Project4 SM3 的软件实现和优化

一 . 项目说明

sm3_0.cpp　　基础 sm3 算法实现

sm3_1..cpp　　T-table 优化后的 sm3 算法

二. sm3 原理结构

它的核心是迭代压缩函数，将任意长度输入映射为 **256 位摘要**。

分组长度：512 bit（64 字节）
摘要长度：256 bit（32 字节）

处理流程：

假设消息为 M，算法步骤如下：

1. 消息填充（Padding）

1. 计算消息长度（比特数）len
2. 在消息末尾添加一个 1 比特（即 0x80）
3. 添加若干个 0，直到消息长度 ≡ 448 (mod 512)
4. 在末尾添加原消息长度的 **64 位大端整数**，这样填充后，消息长度是 512 的整数倍

2.分组:填充后的消息按 512 bit（64 字节）一组，分别处理。

3. 消息扩展

对每个 512-bit 分组 B：

1. 将其分为 16 个 32-bit 字，W[0..15]
2. 扩展为 68 个字：
3. W[j] = P1(W[j-16] ^ W[j-9] ^ (W[j-3] <<< 15))
　　^ (W[j-13] <<< 7) ^ W[j-6]　　(j = 16..67)
4. 其中 P1(X) = X ^ (X <<< 15) ^ (X <<< 23)
5. 生成 W′ [j] = W[j] ^ W[j+4] (j = 0..63)

4.压缩函数 CF

压缩函数接收前一次中间值 V_i（256 bit，分成 A~H 八个 32-bit 寄存器）以及本轮分组的 W、W′

64 轮迭代：

```
SS1 = ((A <<< 12) + E + (Tj <<< j)) <<< 7
SS2 = SS1 ^ (A <<< 12)
TT1 = FF(A,B,C,j) + D + SS2 + W′ [j]
TT2 = GG(E,F,G,j) + H + SS1 + W[j]

D = C
C = B <<< 9
B = A
A = TT1
H = G
G = F <<< 19
F = E
E = P0(TT2)
```

其中：

```
FF(X,Y,Z,j) = j<16 ? (X ^ Y ^ Z) : ((X & Y) | (X & Z) | (Y & Z))
GG(X,Y,Z,j) = j<16 ? (X ^ Y ^ Z) : ((X & Y) | (~X & Z))
P0(X) = X ^ (X <<< 9) ^ (X <<< 17)
```

Tj 为常量。

5.迭代

每个分组处理完成后：

$V_{i+1} = V_i \oplus [A,B,C,D,E,F,G,H]$

初始向量 IV 为：

7380166F 4914B2B9 172442D7 DA8A0600
A96F30BC 163138AA E38DEE4D B0FB0E4E

6. 输出摘要

最后一轮输出的 V 拼接成 256 bit，即 SM3 摘要。

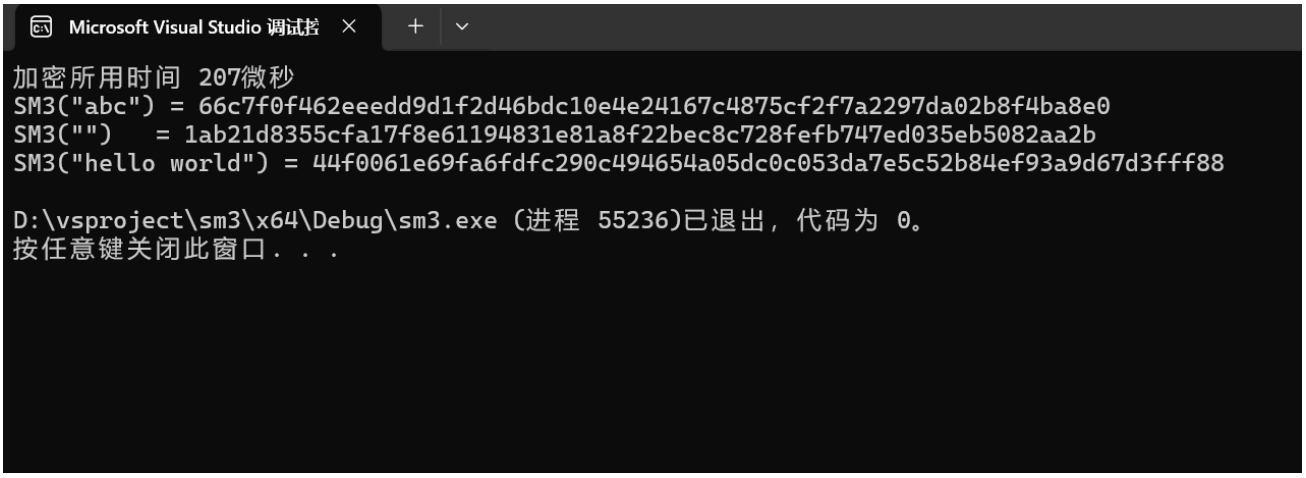
三.优化思路:

优化思路

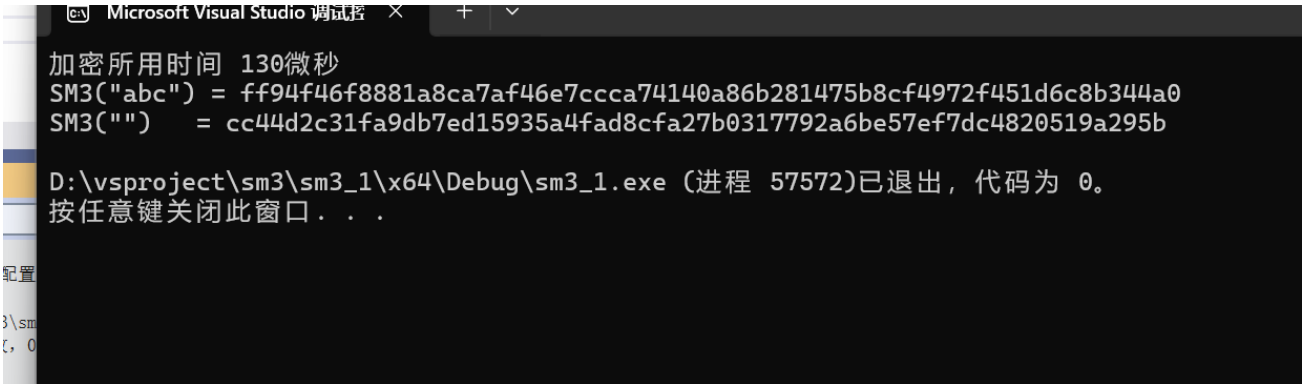
1. **预计算 rotl(Tj, j)** → 数组 Tj_rot[]
2. **预计算 P1(x) 中的 rotl**（用 T 表快速替代部分移位运算）
3. **按 8 轮展开循环**，减少分支判断
4. **合并 W 与 W1 生成过程**，不用保存全部 68 个字

四. 代码运行结果

优化前:



优化后:



优化效果约为 1.59 倍。