1 Lebron and the multiverse

Problem statement: Lebron scored r three-point and t two-point shots in the basketball game, but in the «42» universe, k points are deducted from a player's performance until his performance becomes less than k.

Subtask 2

Let's find the total number of points that Lebron scored, this is $sm = 3 \cdot r + 2 \cdot t$. We will subtract k until $sm \geq k$.

```

r = int(input())
t = int(input())
k = int(input())
sm = 3 * r + 2 * t
while sm >= k:
    sm -= k
print(sm)
```

Subtask 3

To solve for a full score, you need to understand that after executing the loop, we get the remainder of the division by k.

```
r = int(input())
t = int(input())
k = int(input())
print((3 * r + 2 * t) % k)
```

The answer may not fit into a 32-bit integer type, so you should use 64-bit integers such as `long long` in C++ to avoid integer type overflow.

2 Lebron and the problem

Problem statement: Given a rectangle with sides n and m, the coordinates of the lower left corner of the rectangle are $(0; 0)$. It is necessary to find the number of right triangles whose vertices have integer coordinates and are inside the rectangle (the vertices can lie on the sides of the rectangle), and also one of the sides is parallel to the Ox axis, and the second to the Oy axis.

Subtask 1

Constraints allow you to iterate over all points and count the number of suitable triangles.

Subtask 2

Let's fix some point $A(x_1; y_1)$, then the other two points can only be located on the lines $x = x_1$ and $y = y_1$. To count the number of triangles, we will have to iterate over the two remaining points on these lines. Let's estimate the running time of this algorithm: we need to iterate over the coordinates of one point — $O(n \cdot m)$, for the other two — only one coordinate $O(n \cdot m)$, as a result $O(n^2 \cdot m^2)$.

```python
n = int(input())
m = int(input())

ans = 0
for x1 in range(n+1):
    for y1 in range(m+1):
        for y2 in range(m+1):
            if y1 == y2:
                continue

            for x3 in range(n+1):
                if x1 == x3:
                    continue
                ans += 1
print(ans)
```

Subtask 3

To solve the problem for a full score, let's estimate the exact number of iterations that these loops do:

```python
for y2 in range(m+1):
    if y1 == y2:
        continue

    for x3 in range(n+1):
        if x1 == x3:
            continue
        ans += 1
```

A loop with the variable $x3$ does n iterations, and a loop with $y2$ does m iterations, it turns out that in nested loops $m \cdot n$ units are added to the variable ans. Replace these two loops with $ans+ = n \cdot m$. The asymptotics of the solution has now become $O(n \cdot m)$.

Additional subtask

This problem can be solved in $O(1)$ asymptotics, for such a solution, you can once again make an accurate estimate of the number of iterations and modify the formula.

```python
n = int(input())
m = int(input())
print(n*m*(n+1)*(m+1))
```

The answer may not fit into a 32-bit integer type, so you should use 64-bit integers such as `long long` in C++ to avoid integer type overflow.

3 Gift to Lebron

Problem statement: An array of n integers is given. It is necessary to say for each prefix whether there are exactly two different numbers that occur the same number of times on this prefix.

Subtask 1

In the first subtask $a_i \leq 100$, let's iterate over the numbers from 1 to 100 on each prefix and count how many each number occurs. Now it remains to check whether there are two numbers that occur the same number of times.

```python
n = int(input())
a = list(map(int, input().split()))
for i in range(n):
  b = a[:i+1]

  cnt = []
  for j in range(1, 101):
    tmpCnt = b.count(j)
    if tmpCnt > 0:
      cnt.append(tmpCnt)

  ans = False
  for j in range(n+1):
    if cnt.count(j) == 2:
      ans = True
      break

  if ans:
    print("YES")
  else:
    print("NO")
```

Subtask 2

In the second subtask $1 \leq a_i \leq 2$, we will use this condition to solve. Let's create two variables *one* and *two*, in which we will count the number of ones and twos on the prefix, respectively. During the processing of a new element, we will add a unit to one of the variables, after which we need to check whether their number is the same.

```python
n = int(input())
a = list(map(int, input().split()))

one = 0
two = 0

for i in range(n):
  if a[i] == 1:
    one += 1
  else:
    two += 1

  if one == two:
    print("YES")
  else:
    print("NO")
```

Alternative solution to the subtask 1

Before presenting the solution of subtask 3, let us consider another solution of the first subtask. Restrictions on the array element $1 \leq a_i \leq 100$. Let's create an array *cnt*, in which we will maintain how many times k occurs, where $k = 1, 2, \ldots, 100$. When we process a new element a_i, we will add a unit to the cell of the array a_i, that is, $cnt[a_i] + = 1$. Now it remains to find exactly two elements that will have the same value, for this we will use the solutions described above.

```python
n = int(input())
a = list(map(int, input().split()))


mx = max(a)
cnt = [0] * (mx + 1)


for i in range(n):
  cnt[a[i]] += 1

  ans = False
  for j in range(1, n+1):
    if cnt.count(j) == 2:
      ans = True
      break

  if ans:
    print("YES")
  else:
    print("NO")
```

Subtask 3

Note that you can reuse the technique with an array, but to count the number of each element of the array *cnt*. Now, to check that there are exactly two elements on the prefix that occur the same number of times, we need to go through the *cnt*2 array.

```python
n = int(input())
a = list(map(int, input().split()))

mx = max(a)
cnt = [0] * (mx + 1)

cnt2 = [0] * (n+1)
for i in range(n):
  if cnt2[cnt[a[i]]] != 0:
    cnt2[cnt[a[i]]] -= 1

  cnt[a[i]] += 1
  cnt2[cnt[a[i]]] += 1

  ans = False
  for j in range(1, n+1):
    if cnt2[j] == 2:
      ans = True
      break
  if ans:
    print("YES")
  else:
    print("NO")
```

This implementation of the idea scored 50 points, but by slightly improving this solution, it was possible to get 65 points.

Subtask 4

To solve the problem for a full score, it should be noted that when processing a new element a_i, the number of pairs of elements that can satisfy the answer changes by no more than 2. Let's create a variable *ans*, which is responsible for the number of exactly two different numbers that occur the same number of times on this prefix. When processing a new number, there are several options for changing the variable *ans* (we believe that we have already added one to $cnt[a[i]]$ and there have been changes in $cnt2$):

1. if $cnt[a[i]] > 1$ and $cnt2[cnt[a[i]] - 1] == 1$, then $ans- = 1$
2. if $cnt2[cnt[a[i]] - 1] == 2$, then $ans+ = 1$
3. if $cnt2[cnt[a[i]]] == 3$, then $ans- = 1$
4. if $cnt2[cnt[a[i]]] == 2$, then $ans+ = 1$

After changing *ans*, it remains only to check whether this variable is greater than 0, if yes, then the answer is **YES**, if not, then **NO**.

```python
n = int(input())
a = list(map(int, input().split()))

mx = max(a)
ans = 0
cnt = [0] * (mx + 1)
cnt2 = [0] * (n+1)

for i in range(n):
  if cnt2[cnt[a[i]]] != 0:
    cnt2[cnt[a[i]]] -= 1

  if cnt2[cnt[a[i]]] == 1 and cnt[a[i]] > 0:
    ans -= 1

  if cnt2[cnt[a[i]]] == 2:
    ans += 1

  cnt[a[i]] += 1
  cnt2[cnt[a[i]]] += 1

  if cnt2[cnt[a[i]]] == 3:
    ans -= 1

  if cnt2[cnt[a[i]]] == 2:
    ans += 1

  if ans > 0:
    print("YES")
  else:
    print("NO")
```

4 Lebron and dominoes

Problem statement: Given an array of pairs of numbers (dominoes). In a pair, the numbers can be swapped, it is necessary to find the maximum sequence of consecutive pairs in which any two sides touched with the same number.

Subtask 1

To solve the first subtask, it was possible to write a brute force algorithm.

Subtask 3

Note the fact that if we fix a certain position of a pair, then only one option for the location of the next pair is possible. Now let's go through the beginning of the sequence and consider two options for the position of this domino. After that, we will simulate the construction.

```python
n = int(input())
a = []
for i in range(n):
  a.append(list(map(int, input().split())))

ans = 0
for i in range(n):
  tmp = [1, 1]
  for t in range(2):
    last = a[i][t]
    for j in range(i+1, n):
      if last == a[j][0]:
        last = a[j][1]
      elif last == a[j][1]:
        last = a[j][0]
      else:
        break
      tmp[t] += 1
  ans = max(ans, max(tmp))
print(ans)
```

Subtask 4

To solve the problem for a full score, you need to use the dynamic programming method. Let $dp[i][0]$ — be the length of consecutive dominoes ending in i domino without its swap, and $dp[i][1]$ — with a turn. The base of the dynamics is $dp[i][0] = dp[i][1] = 1$, since the minimum length is 1 (one domino is already a sequence). Transition options:

1. $a[i-1][1] == a[i][0]$, then $dp[i][0] = max(dp[i][0], dp[i-1][0] + 1)$
2. $a[i-1][0] == a[i][0]$, then $dp[i][0] = max(dp[i][0], dp[i-1][1] + 1)$
3. $a[i-1][1] == a[i][1]$, then $dp[i][1] = max(dp[i][1], dp[i-1][0] + 1)$
4. $a[i-1][0] == a[i][1]$, then $dp[i][1] = max(dp[i][1], dp[i-1][1] + 1)$

The answer is the maximum value among $dp[i][0]$ and $dp[i][1]$.

```python
n = int(input())
a = []
for i in range(n):
  a.append(list(map(int, input().split())))

```

```python
ans = 1
dp = [[1, 1] for i in range(n)]

for i in range(1, n):
  if a[i - 1][1] == a[i][0]:
    dp[i][0] = max(dp[i][0], dp[i - 1][0] + 1)

  if a[i - 1][0] == a[i][0]:
    dp[i][0] = max(dp[i][0], dp[i - 1][1] + 1)

  if a[i - 1][1] == a[i][1]:
    dp[i][1] = max(dp[i][1], dp[i - 1][0] + 1)

  if a[i - 1][0] == a[i][1]:
    dp[i][1] = max(dp[i][1], dp[i - 1][1] + 1)
  ans = max(ans, dp[i][0], dp[i][1])
print(ans)
```

5 Circles and basketball

Problem statement: Given n inequalities of the form $(x - x_i)^2 + (y - y_i)^2 \leq r_i^2$. It is necessary to count the number of integer points (x, y) that satisfy at least one of the inequalities.

Subtask 1

Constraints allow you to iterate over integer points that can satisfy at least one of the inequalities. In order not to take into account the same points several times, you can use a two-dimensional Boolean array or Set. Asymptotics: $O(200^2 \cdot n)$

```python
n = int(input())
inequalities = []
for i in range(n):
  inequalities.append(list(map(int, input().split())))

used = set()
for x_i, y_i, r_i in inequalities:
  for x in range(x_i - r_i, x_i + r_i+1):
    for y in range(y_i - r_i, y_i + r_i+1):
      if (x_i - x) * (x_i - x) + (y_i - y) * (y_i - y) <= r_i*r_i:
        used.add((x, y))
print(len(used))
```

Subtask 2

After analyzing the constraints, it is clear that the coordinates of the points can be from $-2 \cdot 10^4$ to $2 \cdot 10^4$. Let's iterate over the line $y = j$ and substitute y into the inequality (that is, we intersect the line $y = j$ with the circle $(x - x_i)^2 + (y - y_i)^2 = r_i^2$).

$$\begin{aligned}(x - x_i)^2 + (y - y_i)^2 &= r_i^2\\ (x - x_i)^2 &= r_i^2 - (y - y_i)^2\\ x - x_i &= \pm\sqrt{r_i^2 - (y - y_i)^2}\\ x &= x_i \pm \sqrt{r_i^2 - (y - y_i)^2}\end{aligned}$$

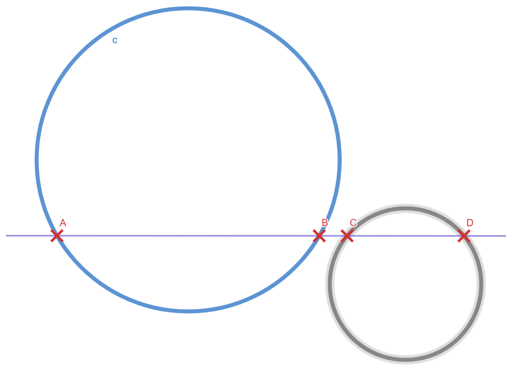
The entry $\lfloor x \rfloor$ denotes rounding down the value of x (for instance, $\lfloor 1.4 \rfloor = 1$, $\lfloor 2 \rfloor = 2$). Since only integer points are suitable, the necessary points will have coordinates $(\lfloor x_i - \sqrt{r_i^2 - (y - y_i)^2} \rfloor, y)$, $(\lfloor x_i - \sqrt{r_i^2 - (y - y_i)^2} \rfloor + 1, y)$, $\ldots$, $(\lfloor x_i + \sqrt{r_i^2 - (y - y_i)^2} \rfloor, y)$. Satisfying points with the coordinate $y = j$ total $\lfloor x_i + \sqrt{r_i^2 - (y - y_i)^2} \rfloor - \lfloor x_i - \sqrt{r_i^2 - (y - y_i)^2} \rfloor + 1 = 2 \cdot \lfloor \sqrt{r_i^2 - (y - y_i)^2} \rfloor + 1$.

```python
from math import sqrt

n = int(input())
inequalities = []
for i in range(n):
  inequalities.append(list(map(int, input().split())))

cnt = 0
for y in range(-20000, 20001):
  for x_i, y_i, r_i in inequalities:
    if abs(y - y_i) > r_i:
      continue
    d = int(sqrt(r_i * r_i - (y - y_i) * (y - y_i)))
    cnt += 2*d + 1
print(cnt)
```

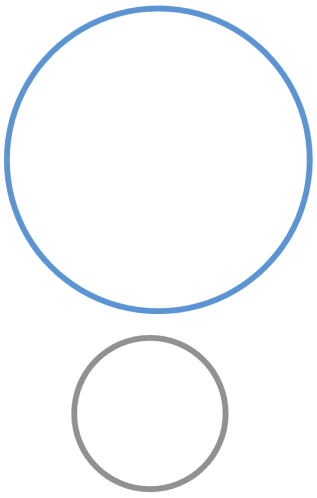
Subtask 3

In this subtask $n = 2$, we will use the method already described in subtask 2. We will intersect the line with circles, but this time we will consider the location of the intersection points with the circle. There are 4 location options in total:

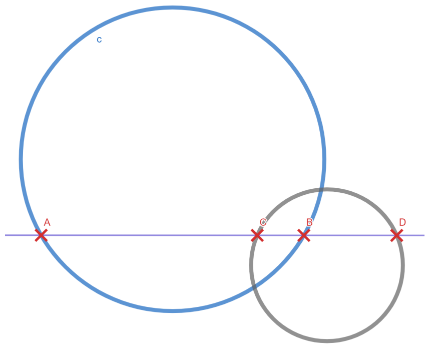
1. In this case, you need to check that the point B is to the left of C, so $x_B \leq x_c$, the number of suitable points: $(x_B - x_A + 1) + (x_D - x_C + 1)$
2. In this case, the straight line will never intersect two circles at the same time
3. Here you need to check that the points are located in this order: $A \to C \to B \to D$, that is $x_A \leq x_C \leq x_B \leq x_D$, the number of suitable points: $x_D - x_A + 1$
4. In the fourth case, one circle is inside the other, that is $x_A \leq x_C \leq x_D \leq x_B$, the number of suitable points: $x_B - x_A + 1$



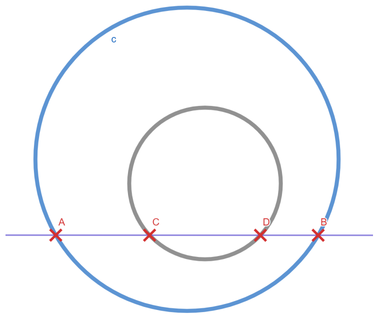
1.



2.



3.



4.

```python
from math import sqrt

n = int(input())
inequalities = []
for i in range(n):
  inequalities.append(list(map(int, input().split())))

cnt = 0
for y in range(-20000, 20001):
  points = []
  for x_i, y_i, r_i in inequalities:
    if abs(y - y_i) > r_i:
      continue
    d = int(sqrt(r_i * r_i - (y - y_i) * (y - y_i)))
    points.append((x_i - d, x_i + d))

  if len(points) == 0:
    continue

  points.sort()
  tmp = 0
  if len(points) == 1:
    A, B = points[0]
    tmp += B - A + 1
  elif len(points) == 2:
    A, B = points[0]
    C, D = points[1]

    if B < C:
      tmp += B - A + 1 + D - C + 1
    elif A <= C <= B <= D:
      tmp += D - A + 1
    else:
      tmp += B - A + 1
  cnt += tmp

print(cnt)
```

Subtask 4

To solve the fourth subtask, it is necessary to process the intersection of several segments for this we will use the scanning line method (ScanLine). It turns out that we need to count the number of points covered by segments on a straight line.

```python
from math import sqrt

n = int(input())

x = [0]*n
y = [0]*n
r = [0]*n
for i in range(n):
  x[i], y[i], r[i] = map(int, input().split())


plY = [[] for i in range(-20000, 20001)]

for i in range(n):
  for lineY in range(y[i] - r[i], y[i]+r[i]+1):
    if(abs(lineY - y[i]) > r[i]):
      continue
    d = int(sqrt(r[i]*r[i] - (lineY - y[i])**2))
    x1 = x[i] - d
    x2 = x[i] + d

    plY[lineY].append((x1, -1))
    plY[lineY].append((x2, 1))

ans = 0
for i in range(-20000, 20001):
  if len(plY[i]) == 0:
    continue

  plY[i].sort()
  cnt = 0
  lastX = 0
  for curX, curType in plY[i]:
    cnt -= curType

    if cnt == 0 and curType == 1:
      ans += curX - lastX + 1
    if cnt == 1 and curType == -1:
      lastX = curX

print(ans)
```