ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2009-10)

Άσκηση 4

Είναι πολλές φορές χρήσιμο όταν διαχειριζόμαστε μεγάλους όγκους δεδομένων, που είτε πρέπει να φυλαχθούν κάπου, π.χ. σε ένα δίσκο, είτε να μεταφερθούν μέσω κάποιας δικτυακής τεχνολογίας, να έχουμε συμπιέσει τα δεδομένα αυτά, ώστε να έχουν μικρότερο συνολικό μέγεθος. Στη βιβλιογραφία, υπάρχουν πολλοί αλγόριθμοι συμπίεσης δεδομένων. Στις περισσότερες περιπτώσεις, μας ενδιαφέρει ο αλγόριθμος συμπίεσης να είναι *χωρίς απώλειες* (lossless), δηλαδή να είναι δυνατόν με έναν αλγόριθμο αποσυμπίεσης να ανακτήσουμε ακριβώς τα αρχικά δεδομένα από τη συμπιεσμένη μορφή τους.

Στην άσκηση αυτή καλείσθε να υλοποιήσετε μία παραλλαγή του lossless αλγορίθμου συμπίεσης LZW (Lempel-Ziv-Welch, από τα ονόματα αυτών που τον πρότειναν). Πριν δώσουμε αναλυτικά τις προδιαγραφές του προγράμματος που πρέπει να υλοποιήσετε, ας περιγράψουμε πρώτα αυτόν τον αλγόριθμο συμπίεσης.

Ο αλγόριθμος LZW διαβάζει την είσοδο byte-byte και δίνει στην έξοδο κωδικούς που αντιστοιχούν ο καθένας είτε σε ένα byte της εισόδου είτε σε ακολουθίες από συνεχόμενα bytes. Για να μπορέσει να διαχειριστεί ο αλγόριθμος τους κωδικούς, συντηρεί ένα λεξικό. Αρχικά, το λεξικό περιέχει 257 κωδικούς. Οι κωδικοί από το 0 έως το 255 αντιστοιχούν, ο καθένας, στο byte με την τιμή αυτή. Υπάρχει και ένας ειδικός κωδικός, το 256, που ονομάζεται *κωδικός εκκαθάρισης* (clear code), του οποίου ο ρόλος θα γίνει φανερός στη συνέχεια. Στο διπλανό σχήμα, φαίνεται η μορφή του λεξικού με το οποίο ξεκινά ο αλγόριθμος. Στα αριστερά είναι οι κωδικοί και, δεξιά, μέσα στον πίνακα, φαίνονται τα bytes στα οποία αντιστοιχεί ο κάθε κωδικός. **Προσοχή**: *Το σχήμα δεν υπονοεί ότι αν κάποιος θέλει να υλοποιήσει το λεξικό του αλγορίθμου συμπίεσης, πρέπει να το κάνει μέσω πίνακα. Για την ακρίβεια, μία τέτοια υλοποίηση δεν είναι καθόλου καλή, γιατί δεν διευκολύνει τις αναζητήσεις που θα χρειαστεί να γίνουν μέσα στο λεξικό.*

0	'\000'
1	'
2	'\002'
.....	.....
10	'\012' (= 10 = '\n')
.....	.....
65	'\101' (= 65 = 'A')
66	'\102' (= 66 = 'B')
67	'\103' (= 67 = 'C')
.....	.....
255	'\377'
256	clear code
.....	.....

Αφού το λεξικό έχει αρχικά 257 κωδικούς, χρειάζονται 9 bits για την αναπαράσταση καθενός από αυτούς (δεδομένου ότι $2^8 = 256$). Με 9 bits μπορούμε να έχουμε συνολικά 2^9 $(= 512)$ κωδικούς. Όσο ο αλγόριθμος δεν χρειάζεται περισσότερους από 512 κωδικούς, στην έξοδό του βγάζει κωδικούς των 9 bits. Αν χρειαστεί και 513ος κωδικός, οι κωδικοί από το σημείο αυτό και μετά θα είναι των 10 bits. Με αυτό το μέγεθος θα βγαίνουν στην έξοδο όλοι οι κωδικοί πλέον, ακόμα και αυτοί που θα τους αρκούσαν 9 bits (αυτοί θα έχουν ένα αρχικό bit 0). Τα 10 bits επαρκούν για κωδικούς μέχρι και το 1024. Από το 1025 και μετά, θέλουμε 11 bits. Αν ο αλγόριθμος αναβαθμίσει το λεξικό από τα 10 στα 11 bits, και οι κωδικοί στην έξοδο θα βγαίνουν, από το σημείο αυτό και μετά, με 11 bits. Ομοίως, μπορεί να γίνει αναβάθμιση από 11 σε 12, κ.ο.κ. Στην παραλλαγή του αλγορίθμου που θα υλοποιήσετε, μπορούμε να έχουμε κωδικούς μέχρι και *maxbits*. Το *maxbits* θα είναι παράμετρος του αλγορίθμου και θα μπορεί να κυμαίνεται από 9 έως 16. Επειδή η πληροφορία αυτή χρειάζεται στον αλγόριθμο αποσυμπίεσης, τα 3 πρώτα bits στην έξοδο, έστω ο αριθμός x (από 0 έως 7), θα περιέχουν την πληροφορία του *maxbits*, το οποίο θα θεωρείται ότι είναι ίσο με $x + 9$.

Ας δούμε τώρα τη λειτουργία του αλγορίθμου. Διαβάζονται από την είσοδο bytes, μέχρις ότου με το που θα διαβαστεί ένα byte, έστω B, η μέχρι στιγμής ακολουθία από bytes, έστω SB, να μην περιέχεται στο λεξικό, ενώ αυτή χωρίς το τελευταίο byte, δηλαδή η S, να περιέχεται. Αυτό που κάνει ο αλγόριθμος είναι να βγάλει στην έξοδο τον κωδικό που υπάρχει στο λεξικό για την S (με πλήθος bits όσο είναι αυτό που ισχύει για τον αλγόριθμο τη στιγμή αυτή), να βάλει την SB στο λεξικό, δίνοντάς της σαν κωδικό τον πρώτο διαθέσιμο (πιθανώς να χρειαστεί και αναβάθμιση του λεξικού με

προσθήκη ενός bit στους κωδικούς) και, στη συνέχεια, επαναλαμβάνεται η διαδικασία με τρέχουσα ακολουθία το B. Σε ψευδογλώσσα, θα γράφαμε αυτόν τον αλγόριθμο ως εξής:

S = διάβασε byte από την είσοδο
ενόσω δεν φτάσαμε στο τέλος της εισόδου
    B = διάβασε byte από την είσοδο
    αν το SB υπάρχει στο λεξικό, τότε
        $S = SB$
    αλλιώς
        βγάλε στην έξοδο τον κωδικό του S
        πρόσθεσε στο λεξικό το SB
        $S = B$
βγάλε τον κωδικό του S στην έξοδο

Ένα παράδειγμα λειτουργίας του αλγορίθμου δίνεται στη συνέχεια. Έστω ότι στην είσοδο δίνονται τα bytes `'A'`, `'B'`, `'A'`, `'B'`, `'C'` και `'\n'`. Υπενθυμίζεται ότι αρχικά το λεξικό περιέχει τους κωδικούς 0 έως 255, καθέναν για το αντίστοιχο byte, συν τον κωδικό εκκαθάρισης (256).

Η είσοδος, λοιπόν, του αλγορίθμου είναι η:

```
  'A'      'B'      'A'      'B'      'C'      '\n'
01000001 01000010 01000001 01000010 01000011 00001010
```

Θεωρούμε ότι έχουμε *maxbits* ίσο με 16, οπότε τα τρία πρώτα bits στην έξοδο θα είναι τα 111.

Αρχικά ο αλγόριθμος διαβάζει το `'A'`, δηλαδή το 01000001, το οποίο υπάρχει στο λεξικό με κωδικό (των 9 bits) τον 001000001. Μετά, διαβάζει το `'B'`, δηλαδή το 01000010. Όμως, η ακολουθία από bytes `'A''B'` δεν υπάρχει στο λεξικό. Βγαίνει, λοιπόν, στην έξοδο ο 9-μπιτος κωδικός του `'A'` (001000001), μπαίνει το `'A''B'` στο λεξικό με κωδικό το 257 (σε 9 bits 100000001) και συνεχίζει ο αλγόριθμός με τρέχον το `'B'`. Μέχρι στιγμής η έξοδος είναι:

```
      'A'
111 001000001
```

Μετά το τρέχον `'B'`, διαβάζεται το `'A'`. Η ακολουθία `'B''A'` δεν περιέχεται στο λεξικό, άρα βγαίνει ο 9-μπιτος κωδικός του `'B'` στην έξοδο (001000010), μπαίνει το `'B''A'` στο λεξικό με κωδικό 258 (σε 9 bits 100000010) και τρέχον είναι το `'A'`. Η μέχρι στιγμής έξοδος:

```
      'A'       'B'
111 001000001 001000010
```

Μετά το τρέχον `'A'`, διαβάζεται το `'B'`. Το `'A''B'` περιέχεται στο λεξικό, άρα γίνεται το νέο τρέχον. Διαβάζεται το `'C'`. Η ακολουθία `'A''B''C'` δεν περιέχεται στο λεξικό, άρα βγαίνει στην έξοδο ο 9-μπιτος κωδικός του `'A''B'` (100000001), μπαίνει το `'A''B''C'` στο λεξικό με κωδικό 259 (σε 9 bits 100000011) και τρέχον είναι το `'C'`. Η μέχρι στιγμής έξοδος:

```
      'A'       'B'      'A''B'
111 001000001 001000010 100000001
```

Μετά το τρέχον `'C'`, διαβάζεται το `'\n'`. Το `'C''\n'` δεν περιέχεται στο λεξικό, άρα βγαίνει ο 9-μπιτος κωδικός του `'C'` στην έξοδο (001000011), μπαίνει το `'C''\n'` στο λεξικό με κωδικό 260 (σε 9 bits 100000100) και τρέχον είναι το `'\n'`. Δεν υπάρχει όμως άλλο byte στην είσοδο, οπότε βγαίνει στην έξοδο και ο 9-μπιτος κωδικός του `'\n'` (000001010). Τελικά, η έξοδος είναι:

```
        'A'       'B'       'A''B'      'C'       '\n'
  111 001000001 001000010 100000001 001000011 000001010
```

Στο προηγούμενο παράδειγμα, στην έξοδο βγήκαν 48 bits, δηλαδή ακριβώς 6 bytes. Αν ο αριθμός των bits στην έξοδο δεν είναι πολλαπλάσιο του 8, τότε θα προστεθεί και ο κατάλληλος αριθμός από 0, ώστε να προκύπτει ακέραιο πλήθος από bytes.

Στο παραπάνω παράδειγμα, που δόθηκε απλώς για επίδειξη της λειτουργίας του αλγορίθμου, βγήκαν στην έξοδο 6 bytes, όσα ακριβώς είχαν δοθεί και στην είσοδο. Δηλαδή, ο αλγόριθμος δεν έκανε καμία απολύτως συμπίεση. Είναι ευνόητο όμως, ότι σε μεγαλύτερες εισόδους, στις οποίες θα τυχαίνει πολλές ακολουθίες από bytes να επαναλαμβάνονται, τότε θα γίνεται συμπίεση, γιατί αντί για κάποιο, πιθανώς μεγάλο, πλήθος bytes της εισόδου θα βγαίνει στην έξοδο απλώς ένας κωδικός (των 16 bits το πολύ).

Επίσης, στο παραπάνω παράδειγμα, λόγω του μικρού μεγέθους της εισόδου, δεν εξαντλήθηκαν οι 9-μπιτοι κωδικοί. Αν όμως χρειαζόμασταν και τον κωδικό 513, τότε οι κωδικοί θα γινόντουσαν των 10 bits, αν θέλαμε και το 1025, θα πηγαίναμε στα 11 bits, κ.ο.κ.

Στο διπλανό σχήμα, φαίνεται η μορφή του λεξικού μετά την επεξεργασία από τον αλγόριθμο LZW της εισόδου που του δόθηκε, δηλαδή τα bytes `'A'`, `'B'`, `'A'`, `'B'`, `'C'` και `'\n'`. **Προσοχή**: *Επισημαίνεται για μία ακόμα φορά ότι η υλοποίηση του λεξικού μέσω πίνακα δεν είναι η καλύτερη δυνατή. Χρειάζεται να υιοθετηθεί κάποια δομή δεδομένων που να επιταχύνει τις αναζητήσεις στο λεξικό.*

0	'\000'
1	'
2	'\002'
.....	.....
10	'\012' (= 10 = '\n')
.....	.....
65	'\101' (= 65 = 'A')
66	'\102' (= 66 = 'B')
67	'\103' (= 67 = 'C')
.....	.....
255	'\377'
256	clear code
257	'A' 'B'
258	'B' 'A'
259	'A' 'B' 'C'
260	'C' '\n'
.....	.....

Όσον αφορά τον κωδικό εκκαθάρισης, μπορεί ο αλγόριθμος να αποφασίσει κάποια στιγμή να αρχικοποιήσει το λεξικό. Αυτό επιτρέπεται να το κάνει όταν το μέγεθος του λεξικού ισούται με κάποια δύναμη του 2. Θα αρχικοποιήσει το λεξικό, αν δει ότι δεν πετυχαίνει συμπίεση. Ένα κριτήριο για να το αποφασίσει αυτό είναι το εξής. Αν το αποτέλεσμα της διαίρεσης του πλήθους των bits που έχει βγάλει στην έξοδο διά 8, από την αρχή ή από την τελευταία έξοδο κωδικού εκκαθάρισης (εξαιρουμένων των 3 πρώτων bits που κωδικοποιούν το *maxbits* ή του κωδικού εκκαθάρισης, αντίστοιχα) είναι μεγαλύτερο από το πλήθος των bytes που έχει διαβάσει στο ίδιο διάστημα, τότε, ουσιαστικά, δεν γίνεται συμπίεση, οπότε βγάζει στην έξοδο τον κωδικό εκκαθάρισης (256), με το τρέχον πλήθος bits, και επαναφέρει το λεξικό στην κατάσταση με τους 9-μπιτους κωδικούς από 0 έως 256.

Θα πρέπει να σημειωθεί ότι ο λόγος που είναι καλό ο αλγόριθμος να κάνει αρχικοποίηση του λεξικού είναι για τις περιπτώσεις που προσπαθεί να συμπιέσει κάποια είσοδο που είναι κάπως "ιδιόρρυθμη", με αποτέλεσμα, αντί να καταφέρει να κάνει συμπίεση, στην έξοδό του να βγάζει μεγαλύτερο όγκο δεδομένων από αυτόν που διαβάζει. Αν συμβαίνει κάτι τέτοιο, μπορεί με μία αρχικοποίηση του λεξικού κάποια στιγμή και υπό την προϋπόθεση ότι η είσοδος δεν θα είναι και στη συνέχεια "ιδιόρρυθμη", να καταφέρει ο αλγόριθμος να "αναστηθεί" και να επιτυγχάνει συμπίεση για την είσοδο που θα ακολουθήσει. Με άλλα λόγια, η αρχικοποίηση του λεξικού, μέσω του κωδικού εκκαθάρισης, δεν είναι απαραίτητη για την ορθότητα του αποτελέσματος, αλλά για την ποιότητά του. Είτε κάνει ο αλγόριθμος συμπίεσης αρχικοποίηση του λεξικού είτε όχι, ο αντίστοιχος αλγόριθμος αποσυμπίεσης θα οφείλει να παράγει την αρχική μη συμπιεσμένη είσοδο. Υπάρχουν διάφορες πολιτικές αρχικοποίησης του λεξικού. Στην προηγούμενη παράγραφο περιγράψαμε μία από τις πιθανές πολιτικές. Μπορείτε είτε να την ακολουθήσετε, είτε να επιλέξετε κάποια άλλη. Αν ακολουθήσετε άλλη πολιτική, απλά να προσπαθήσετε να είναι από τις καλές πολιτικές, όπως αυτή που προτείνεται. Απλώς, έχετε υπόψη σας ότι στις ενδεικτικές εκτελέσεις που θα δοθούν στη συνέχεια, το πρόγραμμα εφαρμόζει την πολιτική που περιγράψαμε. Οπότε, μόνο αν υλοποιήσετε ακριβώς την ίδια, θα μπορείτε να συγκρίνετε τα αποτελέσματα του δικού σας προγράμματος με αυτά που δίνονται εδώ και να επαληθεύσετε ότι είναι

σωστά. Σε κάθε περίπτωση, πάντως, όποια πολιτική αρχικοποίησης και αν εφαρμόσετε, αν το κάνετε σωστά, με την αποσυμπίεση, θα πρέπει να παίρνετε πάλι την αρχική είσοδο.

Αν γράφαμε πάλι τον αλγόριθμο συμπίεσης που δόθηκε νωρίτερα, ώστε να συμπεριλάβουμε και τις διαδικασίες αναβάθμισης των κωδικών του λεξικού κατά ένα bit και της αρχικοποίησης του λεξικού, όταν κρίνεται απαραίτητο με βάση το κριτήριο που αναφέρθηκε, θα το κάναμε ως εξής:

$nbits = 9$
$S =$ διάβασε byte από την είσοδο
ενόσω δεν φτάσαμε στο τέλος της εισόδου
    $B =$ διάβασε byte από την είσοδο
    αν το SB υπάρχει στο λεξικό, τότε
        $S = SB$
    αλλιώς
        βγάλε στην έξοδο τον κωδικό του S με $nbits$
        αν το μέγεθος του λεξικού ισούται με 2^{nbits}
            αν το $nbits$ είναι διάφορο του $maxbits$
                αύξησε το $nbits$ κατά 1
            αν δεν πετυχαίνουμε συμπίεση (με βάση κάποιο κριτήριο)
                βγάλε στην έξοδο τον κωδικό εκκαθάρισης
                διάγραψε όλους τους κωδικούς του λεξικού μετά τον 257ο
                $nbits = 9$
        αν το λεξικό δεν είναι μεγέθους $2^{maxbits}$
            πρόσθεσε στο λεξικό το SB
        $S = B$
βγάλε τον κωδικό του S στην έξοδο

Το πρόγραμμα που θα γράψετε πρέπει να υποστηρίζει τις εξής επιλογές:

`-i <input_file>` : Η είσοδος δίνεται από το `<input_file>`. Αν δεν δοθεί η επιλογή, το πρόγραμμα διαβάζει από την πρότυπη είσοδο (stdin).

`-o <output_file>` : Η έξοδος βγαίνει στο `<output_file>`. Αν δεν δοθεί η επιλογή, το πρόγραμμα εκτυπώνει στην πρότυπη έξοδο (stdout).

`-b <maxbits>` : Ορίζεται ο μέγιστος αριθμός bits για τους κωδικούς του λεξικού (από 9 έως 16). Αν δεν δοθεί η επιλογή, η default τιμή είναι 16.

`-p` : Εκτυπώνεται στην πρότυπη έξοδο για διαγνωστικά μηνύματα (stderr) το ποσοστό συμπίεσης.

`-d` : Γίνεται αποσυμπίεση της εισόδου (μη συμβατή επιλογή με τις `-b` και `-p`). Αν δεν δοθεί η επιλογή, γίνεται συμπίεση. **Προσοχή**: *Η επιλογή αυτή είναι προαιρετικό να υλοποιηθεί. Αν γίνει, θα αντιστοιχεί σε bonus 50% επιπλέον της μέγιστης βαθμολογίας της άσκησης. Σημειώστε ότι στην εκφώνηση δεν σας δίνεται σαφώς διατυπωμένος ο αλγόριθμος αποσυμπίεσης, αλλά δεν ιδιαίτερα δύσκολο να συναχθεί αυτός από τον αλγόριθμο συμπίεσης που περιγράφηκε αναλυτικά.*

Ίσως είναι προφανές ότι θα αντιμετωπίσετε το εξής πρόβλημα στην υλοποίηση του αλγορίθμου συμπίεσης. Ενώ στην έξοδο το πλέον στοιχειώδες μέγεθος δεδομένων που μπορούμε να έχουμε είναι το byte, το πρόγραμμά σας θα πρέπει να "εκτυπώνει" bits. Αυτό μπορεί να υλοποιηθεί μέσω ενός buffer που θα χρησιμοποιείται για την έξοδο bits και το οποίο θα εκτυπώνεται πραγματικά στην έξοδο όταν χρειάζεται. Αντίστοιχη λειτουργία θα πρέπει να γίνει και στην περίπτωση που υλοποιήσετε και την προαιρετική αποσυμπίεση, για την ανάγνωση από την είσοδο σε επίπεδο bits. Για

να διευκολυνθείτε, σας παρέχεται έτοιμη μία βιβλιοθήκη (σε αντικειμενική μορφή) για τη διαχείριση του buffer από bits που θα χρειαστείτε. Μπορείτε να βρείτε τη βιβλιοθήκη αυτή κάτω από το `http://www.di.uoa.gr/~ip/hwfiles/lzw` με όνομα "`bit_io_<arch>.o`", όπου το `<arch>` μπορεί να είναι `solaris`, `linux`, `windows` ή `macosx`, ανάλογα με το σύστημα που σας ενδιαφέρει να δουλέψετε. Το αρχείο επικεφαλίδας της βιβλιοθήκης είναι το "`bit_io.h`", στο οποίο περιγράφονται αναλυτικά οι προδιαγραφές των υλοποιημένων συναρτήσεων για τη διαχείριση του buffer.

Θα ήταν ενδιαφέρον και χρήσιμο να δοκιμάσετε να υλοποιήσετε την παραπάνω βιβλιοθήκη διαχείρισης του buffer σε επίπεδο bits, είτε ακριβώς με τις προδιαγραφές που περιγράφονται, είτε σε κάποια παραλλαγή τους και να χρησιμοποιήσετε τη δική σας υλοποίηση, αντί για αυτήν που σας δίνεται.[1]

Για διευκόλυνσή σας στον έλεγχο ορθότητας της υλοποίησης του αλγορίθμου συμπίεσης που σας ζητείται, στο `http://www.di.uoa.gr/~ip/hwfiles/lzw` μπορείτε να βρείτε σε εκτελέσιμη μορφή μία υλοποίηση του ζητουμένου προγράμματος (συμπεριλαμβανομένης και της επιλογής `-d`), με όνομα "`lzwzip_<arch>`", όπου το `<arch>` είναι `solaris`, `linux`, `windows.exe` ή `macosx`, ανάλογα με το σύστημα που σας ενδιαφέρει.[2]

Κάποιες ενδεικτικές εκτελέσεις του ζητούμενου προγράμματος (έστω ότι το εκτελέσιμο ονομάζεται "`lzwzip`") φαίνονται στη συνέχεια.[3] Το πρόγραμμα "`bd`" που φαίνεται στις εντολές εμφανίζει στην πρότυπη έξοδο (stdout) σε δυαδική μορφή τα bytes που δέχεται από την πρότυπη είσοδο (stdin), αν κληθεί χωρίς ορίσματα, ή από το αρχείο που του δίνεται σαν το μοναδικό όρισμα κατά την κλήση του. Σε κάθε γραμμή της εξόδου, το "`bd`" εμφανίζει 8 bytes (το πολύ) σε δυαδική μορφή, ενώ στην αρχή της οποίας φαίνεται σε δεκαεξαδική μορφή η θέση του πρώτου από αυτά. Και το πρόγραμμα αυτό μπορείτε να το βρείτε σε εκτελέσιμη μορφή στο `http://www.di.uoa.gr/~ip/hwfiles/lzw`, με όνομα "`bd_<arch>`", όπου το `<arch>` είναι `solaris`, `linux`, `windows.exe` ή `macosx`, ανάλογα με το σύστημά σας. Στην ίδια θέση μπορείτε να βρείτε και τα τυχόν αρχεία με δεδομένα που χρειάζονται στις ενδεικτικές εκτελέσεις. Σημειώστε ότι τα αρχεία κειμένου που σας διατίθενται είναι σε Unix format. Δηλαδή, οι αλλαγές γραμμής κωδικοποιούνται απλώς με τον χαρακτήρα `'\012'`. Αν τα αρχεία αυτά μεταφερθούν σε Windows με κάποιο πρόγραμμα μεταφοράς (π.χ. WinSCP) σε text mode, τότε τα μεταφερθέντα αρχεία δεν θα είναι πανομοιότυπα με τα αρχικά, γιατί στα Windows οι αλλαγές γραμμής κωδικοποιούνται στα αρχεία κειμένου με δύο χαρακτήρες, τους `'\015'` και `'\012'`. Αυτό σημαίνει ότι τα αποτελέσματα του προγράμματος "`lzwzip`" στα Windows δεν θα συμφωνούν απόλυτα με αυτά που φαίνονται στις ενδεικτικές εκτελέσεις στη συνέχεια.

```
% ./lzwzip -h
Usage: ./lzwzip [OPTION] ...

Compress or decompress files using LZW algorithm

  -h, type this help message
  -d, decompress (else execute compress)
  -b, set max bits for dictionary codes (for compress)
  -p, type compression ratio at standard error (for compress)
```

[1] Δεν υπάρχει βαθμολογικό bonus για την υλοποίηση της βιβλιοθήκης, αλλά μόνο το bonus της ικανοποίησης που θα αισθανθείτε αν γράψετε τη δική σας εκδοχή που να δουλεύει σωστά, κάτι που είναι, ομολογουμένως, αρκετά δύσκολο για το συγκεκριμένο πρόβλημα!

[2] Στο εκτελέσιμο πρόγραμμα "`lzwzip_<arch>`" που σας δίνεται έχουν υλοποιηθεί και οι επιλογές "`-v`" (για να εκτυπώνεται διαγνωστικό μήνυμα όταν μεταβάλλεται το πλήθος των bits των κωδικών του λεξικού και όταν γίνεται αρχικοποίησή του), "`-n`" (για να απενεργοποιείται η δυνατότητα αρχικοποίησης του λεξικού) και "`-h`" (για μία συνοπτική εκτύπωση των διαθέσιμων επιλογών του προγράμματος). Τις επιλογές αυτές δεν είναι απαραίτητο να τις υλοποιήσετε και εσείς στο δικό σας πρόγραμμα.

[3] Το `%` που φαίνεται στην αρχή κάθε εντολής **δεν είναι τμήμα της εντολής**. Είναι απλώς το λεγόμενο prompt του λειτουργικού που ζητά από τον χρήστη να πληκτρολογήσει την εντολή του.

```
  -n, disable clear code (for compress)
  -v, type code bit resizes at standard error (for compress)
  -i file, give input file (else reads from standard input)
  -o file, give output file (else writes at standard output)

%

% echo ABABC
ABABC
% echo ABABC | ./bd
000000: 01000001 01000010 01000001 01000010 01000011 00001010
% echo ABABC | ./lzwzip > ABABC.lzw
% ./bd ABABC.lzw
000000: 11100100 00010010 00010100 00000100 10000110 00001010
% ./lzwzip -d < ABABC.lzw
ABABC
%

% ./lzwzip -p -i KRexcerpt.txt -o Compressed.lzw
Achieved compression of 25.41%
% ls -l KRexcerpt.txt
-rw-r-----   1 ip       www          484 Nov 21 17:27 KRexcerpt.txt
% ls -l Compressed.lzw
-rw-r-----   1 ip       www          361 Dec  7 19:50 Compressed.lzw
% ./bd Compressed.lzw
000000: 11100100 00110001 00000001 10100100 11100110 00100000 00110000 10001000
000008: 00001100 11100110 01010011 01110001 10010100 11100100 01100001 00110110
000010: 00001011 01001110 00000111 01010011 10010001 11000000 11011110 01110011
000018: 00110010 10001000 00001110 00000111 00100011 01111001 10011110 00011000
000020: 01101101 00110110 10011010 01001101 11000110 01110001 00000001 10110000
000028: 11000011 00100000 00111010 10011000 01100001 00000010 11100001 00000001
000030: 00100100 11101000 00100000 00110100 00011000 01100000 10000110 00100011
000038: 00101100 00100100 01000000 01100011 00110110 01000101 00001100 10100110
000040: 11000011 11001000 00101010 01011110 01110011 00110111 10011000 11001101
000048: 00100110 00010011 10100001 10010100 11001000 00100000 00111011 10011010
000050: 01001110 10000110 10000001 00000101 00100010 00101100 01010101 00100111
000058: 00010010 01001011 00000010 00000011 10011001 11100100 11100111 01000010
000060: 00110110 11010001 01001101 00010000 10111000 10110101 00011110 10001011
000068: 00101111 00010000 00011001 00001100 10100111 01101001 11000001 10111100
000070: 11100001 01000011 00010110 01010100 01100011 11100110 00111000 10110001
000078: 10001000 11011111 01001000 00000101 01010010 11101010 00110101 00111010
000080: 10101100 00010100 11011101 01000100 00110110 11000101 00100101 10000110
000088: 11110011 00110101 00101010 10110001 00010111 10001100 11000110 11001100
000090: 00100110 11011000 00100101 00100010 10000010 00100000 00111001 00011101
000098: 01001101 11000010 00000011 01111110 00100010 10110110 01100001 00111001
0000a0: 01000101 10001110 11100111 00101010 00111101 00001011 00010110 01101110
0000a8: 00000101 00010000 11100101 00100101 01001011 11100100 10001110 01001011
0000b0: 00100111 00110010 11011001 10001101 00000110 11110011 10111101 10000010
0000b8: 00010111 01100110 10000001 10001000 00001101 11010110 11001010 01010001
```

```
0000c0: 10100110 10000111 01001010 00110111 11011100 11001111 00111000 10011000
0000c8: 01010110 00100110 11001001 00001100 00111010 01000111 11100100 00110101
0000d0: 00101010 10100001 10010100 11011010 00001010 00110111 10011100 10000100
0000d8: 00000110 11010011 00001001 10001100 11010001 00011111 00110010 10001110
0000e0: 11101110 01110100 01001000 01101101 00100000 11011110 01110101 00110011
0000e8: 11010010 01101011 01110010 11101001 10000100 11001011 00010000 01100011
0000f0: 10000110 10011011 00110101 10010000 01100001 00010110 11100110 11100101
0000f8: 00011000 10001101 01000111 00100011 11010010 00000000 01010110 01101100
000100: 11001111 00100110 10000100 00001000 10000100 00010011 00011110 10101001
000108: 11010110 00101011 00000010 10010110 01111010 11001100 10100110 01100011
000110: 10101001 10110000 01000000 01100110 11011111 01010001 01110010 00111011
000118: 01101001 00000100 11010000 11011110 01101101 00111000 00011010 01111010
000120: 11100011 10010010 00001000 10010010 00101000 10001011 00011010 00010110
000128: 10100000 10110110 11100000 01010011 10110110 11011101 10001110 01101101
000130: 00101010 01011000 11101000 10111101 00001110 10011000 01000000 11110111
000138: 10101000 10000011 00101000 11100010 10010011 00001101 10001001 11001010
000140: 10001010 10011100 00111110 01100011 10100011 01011100 11001000 00101000
000148: 11101000 10110011 10000010 00110101 00111110 11001110 11101010 11111110
000150: 11000000 10100000 01001100 10101011 10000010 00110111 00110110 00000011
000158: 00100000 11010010 00110011 00001100 11001010 11001000 11011100 10010110
000160: 00001100 10001111 11100000 11000010 10001111 10001110 01100001 01110000
000168: 00010100
% ./lzwzip -o OriginalFile -d < Compressed.lzw
% cmp KRexcerpt.txt OriginalFile
%

% ./lzwzip -p < largefile.txt > largefile.txt.lzw
Achieved compression of 63.88%
% ls -l largefile.txt
-rw-r-----   1 ip       www      1668340 Nov 25 10:40 largefile.txt
% ls -l largefile.txt.lzw
-rw-r-----   1 ip       www       602677 Dec  7 19:45 largefile.txt.lzw
% ./lzwzip -d -i largefile.txt.lzw > newlargefile.txt
% diff largefile.txt newlargefile.txt
% ./lzwzip -v < largefile.txt > temp.lzw
UP (in: 359 bytes, out: 2307 bits)
UP (in: 1385 bytes, out: 7427 bits)
UP (in: 4301 bytes, out: 18691 bits)
UP (in: 10729 bytes, out: 43267 bits)
UP (in: 27148 bytes, out: 96515 bits)
UP (in: 69974 bytes, out: 211203 bits)
UP (in: 154501 bytes, out: 456963 bits)
%

% ./lzwzip -i largefile.txt -b 12 -p -o 12.lzw
Achieved compression of 48.92%
% ls -l 12.lzw
-rw-r-----   1 ip       www       852151 Dec  7 20:07 12.lzw
% ./bd 12.lzw | head
```

```
000000: 01100100 10010011 01110001 10011000 11011110 01110010 00110110 10011010
000008: 01001111 00000010 00000010 01101001 10010100 11100110 01110011 00110000
000010: 10011001 11100001 01100010 00000011 00001001 10111000 11001000 00100000
000018: 00100001 11000001 00001110 01000110 01010011 00011001 11010000 11010010
000020: 01101111 00110111 00011100 11000101 11000010 00001001 00001100 10001010
000028: 01000100 01010000 00111010 10011000 10001101 10000110 10010011 00011001
000030: 10000110 00110111 00011101 00010000 00010001 00100101 01100110 01010001
000038: 11010000 10000000 10001100 01100101 00110001 00011100 10001110 10100111
000040: 00100011 00001001 11001000 11110010 00100000 00011000 10001110 01100111
000048: 11100000 10100010 10110001 10010100 11100100 01110011 10001110 00011011
% ./bd 12.lzw | tail
0d0068: 10110010 00001100 11100011 01100011 11100111 11110100 01010110 00110110
0d0070: 01111001 01001000 01101101 11011001 01001101 01111001 10100110 11000101
0d0078: 00001001 10101000 10110100 01001001 01110010 00100010 01111011 00000011
0d0080: 10101100 11001000 00100011 10110100 10100011 11111110 01101000 00111000
0d0088: 01101100 10100011 01110011 10001001 01111111 11111111 10011010 01101010
0d0090: 01110110 01010000 00010000 11011010 10100000 01011010 00000110 10000000
0d0098: 01101101 00011100 00100101 11100010 00001110 01011001 10100111 01100101
0d00a0: 10001001 00110000 01000000 11010000 10100111 11000100 10001001 00001010
0d00a8: 00000001 01000000 01011010 00000101 10100000 01011010 00000101 10100000
0d00b0: 01011010 00000101 10100000 01011010 00000101 10100100 01010010
% ./lzwzip -d -i 12.lzw -o Origlarge.txt
% diff largefile.txt Origlarge.txt
%

% ./lzwzip -n -p -v -i evil.txt -o evil_no_clear.lzw
UP (in: 261 bytes, out: 2307 bits)
UP (in: 813 bytes, out: 7427 bits)
UP (in: 1956 bytes, out: 18691 bits)
UP (in: 4543 bytes, out: 43267 bits)
UP (in: 10656 bytes, out: 96515 bits)
UP (in: 24930 bytes, out: 211203 bits)
UP (in: 56809 bytes, out: 456963 bits)
Used additional 87.38%
% ./lzwzip -p -v -i evil.txt -o evil_with_clear.lzw
UP (in: 261 bytes, out: 2307 bits)
CL (in: 261 bytes, out: 2307 bits)
UP (in: 523 bytes, out: 4621 bits)
CL (in: 523 bytes, out: 4621 bits)
UP (in: 781 bytes, out: 6935 bits)
CL (in: 781 bytes, out: 6935 bits)
.............................................
UP (in: 130821 bytes, out: 1166249 bits)
CL (in: 130821 bytes, out: 1166249 bits)
UP (in: 131102 bytes, out: 1168563 bits)
CL (in: 131102 bytes, out: 1168563 bits)
UP (in: 163998 bytes, out: 1170877 bits)
UP (in: 426398 bytes, out: 1175997 bits)
Achieved compression of 85.90%
```

```
% ls -l evil*
-rw-------   1 ip       www      1048576 Dec 11 12:57 evil.txt
-rw-------   1 ip       www      1964853 Dec 12 20:19 evil_no_clear.lzw
-rw-------   1 ip       www       147809 Dec 12 20:19 evil_with_clear.lzw
%
```

Στην αξιολόγηση του αποτελέσματος που θα παραδώσετε, θα ληφθεί σοβαρά υπόψη αν το πρόγραμμά σας είναι αποδοτικό στη συμπίεση ακόμα και πολύ μεγάλων αρχείων. Σημειώστε ότι αυτό θα είναι δυνατόν μόνο αν οργανώσετε το λεξικό σας κατάλληλα ώστε οι αναζητήσεις ακολουθιών από bytes μέσα σ΄ αυτό να γίνονται γρήγορα. Μία ιδέα, όχι όμως η μοναδική, είναι **να οργανώσετε το λεξικό σαν ένα ταξινομημένο δυαδικό δέντρο**.

Θεωρώντας δεδομένο ότι θα χρησιμοποιήσετε τη βιβλιοθήκη εισόδου-εξόδου "`bit_io_<arch>.o`" σε επίπεδο bit που σας δίνεται, το πρόγραμμα που θα γράψετε θα πρέπει να είναι δομημένο σε ένα σύνολο από **τουλάχιστον τρία πηγαία αρχεία** C (με κατάληξη `.c`) και **τουλάχιστον δύο αρχεία επικεφαλίδας** (με κατάληξη `.h`), εκτός από το αρχείο επικεφαλίδας "`bit_io.h`" της βιβλιοθήκης. Για να παραδώσετε τη δουλειά σας, θα πρέπει να κάνετε τα εξής:

- Τοποθετήστε όλα τα αρχεία (πηγαία, αρχεία επικεφαλίδας και το αντικειμενικό αρχείο της βιβλιοθήκης που σας δίνεται έτοιμη, αν την χρησιμοποιήσατε) μέσα σ΄ ένα κατάλογο που θα δημιουργήσετε, έστω με όνομα `lzwzip`, σε κάποιο σύστημα Unix. Επίσης, τοποθετήστε στον κατάλογο αυτό και ένα αρχείο με όνομα `README`, στο οποίο να δίνετε οδηγίες για τη μεταγλώττιση των αρχείων και την κατασκευή του τελικού εκτελέσιμου. Προαιρετικά, μπορείτε να παραδώσετε και ένα αρχείο `Makefile` που να αναλαμβάνει όλη τη διαδικασία της κατασκευής του τελικού εκτελέσιμου μέσω της εντολής "`make`" (δώστε "`man make`" για περισσότερες λεπτομέρειες).

- Όντας στον κατάλογο που περιέχει τον κατάλογο `lzwzip`, δημιουργήστε ένα "επιπεδοποιημένο" `tar` αρχείο (έστω με όνομα `lzwzip.tar`) που περιέχει τον κατάλογο `lzwzip` και όλα του τα περιεχόμενα. Αυτό γίνεται με την εντολή "`tar cvf lzwzip.tar lzwzip`".[4]

- Συμπιέστε το αρχείο `lzwzip.tar`, ώστε να δημιουργηθεί το αρχείο `lzwzip.tar.gz`. Αυτό γίνεται με την εντολή "`gzip lzwzip.tar`".[5]

- Το αρχείο `lzwzip.tar.gz` είναι που θα πρέπει να υποβάλετε, με διαδικασία που θα ανακοινωθεί σύντομα.

Σημείωση: Η άσκηση αυτή μπορεί να παραδοθεί και από **ομάδες των δύο ατόμων**. Στην περίπτωση αυτή, θα παραδοθεί μόνο από το ένα μέλος της ομάδας, αλλά μέσα στο αρχείο `README` θα αναφέρονται σαφώς τα στοιχεία των δύο μελών. Ο στόχος της διαδικασίας αυτής είναι να ενισχυθεί η ιδέα της **ισότιμης** συνεργασίας σε μία ομάδα για την επίτευξη ενός στόχου. Αν τα μέλη της ομάδας έχουν υλοποιήσει διαφορετικά τμήματα της άσκησης, θα πρέπει στο αρχείο `README` να αναφέρεται ρητά τι έχει υλοποιήσει κάθε μέλος, έτσι ώστε στην προφορική εξέταση που θα ακολουθήσει, να μην υπάρχει η απαίτηση να έχει κάποιο μέλος της ομάδας πλήρη γνώση του πώς έχουν υλοποιηθεί τα τμήματα στα οποία εκείνο δεν έχει εμπλακεί.

[4] Αν θέλετε να ανακτήσετε την δενδρική δομή που έχει φυλαχθεί σ΄ ένα "επιπεδοποιημένο" `tar` αρχείο `file.tar`, αυτό μπορεί να γίνει με την εντολή "`tar xvf file.tar`".

[5] Αν θέλετε να αποσυμπιέσετε ένα αρχείο `file.gz` που έχει συμπιεσθεί με την εντολή `gzip`, αυτό μπορεί να γίνει με την εντολή "`gzip -d file.gz`".