Οικονομικό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2016

Δομές Δεδομένων - Εργασία 2

Διδάσκων: Ε. Μαρκάκης

Ουρές προτεραιότητας - Προσομοίωση της λειτουργίας ενός print server

Σκοπός της εργασίας είναι η εξοικείωση με τις ουρές προτεραιότητας. Ως εφαρμογή θα δούμε τον τρόπο λειτουργίας ενός print server. Φανταστείτε ένα εργαστήριο όπως το CSLAB, όπου υπάρχει ένας κοινός δικτυακός εκτυπωτής και όλοι οι χρήστες που θέλουν να τυπώσουν κάποιο αρχείο στέλνουν ένα print request στον κοινό εκτυπωτή. Ο εκτυπωτής, για κάθε request, βλέπει τη χρονική στιγμή άφιξης καθώς και το μέγεθος του αρχείου που θέλει να τυπώσει ο χρήστης και διατηρεί μία ουρά με τα αρχεία που περιμένουν να εκτυπωθούν (δεν μπορεί να τυπώσει παράλληλα αρχεία, όταν εκτυπώνεται ένα από τα αρχεία, τα υπόλοιπα περιμένουν στην ουρά).

Μία λύση θα ήταν η ουρά αυτή να είναι ουρά FIFO έτσι ώστε όταν τελειώνει η εκτύπωση ενός αρχείου, ο εκτυπωτής να παίρνει το αρχείο που έχει μείνει τον περισσότερο χρόνο στην ουρά. Αυτή η υλοποίηση όμως αδικεί τα αρχεία μικρού μήκους καθώς αν ένα αρχείο 2 KB φτάσει έστω και 1 δευτερόλεπτο μετά από ένα αρχείο μεγέθους 10 MB, θα πρέπει να περιμένει πολύ ώρα. Πιο αναλυτικά, ας δούμε ένα μικρό παράδειγμα. Για να απλοποιήσουμε το πρόβλημα, έστω ότι το μέγεθος ενός αρχείου συμβολίζει και το πόσα δευτερόλεπτα χρειάζεται για να εκτυπωθεί το αρχείο και έστω ότι ο εκτυπωτής βλέπει τις αφίξεις:

```
t=0, άφιξη αρχείου μεγέθους 15,
t=5, άφιξη αρχείου μεγέθους 100,
t=10, άφιξη αρχείου μεγέθους 3.
```

Με χρήση ουράς FIFO, τη χρονική στιγμή t=5, ο εκτυπωτής θα βάλει στην ουρά του το δεύτερο αρχείο, αφού δεν έχει τελειώσει ακόμα η εκτύπωση του πρώτου. Τη χρονική στιγμή t=10, βλέπουμε ότι θα πρέπει να βάλει στην ουρά και το τρίτο αρχείο. Στη συνέχεια, τη στιγμή t=15, ο εκτυπωτής θα επιλέξει να τυπώσει το δεύτερο αρχείο, αφού έχει μείνει στην ουρά περισσότερη ώρα από το τρίτο. Ιδανικά όμως, θα θέλαμε το τρίτο αρχείο να πάρει μεγαλύτερη προτεραιότητα έτσι ώστε τη χρονική στιγμή t=15, που τελειώνει το πρώτο αρχείο, ο εκτυπωτής να διαλέξει το τρίτο αντί για το δεύτερο αρχείο.

Μέρος Α: Για να προσομοιώσουμε τη λειτουργία ενός print server που δίνει προτεραιότητα σε αρχεία με βάση τη διάρκεια εκτέλεσης τους, θα πρέπει πρώτα να δούμε τι ΑΤΔ θα χρειαστούμε. Συγκεκριμένα, θα πρέπει να υλοποιήσετε σίγουρα τους εξής 2 ΑΤΔ:

ΑΤΔ αρχείου. Χρειαζόμαστε έναν τύπο δεδομένων που αναπαριστά ένα αρχείο προς εκτύπωση. Ονομάστε την κλάση αυτή PrintJob. Αντικείμενα της τάξης PrintJob πρέπει να περιέχουν

- ένα μοναδικό id που ανατίθεται όταν δημιουργείται το κάθε αντικείμενο (χρήσιμο για debugging). Π.χ. δώστε id=1 στο πρώτο αρχείο που έρχεται, id=2 στο δεύτερο, κ.ο.κ.
- ένα πεδίο size με το μέγεθος του αρχείου. Το μέγεθος θα πρέπει να είναι ακέραιος και να κυμαίνεται στο διάστημα [1, MAXSIZE], όπου για τους σκοπούς της εργασίας θα θεωρήσουμε ότι MAXSIZE=128,
- ένα πεδίο waitingTime, το οποίο δηλώνει πόση ώρα βρίσκεται στην ουρά το αρχείο,
- ένα πεδίο arrivalTime που δηλώνει την στιγμή άφιξης,
- ένα πεδίο priority. Η προτεραιότητα θα είναι ακέραιος αριθμός και θα πρέπει να κυμαίνεται στο διάστημα [0, 127].

Μπορείτε να προσθέσετε και όποιο άλλο πεδίο κρίνετε απαραίτητο. Τέλος, η PrintJob θα πρέπει να υλοποιεί το interface Comparable<PrintJob> έτσι ώστε να μπορέσετε να την χρησιμοποιήσετε σε μία ουρά προτεραιότητας. Για την υλοποίηση του interface, ουσιαστικά πρέπει να υλοποιήσετε κατάλληλα τη μέθοδο compareTo, για να μπορείτε να συγκρίνετε αρχεία ως προς την προτεραιότητά τους.

ΑΤΔ ουράς προτεραιότητας. Θα χρειαστείτε και μια αποδοτική δομή δεδομένων για ουρά προτεραιότητας. Μπορείτε είτε να βασιστείτε στην ουρά του εργαστηρίου είτε να φτιάξετε τη δική σας υλοποίηση από την αρχή. Είναι βολικό να χρησιμοποιήσετε υλοποίηση της ουράς με generics, αλλά είναι επίσης αποδεκτό να φτιάξετε ουρά για αντικείμενα τύπου PrintJob. Σε κάθε περίπτωση, η ουρά σας θα πρέπει να ονομάζεται MaxPQ και να διαθέτει τις εξής λειτουργίες με την αντίστοιχη πολυπλοκότητα (έχουμε δει όλες τις μεθόδους εκτός από την peek, στο μάθημα και στο εργαστήριο, η peek επιστρέφει ό,τι και η getmax αλλά χωρίς να κάνει διαγραφή).

Μέθοδος	Λειτουργία	Πολυπλοκότητα
`size`	Μέτρηση αριθμού αντικειμένων	O(1)
`isEmpty`	Ελέγχει αν η ουρά είναι άδεια	O(1)
`insert`	Εισάγει νέο αντικείμενο στην ουρά	O(logn) (*δειτε σχόλιο παρακάτω*)
`getmax`	Αφαιρεί και επιστρέφει το αντικείμενο με τη μέγιστη προτεραιότητα	O(logn)
`peek`	Επιστρέφει χωρίς να διαγράψει το αντικείμενο με τη μέγιστη προτεραιότητα	O(1)

Σχόλιο: Η μέθοδος insert θα πρέπει να περιέχει και μια μέθοδο resize, για να κάνει επέκταση του πίνακα μόνο στην περίπτωση που η ουρά αποκτήσει πληρότητα άνω του 75%. Η απαίτηση για πολυπλοκότητα O(logn) είναι μόνο όταν δεν χρειάζεται να κληθεί και η resize.

Μέρος Β: Σκοπός του Μέρους Β είναι να χρησιμοποιήστε μία ουρά προτεραιότητας για να προσομοιώσετε τη λειτουργία ενός εκτυπωτή ο οποίος δίνει πάντα προτεραιότητα στα αρχεία με μικρότερο μέγεθος. Σε κάθε χρονική στιγμή που ο εκτυπωτής τελειώνει την εκτύπωση ενός αρχείου, θα πρέπει να κοιτάει τα αρχεία που περιμένουν στην ουρά προτεραιότητας και να διαλέγει αυτό με το μικρότερο μέγεθος.

Input και output

Input. Το πρόγραμμα που θα φτιάξετε θα πρέπει να διαβάζει από ένα αρχείο δεδομένα στη μορφή:

t_1 m_1
t_2 m_2
t_3 m_3
. . .

όπου t_i είναι η χρονική στιγμή άφιξης του αρχείου με μέγεθος m_i. Για τα δεδομένα θα πρέπει να ισχύει $0 \le t_1 \le t_2 \le \ldots$ (αλλιώς θα τυπώνετε μήνυμα λάθους). Παρατηρείστε ότι επιτρέπεται να έρθουν πολλά αρχεία την ίδια χρονική στιγμή. Επίσης, τα t_i, m_i θα πρέπει να είναι ακέραιοι αριθμοί και για τα μεγέθη των αρχείων ισχύει ότι $0 < m_i \le$ MAXSIZE. Όταν δεν τηρείται κάποια από αυτές τις προϋποθέσεις θα πρέπει να τυπώνετε το αντίστοιχο μήνυμα λάθους.
Ο print server θα πρέπει να δουλευει ως εξής: όταν έρχεται ενα νέο αρχείο και ο εκτυπωτής δεν είναι απασχολημένος, τότε ξεκινάει η εκτύπωση του αρχείου. Σε αντίθετη περίπτωση, το αρχείο μπαίνει στην ουρά προτεραιότητας. Όταν ο εκτυπωτής τελειώνει την εκτύπωση ενός αρχείου, τότε θα πρέπει να κοιτάζει την ουρά και να επιλέγει το αρχείο με την μεγαλύτερη προτεραιότητα. Αν έρθει ενα νέο αρχείο την ίδια στιγμή που ο εκτυπωτής τελειώνει κάποια εκτύπωση, θα πρέπει πρώτα να βάλετε στην ουρά το νέο αρχείο και μετά ο εκτυπωτής να διαλέξει το επόμενο.

Output. Το πρόγραμμά σας θα πρέπει να τυπώνει σε ένα αρχείο .txt τη σειρά με την οποία έγιναν οι εκτυπώσεις, το μέσο χρόνο αναμονής στην ουρά, αλλά και το μέγιστο χρόνο αναμονής (και από ποιο αρχείο προήλθε). Π.χ. για το παράδειγμα που αναφέραμε, στην 1η σελίδα, θα τυπώνει στο αρχείο:

```
t=15, completed file 1
t=18, completed file 3
t=118, completed file 2
Average waiting time = 6
Maximum waiting time = 13, achieved by file 2
```

Οδηγίες και συμβουλές για την υλοποίηση (υπάρχουν και άλλοι τρόποι για να κάνετε το κομμάτι της προσομοίωσης, μπορείτε να κάνετε ό,τι είναι πιο βολικό για εσάς):

- Το πρόγραμμα πρέπει να διαβάζει την είσοδο από αρχείο. Διαβάστε όλο το αρχείο εισόδου, και αποθηκεύστε όλα τα αντικείμενα PrintJob σε μία λίστα μονής σύνδεσης με αύξουσα σειρά ως προς τη χρονική στιγμή άφιξης. Μην τα αποθηκεύσετε σε πίνακα.
- Είναι βολικό να χρησιμοποιήσετε μία μεταβλητή T που προσομοιώνει το χρόνο. Η μεταβλητή αυτή θα σας βοηθήσει στο να ελέγχετε κάθε πότε πρέπει να προβείτε σε νέες ενέργειες. Π.χ. τη στιγμή $T = t_1$, αφαιρείτε από τη λίστα το πρώτο αντικείμενο (αν υπάρχει μόνο ένα που φτάνει όταν $T = t_1$), το οποίο

και ξεκινάτε να εκτελείτε, αφού δεν υπάρχει κάτι σε εκκρεμότητα εκείνη τη στιγμή. Η επόμενη στιγμή που χρειάζεται να κάνετε κάτι είναι ουσιαστικά τη στιγμή $T = t_1 + m_1$, οπότε και τελειώνει το πρώτο PrintJob. Τότε θα πρέπει να δείτε ποια αντικείμενα έχουν ήδη έρθει στο χρονικό διάστημα $[t_1, t_1 + m_1]$ (διαβάζοντας από τη λίστα που φτιάξατε), να τα βάλετε στην ουρά προτεραιότητας, και να δείτε ποιο πρέπει να επιλέξετε για εκτύπωση. Είναι πολύ σημαντικό όπως συνεχίζεται η εκτέλεση, να προσέχετε να ενημερώνετε πρώτα την ουρά προτεραιότητας, προτού αποφασίσετε ποιο θα είναι το επόμενο αρχείο που θα εκτυπωθεί.

- Δεν είναι αποδεκτό να έχετε ενα for loop όπου θα αυξάνετε την μεταβλητή T κατά 1 και να βλέπετε αν χρειάζεται να κάνετε κάτι εκείνη τη χρονική στιγμή. Δεν είναι αποδοτική μια τέτοια υλοποίηση. Αντιθέτως, θέλουμε να ενημερώνετε τη μεταβλητή του χρόνου μόνο όποτε είναι απαραίτητο (δηλαδή μόνο όταν συμβαίνει κάτι). Προσομοιώσεις τέτοιου τύπου ονομάζονται *event-driven simulations* (σε αντίθεση με τις *time-driven simulations,* που στη συγκεκριμένη εφαρμογή δεν έχουν καλή απόδοση).
- Δείτε την τελευταία σελίδα για την ονομασία του προγράμματος και άλλες σχετικές πληροφορίες.

Μέρος Γ: Με την υλοποίηση του Μέρους Β, το γεγονός ότι τα μικρά αρχεία δεν περιμένουν πολύ ώρα, μπορεί τελικά να επιφέρει μεγάλες καθυστερήσεις στα μεγάλα αρχεία. Έστω ότι επεκτείνουμε το παράδειγμα της 1ης σελίδας, ως εξής:

```
t=0, άφιξη αρχείου μεγέθους 15,
t=5, άφιξη αρχείου μεγέθους 100,
t=10, άφιξη αρχείου μεγέθους 3,
t=17, άφιξη αρχείου μεγέθους 25,
t=40, άφιξη αρχείου μεγέθους 30,
t=70, άφιξη αρχείου μεγέθους 48,
```

Βλέπουμε ότι το δεύτερο αρχείο θα αργήσει πάρα πολύ να εκτυπωθεί, αν δίνουμε συνέχεια προτεραιότητα στα μικρότερα αρχεία. Για να μην δημιοργούνται τέτοια φαινόμενα, μπορούμε να αλλάξουμε την προτεραιότητα του κάθε αρχείου έτσι ώστε σταδιακά να παίρνει υπόψιν το χρόνο παραμονής στην ουρά. Ένας τρόπος που χρησιμοποιείται στην πράξη είναι ο εξής: αρχικά οι προτεραιότητες είναι όπως και στο μέρος Β, αλλά κάθε 15 δευτερόλεπτα (ξεκινώντας από τη χρονική στιγμή που ήρθε το πρώτο αρχείο), η προτεραιότητα ανανεώνεται και γίνεται:

```
priority = min(127, priority + waitingTime)
```

Έτσι σταδιακά τα μεγάλα αρχεία θα εκτυπωθούν καθώς θα υπερισχύσουν έναντι μικρών αρχείων που έχουν εισαχθεί πιο πρόσφατα στην ουρά. Στο παράδειγμα παραπάνω με τα 6 αρχεία, το δεύτερο αρχείο θα εκτελεστεί πριν το τελευταίο, με βάση το νέο αυτό κανόνα.

Υλοποιήστε το νέο αλγόριθμο. Η έξοδος του προγράμματος θα πρέπει να είναι όπως και στο μέρος Β.

Μέρος Δ: Σύγκριση. Στο μέρος αυτό θα κάνετε μία μικρή πειραματική αξιολόγηση για να δείτε τη συμπεριφορά των αλγορίθμων στην πράξη. Χρησιμοποιήστε την Random του πακέτου java.util και δημιουργήστε τυχαία δεδομένα εισόδου για 5 διαφορετικές τιμές του N, όπου με N συμβολίζουμε το πλήθος των αρχείων προς

εκτύπωση. Διαλέξτε 5 τιμές του N στο διάστημα [100, 50.000]. Οι τιμές που θα διαλέξετε θα πρέπει να απέχουν μεταξύ τους τουλάχιστον κατά 5000. Π.χ. μπορείτε να διαλέξτε μια τιμή κάτω από 500, μια κοντά στο 5000, μία κοντά στο 50000 και άλλες 2 ενδιάμεσα. Για κάθε τιμή του N, δημιουργήστε 10 διαφορετικά input files για τους αλγορίθμους του μέρους Β και Γ, συνολικά δηλαδή θα δημιουργήσετε 50 αρχεία τυχαίων δοκιμαστικών δεδομένων.

Στη συνέχεια γράψτε ένα πρόγραμμα που θα εκτελεί και τους 2 αλγορίθμους για τα 50 αρχεία εισόδου, και θα τυπώνει τα αντίστοιχα αρχεία εξόδου. Για κάθε τιμή του N, βρείτε τη μέση τιμή του average waiting time και του maximum waiting time από τα 10 αρχεία που εκτελέσατε με το συγκεκριμένο N.

Παραδοτέα: Για την εργασία θα υποβάλετε τα εξής αρχεία:

1) Τα αρχεία MaxPQ.java και PrintJob.java για τους ΑΤΔ ουράς προτεραιότητας και αρχείου αντίστοιχα.
2) AlgorithmB.java, το πρόγραμμα που εκτελεί τον αλγόριθμο του μέρους Β. Δεν θα έχει main, αλλά θα περιέχει μία static void μέθοδο με όνομα runB, μέσα στην οποία θα καλούνται οι σχετικές μέθοδοι για την ανάγνωση του input και την εκτέλεση του αλγορίθμου.
3) AlgorithmC.java, το πρόγραμμα που εκτελεί τον αλγόριθμο του μέρους Γ. Δεν θα έχει main, αλλά θα περιέχει μία static void μέθοδο με όνομα runC, σε αντιστοιχία με το μέρος Β.
4) Comparisons.java, το πρόγραμμα του Μέρους Δ, που διαβάζει τα τυχαία δεδομένα εισόδου, εκτελεί και τους δύο αλγορίθμους και παράγει τα αρχεία εξόδου που χρειάζονται για να συγκρίνετε τους 2 αλγορίθμους. Το πρόγραμμα θα περιέχει μέθοδο main, η οποία θα καλεί τις runB, και runC που αναφέρονται παραπάνω.
5) Σε ξεχωριστό subdirectory βάλτε τα τυχαία δεδομένα εισόδου που θα παράγετε (50 αρχεία). Δεν απαιτείται ο κώδικας με τον οποίο θα παράγετε τα τυχαία δεδομένα.
6) Ό,τι άλλα αρχεία έχετε χρησιμοποιήσει και απαιτούνται για να τρέξει ο κώδικάς σας.
7) Τέλος, θα πρέπει να υποβάλετε μία αναφορά σε pdf αρχείο με όνομα project2-report.pdf, στην οποία θα αναφέρετε τα μέλη της ομάδας σας, θα περιγράψετε συνοπτικά την υλοποίησή σας και θα συγκρίνετε την αποδοτικότητα των 2 αλγορίθμων ως προς τα δεδομένα με τα οποία τους χρησιμοποιήσατε. Είναι βολικό να χρησιμοποιήσετε πίνακες για να δείξετε συγκεντρωτικά αποτελέσματα. Π.χ. μπορείτε σε ένα πίνακα να δείξετε για κάθε τιμή του N, τη μέση τιμή του average waiting time και τη μέση τιμή του maximum waiting time, από τα 10 αρχεία που έχετε με N αφίξεις. Να αναφέρετε ό,τι παρατηρήσεις έχετε να κάνετε σχετικά με τη συμπεριφορά των 2 αλγορίθμων για το average waiting time και το maximum waiting time. Όποιος θέλει μπορεί να χρησιμοποιήσει και γραφικές παραστάσεις.

Όλα τα παραπάνω αρχεία θα πρέπει να μπουν σε ένα αρχείο zip. Το όνομα που θα δώσετε στο αρχείο αυτό θα είναι ο αριθμός μητρώου σας πχ. 3030056_3030066.zip ή 3030056.zip (αν δεν είστε σε ομάδα). Στη συνέχεια, θα υποβάλλετε το zip αρχείο σας στην περιοχή του μαθήματος «Εργασίες» στο e-class.

<u>Η προθεσμία παράδοσης της εργασίας είναι Δευτέρα, 19 Δεκεμβρίου 2016 και ώρα 23:59.</u>