Code (checked for correctness with NIST example of "abc" and
"abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijk
lmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu" from
example document: http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/SHA512.pdf):

```
#!/usr/bin/python
import sys
sys.path.append( "/home/shay/a/kalita/ece404/HW7/BitVector-3.3.2" )
from BitVector import *
import string
import math
if len(sys.argv) != 2:
    sys.stderr.write("Usage: hw07.py <name of input file.>\n")
    sys.exit(1)

def sig1(x):
    x1 = x.deep_copy()
    x8 = x.deep_copy()
    x7 = x.deep_copy()
    s1 = (x1 >> 1) ^ (x8 >> 8) ^ (x7.shift_right(7))
    return s1

def sig2(x):
    x19 = x.deep_copy()
    x61 = x.deep_copy()
    x6 = x.deep_copy()
    s2 = (x19 >> 19) ^ (x61 >> 61) ^ (x6.shift_right(6))
    return s2

def Ch(e, f, g):
    ret = (e & f) ^ (~e & g)
    return ret

def Maj(a, b, c):
    ret = (a & b) ^ (a & c) ^ (b & c)
    return ret

def siga(a):
    a28 = a.deep_copy()
    a34 = a.deep_copy()
    a39 = a.deep_copy()
    ret = (a28 >> 28) ^ (a34 >> 34) ^ (a39 >> 39)
    return ret

def sige(e):
    e14 = e.deep_copy()
    e18 = e.deep_copy()
    e41 = e.deep_copy()
    ret = (e14 >> 14) ^ (e18 >> 18) ^ (e41 >> 41)
    return ret

def T1(h, e, f, g, Wi, Ki):
    tot = (int(h) + int(Ch(e, f, g)) + int(sige(e)) + int(Wi) + int(Ki)) % (2 ** 64)
    T = BitVector(intVal = tot, size = 64)
    return T

def T2(a, b, c):
    tot = (int(siga(a)) + int(Maj(a, b, c))) % (2 ** 64)
    T = BitVector(intVal = tot, size = 64)
    return T

pt_fh = open(sys.argv[1], 'r') #plain text file open
#plntxt = pt_fh.read() #read the text from the and store it in a variable
plntxt = 'abc'
#plntxt =
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqr
pt_fh.close() #close the input file file handler
```

```
#print(plntxt)
bv = BitVector(textstring = plntxt) #get bitvector representation of i/p text
#print(len(pt_bv))
# Initialize hashcode for the first block. Subsequetnly, the
# output for each 1024-bit block of the input message becomes
# the hashcode for the next block of the message.
h0 = BitVector(hexstring='6a09e667f3bcc908')
#print(h0)
h1 = BitVector(hexstring='bb67ae8584caa73b')
#print(h1)
h2 = BitVector(hexstring='3c6ef372fe94f82b')
#print(h2)
h3 = BitVector(hexstring='a54ff53a5f1d36f1')
#print(h3)
h4 = BitVector(hexstring='510e527fade682d1')
#print(h4)
h5 = BitVector(hexstring='9b05688c2b3e6c1f')
#print(h5)
h6 = BitVector(hexstring='1f83d9abfb41bd6b')
#print(h6)
h7 = BitVector(hexstring='5be0cd19137e2179')
#print(h7)
'''
print('Initial hash value:')
print('H[0] = ' + h0.getHexStringFromBitVector())
print('H[1] = ' + h1.getHexStringFromBitVector())
print('H[2] = ' + h2.getHexStringFromBitVector())
print('H[3] = ' + h3.getHexStringFromBitVector())
print('H[4] = ' + h4.getHexStringFromBitVector())
print('H[5] = ' + h5.getHexStringFromBitVector())
print('H[6] = ' + h6.getHexStringFromBitVector())
print('H[7] = ' + h7.getHexStringFromBitVector())
'''
length = bv.length() #find the length of the i/p text bitvector
#print(length)
bv1 = bv + BitVector(bitstring = "1") #append a '1' to the i/p bitvector
length1 = bv1.length() #length of bv after appending a '1'
#print(length1)
howmanyzeroes = (896 - length1) % 1024 #find the number of zeroes
#print(howmanyzeroes)
zerolist = [0] * howmanyzeroes
bv2 = bv1 + BitVector(bitlist = zerolist)
bv3 = BitVector(intVal = length, size = 128) #the length of i/p text in 128 bits
#print(bv3.length())
bv4 = bv2 + bv3 #the final message with length being multiple of 1024
#print(bv4)
#print(len(bv4))
words = [None] * 80
'''
test=BitVector(bitstring = '1011011')
print(test)
testb=test.deep_copy()
print(testb.shift_right(3))
print(testb)
print(test)
'''
#directly generate K[] vector
K = [None] * 80
K[0] = BitVector(hexstring='428a2f98d728ae22')
K[1] = BitVector(hexstring='7137449123ef65cd')
K[2] = BitVector(hexstring='b5c0fbcfec4d3b2f')
K[3] = BitVector(hexstring='e9b5dba58189dbbc')
K[4] = BitVector(hexstring='3956c25bf348b538')
K[5] = BitVector(hexstring='59f111f1b605d019')
K[6] = BitVector(hexstring='923f82a4af194f9b')
K[7] = BitVector(hexstring='ab1c5ed5da6d8118')
K[8] = BitVector(hexstring='d807aa98a3030242')
```

```
K[9] = BitVector(hexstring='12835b0145706fbe')
K[10] = BitVector(hexstring='243185be4ee4b28c')
K[11] = BitVector(hexstring='550c7dc3d5ffb4e2')
K[12] = BitVector(hexstring='72be5d74f27b896f')
K[13] = BitVector(hexstring='80deb1fe3b1696b1')
K[14] = BitVector(hexstring='9bdc06a725c71235')
K[15] = BitVector(hexstring='c19bf174cf692694')
K[16] = BitVector(hexstring='e49b69c19ef14ad2')
K[17] = BitVector(hexstring='efbe4786384f25e3')
K[18] = BitVector(hexstring='0fc19dc68b8cd5b5')
K[19] = BitVector(hexstring='240ca1cc77ac9c65')
K[20] = BitVector(hexstring='2de92c6f592b0275')
K[21] = BitVector(hexstring='4a7484aa6ea6e483')
K[22] = BitVector(hexstring='5cb0a9dcbd41fbd4')
K[23] = BitVector(hexstring='76f988da831153b5')
K[24] = BitVector(hexstring='983e5152ee66dfab')
K[25] = BitVector(hexstring='a831c66d2db43210')
K[26] = BitVector(hexstring='b00327c898fb213f')
K[27] = BitVector(hexstring='bf597fc7beef0ee4')
K[28] = BitVector(hexstring='c6e00bf33da88fc2')
K[29] = BitVector(hexstring='d5a79147930aa725')
K[30] = BitVector(hexstring='06ca6351e003826f')
K[31] = BitVector(hexstring='142929670a0e6e70')
K[32] = BitVector(hexstring='27b70a8546d22ffc')
K[33] = BitVector(hexstring='2e1b21385c26c926')
K[34] = BitVector(hexstring='4d2c6dfc5ac42aed')
K[35] = BitVector(hexstring='53380d139d95b3df')
K[36] = BitVector(hexstring='650a73548baf63de')
K[37] = BitVector(hexstring='766a0abb3c77b2a8')
K[38] = BitVector(hexstring='81c2c92e47edaee6')
K[39] = BitVector(hexstring='92722c851482353b')
K[40] = BitVector(hexstring='a2bfe8a14cf10364')
K[41] = BitVector(hexstring='a81a664bbc423001')
K[42] = BitVector(hexstring='c24b8b70d0f89791')
K[43] = BitVector(hexstring='c76c51a30654be30')
K[44] = BitVector(hexstring='d192e819d6ef5218')
K[45] = BitVector(hexstring='d69906245565a910')
K[46] = BitVector(hexstring='f40e35855771202a')
K[47] = BitVector(hexstring='106aa07032bbd1b8')
K[48] = BitVector(hexstring='19a4c116b8d2d0c8')
K[49] = BitVector(hexstring='1e376c085141ab53')
K[50] = BitVector(hexstring='2748774cdf8eeb99')
K[51] = BitVector(hexstring='34b0bcb5e19b48a8')
K[52] = BitVector(hexstring='391c0cb3c5c95a63')
K[53] = BitVector(hexstring='4ed8aa4ae3418acb')
K[54] = BitVector(hexstring='5b9cca4f7763e373')
K[55] = BitVector(hexstring='682e6ff3d6b2b8a3')
K[56] = BitVector(hexstring='748f82ee5defb2fc')
K[57] = BitVector(hexstring='78a5636f43172f60')
K[58] = BitVector(hexstring='84c87814a1f0ab72')
K[59] = BitVector(hexstring='8cc702081a6439ec')
K[60] = BitVector(hexstring='90befffa23631e28')
K[61] = BitVector(hexstring='a4506cebde82bde9')
K[62] = BitVector(hexstring='bef9a3f7b2c67915')
K[63] = BitVector(hexstring='c67178f2e372532b')
K[64] = BitVector(hexstring='ca273eceea26619c')
K[65] = BitVector(hexstring='d186b8c721c0c207')
K[66] = BitVector(hexstring='eada7dd6cde0eb1e')
K[67] = BitVector(hexstring='f57d4f7fee6ed178')
K[68] = BitVector(hexstring='06f067aa72176fba')
K[69] = BitVector(hexstring='0a637dc5a2c898a6')
K[70] = BitVector(hexstring='113f9804bef90dae')
K[71] = BitVector(hexstring='1b710b35131c471b')
K[72] = BitVector(hexstring='28db77f523047d84')
K[73] = BitVector(hexstring='32caab7b40c72493')
K[74] = BitVector(hexstring='3c9ebe0a15c9bebc')
K[75] = BitVector(hexstring='431d67c49c100d4c')
```

```
K[76] = BitVector(hexstring='4cc5d4becb3e42b6')
K[77] = BitVector(hexstring='597f299cfc657e2a')
K[78] = BitVector(hexstring='5fcb6fab3ad6faec')
K[79] = BitVector(hexstring='6c44198c4a475817')
'''
for i in range(0,80):
    print(str(i)+':'+str(K[i]))
'''
#now let us begin the block-by-block processing
#print(sig2(bv))
for n in range(0, bv4.length(), 1024): #for each 1024-bit block
    block = bv4[n:n+1024] #get the 1024-bit current block
    '''
    print('Initial hash value..')
    print('H[0] = ' + h0.getHexStringFromBitVector())
    print('H[1] = ' + h1.getHexStringFromBitVector())
    print('H[2] = ' + h2.getHexStringFromBitVector())
    print('H[3] = ' + h3.getHexStringFromBitVector())
    print('H[4] = ' + h4.getHexStringFromBitVector())
    print('H[5] = ' + h5.getHexStringFromBitVector())
    print('H[6] = ' + h6.getHexStringFromBitVector())
    print('H[7] = ' + h7.getHexStringFromBitVector())
    '''
    a, b, c, d, e, f, g, h = h0, h1, h2, h3, h4, h5, h6, h7
    words[0:16] = [block[i:i+64] for i in range(0, 1024, 64)] #words 0 to 15
    for i in range(16, 80): #words 16 to 79
        tot = (int(words[i-16]) + int(sig1(words[i-15])) + int(words[i-7]) + int(sig2(words[i-2]))) % (2
** 64)
        words[i] = BitVector(intVal = tot, size = 64)
    '''
    print('Block Contents:')
    print('W[0] = ' + words[0].getHexStringFromBitVector())
    print('W[1] = ' + words[1].getHexStringFromBitVector())
    print('W[2] = ' + words[2].getHexStringFromBitVector())
    print('W[3] = ' + words[3].getHexStringFromBitVector())
    print('W[4] = ' + words[4].getHexStringFromBitVector())
    print('W[5] = ' + words[5].getHexStringFromBitVector())
    print('W[6] = ' + words[6].getHexStringFromBitVector())
    print('W[7] = ' + words[7].getHexStringFromBitVector())
    print('W[8] = ' + words[8].getHexStringFromBitVector())
    print('W[9] = ' + words[9].getHexStringFromBitVector())
    print('W[10] = ' + words[10].getHexStringFromBitVector())
    print('W[11] = ' + words[11].getHexStringFromBitVector())
    print('W[12] = ' + words[12].getHexStringFromBitVector())
    print('W[13] = ' + words[13].getHexStringFromBitVector())
    print('W[14] = ' + words[14].getHexStringFromBitVector())
    print('W[15] = ' + words[15].getHexStringFromBitVector())
    print('\n')
    '''
    for i in range(0, 80): #for each of the 80 rounds

        a1 = a.deep_copy() #for T1, T2 functions
        b1 = b.deep_copy() #for T1, T2 functions
        c1 = c.deep_copy() #for T1, T2 functions
        e1 = e.deep_copy() #for T1, T2 functions
        f1 = f.deep_copy() #for T1, T2 functions
        g1 = g.deep_copy() #for T1, T2 functions
        h1_tmp = h.deep_copy() #for T1, T2 functions

        #now change h through a based on the algorithm to intermix
        h = BitVector(intVal = int(g), size = 64)
        g = BitVector(intVal = int(f), size = 64)
        f = BitVector(intVal = int(e), size = 64)
        e = BitVector(intVal = ((int(d) + int(T1(h1_tmp,e1,f1,g1,words[i],K[i]))) % (2 ** 64)), size = 64)
        d = BitVector(intVal = int(c), size = 64)
        c = BitVector(intVal = int(b), size = 64)
        b = BitVector(intVal = int(a), size = 64)
```

```
        a = BitVector(intVal = ((int(T2(a1,b1,c1)) + int(T1(h1_tmp,e1,f1,g1,words[i],K[i]))) % (2 **
64)), size = 64)
        '''
        print('i: ' + str(i) +": "+ a.getHexStringFromBitVector() + ' ' + b.getHexStringFromBitVector() +
' ' +  c.getHexStringFromBitVector() + ' ' + d.getHexStringFromBitVector() + ' ' +
e.getHexStringFromBitVector() + ' ' + f.getHexStringFromBitVector() + ' ' +  g.getHexStringFromBitVector
() + ' ' + h.getHexStringFromBitVector())
        '''
    '''
    print('Printing h[0] to h[7]')
    print('H[0] = ' + h0.getHexStringFromBitVector())
    print('H[1] = ' + h1.getHexStringFromBitVector())
    print('H[2] = ' + h2.getHexStringFromBitVector())
    print('H[3] = ' + h3.getHexStringFromBitVector())
    print('H[4] = ' + h4.getHexStringFromBitVector())
    print('H[5] = ' + h5.getHexStringFromBitVector())
    print('H[6] = ' + h6.getHexStringFromBitVector())
    print('H[7] = ' + h7.getHexStringFromBitVector())

    print('Printing a to h')
    print('a = ' + a.getHexStringFromBitVector())
    print('b = ' + b.getHexStringFromBitVector())
    print('c = ' + c.getHexStringFromBitVector())
    print('d = ' + d.getHexStringFromBitVector())
    print('e = ' + e.getHexStringFromBitVector())
    print('f = ' + f.getHexStringFromBitVector())
    print('g = ' + g.getHexStringFromBitVector())
    print('h = ' + h.getHexStringFromBitVector())
    '''
    #now add original hash values before the round processing began to a..h
    h0 = BitVector( intVal = (int(h0) + int(a)) % (2**64), size=64 )
    h1 = BitVector( intVal = (int(h1) + int(b)) % (2**64), size=64 )
    h2 = BitVector( intVal = (int(h2) + int(c)) % (2**64), size=64 )
    h3 = BitVector( intVal = (int(h3) + int(d)) % (2**64), size=64 )
    h4 = BitVector( intVal = (int(h4) + int(e)) % (2**64), size=64 )
    h5 = BitVector( intVal = (int(h5) + int(f)) % (2**64), size=64 )
    h6 = BitVector( intVal = (int(h6) + int(g)) % (2**64), size=64 )
    h7 = BitVector( intVal = (int(h7) + int(h)) % (2**64), size=64 )
    '''
    print('h + a...')
    print('H[0] = ' + h0.getHexStringFromBitVector())
    print('H[1] = ' + h1.getHexStringFromBitVector())
    print('H[2] = ' + h2.getHexStringFromBitVector())
    print('H[3] = ' + h3.getHexStringFromBitVector())
    print('H[4] = ' + h4.getHexStringFromBitVector())
    print('H[5] = ' + h5.getHexStringFromBitVector())
    print('H[6] = ' + h6.getHexStringFromBitVector())
    print('H[7] = ' + h7.getHexStringFromBitVector())
    '''
#now the final hash code of of the final block is the hash of the overall i/p
message_hash = h0 + h1 + h2 + h3 + h4 + h5 + h6 + h7 #add the hashes to get overall hash
hash_hex_string = message_hash.getHexStringFromBitVector() #get the hash string
print(hash_hex_string) #print the hash string
#write to output.txt
f_op = open('output.txt', 'w') #open output.txt for writing to
f_op.write(hash_hex_string) #write to output.txt
f_op.close() #close output.txt file handler
```

Message.txt

```
The time has come to talk of many things: of shoes and ships and sealing-wax of cabbages and kings and
why the sea is boiling hot and whether pigs have wings.
```

Output.txt
357c86d0e34d4767af209fc7b605e0a08fb008ded0e1d9d3657639a6bf4b0ca47a22daafa7c31
68ea6d1d626f1c38c5a8fcee670d1e1ef2c191856f64772b0f1