Ψηφιακές Επικοινωνίες – Προγραμματιστική Εργασία
Υλοποίηση CRC
Γλώσσα προγραμματισμού: Java (11)
Αρχείο υλοποίησης: “CRCOperations.java”

Ονοματεπώνυμο: Μπαρακλιλής Ιωάννης
ΑΕΜ: 3685
email: imparakl@csd.auth.gr

(Σημείωση: κατά την μεταγλώττιση, σε περίπτωση προβλημάτων με κωδικοποίηση μπορεί να προστεθεί το “-encoding UTF-8” στην εντολή (γραμμής εντολών) για μεταγλώττιση. Πχ. να χρησιμοποιηθεί η εντολή “javac CRCOperations.java -encoding UTF-8” για μεταγλώττιση.)

Σύντομη Περιγραφή Σημαντικότερων σημείων στον κώδικα

Δημιουργία τυχαία επιλεγμένων δυαδικών μηνυμάτων των k bits, στο μεταδότη (μπλοκ δεδομένων των k bits, σε κάθε bit των οποίων, το 0 και το 1 έχουν ίση πιθανότητα εμφάνισης)

Η λειτουργία αυτή επιτυγχάνεται μέσω της μεθόδου generateRandomMessage[1] που δέχεται ως όρισμα έναν ακέραιο k και μέσω μίας γεννήτριας τυχαίων αριθμών δημιουργεί ένα τυχαίο δυαδικό μήνυμα των k bits.
Συγκεκριμένα, επιλέγει ένα-ένα k bits και τοποθετεί το ένα μετά το άλλο χρησιμοποιώντας πράξεις bit. Η επιλογή γίνεται δημιουργώντας έναν τυχαίο δεκαδικό αριθμό στο διάστημα [0, 1] και αν αυτός είναι μεγαλύτερος του 0.5 επιλέγεται 1, διαφορετικά επιλέγεται το 0, έτσι εξασφαλίζεται η ισοπίθανη εμφάνιση 0 και 1. Τέλος, επιστρέφει το μήνυμα που κατασκεύασε.

Υπολογισμός του CRC (FCS) που αντιστοιχεί σε κάθε μήνυμα. Ως πρότυπο για τον υπολογισμό του CRC θα χρησιμοποιηθεί ένας δυαδικός αριθμός P που θα δινει ο χρήστης

Η λειτουργία αυτή επιτυγχάνεται μέσω των μεθόδων XORDivisionRemainder[2] και calculateCRC[3] .
Για τον υπολογισμό του CRC αρκεί η κλήση της calculateCRC με κατάλληλα ορίσματα.

Η μέθοδος **XORDivisionRemainder** δέχεται δύο ακεραίους και υπολογίζει και επιστρέφει το υπόλοιπο της διαίρεσης (με modulo-2 αριθμητική) του πρώτου με τον δεύτερο.
Αρχικά, ελέγχεται αν δεν μπορεί να γίνει διαίρεση (αν το μήκος του διαιρέτη είναι μεγαλύτερο από εκείνο του διαιρετέου). Σε εκείνη την περίπτωση, η διαίρεση έχει τελειώσει και, προφανώς, το υπόλοιπο είναι το ίδιος ο διαιρετέος.
Στην συνέχεια, ευθυγραμμίζονται διαιρέτης και διαιρετέος μετακινώντας (με αντίγραφο) τα bit του διαιρετέου προς τα δεξιά (το αποτέλεσμα θα είναι το αρχικό “τρέχον” υπόλοιπο) και αρχίζει η διαίρεση.

---

1 Μέθοδος generateRandomMessage(int) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 8-40.
2 Μέθοδος XORDivisionRemainder(long, long) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 42-112.
3 Μέθοδος calculateCRC(long, long) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 114-131.

Κατά την διαίρεση, γίνεται η πράξη XOR μεταξύ του “τρέχοντος” υπολοίπου (το αποτέλεσμα της προηγούμενης πράξης XOR στο τέλος του οποίου έχουν τοποθετηθεί ψηφία του διαιρετέου που “κατεβαίνουν”) με τον διαιρέτη. Μετά, “κατεβαίνει” κατάλληλος αριθμός από ψηφία του διαιρετέου ώστε να ευθυγραμμιστεί με τον διαιρέτη (η ευθυγράμμιση ελέγχεται αν το πιο αριστερό (leftmost) bit του διαιρέτη και το bit του διαιρέτη στην αντίστοιχη θέση είναι και τα δύο 1). Η διαίρεση συνεχίζεται μέχρις ότου να “εξαντληθούν” τα ψηφία “προς κατέβασμα” του διαιρετέου. Τέλος, γίνεται ένας τελευταίος έλεγχος ευθυγράμμισης εφόσον μπορεί να έχουν “εξαντληθεί” τα ψηφία “προς κατέβασμα” αλλά να μπορεί να είναι ευθυγραμμισμένα και να μπορεί να γίνει μία ακόμη πράξη, αν αυτό ισχύει γίνεται μία ακόμη πράξη XOR. Στην μεταβλητή “τρέχοντος” υπολοίπου υπάρχει το τελικό υπόλοιπο της διαίρεσης το οποίο και επιστρέφεται.

Η μέθοδος **calculateCRC** δέχεται δύο ακεραίους, έναν που περιέχει τα bit του μηνύματος (D) και έναν που περιέχει τον αριθμό ο οποίος θα χρησιμοποιηθεί ως βάση για την παραγωγή του CRC/FCS (P), και υπολογίζει το CRC της ακολουθίας D.
Ο τρόπος με τον οποίο το επιτυγχάνει αυτό είναι τοποθετώντας n-k (n ο αριθμός των bit του τελικού μηνύματος που αποτελείται από το αρχικό μήνυμα που ακολουθείται από τα bits του CRC και k ο αριθμός bit του αρχικού μηνύματος) μηδενικά στα δεξιά του D. Αυτό μπορεί τα επιτευχθεί κάνοντας αριστερή ολίσθηση κατά το (πλήθος των bit του P) - 1 (ο P έχει πλήθος bit n–k+1 άρα n–k+1–1 = n – k). Στην συνέχεια εκτελεί διαίρεση (modulo-2) του αριθμού αυτού με τον P με την βοήθεια της συνάρτησης XORDivisionRemainder, το υπόλοιπο της οποίας είναι η ακολουθία CRC (FCS) την οποία επιστρέφει.

Μετάδοση του μηνύματος και του CRC μέσω ενός ενθόρυβου καναλιού με Bit Error Rate BER και παραλαβή του «αλλοιωμένου» μηνύματος στον αποδέκτη

Η λειτουργία αυτή επιτυγχάνεται μέσω των μεθόδων calculateT[4] και της simulateTransmissionThroughNoisyChannel[5].

Η μέθοδος **calculateT** δέχεται δύο ακεραίους, έναν που περιέχει τα bit μηνύματος προς μετάδοση (D) και έναν που περιέχει τον αριθμό ο οποίος θα χρησιμοποιηθεί ως βάση για την παραγωγή του CRC/FCS (P), και υπολογίζει την τελική ακολουθία προς μετάδοση που αποτελείται από το D ακολουθούμενο από την ακολουθία CRC (FCS).
Ο τρόπος με τον οποίο το κάνει αυτό είναι η τοποθέτηση n-k μηδενικά στα δεξιά του D, υπολογισμό του CRC (FCS) με την βοήθεια της μεθόδου calculateCRC και η ακολουθία προς μετάδοση ισούται με το άθροισμα του μετατοπισμένου D με την ακολουθία CRC (εφόσον ο D έχει n-k μηδενικά στο τέλος του και το CRC (FCS) αποτελείται από n-k ψηφία, το άθροισμα τους αποτελεί “σύντμηση” του D και του CRC) και επιστρέφει το αποτέλεσμα.

Η μέθοδος **simulateTransmissionThroughNoisyChannel** δέχεται ένα δυαδικό μήνυμα με την μορφή (μεγάλου) ακεραίου, το μήκος του μηνύματος αυτού (ακέραιος) και το BER του καναλιού (πραγματικός) και προσομοιώνει μετάδοση μηνύματος μέσω ενός ενθόρυβου καναλιού με Bit Error Rate BER και επιστρέφει το "αλλοιωμένο" μήνυμα.
Αυτό το πετυχαίνει με την βοήθεια μία γεννήτριας τυχαίων αριθμών από την οποία λαμβάνει έναν πραγματικό αριθμό στο διάστημα [0,1] και για κάθε bit του μηνύματος ελέγχει αν ο ληφθέν πραγ-

4 Μέθοδος calculateT(long, long) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 133-153.
5 Μέθοδος simulateTransmissionThroughNoisyChannel(long, int, double) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 167-200.

ματικός είναι μικρότερος από το BER, τότε αλλάζει το συγκεκριμένο bit (από 0 σε 1 και αντιστρόφως). Τέλος, επιστρέφει το τελικό μήνυμα με τα, πιθανώς, αλλαγμένα bits.

Έλεγχος του CRC στον αποδέκτη

Η λειτουργία αυτή επιτυγχάνεται με την μέθοδο checkCRC[6]. Αυτή, δέχεται δύο ακεραίους, έναν που περιέχει τα bit του τελικού μηνύματος και έναν που περιέχει τον αριθμό ο οποίος αποτέλεσε βάση για την παραγωγή του CRC/FCS (P) και ελέγχει (επιστρέφει λογική τιμή) για το αν το τελικό μήνυμα είναι έγκυρο με βάση το αν διαιρείται ακριβώς με το P. Αυτό, επιτυγχάνεται με απλή διαίρεση του τελικού μηνύματος με το P. Αν το υπόλοιπο είναι 0, τότε το μήνυμα είναι έγκυρο, διαφορετικά δεν είναι.

Υπολογισμός ποσοστών και παραδείγματα λειτουργίας κώδικα.

Οι υπολογισμοί ποσοστών και εμφάνιση τους γίνεται στην μέθοδο main[7] με την εκτέλεση του προγράμματος.
Αρχικά, ορίζονται οι σταθερές που δίνονται στην εκφώνηση (k=20, P=110101, BER=10^{-3}) και αρχικοποιούνται οι μετρητές. Στην συνέχεια, για 3000000 τυχαία δημιουργημένα μηνύματα, υπολογίζεται το CRC (FCS) και προς μετάδοση μήνυμα τους, προσομοιώνεται η μετάδοση τους σε ενθόρυβο κανάλι και τέλος ελέγχεται αν εντοπίζεται ως εσφαλμένο από παραβίαση του κανόνα του CRC και έλεγχος (για υπολογισμό στατιστικών) αν πραγματικά έφτασε εσφαλμένο και μετρώνται ο αριθμός των εσφαλμένων (είτε ανιχνεύθηκαν από το CRC, είτε όχι) και εσφαλμένων που εντοπίστηκαν από το CRC. Τέλος, εμφανίζονται στην έξοδο το ποσοστό μηνυμάτων που φτάνουν με σφάλμα (στο block δεδομένων ή στο CRC) στον αποδέκτη, το ποσοστό μηνυμάτων που ανιχνεύονται ως εσφαλμένα από το CRC και το ποσοστό μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη και δεν ανιχνεύονται από το CRC.

Παραδείγματα εκτέλεσης κώδικα:

```
Ποσοστό μηνυμάτων που φτάνουν με σφάλμα (στο block δεδομένων ή στο CRC) στον αποδέκτη: 0.025741 (2.574067%)
Ποσοστό μηνυμάτων που ανιχνεύονται ως εσφαλμένα από το CRC: 0.025729 (2.572933%)
Ποσοστό μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη και δεν ανιχνεύονται από το CRC: 0.000011 (0.001133%)
```

```
Ποσοστό μηνυμάτων που φτάνουν με σφάλμα (στο block δεδομένων ή στο CRC) στον αποδέκτη: 0.025771 (2.577067%)
Ποσοστό μηνυμάτων που ανιχνεύονται ως εσφαλμένα από το CRC: 0.025760 (2.575967%)
Ποσοστό μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη και δεν ανιχνεύονται από το CRC: 0.000011 (0.001100%)
```

```
Ποσοστό μηνυμάτων που φτάνουν με σφάλμα (στο block δεδομένων ή στο CRC) στον αποδέκτη: 0.025581 (2.558133%)
Ποσοστό μηνυμάτων που ανιχνεύονται ως εσφαλμένα από το CRC: 0.025572 (2.557233%)
Ποσοστό μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη και δεν ανιχνεύονται από το CRC: 0.000009 (0.000900%)
```

Όπως μπορούμε να δούμε από τα παραπάνω παραδείγματα (ως τιμές επιλέγω εκείνες του πρώτου) έχουμε:

- Ποσοστό των μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη: 0.025741 (2.574067%)
- Ποσοστό των μηνυμάτων που ανιχνεύονται ως εσφαλμένα από το CRC: 0.025729 (2.572933%)
- Ποσοστό των μηνυμάτων που φθάνουν με σφάλμα στον αποδέκτη και δεν ανιχνεύονται από το CRC: 0.000011 (0.001133%)

6 Μέθοδος checkCRC(long, long) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 155-165.
7 Μέθοδος main(String[]) υλοποιημένη στο αρχείο CRCOperations.java, γραμμές 202-248.