

# od0613

---

[华为OD 机试 2025 B卷 - 文本统计分析 \(C++ & Python & JAVA & JS\)](#)

[华为OD机考 2025 B卷 - 两个字符串间的最短路径\(C++ & Python & JAVA &](#)

[华为OD机试 2025 B卷 - 机器人走迷宫 \(C++ & Python & JAVA & JS\)](#)

[华为OD机试 2025 B卷 - 竖直四子棋 \(C++ & Python & JAVA & JS &](#)

[华为OD机试2025B卷 - 特殊的加密算法\(C++ & Python & JAVA & JS &](#)

[华为OD机试 2025 B卷 - 路口最短时间问题 \(C++ & Python & JAVA & J](#)

[华为OD机试 2025 B卷 - 最多几个直角三角形 \(C++ & Python & JAVA &](#)

[华为OD机试 2025 B卷 - 表达式括号匹配 \(C++ & Python & JAVA & JS\)](#)

[华为OD机试 2025 B卷 - 最大括号深度 \(C++ & Python & JAVA & JS\)](#)

# 华为OD 机试 2025 B卷 - 文本统计分析 (C++ & Python & JAVA & JS)

## 文本统计分析

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

有一个文件，包含以一定规则写作的文本，请统计文件中包含的文本数量。

规则如下：

- 1. 文本以”;"分隔，最后一条可以没有”;"，但空文本不能算语句，比如”COMMAND A; ;”只能算一条语句。注意，无字符/空白字符/制表符都算作”空”文本；
- 2. 文本可以跨行，比如下面，是一条文本，而不是三条；

- 1. 文本支持字符串，字符串为成对的单引号(')或者成对的双引号(")，字符串可能出现用转义字符()处理的单双引号("your input is")和转义字符本身，比如

▼ Plain Text |

```
1  COMMAND A "Say \"hello\"";
```

- 1. 支持注释，可以出现在字符串之外的任意位置注释以"—“开头，到换行结束，比如：

▼ Plain Text |

```
1  COMMAND A; --this is comment
2  COMMAND --comment
3  A AND COMMAND B;
4  123
```

注意字符串内的"—“，不是注释。

## 输入描述

文本文件

## 输出描述

包含的文本数量

## 用例1

### 输入

▼ Plain Text |

```
1  COMMAND TABLE IF EXISTS "UNITED STATE";
2  COMMAND A GREAT (
3  ID ADSAB,
4  download_length INTE-GER, -- test
5  file_name TEXT,
6  guid TEXT,
7  mime_type TEXT,
8  notifica-tionid INTEGER,
9  original_file_name TEXT,
10 pause_reason_type INTEGER,
11 resumable_flag INTEGER,
12 start_time INTEGER,
13 state INTEGER,
14 folder TEXT,
15 path TEXT,
16 total_length INTE-GER,
17 url TEXT
18 );
```

### 输出

## 题解

思路： 模拟 题, 主要注意几个点

- 正确处理 引号、注释 的开始和结束。以及处理这两种情况下的分号。可以考虑使用两个布尔变量 `insideString`, `insideComment` 来进行处理。
- 是否一段字符串只包含空白字符。

具体逻辑可以参照代码。

**C++**

```
1  #include <iostream>
2  #include <string>
3
4  // 统计字符串中的文本数量
5  int countStatements(const std::string& script) {
6      int statementCount = 0;
7      bool insideString = false;
8      bool insideComment = false;
9      char stringQuote = 0;
10     bool currentStatementEmpty = true;
11
12     for (size_t i = 0; i < script.length(); ++i) {
13         char current = script[i];
14         char next = (i + 1 < script.length()) ? script[i + 1] : '\0';
15
16         // 注释块处理：如果在注释中，直到换行才结束注释
17         if (insideComment) {
18             if (current == '\n') {
19                 insideComment = false;
20             }
21             continue;
22         }
23
24         // 如果遇到 "--", 且不在字符串中，进入注释模式
25         if (!insideString && current == '-' && next == '-') {
26             insideComment = true;
27             i++; // 跳过第二个 '-'
28             continue;
29         }
30
31         // 进入字符串模式（单引号或双引号）
32         if (!insideString && (current == '\'' || current == '\"')) {
33             insideString = true;
34             stringQuote = current;
35             currentStatementEmpty = false;
36             continue;
37         }
38
39         // 字符串结束处理（注意转义引号）
40         if (insideString && current == stringQuote) {
41             // 前面是转移字符
42             if (script[i-1] == stringQuote) {
43                 i++; // 转义引号，跳过
44             } else {
45                 insideString = false;
```

```

46         }
47         continue;
48     }
49
50     // 遇到分号，标记为一条语句结束
51     if (!insideString && current == ';') {
52         if (!currentStatementEmpty) {
53             statementCount++;
54             currentStatementEmpty = true;
55         }
56         continue;
57     }
58
59     // 非空白字符表示当前语句非空
60     if (!insideString && !isspace(current)) {
61         currentStatementEmpty = false;
62     }
63 }
64
65 // 如果末尾没有分号，也算一条语句
66 if (!currentStatementEmpty) {
67     statementCount++;
68 }
69
70 return statementCount;
71 }
72
73 int main() {
74     std::string script, line;
75     while (std::getline(std::cin, line)) {
76         script += line + "\n";
77     }
78     std::cout << countStatements(script) << std::endl;
79     return 0;
80 }

```

## JAVA

```
1  import java.util.Scanner;
2
3  public class Main {
4      // 统计字符串中的语句数量
5      public static int countStatements(String script) {
6          int statementCount = 0;
7          boolean insideString = false;
8          boolean insideComment = false;
9          char stringQuote = 0;
10         boolean currentStatementEmpty = true;
11
12         for (int i = 0; i < script.length(); ++i) {
13             char current = script.charAt(i);
14             char next = (i + 1 < script.length()) ? script.charAt(i + 1)
15             : '\0';
16
17             // 注释块处理：如果在注释中，直到换行才结束注释
18             if (insideComment) {
19                 if (current == '\n') {
20                     insideComment = false;
21                 }
22                 continue;
23             }
24
25             // 遇到 "--", 且不在字符串中，进入注释模式
26             if (!insideString && current == '-' && next == '-') {
27                 insideComment = true;
28                 i++; // 跳过第二个 '-'
29                 continue;
30             }
31
32             // 进入字符串模式（单引号或双引号）
33             if (!insideString && (current == '\'' || current == '\"')) {
34                 insideString = true;
35                 stringQuote = current;
36                 currentStatementEmpty = false;
37                 continue;
38             }
39
40             // 字符串结束处理（注意转义引号）
41             if (insideString && current == stringQuote) {
42                 if (i > 0 && script.charAt(i - 1) == stringQuote) {
43                     i++; // 转义引号，跳过
44                 } else {
45                     insideString = false;
46                 }
47             }
48         }
49         return statementCount;
50     }
51 }
```

```

45         }
46         continue;
47     }
48
49     // 遇到分号，标记为一条语句结束
50     if (!insideString && current == ';') {
51         if (!currentStatementEmpty) {
52             statementCount++;
53             currentStatementEmpty = true;
54         }
55         continue;
56     }
57
58     // 非空白字符表示当前语句非空
59     if (!insideString && !Character.isWhitespace(current)) {
60         currentStatementEmpty = false;
61     }
62 }
63
64 // 如果末尾没有分号，也算一条语句
65 if (!currentStatementEmpty) {
66     statementCount++;
67 }
68
69 return statementCount;
70 }
71
72 public static void main(String[] args) {
73     Scanner scanner = new Scanner(System.in);
74     StringBuilder script = new StringBuilder();
75     while (scanner.hasNextLine()) {
76         script.append(scanner.nextLine()).append("\n");
77     }
78     System.out.println(countStatements(script.toString()));
79 }
80 }

```

## Python



```
1  import sys
2
3
4  def count_statements(script):
5      statement_count = 0
6      inside_string = False
7      inside_comment = False
8      string_quote = ''
9      current_statement_empty = True
10     i = 0
11
12     while i < len(script):
13         current = script[i]
14         next_char = script[i+1] if i + 1 < len(script) else '\0'
15
16
17         if inside_comment:
18             if current == '\n':
19                 inside_comment = False
20                 i += 1
21                 continue
22
23
24         if not inside_string and current == '-' and next_char == '-':
25             inside_comment = True
26             i += 2
27             continue
28
29
30         if not inside_string and current in ['\'', '\"']:
31             inside_string = True
32             string_quote = current
33             current_statement_empty = False
34             i += 1
35             continue
36
37
38         if inside_string and current == string_quote:
39             if i > 0 and script[i-1] == string_quote:
40                 i += 1
41             else:
42                 inside_string = False
43                 i += 1
44                 continue
45
```

```

46
47     if not inside_string and current == ';':
48         if not current_statement_empty:
49             statement_count += 1
50             current_statement_empty = True
51         i += 1
52         continue
53
54
55     if not inside_string and not current.isspace():
56         current_statement_empty = False
57
58     i += 1
59
60
61     if not current_statement_empty:
62         statement_count += 1
63
64     return statement_count
65
66 if __name__ == "__main__":
67     script = sys.stdin.read()
68     print(count_statements(script))

```

## JavaScript

```
1  function countStatements(script) {
2      let statementCount = 0;
3      let insideString = false;
4      let insideComment = false;
5      let stringQuote = '';
6      let currentStatementEmpty = true;
7
8      for (let i = 0; i < script.length; i++) {
9          const current = script[i];
10         const next = (i + 1 < script.length) ? script[i + 1] : '\0';
11
12
13         if (insideComment) {
14             if (current === '\n') {
15                 insideComment = false;
16             }
17             continue;
18         }
19
20
21         if (!insideString && current === '-' && next === '-') {
22             insideComment = true;
23             i++;
24             continue;
25         }
26
27
28         if (!insideString && (current === '"' || current === ''')) {
29             insideString = true;
30             stringQuote = current;
31             currentStatementEmpty = false;
32             continue;
33         }
34
35
36         if (insideString && current === stringQuote) {
37             if (i > 0 && script[i - 1] === stringQuote) {
38                 i++;
39             } else {
40                 insideString = false;
41             }
42             continue;
43         }
44
45     }
```

```

46         if (!insideString && current === ';') {
47             if (!currentStatementEmpty) {
48                 statementCount++;
49                 currentStatementEmpty = true;
50             }
51             continue;
52         }
53
54
55         if (!insideString && !/\s/.test(current)) {
56             currentStatementEmpty = false;
57         }
58     }
59
60
61     if (!currentStatementEmpty) {
62         statementCount++;
63     }
64
65     return statementCount;
66 }
67
68
69 let input = '';
70 process.stdin.setEncoding('utf-8');
71 process.stdin.on('data', chunk => input += chunk);
72 process.stdin.on('end', () => {
73     console.log(countStatements(input));
74 });

```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "io"
7      "os"
8      "strings"
9      "unicode"
10 )
11
12
13 func countStatements(script string) int {
14     statementCount := 0
15     insideString := false
16     insideComment := false
17     var stringQuote rune
18     currentStatementEmpty := true
19     runes := []rune(script)
20
21     for i := 0; i < len(runes); i++ {
22         current := runes[i]
23         var next rune
24         if i+1 < len(runes) {
25             next = runes[i+1]
26         } else {
27             next = 0
28         }
29
30
31         if insideComment {
32             if current == '\n' {
33                 insideComment = false
34             }
35             continue
36         }
37
38
39         if !insideString && current == '-' && next == '-' {
40             insideComment = true
41             i++
42             continue
43         }
44
45     }
```

```

46     if !insideString && (current == '\\' || current == '"') {
47         insideString = true
48         stringQuote = current
49         currentStatementEmpty = false
50         continue
51     }
52
53
54     if insideString && current == stringQuote {
55         if i > 0 && runes[i-1] == stringQuote {
56             i++
57         } else {
58             insideString = false
59         }
60         continue
61     }
62
63
64     if !insideString && current == ';' {
65         if !currentStatementEmpty {
66             statementCount++
67             currentStatementEmpty = true
68         }
69         continue
70     }
71
72
73     if !insideString && !unicode.IsSpace(current) {
74         currentStatementEmpty = false
75     }
76 }
77
78
79 if !currentStatementEmpty {
80     statementCount++
81 }
82
83 return statementCount
84 }
85
86 func main() {
87     reader := bufio.NewReader(os.Stdin)
88     var builder strings.Builder
89     for {
90         line, err := reader.ReadString('\n')
91         builder.WriteString(line)
92         if err == io.EOF {
93             break

```

```
94     }  
95     }  
96     fmt.Println(countStatements(builder.String()))  
97 }
```

来自: [华为OD 机试 2025 B卷 – 文本统计分析 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

# 华为OD机考 2025 B卷 - 两个字符串间的最短路径(C++ & Python & JAVA &

## 两个字符串间的最短路径

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

给定两个字符串，分别为字符串A与字符串B。

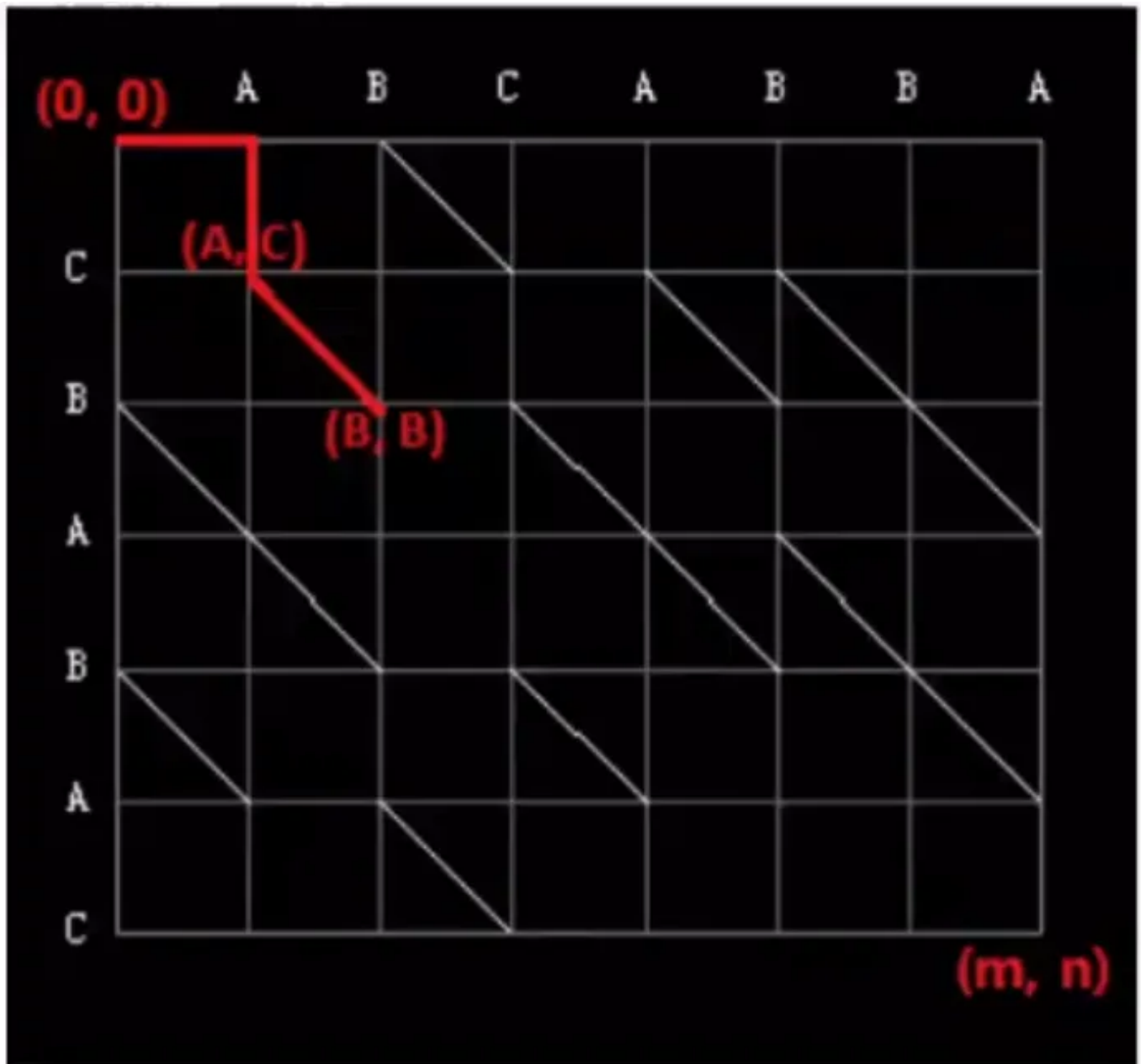
例如A字符串为ABCABBA，B字符串为CBABAC可以得到下图m\*n的[二维数组](#)，定义原点为(0, 0)，终点为(m, n)，水平与垂直的每一条边距离为1，映射成坐标系如下图。

从原点(0, 0)到(0, A)为水平边，距离为1，从(0, A)到(A, C)为垂直边，距离为1；

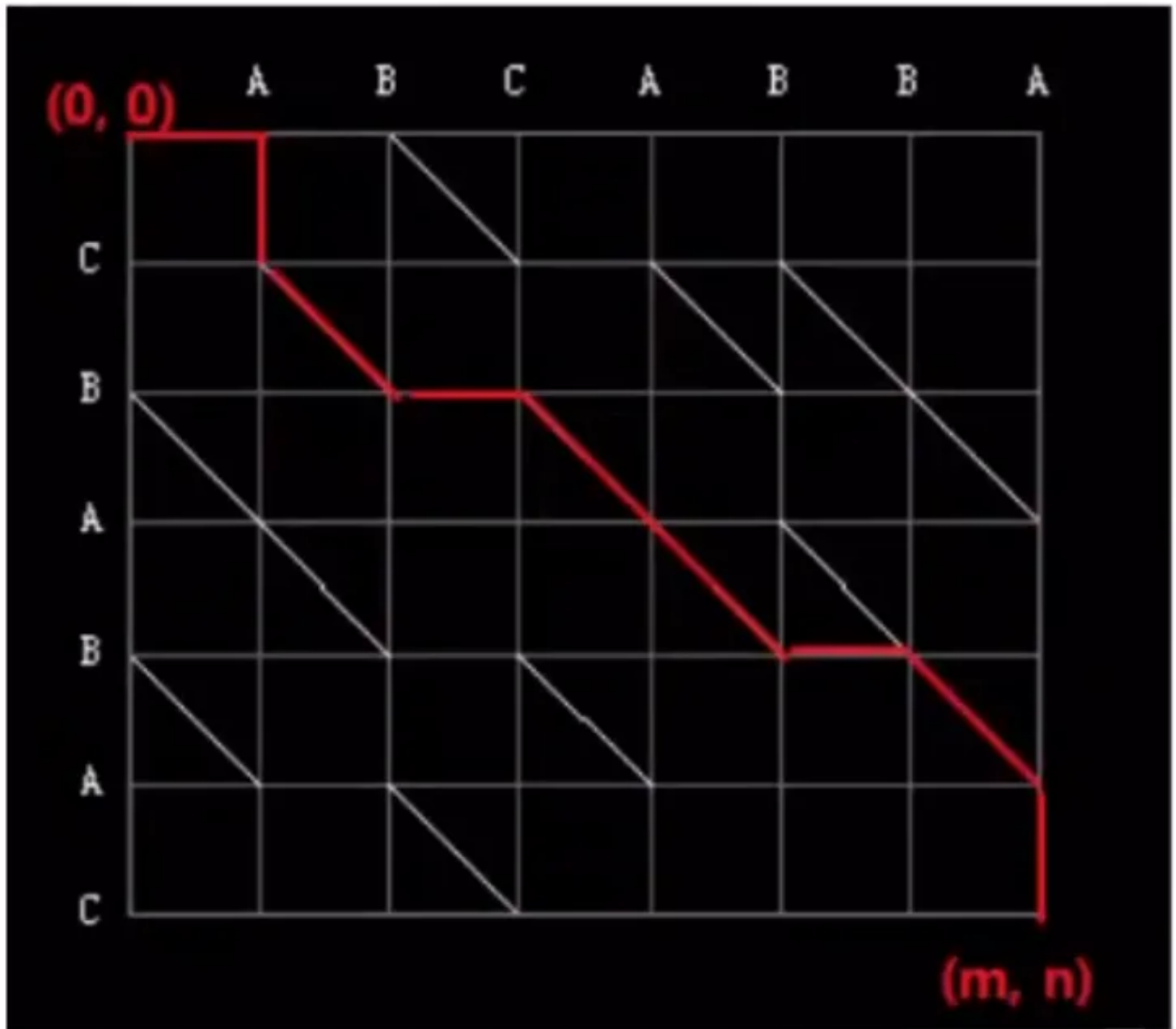
假设两个字符串同一位置的两个字符相同则可以作一个斜边，如(A, C)到(B, B)最短距离为斜边，距离同样为1。

作出所有的斜边如下图，(0, 0)到(B, B)的距离为 1个水平边 + 1个垂直边 + 1个斜边 = 3。





根据定义可知，原点到终点的最短距离路径如下图红线标记，最短距离为：9



## 输入描述

空格分割的两个字符串A与字符串B，字符串不为“空串”，字符格式满足正则规则:[A-Z]，[字符串长度](#) <10000

## 输出描述

原点到终点的最短距离

## 用例1

输入

输出

## 用例2

输入

输出

## 题解

思路：动态规划。

- 定义一个二维数组 `dp[m][n]` ,代表到达 `(m,n)` 位置最短距离。初始设置 `dp[0][0] = 1` .
- 初始化边界条件,设置第一行和第一列的最短距离。
- 使用双层循环遍历A和B的每个字符更新dp数组。
  - 如果 `A[i] != B[j]` 时, `dp[i+1][j+1] = min(dp[i+1][j], dp[i][j+1])+1`
  - 如果 `A[i] == B[j]` 时, `dp[i+1][j+1] = min(dp[i+1][j], dp[i][j+1]), dp[i][j])+1`
- 最后输出 `dp[m][n]` 就是结果。

上述是解决这个题目的基本逻辑。但是本题数据量可以达到 `10000 * 10000 = 1e8` ,很容易爆内存。所以必须想办法压缩内存，可以观察到状态转移方程只涉及到两行，由此定义两个一维数组进行滚动更新，压缩内存。

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 #include<climits>
12 using namespace std;
13
14 int main() {
15     string A,B;
16     cin>>A>>B;
17     int m = A.size(),n = B.size();
18     // 使用两个数组压缩数组大小, 状态转义方程只涉及两行
19     vector<int> dp(m+1,INT_MAX);
20     // 记录上一行的状态
21     vector<int> pre(m+1, INT_MAX);
22
23     for (int i = 0; i <= m; i++) {
24         pre[i] = i;
25     }
26
27     for (int i = 1; i <= n; i++) {
28         // 重新初始化
29         dp.assign(m+1, INT_MAX);
30         // (i,0)到(0,0)的距离
31         dp[0] = i;
32         for (int j = 1; j <= m; j++) {
33             dp[j] = min(pre[j], dp[j-1]) + 1;
34             if (A[j-1] == B[i-1]) {
35                 dp[j] = min(dp[j], pre[j-1] + 1);
36             }
37         }
38         pre = dp;
39     }
40
41     cout << dp[m];
42     return 0;
43 }
```

# JAVA

Plain Text

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          String A = sc.next();
7          String B = sc.next();
8          int m = A.length(), n = B.length();
9
10         // 使用两个数组压缩空间
11         int[] dp = new int[m + 1];
12         int[] pre = new int[m + 1];
13
14         // 初始化 pre 数组
15         for (int i = 0; i <= m; i++) {
16             pre[i] = i;
17         }
18
19         for (int i = 1; i <= n; i++) {
20             // 重新初始化 dp
21             Arrays.fill(dp, Integer.MAX_VALUE);
22             // (i, 0) 到 (0, 0) 的距离
23             dp[0] = i;
24             for (int j = 1; j <= m; j++) {
25                 dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
26                 if (A.charAt(j - 1) == B.charAt(i - 1)) {
27                     dp[j] = Math.min(dp[j], pre[j - 1] + 1);
28                 }
29             }
30             // 注意是深拷贝
31             pre = Arrays.copyOf(dp, dp.length);
32         }
33
34         System.out.println(dp[m]);
35     }
36 }
```

## Python

python3编译器还是会超时，数据量达到了1e8，推荐使用PyPy3进行提交或者切换一种语言。Python解释器对于大数据量处理太慢了。

```
1  import sys
2
3  A, B = sys.stdin.read().split()
4  m, n = len(A), len(B)
5
6
7  pre = [i for i in range(m + 1)]
8  dp = [0] * (m + 1)
9
10 for i in range(1, n + 1):
11
12
13     dp[0] = i
14     for j in range(1, m + 1):
15         dp[j] = min(pre[j], dp[j - 1]) + 1
16         if A[j - 1] == B[i - 1]:
17             dp[j] = min(dp[j], pre[j - 1] + 1)
18
19     for j in range(m+1):
20         pre[j] = dp[j]
21
22 print(dp[m])
```

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let lines = [];
9  rl.on('line', function (line) {
10    lines.push(line);
11    if (lines.length === 1 && lines[0].split(' ').length === 2) {
12      rl.close();
13    }
14  });
15
16  rl.on('close', function () {
17    const [A, B] = lines[0].trim().split(' ');
18    const m = A.length, n = B.length;
19
20
21    let pre = Array.from({ length: m + 1 }, (_, i) => i);
22    let dp = new Array(m + 1).fill(0);
23
24    for (let i = 1; i <= n; i++) {
25      dp = new Array(m + 1).fill(Infinity);
26      dp[0] = i;
27      for (let j = 1; j <= m; j++) {
28        dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
29        if (A[j - 1] === B[i - 1]) {
30          dp[j] = Math.min(dp[j], pre[j - 1] + 1);
31        }
32      }
33
34      pre = [...dp];
35    }
36
37    console.log(dp[m]);
38  });
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func main() {
9     var A, B string
10    fmt.Scan(&A, &B)
11    m, n := len(A), len(B)
12
13
14    pre := make([]int, m+1)
15    for i := 0; i <= m; i++ {
16        pre[i] = i
17    }
18
19    dp := make([]int, m+1)
20
21    for i := 1; i <= n; i++ {
22        for j := 0; j <= m; j++ {
23            dp[j] = math.MaxInt32
24        }
25        dp[0] = i
26        for j := 1; j <= m; j++ {
27            dp[j] = min(pre[j], dp[j-1]) + 1
28            if A[j-1] == B[i-1] {
29                dp[j] = min(dp[j], pre[j-1]+1)
30            }
31        }
32
33        copy(pre, dp)
34    }
35
36    fmt.Println(dp[m])
37 }
38
39 func min(a, b int) int {
40     if a < b {
41         return a
42     }
43     return b
44 }
```



## 两个字符串间的最短路径

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

给定两个字符串，分别为字符串A与字符串B。

例如A字符串为ABCABBA，B字符串为CBABAC可以得到下图m\*n的[二维数组](#)，定义原点为(0, 0)，终点为(m, n)，水平与垂直的每一条边距离为1，映射成坐标系如下图。

从原点(0, 0)到(0, A)为水平边，距离为1，从(0, A)到(A, C)为垂直边，距离为1；

假设两个字符串同一位置的两个字符相同则可以作一个斜边，如(A, C)到(B, B)最短距离为斜边，距离同样为1。

作出所有的斜边如下图，(0, 0)到(B, B)的距离为 1个水平边 + 1个垂直边 + 1个斜边 = 3。



根据定义可知，原点到终点的最短距离路径如下图红线标记，最短距离为：9



## 输入描述

空格分割的两个字符串A与字符串B，字符串不为“空串”，字符格式满足正则规则:[A-Z]，[字符串长度](#)  
<10000

## 输出描述

原点到终点的最短距离

## 用例1

输入

输出

## 用例2

输入

输出

## 题解

思路：动态规划。

- 定义一个二维数组 `dp[m][n]` ,代表到达 `(m,n)` 位置最短距离。初始设置 `dp[0][0] = 1` .
- 初始化边界条件,设置第一行和第一列的最短距离。
- 使用双层循环遍历A和B的每个字符更新dp数组。
  - 如果 `A[i] != B[j]` 时, `dp[i+1][j+1] = min(dp[i+1][j], dp[i][j+1])+1`
  - 如果 `A[i] == B[j]` 时, `dp[i+1][j+1] = min(dp[i+1][j], dp[i][j+1]), dp[i][j])+1`
- 最后输出 `dp[m][n]` 就是结果。

上述是解决这个题目的基本逻辑。但是本题数据量可以达到 `10000 * 10000 = 1e8` ,很容易爆内存。所以必须想办法压缩内存，可以观察到状态转移方程只涉及到两行，由此定义两个一维数组进行滚动更新，压缩内存。

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 #include<climits>
12 using namespace std;
13
14 int main() {
15     string A,B;
16     cin>>A>>B;
17     int m = A.size(),n = B.size();
18     // 使用两个数组压缩数组大小, 状态转义方程只涉及两行
19     vector<int> dp(m+1,INT_MAX);
20     // 记录上一行的状态
21     vector<int> pre(m+1, INT_MAX);
22
23     for (int i = 0; i <= m; i++) {
24         pre[i] = i;
25     }
26
27     for (int i = 1; i <= n; i++) {
28         // 重新初始化
29         dp.assign(m+1, INT_MAX);
30         // (i,0)到(0,0)的距离
31         dp[0] = i;
32         for (int j = 1; j <= m; j++) {
33             dp[j] = min(pre[j], dp[j-1]) + 1;
34             if (A[j-1] == B[i-1]) {
35                 dp[j] = min(dp[j], pre[j-1] + 1);
36             }
37         }
38         pre = dp;
39     }
40
41     cout << dp[m];
42     return 0;
43 }
```

# JAVA

Plain Text

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          String A = sc.next();
7          String B = sc.next();
8          int m = A.length(), n = B.length();
9
10         // 使用两个数组压缩空间
11         int[] dp = new int[m + 1];
12         int[] pre = new int[m + 1];
13
14         // 初始化 pre 数组
15         for (int i = 0; i <= m; i++) {
16             pre[i] = i;
17         }
18
19         for (int i = 1; i <= n; i++) {
20             // 重新初始化 dp
21             Arrays.fill(dp, Integer.MAX_VALUE);
22             // (i, 0) 到 (0, 0) 的距离
23             dp[0] = i;
24             for (int j = 1; j <= m; j++) {
25                 dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
26                 if (A.charAt(j - 1) == B.charAt(i - 1)) {
27                     dp[j] = Math.min(dp[j], pre[j - 1] + 1);
28                 }
29             }
30             // 注意是深拷贝
31             pre = Arrays.copyOf(dp, dp.length);
32         }
33
34         System.out.println(dp[m]);
35     }
36 }
```

# Python

python3编译器还是会超时，数据量达到了1e8，推荐使用PyPy3进行提交或者切换一种语言。Python解释器对于大数据量处理太慢了。

```
1  import sys
2
3  A, B = sys.stdin.read().split()
4  m, n = len(A), len(B)
5
6
7  pre = [i for i in range(m + 1)]
8  dp = [0] * (m + 1)
9
10 for i in range(1, n + 1):
11
12
13     dp[0] = i
14     for j in range(1, m + 1):
15         dp[j] = min(pre[j], dp[j - 1]) + 1
16         if A[j - 1] == B[i - 1]:
17             dp[j] = min(dp[j], pre[j - 1] + 1)
18
19     for j in range(m+1):
20         pre[j] = dp[j]
21
22 print(dp[m])
```

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let lines = [];
9  rl.on('line', function (line) {
10    lines.push(line);
11    if (lines.length === 1 && lines[0].split(' ').length === 2) {
12      rl.close();
13    }
14  });
15
16  rl.on('close', function () {
17    const [A, B] = lines[0].trim().split(' ');
18    const m = A.length, n = B.length;
19
20
21    let pre = Array.from({ length: m + 1 }, (_, i) => i);
22    let dp = new Array(m + 1).fill(0);
23
24    for (let i = 1; i <= n; i++) {
25      dp = new Array(m + 1).fill(Infinity);
26      dp[0] = i;
27      for (let j = 1; j <= m; j++) {
28        dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
29        if (A[j - 1] === B[i - 1]) {
30          dp[j] = Math.min(dp[j], pre[j - 1] + 1);
31        }
32      }
33
34      pre = [...dp];
35    }
36
37    console.log(dp[m]);
38  });
```

Go



```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func main() {
9     var A, B string
10    fmt.Scan(&A, &B)
11    m, n := len(A), len(B)
12
13
14    pre := make([]int, m+1)
15    for i := 0; i <= m; i++ {
16        pre[i] = i
17    }
18
19    dp := make([]int, m+1)
20
21    for i := 1; i <= n; i++ {
22        for j := 0; j <= m; j++ {
23            dp[j] = math.MaxInt32
24        }
25        dp[0] = i
26        for j := 1; j <= m; j++ {
27            dp[j] = min(pre[j], dp[j-1]) + 1
28            if A[j-1] == B[i-1] {
29                dp[j] = min(dp[j], pre[j-1]+1)
30            }
31        }
32
33        copy(pre, dp)
34    }
35
36    fmt.Println(dp[m])
37 }
38
39 func min(a, b int) int {
40     if a < b {
41         return a
42     }
43     return b
44 }
```

来自: [华为OD机考 2025 B卷 – 两个字符串间的最短路径\(C++ & Python & JAVA & JS & GO\)\\_华为2025 机考 a卷-CSDN博客](#)

# 华为OD机试 2025 B卷 - 机器人走迷宫 (C++ & Python & JAVA & JS)

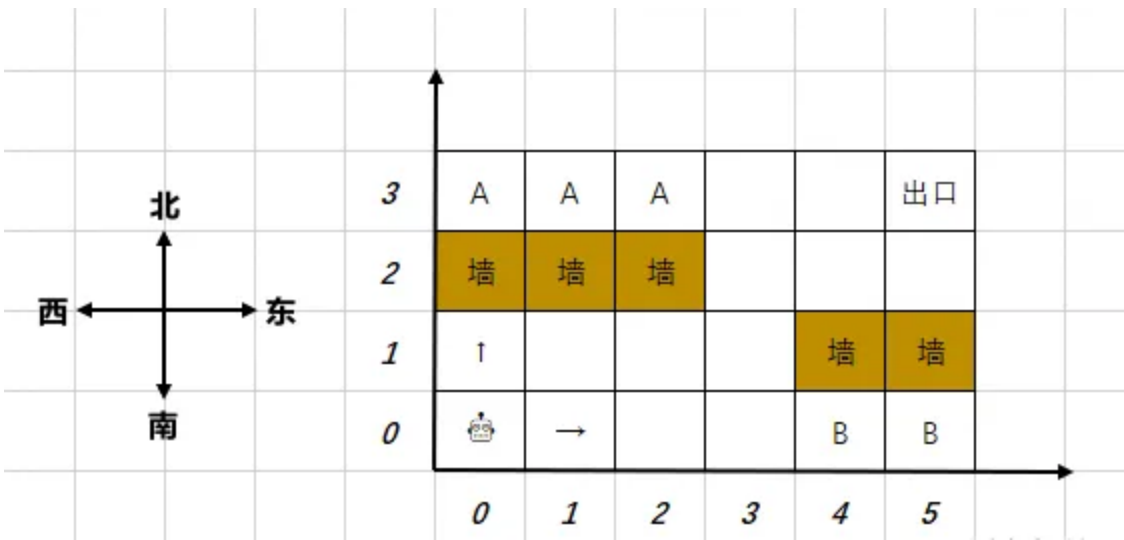
## 机器人走迷宫

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

- 1. 房间由XY的方格组成，例如下图为6\*4的大小。每一个方格以坐标(x, y)描述。
- 2. 机器人固定从方格(0, 0)出发，只能向东或者向北前进。出口固定为房间的最东北角，如下图的方格(5, 3)。用例保证机器人可以从入口走到出口。
- 3. 房间有些方格是墙壁，如(4, 1)，机器人不能经过那儿。
- 4. 有些地方是一旦到达就无法走到出口的，如标记为B的方格，称之为陷阱方格。
- 5. 有些地方是机器人无法到达的，如标记为A的方格，称之为不可达方格，不可达方格不包括墙壁所在的位置。
- 6. 如下示例图中，陷阱方格有2个，不可达方格有3个。
- 7. 请为该机器人实现路径规划功能：给定房间大小、墙壁位置，请计算出陷阱方格与不可达方格分别有多少个。



### 输入描述

- 第一行为房间的X和Y (0 < X,Y <= 1000)
- 第二行为房间中墙壁的个数N (0 <= N < X\*Y)
- 接着下面会有N行墙壁的坐标

同一行中如果有多个数据以一个空格隔开，用例保证所有的输入数据均合法。（结尾不带回车换行）

## 输出描述

陷阱方格与不可达方格数量，两个信息在一行中输出，以一个空格隔开。（结尾不带回车换行）

## 用例1

### 输入

▼ Plain Text |

```
1 6 4
2 5
3 0 2
4 1 2
5 2 2
6 4 1
7 5 1
```

### 输出

## 说明

该输入对应上图示例中的迷宫，陷阱方格有2个，不可达方格有3个

## 用例2

### 输入

## 输出

## 说明

没有陷阱方格，不可达方格有4个，分别是(4,0) (4,1) (5,0) (5,1)

## 题解

思路：DFS 实现

1. 陷阱方格 为机器人可访问，但是访问之后不能达到终点的位置。不可达方格 为机器人不能访问的位置。
2. 接收题目输入，初始将地图所有位置的值设置为0，墙的位置值设置为-1。将终点位置值设置为2(前提是终点位置不为-1)。接下来定义两个规则：
  - a. 经过次方格能够到达终点 的位置值设置为2
  - b. 经过此方格不能到达终点 的位置值设置为1.
3. 了解上面的规则之后，从起点进行DFS递归(向北和向东)两个方向访问，递归过程中如果能够到达终点(为2的位置)，将递归路径上的位置全部设置为2。如果不能到达终点，将递归路径上的所有位置值设置为1.
4. 经过3处理之后，陷阱方格 数量就是地图中值为1的数量。不可达 数量就是地图中值为0的数量。统计一下即可。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  using namespace std;
8  int x,y;
9  int n;
10
11 bool DFS(int currentX, int currentY, vector<vector<int>>& grid) {
12     if (currentX < 0 || currentX >= x) {
13         return false;
14     }
15     if (currentY < 0 || currentY >= y) {
16         return false;
17     }
18     // 墙
19     if (grid[currentX][currentY] == -1) {
20         return false;
21     }
22     // 到达之前标记不可达位置
23     if (grid[currentX][currentY] == 1) {
24         return false;
25     }
26     // 可达终点
27     if (grid[currentX][currentY] == 2) {
28         return true;
29     }
30     bool eastFlag = DFS(currentX + 1, currentY, grid);
31     bool northFlag = DFS(currentX, currentY + 1, grid);
32     // 可达标记为2
33     if (eastFlag || northFlag) {
34         grid[currentX][currentY] = 2;
35         // 说明可以经过此格子但无法到达终点 标记为1 陷阱方格
36     } else {
37         grid[currentX][currentY] = 1;
38     }
39     return eastFlag || northFlag;
40 }
41
42 int main() {
43     // 接收输入
44     cin >> x >> y;
45 }
```

```

46 // 初始化地图
47 vector<vector<int>> grid(x, vector<int>(y, 0));
48 // 处理墙
49 cin >> n;
50 for (int i = 0; i < n; i++) {
51     int x1, y1;
52     cin >> x1 >> y1;
53     // 将墙标志位-1
54     grid[x1][y1] = -1;
55 }
56
57 // 将终点赋值为2, 为2的位置代表可达终点
58 if (grid[x-1][y-1] != -1) {
59     grid[x-1][y-1] = 2;
60 }
61
62 // 深度遍历 获取哪些位置可达 哪些不可达
63
64 DFS(0, 0, grid);
65
66 // 统计陷阱个数和不可达数量
67 int trapCount, untouchCount;
68 trapCount = untouchCount = 0;
69 for (int i = 0; i < x; i++) {
70     for (int j = 0; j < y; j++) {
71         if (grid[i][j] == 0) {
72             untouchCount++;
73         }
74         if (grid[i][j] == 1) {
75             trapCount++;
76         }
77     }
78 }
79
80 cout << trapCount << " " << untouchCount;
81 return 0;
82 }

```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4      static int x, y;
5      static int[][] grid;
6
7      public static boolean DFS(int currentX, int currentY) {
8          if (currentX < 0 || currentX >= x || currentY < 0 || currentY >=
9              y)
10             return false;
11
12             // 墙
13             if (grid[currentX][currentY] == -1)
14                 return false;
15             // 到达之前标记不可达位置
16             if (grid[currentX][currentY] == 1)
17                 return false;
18             // 可达终点
19             if (grid[currentX][currentY] == 2)
20                 return true;
21
22             boolean eastFlag = DFS(currentX + 1, currentY);
23             boolean northFlag = DFS(currentX, currentY + 1);
24
25             // 可达标记为2
26             if (eastFlag || northFlag) {
27                 grid[currentX][currentY] = 2;
28             } else {
29                 // 说明可以经过此格子但无法到达终点, 标记为1
30                 grid[currentX][currentY] = 1;
31             }
32
33             return eastFlag || northFlag;
34         }
35
36         public static void main(String[] args) {
37             Scanner sc = new Scanner(System.in);
38             x = sc.nextInt();
39             y = sc.nextInt();
40             grid = new int[x][y];
41
42             int n = sc.nextInt();
43             for (int i = 0; i < n; i++) {
44                 int x1 = sc.nextInt();
45                 int y1 = sc.nextInt();
```



```

45         grid[x1][y1] = -1; // 墙
46     }
47     if (grid[x - 1][y - 1] != -1) {
48         grid[x - 1][y - 1] = 2; // 终点标记为2
49     }
50
51
52     DFS(0, 0); // DFS遍历
53
54     int trapCount = 0, untouchCount = 0;
55     for (int i = 0; i < x; i++) {
56         for (int j = 0; j < y; j++) {
57             if (grid[i][j] == 0)
58                 untouchCount++;
59             if (grid[i][j] == 1)
60                 trapCount++;
61         }
62     }
63
64     System.out.println(trapCount + " " + untouchCount);
65 }
66 }

```

## Python

```
1  import sys
2
3  sys.setrecursionlimit(2500)
4
5  def DFS(currentX, currentY):
6      if currentX < 0 or currentX >= x or currentY < 0 or currentY >= y:
7          return False
8      if grid[currentX][currentY] == -1:
9          return False
10     if grid[currentX][currentY] == 1:
11         return False
12     if grid[currentX][currentY] == 2:
13         return True
14
15     eastFlag = DFS(currentX + 1, currentY)
16     northFlag = DFS(currentX, currentY + 1)
17
18     if eastFlag or northFlag:
19         grid[currentX][currentY] = 2
20     else:
21         grid[currentX][currentY] = 1
22     return eastFlag or northFlag
23
24 if __name__ == "__main__":
25     x, y = map(int, input().split())
26     grid = [[0 for _ in range(y)] for _ in range(x)]
27
28     n = int(input())
29     for _ in range(n):
30         x1, y1 = map(int, input().split())
31         grid[x1][y1] = -1
32
33     if grid[x - 1][y - 1] != -1:
34
35         grid[x - 1][y - 1] = 2
36
37     DFS(0, 0)
38
39     trapCount = untouchCount = 0
40     for i in range(x):
41         for j in range(y):
42             if grid[i][j] == 0:
43                 untouchCount += 1
44             if grid[i][j] == 1:
45                 trapCount += 1
```

46  
47

```
print(trapCount, untouchCount)
```

## 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', function (line) {
10    inputLines.push(line);
11  }).on('close', function () {
12    let [x, y] = inputLines[0].split(' ').map(Number);
13    let n = parseInt(inputLines[1]);
14    let grid = Array.from({ length: x }, () => Array(y).fill(0));
15
16    for (let i = 0; i < n; i++) {
17      let [x1, y1] = inputLines[2 + i].split(' ').map(Number);
18      grid[x1][y1] = -1;
19    }
20    if (grid[x - 1][y - 1] !== -1) {
21      grid[x - 1][y - 1] = 2;
22    }
23
24
25    function DFS(currentX, currentY) {
26      if (currentX < 0 || currentX >= x || currentY < 0 || currentY >=
y) return false;
27      if (grid[currentX][currentY] === -1) return false;
28      if (grid[currentX][currentY] === 1) return false;
29      if (grid[currentX][currentY] === 2) return true;
30
31      let eastFlag = DFS(currentX + 1, currentY);
32      let northFlag = DFS(currentX, currentY + 1);
33
34      if (eastFlag || northFlag) {
35        grid[currentX][currentY] = 2;
36
37      } else {
38        grid[currentX][currentY] = 1;
39      }
40
41      return eastFlag || northFlag;
42    }
43
44    DFS(0, 0);
```

```
45
46     let trapCount = 0, untouchCount = 0;
47     for (let i = 0; i < x; i++) {
48         for (let j = 0; j < y; j++) {
49             if (grid[i][j] === 0) untouchCount++;
50             if (grid[i][j] === 1) trapCount++;
51         }
52     }
53
54     console.log(`${trapCount} ${untouchCount}`);
55 });
```

**Go**

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 var (
8     x, y int
9     grid [][]int
10 )
11
12 func DFS(currentX, currentY int) bool {
13     if currentX < 0 || currentX >= x || currentY < 0 || currentY >= y {
14         return false
15     }
16     if grid[currentX][currentY] == -1 {
17         return false
18     }
19     if grid[currentX][currentY] == 1 {
20         return false
21     }
22     if grid[currentX][currentY] == 2 {
23         return true
24     }
25
26     eastFlag := DFS(currentX+1, currentY)
27     northFlag := DFS(currentX, currentY + 1)
28
29
30     if eastFlag || northFlag {
31         grid[currentX][currentY] = 2
32     } else {
33         grid[currentX][currentY] = 1
34     }
35     return eastFlag || northFlag
36 }
37
38
39 func main() {
40     var n int
41     fmt.Scan(&x, &y)
42
43
44     grid = make([][]int, x)
45     for i := range grid {
```

```

46     grid[i] = make([]int, y)
47 }
48
49 fmt.Scan(&n)
50 for i := 0; i < n; i++ {
51     var x1, y1 int
52     fmt.Scan(&x1, &y1)
53     grid[x1][y1] = -1
54 }
55     if grid[x-1][y-1] != -1 {
56         grid[x-1][y-1] = 2
57     }
58
59     DFS(0, 0)
60
61     trapCount, untouchCount := 0, 0
62     for i := 0; i < x; i++ {
63         for j := 0; j < y; j++ {
64             if grid[i][j] == 0 {
65                 untouchCount++
66             }
67             if grid[i][j] == 1 {
68                 trapCount++
69             }
70         }
71     }
72
73     fmt.Printf("%d %d\n", trapCount, untouchCount)
74 }

```

来自: [华为OD机试 2025 B卷 – 机器人走迷宫 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

# 华为OD机试 2025 B卷 - 竖直四子棋 (C++ & Python & JAVA & JS &

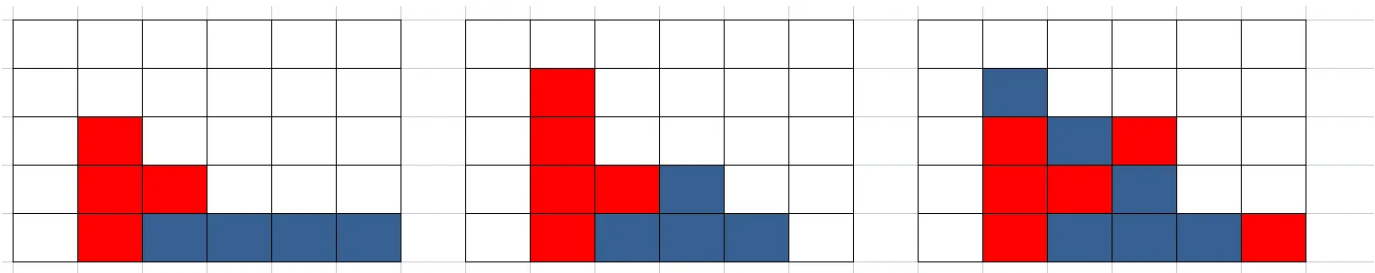
## 竖直四子棋

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](#)

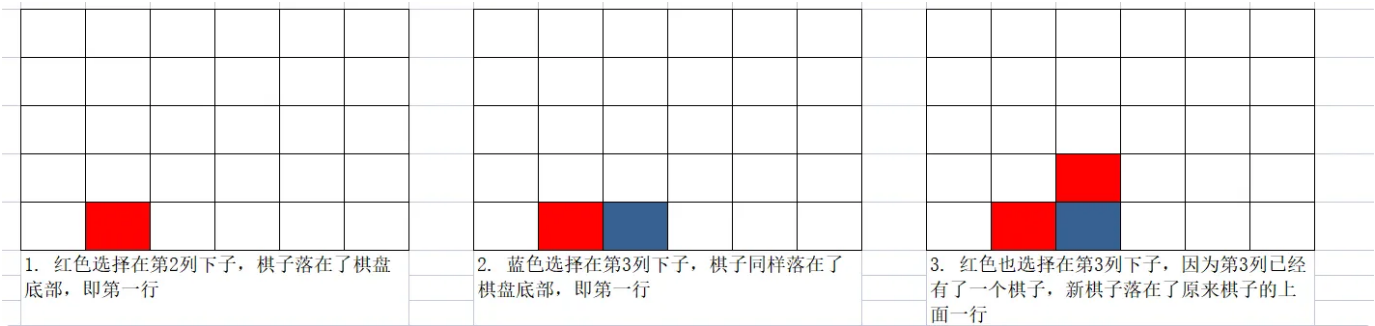
华为OD机试2025B卷 200分题型

### 题目描述

竖直四子棋的棋盘是竖立起来的，双方轮流选择棋盘的一列下子，棋子因重力落到棋盘底部或者其他棋子之上，当一列的棋子放满时，无法再在这列上下子。一方的4个棋子横、竖或者斜方向连成一线时获胜。现给定一个棋盘和红蓝对弈双方的下子步骤，判断红方或蓝方是否在某一步获胜。下面以一个6×5的棋盘图示说明落子过程：



下面给出横、竖和斜方向四子连线的图示：



### 输入描述

输入为2行，第一行指定棋盘的宽和高，为空格分隔的两个数字；  
第二行依次间隔指定红蓝双方的落子步骤，第1步为红方的落子，第2步为蓝方的落子，第3步为红方的落子，以此类推。  
步骤由空格分隔的一组数字表示，每个数字为落子的列的编号（最左边的列编号为1，往右递增）。用例保证数字均为32位有符号数。



## 输出描述

如果落子过程中红方获胜，输出 N,red ；

如果落子过程中蓝方获胜，输出 N,blue ；

如果出现非法的落子步骤，输出 N,error。

N为落子步骤的序号，从1开始。如果双方都没有获胜，输出 0,draw 。

非法落子步骤有两种，一是列的编号超过棋盘范围，二是在一个已经落满子的列上落子。

N和单词red、blue、draw、error之间是英文逗号连接。

## 用例1

### 输入

### 输出

### 说明

在第7步，红方在第4列落下一子后，红方的四个子在第一行连成一线，故红方获胜，输出 7,red。

## 用例2

### 输入

### 输出

### 说明

第1步的列序号为0，超出有效列编号的范围，故输出 1,error。

## 题解

思路： 逻辑分析 + DFS 题

1. 初始化棋盘，初始将所有位置设置为 0 。
2. 根据每一次下棋的所选择列，计算出下落之后所处的行，确定下棋位置。
  - 首先判断位置的合法性(超过棋盘，列和行)。不合法直接输出 序号,error
  - 然后将对应位置设置值规则为 红方下设置为1， 蓝方下设置为2 。
  - 然后判断下完这步棋之后是否能形成 相同棋子同一方向大于等于四子相连情况 。

3. 根据2的逻辑处理完所有下棋逻辑之后，如果不存在 大于等于四子相同棋子相连情况 输出 `0,draw`

**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  using namespace std;
10
11 // 通用 split 函数
12 vector<int> split(const string& str, const string& delimiter) {
13     vector<int> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(stoi(str.substr(start, end - start)));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(stoi(str.substr(start)));
23     return result;
24 }
25
26 // 判断是否存在相同大于等于4连续相连
27 bool judge(int x , int y, int play, vector<vector<int>>& grid, int n, int m) {
28     // 上 左 对角线 反对角线
29     int direct[4][2] = {
30     {-1, 0}, {0, -1}, {-1, -1}, {-1, 1}};
31
32     for (int i = 0; i < 4; i++) {
33         // 指定方向连续相连子
34         int count = 1;
35         int currentX = x;
36         int currentY = y;
37         // 正方向
38         while (true) {
39             currentX += direct[i][0];
40             currentY += direct[i][1];
41             if (currentX >= 1 && currentX <= n && currentY >= 1 && currentY <= m && grid[currentX][currentY] == play) {
42                 count++;
43             } else {
```

```

44         break;
45     }
46 }
47 if (count >= 4) {
48     return true;
49 }
50
51     currentX = x;
52     currentY = y;
53     // 反向
54     while (true) {
55         currentX -= direct[i][0];
56         currentY -= direct[i][1];
57         if (currentX >= 1 && currentX <= n && currentY >= 1 && currentY <= m && grid[currentX][currentY] == play) {
58             count++;
59         } else {
60             break;
61         }
62     }
63     if (count >= 4) {
64         return true;
65     }
66 }
67 return false;
68 }
69
70
71 int main() {
72     int n,m;
73     // 宽 高
74     cin >> m >> n;
75     cin.ignore();
76     string input;
77     getline(cin , input);
78
79     vector<vector<int>> grid(n + 1, vector<int>(m + 1, 0));
80     vector<int> move = split(input, " ");
81
82     int k = move.size();
83     for (int i = 0; i < k; i++) {
84         int x,y;
85         y = move[i];
86         // 合法性判断
87         if (y <= 0 || y > m) {
88             cout << i+1 << ",error";
89             return 0;
90         }

```

```

91         int play = (i % 2 == 0) ? 1 : 2;
92
93         // 计算最终会落到的行
94         x = n + 1;
95         while (x > 1 && grid[x - 1][y] == 0) {
96             x--;
97         }
98         // 合法性判断 列全部满了
99         if (x == n + 1) {
100             cout << i+1 << ",error";
101             return 0;
102         }
103
104         grid[x][y] = play;
105         // 判断是否形成四子相连的情况
106         if (i >= 6 && judge(x, y, play, grid, n, m)) {
107             cout << i+1 << ", "<< ((play == 1) ? "red" : "blue");
108             return 0;
109         }
110     }
111 }
112
113 cout << "0,draw";
114 return 0;
115 }

```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 判断是否有连续 >=4 的相同棋子 (上下、左右、两对角)
5      static boolean judge(int x, int y, int play, int[][] grid, int n, int
6      m) {
7          int[][] directions = {
8              {-1, 0}, {0, -1}, {-1, -1}, {-1, 1}}; // 上、左、对角线、反对角线
9
10         for (int[] dir : directions) {
11             int count = 1;
12             int cx = x, cy = y;
13
14             // 正方向
15             while (true) {
16                 cx += dir[0];
17                 cy += dir[1];
18                 if (cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][c
19 y] == play) {
20                     count++;
21                 } else {
22                     break;
23                 }
24             }
25             if (count >= 4) return true;
26             // 反方向
27             cx = x; cy = y;
28             while (true) {
29                 cx -= dir[0];
30                 cy -= dir[1];
31                 if (cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][c
32 y] == play) {
33                     count++;
34                 } else {
35                     break;
36                 }
37             }
38             if (count >= 4) return true;
39         }
40         return false;
41     }
42
43     public static void main(String[] args) {
44         Scanner sc = new Scanner(System.in);
```

```

43     int m = sc.nextInt(), n = sc.nextInt(); // 宽、高
44     sc.nextLine();
45     String[] moves = sc.nextLine().trim().split(" ");
46
47     int[][] grid = new int[n + 2][m + 2]; // 多加一层防止越界
48
49     for (int i = 0; i < moves.length; i++) {
50         int y = Integer.parseInt(moves[i]);
51
52         // 非法列
53         if (y <= 0 || y > m) {
54             System.out.println((i + 1) + ",error");
55             return;
56         }
57
58         int x = n + 1;
59         while (x > 1 && grid[x - 1][y] == 0) {
60             x--;
61         }
62
63         // 列满了
64         if (x == n + 1) {
65             System.out.println((i + 1) + ",error");
66             return;
67         }
68
69         int play = (i % 2 == 0) ? 1 : 2; // 红蓝交替
70         grid[x][y] = play;
71
72         // 从第 7 步开始判断是否成线
73         if (i >= 6 && judge(x, y, play, grid, n, m)) {
74             System.out.println((i + 1) + "," + (play == 1 ? "red" : "blue"));
75             return;
76         }
77     }
78
79     System.out.println("0,draw"); // 无赢家
80 }
81 }

```

## Python

```
1 def judge(x, y, play, grid, n, m):
2
3     directions = [(-1, 0), (0, -1), (-1, -1), (-1, 1)]
4     for dx, dy in directions:
5         count = 1
6         cx, cy = x, y
7
8
9         while 1 <= cx + dx <= n and 1 <= cy + dy <= m and grid[cx + dx][c
y + dy] == play:
10             cx += dx
11             cy += dy
12             count += 1
13
14             if count >= 4:
15                 return True
16
17             cx, cy = x, y
18             while 1 <= cx - dx <= n and 1 <= cy - dy <= m and grid[cx - dx][c
y - dy] == play:
19                 cx -= dx
20                 cy -= dy
21                 count += 1
22
23                 if count >= 4:
24                     return True
25             return False
26
27 def main():
28     m, n = map(int, input().split())
29     moves = list(map(int, input().split()))
30     grid = [[0] * (m + 2) for _ in range(n + 2)]
31
32     for i, y in enumerate(moves):
33
34         if y <= 0 or y > m:
35             print(f"{i+1},error")
36             return
37
38         x = n + 1
39         while x > 1 and grid[x - 1][y] == 0:
40             x -= 1
41
42
43         if x == n + 1:
```



```
44         print(f"{i+1},error")
45         return
46
47     play = 1 if i % 2 == 0 else 2
48     grid[x][y] = play
49
50
51     if i >= 6 and judge(x, y, play, grid, n, m):
52         print(f"{i+1},{ 'red' if play == 1 else 'blue'}")
53         return
54
55     print("0,draw")
56
57 if __name__ == "__main__":
58     main()
```

## JavaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({ input: process.stdin });
3
4  let lines = [];
5  rl.on('line', line => {
6    lines.push(line.trim());
7    if (lines.length === 2) {
8      solve();
9      rl.close();
10   }
11 });
12
13 function solve() {
14   let [m, n] = lines[0].split(' ').map(Number);
15   let moves = lines[1].split(' ').map(Number);
16   let grid = Array.from({ length: n + 2 }, () => Array(m + 2).fill(0));
17
18   for (let i = 0; i < moves.length; i++) {
19     let y = moves[i];
20
21
22     if (y <= 0 || y > m) {
23       console.log(`${i + 1},error`);
24       return;
25     }
26
27     let x = n + 1;
28     while (x > 1 && grid[x - 1][y] === 0) x--;
29
30
31     if (x === n + 1) {
32       console.log(`${i + 1},error`);
33       return;
34     }
35
36     let play = i % 2 === 0 ? 1 : 2;
37     grid[x][y] = play;
38
39     if (i >= 6 && judge(x, y, play, grid, n, m)) {
40       console.log(`${i + 1},${play === 1 ? 'red' : 'blue'}`);
41       return;
42     }
43   }
44
45   console.log("0,draw");
```

```

46 }
47
48 function judge(x, y, play, grid, n, m) {
49     const directions = [[-1, 0], [0, -1], [-1, -1], [-1, 1]];
50     for (let [dx, dy] of directions) {
51         let count = 1;
52         let cx = x, cy = y;
53
54
55         while (cx + dx >= 1 && cx + dx <= n && cy + dy >= 1 && cy + dy <=
56 m && grid[cx + dx][cy + dy] === play) {
57             cx += dx;
58             cy += dy;
59             count++;
60         }
61         if (count >= 4) return true;
62
63         cx = x; cy = y;
64         while (cx - dx >= 1 && cx - dx <= n && cy - dy >= 1 && cy - dy <=
65 m && grid[cx - dx][cy - dy] === play) {
66             cx -= dx;
67             cy -= dy;
68             count++;
69         }
70         if (count >= 4) return true;
71     }
72     return false;
73 }

```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11
12  func judge(x, y, play int, grid [][]int, n, m int) bool {
13      dirs := [][]int{{-1, 0}, {0, -1}, {-1, -1}, {-1, 1}}
14
15      for _, d := range dirs {
16          count := 1
17          cx, cy := x, y
18
19          for {
20              cx += d[0]
21              cy += d[1]
22              if cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == pla
23  y {
24                  count++
25              } else {
26                  break
27              }
28          }
29          if count >= 4 {
30              return true
31          }
32
33          cx, cy = x, y
34          for {
35              cx -= d[0]
36              cy -= d[1]
37              if cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == pla
38  y {
39                  count++
40              } else {
41                  break
42              }
43          }
```

```

44     if count >= 4 {
45         return true
46     }
47 }
48 return false
49 }
50
51 func main() {
52     scanner := bufio.NewScanner(os.Stdin)
53     scanner.Scan()
54     parts := strings.Fields(scanner.Text())
55     m, _ := strconv.Atoi(parts[0])
56     n, _ := strconv.Atoi(parts[1])
57
58     scanner.Scan()
59     moveStr := strings.Fields(scanner.Text())
60     move := make([]int, len(moveStr))
61     for i, v := range moveStr {
62         move[i], _ = strconv.Atoi(v)
63     }
64
65     grid := make([][]int, n+2)
66     for i := range grid {
67         grid[i] = make([]int, m+2)
68     }
69
70     for i := 0; i < len(move); i++ {
71         y := move[i]
72
73         if y <= 0 || y > m {
74             fmt.Printf("%d,error\n", i+1)
75             return
76         }
77
78         x := n + 1
79         for x > 1 && grid[x-1][y] == 0 {
80             x--
81         }
82
83         if x == n+1 {
84             fmt.Printf("%d,error\n", i+1)
85             return
86         }
87         play := 1
88         if i%2 == 1 {
89             play = 2
90         }
91         grid[x][y] = play

```

```
92
93     if i >= 6 && judge(x, y, play, grid, n, m) {
94         color := "red"
95         if play == 2 {
96             color = "blue"
97         }
98         fmt.Printf("%d,%s\n", i+1, color)
99         return
100     }
101 }
102 fmt.Println("0,draw")
103 }
```

来自: [华为OD机试 2025 B卷 – 竖直四子棋 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

# 华为OD机试2025B卷 - 特殊的加密算法(C++ & Python & JAVA & JS &

## 特殊的加密算法

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

有一种特殊的加密算法，明文为一段数字串，经过密码本查找转换，生成另一段密文数字串。

规则如下：

1. 明文为一段数字串由 0~9 组成
2. 密码本为数字 0~9 组成的二维数组
3. 需要按明文串的数字顺序在密码本里找到同样的数字串，密码本里的数字串是由相邻的单元格数字组成，上下和左右是相邻的，注意：对角线不相邻，同一个单元格的数字不能重复使用。
4. 每一位明文对应密文即为密码本中找到的单元格所在的行和列序号（序号从0开始）组成的两个数字。  
如明文第  $i$  位  $Data[i]$  对应密码本单元格为  $Book[x][y]$ ，则明文第  $i$  位对应的密文为  $X\ Y$ ， $X$ 和 $Y$ 之间用空格隔开。

如果有多条密文，返回字符序最小的密文。

如果密码本无法匹配，返回"error"。

请你设计这个加密程序。

## 输入描述

第一行输入 1 个正整数  $N$ ，代表明文的长度 ( $1 \leq N \leq 200$ )

第二行输入  $N$  个明文组成的序列  $Data[i]$  ( $0 \leq Data[i] \leq 9$ )

第三行输入 1 个正整数  $M$ ，代表密文的长度

接下来  $M$  行，每行  $M$  个数，代表密文矩阵

## 输出描述

输出字典序最小密文，如果无法匹配，输出"error"

## 示例1

### 输入

		Plain Text	
1	2		
2	0 3		
3	3		
4	0 0 2		
5	1 3 4		
6	6 6 4		

输出

## 示例2

输入

		Plain Text	
1	2		
2	0 5		
3	3		
4	0 0 2		
5	1 3 4		
6	6 6 4		

输出

说明

找不到 0 5 的序列，返回error



## 题解

思路：递归回溯

1. 接收明文 `plaintext` 和密文数据 `cipherBook[][]`。
2. 从上到下，从左之后遍密文举证，当找到 `cipherBook[i][j] == plaintext[0]` ,从当前位置使用 递归回溯 算法向四周遍历，尝试获取满足题目要求的坐标字符串。当存在多个满足条件的组合坐标字符串时，取其中字典序最小的结果。
3. 具体逻辑可参照下述详细注释代码。

C++

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6
7  static int plaintextLength, cipherSize;
8
9  vector<vector<int>> cipherBook;
10
11 vector<vector<int>> directions = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
12
13 string minCipherPath;
14
15 bool found = false;
16
17
18 void dfs(const vector<int>& plaintext, int index, int x, int y, vector<vec
tor<bool>>& visited, vector<string>& path);
19
20 int main() {
21     cin >> plaintextLength;
22     vector<int> plaintext(plaintextLength);
23     for (int& num : plaintext) {
24         cin >> num;
25     }
26
27     cin >> cipherSize;
28     cipherBook.resize(cipherSize, vector<int>(cipherSize));
29     for (auto& row : cipherBook) {
30         for (int& num : row) {
31             cin >> num;
32         }
33     }
34
35     vector<vector<bool>> visited(cipherSize, vector<bool>(cipherSize, fals
e));
36
37
38     for (int i = 0; i < cipherSize; ++i) {
39         for (int j = 0; j < cipherSize; ++j) {
40             if (cipherBook[i][j] == plaintext[0]) {
41                 vector<string> path;
42                 dfs(plaintext, 0, i, j, visited, path);
43             }
```

```

44     }
45 }
46
47     cout << (found ? minCipherPath : "error") << endl;
48     return 0;
49 }
50
51 void dfs(const vector<int>& plaintext, int index, int x, int y, vector<vec
52 tor<bool>>& visited, vector<string>& path) {
53     if (index == plaintextLength) {
54         string currentPath = "";
55         for (const string& step : path) {
56             currentPath += step + " ";
57         }
58         if (!found || currentPath < minCipherPath) {
59             minCipherPath = currentPath;
60         }
61         found = true;
62         return;
63     }
64
65
66     if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x]
67 [y] || cipherBook[x][y] != plaintext[index]) {
68         return;
69     }
70
71     visited[x][y] = true;
72     path.push_back(to_string(x) + " " + to_string(y));
73
74     for (const auto& dir : directions) {
75         dfs(plaintext, index + 1, x + dir[0], y + dir[1], visited, path);
76     }
77
78     visited[x][y] = false;
79     path.pop_back();
80 }

```

## Java

```
1  import java.util.*;
2
3  public class Main {
4      static int plaintextLength, cipherSize;
5      static int[][] cipherBook;
6      static final int[][] directions = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
7      static String minCipherPath = "";
8      static boolean found = false;
9
10     public static void main(String[] args) {
11         Scanner scanner = new Scanner(System.in);
12
13         plaintextLength = scanner.nextInt();
14         int[] plaintext = new int[plaintextLength];
15         for (int i = 0; i < plaintextLength; i++) {
16             plaintext[i] = scanner.nextInt();
17         }
18
19         cipherSize = scanner.nextInt();
20         cipherBook = new int[cipherSize][cipherSize];
21         for (int i = 0; i < cipherSize; i++) {
22             for (int j = 0; j < cipherSize; j++) {
23                 cipherBook[i][j] = scanner.nextInt();
24             }
25         }
26
27         boolean[][] visited = new boolean[cipherSize][cipherSize];
28
29
30         for (int i = 0; i < cipherSize; i++) {
31             for (int j = 0; j < cipherSize; j++) {
32                 if (cipherBook[i][j] == plaintext[0]) {
33                     List<String> path = new ArrayList<>();
34                     dfs(plaintext, 0, i, j, visited, path);
35                 }
36             }
37         }
38
39         System.out.println(found ? minCipherPath.trim() : "error");
40     }
41
42     private static void dfs(int[] plaintext, int index, int x, int y, boolean[][] visited, List<String> path) {
43         if (index == plaintextLength) {
44             String currentPath = String.join(" ", path);
```

```

45         if (!found || currentPath.compareTo(minCipherPath) < 0) {
46             minCipherPath = currentPath;
47         }
48         found = true;
49         return;
50     }
51
52     if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] != plaintext[index]) {
53         return;
54     }
55
56     visited[x][y] = true;
57     path.add(x + " " + y);
58
59     for (int[] dir : directions) {
60         dfs(plaintext, index + 1, x + dir[0], y + dir[1], visited, path);
61     }
62
63     visited[x][y] = false;
64     path.remove(path.size() - 1);
65 }
66 }

```

## Python

```

1  import sys
2
3
4  directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
5
6  def dfs(plaintext, index, x, y, visited, path):
7      global min_cipher_path, found
8
9      if index == len(plaintext):
10         current_path = " ".join(path)
11         if not found or current_path < min_cipher_path:
12             min_cipher_path = current_path
13             found = True
14             return
15
16         if x < 0 or y < 0 or x >= cipher_size or y >= cipher_size or visited
[x][y] or cipher_book[x][y] != plaintext[index]:
17             return
18
19         visited[x][y] = True
20         path.append(f"{x} {y}")
21
22         for dx, dy in directions:
23             dfs(plaintext, index + 1, x + dx, y + dy, visited, path)
24
25         visited[x][y] = False
26         path.pop()
27
28  if __name__ == "__main__":
29      input_data = sys.stdin.read().splitlines()
30      plaintext_length = int(input_data[0])
31      plaintext = list(map(int, input_data[1].split()))
32
33      cipher_size = int(input_data[2])
34      cipher_book = [list(map(int, input_data[i + 3].split())) for i in range(cipher_size)]
35
36      visited = [[False] * cipher_size for _ in range(cipher_size)]
37      min_cipher_path = ""
38      found = False
39
40      for i in range(cipher_size):
41          for j in range(cipher_size):
42              if cipher_book[i][j] == plaintext[0]:
43                  dfs(plaintext, 0, i, j, visited, [])

```

44  
45

```
print(min_cipher_path if found else "error")
```

## 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    let index = 0;
13    const plaintextLength = parseInt(inputLines[index++]);
14    const plaintext = inputLines[index++].split(" ").map(Number);
15    const cipherSize = parseInt(inputLines[index++]);
16
17    const cipherBook = [];
18    for (let i = 0; i < cipherSize; i++) {
19      cipherBook.push(inputLines[index++].split(" ").map(Number));
20    }
21
22    const directions = [[0, 1], [1, 0], [-1, 0], [0, -1]];
23    let minCipherPath = "";
24    let found = false;
25    let visited = Array.from({ length: cipherSize }, () => Array(cipherSize).fill(false));
26
27    function dfs(index, x, y, path) {
28      if (index === plaintextLength) {
29        let currentPath = path.join(" ");
30        if (!found || currentPath < minCipherPath) {
31          minCipherPath = currentPath;
32        }
33        found = true;
34        return;
35      }
36
37      if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] !== plaintext[index]) {
38        return;
39      }
40
41      visited[x][y] = true;
42      path.push(`${x} ${y}`);
43

```



```

44         for (const [dx, dy] of directions) {
45             dfs(index + 1, x + dx, y + dy, path);
46         }
47
48         visited[x][y] = false;
49         path.pop();
50     }
51
52     for (let i = 0; i < cipherSize; i++) {
53         for (let j = 0; j < cipherSize; j++) {
54             if (cipherBook[i][j] === plaintext[0]) {
55                 dfs(0, i, j, []);
56             }
57         }
58     }
59
60     console.log(found ? minCipherPath.trim() : "error");
61 });

```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strings"
8      "strconv"
9  )
10
11  var (
12      plaintextLength int
13      cipherSize int
14      cipherBook [][]int
15      directions = [][]int{{0, 1}, {1, 0}, {-1, 0}, {0, -1}}
16      minCipherPath string
17      found bool
18  )
19
20  func dfs(plaintext []int, index, x, y int, visited [][]bool, path []string) {
21      if index == plaintextLength {
22          currentPath := strings.Join(path, " ")
23          if !found || currentPath < minCipherPath {
24              minCipherPath = currentPath
25          }
26          found = true
27          return
28      }
29
30      if x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] != plaintext[index] {
31          return
32      }
33
34      visited[x][y] = true
35      path = append(path, fmt.Sprintf("%d %d", x, y))
36
37      for _, dir := range directions {
38          dfs(plaintext, index+1, x+dir[0], y+dir[1], visited, path)
39      }
40
41      visited[x][y] = false
42  }
43
```

```

44 func main() {
45     scanner := bufio.NewScanner(os.Stdin)
46
47     scanner.Scan()
48     fmt.Sscan(scanner.Text(), &plaintextLength)
49
50     scanner.Scan()
51     plaintext := []int{}
52     for _, v := range strings.Fields(scanner.Text()) {
53         num, _ := strconv.Atoi(v)
54         plaintext = append(plaintext, num)
55     }
56
57     scanner.Scan()
58     fmt.Sscan(scanner.Text(), &cipherSize)
59
60     cipherBook = make([][]int, cipherSize)
61     visited := make([][]bool, cipherSize)
62     for i := range cipherBook {
63         scanner.Scan()
64         cipherBook[i] = make([]int, cipherSize)
65         visited[i] = make([]bool, cipherSize)
66         for j, v := range strings.Fields(scanner.Text()) {
67             cipherBook[i][j], _ = strconv.Atoi(v)
68         }
69     }
70
71     for i := 0; i < cipherSize; i++ {
72         for j := 0; j < cipherSize; j++ {
73             if cipherBook[i][j] == plaintext[0] {
74                 dfs(plaintext, 0, i, j, visited, []string{})
75             }
76         }
77     }
78
79     if found {
80         fmt.Println(minCipherPath)
81     } else {
82         fmt.Println("error")
83     }
84 }

```



# 华为OD机试 2025 B卷 - 路口最短时间问题 (C++ & Python & JAVA & J

## 路口最短时间问题

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

假定街道是棋盘型的，每格距离相等，车辆通过每格街道需要时间均为 `timePerRoad`；

街道的街口（交叉点）有交通灯，灯的周期 `T (=lights[row][col])` 各不相同；

车辆可直行、左转和右转，其中直行和左转需要等相应 `T` 时间的交通灯才可通行，右转无需等待。

现给出  $n * m$  个街口的交通灯周期，以及起止街口的坐标，计算车辆经过两个街口的最短时间。

其中：

1. 起点和终点的交通灯不计入时间，且可以在任意方向经过街口
2. 不可超出  $n * m$  个街口，不可跳跃，但边线也是道路（即：`lights[0][0] -> lights[0][1]` 是有效路径）

### 输入描述

第一行输入 `n` 和 `m`，以空格分隔 `n`和`m`的范围[1,9]

之后 `n` 行输入 `lights`矩阵，矩阵每行`m`个整数，以空格分隔 值范围[0,120]

之后一行输入 `timePerRoad` [0 600]

之后一行输入 `rowStart colStart`，以空格分隔

最后一行输入 `rowEnd colEnd`，以空格分隔

### 输出描述

`lights[ rowStart ][ colStart ]` 与 `lights[rowEnd][colEnd]` 两个街口之间的最短通行时间

### 用例1

#### 输入

```

1  3 3
2  1 2 3
3  4 5 6
4  7 8 9
5  60
6  0 0
7  2 2

```

## 输出

## 说明

行走路线为 (0,0) -> (0,1) -> (1,1) -> (1,2) -> (2,2) 走了4格路，2个右转，1个左转，共耗时  
 $60+0+60+5+60+0+60=245$

## 题解

思路： BFS + 记忆化搜索

1. 题目描述中地图的规模较小，同时又不是简单的 有向图 / 无向图 最短路问题（不适合用最短路算法），所以本题采用 BFS + 记忆化搜索 解决。
2. 题目涉及方向状态，为了快速判断是否为右转(右转不需要等待)的情况，定义偏移数组时按照顺时针定义，顺序为 上 右 下 左 对应数组下标为 0 1 2 3，这样判断右转就非常顺序  $(nextDir - lastDir + 4) \% 4 == 1$  就是右转。这里一下子看不懂的话，可以先自己模拟一下。
3. 使用 grid 来接收每个灯的周期。定义 visisted[n \* n][4] 来记录到达以指定方向到达该位置的最少时间，初始时全部设为一个较大的值。接下来使用栈来模拟BFS扩散，由于需要保持方向的状态

态，所以栈中保存的结构大概是 `{x, y, dir, usedTime}` 横坐标 纵坐标 方向 使用时间。代码基本逻辑如下：

1. 初始从起点出发进行扩散，此时可以从任意方向出发。向队列中进行加入 `{rowStart, colStart, 0, 0}`, `{rowStart, colStart, 1, 0}`, `{rowStart, colStart, 2, 0}`, `{rowStart, colStart, 3, 0}` .并更新对应 `visited[rowStart * m][dir] = 0`
2. 接下来就是进行正常的BFS扩散了，取出队列收尾元素，得到`[x, y, dir, usedTime]`,尝试向四周进行扩散，说几个注意点：
  1. 边界问题。
  2. 右转判断，利用之前定义的方向偏移数组。
  3. 使用 `visited[][]` 数组进行剪枝。
4. 执行第三步的BFS扩散之后，题目的结果其实就是 `visited[rowEnd * m + colEnd][0]` 四个方向的最小值。

## C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<climits>
8  #include<cstring>
9  #include<queue>
10 using namespace std;
11
12 struct State {
13     int x, y, dir;
14     int usedTime;
15     State(int x, int y, int dir, int usedTime): x(x), y(y), dir(dir), used
Time(usedTime){}
16 };
17
18 // 组的规格
19 int n, m;
20 // 所需时间
21 int timePerRoad;
22
23 // 方向定义: 上0, 右1, 下2, 左3 方向定义为顺时针 方便处理右转
24 int dx[4] = {-1, 0, 1, 0};
25 int dy[4] = {0, 1, 0, -1};
26
27 int BFS(int rowStart, int colStart, int rowEnd, int colEnd, vector<vector<
int>> grid) {
28     vector<vector<int>> visited(n * m, vector<int>(4, INT_MAX));
29
30     queue<State> q;
31     // 初始化可以从四个方向除法
32     for (int dir = 0; dir < 4; dir++) {
33         visited[rowStart * m + colStart][dir] = 0;
34         q.push({rowStart, colStart, dir, 0});
35     }
36
37     while (!q.empty()) {
38         auto [x, y, dir, usedTime] = q.front();
39         q.pop();
40         for (int ndir = 0; ndir < 4; ndir++) {
41             int nx = x + dx[ndir];
42             int ny = y + dy[ndir];
43             // 超过边界
```



```

44         if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
45         int currentUsedTime = usedTime;
46         // 判断是否不是右转 上面定义的方向 上 右 下 左 右转就是下一个
47         if ((ndir - dir + 4) % 4 != 1) {
48             // 需要等待灯
49             currentUsedTime += grid[x][y];
50         }
51         // 都需要加
52         currentUsedTime += timePerRoad;
53         // 剪枝
54         if (visited[nx * m + ny][ndir] <= currentUsedTime) {
55             continue;
56         }
57
58         visited[nx * m + ny][ndir] = currentUsedTime;
59         q.push({nx, ny, ndir, currentUsedTime});
60     }
61 }
62
63 int res = INT_MAX;
64 // 获取到达目的地的最少时间
65 for (int i = 0; i < 4; i++) {
66     res = min(res, visited[rowEnd * m + colEnd][i]);
67 }
68
69 return res;
70 }
71
72
73 int main() {
74
75     cin >> n >> m;
76     // 接收地图
77     vector<vector<int>> grid(n, vector<int>(m));
78     for (int i = 0; i < n; i++) {
79         for (int j = 0; j < m; j++) {
80             cin >> grid[i][j];
81         }
82     }
83
84     cin >> timePerRoad;
85     // 开始
86     int rowStart, colStart;
87     // 结束
88     int rowEnd, colEnd;
89     cin >> rowStart >> colStart;
90     cin >> rowEnd >> colEnd;
91     int res = BFS(rowStart, colStart, rowEnd, colEnd, grid);

```

```
92     cout << res;  
93     return 0;  
94 }
```

## JAVA

```

1  import java.util.*;
2
3  public class Main {
4      // 表示状态: 坐标 (x, y), 方向 dir, 到达该点所花的时间 usedTime
5      static class State {
6          int x, y, dir, usedTime;
7
8          State(int x, int y, int dir, int usedTime) {
9              this.x = x;
10             this.y = y;
11             this.dir = dir;
12             this.usedTime = usedTime;
13         }
14     }
15
16     // 地图尺寸
17     static int n, m;
18     // 每段路所需时间
19     static int timePerRoad;
20
21     // 方向定义: 上0, 右1, 下2, 左3 (顺时针, 方便判断右转)
22     static int[] dx = {-1, 0, 1, 0};
23     static int[] dy = {0, 1, 0, -1};
24
25     static int BFS(int rowStart, int colStart, int rowEnd, int colEnd, int[][] grid) {
26         // visited[i][d] 表示到达坐标 i (展开为一维) 并朝向 d 所花费的最少时间
27         int[][] visited = new int[n * m][4];
28         for (int[] arr : visited) {
29             Arrays.fill(arr, Integer.MAX_VALUE);
30         }
31
32         Queue<State> queue = new LinkedList<>();
33
34         // 初始化: 从起点出发, 可以向任意方向走
35         for (int dir = 0; dir < 4; dir++) {
36             visited[rowStart * m + colStart][dir] = 0;
37             queue.offer(new State(rowStart, colStart, dir, 0));
38         }
39
40         while (!queue.isEmpty()) {
41             State cur = queue.poll();
42             int x = cur.x, y = cur.y, dir = cur.dir, usedTime = cur.usedT
ime;
43

```

```

44         // 尝试从当前方向走向 4 个新方向
45         for (int ndir = 0; ndir < 4; ndir++) {
46             int nx = x + dx[ndir];
47             int ny = y + dy[ndir];
48
49             // 越界跳过
50             if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
51
52             int currentUsedTime = usedTime;
53
54             // 判断是否是右转（顺时针下一个方向），否则需要等待灯
55             if ((ndir - dir + 4) % 4 != 1) {
56                 currentUsedTime += grid[x][y];
57             }
58
59             // 无论如何都需要加上通行时间
60             currentUsedTime += timePerRoad;
61
62             // 如果之前到该点该方向所需时间更少，则跳过
63             if (visited[nx * m + ny][ndir] <= currentUsedTime) continue;
64
65             // 更新时间并入队
66             visited[nx * m + ny][ndir] = currentUsedTime;
67             queue.offer(new State(nx, ny, ndir, currentUsedTime));
68         }
69     }
70
71     // 从 4 个方向中取最短时间到达终点
72     int res = Integer.MAX_VALUE;
73     for (int i = 0; i < 4; i++) {
74         res = Math.min(res, visited[rowEnd * m + colEnd][i]);
75     }
76     return res;
77 }
78
79 public static void main(String[] args) {
80     Scanner sc = new Scanner(System.in);
81
82     // 读取行列
83     n = sc.nextInt();
84     m = sc.nextInt();
85
86     // 读取地图代价
87     int[][] grid = new int[n][m];
88     for (int i = 0; i < n; i++) {
89         for (int j = 0; j < m; j++) {
90             grid[i][j] = sc.nextInt();

```

```

91         }
92     }
93
94     // 读取路程通行时间
95     timePerRoad = sc.nextInt();
96
97     // 起点终点坐标
98     int rowStart = sc.nextInt();
99     int colStart = sc.nextInt();
100    int rowEnd = sc.nextInt();
101    int colEnd = sc.nextInt();
102
103    // 执行 BFS 并输出结果
104    int result = BFS(rowStart, colStart, rowEnd, colEnd, grid);
105    System.out.println(result);
106 }
107 }

```

## Python

```
1  from collections import deque
2  import sys
3
4
5  n, m = 0, 0
6
7  time_per_road = 0
8
9
10 dx = [-1, 0, 1, 0]
11 dy = [0, 1, 0, -1]
12
13 def bfs(row_start, col_start, row_end, col_end, grid):
14
15     visited = [[float('inf')] * 4 for _ in range(n * m)]
16
17     q = deque()
18
19     for dir in range(4):
20         visited[row_start * m + col_start][dir] = 0
21         q.append((row_start, col_start, dir, 0))
22
23     while q:
24         x, y, dir, used_time = q.popleft()
25
26         for ndir in range(4):
27             nx, ny = x + dx[ndir], y + dy[ndir]
28             if nx < 0 or nx >= n or ny < 0 or ny >= m:
29                 continue
30
31             current_time = used_time
32
33
34             if (ndir - dir + 4) % 4 != 1:
35                 current_time += grid[x][y]
36
37
38             current_time += time_per_road
39
40             if visited[nx * m + ny][ndir] <= current_time:
41                 continue
42
43             visited[nx * m + ny][ndir] = current_time
44             q.append((nx, ny, ndir, current_time))
45
```

```

46
47     res = float('inf')
48     for i in range(4):
49         res = min(res, visited[row_end * m + col_end][i])
50     return res
51
52 if __name__ == '__main__':
53     n, m = map(int, sys.stdin.readline().split())
54
55     grid = [list(map(int, sys.stdin.readline().split())) for _ in range
56 (n)]
57
58     time_per_road = int(sys.stdin.readline())
59     row_start, col_start = map(int, sys.stdin.readline().split())
60     row_end, col_end = map(int, sys.stdin.readline().split())
61
62     result = bfs(row_start, col_start, row_end, col_end, grid)
63     print(result)

```

## JavaScript

```
1  const readline = require('readline');
2
3
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9  let inputLines = [];
10 rl.on('line', line => {
11     inputLines.push(line);
12 }).on('close', () => {
13     main();
14 });
15
16 function main() {
17     const [n, m] = inputLines[0].split(' ').map(Number);
18     const grid = [];
19
20
21     for (let i = 0; i < n; i++) {
22         grid.push(inputLines[1 + i].split(' ').map(Number));
23     }
24
25     const timePerRoad = parseInt(inputLines[n + 1]);
26     const [rowStart, colStart] = inputLines[n + 2].split(' ').map(Number);
27     const [rowEnd, colEnd] = inputLines[n + 3].split(' ').map(Number);
28
29     const result = bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid);
30     console.log(result);
31 }
32
33
34 const dx = [-1, 0, 1, 0];
35 const dy = [0, 1, 0, -1];
36
37
38 function bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid) {
39     const visited = Array(n * m).fill(0).map(() => Array(4).fill(Infinity));
40     const queue = [];
41
42
```



```

43     for (let dir = 0; dir < 4; dir++) {
44         visited[rowStart * m + colStart][dir] = 0;
45         queue.push({ x: rowStart, y: colStart, dir, usedTime: 0 });
46     }
47
48     while (queue.length > 0) {
49         const { x, y, dir, usedTime } = queue.shift();
50
51         for (let ndir = 0; ndir < 4; ndir++) {
52             const nx = x + dx[ndir];
53             const ny = y + dy[ndir];
54
55
56             if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
57
58             let currentTime = usedTime;
59
60
61             if ((ndir - dir + 4) % 4 !== 1) {
62                 currentTime += grid[x][y];
63             }
64
65
66             currentTime += timePerRoad;
67
68             if (visited[nx * m + ny][ndir] <= currentTime) continue;
69
70             visited[nx * m + ny][ndir] = currentTime;
71             queue.push({ x: nx, y: ny, dir: ndir, usedTime: currentTime
72         });
73     }
74 }
75
76 let res = Infinity;
77 for (let i = 0; i < 4; i++) {
78     res = Math.min(res, visited[rowEnd * m + colEnd][i]);
79 }
80 return res;
81 }

```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "math"
7      "os"
8      "strconv"
9      "strings"
10 )
11
12
13 var dx = []int{-1, 0, 1, 0}
14 var dy = []int{0, 1, 0, -1}
15
16
17 type State struct {
18     x, y, dir, usedTime int
19 }
20
21 func min(a, b int) int {
22     if a < b {
23         return a
24     }
25     return b
26 }
27
28
29 func BFS(n, m int, timePerRoad int, rowStart, colStart, rowEnd, colEnd int, grid [][]int) int {
30
31     visited := make([][]int, n*m)
32     for i := 0; i < n*m; i++ {
33         visited[i] = make([]int, 4)
34         for j := 0; j < 4; j++ {
35             visited[i][j] = math.MaxInt32
36         }
37     }
38
39     queue := []State{}
40
41
42     for dir := 0; dir < 4; dir++ {
43         visited[rowStart*m+colStart][dir] = 0
44         queue = append(queue, State{rowStart, colStart, dir, 0})
```

```

45     }
46
47     for len(queue) > 0 {
48         cur := queue[0]
49         queue = queue[1:]
50
51         for ndir := 0; ndir < 4; ndir++ {
52             nx := cur.x + dx[ndir]
53             ny := cur.y + dy[ndir]
54
55
56             if nx < 0 || nx >= n || ny < 0 || ny >= m {
57                 continue
58             }
59
60             currentTime := cur.usedTime
61
62
63             if (ndir-cur.dir+4)%4 != 1 {
64                 currentTime += grid[cur.x][cur.y]
65             }
66
67
68             currentTime += timePerRoad
69
70             if visited[nx*m+ny][ndir] <= currentTime {
71                 continue
72             }
73
74             visited[nx*m+ny][ndir] = currentTime
75             queue = append(queue, State{nx, ny, ndir, currentTime})
76         }
77     }
78
79
80     res := math.MaxInt32
81     for i := 0; i < 4; i++ {
82         res = min(res, visited[rowEnd*m+colEnd][i])
83     }
84
85     return res
86 }
87
88 func main() {
89     scanner := bufio.NewScanner(os.Stdin)
90     scanner.Scan()
91     dims := strings.Fields(scanner.Text())
92     n, _ := strconv.Atoi(dims[0])

```

```

93     m, _ := strconv.Atoi(dims[1])
94
95     grid := make([][]int, n)
96     for i := 0; i < n; i++ {
97         scanner.Scan()
98         rowStr := strings.Fields(scanner.Text())
99         row := make([]int, m)
100         for j := 0; j < m; j++ {
101             row[j], _ = strconv.Atoi(rowStr[j])
102         }
103         grid[i] = row
104     }
105
106     scanner.Scan()
107     timePerRoad, _ := strconv.Atoi(scanner.Text())
108
109     scanner.Scan()
110     start := strings.Fields(scanner.Text())
111     rowStart, _ := strconv.Atoi(start[0])
112     colStart, _ := strconv.Atoi(start[1])
113
114     scanner.Scan()
115     end := strings.Fields(scanner.Text())
116     rowEnd, _ := strconv.Atoi(end[0])
117     colEnd, _ := strconv.Atoi(end[1])
118
119     result := BFS(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, gr
120 id)
121     fmt.Println(result)
122 }

```

来自: [华为OD机试 2025 B卷 – 路口最短时间问题 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

## 路口最短时间问题

[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

假定街道是棋盘型的，每格距离相等，车辆通过每格街道需要时间均为 `timePerRoad`；

街道的街口（交叉点）有交通灯，灯的周期 `T` (`=lights[row][col]`) 各不相同；

车辆可直行、左转和右转，其中直行和左转需要等相应 `T` 时间的交通灯才可通行，右转无需等待。

现给出  $n * m$  个街口的交通灯周期，以及起止街口的坐标，计算车辆经过两个街口的最短时间。

其中：

- 1. 起点和终点的交通灯不计入时间，且可以在任意方向经过街口
- 2. 不可超出  $n * m$  个街口，不可跳跃，但边线也是道路（即：lights[0][0] -> lights[0][1] 是有效路径）

## 输入描述

第一行输入  $n$  和  $m$ ，以空格分隔  $n$ 和 $m$ 的范围[1,9]

之后  $n$  行输入 lights矩阵，矩阵每行 $m$ 个整数，以空格分隔 值范围[0,120]

之后一行输入 timePerRoad [0 600]

之后一行输入 rowStart colStart，以空格分隔

最后一行输入 rowEnd colEnd，以空格分隔

## 输出描述

lights[ rowStart ][ colStart ] 与 lights[rowEnd][colEnd] 两个街口之间的最短通行时间

## 用例1

### 输入

▼ Plain Text |

```
1 3 3
2 1 2 3
3 4 5 6
4 7 8 9
5 60
6 0 0
7 2 2
```

### 输出

### 说明

行走路线为 (0,0) -> (0,1) -> (1,1) -> (1,2) -> (2,2) 走了4格路，2个右转，1个左转，共耗时  
 $60+0+60+5+60+0+60=245$

## 题解

思路： BFS + 记忆化搜索

1. 题目描述中地图的规模较小，同时又不是简单的 有向图 / 无向图 最短路问题（不适合用最短路算法），所以本题采用 BFS + 记忆化搜索 解决。
2. 题目涉及方向状态，为了快速判断是否为右转(右转不需要等待)的情况，定义偏移数组时按照顺时针定义，顺序为 上 右 下 左 对应数组下标为 0 1 2 3，这样判断右转就非常顺序  $(nextDir - lastDir + 4) \% 4 == 1$  就是右转。这里一下子看不懂的话，可以先自己模拟一下。
3. 使用 grid 来接收每个灯的周期。定义 visisted[n \* n][4] 来记录到达以指定方向到达该位置的最少时间，初始时全部设为一个较大的值。接下来使用栈来模拟BFS扩散，由于需要保持方向的状态，所以栈中保存的结构大概是 {x, y, dir, usedTime} 横坐标 纵坐标 方向 使用时间。代码基本逻辑如下：
  1. 初始从起点出发进行扩散，此时可以从任意方向出发。向队列中进行加入 {rowStart, colStart, 0, 0}, {rowStart, colStart, 1, 0}, {rowStart, colStart, 2, 0}, {rowStart, colStart, 3, 0} .并更新对应 visisted[rowStart \* m][dir] = 0
  2. 接下来就是进行正常的BFS扩散了，取出队列收尾元素，得到[x, y, dir, usedTime],尝试向四周进行扩散，说几个注意点：
    1. 边界问题。
    2. 右转判断，利用之前定义的方向偏移数组。
    3. 使用 visisted[][] 数组进行剪枝。
4. 执行第三步的BFS扩散之后，题目的结果其实就是 visisted[rowEnd \* m + colEnd][] 四个方向的最小值。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<climits>
8  #include<cstring>
9  #include<queue>
10 using namespace std;
11
12 struct State {
13     int x, y, dir;
14     int usedTime;
15     State(int x, int y, int dir, int usedTime): x(x), y(y), dir(dir), used
Time(usedTime){}
16 };
17
18 // 组的规格
19 int n, m;
20 // 所需时间
21 int timePerRoad;
22
23 // 方向定义: 上0, 右1, 下2, 左3 方向定义为顺时针 方便处理右转
24 int dx[4] = {-1, 0, 1, 0};
25 int dy[4] = {0, 1, 0, -1};
26
27 int BFS(int rowStart, int colStart, int rowEnd, int colEnd, vector<vector<
int>> grid) {
28     vector<vector<int>> visited(n * m, vector<int>(4, INT_MAX));
29
30     queue<State> q;
31     // 初始化可以从四个方向除法
32     for (int dir = 0; dir < 4; dir++) {
33         visited[rowStart * m + colStart][dir] = 0;
34         q.push({rowStart, colStart, dir, 0});
35     }
36
37     while (!q.empty()) {
38         auto [x, y, dir, usedTime] = q.front();
39         q.pop();
40         for (int ndir = 0; ndir < 4; ndir++) {
41             int nx = x + dx[ndir];
42             int ny = y + dy[ndir];
43             // 超过边界
```

```

44         if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
45         int currentUsedTime = usedTime;
46         // 判断是否不是右转 上面定义的方向 上 右 下 左 右转就是下一个
47         if ((ndir - dir + 4) % 4 != 1) {
48             // 需要等待灯
49             currentUsedTime += grid[x][y];
50         }
51         // 都需要加
52         currentUsedTime += timePerRoad;
53         // 剪枝
54         if (visited[nx * m + ny][ndir] <= currentUsedTime) {
55             continue;
56         }
57
58         visited[nx * m + ny][ndir] = currentUsedTime;
59         q.push({nx, ny, ndir, currentUsedTime});
60     }
61 }
62
63 int res = INT_MAX;
64 // 获取到达目的地的最少时间
65 for (int i = 0; i < 4; i++) {
66     res = min(res, visited[rowEnd * m + colEnd][i]);
67 }
68
69 return res;
70 }
71
72
73 int main() {
74
75     cin >> n >> m;
76     // 接收地图
77     vector<vector<int>> grid(n, vector<int>(m));
78     for (int i = 0; i < n; i++) {
79         for (int j = 0; j < m; j++) {
80             cin >> grid[i][j];
81         }
82     }
83
84     cin >> timePerRoad;
85     // 开始
86     int rowStart, colStart;
87     // 结束
88     int rowEnd, colEnd;
89     cin >> rowStart >> colStart;
90     cin >> rowEnd >> colEnd;
91     int res = BFS(rowStart, colStart, rowEnd, colEnd, grid);

```



```
92     cout << res;  
93     return 0;  
94 }
```

## JAVA

```

1  import java.util.*;
2
3  public class Main {
4      // 表示状态: 坐标 (x, y), 方向 dir, 到达该点所花的时间 usedTime
5      static class State {
6          int x, y, dir, usedTime;
7
8          State(int x, int y, int dir, int usedTime) {
9              this.x = x;
10             this.y = y;
11             this.dir = dir;
12             this.usedTime = usedTime;
13         }
14     }
15
16     // 地图尺寸
17     static int n, m;
18     // 每段路所需时间
19     static int timePerRoad;
20
21     // 方向定义: 上0, 右1, 下2, 左3 (顺时针, 方便判断右转)
22     static int[] dx = {-1, 0, 1, 0};
23     static int[] dy = {0, 1, 0, -1};
24
25     static int BFS(int rowStart, int colStart, int rowEnd, int colEnd, int[][] grid) {
26         // visited[i][d] 表示到达坐标 i (展开为一维) 并朝向 d 所花费的最少时间
27         int[][] visited = new int[n * m][4];
28         for (int[] arr : visited) {
29             Arrays.fill(arr, Integer.MAX_VALUE);
30         }
31
32         Queue<State> queue = new LinkedList<>();
33
34         // 初始化: 从起点出发, 可以向任意方向走
35         for (int dir = 0; dir < 4; dir++) {
36             visited[rowStart * m + colStart][dir] = 0;
37             queue.offer(new State(rowStart, colStart, dir, 0));
38         }
39
40         while (!queue.isEmpty()) {
41             State cur = queue.poll();
42             int x = cur.x, y = cur.y, dir = cur.dir, usedTime = cur.usedT
ime;
43

```

```

44         // 尝试从当前方向走向 4 个新方向
45         for (int ndir = 0; ndir < 4; ndir++) {
46             int nx = x + dx[ndir];
47             int ny = y + dy[ndir];
48
49             // 越界跳过
50             if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
51
52             int currentUsedTime = usedTime;
53
54             // 判断是否是右转（顺时针下一个方向），否则需要等待灯
55             if ((ndir - dir + 4) % 4 != 1) {
56                 currentUsedTime += grid[x][y];
57             }
58
59             // 无论如何都需要加上通行时间
60             currentUsedTime += timePerRoad;
61
62             // 如果之前到该点该方向所需时间更少，则跳过
63             if (visited[nx * m + ny][ndir] <= currentUsedTime) continue;
64
65             // 更新时间并入队
66             visited[nx * m + ny][ndir] = currentUsedTime;
67             queue.offer(new State(nx, ny, ndir, currentUsedTime));
68         }
69     }
70
71     // 从 4 个方向中取最短时间到达终点
72     int res = Integer.MAX_VALUE;
73     for (int i = 0; i < 4; i++) {
74         res = Math.min(res, visited[rowEnd * m + colEnd][i]);
75     }
76     return res;
77 }
78
79 public static void main(String[] args) {
80     Scanner sc = new Scanner(System.in);
81
82     // 读取行列
83     n = sc.nextInt();
84     m = sc.nextInt();
85
86     // 读取地图代价
87     int[][] grid = new int[n][m];
88     for (int i = 0; i < n; i++) {
89         for (int j = 0; j < m; j++) {
90             grid[i][j] = sc.nextInt();

```

```
91         }
92     }
93
94     // 读取路程通行时间
95     timePerRoad = sc.nextInt();
96
97     // 起点终点坐标
98     int rowStart = sc.nextInt();
99     int colStart = sc.nextInt();
100    int rowEnd = sc.nextInt();
101    int colEnd = sc.nextInt();
102
103    // 执行 BFS 并输出结果
104    int result = BFS(rowStart, colStart, rowEnd, colEnd, grid);
105    System.out.println(result);
106 }
107 }
```

## Python

```
1  from collections import deque
2  import sys
3
4
5  n, m = 0, 0
6
7  time_per_road = 0
8
9
10 dx = [-1, 0, 1, 0]
11 dy = [0, 1, 0, -1]
12
13 def bfs(row_start, col_start, row_end, col_end, grid):
14
15     visited = [[float('inf')] * 4 for _ in range(n * m)]
16
17     q = deque()
18
19     for dir in range(4):
20         visited[row_start * m + col_start][dir] = 0
21         q.append((row_start, col_start, dir, 0))
22
23     while q:
24         x, y, dir, used_time = q.popleft()
25
26         for ndir in range(4):
27             nx, ny = x + dx[ndir], y + dy[ndir]
28             if nx < 0 or nx >= n or ny < 0 or ny >= m:
29                 continue
30
31             current_time = used_time
32
33
34             if (ndir - dir + 4) % 4 != 1:
35                 current_time += grid[x][y]
36
37
38             current_time += time_per_road
39
40             if visited[nx * m + ny][ndir] <= current_time:
41                 continue
42
43             visited[nx * m + ny][ndir] = current_time
44             q.append((nx, ny, ndir, current_time))
45
```

```

46
47     res = float('inf')
48     for i in range(4):
49         res = min(res, visited[row_end * m + col_end][i])
50     return res
51
52 if __name__ == '__main__':
53     n, m = map(int, sys.stdin.readline().split())
54
55     grid = [list(map(int, sys.stdin.readline().split())) for _ in range
56 (n)]
57
58     time_per_road = int(sys.stdin.readline())
59     row_start, col_start = map(int, sys.stdin.readline().split())
60     row_end, col_end = map(int, sys.stdin.readline().split())
61
62     result = bfs(row_start, col_start, row_end, col_end, grid)
63     print(result)

```

## JavaScript

```
1  const readline = require('readline');
2
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout
7  });
8
9  let inputLines = [];
10 rl.on('line', line => {
11   inputLines.push(line);
12 }).on('close', () => {
13   main();
14 });
15
16 function main() {
17   const [n, m] = inputLines[0].split(' ').map(Number);
18   const grid = [];
19
20
21   for (let i = 0; i < n; i++) {
22     grid.push(inputLines[1 + i].split(' ').map(Number));
23   }
24
25   const timePerRoad = parseInt(inputLines[n + 1]);
26   const [rowStart, colStart] = inputLines[n + 2].split(' ').map(Number);
27   const [rowEnd, colEnd] = inputLines[n + 3].split(' ').map(Number);
28
29   const result = bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid);
30   console.log(result);
31 }
32
33
34 const dx = [-1, 0, 1, 0];
35 const dy = [0, 1, 0, -1];
36
37
38 function bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid) {
39   const visited = Array(n * m).fill(0).map(() => Array(4).fill(Infinity));
40   const queue = [];
41
42
```

```

43     for (let dir = 0; dir < 4; dir++) {
44         visited[rowStart * m + colStart][dir] = 0;
45         queue.push({ x: rowStart, y: colStart, dir, usedTime: 0 });
46     }
47
48     while (queue.length > 0) {
49         const { x, y, dir, usedTime } = queue.shift();
50
51         for (let ndir = 0; ndir < 4; ndir++) {
52             const nx = x + dx[ndir];
53             const ny = y + dy[ndir];
54
55
56             if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
57
58             let currentTime = usedTime;
59
60
61             if ((ndir - dir + 4) % 4 !== 1) {
62                 currentTime += grid[x][y];
63             }
64
65
66             currentTime += timePerRoad;
67
68             if (visited[nx * m + ny][ndir] <= currentTime) continue;
69
70             visited[nx * m + ny][ndir] = currentTime;
71             queue.push({ x: nx, y: ny, dir: ndir, usedTime: currentTime
72         });
73     }
74 }
75
76 let res = Infinity;
77 for (let i = 0; i < 4; i++) {
78     res = Math.min(res, visited[rowEnd * m + colEnd][i]);
79 }
80 return res;
81 }

```

Go



```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "math"
7      "os"
8      "strconv"
9      "strings"
10 )
11
12
13 var dx = []int{-1, 0, 1, 0}
14 var dy = []int{0, 1, 0, -1}
15
16
17 type State struct {
18     x, y, dir, usedTime int
19 }
20
21 func min(a, b int) int {
22     if a < b {
23         return a
24     }
25     return b
26 }
27
28
29 func BFS(n, m int, timePerRoad int, rowStart, colStart, rowEnd, colEnd int, grid [][]int) int {
30
31     visited := make([][]int, n*m)
32     for i := 0; i < n*m; i++ {
33         visited[i] = make([]int, 4)
34         for j := 0; j < 4; j++ {
35             visited[i][j] = math.MaxInt32
36         }
37     }
38
39     queue := []State{}
40
41
42     for dir := 0; dir < 4; dir++ {
43         visited[rowStart*m+colStart][dir] = 0
44         queue = append(queue, State{rowStart, colStart, dir, 0})
```

```

45     }
46
47     for len(queue) > 0 {
48         cur := queue[0]
49         queue = queue[1:]
50
51         for ndir := 0; ndir < 4; ndir++ {
52             nx := cur.x + dx[ndir]
53             ny := cur.y + dy[ndir]
54
55
56             if nx < 0 || nx >= n || ny < 0 || ny >= m {
57                 continue
58             }
59
60             currentTime := cur.usedTime
61
62
63             if (ndir-cur.dir+4)%4 != 1 {
64                 currentTime += grid[cur.x][cur.y]
65             }
66
67
68             currentTime += timePerRoad
69
70             if visited[nx*m+ny][ndir] <= currentTime {
71                 continue
72             }
73
74             visited[nx*m+ny][ndir] = currentTime
75             queue = append(queue, State{nx, ny, ndir, currentTime})
76         }
77     }
78
79
80     res := math.MaxInt32
81     for i := 0; i < 4; i++ {
82         res = min(res, visited[rowEnd*m+colEnd][i])
83     }
84
85     return res
86 }
87
88 func main() {
89     scanner := bufio.NewScanner(os.Stdin)
90     scanner.Scan()
91     dims := strings.Fields(scanner.Text())
92     n, _ := strconv.Atoi(dims[0])

```

```

93     m, _ := strconv.Atoi(dims[1])
94
95     grid := make([][]int, n)
96     for i := 0; i < n; i++ {
97         scanner.Scan()
98         rowStr := strings.Fields(scanner.Text())
99         row := make([]int, m)
100         for j := 0; j < m; j++ {
101             row[j], _ = strconv.Atoi(rowStr[j])
102         }
103         grid[i] = row
104     }
105
106     scanner.Scan()
107     timePerRoad, _ := strconv.Atoi(scanner.Text())
108
109     scanner.Scan()
110     start := strings.Fields(scanner.Text())
111     rowStart, _ := strconv.Atoi(start[0])
112     colStart, _ := strconv.Atoi(start[1])
113
114     scanner.Scan()
115     end := strings.Fields(scanner.Text())
116     rowEnd, _ := strconv.Atoi(end[0])
117     colEnd, _ := strconv.Atoi(end[1])
118
119     result := BFS(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, gr
120 id)
121     fmt.Println(result)
122 }

```

来自: [华为OD机试 2025 B卷 – 路口最短时间问题 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

# 华为OD机试 2025 B卷 - 最多几个直角三角形

## (C++ & Python & JAVA &

### 最多几个直角三角形

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

有N条线段，长度分别为 $a[1]-a[n]$ 。

现要求你计算这N条线段最多可以组合成几个直角三角形。

每条线段只能使用一次，每个三角形包含三条线段。

### 输入描述

第一行输入一个正整数T ( $1 \leq T \leq 100$ )，表示有T组测试数据。

对于每组测试数据，接下来有T行，

每行第一个正整数N，表示线段个数 ( $3 \leq N \leq 20$ )，接着是N个正整数，表示每条线段长度，( $0 < a[i] < 100$ )。

### 输出描述

对于每组测试数据输出一行，每行包括一个整数，表示最多能组合的直角三角形个数

### 用例1

#### 输入

#### 输出

#### 说明

可以组成2个直角三角形 (3, 4, 5)、(5, 12, 13)

### 题解

思路: 递归回溯 + 剪枝 处理

对于每组输入使用 递归回溯 + 剪枝 进行求解可组成三角形最大个数。每组数据处理逻辑如下：

1. 数据预处理: 边长存储转换至边长的平方,因为判断直角三角形的公式为  $a^2 + b^2 = c^2$ ,减少后续平方计算,使用 `side[]` 存储。使用 `sideLengthSet` 集合记录出现过的边的平方,用于后续剪枝。将 `side` 进行升序排序。
2. 通过 递归回溯 求解最多能组成多少直角三角形。定义 `used[]` 记录某条边是否被使用过。递归每一层基本逻辑如下:
  - a. 通过第一层循环枚举第一条边 `i`, 如果 `used[i] == true`,直接跳过。
  - b. 通过第二层循环枚举第二层边`j`, 如果 `used[j] == true`, 直接跳过。此时通过两条边可以确定第三条边长度 `side[i] + side[j] == c`,可以通过 `sideLengthSet` 判断 `c` 是否存在来进行剪枝。
  - c. 通过第三层循环第三条边 `m`,如果 `used[m] == true or used[m] != c` 直接跳过。如果 `side[m] > c` 直接结束循环,因为 `side[]` 是递增的。如果存在 `side[m] == c` 将对应三条边标记为已访问,继续下一层递归。
3. 通过2递归回溯逻辑可以计算出每一组输入数据的结果,换行输出即可。

**C++**

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<set>
8  using namespace std;
9
10
11 // 递归回溯枚举三条边 并使用剪枝算法
12 int backtrack(vector<int>& segments, vector<bool>& used, set<int>& sideLengthSet) {
13     int res = 0;
14     int n = segments.size();
15     for (int i = 0; i < n - 2; i++) {
16         // 已使用
17         if (used[i]) {
18             continue;
19         }
20         for (int j = i+1; j < n - 1; j++) {
21             // 已使用
22             if (used[j]) {
23                 continue;
24             }
25             // 第三条边长度平方
26             int c = segments[i] + segments[j];
27             // 不存在指定边 剪枝
28             if (sideLengthSet.find(c) == sideLengthSet.end()) {
29                 continue;
30             }
31             for (int k = j+1; k < n; k++) {
32                 if (used[k]) {
33                     continue;
34                 }
35                 // 剪枝 segments递增的, 后续不可能等于c的边
36                 if (segments[k] > c) {
37                     break;
38                 }
39                 if (segments[k] != c) {
40                     continue;
41                 }
42                 // 递归回溯
43                 used[i] = used[j] = used[k] = true;
44
```

```

45         res = max(res, backtrack(segments, used, sideLengthSet) +
46     1);
47         used[i] = used[j] = used[k] = false;
48     }
49 }
50 }
51 return res;
52 }
53
54 int main() {
55     int t;
56     cin >> t;
57     while (t--) {
58         int n;
59         cin >> n;
60
61         // 存储输入边的平方 因为判断直接公式  $a^2 + b^2 = c^2$ 
62         vector<int> side(n);
63         // 记录出现的边长数 用于后续剪枝
64         set<int> sideLengthSet;
65         // 接收输入
66         for (int i = 0; i < n; i++) {
67             int tmp;
68             cin >> tmp;
69             tmp *= tmp;
70             side[i] = tmp;
71             sideLengthSet.insert(tmp);
72         }
73         // 排序
74         sort(side.begin(), side.end());
75         vector<bool> used(n, false);
76         int res = backtrack(side, used, sideLengthSet);
77         cout << res << endl;
78     }
79     return 0;

```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5      // 递归回溯枚举三条边，并使用剪枝算法
6      public static int backtrack(List<Integer> segments, boolean[] used, Set<Integer> sideLengthSet) {
7          int res = 0;
8          int n = segments.size();
9          for (int i = 0; i < n - 2; i++) {
10             // 已使用
11             if (used[i]) continue;
12             for (int j = i + 1; j < n - 1; j++) {
13                 // 已使用
14                 if (used[j]) continue;
15
16                 // 第三条边长度平方
17                 int c = segments.get(i) + segments.get(j);
18
19                 // 不存在指定边，剪枝
20                 if (!sideLengthSet.contains(c)) continue;
21
22                 for (int k = j + 1; k < n; k++) {
23                     if (used[k]) continue;
24
25                     // 剪枝：segments 递增，后续不可能等于 c 的边
26                     if (segments.get(k) > c) break;
27                     if (!segments.get(k).equals(c)) continue;
28
29                     // 递归回溯
30                     used[i] = used[j] = used[k] = true;
31                     res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
32                     used[i] = used[j] = used[k] = false;
33                 }
34             }
35         }
36         return res;
37     }
38
39     public static void main(String[] args) {
40         Scanner sc = new Scanner(System.in);
41
42         int t = sc.nextInt(); // 读取测试组数
43     }
```



```

44 while (t-- > 0) {
45     int n = sc.nextInt();
46
47     // 存储输入边的平方，因为判断直接公式  $a^2 + b^2 = c^2$ 
48     List<Integer> sides = new ArrayList<>();
49
50     // 记录出现的边长数，用于后续剪枝
51     Set<Integer> sideLengthSet = new HashSet<>();
52
53     // 接收输入
54     for (int i = 0; i < n; i++) {
55         int tmp = sc.nextInt();
56         tmp *= tmp;
57         sides.add(tmp);
58         sideLengthSet.add(tmp);
59     }
60
61     // 排序
62     Collections.sort(sides);
63
64     // 标记每条边是否被使用
65     boolean[] used = new boolean[n];
66
67     int res = backtrack(sides, used, sideLengthSet);
68     System.out.println(res);
69 }
70 }
71 }

```

## Python

```
1 def backtrack(segments, used, side_length_set):
2     """
3     递归回溯枚举三条边，并使用剪枝算法
4     """
5     res = 0
6     n = len(segments)
7     for i in range(n - 2):
8         if used[i]:
9             continue
10        for j in range(i + 1, n - 1):
11            if used[j]:
12                continue
13
14
15            c = segments[i] + segments[j]
16
17
18            if c not in side_length_set:
19                continue
20
21            for k in range(j + 1, n):
22                if used[k]:
23                    continue
24
25
26                if segments[k] > c:
27                    break
28                if segments[k] != c:
29                    continue
30
31
32                used[i] = used[j] = used[k] = True
33                res = max(res, backtrack(segments, used, side_length_set)
34                + 1)
35                used[i] = used[j] = used[k] = False
36
37        return res
38
39 if __name__ == "__main__":
40     t = int(input())
41     for _ in range(t):
42         arr = list(map(int, input().split()))
43         n = arr[0]
44         raw_lengths = arr[1:]
```

```
45
46     side = [x * x for x in raw_lengths]
47
48
49     side_length_set = set(side)
50
51
52     side.sort()
53
54
55     used = [False] * n
56
57     res = backtrack(side, used, side_length_set)
58     print(res)
```

## 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    let t = parseInt(inputLines[0]);
15    let idx = 1;
16
17    for (let test = 0; test < t; test++) {
18      const arr = inputLines[idx++].split(" ").map(Number);
19      const n = arr[0];
20      const rawLengths = arr.slice(1);
21
22
23      const side = rawLengths.map(x => x * x);
24
25
26      const sideLengthSet = new Set(side);
27
28
29      side.sort((a, b) => a - b);
30
31
32      const used = new Array(n).fill(false);
33
34      const res = backtrack(side, used, sideLengthSet);
35      console.log(res);
36    }
37  });
38
39
40  function backtrack(segments, used, sideLengthSet) {
41    let res = 0;
42    const n = segments.length;
43
44    for (let i = 0; i < n - 2; i++) {
45      if (used[i]) continue;
```

```

46
47     for (let j = i + 1; j < n - 1; j++) {
48         if (used[j]) continue;
49
50
51         const c = segments[i] + segments[j];
52
53
54         if (!sideLengthSet.has(c)) continue;
55
56         for (let k = j + 1; k < n; k++) {
57             if (used[k]) continue;
58
59
60             if (segments[k] > c) break;
61             if (segments[k] !== c) continue;
62
63
64             used[i] = used[j] = used[k] = true;
65             res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
66             used[i] = used[j] = used[k] = false;
67         }
68     }
69 }
70 return res;
71 }

```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "sort"
8      "strconv"
9      "strings"
10 )
11
12
13 func backtrack(segments []int, used []bool, sideLengthSet map[int]bool) in
14 t {
15     res := 0
16     n := len(segments)
17     for i := 0; i < n-2; i++ {
18         if used[i] {
19             continue
20         }
21         for j := i + 1; j < n-1; j++ {
22             if used[j] {
23                 continue
24             }
25             c := segments[i] + segments[j]
26
27             if !sideLengthSet[c] {
28                 continue
29             }
30             for k := j + 1; k < n; k++ {
31                 if used[k] {
32                     continue
33                 }
34
35                 if segments[k] > c {
36                     break
37                 }
38                 if segments[k] != c {
39                     continue
40                 }
41
42                 used[i], used[j], used[k] = true, true, true
43                 res = max(res, backtrack(segments, used, sideLengthSet)+1)
44             }
45         }
46     }
47 }
```

```

45         used[i], used[j], used[k] = false, false, false
46     }
47 }
48 }
49 return res
50 }
51 }
52
53 func main() {
54     scanner := bufio.NewScanner(os.Stdin)
55     scanner.Scan()
56     T, _ := strconv.Atoi(scanner.Text())
57
58     for t := 0; t < T; t++ {
59         scanner.Scan()
60         parts := strings.Fields(scanner.Text())
61         n, _ := strconv.Atoi(parts[0])
62
63         side := make([]int, n)
64         sideLengthSet := make(map[int]bool)
65
66         for i := 0; i < n; i++ {
67             val, _ := strconv.Atoi(parts[i+1])
68             val *= val
69             side[i] = val
70             sideLengthSet[val] = true
71         }
72
73         sort.Ints(side)
74         used := make([]bool, n)
75         res := backtrack(side, used, sideLengthSet)
76         fmt.Println(res)
77     }
78 }
79
80 func max(a, b int) int {
81     if a > b {
82         return a
83     }
84     return b
85 }

```

# 最多几个直角三角形

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

有N条线段，长度分别为 $a[1]-a[n]$ 。

现要求你计算这N条线段最多可以组合成几个直角三角形。

每条线段只能使用一次，每个三角形包含三条线段。

## 输入描述

第一行输入一个正整数T ( $1 \leq T \leq 100$ )，表示有T组测试数据。

对于每组测试数据，接下来有T行，

每行第一个正整数N，表示线段个数 ( $3 \leq N \leq 20$ )，接着是N个正整数，表示每条线段长度，( $0 < a[i] < 100$ )。

## 输出描述

对于每组测试数据输出一行，每行包括一个整数，表示最多能组合的直角三角形个数

## 用例1

### 输入

### 输出

## 说明

可以组成2个直角三角形 (3, 4, 5)、(5, 12, 13)

## 题解

思路：递归回溯 + 剪枝 处理

对于每组输入使用 递归回溯 + 剪枝 进行求解可组成三角形最大个数。每组数据处理逻辑如下：

1. 数据预处理: 边长存储转换至边长的平方,因为判断直角三角形的公式为  $a^2 + b^2 = c^2$ ,减少后续平方计算,使用 `side[]` 存储。使用 `sideLengthSet` 集合记录出现过的边的平方,用于后续剪枝。将 `side` 进行升序排序。
2. 通过 递归回溯 求解最多能组成多少直角三角形。定义 `used[]` 记录某条边是否被使用过。递归每一层基本逻辑如下:
  - a. 通过第一层循环枚举第一条边 `i`, 如果 `used[i] == true`,直接跳过。



- b. 通过第二层循环枚举第二层边j, 如果 `used[j] == true` , 直接跳过。此时通过两条边可以确定第三条边长度 `side[i] + side[j] == c` , 可以通过 `sideLengthSet` 判断 `c` 是否存在来进行剪枝。
  - c. 通过第三层循环第三条边 m, 如果 `used[m] == true or used[m] != c` 直接跳过。如果 `side[m] > c` 直接结束循环, 因为 `side[]` 是递增的。如果存在 `side[m] == c` 将对应三条边标记为已访问, 继续下一层递归。
3. 通过2递归回溯逻辑可以计算出每一组输入数据的结果, 换行输出即可。

**C++**

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<set>
8  using namespace std;
9
10
11 // 递归回溯枚举三条边 并使用剪枝算法
12 int backtrack(vector<int>& segments, vector<bool>& used, set<int>& sideLengthSet) {
13     int res = 0;
14     int n = segments.size();
15     for (int i = 0; i < n - 2; i++) {
16         // 已使用
17         if (used[i]) {
18             continue;
19         }
20         for (int j = i+1; j < n - 1; j++) {
21             // 已使用
22             if (used[j]) {
23                 continue;
24             }
25             // 第三条边长度平方
26             int c = segments[i] + segments[j];
27             // 不存在指定边 剪枝
28             if (sideLengthSet.find(c) == sideLengthSet.end()) {
29                 continue;
30             }
31             for (int k = j+1; k < n; k++) {
32                 if (used[k]) {
33                     continue;
34                 }
35                 // 剪枝 segments递增的, 后续不可能等于c的边
36                 if (segments[k] > c) {
37                     break;
38                 }
39                 if (segments[k] != c) {
40                     continue;
41                 }
42                 // 递归回溯
43                 used[i] = used[j] = used[k] = true;
44
```

```

45         res = max(res, backtrack(segments, used, sideLengthSet) +
46     1);
47         used[i] = used[j] = used[k] = false;
48     }
49 }
50 }
51 return res;
52 }
53
54 int main() {
55     int t;
56     cin >> t;
57     while (t--) {
58         int n;
59         cin >> n;
60
61         // 存储输入边的平方 因为判断直接公式  $a^2 + b^2 = c^2$ 
62         vector<int> side(n);
63         // 记录出现的边长数 用于后续剪枝
64         set<int> sideLengthSet;
65         // 接收输入
66         for (int i = 0; i < n; i++) {
67             int tmp;
68             cin >> tmp;
69             tmp *= tmp;
70             side[i] = tmp;
71             sideLengthSet.insert(tmp);
72         }
73         // 排序
74         sort(side.begin(), side.end());
75         vector<bool> used(n, false);
76         int res = backtrack(side, used, sideLengthSet);
77         cout << res << endl;
78     }
79     return 0;
}

```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5      // 递归回溯枚举三条边，并使用剪枝算法
6      public static int backtrack(List<Integer> segments, boolean[] used, Set<Integer> sideLengthSet) {
7          int res = 0;
8          int n = segments.size();
9          for (int i = 0; i < n - 2; i++) {
10             // 已使用
11             if (used[i]) continue;
12             for (int j = i + 1; j < n - 1; j++) {
13                 // 已使用
14                 if (used[j]) continue;
15
16                 // 第三条边长度平方
17                 int c = segments.get(i) + segments.get(j);
18
19                 // 不存在指定边，剪枝
20                 if (!sideLengthSet.contains(c)) continue;
21
22                 for (int k = j + 1; k < n; k++) {
23                     if (used[k]) continue;
24
25                     // 剪枝：segments 递增，后续不可能等于 c 的边
26                     if (segments.get(k) > c) break;
27                     if (!segments.get(k).equals(c)) continue;
28
29                     // 递归回溯
30                     used[i] = used[j] = used[k] = true;
31                     res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
32                     used[i] = used[j] = used[k] = false;
33                 }
34             }
35         }
36         return res;
37     }
38
39     public static void main(String[] args) {
40         Scanner sc = new Scanner(System.in);
41
42         int t = sc.nextInt(); // 读取测试组数
43     }
```

```

44 while (t-- > 0) {
45     int n = sc.nextInt();
46
47     // 存储输入边的平方，因为判断直接公式  $a^2 + b^2 = c^2$ 
48     List<Integer> sides = new ArrayList<>();
49
50     // 记录出现的边长数，用于后续剪枝
51     Set<Integer> sideLengthSet = new HashSet<>();
52
53     // 接收输入
54     for (int i = 0; i < n; i++) {
55         int tmp = sc.nextInt();
56         tmp *= tmp;
57         sides.add(tmp);
58         sideLengthSet.add(tmp);
59     }
60
61     // 排序
62     Collections.sort(sides);
63
64     // 标记每条边是否被使用
65     boolean[] used = new boolean[n];
66
67     int res = backtrack(sides, used, sideLengthSet);
68     System.out.println(res);
69 }
70 }
71 }

```

## Python

```
1 def backtrack(segments, used, side_length_set):
2     """
3     递归回溯枚举三条边，并使用剪枝算法
4     """
5     res = 0
6     n = len(segments)
7     for i in range(n - 2):
8         if used[i]:
9             continue
10        for j in range(i + 1, n - 1):
11            if used[j]:
12                continue
13
14
15            c = segments[i] + segments[j]
16
17
18            if c not in side_length_set:
19                continue
20
21            for k in range(j + 1, n):
22                if used[k]:
23                    continue
24
25
26                if segments[k] > c:
27                    break
28                if segments[k] != c:
29                    continue
30
31
32                used[i] = used[j] = used[k] = True
33                res = max(res, backtrack(segments, used, side_length_set)
34                + 1)
35                used[i] = used[j] = used[k] = False
36
37        return res
38
39 if __name__ == "__main__":
40     t = int(input())
41     for _ in range(t):
42         arr = list(map(int, input().split()))
43         n = arr[0]
44         raw_lengths = arr[1:]
```

```
45
46     side = [x * x for x in raw_lengths]
47
48
49     side_length_set = set(side)
50
51
52     side.sort()
53
54
55     used = [False] * n
56
57     res = backtrack(side, used, side_length_set)
58     print(res)
```

## 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    let t = parseInt(inputLines[0]);
15    let idx = 1;
16
17    for (let test = 0; test < t; test++) {
18      const arr = inputLines[idx++].split(" ").map(Number);
19      const n = arr[0];
20      const rawLengths = arr.slice(1);
21
22
23      const side = rawLengths.map(x => x * x);
24
25
26      const sideLengthSet = new Set(side);
27
28
29      side.sort((a, b) => a - b);
30
31
32      const used = new Array(n).fill(false);
33
34      const res = backtrack(side, used, sideLengthSet);
35      console.log(res);
36    }
37  });
38
39
40  function backtrack(segments, used, sideLengthSet) {
41    let res = 0;
42    const n = segments.length;
43
44    for (let i = 0; i < n - 2; i++) {
45      if (used[i]) continue;
```



```

46
47     for (let j = i + 1; j < n - 1; j++) {
48         if (used[j]) continue;
49
50
51         const c = segments[i] + segments[j];
52
53
54         if (!sideLengthSet.has(c)) continue;
55
56         for (let k = j + 1; k < n; k++) {
57             if (used[k]) continue;
58
59
60             if (segments[k] > c) break;
61             if (segments[k] !== c) continue;
62
63
64             used[i] = used[j] = used[k] = true;
65             res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
66             used[i] = used[j] = used[k] = false;
67         }
68     }
69 }
70 return res;
71 }

```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "sort"
8      "strconv"
9      "strings"
10 )
11
12
13 func backtrack(segments []int, used []bool, sideLengthSet map[int]bool) in
14 t {
15     res := 0
16     n := len(segments)
17     for i := 0; i < n-2; i++ {
18         if used[i] {
19             continue
20         }
21         for j := i + 1; j < n-1; j++ {
22             if used[j] {
23                 continue
24             }
25             c := segments[i] + segments[j]
26
27             if !sideLengthSet[c] {
28                 continue
29             }
30             for k := j + 1; k < n; k++ {
31                 if used[k] {
32                     continue
33                 }
34
35                 if segments[k] > c {
36                     break
37                 }
38                 if segments[k] != c {
39                     continue
40                 }
41
42                 used[i], used[j], used[k] = true, true, true
43                 res = max(res, backtrack(segments, used, sideLengthSet)+1)
44             }
45         }
46     }
47 }
```

```

45         used[i], used[j], used[k] = false, false, false
46     }
47 }
48 }
49 return res
50 }
51 }
52
53 func main() {
54     scanner := bufio.NewScanner(os.Stdin)
55     scanner.Scan()
56     T, _ := strconv.Atoi(scanner.Text())
57
58     for t := 0; t < T; t++ {
59         scanner.Scan()
60         parts := strings.Fields(scanner.Text())
61         n, _ := strconv.Atoi(parts[0])
62
63         side := make([]int, n)
64         sideLengthSet := make(map[int]bool)
65
66         for i := 0; i < n; i++ {
67             val, _ := strconv.Atoi(parts[i+1])
68             val *= val
69             side[i] = val
70             sideLengthSet[val] = true
71         }
72
73         sort.Ints(side)
74         used := make([]bool, n)
75         res := backtrack(side, used, sideLengthSet)
76         fmt.Println(res)
77     }
78 }
79
80 func max(a, b int) int {
81     if a > b {
82         return a
83     }
84     return b
85 }

```



# 华为OD机试 2025 B卷 - 表达式括号匹配 (C++ & Python & JAVA & JS)

## 表达式括号匹配

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

$(1+(2+3)*(3+(8+0))+1-2)$ 这是一个简单的数学表达式,今天不是计算它的值,而是比较它的括号匹配是否正确。

前面这个式子可以简化为 $((()(()))$ 这样的括号我们认为它是匹配正确的,

而 $((()(($ 这样的我们就说他是错误的。注意括号里面的表达式可能是错的,也可能有多个空格,对于这些我们是不用去管的,我们只关心括号是否使用正确。

## 输入描述

给出一行表达式(长度不超过 100)。

## 输出描述

如果匹配正确输出括号的对数, 否则输出-1。

## 用例1

### 输入

### 输出

## 题解

思路: 栈 数据结构运用题。使用栈来处理括号匹配

1. 能够匹配正确前提下, 括号的对数其实就等于 原输入字符串中 ( 或 ) 的数量。
2. 定义栈 `stk` 存储左括号, 用于处理括号的匹配。定义 `count` 存储遇到的左括号数量 (统计右括号也可以), 从前往后遍历输入字符串, 不同字符处理逻辑如下
  - 非括号字符直接跳过。

- 遇到 `(` , `count++`,并压入栈中。
  - 遇到 `)` ,判断此时栈是否为空? 为空则为异常情况, 直接输出-1.不为空, 弹出栈顶字符。
3. 按照2的逻辑处理之后, 最后判断栈是否为空。栈不为空说明, 左右括号数量不一致, 非法输出-1. 为空的情况下输出`count`的值即可。

**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<stack>
10 #include<map>
11 using namespace std;
12
13 int main() {
14     int res = 0;
15     string input;
16     getline(cin, input);
17     // cout << input << endl;
18     // 栈
19     stack<char> st;
20     for (int i = 0; i < input.size(); i++) {
21         char c = input[i];
22         // 遇到非括号字符直接跳过
23         if (c != '(' && c != ')') {
24             continue;
25         }
26         // 左括号压栈
27         if (c == '(' ) {
28             st.push(c);
29         } else {
30             // 不能正常匹配
31             if (st.empty()) {
32                 cout << -1;
33                 return 0;
34             }
35             // 弹出
36             st.pop();
37             res++;
38         }
39     }
40     // 左右数列不相同 不能正确匹配
41     if (!st.empty()) {
42         cout << -1;
43         return 0;
44     }
45     cout << res;
```

```
46     return 0;
47 }
```

## JAVA

Plain Text

```
1  import java.util.*;
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          int res = 0;
8          Stack<Character> stack = new Stack<>();
9
10         for (int i = 0; i < input.length(); i++) {
11             char c = input.charAt(i);
12             // 遇到非括号字符直接跳过
13             if (c != '(' && c != ')') continue;
14
15             // 左括号压栈
16             if (c == '(') {
17                 stack.push(c);
18             } else {
19                 // 不能正常匹配
20                 if (stack.isEmpty()) {
21                     System.out.println(-1);
22                     return;
23                 }
24                 // 弹出
25                 stack.pop();
26                 res++;
27             }
28         }
29
30         // 左右数量不相同, 不能正确匹配
31         if (!stack.isEmpty()) {
32             System.out.println(-1);
33         } else {
34             System.out.println(res);
35         }
36     }
37 }
```



## Python

```
▼ Plain Text |
1  def main():
2      import sys
3      input_str = sys.stdin.readline().strip()
4      stack = []
5      res = 0
6
7      for c in input_str:
8
9          if c not in '()':
10             continue
11
12         if c == '(':
13             stack.append(c)
14         else:
15
16             if not stack:
17                 print(-1)
18                 return
19
20             stack.pop()
21             res += 1
22
23
24     if stack:
25         print(-1)
26     else:
27         print(res)
28
29 if __name__ == "__main__":
30     main()
```

## JavaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({ input: process.stdin });
3
4  rl.on('line', function (input) {
5      const stack = [];
6      let res = 0;
7
8      for (let i = 0; i < input.length; i++) {
9          const c = input[i];
10
11          if (c !== '(' && c !== ')') continue;
12
13          if (c === '(') {
14              stack.push(c);
15          } else {
16
17              if (stack.length === 0) {
18                  console.log(-1);
19                  return;
20              }
21              stack.pop();
22              res++;
23          }
24      }
25
26      if (stack.length !== 0) {
27          console.log(-1);
28      } else {
29          console.log(res);
30      }
31  });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 func main() {
11     reader := bufio.NewReader(os.Stdin)
12     line, _ := reader.ReadString('\n')
13     line = strings.TrimSpace(line)
14
15     stack := []rune{}
16     res := 0
17
18     for _, c := range line {
19
20         if c != '(' && c != ')' {
21             continue
22         }
23
24         if c == '(' {
25             stack = append(stack, c)
26         } else {
27
28             if len(stack) == 0 {
29                 fmt.Println(-1)
30                 return
31             }
32
33             stack = stack[:len(stack)-1]
34             res++
35         }
36     }
37
38     if len(stack) != 0 {
39         fmt.Println(-1)
40     } else {
41         fmt.Println(res)
42     }
43 }
44 }
```

来自: [华为OD机试 2025 B卷 – 表达式括号匹配 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

# 华为OD机试 2025 B卷 - 最大括号深度 (C++ & Python & JAVA & JS)

## 最大括号深度

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

现有一字符串仅由 '(', ')', '{, }', '[, ]' 六种括号组成。

若字符串满足以下条件之一, 则为无效字符串:

1. 任一类型的左右括号数量不相等;
2. 存在未按正确顺序 (先左后右) 闭合的括号。

输出括号的最大嵌套深度, 若字符串无效则输出0。

$0 \leq \text{字符串长度} \leq 100000$

## 输入描述

一个只包括 '(', ')', '{, }', '[, ]' 的字符串

## 输出描述

一个整数, 最大的括号深度

## 用例1

输入

输出

## 用例2

输入

## 输出

## 说明

有效字符串，最大嵌套深度为3

## 用例3

### 输入

### 输出

## 说明

无效字符串，有两种类型的左右括号数量不相等

## 用例4

### 输入

### 输出

## 说明

无效字符串，存在未按正确顺序闭合的括号

## 题解

思路：栈 数据结构运用题。使用 栈 来求最大括号深度，处理逻辑如下：

1. 从前往后遍历输入字符串

- 如果遇到 `[ { (` 之后，直接压入栈。在输入字符串括号匹配无异常的情况下，最大括号深度其实就是为遍历过程中栈的出现过的最大长度
- 如果遇到 `] } )`，判断是否能正确匹配(栈是否为空、栈顶元素是否为对应左括号)，不能正确匹配说明为无效字符串，直接输出0，结束。否则 弹出栈顶元素。

2. 遍历完输入字符串之后，如果栈不为空，说明存在 任一类型的左右括号数量不相等 的情况，直接输

出0。如果为空的情况，输出遍历完过程中记录的最大栈长度即可。

**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<stack>
10 #include<map>
11 using namespace std;
12
13 int main() {
14     int res = 0;
15     string input;
16     getline(cin, input);
17     // 定义括号对应关系, 减少后续进行if判断
18     map<char, char> mp = {
19 {')', '('}, {'}', '{'}, {'}', '['}, {'}', '['}, {'}', '['}, {'}', '['}};
20
21     // 栈
22     stack<char> st;
23     for (int i = 0; i < input.size(); i++) {
24         char c = input[i];
25
26         // 左括号压栈
27         if (c == '(' || c == '{' || c == '[') {
28             st.push(c);
29             res = max(res, (int)st.size());
30         } else {
31             // 无效字符串
32             if (st.empty()) {
33                 cout << 0;
34                 return 0;
35             }
36             char top = st.top();
37             // 无效字符串
38             if (mp[c] != top) {
39                 cout << 0;
40                 return 0;
41             }
42             st.pop();
43         }
44     }
45     // 左右不相同
```



```
46     if (!st.empty()) {  
47         cout << 0;  
48         return 0;  
49     }  
50     cout << res;  
51     return 0;  
52 }
```

## 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          String input = sc.nextLine();
7          int res = 0;
8
9          // 括号映射
10         Map<Character, Character> mp = new HashMap<>();
11         mp.put(')', '(');
12         mp.put(']', '[');
13         mp.put('}', '{');
14
15         Stack<Character> stack = new Stack<>();
16
17         for (int i = 0; i < input.length(); i++) {
18             char c = input.charAt(i);
19
20             // 左括号压栈
21             if (c == '(' || c == '[' || c == '{') {
22                 stack.push(c);
23                 res = Math.max(res, stack.size());
24             } else {
25                 // 无效字符串: 栈空 或者 括号不匹配
26                 if (stack.isEmpty() || stack.peek() != mp.get(c)) {
27                     System.out.println(0);
28                     return;
29                 }
30                 stack.pop();
31             }
32         }
33
34         // 栈未清空, 左右数量不等
35         if (!stack.isEmpty()) {
36             System.out.println(0);
37         } else {
38             System.out.println(res);
39         }
40     }
41 }
```

## Python

```
1  def main():
2      import sys
3      input_line = sys.stdin.readline().strip()
4      stack = []
5      res = 0
6
7
8      mp = {'(': ')', '[': ']', '{': '}'
9
10     for c in input_line:
11
12         if c in '([{':
13             stack.append(c)
14             res = max(res, len(stack))
15         else:
16
17             if not stack or stack[-1] != mp.get(c):
18                 print(0)
19                 return
20             stack.pop()
21
22
23     if stack:
24         print(0)
25     else:
26         print(res)
27
28 if __name__ == '__main__':
29     main()
```

## JavaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({ input: process.stdin });
3
4  rl.on('line', function(line) {
5      const input = line.trim();
6      const stack = [];
7      let res = 0;
8
9      const mp = { ')': '(', ']': '[', '}': '{' };
10
11     for (let c of input) {
12
13         if (c === '(' || c === '[' || c === '{') {
14             stack.push(c);
15             res = Math.max(res, stack.length);
16         } else {
17
18             if (stack.length === 0 || stack[stack.length - 1] !== mp[c]) {
19                 console.log(0);
20                 return;
21             }
22             stack.pop();
23         }
24     }
25
26     if (stack.length !== 0) {
27         console.log(0);
28     } else {
29         console.log(res);
30     }
31 });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strings"
8  )
9
10 func main() {
11     reader := bufio.NewReader(os.Stdin)
12     input, _ := reader.ReadString('\n')
13     input = strings.TrimSpace(input)
14
15     stack := []rune{}
16     res := 0
17
18     mp := map[rune]rune{'(': ')', '[': ']', '{': '}'
19
20     for _, c := range input {
21         if c == '(' || c == '[' || c == '{' {
22             stack = append(stack, c)
23             if len(stack) > res {
24                 res = len(stack)
25             }
26         } else {
27             if len(stack) == 0 || stack[len(stack)-1] != mp[c] {
28                 fmt.Println(0)
29                 return
30             }
31             stack = stack[:len(stack)-1]
32         }
33     }
34
35     if len(stack) != 0 {
36         fmt.Println(0)
37     } else {
38         fmt.Println(res)
39     }
40 }
41
42 }
```

