ΟΜΑΔΑ 0 Πέμπτο Μέρος – Ολοκλήρωση Εργασίας Huffman

- ⭐ Εργασία Δομών Δεδομένων (2020-2021)
- ⭐ Χαροκόπειο Πανεπιστήμιο,τμήμα Πληροφορικής Και Τηλεματικής

Μελέτιος Τσεσμελής (it219105)

Σπυρίδων Μουχλιανίτης(it21958)

Απόστολος Δημητρίου(it219138)

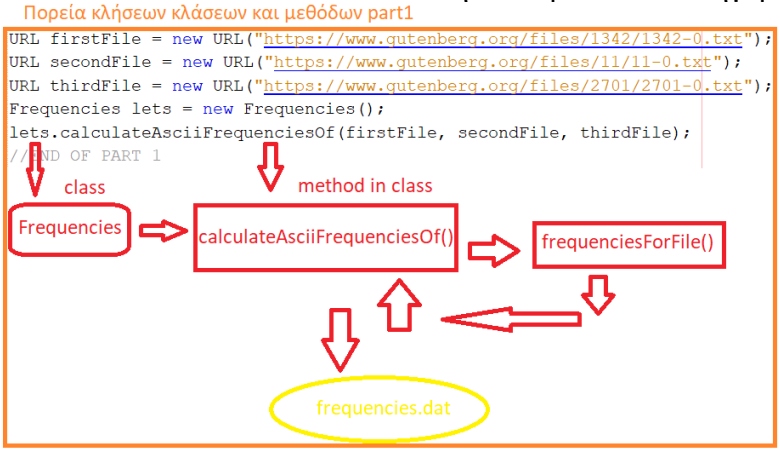
Ο σκοπός της αναφοράς αυτής είναι να αναφερθούμε και να σχολιάσουμε και στα πέντε μέρη της εργασίας μας! Αρχικά θα αναφερθούμε στις **9 κλάσεις** που δημιουργήσαμε καθ όλη την διάρκεια της.
Ονομαστικά έχουμε τις κλάσεις:

1. Frequencies, η οποία βοήθησε στην υλοποίηση και στην ωραιοποίηση του πρώτου ερωτήματος της εργασίας (υπολογισμός πίνακα συχνοτήτων).
2. MakeTree, HuffmanRules, Node, MinHeap, MinHeapInterface, οι οποίες βοήθησαν στην υλοποίηση και την ωραιοποίηση του δεύτερου ερωτήματος της εργασίας (υπολογισμός δέντρου Huffman).
3. Codification, η οποία βοήθησε στην υλοποίηση και ωραιοποίηση του τρίτου ερωτήματος της εργασίας (υπολογισμός κωδικοποίησης).
4. Encode, η οποία περιέχει μια main μέθοδο και πρακτικά χρησιμοποιεί όλες τις παραπάνω για την υλοποίηση του τέταρτου μέρους της εργασίας (Κωδικοποιητής).
5. Decode, η οποία περιέχει μια main μέθοδο και πρακτικά χρησιμοποιεί τις μεθόδους των πρώτων 3 μερών της εργασίας και υλοποιεί το πέμπτο μέρος της εργασίας (Αποκωδικοποιητής).

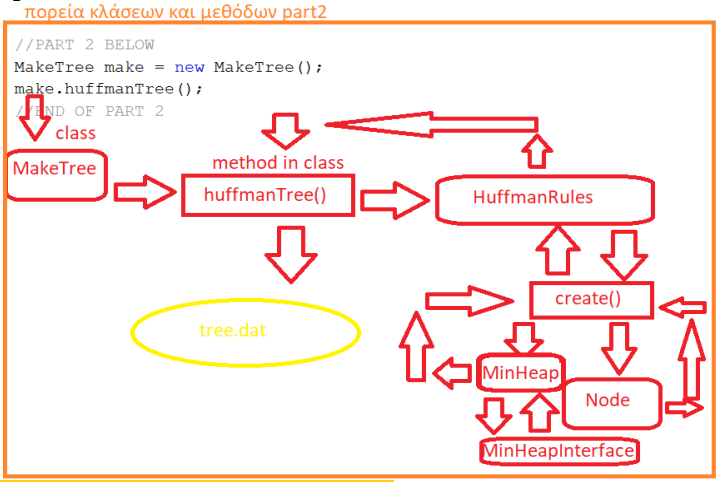
Πρακτικά ένα μικρό τμήμα κώδικα στην αρχή της Encode και της Decode είναι ίδιος με σκοπό να μπορεί ο χρήστης να εκτελέσει όποια από τις δύο θέλει πρώτα, και αν κάποιο αρχείο δεν υπάρχει το δημιουργεί αν όχι παραλείπει την διαδικασία των πρώτων 3 κατηγοριών άρα την δημιουργία των αρχείων frequencies.dat , tree.dat και codes.dat που είχαμε δημιουργήσει νωρίτερα!

Σε αυτό το σημείο πριν προχωρήσουμε στην ανάλυση της κάθε κατηγορίας, θεωρήσαμε σωστό για καλύτερη κατανόηση να φτιάξουμε μερικά σχήματα προκειμένου να δούμε το πως γίνεται η διαδικασία σε κάθε κλάση των πρώτων 3 κατηγοριών-part:

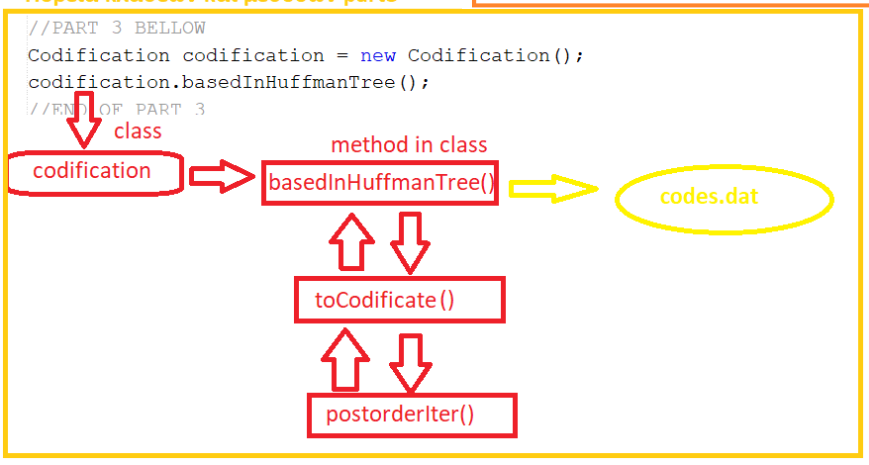
Πορεία κλήσεων κλάσεων και μεθόδων part1



πορεία κλάσεων και μεθόδων part2



Πορεία κλάσεων και μεθόδων part3



Τώρα που έχουμε μια γενική ιδέα για το τι θα ακολουθήσει, πάμε να τα δούμε αναλυτικότερα!

Όσον αφορά την πρώτη κατηγορία και επομένως το **πρώτο μέρος της εργασίας**,η **κλάση Frequencies(1)** όπως βλέπουμε και στο σχήμα περιέχει μια μέθοδο calculateAsciiFrequenciesOf() η οποία δέχεται ως ορίσματα τα 3 URL των files που μας ζητήθηκε να βρούμε. Θα χωρίσουμε την ανάλυση της σε 4 υποκατηγορίες:

1) **Αρχικά,** τον τρόπο που διαβάσαμε τα 3 αρχεία που μας ζητήθηκαν. Αυτό έγινε με την χρήση URL (τα οποία τα πήραμε ως ορίσματα της μεθόδου) όπως συμβουλευτήκαμε από τα βοηθητικά tutorial της oracle και όχι με την εντολή FILE με το σκεπτικό ότι όλα θα παράγονται αυτόματα και δεν θα χρειαστεί να δημιουργήσουμε "σύγχυση" για τα ονόματα των αρχείων και να χρειαστεί να προσαρμοστούν. Αλλά να υπάρξει αυτοματοποίηση.
2) **Δεύτερον,** την δημιουργία ενός πίνακα συχνοτήτων για τα πρώτα 128 στοιχεία του ASCII. Αυτό έγινε με την δημιουργία του 128-θέσεων πίνακα "frequencies".
   - Για τον οποίο καταλήξαμε να δημιουργήσουμε όπως βλέπουμε και στο σχήμα μια μέθοδο frequenciesForFile, στην οποία δίνουμε ένα ένα τα files και τον πίνακα με τα frequencies και παράλληλα σκανάραμε τα αντίστοιχα files ανά γράμμα (και όχι ανά γραμμή καθώς αντιμετωπίσαμε προβλήματα στην ανάγνωση των ειδικών χαρακτήρων) και σε κάθε χαρακτήρα που εμφανιζόταν αθροιστικά και για τα 3 files το τοποθετούσαμε στο αντίστοιχο κελί .
   - Το σκεπτικό πάνω σε αυτό είναι ότι καθώς υπάρχει επόμενο γράμμα, να το εκχωρούμε σε μια μεταβλητή ως Integer για να καταφέρουμε την τροποποίηση του σε αριθμό του ASCII και να ελέγξουμε αν είναι εντός ορίων [0-127] και τότε να αυξάνουμε τον μετρητή του στο αντίστοιχο κελί του πίνακα, και αυτό να γίνεται και για τα 3 files καλώντας την frequenciesForFile με όρισμα και επιστροφή του ίδιου πίνακα για να μπορεί να χρησιμοποιηθεί και σε επόμενα files!
3) **Τρίτον**, την έξοδο όλων αυτών σε αρχείο με την ονομασία frequencies.dat. Σχετικά με αυτό δημιουργήσαμε ένα νέο αρχείο με το όνομα "frequencies.dat" όπως μας ζητήθηκε και έπειτα χρησιμοποιήσαμε τον bufferWriter ώστε να γράψουμε την πληροφορία του κάθε κελιού του πίνακα frequencies, δηλαδή την συχνότητα των 128 πρώτων χαρακτήρων του πίνακα ASCII.
4) Όλα τα παραπάνω θα γίνουν εφόσον δεν υπάρχει ήδη το frequencies.dat!, αν υπάρχει τότε εκτυπώνουμε ένα κατάλληλο μήνυμα και προχωράμε!

Η έξοδος που βγάζει το frequencies.dat είναι:

```
0 -> 0
1 -> 0
2 -> 0
3 -> 0
4 -> 0
5 -> 0
6 -> 0
7 -> 0
8 -> 0
9 -> 0
10 -> 40701
11 -> 0
12 -> 0
13 -> 40701
14 -> 0
15 -> 0
16 -> 0
17 -> 0
18 -> 0
19 -> 0
20 -> 0
21 -> 0
22 -> 0
23 -> 0
24 -> 0
25 -> 0
26 -> 0
27 -> 0
28 -> 0
29 -> 0
30 -> 0
31 -> 0
32 -> 409147
33 -> 2719
34 -> 22
35 -> 4
36 -> 8
37 -> 3
38 -> 2
39 -> 11
40 -> 347
41 -> 347
42 -> 217
43 -> 0
44 -> 31243
45 -> 3279
46 -> 15811
47 -> 63
48 -> 239
49 -> 437
50 -> 173
51 -> 151
52 -> 127
53 -> 149
54 -> 101
55 -> 115
56 -> 127
57 -> 98
58 -> 626
59 -> 5914
60 -> 0
61 -> 0
62 -> 0
63 -> 1674
64 -> 5
65 -> 3609
66 -> 2621
67 -> 2098
68 -> 1450
69 -> 1876
70 -> 1106
71 -> 1082
72 -> 2231
73 -> 6697
74 -> 600
75 -> 308
76 -> 1597
77 -> 2581
78 -> 1273
79 -> 1000
80 -> 1697
81 -> 409
82 -> 1029
83 -> 2744
84 -> 3749
85 -> 294
86 -> 189
87 -> 2149
88 -> 28
89 -> 735
90 -> 23
91 -> 10
92 -> 0
93 -> 10
94 -> 0
95 -> 2144
96 -> 0
97 -> 128252
98 -> 25722
99 -> 38393
100 -> 65761
101 -> 204231
102 -> 34929
103 -> 33606
104 -> 104115
105 -> 107638
106 -> 1781
107 -> 12557
108 -> 69078
109 -> 38706
110 -> 112331
111 -> 120723
112 -> 26916
113 -> 2033
114 -> 92778
115 -> 103575
116 -> 146680
117 -> 46427
118 -> 15360
119 -> 35949
120 -> 2085
121 -> 32138
122 -> 1633
123 -> 0
124 -> 0
125 -> 0
126 -> 0
127 -> 0
```

Να σημειωθεί ότι στο αρχείο όλα βρίσκονται το ένα κάτω από το άλλο για λόγους ευκολίας τραβήξαμε να αντίστοιχα screenshots και τα δομήσαμε έτσι ώστε να είναι πιο ευανάγνωστα!

Όλα αυτά ισχύουν και στην Encode και στην Decode που περιέχουν τις αντίστοιχες main!

Στην συνέχεια λοιπόν θα αναλύσουμε **το δεύτερο μέρος της εργασίας** που είναι ο υπολογισμός του δέντρου Huffman.

Προτού όμως γίνει αυτό ας εξηγήσουμε την λογική του Huffman. Για κάθε χαρακτήρα από τους 128 πρώτους του ASCII σύμφωνα με τον Huffman θα πρέπει :

- να φτιάξουμε ένα δέντρο ενός κόμβου (χωρίς παιδιά και πατέρα) και
- να βρίσκουμε τα μικρότερα βάρη αυτών (με την μικρότερη συχνότητα)
- να αθροίζουμε τις συχνότητες τους και να δημιουργούμε ένα δυαδικό δέντρο με αριστερό και δεξί παιδί αντίστοιχα αυτά τα ελάχιστα

Αυτή η διαδικασία θα γίνεται συνέχεια μέχρις ότου να έχουμε μόνο ένα δέντρο, το οποίο θα είναι ένα βέλτιστο δέντρο που θα ακολουθεί την λογική του Huffman.

Αφού είδαμε την λογική του Huffman, δημιουργήσαμε μια **κλάση "Node"(2)** στην οποία κάναμε implements το Serializable το οποίο θα μας βοηθήσει για την έξοδο του δέντρου στο αρχείο **"tree.dat"** που θα δημιουργήσουμε αφού ολοκληρώσουμε την διαδικασία. Πριν από αυτό, ορίσαμε τα χαρακτηριστικά ενός κόμβου, δηλαδή την συχνότητα (frequency), τον χαρακτήρα-το στοιχείο του πίνακα που έχει την συγκεκριμένη συχνότητα (character), και την δυνατότητα να έχει 2 παιδιά (leftChild-rightChild). Ορίσαμε έναν constructor οπού κατά την αρχικοποίηση ουσιαστικά θα δημιουργεί φύλλα δέντρου (δεν έχει παιδιά) για να πετύχουμε την δημιουργία δέντρου ενός κόμβου για κάθε στοιχείο.

Σε αυτό το σημείο, αφού "δημιουργήσαμε" (θα ασχοληθούμε παρακάτω σχολιάζοντας την κλάση που περιέχει την αντίστοιχη main), θέλουμε κάπως να εκτελέσουμε την δεύτερη κουκίδα και να υπολογίζουμε τα 2 μικρότερα βάρη κάθε φορά. Σε αυτό βοήθησε η **κλάση "MinHeap"(3)** που δημιουργήσαμε στηριζόμενοι στην ArrayMinHeap με το δυαδικό σωρό που είχαμε δει στο εργαστήριο, προσαρμόζοντας την στις υπάρχουσες ανάγκες του προγράμματος μας (για τον σκοπό αυτό κρίναμε πως είναι καλή τεχνική να προσθέσουμε και ένα **interface (4)** για την MinHeap – επομένως κάναμε και Implements το MinHeapInterface στην MinHeap).

Ας σχολιάσουμε πάνω σε αυτό. Φτιάξαμε έναν πίνακα "array, και το size για αυτόν τον πίνακα. Χρησιμοποιήσαμε λοιπόν:

- Έναν απλό constructor MinHeap() για την αρχικοποίηση του πίνακα.
  - Επιλέξαμε να ξεκινάμε την διαδικασία από την θέση 0 του πίνακα αξιοποιώντας τον όλο.
- Έναν constructor MinHeap(Node[] userArray) για να πετύχουμε ουσιαστικά και την διαδικασία της heapify.
  - Αφού αρχικοποιήσαμε τον πίνακα
  - Αντιγράψαμε τα στοιχεία του userArray στον πίνακα array της κλάσης μας (χρησιμοποιήσαμε την arraycopy που μας πρότεινε το NetBeans)
  - χρησιμοποιώ την fixdown με σκοπό να μη χρειάζεται να χάνω σε χρόνο κάνοντας διαρκώς insert αλλά καλώντας πολλές φορές τον constructor να εκτελώ την ίδια διαδικασία.
- Την isEmpty() για να ελέγχουμε αν το size το πίνακα είναι 0 (την χρησιμοποιούμε στην findMin)
- την size() για επιστροφή του size του πίνακα
- την clear() για να διαγράψουμε ουσιαστικά τον πίνακα βάζοντας του null σε όλα τα κελιά και βάζοντας το size=0.
- Την findMin() για να βρούμε το ελάχιστο σε βάρος κόμβο (ουσιαστικά το 1ο στοιχείο του πίνακα αφού η minHeap το τοποθετεί στο πρώτο κελί)
- Την fixdown(int k) προκειμένου να ελέγχουμε ποιο είναι το μικρότερο παιδί (εφόσον έχει) και αν είναι μεγαλύτερο από τον πατέρα, τότε θα τα κάνουμε swap.
- την deleteMin() για να διαγράφουμε το πρώτο στοιχείο του πίνακα (το μικρότερο στοιχείο δηλαδή).
  - Αρά παίρνουμε το αριστερά και τέρμα δεξιά στοιχείο (το τελευταίο) και το γράφουμε στην πρώτη θέση
  - βάζουμε null στο τελευταίο ώστε να το διαγράψουμε

- και κάνουμε fixdown στην πρώτη θέση για να φτιάξουμε πάλι τα στοιχεία

- την fixup(int size) προκειμένου να ελέγχουμε αν η τιμή στον κόμβο που βρίσκομαι είναι μικρότερη από του πατέρα τότε κάνω swap
- την doubleCapacity() η οποία είναι βοηθητική καθώς παίρνει έναν πίνακα, φτιάχνει έναν διπλάσιο πίνακα και αντιγράφει τον πρώτο στο δεύτερο, και τον "παλιό" τον αφήνει να τον μαζέψει ο garbage collector της Java.
- την insert(Node element) για να εισάγουμε νέα elements
  - αν δεν υπάρχει χώρος καλεί την doubleCapacity()
  - γράφει το νέο στοιχείο στο τέλος του πίνακα
  - και καλεί την fixup(στην τελευταία θέση του πίνακα όπου εισάγαμε το στοιχείο)
- .την swap(), βοηθητική συνάρτηση για να κάνω ανταλλαγή στοιχείων στις διαδικασίες που αναφέραμε παραπάνω.

Μετά από αυτήν την εκτενή παρένθεση για να εξηγήσουμε όσα θεωρούσαμε απαραίτητα, επανερχόμαστε στο πως θα υλοποιήσουμε την διαδικασία του Huffman που είχαμε αναφέρει προηγουμένων, πώς δηλαδή θα φτιάξουμε ένα δέντρο το οποίο να ακολουθεί τους "κανόνες" του Huffman. Όπως μας ζητήθηκε για αυτόν τον σκοπό δημιουργήσαμε μια άλλη **κλάση "HuffmanRules"(5)** την οποία καλούμε στην μέθοδο huffmanTree() της **κλάση "MakeTree" (6)** (δείτε το πρώτο σχήμα σελ 1 της αναφοράς!).

Αρχικά στην HuffmanRules κάναμε implements το Serializable για να μας βοηθήσει στην έξοδο αργότερα.

Εδώ κληθήκαμε να ρωτήσουμε τους εαυτούς μας πως θα καταφέρουμε να εισάγουμε τον πίνακα των συχνοτήτων (ο οποίος είναι στην κλάση App που θα σχολιάσουμε παρακάτω) σε αυτή την κλάση και να συνεχίσουμε την διαδικασία. Για τον σκοπό αυτό σκεφτήκαμε και υλοποιήσαμε έναν constructor ο οποίος θα δέχεται έναν πίνακα από Node τα οποία θα έχουν ήδη την πληροφορία (θα μιλήσουμε παρακάτω στην main γιαυτό) και θα αντιγράφει την πληροφορία αυτή σε έναν άλλο πίνακα από Node, ο οποίος θα υπάρχει στην κλάση HuffmanRules.

Λύσαμε, λοιπόν το πρόβλημα για το πώς θα λάβουμε την πληροφορία που θέλαμε των frequencies. Σε αυτή την φάση έπρεπε να βρούμε έναν τρόπο ώστε να μπορέσουμε να δημιουργήσουμε το δέντρο huffman και να επιστρέψουμε πίσω στην main κάποια πληροφορία. Δημιουργήσαμε μια μέθοδο "Create()" η οποία επιστρέφει πληροφορία τύπου Node.

Μέσα στην μέθοδο αυτή:

- δημιουργήσαμε έναν πίνακα με τους κόμβους κάθε στοιχείου
- στα οποία βάλαμε το χαρακτήρα και την συχνότητα κάθε στοιχείου των 128 πρώτων του πίνακα ASCII κάνοντας το insert στην minHeap
- καλέσαμε την minHeap λοιπόν, για να καταφέρουμε να δομήσουμε το δέντρο με το κόμβο που περιέχει κάθε φορά την μικρότερη συχνότητα, στην κορυφή – στο πρώτο στοιχείο του πίνακα.
- Προκειμένου να εκτελέσουμε τα παρακάτω επαναληπτικά, χρησιμοποιήσαμε μια while για όσο θα υπάρχουν παραπάνω από 1 δέντρο. Η λογική δηλαδή είναι αφαιρούμε 2 και επιστρέφουμε 1 κάθε φορά, μέχρι να μην έχουμε να πάρουμε 2 roots καθώς θα υπάρχει μόνο ένα δέντρο.
  - ➔ Δεδομένου ότι θα έχουμε κάθε φορά το minimum στην κορυφή χρησιμοποιήσαμε την deleteMin() και 2 μεταβλητές προκειμένου να αποθηκεύουμε κάθε φορά τα 2 μικρότερα.
  - ➔ Αφού βρήκαμε τα 2 μικρότερα , προσθέσαμε τις συχνότητες τους και το αποτέλεσμα το αποθηκεύσαμε σε μια μεταβλητή
  - ➔ δημιουργήσαμε νέο Node στο οποίο:
    - ✔ θέσαμε ως frequency την μεταβλητή που είχαμε το άθροισμα
    - ✔ θέσαμε αριστερό παιδί και δεξιό παιδί, την μεταβλητή αντίστοιχα που είχαμε αποθηκεύσει τα 2 ελάχιστα
    - ✔ θέσαμε ως χαρακτήρα το '-' για να δηλώσουμε την απουσία χαρακτήρα, ότι εκεί δεν υπάρχει αυτή η πληροφορία
  - ➔ και κάναμε insert αυτού του κόμβου στο δέντρο

- Αφού καταλήξουμε σε ένα δέντρο , σκεφτόμαστε ότι τώρα έχουμε έναν κόμβο ο οποίος δείχνει σε 2 κόμβους και ο καθένας του σε άλλους 2 ή σε null αν είναι φύλλο κ.ο.κ.
  - ➔ Με αυτή την λογική, σκεφτήκαμε πως αρκεί να επιστρέψω τον πρώτο Node , την κορυφή του δέντρου και οι δείκτες θα αναλάβουν την δουλεία.
- Έτσι, γυρνάμε από την μέθοδο μας αυτό το Node, την ρίζα του δέντρου.

Χρησιμοποιούμε αυτή την ρίζα στην κλάση **MakesTree** για να δημιουργήσουμε το δέντρο. Ας σχολιάσουμε αυτή την κλάση πρώτα:

Για ευκολία θα αναφερθούμε σε 3 μέρη:

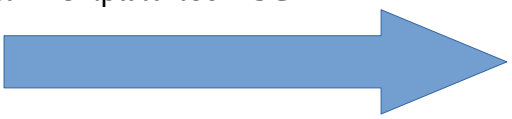
1. <u>Εισαγωγή και διάβασμα του αρχείου frequencies.dat.</u> Σχετικά με αυτό,
   - εφόσον το αρχείο αυτό το χούμε ήδη δημιουργήσει παραπάνω και θα ναι στον ίδιο φάκελο βάλαμε το όνομα του αρχείου για να μην υπάρξουν θέματα κατά την εκτέλεση από άλλους χρήστες, στην προκειμένη να μην χρειαστούν αλλαγές από εσάς.
   - Χρησιμοποιήσαμε την Scanner για να διαβάσουμε το αρχείο.
   - Για την μεταφορά της πληροφορίας, χρησιμοποιήσαμε μια επανάληψη για να διαβάσουμε την πληροφορία με την βοήθεια της reader.hasNext(), όσο έχει λοιπόν κάτι να διαβάσει να κάνει κάποιες ενέργειες
     - διαβάζαμε γραμμή γραμμή το αρχείο
     - και χρησιμοποιώντας την reader.Next() προχωρήσαμε αποθηκεύσαμε την πληροφορία σε μια μεταβλητή data.
     - επειδή σε κάθε γραμμή η πληροφορία στο αρχείο είναι με την δομή “ $2 \rightarrow 4567$” αποθηκεύσαμε μόνο την τρίτη πληροφορία (που αποτελεί την συχνότητα του κάθε στοιχείου των 128 πρώτων του ASCII) αγνοώντας το character και το βελάκι.

2. <u>Δημιουργία του Huffman Tree</u> Σχετικά με αυτό:
   - εφόσον έχουμε έτοιμες όλες τις άλλες κλάσεις, χρησιμοποιήσαμε όπως κάθε φορά που θέλουμε να καλέσουμε μια συνάρτηση από άλλη κλάση όπως έχουμε διαπιστώσει, ένα object για την κλάση Huffman και χρησιμοποιήσαμε την μέθοδο της HuffmanCodıfication δίνοντας τον πίνακα των συχνοτήτων που αναφέραμε παραπάνω.
   - Αφού εκτελέσθηκε η μέθοδος αυτή (όπως αναφέραμε παραπάνω όταν εξηγήσαμε την λειτουργία της) επέστρεψε ένα Node την ρίζα του δέντρου, το οποίο αποθηκεύσαμε σε μια μεταβλητή Node.

3. <u>Έξοδος του δέντρου σε αρχείο</u>. Σχετικά με αυτόματα¨
   - Για την μεταφορά του δέντρου σε ένα αρχείο με το όνομα “tree.dat”, όπως προτάθηκε από εσάς, αφού μελετήσαμε τα αντίστοιχα tutorial της oracle , χρησιμοποιήσαμε ObjectOutputStream
   - προϋπόθεση αυτών να κάνουμε implements το Serializable στις αντίστοιχες κλάσεις όπως αναφέραμε και πιο πάνω.

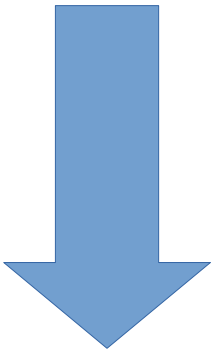
Σε αυτό το σημείο προσπαθώντας να επαληθεύσουμε σε όσο τον δυνατόν περισσότερο βαθμό ότι όλα πήγαν καλά χρησιμοποιήσαμε μια διάσχιση levelOrder και λάβαμε τα παρακάτω αποτελέσματα, τα οποία φαίνονται λογικά και μας διατηρούν αισιόδοξους για την πορεία αυτή!

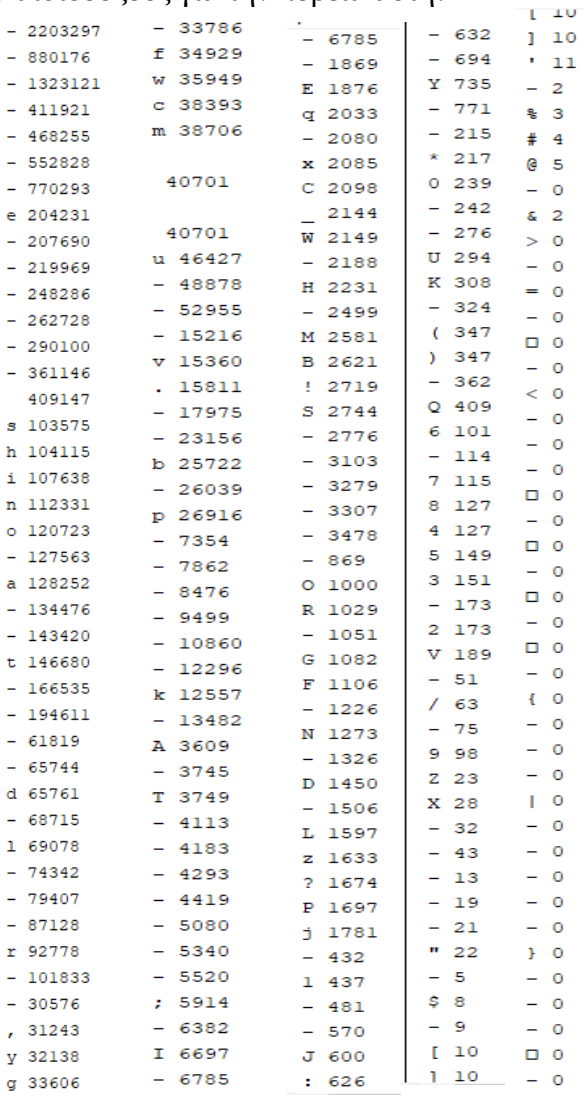
Δίπλα ακολουθεί το αποτέλεσμα αυτής της διαδικασίας "επαλήθευσης" με screenshots, όπως προαναφέραμε η παύλα (-) συμβολίζει τους "πατέρες" τους κόμβους που προστέθηκαν για την δημιουργία του δέντρου και δεν υπήρχαν στα 128 πρώτα του ASCII

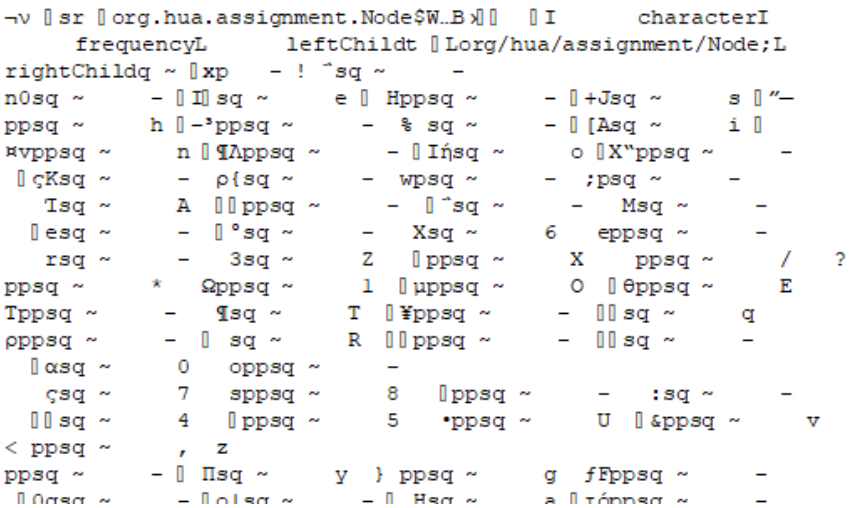


και παρακάτω ακολουθεί ένα screenshot για δείγμα σχετικά με την δημιουργία και το αποτέλεσμα που περιέχει το αρχείο tree.dat που δημιουργήσαμε !

Να σημειωθεί ότι όλα αυτά ισχύουν και στις 2 κλάσεις με τις main (Encode/Decode)!







Τώρα που έχουμε το δέντρο Huffman, πάμε να αναλύσουμε το **τρίτο μέρος της εργασίας,**τον υπολογισμό δηλαδή της κωδικοποίησης.

Όπως παρατηρούμε και στο σχήμα της σελίδας 1 της αναφοράς δημιουργήσαμε μια **κλάση codification (7)** και μέσα σε αυτήν μια μέθοδο basedInHuffmanTree την οποία καλούμε στην αντίστοιχη main για να εκτελέσουμε την παρακάτω διαδικασία.

Στην κλάση αυτή , σκεπτόμενοι το πως θα καταφέρουμε να περάσουμε το root του δέντρου και ταυτόχρονα να περάσουμε το PrintWriter για να το χρησιμοποιήσουμε στην έξοδο μας, αποφασίσαμε να δημιουργήσουμε μια μέθοδο toCodificate την οποία θα την καλούμε στην basedInHuffmanTree μέθοδο που θα σχολιάσουμε σε λίγο και θα γίνεται μια κλήση αυτής για να πραγματοποιηθεί η όλη διαδικασία.

- Συγκεκριμένα στην toCodificate(), χρησιμοποιήσαμε την Deque<String> προκειμένου να έχουμε μια στοίβα όπου θα αποθηκεύουμε κατά την διάρκεια της διάσχισης τους χαρακτήρες “1” ή “0” (θεωρήσαμε όπως αναφέρετε ότι για την κωδικοποίηση λαμβάνουμε τα 1 και 0 ως χαρακτήρες) με τέτοιο τρόπο ώστε κάθε φόρα που κάνουμε αριστερά (προς αριστερό παιδί) θα μπαίνει στην στοίβα το “0” ενώ όταν γίνεται δεξιά κίνηση (προς δεξί παιδί) θα μπαίνει “1” στην στοίβα.
- Δημιουργήσαμε έναν Iterator, τον οποίο θα τον χρησιμοποιήσουμε για να βλέπουμε αν υπάρχει στην στοίβα επόμενο στοιχείο (.hasNext) και θα πηγαίνουμε σε αυτό αν υπάρχει (.next). Με λίγα λόγια για να επιστρέφουμε τα περιεχόμενα της στοίβας.

Για να επιτύχουμε την κωδικοποίηση κληθήκαμε να επιλέξουμε σχετικά με ποια διάσχιση του δέντρου θα υλοποιήσουμε (π.χ inorder, levelorder, postorder, preorder). Αποφασίσαμε ότι η πιο κατάλληλη διάσχιση για το σκεπτικό μας είναι η postorder και αυτή υλοποιήσαμε.

- Συγκεκριμένα δημιουργήσαμε μια μέθοδο “postorderIter” στην οποία δώσαμε ως ορίσματα ένα Node (για να περάσουμε το root), ένα Deque<String> (για να περάσουμε την στοίβα) , ένα τύπου Iterator<String>(για να μπορούμε να επιστρέφουμε όλα τα στοιχεία της στοίβας), ένα τύπου PrintWriter (για να γράψουμε στο αρχείο) και ένα String για να καταφέρουμε την εισαγωγή του 1 ή 0 αντίστοιχα.
- Όσον αφορά την λειτουργία της μεθόδου, η διαδικασία γίνεται ως εξής:
  - Ελέγχουμε αν υπάρχει root επομένως υπάρχει το δέντρο και εφόσον υπάρχει
    - Αν η στοίβα έχει στοιχεία τότε εξάγω το πρώτο
    - Ελέγχουμε αν δεν έχει παιδιά, τότε χρησιμοποιώ τον iterator.hasNext για να δούμε αν υπάρχει επόμενο στοιχείο στον Iterator και αν υπάρχει τότε εκτυπώνουμε το περιεχόμενο της στοίβας μέσω αυτού και εξάγουμε το τελευταίο στοιχείο της στοίβας
    - και κάνουμε return μέσα στην συνθήκη (σκεπτόμενοι ότι δεν θα μπει την πρώτη φορά αφού έχει την root και έχει παιδιά) έτσι ώστε να μπορεί παρακάτω με τις αναδρομικές κλήσεις (καλώντας τον εαυτό της) να επιστρέφει από τον έλεγχο του αριστερού παιδιού στον πατέρα ώστε να μπει στο δεξί παιδί και πάλι βλέπει το return και συνεχίζει τον παρακάτω κώδικα

Επίσης ελέγχουμε αν η στοίβα είναι άδεια και αν δεν είναι τότε κάνουμε remove το τελευταίο στοιχείο προκειμένου να πάει πάλι στον πατέρα του πατέρα (παππού) για να συνεχίσει την διάσχιση

Μόλις τελειώσει αυτή η διαδικασία θα επιστρέψει **στην basedInHuffmanTree** μέθοδο στην οποία την καλέσαμε .Πιο αναλυτικά basedInHuffmanTree χωρίζουμε αυτή την μέθοδο σε 2:

1. <u>Εισαγωγή και διάβασμα του αρχείου “tree.dat”:</u>
   - για το σκοπό αυτό χρησιμοποιήσαμε για να διαβάσουμε το αρχείο το FileInputStream, δημιουργήσαμε ένα αντικείμενο για να το προσπελάσουμε και να πάρουμε εξάγουμε τις πληροφορίες που έχει.
     - Σε αυτό το σημείο για να πάρουμε το root (το οποίο θέλαμε να είναι τύπου Node) κάναμε cast (Node) το object που παίρναμε ως πληροφορία από το αρχείο.
2. <u>Δημιουργία της κωδικοποίησης και έξοδος σε αρχείο “code.dat”:</u>

- Δημιουργήσαμε ένα αρχείο με το όνομα codes.dat
- Δημιουργούμε έναν αντικείμενο PrintWriter (με σκοπό να χρησιμοποιήσουμε την printf όπως αναφέραμε) για να γράψουμε στο αρχείο code.dat.
- Δημιουργούμε πιο πάνω αντικείμενο για την κλάση δημιουργήσαμε και χρησιμοποιούμε την μέθοδο “toCondificate(Node n, PrintWriter pw)” η οποία όπως βλέπουμε παίρνει ως ορίσματα ένα Node (εκεί θα δώσουμε το root) και έναν PrintWriter (με σκοπό να γράφουμε απευθείας τα αποτελέσματα όταν γίνεται η όλη διαδικασία) .
  - Για την υλοποίηση της καλεί την μέθοδο postorderIterator, προκειμένου να διασχίσει το δέντρο!
- Επειδή κατά την διαδικασία μέσα στην μέθοδο όπως αναλύσαμε παραπάνω χρησιμοποιούμε την printf() της PrintWriter για να γράψουμε στο αρχείο τα αποτελέσματα της διαδικασίας, και για να είμαστε σίγουροι πως η διαδικασία έχει τελειώσει , χρησιμοποιήσαμε την .close() στην μέθοδο basedInHuffmanTree, αφού λοιπόν έχει ολοκληρωθεί η διαδικασία.

Να σημειωθεί πως όλα τα παραπάνω σχετικά με το τρίτο μέρος γίνονται όπως και στα προηγούμενα part <u>μόνο αν δεν υπάρχει το αρχείο codes.dat, αν υπάρχει η διαδικασία παραλείπεται!!</u>

Δεξιά ακολουθούν ενδεικτικά screenshots με το αποτέλεσμα της παραπάνω διαδικασίας

Σε αυτό το σημείο μπαίνουμε στην ανάλυση του **τέταρτου μέρους της εργασίας**.

Σε αυτό το part κληθήκαμε να υλοποιήσουμε την κωδικοποίηση Huffman σε επίπεδο bits, χρησιμοποιώντας 2 αρχεία που θα δίνει ο χρήστης (ένα για την είσοδο ενός αρχείου με ASCII χαρακτήρες και ένα για την έξοδο της κωδικοποίησης Huffman (χρησιμοποιώντας αυτών των ASCII χαρακτήρων) και το αρχείο codes.dat που δημιουργήσαμε όπως προείπαμε για να αποθηκεύσουμε την κωδικοποίηση κάθε γράμματος των 128 πρώτων του ASCII.
Θα χωρίσουμε αυτή την διαδικασία σε **3 μέρη.**

< Η παρακάτω διαδικασία γίνεται στην main της κλάσης Encode >

1.Εισαγωγή- έλεγχος και διάβασμα αρχείων.

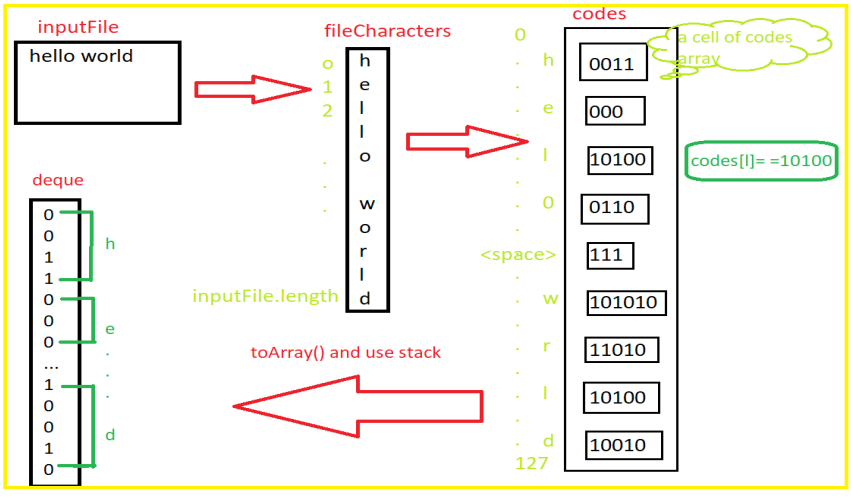
Αρχικά, **εισάγουμε** και **ελέγχουμε** αν ο χρήστης έβαλε τα 2 αρχεία ως arguments (εδώ προσοχή πως πρέπει να είναι ακριβώς 2 αυτά που δίνει και το αρχείο εισόδου να υπάρχει ήδη). Σε αντίθετη περίπτωση βγάζουμε κατάλληλα μηνύματα και το πρόγραμμα σταματάει. Σε περίπτωση που υπάρχει (Δεν είναι αναγκαστικό! Το δημιουργούμε) ήδη το αρχείο που θέλει ο χρήστης να κάνει την έξοδο τότε γίνεται αντικατάσταση αυτού.

Έπειτα **διαβάζουμε** το αρχείο **codes.dat** για να πάρουμε την κωδικοποίηση κάθε γράμματος. Αποφασίσαμε να ταξινομήσουμε το περιεχόμενο του codes.dat και να λάβουμε την πληροφορία της μορφής " h 0011" σε 2 πίνακες.

Έτσι χρησιμοποιήσαμε τον characters[] για να αποθηκεύσουμε το γράμμα (την πρώτη πληροφορία) και τον codes[] για να λάβουμε την κωδικοποίηση του αντίστοιχου γράμματος. Σκοπός αυτού ήταν να κάνουμε μια ταξινόμηση των δυο πινάκων κατά τον ίδιο τρόπο ώστε το στοιχείο character[0] να χει την το γράμμα 0 του ASCII και codes[0] να χει την κωδικοποίηση του γράμματος, άρα θα υπάρχει συγχρονισμός των index των πινάκων και επομένως θεωρήσαμε ότι έτσι είναι πιο προσιτή και εύχρηστη η πληροφορία αφού το codes[h] θα περιέχει την κωδικοποίηση του h για παράδειγμα και αυτό μας έδωσε βοήθησε παρακάτω όπως θα δούμε.

Στην συνέχεια **διαβάζουμε** αυτό το αρχείο. Το διάβασμα του γίνεται ανά χαρακτήρα με σκοπό να ελέγχουμε το κάθε χαρακτήρα και να παράγουμε την κωδικοποίηση του.

Για ευκολία χρησιμοποιούμε έναν πίνακα για να αποθηκεύουμε το κάθε γράμμα αυτού του αρχείου με σκοπό να χρησιμοποιήσουμε την ιδιότητα που είπαμε παραπάνω (ότι codes[h]=0011, άρα πιο γενικά για h= fileCharacters[i] θα πάρουμε την αντίστοιχη κωδικοποίηση). Μια προσπάθεια απεικόνισης όλων αυτών, βασισμένη στα αποτελέσματα μας είναι:



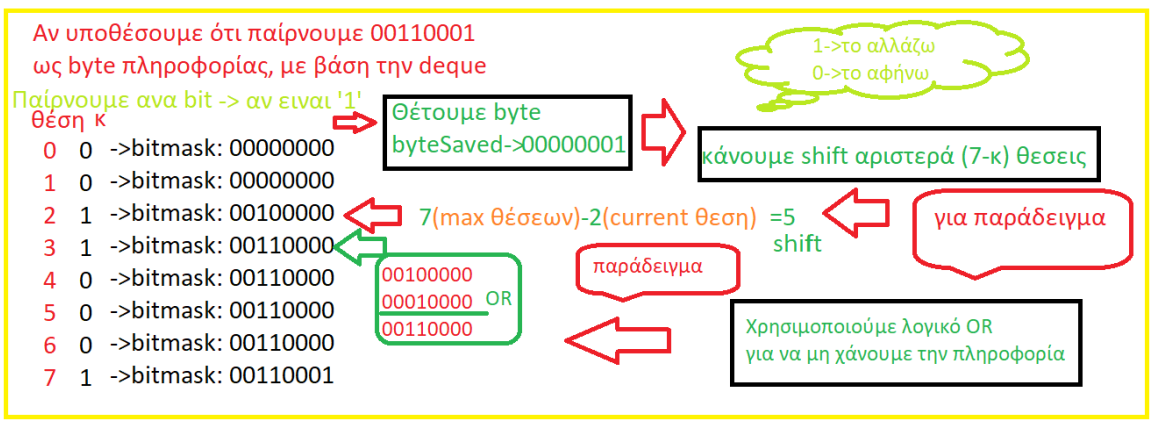
2.Υπολογισμός κωδικοποίησης σε επίπεδο bits

Εφόσον έχουμε, όπως παρατηρούμε και στην εικόνα, την κωδικοποίηση όλων των χαρακτήρων εισόδου που έδωσε ο χρήστης σε μια στοίβα (χρησιμοποιούμε την deque (τύπου Deque) που μας παρέχει η java) τώρα πρέπει να πάρουμε αυτή την πληροφορία και να την μετατρέψουμε σε bits και bytes ανάλογα για να τα περάσουμε στο αρχείο εξόδου.

Για τον σκοπό αυτό παραδειγματιζόμενοι από το tutorial που μας δόθηκε αλλά και από σημειώσεις που ψάξαμε στο διαδίκτυο ή παλαιότερων μαθημάτων (π.χ ένα από αυτά ήταν η "ψηφιακή τεχνολογία και οι εφαρμογές τηλεματικής" όπου είχαμε αναλύσει τις μάσκες και πως λειτουργούν και μπορέσαμε να αναπτύξουμε την παρακάτω λογική).

Συγκεκριμένα, το σκεπτικό μας είναι

- Παίρνουμε ένα ένα τα στοιχεία από την deque (χρησιμοποιώντας έναν Iterator)
- Ελέγχουμε αν η deque κατά την διάρκεια των 8 loops κάθε φόρα στην επανάληψη όπως θα δούμε παρακάτω, βγάζει όλα τα στοιχεία και είναι άδεια
  - αν είναι άδεια δεν εκτελούμε την παρακάτω και απλά προχωράμε
  - αν δεν είναι και έχει στοιχείο τότε εκτελούμε τα παρακάτω
    - Συγκεκριμένα ελέγχουμε αν αυτό που πήραμε είναι "1"
      - Αν όχι τότε δε προβαίνουμε σε αλλαγές
      - Αν ναι τότε
      - χρησιμοποιούμε την μάσκα (την οποία την ξεκινάμε με 0) και μια μεταβλητή savedByte για να αλλάζουμε την πληροφορία από char σε byte με την ολοκλήρωση της διαδικασίας
      - κάνουμε shift left (<<) τόσες θέσεις (7-κ αφού 8bits-(κ+1) επειδή ξεκινάμε από κ==0) ώστε να μπει στη θέση που θέλω το 1
      - κάνω λογικό OR μεταξύ της μάσκας και του savedByte, ώστε στην μάσκα να δημιουργώ το Byte που θα μεταφέρω στο file
      - συνεχίζουμε μέχρι να έχουμε 8 bits == 1 byte
      - εκτυπώνουμε στο αρχείο το byte
      - διαγράφουμε το πρώτο στοιχείο της στοίβας
    - μηδενίζουμε την μάσκα και τον μετρητή κ, και επαναλαμβάνουμε την ίδια διαδικασία μέχρι το τελευταίο bit (όσο η deque δεν είναι άδεια)



Στο **παραπάνω σχήμα** προσπαθήσαμε μέσω ενός παραδείγματος να αναλύσουμε και να εξηγήσουμε όσο πιο συνοπτικά το σκεπτικό μας για την παραπάνω διαδικασία, με απώτερο σκοπό την κατανόηση της λογικής αυτής.

3.Γράψιμο στο αρχείο εξόδου.

Για τον σκοπό αυτό δημιουργήσαμε (ή αντικαταστήσαμε το ήδη υπάρχον) αρχείο που έδωσε ο χρήστης με την βοήθεια του DataOutputStream (καθώς το ObjectOutputStream που είχαμε κάνει δεν έβγαζε σωστά αποτελέσματα για μεγάλες εισόδους για κάποιο περιέργο λόγο του Netbeans...) γράψαμε

- Στην αρχή του αρχείου **πόσα bits έχει το αρχείο**, ώστε στην αποκωδικοποίηση να γνωρίζουμε πόσα θα διαβάσουμε (πόσα δηλαδή είναι τα "χρήσιμα" bits που θα διαβάσουμε, και πόσα προστέθηκαν αυτόματα για να γίνει πολλαπλάσιο του 8) **αντιμετωπίζοντας** έτσι το πρόβλημα που ζητήθηκε, δηλαδή ότι ο αριθμός των bits του αρχείου δεν είναι υποχρεωτικά πολλαπλάσιο του 8.
- και έπειτα ακολουθώντας την επαναληπτική διαδικασία που αναφέραμε, γράψαμε byte – byte την πληροφορία, την κωδικοποίηση.

Να σημειωθεί πως στην κλάση Encode που περιέχει την main ελέγχουμε αν υπάρχουν τα αρχεία frequencies.dat, tree.dat, codes.dat και εκτελούμε τον αντίστοιχο κώδικα δημιουργίας του εφόσον δεν υπάρχει στο φάκελο! Σε περίπτωση δηλαδή που υπάρχουν και τα 2 ήδη, βγάζουμε ένα κατάλληλο μήνυμα σε αυτά τα δυο ότι παραλείπεται η διαδικασία δημιουργίας τους και εκτελούμε μόνο για αυτό που λείπει. Σκοπός είναι να κερδίσουμε σε ταχύτητα! Και να μη κάνουμε άσκοπες εκτελέσεις!

Σε αυτό το σημείο τελειώσαμε με την περιγραφή της διαδικασίας. Όμως για να είμαστε πιο αποτελεσματικοί προσπαθήσαμε να ελέγξουμε και την **ορθότητα** ορισμένων λειτουργιών. Όπως,

- ✔ της μεταφοράς των πληροφοριών. Έτσι εκτυπώσαμε πριν μεταφέρουμε κάθε πληροφορία στο αρχείο και έπειτα διαβάσαμε τις πληροφορίες από το αρχείο και πράγματι (ακολουθεί αποδεικτικό screenshot) καταλήξαμε στις ίδιες πληροφορίες.
- ✔ Του ορθού υπολογισμού των bytes. Σε αυτό το σημείο δεν θα μπορούσαμε να είμαστε πλήρως καλυμμένοι καθώς στο επόμενο part της εργασίας (αποκωδικοποίηση) θα φανεί πιο πρακτικά. Όμως προσπαθήσαμε να προσεγγίσουμε μέσω άλλων παραμέτρων όπως
  - ✔ τον έλεγχο της πορείας της μάσκας, του αθροιστή κ, της μεταβλητής byteSaved πριν και μετά την εκτέλεση τους και όλα φαίνονταν λογικά
  - ✔ το μέγεθος του inputFile και του outputFile, καθώς με την βοήθεια του ίντερνετ ανακαλύψαμε πως η κωδικοποίηση του Huffman χρησιμοποιείται για συμπίεση αρχείων επομένως το λογικό θα είναι το μέγεθος του outputFile να είναι μικρότερο από το μέγεθος του inputFile, και πράγματι έτσι βγήκε και σε μας !

Έτσι, διατηρούμε την **αισιοδοξία** μας για την επαλήθευση στην αποκωδικοποίηση!

Σημείο Προσοχής: Ελέγχουμε αν στο αρχείο εισόδου περιέχεται κάποιος χαρακτήρας εκτός των 128 πρώτων του ASCII και αν ναι βγάζουμε μήνυμα προειδοποίησης στον χρήστη ότι πιθανόν να μη λειτουργήσει σωστά!

Παρακάτω ακολουθούν μερικά screenshots για να υποστηρίξουμε τα όσα είπαμε!

1screenshot

```
Running...
File "frequencies.dat" already exists
File "codes.dat" already exists
Number of bits -> 49
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> 49
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> 74
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> 55
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> -87
```

2screenshot(συνέχεια 1ου)

```
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> -75
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> 73
perasa0
perasa1
perasa2
perasa3
perasa4
perasa5
perasa6
perasa7
bitmask print before write-> 0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  12.633 s
Finished at: 2021-01-10T00:16:27+02:00
------------------------------------------------------------------------
```

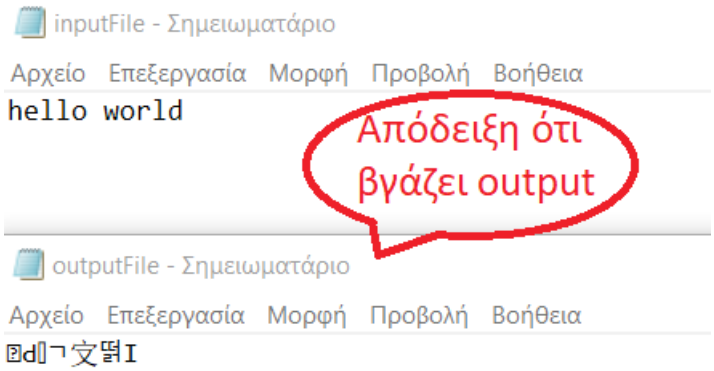
Απόδειξη διαβασματος ανα byte

3screenshot

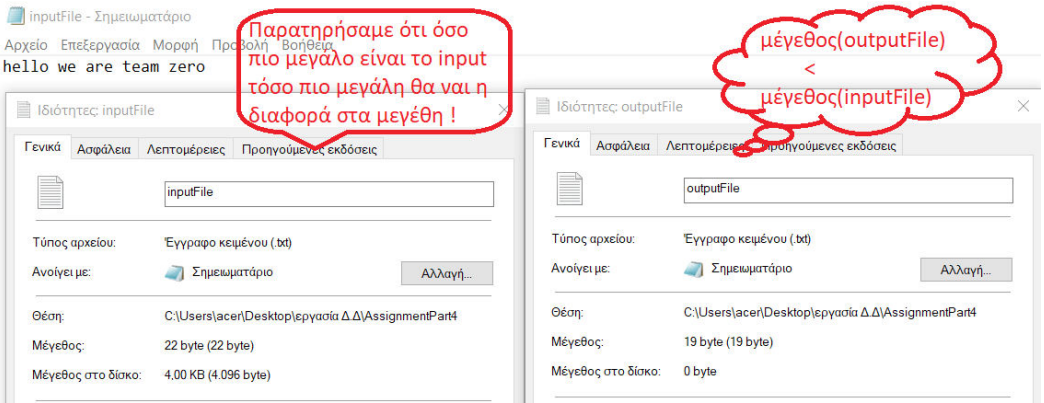
```
Running...
File "frequencies.dat" already exists
File "codes.dat" already exists
Number of bits -> 49
bitmask print before write-> 49
bitmask print before write-> 74
bitmask print before write-> 55
bitmask print before write-> -87
bitmask print before write-> -75
bitmask print before write-> 73
bitmask print before write-> 0


(read outputFile)
49
49
74
55
-87
-75
73
0
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
Total time:  11.860 s
Finished at: 2021-01-10T00:05:44+02:00
------------------------------------------------
```

ορθή μεταφορά δεδομένων

4screenshot



5screenshot



Τέλος, ήρθε η ώρα να σχολιάσουμε την διαδικασία του decode , δηλαδή του **πέμπτου ερωτήματος της εργασίας**.

< Η παρακάτω διαδικασία γίνεται στην main της κλάσης Decode >

Αρχικά, εισάγουμε και ελέγχουμε αν ο χρήστης έβαλε τα 2 αρχεία ως arguments (εδώ προσοχή πως πρέπει να είναι ακριβώς 2 αυτά που δίνει και το αρχείο εισόδου να υπάρχει ήδη). Σε αντίθετη περίπτωση βγάζουμε κατάλληλα μηνύματα και το πρόγραμμα σταματάει. Σε περίπτωση που υπάρχει (Δεν είναι αναγκαστικό! Το δημιουργούμε) ήδη το αρχείο που θέλει ο χρήστης να κάνει την έξοδο τότε γίνεται αντικατάσταση αυτού.

Να σημειωθεί πως και στην κλάση Decode που περιέχει την άλλη main ελέγχουμε αν υπάρχουν τα αρχεία frequencies.dat, tree.dat, codes.dat και εκτελούμε τον αντίστοιχο κώδικα δημιουργίας του εφόσον δεν υπάρχει ήδη στο project! Σε περίπτωση δηλαδή που υπάρχουν τα 2 ήδη, βγάζουμε ένα κατάλληλο μήνυμα σε αυτά τα δυο ότι παραλείπεται η διαδικασία δημιουργίας τους και εκτελούμε μόνο γιαυτό που λείπει -που δεν έχει δημιουργηθεί ακόμα. Σκοπός είναι να κερδίσουμε σε ταχύτητα-να ελαφρύνουμε τον κώδικα! Και να μη κάνουμε άσκοπες εκτελέσεις!

Εισάγουμε,λοιπόν,το αρχείο tree.dat - διαβάζουμε τις πληροφορίες από το αρχείο που θέλει ο χρήστης να αποκωδικοποιήσει. Το πρώτο byte είναι εκείνο που τοποθετήσαμε στο προηγούμενο τμήμα της εργασίας, δηλαδή το byte που χει τον αριθμό των χρήσιμων bit στο τελευταίο byte του αρχείου (δείτε σχετική ανάλυση στο 4ο θέμα).

Με την βοήθεια της .readAllBytes() που προσφέρεται στην InputStream καταφέραμε να διαβάσουμε όλα αυτά τα bytes σε έναν πίνακα τον οποίο ονομάσαμε arrayBytes. Το πρώτο κελί του πίνακα εφόσον η πρώτη πληροφορία- το πρώτο byte που δέχθηκε ήταν ο αριθμός των ωφέλιμων bit οπότε το αποθηκεύσαμε σε μια μεταβλητή.

Έπειτα φτιάξαμε έναν πίνακα με στοιχεία τύπου BitSets

Πιο αναλυτικά για τις παραπάνω ενέργειες, ακολουθήσαμε την λογική:

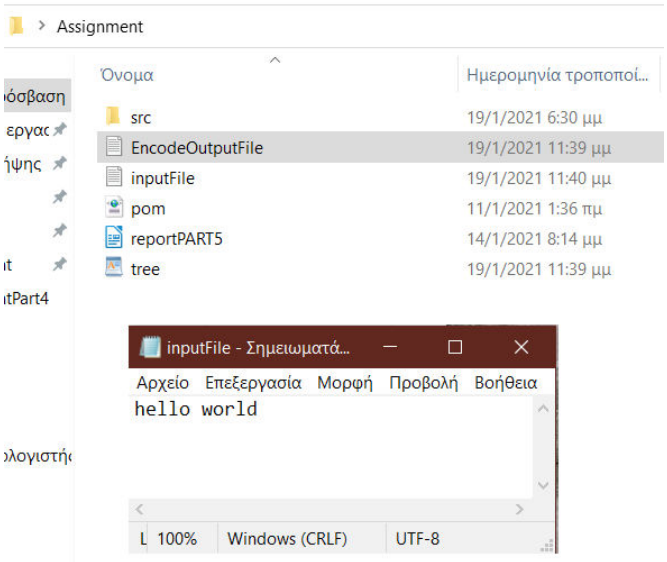
- Όσο είχαμε bytes για να διαβάσουμε
  - αποθηκεύαμε στο αντίστοιχο κελί της bitSet την πληροφορία για το σε ποιες θέσεις εμφανίζονται οι άσσοι δίνοντας του το αντίστοιχο κελί του πίνακα με τα bytes.
  - Για να αντιμετωπίσουμε λοιπόν το πρόβλημα που αναφέραμε παραπάνω σχετικά με την αγνόησή των μηδενικών και τον “καθρεπτισμό” εισάγαμε ένα for το οποίο θα ξεκινάει από το 7 (όπου 7η είναι η θέση που τελειώνει το κάθε byte αφού ξεκινά από το 0) μέχρι και το 0 έχοντας αρνητικό (μειούμενο κατά ένα) βήμα. Αυτό είχε ως σκοπό να εισαχθούν με την ανάποδη κατά μια έννοια σειρά όπως σχολιάσαμε και φέραμε και το παράδειγμα παραπάνω που αναφέραμε το πρόβλημα.
  - Σχετικά με το πρόβλημα του μεταβλητού μήκους byte που είχαμε αναφέρει και στο 4ο μέρος της εργασίας, έπρεπε να χωρίσουμε την διαδικασία σε 2 μέρη
    - αρχικά κάναμε ένα if για να δούμε αν δεν είμαστε στο τελευταίο byte
      - και ακολουθήσαμε την διάσχιση με τον εξής τρόπο:
        - ελέγξαμε αν το αντίστοιχο bit ήταν ένα (με την βοήθεια της get()) και αν αυτό ήταν αληθές πηγαίναμε στο δεξί παιδί του current κόμβου
        - αν ήταν ψευδές πηγαίναμε στο αριστερό παιδί του current κόμβου
        - τέλος, αν δεν είχε κανένα παιδί εκτυπώναμε απευθείας στο αρχείο με το όνομα που έδωσε ο χρήστης την πληροφορία αυτού του φύλλου
        - και βάζαμε το current κόμβο ξανά στην ρίζα του δέντρου
    - Διαφορετικά αν είμαστε στο τελευταίο byte η διαφοροποίηση εδώ ήταν ότι ελέγχαμε πρώτα
      - αν είσαι εντός των χρήσιμων bit (αγνόησε >=8-χρήσιμα)
        - τότε κάνε πάλι την διάσχιση με αυτόν τον τρόπο:
          - ελέγξαμε αν το αντίστοιχο bit ήταν ένα (με την βοήθεια της get()) και αν αυτό ήταν αληθές πηγαίναμε στο δεξί παιδί του current κόμβου
          - αν ήταν ψευδές πηγαίναμε στο αριστερό παιδί του current κόμβου
          - τέλος, αν δεν είχε κανένα παιδί εκτυπώναμε στο αρχείο πάλι την πληροφορία αυτού του φύλλου – δηλαδή το αντίστοιχο γράμμα που αντιστοιχεί στην κάθε κωδικοποίηση.
          - και βάζαμε το current κόμβο ξανά στην ρίζα του δέντρου

Ενδεικτικά screenshots αποτελεσμάτων

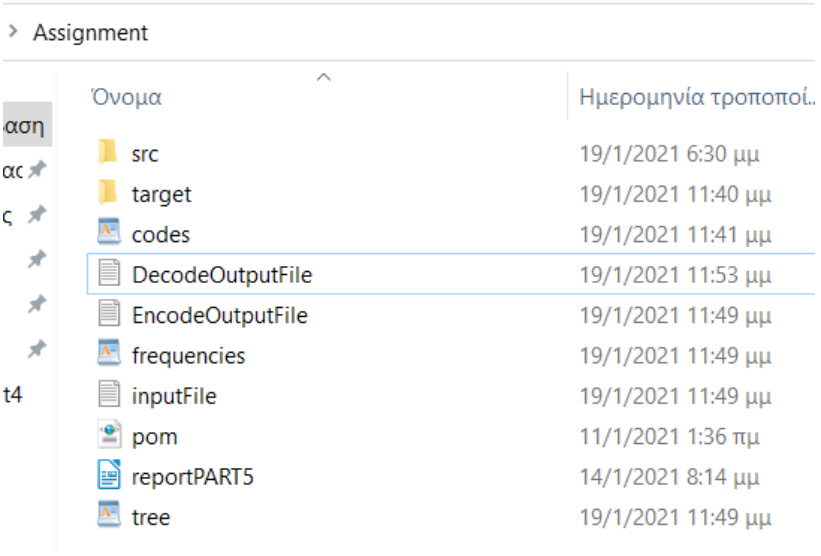
```
] --- exec-maven-plugin:1.5.0:exec (default-cli) @ Assignment ---
 Running...
        frequencies.dat already exist!So I skip creation process!
        tree.dat already exist!So I skip creation process!
        codes.dat already exist!So I skip creation process!
 numberOfUsefulBits is 5
 bytes in file is 3
 In 0 index arrayBytes is 49
 In 1 index arrayBytes is 74
 In 2 index arrayBytes is 48
 bitSets length is 3
 arrayBytes length is 3

 hello
 number of bits: 24
 ------------------------------------------------------------------------
 BUILD SUCCESS
 ------------------------------------------------------------------------
 Total time:  2.671 s
 Finished at: 2021-01-19T23:27:02+02:00
 ------------------------------------------------------------------------
```

Ορθή μεταφορά πληροφοριών από το αρχείο στον κώδικα.
Παράλληψη επανεδημιουργιας των ήδη υπάρχον files ->εξοικονόμηση χρόνου

Πριν την εκτέλεση



Μετά την εκτέλεση



Αποτελέσματα (η εκτύπωση του κειμένου γίνεται για να δείξουμε ότι αφορά το συγκεκριμένο κείμενο- παραλείπεται)

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Assignment ---
Running...
       tree.dat already exist!So I skip creation process!


hello world
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  11.501 s
Finished at: 2021-01-19T23:41:02+02:00
------------------------------------------------------------------------
```

Δοκιμή με τυχαίο κείμενο, αγνοούμε τους χαρακτήρες που δεν είναι στους ascii όπως βλέπουμε στα υπογραμμισμένα

DecodeOutputFile - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
Text art, also called ASCII art or keyboard art is a copy-pasteable digital age art form. It's about making text pictures
with text symbols.
As we now live in informational societies, I bet you've already encountered those ASCII-painted pics somewhere on Internet.
You can copy and paste text ASCII art to Facebook, Instagram, Snapchat and into any comments, chats, blog posts and forums.

I noticed that visitors of my site like artful text pictures. People have been putting text images composed of symbols into
comments on my pages since the first FB comment box stood the source of my website years ago.I actually deleted that Facebook
comment box after some time, as it took as much time to load as the whole page without it.

So I decided to make a collection of this cool text art.  I started collecting funny text art from comment art and profiles.
I, also, searched the net a bit, but I found just a few good text art that work on Facebook. Right now, there's more, as lots of
people actually copied some to their websites from here .

( ( ) ) The wast majority of ASCII text art pictures in here were submitted as comments by creative FSymbols
visitors just like you. Maybe, exactly you, or your friends.. if not them then say "friends of your friends of your" 2 times -
that's them. I've improved some of the arts to look even better. Thanks, everyone!
```

Επιτυχές αποτέλεσμα *αγνόηση χαρακτήρων που δεν ειναι Ascii

inputFile - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
Text art, also called ASCII art or keyboard art is a copy-pasteable digital age art form. It's about making text pictures
with text symbols.
As we now live in informational societies, I bet you've already encountered those ASCII-painted pics somewhere on Internet.
You can copy and paste text ASCII art to Facebook, Instagram, Snapchat and into any comments, chats, blog posts and forums.

I noticed that visitors of my site like artful text pictures. People have been putting text images composed of symbols into
comments on my pages since the first FB comment box stood the source of my website years ago.I actually deleted that Facebook
comment box after some time, as it took as much time to load as the whole page without it.

So I decided to make a collection of this cool text art. ꒰ ◔ ◡ ◔ ꒱ I started collecting funny text art from comment art and profiles.
I, also, searched the net a bit, but I found just a few good text art that work on Facebook. Right now, there's more, as lots of
people actually copied some to their websites from here ツ.

( ͡° ͜ʖ ͡°( ͡ ͜ʖ ͡ ) ͡° ͜ʖ ͡°) The wast majority of ASCII text art pictures in here were submitted as comments by creative FSymbols
visitors just like you. Maybe, exactly you, or your friends.. if not them then say "friends of your friends of your" 2 times -
that's them. I've improved some of the arts to look even better. Thanks, everyone!

Note, that text pics were made to look fine in Lucida Grande, Tahoma and Verdana fonts, which are default on Facebook.
Sadly, some now look a bit flawed as text message art when viewed in Facebook Messenger, or a similar app on iPhone, as iOS
 switched to a text font with which some of this ASCII art text drawings aren't rendered with appropriate symbol width.

Try typing some text below to turn it into big copy pasteable text art font.
```

φυσικά βγάζουμε warning για αυτό , το προσθέσαμε στην διαδικασία του τρίτου μέρους της εργασίας

```
Running...
        codes.dat already exist!So I skip creation process!

        BE CAREFUL! this program works correctly with 128 first characters of ASCII table!
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  11.861 s
Finished at: 2021-01-19T23:49:40+02:00
------------------------------------------------------------------------
```

In case that someone give us no ASCII characters... Return him a warning!

Να σημειωθεί πώς παραπάνω αναλύσαμε όλες τις διαφορές από τα προηγούμενα μέρη της εργασίας.Ήταν αρκετές καθώς η ιδέα για 2 main μας κίνησε το ενδιαφέρον και προσπαθήσαμε να βελτιώσουμε την εικόνα της εργασίας όσο προγραμματιστικά τόσο και την αναφοράς μας.

Ευχαριστούμε πολύ,για τον χρόνο σας!

Καλή συνέχεια !!!