CMPE 310

Andrew Omer

3/6/2025

Hamming Distance Report

How did you come up with this code?

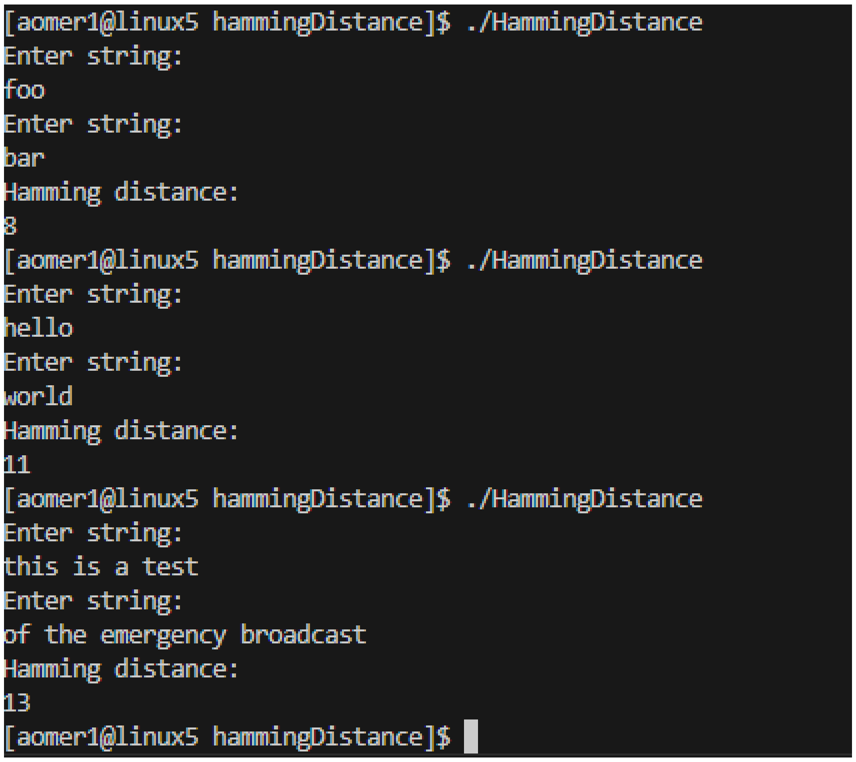
In order to compute the hamming distance between two strings, I started by requesting for the user to input a string two times. We take the user input and store each one in a register, if the string is too big it can be split between two registers. Store them as binary values. Perform an exclusive or operation on the two binary values and store the result. Loop through each result bit and increment the counter if bit value is 1, this sum will be the hamming distance. This integer must be converted to it's ascii value then finally outputted.

The Code

```
section .data
    newLine db 0xA ;New line

    outputString db 'Hamming distance: ', 0x0A ;Final Display
    len1 equ $-outputString

    request db 'Enter string: ', 0x0A      ;prompting for first string
    len2 equ $-request

section .bss
    string1 resd 10
    string2 resd 10
    distance resb 10  ;Reserve space for distance to be stored

section .text
    global _start

_start:

    ;Request string1
    mov eax, 4
    mov ebx, 1
    mov ecx, request
    mov edx, len2
    int 0x80
    ;Read and store string1
    mov eax, 3
    mov ebx, 0
    lea ecx, string1
    mov edx, 100
    int 0x80

    ;Request string2
    mov eax, 4
    mov ebx, 1
    mov ecx, request
    mov edx, len2
    int 0x80

    ;Read and store string2
    mov eax, 3
    mov ebx, 0
    lea ecx, string2
    mov edx, 100
    int 0x80

    ;Put binary value of strings into registers
    mov eax, [string1]
    mov ebx, [string2]

    ;Compare bit values
    xor eax, ebx

    ;Sum the 1's in EAX
    mov ecx, 0 ;Hamming distance counter

.sum_ones:
    test eax, 1 ;Check if 1
    jz .no_bit ;Skip when 0
    inc ecx ;Increment counter

.no_bit:
    shr eax, 1 ;Shift to next bit
    jnz .sum_ones ;Repeat for all bits


    mov eax, ecx ;Move hamming distance into eax
    mov ebx, distance ;Put buffer into ebx
    add ebx, 10 ;Move to end of the buffer
    mov byte [ebx], 0 ;Add the end to buffer
    dec ebx ;Point to the end
.convert_to_ascii:
    xor edx, edx ;Clear
    mov ecx, 10 ;Set divisor to 10
    div ecx ;Divide eax by 10 and put remainder in EDX
    add dl, '0' ;Convert remainder to ASCII
    mov [ebx], dl ;Store the digit
    dec ebx ;Next character
    test eax, eax ;Check if 0
    jnz .convert_to_ascii ;Continue if not 0

    ;Print outputString
    mov eax, 4
    mov ebx, 1
    lea ecx, outputString
    mov edx, len1
    int 0x80

    ;Print hamming distance
    mov eax, 4
    mov ebx, 1
    lea ecx, [distance]
    mov edx, 10
    int 0x80

    ;Print newLine
    mov eax, 4
    mov ebx, 1
    lea ecx, [newLine]
    mov edx, 1
    int 0x80

    ;Exit
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Output

As seen below, the code does compile and works as intended for “foo” and “bar”, however when using longer string inputs it does not function properly.



```
[aomer1@linux5 hammingDistance]$ ./HammingDistance
Enter string:
foo
Enter string:
bar
Hamming distance:
8
[aomer1@linux5 hammingDistance]$ ./HammingDistance
Enter string:
hello
Enter string:
world
Hamming distance:
11
[aomer1@linux5 hammingDistance]$ ./HammingDistance
Enter string:
this is a test
Enter string:
of the emergency broadcast
Hamming distance:
13
[aomer1@linux5 hammingDistance]$
```

Debugging Issues

This version of the code does not include my attempts to split the strings into multiple registers as I was unable to get it to function without producing errors. This the most likely reason why the other cases did not produce the correct result, but there are other possibilities. One of which being the way which I reserved space for the strings in .bss, and the other being the looping/incrementing through bits not behaving as intended.