

LS11 数据格式分析总结和解压缩代码

numdisp

2010-4-16

版本修订号 0.2

声明

本作仅为学习交流，作者对阅读本文所产生的任何直接或间接后果概不负责。文中各公司名称、游戏名称及其缩写对应的注册商标等均为其所有者版权所有。

摘要

LS11 格式的数据文件存在于 KOEI®开发的许多经典游戏中，比如大航海时代系列和三国志系列等等。本文在前人的基础上对 LS11 的详细格式进行了探讨，总结并阐述了 LS11 文件的具体格式，数据压缩方法，还原算法等等。并给出了完整的解压缩程序和源代码。

前言

在网络游戏盛行的今天，许多曾经在 DOS 时代和 WIN95 时代大放光彩的经典游戏依然保持着相当大的玩家群。除了游戏本身开放的设定和耐玩性之外，许多热心网友们制作的修改版本甚至重新开发版本也是其保持长久魅力的重要原因。游戏的修改或是重新制作少不了解读原游戏的许多数据和资源，比如其图片元素，地图数据，人物设定等等。但是，几乎没有一款游戏的资源和数据是以常见的已知数据格式存储的，提取这些资源和数据就需要了解原游戏数据文件的格式。出于种种原因，几乎所有的游戏都有自己特有的数据格式。而这些格式又往往只在游戏开发公司内部流通而不开放，这就给广大游戏修改爱好者们的工作带来了很大的困难。经过许多传奇式的游戏黑客们的不懈努力，大量游戏的数据格式被破解出来，从而带来了形形色色的修改工具。

日本光荣株式会社（KOEI）开发了许多经典的游戏，比如大航海时代系列、三国志系列等等。这些游戏的很多数据均以一种特殊的类 LZW 压缩方式存储。如果用十六进制编辑器打开这些文件，经常会发现开头的几个字节都是显示的“LS11”、“LS12”或是“LS10”等。为了方便起见，我们姑且将这些数据格式统称为“LS11 格式”。这种格式之所以重要，是因为它存在于大量 KOEI 的游戏中，比如大航海时代 2、三国志英杰传、曹操传和孔明传等。要想修改这些游戏的数据，或是从中提取游戏资源，就必须首先搞清楚如何解压缩（或者说解密）这些数据文件以得到其真正的文件内容。

据笔者所知，最早解密 LS11 格式文件的资料是 van 在轩辕春秋文化论坛的设计与修改区（<http://www.xycq.net/forum/forum-48-1.html>）上发表的“英杰传系列LS格式压缩文件解析”一文（van，2004-11-26）。随即以Maxwell为代表的等人在“ls11 格式详解”一文（Maxwell，2004-12-2）中对此进行了比较详细的探讨并给出了部分关键解密代码。在该帖的讨论中，gameplore给出了极为详尽的加密、解密程序和源代码。更有boylinming等人对部分代码进行了优化以加快数据处理速度。Maxwell等人的帖子十分宝贵。然而，在此文中除了主要参与讨论的几位网友外，其它人对该帖讨论的东西似乎并不是很明白，而且，由于论坛的原因，帖中出现的许多代码并不是很容易直接使用。相信看完该帖后，很多没有数据加解密经验的网友可能和我一样，对如何处理LS11 格式还是一知半解。

随着时间的流逝，当年许多热心参与此工作的人现在都很少出现在论坛上了，联系方式也不为人知。这就给现在希望详细了解 LS11 格式的朋友造成了一些不便。为了方便大家，同时也为了减少重复劳动，我在这篇文章中尽我所能用比较通俗的方式解析一下 LS11 格式并给出完整的解密程序。

一、字典的基本概念

数据压缩的原理这里不打算长篇大论的介绍了，这样的书籍和资料相信随便到图书馆或者网上就能得到非常详细的信息。这里只稍微引入一下字典的概念。

许多数据压缩或者加密时，都有一个叫做“字典”的东西。比如下面的字符串：

THIS IS A STRING, THAT IS ANOTHER STRING.

重复的单词有 IS 和 STRING，如果我们把 IS 用编号 1 代替，STRING 用编号 2 代替，上面的字符串就变为

THIS 1 A 2, THAT 1 ANOTHER 2.

这样，数据便得到了压缩（字符串总长度比原来短了）。而上面的 1、2 对应的 IS 和 STRING 就构成了一个“字典”。1 和 2 是字典的索引项，IS 和 STRING 就是字典的内容。在解压缩时，程序只要读到 1 或者 2 这两个索引值，便会到字典里寻找 1 或者 2 对应的真正字符串（想象一下用新华字典根据读音查询汉字写法的情形），然后再反过来用 IS 来代替 1，用 STRING 来代替 2，这样数据便得到了还原（解压缩）。大多数情况下，字典总是以依次增加的数字来做为索引号的，所以在真正保存字典时，诸如 1 和 2 这样的序号值就无需再额外开辟存储空间了，只需要把字典的内容按照其编号依次排列即可。于是，“字典”这个词通常也用来泛指“字典”里的内容。有些压缩算法需要保存字典的内容，有些压缩算法却能够自动还原字典而不需要保存字典的内容。LS11 属于前者，它有一个 256 项的字典项，保存在紧接着文件头的部分。

二、LS11 的文件格式

LS11 的格式的文件总体上来说可以分为四大块：文件头，字典，数据信息段，和数据段。下面的表格给出了一个文件结构的概览。

表 1. LS11 文件格式概览

字节范围（十六进制）	总字节数（十进制）	含义	解释
0x0000 - 0x000f	16	文件头	前 4 个字节为 LS11（或 LS12、LS10 等）的 ASCII 码值，后 12 个字节为空（0x00）。
0x0010 - 0x00f0	256	字典	共 256 个字节，每个字节代表字典中的一项。解密时将根据后面解出的索引号从中提取对应的真实数据。
0x0110 - XXXX	$N \times 12 + 4$	压缩（加密）信息（压缩后长度，原文长度，压缩数据起始地址）	前面的 $N \times 12$ 个字节可以分为 N 个子块，每块 12 个字节，存储着解压缩（解密）所需的信息。最后 4 个字节总为空（0x00）。
XXXX - 文件尾	不定	压缩（加密）后的数据	压缩（加密）后的二进制流。压缩方式是以二进制位为单位的。

表 1 中列出的文件头和字典项的长度是固定的。而第三部分我称之为“数据信息段”，第四部分我称为“数据段”。LS11 整个文件共用一个字典，但是数据却并不是作为一个整个二进制流来压缩的。为了提高压缩效率，它将原始数据根据需要分为 N 段，每段分别压缩，然后依次存储。所以上表中的数据信息段的长度并不是固定的，但却是有规律的，它的长度总等于 $N \times 12$ 再加上 4 个全部为空的字节（用来隔开数据信息段和紧邻的数据段）。如果原始数据是作为一整段压缩的，即 N 等于 1，数据信息段便只有 $12 + 4 = 16$ 个字节。而这 12 个字节又可分为三部分，每部分 4 个字节，分别表示的含义是：压缩后的二进制流长度（以字节为单位），原始二进制流的长度（以字节为单位），和压缩后的数据在该文件中的起始地址（从文件的最开始处算起，以字节为单位）。

紧接着数据信息段的 4 个空字节后，就是依次存放的各段压缩数据了，也就是经过压缩后的字典索引项。如果得到了索引值，只要从字典中提取相应的字典项便可还原（解压）LS11 文件了。

三、LS11 的索引数据压缩方式

LS11 的数据压缩是以二进制位为单位的。其基于这样一种原理（参考 Maxwell 的帖子）：

任意一个二进制数总可以分解为两个 m 位二进制数的和，第一部分我这里称之为掩码，其形式必定是 $1\dots10$ 的形式，也就是说，前面的 m-1 位全是 1，最后一位是 0。而第二部分我称之为因子，是一个 m 位的二进制数（如果不足 m 位可在高位补 0 以达到 m 位）。举几个例子（摘自 Maxwell 的帖子）：

表 2. LS11 二进制数分解示例

原二进制数	分解后的位数(m)	分解后的掩码	分解后的因子
0	1	0	0
1	1	0	1
101	2	10	11
1001011	6	111110	001101

反过来说的话，如果一个压缩后的二进制流是：

11111110 00000110 0 1 1110 0001 0 0

那么可以按下面的步骤依次还原：

1. 扫描到第一个出现的 0，得到掩码 11111110，共 8 位，进而继续扫描下面的 8 位，得到因子 00000110，于是还原出第一个数为 $11111110 + 00000110 = 100000100$ ，即十进制的 260。
2. 继续扫描，得到掩码 0，共 1 位，扫描下面的 1 位，得到因子 1，于是还原出 $0 + 1 = 1$ ，即十进制的 1。

3. 继续扫描，得到掩码 1110 和因子 0001，还原出 $1110 + 0001 = 1111$ ，即十进制的 15。

4. 继续扫描，得到掩码和因子均为 0，还原出十进制的 0。

得到了上述的一系列数（260，1，15，0），它们应该是对应的字典索引值。LS11 的数据段正是以这种方式压缩的。

四、字典项提取规则

有了全部的字典索引值，直接从字典中提取字典内容就可以还原原始数据了。可是，看到上面的 4 个数（260，1，15，0），细心的读者会马上说，前面提到的 LS11 字典只有 256 项，那索引值 260 该如何处理？

LS11 的算法规定，如果还原得到的数 $x \in [0, 255]$ ，也就是说 $0 \leq x \leq 255$ 的话，那么这个值 x 就是真正的索引值，还原程序需要做的就是从字典中找第 x 项来代替这个数字就可以了。但是如果得到大于 256 的值（ $x > 256$ ），便执行一个复制（COPY）操作，COPY 的起始地址便是从原文的当前位置往回偏移 $\text{delta} = x - 256$ 个字节，COPY 的字节数 $\text{nc} = C + x2$ ，其中 C 是常量，在 LS11 算法里恒等于 3，而 $x2$ 就是紧接着 x 被还原出来的下一个数值（在上面的例子中就是紧接着 260 后面的 1）。对于 x 正好等于 256 的情况，如何处理目前尚不得而知。但是，根据下面详细叙述的 COPY 操作的算法，正确压缩的 LS11 数据是不可能解压得到正好等于 256 的 x 的（否则复制的起始地址将是一个无效地址）。我检查了许多 LS11 格式的数据，也的确没有出现 x 等于 256 的情况。

这里的 COPY 操作是还原字典项的关键。上面所说的复制 nc 个字节实际上采用的是一种“滑动窗口”式的办法，即每次复制一个字节，然后移动复制起始地址，这样重复 nc 次。这样的说法也许令人十分困惑。下面结合一个例子来详细说明这个 COPY 操作是如何进行的。

假设我们根据第三节的分解原理还原得到了一系列数 {3, 8, 5, 9, 12, 22, 260, 2, 15, 0}，而字典为：

索引值	字典项	索引值	字典项
0	A	10	
1	C	11	+
2	:	12	K
3	B	13	[
4	Y	14	'
5	T	15	H
6	S
7	=	22	\$
8	W	...	
9	<	255	*

现在读取第 1 个数，为 3，直接得到字典项为 B，这是第一项，所以到目前为止的原文为 B；
读取第 2 个数，为 8，直接得到字典项为 W，加在原文后面，原文变为 BW；

读取第 3 个数，为 5，直接得到字典项为 T，加在原文后，得到新的原文为 BWT

读取第 4 个数，为 9，直接得到字典项为<，新的原文为 BWT<

读取第 5 个数，为 12，直接得到字典项为 K，新的原文为 BWT<K

读取第 6 个数，为 22，直接得到字典项为\$，新的原文为 BWT<K\$

读取第 7 个数，为 260，大于 256，于是计算 $\text{delta} = 260 - 256 = 4$;

继续读取第 8 个数，为 2，于是计算 $\text{nc} = 2 + C = 2 + 3 = 5$ 。

现在在原文“BWT<K\$”的基础上，回退 $\text{delta} = 4$ 个字节，当前位置指向 T，开始复制。

问题出现了：按照“复制 nc 个字节”的说法，从 T 开始数起，只有“T<K\$”4 个字节，后面是没有内容的，现在 nc 等于 5，第 5 个字节从哪儿复制？

现在我们换一个角度来理解这个 COPY 操作，采用下面的算法：

第 1 次复制，复制 1 个字节，为 T，加在原文后，新的原文为 BWT<K\$T；复制的起始位置向后移动一个字节，指向<

第 2 次复制，复制 1 个字节，为<，再加在上一次得到的原文后，新的原文为 BWT<K\$T<；复制的起始位置继续前进一个字节，指向 K

第 3 次复制，复制 1 个字节，为 K，加在上一次的原文后，新的原文为 BWT<K\$T<K；复制起始地址继续前移，指向\$

第 4 次复制，复制 1 个字节，为\$，得到新的原文 BWT<K\$T<K\$；复制起始地址指向 T

第 5 次复制，复制 1 个字节，为 T，得到新的原文 BWT<K\$T<K\$T；复制完毕。

从上面的复制过程可以看出，复制的起始地址其实始终是当前最新原文的末尾回退 delta 个字节的位置。复制 nc 次后，便完成了整个复制过程。

复制完成后，继续读取下一个数，为 15，直接得到字典项为 H，最新的原文为 BWT<K\$T<K\$TH

读取最后一个数为 0，得到字典项 A，最终的原文为 BWT<K\$T<K\$THA；还原过程结束。

五、其它细节问题

了解了 LS11 的文件结构，数据分解原理，字典项提取算法之后，整个 LS11 文件便可以轻易解开了。但是其中还有些需要注意的细节问题，这些问题对有经验的人而言可能是无需啰嗦的，但对与我等新手，还需要仔细观察并留心。

其中开始最令我困惑的是文件信息段中那 12 个字节。虽然我知道每 4 个字节一组，分别表示压缩后的长度，原文长度，和压缩数据起始地址，但需要注意的是这些数据存储的字节序（Endianness，或者叫 Byte Order）问题。

许多人都知道，在 x86 系列的 CPU 上，数据在内存中存储是实际上 **little-endian** 的。这里不准备讨论太多 **Little-endian** 和 **Big-endian** 的基本知识，需要深入了解的朋友可以到网上搜搜。只举个例子：在 x86 上，16 进制数 0xAABBCCDD，它在内存中存放时，实际上是 0xDDCCBBAA 的形式，这样直接写到二进制文件里时，也是 0xDDCCBBAA 的形式。比如下面的一个简单 C 程序：

```
#include <stdio.h>

int main(void)
{
    int n = 0xAABBCCDD;
    FILE *fp;

    if ((fp = fopen("a.dat", "wb")) != NULL) {
        fwrite(&n, sizeof(n), 1, fp);
        fclose(fp);
    }

    return 0;
}
```

在 x86 上运行生成的 a.dat 用 16 进制编辑器打开的话，一般情况下（如果你的十六进制编辑器没有选项选择 **byte order** 的话）会显示 DD CC BB AA。如果再用一个程序来读取刚才生成的 a.dat 的话，数据读入内存后的形式也是 0xDDCCBBAA，而此时 x86 的 CPU 会将其自动解释为 0xAABBCCDD。所以如果你的程序总是在 x86 平台运行的话，一般不会碰到需要手动解释字节顺序这样的问题。

但是很多游戏，比如 KOEI 的大航海，三国等游戏，其发行平台除了 PC 之外，各种游戏机上的平台也有。而许多游戏机的 CPU 采用的 **Big-endian** 方式解释数据。可能为了统一起见吧，KOEI 的 LS11 数据信息段中的数据都是统一以 **Big-endian** 的形式存放的。这样，在 x86 上解析时，就需要手动翻转字节顺序。不然得到的那 12 个字节中的数据将是错误的。

另一个细节问题是数据信息段的 N×12 个字节和最后的数据段并不是完全相邻的，中间总是有 4 个空字节用来分隔它们。所以不要想当然的读完了数据信息段的 N×12 个字节后就开始解析数据了。要以数据信息段中的起始地址为准。

光看这些文字可能并不能使人完全弄清楚 LS11 的存储格式。如果用十六进制编辑器打开一个 LS11 文件，再结合本文中的叙述的话，相信就会比较容易了。

六、完整的 LS11 解压缩程序

基于上面的分析，我制作了一个完整的 **ls11** 解压缩程序，暂时命名为 **ls11dec**。这个程序是基于命令行的，用法很简单：

```
ls11dec FILE1 [FILE2] [FILE3] ...
```

执行后生成 **FILE1.DEC FILE2.DEC FILE3.DEC ...**，**.DEC** 文件不再包含文件头、字典和任何数据信息段，只含有纯粹的解压缩之后的数据。

附带的 **EXE** 文件是支持通配符的编译版本，以便于批量和自动化操作。如果在一个目录下有多个 **LS11** 文件（假设它们都以 **LZW** 结尾），那么就可以使用如下的形式来执行来一次性解压缩全部的文件：

```
ls11dec *.LZW
```

执行后会生成对应的 **DEC** 文件。

需要说明的是，当前版本的程序实现仅识别 **LS11** 一种格式，对于 **LS10** 和 **LS12** 格式，会直接当作 **LS11** 处理。结果正确与否尚未完全测试。

程序的全部源代码只有 3 个文件，**ls11core.h**，**ls11core.c**，和 **ls11main.c**，是基于 **C** 标准库的，无需修改便可以移植到 **Linux** 等平台（虽然好像没啥必要）。可执行程序、源代码均无条件免费使用。对此程序的使用、修改也无需通知原作者。修改后希望（但也不强制）你能保留程序中的作者信息和相关注释。唯一的限制是，**请不要将授权许可改为 GPL 形式**！原作者对直接或间接使用或者修改本程序造成的任何法律纠纷与损失、风险等等均不负任何责任。

七、待解决的问题

早在引言部分就已经说了，文件头有显示 **LS11** 的，也有 **LS10** 和 **LS12** 的。它们之间具体的区别还不得而知，但是解压缩程序对它们都是可以正确工作的。问题是解出来的结果是不是完全正确很难验证。我猜测 **LS11**、**LS10** 和 **LS12** 之间算法上应该是一模一样的，但是其中一些细微的参数，比如 **COPY** 操作的参数可能存在着细微的差别。当然，这仅仅只是猜测而已，目前还没有得到任何验证。

致谢

除了感谢以 van 和 Maxwell 为代表的网友们提供的宝贵信息之外，最后还要感谢 阿尔法孝直验证我的解压缩程序。

主要参考文献

van, 2004-11-26, “英杰传系列LS格式压缩文件解析”, 轩辕春秋文化论坛, 设计与修改区, <http://www.xycq.net/forum/index.php?showtopic=34612>

Maxwell, gameplore等, 2004-12-2, “LS11 格式详解”, 轩辕春秋文化论坛, 设计与修改区, <http://www.xycq.net/forum/index.php?showtopic=35276>