Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 5

“Компьютерная реализация хэш-функций на примере MD5.”

Выполнил студент гр. 753504

Горбачёнок К. Н.

Проверил

Протько М.И.

Минск, 2020

# Постановка задачи

Реализовать программу получающую на вход данные из файла и отдающую на выход дайджест сообщения длиной 128 бит по алгоритму MD5

# Описание алгоритма

Алгоритм состоит из следующих шагов:

***Шаг 1: добавление недостающих битов***

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина≡448 mod 512).

***Шаг 2: добавление длины***

64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита.

***Шаг 3: инициализация MD-буфера***

Используется 128-битный буфер для хранения промежуточных и окончательных результатов *хэш-функции*. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

А = 01234567; В = 89ABCDEF; C = FEDCBA98; D = 76543210

***Шаг 4: обработка последовательности 512-битных (16-словных) блоков***

Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую fF, fG, fH и fI соответственно.

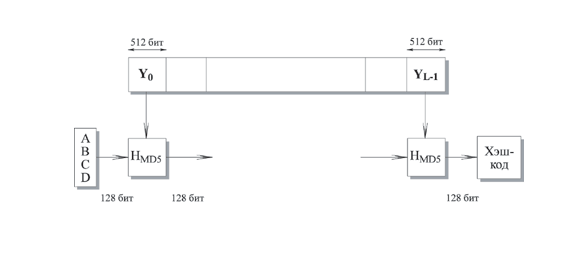
Каждый цикл принимает в качестве входа текущий 512-битный блок Yq, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением *дайджеста*, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы T[1 ... 64], построенной на основе функции sin. i-ый элемент T, обозначаемый T[i], имеет значение, равное целой части от 232 \* abs (sin (i)), i задано в радианах. Так как abs (sin (i)) является числом между 0 и 1, каждый элемент Т является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный" набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

Для получения MDq+1 выход четырех циклов складывается по модулю 232 с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

***Шаг 5: выход***

После обработки всех *L* 512-битных блоков выходом *L*-ой стадии является 128-битный *дайджест сообщения*.

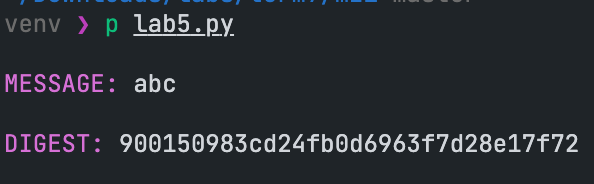
# Блок-схема алгоритма



***Блок-схема реализации хэш-функции MD5***

# Результаты работы программы

***Исходный файл***

******

***Результат работы программы***

# Прграммный код

import struct

from enum import Enum

from math import (

floor,

sin,

)

from bitarray import bitarray

class MD5Buffer(Enum):

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

class MD5(object):

\_string = None

\_buffers = {

MD5Buffer.A: None,

MD5Buffer.B: None,

MD5Buffer.C: None,

MD5Buffer.D: None,

}

@classmethod

def hash(cls, string):

cls.\_string = string

preprocessed\_bit\_array = cls.\_step\_2(cls.\_step\_1())

cls.\_step\_3()

cls.\_step\_4(preprocessed\_bit\_array)

return cls.\_step\_5()

@classmethod

def \_step\_1(cls):

bit\_array = bitarray(endian="big")

bit\_array.frombytes(cls.\_string.encode("utf-8"))

bit\_array.append(1)

while bit\_array.length() % 512 != 448:

bit\_array.append(0)

return bitarray(bit\_array, endian="little")

@classmethod

def \_step\_2(cls, step\_1\_result):

length = (len(cls.\_string) \* 8) % pow(2, 64)

length\_bit\_array = bitarray(endian="little")

length\_bit\_array.frombytes(struct.pack("<Q", length))

result = step\_1\_result.copy()

result.extend(length\_bit\_array)

return result

@classmethod

def \_step\_3(cls):

for buffer\_type in cls.\_buffers.keys():

cls.\_buffers[buffer\_type] = buffer\_type.value

@classmethod

def \_step\_4(cls, step\_2\_result):

def F(x, y, z): return (x & y) | (~x & z)

def G(x, y, z): return (x & z) | (y & ~z)

def H(x, y, z): return x ^ y ^ z

def I(x, y, z): return y ^ (x | ~z)

def rotate\_left(x, n): return (x << n) | (x >> (32 - n))

def modular\_add(a, b): return (a + b) % pow(2, 32)

T = [floor(pow(2, 32) \* abs(sin(i + 1))) for i in range(64)]

N = len(step\_2\_result) // 32

for chunk\_index in range(N // 16):

start = chunk\_index \* 512

X = [step\_2\_result[start + (x \* 32): start + (x \* 32) + 32]

for x in range(16)]

X = [int.from\_bytes(word.tobytes(), byteorder="little")

for word in X]

A = cls.\_buffers[MD5Buffer.A]

B = cls.\_buffers[MD5Buffer.B]

C = cls.\_buffers[MD5Buffer.C]

D = cls.\_buffers[MD5Buffer.D]

for i in range(4 \* 16):

if 0 <= i <= 15:

k = i

s = [7, 12, 17, 22]

temp = F(B, C, D)

elif 16 <= i <= 31:

k = ((5 \* i) + 1) % 16

s = [5, 9, 14, 20]

temp = G(B, C, D)

elif 32 <= i <= 47:

k = ((3 \* i) + 5) % 16

s = [4, 11, 16, 23]

temp = H(B, C, D)

elif 48 <= i <= 63:

k = (7 \* i) % 16

s = [6, 10, 15, 21]

temp = I(B, C, D)

temp = modular\_add(temp, X[k])

temp = modular\_add(temp, T[i])

temp = modular\_add(temp, A)

temp = rotate\_left(temp, s[i % 4])

temp = modular\_add(temp, B)

A = D

D = C

C = B

B = temp

cls.\_buffers[MD5Buffer.A] = modular\_add(

cls.\_buffers[MD5Buffer.A], A)

cls.\_buffers[MD5Buffer.B] = modular\_add(

cls.\_buffers[MD5Buffer.B], B)

cls.\_buffers[MD5Buffer.C] = modular\_add(

cls.\_buffers[MD5Buffer.C], C)

cls.\_buffers[MD5Buffer.D] = modular\_add(

cls.\_buffers[MD5Buffer.D], D)

@classmethod

def \_step\_5(cls):

A = struct.unpack("<I", struct.pack(

">I", cls.\_buffers[MD5Buffer.A]))[0]

B = struct.unpack("<I", struct.pack(

">I", cls.\_buffers[MD5Buffer.B]))[0]

C = struct.unpack("<I", struct.pack(

">I", cls.\_buffers[MD5Buffer.C]))[0]

D = struct.unpack("<I", struct.pack(

">I", cls.\_buffers[MD5Buffer.D]))[0]

return f"{format(A, '08x')}{format(B, '08x')}{format(C, '08x')}{format(D, '08x')}"

from algorithms.md5 import MD5

from helpers.file\_info import read\_text

from helpers.colored\_text import colored\_text

def main():

text = read\_text(filename="resources/lab5.txt")

print(colored\_text("MESSAGE:"), text)

print(colored\_text("DIGEST:"), MD5.hash(text))

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

main()

# Вывод

В ходе написания лабораторной работы была изучена алгоритм хеширования MD5 , а также написана программная реализация.