Task 1

Question 1

- If the length of your prefix file is not multiple of 64, what is going to happen?
- I set the size of `prefix.txt` at 164 which is not a multiple of 64. The `md5sum` command shows no difference between the two executeable since it's padded with zeroes.

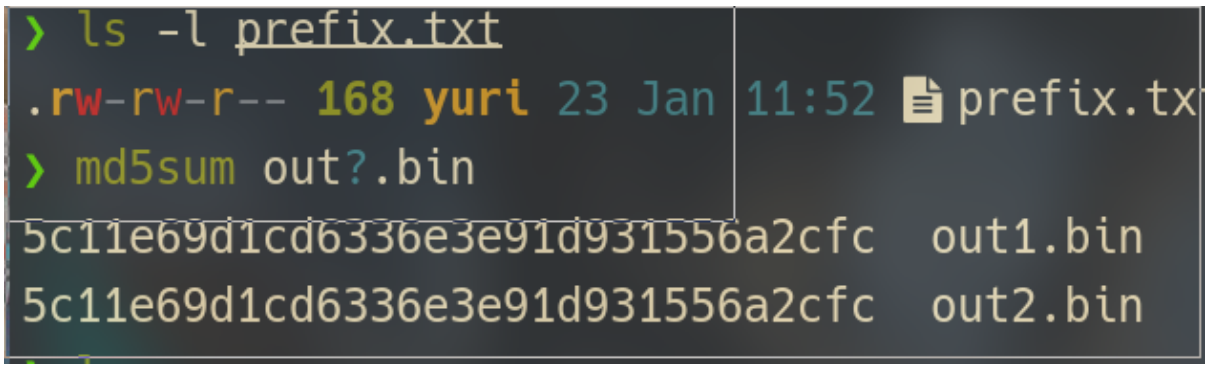

Figure 1: No Difference

- The `diff` command show difference in binary form.

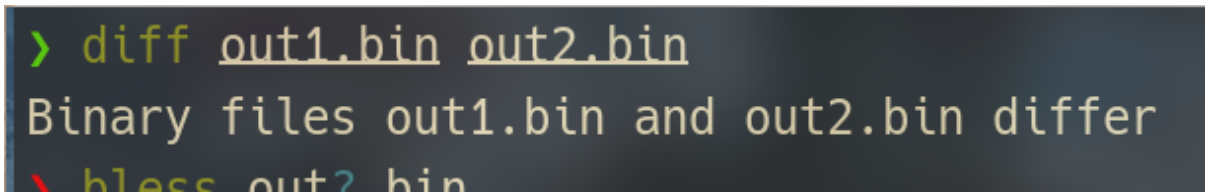

Figure 2: Difference in Binary

Question 2

- Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

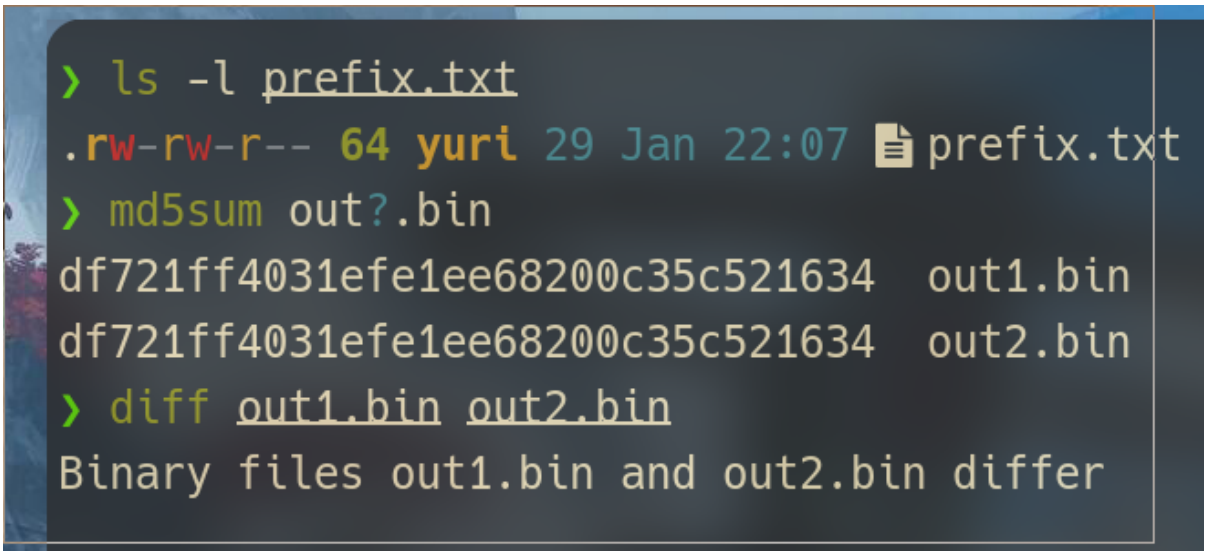

Figure 3: 64 Bytes input

- As seen above, the two files have the same MD5 sum. However, they are two difference files as showed using the `diff` command.

Question 3

- The two files generated by `md5collgen` have some similarities, especially in the beginning. However, they are still different.

Figure 4: Out1.bin

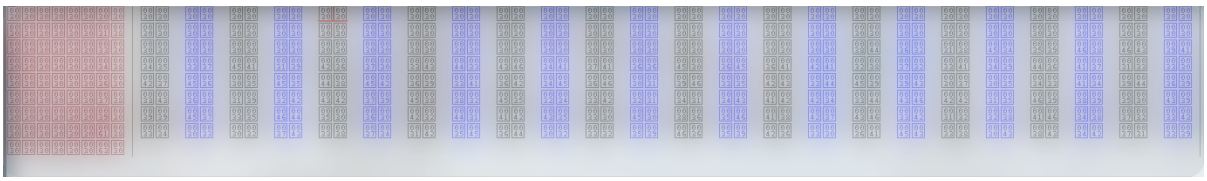

Figure 5: Out2.bin

(Sorry if the screenshot is bluried)

Task 2

- From the previous task, we know that the two files Out1 and Out2 have the same hash. In this case, `MD5{Out1} = MD5{Out2}`.
- I created a text file that contains a word. I concatenated this file with Out1 and Out2 respectively and verified that the MD5 hash results of the two files are the same.

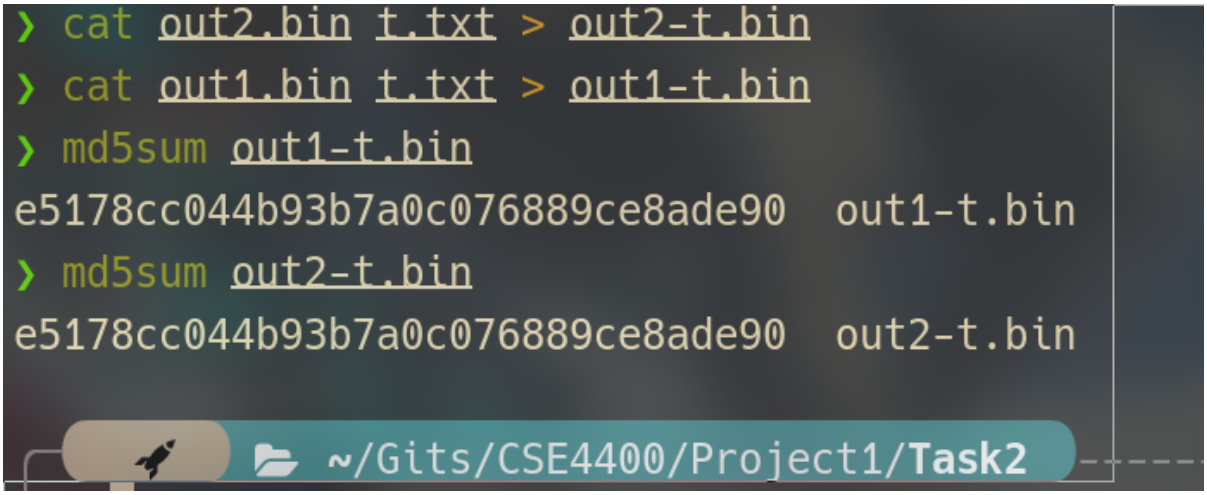

Figure 6: Same output

Task 3

- Using the code in the instruction, I filled the array with 200 character 'A' and using the command `xxd -g1 task3` to display the binary values.

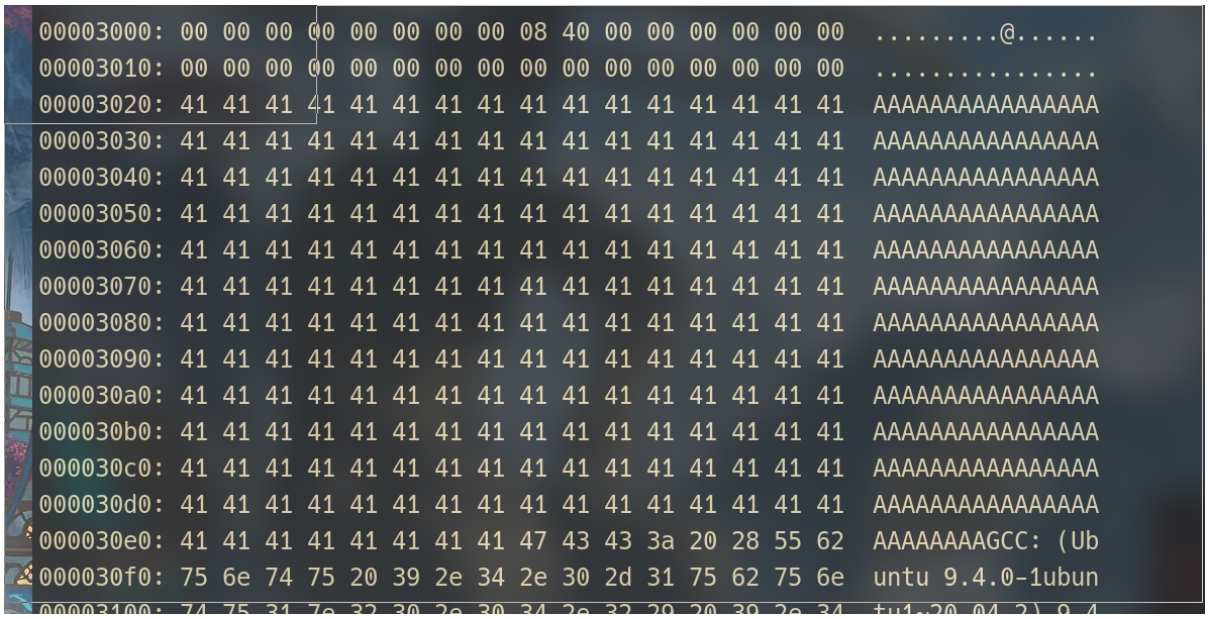

Figure 7: A's location

- We can see that the array starts from `3020` in hex which is `12320` in decimal, but this value is not a multiple of 64. The nearest value which is the multiple of 64 is `12352`. This is our prefix.
- I Run the command `head -c 12352 task3 > prefix`. Use this `prefix` to as input for md5collgen. As a result, I created 2 binary files which have the same md5 hash.

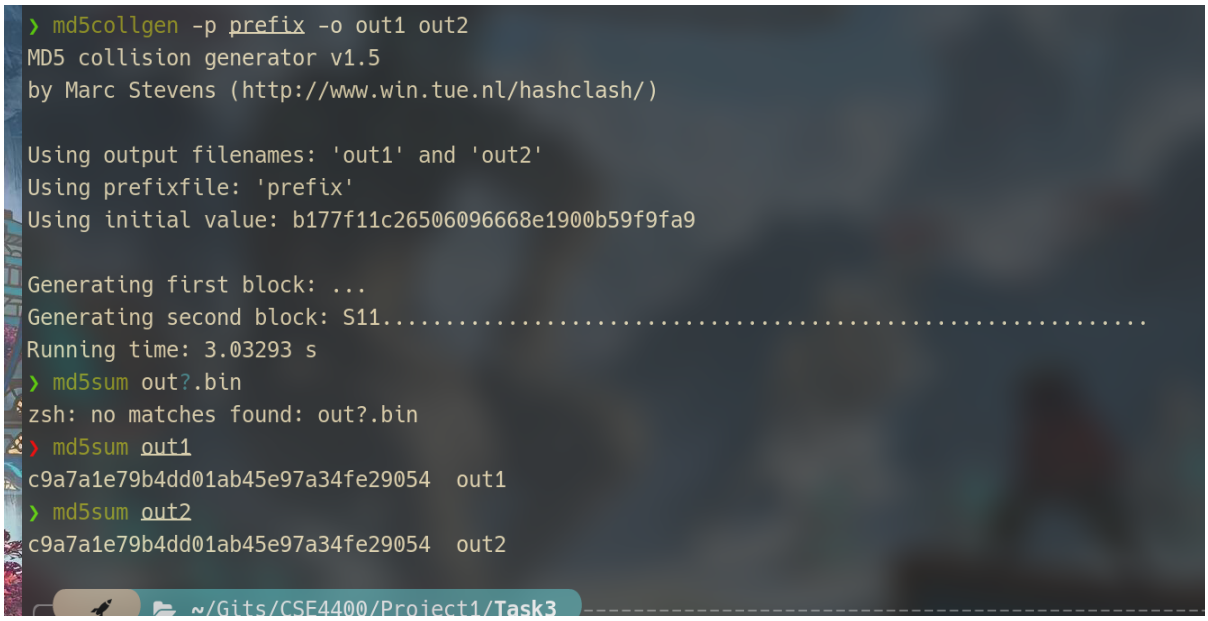

Figure 8: Collision

- Now We can create P and Q using out1 and out2 last 128 bytes. To find the start of the suffix region, we add the value where prefix's end with 128. This yields 12480.

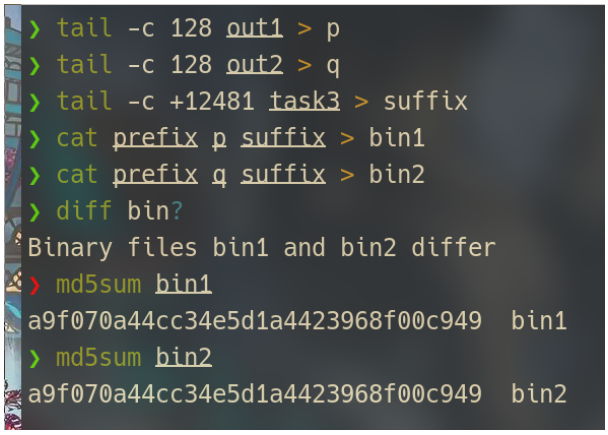

Figure 9: Different Binary but same Hash

Task 4

- Like what we did in task 3, now I created 2 arrays that filled with As and find which binary region they resided on.

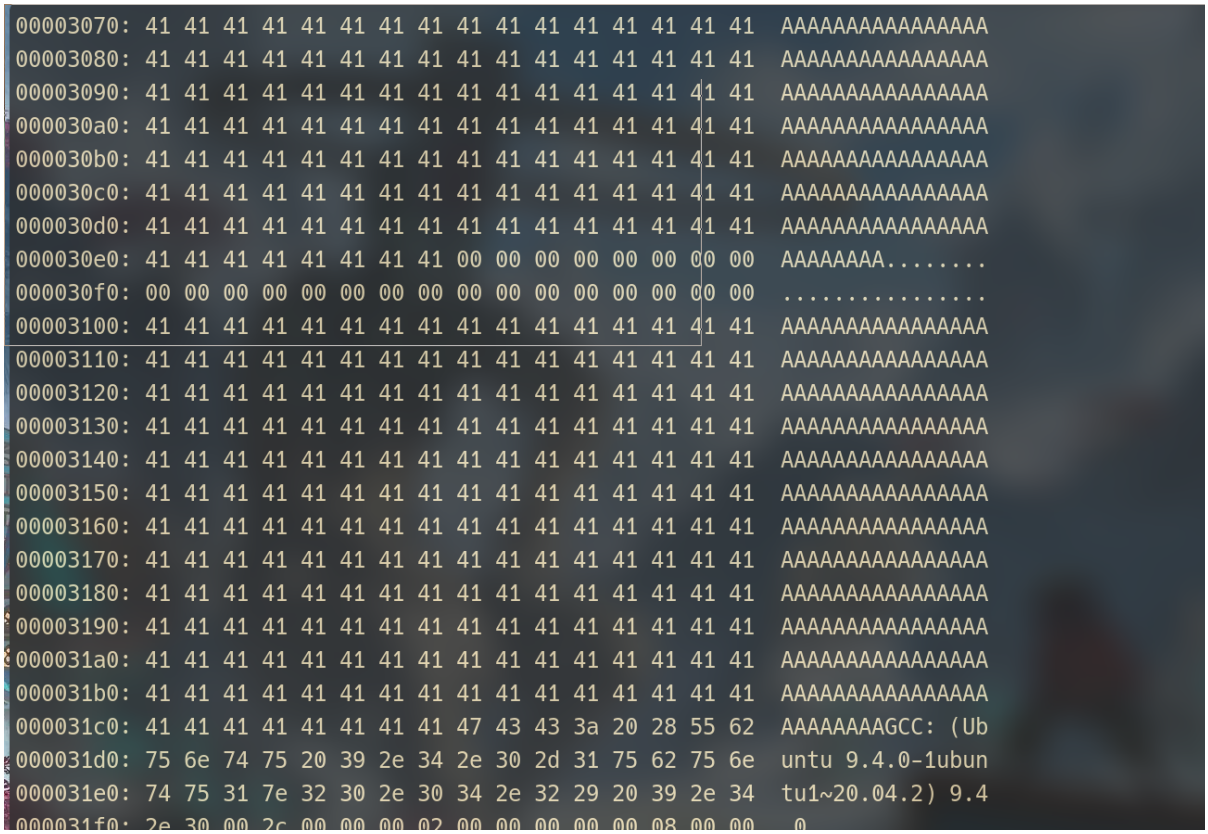

Figure 10: A's location for 2 arrays

- Notice that the prefix is still the same which is `12352`. Therefore, we have the same prefix as we did in part 3. Creating p and q is like what we did in task 3 which take the last 128 bytes of out1 and out2: `tail -c 128 out1.bin > p`. Same for the Suffix.

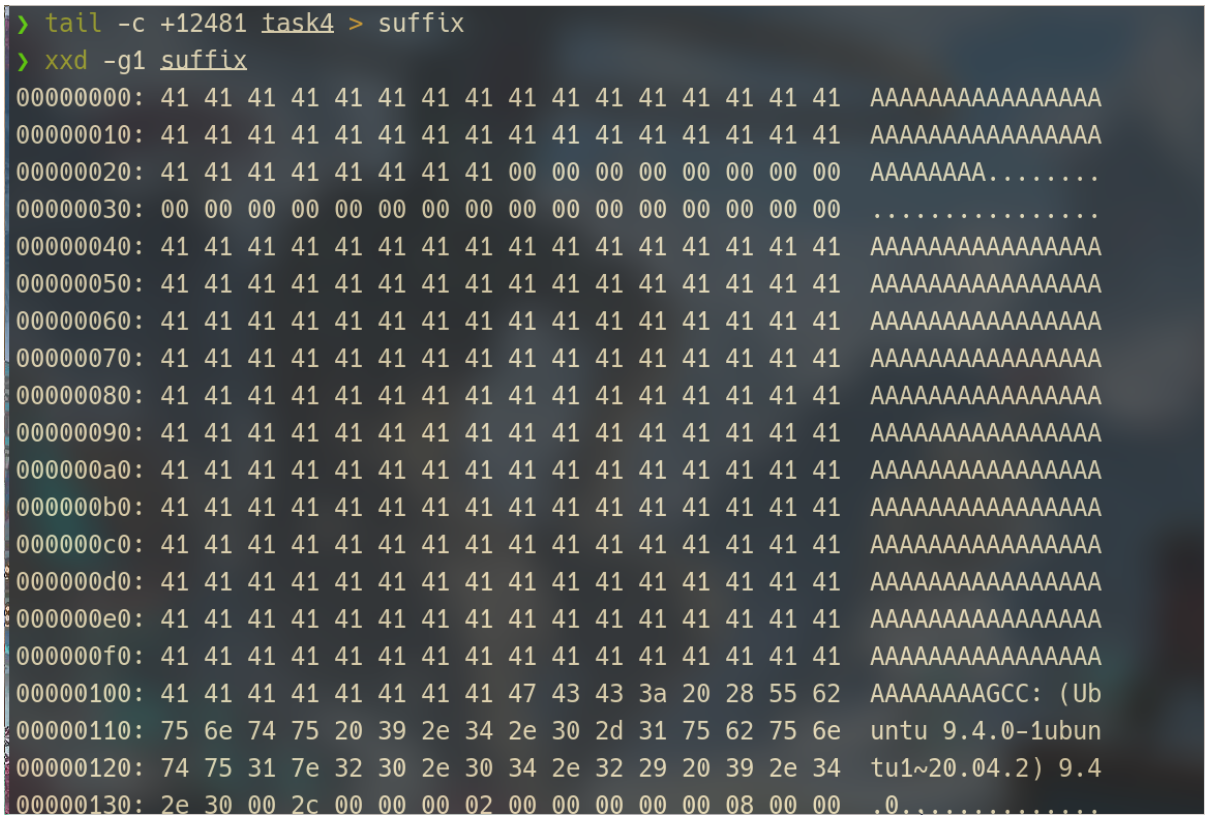

Figure 11: Second array's Location

- Notice from the picture above, we can see the second array starts at 0x0040 which is 64 in decimal. We need to add another 32 A's until we reach the start of the second P. Name this region `buffer_1` which has length of 96
- P's region take 128 bytes. Therefore, we add 128 to `buffer_1`'s region to obtain 224 bytes which is the rest of the region. Name it `buffer_2`.
- We put everything together using the `Cat` command as shown in the picture below:

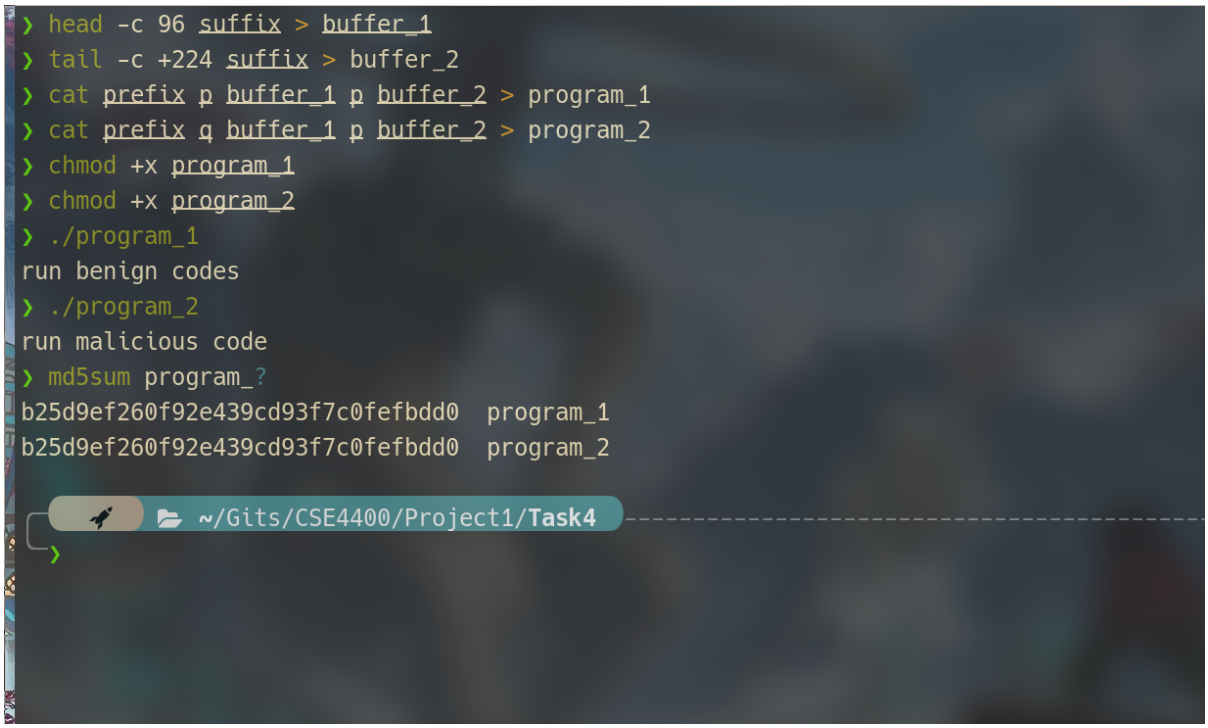

Figure 12: Two different programs with same hashes

Task 5

- I moved the data section and the text section.
- I identify the starting point of my array at offset 4096 which lucky enough to be a multiple of 64. The command to get prefix's region is the same as before: `head -c 4096 task5 > prefix`.
- P and Q are obtained just like we did before: `tail -c 128 out1 > p`
- I compared P and Q and see the difference at

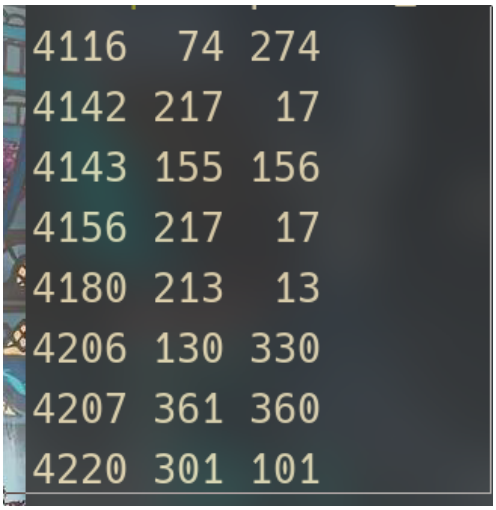

Figure 13: P and Q

- To show some reference, here is the different in ASCII text:

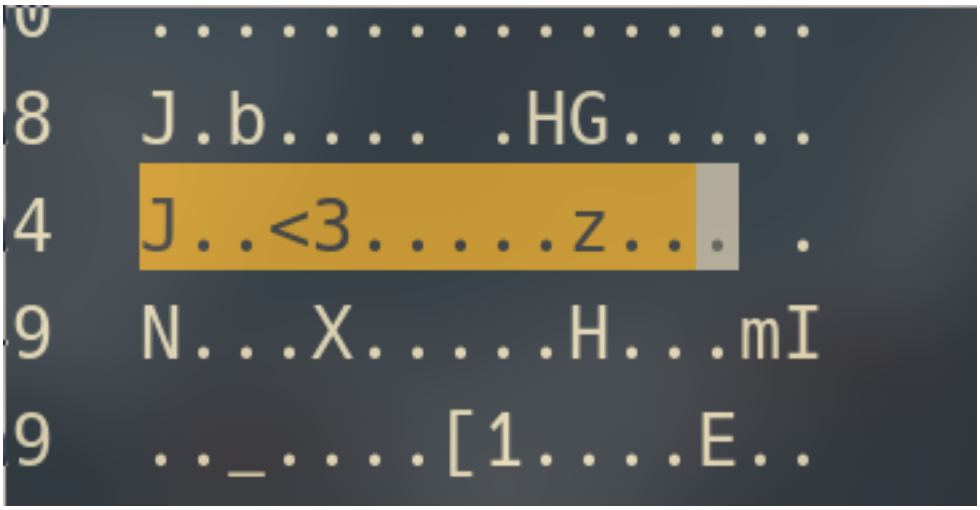

Figure 14: P

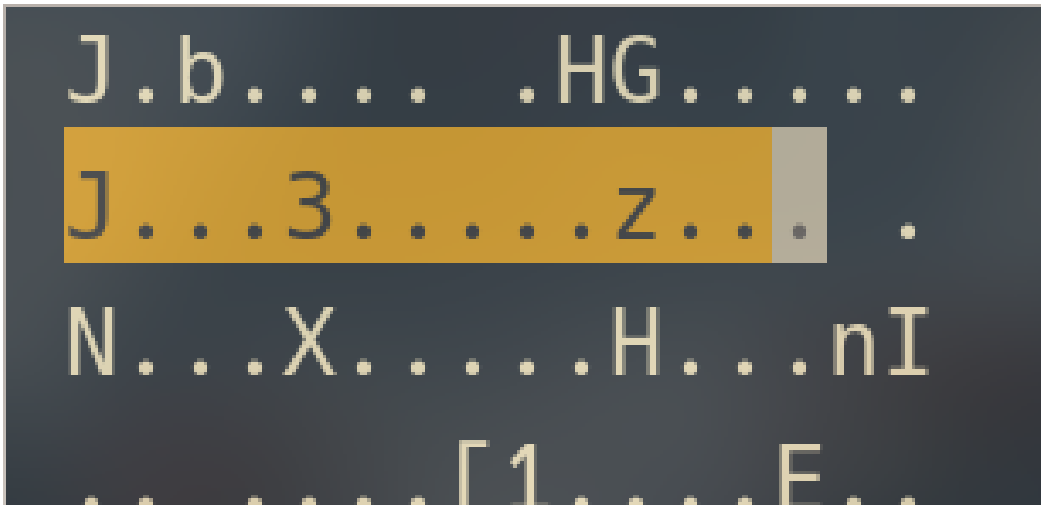

Figure 15: Q

- I marked the different and change the code accordingly. Then, I recompiled the code and find the suffix again.
- I add 128 (P's region) bytes with 4096 and get 4224 as the region for the suffix.
- After I put everything together, we have two different programs.

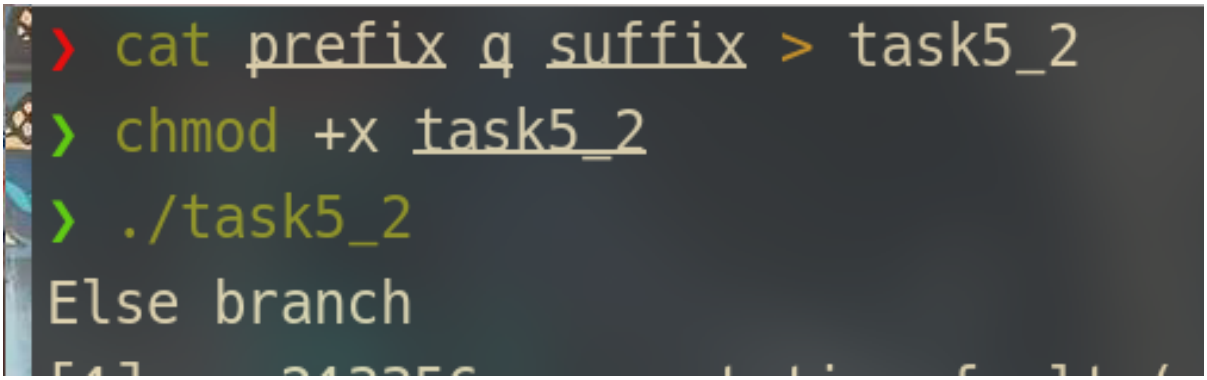

Figure 16: Two difference programs

- However they have the same hashes.

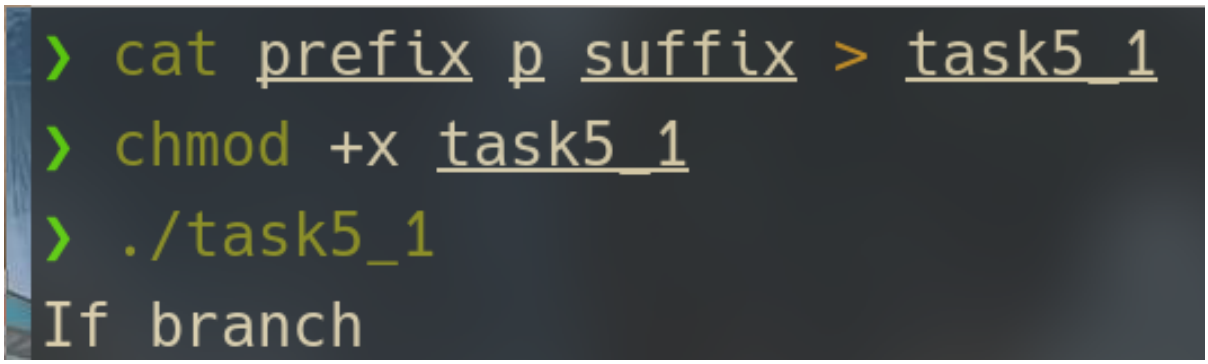

Figure 17: Two difference programs

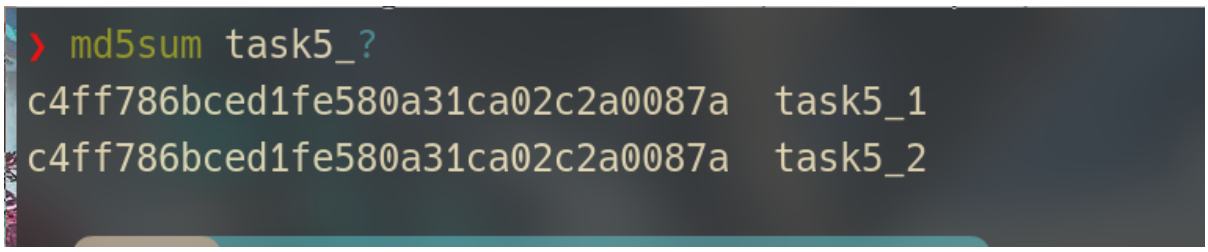

Figure 18: Same Hash