串和数组

一 目的

1、掌握串的存储结构和模式匹配算法；

2、掌握特殊矩阵和稀疏矩阵的压缩存储方法及应用。

二 内容

1、实现串的模式匹配算法。

2、实现稀疏矩阵的三元组表压缩存储方法

3*、三元组表压缩存储方法基础上实现稀疏矩阵的转置，主要涉及矩阵中每一行非零元素的个数的求解方法和转置矩阵的每一行中的第一个元素在其三元组表中的存储位置的计算。

三 设计说明

1. 采用 KMP 方法实现串的模式匹配，KMP 算法是一种改进的字符串匹配算法，关键是利用匹配失败后的信息，尽量减少模式串与主串的匹配次数以达到快速匹配的目的。定义 next 函数来找出下一次目标函数与模式比较的位置.

2. 稀疏矩阵的压缩存储采用三元组的方法实现。其存储规则是：每一个非零元素占一行，每行中包含非零元素所在的行号、列号、非零元素的数值。

四 功能说明

模式串中每个字符的最大真子串构成一个数组，定义为模式串的 next[j]函数。模式串的 next[j]函数定义如下：next[j]函数表示的是模式串 t 中是否存在最大真子串，以及最大真子串的字符个数 k。这里之所以称为最大真子串，是因为：①求出的是所有子串中最大子串；②不允许 k 等于 j。

五 调试分析

1.串的查找操作也称作串的模式匹配操作，模式匹配操作的具体含义是：在主串（也称作目标串）中，从位置 start 开始查找是否存在子串（也称作模式串）。KMP 算法的核心是利用匹配失败后的信息，尽量减少模式串与内容串的匹配次数以达到快速匹配的目的。具体实现就是通过一个 next[]数组来实现，next[]数组本身包含了模式串的局部匹配信息。KMP 算法的时间复杂度 O(m+n)。

六 测试结果

```
    //测试内容串
    String content = "abaabaabbabaaabaabbabaab";

    //测试模式串
    String pattern = "abaabbabaab";
```

```
<已终止> KMP [Java 应用程序] D:\
13
```

```
原矩阵..............
行数：6
列数：7
非零元素个数：6
三元组类：
martrix<1,3>=11.0
martrix<1,5>=17.0
martrix<2,2>=25.0
martrix<4,1>=19.0
martrix<5,4>=37.0
martrix<6,7>=50.0
转置之后的矩阵.........
行数：7
列数：6
非零元素个数：6
三元组类：
martrix<3,1>=11.0
martrix<5,1>=17.0
martrix<2,2>=25.0
martrix<1,4>=19.0
martrix<4,5>=37.0
martrix<7,6>=50.0
```

七 带注释的源代码

串的模式匹配

```
package sj;
public class KMP {
    public static int[] getNext(String pattern) {

        //因为 next 数组下标是从 1 开始的，所以数组长度为字符串长度+1

        int[] next = new int[pattern.length() + 1];
```

```
//数组前两个数字有默认值

//next[0] 默认为 0
next[0] = 0;

//next[1] 默认为 1
next[1] = 1;

//从第二个字符开始计算 next 数组
for (int i = 2; i <= pattern.length(); i++) {

    //获取模式串前缀子串
    String subStr = pattern.substring(0, i);

    //前缀子串的相同前后缀的最大长度
    int max = 0;

    //求前缀子串的相同前后缀的最大长度
    for (int j = 1; j < i; j++) {

        //获得长度为 j 的前缀子串的前缀
        String prefix = subStr.substring(0, j);

        //获得长度为 j 的前缀子串的后缀
        String suffix = subStr.substring(subStr.length() - j);

        //如果前缀子串的前后缀相等（长度为 j）
        if (prefix.equals(suffix)) {

            //更新前缀子串的相同前后缀的最大长度

            //这个肯定会越来越大的，j 会不断++，最后求出最大长度
            max = j;
        }
```

```
            }

            //将求得的模式串的前缀的最大前后缀长度，写入对应下标的 next 数组
            next[i] = max;
        }

        //返回求得的 next 数组
        return next;
    }
    public static int match(String content, String pattern) {
        //如果模式串比内容串还要长
        if (pattern.length() > content.length()) {

            //直接返回-1
            return -1;
        }

        //获取模式串的 next 数组
        int[] next = getNext(pattern);

        //内容串 ci 指针和模式串 pi 指针向前遍历
        for (int ci = 0; ci < content.length(); ci++) {
            for (int pi = 0; pi < pattern.length(); pi++) {
                //如果内容串已经越界了
                if (ci >= content.length()) {
                    //超出范围还没找到，返回-1
                    return -1;
                }
                //当前内容串的比较字符
                char charC = content.charAt(ci);
                //当前模式串的比较字符
                char charP = pattern.charAt(pi);

                //如果两个字符相等
```

```
                if (charC == charP) {

                    //如果匹配到了模式串最后一个字符
                    if (pi == pattern.length() - 1) {

                        //返回索引
                        return ci - pi;
                    //如果还没到模式串最后一个字符
                    } else {
                        //内容串右移一位
                        ci++;
                    }
                //如果两个字符不相等
                } else {
                    //找到内容串需要右移的位数
                    int contentRightOffset = pi - next[pi];
                    //找到内容串的下一个起始坐标
                    int restart = ci - pi + 1 + contentRightOffset;
                    //注意，这里需要-1，应对 for 循环里面的自增
                    ci = restart - 1;
                    //跳出当前模式串的 for 循环，重新匹配模式串
                    break;
                }
            }
        }
        //找完了还没有找到，返回-1
        return -1;
    }
    public static void main(String[] args) {
        //测试内容串
        String content = "abaabaabbabaaabaabbabaab";
        //测试模式串
```

```
        String pattern = "abaabbabaab";
        //输出模式串在内容串的索引号
        System.out.println(match(content, pattern));
    }
}
```

稀疏矩阵的三元组表压缩存储

```
public class Three {

    //行数
    public int row;
    //列数
    public int col;
    //值
    public double value;

    public Three(int row,int col,double value) {
        this.row=row;
        this.col=col;
        this.value=value;
    }
    public Three() {
        this(0,0, 0);
    }

}
package sj;

import java.util.ArrayList;
import java.util.List;

//稀疏矩阵的压缩算法
public class SpanMartrix {

    //行数
    public int rows;
    //列数
```

```
    public int cols;
    //非零元素个数
    public int dNum;

    List<Three> list;

    public SpanMartrix(int max) {
        rows=cols=dNum=0;
        list=new ArrayList<>(max);
    }

    //根据用户传来的三元组类数组来初始化稀疏矩阵
    public void evaluate(int rows,int cols,int dNum,Three three[]) {
        this.rows=rows;
        this.cols=cols;
        this.dNum=dNum;
        for (int i = 0; i < dNum; i++) {
            list.add(i,three[i]);
        }
    }

    //打印稀疏矩阵
    public void printSpanMartrix() {
        System.out.println("行数："+this.rows);

        System.out.println("列数："+this.cols);

        System.out.println("非零元素个数："+this.dNum);

        System.out.println("三元组类：");
        for (int i = 0; i < dNum; i++) {
    System.out.println("martrix<"+list.get(i).row+","+list.get(i).col+">="+
list.get(i).value);
        }
    }

    //矩阵的转置
    public SpanMartrix transport() {
```

```
        SpanMartrix newMartrix=new SpanMartrix(list.size());
        newMartrix.rows=this.cols;
        newMartrix.cols=this.rows;
        newMartrix.dNum=this.dNum;
        for(int i=0;i<dNum;i++) {
            Three three=list.get(i);
            newMartrix.list.add(new Three(three.col,three.row,three.value));
        }

        return newMartrix;
    }
    }
package sj;

public class SpanMain {

    public static void main(String[] args) {
        SpanMartrix mar1=new SpanMartrix(10);
        SpanMartrix mar2;
        Three three[]= new Three[6];
        three[0]= new Three(1,3,11.0);
        three[1]= new Three(1,5,17.0);
        three[2]= new Three(2,2,25.0);
        three[3]= new Three(4,1,19.0);
        three[4]= new Three(5,4,37.0);
        three[5]= new Three(6,7,50.0);

        mar1.evaluate(6, 7, 6, three);

        System.out.println("原矩阵.............");

        mar1.printSpanMartrix();

        System.out.println("转置之后的矩阵.........");

        mar2=mar1.transport();
        mar2.printSpanMartrix();
    }
    }
```