

t714

[华为OD机试2025 B卷 - 通过软盘拷贝文件 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025 B卷 - 计算快递主站点 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025B卷 - 找城市 / 城市聚集度 \(C++ & Python & JAVA & JS & GO\)_华为od上机考试2025年a卷-CSDN博客](#)

[华为od机试2025 B卷 - 字符串化繁为简 \(C++ & Python & JAVA & JS & GO\)_华为机试 字符串化繁为简-CSDN博客](#)

[华为OD机试 2025 B卷 - 最佳植树距离 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 - 租车骑绿岛 \(C++ & Python & JAVA & JS & GO\)_计算最少需要多少辆自行车才能满足条件-CSDN博客](#)

[华为OD机试2025 B卷 - 矩阵元素的边界值 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025 B卷 - 非严格递增连续数字序列 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025 B卷 - 数列描述 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025B卷 - 完全数计算 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025 B卷 - 端口合并 \(C++ & Python & JAVA & JS & GO\)_华为od说考牛客网b卷-CSDN博客](#)

[华为od机试 2025 B卷 - 分积木 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025 B卷 - 服务失效判断 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试2025 B卷 - 通过软盘拷贝文件 (C++ & Python & JAVA & JS & GO)-CSDN博客

通过软盘拷贝文件

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

有一名科学家想要从一台古董电脑中拷贝文件到自己的电脑中加以研究。

但此电脑除了有一个3.5寸软盘驱动器以外, 没有任何手段可以将文件拷贝出来, 而且只有一张软盘可以使用。

因此这一张软盘是唯一可以用来拷贝文件的载体。

科学家想要尽可能多地将计算机中的信息拷贝到软盘中, 做到软盘中文件内容总大小最大。

已知该软盘容量为1474560字节。文件占用的软盘空间都是按块分配的, 每个块大小为512个字节。一个块只能被一个文件使用。拷贝到软盘中的文件必须是完整的, 且不能采取任何压缩技术。

输入描述

第1行为一个整数N, 表示计算机中的文件数量。 $1 \leq N \leq 1000$

接下来的第2行到第N+1行(共N行), 每行为一个整数, 表示每个文件的大小 S_i , 单位为字节。 $0 \leq i <$

$N, 0 \leq S_i$

备注

为了充分利用软盘空间, 将每个文件在软盘上占用的块记录到本子上。即真正占用软盘空间的只有文件内容本身。

输出描述

科学家最多能拷贝的文件总大小

用例1

输入

▼		Plain Text
1	3	
2	737270	
3	737272	
4	737288	

输出

▼		Plain Text
1	1474542	

说明

3个文件中，每个文件实际占用的大小分别为737280字节、737280字节、737792字节。
只能选取前两个文件，总大小为1474542字节。虽然后两个文件总大小更大且未超过1474560字节，但因为实际占用的大小超过了1474560字节，所以不能选后两个文件。

用例2

输入

▼		Plain Text
1	6	
2	400000	
3	200000	
4	200000	
5	200000	
6	400000	
7	400000	

输出

▼		Plain Text
1	1400000	

说明

从6个文件中，选择3个大小为400000的文件和1个大小为200000的文件，得到最大总大小为1400000。

也可以选择2个大小为400000的文件和3个大小为200000的文件，得到的总大小也是1400000

题解

思路：01背包问题 模板题，在这个这个题中重量和价值都是 文件大小。

1. 初始定义 `dp[]` 一维滚动数组，其中 `dp[i]` 表示容量为 `i` (单位为块)的背包可以存储的最大文件大小。
2. 从前往后遍历输入文件，处理逻辑如下：
 - a. 计算该文件所需文件块(向上取整) `weight`。
 - b. 因为使用的是滚动数组，需要从最大容量开始从前往后枚举，直接到等于`weight`结束。状态转移方程为 `dp[j] = max(dp[j], dp[j-weight] + fileSize[i])`
3. 根据2的逻辑，遍历完所有文件之后，输出 `dp[n]` 即可就是结果。

背包文件也属于机考中经常会出现的题，并且这种题解题思路和代码非常套路化，推荐背一背模板。

C++

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main() {
6      int n;
7      cin >> n;
8
9      int fileSizeArray[n];
10     for (int i = 0; i < n; i++) {
11         cin >> fileSizeArray[i];
12     }
13
14     // 计算软盘可以存放的块数
15     int blockCount = 1474560 / 512;
16
17     // 动态规划数组, dp[i] 表示容量为 i 的背包可以存储的最大文件大小
18     int dp[blockCount + 1] = {0};
19
20     for (int i = 0; i < n; i++) {
21         // 把文件大小转换成块数 向上取整
22         int weight = ceil(fileSizeArray[i] / 512.0);
23         int worth = fileSizeArray[i];
24         for (int j = blockCount; j >= weight; j--) {
25             dp[j] = max(dp[j], dp[j - weight] + worth);
26         }
27     }
28
29     cout << dp[blockCount] << endl;
30
31     return 0;
32 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          int n = scanner.nextInt(); // 读取文件数量
7
8          int[] fileSizeArray = new int[n];
9          for (int i = 0; i < n; i++) {
10             fileSizeArray[i] = scanner.nextInt();
11         }
12         scanner.close();
13
14         // 软盘可以存放的块数
15         int blockCount = 1474560 / 512;
16
17         // 动态规划数组, dp[i] 表示容量为 i 的背包可以存储的最大文件大小
18         int[] dp = new int[blockCount + 1];
19
20         for (int fileSize : fileSizeArray) {
21             // 计算文件需要的块数, 向上取整
22             int weight = (int) Math.ceil(fileSize / 512.0);
23             int worth = fileSize;
24
25             for (int j = blockCount; j >= weight; j--) {
26                 dp[j] = Math.max(dp[j], dp[j - weight] + worth);
27             }
28         }
29
30         System.out.println(dp[blockCount]);
31     }
32 }
```

Python

```
1 import sys
2 import math
3
4 def main():
5     # 读取第一行, 表示文件数量
6     n = int(sys.stdin.readline().strip())
7
8     # 读取接下来的 n 行, 获取文件大小
9     fileSizeArray = [int(sys.stdin.readline().strip()) for _ in range(n)]
10
11     # 软盘最大可用块数
12     block_count = 1474560 // 512
13
14     # 动态规划数组
15     dp = [0] * (block_count + 1)
16
17     for file_size in fileSizeArray:
18         weight = math.ceil(file_size / 512)
19         worth = file_size
20
21         for j in range(block_count, weight - 1, -1):
22             dp[j] = max(dp[j], dp[j - weight] + worth)
23
24     print(dp[block_count])
25
26 if __name__ == "__main__":
27     main()
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let fileSizeArray = [];
9  let n = -1;
10
11  rl.on("line", (line) => {
12    if (n === -1) {
13      n = parseInt(line.trim(), 10); // 读取 n
14    } else {
15      fileSizeArray.push(parseInt(line.trim(), 10)); // 读取文件大小
16      if (fileSizeArray.length === n) {
17        rl.close();
18      }
19    }
20  });
21
22  rl.on("close", () => {
23    const blockCount = Math.floor(1474560 / 512);
24    let dp = Array(blockCount + 1).fill(0);
25
26    for (const fileSize of fileSizeArray) {
27      const weight = Math.ceil(fileSize / 512);
28      const worth = fileSize;
29
30      for (let j = blockCount; j >= weight; j--) {
31        dp[j] = Math.max(dp[j], dp[j - weight] + worth);
32      }
33    }
34
35    console.log(dp[blockCount]);
36  });
```

Go


```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "math"
7     "os"
8     "strconv"
9 )
10
11 func main() {
12     scanner := bufio.NewScanner(os.Stdin)
13
14     // 读取第一行, 表示文件数量
15     scanner.Scan()
16     n, _ := strconv.Atoi(scanner.Text())
17
18     // 读取文件大小
19     fileSizeArray := make([]int, n)
20     for i := 0; i < n; i++ {
21         scanner.Scan()
22         fileSizeArray[i], _ = strconv.Atoi(scanner.Text())
23     }
24
25     // 计算软盘块数
26     blockCount := 1474560 / 512
27
28     // 动态规划数组
29     dp := make([]int, blockCount+1)
30
31     for _, fileSize := range fileSizeArray {
32         weight := int(math.Ceil(float64(fileSize) / 512.0))
33         worth := fileSize
34
35         for j := blockCount; j >= weight; j-- {
36             dp[j] = max(dp[j], dp[j-weight]+worth)
37         }
38     }
39
40     fmt.Println(dp[blockCount])
41 }
42
43 // max 函数
44 func max(a, b int) int {
45     if a > b {
```

```
46     return a
47 }
48 return b
49 }
```

来自: [华为OD机试2025 B卷 - 通过软盘拷贝文件 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试2025 B卷 - 计算快递主站点 (C++ & Python & JAVA & JS & GO)-CSDN博客

计算快递主站点

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025B卷 200分题型

题目描述

快递业务范围有N个站点，A站点与B站点可以中转快递，则认为A-B站可达，如果A-B可达，B-C可达，则A-C可达。

现在给N个站点编号0、1、...n-1，用 `s[i][j]` 表示i-j是否可达，`s[i][j]=1`表示i-j可达，`s[i][j]=0`表示i-j不可达。

现用 [二维数组](#) 给定N个站点的可达关系，请计算至少选择从几个主站点出发，才能可达所有站点（覆盖所有站点业务）。

说明：`s[i][j]`与`s[j][i]`取值相同。

输入描述

第一行输入为N（1 < N < 10000），N表示站点个数。

之后N行表示站点之间的可达关系，第i行第j个数值表示编号为i和j之间是否可达

输出描述

输出站点个数，表示至少需要多少个主站点。

用例1

输入

▼ Plain Text

```
1 4
2 1 1 1 1
3 1 1 1 0
4 1 1 1 0
5 1 0 0 1
```

输出

		Plain Text
1	1	

说明

选择0号站点作为主站点，0站点可达其他所有站点，所以至少选择1个站点作为主站才能覆盖所有站点业务。

用例2

输入

		Plain Text
1	4	
2	1 1 0 0	
3	1 1 0 0	
4	0 0 1 0	
5	0 0 0 1	

输出

		Plain Text
1	3	

说明

选择0号站点可以覆盖0、1站点，选择2号站点可以覆盖2号站点，选择3号站点可以覆盖3号站点，所以至少选择3个站点作为主站才能覆盖所有站点业务

题解

思路：并查集

1. 可以将所有可互相访问的快递站点放入一个组，每个组中有一个队长站点。定义一个数组 `ans[]` 每个元素中存储队长站点的id，初始时所有站点的队长站点定义为自身。
2. 遍历输入的二维数组，将 `grid[i][j] = 1` 的两个站点使用 并查集 算法进行合并两个组。新组的队长站点id可以取 合并之前两个组中 站点id较小的那个。
3. 经过2的逻辑之后，计算需要多少个主站点，只需要统计有多少组即可，通过计算队长的数量即可知道有多少个组。

并查集基本上都是套路题，推荐理解理解套路，这种题一般都能稳稳拿下。


```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<cmath>
8  #include<map>
9  using namespace std;
10
11 // 找到组中队长
12 int find(int a, vector<int> &ans) {
13     if (ans[a] != a) {
14         ans[a] = find(ans[a], ans);
15     }
16     return ans[a];
17 }
18
19 // 合并两个组
20 void merge(int a, int b, vector<int> &ans) {
21     int rootA = find(a, ans);
22     int rootB = find(b, ans);
23     int newRoot = min(rootA, rootB);
24     ans[rootA] = newRoot;
25     ans[rootB] = newRoot;
26 }
27
28
29 int main() {
30     int n ;
31     cin >> n;
32     vector<vector<int>> grid(n, vector<int>(n));
33     for (int i = 0; i < n; i++) {
34         for (int j = 0; j < n; j++) {
35             cin >> grid[i][j];
36         }
37     }
38
39     vector<int> ans(n);
40     for (int i = 0; i < n; i++) {
41         ans[i] = i;
42     }
43
44     // 对称的, 只需要遍历一半除对角线
45     for (int i = 0; i < n; i++) {
```

```
46         for (int j = 0; j < i; j++) {
47             if (grid[i][j] == 1) {
48                 merge(i, j, ans);
49             }
50         }
51     }
52 }
53
54 int count = 0;
55 for (int i = 0; i < n; i++) {
56     // 说明当前是队长
57     if (i == find(i, ans)) {
58         count++;
59     }
60 }
61 cout << count;
62 return 0;
}
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 查找组中队长 (带路径压缩)
5      static int find(int a, int[] parent) {
6          if (parent[a] != a) {
7              parent[a] = find(parent[a], parent);
8          }
9          return parent[a];
10     }
11
12     // 合并两个组
13     static void merge(int a, int b, int[] parent) {
14         int rootA = find(a, parent);
15         int rootB = find(b, parent);
16         int newRoot = Math.min(rootA, rootB);
17         parent[rootA] = newRoot;
18         parent[rootB] = newRoot;
19     }
20
21     public static void main(String[] args) {
22         Scanner sc = new Scanner(System.in);
23         int n = sc.nextInt();
24         int[][] grid = new int[n][n];
25
26         for (int i = 0; i < n; i++)
27             for (int j = 0; j < n; j++)
28                 grid[i][j] = sc.nextInt();
29
30         int[] parent = new int[n];
31         for (int i = 0; i < n; i++) parent[i] = i;
32
33         // 对称矩阵, 只遍历下三角 (不含对角线)
34         for (int i = 0; i < n; i++)
35             for (int j = 0; j < i; j++)
36                 if (grid[i][j] == 1)
37                     merge(i, j, parent);
38
39         int count = 0;
40         for (int i = 0; i < n; i++)
41             // 说明是队长
42             if (find(i, parent) == i)
43                 count++;
44
45         System.out.println(count);
```



```
46     }
47 }
```

Python

```
▼ Plain Text |
1  # 查找组中队长 (带路径压缩)
2  def find(a, parent):
3      if parent[a] != a:
4          parent[a] = find(parent[a], parent)
5      return parent[a]
6
7  # 合并两个组
8  def merge(a, b, parent):
9      rootA = find(a, parent)
10     rootB = find(b, parent)
11     newRoot = min(rootA, rootB)
12     parent[rootA] = newRoot
13     parent[rootB] = newRoot
14
15  def main():
16     n = int(input())
17     grid = [list(map(int, input().split())) for _ in range(n)]
18     parent = list(range(n))
19
20     # 对称矩阵, 只需处理下三角
21     for i in range(n):
22         for j in range(i):
23             if grid[i][j] == 1:
24                 merge(i, j, parent)
25
26     # 统计不同组数量 (即根节点数量)
27     count = sum(1 for i in range(n) if find(i, parent) == i)
28     print(count)
29
30  main()
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({ input: process.stdin });
4  let inputLines = [];
5  rl.on('line', line => inputLines.push(line));
6  rl.on('close', () => {
7      const n = parseInt(inputLines[0]);
8      const grid = inputLines.slice(1, n + 1).map(line => line.split(' ').map(Number));
9      const parent = Array.from({ length: n }, (_, i) => i);
10
11     // 查找组中队长 (带路径压缩)
12     const find = (a) => {
13         if (parent[a] !== a) parent[a] = find(parent[a]);
14         return parent[a];
15     };
16
17     // 合并两个组
18     const merge = (a, b) => {
19         const rootA = find(a), rootB = find(b);
20         const newRoot = Math.min(rootA, rootB);
21         parent[rootA] = newRoot;
22         parent[rootB] = newRoot;
23     };
24
25     // 遍历下三角
26     for (let i = 0; i < n; i++) {
27         for (let j = 0; j < i; j++) {
28             if (grid[i][j] === 1) {
29                 merge(i, j);
30             }
31         }
32     }
33
34     // 统计不同组数量
35     let count = 0;
36     for (let i = 0; i < n; i++) {
37         // 说明是队长
38         if (find(i) === i) count++;
39     }
40     console.log(count);
41 });
```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 查找组中队长 (带路径压缩)
12 func find(a int, parent []int) int {
13     if parent[a] != a {
14         parent[a] = find(parent[a], parent)
15     }
16     return parent[a]
17 }
18
19 // 合并两个组
20 func merge(a, b int, parent []int) {
21     rootA := find(a, parent)
22     rootB := find(b, parent)
23     newRoot := rootA
24     if rootB < rootA {
25         newRoot = rootB
26     }
27     parent[rootA] = newRoot
28     parent[rootB] = newRoot
29 }
30
31 func main() {
32     scanner := bufio.NewScanner(os.Stdin)
33     scanner.Scan()
34     n, _ := strconv.Atoi(scanner.Text())
35
36     grid := make([][]int, n)
37     for i := 0; i < n; i++ {
38         scanner.Scan()
39         parts := strings.Fields(scanner.Text())
40         grid[i] = make([]int, n)
41         for j := 0; j < n; j++ {
42             grid[i][j], _ = strconv.Atoi(parts[j])
43         }
44     }
45 }
```

```

46     parent := make([]int, n)
47     for i := 0; i < n; i++ {
48         parent[i] = i
49     }
50
51     // 遍历下三角矩阵
52     for i := 0; i < n; i++ {
53         for j := 0; j < i; j++ {
54             if grid[i][j] == 1 {
55                 merge(i, j, parent)
56             }
57         }
58     }
59
60     // 统计根节点个数
61     count := 0
62     for i := 0; i < n; i++ {
63         // 说明是队长
64         if find(i, parent) == i {
65             count++
66         }
67     }
68     fmt.Println(count)
69 }

```

来自: [华为OD机试2025 B卷 - 计算快递主站点 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试 2025B卷 - 找城市 / 城市聚集度

(C++ & Python & JAVA & JS & GO)_华为od上机

考试2025年a卷-CSDN博客

2025B卷目录点击查看：[华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

2025B卷 200分题型

题目描述

一张地图上有n个城市，城市和城市之间有且只有一条道路相连：要么直接相连，要么通过其它城市中转相连（可中转一次或多次）。城市与城市之间的道路**都不会成环**。

当切断通往某个城市 i 的所有道路后，地图上将分为多个连通的城市群，设该城市i的聚集度为 DP_i

（Degree of Polymerization）， $DP_i = \max$ （城市群1的城市个数，城市群2的城市个数，...城市群m 的城市个数）。

请找出地图上DP值最小的城市（即找到城市j，使得 $DP_j = \min(DP_1, DP_2 \dots DP_n)$ ）

提示：如果有多个城市都满足条件，这些城市都要找出来（可能存在多个解）

提示： DP_i 的计算，可以理解为已知一棵树，删除某个节点后；生成的多个子树，求解多个子数节点数的问题。

输入描述

每个样例：第一行有一个整数N，表示有N个节点。 $1 \leq N \leq 1000$ 。

接下来的N-1行每行有两个整数x, y，表示城市x与城市y连接。 $1 \leq x, y \leq N$

输出描述

输出城市的编号。如果有多个，按照编号升序输出。

用例1

输入

▼ Plain Text

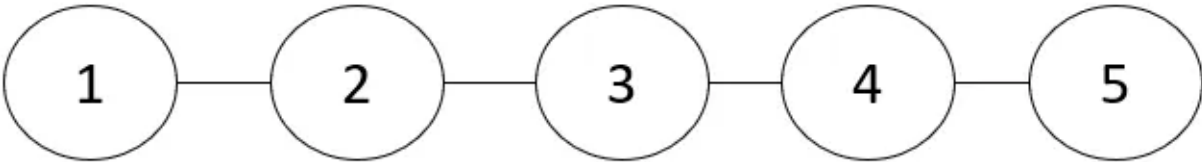
```
1 5
2 1 2
3 2 3
4 3 4
5 4 5
```

输出

▼ Plain Text	
1	3

说明

输入表示的是如下地图：



```
graph LR; 1((1)) --- 2((2)); 2 --- 3((3)); 3 --- 4((4)); 4 --- 5((5))
```

对于城市3，切断通往3的所有道路后，形成2个城市群[（1，2），（4，5）]，其聚集度分别都是2。
DP3 = 2。

对于城市4，切断通往城市4的所有道路后，形成2个城市群[（1，2，3），（5）]，DP4 = max（3，1） = 3。

依次类推，切断其它城市的所有道路后，得到的DP都会大于2，因为城市3就是满足条件的城市，输出是3。

用例2

输入

▼ Plain Text	
1	6
2	1 2
3	2 3
4	2 4
5	3 5
6	3 6

输出

1 2 3

说明

将通往2或者3的所有路径切断，最大城市群数量是3，其他任意城市切断后，最大城市群数量都比3大，所以输出2 3

题解

思路： 并查集

- 可以相互连通的城市，认作为一个城市群。简单说就是分组，这种题型非常适合用 并查集 算法来实现。
- 题目描述数据量比较小，完全可以枚举切割城市 $current\{1 - n\}$,计算切割每个城市边之后的DPI。
 - 切割城市边 转换到 并查集算法中 就是 忽略 有关current的边。
 - 并查集算法合并完所有符合条件的边后，统计各个城市群的数量， $DPI = \text{最大数量}$ 。
- 根据上述描述，记录 DPI最小的 城市。最后按照城市id升序输出。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<map>
9  #include<set>
10 #include<climits>
11 using namespace std;
12
13 // 找到城市群中代表
14 int find(int a, vector<int> &ans) {
15     if (ans[a] != a) {
16         ans[a] = find(ans[a], ans);
17     }
18     return ans[a];
19 }
20
21 // 合并两个城市群
22 void merge(int a, int b, vector<int> &ans) {
23     int rootA = find(a, ans);
24     int rootB = find(b, ans);
25     int newRoot = min(rootA, rootB);
26     ans[rootA] = newRoot;
27     ans[rootB] = newRoot;
28 }
29
30
31
32 vector<int> getDPI(vector<pair<int, int>>& graph, int n) {
33     vector<int> res;
34     int minDPI = INT_MAX;
35
36     // 枚举切换城市
37     for (int i = 1; i <= n; i++) {
38         // 初始化并查集数组
39         vector<int> ans(n+1);
40         for (int j = 0; j <= n; j++) {
41             ans[j] = j;
42         }
43
44         // 并查集进行合并 忽略涉及当前枚举城市的边
45         for (int j = 0; j < graph.size(); j++) {
```



```

46         int x = graph[j].first;
47         int y = graph[j].second;
48         if (x == i || y == i) {
49             continue;
50         }
51         merge(x, y, ans);
52     }
53
54     // 求分割后的最大城市群大小
55     vector<int> count(n+1, 0);
56     int maxValue = 0;
57     for (int i = 1; i <= n; i++) {
58         int tmp = find(i, ans);
59         count[tmp]++;
60         maxValue = max(maxValue, count[tmp]);
61     }
62
63     if (maxValue < minDPI) {
64         res.clear();
65         res.push_back(i);
66         minDPI = maxValue;
67     } else if (maxValue == minDPI) {
68         res.push_back(i);
69     }
70 }
71
72 sort(res.begin(), res.end());
73 return res;
74 }
75
76
77 int main() {
78     int n;
79     cin >> n;
80     vector<pair<int, int>> graph(n-1);
81     for (int i = 0; i < n - 1; i++) {
82         int x, y;
83         cin >> x >> y;
84         graph[i] = {x, y};
85     }
86     vector<int> res = getDPI(graph, n);
87     for (int i = 0; i < res.size(); i++) {
88         cout << res[i];
89         if (i != res.size() - 1) {
90             cout << " ";
91         }
92     }
93     return 0;

```

```
94 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5      // 找到城市群代表
6      static int find(int a, int[] ans) {
7          if (ans[a] != a) {
8              ans[a] = find(ans[a], ans);
9          }
10         return ans[a];
11     }
12
13     // 合并两个城市群
14     static void merge(int a, int b, int[] ans) {
15         int rootA = find(a, ans);
16         int rootB = find(b, ans);
17         int newRoot = Math.min(rootA, rootB);
18         ans[rootA] = newRoot;
19         ans[rootB] = newRoot;
20     }
21
22     static List<Integer> getDPI(List<int[]> graph, int n) {
23         List<Integer> res = new ArrayList<>();
24         int minDPI = Integer.MAX_VALUE;
25
26         // 枚举切断的城市
27         for (int i = 1; i <= n; i++) {
28             int[] ans = new int[n + 1];
29             for (int j = 0; j <= n; j++) {
30                 ans[j] = j;
31             }
32
33             // 合并除当前城市外的其他连接
34             for (int[] edge : graph) {
35                 int x = edge[0];
36                 int y = edge[1];
37                 if (x == i || y == i) continue;
38                 merge(x, y, ans);
39             }
40
41             // 统计每个连通块的大小
42             int[] count = new int[n + 1];
43             int maxVal = 0;
44             for (int j = 1; j <= n; j++) {
45                 if (j == i) continue;
```

```

46         int root = find(j, ans);
47         count[root]++;
48         maxValue = Math.max(maxValue, count[root]);
49     }
50
51     if (maxValue < minDPI) {
52         res.clear();
53         res.add(i);
54         minDPI = maxValue;
55     } else if (maxValue == minDPI) {
56         res.add(i);
57     }
58 }
59
60 Collections.sort(res);
61 return res;
62 }
63
64 public static void main(String[] args) {
65     Scanner sc = new Scanner(System.in);
66     int n = sc.nextInt();
67     List<int[]> graph = new ArrayList<>();
68     for (int i = 0; i < n - 1; i++) {
69         int x = sc.nextInt();
70         int y = sc.nextInt();
71         graph.add(new int[]{x, y});
72     }
73     List<Integer> res = getDPI(graph, n);
74     for (int i = 0; i < res.size(); i++) {
75         System.out.print(res.get(i));
76         if (i != res.size() - 1) {
77             System.out.print(" ");
78         }
79     }
80 }
81 }

```

Python

```
1  # 找到城市群代表
2  def find(a, ans):
3      if ans[a] != a:
4          ans[a] = find(ans[a], ans)
5      return ans[a]
6
7  # 合并两个城市群
8  def merge(a, b, ans):
9      rootA = find(a, ans)
10     rootB = find(b, ans)
11     newRoot = min(rootA, rootB)
12     ans[rootA] = newRoot
13     ans[rootB] = newRoot
14
15 def getDPI(graph, n):
16     res = []
17     minDPI = float('inf')
18
19     # 枚举切断的城市
20     for i in range(1, n + 1):
21         ans = list(range(n + 1))
22
23         # 合并除当前城市外的其他连接
24         for x, y in graph:
25             if x == i or y == i:
26                 continue
27             merge(x, y, ans)
28
29         # 统计每个连通块的大小
30         count = [0] * (n + 1)
31         maxValue = 0
32         for j in range(1, n + 1):
33             if j == i:
34                 continue
35             root = find(j, ans)
36             count[root] += 1
37             maxValue = max(maxValue, count[root])
38
39         if maxValue < minDPI:
40             res = [i]
41             minDPI = maxValue
42         elif maxValue == minDPI:
43             res.append(i)
44
45     res.sort()
```

```
46     return res
47
48 if __name__ == "__main__":
49     n = int(input())
50     graph = []
51     for _ in range(n - 1):
52         x, y = map(int, input().split())
53         graph.append((x, y))
54     res = getDPI(graph, n)
55     print(' '.join(map(str, res)))
```

JavaScript

```
1  // 找到城市群代表
2  function find(a, ans) {
3      if (ans[a] !== a) {
4          ans[a] = find(ans[a], ans);
5      }
6      return ans[a];
7  }
8
9  // 合并两个城市群
10 function merge(a, b, ans) {
11     let rootA = find(a, ans);
12     let rootB = find(b, ans);
13     let newRoot = Math.min(rootA, rootB);
14     ans[rootA] = newRoot;
15     ans[rootB] = newRoot;
16 }
17
18 function getDPI(graph, n) {
19     let res = [];
20     let minDPI = Number.MAX_SAFE_INTEGER;
21
22     // 枚举切断的城市
23     for (let i = 1; i <= n; i++) {
24         let ans = Array.from({ length: n + 1 }, (_, idx) => idx);
25
26         for (let [x, y] of graph) {
27             if (x === i || y === i) continue;
28             merge(x, y, ans);
29         }
30
31         let count = Array(n + 1).fill(0);
32         let maxValue = 0;
33         for (let j = 1; j <= n; j++) {
34             if (j === i) continue;
35             let root = find(j, ans);
36             count[root]++;
37             maxValue = Math.max(maxValue, count[root]);
38         }
39
40         if (maxValue < minDPI) {
41             res = [i];
42             minDPI = maxValue;
43         } else if (maxValue === minDPI) {
44             res.push(i);
45         }
46     }
47 }
```

```

46     }
47
48     res.sort((a, b) => a - b);
49     return res;
50 }
51
52 const readline = require('readline');
53 const rl = readline.createInterface({
54     input: process.stdin,
55     output: process.stdout
56 });
57
58 let input = [];
59 rl.on('line', (line) => {
60     input.push(line.trim());
61 });
62
63 rl.on('close', () => {
64     const n = parseInt(input[0]);
65     const graph = input.slice(1).map(line => line.split(' ').map(Number));
66     const res = getDPI(graph, n);
67     console.log(res.join(' '));
68 });

```

Go


```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9      "sort"
10 )
11
12 // 找到城市群代表
13 func find(a int, ans []int) int {
14     if ans[a] != a {
15         ans[a] = find(ans[a], ans)
16     }
17     return ans[a]
18 }
19
20 // 合并两个城市群
21 func merge(a, b int, ans []int) {
22     rootA := find(a, ans)
23     rootB := find(b, ans)
24     newRoot := min(rootA, rootB)
25     ans[rootA] = newRoot
26     ans[rootB] = newRoot
27 }
28
29 func min(a, b int) int {
30     if a < b {
31         return a
32     }
33     return b
34 }
35
36 func getDPI(graph [][]int, n int) []int {
37     res := []int{}
38     minDPI := int(1e9)
39
40     // 枚举切断的城市
41     for i := 1; i <= n; i++ {
42         ans := make([]int, n+1)
43         for j := 0; j <= n; j++ {
44             ans[j] = j
45         }
```

```

46
47     for _, edge := range graph {
48         x, y := edge[0], edge[1]
49         if x == i || y == i {
50             continue
51         }
52         merge(x, y, ans)
53     }
54
55     count := make([]int, n+1)
56     maxValue := 0
57     for j := 1; j <= n; j++ {
58         if j == i {
59             continue
60         }
61         root := find(j, ans)
62         count[root]++
63         if count[root] > maxValue {
64             maxValue = count[root]
65         }
66     }
67
68     if maxValue < minDPI {
69         res = []int{i}
70         minDPI = maxValue
71     } else if maxValue == minDPI {
72         res = append(res, i)
73     }
74 }
75
76 sort.Ints(res)
77 return res
78 }
79
80 func main() {
81     scanner := bufio.NewScanner(os.Stdin)
82     scanner.Scan()
83     n, _ := strconv.Atoi(scanner.Text())
84     graph := make([][2]int, n-1)
85     for i := 0; i < n-1; i++ {
86         scanner.Scan()
87         parts := strings.Fields(scanner.Text())
88         x, _ := strconv.Atoi(parts[0])
89         y, _ := strconv.Atoi(parts[1])
90         graph[i] = [2]int{x, y}
91     }
92     res := getDPI(graph, n)
93     for i, val := range res {

```

```
94     if i > 0 {
95         fmt.Print(" ")
96     }
97     fmt.Print(val)
98 }
99 }
```

来自: [华为OD机试 2025B卷 – 找城市 / 城市聚集度 \(C++ & Python & JAVA & JS & GO\)_华为od上机考试2025年a卷-CSDN博客](#)

华为od机试2025 B卷 - 字符串化繁为简 (C++ & Python & JAVA & JS & GO)_华为机试 字符串化繁为简-CSDN博客

字符串化繁为简

2025B卷目录点击查看：[华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

2025B卷 200分题型

题目描述

给定一个**输入字符串**，字符串只可能由英文字母（‘a’ ~ ‘z’、‘A’ ~ ‘Z’）和左右小括号（‘(’、‘)’）组成。

当字符串里存在小括号时，小括号是成对的，可以有一个或多个小括号对，小括号对不会嵌套，小括号对内可以包含1个或多个**英文字母**，也可以不包含英文字母。

当小括号对内包含多个英文字母时，这些字母之间是相互等效的关系，而且等效关系可以在不同的小括号对之间传递，即当存在‘a’和‘b’等效和存在‘b’和‘c’等效时，‘a’和‘c’也等效，另外，同一个英文字母的大写字母和小写字母也相互等效（即使它们分布在不同的括号对里）

需要对这个输入字符串做简化，输出一个新的字符串，**输出字符串**里只需保留输入字符串里的没有被小括号对包含的字符（按照输入字符串里的字符顺序），并将每个字符替换为在小括号对里包含的且字典序最小的等效字符。

如果简化后的字符串为空，请输出为"0"。

示例：

输入字符串为"never(dont)give(run)up(f)()"，初始等效字符集合为(‘d’, ‘o’, ‘n’, ‘t’)、(‘r’, ‘u’, ‘n’)，由于等效关系可以传递，因此最终等效字符集合为(‘d’, ‘o’, ‘n’, ‘t’, ‘r’, ‘u’)，将输入字符串里的剩余部分按字典序最小的等效字符替换后得到"devedgivedp”

输入描述

input_string 输入为1行，代表输入字符串 输入字符串的长度在1~100000之间

输出描述

output_string 输出为1行，代表输出字符串

用例1

输入

▼	Plain Text
1	()abd

输出

▼	Plain Text
1	abd

说明

输入字符串里没有被小括号包含的子字符串为"abd"，其中每个字符没有等效字符，输出为"abd"

用例2

输入

▼	Plain Text
1	(abd)demand(fb)()for

输出

▼	Plain Text
1	happwnewwear

用例3

输入

▼	Plain Text
1	()abcdefgAC(a)(Ab)(C)

输出

▼	Plain Text
1	AAcdefgAC

题解

思路： 模拟

1. 通过解析字符串 `()` 找出所有等效关系，使用数组保存各个等效关系。
2. 迭代合并等效关系。直到不能合并为止。 合并的时候需要考虑大小写
3. 根据最终等效关系，替换要输出字符串中各个字符得到最终结果。
4. 输出最终结果， 额外注意最终字符串的长度

C++

```
1  #include <iostream>
2      #include <string>
3      #include <vector>
4      #include <set>
5      #include <algorithm>
6
7      using namespace std;
8
9      int main() {
10          string input;
11          getline(cin, input);
12
13          string output = "";
14          // 记录所有等效关系 set会自动进行字典序的排序
15          vector<set<char>> equivalenceSets;
16          bool insideParentheses = false;
17
18          // 遍历输入字符串
19          for (char c : input) {
20              if (c == '(') {
21                  insideParentheses = true;
22                  equivalenceSets.push_back(set<char>());
23              } else if (c == ')') {
24                  insideParentheses = false;
25                  if (equivalenceSets.back().empty()) equivalenceSets.pop_ba
ck();
26              } else {
27                  if (!insideParentheses) {
28                      output += c;
29                  } else {
30                      equivalenceSets.back().insert(c);
31                  }
32              }
33          }
34
35          // 合并等价集合
36          bool merged = true;
37          while (merged) {
38              merged = false;
39              for (size_t i = 0; i < equivalenceSets.size(); ++i) {
40                  for (size_t j = i + 1; j < equivalenceSets.size(); ++j) {
41                      bool canMerge = false;
42                      for (char c = 'a'; c <= 'z'; ++c) {
43                          char upper = toupper(c);
44                          // 考虑大小写
```

```

45             if ((equivalenceSets[i].count(c) || equivalenceSet
46 s[i].count(upper)) &&
47             (equivalenceSets[j].count(c) || equivalenceSet
48 s[j].count(upper))) {
49                 canMerge = true;
50                 break;
51             }
52             if (canMerge) {
53                 equivalenceSets[i].insert(equivalenceSets[j].begin
54             (), equivalenceSets[j].end());
55                 equivalenceSets.erase(equivalenceSets.begin() +
56             j);
57                 merged = true;
58                 break;
59             }
60             if (merged) break;
61         }
62         // 替换字符为等价集中的字典序最小字符
63         for (const set<char>& set : equivalenceSets) {
64             char smallestChar = *set.begin();
65             for (char& c : output) {
66                 if (set.count(c)) c = smallestChar;
67             }
68         }
69         cout << (output.empty() ? "0" : output) << endl;
70         return 0;
71     }

```

JAVA


```

1  import java.util.*;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          // 读取输入字符串
7          Scanner sc = new Scanner(System.in);
8          String input = sc.nextLine();
9
10         StringBuilder output = new StringBuilder();
11         // 用于存储等价关系集合
12         List<Set<Character>> equivalenceSets = new ArrayList<>();
13         boolean insideParentheses = false;
14
15         // 遍历输入字符串
16         for (char c : input.toCharArray()) {
17             if (c == '(') {
18                 insideParentheses = true;
19                 equivalenceSets.add(new TreeSet<>()); // 使用 TreeSet 以保
持字典序
20             } else if (c == ')') {
21                 insideParentheses = false;
22                 if (equivalenceSets.get(equivalenceSets.size() - 1).isEmpty()
y()) {
23                     equivalenceSets.remove(equivalenceSets.size() - 1);
24                 }
25             } else {
26                 if (!insideParentheses) {
27                     output.append(c);
28                 } else {
29                     equivalenceSets.get(equivalenceSets.size() - 1).add
(c);
30                 }
31             }
32         }
33
34         // 合并等价集合
35         boolean merged = true;
36         while (merged) {
37             merged = false;
38             for (int i = 0; i < equivalenceSets.size(); i++) {
39                 for (int j = i + 1; j < equivalenceSets.size(); j++) {
40                     boolean canMerge = false;
41                     for (char c = 'a'; c <= 'z'; c++) {
42                         char upper = Character.toUpperCase(c);

```

```

43             if ((equivalenceSets.get(i).contains(c) || equival
44 enceSets.get(i).contains(upper)) &&
45             (equivalenceSets.get(j).contains(c) || equ
46 ivalenceSets.get(j).contains(upper))) {
47                 canMerge = true;
48                 break;
49             }
50             if (canMerge) {
51                 equivalenceSets.get(i).addAll(equivalenceSets.get
52 (j));
53                 equivalenceSets.remove(j);
54                 merged = true;
55                 break;
56             }
57             if (merged) break;
58         }
59     }
60     // 替换字符为字典序最小字符
61     for (Set<Character> set : equivalenceSets) {
62         char smallestChar = set.iterator().next(); // 获取字典序最小的字
63         符
64         for (int i = 0; i < output.length(); i++) {
65             if (set.contains(output.charAt(i))) {
66                 output.setCharAt(i, smallestChar);
67             }
68         }
69     }
70     // 输出结果
71     System.out.println(output.length() > 0 ? output : "0");
72 }
73 }

```

Python

```
1 def main():
2     # 读取输入字符串
3     input_string = input().strip()
4
5     output_string = ""
6     equivalence_sets = []
7     inside_parentheses = False
8
9     # 遍历输入字符串
10    for c in input_string:
11        if c == '(':
12            inside_parentheses = True
13            equivalence_sets.append(set())
14        elif c == ')':
15            inside_parentheses = False
16            if not equivalence_sets[-1]:
17                equivalence_sets.pop()
18        else:
19            if not inside_parentheses:
20                output_string += c
21            else:
22                equivalence_sets[-1].add(c)
23
24    # 合并等价集合
25    merged = True
26    while merged:
27        merged = False
28        for i in range(len(equivalence_sets)):
29            for j in range(i + 1, len(equivalence_sets)):
30                can_merge = False
31                for c in 'abcdefghijklmnopqrstuvwxyz':
32                    if c in equivalence_sets[i] or c.upper() in equivalence_sets[i]:
33                        if c in equivalence_sets[j] or c.upper() in equivalence_sets[j]:
34                            can_merge = True
35                            break
36                if can_merge:
37                    equivalence_sets[i].update(equivalence_sets[j])
38                    equivalence_sets.pop(j)
39                    merged = True
40                    break
41        if merged:
42            break
43
```

```
44     # 替换字符为字典序最小字符
45     for eq_set in equivalence_sets:
46         smallest_char = min(eq_set)
47         output_string = ''.join(smallest_char if c in eq_set else c for c
48 in output_string)
49
49     # 输出结果
50     print(output_string if output_string else "0")
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1  const readline = require('readline');
2
3  // 创建接口来处理输入和输出
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9  function main() {
10     rl.question('', (inputString) => {
11         inputString = inputString.trim();
12
13         let outputString = "";
14         let equivalenceSets = [];
15         let insideParentheses = false;
16
17         // 遍历输入字符串
18         for (let c of inputString) {
19             if (c === '(') {
20                 insideParentheses = true;
21                 equivalenceSets.push(new Set());
22             } else if (c === ')') {
23                 insideParentheses = false;
24                 if (equivalenceSets[equivalenceSets.length - 1].size ===
25                     0) {
26                     equivalenceSets.pop();
27                 }
28             } else {
29                 if (!insideParentheses) {
30                     outputString += c;
31                 } else {
32                     equivalenceSets[equivalenceSets.length - 1].add(c);
33                 }
34             }
35         }
36
37         // 合并等价集合
38         let merged = true;
39         while (merged) {
40             merged = false;
41             for (let i = 0; i < equivalenceSets.length; i++) {
42                 for (let j = i + 1; j < equivalenceSets.length; j++) {
43                     let canMerge = false;
44                     for (let c = 'a'.charCodeAt(0); c <= 'z'.charCodeAt(0); c++) {
```

```

44         let lower = String.fromCharCode(c);
45         let upper = String.fromCharCode(c - 32);
46         if (equivalenceSets[i].has(lower) || equivalenceSe
47     ts[i].has(upper)) {
48             if (equivalenceSets[j].has(lower) || equivalen
49 ceSets[j].has(upper)) {
50                 canMerge = true;
51                 break;
52             }
53         }
54         if (canMerge) {
55             equivalenceSets[i] = new Set([...equivalenceSets
56 [i], ...equivalenceSets[j]]);
57             equivalenceSets.splice(j, 1);
58             merged = true;
59             break;
60         }
61         if (merged) break;
62     }
63 }
64
65 // 替换字符为字典序最小字符
66 for (let eqSet of equivalenceSets) {
67     let smallestChar = [...eqSet].sort()[0];
68     outputString = [...outputString].map(c => eqSet.has(c) ? small
69 estChar : c).join('');
70 }
71
72 // 输出结果
73 console.log(outputString || "0");
74 rl.close(); // 关闭 readline 接口
75 }
76
77 main();

```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "sort"
8  )
9
10 func main() {
11     // 创建一个新的扫描器, 并增加其缓冲区大小 (64KB -> 1MB)
12     scanner := bufio.NewScanner(os.Stdin)
13     const maxCapacity = 10000000 // 设置为大约10MB
14     buf := make([]byte, maxCapacity)
15     scanner.Buffer(buf, maxCapacity)
16
17     // 读取输入
18     if scanner.Scan() {
19         inputString := scanner.Text()
20
21         var outputString []rune
22         var equivalenceSets [][]rune
23         insideParentheses := false
24
25         // 遍历输入字符串
26         for _, c := range inputString {
27             if c == '(' {
28                 insideParentheses = true
29                 equivalenceSets = append(equivalenceSets, []rune{})
30             } else if c == ')' {
31                 insideParentheses = false
32                 if len(equivalenceSets) > 0 && len(equivalenceSets[len(equivalenceSets)-1]) == 0 {
33                     equivalenceSets = equivalenceSets[:len(equivalenceSets)-1]
34                 }
35             } else {
36                 if !insideParentheses {
37                     outputString = append(outputString, c)
38                 } else {
39                     equivalenceSets[len(equivalenceSets)-1] = append(equivalenceSets[len(equivalenceSets)-1], c)
40                 }
41             }
42         }
43     }
```

```

44 // 合并等价集合
45 merged := true
46 for merged {
47     merged = false
48     for i := 0; i < len(equivalenceSets); i++ {
49         for j := i + 1; j < len(equivalenceSets); j++ {
50             canMerge := false
51             // 判断是否可以合并两个集合
52             for c := 'a'; c <= 'z'; c++ {
53                 upper := c - 32
54                 if contains(equivalenceSets[i], c) || contains(equivalenceSet
55 s[i], upper) {
56                     if contains(equivalenceSets[j], c) || contains(equivalences
57 ets[j], upper) {
58                         canMerge = true
59                         break
60                     }
61                 }
62             }
63             if canMerge {
64                 // 合并两个集合
65                 equivalenceSets[i] = append(equivalenceSets[i], equivalenceSe
66 ts[j]...)
67                 // 去除重复字符并排序
68                 equivalenceSets[i] = uniqueSort(equivalenceSets[i])
69                 equivalenceSets = append(equivalenceSets[:j], equivalenceSets
70 [j+1:]...)
71                 merged = true
72                 break
73             }
74         }
75     }
76 }
77
78 // 替换字符为字典序最小字符
79 for _, eqSet := range equivalenceSets {
80     sort.Slice(eqSet, func(i, j int) bool {
81         return eqSet[i] < eqSet[j]
82     })
83     smallestChar := eqSet[0]
84     for i, c := range outputString {
85         if contains(eqSet, c) {
86             outputString[i] = smallestChar
87         }
88     }
89 }

```



```

88     }
89
90     // 输出结果
91     if len(outputString) == 0 {
92         fmt.Println("")
93     } else {
94         fmt.Println(string(outputString))
95     }
96 } else {
97     // 输入失败的情况
98     fmt.Println("Input error.")
99 }
100 }
101
102 // 判断字符是否在集合中
103 func contains(set []rune, c rune) bool {
104     for _, item := range set {
105         if item == c {
106             return true
107         }
108     }
109     return false
110 }
111
112 // 去重并排序集合
113 func uniqueSort(slice []rune) []rune {
114     m := make(map[rune]struct{})
115     var result []rune
116     for _, v := range slice {
117         if _, ok := m[v]; !ok {
118             m[v] = struct{}{}
119             result = append(result, v)
120         }
121     }
122     sort.Slice(result, func(i, j int) bool {
123         return result[i] < result[j]
124     })
125     return result
126 }

```

华为OD机试 2025 B卷 - 最佳植树距离 (C++ & Python & JAVA & JS & GO)-CSDN博客

最佳植树距离

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 200分题型

题目描述

按照环保公司要求，小明需要在沙化严重的地区进行植树防沙工作，初步目标是种植一条直线的树带。由于有些区域目前不适合种植树木，所以只能在一些可以种植的点来种植树木。

在树苗有限的情况下，要达到最佳效果，就要尽量散开种植，不同树苗之间的最小间距要尽量大。给你一个适合种情树木的点坐标和一个树苗的数量，请帮小明选择一个最佳的最小种植间距。

例如，适合种植树木的位置分别为1,3,5,6,7,10,13 树苗数量是3，种植位置在1,7,13，树苗之间的间距都是6，均匀分开，就达到了散开种植的目的，最佳的最小种植间距是6

输入描述

第1行表示适合种树的坐标数量
第2行是适合种树的坐标位置
第3行是树苗的数量

备注

- 位置范围为1~10000000
- 种植树苗的数量范围2~10000000
- 用例确保种桔的树苗数量不会超过有效种桔坐标数量

输出描述

最佳的最小种植间距

用例1

输入

▼ Plain Text

```
1 7
2 1 5 3 6 10 7 13
3 3
```

输出

▼	Plain Text
1	6

说明

▼	Plain Text
1	3棵树苗分别种植在1，7，13位置时，树苗种植的最均匀，最小间距为6

题解

思路：二分

1. 使用数组(position[])接收输入适合种树的坐标位置，进行升序排序。
2. 初始设置二分的左右边界 `left = 1, right= position[n-1] - position[0]` (假设了数组长度为0) 接下来就是执行二分算法的基本操作了，每次检验 `mid =(left + right + 1) /2` 是否可以将数全部种植：
 - a. 可以，则更新 `left = mid`
 - b. 不可以，则更新 `right = mid - 1`
3. 至于检验是否可以全部种植的逻辑：就是种植每棵树间隔距离 $\geq mid$,种植的数量是否大于等于树苗的数量。
4. 二分结束结束循环的标志为 `left == right` .此时的left或者right就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  using namespace std;
8
9  // 判断是否允许指定距离 ans 可种植坐标 m 树苗数量 dis 指定距离
10 bool check(vector<int> ans, int m , int dis) {
11     int count = 1;
12     int currentPosoition = ans[0];
13     for (int i = 1; i < ans.size(); i++) {
14         if (ans[i] - currentPosoition >= dis) {
15             count++;
16             currentPosoition = ans[i];
17         }
18     }
19     return count >= m;
20 }
21
22
23 int main() {
24     int n ;
25     cin >> n;
26     vector<int> ans(n, 0);
27     for (int i = 0; i < n; i++) {
28         cin >> ans[i];
29     }
30     int count;
31     cin >> count;
32     sort(ans.begin(), ans.end());
33     // 二分设置边界
34     int left = 1, right = ans[n - 1] - ans[0];
35     while (left < right) {
36         int mid = (left + right + 1) >> 1;
37         if (check(ans, count, mid)) {
38             left = mid;
39         } else {
40             right = mid - 1;
41         }
42     }
43     cout << left;
44     return 0;
45 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 判断是否可以以 dis 为最小间隔, 放置 m 棵树
5      public static boolean check(List<Integer> positions, int m, int dis) {
6          int count = 1; // 第一个点默认放置
7          int currentPosition = positions.get(0);
8
9          for (int i = 1; i < positions.size(); i++) {
10             if (positions.get(i) - currentPosition >= dis) {
11                 count++;
12                 currentPosition = positions.get(i);
13             }
14         }
15         return count >= m;
16     }
17
18     public static void main(String[] args) {
19         Scanner scanner = new Scanner(System.in);
20
21         int n = scanner.nextInt();
22         List<Integer> positions = new ArrayList<>();
23
24         for (int i = 0; i < n; i++) {
25             positions.add(scanner.nextInt());
26         }
27
28         int count = scanner.nextInt();
29         scanner.close();
30
31         // 排序
32         Collections.sort(positions);
33
34         // 二分搜索范围
35         int left = 1, right = positions.get(n - 1) - positions.get(0);
36
37         while (left < right) {
38             int mid = (left + right + 1) >> 1; // 向上取整
39             if (check(positions, count, mid)) {
40                 left = mid; // 继续尝试更大间隔
41             } else {
42                 right = mid - 1;
43             }
44         }
45     }
```

```
46         System.out.println(left);
47     }
48 }
```

Python

▼ Plain Text

```
1  def check(positions, m, dis):
2      count = 1 # 默认放置第一棵树
3      current_position = positions[0]
4
5      for i in range(1, len(positions)):
6          if positions[i] - current_position >= dis:
7              count += 1
8              current_position = positions[i]
9
10     return count >= m
11
12 def main():
13     # 读取输入
14     n = int(input().strip())
15     positions = list(map(int, input().split()))
16     m = int(input().strip())
17
18     # 排序
19     positions.sort()
20
21     # 二分搜索范围
22     left, right = 1, positions[-1] - positions[0]
23
24     while left < right:
25         mid = (left + right + 1) // 2 # 向上取整
26         if check(positions, m, mid):
27             left = mid # 继续尝试更大间隔
28         else:
29             right = mid - 1
30
31     print(left)
32
33 if __name__ == "__main__":
34     main()
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let inputLines = [];
9  rl.on("line", (line) => {
10    inputLines.push(line);
11  }).on("close", () => {
12    main(inputLines);
13  });
14
15  function check(positions, m, dis) {
16    let count = 1; // 默认放置第一棵树
17    let currentPosition = positions[0];
18
19    for (let i = 1; i < positions.length; i++) {
20      if (positions[i] - currentPosition >= dis) {
21        count++;
22        currentPosition = positions[i];
23      }
24    }
25
26    return count >= m;
27  }
28
29  function main(inputLines) {
30    let n = parseInt(inputLines[0].trim());
31    let positions = inputLines[1].trim().split(" ").map(Number);
32    let m = parseInt(inputLines[2].trim());
33
34    positions.sort((a, b) => a - b); // 排序
35
36    let left = 1, right = positions[n - 1] - positions[0];
37
38    while (left < right) {
39      let mid = Math.floor((left + right + 1) / 2); // 向上取整
40      if (check(positions, m, mid)) {
41        left = mid; // 继续尝试更大间隔
42      } else {
43        right = mid - 1;
44      }
45    }
```



```
46  
47  
48     console.log(left);  
    }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 // check 判断是否可以以 dis 作为最小间隔放置 m 棵树
13 func check(positions []int, m int, dis int) bool {
14     count := 1 // 默认放置第一棵树
15     currentPosition := positions[0]
16
17     for i := 1; i < len(positions); i++ {
18         if positions[i]-currentPosition >= dis {
19             count++
20             currentPosition = positions[i]
21         }
22     }
23     return count >= m
24 }
25
26 func main() {
27     // 使用 bufio 读取输入
28     scanner := bufio.NewScanner(os.Stdin)
29
30     // 读取 n
31     scanner.Scan()
32     n, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
33
34     // 读取坐标列表
35     scanner.Scan()
36     positionStr := strings.Fields(scanner.Text())
37     positions := make([]int, n)
38     for i := 0; i < n; i++ {
39         positions[i], _ = strconv.Atoi(positionStr[i])
40     }
41
42     // 读取 m
43     scanner.Scan()
44     m, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
45 }
```

```

46 // 排序位置数组
47 sort.Ints(positions)
48
49 // 二分查找最大间距
50 left, right := 1, positions[n-1]-positions[0]
51 for left < right {
52     mid := (left + right + 1) / 2 // 向上取整
53     if check(positions, m, mid) {
54         left = mid // 继续尝试更大间隔
55     } else {
56         right = mid - 1
57     }
58 }
59
60 // 输出最终结果
61 fmt.Println(left)
62 }

```

来自: [华为OD机试 2025 B卷 – 最佳植树距离 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

华为OD机试 - 租车骑绿岛 (C++ & Python & JAVA & JS & GO)_计算最少需要多少辆自行车才能满足条件-CSDN博客

租车骑绿岛

2025B卷目录点击查看：[华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

2025B卷 100分题型

题目描述

部门组织绿岛骑行团建活动。租用公共双人自行车，每辆自行车最多坐两人，最大载重M。给出部门每个人的体重，请问最多需要租用多少双人自行车。

输入描述

- 第一行两个数字m、n，分别代表自行车限重，部门总人数。
- 第二行，n个数字，代表每个人的体重，体重都小于等于自行车限重m。
- $0 < m \leq 200$
 - $0 < n \leq 1000000$

输出描述

最小需要的双人自行车数量。

用例1

输入

▼	Plain Text
1 3 4	
2 3 2 2 1	

输出

▼	Plain Text
1 3	

说明

无

题解

本题需要最少的车辆，即尽可能组合出重量小于等于m的两人组。贪心算法 + 双指针

- 按体重进行升序排序。left 只想最小体重， right指向最大体重。
 - $arr[left] + arr[right] \leq m$ left ++, right -
 - $arr[left] + arr[right] > m$ right -，优先减少体重大的

C++

```
1  #include <ios>
2  #include<iostream>
3  #include<vector>
4  #include<string>
5  #include <utility>
6  #include <sstream>
7  #include<algorithm>
8  using namespace std;
9
10 int main() {
11     int m, n;
12     cin >> m >> n;
13     int res= 0;
14     vector<int> ans(n);
15     for (int i = 0; i < n; i++) {
16         cin >> ans[i];
17     }
18     // 排序
19     sort(ans.begin(), ans.end());
20     int right = n - 1;
21     int left = 0;
22     while (left <= right) {
23         int tmp = ans[left] + ans[right];
24         // 说明不能同时两个人乘坐
25         if (tmp > m) {
26             right--;
27             res++;
28             // 能够同时乘坐
29         } else {
30
31             right--;
32             left++;
33             res++;
34         }
35     }
36
37     cout << res;
38     return 0;
39 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          // 读取输入
8          int m = scanner.nextInt();
9          int n = scanner.nextInt();
10         int[] ans = new int[n];
11
12         for (int i = 0; i < n; i++) {
13             ans[i] = scanner.nextInt();
14         }
15
16         // 排序
17         Arrays.sort(ans);
18         int left = 0, right = n - 1;
19         int res = 0;
20
21         while (left <= right) {
22             int tmp = ans[left] + ans[right];
23             if (tmp > m) { // 不能同时乘坐
24                 right--;
25                 res++;
26             } else { // 可以同时乘坐
27                 right--;
28                 left++;
29                 res++;
30             }
31         }
32
33         System.out.println(res);
34         scanner.close();
35     }
36 }
```

Python

```
1  import sys
2
3  def main():
4      # 读取输入
5      m, n = map(int, sys.stdin.readline().split())
6      ans = list(map(int, sys.stdin.readline().split()))
7
8      # 排序
9      ans.sort()
10     left, right = 0, n - 1
11     res = 0
12
13     while left <= right:
14         tmp = ans[left] + ans[right]
15         if tmp > m: # 不能同时乘坐
16             right -= 1
17             res += 1
18         else: # 可以同时乘坐
19             right -= 1
20             left += 1
21             res += 1
22
23     print(res)
24
25 if __name__ == "__main__":
26     main()
```

JavaScript


```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let inputLines = [];
9
10 rl.on("line", (line) => {
11   inputLines.push(line.trim());
12   if (inputLines.length === 2) {
13     rl.close();
14   }
15 });
16
17 rl.on("close", () => {
18   let [m, n] = inputLines[0].split(" ").map(Number);
19   let ans = inputLines[1].split(" ").map(Number);
20
21   // 排序
22   ans.sort((a, b) => a - b);
23   let left = 0, right = n - 1;
24   let res = 0;
25
26   while (left <= right) {
27     let tmp = ans[left] + ans[right];
28     if (tmp > m) { // 不能同时乘坐
29       right--;
30       res++;
31     } else { // 可以同时乘坐
32       right--;
33       left++;
34       res++;
35     }
36   }
37
38   console.log(res);
39 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 func main() {
13     // 读取输入
14     scanner := bufio.NewScanner(os.Stdin)
15     scanner.Scan()
16     firstLine := strings.Split(scanner.Text(), " ")
17     m, _ := strconv.Atoi(firstLine[0])
18     n, _ := strconv.Atoi(firstLine[1])
19
20     scanner.Scan()
21     inputNumbers := strings.Split(scanner.Text(), " ")
22     ans := make([]int, n)
23     for i := 0; i < n; i++ {
24         ans[i], _ = strconv.Atoi(inputNumbers[i])
25     }
26
27     // 排序
28     sort.Ints(ans)
29     res := 0
30     right := n - 1
31     left := 0
32
33     for left <= right {
34         tmp := ans[left] + ans[right]
35         // 说明不能同时两个人乘坐
36         if tmp > m {
37             right--
38             res++
39         } // 能够同时乘坐
40         else {
41             right--
42             left++
43             res++
44         }
45     }
```

```
46  
47     fmt.Println(res)  
48 }
```

来自: [华为OD机试 - 租车骑绿岛 \(C++ & Python & JAVA & JS & GO\)_计算最少需要多少辆自行车才能满足条件-CSDN博客](#)

华为OD机试2025 B卷 - 矩阵元素的边界值 (C++ & Python & JAVA & JS & GO)-CSDN博客

矩阵元素的边界值

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

给定一个N*M矩阵，请先找出M个该矩阵中每列元素的**最大值**，然后输出这M个值中的最小值

输入描述

给定一个N*M矩阵，N和M的取值范围均为：[0, 100]

输出描述

M个该矩阵中每列元素的最大值，然后输出这M个值中的**最小值**

示例1

输入

▼ Plain Text

1 [[1,2],[3,4]]

输出

▼ Plain Text

1 3

说明

第一列元素为：1和3，最大值为3；
第二列元素为：2和4，最大值为4
各列最大值3和4的最小值为3

题解

思路： 栈模拟

1. 使用栈模拟来实现，使用数组来模拟栈，在栈中维持一个连续非严格递增的序列，定义 `pos` 作为栈的索引。使用 `res` 记录出现过最长的非严格递增连续数字序列的长度。
2. 具体逻辑：从前往后遍历输入字符串，根据不同字符进行处理情况如下
 - 当前字符为数字字符
 - 当前栈为空 或者 当前字符 \geq 栈顶字符，将当前字符加入栈中，并将 `pos++`
 - 当前字符 $<$ 栈顶字符 时，说明不满足连续非递增序列，更新 `res = max(res, pos)` .并将栈清空 `pos = 0` ,此时当前数字字符会形成一个新的序列，将当前字符压入栈中，并更新 `pos = 1`
 - 当前字符为字母字符，说明不满足连续非递增序列。更新 `res = max(res, pos)` .并将栈清空 `pos = 0`
3. 执行2的逻辑，遍历完输入字符串后，需要进行收尾操作。更新 `res = max(res, pos)` 。这主要是考虑输入字符串最后一段满足连续非严格递增序列。
4. 输出记录的 `res` 就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<cmath>
8  #include<map>
9  #include<climits>
10 using namespace std;
11
12 // 通用 split 函数
13 vector<string> split(const string& str, const string& delimiter) {
14     vector<string> result;
15     size_t start = 0;
16     size_t end = str.find(delimiter);
17     while (end != string::npos) {
18         result.push_back(str.substr(start, end - start));
19         start = end + delimiter.length();
20         end = str.find(delimiter, start);
21     }
22     // 添加最后一个部分
23     result.push_back(str.substr(start));
24     return result;
25 }
26
27
28 int solve(vector<vector<int>>& grid) {
29     int n = grid.size();
30     if (n == 0 || grid[0].size() == 0) {
31         return 0;
32     }
33     int m = grid[0].size();
34
35     int res = INT_MAX;
36     for (int j = 0; j < m; j++) {
37         // 当前列最大值
38         int tmp = INT_MIN;
39         for (int i = 0; i < n; i++) {
40             tmp = max(tmp, grid[i][j]);
41         }
42         // 所有列中最小值
43         res = min(res, tmp);
44     }
45     return res;
```

```

46 }
47
48 int main() {
49     string input;
50     getline(cin, input);
51
52     if (input.empty() || input.size() == 2) {
53         cout << 0;
54         return 0;
55     }
56     // 处理输入字符串 [[1,2],[3,4]] => 1,2],[3,4
57     input = input.substr(2, input.size() - 4);
58
59     vector<string> rows = split(input, "],[");
60     int n = rows.size();
61     vector<vector<int>> grid(n);
62     for (int i = 0; i < n; i++) {
63         string rowsInput = rows[i];
64
65         vector<string> values = split(rowsInput, ",");
66
67         for (int j = 0; j < values.size(); j++) {
68             grid[i].push_back(stoi(values[j]));
69         }
70     }
71
72     int res = solve(grid);
73     cout << res;
74     return 0;
75 }

```

Java

```
1  import java.util.*;
2
3  public class Main {
4      public static int solve(List<List<Integer>> grid) {
5          int n = grid.size();
6          if (n == 0 || grid.get(0).isEmpty()) return 0;
7          int m = grid.get(0).size();
8          // 记录所有列最大值其中的最小值
9          int res = Integer.MAX_VALUE;
10
11         for (int j = 0; j < m; j++) {
12             // 每一列最大值
13             int colMax = Integer.MIN_VALUE;
14             for (int i = 0; i < n; i++) {
15                 colMax = Math.max(colMax, grid.get(i).get(j));
16             }
17             res = Math.min(res, colMax);
18         }
19
20         return res;
21     }
22
23     public static void main(String[] args) {
24         Scanner sc = new Scanner(System.in);
25         String input = sc.nextLine().trim();
26
27         if (input.length() <= 2) {
28             System.out.println(0);
29             return;
30         }
31
32         // 去掉 "[" 和 "]"
33         input = input.substring(2, input.length() - 2);
34         // 按 "],[" 切分 => 每一行
35         String[] rowStrings = input.split("\\],\\[");
36
37         List<List<Integer>> grid = new ArrayList<>();
38         for (String row : rowStrings) {
39             String[] vals = row.split(",");
40             List<Integer> nums = new ArrayList<>();
41             for (String v : vals) {
42                 nums.add(Integer.parseInt(v));
43             }
44             grid.add(nums);
45         }
46     }
47 }
```



```
46
47         System.out.println(solve(grid));
48     }
49 }
```

Python

```
▼ Plain Text |
1  def solve(grid):
2      if not grid or not grid[0]:
3          return 0
4      n, m = len(grid), len(grid[0])
5      res = float('inf')
6
7      for j in range(m):
8          col_max = max(grid[i][j] for i in range(n))
9          res = min(res, col_max)
10     return res
11
12     def main():
13         input_str = input().strip()
14         if len(input_str) <= 2:
15             print(0)
16             return
17         # 转换字符串为二维数组
18         input_str = input_str[2:-2] # 去掉 [[ 和 ]]
19         row_strs = input_str.split(",")
20         grid = [list(map(int, row.split(","))) for row in row_strs]
21         print(solve(grid))
22
23     if __name__ == "__main__":
24         main()
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  rl.on('line', function (line) {
9    line = line.trim();
10    if (line.length <= 2) {
11      console.log(0);
12      rl.close();
13      return;
14    }
15
16    // 去掉 [[ 和 ]]
17    line = line.slice(2, -2);
18    // 切分为行
19    const rowStrs = line.split("\n");
20    const grid = rowStrs.map(row => row.split(",").map(Number));
21
22    const n = grid.length;
23    const m = grid[0].length;
24    let res = Infinity;
25
26    for (let j = 0; j < m; j++) {
27      let colMax = -Infinity;
28      for (let i = 0; i < n; i++) {
29        colMax = Math.max(colMax, grid[i][j]);
30      }
31      res = Math.min(res, colMax);
32    }
33
34    console.log(res);
35    rl.close();
36  });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11 // 求每列的最大值，再取最小的那个
12 func solve(grid [][]int) int {
13     if len(grid) == 0 || len(grid[0]) == 0 {
14         return 0
15     }
16     n, m := len(grid), len(grid[0])
17     res := int(1e9)
18
19     for j := 0; j < m; j++ {
20         colMax := -1 << 31
21         for i := 0; i < n; i++ {
22             if grid[i][j] > colMax {
23                 colMax = grid[i][j]
24             }
25         }
26         if colMax < res {
27             res = colMax
28         }
29     }
30     return res
31 }
32
33 // 切分字符串
34 func split(s string, delim string) []string {
35     var result []string
36     for {
37         idx := strings.Index(s, delim)
38         if idx == -1 {
39             break
40         }
41         result = append(result, s[:idx])
42         s = s[idx+len(delim):]
43     }
44     result = append(result, s)
45     return result
}
```

```

46 }
47
48 func main() {
49     reader := bufio.NewReader(os.Stdin)
50     input, _ := reader.ReadString('\n')
51     input = strings.TrimSpace(input)
52
53     // 边界判断
54     if len(input) <= 2 {
55         fmt.Println(0)
56         return
57     }
58
59     // 去除头尾 "[" 和 "]"
60     input = input[2 : len(input)-2]
61
62     // 使用 "],[" 分割每一行
63     rowStrs := split(input, "],[")
64
65     var grid [][]int
66     for _, row := range rowStrs {
67         parts := strings.Split(row, ",")
68         var nums []int
69         for _, p := range parts {
70             val, err := strconv.Atoi(p)
71             if err != nil {
72                 fmt.Println(0)
73                 return
74             }
75             nums = append(nums, val)
76         }
77         grid = append(grid, nums)
78     }
79
80     fmt.Println(solve(grid))
81 }

```

矩阵元素的边界值

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

给定一个N*M矩阵，请先找出M个该矩阵中每列元素的**最大值**，然后输出这M个值中的最小值

输入描述

给定一个N*M矩阵，N和M的取值范围均为：[0, 100]

输出描述

M个该矩阵中每列元素的最大值，然后输出这M个值中的最小值

示例1

输入

▼ Plain Text |

```
1  [[1,2],[3,4]]
```

输出

▼ Plain Text |

```
1  3
```

说明

第一列元素为：1和3，最大值为3；
第二列元素为：2和4，最大值为4
各列最大值3和4的最小值为3

题解

思路： 栈模拟

- 使用栈模拟来实现，使用数组来模拟栈，在栈中维持一个连续非严格递增的序列，定义 `pos` 作为栈的索引。使用 `res` 记录出现过最长的非严格递增连续数字序列的长度。
- 具体逻辑：从前往后遍历输入字符串，根据不同字符进行处理情况如下
 - 当前字符为数字字符
 - 当前栈为空 或者 当前字符 `>=` 栈顶字符，将当前字符加入栈中，并将`pos++`
 - 当前字符 `<` 栈顶字符 时，说明不满足连续非递增序列，更新 `res = max(res, pos)` .并将栈清空 `pos = 0` ,此时当前数字字符会形成一个新的序列，将当前字符压入栈中，并更新 `pos = 1`
 - 当前字符为字母字符，说明不满足连续非递增序列。更新 `res = max(res, pos)` .并将栈清空 `pos = 0`
- 执行2的逻辑，遍历完输入字符串后，需要进行收尾操作。更新 `res = max(res, pos)` 。这主要是考虑输入字符串最后一段满足连续非严格递增序列。
- 输出记录的 `res` 就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<cmath>
8  #include<map>
9  #include<climits>
10 using namespace std;
11
12 // 通用 split 函数
13 vector<string> split(const string& str, const string& delimiter) {
14     vector<string> result;
15     size_t start = 0;
16     size_t end = str.find(delimiter);
17     while (end != string::npos) {
18         result.push_back(str.substr(start, end - start));
19         start = end + delimiter.length();
20         end = str.find(delimiter, start);
21     }
22     // 添加最后一个部分
23     result.push_back(str.substr(start));
24     return result;
25 }
26
27
28 int solve(vector<vector<int>>& grid) {
29     int n = grid.size();
30     if (n == 0 || grid[0].size() == 0) {
31         return 0;
32     }
33     int m = grid[0].size();
34
35     int res = INT_MAX;
36     for (int j = 0; j < m; j++) {
37         // 当前列最大值
38         int tmp = INT_MIN;
39         for (int i = 0; i < n; i++) {
40             tmp = max(tmp, grid[i][j]);
41         }
42         // 所有列中最小值
43         res = min(res, tmp);
44     }
45     return res;
```

```

46 }
47
48 int main() {
49     string input;
50     getline(cin, input);
51
52     if (input.empty() || input.size() == 2) {
53         cout << 0;
54         return 0;
55     }
56     // 处理输入字符串 [[1,2],[3,4]] => 1,2],[3,4
57     input = input.substr(2, input.size() - 4);
58
59     vector<string> rows = split(input, "],[");
60     int n = rows.size();
61     vector<vector<int>> grid(n);
62     for (int i = 0; i < n; i++) {
63         string rowsInput = rows[i];
64
65         vector<string> values = split(rowsInput, ",");
66
67         for (int j = 0; j < values.size(); j++) {
68             grid[i].push_back(stoi(values[j]));
69         }
70     }
71
72     int res = solve(grid);
73     cout << res;
74     return 0;
75 }

```

Java

```
1  import java.util.*;
2
3  public class Main {
4      public static int solve(List<List<Integer>> grid) {
5          int n = grid.size();
6          if (n == 0 || grid.get(0).isEmpty()) return 0;
7          int m = grid.get(0).size();
8          // 记录所有列最大值其中的最小值
9          int res = Integer.MAX_VALUE;
10
11         for (int j = 0; j < m; j++) {
12             // 每一列最大值
13             int colMax = Integer.MIN_VALUE;
14             for (int i = 0; i < n; i++) {
15                 colMax = Math.max(colMax, grid.get(i).get(j));
16             }
17             res = Math.min(res, colMax);
18         }
19
20         return res;
21     }
22
23     public static void main(String[] args) {
24         Scanner sc = new Scanner(System.in);
25         String input = sc.nextLine().trim();
26
27         if (input.length() <= 2) {
28             System.out.println(0);
29             return;
30         }
31
32         // 去掉 "[" 和 "]"
33         input = input.substring(2, input.length() - 2);
34         // 按 "],[" 切分 => 每一行
35         String[] rowStrings = input.split("\\],\\[");
36
37         List<List<Integer>> grid = new ArrayList<>();
38         for (String row : rowStrings) {
39             String[] vals = row.split(",");
40             List<Integer> nums = new ArrayList<>();
41             for (String v : vals) {
42                 nums.add(Integer.parseInt(v));
43             }
44             grid.add(nums);
45         }
46     }
47 }
```



```
46
47     System.out.println(solve(grid));
48 }
49 }
```

Python

▼ Plain Text

```
1 def solve(grid):
2     if not grid or not grid[0]:
3         return 0
4     n, m = len(grid), len(grid[0])
5     res = float('inf')
6
7     for j in range(m):
8         col_max = max(grid[i][j] for i in range(n))
9         res = min(res, col_max)
10    return res
11
12 def main():
13     input_str = input().strip()
14     if len(input_str) <= 2:
15         print(0)
16         return
17     # 转换字符串为二维数组
18     input_str = input_str[2:-2] # 去掉 [[ 和 ]]
19     row_strs = input_str.split(",")
20     grid = [list(map(int, row.split(","))) for row in row_strs]
21     print(solve(grid))
22
23 if __name__ == "__main__":
24     main()
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  rl.on('line', function (line) {
9    line = line.trim();
10    if (line.length <= 2) {
11      console.log(0);
12      rl.close();
13      return;
14    }
15
16    // 去掉 [[ 和 ]]
17    line = line.slice(2, -2);
18    // 切分为行
19    const rowStrs = line.split("\n");
20    const grid = rowStrs.map(row => row.split(",").map(Number));
21
22    const n = grid.length;
23    const m = grid[0].length;
24    let res = Infinity;
25
26    for (let j = 0; j < m; j++) {
27      let colMax = -Infinity;
28      for (let i = 0; i < n; i++) {
29        colMax = Math.max(colMax, grid[i][j]);
30      }
31      res = Math.min(res, colMax);
32    }
33
34    console.log(res);
35    rl.close();
36  });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 求每列的最大值, 再取最小的那个
12 func solve(grid [][]int) int {
13     if len(grid) == 0 || len(grid[0]) == 0 {
14         return 0
15     }
16     n, m := len(grid), len(grid[0])
17     res := int(1e9)
18
19     for j := 0; j < m; j++ {
20         colMax := -1 << 31
21         for i := 0; i < n; i++ {
22             if grid[i][j] > colMax {
23                 colMax = grid[i][j]
24             }
25         }
26         if colMax < res {
27             res = colMax
28         }
29     }
30     return res
31 }
32
33 // 切分字符串
34 func split(s string, delim string) []string {
35     var result []string
36     for {
37         idx := strings.Index(s, delim)
38         if idx == -1 {
39             break
40         }
41         result = append(result, s[:idx])
42         s = s[idx+len(delim):]
43     }
44     result = append(result, s)
45     return result
}
```

```

46 }
47
48 func main() {
49     reader := bufio.NewReader(os.Stdin)
50     input, _ := reader.ReadString('\n')
51     input = strings.TrimSpace(input)
52
53     // 边界判断
54     if len(input) <= 2 {
55         fmt.Println(0)
56         return
57     }
58
59     // 去除头尾 "[" 和 "]"
60     input = input[2 : len(input)-2]
61
62     // 使用 "],[" 分割每一行
63     rowStrs := split(input, "],[")
64
65     var grid [][]int
66     for _, row := range rowStrs {
67         parts := strings.Split(row, ",")
68         var nums []int
69         for _, p := range parts {
70             val, err := strconv.Atoi(p)
71             if err != nil {
72                 fmt.Println(0)
73                 return
74             }
75             nums = append(nums, val)
76         }
77         grid = append(grid, nums)
78     }
79
80     fmt.Println(solve(grid))
81 }

```

来自: [华为OD机试2025 B卷 – 矩阵元素的边界值 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

来自: [华为OD机试2025 B卷 – 矩阵元素的边界值 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

华为OD机试2025 B卷 - 非严格递增连续数字序列 (C++ & Python & JAVA & JS & GO)-CSDN博客

非严格递增连续数字序列

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

输入一个字符串仅包含大小写字母和数字，求字符串中包含的最长的非严格递增连续数字序列的长度，（比如12234属于非严格递增连续数字序列）。

输入描述

输入一个字符串仅包含大小写字母和数字，输入的字符串最大不超过255个字符。

输出描述

最长的非严格递增连续数字序列的长度

示例1

输入

▼	Plain Text
1	abc2234019A334bc

输出

▼	Plain Text
1	4

说明

2234为最长的非严格递增连续数字序列，所以长度为4。

题解

思路： 栈模拟

1. 使用栈模拟来实现，使用数组来模拟栈，在栈中维持一个连续非严格递增的序列，定义 `pos` 作为栈的索引。使用 `res` 记录出现过最长的非严格递增连续数字序列的长度。
2. 具体逻辑：从前往后遍历输入字符串，根据不同字符进行处理情况如下
 - 当前字符为数字字符
 - 当前栈为空 或者 当前字符 \geq 栈顶字符，将当前字符加入栈中，并将 `pos++`
 - 当前字符 $<$ 栈顶字符 时，说明不满足连续非递增序列，更新 `res = max(res, pos)` .并将栈清空 `pos = 0` ,此时当前数字字符会形成一个新的序列，将当前字符压入栈中，并更新 `pos = 1`
 - 当前字符为字母字符，说明不满足连续非递增序列。更新 `res = max(res, pos)` .并将栈清空 `pos = 0`
3. 执行2的逻辑，遍历完输入字符串后，需要进行收尾操作。更新 `res = max(res, pos)` 。这主要是考虑输入字符串最后一段满足连续非严格递增序列。
4. 输出记录的 `res` 就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<cmath>
8  #include<map>
9
10 using namespace std;
11
12 int main (){
13     string input;
14     getline(cin, input);
15     int res = 0;
16     // 数组模拟栈 维持一个非严格递增的
17     vector<char> st(255);
18     // 记录数组已使用长度
19     int pos = 0;
20
21     int n = input.size();
22     for (int i = 0; i < n; i++) {
23         char c = input[i];
24         // 数字字符
25         if (c <= '9' && c >= '0') {
26             // 空或者 不小于
27             if (pos == 0 || st[pos-1] <= c) {
28                 st[pos] = c;
29                 pos++;
30
31             } else {
32                 res = max(res, pos);
33                 // 构建新的序列
34                 st[0] = c;
35                 pos = 1;
36             }
37             // 字母 会终端数字序列
38         } else {
39             res = max(res, pos);
40             pos = 0;
41         }
42     }
43     // 收尾操作
44     res = max(res, pos);
45     cout << res;
```

Java

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          String input = sc.nextLine();
7
8          int res = 0;
9          // 用字符数组模拟栈，维护一个非严格递增的数字序列
10         char[] st = new char[255];
11         int pos = 0;
12
13         for (int i = 0; i < input.length(); i++) {
14             char c = input.charAt(i);
15             if (Character.isDigit(c)) {
16                 if (pos == 0 || st[pos - 1] <= c) {
17                     st[pos++] = c;
18                 } else {
19                     res = Math.max(res, pos);
20                     st[0] = c;
21                     pos = 1;
22                 }
23             } else {
24                 // 非数字字符终止序列
25                 res = Math.max(res, pos);
26                 pos = 0;
27             }
28         }
29         res = Math.max(res, pos);
30         System.out.println(res);
31     }
32 }
```

Python


```
1 def main():
2     input_str = input()
3     res = 0
4     # 用列表模拟栈，维护非严格递增的数字序列
5     st = [''] * 255
6     pos = 0
7
8     for c in input_str:
9         if '0' <= c <= '9':
10             if pos == 0 or st[pos - 1] <= c:
11                 st[pos] = c
12                 pos += 1
13             else:
14                 res = max(res, pos)
15                 st[0] = c
16                 pos = 1
17         else:
18             # 非数字字符终止当前序列
19             res = max(res, pos)
20             pos = 0
21
22     res = max(res, pos)
23     print(res)
24
25 if __name__ == "__main__":
26     main()
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  rl.on('line', (line) => {
9    const input = line.trim();
10    let res = 0;
11    // 模拟栈，维护非严格递增数字序列
12    const st = new Array(255);
13    let pos = 0;
14
15    for (let i = 0; i < input.length; i++) {
16      const c = input[i];
17      if (c >= '0' && c <= '9') {
18        if (pos === 0 || st[pos - 1] <= c) {
19          st[pos++] = c;
20        } else {
21          res = Math.max(res, pos);
22          st[0] = c;
23          pos = 1;
24        }
25      } else {
26        // 非数字字符终止序列
27        res = Math.max(res, pos);
28        pos = 0;
29      }
30    }
31
32    res = Math.max(res, pos);
33    console.log(res);
34    rl.close();
35  });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7  )
8
9  func main() {
10     reader := bufio.NewReader(os.Stdin)
11     input, _ := reader.ReadString('\n')
12     input = input[:len(input)-1] // 去掉换行符
13
14     res := 0
15     // 模拟栈，维护非严格递增数字序列
16     st := make([]byte, 255)
17     pos := 0
18
19     for i := 0; i < len(input); i++ {
20         c := input[i]
21         if c >= '0' && c <= '9' {
22             if pos == 0 || st[pos-1] <= c {
23                 st[pos] = c
24                 pos++
25             } else {
26                 if pos > res {
27                     res = pos
28                 }
29                 st[0] = c
30                 pos = 1
31             }
32             // 非数字字符终止序列
33         } else {
34             if pos > res {
35                 res = pos
36             }
37             pos = 0
38         }
39     }
40     if pos > res {
41         res = pos
42     }
43     fmt.Println(res)
44 }
```

非严格递增连续数字序列

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

输入一个字符串仅包含大小写字母和数字，求字符串中包含的最长的非严格递增连续数字序列的长度，（比如12234属于非严格递增连续数字序列）。

输入描述

输入一个字符串仅包含大小写字母和数字，输入的字符串最大不超过255个字符。

输出描述

最长的非严格递增连续数字序列的长度

示例1

输入

▼ Plain Text |

1 abc2234019A334bc

输出

▼ Plain Text |

1 4

说明

2234为最长的非严格递增连续数字序列，所以长度为4。

题解

思路： 栈模拟

- 使用栈模拟来实现，使用数组来模拟栈，在栈中维持一个连续非严格递增的序列，定义 `pos` 作为栈的索引。使用 `res` 记录出现过最长的非严格递增连续数字序列的长度。
- 具体逻辑：从前往后遍历输入字符串，根据不同字符进行处理情况如下
 - 当前字符为数字字符
 - 当前栈为空 或者 当前字符 `>=` 栈顶字符，将当前字符加入栈中，并将`pos++`
 - 当前字符 `<` 栈顶字符 时，说明不满足连续非递增序列，更新 `res = max(res,`

`pos)` .并将栈清空 `pos = 0` ,此时当前数字字符会形成一个新的序列, 将当前字符压入栈中, 并更新 `pos = 1`

- 当前字符为字母字符, 说明不满足连续非递增序列。更新 `res = max(res, pos)` .并将栈清空 `pos = 0`

3. 执行2的逻辑, 遍历完输入字符串后, 需要进行收尾操作。更新 `res = max(res, pos)` 。这主要是考虑输入字符串最后一段满足连续非严格递增序列。

4. 输出记录的 `res` 就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<cmath>
8  #include<map>
9
10 using namespace std;
11
12 int main (){
13     string input;
14     getline(cin, input);
15     int res = 0;
16     // 数组模拟栈 维持一个非严格递增的
17     vector<char> st(255);
18     // 记录数组已使用长度
19     int pos = 0;
20
21     int n = input.size();
22     for (int i = 0; i < n; i++) {
23         char c = input[i];
24         // 数字字符
25         if (c <= '9' && c >= '0') {
26             // 空或者 不小于
27             if (pos == 0 || st[pos-1] <= c) {
28                 st[pos] = c;
29                 pos++;
30             } else {
31                 res = max(res, pos);
32                 // 构建新的序列
33                 st[0] = c;
34                 pos = 1;
35             }
36         }
37         // 字母 会终端数字序列
38     } else {
39         res = max(res, pos);
40         pos = 0;
41     }
42 }
43 // 收尾操作
44 res = max(res, pos);
45 cout << res;
```

Java

▼ Plain Text

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          String input = sc.nextLine();
7
8          int res = 0;
9          // 用字符数组模拟栈，维护一个非严格递增的数字序列
10         char[] st = new char[255];
11         int pos = 0;
12
13         for (int i = 0; i < input.length(); i++) {
14             char c = input.charAt(i);
15             if (Character.isDigit(c)) {
16                 if (pos == 0 || st[pos - 1] <= c) {
17                     st[pos++] = c;
18                 } else {
19                     res = Math.max(res, pos);
20                     st[0] = c;
21                     pos = 1;
22                 }
23             } else {
24                 // 非数字字符终止序列
25                 res = Math.max(res, pos);
26                 pos = 0;
27             }
28         }
29         res = Math.max(res, pos);
30         System.out.println(res);
31     }
32 }
```

Python

```
1 def main():
2     input_str = input()
3     res = 0
4     # 用列表模拟栈，维护非严格递增的数字序列
5     st = [''] * 255
6     pos = 0
7
8     for c in input_str:
9         if '0' <= c <= '9':
10             if pos == 0 or st[pos - 1] <= c:
11                 st[pos] = c
12                 pos += 1
13             else:
14                 res = max(res, pos)
15                 st[0] = c
16                 pos = 1
17         else:
18             # 非数字字符终止当前序列
19             res = max(res, pos)
20             pos = 0
21
22     res = max(res, pos)
23     print(res)
24
25 if __name__ == "__main__":
26     main()
```

JavaScript


```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  rl.on('line', (line) => {
9    const input = line.trim();
10    let res = 0;
11    // 模拟栈，维护非严格递增数字序列
12    const st = new Array(255);
13    let pos = 0;
14
15    for (let i = 0; i < input.length; i++) {
16      const c = input[i];
17      if (c >= '0' && c <= '9') {
18        if (pos === 0 || st[pos - 1] <= c) {
19          st[pos++] = c;
20        } else {
21          res = Math.max(res, pos);
22          st[0] = c;
23          pos = 1;
24        }
25      } else {
26        // 非数字字符终止序列
27        res = Math.max(res, pos);
28        pos = 0;
29      }
30    }
31
32    res = Math.max(res, pos);
33    console.log(res);
34    rl.close();
35  });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7  )
8
9  func main() {
10     reader := bufio.NewReader(os.Stdin)
11     input, _ := reader.ReadString('\n')
12     input = input[:len(input)-1] // 去掉换行符
13
14     res := 0
15     // 模拟栈，维护非严格递增数字序列
16     st := make([]byte, 255)
17     pos := 0
18
19     for i := 0; i < len(input); i++ {
20         c := input[i]
21         if c >= '0' && c <= '9' {
22             if pos == 0 || st[pos-1] <= c {
23                 st[pos] = c
24                 pos++
25             } else {
26                 if pos > res {
27                     res = pos
28                 }
29                 st[0] = c
30                 pos = 1
31             }
32             // 非数字字符终止序列
33         } else {
34             if pos > res {
35                 res = pos
36             }
37             pos = 0
38         }
39     }
40     if pos > res {
41         res = pos
42     }
43     fmt.Println(res)
44 }
```

来自: [华为OD机试2025 B卷 – 非严格递增连续数字序列 \(C++ & Python & JAVA & JS & GO\)–CSDN 博客](#)

来自: [华为OD机试2025 B卷 – 非严格递增连续数字序列 \(C++ & Python & JAVA & JS & GO\)–CSDN 博客](#)

华为OD机试2025 B卷 - 数列描述 (C++ & Python & JAVA & JS & GO)-CSDN博客

数列描述

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

有一个数列 $a[N]$ ($N=60$), 从 $a[0]$ 开始, 每一项都是一个数字。数列中 $a[n+1]$ 都是 $a[n]$ 的描述。其中 $a[0]=1$ 。规则如下:

- 1. $a[0]:1$
- 2. $a[1]:11$ (含义: 其前一项 $a[0]=1$ 是1个1, 即“11”。表示 $a[0]$ 从左到右, 连续出现了1次“1”)
- 3. $a[2]:21$ (含义: 其前一项 $a[1]=11$, 从左到右: 是由两个1组成, 即“21”。表示 $a[1]$ 从左到右, 连续出现了两次“1”)
- 4. $a[3]:1211$ (含义: 其前一项 $a[2]=21$, 从左到右: 是由一个2和一个1组成, 即“1211”。表示 $a[2]$ 从左到右, 连续出现了1次“2”, 然后又连续出现了1次“1”)
- 5. $a[4]:111221$ (含义: 其前一项 $a[3]=1211$, 从左到右: 是由一个1、一个2、两个1组成, 即“111221”。表示 $a[3]$ 从左到右, 连续出现了1次“1”, 连续出现了1次“2”, 连续出现了两次“1”)

请输出这个数列的第 n 项结果 ($a[n]$, $0 \leq n \leq 59$)。

输入描述

数列的第 n 项($0 \leq n \leq 59$)。

输出描述

数列的内容

示例1

输入

Plain Text

14

输出



Plain Text

```
1  111221
```

题解

思路： 模拟

1. 根据题目描述可以得知 下一个的值依赖上一个值。
2. 初始定义 `tmp = "1"` ,根据题目规则依次计算[1,n]的值。
3. 最后输出n的值即可。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <sstream>
5  using namespace std;
6
7  // 生成下一个数列
8  string getNext(const string& tmp) {
9      ostringstream oss;
10     // 连续数量
11     int count = 1;
12     char c = tmp[0];
13
14     for (int i = 1; i < tmp.size(); i++) {
15         if (tmp[i] == c) {
16             count++;
17         } else {
18             oss << count << c;
19             c = tmp[i];
20             count = 1;
21         }
22     }
23     oss << count << c; // 处理最后一段
24     return oss.str();
25 }
26
27 int main() {
28     int n;
29     cin >> n;
30     string tmp = "1";
31     // 迭代
32     for (int i = 1; i <= n; i++) {
33         tmp = getNext(tmp);
34     }
35
36     cout << tmp << endl;
37     return 0;
38 }
```

Java

```
1  import java.util.Scanner;
2
3  public class Main {
4      // 生成下一个数列
5      static String getNext(String tmp) {
6          StringBuilder sb = new StringBuilder();
7          int count = 1;
8          char c = tmp.charAt(0);
9          for (int i = 1; i < tmp.length(); i++) {
10             if (tmp.charAt(i) == c) {
11                 count++;
12             } else {
13                 sb.append(count).append(c);
14                 c = tmp.charAt(i);
15                 count = 1;
16             }
17         }
18         sb.append(count).append(c); // 处理最后一段
19         return sb.toString();
20     }
21
22     public static void main(String[] args) {
23         Scanner sc = new Scanner(System.in);
24         int n = sc.nextInt();
25         String tmp = "1";
26         // 迭代生成
27         for (int i = 1; i <= n; i++) {
28             tmp = getNext(tmp);
29         }
30         System.out.println(tmp);
31     }
32 }
```

Python

```
1  # 生成下一个数列
2  def get_next(tmp):
3      res = []
4      count = 1
5      c = tmp[0]
6      for i in range(1, len(tmp)):
7          if tmp[i] == c:
8              count += 1
9          else:
10             res.append(f"{count}{c}")
11             c = tmp[i]
12             count = 1
13     res.append(f"{count}{c}") # 处理最后一段
14     return ''.join(res)
15
16 def main():
17     n = int(input())
18     tmp = "1"
19     # 迭代生成
20     for _ in range(1, n + 1):
21         tmp = get_next(tmp)
22     print(tmp)
23
24 if __name__ == "__main__":
25     main()
```

JavaScript


```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let input = [];
9
10 rl.on('line', line => {
11   input.push(line.trim());
12 }).on('close', () => {
13   const n = parseInt(input[0]);
14   let tmp = '1';
15
16   // 生成下一个数列
17   function getNext(str) {
18     let res = '';
19     let count = 1;
20     let c = str[0];
21     for (let i = 1; i < str.length; i++) {
22       if (str[i] === c) {
23         count++;
24       } else {
25         res += count + c;
26         c = str[i];
27         count = 1;
28       }
29     }
30     res += count + c; // 处理最后一段
31     return res;
32   }
33
34   for (let i = 1; i <= n; i++) {
35     tmp = getNext(tmp);
36   }
37
38   console.log(tmp);
39 });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11 // 生成下一个数列
12 func getNext(tmp string) string {
13     var sb strings.Builder
14     count := 1
15     c := tmp[0]
16     for i := 1; i < len(tmp); i++ {
17         if tmp[i] == c {
18             count++
19         } else {
20             sb.WriteString(strconv.Itoa(count))
21             sb.WriteByte(c)
22             c = tmp[i]
23             count = 1
24         }
25     }
26     sb.WriteString(strconv.Itoa(count))
27     sb.WriteByte(c)
28     return sb.String()
29 }
30
31 func main() {
32     scanner := bufio.NewScanner(os.Stdin)
33     scanner.Scan()
34     n, _ := strconv.Atoi(scanner.Text())
35     tmp := "1"
36     // 迭代生成
37     for i := 1; i <= n; i++ {
38         tmp = getNext(tmp)
39     }
40     fmt.Println(tmp)
41 }
```


华为OD机试 2025B卷 - 完全数计算 (C++ & Python & JAVA & JS & GO)-CSDN博客

2025B卷目录点击查看：[华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

2025B卷 100分题型

题目描述

完全数（Perfect number），又称完美数或完备数，是一些特殊的自然数。它所有的真因子（即除了自身以外的约数）的和（即因子函数），恰好等于它本身。例如：28，它有约数1、2、4、7、14、28，除去它本身28外，其余5个数相加， $1+2+4+7+14=28$ 。输入n，请输出n以内(含n)完全数的个数。

输入描述

输入一个数字n。

输出描述

输出不超过n的完全数的个数。

用例1

输入

▼ Plain Text

1 6

输出

▼ Plain Text

1 1

题解

思路：模拟

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<queue>
9  #include<map>
10 #include<set>
11 using namespace std;
12
13 // 判断是否为完全数
14 bool judge(int num) {
15     int sum = 1;
16     for (int i = 2; i * i <= num; i++) {
17         if (num % i == 0) {
18             sum += i;
19             if (i != num / i) {
20                 sum += num / i;
21             }
22         }
23     }
24     return sum == num;
25 }
26
27
28 int main() {
29     int n;
30     int res = 0;
31     cin >> n;
32     for (int i = 2; i <= n; i++) {
33         if (judge(i)) {
34             res++;
35             cout << i << endl;
36         }
37     }
38     cout << res;
39     return 0;
40 }
```

JAVA

```
1  import java.util.Scanner;
2
3  public class Main {
4      // 判断是否为完全数
5      public static boolean judge(int num) {
6          int sum = 1;
7          for (int i = 2; i * i <= num; i++) {
8              if (num % i == 0) {
9                  sum += i;
10                 if (i != num / i) {
11                     sum += num / i;
12                 }
13             }
14         }
15         return sum == num;
16     }
17
18     public static void main(String[] args) {
19         Scanner sc = new Scanner(System.in);
20         int n = sc.nextInt();
21         int res = 0;
22         for (int i = 2; i <= n; i++) {
23             if (judge(i)) {
24                 res++;
25                 System.out.println(i);
26             }
27         }
28         System.out.println(res);
29     }
30 }
```

Python

```
1  # 判断是否为完全数
2  def judge(num):
3      total = 1
4      for i in range(2, int(num**0.5) + 1):
5          if num % i == 0:
6              total += i
7              if i != num // i:
8                  total += num // i
9      return total == num
10
11 if __name__ == "__main__":
12     n = int(input())
13     res = 0
14     for i in range(2, n + 1):
15         if judge(i):
16             res += 1
17             print(i)
18     print(res)
```

JavaScript

```
1  // 判断是否为完全数
2  function judge(num) {
3      let sum = 1;
4      for (let i = 2; i * i <= num; i++) {
5          if (num % i === 0) {
6              sum += i;
7              if (i !== num / i) {
8                  sum += num / i;
9              }
10         }
11     }
12     return sum === num;
13 }
14
15 const readline = require('readline');
16
17 const rl = readline.createInterface({
18     input: process.stdin,
19     output: process.stdout
20 });
21
22 let input = '';
23 rl.on('line', function(line) {
24     input = parseInt(line.trim());
25     let res = 0;
26     for (let i = 2; i <= input; i++) {
27         if (judge(i)) {
28             res++;
29             console.log(i);
30         }
31     }
32     console.log(res);
33     rl.close();
34 });
```

Go


```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11 // 判断是否为完全数
12 func judge(num int) bool {
13     sum := 1
14     for i := 2; i*i <= num; i++ {
15         if num%i == 0 {
16             sum += i
17             if i != num/i {
18                 sum += num / i
19             }
20         }
21     }
22     return sum == num
23 }
24
25 func main() {
26     reader := bufio.NewReader(os.Stdin)
27     line, _ := reader.ReadString('\n')
28     line = strings.TrimSpace(line)
29     n, _ := strconv.Atoi(line)
30
31     res := 0
32     for i := 2; i <= n; i++ {
33         if judge(i) {
34             res++
35             fmt.Println(i)
36         }
37     }
38     fmt.Println(res)
39 }
```

华为OD机试 2025 B卷 - 端口合并 (C++ & Python & JAVA & JS & GO)_华为od说考牛客网b卷-CSDN博客

端口合并

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

有M个端口组($1 \leq M \leq 10$),
每个端口组是长度为N的整数数组($1 \leq N \leq 100$),
如果端口组间存在2个及以上不同端口相同,则认为这2个端口组互相关联,可以合并。

输入描述

第一行输入端口组个数M,再输入M行,每行逗号分割,代表端口组。
备注:端口组内数字可以重复。

输出描述

输出合并后的端口组,用二维数组表示。

- 组内相同端口仅保留一个,从小到大排序。
- 组外顺序保持输入顺序

备注:M,N不在限定范围内,统一输出一组空数组[[[]]]

用例1

输入

▼ Plain Text

```
1 4
2 4
3 2,3,2
4 1,2
5 5
```

输出

▼	Plain Text
1	[[4],[2,3],[1,2],[5]]

说明

仅有一个端口2相同，不可以合并。

用例2

输入

▼	Plain Text
1	3
2	2,3,1
3	4,3,2
4	5

输出

▼	Plain Text
1	[[1,2,3,4],[5]]

用例3

输入

▼	Plain Text
1	6
2	10
3	4,2,1
4	9
5	3,6,9,2
6	6,3,4
7	8

输出

▼	Plain Text
1	[[10],[1,2,3,4,6,9],[9],[8]]

用例4

输入

▼	Plain Text
1	11

输出

▼	Plain Text
1	[[[]]]

题解

思路: 模拟

1. 接收输入数据，判断输入数据中对应的 `n m` 是否属于题目要求的范围，不属于则直接输出 `[[[]]]`
2. 将所有输入的端口组内部按照端口大小进行升序排序，便于后期判断相同端口数量使用。
3. 接下来进行循环迭代端口组合并，枚举两个不同的端口组判断相同端口数量，如果相同端口数量 ≥ 2 ，则进行合并端口组，并移除被合并端口组。
4. 重复3的逻辑，直到任意两个端口组不能合并时结束循环合并。
5. 输出结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<queue>
9  #include<set>
10 using namespace std;
11
12 // 通用 split 函数
13 vector<string> split(const string& str, const string& delimiter) {
14     vector<string> result;
15     size_t start = 0;
16     size_t end = str.find(delimiter);
17     while (end != string::npos) {
18         result.push_back(str.substr(start, end - start));
19         start = end + delimiter.length();
20         end = str.find(delimiter, start);
21     }
22     // 添加最后一个部分
23     result.push_back(str.substr(start));
24     return result;
25 }
26
27 // 判断相同端口数量是否>=2
28 bool countSame(vector<int>& v1, vector<int>& v2) {
29     int count = 0;
30     int i = 0;
31     int j = 0;
32
33     while (i < v1.size() && j < v2.size()) {
34         if (v1[i] < v2[j]) {
35             i++;
36         } else if (v1[i] > v2[j]) {
37             j++;
38         } else {
39             count++;
40             i += 1;
41             j += 1;
42             if (count == 2) {
43                 return true;
44             }
45         }
46     }
```

```

46     }
47     return false;
48 }
49
50 // 递归合并
51 bool DFS(vector<vector<int>>& ans) {
52     bool flag = false;
53     for (int i = ans.size()-1; i >= 0; i--) {
54         for (int j = i -1; j >= 0; j--) {
55             if (countSame(ans[i], ans[j])) {
56                 // 合并
57                 ans[j].insert(ans[j].end(),ans[i].begin(), ans[i].end());
58                 // 移除被合并的数量
59                 ans.erase(ans.begin() + i);
60                 // 排序
61                 sort(ans[j].begin(), ans[j].end());
62                 flag = true;
63                 break;
64             }
65         }
66     }
67     return flag;
68 }
69
70
71 int main() {
72     int n;
73     cin >> n;
74     cin.ignore();
75     // 判断是否超过范围
76     if (n > 10 || n < 1) {
77         cout << "[[]]";
78         return 0;
79     }
80     vector<vector<int>> ans(n);
81     for (int i = 0; i < n; i++) {
82         string s;
83         getline(cin, s);
84         vector<string> tmp = split(s, ",");
85         int arrSize = tmp.size();
86         // 判断是否超过范围
87         if (arrSize < 1 || arrSize > 100) {
88             cout << "[[]]";
89             return 0;
90         }
91         for (int j = 0; j < arrSize; j++) {
92             ans[i].push_back(stoi(tmp[j]));
93         }

```

```

94         // 升序判断 便于后面进行判断
95         sort(ans[i].begin(), ans[i].end());
96     }
97
98     bool flag = true;
99     // 循环递归进行合并
100    while (flag) {
101        flag = DFS(ans);
102    }
103
104    string res = "[";
105    for (int i = 0; i < ans.size(); i++) {
106        string cuurrStr = "[";
107        set<int> tmp(ans[i].begin(), ans[i].end());
108        for (auto x : tmp) {
109            cuurrStr += to_string(x) + ",";
110        }
111        if (!ans[i].empty()) {
112            cuurrStr.pop_back();
113        }
114
115        cuurrStr += + "],";
116        res += cuurrStr;
117    }
118    if (!ans.empty()) {
119        res.pop_back();
120    }
121    res += "]";
122    cout << res;
123    return 0;
124 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5
6      // 判断相同端口数量是否 >= 2
7      public static boolean countSame(List<Integer> v1, List<Integer> v2) {
8          int count = 0, i = 0, j = 0;
9          while (i < v1.size() && j < v2.size()) {
10             if (v1.get(i) < v2.get(j)) {
11                 i++;
12             } else if (v1.get(i) > v2.get(j)) {
13                 j++;
14             } else {
15                 count++;
16                 i++;
17                 j++;
18                 if (count == 2) {
19                     return true;
20                 }
21             }
22         }
23         return false;
24     }
25
26     // 递归合并
27     public static boolean DFS(List<List<Integer>> ans) {
28         boolean flag = false;
29         for (int i = ans.size() - 1; i >= 0; i--) {
30             for (int j = i - 1; j >= 0; j--) {
31                 if (countSame(ans.get(i), ans.get(j))) {
32                     // 合并
33                     ans.get(j).addAll(ans.get(i));
34                     // 移除被合并端口组
35                     ans.remove(i);
36                     // 排序
37                     Collections.sort(ans.get(j));
38                     flag = true;
39                     break;
40                 }
41             }
42         }
43         return flag;
44     }
45 }
```



```

46 public static void main(String[] args) {
47     Scanner scanner = new Scanner(System.in);
48     int n = Integer.parseInt(scanner.nextLine().trim());
49     // 判断是否超出范围
50     if (n > 10 || n < 1) {
51         System.out.println("[[]]");
52         return;
53     }
54
55     List<List<Integer>> ans = new ArrayList<>();
56     for (int i = 0; i < n; i++) {
57         String line = scanner.nextLine().trim();
58         List<String> tmp = Arrays.asList(line.split(","));
59         // 判断是否超出范围
60         if (tmp.size() < 1 || tmp.size() > 100) {
61             System.out.println("[[]]");
62             return;
63         }
64
65         List<Integer> numbers = new ArrayList<>();
66         for (String s : tmp) {
67             numbers.add(Integer.parseInt(s));
68         }
69         // 排序便于后期判断相同个数
70         Collections.sort(numbers);
71         ans.add(numbers);
72     }
73
74     boolean flag = true;
75     // 循环迭代合并
76     while (flag) {
77         flag = DFS(ans);
78     }
79
80     StringBuilder res = new StringBuilder("[");
81     for (List<Integer> group : ans) {
82         res.append("[");
83         Set<Integer> unique = new TreeSet<>(group);
84         for (int num : unique) {
85             res.append(num).append(",");
86         }
87         if (!group.isEmpty()) {
88             res.setLength(res.length() - 1); // 移除最后的逗号
89         }
90         res.append("],");
91     }
92     if (!ans.isEmpty()) {
93         res.setLength(res.length() - 1); // 移除最后的逗号

```

```
94         }  
95         res.append("]");  
96         System.out.println(res);  
97     }  
98 }
```

Python

```
1  import sys
2
3
4  # 判断相同端口数量是否 >= 2
5  def count_same(v1, v2):
6      count, i, j = 0, 0, 0
7      while i < len(v1) and j < len(v2):
8          if v1[i] < v2[j]:
9              i += 1
10         elif v1[i] > v2[j]:
11             j += 1
12         else:
13             count += 1
14             i += 1
15             j += 1
16             if count == 2:
17                 return True
18         return False
19
20 # 递归合并
21 def dfs(ans):
22     flag = False
23     for i in range(len(ans) - 1, -1, -1):
24         for j in range(i - 1, -1, -1):
25             if count_same(ans[i], ans[j]):
26                 # 合并
27                 ans[j].extend(ans[i])
28                 # 移除被合并端口组
29                 ans.pop(i)
30                 # 排序
31                 ans[j].sort()
32                 flag = True
33                 break
34     return flag
35
36 def main():
37     n = int(sys.stdin.readline().strip())
38     # 判断是否不属于限定范围
39     if n > 10 or n < 1:
40         print("[[]]")
41         return
42
43     ans = []
44     for _ in range(n):
45         line = sys.stdin.readline().strip()
```

```

46         tmp = line.split(",")
47         # 判断是否不属于限定范围
48         if len(tmp) < 1 or len(tmp) > 100:
49             print("[[]]")
50             return
51
52         numbers = list(map(int, tmp))
53         # 升序排序 便于后期进行判断相同个数
54         numbers.sort()
55         ans.append(numbers)
56
57     flag = True
58     # 循环迭代合并
59     while flag:
60         flag = dfs(ans)
61
62     res = "["
63     for group in ans:
64         res += "[" + ",".join(map(str, sorted(set(group)))) + "], "
65     res = res.rstrip(",") + "]"
66     print(res)
67
68 if __name__ == "__main__":
69     main()

```

JavaScript

```
1  const readline = require("readline");
2
3
4
5  // 判断两个数组中相同端口是否 >= 2
6  function countSame(v1, v2) {
7      let count = 0, i = 0, j = 0;
8      while (i < v1.length && j < v2.length) {
9          if (v1[i] < v2[j]) {
10             i++;
11         } else if (v1[i] > v2[j]) {
12             j++;
13         } else {
14             count++;
15             i++;
16             j++;
17             if (count === 2) return true;
18         }
19     }
20     return false;
21 }
22
23 // 合并具有两个及以上公共元素的集合
24 function DFS(ans) {
25     let flag = false;
26     for (let i = ans.length - 1; i >= 0; i--) {
27         for (let j = i - 1; j >= 0; j--) {
28             if (countSame(ans[i], ans[j])) {
29                 // 合并端口组
30                 ans[j] = ans[j].concat(ans[i]).sort((a, b) => a - b);
31                 // 移除被合并的端口组
32                 ans.splice(i, 1);
33                 flag = true;
34                 break;
35             }
36         }
37     }
38     return flag;
39 }
40
41 // 主函数
42 function main(inputLines) {
43     let n = parseInt(inputLines[0]);
44     // 是否不属于限定范围
45     if (n > 10 || n < 1) {
```

```

46         console.log("[[]]");
47         return;
48     }
49
50     let ans = [];
51     for (let i = 1; i <= n; i++) {
52         let tmp = inputLines[i].split(",").map(Number);
53         // 是否不属于限定范围
54         if (tmp.length < 1 || tmp.length > 100) {
55             console.log("[[]]");
56             return;
57         }
58         // 升序排序 便于后期进行相同端口数判断
59         ans.push(tmp.sort((a, b) => a - b));
60     }
61
62     let flag = true;
63     // 循环迭代进行判断
64     while (flag) {
65         flag = DFS(ans);
66     }
67
68     let res = "[" + ans.map(arr => "[" + [...new Set(arr)].join(",") +
69 "]" ).join(",") + "]";
70     console.log(res);
71 }
72
73 // 读取输入
74 const rl = readline.createInterface({
75     input: process.stdin,
76     output: process.stdout
77 });
78
79 let inputLines = [];
80
81 rl.on("line", function (line) {
82     inputLines.push(line.trim());
83 });
84
85 rl.on("close", function () {
86     main(inputLines);
87 });

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12
13
14 // 判断相同端口数量是否 >= 2
15 func countSame(v1, v2 []int) bool {
16     count, i, j := 0, 0, 0
17     for i < len(v1) && j < len(v2) {
18         if v1[i] < v2[j] {
19             i++
20         } else if v1[i] > v2[j] {
21             j++
22         } else {
23             count++
24             i++
25             j++
26             if count == 2 {
27                 return true
28             }
29         }
30     }
31     return false
32 }
33
34 // 递归合并
35 func DFS(ans *[][]int) bool {
36     flag := false
37     for i := len(*ans) - 1; i >= 0; i-- {
38         for j := i - 1; j >= 0; j-- {
39             if countSame((*ans)[i], (*ans)[j]) {
40                 (*ans)[j] = append((*ans)[j], (*ans)[i]...)
41                 (*ans) = append((*ans)[:i], (*ans)[i+1:]...) // 删除合并的元素
42                 sort.Ints((*ans)[j]) // 对合并后的元素排序
43                 flag = true
44                 break
45             }
46         }
47     }
48     return flag
49 }
```

```

46     }
47 }
48 return flag
49 }
50
51 func main() {
52     // 使用 bufio 读取标准输入, 提高性能
53     scanner := bufio.NewScanner(os.Stdin)
54     scanner.Scan()
55     n, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
56     // 判断是否属于范围内
57     if n > 10 || n < 1 {
58         fmt.Println("[[]]")
59         return
60     }
61
62     ans := make([][]int, 0, n)
63
64     // 读取输入
65     for i := 0; i < n; i++ {
66         scanner.Scan()
67         line := strings.TrimSpace(scanner.Text())
68         tmp := strings.Split(line, ",")
69         // 判断是否属于范围内
70         if len(tmp) < 1 || len(tmp) > 100 {
71             fmt.Println("[[]]")
72             return
73         }
74
75         numbers := make([]int, len(tmp))
76         for j, s := range tmp {
77             numbers[j], _ = strconv.Atoi(s)
78         }
79         // 升序 便于后期进行相同端口数判断
80         sort.Ints(numbers)
81         ans = append(ans, numbers)
82     }
83
84     // 递归合并
85     for DFS(&ans) {
86     }
87
88     // 构造输出结果
89     var sb strings.Builder
90     sb.WriteString("[")
91     for _, group := range ans {
92         sb.WriteString("[")
93         uniqueSet := make(map[int]bool)

```



```

94     first := true
95     for _, num := range group {
96         if !uniqueSet[num] {
97             if !first {
98                 sb.WriteString(",")
99             }
100            sb.WriteString(strconv.Itoa(num))
101            uniqueSet[num] = true
102            first = false
103        }
104    }
105    sb.WriteString("],")
106 }
107
108 sbStr := sb.String()
109 if len(sbStr) > 1 {
110     sbStr = sbStr[:len(sbStr)-1] // 去掉最后一个逗号
111 }
112 sbStr += "]"
113
114 fmt.Println(sbStr)
115 }

```

来自: [华为OD机试 2025 B卷 – 端口合并 \(C++ & Python & JAVA & JS & GO\)_华为od说考牛客网b卷-CSDN博客](#)

华为od机试 2025 B卷 - 分积木 (C++ & Python & JAVA & JS & GO)-CSDN博客

分积木

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

Solo和koko是两兄弟，妈妈给了他们一大堆积木，每块积木上都有自己的重量。
现在他们想要将这些积木分成两堆。哥哥Solo负责分配，弟弟koko要求两个人获得的积木总重量“相等”（根据Koko的逻辑），个数可以不同，不然就会哭，但koko只会先将两个数转成二进制再进行加法，而且总会忘记进位（每个进位都忘记）。如当25（11101）加11（01011）时，koko得到的计算结果是18（10010）：

▼ Plain Text

```
1  11001
2  +01011
3  -----
4  10010
```

Solo想要尽可能使自己得到的积木总重量最大，且不让koko哭。

输入描述

第一行是一个整数N($2 \leq N \leq 100$)，表示有多少块积木；
第二行为空格分开的N个整数Ci($1 \leq Ci \leq 10^6$)，表示第i块积木的重量。

输出描述

如果能让koko不哭，输出Solo所能获得积木的最大总重量；否则输出“NO”。

用例1

输入

	Plain Text
1	3
2	3 5 6

输出

	Plain Text
1	11

题解

相同的题: [2025B卷 – 分苹果](#)

思路: 数学原理题

- 不进位的加法其实就是 异或计算 。
- 按照题目要求最终弟弟和哥哥 分的积木重量的异或和 要想相等, 假设为x, 那么所有积木的异或和 $x \oplus x = 0$,所有积木的异或和必须为 0, 否则输出NO.
- 另外一个数学定理 $x \oplus 0 = x$ (交换律),我们的目的是让各个获得最多重要的苹果, 所以给弟弟分一个最小重量积木 就行。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<queue>
9  #include<map>
10 #include<set>
11 using namespace std;
12
13 int main() {
14     int n;
15     int minValue = 10000009;
16     // 记录异或计算结果
17     int sum = 0;
18     // 记录实际重量和
19     int totalWeight = 0;
20     cin >> n;
21     for (int i = 0; i < n; i++) {
22         int tmp;
23         cin >> tmp;
24         sum ^= tmp;
25         minValue = min(minValue, tmp);
26         totalWeight += tmp;
27     }
28     // 不能平分
29     if (sum != 0) {
30         cout << "NO";
31     } else {
32         cout << totalWeight - minValue;
33     }
34     return 0;
35 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6
7          int n = sc.nextInt();
8          int sum = 0; // 异或和
9          int totalWeight = 0; // 总重量
10         int minValue = Integer.MAX_VALUE; // 最小值
11
12         for (int i = 0; i < n; i++) {
13             int tmp = sc.nextInt();
14             sum ^= tmp;
15             minValue = Math.min(minValue, tmp);
16             totalWeight += tmp;
17         }
18
19         if (sum != 0) {
20             System.out.println("NO");
21         } else {
22             System.out.println(totalWeight - minValue);
23         }
24     }
25 }
```

Python

```
1  # -*- coding: utf-8 -*-
2  n = int(input())
3  arr = list(map(int, input().split()))
4
5  sum_xor = 0 # 异或和
6  total_weight = 0 # 总重量
7  min_value = float('inf') # 最小值
8
9  for num in arr:
10     sum_xor ^= num
11     total_weight += num
12     min_value = min(min_value, num)
13
14  if sum_xor != 0:
15     print("NO")
16  else:
17     print(total_weight - min_value)
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let inputLines = [];
9  rl.on('line', line => {
10    inputLines.push(line.trim());
11  });
12
13  rl.on('close', () => {
14    const n = parseInt(inputLines[0]);
15    const arr = inputLines[1].split(' ').map(Number);
16
17    let sum = 0; // 异或和
18    let totalWeight = 0; // 总重量
19    let minValue = Infinity; // 最小值
20
21    for (let num of arr) {
22      sum ^= num;
23      totalWeight += num;
24      minValue = Math.min(minValue, num);
25    }
26
27    if (sum !== 0) {
28      console.log("NO");
29    } else {
30      console.log(totalWeight - minValue);
31    }
32  });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13
14     // 读取 n
15     line, _ := reader.ReadString('\n')
16     n, _ := strconv.Atoi(strings.TrimSpace(line))
17
18     // 读取数组
19     line, _ = reader.ReadString('\n')
20     parts := strings.Fields(line)
21
22     sum := 0           // 异或和
23     totalWeight := 0   // 总重量
24     minValue := 1000000009 // 重量最小值
25
26     for i := 0; i < n; i++ {
27         num, _ := strconv.Atoi(parts[i])
28         sum ^= num
29         totalWeight += num
30         if num < minValue {
31             minValue = num
32         }
33     }
34
35     if sum != 0 {
36         fmt.Println("NO")
37     } else {
38         fmt.Println(totalWeight - minValue)
39     }
40 }
```


华为OD机试 2025 B卷 - 服务失效判断 (C++ & Python & JAVA & JS & GO)-CSDN博客

服务失效判断

华为OD机试真题目录点击查看: [华为OD机试2025B卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

华为OD机试2025B卷 100分题型

题目描述

某系统中有众多服务，每个服务用字符串（只包含字母和数字，长度 ≤ 10 ）唯一标识，服务间可能有[依赖关系](#)，如A依赖B，则当B故障时导致A也故障。依赖具有传递性，如A依赖B，B依赖C，当C故障时导致B故障，也导致A故障。给出所有依赖关系，以及当前已知故障服务，要求输出所有正常服务。依赖关系：服务1-服务2 表示“服务1”依赖“服务2”不必考虑输入异常，用例保证：依赖关系列表、故障列表非空，且依赖关系数，故障服务数都不会超过3000，服务标识格式正常。

输入描述

- 半角逗号分隔的依赖关系列表（换行）
- 半角逗号分隔的故障服务列表

输出描述

- 依赖关系列表中提及的所有服务中可以正常工作的服务列表，用半角逗号分隔，按依赖关系列表中出现的次序排序。
- 特别的，没有正常节点输出单独一个半角逗号

用例1

输入

▼

Plain Text

```
1  a1-a2
2  a2
```

输出

▼	Plain Text
1	,

用例2

输入

▼	Plain Text
1	a1-a2,a5-a6,a2-a3
2	a5,a2

输出

▼	Plain Text
1	a6,a3

题解

思路： DFS + 哈希表

1. 使用哈希表记录出现过的服务名以及对应状态 `serviceStatus`，使用哈希表记录 被依赖服务 -> [服务] 的关系，使用 `serviceSequence` 哈希表记录服务出现的顺序。
2. 循环遍历失效服务，通过DFS修改依赖失效服务的服务状态为失效状态。
3. 通过服务状态哈希表汇总未失效服务，存入数组中。若不存在未失效服务，直接输出 `,`。
4. 存在未失效服务，将数组自定排序，排序规则按照服务出现顺序进行排序(借助之前 `serviceSequence`)实现。排序之后按顺序输出服务。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <sstream>
5  #include <unordered_map>
6  #include <unordered_set>
7  #include <algorithm>
8
9  using namespace std;
10
11 // split 函数
12 vector<pair<string, string>> parseDependencies(const string &str) {
13     vector<pair<string, string>> result;
14     size_t start = 0, end;
15     while ((end = str.find(',', start)) != string::npos) {
16         string dep = str.substr(start, end - start);
17         size_t mid = dep.find('-');
18         result.emplace_back(dep.substr(0, mid), dep.substr(mid + 1));
19         start = end + 1;
20     }
21     // 添加最后一个
22     string dep = str.substr(start);
23     size_t mid = dep.find('-');
24     result.emplace_back(dep.substr(0, mid), dep.substr(mid + 1));
25     return result;
26 }
27
28 // 递归传递失效状态
29 void dfs(unordered_map<string, bool> &serviceStatus, unordered_map<string, unordered_set<string>> &mp, const string &invalidService) {
30     if (!serviceStatus[invalidService]) return;
31     serviceStatus[invalidService] = false;
32     for (const auto &x : mp[invalidService]) {
33         dfs(serviceStatus, mp, x);
34     }
35 }
36
37 int main() {
38     string service, invalidService;
39     getline(cin, service);
40     getline(cin, invalidService);
41
42     // 解析依赖关系
43     vector<pair<string, string>> dependencies = parseDependencies(service);
```

```

44
45     vector<string> invalidServiceArr;
46     stringstream ss(invalidService);
47     string tmp;
48     while (getline(ss, tmp, ',')) {
49         invalidServiceArr.push_back(tmp);
50     }
51
52     // 依赖关系表
53     unordered_map<string, unordered_set<string>> mp;
54     // 记录服务出现的顺序
55     unordered_map<string, int> serviceSequence;
56     // 记录服务状态
57     unordered_map<string, bool> serviceStatus;
58
59     for (int i = 0; i < dependencies.size(); ++i) {
60         const string &serviceA = dependencies[i].first;
61         const string &serviceB = dependencies[i].second;
62
63         if (serviceSequence.find(serviceA) == serviceSequence.end()) {
64             serviceSequence[serviceA] = i * 2;
65         }
66         if (serviceSequence.find(serviceB) == serviceSequence.end()) {
67             serviceSequence[serviceB] = i * 2 + 1;
68         }
69         // 反向存储依赖关系 (serviceB 故障会影响 serviceA)
70         mp[serviceB].insert(serviceA);
71         serviceStatus[serviceA] = true;
72         serviceStatus[serviceB] = true;
73     }
74
75     // 递归处理所有故障服务
76     for (const auto &invalid : invalidServiceArr) {
77         dfs(serviceStatus, mp, invalid);
78     }
79
80     // 过滤正常服务
81     vector<string> res;
82     for (const auto &p : serviceStatus) {
83         if (p.second) {
84             res.push_back(p.first);
85         }
86     }
87
88     if (res.empty()) {
89         cout << ","; // 无正常服务
90         return 0;
91     }

```

```
92
93 // 按 `serviceSequence` 记录的顺序排序
94 sort(res.begin(), res.end(), [&](const string &a, const string &b) {
95     return serviceSequence[a] < serviceSequence[b];
96 });
97
98 // 输出
99 cout << res[0];
100 for (size_t i = 1; i < res.size(); ++i) {
101     cout << "," << res[i];
102 }
103 return 0;
104 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 通用 split 函数
5      public static List<Map.Entry<String, String>> parseDependencies(String str) {
6          List<Map.Entry<String, String>> result = new ArrayList<>();
7          int start = 0, end;
8          while ((end = str.indexOf(',', start)) != -1) {
9              String dep = str.substring(start, end);
10             int mid = dep.indexOf('-');
11             result.add(new AbstractMap.SimpleEntry<>(dep.substring(0, mid), dep.substring(mid + 1)));
12             start = end + 1;
13         }
14         // 添加最后一个
15         String dep = str.substring(start);
16         int mid = dep.indexOf('-');
17         result.add(new AbstractMap.SimpleEntry<>(dep.substring(0, mid), dep.substring(mid + 1)));
18         return result;
19     }
20
21     // 递归传递失效状态
22     public static void dfs(Map<String, Boolean> serviceStatus, Map<String, Set<String>> mp, String invalidService) {
23         if (!serviceStatus.get(invalidService)) return;
24         serviceStatus.put(invalidService, false);
25         if (mp.get(invalidService) == null) {
26             return;
27         }
28         for (String x : mp.get(invalidService)) {
29             dfs(serviceStatus, mp, x);
30         }
31     }
32
33     public static void main(String[] args) {
34         Scanner sc = new Scanner(System.in);
35         String service = sc.nextLine();
36         String invalidService = sc.nextLine();
37
38         // 解析依赖关系
39         List<Map.Entry<String, String>> dependencies = parseDependencies(service);
40         String[] invalidServiceArr = invalidService.split(",");
```

```

41
42 // 依赖关系表
43 Map<String, Set<String>> mp = new HashMap<>();
44 // 记录服务出现的顺序
45 Map<String, Integer> serviceSequence = new HashMap<>();
46 // 记录服务状态
47 Map<String, Boolean> serviceStatus = new HashMap<>();
48
49 for (int i = 0; i < dependencies.size(); ++i) {
50     String serviceA = dependencies.get(i).getKey();
51     String serviceB = dependencies.get(i).getValue();
52
53     serviceSequence.putIfAbsent(serviceA, i * 2);
54     serviceSequence.putIfAbsent(serviceB, i * 2 + 1);
55
56     mp.computeIfAbsent(serviceB, k -> new HashSet<>()).add(service
57 A);
58     serviceStatus.put(serviceA, true);
59     serviceStatus.put(serviceB, true);
60 }
61
62 // 递归处理所有故障服务
63 for (String invalid : invalidServiceArr) {
64     dfs(serviceStatus, mp, invalid);
65 }
66
67 // 过滤正常服务
68 List<String> res = new ArrayList<>();
69 for (Map.Entry<String, Boolean> p : serviceStatus.entrySet()) {
70     if (p.getValue()) {
71         res.add(p.getKey());
72     }
73 }
74
75 if (res.isEmpty()) {
76     System.out.println(",");
77     return;
78 }
79
80 // 按 `serviceSequence` 记录的顺序排序
81 res.sort(Comparator.comparingInt(serviceSequence::get));
82
83 // 输出
84 System.out.println(String.join(",", res));
85 }

```


Python

```
1 def parse_dependencies(s: str):
2     result = []
3     start = 0
4     while (end := s.find(',', start)) != -1:
5         dep = s[start:end]
6         mid = dep.find('-')
7         result.append((dep[:mid], dep[mid + 1:]))
8         start = end + 1
9     # 添加最后一个
10    dep = s[start:]
11    mid = dep.find('-')
12    result.append((dep[:mid], dep[mid + 1:]))
13    return result
14
15
16 # 递归传递失效状态
17 def dfs(service_status, mp, invalid_service):
18     if not service_status[invalid_service]:
19         return
20     service_status[invalid_service] = False
21     if invalid_service not in mp:
22         return
23     for x in mp[invalid_service]:
24         dfs(service_status, mp, x)
25
26
27 if __name__ == "__main__":
28     service = input()
29     invalid_service = input()
30
31     # 解析依赖关系
32     dependencies = parse_dependencies(service)
33     invalid_service_arr = invalid_service.split(',')
34
35     # 依赖关系表
36     mp = {}
37     # 记录服务出现的顺序
38     service_sequence = {}
39     # 记录服务状态
40     service_status = {}
41
42     for i, (service_a, service_b) in enumerate(dependencies):
43         if service_a not in service_sequence:
44             service_sequence[service_a] = i * 2
45         if service_b not in service_sequence:
```

```
46         service_sequence[service_b] = i * 2 + 1
47     mp.setdefault(service_b, set()).add(service_a)
48     service_status[service_a] = True
49     service_status[service_b] = True
50
51     # 递归处理所有故障服务
52     for invalid in invalid_service_arr:
53         dfs(service_status, mp, invalid)
54
55     # 过滤正常服务
56     res = [key for key, value in service_status.items() if value]
57
58     if not res:
59         print(",")
60     else:
61         # 按 `service_sequence` 记录的顺序排序
62         res.sort(key=lambda x: service_sequence[x])
63         # 输出
64         print(",".join(res))
```

JavaScript

```
1  const readline = require('readline');
2
3  // 通用 split 函数
4  function parseDependencies(str) {
5      let result = [];
6      let start = 0;
7      let end;
8      while ((end = str.indexOf(',', start)) !== -1) {
9          let dep = str.substring(start, end);
10         let mid = dep.indexOf('-');
11         result.push([dep.substring(0, mid), dep.substring(mid + 1)]);
12         start = end + 1;
13     }
14     // 添加最后一个
15     let dep = str.substring(start);
16     let mid = dep.indexOf('-');
17     result.push([dep.substring(0, mid), dep.substring(mid + 1)]);
18     return result;
19 }
20
21 // 递归传递失效状态
22 function dfs(serviceStatus, mp, invalidService) {
23     if (!serviceStatus[invalidService]) return;
24     serviceStatus[invalidService] = false;
25     if (!mp[invalidService]) {
26         return;
27     }
28     for (let x of mp[invalidService]) {
29         dfs(serviceStatus, mp, x);
30     }
31 }
32
33 // Create a readline interface for input
34 const rl = readline.createInterface({
35     input: process.stdin,
36     output: process.stdout
37 });
38
39 let service, invalidService;
40 rl.question('', (serviceInput) => {
41     service = serviceInput;
42     rl.question('', (invalidServiceInput) => {
43         invalidService = invalidServiceInput;
44     });
45     // 解析依赖关系
```

```

46     const dependencies = parseDependencies(service);
47     const invalidServiceArr = invalidService.split(',');
48
49     // 依赖关系表
50     const mp = {};
51     // 记录服务出现的顺序
52     const serviceSequence = {};
53     // 记录服务状态
54     const serviceStatus = {};
55
56     dependencies.forEach(([serviceA, serviceB], i) => {
57         if (!(serviceA in serviceSequence)) {
58             serviceSequence[serviceA] = i * 2;
59         }
60         if (!(serviceB in serviceSequence)) {
61             serviceSequence[serviceB] = i * 2 + 1;
62         }
63         mp[serviceB] = mp[serviceB] || new Set();
64         mp[serviceB].add(serviceA);
65         serviceStatus[serviceA] = true;
66         serviceStatus[serviceB] = true;
67     });
68
69     // 递归处理所有故障服务
70     invalidServiceArr.forEach(invalid => {
71         dfs(serviceStatus, mp, invalid);
72     });
73
74     // 过滤正常服务
75     let res = Object.keys(serviceStatus).filter(key => serviceStatus[k
76 ey]);
77
78     if (res.length === 0) {
79         console.log(',');
80     } else {
81         // 按 `serviceSequence` 记录的顺序排序
82         res.sort((a, b) => serviceSequence[a] - serviceSequence[b]);
83         // 输出
84         console.log(res.join(','));
85     }
86     rl.close();
87 });
88 });

```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "strings"
6     "sort"
7 )
8
9 func parseDependencies(str string) []struct{ A, B string } {
10     var result []struct{ A, B string }
11     start := 0
12     for {
13         end := strings.Index(str[start:], ",")
14         if end == -1 {
15             break
16         }
17         dep := str[start : start+end]
18         mid := strings.Index(dep, "-")
19         result = append(result, struct{ A, B string }{dep[:mid], dep[mid+
20 1:]})
21         start = start + end + 1
22     }
23     // 添加最后一个
24     dep := str[start:]
25     mid := strings.Index(dep, "-")
26     result = append(result, struct{ A, B string }{dep[:mid], dep[mid+1:]})
27     return result
28 }
29
30 func dfs(serviceStatus map[string]bool, mp map[string]map[string]bool, in
31 validService string) {
32     if !serviceStatus[invalidService] {
33         return
34     }
35     serviceStatus[invalidService] = false
36
37     for x := range mp[invalidService] {
38         dfs(serviceStatus, mp, x)
39     }
40 }
41
42 func main() {
43     var service, invalidService string
44     fmt.Scanln(&service)
45     fmt.Scanln(&invalidService)
```

```

44
45 // 解析依赖关系
46 dependencies := parseDependencies(service)
47 invalidServiceArr := strings.Split(invalidService, ",")
48
49 // 依赖关系表
50 mp := make(map[string]map[string]bool)
51 // 记录服务出现的顺序
52 serviceSequence := make(map[string]int)
53 // 记录服务状态
54 serviceStatus := make(map[string]bool)
55
56 for i, dep := range dependencies {
57     serviceA := dep.A
58     serviceB := dep.B
59
60     if _, found := serviceSequence[serviceA]; !found {
61         serviceSequence[serviceA] = i * 2
62     }
63     if _, found := serviceSequence[serviceB]; !found {
64         serviceSequence[serviceB] = i*2 + 1
65     }
66
67     if _, found := mp[serviceB]; !found {
68         mp[serviceB] = make(map[string]bool)
69     }
70     mp[serviceB][serviceA] = true
71     serviceStatus[serviceA] = true
72     serviceStatus[serviceB] = true
73 }
74
75 // 递归处理所有故障服务
76 for _, invalid := range invalidServiceArr {
77     dfs(serviceStatus, mp, invalid)
78 }
79
80 // 过滤正常服务
81 var res []string
82 for key, value := range serviceStatus {
83     if value {
84         res = append(res, key)
85     }
86 }
87
88 if len(res) == 0 {
89     fmt.Println(",")
90     return
91 }

```

```
92
93 // 按 `serviceSequence` 记录的顺序排序
94 sort.Slice(res, func(i, j int) bool {
95     return serviceSequence[res[i]] < serviceSequence[res[j]]
96 })
97
98 // 输出
99 fmt.Println(strings.Join(res, ","))
100 }
```

来自: [华为OD机试 2025 B卷 – 服务失效判断 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)