ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 2

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2021-2022

Διδάσκων: Ε. Μαρκάκης

Προθεσμία παράδοσης: Πέμπτη, 23/12/2021, 23:59

Ταξινόμηση και Ουρές Προτεραιότητας

Σκοπός της 2ης εργασίας είναι η εξοικείωση με τους αλγορίθμους ταξινόμησης και με τη χρήση ουράς προτεραιότητας, μία από τις πιο βασικές δομές δεδομένων στην σχεδίαση αλγορίθμων. Η εργασία αφορά ερωτήματα που, μεταξύ άλλων, συναντώνται σε εφαρμογές 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.

Μέρος Α [15 μονάδες]: 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 Processor πρέπει

- να έχει μοναδικό, θετικό ακέραιο id που ανατίθεται όταν δημιουργείται (χρήσιμο για debugging),
- να περιέχει μια λίστα με όνομα `processedTasks`, που αρχικοποιείται ως κενή, και κατά την εκτέλεση ενός αλγορίθμου θα περιέχει τις διεργασίες που έχει ήδη επεξεργαστεί ο συγκεκριμένος επεξεργαστής (η λίστα αυτή θα ενημερώνεται από τον αλγόριθμο του Μέρους Β, και κάθε κόμβος της λίστας θα αντιστοιχεί σε μία διεργασία),
- 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 `processedTasks` 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, χρειάζεται απλά να υλοποιήσετε την συνάρτηση compareTo, στην οποία θα συγκρίνετε το active time των επεξεργαστών, δηλαδή, με μια κλήση της μορφής *A.compareTo(B),* θα πρέπει να συμπεραίνετε αν ο επεξεργαστής Α έχει κάνει περισσότερη δουλειά από τον επεξεργαστή Β. Σε περίπτωση ισοβαθμίας, δώστε προβάδισμα στον επεξεργαστή με το μικρότερο id (δείτε τον αλγόριθμο στο Μέρος Β).

ΑΤΔ διεργασίας. Θα έχετε κι έναν τύπο δεδομένων που αναπαριστά μια διεργασία. Ονομάστε την κλάση αυτή Task. Ένα αντικείμενο της Task πρέπει

- να έχει μοναδικό, θετικό ακέραιο id (χρήσιμο για debugging),
- να περιέχει ένα πεδίο `time` που να αντιστοιχεί στον χρόνο επεξεργασίας της διεργασίας.

ΑΤΔ ουράς προτεραιότητας. Θα χρειαστείτε μια αποδοτική δομή δεδομένων για ουρά προτεραιότητας, όπου θα αποθηκεύετε αντικείμενα τύπου Processor. Η υλοποίηση της ουράς θα γίνει με χρήση μεγιστοστρεφούς σωρού, όπως έχουμε δει στο μάθημα και στο εργαστήριο. Μπορείτε να βασιστείτε στην ουρά προτεραιότητας του Εργαστηρίου 6 ή σε ό,τι είδαμε στην Ενότητα 11 των διαφανειών, με κατάλληλες τροποποιήσεις, ή μπορείτε να φτιάξετε εξ' ολοκλήρου την δική σας ουρά. Η ουρά σας θα πρέπει να υλοποιεί το παρακάτω interface (μπορείτε να κάνετε την ουρά με generics, όπου τότε θα τροποποιήσετε κατάλληλα το interface).

```
public interface PQInterface {
public boolean isEmpty(); //check if the queue is empty
public int size(); //return the number of active elements in the queue
public void insert(Processor x); // insert the object x to the queue
public Processor max(); /*return without removing the object with
                         the highest priority */
public Processor getmax();/* remove and return the object
                              with the highest priority*/
}
```

Η πολυπλοκότητα που πρέπει να έχει η κάθε μέθοδος, σε μια ουρά με n αντικείμενα δίνεται στον παρακάτω πίνακα.

Μέθοδος	Πολυπλοκότητα
`size, isEmpty, max`	O(1)
`insert`	O(logn) (*δειτε σχόλια*)
`getmax`	O(logn)

Σχόλια:

- Η υλοποίηση της ουράς προτεραιότητας θα πρέπει να είναι στο αρχείο MaxPQ.java.
- Η insert θα πρέπει να καλεί και μια μέθοδο resize που θα κάνει διπλασιασμό του πίνακα όταν η ουρά έχει γεμίσει κατά το 75%. Συνεπώς, η απαίτηση για O(logn) στην insert είναι μόνο όταν δεν χρειάζεται να καλέσει τη resize για την επέκταση του πίνακα.

Μέρος Β [30 μονάδες]: Ο πρώτος αλγόριθμος που καλείστε να υλοποιήσετε για τη δρομολόγηση είναι ο εξής:

Αλγόριθμος 1 (Greedy)

Επεξεργάσου τις διεργασίες με τη σειρά που εμφανίζονται στο αρχείο εισόδου. Για i=1,...,N, βάλε τη διεργασία i στον επεξεργαστή ο οποίος εκείνη τη χρονική στιγμή έχει επιτελέσει το λιγότερο έργο (δηλαδή έχει ξοδέψει το λιγότερο χρόνο σε επεξεργασία διεργασιών). Αν υπάρχουν ισοβαθμίες, η διεργασία θα πάει στον επεξεργαστή με το μικρότερο id.

Παράδειγμα. Έστω ότι έχουμε 3 επεξεργαστές P1, P2, P3 (με ids αντίστοιχα 1, 2, 3) και 5 διεργασίες με χρόνους επεξεργασίας κατά σειρά άφιξης 25, 30, 20, 35, 50. 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 την χρονική στιγμή που αποφασίζουμε για την 4η, ο P3 έχει δουλέψει μόνο για 20 λεπτά. 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 να τυπώσουμε σε αυτή την τελική 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 σε κάθε επόμενη γραμμή το id μιας διεργασίας και τον χρόνο επεξεργασίας της. Όλα τα δεδομένα θα είναι θετικοί ακέραιοι. Για το παράδειγμά μας, το αρχείο εισόδου θα μπορούσε να έχει την παρακάτω μορφή (τα id παρακάτω είναι απλά τυχαία νούμερα)

```
3
5
12 25
1 30
32 20
128 35
4 50
```

Αν ο αριθμός των διεργασιών στην $2^η$ γραμμή δεν αντιστοιχεί στο πόσες γραμμές υπάρχουν μετά, θα πρέπει να τυπώνετε μήνυμα λάθους και να τερματίζει το πρόγραμμα. Μην ασχοληθείτε με το τι να κάνετε αν το αρχείο δεν είναι στο παραπάνω format. Επίσης θεωρήστε ότι τα id των διεργασιών θα είναι μοναδικά (οι χρόνοι επεξεργασίας δεν θα είναι απαραίτητα μοναδικοί).

Output Το πρόγραμμα θα πρέπει να υπολογίζει και να τυπώνει ποια είναι η χρονική στιγμή στην οποία θα έχουν ολοκληρωθεί όλες οι διεργασίες (δηλαδή το makespan), με βάση τον αλγόριθμο Greedy. Επίσης αν ο αριθμός των διεργασιών είναι μικρότερος από 50 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. Ενδεικτικά, για το παράδειγμά μας μπορείτε να τυπώσετε το output στην παρακάτω μορφή:

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

Οδηγίες:

- Το πρόγραμμά του Μέρους Β **πρέπει** να λέγεται Greedy.java. Θα περιέχει μέθοδο main μέσα στην οποία μπορείτε να καλείτε μεθόδους 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 αισθητά.
- **Προσοχή στη χρήση της MaxPQ:** η MaxPQ θα βασίζεται σε μεγιστοστρεφή σωρό, δηλαδή η getmax φέρνει πάντα το στοιχείο με την μέγιστη προτεραιότητα. Στον Αλγόριθμο 1 όμως θέλουμε να επιλέγουμε κάθε φορά τον επεξεργαστή με το 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. Αν ακολουθήσετε το υπόδειγμα του Εργαστηρίου 6, πρέπει να σκεφτείτε πως θα ορίσετε την προτεραιότητα ενός επεξεργαστή και να κατασκευάσετε τον κατάλληλο Comparator (με χρήση της compareTo της κλάσης Processor).
- Για το διάβασμα του αρχείου εισόδου, μπορείτε να χρησιμοποιήσετε έτοιμες μεθόδους της Java για άνοιγμα αρχείων όπως και στην Εργασία 1.
- **Δεν επιτρέπεται σε κανένα μέρος της εργασίας να χρησιμοποιήσετε έτοιμες υλοποιήσεις δομών τύπου λίστας, στοίβας, ουράς, από την βιβλιοθήκη της Java** (π.χ. Vector, ArrayList, κτλ). Μπορείτε φυσικά να χρησιμοποιήσετε κώδικα από τα εργαστήρια.

Μέρος Γ [15 μονάδες]. Στο προηγούμενο παράδειγμα, φαίνεται ότι θα μπορούσαμε να βελτιώσουμε το makespan αν βάζαμε την πιο χρονοβόρα διεργασία με χρόνο 50 σε έναν επεξεργαστή μόνη της. Π.χ., η ανάθεση P1: {50}, P2: {35, 20}, P3: {30, 25} 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 πρέπει να υλοποιήσετε έναν αλγόριθμο ταξινόμησης. Ο αλγόριθμος που θα χρησιμοποιηθεί θα είναι είτε η Heapsort είτε η Quicksort. **Απαγορεύεται να χρησιμοποιήσετε έτοιμες μεθόδους ταξινόμησης που παρέχονται από την Java.** Θα πρέπει να υλοποιήσετε την δική σας μέθοδο, είτε με χρήση πίνακα είτε με χρήση λίστας.

- Αν ο ΑΜ σας λήγει σε άρτιο αριθμό, θα πρέπει να υλοποιήσετε την Heapsort.
- Αν λήγει σε περιττό, θα υλοποιήσετε την Quicksort.
- Αν είστε σε ομάδα και λήγουν και οι 2 ΑΜ σε άρτιο ή και οι 2 σε περιττό, ακολουθείτε τα παραπάνω.
- Σε διαφορετική περίπτωση διαλέξτε εσείς όποια από τις 2 μεθόδους θέλετε.

Μοναδικό παραδοτέο του Μέρους Γ είναι η κλάση Sort.java με τον αλγόριθμο ταξινόμησης (δεν απαιτείται να υπάρχει μέθοδος main εκεί).

Μέρος Δ [30 μονάδες]. Στο μέρος αυτό θα κάνετε μία μικρή πειραματική αξιολόγηση για να διαπιστώσετε ποιος από τους 2 αλγορίθμους είναι καλύτερος στην πράξη, σε σχέση με την επίδοση στο makespan. Χρησιμοποιήστε την τάξη Random του πακέτου java.util και δημιουργήστε τυχαία δεδομένα εισόδου (φτιάξτε δηλαδή τυχαίους χρόνους επεξεργασίας για τις διεργασίες). Χρησιμοποιήστε *τουλάχιστον* 3 διαφορετικές τιμές για τον αριθμό των διεργασιών.

Ενδεικτικά, μπορείτε να χρησιμοποιήσετε N = 100, 250, 500. 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 για κάθε αλγόριθμο, 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.

Πρόσθετες οδηγίες υλοποίησης:

- 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 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 καταγράψετε το makespan που βρήκε κάθε φορά.
- Για την καλύτερη ανάπτυξη του κώδικά, είναι βολικό να χωρίσετε τα προγράμματα σας σε modules ανάλογα με την λειτουργία τους. Για παράδειγμα (δεν είναι υποχρεωτικό να το κάνετε έτσι),
  - Μπορείτε στο Μέρος Β να γράψετε ξεχωριστή μέθοδο που διαβάζει το αρχείο εισόδου και αποθηκεύει τις διεργασίες σε κάποιο πίνακα ή λίστα.
  - Μπορείτε να υλοποιήσετε 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.

Μέρος Ε - Αναφορά παράδοσης [10 μονάδες]. Ετοιμάστε μία σύντομη αναφορά σε pdf αρχείο (μην παραδώσετε Word ή txt αρχεία!) 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, και αν το κάνατε με χρήση πίνακα ή λίστας (άνω όριο μισή 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).

d. Φροντίστε να εξηγήσετε με ποιο τρόπο δίνετε το μονοπάτι για το αρχείο εισόδου, καθώς και οποιαδήποτε πληροφορία κρίνετε απαραίτητη για να γνωρίζουν οι βοηθοί του μαθήματος πώς εκτελείται ο κώδικάς σας. Ως παράδειγμα, αν στο project σας ο κώδικας είναι στο φάκελο src, και το αρχείο εισόδου έχει όνομα inputfile.txt και είναι στον φάκελο Data, η εκτέλεση της main του Μέρους Β από τη γραμμή εντολών μέσα στον φάκελο src μπορεί να γίνεται ως εξής:

```
java Greedy  ../Data/inputfile.txt
```

Το συνολικό μέγεθος της αναφοράς πρέπει να είναι τουλάχιστον 2 σελίδες και το πολύ 6 σελίδες.

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. Χρησιμοποιήστε τα ονόματα των κλάσεων όπως δίνονται. 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. Την αναφορά παράδοσης

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. 3130056_3130066.zip ή 3130056.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.