



**数据结构实验报告**

题目名称 抽象数据类型的实现

学生学院 计算机

专业班级 软件工程17(4)班

学 号 3117004958

学生姓名 章世楠

指导教师 曾孜

2018 年 12 月 30 日

目录

[1. 简介 1](#_Toc32559)

[1.1. 设计任务 1](#_Toc19843)

[1.2. 设计要求 1](#_Toc21421)

[1.3. 软件环境以及工具 1](#_Toc24176)

[2. 总体设计 2](#_Toc21583)

[2.1. 抽象数据类型定义 2](#_Toc9835)

[2.2. 基本操作的简要描述 3](#_Toc29278)

[3. 详细设计 4](#_Toc28514)

[3.1. 创建B树 4](#_Toc7854)

[3.2. 插入节点 5](#_Toc754)

[4. 测试分析 6](#_Toc4122)

[5. 思考与分析 6](#_Toc18808)

[6. 总结与体会 6](#_Toc3792)

# 简介

## 设计任务

以教材中讨论的各种抽象数据类型为对象，利用C语言的数据类型表示和实现其中某个抽象数据类型。此次选择实现B树抽象数据类型作为我的设计任务。

## 设计要求

基本要求：实现创建B树、B树的增加节点、删除节点、查找节点、中序遍历B树、后序遍历B树、前序遍历B树、查找B树的节点个数、摧毁B树9种基本操作。

扩展要求：拥有可视化的界面、良好的交互、程序的鲁棒性。

|  |  |
| --- | --- |
| 要求 | 实现函数 |
| 创建B树 | void CreateBTreeByInput(BTree &T) |
| 增加节点到B树 | void InsertBTreeByInput(BTree &T) |
| 删除B树的节点 | void DeleteBTreeNode(BTree &BT) |
| 查找节点 | void FineKeyInBTree(BTree t) |
| 中序遍历B树 | void TravelBTreeInorder(Btree BT) |
| 后序遍历B树 | void TravelBTreePosterorder(BTree t) |
| 前序遍历B树 | void ShowBTree(BTree t,int x) |
| 查找B树的节点个数 | void CountBTreeNodeNum(BTree t,int& num) |
| 销毁B树 | void DestroyBTree(BTree T) |
| 可视化界面(输出菜单界面) | int menu() |

## 软件环境以及工具

此次在visio studio 2017的集成开发软件中生成解决方案以及在windows下运行的exe文件。visio studio 2017是一款强大的、高效的c\c++编译器，所以此次采用其作为我的开发工具。

# 总体设计

## 抽象数据类型定义

1. B树的节点的数据定义：

typedef struct BTNode {

int keynum; //节点当前的关键字个数

KeyType key[M + 1]; //关键字数组 ，key[0]未用

struct BTNode \* parent; //双亲节点指针

struct BTNode \* ptr[M+ 1]; //孩子节点指针数组

Record \* recptr[M + 1]; //记录指针向量，0号单元未用

}BTNode,\* BTree;

B树由如上的节点所构成，拥有关键字数组、孩子节点数组、关键字个数、双亲节点指以及记录指针向量。

B树节点满足以下条件：

（1）树中每个结点最多含有m棵子树；

（2）若根结点不是叶子结点，则至少有2个子树；

（3）除根结点之外的所有非终端结点至少有┌m/2┐棵子树；

（4）每个非终端结点中包含信息：（n，A0，K1，A1，K2，A2，…，Kn，An）。其中：

1）Ki（1<=i<=n）为关键字，且关键字按升序排序；

2）指针Ai（0<=i<=n）指向子树的根结点，Ai-1指向子树中所有结点的关键字均小于Ki，且大于Ki-1；

3）关键字的个数n必须满足：┌m/2┐-1<=n<=m-1。 （5）所有的叶子节点都在同一层，子叶结点不包含任何信息。

（5）所有的叶子节点都在同一层，子叶结点不包含任何信息。

1. 状态的定义

//定义状态

typedef enum {

ERROR, OK, OVERFLOW

}Status;

这是一个枚举数据类型，用来描述失败(ERROR)、成功(OK)、溢出（OVERFLOW）三种状态。

1. 查找结果结构体的定义

typedef struct {

BTree pt;

int i;

int tag;

}result;

该结构体用来描述其查找的结果，tag标识是否查找成功，如若是，取值为1，i表示该关键字位于pt关键字数据的第i个关键字。

## 基本操作的简要描述

Status CreateBTree(BTree &T, int n, int rootNum);

功能：创建B树，其中根节点的值为rootNum,在随机生成n个关键字，插入到B树中

参数: B树的根节点，关键字数n，根节点的植rootNum

返回值：返回状态(成功、失败、溢出)

void InsertBTree(BTree &t, int k, BTree q, int i)

功能：插入关键字 m 以及孩子指针 q 到B树的t节点中，关键字m插入到t的关键 字数组key[i]中，孩子指针q插入到孩子指针数组ptrp[i]中

void spilt(BTree &q, int s, BTree &ap)

功能：分裂B树，把q根据s为中点，分裂成俩部分，前半部分放在q，后半部分放在 ap,当q需要的关键字个数大于等于m时，需要进行分裂操作。

void newRoot(BTree &t,BTree p,int x,BTree ap)

功能：新建一个节点，其值为x，孩子数组ptr[0]为p,ptr[1]为ap，当在进行插入操作， 关键字太多，导致分裂不断向上进行时，如果父节点需要分裂，那么就需要新建一个根 节点，把原来父节点以s分裂出去的俩部分作为新根节点的俩个孩子。

int Search(BTree p, int k)

功能：在p节点的关键字数组中查找大于等于关键字k的下标，如果找不到，返回该关 键字数组的最大个数，通过该函数，可以更加方便地找到关键字k

返回值：关键字的下标

void SearchBTree(BTree T, int k, result &r)

功能：在m阶B树T上查找关键字k，返回(pt,i,tag) 若查找成功，则特征值tag=1, 指针pt所致结点中第i个关键字等于k;否则 特征值tag=0，等于k的关键字记录应插 入在指针pt所指结点中第i-1个和第i个关键字间。

void Successor(BTree &p,int i)

功能：在p的Ai子树下找到最小的 最下层非终端节点，在进行非底层节点删除时，需 要调用该函数去做替换，比如要删除54，54不在最底层，就找到54的Ai子树的最底 层的节点，例如 60，把54替换成60，相当于删除了54，然后再删除60，就可以直 接调用删除底层非终端节点的操作了。

void Remove(BTree &p,int i)

功能：删除p的第i个关键字，其操作很简单，就是把i之后的元素都往前挪，然后p 节点数-1

void Restore(BTree &p,int i)

功能:调整B树，这是整个B树设计的一个难点，其功能就是调整B树，使得B树在完 成删除操作时，还是能够维持之前B树的形状，每个节点都能满足原来的要求，即关 键字的个数n必须满足：┌m/2┐-1<=n<=m-1。

void DeleteBTree(BTree &p,int i)

功能：删除B树p节点的第i个关键字

void ShowBTree(BTree t,int x)

功能：先序遍历B树，并把结果用缩进的效果打印出来，即凹凸表

void TravelBTreeInorder(BTree t)

功能：中序遍历B树

void TravelBTreePosterorder(BTree t)

功能：后序遍历B树

int CountBTreeKeyNum(BTree t)

功能：计算节点个数

void DestroyBTree(BTree T)

功能：摧毁B树

# 详细设计

## 创建B树

B树的创建流程为：先读取输入，获取创建的前提条件后，再进行创建。

先让用户输入，B树阶数，再让用户选择初始化要产生多少个节点(n)，再让用户设置根节点的值。读取输入完成后，根据用户的输入，设置该B树的阶数，设置根节点的值，以及产生n个随机数取构造节点，这n个随机数通过调用SearchBTree（）函数判断是否已经存在，如若不存在，则调用InsertBTree(BTree &t, int k, BTree q, int i)插入函数，有序地插入到B树中。

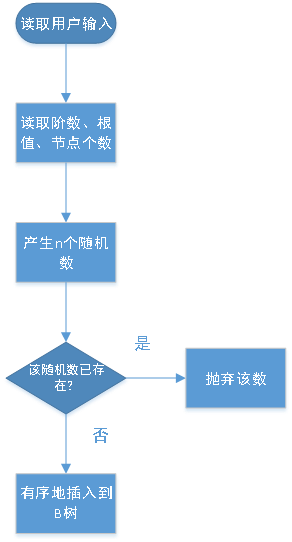


图 1 创建B树流程图

## 插入节点

节点的插入首先需要调用SearchBTree（）函数进行查找，查找的目的有俩个，判断该数是否已经存在以及找到合适的插入位置。如果该数已经存在，无法插入。如果不存在，则需要找到合适的插入位置，进行插入。

插入完成后，需要判断其节点是否满足B树节点的要求，即需要检测其关键字数是否大于m-1。如若关键字大于m-1，需要进行分裂操作。

分裂操作：把原来一个节点(p)，分成俩部分，先取s为该节点的中点，然后新建一个节点(q)，p的前半部分，即s之间的关键字数组以及孩子指针都留在p，把p的后半部分，包括孩子指针赋值给q，然后下标为s的关键字赋值给到父节点。即把一个节点p分成3部分，前半部分仍留在p,后半部分赋值给q，然后中间的关键字有序地插入到父节点。而p,q则作为父节点相邻的左、右孩子指针。

分裂操作完成后需要判断其父节点是否满足B树节点的要求，如果其关键字书大于m-1，则需要再进行分类，就这样一直向上分裂，直到根节点，如果根节点也需要分裂，那么需要重新再建一个根节点。

新建根节点跟分裂操作类似：把根节点(p)以s（m/2)作为中点分成俩部分，前半部分留在p,后半部分新建节点q,然后中间值赋值为新根节点(root)的值。P、q分别作为左右孩子加入到新根节点root中。

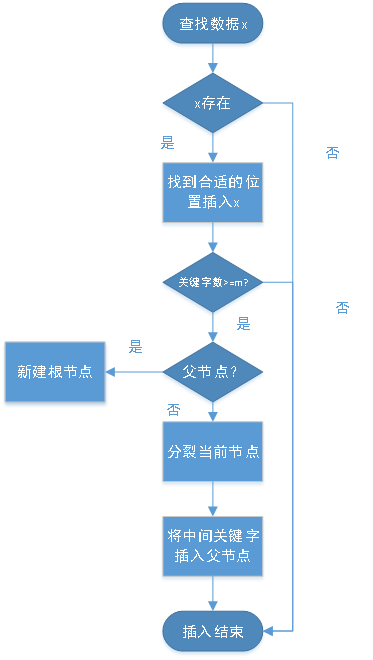


图 2 插入操作

## 删除关键字

删除关键字有3种情况，7个步骤。

情况S1:被删除的关键字所位于的节点是最底层的非终端节点，且删除后该节点的个数仍满足n>=⌈m/2⌉.

情况S2：被删除的关键字不是位于最底层的非终端节点

情况S3：被删除的关键字所位于的节点是最底层的非终端节点，且删除后该节点的个数不满足n>=⌈m/2⌉.

步骤：

1. 判断删除的关键字(key)所位于的节点(p)是否位于最底层的非终端节点,如果是，则进行删除操作，即直接用后面的关键字覆盖其值，并且关键字数减一。
2. 判断删除后的节点的关键字数是否满足n>=⌈m/2⌉，如若满足，则结束该步骤。
3. 如果不满足(2)的条件，即是情况S3，则需要进行调整。

# 测试分析

# 思考与分析

# 总结与体会