

【华为OD机考 统一考试机试C卷】最多几个直角三角形 (C++ Java JavaScript Python C语言)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录 (C卷 + D卷 + B卷 + A卷) + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

题目描述

有N条线段，长度分别为 $a[1]-a[n]$ 。

现要求你计算这N条线段最多可以组合成几个直角三角形。

每条线段只能使用一次，每个三角形包含三条线段。

输入描述

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

对于每组测试数据，接下来有T行，

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

输出描述

对于每组测试数据输出一行，每行包括一个整数，表示最多能组合的直角三角形个数

用例

输入

```
1 | 1
2 | 7 3 4 5 6 5 12 13
```

输出

```
1 | 2
```

说明

可以组成2个直角三角形 (3, 4, 5) 、 (5, 12, 13)

C++ 递归

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  // dfs函数用于查找最多可以组成的直角三角形数量
6  int dfs(const std::vector<int>& arr, std::vector<bool>& used, int index, int count) {
7      // 初始化答案为当前已找到的直角三角形数量
8      int ans = count;
9
10     // 遍历线段数组, 从index开始
11     for (int i = index; i < arr.size(); i++) {
12         // 如果当前线段已被使用, 跳过
13         if (used[i]) continue;
14         // 遍历后续线段, 从i+1开始
15         for (int j = i + 1; j < arr.size(); j++) {
16             // 如果当前线段已被使用, 跳过
17             if (used[j]) continue;
18             // 遍历后续线段, 从j+1开始
19             for (int k = j + 1; k < arr.size(); k++) {
20                 // 如果当前线段已被使用, 跳过
21                 if (used[k]) continue;
22
23                 // 检查当前三条线段是否满足勾股定理
24                 if (arr[i] * arr[i] + arr[j] * arr[j] == arr[k] * arr[k]) {
25
```

```
26         // 标记当前线段为已使用
27         used[i] = true;
28         used[j] = true;
29         used[k] = true;
30
31         // 递归调用dfs函数, 并更新答案
32         ans = std::max(ans, dfs(arr, used, i + 1, count + 1));
33
34         // 回溯, 将当前线段标记为未使用
35         used[i] = false;
36         used[j] = false;
37         used[k] = false;
38     }
39 }
40 }
41 }
42
43 // 返回答案
44 return ans;
45 }
46
47 int main() {
48     // 读取测试数据组数
49     int t;
50     std::cin >> t;
51     // 创建一个二维数组来存储每组测试数据
52     std::vector<std::vector<int>> cases(t);
53
54     // 读取每组测试数据
55     for (int i = 0; i < t; i++) {
56         // 读取线段数量
57         int n;
58         std::cin >> n;
59         // 创建一个数组来存储线段长度
60         std::vector<int> arr(n);
61         // 读取每条线段的长度
62         for (int j = 0; j < n; j++) {
63             std::cin >> arr[j];
64         }
65         // 将线段长度数组存入cases数组
66     }
```

```
67     cases[i] = arr;
68 }
69
70 // 遍历每组测试数据
71 for (const auto& arr : cases) {
72     // 对线段长度进行升序排序
73     std::vector<int> sorted_arr = arr;
74     std::sort(sorted_arr.begin(), sorted_arr.end());
75     // 创建一个布尔数组表示线段是否已使用
76     std::vector<bool> used(sorted_arr.size(), false);
77     // 调用dfs函数并输出结果
78     std::cout << dfs(sorted_arr, used, 0, 0) << std::endl;
79 }
80
81 return 0;
82 }
```

C++ - 栈

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <stack>
5
6  // State结构体用于存储每个状态的信息，包括当前索引、已找到的直角三角形数量和已使用的线段
7  struct State {
8      int index;
9      int count;
10     std::vector<bool> used;
11
12     State(int index, int count, const std::vector<bool>& used)
13         : index(index), count(count), used(used) {}
14 };
15
16 int non_recursive_dfs(const std::vector<int>& lengths);
17
18 int main() {
19     // 读取测试数据组数
20 }
```

```
20     int test_cases;
21     std::cin >> test_cases;
22
23     // 创建一个二维数组来存储每组测试数据
24     std::vector<std::vector<int>> input_data(test_cases);
25
26     // 读取每组测试数据
27     for (int i = 0; i < test_cases; ++i) {
28         // 读取线段数量
29         int segments;
30         std::cin >> segments;
31
32         // 创建一个数组来存储线段长度
33         std::vector<int> lengths(segments);
34
35         // 读取每条线段的长度
36         for (int j = 0; j < segments; ++j) {
37             std::cin >> lengths[j];
38         }
39
40         // 将线段长度数组存入input_data数组
41         input_data[i] = lengths;
42     }
43
44     for (auto& lengths : input_data) {
45         // 对线段长度进行升序排序
46         std::sort(lengths.begin(), lengths.end());
47
48         // 调用non_recursive_dfs函数并输出结果
49         std::cout << non_recursive_dfs(lengths) << std::endl;
50     }
51
52     return 0;
53 }
54
55 int non_recursive_dfs(const std::vector<int>& lengths) {
56     int max_triangles = 0;
57     std::vector<bool> used_segments(lengths.size(), false);
58     std::stack<State> stack;
59     stack.push(State(0, 0, used_segments));
60 }
```

```
61
62 // 遍历状态栈
63 while (!stack.empty()) {
64     State current_state = stack.top();
65     stack.pop();
66     int current_index = current_state.index;
67     int current_count = current_state.count;
68     used_segments = current_state.used;
69
70     max_triangles = std::max(max_triangles, current_count);
71
72     // 遍历线段数组, 从current_index开始
73     for (int i = current_index; i < lengths.size(); ++i) {
74         if (used_segments[i]) continue;
75         for (int j = i + 1; j < lengths.size(); ++j) {
76             if (used_segments[j]) continue;
77             for (int k = j + 1; k < lengths.size(); ++k) {
78                 if (used_segments[k]) continue;
79
80                 // 检查当前三条线段是否满足勾股定理
81                 if (lengths[i] * lengths[i] + lengths[j] * lengths[j] == lengths[k] * lengths[k]) {
82                     std::vector<bool> new_used_segments = used_segments;
83                     new_used_segments[i] = true;
84                     new_used_segments[j] = true;
85                     new_used_segments[k] = true;
86                     stack.push(State(i + 1, current_count + 1, new_used_segments));
87                 }
88             }
89         }
90     }
91 }
92
93 return max_triangles;
94 }
```

javascript 递归

```
1 const readline = require('readline');
2 const rl = readline.createInterface({
3     input: process.stdin,
```

```
3     input: process.stdin,
4     output: process.stdout
5 });
6 // 查找最多可以组成的直角三角形数量
7 function dfs(arr, used, index, count) {
8     let ans = count;
9
10    for (let i = index; i < arr.length; i++) {
11        if (used[i]) {
12            continue;
13        }
14        for (let j = i + 1; j < arr.length; j++) {
15            if (used[j]) {
16                continue;
17            }
18            for (let k = j + 1; k < arr.length; k++) {
19                if (used[k]) {
20                    continue;
21                }
22
23                if (arr[i] ** 2 + arr[j] ** 2 === arr[k] ** 2) {
24                    used[i] = true;
25                    used[j] = true;
26                    used[k] = true;
27
28                    ans = Math.max(ans, dfs(arr, used, i + 1, count + 1));
29
30                    used[i] = false;
31                    used[j] = false;
32                    used[k] = false;
33                }
34            }
35        }
36    }
37
38    return ans;
39 }
40 // 读取测试数据组数
41 rl.on('line', (test_cases) => {
42     test_cases = parseInt(test_cases);
43     ..
```



```
44     const input_data = [];  
45     let read_count = 0;  
46  
47     // 读取每组测试数据  
48     rl.on('line', (line) => {  
49         input_data.push(line.split(' ').slice(1).map(Number));  
50         read_count++;  
51  
52         if (read_count === test_cases) {  
53             rl.close();  
54  
55             for (const testCase of input_data) {  
56                 // 对线段长度进行升序排序  
57                 testCase.sort((a, b) => a - b);  
58  
59                 // 调用non_recursive_dfs函数并输出结果  
60                 console.log(dfs(testCase, Array(testCase.length).fill(false), 0, 0));  
61             }  
62         }  
63     });  
64 });
```

JavaScript - 栈

```
1  const readline = require('readline');  
2  const rl = readline.createInterface({  
3      input: process.stdin,  
4      output: process.stdout  
5  });  
6  
7  // State类用于存储每个状态的信息, 包括当前索引、已找到的直角三角形数量和已使用的线段  
8  class State {  
9      constructor(index, count, used) {  
10         this.index = index;  
11         this.count = count;  
12         this.used = used;  
13     }  
14 }  
15  
16
```

```
16 function combinations(arr, k) {
17   if (k > arr.length || k <= 0) {
18     return [];
19   }
20
21   if (k === arr.length) {
22     return [arr];
23   }
24
25   if (k === 1) {
26     return arr.map((value) => [value]);
27   }
28
29   const result = [];
30   for (let i = 0; i < arr.length - k + 1; i++) {
31     const head = arr.slice(i, i + 1);
32     const tailcombs = combinations(arr.slice(i + 1), k - 1);
33     for (let j = 0; j < tailcombs.length; j++) {
34       result.push(head.concat(tailcombs[j]));
35     }
36   }
37   return result;
38 }
39
40 function non_recursive_dfs(lengths) {
41   let max_triangles = 0;
42   const used_segments = Array(lengths.length).fill(false);
43   const stack = [new State(0, 0, used_segments)];
44
45   // 遍历状态栈
46   while (stack.length > 0) {
47     const current_state = stack.pop();
48     const current_index = current_state.index;
49     const current_count = current_state.count;
50     const used_segments = current_state.used;
51
52     max_triangles = Math.max(max_triangles, current_count);
53
54     // 遍历线段数组, 从current_index开始
55     for (const [i, j, k] of combinations([...Array(lengths.length - current_index).keys()].map(x => x + current_index), 3)) {
56
```

```
57     if (used_segments[i] || used_segments[j] || used_segments[k]) {
58         continue;
59     }
60
61     // 检查当前三条线段是否满足勾股定理
62     if (lengths[i] ** 2 + lengths[j] ** 2 === lengths[k] ** 2) {
63         const new_used_segments = used_segments.slice();
64         new_used_segments[i] = true;
65         new_used_segments[j] = true;
66         new_used_segments[k] = true;
67         stack.push(new State(i + 1, current_count + 1, new_used_segments));
68     }
69 }
70 }
71
72 return max_triangles;
73 }
74
75 // 读取测试数据组数
76 rl.on('line', (test_cases) => {
77     test_cases = parseInt(test_cases);
78     const input_data = [];
79     let read_count = 0;
80
81     // 读取每组测试数据
82     rl.on('line', (line) => {
83         input_data.push(line.split(' ').slice(1).map(Number));
84         read_count++;
85
86         if (read_count === test_cases) {
87             rl.close();
88
89             for (const testCase of input_data) {
90                 // 对线段长度进行升序排序
91                 testCase.sort((a, b) => a - b);
92
93                 // 调用non_recursive_dfs函数并输出结果
94                 console.log(non_recursive_dfs(testCase));
95             }
96         }
97     });
```

```
98     });  
99 });
```

java-栈解法

```
1  import java.util.Arrays;  
2  import java.util.Scanner;  
3  import java.util.Stack;  
4  
5  public class Main {  
6      public static void main(String[] args) {  
7          // 创建Scanner对象以读取用户输入  
8          Scanner scanner = new Scanner(System.in);  
9  
10         // 读取测试数据组数  
11         int testCases = scanner.nextInt();  
12         // 创建一个二维数组来存储每组测试数据  
13         int[][] inputData = new int[testCases][];  
14  
15         // 读取每组测试数据  
16         for (int i = 0; i < testCases; i++) {  
17             // 读取线段数量  
18             int segments = scanner.nextInt();  
19             // 创建一个数组来存储线段长度  
20             int[] lengths = new int[segments];  
21             // 读取每条线段的长度  
22             for (int j = 0; j < segments; j++) {  
23                 lengths[j] = scanner.nextInt();  
24             }  
25             // 将线段长度数组存入inputData数组  
26             inputData[i] = lengths;  
27         }  
28  
29         // 遍历每组测试数据  
30         for (int[] lengths : inputData) {  
31             // 对线段长度进行升序排序  
32             Arrays.sort(lengths);  
33             // 调用nonRecursiveDFS函数并输出结果  
34         }
```

```
34         System.out.println(nonRecursiveDFS(lengths));
35     }
36 }
37
38 public static int nonRecursiveDFS(int[] lengths) {
39     int maxTriangles = 0;
40     boolean[] usedSegments = new boolean[lengths.length];
41     Stack<State> stack = new Stack<>();
42     stack.push(new State(0, 0, usedSegments));
43
44     // 遍历状态栈
45     while (!stack.isEmpty()) {
46         State currentState = stack.pop();
47         int currentIndex = currentState.index;
48         int currentCount = currentState.count;
49         usedSegments = currentState.used.clone();
50
51         maxTriangles = Math.max(maxTriangles, currentCount);
52
53         // 遍历线段数组, 从currentIndex开始
54         for (int i = currentIndex; i < lengths.length; i++) {
55             if (usedSegments[i]) continue;
56             for (int j = i + 1; j < lengths.length; j++) {
57                 if (usedSegments[j]) continue;
58                 for (int k = j + 1; k < lengths.length; k++) {
59                     if (usedSegments[k]) continue;
60
61                     // 检查当前三条线段是否满足勾股定理
62                     if (lengths[i] * lengths[i] + lengths[j] * lengths[j] == lengths[k] * lengths[k]) {
63                         boolean[] newUsedSegments = usedSegments.clone();
64                         newUsedSegments[i] = true;
65                         newUsedSegments[j] = true;
66                         newUsedSegments[k] = true;
67                         stack.push(new State(i + 1, currentCount + 1, newUsedSegments));
68                     }
69                 }
70             }
71         }
72     }
73 }
74
--
```

```
75     return maxTriangles;
76 }
77
78 // State类用于存储每个状态的信息, 包括当前索引、已找到的直角三角形数量和已使用的线段
79 static class State {
80     int index;
81     int count;
82     boolean[] used;
83
84     State(int index, int count, boolean[] used) {
85         this.index = index;
86         this.count = count;
87         this.used = used;
88     }
89 }
90 }
```

Java - 递归解法

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         // 创建Scanner对象以读取用户输入
7         Scanner sc = new Scanner(System.in);
8
9         // 读取测试数据组数
10        int t = sc.nextInt();
11        // 创建一个二维数组来存储每组测试数据
12        int[][] cases = new int[t][];
13
14        // 读取每组测试数据
15        for (int i = 0; i < t; i++) {
16            // 读取线段数量
17            int n = sc.nextInt();
18            // 创建一个数组来存储线段长度
19            int[] arr = new int[n];
20            // 读取每条线段的长度
21            for (int j = 0; j < n; j++) {
22                arr[j] = sc.nextInt();
23            }
24            cases[i] = arr;
25        }
26    }
27 }
```

```
21     for (int j = 0; j < n; j++) {
22         arr[j] = sc.nextInt();
23     }
24     // 将线段长度数组存入cases数组
25     cases[i] = arr;
26 }
27
28 // 遍历每组测试数据
29 for (int[] arr : cases) {
30     // 对线段长度进行升序排序
31     Arrays.sort(arr);
32     // 调用dfs函数并输出结果
33     System.out.println(dfs(arr, new boolean[arr.length], 0, 0));
34 }
35 }
36
37 // dfs函数用于查找最多可以组成的直角三角形数量
38 public static int dfs(int[] arr, boolean[] used, int index, int count) {
39     // 初始化答案为当前已找到的直角三角形数量
40     int ans = count;
41
42     // 遍历线段数组, 从index开始
43     for (int i = index; i < arr.length; i++) {
44         // 如果当前线段已被使用, 跳过
45         if (used[i]) continue;
46         // 遍历后续线段, 从i+1开始
47         for (int j = i + 1; j < arr.length; j++) {
48             // 如果当前线段已被使用, 跳过
49             if (used[j]) continue;
50             // 遍历后续线段, 从j+1开始
51             for (int k = j + 1; k < arr.length; k++) {
52                 // 如果当前线段已被使用, 跳过
53                 if (used[k]) continue;
54
55                 // 检查当前三条线段是否满足勾股定理
56                 if (arr[i] * arr[i] + arr[j] * arr[j] == arr[k] * arr[k]) {
57                     // 标记当前线段为已使用
58                     used[i] = true;
59                     used[j] = true;
60                     used[k] = true;
```

```
62
63 // 递归调用dfs函数, 并更新答案
64 ans = Math.max(ans, dfs(arr, used, i + 1, count + 1));
65
66 // 回溯, 将当前线段标记为未使用
67 used[i] = false;
68 used[j] = false;
69 used[k] = false;
70 }
71 }
72 }
73 }
74
75 // 返回答案
76 return ans;
77 }
78 }
```

Python - 栈

```
1 from itertools import combinations
2
3 # State类用于存储每个状态的信息, 包括当前索引、已找到的直角三角形数量和已使用的线段
4 class State:
5     def __init__(self, index, count, used):
6         self.index = index
7         self.count = count
8         self.used = used
9
10 def non_recursive_dfs(lengths):
11     max_triangles = 0
12     used_segments = [False] * len(lengths)
13     stack = [State(0, 0, used_segments)]
14
15     # 遍历状态栈
16     while stack:
17         current_state = stack.pop()
18         current_index = current_state.index
19         current_count = current_state.count
```



```
20    used_segments = current_state.used
21
22    max_triangles = max(max_triangles, current_count)
23
24    # 遍历线段数组, 从current_index开始
25    for i, j, k in combinations(range(current_index, len(lengths)), 3):
26        if used_segments[i] or used_segments[j] or used_segments[k]:
27            continue
28
29        # 检查当前三条线段是否满足勾股定理
30        if lengths[i] ** 2 + lengths[j] ** 2 == lengths[k] ** 2:
31            new_used_segments = used_segments.copy()
32            new_used_segments[i] = True
33            new_used_segments[j] = True
34            new_used_segments[k] = True
35            stack.append(State(i + 1, current_count + 1, new_used_segments))
36
37    return max_triangles
38
39 # 读取测试数据组数
40 test_cases = int(input())
41
42 # 创建一个二维数组来存储每组测试数据
43 input_data = []
44
45 # 读取每组测试数据
46 for _ in range(test_cases):
47
48     input_data = [list(map(int, input().split()))[1:] for i in range(test_cases)]
49
50 for testCase in input_data:
51     # 对线段长度进行升序排序
52     testCase.sort()
53     # 调用non_recursive_dfs函数并输出结果
54     print(non_recursive_dfs(testCase))
55
```

python 递归

```
1
2 import sys
3 from typing import List, Tuple
4
5 # 查找最多可以组成的直角三角形数量
6 def dfs(arr: List[int], used: List[bool], index: int, count: int) -> int:
7     ans = count
8
9     for i in range(index, len(arr)):
10         if used[i]:
11             continue
12         for j in range(i + 1, len(arr)):
13             if used[j]:
14                 continue
15             for k in range(j + 1, len(arr)):
16                 if used[k]:
17                     continue
18
19                 if arr[i] ** 2 + arr[j] ** 2 == arr[k] ** 2:
20                     used[i] = True
21                     used[j] = True
22                     used[k] = True
23
24                     ans = max(ans, dfs(arr, used, i + 1, count + 1))
25
26                     used[i] = False
27                     used[j] = False
28                     used[k] = False
29
30     return ans
31 # 读取测试数据组数
32 test_cases = int(input())
33
34 # 创建一个二维数组来存储每组测试数据
35 input_data = []
36
37 # 读取每组测试数据
38 for _ in range(test_cases):
39
40     input_data = [list(map(int, input().split()))[1:] for i in range(test_cases)]
41
```

```
41  
42  
43 for testCase in input_data:  
44     # 对线段长度进行升序排序  
45     testCase.sort()  
46     # 调用non_recursive_dfs函数并输出结果  
47     print(dfs(testCase, [False] * len(testCase), 0, 0))
```

C语言 - 递归

```
1  #include <stdio.h>  
2  #include <stdbool.h>  
3  #include <stdlib.h>  
4  
5  #define MAX_T 100  
6  #define MAX_N 20  
7  
8  // dfs函数用于查找最多可以组成的直角三角形数量  
9  int dfs(int arr[], bool used[], int size, int index, int count) {  
10     int ans = count; // 初始化答案为当前已找到的直角三角形数量  
11  
12     // 遍历线段数组, 从index开始  
13     for (int i = index; i < size; i++) {  
14         // 如果当前线段已被使用, 跳过  
15         if (used[i]) continue;  
16         // 遍历后续线段, 从i+1开始  
17         for (int j = i + 1; j < size; j++) {  
18             // 如果当前线段已被使用, 跳过  
19             if (used[j]) continue;  
20             // 遍历后续线段, 从j+1开始  
21             for (int k = j + 1; k < size; k++) {  
22                 // 如果当前线段已被使用, 跳过  
23                 if (used[k]) continue;  
24  
25                 // 检查当前三条线段是否满足勾股定理  
26                 if (arr[i] * arr[i] + arr[j] * arr[j] == arr[k] * arr[k] ||  
27                     arr[i] * arr[i] + arr[k] * arr[k] == arr[j] * arr[j] ||  
28                     arr[j] * arr[j] + arr[k] * arr[k] == arr[i] * arr[i]) {  
29                     // 标记当前线段为已使用  
30                     used[i] = true;
```

```
31         used[j] = true;
32         used[k] = true;
33
34         // 递归调用dfs函数, 并更新答案
35         int newCount = dfs(arr, used, size, i + 1, count + 1);
36         if (newCount > ans) {
37             ans = newCount;
38         }
39
40         // 回溯, 将当前线段标记为未使用
41         used[i] = false;
42         used[j] = false;
43         used[k] = false;
44     }
45 }
46 }
47 }
48
49 // 返回答案
50 return ans;
51 }
52
53 // 用于比较两个整数的函数, 用于qsort
54 int compare(const void *a, const void *b) {
55     return (*(int *)a - *(int *)b);
56 }
57
58 int main() {
59     int t;
60     scanf("%d", &t); // 读取测试数据组数
61
62     while (t--) {
63         int n;
64         scanf("%d", &n); // 读取线段数量
65         int arr[MAX_N];
66         bool used[MAX_N] = {false}; // 创建一个布尔数组表示线段是否已使用
67
68         for (int i = 0; i < n; i++) {
69             scanf("%d", &arr[i]); // 读取每条线段的长度
70         }
71     }
72 }
```

```
72 |
73 |     // 对线段长度进行升序排序
74 |     qsort(arr, n, sizeof(int), compare);
75 |
76 |     // 调用dfs函数并输出结果
77 |     printf("%d\n", dfs(arr, used, n, 0, 0));
78 | }
79 |
80 | return 0;
    | }
```

C语言 - 栈

```
1 | #include <stdio.h>
2 | #include <stdlib.h>
3 | #include <stdbool.h>
4 |
5 | #define MAX_N 20
6 |
7 | // State结构体用于存储每个状态的信息, 包括当前索引、已找到的直角三角形数量和已使用的线段
8 | typedef struct State {
9 |     int index;
10 |    int count;
11 |    bool used[MAX_N];
12 | } State;
13 |
14 | // 用于比较两个整数的函数, 用于qsort
15 | int compare(const void *a, const void *b) {
16 |     return (*(int *)a - *(int *)b);
17 | }
18 |
19 | // non_recursive_dfs函数用于非递归地进行深度优先搜索
20 | int non_recursive_dfs(int lengths[], int size) {
21 |     int max_triangles = 0;
22 |     bool used_segments[MAX_N] = {false};
23 |     State stack[MAX_N * MAX_N * MAX_N]; // 堆栈大小取决于可能的状态数
24 |     int top = -1; // 栈顶初始化为-1, 表示栈为空
25 |
26 |     // 初始状态入栈
27 |     stack[++top] = (State){0, 0, {false}};
28 | }
```

```
28
29 // 遍历状态栈
30 while (top != -1) {
31     State current_state = stack[top--]; // 弹出栈顶元素
32     int current_index = current_state.index;
33     int current_count = current_state.count;
34
35     for (int i = 0; i < size; ++i) {
36         used_segments[i] = current_state.used[i];
37     }
38
39     if (current_count > max_triangles) {
40         max_triangles = current_count;
41     }
42
43     // 遍历线段数组, 从current_index开始
44     for (int i = current_index; i < size; ++i) {
45         if (used_segments[i]) continue;
46         for (int j = i + 1; j < size; ++j) {
47             if (used_segments[j]) continue;
48             for (int k = j + 1; k < size; ++k) {
49                 if (used_segments[k]) continue;
50
51                 // 检查当前三条线段是否满足勾股定理
52                 if (lengths[i] * lengths[i] + lengths[j] * lengths[j] == lengths[k] * lengths[k] ||
53                     lengths[i] * lengths[i] + lengths[k] * lengths[k] == lengths[j] * lengths[j] ||
54                     lengths[j] * lengths[j] + lengths[k] * lengths[k] == lengths[i] * lengths[i]) {
55                     used_segments[i] = true;
56                     used_segments[j] = true;
57                     used_segments[k] = true;
58                     stack[++top] = (State){i + 1, current_count + 1, {0}};
59                     for (int l = 0; l < size; ++l) {
60                         stack[top].used[l] = used_segments[l];
61                     }
62                     used_segments[i] = false;
63                     used_segments[j] = false;
64                     used_segments[k] = false;
65                 }
66             }
67         }
68     }
```

```
69     }
70 }
71
72 return max_triangles;
73 }
74
75 int main() {
76     int test_cases;
77     scanf("%d", &test_cases); // 读取测试数据组数
78
79     while (test_cases--) {
80         int segments;
81         scanf("%d", &segments); // 读取线段数量
82         int lengths[MAX_N];
83
84         for (int i = 0; i < segments; ++i) {
85             scanf("%d", &lengths[i]); // 读取每条线段的长度
86         }
87
88         // 对线段长度进行升序排序
89         qsort(lengths, segments, sizeof(int), compare);
90
91         // 调用non_recursive_dfs函数并输出结果
92         printf("%d\n", non_recursive_dfs(lengths, segments));
93     }
94
95     return 0;
96 }
```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例

C++ 递归

C++ - 栈

javascript 递归
JavaScript - 栈
java-栈解法
Java - 递归解法
Python - 栈
python 递归
C语言 - 递归
C语言 - 栈

机考真题 华为OD



CSDN @算法大师