**Blockchain Technology Lab**

**Lab – 2**

**Aim :** Implement SHA-1 and apply it on Doubly Linked List data

**Code:**

*import* json

*import* struct

*# Node class for the doubly linked list*

*class* Node:

*def* \_\_init\_\_(self, admission\_number, marks, branch):

*self*.admission\_number = admission\_number

*self*.marks = marks

*self*.branch = branch

*self*.prev = None

*self*.next = None

*# Doubly Linked List class*

*class* DoublyLinkedList:

*def* \_\_init\_\_(self):

*self*.head = None

*def* append(self, admission\_number, marks, branch):

        new\_node = Node(admission\_number, marks, branch)

*if* *not* *self*.head:

*self*.head = new\_node

*return*

        temp = *self*.head

*while* temp.next:

            temp = temp.next

        temp.next = new\_node

        new\_node.prev = temp

*def* to\_dict(self):

        """Convert the linked list to a list of dictionaries for JSON serialization."""

        result = []

        temp = *self*.head

*while* temp:

            result.append({

                "admission\_number": temp.admission\_number,

                "marks": temp.marks,

                "branch": temp.branch

            })

            temp = temp.next

*return* result

*def* sha1(self):

        """Compute SHA-1 hash of the concatenated data in the linked list."""

        concatenated\_data = *b*""  *# Use bytes for concatenation*

        temp = *self*.head

*while* temp:

            concatenated\_data += *f*"{temp.admission\_number}{temp.marks}{temp.branch}".encode('utf-8')

            temp = temp.next

*return* sha1\_hash(concatenated\_data)

*# SHA-1 implementation*

*def* sha1\_hash(data):

*# Initial hash values*

    h = [

*0x*67452301,

*0x*EFCDAB89,

*0x*98BADCFE,

*0x*10325476,

*0x*C3D2E1F0

    ]

*# Preprocessing*

    original\_byte\_len = len(data)

    original\_bit\_len = original\_byte\_len \* 8

*# Append the '1' bit*

    data += *b*'\x80'

*while* (len(data) \* 8) % 512 != 448:

        data += *b*'\x00'

*# Append the original length as a 64-bit big-endian integer*

    data += struct.pack('>Q', original\_bit\_len)

*# Process each 512-bit chunk*

*for* i *in* range(0, len(data), 64):

        chunk = data[i:i + 64]

        w = list(struct.unpack('>16L', chunk)) + [0] \* 64  *# Adjusted to 64*

*for* j *in* range(16, 80):

            w[j] = left\_rotate(w[j - 3] ^ w[j - 8] ^ w[j - 14] ^ w[j - 16], 1)

        a, b, c, d, e = h

*for* j *in* range(80):

*if* 0 <= j <= 19:

                f = (b & c) | (~b & d)

                k = *0x*5A827999

*elif* 20 <= j <= 39:

                f = b ^ c ^ d

                k = *0x*6ED9EBA1

*elif* 40 <= j <= 59:

                f = (b & c) | (b & d) | (c & d)

                k = *0x*8F1BBCDC

*else*:

                f = b ^ c ^ d

                k = *0x*CA62C1D6

            temp = (left\_rotate(a, 5) + f + e + k + w[j]) & *0x*FFFFFFFF

            e = d

            d = c

            c = left\_rotate(b, 30)

            b = a

            a = temp

*# Add the compressed chunk to the current hash value*

        h = [(x + y) & *0x*FFFFFFFF *for* x, y *in* zip(h, [a, b, c, d, e])]

*# Produce the final hash value (big-endian) as a 40-digit hexadecimal number*

*return* ''.join(*f*'{x*:08x*}' *for* x *in* h)

*# Rotate left operation*

*def* left\_rotate(n, b):

*return* ((n << b) | (n >> (32 - b))) & *0x*FFFFFFFF

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    dll = DoublyLinkedList()

*# Sample data*

    dll.append(101, 85, "CS")

    dll.append(102, 90, "IT")

    dll.append(103, 78, "ME")

*# Compute SHA-1 hash*

    sha1\_hash\_value = dll.sha1()

    print(*f*"SHA-1 Hash: {sha1\_hash\_value}")

*# Convert to JSON and save to file*

    linked\_list\_data = dll.to\_dict()

*with* open("linked\_list\_data.json", "w") *as* json\_file:

        json.dump(linked\_list\_data, json\_file, indent=4)

    print("Linked list data saved to 'linked\_list\_data.json'.")

**Output:**

**A screenshot of a computer

Description automatically generated**