目录

第二次作业报告

姓名：汪忠毅

班级：软件2101

U265022 006. 和的平方与平方的和之间的差值

题目链接：U265022 006. 和的平方与平方的和之间的差值 - 洛谷

方法一

时间复杂度：$O(n)$

空间复杂度：$O(1)$

思路

从 $1 \sim n$ 遍历计算即可.

代码

```java
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265022">U265022 006. 和的平方与平方的和之间的差值 - 洛谷</a>
 * @date 2023-10-22 19:18:30
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        long ans1 = 1, ans2 = 1;
        for (int i = 2; i ⩽ n; ++i) {
            ans1 += (long) i * i;
            ans2 += i;
        }
```

```
    }
}
```

方法二

时间复杂度：$O(1)$

空间复杂度：$O(1)$

思路

用数学公式：

- 自然数的平方的和：$\sum_{k=1}^{n} = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$
- 等差数列前 n 项和：$=1 + 2 + 3 + \cdots + n = \frac{(1+n)n}{2}$

代码

```java
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265022">U265022 006. 和的平方与平方的和之间的差值 - 洛谷</a>
 * @date 2023-10-22 19:18:30
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        long ans1 = n * (n + 1) * (2L * n + 1) / 6, ans2 = (long) (n + 1) * n / 2;
        System.out.println(ans2 * ans2 - ans1);
    }
}
```

U265129 008. 序列中的最大乘积

题目链接：U265129 008. 序列中的最大乘积 - 洛谷

分析一下（数据范围），判断是否需要用 BigInteger .连续 n 个数的最大乘积，已知 $2 \le len \le 5000, 2 \le n \le min(17, len)$ ，即 $n_{max} = 17$，17 位的整数最大为 9^{17}，而 long 的最大值是 $2^{63} > 9^{17}$， 所以结果 不会超出 long 的表示范围.

因此，计算过程选择 long 类型来存储结果.

方法一

时间复杂度：$O((len - n) * n)$

空间复杂度：$O(1)$

思路

遍历第 $1 \sim len - n + 1$ 位上的数，每次连续 $*13$ 个数，结果比大小求最大即可.

代码

```java
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265129">U265129 008. 序列中的最大乘积 - 洛谷</a>
 * @date 2023-10-22 20:15:07
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int len = scan.nextInt();
        scan.nextLine(); // 若不写这一行，s 所读入的长度就会为 0，结果会显示运行错误 RE
        String s = scan.nextLine();
        int n = scan.nextInt();
        long ans = 0;
        for (int i = 0; i ⩽ len - n; ++i) {
            long cur = 1;
            for (int j = 0; j < n; ++j) {
                cur *= s.charAt(i + j) - '0';
            }
            ans = Math.max(ans, cur);
        }
        System.out.println(ans);
    }
}
```

方法二

时间复杂度：$O(len)$

空间复杂度：$O(1)$

思路

双指针优化

- 在上述计算过程中，我们发现除第一个数外，每个数都计算了多次，那是不是可以选择将 中间数的计算结果 作为一个变量存储下来，每次计算连续的 13 个数的乘积只需要 除去最前面的一个，乘上第 13 个数 就行了.
- 那么有什么要注意的点呢？我们需要除去最前面的一个数，那我们怎么知道最前面的那个数是多少呢？所以还需要记录最前面那个数的值；同时，计算过程中用到了除法和乘法，如果除 0，会发生计算错误，如果乘 0，再继续运行对当前乘积都不会有影响，所以我们需要 特别判断 0 的情况，跳过对 0 的计算.

代码

```java
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265129">U265129 008. 序列中的最大乘积 - 洛谷</a>
 * @date 2023-10-22 20:15:07
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int len = scan.nextInt();
        scan.nextLine(); // 若不写这一行，s 所读入的长度就会为 0，结果会显示运行错误 RE
        String s = scan.nextLine();
        int n = scan.nextInt();
        long ans = 0;
        long curMul = 1;
        for (int i = 0, j = 0; i < len; ++i) {
            if (s.charAt(i) == '0') {
                while (i < len && s.charAt(i) == '0') ++i;
                if (i == len) break;
                j = i;
                curMul = s.charAt(i) - '0';
            } else {
                curMul *= s.charAt(i) - '0';
                if (i - j + 1 == n) {
                    ans = Math.max(ans, curMul);
                    curMul /= s.charAt(j) - '0';
                    ++j;
                }
            }
        }
        System.out.println(ans);
    }
}
```

U265594 015. 格子路径

题目链接：U265594 015. 格子路径 - 洛谷

方法一

时间复杂度：$O((n+m)^2)$

空间复杂度：$O(1)$

思路

递推公式组合数

总共要走 $n+m$ 步，选出 n 步向下走，选出 m 步向右走，且组合数 $C(n+m,n)=C(n+m,m)$，即：

$$C_a^b=C_{a-1}^{b-1}+C_{a-1}^b \tag{1}$$

代码

```java
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265594">U265594 015. 格子路径 - 洛谷</a>
 * @date 2023-10-22 21:09:13
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt(), m = scan.nextInt(), all = n + m;
        long[][] combination = new long[all + 1][all + 1];
        for (int i = 0; i ⩽ all; ++i) {
            combination[i][0] = combination[i][i] = 1;
            for (int j = 1; j ⩽ i; ++j) {
                combination[i][j] = combination[i - 1][j - 1] + combination[i - 1][j];
            }
        }
        System.out.println(combination[all][n]);
    }
}
```

方法二

时间复杂度：$O(n+m)$

空间复杂度：$O(1)$

思路

阶乘求组合数

数字太大，应使用 `BigInteger` ：

$$C_a^b = \frac{a!}{b!\times(a-b)!} \tag{2}$$

代码

```java
import java.math.BigInteger;
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265594">U265594 015. 格子路径 - 洛谷</a>
 * @date 2023-10-22 21:09:13
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
```

```
        int n = scan.nextInt(), m = scan.nextInt(), all = n + m;
        BigInteger num = new BigInteger("1");
        for (int i = 2; i ⩽ all; i++) num = num.multiply(BigInteger.valueOf(i));
        for (int i = 2; i ⩽ n; i++) num = num.divide(BigInteger.valueOf(i));
        for (int i = 2; i ⩽ m; i++) num = num.divide(BigInteger.valueOf(i));
        System.out.println(num);
    }
}
```

方法三

时间复杂度：$O(nm+n+m)$

空间复杂度：$O((n+1)*(m+1))$

思路

动态规划

因为只能向右或向下移动，所以到达点 (x,y) 的路径数等于到达 $(x-1,y)$ 的路径数 + 到达 $(x,y-1)$ 的路径数，边界条件是，一旦 x 或者 y 的坐标为 0，则到达该点的路径只有 1 条.

因此，本题动态规划的 状态转移方程：

$$p(x,y)=\begin{cases}p(x-1,y)+p(x,y-1) & x,y>0\\ 1 & x=0或y=0\end{cases} \tag{3}$$

代码

```
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U265594">U265594 015. 格子路径 - 洛谷
</a>
 * @date 2023-10-22 21:09:13
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt(), m = scan.nextInt();
        long[][] dp = new long[n + 1][m + 1];
        for (int i = 0; i ⩽ n; i++) dp[i][0] = 1;
        for (int i = 0; i ⩽ m; i++) dp[0][i] = 1;
        for (int i = 1; i ⩽ n; i++) {
            for (int j = 1; j ⩽ m; j++) {
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
            }
        }
        System.out.println(dp[n][m]);
    }
}
```

U273769 020. 阶乘各位数的和

题目链接：U273769 020. 阶乘各位数的和 - 洛谷

方法一

时间复杂度：$O(n+\log n!)$

空间复杂度：$O(\log n!)$

思路

先直接求阶乘 $n!$（很大，要使用 BigInteger ），然后转换成字符串遍历每一位求和即可.

代码

```java
import java.math.BigInteger;
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U273769">U273769 020. 阶乘各位数的和 - 洛谷</a>
 * @date 2023-10-23 17:29:59
 */
public class Main {
    /**
     * 求 n!
     * @param n 正整数
     * @return n!
     */
    public static BigInteger factorial(int n) {
        BigInteger prod = BigInteger.ONE;
        for (int i = 2; i ⩽ n; i++)
            prod = prod.multiply(BigInteger.valueOf(i));
        return prod;
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        String result = factorial(n).toString();
        int sum = 0;
        for (int i = 0; i < result.length(); i++)
            sum += result.charAt(i) - '0';
        System.out.println(sum);
    }
}
```

U273777 024. 字典排列

题目链接：U273777 024. 字典排列 - 洛谷

方法一

时间复杂度：$O(10!)$

空间复杂度：$O(T)$

思路

深度优先搜索算法

- （Depth First Search，简称 DFS）：一种用于 遍历或搜索树或图 的算法.沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当 节点 v 的所在边都已被探寻过或者在搜寻时结点不满足条件，搜索将回溯到发现节点 v 的那条边的起始节点.整个进程反复进行直到所有节点都被访问为止.
- 【参考博客】：DFS入门级（模板）_dfs模型-CSDN博客

代码

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

/**
 * @author Re_Gin
 * @link <a href="https://www.luogu.com.cn/problem/U273777">U273777 024. 字典排列 - 洛谷
</a>
 * @date 2023-10-23 18:37:43
 */
public class Main {
    private static final int nummax = 10; // 0~9 的个数
    private static int count_n = 0; // 第 count_n 个排列数
    private static int[] nums = new int[nummax]; // 一个排列
    private static boolean[] visited = new boolean[nummax];  // 数字是否被选过

    public static void dfs(Map<Integer, String> map, int num) {
        /* 排列数达到10位 */
        if (num == nummax) {
            ++count_n;
            if (map.containsKey(count_n)) {
                StringBuilder str = new StringBuilder();
                for (int i = 0; i < nummax; ++i) {
                    str.append(nums[i]);
                }
                map.put(count_n, String.valueOf(str));
            }
            return;
        }
        for (int i = 0; i < nummax; ++i) {
            if (visited[i]) continue;
            visited[i] = true;
            nums[num] = i;
            dfs(map, num + 1);
```

```
            visited[i] = false;
        }
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int T = scan.nextInt(), i;
        int[] n = new int[T];
        Map<Integer, String> map = new HashMap<>(); // (n[i],排列)
        for (i = 0; i < T; ++i) {
            n[i] = scan.nextInt();
            map.put(n[i], "WustJavaClub"); // 默认不存在第 n[i] 个数
        }
        dfs(map, 0);
        for (int v : n) {
            System.out.println(map.get(v));
        }
    }
}
```

注:

- 简单总结一下 DFS 算法的模板:

```
function dfs(当前状态) {
    if(当前状态 == 目的状态) {
        ...
    }
    for(...寻找新状态) {
        if(状态合法) {
            vis[访问该点];
            dfs(新状态);
            ?是否需要恢复现场→vis[恢复访问]
        }
    }
    if(找不到新状态) {
        ...
    }
}
```

- 【扩展资料】: Next lexicographical permutation algorithm (nayuki.io)

452. 用最少数量的箭引爆气球

题目链接: 452. 用最少数量的箭引爆气球 - 力扣 (LeetCode)

方法一

时间复杂度: $O(n \log n)$ (n 为气球个数)

空间复杂度: $O(\log n)$

思路

排序 + 贪心

- 【官方思路】：452. 用最少数量的箭引爆气球 - 官方题解思路

代码

```java
class Solution {
    public int findMinArrowShots(int[][] points) {
        if (points.length == 0) return 0;
        Arrays.sort(points, Comparator.comparingInt(a -> a[1]));
        int curPos = points[0][1];
        int ret = 1;
        for (int i = 1; i < points.length; i++) {
            if (points[i][0] <= curPos) {
                continue;
            }
            curPos = points[i][1];
            ++ret;
        }
        return ret;
    }
}
```

方法二

时间复杂度：$O(n \log n)$

空间复杂度：$O(1)$

思路

快速排序

- 【快速排序详解】：快速排序（java实现）_java快速排序-CSDN博客

代码

```java
class Solution {
    public int findMinArrowShots(int[][] points) {
        quick_sort(points, 0, points.length - 1);
        int ans = 1;
        int right = points[0][1];
        for(int[] point : points){
            if(right < point[0]){
                ++ans;
                right = point[1];
            }
        }
        return ans;
    }

    private void quick_sort(int[][] q, int l, int r){
        if(l >= r) return;
        int x = q[l + r >> 1][1], i = l - 1, j = r + 1;
```

```
        while(i < j){
            while(q[++i][1] < x);
            while(q[--j][1] > x);
            if(i < j){
                int[] t = q[i];
                q[i] = q[j];
                q[j] = t;
            }
        }
        quick_sort(q, l, j);
        quick_sort(q, j + 1, r);
    }
}
```

455. 分发饼干

题目链接：455. 分发饼干 - 力扣（LeetCode）

方法一

时间复杂度：$O(m \log m + n \log n)$

空间复杂度：$O(\log m + \log n)$

思路

排序 + 双指针 + 贪心

- 【官方思路】：455. 分发饼干 - 官方题解思路

代码

```
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        Arrays.sort(g);
        Arrays.sort(s);
        int i = 0, j = 0;
        while (i < g.length && j < s.length) {
            if (g[i] <= s[j]) i++;
            j++;
        }
        return i;
    }
}
```

11. 盛最多水的容器

题目链接：11. 盛最多水的容器 - 力扣（LeetCode）

方法一

时间复杂度：$O(n)$

空间复杂度：$O(1)$

思路

- 【官方思路】：11. 盛最多水的容器 - 官方题解思路

代码

```java
class Solution {
    public int maxArea(int[] height) {
        int l = 0, r = height.length - 1, ans = 0;
        while (l < r) {
            int area = Math.min(height[l], height[r]) * (r - l);
            ans = Math.max(ans, area);
            if (height[l] ⩽ height[r]) ++l;
            else --r;
        }
        return ans;
    }
}
```

方法二

时间复杂度：$O(n)$

空间复杂度：$O(1)$

思路

剪枝

- 【剪枝详解】：剪枝 - Cattle_Horse - 博客园 (cnblogs.com)

代码

```java
class Solution {
    public int maxArea(int[] height) {
        int l = 0, r = height.length - 1;
        int maxArea = 0;
        while (l < r) {
            maxArea = Math.max(maxArea, (r - l) * Math.min(height[l], height[r]));
            if (height[l] < height[r]) {
                int curLeftHeight = height[l];
                // 剪枝
                while (height[l] ⩽ curLeftHeight && l < r) {
                    l++;
                }
            } else {
                int curRightHeight = height[r];
                // 剪枝
                while (height[r] ⩽ curRightHeight && l < r) {
```

```
                    r--;
                }
            }
        }
        return maxArea;
    }
}
```