算法知识总结

张兴锐

May 24, 2018

Contents

1 暴力法

暴力穷举**填空题**常用。

- 暴力破解中的实用性原则
- 枚举法, 从前往后推。
- 逆向解法，从后往前推。

1.1 枚举法：年龄问题

美国数学家维纳 (N.Wiener) 智力早熟，11 岁就上了大学。他曾在 1935 1936 年应邀来中国清华大学讲学。一次，他参加某个重要会议，年轻的脸孔引人注目。于是有人询问他的年龄，他回答说：“我年龄的立方是个 4 位数。我年龄的 4 次方是个 6 位数。这 10 个数字正好包含了从 0 到 9 这 10 个数字，每个都恰好出现 1 次。”请你推算一下，他当时到底有多年轻?

```
package algorithm.violence;

import java.util.HashSet;
import java.util.Set;

/**
 * 美国数学家维纳(N.Wiener)智力早熟， 11 岁就上了大学。 他曾在 1935~1936 年应 邀来中
     国清华大学讲学。
 * 一次，他参加某个重要会议，年轻的脸孔引人注目。 于是 有人询问他的年龄，他回答说:
     “我年龄的立方是个 4 位数。我年龄的 4 次方是个 6 位数。
 * 这 10 个数字正好包含了从 0 到 9 这 10 个数字，每个都恰好出现 1 次。 ” 请你推算一
     下，他当时到底有多年轻
 * @author zxr
 */
public class AgeProblem {
    public static void main(String[] args) {
        for (int i = 11; i <=100; i++) {
            String r1 = Integer.toString((int)Math.pow(i, 3));
            String r2 = Integer.toString((int)Math.pow(i, 4));
            if (r1.length() == 4 && r2.length() == 6) {
                String r = r1 + r2;
                if (check(r)) {
                    System.out.println(i);
                    break;
                }
            }
        }
    }
```

```
    private static boolean check(String r) {
        // TODO Auto-generated method stub
        Set<Character> sets = new HashSet<Character>();
        for(int i = 0;i<r.length();i++) {
            sets.add(r.charAt(i));
        }
        if(sets.size() == 10) {
            return true;
        }
        return false;
    }
}
```

1.2 水仙花数

这里不给出具体代码，对于这个问题既可以逆向思维也可以正向推。比如求解三位数的水仙花数字：可以从 100-999 进行循环取出各个位数进行比较，也可以嵌套撒能循环，i1=0-9,i2=0-9,i3=0-9。这个三个数字生成的数字的是否满足条件。两种情况时间复杂度也是相同的，都要循环 1000 次。

2 递归解法

递归解法用处广泛，不够总体上分为两种：

1. **最终数量型**，对于这种问题，主要是考虑问题递归的前后联系关系，尽量不要使用到公共结构和全局变量，因为不需要输出最终结果。**思想常常是当前状态下可有几种选择，有几种就 + 上几种进行迭代，当然也需要设计好出口条件**
   (a) 求解全排列的个数，或者求解组合个数。
2. **最终结果型**，或者在上述情况下无法方便求出的情况下，可以考虑这种方法，常常需要使用到公共数据结构和全局变量，**回溯法特别常用**。
   (a) 具有边界条件的问题，如裁剪问题
   (b) 输出全排列，利用交换思想或者利用填数思想。
   (c) 输出具有重复数字的组合。
   (d) n 皇后问题，等等。

2.1 类型一：求解最终数量

2.1.1 观光台售票问题

公园票价为 5 角。假设每位游客只持有两种币值的货币：5 角、1 元。再假设持有 5 角的有 m 人，持有 1 元的有 n 人。由于特殊情况，开始的时候，售票员没有零钱可找。我们想知道这 m+n 名游客以什么样的顺序购票则可以顺利完成购票过程。显然，m < n 的时候，无论如何都不能完成；m>=n 的时候，有些情况也不行。比如，第一个购票的乘客就持有 1 元。请计算出这 m+n 名游客所有可能顺利完成购票的不同情况的组合数目。注意：只关心 5 角和 1 元交替出现的次序的不同排列，持有同样币值的两名游客交换位置并不算做一种新的情况来计数。

```java
package algorithm.recursion.course;

/**
 * 找零钱
 *
 * @author zxr
 *
 */
public class ChangeProblem {
    public static void main(String[] args) {
        int n = 1;
        int m = 2;
        System.out.println(f(n, m));
        System.out.println(f(n, m, 0));
    }

    /**
     * 从后往前追溯
     *
     * @param n
     * 持有1元的
     * @param m
     * 持有5角的
     * @return 当前队列有多少种可能的解法
     */
    public static int f(int n, int m) {
        if (m < n) {
            return 0;
        }
        if (n == 0) {
            return 1;
        }
        return f(n - 1, m) + f(n, m - 1);
    }

    public static int f(int n, int m, double sum) {
        if (sum < 0) {
            return 0;
        }
        if (n == 0) {
            return 1;
        }
        if (m == 0) {
            return f(n - 1, 0, sum-0.5);
        }
        // 当前位置排列1
        int count = f(n - 1, m, sum - 0.5);
        //当前位置排0
        return count + f(n, m - 1, sum + 0.5);
    }
}
```

2.1.2 甲虫出站问题

```java
package algorithm.recursion.course;

public class OutStack {


    public static void main(String[] args) {
```

```
        System.out.println(f(5,0));
    }
    /**
     *
     * @param n 等待进站
     * @param m 已经进站
     * @return
     */
    public static int f(int n,int m) {
        if(n == 0) {
            return 1;
        }
        if(m == 0) {
            return f(n-1,1);
        }
        return f(n-1,m+1) + f(n,m-1);
    }

}
```

2.2 类型二：最终结果型

2.2.1 求解全排列

暴力破解中常用到，所以特别指出

```
package problems;
import java.math.BigDecimal;
import java.text.Bidi;
/**
 * 测试类，写算法过程中，一些不确定的东西可以用来测试
 *
 * @author zxr
 *
 */
/*
 * JAVA中的无穷大和NaN的构造方法
 */
public class Test {
    public static double NaN = 0.0 / 0.0;
    public static double INF = 1.0 / 0.0;
    public static void main(String[] args) {
        String origin = "ABCDE";
        char[] o = origin.toCharArray();
        f(o,0);
        System.out.println(count);
    }

    static int count;
    public static void f(char[] o,int index) {
        if(index == o.length) {
            System.out.println(o);
            count++;
            return;
        }
        for(int i = index;i<o.length;i++) {
            char c = o[i];o[i] = o[index];o[index] = c;
            f(o,index+1);
            c = o[i];o[i] = o[index];o[index] = c;
        }
    }
```

```
}
```

2.2.2 具有重复字母的排列问题

给定一个具有重复数字的字符集合，如 “AABBCCD”，从中取 4 个数字，有多少种解法。

```
package algorithm.recursion.course;

import java.util.Arrays;

/**
 * "AABBC" 取三个数字，能够取多少种
 * @author zxr
 *
 */
public class TakeNumProblem {
    static int count = 0;
    public static void main(String[] args) {
        int[] data = {2,2,1};
        int[] x = new int[data.length];
        f(data,x,0,3);
        System.out.println(count);
    }
    public static void f(int[] data,int[] x,int index,int target) {
        if(sum(x) == target) {
            count++;
            System.out.println(Arrays.toString(x));
            return;
        }
        if(index == 3) {
            return;
        }
        for(int i = 0;i<=data[index];i++) {
            x[index] = i;
            f(data,x,index+1,target);
        }
    }
    private static int sum(int[] x) {
        // TODO Auto-generated method stub
        int sum = 0;
        for (int i : x) {
            sum+=i;
        }
        return sum;
    }
}
```

2.2.3 裁剪纸问题

```
package problems.zhenti.b2017;
/**
 * 寻找对称图案
 * @author 张兴锐
 *
 */
public class Probleam4 {
    public static void main(String[] args) {
        visted[3][3] = true;
```

```
        deep(3,3);
        System.out.println(count/4);
    }
    static boolean[][] visted = new boolean[7][7];
    static int count = 0;

    public static void deep(int x,int y){
        if(x == 6 || y == 6 || x == 0 || y == 0){
            count++;
            return;
        }
        int i = 0;
        int saveX = x;
        int saveY = y;
        while(i <= 3){
            if(i == 0){
                //左
                x -= 1;
            }else if(i == 1){
                //右
                x += 1;
            }else if(i == 2){
                //上
                y -= 1;
            }else if(i == 3){
                //下
                y+=1;
            }
            if(!visted[x][y]){
                visted[x][y] = true;
                visted[6-x][6-y] = true;
                deep(x,y);
                //回溯
                visted[x][y] = false;
                visted[6-x][6-y] = false;
            }
            x = saveX;
            y = saveY;
            i++;
        }


    }
}
```

3 算法中的数学问题

3.1 进制的巧妙使用

在题目中有明显的进制迭代关系，可以考虑使用本方法.

3.1.1 捐赠问题

地产大亨 Q 先生临终的遗愿是：拿出 100 万元给 X 社区的居民抽奖，以稍慰藉心中愧疚。麻烦的是，他有个很奇怪的要求：1. 100 万元必须被正好分成若干份（不能剩余）。

每份必须是 7 的若干次方元。比如：1 元, 7 元，49 元，343 元，...

2. 相同金额的份数不能超过 5 份。

3. 在满足上述要求的情况下，分成的份数越多越好！

请你帮忙计算一下，最多可以分为多少份？

对于本题，*1,7, 49,343* 是明显的 *7* 的次方，可以采用 *7* 进制进行求解。

```
package algorithm.math;

/**
 * 地产大亨Q先生临终的遗愿是：拿出100万元给X社区的居民抽奖，以稍慰藉心中愧疚。 麻烦的是，他有个很奇怪的要求： 1.
 * 100万元必须被正好分成若干份（不能剩余）。 每份必须是7的若干次方元。 比如：1元，7元，49元，343元，...
 *
 * 2. 相同金额的份数不能超过5份。
 *
 * 3. 在满足上述要求的情况下，分成的份数越多越好！
 *
 * 请你帮忙计算一下，最多可以分为多少份？
 *
 * 采用进行转换，巧妙解法
 * @author zxr
 *
 */
public class FenQian {

   public static void main(String[] args) {
      // TODO Auto-generated method stub
      String s = Integer.toString(1000*1000, 7);
      System.out.println(s);
      int sum = 0;
      for(int i =0;i<s.length();i++) {
         sum+=s.charAt(i)-'0';
      }
      System.out.println(sum);
   }

}
```

3.1.2 天平称重问题-改造的 3 进制问题

用天平称重时，我们希望用尽可能少的砝码组合称出尽可能多的重量。如果只有 5 个砝码，重量分别是 1，3，9，27，81 则它们可以组合称出 1 到 121 之间任意整数重量（砝码允许放在左右两个盘中）。本题目要求编程实现：对用户给定的重量，给出砝码组合方案。例如：

用户输入：

5

程序输出：

9-3-1

用户输入：

19

程序输出：

27-9+1

要求程序输出的组合总是大数在前小数在后。

可以假设用户的输入的数字符合范围 1 121。

```
package algorithm.math;

public class TianPingProblem {
```

```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for(int i =1;i<100;i++) {
            System.out.println(i+":"+f(i));
        }

    }
    public static String f(int value) {
        String ans = "";
        int sh = value / 3;
        int k = 0;
        while (value != 0) {
            if (value % 3 == 2) {
                sh++;
                ans = "-" + ((int)Math.pow(3, k)) + ans;
            } else if (value % 3 == 1) {
                ans = "+" + ((int)Math.pow(3, k)) + ans;
            } else if (value % 3 == 0) {
//    ans = ans;
            }
            value = sh;
            sh = value / 3;
            k++;
        }

        return ans.substring(1);
    }

}
```

3.2 最小最大公约数

最大公约数欧几里得定理：辗转相处法。$gcd(a,b) = gcd(b, a\%b)$ $b = 0$ **最小公倍数** $a * b/gcd(a,b)$

3.3 无误差有理数计算

如果求 $1/2 + 1/3 + 1/4 + 1/5 + 1/6 + ... + 1/100 = ?$ 要求绝对精确，不能有误差。

```
package algorithm.math;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
/**
 * 精度问题
 *
 * @author zxr
 *
 */
public class FuDianJingDu {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Ra sum = new Ra(BigInteger.ZERO);
        for(int i = 2;i<=100;i++) {
```

```
            sum=sum.add(new Ra(BigInteger.ONE,new BigInteger(i+"")));
        }
        System.out.println(sum);
    }



    static class Ra {
        BigInteger[] num = { BigInteger.ZERO, BigInteger.ONE };

        /**
         * 构造函数
         */
        public Ra() {
        }

        public Ra(BigInteger numerator, BigInteger denominator) {
            BigInteger c = gcd(numerator, denominator);
            num[0] = numerator.divide(c);
            num[1] = denominator.divide(c);
        }

        public Ra(BigInteger numerator) {
            this(numerator, BigInteger.ONE);
        }

        /**
         * 求解最大公约数
         *
         * @param a
         * @param b
         * @return
         */
        public BigInteger gcd(BigInteger a, BigInteger b) {
            if (b.equals(BigInteger.ZERO)) {
                return a;
            }
            return gcd(b, a.remainder(b));
        }

        /**
         * 四则运算
         */
        public Ra add(Ra b) {
            BigInteger numerator = this.num[0].multiply(b.num[1]).add(
                    this.num[1].multiply(b.num[0]));
            BigInteger denominator = this.num[1].multiply(b.num[1]);
            return new Ra(numerator, denominator);
        }

        public Ra subtract(Ra b) {
            return add(negate(b));
        }

        public Ra multiply(Ra b) {
            return new Ra(this.num[0].multiply(b.num[0]),
                    this.num[1].multiply(b.num[1]));
        }

        public Ra divide(Ra b) throws Exception {
            if (b.equals(BigInteger.ZERO)) {
                throw new Exception("分母为0");
```

```
            }
            return new Ra(this.num[0].multiply(b.num[1]),
                    this.num[1].multiply(b.num[0]));
        }

        /**
         * 其他运算
         */
        public Ra negate(Ra b) {
            b.num[0] = b.num[0].negate();
            return b;
        }

        public boolean equals(Ra b) {
            if (this.subtract(b).num[0].equals(BigInteger.ZERO)) {
                return true;
            }
            return false;
        }

        public String toString() {
            String str = "";
            if (this.num[1].equals(BigInteger.ONE)) {
                str = num[0] + "";
            } else {
                str = num[0] + "/" + num[1];
            }
            return str;

        }
    }
}
```

3.4 筛选法求解素数

第 1 个素数是 2，第 2 个素数是 3，... 求第 100002（十万零二）个素数

其基本思路是数字排成串，从首位向后面扫描，将所有能够约束的给删除。另外 *java* 中 *linklist* 的 *add* 操作更快，*arraylist* 的 *get* 操作更快

```
package problems.base.pratice;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/**
 * 寻找素数，筛选法
 * @author zxr
 *
 */
public class SuShuFind {
    public static void main(String[] args) {
        List<Integer> origin = new LinkedList<>();
        for(int i = 2;i<=1000*1000;i++) {
            origin.add(i);
        }
        int target = 10002;
        int k = 0; //筛选指针
        while(k != target -1) {
            Iterator<Integer> iter = origin.iterator();
            int base = origin.get(k);
```

```
            iter.next();
            while(iter.hasNext()) {
                if(iter.next() % base == 0) {
                    iter.remove();
                }
            }
            k++;
        }
        System.out.println(origin.get(k));
    }
}
```

3.5 不定方程的求解

方法一：暴力求解
通过暴力求解，答题上有几个变量就会有几层循环，但是通常可以通过一定的变形将问题的循环层数减少，例如求解

$$ax + by = c$$

可以通过两层循环进行试探，也可以变形为

$$x = (c - by)/a$$

循环变为一层，即循环 y，当上式子右边能够整除 a 时，得到结果。
方法二：拓展欧几里得
拓展欧几里得定理：

$$ax_1 + by_1 = gcd(a, b)$$
$$bx_2 + (a\%b)y - 2 = gcd(b, a\%b)$$
$$\text{结束条件}, a\%b = 0$$

所以可以设计递归进行求解。

```
package algorithm.math;

/**
 * 扩展欧几里得
 * @author zxr
 * 97x+127y=1
 */
public class ExpandOJLiDe {
    public static void main(String[] args) {
        int[] xy = new int[2];
        int a = 30;
        int b = 40;
        e_gcd(a,b,xy);
    }
    static int gcd(int a,int b) {
        if(b == 0) {
            return a;
        }
        return gcd(b,a%b);
    }
    /**
     * @param a
```

```
     * @param b
     * @param xy
     * @return
     */
    static int e_gcd(int a,int b,int[] xy) {
        if(b == 0) {
            xy[0] = 1;
            xy[1]= 0;
            return a;
        }
        int res = e_gcd(b,a%b,xy);
        int t = xy[0];
        xy[0] = xy[1];
        xy[1] = t - a/b*xy[1];
        return res;
    }

}
```

4 博弈论

博弈论，常有两种解法:

1. 固定套路，采用暴力解
2. 尼姆堆定理

4.1 暴力解法

4.1.1 伪代码如下:

```
/**
 *暴力破解-博弈问题的伪代码--只有胜负两种情况
 *
 */
 f(局面x) --> 赢or输
 边界条件
 for(所有可能的走法)
    x --> y
    t = f(y)
    if(t == 输) return 赢; 对方输，我方赢
    //回溯，恢复局面
//所有局面都完成，我方均无法赢，则:
return 输;

/**
 * 暴力破解-博弈问题的伪代码--有胜负平三种情况
 */
f(局面x) -- > 0 1 2 胜负平
tag = 负
for(所有局面){
    试走 --> 局面y
    结果t = f(y)
    if(t == 负) return 赢;
    if(t == 平) tag = 平;
    //回溯
}
return tag;
```

4.1.2 取球问题

今盒里有 n 个小球，A、B 两人轮流从盒中取球。每个人都可以看到另一个人取了多少个，也可以看到盒中还剩下多少个。两人都很聪明，不会做出错误的判断。

每个人从盒子中取出的球的数目必须是：1，3，7 或者 8 个。轮到某一方取球时不能弃权！A 先取球，然后双方交替取球，直到取完。

被迫拿到最后一个球的一方为负方（输方）

编程确定出在双方都不判断失误的情况下，对于特定的初始球数，A 是否能赢?

```
package algorithm.boyi;

/**
 * 博弈论的暴力枚举
 * 今盒里有n个小球，A、B两人轮流从盒中取球。 每个人都可以看到另一个人取了多少个，也可
      以看到盒中还剩下多少个。
 * 两人都很聪明，不会做出错误的判断。
 * 每个人从盒子中取出的球的数目必须是：1，3，7或者8个。 轮到某一方取球时不能弃权！ A
      先取球，然后双方交替取球，直到取完。
 * 被迫拿到最后一个球的一方为负方（输方）
 * 编程确定出在双方都不判断失误的情况下，对于特定的初始球数，A是否能赢？
 *
 * @author zxr
 *
 */
public class TakeBall {
   static int[] type = { 1, 3, 7, 8 };

   public static void main(String[] args) {
      for (int i = 1; i <= 50; i++) {
         System.out.println(f(i));
      }
   }

   public static boolean f(int n) {
      if (n == 0) {
         return true;
      }
      for (int i = 0; i < type.length; i++) {
         if (n >= type[i]) {
            boolean y = f(n - type[i]);
            if (!y) {
               return true;
            }
         }
      }
      return false;
   }
}
```

4.1.3 尼姆堆

有 3 堆硬币，分别是 3,4,5

二人轮流取硬币。

每人每次只能从某一堆上取任意数量。

不能弃权。

取到最后一枚硬币的为赢家。

求先取硬币一方有无必胜的招法。

尼姆定理：将所有堆的数量进行异或操作，如果得到的结果非全零（非平衡状态）则先手赢，得到的是全零则先手输

所有无偏的二人游戏都可以用尼姆定理进行求解，关键在于如何将问题转化为尼姆堆

```
package algorithm.math;
/**
 * 有3堆硬币，分别是3,4,5
二人轮流取硬币。
每人每次只能从某一堆上取任意数量。
不能弃权。
取到最后一枚硬币的为赢家。

求先取硬币一方有无必胜的招法。
 * @author zxr
 * 运用了模 2 加的方法,如果所有堆数量的模 2 加后等于 0 ，则当前人必输，否则可以赢。
 */
public class NiMuDui {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] d = { 3,4,5 };
        f(d);
    }

    static void f(int[] a) {
        int sum = 0;
        for (int i : a) {
            sum ^= i;
        }
        if (sum == 0) {
            System.out.println("输");
            return;
        }
        for (int i : a) {
            int x = sum ^ i;
            if (x < i) {
                System.out.println(i + "-->" + x);
            }
        }

    }

}
```

5 随机算法

如果题目要求的是一种可行解，这种方法非常适合，较优解这种方式也可用。推荐使用**蒙特卡洛和模拟退火**，稍微好实现一些。

5.1 蒙特卡洛算法

题目：给定 4 个数，看它是否能通过 + - * / 四种运算符得到 24 点。
这个题有两个值得学习的地方：

1. 逆波兰表达式计算

2. 抛异常跳出大循环，原来一直在想 exit() 函数的问题，可以用这种方式来实现。

3. 使用 try-catch-finally，语句，即使在 try 或者 catch 语句中执行了 return，finally 语句也会被执行。

```
package algorithm.random;

import java.util.Arrays;
import java.util.EmptyStackException;
import java.util.Random;
import java.util.Stack;

/**
 * 24点计算,给定4个数字求解是否有可能经过四则运算得到24点
 *
 * @author zxr
 *
 */
public class CalCu24 {
    public static void main(String[] args) {
        int[] origin = new int[] {3,4,5,6};
        f(origin);
    }

    /**
     * 试着计算
     */
    static Random r = new Random();

    public static void f(int[] origin) {
        int[] buff = new int[7];
        for (int i = 0; i < 4; i++) {
            buff[i] = origin[i];
        }
        int N1 = 10000 * 1000; // 实验次数
        int N2 = 5; // 每种运算符号使用多少次
        for (int i = 0; i < N1; i++) {
            randomSwap(buff);
            generateOpr(buff);
            for (int j = 0; j < N2; j++) {
                swapOpr(buff);
                // 计算开始
                try {
                    calculate(buff);
                } catch (EmptyStackException e) {
                    continue;
                } catch (Exception e) {
                    return;
                }
            }
        }
        System.out.println("don't␣find␣a␣answer");
    }

    private static void calculate(int[] buff) throws EmptyStackException,
         Exception {
        Stack<Integer> s = new Stack<Integer>();
        for (int i = 0; i < 4; i++) {
            s.push(buff[i]);
        }
```

```
        for (int i = 4; i < 7; i++) {
            switch ((int) buff[i]) {
            case 0:
                // +
                s.push(s.pop() + s.pop());
                break;
            case 1:
                // -
                s.push(s.pop() - s.pop());
                break;
            case 2:
                // *
                s.push(s.pop() * s.pop());
                break;
            case 3:
                // /
                int m1 = s.pop();
                int m2 = s.pop();
                if (m2 == 0 || m1 % m2 != 0) {
                    return;
                }
                s.push(s.pop() / s.pop());
                break;
            default:
                break;
            }
        }
        if (s.pop() == 24) {
            show(buff);
            throw new Exception("find a answeer");
        }
    }

    private static void show(int[] buff) {
        Stack<String> s = new Stack<>();
        for (int i = 0; i < 4; i++) {
            s.push(buff[i]+"");
        }

        for (int i = 4; i < 7; i++) {
            switch ((int) buff[i]) {
            case 0:
                // +
                s.push("("+s.pop() +"+"+ s.pop()+")");
                break;
            case 1:
                // -
                s.push("("+s.pop() +"-" + s.pop()+")");
                break;
            case 2:
                // *
                s.push("("+s.pop()+ "*"+ s.pop()+")");
                break;
            case 3:
                // /
                s.push("("+s.pop() +"/" +s.pop()+")");
                break;
            default:
                break;
            }
        }
        System.out.println(s.pop());
```

```
    }

    private static void generateOpr(int[] buff) {
        for (int i = 4; i < 7; i++) {
            buff[i] = r.nextInt(4);
        }
    }

    public static void randomSwap(int[] array) {
        int i1 = r.nextInt(4);
        int i2 = r.nextInt(4);
        int temp = array[i1];
        array[i1] = array[i2];
        array[i2] = temp;
    }

    public static void swapOpr(int[] array) {
        int i1 = r.nextInt(3) + 4;
        int i2 = r.nextInt(3) + 4;
        int temp = array[i1];
        array[i1] = array[i2];
        array[i2] = temp;
    }
}
```

5.2 模拟退火

5.2.1 伪代码

```
/*
* J(y): 在状态y时的评价函数值
* Y(i): 表示当前状态
* Y(i+1): 表示新的状态
* r:  用于控制降温的快慢
* T:  系统的温度，系统初始应该要处于一个高温的状态
* T_min ：温度的下限，若温度T达到T_min，则停止搜索
*/
while( T > T_min )
{
    dE = J( Y(i+1) ) - J( Y(i) ) ;

    if ( dE >=0 ) //表达移动后得到更优解，则总是接受移动
      Y(i+1) = Y(i) ; //接受从Y(i)到Y(i+1)的移动
    else
    {
      // 函数exp( dE/T )的取值范围是(0,1) ，dE/T越大，则exp( dE/T )也
      if ( exp( -dE/T ) > random( 0 , 1 ) )
         Y(i+1) = Y(i) ; //接受从Y(i)到Y(i+1)的移动
    }
     T = r * T ; //降温退火 ，0<r<1 。r越大，降温越慢；r越小，降温越快
    /*
    * 若r过大，则搜索到全局最优解的可能会较高，但搜索的过程也就较长。若r过小，则搜索的
     过程会很快，但最终可能会达到一个局部最优值
    */
    i ++ ;
}
```

5.2.2 求解 $\sin x$ $(-\pi, \pi)$ 之间的最小值

```java
package algorithm;

/**
 * 模拟退火
 *
 * @author zxr
 *
 */
public class SA {
    static double pi = Math.PI;

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // 初始参数
        double T0 = 100;
        double t = T0;
        double Te = 3;
        double a = 0.99;
        double x0 = rand();
        double Ecurrent = target(x0);
        double Ebest = Ecurrent;
        int markvon = 100;
        while (t > Te) {
            for (int i = 0; i < markvon; i++) {
                x0 = rand();
                double Enew = target(x0);
                if (Enew < Ecurrent) {
                    Ecurrent = Enew;
                }else if(Math.exp(-(Enew-Ecurrent)/t) > Math.random()){
                    Ecurrent = Enew;
                }
                if(Ecurrent < Ebest) {
                    Ebest = Ecurrent;
                }
            }
            t *= a;
        }
        System.out.println(Ebest);
    }

    static double rand() {
        return -pi + (Math.random()) * (2 * pi);
    }

    static double target(double x) {
        return Math.sin(x);
    }

}
```

6 动态规划

垒骰子

赌圣 atm 晚年迷恋上了垒骰子，就是把骰子一个垒在另一个上边，不能歪歪扭扭，要垒成方柱体。经过长期观察，atm 发现了稳定骰子的奥秘：有些数字的面贴着会互相排斥！我们先来规范一下骰子：1 的对面是 4，2 的对面是 5，3 的对面是 6。假设有 m 组互斥现象，每组中的那两个数字的面紧贴在一起，骰子就不能稳定的垒起来。atm 想计算一下有多少种不同的可能的垒骰子方式。

两种垒骰子方式相同，当且仅当这两种方式中对应高度的骰子的对应数字的朝向都相同。由于方案数可能过多，请输出模 10^9+7 的结果。

不要小看了 atm 的骰子数量哦 ~

「输入格式」第一行两个整数 n m n 表示骰子数目接下来 m 行，每行两个整数 a b ，表示 a 和 b 不能紧贴在一起。

「输出格式」一行一个数，表示答案模 10^9+7 的结果。

「样例输入」2 1 1 2

「样例输出」544

本题注意两点：

- 数据量大，暴力解肯定超过范围，所以使用 DP。
- 大规模余数，将余数因子写成变量，且在用科学表达式时要先进行强制转换并赋值给变量。不要直接使用。

```
package 国赛恢复训练.省赛2015;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
public class DP1 {

    /**
     * 暴力破解显然不够，选择DP求解 状态D(i,j) 表示第i层地面为j时可选择的方案数
     *
     * @author 张兴锐
     *
     */
    static long count = 0;
    static int[] duimian = { 0, 4, 5, 6, 1, 2, 3 };
    static Map<Integer, List<Integer>> paichi = new HashMap<Integer, List<Integer
        >>();
    static int n;
    static int[][] D;
    static int rem = (int) (1E9 + 7);// java科学计数法生成的是double。求余操作把余
        数写出来

    public static void main(String[] args) {
        // 初始化
        Scanner input = new Scanner(System.in);
        n = input.nextInt();
        int m = input.nextInt();
        D = new int[n + 1][7];// 免得换算，空间一般都是够的。
        for (int i = 0; i < m; i++) {
            int m1 = input.nextInt();
            int m2 = input.nextInt();
            if (paichi.containsKey(m1)) {
                paichi.get(m1).add(m2);
            } else {
                List<Integer> list = new ArrayList<Integer>();
                list.add(m2);
                paichi.put(m1, list);
            }
            if (paichi.containsKey(m2)) {
                paichi.get(m2).add(m1);

            } else {
                List<Integer> list = new ArrayList<Integer>();
                list.add(m1);
```

```
                paichi.put(m2, list);
            }
        }
        input.close();
        // DP
        // 初始化
        for (int i = 1; i <= 6; i++) {
            D[1][i] = 1;
        }
        for (int i = 2; i <= n; i++) {
            for (int j = 1; j <= 6; j++) {
                List<Integer> list = init();
                // 得到禁忌表
                List<Integer> forbid = paichi.get(j);
                int sum = 0;
                if (forbid == null) {
                    for (int k = 1; k <= 6; k++) {
                        sum = add(sum, D[i - 1][k]);
                    }
                } else {
                    list.removeAll(forbid);
                    for (Integer v : list) {
                        sum = add(sum, D[i - 1][duimian[v]]);
                    }
                }
                D[i][j] = sum;
            }
        }
        int sum = 0;
        for (int i = 1; i <= 6; i++) {
            sum = add(sum, D[n][i]);
        }
        System.out.println((sum * Math.pow(4, n) % (rem)));
    }

    public static List<Integer> init() {
        List<Integer> list = new ArrayList<Integer>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        list.add(6);
        return list;
    }

    public static int add(int base, int added) {
        return (base % rem + added % rem) % rem;
    }
}
```

7 其他技巧

7.1 树

7.1.1 二叉树

1. 二叉排序树：通过二分进行建树，中序遍历即可
2. 哈弗曼编码树：最小无重复前缀编码方式

3. 区间树:

4. 线段树：经常对某个区间的某个值进行修改，对某个区间进行查询。

7.1.2 哈弗曼编码树

```
package 国赛恢复训练.省赛2015;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.List;

/**
 * 哈夫曼编码
 * @author 张兴锐
 *
 */
public class Problem18 {
    static final String NOCODE = "no";
    static List<Node> nodes = new ArrayList<Node>(Arrays.asList(
                new Node(1),
                new Node(10),
                new Node(103),
                new Node(5),
                new Node(51),
                new Node(103),
                new Node(14)
            ));
    public static void main(String[] args) {
        Collections.sort(nodes);
        while(nodes.size() != 1) {
            Node m1 = nodes.get(0);
            Node m2 = nodes.get(1);
            Node m = new Node(m1.value+m2.value,m1,m2,NOCODE);
            nodes.remove(0);
            nodes.remove(0);
            insert(m);
        }
        Node root = nodes.get(0);
        code(root,"");
        preOrder(root);
    }
    public static void code(Node n,String c) {
        if(n == null) {
            return;
        }
        if(n.code == null) {
            n.code = c;
        }
        code(n.left,c+"0");
        code(n.right,c+"1");
    }
    public static void preOrder(Node n) {
        if(n == null) {
            return;
        }
        if(!n.code.equals(NOCODE)) {
            System.out.println(n.value+"␣"+n.code);
        }
```

```
        preOrder(n.left);
        preOrder(n.right);
    }
    public static void insert(Node m) {
        for(int i = 0;i<nodes.size();i++) {
            if(m.value <= nodes.get(i).value) {
                nodes.add(i,m);
                return;
            }
        }
        nodes.add(m);
    }
    static class Node implements Comparable<Node>{
        int value;
        Node left;
        Node right;
        String code;
        public Node(int value, Node left, Node right,String code) {
            super();
            this.value = value;
            this.left = left;
            this.right = right;
            this.code = code;
        }
        public Node(int value) {
            this.value = value;
        }
        public int compareTo(Node arg0) {
            return this.value - arg0.value;
        }

    }
}
```

7.1.3 线段树

基本操作：

- 修改区间值
- 查询区间特征值（视情况而定）

注意事项：树的申请空间为原数据的 4 倍

```
package algorithm.boyilun;
/**
 * 线段树
 * @author 张兴锐
 *
 */
public class SegmentTree {
    static int[] a = new int[]{-1,1,4,1,5,13,2,51,4};
    static Tree[] tree = new Tree[4*(a.length-1)];//0位不使用
    public static void main(String[] args) {
        build(1,1,a.length-1);
        System.out.println(query_sum(1,4,a.length-1));
        System.out.println(query_max(1,4,a.length-1));
    }
    public static void build(int c,int l,int r){
        tree[c] = new Tree();
        tree[c].begin = l;
```

```
        tree[c].end = r;
        if(l == r){
            tree[c].sum = a[l];
            tree[c].max = a[l];
            return;
        }
        int mid = (tree[c].begin + tree[c].end)/2;
        build(c<<1,l,mid);
        build(c<<1|1,mid+1,r);
        //刷新
        push_up(c);
    }
    public static void push_up(int c){
        tree[c].sum = tree[c<<1].sum + tree[c<<1|1].sum;
        tree[c].max = Math.max(tree[c<<1].max, tree[c<<1|1].max);
    }
    public static void update(int c,int x,int val){
        if(tree[c].begin == tree[c].end){
            tree[c].sum = val;
            tree[c].max = val;
            return;
        }
        //否则
        int mid = (tree[c].begin+tree[c].end)/2;
        if(x <= mid){
            update(c<<1,x,val);
        }else if(x > mid){
            update(c<<1|1,x,val);
        }
        push_up(c);
    }
    public static int query_sum(int c,int l,int r){
        if(tree[c].begin == l && tree[c].end == r){
            return tree[c].sum;
        }
        int mid = (tree[c].begin+tree[c].end)/2;
        if(l > mid){
            return query_sum(c<<1|1,l,r);
        }
        if(r<= mid){
            return query_sum(c<<1,l,r);
        }
        return query_sum(c<<1,l,mid)+query_sum(c<<1|1,mid+1,r);
    }
    public static int query_max(int c,int l,int r){
        if(tree[c].begin == l && tree[c].end == r){
            return tree[c].max;
        }
        int mid = (tree[c].begin + tree[c].end)/2;
        if( l > mid){
            return query_max(c<<1|1,l,r);
        }
        if( r <= mid){
            return query_max(c<<1,l,r);
        }
        return Math.max(query_max(c<<1,l,mid), query_max(c<<1|1,mid+1,r));
    }
    static class Tree{
        int begin;
        int end;
        int sum;
        int max;
```

```
    }
}
```

7.1.4 树 dp

没有上司的舞会。**可以学到:**

1. 如何建树。
2. 树 dp。从下往上更新。

```
package problems;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

/**
 *
 * @author zxr
 *
 */
public class NoShangSi {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner input = new Scanner(System.in);
        int num = input.nextInt();
        int[][] a = new int[num][3];// 第1列为父亲，第2列为儿子,第3列为儿子节点的权
            至
        for (int i = 0; i < num; i++) {
            a[i][0] = input.nextInt();
            a[i][1] = input.nextInt();
            a[i][2] = input.nextInt();
        }
        int root_val = input.nextInt();
        input.close();
        // 建树
        build(a);
        // 尝试遍历
        Node root = findRoot();
        root.value = root_val;
        visted(root);
        int[][] d = new int[count + 1][2];
        dfs(d, root);
        System.out.println(Math.max(d[root.index][0], d[root.index][1]));
    }

    static void dfs(int[][] d, Node node) {
        int index = node.index;
        d[index][0] = 0;
        d[index][1] = node.value;
        for (Node son : node.sons) {
            dfs(d, son);
            d[index][0] += Math.max(d[son.index][0], d[son.index][1]);
            d[index][1] = d[son.index][0] + node.value;
            // d[index][1] > d[son.index][0]+node.value?d[index][1]:
        }
```

```
}

static int count;

static void visted(Node node) {
    if (node == null) {
        return;
    }
    count++;
    System.out.println(node.index + "-->" + node.value);
    for (Node son : node.sons) {
        visted(son);
    }
}

static Set<Integer> check_con = new HashSet<>();// 检查是否已经添加过节点
static List<Node> list = new ArrayList<>();

static Node findNode(int index) {
    for (Node node : list) {
        if (node.index == index) {
            return node;
        }
    }
    return null;
}

static Node findRoot() {
    Node current = list.get(0);
    while (current.parent != null) {
        current = current.parent;
    }
    return current;
}

static void build(int[][] input) {
    for (int[] is : input) {
        int father = is[0];
        int son = is[1];
        int value = is[2];
        Node f;
        if (!check_con.contains(father)) {
            check_con.add(father);
            f = new Node(father);
            list.add(f);
        } else {
            f = findNode(father);
        }
        check_con.add(son);
        Node s = new Node(son);// 儿子只能有一个父亲节点
        list.add(s);
        s.parent = f;
        s.value = value;
        f.sons.add(s);
    }
}

static class Node {
    int index;
    int value;
    List<Node> sons = new ArrayList<>();
    Node parent;
```

```
        public Node(int index) {
            super();
            this.index = index;
        }

    }
}
```

7.2 正则表达式

7.2.1 关键词

符号	含义
$\backslash b$	匹配一个字的边界
$\backslash B$	非字符边界
$\backslash d$	数字
$\backslash D$	非数字
$\backslash s$	空白字符
$\backslash S$	非空白字符
$\backslash w$	单词字符
$\backslash W$	非单词字符

7.2.2 去叠词

```
    public class Test {
    public static void main(String[] args) {
        String test = "22223131413718623872193132121111";
        System.out.println(test.replaceAll("(.)\\1{2,}", "$1"));
        System.out.println(test.replaceAll("(.)\\1+", "$1"));
    }
    }
```