Java Standard Code Library

for ACM-ICPC

SiriYang

2019.5.6

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，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

1.1 对数器

```java
import java.util.Arrays;

public class InspectionMachine {

    public static void main(String[] args) {

        int testTime = 500000;
        int size = 10;
        int value = 100;
        boolean succeed = true;

        for(int i=0;i<testTime;i++){
            int[] arr1=generateRandomArray(size,value);
            int[] arr2=copyArray(arr1);
            int[] arr3=copyArray(arr1);

            Arrays.sort(arr2);
            rightMathod(arr3);
            if (!isEqual(arr2,arr3)){
                succeed=false;
                printArray(arr1);
                break;
            }
        }
        System.out.println(succeed ? "Nice!":"Fucking fucked!");
    }

    //绝对正确的方法
    public static void rightMathod(int[] arr) {
        Arrays.sort(arr);
    }

    //随机数组生成器，用于生成数据
    public static int[] generateRandomArray(int size, int value) {
        //Math.random() -> double [0,1)
        //(int)((size+1)*Math.random()) -> [0,size] 整数
        //size = 6, size + 1 = 7;
        //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;
```

```
        for(int i:arr){
            System.out.print(i+" ");
        }
        System.out.println();
    }
}

```

1.2 日期时间

1.2.1 Java8 中使用类计算日期差

```
import java.time.LocalDate;
import java.time.Month;
import java.time.Period;
import java.time.temporal.ChronoUnit;

public class Date {
    public static void main(String[] args) {
        LocalDate startDate = LocalDate.of(1993, Month.OCTOBER, 19);
        System.out.println("开始时间  : " + startDate);

        LocalDate endDate = LocalDate.of(2017, Month.JUNE, 16);
        System.out.println("结束时间 : " + endDate);

        long daysDiff = ChronoUnit.DAYS.between(startDate, endDate);
        System.out.println("两天之间的差在天数   : " + daysDiff);

        Period p = Period.between(startDate, endDate);
        System.out.printf("两天之间的差 : %d 年 %d 月 %d 日
    ", p.getYears(), p.getMonths(), p.getDays());
    }
}
```

输出结果：

```
开始时间  : 1993-10-19
结束时间 : 2017-06-16
两天之间的差在天数   : 8641
两天之间的差 : 23 年 7 月 28 日
```

1.2.2 Java8 中使用类计算时间差

```
import java.time.Duration;
import java.time.Instant;

public class Time {

    public static void main(String[] args) {
        Instant inst1 = Instant.now();
        System.out.println("Inst1 : " + inst1);
        Instant inst2 = inst1.plus(Duration.ofSeconds(10));
        System.out.println("Inst2 : " + inst2);

        System.out.println("Difference in milliseconds : " + Duration.betwee
n(inst1, inst2).toMillis());

        System.out.println("Difference in seconds : " + Duration.between(ins
t1, inst2).getSeconds());
    }
}
```

输出结果：

```
Inst1 : 2019-05-06T14:24:21.037Z
Inst2 : 2019-05-06T14:24:31.037Z
Difference in milliseconds : 10000
Difference in seconds : 10
```

2 数论

2.1 阶乘

```
public class Factorial {

    public static long factorial(long n){
        if(n==0)
            return 1;//0 的阶乘为 1
        else
            return factorial(n-1)*n;
```

```
    }
}
```

2.2 有关素数的基础算法

2.2.1 素数判定

```java
public class IsPrime {

    public static boolean isPrime(int n) {
        if (n < 2)
            return false;
        if (n == 2)
            return true;
        if (n % 2 == 0)
            return false;
        for (int i = 3; i * i <= n; i += 2)
            if (n % i == 0)
                return false;
        return true;
    }
}
```

2.2.2 素数的个数（埃氏筛法）

```java
public class Sieve {

    public static int[] prime = new int[10000000];
    public static boolean[] is_prime = new boolean[1000000 + 1];

    public static int sieve(int n) {
        int p = 0;
        for (int i = 0; i <= n; i++) is_prime[i] = true;
        is_prime[0] = is_prime[1] = false;
        for (int i = 2; i <= n; i++) {
            if (is_prime[i]) {
                prime[p++] = i;
                for (int j = 2 * i; j <= n; j += i) is_prime[j] = false;
            }
        }
        return p;
    }
```

```
}
```

2.3 辗转相除法

```
public class GCD {

    public static int gcd(int a, int b) {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }
}
```

2.4 快速幂

```
public class ModPow {

    public static long mod_pow(long x, long n, long mod) {
        if (n == 0)
            return 1;
        long res = mod_pow(x * x, n / 2, mod);
        if ((n & 1) == 1)
            res = res * x % mod;
        return res;
    }
}
```

2.5 矩阵快速幂

```
public class MatrixModPow {

    public static long[][] matrixModPow(int k, int n, long[][] A) {
        long[][] res = new long[n][n];
        for (int i = 0; i < res.length; i++) {
            for (int j = 0; j < res[i].length; j++) {
```

```
                if (i == j) {
                    res[i][j] = 1;
                } else {
                    res[i][j] = 0;
                }
            }
        }
        while (k != 0) {
            if ((k & 1) == 1) res = matMult(res, A);
            k >>= 1;//k/=2;
            A = matMult(A, A);
        }
        return res;
    }

    public static long[][] matMult(long[][] A, long[][] B) {
        long res[][] = new long[A.length][B.length];
        for (int i = 0; i < res.length; i++) {
            for (int j = 0; j < res[i].length; j++) {
                for (int k = 0; k < A[0].length; k++) {
                    res[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        return res;
    }
}
```

3 线性表&矩阵

3.1 全排列

```
public class Permutate {
    public static int total = 0;

    public static void swap(String[] str, int i, int j) {
        String temp = new String();
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
```

```
    }

    public static void arrange(String[] str, int st, int len) {
        if (st == len - 1) {
            for (int i = 0; i < len; i++) {
                System.out.print(str[i] + "  ");
            }
            System.out.println();
            total++;
        } else {
            for (int i = st; i < len; i++) {
                swap(str, st, i);
                arrange(str, st + 1, len);
                swap(str, st, i);
            }
        }
    }
}
```

3.2 快速排序

```
public class QuickSort {

    public static void quickSort(int[] num, int left, int right) {
        if (left < right) {
            int l = left;
            int r = right;
            int temp = num[left];
            while (l != r) {
                while (num[r] >= temp && l < r) r--;
                while (num[l] <= temp && l < r) l++;
                if (l < r) {
                    int t;
                    t = num[l];
                    num[l] = num[r];
                    num[r] = t;
                }
            }
            num[left] = num[l];
            num[l] = temp;
            quickSort(num, left, l - 1);
            quickSort(num, l + 1, right);
```

```
        }
    }
}
```

3.3 二分查找

```
//有序数组的二分查找
public class BinarySearch {

    //查找 v 出现的第一个位置
    public static int lowerBound(int[] nums, int l, int r, int v) {
        while (l < r) {
            int m = l + (r - l) / 2;
            if (nums[m] >= v)
                r = m;// 因为是寻找下界，不考虑右边还有没有元素
            else if (nums[m] < v)
                l = m + 1;
            if(l==r&&nums[l]!=v)//查找的数不存在，返回该数插入仍使数组有序的位置
                return -(m+1);
        }
        return l;
    }

    //查找 v 出现的最后一个位置
    public static int upperBound(int[] nums, int l, int r, int v) {
        while (l < r) {
            int m = l + (r - l) / 2;
            if (nums[m] <= v)
                l = m + 1;
            else if (nums[m] > v)
                r = m;
            if(l==r&&nums[m]!=v)
                return -(m+1);
        }
        return l;
    }
}
```

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. }
```

3.5 使用递归将栈倒置

```
1.  import java.util.Stack;
2.
3.  public class ReverseStackUsingRecursive {
4.
5.      public static void reverse(Stack<Integer> stack){
6.          if (stack.isEmpty())
7.              return;
8.          int i=getAndRemoveLastElement(stack);
9.          reverse(stack);
10.         stack.push(i);
11.     }
12.
13.     public static int getAndRemoveLastElement(Stack<Integer> stack){
14.         int result=stack.pop();
15.         if (stack.isEmpty()){
16.             return result;
```

```
        }else {
            int last=getAndRemoveLastElement(stack);
            stack.push(result);
            return last;
        }
    }
}
```

4 树

4.1 前缀树

```
import java.util.HashMap;

public class TrieTree {

    public static class TrieNode {
        public int pass;
        public int end;

        public HashMap<Integer, TrieNode> nexts;

        public TrieNode() {
            pass = 0;
            end = 0;
            nexts = new HashMap<Integer, TrieNode>();
        }
    }

    public static class Trie {
        private TrieNode root;

        public Trie() {
            root = new TrieNode();
        }

        public void insert(String word) {
            if (word == null)
                return;
            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;
```

```
        }

        public int preixNumber(String pre){
            if (pre==null){
                return 0;
            }
            char[] chs = pre.toCharArray();
            TrieNode node = root;
            int index = 0;
            for (int i = 0; i < chs.length; i++) {
                index=chs[i]-'a';
                if(!node.nexts.containsKey(index)){
                    return 0;
                }
                node=node.nexts.get(index);
            }
            return  node.pass;
        }
    }
}
```

5 图

5.1 图基本数据结构

5.1.1 边

```
public class Edge {
    public int value;
    public Node from;
    public Node to;

    public Edge(int value, Node from, Node to) {
        this.value = value;
        this.from = from;
        this.to = to;
    }
}
```

5.1.2 结点

```java
import java.util.ArrayList;

public class Node {
    public int value;
    public int in;
    public int out;
    public ArrayList<Node> nexts;
    public ArrayList<Edge> edges;

    public Node(int value) {
        this.value = value;
        in = 0;
        out = 0;
        nexts = new ArrayList<>();
        edges = new ArrayList<>();
    }

}
```

5.1.3 图

```java
import java.util.HashMap;
import java.util.HashSet;

public class Graph {

    public HashMap<Integer, Node> nodes;
    public HashSet<Edge> edges;

    public Graph() {
        nodes = new HashMap<>();
        edges = new HashSet<>();
    }
}
```

5.1.4 构造器

```java
public class GraphGenerator {

    public static Graph createGraph(Integer[][] matrix) {
        Graph graph = new Graph();
        for (int i = 0; i < matrix.length; i++) {
            Integer weight = matrix[i][0];
            Integer from = matrix[i][1];
            Integer to = matrix[i][2];
            if (!graph.nodes.containsKey(to)) {
                graph.nodes.put(to, new Node(to));
            }
            Node fromNode = graph.nodes.get(from);
            Node toNode = graph.nodes.get(to);
            Edge newEdge = new Edge(weight, fromNode, toNode);
            fromNode.nexts.add(toNode);
            fromNode.out++;
            toNode.in++;
            fromNode.edges.add(newEdge);
            graph.edges.add(newEdge);
        }
        return graph;
    }
}
```

5.2 Dijkstra

```java
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Vector;

class edge{
    int to;
    int cost;
}

class pair{
    int first;
    int second;
    pair(int n1,int n2){
```

```
        first=n1;
        second=n2;
    }
}

public class Main {

    public static int MAX_V=1000,INF=99999;
    public static int V=1000;
    public static int[] d=new int[MAX_V];
    public static Vector<edge>[] G=new Vector[MAX_V];

    public static void dijkstra(int s){
        for (int i = 0; i < V; i++) {
            d[i]=INF;
            G[i]=new Vector<edge>();
        }
        PriorityQueue<pair> que=new PriorityQueue<pair>(11,new Comparator<pa
ir>(){
            public int compare(pair p1,pair p2){
                return p1.first-p2.first;
            }
        });
        d[s]=0;
        que.offer(new pair(0,s));

        while(!que.isEmpty()){
            pair p=que.poll();
            int v=p.second;
            if(d[v]<p.first)
                continue;
            for(int i=0;i<G[v].size();i++){
                edge e=G[v].get(i);
                if(d[e.to]>d[v]+e.cost){
                    d[e.to]=d[v]+e.cost;
                    que.offer(new pair(d[e.to],e.to));
                }
            }
        }
    }
}
```

5.3 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;
```

```
        }

        for (int i = 0; i < V; i++) {
            for (int j = 0; j < E; j++) {
                edge e=es[j];
                if (d[e.to]>d[e.from]+e.cost) {
                    d[e.to]=d[e.from]+e.cost;

                    //如果第 n 次仍然更新了，则存在负圈
                    if(i==V-1)
                        return true;
                }
            }
        }
        return false;
    }
}
```

5.4 Floyd_Warshall

```
public class Main {

    public static int MAX_V=1000,INF=99999;
    public static int V=1000;//顶点数
    //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)
    public static int[][] d=new int[MAX_V][MAX_V];

    public static void warshall_floyd(){
        for (int k = 0; k < V; k++) {
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    d[i][j]=Math.min(d[i][j], d[i][k]+d[k][j]);
                }
            }
        }
    }
}
```

6 其他

6.1 并查集

```java
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

public class UnionSet {

    public static class Data {

    }

    public static class UnionFindSet {
        //(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
        public HashMap<Data, Data> fatherMap;
        public HashMap<Data, Integer> sizeMap;

        public UnionFindSet(List<Data> nodes) {
            fatherMap = new HashMap<Data, Data>();
            sizeMap = new HashMap<Data, Integer>();
            makeSets(nodes);
        }

        private void makeSets(List<Data> nodes) {
            fatherMap.clear();
            sizeMap.clear();
            for (Data node : nodes) {
                fatherMap.put(node, node);
                sizeMap.put(node, 1);
            }
        }

        private Data findHead(Data node) {
            Data father = fatherMap.get(node);
            if (father != node)
                father = findHead(father);
            fatherMap.put(node, father);
            return father;
        }
```

```
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. }
```