Java Standard Code Library

for ACM-ICPC

SiriYang

2019.5.7

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 链接下载使用。

模板终究只是赛场上的辅助工具，帮助选手快速正确的编写出代码。唯有在亲自编写运行过模板，熟悉每一个参数和代码后才能熟练运用，并根据不同的题目要求修改模板。所以在未完全掌握模板代码之前切勿盲目依赖模板！

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

参考资料：

左程云算法课程基础班教程

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 Java 常用类

1.2.1 日期时间

1.2.1.1 使用 Java8 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
```

1.2.1.2 使用 Java8 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

```
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）

```
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;
    }
}
```

2.2.3 分解质因数

```
public class DecompositionFactor {

    //求 n 的因数的个数
    public static int factorNum(int n) {
        if (n <= 0)
            return -1;
        int tot = 1;
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                int x = 0;
                while (n % i == 0) {
                    n /= i;
                    x++;
                }
                tot *= (x + 1);
            }
        }
        if (n > 1) tot *= 2;
        return tot;
    }

    //求 n 所有因数的和
    public static int factorSum(int n) {
        if (n <= 0)
            return -1;
        int tot = 1;
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                int mul = 1;
                while (n % i == 0) {
                    n /= i;
                    mul *= i;
                }
                tot *= (mul * i - 1) / (i - 1);
            }
```

```
        }
        if (n > 1) tot *= (n + 1);
        return tot;
    }
}
```

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 双向链表

```
1.  public class DoublyLinkedList {
2.
3.      public static class Node {
4.          public int value;
5.          public Node next;
6.          public Node last;
7.
8.          public Node(int value) {
9.              this.value = value;
10.             next = null;
11.             last = null;
12.         }
13.     }
14.
15.     public static void addHead(Node node1, Node node2) {
16.         Node head = getHead(node1);
17.         node2.next = head;
18.         head.last = node2;
19.     }
20.
21.     public static void addTail(Node node1, Node node2) {
22.         Node tail = node1;
23.         while (tail.next != null) tail = tail.next;
24.         node2.last = tail;
25.         tail.next = node2;
26.     }
27.
28.     public static void addBefor(Node node1, Node node2) {
29.         if (node1.last == null) {
30.             node2.next = node1;
31.             node1.last = node2;
32.         } else {
33.             node1.last.next = node2;
34.             node2.last = node1.last;
35.             node2.next = node1;
36.             node1.last = node2;
37.         }
38.     }
39.
40.     public static void addAfter(Node node1, Node node2) {
```

```
41.         if (node1.next == null) {
42.             node2.last = node1;
43.             node1.next = node2;
44.         } else {
45.             node1.next.last = node2;
46.             node2.next = node1.next;
47.             node2.last = node1;
48.             node1.next = node2;
49.         }
50.     }
51.
52.     public static Node deleteNode(Node node) {
53.         Node head = getHead(node);
54.         if (node.last == null) {
55.             head = node.next;
56.             node.next.last = null;
57.             node.next = null;
58.         } else if (node.next == null) {
59.             node.last.next = null;
60.             node.last = null;
61.         } else {
62.             node.last.next = node.next;
63.             node.next.last = node.last;
64.             node.next = null;
65.             node.last = null;
66.         }
67.         return head;
68.     }
69.
70.     public static Node deleteHead(Node node) {
71.         return deleteNode(getHead(node));
72.     }
73.
74.     public static Node deleteTail(Node node) {
75.         return deleteNode(getTail(node));
76.     }
77.
78.     public static Node getHead(Node node) {
79.         Node head = node;
80.         while (head.last != null) head = head.last;
81.         return head;
82.     }
83.
84.     public static Node getTail(Node node) {
```

```
85.         Node tail = node;
86.         while (tail.next != null) tail = tail.next;
87.         return tail;
88.     }
89.
90.     public static int length(Node node) {
91.         Node head = getHead(node);
92.         int length = 0;
93.         while (head != null) {
94.             length++;
95.             head = head.next;
96.         }
97.         return length;
98.     }
99. }
```

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;
```

```
            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.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 weight;
    public Node from;
    public Node to;

    public Edge(int weight, Node from, Node to) {
```

```
        this.weight = weight;
        this.from = from;
        this.to = to;
    }
}
```

5.1.2 结点

```
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

```
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

```
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(from)) {
                graph.nodes.put(from, new Node(from));
            }
            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;
    }
}
```

5.1.5 并查集（Node 版）

```
import java.util.Collection;
import java.util.HashMap;

public class UnionFind {
    private HashMap<Node, Node> fatherMap;
    private HashMap<Node, Integer> rankMap;
```

```
7.
8.      public UnionFind() {
9.          fatherMap = new HashMap<Node, Node>();
10.         rankMap = new HashMap<Node, Integer>();
11.     }
12.
13.     private Node findFather(Node n) {
14.         Node father = fatherMap.get(n);
15.         if (father != n) {
16.             father = findFather(father);
17.         }
18.         fatherMap.put(n, father);
19.         return father;
20.     }
21.
22.     public void makeSets(Collection<Node> nodes){
23.         fatherMap.clear();
24.         rankMap.clear();
25.         for (Node node:nodes){
26.             fatherMap.put(node,node);
27.             rankMap.put(node,1);
28.         }
29.     }
30.
31.     public boolean isSameSet(Node a,Node b){
32.         return findFather(a)==findFather(b);
33.     }
34.
35.     public void union(Node a,Node b){
36.         if (a == null || b == null)
37.             return;
38.         Node aFather = findFather(a);
39.         Node bFather = findFather(b);
40.         if (aFather != bFather) {
41.             int aFrank = rankMap.get(aFather);
42.             int bFrank = rankMap.get(bFather);
43.             if (aFrank <= bFrank) {
44.                 fatherMap.put(aFather, bFather);
45.                 rankMap.put(bFather, aFrank + bFrank);
46.             } else {
47.                 fatherMap.put(bFather, aFather);
48.                 rankMap.put(aFather, aFrank + bFrank);
49.             }
50.         }
```

```
    }
}
```

5.2 深度优先遍历

```
import java.util.HashSet;
import java.util.Stack;

public class DFS {

    public static void dfs(Node node) {
        if (node == null)
            return;
        Stack<Node> stack = new Stack<>();
        HashSet<Node> set = new HashSet<>();
        stack.add(node);
        set.add(node);
        System.out.println(node.value);//根据题目调整该行代码
        while (!stack.isEmpty()) {
            Node cur = stack.pop();
            for (Node next : cur.nexts) {
                if (!set.contains(next)) {
                    stack.push(cur);
                    stack.push(next);
                    set.add(next);
                    System.out.println(next.value);//根据题目调整该行代码
                    break;
                }
            }
        }
    }
}
```

5.3 广度优先遍历

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;

```

```java
public class BFS {

    public static void bfs(Node node) {
        if (node == null)
            return;
        Queue<Node> queue = new LinkedList<>();
        HashSet<Node> set = new HashSet<>();
        queue.add(node);
        set.add(node);
        while (!queue.isEmpty()) {
            Node cur = queue.poll();
            System.out.println(cur.value);//根据题目调整该行代码
            for (Node next : cur.nexts) {
                if (!set.contains(next)) {
                    set.add(next);
                    queue.add(next);
                }
            }
        }
    }
}
```

5.4 拓扑排序

```java
import java.util.*;

public class TopologySort {

    //directed graph and no loop
    public static List<Node> sortedTopology(Graph graph) {
        HashMap<Node, Integer> inMap = new HashMap<>();
        Queue<Node> zeroInQueue = new LinkedList<>();
        for (Node node : graph.nodes.values()) {
            inMap.put(node, node.in);
            if (node.in == 0)
                zeroInQueue.add(node);
        }
        List<Node> result = new ArrayList<>();
        while (!zeroInQueue.isEmpty()) {
            Node cur = zeroInQueue.poll();
            result.add(cur);
            for (Node next : cur.nexts) {
```

```
                inMap.put(next, inMap.get(next) - 1);
                if (inMap.get(next) == 0)
                    zeroInQueue.add(next);
            }
        }
        return result;
    }
}
```

5.5 最小生成树

5.5.1 Kruskal 最小生成树

```
import java.util.*;

public class KruskalMST {

    public  static class EdgeComparator implements Comparator<Edge>{
        @Override
        public int compare(Edge o1, Edge o2) {
            return o1.weight-o2.weight;
        }
    }

    public static Set<Edge> kruskalMST(Graph graph){
        UnionFind unionFind =new UnionFind();
        unionFind.makeSets(graph.nodes.values());
        PriorityQueue<Edge> priorityQueue=new PriorityQueue<>(new EdgeComparator());
        for (Edge edge:graph.edges){
            priorityQueue.add(edge);
        }
        Set<Edge> result=new HashSet<>();
        while (!priorityQueue.isEmpty()){
            Edge edge=priorityQueue.poll();
            if(!unionFind.isSameSet(edge.from,edge.to)){
                result.add(edge);
                unionFind.union(edge.from,edge.to);
            }
        }
        return result;
```

```
    }
}
```

5.5.2 Prim 最小生成树

```
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;
import java.util.Set;

public class PrimMST {

    public static class EdgeComparator implements Comparator<Edge> {
        @Override
        public int compare(Edge o1, Edge o2) {
            return o1.weight - o2.weight;
        }
    }

    public static Set<Edge> primMST(Graph graph) {
        PriorityQueue<Edge> priorityQueue = new PriorityQueue<>(new EdgeComparator());
        HashSet<Node> set = new HashSet<>();
        Set<Edge> result = new HashSet<>();
        for (Node node : graph.nodes.values()) {
            if (!set.contains(node)) {
                set.add(node);
                for (Edge edge : node.edges)
                    priorityQueue.add(edge);
                while (!priorityQueue.isEmpty()) {
                    Edge edge = priorityQueue.poll();
                    Node toNode = edge.to;
                    if (!set.contains(toNode)) {
                        set.add(toNode);
                        result.add(edge);
                        for (Edge nextEdge : toNode.edges)
                            priorityQueue.add(nextEdge);
                    }
                }
            }
        }
        return result;
```

```
    }
}
```

5.6 Dijkstra

```
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));
                }
            }
        }
    }
}
```

5.7 Bellman-Ford

```
//从顶点 from 指向顶点 to 的权值为 cost 的边
class edge{
    int from,to,cost;
}

public class Main {

    public static int MAX_V=1000,MAX_E,INF=99999;
    public static int V=1000,E=10000;//V 顶点数,E 边数
    public static int[] d=new int[MAX_V];//最短距离
    public static edge[] es=new edge[MAX_E];//边

    //求解从顶点 s 出发到所有点的最短距离
    public static void shortest_path(int s){
        for (int i = 0; i < V; i++) {
            d[i]=INF;
        }
        for (int i = 0; i < E; i++) {
            es[i]=new edge();
        }
        d[s]=0;

        while(true){
```

```
            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;
    }
}
```

5.8 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 其他数据结构

6.1 并查集

```
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) {
```

```
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. }
```

7 经典例题

7.1 常规

7.1.1 荷兰国旗问题

```java
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;
    }
}
```

7.2 递归

7.2.1 汉诺塔

```java
public class Hannota {

    public static void hannota(int n,String form,String to,String help){
        if (n==1){
            System.out.println("Move 1 from "+form+" to "+to);
```

```
            return;
        }else {
            hannota(n-1,form,help,to);
            System.out.println("Move "+n+" from "+form+" to "+to);
            hannota(n-1,help,to,form);
        }
    }
}
```

7.2.2 使用递归将栈倒置

```
import java.util.Stack;

public class ReverseStackUsingRecursive {

    public static void reverse(Stack<Integer> stack){
        if (stack.isEmpty())
            return;
        int i=getAndRemoveLastElement(stack);
        reverse(stack);
        stack.push(i);
    }

    public static int getAndRemoveLastElement(Stack<Integer> stack){
        int result=stack.pop();
        if (stack.isEmpty()){
            return result;
        }else {
            int last=getAndRemoveLastElement(stack);
            stack.push(result);
            return last;
        }
    }
}
```

7.3 贪心

7.4 动态规划

7.4.1 背包问题

7.4.1.1 01 背包

```java
import java.util.Arrays;

public class ZeroOneBackpackProblem {

    public static void main(String[] agrs) {

        n = 4;
        W = 5;
        w[0] = 2;v[0] = 3;
        w[1] = 1;v[1] = 2;
        w[2] = 3;v[2] = 4;
        w[3] = 2;v[3] = 2;

        init(-1);//记忆搜索初始化为-1
        System.out.println(zeroOne(0, W));

        init(0);//dp 初始化为 0
        zeroOne2();
        System.out.println(dp[0][W]);

        Arrays.fill(dp2,0);
        zeroOne3();
        System.out.println(dp2[W]);
    }

    public static int MAX_N = 100;//物品个数
    public static int MAX_W = 1000;//背包重量
    public static int n;//物品个数
    public static int W;//背包容量
    public static int[] w = new int[MAX_N];//物品重量
    public static int[] v = new int[MAX_N];//物品价值

    //还未初始化，用之前一定要记得使用初始化
```

```
34.     public static int[][] dp = new int[MAX_N + 1][MAX_W + 1];
35.
36.     //初始化 dp 矩阵
37.     //Arrays.fill 函数本质也是循环，并不能加速初始化且只能初始化一维数组
38.     public static void init(int v) {
39.         for (int i = 0; i <= MAX_N; i++)
40.             for (int j = 0; j <= MAX_W; j++)
41.                 dp[i][j] = v;
42.     }
43.
44.     //递归记忆搜索方法 O(nW)
45.     public static int zeroOne(int i, int j) {
46.         if (dp[i][j] >= 0) {
47.             //已经经过计算的话直接使用之前的结果
48.             return dp[i][j];
49.         }
50.         int res;
51.         if (i == n) {
52.             //已经没有剩余物品了
53.             res = 0;
54.         } else if (j < w[i]) {
55.             //无法挑选这个物品
56.             res = zeroOne(i + 1, j);
57.         } else {
58.             //挑选和不挑选两种情况都尝试一下
59.             res = Math.max(zeroOne(i + 1, j), zeroOne(i + 1, j - w[i]) + v[i
    ]);
60.         }
61.         return res;
62.     }
63.
64.     //dp 方法 O(nW)
65.     public static void zeroOne2() {
66.         for (int i = n - 1; i >= 0; i--) {
67.             for (int j = 0; j <= W; j++) {
68.                 if (j < w[i]) {
69.                     dp[i][j] = dp[i + 1][j];
70.                 } else {
71.                     dp[i][j] = Math.max(dp[i + 1][j], dp[i + 1][j - w[i]] +
    v[i]);
72.                 }
73.             }
74.         }
75.     }
```

```
76.
77.     //dp 方法 2，只用一维数组
78.     public static int[] dp2 = new int[MAX_N + 1];
79.
80.     public static void zeroOne3() {
81.         for (int i = 0; i < n; i++) {
82.             for (int j = W; j >= w[i]; j--) {
83.                 dp2[j] = Math.max(dp2[j], dp2[j - w[i]] + v[i]);
84.             }
85.         }
86.     }
87. }
```

输入数据：

```
1.  n = 4
2.  (w, v) = {(2, 3), (1, 2), (3, 4), (2, 2)}
3.  W=5
```

输出结果：

```
1.  7(选择第 0、1、3 号物品)
```

7.4.1.2 完全背包问题

```
1.  import java.util.Arrays;
2.
3.  public class FullBackpackProblem {
4.
5.      public static void main(String[] agrs) {
6.
7.          n = 3;
8.          W = 7;
9.          w[0] = 3;v[0] = 4;
10.         w[1] = 4;v[1] = 5;
11.         w[2] = 2;v[2] = 3;
12.
13.         init(0);
14.         full();
15.         System.out.println(dp[n][W]);
16.
```

```
17.         Arrays.fill(dp2,0);
18.         full2();
19.         System.out.println(dp2[W]);
20.     }
21.
22.     public static int MAX_N = 100;//物品个数
23.     public static int MAX_W = 1000;//背包重量
24.     public static int n;//物品个数
25.     public static int W;//背包容量
26.     public static int[] w = new int[MAX_N];//物品重量
27.     public static int[] v = new int[MAX_N];//物品价值
28.
29.     //还未初始化，用之前一定要记得使用初始化
30.     public static int[][] dp = new int[MAX_N + 1][MAX_W + 1];
31.
32.     //初始化 dp 矩阵
33.     public static void init(int v) {
34.         for (int i = 0; i <= MAX_N; i++)
35.             for (int j = 0; j <= MAX_W; j++)
36.                 dp[i][j] = v;
37.     }
38.
39.     public static void full() {
40.         for (int i = 0; i < n; i++)
41.             for (int j = 0; j <= W; j++) {
42.                 if (j < w[i]) {
43.                     dp[i + 1][j] = dp[i][j];
44.                 } else {
45.                     dp[i + 1][j] = Math.max(dp[i][j], dp[i + 1][j - w[i]] +
    v[i]);
46.                 }
47.             }
48.     }
49.
50.     //只用一维数组
51.     public static int[] dp2 = new int[MAX_N + 1];
52.     public static void full2() {
53.         for (int i = 0; i < n; i++)
54.             for (int j = w[i]; j <= W; j++) {
55.                 dp2[j] = Math.max(dp2[j], dp2[j - w[i]] + v[i]);
56.             }
57.     }
58. }
```

输入数据：

```
n = 3
(w, v) = {(3, 4), (4, 5), (2, 3)}
W = 7
```

输出结果：

```
10（0 号物品选 1 个，2 号物品选 2 个）
```

7.4.2 最长公共子序列

```
public class LongestCommonCubsequence {

    public static void main(String[] args) {

        s="leiliu";
        t="lqweileoui";

        longestCommonCubsequence();
        System.out.println(dp[s.length()][t.length()]);
    }

    public static String s;
    public static String t;

    public static int MAX_N = 100;
    public static int MAX_M = 100;
    public static int[][] dp = new int[MAX_N + 1][MAX_M + 1];

    public static void longestCommonCubsequence() {
        for (int i = 0; i < s.length(); i++)
            for (int j = 0; j < t.length(); j++) {
                if (s.charAt(i) == t.charAt(j)) {
                    dp[i + 1][j + 1] = dp[i][j]+1;
                } else {
                    dp[i + 1][j + 1] = Math.max(dp[i][j + 1], dp[i + 1][j]);

                }
            }
    }
```

```
29. }
```

输入数据：

```
1.  s="leiliu"
2.  t="lqweileoui"
```

输出结果：

```
1.  5
```