Δομές Δεδομένων

Τέταρτο Παραδοτέο

Γεώργιος-Νεκτάριος Εγγλέζος it21824
Αντώνης Δρόσος it218115
Ανδρέας Σαραντόπουλος it218140

Εισαγωγή

Για την υλοποίηση του Τέταρτου Παραδοτέου της εργασίας φτιάξαμε μια καινούργια κλάση την Fourth.

Αλλαγές σε προηγούμενα παραδοτέα:

1. Βγάλαμε από την κλάση HuffmanTree την κλάση code.
2. Αλλάξαμε την μορφή που αποθηκεύεται το codes.dat από Χαρακτήρα:κωδικό σε νούμερο:κωδικό.
3. Στην κλάση Third(Για το τρίτο παραδοτέο) έβαλα στο τέλος και έναν ακόμα χαρακτήρα που θα είναι ο EOF (με μορφή στο αρχείο codes.dat “-1:Κωδικός”) για να ξέρω που τελειώνει το encoded κείμενο.
4. Στην κλάση Second (Δεύτερο Παραδοτέο) κατά την δημιουργία των nodes βγάλαμε μια συνθήκη που έβλεπε αν το frequency των χαρακτήρων ascii είναι μεγαλύτερο του μηδενός και τότε τα έφτιαχνε (Σε προηγούμενο παραδοτέο κλάση Second γραμμή 62). Οπότε στο τελικό codes.dat που φτιαχνόταν αργότερα στο τρίτο παραδοτέο δεν αποθηκεύονταν όλοι οι 128 χαρακτήρες με κωδικούς. Τώρα έχουμε κωδικούς και για τους 128 χαρακτήρες ascii.(+Τον χαρακτήρα EOF)
5. Και στην κλάση First βγάλαμε στο σημείο που διαβάζαμε τα κείμενα μια συνθήκη που έλεγχε για δεύτερη φορά αν οι γραμμές του κειμένου που διαβάζαμε ήταν κενές. (Είχαμε currentLine!=null και currentLine!=””)
6. Στο root του φακέλου είχαμε τα τρία αρχεία που φτιάχναμε το frequencies.dat από αυτά. Τώρα τα βάλαμε μέσα σε ένα φάκελο στο root “Files_To_Read” για να είναι λίγο πιο τακτοποιημένα.

Main.java

Βάλαμε έναν έλεγχο για το πόσες λέξεις έδωσε ο χρήστης ως είσοδο από το terminal και αν είναι δύο ακριβώς καλούμε την μέθοδο fourthDeliverable της κλάσης Fourth δίνοντας ως παραμέτρους τις τιμές που έδωσε ο χρήστης.

Fourth.java

Καλείται από την main και παίρνει ως παραμέτρους τα δύο ονόματα που δίνει ο χρήστης από το terminal. Στην συνέχεια διαβάζει τα codes.dat και το αρχείο κειμένου που δίνει ο χρήστης ως είσοδο το μετατρέπει σε ένα String από μηδέν και ένα και μετά το αποθηκεύει σε έναν πίνακα byte για να το γράψει στο αρχείο εξόδου.

Κώδικας

Main.java

```
if (args.length == 2){
  Fourth.fourthDeliverable(args[0], args[1]);
}else{
    System.out.println("Expected two words(file for encoding and name of decoded file)");
}
```

Ελέγχουμε πόσες λέξεις έδωσε ο χρήστης ως είσοδο (αποθηκευμένες στο array args) και αν είναι δύο ακριβώς καλούμε την μέθοδο fourthDeliverable της κλάσης Fourth δίνοντας ως παραμέτρους τις τιμές που έδωσε ο χρήστης. Αν δεν έδωσε 2 εκτυπώνει ανάλογο μήνυμα και τερματίζει το πρόγραμμα.

Third.java

Η έξτρα μέθοδος που προσθέσαμε για την υλοποίηση του τετάρτου παραδοτέου:

```
private static void makeEOFcode(ArrayList<code> codeList){
  String eofCode="0";

  for (code c:codeList){
     if (c.getCode().length()>eofCode.length()){
        eofCode=c.getCode();
     }
  }

  eofCode+="1";
  codeList.add(new code(-1,eofCode));
}
```

Η μέθοδος αυτή φτιάχνει και αποθηκεύει στην λίστα ένα κωδικό EOF για να ξέρω κατά την δημιουργία του αρχείου εξόδου στο τέταρτο παραδοτέο που τελειώνει το γράψιμο του αρχείου. Έχουμε μια μεταβλητή String eofCode που αρχικοποιούμε με 0. Στην συνέχεια με έναν Iterator ψάχνουμε για τον μεγαλύτερο κωδικό στην λίστα codelist συγκρίνοντας το μέγεθος του eofCode με το μέγεθος του κωδικού κάθε index της λίστας codeList (c.getCode().length()>eofCode.length()). Αν το String eofCode είναι μικρότερο τότε αποθηκεύουμε σε αυτό το κωδικό του c.getCode(). Όταν τελειώσουμε με το ψάξιμο του μεγαλύτερου κωδικού προσθέτουμε στο eofCode το “1” ώστε να είναι μοναδικός ο κωδικός του και το βάζουμε στην λίστα.

Fourth.java

Μεταβλητές

```
static private String currentLine;
private static ArrayList<code> codeList = new ArrayList<>();
static StringBuilder encodedText=new StringBuilder();
static ArrayList<Character> notEncodedList=new ArrayList<>();
```

- **codeList είναι η λίστα που αποθηκεύω τους κωδικούς του Codes.dat**
- **encodedText είναι ένα StringBuilder με το κείμενο σε μορφή 0 και 1**
- **notEncodedList είναι μια λίστα που αποθηκεύω όλους τους χαρακτήρες που δεν μπόρεσα να αποθηκεύσω για να ενημερώσω τον χρήστη**

fourthDeliverable()

```
public static void fourthDeliverable(String inputFile, String outputFile) throws
IOException {
  //Extract the codes
  loadCodeList();

  //Read and encode the input file
  encodedText = encodeInputFile(inputFile);

  //make a byte array
  //determine the size of the array
  int size = encodedText.length() / 8;
  if (encodedText.length() % 8 != 0) {
     size = encodedText.length() / 8 + 1;
  }

  //Fill the array
  byte[] byteArr = new byte[size];
  getBytes(byteArr, encodedText.toString());

  //make the encoded file
  makeEncodedFile(outputFile, byteArr);

  //Those characters weren't encoded
  if (notEncodedList.size() > 0) {
     String tmp = "Those characters couldn't be encoded:";
     for (Character c : notEncodedList) {
        tmp += "Char: " + c + " ";
     }
     System.out.println(tmp);
  }
}
```

Παίρνει ως παραμέτρους δύο String. Διαβάζει το αρχείο codes.dat καλώντας την μέθοδο ***loadCodeList()*** (Θα αναλύσουμε παρακάτω). Καλεί μέθοδο ***encodeInputFile()*** (Θα αναλύσουμε παρακάτω) που επιστρέφει ένα StringBuilder με το κείμενο σε μορφή 0 και 1. Στην συνέχεια υπολογίζει το μέγεθος του κειμένου ώστε να φτιάξουμε ένα array τύπου byte. Καλούμε την μέθοδο ***getBytes***(Θα αναλύσουμε παρακάτω) για να γεμίσουμε αυτόν τον πίνακα και περνάμε ως παραμέτρους τον πίνακα και το encoded κείμενο. Καλούμε την συνάρτηση ***makeEncodedFile***(Θα αναλύσουμε παρακάτω) για να φτιάξουμε το αρχείο εξόδου δίνοντας ως παραμέτρους το όνομα του αρχείου εξόδου και τον πίνακα. Τέλος φτιάχνουμε και εκτυπώνουμε ένα String με όλους τους χαρακτήρες που χρησιμοποίησε ο χρήστης αλλά δεν υπάρχουν στο codes.dat. Το String το γεμίζουμε χρησιμοποιώντας ένα Iterator για να πάρουμε όλες τις τιμές της λίστας notEncodedList.

readFile()

```
private static BufferedReader readFile(String fileName)
{
  try {
     BufferedReader inputStream = new BufferedReader(new FileReader(fileName));
     return inputStream; //returns the whole txt file as a bufferedReader
  }
  catch (FileNotFoundException ex) { //If one or more files aren't in the project folder
     System.out.println("file " + fileName + " was not found.");
     return null;
  }
}
```

Χρησιμοποιώντας ένα BufferedReader διαβάζει αρχείο που δίνουμε ως παράμετρο και επιστρέφει τον BufferedReader. Με Try-Catch ελέγχουμε αν το αρχείο υπάρχει και αν δεν υπάρχει εμφανίζουμε ανάλογο μήνυμα και επιστρέφουμε null στην μέθοδο που την κάλεσε.

loadCodeList()

```
private static void loadCodeList() throws IOException {
BufferedReader inputCodesStream = readFile("codes.dat");
if (inputCodesStream == null) {
  System.out.println("Didn't find codes.dat, exiting now!");
  System.exit(0);
  return;
}
  while ((currentLine = inputCodesStream.readLine()) != null) {
     String tmp;
     String c = null;
     StringTokenizer st2 = new StringTokenizer(currentLine);
     int index = 0;
     while (st2.hasMoreTokens()) {
        tmp = st2.nextToken(":");
        if (index == 0) {
           index++;
           c = tmp;
        } else {
           codeList.add(new code(Integer.parseInt(c), tmp));
        }
     }
  }
}
```

Για το διάβασμα του αρχείου codes.dat καλούμε την συνάρτηση readFile που εξηγήσαμε από πάνω και παίρνουμε το bufferedReader **inputCodesStream**. Σε περίπτωση που το **inputCodesStream** είναι null δηλαδή δεν βρήκε το αρχείο εμφανίζουμε ανάλογο μήνυμα και τερματίζουμε το πρόγραμμα. Αν δεν είναι null τρέχουμε μια while loop όσο υπάρχουν

γραμμές ***currentLine*=inputCodesStream.readLine()** στο buffered reader και με ένα StringTokenizer σπάμε την γραμμή στα δύο κομμάτια που είναι το ascii και ο κωδικός και τα αποθηκεύουμε στην λίστα Arraylist<code> codeList (Η κλάση code φτιάχτηκε σε προηγούμενο παραδοτέο και περιέχει τις δύο μεταβλητές του ascii και κωδικού ,constructor, getters και setters).

getCode()

```
private static String getCode(ArrayList<code> codeList, char c) {
  String code = "";
  boolean found = false;
  for (code current : codeList) {
     if ((int)current.getC() ==(int) c) {
        code = current.getCode();
        found = true;
        break;
     }
  }
  if (!found) {
     boolean inList = false;
     for (Character ch : notEncodedList) {
        if (ch == c) {
           inList = true;
        }
     }
     if (!inList) {
        notEncodedList.add(c);
     }
  }
  return code;
}
```

Η μέθοδος αυτή είναι για να πάρουμε τον κωδικό ενός χαρακτήρα. Παίρνει ως παραμέτρους την λίστα codeList και τον χαρακτήρα c. Εκεί με έναν iterator περνάει όλη την λίστα και συγκρίνει συνέχεια τον χαρακτήρα c με το **current.getC()** που είναι ο getter της κλασης code που μας δίνει το ascii. Αν βρει τον κωδικό ορίζει μια μεταβλητή boolean **found=true**. Αν το found είναι false σημαίνει πως ο χαρακτήρας δεν ήταν στην λίστα codeList οπότε πρέπει να δω αν τον έχω βάλει στην λίστα notEncodedList με τους χαρακτήρες που δεν έχουν κωδικούς. Αυτό γίνεται πάλι με έναν iterator και έναν απλό έλεγχο. Αν βρει τον χαρακτήρα μέσα στην λίστα ορίζει μια μεταβλητή boolean inList=true. Αν το inList είναι false αποθηκεύουμε τον χαρακτήρα στην λίστα notEncodedList για να τον εκτυπώσουμε μετά ως έξτρα πληροφορία στον Χρήστη.

encodeInputFile()

```
private static StringBuilder encodeInputFile(String inputFile) throws IOException {
  BufferedReader inputFileStream = readFile(inputFile);
  if (inputFileStream == null) { //If
     System.out.println("inputFile doesn't exist");
     System.exit(0);
  }
  while ((currentLine = inputFileStream.readLine()) != null) {
     char[] arr;
     arr = currentLine.toCharArray();//To get the characters from the Line
     for (int i = 0; i < arr.length; i++) {
        String code = getCode(codeList, arr[i]); //get the code of the character
        encodedText.append(code);
     }
  }
  encodedText.append(codeList.get(codeList.size() - 1).getCode()); //EOF
  return encodedText;
}
```

Η μέθοδος αυτή είναι υπεύθυνη για να φτιάξει το String με τα bits. Παίρνει ως παράμετρο το όνομα του inputFile και με την μέθοδο readFile() παίρνει το κείμενο σε έναν bufferedReader. Αν ο bufferedReader **inputFileStream** είναι null τερματίζουμε το πρόγραμμα. Στην συνέχεια προσπελάσουμε τις γραμμές του κειμένου. Με το *currentLine*.toCharArray() σπάμε την γραμμή που είναι String σε έναν πίνακα χαρακτήρων. Με μια for loop βρίσκουμε για κάθε index του πίνακα των κωδικό του χαρακτήρα χρησιμοποιώντας την μέθοδο getCode() που αναλύσαμε παραπάνω. Ότι επιστρέφει η μέθοδος getCode το κάνουμε append σε ένα StringBuilder (*encodedText*.append(code)). Αφού τελειώσουμε με την προσπέλαση του κειμένου κάνουμε append στο StringBuilder και τον χαρακτήρα EOF που φτιάξαμε στο Τρίτο παραδοτέο.Το EOF βρίσκεται στην τελευταία θέση της λίστας codeList (*codeList*.size() - 1).

getBytes()

```
private static void getBytes(byte[] byteArr, String s) {
  int counter = 0;
  while (s.length() > 7) {
     String currentByte = s.substring(0, 8);
     s = s.substring(8);
     byte mybyte = 0;
     char[] myInput = currentByte.toCharArray();

     for (int j = 0; j < myInput.length; j++) {
        if (myInput[j] == '1') {
           mybyte |= 1 << 7 - j;
        }
     }
     byteArr[counter++] = mybyte;
  }
  if (s.length() > 0) {
     byte myByte = 0;
     for (int i = s.length() - 1; i < 8; i++) {
        s += 0;
     }
     char[] myInput = s.toCharArray();
     for (int j = 0; j < myInput.length; j++) {
        if (myInput[j] == '1') {
           myByte |= 1 << 7 - j;
        }
     }
     byteArr[counter++] = myByte;
  }
}
```

Η μέθοδος αυτή είναι για να γεμίζει τον πίνακα με τα bytes. Παίρνει ως παραμέτρους το String s (μορφής μηδέν και ένα) και το byte[] array που φτιάξαμε στην fourthDeliverable(). Στην συνέχεια έχουμε μια while επανάληψη που τρέχει όσο τα bits της μεταβλητής s είναι περισσότερα από 7. Όσο υπάρχουν τουλάχιστον 8 bits αποθηκεύουμε σε μια άλλη μεταβλητή τα πρώτα 8 bits και στην συνέχεια τα αφαιρούμε από το αρχικό String s. Αυτό γίνεται με την χρήση του **substring**. Για την αποθήκευση των bits σε ένα index του πίνακα byte[] σπάμε το String σε έναν πίνακα χαρακτήρων (char[] myInput = currentByte.toCharArray()) και με μια επανάληψη ελέγχουμε όλες τις τιμές του πίνακα. Αν η τιμή είναι ίση με “1” χρησιμοποιούμε τα bitwise operations << “left swift” και | “Inclusive OR” για να φτιάξουμε την τιμή μιας προσωρινής μεταβλητής ως συμπλήρωμα του 2. Μόλις βγούμε από την επανάληψη αυτή το αποθηκεύουμε στον πίνακα. Όταν τελειώσουμε και με την μεγάλη επανάληψη ελέγχουμε πόσα bits έμειναν στην μεταβλητή s. Αν είναι περισσότερα από 0 συμπληρώνουμε ώστε να φτάσουν 8 ακριβώς bits και επαναλαμβάνουμε την από πάνω διαδικασία. (Ενδέχεται να υπάρχουν και πολύ πιο απλοί/καλοί τρόποι για την δημιουργία των bytes αλλά ήταν το μόνο που δούλεψε για εμάς. Στο επόμενο παραδοτέο ίσως δούμε την κλάση bitset που αναφέρθηκε στην εκφώνηση)

makeEncodedFile()

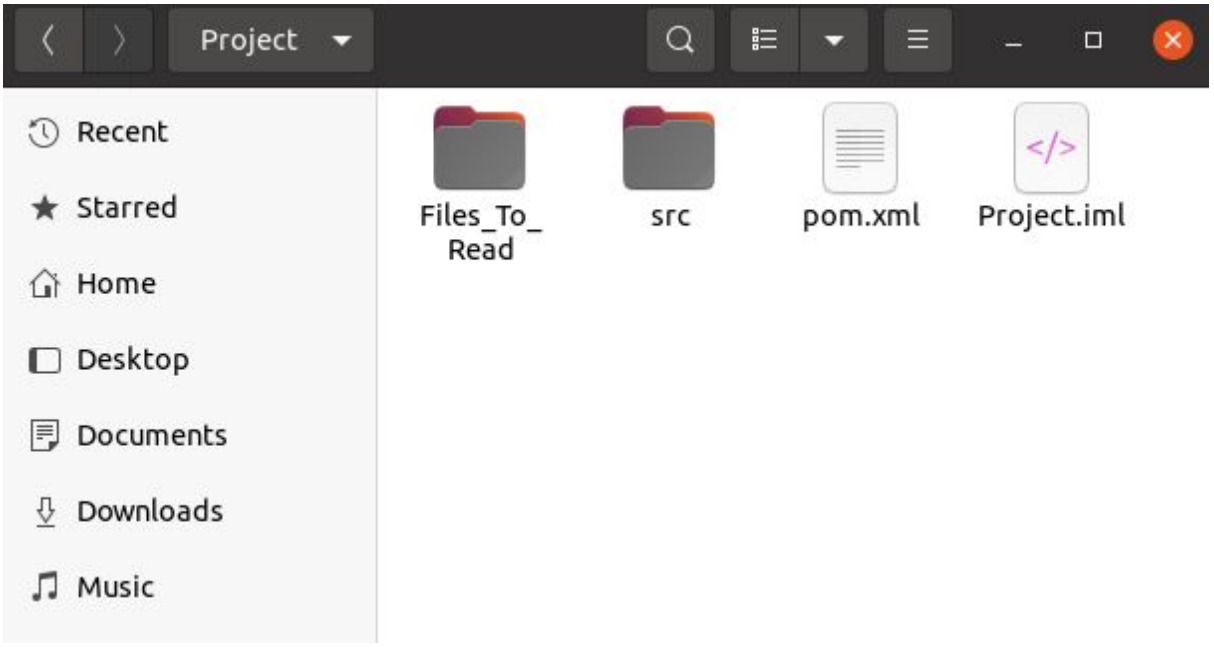
```
private static void makeEncodedFile(String outputName, byte[] byteList) throws
IOException {

   OutputStream outputStream = new FileOutputStream(outputName);
   outputStream.write(byteList);
   outputStream.close();
   System.out.println(outputName+" is ready in the project folder!");
}
```

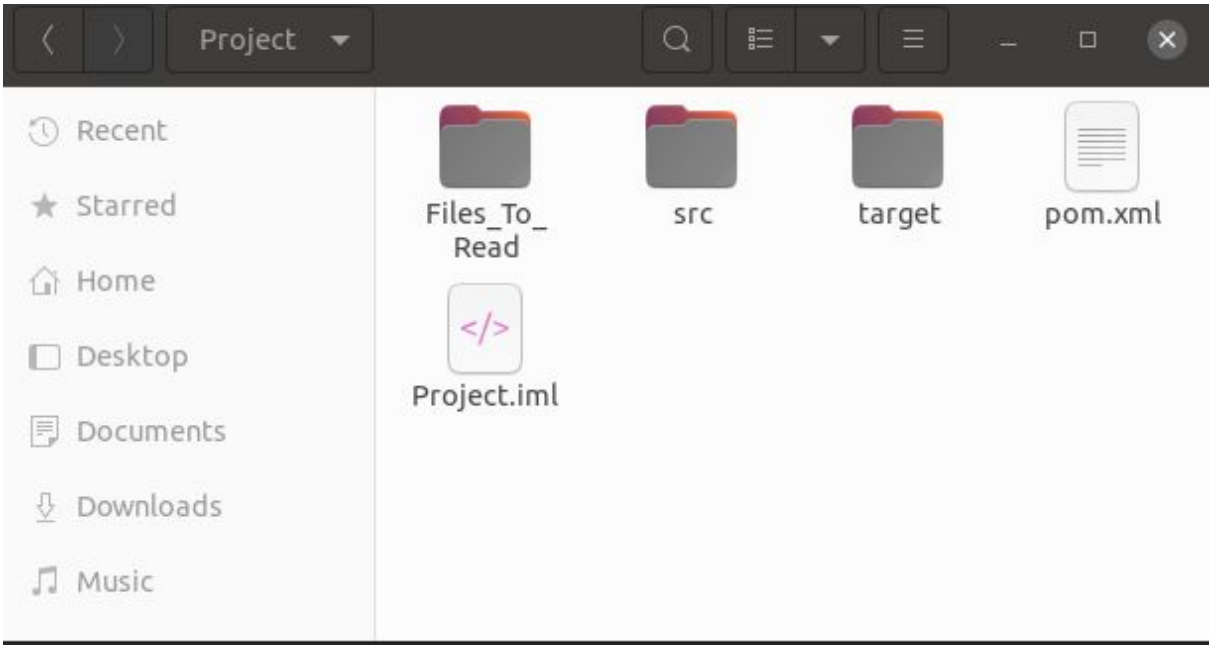
Η μέθοδος αυτή δημιουργεί το αρχείο εξόδου. Παίρνει ως παράμετρο το όνομα του αρχείου εξόδου και τον πίνακα με τα bytes. Φτιάχνει ένα FileOutputStream , γράφει τον πίνακα στο αρχείο και κλείνει το stream.

Ενδεικτικό Αποτέλεσμα

Αρχικά στον φάκελο του Project υπάρχουν μόνο τα βασικά αρχεία και ο φάκελος με τα αρχεία κειμένου Files_To_Read.



Τρέχουμε την εντολή “**mvn package**” για να φτιαχτεί ο φάκελος target με το αρχείο jar

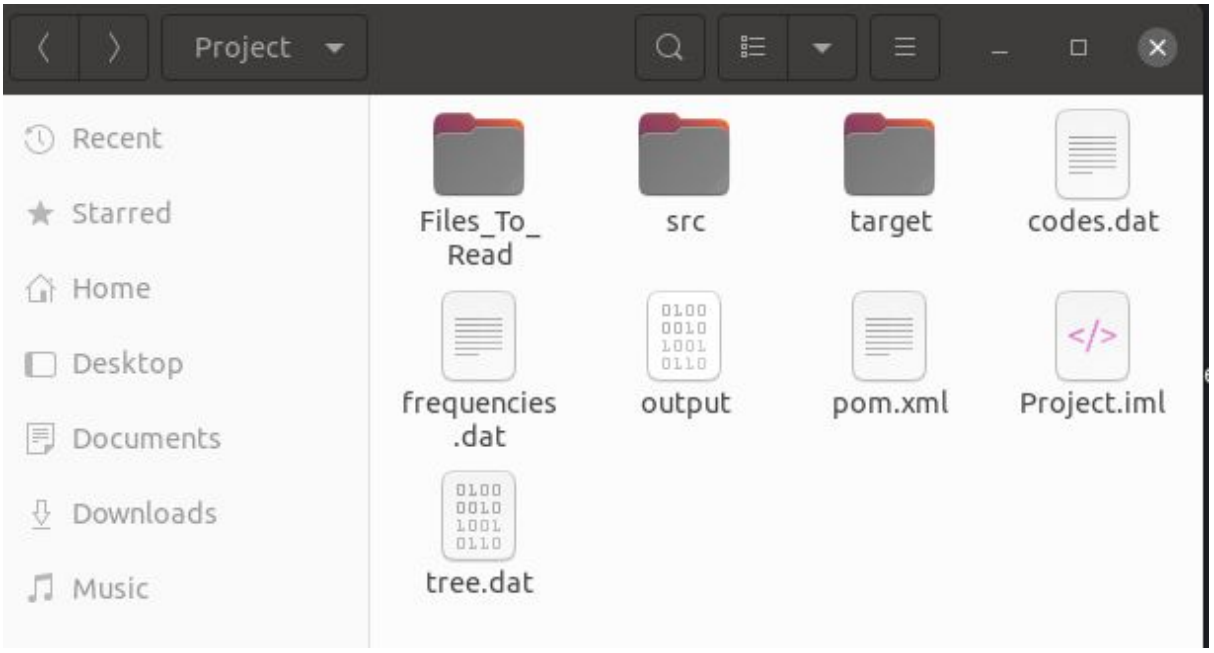


Σωστή Εκτέλεση

Τέλος τρέχουμε την εντολή “**java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main Files_To_Read/test_Input.txt output**” για να τρέξει το πρόγραμμα με input ένα αρχείο που βάλαμε στον φάκελο **Files_To_Read** και output το όνομα που θέλουμε να χει το αρχείο εξόδου.

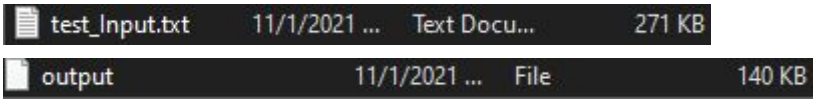
```
 java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main Files_To_Read/test_Input.txt output
Frequencies.dat is ready in the project folder!
The tree is ready!
tree.dat is ready in the Project Folder!
codes.dat is ready in the project folder!
output is ready in the project folder!
Those characters couldn't be encoded:Char: “ Char: ” Char: ’ Char: – Char: ‘ Char: ê Char: à Char:
é
```

και έτσι στο τέλος έχουμε αυτά τα αρχεία στο root του φακέλου Project:



Σύγκριση αρχείων εισόδου και εξόδου:

Το αρχείο εισόδου ήταν 271kb ενώ το αρχείο εξόδου είναι 140kb



Λάθος Εκτέλεση

1)
Δίνουμε μόνο ένα ή κανένα ή περισσότερα από δύο args όταν τρέχουμε το πρόγραμμα.
Π.χ. Τρέχουμε την εντολή **“java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main Files_To_Read/test_Input.txt”** που δίνει μόνο το όνομα του αρχείου εισόδου. Και εμφανίζει στο terminal:

```
 java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main Files_To_Read/test_Input.txt
Frequencies.dat is ready in the project folder!
The tree is ready!
tree.dat is ready in the Project Folder!
codes.dat is ready in the project folder!
Expected two words!(file for encoding and name of decoded file)
```

2)
Δίνουμε αρχείο εισόδου που δεν υπάρχει **“java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main file output”**:

```
java -cp target/Project-1.0-SNAPSHOT.jar org.hua.project.Main file output
Frequencies.dat is ready in the project folder!
The tree is ready!
tree.dat is ready in the Project Folder!
codes.dat is ready in the project folder!
file file was not found.
inputFile doesn't exist
```