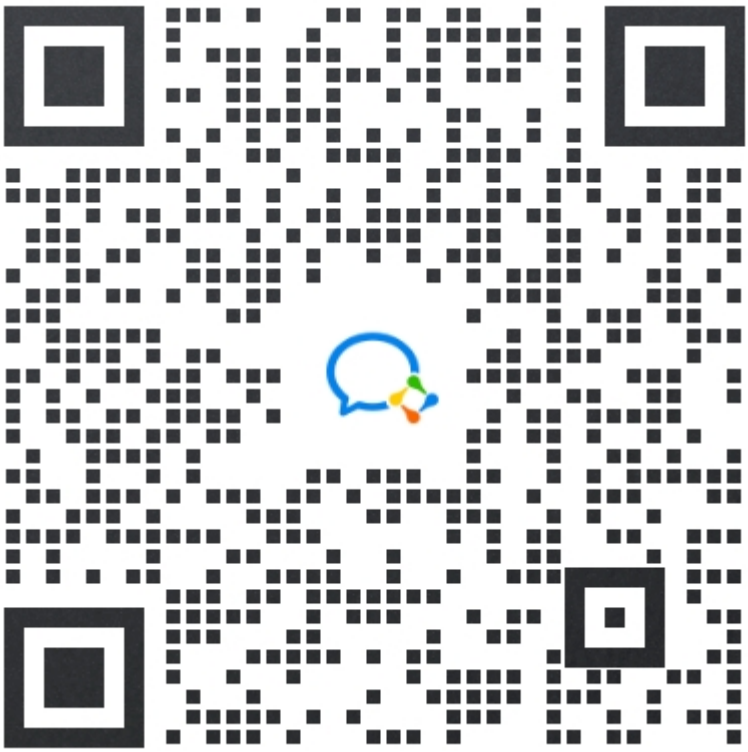
笔试强训第 03 周

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

Day13

1. 牛牛冲钻五（模拟）

（题号：2092058）

1. **题目链接：** 牛牛冲钻五

2. **题目描述：**

> 题目描述：牛牛最近在玩炉石传说，这是一款一对一对战的卡牌游戏，牛牛打算努力冲上钻五分段，获得丰厚的天梯奖励。
>
> 炉石传说的段位可以用星数来表示，具体规则为：若牛牛本场失败，则扣除一星；若牛牛本场获胜，需要看牛牛是否触发了连胜奖励，若牛牛获得了至少三连胜（即本局对局的上一局和上上局都获胜）则获得k星，否则获得一星。
>
> 现在给出牛牛游玩的n场记录，请你判断牛牛最终的星数和初始星数的差。
>
> 输入描述：第一行输入一个整数$T(1 \leq T \leq 10^4)$，测试组数。
>
> 每个测试的第一行输入两个整数$n(3 \leq n \leq 10^5)$和$k(2 \leq k \leq 100)$，牛牛共打了几场比赛与连胜奖励的星数。
>
> 接下来一行输入一个长为n的字符串，之中W表示获胜，L表示失败，保证只含有这两种字母。
>
> 保证所有用例的$\Sigma n \leq 10^5$。
>
> 输出描述：对每个测试用例，输出一个整数，表示牛牛最终的星数和初始星数的差。
>
> 补充说明：

> 示例1
> 输入：2
> 5 3
> WWWLW
> 5 3
> WLLLL
> 输出：5
> -3

3. **解法：**

算法思路：

简单模拟题，属于每次笔试的热身题~

C++ 算法代码：

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

using namespace std;

int t, n, k;
string s;
```

```
8
9  int fun()
10 {
11     int ret = 0;
12     for(int i = 0; i < s.size(); i++)
13     {
14         if(s[i] == 'L')
15         {
16              ret--;
17         }
18         else
19         {
20             if(i - 1 >= 0 && i - 2 >= 0 && s[i - 1] == 'W' && s[i - 2] == 'W')
21             {
22                 ret += k;
23             }
24             else
25             {
26                 ret += 1;
27             }
28         }
29     }
30     return ret;
31 }
32
33 int main()
34 {
35     cin >> t;
36     while(t--)
37     {
38         cin >> n >> k >> s;
39         cout << fun() << endl;
40     }
41
42     return 0;
43 }
```

Java 算法代码：

```
1  import java.util.*;
2
3  public class Main
4  {
5      public static void main(String[] args)
```

```java
    {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        while(t-- != 0)
        {
            int n = in.nextInt();
            int k = in.nextInt();
            char[] s = in.next().toCharArray();

            int ret = 0;
            for(int i = 0; i < s.length; i++)
            {
                if(s[i] == 'L') ret -= 1;
                else
                {
                    if(i - 1 >= 0 && i - 2 >= 0 && s[i - 1] == 'W' && s[i - 2] == 'W')
                    {
                        ret += k;
                    }
                    else
                    {
                        ret += 1;
                    }
                }
            }
            System.out.println(ret);
        }
    }
}
```

2. 最长无重复子数组（滑动窗口）

（题号：1008889）

1. **题目链接：** **NC41 最长无重复子数组**
2. **题目描述：**

题目描述：给定一个长度为n的数组arr，返回arr的最长无重复元素子数组的长度，无重复指的是所有数字都不相同。
子数组是连续的，比如[1,3,5,7,9]的子数组有[1,3]，[3,5,7]等等，但是[1,3,7]不是子数组

数据范围：$0 \leq arr.length \leq 10^5$，$0 < arr[i] \leq 10^5$

补充说明：

示例1
输入：[2,3,4,5]
输出：4
说明：[2,3,4,5]是最长子数组

示例2
输入：[2,2,3,4,3]
输出：3
说明：[2,3,4]是最长子数组

示例3
输入：[9]
输出：1
说明：

示例4
输入：[1,2,3,1,2,3,2,2]
输出：3
说明：最长子数组为[1,2,3]

3. 解法：

算法思路：

经典滑动窗口题目。

C++ 算法代码：

```
class Solution
{
    int hash[100010] = { 0 };

public:
    int maxLength(vector<int>& arr)
    {
        int left = 0, right = 0, n = arr.size();
        int ret = 0;
        while(right < n)
        {
            hash[arr[right]]++; // 进窗口
            while(hash[arr[right]] > 1) // 判断
            {
                hash[arr[left]]--; // 出窗口
                left++;
```

```
            }
            ret = max(ret, right - left + 1); // 更新结果
            right++;
        }
        return ret;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int maxLength (int[] arr)
    {
        int[] hash = new int[100010];
        int left = 0, right = 0, n = arr.length;
        int ret = 0;
        while(right < n)
        {
            hash[arr[right]]++; // 进窗口
            while(hash[arr[right]] > 1) // 判断
            {
                // 出窗口
                hash[arr[left]]--;
                left++;
            }
            ret = Math.max(ret, right - left + 1); // 更新结果
            right++;
        }
        return ret;
    }
}
```

3. 重排字符串（贪心 + 构造）

（题号：1748104）

1. **题目链接：** 重排字符串
2. **题目描述：**

题目描述：小红拿到了一个只由小写字母组成的字符串。她准备把这个字符串重排（只改变字母的顺序，不改变数量）
重排后小红想让新字符串不包含任意两个相同的相邻字母。
你能帮帮她吗？

输入描述：第一行一个正整数 n，代表字符串的长度。$(1 \leq n \leq 10^5)$
第二行为一个长度为 n 的、只由小写字母组成的字符串。

输出描述：如果可以完成重排，请在第一行输出一个“yes”，第二行输出重排后的字符串。如果有多个正解，输出任意即可。
如果不能重排，则直接输出“no”

补充说明：

示例1
输入：5
aaaaa
输出：no
说明：

示例2
输入：7
aabbccc
输出：yes
cabcabc
说明：bcbcaca也是正确答案（正确答案还有很多，输出任意即可）

3. 解法：

算法思路：

力扣有一道《距离相等的条形码》和这道题是差不多滴~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 100010;

int n;
char s[N];
char ret[N];

int main()
{
    cin >> n >> s;

    int hash[26] = { 0 }; // 统计每个字符的频次
    int maxIndex, maxCount = 0;
    for(int i = 0; i < n; i++)
    {
```

```
        if(maxCount < ++hash[s[i] - 'a'])
        {
            maxCount = hash[s[i] - 'a'];
            maxIndex = s[i] - 'a';
        }
    }

    if(maxCount > (n + 1) / 2) cout << "no" << endl;
    else
    {
        cout << "yes" << endl;
        int index = 0;
        // 先去摆放出现次数最多的
        while(maxCount--)
        {
            ret[index] = maxIndex + 'a';
            index += 2;
        }
        // 处理剩下的
        for(int i = 0; i < 26; i++)
        {
            if(hash[i] && i != maxIndex)
            {
                while(hash[i]--)
                {
                    if(index >= n) index = 1;
                    ret[index] = i + 'a';
                    index += 2;
                }
            }
        }
        // 打印结果
        for(int i = 0; i < n; i++) cout << ret[i];
        cout << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
```

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

        char maxChar = '0';
        int maxCount = 0;
        int[] hash = new int[26];
        // 找出现次数最多的字符以及次数
        for(int i = 0; i < n; i++)
        {
            char ch = s[i];
            if(++hash[ch - 'a'] > maxCount)
            {
                maxChar = ch;
                maxCount = hash[ch - 'a'];
            }
        }

        // 判断是否能重排
        if(maxCount > (n + 1) / 2)
        {
            System.out.println("no");
        }
        else
        {
            System.out.println("yes");
            char[] ret = new char[n];
            int i = 0;
            // 重新排列
            // 1. 先处理出现次数最多的字符
            while(maxCount-- != 0)
            {
                ret[i] = maxChar;
                i += 2;
            }
            // 2. 处理剩下的字符
            for(int j = 0; j < 26; j++)
            {
                if(hash[j] != 0 && (char)(j + 'a') != maxChar)
                {
                    while(hash[j]-- != 0)
                    {
                        if(i >= n)
                        {
```

```
51                              i = 1;
52                          }
53                          ret[i] = (char)(j + 'a');
54                          i += 2;
55                      }
56                  }
57              }
58              for(int j = 0; j < n; j++)
59              {
60                  System.out.print(ret[j]);
61              }
62          }
63      }
64  }
```

Day14

1. 乒乓球筐（哈希）

（题号：2600431）

1. **题目链接：** [编程题]乒乓球筐

2. **题目描述：**

> 题目描述：给定两个乒乓球筐，每个乒乓球的种类用不同的大写字母表示。请问第一个乒乓球筐内是否完全包含第二个乒乓球筐内所有乒乓球的种类和数量?
>
> 输入描述：输入两行，每行输入一个仅包含大写字母的字符串，代表乒乓球筐内的乒乓球。
> 两个 字符串的长度均不超过200000
>
> 输出描述：如果第一个乒乓球筐包含第二个的所有种类和数量，则输出"Yes"。否则输出"No"
>
> 补充说明:

```
示例1
输入：ABCBA
      AABB
输出：Yes
说明：


示例2
输入：CQC
      QQ
输出：No
说明：
```

3. **解法：**

算法思路：

简单查询题目，可以用哈希表帮助我们解决。

C++ 算法代码：

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

int main()
{
    string s1, s2;

    while(cin >> s1 >> s2) // 未知组数的输入
    {
        int hash[26] = { 0 };
        for(auto ch : s1) hash[ch - 'A']++;
        bool ret = true;
        for(auto ch : s2)
        {
            if(--hash[ch - 'A'] < 0)
            {
                ret = false;
                break;
            }
        }
        cout << (ret ? "Yes" : "No") << 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);
        while(in.hasNext()) // 未知组数的输入
        {
```

```
            char[] s1 = in.next().toCharArray();
            char[] s2 = in.next().toCharArray();
            int[] hash = new int[26];

            for(int i = 0; i < s1.length; i++)
            {
                hash[s1[i] - 'A']++;
            }
            boolean ret = true;
            for(int i = 0; i < s2.length; i++)
            {
                if(--hash[s2[i] - 'A'] < 0)
                {
                    ret = false;
                    break;
                }
            }
            System.out.println(ret ? "Yes" : "No");
        }
    }
}
```

2. 组队竞赛（贪心）

（题号：100347）

1. **题目链接：** [编程题]组队竞赛

2. **题目描述：**

> 牛牛举办了一次编程比赛,参加比赛的有3*n个选手,每个选手都有一个水平值a_i.现在要将这些选手进行组队,一共组成n个队伍,即每个队伍3人.牛牛发现队伍的水平值等于该队伍队员中第二高水平值。
> 例如:
> 一个队伍三个队员的水平值分别是3,3,3.那么队伍的水平值是3
> 一个队伍三个队员的水平值分别是3,2,3.那么队伍的水平值是3
> 一个队伍三个队员的水平值分别是1,5,2.那么队伍的水平值是2
> 为了让比赛更有看点,牛牛想安排队伍使所有队伍的水平值总和最大。
> 如样例所示:
> 如果牛牛把6个队员划分到两个队伍
> 如果方案为:
> team1:{1,2,5}, team2:{5,5,8}, 这时候水平值总和为7.
> 而如果方案为:
> team1:{2,5,8}, team2:{1,5,5}, 这时候水平值总和为10.

没有比总和为10更大的方案,所以输出10.

输入描述:

输入的第一行为一个正整数n(1 ≤ n ≤ 10^5)

第二行包括3*n个整数a_i(1 ≤ a_i ≤ 10^9),表示每个参赛选手的水平值.

输出描述:

输出一个整数表示所有队伍的水平值总和最大值.

示例1

输入

```
2
5 2 8 5 1 5
```

输出

```
10
```

3. **解法:**

算法思路:

小贪心。

最高分我们要不到，只能退而求其次拿到倒数第二个人的分数。然后一直递归这个策略~

C++ 算法代码:

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

```cpp

typedef long long LL;

const int N = 1e5 + 10;

int n;
LL arr[N * 3];

int main()
{
    cin >> n;
    for(int i = 0; i < 3 * n; i++) cin >> arr[i];
    sort(arr, arr + 3 * n);

    int pos = 3 * n - 2, count = 1;
    LL ret = 0;
    while(count++ <= n)
    {
        ret += arr[pos];
        pos -= 2;
    }

    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();
        int[] arr = new int[n * 3];
        for(int i = 0; i < n * 3; i++)
        {
            arr[i] = in.nextInt();
        }

```

```
        Arrays.sort(arr);

        int pos = n * 3 - 2, count = 1;
        long ret = 0;
        while(count++ <= n)
        {
            ret += arr[pos];
            pos -= 2;
        }
        System.out.println(ret);
    }
}
```

3. 删除相邻数字的最大分数（动态规划 - 线性 dp）

（题号：2362325）

1. **题目链接：** **DP25 删除相邻数字的最大分数**

2. **题目描述：**

> 题目描述：给定一个长度为 n 的仅包含正整数的数组，另外有一些操作，每次操作你可以选择数组中的任意一个元素 a_i ，同时数组中所有等于 $a_i - 1$ 和 $a_i + 1$ 的元素会被全部移除，同时你可以得到 a_i 分，直到所有的元素都被选择或者删除。
> 请你计算最多能得到多少分。
> 数据范围：数组长度满足 $1 \le n \le 10^5$ ，数组中的元素大小都满足 $1 \le a_i \le 10^4$
>
> 输入描述：第一行输入一个正整数 n 表示数组的长度
> 第二行输入 n 个数字表示数组的各个元素值。
>
> 输出描述：输出能得到的最大分数。
>
> 补充说明：
>
> 示例1
> 输入：2
> 1 2
> 输出：2
> 说明：直接选择元素 2，然后 1 被同时移除。
>
> 示例2
> 输入：3
> 1 2 3
> 输出：4
> 说明：先选择 3，同时 2 被移除，再选择 1，即得到 4 分。

3. **解法：**

算法思路：

打家劫舍拓展版~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 1e4 + 10;

int sum[N]; // sum[i] 表示 i 出现的总和
int n;
int f[N], g[N];

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

    for(int i = 1; i < N; i++)
    {
        f[i] = g[i - 1] + sum[i];
        g[i] = max(f[i - 1], g[i - 1]);
    }

    cout << max(f[N - 1], g[N - 1]) << 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
{
    static int N = (int)1e4 + 10;
    static int[] sum = new int[N]; // sum[i] 表示 i 这个数的总和
    static int[] f = new int[N];
    static int[] g = new int[N];
```

```
10
11      public static void main(String[] args)
12      {
13          Scanner in = new Scanner(System.in);
14          int n = in.nextInt();
15          int x = 0;
16          for(int i = 0; i < n; i++)
17          {
18              x = in.nextInt();
19              sum[x] += x;
20          }
21
22          for(int i = 1; i < N; i++)
23          {
24              f[i] = g[i - 1] + sum[i];
25              g[i] = Math.max(g[i - 1], f[i - 1]);
26          }
27
28          System.out.println(Math.max(f[N - 1], g[N - 1]));
29      }
30  }
```

Day15

1. 平方数（数学）

（题号：949014）

1. **题目链接：** 平方数
2. **题目描述：**

题目描述：牛妹是一个喜欢完全平方数的女孩子。
牛妹每次看到一个数 x，都想求出离 x 最近的完全平方数 y。
每次手算太麻烦，所以牛妹希望你能写个程序帮她解决这个问题。
形式化地讲，你需要求出一个正整数 y，满足 y 可以表示成 a^2（a 是正整数），使得 |x-y| 的值最小。可以证明这样的 y 是唯一的。

输入描述：一行，一个整数 $x\ (1 \le x \le 10^{12})$，表示牛妹询问的数。

输出描述：一行，一个整数 y，表示离 x 最近的完全平方数 y。

补充说明：

示例1
输入：5
输出：4
说明：

示例2
输入：7
输出：9
说明：

3. 解法：

算法思路：

判断一个数开根号之后左右两个数的平方，哪个最近即可。

C++ 算法代码：

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

using namespace std;

typedef long long LL;

int main()
{
    LL x;
    cin >> x;
    LL a = sqrt(x);
    LL x1 = a * a, x2 = (a + 1) * (a + 1);
    if(x - x1 < x2 - x) cout << x1 << endl;
    else cout << x2 << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        long x = in.nextLong();
        long a = (long)Math.sqrt(x);
        long x1 = a * a, x2 = (a + 1) * (a + 1);
        if(x - x1 < x2 - x) System.out.println(x1);
        else System.out.println(x2);
    }
}
```

2. 分组（枚举 + 二分）

（题号：2203267）

1. 题目链接： 分组

2. 题目描述：

题目描述：dd当上了宣传委员，开始组织迎新晚会，已知班里有n个同学，每个同学有且仅有一个擅长的声部，把同学们分成恰好m组，为了不搞砸节目，每一组里的同学都必须擅长同一个声部，当然，不同组同学擅长同一个声部的情况是可以出现的，毕竟一个声部也可以分成好几个part进行表演，但是他不希望出现任何一组的人过多，否则可能会导致场地分配不协调，也就是说，她希望人数最多的小组的人尽可能少，除此之外，对组内人员分配没有其他要求，她希望你告诉她，这个值是多少，如果无法顺利安排，请输出-1

输入描述：第一行两个数个数n,m(1≤m≤n≤100000)表示人数
接下来一行n个数,a[i](1≤a[i]≤n)表示第i个学生的擅长声部

输出描述：输出一个数，表示人数最多的小组的人数

补充说明：

示例1
输入：5 3
2 2 3 3 3
输出：2
说明：

3. 解法：

算法思路：

暴力枚举：

从小到大枚举所有可能的最大值，找到第一个符合要求的最大值。

二分优化枚举：

二分出符合要求的最大值。

C++ 算法代码：

```cpp
#include <iostream>
#include <unordered_map>

using namespace std;

int n, m;
unordered_map<int, int> cnt; // 统计每种声部的人数

bool check(int x) // 判断最多人数为 x 时，能否分成 m 组
{
    int g = 0; // 能分成多少组
    for(auto& [a, b] : cnt)
    {
        g += b / x + (b % x == 0 ? 0 : 1);
    }
    return g <= m;
}

int main()
{
    cin >> n >> m;
    int hmax = 0; // 统计声部最多的人数
    for(int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        hmax = max(hmax, ++cnt[x]);
    }

    int kinds = cnt.size();
    if(kinds > m) // 处理边界情况
    {
        cout << -1 << endl;
    }
    else
    {
//         // 暴力枚举
//         for(int i = 1; i <= hmax; i++) // 枚举所有的最多人数
//         {
//             if(check(i))
//             {
```

```
//                 cout << i << endl;
//                 break;
//             }
//         }
        // 二分解法
        int l = 1, r = hmax;
        while(l < r)
        {
            int mid = (l + r) / 2;
            if(check(mid)) r = mid;
            else l = mid + 1;
        }
        cout << l << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static int n, m;
    public static Map<Integer, Integer> hash = new HashMap<>(); // 统计每种声部的人数

    public static boolean check(int x) // 判断最多人数为 x 时，能否分成 m 组
    {
        int g = 0; // 统计能分成多少组
        for(int a : hash.values())
        {
            g += a / x + (a % x == 0 ? 0 : 1);
        }
        return g <= m;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); m = in.nextInt();
        int hmax = 0; // 所有声部的最大值
        for(int i = 0; i < n; i++)
```

```
        {
            int x = in.nextInt();
            hash.put(x, hash.getOrDefault(x, 0) + 1);
            hmax = Math.max(hmax, hash.get(x));
        }

        // 边界情况
        int kinds = hash.size();
        if(kinds > m)
        {
            System.out.println(-1);
        }
        else
        {
//              // 暴力解法
//              for(int i = 1; i <= hmax; i++)
//              {
//                  if(check(i))
//                  {
//                      System.out.println(i);
//                      break;
//                  }
//              }
            // 二分解法
            int l = 1, r = hmax;
            while(l < r)
            {
                int mid = (l + r) / 2;
                if(check(mid)) r = mid;
                else l = mid + 1;
            }
            System.out.println(l);
        }
    }
}
```

3. 【模板】拓扑排序（拓扑排序）

（题号：2369540）

1. **题目链接：** **AB13 【模板】拓扑排序**
2. **题目描述：**

题目描述：给定一个包含n个点m条边的有向无环图，求出该图的拓扑序。若图的拓扑序不唯一，输出任意合法的拓扑序即可。若该图不能拓扑排序，输出-1。

输入描述：第一行输入两个整数n, m（$1 \leq n, m \leq 2 \cdot 10^5$），表示点的个数和边的条数。
接下来的m行，每行输入两个整数u_i, v_i（$1 \leq u, v \leq n$），表示u_i到v_i之间有一条有向边。

输出描述：若图存在拓扑序，输出一行n个整数，表示拓扑序。否则输出-1。

补充说明：

示例1
输入：5 4
1 2
2 3
3 4
4 5
输出：1 2 3 4 5
说明：

3. **解法：**

算法思路：

拓扑排序裸题：

a. 建图；

b. 入队为 0 的点入队；

c. 来一次层序遍历即可。

C++ 算法代码：

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

using namespace std;

const int N = 2e5 + 10;

vector<vector<int>> edges(N); // edges[i] 表示 i 这个点所连接的边的信息
int in[N]; // 统计入度信息

int n, m;
queue<int> q;
vector<int> ret; // 记录最终结果

int main()
{
    cin >> n >> m;
    while(m--)
    {
```

```cpp
        int a, b;
        cin >> a >> b;
        edges[a].push_back(b); // 存储边的信息
        in[b]++; // 存储入度
    }

    for(int i = 1; i <= n; i++) // 把入度为 0 的点放进队列中
    {
        if(in[i] == 0)
        {
            q.push(i);
        }
    }

    while(q.size())
    {
        int a = q.front();
        q.pop();
        ret.push_back(a);

        for(auto b : edges[a])
        {
            if(--in[b] == 0)
            {
                q.push(b);
            }
        }
    }

    // 判断
    if(ret.size() == n)
    {
        for(int i = 0; i < n - 1; i++)
        {
            cout << ret[i] << " ";
        }
        cout << ret[n - 1]; // 测评会考虑最后一个元素的空格
    }
    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 int n, m;
    public static Map<Integer, List<Integer>> edges = new HashMap<>(); // 存储图
    public static int[] cnt; //统计入度信息

    public static void main(String[] args) throws IOException
    {
        n = in.nextInt(); m = in.nextInt();
        cnt = new int[n + 1];

        // 1. 建图
        for(int i = 0;  i < m; i++)
        {
            int a = in.nextInt(), b = in.nextInt();
            // a -> b
            cnt[b]++;
            if(!edges.containsKey(a))
            {
                edges.put(a, new ArrayList<>());
            }
            edges.get(a).add(b);
        }

        // 2. 拓扑排序
        Queue<Integer> q = new LinkedList<>();
        for(int i = 1; i <= n; i++)
        {
            if(cnt[i] == 0)
            {
                q.add(i);
            }
        }

        int[] ret = new int[n]; // 统计拓扑排序的结果
        int pos = 0;
```

```
43
44          while(!q.isEmpty())
45          {
46              int t = q.poll();
47              ret[pos++] = t;
48
49              for(int a : edges.getOrDefault(t, new ArrayList<>()))
50              {
51                  cnt[a]--;
52                  if(cnt[a] == 0)
53                  {
54                      q.add(a);
55                  }
56              }
57          }
58
59          if(pos == n)
60          {
61              // 输出结果：输出最后一个数的时候，不能有空格
62              for(int i = 0; i < n - 1; i++)
63              {
64                  out.print(ret[i] + " ");
65              }
66              out.println(ret[n - 1]);
67          }
68          else
69          {
70              out.println(-1);
71          }
72
73          out.close();
74      }
75  }
76
77  class Read // 自定义快速读入
78  {
79      StringTokenizer st = new StringTokenizer("");
80      BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
81      String next() throws IOException
82      {
83          while(!st.hasMoreTokens())
84          {
85              st = new StringTokenizer(bf.readLine());
86          }
87          return st.nextToken();
88      }
89
```

```
    String nextLine() throws IOException
    {
        return bf.readLine();
    }

    int nextInt() throws IOException
    {
        return Integer.parseInt(next());
    }

    long nextLong() throws IOException
    {
        return Long.parseLong(next());
    }

    double nextDouble() throws IOException
    {
        return Double.parseDouble(next());
    }
}
```

Day16

1. 字符串替换（模拟）

（题号：2590852）

1. **题目链接：** QR6 字符串替换

2. **题目描述：**

> 题目描述：请你实现一个简单的字符串替换函数。原串中需要替换的占位符为"%s",请按照参数列表的顺序一一替换占位符。若参数列表的字符数大于占位符个数。则将剩下的参数字符添加到字符串的结尾。
> 给定一个字符串，同时给定一个参数数组。
> 题目保证参数列表字符不少于占位符个数。
>
> 补充说明：保证原串除"%s"外均由大小写英文字母组成，同时长度小于等于500，参数数组长度不超过100。
>
> 示例1
> 输入："A%sC%sE",[B,D,F]
> 输出："ABCDEF"
> 说明：

3. **解法：**

算法思路：

简单模拟题~

C++ 算法代码：

```cpp
class StringFormat
{
public:
    string formatString(string A, int n, vector<char> arg, int m)
    {
        int j = 0;
        string ret;

        for(int i = 0; i < n; i++)
        {
            if(A[i] == '%')
            {
                if(i + 1 < n && A[i + 1] == 's')
                {
                    ret += arg[j++];
                    i++;
                }
                else
                {
                    ret += A[i];
                }
            }
            else
            {
                ret += A[i];
            }
        }

        while(j < m)
        {
            ret += arg[j++];
        }

        return ret;
    }
};
```

Java 算法代码：

```java
import java.util.*;

```

```java
public class StringFormat
{
    public String formatString(String A, int n, char[] arg, int m)
    {
        StringBuffer ret = new StringBuffer();
        char[] s = A.toCharArray();

        int j = 0;
        for(int i = 0; i < n; i++)
        {
            if(s[i] != '%')
            {
                ret.append(s[i]);
            }
            else
            {
                if(i + 1 < n && s[i + 1] == 's')
                {
                    ret.append(arg[j++]);
                    i++;
                }
                else
                {
                    ret.append(s[i]);
                }
            }
        }

        while(j < m)
        {
            ret.append(arg[j++]);
        }

        return ret.toString();
    }
}
```

2. 神奇数（数学）

（题号：100343）

1. **题目链接：** [编程题]神奇数

2. **题目描述：**

题目描述：给出一个区间[a, b]，计算区间内“神奇数”的个数。
神奇数的定义：存在不同位置的两个数位，组成一个两位数（且不含前导0），且这个两位数为质数。
比如：153，可以使用数字3和数字1组成13，13是质数，满足神奇数。同样153可以找到31和53也为质数，只要找到一个质数即满足神奇数。

输入描述：输入为两个整数a和b，代表[a, b]区间 (1 ≤ a ≤ b ≤ 100000)。

输出描述：输出为一个整数，表示区间内满足条件的整数个数。

补充说明：

示例1
输入：11 20
输出：6
说明：

3. 解法：

算法思路：

根据题意模拟就好了，注意一些细节问题，比如不要出现前导 0。

C++ 算法代码：

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

using namespace std;

int a, b;

bool isprim(int n) // 判断是否是质数
{
    if(n < 2) return false;
    for(int i = 2; i <= sqrt(n); i++) // 试除法
    {
        if(n % i == 0) return false;
    }
    return true;
}

int check(int n) // 判断是否是神奇数
{
    vector<int> num;
    while(n)
    {
        num.push_back(n % 10);
        n /= 10;
    }

```

```
    for(int i = 0; i < num.size(); i++) // 枚举十位数
    {
        for(int j = 0; j < num.size(); j++) // 枚举个位数
        {
            if(i != j && num[i] != 0)
            {
                if(isprim(num[i] * 10 + num[j]))
                {
                    return 1;
                }
            }
        }
    }
    return 0;
}

int main()
{
    cin >> a >> b;

    int ret = 0;
    for(int i = max(a, 10); i <= b; i++)
    {
        ret += check(i);
    }
    cout << ret << 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 boolean isprim(int x) // 判断是否是质数
    {
        if(x < 2) return false;
        for(int i = 2; i <= Math.sqrt(x); i++) // 试除法
        {
            if(x % i == 0) return false;
        }
```

```java
        return true;
    }

    public static int check(int x) // 判断是否是神奇数
    {
        int[] num = new int[10];
        int n = 0;
        while(x != 0)
        {
            num[n++] = x % 10;
            x /= 10;
        }

        for(int i = 0; i < n; i++) // 枚举十位数
        {
            for(int j = 0; j < n; j++) // 枚举个位数
            {
                if(num[i] != 0 && i != j)
                {
                    if(isprim(num[i] * 10 + num[j]))
                    {
                        return 1;
                    }
                }
            }
        }
        return 0;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt(), b = in.nextInt();

        int ret = 0;
        for(int i = Math.max(a, 10); i <= b; i++)
        {
            ret += check(i);
        }
        System.out.println(ret);
    }
}
```

3. DNA 序列（滑动窗口）

（题号：10055182）

1. 题目链接： HJ63 DNA序列

2. 题目描述：

> 题目描述：一个DNA序列由A/C/G/T四个字母的排列组合组成。G和C的比例（定义为GC-Ratio）是序列中G和C两个字母的总的出现次数除以总的字母数目（也就是序列长度）。在基因工程中，这个比例非常重要。因为高的GC-Ratio可能是基因的起始点。
> 给定一个很长的DNA序列，以及要求的最小子序列长度，研究人员经常会需要在其中找出GC-Ratio最高的子序列。
>
> DNA序列为 ACGT 的子串有: ACG , CG , CGT 等等，但是没有 AGT ， CT 等等
>
> 数据范围：字符串长度满足 $1 \leq n \leq 1000$ ，输入的字符串只包含 A/C/G/T 字母
>
> 输入描述：输入一个string型基因序列，和int型子串的长度
>
> 输出描述：找出GC比例最高的子串,如果有多个输出第一个的子串
>
> 补充说明：
>
> 示例1
> 输入：AACTGTGCACGACCTGA
> 5
> 输出：GCACG
> 说明：

3. 解法：

算法思路：

长度固定的滑动窗口。

C++ 算法代码：

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

using namespace std;

string s;
int x;

int main()
{
    cin >> s >> x;

    int begin = -1; // 标记结果的起始位置
    int maxCount = 0; // 存储之前的窗口里面 C + G 的个数
    int count = 0; // 统计窗口内 C + G
    int left = 0, right = 0, n = s.size();

```

```
    while(right < n)
    {
        if(s[right] == 'C' || s[right] == 'G') count++;
        while(right - left + 1 > x)
        {
            if(s[left] == 'C' || s[left] == 'G') count--;
            left++;
        }
        if(right - left + 1 == x)
        {
            if(count > maxCount)
            {
                begin = left;
                maxCount = count;
            }
        }
        right++;
    }

    cout << s.substr(begin, x) << 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);
        char[] s = in.next().toCharArray();
        int x = in.nextInt();

        int begin = 0; // 标记结果的起始位置
        int maxCount = 0; // 统计以前窗口内 C + G 的最大值
        int left = 0, right = 0, n = s.length;
        int count = 0; // 统计窗口内 C + G
        while(right < n)
        {
            if(s[right] == 'C' || s[right] == 'G') count++;
```

```
            while(right - left + 1 > x)
            {
                if(s[left] == 'C' || s[left] == 'G') count--;
                left++;
            }
            if(right - left + 1 == x)
            {
                if(count > maxCount)
                {
                    begin = left;
                    maxCount = count;
                }
            }
            right++;
        }

        for(int i = begin; i < begin + x; i++)
        {
            System.out.print(s[i]);
        }
    }
}
```

Day17

1. 小乐乐改数字（模拟）

（题号：632017）

1. **题目链接：** **BC45 小乐乐改数字**
2. **题目描述：**

题目描述：小乐乐喜欢数字，尤其喜欢0和1。他现在得到了一个数，想把每位的数变成0或1。如果某一位是奇数，就把它变成1，如果是偶数，那么就把它变成0。请你回答他最后得到的数是多少。

输入描述：输入包含一个整数n ($0 \le n \le 10^9$)

输出描述：输出一个整数，即小乐乐修改后得到的数字。

补充说明：

示例1
输入：222222
输出：0
说明：

示例2
输入：123
输出：101
说明：

3. 解法：

算法思路：

小技巧：题目虽然说输入一个数，但是我们可以把它当成字符串读进来，直接操作数字字符串岂不是美滋滋~

C++ 算法代码：

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

using namespace std;

string s;

int main()
{
    cin >> s;
    for(int i = 0; i < s.size(); i++)
    {
        if(s[i] % 2 == 0) s[i] = '0';
        else s[i] = '1';
    }

    cout << stoi(s) << endl; // 自动处理前导零

    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);
        char[] s = in.next().toCharArray();

        for(int i = 0; i < s.length; i++)
        {
            if(s[i] % 2 == 0) s[i] = '0';
            else s[i] = '1';
        }

        // 处理一下前导零
        int i = 0;
        while(i < s.length - 1 && s[i] == '0') i++;
        while(i < s.length)
        {
            System.out.print(s[i++]);
        }
    }
}
```

2. 十字爆破（预处理 + 模拟）

（题号：955384）

1. **题目链接：** 十字爆破
2. **题目描述：**

题目描述：牛牛在玩一个游戏：
一共有n行m列共n*m个方格，每个方格中有一个整数。
牛牛选择一个方格，可以得到和这个方格同行、同列的所有数之和的得分。
例如：对于一个2*2的方格：
1 2
3 4
牛牛选择每个方格的得分如下：
6 7
8 9
因为1+2+3=6，1+2+4=7，1+3+4=8，2+3+4=9。
现在牛牛想知道下一步选择每个格子的得分情况，你可以帮帮他吗?

输入描述：第一行有两个正整数 n 和 m ，代表方格的行数和列数。（$1 \leq n * m \leq 1000000$）
接下来的 n 行，每行有 m 个数 a_{ij}，代表每个方格中的整数。（$-10^9 \leq a_{ij} \leq 10^9$）

输出描述：输出 n 行 m 列整数，分别代表选择每个位置方格的得分情况。

补充说明：本题输入和输出数据量较大，尽量使用scanf或更快的IO方式~

示例1
输入：2 2
1 2
3 4
输出：6 7
8 9
说明：

3. 解法：

算法思路：

模拟即可，但是由于数据量过大，我们可以提前把每一行以及每一列的和存起来，方便统计总和。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 1e6 + 10;

typedef long long LL;

LL n, m;
LL row[N], col[N];

int main()
{
    scanf("%ld %ld", &n, &m);
    LL arr[n][m];
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
```

```
        {
            scanf("%ld", &arr[i][j]);
            row[i] += arr[i][j];
            col[j] += arr[i][j];
        }
    }

    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            printf("%ld ", row[i] + col[j] - arr[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Java 算法代码：

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

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

    public static void main(String[] args) throws IOException
    {
        int n = in.nextInt(), m = in.nextInt();
        long[][] arr = new long[n][m];
        long[] row = new long[n];
        long[] col = new long[m];

        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                arr[i][j] = in.nextInt();
                row[i] += arr[i][j];
                col[j] += arr[i][j];
```

```
            }
        }

        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                System.out.print((row[i] + col[j] - arr[i][j]) + " ");
            }
            System.out.println("");
        }

        //out.close();
    }
}

class Read
{
    StringTokenizer st = new StringTokenizer("");
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    String next() throws IOException
    {
        while(!st.hasMoreTokens())
        {
            st = new StringTokenizer(bf.readLine());
        }
        return st.nextToken();
    }

    String nextLine() throws IOException
    {
        return bf.readLine();
    }

    int nextInt() throws IOException
    {
        return Integer.parseInt(next());
    }

    long nextLong() throws IOException
    {
        return Long.parseLong(next());
    }

    double nextDouble() throws IOException
    {
        return Double.parseDouble(next());
```

```
70      }
71  }
```

3. 比那名居的桃子（前缀和 / 滑动窗口）

（题号：1928660）

1. **题目链接：** 比那名居的桃子

2. **题目描述：**

> 题目描述：小红有一天看到了一只桃子，由于桃子看上去就很好吃，小红很想把它吃掉。
> 已知吃下桃子后，每天可以获得 a_i 的快乐值，但是每天会获得 b_i 的羞耻度。桃子的持续效果一共为 k 天。
> 小红想知道，自己在哪一天吃下果实，可以获得尽可能多的快乐值？
>
> 如果有多个答案获得的快乐值相等，小红希望获得尽可能少的羞耻度。
> 如果有多个答案的快乐值和羞耻度都相等，由于小红实在太想吃桃子了，她希望尽可能早的吃下桃子。
>
> 输入描述：第一行有两个正整数 n 和 k ，分别代表桃子的有效期总天数，以及桃子效果的持续天数。
> （桃子的有效期是指，无论桃子在何时服用，桃子的特殊效果只在这段时间之内有效）
> 第二行有 n 个正整数 a_i ，分别代表每天可以获得的快乐值。
> 第三行有 n 个正整数 b_i ，分别代表每天可以获得的羞耻度。
>
> $1 \leq k \leq n \leq 10^5$
> $1 \leq a_i, b_i \leq 10^9$
>
> 输出描述：一个正整数，代表小红是第几天吃下桃子的。
>
> 补充说明：

> 示例1
> 输入：4 2
> 3 5 1 7
> 4 6 5 1
> 输出：3
> 说明：选择在第三天吃下桃子，可以获得8快乐值和6羞耻度。

3. **解法：**

算法思路：

滑动窗口的思想：

由题意得，我们是要枚举所有大小为 k 的子数组，并且求出这段子数组中快乐值和羞耻度之和。因此，可以利用滑动窗口的思想，用两个变量维护大小为 k 的窗口的总和，并且不断更新即可。

前缀和思想：

这个就比较简单了，先预处理出来快乐值和羞耻度的前缀和数组，然后枚举的过程中直接求出一段区间的和即可。但是相比较于滑动窗口的思想，会有空间消耗。

C++ 算法代码：

```cpp
// 滑动窗口
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 1e5 + 10;

LL n, k;
LL h[N], s[N];

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

    LL left = 0, right = 0;
    LL hSum = 0, sSum = 0, hMax = 0, sMin = 0, begin = 0;

    while(right <= n)
    {
        hSum += h[right];
        sSum += s[right];
        while(right - left + 1 > k)
        {
            hSum -= h[left];
            sSum -= s[left];
            left++;
        }
        if(right - left + 1 == k)
        {
            if(hSum > hMax)
            {
                begin = left;
                hMax = hSum;
                sMin = sSum;
            }
            else if(hSum == hMax && sSum < sMin)
            {
                begin = left;
                hMax = hSum;
                sMin = sSum;
```

```
            }
        }
        right++;
    }

    cout << begin << 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(), k = in.nextInt();
        long[] a = new long[n + 1];
        long[] b = new long[n + 1];
        for(int i = 1; i <= n; i++) a[i] = in.nextLong();
        for(int i = 1; i <= n; i++) b[i] = in.nextLong();

        int left = 1, right = 1;
        long hSum = 0, sSum = 0, hMax = 0, sMin = 0, begin = 0;
        while(right <= n)
        {
            // 进窗口
            hSum += a[right]; sSum += b[right];

            // 出窗口
            while(right - left + 1 > k)
            {
                hSum -= a[left]; sSum -= b[left];
                left++;
            }

            // 更新结果
            if(right - left + 1 == k)
            {
```

```
                if(hSum > hMax)
                {
                    begin = left;
                    hMax = hSum;
                    sMin = sSum;
                }
                else if(hSum == hMax && sSum < sMin)
                {
                    begin = left;
                    sMin = sSum;
                }
            }

            right++;
        }

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

Day18

1. 压缩字符串（一）（字符串 + 模拟）

（题号：2049875）

1. **题目链接：** NC101 压缩字符串(一)
2. **题目描述：**

> 题目描述：利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。比如，字符串aabcccccaaa会变为a2bc5a3。
> 1.如果只有一个字符，1不用写
> 2.字符串中只包含大小写英文字母（a至z）。
>
> 数据范围:
> 0<=字符串长度<=50000
>
> 要求：时间复杂度O(N)
>
> 补充说明:

```
示例1
输入："aabcccccaaa"
输出："a2bc5a3"
说明:


示例2
输入："shopeew"
输出："shope2w"
说明:
```

3. **解法：**

算法思路：

利用双指针模拟即可。

C++ 算法代码：

```
class Solution
{
public:
    string compressString(string param)
    {
        string ret;
        int left = 0, right = 0, n = param.size();
        while(left < n)
        {
            while(right + 1 < n && param[right] == param[right + 1]) right++;
            int len = right - left + 1;
            ret += param[left];
            if(len > 1)
            {
                ret += to_string(len);
            }
            left = right + 1;
            right = left;
        }
        return ret;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public String compressString (String param)
    {
        StringBuffer ret = new StringBuffer();
        char[] s = param.toCharArray();
        int left = 0, right = 0, n = s.length;
```

```
10
11          while(left < n)
12          {
13              while(right + 1 < n && s[right + 1] == s[right]) right++;
14              int len = right - left + 1;
15              ret.append(s[left]);
16              if(len > 1)
17              {
18                  ret.append(len);
19              }
20              left = right + 1;
21              right = left;
22          }
23          return ret.toString();
24      }
25  }
```

2. chika和蜜柑（排序 / topK）

（题号：374977）

1. **题目链接：** chika和蜜柑

2. **题目描述：**

> 题目描述：chika很喜欢吃蜜柑。每个蜜柑有一定的酸度和甜度，chika喜欢吃甜的，但不喜欢吃酸的。
> 一共有n个蜜柑，chika吃k个蜜柑，将获得所吃的甜度之和与酸度之和。chika想获得尽可能大的甜度总和。如果有多种方案，她希望总酸度尽可能小。
> 她想知道，最终的总酸度和总甜度是多少?
>
> 输入描述：第一行有两个正整数n和k，分别代表蜜柑总数和chika吃的蜜柑数量。(1≤k≤n≤200000)
> 第二行有n个正整数ai，分别代表每个蜜柑的酸度。（1≤ai≤1e9）
> 第二行有n个正整数bi，分别代表每个蜜柑的甜度。（1≤bi≤1e9）
>
> 输出描述：两个正整数，用空格隔开。分别表示总酸度和总甜度。
>
> 补充说明：

```
示例1
输入：3 2
      1 3 4
      2 2 5
输出：5 7
说明：选择1号和3号蜜柑，总酸度为5，总甜度为7，为最优解。
```

3. **解法：**

算法思路：

排序：

将每个橘子按照甜度由高到低排序，相同甜度的橘子按照酸度由低到高排序。

然后提取排序后的前 k 个橘子就好了。

C++ 算法代码：

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

using namespace std;

const int N = 2e5 + 10;

typedef pair<int, int> PII; // <酸度，甜度>

PII arr[N];
int n, k;

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

    sort(arr, arr + n, [&](const PII& a, const PII& b)
         {
             if(a.second != b.second) return a.second > b.second;
             else return a.first < b.first;
         });

    long long s = 0, t = 0;
    for(int i = 0; i < k; i++)
    {
        s += arr[i].first;
        t += arr[i].second;
    }

    cout << s << " " << t << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

class Orange
{
    int a; // 酸度
    int b; // 甜度
    Orange(){}

    Orange(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
}

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), k = in.nextInt();
        Orange[] o = new Orange[n];
        for(int i = 0; i < n; i++)
        {
            o[i] = new Orange();
            o[i].a = in.nextInt();
        }
        for(int i = 0; i < n; i++)
        {
            o[i].b = in.nextInt();
        }

        // 排序
        Arrays.sort(o, (x, y) ->
        {
            if(x.b == y.b) return x.a - y.a;
            return y.b - x.b;
        });

        // 提取结果
        long x = 0, y = 0;
        for(int i = 0; i < k; i++)
        {
            x += o[i].a;
            y += o[i].b;
        }

```

```
48          System.out.println(x + " " + y);
49      }
50  }
```

3. 01 背包（动态规划 - 01背包）

（题号：1266316）

1. **题目链接：** NC145 01背包
2. **题目描述：**

> 题目描述：已知一个背包最多能容纳体积之和为v的物品
>
> 现有 n 个物品，第 i 个物品的体积为 v_i , 重量为 w_i
>
> 求当前背包最多能装多大重量的物品?
>
> 数据范围： $1 \leq v \leq 1000$ ， $1 \leq n \leq 1000$ ， $1 \leq v_i \leq 1000$ ， $1 \leq w_i \leq 1000$
>
> 进阶： $O(n \cdot v)$
>
> 补充说明:
>
> 示例1
> 输入：10,2,[[1,3],[10,4]]
> 输出：4
> 说明：第一个物品的体积为1，重量为3，第二个物品的体积为10，重量为4。只取第二个物品可以达到最优方案，取物重量为4
>
> 示例2
> 输入：10,2,[[1,3],[9,8]]
> 输出：11
> 说明：两个物品体积之和等于背包能装的体积，所以两个物品都取是最优方案

3. **解法：**

算法思路：

01 背包模板题：

1. 状态表示：

   `dp[i][j]` 表示从前 `i` 个物品中挑选，总体积不超过 `j` 的情况下，最大重量是多少。

2. 状态转移方程：

   根据「最后一步」的状况，来分情况讨论：

   i. 不选第 `i` 个物品：相当于就是去前 `i - 1` 个物品中挑选，并且总体积不超过 `j` 。此时 `dp[i][j] = dp[i - 1][j]` ；

ii. 选择第 `i` 个物品：那么我就只能去前 `i - 1` 个物品中，挑选总体积不超过 `j - v[i]` 的物品。此时 `dp[i][j] = dp[i - 1][j - v[i]] + w[i]`。但是这种状态不一定存在，因此需要特判一下。

综上，状态转移方程为：`dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - v[i]] + w[i])`。

C++ 算法代码：

```
class Solution
{
    int dp[1010] = { 0 };
public:
    int knapsack(int V, int n, vector<vector<int> >& vw)
    {
        for(int i = 0; i < n; i++)
        {
            for(int j = V; j >= vw[i][0]; j--)
            {
                dp[j] = max(dp[j], dp[j - vw[i][0]] + vw[i][1]);
            }
        }
        return dp[V];
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    int[] dp = new int[1010];
    public int knapsack (int V, int n, int[][] vw)
    {
        for(int i = 0; i < n; i++)
        {
            for(int j = V; j >= vw[i][0]; j--)
            {
                dp[j] = Math.max(dp[j], dp[j - vw[i][0]] + vw[i][1]);
            }
```

```
        }
        return dp[V];
    }
}
```