Второй отборочный этап

Индивидуальный модуль

Задача И1. Кессонный эффект (2,25 баллов)

Воздушный шарик запускают со дна Марианской впадины (в качестве **глубины** возьмем (https://www.rgo.ru/ru/article/rossiyskiy-podvodnyy-apparat-vityaz-dostig-dna-marianskoy-vpadiny) 10 028 м). Шарик наполнен **гелием**, **масса** которого известна, а также обладает предельной **площадью** поверхности, по превышении которой он рвется.

Вычислите, на какой **глубине** шарик лопнет. Гарантируется, что он **точно** лопнет, не достигнув поверхности океана.

Плотность воды считать постоянной и равной 1030 кг/м^3, аналогично постоянна ее **температура** — +4°C, атмосферное давление — 101 325 Па, ускорение свободного падения — 9.81 м/с^2. **Форму** шара считать идеальной **сферической**, его **натяжением** пренебречь, а **газ** внутри него считать **идеальным**.

Входной формат: два вещественных числа через пробел, масса газа в шарике (в граммах) и предельная площадь его поверхности (в м^2).

Выходной формат: единственное вещественное число, глубина в метрах, на которой шарик лопнет.

Балл за решение зависит от **точности** вашего ответа. Если он отличается от авторского **не более, чем на** 1 метр, вы получаете полный балл. При расхождении **не более, чем на** 5 метров, вы получаете 50%. При большем расхождении решение **не засчитывается**.

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

Для решения этой задачи у вас есть 15 попыток.

Алгоритм решения

Решение этой задачи целиком полагается на работу с физической моделью. Для начала мы находим критические параметры шара, исходя из предельной (критической) площади поверхности:

$$R_K = \sqrt{\frac{S_K}{4\pi}}; V_K = \frac{4\pi}{3} R_K^3$$

Т.к. мы имеем дело с идеальным газом, применяем уравнение Менделеева – Клапейрона для получения критического давления гелия внутри шара:

$$pV_K = MR_KT/m \Rightarrow p_K = MRT/(mV_K)$$

Предельная глубина достигается в той точке, где критическое давление гелия равно внешнему давлению, которое складывается из гидростатического и атмосферного:

$$p_K = p_0 + \rho g h_K \Rightarrow h_K = (p_K - p_0)/\rho g \Rightarrow h_K = (MRT/(mV_K) - p_0)/\rho g$$

Задача И2. Ученью — свет (3,75 баллов)

Энергетика многогранна, и помимо интересных исследовательских задач есть **рутинные**, но очень **важные**. Например, строительство **подстанций**, обеспечивающих электричеством строящиеся микрорайоны. Для правильного выбора их мощности необходимо предварительно рассчитать **ориентировочный объем** потребления.

Вам предлагается определить **расчетную активную нагрузку** подстанции, подключающей университетский кампус к городской электросети.

Опираться следует на значения мощности, приведенные в СН-167-61 (https://www.websor.ru/raschet_nagruzok_zdanii.html) (заметим, документ давно вытеснен более актуальной версией, но в рамках задачи мы воспользуемся этим). Гарантируется, что все объекты можно **однозначно** определить в указанном стандарте. То есть они все **присутствуют** в документе, и их тип для расчета коэффициента **прописан** явным образом. Так как во всех зданиях предполагается централизованное горячее водоснабжение и возможность установки кондиционеров, во всех **диапазонах** нагрузок и **коэффициентов** для общественных и коммунальных зданий берется **верхняя** граница (то есть число мест влияет лишь на **категорию**, но не на значение внутри диапазона). Коэффициент одновременности считается **отдельно** для каждого дома. Нагрузки от освещения и лифтов в данной задаче **не учитываются**.

Входной формат: на первой строке приводится **численность населения** города, после чего произвольное число строк (не больше 20), на каждой из которых приведено краткое словесное описание объекта. Например:

население 539 тыс. чел.

20 жилых домов на 40 квартир площадью 27 м^2 с газовыми плитами

2 столовых на 150 мест

детский сад на 135 мест с электроплитами

больница на 50 коек

поликлиника

Выходной формат: единственное вещественное число, **расчетная активная нагрузка** подстанции в КВт.

Коэффициент одновременности считать для отдельного дома.

Решение засчитывается, если ваш ответ отличается от авторского не более, чем на 10 КВт.

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

Для решения этой задачи у вас есть 15 попыток.

Алгоритм решения

Решение этой задачи прямолинейно: открыть стандарт, извлечь из него необходимые константы согласно входных данных и посчитать нагрузку. Главной сложностью здесь является именно извлечение нужных для расчета данных из стандарта. Рассмотрим таблицы с этими данными. Для расчета нагрузки для квартиры требуется знать ее тип и площадь. Исходя из таблицы 3–12, получаем следующую таблицу:

Тип квартир	**Нагрузка в пределах 25–35 м^2 (кВт)**	**Нагрузка на площадь (Вт/м^2)**
Без электроплит, население до 1 млн чел.	0,8	27
Без электроплит, население от 1 млн чел.	1	33
С электроплитами, население до 1 млн чел.	1	33
С электроплитами, население от 1 млн чел.	1,2	40

Из числа квартир и их типа из таблицы 3–13 мы получаем коэффициент спроса:

Число квартир в доме	**КС без электроплит**	**КС с электроплитами**
5	0,7	0,62
10	0,62	0,47
20	0,5	0,4
...	...	...

Так, нагрузка для жилого комплекса равна **произведению** числа домов, числа квартир, расчетной нагрузки каждой квартиры и коэффициента спроса. Для остальных зданий нагрузка считается как произведение основной нагрузки (таблица 3–14) на коэффициент участия (таблица 3–15). В задаче использованы только два типа зданий:

Тип заведения	**Коэффициент участия**
Школы, детские и медициские учреждения, общепит, бытовое обслуживание	0,7
Магазины и зрелищные предприятия	1

Так, для каждого здания расчетная нагрузка будет равна:

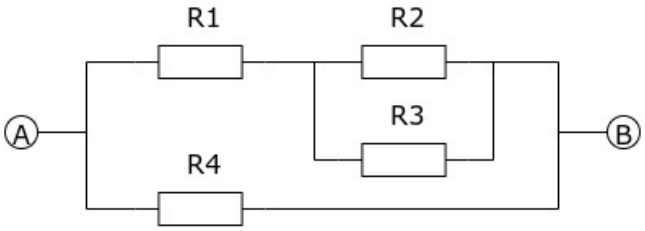
Вид заведения	**Расчётная нагрузка**
Ресторан	$210 \times 0,7 = 147$
Больница	Более 50 коек: $450 \times 0,7 = 315$ Менее 50 коек: $75 \times 0,7 = 52,5$
Школа	От 1000 мест: $160 \times 0,7 = 112$ До 1000 мест: $90 \times 0,7 = 63$
Кинотеатр	От 500 мест: 180 До 500 мест: 150
Магазин	60 с холодильниками, 30 без них

Вид заведения	**Расчётная нагрузка**
Детский сад	С электроплитами: $75 \times 0,7 = 52,5$ Без электроплит: $30 \times 0,7 = 21$
Универмаг	120
Ремонтная мастерская	$50 \times 0,7 = 35$
Поликлиника	$150 \times 0,7 = 105$
Столовая	$100 \times 0,7 = 70$

Суммируя расчетные нагрузки для каждой строки, мы получаем общую расчетную нагрузку, которая и является ответом на задачу.

Задача И3. Сопротивление на удачу (5,25 баллов)

Излишне решительный электронщик Саша взялся ремонтировать найденный на улице старый радиоприемник. Осмотрев платы, Саша заметил, что один из участков схемы представляет собой сопротивление из нескольких **резисторов**, собранных в **цепь**, как на рисунке.



Проблема лишь в том, что кто-то до Саши аккуратно выпаял эти резисторы, и теперь ему предстоит **восстановить** эту цепь. Саша решил подобрать номинал самым решительным образом — случайным **перебором** номиналов. Так, он впаивает резисторы в цепь, выбирая их **случайно** из большой кучи. Настолько большой, что вероятность достать тот или иной номинал **не зависит** от того, какие резисторы взяты ранее. При этом вероятность вытащить любой номинал такая же, как вероятность вытащить любой другой.

Заработает ли прибор после такого, мы сказать не можем. Но мы можем посчитать **математическое ожидание** сопротивления участка этой цепи, что и должна сделать ваша **программа**. Номиналы резисторов **известны** и передаются на вход.

Входной формат: 10 целых чисел через пробел, каждое соответствует номиналу доступного резистора в омах. Значение номинала находится в пределах от 0 до 10 кОм.

Выходной формат: единственное вещественное число, математическое ожидание сопротивления цепи в омах.

Решение **засчитывается**, если ответы, выдаваемые вашей программой, различаются от правильных **не более, чем на** 10^{-4}.

Ограничение по времени выполнения программы — 3 секунды.

На решение этой задачи у вас есть 15 попыток.

Алгоритм решения

В этой задаче мы имеем дело с тривиальной параллельно-последовательной цепью, вычисление которой не требует подробного описания (формулы для расчета соединения сопротивлений есть в школьном курсе физики). Единственный нестандартный момент — наличие нулевого резистора, который обнуляет сопротивление параллельного соединения, но если это не учесть, программа будет выдавать исключение из-за деления на нуль.

Математическое ожидание — это сумма произведений значений случайной величины и вероятности выпадения этой величины. В случае с равновероятными исходами (а в нашем случае все резисторы выбираются независимо и равновероятно) значением матожидания будет среднее арифметическое всех возможных значений сопротивления цепи.

Для четырех резисторов из 10 номиналов существует 10000 вариантов выбора, которые легко перебрать даже за 1 секунду. В этом и заключается задача: перебрать все номиналы в четырех вложенных циклах, для каждой комбинации рассчитать сопротивление цепи, после чего вычислить среднее арифметическое и выдать его в качестве ответа.

Авторское решение

```
def resist(a, b, c, d):
    bc = 0 if (b == 0 or c == 0) else 1 / (1/b + 1/c)
    r = 0 if (d == 0 or bc + a == 0) else 1 / (1/d + 1/(bc+a))
    return r

noms = list(map(int, input().split()))
su = sum(resist(a, b, c, d)
         for a in noms for b in noms
         for c in noms for d in noms)
co = len(noms) ** 4
print(su/co)
```

Задача И4. Цепной забор (0,75 баллов)

Есть **последовательность** точек на декартовой плоскости, из нее строится следующая **фигура**. Первые две точки образуют между собой **отрезок**. У этого отрезка берется **середина**, от нее строится отрезок со **следующей** точкой в последовательности. Затем, у **этого** отрезка берется середина... Так, каждая **последующая** точка образует отрезок с серединой **предыдущего** отрезка.

Ваша **задача** — вычислить **суммарную длину** такого «забора». Ради науки, конечно же.

Входной формат: N строк (до 200), в каждой из которых по два вещественных числа через пробел. Эти пары соответствуют координатам опорных точек в порядке перечисления.

Выходной формат: единственное вещественное число, сумма длин отрезков в полученной фигуре.

Решение **засчитывается**, если ваш ответ различается с правильным **не более,**

чем на 10^{-4}.

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

На решение этой задачи у вас есть 15 попыток.

Алгоритм решения

Математическая модель этой задачи тривиальна: берем две первые точки, вычисляем длину отрезка и кладем ее в сумму. Берем середину отрезка и следующую точку, прибавляем в сумму длину нового отрезка. Повторяем описанное в прошлом предложении, пока точки не закончатся.

Единственным нетривиальным моментом будет тот факт, что точек на входе достаточно много (до 200), и ручные вычисления здесь очевидно неразумны. Нужно писать программу, либо оформлять электронную таблицу. В обоих случаях реализация достаточно проста и не требует приведения в этом тексте.

Задача И5. Чистая дилемма (3 балла)

В теории игр нередко применяют компьютерное моделирование, чтобы проанализировать поведение типовых **стратегий** в той или иной игре. Одной из распространенных форм представления моделируемой игры является платежная матрица (`https://ru.wikipedia.org/wiki/Нормальная_форма_игры`), где в зависимости от поведения каждого из игроков определяется изменение их счета.

Промоделируем ситуацию, когда А и Б находят куст с ягодами и решают его поделить. Каждый из игроков может либо **подождать**, чтобы куст дозрел, либо **собирает** с него ягоды сразу. Пример **платежной матрицы** для такой игры:

	Б ждет	**Б собирают**
А ждет	А получает 25 ягод Б получает 25 ягод	А получает 0 ягод Б получает 40 ягод
А собирает	А получает 40 ягод Б получает 0 ягод	А получает 10 ягод Б получает 10 ягод

Пусть А и Б находят 10 разных кустов (т. е. игра моделируется 10 раундов подряд). Вы решили спросить знакомых, как бы они поступили в такой ситуации. Вам предложили следующие варианты (назовем их алгоритмами):

- **«Кооператор»** (код К), который всегда соглашается подождать;
- **«Предатель»** (код П), который всегда собирает сразу;
- **«Око за око»** (код О), который сначала ждет, а затем поступает так, как поступил соперник в прошлом раунде.

Так как все алгоритмы ведут себя строго детерминированно, вы можете **посчитать** результат каждого из турниров между ними. В этом и заключается задача.

Входной формат: 2 строки по 4 целых числа, формирующих платежную матрицу игры. Так, следующий пример соответствует матрице, приведенной выше:

25 25 0 40

40 0 10 10

Выходной формат: **9** строк, содержащих **результат** игры трех алгоритмов между собой. Порядок строк может быть **любым**. Формат следующий: сначала идут коды сыгравших алгоритмов (см. выше), после них через пробел их суммарные баллы за турнир. Так, строка

К-О 314 159

означает, что алгоритм «Кооператор» играл в роли А против алгоритма «Око за око» в роли Б и получил 314 очков, когда как «Око за око» получил 159 очков.

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

На решение этой задачи у вас есть 15 попыток.

Алгоритм решения

В этой задаче используются чистые стратегии, а используемая матрица постоянна в течение всех раундов и турниров. Таким образом, все раунды между в пределах турнира между одной парой стратегий заканчиваются одинаково, и лишь для стратегии «Око за око» отличается первый раунд, причем только в турнире против «Предателя». Это позволяет рассчитать исход по формуле (как вручную, так и программно без циклов). Сложность может возникнуть лишь при выборе корректного значения из матрицы, но пример входных данных специально подобран в соответствии с таблицей из условия.

Также упрощает задачу тот факт, что генерируемые платежные матрицы являются симметричными (а также соответствуют дилемме заключенного), поэтому каждую пару стратегий нужно рассчитывать только один раз.

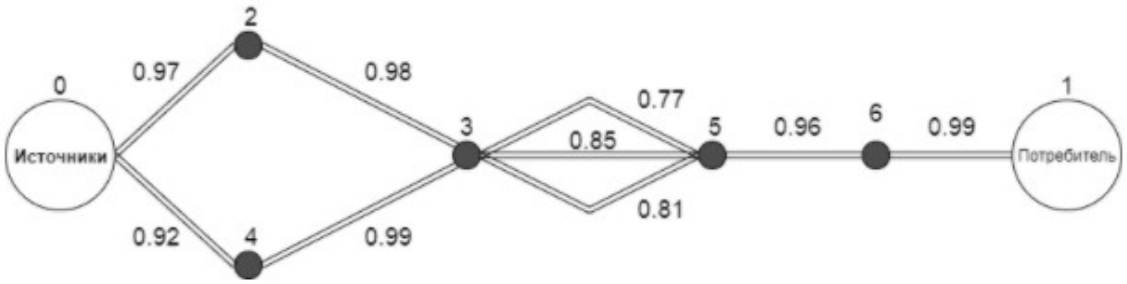
Командный модуль

Основные задачи

Задача К1. Вероятный отказ (13,82 баллов)

Система **электроснабжения** соединяет источники электроэнергии с потребителями. Это сложная система из большого числа элементов. Упрощенно ее можно представить как схему, в которой источники соединяются с потребителями **каналами передачи** электроэнергии. Каналы передачи электроэнергии тоже являются сложными системами, в которых могут происходить **поломки, отказы** и т. п. Поэтому в системах снабжения важно рассчитывать показатели **надежности**.

Пример **схемы** с показателями надежности (вероятность безотказной работы на определенном интервале времени):



Если два элемента соединены **последовательно**, то их общая надежность определяется как произведение:

$$P_{AB} = P_A \cdot P_B$$

Если два элемента соединены **параллельно**, то их общая надежность определяется так:

$$P_{AB} = 1 - (1 - P_A) \cdot (1 - P_B)$$

Дан список линий электропередачи. Для каждой линии указано, какие узлы она **соединяет** и вероятность того, что она находится в **рабочем** состоянии в данный момент времени. Вам необходимо написать **программу**, вычисляющую **надежность** такой схемы.

В этой задаче рассматриваются схемы **только** с параллельными и последовательными соединениями. Номер **Источника** всегда равен 0, номер **Потребителя** — 1.

Входной формат: список троек чисел, где первые два целых числа соответствуют **номерам** соединяемых узлов (до 160), а третье вещественное от 0 до 1 — показатель **надежности**. Например для рисунка,

[(0, 2, 0.97), (0, 4, 0.92), (2, 3, 0.98), (4, 3, 0.99), (3, 5, 0.77), (3, 5, 0.85), (3, 5, 0.81), (5, 6, 0.96), (6, 1, 0.99)]

Выходной формат: единственное вещественное число от 0 до 1, вероятность **продолжения работы** Потребителя от Источника в данный момент времени. Необходимо дать ответ с точностью до 5-го знака после запятой.

Ограничение по времени выполнения программы — 3 секунды.

Для решения этой задачи у вашей команды есть 20 попыток.

Алгоритм решения

Схема представляет собой граф, формирующий параллельно- последовательное соединение линий между точками 0 и 1. Это соединение похоже на цепь из резисторов, отличаются лишь формулы пересчета параметров эквивалентных линий. В данном случае (учитывая ограничение в 3 секунды на максимум 250 линий) рациональнее всего реализовать «схлопывание» графа таким же образом, как производится сокращение цепи из резисторов при вычислении общего сопротивления.

Для этого требуется две структуры данных типа словарь: список смежности, т. е. какие вершины смежны между собой (ключ — вершина, значение — множество вершин); и значения параллельных ребер, соединяющих искомую пару вершин (ключ — пара вершин, значение — список значений надежности).

После этого выполняется цикл до тех пор, пока за очередную итерацию не будет выполнено ни одного сокращения. В каждой итерации перебираются вершины (кроме

0 и 1). Если вершина имеет двух соседей — она сокращается, а две линии между соседями соединяются в одну с пересчетом надежности получившейся линии. Так, если какая-то из линий является параллельным соединением (т. е. вершина соединена более чем одним ребром) — оно сокращается в одну линию, после чего значения для обеих линий перемножаются. Обе формулы приведены в условии.

После цикла гарантированно остается единственное соединение 0–1, которое по необходимости сокращается из параллельного, и его значение пишется в качестве ответа.

Авторское решение

```
from collections import defaultdict

def mult(c):
    acc = 1
    for x in c: acc *= x
    return acc
def foldP(l):
    return 1 - (mult(1 - x for x in l)) \
           if len(l) >= 2 else l[0]
def ts(pair):
    return tuple(sorted(pair))

wires = eval(input())
inc = defaultdict(set)
nodes = defaultdict(list)
for (a, b, R) in wires:
    inc[a].add(b); inc[b].add(a)
    nodes[ts([a, b])].append(R)

while True:
    flag = False
    for x in inc:
        if x == 0 or x == 1: continue
        if len(inc[x]) == 2:
            e = (a, b) = ts(list(inc[x]))
            l = ts([a, x]); r = ts([b, x])

            Rl = foldP(nodes[l]); Rr = foldP(nodes[r])
            nodes[e].append(Rl*Rr)
            del nodes[l]; del nodes[r]
            inc[a].add(b); inc[b].add(a)
            inc[a].remove(x); inc[b].remove(x)
            inc[x].remove(a); inc[x].remove(b)
            flag = True
    if not flag: break

print(foldP(nodes[(0, 1)]))
```

Задача K2. Солнцестояние (6,91 баллов)

Рассмотрим солнечную электростанцию как систему солнечных панелей. Для повышения выработки электроэнергии панель должна быть направлена в сторону **солнца**. Это означает, что угол между плоскостью панели и лучами солнца должен быть как можно ближе к 90°. Поэтому солнечные панели оснащены **поворотным**

механизмом.

Пусть для i-й панели поворотный механизм **включается** каждые T_{pi} минут. И выполняет **поворот** в течение T_{wi} секунд. Используются солнечные панели разных моделей, поэтому моменты включения поворотных механизмов и время вращения разных панелей друг с другом могут **отличаться**.

Есть беспилотная мобильная роботизированная платформа (**БМРП**), выполняющая осмотр солнечных панелей **по очереди**. **Осмотр** каждой панели длится ровно T_c минут, при этом панель должна быть **неподвижна**. Если во время осмотра панель начинает вращаться, то БМРП ждет, пока панель **завершит** вращение, и продолжает осмотр. Суммарное **время осмотра** за именно неподвижной панелью в любом случае будет ровно T_c минут.

Известно, что Tc меньше, чем $T_p - T_w$ любой панели. Время перемещения БМРП между панелями равно T_t минут. Порядок осмотра всегда одинаковый и соответствует. Время начала осмотра и текущие фазы электростанций неизвестны, так что нельзя определить, в какой точно момент БМРП подъедет к какой панели.

Нужно определить **среднее время осмотра** электростанции, считая от момента времени, когда БМРП **оказалась** у первой платформы, до того момента времени, когда она **закончила** осмотр последней.

Входной формат: три объекта через запятую: список пар параметров солнечных панелей (T_w и T_p соответственно) в порядке осмотра, время осмотра T_c и время перемещения T_t. Все числа целые ненулевые. Всего панелей до 30. Например,

[(5, 30), (10, 40), (30, 60)], 2, 15

Выходной формат: единственное вещественное число, **среднее время** осмотра всех панелей **в секундах**. Решение засчитывается, если ответ вашей программы отличается от авторского менее, чем на 10^{-2} с.

Ограничение по времени выполнения программы — 3 секунды.

Для решения этой задачи у вашей команды есть 20 попыток.

Алгоритм решения

Математическая модель задачи сводится к вычислению **математического ожидания** времени проезда. Благодаря линейности матожидания мы можем разложить время проезда (а это сумма времен осмотра каждой панели и проезда между ними) на составляющие и рассмотреть каждый элемент суммы в отдельности.

Сразу замечаем, что время переезда между панелями T_t затрачивается всегда (т. е. с вероятностью 100%) и никак не варьируется. То есть мы можем его сразу вынести в ответ, например, $T_t \cdot (N-1)$. Так, задача сводится к поиску матожидания времени **осмотра** отдельной панели.

Здесь мы имеем три варианта прибытия БМРП к панели.

Первый — платформа прибыла, а панель уже (или еще) вращается. Вероятность этого варианта равна T_w/T_p, а время равно $T_c + T_w/2$.

Последнее слагаемое ($T_w/2$) имеет такое значение за счет свойства матожидания непрерывной линейной функции с равнораспределенной вероятностью. Другими словами, если мы приезжаем к началу вращения, мы ждем T_w секунд, если к концу

вращения, то 0. Остальные возможные значения времени ожидания зависят от времени приезда и считаются линейно. А так как вероятность попасть в тот или иной момент этого промежутка распределена равномерно, матожидание времени вращения, которое платформа будет ждать, равно **среднему арифметическому**, т. е. $T_w/2$.

Второй вариант — платформа начала осматривать панель, но та начала вращаться во время осмотра. Вероятность этого варианта равна T_c/T_p (т.к. мы должны попасть в промежуток времени величиной T_c перед началом осмотра панели), а время равно $T_c + T_w$ (ждем окончание вращения и затем заканчиваем осмотр).

Третий вариант — панель стояла смирно. Время будет равно — T_c, а вероятность — $1-(T_c+T_w)/T_p$, т. е. вероятность события, обратного двум указанным выше.

Матожидание равно сумме произведений величины на соответвующую ей вероятность. Но перед этим мы можем заметить, что в каждой величине используется T_c. Т.к. три варианта формируют 100% вероятность, мы можем вынести время осмотра T_c для каждой панели в постоянную часть, и у нас остается матожидание **простоя**, которое будет равно:

$$M = T_w/T_p \cdot T_w/2 + T_c/T_p \cdot T_w = T_w \cdot (T_w/2 + T_c)/T_p$$

Остается перенести получившуюся формулу в код, внимательно конвертируя минуты в секунды.

Авторское решение

```
a, tc, tt = eval(input())
tc, tt = tc*60, tt*60
s = 0
for (tw, tp) in a:
    tp *= 60
    s += tw*(tc+tw/2)/tp + tc + tt
print(s - tt)
```

Задача К3. Немного детерминированный конь (15,20 баллов)

На заданной клетке доски стоит конь и делает случайные ходы.

Подзадача 1. На заданной клетке доски стоит конь и делает случайные ходы. Какова вероятность, что через 10 ходов конь окажется на клетке A1?

Входной формат: координаты клетки в шахматной нотации (нижний регистр). Например, a1.

Выходной формат: единственное вещественное число, вероятность искомого события.

Ваш ответ засчитывается, если он отличается от авторского менее, чем на 10^{-6}. Балл за подзадачу 4,15.

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

Подзадача 2. На заданной клетке доски стоит **конь** и делает случайные ходы. Какова вероятность, что в течение N **ходов** конь **ни разу не поставит шах** королю

на клетке **D6**? Условимся при этом, что король всё это время никуда **не двигается**, и ставить коня на него **нельзя**.

Входной формат: через пробел целое число, количество ходов N (от 90 до 100), и координаты клетки в шахматной нотации (нижний регистр). Например, 95 a1. Гарантируется, что этой клеткой не будет D6.

Выходной формат: единственное вещественное число, вероятность искомого события.

Ограничение по времени выполнения программы — 3 секунды.

Ваше решение **засчитывается**, если выдаваемые им ответы отличаются от авторского решения **не более**, чем на 1%. Балл за подзадачу 11,05.

Для решения каждой подзадачи у вашей команды есть 20 попыток.

Алгоритм решения

Задача требует рассчитать вероятность события с большим числом вариантов. В первой подзадаче их будет около миллиарда ($8^{10} = 1073741824$). Конечно, в этом случае отсутствие ограничения по времени может подтолкнуть к перебору, но здесь он не даст решение за разумное время.

Поэтому здесь нужно прибегнуть к динамическому программированию, а конкретно в этой задаче:

- завести матрицу 8 на 8, где каждой ячейке соответствует вероятность искомого события, если конь начинает с соответствующей клетки;
- заполнить матрицу для конечного положения («через 0 ходов»);
- пересчитать матрицу N раз (N — искомое число ходов), где каждая клетка пересчитывается исходя из клеток, откуда можно сходить конем;
- взять ячейку, соответствующую искомой клетке, и вывести ее в качестве ответа.

Для **первой подзадачи** (вероятность того, что за N ходов конь с этой клетки попадет на A1) стартовой матрицей будет та, где значение для клетки A1 равно 1, т. е. 100%, а новое значение клетки равно среднему арифметическому вероятностей для всех клеток, откуда можно попасть на текущую.

Для **второй подзадачи** (вероятность того, что в течение N ходов конь ни разу не атакует короля) мы начинаем с матрицы, где для всех клеток, кроме атакуемых, вероятность равна единице, а пересчет выполняется аналогично первой подзадаче, но для клеток, с которых атакуется король, вероятность считается нулевой.

Причем в случае со второй задачей есть небольшая **оптимизация**: если на вход передается клетка, с которой атакуется король, то ответ очевидно будет нулевым.

Учитывая большую глубину перебора, для более точного ответа имеет смысл использовать типы данных, более аккуратно работающие с дробными числам (decimal или Fraction).

Авторское решение

Первая подзадача:

```python
from fractions import Fraction as frac

def get(a, i, j):
    if 0 <= i < 8 and 0 <= j < 8:
        return a[i][j], 1
    return 0, 0

n = 8
data = input()
x, y = ord(data[0]) --- ord('a'), ord(data[1]) - ord('1')
probs = [[frac(0) for _ in range(n)] for _ in range(n)]
new_probs = [[frac(0) for _ in range(n)] for _ in range(n)]
probs[0][0] = frac(1)
for mv in range(10):
    for i in range(n):
        for j in range(n):
            a, b = map(sum, zip(
                get(probs, i+2, j+1), get(probs, i+2, j-1),
                get(probs, i-2, j+1), get(probs, i-2, j-1),
                get(probs, i+1, j+2), get(probs, i+1, j-2),
                get(probs, i-1, j+2), get(probs, i-1, j-2)
            ))
            new_probs[i][j] = a / b
    new_probs, probs = probs, new_probs
print(float(probs[x][y]))
```

Вторая подзадача:

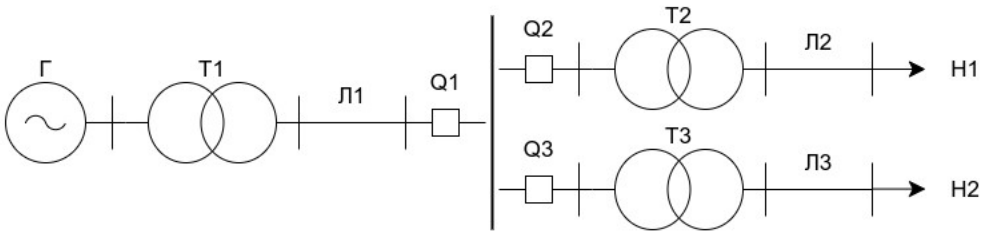
```python
from fractions import Fraction as frac

n = 8    # 3, 5 = d6
KINGCS = {(3, 5), (2, 7), (4, 3), (5, 4),
          (1, 4), (2, 3), (5, 6), (1, 6), (4, 7)}

def get(a, i, j):
  if 0 <= i < 8 and 0 <= j < 8 and not (i == 3 and j == 5):
    return (a[i][j] if (i, j) not in KINGCS else frac(0)), 1
  return 0, 0

moves, data = input().split()
moves = int(moves)
x, y = ord(data[0]) --- ord('a'), ord(data[1]) - ord('1')
if (x, y) in KINGCS:
    print("0")
    exit()
probs = [[frac(1) for _ in range(n)] for _ in range(n)]
new_probs = [[frac(0) for _ in range(n)] for _ in range(n)]

for mv in range(moves):
    for i in range(n):
        for j in range(n):
            a, b = map(sum, zip(
                get(probs, i+2, j+1), get(probs, i+2, j-1),
                get(probs, i-2, j+1), get(probs, i-2, j-1),
                get(probs, i+1, j+2), get(probs, i+1, j-2),
                get(probs, i-1, j+2), get(probs, i-1, j-2)
            ))
            new_probs[i][j] = a / b
    new_probs, probs = probs, new_probs
```

```
32
33 print(f"{float(probs[x][y]):.16f}")
```

Задача К4. Баланс в напряжении (17,28 баллов)

Известно, что электроэнергия передается по проводам, но до потребителя доходит **меньше**, чем было произведено. Все дело в **потерях**. Чаще всего они возникают на проводниках и трансформаторах.

Есть **схема автономной системы электроснабжения** (см. рисунок).



Состав схемы следующий:

- **Т1**: Повышающий трансформатор марки ТМ 630/10. Номинальная мощность $S_{HT1} = 630$ кВА; номинальные напряжения $(U_{BH}/U_{HH}) = \frac{10{,}5\text{ кВ}}{0{,}4\text{ кВ}}$; потери холостого хода $\Delta P_{XX1} = 1{,}1$ кВт, потери короткого замыкания $\Delta P_{KZ1} = 8{,}1$ кВт.
- **Т2, Т3**: Понижающие трансформаторы марки ТМ 250/10. Номинальная мощность $S_{HT2(3)} = 250$ кВА; номинальные напряжения $(U_{BH}/U_{HH}) = \frac{10{,}5\text{ кВ}}{0{,}4\text{ кВ}}$; потери холостого хода $\Delta P_{XX2(3)} = 0{,}56$ кВт, потери короткого замыкания $\Delta P_{KZ2(3)} = 4{,}1$ кВт.
- **Л1, Л2, Л3**: кабельная линия. Каждая из них имеет свое удельное сопротивление r_{0Li} Ом/км, и протяженность l_i км.
- **Н1, Н2**: потребители электрической энергии. Каждый из них имеет мощность P_{Hi} кВт.
- **Q1, Q2, Q3**: силовые выключатели, сопротивление пренебрежимо мало.

Ваша задача — рассчитать **мощность генератора**, составленную из сумм мощностей нагрузок и потерь во всех элементах сетей.

Схема **управляется** тремя силовыми выключателям Q1, Q2, Q3. Задачу нужно рассмотреть для двух случаев:

1. все выключатели находятся в положении «вкл»;
2. выключатели Q1 и Q2 находятся в положении «вкл», а выключатель Q3 в положении «откл».

После расчета мощности генерации для обоих случаев, необходимо также сравнить **потери** в трансформаторе Т1 и линии Л1 и оценить **в процентах** величину снижения потерь в данных элементах до отключения выключателя Q3 и после отключения.

Ток в системе **трехфазный**, $\cos\varphi = 1$. Наличием **реактивной** мощности в системе можно **пренебречь**.

Входной формат: четыре строки. На первых трех строках приведены параметры **кабельных линий** в виде пары вещественных чисел, соответственно **удельное**

сопротивление (Ом/км) и **протяженность** (км). На четвертой строке через пробел приведены два вещественных числа, **мощности соответствующих потребителей** (кВт).

Выходной формат: четыре строки, на каждой из них по одному вещественному числу, соответственно мощность генератора (кВт) в первом и втором случае, процент снижения потерь в трансформаторе Т1 и линии Л1.

Ваше решение **принимается**, если каждая из этих величин отличаются от эталонной **не более**, чем **на единицу** (кВт или %).

Задача предполагает получение индивидуального набора данных и **ручной** ввод ответа.

Для решения этой задачи у вашей команды есть 20 попыток.

Алоритм решения

Общая мощность генератора складывается из суммы мощностей нагрузок и потерь на узлах цепи (здесь они считаются независимо, но подробнее этот момент затронут в конце описания решения).

$$P_r = \sum_{i=1}^{2} P_{H_i} + \sum_{j=1}^{n} \Delta P_j$$

Все потери можно считать как с помощью токов, так и мощностей, причем последний вариант предпочтительнее, т.к. позволяет миновать вычисление токов, которые при выведении конечной формулы сокращаются. Так потери на линиях считаются следующим образом:

$$\Delta P_{\text{Л}} = P^2 \cdot R / U_H^2; R = r_0 \cdot l / 1000$$

$$\Delta P_{\text{Л1}} = ((P_{H1} + P_{H2})/U_{\text{ВН}})^2 \cdot (r_{0_{\text{Л1}}} \cdot l_1/1000)$$
$$\Delta P_{\text{Л2}} = (P_{H1}/U_{HH})^2 \cdot (r_{0_{\text{Л2}}} \cdot l_2/1000)$$
$$\Delta P_{\text{Л3}} = (P_{H2}/U_{HH})^2 \cdot (r_{0_{\text{Л3}}} \cdot l_3/1000)$$

Потери на трансформаторах вычисляются исходя из соотношения действующей и номинальной нагрузок:

$$\Delta P_T = \Delta P_{xx} + \Delta P_{K3}(S/S_H)^2$$

$$\Delta P_{T1} = \Delta P_{xx} + \Delta P_{K3}((P_{H1} + P_{H2})/S_{HT1})^2$$
$$\Delta P_{T2} = \Delta P_{xx} + \Delta P_{K3}(P_{H1}/S_{HT2})^2$$
$$\Delta P_{T2} = \Delta P_{xx} + \Delta P_{K3}(P_{H2}/S_{HT3})^2$$

Для второго варианта цепи, когда отключена ветка Q3, достаточно перечитать потери Т1 и Л1, исходя из нагрузки, равной P_{H1}:

$$\Delta P'_{\text{Л1}} = (P_{H1}/U_{\text{вн}})^2 \cdot (r_{0_{\text{Л1}}} \cdot l/1000)$$

$$\Delta P'_{T1} = \Delta P_{xx} + \Delta P_{K3}(P_{H1}/S_{HT1})^2$$

$$\delta_{\text{Л1}} = (\Delta P_{\text{Л1}} - \Delta P'_{\text{Л1}})/\Delta P_{\text{Л1}} \cdot 100\%$$
$$\delta_{\text{Т1}} = (\Delta P_{\text{Т1}} - \Delta P'_{\text{Т1}})/\Delta P_{\text{Т1}} \cdot 100\%$$

Стоит заметить, что приведенное выше решение является авторским и представляет собой первичный расчет цепи, где потери считаются независимо. Конкретно в этой задаче и для данной цепи является допустимым произвести так называемый расчет установившегося режима, когда расчет потерь очередного узла опирается на потери в нижележащих узлах цепи (Л2 для Т2, Л3 для Т3; Л2, Л3, Т2 и Т3 для Л1 и все перечисленное для Т1). Так, альтернативное решение отличается лишь учетом мощностей и потерь.

Задача К5. Баланс в напряжении (13,27 баллов)

Подвижная материальная точка притянута **пружинами** к N неподвижных материальных точек-опор. Кроме координат неподвижных точек, известны длины в покое и коэффициенты жесткости каждой из пружин. Найдите **равновесное положение** подвижной точки.

Входной формат: на первой строке единственное целое число $\boldsymbol{N}$ (до 10), после этого N строк, на каждой из которых приведены параметры опоры и закрепленной на ней резинки, а именно **координаты** X и Y, **длина** пружины в состоянии **покоя** и коэффициент ее **жесткости**. Например,

2

3 5 3 2.5

0 0 7 0.54

Выходной формат: два вещественных числа через пробел, соответственно координаты X и Y равновесного положения.

Решение засчитывается, если длина вектора результирующей силы для указанных координат будет составлять не более, чем 10^{-4}.

Ограничение по времени выполнения программы — 6 секунд.

Для решения этой задачи у вашей команды есть 20 попыток.

Алгоритм решения

Краткая суть задачи — **оптимизация вектора равнодействующей** силы, то есть итерационное приближение подвижной точки к равновесному положению (где вектор равнодействующей силы максимально близок к нулю). Учитывая, что на вход передается максимум 10 точек, а на выполнение программы дается 6 секунд, для решения подойдет любой итерационный алгоритм. В авторском решении приводится простейший его вариант: посчитать равнодействующую, сдвинуться в ее сторону на небольшую долю (на всю длину равнодействующего вектора сдвигаться нельзя, алгоритм может зациклиться), пересчитать вектор и остановиться, когда его длина станет близка к нулю.

Вычисление равнодействующей опирается на закон Гука, где сила, действующая на точку со стороны пружины, зависит от растяжения Δx и коэффициента k, при этом опирается на вектор между искомой точкой M и неподвижной точкой S:

$$\overline{F_p} = \sum \overline{F_H} = \sum k\Delta x\overline{SM}$$

Авторское решение

```
from decimal import Decimal as deci
def hypot(x, y):
    return (x ** 2 + y ** 2) ** deci("0.5")
def force(x, y, springs):
    fx, fy = 0, 0
    for (xs, ys, l, k) in springs:
        d = hypot(x-xs, y-ys)
        ks = l / d
        fx += (x-xs) * (ks - 1) * k
        fy += (y-ys) * (ks - 1) * k
    return fx, fy
n = int(input())
springs = [[deci(x) for x in input().split()]
           for l in range(n)]
x, y, step = deci("0"), deci("0"), deci("0.1")
while True:
    fx, fy = force(x, y, springs)
    if hypot(fx, fy) <= 1e-6: break
    x += fx * step
    y += fy * step
print(x, y)
```

Бонусные задачи

Задача Б1. Перекладка сетей (5,53 баллов)

Прогресс не стоит на месте, в одном микрорайоне потребовалось **заменить** устаревшую систему ЛЭП на более надежную. Но энергетика связана с ресурсами, а ресурсы нужно расходовать бережно.

Вам известны **координаты** точек подключения на условной декартовой плоскости (обусловимся, что первая точка соответствует подстанции, питающей этот микрорайон). Необходимо **соединить все точки** в новую систему ЛЭП так, чтобы **минимизировать суммарную длину** проброшенных линий и только ее. Линии протягиваются между **парами** точек, к одной точке может быть подключено **несколько** линий. В этой задаче мы **не учитываем** потери и прочие параметры.

Входной формат: список пар целых чисел, где каждая пара — **координаты** соответствующей точки подключения. Например,

[(-22, 42), (-18, 39), (-64, -10), (-77, 38), (-17, 62), (68, -44), (-23, 21), (71, 74), (-98, 67)]

Выходной формат: список пар целых чисел, где каждая пара описывает отдельную линию между точками подключения, а число в этой паре означает **порядковый номер** точки во входном списке (**нумерация с 0**). Например,

[(0, 1), (1, 0), (2, 13), (1, 5)]

Возможно, ваше решение найдет другой, а то и более оптимальный ответ, чем авторское. В случае, если длина линий в полученной сети не больше, чем в «автор-

ской», на 10^{-5}, ваше решение будет зачтено при условии, что ваша сеть **связна и оптимальна**.

Ограничение по времени выполнения программы — 3 секунды.

Для решения этой задачи у вашей команды есть 20 попыток.

Алгоритм решения

Описание задачи подтакливает к тому, что программа должна сформировать полный граф между всеми перечисленными на входе узлами, где вес ребра — эвклидово расстояние между парой узлов, а затем построить на этом графе **минимальное остовное дерево**. Данная задача имеет несколько типовых решений, а именно **алгоритмы Прима и Крускала**, описание которых приведено в Интернете, из-за чего дублировать их не представляется разумным. Стоит лишь упомянуть, что ограничение в 3 секунды позволяет использовать любой из них, а авторское решение реализует алгоритм Прима с применением непересекающихся множеств для хранения текущей компоненты связности.

Авторское решение

```
from math import hypot

nodes = eval(input())
n = len(nodes)
p = [i for i in range(n)]

def dist(a, b):
    return hypot(a[0]-b[0], a[1]-b[1])

def get(a):
    if p[a] != a:
        p[a] = get(p[a])
    return p[a]

edges = []
for i in range(n):
    for j in range(i+1, n):
        edges.append((dist(nodes[i], nodes[j]), i, j))
edges.sort()

ans = []
for (_, a, b) in edges:
    da, db = get(a), get(b)
    if da != db:
        if p[da] != p[db]:
            p[da] = db
        ans.append((a, b))
print(ans)
```

Задача Б2. «Зделай» сервер (5,53 баллов)

С графическим интерфейсом все могут, а ты попробуй без!

Интернет стоит на серверах, а они в свою очередь держатся на умелых руках администраторов и разработчиков. Вам предстоит решить задачу обоих специали-

стов, создав и запустив **сервер для хранения произвольных текстовых данных** (key-value хранилище, по сути, dict с доступом по HTTP), по следующему протоколу (сервер должен быть доступен по любому адресу через порт 2020):

Запрос	**Ответ**
`GET /{key}`	код **200** с содержимым соответствующей ячейки в теле **ответа** если ключ отсутствует, код **404**
`POST /{key}` тело с текстом	код **202** с содержимым в формате «**[key] => [value]**» (например, «**1 => potato**») и записать текст в соответствующую ячейку по ключу
`DELETE /{key}`	код **204** с пустым телом и заголовком «**X-Value**», в котором находится удаленное содержимое (т. е. соответствующая ячейка удаляется) если ключ отсутствует, код **404**
`GIVE /` заголовок `"Gift"`	код **255** с сообщением «**Gift Accepted**» текстовое тело: «**Thanks for your [gift]!**», где [gift] — содержимое заголовка "Gift"

Эта задача выполняется **на виртуальной машине Ubuntu 18.04. ВМ** предоставлена внешним сервисом, интегрированным в Степик, и доступ к ней осуществляется через **удаленный текстовый терминал**. Нажатие на кнопку «Начать решение» или «Попробовать снова» **запустит таймер на 60 минут**, в течение которых вам будет доступен сеанс ВМ, терминал для которого можно открыть нажатием на «**Open Terminal**».

Вам необходимо программно **реализовать** HTTP-сервер внутри ВМ, используя исключительно доступные по умолчанию инструменты (apt не настроен, мы проверяли). Мы рекомендуем использовать **Python 3**, в его стандартную библиотеку уже включен **модуль** для создания веб-сервера. Но если вам удастся решить задачу на другом языке, об этом никто не узнает.

Чтобы задача была **зачтена**, вам необходимо **запустить** реализованный сервер и, **оставив его работать** (можно не закрывать терминал), нажать «Отправить». Проверочная система выполнит ряд **HTTP-запросов** по адресу виртуальной машины, чтобы проверить функциональность сервера. Если все ответы на запрос будут соответствовать **ожидаемым**, вы получите **полный балл**. Если один из ответов проваливает проверку, вы теряете попытку, при этом **ВМ и сервер продолжают работать**. Но напомним, что сеанс ВМ доступен лишь **в течение часа**, после чего ВМ будет уничтожена.

Главная проблема здесь — передача написанного снаружи **кода программы** внутрь ВМ. Обычная вставка текста в консоль у нас **не сработала** (хотя в Chromium можно сделать вставку через контекстное меню), но есть как минимум **два** известных способа решить проблему. Второй — использовать **scp** или **curl**, чтобы передать текст с другого сервера. Самый простой вариант — использовать **pastebin.com**. Загрузив на него текст программы, можно скачать его в файл внутри ВМ с помощью команды *curl https://pastebin.com/raw/ABCDEFG > server.py*. Вместо ABCDEFG будет идентификатор вашей «пасты». Будьте внимательны, выставляя настройки видимости.

Для решения этой задачи у вашей команды есть 20 попыток.

Решение

Суть задачи заключалась в работе с удаленной виртуальной машиной на базе операционной системы Ubuntu и сводилась к реализации программы-сервера и запуску ее внутри ВМ. При отправке решения система подключается к ВМ и выполняет тестирование по следующему **сценарию** (слева от стрелки HTTP-запрос, справа — ожидаемый код и дополнительные параметры ответа):

1. *(генерируется до 10 пар ключ-значение)*
2. GET /45 → 404
3. POST /32 + тело «0.232323» → 202 + тело «32 => 0.232323»
4. *(повторяем шаг 3 для всех пар)*
5. GET /32 → 200 + тело «0.232323»
6. *(повторяем шаг 5 для всех пар)*
7. DELETE /32 → 204 + X-Value: «0.232323»
8. GET /32 → 404
9. GIVE / + Gift: «jaba» → 255 (Gift Accepted) + тело: «Thanks for your jaba!»

Есть три типовые проблемы в решении этой задачи.

Первая — взаимодействие с **консолью** виртуальной машины, а именно как **передать** внутрь исходные коды серверной программы. Решение этой проблемы при отсутствии вставки в консоль обозначено прямым текстом в условии: передать исходники **через сеть**, загрузив их в сервис хранения текстов pastebin.com и скачать их на ВМ с помощью команды curl или wget.

Вторая — запуск самого сервера **встроенным инструментами** чистой Ubuntu. Опять же, рекомендуемое решение обозначено в условии — использовать **Python 3**. В его стандартную библиотеку входит модуль **http.server**, который хорошо документирован и позволяет решить поставленную задачу (код будет приведен ниже).

И наконец, **нестандартный** запрос GIVE и нестандартный ответ «Gift Accepted». В этом отношении стоит вспомнить, что в основе протокола HTTP лежит **обычный текст**, его спецификация гибкая, и хотя она оговаривает типовые запросы (так называемые глаголы) и коды ответа, допускается применять нестандартные параметры, что и подразумевается этой задачей.

Реализация хранилища по мнению автора проблемой не является, для этого в случае http.server достаточно поместить словарь в обработчик запросов.

Авторское решение

```
#!/usr/bin/env python3
from http.server import BaseHTTPRequestHandler, HTTPServer
import logging

class S(BaseHTTPRequestHandler):
    # ...

def run():
    logging.basicConfig(level=logging.INFO)
    server_address = ('', 2020)
    httpd = HTTPServer(server_address, S)
    try:
```

```
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()

if __name__ == '__main__':
    run()
class S(BaseHTTPRequestHandler):

    data = dict()

    def _set_response(self, code=200, reason=None, value=None):
        self.send_response(code, message=reason)
        self.send_header('Content-type', 'text/html')
        if value is not None:
            self.send_header('X-Value', value)
        self.end_headers()

    def do_GET(self):
        key = self.path[1:]
        value = self.data.get(key, None)
        exists = value is not None
        self._set_response(code=200 if exists else 404)
        if exists:
            self.wfile.write(value.encode('utf-8'))

    def do_POST(self):
        key = self.path[1:]
        content_length = int(self.headers['Content-Length'])
        value = self.rfile.read(content_length).decode('utf-8')
        self.data[key] = value
        self._set_response(code=202)
        self.wfile.write("{} => {}".format(key, value)
                         .encode('utf-8'))

    def do_DELETE(self):
        key = self.path[1:]
        Value = self.data.get(key, None)
        exists = value is not None
        if exists:
            del self.data[key]
            self._set_response(code=204, value=value)
        else: self._set_response(code=404)

    def do_GIVE(self):
        gift = self.headers['Gift']
        self._set_response(code=255, reason="Gift Accepted")
        self.wfile.write("Thanks for your {}!"
                         .format(gift).encode('utf-8'))
```

Задача Б3. Броуновский кузнечик (10,37 баллов)

Кузнечик делает от кустика **три** случайных прыжка на плоскости (обусловимся, что она декартова и бесконечная). Направление и дальность выбираются **случайно** и **равномерно**, предельная дальность прыжка известна.

Напишите программу, которая находит **математическое ожидание** расстояния кузнечика от кустика после трех прыжков.

Входной формат: единственное целое число, предельная дальность прыжка в сантиметрах.

Выходной формат: вещественное число, математическое ожидание расстояния от кустика в сантиметрах.

Решение **засчитывается**, если ответ, выдаваемый вашим решением, отличается от авторского **не более, чем на указанный в подзадаче допуск**.

Задача состоит из двух подзадач с разными ограничениями:

1. 10 секунд на выполнение, допуск 1% (при выполнении задачи команда получает 4/5 от максимального балла).
2. 1 секунда на выполнение. допуск 5% (при выполнении задачи команда получает 1/5 от максимального балла).

Для решения каждой подзадачи у вашей команды есть 20 попыток.

Решение

Математическая модель задачи имеет следующий вид. Даны три вектора со случайным углом и длиной в диапазоне от 0 до L (предельная дальность из входных данных). Необходимо найти математическое ожидание длины суммы этих трех векторов:

$$D = J_1 + J_2 + J_3 = \sum_{i=1}^{3} J_i;\ L_i \cdot \widehat{\phi_i},\ L_i \in [0; L],\ \phi_i \in [0; 2L)$$

Так, с одной стороны, мы замечаем, что ответ зависит линейно от предельной длины прыжка, но с другой стороны, аналитическое решение задачи будет выглядеть следующим образом (заметьте, ответ не зависит от угла первого прыжка, поэтому мы его опустили):

$$ans = M[|D|] = M[\left|\sum_{i=1}^{3} J_i\right|] = M[\left|\sum_{i=1}^{3} L_i \cdot \widehat{\phi_i}\right|] = L \cdot M[\left|\sum_{i=1}^{3} \frac{L_i}{L}\widehat{\phi_i}\right|] =$$

$$= L \cdot M[\left|\sum_{i=1}^{3} R_i \cdot \widehat{\phi_i}\right|],\ R_i \in [0; 1]$$

Очевидно, что вычисление этого интеграла не представляется разумным, поэтому предлагается решить эту задачу **методом Монте – Карло**, а именно многократным (порядка миллиона раз) вычислением величины, находящейся под пятикратным интегралом выше с последующим взятием среднего арифметического от результатов всех вычислений.

$$M[\left|\sum_{i=1}^{3} R_i \cdot \widehat{\phi_i}\right|] =$$

$$= \iiiint \int \sqrt{R_1 + R_2 cos\phi_2 + R_3 cos\phi_3)^2 + (R_2 sin\phi_2 + R_3 sin\phi_3)^2} dR_1 dR_2 dR_3 d\phi_2 d\phi_3$$

Учитывая линейную зависимость конечной величины от предельного расстояния, мы можем рассчитать ответ для $L = 1$ и использовать его в качестве константного множителя в решении второй задачи (а то и обеих, при достаточном числе итераций).

Авторское решение

```
from math import sin, cos, pi, hypot
from random import uniform
L = int(input())
result, shots = 0, 1000000
for _ in range(shots):
    x, y = 0, 0
    for _ in range(3):
        a, d = uniform(0, 2 * pi), uniform(0, L)
        x += d * cos(a)
        y += d * sin(a)
    d = hypot(x, y)
    result += d
print(result / shots)
```

Решение второй подзадачи (допустимо применить и для первой подзадачи):

```
print(int(input()) * 0.888991)
```