ACM Standard Code Library

Java edition

SiriYang

2019.5.5

目录

1 对数器

```java
import java.util.Arrays;

public class Main {
    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
        //double -> int [0,6] -> int

        //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. }
```

2 数论

2.1 阶乘

```
1.  public class Main {
2.
3.          public static long Factorial(long n) {
4.
5.          if(n==0)
6.              return 1;
7.          else
8.              return Factorial(n-1)*n;
9.      }
10. }
```

2.2 判断素数

```
1.  public class Main {
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 Main {
```

```
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;
```

```
    }

    public static long[][] f(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);
            }
```

```
        }
    }

    public static void main(String[] args) {
         String str[] = {"a","b","c"};
         arrange(str, 0, str.length);
         System.out.println(total);
    }
}
```

3.2 快速排序

```
public class Main {

        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

```
public class Main {

    //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[m]!=v)//查找的数不存在，返回改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

```
public class NetherlandsFlag {

    public static int[] partition(int[] arr, int L, int R, int num) {
        int less = L - 1;
        int more = R + 1;
        while (L < more) {
            if (arr[L] < num)
                swap(arr, ++less, L++);
            else if (arr[L] > num)
                swap(arr, --more, L);
            else
                L++;
        }
```

```
        return new int[]{less + 1, more - 1};
    }

    public static void swap(int[] arr, int i, int j) {
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
}
```

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)
```

```
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);
```

```
            }
            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;
        }
    }
}
```

4.2 并查集

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

public class Union {

    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>();
```

```
18.             makeSets(nodes);
19.         }
20.
21.         private void makeSets(List<Data> nodes) {
22.             fatherMap.clear();
23.             sizeMap.clear();
24.             for (Data node : nodes) {
25.                 fatherMap.put(node, node);
26.                 sizeMap.put(node, 1);
27.             }
28.         }
29.
30.         private Data findHead(Data node) {
31.             Data father = fatherMap.get(node);
32.             if (father != node)
33.                 father = findHead(father);
34.             fatherMap.put(node, father);
35.             return father;
36.         }
37.
38.         public boolean isSameSet(Data a, Data b) {
39.             return findHead(a) == findHead(b);
40.         }
41.
42.         public void union(Data a, Data b) {
43.             if (a == null || b == null)
44.                 return;
45.             Data aHead = findHead(a);
46.             Data bHead = findHead(b);
47.             if (aHead != bHead) {
48.                 int aSetSize = sizeMap.get(aHead);
49.                 int bSetSize = sizeMap.get(bHead);
50.                 if (aSetSize <= bSetSize) {
51.                     fatherMap.put(aHead, bHead);
52.                     sizeMap.put(bHead, aSetSize + bSetSize);
53.                 } else {
54.                     fatherMap.put(bHead, aHead);
55.                     sizeMap.put(aHead, aSetSize + bSetSize);
56.                 }
57.             }
58.         }
59.     }
60. }
```

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<pa
    ir>(){
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;
```

```
            for (int i = 0; i < E; i++) {
                edge e=es[i];
                if (d[e.from]!=INF&&d[e.to]>d[e.from]+e.cost) {
                    d[e.to]=d[e.from]+e.cost;
                    update=true;
                }
            }
            if (!update) {
                break;
            }
        }
    }

    //如果返回 true 则存在负圈
    public static boolean find_negative_loop(){
        for (int i = 0; i < d.length; i++) {
            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;
    }
}
```

5.3 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];
```

```
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. }
```