Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №11»**

«Исследование криптографических хеш-функций»

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Супрунюк Евгений Андреевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования Python и позволяет провести хеширование по алгоритму SHA-1.

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложения, которое реализовывает хеширование при помощи SHA-1. На листинге 2.1 представлены функции, реализующие данную функциональность.

|  |
| --- |
| def read\_file(name\_file: str, size\_buffer: int):      file\_for\_only\_read = open(name\_file, 'r', encoding="utf8")      return file\_for\_only\_read.read() if size\_buffer <= -1 else file\_for\_only\_read.read()[:size\_buffer]  def addition(message: str, value\_l: int):      rate = 1088      possible\_values\_l = [1, 2, 3, 4, 5, 6]      if value\_l is None or value\_l not in possible\_values\_l:          value\_l = 6      message\_bin = ''.join([(16 - len(bin(ord(i))[2:])) \* '0' + bin(ord(i))[2:] for i in message])      message\_bin\_add = message\_bin + '1' + '0' \* (rate - len(message\_bin) % rate - 2) + '1'      array\_n = [message\_bin\_add[i \* rate: (i + 1) \* rate] for i in range(len(message\_bin\_add) // rate)]      return array\_n  def build\_histogram(x, y):      plt.figure(figsize=(9, 9))      plt.bar(x, y)      plt.title('Быстродействие алгоритма хеширования SHA-1')      plt.xlabel('Длина сообщение')      plt.ylabel('Время вычисления хеша')      plt.show()  def expansion\_to\_multiplicity\_of\_512(message\_binary):      if len(message\_binary) % 512 < 448 and len(message\_binary) % 512 != 0:          extended\_block = '1' + (448 - len(message\_binary) % 512 - 1) \* '0' \                           + (64 - len(bin(len(message\_binary))                              [2:])) \* '0' + bin(len(message\_binary))[2:]      elif 448 < len(message\_binary) % 512 < 512:          extended\_block = '1' + (512 - 1 - len(message\_binary) % 512) \* '0' \                           + 448 \* '0' + \              (64 - len(bin(len(message\_binary))[2:])               ) \* '0' + bin(len(message\_binary))[2:]      else:          extended\_block = '1' + 447 \* '0' + \              (64 - len(bin(len(message\_binary))[2:])               ) \* '0' + bin(len(message\_binary))[2:]      extended\_message = message\_binary + extended\_block      return extended\_message  def split\_into\_blocks(block, size\_blocks):      return [block[i \* size\_blocks: (i + 1) \* size\_blocks] for i in range(len(block) // size\_blocks)]  def expansion\_to\_80\_blocks(blocks\_16\_32):      blocks\_80\_32 = [int(i, 2) for i in blocks\_16\_32]      for i in range(16, 80):          blocks\_80\_32.append(              (blocks\_80\_32[i - 3] ^ blocks\_80\_32[i - 8] ^ blocks\_80\_32[i - 14] ^ blocks\_80\_32[i - 16]) << 1)      return blocks\_80\_32  def sha\_1(message: str, output\_in\_console: bool):      message\_binary = ''.join(          [(16 - len(bin(ord(i))[2:])) \* '0' + bin(ord(i))[2:] for i in message])      extended\_message = expansion\_to\_multiplicity\_of\_512(message\_binary)      blocks\_512 = split\_into\_blocks(extended\_message, 512)      A = 0x67452301      B = 0xEFCDAB89      C = 0x98BADCFE      D = 0x10325476      E = 0xC3D2E1F0      for block in blocks\_512:          blocks\_16\_32 = split\_into\_blocks(block, 32)          blocks\_80\_32 = expansion\_to\_80\_blocks(blocks\_16\_32)          a, b, c, d, e = A, B, C, D, E          for i in range(80):              if 0 <= i <= 19:                  f = (b & c) | (~b & d)                  k = 0x5A827999              elif 20 <= i <= 39:                  f = b ^ c ^ d                  k = 0x6ED9EBA1              elif 40 <= i <= 59:                  f = (b & c) | (b & d) | (c & d)                  k = 0x8F1BBCDC              else:                  f = b ^ c ^ d                  k = 0xCA62C1D6              temp = ((a << 5) + f + e + k + blocks\_80\_32[i]) % (2 \*\* 32)              e = d              d = c              c = b << 30              b = a              a = temp          A += a          B += b          C += c          D += d          E += e      A\_32 = (32 - len(bin((A + a) % (2 \*\* 32))              [2:])) \* '0' + bin((A + a) % (2 \*\* 32))[2:]      B\_32 = (32 - len(bin((B + b) % (2 \*\* 32))              [2:])) \* '0' + bin((B + b) % (2 \*\* 32))[2:]      C\_32 = (32 - len(bin((C + c) % (2 \*\* 32))              [2:])) \* '0' + bin((C + c) % (2 \*\* 32))[2:]      D\_32 = (32 - len(bin((D + d) % (2 \*\* 32))              [2:])) \* '0' + bin((D + d) % (2 \*\* 32))[2:]      E\_32 = (32 - len(bin((E + e) % (2 \*\* 32))              [2:])) \* '0' + bin((E + e) % (2 \*\* 32))[2:]      digest\_2 = A\_32 + B\_32 + C\_32 + D\_32 + E\_32      digest\_16 = ''.join([hex(int(digest\_2[i \* 4: (i + 1) \* 4], 2))[2:]                          for i in range(len(digest\_2) // 4)]).upper()      if output\_in\_console:          print(f'Исходное сообщение и его длина: {len(message\_binary)} {message\_binary}\n'                f'Расширенное сообщение и его длина: {len(extended\_message)} {extended\_message}\n'                f'Сообщение, разбитое на 512 битные блоки: {blocks\_512}\n'                f'Полученный хэш (дайджест) в двоичной системе: {digest\_2}\n'                f'Длина хэша (дайджест) в двоичной системе: {len(digest\_2)}\n'                f'Полученный хэш (дайджест) в шестнадцатеричной системе: {digest\_16}\n'                f'Длина хэша (дайджест) в шестнадцатеричной системе: {len(digest\_16)}\n'                ) |

Листинг 2.1 –код программы, реализующие заданную ранее функциональность

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение. Рисунок 3.1 и 3.2 показывают требуемые в данной лабораторной работе результаты.

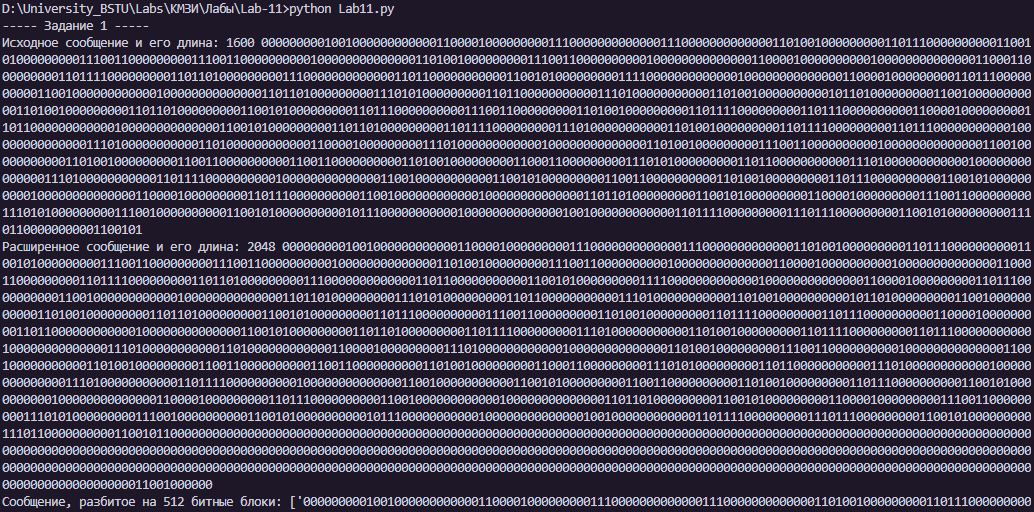


Рисунок 3.1 – Результат работы

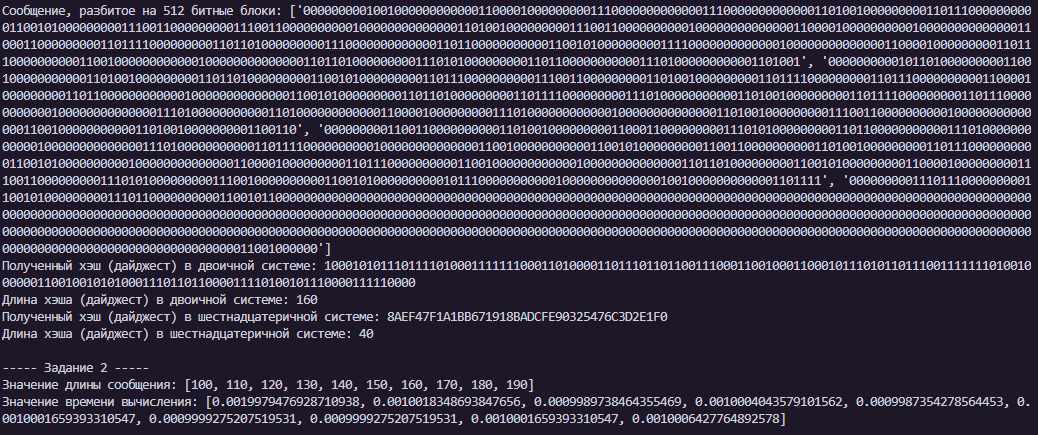


Рисунок 3.2 – Результат работы

**Вывод**

В ходе лабораторной работы было разработано приложение для хеширования при помощи SHA-1.