World War E.

- 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)

In a crossover fantasy universe, Houin Kyoma is up in a battle against a powerful monster Nomu that can kill him in a single blow. However being a brilliant scientist Kyoma found a way to pause time for exactly M seconds. Each second, Kyoma attacks Nomu with certain power, which will reduce his health points by that exact power. Initially Nomu has H Health Points. Nomu dies when his Health Points reach 0. Normally Kyoma performs Normal Attack with power A. Besides from Kyoma's brilliance, luck plays a major role in events of this universe. Kyoma's Luck L is defined as probability of performing a super attack. A super attack increases power of Normal Attack by C. Given this information calculate and print the probability that Kyoma kills Nomu and survives. If Kyoma dies print RIP.

- Constraints
  - 0 < T <= 50
  - 1 <= A, H, C, L1, L2 <= 1000
  - 1 <= M <= 20.
  - L1<=L2
- Input Format
  - First line is integer T denoting number of test cases
  - Each test case consist of single line with space separated numbers A H L1 L2 M C.
  - Where luck L is defined as L1/L2. Other numbers are, as described above.
- Output
  - Print probability that Kyoma kills Nomu in form P1/P2 where P1<=P2 and gcd(P1,P2)=1
  - If impossible, print RIP without quotes.
- Example Input 1
  - 2
  - 10 33 7 10 3 2
  - 10 999 7 10 3 2
- Output
  - 98/125
  - RIP

Taking Inputs.

In [65]:
```
T = int(input())  # Number of test cases.
```

2

In [66]:
```
test_cases = []
for ii in range(T):
    temp = [int(jj) for jj in input().split()]
    test_cases.append(temp)
```

```
10 33 7 10 3 2
10 999 7 10 3 2
```

Functions that we need to get through.

1. To calculate the combination nCr.
2. To determine the numerator of the Binomial Probability.
3. To convert the floating point to a rational number.

In [70]:
```
from math import gcd, factorial as fact

# Combination : nCr.
combination = lambda n,r: fact(n) / (fact(r) * fact(n - r))

# Binomial Distribution with parameters n,p; 'p' being the probability of success
# Note that p = l1 / l2.
binomial_distribution_modified = lambda n,l1, l2,x: combination(n,x) * (l1**x) *
```

In [73]:
```
# Conversion of floating point to fraction.
def float_to_ratio(flt):
    if int(flt) == flt:        # to prevent 3.0 -> 30/10
        return int(flt), 1
    flt_str = str(flt)
    flt_split = flt_str.split('.')
    numerator = int(''.join(flt_split))
    denominator = 10 ** len(flt_split[1])
    GCD = gcd(numerator, denominator)

    # Simplifying into simple ratio.
    while GCD != 1:
        numerator /= GCD
        denominator /= GCD
        GCD = gcd(int(numerator), int(denominator))

    return int(numerator), int(denominator)
```

In [76]:
```
for case in test_cases:
    A, H, L1, L2, M, C = case

    ##### Parameters:
    # A: Health taken using normal attack.
    # H: Full health of the monster.
    # L1: Numerator of the probability of super - attack.
    # L2: Denominator of the probability of super - attack.
    # M: Number of seconds frozen by the protagonist.
    # C: Extra points gained on attack if super attack is used.

    assert(L1 <= L2)

    max_points_normal_attack = A * M                  # Max. points that can be ga
    points_lag = H - max_points_normal_attack         # Points that will be lagged
    super_attack_point = A + C                        # Value of each super_attack

    if C*M < points_lag:                              # "Impossible - to - win" co
        print('RIP')

    else:
        X = range(M+1)                                # X is a Random variable whi

        # Checking for the minimum number of super attacks needed.
        for x in X:
            if x*C > points_lag:
                min_x = x
                break

        # Calculate the binomial probability for X > min_x.
        probability_accumulator = 0
        for r in range(min_x, M+1):
            probability_accumulator += binomial_distribution_modified(n = M, l1 =

        # Converting the probabilty to a simple fraction.
        numerator, denominator = float_to_ratio(probability_accumulator / (L2 **
        print(str(int(numerator)) + '/' + str(int(denominator)))
```

```
98/125
RIP
```

Break - down of the first test - case.

In [91]:
```
A, H, L1, L2, M, C = 10, 33, 7, 10, 3, 2
```

In [92]:
```
max_points_normal_attack = A * M
```

In [93]:
```
max_points_normal_attack
```

Out[93]:
```
30
```

In [94]:
```
points_lag = H - max_points_normal_attack
```

In [95]:
```
points_lag
```

Out[95]:
```
3
```

In [96]:
```
super_attack_point = A + C
```

In [97]:
```
super_attack_point
```

Out[97]:
```
12
```

In [99]:
```
if C*M < points_lag:
    print('RIP')
else:
    X = range(M+1)
    # Checking for the minimum number of super attacks needed.
    for x in X:
        if x*C > points_lag:
            min_x = x
            break

    print('The number of super attacks should be at least ', min_x, '.')
    print('Range of X: ' + str(min_x) + ' to ' + str(M))

    # Calculate the binomial probability for X> min_x.
    probability_accumulator = 0
    for r in range(min_x, M+1):
        probability_accumulator += binomial_distribution_modified(n = M, l1 = L1,


    # Conversion of floating point to fraction.
    def float_to_ratio(flt):
        if int(flt) == flt:        # to prevent 3.0 -> 30/10
            return int(flt), 1
        flt_str = str(flt)
        flt_split = flt_str.split('.')
        numerator = int(''.join(flt_split))
        denominator = 10 ** len(flt_split[1])
        return numerator, denominator

    numerator, denominator = float_to_ratio(probability_accumulator /(L2**M))   #

    from math import gcd
    GCD = gcd(numerator, denominator)

    while GCD != 1:
        numerator /= GCD
        denominator /= GCD
        GCD = gcd(int(numerator), int(denominator))

    print('\nProbability of using super attacks greater than or equal to ' + str(
          str(int(numerator)) + '/' + str(int(denominator)))
```

```
The number of super attacks should be at least  2 .
Range of X: 2 to 3

Probability of using super attacks greater than or equal to 2: 98/125
```

The End.