

Assignment Report

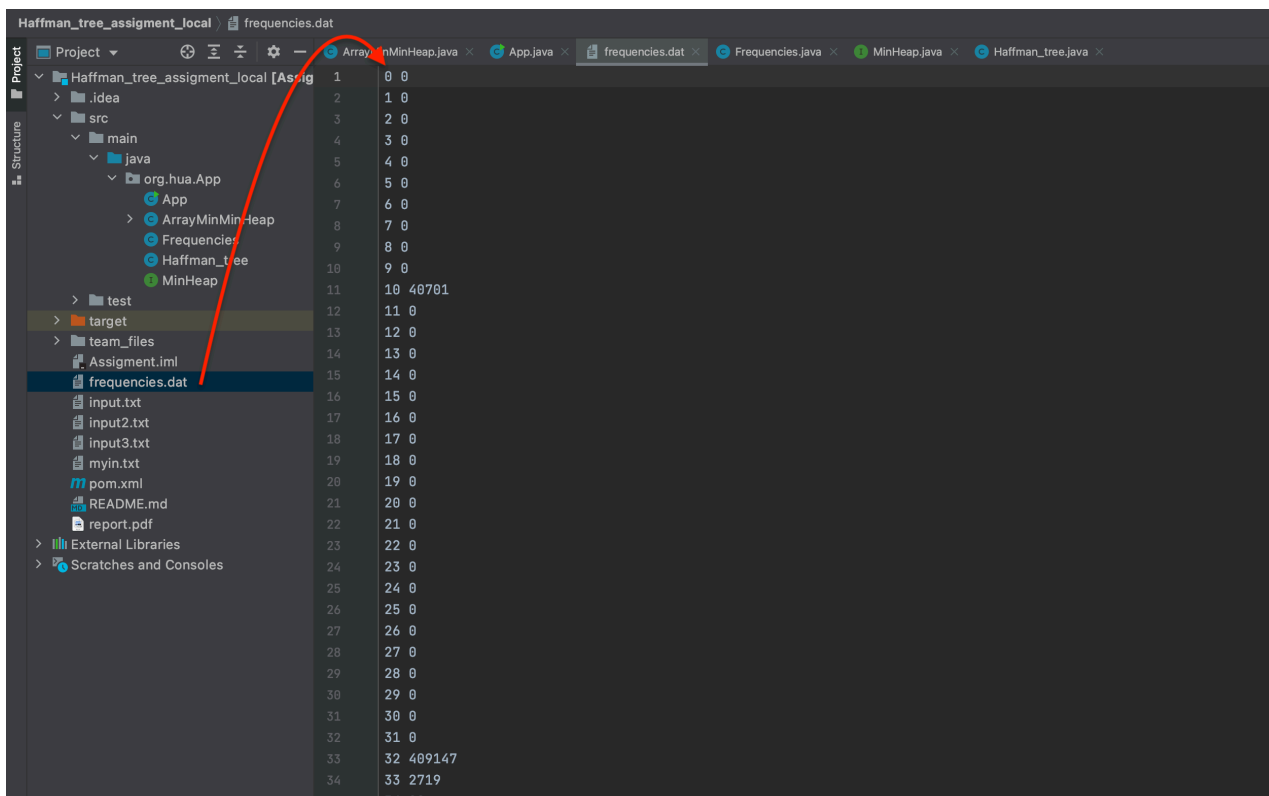
N.Liapis-it21950
A.Chourlias-it219113
C.Zalaxoris-it21922

D.Michail
11 January 2021

■ 1ST PART

Αρχικά Δημιουργήσαμε μια κλάση Frequencies, η οποία μέσω της fileReader και της bufferReader ανοίγει τα αρχεία .txt. Έπειτα δημιουργούμε έναν πίνακα συχνοτήτων, απο long's, για τους ASCII χαρακτήρες και μια private μέθοδο Readcounter η οποία ξεκινάει να διαβάζει με την σειρά κάθε χαρακτήρα μέσα από κάθε αρχείο, την οποία μέθοδο αναλαμβάνει να καλέσει ο constractor για χάρη του χρήστη ώστε στην επόμενη γραμμή του κώδικα ο χρήστης να έχει έναν πίνακα συχνοτήτων(public final count) να επεξεργαστεί. Ακόμη, μέσω της μεθόδου write τυπώνει το αποτέλεσμα σε ένα άλλο αρχείο frequencies.dat, αφού τελειώσει η διαδικασία του writing στο αρχείο εξόδου, μέσω του try-with-resources statement αυτόματα απελευθερώνουμε πόρους οι οποίοι έγιναν κατάληψη ώστε να εκτελεστεί η παραπάνω διαδικασία. Τέλος, στην κύρια κλάση, πρώτα δημιουργούμε 3 αντικείμενα, κάθε αντικείμενο για κάθε αρχείο, με μία δομή for διασχίζουμε τους πίνακες συχνοτήτων και τους προσθέτουμε ώστε να πάρουμε την συνολική συχνότητα από κάθε γράμμα του πίνακα ASCII σύμφωνα με την εμφάνιση του σε κάθε αρχείο(αντικείμενο), και στην συνέχεια καλούμε την write για το αντικείμενο στο οποίο προσθέσαμε τις συχνότητες των άλλων δυο.

Στην συνέχεια, για το πρώτο κομμάτι της εργασίας δίνεται ένα στιγμιότυπο οθόνης(1.1), και στην συνέχεια μία εξήγηση σχετικά με τα αποτελέσματα τα οποία παρουσιάζονται σε αυτό το στιγμιότυπο.



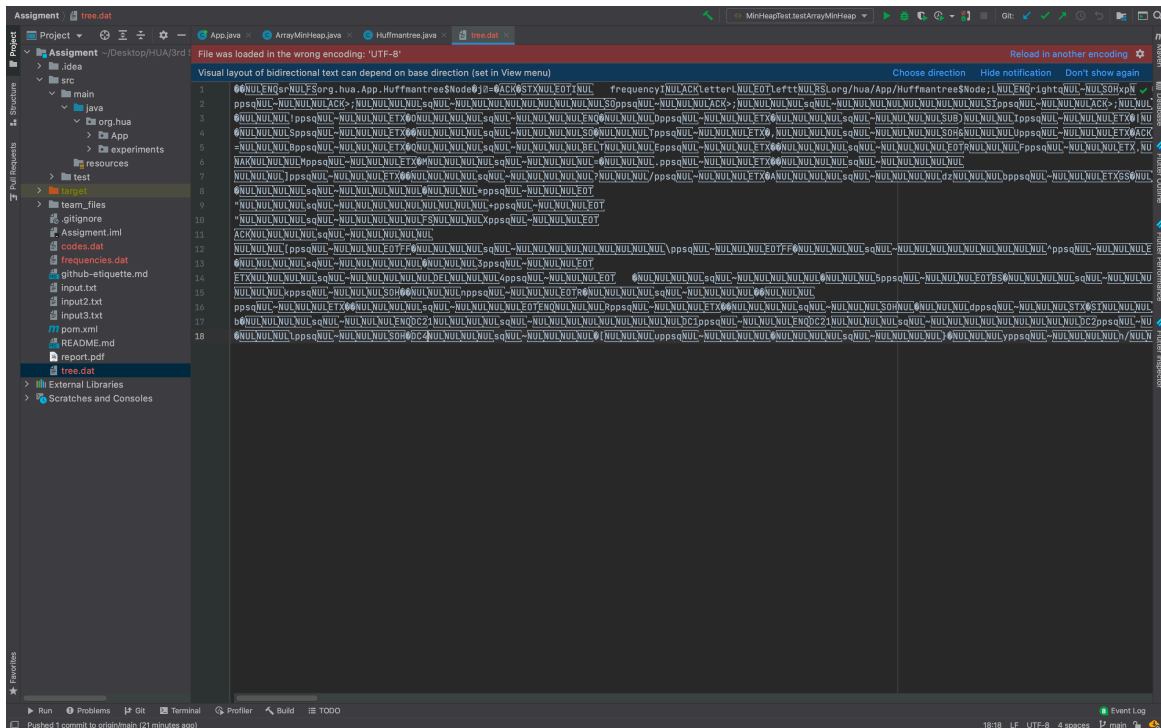
The screenshot shows an IDE window with the project structure on the left and the contents of the frequencies.dat file in the main editor. The project structure includes a main directory with a java subdirectory containing org.hua.App, ArrayMinMinHeap, Frequencies, Huffman_tree, and MinHeap. The frequencies.dat file is highlighted in the project structure. The main editor displays the contents of frequencies.dat, which is a list of numbers representing character frequencies.

Line	Frequency
1	0 0
2	1 0
3	2 0
4	3 0
5	4 0
6	5 0
7	6 0
8	7 0
9	8 0
10	9 0
11	10 40701
12	11 0
13	12 0
14	13 0
15	14 0
16	15 0
17	16 0
18	17 0
19	18 0
20	19 0
21	20 0
22	21 0
23	22 0
24	23 0
25	24 0
26	25 0
27	26 0
28	27 0
29	28 0
30	29 0
31	30 0
32	31 0
33	32 409147
34	33 2719
35	34 22

Στο πιο πάνω στιγμιότυπο βλέπουμε το περιεχόμενο του αρχείου, frequencies.dat, όπου σε κάθε γραμμή υπάρχουν δύο αριθμοί όπου ο πρώτος αναπαριστά την δεκαδική τιμή του χαρακτήρα στον πίνακα ASCII, ενώ ο δεύτερος αριθμός αναπαριστά την συχνότητα του αντίστοιχου χαρακτήρα η οποία έχει υπολογιστεί σύμφωνα με τα παραπάνω

■ 2ND PART

Ερευνώντας την κωδικοποίηση του Huffman αποκτάμε μια ιδέα για το τι θα ακολουθήσει στα επόμενα κομμάτια της άσκησης. Στο συγκεκριμένο κομμάτι υλοποιούμε την δομή στην οποία αποθηκεύονται τα στοιχεία που συλλέξαμε στο προηγούμενο ερώτημα. Συγκεκριμένα διαβάζουμε τις συχνότητες από το αρχείο frequencies.dat και τις εισχωρούμε στην min heap που υλοποιήσαμε. Αυτή όπως είδαμε και στο εργαστήριο είναι ένας διάδικος σωρός στον οποίο υπάρχει προτεραιότητα όπου η ριζά είναι πάντα το ελάχιστο κλειδί μεταξύ όλων των άλλων. Αποτελείται από τις βασικές μεθόδους insert, getMin και extractMin, οι οποίες αντίστοιχα προσθέτουν στοιχεία μέσα σε ένα πίνακα, βρίσκουν το στοιχείο με την ελάχιστη τιμή και το εξάγουν αν χρειαστεί. Φυσικά για να λειτουργήσουν αυτές οι μέθοδοι θα πρέπει να υλοποιηθεί η βασική λειτουργία της min heap η οποία είναι η fixup και η fixdown. Η μέθοδος fixup με την σειρά της ξεκινά να ελέγχει το τελευταίο στοιχείο που προστέθηκε, αν είναι μικρότερο από τον πατέρα του αλλάζει θέση με αυτόν(σύμφωνα με την swap που υλοποιούμε) και συνεχίζεται η ίδια διαδικασία οπότε χρειαστεί. Αντίστοιχα η fixdown ξεκινάει με την ριζά ελέγχει εφόσον έχει παιδιά ποιο από τα δυο της παιδιά είναι μικρότερο και έπειτα ελέγχει αν το στοιχείο της ρίζας είναι μικρότερο και από αυτό. Αν δεν είναι τότε αλλάζει τα στοιχεία μέσω της swap. Παίρνοντας μια γενική ιδέα για την λειτουργία της min heap που υλοποιήσαμε μπορούμε να εξηγήσουμε την δομή του δέντρου Huffman. Αρχικά οφείλουμε να διαβάσουμε το αρχείο των συχνοτήτων από το προηγούμενο κομμάτι της εργασίας. Αυτό το καταφέρνουμε διαβάζοντας σε ένα πίνακα μόνο τις συχνότητες που θα χρειαστούμε και όχι τον χαρακτήρα που αντιπροσωπεύουν. Στην συνέχεια δημιουργούμε την κλάση για την δομή των κόμβων στο δέντρο μας, η οποία αποτελείται από μια μεταβλητή για τις συχνότητες μια για τους χαρακτήρες και τα δυο παιδιά των κόμβων. Δημιουργούμε μεθόδους για να ελέγχουμε αν ο κόμβος έχει παιδιά και έπειτα για την σύγκριση των συχνοτήτων του κάθε κόμβου. Έχοντας λοιπόν δημιουργήσει τους κόμβους είμαστε έτοιμη να υλοποιήσουμε την μορφή ενός δέντρου Huffman. Συγκεκριμένα, δημιουργούμε ένα αντικείμενο για την min heap και ξεκινάμε να εισχωρούμε τις συχνότητες που διαβάσαμε μέσα σε αυτή σε μορφή κόμβων όπου τα παιδιά τους δείχνουν σε null. Στην συνέχεια, όσο η min heap έχει στοιχεία μέσα της εμείς εξάγουμε αρχικά στο αριστερό και έπειτα στο δεξί παιδί του κόμβου τα ελάχιστα στοιχεία και δημιουργούμε με αυτά έναν πατέρα ο οποίος έχει για χαρακτήρα το κενό και για συχνότητα την συνολική από τα δυο παιδιά. Μόλις δημιουργηθεί ο πατέρας τον εισάγουμε πάλι μέσα στην min heap. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να γυρίσει η ριζά η οποία έχει την συνολική συχνότητα όλων των κόμβων. Εφόσον τελειώσουμε με την βασική αυτή λειτουργία οφείλουμε να γραφουμε το αντικείμενο σε σειριακή μορφή μέσα στο



2.1

αρχείο tree.dat. Την διαδικασία αυτή επιτυγχάνουμε μέσω της μεθόδου storeTree η οποία χρησιμοποιεί τα αντικείμενα της java , FileOutputStream και ObjectOutputStream ώστε να αποθηκεύσουν την ρίζα του δέντρου μας. Στην συνέχεια, για το δεύτερο κομμάτι της εργασίας δίνεται ένα στιγμιότυπο οθόνης(2.1), και στην συνέχεια μία εξήγηση σχετικά με τα αποτελέσματα τα οποία παρουσιάζονται σε αυτό το στιγμιότυπο.

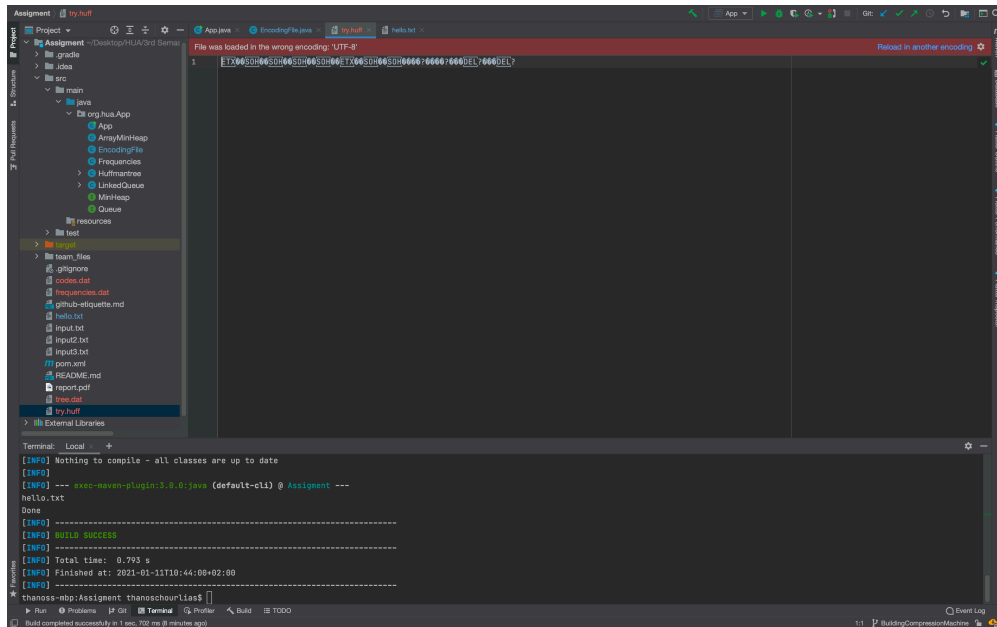
3RD PART

Στο συγκεκριμένο κομμάτι της εργασίας σκοπός ήταν να δημιουργήσουμε εάν πρόγραμμα ώστε να διαβάζουμε το δέντρο Huffman και να αποθηκεύουμε σε εάν αρχείο την δυαδική κωδικοποίηση των συχνοτήτων του κάθε χαρακτήρα του ASCII. Αναλυτικότερα, διαβάζουμε το αρχείο tree.dat, το οποίο περιέχει σε μια καλά optimised από την Oracle μορφή το Σύνθετο αντικείμενο(δέντρο huffman) το οποίο σε προηγούμενο κομμάτι αποθηκεύσανε, με την μέθοδο getTree .Αυτή ουσιαστικά χρησιμοποιεί τα έτοιμα αντικείμενα της java, ObjectInputStream και BufferedInputStream ώστε να διαβάσει το αρχείο και να αντιστρέψει την λειτουργία της storeTree. Έπειτα δημιουργούμε μια μέθοδο buildEncodingMap, η οποία δέχεται ως παράμετρο την ρίζα του δέντρου και γυρίζει εάν HashMap με Integer(Αριθμός ascii) και String(Κωδικοποίηση του χαρακτήρα από το δέντρο Huffman). Μέσω της βοηθητικής μεθόδου lookupTableImpl, με παραμέτρους τον τρέχον κόμβο και μια συνδεδεμένη λίστα που υλοποιήσαμε, ελέγχουμε αν ο κόμβος έχει παιδιά .Αν φυσικά έχει, εκτελούμε αναδρομικά την εξής διαδικασία: ανάλογα αν το παιδί είναι δεξί κάνουμε push string "1" στην LinkedList και έπειτα pop ολόκληρο το string που έχει σχηματιστεί, αλλιώς αν το παιδί είναι αριστερό κάνουμε push string "0" και αντίστοιχα pop. Αν τελικά ο κόμβος δεν έχει παιδιά τότε μέσω του "νέου" for της java και με την βοήθεια του iterator της λίστας τραβάμε από την λίστα ολόκληρο το string που δημιουργήθηκε και το εισχωρούμε στο HashMap .Για να αποθηκεύσουμε στο τέλος το αποτέλεσμα δημιουργούμε μια

μέθοδο `storeEncodingMap`, η οποία μέσω της `bufferedWriter` και μιας επανάληψης για όλους τους χαρακτήρες ASCII γράφει την κωδικοποίηση που περιέχει το `HashMap` μας. Παράλληλα για την λειτουργία του προγράμματος χρησιμοποιήσαμε την `LinkedList` που προαναφέραμε. Αυτή με την σειρά της αποτελείται από μια κλάση `Node`, η οποία αναπαριστά τον κάθε κόμβο της συνδεδεμένης λίστας και αποθηκεύει τα δεδομένα που εισάγονται και την αναφορά στον επόμενο κόμβο. Η λίστα είναι απλά συνδεδεμένη και έχουμε ορίσει `head node` και `tail node` αντίστοιχα, ενώ ταυτόχρονα αποτελείται από τις μεθόδους `push`, `pop`, `first`, `getHead` και `iterator`. Συγκεκριμένα, μέσω της `push` προσθέτουμε το κάθε στοιχείο στη λίστα, ελέγχοντας αν αυτή είναι άδεια όπου το `head` θα γίνει το αντικείμενο αυτό π προσθέσαμε (το πρώτο δηλαδή). Αντιθέτως, προσθέτουμε καινούριο κόμβο στο τέλος της λίστας και αυξάνουμε το μέγεθος της. Έπειτα μέσω της `pop` αφαιρούμε τα στοιχεία από την λίστα τα οποία έχουμε προσθέσει, κάνοντας τους καταλλήλους ελέγχους. Τέλος, για την δημιουργία του `iterator` χρησιμοποιούμε την κλάση `QueueIterator`, στην οποία έχουμε υλοποιήσει τις δυο βασικές μεθόδους ενός `iterator`, `hasNext()` και `next()`, όπου η πρώτη ελέγχει αν υπάρχει επόμενος κόμβος δηλαδή αυτός π είμαστε τώρα δεν δείχνει σε `Null` και η δεύτερη μας επιστρέφει τα περιεχόμενα του κόμβου στον οποίο βρισκόμαστε (σε αυτή την περίπτωση ένα `bit` σε μορφή `string` το οποίο υποδηλώνει ένα μέρος του μονοπατιού το οποίο έχουμε βρει από το δέντρο `huffman`) και ορίζει τον κόμβο που βρισκόμαστε να πάρει την τιμή του επόμενου κόμβου της λίστας (χωρίς να είναι αναγκαίος ο έλεγχος για την ύπαρξη του επόμενου κόμβου).

■ 4TH PART

Τελειώνοντας με τα προηγούμενα κομμάτια της εργασίας δημιουργήσαμε εξ ολοκλήρου την δομή μιας κωδικοποίησης `Huffman`. Συνεπώς στο συγκεκριμένο κομμάτι οφείλουμε να υλοποιήσουμε το πρόγραμμα που εκμεταλλεύεται την δομή αυτή και εισχωρώντας ένα αρχείο εισόδου και ένα εξόδου καταφέρνει να εκτελέσει τις προηγούμενες λειτουργίες και τελικά να συμπίεσει την είσοδο σύμφωνα με την λογική της κωδικοποίησης `huffman`. Ουσιαστικά λοιπόν δημιουργούμε μια νέα κλάση `EncodingFile`, η οποία αναλαμβάνει αυτή την λειτουργία. Συγκεκριμένα αποτελείται από μια μέθοδο `compress` η οποία δέχεται ως παραμέτρους το αρχείο `codes.dat` (περιέχει την κωδικοποίηση κάθε χαρακτήρα `ascii` σύμφωνα με τα αποτελέσματα που μας γύρισε ο αλγόριθμος `huffman` στο αρχείο) και τα αρχεία εισόδου και εξόδου τα οποία δίνει ο χρήστης μέσω του `command line`. Μέσω λοιπόν της `readCodes` διαβάζουμε τις κωδικοποιήσεις `Huffman` κάθε χαρακτήρα από το `codes.dat` και τις γυρνάμε σε ένα πίνακα `string cdmap 128` θέσεων. Έπειτα, διαβάζουμε έναν έναν τους χαρακτήρες μέσα από το `inputFile`, για κάθε χαρακτήρα η `read` μέθοδος του `BufferedReader` αντικειμένου μας γυρνάει τον αριθμό του γράμματος το οποίο διάβασε, στην συνέχεια κρατάμε αυτό τον αριθμό του γράμματος και κάνουμε και έναν τυπικό έλεγχο όπως είχαμε κάνει και σε προηγούμενο κομμάτι (στην μέθοδο `ReadCounter()` της κλάσης `frequencies`) για το αν είναι χαρακτήρας του ASCII. Έπειτα αν μιλάμε για χαρακτήρα ASCII Γράφουμε στο αρχείο εξόδου το οποίο έδωσε ο χρήστης ένα `Byte array` με την βοήθεια της μεθόδου `toByteArray()` η οποία στην ουσία μετατρέπει την έξοδο της `Bitset` σε ένα `Array` από `Bytes/Byte`,



4.1

αυτή τώρα η μετατροπή γίνεται με την βοήθεια της συνάντησης `settingBits()` η οποία δημιουργεί ένα αντικείμενο τύπου `Bitset` και παίρνει ως παράμετρο την κωδικοποιημένη μορφή του γράμματος όπου είναι αποθηκευμένη στον πίνακα `cdmap` και στην συνέχεια φτιάχνει ένα `Byte array` ανάλογου μεγέθους σύμφωνα με τα bits τα οποία διαβάζει από `string` της κωδικοποιημένης μορφής με την βοήθεια ενός `for loop`. Τέλος γυρνάει αυτό το `Byte array` ώστε να το εκμεταλλευτεί η `toByteArray()` μέθοδος. Στην παραπάνω εικόνα (4.1) μπορούμε να διακρίνουμε το αποτέλεσμα από ένα δοκιμαστικό αρχείο το οποίο είχε την πρόταση "Hello world!", το οποίο πολύ λογικά δεν μπορούμε να διαβάσουμε καθώς βρίσκεται σε binary μορφή.