**实验七 时空权衡**

1. **实验目的**
2. 掌握时空权衡的空间换时间的含义。
3. 熟练掌握分布计数排序的设计思想和实现条件。
4. 熟练掌握字符串匹配中时间换空间的思想。
5. 理解散列法的设计思想。
6. **实验要求**
7. 熟悉把算法变成代码的方法；
8. 掌握Horspool算法和Boyer-Moore算法的设计过程。
9. 掌握散列法的实现过程
10. **主要仪器及耗材**

计算机及相关软件

1. **基础实验**

1. 假设列表可能值属于集合{a,b,c,d},编程实现课本上的分布计数算法将下面的列表按照字母顺序排序。

*b, c, d, c, b, a, a, b*

***package count;***

***import java.util.Arrays;***

***import java.util.Scanner;***

***public class distributionCountingSort {***

***/\*\****

***\* 分布计数法，对来自有限范围整数的一个数组进行排序***

***\* 输入：数组A[0...n-1]，数组中整数位于l和u之间***

***\* 输出：A中元素构成的非降序数组S[0...n-1]***

***\*/***

***public static String disCountSort(char[] A,char l,char u) {***

***int gap=u-l+1;***

***//初始化频率数组D***

***int[] D=new int[gap];***

***char[] S=new char[A.length];***

***for(int j=0;j<gap;j++) {***

***D[j]=0;***

***}***

***//计算频率值***

***for(int i=0;i<A.length;i++) {***

***D[A[i]-l]=D[A[i]-l]+1;***

***}***

***System.out.println("频率分布为："+Arrays.toString(D));***

***//重用于分布***

***for(int j=1;j<=u-l;j++) {***

***D[j]=D[j-1]+D[j];***

***}***

***for(int k=A.length-1;k>=0;k--) {***

***int j=A[k]-l;***

***S[D[j]-1]=A[k];***

***D[j]=D[j]-1;***

***}***

***return Arrays.toString(S);***

***}***

***public static void main(String[] args) {***

***Scanner input=new Scanner(System.in);***

***System.out.println("请输入数组元素：");***

***//声明变量并赋值***

***String str=input.nextLine();***

***String[] array=str.split(",");***

***String s="";***

***for(int i=0;i<array.length;i++) {***

***s+=array[i];***

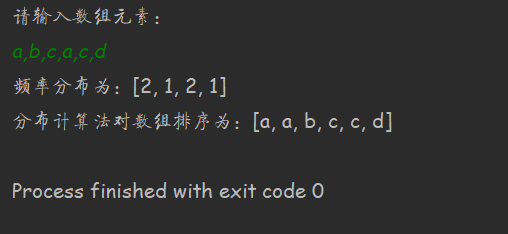
***}***

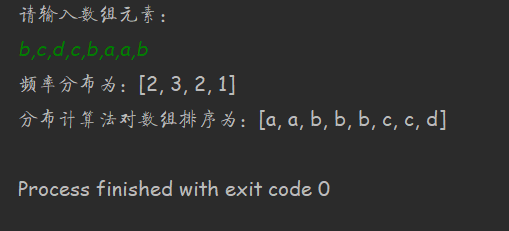
***char[] A=s.toCharArray();***

***System.out.println("分布计算法对数组排序为："+disCountSort(A, 'a', 'd'));***

***}***

***}***





1. 写一个程序，处理两个次数分别是m和n的稀疏多项式p(x)和q(x)的相乘。

**package count;**

**import java.util.Arrays;**

**import java.util.Scanner;**

**public class matrixMain {**

**/\*\***

**\* 稀疏矩阵相乘（矩阵中0的数量比1多）**

**\*/**

**public static void matrix(int[][] data1,int[][] data2) {**

**int row1=data1.length;**

**int col1=data1[0].length;**

**int row2=data2.length;**

**int col2=data2[0].length;**

**int[][] result=new int[row1][col2];**

**for(int i=0;i<row1;i++) {**

**for(int j=0;j<col1;j++) {**

**for(int k=0;k<col2;k++) {**

**result[i][j]+=data1[i][k]\*data2[k][j];**

**}**

**}**

**}**

**for(int i=0;i<row1;i++) {**

**for(int j=0;j<col2;j++) {**

**System.out.print(result[i][j]+" ");**

**}**

**System.out.println();**

**}**

**}**

**public static void main(String[] args) {**

**Scanner input=new Scanner(System.in);**

**System.out.println("请输入第一个稀疏矩阵的行和列:");**

**String num1=input.nextLine();**

**int row1=Integer.valueOf(num1.split(" ")[0]); //行**

**int col1=Integer.valueOf(num1.split(" ")[1]); //列**

**int[][] matrix1=new int[row1][col1];**

**for(int i=0;i<row1;i++) {**

**System.out.println("请输入第"+(i+1)+"行的元素：");**

**String s=input.nextLine();**

**String[] sArray=s.split(" ");**

**for(int j=0;j<col1;j++) {**

**matrix1[i][j]=Integer.valueOf(sArray[j]);**

**}**

**}**

**System.out.println("请输入第二个稀疏矩阵的行和列:");**

**String num2=input.nextLine();**

**int row2=Integer.valueOf(num2.split(" ")[0]); //行**

**int col2=Integer.valueOf(num2.split(" ")[1]); //列**

**int[][] matrix2=new int[row2][col2];**

**for(int i=0;i<row2;i++) {**

**System.out.println("请输入第"+(i+1)+"行的元素：");**

**String s=input.nextLine();**

**String[] sArray=s.split(" ");**

**for(int j=0;j<col2;j++) {**

**matrix2[i][j]=Integer.valueOf(sArray[j]);**

**}**

**}**

**matrix(matrix1, matrix2);**

**}**

**}**

3. 任意选择一种语言实现Horspool算法、Boyer-Moore算法，并和前面的蛮力字符串匹配算法做一个实验来比较它们在做下列匹配时的效率。

a. 在随机二进制文本中匹配随机的二进制模式。

b. 在自然语言文本中匹配随机的自然语言模式。

**package count;**

**import java.util.HashMap;**

**public class Horspool {**

**/\*\***

**\* Horspool算法匹配文本**

**\* @author**

**\*/**

**public static final char[] CHAR\_TABLE = { 'a', 'b', 'c', 'd', 'e', 'f',**

**'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',**

**'t', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F',**

**'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',**

**'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5',**

**'6', '7', '8', '9', '(', ')', '{', '}', '[', ']', '<', '>', ',',**

**'.', '\_', '-', '+', '=', '~', '/', '?', ';', ':', '"', '|', '!',**

**'@', '#', '$', '%', '^', '&', '\*' };**

**public static HashMap<Character, Integer> ShiftTable(String p) {**

**int m = p.length();**

**HashMap<Character, Integer> table = new HashMap<Character, Integer>();**

**for (int i = 0; i < CHAR\_TABLE.length; i++)**

**table.put(CHAR\_TABLE[i], m);**

**for (int i = 0; i < m - 1; i++)**

**table.put(p.charAt(i), m - 1 - i);**

**return table;**

**}**

**public static int HorspoolMatching(String p, String t) {**

**HashMap<Character, Integer> table = ShiftTable(p);**

**int m = p.length();**

**int n = t.length();**

**int i = m - 1; // 模式左右边的位置**

**while (i <= n - 1) {**

**int k = 0;**

**while ((k <= m - 1) && p.charAt(m - 1 - k) == t.charAt(i - k)) {**

**k++;**

**}**

**if (k == m)**

**return i - (m - 1);**

**else**

**i += table.get(t.charAt(i));**

**}**

**return -1;**

**}**

**public static void main(String[] argv) {**

**long startTime = System.currentTimeMillis();//获取当前时间**

**System.out.println("\*\*\*\*\*\*Horspool算法匹配字符串\*\*\*\*\*");**

**String p="BARBER";**

**System.out.println("模式匹配："+p);**

**String t="JIM\_SAW\_BOY\_ME\_IN\_A\_BARBERSHOP\_DL\_UNIVERSITY";**

**System.out.println("匹配字符串："+t);**

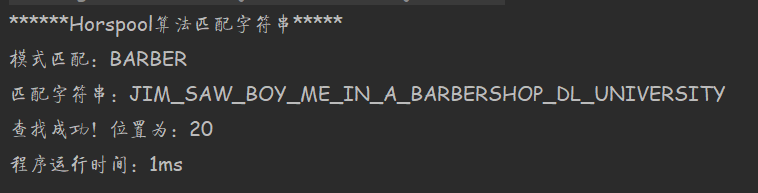
**System.out.println("查找成功！位置为："+HorspoolMatching(p, t));**

**long endTime = System.currentTimeMillis();//获取当前时间**

**System.out.println("程序运行时间："+(endTime-startTime)+"ms");**

**}**

**}**



**package count;**

**public class BoyerMoore {**

**/\*\***

**\* 构造移动表**

**\*/**

**public static void getRight(String pat, int[] right) {**

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

**right[i] = -1;**

**}**

**for (int i = 0; i < pat.length(); i++) {**

**right[pat.charAt(i)] = i;**

**}**

**}**

**/\*\***

**\* Boyer-Moore算法匹配字符串**

**\*/**

**public static int BoyerMooreSearch(String txt, String pat, int[] right) {**

**int M = txt.length();**

**int N = pat.length();**

**int skip;**

**for (int i = 0; i <= M - N; i += skip) {**

**skip = 0;**

**for (int j = N - 1; j >= 0; j--) {**

**if (pat.charAt(j) != txt.charAt(i + j)) {**

**skip = j - right[txt.charAt(i + j)];**

**if (skip < 1){**

**skip = 1;**

**}**

**break;**

**}**

**}**

**if (skip == 0)**

**return i;**

**}**

**return -1;**

**}**

**public static void main(String[] args) {**

**long startTime = System.currentTimeMillis();//获取当前时间**

**System.out.println("\*\*\*\*\*\*Boyer-Moore算法匹配字符串\*\*\*\*\*");**

**String txt = "JIM\_SAW\_BOY\_ME\_IN\_A\_BARBERSHOP\_DL\_UNIVERSITY";**

**System.out.println("匹配字符串："+txt);**

**String pat = "BARBER";**

**System.out.println("模式匹配："+pat);**

**int[] right = new int[256];**

**getRight(pat,right);**

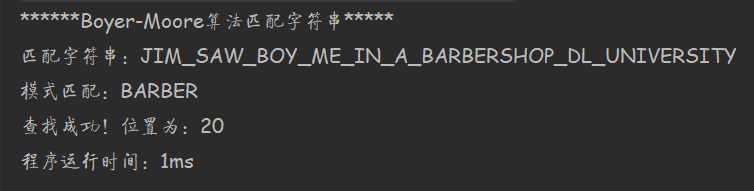
**System.out.println("查找成功！位置为："+BoyerMooreSearch(txt, pat, right));**

**long endTime = System.currentTimeMillis();//获取当前时间**

**System.out.println("程序运行时间："+(endTime-startTime)+"ms");**

**}**

**}**



1. 写一个计算机程序，它使用散列技术来处理下面的问题。对于给定的自然语言文本，生成一个不重复词的列表，包括每个词在文本中出现次数。在程序中插入恰当的计数器，对散列法的效率做经验分析，并拿结果和相应的理论结果做比较。

**package count;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.Scanner;**

**public class HashTables {**

**int[] elem; //散列表数据存储数组**

**public int counts; //散列表实际存储数据量**

**private int maxSize =10 ; //散列表的最大容量**

**public final int NULLKEY = 10; //散列表初始值**

**public final int SUCCESS = 1;**

**public final int UNSUCCESS = 0;**

**public HashTables() {**

**this.elem = new int[maxSize];**

**this.initHashTable();**

**}**

**public HashTables(int maxsize) {**

**this.maxSize = maxsize;**

**this.elem = new int[maxSize];**

**this.initHashTable();**

**}**

**//初始化散列表**

**public void initHashTable() {**

**for (int i = 0; i < maxSize; i++) {**

**this.elem[i]= NULLKEY;**

**}**

**}**

**/\*\***

**\* 散列函数**

**\* 除留取余法**

**\* @param key**

**\* @return**

**\*/**

**public int Hash(int key) {**

**return key % maxSize;**

**}**

**/\*\***

**\* 散列表插入操作**

**\* @param key**

**\*/**

**public void insertHash(int key) {**

**int addr = Hash(key); //求散列地址**

**while(this.elem[addr] != NULLKEY) {**

**addr = Hash(addr + 1); //开放定址法的线性探测**

**}**

**this.elem[addr] = key;**

**}**

**/\*\***

**\* 查找元素**

**\* @param key**

**\*/**

**public int searchHash(int key) {**

**int addr = Hash(key);**

**while(this.elem[addr] != key) {**

**addr = Hash(addr + 1); //开放定址法的线性探测**

**if(this.elem[addr] == NULLKEY || addr == Hash(key)) { //如果循环回到原点**

**return UNSUCCESS; //说明关键字不存在**

**}else if(this.elem[addr]==key) {**

**return UNSUCCESS;**

**}**

**}**

**return SUCCESS;**

**}**

**/\*\***

**\* 查找元素出现的次数**

**\* @param key**

**\* return cot次数**

**\*/**

**public int searchElemNum(int key) {**

**int cot=0;**

**for(int i=0;i<elem.length;i++) {**

**if(key==elem[i]) {**

**cot++;**

**}**

**}**

**return cot;**

**}**

**public static void main(String[] args) {**

**HashTables h = new HashTables();**

**Scanner input=new Scanner(System.in);**

**System.out.println("请输入10个元素（用空格隔开）：");**

**String[] str=input.nextLine().split(" ");**

**System.out.println("原序列为："+Arrays.toString(str));**

**//依次向散列表中插入元素**

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

**h.insertHash(Integer.valueOf(str[i]));**

**}**

**System.out.println("散列表为："+Arrays.toString(h.elem));**

**List list = new ArrayList();**

**for(int j=0;j<str.length;j++) {**

**if(!list.contains(Integer.valueOf(str[j]))) {**

**list.add(Integer.valueOf(str[j]));**

**}**

**}**

**//查找重复元素出现的次数**

**for (int k=0;k<list.size();k++) {**

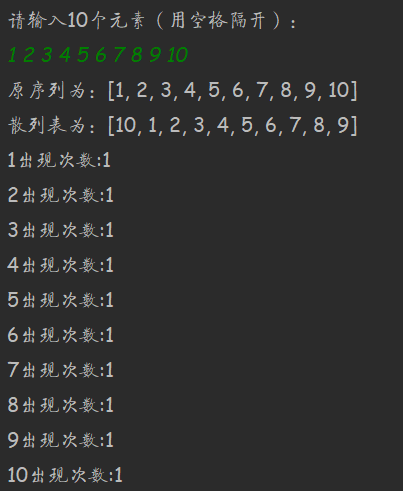
**System.out.println(list.get(k)+"出现次数:"+h.searchElemNum((int) list.get(k)));**

**}**

**input.close();**

**}**

**}**



1. **注意事项**

1. 注意分析和比较同一个问题不同的解决方法的优劣，在什么情况下采取哪种解决方法比较合理。

2. 注意在实现散列查找时散列函数的设计方法和不同的解决冲突的方法对查找效率的影响。