## 



Harbin Institute of Technology

实验报告

课程名称： 数据结构与算法

实验题目： BST和折半查找的分析与比较

实验类型： 必 修

所在院系： 计算机科学与技术学院

学生类别： 本科生

指导教师： 李秀坤

学生姓名： 易亚玲

班级和学号： 1703005 1170300511

实验时间： 2018 年 秋 季学期

哈尔滨工业大学

1. **实验目的**

实现BST查找和折半查找，比较BST查找和折半查找的性能

1. **实验内容和要求**

**2.1.实验内容**

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。

**2.2实验要求**

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，考察比较两种查找算法的时间性能，并与理论结果比较。以下具体做法可作参考：

（1）第一组测试数据：n=1024个已经排序的整数序列（如0至2048之间的奇数）；第2组测试数据：第一组测试数据的随机排序。

（2）按上诉两种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树的查找成功和查找失败的平均长度（主要是改造search算法，对“比较”进行计数），并与理论结果作比较。

（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，并与理论值作比较。

（5）以上数据能否说明：就平均性能而言，BST查找与折半查找差不多，为什么？

**（三）程序实现**

**3.1变量及函数**

#define MAX\_SIZE 1024 //最大容量，可修改

typedef struct yi{

int num;

struct yi \*lc,\*rc;

}\*node,N; //BST树的节点

int flag = 0; //在BST删除一个树中没有的元素时，作为标志打印错误信息

int cnt; //计数

int a[MAX\_SIZE]; //保存测试数据

void printmenu(); //打印菜单

void insert\_BST(node &F,int data); //插入

void delete\_BST(node &F,int data); //删除

int deletemin(node &F,int data); //寻找右儿子的最左节点，并删除改节点

void creat\_BST(node &F); //建立BST查找树

void halfsearch(int data); //折半查找

void writetest(); //生成测试数据

void getsort(node F); //得到BST树的中序遍历序列，保存在数组a[]中

void search\_BST(node F,int data); //在BST查找树中查找元素data

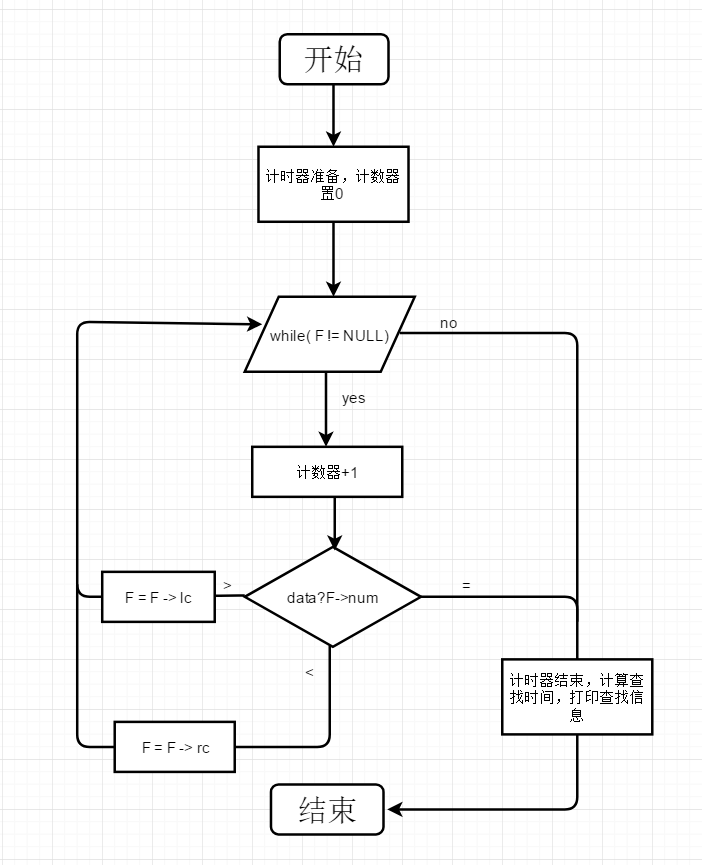
void countBST(node F); //计算BST的查找成功与失败的平均次数

void counthalf(); //计算BST查找成功和失败的平均次数

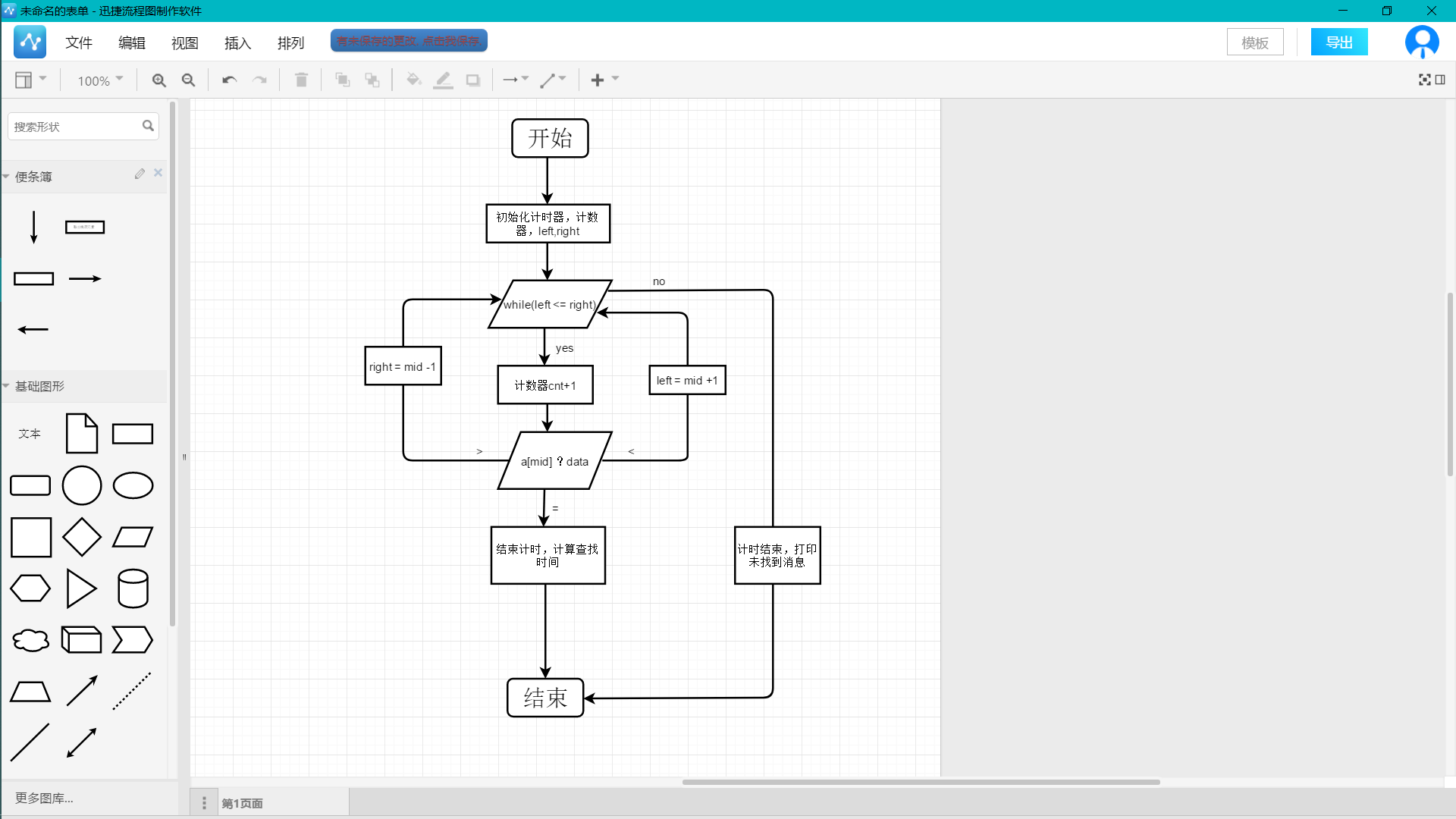
**3.2算法设计**

关键函数的思维导图：

1. BST查找

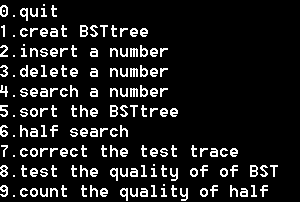


1. 折半查找



**（四）系统测试**

**菜单**

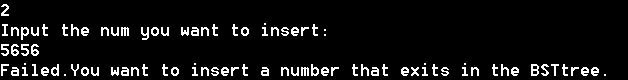


**提示：若需要使用BST，需要先初始化测试数据，即选择功能7，再输入1选择顺序，输入2，选择乱序，然后选择功能1建立BST树以后才能正常使用；使用折半查找，需选择功能7输入1或者选择功能7输入2再选择功能5初始化数组。**

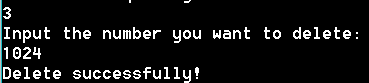
1.向BST插入任意一个数



再插入这个数时，会显示已经存在，因此插入成功



2从BST中删除一个数

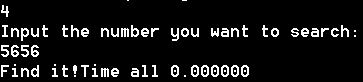


再次删除这个数时，会显示这个数不在BST中，证明之前删除成功

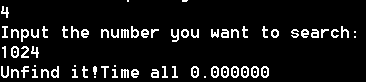


1. 在BST中查找一个数

查找刚刚插入的5656，成功

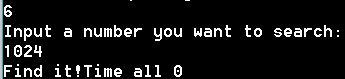


查找刚刚删除的1024，失败

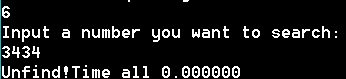


1. 折半查找

查找一个数组中有的数



查找一个数组中没有的数



1. BST在顺序情况下查找成功和失败的平均次数测试



1. BST在乱序情况下查找成功和失败的平均次数测试



1. 折半查找查找成功和失败的平均次数



**（五）实验反思**

从建立BST树的过程中，我意识到我对之前链表的理解还不够深入，或者是有些要点已经有的忘记了，所以在写实验的过程中遇到了很多平时容易忽视的小错误。平时应该多多回顾以前遇到的问题，科学记忆并结合适当的练习加以巩固。折半查找的实现，更让我感受到了二分的魅力，二分是大自然的智慧，以后再遇到问题的时候一定要想到用二分去优化问题，肯定会有意想不到的效果。

**（六）源代码**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_SIZE 1024 //最大容量，可修改

typedef struct yi{

int num;

struct yi \*lc,\*rc;

}\*node,N; //BST树的节点

int flag = 0; //在BST删除一个树中没有的元素时，作为标志打印错误信息

int cnt; //计数

int a[MAX\_SIZE]; //保存测试数据

void printmenu(); //打印菜单

void insert\_BST(node &F,int data); //插入

void delete\_BST(node &F,int data); //删除

int deletemin(node &F,int data); //寻找右儿子的最左节点，并删除改节点

void creat\_BST(node &F); //建立BST查找树

void halfsearch(int data); //折半查找

void writetest(); //生成测试数据

void getsort(node F); //得到BST树的中序遍历序列，保存在数组a[]中

void search\_BST(node F,int data); //在BST查找树中查找元素data

void countBST(node F); //计算BST的查找成功与失败的平均次数

void counthalf(); //计算BST查找成功和失败的平均次数

int main()

{

node F = NULL; //BST树

int sel,data;

while(1){

printmenu();

scanf("%d",&sel);

switch(sel){

case 0:exit(0);

case 1:creat\_BST(F);

break;

case 2:printf("Input the num you want to insert:\n");

scanf("%d",&data);

insert\_BST(F,data);

break;

case 3:printf("Input the number you want to delete:\n");

scanf("%d",&data);

flag = 0;

delete\_BST(F,data);

if(!flag) printf("Cant't find this number in the BSTtree\n");

break;

case 4:printf("Input the number you want to search:\n");

scanf("%d",&data);

search\_BST(F,data);

break;

case 5:cnt = 0;

getsort(F);

break;

case 6:printf("Input a number you want to search:\n");

scanf("%d",&data);

halfsearch(data);

break;

case 7:writetest();

break;

case 8:countBST(F);

break;

case 9:counthalf();

break;

default:

printf("Input error!\n");

}

system("pause");

}

return 0;

}

void printmenu(){

printf("0.quit\n");

printf("1.creat BSTtree\n");

printf("2.insert a number\n");

printf("3.delete a number\n");

printf("4.search a number\n");

printf("5.sort the BSTtree\n");

printf("6.half search\n");

printf("7.correct the test trace\n");

printf("8.test the quality of of BST\n");

printf("9.count the quality of half\n");

}

void insert\_BST(node &F,int data){

node p;

if(F == NULL) //将data插入到叶的位置

{

p = (node)malloc(sizeof(N));

p -> num = data;

p -> rc = NULL;

p -> lc = NULL;

F = p;

}

else if(F -> num == data){ //如果BST树中已经存在data

printf("Failed.You want to insert a number that exits in the BSTtree.\n");

}

else if(F -> num > data) //data小于根节点的值，插入左边

insert\_BST(F -> lc,data);

else //data大于根节点的值，插入右边

insert\_BST(F -> rc,data);

}

void delete\_BST(node &F,int data){

if(F == NULL) //如果BST为空，直接返回

return;

if(data > F -> num ) //如果data的值大于根节点的值，删右边

delete\_BST(F -> rc ,data);

else if(data < F -> num) //如果data的值小于根节点的值，删左边

delete\_BST(F -> lc,data);

else{ //找到要删除的节点

flag = 1; //BST查找树中有要删的数据

printf("Delete successfully!\n");

//寻找继承节点

if(F -> lc == NULL) //左儿子为空，右儿子继承

F = F -> rc;

else if(F -> rc == NULL) //右儿子为空，左儿子继承

F = F -> lc;

else

F -> num = deletemin(F -> rc,data); //左右儿子都不空，寻找右儿子的最左节点继承

}

}

int deletemin(node &F,int data){

int tmp;

node p;

if(F -> lc == NULL){ //找到最左节点

tmp = F -> num; //保存返回数据

p = F;

F = F -> rc; //该节点左儿子为空，直接右儿子继承

free(p); //删除该节点

return tmp;

}

else

return deletemin(F -> lc,data); //左儿子不为空，继续找左儿子的左儿子

}

void creat\_BST(node &F){

int i;

for(i = 0;i < MAX\_SIZE;i++)

insert\_BST(F,a[i]); //将测试数据一个一个依次插入BST树中

}

void search\_BST(node F,int data){

clock\_t start,finish;//用于保存时刻

double time; //时间段

cnt = 0; //计数器置0

start = clock(); //计时器开始

while(F != NULL){

cnt ++; //比较次数+1

if(F -> num == data){ //找到

finish = clock(); //计时器结束

time = (double)(finish - start); //计算查找时间

printf("Find it!Time all %lf\n",time / CLOCKS\_PER\_SEC); //打印查找成功信息

return;

}

else if(F -> num > data) //data小于根节点的值，根节点更新为左儿子，继续查找

F = F -> lc;

else //data大于根节点的值，找右儿子

F = F -> rc;

}

finish = clock(); //没有找到，计时结束

time = (double)(finish - start); //求不成功时间

printf("Unfind it!Time all %lf\n",time/CLOCKS\_PER\_SEC); //打印失败信息

return ;

}

void getsort(node F){

if(F != NULL){

if(F -> lc) //左不空，遍历左

getsort(F -> lc);

a[cnt++] = F -> num;//访问根节点

if(F -> rc) //右不空，遍历右

getsort(F -> rc);

}

}

void halfsearch(int data){

int left,right,mid;

double time;

clock\_t start,finish; //记录时刻

start = clock(); //计时开始

cnt = 0; //计数器置0

left = 0;right = MAX\_SIZE - 1; //初始化left,right

while(left <= right){

mid = (left+right)>>1; //参考位置

cnt ++; //比较次数+1

if(a[mid] == data){ //找到

finish = clock();//计时结束

time = (double)(finish - start); //计算查找时间

printf("Find it!Time all %.lf\n",time); //打印查找成功的消息

return;

}

//未找到

else if(a[mid] > data)//比中间的小，上左边找

right = mid - 1;

else //比中间的大，上右边找

left = mid + 1;

}

//查找失败

finish = clock(); //计时结束

time = (double)(finish - start); //计算查找时间

printf("Unfind!Time all %lf\n",time/CLOCKS\_PER\_SEC); //打印失败时间

}

void writetest(){

int sel,i,j,t,m;

printf("Input number 2 or 1 to choose a test data example.\n");

scanf("%d",&sel);

if(sel == 2||sel == 1){ //选择一个测试序列，其中1为顺序，2为乱序

for(i = 0;i < MAX\_SIZE;i++)

a[i] = (i+1)<<1;

if(sel == 2)

for(i = 0;i < (MAX\_SIZE >> 7);i++){ //调整次数控制

srand((int)time(NULL));

for(j = 1;j < MAX\_SIZE - 3;j++){

m = rand();

if(m%3){ //根据随机数是否被3整除来决定是否需要交换

t = a[j];

a[j] = a[MAX\_SIZE - (m%MAX\_SIZE)];

a[MAX\_SIZE - (m%MAX\_SIZE)] = t;

}

}

}

}

else

printf("You input a invalid number!\n");

}

void countBST(node F){

long sum = 0;

int i;

//查找成功平均次数计算

for(i = 0;i < MAX\_SIZE;i++){

search\_BST(F,a[i]); //查一个计数一个

sum += cnt;

}

sum = sum / MAX\_SIZE;

printf("The BST search successfully for %ld times\n",sum);

//查找失败平均次数计算

sum = 0;

for(i = 0;i < MAX\_SIZE;i++){

search\_BST(F,a[i]-1);

sum += cnt;

}

sum = sum / MAX\_SIZE;

printf("The BST search unsuccessfully for %ld times\n",sum);

}

void counthalf(){

long sum = 0;

int i;

//查找成功平均次数计算

for(i = 0;i < MAX\_SIZE;i++){

halfsearch(a[i]);

sum += cnt;

}

sum = sum / MAX\_SIZE;

printf("The half search successfully for %ld times\n",sum);

//查找失败平均次数计算

sum = 0;

for(i = 0;i < MAX\_SIZE;i++){

halfsearch(a[i]+1);

sum += cnt;

}

sum = sum / MAX\_SIZE;

printf("The half search unsuccessfully for %ld times\n",sum);

}

**参考文献**

[1]廖明宏，李秀坤，张岩，等.数据结构与算法。[M].北京：高等教育出版社，2007年