PROBLEM 2 - PROJECT REPORT

Implemented birthday attack algorithm in python to find the collisions in the hash function BadHash40 with SHA-256 as hashing algorithm that outputs the first 40 bits of the argument.

Code Execution:
The command line for running the code is,

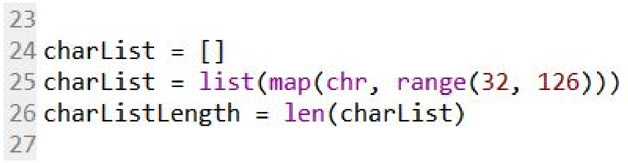
```
python main.py -lhash NN
```

Where, lhash parameter sets the length of the hash input/output list and NN is an integer number that denotes the number of messages to be generated and checked against each other.
-lhash being an optional switch with NN as its argument, running the code with the command line 'python main.py' will cause the program to execute until a collision is found.

Output Returned:
Whether the -lhash parameter is passed or not, the program will generate a hash.data file (file name is defaulted in the code) and saves it to the same location. It contains the random message generated and the first 40 bits of its corresponding hash value, one message per line.
Once the collision is found it stops writing to the hash.data file.

Code Description:

- Imported the following libraries
  - ❏ Sys - To access the list of command line arguments passed
  - ❏ Random - To choose random integer for generating input messages
  - ❏ Hashlib - Contains the implementation of SHA256 hashing algorithm
  - ❏ Datetime - To compute the total time taken
- Initialize an empty list and map all the characters from the defaulted range of ASCII values which can be changed accordingly. 32 represents ' ' and 126 represents '~'. And hence the generated random message includes the characters ' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-', '.', '/', 0 to 9, ':', ';', '<', '=', '>', '?', '@', A to Z, '[', '\', ']', '^', '_', '`', a to z, '{', '|', '}', '~'.



```

charList = []
charList = list(map(chr, range(32, 126)))
charListLength = len(charList)

```

- Input message length is defaulted to 256 bits (32 characters as 1 character = 8 bits). Change it accordingly to try for different lengths of messages.
- To speed up the collision search, we defined a class called Tree for the implementation of binary tree structure. Object of the class contains the random message generated, it's

BadHash value (first 40 bits) and it's id which is an integer representation of the hash value.

- The below __init__ method is called as soon as an object of the class is initiated.

```python
    def __init__(self, message='', hashValue=''):
        self.left = None
        self.right = None
        self.message = message
        self.hashValue = hashValue
        if (self.hashValue):
            self.id = int(self.hashValue, 16)
        else:
            self.id = 0
```

- addNode function will add the new node to the tree based on its id, and the findNode function finds a node in the tree and returns it if found else none. __str__ function is used to represent the node in its string format.
- Badhash40 function computes SHA256 hash of the input message. Upon ensuring that the message is a byte string, we will get the hexadecimal format of the encoded data and return the first 40 bits (5 bytes) of the hash digest.

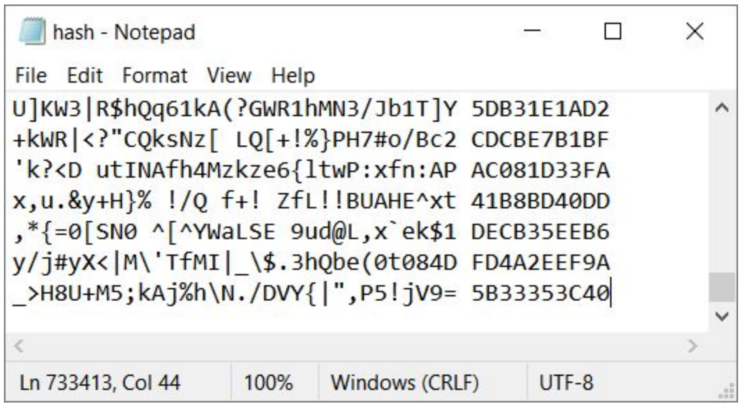
```python

def BadHash40(message):
    if (isinstance(message, str)):
        message = message.encode('utf-8')
        hashString = hashlib.sha256(message).hexdigest()
    return hashString[0:10].upper()

```

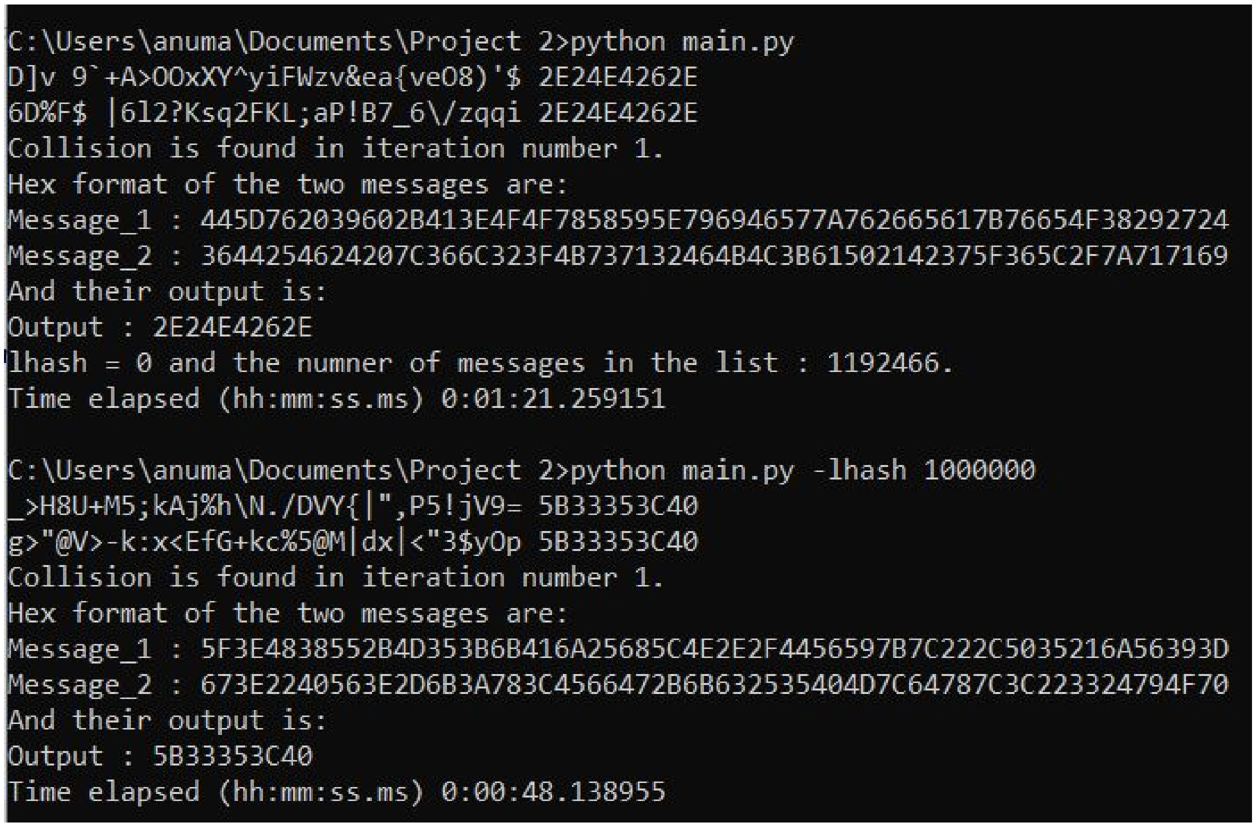
- In the main function, we will retrieve the NN parameter if passed and continue accordingly. When lhash is not given, random messages are generated until the hash values of any two of them matches. If lhash is passed, random messages are generated until the lhash count is reached, if a collision is found, it returns the messages and hash value. If the collision is not found within the lhash count, it restarts and computes the list again. This continues until the collision is found.

Output:



```
U]KW3|R$hQq61kA(?GWR1hMN3/Jb1T]Y 5DB31E1AD2
+kWR|<?"CQksNz[ LQ[+!%}PH7#o/Bc2 CDCBE7B1BF
'k?<D utINAfh4Mzkze6{ltwP:xfn:AP AC081D33FA
x,u.&y+H}% !/Q f+! ZfL!!BUAHE^xt 41B8BD40DD
,*{=0[SN0 ^[^YWaLSE 9ud@L,x`ek$1 DECB35EEB6
y/j#yX<|M\'TfMI|_\$.3hQbe(0t084D FD4A2EEF9A
_>H8U+M5;kAj%h\N./DVY{|",P5!jV9= 5B33353C40
```

Ln 733413, Col 44 | 100% | Windows (CRLF) | UTF-8



```
C:\Users\anuma\Documents\Project 2>python main.py
D]v 9`+A>OOxXY^yiFWzv&ea{veO8)'$ 2E24E4262E
6D%F$ |6l2?Ksq2FKL;aP!B7_6\/zqqi 2E24E4262E
Collision is found in iteration number 1.
Hex format of the two messages are:
Message_1 : 445D762039602B413E4F4F7858595E796946577A762665617B76654F38292724
Message_2 : 3644254624207C366C323F4B737132464B4C3B61502142375F365C2F7A717169
And their output is:
Output : 2E24E4262E
lhash = 0 and the numner of messages in the list : 1192466.
Time elapsed (hh:mm:ss.ms) 0:01:21.259151

C:\Users\anuma\Documents\Project 2>python main.py -lhash 1000000
_>H8U+M5;kAj%h\N./DVY{|",P5!jV9= 5B33353C40
g>"@V>-k:x<EfG+kc%5@M|dx|<"3$yOp 5B33353C40
Collision is found in iteration number 1.
Hex format of the two messages are:
Message_1 : 5F3E4838552B4D353B6B416A25685C4E2E2F4456597B7C222C5035216A56393D
Message_2 : 673E2240563E2D6B3A783C4566472B6B632535404D7C64787C3C223324794F70
And their output is:
Output : 5B33353C40
Time elapsed (hh:mm:ss.ms) 0:00:48.138955
```

In the second execution with 1000000 as the length of the list we got 5B33353C40 as output for the messages
5F3E4838552B4D353B6B416A25685C4E2E2F4456597B7C222C5035216A56393D and
673E2240563E2D6B3A783C4566472B6B632535404D7C64787C3C223324794F70.

References:

https://www.tutorialspoint.com/python_data_structure/python_binary_search_tree.htm