Road Rash Problem.

- Prepared by: Sagun Shakya (https://github.com/sagsshakya)
- MSc. Data Science
- GITAM Institute of Science, Visakhapatnam.
- Email: sags.shakya@gmail.com (mailto:sags.shakya@gmail.com)

On a busy road, multiple cars are passing by. A simulation is run to see what happens if brakes fail for all cars on the road. The only way for them to be safe is if they don't collide and pass by each other. The goal is to identify whether any of the given cars would collide or pass by each other safely around a Roundabout. Think of this as a reference point O (Origin with coordinates (0,0)), but instead of going around it, cars pass through it.

Considering that each car is moving in a straight line towards the origin with individual uniform speed. Cars will continue to travel in that same straight line even after crossing origin. Calculate the number of collisions that will happen in such a scenario.

Calculate collisions only at origin. Ignore the other collisions. Assume that each car continues on its respective path even after the collision without change of direction or speed for an infinite distance.

- Constraints
  - 1<=C<=10^5
  - -10^9 <= x,y <= 10^9
  - 0 < v < 10^9.
- Input Format
  - The first line contains an integer C, denoting the number of cars being considered that are passing by around the origin.
  - Next C lines contain 3 space delimited values, first two of them being for position coordinates (x,y) in 2D space and the third one for speed (v).
    Output
- A single integer Q denoting the number of collisions at origin possible for given set of cars.
  Example
- Input 1
  5
  5 12 1
  16 63 5
  -10 24 2
  7 24 2
  -24 7 2

- Output
  4
- Explanation
  Let the 5 cars be A, B, C, D, and E respectively.

4 Collisions are as follows -
A & B
A & C
B & C
D & E

In [29]:
```
from math import factorial as fact
```

Getting the inputs.

In [73]:
```
C = int(input())    # Number of cars passing by.
```

```
5
```

In [72]:
```
co_ordinates = []
velocity = []

for jj in range(C):
    temp_data = [int(jj) for jj in input().split()]    #[co - ordinates, velocity
    co_ordinates.append(tuple(temp_data[:-1]))
    velocity.append(temp_data[-1])
```

```
5 12 1
16 63 5
-10 24 2
7 24 2
-24 7 2
```

Functions for calculating Combination and Euclidean Distance.

In [74]:
```
# To calculate combinations.
combination = lambda n,r: int(fact(n) / (fact(r) * fact(n - r)))
# To calculate 2D distance from the origin.
standard_euclidean_distance = lambda x,y: ((x**2) + (y**2))**.5
```

To calculate the Euclidean distance from the origin to the respective points.

In [75]:
```
distance = [standard_euclidean_distance(*co_ordinates[ii]) for ii in range(len(cc
```

In [76]:
```
distance
```

Out[76]:
```
[13.0, 65.0, 26.0, 25.0, 25.0]
```

To calculate the time taken for the vehicles to reah the origin given their velocities and distance.

- t = d / v

In [ ]:
```
time = [distance[ii] / velocity[ii] for ii in range(len(distance))]
```

In [77]:
```
time
```

Out[77]:
```
array([13. , 13. , 13. , 12.5, 12.5])
```

Counting the number of unique values of time.

In [78]:
```
from collections import Counter

unique = tuple(Counter(time).values())
```

In [79]:
```
unique
```

Out[79]:
```
(3, 2)
```

For count > 1, there is no collision. So the total collisions = C(n1, 2) + C(n2, 2) + ... + C(nk, 2)

In [80]:
```
print(sum((combination(n = ii, r = 2) for ii in unique if ii>1)))
```

```
4
```

Aliter: Using Vectorization.

In [81]:
```
from math import factorial as fact
import numpy as np
```

In [82]:
```
C = int(input())

data = []
for jj in range(C):
    temp_data = [int(jj) for jj in input().split()]
    data.append(temp_data)

# Functions.
combination = lambda n,r: int(fact(n) / (fact(r) * fact(n - r)))
standard_euclidean_distance = lambda x,y: ((x**2) + (y**2))**.5

# Numpy array for the data.
data = np.array(data)

# Segmentation.
co_ordinates = data[:, :-1]
velocity = data[:, -1]

distance = standard_euclidean_distance(co_ordinates[:,0], co_ordinates[:,1])

time = distance / velocity
```

```
5
5 12 1
16 63 5
-10 24 2
7 24 2
-24 7 2
```

In [83]:
```
time
```

Out[83]:
```
array([13. , 13. , 13. , 12.5, 12.5])
```

Getting the number of unique values in numpy.

In [69]:
```
_, unique = np.unique(time, return_counts = True)
```

Output:

In [70]:
```
print(sum((combination(n = ii, r = 2) for ii in unique if ii>1)))
```

```
4
```

The End.