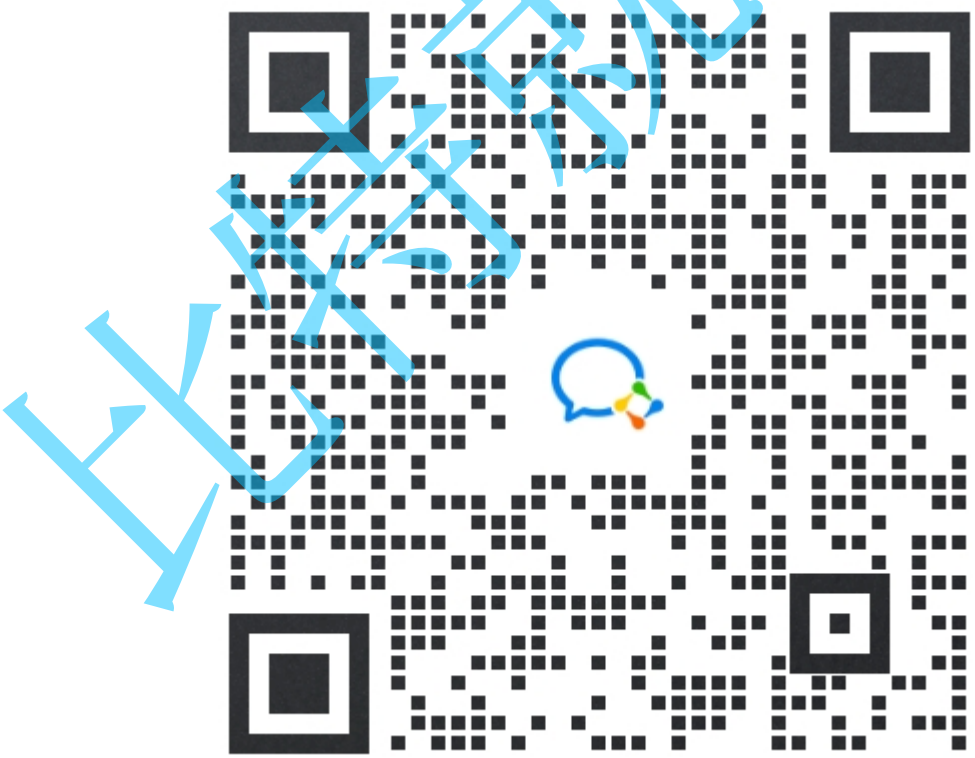
笔试强训第 05 周

版权说明

对比特算法感兴趣，可以联系这个微信。



板书链接

https://gitee.com/wu-xiaozhe/written-exam-training

Day25

1. 笨小猴（模拟）

（题号：170466）

1. **题目链接：** BC145 [NOIP2008]笨小猴

2. **题目描述：**

> 题目描述：笨小猴的词汇量很小，所以每次做英语选择题的时候都很头疼。但是他找到了一种方法，经试验证明，用这种方法去选择选项的时候选对的几率非常大！
> 这种方法的具体描述如下：假设maxn是单词中出现次数最多的字母的出现次数，minn是单词中出现次数最少的字母的出现次数，如果maxn-minn是一个质数，那么笨小猴就认为这是个Lucky Word，这样的单词很可能就是正确的答案。
>
> 输入描述：只有一行，是一个单词，其中只可能出现小写字母，并且长度小于100。
>
> 输出描述：共两行，第一行是一个字符串，假设输入的的单词是Lucky Word，那么输出“Lucky Word”，否则输出“No Answer”；
> 第二行是一个整数，如果输入单词是Lucky Word，输出maxn-minn的值，否则输出0。
>
> 补充说明：

> 示例1
> 输入：error
> 输出：Lucky Word
> 2
> 说明：单词error中出现最多的字母r出现了3次，出现次数最少的字母出现了1次，3-1=2，2是质数。
>
> 示例2
> 输入：Olympic
> 输出：No Answer
> 0
> 说明：单词olympic中出现每个字母都只出现一次，即出现次数最多的字母出现了1次，出现次数最少的字母出现了1次，1-1=0，0不是质数。

3. **解法：**

算法思路：

根据题意模拟，但是要注意细节。

C++ 算法代码：

```cpp
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

string s;

bool isprim(int n) // 判断 n 是否是质数
{
```

```
    if(n < 2) return false;
    for(int i = 2; i <= sqrt(n); i++)
    {
        if(n % i == 0) return false;
    }
    return true;
}

int main()
{
    cin >> s;
    int hash[26] = { 0 };
    for(auto ch : s)
    {
        hash[ch - 'a']++;
    }

    int minn = 1000, maxn = 0;
    for(int i = 0; i < 26; i++)
    {
        if(hash[i])
        {
            minn = min(minn, hash[i]);
            maxn = max(maxn, hash[i]);
        }
    }

    if(isprim(maxn - minn))
    {
        cout << "Lucky Word" << endl;
        cout << maxn - minn << endl;
    }
    else
    {
        cout << "No Answer" << endl;
        cout << 0 << endl;
    }

    return 0;
}
```


Java 算法代码：

```
import java.util.Scanner;
```

```java

public class Main
{
    public static boolean isprim(int n) // 判断 n 是否是质数
    {
        if(n < 2) return false;
        for(int i = 2; i <= Math.sqrt(n); i++)
        {
            if(n % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        char[] s = in.next().toCharArray();

        int[] hash = new int[26];
        for(int i = 0; i < s.length; i++)
        {
            hash[s[i] - 'a']++;
        }

        int minn = 1000, maxn = 0;
        for(int i = 0; i < 26; i++)
        {
            if(hash[i] != 0)
            {
                minn = Math.min(minn, hash[i]);
                maxn = Math.max(maxn, hash[i]);
            }
        }

        if(isprim(maxn - minn))
        {
            System.out.println("Lucky Word");
            System.out.println(maxn - minn);
        }
        else
        {
            System.out.println("No Answer");
            System.out.println(0);
        }
    }
}
```

2. 主持人调度（一）（排序）

（题号：2427095）

1. **题目链接：** NC383 主持人调度（一）

2. **题目描述：**

> 题目描述：有 n 个活动即将举办，每个活动都有开始时间与活动的结束时间，第 i 个活动的开始时间是 $start_i$,第 i 个活动的结束时间是 end_i ,举办某个活动就需要为该活动准备一个活动主持人。
>
> 一位活动主持人在同一时间只能参与一个活动。并且活动主持人需要全程参与活动，换句话说，一个主持人参与了第 i 个活动，那么该主持人在 $(start_i,end_i)$ 这个时间段不能参与其他任何活动。请问一个只有一个主持人能否举办全部活动。
>
> 数据范围：$1 \leq n \leq 10^5$ ， $0 \leq start_i, end_i \leq 10^9$
>
> 补充说明：
>
> 示例1
> 输入：[[0,10],[10,20],[20,30]]
> 输出：true
> 说明：
>
> 示例2
> 输入：[[0,10],[10,20],[15,30]]
> 输出：false
> 说明：

3. **解法：**

算法思路：

区间问题技巧：左端点排序或者按照右端点排序

左端点排序后，我们仅需考虑后续区间是否能与前一个区间重叠即可，美滋滋。

C++ 算法代码：

```cpp
class Solution
{
public:
    bool hostschedule(vector<vector<int> >& schedule)
    {
        sort(schedule.begin(), schedule.end());

        for(int i = 1; i < schedule.size(); i++)
        {
            if(schedule[i][0] < schedule[i - 1][1]) return false;
        }
```

```
        return true;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public boolean hostschedule (ArrayList<ArrayList<Integer>> sc)
    {
        int m = sc.size(), n = sc.get(0).size();
        int[][] schedule = new int[m][n];
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                schedule[i][j] = sc.get(i).get(j);
            }
        }

        Arrays.sort(schedule, (v1, v2) ->
        {
            return v1[0] - v2[0];
        });

        for(int i = 1; i < schedule.length; i++)
        {
            if(schedule[i][0] < schedule[i - 1][1]) return false;
        }
        return true;
    }
}
```


3. 分割等和子集（动态规划 - 01背包）

（题号：2383964）

1. **题目链接：** DP45 分割等和子集
2. **题目描述：**

题目描述：给定一个只包含正整数的数组 nums ， 请问能否把这个数组取出若干个数使得取出的数之和和剩下的数之和相同。

数据范围： $1 \leq n \leq 500$ ， 数组中的元素满足 $1 \leq nums_i \leq 100$

输入描述：第一行输入一个正整数 n ， 表示数组 nums 的长度。
第二行输入 n 个正整数，表示数组中的值。

输出描述：如果满足题目条件，输出 true ， 否则输出 false

补充说明：

示例1
输入：4
1 5 11 5
输出：true
说明：

示例2
输入：4
1 2 3 5
输出：false
说明：

3. **解法：**

算法思路：

01 背包问题：原问题转换成，从 n 个数中选，总和恰好为 sum / 2，能否挑选出来。

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 510, M = 510 * 110 / 2;

int n;
int arr[N];
int dp[N][M];

int main()
{
    cin >> n;
    int sum = 0;
    for(int i = 1; i <= n; i++)
    {
        cin >> arr[i];
        sum += arr[i];
    }

```

```
    if(sum % 2 == 1) cout << "false" << endl;
    else
    {
        sum /= 2;
        dp[0][0] = true;
        for(int i = 1; i <= n; i++)
        {
            for(int j = 0; j <= sum; j++)
            {
                dp[i][j] = dp[i - 1][j];
                if(j >= arr[i])
                {
                    dp[i][j] = dp[i][j] || dp[i - 1][j - arr[i]];
                }
            }
        }
        if(dp[n][sum]) cout << "true" << endl;
        else cout << "false" << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static int N = 510, M = 510 * 110 / 2;
    public static int n;
    public static int[] arr = new int[N];
    public static boolean[][] dp = new boolean[N][M];

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        int sum = 0;
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
            sum += arr[i];
```

```java
        }

        if(sum % 2 == 1) System.out.println("false");
        else
        {
            sum /= 2;
            dp[0][0] = true;
            for(int i = 1; i <= n; i++)
            {
                for(int j = 0; j <= sum; j++)
                {
                    dp[i][j] = dp[i - 1][j];
                    if(j >= arr[i])
                    {
                        dp[i][j] = dp[i][j] || dp[i - 1][j - arr[i]];
                    }
                }
            }
            if(dp[n][sum]) System.out.println("true");
            else System.out.println("false");
        }
    }
}
```

Day26

1. 小红的ABC（字符串 + 找规律）

（题号：2305522）

1. **题目链接：** 小红的ABC
2. **题目描述：**

题目描述：小红拿到了一个只包含 'a' , 'b' , 'c' 三种字符的字符串。
小红想知道，这个字符串最短的、长度超过 1 的回文子串的长度是多少？
子串定义：字符串取一段连续的区间。例如"abcca"的子串有"ab"、"bcca"等，但"aca"则不是它的子串。
回文的定义：一个字符串正着读和倒着读都是相同的，那么定义它的回文的。

输入描述：一个只包含 'a' , 'b' , 'c' 三种字符的字符串。
数据范围：字符串长度不小于2，且不超过100

输出描述：如果不存在长度超过1的回文子串，则输出-1。
否则输出长度超过1的最短回文子串的长度。

补充说明：

示例1
输入：abcca
输出：2
说明："cc"即为其最短回文子串。

示例2
输入：abcab
输出：-1
说明：

3. **解法：**

算法思路：

由于题目要找的是最短的回文子串，并且只有三个字母 a b c，因此最短的回文子串的长度要么是 2，要么是 3。

因此，我们仅需枚举所有的二元组以及三元组就好了。

C++ 算法代码：

```cpp
#include <iostream>
#include <string>

using namespace std;

string s;

int main()
{
    cin >> s;
    int ret = -1; // 有可能并没有回文串
    int n = s.size();

    for(int i = 0; i < n; i++)
    {
        if(i + 1 < n && s[i] == s[i + 1]) // 判断长度为 2 的子串
        {
            ret = 2;
```

```
            break;
        }
        if(i + 2 < n && s[i] == s[i + 2]) // 判断长度为 3 的子串
        {
            ret = 3;
        }
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        char[] s = in.next().toCharArray();

        int ret = -1; // 并没有回文串
        int n = s.length;
        for(int i = 0; i < n; i++)
        {
            if(i + 1 < n && s[i] == s[i + 1]) // 判断长度为 2 的子串
            {
                ret = 2;
                break;
            }
            if(i + 2 < n && s[i] == s[i + 2]) // 判断长度为 3 的子串
            {
                ret = 3;
            }
        }
        System.out.println(ret);
    }
}
```

2. 不相邻取数（动态规划 - 线性 dp）

（题号：1832016）

1. **题目链接：** DP23 不相邻取数

2. **题目描述：**

> 题目描述：小红拿到了一个数组。她想取一些不相邻的数，使得取出来的数之和尽可能大。你能帮帮她吗？
>
> 输入描述：第一行输入一个正整数 n ，代表数组长度
> 第二行输入 n 个正整数 a_i ，代表整个数组。
> $1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9$
>
> 输出描述：不相邻的数的最大和。
>
> 补充说明：
>
> 示例1
> 输入：4
> 2 6 4 1
> 输出：7
> 说明：取 6 和 1 即可

3. **解法：**

算法思路：

打家劫舍~

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 2e5 + 10;

int n;
int arr[N];
int f[N], g[N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];

    for(int i = 1; i <= n; i++)
```

```
    {
        f[i] = g[i - 1] + arr[i];
        g[i] = max(f[i - 1], g[i - 1]);
    }

    cout << max(f[n], g[n]) << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();

        int[] arr = new int[n + 1];
        int[] f = new int[n + 1];
        int[] g = new int[n + 1];

        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }

        for(int i = 1; i <= n; i++)
        {
            f[i] = g[i - 1] + arr[i];
            g[i] = Math.max(f[i - 1], g[i - 1]);
        }

        System.out.println(Math.max(f[n], g[n]));
    }
}
```

3. 空调遥控（二分 / 滑动窗口）

（题号：2222857）

1. **题目链接：** 空调遥控

2. **题目描述：**

> 题目描述：dd作为集训队的队长，一直掌管着集训室的空调遥控器，她需要调整温度使队员们更好地进入训练状态，已知集训室一共有n名队员，每位队员都有一个温度诉求$a[i](1 \le i \le n)$，当室内温度为K时，当且仅当$|a[i]-K| \le p$时，这个队员能够正常进入训练状态，否则就会开始躁动，作为队长，dd需要调整好温度，她想知道，在最佳情况下，最多有多少队员同时进入训练状态
>
> 输入描述：第一行两个数n,p(1≤n,p≤1000000),含义如题面描述
> 接下来一行n个数a[i](1≤a[i]≤1000000)表示每个队员的温度诉求
>
> 输出描述：输出一个数字，表示最多有多少队员同时进入训练状态
>
> 补充说明：
>
> 示例1
> 输入：6 2
> 1 5 3 2 4 6
> 输出：5
> 说明：温度调成3或4，都可以满足5名队员同时进入训练状态

3. **解法：**

算法思路：

先排序。

解法一：滑动窗口

维护窗口内最大值与最小值的差在 `2 * p` 之间。

解法二：二分查找

枚举所有的温度，二分出符合要求的学生区间，然后统计个数。

C++ 算法代码：

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1e6 + 10;

int n, p;
int arr[N];

int main()
```

```
{
    cin >> n >> p;
    for(int i = 0; i < n; i++) cin >> arr[i];
    sort(arr, arr + n);

    int ret = 0, left = 0, right = 0;
    p *= 2;

    while(right < n)
    {
        while(arr[right] - arr[left] > p)
        {
            left++;
        }
        ret = max(ret, right - left + 1);
        right++;
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), p = in.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++)
        {
            arr[i] = in.nextInt();
        }
        Arrays.sort(arr);

        int left = 0, right = 0, ret = 0;
        p *= 2;
        while(right < n)
        {
```

```
            while(arr[right] - arr[left] > p)
            {
                left++;
            }
            ret = Math.max(ret, right - left + 1);
            right++;
        }
        System.out.println(ret);
    }
}
```

Day27

1. kotori和气球（组合数学）

（题号：500541）

1. **题目链接：** kotori和气球
2. **题目描述：**

> 题目描述：kotori最近迷上了摆气球的游戏。她一共有n种气球，每种气球有无数个。她要拿出若干个气球摆成一排。
> 但是，由于气球被施放了魔法，同样种类的气球如果相邻会发生爆炸，因此若两个相邻的气球种类相同被视为不合法的。
> kotori想知道，摆成一排m个一共有多少种不同的方案？
> 由于该数可能过大，只需要输出其对109取模的结果。
>
> 输入描述：输入仅有一行，为两个整数n和m(1≤n,m≤100)
>
> 输出描述：输出一个整数，为方案数对109取模的结果。
>
> 补充说明：
>
> 示例1
> 输入：3 2
> 输出：6
> 说明：假设3种气球标记为1、2、3，那么共有以下6种方案：[1,2] [1,3] [2,1] [2,3] [3,1] [3,2]。

3. **解法：**

算法思路：

简单的排列组合问题，结果等于 n 与 m 个 n - 1 的乘积。

C++ 算法代码：

```
#include <iostream>

using namespace std;
```

```
4
5  const int MOD = 109;
6
7  int main()
8  {
9      int n, m;
10     cin >> n >> m;
11     int ret = n;
12     for(int i = 0; i < m - 1; i++)
13     {
14         ret = ret * (n - 1) % MOD;
15     }
16
17     cout << ret << endl;
18
19     return 0;
20 }
```

Java 算法代码：

```
1  import java.util.*;
2
3  public class Main
4  {
5      public static int MOD = 109;
6
7      public static void main(String[] args)
8      {
9          Scanner in = new Scanner(System.in);
10         int n = in.nextInt(), m = in.nextInt();
11
12         int ret = n;
13         for(int i = 0; i < m - 1; i++)
14         {
15             ret = ret * (n - 1) % MOD;
16         }
17
18         System.out.println(ret);
19     }
20 }
```

2. 走迷宫（BFS）

（题号：2373924）

1. **题目链接：** AB20 走迷宫

2. **题目描述：**

> 题目描述：给定一个$n \times m$的网格，在网格中每次在不超过边界的情况下可以选择向上、向下、向左、向右移动一格。网格中的一些格子上放置有障碍物，放有障碍物的格子不能到达。求从(x_s, y_s)到(x_t, y_t)最少的移动次数。若不能到达，输出-1。
>
> 注：若要使用Python通过此题，请选择PyPy提交。
>
> 输入描述：第一行输入两个整数n, m $(1 \leq n, m \leq 1000)$，表示网格大小。
> 第二行输入四个整数x_s, y_s, x_t, y_t $(1 \leq x_s, x_t \leq n, 1 \leq y_s, y_t \leq m)$，表示起点和终点的坐标。
> 接下来的n行，每行输入一个长度为m的字符串。其中，第i行第j个字符表示第i行第j列的格子上的障碍物情况，若字符为'*'，则格子上有障碍物，若字符为'.'，则格子上没有障碍物。
> 保证起点不存在障碍物。
>
> 输出描述：输出一行一个整数，表示从(x_s, y_s)到(x_t, y_t)最少的移动次数。
>
> 补充说明：
>
> 示例1
> 输入：5 5
> 1 1 5 5
> .....
> ****.
> .....
> **.**
> .....
> 输出：12
> 说明：

3. **解法：**

算法思路：

简单 bfs 应用题。

C++ 算法代码：

```cpp
#include <iostream>
#include <cstring>
#include <queue>

using namespace std;

const int N = 1010;

int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};

int n, m;
int x1, y1, x2, y2;
```

```cpp
char arr[N][N];
int dist[N][N]; // [i, j] 位置是否已经搜索过，以及到达 [i, j] 位置的最短距离

int bfs()
{
    if(arr[x2][y2] == '*') return -1;

    memset(dist, -1, sizeof dist); // 表示还没开始搜索
    queue<pair<int, int>> q;
    q.push({x1, y1});
    dist[x1][y1] = 0;

    while(q.size())
    {
        auto [a, b] = q.front();
        q.pop();
        for(int i = 0; i < 4; i++)
        {
            int x = a + dx[i], y = b + dy[i];
            if(x >= 1 && x <= n && y >= 0 && y <= m && arr[x][y] == '.' &&
dist[x][y] == -1)
            {
                q.push({x, y});
                dist[x][y] = dist[a][b] + 1;
                if(x == x2 && y == y2) return dist[x2][y2];
            }
        }
    }
    return -1;
}

int main()
{
    cin >> n >> m >> x1 >> y1 >> x2 >> y2;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            cin >> arr[i][j];
        }
    }

    cout << bfs() << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static int N = 1010;
    public static int[] dx = {0, 0, 1, -1};
    public static int[] dy = {-1, 1, 0, 0};

    public static int n, m, x1, y1, x2, y2;
    public static char[][] arr = new char[N][N];
    public static int[][] dist = new int[N][N]; // 标记当前位置有没有搜索过，以及走到该位置时候的最短步数

    public static int bfs()
    {
        if(arr[x2][y2] == '*') return -1;

        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++)
                dist[i][j] = -1; // 表明刚开始每个位置都没有搜索过

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{x1, y1});
        dist[x1][y1] = 0;

        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 1 && x <= n && y >= 1 && y <= m && arr[x][y] == '.' && dist[x][y] == -1)
                {
                    q.add(new int[]{x, y});
                    dist[x][y] = dist[a][b] + 1;
                    if(x == x2 && y == y2)
                    {
                        return dist[x][y];
                    }
                }
```

```
            }
        }
        return -1;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); m = in.nextInt();
        x1 = in.nextInt(); y1 = in.nextInt();
        x2 = in.nextInt(); y2 = in.nextInt();
        for(int i = 1; i <= n; i++)
        {
            String tmp = in.next();
            for(int j = 1; j <= m; j++)
            {
                arr[i][j] = tmp.charAt(j - 1);
            }
        }

        System.out.println(bfs());
    }
}
```

3. 主持人调度（二）（贪心 + 优先级队列）

（题号：1267319）

1. **题目链接：** NC147 主持人调度（二）
2. **题目描述：**

题目描述：有 n 个活动即将举办，每个活动都有开始时间与活动的结束时间，第 i 个活动的开始时间是 $start_i$,第 i 个活动的结束时间是 end_i ,举办某个活动就需要为该活动准备一个活动主持人。

一位活动主持人在同一时间只能参与一个活动。并且活动主持人需要全程参与活动，换句话说，一个主持人参与了第 i 个活动，那么该主持人在 $(start_i,end_i)$ 这个时间段不能参与其他任何活动。求为了成功举办这 n 个活动，最少需要多少名主持人。

数据范围：$1 \le n \le 10^5$ ， $-2^{32} \le start_i \le end_i \le 2^{31}-1$

复杂度要求：时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$

补充说明：$1 \le n \le 10^5$
$start_i, end_i$ 在int范围内

示例1
输入：2,[[1,2],[2,3]]
输出：1
说明：只需要一个主持人就能成功举办这两个活动

示例2
输入：2,[[1,3],[2,4]]
输出：2
说明：需要两个主持人才能成功举办这两个活动

3. **解法：**

算法思路：

左端点排序，然后搞个堆：

a. 先把第一个区间的右端点加入到堆中；
b. 遍历后面的区间，分情况讨论：
   i. 如果左端点大于等于堆顶元素，能接在后面，干掉堆顶，然后把这个区间的右端点加入堆；
   ii. 否则的话，只能再来一个人，只把这个区间的右端点加入堆。
c. 最后堆的大小就是需要的人数。

C++ 算法代码：

```
class Solution
{
public:
    int minmumNumberOfHost(int n, vector<vector<int> >& startEnd)
    {
        sort(startEnd.begin(), startEnd.end());
        priority_queue<int, vector<int>, greater<int>> heap; // 创建一个小根堆
        heap.push(startEnd[0][1]); // 先把第一个区间放进去

        for(int i = 1; i < n; i++) // 处理剩下的区间
        {
            int a = startEnd[i][0], b = startEnd[i][1];
            if(a >= heap.top()) // 没有重叠
```

```
            {
                heap.pop();
                heap.push(b);
            }
            else // 有重叠
            {
                heap.push(b); // 重新安排一个人
            }
        }
        return heap.size();
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int minmumNumberOfHost (int n, int[][] startEnd)
    {
        Arrays.sort(startEnd, (a, b) ->
        {
            //return a[0] <= b[0] ? -1 : 1;
            return a[0] - b[0];
        });

        PriorityQueue<Integer> heap = new PriorityQueue<>(); // 小根堆
        heap.offer(startEnd[0][1]);

        for(int i = 1; i < n; i++)
        {
            int a = startEnd[i][0], b = startEnd[i][1];
            if(a >= heap.peek()) // 无重叠
            {
                heap.poll();
                heap.offer(b);
            }
            else // 有重叠
            {
                heap.offer(b);
            }
        }

```

```
30          return heap.size();
31      }
32  }
```

Day28

1. 游游的重组偶数（数学）

（题号：10274325）

1. **题目链接：** [编程题]游游的重组偶数
2. **题目描述：**

题目描述：游游拿到了一个正整数，她希望你能重排这个正整数的数位，使得它变成偶数（不能有前导零）。你能帮帮她吗？
注：重排后可以和原数相等。
一共有 q 次询问。

输入描述：第一行输入一个正整数 q，代表询问次数。
接下来的 q行，每行输入一个正整数 x。
$1 \leq q \leq 10^5$
$1 \leq x \leq 10^9$

输出描述：输出q行，每行代表一次询问。
如果存在合法解，请输出一个重排后的正整数，务必保证其为偶数。有多解时输出任意即可。
如果不存在合法解，直接输出-1。

补充说明：

示例1
输入：3
13
123
24
输出：-1
132
24
说明：

3. **解法：**

算法思路：

偶数的性质就是最后一位能被 2 整除即可。

因此，把该数中能被 2 整除的数字放在最后就好了，但是注意不要出现前导零。

C++ 算法代码：

```
1  #include <iostream>
2  #include <string>
```

```cpp

using namespace std;

int main()
{
    int q;
    string s;
    cin >> q;

    while(q--)
    {
        cin >> s; // 把每个数当成字符串处理
        int n = s.size();
        for(int i = n - 1; i >= 0; i--)
        {
            if((s[i] - '0') % 2 == 0)
            {
                swap(s[i], s[n - 1]);
                break;
            }
        }
        if((s[n - 1] - '0') % 2 == 0) cout << s << endl;
        else cout << -1 << endl;
    }

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;
import java.io.*;

public class Main
{
    public static PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));
    public static Read in = new Read();

    public static void Swap(char[] s, int i, int j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
```

```
14      }
15
16      public static void main(String[] args) throws IOException
17      {
18          int q = in.nextInt();
19          while(q-- != 0)
20          {
21              char[] s = in.next().toCharArray();
22              int n = s.length;
23
24              for(int i = n - 1; i >= 0; i--)
25              {
26                  if((s[i] - '0') % 2 == 0)
27                  {
28                      Swap(s, i, n - 1);
29                      break;
30                  }
31              }
32
33              if((s[n - 1] - '0') % 2 == 0)
34              {
35                  out.println(s);
36              }
37              else
38              {
39                  out.println(-1);
40              }
41          }
42
43          out.close();
44      }
45  }
46
47  class Read
48  {
49      StringTokenizer st = new StringTokenizer("");
50      BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
51      String next() throws IOException
52      {
53          while(!st.hasMoreTokens())
54          {
55              st = new StringTokenizer(bf.readLine());
56          }
57          return st.nextToken();
58      }
59
60      String nextLine() throws IOException
```

```
61      {
62          return bf.readLine();
63      }
64
65      int nextInt() throws IOException
66      {
67          return Integer.parseInt(next());
68      }
69
70      long nextLong() throws IOException
71      {
72          return Long.parseLong(next());
73      }
74
75      double nextDouble() throws IOException
76      {
77          return Double.parseDouble(next());
78      }
79  }
```

2. 体操队形（DFS + 枚举）

（题号：2227705）

1. **题目链接：** 体操队形

2. **题目描述：**

> 题目描述：dd作为体操队队长，在给队员们排队形，体操队形为一个单独的纵列，体操队有n个同学，标号为$1 \sim n$，对于$i(1 \le i \le n)$号队员，会有一个诉求$(1 \le a[i] \le n)$，表示他想排在$a[i]$号队员前面，当$a[i] = i$时，我们认为他没有位置需求，随便排哪儿都行，dd想知道有多少种队形方案，可以满足所有队员的要求。
>
> 输入描述：读入第一行一个数字n(2≤n≤10)
> 第二行n个数字，表示a[i],保证1≤a[i]≤n
>
> 输出描述：输出一行，表示方案数
>
> 补充说明：

```
示例1
输入：3
      1 1 2
输出：1
说明：
```

3. **解法：**

算法思路：

画出决策树，注意剪枝策略~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 15;

int ret;
int n;
bool vis[N];
int arr[N];

void dfs(int pos)
{
    if(pos == n + 1) // 找到一种合法方案
    {
        ret++;
        return;
    }

    for(int i = 1; i <= n; i++)
    {
        if(vis[i]) continue; // 当前 i 已经放过了 - 剪枝
        if(vis[arr[i]]) return; // 剪枝
        vis[i] = true; // 相当于放上 i 号队员
        dfs(pos + 1);
        vis[i] = false; // 回溯- 还原现场
    }
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];

    dfs(1);

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

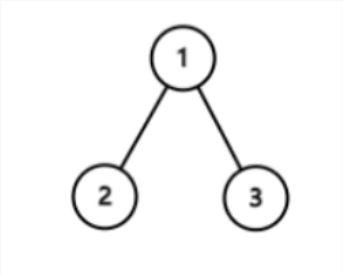
```java
import java.util.*;

public class Main
{
    public static int N = 15;
    public static int n, ret;
    public static boolean[] vis = new boolean[N];
    public static int[] arr = new int[N];

    public static void dfs(int pos)
    {
        if(pos == n + 1) // 找到一种合法情况
        {
            ret++;
            return;
        }

        for(int i = 1; i <= n; i++)
        {
            if(vis[i] == true) continue; // 剪枝 - i 号队员已经放过了
            if(vis[arr[i]]) return; // 剪枝
            vis[i] = true; // 相当于已经放上 i 号队员
            dfs(pos + 1);
            vis[i] = false; // 回溯 - 恢复现场
        }
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }

        dfs(1);

        System.out.println(ret);
    }
}
```

3. 二叉树中的最大路径和（树形dp）

（题号：622）

1. **题目链接：** **NC6 二叉树中的最大路径和**

2. **题目描述：**

> 题目描述：二叉树里面的路径被定义为:从该树的任意节点出发，经过父=>子或者子=>父的连接，达到任意节点的序列。
> 注意:
> 1.同一个节点在一条二叉树路径里中最多出现一次
> 2.一条路径至少包含一个节点，且不一定经过根节点
>
> 给定一个二叉树的根节点root，请你计算它的最大路径和
>
> 例如:
> 给出以下的二叉树，
>
> 
>
>
> 最优路径是:2=>1=>3，或者3=>1=>2，最大路径和=2+1+3=6
>
> 数据范围：节点数满足 $11n110^5$ ，节点上的值满足 $|val|11000$
> 要求：空间复杂度 $O(1)$，时间复杂度 $O(n)$

3. **解法：**

算法思路：

树形dp：

a. 左子树收集：以左子树为起点的最大单链和；

b. 右子树收集：以右子树为起点的最大单链和；

c. 根节点要做的事情：整合左右子树的信息，得到经过根节点的最大路径和；

d. 向上返回：以根节点为起点的最大单链和

C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     int ret = -1010;
5
```

```
    int maxPathSum(TreeNode* root)
    {
        dfs(root);
        return ret;
    }

    int dfs(TreeNode* root)
    {
        if(root == nullptr) return 0;

        int l = max(0, dfs(root->left));// 左子树的最大单链和
        int r = max(0, dfs(root->right)); // 右子树的最大单链和
        // 经过root的最大路径和
        ret = max(ret, root->val + l + r);

        return root->val + max(l, r);
    }
};
```

Java 算法代码：

```
import java.util.*;

/*
 * public class TreeNode {
 *   int val = 0;
 *   TreeNode left = null;
 *   TreeNode right = null;
 *   public TreeNode(int val) {
 *     this.val = val;
 *   }
 * }
 */
public class Solution
{
    int ret = -1010;

    public int maxPathSum (TreeNode root)
    {
        dfs(root);
        return ret;
    }

    int dfs(TreeNode root)
```

```
24      {
25          if(root == null) return 0;
26
27          int l = Math.max(0, dfs(root.left)); // 左子树为根的最大单链和
28          int r = Math.max(0, dfs(root.right)); // 右子树为根的最大单链和
29          // 经过root的最大路径和
30          ret = Math.max(ret, root.val + l + r);
31
32          return root.val + Math.max(l, r);
33      }
34  }
35
```

Day29

1. 排序子序列（模拟）

（题号：100346）

1. **题目链接：** [编程题]排序子序列

2. **题目描述：**

题目描述：牛牛定义排序子序列为一个数组中一段连续的子序列，这段子序列是非递增或者非递减排序的。
牛牛有一个长度为 n 的整数数组 a，他现在需要把数组 a 分为若干段排序子序列，牛牛想知道最少可以把这个数组分为几段排序子序列。

输入描述：第一行输入一个正整数 n。
第二行输入 n 个正整数 a_i，表示数组的每个数。

$$1 \leq n \leq 10^5$$
$$1 \leq a_i \leq 10^9$$

输出描述：输出一个整数，可以将 a 最少划分为多少段排序子序列。

补充说明：

示例1
输入：6
1 2 3 2 2 1
输出：2
说明：可以把划分为[1,2,3]和[2,2,1]两个排序子序列。

3. **解法：**

算法思路：

根据题意，用指针模拟即可。

（注意：本道题的测试数据不严谨，有可能错误的代码也能提交过~）

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 1e5 + 10;

int n;
int arr[N];

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++) cin >> arr[i];

    int ret = 0, i = 0;
    while(i < n)
    {
        if(i == n - 1)
        {
            ret++;
            break;
        }

        if(arr[i] < arr[i + 1])
        {
            while(i + 1 < n && arr[i] <= arr[i + 1]) i++;
            ret++;
        }
        else if(arr[i] > arr[i + 1])
        {
            while(i + 1 < n && arr[i] >= arr[i + 1]) i++;
            ret++;
        }
        else
        {
            while(i + 1 < n && arr[i] == arr[i + 1]) i++;
        }
        i++;
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++)
        {
            arr[i] = in.nextInt();
        }

        int ret = 0, i = 0;
        while(i < n)
        {
            if(i == n - 1)
            {
                ret++;
                break;
            }

            if(arr[i] < arr[i + 1])
            {
                while(i + 1 < n && arr[i] <= arr[i + 1]) i++;
                ret++;
            }
            else if(arr[i] > arr[i + 1])
            {
                while(i + 1 < n && arr[i] >= arr[i + 1]) i++;
                ret++;
            }
            else
            {
                while(i + 1 < n && arr[i] == arr[i + 1]) i++;
            }
            i++;
        }

        System.out.println(ret);
    }
```

```
44 }
```

2. 削减整数（贪心）

（题号：1389171）

1. **题目链接：** 消减整数

2. **题目描述：**

题目描述：给出一个正整数H，从1开始减，第一次必须减1，每次减的数字都必须和上一次相同或者是上一次的两倍，请问最少需要几次能把H恰好减到0。

输入描述：第一行给出一个正整数T，$1 \leq T \leq 10^4$
接下来T行每行一个$H, 1 \leq H \leq 10^9$

输出描述：每行一个正整数代表最少的次数

补充说明：

示例1
输入：3
3
5
7
输出：2
3
3
说明：

3. **解法：**

算法思路：

贪心 + 数学。

a. 尽可能的翻倍；

b. 不能无脑翻倍，只能是 `2 * cur` 的倍数时，才能翻倍。

C++ 算法代码：

```
#include <iostream>

using namespace std;

int t, h;

int fun()
{
    int ret = 0, a = 1;
```

```
    while(h)
    {
        h -= a;
        ret++;
        if(h % (a * 2) == 0)
        {
            a *= 2;
        }
    }
    return ret;
}

int main()
{
    cin >> t;
    while(t--)
    {
        cin >> h;
        cout << fun() << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        while(t-- != 0)
        {
            int h = in.nextInt();
            int ret = 0, a = 1;
            while(h != 0)
            {
                h -= a;
                ret++;
                if(h % (a * 2) == 0)
                {
```

```
                    a *= 2;
                }
            }
            System.out.println(ret);
        }
    }
}
```

3. 最长上升子序列(二)（贪心 + 二分）

（题号：2281421）

1. **题目链接：** NC164 最长上升子序列(二)

2. **题目描述：**

> 题目描述：给定数组 arr，设长度为 n，输出 arr 的最长上升子序列。（如果有多个答案，请输出其中 按数值(注：区别于按单个字符的ASCII码值)进行比较的 字典序最小的那个）
>
> 数据范围：$0 \le n \le 200000, 0 \le arr_i \le 1000000000$
> 要求：空间复杂度 $O(n)$，时间复杂度 $O(nlogn)$
>
> 补充说明：$n \le 10^5$
> $1 \le arr_i \le 10^9$

示例1
输入：[2,1,5,3,6,4,8,9,7]
输出：[1,3,4,8,9]
说明：

示例2
输入：[1,2,8,6,4]
输出：[1,2,4]
说明：其最长递增子序列有3个，（1，2，8）、（1，2，6）、（1，2，4）其中第三个 按数值进行比较的字典序 最小，故答案为（1，2，4）

3. **解法：**

算法思路：

我们在考虑最长递增子序列的长度的时候，其实并不关心这个序列长什么样子，我们只是关心最后一个元素是谁。这样新来一个元素之后，我们就可以判断是否可以拼接到它的后面。

因此，我们可以创建一个数组，统计长度为 `x` 的递增子序列中，最后一个元素是谁。为了尽可能的让这个序列更长，我们仅需统计长度为 `x` 的所有递增序列中最后一个元素的「最小值」。

统计的过程中发现，数组中的数呈现「递增」趋势，因此可以使用「二分」来查找插入位置。

C++ 算法代码：

```cpp
class Solution
{
    int dp[100010] = { 0 }; // dp[i] 表示：长度为 i 的最小末尾
    int pos = 0;

public:
    int LIS(vector<int>& a)
    {
        for(auto x : a)
        {
            // 查找 x 应该放在哪个位置
            if(pos == 0 || x > dp[pos])
            {
                dp[++pos] = x;
            }
            else
            {
                // 二分查找插入位置
                int l = 1, r = pos;
                while(l < r)
                {
                    int mid = (l + r) / 2;
                    if(dp[mid] >= x) r = mid;
                    else l = mid + 1;
                }
                dp[l] = x;
            }
        }
        return pos;
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{

    public int LIS (int[] a)
    {
        int n = a.length;

        int[] dp = new int[n + 1]; // dp[i] 表示长度为 i 的最小末尾
```

```
        int pos = 0;

        for(int x : a)
        {
            // 找 x 应该放在哪里
            if(pos == 0 || x > dp[pos])
            {
                dp[++pos] = x;
            }
            else
            {
                // 二分查找插入位置
                int l = 1, r = pos;
                while(l < r)
                {
                    int mid = (l + r) / 2;
                    if(dp[mid] >= x) r = mid;
                    else l = mid + 1;
                }
                dp[l] = x;
            }
        }
        return pos;
    }
}
```


Day30

1. 爱吃素（数学）

（题号：2092095）

1. **题目链接：** 爱吃素
2. **题目描述：**

题目描述：牛妹是一个爱吃素的小女孩，所以很多素数都害怕被她吃掉。

一天，两个数字a和b为了防止被吃掉，决定和彼此相乘在一起，这样被吃掉的风险就会大大降低，但仍有一定的可能被吃掉，请你判断他们相乘后是否仍有被吃掉的风险。

也就是说，请你判断$a \times b$是否是素数。

素数是指大于1的正整数中，有且仅有两个因子的数。

输入描述：输入第一行是一个整数$T(1 \leq T \leq 10)$，表示测试组数。

接下来T行，每一行两个整数$a, b(1 \leq a, b \leq 10^{11})$。

输出描述：对于每一行输入，若输入满足$a \times b$是素数，输出一行"YES"，否则输出一行"NO"（没有引号）。

补充说明：

示例1
输入：3
2 3
1 7
1 4
输出：NO
YES
NO
说明：

3. 解法：

算法思路：

判断两数相乘是否是素数。

- 不能直接乘起来然后判断，因为数据量太大了，不仅存不下，而且会超时；
- 因此根据素数的性质，分类讨论。

C++ 算法代码：

```cpp
#include <iostream>
#include <cmath>

using namespace std;

typedef long long LL;

bool isprim(LL x)
{
    if(x < 2) return false;
    for(int i = 2; i <= sqrt(x); i++)
    {
        if(x % i == 0) return false;
    }
    return true;
}
```

```
17
18 int main()
19 {
20     int t;
21     cin >> t;
22     while(t--)
23     {
24         LL a, b;
25         cin >> a >> b;
26         if((a == 1 && isprim(b)) || (b == 1 && isprim(a)))
27         {
28             cout << "YES" << endl;
29         }
30         else
31         {
32             cout << "NO" << endl;
33         }
34     }
35
36     return 0;
37 }
```

Java 算法代码：

```
1  import java.util.*;
2
3  public class Main
4  {
5      public static boolean isprim(long x)
6      {
7          if(x < 2) return false;
8          for(int i = 2; i <= Math.sqrt(x); i++)
9          {
10             if(x % i == 0) return false;
11         }
12         return true;
13     }
14
15     public static void main(String[] args)
16     {
17         Scanner in = new Scanner(System.in);
18         int t = in.nextInt();
19         while(t-- != 0)
20         {
```

```
            long a = in.nextLong(), b = in.nextLong();
            if((a == 1 && isprim(b)) || (b == 1 && isprim(a)))
            {
                System.out.println("YES");
            }
            else
            {
                System.out.println("NO");
            }
        }
    }
}
```

2. 相差不超过k的最多数（滑动窗口）

（题号：2403293）

1. **题目链接：** **AB33 相差不超过k的最多数**

2. **题目描述：**

题目描述：给定一个数组，选择一些数，要求选择的数中任意两数差的绝对值不超过 k 。问最多能选择多少个数?

输入描述：第一行输入两个正整数 n和k。
第二行输入 n 个正整数a_i，用空格隔开，表示这个数组。

输出描述：一个正整数，代表能选的最多数量。
数据范围：
$1 \leq n \leq 2 \times 10^5$
$1 \leq k, a_i \leq 10^9$

补充说明：

示例1
输入：5 3
2 1 5 3 2
输出：4
说明：显然，1和5不能同时选。所以最多只能选4个数。

3. **解法：**

算法思路：

排序 + 滑动窗口

C++ 算法代码：

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 2e5 + 10;

int n, k;
int arr[N];

int main()
{
    cin >> n >> k;
    for(int i = 0; i < n; i++) cin >> arr[i];
    sort(arr, arr + n);

    int left = 0, right = 0, ret = 1;
    while(right < n)
    {
        while(arr[right] - arr[left] > k)
        {
            left++;
        }
        ret = max(ret, right - left + 1);
        right++;
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), k = in.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++)
```

```
        {
            arr[i] = in.nextInt();
        }

        Arrays.sort(arr);
        int left = 0, right = 0, ret = 0;
        while(right < n)
        {
            while(arr[right] - arr[left] > k)
            {
                left++;
            }
            ret = Math.max(ret, right - left + 1);
            right++;
        }
        System.out.println(ret);
    }
}
```

3. 最长公共子序列(一)（动态规划 - LCS）

（题号：2357875）

1. **题目链接：** **DP19 最长公共子序列(一)**
2. **题目描述：**

题目描述：给定两个字符串 s1 和 s2，长度为 n 和 m 。求两个字符串最长公共子序列的长度。
所谓子序列，指一个字符串删掉部分字符（也可以不删）形成的字符串。例如：字符串 "arcaea" 的子序列有 "ara" 、 "rcaa" 等。但 "car" 、 "aaae" 则不是它的子序列。
所谓 s1 和 s2 的最长公共子序列，即一个最长的字符串，它既是 s1 的子序列，也是 s2 的子序列。
数据范围：$1 \leq m, n \leq 1000$ 。保证字符串中的字符只有小写字母。
要求：空间复杂度 $O(mn)$，时间复杂度 $O(mn)$
进阶：空间复杂度 $O(min(m, n))$，时间复杂度 $O(mn)$

输入描述：第一行输入一个整数 n 和 m ，表示字符串 s1 和 s2 的长度。
接下来第二行和第三行分别输入一个字符串 s1 和 s2。

输出描述：输出两个字符串的最长公共子序列的长度

补充说明：

示例1
输入：5 6
abcde
bdcaaa
输出：2
说明：最长公共子序列为 "bc" 或 "bd" ，长度为2

示例2
输入：3 3
abc
xyz
输出：0
说明：

3. **解法：**

算法思路：

经典两个字符串之间的 dp 问题。

1. 状态表示：`dp[i][j]` 表示：字符串 s1 中 `[0, i]` 区间与字符串 s2 中 `[0, j]` 区间内所有的子序列中，最长公共子序列的长度是多少。
2. 状态转移方程：根据最后一个位置的字符情况，划分问题：
   a. `s1[i] == s2[j]: dp[i][j] = dp[i - 1][j - 1] + 1;`
   b. `s1[i] != s2[j]: dp[i][j] = max(dp[i - 1][j], do[i][j - 1])。`

为了防止越界，我们把字符串的起始位置从 1 开始计算。

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 1010;

int n, m;
```

```cpp
char s1[N];
char s2[N];
int dp[N][N];

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> s1[i];
    for(int i = 1; i <= m; i++) cin >> s2[i];

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            if(s1[i] == s2[j]) dp[i][j] = dp[i - 1][j - 1] + 1;
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    cout << dp[n][m] << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), m = in.nextInt();
        char[] s1 = in.next().toCharArray();
        char[] s2 = in.next().toCharArray();
        int[][] dp = new int[n + 1][m + 1];

        for(int i = 1; i <= n; i++)
        {
            for(int j = 1; j <= m; j++)
            {
                if(s1[i - 1] == s2[j - 1]) dp[i][j] = dp[i - 1][j - 1] + 1;
```

```
19                 else dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
20             }
21         }
22         System.out.println(dp[n][m]);
23 
24     }
25 }
```