ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 2

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2017-2018

Διδάσκων: Ε. Μαρκάκης

Ουρές Προτεραιότητας και προβλήματα χρονοπρογραμματισμού

Σκοπός της εργασίας είναι η εξοικείωση με τις ουρές προτεραιότητας μέσω μίας εφαρμογής για λειτουργικά συστήματα και προβλήματα χρονοπρογραμματισμού (job scheduling). Έστω ότι έχουμε ένα σύστημα με ***M*** επεξεργαστές, στο οποίο θέλουμε να δρομολογούμε διεργασίες. Έστω π.χ. ότι έχουν καταφτάσει ***N*** διεργασίες και για κάθε i = 1, ..., N, η διεργασία i απαιτεί χρόνο επεξεργασίας (processing time) ίσο με p_i, ίδιο σε όλους τους επεξεργαστές. Θέλουμε να αναθέσουμε κάθε διεργασία σε κάποιον επεξεργαστή, με απώτερο στόχο να ολοκληρωθεί η εκτέλεση όλων των διεργασιών σε όσο το δυνατόν μικρότερο χρόνο. Κάθε διεργασία θα εκτελεστεί αποκλειστικά σε έναν επεξεργαστή και η εκτέλεσή της εφόσον ξεκινήσει δεν διακόπτεται.

Μέρος Α. Προτού προχωρήσουμε στους αλγορίθμους που θα υλοποιήσετε, ας ξεκινήσουμε με τους ΑΤΔ που θα χρειαστείτε.

ΑΤΔ επεξεργαστή: Θα πρέπει πρώτα να υλοποιήσετε έναν τύπο δεδομένων που αναπαριστά έναν επεξεργαστή. Ονομάστε την κλάση αυτή Processor. Ένα αντικείμενο της τάξης Processor πρέπει

- να έχει μοναδικο id που ανατίθεται όταν δημιουργείται (χρήσιμο για debugging),
- να περιέχει μια λίστα με όνομα `processed_jobs`, που αρχικοποιείται ως κενή, και κατά την εκτέλεση ενός αλγορίθμου θα έχει τις διεργασίες που έχει ήδη επεξεργαστεί ο συγκεκριμένος επεξεργαστής (η λίστα αυτή θα ενημερώνεται από τον αλγόριθμο του Μέρους Β),
- να διαθέτει τη μέθοδο `getActiveTime()`, η οποία επιστρέφει τον χρόνο που έχει ξοδέψει ο επεξεργαστής με τις διεργασίες που έχει επεξεργαστεί (δηλαδή το άθροισμα των processing times των διεργασιών από τη λίστα `processed_jobs` ή 0 αν δεν έχει επεξεργαστεί τίποτα).

Στην κλάση Processor μπορείτε να προσθέσετε και ό,τι άλλο πεδίο θεωρείτε εσείς χρήσιμο. Τέλος, η Processor θα πρέπει να υλοποιεί το interface Comparable<Processor> έτσι ώστε να μπορέσετε να την χρησιμοποιήσετε σε μία ουρά προτεραιότητας. Για την υλοποίηση του interface (και συνεπώς της συνάρτησης compareTo) συγκρίνετε το active time των επεξεργαστών: αν ο επεξεργαστής A έχει κάνει περισσότερη δουλειά από τον επεξεργαστή B τότε η πράξη

A.compareTo(B)

πρέπει να επιστρέφει 1. Στην αντίθετη περίπτωση επιστρέφει -1, ενώ σε ισοβαθμία επιστρέφει 0.
Για τη λίστα processed_jobs, δεν επιτρέπεται να χρησιμοποιήσετε έτοιμες υλοποιήσεις δομών τύπου λίστας από την βιβλιοθήκη της Java (π.χ. Vector, ArrayList κλπ). Όπως και στην Εργασία 1, μπορείτε να χρησιμοποιήσετε είτε τη λίστα του εργαστηρίου, είτε να φτιάξετε τη δική σας.

ΑΤΔ ουράς προτεραιότητας. Για την εργασία, θα χρειαστείτε μια αποδοτική δομή δεδομένων για ουρά προτεραιότητας. Μπορείτε είτε να χρησιμοποιήσετε την ουρά του εργαστηρίου είτε να φτιάξετε τη δική σας ουρά, βασισμένοι στις διαφάνειες των διαλέξεων. Ονομάστε το αρχείο αυτό MaxPQ.java.

Μέρος Β. Ο πρώτος αλγόριθμος που καλείστε να υλοποιήσετε για τη δρομολόγηση είναι ο εξής:

Αλγόριθμος 1 (Greedy)

Επεξεργάσου τις διεργασίες με τη σειρά που εμφανίζονται. Για i=1,...,N, βάλε τη διεργασία i στον επεξεργαστή ο οποίος μέχρι εκείνη τη χρονική στιγμή έχει επιτελέσει το λιγότερο έργο (δηλαδή έχει ξοδέψει το λιγότερο χρόνο σε επεξεργασία διεργασιών). Αν υπάρχουν ισοβαθμίες, η διεργασία μπορεί να πάει σε οποιονδήποτε από τους επεξεργαστές που ισοβαθμούν.

Παράδειγμα. Έστω ότι έχουμε 3 επεξεργαστές P1, P2, P3, και 5 διεργασίες με χρόνους επεξεργασίας κατά σειρά άφιξης 25, 30, 20, 35, 50. Έστω ότι σε ισοβαθμίες επιλέγουμε πάντα τον επεξεργαστή με το χαμηλότερο δείκτη. Τότε ο αλγόριθμος θα έβαζε τις πρώτες 3 διεργασίες σε διαφορετικούς επεξεργαστές (την $1^η$ στον P1, τη $2^η$ στον P2, και την $3^η$ στον P3), και στη συνέχεια θα έβαζε την $4^η$ διεργασία στον P3, αφού σε εκείνη τη χρονική στιγμή, ο P3 έχει δουλέψει μόνο για 20 μονάδες. Τέλος, η $5^η$ διεργασία θα πήγαινε στον P1 και η τελική ανάθεση θα ήταν: P1: {25, 50}, P2: {30}, P3: {20, 35}. Η ποσότητα που μας ενδιαφέρει σε αυτή την ανάθεση είναι η χρονική στιγμή στην οποία έχει ολοκληρωθεί η εκτέλεση όλων των διεργασιών. Η ποσότητα αυτή ονομάζεται makespan και καθορίζεται από τον πιο «φορτωμένο» επεξεργαστή. Στο συγκεκριμένο παράδειγμά, έχουμε ότι

```
makespan = 75
```

Input και output

Input. Το πρόγραμμα πελάτης θα διαβάζει από ένα txt αρχείο εισόδου πρώτα τον αριθμό των επεξεργαστών, μετά τον αριθμό των διεργασιών και μετά τους χρόνους επεξεργασίας των διεργασιών. Για το παράδειγμά μας, το αρχείο εισόδου θα έχει την παρακάτω μορφή

```
3
5
25
30
20
35
50
```

Αν το αρχείο δεν είναι σε αυτήν την μορφή θα πρέπει να τυπώνετε μήνυμα λάθους και να τερματίζει το πρόγραμμα. Επίσης θεωρούμε ότι οι χρόνοι επεξεργασίας είναι ακέραιοι. Αν το αρχείο περιέχει και πραγματικούς αριθμούς να τυπώνετε αντίστοιχο μήνυμα λάθους.

Output Το πρόγραμμα θα πρέπει να υπολογίζει και να τυπώνει τη χρονική στιγμή στην οποία θα έχουν ολοκληρωθεί όλες οι διεργασίες (δηλαδή το makespan). Επίσης αν ο αριθμός των διεργασιών είναι μικρότερος από 100 να τυπώνετε και τους επεξεργαστές σε σειρά αύξουσα σε σχέση με το χρόνο που ήταν ενεργός ο καθένας. Για κάθε επεξεργαστή, τυπώστε το id του, το συνολικό χρόνο που ξόδεψε για επεξεργασία και στην συνέχεια το processing time κάθε διεργασίας που επεξεργάστηκε. Για το παράδειγμά μας το σχετικό output είναι:

```
id 2, load=30: 30
id 3, load=55: 20 35
id 1, load=75: 25 50
Makespan = 75
```

Το output δείχνει π.χ. ότι ο επεξεργαστής με id 1 έχει επεξεργαστεί 2 διεργασίες με χρόνους 25 και 50 αντίστοιχα.

Μέρος Γ. Στο παράδειγμα της $2^{ης}$ σελίδας, φαίνεται ότι θα μπορούσαμε να βελτιώσουμε το makespan αν βάζαμε τη διεργασία με χρόνο 50 σε έναν επεξεργαστή μόνη της. Π.χ., η ανάθεση P1: {25, 50}, P2: {30}, P3: {20, 35} θα έδινε makespan = 55. Αυτή η παρατήρηση οδηγεί στην εξής τροποποίηση του Αλγορίθμου 1:

Αλγόριθμος 2 (Greedy-decreasing)

Διάταξε τις διεργασίες με σειρά από τη μεγαλύτερη προς τη μικρότερη και μετά εφάρμοσε τον Αλγόριθμο 1.

Πρέπει λοιπόν να επεξεργαστείτε τη σειρά των διεργασιών έτσι ώστε να είναι σε φθίνουσα σειρά. Για να γίνει αυτό θα υλοποιήσετε έναν αλγόριθμο ταξινόμησης με όνομα Sort.java. Αφού ταξινομήσετε τις διεργασίες από τη μεγαλύτερη προς τη μικρότερη, θα μπορείτε μετά να καλείτε τον αλγόριθμο 1 δίνοντας την είσοδο ταξινομημένη.

ΠΡΟΣΟΧΗ: Απαγορεύεται να χρησιμοποιήσετε τις έτοιμες συναρτήσεις ταξινόμησης της βιβλιοθήκης της Java (ή οποιασδήποτε άλλης βιβλιοθήκης). Στα πλαίσια της εργασίας θα πρέπει να υλοποιήσετε μόνοι σας μία από τις μεθόδους Mergesort, Quicksort ή Heapsort. Μπορείτε να χρησιμοποιήσετε όποια μέθοδο θέλετε από αυτές τις τρεις, και μπορείτε να κάνετε ταξινόμηση είτε σε πίνακα είτε σε λίστα. Αν χρησιμοποιήσετε την Heapsort, μπορείτε να στηριχθείτε στην MaxPQ ξανά.

Μέρος Δ. Στο μέρος αυτό θα κάνετε μία μικρή πειραματική αξιολόγηση για να διαπιστώσετε ποιος αλγόριθμος είναι καλύτερος στην πράξη. Χρησιμοποιήστε την τάξη Random του πακέτου java.util και δημιουργήστε τυχαία δεδομένα εισόδου. Χρησιμοποιήστε τουλάχιστον 3 διαφορετικές τιμές για τον αριθμό των διεργασιών. Ενδεικτικά, μπορείτε να χρησιμοποιήσετε N = 100, 500, 1.000. Για κάθε τιμή του N, δημιουργήστε 10 διαφορετικά txt αρχεία εισόδου για τους αλγορίθμους 1 και 2. Συνολικά δηλαδή θα δημιουργήσετε τουλάχιστον 30 αρχεία τυχαίων δοκιμαστικών δεδομένων. Για κάθε N, χρησιμοποιήστε $M = \lfloor \sqrt{N} \rfloor$ επεξεργαστές.

Στη συνέχεια γράψτε ένα πρόγραμμα που θα εκτελεί και θα συγκρίνει τους δύο αλγόριθμους. Για κάθε αρχείο εισόδου καταγράψτε το makespan σε κάθε αλγόριθμο. Στη συνέχεια, υπολογίστε για κάθε τιμή του N, ποιο ήταν *κατά μέσο όρο* το makespan των δύο αλγορίθμων για τα 10 δοκιμαστικά αρχεία που αφορούν την συγκεκριμένη τιμή του N. Σχολιάστε τα αποτελέσματα που βλέπετε από την αξιολόγηση αυτή.

Προαιρετικά: Μπορείτε να χρησιμοποιήσετε περισσότερες τιμές για το N, και περισσότερα από 10 αρχεία για καθε τιμή του N, και να περάσετε τα αποτελέσματα σε ένα λογιστικό φύλλο (Microsoft Excel, OpenOffice Calc κλπ). Χρησιμοποιήστε τις δυνατότητες του λογιστικού φύλλου για να σχεδιάσετε ένα διάγραμμα όπου απεικονίζεται η σχέση του makespan ως συνάρτηση του πλήθους των διεργασιών. Στο ίδιο διάγραμμα μπορούν να απεικονίζονται δύο καμπύλες: μία για κάθε αλγόριθμο.

Hints και οδηγίες υλοποίησης για τα Μέρη Β, Γ και Δ:

- Το πρόγραμμά του Μέρους Β **πρέπει** να λέγεται Greedy.java. Θα περιέχει μέθοδο main μέσα στην οποία θα καλούνται οι σχετικές μέθοδοι για την ανάγνωση του input και την εκτέλεση του Αλγορίθμου 1.
- Η υλοποίησή του Αλγορίθμου 1 **πρέπει** να κάνει χρήση της ουράς προτεραιότητας MaxPQ. Χωρίς την ουρά, για να βρίσκετε σε κάθε βήμα σε ποιον επεξεργαστή θα αναθέσετε την επόμενη εργασία, θα έπρεπε είτε να σαρώνετε όλους τους επεξεργαστές και να βρείτε ποιος έχει κάνει τη λιγότερη δουλειά είτε να τους κρατάτε ταξινομημένους. Με τη χρήση της ουράς, η πολυπλοκότητα για την εκτέλεση κάθε βήματος βελτιώνεται.
- **Προσοχή στη χρήση της MaxPQ:** η MaxPQ καθώς και οι ουρές που είδαμε στο μάθημα κάνουν εξαγωγή στοιχείου με το μέγιστο κλειδί (μέγιστη προτεραιότητα). Στον Αλγόριθμο 1 όμως θέλουμε να επιλέγουμε κάθε φορά τον επεξεργαστή με τον ελάχιστο active time μέχρι εκείνη τη χρονική στιγμή. Σκεφτείτε πώς μπορείτε να χρησιμοποιήσετε τη MaxPQ έτσι ώστε να επιλέγετε το σωστό επεξεργαστή σε κάθε βήμα.
- Για την καλύτερη ανάπτυξη του κώδικά σας, προσπαθήστε να χωρίσετε τα προγράμματα σας σε modules ανάλογα με την λειτουργία τους. Για παράδειγμα,
  - γράψτε ξεχωριστή μέθοδο που διαβάζει το αρχείο εισόδου και το αποθηκεύει σε κάποιο πίνακα ή λιστα (αν χρησιμοποιήσετε πίνακες αντί για λίστες, μπορείτε πρώτα να μετρήσετε τον αριθμό των γραμμών του αρχείου εισόδου, έτσι ώστε να ξέρετε τι μέγεθος απαιτείται για να αποθηκεύσετε τα δεδομένα).
  - Υλοποιήστε τον αλγόριθμο 1 ώστε να παίρνει είσοδο τον πίνακα ή τη λίστα που έχετε διαβάσει έτσι ώστε να μην ασχοληθείτε περαιτέρω με το txt αρχείο εισόδου.
  - Με αυτό τον τρόπο, η υλοποίηση του Αλγορίθμου 2 απαιτεί μόνο την ταξινόμηση του πίνακα/λίστας και μπορείτε να επαναχρησιμοποιήσετε τα προηγούμενα κομμάτια του κώδικά σας από το Μέρος Β.
- Το πρόγραμμα ταξινόμησης του Μέρους Γ **πρέπει** να λέγεται Sort.java (δεν απαιτείται να υπάρχει μέθοδος main).
- Το πρόγραμμα του Μέρους Δ **πρέπει** να λέγεται Comparisons.java. Η μέθοδος main του προγράμματος αυτού θα διαβάζει την είσοδο, θα τρέχει τον Αλγόριθμο 1, και στη συνέχεια για να εκτελέσει τον Αλγόριθμο 2, θα χρησιμοποιεί την ταξινόμηση του Μέρους Γ, και θα ξανατρέχει τον Αλγόριθμο 1 με ταξινομημένη είσοδο. Το πρόγραμμα σας είτε θα τρέχει σε ένα for loop τους 2 αλγορίθμους για όλα τα δοκιμαστικά αρχεία εισόδου που θα φτιάξετε, είτε θα κάνει 1 εκτέλεση των αλγορίθμων, για 1 αρχείο εισόδου. Αν επιλέξετε το δεύτερο, θα πρέπει να το τρέξετε τόσες φορές όσες και τα αρχεία εισόδου που θα έχετε φτιάξει και να σημειώνετε το makespan που βρήκε κάθε φορά.

Οδηγίες Παράδοσης

Η εργασία σας θα πρέπει να μην έχει συντακτικά λάθη και να μπορεί να μεταγλωττίζεται. Εργασίες που δεν μεταγλωττίζονται χάνουν το 50% της συνολικής αξίας.

Η εργασία θα αποτελείται από:

1. Τον πηγαίο κώδικα (source code). Τοποθετήστε σε ένα φάκελο με όνομα **src** τα αρχεία java που έχετε φτιάξει. Ενδεικτικά θα πρέπει να περιέχει (χρησιμοποιήστε τα όνοματα των κλάσεων όπως ακριβώς δίνονται στην εκφώνηση):
    a. Τα αρχεία Greedy.java, Sort.java, Comparisons.java.
    b. Την υλοποίηση της ουράς προτεραιότητας MaxPQ και τον ΑΤΔ Processor.
    c. Επιπλέον, φροντίστε να συμπεριλάβετε όποια άλλα αρχεία πηγαίου κώδικα φτιάξατε και απαιτούνται για να μεταγλωττίζεται η εργασία σας. Φροντίστε επίσης να προσθέσετε επεξηγηματικά σχόλια όπου κρίνετε απαραίτητο στον κώδικά σας.
2. Τα δοκιμαστικά δεδομένα που φτιάξατε. Τοποθετήστε τα σε ένα subdirectory με όνομα **data.** Δεν απαιτείται να παραδώσετε τον κώδικα με τον οποίο τα φτιάξατε.
3. Μία σύντομη αναφορά σε pdf αρχείο (όχι Word ή txt!) με όνομα project2-report.pdf, στην οποία θα αναφερθείτε στα εξής:
    a. Εξηγήστε πώς χρησιμοποιήσατε την ουρά προτεραιότητας για την υλοποίηση του Αλγορίθμου 1, καθώς και τα υπόλοιπα βήματα του προγράμματος σας, π.χ. πώς διαβάζετε την είσοδο, πού την αποθηκεύετε, κτλ (άνω όριο 2 σελίδες).
    b. Περιγράψτε ποιον αλγόριθμο ταξινόμησης επιλέξατε να υλοποιήσετε, αν το κάνατε με πίνακα ή λίστα, και γιατί κάνατε αυτές τις επιλογές (άνω όριο 1 σελίδα).
    c. Για το Μέρος Δ, εξηγήστε αναλυτικά τα συμπεράσματα που προέκυψαν από την σύντομη αυτή πειραματική αξιολόγηση των 2 αλγορίθμων. Μπορείτε π.χ. να προσθέσετε κάποιο πινακάκι όπου να φαίνονται οι μέσες τιμές του makespan για καθε τιμή του N ή να βάλετε κάποιο διάγραμμα από λογιστικό φύλλο (άνω όριο 3 σελίδες).

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. Δεν χρειάζεται υποβολή και από τους 2 φοιτητές μιας ομάδας.

Η προθεσμία παράδοσης της εργασίας είναι Παρασκευή, 22 Δεκεμβρίου 2017 και ώρα 23:59.