数据结构作业报告

第 2 次

姓名	凌晨
班级	软件 2104 班
学号	2214414320
电话	18025402131
Email	lingchen47@outlook.com
日期	2022-12-22

目录

任务 1：指定的 List ADT 实现

题目：见 pdf

1、使用顺序数组作为存储表示：

数据设计：

```
1.  private int MAXLEN;
2.  private char[] seqList = null;
3.  private int cursor = -1;
4.  private int tail = 0;
```

在使用顺序数组作为存储表示中，设计了以上几个私有变量，下面一一进行说明。
MAXLEN 表示顺序数组的最大储存空间，这是因为顺序数组是依靠数组的存储空间实现的，必须在一开始规定好其数组的大小。
seqList 表示顺序数组，存储字符，初始化为 null。
cursor 表示顺序数组的起始下标，初始化为-1。
tail 表示顺序数组的结尾下标，初始化为 0。

算法设计：

```
1.  public void insert(char newElement) {
2.          //  after the cursor
3.          if (tail>=MAXLEN){
4.              return;
5.          }
6.  //          空数组插入
7.          if (tail==0){
8.              cursor++;
9.              seqList[cursor] = newElement;
10.             tail++;
11.             return;
12.         }
13. //          非空插入
14.         for (int i=tail-1;i>cursor;i--){
15.             seqList[i+1] = seqList[i];
16.         }
17.         cursor++;
18.         seqList[cursor] = newElement;
19.         tail++;
20.     }
```

根据题目中的插入要求，分成了三种基本情况：满数组，空数组，正常插入（非空数组）。

根据要求可知道，要先判断满数组和空数组，若都不是，再进行正常插入。因为是顺序数组，所以说进行插入操作需要复制插入位置之后的所有元素，又因为剩下的操作均在常数项时间内能完成，由此插入操作的时间复杂度为 $O(n)$ 。

```
1.  public void remove() {
2.          tail--;
3.          //        空数组
4.          if (tail <= 0){
5.              tail = 0;
6.              cursor = -1;
7.              seqList = new char[MAXLEN];
8.              return;
9.          }
10.         for(int i=cursor;i<tail;i++){
11.             seqList[i] = seqList[i+1];
12.         }
13.         cursor = cursor%tail;
14.     }
```

删除操作也分为两种基本情况：空数组，正常删除。在删除操作中，要把需要删除的元素后面的元素往前移一位，其它操作均为常数项，因此时间复杂度为 $O(n)$ 。需要注意的是，在删除操作中 cursor = cursor%tail 很重要，这是因为当删除的元素位于数组的 tail 位置时，cursor 需要指向首元素，因此需要上述操作。

```
1.  public void replace(char newElement) {
2.          if (cursor < 0){
3.              return;
4.          }
5.          seqList[cursor] = newElement;
6.      }
```

替换操作比较简单，不进行说明，唯一需要注意的是当空数组时，无需操作，直接返回。时间复杂度为 $O(1)$ 。

```
1.  public void clear() {
2.          seqList = new char[MAXLEN];
3.          cursor = -1;
4.          tail = 0;
5.      }
```

清空操作比较简单，不进行说明，时间复杂度为 O(1) 。

```
1.  public boolean isEmpty() {
2.          if (tail == 0){
3.              return true;
4.          }
5.          return false;
```

```
    }
public boolean isFull() {
        if (tail == seqList.length){
            return true;
        }
        return false;
    }
```

根据指向顺序数组的尾部的变量来判断顺序数组是否为空或者满，时间复杂度为 O(1) 。

```
public boolean gotoBeginning() {
        if (this.isEmpty()){
            return false;
        }
        cursor = 0;
        return true;
    }
public boolean gotoEnd() {
        if (this.isEmpty()){
            return false;
        }
        cursor = tail-1;
        return true;
    }
public boolean gotoNext() {
        if (cursor == tail-1){
            return false;
        }
        cursor++;
        return true;
    }
public boolean gotoPrev() {
        if (cursor <= 0){
            return false;
        }
        cursor--;
        return true;
    }
```

以上方法都是改变 cursor 指向位置，不改变元素，因此处理方法大同小异，需要注意空数组，满数组等特殊情况即可。时间复杂度为 O(1) 。

```
public char getCursor() {
        return seqList[cursor];
    }
```

返回即可，时间复杂度为 O(1) 。

```
public void showStructure(){
    if (tail == 0){
        System.out.printf("Empty list -1%n");
        return;
    }
    for(int i=0;i<tail;i++){
        System.out.printf("%c ",seqList[i]);
    }
    System.out.printf("%d%n",cursor);
}
```

展示顺序数组所含元素以及当前指向元素位置，时间复杂度为 $O(n)$

2、使用单向链表作为存储表示

数据设计：

```
//    成员变量
    public char element = ' ';//选择一个没有意义的字符作为初始化字符
    public LList next = null;

//    类变量
    static public int cursor = -1;   //记录当前元素的位置的前一个位置
    static public LList dummy = new LList();
    static public LList head = dummy;//    初始化为哑节点
    static public LList cur = dummy;//指向当前元素的指针,初始化为哑节点
```

数据分为成员变量和类变量，注释中已说明具体含义。需要重点说明的一点为，我编写的单向链表采用了哑节点的方式，虽然会带来操作上的一些理解困难，但是减少了一些操作的时间。需要和顺序数组区分的是，以链表实现的存储表示不需要提前确定最大容量。

算法设计：

```
    public void insert(char newElement) {
        LList tem = new LList();
        tem.element = newElement;
//        判断是否为空，是的话直接插入
        if (isEmpty()){
            cur.next = tem;
            return;
        }
//        判断是否在尾部，是的话直接插入并且转化 cur
        if (isFull()){
            cur.next.next = tem;
```

```
            cur = cur.next;
            return;
        }
//        后面有元素
        tem.next = cur.next.next;
        cur.next.next = tem;
        cur = cur.next;
    }
```

插入操作分为三种情况：空数组，满数组，正常插入。由于使用了链表结构，插入操作难度大幅降低，只需要把需要插入位置前的元素指向插入元素，插入元素指向插入位置后的元素。因此，时间复杂度为 O(1) 。

```
public void remove() {
        if (isEmpty()){
            return;
        }
        if (isFull()){
            cur.next = null;
            cur = dummy;
            return;
        }
        cur.next = cur.next.next;
    }
```

删除操作与插入操作大同小异，也是三种情况：空数组，满数组，正常删除。但是，由于采用了哑节点技术，使得我们删除操作得到了极大的优化，只需要把 cur 指针往前移即可。因此，时间复杂度为 O(1) 。

```
public void replace(char newElement) {
        if (cur.next == null){
            return;
        }
        cur.next.element = newElement;
    }
```

代替操作不能直接代替，需要先判断 cur.next 是否为 null。因此，时间复杂度为 O(1) 。

```
public void clear() {
        cur =dummy;
        dummy.next = null;
    }
```

清除操作只需要简单地改变两个类变量的值即可。因此，时间复杂度为 O(1) 。

```
public boolean isEmpty() {
        return head.next == null;
    }
```

```
4.  public boolean isFull() {
5.          return cur.next.next == null;
6.      }
```

判断是否为空数组和满数组时，只需要看各个指针的情况即可。因此，时间复杂度为O(1) 。

```
1.  public boolean gotoBeginning() {
2.          if (isEmpty()){
3.              return false;
4.          }
5.          cur = head;
6.          return true;
7.      }
8.  public boolean gotoEnd() {
9.          if (isEmpty()){
10.             return false;
11.         }
12.         while (cur.next.next != null){
13.             cur = cur.next;
14.         }
15.         return true;
16.     }
17. public boolean gotoNext() {
18.         if (isEmpty()){
19.             return false;
20.         }
21.         if (isFull()){
22.             return false;
23.         }
24.         cur = cur.next;
25.         return true;
26.     } public boolean gotoPrev() {
27.         if (isEmpty()){
28.             return false;
29.         }
30.         if (cur == head){
31.             return true;
32.         }
33.         LList tem = head;
34.         while (tem.next != cur){
35.             tem = tem.next;
36.         }
37.         cur = tem;
38.         return true;
```

```
    }
```

以上方法都是改变 cur 的指向位置，不改变元素，因此处理方法大同小异，需要注意空数组，满数组，在第一元素时不能向前移动，在最后一个元素不能向后移动等特殊情况即可。时间复杂度为 O(1) 。

```
public char getCursor() {
        return cur.next.element;
    }
```

返回指向元素，时间复杂度为 O(1) 。

```
public void showStructure() {
        if(isEmpty()){
            System.out.printf("Empty list -1%n");
            return;
        }
        LList tem = head.next;
        while (tem.next != null){
            System.out.printf("%c ",tem.element);
            tem = tem.next;
        }
        cursor = getIntCursor();
        System.out.printf("%c %d%n",tem.element,cursor);
    }
```

展示单向数组所含元素以及当前指向元素位置，cursor 是获取当前指向元素在单向链表的下标，因此可以知道时间复杂度为 O(n)。同时展示元素要一一遍历，时间复杂度为 O(n)。

3、使用双向链表作为存储表示

数据设计：

```
//    成员变量
public char element = ' ';//选择一个没有意义的字符作为初始化字符
public DList next = null;
public DList pre = null;

//    类变量
static public int cursor = -1;   //记录当前元素的位置,无需提前一个位置
static public DList head = null;//    初始化
static public DList cur = head;//指向当前元素的指针
```

双向链表的数据变量分为成员变量和类变量，具体含义在注释中已说明，不赘述。需要注意的是，在双向链表中，由于双向链表的特性，没有采用哑节点，指针指向当前元素的位置。

算法设计：

```
1.  public void insert(char newElement) {
2.      DList tem = new DList();
3.      tem.element = newElement;
4.        判断是否为空，是的话开始创建
5.      if (isEmpty()){
6.          cur = tem;
7.          head =cur;
8.          return;
9.      }
10.     tem.next = cur.next;
11.     tem.pre = cur;
12.     cur.next = tem;
13.     if (tem.next!=null){
14.         tem.next.pre = tem;
15.     }
16.     cur = tem;
17. }
```

插入操作分为三种情况：空数组，正常插入。操作与单向链表类似，但是需要进行 `tem.next!=null` 判断，避免满数组插入时导致空指针错误，总而言之，双链表的插入操作时间复杂度为 O(1) 。

```
1.  public void remove() {
2.      if (isEmpty()){
3.          return;
4.      }
5.      if (cur == head){
6.          cur = head.next;
7.          head = cur;
8.          return;
9.      }
10.     if (isFull()){
11.         cur.pre.next = null;
12.         cur = head;
13.         return;
14.     }
15.     cur.next.pre = cur.pre;
16.     cur = cur.next;
17.     cur.pre.next = cur;
18. }
```

由于没有采用哑节点，因此需要判断的条件相较于单向链表较多，一共有四种情况：空列表，只有一个元素的列表，满列表，正常插入。尽管情况较多，但由于双向链表有指向前一元素的指针，因此删除操作只需要把前后元素“相连”，时间复杂度为 O(1) 。

```
public void replace(char newElement) {
    if (isEmpty()){
        return;
    }
```

替换操作，时间复杂度为 O(1) 。

```
public void clear() {
    head = null;
    cur = head;
}
public boolean isEmpty() {
    return cur == null;
}
public boolean isFull() {
    return cur==null||cur.next==null;
}
```

如上操作，都是对 head，cur 指针进行判断或者赋值，时间复杂度为 O(1) 。

```
public boolean gotoBeginning() {
    if (isEmpty()){
        return false;
    }
    cur = head;
    return true;
}
public boolean gotoEnd() {
    if (isEmpty()){
        return false;
    }
    while (cur.next != null){
        cur = cur.next;
    }
    return true;
}
public boolean gotoNext() {
    if (isEmpty()){
        return false;
    }
    if (isFull()){
        return false;
    }
    cur = cur.next;
    return true;
}
```

```
27. public boolean gotoPrev() {
28.     if (isEmpty()){
29.         return false;
30.     }
31.     if (cur == head){
32.         return true;
33.     }
34.     cur = cur.pre;
35.     return true;
36. }
```

如上操作均为改变 cur 指针位置，只需要注意空数组，满数组等特殊情况即可。时间复杂度为 O(1) 。

```
1.  public char getCursor() {
2.      return cur.element;
3.  }
```

返回 cur 指针当前元素即可，时间复杂度为 O(1) 。

```
1.  public void showStructure() {
2.      if(isEmpty()){
3.          System.out.printf("Empty list -1%n");
4.          return;
5.      }
6.      DList tem = head;
7.      while (tem.next != null){
8.          System.out.printf("%c ",tem.element);
9.          tem = tem.next;
10.     }
11.     cursor = getIntCursor();
12.     System.out.printf("%d%n",cursor);
13. }
```

与单向链表相似，不再赘述，时间复杂度为 O(n)。

通过上述描述，时间复杂度整理成如下表格:

	insert	remove	replace	clear	isEmpty	isFull	gotoBeginning	gotoEnd	gotoNext	gotoPrev	getCursor	showStructure
顺序数组	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(n)
单向链表	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(n)
双向链表	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(n)

测试：

根据题目要求，编写了测试类（见附录），下面就关键代码进行说明:

```
1.  switch (tem){
2.      case '+':
```

```
            tem = row.charAt(pivot++);
            list.insert(tem);
            break;
        case '-':
            list.remove();
            break;
        case '=':
            tem = row.charAt(pivot++);
            list.replace(tem);
            break;
        case '#':
            list.gotoBeginning();
            break;
        case '*':
            list.gotoEnd();
            break;
        case '>':
            list.gotoNext();
            break;
        case '<':
            list.gotoPrev();
            break;
        case '~':
            list.clear();
            break;
        default:
            break;
    }
```

这是测试类最关键的代码，均按照题目要求进行相应的操作，这里的 tem 代表读取 txt 文件中的单个字符。

```
Scanner in = new Scanner(new File("src/test.txt"));
Scanner in2 = new Scanner(new File("src/list_result.txt"));
```

读取相应文件，一边后面的操作和校对答案。

```
myAns = list.toString();
```

myAns 为字符串，由于 showStructure 无返回值，不适合大量校准答案，因此我在顺序数组，单向链表，双向链表都重写了 toString 方法，这样子可以返回 String 类型，适合校准答案，返回的字符串为 showStructure 输出的字符串。

```
boolean equals = myAns.strip().equals(ans.strip());
```

这里使用了 strip()方法，处理字符串后面的空格和换行符。

测试结果：

```
经检验测试结果和标准答案：true
经检验测试结果和标准答案：true
经检验测试结果和标准答案：true
```

三种数据结构均为正确。

总结与收获：

通过上面三种数据结构的编写，我充分理解了顺序数组，单向链表和双向链表的构成，具体实现代码和时间空间的优缺点。

其中最让我感到印象深刻的就是单向链表的哑节点技术的应用，如果让没有听课的我进行单向链表的实现可能就不会采用哑节点技术，这将导致单向链表的删除操作的时间复杂度增加到 O(n)，这从性能的变化来讲是十分差劲的，因此一个小小的编程思想的影响导致时间复杂度大大降低，让我印象深刻。但是，我也注意到了，当我为了实现哑节点，增加了不少常系数时间的判断，这可能在数据量较少的情况下不如不使用哑节点。

然后，我还想说，双向链表的设计存在十分不足，我没有使用任何技巧，完全采用了个人的理解，这是因为作业的时间紧迫导致的，我现在想来也许可以增加两个无用的节点，大大简化判断流程和代码编写，希望未来有时间可以实现。

任务 2：栈和递归之间的关系

题目：略

1、使用递归思想，编写一个函数 permutationByRecursion，该函数用来生成给定的字符串的全排列结果。

数据说明：

```
static String str;
static char[] res;
```

str 为录入的字符串，但是不方便处理，更改其中的字符。

res 为结果字符数组，因为是字符数组方便输出，更改字符，交换字符。

算法说明：

```
res = new char[str.length()];
for(int i = 0;i<str.length();i++){
    res[i] = str.charAt(i);
}
```

把字符串转换为字符数组，方便后续操作。

```
public static void permutationByRecursion(int left,int right){
    if (left == right){
        System.out.print(res);
        System.out.print(" ");
```

```
5.      }
6.      for(int i=left;i<=right;i++){
7.          swap(res,left,i);
8.             //进入递归
9.          permutationByRecursion(left+1,right);
10.         swap(res,left,i);
11.     }
12. }
```

1、基准情况就是当 left == right，这个时候输出 res，即一种情况。
2、输入时 left=0，right=数组长度-1，因此第一次的 for 循环作用就是遍历数组，这保证了后面输出结果时不遗漏。每次操作如下：将 i 与 left 交换，然后再将剩下的 left+1 到 right 递归全排序，递归完成后再交换回来。在递归时会往基准情况靠近，因此避免了死递归，而且当到达基准情况时就会输出一种不重复全排序的情况，这是因为递归完成交换回来的操作。
3、全排序的时间复杂度明显为 O(n!)，而且无任何优化空间。

测试结果：

1、测试用例"abd"，结果如下：
acd adc cad cda dca dac

2、测试用例"abcde"，结果如下：
abcde abced abdce abdec abedc abecd acbde acbed acdbe acdeb acedb acebd adcbe adceb adbce adbec adebc adecb aecdb aecbd aedcb aedbc aebdc aebcd bacde baced badce badec baedc baecd bcade bcaed bcdae bcdea bceda bcead bdcae bdcea bdace bdaec bdeac bdeca becda becad bedca bedac beadc beacd cbade cbaed cbdae cbdea cbeda cbead cabde cabed cadbe cadeb caedb caebd cdabe cdaeb cdbae cdbea cdeba cdeab ceadb ceabd cedab cedba cebda cebad dbcae dbcea dbace dbaec dbeac dbeca dcbae dcbea dcabe dcaeb dceab dceba dacbe daceb dabce dabec daebc daecb decab decba deacb deabc debac debca ebcda ebcad ebdca ebdac ebadc ebacd ecbda ecbad ecdba ecdab ecadb ecabd edcba edcab edbca edbac edabc edacb eacdb eacbd eadcb eadbc eabdc eabcd

2、使用栈数据结构，将 1 中编写的算法转换成非递归函数

分析：本题与上课讲的实例汉诺塔基本一致，上课讲的汉诺塔是第一步将 A 柱上的 n-1 个盘子借助 C 柱移向 B 柱，第二步将 A 柱上仅剩的最后一个盘子移向 C 柱，第三步将 B 柱上的 n-1 个盘子借助 A 柱移向 C 柱，当改为栈结构时，需要再创建一个辅助类。但是，由于本题的特殊性，即字符串的全排序就是数字的全排序，可以进一步转化为数组下标的全排序，这样省去了创建一个类的工作，下面介绍“取巧”做法。

数据结构：

由于题目要求，我们创建了属于自己的栈：

```
public class MyStack {
    public int[] listArr;
    private int tail = -1;//不仅是指向尾部的指针，也代表了元素大小
    public MyStack()
    public MyStack(int MAX)
    public void push(int t)
    public int pop()
    public boolean isHas(int x)
    public boolean isEmpty()
    public void printAns(String s)
}
```

具体的实现过程不再给出，详细的实现过程可以查看附录代码，我详细说明一下变量含义和方法参数及返回值和作用。

1、变量。有 `int[] listArr` 这个是数组，用来存储数据，充当栈。`int tail` 如注释所描述，不赘述。

2、各个方法。pop 是出栈并且返回整数，push 是入栈，isHas 是判断栈中是否有参数 x，isEmpty 是判断栈是否为空，空返回 true，反之。printAns 是根据栈中的数字按照不同顺序输出字符串 s。

算法设计：

下面给出核心代码：

```
while (!myStack.isEmpty()){
    int i = myStack.pop() + 1;//加一是为了与其位置匹配
    while (i < s.length()){
        if (!myStack.isHas(i)){
            myStack.push(i);
            //寻找未进栈的元素进栈
            for (int j = 0; j < s.length(); j++){
                if (!myStack.isHas(j)){
                    myStack.push(j);
                }
            }
            myStack.printAns(s);
            break;
        }
        i++;
    }
}
```

这是最关键的代码段，其核心作用就是将栈中数字的排序从小到大改为从大到小，而且每当出现一种新的排序时就输出字符串。

现在来具体讲解。

1、出栈元素，要加 1 是为了与其位置匹配。

2、进行循环，判断栈中是否含有 i，若不含有则压入栈，注意我们虽然压入的都是 i，但是

i 已经进行了+1 操作，所以实现了从大到小的变换。接着，寻找未进栈的元素并且将其压入栈，注意这里要从 0 到 n，因为这样子保证了从小到大压入，保证了操作的一致性。当全部压入后，则找到了一个全新的排序，输出，停止循环。
3、若栈中含有 i，则说明出栈元素前的栈中存在一个比出栈元素大一的元素（有点拗口），则通过不断地 i++把这个 i 排除掉，然后进行下一次的 `while (!myStack.isEmpty())`循环操作。
总的来说，该算法实现了数字排序从小到大改为从大到小。由于我们录入数据是 0-n，因此在排序过程中一定实现了全排序。

测试：
测试类不再给出，较简单。

测试结果：

测试用例“abc”，结果如下：
abc acb bac bca cab cba
测试用例“abcd”，结果如下：
abcd abdc acbd acdb adbc adcb bacd badc bcad bcda bdac bdca cabd cadb cbad cbda cdab cdba dabc dacb dbac dbca dcab dcba

变形 1：当字符串中出现相同字符时，只给出完全不一样的排列组合

设计：数据设计，算法设计与 1 中的递归设计基本一致，只需要在递归之前加入相同字符不交换递归即可，如下代码。

```
1.  if (res[left]!=res[i]||i==left){
2.      swap(res,left,i);
3.      permutationByRecursion(left+1,right);
4.      swap(res,left,i);
5.  }
```

测试：

1、测试用例“aac”，结果如下：
aac aca caa
2、测试用例“aaaa”，结果如下：
aaaa
3、测试用例“aabc”，结果如下：
aabc aacb abac abca acba acab baac baca bcaa caba caab cbaa
（有些测试用例输出过大，附录和这里都不在给出，可以通过测试类测试）

变形 2：输出长度为 n 的字符串中取 k 个字符构成的所有全排列。

分析：因为在 1 递归中，直接定好了 char[] res 的长度，导致只能输出 n 个字符，而题目要求输出 k 个因此需要在后续进行修改，增加新的变量。同时，1 中的递归思想和方法大体

不变，只需要对具体实现过程进行小幅修改，改变递归基准条件，使其能取 k 个字符后输出而且保证是全排序。

数据设计：

```
static String str;
static char[] strArr;
static char[] res;
```

str 还是录入的字符串
strArr 是将录入的字符串转为字符数组
res 和 1 递归中的 res 作用一致，不过长度不再为 n，而为 k

算法设计：

```
    private static void permutationByRecursion(int k){
//        递归条件
        if (k==0){
            System.out.print(res);
            System.out.print(" ");
            return;
        }
//        把不重复字符加入(十分暴力^_^)
        for(char strArr:strArr){
            if (!has(res,strArr)){
                res[res.length-k] = strArr;
            } else{
                continue;
            }
//            进入递归
            permutationByRecursion(k-1);
//            重新置为表示字符
            res[res.length - k] = ' ';
        }
    }
```

1、基准条件改为 k==0，这与 1 递归中的 left==right 的设计思想是一模一样的；
2、第二步，把不重复的字符加入。has(res,strArr)这个函数是判断 res 字符数组中是否有与 strArr 字符一致的字符，若无，加入到 res 数组里面；若有，则忽略，继续。然后进入递归。最后再将其置为初始化的字符‘ ’。上述步骤与 1 递归中的先交换再递归再交换的核心思想是一模一样的，因此可以保证该算法的正确性，同时可以知道该算法的前置条件为一为无重复字符元素，二为‘ ’字符不参与排序。
3、总的来说，这个算法虽然具体实现过程相差 1 递归的程序有相差，但是，核心思想不变，都是先提取一个字符，让剩下的 n-1 个字符进行递归，当剩余元素为 0 时，达到基准情况返回一种全排序结果，而且由于 for 循环整个数组和递归完后的“善尾工作”——再初始化或者再交换，可以保证全排序的结果不重复不遗漏。
4、时间复杂度为题目中公式所示，证明略。

测试：

测试用例“abcd” 2， 结果如下：
ab ac ad ba bc bd ca cb cd da db dc

测试用例“abcde” 4 结果如下：
abcd abce abdc abde abec abed acbd acbe acdb acde aceb aced adbc adbe adcb adce adeb adec aebc aebd aecb aecd aedb aedc bacd bace badc bade baec baed bcad bcae bcda bcde bcea bced bdac bdae bdca bdce bdea bdec beac bead beca becd beda bedc cabd cabe cadb cade caeb caed cbad cbae cbda cbde cbea cbed cdab cdae cdba cdbe cdea cdeb ceab cead ceba cebd ceda cedb dabc dabe dacb dace daeb daec dbac dbae dbca dbce dbea dbec dcab dcae dcba dcbe dcea dceb deab deac deba debc deca decb eabc eabd eacb eacd eadb eadc ebac ebad ebca ebcd ebda ebdc ecab ecad ecba ecbd ecda ecdb edab edac edba edbc edca edcb

有些用例的输出过大，附录和这里都不在给出，可以尝试使用附录的测试类测试。

总结与收获：
在本次任务中，我编写了字符串的全排序问题，并且进行相应的改善，这是十分有趣的。下面我对三次改善谈谈自己的收获。
首先，就是把递归改成了栈。在这一问题上，我花费了不小的力气，说实话，我上课听懂了汉诺塔，可后面自己写的时候，常常看着递归的代码想不出来怎么改成栈，不知道怎么组织类的结构来进行更改，因此我放弃了老师上课讲的汉诺塔改为栈的方法，当然我希望老师能把上课讲的汉诺塔改成栈的具体代码发给我，嘿嘿。言归正传，我在网上找到了更巧妙地改为栈的方法，加以自己的理解写出了如上代码，在编写过程中，我还看了几次网上的解释才理解通透，可以说这个递归改成栈是十分困难的，今后要多下功夫了。
然后是两次更改递归条件的问题。第一个主要判断是否相同，不相同再调用递归，就达到了问题要求，一样的输出只输出一次。第二个，由于一开始写程序就定死了输出个数，因此我按照相同的逻辑把具体实现过程进行了更改，改成了可以输出个数变化的情况。

任务 3：创建一个可自动调整空间大小的 Queue 数据结构

题目：略

数据设计：

```
1.  private int maxSize = 2;
2.  private int front = 0;
3.  private int rear = 0;
4.  private T[] listArray =(T[]) new Object[2];
```

下面对其变量进行说明。
maxSize 为当前队列所能容纳的最多元素，这里需要说明因为采用了循环数组，因此能存储的元素数量总是小于 maxSize，等于 maxSize-1，这对于后面的程序编写会有一定的影响；
front 可以理解为队头指针，为了判断满队列，空队列，需要队头指针的前一位才是储存元素的首元素；
rear 可以理解为队尾指针；
listArray 就是数组，大小初始化为 2。

算法设计：

```
1.  public ResizingQueue(){}
```

构造方法，无需指定大小。

```
1.  public void enqueue(T element) throws IllegalArgumentException{
2.      if (element == null){
3.          throw new IllegalArgumentException();
4.      }
5.      if (isExpand()){
6.            expand()方法改变了大小，改变了 front,rear,maxSize 数值
7.          listArray = expand(maxSize,(maxSize-1)*2+1);
8.      }
9.      rear = (rear+1)%maxSize;
10.     listArray[rear] = element;
11. }
```

enqueue 方法将元素 element 入队，如果队列满，则需要完成空间大小的调整。使用 isExpand()方法判断是否为队列满，如果满队列，使用 expand 方法进行扩充，其中 expand 的两个参数为变换前的最大容量和变化后的最大容量，该方法返回 T 数组。扩充完毕后，再进行入队操作。

```
1.  public T dequeue() throws NoSuchElementException{
2.      if (isEmpty()){
3.          throw new NoSuchElementException();
4.      }
5.      listArray[front] = null;
6.      front = (front+1)%maxSize;
7.      T res =listArray[front];
8.      if (isShrink()){
9.          listArray = shrink(maxSize,(maxSize-1)/2);
10.     }
11.     return res;
12. }
```

dequeue()从队列中将队头元素删除并返回，如果队列的元素个数是当前容量的 1/4，那么完成空间大小的调整。首先进行空数组判断，若为空数组，抛出 NoSuchElementException。若不为空数组，则先进行删除，再判断是否需要收缩数组。isShrink()判断队列的元素个数是当前容量的 1/4，若满足，则调用 shrink 方法，shrink 方法的第一个参数为当前最大容量，

后一个为收缩完成后的最大容量，返回值为 T 数组。

```
public int size(){
    if (rear>=front){
        return rear-front;
    }
    return maxSize-front+rear;
}
```

由于使用了循环数组，因此通过 front 和 rear 的两种情况的相对位置即可判断数组容量。

```
public String toString() {
    String res = "";
    res += "[";
      不大于 20 的情况
    if (this.size()<=20){
        if (rear>=front){
            for (int i = front+1;i<=rear;i++){
                res = res + listArray[i] + " ";
            }
        }
        else {
            for (int i = front+1;i<maxSize;i++){
                res = res + listArray[i] + " ";
            }
            for (int i = 0;i<=rear;i++){
                res = res + listArray[i] + " ";
            }
        }
    }
      大于 20 的情况
    else {
        int tem = (front+1)%maxSize;
        int count = 0;
          去前五个元素
        do{
            res = res + listArray[tem] + " ";
            tem = (tem+1)%maxSize;
            count++;
        }
        while (count<5);
        res += " ... ";
          取后五个元素
        while (count<this.size()-5){
            tem = (tem+1)%maxSize;
```

```
            count++;
        }
        while (count<this.size()){
            res = res + listArray[tem] + " ";
            tem = (tem+1)%maxSize;
            count++;
        }
    }
    res = res.strip()+"]";
    res += "\nelements: " + this.size() + " size:"+(this.maxSize-1);
    return res;
}
```

虽然看上去比较“冗杂”，但都是针对不同情况的重复性的工作，按照题目要求分为不大于 20 的表示方式和大于 20 的表示方式，由于采用了循环数组，因此必须判断 rear 和 front 的相对位置。

测试：

编写了测试类（见附录），下面对关键代码进行说明：

```
Scanner in = new Scanner(new File("src/test5000.txt"));
Scanner in2 = new Scanner(new File("src/ result5000.txt"));
```

读取 txt 文件，便于后续操作。

```
if (in.hasNextInt()){
    nextInt = in.nextInt();
//  System.out.printf("读取到的数字为：%d",nextInt);
    resizingQueue.enqueue(nextInt);
}
```

这是当读取到数字时，采用入队操作。

```
if (nextChar == '-'){
    resizingQueue.dequeue();
}
if (nextChar == '?'){
    System.out.println(resizingQueue.toString());
    String tem1 = in2.nextLine();
    String tem2 = in2.nextLine();
    String res = tem1 + "\n" + tem2;
    System.out.println("res 为："+res);
    boolean temVertify = res.strip().equals(resizingQueue.toString().strip());
    System.out.println("比较结果为："+temVertify);
    vertify = vertify && temVertify;
```

```
13. }
```

当读取到'-'时，使用出队操作。当读取到'?'，需要调用 toString 方法，同时与结果进行比较。

测试结果：

```
1.  经检验输出的字符串与结果符合判定：true
2.  经检验输出的字符串与结果符合判定：true
```

对 result1000.txt 和 result5000.txt 分别调用测试类，结果如上。

总结与收获：

队列的编写有了前面顺序数组，单向链表，双向链表代码编写的经验，写起来速度了不少，遇到的困难和小问题都在调试的过程中解决了，总体来说，编程思想和前面三个有相似之处。

任务 4：基数排序

题目： 略

1、当数据序列是整数类型的数据的时候，数据序列中每个数据的位数不要求等宽

数据设计：

```
1.  static ResizingQueue[] resizingQueue = new ResizingQueue[10];
```

数组大小为 10 是因为数字是 0-9。

算法设计：

```
1.      static void sort(int[] arr,int len){
2.  //          初始化队列
3.          for(int i=0;i<resizingQueue.length;i++){
4.              resizingQueue[i] = new ResizingQueue<Integer>();
5.          }
6.  //          变量+初始化
7.          boolean flag = true;
8.          int modNum = 0;
9.          int tem=0;
10. //          一位一位操作，直到最长的数字结束
11.         while (flag){
12.             modNum++;
13. //              入队，将数组数字按照顺序入队
14.             for (int i=0;i<len;i++){
15.                 int pivot = mod(arr[i],modNum);
16.                 tem = Math.max(tem,pivot);
```

```
                resizingQueue[pivot].enqueue(arr[i]);
                flag = (tem != 0);
            }
            tem = 0;
            int arrPivot = 0;
//            出队，重新给数组赋值
            for (int i=0;i<resizingQueue.length;i++){
                while (!resizingQueue[i].isEmpty()){
                    arr[arrPivot++] = (int)resizingQueue[i].dequeue();
                }
            }
        }
//        输出
        for (int i=0;i<len;i++){
            System.out.printf("%d ",arr[i]);
        }
    }
```

注释已经有部分讲解，下面进行更详细的讲解。

1、sort 方法的第一个参数为数组，第二个参数为数组的长度。

2、定义了几个变量，进行说明。flag 判断是否继续进行排序操作,当所有数字取模均为 0 时就不再继续；modNum 记录取模次数，为 mod 方法的第二个参数；tem 仅为辅助变量，改变 flag 的值。

3、入队操作，按照数字的每位数取模后的大小进入相应的队列数组中。

4、出队操作，按照顺序从队列数组的第 0 位到第 9 位出队，由于是队列，遵守先进先出原则，因此满足先后顺序。

5、当所有数字取模均为 0 时，数组有序，完成排序。由于数字量远远大于桶的数量（队列数组大小），因此时间复杂度为 O（n）。

测试：

编写了主函数作为测试（见附录），主要思想为把文件中数字放入数组中，再调用 sort 方法。下面进行主要代码分析。

```
while (in.hasNextInt()){
    int tem = in.nextInt();
    System.out.println(tem);
    arr[pivot++] = tem;
}
```

当存在下一个数字时，存入数组。记录位置。

测试结果：

见附录

2、当数据序列是字符串类型的数据的时候，数据序列中每个字符串都是等宽的

数据设计：

```
1.  static ResizingQueue[] resizingQueue = new ResizingQueue[26];
```

数组大小为 26 是因为字母是 a-z。由于题目未明确说明，我把大写字母和小写字母视为一致。

算法设计：

```
1.      static void sort(String[] arr,int len){
2.  //          初始化队列数组
3.          for(int i=0;i<resizingQueue.length;i++){
4.              resizingQueue[i] = new ResizingQueue<String>();
5.          }
6.  //          因为等宽，所以取第一个字符串长度作为长度
7.          int stringLen = arr[0].length();
8.          int tem=0;
9.          for (int k=0;k<stringLen;k++){
10. //              入队操作
11.             for (int i=0;i<len;i++){
12.                 int pivot = arr[i].toLowerCase().charAt(stringLen-k-
    1)-'a';
13.                 resizingQueue[pivot].enqueue(arr[i]);
14.             }
15.             tem = 0;
16.             int arrPivot = 0;
17. //              出队操作
18.             for (int i=0;i<resizingQueue.length;i++){
19.                 while (!resizingQueue[i].isEmpty()){
20.                     arr[arrPivot++] = (String) resizingQueue[i].deque
    ue();
21.                 }
22.             }
23.         }
24.         for (int i=0;i<len;i++){
25.             System.out.printf(arr[i]+" ");
26.         }
27.     }
```

sort 方法的参数，步骤和核心思想与数字的基数排序基本一致，不再赘述，下面讲几个相异点：

1、由于字符串长度一致，因此不需要数字基数排序中的 flag。

2、由于不区分大小写字母，因此先取小写（toLowerCase()的方法）再比较。

3、数字是依靠模运算取位数，字符串是运用 charAt（）方法取字符串的单个字符。

测试：

编写了主函数作为测试，核心思想为把文件中字符串放入数组中，再调用 sort 方法。下面进行关键代码分析。

```
while (in.hasNext()){
    arr[pivot++] = in.next();
}
```

当存在下一个字符串时，把字符串存入数组，pivot 记录存入下标。

测试结果：

见附录

总结与收获：

基数排序还是基于桶排序，按照从低位到高位依次桶排序，最后输出结果即可，可以注意到桶的个数，位数的个数，数据量共同决定的时间复杂度，这与课上学习的一致，强化了学习结果。

附录

任务一

```
public class SeqList implements List{
    /*        这个是顺序数组实现的数据结构      */
    private int MAXLEN;
    private char[] seqList = null;
    private int cursor = -1;
    private int tail = 0;

//  构造函数
    public SeqList(){
        this(10000);
    }
    public SeqList(int MAXLEN){
        this.MAXLEN = MAXLEN;
        seqList = new char[MAXLEN];
    }

    @Override
    public void insert(char newElement) {
        //  after the cursor
        if (tail>=MAXLEN){
            return;
        }
```

```
23. //          空数组插入
24.         if (tail==0){
25.             cursor++;
26.             seqList[cursor] = newElement;
27.             tail++;
28.             return;
29.         }
30. //          非空插入
31.         for (int i=tail-1;i>cursor;i--){
32.             seqList[i+1] = seqList[i];
33.         }
34.         cursor++;
35.         seqList[cursor] = newElement;
36.         tail++;
37.     }
38.
39.     @Override
40.     public void remove() {
41.         tail--;
42.         //          空数组
43.         if (tail <= 0){
44.             tail = 0;
45.             cursor = -1;
46.             seqList = new char[MAXLEN];
47.             return;
48.         }
49.         for(int i=cursor;i<tail;i++){
50.             seqList[i] = seqList[i+1];
51.         }
52.         cursor = cursor%tail;
53.     }
54.
55.     @Override
56.     public void replace(char newElement) {
57.         if (cursor < 0){
58.             return;
59.         }
60.         seqList[cursor] = newElement;
61.     }
62.
63.     @Override
64.     public void clear() {
65.         seqList = new char[MAXLEN];
66.         cursor = -1;
```

```
67.         tail = 0;
68.     }
69.
70.     @Override
71.     public boolean isEmpty() {
72.         if (tail == 0){
73.             return true;
74.         }
75.         return false;
76.     }
77.
78.     @Override
79.     public boolean isFull() {
80.         if (tail == seqList.length){
81.             return true;
82.         }
83.         return false;
84.     }
85.
86.     @Override
87.     public boolean gotoBeginning() {
88.         if (this.isEmpty()){
89.             return false;
90.         }
91.         cursor = 0;
92.         return true;
93.     }
94.
95.     @Override
96.     public boolean gotoEnd() {
97.         if (this.isEmpty()){
98.             return false;
99.         }
100.            cursor = tail-1;
101.            return true;
102.        }
103.
104.        @Override
105.        public boolean gotoNext() {
106.            if (cursor == tail-1){
107.                return false;
108.            }
109.            cursor++;
110.            return true;
```

```
111.        }
112.
113.        @Override
114.        public boolean gotoPrev() {
115.            if (cursor <= 0){
116.                return false;
117.            }
118.            cursor--;
119.            return true;
120.        }
121.
122.        @Override
123.        public char getCursor() {
124.            return seqList[cursor];
125.        }
126.
127.        @Override
128.        public void showStructure(){
129.            if (tail == 0){
130.                System.out.printf("Empty list -1%n");
131.                return;
132.            }
133.            for(int i=0;i<tail;i++){
134.                System.out.printf("%c ",seqList[i]);
135.            }
136.            System.out.printf("%d%n",cursor);
137.        }
138.
139.        @Override
140.        public String toString() {
141.            return myResult();
142.        }
143.
144.        public String myResult(){
145.            String myAns = "";
146.            if (tail == 0){
147.                myAns = "Empty list -1 ";
148.                return myAns;
149.            }
150.            for(int i=0;i<tail;i++){
151.                myAns += seqList[i];
152.                myAns += " ";
153.            }
154.            myAns += cursor;
```

```
            myAns += " ";
            return myAns;
        }
    }
```

```
public class LList implements List {

//    成员变量
    public char element = ' ';//选择一个没有意义的字符作为初始化字符
    public LList next = null;

//    类变量
    static public int cursor = -1;   //记录当前元素的位置的前一个位置
    static public LList dummy = new LList();
    static public LList head = dummy;//    初始化为哑节点
    static public LList cur = dummy;//指向当前元素的指针,初始化为哑节点

    public LList(){

    }
    @Override
    public void insert(char newElement) {
        LList tem = new LList();
        tem.element = newElement;
//        判断是否为空，是的话直接插入
        if (isEmpty()){
            cur.next = tem;
            return;
        }
//        判断是否在尾部，是的话直接插入并且转化 cur
        if (isFull()){
            cur.next.next = tem;
            cur = cur.next;
            return;
        }
//        后面有元素
        tem.next = cur.next.next;
        cur.next.next = tem;
        cur = cur.next;
    }

    @Override
    public void remove() {
```

```
39.         if (isEmpty()){
40.             return;
41.         }
42.         if (isFull()){
43.             cur.next = null;
44.             cur = dummy;
45.             return;
46.         }
47.         cur.next = cur.next.next;
48.     }
49.
50.     @Override
51.     public void replace(char newElement) {
52.         if (cur.next == null){
53.             return;
54.         }
55.         cur.next.element = newElement;
56.     }
57.
58.     @Override
59.     public void clear() {
60.         cur =dummy;
61.         dummy.next = null;
62.     }
63.
64.     @Override
65.     public boolean isEmpty() {
66.         return head.next == null;
67.     }
68.
69.     @Override
70.     public boolean isFull() {
71.         return cur.next.next == null;
72.     }
73.
74.     @Override
75.     public boolean gotoBeginning() {
76.         if (isEmpty()){
77.             return false;
78.         }
79.         cur = head;
80.         return true;
81.     }
82.
```

```
83.     @Override
84.     public boolean gotoEnd() {
85.         if (isEmpty()){
86.             return false;
87.         }
88.         while (cur.next.next != null){
89.             cur = cur.next;
90.         }
91.         return true;
92.     }
93.
94.     @Override
95.     public boolean gotoNext() {
96.         if (isEmpty()){
97.             return false;
98.         }
99.         if (isFull()){
100.                return false;
101.            }
102.            cur = cur.next;
103.            return true;
104.        }
105.
106.        @Override
107.        public boolean gotoPrev() {
108.            if (isEmpty()){
109.                return false;
110.            }
111.            if (cur == head){
112.                return true;
113.            }
114.            LList tem = head;
115.            while (tem.next != cur){
116.                tem = tem.next;
117.            }
118.            cur = tem;
119.            return true;
120.        }
121.
122.        @Override
123.        public char getCursor() {
124.            return cur.next.element;
125.        }
126.
```

```
127.     @Override
128.     public void showStructure() {
129.         if(isEmpty()){
130.             System.out.printf("Empty list -1%n");
131.             return;
132.         }
133.         LList tem = head.next;
134.         while (tem.next != null){
135.             System.out.printf("%c ",tem.element);
136.             tem = tem.next;
137.         }
138.         cursor = getIntCursor();
139.         System.out.printf("%c %d%n",tem.element,cursor);
140.     }
141.
142.     private String myRes(){
143.         if(isEmpty()){
144.             return "Empty list -1";
145.         }
146.         String myRes = "";
147.         LList tem = head.next;
148.         while (tem.next != null){
149.             myRes = myRes + " " + tem.element;
150.             tem = tem.next;
151.         }
152.         cursor = getIntCursor();
153.         myRes= myRes+" "+tem.element+" "+cursor;
154.         return myRes;
155.     }
156.
157.     @Override
158.     public String toString() {
159.         return myRes();
160.     }
161.
162.     private int getIntCursor(){
163.         if (isEmpty()){
164.             return -1;
165.         }
166.         LList tem = head;
167.         int intCursor = 0;
168.         while (tem.next!=cur.next){
169.             intCursor++;
170.             tem = tem.next;
```

```
            }
            return intCursor;
        }
    }
```

```
public class DList implements List {
    //    成员变量
    public char element = ' ';//选择一个没有意义的字符作为初始化字符
    public DList next = null;
    public DList pre = null;

    //    类变量
    static public int cursor = -1;   //记录当前元素的位置,无需提前一个位置
    static public DList head = null;//    初始化
    static public DList cur = head;//指向当前元素的指针

    public DList(){
    }
    @Override
    public void insert(char newElement) {
        DList tem = new DList();
        tem.element = newElement;
//        判断是否为空，是的话开始创建
        if (isEmpty()){
            cur = tem;
            head =cur;
            return;
        }
        tem.next = cur.next;
        tem.pre = cur;
        cur.next = tem;
        if (tem.next!=null){
            tem.next.pre = tem;
        }
        cur = tem;
    }

    @Override
    public void remove() {
        if (isEmpty()){
            return;
        }
        if (cur == head){
```

```
39.             cur = head.next;
40.             head = cur;
41.             return;
42.         }
43.         if (isFull()){
44.             cur.pre.next = null;
45.             cur = head;
46.             return;
47.         }
48.         cur.next.pre = cur.pre;
49.         cur = cur.next;
50.         cur.pre.next = cur;
51.     }
52.
53.     @Override
54.     public void replace(char newElement) {
55.         if (isEmpty()){
56.             return;
57.         }
58.         cur.element = newElement;
59.     }
60.
61.     @Override
62.     public void clear() {
63.         head = null;
64.         cur = head;
65.     }
66.
67.     @Override
68.     public boolean isEmpty() {
69.         return cur == null;
70.     }
71.
72.     @Override
73.     public boolean isFull() {
74.         return cur==null||cur.next==null;
75.     }
76.
77.     @Override
78.     public boolean gotoBeginning() {
79.         if (isEmpty()){
80.             return false;
81.         }
82.         cur = head;
```

```
83.         return true;
84.     }
85.
86.     @Override
87.     public boolean gotoEnd() {
88.         if (isEmpty()){
89.             return false;
90.         }
91.         while (cur.next != null){
92.             cur = cur.next;
93.         }
94.         return true;
95.     }
96.
97.     @Override
98.     public boolean gotoNext() {
99.         if (isEmpty()){
100.             return false;
101.         }
102.         if (isFull()){
103.             return false;
104.         }
105.         cur = cur.next;
106.         return true;
107.     }
108.
109.     @Override
110.     public boolean gotoPrev() {
111.         if (isEmpty()){
112.             return false;
113.         }
114.         if (cur == head){
115.             return true;
116.         }
117.         cur = cur.pre;
118.         return true;
119.     }
120.
121.     @Override
122.     public char getCursor() {
123.         return cur.element;
124.     }
125.
126.     @Override
```

```
127.      public void showStructure() {
128.          if(isEmpty()){
129.              System.out.printf("Empty list -1%n");
130.              return;
131.          }
132.          DList tem = head;
133.          while (tem.next != null){
134.              System.out.printf("%c ",tem.element);
135.              tem = tem.next;
136.          }
137.          cursor = getIntCursor();
138.          System.out.printf("%d%n",cursor);
139.      }
140.
141.      private String myRes(){
142.          if(isEmpty()){
143.              return "Empty list -1";
144.          }
145.          String myRes = "";
146.          DList tem = head;
147.          while (tem.next != null){
148.              myRes = myRes + " " + tem.element;
149.              tem = tem.next;
150.          }
151.          cursor = getIntCursor();
152.          myRes= myRes+" "+tem.element+" "+cursor;
153.          return myRes;
154.      }
155.
156.      @Override
157.      public String toString() {
158.          return myRes();
159.      }
160.
161.      private int getIntCursor(){
162.          if (isEmpty()){
163.              return -1;
164.          }
165.          DList tem = head;
166.          int intCursor = 0;
167.          while (tem != cur){
168.              intCursor++;
169.              tem = tem.next;
170.          }
```

```
            return intCursor;
        }
    }
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFile {
    public static void readFile() throws FileNotFoundException {
//        List list = new SeqList();
//        List list = new LList();
        List list = new DList();
        Scanner in = new Scanner(new File("src/test.txt"));
        Scanner in2 = new Scanner(new File("src/list_result.txt"));
        String row;
        String ans;
        String myAns;
        boolean vertify = true;
        while (in.hasNextLine() && in2.hasNextLine()){
            row = in.nextLine();
            ans = in2.nextLine();
            int pivot = 0;
            char tem;
            while(pivot!=row.length()){
                tem = row.charAt(pivot++);
                switch (tem){
                    case '+':
                        tem = row.charAt(pivot++);
                        list.insert(tem);
                        break;
                    case '-':
                        list.remove();
                        break;
                    case '=':
                        tem = row.charAt(pivot++);
                        list.replace(tem);
                        break;
                    case '#':
                        list.gotoBeginning();
                        break;
                    case '*':
                        list.gotoEnd();
                        break;
```

```
                    case '>':
                        list.gotoNext();
                        break;
                    case '<':
                        list.gotoPrev();
                        break;
                    case '~':
                        list.clear();
                        break;
                    default:
                        break;
                }
            }
            myAns = list.toString();
            System.out.println(myAns.strip());
            System.out.println(ans.strip());
            boolean equals = myAns.strip().equals(ans.strip());
            System.out.println(equals);
            vertify = vertify&& equals;
        }
        System.out.printf("经检验测试结果和标准答案：%b",vertify);
    }

    public static void main(String[] args) throws FileNotFoundExcepti
on {
        readFile();
    }
}
```

任务二

问题一第一问——递归版

```
public class PermutationByRecursion {
    static String str;
    static char[] res;
//    初始化函数
    public static void permutationByRecursion(String s){
        str = s;
        res = new char[str.length()];
        for(int i = 0;i<str.length();i++){
            res[i] = str.charAt(i);
        }
        permutationByRecursion(0, res.length-1);
    }
```

```
    public static void permutationByRecursion(int left,int right){
        if (left == right){
            System.out.print(res);
            System.out.print(" ");
        }
        for(int i=left;i<=right;i++){
            swap(res,left,i);
//            进入递归
            permutationByRecursion(left+1,right);
            swap(res,left,i);
        }
    }
//交换方法
    private static void swap(char[] res,int left,int right){
        char temp = res[left];
        res[left] = res[right];
        res[right] = temp;
    }
}
```

问题一栈版本

```
public class MyStack {
    public int[] listArr;
    private int tail = -1;//不仅是指向尾部的指针，也代表了元素大小
    public MyStack(){
        this(10000);
    }
    public MyStack(int MAX){
        listArr = new int[MAX];
    }
    public void push(int t){
        listArr[++tail] = t;
    }
    public int pop(){
        return listArr[tail--];
    }
    public boolean isHas(int x){
        for (int i=0;i<=tail;i++){
            if (listArr[i]==x){
                return true;
            }
        }
        return false;
```

```
    }
    public boolean isEmpty(){
        return tail==-1;
    }
    public int size(){
        return tail+1;
    }
    public void printAns(String s){
        for(int i=0;i<=tail;i++){
            System.out.printf("%c",s.charAt(listArr[i]));
        }
        System.out.printf(" ");
    }
}
```

```
public static void permutationByNoRecursion(String s){
    MyStack myStack = new MyStack();
    for (int i = 0; i < s.length(); i++){
        myStack.push(i);
    }
      从低到高排序输出
    myStack.printAns(s);
      开始循环
    while (!myStack.isEmpty()){
        int i = myStack.pop() + 1;//加一是为了与其位置匹配
        while (i < s.length()){
            if (!myStack.isHas(i)){
                myStack.push(i);
                //寻找未进栈的元素进栈
                for (int j = 0; j < s.length(); j++){
                    if (!myStack.isHas(j)){
                        myStack.push(j);
                    }
                }
                myStack.printAns(s);
                break;
            }
            i++;
        }
    }
}
```

问题二变形一

```
    public static void permutationByRecursion(int left,int right){
```

```
        if (left == right){
            System.out.print(res);
            System.out.print(" ");
        }
        for(int i=left;i<=right;i++){
//            在递归前加入不重复的条件
            if (res[left]!=res[i]||i==left){
                swap(res,left,i);
                permutationByRecursion(left+1,right);
                swap(res,left,i);
            }
        }
    }
```

问题二变形二

```
public class PermutationByRecursion3 {
    static String str;
    static char[] strArr;
    static char[] res;
    public static void permutationByRecursion(String s,int k){
        str = s;
        strArr = new char[str.length()];
        res = new char[k];
        for(int i = 0;i<str.length();i++){
            strArr[i] = str.charAt(i);
        }
//        引入不会出现的字符作为区分符号‘ ’
        for(int i = 0;i<k;i++){
            res[i] = ' ';
        }
        permutationByRecursion(k);
    }
    private static void permutationByRecursion(int k){
//        递归条件
        if (k==0){
            System.out.print(res);
            System.out.print(" ");
            return;
        }
//        把不重复字符加入(十分暴力^_^)
        for(char strArr:strArr){
            if (!has(res,strArr)){
                res[res.length-k] = strArr;
            } else{
```

```
                continue;
            }
//              进入递归
            permutationByRecursion(k-1);
//              重新置为表示字符
            res[res.length - k] = ' ';
        }
    }
//    判断字符是否在字符数组内
    private static boolean has(char[] res,char s){
        for (char tem:res){
            if (tem == s){
                return true;
            }
        }
        return false;
    }
}
```

测试类（适用问题一问题二）

```
    public static void main(String[] args) {
//        PermutationByRecursion.permutationByRecursion("abcde");
//        PermutationByRecursion2.permutationByRecursion("abcde");
        PermutationByRecursion3.permutationByRecursion("abcde",4);
    }
```

任务三

```
import java.util.NoSuchElementException;
import java.lang.IllegalArgumentException;

public class ResizingQueue<T> {
//    成员变量外加初始化
    private int maxSize = 2;
    private int front = 0;
    private int rear = 0;
    private T[] listArray =(T[]) new Object[2];
//    无参构造方法
    public ResizingQueue(){}
//    入队操作，要注意空队列，满队列等情况
    public void enqueue(T element) throws IllegalArgumentException{
        if (element == null){
            throw new IllegalArgumentException();
```

```
        }
        if (isExpand()){
//            expand()方法改变了大小，改变了 front,rear,maxSize 数值
            listArray = expand(maxSize,(maxSize-1)*2+1);
        }
        rear = (rear+1)%maxSize;
        listArray[rear] = element;
    }
    public T dequeue() throws NoSuchElementException{
        if (isEmpty()){
            throw new NoSuchElementException();
        }
        listArray[front] = null;
        front = (front+1)%maxSize;
        T res =listArray[front];
        if (isShrink()){
            listArray = shrink(maxSize,(maxSize-1)/2);
        }
        return res;
    }
    public int size(){
        if (rear>=front){
            return rear-front;
        }
        return maxSize-front+rear;
    }
    public boolean isFull(){
        return front == (rear+1)%maxSize;
    }
    public boolean isEmpty(){
        return rear == front;
    }
    private boolean isExpand(){
        return isFull();
    }
    private boolean isShrink(){
        return size()<=(maxSize-1)/4;
    }
    private T[] expand(int eSize, int nSize){
        T[] tem = (T[]) new Object[nSize];
        int temPivot = 0;
        if (front>rear){
            for (int i=front;i<eSize;i++){
                tem[temPivot++] = listArray[i];
```

```
60.             }
61.             for (int i=0;i<=rear;i++){
62.                 tem[temPivot++] = listArray[i];
63.             }
64.         }
65.         else {
66.             for (int i=front;i<=rear;i++){
67.                 tem[temPivot++] = listArray[i];
68.             }
69.         }
70.         maxSize = nSize;
71.         front = 0;
72.         rear = temPivot-1;
73.         return tem;
74.     }
75.     private T[] shrink(int eSize,int nSize){
76. //         需要对 nSize 进行判断
77.         nSize = (nSize<=3)?2:nSize;
78.         T[] tem = (T[]) new Object[nSize];
79.         int temPivot = 0;
80.         if (rear>=front){
81.             for (int i=front;i<=rear;i++){
82.                 tem[temPivot++] = listArray[i];
83.             }
84.         }
85.         else{
86.             for (int i=front;i<eSize;i++){
87.                 tem[temPivot++] = listArray[i];
88.             }
89.             for (int i=0;i<=rear;i++){
90.                 tem[temPivot++] = listArray[i];
91.             }
92.         }
93.         front = 0;
94.         rear = temPivot - 1;
95.         maxSize = nSize;
96.         return tem;
97.     }
98.     @Override
99.     public String toString() {
100.         String res = "";
101.         res += "[";
102. //         不大于 20 的情况
103.         if (this.size()<=20){
```

```
104.                 if (rear>=front){
105.                     for (int i = front+1;i<=rear;i++){
106.                         res = res + listArray[i] + " ";
107.                     }
108.                 }
109.                 else {
110.                     for (int i = front+1;i<maxSize;i++){
111.                         res = res + listArray[i] + " ";
112.                     }
113.                     for (int i = 0;i<=rear;i++){
114.                         res = res + listArray[i] + " ";
115.                     }
116.                 }
117.             }
118.     //         大于 20 的情况
119.             else {
120.                 int tem = (front+1)%maxSize;
121.                 int count = 0;
122.     //            去前五个元素
123.                 do{
124.                     res = res + listArray[tem] + " ";
125.                     tem = (tem+1)%maxSize;
126.                     count++;
127.                 }
128.                 while (count<5);
129.                 res += " ... ";
130.     //            取后五个元素
131.                 while (count<this.size()-5){
132.                     tem = (tem+1)%maxSize;
133.                     count++;
134.                 }
135.                 while (count<this.size()){
136.                     res = res + listArray[tem] + " ";
137.                     tem = (tem+1)%maxSize;
138.                     count++;
139.                 }
140.             }
141.             res = res.strip()+"]";
142.             res += "\nelements: " + this.size() + " size:"+(this.maxSi
    ze-1);
143.             return res;
144.         }
145.     }
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Test {
    public static void readFile() throws FileNotFoundException {
//        Scanner in = new Scanner(new File("src/test1000.txt"));
//        Scanner in2 = new Scanner(new File("src/result1000.txt"));
        Scanner in = new Scanner(new File("src/test5000.txt"));
        Scanner in2 = new Scanner(new File("src/result5000.txt"));
        ResizingQueue resizingQueue = new ResizingQueue<Integer>();
        int nextInt = 0;
        String nextString;
        boolean vertify = true;
        while (in.hasNext()){
            if (in.hasNextInt()){
                nextInt = in.nextInt();
//                System.out.printf("读取到的数字为：%d",nextInt);
                resizingQueue.enqueue(nextInt);
            }
            else{
                nextString = in.next();
                char nextChar;
                int pivot=0;
                while(pivot!=nextString.length()){
                    nextChar = nextString.charAt(pivot);
                    pivot++;
                    if (nextChar == '-'){
                        resizingQueue.dequeue();
                    }
                    if (nextChar == '?'){
                        System.out.println(resizingQueue.toString());

                        String tem1 = in2.nextLine();
                        String tem2 = in2.nextLine();
                        String res = tem1 + "\n" + tem2;
                        System.out.println("res 为："+res);
                        boolean temVertify = res.strip().equals(resizingQueue.toString().strip());
                        System.out.println("比较结果为："+temVertify);
                        vertify = vertify && temVertify;
                    }
```

```
41. //                    System.out.printf("当前读取到的符号为：%c%n",nextChar);
42.                 }
43.             }
44. //            System.out.printf("当前队列最大容量：%d,当前队列容量：%d front 为：%d rear 为：%d%n",resizingQueue.getMaxSize(),resizingQueue.size(),resizingQueue.getFront(),resizingQueue.getRear());
45. //            System.out.printf(resizingQueue.toString()+"%n");
46.         }
47.         System.out.printf("经检验输出的字符串与结果符合判定：%b",vertify);
48.     }
49.
50.     public static void main(String[] args) throws FileNotFoundException {
51.         readFile();
52.     }
53. }
```

任务四

```
1.  import java.io.File;
2.  import java.io.FileNotFoundException;
3.  import java.util.Scanner;
4.
5.  public class RadixSortInt {
6.      static ResizingQueue[] resizingQueue = new ResizingQueue[10];
7.  //    static int[] arr = {27,91,100,9,17,23,84,28,72,5,67,25};//仅为测试数组
8.      static void sort(int[] arr,int len){
9.  //        初始化队列
10.         for(int i=0;i<resizingQueue.length;i++){
11.             resizingQueue[i] = new ResizingQueue<Integer>();
12.         }
13. //        变量+初始化
14.         boolean flag = true;
15.         int modNum = 0;
16.         int tem=0;
17. //        一位一位操作，直到最长的数字结束
18.         while (flag){
19.             modNum++;
20. //            入队，将数组数字按照顺序入队
```

```
            for (int i=0;i<len;i++){
                int pivot = mod(arr[i],modNum);
                tem = Math.max(tem,pivot);
                resizingQueue[pivot].enqueue(arr[i]);
                flag = (tem != 0);
            }
            tem = 0;
            int arrPivot = 0;
//              出队，重新给数组赋值
            for (int i=0;i<resizingQueue.length;i++){
                while (!resizingQueue[i].isEmpty()){
                    arr[arrPivot++] = (int)resizingQueue[i].dequeue();

                }
            }
        }
//          输出
        for (int i=0;i<len;i++){
            System.out.printf("%d ",arr[i]);
        }
    }

    private static int mod(int num,int modNum){
        for(int i=0;i<modNum-1;i++){
            num = num/10;
        }
        return num%10;
    }
    public static void main(String[] args) throws FileNotFoundExcepti
on {
        int[] arr = new int[10000];
        int pivot = 0;
        Scanner in = new Scanner(new File("src/radixSort1.txt"));
        while (in.hasNextInt()){
            int tem = in.nextInt();
            System.out.println(tem);
            arr[pivot++] = tem;
        }
        pivot--;
        sort(arr,pivot);
    }
}
```

测试结果:

0 6 10 16 18 18 21 23 27 28 33 37 37 37 41 42 43 47 48 51 56 68 69 70 85 87 91 96 97 102
105 111 112 124 126 128 133 140 142 142 142 146 147 148 149 154 165 166 169 175 175
178 188 200 200 202 207 215 217 221 229 236 236 237 255 263 264 264 265 270 271 277
279 280 282 286 289 296 298 299 308 313 314 317 319 319 322 324 326 335 339 343 343
345 354 359 360 361 362 363 363 363 368 368 377 379 384 384 386 392 395 404 413 414
414 418 419 435 436 438 439 442 444 445 447 449 457 457 463 466 468 473 476 478 483
492 497 500 514 520 528 533 535 550 552 553 554 560 563 567 574 575 575 576 577 588
590 600 603 614 615 617 618 637 638 643 643 653 654 657 657 660 664 668 671 692 695
714 720 724 731 735 736 737 743 744 746 753 753 759 764 771 783 786 792 798 798 800
802 803 805 813 814 824 829 838 843 862 865 869 874 875 877 881 882 882 887 891 891
895 900 902 908 909 909 912 913 916 936 936 937 939 944 946 947 951 953 961 964 967
967 968 972 974 975 975 977 985 985 985 990 997 998 1000 1002 1003 1017 1018 1019
1020 1031 1031 1033 1037 1041 1050 1054 1058 1063 1070 1077 1079 1082 1084 1084
1086 1087 1087 1089 1096 1097 1098 1100 1103 1103 1109 1114 1127 1130 1132 1138
1148 1155 1159 1161 1162 1164 1164 1174 1181 1185 1186 1186 1187 1189 1212 1213
1216 1219 1222 1224 1227 1228 1230 1231 1236 1239 1240 1241 1244 1274 1282 1284
1293 1295 1296 1297 1307 1309 1313 1321 1330 1332 1332 1339 1339 1342 1343 1345
1347 1349 1350 1359 1361 1363 1367 1380 1381 1381 1385 1395 1398 1404 1406 1407
1409 1412 1424 1426 1427 1437 1438 1439 1439 1441 1445 1447 1451 1451 1453 1465
1470 1474 1480 1480 1491 1499 1514 1516 1520 1523 1532 1533 1539 1541 1559 1562
1562 1567 1568 1568 1579 1605 1611 1612 1614 1615 1615 1619 1620 1622 1622 1623
1624 1625 1628 1632 1639 1647 1652 1654 1665 1666 1668 1674 1676 1678 1682 1682
1686 1692 1695 1697 1709 1710 1716 1719 1727 1735 1737 1737 1737 1738 1742 1742
1743 1747 1759 1761 1765 1767 1770 1771 1774 1774 1776 1776 1780 1782 1782 1784
1786 1786 1786 1793 1794 1794 1804 1808 1810 1810 1813 1818 1822 1848 1848 1849
1850 1856 1860 1870 1873 1875 1887 1887 1890 1892 1900 1912 1914 1914 1917 1919
1921 1925 1925 1934 1944 1951 1952 1956 1958 1969 1972 1974 1975 1978 1978 1978
1979 1980 1981 1981 1981 1982 1986 1986 1991 1996 2002 2009 2009 2010 2010 2014
2018 2027 2036 2036 2043 2046 2050 2051 2051 2052 2052 2054 2054 2070 2070 2076
2077 2078 2084 2090 2093 2095 2099 2099 2103 2109 2127 2141 2145 2146 2151 2153
2160 2160 2161 2163 2164 2164 2169 2171 2172 2172 2174 2184 2192 2197 2201 2215
2233 2233 2237 2241 2243 2245 2245 2248 2248 2257 2260 2261 2262 2269 2269 2271
2274 2274 2287 2292 2305 2311 2313 2318 2319 2325 2326 2333 2334 2341 2345 2351
2354 2355 2359 2359 2360 2367 2371 2373 2379 2384 2389 2391 2391 2392 2392 2394
2401 2412 2418 2420 2425 2427 2430 2430 2439 2444 2449 2450 2451 2458 2462 2465
2468 2471 2473 2476 2481 2489 2490 2495 2496 2513 2514 2520 2525 2530 2530 2533
2534 2540 2547 2548 2555 2558 2562 2565 2569 2570 2571 2574 2574 2578 2580 2589
2590 2596 2607 2612 2614 2614 2622 2627 2628 2640 2642 2642 2645 2648 2649 2652
2657 2657 2661 2664 2682 2696 2698 2700 2700 2707 2710 2722 2733 2736 2742 2742
2744 2745 2748 2752 2759 2765 2765 2766 2770 2773 2776 2776 2784 2787 2794 2804
2806 2814 2828 2830 2831 2836 2845 2846 2848 2856 2861 2862 2862 2863 2864 2870
2876 2883 2891 2892 2896 2897 2900 2901 2901 2908 2910 2912 2914 2918 2919 2922
2922 2936 2936 2941 2941 2944 2947 2952 2957 2959 2961 2966 2970 2971 2972 2975
2975 2975 2982 2986 2988 2989 2997 2998 3003 3004 3006 3011 3011 3015 3024 3026

3030 3035 3035 3040 3041 3042 3045 3058 3059 3064 3072 3073 3077 3086 3095 3097
3098 3100 3101 3103 3103 3106 3113 3119 3125 3126 3127 3133 3133 3136 3138 3140
3143 3144 3146 3155 3156 3161 3167 3172 3172 3175 3175 3177 3180 3189 3192 3193
3196 3212 3213 3214 3216 3223 3224 3224 3227 3229 3233 3236 3237 3238 3240 3241
3250 3254 3254 3263 3267 3271 3275 3277 3278 3278 3280 3296 3297 3300 3301 3309
3312 3313 3315 3319 3320 3323 3325 3329 3330 3332 3334 3338 3341 3357 3368 3369
3375 3391 3393 3398 3399 3399 3401 3409 3411 3411 3413 3417 3418 3419 3421 3421
3424 3426 3427 3433 3434 3449 3450 3451 3461 3465 3466 3475 3477 3483 3490 3492
3498 3502 3504 3507 3513 3519 3524 3526 3528 3531 3534 3542 3546 3557 3558 3569
3570 3572 3573 3587 3588 3596 3599 3601 3606 3615 3617 3630 3631 3631 3631 3634
3634 3638 3641 3651 3657 3662 3672 3672 3677 3681 3687 3687 3688 3695 3695 3696
3702 3715 3724 3726 3732 3733 3735 3740 3744 3747 3749 3759 3762 3769 3773 3777
3780 3783 3789 3795 3796 3797 3806 3810 3832 3836 3848 3851 3852 3852 3855 3861
3867 3873 3877 3881 3882 3884 3887 3889 3890 3895 3901 3907 3912 3915 3922 3925
3926 3926 3927 3934 3935 3936 3939 3942 3943 3945 3954 3973 3974 3977 3982 3988
3994 4001 4001 4005 4014 4022 4023 4026 4032 4037 4038 4038 4040 4042 4043 4045
4049 4057 4069 4070 4078 4083 4084 4090 4093 4093 4095 4099 4102 4110 4113 4113
4121 4128 4143 4145 4152 4156 4157 4165 4167 4168 4173 4174 4175 4176 4178 4181
4181 4183 4186 4188 4188 4189 4190 4192 4193 4197 4204 4204 4208 4208 4213 4214
4217 4219 4233 4234 4238 4249 4252 4256 4258 4261 4263 4266 4270 4276 4277 4279
4280 4282 4284 4290 4297 4302 4304 4318 4326 4328 4328 4328 4332 4337 4338 4344
4346 4347 4350 4351 4359 4361 4361 4365 4366 4368 4371 4374 4375 4382 4382 4384
4385 4389 4393 4394 4403 4412 4418 4422 4432 4435 4436 4437 4439 4444 4444 4447
4451 4452 4456 4456 4456 4461 4466 4467 4471 4472 4475 4478 4483 4493 4499 4499
4506 4507 4507 4508 4517 4522 4530 4541 4542 4542 4548 4550 4550 4558 4558 4563
4563 4565 4566 4569 4580 4584 4587 4591 4593 4597 4600 4604 4606 4614 4616 4618
4618 4623 4631 4644 4651 4651 4663 4664 4669 4670 4671 4693 4695 4695 4698 4699
4711 4712 4715 4716 4718 4733 4740 4749 4751 4754 4756 4757 4758 4759 4768 4774
4776 4793 4796 4802 4810 4810 4812 4818 4819 4823 4823 4824 4834 4839 4840 4844
4845 4852 4857 4858 4858 4862 4863 4865 4877 4880 4891 4900 4900 4907 4908 4908
4911 4914 4917 4925 4929 4934 4938 4943 4943 4944 4949 4951 4953 4956 4962 4976
4980 4990 4991 4993 5004 5006 5008 5013 5016 5019 5022 5022 5024 5032 5032 5033
5038 5044 5049 5050 5051 5062 5083 5086 5088 5088 5093 5093 5094 5107 5111 5111
5114 5131 5135 5135 5137 5144 5145 5148 5149 5157 5161 5187 5187 5192 5200 5203
5211 5223 5224 5228 5238 5239 5239 5240 5244 5248 5255 5258 5259 5273 5276 5277
5281 5287 5294 5309 5312 5324 5325 5332 5332 5336 5344 5351 5351 5352 5353 5358
5372 5372 5384 5385 5389 5398 5398 5405 5407 5408 5412 5412 5413 5416 5420 5421
5424 5426 5428 5433 5438 5442 5444 5444 5458 5459 5461 5463 5466 5468 5480 5483
5493 5496 5500 5503 5504 5508 5512 5514 5524 5536 5537 5542 5543 5543 5544 5544
5547 5551 5559 5560 5560 5561 5567 5568 5569 5571 5574 5578 5578 5583 5597 5598
5599 5602 5618 5622 5628 5628 5645 5652 5655 5659 5662 5665 5669 5669 5669 5674
5677 5679 5680 5682 5686 5690 5694 5697 5700 5703 5706 5715 5717 5719 5724 5728
5740 5744 5749 5754 5755 5756 5774 5777 5782 5784 5788 5788 5789 5791 5791 5791
5796 5799 5806 5809 5812 5814 5815 5822 5835 5836 5836 5838 5840 5840 5857 5859

5860 5863 5863 5870 5874 5876 5879 5882 5886 5887 5888 5891 5891 5893 5893 5896
5902 5910 5924 5926 5930 5932 5932 5933 5935 5937 5937 5941 5948 5949 5959 5962
5967 5969 5970 5973 5974 5975 5978 5983 5992 5999 6006 6010 6012 6012 6014 6016
6019 6019 6020 6021 6030 6034 6035 6043 6043 6045 6053 6054 6055 6057 6065 6065
6066 6066 6073 6075 6084 6088 6099 6101 6104 6108 6110 6110 6112 6117 6123 6125
6131 6148 6149 6160 6164 6168 6169 6173 6178 6182 6184 6189 6189 6190 6193 6195
6199 6210 6211 6216 6230 6233 6233 6239 6243 6244 6257 6262 6264 6270 6271 6273
6274 6276 6277 6286 6287 6293 6293 6293 6295 6297 6300 6308 6308 6332 6336 6344
6348 6350 6359 6360 6362 6362 6365 6369 6386 6387 6392 6399 6403 6406 6406 6406
6411 6412 6413 6415 6416 6418 6419 6425 6432 6445 6448 6454 6454 6457 6495 6496
6502 6515 6520 6520 6522 6524 6529 6534 6538 6539 6542 6551 6552 6556 6557 6564
6565 6571 6575 6578 6589 6589 6592 6595 6596 6597 6600 6605 6607 6612 6614 6616
6631 6639 6641 6642 6646 6646 6647 6652 6652 6653 6661 6663 6665 6667 6676 6678
6679 6680 6683 6688 6695 6701 6716 6721 6721 6726 6728 6730 6731 6732 6738 6756
6756 6764 6765 6773 6778 6782 6786 6792 6795 6800 6807 6810 6811 6824 6828 6834
6838 6839 6839 6842 6848 6852 6856 6859 6865 6866 6869 6870 6870 6871 6873 6875
6880 6880 6884 6886 6887 6887 6891 6894 6896 6897 6899 6899 6900 6902 6903 6904
6905 6910 6914 6916 6917 6921 6923 6924 6927 6931 6934 6937 6943 6943 6959 6961
6963 6963 6965 6969 6973 6976 6983 6986 6988 6993 6994 6999 7001 7010 7016 7021
7021 7029 7035 7036 7042 7043 7044 7047 7056 7066 7069 7073 7076 7081 7086 7098
7099 7101 7104 7108 7109 7110 7110 7111 7114 7117 7119 7121 7122 7132 7133 7134
7134 7135 7150 7151 7152 7162 7163 7168 7177 7177 7179 7197 7197 7199 7202 7212
7213 7214 7218 7222 7223 7228 7233 7240 7244 7246 7249 7254 7256 7257 7259 7262
7264 7266 7267 7275 7275 7280 7285 7287 7290 7301 7302 7315 7320 7320 7322 7322
7333 7333 7335 7338 7340 7341 7345 7345 7353 7369 7378 7380 7382 7384 7386 7387
7387 7389 7399 7402 7416 7417 7419 7419 7436 7439 7440 7445 7447 7447 7451 7457
7463 7463 7464 7464 7470 7470 7473 7474 7474 7476 7477 7486 7490 7491 7491 7492
7495 7496 7496 7504 7511 7514 7517 7524 7525 7525 7538 7538 7545 7546 7552 7553
7561 7564 7565 7570 7588 7590 7591 7601 7603 7611 7620 7630 7636 7640 7650 7651
7652 7659 7660 7660 7668 7674 7677 7678 7678 7680 7691 7691 7707 7709 7714 7715
7724 7740 7743 7745 7747 7768 7782 7792 7799 7803 7804 7807 7808 7811 7815 7821
7822 7825 7826 7841 7841 7842 7849 7850 7855 7863 7866 7869 7870 7873 7876 7878
7891 7909 7911 7913 7918 7919 7921 7929 7943 7945 7947 7948 7948 7953 7955 7958
7963 7965 7968 7994 7994 7995 7995 7995 7996 7999 8281 8561 10094 10421 10941
13085 14538 14717 15269 15570 16329 16350 16801 17203 17313 18052 18182 18554
19136 20388 21342 21671 22973 23341 23899 24127 24728 26152 26270 27050 27578
27719 29376 29380 30335 31143 31403 31914 32449 32611 32781 33094 33209 33583
33658 33913 34804 35734 36035 36265 36517 37846 38058 38512 38840 40717 40733
41555 41646 41921 42140 42599 43295 43667 43766 44338 44464 44930 45213 45971
46182 46799 46822 48116 48277 50043 51830 52660 53191 53419 53436 53895 53923
57375 57890 58498 59375 59407 59834 60683 60973 62871 62983 63489 63943 66284
66553 67026 67973 68254 68715 69722 69811 69885 70568 70911 71116 72477 72754
73040 74532 74841 74895 74917 75076 76449 76595 77176 79514 79528 80261 81852
82278 82416 82425 82762 82774 83495 84465 84814 84854 85295 86491 87193 88032

89419 90641 90855 91281 91334 91381 92440 93378 94294 95031 95333 95450 95636
96476 96702 96820 97951 97974 98311 98360 98363 98483 98677 98911 99125 99476
100108 100950 101126 101182 101914 102942 103443 104704 105347 106542 106902
107468 108411 109014 109087 109538 109684 110138 111086 111201 114875 115202
115503 115835 116227 116707 117027 117289 117328 117338 118047 118914 119603
120266 120665 120915 120978 121543 121556 121588 121847 122373 123355 126162
126487 128625 128677 128820 129010 129108 129712 131501 132200 132264 133520
134315 135246 136069 137630 137959 138786 140974 141053 141577 142157 143321
143592 144660 144823 144992 145395 146771 147212 147538 148175 148668 149337
151194 152070 152734 152995 155616 156133 156281 157143 158899 159487 160092
160767 161344 161531 161839 161915 162087 162773 163305 164275 164740 165178
165515 165655 166647 166923 167002 167158 167416 167503 167705 167752 167815
168942 169121 169654 169752 170082 170521 170764 171456 172605 172929 173400
175047 175503 175516 176297 176777 177402 178602 178664 180349 183782 183885
184816 184914 186767 187143 187760 188132 188333 188864 189378 189467 190488
190496 192701 192771 193510 194614 194796 196603 197267 199323 199394 199613
199757 199837 201218 201890 201960 202078 202190 202216 202859 203642 204632
204814 204846 205713 206087 206520 207497 207972 208266 209162 209296 209397
211340 214100 215079 215182 215549 216202 216803 217684 218896 220156 220218
220320 222028 222111 222281 223334 224303 224634 224883 225709 225863 227820
228608 228634 228710 229910 230498 231007 231289 232013 232044 232259 232559
233571 236708 236826 236912 238375 239820 240512 241518 242258 242747 242797
244167 245306 246067 246437 246988 247236 248016 249296 249788 251299 251949
252507 252559 253515 255312 255944 256109 256264 256392 257444 257913 258079
258969 259095 259281 260241 260948 261293 261347 261741 262503 262738 264105
265050 265138 265525 265811 266108 266596 268057 269001 269375 269976 270038
270338 270367 271947 273170 273189 274035 274330 274483 274774 277152 277454
277784 278102 278346 279861 279954 281298 281515 282347 282444 282932 282940
283007 283016 283506 284497 284731 285437 285564 287148 287343 287350 287658
288432 288680 289175 289488 289515 289621 289722 290001 290006 290117 290924
290975 292593 293362 293645 294532 296007 296983 296992 297174 298660 298797
298895 299457 300806 301118 302048 302421 302576 303457 303554 304663 305159
305307 305435 306103 306308 306544 306718 306935 307181 307493 307926 308640
308902 309129 309213 309217 309244 309882 310119 310598 311065 311404 311458
311966 313050 313363 314577 314751 315087 315570 315735 316508 316514 317090
318018 318397 318909 319244 319758 322261 322968 323695 323754 324024 324407
325040 325192 326006 326265 326287 326368 326955 327597 328296 328581 328591
328777 329546 330312 330647 330808 330906 330957 331298 331379 331634 331641
332694 332992 334032 334663 334954 335457 336235 339204 339443 341041 341248
341867 342190 342267 344002 344974 345459 345679 345756 346368 346518 346888
347390 348761 348963 350043 350874 351336 353346 353812 354371 354953 355032
355665 355745 356222 356688 357287 357421 358411 358715 359764 359986 360470
360582 361337 361536 362220 363043 364242 365267 365694 366018 366102 366614
367155 369586 371367 373020 373288 374877 374945 374970 375552 375718 375989

376208 377022 378203 379172 379640 379940 380557 380654 380859 382426 382442
383226 383422 384015 384703 385171 385831 386135 386216 386317 386416 387453
388384 388561 389367 390042 390309 390637 391123 391453 391789 391869 394022
395368 395625 395673 395791 396156 396727 397056 397948 398615 398740 399585
399712 401294 401887 402870 403600 403828 405172 405300 405360 405379 406930
408419 408562 409076 411102 411542 411570 411740 412167 412287 412923 413176
414294 414482 414966 414996 415497 415712 415760 416427 416956 417240 419210
419337 419998 420890 421363 421760 421858 422174 422402 423034 423140 423780
425046 425866 426652 426930 427712 428450 429571 430009 430189 431648 432970
433318 434737 435293 435608 435725 436272 436400 436693 436788 437025 437028
437124 437592 438222 438262 439026 440686 440760 441530 443014 444379 444507
444570 444666 444754 445474 446418 446699 447754 447940 448294 449023 449142
449811 450174 450899 452359 453065 454446 454614 455166 456044 456392 456511
457748 457765 458013 458344 459844 460037 460839 461816 463117 463883 464288
466223 466285 466288 466844 467167 467813 467894 468293 468591 470182 470222
470430 470698 470791 470968 471968 472064 472585 472831 473329 473553 473612
474102 475720 475727 476123 477073 477380 477863 479786 480451 480589 480693
481514 481810 482088 483513 484273 484579 485360 485715 486713 487348 487782
489478 489493 490442 490640 491238 491348 495103 496095 496217 497027 497148
497556 498361 498388 499184 500473 500928 500985 501241 502395 503654 503962
504782 505471 506632 507392 507883 509540 510422 511126 511680 514678 514751
516126 516642 517993 518973 519727 519761 521121 521259 522828 523968 524450
526499 526647 526741 526938 527208 527289 527564 528292 529406 530104 530775
531002 531104 531301 531556 533180 533430 534517 534564 534843 535890 536121
536252 536412 536859 536904 542971 543017 543617 544229 546100 546314 546623
547606 548134 549488 550187 550535 551138 552317 552397 553343 553919 554028
554056 554194 554358 556224 556421 556570 557243 557370 558356 558391 558475
558569 558933 560143 560587 560605 561331 561333 562104 562445 562751 563203
563455 564164 566144 566244 566657 567588 568093 568431 570282 570710 570788
571186 571874 572091 572862 572965 573760 573948 574191 574212 574783 574828
575360 576477 577546 577796 579506 580503 580563 580873 582087 583238 583347
583632 583691 583876 584607 584835 585178 585207 585924 585935 586577 586952
587011 589247 589640 589836 590205 590286 590600 590610 590890 591235 591462
591945 591950 591955 592779 592996 593085 593368 593599 593727 594818 595576
596605 597162 597915 597996 598452 600033 600191 600317 600936 603461 603722
603735 604850 605070 605641 607804 608138 608369 609466 610848 611413 612573
613221 614227 615667 616156 616566 618866 619202 619551 619639 619855 621146
621184 621542 622129 623880 625018 625475 625515 625730 627743 628792 628893
629103 629240 630692 630762 631286 631590 631780 631796 631796 631944 631961
632259 632771 634010 634053 634228 634963 635400 635915 636230 636452 636962
637702 637858 638752 639771 640672 641190 641664 642167 643093 643233 643261
644335 644541 644714 645036 645624 649230 649376 649874 650719 651793 652023
652470 652624 652689 652766 652836 652953 653121 653808 653976 654450 655166
656838 656967 658473 658575 659106 659946 660356 661065 661273 661354 662195

662400 663355 664951 666465 666746 667413 668084 668377 668449 669474 669544
672150 674038 675631 675698 675719 678063 678306 678828 678875 679556 679557
679689 679718 680477 680721 682517 682669 682720 683073 683524 683856 685422
685609 685845 686043 686164 686818 686988 687280 687294 687332 688715 688958
690087 690401 690406 690734 690848 691357 693218 695141 695515 695679 696511
697795 698010 698770 699043 699384 700023 700541 701223 702234 702424 702648
703216 704430 705352 705551 705586 705591 706111 706670 707217 707308 708124
708404 709032 710715 711250 711570 711699 713089 714286 714895 715172 715762
716047 716260 716339 717303 718543 718578 718943 719469 719585 720702 724078
724833 725173 725294 726904 727064 727157 727268 727664 727975 728451 729169
730325 731858 732387 732810 732895 732920 733435 733753 733955 734050 734851
735622 735952 736213 736314 737720 738089 738217 738290 740184 742542 742692
743162 744233 744998 745643 746030 746388 746654 747067 747625 747682 748489
748745 748973 749119 750570 751128 752331 752529 753394 753481 753589 753758
753759 754439 754530 756271 756575 756954 757118 757428 758287 758445 758588
758998 759166 759648 759709 759793 759929 760570 760970 761076 761461 761605
761873 762168 763281 763585 763848 764994 765766 765900 766238 766338 766606
767969 768112 768470 768586 768805 769602 769622 769936 770618 771441 772513
772625 772939 772961 772965 774174 774223 776221 776489 777278 777353 777631
778278 778660 779880 779995 780243 780399 780883 782262 782490 782681 783550
784026 784031 784093 784709 785023 786503 788194 788287 789871 790336 791517
792737 793823 794009 794879 794971 795646 796025 799122 800864 802189 802488
803948 805162 805638 805966 806098 806243 806877 807741 808043 808073 808309
809546 809680 809838 810009 810649 813963 815231 815849 816259 816545 816963
818009 818340 818743 818830 819361 821386 822050 822539 825038 826376 826898
827283 827827 828457 828979 829622 830404 831050 831130 831690 831868 832434
833664 834260 835262 836874 836954 839103 839597 840032 840100 840245 840289
840577 840607 840908 841380 841534 842390 842991 843165 843182 843883 844420
845801 846507 848195 848654 848882 850664 850950 851301 852268 852765 853441
854407 854537 855113 855263 855878 856364 856977 857902 859148 859320 860182
862751 863431 863837 864431 864838 866293 866901 867114 867180 867895 868972
869423 871067 871500 871735 872150 872404 872667 872771 872958 873793 874072
874148 874339 874375 875111 875656 875788 876099 876237 877580 878571 879709
879885 880872 881108 882335 883161 883287 883297 884846 885330 885418 885684
886753 886795 887053 887634 888347 888683 889361 891016 891849 892161 892679
892875 893262 893649 893777 894125 894205 894634 895271 895660 895965 896137
897124 897223 898081 898135 898198 900471 901033 902453 903379 904291 905876
906300 906354 908769 909008 909247 909377 909439 911010 911140 912050 912054
914237 914919 915030 915233 915819 916334 916415 917466 917608 917915 918095
918184 918616 918666 919840 920030 920373 920788 921475 921946 924142 924366
925598 926759 927315 927651 928419 929047 930148 930599 932963 933678 934896
935623 936389 937868 939216 939551 940102 940483 941532 942289 942360 942415
943083 943227 943803 944057 945564 945581 945994 947265 947301 947403 948123
949562 949885 949886 949899 950101 950252 950553 950582 951153 951398 951950

952501 953134 954174 954600 955728 956873 957004 957669 957820 958053 958208
958379 958599 959043 959282 959382 959863 960220 961300 961485 961508 962908
963282 963608 964064 965153 965457 965743 966612 966936 967133 969119 969484
970704 970876 971533 971717 973466 973695 973845 974151 975623 976022 977105
977333 978629 978695 978855 979390 980495 980835 982054 982377 982483 982715
982971 984587 986180 986548 986573 986741 987071 987071 987601 987985 989195
990042 990839 992677 992915 994091 994146 994329 994841 995519 995875 996700
997199 997314 997633 998232 999134 1002201 1002281 1002689 1003154 1004168
1004758 1004797 1005225 1005517 1005578 1006037 1006452 1007830 1007831 1008070
1008665 1008961 1009180 1009487 1009956 1012393 1012418 1012605 1013477 1013883
1014865 1015263 1016067 1016426 1017733 1018567 1019626 1020040 1020527 1021161
1021523 1021870 1022150 1024129 1024876 1024944 1025756 1027688 1027791 1027903
1028034 1029705 1029940 1030482 1031562 1031889 1032303 1032540 1033317 1033764
1033773 1034602 1035703 1036803 1038056 1038918 1041237 1041353 1041621 1042056
1042991 1043976 1044311 1045797 1045926 1046184 1046569 1047394 1047480 1047940
1048009 1048072 1048520 1049546 1051097 1051762 1051832 1052527 1052854 1053133
1054154 1054231 1054900 1055184 1056249 1056966 1058092 1060612 1062929 1063760
1064655 1064659 1064756 1065417 1066666 1067256 1067714 1067739 1068013 1068039
1070862 1070955 1071746 1072628 1074011 1074088 1076501 1076620 1076948 1077433
1078142 1080096 1080328 1080774 1082405 1083440 1083890 1083989 1084509 1084546
1086065 1086809 1087383 1087683 1088488 1088631 1089003 1089956 1090201 1090972
1092024 1092189 1092692 1092927 1093078 1094687 1095673 1097685 1100058 1100687
1101311 1102232 1103190 1103453 1103650 1104039 1105720 1108402 1110509 1111005
1111639 1113346 1113425 1114002 1114106 1115015 1118243 1118535 1118981 1122375
1122440 1124109 1124240 1124401 1124560 1124960 1125876 1128908 1128976 1129114
1129244 1129577 1129869 1130717 1132250 1133015 1134215 1134542 1136467 1137253
1137502 1138252 1138498 1138808 1139237 1140000 1140217 1141194 1142686 1145847
1146680 1147051 1147200 1147608 1148092 1148364 1148910 1149127 1149481 1149689
1150093 1150142 1150798 1152640 1152970 1153389 1153483 1153878 1153943 1154110
1156861 1161139 1161462 1162076 1162342 1163524 1164479 1164499 1164561 1164718
1165166 1165534 1167366 1168040 1169896 1170951 1171260 1173378 1173408 1174101
1176164 1177414 1178092 1178337 1179383 1179499 1180574 1181343 1181623 1183479
1183684 1184024 1184391 1184917 1188641 1188971 1189185 1189582 1189753 1190630
1191226 1191454 1191564 1192946 1194653 1195465 1195705 1196882 1198179 1199314
1199819 1200850 1201068 1201257 1202775 1203380 1203540 1203580 1203599 1204609
1207368 1207500 1207957 1210087 1210693 1211032 1213413 1214466 1214714 1214914
1216408 1217347 1218801 1219176 1219773 1220457 1221077 1221546 1221558 1221831
1222154 1222410 1222540 1222578 1223702 1224240 1224891 1225525 1225955 1226036
1227416 1228735 1228800 1230213 1230231 1231992 1233870 1234264
Process finished with exit code 0

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```

```
4.
5.  public class RadixSortString {
6.      static ResizingQueue[] resizingQueue = new ResizingQueue[26];
7.  //    static String[] arr = {"Abc","bde","fad","abd","bef","fdd","abe
    "};//仅为测试数组
8.      static void sort(String[] arr,int len){
9.  //        初始化队列数组
10.         for(int i=0;i<resizingQueue.length;i++){
11.             resizingQueue[i] = new ResizingQueue<String>();
12.         }
13. //        因为等宽，所以取第一个字符串长度作为长度
14.         int stringLen = arr[0].length();
15.         int tem=0;
16.         for (int k=0;k<stringLen;k++){
17. //            入队操作
18.             for (int i=0;i<len;i++){
19.                 int pivot = arr[i].toLowerCase().charAt(stringLen-k-
    1)-'a';
20.                 resizingQueue[pivot].enqueue(arr[i]);
21.             }
22.             tem = 0;
23.             int arrPivot = 0;
24. //            出队操作
25.             for (int i=0;i<resizingQueue.length;i++){
26.                 while (!resizingQueue[i].isEmpty()){
27.                     arr[arrPivot++] = (String) resizingQueue[i].deque
    ue();
28.                 }
29.             }
30.         }
31.         for (int i=0;i<len;i++){
32.             System.out.printf(arr[i]+" ");
33.         }
34.     }
35.     public static void main(String[] args) throws FileNotFoundExcepti
    on {
36.         String[] arr = new String[1000000];
37.         int pivot = 0;
38.         Scanner in = new Scanner(new File("src/radixSort2.txt"));
39.         while (in.hasNext()){
40.             arr[pivot++] = in.next();
41.         }
42.         pivot--;
43.         sort(arr,pivot);
```

```
    }
}
```

测试结果:

AanVzwts aBxUjUgM acCeUplN ACgnsNit aClOpVym ADBJMszB ADjSFRsM ADRKabPw ADRKsblB aduHSYDE aeauyVZZ AEppMxnC afxSnVUq AGlWmdlX aGPhNIPi agPSTilR AgYNuVcd ajRFiFEH aKEXlXSx akFxAMWQ AKpTKsKE akVCMeil aldfgZaB AlocZkRl ALPQrDwG AlRpOQFn aLssZMEa aLZVEfWT amcHclwZ AmpCpEzH amtVhhLV aNjwXjuC aNMyTANF anNpbNZy anpQfLeK aOwDbbJa apEbFpCn APOeBcfB aPXvpDSB ApyjWZtA AqcsZvdQ AQdwNrjL aQfLeTLC aQmaHxPe aRoJoxNe arrpYERX ARzprbYP AsasKDXt asyDvTbi atGwgPKl atJiSRmO AuAkCOZp AUNkreFd auqaXFCf avmHmwRl AWtDOqOs AXQvcSbs AYbuimYA AyeMQadX AYOGHDOX AYvhbcNk aYZrqEKE AzbxOQjz AztMmxSg bAoKcwcL bbHAEQcd BBoJEHQO bBtEYwCX bbVctifw BcfSsGll BCKKRHgK BCVEDsin BdldiavX BebCMCFG becTqPrx BEeMSeCF BeESGied BeNiNNeP bEqylGhY BeYlGzDR bFbkrrTu BFPClXne BfQXQrSq BFsZzJdJ bfUOZwul bfyQEKow BGbPBCFy bGUyDEdC bgvPslyl BhdMwoqp bhNnMgxJ bHPxCwvV BHXvhnwU bHxYCKKz BlaGgssZ blguGClJ blMkWAbl BlpcyQtc blsTPcZt bJGVzDvr BjlpWzKO bJuJAJqO bKSQYgkH BKthFwAh BldNMuoT BlMgPzth BLpdDeyE BLrxGkDU BLSapdFF BluZbTNF BmAOdcVg BMSgeojg BmWJuUBp bMxKLFOR BNDToivF BnJDKKnP BNoQwkio boEtZqqb bOgkXwju BoqxeRfF bosfvKtp BoWLAtbX BPEewCuy BpLKtynl bpojasCw bPXvlmRi BQLpANqw BqUXvyKK bqVzqByS BRdegocn BrljOwPU BrOwZlQJ BRvuBbJk BrwMILAt brXfiQjL brZRFWQO BsMzNwgB BsOLVCxt BSvePbmL BtlZCAlO BtPvaoGP bTxZJJcl buekjsPy BVowzoaV bWFRXoDE BwKtXaWi bwljZEjw bXjWlkZQ BXsNOPsc bYEOyuMn BYExhMvV byqLAeFl bYqtsUPT BZatOZlY bZndjSKl BZROZEhw BZWGzKsB BzwqTzbP bZYLKTso bZyWqslW CbjEjRiS CbkthTGB CBRRnnMj cdqpWZhG cEFMLQxR cFWvAXbu cGDoOMWO cHcJlpbK chHoPsOS cHqTsiZK ChVrQQNP chvzhPcf CHYPClDu ChYqdePX CKFFxeLy CKzDQoKU clfcyHFf cMrvPzJX CNTvDfNQ COfaEqsa cOMchFiA coWqXZJA coZMjKLN cpTXuvAZ CpUMDOkm cpXszFEi CQXrSMyS cRTgoFzS csEgeBrx CSNLVWRY CsOLIPXE CsReToVT cSuJZDfw cSwsttdO ctbLJzlG cTxdXfoP CtyoeBif cuDdVaXU cuJURfHR CuYqXlxc cVFOTEAg CvVFDJGp cWFWiUEC CWjPhsZl cWQjhNby cxPDoJAp cXUSBCZv CxwtNrcj cYHmbzzb cyMXABKj cYUdHdPA CZGygfyD CzUcMLbQ cZxztxvA CzYvaoji dAAAOmyl DAieCqpw DAwHxUIl dbfJCiPi dBGUrySV dBgwujon dCaSLtAe dCHGxvED dcozlRFR DcuwFuDM DdbKZMDY dDeyHoyn dDKdrFPv ddLzbgWI DdXkQqWb deBtHgGf DECHGGpr deihxyoD DeQbsPHB DErGCsXW DfiAuKcs dfpBgAjZ dFrcWkoF dFtWhCJs dfuwEnGN dFwUDqer DGEJHqLJ DhBCTDdg dhBhRvAS DhJkthfw dhXlnTBm DhxNmoMR dhZrbQyC DibxlsnC djAGkEUU DJAlTpBp djjMbfjY djJvLbgq DJVEqGKh dJzbaadG dKWTPcKX dlldSEkY DlnBLKaO DmRlOlwm DNEHJNNH DnQENUKB dNTXcCcF DOIzzAlY DoMuQsvu DoPLqppt DoqJhoSA DotyEsrg dpmBkywy dpVdOuNc DqfgvmYH DqMuGCER DQpJSjfZ dRCaMZUR DrOcfQgB dRvBcWPQ dRXqZQDC DsAUykiF DSDJEFTl dSvXtQRW DTGEtGol dUGZNoZx DUmmGmkV duUzOVLl dvBnbdKu DVlUADcP dvnJGacx DVrdTWAG DvTjTKqs dVzRrdqe dvzrUePg dXKLEEoE DxoBFntt dxUpGVee DyFEruoL DYYJZWot dZcFltCk DZpvQiND EaQGvbgP EavcnujP eawdmqWx EbFGvKAK EBfyRheE EbhwbxKm EBuAWzjK ebzjmlld EBzwtuMw EDbqMzSi EDdYGnPy EdMuaVYD EDqtfbOB eDulsMpl eDyKdArB EeVKdyDj eFJudoiW eFKyBQdn EFMhqVlW egdmkYqA EgJSSaAA eGkyjDkA eHhbJDTe EHPZsHQd EhWUEJTY

eIBbSzbC EIfVfFvB eigVkzkt eiMbInQN EiWwMEPB EJEphBCl ejMHqDXa ejNTRtBA ejTRjplj EKaMXHEY EKcuHzqc eKKaVvhi elOJoxnY ElqPuFvK EmfZTcBY EmNnwxNS EmWlwMDL EnlZuTza eNkYYTNU enOJCULs enRqFMsw EOjntupu EpJzRNVk epOAbQUU ePrgTAKY EPVRgPsz EQLOjpPF eQMikRBP ErAfCDdQ ErMskxET ERPmqnsu erPxabev eRsbRvAN ervzNrWH EsCgaPrL esZuPqjv eTDcmVZi ETLeOJFX ETLkqTdN etqpKJNM etVfEwbP etXEVtAo EuiVkpSF eUnozARR EUwmjxbc evlurWQc EVJfyFIL EVsWYzAH EvUmGRDK EVzZBZzk ExQSCFfk ExTgEnDT eXXPbTmn eyCQbBKa EYgWhHAW eYqgXHhm EyVBNyXU EyXXtZNf EYYjGsyl EYzBBkry eZhPisTr ezQXpxkf FarfTEoH fbYvWGMu FcBidqcH FCcrHYuC fCqnZaOZ fCRMYEpg FCwTVUAF fDFkZoPO fdGGXKip FDgoZVjN fDOeqYWM FdyOAIWu FExMDbUb FFDsrSzo FFnKjQdk fFvMjIpM FGIovVOW fGiTCsjf fGjqmkXu fHaaxVZJ FHtidLbg FHtlpsYc FHwTfrlZ fiFwlNUl FlsIkULW fJAgqxMp fJHTGQdz FKOgQcPo FkQlOfbd fKtNvefu fKxvqrrB FKyEBPkm fLbItFFC flFzJCac FllabauB fllSCSOY fLLFOAQF FlUlMWKu fLxbDJCY fLXrtXiU Flzxeyxy fMgWoaAV FMGzZAay FMzPHWAd fNDuVwDl FOlvRYDO fOuSGUVc FOvLcbrt FPahkMUe FpKTiymA fpTuTyXY FqRPxuNQ fqUZLMOU fQvpduqD FQXBjkpq fRneYOEB fsujbOcB ftcmWpmn fuiwDmDo FuiyaYQf fulXydPp fUnpuJNp fUpNkxbj fuUDEvVb fvewmtSB FvLybtNi fvnNEaIL FvUMRdJb FvzQlmFY fWnAZXUf FXGzDnhz fxHPDwyq FxSEupcd FYbtPAtY fyftqzRx fyyugKhE FZeedXhF FZjCpirC FzPLBzSG FZWZHTZd FzYBLtbD FZYPLNzB GbDmWrQS gbeDvfaZ gBjcGPyS GbTAtKSK GCAEFLTk GCaZWpgj gcBVMJVU GCDKlEdF GcGcWuhr GCSBZHTy GdOdMtZE GeQlSTiv GFBsWBPD gfymcHrk gFYUXlCY gHHFlylf gIAEvZNC gIgkXLwG gIlgdmjp GIQTmzyf gissLHJq gJbqhzMK GJDFwmVn GJJnbhEN gjkiMJbi GjLPfZHC gkaxkQMD GkBrkYom GKCrVZLk GknhihhS GkTaNKjv glgbuXNo GLJmJhMi gLxdoonr GMfKdNth GmiOEHXJ gmJYVDbN gmXlVqPw GmYBVDNc gnLvFpkP gnrJtNif gnUeZiUw gOahyfyR GocDeEWt GOdXKHzs gOLsdkrf GoNjvkPw GOtSKGMU gOwOsXdf goYuAVDh gQeeftsv GqQzoPeT GRhiGRFe GRJDeiZB grlvQWtt gRoaIEEw gsaDgKMp GSFReRbe GsLZqbvg GsQvOpmv GsqYNBiS gsWRZHPe GsXxYZng gTDMYVBh gTHwEbxM gtOkHeBL gUacCsRX GvCchkeM GvDeIxPA gvdqxYyq GveBXjaT gWfvHIjE GwGMPTAV gWiPSmMz GwOvKxwx gWRSDdPx GWUyyJLR GxAeJGwe GxDrHrrw gxjBNxQr GxlKpiuo GxtFTxoz gxWoArcZ GYcECmFf gyeajVmV GyjtDFzN gynKvpTl GYskJRfk gyzQgCuW GZoUZcdp gzRcMnBm HaAHNCEq HArnkdEU HaxQWgji hBEfVRfj HbHesScN HbWKhIqv hCbkHjoJ hCJFWJOq hctrKjUu HDLDYAFB HDSfOQOo HdSremou HdTbOYhD heFohqCL HfCZgMae hFFrJjKe HfLvPYeo hFNkTcHm HgJnRIuX HhkxyXgc hHnhcMOP HhUHmOmy higFpQMV HIlKjvWQ HijkUkta HioUtMXo HISnaERR hiUQjUZG HlaeevsN HLSojusj hLWOebuB HMGRFufh hMOTNUtL hnDbcqIV hOQeSAjW HPBUhuxS HPdRBtMe HQGuqCop HqOFVOuy HraqLdVX HRdwZnvS HsBcxRwD hsJDwvBJ HSuLGxxR htBUNKzA HTTfskeL htwzlHuO HVRgSQaT hVwJVPfD hvXSwkYr HwCvDotb HwSJCmrl hwuVMeBt hXqUBBxf HYarFmDN HYCxLykW HYGCVEPI hYJVLIFg HyRiZEIz hZlkttNZ HZKTnVPH hztBlVCQ HZUGIhGR hZZLCcKL iAfIJEvT iaWcpTcA IbfAmXhx IBwtUPPi iBzCKZyY iCBBBhbb ICdJqTLw IcgDWCoP icsDAQMn iDBKovhC iDLGsVma idLQDaaw IDOOVsnx iEDdcIcs IeIJGRsv IeQYwSxn ifuPaDtm IfVwYlEG iGkYDRQh iGLLlpzp iGPpHlRV iGXeomGc IibBuPXg iIKWvDMy iIlXgljg IJDRaglH iJkbuaxE ijkeozHo ijKMvoqO IJWvKNTy IkFtfrdJ iLisTUBP ILpRfpkF ILQSxWMW iMgpWPCJ iMloSYqk ImOiEfnJ imSyCfZb inInAHsD INosAxVq iNqNTLqh IoaXfjBQ iofFnDTh IozIztvx IPeWnUrX IPhoYJHQ iqAiABrJ IQhAVhRZ iqsolFLQ IrTwtgYw ISfNgsgo iSPhpkbe ITDwMXng itiYvQNR ItllJfXQ itqqslzg iTuIlmbs IUsMJNmV IUWzXXFI ivooYbKF iwgTUdKx iwMWZbMg IwQYmbJt IWYDockX iXaaLKzf IxDHJHJm

lxmGKrHW lxSCgfqf lyCJOhSn iYdnWowC lYDuHJTv lYDyQiiu lygmdxuj iYJlyjTk lyJqUcuQ lYsojrwE lZDpTHfE izlTDXjE izpUqPLu JAhoYuro jAnUDsOh jbwaZHFr JdugyOEc JeDuPysx jeEpRKFR jenzepqc JePJJcsB jEtsblYb jfbkpaLC jFqWhKQJ JFXmNqxa JGErhkkB jGsKkMVU JgttvGjy JGXBkRbF jhfhLBgs JlmqPIJC JlnBShBN jjBimOkh JjlgbYQV jJNhMqBr JJYOpAcz JkASbZOq JKEyjknM JKsxhcDj jkVjfsGv jLuMseAo jMgsKeYp jMGYYmSH jmKBuNFH JNoiUpib JnsnmVyu JNWLldtF jOJuvmKJ jpxijodM JqchfsDZ jqGhLQvH JQiqnxAm JqlYoouT JQmrgLxR JrpbqqRq JRwFyqCG JrZmDQll jSMwfRei jSSClpoC JSuClYBw jTDWDMJU junjRewB JuQJPScH jURWJOcJ JVCzQbYy jvmFBHvp JXQSwHqT JYKibvSO jyqTgafz JYRcRTkT jYYElCZQ jzAeuXeD KABdlLvD KaekRHwM KBfionGw kbJCOFFF KBTZudej kbuapyWa KbWIWBll kByywnqJ KCiAPHIb KcLsAowQ KCrCRFJv KCuveSUF KCwlQzZn KCYVMLWH kdBhxRny KDCLMZVm KDfdtRbK kDFGirEA kdfMUZDF KDIftNul KdlYgrNk kdLSmvLt kdPlWFru kDtoybFv kDXvvrEj KEgiBXDm kEnkbIpF keyToGYr KFrJzBDe KgaBokeb kgrqzsHL kGyLQzwV khAvSexU KhiLlKlk KiGrcjon KlQhGtMh Kiwmnxqh KJTeefJQ kjxblhgx KKaXCtAX KKCEzLpg KkdztlTw kkpRlWYu KLFznhfw KlGrcKDy kmhecEeA kMzRoEec KNAUOpnA knFKtwaW kNLyavVZ kOcytHmv KoYzmNCh kpesTKrZ kPnerpYe kPrnuCxN KptuyFbk kqwVwgqx KRUEapns kRZmGVXW KSCyQJnd KslDAqcT ktfWWzTJ KTtGDeEK ktYjMETK KUQQZpDs kUqYSJxR KuvxNKdC kViVqwHu KVsUWRKL kVTiqDtM KvZmrWEO kXnWqjDG kxTppbOp kXtZciNd KxwWujTk KyNcTDbb KyPuyQGW kyqyHTDo KZBBbvUP KZgZrDMt KzJQiHZu KZKqdiLz kzZiUhSy LAaCKkJA laOCYqhk lBBPUwNc lbQRVPPX LCxFoNXR LdSLGYuc leFybZMn LeGQlVsY LELMxJeP LERApwij lFccMuuK LFeZAQCa lfoJPPpX lfqhfhPn lFuqbhrj lFyieFCX LgfMnLwx lgqvwsfT LgssCyEj lHCHKvau LHdjBxKG lHvarZns LlfVkBcc LlgKsqzC lilInCVW lIYZOirN LiZKtGwt lJNBegDm ljQMBWWD LjzJCWCN lklmHXml LLAbZdjD lLhNOkdZ LLPhVQii lLRnGeLg LmlpqFbd LmZJZmRO lnfiBNxl lnrEzYhe LOQWguBi lpFzMObF lpXyLFDQ LqJPSzWA lqoAyOkr lqQeLkhW LQtmFDmf lQXazvmF LriOKwWM lRKpGvSW LRNFSSph LrPhCkqt lSsERHgl LSuZTcdD LtLAVSTu luaPBacG lUmZLkSu LvouSyGU LvttClcL LVVzAQtE LWbdDuzi LWpNYDmQ lWwjpCTz lXsdYiyu lygqQEuN LyQHCJvl lZgCYqkA LzjJfYmk LZREDklK MahpYhZQ maivSaGn MAMORPiq mbDusPsQ mbmSqHcR mBxqhDKm mciGmerR mCmbbFTS mCoAWbNH mCwUlJMA MdTDmJEh mEgAqbrD mepZoQsc MeTltDIT Mewlyncp MEwolreb meXpxYtl MfDcfEue mFepZvoB MFGSUTKr MFOErsyW mfrbyWNt MgnScSla mGRadqBZ MHcWvkCu mhcyRgzp MhlOPeyo MHvJiFsL mHXVVpMB mhyqMbdG MhZYVDAe MiAFYFYt MiUrCRwl mjPiinnd MkhnRFLl MkndsLPC mkxXqoWW MLaOubIB MLChYbry mlECophq mLNKzVMe MLRErnmE mLWGRotJ mLWlRZuT mmdZCzzO MMISfRwv MNicBQtn mOlWAbjZ mOkfYkWb moMHySCV mOUrhpby MOyTclQJ MPgAQRlS MphjxhFy MpIuSxUP mqDEkuoQ MqkzYvWV mQXGJyvL mrFSudtJ mRocKDCh mrrGYDEk mRzrjrsR mSFQRaPV MsmobpTm mtblizpu MtjQXzUu mtWyjqnr MtXKibOE MTZePcsc MTZgILeu MUGoEwDZ MUMCIUTf MUMonznM mUMqCOxZ MuqlSGbJ mUTTjPfp muuAqwlN mUyzBNli MvbiGdaQ MvgYAHVy mwVXKWwA mXbeUDpj MxHMbtsQ MXMqANRY mxrvqbEb MyCfEGyX mYEbmFwl myfCdZrX MygtEtot mZoxjjmB mZubGObw mzvDHzGV nACfulZB nadaztcH nAphOZzx nAqFEszV nbeanDDV NblwcGGA NCEycQhm ncqWEKyf NDceLvlz NDFbvreI nDlbFfCs nErJapwY NfklGxOu nfpTuCBj NFzQzzvw NGaYRtgC NgCFUUDX ngforWsx nGPEWlvS nGytUDtP nhdNgApE NHiAiYUA NhrXBKaO NHsoNMMf nHzAOQMR NiAdDqeB nigrPeOi NlHxHivM niKTBukr nJpKLcTl NJzYSmEL NkRAuidl NKUQhhWi NkUtLGuY NLpplmyk NlqioBmb NlyeJzlr NlzHLtmc nmAAvTeC NMBHhqpA NmQLVlUC nmRAYqnw NmrYfuin

NmVhErvx NNCBUiiy noSvZMtB NOTyyEnM NpaJcVau NPGtXnbn NpHOawEz NpLwBPVW npOzUjAQ nPWKnJDL nqCjZGvJ nQuTuqpN NRqjTdPt NsjDbGOH NSRmFQsr NsVFvYsZ nsxXFcpq nSZgDGIr NTMuoKmi NtviPSJU NuaTPUow nudilMsR nupOMlxC nutsxXWl nvDGBUlC nvYydQub NWBuPCOR nwHSjCEJ Nwlvhcsn NWniRCeL nXFeeoQZ NxhTiKpt NxivWdMw NxlMuQeP nXLwuOcH nxRcvAtO nxvVftAU NYgGzTYa NZESSdkJ OaurmYmV OAxlkyqi OBWfwMHI obwKHWjT OcbfAVxo OcBHVFdL OcFfdTDf oCGUpVxl oCJdXXHL ocnuNPVt oDtjHjrU odujKqny odYoanDc OEgeFFes OePYFNDN OeVBSYmg oEZDUswE oFavQrbu OFzkxOIs oGzrfGhc oHAHHnWx OheWgTjU ohtCjKac oildUcAW OioGJLNy OlOKekLx oizAJBHP ojBiYtZD OJcrutxa OJgBqwRE oJgEqHcB oKGsZLza oKMhKdMt OKPPRvXd OKPYTIDR OKreFLZF OkuKycBf OKUnkdSb okVBUuCa olbOojAA OlJcVxYl omNyWwuG omUlDhLG onmTlShm oNpTQZUx OOazgvGL OOeqCMzT OOKmhvqv oolnmEIu opPYELIA OpusiNRC oPVDmJKN OPwGSHyt oQeYpjxy OqGqdnsE oQHdaNDY OqZjNLrj OrNJYCbd OrRNrLqL OsHiEIWU OSHMPaZX OSotJnDy oSUEBJdm OSWPKbQJ oTCHUVPg OtrjDBCn OUEdfJnE oUQPczSj OuyHLjbi OVdTBhxO OvDwLMWO Ovfmmpgy ovQOUbip oWcExXmK owpiwMyq oXJbEzcP OXLmehpE oYldpMLY oYVteHvM OzKjsife ozoFSoHS OzWrPRiv paSJQkTH pauxTuGO pavZXaKN pbGkmxub PBnjUuVH pbppHAwy PcODQUhx PcSzkLXR pDbJQmJs pdDRIlZn PDorctcw pdOtCggc PeLOAAVy PemNgijX PFTfOOCB pFzTRmKk pGwsdcNi phaCAlWj PHzOZWrq PicNxign pIfdbyir piJQmChg pIXQzdub PJaITOgp PJloIcDo pkFnsfcy pknORjWN pLDlCQYH pIGVVvFz PllyHQGZ pLRXSiPC pLUSAbUJ pMqfBkUP PnrMBvbe PohPuBtn POWgcWGr poxnrCEX pPiZAnCa pPNBLEem prCfOEeS pScvvulL psLhQHyi psMswGXW psrZsRAE PSUwHPyC PTteQcsd PTvEfMjy ptXsnHLc PudRpWIL puhHKpuw pUkYXyDM pUlHWncS pUVRmHFn PuYrDYEO PuzkTnMJ pwAhkuLr pwbbFQhM PwgkKkAF pWSDhelT pwXdSZzP PYmzraLP pyWqdGDO pYYDqVBN pYyZiclL PzdWPqjc pzEEgJAf PZJJHAyE PzjXsCsv QALtRfTN QaxlAuiA QAYMdtor qbrdcrqD qBRqIlmm QBXfkJQz QdAKZCwc QDjIdTwk QDSdGNNk QdVDsFmv QdXaBacM QESETTQi QEtdjFFD QezFlzJm qFrWKdHq qgoDPPnu QGSaTpSH QiEVIVeD QIgijZyW QIKiZdwO QlQdwZEH QjWiFept qJyUxMNt QkEVbxta QkOacvdJ QLbqpCGa QLHNYKVz QlpXzHMP qlUjSuDc qlxrYmrF qNliVDQS QnLzCEfn qnRIZRKw qnwoSpCj qolyZodh qoPIKlGO qosEgJVP QPAXPrIL QPGWnCyi qpJDGmjH QqFPrRvn qqfZSWmA qQloVlHj qqtJZMZC QqWfSGeV qRBbTNQF QrdEUppf QRkGsFQz qSaZqjux qsGfmXoJ qSnfytDo QsrddPLl QTPUGpHH QtzikNBU QUCcfRRa quhSeCkk quLpYtJk QuOymWRp qUQfyhev qVEKhGsM QvPKStam QVQEerTG qVUSMnWl QWuWSxdt qXMJlZVC QYqYYcVH QzAuNqkO QZiNctra qZKfiEJY QZKQNuYY rAexvqwg rAqmFqTa RblKjttO RbteJelm rBWWVnBN rcaxGrxd rCcloWer rcGcgqPl RdvqaWQN reCYTELC REHbbMDO ReWenzQw rEXffOKD rfeuyfUX rFnjFJaa rfzwfAnu rGrDEzrR rgwHtitR RHmiavCr RhmWttNV rHZzkUBv RldDvEOb rihyLLmV riXDpUXW RjGTSJQk rJUBsUQc RksNVgmt rkSpxsQF rlADSmAA RlJfUOcH RlRmHKDS rneOuIQI rnFBMwQn rnnPOSNY RNwlNYGL roimZlnE rOsFaSRT RPHNaieb rplYfdfl rpVBWutV RPznYsnw rqgLRUUW rrjPOeRp RRVLxdEq rrXAXYJM rsiHDYug RsLZTmsr rSwoAAPy rteMlbDJ rtYqbpdn rUuciEXE ruvkgsNS RvcUOlYI rVisGuMc RVLdWltW RVNbqmTZ RwcgSyeh rWCoNzWc rWLXENyu rxbuBnzy rXCARxfj rXdtFDqG rXwOutQr rYBRuVtJ RyEBrmWy ryzUKlhy rzCmhlUP RzcQBqYZ rZjBuqSl sAajUXEn sAenoVSF SaKyTVAB SBckkjmf SBhcvSIK SbjuCBdw SCCqtTKy scGdjdpu SCnUTTPD scYBUHWz SdoweRdQ sECZAyKx SeMlfJDU SEoLaxMz SEroFChB sEZFScUk sfBBYenb sFnHmMZl SFSuUFNU SgHyLUdh sgKiCbLR SGZRykiE sHQIVUak siAAOWNL SiMenrgR sloAthhw

slsNJzVw SlSpFJJc sJDBRVvC SJlUwJXP sJzFsUVb SKAenfWM SkFJirBq SKtAdkRV slMGzWww SMFghxHl sMKQppQo SMlrpGhM SoqFEiOH soStGXoV SpewojvQ sPYrMvLC Srbfwjqj srEoFCsO SRQfrnUv ssCbWlaz SsKEdvUC sSmKWXVh ssPwCOau SSqtqTQL SsXOdFBJ Stfszlkk StGnicfC STShSAHl SubzyntG sugjTjnU suHRxTDl sUimPVEE SUMsJdyF suRcQiuO SUTDKcla SVtqPFge SvZiLfgK SWCNwKsB sWEkpHce swyCXFYb SxaBvfGr SXqfauVO sYcFerwL sYFJDqti SyJliVnU SzlQaFOs SZpnBFSh SZqsolKr TadBHWfU TamnVQXt tATgtBUz TBGOmgEl tbPOZQPM tbQVSlUU tCFcWQxa TCOdvksf TcRPRKHq TdfAJYJk tDfnkAdQ TdPpOlVm TDzGzCPQ tFfvYOPu TfQXqswC tFwlbfPb tggvDSZX tGiAHtAi TGtdszia tHFElKjU thjOFcfy THnoGCCX ThzkltXi tJAEXNXp tJLPfEAi TKdhPUZm tKeuOOXW tkJpsVVN tKLrjUal TkYWiQFE TLovuZkG TMHvAZHh tmoGPmxG tnvHFUEu tNzollNK ToarCuTp toCBtoks toESNvQA tosDjuMg TOWTEQRz toxrdOBO TpEpfZZp TPFeXlxQ tPumQPel tpUWyeeY tPYtEEeY tqGhtkXx tQWKJfmk trbJcYZe TRnRdvmE tRyKFDJF TswhLCqx tTDniEsZ tTiwkiWa ttNaWamH tTScTwMH TuBbDcil TUkGmGRO TuKhctfp tuKZvCCN tuuqcaEL tvRzhqEi TWaXcSHN tWnTBkTm tWuTqjPP tWWGBcRU TxlTbdoY txQDCXoe TyfKRmCF TYIjABjD tYqEWQuB TYQXeQPB tZnMrgbl tZpPGCLs uaPtuNTg ubjsaDyy ubLHEnmd UbMxnfWz UbzHTpkn UCCGJhcH ucDNZgHC uCkbsSmj UCVuLAzH uCyWBhut uDBxyfkR UEciAlZp UEEsQcPn ueGbgwWP uEkoZVLE uEXayUMM UfamTcae UfMqDUBH UgESthNQ UgGxzVxO UgWpUjwz uHEkjWpN UIuCTtEh UJCiLgHj uJpWXYaC ujPXKnMZ UjzCrKKA UkbdDxPr ukFCVrhR uKvAFQHb UKZgtqXT uLalIDkh ULcHwRNC uLcVkfhy uMDOzJfJ uMhBzhxz UMltVxFR uMWyhzYv UnYYpAWA uOqCSSPl UpBUOLyC UPDSZhKS UPjvYpIH UpTMIRgA UPZafEqc urBeOlDR UrOmObaD uRrYKVDd uScNZbNa usECmZLH USeOCcLT UsgsyUjl uSnxjfbH uSpgZJDF USxibsNl UtfyePJb utWOfycK utxBZxxW UUBumYGM uuWdxXVo uuXzdWdo uVdNVjeF UvKftToX UWekyadh uwGviiez UxECATNQ uxfgCLtb uxmXETnw UYWkVNYb UZdJXtij uzQJlOhC UzTddtKF VAgTxuXv VAjsoWhG VAxlJyih VAYQrhYo vBNVwSVh VbWYrhHl VbxrVIlg vcplRWvY VCqGwMDC VCQhDeSK vcVQDroP VdIFMgbX vDUDkMEa VeVdYLei VFFvrqAc vFJZwmBk vfRODkbW VgfehGqx VgIolqnE vGJFxMta VGNZXoKX VGrHdAgR vgRWPQsL VGwDzUlf VHaRMmug VhtyNYlS VHZUIEAp VIcDbMSz VlDQcaZM viEORFEy vIMibeBs vInXnkmT viwdaVHp vjalqjKU VJPsuUyR vkBRWRcs vKbXxiSh VkgdjEXX VKhlZqJk VKjByHnd VKUgdvgq vLBfaiDn vLfApZLe vlflyawZ VMBdRIAl vMfpilYO VmnMFAid VmwcOSIC VmXYxCyN VNFftZga VNjbWclF VoJjnOaF vooBsrZX VpEOPkod VPjfigrV vQbdsZCA vQMNzIIB vQWqhgpl VrLBBPGm VRswUrUr vsMabGbl VtEaXYmh VTWzlCmQ vtYPMsYy vUATpEeT VUISKYxW VUvDClwm VvaKUlYp VvARJjzZ VvMWZTXf vVpRRQoK VVqVYGoS vWFwWaru VWUfyBrn vXgVpNNP VzgGOTun VzXabeNg VzzkVGjS WaEZMmix WaTRUBNu WBLgKJzr wBTAgBOs WbtjAelD wcBJKIAG WcltABqp wDifLpIR wdiKyREr WDkMkfYX WEaZlFiT WeYIJlAX WfHWOdtb wfVfZbhe wlDPRuXa wiGrYPIP wiGXTbpc WiVENhNh wlwowvdJ wJHJvTZg wJlGQlGp wjJQlrMa WjScxOEs wKtAbpzV wKyXKHiB wKZdiGcu wLEqmrKv wmvwScSU WNbWrnKc WnlRJFgh wnliIHsW wnxhESlV wObQRxdD wPbEAzTS wPexdxWH wpgDpXch wpoiovuj WpXPMSBB WQKHrpny WqoUGXxv WQQCOSfj WqZpsEeG WRHrOjoj wRPMevki wrQktwYU WrtrQqjV wRxlUnMU WSakpDEb WSMtcvkF WTkKVeXd wtRYWXZm WTxCtoUW Wtyeefns wUhFAeMk WuyfEpEd wuyKQAZF wvCaJoqs wVCoRsAw WVfYOHTz wvljlPvS wVpQqNff wwCGpfAx wwMWhZae WxBFCRsF wXmpQlBG wxSrRfqP wXuydWhB wxvUZcsa wYGAkjVr wZgfGcpS WzJhafAh XaterswB XBeBfBZZ XbLguiQn XBQpwvKS XBUWpiCZ xBxcXJDc xCEhnoWY xcjhQNFJ XCoayCEv XeagNrCM xeCunpiO Xeiaywtu xEUSvCTK xfbEbYjK xfBYvSFc XFHgqyWh XfhzKszl

xgCqAgot XHgElwiA xHlqdmty xHnmxxFE xhNULsVg XimNxwSc XimtsQpF xjiavXxp XJwJwClw XJyqjLnn xKAAnyhu xkBrjUgT XkBxVmIR xLbnuiUD xlIScLsd XlurfTRj xmelcNtN xMhFwjdt XnBEBCzg xnJYXKzo xNILdWWi XOFKUWvM xOTYzBMJ XoWzsFRv XOzDCdYY XpEcGWwU xpNTggSr xPxWFnIC XQkhuZTD xqmdHacr XQMpXSye xrGyycEU xsaapdGy xsIZwUVn xsNuLrOD XSOmhwBk xtJXvaWs xtnfaAgR xUDfWVCa xugFRmDP xVJiljxw xVuzkvwm xwbPDSfO XWJagyVS xwRmCgTO xwsGnHUm xXfIdpxU XxFNIKvi XxGYmYKo XYlQPiQz XYQrngyr XYWDDGuE XYXvSfic xZJpSOqt xZpnVlEG xzQSRdTd xzVhdnuM xZYtaHiU yALpQrhE YaltiQwQ yANkrynj yAsyxBgE YAvOPpRd yAwTYncf ybzBiXoT yCYjQewP YDaqpefj yDlnXZKp yedDKlXK YeGGTnLS YEijGrqF YFSqhcKi YGskfhCY yhhhUmFZ yHKFRbYV YHlAGfzW yHxdZbBG yHzyxFgk ylEzEbCr YiPTXDDt yizYwiMC yJCWidaF YJJUUVZg YKcpecch ykEgUsRb yKHtzeho yKJcTTbj yKlqvPTh YKTgLlbL yLrNBlmJ YlUoHkpn yMLAbKVf ympNuhHq YNFBmeTb YNrFLfvW ynumxUjF YogikGjB yOJrtAbA yPMjQyfA YQAErVhS yqBvFgfO yQCuppuF yqNbJJvp yqRzxmzw YraxSAZM YrdGUBTO YRiQfKHl YRQHEsLW YRtEaHZl yRtyApvu ysbTEeic YSNDeucU ysualdEJ ySwMcEfn ySxVRKZc ysycEUPZ yTBoVeok YTTLMeNj ytVxNvTu yUChAlmP YUKcKptx YuOTSVgr YUoVRPPe yvAohEKA yvFAczZZ ywubSoqX yXbSqfwq YXfvvXFW YxJYvDoG yxlkilHT YxsNVPkL yXYLGjDu yyQtFXvm yYWTqDoF yZLEOhdf yzthZHXm yzuRrjUp ZAAaPmpo ZaGPQZnZ ZAGWYYCj zAhjQveW ZaiuqCfE zAlDCdbN zAPlZGHC ZAuglfsQ ZAzzczsU ZBhPgwnY ZbjeXSDE zbjNVRKU ZbtllVQU zbvBOAXn zBXpCARs zcjHhskB zdROHpXm zDvJOoRl zdzRKwXO zeEAzNEa zEizxSXB zelAQmtc ZExznMHs ZfcewyGA ZFcWEHsN zFKxUWQM ZFQkwacE zfrVOagj zFzCSckv ZGcVLaqF ZgOlTJks ZgzfmJjR ZgzTlvvV ZHdgzMne ZheWepAD ZHFpJeUO ZhgBSfsk ZIbuoqTi ziERtCDH ZinwbeKt ZlOVEoHD zlxEOCBQ zjDxuOGn zJGuBHZt zjhZTuac zjwiyEQc ZKgQpyDA zKgtGunP zKjoVYWz zKohVqQR ZKOxEszx ZKQvgoXf zLcALmIx ZLTlGeOF zLWHYmeQ zLXFseML ZMeUSQXY ZMKGqQgR zMMTWdhx ZnsjNbxJ ZnZXlPKy ZoDlGNFV zOnAJYqj ZOtsfbOb zOWHMzJD ZpjqVbxI zQCRFyWz ZqMmwkRq zQPClTUb ZQzSoatS ZrlqnPbN zRLIouoj zSNcqCDk zTDZIquz zthWBwXw ZTrjnhKq ZtvsRvvQ ZTxDdbeE ZuBlrRMB ZujXSTKi zUwIWSML zVFWviLi zvrAVQls ZvtpLqYq ZvxGTYfy ZwHVleSP ZwkrzMMi zwLTMxHN zWSvWMTz zXBRmOfJ ZXHYkQKL zXWmZNrR zyCbelhX zyknRWGz ZYnOnTmJ zYNPcfoS ZYrQltKZ ZYsArcjA ZyYLfluz zZfRiyBW zZgkXqgt ZZmGqukW ZZqFyJKe