ΧΡΉΣΗ ΟΥΡΆΣ ΠΡΟΤΕΡΑΙΌΤΗΤΑΣ

Η υλοποίηση του Αλγορίθμου 1 (Greedy Without Sorting) για τον προγραμματισμό καθηκόντων βασίζεται σε μια στρατηγική που επιτρέπει την αποδοτική κατανομή των εργασιών σε διαθέσιμους επεξεργαστές, χωρίς την ανάγκη ταξινόμησης των δεδομένων. Ο αλγόριθμος απαιτεί μία καθορισμένη μορφή εισόδου, που περιλαμβάνει μια λίστα καθηκόντων, καθένα με χρόνο εκτέλεσης και προτεραιότητα.

Μορφή Εισόδου Δεδομένων

Η είσοδος δεδομένων περιλαμβάνει μια λίστα καθηκόντων, όπου κάθε εργασία αναπαρίσταται από μια δομή δεδομένων που περιλαμβάνει:

- Διαβάζουμε τις διεργασίες και τους χρόνους επεξεργασίας από αρχεία εισόδου μορφής `test_N_X.txt` .
- Το αρχείο περιλαμβάνει αριθμό επεξεργαστών , αριθμό διεργασιών , και για κάθε διεργασία, την τιμή ID και τον χρόνο επεξεργασίας.

Δομές Δεδομένων:

- Χρησιμοποιούμε την ουρά προτεραιότητας `MaxPQ` , που υλοποιείται με σωρό (Max Heap), για να διατηρούμε τους επεξεργαστές ταξινομημένους με βάση τον τρέχοντα φόρτο τους.
- Η κλάση `Processor` περιέχει μία διπλά συνδεδεμένη λίστα (Jobnode) που αποθηκεύει τις διεργασίες κάθε επεξεργαστή.

Ανάθεση Εργασιών:

- Για κάθε διεργασία, εντοπίζεται ο λιγότερο φορτωμένος επεξεργαστής μέσω της μεθόδου `getmax()` της ουράς προτεραιότητας.

Η διεργασία προστίθεται στη λίστα του επεξεργαστή, και ο επεξεργαστής επανεισάγεται στην ουρά προτεραιότητας.

Υπολογισμός Makespan:

- Ο makespan υπολογίζεται ως το μέγιστο φορτίο μεταξύ όλων των επεξεργαστών, που προκύπτει από τη μέθοδο `getTotalProcessingTime()` της κλάσης `Processor` .

ΑΛΓΌΡΙΘΜΟΣ ΤΑΞΙΝΌΜΗΣΗΣ

Για την υλοποίηση του Greedy-Decreasing (Algorithm 2), επιλέξαμε να υλοποιήσουμε δύο αλγορίθμους ταξινόμησης:

HeapSort:

- Χρησιμοποιείται όταν τα δύο ΑΜ είναι άρτια.
- Υλοποιείται με τη χρήση σωρού (Max Heap).

QuickSort:

- Χρησιμοποιείται όταν τα δύο ΑΜ είναι περιττά.
- Βασίζεται στη μέθοδο "divide and conquer" για ταχύτερη ταξινόμηση.

Η ταξινόμηση εφαρμόζεται σε πίνακα τύπου `Job[]`, όπου οι διεργασίες ταξινομούνται σε φθίνουσα σειρά με βάση τον χρόνο επεξεργασίας. Η ταξινόμηση βοηθά στη βέλτιστη ανάθεση των μεγαλύτερων εργασιών πρώτα, μειώνοντας τον makespan.

ΣΥΜΠΕΡΆΣΜΑΤΑ ΑΠΌ ΤΗΝ ΠΕΙΡΑΜΑΤΙΚΉ ΑΞΙΟΛΌΓΗΣΗ

Η πειραματική αξιολόγηση πραγματοποιήθηκε με , με 10 αρχεία εισόδου για κάθε τιμή του . Τα αποτελέσματα συγκρίνονται για τον Greedy και τον Greedy-Decreasing. Ο πίνακας παρακάτω παρουσιάζει τον μέσο makespan για κάθε αλγόριθμο:

Μέγεθος Δεδομένων	Greedy Without Sorting	Greedy-Decreasing
100	563.8	520.3
250	880.6	835.7
500	1186.2	1137.5

Συμπεράσματα:

- Ο Greedy-Decreasing μειώνει τον makespan σε σύγκριση με τον Greedy.
- Η διαφορά στην απόδοση αυξάνεται με το μέγεθος των δεδομένων, καθώς η ταξινόμηση συμβάλλει στην καλύτερη εκχώρηση των μεγαλύτερων διεργασιών.

- Ο HeapSort είναι αποδοτικός για μεγάλα σύνολα δεδομένων, καθώς έχει εγγυημένη πολυπλοκότητα O(n log n).
- Για μικρότερα σύνολα δεδομένων, η επιλογή QuickSort μπορεί να είναι εξίσου αποδοτική, αν και η απόδοσή του εξαρτάται από τη διάταξη των δεδομένων.

ΤΡΌΠΟΣ ΕΚΤΈΛΕΣΗΣ ΚΑΙ ΔΟΜΉ ΕΙΣΌΔΟΥ

Τα αρχεία εισόδου δημιουργούνται από το πρόγραμμα `GenerateTestCases.java`. Κάθε αρχείο περιλαμβάνει:

- Τον αριθμό επεξεργαστών .
- Τον αριθμό διεργασιών .
- Τιμές και για κάθε εργασία.

Αρχιτεκτονική Φακέλων

- Ο κώδικας βρίσκεται στον φάκελο `src`.
- Τα αρχεία εισόδου αποθηκεύονται στον φάκελο `Data`.

Διαδικασία Εκτέλεσης

1. Για τον Greedy:

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

1. Για την πειραματική σύγκριση (Comparisons):

```
java Comparisons
```

ΠΕΙΡΑΜΑΤΙΚΈΣ ΣΥΓΚΡΊΣΕΙΣ

Προκειμένου να γίνουν πειραματικές συγκρίσεις μεταξύ των αλγόριθμων, μπορεί να χρησιμοποιηθεί το ακόλουθο σενάριο:

1. Δημιουργία διαφόρων αρχείων εισόδου με διαφορετικά μεγέθη δεδομένων (π.χ. μικρό, μεσαίο, μεγάλο).
2. Εκτέλεση του Greedy αλγορίθμου για κάθε αρχείο εισόδου.
3. Καταγραφή των αποτελεσμάτων, συμπεριλαμβανομένου του χρόνου εκτέλεσης και του makespan.