多路归并外部排序的简单实现

费政聪

1 摘要

本文提出了一种基于败方树算法的多路归并外排序程序设计和实现方案。经过数据测试，可以在 200 秒内对 2.5 亿浮点数进行排序和输出。首先，本文给出了程序的整体设计架构。其次，提出了一些生成大规模随机数的算法用于生成测试集，比较了二进制文件分块读取、文件流读取等不同读写方式的效率和数据有效性等常见问题。然后，讨论不同排序算法在本问题上的优点和局限性，并给出了多路归并外排序算法的实现细节。最后，本文讨论了程序的结果和未来改进的方向。

2 问题简介

外部排序指的是大文件的排序，即待排序的记录存储在外存储器上，待排序的文件无法一次装入内存，需要在内存和外部存储器之间进行多次数据交换，以达到排序整个文件的目的。外部排序最常用的算法是多路归并排序，即将原文件分解成多个能够一次性装入内存的部分，分别把每一部分调入内存完成排序。然后，对已经排序的子文件进行多路归并排序。

本文需要实现一个可以对“海量”浮点数进行外部排序的程序 Sort.exe——读取文件路径和其他配置信息，将“输入文本文件”里面的浮点数从小到大排序，然后输出到另外一个“输出文本文件”中。完成后，用户双击 Sort.exe，程序即自动加载同目录下的 Sort.param，而后开始运行。Sort.param 里面放着“输入”和“输出”的文本文件（如.txt）的路径及其他信息。

3 程序设计与实现

3.1 程序框架

本文设计的程序主要框架如图 3-1，主要分为测试集生成和外部归并排序两个模块。

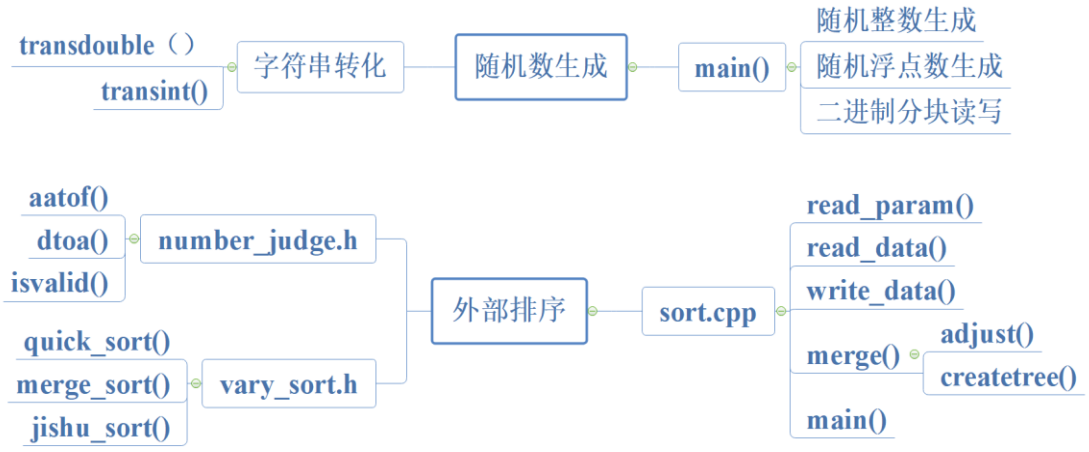

图 3-1 程序设计框架图

其中，每个模块中实现的 API 名称及其功能见表 3-1。

表 3-1 程序 API 功能实现设计表

文件	API	主要功能
Sort.cpp	Read_param()	读取输入输出文件地址
	Read_data()	分块读取数据
	Write_data()	排序结果写入临时文件
	Merge()	多路归并到输出文件
Number_judge.h	Aatof()	字符串转为浮点数
	Dtoa()	浮点数转为字符串
	Isvalid()	判断是否为有效数字
Generate.cpp	Trandouble()	浮点数转为字符串
	Transint()	整型转为字符串

3.2 随机数生成

程序中经常会需要用到随机数，所谓随机数，就是随机生成一个数字供程序

测试使用。大部分语言都有随机数生成器的函数，比如 C/C++就有个最简单随机函数：rand，它可以生成一个“伪随机”的均匀分布的整数，范围在 0 到系统相关的一个最大值之间。这里的均匀是指随机数的分布是均匀的，c++11 提供了 API，可以生成非均匀分布的随机数，比如正态分布的。

rand 函数只能生成均匀分布的整数，但程序有时候需要浮点数、非均匀分布、其他范围的分布等等，这就需要开发者去进行转换，而这会导入非随机因素，且不方便。因此，本文使用了头文件“random”的随机数库中提供的工具。具体参数设置要求可以参见文献[1]。

以本文中 generate.cpp 为例，设置过程如下：

- 首先，设置随机数引擎和种子。

  default_random_engine e(time(0))

- 其次，确定随机数范围和随机数的类型。这里分别随机产生$-308\sim308$的整型作为指数，$-10.0\sim10.0$的浮点数作为小数部分。

  uniform_int_distribution<int> u4(-308,308);

  uniform_real_distribution<double> u(-10.0,10.0);

- 最后，通过 API 转为字符串形式，拼出双精度浮点数。

测试集中需要有非法项目来测试代码的鲁棒性。在生成测试集程序中，需要用户输入生成数据数目 n 和非法项目的个数 m，且 n 必须大于等于 m。然后，要随机确定插入的位置。需要注意，如果直接产生范围是$1\sim n$的 m 个随机数，是有问题的。因为可能产生两个相同的位置，程序运行完毕后，实际非法项目少于用户给定 m。因此，需要生成 m 个不重复的随机数作为插入的位置。

文献[2]给出了随机生成不重复大数据（千万级）的算法：

```
for (n = 1; n <= size; n++)
    num[n] = n;
srand((unsigned)time(NULL));
int i, j;
for (n = 0; n < size; n++)
{
    i = (rand() * RAND_MAX + rand()) % 10000000;
    j = (rand() * RAND_MAX + rand()) % 10000000;
    swap(num[i], num[j]);
}
```

核心思想是先顺序生成序列，再随机交换两个位置的数字。

由于本文测试集不会有很复杂的非法输入条目，数目也不会很多，所以，这里使用的方案是每次产生一个随机数，遍历一遍数组，如果没有出现，则放入数组，否则，重新生成再判定。

另外，本模块需要注意的地方：

1. 不要忘记包含头文件 random
2. 如果生成的随机数超出 double 范围，例如8.3×10^{308}，则需要重新生成。
3. 浮点数生成字符串，截止状态判断不能等于 0.00，应设置小于 eps 时停止，否则会不正确。
4. 字符串结尾要加上 0x0a，而不是回车换行。
5. 输入非法项目的个数必须要保证小于等于数据的数目，否则需要重新输入。

3.3 有效数字的判定和文件分块读写

程序需要输出数据文件中非法项目的个数，因此在读取时，需要对数字的有效性进行判断。在 leetcode 上有一道 hard 难度的题，就是对输入任意字符串判断是否为有效数字。文献[3]对该题进行了分析，并根据所有出现的情况进行了分类讨论。文献[4]利用有限自动机 Finite Automata Machine 的算法，实现的简洁优雅。另外值得一提，文献[5]利用正则表达式，只用了一行代码：

```
String regex = "[-+]?(\\d+\\.?|\\.\\d+)\\d*(e[-+]?\\d+)?";
```

就精彩的实现了判定。

本文，因为数据格式已经符合浮点数的标准，仅需要考虑是否有异常字符出现，小数点和字母 E（e）只出现一次，且小数点必须出现在字母 E（e）前面，指数必须为整数等情况即可。函数实现细节如下：

```
bool isvalid(char *s,int l){
    bool dotexist=false; //e,小数点只能出现1次。
    bool eexist=false;
    int i=0;
    char *p=s;
    while(i<l){
        if(isdigital(p)||*p=='+'||*p=='-') {
            ++p; ++i;
        }
        else if((*p=='E'||*p=='e')&&!eexist){
            ++p; eexist=true; ++i;
        }else if(*p=='.'&&!dotexist&&!eexist){ //指数部分不能是小数。
            ++p; dotexist=true; ++i;
        }else{
            return false;
        }
    }
    if(i==l) return true;
    else return false;
}
```

接下来，考虑文件的读写，根据每次读取的数量可分为逐个读取，逐行读取和指定字节块读取，根据打开文件的方式可以分为二进制打开和文本文件打开。本文采用 fread()和 fwrite()函数分块读取和二进制文件操作方式。关于这两个函数的参数设置，可以参见文献[6]。

下面介绍 fread()函数使用需要注意的地方。

批量读取，一般首先想到的，是建立 double 数组，每次读入 100 万个 double 进行排序，下面是一个小的测试例子。图 3-2 是 fread 读取的结果，图 3-3 是 txt 中的数据，图 3-4 是 txt 文件用 UltraEdit 查看二进制的数据。

```
double a[100];
int main(){
    FILE* fp;
    fp=fopen("M1.txt","rb");
    fread(a,sizeof(double)+1,4,fp);
    for(int i=0;i<4;i++) cout<<a[i]<<endl;
    fclose(fp);
    return 0;
}
```

```
3.42631e-086
6.36835e-067
2.56056e+025
1.51979e-047
```

图 3-2 fread 读取的结果

M1.txt

```
1 2.123
2 1.7564E+123
3 -12.5E-12
4 1666
```

图 3-3 txt 文件存储的数据

```
00000000h: 32 2E 31 32 33 0A 31 2E 37 35 36 34 45 2B 31 32 ; 2.123.1.7564E+12
00000010h: 33 0A 2D 31 32 2E 35 45 2D 31 32 0A 31 36 36 36 ; 3.-12.5E-12.1666
```

图 3-5 用 UltraEdit 查看二进制的显示数据

实际测试发现数组中存放的数和 txt 文件中的不同。因为测试的文件不是直接用 double 写进去的。如果是用 fwrite(&s[0],sizeof(double),sizeof(s),fp)直接写入，那么读取数据是正确的。否则，需要先用 char 形式读入缓冲区，把每个数字根据 0x0a 分开，然后转换成 double。另外，实验显示，对于读入 char 数组，设置 fread(a,1,8,fp)和 fread(a,8,1,fp)结果是一样的。

其次，fread()函数，如果读入数据小于设定的值，则会返回实际读取个数。可以用来判断是否读到数据文件的末尾。

最后，考虑读取文件分块。每次读取大小为 mem_size 字节数的数据块。根据 0x0a 划分得到每个浮点数。特别需要注意边界结尾处，需要分为两种情况，第一种，最后一个字符为 0x0a，说明数据块正好分在浮点数后面。第二种，最后一个字符不是 0x0a，则需要要最后的一串字符放入临时数组，并标志 resident 为 1。每次读入数据块前，需要先判断是否上一次读取有剩余。另外，如果需要在每次读取数据块后面加上特殊字符作为结束标志位，方便情况判定的代码书写。特殊字符的选择诸如‘#’，不能出现在非法项目中，否则会导致提前结束写入，最终数据量小于测试集。

对于 mem_size 字节大小的选择，本文默认 100 万字节。那么，每次读取大约 mem_size 字节，假设每个数字平均下来，形式如x. xxxE + xx，则需要 10 个字节，每次数组大约存放mem_size/10=10 万个浮点数。一般 1000 万级别数量，排序时间大约在数秒，可以接受。因此，这里累计到 1000 万，即分块读取 100 次后，对数组进行排序，并写入临时文件。

本模块需要注意的地方：

1. 数字取舍的技巧：

   向下取整 (int)a

   向上取整 (a>(int)a?(int)a+1:(int)a)

   四舍五入(int)(a+0.5)

   因此，在这里，可以通过 tmp+=0.0000000005 这样去尾后能保持 9 位小数的四舍五入。

2. 读操作 fread()和写操作 fwrite()后必须关闭流 fclose()。
3. fopen 打开二进制文件要设置“rb”
4. 如果已经读到文件末尾，不要忘记将剩余的部分写入临时文件，否则会造成读书数据偏少。
5. 转化关系：1bytle=8bit=2 个十六进制位

   double 8 个字节，char 1 个字节。

3.4 排序算法

对于大数据（千万级），最快的是位图排序算法。

参考《编程珠玑》[7]一书上的位图方案，以10^7个数据量的磁盘文件排序为例，可以这么考虑，由于每个 7 位十进制整数表示一个小于 1000 万的整数。使用一个具有 1000 万个位的字符串来表示这个文件，其中，当且仅当整数 i 在文件中存在时，第 i 位为 1。采取这个位图排序的特殊性：1、输入数据限制在相对较小的范围内，2、数据没有重复，3、其中的每条记录都是单一的整数，没有任何其它与之关联的数据。

所以，用位图的方案一般分为以下三步进行解决：

第一步，将所有的位都置为 0，从而将集合初始化为空。

第二步，通过读入文件中的每个整数来建立集合，将每个对应的位都置为 1。

第三步，检验每一位，如果该位为 1，就输出对应的整数。

由于测试集是浮点数，使用 bit-map 并不方便，而且数据可能出现重复。

本文对临时文件的排序，在 vary_sort 头文件中，提供了多种不同的解决方案，推荐使用 qsort 函数，是采用的三数中值的快速排序（个数小于 3 用插入排序）实现的。

最后，给出根据数据规模选择排序算法的建议：

1. 数据量较小，考虑使用随机快排，并结合考虑是否需要排序的稳定性。
2. 10W 量级考虑使用 STL 的 sort 函数。sort 函数默认是升序排序，要降序排序可以传 cmp 函数过去或者 sort 完了 reverse
3. 1000W 量级考虑使用计数排序或者基数排序，这两种排序对被排序数字的大

小有一定限制。

4. 如果是 1000W 量级以上的实数排序，或者数字范围可能很大的话，那么还是回到之前说的 STL 库函数 sort。
5. 具体情况还需要考虑数据范围、代码存储空间等因素。

3.5 多路归并和败方树

多路归并的主要步骤：首先分多次从数据文件中读取 M（百万级）个整数，每次将 M 个整数在内存中使用特定的排序算法之后存入临时文件，然后使用多路归并将各个临时文件中的数据再次整体排好序后写入输出文件。显然，该排序算法需要对每个整数做 2 次磁盘读和 2 次磁盘写。以下是 merge 模块的流程图：

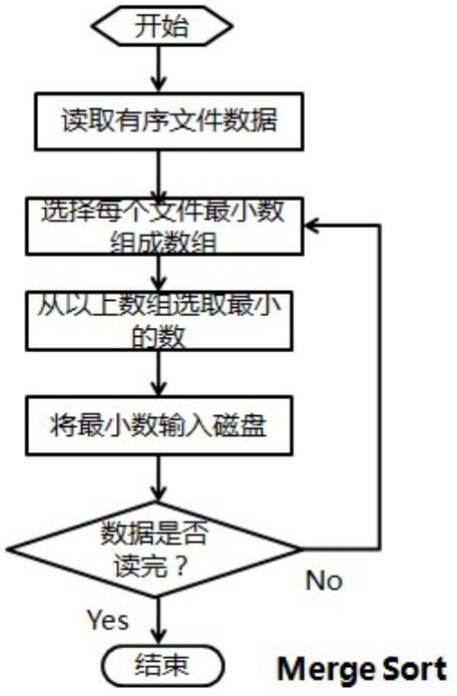

图 3-6 归并算法流程图

本文归并实现步骤：

- ✓ 将每个文件最开始的数读入(由于有序，所以为该文件最小数)，存放在一个大小为 fcount 的 arr 数组中；
- ✓ 选择 arr 数组中最小的数，及其对应的文件索引 index；
- ✓ 将 arr 数组中最小的数写入数组 write，然后更新数组 arr(根据 index 读取该文件下一个数代替)；

✓ 判断是否所有数据都读取完毕，否则返回 2。

文献[2]通过循环顺序扫描数组，找到最小数。时间复杂度O（n）。文献[8]详细比较了堆、胜者树、败方树算法在归并算法上的相同点和不同点。简单来说，共同点在于空间和时间复杂度都是一样的。调整一次的时间复杂度都是O(logN)。不同点则是调整堆的时候，每次都要选出父节点的两个孩子节点的最小值，然后再用孩子节点的最小值和父节点进行比较，所以每调整一层需要比较两次。而胜者树每次比较只用跟自己的兄弟节点进行比较，所以用胜者树可以比堆少一半的比较次数。在使用败者树的时候，每个新元素上升时，只需要获得父节点并比较即可。所以总的来说，减少了访存的时间。

因此，本文选择败方树算法来实现多路合并。文献[9]给出了败者树 adjust、createtree 等过程的源码。

本模块需要注意的地方：

1. fread(arr[i],sizeof(double),mem_size+50,tmp[i]); 利用其返回值记录每个临时文件中浮点数的个数。
2. float 头文件中包含了 double 的最大值 DBL_MAX 和最小值 DBL_MIN。
3. new 分配的空间，最后需要 delete。

4 结果讨论

本文测试时，使用 8 代 i5cpu，运行内存 16G 的电脑。

首先，产生随机数据测试，非法项目设置为 200 个。根据测试要求，分别产生数据量为 100 万、5000 万和 2.5 亿的数据，并记录花费的时间，见表 4-1。

表 4-1 测试集数据量与花费时间

生成数据规模	100 万	5000 万	2.5 亿
花费时间	0.307 秒	14.344 秒	71.504 秒

然后，配置参数文件 Sort.param，使用 Sort.exe 对测试集进行排序，记录花费时间，见表 4-2。

表 4-2 排序数据量与花费时间

生成数据规模	100 万	5000 万	2.5 亿
花费时间	0.769 秒	31.561 秒	154.197 秒

最后，我们对排序结果进行审阅，检测到非法项目 200 个，和预期设置一致，排序格式符合要求，以 0x0a 结尾。

本文简单实现了对亿级大数据的外部排序，并使用败方树等算法优化了运行时间，达到了百秒级别时间内完成排序。接下来，可以考虑完善的方向：

- 充分利用内存，提高内存使用率。
- 进行了多线程操作。
- 设计 UI 界面，提高用户使用体验。

参考文献

[1] https://www.jianshu.com/p/cc6e48b2d670

[2] https://blog.csdn.net/v_JULY_v/article/details/6451990

[3] https://www.cnblogs.com/ariel-dreamland/p/9151000.html

[4] https://blog.csdn.net/kenden23/article/details/18696083

[5] https://blog.csdn.net/fightforyourdream/article/details/12900751

[6] https://www.cnblogs.com/xudong-bupt/p/3478297.html

[7] Jon Bentley. 编程珠玑[M]. 北京：人民邮电出版社，2008：10-19

[8] https://blog.csdn.net/haolexiao/article/details/53488314

[9] https://www.cnblogs.com/iyjhabc/p/3141665.html