else
     temp=temp->l;
     if ((temp->l==NULL) && (temp->r==NULL))
      {
        fwrite(&temp->c,sizeof(char),1,stdout);
        temp=head->next;

                    }
 }

}
```

Μετρήσεις:

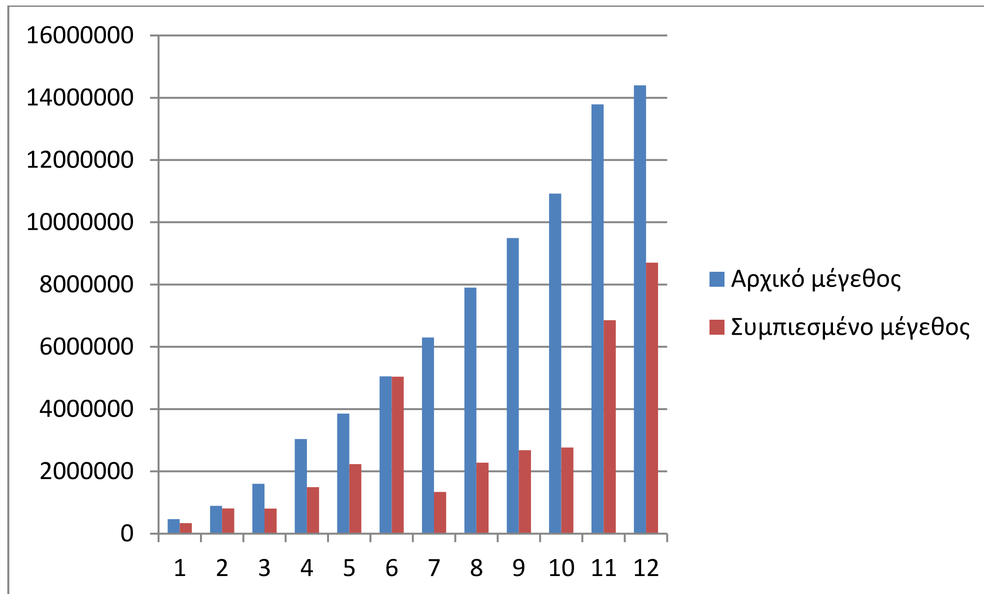
Για να πραγματοποιηθούν οι μετρήσεις χρησιμοποιήθηκαν κάποια αρχεία (tests) που η κατάληξη τους ήταν ως επί το πλείστον .doc (αρχεία Microsoft Word). Η επιλογή αυτής της κατάληξης δεν έγινε τυχαία αφού τα αρχεία Word έχουν μεγάλα περιθώρια συμπίεσης οπότε και τα ποσοστά συμπίεσης τους

(Πίνακας 1) ήταν ικανοποιητικά. Αντίθετα άλλα αρχεία τύπου .JPEG ή .MOV δεν είχαν καλό ποσοστό συμπίεσης επειδή αυτά έχουν υποστεί εκ των προτέρων κάποια μορφή συμπίεσης. Τα input tests είναι κυρίως τυχαία κείμενα από το internet καθώς και δικά μου αρχεία word που με βοήθησαν για την παραγωγή των μετρήσεων.

Tests	Αρχικό μέγεθος	Συμπιεσμένο μέγεθος	Χρόνος	Όριο πολυπλοκότητας	Ποσοστά συμπίεσης
1	467456	338894	2227276	8804287	27,50%
2	890368	809870	5572171	17597271	9,04%
3	1601024	803182	4808014	32998007	49,83%
4	3037696	1495739	8911798	65415404	50,76%
5	3857920	2235256	14007708	84408943	42,06%
6	5050880	5041565	35265218	112473518	0,18%
7	6297600	1340611	4410868	142239928	78,71%
8	7901184	2279983	10322258	181044865	71,14%
9	9494528	2679025	11921251	220070482	71,78%
10	10921984	2766033	11189857	255363977	74,67%
11	13788672	6853124	41019904	327025661	50,30%
12	14399488	8703288	55210393	342412829	39,56%

Πίνακας 1.

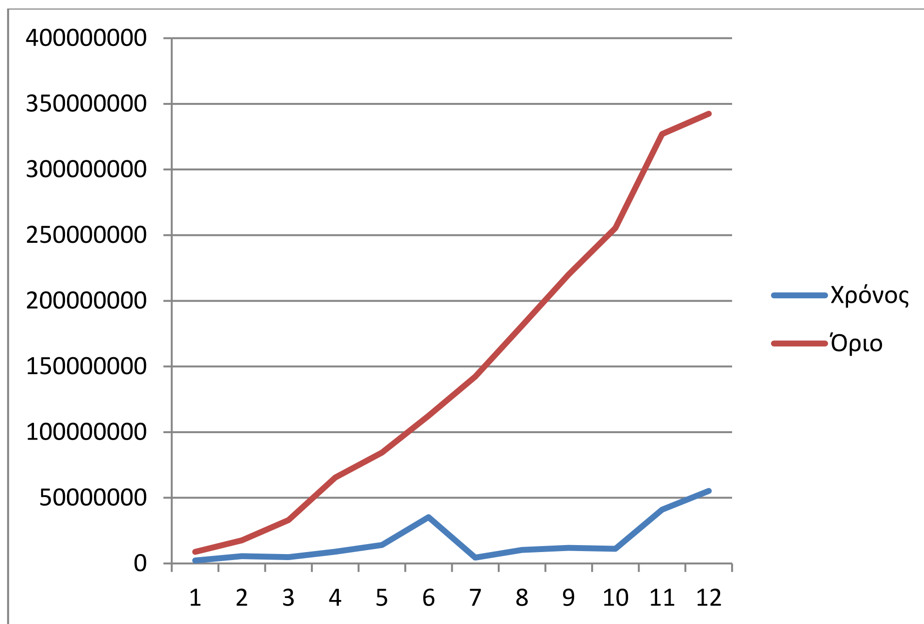
*1. Παρακάτω εμφανίζεται το γράφημα των **συμπιεσμένων αρχείων** σε σχέση με το αρχικό του μέγεθος:*



Γράφημα 1.

Η συμπίεση των δεδομένων όπως φαίνεται και στην στήλη "ποσοστά συμπίεσης" του πίνακα 1 είναι σε ικανοποιητικό επίπεδο. Στις περιπτώσεις του Test 2 και Test 6 παρατηρούμε ότι είναι πολύ κοντά το συμπιεσμένο με το αρχικό μέγεθος και αυτό οφείλεται γιατί εμφανίζονται πολλά στοιχεία που ήταν ήδη συμπιεσμένα (κυρίως εικόνες .JPEG)καθώς και πολλά στοιχεία με ίδια συχνότητα. Έτσι το δένδρο του Huffman είναι μεγαλύτερο με αποτέλεσμα και η κωδικοποίηση των στοιχείων είναι μεγαλύτερη οπότε χρειάζεται και περισσότερο χώρο.

*2.Παρακάτω εμφανίζεται το γράφημα **χρόνου** του αλγορίθμου:*



Γράφημα 2.

Βλέπουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου (μπλε γραφική παράσταση) είναι κάτω από το Όριο της πολυπλοκότητας O(nlogn) (κόκκινη γραφική παράσταση) και όσο αυξάνονται τα μεγέθη των αρχείων που συμπιέζουμε τόσο μεγαλώνει η διαφορά του Χρόνου των μετρήσεων από το Όριο.

Βιβλιογραφία

[1] A simple storage scheme for strings achieving entropy bounds, Paolo Ferragina, Rossano Venturini, Italy, 2006.

[2] Π.Δ. Μποζάνης, Δομές Δεδομένων, Βόλος, 2006.

[3] Π.Δ. Μποζάνης, Αλγόριθμοι, Βόλος, 2005.

[4] The Burrows-Wheeler Transform: Theory and Practice, Italy, Giovanni Manzini.

[5] Burrows Wheeler Transformation, Geissberger Jay, TU Wien.

[6] Data Compression with the Burrows-Wheeler Transform, Mark Nelson,1996.
http://marknelson.us/1996/09/01/bwt/

[7] Συμπίεση δεδομένων (Γενικές πληροφορίες)
http://www.it.uom.gr/project/mycomputer/storage/optic/dvd/compress.html .

[8] http://el.wikipedia.org/wiki/Mp3

[9]http://www.cs.princeton.edu/courses/archive/spr03/cs226/assignments/burrows.html