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

输入	1 7 3 4 5 6 5 12 13
输出	2
说明	可以组成2个直角三角形（3，4，5）、（5，12，13）

题目解析

~~简单的全组合求解，即从n个数中选3个，并通过勾股定理验证选择的3个数是否可以组成直角三角形。~~

~~需要注意的是，用例中有两个5，理论上可以形成两组3，4，5，和两组5，12，13，即最终有四个符合要求的组合，但是这里用例只输出两个符合组合，因此需要对线段长度去重。~~

~~这里的去重可以在求解全组合之前做，即依赖于Set去重，也可以在求解全组合的过程中做，即依赖于树层去重。其中树层去重的性能更好。~~

~~另外判断直接三角形时，我们需要选择的线段升序排序，因为根据勾股定理可知，斜边必然要大于两个直角边，因此升序后，最后一个元素就是最长的线段，即斜边。~~

根据网友指正，本题并不是简单的全组合求解，比如我们看用例1:

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

如果单纯以全组合角度来求解组成直角三角形的线段的话，有如下情况：

- 3 4 5
- 3 4 5
- 5 12 13
- 5 12 13

一共四组，造成重复的原因是，存在两个5。

现在要求的是，以给的的线段，能组合出来的最多直角三角形数量。这句话的意思是，我们能用3 4 5 6 5 12 13组合出最多几个直角三角形?

比如，我们已经用了3 4 5组成一个直角三角形，那么给的线段还剩下6 5 12 13，而剩下的线段中，只能组合出一个直角三角形5 12 13。

那么该如何实现一种算法找出最多的呢?

我的解题思路如下，首先用全组合，求出所有直角三角形的组合可能：

- 3 4 5
- 3 4 5

题目解析

~~简单的全组合求解，即从n个数中选3个，并通过勾股定理验证选择的3个数是否可以组成直角三角形。~~

~~需要注意的是，用例中有两个5，理论上可以形成两组3，4，5，和两组5，12，13，即最终有四个符合要求的组合，但是这里用例只输出两个符合组合，因此需要对线段长度去重。~~

~~这里的去重可以在求解全组合之前做，即依赖于Set去重，也可以在求解全组合的过程中做，即依赖于树层去重。其中树层去重的性能更好。~~

~~另外判断直接三角形时，我们需要选择的线段升序排序，因为根据勾股定理可知，斜边必然要大于两个直角边，因此升序后，最后一个元素就是最长的线段，即斜边。~~

根据网友指正，本题并不是简单的全组合求解，比如我们看用例1：

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

如果单纯以全组合角度来求解组成直角三角形的线段的话，有如下情况：

- 3 4 5
- 3 4 5
- 5 12 13
- 5 12 13

一共四组，造成重复的原因是，存在两个5。

现在要求的是，以给的的线段，能组合出来的最多直角三角形数量。这句话的意思是，我们能用3 4 5 6 5 12 13**组合出最多几个直角三角形?**

比如，我们已经用了3 4 5组成一个直角三角形，那么给的线段还剩下6 5 12 13，而剩下的线段中，只能组合出一个直角三角形5 12 13。

那么该如何实现一种算法找出最多的呢?

我的解题思路如下，首先用全组合，求出所有直角三角形的组合可能：

- 3 4 5
- 3 4 5
- 5 12 13
- 5 12 13

然后统计出给的各种长度线段对应的数量，比如用例1可以统计如下：

```
{
  3: 1, // 长度为3的线段有1个
  4: 1,
  5: 2,
  6: 1,
  12: 1,
  13: 1
}
```

然后我们通过回溯算法，比如遍历出（前面全组合求解出来的）第一个可能的直角三角形组合3 4 5，然后上面统计数量变为：

```
{
  3: 0,
  4: 0,
  5: 1,
  6: 1,
  12: 1,
  13: 1
}
```

然后，继续遍历下一个可能直角三角形组合3 4 5，发现统计的3的数量已经为0了，因此这个三角形无法组合，继续遍历下一个可能直角三角形组合 5 12 13，然后统计数量变为

```
{
  3: 0,
  4: 0,
  5: 0,
  6: 1,
  12: 0,
  13: 0
}
```

然后继续遍历下一个可能组合5 12 13，发现对应长度线段的数量都变为了0，因此无法组合。

继续遍历，发现没有下一个可能的组合了，因此，计算出该种情况可以得到2个直角三角形组合：3 4 5以及5 12 13

```
{
  3: 0,
  4: 0,
  5: 1,
  6: 1,
  12: 1,
  13: 1
}
```

然后，继续遍历下一个可能直角三角形组合3 4 5，发现统计的3的数量已经为0了，因此这个三角形无法组合，继续遍历下一个可能直角三角形组合 5 12 13，然后统计数量变为

```
{
  3: 0,
  4: 0,
  5: 0,
  6: 1,
  12: 0,
  13: 0
}
```

然后继续遍历下一个可能组合5 12 13，发现对应长度线段的数量都变为了0，因此无法组合。

继续遍历，发现没有下一个可能的组合了，因此，计算出该种情况可以得到2个直角三角形组合：3 4 5以及5 12 13

下面通过回溯算法，开始从第二个可能的组合开始向后遍历。

最终，最多的组合数就是题解。

我们可以尝试下这个自测用例：

```
1
7 3 4 5 12 13 84 85
```

其中有三种可能得直角三角形组合，分别是：

- 3 4 5
- 5 12 13
- 13 84 85

如果我们选择组合3 4 5，则还可以组合一个13 84 85

如果我们选择组合5 12 13，则无法组合出其他直角三角形

因此，最终本用例返回2，最多可以组合出2个直角三角形，分别是3 4 5和13 84 85

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

const lines = [];
let t;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    t = lines[0] - 0;
  }

  if (t && lines.length === t + 1) {
    const cases = lines
      .slice(1)
      .map((line) => line.split(" ").map(Number).slice(1));

    getResult(cases);
    lines.length = 0;
  }
});

function getResult(cases) {
  for (let arr of cases) {
    // 对每组测试线段升序排序
    arr.sort((a, b) => a - b);

    const res = [];
    dfs(arr, 0, [], res);

    const count = new Array(100).fill(0);
    for (let i of arr) {
      count[i]++;
    }
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

const lines = [];
let t;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    t = lines[0] - 0;
  }

  if (t && lines.length === t + 1) {
    const cases = lines
      .slice(1)
      .map((line) => line.split(" ").map(Number).slice(1));

    getResult(cases);
    lines.length = 0;
  }
});

function getResult(cases) {
  for (let arr of cases) {
    // 对每组测试线段升序排序
    arr.sort((a, b) => a - b);

    const res = [];
    dfs(arr, 0, [], res);

    const count = new Array(100).fill(0);
    for (let i of arr) {
      count[i]++;
    }

    const ans = [];
    canCombine(res, 0, count, 0, ans);
    console.log(Math.max.apply(null, ans));
  }
}

// 全组合求解，即n个数中选3个
function dfs(arr, index, path, res) {
  if (path.length === 3) {
    if (isRightTriangle(path)) res.push([...path]);
    return;
  }

  for (let i = index; i < arr.length; i++) {
    path.push(arr[i]);
    dfs(arr, i + 1, path, res);
    path.pop();
  }
}

// 判断三条边是否可以组成直角三角形
function isRightTriangle(path) {
  const [x, y, z] = path;
  return x * x + y * y === z * z;
}

// 求解当前直角三角形中不超用线段的最多组合数
function canCombine(ts, index, count, num, ans) {
  if (index >= ts.length) {
    ans.push(num);
    return;
  }

  for (let i = index; i < ts.length; i++) {
    const [a, b, c] = ts[i];

    if (count[a] > 0 && count[b] > 0 && count[c] > 0) {
      count[a]--;
      count[b]--;
      count[c]--;
      num++;
      canCombine(ts, i + 1, count, num, ans);
      num--;
      count[a]++;
      count[b]++;
      count[c]++;
    }
  }

  ans.push(num);
}
```

```
  }

  for (let i = index; i < arr.length; i++) {
    path.push(arr[i]);
    dfs(arr, i + 1, path, res);
    path.pop();
  }
}

// 判断三条边是否可以组成直角三角形
function isRightTriangle(path) {
  const [x, y, z] = path;
  return x * x + y * y === z * z;
}

// 求解当前直角三角形中不超用线段的最多组合数
function canCombine(ts, index, count, num, ans) {
  if (index >= ts.length) {
    ans.push(num);
    return;
  }

  for (let i = index; i < ts.length; i++) {
    const [a, b, c] = ts[i];

    if (count[a] > 0 && count[b] > 0 && count[c] > 0) {
      count[a]--;
      count[b]--;
      count[c]--;
      num++;
      canCombine(ts, i + 1, count, num, ans);
      num--;
      count[a]++;
      count[b]++;
      count[c]++;
    }
  }

  ans.push(num);
}
```

Java算法源码

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

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

    int t = sc.nextInt();
    int[][] cases = new int[t][];

    for (int i = 0; i < t; i++) {
      int n = sc.nextInt();
      int[] arr = new int[n];
      for (int j = 0; j < n; j++) {
        arr[j] = sc.nextInt();
      }
      cases[i] = arr;
    }

    getResult(cases);
  }

  public static void getResult(int[][] cases) {
    for (int[] arr : cases) {
      // 对每组测试线段升序排序
      Arrays.sort(arr);

      ArrayList<Integer[]> res = new ArrayList<>();
      dfs(arr, 0, new LinkedList<>(), res);

      int[] count = new int[100];
      for (int i : arr) {
        count[i]++;
      }

      ArrayList<Integer> ans = new ArrayList<>();
      canCombine(res, 0, count, 0, ans);
      System.out.println(ans.stream().max((a, b) -> a - b).orElse(0));
    }
  }

  // 全组合求解，即n个数中选3个
  public static void dfs(int[] arr, int index, LinkedList<Integer> path, ArrayList<Integer[]> res) {
    if (path.size() == 3) {
      if (isRightTriangle(path)) {
        res.add(path.toArray(new Integer[3]));
      }
      return;
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

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

    int t = sc.nextInt();
    int[][] cases = new int[t][];

    for (int i = 0; i < t; i++) {
      int n = sc.nextInt();
      int[] arr = new int[n];
      for (int j = 0; j < n; j++) {
        arr[j] = sc.nextInt();
      }
      cases[i] = arr;
    }

    getResult(cases);
  }

  public static void getResult(int[][] cases) {
    for (int[] arr : cases) {
      // 对每组测试线段升序排序
      Arrays.sort(arr);

      ArrayList<Integer[]> res = new ArrayList<>();
      dfs(arr, 0, new LinkedList<>(), res);

      int[] count = new int[100];
      for (int i : arr) {
        count[i]++;
      }

      ArrayList<Integer> ans = new ArrayList<>();
      canCombine(res, 0, count, 0, ans);
      System.out.println(ans.stream().max((a, b) -> a - b).orElse(0));
    }
  }

  // 全组合求解，即n个数中选3个
  public static void dfs(int[] arr, int index, LinkedList<Integer> path, ArrayList<Integer[]> res) {
    if (path.size() == 3) {
      if (isRightTriangle(path)) {
        res.add(path.toArray(new Integer[3]));
      }
      return;
    }

    for (int i = index; i < arr.length; i++) {
      path.add(arr[i]);
      dfs(arr, i + 1, path, res);
      path.removeLast();
    }
  }

  // 判断三条边是否可以组成直角三角形
  public static boolean isRightTriangle(LinkedList<Integer> path) {
    // 注意，path中元素是升序的，因为path是取自arr的组合，而arr是升序的
    int x = path.get(0);
    int y = path.get(1);
    int z = path.get(2);

    return x * x + y * y == z * z;
  }

  // 求解当前直角三角形中不超用线段的最多组合数
  public static void canCombine(
      ArrayList<Integer[]> ts, int index, int[] count, int num, ArrayList<Integer> ans) {
    if (index >= ts.size()) {
      ans.add(num);
      return;
    }

    for (int i = index; i < ts.size(); i++) {
      Integer[] tri = ts.get(i);
      int a = tri[0];
      int b = tri[1];
      int c = tri[2];

      if (count[a] > 0 && count[b] > 0 && count[c] > 0) {
        count[a]--;
        count[b]--;
        count[c]--;
        num++;
        canCombine(ts, i + 1, count, num, ans);
        num--;
        count[a]++;
```

```java
  }

  // 全组合求解，即n个数中选3个
  public static void dfs(int[] arr, int index, LinkedList<Integer> path, ArrayList<Integer[]> res) {
    if (path.size() == 3) {
      if (isRightTriangle(path)) {
        res.add(path.toArray(new Integer[3]));
      }
      return;
    }

    for (int i = index; i < arr.length; i++) {
      path.add(arr[i]);
      dfs(arr, i + 1, path, res);
      path.removeLast();
    }
  }

  // 判断三条边是否可以组成直角三角形
  public static boolean isRightTriangle(LinkedList<Integer> path) {
    // 注意，path中元素是升序的，因为path是取自arr的组合，而arr是升序的
    int x = path.get(0);
    int y = path.get(1);
    int z = path.get(2);

    return x * x + y * y == z * z;
  }

  // 求解当前直角三角形中不超用线段的最多组合数
  public static void canCombine(
      ArrayList<Integer[]> ts, int index, int[] count, int num, ArrayList<Integer> ans) {
    if (index >= ts.size()) {
      ans.add(num);
      return;
    }

    for (int i = index; i < ts.size(); i++) {
      Integer[] tri = ts.get(i);
      int a = tri[0];
      int b = tri[1];
      int c = tri[2];

      if (count[a] > 0 && count[b] > 0 && count[c] > 0) {
        count[a]--;
        count[b]--;
        count[c]--;
        num++;
        canCombine(ts, i + 1, count, num, ans);
        num--;
        count[a]++;
        count[b]++;
        count[c]++;
      }
    }

    ans.add(num);
  }
}
```

Python算法源码

```python
# 输入获取
t = int(input())
cases = [list(map(int, input().split()))[1:] for i in range(t)]


# 算法入口
def getResult(cases):
    for case in cases:
        # 对每组测试线段升序排序
        case.sort()

        res = []
        dfs(case, 0, [], res)

        count = [0 for i in range(100)]
        for val in case:
            count[val] += 1

        ans = []
        canCombine(res, 0, count, 0, ans)
        print(max(ans))


# 全组合求解，即n个数中选3个
def dfs(arr, index, path, res):
    if len(path) == 3:
        if isRightTriangle(path):
            res.append(path[:])
        return

    for i in range(index, len(arr)):
        path.append(arr[i])
        dfs(arr, i + 1, path, res)
```

Python算法源码

复制

```python
# 输入获取
t = int(input())
cases = [list(map(int, input().split()))[1:] for i in range(t)]


# 算法入口
def getResult(cases):
    for case in cases:
        # 对每组测试线段升序排序
        case.sort()

        res = []
        dfs(case, 0, [], res)

        count = [0 for i in range(100)]
        for val in case:
            count[val] += 1

        ans = []
        canCombine(res, 0, count, 0, ans)
        print(max(ans))


# 全组合求解，即n个数中选3个
def dfs(arr, index, path, res):
    if len(path) == 3:
        if isRightTriangle(path):
            res.append(path[:])
        return

    for i in range(index, len(arr)):
        path.append(arr[i])
        dfs(arr, i + 1, path, res)
        path.pop()


# 判断三条边是否可以组成直角三角形
def isRightTriangle(path):
    # 注意，path中元素是升序的，因为path是取自arr的组合，而arr是升序的
    x, y, z = path
    return x ** 2 + y ** 2 == z ** 2


# 求解当前直角三角形中不超用线段的最多组合数
def canCombine(ts, index, count, num, ans):
    if index >= len(ts):
        ans.append(num)
        return

    for i in range(index, len(ts)):
        a, b, c = ts[i]

        if count[a] > 0 and count[b] > 0 and count[c] > 0:
            count[a] -= 1
            count[b] -= 1
            count[c] -= 1
            num += 1
            canCombine(ts, i + 1, count, num, ans)
            num -= 1
            count[a] += 1
            count[b] += 1
            count[c] += 1

    ans.append(num)


# 算法调用
getResult(cases)
```