

MD5算法实验

数据科学与计算机学院

16340296 张子权

算法原理概述

MD5算法是一种hash算法用于确保信息传输完整和一致性，是一种hash函数，能将任意长度的消息转换为固定长度的散列码。这还是一种无密钥的hash函数，不使用密钥。

所谓哈希就是输出和输入能做到唯一对应，不同的信息对应不同的信息摘要，这要就能保证消息的完整性，验证消息是否被破坏。若消息被改变，则对应的信息摘要也会被改变。

MD5的主要方法就是将信息以512位来进行分组处理，对每一个分组进行HMD5压缩，得出结果后再用于下一个分组的压缩，最后得出的结果即是MD5的值。

总体结构



- 大致结构
 1. 输入信息（任意长度的信息）
 2. 对信息进行填充
 3. 根据填充后的信息进行分块
 4. 初始化缓冲区，然后进行迭代
 5. 用缓存区的cv对每一个分块进行HMD5压缩
 6. 输出缓冲区内的内容

模块分解

- 输入信息 可输入任意长度信息，无论字符串，文件流等等
- 信息填充：
 1. 先得到信息的长度。
 2. 在尾部填充长度P的b标识100...0，其中 $K+P \equiv 448 \pmod{512}$ 。
 3. 在尾部再填充64位的信息的长度的值。
 4. 由上述可得组数为 $(K/512) + 1$
- 信息分块
 1. 将填充后的信息分割成多个512位的分组。
 2. 因为缓冲区分4个即一个部分为32位，所以每组的存储使用int存储分成16个int。
 3. 每次进入循环先取16个int作为数组传进去压缩。
- 初始化缓冲区

1. 初始化一个128bit的缓冲区，分成4个寄存器A,B,C,D进行迭代。
2. 因为每个寄存器都是32bit，正好是一个int，所以用int存储。

- 循环HMD5压缩

1. 每轮压缩有64次迭代，1-16次使用F函数，17-32为G，33-48为H，49-64为I。
2. 生成函数所生成的值结合T表对应元素+消息的某部分运算。
3. 详细来所就是 $a = a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i]) \ll s)$ 缓冲区轮换: $(B, C, D, A) \leftarrow (A, B, C, D)$
4. 然后得出的输出结果作为下一次的输入，所以直接使用寄存器存储（全局变量）



- 输出缓冲区

1. 循环结束后需要输出寄存器内容，即128bit，32个十六进制数。
2. 将每个寄存器32位分成4个字节。每个字节变为2个十六进制进行计算。
3. 因为C++是小端模式，所以注意要从后往前输出，且每个字节内还是按大端的。
4. 转十六进制可以用到itoa，直接变为字符串。

数据结构

逻辑结构

集合结构

物理结构

- MD5类
 - int 一维数组存储
 - 算法使用的128bit寄存器，T表，用于左移的S表
 - string
 - 输入的消息表示以及输出信息摘要表示。
 - int 整型
 - 用于表示存储的32位，存储信息。

C语言源代码

```
#include <iostream>
#include <string>
#include <string.h>
using namespace std;

#define shift(x, n) (((x) << (n)) | ((x) >> (32 - (n)))) //右移的时候，高位一定要补零，而不是补充符号位
#define F(b, c, d) ((b & c) | (~b & d))
#define G(b, c, d) ((b & d) | (c & ~d))
#define H(b, c, d) (b ^ c ^ d)
```

```

#define I(b, c, d) (c ^ (b | ~d))

//T表, 可用int(2^32|sin(i)|)生成
const unsigned int T[] = {
    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501, 0x698098d8,
    0x8b44f7af, 0xfffff5bb1, 0x895cd7be, 0x6b901122, 0xfd987193,
    0xa679438e, 0x49b40821, 0xf61e2562, 0xc040b340, 0x265e5a51,
    0xe9b6c7aa, 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed, 0xa9e3e905,
    0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a, 0xffffa3942, 0x8771f681,
    0x6d9d6122, 0xfde5380c, 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60,
    0xbefbfc70, 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665, 0xf4292244,
    0x432aff97, 0xab9423a7, 0xfc93a039, 0x655b59c3, 0x8f0ccc92,
    0xffefff47d, 0x85845dd1, 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314,
    0x4e0811a1, 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};

//每次迭代运算左循环移位的s值
const unsigned int S[] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17,
22,
                                5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
                                4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16,
23,
                                6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15,
21};

class MD5
{
public:
    string getMD5hash(string str);
    MD5();

private:
    unsigned int *padding(string str); //进行数据填充
    void mainloop(unsigned int *temp);
    void hmd5(unsigned int *yi);
    string bin2hex(int bin);
    unsigned int groupnum;
    unsigned int A, B, C, D;
};

MD5::MD5()
{
    A = 0x67452301;
    B = 0x0EFCDA89;
    C = 0x98BADCFE;
    D = 0x10325476;
    groupnum = 0;
}

//获得MD5hash
string MD5::getMD5hash(string str)
{

```

```

    unsigned int *groups = padding(str); //填充
    mainloop(groups); //分块、缓冲区初始化、循环压缩
    return bin2hex(A) + bin2hex(B) + bin2hex(C) + bin2hex(D);
}

//进行数据填充
unsigned int *MD5::padding(string str)
{
    int slength = str.length(); //计算输入字符串的长度

    // 计算总共的组数, 64个字节一组
    unsigned int fullsize = (slength + 8) / 64 + 1;
    groupnum = fullsize; //记录分组数量便于循环划分

    unsigned int *strbyte = new unsigned int[fullsize * 16]; //使用整数存储, 一个整数4个字节

    memset(strbyte, 0, fullsize * 16); // 其它位补0。
    for (unsigned int i = 0; i < slength; i++)
    {
        strbyte[i >> 2] |= (str[i]) << ((i % 4) * 8); //一个字符1个字节, 一个整数存储4个字符
    }

    strbyte[slength >> 2] |= 0x80 << ((slength % 4) * 8); // 和上面一样, 在尾部补上1
    strbyte[fullsize * 16 - 2] = slength * 8; //消息尾部附加str长度的低64位。

    return strbyte;
}

//对各个分组进行HMD5压缩, 最后输出的CV1即是结果
void MD5::mainloop(unsigned int *temp)
{
    unsigned int loop = groupnum; //得到分组数
    for (unsigned int i = 0; i < loop; i++)
    {
        unsigned int group[16];
        memcpy(group, &temp[16 * i], 16); //获得各个分组, 并进行HMD5压缩
        hmd5(group);
    }
}

//HMD5压缩函数
void MD5::hmd5(unsigned int *yi)
{
    unsigned int gresult, k;
    unsigned int bufa = A;
    unsigned int bufb = B;
    unsigned int bufc = C;
    unsigned int bufd = D;
    for (unsigned int i = 0; i < 64; i++)
    {
        //各轮循环迭代
    }
}

```

```
    if (i < 16)
    {
        gresult = F(bufb, bufc, bufd); //生成函数F,G,H,I,下同
        k = i;                          //对应的T表元素索引
    }
    else if (i < 32)
    {
        gresult = G(bufb, bufc, bufd);
        k = (1 + 5 * i) % 16;
    }
    else if (i < 48)
    {
        gresult = H(bufb, bufc, bufd);
        k = (5 + 3 * i) % 16;
    }
    else
    {
        gresult = I(bufb, bufc, bufd);
        k = (7 * i) % 16;
    }


    //循环轮换
    unsigned temp = bufd;
    bufd = bufc;
    bufc = bufb;
    bufb = bufb + shift((bufa + gresult + yi[k] + T[i]), S[i]);
    bufa = temp;
}
//循环结束，生成新的CV作为下一次的输入
A += bufa;
B += bufb;
C += bufc;
D += bufd;
}

//二进制转十六进制
string MD5::bin2hex(int bin)
{
    int num = 0;
    string res;
    for (int i = 0; i < 8; i++) //将32位划分为4位，即一个十六进制数
    {
        num = ((bin >> i * 4) % (1 << 8)) & 0xf; // 得到每一个十六进制数。
        char hex[2];
        itoa(num, hex, 16);
        res += hex;
    }
    return res;
}

int main(int argc, char const *argv[])
{
    MD5 md5 = MD5();
    cout << md5.getMD5hash("abc");
}
```

```
/* code */  
return 0;  
}
```

编译运行结果

 编译运行

根据网上的MD5加密解密，和自己生成的结果是完全符合的。

test1 test2 test3

实验思考

本次的MD5算法还算是比较简单的了，较上次的DES算法来说。而且顺便也学习了一种哈希方法，现在经常都能听到哈希函数的应用，如区块链什么的，当自己用过一遍之后才明白是怎么回事。只是很奇怪这是怎么保证符合哈希的，即无碰撞，但根据实例来看MD5也不能保证没有完全的碰撞，只是只能找到强无碰撞。

加密的过程总让我想起区块链那种，每次生成的时候使用上一区块的哈希加进来做哈希，就好像让新生的区块附上原来的印记的感觉。从而保证一部分改变而全部改变。