

Java Standard Code Library

for ACM-ICPC

SiriYang

2019.5.5

前言

本项目为 Java 语言版本的标准算法代码库，记录了较为常用的一些算法模板，代码全部为平时编程学习所积累，免费开源定期维护，主要为 ACM-ICPC 比赛而准备也可用于平时的项目开发需要。代码虽都经过作者亲自编译调试，但也不保证存在错误。

为代码阅读的美观，设定了一些字体高亮等格式，导致从文档中复制的代码无法直接使用，如需要源码可从下方 GitHub 仓库下载使用。

GitHub 项目地址：<https://github.com/SiriYXR/JSCL>

参考资料：

挑战程序设计竞赛（第二版）[秋叶拓哉 岩田阳一 北川宜稔 人民邮电出版社]

ACM-IPCP 基本算法 [滕国文 李昊 清华大学出版社]

ACM 国际大学生程序设计竞赛算法与实现 [余勇 清华大学出版社]

目录

1 工具	3
1.1 对数器	3
2 数论	5
2.1 阶乘	5
2.2 判断素数	5
2.3 辗转相除法	6
2.4 快速幂	6
2.5 矩阵快速幂	6
3 线性表&矩阵	7
3.1 全排列	7
3.2 快速排序	8
3.3 二分查找	9
3.4 荷兰国旗问题	9
4 树	10
4.1 前缀树	10
5 图	12
5.1 Dijkstra	12
5.2 Bellman-Ford	14
5.3 Floyd_Warshall	15
6 其他	16
6.1 并查集	16

1 工具

1.1 对数器

```
1. import java.util.Arrays;
2.
3. public class InspectionMachine {
4.
5.     public static void main(String[] args) {
6.
7.         int testTime = 500000;
8.         int size = 10;
9.         int value = 100;
10.        boolean succeed = true;
11.
12.        for(int i=0;i<testTime;i++){
13.            int[] arr1=generateRandomArray(size,value);
14.            int[] arr2=copyArray(arr1);
15.            int[] arr3=copyArray(arr1);
16.
17.            Arrays.sort(arr2);
18.            rightMathod(arr3);
19.            if (!isEqual(arr2,arr3)){
20.                succeed=false;
21.                printArray(arr1);
22.                break;
23.            }
24.        }
25.        System.out.println(succeed ? "Nice!":"Fucking fucked!");
26.    }
27.
28.    //绝对正确的方法
29.    public static void rightMathod(int[] arr) {
30.        Arrays.sort(arr);
31.    }
32.
33.    //随机数组生成器，用于生成数据
34.    public static int[] generateRandomArray(int size, int value) {
35.        //Math.random() -> double [0,1)
36.        //(int)((size+1)*Math.random()) -> [0,size] 整数
37.        //size = 6, size + 1 = 7;
38.        //Math.random() -> [0,1) * 7 -> [0,7) double
```

```

39.         //double -> int [0,6] -> int
40.
41.         //生成长度随机的数组
42.         int[] arr = new int[(int) ((size + 1) * Math.random())];
43.         for (int i = 0; i < arr.length; i++) {
44.             arr[i] = (int) ((value + 1) * Math.random()) - (int) (value * Ma
th.random());
45.         }
46.         return arr;
47.     }
48.
49.     //拷贝数组
50.     public static int[] copyArray(int[] arr) {
51.         if (arr == null) {
52.             return null;
53.         }
54.         int[] res = new int[arr.length];
55.         for (int i = 0; i < arr.length; i++) {
56.             res[i] = arr[i];
57.         }
58.         return res;
59.     }
60.
61.     //判断数组是否相等
62.     public static boolean isEqual(int[] arr1, int[] arr2) {
63.         if ((arr1 == null && arr2 != null) || (arr1 != null && arr2 == null)
)
64.             return false;
65.         if (arr1 == null && arr2 == null)
66.             return true;
67.         if (arr1.length != arr2.length)
68.             return false;
69.         for (int i = 0; i < arr1.length; i++) {
70.             if (arr1[i] != arr2[i]) {
71.                 return false;
72.             }
73.         }
74.         return true;
75.     }
76.
77.     //打印数组
78.     public static void printArray(int[] arr) {
79.         if (arr == null)
80.             return;

```

```

81.         for(int i:arr){
82.             System.out.print(i+" ");
83.         }
84.         System.out.println();
85.     }
86. }
87.

```

2 数论

2.1 阶乘

```

1. public class Factorial {
2.
3.     public static long factorial(long n){
4.         if(n==0)
5.             return 1; //0 的阶乘为 1
6.         else
7.             return factorial(n-1)*n;
8.     }
9. }

```

2.2 判断素数

```

1. public class Prime {
2.
3.     public static boolean isPrime(int n) {
4.         if (n < 2)
5.             return false;
6.         if (n == 2)
7.             return true;
8.         if (n % 2 == 0)
9.             return false;
10.        for (int i = 3; i * i <= n; i += 2)
11.            if (n % i == 0)
12.                return false;
13.        return true;
14.    }
15. }

```

2.3 辗转相除法

```
1. public class GCD {
2.
3.     public static int gcd(int a, int b) {
4.         if (b == 0)
5.             return a;
6.         return gcd(b, a % b);
7.     }
8. }
```

2.4 快速幂

```
1. public class Main {
2.
3.     public static long mod_pow(long x, long n, long mod) {
4.         if(n==0)
5.             return 1;
6.         long res=mod_pow(x*x,n/2,mod);
7.         if((n&1)==1)
8.             res=res*x%mod;
9.         return res;
10.    }
11. }
```

2.5 矩阵快速幂

```
1. public class Main {
2.
3.     public static long[][] mut(int k, int n, long[][] A){
4.         long [][] res = new long[n][n];
5.         for(int i = 0 ; i < res.length ; i++){
6.             for(int j = 0 ; j < res[i].length ; j++){
7.                 if(i==j){
8.                     res[i][j] = 1;
9.                 }else{
10.                     res[i][j] = 0;
11.                 }
12.             }
13.         }
```

```

14.     while(k!=0){
15.         if((k&1)==1) res = f(res,A);
16.         k>>=1;//k/=2;
17.         A = f(A,A);
18.     }
19.     return res;
20. }
21.
22. public static long[][] f(long[][] A,long[][] B){
23.     long res[][] = new long[A.length][B.length];
24.     for(int i = 0 ; i < res.length ;i++){
25.         for(int j = 0 ; j< res[i].length ;j++){
26.             for(int k = 0 ; k < A[0].length ;k++){
27.                 res[i][j] += A[i][k]*B[k][j];
28.             }
29.         }
30.     }
31.     return res;
32. }
33. }

```

3 线性表&矩阵

3.1 全排列

```

1. public class permutate {
2.     public static int total = 0;
3.     public static void swap(String[] str, int i, int j) {
4.         String temp = new String();
5.         temp = str[i];
6.         str[i] = str[j];
7.         str[j] = temp;
8.     }
9.
10.    public static void arrange (String[] str, int st, int len) {
11.        if (st == len - 1) {
12.            for (int i = 0; i < len; i++) {
13.                System.out.print(str[i]+ " ");
14.            }
15.            System.out.println();
16.            total++;
17.        }

```



```

18.         else {
19.             for (int i = st; i < len; i ++ ) {
20.                 swap(str, st, i);
21.                 arrange(str, st + 1, len);
22.                 swap(str, st, i);
23.             }
24.         }
25.     }
26.
27.     public static void main(String[] args) {
28.         String str[] = {"a","b","c"};
29.         arrange(str, 0, str.length);
30.         System.out.println(total);
31.     }
32. }

```

3.2 快速排序

```

1.  public class Main {
2.
3.         public static void quicksort(int[] num,int left,int right) {
4.             if(left<right) {
5.                 int l=left;
6.                 int r=right;
7.                 int temp=num[left];
8.                 while(l!=r) {
9.                     while(num[r]>=temp&&l<r)r--;
10.                    while(num[l]<=temp&&l<r)l++;
11.                    if(l<r) {
12.                        int t;
13.                        t=num[l];
14.                        num[l]=num[r];
15.                        num[r]=t;
16.                    }
17.                }
18.                num[left]=num[l];
19.                num[l]=temp;
20.                quicksort(num, left, l-1);
21.                quicksort(num,l+1,right);
22.            }
23.        }
24.    }

```

3.3 二分查找

```
1. public class Main {
2.
3.     //查找 v 出现的第一个位置
4.     public static int lowerBound(int[] nums, int l, int r, int v) {
5.         while (l < r) {
6.             int m = l + (r - l) / 2;
7.             if (nums[m] >= v)
8.                 r = m; // 因为是寻找下界，不考虑右边还有没有元素
9.             else if (nums[m] < v)
10.                 l = m + 1;
11.             if(l==r&&nums[m]!=v)//查找的数不存在，返回改数插入仍使数组有序的位置
12.                 return -(m+1);
13.         }
14.         return l;
15.     }
16.
17.     //查找 v 出现的最后一个位置
18.     public static int upperBound(int[] nums, int l, int r, int v) {
19.         while (l < r) {
20.             int m = l + (r - l) / 2;
21.             if (nums[m] <= v)
22.                 l = m + 1;
23.             else if (nums[m] > v)
24.                 r = m;
25.             if(l==r&&nums[m]!=v)
26.                 return -(m+1);
27.         }
28.         return l;
29.     }
30. }
```

3.4 荷兰国旗问题

```
1. public class NetherlandsFlag {
2.
3.     public static int[] partition(int[] arr, int L, int R, int num) {
4.         int less = L - 1;
5.         int more = R + 1;
6.         while (L < more) {
7.             if (arr[L] < num)
```

```

8.         swap(arr, ++less, L++);
9.         else if (arr[L] > num)
10.            swap(arr, --more, L);
11.         else
12.            L++;
13.     }
14.     return new int[]{less + 1, more - 1};
15. }
16.
17. public static void swap(int[] arr, int i, int j) {
18.     int tmp = arr[i];
19.     arr[i] = arr[j];
20.     arr[j] = tmp;
21. }
22. }

```

4 树

4.1 前缀树

```

1. import java.util.HashMap;
2.
3. public class TrieTree {
4.
5.     public static class TrieNode {
6.         public int pass;
7.         public int end;
8.
9.         public HashMap<Integer, TrieNode> nexts;
10.
11.         public TrieNode() {
12.             pass = 0;
13.             end = 0;
14.             nexts = new HashMap<Integer, TrieNode>();
15.         }
16.     }
17.
18.     public static class Trie {
19.         private TrieNode root;
20.

```

```

21.     public Trie() {
22.         root = new TrieNode();
23.     }
24.
25.     public void insert(String word) {
26.         if (word == null)
27.             return;
28.         char[] chs = word.toCharArray();
29.         TrieNode node = root;
30.         int index = 0;
31.         for (int i = 0; i < chs.length; i++) {
32.             index = chs[i] - 'a';
33.             if (!node.nexts.containsKey(index)) {
34.                 node.nexts.put(index, new TrieNode()); //添加结点
35.             }
36.             node = node.nexts.get(index);
37.             node.pass++;
38.         }
39.         node.end++;
40.     }
41.
42.     public void delete(String word) {
43.         if (search(word) != 0) {
44.             char[] chs = word.toCharArray();
45.             TrieNode node = root;
46.             int index = 0;
47.             for (int i = 0; i < chs.length; i++) {
48.                 index = chs[i] - 'a';
49.                 if (--node.nexts.get(index).pass == 0) {
50.                     node.nexts.remove(index); //删除结点
51.                     return;
52.                 }
53.                 node = node.nexts.get(index);
54.             }
55.             node.end--;
56.         }
57.     }
58.
59.     public int search(String word) {
60.         if (word == null)
61.             return 0;
62.         char[] chs = word.toCharArray();
63.         TrieNode node = root;
64.         int index = 0;

```

```

65.         for (int i = 0; i < chs.length; i++) {
66.             index = chs[i] - 'a';
67.             if (!node.nexts.containsKey(index)) {
68.                 return 0;
69.             }
70.             node = node.nexts.get(index);
71.         }
72.         return node.end;
73.     }
74.
75.     public int preixNumber(String pre){
76.         if (pre==null){
77.             return 0;
78.         }
79.         char[] chs = pre.toCharArray();
80.         TrieNode node = root;
81.         int index = 0;
82.         for (int i = 0; i < chs.length; i++) {
83.             index=chs[i]-'a';
84.             if(!node.nexts.containsKey(index)){
85.                 return 0;
86.             }
87.             node=node.nexts.get(index);
88.         }
89.         return node.pass;
90.     }
91. }
92. }

```

5 图

5.1 Dijkstra

```

1. import java.util.Comparator;
2. import java.util.PriorityQueue;
3. import java.util.Vector;
4.
5. class edge{
6.     int to;
7.     int cost;
8. }
9.

```

```

10. class pair{
11.     int first;
12.     int second;
13.     pair(int n1,int n2){
14.         first=n1;
15.         second=n2;
16.     }
17. }
18.
19. public class Main {
20.
21.     public static int MAX_V=1000,INF=99999;
22.     public static int V=1000;
23.     public static int[] d=new int[MAX_V];
24.     public static Vector<edge>[] G=new Vector[MAX_V];
25.
26.     public static void dijkstra(int s){
27.         for (int i = 0; i < V; i++) {
28.             d[i]=INF;
29.             G[i]=new Vector<edge>();
30.         }
31.         PriorityQueue<pair> que=new PriorityQueue<pair>(11,new Comparator<pair>(){
32.             public int compare(pair p1,pair p2){
33.                 return p1.first-p2.first;
34.             }
35.         });
36.         d[s]=0;
37.         que.offer(new pair(0,s));
38.
39.         while(!que.isEmpty()){
40.             pair p=que.poll();
41.             int v=p.second;
42.             if(d[v]<p.first)
43.                 continue;
44.             for(int i=0;i<G[v].size();i++){
45.                 edge e=G[v].get(i);
46.                 if(d[e.to]>d[v]+e.cost){
47.                     d[e.to]=d[v]+e.cost;
48.                     que.offer(new pair(d[e.to],e.to));
49.                 }
50.             }
51.         }
52.     }

```

53. }

5.2 Bellman-Ford

```
1. //从顶点 from 指向顶点 to 的权值为 cost 的边
2. class edge{
3.     int from,to,cost;
4. }
5.
6. public class Main {
7.
8.     public static int MAX_V=1000,MAX_E,INF=99999;
9.     public static int V=1000,E=10000;//V 顶点数,E 边数
10.    public static int[] d=new int[MAX_V];//最短距离
11.    public static edge[] es=new edge[MAX_E];//边
12.
13.    //求解从顶点 s 出发到所有点的最短距离
14.    public static void shortest_path(int s){
15.        for (int i = 0; i < V; i++) {
16.            d[i]=INF;
17.        }
18.        for (int i = 0; i < E; i++) {
19.            es[i]=new edge();
20.        }
21.        d[s]=0;
22.
23.        while(true){
24.            boolean update=false;
25.            for (int i = 0; i < E; i++) {
26.                edge e=es[i];
27.                if (d[e.from]!=INF&&d[e.to]>d[e.from]+e.cost) {
28.                    d[e.to]=d[e.from]+e.cost;
29.                    update=true;
30.                }
31.            }
32.            if (!update) {
33.                break;
34.            }
35.        }
36.    }
37.
38.    //如果返回 true 则存在负圈
39.    public static boolean find_negative_loop(){
```

```

40.     for (int i = 0; i < d.length; i++) {
41.         d[i]=0;
42.     }
43.
44.     for (int i = 0; i < V; i++) {
45.         for (int j = 0; j < E; j++) {
46.             edge e=es[j];
47.             if (d[e.to]>d[e.from]+e.cost) {
48.                 d[e.to]=d[e.from]+e.cost;
49.
50.                 //如果第 n 次仍然更新了, 则存在负圈
51.                 if(i==V-1)
52.                     return true;
53.             }
54.         }
55.     }
56.     return false;
57. }
58. }

```

5.3 Floyd_Warshall

```

1. public class Main {
2.
3.     public static int MAX_V=1000,INF=99999;
4.     public static int V=1000;//顶点数
5.     //d[u][v]表示边 e=(u,v)的权值(不存在时设为 INF,不过 d[i][i]=0)
6.     public static int[][] d=new int[MAX_V][MAX_V];
7.
8.     public static void warshall_floyd(){
9.         for (int k = 0; k < V; k++) {
10.            for (int i = 0; i < V; i++) {
11.                for (int j = 0; j < V; j++) {
12.                    d[i][j]=Math.min(d[i][j], d[i][k]+d[k][j]);
13.                }
14.            }
15.        }
16.    }
17. }

```


6 其他

6.1 并查集

```
1. import java.util.HashMap;
2. import java.util.LinkedList;
3. import java.util.List;
4.
5. public class UnionSet {
6.
7.     public static class Data {
8.
9.     }
10.
11.     public static class UnionFindSet {
12.         //(key,value)表示, key 的父节点, 是 value, (Data_A, Data_B) 代表,
           Data_A 的父节点是 Data_B
13.         public HashMap<Data, Data> fatherMap;
14.         public HashMap<Data, Integer> sizeMap;
15.
16.         public UnionFindSet(List<Data> nodes) {
17.             fatherMap = new HashMap<Data, Data>();
18.             sizeMap = new HashMap<Data, Integer>();
19.             makeSets(nodes);
20.         }
21.
22.         private void makeSets(List<Data> nodes) {
23.             fatherMap.clear();
24.             sizeMap.clear();
25.             for (Data node : nodes) {
26.                 fatherMap.put(node, node);
27.                 sizeMap.put(node, 1);
28.             }
29.         }
30.
31.         private Data findHead(Data node) {
32.             Data father = fatherMap.get(node);
33.             if (father != node)
34.                 father = findHead(father);
35.             fatherMap.put(node, father);
36.             return father;
37.         }
}
```

```

38.
39.     public boolean isSameSet(Data a, Data b) {
40.         return findHead(a) == findHead(b);
41.     }
42.
43.     public void union(Data a, Data b) {
44.         if (a == null || b == null)
45.             return;
46.         Data aHead = findHead(a);
47.         Data bHead = findHead(b);
48.         if (aHead != bHead) {
49.             int aSetSize = sizeMap.get(aHead);
50.             int bSetSize = sizeMap.get(bHead);
51.             if (aSetSize <= bSetSize) {
52.                 fatherMap.put(aHead, bHead);
53.                 sizeMap.put(bHead, aSetSize + bSetSize);
54.             } else {
55.                 fatherMap.put(bHead, aHead);
56.                 sizeMap.put(aHead, aSetSize + bSetSize);
57.             }
58.         }
59.     }
60. }
61. }

```