Experiment 05

AIM - Design and implement a Hashing Function(RandomizedXOR)

Learning Objective: Students should be able to design and implement a hashing function

Tools: C/C++/Java/Python or any computational software

Theory:

The "RandomizedXOR" hashing algorithm is a custom hashing method designed to transform input strings into fixed-size bit sequences, termed hash values. It integrates several key steps to achieve this objective:

- **ASCII Conversion:** Initially, the input string is converted into a string of ASCII values. Each character in the input string is replaced by its corresponding ASCII code.
- **Randomization:** Subsequently, the ASCII string is randomized by shuffling its characters. This step adds an element of unpredictability to the hashing process.
- **Left Shift:** The randomized string undergoes a left shift operation by its length. This manipulation further alters the sequence of bits within the string.
- **Bit Selection (Y):** From the shifted string, every 4th bit is selected to form the Y string. This selection pattern helps in generating a unique pattern for Y.
- **Y Repetition:** The Y string is repeated such that its length matches that of the randomized string. This ensures compatibility for the subsequent XOR operation.
- **Bitwise XOR Operation:** Finally, the randomized string and the repeated Y string undergo bitwise XOR operation. This operation combines the two strings to produce the hash value.

Working:

1. **Custom Hashing Function (custom_hash):** This function accepts an input string and performs the aforementioned steps to generate the hash value.
2. **ASCII Conversion and Randomization:** The input string is converted to ASCII and randomized using random.sample.
3. **Left Shift and Y Selection:** The randomized string is left-shifted by its length, and every 4th bit is selected to form the Y string.
4. **Y Repetition:** The Y string is repeated to match the length of the randomized string.
5. **Bitwise XOR Operation:** The randomized string and the repeated Y string undergo bitwise XOR operation to produce the hash value.

Implementation:

```
import random
def custom_hash(s):
    k = ""
    for i in s:
        k += str(ord(i))
    k = "".join(random.sample(k, len(k)))
    k = str(int(k) << len(k))
```

```
    y = k[::4]
    y = (y * ((len(k) // len(y)) + 1))[: len(k)]
    hashed_value = int(k) ^ int(y)
    return hashed_value
input_string = "hello"
hashed_value = custom_hash(input_string)
print("Hashed value:", hashed_value)
```

Output:

```
PS C:\College\YR3\CSS\5> python .\5.1.py
Hashed value: 17114022823441559359
```

Result & Discussion:

The "RandomizedXOR" hashing algorithm transforms input strings into unique hash values by combining ASCII conversion, randomization, left shift, bit selection, repetition, and bitwise XOR operation. This process ensures that each input string generates a distinct hash value, facilitating tasks such as data integrity verification and cryptographic operations. The algorithm provides a balance between randomness and determinism, making it suitable for a variety of applications requiring consistent and secure hashing mechanisms.

Learning Outcomes:

The student should be able to design & implement hashing functions
LO1: To describe & understand hashing functions
LO2: To implement hashing functions

Course Outcomes: Upon completion of the course students will be able to understand & implement hashing functions.

Conclusion:

In conclusion, we were able to develop a new hashing algorithm,'RandomizedXOR' using the concepts of ASCII conversion, left shift and bitwise XOR operation.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				