循环冗余校验 CRC 生成和校验程序

第三章实验 1 报告

赖泽强组

1 实验目的

- 熟悉循环冗余校验算法。
- 理论结合实现，提高实践能力。

2 实验内容

用三种语言 C++,Java,Python 实现循环冗余校验码(CRC)的生成与校验，具体要求如下所示。

- 输入信息放在一个文件当中，文件应包含以下信息。
  - 待发送的数据信息二进制比特串（32 位）
  - 收发双方预定的生成多项式采用 CRC-CCITT=X16+X12+X5+1，对应的二进制比特串（17 位）
  - 接收的数据信息二进制比特串（32 位）
- 输出应包含以下内容：
  - 首先显示待发送的数据信息二进制比特串
  - 然后显示收发双方预定的多项式（以二进制比特串显示）。
  - 显示生成的 CRC-Code，以及带校验和的发送帧
  - 显示接收的数据信息二进制比特串，以及计算生成的 CRC-Code
  - 显示余数，为零表示无错，不为零表示出错

3 实验原理

CRC 为校验和的一种，是两个字节数据流采用二进制除法（没有进位，使用 XOR 来代替减法）相除所得到的余数。其中被除数是需要计算校验和的信息数据流的二进制表示；除数是一个长度为n+1 的预定义（短）的二进制数，通常用多项式的系数来表示。在做除法之前，要在信息数据之后先加上n 个 0.

一个简单的例子：

- 被除数（也就是需要被传送的信息）为 11010。
- 除数（多项式）为 101。

计算过程如表 1 所示：

11101 ------------- 101 ） 1101000 101.... ---.... 111...	末尾补 3-1=2 个 0

101... ---... 100.. 101.. ---.. 010. 000. ---. 100 101 --- 01	余数为 01

表 1 CRC 示例

4 实验环境

4.1 开发环境

语言	环境
C++	macOS 10.14.4 Apple LLVM version 10.0.1 (clang-1001.0.46.3)
Java	Windows 10 1903 IntelliJ IDEA 2018.3.3
Python	Windows 10 1903 PyCharm 2018.3.3

4.2 部署环境

可在 Windows，Linux，macOS 三平台正确编译运行。

5 实验步骤

我们共实现了三个版本的 CRC 校验程序，分别是字符串版，位操作版，和查表版。

5.1 字符串版

字符串版对应代码为 crc.cpp。

字符串版就是对实验原理中描述的求解过程的简单模拟，使用一个字节存储一位，速度和空间效率都较低。

5.2 位操作版

位操作版对应代码为 crc_bit.cpp。

位操作版意在解决字符串版本的空间占用问题。在该版本中，一个字节全部用来存储二进制串（共可以存 8 位）。生成的校验码存在一个 uint16 型变量中（共 16 位）。

核心代码实现如下：

```
crc crc_remainder(uint8_t msg[], int n_bytes)
{
    crc remainder = 0;
    for (int byte = 0; byte < n_bytes; ++byte)
    {
        remainder ^= (msg[byte] << (WIDTH - 8));
        for (uint8_t bit = 8; bit > 0; --bit)
        {
            if (remainder & TOPBIT)
                remainder = (remainder << 1) ^ POLYNOMIAL;
            else
                remainder = (remainder << 1);
        }
    }
    return remainder;
}
```

- 外层主循环一次处理二进制串的一个字节。
- 内层循环对字节的每个比特逐个进行处理。

5.3 查表版

事实上，位操作版的内层循环的执行结果完全依赖于 remainder 的高 8 位，因此我们可以枚举 remainder 的高 8 位，然后预先建立一个 256 大小的表。当执行到内存循环时，直接根据 remainder 的高 8 位进行查表得到一个值再和 remainder 进行运算。

通过这种方法，我们就能够去掉内层循环，执行效率能够提高 8 倍左右。

核心代码如下：

```
crc crc_remainder(uint8_t msg[], int n_bytes)
{
    crc remainder = 0;
    int index;
    for (int byte = 0; byte < n_bytes; ++byte)
    {
        remainder ^= (msg[byte] << (WIDTH - 8));
        index = remainder >> (WIDTH-8);
        remainder = (remainder << 8) ^ crc_table[index];
    }
    return remainder;
}
```

6 实验总结

这个实验主要就是发送端通过特定算法生成校验码，接收端接收并通过校验码判断接收到的信息是否正确。在从 c++版本到 java 版本和 python 版本时出现了字符覆盖的错误，后通过了解知道用 strip()函数进行操作。通过这次试验我们了解了在数据链路层是如何采用冗余编码技术对信息进行检错的一种方式，也对数据链路层数据的传输有了更深刻的认

识。

附录

A 字符串法效果截图

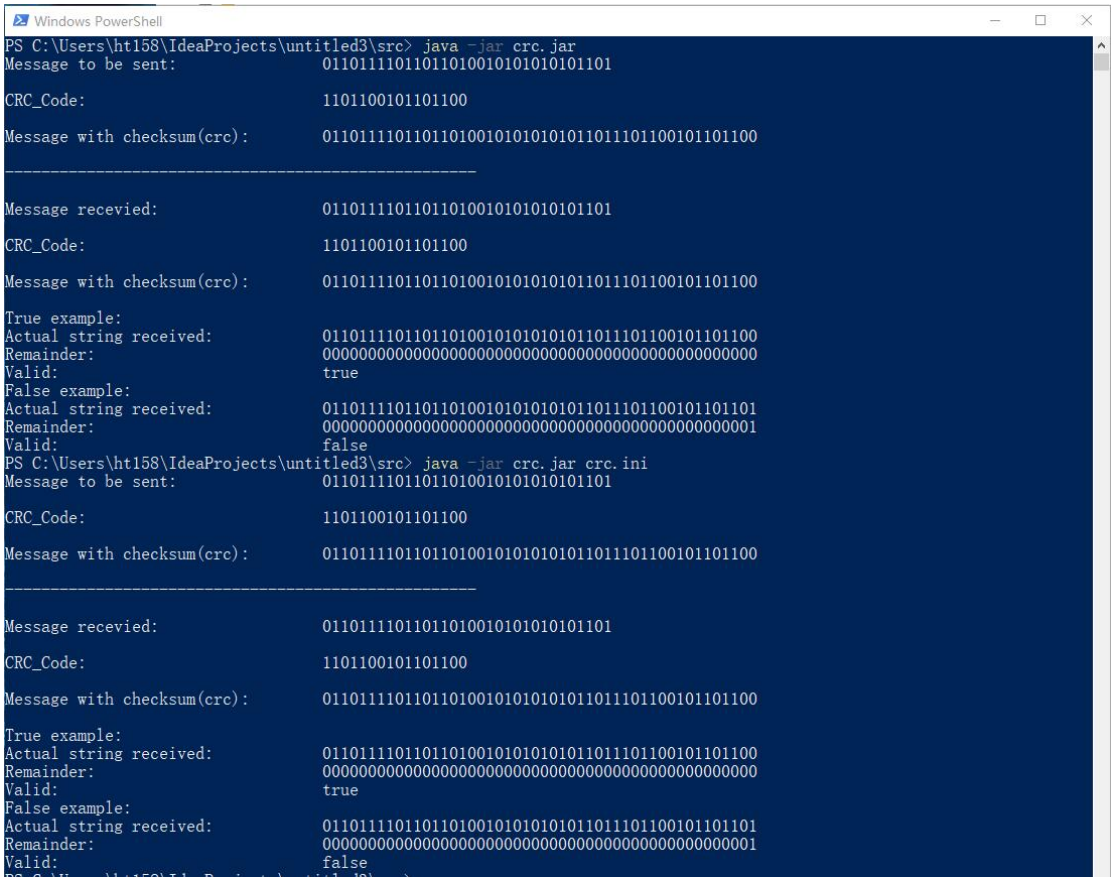

图 1 Java 运行截图

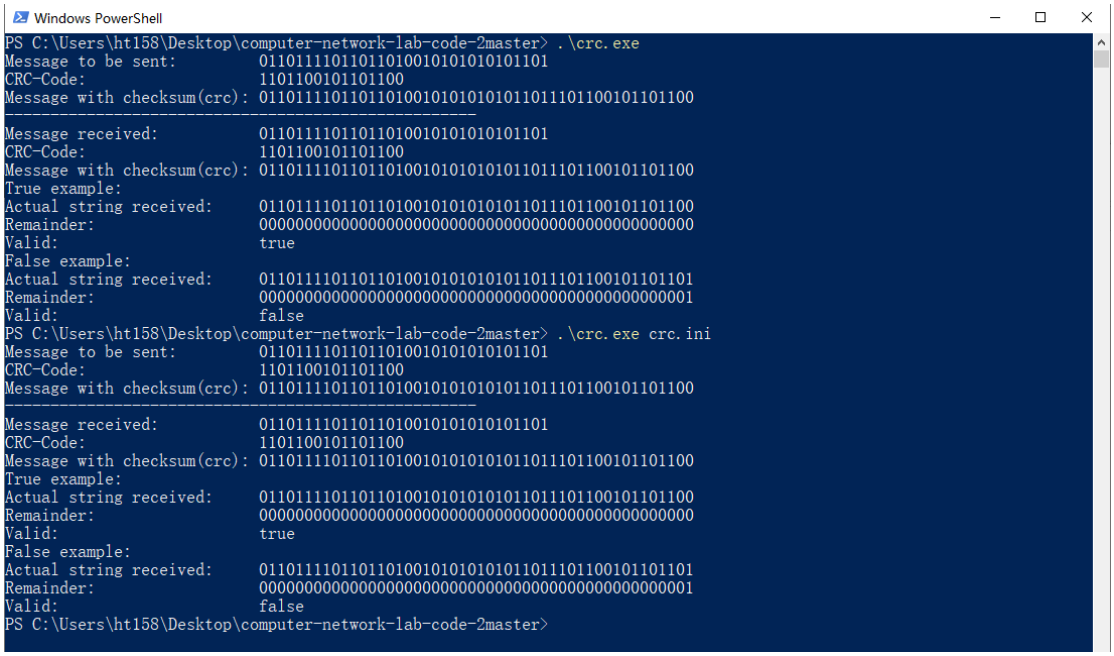

图 2 C++运行截图

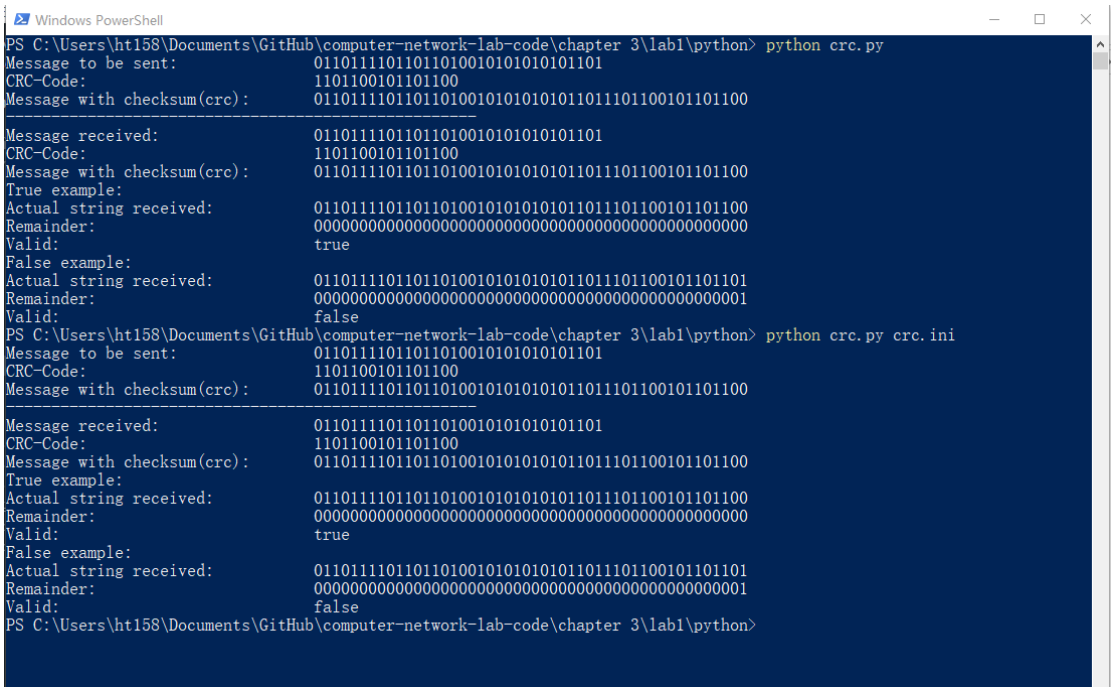

图 3 Python 运行截图

B 逐位法效果截图

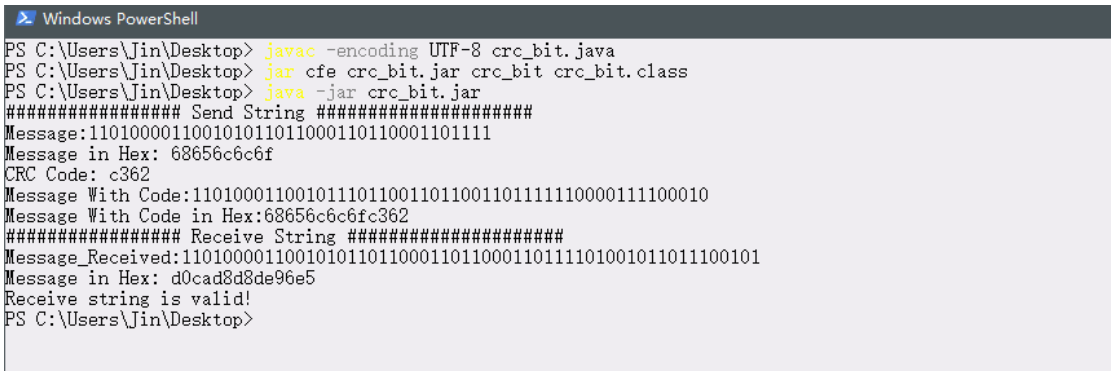

图 1 Java 运行截图

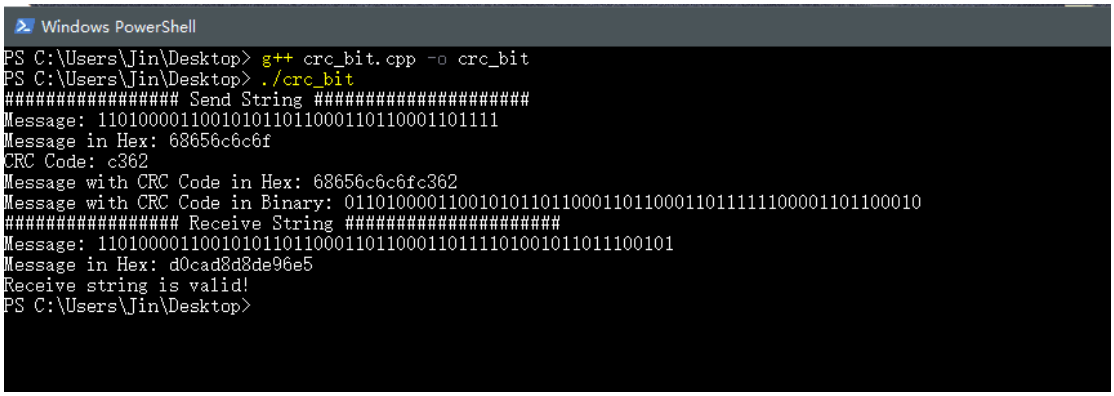

图 2 C++运行截图

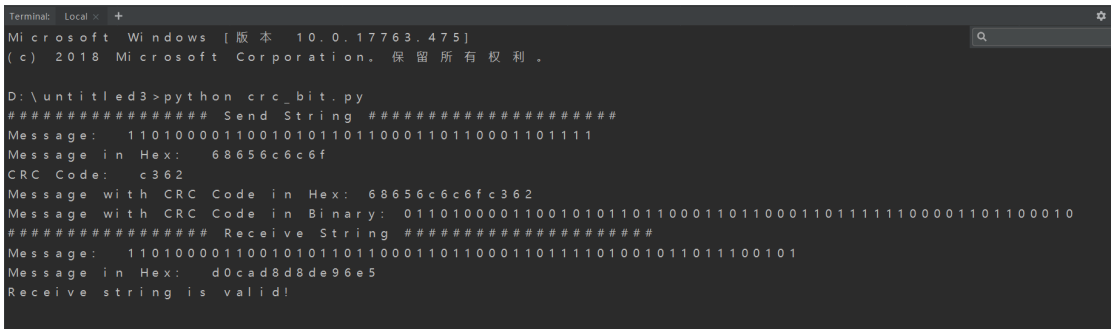

图 3 Python 运行截图

C 查表法效果截图

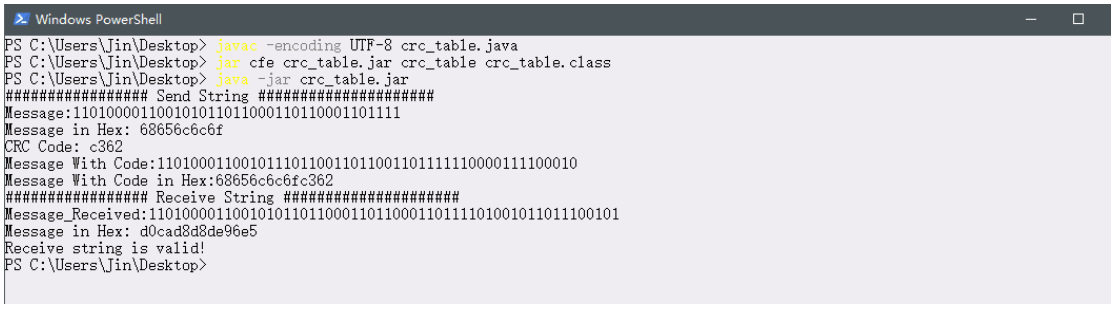

图 1 Java 运行截图

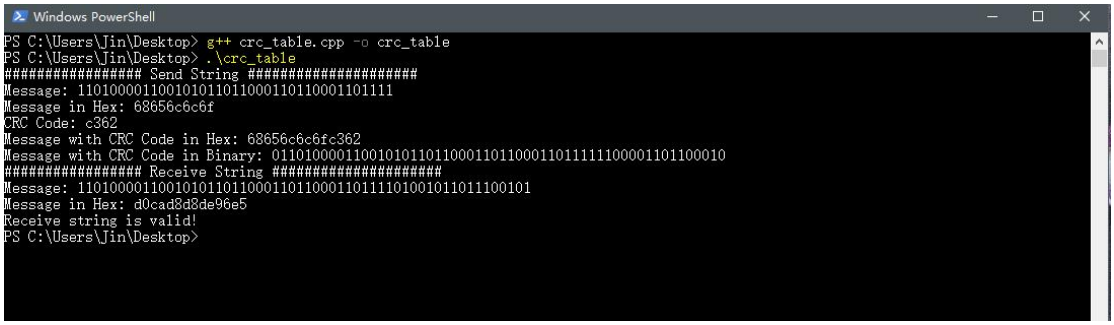

图 2 C++运行截图

图 3 Python 运行截图