Sprawozdanie

Technologie sieciowe – lista 3

Aleksandra Malinowska, WPPT INF, 4 semestr, kwiecień 2019

1. Ramkowanie

1.1. Program testujący

W zadaniu pierwszym należało napisać program do kodowania i dekodowania strumieni bitowych techniką rozpychania bitów oraz przetestować jego działanie. Napisany przeze mnie program kodujący pobiera strumień tekstowy z pliku, zamienia go z typu **String** na typ **byte**, oblicza wartość kodu CRC dla wiadomości, a następnie wykonuję operację rozpychania bitów i przesyła wynik do pliku. Program dekodujący odczytuje strumień z pliku, usuwa markery oraz kody „escape" następnie oddziela kod CRC od wiadomości i porównuje z wartością obliczoną dla otrzymanej wiadomości. W razie niepowodzenia rzuca wyjątek typu **DecodeException**. Poniżej przedstawiam kilka eksperymentów przeprowadzonych w programie.

1.2. Eksperymenty

1.3.1. Poprawne działanie programu

Main.java | Frame.java | DecodeException.java | FileDecoder.java

```
package com.tester;

import com.tester.framer.FileDecoder;

public class Main {
    public static void main(String[] args) {
        FileDecoder decoder = new FileDecoder();
        decoder.encodeFile( inputFileName: "input.txt",  outputFileName: "output.txt");
        decoder.decodeFile( inputFileName: "output.txt",  outputFileName: "result.txt");
    }
}
```

Main › main()

input.txt

```
0101010100000100101001001000111101001010101000001010110101010111111100101010111111111001
1111111111111111
0000000000000000
01111110
00000000
11
```

output.txt

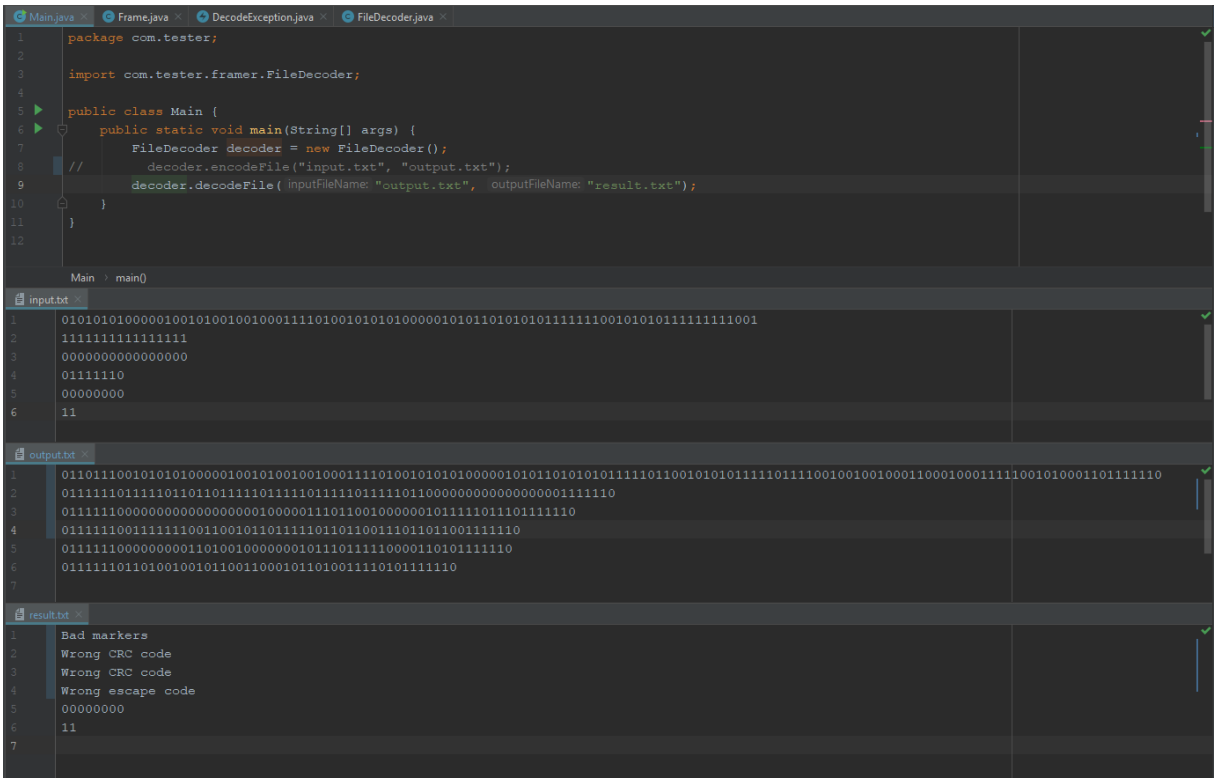
```
0111111001010101000001001010010010001111010010101010000010101101010101111101100101010111101111001001001000110001000111100101000110111111
0111111011111011111011111011111011111011111011000000000000000001111110
0111111000000000000000000100000111011001000100101111011101111110
0111111001111101001100101101111101101100111011011001111110
0111111000000000110100100000001011101111000011010111110
0111111011010010010110011000101101001111010111110
```

result.txt

```
0101010100000100101001001000111101001010101000001010110101010111111100101010111111111001
1111111111111111
0000000000000000
01111110
00000000
11
```

Powyżej program otrzymuje dane z pliku **input.txt**, koduje je do pliku **output.txt**, a następnie odkodowuje zawartość tego pliku do pliku **result.txt**. Poprawności programu dowodzi identyczność plików **input.txt** oraz **result.txt**.

1.3.2. Celowo zniszczone dane



```
Main.java | Frame.java | DecodeException.java | FileDecoder.java

package com.tester;

import com.tester.framer.FileDecoder;

public class Main {
    public static void main(String[] args) {
        FileDecoder decoder = new FileDecoder();
//          decoder.encodeFile("input.txt", "output.txt");
        decoder.decodeFile( inputFileName: "output.txt",  outputFileName: "result.txt");
    }
}

Main › main()
```

input.txt

```
01010101000001001010010001111010010101010000010101101010101111111001010101111111111001
1111111111111111
0000000000000000
01111110
00000000
11
```

output.txt

```
01101110010101010000010010100100100011110100101010100000101011010101011111011100100100011000100011111001010001101111110
01111110111110110110111110111110111111011111011000000000000000001111110
01111110000000000000000010000011101100100000010111110111011111110
01111110011111110011001011011111011011001110110110011111110
01111110000000001101001000000010111011111000011010111110
01111110110100100101100110001011010011110101111110
```

result.txt

```
Bad markers
Wrong CRC code
Wrong CRC code
Wrong escape code
00000000
11
```

' W tej próbie zmieniłam kilka zer na jedynki i kilka jedynek na zera w pliku **output.txt** oraz uruchomiłam tylko metodę do dekodowania pliku. Program w miejscu błędnych danych umieścił wiadomość o tym, co spowodowało błąd. Widać, że błędy w pierwszych wiadomościach nie miały wpływu na dalsze przetwarzanie pliku.

2. Symulacja sieci Ethernet

2.1. Program symulujący

W zadaniu 2 należało napisać program symulujący działanie sieci Ethernet. Miał on wysyłać wiadomości na łaczę i rozwiązywać konflikty. W mojej implementacji program wyświetla stan łącza po każdym interwale oraz na standardowym wyjściu błędów umieszcza informację o ewentualnych konfliktach i ich rozwiązaniu. Poniżej przedstawię przykładowe działanie programu.

2.2. Eksperymenty

2.2.1. Dwie stacje

2.2.1.1. Parametry symulacji

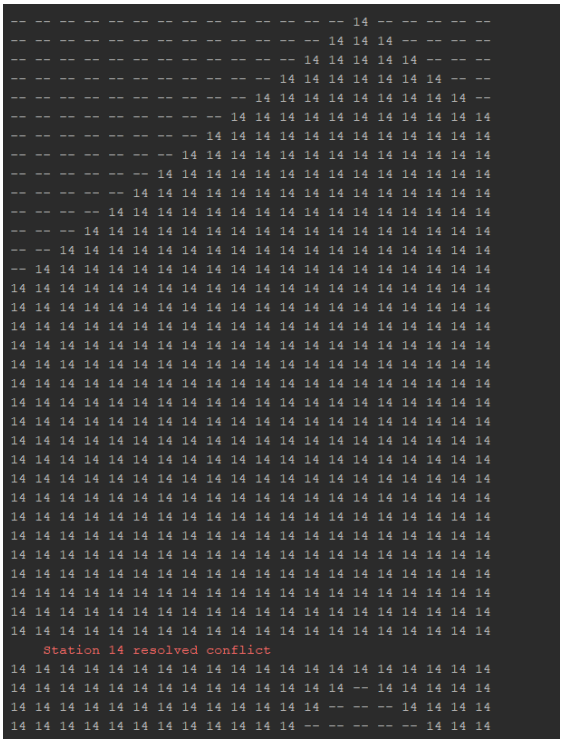
```
public class Main {

    public static void main(String[] args) {
        ConnectionSimulator simulator =
            new ConnectionSimulator( stationNumber: 2, interval: 50, wireLength: 20,  stationSendingProbability: 0.1);
        simulator.run();
    }
}
```

2.2.2. Przebieg

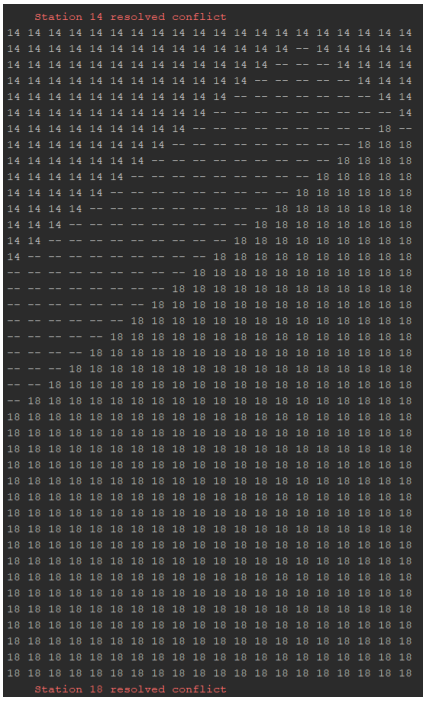
Po uruchomieniu stacje zaczynają losowo wysyłać sygnały.

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- -- 14 -- -- -- -- --
-- -- -- -- -- -- -- -- -- -- -- -- -- 14 14 14 -- -- 18 --
-- -- -- -- -- -- -- -- -- -- -- -- 14 14 14 14 14 18 18 18
-- -- -- -- -- -- -- -- -- -- -- 14 14 14 14 14  #  # 18 18
    Station 18 is conflicted for the 1 time. Waits: 80
-- -- -- -- -- -- -- -- -- -- 14 14 14 14 14  #  #  #  # 18
    Station 14 is conflicted for the 1 time. Waits: 40
-- -- -- -- -- -- -- -- -- 14 14 14 14 14  #  #  #  # 14  #
-- -- -- -- -- -- -- -- 14 14 14 14 14  # 18  #  # 14 14 14
-- -- -- -- -- -- -- 14 14 14 14 14  # 18 18 18 14 14 14 14
-- -- -- -- -- -- 14 14 14 14 14  # 18 18 18 -- -- 14 14 14
-- -- -- -- -- 14 14 14 14 14  # 18 18 18 -- -- -- -- 14 14
-- -- -- -- 14 14 14 14 14  # 18 18 18 -- -- -- -- -- -- 14
-- -- -- 14 14 14 14 14  # 18 18 18 -- -- -- -- -- -- -- --
-- -- 14 14 14 14 14  # 18 18 18 -- -- -- -- -- -- -- -- --
-- 14 14 14 14 14  # 18 18 18 -- -- -- -- -- -- -- -- -- --
14 14 14 14 14  # 18 18 18 -- -- -- -- -- -- -- -- -- -- --
14 14 14 14  # 18 18 18 -- -- -- -- -- -- -- -- -- -- -- --
14 14 14  # 18 18 18 -- -- -- -- -- -- -- -- -- -- -- -- --
14 14  # 18 18 18 -- -- -- -- -- -- -- -- -- -- -- -- -- --
14  # 18 18 18 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
 # 18 18 18 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18 18 18 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18 18 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
18 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Stacje wysłały sygnały niemal w tym samym momencie dlatego szybko nastąpił konflikt.

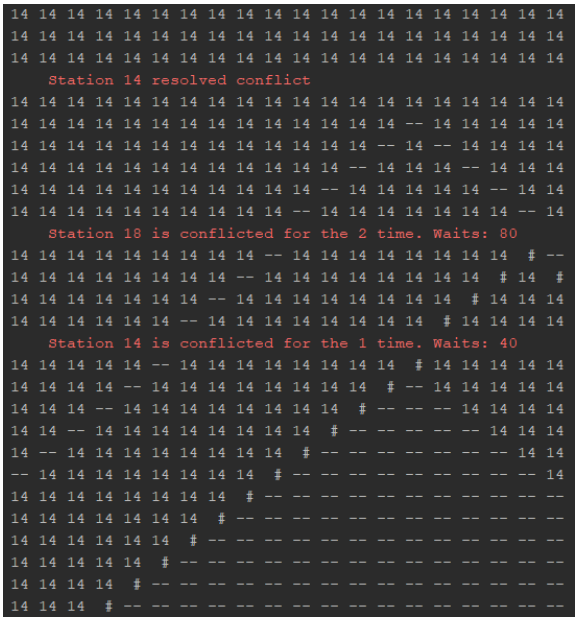


Po odczekaniu wyznaczonego czasu stacja nr 14 rozwiązała konflikt.



To samo udało się stacji nr 18.

Po kliku udanych próbach wysyłania sygnału, stacje znowu napotkały konflikt. Tym razem jednak stacja nr 14 rozwiązała konflikt od razu, ale stacja nr 18 rozpoczęła rozwiązywanie swojego konfliktu, gdy stacja 14 zaczęła rozsyłać kolejny komunikat.



Tym sposobem stacja 18 rozpoczęła swój drugi konflikt, a 14 pierwszy (dotyczy innej wiadomości).Stacje rozwiązały go przy pierwszym podejściu. W dalszej części doświadczenia występowały podobne sytuacje.

2.2.2. Więcej stacji

2.2.2.1. Średnia długość konfliktu

Dla większej liczby stacji policzyłam średnią długość trwania konfliktu. Przeprowadziłam dwa eksperymenty. Dla 5 stacji na łączu długości 50 (prawdopodobieństwo wysłania sygnału przez stację 0.2) średnia długość trwania konfliktu wyniosła **3.4**. Dla 20 stacji na łączu długości 100 (z prawdopodobieństwem 0.5) wynik to **5.55**.

2.3. Wnioski

Z powyższych testów wynika, że eksperymenty na większej liczbie stacji oraz dłuższym łączu skutkują znacznie częstszym występowaniem konfliktów. Nie są one również rozwiązywane po pierwszej próbie, lecz, zależnie od warunków początkowych, po kilku lub kilkunastu.