讲评

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.15

启航舱
中小学拔尖创新人才培养中心

目录

01

A

TEACHING
COURSEWARE

TEACH

A

题目描述

给定 T 个 6 位正整数 $N_1, N_2, ..., N_T$。

对于每个整数 N_i，请判断其是否满足以下所有条件：

1. 在 N_i 的各位数字中，数字 `1` 恰好出现 **1 次**。
2. 在 N_i 的各位数字中，数字 `2` 恰好出现 **2 次**。
3. 在 N_i 的各位数字中，数字 `3` 恰好出现 **3 次**。

若同时满足以上所有条件，该整数为「有效数字」；否则为「无效数字」。

输入格式

输入从标准输入按以下格式给出：

> 第一行：一个整数 T（表示测试用例的个数，$1 \le T \le 100$）
> 接下来 T 行：每行一个 6 位正整数 N_i（满足 $100000 \le N_i \le 999999$）

输出格式

对于每个测试用例，输出一行结果：

- 若 N_i 是有效数字，输出 `Yes`；
- 否则输出 `No`。

A

输入输出样例

输入样例

Copy

```
5
123233
123234
323132
500000
233123
```

输出样例

Copy

```
Yes
No
Yes
No
Yes
```

A

题目要求判断 6 位整数是否满足 “1 出现 1 次、2 出现 2 次、3 出现 3 次”。由于 6 位数字的总个数（6）恰好等于 1+2+3，因此该整数的数字只能由 1、2、3 组成，且各自出现次数严格匹配条件。

拆位统计：对每个 6 位整数，通过 “取余 10” 获取当前个位数字，通过 “整除 10” 移除当前个位数字，循环 6 次即可拆分所有位。

条件判断：用三个变量分别统计 1、2、3 的出现次数，最终检查是否满足 “1 出现 1 次、2 出现 2 次、3 出现 3 次”，满足则输出Yes，否则输出No

A

```cpp
#include <iostream>
using namespace std;
int main() {
    int T;
    cin >> T;
    while (T--) {
        int n;
        cin >> n;
        int a = 0, b = 0, c = 0; // 分别统计1、2、3的出现次数
        // 拆位统计（6位数字，循环6次）
        for (int i = 0; i < 6; ++i) {
            int digit = n % 10; // 取当前个位数字
            if (digit == 1) a++;
            else if (digit == 2) b++;
            else if (digit == 3) c++;
            n /= 10; // 移除当前个位数字
        }
        // 判断是否满足所有条件
        if (a == 1 && b == 2 && c == 3) {
            cout << "Yes\n";
        } else {
            cout << "No\n";
        }
    }
    return 0;
}
```

02

B

TEACHING
COURSEWARE

TEACH

B

题目描述

> 求九九乘法表中的 81 个整数（得数）中非 X 的总和。

有一个由纵 9 格、横 9 格的网格。
网格中每个格都写有一个整数，网格上起第 i 行、左起第 j 列的格写有 $i \times j$ 。
给定整数 X。求网格中写有的 81 个整数中非 X 整数之和。注意不同格中相同的数字要分别相加。

输入格式

输入按照如下格式由标准输入给出。

> X

输出格式

输出网格中写有的 81 个整数中非 X 整数之和。

输入输出样例 #1

输入 #1

```
1
```

Copy

输出 #1

```
2024
```

Copy

B

说明/提示

约定

- X 为 1 以上 81 以下的整数。

样例解释 1

网格中唯一写有 1 的格是在上起第 1 行和左起第 1 列。将非 1 的所有整数相加，和为 2024。

样例解释 2

网格中没有写有 11 的格。因此答案是 81 个整数的和 2025。

限制与约定

- 时间限制：$1s$
- 空间限制：$512MB$

B

计算九九乘法表总和：先算出 9 行 9 列中所有 i*j（i、j 从 1 到 9）的总和。通过双重循环遍历每行每列，累加所有乘积即可。

统计 X 出现的次数：再次遍历乘法表，统计值等于 X 的乘积出现的次数（记为 cnt）。

计算结果：用乘法表的总和减去 X * cnt，得到所有非 X 整数的总和（若 X 未出现，cnt=0，结果就是总和）。

B


```cpp
#include <iostream>
using namespace std;
int main() {
    int X;
    cin >> X;
    int total = 0;  // 乘法表所有数的总和
    // 第一步：计算乘法表总和
    for (int i = 1; i <= 9; ++i) {
        for (int j = 1; j <= 9; ++j) {
            total += i * j;
        }
    }
    int cnt = 0;  // X 出现的次数
    // 第二步：统计 X 出现的次数
    for (int i = 1; i <= 9; ++i) {
        for (int j = 1; j <= 9; ++j) {
            if (i * j == X) {
                cnt++;
            }
        }
    }
    // 第三步：非 X 的总和 = 总和 - X*出现次数
    cout << total - X * cnt << endl;
    return 0;
}
```

03

C

TEACHING
COURSEWARE

TEACH

C

题目描述

给定不超过 100 组测试用例，每组测试用例包含整数 K 和两个仅由小写英文字母组成的字符串 S、T。

对于每组测试用例，你可以对字符串 S 进行 0 次到 K 次以下操作（操作类型可任选），判断是否能将 S 变为 T：

- **插入**：在 S 的任意位置（包括首尾）插入任意一个小写英文字母。
- **删除**：删除 S 中的任意一个字符。
- **修改**：将 S 中的任意一个字符修改为另一个小写英文字母。

注意：题目限制 K 的取值恒为 1（即最多只能执行 1 次操作）。

输入格式

输入通过标准输入给出，格式如下：

> 第一行：一个整数 T（表示测试用例的个数，$1 \le T \le 100$）
> 接下来 T 行：每行包含一个整数 K 和两个字符串 S、T（三者用空格分隔）

输出格式

对于每组测试用例，若能在不超过 K 次操作内将 S 变为 T，输出 `Yes`；否则输出 `No`。

数据限制

- 字符串 S、T 的长度均为 $1 \le \text{len}(S), \text{len}(T) \le 1000$
- K 的取值恒为 1（输入保证此条件）
- 所有输入的字符串仅包含小写英文字母

C

输入输出样例

输入

Copy

```
6
1 abc agc
1 abc awtf
1 abc ac
1 back black
1 same same
1 leap read
```

输出

Copy

```
Yes
No
Yes
Yes
Yes
No
```

C

样例解释

1. **第 1 组**：将 `abc` 的第 2 个字符 `b` 修改为 `g`（1 次修改操作），得到 `agc` → 输出 `Yes`。
2. **第 2 组**：`abc`（长度 3）与 `awtf`（长度 4），长度差 1，但插入/删除/修改任意 1 次均无法得到目标字符串 → 输出 `No`。
3. **第 3 组**：删除 `abc` 的第 2 个字符 `b`（1 次删除操作），得到 `ac` → 输出 `Yes`。
4. **第 4 组**：在 `back` 的第 1 个与第 2 个字符之间插入 `l`（1 次插入操作），得到 `black` → 输出 `Yes`。
5. **第 5 组**：初始时 $S = T$（0 次操作），满足要求 → 输出 `Yes`。
6. **第 6 组**：`leap` 与 `read` 长度相同，但存在 4 处字符不同，1 次修改无法完成变换 → 输出 `No`。

限制与约定

- 时间限制： $1s$
- 空间限制： $512MB$ 特殊性质： 测试点1：无

测试点2,3,4: $|S| = |T| - 1$

测试点5,6,7: $|S| = |T|$

测试点8,9,10: $|S| = |T| + 1$

C

直接相等（无需操作）：

若S == T，直接满足条件（0 次操作≤1 次）。

长度差超过 1（无法实现）：

插入 / 删除 1 次只能改变长度 1，若|len(S)-len(T)|>1，1 次操作不可能让两者长度匹配，直接排除。

长度相同（仅需检查修改）：此时只能通过修改 1 个字符实现转换。统计S与T对应位置的字符差异数，若差异数≤1，则可行；否则不可行。

长度差 1（检查插入 / 删除）：插入和删除是互逆操作（如S删 1 个字符 = T插 1 个字符），只需统一处理：

让S为较长字符串，T为较短字符串。

双指针遍历两字符串，遇到第一个不匹配时，跳过较长串的当前字符（模拟删除）；若后续仍有不匹配，说明无法通过 1 次操作实现。

C


```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int __tcase;
    cin >> __tcase;  // 读取测试用例数量
    while (__tcase--) {  // 处理每组测试用例
        int k;
        string s, t;
        cin >> k >> s >> t;  // k恒为1，无需额外判断
```

C


```
// 情况1：S和T完全相同（0次操作）
if (s == t) {
    cout << "Yes\n";
    continue;
}
int len_s = s.size(), len_t = t.size();
// 情况2：长度差超过1，1次操作无法处理
if (abs(len_s - len_t) > 1) {
    cout << "No\n";
    continue;
}
```

C

```
// 情况3：长度相同，检查是否仅需1次修改
if (len_s == len_t) {
    int cnt = 0;  // 统计字符差异数
    for (int i = 0; i < len_s; ++i) {
        if (s[i] != t[i]) {
            cnt++;
            // 差异数超过1，提前退出（无需继续统计）
            if (cnt > 1) break;
        }
    }
    // 差异数≤ 1则可行
    if(cnt <= 1)
    {
        cout<<"Yes\n";
    }
    else cout<<"No\n";
    continue;
}
```

C

```
        if (len_s < len_t) swap(s, t);
        bool fg = true;
        // i遍历较长串s，j遍历较短串t
        for (int i = 0, j = 0; i < s.size(); ++i, ++j) {
            if (s[i] != t[j]) {
                // 第一次不匹配：跳过s的当前字符（模拟删除），j不移动
                if (i == j) {  // 确保只跳过1次（i比j多走1步）
                    j--;  // j保持原位，下一轮继续对比t[j]和s[i+1]
                    continue;
                }
                // 第二次不匹配：无法通过1次操作解决
                fg = false;
                break;
            }
        }
        if(fg == true)
        {
            cout<<"Yes\n";
        }
        else cout<<"No\n";
    }

    return 0;
}
```

04

D

TEACHING
COURSEWARE

TEACH

D

题目背景

olinr 是一个资深二刺螈，这天他遇到了大卫哥，大卫哥给他出了一道题，只要答对了就能让 olinr 进入 二次元世界。

题目描述

David 发现 olinr 刚好在学九九乘法表，于是他立刻就想到了二维数组，如果用 $f[i][j]$ 表示第 i 行 $\times$第 j列的结果，那么乘法表的每一个式子都可以用这个式子表示：$i*j=f[i][j]$ 于是他直接思考一个 $n*m$ 的二维数组 $f[i][j]=i*j$ 相关的问题 现在他问 olinr，对于所有的$f[i][j]$，第 k小的值是多少 显然 olinr 并不会，并将问题甩给了机智的你

输入格式

一行三个用空格分隔的整数n,m,k

输出格式

一行一个整数，表示第 k 小的值

输入样例1

```
4 4 7
```

Copy

输出样例1

```
4
```

Copy

D

样例1解释

一个 4×4 的二维数组的值如下图所示:

行号 \ 列号	1（列 1）	2（列 2）	3（列 3）	4（列 4）
1（行 1）	1×1=1	1×2=2	1×3=3	1×4=4
2（行 2）	2×1=2	2×2=4	2×3=6	2×4=8
3（行 3）	3×1=3	3×2=6	3×3=9	3×4=12
4（行 4）	4×1=4	4×2=8	4×3=12	4×4=16

显而易见，这 16 个数字第 7 小的是 4

数据范围

对于前 20% 的数据，满足$1 \leq n,\ m \leq 500$ 。

对于额外 20% 的数据，满足 $n = 1$。

对于 100% 的数据，满足 $1 \leq n, m \leq 5 \times 10^5, 1 \leq k \leq n * m$。

D

生成所有元素：通过双重循环遍历乘法表的每一行i（1~n）和每一列j（1~m），计算i*j并存储到数组中。

排序数组：对存储所有元素的数组按从小到大排序。

取第 k 小值：排序后数组的第k-1个索引（数组从 0 开始）即为第 k 小的值。

D

二分循环：
计算中间值mid = (l + r) / 2。
调用chk(mid)计算乘法表中≤mid 的元素个数 sum：
若 sum ≥ k：说明第 k 小的元素≤mid，记录 mid 为候选答案，并缩小右边界（r=mid-1）。
若 sum <k：说明第 k 小的元素> mid，缩小左边界（l=mid+1）。
终止条件：当 l > r 时，循环结束，此时记录的 ans 即为第 k 小的元素。

D

在n行m列的乘法表中，第i行的元素是：
i×1，i×2，i×3，...，i×m
这些元素的特点是：每个元素都是i与列号j的乘积。

例如第3行（i=3），元素为3×1=3、3×2=6、3×3=9、…、3×m。

如何统计第i行中≤x的元素个数？
对于第i行，我们需要找到「满足i×j ≤ x的j的数量」。
对不等式i×j ≤ x变形可得：j ≤ x/i（i是正整数，不等号方向不变）。

这意味着：

当j取1，2，...，floor(x/i)时，i×j的值都≤x（floor(x/i)表示对x/i向下取整）。

但j的最大值不能超过乘法表的列数m（因为第i行最多只有m个元素）。

因此，第i行中≤x的元素个数为：min(m，floor(x/i))。

D


```cpp
#include <iostream>
using namespace std;
#define int long long
int n, m, k;
// 判断x是否大于等于第k小的元素：计算<=x的元素个数，若>=k则返回true
bool chk(int x) {
    int sum = 0;
    // 遍历每一行i，计算该行中<=x的元素个数
    for (int i = 1; i <= n; ++i) {
        // 第i行元素为i× 1, i× 2, ..., i× m，其中<=x的最大j为min(m, x/i)
        sum += min(m, x / i);
    }
    return sum >= k;
}
```

D

```cpp
int main() {
    cin >> n >> m >> k;
    int l = 1, r = n * m;  // 二分查找的左右边界
    int ans;  // 存储第k小的元素
    while (l <= r) {
        int mid = (l + r) / 2;  // 中间值
        if (chk(mid)) {
            // 若<=mid的元素个数>=k，说明答案可能是mid或更小的值
            ans = mid;
            r = mid - 1;
        } else {
            // 若<=mid的元素个数<k，说明答案一定大于mid
            l = mid + 1;
        }
    }
    cout << ans << endl;
    return 0;
}
```