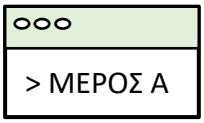
2η Εργασία Δομές Δεδομένων - Report

Μιλτιάδης Τσόλκας : 3200213

Χαράλαμπος Καρακώστας : 3200065

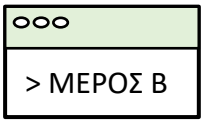


Όσον αφορά το μέρος Α , φτιάξαμε 4 κλάσεις με ονόματα Processor, MaxPQ, TasksList και Task. Στο αρχείο Processor.java έχουμε κατασκευαστή αντικειμένων Processor, όπου ο καθένας έχει και την δικιά του TasksList από την οποία προκύπτει με την χρήση της μεθόδου getActiveTime ο χρόνος που έχει ξοδέψει ο επεξεργαστής με τις διεργασίες που έχει επεξεργαστεί. Η μέθοδος compareTo ενός Processor παίρνει ως όρισμα ένα άλλο αντικείμενο Processor και επιστρέφει την διαφορά των συνολικών χρόνων των δύο αυτών Processors. Επίσης, υπάρχουν και διάφορες μέθοδοι get που χρησιμεύουν σε διάφορα σημεία της εργασίας, όπου θέλουμε πρόσβαση σε στοιχεία αντικειμένων Processor.

Στο αρχείο Task.java έχουμε κατασκευαστή αντικειμένων Task και δύο μεθόδους get, όπου η μία επιστρέφει το ξεχωριστό id του κάθε Task, ενώ η άλλη τον χρόνο της διεργασίας του κάθε Task. Στη συνέχεια, στο αρχείο TasksList.java δημιουργείται μια λίστα από Tasks (η λίστα με τα Task του κάθε Processor) και η εσωτερική κλάση Node φτιάχνει τους κόμβους που περιλαμβάνουν τα διάφορα Task. Η μέθοδος isEmpty επιστρέφει boolean τιμή ανάλογα με το αν η λίστα είναι άδεια, η μέθοδος insertAtBack προσθέτει ένα Task στη TasksList από πίσω. Η peekFromBack επιστρέφει πάντα το Task στο πίσω μέρος της λίστας, ενώ η peekFromFront εκείνο που είναι στο μπροστά μέρος της λίστας.

Στο αρχείο MaxPQ.java δημιουργούμε την ουρά προτεραιότητας στην οποία έχει πάντα μεγαλύτερη προτεραιότητα ο επεξεργαστής με το μικρότερο activeTime. Ο κατασκευαστής αντικειμένων MaxPQ δημιουργεί την ουρά, η οποία χρησιμοποιεί πίνακα αντικειμένων Processor που είναι ο μεγιστοστρεφής σωρός. Η μέθοδος comparison παίρνει ως όρισμα δύο Processor και επιστρέφει αυτόν με την μεγαλύτερη προτεραιότητα, χρησιμοποιώντας την μέθοδο compareTo. Η μέθοδος insert προσθέτει Processor στην ουρά, το μέγεθος της οποίας προσαρμόζεται ανάλογα με το πόσο γεμάτη είναι από την μέθοδο resize, ενώ η μέθοδος removeItem αφαιρεί στοιχείο και ανανεώνει το μέγεθος της ουράς. Επίσης, η μέθοδος max επιστρέφει τον Processor που έχει κάθε φορά την μεγαλύτερη προτεραιότητα. Αντίθετα, η getmax επιστρέφει και αφαιρεί αυτό το στοιχείο ενημερώνοντας το μέγεθος της ουράς. Οι μέθοδοι swim και sink προσαρμόζουν τα στοιχεία της ουράς, έτσι ώστε να διατηρείται συνεχώς η σωστή προτεραιότητα. Τέλος, η μέθοδος swap αλλάζει θέση στα στοιχεία που της δίνοντας ως ορίσματα.

Το αρχείο Greedy.java μεταξύ άλλων υλοποιεί τον Αλγόριθμο 1, ο οποίος εκτελείται αν επιλεχθεί από τον χρήστη ο αριθμός 1. Αρχικά, δημιουργείται αντικείμενο MaxPQ και ένα αντικείμενο BufferedReader που διαβάζει ένα txt αρχείο (πόσοι Processors θα χρειαστούν, πόσα operations θα γίνουν και έπειτα κάθε operation ξεχωριστά). Ενημερώνει την MaxPQ με τους Processors και την λίστα του κάθε Processor με tasks ανάλογα με τα στοιχεία του αρχείου. Στο τέλος της Greedy (αφού έχουμε ελέγξει ότι δεν είναι προβληματικό το αρχείο) ξεκινάμε να αφαιρούμε με την getmax, σε μια δομή επανάληψης, τον Processor που έχει κάθε φορά την μεγαλύτερη προτεραιότητα και προστίθεται ο συνολικός χρόνος που επεξεργάστηκε τις διεργασίες σε μια μεταβλητή makespan που είχε αρχικοποιηθεί με 0. Επιπλέον, αν οι διεργασίες είναι λιγότερες από 50 με κάθε Processor που επιστρέφει η getmax εμφανίζουμε το id, το activeTime του και την λίστα με τους χρόνους των διεργασιών που εκτέλεσε ο συγκεκριμένος Processor, χρησιμοποιώντας τις διάφορες μεθόδους get που έχουμε αναφέρει πιο πάνω.



Όσον αφορά το μέρος Γ της εργασίας, χρησιμοποιήσαμε τον αλγόριθμο ταξινόμησης Quicksort που γράψαμε στο αρχείο Sort.java, λόγω του ότι οι αριθμοί μητρώου μας λήγουν σε περιττό αριθμό. Η υλοποίηση του αλγορίθμου γίνεται με χρήση πίνακα που περιέχει αντικείμενα τύπου Task. Στην Greedy.java δημιουργούμε αντικείμενο τύπου Sort που παίρνει σαν όρισμα ένα πίνακα με Tasks ο οποίος έχει μέγεθος ανάλογο με τα operations. Κάθε operation προστίθεται στον πίνακα των Task ο οποίος ταξινομείται με την κλήση της μεθόδου Quicksort πριν γίνει η εμφάνιση των δεδομένων.

Όσον αφορά το μέρος Δ, καλούμε τον αλγόριθμο του 1ου και 2ου μέρους δίνοντας ως όρισμα το ότι βρισκόμαστε στο μέρος Δ, έτσι ώστε να μην εμφανίζουμε τα αποτελέσματα στο χρήστη, αλλά να τα αποθηκεύουμε για να γίνει σύγκριση και εμφάνιση της αποτελεσματικότητας των 2 αλγορίθμων σε σχέση με την επίδοση τους στο makespan. Χρησιμοποιώντας τις τιμές 100,225 και 625 που δίνουμε στο N για τον αριθμό των διεργασιών, δημιουργούνται 10 αρχεία txt για κάθε τιμή του N και προκύπτει ο παρακάτω πίνακας , ο οποίος μας παρουσιάζει τα average makespan για κάθε τιμή:

N	Greedy	Greedy Decreasing
100	2658	2484
225	6240	5857
625	17589	16893

Από τον παραπάνω πίνακα συνεπάγεται το συμπέρασμα ότι σε κάθε περίπτωση ο αλγόριθμος Greedy Decreasing δίνει μικρότερο makespan από τον αλγόριθμο Greedy. Δηλαδή αυτό σημαίνει ότι και το average makespan είναι μικρότερο από αυτό του Greedy, με αποτέλεσμα να καταλήγουμε στο ότι ο Greedy Decreasing είναι αποδοτικότερος, αφού καθορίζει ορθότερα την προτεραιότητα των διεργασιών και έτσι πραγματοποιούνται όλες σε λιγότερο χρόνο.