2018

Cryptography

LAURENCE BURTON (15003639)

Contents

Introduction.

For the module Cryptography the class were given four programming tasks to complete. This report will describe what these tasks included and compare the different algorithms used.

The Tasks

Task 1

The first task was to produce a program that was able to validate a 10-digit ISBN number and a 16-digit credit card number. It would allow a user to input a number and select if they're validating an ISBN or a credit card number. The program would first check if the number is the correct length. It would then run the number through an algorithm to check if the numbers were valid.

Task 2

For task 2 a Bose–Chaudhuri–Hocquenghem (BCH) error correcting code was implemented. This is used for reliable communication. (Sunwoo & Jeong, 2017) The program that was created would allow users to input a 10-digit number and test if it was valid. It was able to check and correct up to 2 errors. If there were any more, it would return saying "Three or more errors have been detected".

Task 3

Task 3 was to implement a Brute Force algorithm that would be able to crack a 6-digit password with characters between a-z and 0-9. Two methods where implemented. The first method was a recursive method that was very dynamic as it allowed users to enter how many characters they wanted the password to be. The second method was less dynamic, but it was easier to understand. It used 6 embedded for-loops. Each loop would edit a different character within the password. Once the brute force had been implemented it would decode 12 hashes which used Shai-1 encryption.

The results from the second implementation can be found in Appendix A.

These results show that using more characters in a password it can drastically change how long it takes to brute forces password.

When only two characters were used the program was able to crack the password instantly whereas using a 6-digit password with a combination of letters and numbers it took over 44 minutes. However, we did limit this test as it only had 6 digits and we didn't include capital letters or special characters. But it's expected if these parameters were included it would have taken over two hours.

Unfortunately, when testing the first implementation it took over 24 hours to crack a six-digit number and a timing function hadn't been implemented at the point. Because of this there isn't any official way to compare the two methods. But you can see that the second method is a lot faster.

Task 4

The final task was to implement two algorithms that were able to generate the prime factors of a given number. The first algorithm is called Fermat Factorisation and the second is called Dixon Factorisation. Fermat algorithm is $a^2 - N = b^2$ which translates to N = (A+B) *(A-B). Whereas the Dixon algorithm is $X^2 = Y^2$ mod N.

To implement the Dixon algorithm a random number is generated between the square root of the given number and the number itself. This is tested to see if it is 7-smooth. This means that all the factors are prime number up to 7. For example, 45 is 7-smooth because its factors can be 3*3*5 or 3^2*5.

A second number is generated under the same conditions. Once the prime factors are generated they're checked if the powers add up to an even number. If they don't this number is stored and another number is generated. This is repeated until the required condition is met. After these numbers have been formed they are manipulated and used to create the prime factors of the original number.

After generating the code for these two tasks we were given some test data. This included large integers which meant we had to use a special data type to save the values. Whilst running the factorisation for each number the time was recorded. The results for both algorithms can be found in Appendix B and C. From these results you can see that the Fermat algorithm ran quicker. However, this depended on the random numbers that were generated in the Dixon algorithm. Because it would run quicker if it only had to generate two numbers.

Conclusion

In conclusion five algorithms were created in Java. The first one was to validate ISBN and Credit Card numbers. Task 2 was for error correcting using BCH. Task 3 was a Brute-Force algorithm which was able to brute force a 6-character password. By implementing this it showed how adding an extra character could drastically increase how long it would take someone to crack your password. The final task was to implement two algorithms that were able to generate the prime factors of a large integer. The algorithms were called Fermat and Dixon. After implementing them it was found that Dixon could be quick, but it relied on a random number which made it unreliable. In contrast Fermat's time was more consistent.

Word Count (828)

References

Sunwoo, M.H. & Jeong, S.L. (2017) Low Power BCH Decoder using Early Termination. *ISOCC*, pp.260-61.

Appendix

Appendix A: Results from Task 3(Brute force)

Decrypted text: is
Time: 0 Minutes 0 seconds
Encrypted Text: b47f363e2b430c0647f14deea3eced9b0ef300ce

Decrypted text: this
Time: 0 Minutes 1 seconds
Encrypted Text: c2543fff3bfa6f144c2f06a7de6cd10c0b650cae

Decrypted text: very
Time: 0 Minutes 2 seconds
Encrypted Text: e74295bfc2ed0b52d40073e8ebad555100df1380

Decrypted text: fail7
Time: 0 Minutes 14 seconds
Encrypted Text: 77cfc481d3e76b543daf39e7f9bf86be2e664959

Decrypted text: 1you1
Time: 1 Minutes 7 seconds
Encrypted Text: 7302ba343c5ef19004df7489794a0adaee68d285

Decrypted text: 5you5
Time: 1 Minutes 17 seconds
Encrypted Text: 5cc48a1da13ad8cef1f5fad70ead8362aabc68a1

Decrypted text: 6will
Time: 1 Minutes 19 seconds
Encrypted Text: 57864da96344366865dd7cade69467d811a7961b

Decrypted text: cannot
Time: 4 Minutes 13 seconds
Encrypted Text: 6ef80072f39071d4118a6e7890e209d4dd07e504

Decrypted text: simple
Time: 26 Minutes 8 seconds
Encrypted Text: 0f7d0d088b6ea936fb25b477722d734706fe8b40

Decrypted text: 00if00
Time: 38 Minutes 24 seconds
Encrypted Text: 21e7133508c40bbdf2be8a7bdc35b7de0b618ae4

Decrypted text: 3crack
Time: 42 Minutes 2 seconds
Encrypted Text: 4bcc3a95bdd9a11b28883290b03086e82af90212

Decrypted text: 4this4
Time: 44 Minutes 15 seconds
Encrypted Text: 02285af8f969dc5c7b12be72fbce858997afe80a

Appendix B: Results from Task 4a(Fermat)

ID: 1
The factor of 224573 is (1617^2) - (1546^2)
Time take: 1 milliseconds

ID: 2
The factor of 299203 is (562^2) - (129^2)
Time take: 0 milliseconds

ID: 3
The factor of 1963867 is (1514^2) - (573^2)
Time take: 0 milliseconds

ID: 4
The factor of 6207251 is (4050^2) - (3193^2)
Time take: 0 milliseconds

ID: 5
The factor of 14674291 is (4370^2) - (2103^2)
Time take: 0 milliseconds

ID: 6
The factor of 23128513 is (4937^2) - (1116^2)
Time take: 0 milliseconds

ID: 7
The factor of 254855791 is (250604^2) - (250095^2)
Time take: 9 milliseconds

ID: 8
The factor of 428279361 is (21281^2) - (4960^2)
Time take: 0 milliseconds

ID: 9
The factor of 159649552547 is (403674^2) - (57473^2)
Time take: 1 milliseconds

ID: 10
The factor of 189061250479 is (440060^2) - (67761^2)
Time take: 1 milliseconds

ID: 11
The factor of 2211744201787 is (2459006^2) - (1958307^2)
Time take: 56 milliseconds

ID: 12
The factor of 7828669742987 is (2957166^2) - (957163^2)
Time take: 29 milliseconds

ID: 13
The factor of 48560209712519 is (6969540^2) - (119491^2)
Time take: 4 milliseconds

ID: 14
The factor of 35872004189003 is (6308442^2) - (1981019^2)
Time take: 59 milliseconds

ID: 15
The factor of 737785058178599 is (27166980^2) - (509651^2)
Time take: 15 milliseconds

ID: 16
The factor of 576460921650883 is (24674918^2) - (5691279^2)
Time take: 180 milliseconds

ID: 17
The factor of 1957432135202107 is (44285386^2) - (1939917^2)
Time take: 61 milliseconds

ID: 18
The factor of 2450609331732137 is (51552021^2) - (14387548^2)
Time take: 453 milliseconds

Appendix C: Results from Task 4b(Dixon)

ID: 1
The factor of 224573 is 71 * 3163
Time take: 12 milliseconds

ID: 2
The factor of 299203 is 433 * 691
Time take: 5 milliseconds

ID: 3
The factor of 1963867 is 2087 * 941
Time take: 46 milliseconds

ID: 4
The factor of 6207251 is 7243 * 857
Time take: 29 milliseconds

ID: 5
The factor of 14674291 is 2267 * 6473
Time take: 88 milliseconds

ID: 6
The factor of 23128513 is 3821 * 6053
Time take: 99 milliseconds

ID: 7
The factor of 254855791 is 500699 * 509
Time take: 1160 milliseconds

ID: 8
The factor of 428279361 is 2577 * 498579
Time take: 272 milliseconds

ID: 9
The factor of 159649552547 is 346201 * 461147
Time take: 12300 milliseconds

ID: 10
The factor of 189061250479 is 507821 * 372299
Time take: 11415 milliseconds

ID: 11
The factor of 2211744201787 is 4417313 * 500699
Time take: 564005 milliseconds

ID: 12
The factor of 7828669742987 is 2000003 * 3914329
Time take: 1685670 milliseconds