ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2014-15)

Άσκηση 3

Το ζητούμενο της άσκησης αυτής είναι η "όμορφη" μορφοποίηση κειμένων σαν μία ακολουθία από γραμμές, όπου κάθε γραμμή δεν μπορεί να υπερβαίνει δεδομένο μήκος. Ένα κείμενο προς μορφοποίηση είναι μία ακολουθία από λέξεις που χωρίζονται μεταξύ τους με λευκό διάστημα (ένα ή περισσότερους κενούς χαρακτήρες ' ', χαρακτήρες στηλογνώμονα '\t' ή χαρακτήρες αλλαγής γραμμής '\n'). Κάθε λέξη είναι μία ακολουθία από χαρακτήρες οποιουδήποτε είδους (κεφαλαία γράμματα, μικρά γράμματα, αριθμητικά ψηφία, ειδικά σύμβολα, όπως σημεία στίξης, κλπ.), εξαιρουμένων αυτών που χαρακτηρίζονται ως λευκό διάστημα. Στο μορφοποιημένο κείμενο, οι λέξεις μίας γραμμής χωρίζονται μεταξύ τους με (ακριβώς ένα) κενό χαρακτήρα και μετά από κάθε γραμμή, υπάρχει (ακριβώς μία) αλλαγή γραμμής. Επίσης, στο μορφοποιημένο κείμενο, καμία γραμμή δεν μπορεί να εκτείνεται πέρα από ένα ορισμένο μήκος, έστω W, το οποίο είναι δεδομένο και θα δίνεται ως όρισμα στο πρόγραμμα που θα γράψετε. Το πότε μία μορφοποίηση είναι "όμορφη" ορίζεται στη συνέχεια.

Ορισμός κόστους μορφοποίησης

Είναι προφανές ότι για δεδομένο μη μορφοποιημένο κείμενο, αυτό μπορεί να μορφοποιηθεί με πολλούς τρόπους. Επειδή η σειρά των λέξεων του αρχικού κειμένου πρέπει να διατηρηθεί και επειδή δεν είναι δυνατόν μία λέξη να σπάσει σε περισσότερες από μία γραμμή και επειδή μεταξύ των λέξεων μίας γραμμής υπάρχει ακριβώς ένας κενός χαρακτήρας, είναι επόμενο μετά την τελευταία λέξη κάθε γραμμής να απομένει συνήθως κάποιος κενός χώρος μέχρι το μέγιστο μήκος γραμμής W. Αυτός ο κενός χώρος, ιδιαίτερα όταν είναι μεγάλος, δείχνει άσχημος οπτικά και κάνει το αποτέλεσμα να μην είναι αισθητικά αποδεκτό. Η "όμορφη" μορφοποίηση ενός κειμένου έγκειται στην ελαχιστοποίηση αυτού του κενού χώρου στο τέλος των γραμμών.

Για κάθε πιθανή μορφοποίηση ενός κειμένου, θα ορίσουμε και το *κόστος* της ως εξής. Έστω ότι το κείμενο προς μορφοποίηση περιέχει n λέξεις. Το μήκος της γραμμής στο μορφοποιημένο κείμενο που περιέχει ακριβώς τις λέξεις του αρχικού κειμένου από την i έως και την j μπορεί να υπολογιστεί ως εξής:

$$w_{ij} = \Big(\sum_{k=i}^{j}(l_k + 1)\Big) - 1$$

όπου l_k είναι ο αριθμός των χαρακτήρων της λέξης k. Το $+1$ που φαίνεται στον τύπο αντιπροσωπεύει τον ένα κενό χαρακτήρα μετά από κάθε λέξη και το -1 οφείλεται στο ότι μετά την τελευταία λέξη της γραμμής δεν υπάρχει κενός χαρακτήρας, αλλά αλλαγή γραμμής. Αν W είναι το μέγιστο επιτρεπόμενο μήκος γραμμής, πρέπει προφανώς, για κάθε γραμμή που περιέχει τις λέξεις από την i έως και την j, να ισχύει $w_{ij} \leq W$. Επίσης, για να είναι δυνατή η μορφοποίηση, καμία λέξη δεν θα πρέπει να έχει μήκος μεγαλύτερο από το μέγιστο μήκος γραμμής, δηλαδή $l_k \leq W$. Το κόστος της κάθε γραμμής ορίζεται ως το τετράγωνο του αριθμού των χαρακτήρων που παραμένουν κενοί μετά την τελευταία λέξη της γραμμής μέχρι το W, δηλαδή:

$$c_{ij} = (W - w_{ij})^2$$

όπου w_{ij} το μήκος της γραμμής, όπως ορίστηκε παραπάνω. Στην πράξη, ο κενός χώρος μετά το τέλος της τελευταίας γραμμής δε μας ενδιαφέρει να συνεισφέρει στο συνολικό κόστος μορφοποίησης. Μπορούμε να προσθέσουμε και αυτή την παράμετρο στον προηγούμενο μαθηματικό ορισμό, θεωρώντας $c_{ij} = 0$ όταν $j = n$, δηλαδή πρόκειται για την τελευταία γραμμή του κειμένου. Οπότε, αθροίζοντας τα κόστη όλων των γραμμών του μορφοποιημένου κειμένου, πλην της τελευταίας, έχουμε το συνολικό κόστος της μορφοποίησης. Προφανώς, αν όλες οι λέξεις του κειμένου μπορούσαν να μορφοποιηθούν

σε μία μόνο γραμμή μήκους το πολύ W, η οποία θα ήταν και η τελευταία, το συνολικό κόστος της μορφοποίησης θα ήταν 0, αλλά αυτό δεν συμβαίνει στη γενική περίπτωση.

Εναλλακτικά, το κόστος μορφοποίησης κειμένου μπορεί να οριστεί και αναδρομικά ως εξής. Αν f_j είναι το κόστος μορφοποίησης κειμένου j λέξεων, τότε ισχύει:

$$f_j = f_{i-1} + c_{ij}$$

αν στην τελευταία γραμμή τοποθετηθούν οι λέξεις από την i έως και την j. Θεωρούμε ότι $f_0 = 0$. Το κόστος μορφοποίησης των πρώτων $i-1$ λέξεων μπορεί να υπολογισθεί αναδρομικά. Προφανώς, το κόστος που ορίστηκε προηγουμένως ως το άθροισμα του κόστους των επί μέρους γραμμών ισούται με f_n, για ένα κείμενο n λέξεων.

Άπληστη μορφοποίηση

Μία μέθοδος που μπορούμε να εφαρμόσουμε για να περιορίσουμε τον κενό χώρο στο τέλος των γραμμών και να επιτύχουμε μικρό κόστος μορφοποίησης (αλλά όχι το ελάχιστο, κατ΄ ανάγκη) είναι να τοποθετούμε κατά σειρά τις λέξεις του αρχικού κειμένου σε μία γραμμή, μέχρι να μην μπορεί να τοποθετηθεί άλλη, επειδή θα παραβιασθεί ο περιορισμός του μέγιστους μήκους W ανά γραμμή, οπότε αυτή γίνεται η πρώτη λέξη της επόμενης γραμμής. Αυτή η μέθοδος ονομάζεται *άπληστη* (greedy) και χρησιμοποιείται από προγράμματα επεξεργασίας κειμένου, όπως το Microsoft Word. Για παράδειγμα, το κείμενο

```
This is a sample text to test various wrapping methods.
```

θα έπρεπε να μορφοποιηθεί με $W = 22$, από την άπληστη μέθοδο, ως εξής:

```
This is a sample text
to test various
wrapping methods.
```

Το κόστος αυτής της μορφοποίησης ισούται με 50.

Βέλτιστη μορφοποίηση

Παρατηρούμε ότι η μορφοποίηση που έκανε η άπληστη μέθοδος στο προηγούμενο παράδειγμα δεν ήταν με βέλτιστο κόστος. Αν το είχε μορφοποιήσει ως εξής (με $W = 22$):

```
This is a sample
text to test various
wrapping methods.
```

το κόστος θα ήταν 40. Μάλιστα, η συγκεκριμένη μορφοποίηση είναι η καλύτερη δυνατή που θα μπορούσε να γίνει, από πλευράς κόστους, για το δεδομένο W. Η συγγραφή της μεθόδου που επιτυγχάνει τη βέλτιστη μορφοποίηση μπορεί να γίνει με βάση την αναδρομική συνάρτηση που ορίζει το πρόβλημα. Πιο συγκεκριμένα:

$$f_j = \min_{1 \le i \le j} \Big(f_{i-1} + c_{ij} \Big)$$

όπου πρέπει να ισχύουν οι συνθήκες για τα c_{ij} όπως ορίστηκαν αρχικά. Επίσης, $f_0 = 0$.

Το πρόβλημα της αναδρομικής εκδοχής είναι ότι υπολογίζει ξανά ενδιάμεσα αποτελέσματα. Ο συνολικός χρόνος εκτέλεσης αυξάνει εκθετικά με το μέγεθος της εισόδου, δηλαδή τον αριθμό των λέξεων. Είναι δυνατόν να υλοποιηθεί όμως και μία προσέγγιση, αναδρομικά ή επαναληπτικά, η οποία να αποθηκεύει τα ενδιάμεσα αποτελέσματα αντί να τα υπολογίζει πάλι. Αυτά τα ενδιάμεσα αποτελέσματα

είναι ουσιαστικά τα κόστη c_{ij} να βρίσκονται οι λέξεις από την i έως και την j σε μία γραμμή, καθώς και ενδιάμεσα αποτελέσματα για την f. Η πολυπλοκότητα της μεθόδου με αποθήκευση ενδιάμεσων αποτελεσμάτων είναι πολυωνυμική. Η άπληστη μέθοδος έχει γραμμική πολυπλοκότητα, αλλά δεν εγγυάται την εύρεση βέλτιστης λύσης.

Η βελτιστοποιημένη μορφοποίηση χρησιμοποιείται από προγράμματα επεξεργασίας κειμένου, όπως το TEX του Donald Knuth. Μία εξελιγμένη μορφή του TEX, το LATEX, χρησιμοποιήθηκε για τη συγγραφή της εκφώνησης αυτής.

Πρόγραμμα

Γράψτε ένα πρόγραμμα C το οποίο να διαβάζει από την είσοδο ένα κείμενο και να το χωρίζει αρχικά σε λέξεις που θα αποθηκεύονται σε δυναμικά δεσμευμένη μνήμη. Αφού εκτελέσει έναν αλγόριθμο μορφοποίησης για να τοποθετήσει τις λέξεις σε γραμμές, θα τυπώνει το αποτέλεσμα στην έξοδο. Θα υλοποιήσετε τρεις διαφορετικούς αλγορίθμους μορφοποίησης. Ο πρώτος θα χρησιμοποιεί την άπληστη μέθοδο, ο δεύτερος θα χρησιμοποιεί τη βελτιστοποιημένη μέθοδο με αναδρομή, χωρίς αποθήκευση ενδιάμεσων αποτελεσμάτων, και ο τρίτος θα χρησιμοποιεί τη βελτιστοποιημένη μέθοδο (αναδρομικά ή επαναληπτικά), με αποθήκευση ενδιάμεσων αποτελεσμάτων. Ο μέγιστος αριθμός χαρακτήρων ανά γραμμή θα δίνεται σαν όρισμα στη γραμμή εντολής. Αν δεν δοθεί όρισμα, η προεπιλεγμένη τιμή να είναι 80. Επίσης, το πρόγραμμα να εκτυπώνει το κόστος της μορφοποίησης.

Για να απλοποιήσουμε την επεξεργασία της εισόδου, θεωρούμε το "`methods.`", όπως εμφανίζεται στο παράδειγμα, σαν μία ενιαία λέξη. Αν είχε κενό πριν την τελεία, θα το θεωρούσαμε δύο λέξεις και θα έπρεπε να βάλουμε κενό ανάμεσά τους. Εν ολίγοις, η κάθε λέξη διαχωρίζεται από τις άλλες με λευκό διάστημα. Τους χαρακτήρες λευκού διαστήματος (`' '`, `'\t'`, `'\n'`) μπορείτε να τους αναγνωρίζετε μέσω της συνάρτησης `isspace()` που ορίζεται στο αρχείο επικεφαλίδας `ctype.h`. Στο παράδειγμα που χρησιμοποιήσαμε προηγουμένως, η είσοδος θα μπορούσε να έχει την παρακάτω μορφή, αλλά θα πρέπει να παράγει το ίδιο αποτέλεσμα.

```
   This    is  a
     sample   text
to test  various wrapping
       methods.
```

Δεν υπάρχει κανένας περιορισμός για το μέγεθος της εισόδου (εκτός ίσως από τη μνήμη του υπολογιστή σας), ούτε για το πλήθος των χαρακτήρων σε κάθε γραμμή της εισόδου. Δηλαδή, το πρόγραμμά σας θα πρέπει να λειτουργεί σωστά, οσοδήποτε μεγάλη και να είναι η είσοδος, οσοδήποτε μεγάλες και να είναι οι γραμμές της εισόδου. Αν στην είσοδο δοθεί λέξη με πλήθος χαρακτήρων μεγαλύτερο από το W, το πρόγραμμα θα πρέπει να τερματίζει με κατάλληλο μήνυμα λάθους, στην πρώτη τέτοια λέξη που θα βρει, χωρίς να κάνει τη μορφοποίηση.

Οι τρεις διαφορετικοί αλγόριθμοι μορφοποίησης θα πρέπει να είναι εναλλακτικές υλοποιήσεις της ίδιας συνάρτησης, δηλαδή με ίδιο όνομα, ίδιο τύπο επιστροφής και ίδιες τυπικές παραμέτρους. Η κάθε συνάρτηση θα πρέπει να είναι σε διαφορετικό αρχείο πηγαίου κώδικα και αναλόγως με το ποια συνδέεται με το υπόλοιπο πρόγραμμα, θα πρέπει να εκτελείται και ο αντίστοιχος αλγόριθμος.

Αν και στην άπληστη μέθοδο θα ήταν δυνατόν να γίνεται η μορφοποίηση ενόσω διαβάζεται η είσοδος, ακόμα και σε αυτή την περίπτωση, θα πρέπει πρώτα να διαβάσετε την είσοδο και μετά να κάνετε τη μορφοποίηση, για λόγους ομοιομορφίας στις προδιαγραφές των τριών συναρτήσεων.

Παραδοτέο

Θα πρέπει να δομήσετε το πρόγραμμά σας σε ένα σύνολο από **τουλάχιστον δύο πηγαία αρχεία** C (με κατάληξη `.c`) και **τουλάχιστον ένα αρχείο επικεφαλίδας** (με κατάληξη `.h`). Εννοείται

ότι αν υλοποιήσετε και τις τρεις μεθόδους, τα πηγαία αρχεία σας θα πρέπει να είναι τουλάχιστον τέσσερα, αφού κάθε μέθοδος θα πρέπει να υλοποιείται από διαφορετική εκδοχή της ίδιας συνάρτησης.

Για να παραδώσετε το σύνολο των αρχείων που θα έχετε δημιουργήσει για την άσκηση αυτή, ακολουθήστε την εξής διαδικασία. Τοποθετήστε όλα τα αρχεία μέσα σ' ένα κατάλογο που θα δημιουργήσετε, έστω με όνομα `wrap`, στους σταθμούς εργασίας του Τμήματος. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r wrap.zip wrap
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `wrap.zip`, στο οποίο περιέχεται ο κατάλογος `wrap` μαζί με όλα τα περιεχόμενά του.[1] Το αρχείο αυτό είναι που θα πρέπει να υποβάλετε μέσω του συστήματος υποβολής στη διεύθυνση `http://hwadm.di.uoa.gr`.[2]

Ενδεικτικές εκτελέσεις

Έστω ότι τα εκτελέσιμα προγράμματα που θα κατασκευάσετε τελικά είναι τα "`wrap_gre`", "`wrap_rec`" και "`wrap_mem`", για την άπληστη, την αναδρομική χωρίς αποθήκευση ενδιάμεσων αποτελεσμάτων και αυτή με αποθήκευση ενδιάμεσων αποτελεσμάτων, αντίστοιχα. Κάποιες ενδεικτικές εκτελέσεις αυτών των προγραμμάτων, στο `linux28.di.uoa.gr` φαίνονται στη συνέχεια.[3]

```
$ ./wrap_gre 22
   This    is  a
     sample text
to test   various wrapping
       methods.
^D
This is a sample text
to test various
wrapping methods.
Words: 10
Width: 22
Cost:  50
$
$ cat small.txt
Like most imperative languages in the ALGOL tradition, C has facilities
for structured programming and allows lexical variable scope and recursion,
while a static type system prevents many unintended operations.
$
$ ./wrap_gre 33 < small.txt
Like most imperative languages in
the ALGOL tradition, C has
facilities for structured
programming and allows lexical
variable scope and recursion,
while a static type system
prevents many unintended
operations.
Words: 30
```

[1] Αρχεία `zip` μπορείτε να δημιουργήσετε και στα Windows, με διάφορα προγράμματα, όπως το WinZip.

[2] Μην υποβάλετε ασυμπίεστα αρχεία ή αρχεία που είναι συμπιεσμένα σε άλλη μορφή εκτός από `zip` (π.χ. `rar`, `7z`, `tar`, `gz`, κλπ.), γιατί δεν θα γίνονται δεκτά για αξιολόγηση.

[3] Για τα αρχεία εισόδου/εξόδου των ενδεικτικών εκτελέσεων: `http://www.di.uoa.gr/~ip/hwfiles/wrap/`

```
Width: 33
Cost:  268
$
$ cat KRExcerpt.txt
C is a general-purpose programming language. It has been closely
associated with the UNIX system where it was developed, since both
the system and most of the programs that run on it are written in
C. The language, however, is not tied to any one operating system
or machine; and although it has been called a "system programming
language" because it useful for writing compilers and operating
systems, it has been used equally well to write major programs in
many different domains.
$
$ ./wrap_gre 75 < KRExcerpt.txt
C is a general-purpose programming language. It has been closely associated
with the UNIX system where it was developed, since both the system and most
of the programs that run on it are written in C. The language, however, is
not tied to any one operating system or machine; and although it has been
called a "system programming language" because it useful for writing
compilers and operating systems, it has been used equally well to write
major programs in many different domains.
Words: 82
Width: 75
Cost:  70
$
$ ./wrap_gre 75 < KRExcerpt.txt > KRExcerpt_wrapped75_gre.txt
Words: 82
Width: 75
Cost:  70
$
$ ./wrap_gre < turing.txt > turing_wrapped80_gre.txt
Words: 373
Width: 80
Cost:  900
$
$ time ./wrap_gre 1000 < dict.txt > dict_wrapped1000_gre.txt
Words: 25143
Width: 1000
Cost:  4847
0.012u 0.000s 0:00.57 1.7%      0+0k 0+0io 0pf+0w
$
$
$
$ ./wrap_rec 22
   This    is  a
     sample text
to test   various wrapping
       methods.
^D
```

```
This is a sample
text to test various
wrapping methods.
Words: 10
Width: 22
Cost:  40
$
$ time ./wrap_rec 33 < small.txt
Like most imperative languages
in the ALGOL tradition, C
has facilities for structured
programming and allows lexical
variable scope and recursion,
while a static type system
prevents many unintended
operations.
Words: 30
Width: 33
Cost:  244
9.200u 0.000s 0:09.20 100.0%    0+0k 0+0io 0pf+0w
$
$ ./wrap_rec 75 < KRExcerpt.txt
^C                  //   ... bored waiting
$
$
$
$ ./wrap_mem 22
   This    is  a
     sample text
to test   various wrapping
       methods.
^D
This is a sample
text to test various
wrapping methods.
Words: 10
Width: 22
Cost:  40
$
$ time ./wrap_mem 33 < small.txt
Like most imperative languages
in the ALGOL tradition, C
has facilities for structured
programming and allows lexical
variable scope and recursion,
while a static type system
prevents many unintended
operations.
Words: 30
```

```
Width: 33
Cost:  244
0.000u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./wrap_mem 75 < KRExcerpt.txt > KRExcerpt_wrapped75_mem.txt
Words: 82
Width: 75
Cost:  52
$
$ ./wrap_mem < turing.txt > turing_wrapped80_mem.txt
Words: 373
Width: 80
Cost:  760
$
$ time ./wrap_mem 1000 < dict.txt > dict_wrapped1000_mem.txt
Words: 25143
Width: 1000
Cost:  4257
0.052u 0.008s 0:00.34 14.7%     0+0k 0+0io 0pf+0w
$
```