《数据结构与算法》实验报告

学生姓名		**院（系）**	计算机科学与技术
学　　号		**专　业**	
实验时间		**实验地点**	
实验项目	**实验 4/4：**查找结构的实验比较		

实验目的：将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。

实验要求：灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。

实验内容：BST 查找结构与折半查找方法的实现与实验比较

本实验要求编写程序实现 BST 存储结构的建立（插入）、删除、查找和排序算法；实现折半查找算法；比较 BST 查找结构与折半查找的时间性能。

1. 设计 BST 的左右链存储结构，并实现 BST 插入（建立）、删除、查找和排序算法。
2. 实现折半查找算法。
3. 实验比较：设计并产生实验测试数据，考察比较两种查找方法的时间性能，并与理论结果进行比较。以下具体做法可作为参考：

（1）第 1 组测试数据：n=1024 个已排序的整数序列（如 0 至 2048 之间的奇数）；第 2 组测试数据：第 1 组测试数据的随机序列。

（2）按上述两组序列的顺序作为输入顺序，分别建立 BST。

（3）编写程序计算所建的两棵 BST 的查找成功和查找失败的平均查找长度（主要是改造 Search 算法，对“比较”进行计数），并与理论结果比较。

（4）以上述 BST 的中序遍历序列作为折半查找的输入，编写程序分别计算折半查找的查找成功和查找失败的平均查找长度，并与理论结果比较。

（5）以上实验能否说明：就平均性能而言，BST 的查找与折半查找差不多，为什么？

数据结构定义：

```
/*结点定义*/
typedef struct CellType
{
    records data;   /*数据*/
    struct CellType * lchild;   /*左子树*/
    struct CellType * rchild;   /*右子树*/
```

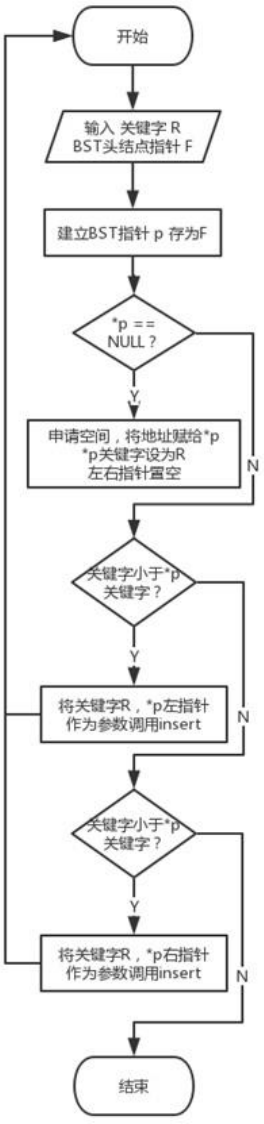
```
} * BST;

int arr1[MAX_LENTH] = {0};   /*1024 奇数有序数组*/
int arr2[MAX_LENTH] = {0};   /*1024 奇数乱序数组*/
int C = 0;   /*ASL 次数（路径长度）记录变量*/
int cot = 0;   /*记录失败查找的结点数量*/
int asl = 0;   /*asl 开关*/
int cnt = 0;   /*数组下标变量*/
```

算法设计与分析（要求画出核心内容的程序流程图）：

1. BST 的建立算法

BST 的建立（插入）算法，输入 BST 的头结点指针，通过关键字与结点关键字的大小比较，递归寻找插入点。

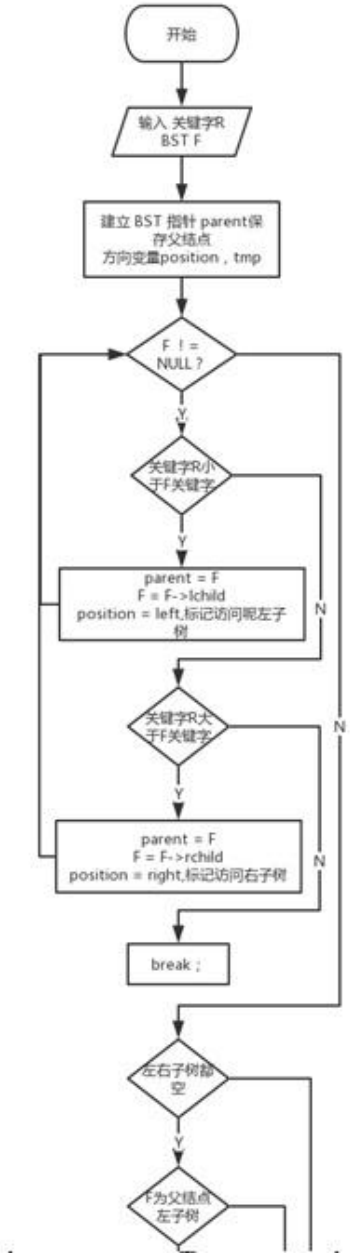


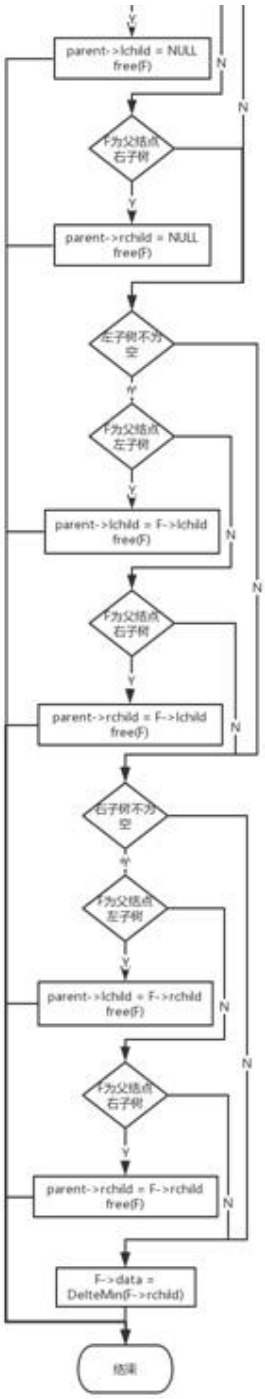
2. BST 的删除算法

删除分为三种情况：

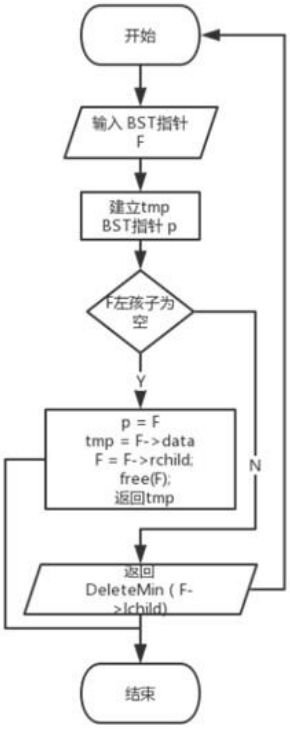
1. 删除结点为叶结点
2. 删除结点有一个子树为空
3. 删除结点两子树都不为空

叶结点比较简单，只需要将父结点的指向指针置为 NULL，一个子树情况，需要将该子树与父结点相连，两个子树时，需要找到左子树的最大值或右子树的最小值与结点交换，再删除那个叶结点。





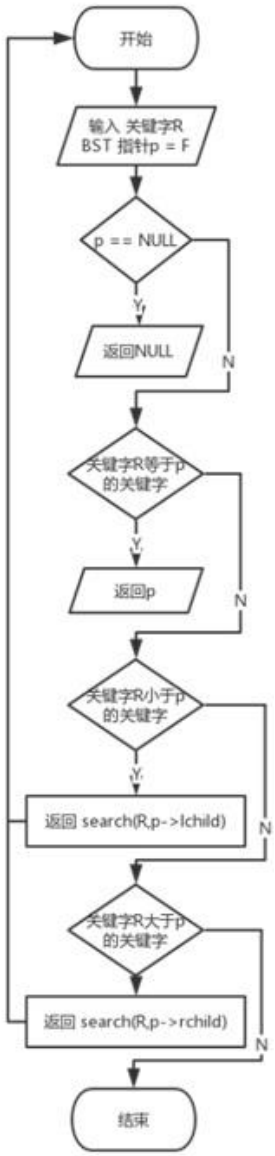
Delete 算法



DeleteMin，找到右子树最小

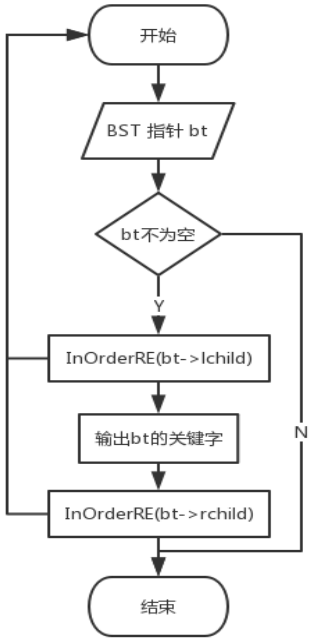
3. BST 的查找算法

使用递归，查找结点地址，找到返回结点地址，若未找到，返回 NULL



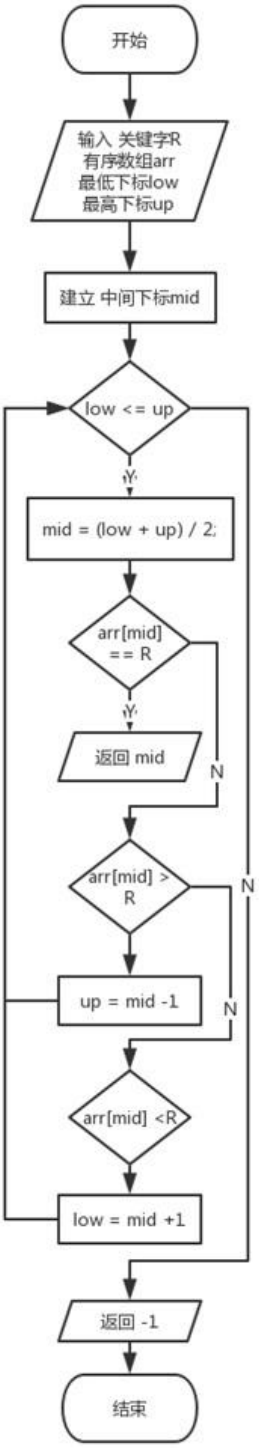
4. BST 的排序算法

   即为 BST 的中序遍历算法



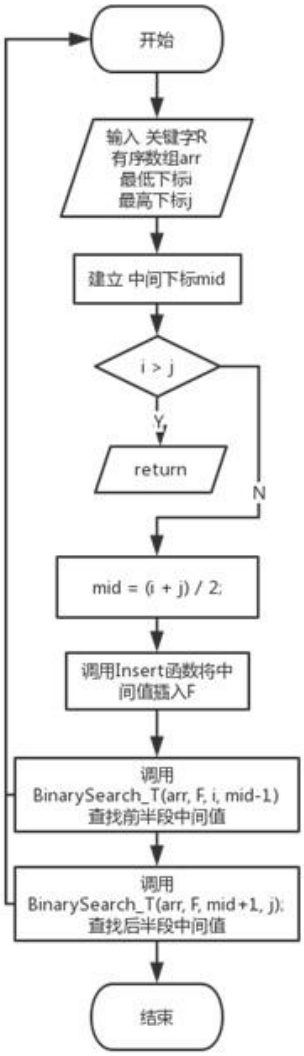
5. 折半查找算法（数组）

   折半查找的非递归算法



6. 折半查找树建立算法

将折半查找算法改造为折半查找树的建立算法，每次取有序数组中间值插入 BST 中。

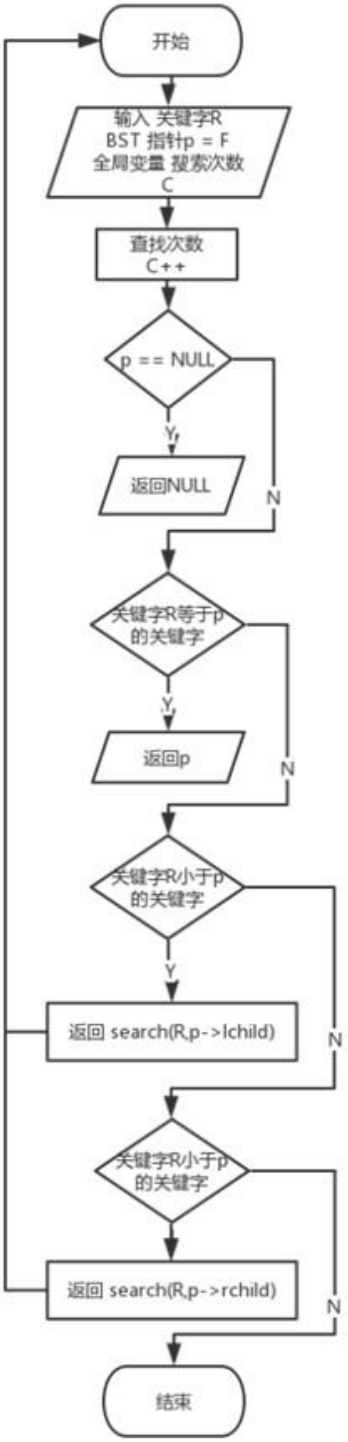


7. 成功查找次数平均长度算法

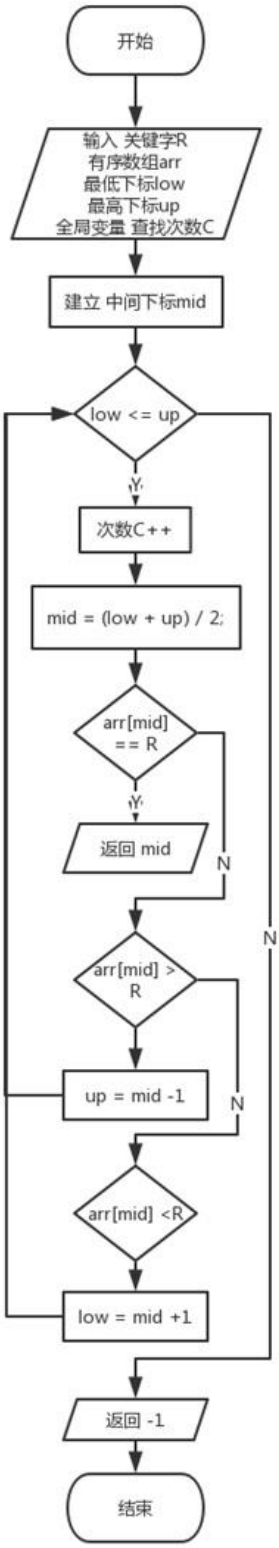
将 search 算法改造，每次需要访问其左右孩子时，访问路长（次数）加一，直到查找结束，将每个结点的查找长度的总和除以结点数，得到成功 ASL，包括两种算法，一种是对于二叉查找树的 search 算法，一种是折半查找的 ASL 算法。

通过调用改造 search 函数对 1024 个奇数进行搜索，计算每次的 C，乘上概率 P，累加得到 ASL。

折半查找相同，通过调用改造过的折半查找函数对 1024 个奇数进行搜索，计算每次的 C，乘上概率 P 1/1024，累加得到 ASL。



改造后的 search 算法



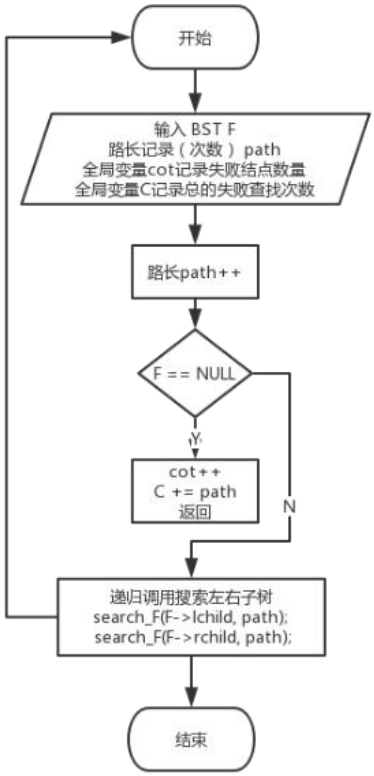
改造后的折半查找算法

8. 失败查找次数平均长度算法

失败查找次数实质上就是访问 BST 的空节点数量，所以使用搜索的递归算法，每当访问到一个空节点，结点统计变量 cot 加一，同时总路长 C 加上

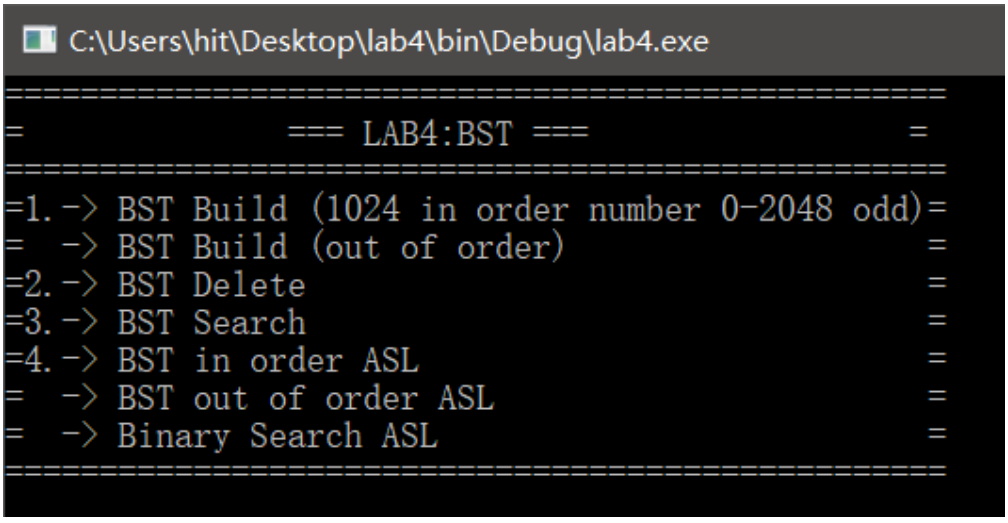
该节点访问次数，最后得到失败 ASL。

最后将 C 除以失败结点数量 cot，得到 ASL

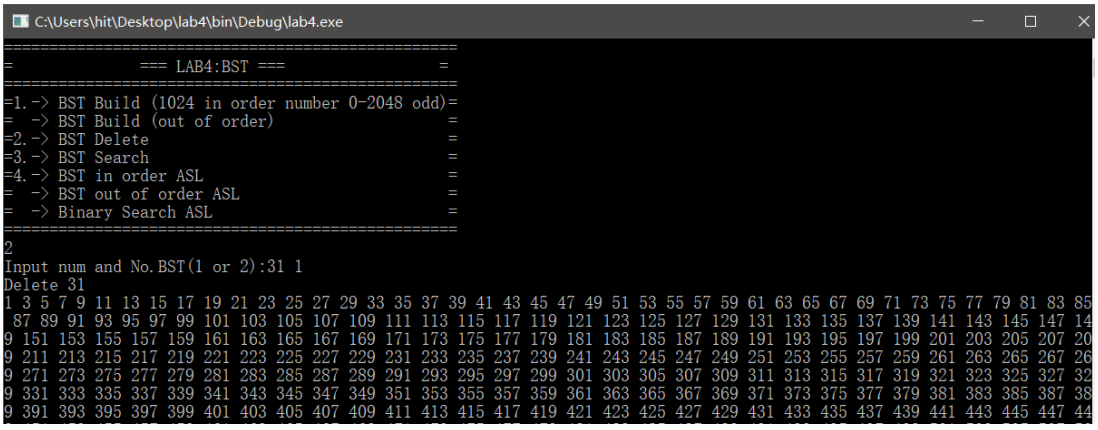


失败查找算法

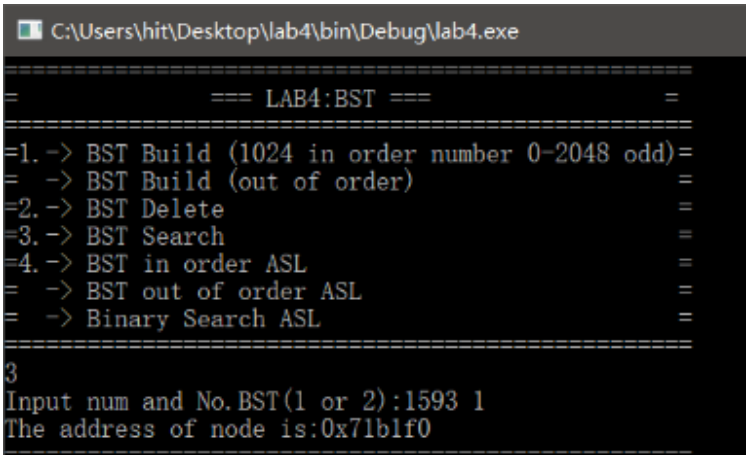
实验测试结果及结果分析：



主界面



删除结点（31）



查找结点，找到返回结点地址，图为查找顺序 BST 的关键字为 1593 的结点

实验产生两组数据

1. 1024 个顺序的在 0 到 2048 间的奇数。（1.3.5.7.……）
2. 1 中的数据的乱序

分别用 1 和 2 中数据插入生成 BST F1 和 F2，计算不同 BST 中的成功失败 ASL，然后将中序遍历顺序作为输入计算折半查找的成功失败 ASL

```
================================================
=              === LAB4:BST ===                 =
================================================
=1.-> BST Build (1024 in order number 0-2048 odd)=
=  -> BST Build (out of order)                  =
=2.-> BST Delete                                =
=3.-> BST Search                                =
=4.-> BST in order ASL                          =
=  -> BST out of order ASL                      =
=  -> Binary Search ASL                         =
================================================
4
In order Success ASL: 512.500
In order Failed ASL: 513.999
Out of order Success ASL: 15.086
Out of order Failed ASL: 17.070
Binary Search Success ASL: 9.012
Binary Search Failed ASL: 11.002
```

```
In order Success ASL: 512.500
In order Failed ASL: 513.999
Out of order Success ASL: 11.760
Out of order Failed ASL: 13.747
Binary Search Success ASL: 9.012
Binary Search Failed ASL: 11.002
```

```
In order Success ASL: 512.500
In order Failed ASL: 513.999
Out of order Success ASL: 16.681
Out of order Failed ASL: 18.663
Binary Search Success ASL: 9.012
Binary Search Failed ASL: 11.002
```

计算不同情况下的 ASL（多次随机）

（顺序数组生成 BST 的成功 ASL

顺序数组生成 BST 的失败 ASL

乱序数组生成 BST 的成功 ASL

乱序数组生成 BST 的失败 ASL

中序输入的折半查找成功 ASL

中序输入的折半查找失败 ASL）

由上述内容可见，

1. 顺序整数序列的 BST 查找成功 ASL 为 512.5，查找失败 ASL 为 513.999
2. 随机整数序列的 BST 查找成功 ASL 最好情况为 11.689，查找失败 ASL 为 13.677
3. 折半查找的查找成功 ASL 为 9.012，查找失败 ASL 为 11.002

一、

顺序输入建立的 BST 是一条只有右子树的二叉树，成功查找的理论 ASL 为

$$\frac{(1+1024)*512}{1024}=512.5$$

与实际结果相同，查找失败的理论 ASL 为

$$\frac{(2+1025)*512+1025}{1025}=513.999$$

与实际结果相同。

二、

乱序输入建立的 BST 具体情况很难计算，理想情况 BST 是一个折半查找树，折半查找树的查找成功理论 ASL 为

$$\frac{\sum_{i=1}^{10} i*2^{i-1}+11}{1024}=9.012$$

乱序输入的 BST 查找成功的 ASL 会接近与 9.012，与实际符合。

折半查找树的查找失败理论 ASL 为

$$\frac{11*1023+12*2}{1025}=11.002$$

乱序输入 BsT 查找失败 ASL 会接近于 11.002，与实际也相符。

三、

折半查找实际上就是对于折半查找树的搜索，由上述计算可知，折半查找理论的查找成功 ASL 和失败 ASL 分别为 9.012 和 11.002，与程序输出结果相符。

四、

不能说明

实验结果输出看出，乱序输入的 BST 的 ASL 与折半查找的 ASL 很接近在时间复杂度上，两者基本相同。

但是折半查找对于数据的修改，比如插入和删除，需要移动其他的结点，维持有序的时间代价为 O(n)，而二叉查找树只需要简单的查找和删除单个结点，时间复杂度仅为 O(log2n)。

因此，不能说它们性能相近，对于静态查找表，使其有序用折半查找，性能要比二叉查找树好，而对于动态查找表，并且数据量大而且无序，使用二叉查找树性能是比折半查找要好很多。

问题及解决方法：

1. 如何计算查找失败的 ASL

   计算查找成功的 ASL 是对 BST 中每一个结点进行查找，计算查找次数。由失败查找次数计算公式可知，失败查找实际上就是查找 BST 中的空结点，所以只要计算查找到 NULL 的次数进行计算即可。

2. 如何计算折半查找的失败 ASL

   折半查找算法是基于数组的结构之上，所以失败查找方法在数组上不可行。因此需要构建一个二分查找树，然后在用查找 NULL 结点的方法计算失败 ASL

3. 如何构建折半查找树

   折半查找树实际上是由有序数组构建而来，每次将一段数据的中间值插入树中，这样，每一个结点都是左右子树的中间值，这里可以使用折半查找的递归算法，计算每组的中间值，插入 BST 中，构建折半查找树。

源程序名称：1163450201.c

注意：正文文字为宋体小 4 号，图中文字为宋体 5 号。行距为多倍行距 1.25。
源程序与此报告打包提交，压缩包采用学号命名。