

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：

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	4
1.1 对数器	4
1.2 日期时间	6
1.2.1 Java8 中使用类计算日期差	6
1.2.2 Java8 中使用类计算时间差	7
2 数论	7
2.1 阶乘	7
2.2 有关素数的基础算法	8
2.2.1 素数判定	8
2.2.2 素数的个数（埃氏筛法）	8
2.3 辗转相除法	9
2.4 快速幂	9
2.5 矩阵快速幂	9
3 线性表&矩阵	10
3.1 全排列	10
3.2 快速排序	11
3.3 二分查找	12
3.4 荷兰国旗问题	13
3.5 使用递归将栈倒置	13
4 树	14
4.1 前缀树	14
5 图	16
5.1 图基本数据结构	16
5.1.1 边	16
5.1.2 结点	17
5.1.3 图	17
5.1.4 构造器	18
5.1.5 并查集（Node 版）	18
5.2 深度优先遍历	20
5.3 广度优先遍历	20
5.4 拓扑排序	21
5.5 最小生成树	22
5.5.1 Kruskal 最小生成树	22
5.5.2 Prim 最小生成树	23
5.6 Dijkstra	24
5.7 Bellman-Ford	25
5.8 Floyd_Warshall	26
6 其他	27
6.1 并查集	27

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.

```

1.2 日期时间

1.2.1 Java8 中使用类计算日期差

```

1. import java.time.LocalDate;
2. import java.time.Month;
3. import java.time.Period;
4. import java.time.temporal.ChronoUnit;
5.
6. public class Date {
7.     public static void main(String[] args) {
8.         LocalDate startDate = LocalDate.of(1993, Month.OCTOBER, 19);
9.         System.out.println("开始时间 : " + startDate);
10.
11.         LocalDate endDate = LocalDate.of(2017, Month.JUNE, 16);
12.         System.out.println("结束时间 : " + endDate);
13.
14.         long daysDiff = ChronoUnit.DAYS.between(startDate, endDate);
15.         System.out.println("两天之间的差在天数 : " + daysDiff);
16.
17.         Period p = Period.between(startDate, endDate);
18.         System.out.printf("两天之间的差 : %d 年 %d 月 %d 日
19.             ", p.getYears(), p.getMonths(), p.getDays());
20.     }
21. }

```

输出结果：

```

1. 开始时间 : 1993-10-19
2. 结束时间 : 2017-06-16
3. 两天之间的差在天数 : 8641
4. 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

```
1. import java.time.Duration;
2. import java.time.Instant;
3.
4. public class Time {
5.
6.     public static void main(String[] args) {
7.         Instant inst1 = Instant.now();
8.         System.out.println("Inst1 : " + inst1);
9.         Instant inst2 = inst1.plus(Duration.ofSeconds(10));
10.        System.out.println("Inst2 : " + inst2);
11.
12.        System.out.println("Difference in milliseconds : " + Duration.between(
            inst1, inst2).toMillis());
13.
14.        System.out.println("Difference in seconds : " + Duration.between(in
            st1, inst2).getSeconds());
15.    }
16. }
```

输出结果：

```
1. Inst1 : 2019-05-06T14:24:21.037Z
2. Inst2 : 2019-05-06T14:24:31.037Z
3. Difference in milliseconds : 10000
4. Difference in seconds : 10
```

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 有关素数的基础算法

2.2.1 素数判定

```
1. public class IsPrime {  
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. }
```

2.2.2 素数的个数（埃氏筛法）

```
1. public class Sieve {  
2.  
3.     public static int[] prime = new int[10000000];  
4.     public static boolean[] is_prime = new boolean[10000000 + 1];  
5.  
6.     public static int sieve(int n) {  
7.         int p = 0;  
8.         for (int i = 0; i <= n; i++) is_prime[i] = true;  
9.         is_prime[0] = is_prime[1] = false;  
10.        for (int i = 2; i <= n; i++) {  
11.            if (is_prime[i]) {  
12.                prime[p++] = i;  
13.                for (int j = 2 * i; j <= n; j += i) is_prime[j] = false;  
14.            }  
15.        }  
16.        return p;  
17.    }
```

```
18. }
```

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 ModPow {  
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 MatrixModPow {  
2.  
3.     public static long[][] matrixModPow(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 = matMult(res, A);
16.     k >>= 1; //k/=2;
17.     A = matMult(A, A);
18. }
19. return res;
20. }
21.
22. public static long[][] matMult(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.
4.     public static void swap(String[] str, int i, int j) {
5.         String temp = new String();
6.         temp = str[i];
7.         str[i] = str[j];
8.         str[j] = temp;

```

```

9.     }
10.
11.     public static void arrange(String[] str, int st, int len) {
12.         if (st == len - 1) {
13.             for (int i = 0; i < len; i++) {
14.                 System.out.print(str[i] + " ");
15.             }
16.             System.out.println();
17.             total++;
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. }

```

3.2 快速排序

```

1. public class QuickSort {
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. //有序数组的二分查找
2. public class BinarySearch {
3.
4.     //查找 v 出现的第一个位置
5.     public static int lowerBound(int[] nums, int l, int r, int v) {
6.         while (l < r) {
7.             int m = l + (r - l) / 2;
8.             if (nums[m] >= v)
9.                 r = m; // 因为是寻找下界，不考虑右边还有没有元素
10.            else if (nums[m] < v)
11.                l = m + 1;
12.            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
13.                return -(m+1);
14.        }
15.        return l;
16.    }
17.
18.    //查找 v 出现的最后一个位置
19.    public static int upperBound(int[] nums, int l, int r, int v) {
20.        while (l < r) {
21.            int m = l + (r - l) / 2;
22.            if (nums[m] <= v)
23.                l = m + 1;
24.            else if (nums[m] > v)
25.                r = m;
26.            if(l==r&&nums[m]!=v)
27.                return -(m+1);
28.        }
29.        return l;
30.    }
31. }

```

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;
```

```

17.         }else {
18.             int last=getAndRemoveLastElement(stack);
19.             stack.push(result);
20.             return last;
21.         }
22.     }
23. }

```

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 图基本数据结构

5.1.1 边

```

1. public class Edge {
2.     public int weight;
3.     public Node from;
4.     public Node to;
5.
6.     public Edge(int weight, Node from, Node to) {
7.         this.weight = weight;
8.         this.from = from;
9.         this.to = to;
10.    }
11. }

```

5.1.2 结点

```
1. import java.util.ArrayList;
2.
3. public class Node {
4.     public int value;
5.     public int in;
6.     public int out;
7.     public ArrayList<Node> nexts;
8.     public ArrayList<Edge> edges;
9.
10.    public Node(int value) {
11.        this.value = value;
12.        in = 0;
13.        out = 0;
14.        nexts = new ArrayList<>();
15.        edges = new ArrayList<>();
16.    }
17.
18. }
```

5.1.3 图

```
1. import java.util.HashMap;
2. import java.util.HashSet;
3.
4. public class Graph {
5.
6.     public HashMap<Integer, Node> nodes;
7.     public HashSet<Edge> edges;
8.
9.     public Graph() {
10.         nodes = new HashMap<>();
11.         edges = new HashSet<>();
12.     }
13. }
```

5.1.4 构造器

```
1. public class GraphGenerator {
2.
3.     public static Graph createGraph(Integer[][] matrix) {
4.         Graph graph = new Graph();
5.         for (int i = 0; i < matrix.length; i++) {
6.             Integer weight = matrix[i][0];
7.             Integer from = matrix[i][1];
8.             Integer to = matrix[i][2];
9.             if (!graph.nodes.containsKey(from)) {
10.                 graph.nodes.put(from, new Node(from));
11.             }
12.             if (!graph.nodes.containsKey(to)) {
13.                 graph.nodes.put(to, new Node(to));
14.             }
15.             Node fromNode = graph.nodes.get(from);
16.             Node toNode = graph.nodes.get(to);
17.             Edge newEdge = new Edge(weight, fromNode, toNode);
18.             fromNode.nexts.add(toNode);
19.             fromNode.out++;
20.             toNode.in++;
21.             fromNode.edges.add(newEdge);
22.             graph.edges.add(newEdge);
23.         }
24.         return graph;
25.     }
26. }
```

5.1.5 并查集 (Node 版)

```
1. import java.util.Collection;
2. import java.util.HashMap;
3.
4. public class UnionFind {
5.     private HashMap<Node, Node> fatherMap;
6.     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.         }
51.     }
52. }

```

5.2 深度优先遍历

```
1. import java.util.HashSet;
2. import java.util.Stack;
3.
4. public class DFS {
5.
6.     public static void dfs(Node node) {
7.         if (node == null)
8.             return;
9.         Stack<Node> stack = new Stack<>();
10.        HashSet<Node> set = new HashSet<>();
11.        stack.add(node);
12.        set.add(node);
13.        System.out.println(node.value); //根据题目调整该行代码
14.        while (!stack.isEmpty()) {
15.            Node cur = stack.pop();
16.            for (Node next : cur.nexts) {
17.                if (!set.contains(next)) {
18.                    stack.push(cur);
19.                    stack.push(next);
20.                    set.add(next);
21.                    System.out.println(next.value); //根据题目调整该行代码
22.                    break;
23.                }
24.            }
25.        }
26.    }
27. }
```

5.3 广度优先遍历

```
1. import java.util.HashSet;
2. import java.util.LinkedList;
3. import java.util.Queue;
4.
5. public class BFS {
6.
7.     public static void bfs(Node node) {
8.         if (node == null)
9.             return;
```

```

10.     Queue<Node> queue = new LinkedList<>();
11.     HashSet<Node> set = new HashSet<>();
12.     queue.add(node);
13.     set.add(node);
14.     while (!queue.isEmpty()) {
15.         Node cur = queue.poll();
16.         System.out.println(cur.value); //根据题目调整该行代码
17.         for (Node next : cur.nexts) {
18.             if (!set.contains(next)) {
19.                 set.add(next);
20.                 queue.add(next);
21.             }
22.         }
23.     }
24. }
25. }

```

5.4 拓扑排序

```

1. import java.util.*;
2.
3. public class TopologySort {
4.
5.     //directed graph and no loop
6.     public static List<Node> sortedTopology(Graph graph) {
7.         HashMap<Node, Integer> inMap = new HashMap<>();
8.         Queue<Node> zeroInQueue = new LinkedList<>();
9.         for (Node node : graph.nodes.values()) {
10.             inMap.put(node, node.in);
11.             if (node.in == 0)
12.                 zeroInQueue.add(node);
13.         }
14.         List<Node> result = new ArrayList<>();
15.         while (!zeroInQueue.isEmpty()) {
16.             Node cur = zeroInQueue.poll();
17.             result.add(cur);
18.             for (Node next : cur.nexts) {
19.                 inMap.put(next, inMap.get(next) - 1);
20.                 if (inMap.get(next) == 0)
21.                     zeroInQueue.add(next);
22.             }
23.         }

```

```
24.         return result;
25.     }
26. }
```

5.5 最小生成树

5.5.1 Kruskal 最小生成树

```
1. import java.util.*;
2.
3. public class KruskalMST {
4.
5.     public static class EdgeComparator implements Comparator<Edge>{
6.         @Override
7.         public int compare(Edge o1, Edge o2) {
8.             return o1.weight-o2.weight;
9.         }
10.    }
11.
12.    public static Set<Edge> kruskalMST(Graph graph){
13.        UnionFind unionFind =new UnionFind();
14.        unionFind.makeSets(graph.nodes.values());
15.        PriorityQueue<Edge> priorityQueue=new PriorityQueue<>(new EdgeCompar
        ator());
16.        for (Edge edge:graph.edges){
17.            priorityQueue.add(edge);
18.        }
19.        Set<Edge> result=new HashSet<>();
20.        while (!priorityQueue.isEmpty()){
21.            Edge edge=priorityQueue.poll();
22.            if(!unionFind.isSameSet(edge.from,edge.to)){
23.                result.add(edge);
24.                unionFind.union(edge.from,edge.to);
25.            }
26.        }
27.        return result;
28.    }
29. }
```

5.5.2 Prim 最小生成树

```
1. import java.util.Comparator;
2. import java.util.HashSet;
3. import java.util.PriorityQueue;
4. import java.util.Set;
5.
6. public class PrimMST {
7.
8.     public static class EdgeComparator implements Comparator<Edge> {
9.         @Override
10.        public int compare(Edge o1, Edge o2) {
11.            return o1.weight - o2.weight;
12.        }
13.    }
14.
15.    public static Set<Edge> primMST(Graph graph){
16.        PriorityQueue<Edge> priorityQueue = new PriorityQueue<>(new EdgeComp
17.            arator());
18.        HashSet<Node> set = new HashSet<>();
19.        Set<Edge> result = new HashSet<>();
20.        for (Node node : graph.nodes.values()) {
21.            if (!set.contains(node)) {
22.                set.add(node);
23.                for (Edge edge : node.edges)
24.                    priorityQueue.add(edge);
25.                while (!priorityQueue.isEmpty()) {
26.                    Edge edge = priorityQueue.poll();
27.                    Node toNode = edge.to;
28.                    if (!set.contains(toNode)) {
29.                        set.add(toNode);
30.                        result.add(edge);
31.                        for (Edge nextEdge : toNode.edges)
32.                            priorityQueue.add(nextEdge);
33.                    }
34.                }
35.            }
36.            return result;
37.        }
38.    }
```


5.6 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. }

```

5.7 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. }

```

5.8 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,
13.         Data_A 的父节点是 Data_B
14.         public HashMap<Data, Data> fatherMap;
15.         public HashMap<Data, Integer> sizeMap;
16.
17.         public UnionFindSet(List<Data> nodes) {
18.             fatherMap = new HashMap<Data, Data>();
19.             sizeMap = new HashMap<Data, Integer>();
20.             makeSets(nodes);
21.         }
22.
23.         private void makeSets(List<Data> nodes) {
24.             fatherMap.clear();
25.             sizeMap.clear();
26.             for (Data node : nodes) {
27.                 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. }

```