

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ

Τμήμα: Πληροφορικής και Τηλεματικής του Χαροκοπείου
Πανεπιστημίου

Μάθημα: Δομές Δεδομένων

Εξάμηνο: Τρίτο

Υπεύθυνος Καθηγητής: κ. Μιχαήλ

Τίτλος: Huffman Project

Φοιτητές: Ντελής Αθανάσιος, it: 21963

Ντυμένος Ραφαήλ, it: 21965

Πράττη Σταυρούλα, it: 219151

ΠΩΣ ΞΕΚΙΝΗΣΑΜΕ

Η συνεργασία μας έγινε μέσω της πλατφόρμας του TeamViewer. Αφού μελετήσαμε καλά τα περιεχόμενα του συνδέσμου

<https://docs.oracle.com/javase/tutorial/essential/io/index.html>,

προχωρήσαμε στη συγγραφή του κώδικά μας. Το πρόγραμμα υλοποιήθηκε στο περιβάλλον του Apache NetBeans IDE 12.0, συγκεκριμένα στην κατηγορία Java with Maven και επιλογή Java Application, στο μενού, ενώ εκτελείται μέσω του JVM.

ΚΕΦΑΛΑΙΟ 1

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Το πρόγραμμά μας αποτελείται από δυο κλάσεις: την FileHandling.java και την App.java.

Η πρώτη έχει ως λειτουργία να δέχεται και να διαβάζει ένα link. Στην συνέχεια, με την βοήθεια ενός counter τοποθετεί κάθε χαρακτήρα ascii

(από 0 έως 127) στις αντίστοιχες θέσεις ενός πίνακα συχνοτήτων. Έπειτα, κλείνει τα stream που υπάρχουν κι αφού ολοκληρωθεί η δημιουργία του πίνακα συχνοτήτων, τα δεδομένα αποθηκεύονται σε ένα αρχείο με όνομα frequencies.dat.

Μέσα στην App κλάση βρίσκεται η main, η οποία ,με τη σειρά της, περιέχει τον πίνακα συχνοτήτων. Εκεί δημιουργούμε τρία objects της FileHandling και χρησιμοποιούμε την μέθοδο setGutenbergUrl του κάθε object, για να περάσουμε στην κλάση FileHandling τα τρία ζητούμενα link. Στη συνέχεια, με τη βοήθεια της μεθόδου CountLettersFrequency, περνάμε τον πίνακα συχνοτήτων, ώστε να πάρουμε τα αποτελέσματα που επιθυμούμε.

~

Αποφασίσαμε να δημιουργήσουμε την κλάση FileHandling, καθώς παρατηρήσαμε πως θα χρειαστεί να χρησιμοποιήσουμε τον ίδιο κώδικα τρεις φορές για τα τρία links. Έτσι καλώντας αυτήν την μέθοδο αποφύγαμε την επανάληψη κώδικα, εκμεταλλευόμενοι την κληρονομικότητα της java.

ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

Δείγμα από Frequency.dat:

```
Open  ▾  [?]  frequencies.dat  Save  ≡  _  □  X
~/prog/J/data_struct/project/huffman

1 Letter with ASCII 0 appered: 0 times
2 Letter with ASCII 1 appered: 0 times
3 Letter with ASCII 2 appered: 0 times
4 Letter with ASCII 3 appered: 0 times
5 Letter with ASCII 4 appered: 0 times
6 Letter with ASCII 5 appered: 0 times
7 Letter with ASCII 6 appered: 0 times
8 Letter with ASCII 7 appered: 0 times
9 Letter with ASCII 8 appered: 0 times
10 Letter with ASCII 9 appered: 0 times
11 Letter with ASCII 10 appered: 40701 times
12 Letter with ASCII 11 appered: 0 times
13 Letter with ASCII 12 appered: 0 times
14 Letter with ASCII 13 appered: 40701 times
15 Letter with ASCII 14 appered: 0 times
16 Letter with ASCII 15 appered: 0 times
17 Letter with ASCII 16 appered: 0 times
18 Letter with ASCII 17 appered: 0 times
19 Letter with ASCII 18 appered: 0 times
20 Letter with ASCII 19 appered: 0 times
21 Letter with ASCII 20 appered: 0 times
22 Letter with ASCII 21 appered: 0 times
23 Letter with ASCII 22 appered: 0 times
24 Letter with ASCII 23 appered: 0 times
25 Letter with ASCII 24 appered: 0 times
26 Letter with ASCII 25 appered: 0 times
27 Letter with ASCII 26 appered: 0 times
28 Letter with ASCII 27 appered: 0 times
29 Letter with ASCII 28 appered: 0 times
30 Letter with ASCII 29 appered: 0 times
31 Letter with ASCII 30 appered: 0 times
32 Letter with ASCII 31 appered: 0 times
33 Letter with ASCII 32 appered: 409147 times
34 Letter with ASCII 33 appered: 2719 times
35 Letter with ASCII 34 appered: 22 times
36 Letter with ASCII 35 appered: 4 times
37 Letter with ASCII 36 appered: 8 times
```

ΠΗΓΕΣ

<https://docs.oracle.com/javase/tutorial/essential/io/index.html>

<https://gist.github.com/DomPizzie/7a5ff55ffa9081f2de27c315f5018afc/>

<https://www.youtube.com/watch?v=pTCROLZLhDM/>

<https://alvinalexander.com/blog/post/java/how-open-read-url-java-url-class-example-code/>

ΚΕΦΑΛΑΙΟ 2

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Σε πρώτο στάδιο θέλαμε να ανοίξουμε το αρχείο που δημιουργήσαμε στο προηγούμενο μέρος. Αυτό πραγματοποιήθηκε μέσω του tutorial I/O του site της Oracle

(<https://docs.oracle.com/javase/tutorial/essential/io/index.html>).

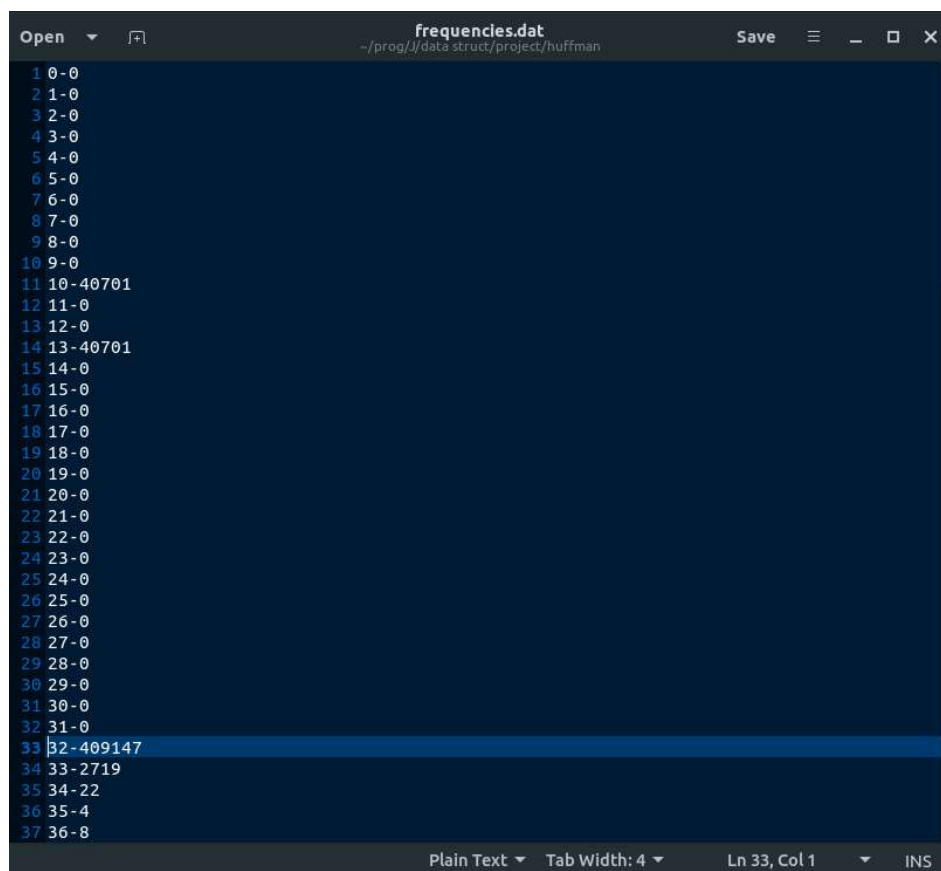
Δημιουργήσαμε τις εξής κλάσεις: HuffmanTree, TreeNodes και πήραμε απ' το εργαστήριο την Interface MinHeap και την κλάση MinHeapArray.

- HuffmanTree: Στην μέθοδό της getFileData ανοίγουμε το αρχείο frequencies.dat με έναν BufferedReader και ανά γραμμή παίρνουμε το πρώτο στοιχείο ως τον χαρακτήρα ascii, το δεύτερο ως τη συχνότητά του και τα αποθηκεύουμε σε ένα νέο object του TreeNodes. Τέλος προσθέτει κάθε node που δημιουργήθηκε σε μια priority queue και κλείνει το stream. Ενώ στη μέθοδό της huffmanCoding παίρνουμε την priority queue που δημιουργήθηκε από την getFileData και φτιάχνουμε το Huffman Tree. Η ιδέα είναι ότι εξάγουμε το node με την μικρότερη συχνότητα από το priority queue, το αποθηκεύουμε προσωρινά και το διαγράφουμε από την ουρά. Στη συνέχεια, εξάγουμε πάλι το πλέον μικρότερο node, το αποθηκεύουμε προσωρινά και το διαγράφουμε από την ουρά. Εφόσον πλέον έχουμε τα δυο node με την μικρότερη συχνότητα τα βάζουμε σε μια μεταβλητή fusedNode ως το αριστερό και το δεξί παιδί (η μικρότερη συχνότητα πάει στο αριστερό και η άλλη στο δεξί), ενώ στο ASCIIletter έχουμε βάλει μια μεταβλητή εκτός ορίων ίση με -1 και στο frequency το άθροισμα των συχνοτήτων των δυο παιδιών. Όλη αυτή η διαδικασία επαναλαμβάνεται μέχρι να μείνει ένα node μέσα στην ουρά που θα είναι το ολοκληρωμένο δέντρο μας. Τέλος, αποθηκεύουμε όλο το δέντρο σε ένα αρχείο tree.dat, με τη βοήθεια της interface Serializable.
- TreeNodes: Περιέχει τις μεταβλητές ASCIIletter, frequency, leftChild και rightChild , setters και getters για αυτές αλλά και μια απλή υλοποίηση της compareTo για την σύγκριση των node.

Η ουρά που χρησιμοποιούμε είναι ο κώδικας της MinHeapArray και το Interface MinHeap, που υλοποιήσαμε στο εργαστήριο 3 του μαθήματος, τον οποίο μελετήσαμε.

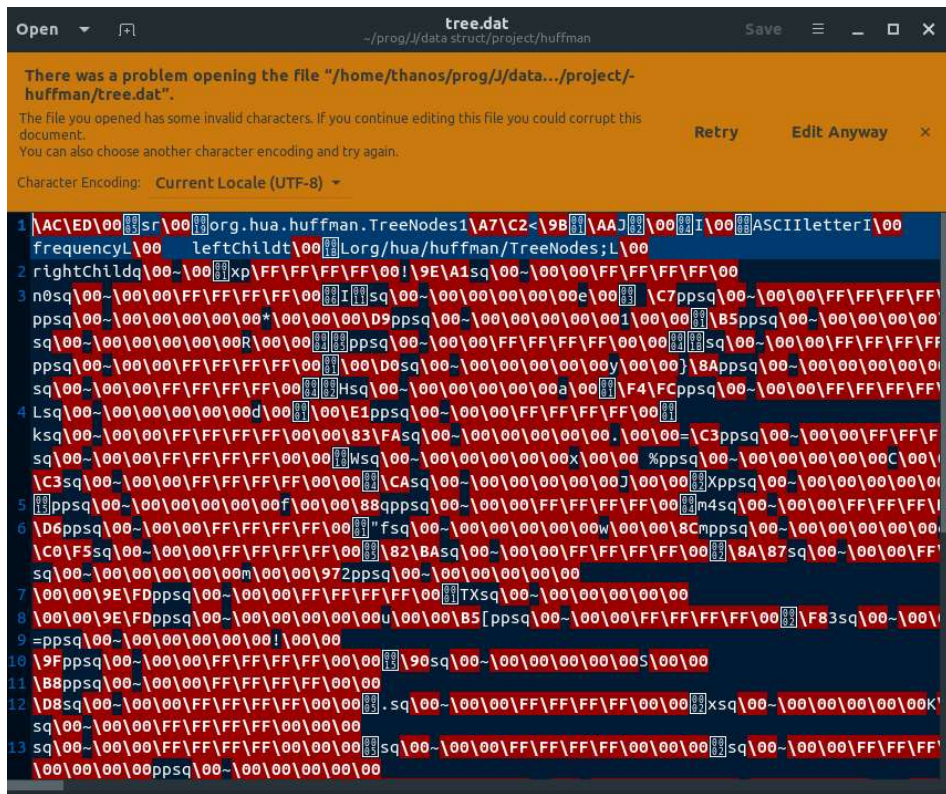
ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

Δείγμα από Frequency.dat:



```
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 13-40701
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
36 35-4
37 36-8
```

Δείγμα από tree.dat:



ΠΗΓΕΣ

<https://docs.oracle.com/javase/tutorial/essential/io/index.html>

https://www.tutorialspoint.com/java/io/bufferedReader_readline.htm

<https://www.guru99.com/how-to-split-a-string-in-java.html>

[https://docs.oracle.com/javase/tutorial/essential/
io/objectstreams.html](https://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html)

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

ΚΕΦΑΛΑΙΟ 3

(Στο δεύτερο μέρος ξεχάσαμε κάθε φορά που δημιουργούμε ένα νέο `fusedNode` να το προσθέτουμε ξανά στην ουρά μας `queue`. Σε αυτό το μέρος το διορθώσαμε, όπως και τη φωτογραφία στις πηγές του κεφαλαίου 2..)

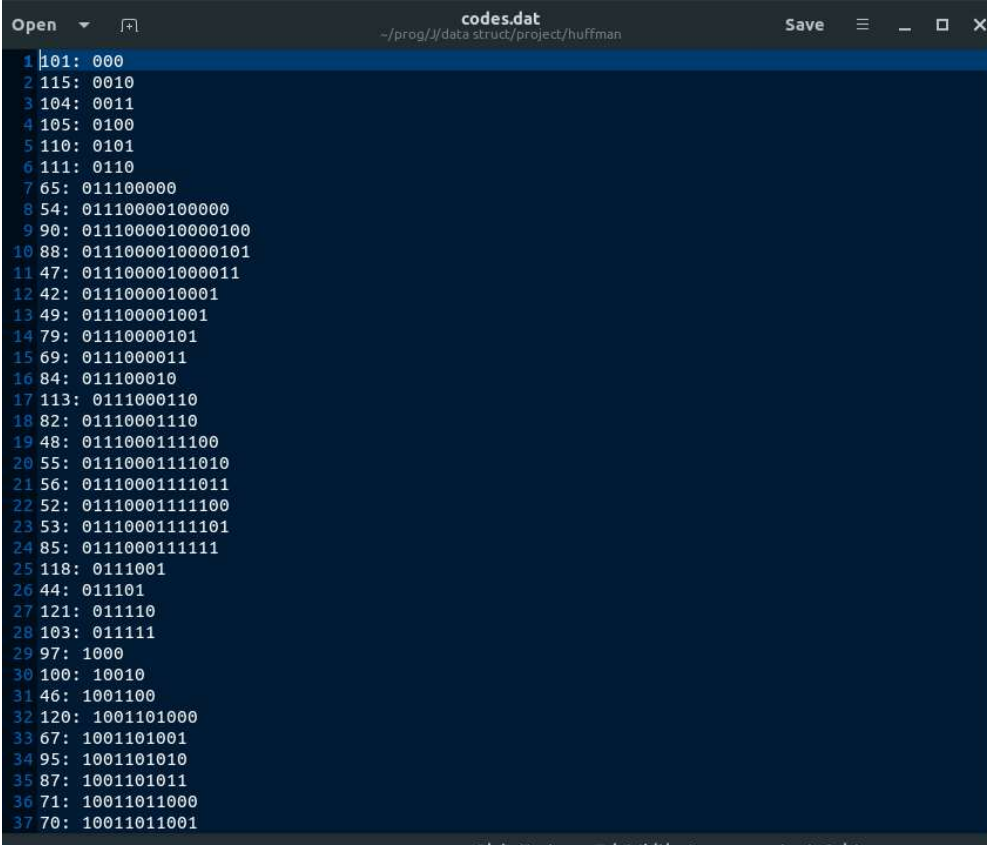
ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Σε αυτό το μέρος δημιουργήσαμε μια νέα κλάση, την `HuffmanTreeCoding` και μέσα της ορίζουμε ένα `treeNodes`, το οποίο θα κρατάει την ρίζα του δέντρου. Δημιουργούμε την μέθοδο `coding`, που δημιουργεί ένα νέο αντικείμενο της κλάσης `HuffmanTree` και καλεί την μέθοδο της `huffmanCoding` κι έπειτα ανοίγουμε το αρχείο `tree.dat` για να διαβάσουμε από μέσα του το δέντρο μας (έχοντας μελετήσει το link με το tutorial της oracle για το διάβασμα αντικειμένων από αρχείο, ξέρουμε ότι με το να πάρουμε μόνο την ρίζα παίρνουμε και όλο το δέντρο, αφού σε κάθε node υπάρχει αναφορά σε άλλα nodes). Εφόσον έχουμε πάρει πλέον το δέντρο ξεκινάμε την διάσχυση, φτιάχνοντας μια νέα μέθοδο για τον λόγο αυτό, την `treeTraverse`. Ξεκινώντας από το αριστερό παιδί του `root`, ξέρουμε ότι κάνουμε όλη τη διάσχυση του αριστερού μέρους της ρίζας, οπότε στην `treeTraverse` περνάμε το παιδί αυτό κι ένα 0 που είναι το κοινό πρώτο στοιχείο των κωδικοποιήσεων των αριστερών, `ascii` γραμμάτων, φύλλων. Στην `treeTraverse` πρώτα ελέγχουμε εάν έχουμε φτάσει στο τέλος του δέντρου και επιστρέφουμε, εάν όχι συνεχίζουμε και ελέγχουμε εάν έχουμε φτάσει σε φύλλο. Σε

αυτήν την περίπτωση γράφουμε στο αρχείο codes.dat το γράμμα του φύλλου και την κωδικοποίηση μέχρι εκεί. Για να φτάσουμε μέχρι το δέντρο καλούμε αναδρομικά την treeTraverse, πρώτα για τα αριστερά παιδιά (όπου προσθέτουμε και σε ένα string ένα επιπλέον μηδενικό για την σωστή κωδικοποίηση) και έπειτα για τα δεξιά (όπου και πάλι στο ίδιο string προσθέτουμε έναν επιπλέον άσσο αυτή τη φορά). Με αυτόν τον τρόπο έχουμε επιτύχει την διάσχυση και κωδικοποίηση όλων των αριστερών από τη ρίζα φύλλων. Με τον ίδιο αναδρομικό τρόπο κάνουμε τις κωδικοποιήσεις και για τα δεξιά από τη ρίζα φύλλα, περνώντας στην treeTraverse αρχικά το δεξί παιδί της αλλά και έναν άσσο, αφού όλα τα δεξιά ξεκινούν με κωδικοποίηση 1. Να προσθέσουμε πως παρατηρήσαμε ότι κάθε φορά που ξανατρέχαμε τον κώδικα και υπήρχε ήδη το αρχείο codes.dat έκανε append και γι' αυτό πριν ξεκινήσουμε την διάσχυση στην coding, το διαγράψαμε.

ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

Δείγμα από codes.dat



```
1 h: 000
2 i: 0010
3 t: 0011
4 o: 0100
5 n: 0101
6 l: 0110
7 c: 011100000
8 s: 01110000100000
9 p: 0111000010000100
10 r: 0111000010000101
11 u: 011100001000011
12 a: 0111000010001
13 q: 011100001001
14 f: 01110000101
15 g: 0111000011
16 b: 011100010
17 m: 0111000110
18 k: 01110001110
19 x: 0111000111100
20 v: 01110001111010
21 y: 01110001111011
22 z: 01110001111100
23 j: 01110001111101
24 w: 0111000111111
25 e: 0111001
26 d: 011101
27 N: 011110
28 3: 011111
29 7: 1000
30 0: 10010
31 6: 1001100
32 2: 1001101000
33 8: 1001101001
34 9: 1001101010
35 5: 1001101011
36 1: 10011011000
37 4: 10011011001
```


ΠΗΓΕΣ

<https://docs.oracle.com/javase/tutorial/essential/io/index.html>

<https://www.baeldung.com/java-write-to-file>

<https://www.geeksforgeeks.org/dfs-traversal-of-a-tree-using-recursion/>

<https://java2blog.com/add-character-to-string-java/>

<https://www.geeksforgeeks.org/delete-file-using-java/>

ΚΕΦΑΛΑΙΟ 4

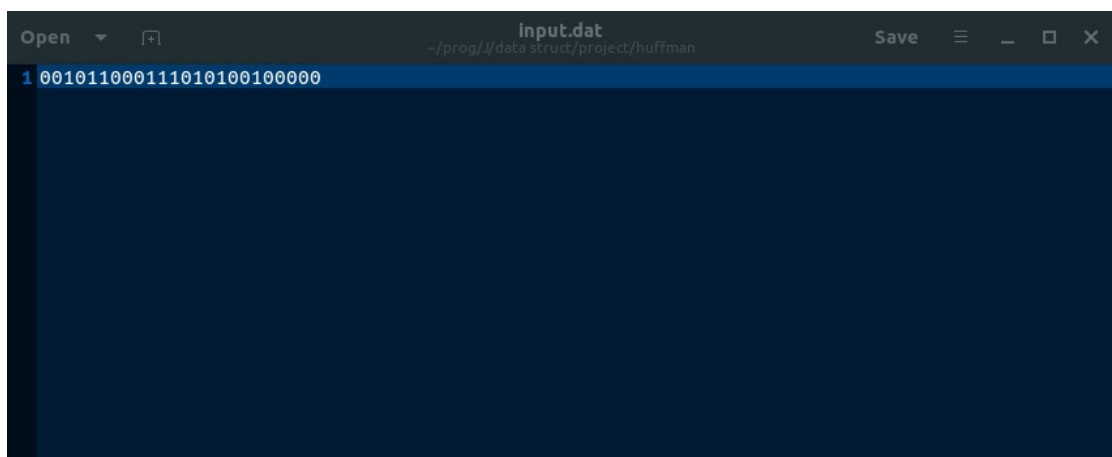
ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Σε αυτό το μέρος κληθήκαμε να υλοποιήσουμε έναν κωδικοποιητή. Συνεπώς δημιουργήσαμε μια νέα κλάση, την Encoder, η οποία περιέχει την readASCII που παίρνει ως παράμετρο τις παραμέτρους της γραμμής εντολών. Αρχικά ελέγχει εάν οι παράμετροι αυτοί είναι 2 σε πλήθος, εάν όχι το πρόγραμμα τερματίζει. Σε αντίθετη περίπτωση ανοίγουμε το αρχείο εισόδου σε πρώτη φάση για να μετρήσουμε πόσο θα είναι το μέγεθος σε bits του αρχείου εξόδου. Αυτό το πετυχαίνουμε παίρνοντας κάθε φορά μια οχτάδα από το αρχείο εισόδου και κρατώντας σε έναν αθροιστή το μέγεθος σε χαρακτήρες της αντίστοιχης Huffman κωδικοποίησης της. Τον αθροιστή αυτόν τον γράφουμε στην πρώτη θέση του αρχείου εξόδου. Στη συνέχεια ανοίγουμε το πρώτο αρχείο που παίρνουμε ως είσοδο, κρατάμε το μήκος του και το κάνουμε προσπέλαση, κρατώντας κάθε φορά 8 bits σε έναν πίνακα τύπου byte. Έχοντας πλέον στη μνήμη τα 8 bits που αντιπροσωπεύουν έναν χαρακτήρα ascii, μετατρέπουμε τον πίνακα σε μια ακολουθία string, ώστε στη συνέχεια να την περάσουμε σαν παράμετρο στην μέθοδο parseInt και να το μετατρέψουμε στην αντίστοιχη δεκαδική του μορφή και τέλος καλούμε την μέθοδο searchCodesFile. Αυτή η μέθοδος ανοίγει το αρχείο που ήδη έχουμε φτιάξει, το codes.dat, το διαβάζει ανά γραμμή και κάνει διαχωρισμό σε αριστερό και δεξί μέρος, με

διαχωριστικό την παύλα(-) χρησιμοποιώντας τη μέθοδο `split` που μας παρέχει η κλάση `String`. Τα δυο αυτά μέρη τα κρατάμε σε δυο θέσεις ενός πίνακα. Έπειτα συγκρίνουμε την πρώτη θέση του πίνακα που φτιάξαμε προηγουμένως, που είναι ο δεκαδικός αριθμός που αντιστοιχεί στην κωδικοποίηση Huffman στο αρχείο `codes.dat`, με τη δεκαδική αναπαράσταση του χαρακτήρα του αρχείου εισόδου που δημιουργήσαμε προηγουμένως στην μέθοδο `readASCII`, και την οποία μετατρέψαμε σε `String` για να μπορούμε να κάνουμε σύγκριση μεταξύ ίδιων τύπων. Εάν βρεθεί ο δεκαδικός που ψάχνουμε τότε παίρνουμε το δεύτερο στοιχείο του πίνακα που κάναμε `split` από το αρχείο `codes.dat` και το γράφουμε στο αρχείο εξόδου της γραμμής εντολών. Έτσι δημιουργούμε ένα αρχείο ίδιο με αυτό της εισόδου αλλά σε κωδικοποίηση Huffman με βοηθητικό το αρχείο `codes.dat` (όπου στην αρχή του περιέχει το μέγεθος του αρχείου). Να τονίσουμε ότι οτιδήποτε βρίσκεται εκτός του εύρους του 0-127 `ascii` δεν το λαμβάνουμε υπόψη.

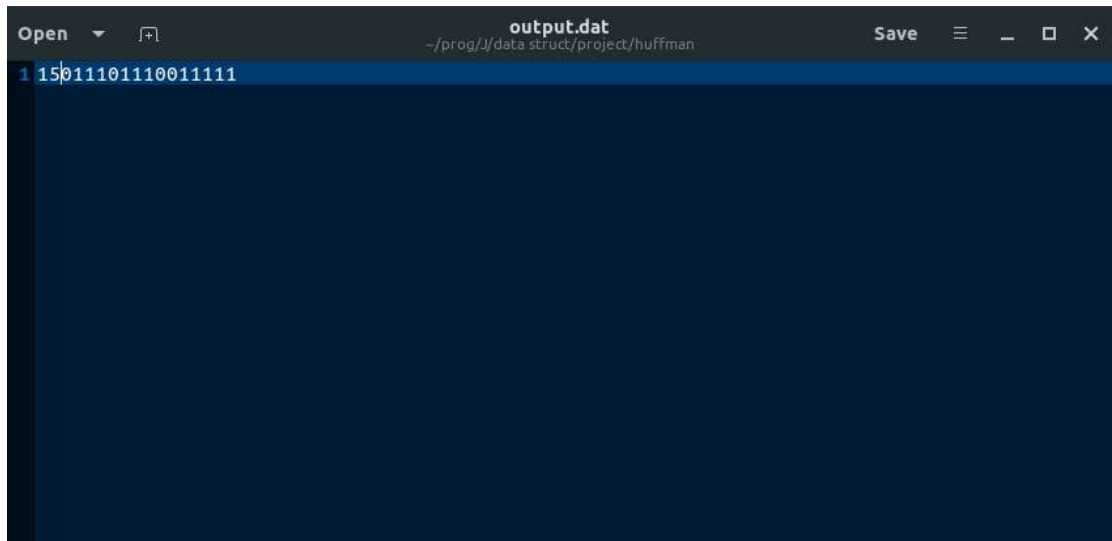
ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

Δείγμα από ένα τυχαίο `input.dat` (που περιέχει τους χαρακτήρες 44,117,32 σε κωδικοποίηση ASCII)



```
Open  ▾  [icon]  input.dat  Save  ≡  _  □  ✕
~/prog/4/data_struct/project/huffman
1 001011000111010100100000
```

Δείγμα από ένα output.dat βασισμένο στο παραπάνω input.dat



```
Open  ▾  [icon]  output.dat  Save  ≡  -  □  ×
~/prog/J/data struct/project/huffman
1 150111101110011111
```

ΠΗΓΕΣ

<https://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html>

<https://stackoverflow.com/questions/9168759/netbeans-how-to-set-command-line-arguments-in-java>

<https://www.roseindia.net/java/javafile/java-read-binary-file-into-byte-array.shtml>

<https://www.javatpoint.com/java-binary-to-decimal>

<https://www.javatpoint.com/java-int-to-string>

<https://stackoverflow.com/questions/14478968/get-total-size-of-file-in-bytes>

ΚΕΦΑΛΑΙΟ 5

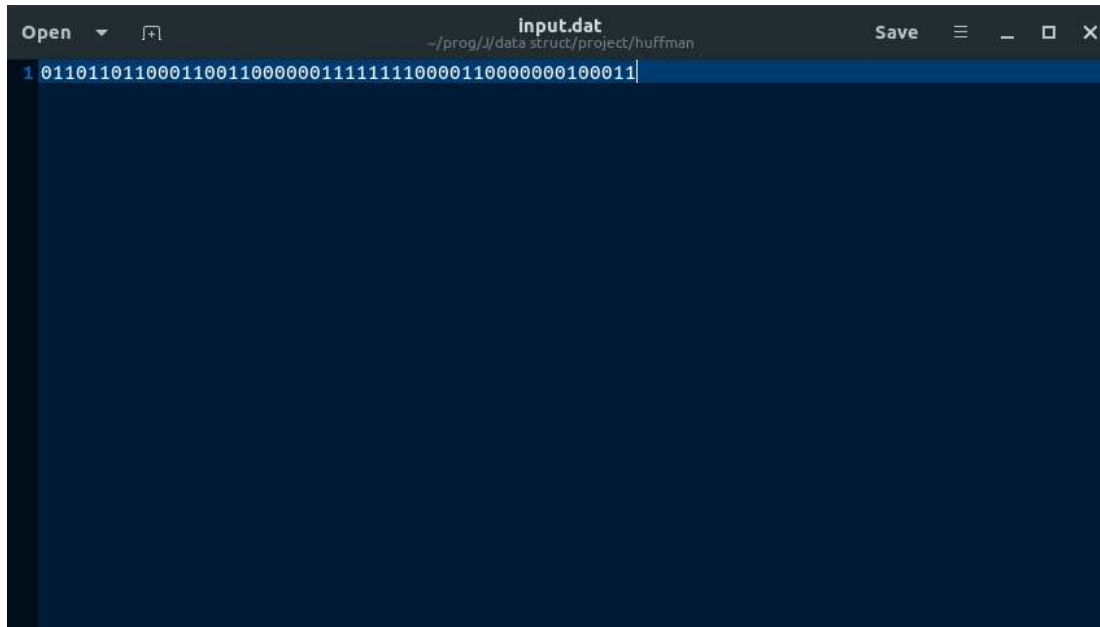
ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΚΩΔΙΚΑ

Για την υλοποίηση αυτού του μέρους χρειάστηκε να δημιουργήσουμε μια δεύτερη main. Οπότε εάν θέλουμε να εκτελέσουμε τον κωδικοποιητή θα πρέπει να εκτελέσουμε την main της κλάσης AppEncoder, ενώ εάν θέλουμε τον αποκωδικοποιητή επιλέγουμε την main της κλάσης AppDecoder.

Στο τελευταίο μέρος της εργασίας υλοποιήσαμε τον ζητούμενο αποκωδικοποιητή. Γι' αυτόν τον λόγο δημιουργήσαμε την νέα κλάση Decoder. Η Decoder περιέχει την μέθοδο readHuffman, η οποία στην αρχή ελέγχει τον αριθμό των παραμέτρων και διαγράφει το αρχείο εξόδου –εάν υπάρχει- για να μην γίνει append. Στην συνέχεια ανοίγει το αρχείο εισόδου, κρατάει το length του και περνάει ένα ένα τα στοιχεία σε έναν πίνακα ακεραίων. Είχαμε ένα πρόβλημα με την μέθοδο read, η οποία μας επέστρεφε τα στοιχεία 0 και 1 σε δεκαδική μορφή ascii (48,49), το οποίο λύθηκε με ένα if-else, οπότε πλέον ο παραπάνω πίνακας περιέχει 0 και 1. Αφού συμπληρώσουμε τον πίνακα, τον περνάμε στην μέθοδο traverseTreeFile. Η μέθοδος αυτή ανοίγει το αρχείο tree.dat που δημιουργήσαμε στο δεύτερο μέρος και αποθηκεύει την ρίζα από την οποία μπορούμε να διασχίσουμε κι όλο το δέντρο. Έχοντας πλέον το δέντρο, ξεκινάμε από την ρίζα του, την κρατάμε σε ένα currentNode και ελέγχουμε ένα ένα τα στοιχεία του πίνακα που αποθηκεύσαμε το αρχείο εισόδου, εάν είναι 0 πηγαίνουμε στο αριστερό παιδί και βάζουμε αυτό σαν currentNode, ενώ εάν είναι 1 βάζουμε το δεξί σαν currentNode. Αυτή τη διαδικασία την επαναλαμβάνουμε μέχρι να φτάσουμε σε ένα φύλλο. Από το φύλλο πέρνουμε τον ascii χαρακτήρα που περιέχει, τον συμπληρώνουμε αριστερά με 0 (εάν χρειάζεται), τον γράφουμε στο αρχείο εξόδου και κάνουμε “reset” το δέντρο πηγαίνοντας το currentNode ξανά πίσω στην ρίζα. Όλη αυτή τη διαδικασία την επαναλαμβάνουμε μέχρι να τελειώσουν τα στοιχεία του πίνακα. Έτσι τελικά παίρνουμε ως έξοδο ένα αρχείο βασισμένο στο αρχείο εισόδου αλλά σε κωδικοποίηση ascii.

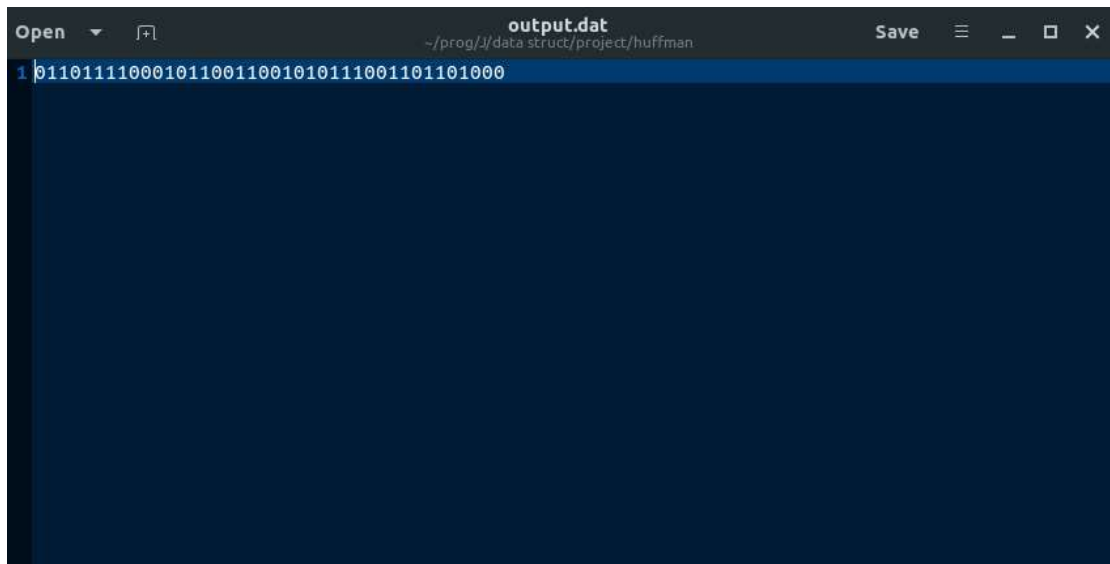
ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ

Δείγμα από ένα τυχαίο input.dat (που περιέχει τους χαρακτήρες 111,22,101,115,104 σε κωδικοποίηση huffman)

A screenshot of a text editor window titled 'input.dat' with a dark blue background. The window has a menu bar with 'Open', 'Save', and window control icons. The main text area shows a single line of binary code: 011101101100011001100000011111110000110000000100011. A cursor is positioned at the end of the line.

```
1 011101101100011001100000011111110000110000000100011
```

Δείγμα από ένα output.dat βασισμένο στο παραπάνω input.dat

A screenshot of a text editor window titled 'output.dat' with a dark blue background. The window has a menu bar with 'Open', 'Save', and window control icons. The main text area shows a single line of binary code: 0110111100010110011001010111001101101000. A cursor is positioned at the end of the line.

```
1 0110111100010110011001010111001101101000
```

ΠΗΓΕΣ

<https://www.javatpoint.com/java-decimal-to-binary>

<https://stackoverflow.com/questions/22539995/how-to-add-0s-in-front-of-a-binary-number>

<https://www.rapidtables.com/convert/number/binary-to-decimal.html>

~

ΤΕΛΟΣ