目 录

[实验一 顺序表操作实现及应用 1](#_Toc115027516)

[实验二 栈操作实现及应用 5](#_Toc115027517)

[实验三 队列操作实现及应用 9](#_Toc115027518)

[实验四 串的基本操作及模式匹配算法 13](#_Toc115027519)

[实验五 二叉树的建立、遍历及应用 17](#_Toc115027520)

[实验六 基于哈夫曼树的数据压缩算法 20](#_Toc115027521)

[实验七 图的建立和遍历 25](#_Toc115027522)

[实验八 查找算法实现与应用 29](#_Toc115027523)

# 实验一 顺序表操作实现及应用

**学时： 2 实验类型：设计性实验**

## 一、实验目的

1.掌握顺序表的各种基本运算算法实现，包括查找、插入和删除等算法。

2.掌握顺序表的应用算法设计。

3.掌握Java中顺序表容器ArrayList的使用。

## 二、实验内容

编写一个简单学生成绩管理程序，每个学生记录包含学号、姓名、课程和分数成员，采用顺序表存储，完成以下功能：

（1）屏幕显示所有学生记录。

（2）输入一个学生记录。

（3）按学号和课程删除一个学生记录。

## 三、实验算法实现

（1）设计学生记录类Stud，包含学生学号、姓名、课程和分数成员，以及构造方法、输出一个学生记录的方法和用于排序的get方法。

class Stud //学生记录类

{

int no; //学号

String name; //姓名

String course; //课程

int fraction; //分数

public Stud(int no1,String name1,String course1,int fraction1) //构造方法

{

no=no1;

name=name1;

course=course1;

fraction=fraction1;

}

public int getno()

{

return no;

}

public String getcourse()

{

return course;

}

public int getfraction()

{

return fraction;

}

public void disp()

{

System.out.println(" 学号:"+no+" 姓名:"+name+" 课程:"+course+" 分数:"+fraction);

}

}

（2）再设计包括相关基本运算算法的StudList类，其中采用ArrayList类对象sl作为学生顺序表，存放输入的所有学生记录。学生记录的插入、删除和排序等直接利用ArrayList类的方法实现。

class StudList

{

ArrayList<Stud> sl; //顺序表

public StudList() //构造方法

{

sl=new ArrayList<Stud>();

}

public void Addstud() //输入一个学生记录

{

Scanner in=new Scanner(System.in); //使用Scanner类定义对象

System.out.println(" 输入一个学生记录");

System.out.print(" 学号: ");

int no1=in.nextInt();

System.out.print(" 姓名: ");

String name1=in.next();

System.out.print(" 课程: ");

String course1=in.next();

System.out.print(" 分数: ");

int fraction1=in.nextInt();

sl.add(new Stud(no1,name1,course1,fraction1));

}

public void Dispstud() //输出所有学生记录

{

if (sl.size()>0)

{

System.out.println(" \*\*所有学生记录");

for (int i=0;i<sl.size();i++)

sl.get(i).disp();

}

else

System.out.println(" \*\*没有任何学生记录");

}

public void Delstud() //删除指定学号的学生记录

{

Scanner in=new Scanner(System.in); //使用Scanner类定义对象

System.out.print(" 删除的学号: ");

int no1=in.nextInt();

System.out.print(" 课程: ");

String course1=in.next();

int i=0;

boolean find=false;

while (i<sl.size())

{

Stud s=sl.get(i);

if (s.no==no1 && s.course.equals(course1))

{

find=true;

break;

}

i++;

}

if (find)

{

sl.remove(i);

System.out.println(" \*\*成功删除该学号学生的成绩记录");

}

else

System.out.println(" \*\*没有找到该学生的成绩记录");

}

public void Sort1() //按学号递增排序

{

sl.sort(Comparator.comparing(Stud::getno));

Dispstud();

}

public void Sort2() //按课程、分数递减排序

{

sl.sort(Comparator.comparing(Stud::getcourse).

thenComparing(Stud::getfraction).reversed());

Dispstud();

}

}

（3）最后设计包含主方法的Exp1类如下：

public class Exp1

{

public static void main(String[] args) throws IOException,NotSerializableException

{

int sel;

StudList L=new StudList();

Scanner in=new Scanner(System.in); //使用Scanner类定义对象

while(true){

System.out.print("1.显示全部记录 2.输入 3.删除 4.学号排序 5.课程排序 0:退出");

System.out.print(" 请选择:");

sel=in.nextInt();

switch(sel)

{

case 0:break;

case 1:L.Dispstud();break;

case 2:L.Addstud();break;

case 3:L.Delstud();break;

case 4:L.Sort1();break;

case 5:L.Sort2();break;

}

if (sel==0) break;

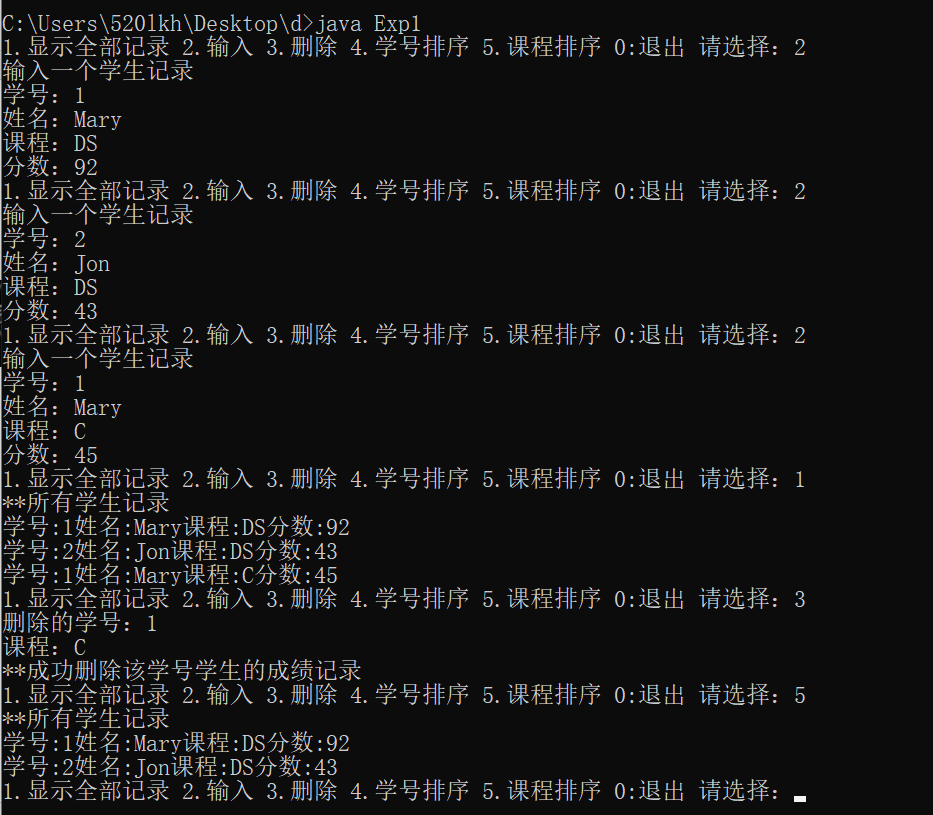
}

}

}

## 四、实验结果

简单学生成绩管理程序



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验二 栈操作实现及应用

**学时： 2 实验类型：验证性实验**

## 一、实验目的

1.掌握顺序栈存储结构和顺序栈中各种基本运算算法设计。

2.掌握顺序栈的应用算法设计。

## 二、实验内容

用一个一维数组S(设固定容量MaxSize为5，元素类型为int)作为两个栈的共享空间。编写一个程序，采用例3.7的共享栈方法设计其判栈空运算empty(i)、栈满运算full()、进栈运算push(i，x)和出栈运算pop(i)，其中i为1或2，用于表示栈号，x为进栈元素。采用相关数据进行测试。

## 三、实验算法实现

利用教材第3章例3.7的原理设计程序如下：

class BSTACK //共享栈类

{

final int MaxSize=5;

int[] S; //存放共享栈中的元素

int top1,top2; //两个栈顶指针

public BSTACK() //构造方法，栈初始化

{

S=new int[MaxSize];

top1=-1;

top2=MaxSize;

}

//-----判栈空算法------

public boolean empty(int i) //i=1：栈1，i=2：栈2

{ if (i==1)

return(top1==-1);

else if (i==2) //i=2

return(top2==MaxSize);

else

throw new IllegalArgumentException("i错误");

}

//-----判栈满算法------

public boolean full()

{

return top1==top2-1;

}

//-----进栈算法------

public void push(int i,int x) //i=1：栈1，i=2：栈2

{ if (full()) //栈满

throw new IllegalArgumentException("栈满");

if (i==1) //x进栈S1

{

top1++;

S[top1]=x;

}

else if (i==2) //x进栈S2

{ top2--;

S[top2]=x;

}

else //参数i错误返回0

throw new IllegalArgumentException("i错误");

}

//-----出栈算法------

public int pop(int i) //i=1：栈1，i=2：栈2

{ int x;

if (i==1) //S1出栈

{ if (empty(1)) //S1栈空

throw new IllegalArgumentException("栈1空");

else //出栈S1的元素

{ x=S[top1];

top1--;

}

}

else if (i==2) //S2出栈

{ if (empty(2)) //S2栈空

throw new IllegalArgumentException("栈2空");

else //出栈S2的元素

{ x=S[top2];

top2++;

}

}

else //参数i错误返回0

throw new IllegalArgumentException("i错误");

return x; //操作成功返回1

}

}

public class Exp2

{

public static void main(String[] args)

{

BSTACK st=new BSTACK();

System.out.println("");

System.out.println(" (1)新建立栈st");

System.out.println(" 栈1空? "+st.empty(1));

System.out.println(" 栈2空? "+st.empty(2));

int [] a={1,2,3};

int [] b={4,5,6,7};

System.out.println(" (2)栈1的进栈操作");

for (int i=0;i<a.length;i++)

{

if (!st.full())

{ System.out.println(" "+a[i]+"进栈1");

st.push(1,a[i]);

}

else

System.out.println(" "+a[i]+"进栈:栈满不能进栈");

}

System.out.println(" 栈1空? "+st.empty(1));

System.out.println(" (3)栈2的进栈操作");

for (int i=0;i<b.length;i++)

{

if (!st.full())

{ System.out.println(" "+b[i]+"进栈2");

st.push(2,b[i]);

}

else

System.out.println(" "+b[i]+"进栈:栈满不能进栈");

}

System.out.println(" 栈2空? "+st.empty(2));

System.out.println(" (4)栈1的出栈操作");

while (!st.empty(1))

System.out.println(" 出栈1元素"+st.pop(1));

System.out.println(" (5)栈2的出栈操作");

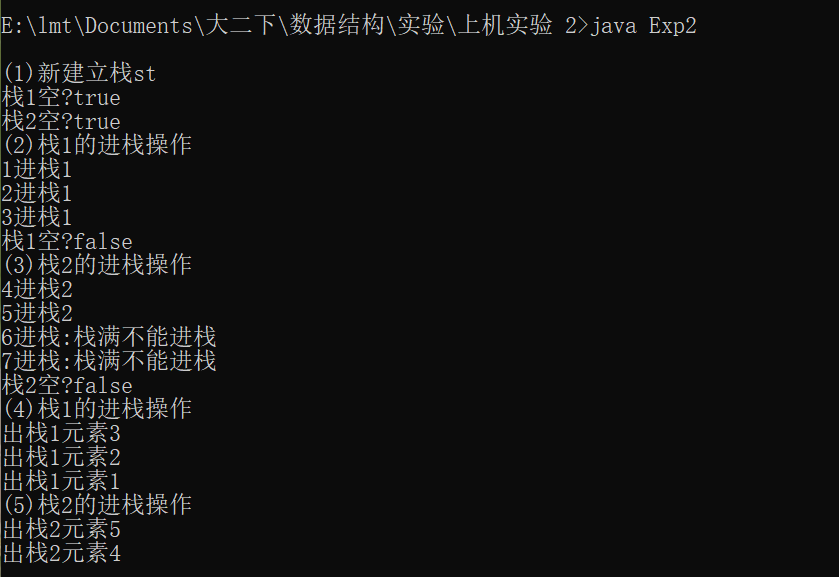
while (!st.empty(2))

System.out.println(" 出栈2元素"+st.pop(2));

}

}

## 四、实验结果



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验三 队列操作实现及应用

**学时： 2 实验类型：设计性实验**

## 一、实验目的

1.掌握队列存储结构和队列中各种基本运算算法设计。

2.掌握队列在求解迷宫问题中的应用。

## 二、实验内容

POJ3984—迷宫问题。

定义一个二维数组，它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，注意只能横着走或竖着走，不能斜着走，要求编程找出从左上角到右下角的最短路径。输入格式：一个5🞨5的二维数组，表示一个迷宫。数据保证有唯一解。输出格式：从左上角到右下角的最短路径。具体输入输出样例详见教材 3.4.2第5题。

## 三、实验算法实现

由于求的是最短路径，采用队列求迷宫问题的方法：

import java.io.\*;

import java.util.\*;

class Box //方块结构体类型

{ int i; //方块的行号

int j; //方块的列号

Box pre; //前驱方块

public Box(int i1,int j1) //构造方法

{

i=i1;

j=j1;

pre=null;

}

}

class MazeClass //用队列求解一条迷宫路径类

{

final int MaxSize=6;

int [][] mg; //迷宫数组

int m,n; //迷宫行列数

Queue<Box> qu; //队列

public MazeClass(int m1,int n1)

{

m=m1;

n=n1;

mg=new int[MaxSize][MaxSize];

qu=new LinkedList<Box>(); //创建一个空队qu

}

public void Setmg(int [][] a) //数组迷宫数组

{

for (int i=0;i<m;i++)

for (int j=0;j<n;j++)

mg[i][j]=a[i][j];

}

public void disppath(Box p) //从p出发找一条迷宫路径

{ Box[] path = new Box[20];

int cnt=0;

while (p!=null) //找到入口为止

{ path[cnt]=p;

cnt++;

p=p.pre;

}

for(int k=cnt-1;k>=0;k--)

System.out.println("("+path[k].i+", "+path[k].j+")");

}

boolean mgpath(int xi,int yi,int xe,int ye)//求(xi,yi)到(xe,ye)的一条路径

{

int i,j,i1=0,j1=0;

Box b,b1;

b=new Box(xi,yi); //建立入口结点b

qu.offer(b); //结点b进队

mg[xi][yi]=-1; //进队方块mg值置为-1

while (!qu.isEmpty()) //队不空时循环

{

b=qu.poll(); //出队一个方块b

if (b.i==xe && b.j==ye) //找到了出口,输出路径

{ disppath(b); //从b出发回推导出迷宫路径并输出

return true; //找到一条路径时返回true

}

i=b.i; j=b.j;

for (int di=0;di<4;di++) //循环扫描每个方位,把每个可走的方块进队

{

switch(di)

{

case 0:i1=i-1; j1=j; break;

case 1:i1=i; j1=j+1; break;

case 2:i1=i+1; j1=j; break;

case 3:i1=i; j1=j-1; break;

}

if (i1>=0 && i1<m && j1>=0 && j1<n && mg[i1][j1]==0)//找到一个相邻可走方块

{

b1=new Box(i1,j1); //建立后继方块结点b1

b1.pre=b; //设置其前驱方块为b

qu.offer(b1); //b1进队

mg[i1][j1]=-1; //进队的方块置为-1

}

}

}

return false; //未找到任何路径时返回false

}

}

public class Main

{

public static void main(String[] args)

{

Scanner fin=new Scanner(System.in);

MazeClass mz=new MazeClass(5,5);

int [][] a=new int[5][5];

for(int i=0;i<5;i++)

for(int j=0;j<5;j++)

a[i][j]=fin.nextInt();

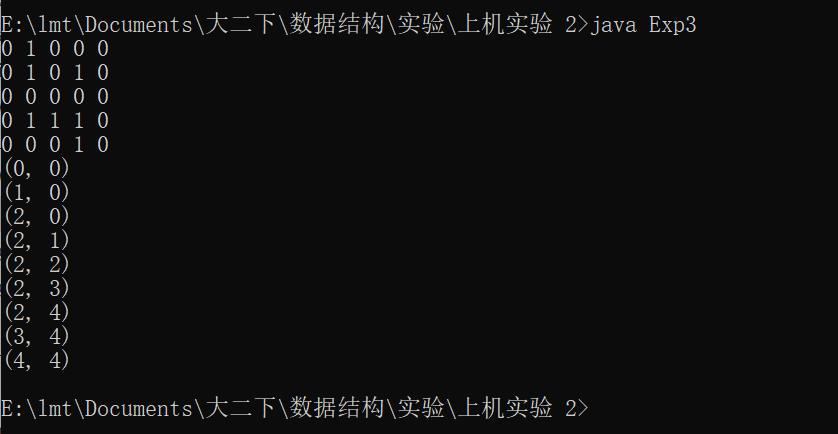
mz.Setmg(a);

mz.mgpath(0,0,4,4);

}

}

## 四、实验结果



## 五、实验要求

1.按要求完成实验报告。

2.实验内容“POJ3984—迷宫问题” 需要在[http://poj.org](http://poj.org/)平台在线提交，将通过证明截图并放入实验报告中。

3.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验四 串的模式匹配算法应用

**学时： 2 实验类型：验证性实验**

## 一、实验目的

1.掌握串的模式匹配算法。

2.掌握串的模式匹配应用算法设计。

## 二、实验内容

1.编写一个实验程序，求字符串s中出现的最长的可重叠的重复子串。例如：s=“ababababa”,输出结果为“abababa”。

2.编写一个实验程序，求字符串s中出现的最长的不重叠的重复子串。例如：s=“ababababa”,输出结果为“abab”。

## 三、实验算法实现

1.采用简单匹配算法的思路：

import java.lang.\*;

import java.util.\*;

public class Exp2

{

public static String maxsubstr(String s)

{ int maxi=0,maxlen=0,len,i=0,j,k;

while (i<s.length()) //i遍历s

{ j=i+1; //从i+1开始

while (j<s.length()) //查找与si开头的相同重复子串

{

if (s.charAt(i)==s.charAt(j)) //遇到首字符相同

{ len=1;

while (j+len<s.length() && s.charAt(i+len)==s.charAt(j+len))

len++; //找到重复子串(i,len)

if (len>maxlen) //存放较长子串(maxi,maxlen)

{ maxi=i;

maxlen=len;

}

j+=len; //j跳过len个字符继续在后面找子串

}

else j++; //首字符不相同，j递增

}

i++; //i后移

}

String t=""; //构造最长重复子串

if (maxlen==0)

t="null";

else

{

for (i=0;i<maxlen;i++)

t+=s.charAt(maxi+i);

}

return t;

}

public static void main(String[] args)

{

System.out.println("");

System.out.println(" 测试1");

String s="ababcabccabdce";

System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

System.out.println(" 测试2");

s="abcd";

System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

System.out.println(" 测试3");

s="ababababa";

System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

}

}

2.与上一个实验思路类似：

import java.lang.\*;

import java.util.\*;

public class Exp3

{

public static String maxsubstr(String s)

{ int maxi=0,maxlen=0,len,i=0,j,k;

while (i<s.length()) //i遍历s

{ j=i+1; //从i+1开始

while (j<s.length()) //查找与si开头的相同重复子串

{

if (s.charAt(i)==s.charAt(j)) //遇到首字符相同

{ len=1;

while (j+len<s.length() && s.charAt(i+len)==s.charAt(j+len))

{

len++; //找到重复子串(i,len)

if (i+len+1>j) break;//重叠时退出

}

if (len>maxlen) //存放较长子串(maxi,maxlen)

{ maxi=i;

maxlen=len;

}

j+=len; //j跳过len个字符继续在后面找子串

}

else j++; //首字符不相同，j递增

}

i++; //i后移

}

String t=""; //构造最长重复子串

if (maxlen==0)

t="null";

else

{

for (i=0;i<maxlen;i++)

t+=s.charAt(maxi+i);

}

return t;

}

public static void main(String[] args)

{

System.out.println("");

System.out.println(" 测试1");

String s="ababcabccabdce";

System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

System.out.println(" 测试2");

s="abcd";

System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

System.out.println(" 测试3");

s="ababababa";

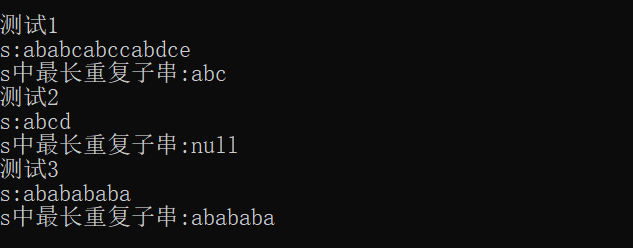
System.out.println(" s: "+s);

System.out.println(" s中最长重复子串: "+maxsubstr(s));

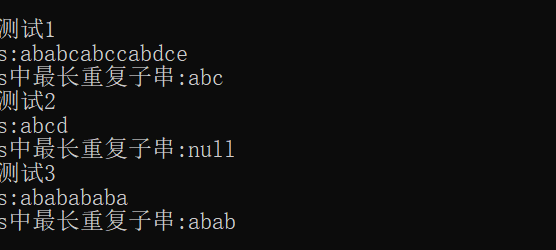
}

## }四、实验结果

1.第一个实验结果：



2.第二个实验结果：



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验五 二叉树的遍历算法应用

**学时： 2 实验类型：验证性实验**

## 一、实验目的

1.掌握二叉树的先序、中序、后序三种遍历算法。

2.掌握二叉树的层次遍历算法。

## 二、实验内容

编写一个实验程序，假设二叉树采用二叉链存储结构，所有结点值为单个字符且不相同。采用先序遍历、层次遍历方式，输出二叉树中从根结点到每个叶子结点的路径，并且利用教材图7.17进行测试。

## 三、实验算法实现

先序遍历求解思路见教材例7.11的解法2，层次遍历求解思路见教材例7.16。

public class Exp4

{

public static void AllPath1(BTreeClass bt) //解法1：先序遍历

{

char[]path=new char[100];

int d=-1;

PreOrder(bt.b,path,d);

}

private static void PreOrder(BTNode<Character> t,char[]path,int d) //先序遍历输出结果

{

if (t!=null)

{

if (t.lchild==null && t.rchild==null) //t为叶子结点

{

System.out.printf(" 根结点到%c的路径: ",t.data);

for (int i=0;i<=d;i++)

System.out.print(path[i]+" ");

System.out.println(t.data);

}

else

{ d++; //路径长度增1

path[d]=t.data; //将当前结点放入路径中

PreOrder(t.lchild,path,d); //递归遍历左子树

PreOrder(t.rchild,path,d); //递归遍历右子树

}

}

}

public static void AllPath3(BTreeClass bt) //解法2：层次遍历

{

class QNode

{

BTNode<Character> node; //结点引用

QNode parent; //结点的双亲

public QNode(BTNode<Character> no,QNode pa) //构造方法

{ node=no;

parent=pa;

}

}

Queue<QNode> qu=new LinkedList<QNode>(); //定义一个队列qu

QNode p;

qu.offer(new QNode(bt.b,null)); //根结点的双亲为null

while (!qu.isEmpty()) //队列不为空

{

p=qu.poll(); //出队一个结点

if (p.node.lchild==null && p.node.rchild==null) //p为叶子结点

{

String str="";

QNode f=p;

while (f!=null)

{

str+=f.node.data;

f=f.parent;

}

System.out.printf(" 根结点到%c的路径: ",p.node.data);

for(int i=str.length()-1;i>=0;i--)

System.out.print(str.charAt(i)+" ");

System.out.println();

}

if (p.node.lchild!=null) //左孩子结点进队

qu.offer(new QNode(p.node.lchild,p));

if (p.node.rchild!=null) //右孩子结点进队

qu.offer(new QNode(p.node.rchild,p));

}

}

public static void solve(BTreeClass bt)

{

System.out.println();

System.out.println(" 求解结果");

System.out.println(" 解法1:");

AllPath1(bt);

System.out.println(" 解法2:");

AllPath2(bt);

System.out.println(" 解法3:");

AllPath3(bt);

}

public static void main(String[] args)

{ String s="A(B(D(,F(H)),E(G)),C)";

BTreeClass bt=new BTreeClass();

bt.CreateBTree(s);

System.out.println();

System.out.println(" bt: "+bt.toString());

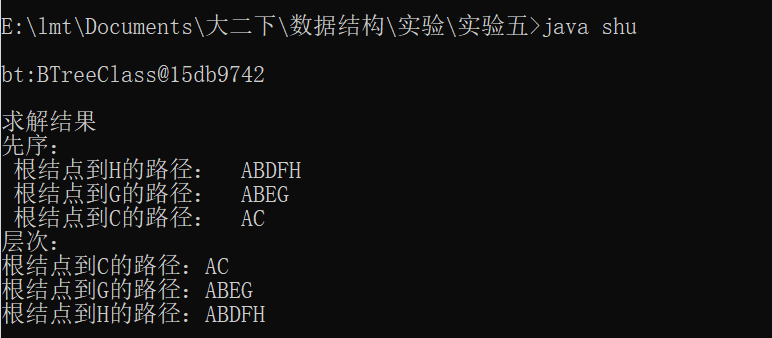
solve(bt);

//PostOrder2(bt);

}

}

## }四、实验结果



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验六 基于哈夫曼树的数据压缩算法

**学时： 2 实验类型：设计性实验**

## 一、实验目的

1.掌握哈夫曼树的构造算法。

2.掌握哈夫曼编码的构造算法。

## 二、实验内容

编写一个实验程序，给定一个字符串，统计其中ASCII码为0-255的字符出现的次数，构造出对应的哈夫曼编码，用其替换产生加密码（其他字符保持不变），再由加密码解密产生原字符串，并且利用相关数据进行测试。

## 三、实验算法实现

采用教材7.7节中的原理构造哈夫曼树并产生哈夫曼编码，由原字符串s产生加密码str，再由str解密产生s。对应的算法程序如下：

import java.lang.\*;

import java.util.\*;

@SuppressWarnings("unchecked")

class HTNode //哈夫曼树结点类

{ char data; //结点值,假设为单个字符

double weight; //权值

public HTNode parent; //双亲结点

HTNode lchild; //左孩子结点

HTNode rchild; //右孩子结点

boolean flag; //标识是双亲的左或者右孩子

public HTNode() //构造方法

{ parent=null;

lchild=null;

rchild=null;

data='A';

}

public double getw() //取结点权值的方法

{

return weight;

}

}

public class Exp6

{ final static int MAXN=562; //最多结点个数

static HTNode[] ht=new HTNode[MAXN]; //存放哈夫曼树

static HTNode root; //哈夫曼树的根结点

static String[] hcd=new String[MAXN]; //存放哈夫曼编码

static int[] w=new int[MAXN]; //权值数组

static char[] d=new char[MAXN]; //存放字符

static int n0; //存放叶子结点个数

static int[] map=new int[MAXN]; //ASCII到下标转换

public static void CreateHT() //构造哈夫曼树

{ Comparator<HTNode> priComparator //定义priComparator

= new Comparator<HTNode>()

{ public int compare(HTNode o1,HTNode o2) //用于创建小根堆

{

return (int)(o1.getw()-o2.getw()); //按weight越小越优先

}

};

PriorityQueue<HTNode> pq=new PriorityQueue<>(MAXN,priComparator); //定义优先队列

for (int i=0;i<n0;i++) //建立n0个叶子结点并进队

{ ht[i]=new HTNode(); //建立ht[i]结点

ht[i].parent=null; //双亲设置为空

ht[i].data=d[i];

ht[i].weight=w[i];

pq.offer(ht[i]);

}

for (int i=n0;i<(2\*n0-1);i++) //合并操作

{ HTNode p1=pq.poll(); //出队两个权值最小的结点p1和p2

HTNode p2=pq.poll();

ht[i]=new HTNode(); //建立ht[i]结点

p1.parent=ht[i]; //设置p1和p2的双亲为ht[i]

p2.parent=ht[i];

ht[i].weight=p1.weight+p2.weight; //求权值和

ht[i].lchild=p1; //p1作为双亲ht[i]的左孩子

p1.flag=true;

ht[i].rchild=p2; //p2作为双亲ht[i]的右孩子

p2.flag=false;

pq.offer(ht[i]); //ht[i]结点进队

}

root=pq.poll(); //队中最后一个结点就是根结点

}

private static String reverse(String s) //逆置字符串s

{ String t="";

for (int i=s.length()-1;i>=0;i--)

t+=s.charAt(i);

return t;

}

public static void CreateHCode() //根据哈夫曼树求哈夫曼编码

{ for (int i=0;i<n0;i++) //遍历下标从0到n0-1的叶子结点

{ hcd[i]="";

HTNode p=ht[i]; //从ht[i]开始找双亲结点

while (p.parent!=null)

{ if (p.flag) //p结点是双亲的左孩子

hcd[i]+='0';

else //p结点是双亲的右孩子

hcd[i]+='1';

p=p.parent;

}

hcd[i]=reverse(hcd[i]); //逆置

}

}

public static void DispHuffman() //输出哈夫曼编号

{ String str;

for (int i=0;i<n0;i++)

{ str=" "+ht[i].data+": "+hcd[i];

if((i+1)%3==0) //每行输出3个字符

System.out.println(str);

else

{ System.out.print(str);

if (str.length()<8)

System.out.print("\t\t");

else

System.out.print("\t");

}

}

}

public static String encryption(String s) //加密方法

{ String str="";

for (int i=0;i<s.length();i++)

{ int cd=(int)s.charAt(i); //取字符s[i]的ASCII码cd

if (cd>=0 && cd<256) //有效字符

{

int j=map[cd]; //s[i]对应ht/hcd中的下标为j

str+=hcd[j]; //替换为哈夫曼编码

}

else //其他字符保持不变

str+=s.charAt(i);

}

return str;

}

public static String Decrypt(String s) //解密方法

{ String str="";

int i=0;

while (i<s.length())

{ int cd=s.charAt(i);

if (cd>=0 && cd<256) //有效字符

{ HTNode p=root; //从根结点开始匹配

while (i<s.length())

{ if (s.charAt(i)=='0')

p=p.lchild;

else

p=p.rchild;

i++;

if (p.lchild==null && p.rchild==null)

break; //匹配叶子结点时退出

}

str+=p.data;

}

else //其他字符保持不变

{ str+=s.charAt(i);

i++;

}

}

return str;

}

public static void DispStr(String str) //输出字符串str

{ System.out.print(" ");

for (int i=0;i<str.length();i++)

{ if ((i+1)%60==0)

{ System.out.println(str.charAt(i));

System.out.print(" ");

}

else

System.out.print(str.charAt(i));

}

System.out.println();

}

public static void main(String[] args)

{ String s="China Connect is 好the 样largest的 European gathering.";

System.out.println("\n 原字符串:"); DispStr(s);

int[] cnt=new int[256];

int cd;

for (int i=0;i<s.length();i++)

{ cd=(int)s.charAt(i);

if (cd>=0 && cd<256)

cnt[cd]++;

}

n0=0;

for (int i=0;i<256;i++)

if (cnt[i]>0)

{ d[n0]=(char)i; //存放ASCII为i的字符

map[i]=n0; //ASCII为i对应ht/hcd的下标n0

w[n0]=cnt[i]; //该字符出现的次数

n0++;

}

CreateHT();

CreateHCode();

System.out.println(" 各个字符的哈夫曼编码:");

DispHuffman();

System.out.println(" 加密结果:");

String str=encryption(s); DispStr(str);

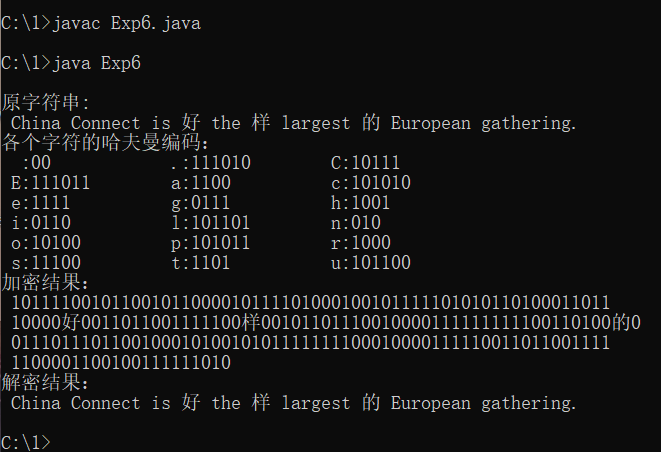
System.out.println(" 解密结果:");

s=Decrypt(str); DispStr(s);

}

}

## }四、实验结果



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验七 图的建立和遍历

**学时： 2 实验类型：验证性实验**

## 一、实验目的

1.掌握使用邻接矩阵或邻接表存储一个图。

2.掌握图的深度优先遍历算法及应用。

3.掌握图的广度优先遍历算法及应用。

## 二、实验内容

1.一个图G采用邻接矩阵作为存储结构，设计一个算法采用广度优先遍历判断顶点i到顶点j是否有路径（假设顶点i和j都是G中的顶点）。并给定一个图，其邻接矩阵数组为：{{0,1,0,0,1},{1,0,0,0,1},{0,0,0,1,1},{0,0,1,0,1},{1,1,1,1,0}}，顶点数n=5，边数e=6，利用设计的算法判断从顶点0到顶点3是否有路径。

2.一个图G采用邻接表作为存储结构，设计一个算法采用深度优先遍历判断顶点i到顶点j是否存在不包含顶点k的路径（假设顶点i、j、k都是G中的顶点并且不相同）。并给定一个图，其邻接矩阵数组为：{{0,1,0,0,1},{1,0,0,0,1},{0,0,0,1,1},{0,0,1,0,1},{1,1,1,1,0}}，顶点数n=5，边数e=6，利用设计的算法判断从顶点0到顶点2是否存在不包含顶点4的路径。

## 三、实验算法实现

1.先置visited数组的所有元素为0，从顶点i出发进行广度优先遍历，遍历完毕的若visited[j]为1，则顶点i到顶点j有路径，否则没有路径。对应的算法如下：

import java.lang.\*;

import java.util.\*;

public class Excise4

{

static final int MAXV=100; //表示最多顶点个数

static int[] visited=new int[MAXV]; //全局变量数组

static void BFS2(MatGraphClass g,int v) //邻接矩阵g中从顶点v出发的广度优先遍历

{ Queue<Integer> qu=new LinkedList<Integer>();//定义一个队列

visited[v]=1; //置已访问标记

qu.offer(v); //v进队

while (!qu.isEmpty()) //队列不空循环

{ v=qu.poll(); //出队顶点v

for (int w=0;w<g.n;w++)

{ if (g.edges[v][w]!=0 &&g.edges[v][w]!=g.INF)

{ if (visited[w]==0) //存在边<v,w>并且w未访问

{ visited[w]=1; //置已访问标记

qu.offer(w); //w进队

}

}

}

}

}

static boolean Pathij(MatGraphClass g,int i,int j) //求解算法

{

Arrays.fill(visited,0);

BFS2(g,i);

if (visited[j]==1)

return true;

else

return false;

}

public static void main(String[] args)

{

MatGraphClass g=new MatGraphClass();

int n=5,e=6;

int[][] a={{0,1,0,0,1}, {1,0,0,0,1},

{0,0,0,1,1},{0,0,1,0,1},{1,1,1,1,0}};

g.CreateMatGraph(a,n,e);

System.out.println("g:"); g.DispMatGraph();

int i=0,j=3;

if (Pathij(g,i,j))

System.out.println("Yes");

else

System.out.println("No");

}

}

2.采用深度优先遍历的算法如下：

import java.lang.\*;

import java.util.\*;

public class Excise5

{

static int MAXV=100;

static int[] visited=new int[MAXV]; //全局变量数组

static void DFS2(AdjGraphClass G,int v) //从顶点v出发深度优先遍历

{ ArcNode p;

visited[v]=1;

p=G.adjlist[v].firstarc;

while (p!=null)

{

if (visited[p.adjvex]==0)

DFS2(G,p.adjvex);

p=p.nextarc;

}

}

static boolean Pathijk(AdjGraphClass G,int i,int j,int k) //求解算法

{

Arrays.fill(visited,0);

visited[k]=1;

DFS2(G,i);

if (visited[j]==1)

return true;

else

return false;

}

public static void main(String[] args)

{

AdjGraphClass G=new AdjGraphClass();

int n=5,e=6;

int[][] a={{0,1,0,0,1}, {1,0,0,0,1},

{0,0,0,1,1},{0,0,1,0,1},{1,1,1,1,0}};

G.CreateAdjGraph(a,n,e);

System.out.println("G:"); G.DispAdjGraph();

int i=0,j=2,k=4;

if (Pathijk(G,i,j,k))

System.out.println("Yes");

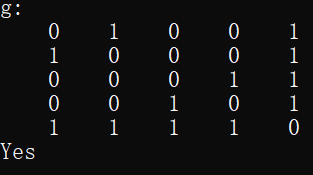
else

System.out.println("No");

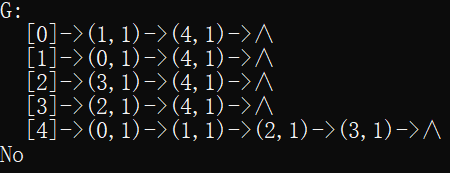
}

## 四、实验结果

1.第一个实验结果：



2.第二个实验结果：



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 实验八 查找算法实现与应用

**学时： 3 实验类型：设计性实验**

## 一、实验目的

1.掌握二叉排序树的创建算法。

2.理解二叉排序树的查找方法。

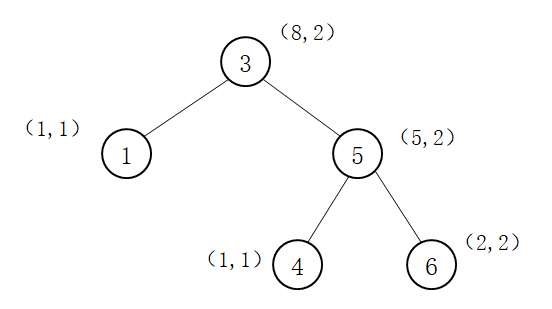
3.掌握二叉排序树查找的算法实现及应用。

## 二、实验内容

有一个整数序列，其中存在相同的整数，创建一棵二叉排序树。按递增顺序输出所有不同整数的名次（第几小的整数，从1开始计）。例如：整数序列为（3,5,4,6,6,5,1,3），求解结果是：1的名次为1、3的名次为2、4的名次为4、5的名次为5、6的名次为7。

## 三、实验算法实现

在二叉排序树的结点中增加size成员表示以这个结点为根的子树中结点的个数（包括自己），cnt成员表示相同关键字出现的次数（把key相同的整数放在一个结点中）。采用教材中第9章的方法创建这样的二叉排序树bt，例如由整数序列（3,5,4,6,6,5,1,3）创建的二叉排序树如下图所示，结点旁边的（x，y），x表示size，y表示cnt。



求二叉排序树中关键字k的名次，即树中小于等于k的结点个数，这里是最小排名，若有多个关键字k，返回第一个k的名次。在以结点p为根结点的子树中求关键字k的名次过程如下：

（1）若当前结点p的关键字等于k，返回其左子树结点个数+1.

（2）若k小于结点p的关键字，说明结果在左子树中，返回在左子树中查找关键字k的名次的结果。

（3）若k大于结点p的关键字，说明结果在右子树中，则需要将右子树查找到的关键字k的结果加上左子树和根的名次。

import java.util.\*;

class BSTNode //二叉排序树结点类

{ public int key; //存放关键字,假设关键字为int类型

int size; //以这个结点为根的子树中结点个数

int cnt; //相同关键字出现次数

public BSTNode lchild; //存放左孩子指针

public BSTNode rchild; //存放右孩子指针

BSTNode() //构造方法

{

lchild=rchild=null;

size=cnt=1;

}

}

class BSTClass //二叉排序树类

{ public BSTNode r; //二叉排序树根结点

public BSTClass() //构造方法

{ r=null; }

public void InsertBST(int k) //插入一个关键字为k的结点

{

InsertBST1(r,k);

}

private int getsize(BSTNode p) //求结点p的size

{

if (p==null)

return 0;

else

return p.size;

}

private BSTNode InsertBST1(BSTNode p,int k) //在以p为根的BST中插入关键字为k的结点

{

if (p==null) //原树为空,新插入的元素为根结点

{ p=new BSTNode();

p.key=k;

}

else if (k==p.key)

{

p.cnt++;

}

else if (k<p.key)

p.lchild=InsertBST1(p.lchild,k); //插入到p的左子树中

else if (k>p.key)

p.rchild=InsertBST1(p.rchild,k); //插入到p的右子树中

p.size=getsize(p.lchild)+getsize(p.rchild)+p.cnt; //维护结点p的size值

return p;

}

public void CreateBST(int[] a) //由关键字序列a创建一棵二叉排序树

{ r=new BSTNode(); //创建根结点

r.key=a[0];

for (int i=1;i<a.length;i++) //创建其他结点

InsertBST1(r,a[i]); //插入关键字a[i]

}

public int Rank(int k) //求关键字k的名次

{

return Rank1(r,k);

}

private int Rank1(BSTNode p,int k) //在结点p的子树中求关键字k的名次

{ if (p==null) return 0;

if(k==p.key) //找到关键字为k的结点

return getsize(p.lchild)+1;

if(k<p.key)

return Rank1(p.lchild,k);

else

return Rank1(p.rchild,k)+getsize(p.lchild)+p.cnt;

}

public void DispBST() //输出二叉排序树的括号表示串

{

DispBST1(r);

}

private void DispBST1(BSTNode p) //被DispBST方法调用

{ if (p!=null)

{

System.out.print(p.key+"["+p.size+","+p.cnt+"]");//输出根结点值

if (p.lchild!=null || p.rchild!=null)

{ System.out.print("("); //有孩子结点时才输出“(”

DispBST1(p.lchild); //递归处理左子树

if (p.rchild!=null)

System.out.print(",");//有右孩子结点时才输出“,”

DispBST1(p.rchild); //递归处理右子树

System.out.print(")"); //有孩子结点时才输出“)”

}

}

}

}

public class Exp3

{

public static void main(String[] args)

{

int[] a={3,5,4,6,6,5,1,3};

BSTClass bt=new BSTClass();

bt.CreateBST(a);

System.out.println();

System.out.print(" BST: "); bt.DispBST(); System.out.println();

TreeSet<Integer> s=new TreeSet();

for (int i=0;i<a.length;i++) //采用TreeSet集合去重

s.add(a[i]);

System.out.println(" 求解结果");

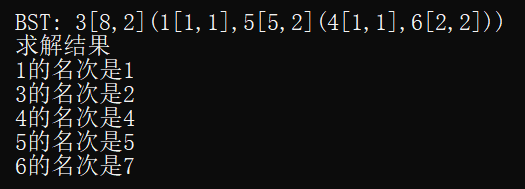
for (Integer e:s)

System.out.println(" "+e+"的名次是"+bt.Rank(e));

}

}

## 四、实验结果



## 五、实验要求

1.按要求完成实验报告。

2.将实验报告word文档命名为“学号+姓名”，并将实验报告文档上传提交至雨课堂平台。

# 