Experiment 05

Aim: Develop your own hashing algorithm based on your logic.

Tools: Visual Studio Code(C++)

Theory: Hashing algorithms

Hashing algorithms are cryptographic functions that convert input data of arbitrary size into a fixed-size string of bytes, typically for the purpose of data integrity verification, password hashing, and digital signatures. These algorithms play a critical role in ensuring the security and integrity of data in various applications.

Hashing serves several important purposes despite being irreversible:

1. Data Integrity: Hashing is commonly used to verify the integrity of data. By storing the hash of a piece of data alongside the data itself, you can later recompute the hash and compare it to the stored hash. If the hashes match, it's highly likely that the data hasn't been altered. This is commonly used in cryptographic protocols, file verification, and ensuring data consistency.

2. Data Security: Hashing is a fundamental component of many cryptographic algorithms. Passwords, for example, are often hashed before being stored in a database. When a user attempts to log in, the system hashes the provided password and compares it to the stored hash. This way, even if the database is compromised, the actual passwords are not easily retrievable.

3. Efficient Data Retrieval: Hashing is used in data structures like hash tables to efficiently store and retrieve data based on keys. In this context, the hash serves as an index to quickly locate the associated data.

4. Anonymization: Hashing can be used to anonymize sensitive information. For example, hashing email addresses or user IDs before storing or transmitting them can protect privacy while still allowing for certain operations (like comparing two hashed values).

5. Cryptographic Techniques: Hashing is used in cryptographic techniques like digital signatures and message authentication codes (MACs) to ensure the authenticity and integrity of messages without revealing the contents.

So, even though hashing is irreversible, it's an essential tool in modern computing for various purposes related to security, data integrity, and efficiency.

Algorithm Development:

1. Initialize Variables: Set the initial hash value (`hash`) to 0 and the initial power value (`power`) to 1.

2. Iterate Through Characters: Loop through each character `c` in the input string.

3. Convert Character to Integer: Convert the character `c` to an integer representing its position in the alphabet by subtracting the ASCII value of 'a' from the ASCII value of `c`.

This gives a value between 0 and 25 for lowercase letters. For uppercase letters, you would subtract the ASCII value of 'A' instead.

4. Update Hash Value: Update the hash value by adding the product of `(c - 'a' + 1)` (the converted integer value) and `power` to the current hash value (`hash`). This step is equivalent to multiplying the current hash value by a base (here, 26) and adding the converted integer value.

5. Update Power: Update the power value by multiplying it by a prime number (here, 31) to prevent overflow and keep the hash values spread out.

6. Repeat: Repeat steps 2-5 for each character in the input string.

7. Modular Arithmetic: After processing all characters, take the hash value modulo a large prime number (here, 1000000007) to ensure the hash value stays within a reasonable range and to reduce the likelihood of collisions.

8. Return Hash Value: Return the final hash value.

Implementation:

Code:

```
#include <bits/stdc++.h>
using namespace std;
unsigned int customHash(const string& input) {
    const unsigned int prime = 31;
    const unsigned int mod = 1000000007;

    unsigned int hash = 0;
    unsigned int power = 1;

    for (char c : input) {
        hash = (hash + (c - 'a' + 1) * power) % mod;
        power = (power * prime) % mod;
    }
    return hash;
}
int main() { string
    message;
    cout<<"Enter the message : ";
    getline(cin,message);
    unsigned int hashValue = customHash(message);
    cout << "Hash value of '" << message << "': " << hashValue << endl;

    return 0;
}
```

Output:

```
Enter the message : hello Aakash
Hash value of 'hello Aakash': 216393337
PS C:\Users\Admin\Documents>
```

Result and Discussion:
In this experiment, we developed and implemented a custom hashing algorithm using various mathematical operations. The custom hash generated for the input text was obtained successfully.

Learning Outcomes:

1. Participants gained insight into the foundational principles of hashing algorithms, such as modular arithmetic, prime numbers, and bitwise operations. They learned how these concepts contribute to the design of hashing algorithms and their effectiveness in generating unique hash values.
2. Participants acquired practical experience in developing a custom hashing algorithm using C++. By implementing the algorithm, they learned how to translate theoretical concepts into functional code, including iterating through input data, performing arithmetic operations, and managing data types.
3. Participants learned about the practical applications of hashing algorithms in ensuring data security and integrity. They understood how hashing is utilized in cryptographic protocols, such as password hashing and data authentication, to protect sensitive information, prevent tampering, and verify the authenticity of data

Conclusion: Through this experiment, participants gained valuable insights into the intricacies of hashing algorithms and their practical implementation. Additionally, participants developed a deeper understanding of the role of hashing algorithms in data security and cryptography, which is essential in today's digital age.

For Faculty Use

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