**数据库课程设计实验报告**

**第一部分 - 小组成员**

余明静 15331382 任务一，任务四

张凌赟 15331396 任务二，任务三，实验报告的编写

曾志贤 15332003 任务二，任务三，实验报告的编写

张宇飞 15331408 任务四

田桂雄 15332014 实验报告的编写

**第二部分 - 实验环境**

操作系统： windows

程序语言：C++

**第三部分 - 实验内容**

根据论文《Maximum Inner-Product Search Using Cone Trees》实现Ball-tree的C++外存版，具体包括如下任务：

任务1：实现ball-tree的建树过程

 任务2：实现将ball-tree写进外存的功能，按照定长记录存储

* 要求按二进制的页格式存储，缓存页的大小设为𝐵 = 64𝐾
* 每个槽存储一个树节点
* 给每个树节点指定ID，按树节点ID查找和存储
* 叶子结点和非叶子结点分开存储
* 数据对象直接存放在叶子结点中

 任务3：实现从外存中载入ball-tree的功能

* 不要将整棵树载进内存
* 用尽可能少的内存消耗，完成尽快的查询

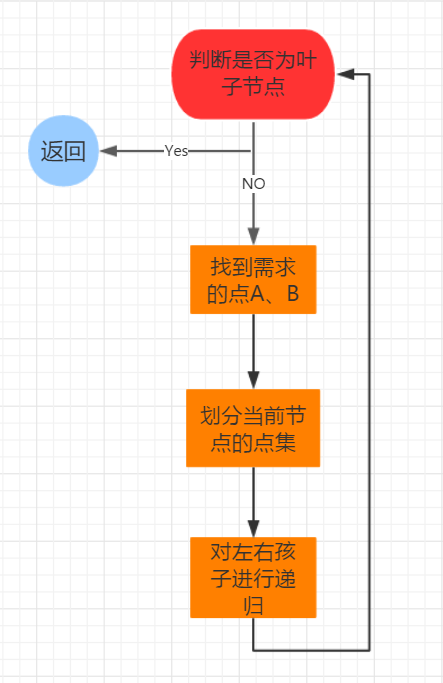
 任务4：实现查询阶段找到最大内积对象并剪枝的功能

**第四部分 - 实验思路和设计**

1. **实现ball-tree的建树过程**

* **建树算法：**任取一点为x，找到距离他最远的点A，再找距离A最远的点B，根据点距A、B的距离将点划分在不同的圆内，运用递归，直到圆内部的点的个数少于20个时，为叶子节点。当所有节点内点少于20个时，建树完成。
* **建树过程：**

1. 增添BallTreeNode结构作为树的节点进行存储
2. 在BallTree类里buildTree函数中调用MakeBallTree(root, n, d, data)进行建树，其中n为圆内点的个数，d为点的维数（每个点为50维，故d = 50）
3. MakeBallTree（）中实现具体的建树过程。



①新建一个树节点，判断是否为叶子节点，如果是则返回，否则向下执行

②通过MakeBallTreeSplit（）找到满足条件的点A、B

③将父节点圆内的点根据距离A、B的距离比较判断点应该划分到哪个子节 点的圆内

④对父节点的左右孩子进行递归直至到叶子节点

1. 当所有节点圆内的点的个数都少于20的时候建树完成

* **使用函数：**

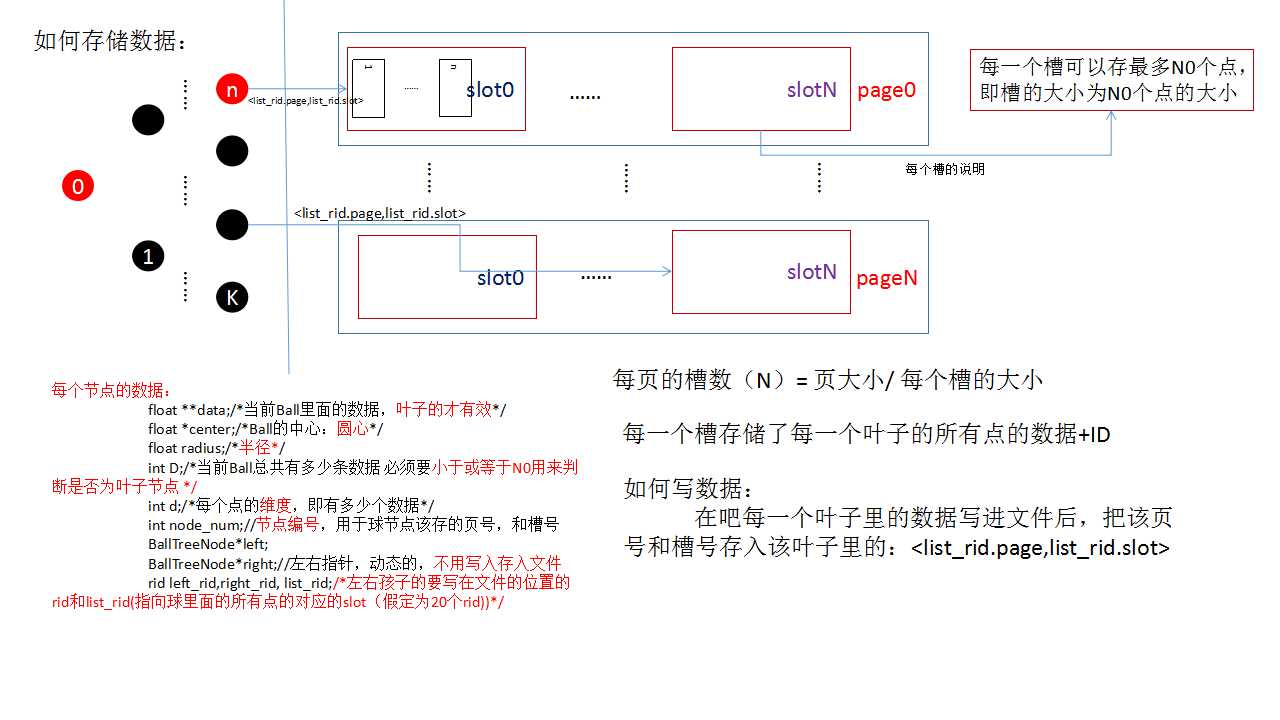
buildTree(int , int , float \*\*) // 建树

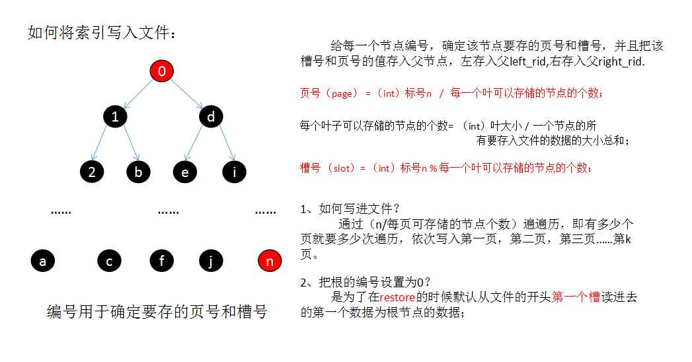
MakeBallTree(BallTreeNode \*&, int , int , float \*\*&) // 递归函数

MakeBallTreeSplit(float \*\*& , int , int , int & , int & ) // 获取点A、B

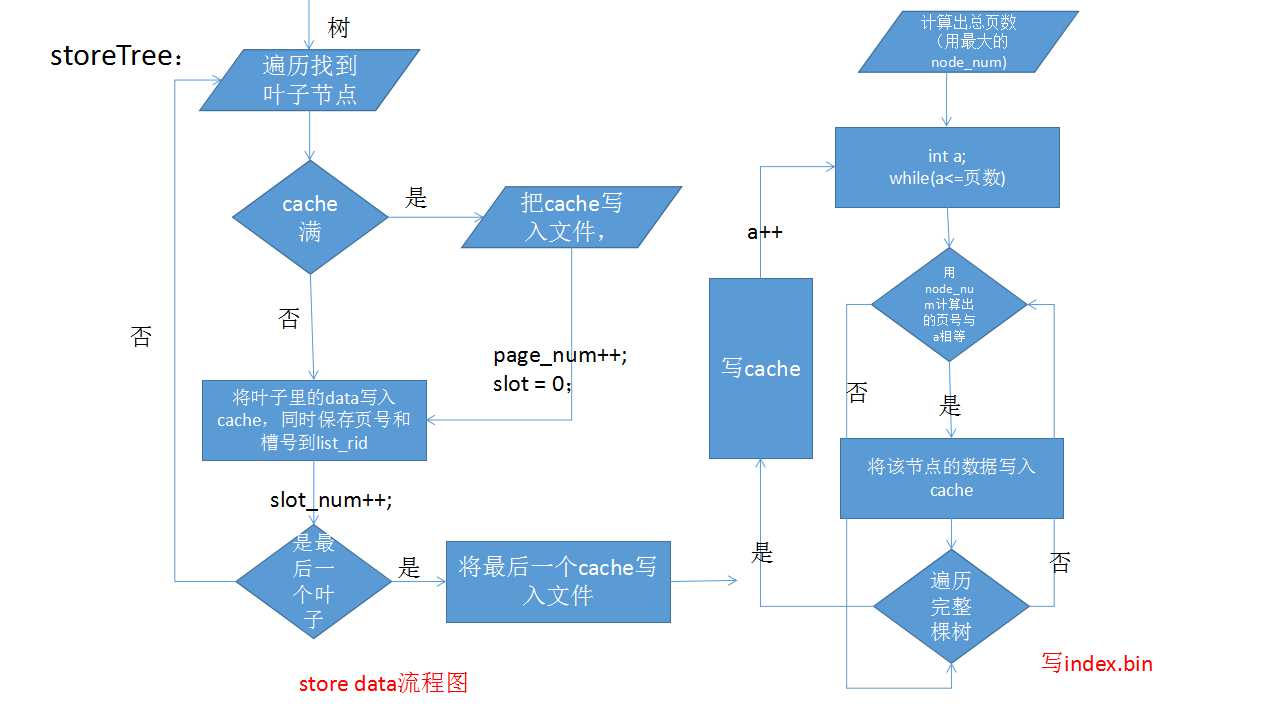
1. **实现将ball-tree写进外存**

* **实现思路和实现过程：**





* **实现流程图：**



* **实现过程：**

1. 将数据写入文件，在叶子上的list\_rid上记录页号和槽号
2. 给每一个节点编号；
3. 计算每一个节点要存的页号和槽号，并存入父亲的rid中；
4. 依次把存入第0页，第一页……的节点的数据存入缓存，把缓存写入文件

**（三）实现从外存中载入ball-tree的功能**

* **实验思路：**

1. **只载入根节点**：默认在索引文件的第一页第一个槽

bool BallTree::restoreTree(const char \* index\_path)；

只是载入了根，没有载入其他。

1. **载入左右孩子或者是叶子数据：**

实现另一个函数：

void prebulidtree\_index(BallTreeNode\* &root,FILE \* &index\_bin, float cache[PAGE\_SIZE/sizeof(float)])；

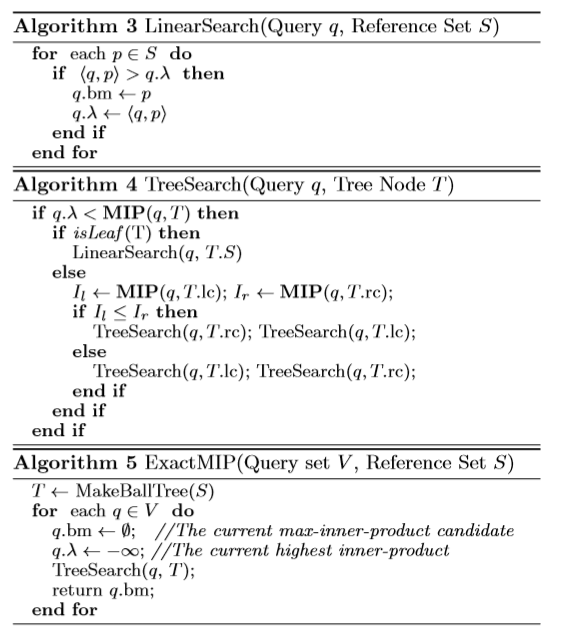
传入父亲的节点，并载入他的左右孩子，如果是叶子，就是载入数据。

这样在查询的时候就可以用这个函数，**不用一次载入整棵树。**

如果这个函数放在restoreTree中执行，且是递归，那么久是载入整个树了。

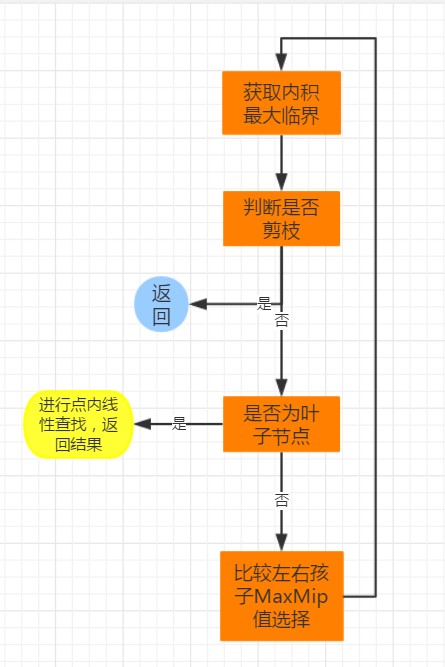
**（四）实现查询阶段找到最大内积对象并剪枝的功能**

* **实现思路**：根据论文内容，实现伪代码



* **实现过程：**

1. 建立Query结构，添加查找
2. TreeSearch() 函数里面将进行叶子节点的查找



① 首先，在MaxMip() 里获取当前节点内积最大临界值，通过比较查找与内积最大临界值来判断是否剪枝，如果需要剪枝，则直接返回，否则继续执行

② 根据点的个数D判断当前是否为叶子节点，如果为叶子节点则进入LinearSearch()函数在D个50维的点内进行数据的线性查找返回结果，否则继续执行

③ 当前节点为非叶子节点，算出左右孩子的MaxMip，哪个孩子的值大，则往该孩子节点进行递归遍历，最终将只有一条路线

* **使用函数：**

TreeSearch(float\* , BallTreeNode\*, int , Query&) // 查找

MaxMip(float \*, BallTreeNode\*, int ) // 获取最大临界内积值

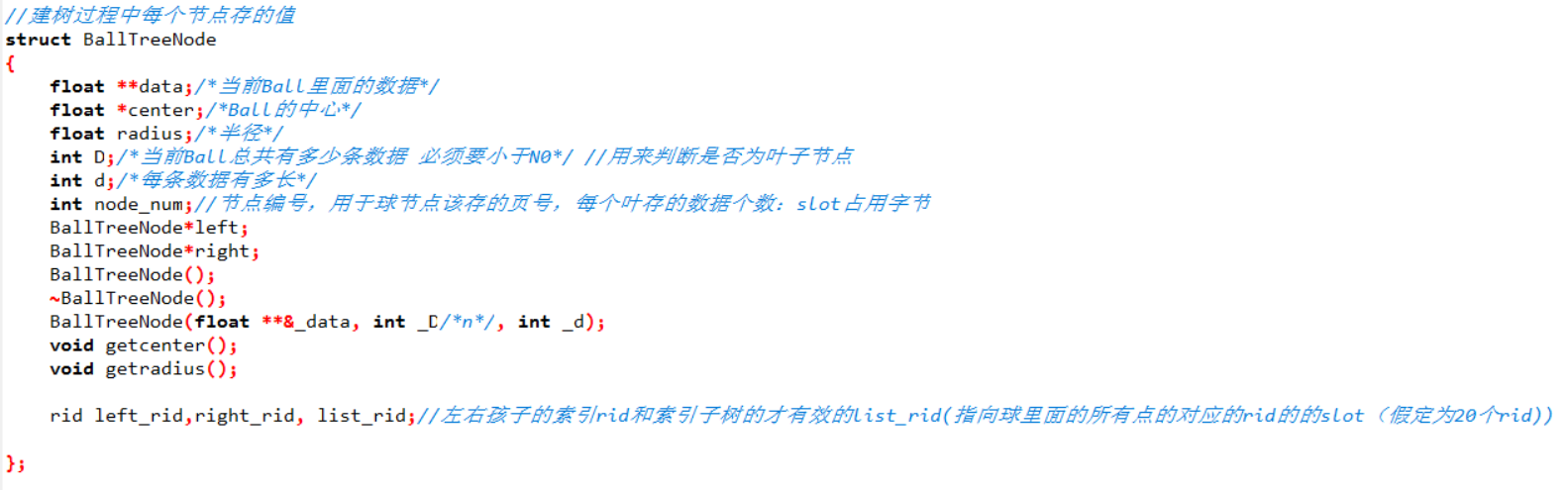
InnerProduct(float\*, float\*, int ) // 求内积

LinearSearch(float\*, BallTreeNode\* , int , Query&) // 点内的线性查找

**第五部分 - 代码结构与实现**

1. **实现ball-tree的建树过程**

* **每个节点存储的值**



* **使用函数：**

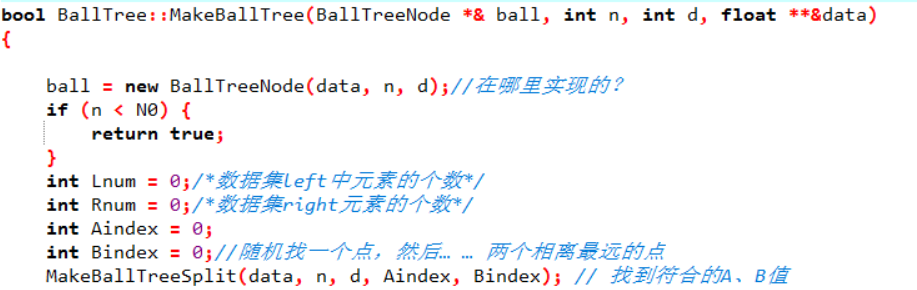
buildTree(int , int , float \*\*) // 建树

MakeBallTree(BallTreeNode \*&, int , int , float \*\*&) // 递归函数

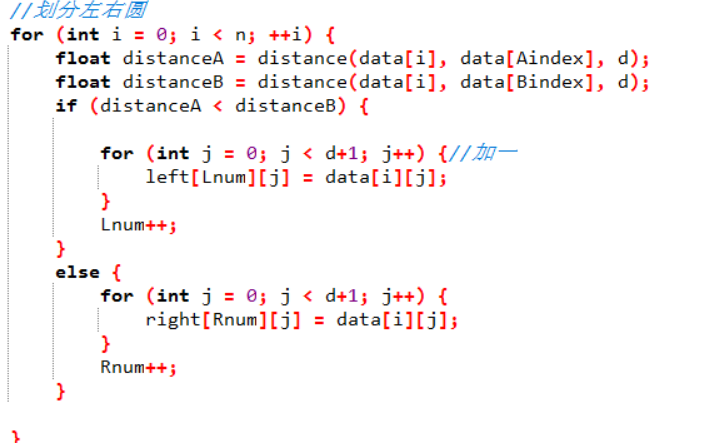
MakeBallTreeSplit(float \*\*& , int , int , int & , int & ) // 获取点A、B

* **实现思路**

具体物理量：



递归划分左右圆：



1. **实现将ball-tree写进外存**

* **使用函数**

storeTree(const char \* index\_path)

write\_data(data\_bin, root, page\_num, slot\_num,cache); // 写入数据的递归函数

write\_index(root,index\_bin); // 写入索引

setnode\_num(root);//给每一个节点编号1.2.3.4……用于确认页号和slot

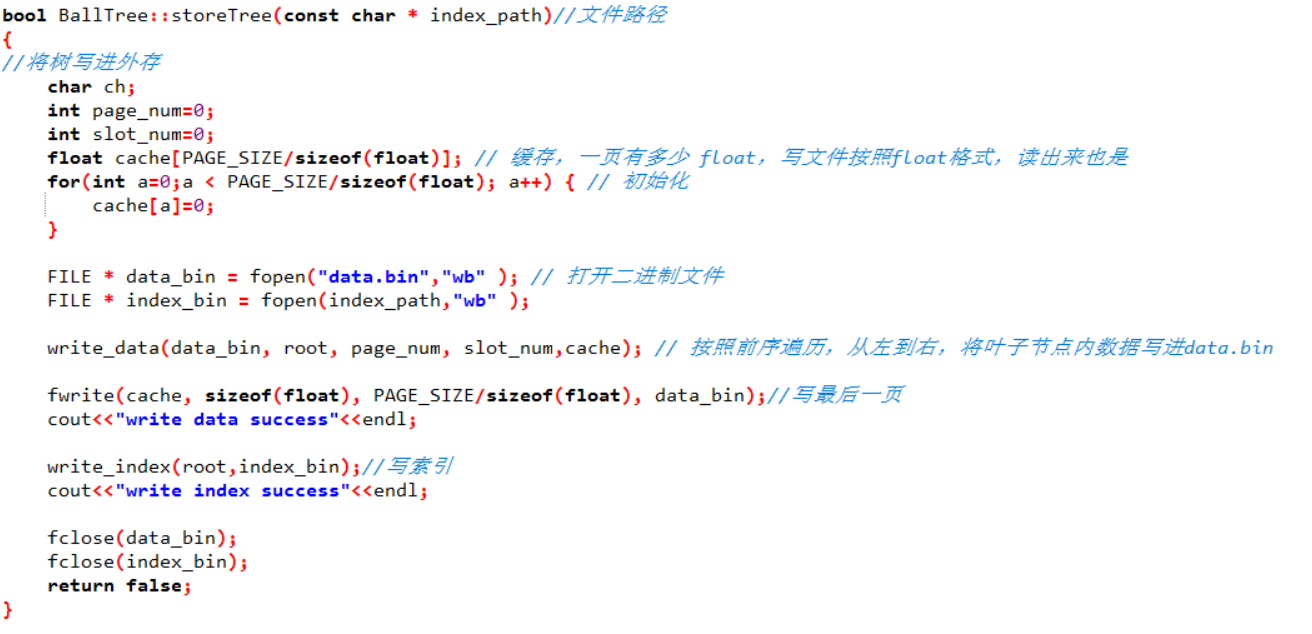
set\_page\_slot(root);//确定每个节点的左右子树要存的位置，页号 和 slot

w\_n\_to\_index(root,cache,index\_bin); // 写进index.bin，存的是每个节点的全部数据

write\_apage(root,cache,slot\_num,a); // 写第a页到cache

preorder\_setnum(BallTreeNode\* &root,int & num); //前序遍历每个节点id号

* **实现思路**



1. **实现从外存中载入ball-tree的功能**

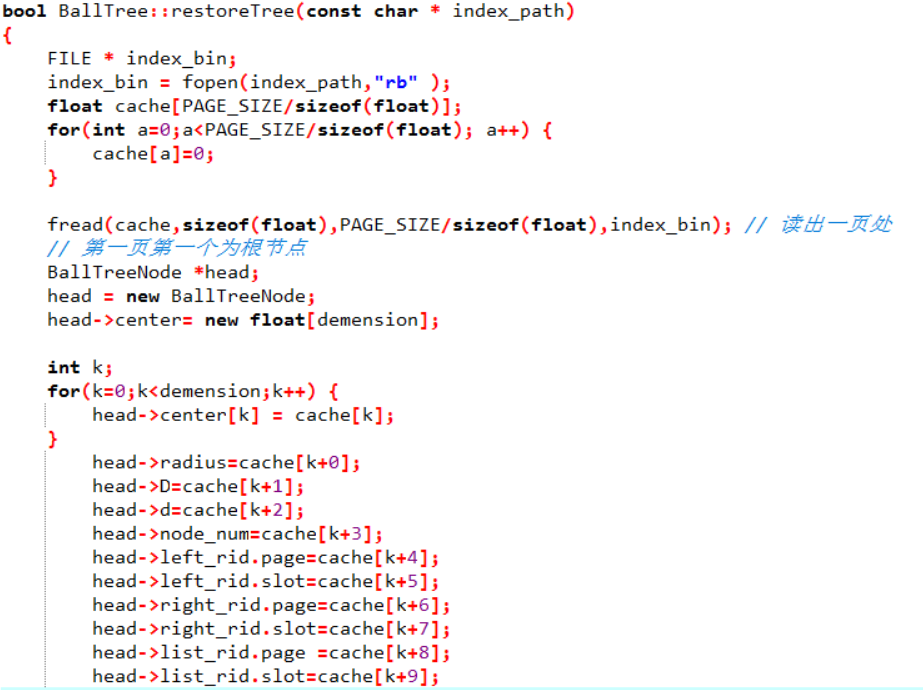
* **使用函数**

restoreTree(const char \* index\_path)；//从外存读出树

prebulidtree\_index(BallTreeNode\* &root,FILE \* &index\_bin, float cache[PAGE\_SIZE/sizeof(float)]); //获取槽内内容

cin\_data(BallTreeNode\* &root); // 根据叶子节点的页号槽号在data.bin找出对应的数据读进内存

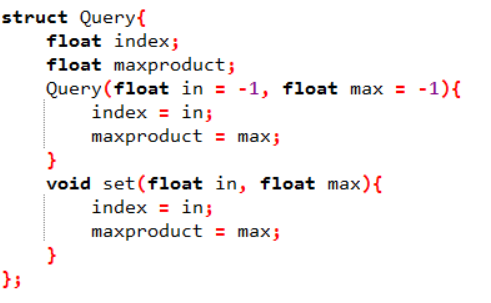
* **实现思路**





1. **实现查询阶段找到最大内积对象并剪枝的功能**

* **查找结构**



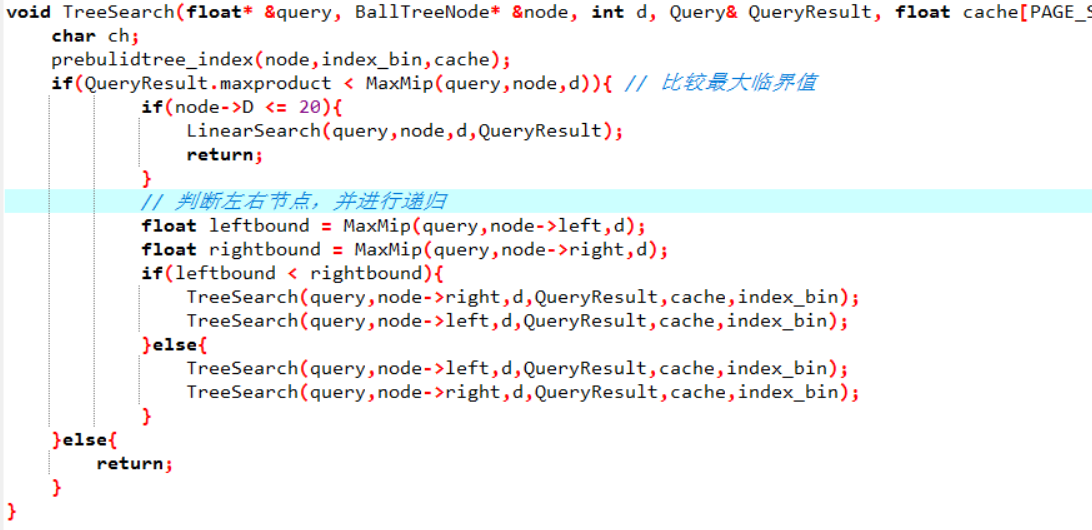
* **使用函数：**

TreeSearch(float\* , BallTreeNode\*, int , Query&) // 查找

MaxMip(float \*, BallTreeNode\*, int ) // 获取最大临界内积值

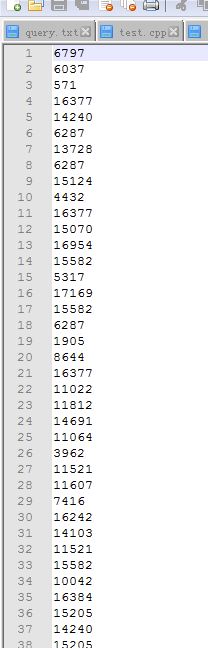
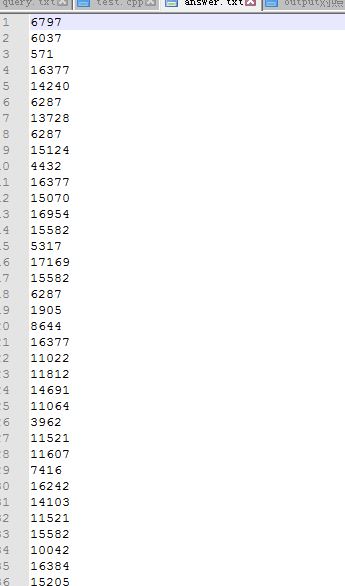
InnerProduct(float\*, float\*, int ) // 求内积

LinearSearch(float\*, BallTreeNode\* , int , Query&) // 点内的线性查找

* **设计思路**

**第六部分 - 实验结果与性能比较**

通过存储树前的查询结果 和 存储后倒入的树的查询结果对比可以知道存储树和导入树是没有问题的



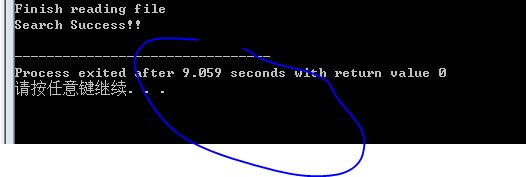
* **性能：**

1.整个过程：建树，存储树，导入树，查找1000个



内存开销700兆左右，内存释放不完全。

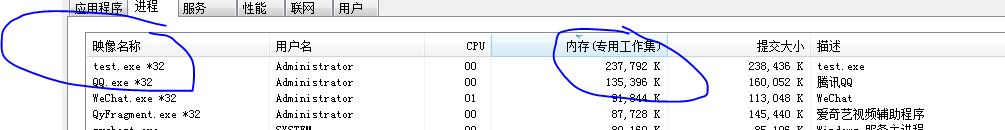
2.用时：整个过程



查询1个数据的用时：



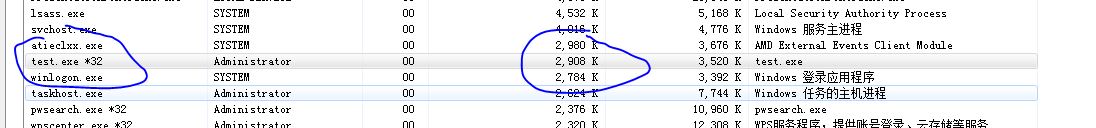
3.没有建树，只执行导入树，内存消耗，查找1000个



200兆左右，内存消耗较大，应为倒入后就没有释放的过程；

**说明没有倒入整棵树：**

没有建树，只执行导入树，内存消耗，查找1个



内存消耗3兆左右，说明是没有一次性载入所有的树。

**第七部分 - 心得与体会**

因为已经较长时间没有接触C++编程了，组内成员在编写之前明确了集体编程需要注意的一些问题。从整体编程看，节点内部存储变量较多，在集体编程中，对每一个变量进行正确命名很重要，给了大家一个共享变量的便利。其次，彼此函数之间需要提供好用的接口供大家使用，前一个任务要根据后一个任务提出的需求进行接口的完善与供应，使得项目可以正常的执行。同时，还需要适量增加代码注释，让后面的同学可以较好的理解前面的代码。

【编程过程】

由于任务的较细分配，每个人的任务针对性都比较强，需要掌握的知识点以及参考的资料都不尽相同。在建树任务中，重点是了解清楚ppt上的建树算法思路，之后根据该思路进行编程。增加了一个BallTreeNode结构用于存储各个节点的变量，之后所需要的函数中需要用到众多变量、指针，变量使用上很容易造成混淆，所以在debug过程中使用了肉眼debug，艰难啊。最终test的时候，还是发生了数据的崩溃，小数量的可以正常执行，但是大数量的就会爆掉，后来发现是一处判断出了错，导致圆内的点个数多于1个时不能完全存进去。

而在存进外存的过程中，真的是比较遇到的坑比较多，我们推翻了第一次的思路，而用了另外的方法。在第一次的预想中，想用三个文件，索引文件中的槽存叶子节点的相关值以及20个指向<list-pageid, list-slotid>的索引，但是后来难于实现便讨论着换了另一个方法，在这个过程中经历了放弃一个想法到实现另一个想法的过程，着实艰难与纠结。最后的搜索也是经历了爆掉再debug到完善的过程，重在实现论文中的伪代码。

【体验】

这次试验实现了从0到1的一个实现过程，第一次仔细阅读学术性的论文，第一次建立自己的实现思路并且从推翻到重建。对C++有了巩固，同时也对query有了更深的理解，小组成员的多次第一次组成了这次的结果，熬了好几次夜终于把任务完成了，很辛苦但是也很开心。在过程中，了解了数据库的实现，将平时课上的页、槽的理论概念应用于实践之中，对他们的概念有了更深的理解，而不是拘泥于课本的几个图和几句话，实践出真知，将实践与理论结合，更好的学习数据库，更好的了解程序。