**随机森林实验报告**

14331189 刘子毅 计应 2017年6月22日16:29:23

**问题分析**

本次实验一共是1866819个训练数据，494851个测试数据，样本一共有两类，训练和测试数据样本都含有缺失值，对于数据中没有的特征，默认该特征的值为0。

**开发环境**

处理器：双核，Intel(R) Core(TM) i5-4258U

操作系统：win10专业版

开发工具：Visual Studio 2017

开发语言：c/c++

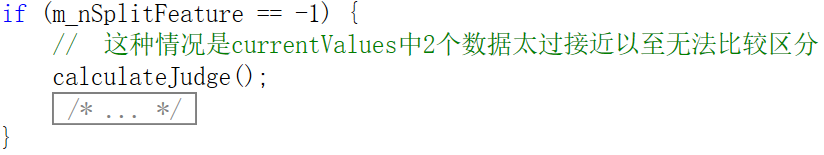
**算法分析与设计**

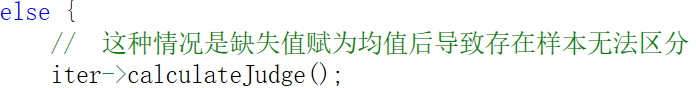
**CART分类树的构建**

这次使用CART树来分类，我用Gini系数最小化准则来进行特征选择，对每个结点进行以下操作，生成二叉树：

a.设结点的训练数据集为D，从D中算出所有特征值不为0的特征（对特征值为0的特征处理见后文），假如总共有p个特征，随机选出（向下取整）个特征，作为SelectedFeatures集合，对SelectedFeatures的每一个特征计算所有分裂点，通过所有特征的所有分裂点对数据进行划分，计算划分后的数据集D1,D2的Gini系数，再求出加权和sum，选出使得sum最小的特征和分裂点作为该节点的splitFeature和splitPoint，依照这两个值，将现结点数据集D划分为D1,D2来作为子节点。

b.对子节点重复a的操作，除非父节点Gini系数为0或者所有特征的值要么太过接近无法再划分为两个子节点要么特征值缺失导致其赋为均值后无法再划分为两个子节点：





我们这里让CART树完全生长，不剪枝，因为随机选取样本和随机选取特征完成了防止过拟合的工作。

CART树的构建原理参考博客：

<http://blog.csdn.net/baimafujinji/article/details/53269040>

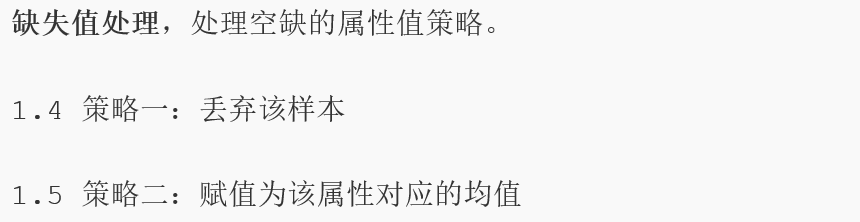
**随机森林的构建**

从1866819个训练数据中，随机选出1000个训练集中的数据作为根节点的数据集D去构建1棵CART树，再从1866819个训练数据中，随机选出1000个训练集中的数据去构建第2棵CART树，以此类推构建1866棵CART树。

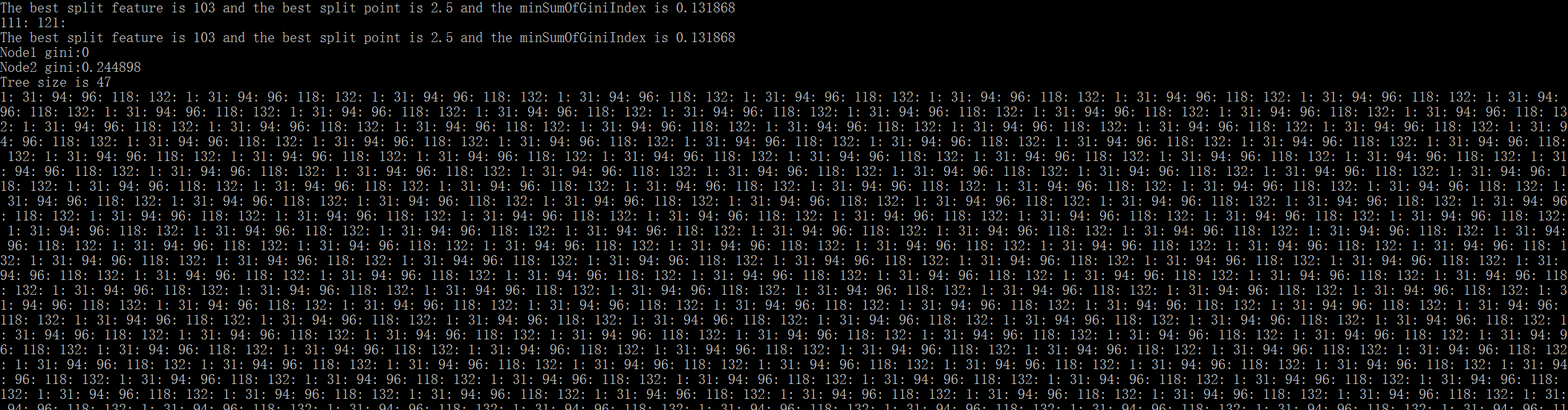
**缺失值处理**

缺失值的处理参考该博客，我采用了策略二处理缺失值：

<http://blog.csdn.net/yujianmin1990/article/details/47406037>



但是采用策略二会导致的问题是如果两个样本s1、s2类标不同，选中的特征是f1和f2，期中s1的f1缺省，s2的f2缺省，且没有别的样本，会导致无法分类的情况，如下图：

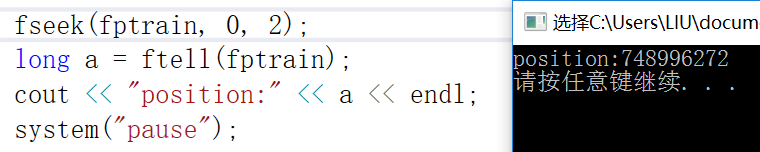


另外一种情况就是两个特征值太过接近，浮点数判断<=时无法区分。

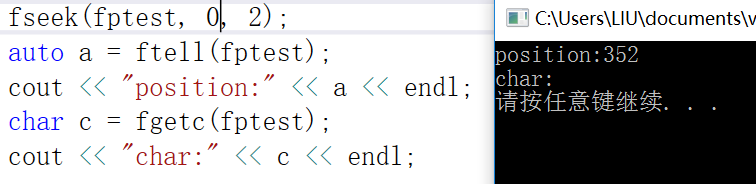
**数据处理**

由下图可知测试集总共大小为748996272个字节，约等于文件实际大小731442KB：





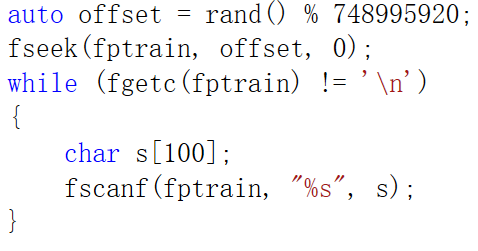
且最后一行大小为352个字节：



计算这个是为了能够随机选取样本。

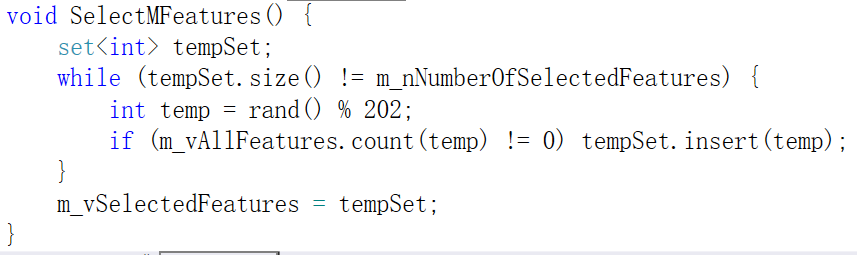
**随机选取样本**

已知数据共748996272个字节，减掉最后一行352个字节，获得748995920，从中随机选一个数，读取该数对应的下一行，作为一个样本，train\_data.txt最前面开手动添加了一个‘1\n’，确保第offset等于0的时候能读到第一个数据，从748995920中选一个数是为了保证最后一行的数据也能被读到：



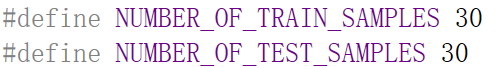
**随机选取特征**

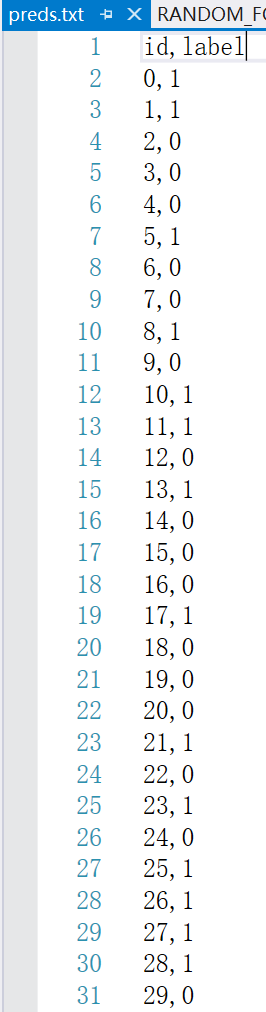
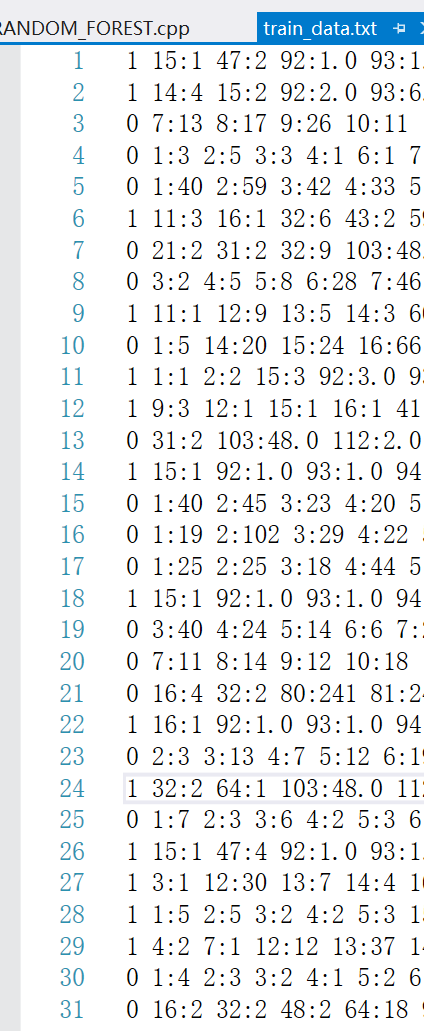
从所有非0特征中选取个特征：



**集成测试**

可以通过设置这个作为一个小的集成测试，在小样本上检测决策树是否正确，比如说取训练集30个，来训练和测试，结果发现分类争取率达到100%：

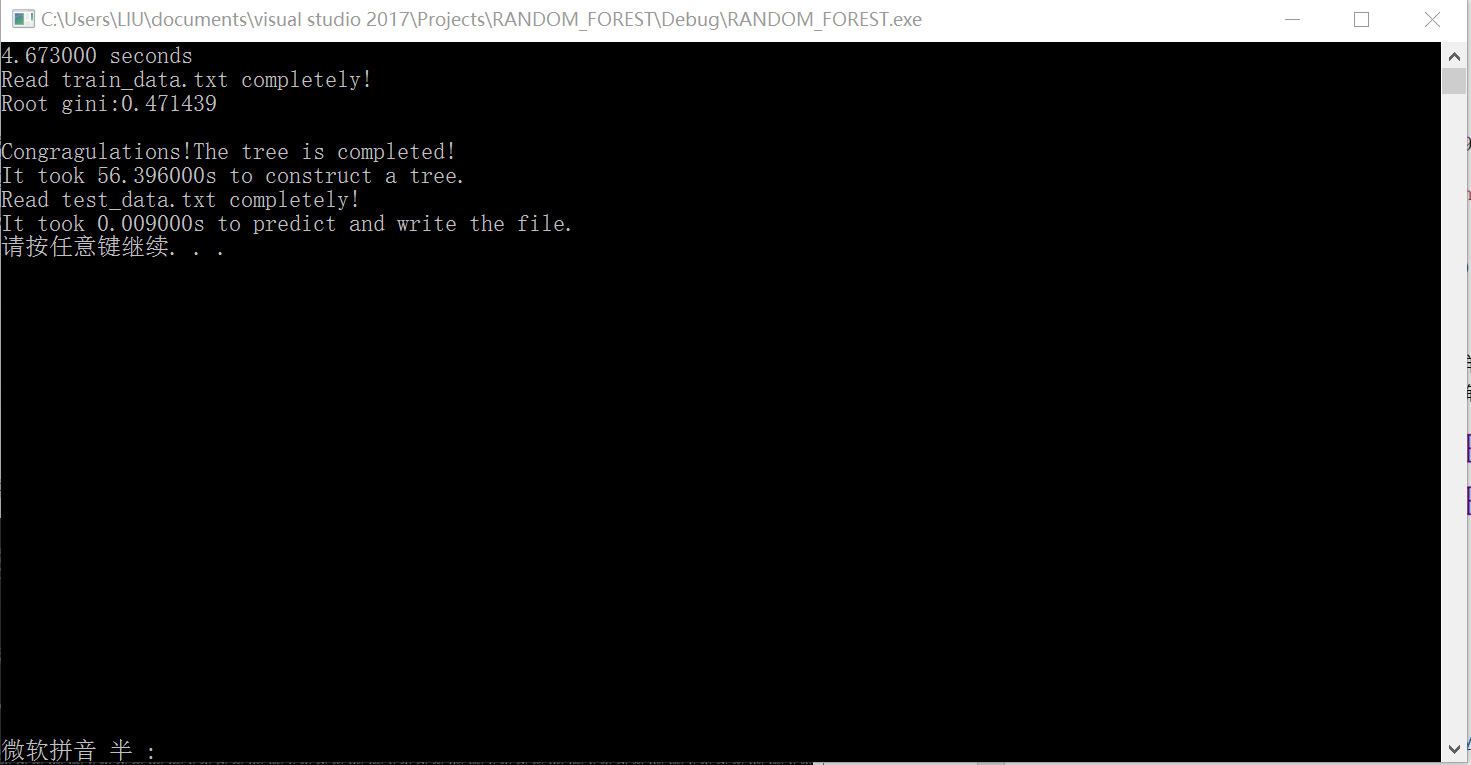




开发至此，基本确认数据的读入、处理、一棵决策树的构建、预测和输出都正确。

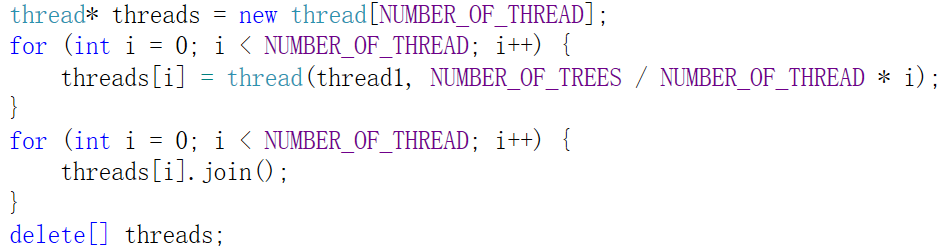
**效率优化**

接下来是效率的问题，最一开始程序运行非常慢，分裂第一个节点要6分钟的时间。经过排查之后，确定因为是STL的滥用。比如说CSAMPLE类中，除了使用set来保存特征，还用了map来存储特征值，这显然是一种浪费，还有就是CNode的Split()函数使用了非常多STL作为局部变量和函数调用，这就大大降低了程序的速度。经过一系列的优化，将NUMBER\_OF\_TRAIN\_SAMPLES设为2000，建树的效率和预测的效率达到了可以接受的程度：



**算法的并行化**

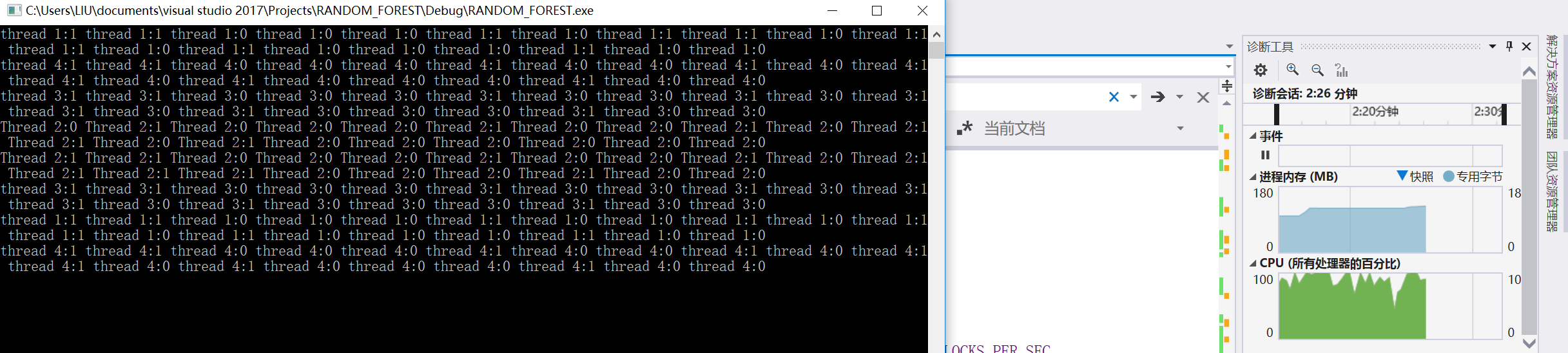
主要使用多线程来实现并行化建树，c++11提供了不少新特性，包括thread类，mutex类，使用这两个类基本就能实现多线程建树以及访问共享对象的加锁操作。并行化建树的实现，其中被调用函数thread1包含了随机选取样本、建树和预测的过程。



只使用两个CPU进行并行化训练，每棵树使用500个样本，建立16棵树，程序会卡在这里，发生死锁：



对所要读的文件加了锁后，建立四个线程，程序可以正常执行，CPU（所有处理器百分比接近100%）：

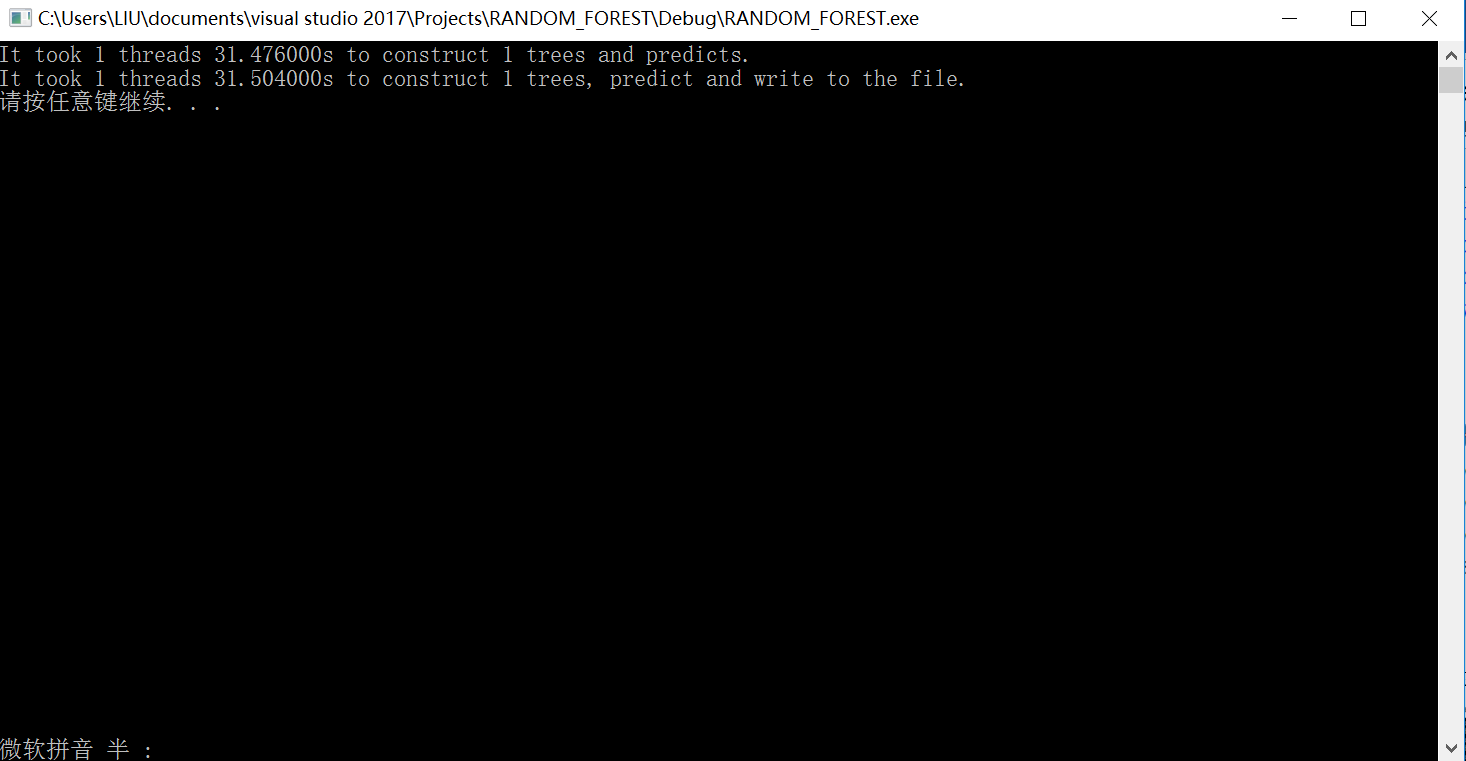


并行教程参考博客：<http://www.cnblogs.com/haippy/p/3236136.html>

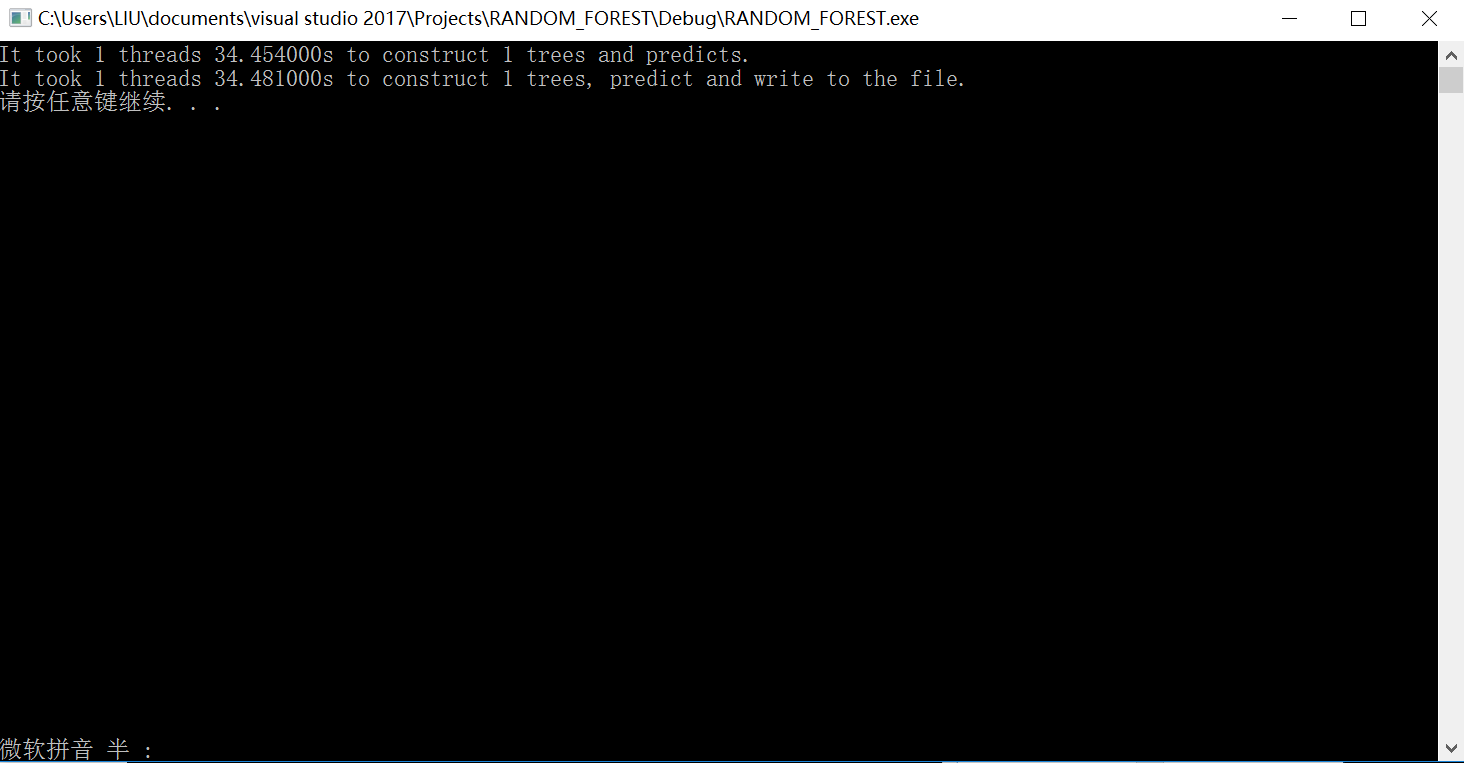
**结果分析**

因为电脑只有双核，所以我建的线程数都是2的倍数，下面展示优化前和优化后的结果：

1000个样本，10000个测试样本，1个线程，1棵树，花了31秒，以这个为基线：

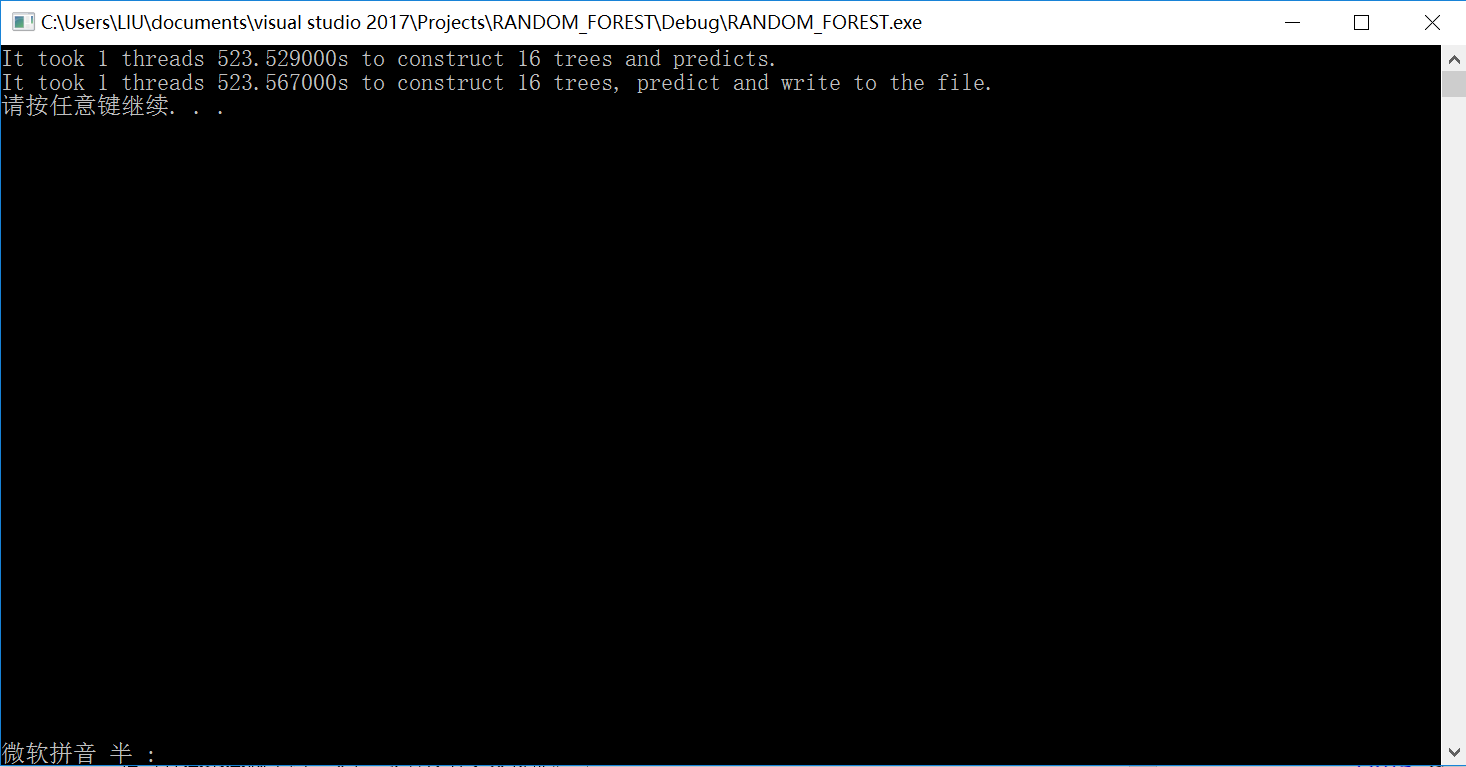


优化数据结构，去掉加锁操作后，10000个测试样本，1个线程，1棵树，花了34秒：

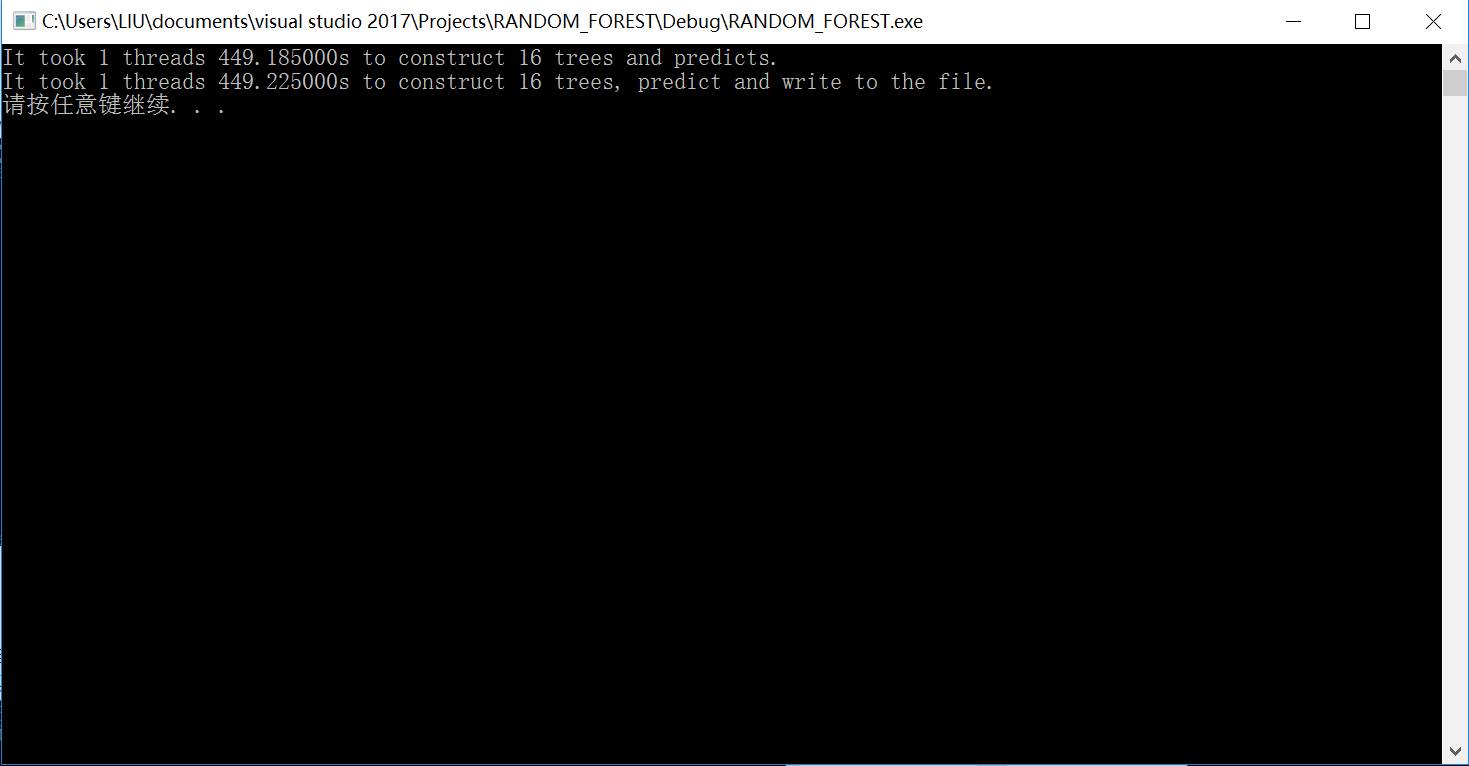


因为优化是对多线程和多棵树进行优化，单线程和单棵树的时间变化主要还是体现在建树所选的样本和特征的随机性上。

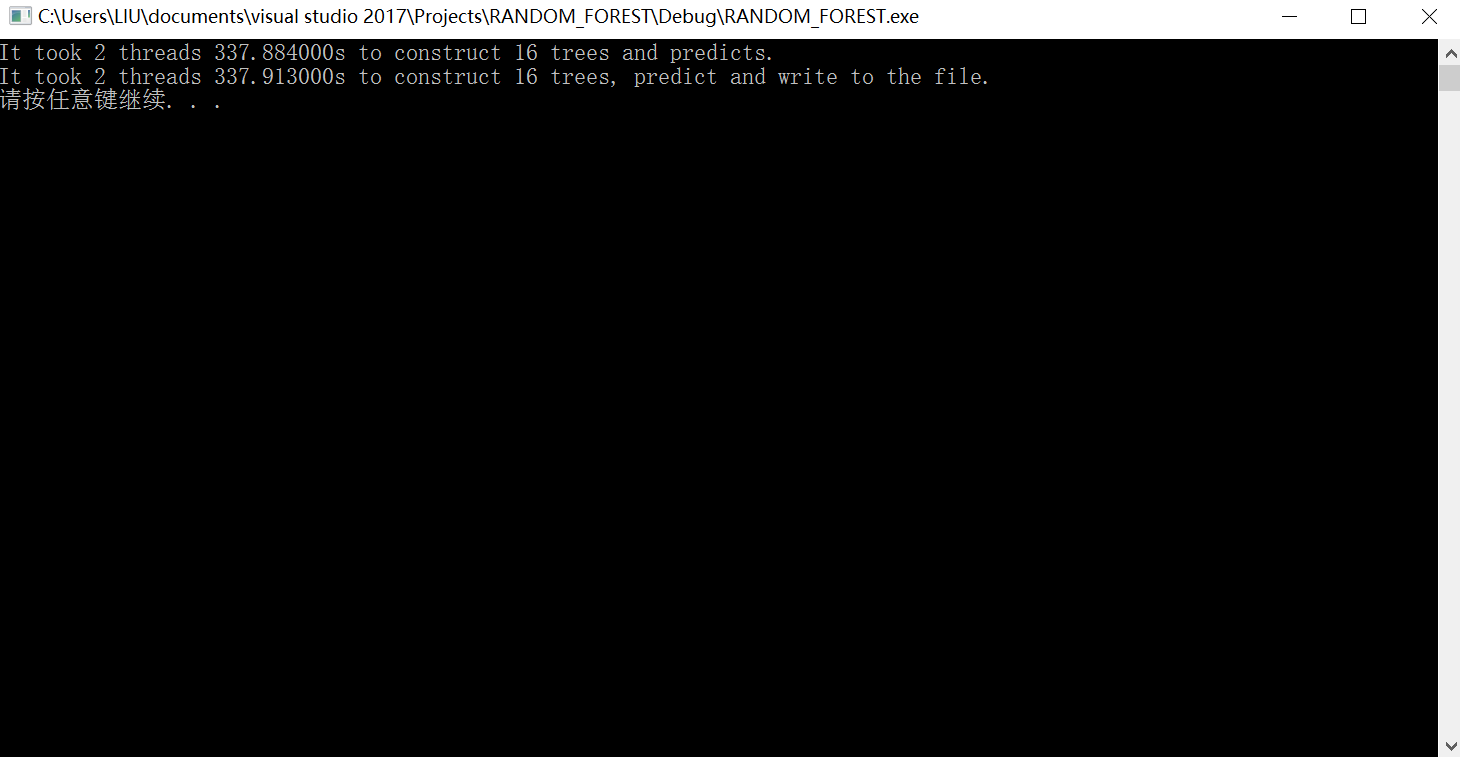
1000个样本，10000个测试样本，1个线程，16棵树，花了523秒，建1棵树的平均时间为32秒：



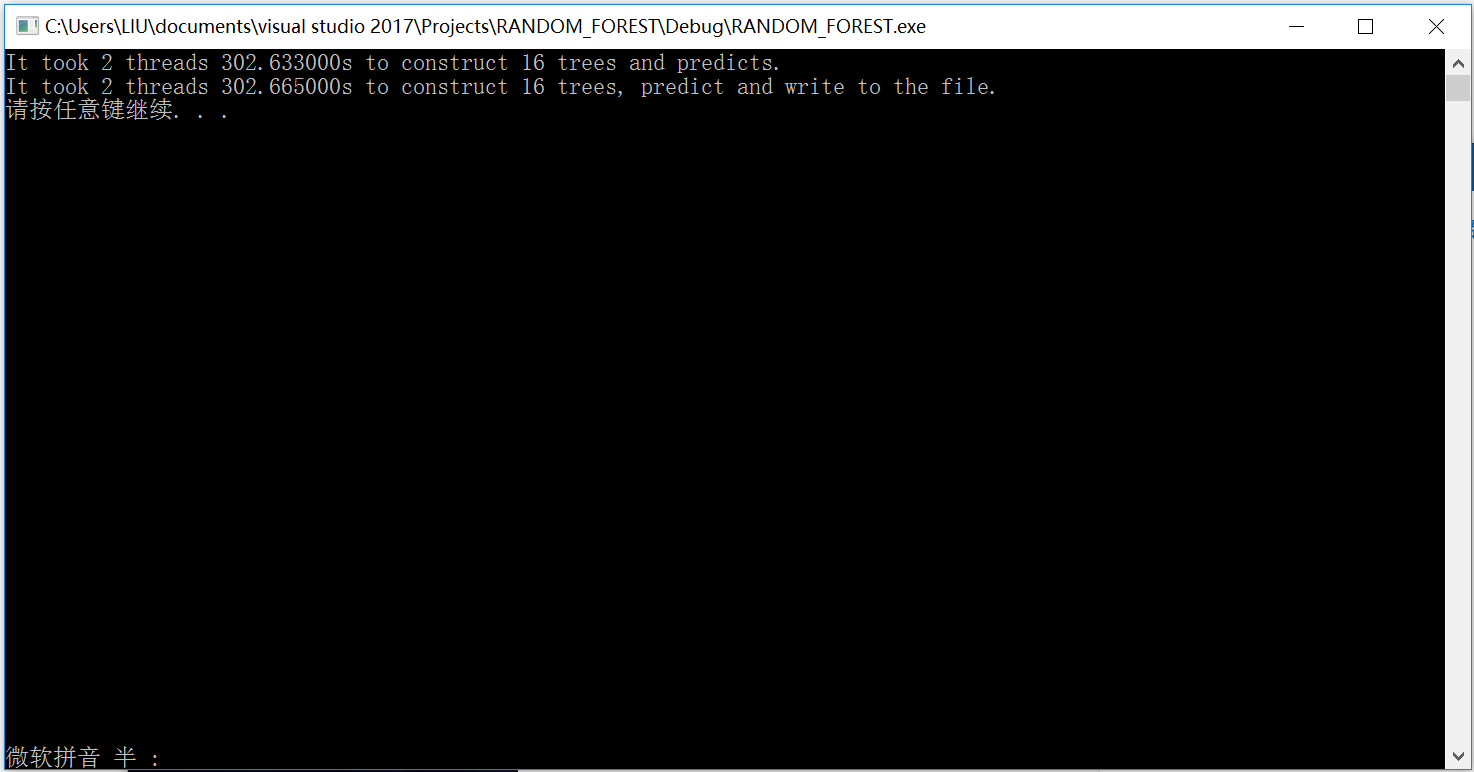
优化数据结构，去掉加锁操作后，1000个样本，10000个测试样本，1个线程，16棵树，花了449秒，建1棵树的平均时间为28秒：



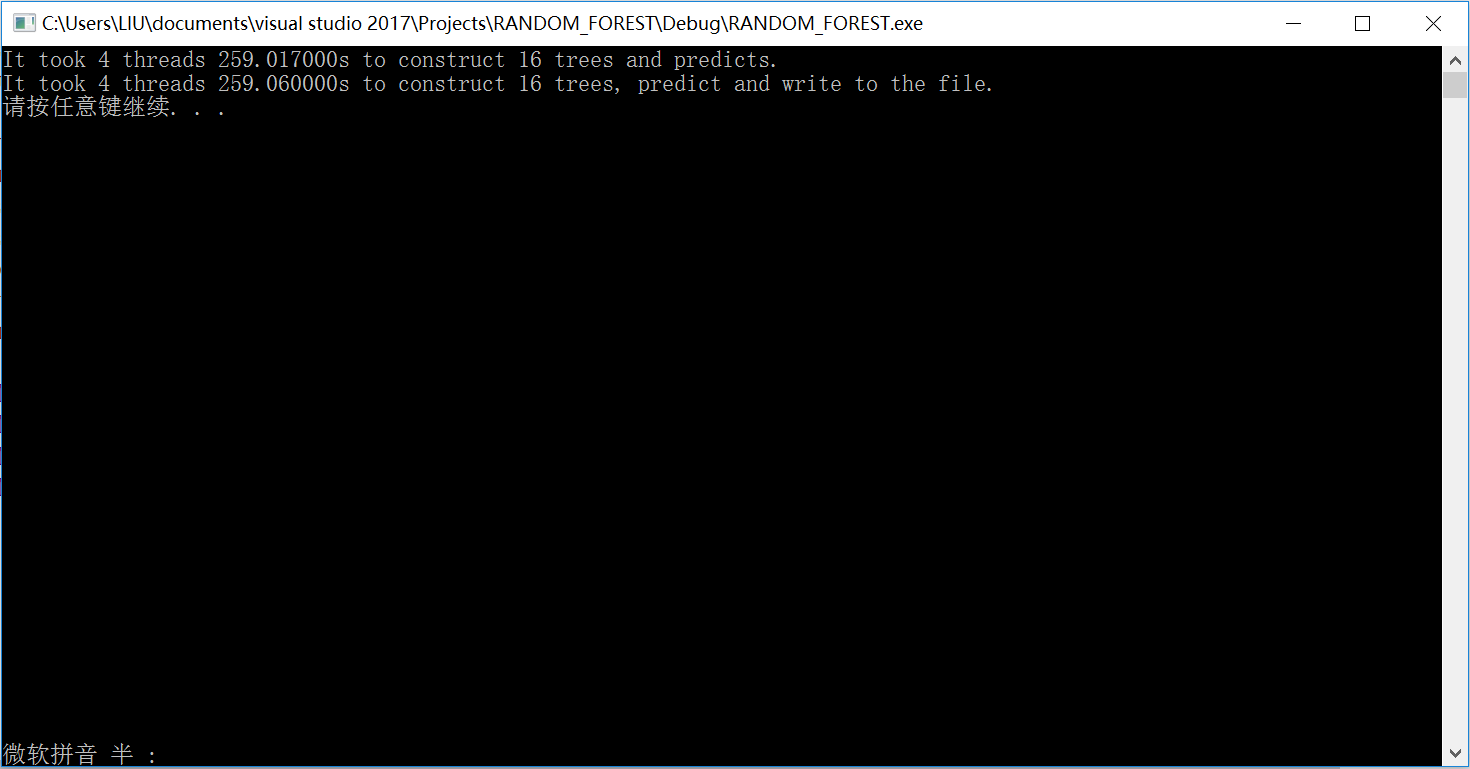
1000个样本，10000个测试样本，2个线程，16棵树，花了337秒，建1棵树的平均时间为21秒：



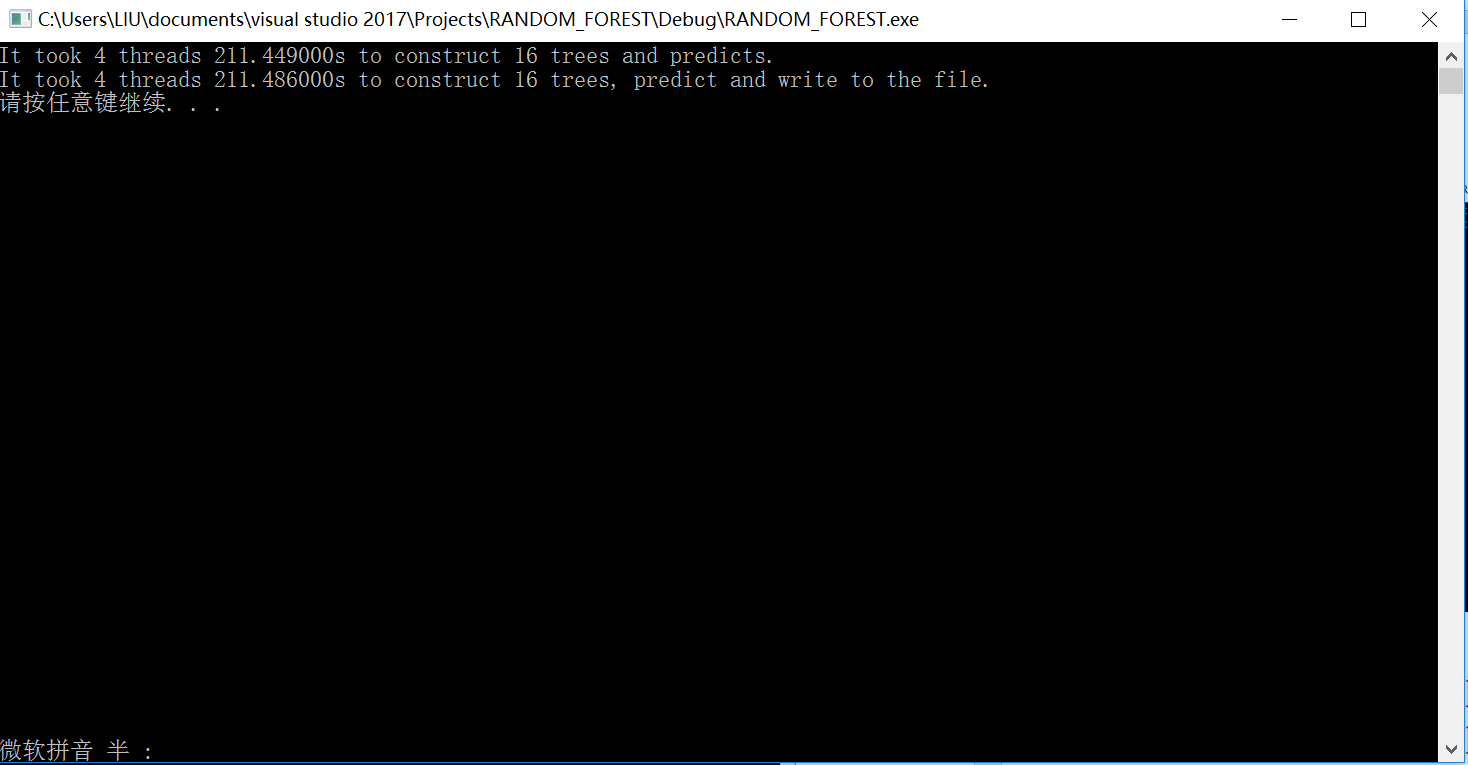
优化数据结构，去掉加锁操作后，1000个样本，10000个测试样本，2个线程，16棵树，花了302秒，建1棵树的平均时间为19秒：



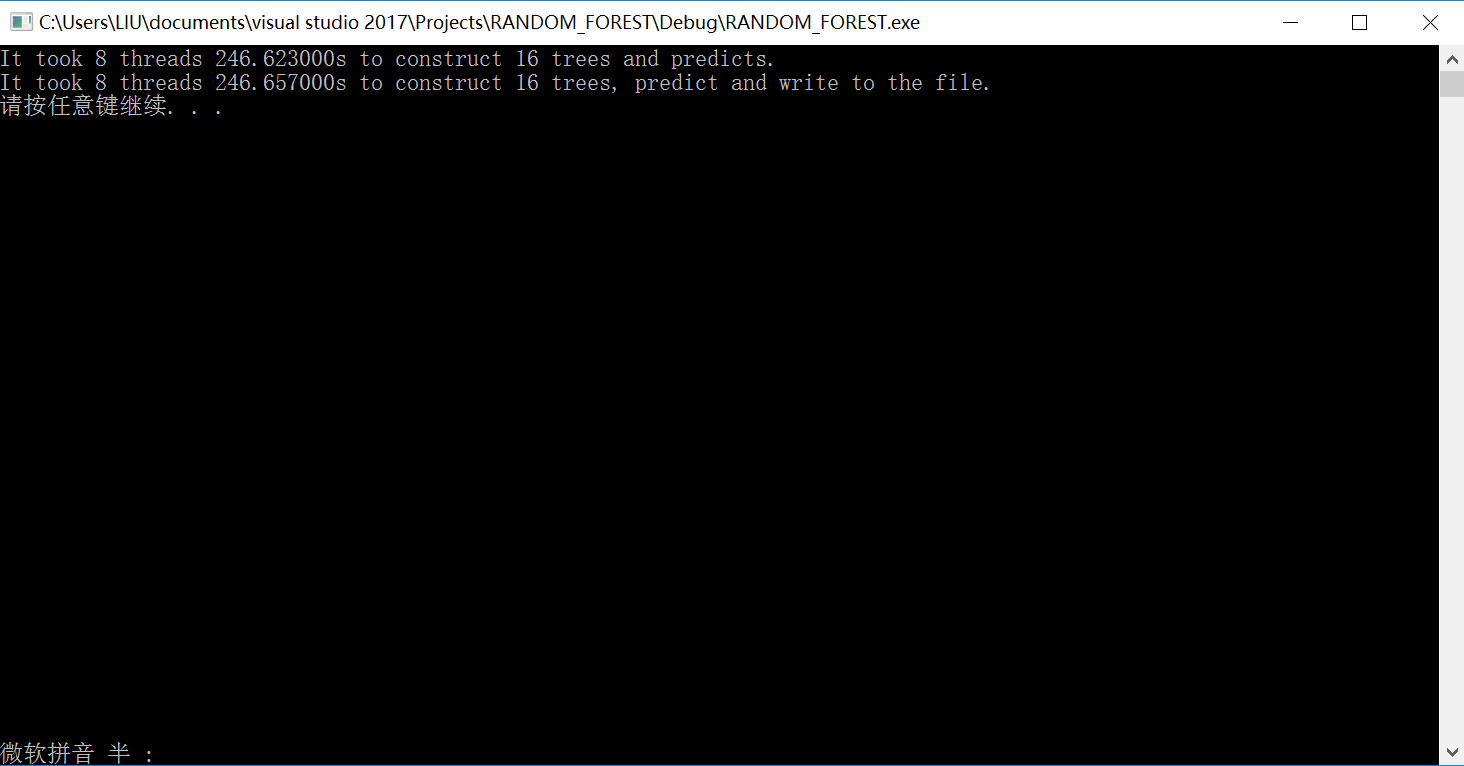
1000个样本，10000个测试样本，4个线程，16棵树，花了259秒，建1棵树的平均时间为16秒：



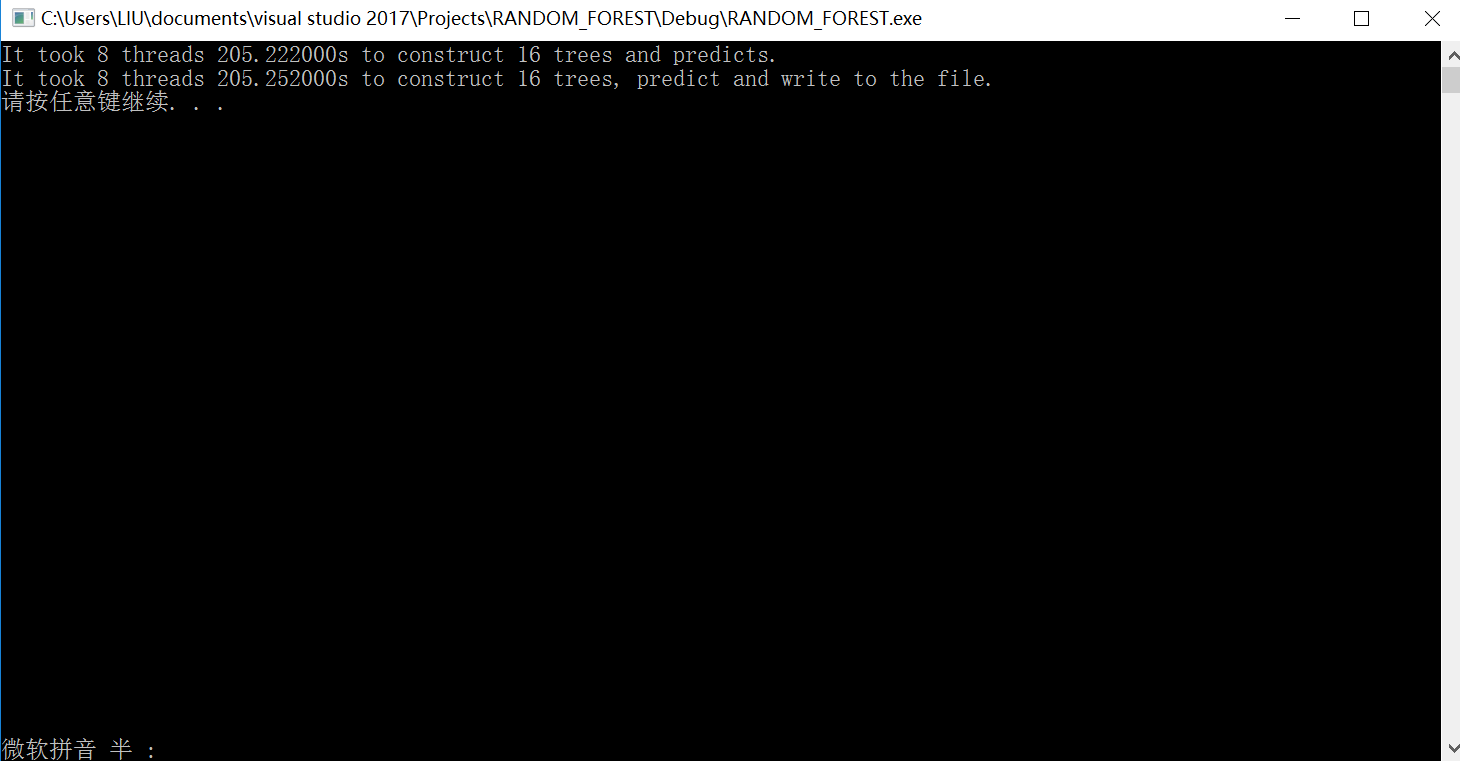
优化数据结构，去掉加锁操作后，1000个样本，10000个测试样本，4个线程，16棵树，花了211秒，建1棵树的平均时间为13秒：



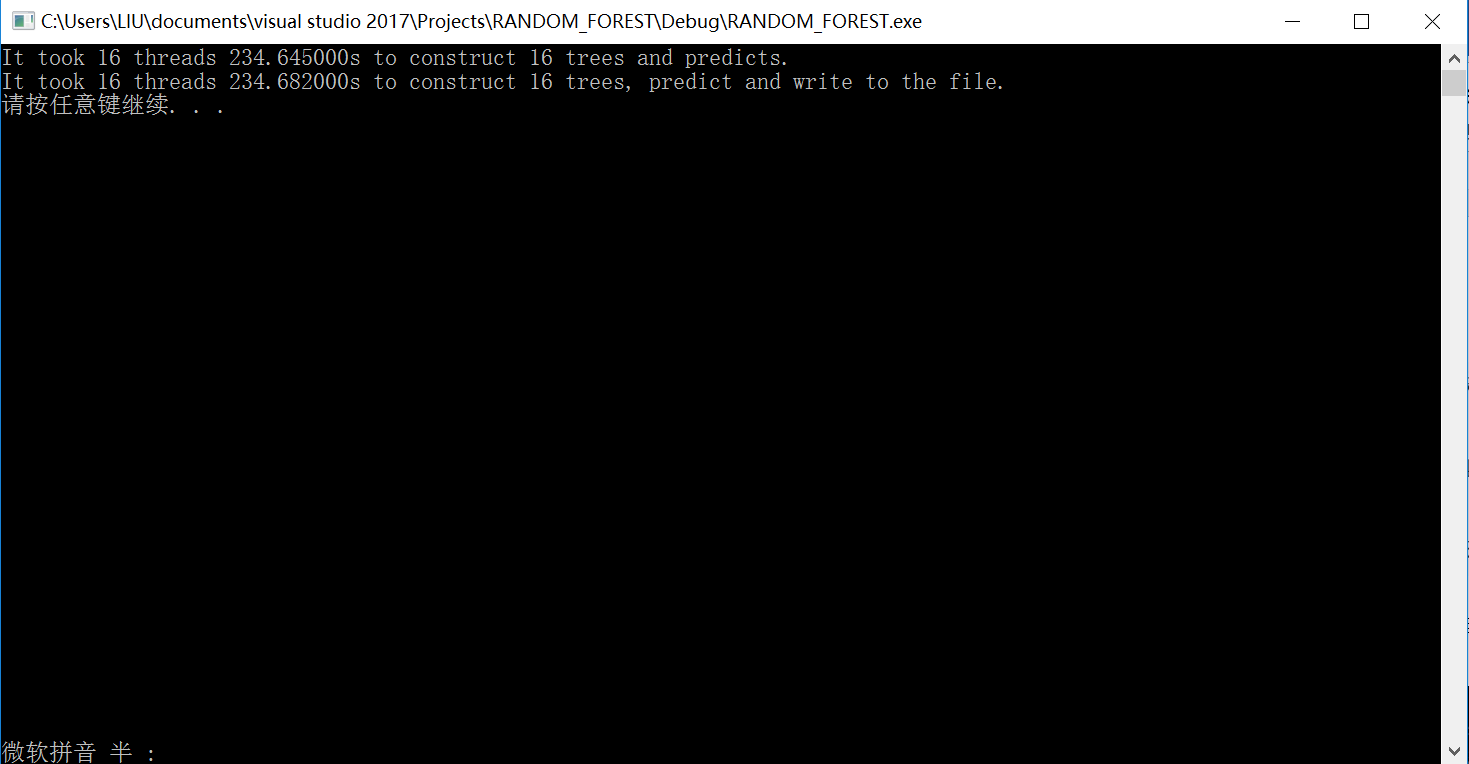
1000个样本，10000个测试样本，8个线程，16棵树，花了236秒，建1棵树的平均时间为14秒：



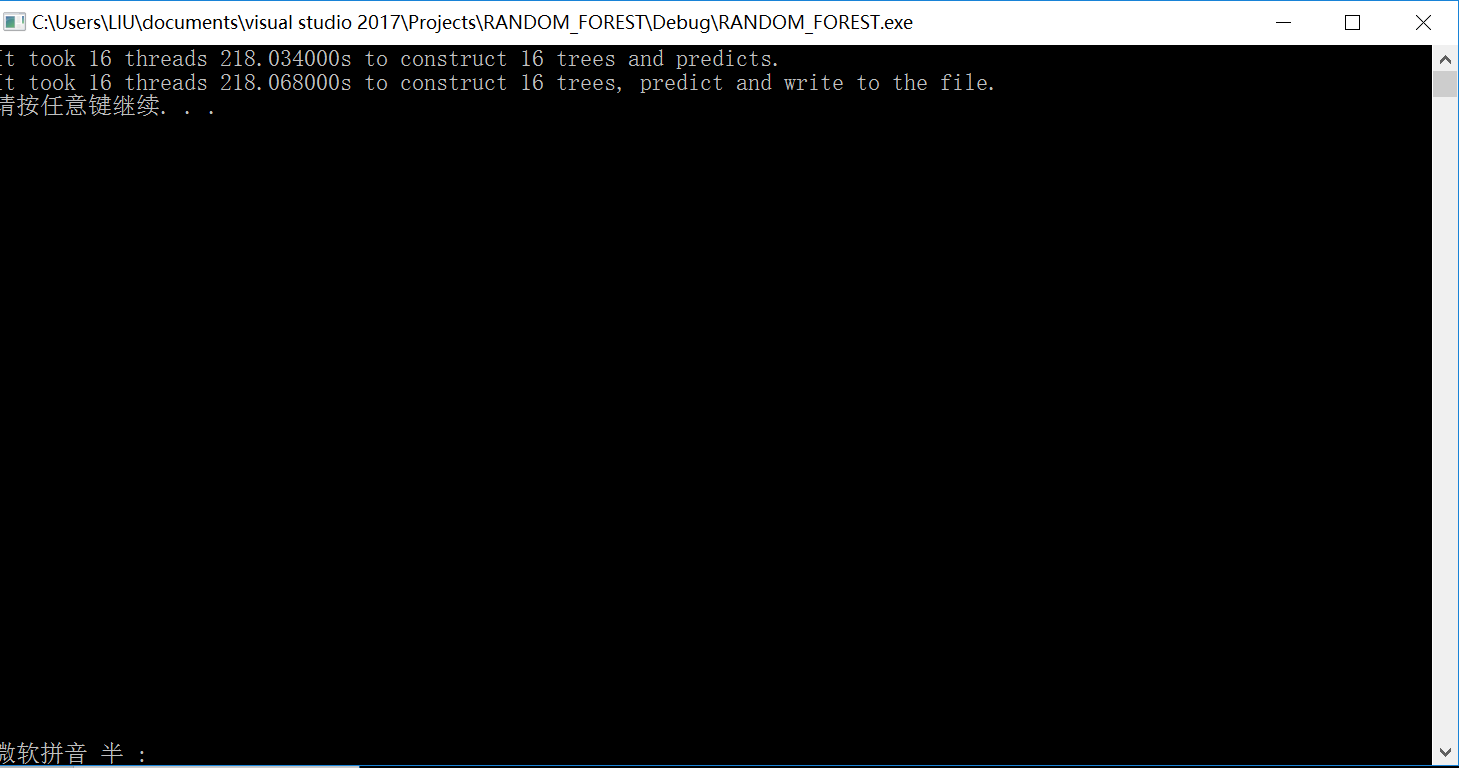
优化数据结构，去掉加锁操作后，1000个样本，10000个测试样本，8个线程，16棵树，花了205秒，建1棵树的平均时间为13秒：



1000个样本，10000个测试样本，16个线程，16棵树，花了234秒，建1棵树的平均时间为14秒：



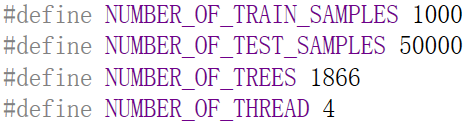
优化数据结构，去掉加锁操作后，1000个样本，10000个测试样本，16个线程，16棵树，花了218秒，建1棵树的平均时间为13秒：



从下图中不难看出，结合我电脑是双核的条件，设置4个线程应该是我电脑所能跑得最好的效率，当然如果电脑是四核的，线程数可以适当增加以提高效率：

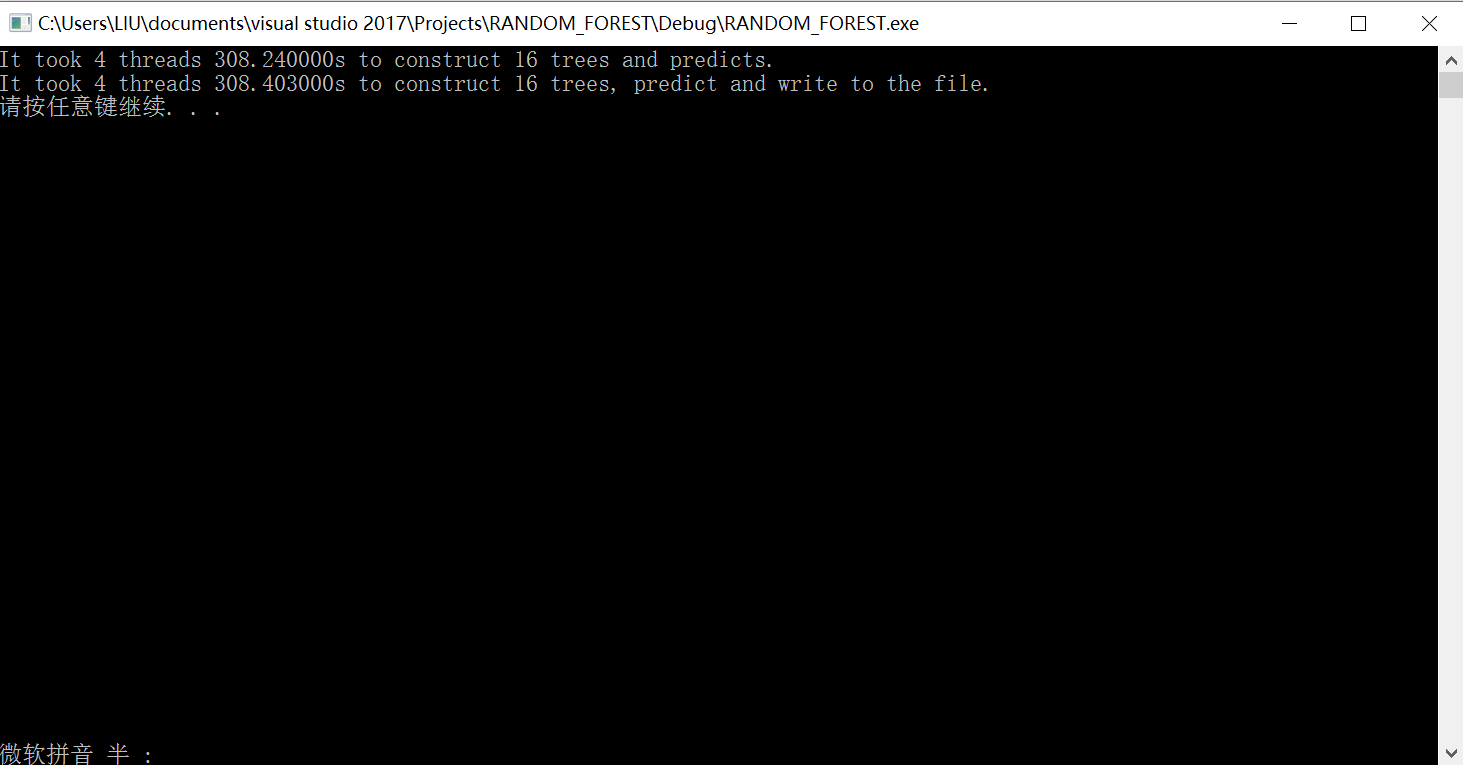


最终实际进行数据挖掘时，如下设置参数，采集到的样本数为1866000约等于实际数据量1866819。程序至少需要跑10次，预测的数据达500000条超出实际待预测的494851条数据，因此第五次运行程序的时候要将NUMBER\_OF\_TEST\_SAMPLES改成44851：



因为内存容量有限，无法打开一个494851\*201的数组，最终选取50000\*201的数组来存储待测数据是因为预测数据要push\_back进入一个容器中，容器超出了一定大小之后效率又变得很低，读测试文件要一段时间，内存占用超过1G，因此最终选取50000\*201的数组来存储待测数据。每次预测完50000个数后，就删除test\_data.txt的已预测的50000个数据来进行下一轮的预测，最后将10个输出文件合并到一个文件里面提交kaggle上，虽然这样不太符合最初的随机森林算法，但是从理论上来说这只是建了好10个随机森林来分别预测10%的测试数据，结果出来还不错，kaggle上有0.90754的预测率。

因为当时没有截图，但是现在可以大概估计跑一次程序的时间，以下是1000个样本，50000个测试样本，4个线程，16棵树的结果，总共花了308秒，可以算出1000个样本，50000个测试样本，4个线程，1866棵树大概需要10个小时才能完成：



**小结**

对于前期开发过程，大量的时间都在处理效率的问题上，因为一开始建树和预测都需要花大量时间，这个时间是不可接受的，为此多次调整了数据结构，真的是越到后期，调整的代码量就越大。

从这次的优化的经验中，我得出的总结如下：慎用大的容器和大的对象，容器里面如果包含大的容器或者对象，在赋值的时候会极大的降低效率。CPredictNode类的使用完全就是为了节省时间，因为CNode的变量和函数很多，使用CNode来遍历会非常慢，这个差距会是上千倍的差距。在测试时，测试集和训练集都选择大小为500，结果建一棵树平均需要16秒，而使用CNode完成预测将需要250秒，而使用CPredictNode类完成预测的时间不到0.1秒。

后面并行化的时间主要用在处理死锁的问题，这个真的是非常头疼，因为是第一次遇到，曾经看着代码一小时不知道从何下手，因为不知道死锁发生在什么位置，最后还是通过强行对所有线程都加上锁，使建树过程能够按找线程顺序执行，再慢慢缩小锁住的范围来解决问题。而最后改变数据结构的时候，去掉mtx.lock()和mtx.unlock()程序也能正常运行，因此还能提高了效率。

这个程序前前后后打了半个月，代码行量有500多，应该是纯c/c++自己写过最多代码量的程序，比起BP神经网络，梯度下降那些有数学公式摆在那里的算法，随机森林不但涉及数据结构，还设计并行化设计，虽然随机森林的原理并不难，但实现起来还是和做决策树的题目完全不一样，要考虑到空间，效率的问题，在不能超出内存的空间大小来编码的同时，还要注意程序效率的问题，否则一不留神就要花好几个月才能得出预测结果。虽然做了很多的优化工作，但是因为数据集和测试集本身就很庞大，为了得到第一个可以提交的txt，我每天晚上睡觉之前跑程序，早上醒来看结果，这样重复了10天，可能一台高配置的电脑或者像我同学拿到别人服务器上跑会比较快吧。