

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Πληροφορικής

**Έβδομη εργαστηριακή αναφορά για
NCO-04-05**

Έλεγχος ισοτιμίας

Αλέξανδρος Κόρκος
alexkork@csd.auth.gr
3870

Θεσσαλονίκη, 28 Δεκεμβρίου 2022



Το έργο αυτό διατίθεται υπό τους όρους της άδειας **Create Commons**
"Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0
Διεθνές".

Άσκηση 1

Εκφώνηση: Περιγράψτε την έννοια της ισοτιμίας και δημιουργήστε κώδικα, που να υπολογίζει άρτια ισοτιμία για ένα μήνυμα των 14 bits και να προσαρτά σε αυτό ένα bit ισοτιμίας. Τρέξτε τον κώδικά σας για το μήνυμα 10001101011000.

Απάντηση: Η ισοτιμία είναι μια πολύ απλή μέθοδος, εύρεσης σφαλμάτων σε (δυαδικά μηνύματα) όπου εξετάζεται το πλήθος των bit με τιμή "1" σε ένα μήνυμα. Η διαδικασία έχει ως εξής:

- Ο πομπός και ο δέκτης, θα πρέπει να έχουν συμφωνήσει εάν θα χρησιμοποιηθεί περιττή ή άρτια ισοτιμία.
- Ο πομπός, σε ένα μήνυμα μεγέθους n bit, προσθέτει στο τέλος του μηνύματος ένα ακόμα bit όπου η τιμή του εξαρτάται ανάλογα με τι είδους ισοτιμίας έχει συμφωνηθεί (δηλ. εάν έχουμε άρτια ισοτιμία, και το μήνυμα είναι 1010110 θα προστεθεί 0 ενώ για 1010100 θα προστεθεί 1).
- Εάν έχει συμφωνηθεί άρτια ισοτιμία, ο δέκτης εξετάζει εάν το πλήθος των "1" του μηνύματος είναι άρτιο ενώ για περιττή εξετάζει για περιττό πλήθος "1".
- Εάν το πλήθος είναι το συμφωνημένο, τότε το μήνυμα γίνεται αποδεκτό διαφορετικά απορρίπτεται.

Για τον υπολογισμό της άρτιας ισοτιμίας, υλοποιήθηκε η παρακάτω συνάρτηση (computeParity):

```

1 function msg = computeParity(x)
2     N = length(x);
3     countOnes = 0;
4     for i = 1:N
5         if (x(i))
6             countOnes = countOnes + 1;
7         endif
8     endfor
9     if (mod(countOnes, 2) ~= 0)
10        msg = [x 1];
11    else
12        msg = [x 0];
13    endif
14 endfunction

```

Κώδικας 1: Υλοποίηση αλγορίθμου άρτιας ισοτιμίας (στο αρχείο computeParity.m)

Παρακάτω ακολουθεί η υλοποίηση της πρώτης άσκησης:

```

1 msg = input("Give binary number (14-bit): ");
2 N = length(msg);
3
4 while (N ~= 14)
5     msg = input("Give binary number (14-bit): ");
6     N = length(msg);
7 endwhile
8
9 msg = computeParity(msg);
10
11 disp("Original message + parity bit: "), disp(msg)

```

Κώδικας 2: Υλοποίηση της πρώτης άσκησης

Δίνοντας στον κώδικά 2 το δυαδικό μήνυμα 10001101011000¹ επιστρέφεται ένα το μήνυμα μαζί με το bit ισοτιμίας, το οποίο είναι: 100011010110000.

Άσκηση 2

Εκφώνηση: Δημιουργείστε κώδικα που να ελέγχει αν υπάρχει σφάλμα σε ένα bit για το μήνυμα των 14 bits που συνοδεύεται από ένα bit άρτιας ισοτιμίας. Τρέξτε τον κώδικά σας για να ελέγξετε το αν έχει φτάσει χωρίς σφάλμα το μήνυμα 10001101011000 του προηγούμενου σκέλους της άσκησης (και το bit ισοτιμίας που το συνοδεύει) στις εξής περιπτώσεις:

1. Άφιξη στον αποδέκτη χωρίς σφάλμα.
2. Άφιξη στον αποδέκτη με ένα σφάλμα (στο 6ο bit, με αρίθμηση των bit μετρώντας από αριστερά).
3. Άφιξη στον αποδέκτη με δύο σφάλματα (στο 6ο bit και στο 12ο bit, με αρίθμηση των bit μετρώντας από αριστερά).

Απάντηση:

Για να ελεγχθεί εάν ένα μήνυμα έχει φτάσει χωρίς σφάλμα στον δέκτη, δημιουργήθηκε η συνάρτηση `validateParity` όπως φαίνεται και παρακάτω:

¹Η εισαγωγή του μηνύματος στο περιβάλλον Octave πρέπει να γίνει με αυτή την μορφή: [1 0 0 0 1 1 0 1 0 1 1 0 0 0].

```

1 function isValid = validateParity(msg)
2     N = length(msg);
3     countOnes = 0;
4     for i = 1:N
5         if (msg(i))
6             countOnes = countOnes + 1;
7         endif
8     endfor
9     if (mod(countOnes, 2) == 0)
10         isValid = 1;
11     else
12         isValid = 0;
13     endif
14 endfunction

```

Κώδικας 3: Περιεχόμενο του αρχείου validateParity.m

Χρησιμοποιώντας της συναρτήσεις 1 και 3 υλοποιείται η δεύτερη άσκηση.

```

1 msg = input("Give binary number (14-bit): ");
2
3 msg = computeParity(msg);
4
5 msg = bitManipulation(msg, 6);
6
7 if (validateParity(msg))
8     disp("Message is OK.");
9 else
10    disp("Message is corrupted.");
11 endif
12
13 msg = bitManipulation(msg, 12);
14
15 if (validateParity(msg))
16     disp("Message is OK.");
17 else
18     disp("Message is corrupted.");
19 endif

```

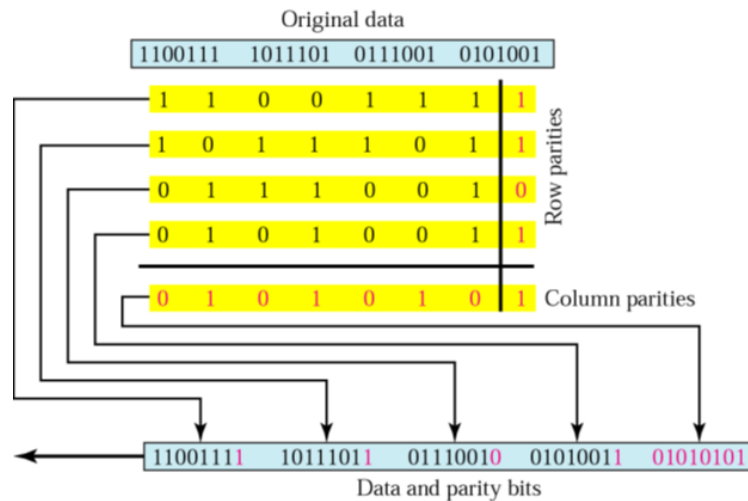
Κώδικας 4: Υλοποίηση δεύτερης άσκησης

Εκτελώντας τον κωδικά 4 για 10001101011000 παρατηρείται πως για την άφιξη με ένα σφάλμα γίνεται αντιληπτό από τον αλγόριθμο ενώ για δυο σφάλματα όχι. Αυτό γίνεται γιατί, ενώ αλλοιώνεται το μήνυμα το πλήθος των bit με τιμή "1" παραμένει ίδιο.

Άσκηση 3

Εκφώνηση: Δημιουργείστε κώδικα που να διαιρεί ένα μήνυμα των 28 bits σε 4 γραμμές των 7 bits (όπως φαίνεται στο σχήμα), να υπολογίζει τα bits άρτιας

δισδιάστατης ισοτιμίας για κάθε γραμμή και κάθε στήλη και να τα προσαρτά σε αυτές.



Σχήμα 1: Τρίτη άσκηση

Τρέξτε τον κώδικά σας για το συγκεκριμένο μήνυμα που φαίνεται στο παραπάνω σχήμα (original data).

Απάντηση:

Για την υλοποίηση αυτής της άσκησης δημιουργήθηκαν δυο συναρτήσεις.

```

1 function msgMatrix = createMatrix(msg, M, len)
2     i = 1;
3     j = 1;
4
5     for k = 1 : len
6         msgMatrix(i, j) = msg(k);
7         if (mod(k, M) == 0)
8             i = i + 1;
9             j = 0;
10        endif
11        j = j + 1;
12    endfor
13 endfunction

```

Κώδικας 5: Συνάρτηση δημιουργίας του πίνακα

Το παραπάνω απόσπασμα κώδικα 5 υλοποιεί μια απλή δημιουργία ενός πίνακα για την αποθήκευση του αρχικού μηνύματος σε "σπασμένα" μέρη.

Για να ελεγχθεί εάν ένα μήνυμα έχει φτάσει χωρίς σφάλμα στον δέκτη, δημιουργήθηκε η συνάρτηση validateParity όπως φαίνεται και παρακάτω:

```

1 function msg = compute2dParity(msg, N, M)
2     for i = 1 : N
3         countOnes = 0;
4         for j = 1 : M
5             if (msg(i, j))
6                 countOnes = countOnes + 1;
7             endif
8         endfor
9         if (mod(countOnes, 2) == 0)
10            msg(i, M + 1) = 0;
11        else
12            msg(i, M + 1) = 1;
13        endif
14    endfor
15
16    for i = 1 : M + 1
17        countOnes = 0;
18        for j = 1 : N
19            if (msg(j, i))
20                countOnes = countOnes + 1;
21            endif
22        endfor
23        if (mod(countOnes, 2) == 0)
24            msg(N + 1, i) = 0;
25        else
26            msg(N + 1, i) = 1;
27        endif
28    endfor
29 endfunction

```

Κώδικας 6: Υπολογισμός 2 διαστάσεων ισοτιμίας

Τέλος, οι παραπάνω συναρτήσεις (5 και 6) καλούνται στο αρχείο `exe3.m`.

```

1 msg = input("Give binary number (28-bit): ");
2
3 N = 4;
4 M = 7;
5 len = 28;
6
7 msgMatrix = zeros(N + 1, M + 1);
8
9 msgMatrix = createMatrix(msg, M, len);
10
11 msgMatrix = compute2dParity(msgMatrix, N, M);
12
13 disp(msgMatrix);

```

Κώδικας 7: Υλοποίηση δεύτερης άσκησης

Όπου για είσοδο 1100111101110101110010101001² επιστρέφεται:

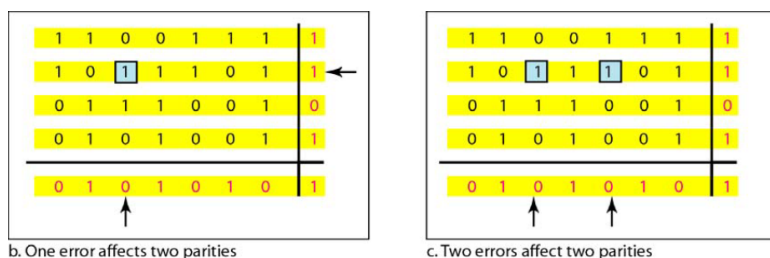
$$\left\{ \begin{array}{c} 11001111 \\ 10111011 \\ 01110010 \\ 01010011 \\ 01010101 \end{array} \right\}$$

Άσκηση 4

Εκφώνηση: Δημιουργείστε κώδικα που να ανιχνεύει σφάλματα στον πίνακα δύο διαστάσεων (αποτελούμενο από bits μηνύματος και bits ισοτιμίας) που παράγεται από τον κώδικα του 3ου σκέλους της άσκησης.

Τρέξτε τον κώδικά σας για το συγκεκριμένο μήνυμα του 3ου σκέλους της άσκησης και για τις εξής περιπτώσεις:

1. Αφίξη στον αποδέκτη χωρίς σφάλμα.
2. Ο πίνακας φτάνει με σφάλματα σε ένα bit (στο συγκεκριμένο bit που φαίνεται με γαλάζιο χρώμα στο σχήμα 2b).
3. Ο πίνακας φτάνει με σφάλμα σε δύο bits (στα συγκεκριμένα bits που φαίνονται με γαλάζιο χρώμα στο σχήμα 2c).



Σχήμα 2: Τέταρτη άσκηση

Απάντηση: Για τον εντοπισμό εάν έχει υποστεί κάποια αλλοίωσή το μήνυμα χρησιμοποιήθηκε η παρακάτω συνάρτηση:

²H εισαγωγή του μηνύματος στο περιβάλλον Octave πρέπει να γίνει με αυτή την μορφή: [1 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 1].


```
1 function isValid = validate2dParity (msg, N, M)
2 isValid1 = true; isValid2 = true;
3 for i = 1 : N + 1
4     countOnes = 0;
5     for j = 1 : M + 1
6         if (msg(i, j))
7             countOnes = countOnes + 1;
8         endif
9     endfor
10    if (mod(countOnes, 2) == 0 && isValid1)
11        isValid1 = true;
12    else
13        isValid1 = false;
14    endif
15 endfor
16 for i = 1 : M + 1
17     countOnes = 0;
18     for j = 1 : N + 1
19         if (msg(j, i))
20             countOnes = countOnes + 1;
21         endif
22     endfor
23    if (mod(countOnes, 2) == 0 && isValid2)
24        isValid2 = true;
25    else
26        isValid2 = false;
27    endif
28 endfor
29 isValid = isValid1 && isValid2;
30 endfunction
```

Κώδικας 8: Περιεχόμενο του αρχείου validate2dParity.m

Χρησιμοποιώντας τις συναρτήσεις 5, 6 και 8 υλοποιείται και η τέταρτη άσκηση (exe4.m).

```

1 msg = input("Give binary number (28-bit): ");
2
3 N = 4;
4 M = 7;
5 len = 28;
6
7 msgMatrix = zeros(N + 1, M + 1);
8
9 msgMatrix = createMatrix(msg, M, len);
10
11 msgMatrix = compute2dParity(msgMatrix, N, M);
12
13 msgMatrix = bitManipulation(msgMatrix, 2, 3);
14
15 if (validate2dParity(msgMatrix, N, M))
16     disp("Message is OK.");
17 else
18     disp("Message is corrupted.");
19 endif
20
21 msgMatrix = bitManipulation(msgMatrix, 2, 5);
22
23 if (validate2dParity(msgMatrix, N, M))
24     disp("Message is OK.");
25 else
26     disp("Message is corrupted.");
27 endif

```

Κώδικας 9: Υλοποίηση τέταρτης άσκησης

Και στις δυο περιπτώσεις (που προέκυψαν σφάλματα) ο αλγόριθμος -άρτιας- διδιάστατης ισοτιμίας εντοπίζει πως έχει υποστεί το μήνυμα κάποια αλλοίωση.

Άσκηση 5

Ερώτηση: Υπάρχουν κάποια πρότυπα σφάλματος που δεν ανιχνεύονται στη διδιάστατη ισοτιμία; Αν ναι, ποια είναι αυτά;

Απάντηση: Η διδιάστατη ισότητα φαίνεται να είναι ιδιαίτερα αποδοτική στον εντοπισμό διάσπαρτων σφαλμάτων. Ωστόσο, στην περίπτωσή όμως που σχηματιστεί ένα τετράγωνο από σφάλματα, όπως φαίνεται στο σχήμα 4, τότε αδυνατεί να εντοπίσει την αλλοίωση των δεδομένων.

$$\begin{bmatrix} 11001111 \\ 10111011 \\ 01110010 \\ 01010011 \\ 01010101 \end{bmatrix}$$

Σχήμα 3: Πριν την αλλοίωση

$$\begin{bmatrix} 11001111 \\ 10111011 \\ 01110010 \\ 01010011 \\ 01010101 \end{bmatrix}$$

Σχήμα 4: Αφού έχουν υποστεί "τετραγωνική" αλλοίωση

Έστω ότι στο παραπάνω σχήμα οι αριθμοί που βρίσκονται στους κύκλους αλλάζουν στην αντίθετη τιμή τους, τότε ο αλγόριθμος δεν θα είναι σε θέση να εντοπίσει κάποιο σφάλμα και θα θεωρήσει πως τα δεδομένα είναι ορθά.

Βιβλιογραφία

- [1] Stallings W., *Επικοινωνίες Υπολογιστών και Δεδομένων*, Εκδόσεις Τζιόλα, 2011.