Algorithms Assignment 2

Your Name

May 22, 2023

Find

(a) $3^{1500}\text{mod}11$

We are using the Repeating Square method.

Hence, we find the binary representation of 1500, which is 10111011100.

Now we start the Repeating Square process:

$3^1 = 3\text{mod}11$, remainder is 3

$3^2 = 3^2\text{mod}11$, remainder is 9

$3^4 = 9^2\text{mod}11$, remainder is 4

$3^8 = 4^2\text{mod}11$, remainder is 5

$3^{16} = 5^2\text{mod}11$, remainder is 3

$3^{32} = 3^2\text{mod}11$, remainder is 9

$3^{64} = 9^2\text{mod}11$, remainder is 4

$3^{128} = 4^2\text{mod}11$, remainder is 5

$3^{256} = 5^2\text{mod}11$, remainder is 3

$3^{512} = 3^2\text{mod}11$, remainder is 9

$3^{1024} = 9^2\text{mod}11$, remainder is 4

Now, we multiply the remainders corresponding to the binary digit 1 in the binary representation of 1500:

$$3^{1024} \times 3^{256} \times 3^{128} \times 3^{64} \times 3^{16} \times 3^8 \times 3^4\text{mod}11$$

$$= 4 \times 3 \times 5 \times 4 \times 3 \times 5 \times 4\text{mod}11$$

$$= 16 \times 25 \times 36\text{mod}11 = 5 \times 3 \times 3\text{mod}11$$

$$= 45\text{mod}11 = \text{remainder is } 1$$

Hence, the remainder for $3^{1500}\text{mod}11$ is 1.

(b) $5^{4358}\text{mod}10$

We are using the Repeating Square method.

Hence, we find the binary representation of 4358, which is 1000100000110.

Now we start the Repeating Square process:

$5^1 = 5 \text{mod} 10$, remainder is 5

$5^2 = 5^2 \text{mod} 10$, remainder is 5

$5^4 = 5^2 \text{mod} 10$, remainder is 5

$5^8 = 5^2 \text{mod} 10$, remainder is 5

$5^{16} = 5^2 \text{mod} 10$, remainder is 5

$5^{32} = 5^2 \text{mod} 10$, remainder is 5

$5^{64} = 5^2 \text{mod} 10$, remainder is 5

$5^{128} = 5^2 \text{mod} 10$, remainder is 5

$5^{256} = 5^2 \text{mod} 10$, remainder is 5

$5^{512} = 5^2 \text{mod} 10$, remainder is 5

$5^{1024} = 5^2 \text{mod} 10$, remainder is 5

$5^{2048} = 5^2 \text{mod} 10$, remainder is 5

$5^{4096} = 5^2 \text{mod} 10$, remainder is 5

Now, we multiply the remainders corresponding to the binary digit 1 in the binary representation of 4358:

$$5^{4096} \times 5^{256} \times 5^4 \times 5^2 \text{mod} 10$$

$$= 5 \times 5 \times 5 \times 5 \text{mod} 10 \text{ then also the remainder is } 5$$

Hence, $5^{4358} \text{mod} 10$ is 5, and we can conclude that for 5 raised to any power mod 10, the remainder will be 5.

(c) $6^{22345} \text{mod} 7$

We are using the Repeating Square method.

Hence, we find the binary representation of 22345, which is 101011101001001.

Now we start the Repeating Square process:

$6^1 \text{mod} 7$, remainder is 6

$6^2 = 6^2 \text{mod} 7$, remainder is 1

$6^4 = 1^2 \text{mod} 7$, remainder is 1

$6^8 = 1^2 \text{mod} 7$, remainder is 1

$6^{16} = 1^2 \text{mod} 7$, remainder is 1

$6^{32} = 1^2 \text{mod} 7$, remainder is 1

$6^{64} = 1^2 \text{mod} 7$, remainder is 1

$6^{128} = 1^2 \text{mod} 7$, remainder is 1

$6^{256} = 1^2 \text{mod} 7$, remainder is 1

$6^{512} = 1^2 \text{mod} 7$, remainder is 1

$6^{1024} = 1^2 \text{mod} 7$, remainder is 1

$6^{2048} = 1^2 \text{mod} 7$, remainder is 1

$6^{4096} = 1^2 \text{mod} 7$, remainder is 1

$6^{8192} = 1^2 \text{mod} 7$, remainder is 1

$6^{16384} = 1^2 \text{mod} 7$, remainder is 1

Now, we multiply the remainders corresponding to the binary digit 1 in the binary representation of 22345:

$$6^{16384} \times 6^{4096} \times 6^{1024} \times 6^{512} \times 6^{256} \times 6^{64} \times 6^{8} \times 6 \text{mod} 7$$

$$= 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 6 \text{mod} 7 = 6$$

Hence, the value of 6^{22345}mod7 is 6.

Problem 2

(a) GCD(648, 124)

To find the greatest common divisor (GCD) of 648 and 124, you can use the Euclidean algorithm. Here's how it works:

1. Divide 648 by 124: $648 \div 124 = 5$ remainder $28 \Rightarrow \text{GCD}(124, 28)$
2. Divide 124 by 28: $124 \div 28 = 4$ remainder $12 \Rightarrow \text{GCD}(28, 12)$
3. Divide 28 by 12: $28 \div 12 = 2$ remainder $4 \Rightarrow \text{GCD}(12, 4)$
4. Divide 12 by 4: $12 \div 4 = 3$ remainder $0 \Rightarrow \text{GCD}(4, 0)$ (this is the final form we need)

Since the remainder is now 0, we have found that the GCD of 648 and 124 is the last non-zero divisor, which is 4.

Therefore, the GCD of 648 and 124 is 4.

(b) GCD(123456789, 123456788)

To find the greatest common divisor (GCD) of 123456789 and 123456788, you can use the Euclidean algorithm. Here's how it works:

1. Divide 123456789 by 123456788:

$123456789 \div 123456788 = 1$ remainder $1 \Rightarrow \text{GCD}(123456788, 1)$

2. Divide 123456788 by 1:

$123456788 \div 1 = 123456788$ remainder $0 \Rightarrow \text{GCD}(1, 0)$

Since the remainder is now 0, we have found that the GCD of 123456789 and 123456788 is the last non-zero remainder, which is 1. We can also say that the GCD of any consecutive numbers is also 1.

Therefore, the GCD of 123456789 and 123456788 is 1.

(c) GCD($2^{300} \times 3^{200}$, 2^{200})

In this case, since both numbers are in the form of prime factorizations, the GCD between them will be the common highest power.

$2^{300} \times 3^{200}$

2^{200}

And GCD($2^{300} \times 3^{200}$, 2^{200}) = GCD(2^{200}, 0)

Hence, 2^{200} will be the GCD of the given numbers.

Problem 3

Alice wants to secretly send Bob a specific number. They can communicate only over a public (non-secret, insecure) channel. How can they do it using the

Diffie-Hellman Key Exchange Protocol?

Reminder: The Diffie-Hellman Key Exchange Protocol allows Alice and Bob to jointly establish a shared secret number over an insecure channel. But this secret number does not necessarily coincide with the specific number that Alice wants to send to Bob.

To use the Diffie-Hellman Key Exchange Protocol to secretly send a specific number from Alice to Bob, they can follow these steps:

1. Agree on parameters:

- Choose a large prime number, p, and a generator, g.

- These parameters should be known to both Alice and Bob but can be shared publicly.

2. Alice's steps:

- Choose a private random number, a.

- Calculate $A = g^a \mathrm{mod} p$.

- Send A to Bob over the public channel.

3. Bob's steps:

- Choose a private random number, b.

- Calculate $B = g^b \mathrm{mod} p$.

- Send B to Alice over the public channel.

4. Secret calculation:

- Alice calculates the secret number as $s = B^a \mathrm{mod} p$.

- Bob calculates the secret number as $s = A^b \mathrm{mod} p$.

5. Result:

- Now Alice and Bob both have the same secret number, s.

- Alice can encode her specific number by adding it modulo p to the secret number.

- Alice sends the encoded number to Bob over the public channel.

- Bob decodes the received number by subtracting the shared secret number modulo p.

- Now Bob has the specific number that Alice wanted to send him.

By following this protocol, Alice and Bob can establish a shared secret number and use it to encode/decode the specific number Alice wants to send. The protocol ensures that the shared secret is not revealed over the public channel.

Certainly! Here's the Diffie-Hellman key exchange protocol example with the values you provided, expressed in LaTeX format:

1. Setup:

- Alice and Bob agree on a prime number, let's say $p = 19$, which will be the modulus. - They also agree on a base number, let's say $g = 2$, which will be the generator.

2. Private Key Generation:

- Alice chooses a private key, let's say $a = 23$. - Bob chooses a private key, let's say $b = 31$.

3. Public Key Exchange:

- Alice calculates her public key by performing the following calculation: $A = g^a \mod p$. In this case, $A = 2^{23} \mod 19$, which equals 9. - Bob calculates

his public key by performing the following calculation: $B = g^b \mod p$. In this case, $B = 2^{31} \mod 19$, which equals 16.

4. Key Calculation:

- Alice receives Bob's public key, $B = 16$. - Bob receives Alice's public key, $A = 9$. - Alice calculates the shared secret key by performing the following calculation: $s = B^a \mod p$. In this case, $s = 16^{23} \mod 19$, which equals 17. - Bob calculates the shared secret key by performing the following calculation: $s = A^b \mod p$. In this case, $s = 9^{31} \mod 19$, which also equals 17.

5. Shared Secret Key:

- Alice and Bob now have the same shared secret key, which is $s = 17$.

The shared secret key can be used by Alice and Bob for further encryption or cryptographic purposes.

Encryption:

Message: I love algorithms
Shift: 17

To perform the encryption, we shift each letter in the message by 17 positions in the alphabet:

I + 17 = Z
space remains unchanged
l + 17 = c
o + 17 = f
v + 17 = s
e + 17 = v
space remains unchanged
a + 17 = r
l + 17 = c
g + 17 = x
o + 17 = f
r + 17 = i
i + 17 = z
t + 17 = g
h + 17 = y
m + 17 = d
s + 17 = j

Encrypted message: Z cfsv rxczbgnyq

So, "I love algorithms" encrypted with the shared secret key as a Caesar cipher would result in the message "Z cfsv rxczbgnyq".

Decryption:

To decrypt the message, we would shift each letter in the encrypted message back by 17 positions in the alphabet.

Encrypted message: Z cfsv rxczbgnyq
Shift: -17

Performing the decryption:

Z - 17 = I
space remains unchanged

c - 17 = l
f - 17 = o
s - 17 = v
v - 17 = e
space remains unchanged
r - 17 = a
x - 17 = l
c - 17 = g
z - 17 = o
b - 17 = r
g - 17 = i
n - 17 = t
y - 17 = h
q - 17 = m

Decrypted message: I love algorithms

By shifting each letter in the encrypted message back by 17 positions, we retrieve the original message "I love algorithms" as the decrypted result.

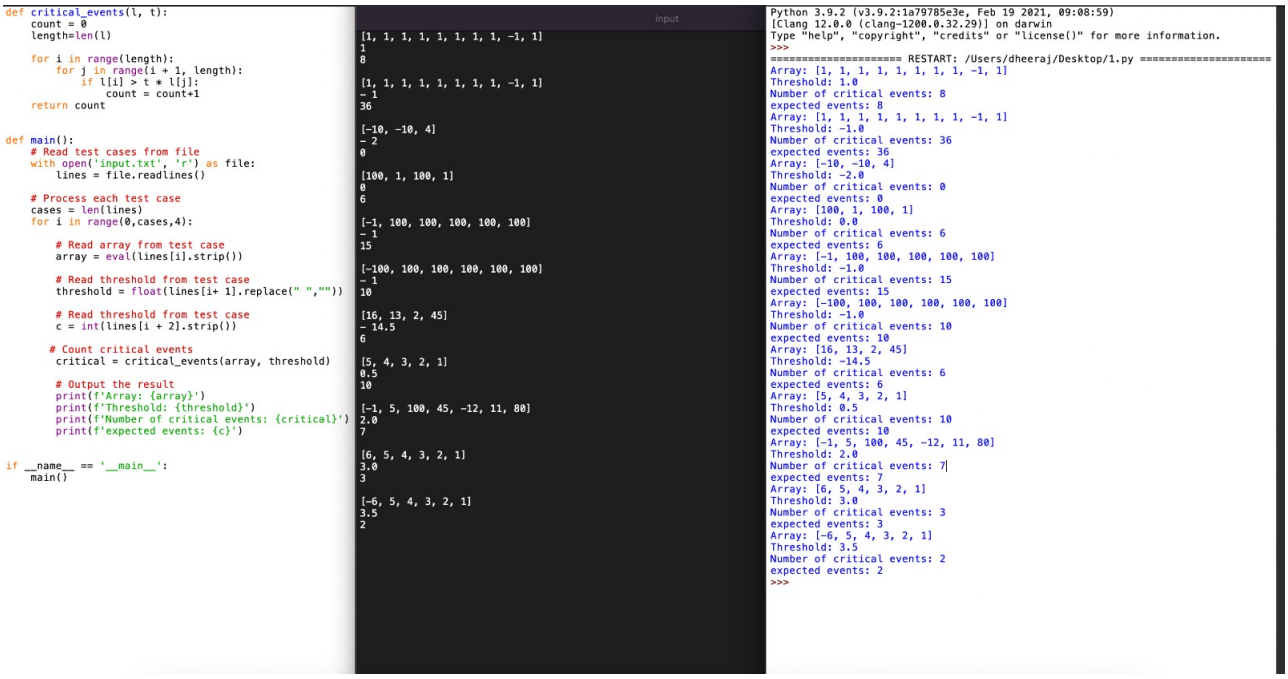
Please note that in the Caesar cipher, the decryption process involves shifting the letters in the opposite direction of the encryption shift.

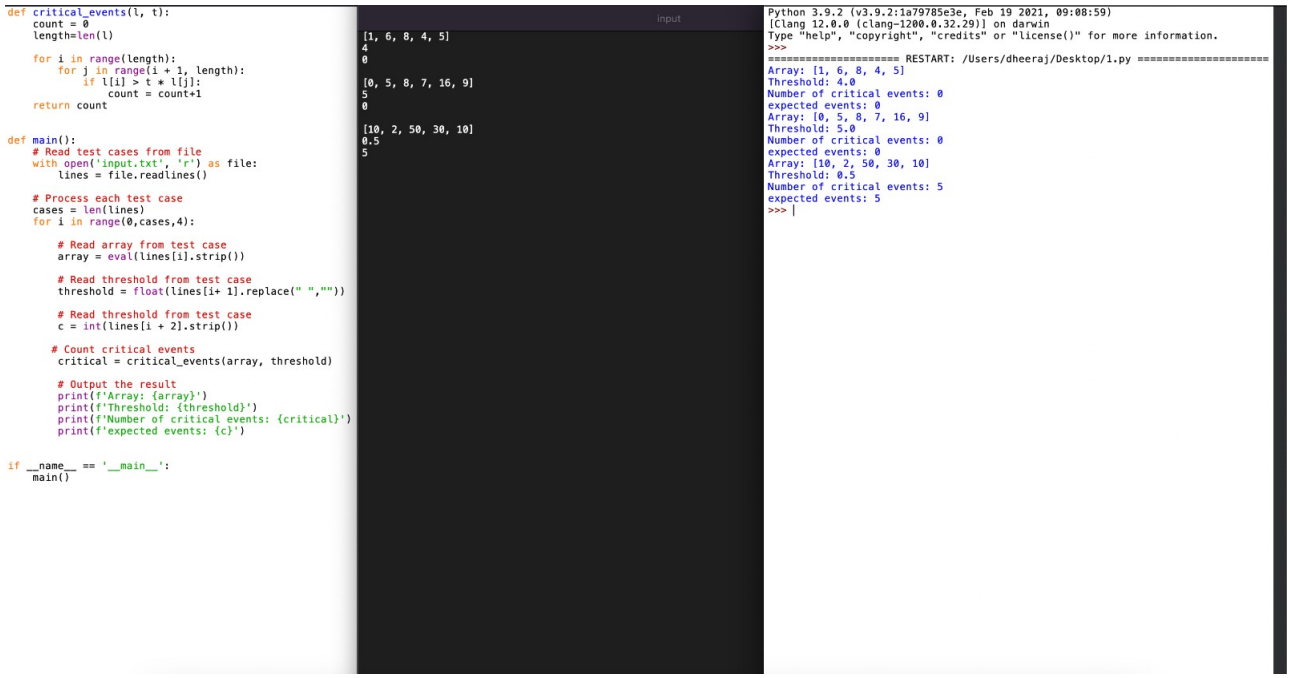
like wise one can pass messages through this secret code.

This code takes the input file and reads the array,threshold value, expected value and returns the

```
    def critical_events(l, t):
    count = 0
    length=len(l)

    for i in range(length):
        for j in range(i + 1, length):
            if l[i] > t * l[j]:
                count = count+1
    return count


def main():
    # Read test cases from file
    with open('input.txt', 'r') as file:
        lines = file.readlines()

    # Process each test case
    cases = len(lines)
    for i in range(0,cases,4):

        # Read array from test case
        array = eval(lines[i].strip())
```

```
        # Read threshold from test case
        threshold = float(lines[i+ 1].replace(" ",""))

        # Read threshold from test case
        c = int(lines[i + 2].strip())

       # Count critical events
        critical = critical_events(array, threshold)

        # Output the result
        print(f'Array: {array}')
        print(f'Threshold: {threshold}')
        print(f'Number of critical events: {critical}')
        print(f'expected events: {c}')


if __name__ == '__main__':
    main()
```





```
def critical_events(l, t):
    count = 0
    length=len(l)

    for i in range(length):
        for j in range(i + 1, length):
            if l[i] > t * l[j]:
                count = count+1
    return count


def main():
    # Read test cases from file
    with open('input.txt', 'r') as file:
        lines = file.readlines()

    # Process each test case
    cases = len(lines)
    for i in range(0,cases,4):

        # Read array from test case
        array = eval(lines[i].strip())

        # Read threshold from test case
        threshold = float(lines[i+ 1].replace(" ",""))

        # Read threshold from test case
        c = int(lines[i + 2].strip())

       # Count critical events
        critical = critical_events(array, threshold)

        # Output the result
        print(f'Array: {array}')
        print(f'Threshold: {threshold}')
        print(f'Number of critical events: {critical}')
        print(f'expected events: {c}')


if __name__ == '__main__':
    main()
```

input

```
[1, 6, 8, 4, 5]
4
0

[0, 5, 8, 7, 16, 9]
5
0

[10, 2, 50, 30, 10]
0.5
5
```

```
Python 3.9.2 (v3.9.2:1a79785e3e, Feb 19 2021, 09:08:59)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /Users/dheeraj/Desktop/1.py ==========
Array: [1, 6, 8, 4, 5]
Threshold: 4.0
Number of critical events: 0
expected events: 0
Array: [0, 5, 8, 7, 16, 9]
Threshold: 5.0
Number of critical events: 0
expected events: 0
Array: [10, 2, 50, 30, 10]
Threshold: 0.5
Number of critical events: 5
expected events: 5
>>>
```