学生学号	0121710880223	实验课成绩	

武漢理工大學

学 生 实 验 报 告 书

实验课程名称	数据结构
开 课 学 院	计算机科学与技术学院
指导教师姓名	胡燕
学 生 姓 名	刘佳迎
学生专业班级	计算机类 m1702 班

2018 -- 2019 学年 第 一 学期

实验教学管理基本规范

实验是培养学生动手能力、分析解决问题能力的重要环节；实验报告是反映实验教学水平与质量的重要依据。为加强实验过程管理，改革实验成绩考核方法，改善实验教学效果，提高学生质量，特制定实验教学管理基本规范。

1、 本规范适用于理工科类专业实验课程，文、经、管、计算机类实验课程可根据具体情况参照执行或暂不执行。

2、 每门实验课程一般会包括许多实验项目，除非常简单的验证演示性实验项目可以不写实验报告外，其他实验项目均应按本格式完成实验报告。

3、 实验报告应由实验预习、实验过程、结果分析三大部分组成。每部分均在实验成绩中占一定比例。各部分成绩的观测点、考核目标、所占比例可参考附表执行。各专业也可以根据具体情况，调整考核内容和评分标准。

4、 学生必须在完成实验预习内容的前提下进行实验。教师要在实验过程中抽查学生预习情况，在学生离开实验室前，检查学生实验操作和记录情况，并在实验报告第二部分教师签字栏签名，以确保实验记录的真实性。

5、 教师应及时评阅学生的实验报告并给出各实验项目成绩，完整保存实验报告。在完成所有实验项目后，教师应按学生姓名将批改好的各实验项目实验报告装订成册，构成该实验课程总报告，按班级交课程承担单位（实验中心或实验室）保管存档。

6、 实验课程成绩按其类型采取百分制或优、良、中、及格和不及格五级评定。

附表：实验考核参考内容及标准

	观测点	考核目标	成绩组成
实验预习	1. 预习报告 2. 提问 3. 对于设计型实验，着重考查设计方案的科学性、可行性和创新性	对实验目的和基本原理的认识程度，对实验方案的设计能力	20%
实验过程	1. 是否按时参加实验 2. 对实验过程的熟悉程度 3. 对基本操作的规范程度 4. 对突发事件的应急处理能力 5. 实验原始记录的完整程度 6. 同学之间的团结协作精神	着重考查学生的实验态度、基本操作技能；严谨的治学态度、团结协作精神	30%
结果分析	1. 所分析结果是否用原始记录数据 2. 计算结果是否正确 3. 实验结果分析是否合理 4. 对于综合实验，各项内容之间是否有分析、比较与判断等	考查学生对实验数据处理和现象分析的能力；对专业知识的综合应用能力；事实求实的精神	50%

实验课程名称：_____数据结构_____

实验项目名称	实验四			实验成绩	
实 验 者	刘佳迎	专业班级	计算机类 m1702 班	组　　别	
同 组 者				实验日期	2018 年 12 月 31 日

一部分：实验预习报告（包括实验目的、意义，实验基本原理与方法，主要仪器设备及耗材，实验方案与技术路线等）

实验内容：

【问题描述】读入一串整数，构造一棵二叉排序树。然后在此树上查找值为 x 的结点。

【实现提示】二叉排序树的构成，可从空的二叉树开始，每输入一个结点，就将其插入到当前已形成的二叉排序树中，所以，它实际是利用了二叉排序树的插入算法。

【测试数据】

读入一串整数：10,8,23,5,67,98,34,20，构造 BST。

实验基本原理与方法及技术路线：

1.抽象数据类型

二叉树中任意结点的值大于左节点的值，小于右节点的值，满足 BST 树的性质，同时本题在建树时需要大量的插入操作，运用链式结构比较方便，所以用链式结构的二叉树来满足 BST 树的构建。

2.ADT

①. 二叉树的 ADT：

数据对象 D：D 是 BinNode 类的数据元素的集合

数据关系 R：

若 D 为空集，则称为空树 。

否则:

(1) 在 D 中存在唯一的称为根的数据元素 root；

(2) 当 n>1 时，其余结点可分为 m (m>0)个互不相交的有限集 T1, T2, …, Tm，其中每一棵子集本身又是一棵符合本定义的树，称为根 root 的子树。

基本操作：

```
 bool InitBST(BST *b) //初始化二叉树
bool InitBSTNode(BSTNode * &n)//初始化节点
bool clearBST(BSTNode * &n) //销毁 BST
```

②结点的 ADT

数据对象：包含结点的值，同时包含结点的左右指针

数据关系：每个结点都有各自的值

若结点左右指针为空，则该节点称为叶子结点

基本操作：

```
//结点的初始化
BinNodePtr(){lc=rc=NULL}
BinNodePtr(Elem e,BinNodePtr*  l=NULL;BinNodePtr*  r=NULL)
{it=e;lc=l;rc=r;}
//判断是否是叶子结点
bool isleaf(){return(lc==NULL)&&(rc==NULL)};
```

3.算法的基本思想

构建 BST 树：输入节点数后，依次输入每个结点的值，将第一个结点作为根结点，插入一个数，这个数与根节点先比较，若小于则再与根结点的左子树相比较，若大于则与根节点的右子树相比较。比较时，若小于根结点且根结点的左子树为空，则这个数为根结点左子树的值，若小于根结点又大于根结点的左子树，则运用指针将根结点的左指针指向这个值得地址，这个值地址的指针再指向原来根结点左子树；大于的情况同理。

查找：设置一个计数器，每查找一次则加一。从根节点开始，在 BST 中检索值 K。如果根节点存储的值为 K，则检索结束。如果不是，必须检索数的更深的一层。BST 的效率在于只需检索两棵子树之一。如果 K 小于根节点的值，则只需检索左子树；若果 K 结点大于根结点的值，则检索右子树。这个过程一直持续到 K 被知道或者遇到一个叶子结点为止。如果叶子结点仍没有发现 K，那么 K 就不在 BST 中。

4.程序的流程

程序由三个模块组成：

输入模块：输入结点数目初始数据，构建二叉查找树
查找模块：判断需要查找的值是否在该 BST 中
输出模块：输出查找成功与否，并输出比较的次数

主要仪器设备及耗材

Vs2017

第二部分：实验过程记录（可加页）（包括实验原始数据记录，实验现象记录，实验过程发现的问题等）

1.算法的实现

```
//初始化二叉树
 bool InitBST(BST *b) {
    b->root = NULL;
    return true;
}


//销毁 BST
bool clearBST(BSTNode * &n)
{
    if (n)
        return  false;
    if (n->lchild)
        clearBST(n->lchild);
    if (n->rchild)
        clearBST(n->rchild);
    free(n);
    return true;
}
//初始化节点
bool InitBSTNode(BSTNode * &n)
{
    n = (BSTNode *)malloc(sizeof(BSTNode));
    n->lchild = NULL;
    n->rchild = NULL;
    return true;
}
```

结点的 ADT 具体实现

```
BinNodePtr(){lc=rc=NULL}//结点的初始化
BinNodePtr(Elem e,BinNodePtr*  l=NULL;BinNodePtr*  r=NULL)
{it=e;lc=l;rc=r;}
//判断是否是叶子结点
bool isleaf()
```

2.算法的具体步骤

①. 结点输入操作：先输入结点数，在输入每个结点的值，通过循环调用 insert 函数将每个结点进行插入

```
cout << "BST 结点数目：" << endl;
    cin >> n;
for (int i = 0; i<n; i++)
    {
        cin >> p[i];
        if (p[i] <= 0)
        {
            cout << "无效的结点输入" << endl;
             system("pause");
            return;
        }
        insert(&b, p[i]);
    }
```

②. 头结点的建立：对于第一个调用 insert 函数插入的值，作为头结点建立二叉树

```
if (b->root == NULL)
    {
        b->root = n;
        return true;
```

```
    }
```

④. 之后结点的插入：插入一个数，这个数与根结点先比较，若小于则再与根结点的左子树相比较，若大于则与根节点的右子树相比较。比较时，若小于根结点且根结点的左子树为空，则这个数为根结点左子树的值，若小于根结点又大于根结点的左子树，则运用指针将根结点的左指针指向这个值得地址，这个值地址的指针再指向原来根结点左子树；大于的情况同理。

```
while (1)//循环比较
    {
        if (e<m->data)//小于根节点则插入左子树
        {
            if (m->lchild == NULL)
            {
                m->lchild = n;//给左孩子赋值
                return  true;
            }
            if (m->lchild != NULL&&e>m->lchild->data)
            {
                n->lchild = m->lchild;
                m->lchild = n;
                return true;
            }
            else  m = m->lchild;
            continue;
        }
        else//大于根节点则插入右子树
        {
            if (m->rchild == NULL)
            {
                m->rchild = n; //给右孩子赋值
                return  true;
            }
```

```
            if (m->rchild != NULL&&e<m->rchild->data)
            {
                n->rchild = m->rchild;
                m->rchild = n;
                return true;
            }
            else
                m = m->rchild;
            continue;
        }
    }
```

⑤. 完整的 insert 插入操作：插入元素 e 时，先判断该二叉树是否为空，若为空，将 e 作为该二叉树的根节点。否则，从根节点开始，比较 e 与节点 n 的大小。如果 e 的值更小，判断 n 的左子树是否为空，若为空，将 e 作为节点 n 的左孩子并返回 e，否则比较 e 与 n 左孩子的值，依此循环下去；如果 e 的值更大，判断 n 的右子树是否为空，若为空，将 e 作为节点 n 的右孩子并返回 e，否则比较 e 与 n 右孩子的值，依此循环下去。

```
bool insert(BST *b, ElemType e)//把结点插入 BST
{
    BSTNode *n, *m;
    InitBSTNode(n);
    n->data = e;
    if (b->root == NULL)
    {
        b->root = n;
        return true;
    }
    m = b->root;
    while (1)//循环比较
    {
        if (e<m->data)//小于根节点则插入左子树
```

```
        {
            if (m->lchild == NULL)
            {
                m->lchild = n;//给左孩子赋值
                return  true;
            }
            if (m->lchild != NULL&&e>m->lchild->data)
            {
                n->lchild = m->lchild;
                m->lchild = n;
                return true;
            }
            else  m = m->lchild;
            continue;
        }
        else//大于根节点则插入右子树
        {
            if (m->rchild == NULL)
            {
                m->rchild = n; //给右孩子赋值
                return  true;
            }
            if (m->rchild != NULL&&e<m->rchild->data)
            {
                n->rchild = m->rchild;
                m->rchild = n;
                return true;
            }
            else
```

```
            m = m->rchild;continue;}}}
```

②find 查找操作，查找元素时，从根节点开始，比较 e 与节点 x 的大小，若相等，返回 true;如果 e 比节点 x 的值小,判断 x 的左子树是否为空,若为空,返回 false，不为空则比较 e 与 x 左孩子的值，依次循环下去；如果 e 比节点 x 的值大，判断 x 的右子树是否为空，若为空，返回 false，不为空则比较 e 与 x 右孩子的值，依次循环下去。

```
bool find(BST *b, ElemType e) //查询元素 e,记录比较的次数查询成功返回 true，否则返回 false
{
    int count = 0;
    BSTNode *x = b->root;
    while (1)//循环比较
    {
        count++;//设置计数器
        if (e<x->data)//小于根节点则在左子树中查找
        {
            if (x->lchild == NULL)
            {
                cout << "查找失败,查找次数:  " << count << endl;
                return false;//左子树为空则查找失败
            }
            x = x->lchild;//继续与左孩子的值比较
            continue;
        }
        if (e>x->data) //大于根节点则在右子树中查找
        {
            if (x->rchild == NULL)
            {
                cout << "查找失败,查找次数:  " << count << endl;
                return false;//右子树为空则查找失败
            }
```

```
            x = x->rchild;//继续与右孩子的值比较
            continue;
        }
        if (e == x->data)
        {
            cout << "查找成功,查找次数:  " << count << endl;
            //cout << count;
            return  true;
        }}}
```

3.算法的时空分析

查找元素需要的比较次数由树的深度决定查找，最好时间复杂度O(logN)，最坏时间复杂度O(N)(只有左子树或右子树的情况)。

4.输入和输出的格式

输入BST结点数

```
BST结点数目:      //等待输入
          cout << "BST结点数:" << endl;
    cin >> n;
```

输入BST结点数据

```
BST节点数据:      //等待输入
for (int i = 0; i<n; i++)
    {
        cin >> p[i];
        if (p[i] <= 0)
        {
            cout << "无效的结点输入" << endl;
             system("pause");
            return;
        }
```

输入要查找的数据

```
cout << "输入要查找的数据（输入-1 结束查找）" << endl;
        cin >> m;
```

若 BST 结点数目输入错误：

输出 “无效的结点数目输入”

```
if (n <= 0)
    {
        cout << "无效的结点数目输入" << endl;
        system("pause");
        return;

    }
```

若 BST 节点数据输入错误：

输出 “无效的结点输入”

```
if (p[i] <= 0)
        {
            cout << "无效的结点输入" << endl;
             system("pause");
            return;
        }
```

若查找成功

输出 查找次数：//输出次数

```
if (e == x->data)
        {
            cout << "查找成功,查找次数：  " << count << endl;
            return  true;
```

```
    }

查找失败，查找次数：//输出次数
  if (x->rchild == NULL)
        {
            cout << "查找失败,查找次数：  " << count << endl;
            return false;//右子树为空则查找失败
        }
```

教师签字_____

第三部分　结果与讨论（可加页）

一、**实验结果分析**（包括数据处理、实验现象分析、影响因素讨论、综合分析和结论等）

```
BST节点数目:
8
BST节点数据:
10 8 23 5 67 98 34 20
输入要查找的数据（输入-1结束查找）
10
查找成功,查找次数:    1
输入要查找的数据（输入-1结束查找）
8
查找成功,查找次数:    2
输入要查找的数据（输入-1结束查找）
23
查找成功,查找次数:    3
输入要查找的数据（输入-1结束查找）
67
查找成功,查找次数:    5
输入要查找的数据（输入-1结束查找）
5
查找成功,查找次数:    3
输入要查找的数据（输入-1结束查找）
98
查找成功,查找次数:    6
输入要查找的数据（输入-1结束查找）
34
查找成功,查找次数:    4
输入要查找的数据（输入-1结束查找）
20
查找成功,查找次数:    2
输入要查找的数据（输入-1结束查找）
-1
```

1. 本程序会将第一个输入的值作为 root，如果第一个值输入过大，导致所有数据都被存放进左子树，导致树的长度 n 过长，接下来的查找效率过低，因为最好在输入前就大致考虑一下中间值是多少，尽量避免树过长。

2. 一开始 insert 函数并没有考虑太多，后来发现再输入节点的值时有限制，必须输入比之前输入所有节点值的最小值还小，或者比之前输入最大值还要大。

二、**小结、建议及体会**

1. 由于对时间复杂度的不满意，查阅了有关最优二叉查找树资料

2. 动态规划方法生成最优二叉查找树

3. 基于统计先验知识，我们可统计出一个数表（集合）中各元素的查找概率，理解为集合各元素的出现频率。比如中文输入法字库中各词条（单字、词组等）的先验概率，针对用户习惯可以自动调整词频——所谓动态调频、高频先现原则，以减少用户翻查次数。这就是最优二叉查找树问题：查找过程中键值比较次数最少，或者说希望用最少的键值比较次数找到每个关键码（键值）。为解决这样的问题，显然需要对集合的每个元素赋予一个特殊属性——查找概率。这样我们就需要构造一颗最优二叉查找树。

n 个键{a1,a2,a3.....an},其相应的查找概率为{p1,p2,p3.....pn}。构成最优 BST,表示为 T1n ，求这棵树的平均查找次数 C[1, n]（耗费最低)。换言之，如何构造这棵最优 BST，使得 C[1, n] 最小。

动态规划法策略是将问题分成多个阶段，逐段推进计算，后继实例解由其直接前趋实例解计算得到。对于最优 BST 问题，利用减一技术和最优性原则，如果前 n-1 个节点构成最优 BST，加入一个节点 an 后要求构成规模 n 的最优 BST。按 n-1, n-2 , ... , 2, 1 递归，问题可解。自底向上计算：C[1, 2] →C[1, 3] →... →C[1, n]。为不失一般性用

C[i, j] 表示由{a1,a2,a3.....an}构成的 BST 的耗费。其中 1≤i ≤j ≤n。这棵树表示为 Tij。从中选择一个键 ak 作根节点，它的左子树为 Tik-1，右子树为 Tk+1j。要求选择的 k 使得整棵树的平均查找次数 C[i, j]最小。左右子树递归执行此过程。(根的生成过程）