od0613

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

```
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：

```
COMMAND A; --this is comment
COMMAND --comment
A AND COMMAND B;
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
```

```
COMMAND TABLE IF EXISTS "UNITED STATE";
COMMAND A GREAT (
ID ADSAB,
download_length INTE-GER, -- test
file_name TEXT,
guid TEXT,
mime_type TEXT,
notifica-tionid INTEGER,
original_file_name TEXT,
pause_reason_type INTEGER,
resumable_flag INTEGER,
start_time INTEGER,
state INTEGER,
folder TEXT,
path TEXT,
total_length INTE-GER,
url TEXT
);
```

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++

Plain Text

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

// 统计字符串中的文本数量
int countStatements(const std::string& script) {
    int statementCount = 0;
    bool insideString = false;
    bool insideComment = false;
    char stringQuote = 0;
    bool currentStatementEmpty = true;

    for (size_t i = 0; i < script.length(); ++i) {
        char current = script[i];
        char next = (i + 1 < script.length()) ? script[i + 1] : '\0';

        // 注释块处理：如果在注释中，直到换行才结束注释
        if (insideComment) {
            if (current == '\n') {
                insideComment = false;
            }
            continue;
        }

        // 如果遇到 "--"，且不在字符串中，进入注释模式
        if (!insideString && current == '-' && next == '-') {
            insideComment = true;
            i++; // 跳过第二个 '-'
            continue;
        }

        // 进入字符串模式（单引号或双引号）
        if (!insideString && (current == '\'' || current == '\"')) {
            insideString = true;
            stringQuote = current;
            currentStatementEmpty = false;
            continue;
        }

        // 字符串结束处理（注意转义引号）
        if (insideString && current == stringQuote) {
            // 前面是转移字符
            if (script[i-1] == stringQuote) {
                i++; // 转义引号，跳过
            } else {
                insideString = false;
```

```
            }
            continue;
        }

        // 遇到分号，标记为一条语句结束
        if (!insideString && current == ';') {
            if (!currentStatementEmpty) {
                statementCount++;
                currentStatementEmpty = true;
            }
            continue;
        }

        // 非空白字符表示当前语句非空
        if (!insideString && !isspace(current)) {
            currentStatementEmpty = false;
        }
    }

    // 如果末尾没有分号，也算一条语句
    if (!currentStatementEmpty) {
        statementCount++;
    }

    return statementCount;
}

int main() {
    std::string script, line;
    while (std::getline(std::cin, line)) {
        script += line + "\n";
    }
    std::cout << countStatements(script) << std::endl;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.Scanner;

public class Main {
    // 统计字符串中的语句数量
    public static int countStatements(String script) {
        int statementCount = 0;
        boolean insideString = false;
        boolean insideComment = false;
        char stringQuote = 0;
        boolean currentStatementEmpty = true;

        for (int i = 0; i < script.length(); ++i) {
            char current = script.charAt(i);
            char next = (i + 1 < script.length()) ? script.charAt(i + 1)
: '\0';

            // 注释块处理：如果在注释中，直到换行才结束注释
            if (insideComment) {
                if (current == '\n') {
                    insideComment = false;
                }
                continue;
            }

            // 遇到 "--"，且不在字符串中，进入注释模式
            if (!insideString && current == '-' && next == '-') {
                insideComment = true;
                i++; // 跳过第二个 '-'
                continue;
            }

            // 进入字符串模式（单引号或双引号）
            if (!insideString && (current == '\'' || current == '\"')) {
                insideString = true;
                stringQuote = current;
                currentStatementEmpty = false;
                continue;
            }

            // 字符串结束处理（注意转义引号）
            if (insideString && current == stringQuote) {
                if (i > 0 && script.charAt(i - 1) == stringQuote) {
                    i++; // 转义引号，跳过
                } else {
                    insideString = false;
```

```
                }
                continue;
            }

            // 遇到分号，标记为一条语句结束
            if (!insideString && current == ';') {
                if (!currentStatementEmpty) {
                    statementCount++;
                    currentStatementEmpty = true;
                }
                continue;
            }

            // 非空白字符表示当前语句非空
            if (!insideString && !Character.isWhitespace(current)) {
                currentStatementEmpty = false;
            }
        }

        // 如果末尾没有分号，也算一条语句
        if (!currentStatementEmpty) {
            statementCount++;
        }

        return statementCount;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StringBuilder script = new StringBuilder();
        while (scanner.hasNextLine()) {
            script.append(scanner.nextLine()).append("\n");
        }
        System.out.println(countStatements(script.toString()));
    }
}
```

Python

Plain Text

```
import sys


def count_statements(script):
    statement_count = 0
    inside_string = False
    inside_comment = False
    string_quote = ''
    current_statement_empty = True
    i = 0

    while i < len(script):
        current = script[i]
        next_char = script[i+1] if i + 1 < len(script) else '\0'


        if inside_comment:
            if current == '\n':
                inside_comment = False
            i += 1
            continue


        if not inside_string and current == '-' and next_char == '-':
            inside_comment = True
            i += 2
            continue


        if not inside_string and current in ['\'', '"']:
            inside_string = True
            string_quote = current
            current_statement_empty = False
            i += 1
            continue


        if inside_string and current == string_quote:
            if i > 0 and script[i-1] == string_quote:
                i += 1
            else:
                inside_string = False
            i += 1
            continue

```

```
            if not inside_string and current == ';':
                if not current_statement_empty:
                    statement_count += 1
                    current_statement_empty = True
                i += 1
                continue


            if not inside_string and not current.isspace():
                current_statement_empty = False

            i += 1


        if not current_statement_empty:
            statement_count += 1

        return statement_count

    if __name__ == "__main__":
        script = sys.stdin.read()
        print(count_statements(script))
```

JavaScript

Plain Text

```
function countStatements(script) {
    let statementCount = 0;
    let insideString = false;
    let insideComment = false;
    let stringQuote = '';
    let currentStatementEmpty = true;

    for (let i = 0; i < script.length; i++) {
        const current = script[i];
        const next = (i + 1 < script.length) ? script[i + 1] : '\0';


        if (insideComment) {
            if (current === '\n') {
                insideComment = false;
            }
            continue;
        }


        if (!insideString && current === '-' && next === '-') {
            insideComment = true;
            i++;
            continue;
        }


        if (!insideString && (current === '\'' || current === '"')) {
            insideString = true;
            stringQuote = current;
            currentStatementEmpty = false;
            continue;
        }


        if (insideString && current === stringQuote) {
            if (i > 0 && script[i - 1] === stringQuote) {
                i++;
            } else {
                insideString = false;
            }
            continue;
        }


```

```
        if (!insideString && current === ';') {
            if (!currentStatementEmpty) {
                statementCount++;
                currentStatementEmpty = true;
            }
            continue;
        }


        if (!insideString && !/\s/.test(current)) {
            currentStatementEmpty = false;
        }
    }


    if (!currentStatementEmpty) {
        statementCount++;
    }

    return statementCount;
}


let input = '';
process.stdin.setEncoding('utf-8');
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    console.log(countStatements(input));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "io"
  "os"
  "strings"
  "unicode"
)


func countStatements(script string) int {
  statementCount := 0
  insideString := false
  insideComment := false
  var stringQuote rune
  currentStatementEmpty := true
  runes := []rune(script)

  for i := 0; i < len(runes); i++ {
    current := runes[i]
    var next rune
    if i+1 < len(runes) {
      next = runes[i+1]
    } else {
      next = 0
    }


    if insideComment {
      if current == '\n' {
        insideComment = false
      }
      continue
    }


    if !insideString && current == '-' && next == '-' {
      insideComment = true
      i++
      continue
    }


```

```
      if !insideString && (current == '\'' || current == '"') {
        insideString = true
        stringQuote = current
        currentStatementEmpty = false
        continue
      }


      if insideString && current == stringQuote {
        if i > 0 && runes[i-1] == stringQuote {
          i++
        } else {
          insideString = false
        }
        continue
      }


      if !insideString && current == ';' {
        if !currentStatementEmpty {
          statementCount++
          currentStatementEmpty = true
        }
        continue
      }


      if !insideString && !unicode.IsSpace(current) {
        currentStatementEmpty = false
      }
    }


    if !currentStatementEmpty {
      statementCount++
    }

    return statementCount
}

func main() {
    reader := bufio.NewReader(os.Stdin)
    var builder strings.Builder
    for {
      line, err := reader.ReadString('\n')
      builder.WriteString(line)
      if err == io.EOF {
        break
```

```
        }
      }
      fmt.Println(countStatements(builder.String()))
  }
```

> 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 &

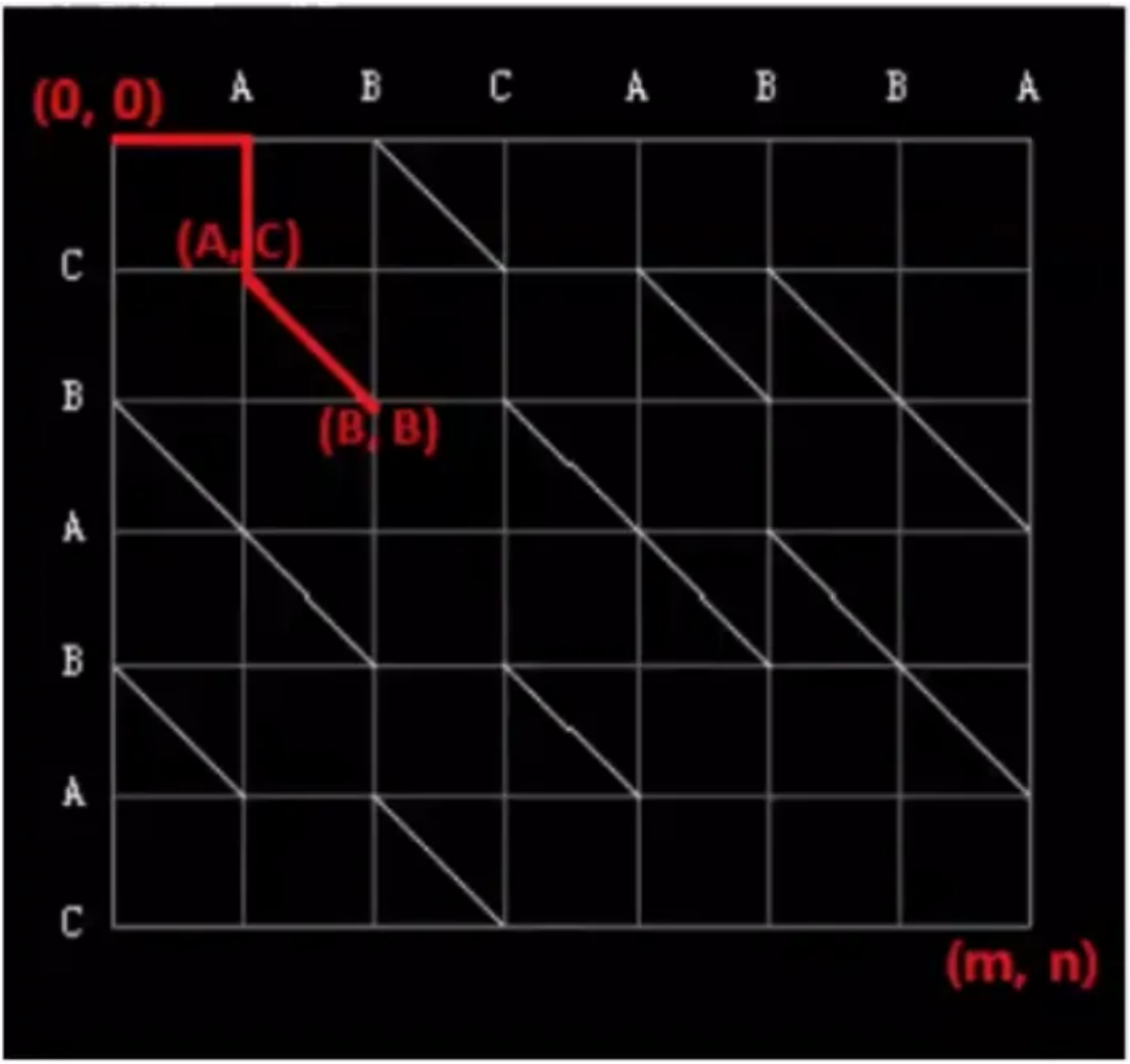
两个字符串间的最短路径

2025B卷目录点击查看： 华为OD机试2025B卷真题题库目录丨机考题库 + 算法考点详解

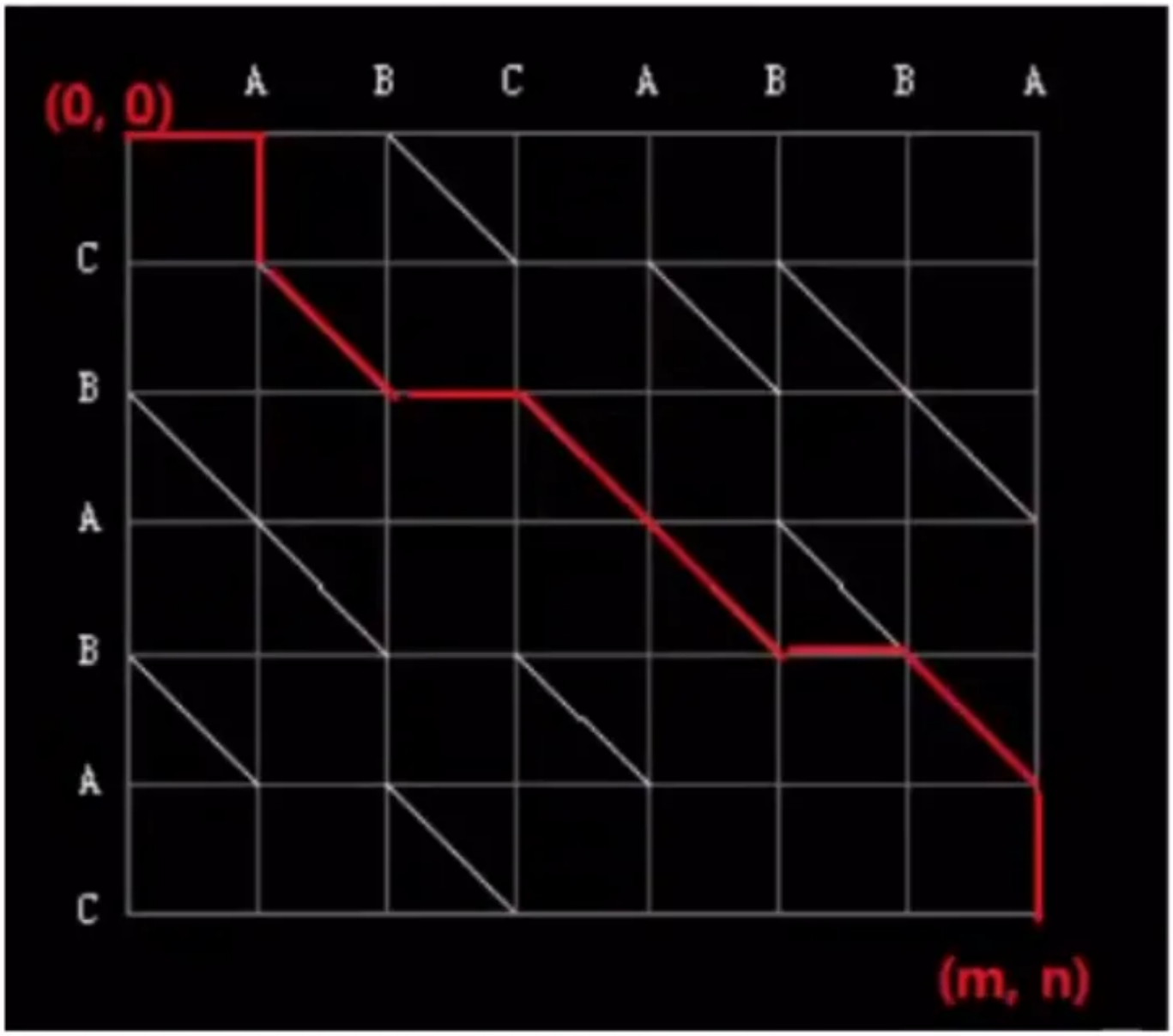
> 2025B卷 200分题型

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，可以观察到状态转义方程只涉及到两行，由此定义两个一维数组进行滚动更新，压缩内存。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<set>
#include<climits>
using namespace std;

int main() {
    string A,B;
    cin>>A>>B;
    int m = A.size(),n = B.size();
    // 使用两个数组压缩数组大小，状态转义方程只涉及两行
    vector<int> dp(m+1,INT_MAX);
    // 记录上一行的状态
    vector<int> pre(m+1, INT_MAX);

    for (int i = 0; i <= m; i++) {
        pre[i] = i;
    }

    for (int i = 1; i <= n; i++) {
        // 重新初始化
        dp.assign(m+1, INT_MAX);
        // (i,0)到(0,0)的距离
        dp[0] = i;
        for (int j = 1; j <= m; j++) {
            dp[j] = min(pre[j], dp[j-1]) + 1;
            if (A[j-1] == B[i-1]) {
                dp[j] = min(dp[j], pre[j-1] + 1);
            }
        }
        pre = dp;
    }

    cout << dp[m];
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String A = sc.next();
        String B = sc.next();
        int m = A.length(), n = B.length();

        // 使用两个数组压缩空间
        int[] dp = new int[m + 1];
        int[] pre = new int[m + 1];

        // 初始化 pre 数组
        for (int i = 0; i <= m; i++) {
            pre[i] = i;
        }

        for (int i = 1; i <= n; i++) {
            // 重新初始化 dp
            Arrays.fill(dp, Integer.MAX_VALUE);
            // (i, 0) 到 (0, 0) 的距离
            dp[0] = i;
            for (int j = 1; j <= m; j++) {
                dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
                if (A.charAt(j - 1) == B.charAt(i - 1)) {
                    dp[j] = Math.min(dp[j], pre[j - 1] + 1);
                }
            }
            // 注意是深拷贝
            pre = Arrays.copyOf(dp, dp.length);
        }

        System.out.println(dp[m]);
    }
}
```

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。

```
import sys

A, B = sys.stdin.read().split()
m, n = len(A), len(B)


pre = [i for i in range(m + 1)]
dp = [0] * (m + 1)

for i in range(1, n + 1):


    dp[0] = i
    for j in range(1, m + 1):
        dp[j] = min(pre[j], dp[j - 1]) + 1
        if A[j - 1] == B[i - 1]:
            dp[j] = min(dp[j], pre[j - 1] + 1)

    for j in range(m+1):
        pre[j] = dp[j]

print(dp[m])
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', function (line) {
    lines.push(line);
    if (lines.length === 1 && lines[0].split(' ').length === 2) {
        rl.close();
    }
});

rl.on('close', function () {
    const [A, B] = lines[0].trim().split(' ');
    const m = A.length, n = B.length;


    let pre = Array.from({ length: m + 1 }, (_, i) => i);
    let dp = new Array(m + 1).fill(0);

    for (let i = 1; i <= n; i++) {
        dp = new Array(m + 1).fill(Infinity);
        dp[0] = i;
        for (let j = 1; j <= m; j++) {
            dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
            if (A[j - 1] === B[i - 1]) {
                dp[j] = Math.min(dp[j], pre[j - 1] + 1);
            }
        }

        pre = [...dp];
    }

    console.log(dp[m]);
});
```

Go

Plain Text

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var A, B string
    fmt.Scan(&A, &B)
    m, n := len(A), len(B)


    pre := make([]int, m+1)
    for i := 0; i <= m; i++ {
        pre[i] = i
    }

    dp := make([]int, m+1)

    for i := 1; i <= n; i++ {
        for j := 0; j <= m; j++ {
            dp[j] = math.MaxInt32
        }
        dp[0] = i
        for j := 1; j <= m; j++ {
            dp[j] = min(pre[j], dp[j-1]) + 1
            if A[j-1] == B[i-1] {
                dp[j] = min(dp[j], pre[j-1]+1)
            }
        }

        copy(pre, dp)
    }

    fmt.Println(dp[m])
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}
```

> 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

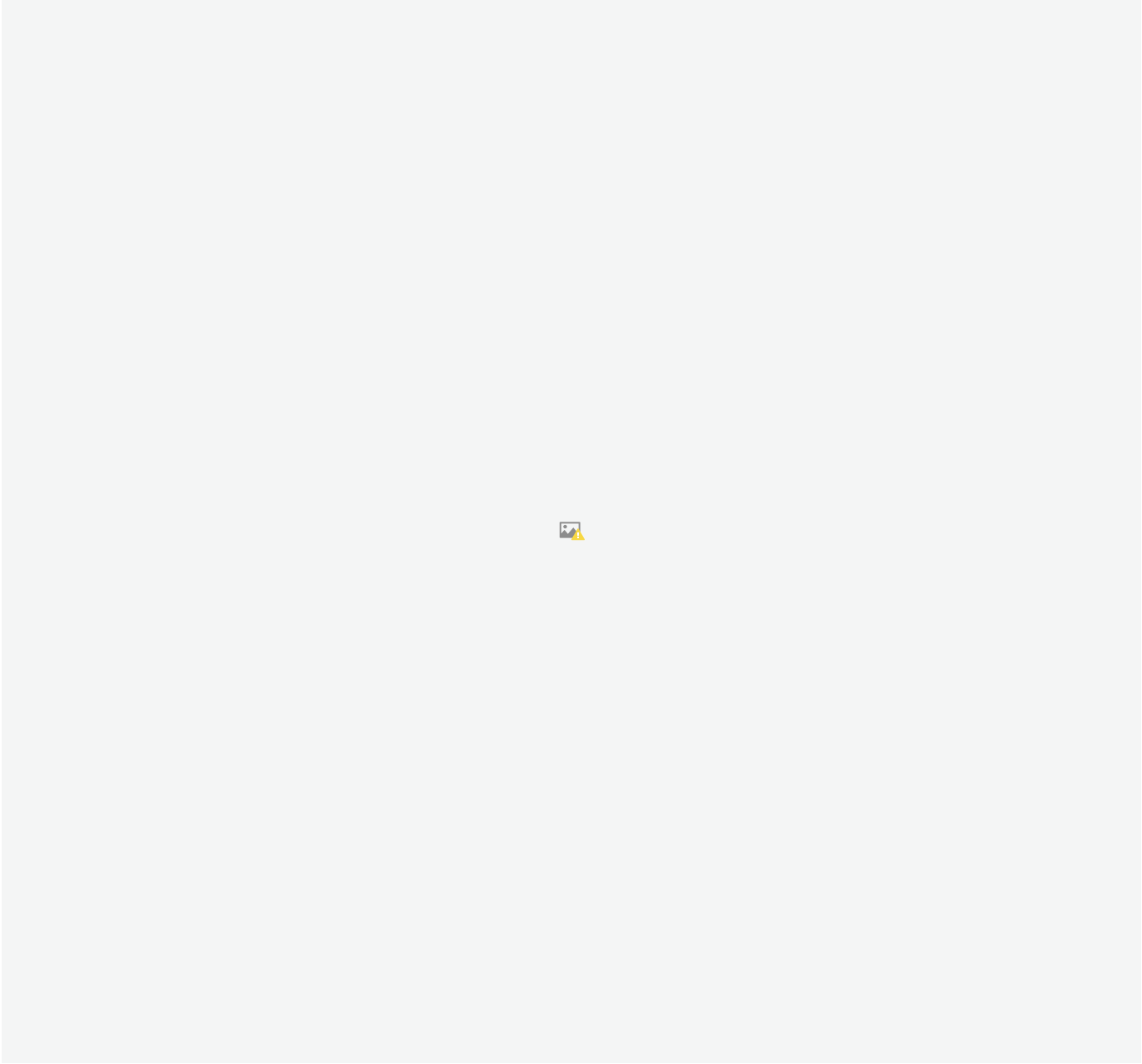
两个字符串间的最短路径

2025B卷目录点击查看： 华为OD机试2025B卷真题题库目录丨机考题库 + 算法考点详解

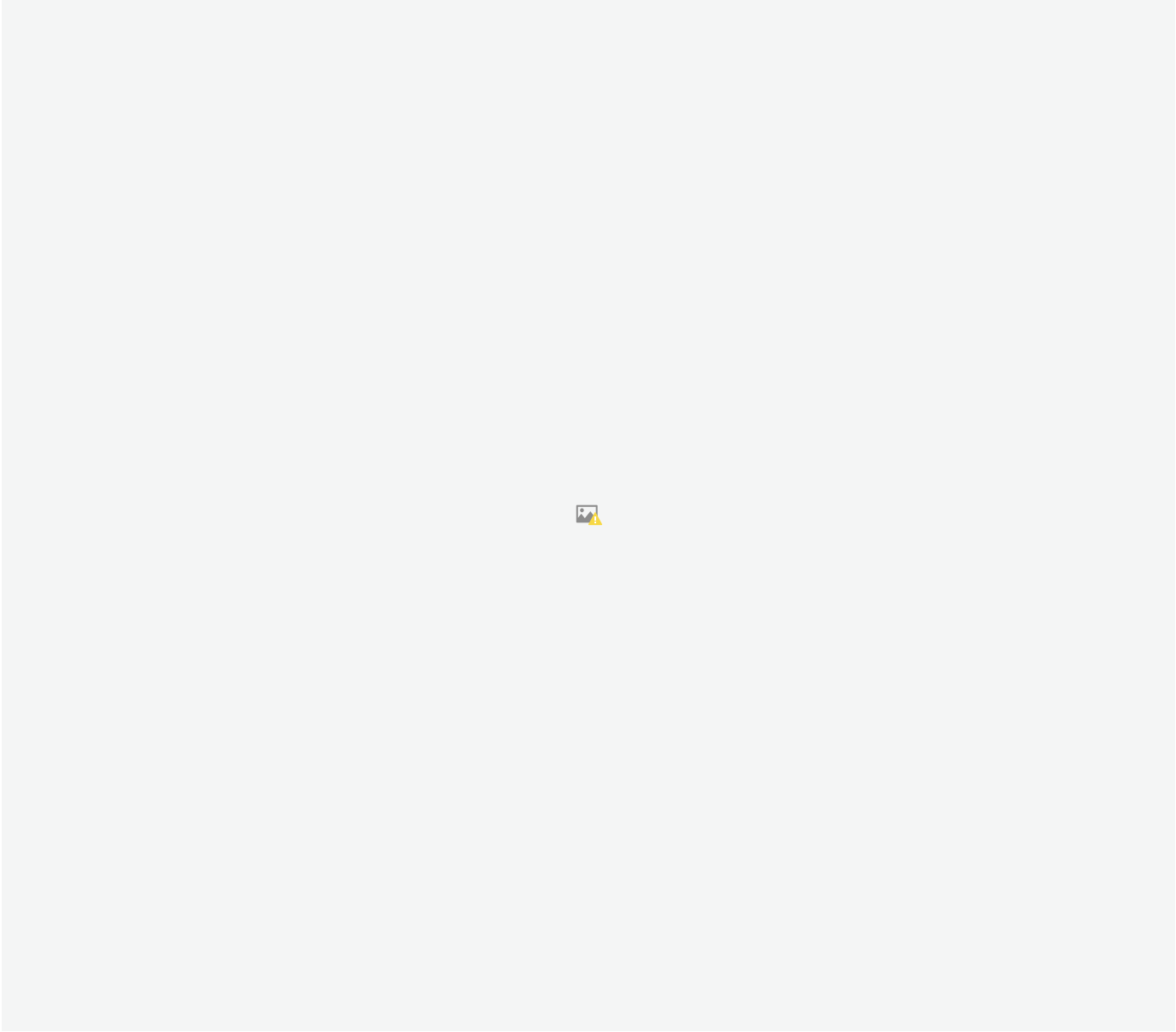
> `2025B卷 200分题型`

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，可以观察到状态转义方程只涉及到两行，由此定义两个一维数组进行滚动更新，压缩内存。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<set>
#include<climits>
using namespace std;

int main() {
    string A,B;
    cin>>A>>B;
    int m = A.size(),n = B.size();
    // 使用两个数组压缩数组大小，状态转义方程只涉及两行
    vector<int> dp(m+1,INT_MAX);
    // 记录上一行的状态
    vector<int> pre(m+1, INT_MAX);

    for (int i = 0; i <= m; i++) {
        pre[i] = i;
    }

    for (int i = 1; i <= n; i++) {
        // 重新初始化
        dp.assign(m+1, INT_MAX);
        // (i,0)到(0,0)的距离
        dp[0] = i;
        for (int j = 1; j <= m; j++) {
            dp[j] = min(pre[j], dp[j-1]) + 1;
            if (A[j-1] == B[i-1]) {
                dp[j] = min(dp[j], pre[j-1] + 1);
            }
        }
        pre = dp;
    }

    cout << dp[m];
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String A = sc.next();
        String B = sc.next();
        int m = A.length(), n = B.length();

        // 使用两个数组压缩空间
        int[] dp = new int[m + 1];
        int[] pre = new int[m + 1];

        // 初始化 pre 数组
        for (int i = 0; i <= m; i++) {
            pre[i] = i;
        }

        for (int i = 1; i <= n; i++) {
            // 重新初始化 dp
            Arrays.fill(dp, Integer.MAX_VALUE);
            // (i, 0) 到 (0, 0) 的距离
            dp[0] = i;
            for (int j = 1; j <= m; j++) {
                dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
                if (A.charAt(j - 1) == B.charAt(i - 1)) {
                    dp[j] = Math.min(dp[j], pre[j - 1] + 1);
                }
            }
            // 注意是深拷贝
            pre = Arrays.copyOf(dp, dp.length);
        }

        System.out.println(dp[m]);
    }
}
```

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。

```
import sys

A, B = sys.stdin.read().split()
m, n = len(A), len(B)


pre = [i for i in range(m + 1)]
dp = [0] * (m + 1)

for i in range(1, n + 1):


    dp[0] = i
    for j in range(1, m + 1):
        dp[j] = min(pre[j], dp[j - 1]) + 1
        if A[j - 1] == B[i - 1]:
            dp[j] = min(dp[j], pre[j - 1] + 1)

    for j in range(m+1):
        pre[j] = dp[j]

print(dp[m])
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', function (line) {
    lines.push(line);
    if (lines.length === 1 && lines[0].split(' ').length === 2) {
        rl.close();
    }
});

rl.on('close', function () {
    const [A, B] = lines[0].trim().split(' ');
    const m = A.length, n = B.length;


    let pre = Array.from({ length: m + 1 }, (_, i) => i);
    let dp = new Array(m + 1).fill(0);

    for (let i = 1; i <= n; i++) {
        dp = new Array(m + 1).fill(Infinity);
        dp[0] = i;
        for (let j = 1; j <= m; j++) {
            dp[j] = Math.min(pre[j], dp[j - 1]) + 1;
            if (A[j - 1] === B[i - 1]) {
                dp[j] = Math.min(dp[j], pre[j - 1] + 1);
            }
        }

        pre = [...dp];
    }

    console.log(dp[m]);
});
```

Go

Plain Text

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var A, B string
    fmt.Scan(&A, &B)
    m, n := len(A), len(B)


    pre := make([]int, m+1)
    for i := 0; i <= m; i++ {
        pre[i] = i
    }

    dp := make([]int, m+1)

    for i := 1; i <= n; i++ {
        for j := 0; j <= m; j++ {
            dp[j] = math.MaxInt32
        }
        dp[0] = i
        for j := 1; j <= m; j++ {
            dp[j] = min(pre[j], dp[j-1]) + 1
            if A[j-1] == B[i-1] {
                dp[j] = min(dp[j], pre[j-1]+1)
            }
        }

        copy(pre, dp)
    }

    fmt.Println(dp[m])
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}
```

> 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

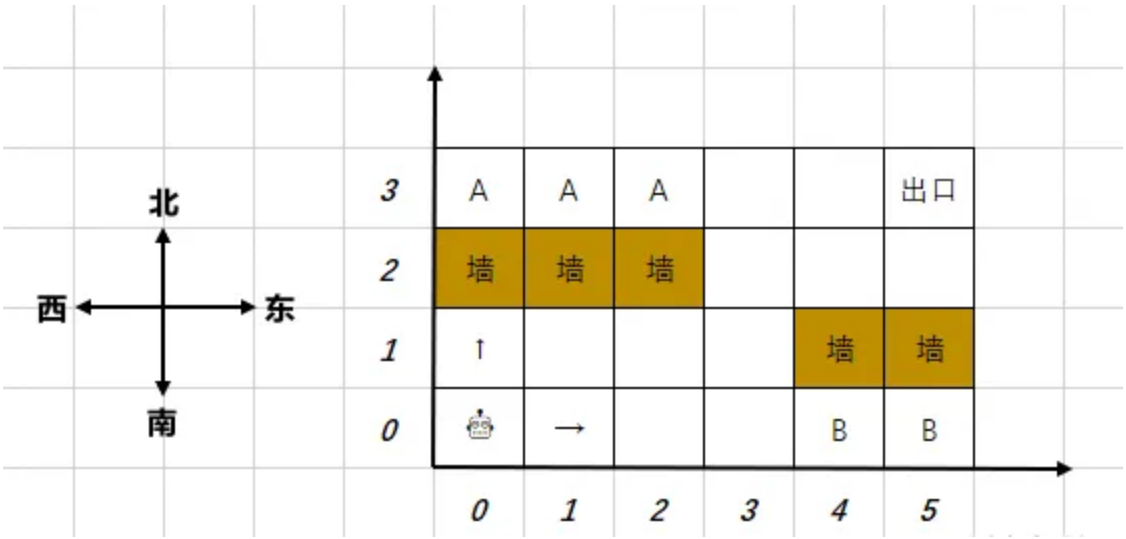
机器人走迷宫

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

```
6 4
5
0 2
1 2
2 2
4 1
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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
using namespace std;
int x,y;
int n;

bool DFS(int currentX, int currentY,  vector<vector<int>>& grid) {
    if (currentX < 0 || currentX >= x) {
        return false;
    }
    if (currentY < 0 || currentY >= y) {
        return false;
    }
    // 墙
    if (grid[currentX][currentY] == -1) {
        return false;
    }
    // 到达之前标记不可达位置
    if (grid[currentX][currentY] == 1) {
        return false;
    }
    // 可达终点
    if (grid[currentX][currentY] == 2) {
        return true;
    }
    bool eastFlag = DFS(currentX + 1, currentY, grid);
    bool northFlag = DFS(currentX, currentY + 1, grid);
    // 可达标记为2
    if (eastFlag || northFlag) {
        grid[currentX][currentY] = 2;
    // 说明可以经过此格子但无法到达终点 标记为1 陷阱方格
    } else {
        grid[currentX][currentY] = 1;
    }
    return eastFlag || northFlag;
}

int main() {
    // 接收输入
    cin >> x >> y;

```

```
    // 初始化地图
    vector<vector<int>> grid(x, vector<int>(y, 0));
    // 处理墙
    cin >> n;
    for (int i = 0; i < n; i++) {
        int x1,  y1;
        cin >> x1 >> y1;
        // 将墙标志位-1
        grid[x1][y1] = -1;
    }

    // 将终点赋值为2，为2的位置代表可达终点
    if (grid[x-1][y -1] != -1) {
        grid[x-1][y-1] = 2;
    }

    // 深度遍历 获取哪些位置可达 哪些不可达

    DFS(0, 0, grid);

    // 统计陷阱放个和不可达数量
    int trapCount, untouchCount;
    trapCount = untouchCount = 0;
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            if (grid[i][j] == 0) {
                untouchCount++;
            }
            if (grid[i][j] == 1) {
                trapCount++;
            }
        }
    }

    cout << trapCount << " " << untouchCount;
    return 0;
}
```

JAVA

```
import java.util.*;

public class Main {
    static int x, y;
    static int[][] grid;

    public static boolean DFS(int currentX, int currentY) {
        if (currentX < 0 || currentX >= x || currentY < 0 || currentY >=
y)
            return false;

        // 墙
        if (grid[currentX][currentY] == -1)
            return false;
        // 到达之前标记不可达位置
        if (grid[currentX][currentY] == 1)
            return false;
        // 可达终点
        if (grid[currentX][currentY] == 2)
            return true;

        boolean eastFlag = DFS(currentX + 1, currentY);
        boolean northFlag = DFS(currentX, currentY + 1);

        // 可达标记为2
        if (eastFlag || northFlag) {
            grid[currentX][currentY] = 2;
        } else {
            // 说明可以经过此格子但无法到达终点，标记为1
            grid[currentX][currentY] = 1;
        }

        return eastFlag || northFlag;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        x = sc.nextInt();
        y = sc.nextInt();
        grid = new int[x][y];

        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int x1 = sc.nextInt();
            int y1 = sc.nextInt();
```

```
                grid[x1][y1] = -1; // 墙
            }
            if (grid[x - 1][y - 1] != -1) {
                grid[x - 1][y - 1] = 2; // 终点标记为2
            }


            DFS(0, 0); // DFS遍历

            int trapCount = 0, untouchCount = 0;
            for (int i = 0; i < x; i++) {
                for (int j = 0; j < y; j++) {
                    if (grid[i][j] == 0)
                        untouchCount++;
                    if (grid[i][j] == 1)
                        trapCount++;
                }
            }

            System.out.println(trapCount + " " + untouchCount);
        }
    }
```

Python

Plain Text

```
import sys

sys.setrecursionlimit(2500)

def DFS(currentX, currentY):
    if currentX < 0 or currentX >= x or currentY < 0 or currentY >= y:
        return False
    if grid[currentX][currentY] == -1:
        return False
    if grid[currentX][currentY] == 1:
        return False
    if grid[currentX][currentY] == 2:
        return True

    eastFlag = DFS(currentX + 1, currentY)
    northFlag = DFS(currentX, currentY + 1)

    if eastFlag or northFlag:
        grid[currentX][currentY] = 2
    else:
        grid[currentX][currentY] = 1
    return eastFlag or northFlag

if __name__ == "__main__":
    x, y = map(int, input().split())
    grid = [[0 for _ in range(y)] for _ in range(x)]

    n = int(input())
    for _ in range(n):
        x1, y1 = map(int, input().split())
        grid[x1][y1] = -1

    if grid[x - 1][y - 1] != -1:

        grid[x - 1][y - 1] = 2

    DFS(0, 0)

    trapCount = untouchCount = 0
    for i in range(x):
        for j in range(y):
            if grid[i][j] == 0:
                untouchCount += 1
            if grid[i][j] == 1:
                trapCount += 1
```

```
46
47
        print(trapCount, untouchCount)
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', function (line) {
    inputLines.push(line);
}).on('close', function () {
    let [x, y] = inputLines[0].split(' ').map(Number);
    let n = parseInt(inputLines[1]);
    let grid = Array.from({ length: x }, () => Array(y).fill(0));

    for (let i = 0; i < n; i++) {
        let [x1, y1] = inputLines[2 + i].split(' ').map(Number);
        grid[x1][y1] = -1;
    }
    if (grid[x - 1][y - 1] != -1) {
        grid[x - 1][y - 1] = 2;
    }


    function DFS(currentX, currentY) {
        if (currentX < 0 || currentX >= x || currentY < 0 || currentY >=
y) return false;
        if (grid[currentX][currentY] === -1) return false;
        if (grid[currentX][currentY] === 1) return false;
        if (grid[currentX][currentY] === 2) return true;

        let eastFlag = DFS(currentX + 1, currentY);
        let northFlag = DFS(currentX, currentY + 1);

        if (eastFlag || northFlag) {
            grid[currentX][currentY] = 2;

        } else {
            grid[currentX][currentY] = 1;
        }

        return eastFlag || northFlag;
    }

    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

Plain Text

```
package main

import (
  "fmt"
)

var (
  x, y int
  grid [][]int
)

func DFS(currentX, currentY int) bool {
  if currentX < 0 || currentX >= x || currentY < 0 || currentY >= y {
    return false
  }
  if grid[currentX][currentY] == -1 {
    return false
  }
  if grid[currentX][currentY] == 1 {
    return false
  }
  if grid[currentX][currentY] == 2 {
    return true
  }

  eastFlag := DFS(currentX+1, currentY)
  northFlag := DFS(currentX, currentY + 1)


  if eastFlag || northFlag {
    grid[currentX][currentY] = 2

  } else {
    grid[currentX][currentY] = 1
  }
  return eastFlag || northFlag
}

func main() {
  var n int
  fmt.Scan(&x, &y)


  grid = make([][]int, x)
  for i := range grid {
```

```
    grid[i] = make([]int, y)
  }

  fmt.Scan(&n)
  for i := 0; i < n; i++ {
    var x1, y1 int
    fmt.Scan(&x1, &y1)
    grid[x1][y1] = -1
  }
    if grid[x-1][y-1] != -1 {
    grid[x-1][y-1] = 2
  }

  DFS(0, 0)

  trapCount, untouchCount := 0, 0
  for i := 0; i < x; i++ {
    for j := 0; j < y; j++ {
      if grid[i][j] == 0 {
        untouchCount++
      }
      if grid[i][j] == 1 {
        trapCount++
      }
    }
  }

  fmt.Printf("%d %d\n", trapCount, untouchCount)
}
```

> 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 &

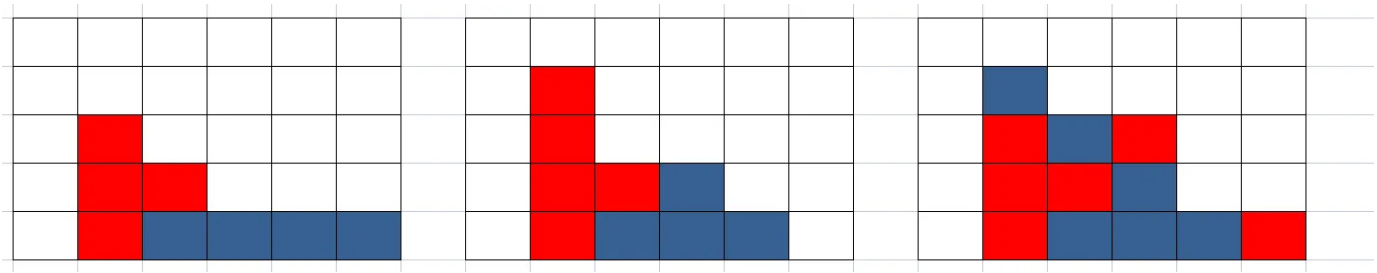
竖直四子棋

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

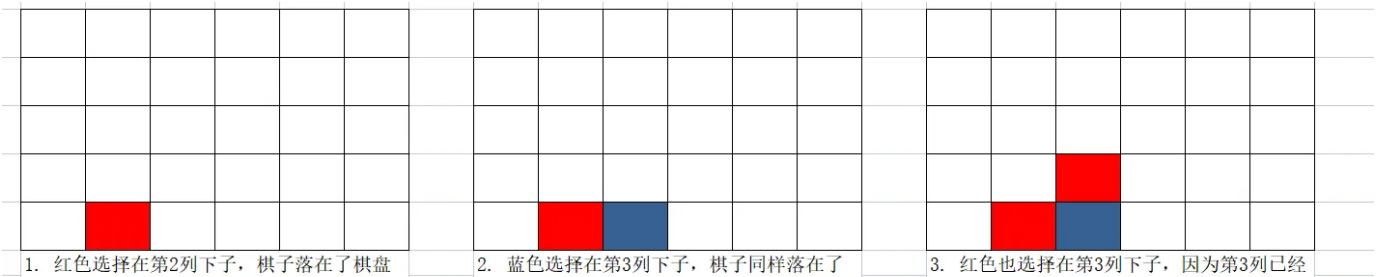
> 华为OD机试2025B卷 200分题型

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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
using namespace std;

// 通用 split 函数
vector<int> split(const string& str, const string& delimiter) {
    vector<int> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(stoi(str.substr(start, end - start)));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(stoi(str.substr(start)));
    return result;
}

// 判断是否存在相同大于等于4连续相连
bool judge(int x , int y, int play, vector<vector<int>>& grid, int n, in
t m) {
    // 上 左 对角线 反对角线
    int direct[4][2] = {
{-1, 0}, {0, -1}, {-1, -1}, {-1, 1}};

    for (int i = 0; i < 4; i++) {
        // 指定方向连续相连子
        int count = 1;
        int currentX = x;
        int currentY = y;
        // 正方向
        while (true) {
            currentX += direct[i][0];
            currentY += direct[i][1];
            if (currentX >= 1 && currentX <= n && currentY >= 1 && curren
tY <= m && grid[currentX][currentY] == play) {
                count++;
            } else {
```

```
                break;
            }
        }
        if (count >= 4) {
            return true;
        }

        currentX = x;
        currentY = y;
        // 反向
        while (true) {
            currentX -= direct[i][0];
            currentY -= direct[i][1];
            if (currentX >= 1 && currentX <= n && currentY >= 1 && curren
tY <= m && grid[currentX][currentY] == play) {
                count++;
            } else {
                break;
            }
        }
        if (count >= 4) {
            return true;
        }
    }
    return false;
}


int main() {
    int n,m;
    // 宽 高
    cin >> m >> n;
    cin.ignore();
    string input;
    getline(cin , input);

    vector<vector<int>> grid(n + 1, vector<int>(m + 1, 0));
    vector<int> move = split(input, " ");

    int k = move.size();
    for (int i = 0; i < k; i++) {
        int x,y;
        y = move[i];
        // 合法性判断
        if (y <= 0 || y > m) {
            cout << i+1 << ",error";
            return 0;
        }
```

```
            int play = (i % 2 == 0) ? 1 : 2;

            // 计算最终会落到的行
            x = n + 1;
            while (x > 1 && grid[x - 1][y] == 0) {
                x--;
            }
            // 合法性判断 列全部满了
            if (x == n + 1) {
                cout << i+1 << ",error";
                return 0;
            }

            grid[x][y] = play;
            // 判断是否形成四子相连的情况
            if (i >= 6 && judge(x, y, play, grid, n, m)) {
                cout << i+1 << ","<< ((play ==1) ? "red" : "blue");
                return 0;
            }
        }

        cout << "0,draw";
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 判断是否有连续 >=4 的相同棋子（上下、左右、两对角）
    static boolean judge(int x, int y, int play, int[][] grid, int n, int m) {
        int[][] directions = {{-1, 0}, {0, -1}, {-1, -1}, {-1, 1}}; // 上、左、对角线、反对角线

        for (int[] dir : directions) {
            int count = 1;
            int cx = x, cy = y;

            // 正方向
            while (true) {
                cx += dir[0];
                cy += dir[1];
                if (cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == play) {
                    count++;
                } else {
                    break;
                }
            }
            if (count >= 4) return true;
            // 反方向
            cx = x; cy = y;
            while (true) {
                cx -= dir[0];
                cy -= dir[1];
                if (cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == play) {
                    count++;
                } else {
                    break;
                }
            }

            if (count >= 4) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        int m = sc.nextInt(), n = sc.nextInt(); // 宽、高
        sc.nextLine();
        String[] moves = sc.nextLine().trim().split(" ");

        int[][] grid = new int[n + 2][m + 2]; // 多加一层防止越界

        for (int i = 0; i < moves.length; i++) {
            int y = Integer.parseInt(moves[i]);

            // 非法列
            if (y <= 0 || y > m) {
                System.out.println((i + 1) + ",error");
                return;
            }

            int x = n + 1;
            while (x > 1 && grid[x - 1][y] == 0) {
                x--;
            }

            // 列满了
            if (x == n + 1) {
                System.out.println((i + 1) + ",error");
                return;
            }

            int play = (i % 2 == 0) ? 1 : 2; // 红蓝交替
            grid[x][y] = play;

            // 从第 7 步开始判断是否成线
            if (i >= 6 && judge(x, y, play, grid, n, m)) {
                System.out.println((i + 1) + "," + (play == 1 ? "red" : "b
lue"));
                return;
            }
        }

        System.out.println("0,draw"); // 无赢家
    }
}
```

Python

Plain Text

```
def judge(x, y, play, grid, n, m):

    directions = [(-1, 0), (0, -1), (-1, -1), (-1, 1)]
    for dx, dy in directions:
        count = 1
        cx, cy = x, y


        while 1 <= cx + dx <= n and 1 <= cy + dy <= m and grid[cx + dx][cy + dy] == play:
            cx += dx
            cy += dy
            count += 1

        if count >= 4:
            return True

        cx, cy = x, y
        while 1 <= cx - dx <= n and 1 <= cy - dy <= m and grid[cx - dx][cy - dy] == play:
            cx -= dx
            cy -= dy
            count += 1

        if count >= 4:
            return True
    return False

def main():
    m, n = map(int, input().split())
    moves = list(map(int, input().split()))
    grid = [[0] * (m + 2) for _ in range(n + 2)]

    for i, y in enumerate(moves):

        if y <= 0 or y > m:
            print(f"{i+1},error")
            return

        x = n + 1
        while x > 1 and grid[x - 1][y] == 0:
            x -= 1


        if x == n + 1:
```

```
            print(f"{i+1},error")
            return

        play = 1 if i % 2 == 0 else 2
        grid[x][y] = play


        if i >= 6 and judge(x, y, play, grid, n, m):
            print(f"{i+1},{'red' if play == 1 else 'blue'}")
            return

    print("0,draw")

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require('readline');
const rl = readline.createInterface({ input: process.stdin });

let lines = [];
rl.on('line', line => {
    lines.push(line.trim());
    if (lines.length === 2) {
        solve();
        rl.close();
    }
});

function solve() {
    let [m, n] = lines[0].split(' ').map(Number);
    let moves = lines[1].split(' ').map(Number);
    let grid = Array.from({ length: n + 2 }, () => Array(m + 2).fill(0));

    for (let i = 0; i < moves.length; i++) {
        let y = moves[i];


        if (y <= 0 || y > m) {
            console.log(`${i + 1},error`);
            return;
        }

        let x = n + 1;
        while (x > 1 && grid[x - 1][y] === 0) x--;


        if (x === n + 1) {
            console.log(`${i + 1},error`);
            return;
        }

        let play = i % 2 === 0 ? 1 : 2;
        grid[x][y] = play;

        if (i >= 6 && judge(x, y, play, grid, n, m)) {
            console.log(`${i + 1},${play === 1 ? 'red' : 'blue'}`);
            return;
        }
    }

    console.log("0,draw");
```

```
}

function judge(x, y, play, grid, n, m) {
    const directions = [[-1, 0], [0, -1], [-1, -1], [-1, 1]];
    for (let [dx, dy] of directions) {
        let count = 1;
        let cx = x, cy = y;


        while (cx + dx >= 1 && cx + dx <= n && cy + dy >= 1 && cy + dy <= m && grid[cx + dx][cy + dy] === play) {
            cx += dx;
            cy += dy;
            count++;
        }
        if (count >= 4) return true;

        cx = x; cy = y;
        while (cx - dx >= 1 && cx - dx <= n && cy - dy >= 1 && cy - dy <= m && grid[cx - dx][cy - dy] === play) {
            cx -= dx;
            cy -= dy;
            count++;
        }

        if (count >= 4) return true;
    }
    return false;
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)


func judge(x, y, play int, grid [][]int, n, m int) bool {
  dirs := [][2]int{{-1, 0}, {0, -1}, {-1, -1}, {-1, 1}}

  for _, d := range dirs {
    count := 1
    cx, cy := x, y


    for {
      cx += d[0]
      cy += d[1]
      if cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == pla
y {
        count++
      } else {
        break
      }
    }
    if count >= 4 {
      return true
    }

    cx, cy = x, y
    for {
      cx -= d[0]
      cy -= d[1]
      if cx >= 1 && cx <= n && cy >= 1 && cy <= m && grid[cx][cy] == pla
y {
        count++
      } else {
        break
      }
    }

```

```
      if count >= 4 {
        return true
      }
    }
    return false
  }

  func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    parts := strings.Fields(scanner.Text())
    m, _ := strconv.Atoi(parts[0])
    n, _ := strconv.Atoi(parts[1])

    scanner.Scan()
    moveStr := strings.Fields(scanner.Text())
    move := make([]int, len(moveStr))
    for i, v := range moveStr {
      move[i], _ = strconv.Atoi(v)
    }

    grid := make([][]int, n+2)
    for i := range grid {
      grid[i] = make([]int, m+2)
    }

    for i := 0; i < len(move); i++ {
      y := move[i]

      if y <= 0 || y > m {
        fmt.Printf("%d,error\n", i+1)
        return
      }

      x := n + 1
      for x > 1 && grid[x-1][y] == 0 {
        x--
      }

      if x == n+1 {
        fmt.Printf("%d,error\n", i+1)
        return
      }
      play := 1
      if i%2 == 1 {
        play = 2
      }
      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 ≤ N ≤ 200）
第二行输入 N 个明文组成的序列 Data[i]（0 ≤ Data[i] ≤ 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

```
2
0 3
3
0 0 2
1 3 4
6 6 4
```

输出

示例2

输入

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

Plain Text

```
#include <iostream>
#include <vector>

using namespace std;


static int plaintextLength, cipherSize;

vector<vector<int>> cipherBook;

vector<vector<int>> directions = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

string minCipherPath;

bool found = false;


void dfs(const vector<int>& plaintext, int index, int x, int y, vector<vec
tor<bool>>& visited, vector<string>& path);

int main() {
    cin >> plaintextLength;
    vector<int> plaintext(plaintextLength);
    for (int& num : plaintext) {
        cin >> num;
    }

    cin >> cipherSize;
    cipherBook.resize(cipherSize, vector<int>(cipherSize));
    for (auto& row : cipherBook) {
        for (int& num : row) {
            cin >> num;
        }
    }

    vector<vector<bool>> visited(cipherSize, vector<bool>(cipherSize, fals
e));


    for (int i = 0; i < cipherSize; ++i) {
        for (int j = 0; j < cipherSize; ++j) {
            if (cipherBook[i][j] == plaintext[0]) {
                vector<string> path;
                dfs(plaintext, 0, i, j, visited, path);
            }
```

```
        }
    }

    cout << (found ? minCipherPath : "error") << endl;
    return 0;
}

void dfs(const vector<int>& plaintext, int index, int x, int y, vector<vector<bool>>& visited, vector<string>& path) {
    if (index == plaintextLength) {

        string currentPath = "";
        for (const string& step : path) {
            currentPath += step + " ";
        }
        if (!found || currentPath < minCipherPath) {
            minCipherPath = currentPath;
        }
        found = true;
        return;
    }


    if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] != plaintext[index]) {
        return;
    }

    visited[x][y] = true;
    path.push_back(to_string(x) + " " + to_string(y));

    for (const auto& dir : directions) {
        dfs(plaintext, index + 1, x + dir[0], y + dir[1], visited, path);
    }

    visited[x][y] = false;
    path.pop_back();
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    static int plaintextLength, cipherSize;
    static int[][] cipherBook;
    static final int[][] directions = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
    static String minCipherPath = "";
    static boolean found = false;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        plaintextLength = scanner.nextInt();
        int[] plaintext = new int[plaintextLength];
        for (int i = 0; i < plaintextLength; i++) {
            plaintext[i] = scanner.nextInt();
        }

        cipherSize = scanner.nextInt();
        cipherBook = new int[cipherSize][cipherSize];
        for (int i = 0; i < cipherSize; i++) {
            for (int j = 0; j < cipherSize; j++) {
                cipherBook[i][j] = scanner.nextInt();
            }
        }

        boolean[][] visited = new boolean[cipherSize][cipherSize];


        for (int i = 0; i < cipherSize; i++) {
            for (int j = 0; j < cipherSize; j++) {
                if (cipherBook[i][j] == plaintext[0]) {
                    List<String> path = new ArrayList<>();
                    dfs(plaintext, 0, i, j, visited, path);
                }
            }
        }

        System.out.println(found ? minCipherPath.trim() : "error");
    }

    private static void dfs(int[] plaintext, int index, int x, int y, bool
ean[][] visited, List<String> path) {
        if (index == plaintextLength) {
            String currentPath = String.join(" ", path);
```

```
            if (!found || currentPath.compareTo(minCipherPath) < 0) {
                minCipherPath = currentPath;
            }
            found = true;
            return;
        }

        if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] != plaintext[index]) {
            return;
        }

        visited[x][y] = true;
        path.add(x + " " + y);

        for (int[] dir : directions) {
            dfs(plaintext, index + 1, x + dir[0], y + dir[1], visited, path);
        }

        visited[x][y] = false;
        path.remove(path.size() - 1);
    }
}
```

Python

Plain Text

```
import sys


directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]

def dfs(plaintext, index, x, y, visited, path):
    global min_cipher_path, found

    if index == len(plaintext):
        current_path = " ".join(path)
        if not found or current_path < min_cipher_path:
            min_cipher_path = current_path
        found = True
        return

    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]:
        return

    visited[x][y] = True
    path.append(f"{x} {y}")

    for dx, dy in directions:
        dfs(plaintext, index + 1, x + dx, y + dy, visited, path)

    visited[x][y] = False
    path.pop()

if __name__ == "__main__":
    input_data = sys.stdin.read().splitlines()
    plaintext_length = int(input_data[0])
    plaintext = list(map(int, input_data[1].split()))

    cipher_size = int(input_data[2])
    cipher_book = [list(map(int, input_data[i + 3].split())) for i in range(cipher_size)]

    visited = [[False] * cipher_size for _ in range(cipher_size)]
    min_cipher_path = ""
    found = False

    for i in range(cipher_size):
        for j in range(cipher_size):
            if cipher_book[i][j] == plaintext[0]:
                dfs(plaintext, 0, i, j, visited, [])
```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line);
}).on('close', () => {
    let index = 0;
    const plaintextLength = parseInt(inputLines[index++]);
    const plaintext = inputLines[index++].split(" ").map(Number);
    const cipherSize = parseInt(inputLines[index++]);

    const cipherBook = [];
    for (let i = 0; i < cipherSize; i++) {
        cipherBook.push(inputLines[index++].split(" ").map(Number));
    }

    const directions = [[0, 1], [1, 0], [-1, 0], [0, -1]];
    let minCipherPath = "";
    let found = false;
    let visited = Array.from({ length: cipherSize }, () => Array(cipherSize).fill(false));

    function dfs(index, x, y, path) {
        if (index === plaintextLength) {
            let currentPath = path.join(" ");
            if (!found || currentPath < minCipherPath) {
                minCipherPath = currentPath;
            }
            found = true;
            return;
        }

        if (x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x][y] || cipherBook[x][y] !== plaintext[index]) {
            return;
        }

        visited[x][y] = true;
        path.push(`${x} ${y}`);

```

```
            for (const [dx, dy] of directions) {
                dfs(index + 1, x + dx, y + dy, path);
            }

            visited[x][y] = false;
            path.pop();
        }

        for (let i = 0; i < cipherSize; i++) {
            for (let j = 0; j < cipherSize; j++) {
                if (cipherBook[i][j] === plaintext[0]) {
                    dfs(0, i, j, []);
                }
            }
        }

        console.log(found ? minCipherPath.trim() : "error");
    });
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strings"
  "strconv"
)

var (
  plaintextLength int
  cipherSize int
  cipherBook [][]int
  directions = [][2]int{{0, 1}, {1, 0}, {-1, 0}, {0, -1}}
  minCipherPath string
  found bool
)

func dfs(plaintext []int, index, x, y int, visited [][]bool, path []strin
g) {
  if index == plaintextLength {
    currentPath := strings.Join(path, " ")
    if !found || currentPath < minCipherPath {
      minCipherPath = currentPath
    }
    found = true
    return
  }

  if x < 0 || y < 0 || x >= cipherSize || y >= cipherSize || visited[x]
[y] || cipherBook[x][y] != plaintext[index] {
    return
  }

  visited[x][y] = true
  path = append(path, fmt.Sprintf("%d %d", x, y))

  for _, dir := range directions {
    dfs(plaintext, index+1, x+dir[0], y+dir[1], visited, path)
  }

  visited[x][y] = false
}

```

```
func main() {
  scanner := bufio.NewScanner(os.Stdin)

  scanner.Scan()
  fmt.Sscan(scanner.Text(), &plaintextLength)

  scanner.Scan()
  plaintext := []int{}
  for _, v := range strings.Fields(scanner.Text()) {
    num, _ := strconv.Atoi(v)
    plaintext = append(plaintext, num)
  }

  scanner.Scan()
  fmt.Sscan(scanner.Text(), &cipherSize)

  cipherBook = make([][]int, cipherSize)
  visited := make([][]bool, cipherSize)
  for i := range cipherBook {
    scanner.Scan()
    cipherBook[i] = make([]int, cipherSize)
    visited[i] = make([]bool, cipherSize)
    for j, v := range strings.Fields(scanner.Text()) {
      cipherBook[i][j], _ = strconv.Atoi(v)
    }
  }

  for i := 0; i < cipherSize; i++ {
    for j := 0; j < cipherSize; j++ {
      if cipherBook[i][j] == plaintext[0] {
        dfs(plaintext, 0, i, j, visited, []string{})
      }
    }
  }

  if found {
    fmt.Println(minCipherPath)
  } else {
    fmt.Println("error")
  }
}
```

> 来自: 华为OD机试2025B卷 – 特殊的加密算法(C++ & Python & JAVA & JS & GO)-CSDN博客

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

```
3 3
1 2 3
4 5 6
7 8 9
60
0 0
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 `visitd[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 `visistd[rowEnd * m + colEnd][]` 四个方向的最小值。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<climits>
#include<cstring>
#include<queue>
using namespace std;

struct State {
    int x, y, dir;
    int usedTime;
    State(int x, int y, int dir, int usedTime): x(x), y(y), dir(dir), used
Time(usedTime){}
};

// 组的规格
int n, m;
// 所需时间
int timePerRoad;

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

int BFS(int rowStart, int colStart, int rowEnd, int colEnd, vector<vector<
int>> grid) {
    vector<vector<int>> visited(n * m, vector<int>(4, INT_MAX));

    queue<State> q;
    // 初始化可以从四个方向除法
    for (int dir = 0; dir < 4; dir++) {
        visited[rowStart * m + colStart][dir] = 0;
        q.push({rowStart, colStart, dir, 0});
    }

    while (!q.empty()) {
        auto [x, y, dir, usedTime] = q.front();
        q.pop();
        for (int ndir = 0; ndir < 4; ndir++) {
            int nx = x + dx[ndir];
            int ny = y + dy[ndir];
            // 超过边界
```

```
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            int currentUsedTime = usedTime;
            // 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
            if ((ndir - dir + 4) % 4 != 1) {
                // 需要等待灯
                currentUsedTime += grid[x][y];
            }
            // 都需要加
            currentUsedTime += timePerRoad;
            // 剪枝
            if (visited[nx * m + ny][ndir] <= currentUsedTime) {
                continue;
            }

            visited[nx * m + ny][ndir]  = currentUsedTime;
            q.push({nx, ny, ndir, currentUsedTime});
        }
    }

    int res = INT_MAX;
    // 获取到达目的地的最少时间
    for (int i = 0; i < 4; i++) {
        res = min(res, visited[rowEnd * m + colEnd][i]);
    }

    return res;
}


int main() {

    cin >> n >> m;
    // 接收地图
    vector<vector<int>> grid(n, vector<int>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    cin >> timePerRoad;
    // 开始
    int rowStart,colStart;
    // 结束
    int rowEnd,colEnd;
    cin >> rowStart >> colStart;
    cin >> rowEnd >> colEnd;
    int res = BFS(rowStart, colStart, rowEnd, colEnd, grid);
```

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

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 表示状态：坐标 (x, y)，方向 dir，到达该点所花的时间 usedTime
    static class State {
        int x, y, dir, usedTime;

        State(int x, int y, int dir, int usedTime) {
            this.x = x;
            this.y = y;
            this.dir = dir;
            this.usedTime = usedTime;
        }
    }

    // 地图尺寸
    static int n, m;
    // 每段路所需时间
    static int timePerRoad;

    // 方向定义：上0，右1，下2，左3（顺时针，方便判断右转）
    static int[] dx = {-1, 0, 1, 0};
    static int[] dy = {0, 1, 0, -1};

    static int BFS(int rowStart, int colStart, int rowEnd, int colEnd, int[][] grid) {
        // 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
        int[][] visited = new int[n * m][4];
        for (int[] arr : visited) {
            Arrays.fill(arr, Integer.MAX_VALUE);
        }

        Queue<State> queue = new LinkedList<>();

        // 初始化：从起点出发，可以向任意方向走
        for (int dir = 0; dir < 4; dir++) {
            visited[rowStart * m + colStart][dir] = 0;
            queue.offer(new State(rowStart, colStart, dir, 0));
        }

        while (!queue.isEmpty()) {
            State cur = queue.poll();
            int x = cur.x, y = cur.y, dir = cur.dir, usedTime = cur.usedTime;

```

```
            // 尝试从当前方向走向 4 个新方向
            for (int ndir = 0; ndir < 4; ndir++) {
                int nx = x + dx[ndir];
                int ny = y + dy[ndir];

                // 越界跳过
                if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;

                int currentUsedTime = usedTime;

                // 判断是否是右转（顺时针下一个方向），否则需要等待灯
                if ((ndir - dir + 4) % 4 != 1) {
                    currentUsedTime += grid[x][y];
                }

                // 无论如何都需要加上通行时间
                currentUsedTime += timePerRoad;

                // 如果之前到该点该方向所需时间更少，则跳过
                if (visited[nx * m + ny][ndir] <= currentUsedTime) contin
ue;

                // 更新时间并入队
                visited[nx * m + ny][ndir] = currentUsedTime;
                queue.offer(new State(nx, ny, ndir, currentUsedTime));
            }
        }

        // 从 4 个方向中取最短时间到达终点
        int res = Integer.MAX_VALUE;
        for (int i = 0; i < 4; i++) {
            res = Math.min(res, visited[rowEnd * m + colEnd][i]);
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // 读取行列
        n = sc.nextInt();
        m = sc.nextInt();

        // 读取地图代价
        int[][] grid = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = sc.nextInt();
```

```
                }
            }

            // 读取路程通行时间
            timePerRoad = sc.nextInt();

            // 起点终点坐标
            int rowStart = sc.nextInt();
            int colStart = sc.nextInt();
            int rowEnd = sc.nextInt();
            int colEnd = sc.nextInt();

            // 执行 BFS 并输出结果
            int result = BFS(rowStart, colStart, rowEnd, colEnd, grid);
            System.out.println(result);
        }
    }
```

Python

Plain Text

```
from collections import deque
import sys


n, m = 0, 0

time_per_road = 0


dx = [-1, 0, 1, 0]
dy = [0, 1, 0, -1]

def bfs(row_start, col_start, row_end, col_end, grid):

    visited = [[float('inf')] * 4 for _ in range(n * m)]

    q = deque()

    for dir in range(4):
        visited[row_start * m + col_start][dir] = 0
        q.append((row_start, col_start, dir, 0))

    while q:
        x, y, dir, used_time = q.popleft()

        for ndir in range(4):
            nx, ny = x + dx[ndir], y + dy[ndir]
            if nx < 0 or nx >= n or ny < 0 or ny >= m:
                continue

            current_time = used_time


            if (ndir - dir + 4) % 4 != 1:
                current_time += grid[x][y]


            current_time += time_per_road

            if visited[nx * m + ny][ndir] <= current_time:
                continue

            visited[nx * m + ny][ndir] = current_time
            q.append((nx, ny, ndir, current_time))

```

```
    res = float('inf')
    for i in range(4):
        res = min(res, visited[row_end * m + col_end][i])
    return res

if __name__ == '__main__':
    n, m = map(int, sys.stdin.readline().split())

    grid = [list(map(int, sys.stdin.readline().split())) for _ in range
(n)]

    time_per_road = int(sys.stdin.readline())
    row_start, col_start = map(int, sys.stdin.readline().split())
    row_end, col_end = map(int, sys.stdin.readline().split())

    result = bfs(row_start, col_start, row_end, col_end, grid)
    print(result)
```

JavaScript

Plain Text

```
const readline = require('readline');


const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', line => {
    inputLines.push(line);
}).on('close', () => {
    main();
});

function main() {
    const [n, m] = inputLines[0].split(' ').map(Number);
    const grid = [];


    for (let i = 0; i < n; i++) {
        grid.push(inputLines[1 + i].split(' ').map(Number));
    }

    const timePerRoad = parseInt(inputLines[n + 1]);
    const [rowStart, colStart] = inputLines[n + 2].split(' ').map(Number);
    const [rowEnd, colEnd] = inputLines[n + 3].split(' ').map(Number);

    const result = bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colE
nd, grid);
    console.log(result);
}


const dx = [-1, 0, 1, 0];
const dy = [0, 1, 0, -1];


function bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid)
{
    const visited = Array(n * m).fill(0).map(() => Array(4).fill(Infinit
y));
    const queue = [];


```

```
    for (let dir = 0; dir < 4; dir++) {
        visited[rowStart * m + colStart][dir] = 0;
        queue.push({ x: rowStart, y: colStart, dir, usedTime: 0 });
    }

    while (queue.length > 0) {
        const { x, y, dir, usedTime } = queue.shift();

        for (let ndir = 0; ndir < 4; ndir++) {
            const nx = x + dx[ndir];
            const ny = y + dy[ndir];


            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;

            let currentTime = usedTime;


            if ((ndir - dir + 4) % 4 !== 1) {
                currentTime += grid[x][y];
            }


            currentTime += timePerRoad;

            if (visited[nx * m + ny][ndir] <= currentTime) continue;

            visited[nx * m + ny][ndir] = currentTime;
            queue.push({ x: nx, y: ny, dir: ndir, usedTime: currentTime
});
        }
    }


    let res = Infinity;
    for (let i = 0; i < 4; i++) {
        res = Math.min(res, visited[rowEnd * m + colEnd][i]);
    }
    return res;
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "math"
  "os"
  "strconv"
  "strings"
)


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


type State struct {
  x, y, dir, usedTime int
}

func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}


func BFS(n, m int, timePerRoad int, rowStart, colStart, rowEnd, colEnd in
t, grid [][]int) int {

  visited := make([][]int, n*m)
  for i := 0; i < n*m; i++ {
    visited[i] = make([]int, 4)
    for j := 0; j < 4; j++ {
      visited[i][j] = math.MaxInt32
    }
  }

  queue := []State{}


  for dir := 0; dir < 4; dir++ {
    visited[rowStart*m+colStart][dir] = 0
    queue = append(queue, State{rowStart, colStart, dir, 0})
```

```
  }

  for len(queue) > 0 {
    cur := queue[0]
    queue = queue[1:]

    for ndir := 0; ndir < 4; ndir++ {
      nx := cur.x + dx[ndir]
      ny := cur.y + dy[ndir]


      if nx < 0 || nx >= n || ny < 0 || ny >= m {
        continue
      }

      currentTime := cur.usedTime


      if (ndir-cur.dir+4)%4 != 1 {
        currentTime += grid[cur.x][cur.y]
      }


      currentTime += timePerRoad

      if visited[nx*m+ny][ndir] <= currentTime {
        continue
      }

      visited[nx*m+ny][ndir] = currentTime
      queue = append(queue, State{nx, ny, ndir, currentTime})
    }
  }


  res := math.MaxInt32
  for i := 0; i < 4; i++ {
    res = min(res, visited[rowEnd*m+colEnd][i])
  }

  return res
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  dims := strings.Fields(scanner.Text())
  n, _ := strconv.Atoi(dims[0])
```

```
    m, _ := strconv.Atoi(dims[1])

    grid := make([][]int, n)
    for i := 0; i < n; i++ {
      scanner.Scan()
      rowStr := strings.Fields(scanner.Text())
      row := make([]int, m)
      for j := 0; j < m; j++ {
        row[j], _ = strconv.Atoi(rowStr[j])
      }
      grid[i] = row
    }

    scanner.Scan()
    timePerRoad, _ := strconv.Atoi(scanner.Text())

    scanner.Scan()
    start := strings.Fields(scanner.Text())
    rowStart, _ := strconv.Atoi(start[0])
    colStart, _ := strconv.Atoi(start[1])

    scanner.Scan()
    end := strings.Fields(scanner.Text())
    rowEnd, _ := strconv.Atoi(end[0])
    colEnd, _ := strconv.Atoi(end[1])

    result := BFS(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, gr
id)
    fmt.Println(result)
}
```

> 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

```
3 3
1 2 3
4 5 6
7 8 9
60
0 0
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`visitd[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`visistd[rowEnd * m + colEnd][]`四个方向的最小值。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<climits>
#include<cstring>
#include<queue>
using namespace std;

struct State {
    int x, y, dir;
    int usedTime;
    State(int x, int y, int dir, int usedTime): x(x), y(y), dir(dir), usedTime(usedTime){}
};

// 组的规格
int n, m;
// 所需时间
int timePerRoad;

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

int BFS(int rowStart, int colStart, int rowEnd, int colEnd, vector<vector<int>> grid) {
    vector<vector<int>> visited(n * m, vector<int>(4, INT_MAX));

    queue<State> q;
    // 初始化可以从四个方向除法
    for (int dir = 0; dir < 4; dir++) {
        visited[rowStart * m + colStart][dir] = 0;
        q.push({rowStart, colStart, dir, 0});
    }

    while (!q.empty()) {
        auto [x, y, dir, usedTime] = q.front();
        q.pop();
        for (int ndir = 0; ndir < 4; ndir++) {
            int nx = x + dx[ndir];
            int ny = y + dy[ndir];
            // 超过边界
```

```
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            int currentUsedTime = usedTime;
            // 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
            if ((ndir - dir + 4) % 4 != 1) {
                // 需要等待灯
                currentUsedTime += grid[x][y];
            }
            // 都需要加
            currentUsedTime += timePerRoad;
            // 剪枝
            if (visited[nx * m + ny][ndir] <= currentUsedTime) {
                continue;
            }

            visited[nx * m + ny][ndir]  = currentUsedTime;
            q.push({nx, ny, ndir, currentUsedTime});
        }
    }

    int res = INT_MAX;
    // 获取到达目的地的最少时间
    for (int i = 0; i < 4; i++) {
        res = min(res, visited[rowEnd * m + colEnd][i]);
    }

    return res;
}


int main() {

    cin >> n >> m;
    // 接收地图
    vector<vector<int>> grid(n, vector<int>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    cin >> timePerRoad;
    // 开始
    int rowStart,colStart;
    // 结束
    int rowEnd,colEnd;
    cin >> rowStart >> colStart;
    cin >> rowEnd >> colEnd;
    int res = BFS(rowStart, colStart, rowEnd, colEnd, grid);
```

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

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 表示状态：坐标 (x, y)，方向 dir，到达该点所花的时间 usedTime
    static class State {
        int x, y, dir, usedTime;

        State(int x, int y, int dir, int usedTime) {
            this.x = x;
            this.y = y;
            this.dir = dir;
            this.usedTime = usedTime;
        }
    }

    // 地图尺寸
    static int n, m;
    // 每段路所需时间
    static int timePerRoad;

    // 方向定义：上0，右1，下2，左3（顺时针，方便判断右转）
    static int[] dx = {-1, 0, 1, 0};
    static int[] dy = {0, 1, 0, -1};

    static int BFS(int rowStart, int colStart, int rowEnd, int colEnd, int[][] grid) {
        // 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
        int[][] visited = new int[n * m][4];
        for (int[] arr : visited) {
            Arrays.fill(arr, Integer.MAX_VALUE);
        }

        Queue<State> queue = new LinkedList<>();

        // 初始化：从起点出发，可以向任意方向走
        for (int dir = 0; dir < 4; dir++) {
            visited[rowStart * m + colStart][dir] = 0;
            queue.offer(new State(rowStart, colStart, dir, 0));
        }

        while (!queue.isEmpty()) {
            State cur = queue.poll();
            int x = cur.x, y = cur.y, dir = cur.dir, usedTime = cur.usedTime;

```

```
            // 尝试从当前方向走向 4 个新方向
            for (int ndir = 0; ndir < 4; ndir++) {
                int nx = x + dx[ndir];
                int ny = y + dy[ndir];

                // 越界跳过
                if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;

                int currentUsedTime = usedTime;

                // 判断是否是右转（顺时针下一个方向），否则需要等待灯
                if ((ndir - dir + 4) % 4 != 1) {
                    currentUsedTime += grid[x][y];
                }

                // 无论如何都需要加上通行时间
                currentUsedTime += timePerRoad;

                // 如果之前到该点该方向所需时间更少，则跳过
                if (visited[nx * m + ny][ndir] <= currentUsedTime) contin
ue;

                // 更新时间并入队
                visited[nx * m + ny][ndir] = currentUsedTime;
                queue.offer(new State(nx, ny, ndir, currentUsedTime));
            }
        }

        // 从 4 个方向中取最短时间到达终点
        int res = Integer.MAX_VALUE;
        for (int i = 0; i < 4; i++) {
            res = Math.min(res, visited[rowEnd * m + colEnd][i]);
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // 读取行列
        n = sc.nextInt();
        m = sc.nextInt();

        // 读取地图代价
        int[][] grid = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = sc.nextInt();
```

```
                }
            }

            // 读取路程通行时间
            timePerRoad = sc.nextInt();

            // 起点终点坐标
            int rowStart = sc.nextInt();
            int colStart = sc.nextInt();
            int rowEnd = sc.nextInt();
            int colEnd = sc.nextInt();

            // 执行 BFS 并输出结果
            int result = BFS(rowStart, colStart, rowEnd, colEnd, grid);
            System.out.println(result);
        }
    }
```

Python

Plain Text

```
from collections import deque
import sys


n, m = 0, 0

time_per_road = 0


dx = [-1, 0, 1, 0]
dy = [0, 1, 0, -1]

def bfs(row_start, col_start, row_end, col_end, grid):

    visited = [[float('inf')] * 4 for _ in range(n * m)]

    q = deque()

    for dir in range(4):
        visited[row_start * m + col_start][dir] = 0
        q.append((row_start, col_start, dir, 0))

    while q:
        x, y, dir, used_time = q.popleft()

        for ndir in range(4):
            nx, ny = x + dx[ndir], y + dy[ndir]
            if nx < 0 or nx >= n or ny < 0 or ny >= m:
                continue

            current_time = used_time


            if (ndir - dir + 4) % 4 != 1:
                current_time += grid[x][y]


            current_time += time_per_road

            if visited[nx * m + ny][ndir] <= current_time:
                continue

            visited[nx * m + ny][ndir] = current_time
            q.append((nx, ny, ndir, current_time))

```

```
    res = float('inf')
    for i in range(4):
        res = min(res, visited[row_end * m + col_end][i])
    return res

if __name__ == '__main__':
    n, m = map(int, sys.stdin.readline().split())

    grid = [list(map(int, sys.stdin.readline().split())) for _ in range
(n)]

    time_per_road = int(sys.stdin.readline())
    row_start, col_start = map(int, sys.stdin.readline().split())
    row_end, col_end = map(int, sys.stdin.readline().split())

    result = bfs(row_start, col_start, row_end, col_end, grid)
    print(result)
```

JavaScript

Plain Text

```
const readline = require('readline');


const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', line => {
    inputLines.push(line);
}).on('close', () => {
    main();
});

function main() {
    const [n, m] = inputLines[0].split(' ').map(Number);
    const grid = [];


    for (let i = 0; i < n; i++) {
        grid.push(inputLines[1 + i].split(' ').map(Number));
    }

    const timePerRoad = parseInt(inputLines[n + 1]);
    const [rowStart, colStart] = inputLines[n + 2].split(' ').map(Number);
    const [rowEnd, colEnd] = inputLines[n + 3].split(' ').map(Number);

    const result = bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colE
nd, grid);
    console.log(result);
}


const dx = [-1, 0, 1, 0];
const dy = [0, 1, 0, -1];


function bfs(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, grid)
{
    const visited = Array(n * m).fill(0).map(() => Array(4).fill(Infinit
y));
    const queue = [];


```

```
    for (let dir = 0; dir < 4; dir++) {
        visited[rowStart * m + colStart][dir] = 0;
        queue.push({ x: rowStart, y: colStart, dir, usedTime: 0 });
    }

    while (queue.length > 0) {
        const { x, y, dir, usedTime } = queue.shift();

        for (let ndir = 0; ndir < 4; ndir++) {
            const nx = x + dx[ndir];
            const ny = y + dy[ndir];


            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;

            let currentTime = usedTime;


            if ((ndir - dir + 4) % 4 !== 1) {
                currentTime += grid[x][y];
            }


            currentTime += timePerRoad;

            if (visited[nx * m + ny][ndir] <= currentTime) continue;

            visited[nx * m + ny][ndir] = currentTime;
            queue.push({ x: nx, y: ny, dir: ndir, usedTime: currentTime
});
        }
    }


    let res = Infinity;
    for (let i = 0; i < 4; i++) {
        res = Math.min(res, visited[rowEnd * m + colEnd][i]);
    }
    return res;
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "math"
  "os"
  "strconv"
  "strings"
)


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


type State struct {
  x, y, dir, usedTime int
}

func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}


func BFS(n, m int, timePerRoad int, rowStart, colStart, rowEnd, colEnd in
t, grid [][]int) int {

  visited := make([][]int, n*m)
  for i := 0; i < n*m; i++ {
    visited[i] = make([]int, 4)
    for j := 0; j < 4; j++ {
      visited[i][j] = math.MaxInt32
    }
  }

  queue := []State{}


  for dir := 0; dir < 4; dir++ {
    visited[rowStart*m+colStart][dir] = 0
    queue = append(queue, State{rowStart, colStart, dir, 0})
```

```
  }

  for len(queue) > 0 {
    cur := queue[0]
    queue = queue[1:]

    for ndir := 0; ndir < 4; ndir++ {
      nx := cur.x + dx[ndir]
      ny := cur.y + dy[ndir]


      if nx < 0 || nx >= n || ny < 0 || ny >= m {
        continue
      }

      currentTime := cur.usedTime


      if (ndir-cur.dir+4)%4 != 1 {
        currentTime += grid[cur.x][cur.y]
      }


      currentTime += timePerRoad

      if visited[nx*m+ny][ndir] <= currentTime {
        continue
      }

      visited[nx*m+ny][ndir] = currentTime
      queue = append(queue, State{nx, ny, ndir, currentTime})
    }
  }


  res := math.MaxInt32
  for i := 0; i < 4; i++ {
    res = min(res, visited[rowEnd*m+colEnd][i])
  }

  return res
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  dims := strings.Fields(scanner.Text())
  n, _ := strconv.Atoi(dims[0])
```

```
    m, _ := strconv.Atoi(dims[1])

    grid := make([][]int, n)
    for i := 0; i < n; i++ {
      scanner.Scan()
      rowStr := strings.Fields(scanner.Text())
      row := make([]int, m)
      for j := 0; j < m; j++ {
        row[j], _ = strconv.Atoi(rowStr[j])
      }
      grid[i] = row
    }

    scanner.Scan()
    timePerRoad, _ := strconv.Atoi(scanner.Text())

    scanner.Scan()
    start := strings.Fields(scanner.Text())
    rowStart, _ := strconv.Atoi(start[0])
    colStart, _ := strconv.Atoi(start[1])

    scanner.Scan()
    end := strings.Fields(scanner.Text())
    rowEnd, _ := strconv.Atoi(end[0])
    colEnd, _ := strconv.Atoi(end[1])

    result := BFS(n, m, timePerRoad, rowStart, colStart, rowEnd, colEnd, gr
id)
    fmt.Println(result)
}
```

> 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<=T<=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<=N<=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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<set>
using namespace std;


// 递归回溯枚举三条边  并使用剪枝算法
int backtrack(vector<int>& segments, vector<bool>& used, set<int>& sideLen
gthSet) {
    int res = 0;
    int n = segments.size();
    for (int i = 0; i < n -2; i++) {
        // 已使用
        if (used[i]) {
            continue;
        }
        for (int j = i+1; j < n - 1; j++) {
            // 已使用
            if (used[j]) {
                continue;
            }
            // 第三条边长度平方
            int c = segments[i] + segments[j];
            // 不存在指定边 剪枝
            if (sideLengthSet.find(c) == sideLengthSet.end()) {
                continue;
            }
            for (int k = j+1; k < n; k++) {
                if (used[k]) {
                    continue;
                }
                // 剪枝 segments递增的，后续不可能等于c的边
                if (segments[k] > c) {
                    break;
                }
                if (segments[k] != c) {
                    continue;
                }
                // 递归回溯
                used[i] = used[j] = used[k] = true;

```

```
                    res = max(res, backtrack(segments, used, sideLengthSet) +
1);
                    used[i] = used[j] = used[k] = false;
                }
            }
        }
        return res;
}


int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;

        // 存储输入边的平方 因为判断直接公式 a^2 + b ^2 = c^2
        vector<int> side(n);
        // 记录出现的边长数 用于后续剪枝
        set<int> sideLengthSet;
        // 接收输入
        for (int i = 0; i < n; i++) {
            int tmp;
            cin >> tmp;
            tmp *= tmp;
            side[i] = tmp;
            sideLengthSet.insert(tmp);
        }
        // 排序
        sort(side.begin(), side.end());
        vector<bool> used(n, false);
        int res = backtrack(side, used, sideLengthSet);
        cout << res << endl;
    }
    return 0;
```

JAVA

Plain Text

```
import java.util.*;

public class Main {

    // 递归回溯枚举三条边，并使用剪枝算法
    public static int backtrack(List<Integer> segments, boolean[] used, Set<Integer> sideLengthSet) {
        int res = 0;
        int n = segments.size();
        for (int i = 0; i < n - 2; i++) {
            // 已使用
            if (used[i]) continue;
            for (int j = i + 1; j < n - 1; j++) {
                // 已使用
                if (used[j]) continue;

                // 第三条边长度平方
                int c = segments.get(i) + segments.get(j);

                // 不存在指定边，剪枝
                if (!sideLengthSet.contains(c)) continue;

                for (int k = j + 1; k < n; k++) {
                    if (used[k]) continue;

                    // 剪枝：segments 递增，后续不可能等于 c 的边
                    if (segments.get(k) > c) break;
                    if (!segments.get(k).equals(c)) continue;

                    // 递归回溯
                    used[i] = used[j] = used[k] = true;
                    res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
                    used[i] = used[j] = used[k] = false;
                }
            }
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int t = sc.nextInt(); // 读取测试组数

```

```
        while (t-- > 0) {
            int n = sc.nextInt();

            // 存储输入边的平方，因为判断直接公式 a^2 + b^2 = c^2
            List<Integer> sides = new ArrayList<>();

            // 记录出现的边长数，用于后续剪枝
            Set<Integer> sideLengthSet = new HashSet<>();

            // 接收输入
            for (int i = 0; i < n; i++) {
                int tmp = sc.nextInt();
                tmp *= tmp;
                sides.add(tmp);
                sideLengthSet.add(tmp);
            }

            // 排序
            Collections.sort(sides);

            // 标记每条边是否被使用
            boolean[] used = new boolean[n];

            int res = backtrack(sides, used, sideLengthSet);
            System.out.println(res);
        }
    }
}
```

Python

Plain Text

```
def backtrack(segments, used, side_length_set):
    """
    递归回溯枚举三条边，并使用剪枝算法
    """
    res = 0
    n = len(segments)
    for i in range(n - 2):
        if used[i]:
            continue
        for j in range(i + 1, n - 1):
            if used[j]:
                continue


            c = segments[i] + segments[j]


            if c not in side_length_set:
                continue

            for k in range(j + 1, n):
                if used[k]:
                    continue


                if segments[k] > c:
                    break
                if segments[k] != c:
                    continue


                used[i] = used[j] = used[k] = True
                res = max(res, backtrack(segments, used, side_length_set)
+ 1)
                used[i] = used[j] = used[k] = False
    return res


if __name__ == "__main__":
    t = int(input())
    for _ in range(t):
        arr = list(map(int, input().split()))
        n = arr[0]
        raw_lengths = arr[1:]

```

```
        side = [x * x for x in raw_lengths]


        side_length_set = set(side)


        side.sort()


        used = [False] * n

        res = backtrack(side, used, side_length_set)
        print(res)
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

let inputLines = [];
rl.on("line", (line) => {
  inputLines.push(line.trim());
});

rl.on("close", () => {
  let t = parseInt(inputLines[0]);
  let idx = 1;

  for (let test = 0; test < t; test++) {
    const arr = inputLines[idx++].split(" ").map(Number);
    const n = arr[0];
    const rawLengths = arr.slice(1);


    const side = rawLengths.map(x => x * x);


    const sideLengthSet = new Set(side);


    side.sort((a, b) => a - b);


    const used = new Array(n).fill(false);

    const res = backtrack(side, used, sideLengthSet);
    console.log(res);
  }
});


function backtrack(segments, used, sideLengthSet) {
  let res = 0;
  const n = segments.length;

  for (let i = 0; i < n - 2; i++) {
    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

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
  "strings"
)


func backtrack(segments []int, used []bool, sideLengthSet map[int]bool) in
t {
  res := 0
  n := len(segments)

  for i := 0; i < n-2; i++ {
    if used[i] {
      continue
    }
    for j := i + 1; j < n-1; j++ {
      if used[j] {
        continue
      }
      c := segments[i] + segments[j]


      if !sideLengthSet[c] {
        continue
      }
      for k := j + 1; k < n; k++ {
        if used[k] {
          continue
        }

        if segments[k] > c {
          break
        }
        if segments[k] != c {
          continue
        }

        used[i], used[j], used[k] = true, true, true
        res = max(res, backtrack(segments, used, sideLengthSet)+1)
```

```
          used[i], used[j], used[k] = false, false, false
        }
      }
    }
    return res
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    T, _ := strconv.Atoi(scanner.Text())

    for t := 0; t < T; t++ {
        scanner.Scan()
        parts := strings.Fields(scanner.Text())
        n, _ := strconv.Atoi(parts[0])

        side := make([]int, n)
        sideLengthSet := make(map[int]bool)

        for i := 0; i < n; i++ {
            val, _ := strconv.Atoi(parts[i+1])
            val *= val
            side[i] = val
            sideLengthSet[val] = true
        }


        sort.Ints(side)
        used := make([]bool, n)
        res := backtrack(side, used, sideLengthSet)
        fmt.Println(res)
    }
}

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}
```

来自: 华为OD机试 2025 B卷 – 最多几个直角三角形 (C++ & Python & JAVA & JS & GO)-CSDN博客

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<=T<=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<=N<=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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<set>
using namespace std;


// 递归回溯枚举三条边  并使用剪枝算法
int backtrack(vector<int>& segments, vector<bool>& used, set<int>& sideLengthSet) {
    int res = 0;
    int n = segments.size();
    for (int i = 0; i < n -2; i++) {
        // 已使用
        if (used[i]) {
            continue;
        }
        for (int j = i+1; j < n - 1; j++) {
            // 已使用
            if (used[j]) {
                continue;
            }
            // 第三条边长度平方
            int c = segments[i] + segments[j];
            // 不存在指定边 剪枝
            if (sideLengthSet.find(c) == sideLengthSet.end()) {
                continue;
            }
            for (int k = j+1; k < n; k++) {
                if (used[k]) {
                    continue;
                }
                // 剪枝 segments递增的，后续不可能等于c的边
                if (segments[k] > c) {
                    break;
                }
                if (segments[k] != c) {
                    continue;
                }
                // 递归回溯
                used[i] = used[j] = used[k] = true;

```

```
                    res = max(res, backtrack(segments, used, sideLengthSet) +
1);
                    used[i] = used[j] = used[k] = false;
                }
            }
        }
        return res;
}


int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;

        // 存储输入边的平方 因为判断直接公式 a^2 + b ^2 = c^2
        vector<int> side(n);
        // 记录出现的边长数 用于后续剪枝
        set<int> sideLengthSet;
        // 接收输入
        for (int i = 0; i < n; i++) {
            int tmp;
            cin >> tmp;
            tmp *= tmp;
            side[i] = tmp;
            sideLengthSet.insert(tmp);
        }
        // 排序
        sort(side.begin(), side.end());
        vector<bool> used(n, false);
        int res = backtrack(side, used, sideLengthSet);
        cout << res << endl;
    }
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {

    // 递归回溯枚举三条边，并使用剪枝算法
    public static int backtrack(List<Integer> segments, boolean[] used, Set<Integer> sideLengthSet) {
        int res = 0;
        int n = segments.size();
        for (int i = 0; i < n - 2; i++) {
            // 已使用
            if (used[i]) continue;
            for (int j = i + 1; j < n - 1; j++) {
                // 已使用
                if (used[j]) continue;

                // 第三条边长度平方
                int c = segments.get(i) + segments.get(j);

                // 不存在指定边，剪枝
                if (!sideLengthSet.contains(c)) continue;

                for (int k = j + 1; k < n; k++) {
                    if (used[k]) continue;

                    // 剪枝：segments 递增，后续不可能等于 c 的边
                    if (segments.get(k) > c) break;
                    if (!segments.get(k).equals(c)) continue;

                    // 递归回溯
                    used[i] = used[j] = used[k] = true;
                    res = Math.max(res, backtrack(segments, used, sideLengthSet) + 1);
                    used[i] = used[j] = used[k] = false;
                }
            }
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int t = sc.nextInt(); // 读取测试组数

```

```
        while (t-- > 0) {
            int n = sc.nextInt();

            // 存储输入边的平方，因为判断直接公式 a^2 + b^2 = c^2
            List<Integer> sides = new ArrayList<>();

            // 记录出现的边长数，用于后续剪枝
            Set<Integer> sideLengthSet = new HashSet<>();

            // 接收输入
            for (int i = 0; i < n; i++) {
                int tmp = sc.nextInt();
                tmp *= tmp;
                sides.add(tmp);
                sideLengthSet.add(tmp);
            }

            // 排序
            Collections.sort(sides);

            // 标记每条边是否被使用
            boolean[] used = new boolean[n];

            int res = backtrack(sides, used, sideLengthSet);
            System.out.println(res);
        }
    }
}
```

Python

Plain Text

```
def backtrack(segments, used, side_length_set):
    """
    递归回溯枚举三条边，并使用剪枝算法
    """
    res = 0
    n = len(segments)
    for i in range(n - 2):
        if used[i]:
            continue
        for j in range(i + 1, n - 1):
            if used[j]:
                continue


            c = segments[i] + segments[j]


            if c not in side_length_set:
                continue

            for k in range(j + 1, n):
                if used[k]:
                    continue


                if segments[k] > c:
                    break
                if segments[k] != c:
                    continue


                used[i] = used[j] = used[k] = True
                res = max(res, backtrack(segments, used, side_length_set)
+ 1)
                used[i] = used[j] = used[k] = False
    return res


if __name__ == "__main__":
    t = int(input())
    for _ in range(t):
        arr = list(map(int, input().split()))
        n = arr[0]
        raw_lengths = arr[1:]

```

```
        side = [x * x for x in raw_lengths]


        side_length_set = set(side)


        side.sort()


        used = [False] * n

        res = backtrack(side, used, side_length_set)
        print(res)
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

let inputLines = [];
rl.on("line", (line) => {
  inputLines.push(line.trim());
});

rl.on("close", () => {
  let t = parseInt(inputLines[0]);
  let idx = 1;

  for (let test = 0; test < t; test++) {
    const arr = inputLines[idx++].split(" ").map(Number);
    const n = arr[0];
    const rawLengths = arr.slice(1);


    const side = rawLengths.map(x => x * x);


    const sideLengthSet = new Set(side);


    side.sort((a, b) => a - b);


    const used = new Array(n).fill(false);

    const res = backtrack(side, used, sideLengthSet);
    console.log(res);
  }
});


function backtrack(segments, used, sideLengthSet) {
  let res = 0;
  const n = segments.length;

  for (let i = 0; i < n - 2; i++) {
    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

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
  "strings"
)


func backtrack(segments []int, used []bool, sideLengthSet map[int]bool) in
t {
  res := 0
  n := len(segments)

  for i := 0; i < n-2; i++ {
    if used[i] {
      continue
    }
    for j := i + 1; j < n-1; j++ {
      if used[j] {
        continue
      }
      c := segments[i] + segments[j]


      if !sideLengthSet[c] {
        continue
      }
      for k := j + 1; k < n; k++ {
        if used[k] {
          continue
        }

        if segments[k] > c {
          break
        }
        if segments[k] != c {
          continue
        }

        used[i], used[j], used[k] = true, true, true
        res = max(res, backtrack(segments, used, sideLengthSet)+1)
```

```
          used[i], used[j], used[k] = false, false, false
        }
      }
    }
    return res
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    T, _ := strconv.Atoi(scanner.Text())

    for t := 0; t < T; t++ {
        scanner.Scan()
        parts := strings.Fields(scanner.Text())
        n, _ := strconv.Atoi(parts[0])

        side := make([]int, n)
        sideLengthSet := make(map[int]bool)

        for i := 0; i < n; i++ {
            val, _ := strconv.Atoi(parts[i+1])
            val *= val
            side[i] = val
            sideLengthSet[val] = true
        }


        sort.Ints(side)
        used := make([]bool, n)
        res := backtrack(side, used, sideLengthSet)
        fmt.Println(res)
    }
}

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}
```

来自: 华为OD机试 2025 B卷 – 最多几个直角三角形 (C++ & Python & JAVA & JS & GO)-CSDN博客

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 `)` m,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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
#include<stack>
#include<map>
using namespace std;

int main() {
    int res = 0;
    string input;
    getline(cin, input);
    // cout << input << endl;
    // 栈
    stack<char> st;
    for (int i = 0; i < input.size(); i++) {
        char c = input[i];
        // 遇到非括号字符直接跳过
        if (c != '(' && c != ')') {
            continue;
        }
        // 左括号压栈
        if (c == '(' )  {
            st.push(c);
        } else {
            // 不能正常匹配
            if (st.empty()) {
                cout  << -1;
                return 0;
            }
            // 弹出
            st.pop();
            res++;
        }
    }
    // 左右数列不相同  不能正确匹配
    if (!st.empty()) {
        cout << -1;
        return 0;
    }
    cout << res;
```

```
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        int res = 0;
        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);
            // 遇到非括号字符直接跳过
            if (c != '(' && c != ')') continue;

            // 左括号压栈
            if (c == '(') {
                stack.push(c);
            } else {
                // 不能正常匹配
                if (stack.isEmpty()) {
                    System.out.println(-1);
                    return;
                }
                // 弹出
                stack.pop();
                res++;
            }
        }

        // 左右数量不相同，不能正确匹配
        if (!stack.isEmpty()) {
            System.out.println(-1);
        } else {
            System.out.println(res);
        }
    }
}
```

Python

```
Plain Text
```

```
def main():
    import sys
    input_str = sys.stdin.readline().strip()
    stack = []
    res = 0

    for c in input_str:

        if c not in '()':
            continue

        if c == '(':
            stack.append(c)
        else:

            if not stack:
                print(-1)
                return

            stack.pop()
            res += 1


    if stack:
        print(-1)
    else:
        print(res)

if __name__ == "__main__":
    main()
```

JavaScript

Plain Text

```
const readline = require('readline');
const rl = readline.createInterface({ input: process.stdin });

rl.on('line', function (input) {
    const stack = [];
    let res = 0;

    for (let i = 0; i < input.length; i++) {
        const c = input[i];

        if (c !== '(' && c !== ')') continue;

        if (c === '(') {
            stack.push(c);
        } else {

            if (stack.length === 0) {
                console.log(-1);
                return;
            }
            stack.pop();
            res++;
        }
    }

    if (stack.length !== 0) {
        console.log(-1);
    } else {
        console.log(res);
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strings"
)

func main() {
  reader := bufio.NewReader(os.Stdin)
  line, _ := reader.ReadString('\n')
  line = strings.TrimSpace(line)

  stack := []rune{}
  res := 0

  for _, c := range line {

    if c != '(' && c != ')' {
      continue
    }

    if c == '(' {
      stack = append(stack, c)
    } else {

      if len(stack) == 0 {
        fmt.Println(-1)
        return
      }

      stack = stack[:len(stack)-1]
      res++
    }
  }


  if len(stack) != 0 {
    fmt.Println(-1)
  } else {
    fmt.Println(res)
  }
}
```

> 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≤字符串长度≤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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
#include<stack>
#include<map>
using namespace std;

int main() {
    int res = 0;
    string input;
    getline(cin, input);
    // 定义括号对应关系, 减少后续进行if判断
    map<char, char> mp = {
{')', '('}, {']', '['}, {'}', '{'}};

    // 栈
    stack<char> st;
    for (int i = 0; i < input.size(); i++) {
        char c = input[i];

        // 左括号压栈
        if (c == '(' || c == '{' || c == '[')  {
            st.push(c);
            res = max(res, (int)st.size());
        } else {
            // 无效字符串
            if (st.empty()) {
                cout  << 0;
                return 0;
            }
            char top = st.top();
            // 无效字符串
            if (mp[c] != top) {
                cout << 0;
                return 0;
            }
            st.pop();
        }
    }
    // 左右不相同
```

```
    if (!st.empty()) {
        cout << 0;
        return 0;
    }
    cout << res;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        int res = 0;

        // 括号映射
        Map<Character, Character> mp = new HashMap<>();
        mp.put(')', '(');
        mp.put(']', '[');
        mp.put('}', '{');

        Stack<Character> stack = new Stack<>();

        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);

            // 左括号压栈
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
                res = Math.max(res, stack.size());
            } else {
                // 无效字符串：栈空 或者 括号不匹配
                if (stack.isEmpty() || stack.peek() != mp.get(c)) {
                    System.out.println(0);
                    return;
                }
                stack.pop();
            }
        }

        // 栈未清空，左右数量不等
        if (!stack.isEmpty()) {
            System.out.println(0);
        } else {
            System.out.println(res);
        }
    }
}
```

Python

Plain Text

```
def main():
    import sys
    input_line = sys.stdin.readline().strip()
    stack = []
    res = 0


    mp = {')': '(', ']': '[', '}': '{'}

    for c in input_line:

        if c in '([{':
            stack.append(c)
            res = max(res, len(stack))
        else:

            if not stack or stack[-1] != mp.get(c):
                print(0)
                return
            stack.pop()


    if stack:
        print(0)
    else:
        print(res)

if __name__ == '__main__':
    main()
```

JavaScript

Plain Text

```
const readline = require('readline');
const rl = readline.createInterface({ input: process.stdin });

rl.on('line', function(line) {
    const input = line.trim();
    const stack = [];
    let res = 0;

    const mp = { ')': '(', ']': '[', '}': '{' };

    for (let c of input) {

        if (c === '(' || c === '[' || c === '{') {
            stack.push(c);
            res = Math.max(res, stack.length);
        } else {

            if (stack.length === 0 || stack[stack.length - 1] !== mp[c]) {
                console.log(0);
                return;
            }
            stack.pop();
        }
    }

    if (stack.length !== 0) {
        console.log(0);
    } else {
        console.log(res);
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strings"
)

func main() {
  reader := bufio.NewReader(os.Stdin)
  input, _ := reader.ReadString('\n')
  input = strings.TrimSpace(input)

  stack := []rune{}
  res := 0


  mp := map[rune]rune{')': '(', ']': '[', '}': '{'}

  for _, c := range input {
    if c == '(' || c == '[' || c == '{' {
      stack = append(stack, c)
      if len(stack) > res {
        res = len(stack)
      }
    } else {

      if len(stack) == 0 || stack[len(stack)-1] != mp[c] {
        fmt.Println(0)
        return
      }
      stack = stack[:len(stack)-1]
    }
  }

  if len(stack) != 0 {
    fmt.Println(0)
  } else {
    fmt.Println(res)
  }
}
```

> 来自: 华为OD机试 2025 B卷 – 最大括号深度 (C++ & Python & JAVA & JS & GO)-CSDN博客