Error Correction

CSE1010 Project 8, Fall 2012

Date: 12/1/2012

Name: William Dickson

Section: 009L

TA: Levon Nazaryan

Instructor: Jeffrey A. Meunier

1. **Introduction**

In this project you will use arrays of numbers to create groups of bits (binary digits) that

would be sent over a simulated communication channel from one computer to another.

The sender uses parity bits to encode extra information in the data that allows the receiver

to detect and correct errors in the transmitted bits. The simulated communications channel

will be a function that with low probability flips bits randomly.

1. **Inputs & outputs**

The user is expected to supply the character string for the program. The program will take this character string and convert it into binary, store it in a 3x3 matrix and simulate a transmission of the matrix. Before transmission, the program adds an extra row and column to the matrix to make the number of 1s in each row and column even (creates parity). After transmission the program detects how many 1s have changed by looking at the parity and then corrects any value it is able to detect the location of (by using the row and column of the errorous bit). It then will return the converted final string back from binary.

1. **Sample Run**

>> p8

Enter a string: Hello, World! Nice day?

number of bits flipped: 4

frame 1 could not be repaired

frame 2 is correct

frame 3 has been repaired

recieved string =

Hehlo, World! Nice day?

frames containing errors:

1

>> p8

Enter a string: Hello, World! Nice day?

number of bits flipped: 2

frame 1 has been repaired

frame 2 has been repaired

frame 3 is correct

recieved string =

Hello, World! Nice day?

frames containing errors:

1. **Source Code**

% Error Correction

% CSE1010 Project 8, Fall 2012

% Date: 12/1/2012

% Name: William Dickson

% Section: 009L

% TA: Levon Nazaryan

% Instructor: Jeffrey A. Meunier

clc % clear command window

clear % clear all variables

ERROR\_PROB = .01;

DESIRED\_PARITY = 0;

reseedRand

string = input('Enter a string: ', 's');

blocks = string2blocks(string);

txframes = appendParityToBlocks(blocks,DESIRED\_PARITY);

rxframes = transmitFrames(txframes,ERROR\_PROB);

[repframes errFrames] = repairFrames(rxframes,DESIRED\_PARITY);

repblocks = stripFrames(repframes);

rxstring = blocks2string(repblocks);

disp('recieved string = ')

disp(rxstring)

disp('frames containing errors:')

disp(errFrames)

function [b,n] = addNoise(b,prob)

% Given a vector of numbers and a probability of error (on each element of

% the vector) this function will randomly flip a 0 to a 1 or vise versa at

% the given probability.

% Use: addNoise(b,prob)

n = 0;

for i=1:length(b)

if rand < prob

if b(i) == 0

b(i) = 1;

else

b(i) = 0;

end

n = n + 1;

end

end

end

function b = appendParity(b,desired\_par)

% Given a vector of numbers and a parity value (0 or 1) this function will

% add a value to the end of the vector to make the vector match the parity

% Use: appendParity(b,parity\_bit)

b(9) = parityOf(b,desired\_par);

end

function b\_mat\_pc = appendParityColumn(b\_mat,desired\_par)

% Given a matrix of numbers and a parity value (0 or 1) this function will

% add a value to the end of each column of the matrix to make it match the parity

% Use: appendParityColumn(b\_mat,desired\_par)

for n = 1:size(b\_mat,1)

b\_mat\_pc(n,:) = appendParity(b\_mat(n,:),desired\_par);

end

end

function b\_mat\_pr = appendParityRow(b\_mat,desired\_par)

% Given a matrix of numbers and a parity value (0 or 1) this function will

% add a value to the end of each row of the matrix to make it match the parity

% Use: appendParityRow(b\_mat,desired\_par)

b\_mat\_pr = appendParityColumn(b\_mat',desired\_par);

b\_mat\_pr = b\_mat\_pr';

end

function b\_mat\_pb = appendParityToBlock(b\_mat,desired\_par)

% Given a matrix of numbers and a parity value (0 or 1) this function will

% add a value to the end of each row and column of the matrix to make it match the parity

% Use: appendParityToBlock(b\_mat,desired\_par)

b\_mat\_pc = appendParityColumn(b\_mat,desired\_par);

b\_mat\_pb = appendParityRow(b\_mat\_pc,desired\_par);

end

function b\_mat\_p = appendParityToBlocks(b\_mat,desired\_par)

% Given a 3-D matrix of numbers and a parity value (0 or 1) this function will

% add a value to the end of each row and column of each 2-D matrix to make it match the parity

% Use: appendParityToBlocks(b\_mat,desired\_par)

for n = 1:size(b\_mat,3)

b\_mat\_p(:,:,n) = appendParityToBlock(b\_mat(:,:,n),desired\_par);

end

end

function c = bin2char(b)

% Given a vector of binary numbers this will convert the vector to the

% corresponding ASCII character defined by them.

% Use: bin2char(b)

b\_str = char(b+48);

c = char(bin2dec(b\_str));

end

function string = bin2string(block)

% Given a block of binary numbers this will convert the block to the

% corresponding characters (in a string) defined by them.

% Use: bin2string(block)

numRows = size(block, 1);

string = '';

for i=1:numRows

row = block(i, :);

chr = bin2char(row);

if chr == 0

break;

end

string = [string chr]; %#ok<AGROW>

end

end

function string = blocks2string(blocks)

% Given a 3-D block of binary numbers this will convert the block to the

% corresponding characters (in a string) defined by them.

% Use: bin2string(block)

string = '';

for n = 1:size(blocks,3)

string = [string,bin2string(blocks(:,:,n))];

end

end

function b = char2bin(c)

% Given a character this will convert the vector to the

% corresponding ASCII number and then the binary number that it corresponds to.

% Use: char2bin(c)

bin\_str = dec2bin(c);

bin\_str\_long = sprintf('%08s',bin\_str);

b\_vec = bin\_str\_long - 48;

b = logical(b\_vec);

end

function result = checkParity(bits, parity)

% This function will take a group of bits and if the parity of the bits in

% positions 1-8 are equal the parity given by the ParityOf function, then

% the result returned is true, otherwise false.

% Use: checkParity(bits, parity)

if bits(9) == parityOf(bits(1:8), parity)

result = 1;

else

result = 0;

end

function [r c] = checkParityOfFrame(matrix,parity)

% This function will take a matrix of bits and compare the parity of each

% row and column to that of the desired parity. It will then find the

% locations in the matrix of the flipped bits and return the locations (row

% and column vectors) of the errors.

% Use: checkParityOfFrame(matrix,parity)

newmat = appendParityToBlock(matrix(1:8,1:8), parity);

calc\_par\_c = newmat(1:8,9);

calc\_par\_r = newmat(9,1:9);

rec\_par\_c = matrix(1:8,9);

rec\_par\_r = matrix(9,1:9);

r = [];

c = [];

for n=1:8

if calc\_par\_c(n) ~= rec\_par\_c(n)

r(length(r)+1) = n;

end

end

for n=1:9

if calc\_par\_r(n) ~= rec\_par\_r(n)

c(length(c)+1) = n;

end

end

end

function parityPit = parityOf(bits,parity)

% This function determines the parity of a vector and if it is equal to the

% desired parity then it sets the 'parityPit' equal to 0, otherwise it

% sets it eqaul to 1.

% Use: parityOf(bits,parity)

a = sum(bits) + parity;

if rem(a, 2) == 0

parityPit = 0;

else

parityPit = 1;

end

end

function [r\_frame,status] = repairFrame(matrix,r,c)

% This function takes a frame (9x9), a vector of error rows, and a vector of error columns,

% and tries to repair the frame.

% Use: repairFrame(matrix,r,c)

if isempty(r) || isempty(c)

status = 1;

r\_frame = matrix;

elseif length(r) == 1 && length(c) == 2

status = 2;

matrix(r(1),c(1)) = 1 - matrix(r(1),c(1));

r\_frame = matrix;

else

status = 0;

r\_frame = matrix;

end

end

function [repairedFrames errorFrames] = repairFrames(frames, desiredParity)

% This function uses the repairFrame function to repair the frames of a 3-D

% matrix. It will determine whether or not the frame can be repaired. This

% function will also output the number of each frame that cannot be

% repaired and will display a statement for each frame describing the

% result of each frame.

% Use: repairFrames(frames, desiredParity)

repairedFrames = [];

errorFrames = [];

numFrames = size(frames, 3);

for i = 1:numFrames

frame = frames(:, :, i);

[errRows errCols] = checkParityOfFrame(frame, desiredParity);

[repairedFrame repairStatus] = repairFrame(frame, errRows, errCols);

switch repairStatus

case 0

fprintf('frame %g could not be repaired\n', i);

errorFrames(end+1) = i; %#ok<AGROW>

case 1

fprintf('frame %g is correct\n', i);

case 2

fprintf('frame %g has been repaired\n', i);

end

repairedFrames(:, :, i) = repairedFrame; %#ok<AGROW>

end

end

function reseedRand

% This function resets the random number generator to make a 'more' random

% number.

% Use: reseedRand

rng('shuffle')

end

function segments = segmentString(s)

% This function takes any string and returns an n x 8 array of characters.

% Use: segmentString(s)

len = length(s);

remainder = rem(len, 8);

if(remainder ~= 0)

newLen = len + 8 - remainder;

s(newLen) = 0;

else

newLen = len;

end

% Notice that here I'm using 8 as the number of rows and

% newLen/8 as the number of columns. This builds the segments

% as a group of columns. Then I transpose the columns into rows

% using the apostrophe.

segments = reshape(s, 8, newLen/8)';

end

function b\_mat = string2bin(s)

% This function takes a string and converts it to a binary vector

% Use: string2bin(s)

for n=1:length(s)

b\_mat(n,:) = char2bin(s(n));

end

end

function block = string2blocks(seg\_s)

% This function takes a string of any length and cuts it into 8-character

% segments using the segmentString function, then calls the string2bin

% function on each segment.

% Use: string2blocks(seg\_s)

b = string2bin(seg\_s);

b(ceil(size(b,1)/8)\*8,8) = 0;

for n = 1:ceil(size(b,1)/8)

block(:,:,n) = b(1+(n-1)\*8:8\*n,:);

end

end

function blocks = stripFrames(frames)

% This function takes a 3-d matrix of 9x9 frames and removes the parity row

% and column, returning a 3-d matrix of 8x8 payload bits.

% Use: stripFrames(frames)

blocks = frames(1:8, 1:8, :);

end

function trans\_frames = transmitFrames(frames,error\_prob)

% This function takes a 3-d matrix of bits (the frames) and an error

% probability to simulate a transmission of the frame with a error\_prob

% probability of flipping each bit.

% Use: transmitFrames(frames,error\_prob)

c = 0;

count = 0;

for n = 1:size(frames,3)

for m = 1:size(frames,2)

count = c + count;

[trans\_frames(m,:,n) c] = addNoise(frames(m,:,n),error\_prob);

end

end

fprintf('number of bits flipped: %0.f \n',count)

end