Código de Hamming

Rodrigo Fortes, Álisson Paixão

September 11, 2019

1 Enunciado

A classe do código de Hamming será composta pelas seguintes operações:

- Encode: Recebe o dado que será enviado e retorna o dado codificado para enviar.
  - pré-condição: A mensagem inicial deve conter bits de 1 até n, onde n $\in$ N.
  - pós-condição: A mensagem de saída deve ter uma quantidade de bits igual a mensagem de entrada com adição da fórmula de equação 1.
- Decode: Recebe o dado codificado, decodifica, detecta e arruma um único erro, retornando a mensagem decodificada e corrigida se necessário.
  - pré-condição: A mensagem recebida deve possuir uma quantidade de bits mínima de três bits, pois a mensagem recebida precisa estar codificada (pelo código de Hamming), e o tamanho mínimo de mensagem codificada é de 3 bits (mensagem de 1 bit com 2 bits de paridade).
  - pós-condição: A mensagem de saída deve ter uma quantidade de bits igual a equação 2.

2 Ferramenta

As ferramenta utilizada para a Implementação deste programa foi o Eclipse com o plugin de JBehave para realização de cenário.

3 Implementação

Para a implementação em Java foi criada uma classe chamada HammingCode, aonde há métodos para a codificação e para a decodificação de mensagens. Há também no nosso projeto a classe App, aonde há envios de mensagens para a classe HammingCode com o objetivo de testar o funcionamento desta, bem como, mostrar a corretura do código.

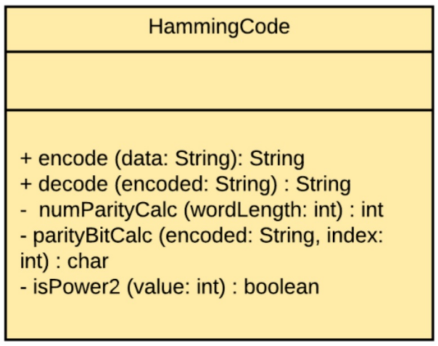

Figure 1: Diagrama de Classe

3.1 App

```
1  import java.util.Random;
2
3  public class App{
4
5          static public String flip(String data){
6
7                          Random r = new Random();
8
9                          int index = r.nextInt(data.length());
10
11                         char errBit = '0';
12
13                         if(data.charAt(index) == '0'){
14                                 errBit = '1';
15                         }
16
17                         String errData = "";
18
19                         for(int i = 0; i < data.length(); i++){
20                                 if(i == index){
21                                         errData += errBit;
22                                 }else{
23                                         errData += data.charAt(i);
24                                 }
25                         }
26
27                         return errData;
28         }
29
30         static public void main(String [] args){
31                 String msg = args[0];
32                 String encoded = HammingCode.encode(msg);
```

```
33                  String received = flip(encoded);
34                  String decoded = HammingCode.decode(received);
35                  System.out.println("Message length: " + msg.length());
36                  System.out.println("Encoded message length: " + encoded.length());
37                  System.out.println("Sending message: " + msg);
38                  System.out.println("Encoded message: " + encoded);
39                  System.out.println("Received message: " + received);
40                  System.out.println("Decoded message: " + decoded);
41          }
42  }
```

3.2 HammingCode

```
1   public class HammingCode{
2
3           static public String encode(String data){
4                   int numParityBits = numParityCalc(data.length());
5
6                   String encoded = "";
7
8                   int i = 1;
9                   int j = 1;
10
11                  while(i <= data.length() + numParityBits){
12                          if(isPower2(i)){
13                                  encoded += '0';
14                          }else{
15                                  encoded += data.charAt(j-1);
16                                  j++;
17                          }
18                          i++;
19                  }
20
21                  i = 1;
22                  j = 1;
23
24                  char [] encodedArray = encoded.toCharArray();
25
26                  while(i <= numParityBits){
27                          encodedArray[j-1] = parityBitCalc(encoded, j);
28                          j = j * 2;
29                          i++;
30                  }
31
32                  encoded = "";
33
34                  for(i = 0; i < encodedArray.length; i++){
35                          encoded += encodedArray[i];
36                  }
37
38                  return encoded;
39          }
40
41          static public String decode(String encoded){
42
```

```
43                  int numParityBits = (int) Math.floor(
44                          Math.log(encoded.length())/Math.log(2.0))+1;
45
46                  int i = 1;
47                  int j = 1;
48
49                  char [] encodedArray = encoded.toCharArray();
50
51                  int errPos = 0;
52
53                  String decoded = "";
54
55                  while(i <= numParityBits){
56                          if(encodedArray[j-1] != parityBitCalc(encoded, j)){
57                                  errPos += j;
58                          }
59
60                          j = j * 2;
61                          i++;
62                  }
63
64                  if(errPos != 0){
65                          char err = encodedArray[errPos-1];
66
67                          if(err == '1'){
68                                  encodedArray[errPos-1] = '0';
69                          }else{
70                                  encodedArray[errPos-1] = '1';
71                          }
72                  }
73
74                  for(i = 1; i <= encodedArray.length; i++){
75                          if(!isPower2(i)){
76                                  decoded += encodedArray[i-1];
77                          }
78                  }
79
80                  return decoded;
81          }
82
83          static private int numParityCalc(int wordLength){
84
85                  int numParity = 0;
86
87                  while(Math.pow(2,numParity) < wordLength + numParity + 1){
88                          numParity++;
89                  }
90
91                  return numParity;
92          }
93
94          static private char parityBitCalc(String encoded, int index){
95
96                  int count1 = 0;
97
98                  for(int i = 1; i <= encoded.length(); i++){
99                          if(!isPower2(i) && (i & index) != 0 &&
```

```
                                encoded.charAt(i-1) == '1'){
                                count1++;
                        }
                }

                if(count1 % 2 == 0){
                        return '0';
                }

                return '1';
        }

        static private boolean isPower2(int value){
                return (value & (value - 1)) == 0;
        }
}
```

3.3 Cenário

```
Narrative:
Find Error in Data Transmission with Hamming code
Given a Message if a error occurs while transmitting
Then The algorithm should correct that error


Scenario:  Error-Correction
Given the message 01010101
When fliping one of the bits in the message
Then the resulting message should be what was given (01010101)
```

4 Repositório Git-Hub

- https://github.com/Bregnets/BDD_Hamming