ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 2

Τμήμα Πληροφορικής

Χειμερινό Εξάμηνο 2021- 2022

Διδάσκων: Ε. Μαρκάκης

Διονύσιος Ρηγάτος (3200262)
Αθανάσιος Τριφώνης (3200298)

A: Για την υλοποίηση του αλγορίθμου 1 χρησιμοποιήσαμε την ουρά προτεραιότητας ώστε να εισάγουμε σε αυτή τους επεξεργαστές της κλάσης Processor, με τρόπο τέτοιο ώστε ο επεξεργαστής ο οποίος έχει το μικρότερο συνολικό χρόνο εκτέλεσης διεργασιών να είναι αυτός με την μέγιστη προτεραιότητα, κατασκευάζοντας έτσι έναν μεγιστοτρεφή σωρό ώστε να ικανοποιούνται οι απαιτήσεις μας. Το διάβασμα του αρχείου γίνεται μέσω της FileHandler η οποία αναλαμβάνει με τις μεθόδους της να κατασκευάσει έναν πίνακα με όλα τα tasks του αρχείου (έχοντας κάνει όλους τους απαραίτητους ελέγχους αρχικά πως ο αριθμός των tasks στην δεύτερη γραμμή του αρχείου συμφωνεί με τον αριθμό των γραμμών που αντιστοιχούν σε task).
Η FileHandler για τον αλγόριθμο 1 χρησιμοποιεί συγκεκριμένα 2 μεθόδους, την extract_tasks(String filename) η οποία επιστρέφει έναν πίνακα με όλα τα tasks, τον οποίο κατασκεύασε διαβάζοντας γραμμή γραμμή όλες τις γραμμές μετά τις δύο πρώτες του αρχείου, και εισάγοντας κάθε φορά ένα αντικείμενο Task με δύο πεδία, id και time (τα οποία διαβάζονται από την γραμμή του αρχείου σε ένα string και έπειτα χωρίζονται στο κενό ανάμεσα από το id και time). Η δεύτερη μέθοδος που χρησιμοποιεί είναι η processor_count(String filename) η οποία επιστρέφει τον αριθμό των επεξεργαστών που διαβάστηκε στην πρώτη γραμμή του αρχείου, κάνοντας τους απαραίτητους ελέγχους εγκυρότητας για τον αριθμό των tasks.

Το διάβασμα επιτυγχάνεται μέσω του BufferedReader που γεμίζει τον buffer με τα μέρη του αρχείου filename που θέλουμε σε κάθε συνάρτηση από τις προαναφερθέντες.

Ο αλγόριθμος 1 οπότε παίρνει από την FileHandler τον αριθμό επεξεργαστών καθώς και τον πίνακα με τις διεργασίες και τρέχει την μέθοδο Greedy(Task[] tasks, int processors) με αυτά τα δεδομένα. Δημιουργεί την ουρά προτεραιότητας priority_manager και εισάγει με την μέθοδο insert της MaxPQ όσους επεξεργαστές λέει η παράμετρος processors που πήρε, έναν έναν με ID που ξεκινάει από το 0.

Έπειτα για κάθε task του πίνακα tasks κάνει την ανάθεση του στον επεξεργαστή με την μέγιστη προτεραιότητα μέσω της getMax( ) της MaxPQ, ο οποίος είναι αυτός με τον ελάχιστο συνολικό χρόνο εκτέλεσης διεργασιών. Με κάθε ανάθεση διεργασίας η προτεραιότητα μεταβάλλεται, στην παρούσα εργασία κάθε φορά που πρόκειται να αναθέσουμε κάποιο task, χρησιμοποιούμε την getMax( ) για να παρουμε τον επεξεργαστή που θα αναθέσουμε το task και στη συνέχεια του δίνουμε το task μέσω της addTask(Task task) που διαθέτει η κλάση processor. Τέλος τον εισάγουμε ξανά στην ουρά προτεραιότητας στην τελευταία θέση με την μέθοδο insert(T item) της MaxPQ και με τις απαραίτητες αναδύσεις που υλοποιούνται στην MaxPQ μετακινείται στην σωστή θέση βάση της νέας του προτεραιότητας.***** Παρακάτω περιγράφεται αναλυτικότερα η MaxPQ.

Η κλάση MaxPQ επεκτείνει την Comparable και χρησιμοποιεί το PQInterface. Έχει μία μεταβλητή size η οποία καταδεικνύει το μέγεθος της ουράς προτεραιότητας.
Μέθοδοι κλάσης MaxPQ:

- swim(int k): Μετακίνηση κόμβου προς τα πάνω ώστε να πάρει την σωστή του θέση. Συμβαίνει στις εισαγωγές, όσο ο κόμβος έχει μεγαλύτερη τιμή από τον γονιό του και όσο ο κόμβος δεν είναι ρίζα. Αλλάζει θέση ο κόμβος (k) με τον γονιό του (k/2).
- sink(int k): Μετακίνηση κόμβου προς τα κάτω ώστε να πάρει την σωστή θέση. Συμβαίνει στις εξαγωγές, όσο ο κόμβος (k) έχει παιδιά, βρίσκει το μεγαλύτερο

παιδί του και ελέγχει αν είναι μεγαλύτερο από τον k. Αν είναι μεγαλύτερο τότε ανταλλάζουν θέσεις και συνεχίζεται το loop, αν ο κόμβος είναι μεγαλύτερος από το μεγαλύτερο εκ των δύο παιδιών του σταματάει.

- exchange(int i, int j): Ανταλλάζει θέσεις σε δύο αντικείμενα στον πίνακα της ουράς μας.
- insert(T item): Εισάγει αντικείμενο στην τελευταία θέσης της ουράς προτεραιότητας, κάνει resize αν υπάρχει πληρότητα 75% (διπλασιάζοντας την χωρητικότητα στην τιμή του μήκους της ουράς) και έπειτα κάνει swim το αντικείμενο που μπήκε.
- max( ) : Επιστρέφει το πρώτο αντικείμενο της ουράς προτεραιότητας(στην μεγιστοστρεφή σωρό είναι το στοιχείο με δείκτη 1 και έχει την μέγιστη προτεραιότητα) χωρίς να το διαγράψει. Δίνει σφάλμα αν η ουρά προτεραιότητας είναι άδεια.
- getMax( ): Αποθηκεύει το στοιχείο με την μέγιστη προτεραιότητα ώστε να επιστραφεί στο τέλος, το αλλάζει θέση με το τελευταίο στοιχείο της ουράς προτεραιότητας, κάνει το πρώην μέγιστο στοιχείο null, έπειτα κάνει sink το πρώην τελευταίο στοιχείο και τέλος επιστρέφει το αποθηκευμένο στοιχείο που είχε κρατήσει αρχικά. Δίνει σφάλμα αν η ουρά προτεραιότητας είναι άδεια.
- isEmpty( ): Επιστρέφει true αν η ουρά έχει μέγεθος 0, διαφορετικά επιστρέφει false.
- less( ): Επιστρέφει boolean τιμή true αν το αντικείμενο της PQ που βρίσκεται στην θέση left είναι μικρότερο από αυτό στην θέση right της PQ, διαφορετικά false.
- size( ): Επιστρέφει το μέγεθος της PQ.
- resize( ): Μεταβάλει την χωρητικότητα της ουράς προτεραιότητας, δημιουργώντας έναν νέο πίνακα στον οποίο αντιγράφει τα ήδη υπάρχοντα δεδομένα.

Αφού ανατεθούν όλες οι διεργασίες, αν έχουμε λιγότερες από 50 διεργασίες τυπώνουμε έναν έναν τους επεξεργαστές με σειρά προτεραιότητας, το ID τους και τις διεργασίες που περιλαμβλάνει ο καθένας. Ο επεξεργαστής που τυπώνουμε τελευταίο (με την μικρότερη προτεραιότητα) είναι αυτός που καθορίζει το makespan.

Οι επεξεργαστές υλοποιούν την μέθοδο getActiveTime( ) η οποία επιστρέφει τον συνολικό χρόνο εκτέλεσης διεργασιών του συγκεκριμένου επεξεργαστή. Ο κάθε επεξεργαστής έχει μία ElementaryArrayList που λέγεται processedTasks και περιλαμβάνει τις διεργασίες που του έχουν ανατεθεί. Η ElementaryArrayList είναι μία δομή που ουσιαστικά λειτουργεί παρόμοια με μία ArrayList, έχει μεθόδους add, remove, get, size, resize οι οποίες προσθέτουν, αφαιρούν, επιστρέφουν το αντικείμενο από έναν συγκεκριμένο δείκτη, επιστρέφουν το μέγεθος της δομής και μεταβάλλουν την χωρητικότητά της αντίστοιχα.

B: Η υλοποίηση έγινε με τον αλγόριθμο heapsort μέσω πίνακα. Ο αλγόριθμος είναι στην κλάση Sort και παίρνει έναν αταξινόμητο πίνακα τον οποίο μετατρέπει σε σωρό κάνοντας διαδοχικές καταδύσεις στα υποδέντρα ξεκινώντας από τον τελευταίο γονέα. Τέλος επιτυγχάνεται ταξινόμηση μέσω διαδοχικών exchange( ) του πρώτου στοιχείου με το τελευταίο και έπειτα sink( ) ώστε το στοιχείο που έγινε το νέο πρώτο στοιχείο να μετακινηθεί στην σωστή του θέση στην σωρό. Αυτό γίνεται μέχρι να κάνουμε με την σειρά exchange( ) όλα τα στοιχεία της σωρού και το αποτέλεσμα να είναι ένας πίνακας σε φθίνουσα ταξινόμηση. Λόγω του ότι ο πίνακας έχει στοιχεία που ξεκινούν από τον δείκτη 0 ενώ η σωρός από δείκτη 1, έγιναν οι κατάλληλες μετατροπές στις μεθόδους exchange( ) και greater( ) του αλγορίθμου. Η υλοποίηση αυτή πραγματοποιήται απευθείας πάνω στον πίνακα που επιθυμούμε να ταξινομήσουμε, χωρίς βοηθητικούς πίνακες.

C: Στο Μέρος Δ εκτελούνται οι αλγόριθμοι Greedy και Greedy-Decreasing διαδοχικά, χρησιμοποιώντας 8 διαφορετικά task configurations (8 διαφορετικές τιμές για το πλήθος των tasks) για να παρακολουθήσουμε την μεταβολή της βελτίωσης που παρέχει ο αλγόριθμος 2 έναντι του αλγορίθμου 1 σε ένα ευρύτερο σύνολο πλήθους διεργασιών (η ποσοστιαία βελτίωση εμφανίζεται από την σχέση Reduction% = (unsorted average makespan – sorted average makespan) / unsorted average makespan * 100).

Κατασκευάζονται 20 αρχεία ανά configuration ώστε να έχουμε περισσότερο ακριβείς μέσους όρους makespan σε κάθε configuration (λόγω του αυξημένου αριθμού πειραματικών δειγμάτων), στους ο οποίους θα επηρεάζεται λιγότερο η διαφορά χρόνου των δύο αλγορίθμων από κάποιο ή κάποια μεμονωμένα αρχεία λόγω της τυχαιότητας των παραγόμενων χρόνων διεργασίας.

Ο αριθμός των διεργασιών ξεκινάει από 50 στο πρώτο configuration και αυξάνεται επί 1.5 σε κάθε configuration φτάνοντας τα 850 task στο τελευταίο. Η γεννήτρια τυχαίων αριθμών παράγει ακέραιες τιμές από το 1 μέχρι το 100 για τον χρόνο της κάθε διεργασίας.

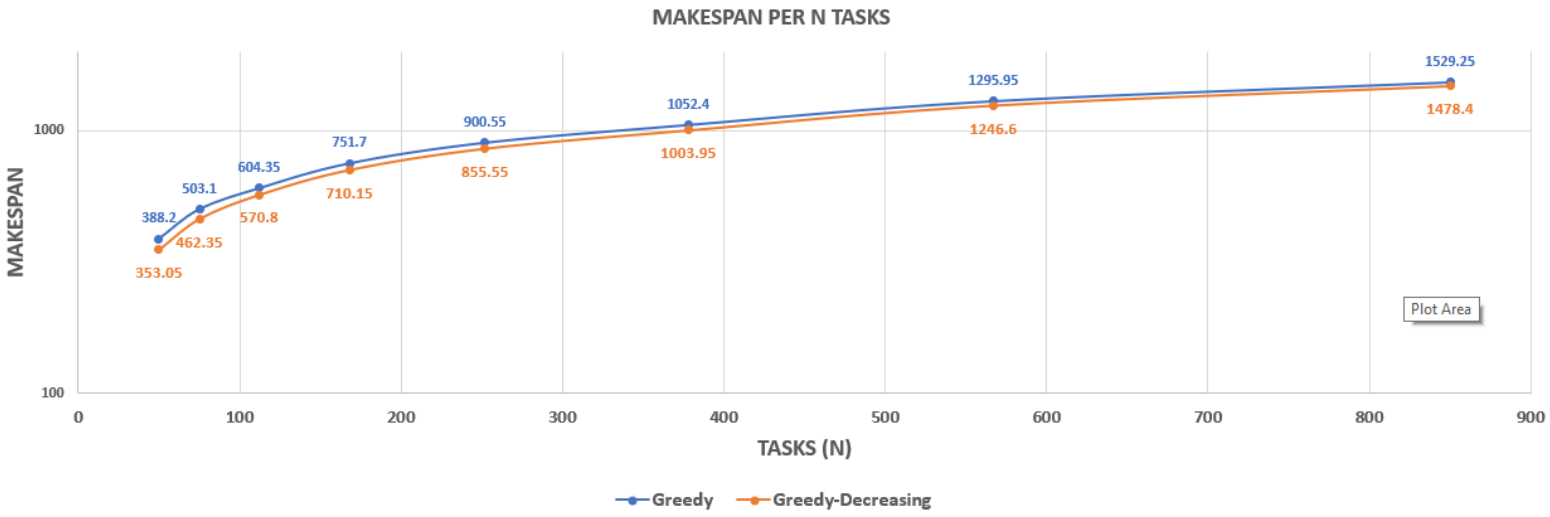
Συγκρίνοντας τα αποτελέσματα των μέσων χρόνων στα διαφορετικά configuration των δύο αλγορίθμων, οι τιμές για το makespan προφανώς ήταν πάντα ταχύτερες για τον αλγόριθμο 2 (Gready-Decreasing) σε σχέση με τον αλγόριθμο 1 (Gready). Ο λόγος που οι ταξινομημένες τιμές των χρόνων διεργασιών αποδίδουν καλύτερο χρόνο makespan στον αλγόριθμο 2 είναι πως ουσιαστικά γλιτώνουμε από την περίπτωση στην οποία κάποιες μεγαλύτερες χρονικά διεργασίες μπορούν να καταφθάσουν τελευταίες και να κατανεμηθούν σε έναν επεξεργαστή που προηγουμένως ίσως είχε κοντινό activeTime με τους υπόλοιπους επεξεργαστές και πλέον θα έχει αυξηθεί λόγω της μεγάλης διεργασίας μονοπωλόντας έτσι το makespan.

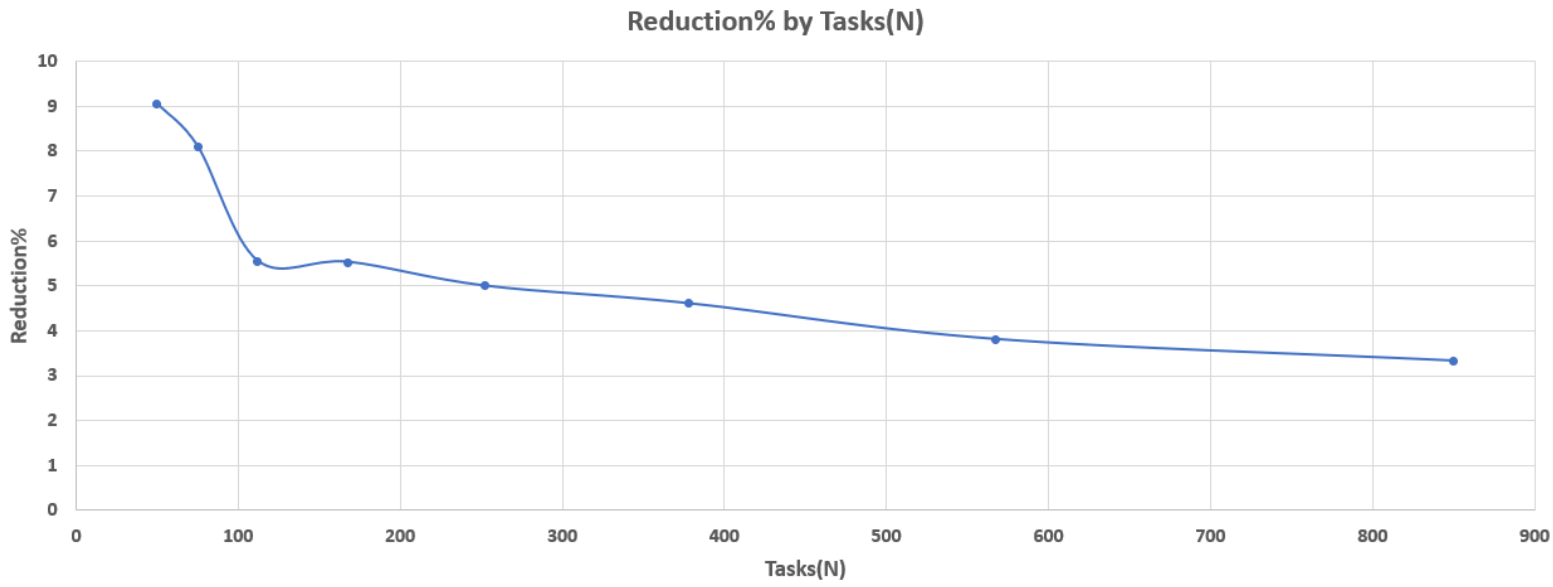
Επίσης παρατηρήθηκε πως όσο αυξανόταν η τιμή του αριθμού διεργασιών N τόσο ο μέσος όρος των makespan που έδιναν οι δύο αλγόριθμοι είχε μικρότερη ποσοστιαία διαφορά (το διάγραμμα ποσοστού βελτίωσης του makespan σε σχέση με τον αριθμό διεργασιών είναι μία φθίνουσα συνάρτηση). Ξεκινώντας από ένα reduction 9.05% χρόνου στις 50 διεργασίες, φτάσαμε σε 3.32% στις 850 από την χρήση του αλγοριθμου 2 έναντι του αλγορίθμου 1.

Συμπερασματικά τόσο από τις τιμές που βγήκαν από τους αλγορίθμους για τις διαφορές στο average makespan, όσο και από τα γραφήματα τους, που φαίνεται πως προσεγγίζουν το ένα το άλλο για μεγαλύτερες τιμές N, η φθίνουσα βελτίωση του χρόνου με τον αλγόριθμο 2 έναντι του αλγορίθμου 1 εξηγείται από το γεγονός πως όσο αυξάνεται το N κατανέμονται περισσότερο ομοιόμορφα οι χρόνοι των διεργασιών στους επεξεργαστές.

Σε κάποιο configuration με λιγότερες διεργασίες, η ταξινόμηση των διεργασιών πρίν την κατανομή τους, βοηθάει στο να επιτευχθεί μία βελτιωμένη κατανομή. Αν όμως έχουμε ένα αρκετά μεγαλύτερο N, το γεγονός πως οι επεξεργαστές δεν αυξάνονται με τον ρυθμό που αυξάνεται το N σε συνδυασμό με το μεγαλύτερο αριθμό τιμών από το ίδιο σύνολο ([1,100] σύνολο από το οποίο παράγονται χρόνοι διεργασιών στην εργασία) θα επέτρεπε να μοιραστούν περισσότερο ομοιόμορφα οι διεργασίες στους επεξεργαστές ακόμα και χωρίς να ταξινομηθούν εκ των προτέρων με αποτέλεσμα να βλέπουμε μικρότερες ποσοστιαία βελτιώσεις μεταξύ των δύο αλγορίθμων στα μεγάλα N.

Tasks(N)	Greedy	Greedy-Decreasing	Reduction %
50	388.2	353.05	9.054611025
75	503.1	462.35	8.099781356
112	604.35	570.8	5.55141888
168	751.7	710.15	5.527471066
252	900.55	855.55	4.996946311
378	1052.4	1003.95	4.603762828
567	1295.95	1246.6	3.808017285
850	1529.25	1478.4	3.325159392





D:

Αλγόριθμος 1 – Greedy.java: Για command line execution αρκέι το **java Greedy data\FILENAME.txt**, όπου στο FILENAME δίνουμε το ανάλογο όνομα.

Αλγόριθμος 2 – Comparisons.java: Για command line execution αρκέι το **java Comparisons**, και για generation καινούργιων testfiles αρκεί το argument GENERATE, **java Comparisons GENERATE**.