Alunos: Elias Fank; João Gehlen; Ricardo Zanuzzo

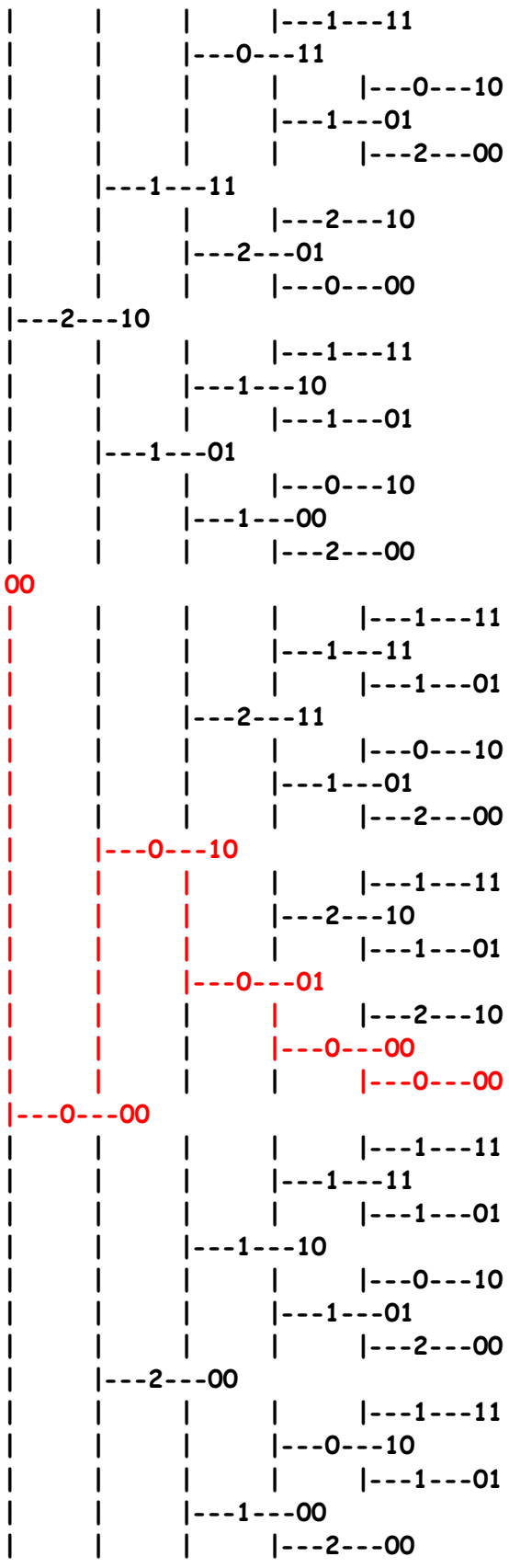
Descrição geral do algoritmo:

De forma o geral o algoritmo tem como principal objetivo decodificar uma mensagem que sofreu algum tipo de interferência - o que deixa a mensagem com algum ruído, alguns bits errados. A nossa implementação do algoritmo de viterbi funciona basicamente da seguinte forma:

Primeiro passo é definir uma sequência de bits (a entrada do algoritmo) que vai ser a mensagem, depois adiciona-se os dois bits extras na mensagem e então codifica-se essa mensagem, gerando uma sequência de bits para ser “enviada” e posteriormente decodificada. Para simular o envio da mensagem e o ruído que ela terá, nós utilizamos um ruído do tipo uniforme. A quantidade de ruído é definido na variável **ruído** e deve ser informado em porcentagem, por exemplo, 0.05 para 5%.

Depois de incluído o ruído o algoritmo faz a decodificação utilizando uma árvore binária. Essa árvore é gerada de acordo com a tabela de transições de estados, onde para cada estado(nó da árvore) você tem duas possibilidades para seguir: 1 - supondo que o próximo bit seja 0; 2 - supondo que o próximo bit seja 1. Como essa abordagem é exponencial, ou seja a árvore teria tamanho 2^n, foi realizada a operação de poda em cada nível da árvore. Essa poda consiste em deixar apenas os 8 melhores (os de menor distância até a raiz, a distância de cada nó até a raiz foi sendo armazenada em cada respectivo nó) nós em cada nível, isso faz com que o desempenho do algoritmo melhore bastante mesmo sem perder em encontrar o melhor caminho final. Por fim, o algoritmo “olha” para todas as folhas da árvore (no máximo 8) pega a menor e faz o caminho inverso até a raiz. Nesse ponto apenas foram feitas as transições pela tabela de estado para saber qual foi o bit, 0 ou 1, que originou cada estado. Depois de fazer a decodificação da mensagem com ruído, o algoritmo faz a comparação com a mensagem original e calcula quantos bits diferentes possui. No final mostra a diferença entra a mensagem original e a mensagem decodificada.

Na página a seguir temos um exemplo da árvore gerada na decodificação para os seguintes dados: entrada: 01000 -> entrada codificada: 0011101100



Aqui então a folha com o somatório de menor erro até a raiz é a folha destacada em vermelho. Fazendo agora o caminho inverso até a raiz obtemos: 00 00 01 10 00 00.

E agora pela tabela de transições, os bits que geraram essa sequência foram: 01000.

Assim encontramos a sequência original de bits.

Descrição dos problemas e soluções usadas:

- Solução para **codificação**: Na codificação nenhum problema muito relevante foi encontrado, apenas seguimos as regras de codificação disponibilizada pelo professor na descrição do trabalho. O ruído também não teve nenhuma questão muito complexa, já que implementamos o ruído mais simples, em que aleatoriamente trocamos alguns bits de acordo com a quantidade de ruído informada na entrada.
- Para a **decodificação** representamos o algoritmo de Viterbi utilizando uma árvore binária com poda, deixando apenas no máximo 8 nós por nível, como pode ser observado na página anterior;
- Um **problema** que tivemos foi que no início não lembramos de fazer a poda, sendo assim o programa demorava muito para executar com uma entrada de 50 ou mais bits. Depois de implementada a poda, esse problema foi solucionado.

Exemplos de codificação/decodificação alcançados pelo programa:

1. Um exemplo de resultado alcançado pelo algoritmo para validar o funcionamento:

Para a entrada foi utilizado a seguinte sequência de bits:

011011100011110

Essa entrada foi codificada e gerou a seguinte sequência de bits:

00110101000110011100110110100111100

Com um ruído de 7%, foram trocados 2 bits, o que gerou a seguinte sequência:

00110101000110111110110110100111100

Essa sequência foi decodificado, o que gerou a seguinte sequência:

011011100011110

Comparando com a entrada:

011011100011110 (entrada)

011011100011110 (saída)

Conclui-se que não teve diferença entra a mensagem decodificada e a mensagem original.

2. Outro exemplo de saída do algoritmo:

Para a entrada foi utilizado a seguinte sequência de bits:

01001110010110101101110111011111100

Essa entrada foi codificada e gerou a seguinte sequência de bits:

00111011110110011111100001010010000101000110010001101001000110101010100111

Com um ruído de 20%, foram trocados 7 bits, o que gerou a seguinte sequência:

00001011010110111011100001000010000101000110110001101001000110101010100111

Essa sequência foi decodificado, o que gerou a seguinte sequência:

00011001000010101101110111011111100

Comparando com a entrada:

01001110010110101101110111011111100 (entrada)

00011001000010101101110111011111100 (saída)

Conclui-se que teve uma diferença de apenas 7 bits

- Mais um exemplo com um entrada um pouco maior:

Para a entrada foi utilizado a seguinte sequência de bits:

11100111110001011010010010101011010110010100111110111111111111011001010000100100010010001001011001110011011001000010001101001101101010111100000011010101111011111100111011111101011001111101011011000110100111010100010101111111000

Essa entrada foi codificada e gerou a seguinte sequência de bits:

11011001111101101010011100111000010100101111101111100010001000010100100001011111100010111101101010010001101010101010101010100100010111111000101100001110111111011001110111111011001110111110000101111110110011111010100010101111111011000011101100110101001011110101000101001000100001101001111000000001101010010001000011010010001101010100111110110010001100100001011111011010100100100001010001011100110101001011110111001001000101100111000100001101010100111000

Com um ruído de 0.05 (5 %), foram trocados 23 bits, o que gerou a seguinte sequência:

1111100111110110101001110011100001011010111110111010001000100001010
0101101011111000011111101101010010001101010111010101010100100010111
1110001011000011101111101100111011111011000110111010000101111101100
1111101010000011111101100001110110011010100101110010000010101100010
0001101001110000000011010100100010000110100100011001101001111101100
1000110101010110010000101111101101010010010000101000100110010010100
1111110110010011011011001110001000010010101001110000

Essa sequência foi decodificado, o que gerou a seguinte sequência:

1110011111000101101001001010101101011001010011111011111111111101100
1010000100100010010001001011001110011011001000010001101001101101010
1111000000110101011110111111001110111111010110011111010110110001101
0011101010001010111111000 0

Comparando a entrada com a saída:

1110011111000101101001001010101101011001010011111011111111111101100
1010000100100010010001001011001110011011001000010001101001101101010
1111000000110101011110111111001110111111010110011111010110110001101
0011101010001010111111000 0 (entrada)
1110011111000101101001001010101101011001010011111011111111111101100
1010000100100010010001001011001110011011001000010001101001101101010
1111000000110101011110111111001110111111010110011111010110110001101
0011101010001010111111000 0 (saída)

Neste exemplo, com 5% de ruído, conclui-se que não teve diferença entra a mensagem decodificada e a mensagem original.

Como compilar e executar o programa :

Por Makefile:
$ make all
$ make run <qtd_bits> <ruido>

Onde <qtd_bits> deve ser informado a quantidade de bits para gerar a sequência aleatória. E <ruido> deve ser informado a porcentagem de ruído, por exemplo 5 para 5% de ruído.

Exemplo de execução:

$ make all
$ make run 100 5

Nessa execução vai gerar uma sequência de 100 bits na entrada e vai usar um ruído de 5%