《罗瑶光极速小高峰缺陷过滤象契字符串快排算法》

10年研究历史记录备份

感谢：

1 算法导论前4代 与 印度基督大学 的数据结构基础研究理论课程

2 罗瑶林 帮我验证纠正了一次 等比 符号。

3 Mir 比较系统的跟我讲解了函数和变量名的命名规范。

4 曾培养过我基础教育的每一所学校和老师。

5 Java 语言之父。

Quick_1D_Sort

算法思想：《数据结构导论》

研发实现：罗瑶光

```
================================================================================
package sortProcessor;
public class Quick_1D_Sort{
    public int[] sort(int [] a) {
        int lp=0;
        int rp=a.length-1;
        while(lp<rp)
            quick(a,lp++,rp);
        return a;
    }
    private void quick(int[] a, int lp, int rp) {
        while(lp<rp){
            if(a[lp]>a[rp]){
                int temp=a[lp];
                a[lp]=a[rp];
                a[rp]=temp;
            }
            rp-=1;
        }
    }
}
```

Quick_2D_Sort
算法思想：《数据结构导论》
研发实现：罗瑶光
==========

```
package sortProcessor;
public class Quick_2D_Sort{
	public int[] sort(int [] a) {
		int v=(int) Math.log(a.length);
		for(int i=0;i<v*3;i++)
			sort(a,0,a.length-1);
		return a;
	}
	private void sort(int[] a, int l, int r) {
		int m=(l+r)/2;
		quick(a,l,r);
		if(m>l)
		{
			sort(a,l,m);
			sort(a,m,r);
		}
	}
	private void quick(int[] a, int l, int r) {
		while(l<r){
			if(a[l]>a[r]){
				int temp=a[l];
				a[l]=a[r];
				a[r]=temp;
			}
			l++;
			r--;
		}
	}
}
```

Quick_3D_Sort
算法思想：《数据结构导论》
研发实现：罗瑶光
==========

```
package sortProcessor;
public class Quick_3D_Sort{
	public int[] sort(int [] a) {
		int pos[]=new int[1];
		int lp=0;
		int rp=a.length-1;
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
		return a;
	}
	public int[] sort(int [] a,int n) {
		int pos[]=new int[1];
		int lp=0;
		int rp=n-1;
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
		return a;
	}

	private void quick2d(int[] a, int lp, int rp) {
		// TODO Auto-generated method stub
		int pos[]=new int[1];
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
	}
	private void partition(int[] a, int lp, int rp, int[]pos) {
		// TODO Auto-generated method stub
		int x,lp1,rp1,temp;
		x=a[lp];rp1=rp;lp1=lp;
		while(lp1<rp1)
		{while((a[lp1]<=x)&&(lp1<rp)) lp1++;
		while(a[rp1]>x)rp1--;
		if(lp1<rp1)
		{temp=a[rp1];a[rp1]=a[lp1];a[lp1]=temp;}}
```

```
            a[lp]=a[rp1];
            a[rp1]=x;
            pos[0]=rp1;
        }
    }
```

Quick_4D_Sort
算法思想：《数据结构导论》
研发实现：罗瑶光
==========

```
package sortProcessor;
public class Quick_4D_Sort{
	public int[] sort(int [] a) {
		quick2d(a,0,a.length-1);
		return a;
	}
	public int[] sort(int [] a,int n) {
		int pos[]=new int[1];
		int lp=0;
		int rp=n-1;
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
		return a;
	}
	private void quick2d(int[] a, int lp, int rp) {
		// TODO Auto-generated method stub
		int pos[]=new int[1];
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
	}
	private void partition(int[] a, int lp, int rp, int[]pos) {
		// TODO Auto-generated method stub
		int x,lp1,rp1,temp;
		x=a[lp];rp1=rp;lp1=lp;
		while(lp1<rp1){
			while((a[lp1]<=x)&&(lp1<rp1)) lp1++;
			while(a[rp1]>x)rp1--;
			if(lp1<rp1){
				temp=a[rp1];
				a[rp1]=a[lp1];
				a[lp1]=temp;
			}
		}
		a[lp]=a[rp1];
		a[rp1]=x;
		pos[0]=rp1;
```

```
        }
    }
```

Quick_5D_Sort
算法思想：罗瑶光左右比对法
研发实现：罗瑶光
==========

```
package sortProcessor;
public class Quick_5D_Sort{
	public int[] sort(int [] a) {
		quick2d(a,0,a.length-1);
		return a;
	}
	private void quick2d(int[] a, int lp, int rp) {
		// TODO Auto-generated method stub
		int pos[]=new int[1];
		if(lp<rp)
		{partition(a,lp,rp,pos);
		quick2d(a,lp,pos[0]-1);
		quick2d(a,pos[0]+1,rp);}
	}
	@SuppressWarnings("unused")
	private void partition(int[] a, int lp, int rp, int[]pos) {
		// TODO Auto-generated method stub
		int x,lp1,rp1,temp;
		x=a[lp];rp1=rp;lp1=lp;
		int m=(rp+lp)/2;
		int y=a[rp];
		if(x<y){
			while(lp1<rp1){
				while((a[lp1]<=x)&&(lp1<rp1)) lp1++;
				while(a[rp1]>x)rp1--;
				if(lp1<rp1){
					temp=a[rp1];
					a[rp1]=a[lp1];
					a[lp1]=temp;
				}
			}
			a[lp]=a[rp1];
			a[rp1]=x;
		}else{
			while(lp1<rp1){
				while((a[lp1]<=y)&&(lp1<rp1)) lp1++;
				while(a[rp1]>y)rp1--;
				if(lp1<rp1){
					temp=a[rp1];
					a[rp1]=a[lp1];
```

```
                        a[lp1]=temp;
                    }
                }
                a[lp]=a[rp1];
                a[rp1]=y;
            }
            pos[0]=rp1;
        }
    }
```

Quick_Luoyaoguang_3D
算法思想：罗瑶光左右比对平均高峰过滤法
研发实现：罗瑶光
==========

```
package sortProcessor;
import timeProcessor.TimeCheck;
//第三代罗瑶光小高峰平均高峰过滤快排思想设计中。小高峰高峰过滤快速排序
//同频函数减少
//同频算子减少
//同频变量减少
public class Quick_Luoyaoguang_3D{
	public int[] sort(int[] a) {
		TimeCheck imeCheck= new TimeCheck();
		imeCheck.begin();
		quick2ds(a, 0, a.length- 1);
		imeCheck.end();
		imeCheck.duration();
		return a;
	}
	private void quick2ds(int[] a, int lp, int rp) {
		if(lp< rp){
			int c= rp- lp; if(c< 7){ int j;
			for(int i= 1+ lp; i<= lp + c; i++){
				j= i;while(j>= 1+ lp){
					if(a[j]< a[j- 1]){
						int temp= a[j]; a[j]= a[j- 1]; a[j- 1]= temp;
					}
					j--;
				}
			}
			return;
			}
			int pos = partition(a, lp, rp);
			quick2ds(a, lp, pos-1);
			quick2ds(a, pos+1, rp);
		}
	}
	private int partition(int[] a, int lp, int rp) {
		int x= a[lp]< a[rp]? a[lp]: a[rp];
		int lp1= lp;
		while(lp1< rp){//我总觉得这里可以进行一种积分算法优化，我一直在思考，别让那么
快想到。
			while(a[lp1]<= x&& lp1< rp) {
				lp1++;
```

```
            }
            while(a[rp]> x){
                rp--;
            }
            if(lp1< rp){
                int temp= a[rp]; a[rp]= a[lp1]; a[lp1]= temp;
            }
        }
        a[lp]= a[rp]; a[rp]= x;
        return rp;
}

public String[][] sort(String[][] a) {
        quick2dsString(a, 0, a.length-1);
        return a;
}
private void quick2dsString(String[][] a, int lp, int rp) {
        if(lp< rp){
            int c= rp- lp; if(c< 7){ int j;
            for(int i= 1 + lp; i<= lp+ c; i++){
                j= i; while(j>= 1+ lp){
                    if(Double.valueOf(a[j][1])<Double.valueOf(a[j- 1][1])){
                        String []temp= a[j];
                        a[j]= a[j- 1];
                        a[j-1]= temp;
                    }
                    j--;
                }
            }
            return;
            }
            int pos= partitionString(a, lp, rp);
            quick2dsString(a, lp, pos- 1);
            quick2dsString(a, pos+ 1, rp);
        }
}
private int partitionString(String[][] a, int lp, int rp) {
        String[] x= a[lp]; int rp1= rp; int lp1= lp;
        if(Double.valueOf(x[1])>= Double.valueOf(a[rp][1])){
            x= a[rp];
        }
        while(lp1<rp1){
            while((Double.valueOf(a[lp1][1])<=Double.valueOf(x[1]))&&(lp1<rp1)){
                lp1++;
```

```
            }
            while(Double.valueOf(a[rp1][1])>Double.valueOf(x[1])){
                rp1--;
            }
            if(lp1<rp1){
                String[] temp=a[rp1];
                a[rp1]=a[lp1];a[lp1]=temp;
            }
        }
        a[lp]=a[rp1];a[rp1]=x;
        return rp1;
    }
}
```

Quick_Luoyaoguang_4D
算法思想：罗瑶光计算小高峰过滤法
研发实现：罗瑶光
==========

```
package sortProcessor;
import timeProcessor.TimeCheck;
//第三代罗瑶光小高峰平均高峰过滤快排思想设计中。小高峰高峰过滤快速排序
//同频函数减少
//同频算子减少
//同频变量减少
public class Quick_Luoyaoguang_4D{
	public int[] sort(int[] a) {
		quick2ds(a, 0, a.length-1);
		return a;
	}

	private void quick2ds(int[] a, int lp, int rp) {
		if(lp< rp){
			int c = rp - lp; if(c < 7){ int j;
			for(int i = 1 + lp; i <= lp + c; i++){
				j = i;while(j>=1+lp){
					if(a[j]<a[j-1]){
						int temp=a[j];a[j]=a[j-1];a[j-1]=temp;
					}
					j--;
				}
			}
			return;
			}
			int pos = partition(a, lp, rp);
			quick2ds(a, lp, pos-1);
			quick2ds(a, pos+1, rp);
		}
	}
	private int partition(int[] a, int lp, int rp) {
		int x= a[lp]< a[rp]? a[rp]: a[lp];
		int lp1= lp;
		while(lp1< rp){//我总觉得这里可以进行一种积分算法优化，我一直在思考，别让那么
快想到。
//			while(a[lp1]<= x&& lp1< rp) {
//				lp1++;
//			}
			while(!(a[lp1]>x|| lp1>= rp)) {
				lp1++;
```

```
            } //今天想到了一些优化，
            while(a[rp]>x){
                rp--;
            }
            if(lp1<rp){
                int temp=a[rp];a[rp]=a[lp1];a[lp1]=temp;
            }
        }
        a[lp]=a[rp];a[rp]=x;
        return rp;
    }
}
```

LYG4DWithChineseMixStringSort1D
算法思想：罗瑶光混合字符串排序
研发实现：罗瑶光
==========

```
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort1D{
	Map<String, Boolean> find= new HashMap<>();
	Map<String, String> pinyin;
	public void quick4DChineseStringArray(String[] a, int lp, int rp, int scale
, Map<String, String> map) {
		this.pinyin= map;
		String[][] kernel= new String[a.length][3];
		for(int i= 0; i< a.length; i++) {
			kernel[i][0]= a[i].toString();
		}
		processKernel(kernel, lp, rp, scale, 0);
		for(int i= 0; i< a.length; i++) {
			 a[i]= kernel[i][0].toString();
		}
	}

	private void processKernel(String[][] kernel, int lp, int rp, int scale, int point) {
		int rp1= rp;
		if(point> scale) {
			return;
		}
		processSort(kernel, lp, rp, scale, point);
		int i;
		for(i= lp; i<= rp; i++) {
			if(kernel[i][0].charAt(0)!= kernel[lp][0].charAt(0)) {
				rp1= i-1;
				processKernel(kernel, lp, rp1, scale, point+1);
				lp= i;
			}
		}
		if(lp!=rp) {
			processKernel(kernel, lp, i-1, scale, point+1);
		}
	}

	private void processSort(String[][] kernel, int lp, int rp, int scale, int point) {
		if(point> scale) {
```

```
            return;
        }
        for(int i= lp; i<= rp; i++) {
            Here:
            for(int j= lp; j<= rp; j++) {
                if(i==j) {
                    continue Here;
                }
                //que
                if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
                    if(kernel[i][0].length()< kernel[j][0].length()) {//长在上
                        boolean find= true;
                        for(int p= 0; p< scale; p++) {
                            if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                                find= false;
                            }
                        }
                        if(find) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
                    }
                }else if(pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                    String[] js= new String[2];
                    js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
                    js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
                    boolean change= processSortPinYin(js, 3);
                    if(change&& i< j) {
                        String[] temp= kernel[i].clone();
                        kernel[i]= kernel[j].clone();
                        kernel[j]= temp;
                    }
                }else if(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                    if(i< j) {
                        if(!(i== rp || j== rp)) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
                    }
                }else if(!pinyin.containsKey(""
```

```
+ kernel[i][0].charAt(point)) && !pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        if(kernel[i][0].toLowerCase().charAt(point)
> kernel[j][0].toLowerCase().charAt(point)) {
                            if(i< j) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }else if(kernel[i][0].toLowerCase().charAt(point
)== kernel[j][0].toLowerCase().charAt(point)) {
                            if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                                if(i< j) {
                                    String[] temp= kernel[i].clone();
                                    kernel[i]= kernel[j].clone();
                                    kernel[j]= temp;
                                }
                            }
                        }
                    }
                }
            }
        }
        private boolean processSortPinYin(String[] kernel, int scale) {
            for(int k= 0; k< scale; k++) {
                if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                    if(kernel[0].length()< kernel[1].length()) {//长在上
                        return true;
                    }
                    return false;
                }
                if(kernel[0].toLowerCase().charAt(k)> kernel[1].toLowerCase().charAt(k)) {
                    return true;
                }else if(kernel[0].toLowerCase().charAt(k)< kernel[1].toLowerCase().charAt(k)) {
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {//长在上
                return true;
            }
            return false;
        }
}
```

LYG4DWithChineseMixStringSort2D

算法思想：罗瑶光混合象契字符串排序优化

研发实现：罗瑶光

==========

```
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort2D{
    Map<String, Boolean> find= new HashMap<>();
    Map<String, String> pinyin;
    public void quick4DChineseStringArray(String[] a, int lp, int rp, int scale, Map<String, String>
map) {
        this.pinyin= map;
        String[][] kernel= new String[a.length][3];
        for(int i= 0; i< a.length; i++) {
            kernel[i][0]= a[i].toString();
        }
        processKernel(kernel, lp, rp, scale, 0);
        for(int i= 0; i< a.length; i++) {
            a[i]= kernel[i][0].toString();
        }
    }
    private void processKernel(String[][] kernel, int lp, int rp, int scale, int point) {
        int rp1= rp;
        if(point> scale) {
            return;
        }
        processSort(kernel, lp, rp, scale, point);
        int i;
        for(i= lp; i<= rp; i++) {
            if(kernel[i][0].length()> point&& kernel[lp][0].length()> point) {
                if(kernel[i][0].charAt(point)!= kernel[lp][0].charAt(point)){
                    rp1= i-1;
                    processKernel(kernel, lp, rp1, scale, point+1);
                    lp= i;
                }
            }
        }
        if(lp!=rp) {
            processKernel(kernel, lp, i-1, scale, point+1);
        }
    }
    private void processSort(String[][] kernel, int lp, int rp, int scale, int point) {
        if(point> scale) {
```

```
                return;
            }
            for(int i= lp; i<= rp; i++) {
                Here:
                for(int j= lp; j<= rp; j++) {
                    if(i==j) {
                        continue Here;
                    }
                    //que
                    if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
                        if(kernel[i][0].length()< kernel[j][0].length()) {//长在上
                            boolean find= true;
                            for(int p= 0; p< scale; p++) {
                                //左右
                                if(kernel[i][0].length()> p&& kernel[j][0].length()> p) {
                                    if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                                        find= false;
                                    }
                                }
//                                else{
//                                    if(!find) {
//                                        if(kernel[i][0].length()< kernel[j][0].length()) {
//                                            find= false;
//                                        }
//                                    }
//                                }
                            }
                            if(find) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }else if(pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        String[] js= new String[2];
                        js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
                        js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
                        boolean change= processSortPinYin(js, 3);
                        if(change&& i< j) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
```

```
                    }else if(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        if(i< j) {
                            if(!(i== rp || j== rp)) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }else if(!pinyin.containsKey(""
+ kernel[i][0].charAt(point)) && !pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        if(kernel[i][0].toLowerCase().charAt(point)>
kernel[j][0].toLowerCase().charAt(point)) {
                            if(i< j) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }else if(kernel[i][0].toLowerCase().charAt(point)
== kernel[j][0].toLowerCase().charAt(point)) {
                            if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                                if(i< j) {
                                    String[] temp= kernel[i].clone();
                                    kernel[i]= kernel[j].clone();
                                    kernel[j]= temp;
                                }
                            }
                        }
                    }
                }
            }
        }
        private boolean processSortPinYin(String[] kernel, int scale) {
            for(int k= 0; k< scale; k++) {
                if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                    if(kernel[0].length()< kernel[1].length()) {//长在上
                        return true;
                    }
                    return false;
                }
                if(kernel[0].toLowerCase().charAt(k)> kernel[1].toLowerCase().charAt(k)) {
                    return true;
                }else if(kernel[0].toLowerCase().charAt(k)< kernel[1].toLowerCase().charAt(k)) {
                    return false;
```

```
                }
            }
            if(kernel[0].length()< kernel[1].length()) {//长在上
                return true;
            }
            return false;
        }
    }
```

LYG4DWithChineseMixStringSort3D

算法思想：罗瑶光混合象契字符串小高峰过滤快速排序

研发实现：罗瑶光

==========

```
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort3D{
	Map<String, Boolean> find= new HashMap<>();
	Map<String, String> pinyin;
	public void quick4DChineseStringArray(String[] a, int lp, int rp, int scale
, Map<String, String> map) {
		this.pinyin= map;
		String[][] kernel= new String[a.length][3];
		for(int i= 0; i< a.length; i++) {
			kernel[i][0]= a[i].toString();
		}
		processKernel(kernel, lp, rp, scale, 0);
		for(int i= 0; i< a.length; i++) {
			a[i]= kernel[i][0].toString();
		}
	}
	private void processKernel(String[][] kernel, int lp, int rp, int scale, int point) {
		int rp1= rp;
		if(point> scale) {
			return;
		}
		processQS4DLYG4D(kernel, lp, rp, scale, point);
		int i;
		for(i= lp; i<= rp; i++) {
			if(kernel[i][0].length()> point&& kernel[lp][0].length()> point) {
				if(kernel[i][0].charAt(point)!= kernel[lp][0].charAt(point)){
					rp1= i-1;
					processKernel(kernel, lp, rp1, scale, point+1);
					lp= i;
				}
			}
		}
		if(lp!=rp) {
			processKernel(kernel, lp, i-1, scale, point+1);
		}
	}
	private void processSort(String[][] kernel, int lp, int rp, int scale, int point) {
		if(point> scale) {
```

```
                return;
            }
            for(int i= lp; i<= rp; i++) {
                Here:
                for(int j= lp; j<= rp; j++) {
                    if(i==j) {
                        continue Here;
                    }
                    //que
                    if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
                        if(kernel[i][0].length()< kernel[j][0].length()) {//长在上
                            boolean find= true;
                            for(int p= 0; p< scale; p++) {
                                //左右
                                if(kernel[i][0].length()> p&& kernel[j][0].length()> p) {
                                    if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                                        find= false;
                                    }
                                }
//                                else{
//                                    if(!find) {
//                                        if(kernel[i][0].length()< kernel[j][0].length()) {
//                                            find= false;
//                                        }
//                                    }
//                                }
                            }
                            if(find) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }else if(pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        String[] js= new String[2];
                        js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
                        js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
                        boolean change= processSortPinYin(js, 3);
                        if(change&& i< j) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
```

```
                    }else if(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
&& pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        if(i< j) {
                            if(!(i== rp || j== rp)) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }else if(!pinyin.containsKey(""
+ kernel[i][0].charAt(point)) && !pinyin.containsKey(""+ kernel[j][0].charAt(point))){
                        if(kernel[i][0].toLowerCase().charAt(point)>
kernel[j][0].toLowerCase().charAt(point)) {
                            if(i< j) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }else if(kernel[i][0].toLowerCase().charAt(point)
== kernel[j][0].toLowerCase().charAt(point)) {
                            if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                                if(i< j) {
                                    String[] temp= kernel[i].clone();
                                    kernel[i]= kernel[j].clone();
                                    kernel[j]= temp;
                                }
                            }
                        }
                    }
                }
            }
        private void processQS4DLYG4D(String[][] kernel, int lp, int rp, int scale, int point) {
            if(lp< rp){
                int c= rp- lp;
                if(c< 3){
                    processSort(kernel, lp, rp, scale, point);
                    return;
                }
                int pos= partition(kernel, lp, rp, scale, point);
                processQS4DLYG4D(kernel, lp, pos- 1, scale, point);
                processQS4DLYG4D(kernel, pos+ 1, rp, scale, point);
            }
        }
```

```
    private boolean findSmall(String[][] kernel, int scale, int point, int i, int j, int rp) {
        if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
            if(kernel[i][0].length()< kernel[j][0].length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(kernel[i][0].length()> p&& kernel[j][0].length()> p) {
                        if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                            find= false;
                        }
                    }
                }
                if(find) {
                    return true;
                }
            }
        }else if(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                && pinyin.containsKey(""+ kernel[j][0].charAt(point))){
            String[] js= new String[2];
            js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
            js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
            boolean change= processSortPinYin(js, 3);
            if(change&& i< j) {
                return true;
            }
        }else if(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
                && pinyin.containsKey(""+ kernel[j][0].charAt(point))){
            if(i< j) {
                if(!(i== rp || j== rp)) {
                    return true;
                }
            }
        }else if(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
                && !pinyin.containsKey(""+ kernel[j][0].charAt(point))){
            if(kernel[i][0].toLowerCase().charAt(point)> kernel[j][0].toLowerCase().charAt(point))
{
                if(i< j) {
                    return true;
                }
            }else                                    if(kernel[i][0].toLowerCase().charAt(point)==
kernel[j][0].toLowerCase().charAt(point)) {
                if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                    if(i< j) {
                        return true;
                    }
```

```
                    }
                }
            }
            return false;
        }
        private boolean findSmallWithTwoChar(String x1, String x2, int scale, int point) {
            if(x1.length()<= point|| x2.length()<= point) {
                if(x1.length()< x2.length()) {//
                    boolean find= true;
                    for(int p= 0; p< scale; p++) {
                        if(x1.length()> p&& x2.length()> p) {
                            if(x1.charAt(p)!= x2.charAt(p)) {
                                find= false;
                            }
                        }
                    }
                    if(find) {
                        return true;
                    }
                }
            }else if(pinyin.containsKey(""+ x1.charAt(point))
&& pinyin.containsKey(""+ x2.charAt(point))){
                String[] js= new String[2];
                js[0]= this.pinyin.get(""+ x1.charAt(point));
                js[1]= this.pinyin.get(""+x2.charAt(point));
                boolean change= processSortPinYin(js, 3);
                if(change) {
                    return true;
                }
            }else if(!pinyin.containsKey(""+ x1.charAt(point))
&& pinyin.containsKey(""+ x2.charAt(point))){
                    return true;
            }else if(!pinyin.containsKey(""+ x1.charAt(point))
&& !pinyin.containsKey(""+ x2.charAt(point))){
                if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                    return true;
                }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                    if(x1.charAt(point)> x2.charAt(point)) {
                        return true;
                    }
                }
            }
            return false;
        }
```

```
private int partition(String[][] a, int lp, int rp, int scale, int point) {
        String[] x= !findSmall(a, scale, point, lp, rp, rp)? a[lp]: a[rp];
        int lp1= lp;
        int count= 0;
        int lastCount= 0;
        while(lp1< rp) {
            while(!findSmallWithTwoChar(a[lp1][0], x[0], scale, point)&&lp1< rp) {
                lp1++;
                count++;
            }
            while(findSmallWithTwoChar(a[rp][0], x[0], scale, point)){
                rp--;
                count++;
            }
            if(lp1< rp){
                String[] temp= a[rp].clone();
                a[rp]= a[lp1].clone();
                a[lp1]= temp;
            }
            if(count!= lastCount) {
                if(lp1< rp){
                    String[] temp= a[rp].clone();
                    a[rp]= a[lp1].clone();
                    a[lp1]= temp;
                }
                lastCount= count;
            }else {
                rp--;
            }
        }
        a[lp]=a[rp].clone();a[rp]=x;
        return rp;
}
private boolean processSortPinYin(String[] kernel, int scale) {
        for(int k= 0; k< scale; k++) {
            if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                if(kernel[0].length()< kernel[1].length()) {
                    return true;
                }
                return false;
            }
            if(kernel[0].toLowerCase().charAt(k)> kernel[1].toLowerCase().charAt(k)) {
                return true;
            }else if(kernel[0].toLowerCase().charAt(k)< kernel[1].toLowerCase().charAt(k)) {
```

```
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {
                return true;
            }
            return false;
        }
    }
```

LYG4DWithChineseMixStringSort4D

算法思想：罗瑶光混合象契字符串小高峰过滤排序缺陷峰值理论优化

研发实现：罗瑶光

```
==========================================================================
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort4D{
    Map<String, Boolean> find= new HashMap<>();
    Map<String, String> pinyin;
    public void quick4DChineseStringArray(String[] a, int lp, int rp, int scale, Map<String, String>
map) {
        this.pinyin= map;
        String[][] kernel= new String[a.length][3];
        for(int i= 0; i< a.length; i++) {
            kernel[i][0]= a[i].toString();
        }
        processKernel(kernel, lp, rp, scale, 0);
        for(int i= 0; i< a.length; i++) {
            a[i]= kernel[i][0].toString();
        }
    }

    private void processKernel(String[][] kernel, int lp, int rp, int scale, int point) {
        int rp1= rp;
        if(point> scale) {
            return;
        }
        processQS4DLYG4D(kernel, lp, rp, scale, point);
        int i;
        for(i= lp; i<= rp; i++) {
            if(!(kernel[i][0].length()<= point|| kernel[lp][0].length()<= point)) {
                if(kernel[i][0].charAt(point)!= kernel[lp][0].charAt(point)){
                    rp1= i-1;
                    processKernel(kernel, lp, rp1, scale, point+1);
                    lp= i;
                }
            }
        }
        if(lp!=rp) {
            processKernel(kernel, lp, i-1, scale, point+1);
        }
    }
    private void processSort(String[][] kernel, int lp, int rp, int scale, int point) {
```

```
if(point> scale) {
	return;
}
for(int i= lp; i<= rp; i++) {
	Here:
	for(int j= i; j<= rp; j++) {
		if(i==j) {
			continue Here;
		}
		if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
			if(kernel[i][0].length()< kernel[j][0].length()) {
				boolean find= true;
				for(int p= 0; p< scale; p++) {
					if(kernel[i][0].length()> p&& kernel[j][0].length()> p) {
						if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
							find= false;
						}
					}
				}
				if(find) {
					String[] temp= kernel[i].clone();
					kernel[i]= kernel[j].clone();
					kernel[j]= temp;
				}
			}
		}else if(!(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
				|| !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
			String[] js= new String[2];
			js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
			js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
			boolean change= processSortPinYin(js, 3);
			if(change&& i< j) {
				String[] temp= kernel[i].clone();
				kernel[i]= kernel[j].clone();
				kernel[j]= temp;
			}
		}else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
				|| !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
			if(i< j) {
				if(!(i== rp || j== rp)) {
					String[] temp= kernel[i].clone();
					kernel[i]= kernel[j].clone();
					kernel[j]= temp;
				}
```

```
                    }
                }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                        || pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
                    if(kernel[i][0].toLowerCase().charAt(point)
                            > kernel[j][0].toLowerCase().charAt(point)) {
                        if(i< j) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
                    }else if(kernel[i][0].toLowerCase().charAt(point)
                            == kernel[j][0].toLowerCase().charAt(point)) {
                        if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                            if(i< j) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }
                }
            }
        }
    }
    private void processQS4DLYG4D(String[][] kernel, int lp, int rp, int scale, int point) {
        if(lp< rp){
            int c= rp- lp;
            if(c< 3){
                processSort(kernel, lp, rp, scale, point);
                return;
            }
            int pos= partition(kernel, lp, rp, scale, point);
            processQS4DLYG4D(kernel, lp, pos- 1, scale, point);
            processQS4DLYG4D(kernel, pos+ 1, rp, scale, point);
        }
    }
    private boolean findSmall(String[][] kernel, int scale, int point, int i, int j, int rp) {
        if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
            if(kernel[i][0].length()< kernel[j][0].length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(kernel[i][0].length()> p&& kernel[j][0].length()> p) {
                        if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                            find= false;
```

```
                        }
                    }
                }
                if(find) {
                    return true;
                }
                return false;
            }
            return false;
        }else if(!(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
                || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
            String[] js= new String[2];
            js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
            js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
            boolean change= processSortPinYin(js, 3);
            if(change&& i< j) {
                return true;
            }
            return false;
        }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                || pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
            if(kernel[i][0].toLowerCase().charAt(point)
                    > kernel[j][0].toLowerCase().charAt(point)) {
                if(i< j) {
                    return true;
                }
                return false;
            }else if(kernel[i][0].toLowerCase().charAt(point)
                    == kernel[j][0].toLowerCase().charAt(point)) {
                if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                    if(i< j) {
                        return true;
                    }
                    return false;
                }
                return false;
            }
            return false;
        }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
            if(i< j) {
                if(!(i== rp || j== rp)) {
                    return true;
                }
```

```
                return false;
            }
            return false;
        }
        return false;
    }

    private boolean findSmallWithTwoChar(String x1, String x2, int scale, int point) {
        if(x1.length()<= point|| x2.length()<= point) {
            if(x1.length()< x2.length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(x1.length()> p&& x2.length()> p) {
                        if(x1.charAt(p)!= x2.charAt(p)) {
                            find= false;
                        }
                    }
                }
                if(find) {
                    return true;
                }
                return false;
            }
            return false;
        }else if(pinyin.containsKey(""+ x1.charAt(point))
                && pinyin.containsKey(""+ x2.charAt(point))){
            String[] js= new String[2];
            js[0]= this.pinyin.get(""+ x1.charAt(point));
            js[1]= this.pinyin.get(""+x2.charAt(point));
            boolean change= processSortPinYin(js, 3);
            if(change) {
                return true;
            }
            return false;
        }else if(!pinyin.containsKey(""+ x1.charAt(point))
                && !pinyin.containsKey(""+ x2.charAt(point))){
            if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                return true;
            }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                if(x1.charAt(point)> x2.charAt(point)) {
                    return true;
                }
                return false;
            }
```

```
            return false;
        }else if(!pinyin.containsKey(""+ x1.charAt(point))
                && pinyin.containsKey(""+ x2.charAt(point))){
                return true;
        }
        return false;
}

private int partition(String[][] a, int lp, int rp, int scale, int point) {
        String[] x= !findSmall(a, scale, point, lp, rp, rp)? a[lp]: a[rp];
        int lp1= lp;
        int count= 0;
        int lastCount= 0;
        while(lp1< rp) {
            while(!(findSmallWithTwoChar(a[lp1][0], x[0], scale, point)|| lp1>= rp)) {
                lp1++;
                count++;
            }
            while(findSmallWithTwoChar(a[rp][0], x[0], scale, point)){
                rp--;
                count++;
            }
            if(lp1< rp){
                String[] temp= a[rp].clone();
                a[rp]= a[lp1].clone();
                a[lp1]= temp;
            }
            if(count!= lastCount) {
                if(lp1< rp){
                    String[] temp= a[rp].clone();
                    a[rp]= a[lp1].clone();
                    a[lp1]= temp;
                }
                lastCount= count;
            }else {
                rp--;
            }
        }
        a[lp]=a[rp].clone();a[rp]=x;
        return rp;
}
private boolean processSortPinYin(String[] kernel, int scale) {
        for(int k= 0; k< scale; k++) {
            if(kernel[0].length()<= k|| kernel[1].length()<= k) {
```

```
                    if(kernel[0].length()< kernel[1].length()) {
                        return true;
                    }
                    return false;
                }
                if(kernel[0].toLowerCase().charAt(k)> kernel[1].toLowerCase().charAt(k)) {
                    return true;
                }else if(kernel[0].toLowerCase().charAt(k)< kernel[1].toLowerCase().charAt(k)) {
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {
                return true;
            }
            return false;
        }
    }
```

LYG4DWithChineseMixStringSort5D

算法思想：罗瑶光混合象契字符串小高峰过滤排序缺陷峰值理论优化版

研发实现：罗瑶光

```
==========================================================================
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort5D{
    Map<String, Boolean> find= new HashMap<>();
    Map<String, String> pinyin;
    public void quick4DChineseStringArray(String[] a, int lp, int rp, int scale, Map<String, String>
map) {
        this.pinyin= map;
        String[][] kernel= new String[a.length][3];
        for(int i= 0; i< a.length; i++) {
            kernel[i][0]= a[i].toString();
        }
        processKernel(kernel, lp, rp, scale, 0);
        for(int i= 0; i< a.length; i++) {
            a[i]= kernel[i][0].toString();
        }
    }
    private void processKernel(String[][] kernel, int lp, int rp, int scale, int point) {
        int rp1= rp;
        if(point> scale) {
            return;
        }
        processQS4DLYG4D(kernel, lp, rp, scale, point);
        int i;
        for(i= lp; i<= rp; i++) {
            if(!(kernel[i][0].length()<= point|| kernel[lp][0].length()<= point)) {
                if(kernel[i][0].charAt(point)!= kernel[lp][0].charAt(point)){
                    rp1= i-1;
                    processKernel(kernel, lp, rp1, scale, point+1);
                    lp= i;
                }
            }
        }
        if(lp!=rp) {
            processKernel(kernel, lp, i-1, scale, point+1);
        }
    }
    private void processSort(String[][] kernel, int lp, int rp, int scale, int point) {
        if(point> scale) {
```

```
        return;
    }
    for(int i= lp; i<= rp; i++) {
        Here:
        for(int j= i; j<= rp; j++) {
            if(i==j) {
                continue Here;
            }
            if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
                if(kernel[i][0].length()< kernel[j][0].length()) {
                    boolean find= true;
                    for(int p= 0; p< scale; p++) {
                        if(!(kernel[i][0].length()<= p|| kernel[j][0].length()<= p)) {
                            if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                                find= false;
                            }
                        }
                    }
                    if(find) {
                        String[] temp= kernel[i].clone();
                        kernel[i]= kernel[j].clone();
                        kernel[j]= temp;
                    }
                }
            }else if(!(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
                    || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
                String[] js= new String[2];
                js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
                js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
                boolean change= processSortPinYin(js, 3);
                if(!(!change|| i>= j)) {
                    String[] temp= kernel[i].clone();
                    kernel[i]= kernel[j].clone();
                    kernel[j]= temp;
                }
            }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                    || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
                if(i< j) {
                    if(!(i== rp+1 || j== rp+1)) {
                        String[] temp= kernel[i].clone();
                        kernel[i]= kernel[j].clone();
                        kernel[j]= temp;
                    }
                }
```

```
                }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
                        || pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
                    if(kernel[i][0].toLowerCase().charAt(point)
                            > kernel[j][0].toLowerCase().charAt(point)) {
                        if(i< j) {
                            String[] temp= kernel[i].clone();
                            kernel[i]= kernel[j].clone();
                            kernel[j]= temp;
                        }
                    }else if(kernel[i][0].toLowerCase().charAt(point)
                            == kernel[j][0].toLowerCase().charAt(point)) {
                        if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                            if(i< j) {
                                String[] temp= kernel[i].clone();
                                kernel[i]= kernel[j].clone();
                                kernel[j]= temp;
                            }
                        }
                    }
                }
            }
        }
    }
    private void processQS4DLYG4D(String[][] kernel, int lp, int rp, int scale, int point) {
        if(lp< rp){
            int c= rp- lp;
            if(c< 3){
                processSort(kernel, lp, rp, scale, point);
                return;
            }
            int pos= partition(kernel, lp, rp, scale, point);
            processQS4DLYG4D(kernel, lp, pos- 1, scale, point);
            processQS4DLYG4D(kernel, pos+ 1, rp, scale, point);
        }
    }
    private boolean findSmall(String[][] kernel, int scale, int point, int i, int j, int rp) {
        if(kernel[i][0].length()<= point|| kernel[j][0].length()<= point) {
            if(kernel[i][0].length()< kernel[j][0].length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(!(kernel[i][0].length()<= p|| kernel[j][0].length()<= p)) {
                        if(kernel[i][0].charAt(p)!= kernel[j][0].charAt(p)) {
                            find= false;
                        }
```

```
                }
            }
            if(find) {
                return true;
            }
            return false;
        }
        return false;
    }else if(!(!pinyin.containsKey(""+ kernel[i][0].charAt(point))
            || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
        String[] js= new String[2];
        js[0]= this.pinyin.get(""+ kernel[i][0].charAt(point));
        js[1]= this.pinyin.get(""+ kernel[j][0].charAt(point));
        boolean change= processSortPinYin(js, 3);
        if(!(!change|| i>= j)) {
            return true;
        }
        return false;
    }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
            || pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
        if(kernel[i][0].toLowerCase().charAt(point)
                > kernel[j][0].toLowerCase().charAt(point)) {
            if(i< j) {
                return true;
            }
            return false;
        }else if(kernel[i][0].toLowerCase().charAt(point)
                == kernel[j][0].toLowerCase().charAt(point)) {
            if(kernel[i][0].charAt(point)> kernel[j][0].charAt(point)) {
                if(i< j) {
                    return true;
                }
                return false;
            }
            return false;
        }
        return false;
    }else if(!(pinyin.containsKey(""+ kernel[i][0].charAt(point))
            || !pinyin.containsKey(""+ kernel[j][0].charAt(point)))){
        if(i< j) {
            if(!(i== rp || j== rp)) {
                return true;
            }
            return false;
```

```
            }
            return false;
        }
        return false;
    }
    private boolean findSmallWithTwoChar(String x1, String x2, int scale, int point) {
        if(x1.length()<= point|| x2.length()<= point) {
            if(x1.length()< x2.length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(!(x1.length()<= p|| x2.length()<= p)) {
                        if(x1.charAt(p)!= x2.charAt(p)) {
                            find= false;
                        }
                    }
                }
                if(find) {
                    return true;
                }
                return false;
            }
            return false;
        }else if(!(!pinyin.containsKey(""+ x1.charAt(point))
                || !pinyin.containsKey(""+ x2.charAt(point)))){
            String[] js= new String[2];
            js[0]= this.pinyin.get(""+ x1.charAt(point));
            js[1]= this.pinyin.get(""+x2.charAt(point));
            boolean change= processSortPinYin(js, 3);
            if(change) {
                return true;
            }
            return false;
        }else if(!(pinyin.containsKey(""+ x1.charAt(point))
                || pinyin.containsKey(""+ x2.charAt(point)))){
            if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                return true;
            }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                if(x1.charAt(point)> x2.charAt(point)) {
                    return true;
                }
                return false;
            }
            return false;
        }else if(!(pinyin.containsKey(""+ x1.charAt(point))
```

```
                || !pinyin.containsKey(""+ x2.charAt(point)))){
                return true;
        }
        return false;
}
private int partition(String[][] a, int lp, int rp, int scale, int point) {
        String[] x= findSmall(a, scale, point, lp, rp, rp)? a[rp]: a[lp];
        int lp1= lp;
        int count= 0;
        int lastCount= 0;
        while(lp1< rp) {
                while(!(findSmallWithTwoChar(a[lp1][0], x[0], scale, point)|| lp1>= rp)) {
                        lp1++;
                        count++;
                }
                while(findSmallWithTwoChar(a[rp][0], x[0], scale, point)){
                        rp--;
                        count++;
                }
                if(lp1< rp){
                        String[] temp= a[rp].clone();
                        a[rp]= a[lp1].clone();
                        a[lp1]= temp;
                }
                if(count!= lastCount) {
                        if(lp1< rp){
                                String[] temp= a[rp].clone();
                                a[rp]= a[lp1].clone();
                                a[lp1]= temp;
                        }
                        lastCount= count;
                }else {
                        rp--;
                }
        }
        a[lp]=a[rp].clone();a[rp]=x;
        return rp;
}
private boolean processSortPinYin(String[] kernel, int scale) {
        for(int k= 0; k< scale; k++) {
                if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                        if(kernel[0].length()< kernel[1].length()) {
                                return true;
                        }
```

```
                    return false;
                }
                if(kernel[0].toLowerCase().charAt(k)> kernel[1].toLowerCase().charAt(k)) {
                    return true;
                }else if(kernel[0].toLowerCase().charAt(k)< kernel[1].toLowerCase().charAt(k)) {
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {
                return true;
            }
            return false;
        }
}
```

LYG4DWithChineseMixStringSort6D

算法思想：罗瑶光混合象契字符串小高峰过滤排序缺陷峰值理论优化版

研发实现：罗瑶光

==========

```
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort6D{
	Map<String, Boolean> find= new HashMap<>();
	Map<String, String> pinyin;
	public void quick4DChineseStringArray(String[] strings, int leftPosition
			, int rightPosition, int scale, Map<String, String> map) {
		this.pinyin= map;
		processKernel(strings, leftPosition, rightPosition, scale, 0);
	}

	private void processKernel(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		int rightPositionReflection= rightPosition;
		if(point> scale) {
			return;
		}
		processQS4DLYG4D(kernel, leftPosition, rightPosition, scale, point);
		int i;
		for(i= leftPosition; i<= rightPosition; i++) {
			if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {
				if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){
					rightPositionReflection= i- 1;
					processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);
					leftPosition= i;
				}
			}
		}
		if(leftPosition!= rightPosition) {
			processKernel(kernel, leftPosition, i- 1, scale, point+ 1);
		}
	}
	private void processSort(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		if(point> scale) {
			return;
		}
		for(int i= leftPosition; i<= rightPosition; i++) {
			Here:
```

```
for(int j= i; j<= rightPosition; j++) {
        if(i== j) {
                continue Here;
        }
        if(kernel[i].length()<= point|| kernel[j].length()<= point) {
                if(kernel[i].length()< kernel[j].length()) {
                        boolean find= true;
                        for(int p= 0; p< scale; p++) {
                                if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
                                        if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
                                                find= false;
                                                break;
                                        }
                                }
                        }
                        if(find) {
                                String temp= kernel[i].toString();;
                                kernel[i]= kernel[j].toString();;
                                kernel[j]= temp;
                        }
                }
                continue Here;
        }else {
                boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
                boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
                if(!(!hasXi|| !hasXj)){
                        String[] js= new String[2];
                        js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
                        js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
                        boolean change= processSortPinYin(js, 3);
                        if(!(!change|| i>= j)) {
                                String temp= kernel[i].toString();;
                                kernel[i]= kernel[j].toString();;
                                kernel[j]= temp;
                        }
                        continue Here;
                }else if(!(hasXi|| !hasXj)){
                        if(i< j) {
                                if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                        String temp= kernel[i].toString();
                                        kernel[i]= kernel[j].toString();
                                        kernel[j]= temp;
                                }
                        }
```

```
                                continue Here;
                            }else if(!(hasXi|| hasXj)){
                                if(kernel[i].toLowerCase().charAt(point)
                                        > kernel[j].toLowerCase().charAt(point)) {
                                    if(i< j) {
                                        String temp= kernel[i].toString();
                                        kernel[i]= kernel[j].toString();
                                        kernel[j]= temp;
                                    }
                                    continue Here;
                                }
                                if(kernel[i].toLowerCase().charAt(point)
                                        == kernel[j].toLowerCase().charAt(point)) {
                                    if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                                        if(i< j) {
                                            String temp= kernel[i].toString();
                                            kernel[i]= kernel[j].toString();
                                            kernel[j]= temp;
                                        }
                                    }
                                }
                            }
                        }
                    }
        }
    }
    private void processQS4DLYG4D(String[] kernel, int leftPosition
            , int rightPosition, int scale, int point) {
        if(leftPosition< rightPosition){
            int c= rightPosition- leftPosition;
            if(c< 7){
                processSort(kernel, leftPosition, rightPosition, scale, point);
                return;
            }
            int pos= partition(kernel, leftPosition, rightPosition, scale, point);
            processQS4DLYG4D(kernel, leftPosition, pos- 1, scale, point);
            processQS4DLYG4D(kernel, pos+ 1, rightPosition, scale, point);
        }
    }

    private boolean findSmall(String[] kernel, int scale, int point
            , int i, int j, int rightPosition) {
        if(kernel[i].length()<= point|| kernel[j].length()<= point) {
            if(kernel[i].length()< kernel[j].length()) {
```

```
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
                        if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
                            find= false;
                            break;
                        }
                    }
                }
                if(find) {
                    return true;
                }
                return false;
            }
            return false;
        }else {
            boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
            boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
            if(!(!hasXi|| !hasXj)){
                String[] js= new String[2];
                js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
                js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
                boolean change= processSortPinYin(js, 3);
                if(!(!change|| i>= j)) {
                    return true;
                }
                return false;
            }else if(!(hasXi|| hasXj)){
                if(kernel[i].toLowerCase().charAt(point)
                        > kernel[j].toLowerCase().charAt(point)) {
                    if(i< j) {
                        return true;
                    }
                    return false;
                }else if(kernel[i].toLowerCase().charAt(point)
                        == kernel[j].toLowerCase().charAt(point)) {
                    if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                        if(i< j) {
                            return true;
                        }
                        return false;
                    }
                    return false;
                }
```

```
                return false;
            }else if(!(hasXi|| !hasXj)){
                if(i< j) {
                    if(!(i== rightPosition || j== rightPosition)) {
                        return true;
                    }
                    return false;
                }
                return false;
            }
        }
        return false;
    }

    private boolean findSmallWithTwoChar(String x1, String x2
            , int scale, int point) {
        if(x1.length()<= point|| x2.length()<= point) {
            if(x1.length()< x2.length()) {
                boolean find= true;
                for(int p= 0; p< scale; p++) {
                    if(!(x1.length()<= p|| x2.length()<= p)) {
                        if(x1.charAt(p)!= x2.charAt(p)) {
                            find= false;
                            break;
                        }
                    }
                }
                if(find) {
                    return true;
                }
                return false;
            }
            return false;
        }else {
            boolean hasX1= pinyin.containsKey(""+ x1.charAt(point));
            boolean hasX2= pinyin.containsKey(""+ x2.charAt(point));
            if(!(!hasX1|| !hasX2)){
                String[] js= new String[2];
                js[0]= this.pinyin.get(""+ x1.charAt(point));
                js[1]= this.pinyin.get(""+x2.charAt(point));
                boolean change= processSortPinYin(js, 3);
                if(change) {
                    return true;
                }
```

```
                return false;
            }else if(!(hasX1|| hasX2)){
                if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                    return true;
                }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                    if(x1.charAt(point)> x2.charAt(point)) {
                        return true;
                    }
                    return false;
                }
                return false;
            }else if(!(hasX1|| !hasX2)){
                return true;
            }
        }
        return false;
}

private int partition(String[] array, int leftPosition, int rightPosition
            , int scale, int point) {
        String x= findSmall(array, scale, point, leftPosition, rightPosition
                , rightPosition)? array[rightPosition]: array[leftPosition];
                int leftPositionReflection= leftPosition;
                int count= 0;
                int lastCount= 0;
                while(leftPositionReflection< rightPosition) {
                    while(!(findSmallWithTwoChar(array[leftPositionReflection]
                            , x, scale, point)|| leftPositionReflection>= rightPosition)) {
                        leftPositionReflection++;
                        count++;
                    }
                    while(findSmallWithTwoChar(array[rightPosition], x, scale, point)){
                        rightPosition--;
                        count++;
                    }
                    if(leftPositionReflection< rightPosition){
                        String temp= array[rightPosition].toString();;
                        array[rightPosition]= array[leftPositionReflection].toString();;
                        array[leftPositionReflection]= temp;
                    }
                    if(count!= lastCount) {
                        lastCount= count;
                    }else {
                        rightPosition--;
```

```
                    }
                }
                array[leftPosition]= array[rightPosition].toString();array[rightPosition]=x;
                return rightPosition;
        }
        private boolean processSortPinYin(String[] kernel, int scale) {
            for(int k= 0; k< scale; k++) {
                if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                    if(kernel[0].length()< kernel[1].length()) {
                        return true;
                    }
                    return false;
                }
                if(kernel[0].toLowerCase().charAt(k)
                        > kernel[1].toLowerCase().charAt(k)) {
                    return true;
                }
                if(kernel[0].toLowerCase().charAt(k)
                        < kernel[1].toLowerCase().charAt(k)) {
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {
                return true;
            }
            return false;
        }
}
```

LYG4DWithChineseMixStringSort7D

算法思想：罗瑶光混合象契字符串小高峰过滤排序缺陷峰值理论当前稳定版

研发实现：罗瑶光

==========

```
package org.deta.tinos.string;
import java.util.HashMap;
import java.util.Map;
public class LYG4DWithChineseMixStringSort7D{
	Map<String, Boolean> find= new HashMap<>();
	Map<String, String> pinyin;
	int range;
	public void quick4DChineseStringArray(String[] strings, int leftPosition
			, int rightPosition, int scale, Map<String, String> map, int range) {
		this.pinyin= map;
		this.range= range;
		processKernel(strings, leftPosition, rightPosition, scale, 0);
	}

	private void processKernel(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		int rightPositionReflection= rightPosition;
		if(point> scale) {
			return;
		}
		processQS4DLYG4D(kernel, leftPosition, rightPosition, scale, point);
		int i;
		for(i= leftPosition; i<= rightPosition; i++) {
			if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {
				if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){
					rightPositionReflection= i- 1;
					processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);
					leftPosition= i;
				}
			}
		}
		if(leftPosition!= rightPosition) {
			processKernel(kernel, leftPosition, i- 1, scale, point+ 1);
		}
	}
	private void processSort(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		if(point> scale) {
			return;
		}
```

```
for(int i= leftPosition; i<= rightPosition; i++) {
        Here:
                for(int j= i; j<= rightPosition; j++) {
                        if(i== j) {
                                continue Here;
                        }
                        if(kernel[i].length()<= point|| kernel[j].length()<= point) {
                                if(kernel[i].length()< kernel[j].length()) {
                                        for(int p= 0; p< scale; p++) {
                                                if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
                                                        if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
                                                                continue Here;
                                                        }
                                                }
                                        }
                                        String temp= kernel[i].toString();;
                                        kernel[i]= kernel[j].toString();;
                                        kernel[j]= temp;
                                }
                                continue Here;
                        }else {
                                boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
                                boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
                                if(!(!hasXi|| !hasXj)){
                                        String[] js= new String[2];
                                        js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
                                        js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
                                        boolean change= processSortPinYin(js, 3);
                                        if(!(!change|| i>= j)) {
                                                String temp= kernel[i].toString();;
                                                kernel[i]= kernel[j].toString();;
                                                kernel[j]= temp;
                                        }
                                        continue Here;
                                }else if(!(hasXi|| !hasXj)){
                                        if(i< j) {
                                                if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                        String temp= kernel[i].toString();
                                                        kernel[i]= kernel[j].toString();
                                                        kernel[j]= temp;
                                                }
                                        }
                                        continue Here;
                                }else if(!(hasXi|| hasXj)){
```

```
                                    if(kernel[i].toLowerCase().charAt(point)
                                            > kernel[j].toLowerCase().charAt(point)) {
                                        if(i< j) {
                                            String temp= kernel[i].toString();
                                            kernel[i]= kernel[j].toString();
                                            kernel[j]= temp;
                                        }
                                        continue Here;
                                    }
                                    if(kernel[i].toLowerCase().charAt(point)
                                            == kernel[j].toLowerCase().charAt(point)) {
                                        if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                                            if(i< j) {
                                                String temp= kernel[i].toString();
                                                kernel[i]= kernel[j].toString();
                                                kernel[j]= temp;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                private void processQS4DLYG4D(String[] kernel, int leftPosition
                        , int rightPosition, int scale, int point) {
                    if(leftPosition< rightPosition){
                        int c= rightPosition- leftPosition;
                        if(c< this.range){
                            processSort(kernel, leftPosition, rightPosition, scale, point);
                            return;
                        }
                        int pos= partition(kernel, leftPosition, rightPosition, scale, point);
                        processQS4DLYG4D(kernel, leftPosition, pos- 1, scale, point);
                        processQS4DLYG4D(kernel, pos+ 1, rightPosition, scale, point);
                    }
                }
                private boolean findSmall(String[] kernel, int scale, int point
                        , int i, int j, int rightPosition) {
                    if(kernel[i].length()<= point|| kernel[j].length()<= point) {
                        if(kernel[i].length()< kernel[j].length()) {
                            for(int p= 0; p< scale; p++) {
                                if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
                                    if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
```

```
                        return false;
                    }
                }
            }
            return true;
        }
        return false;
    }else {
        boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
        boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
        if(!(!hasXi|| !hasXj)){
            String[] js= new String[2];
            js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
            js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
            boolean change= processSortPinYin(js, 3);
            if(!(!change|| i>= j)) {
                return true;
            }
            return false;
        }else if(!(hasXi|| hasXj)){
            if(kernel[i].toLowerCase().charAt(point)
                    > kernel[j].toLowerCase().charAt(point)) {
                if(i< j) {
                    return true;
                }
                return false;
            }else if(kernel[i].toLowerCase().charAt(point)
                    == kernel[j].toLowerCase().charAt(point)) {
                if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                    if(i< j) {
                        return true;
                    }
                    return false;
                }
                return false;
            }
            return false;
        }else if(!(hasXi|| !hasXj)){
            if(i< j) {
                if(!(i== rightPosition || j== rightPosition)) {
                    return true;
                }
                return false;
            }
```

```
                return false;
            }
        }
        return false;
    }
    private boolean findSmallWithTwoChar(String x1, String x2
            , int scale, int point) {
        if(x1.length()<= point|| x2.length()<= point) {
            if(x1.length()< x2.length()) {
                for(int p= 0; p< scale; p++) {
                    if(!(x1.length()<= p|| x2.length()<= p)) {
                        if(x1.charAt(p)!= x2.charAt(p)) {
                            return false;
                        }
                    }
                }
                return true;
            }
            return false;
        }else {
            boolean hasX1= pinyin.containsKey(""+ x1.charAt(point));
            boolean hasX2= pinyin.containsKey(""+ x2.charAt(point));
            if(!(!hasX1|| !hasX2)){
                String[] js= new String[2];
                js[0]= this.pinyin.get(""+ x1.charAt(point));
                js[1]= this.pinyin.get(""+x2.charAt(point));
                boolean change= processSortPinYin(js, 3);
                if(change) {
                    return true;
                }
                return false;
            }else if(!(hasX1|| hasX2)){
                if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                    return true;
                }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                    if(x1.charAt(point)> x2.charAt(point)) {
                        return true;
                    }
                    return false;
                }
                return false;
            }else if(!(hasX1|| !hasX2)){
                return true;
            }
```

```
        }
        return false;
    }
    private int partition(String[] array, int leftPosition, int rightPosition, int scale, int point) {
        String x= findSmall(array, scale, point, leftPosition, rightPosition, rightPosition)
                    ? array[rightPosition]: array[leftPosition];
        int leftPositionReflection= leftPosition;
        int count= 0;
        int lastCount= 0;
        while(leftPositionReflection< rightPosition) {
            while(!(findSmallWithTwoChar(array[leftPositionReflection]
                        , x, scale, point)|| leftPositionReflection>= rightPosition)) {
                leftPositionReflection++;
                count++;
            }
            while(findSmallWithTwoChar(array[rightPosition], x, scale, point)){
                rightPosition--;
                count++;
            }
            if(leftPositionReflection< rightPosition){
                String temp= array[rightPosition].toString();;
                array[rightPosition]= array[leftPositionReflection].toString();;
                array[leftPositionReflection]= temp;
            }
            if(count!= lastCount) {
                lastCount= count;
            }else {
                rightPosition--;
            }
        }
        array[leftPosition]= array[rightPosition].toString();
        array[rightPosition]=x;
        return rightPosition;
    }
    private boolean processSortPinYin(String[] kernel, int scale) {
        for(int k= 0; k< scale; k++) {
            if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                if(kernel[0].length()< kernel[1].length()) {
                    return true;
                }
                return false;
            }
            if(kernel[0].toLowerCase().charAt(k)
                        > kernel[1].toLowerCase().charAt(k)) {
```

```
                    return true;
                }
                if(kernel[0].toLowerCase().charAt(k)
                        < kernel[1].toLowerCase().charAt(k)) {
                    return false;
                }
            }
            if(kernel[0].length()< kernel[1].length()) {
                return true;
            }
            return false;
        }
}
```
==========

Refer 地址

Linkedin

https://www.linkedin.com/in/yaoguangluo/

Doc: 极速象契字符串数列混排思想 5.0 PPT&PDF(Quick Mixed Array String Sort 5.0 PPT&PDF)

出版日期 2019 年 10 月 30 日 作品说明 Gitee, Github, DetaOSS

作品说明 Github：

https://github.com/yaoguangluo/Deta_Resource/blob/master/%E8%B1%A1%E5%A5%91%E6%B7%B7%E5%88%86%E6%80%9D%E6%83%B35.0.pdf

Gitee：

https://gitee.com/DetaChina/Deta_Resource/blob/master/%E8%B1%A1%E5%A5%91%E6%B7%B7%E5%88%86%E6%80%9D%E6%83%B35.0.pdf

Source：常用函数逻辑化简手稿 20190923

出版日期 2019 年 9 月 23 日 作品说明 Gitee, Github, DetaOSS

https://github.com/yaoguangluo/Deta_Resource/blob/master/%E5%B8%B8%E7%94%A8%E5%87%BD%E6%95%B0%E9%80%BB%E8%BE%91%E5%8C%96%E7%AE%80%E6%89%8B%E7%A8%BF.jpg

Paper: Theory on Yaoguang's Split Peak Defect 1.020190908 FIX

出版日期 2019 年 9 月 8 日 作品说明 Gitee, Github, DetaOSS

For Gitee:

https://gitee.com/DetaChina/Deta_Resource/blob/master/Theory%20on%20Yaoguang's%20Split%20Peak%20Defect%201.020190901.pdf

Github

https://github.com/yaoguangluo/Deta_Resource/blob/master/Theory%20on%20Yaoguang's%20Split%20Peak%20Defect%201.020190908%20FIX.pdf

Source: 两种比较领先的 Quick Sort Kernel 思维对比

出版日期 2019 年 9 月 7 日 作品说明 Gitee, Github, DetaOSS

作品说明 https://github.com/yaoguangluo/Deta_Resource/blob/master/两种比较领先的排序思维对比.pdf

https://github.com/yaoguangluo/Deta_Resource/blob/master/%E4%B8%A4%E7%A7%8D%E6%AF%94%E8%BE%83%E9%A2%86%E5%85%88%E7%9A%84%E6%8E%92%E5%BA%8F%E6%80%9D%E7%BB%B4%E5%AF%B9%E6%AF%94.docx

Gitee:

https://gitee.com/DetaChina/Deta_Resource/blob/master/%E4%B8%A4%E7%A7%8D%E6%AF%94%E8%BE%83%E9%A2%86%E5%85%88%E7%9A%84%E6%8E%92%E5%BA%8F%E6%80%9D%E7%BB%B4%E5%AF%B9%E6%AF%94.pdf

https://gitee.com/DetaChina/Deta_Resource/blob/master/%E4%B8%A4%E7%A7%8D%E6%AF%94%E8%BE%83%E9%A2%86%E5%85%88%E7%9A%84%E6%8E%92%E5%BA%8F%E6%80%9D%E7%BB%B4%E5%AF%B9%E6%AF%94.docx