MD5 算法

15331191 廖颖泓

一、 算法概述

MD5 **消息摘要算法**（英语：MD5 Message-Digest Algorithm），一种被广泛使用的密码散列函数，可以产生出一个 128 位（16 字节）的散列值（hash value），用于确保信息传输完整一致。MD5 由罗纳德·李维斯特设计，于 1992 年公开，用以取代 MD4 算法。这套算法的程序在 RFC 1321 中被加以规范。

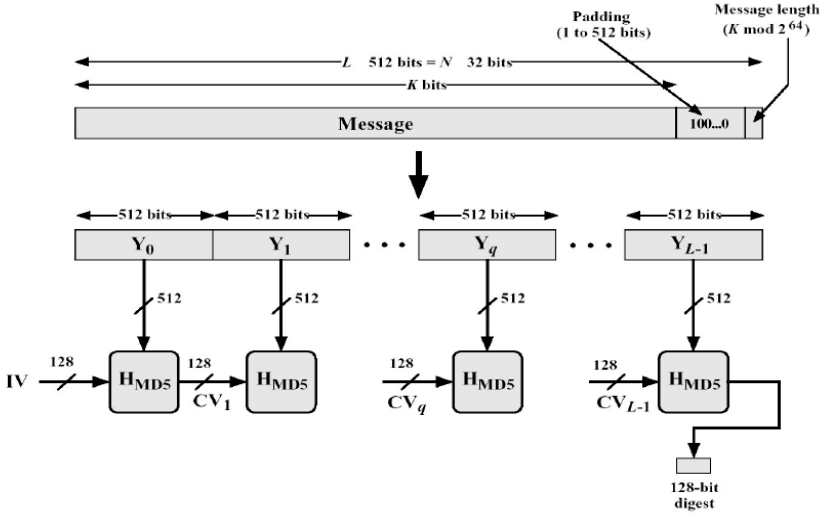
将数据（如一段文字）运算变为另一固定长度值，是散列算法的基础原理。

二、 总体结构

MD5使用little-endian，输入任意不定长度信息，以512位长进行分组，生成四个32位数据，最后联合起来输出固定128位长的信息摘要。

MD5算法的基本过程为：求余、取余、调整长度、与链接变量进行循环运算、得出结果。

总体流程示意图如下



三、 模块分解

将一个大小为 b bits 的信息作为输入，想得到它的信息摘要，在这里 b 是任意非负整数。我们将该信息写成下面这种形式：

m_0 m_1 ... m_{b-1}

要计算这个信息的信息摘要，需要以下五个步骤：

(1) 填充标识位

在原始消息数据尾部填充标识100…0，填充后的消息位数$L \equiv 448 (\mathrm{mod}\ 512)$。

至少要填充1个位，所以标识长度为1到512位。

(2) 填充长度

再向上述填充好的消息尾部附加原始消息的位数的低64位，最后得到一个长度L是512位整数倍的消息。由于位数b大于2^64可能性很小，只需要填充位数的低64位。这些位填充时，分成高低32位来填充，并且先填充的是低32位。

把填充后的消息结果分割为L个512位的分组：$Y_0, Y_1, \ldots, Y_{L-1}$。结果也表示成N个32位长的字$M_0, M_1, \ldots, M_{N-1}$，$N = L \times 16$。

(3) 初始化MD缓冲区

寄存器(A, B, C, D)置16进制初值作为初始向量IV，并采用小端存储(little-endian)的存储结构：

- *A* = 0x67452301
- *B* = 0xEFCDAB89
- *C* = 0x98BADCFE
- *D* = 0x10325476

Word A	01	23	45	67
Word B	89	AB	CD	EF
Word C	FE	DC	BA	98
Word D	76	54	32	10

(4) 在大小为16个字的块中处理消息

1.轮函数

我们先定义4个使用的辅助生成函数(轮函数)，输入为3个32位的数和输出为1个32位的数，相应的定义如下：

轮次	Function *g*	*g*(*b*, *c*, *d*)
1	$F(b,c,d)$	$(b \wedge c) \vee (\neg b \wedge d)$
2	$G(b,c,d)$	$(b \wedge d) \vee (c \wedge \neg d)$
3	$H(b,c,d)$	$b \oplus c \oplus d$
4	$I(b,c,d)$	$c \oplus (b \vee \neg d)$

2.T表的生成

T表包含64个元素，决定定义如下

$$T[i] = \text{int}(2^{32} \times |sin(i)|)$$

其中int为取整函数，sin为正弦函数，以i为弧度输入。

经过运算，T表生成如下：

T[1.. 4] = { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }

T[5.. 8] = { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }

T[9..12] = { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }

T[13..16] = { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }

T[17..20] = { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }

T[21..24] = { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }

T[25..28] = { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }

T[29..32] = { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }

T[33..36] = { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }

T[37..40] = { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }

T[41..44] = { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }

T[45..48] = { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }

T[49..52] = { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }

T[53..56] = { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }

T[57..60] = { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }

T[61..64] = { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

3. 压缩算法

a. 总体压缩过程

以512位消息分组为单位，每一分组Yq（q = 0，1，…，L-1）经过4个循环的压缩算法，表示为(IV为初始向量)：

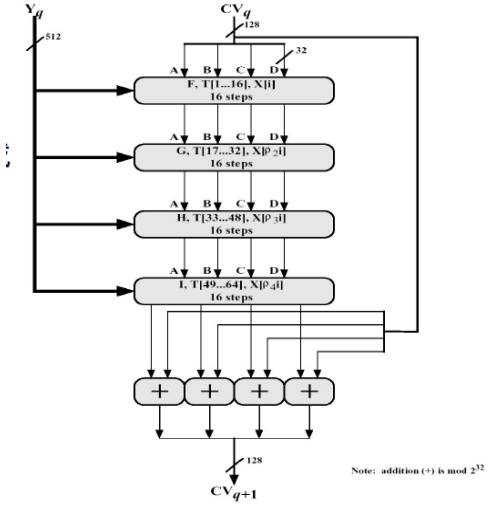
$$CV_0 = IV$$

$$CV_i = H_{MD5}(CV_i - 1 , Y_i)$$

输出结果：

$$MD = CV_L$$

总体压缩过程如图所示



b. MD5压缩函数H_{MD5}

MD5压缩函数H_{MD5}从CV输入128位，从消息分组输入512位，完成4轮循环后，输出128位，用于下一轮输入的CV值。每轮循环分别固定不同的生成函数F, G, H, I，结合指定的T 表元素T[]和消息分组的不同部分X[]做16次运算，生成下一轮循环的输入。总共有64次迭代运算。

每轮循环中的一步运算逻辑如下

$$a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) <<< s)$$

其中，

a，b，c，d ： MD缓冲区(A, B, C, D) 的当前值；

g：轮函数*(F, G, H, I* 中的一个*)*；

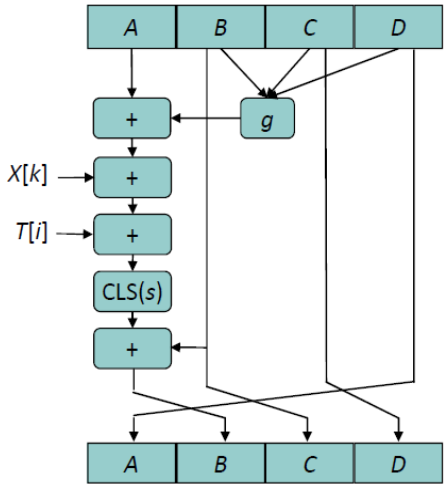
<<<*s* ： 将32位输入循环左移(CLS) *s* 位；

X[*k*] ： 当前处理消息分组的第*k*个32位字，即$M_{q\times16+k}$；

T[*i*]： T 表的第*i*个元素，32位字；

+ ： 模2^{32}加法。

运算逻辑可用下图表示：



各轮迭代中*X*[*k*]之间的关系：

第1轮迭代：*X*[*j*] , *j* = 1..16.

顺序使用*X*[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]

第2轮迭代：*X*[2(*j*)], 2(*j*) = (1 + 5*j*) mod 16, *j* = 1..16.

顺序使用*X*[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12]

第3轮迭代：*X*[3(*j*)], 3(*j*) = (5 + 3*j*) mod 16, *j* = 1..16.

顺序使用*X*[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]

第4轮迭代：*X*[4(*j*)], 4(*j*) = 7*j* mod 16, *j* = 1..16.

顺序使用*X*[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

各次迭代运算采用的左循环移位的*s*值：

s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }

s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }

s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }

s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

(5) 输出结果

经过迭代运算之后，将缓冲区中的 A,B,C,D 四个寄存器值拼在一起，以 A 为低位，D 为高位。输出结果以 16 进制的字符表示。

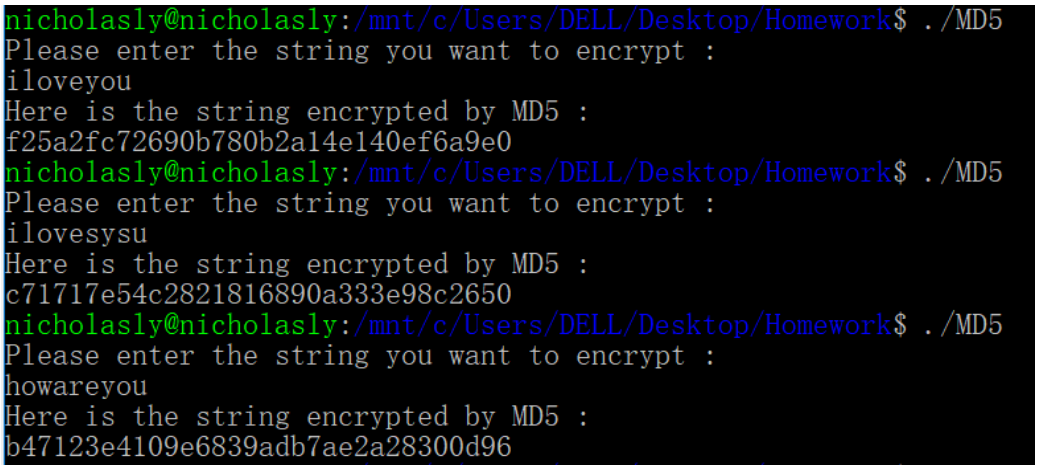
四、 数据结构和类 C 语言算法过程

在这次作业中，我用 C++实现了 MD5 算法。在数据结构方面，我采用了数组存储了的 T 表和位移表 *s*，便于数据的访问。另外在程序中调用了 string 和 vector 两个库，其中 vector 用于存储消息数据，分别使用了 unsigned char 和 unsigned int 类型，vector 的 push_back()函数在用于消息填充十分方便。

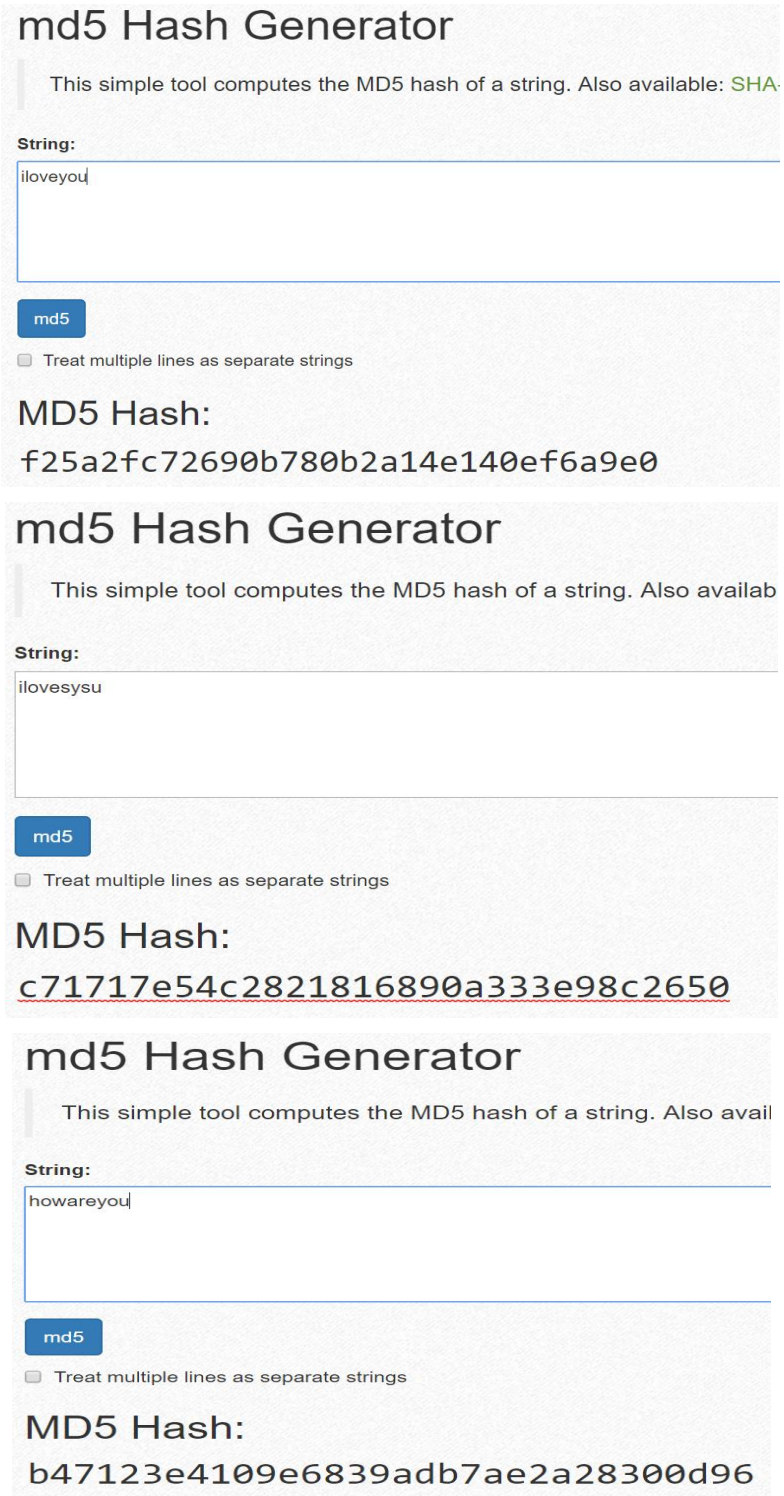
我算法实现步骤如下：

1. 用 string 存读入字符串；
2. 用 vector<unsigned char>存入 string 中的数据，并用 push_back()进行 10….0 填充和长度填充。其中，在长度填充时，考虑到 int 为 32 位，数据长度一般不会大于 2^{23}，填入长度高 32 位全部设为 0，低 32 位设为正常的长度值。这里还需要注意用小端规则存储长度值；
3. 将 vector<unsigned char>转换成 vector<unsigned int>，其中 unsigned char 为 8 位，unsigned int 为 32 位，转换时需要进行移位后二进制加法，使得 unsigned int 的二进制形式正确表示信息。
4. 对 vector<unsigned int>进行分组，用 vector< vector<unsigned int>>来存取分组，一个分组 vector<unsigned int>有 16 个 unsigned int 数；
5. 初始化缓冲区中的 A,B,C,D 四个寄存器，设计哈希函数，根据以上模块对每一个分组进行迭代计算，里面需要用到 T 表和移位数组 s 以及根据迭代次数不同而不同的轮函数，最终 A,B,C,D 四个寄存器得到最后的迭代结果，结果类型均为 unsigned int；
6. 将 A,B,C,D 四个寄存器拼接起来，并转换成 vector<unsigned char>形式以十六进制的字符输出。

下图是我编写程序的输出结果



为了验证这些结果的准确性，我在有 MD5 加密功能的网站上进行同样的加密：



对比我的程序的输出结果，发现加密的数据完全正确。