高效 Unicode/ GB 编码转换算法的设计和实现

倪晓军
(南京邮电大学 计算机学院,江苏 南京 210003)

摘 要:为了在存储空间和运算能力严格受限的嵌入式系统中实现 Unicode 和 GB2312 编码的相互转换,设计了一种高效率的编码转换算法。该算法通过提取数据表中公共部分实现压缩存储,采用索引和二分法查找相结合的方式进行快速查找,和传统的转换算法相比约节省 25 %的存储空间,查找效率最高约提高 3 倍。该算法可在无操作系统支持的嵌入式系统中实现汉字编码之间的高效率的转换。

关键词:汉字编码;GB2312;Unicode;索引;二分法查找
中图分类号:TP311 **文献标识码**:A **文章编号**:1673 - 629X(2009)09 - 0021 - 04

A High - Performance Unicode/ GB Transcoding Algorithm

NI Xiao-jun
(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: To address the characteristics of strict limitation on storage space and computing ability in embedded systems, a novel transformation method for the exchange between GB2312 code and Unicode is presented in this paper. The proposed method require less storage space by extracting the common part of converting table, and realize quick searching speed by combining index and binary searching method. Compared with the traditional transformation algorithm, it can save about 25 % storage space while about three times faster, it can be run efficiently without the support of operating system.

Key words: Chinese coding; GB2312; Unicode; index; binary searching

0 引 言

字符必须编码后才能被计算机处理。我国在 1980 年提出了 GB2312 汉字编码方案,共收录了 7445 个字符,包括 6763 个汉字和 682 个其它符号。相对于庞大的汉字字符,GB2312 所收录的汉字太少,因此 1995 年国家推出了汉字扩展规范 GBK1.0,共收录了 21886 个汉字和图形符号。2000 年又制定了 GB18030 编码方案,是取代 GBK1.0 的正式国家标准。该标准共收录了 27484 个汉字,同时还收录了藏文、蒙文、维吾尔文等主要的少数民族文字[1]。

从 GB2312、GBK 到 GB18030,这些编码方案都是向下兼容的,即同一个字符在这些方案中总是有相同的编码,后续的标准支持更多的字符。在这些编码中,英文和中文可以统一地处理。GB2312 编码标准把汉字分成 94 个区,每个区内包含 94 个汉字,用区号和位号组成的 2 字节编码来表示一个汉字,称为汉字的区位码,区码 01～09 主要存放各种字母、符号及制表符等内容,区码 16～87 则存放按汉语拼音字母顺序排列的汉字。为了在使用中区分汉字编码和 ASCII 编码,需要将汉字区位码的高低字节各加上 0xA0,才是该汉字的 GB2132 编码。通过计算(区号 - 1) * 94 + (位号 - 1)可计算出某汉字在整个汉字库中的位置序号。

Unicode 是由国际组织设计的一种字符编码方案,每个编码由两个字节组成,共可表示 65536 个字符,由于编码空间较大,Unicode 可以容纳全球所有的语言文字[2]。正因为如此,在国际化趋势的推动下,越来越多的计算机系统采用 Unicode 作为字符编码。

目前国家标准只要求所有的计算机平台必须支持 GB18030,所以很多嵌入式应用目前在处理汉字时使用的还是传统的 GB2312 编码。但现在要求使用 Unicode 编码的场合越来越多。例如,目前的 GSM 短消息收发中文数据时,中文都是以 Unicode 进行编码,放入用户数据(UD)字段中进行传输的[3,4]。因此,当类似的应用系统需要收发短消息或和其它计算机系统进行数据交换时,必然会涉及到 GB2132 编码和 Unicode 编码之间的转换工作[5]。

收稿日期:2009 - 01 - 05;**修回日期**:2009 - 03 - 06
基金项目:教育部实验室建设专项基金(07J02)
作者简介:倪晓军(1969 -),男,江苏南京人,工程师,从事单片机及嵌入式系统教学与研究工作。

Unicode编码和GB2312码之间并无固定的转换规则,它们之间的转换必须通过查表的方式完成。一般的操作系统都提供了这种编码转换的功能,应用程序设计者只要调用相应的API就可以完成编码的转换[6]。但在计算能力和存储资源都严格受限的嵌入式系统中,很多应用甚至都没有使用操作系统,更没有对Unicode和GB2132编码转换的支持。文中以Unicode和GB2312编码互相转换为目的,针对这种情况设计了一种存储空间需求少、查找效率高的Unicode/GB2312编码转换算法,并在不同的嵌入式应用平台上进行了测试和比较。

1 Unicode/GB2312编码转换算法的设计

1.1 GB2132码转换为Unicode码

由于GB2132编码是按顺序排列的,GB2312码转Unicode的算法很简单,只要预先编制出每个GB2312编码对应的Unicode编码,再将其按照GB2132码汉字的排列顺序排列,存储于一个数组中即可。当需要进行GB2312编码到Unicode编码的转换时,根据汉字的GB2312编码计算出其在汉字库中的索引值,以此为下标访问该数组即可得到对应的Unicode。

1.2 Unicode转换为GB2312码

汉字Unicode编码的排列和GB2132编码的排列完全不同,由于GB2312编码收录的只是常用汉字,故对应的Unicode编码间有不连续的现象,因此Unicode转换为GB2132只能通过查表的方式进行。为此作者设计了一种基于二级索引和二分法查找的Unicode编码到GB2132编码的转换算法,以达到节省存储空间和提高查询速度的目的。

1.2.1 二级索引存储结构的设计

GB2132编码到Unicode编码的转换表可以从Unicode组织的FTP上下载,网络上也有很多其它的下载源。转换表通过文本的方式描述了每个GB2132编码所对应的Unicode编码,例如对于字母符号区:

```
0x2121 0x3000  #  IDEOGRAPHIC SPACE
0x2122 0x3001  #  IDEOGRAPHIC COMMA
0x2123 0x3002  #  IDEOGRAPHIC FULL STOP
0x2124 0x30FB  #  KATAKANA MIDDLE DOT
……
```

对于汉字区:

```
0x3021 0x554A  #  <CJK>
0x3022 0x963F  #  <CJK>
0x3023 0x57C3  #  <CJK>
0x3024 0x6328  #  <CJK>
……
```

前两列是用文本方式的十六进制数表示的。第一列数据加上0x8080是字符的GB2132编码,减去0x2020则是区位码。第二列是对应的Unicode编码,第三列是Unicode的名称。表格中的各列可以是逗号分隔的,也可以是Tab或空格分隔的。

根据这个转换表格可以看出,不管是字母区还是汉字区,GB2312编码的数据是连续的,但是对应的Unicode编码则存在不连续的现象。

为了设计基于二级索引方式的存储结构,需要对上述的编码对照表进行以下几个步骤的处理:

第一步,将上述编码对照表转化成按照Unicode编码排序的表格;

这个工作借助高级语言在计算机上很容易实现,代码简单描述如下:

```
unsigned int  GBCode,Unicode,ArrayUnicode[65536][2];
char s[256];
int i;
FILE  *fp;
fseek(fp,0,SEEK_SET);//  文件指针移到文件头
for(i=0;i<0xffff;i++)//  数组所有项清零
{
ArrayUnicode[i][0]=0;
ArrayUnicode[i][1]=0;
}
while(!feof(fp))//处理整个文件中的每一行
{
fgets(s,100,fp);//读取一行
sscanf(s,"%X\t%X",&GBCode,&Unicode);//格式化数据
GBCode  +=0x8080;//生成GB2312编码
ArrayUnicode[Unicode][0]=Unicode;//以Unicode值为行索引
ArrayUnicode[Unicode][1]=GBCode;//第一列存放Unicode编码,第二列存放GB2312编码
}
```

上述程序段执行完毕后,ArrayUnicode数组中所有ArrayUnicode[x][0]项不为0的就表示该Unicode编码(x)有对应的GB2312编码,否则就表示此Unicode编码(x)没有GB2312编码与之对应。

第二步,去掉所有Unicode编码项为0的数组单元;

扫描ArrayUnicode,将所有ArrayUnicode[x][0]为0的项全部去掉,生成的就是只有与GB2312编码对应的、按Unicode编码升序排列的转换表。此时ArrayUnicode[Unicode][0]中存储的是Unicode编码,ArrayUnicode[Unicode][1]中存储的则是对应的GB2312编码。

第三步,生成压缩存储表及地址索引表。

Unicode编码为16位,分成高低两个字节,存放在ArrayUnicode数组的第一列中且按升序排列,这样U-

nicode 编码高字节相同的所有编码项都排列在一个连续的存储空间中。因此可以将 ArrayUnicode[Unicode][0]中的 Unicode 编码按高字节进行分组,即所有高字节相同的 Unicode 编码分为一组,只存储该 Unicode 的低字节以及与之对应的 GB2312 编码。按照这种方法生成的编码表每项由三个字节组成:

Unicode 编码低字节	GB2312 编码高字节	GB2312 编码低字节

按这种方式组织的转换表每个表项节约了 25 % 的存储空间。由于存储表中只保存 Unicode 编码的低字节,因此在生成存储表的同时,还要生成一个索引表,记录每个高字节的 Unicode 编码转换表项在存储表中的起始位置,以便确认查找范围。索引表的结构为:

Unicode 编码高字节	存储表地址高字节	存储表地址低字节

有了索引表的帮助,可以根据 Unicode 编码的高字节控制查找范围,节省查找时间。索引表和存储表的组织及相互关系如图 1 所示。

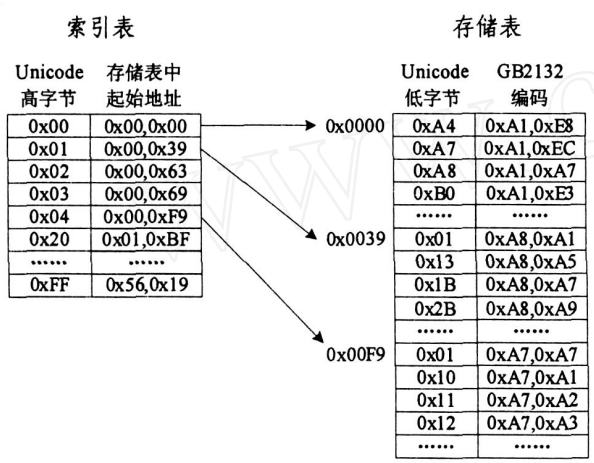

图 1　索引表及压缩存储表结构图

1.2.2　二分法查找的 Unicode 编码到 GB2132 编码的转换算法的实现

二分法查找又称为折半查找,它是针对一个有序数组的高效率的查找方法。假设一个按升序排列的数组 a,共有 n 个元素,当使用二分法查找元素 x 时,该算法的基本思想是首先将这 n 元素分成个数大致相同的两半,取 $a[n/2]$ 与欲查找的 x 作比较,如果 x 等于 $a[n/2]$ 则找到 x,算法终止。如果 $x < a[n/2]$,则只要在数组 a 的左半部继续搜索 x;如果 $x > a[n/2]$,则只要在数组 a 的右半部继续搜索 x,按此规则不断循环即可。在最坏的情况下,这种查找算法的时间复杂度为 $O(\log n)$[7]。

由于索引表和存储表分别是按照 Unicode 的高低字节呈升序排列的,因此对索引表和存储表都可以使用二分法查找,算法的实现比较简单,具体函数如下:

```
unsigned int BSearch(unsigned char * Table,unsigned int Addr1,
unsigned int Addr2,unsigned char Code)
{
    int low,mid,high;
    unsigned char c;
    low = Addr1;
    high = Addr2;//设置查找区间上下限值
    while(low <= high)
    {
      mid = (low + high + 1)/2;//找中间点,上取整
      c = *(Table + mid *3);
      if(Code =c) //如果找到匹配的值
        return (unsigned int)(mid *3);//查找成功,返回该项地址
      if(Code <c) high = mid - 1;//调整上限值,查找左子集
      elselow = mid + 1;//调整下限值,查找右子集
    }
    return 0xffff;//查找不成功
}
```

BSearch 函数通过传递参数的方式给定要查找的表 Table 的起始地址、查找范围的下限地址 Addr1 和上限地址 Addr2 以及待查找的数据 Code,在查找过程中则根据中间点数据和待查数据的大小关系,不断调整上限地址和下限地址实现二分法查找。函数返回的是待查数据 Code 所在的地址。因为所有表格的大小都小于 64k,所以返回 0xffff 表示查找不成功。需要注意的是由于索引表和存储表中的每一项都是 3 字节的,因此取中间点处的数据以及返回中间点地址时均需要把中间项乘以 3。

根据上述分析,可得到使用二分法实现的 Unicode 编码到 GB2312 编码转换的函数:

```
unsigned int UnicodeToGB_BS(unsigned int Unicode)
{
  unsigned int GBCode,StartPos,Index1,Index2;
  unsigned char HiByte,LowByte;

  //分别取 Unicode 的高字节和低字节
    HiByte = (unsigned char)(Unicode/256);
    LowByte = (unsigned char)(Unicode %256);

  //在索引表中查找该高字节 Unicode 的位置(索引表共 98 项)
    StartPos = BSearch(PositionIndex,0,98,HiByte);
    if(StartPos == 0xffff) return 0;//如果没查到则返回 0(无效的 GB2312 编码)

    //取出高字节 Unicode 后面的两个字节组成存储表起始查找地址
    Index1 = PositionIndex[StartPos + 1] *256 + PositionIndex
[StartPos + 2];
    //取出下一个 Unicode 高字节后的两个字节组成存储表查
```

```
找结束地址
    Index2 = PositionIndex[StartPos + 4] * 256 + PositionIndex
[StartPos + 5];

  //以 Index1 和 Index2 作为上下限在存储表中查找 Unicode 的
低字节的存储地址
    StartPos = BSearch(UnicodeToGBTable, Index1/3, Index2/3,
LowByte);
    if(StartPos == 0xffff) return 0;//如果没查到则返回 0(无
效的 GB2312 编码)

    //取出低字节 Unicode 存储地址后面两个地址的内容(字
节)组成 GB2312 编码
    GBCode = UnicodeToGBTable[StartPos + 1] * 256 + Uni-
codeToGBTable[StartPos + 2];
    return GBCode;
}
```

2 算法的效率分析

2.1 存储效率

因为 GB2312 编码是按顺序排列的,GB2312 编码转换为 Unicode 编码的表格不需要存储 GB2312 编码本身,因此其占用的存储空间为编码数 ×2 字节,即 7445 ×2 = 14890 字节。在 Unicode 编码转换为 GB2312 编码时,采用了按 Unicode 编码排序的二级索引方式存储,表格的大小为编码数 ×3 字节 + 索引项数 ×3 字节,即 7445 ×3 + 98 ×3 = 22629 字节。和使用完整的 Unicode 编码 + GB2312 编码对照表的方式相比大约节约了 25 %的存储空间。

该算法设计的 Unicode 编码和 GB2312 编码的双向转换表的总大小为 37519 字节,即使是程序存储空间只有 64k 字节的 MCS - 51 单片机系统,也可以使用该算法。

2.2 查找效率

在进行 GB2312 编码转换为 Unicode 编码的查找时,程序只需要根据 GB2312 编码计算出该编码在存储表中的位置作为下标访问数组即可,这是一个简单的线性数组访问过程,效率极高。在进行 Unicode 编码转换为 GB2312 编码的查找时,需要进行两次二分法查找,索引表为 98 项,使用二分法查找,最差情况下需要 7 次搜索,存储表中 Unicode 编码低字节对应的每个子项几乎都不超过 128 项,使用二分法最差情况也只需要 7 次搜索,而两次搜索的总搜索次数在绝大多数情况下都不需要 14 次。同时,把 Unicode 编码分成高低两个字节分别处理,对 8 位 CPU 来说更容易获得较高的处理效率。经实际程序测试,对于 MCS - 51 单片机,在 24MHz 晶振频率的情况下,二分法查找方式下每个字符的平均查找时间大约为 1 毫秒,比二级索引顺序搜索快 3 倍左右。而对于 32 位 ARM7 的 CPU,两种查找方法相比,二分法查找和二级索引顺序查找法相比速度只提高了约 20 %,这主要是因为算法中涉及到不少 16 位数的乘除法运算,8 位 CPU 必须调用库函数才能实现,CPU 负担较重。二级索引及二分法查找减少了进行乘除法运算的次数,因此在 8 位 CPU 的应用场合可以获得较好的结果,而 32 位 ARM7 的 CPU 内部具有乘法器,减少乘除运算带来的效果就不太明显了。

3 结束语

采用文中所设计的 Unicode 编码和 GB2312 编码的双向转换算法可以有效地减少转换表的存储容量,提高查找效率,特别对于编码容量固定但无统一转换规则的编码之间的转换有较普遍的实用意义,使存储空间和计算能力都有限的嵌入式系统可以留出宝贵的存储空间和 CPU 处理时间供程序使用。该算法使用了数据结构课程中的相关理论内容,结合实际的 C 语言编程实现,非常适合在嵌入式系统教学中作为设计性实验,培养学生理论联系实际及实践能力。目前该算法已在作者设计的单片机实验教学系统以及多个 GSM 短消息有关的应用中使用,负责短消息数据包中用 Unicode 编码的汉字和 GB2312 编码的汉字之间的相互转换。算法占用的存储空间较小,查找效率高,效果很好。

参考文献:

[1] 邱发林. Unicode 及中文到 Unicode 转换[J]. 科技信息,2006(3):20 - 21.

[2] 张晓培,李 祥. 从 Unicode 到 GBK 的内码转换[J]. 微计算机应用,2006(6):757 - 759.

[3] ETSI GTS GSM 03.40 Version 5.3.0 Digital Cellular TelecommunicaionsSystem (Phase 2 +); Technical Realization of the Short Message Service (SMS) Point - to - Point (PP)[EB/OL]. 1996. http://www.etsi.org.

[4] ETSI GTS GSM 07.05 Version 5.5.0[EB/OL]. 1998 - 01. http://www.etsi.org,1998.

[5] 刘 晋. 手机短消息服务在 OA 系统中的实现[J]. 计算机工程,2004(S1):202 - 203.

[6] 徐妙君,张晓霞. 短消息控件的设计与实现[J]. 计算机技术与发展,2007,17(8):64 - 66.

[7] 陈慧南. 数据结构 ——使用 C + + 语言描述[M]. 南京:东南大学出版社,2001.