§2 Второй отборочный этап

Краткая справка

Включает задачи, для решения которых достаточно школьных знаний и умений программы 10-11 класса, навыков использовать школьные знания для решения новых задач. Включает 5 задач, целью которых является подготовка к финальному командному туру. Данный тур позволяет очертить область предметных знаний необходимую для участия в профиле. Методические рекомендации к данному туру позволяют очертить область знаний и навыков для самостоятельного изучения.

В таблице приведены номера задач второго этапа, элементы решения которых или полученное в результате решения понимание могут быть использованы для решения задач экспериментального тура в финале.

№ задачи 2-го этапа	Знания и навыки, на выявление и развитие которых направлена задача
2.1.1	Нацелена на развитие навыков по численным математическим методам, понимания принципов физических измерений, алгоритмического мышления Для решения задачи необходимы разделы информатики: программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения необходимы разделы математики: линейные функции.
2.1.2	Нацелена на развитие навыков работы с задачами, плохо поддающимися аналитическому решению, но которые возможно решать перебором. Для решения задачи необходимы разделы информатики: сложность алгоритмов, программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения необходимы следующие разделы и понятия математики: частичный порядок на множестве, экспоненциальный рост.
2.1.3	Нацелена на развитие навыков алгоритмического мышления, методов и способностей к аналитической работе с большими массивами данных, алгоритмическому мышлению и навыкам программной реализации решения математических задач, развитие понимания поведения суммы периодических функций. Для решения задачи необходимы разделы информатики: программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения задачи необходимы разделы математики: периодические функции, поиск оптимума функции, работа с числовыми рядами. дифференцирование.
2.1.4	Нацелена на развитие алгоритмического мышления, навыков работы с графами, использованию законов электрического тока в сложных схемах. Для решения задачи необходимы разделы информатики: численное представление графов, определение связности графа, программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения задачи необходимы следующие разделы математики: графы. Для решения задачи необходимы следующие разделы физики: закон Кирхгофа, закон Ома.

2.1.5	Нацелена на развитие навыков по численной работе со случайными величинами. Для решения задачи необходимы следующие разделы информатики: программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения задачи необходимы следующие разделы математики: теория вероятностей.
2.1.6.1	Задача направлена на развитие пространственного мышления и развитие навыков работы с большими массивами данных. Для решения необходимы следующие разделы информатики: программирование на языке Python, программная реализация алгоритмов решения математических задач. Для решения необходимы следующие разделы математики: сечения, планиметрия.
2.1.6.2	Задача направлена на развитие понимания физических принципов поведения сложных механизмов, навыки по анализу рядов данных, понимание принципов отказоустойчивости. Необходимые разделы информатики: программирование на языке Python, программная реализация алгоритмов решения математических задач. Необходимые разделы физики: работа электрического тока, принципы работы электрогенераторов.
2.1.7.1	Задача направлена на развитие пространственного мышления, навыков работы с задачами, решение которых требует анализа большого числа вариантов. Разделы математики, необходимые для решения задачи: теория вероятностей.
2.1.7.2	Задача направлена на развитие навыков программной реализацией математических и физических задач. Необходимые разделы информатики: программирование на любом языке. Необходимые разделы математики: теория вероятностей. Необходимые разделы физики: закон Ома.
2.1.7.3	Задача направлена на развитие навыков численного решения математических задач. Необходимые разделы математики: планиметрия, тригонометрия.
2.1.7.4	Задача направлена на развитие понимания поведения электрического тока, понятий управляющих воздействия и обратной связи принципов устройства электрических сетей. Необходимые разделы физики: закон Ома, закон Кирхгофа, мощность электрического тока.

Все задачи решались через платформу Stepik. В заданное время всем участникам открывался доступ к задаче (её условию и проверяющей системе). Во всех задачах, где это не указано отдельно, участники должны были написать программу на языке Python3 и загрузить её текст на сервер. На сервере выполнялись тесты загруженной программы: её ответы сверялись с ответами эталонного решения, составленного авторами задачи. Большинство задач требовало базовых навыков программирования на языке Python, поскольку это является необходимой частью финального задания, однако большинство задач специфичных навыков по информатике не требовали, и программирование требовалось только для программной реализации решения математической или физической задачи.

Ввод данных в программу осуществлялся через стандартный ввод (stdin).

Вывод данных осуществлялся через стандартный вывод (stdout).

Случайные числа, используемые для генерации тестов, являлись псевдослучайными. Все проверки программ всех участников производились на одинаковых данных.

Решение, загруженное любым участником команды, засчитывалось всем участникам команды.

Неправильные попытки решения учитывались только при одинаковых баллах двух команд. Такой ситуации не возникло.

Проходной балл тура составил 9,9 балла (команды с этими баллами и выше прошли на финал ОНТИ-2018 по этому направлению).

2.1 Задачи второго отборочного этапа

Задача 2.1.1 Тривиполяция (1 балл)

Условие

Даже если мы точно знаем, что, например, зависимость силы тока от напряжения при постоянном сопротивлении есть линейная функция I(U)=UR, если мы решим это проверить, в результате измерений мы получим нечто совсем другое.

У нас получится функция, которая будет состоять из намного меньшего количества точек — в ней будут только те, которые мы измерили (погрешностями измерений в нашем примере мы пренебрежём). И что бы мы ни делали, измерить мы можем только конечное число значений. Об остальных мы можем только догадываться.

А в общем случае мы можем даже не знать, значения какой именно функции мы измеряем.

Например, мы можем измерять зависимость силы тока от напряжения на стабилитроне. В этом случае часто даже те, кто знает, что такое стабилитрон и как выглядит его Вольт-Амперная характеристика, не возьмутся с большой точностью предугадать эту зависимость, особенно если исследуемый стабилитрон бракованный. О точном математическом выражении этой зависимости и вовсе не идёт речи, по крайней мере до тех пор, пока мы её досконально не изучим.

В таком случае значения, которые мы измерили — это просто набор точек, например, [(1,0;1,2),(2,0;1,3),(3,0;2,7)].

Другие значения в природе существуют, но мы их не измеряли. Мы можем только предположить, какими должны быть значения между измеренными нами точками. Важно, что мы в действительности не знаем, каковы они — мы строим предположения исходя из нашего понимания природы измеряемой величины и здравого смысла

Процедура «заполнения» промежутков между известными точками некоторой функции называется интерполяцией.

Существует множество различных техник интерполяции, и самая простая — линейная. В этом случае мы просто считаем, что функция между измеренными точками ведёт себя как линейная. По сути мы просто соединяем соседние точки на графике прямыми линиями.

В этой задаче вам нужно написать программу, которая для заданных точек некоторой функции проводит линейную интерполяцию и находит предполагаемые значения функции в десяти заданных точках. Одному значению абсциссы может соответствовать только одно значение ординаты.

Описание формата данных	Входные данные: кортеж из двух элементов. Первый элемент кортежа: массив кортежей *(абсцисса, ордината)* координат

	известных точек. Второй элемент кортежа — массив абсцисс точек, для которых значения нужно проинтерполировать. Выходные данные: массив ординат точек. Располагать их можно в любом порядке. Пример входных данных является ответом к примеру выходных данных.
Пример входных данных	([(0, 0.599944202170653), (1, 0.008386994101348244), (0.746745664788091, 0.6190939885482903), (0.6285586245444911, 0.6643252763930065), (0.9045389793236045, 0.9918643657478213), (0.4289354717049702 6, 0.898226324048021 5), (0.4152396899288065 5, 0.2115133838152604 1), (0.3987080301408766 3, 0.667795678097451 8), (0.9970287353084295, 0.6872690035523316), (0.0948335318160858 9, 0.479320386139811 3), (0.5076394320118685, 0.1835667078424845 2), (0.6138807341577017, 0.7035016147062624), (0.007496700168104109, 0.7998904675917943), (0.887961831271013, 0.27408297901234957), (0.5178297061405391, 0.9506460706772175), (0.0754267440010083 5, 0.299309871295927 1), (0.6142809975504684, 0.8891026774736874), (0.4152563484164909 5, 0.377515181827566 94), (0.2189728833368792 5, 0.4881040696975873 5), (0.4667259163982871, 0.234578241754141), (0.2864981333102281, 0.3802825750062660 5), (0.9860184557269054, 0.1838143732007103 7), (0.985072468169593 3, 0.3698409731039223), (0.2879453510659104 6, 0.2830780663947309), (0.5938885548514143, 0.5553553596293965)], [0.3544605759016911, 0.01748327611313305, 0.992349107338985, 0.3142275010714717, 0.727109158374748, 0.536193967048847, 0.8754630774589253, 0.3252787787406749, 0.28438589911681866, 0.05961154324966311])
Пример выходных данных	[0.5141087588120886, 0.7262987862171524, 0.4732889359168766, 0.3743651864336811, 0.6266090635684944, 0.8552038982627046, 0.304619197195068, 0.41275014774616336, 0.3836553022151459, 0.4158530409444131]
Ограничения вычислительных ресурсов	Время выполнения программы на сервере не более 27 секунд. Требуемая память на сервере не более 256 мегабайт.

Решение

Задача требует минимальных технических навыков в программировании и минимальных математических навыков. Ограничения вычислительных ресурсов несущественны.

Проверяющий код

```
import random
import fileinput
import functools
import math
import ast
class Dot():
    x = 0
    y = 0
    def __init__(self,x,y):
        self.x=x
        self.y=y
def genDot():
    x = random.random()
    y = random.random()
    return Dot(x,y)
def addDot(dots):
    a = genDot()
    for b in dots:
```

```
        if a.x == a.y:
            return addDot(dots)
    return a
def genTable():
    table = []
    table.append(Dot(0,random.random()))
    table.append(Dot(1,random.random()))
    for i in range(23):
        table.append(addDot(table))
    return table
def findPair(value, table):
    lower = table[0]
    upper = table[24]
    for i in table:
        if value > i.x and lower.x < i.x:
            lower = i
        if value < i.x and upper.x > i.x:
            upper = i
    return (lower,upper)
def printDot(dot):
    print("DOT:")
    print(dot.x)
    print(dot.y)
def interpolate(pair,value):
    (lower,upper)=pair
    k = (lower.y-upper.y)/(lower.x-upper.x)
    b = lower.y - k*lower.x
    return k*value + b
def genQuests():
    quest = []
    for i in range(10):
        quest.append(random.random())
    return quest
def findValue(table,value):
    pair = findPair(value,table)
    ans = interpolate(pair,value)
    return ans
def test():
    table = genTable()
    quests = genQuests()
    for i in quests:
        print("==================")
        print(findValue(table,i))
#test()
#This is a sample Code Challenge
#Learn more: https://stepik.org/lesson/9173
#Ask your questions via support@stepik.org
def generate():
    num_tests = 10
    tests = []
    for test in range(num_tests):
        table = genTable()
        tab = []
        for t in table:
            tab.append((t.x,t.y))
        quest = genQuests()
        test_case=str((tab,quest))+"\n"
        #test_case = "{} {}\n".format(a, b)
        tests.append(test_case)
        #print(test_case)
    return tests
def solve(dataset):
    (tab,quests) = ast.literal_eval(dataset)
    table = []
```

```
    tab.sort()
    #print(tab)
    for (a,b) in tab:
        table.append(Dot(a,b))
    answers = []
    # print(dataset)
    for q in quests:
        answers.append(findValue(table,q))
    #print(answers)
    return str(answers)
def check(reply, clue):
    r = ast.literal_eval(reply)
    a = ast.literal_eval(clue)
    r.sort()
    a.sort()
    que = True
    for i in range(len(a)):
        foo = (abs(r[i]-a[i]) < 1e-12)
        que = que and foo
    return que
def test2():
    tests = generate()
    for t in tests:
        solution = solve(t)
        q = check(solve(t),solution)
        print("~~~~~~~~~~~~~~~~~")
        print(q)
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.2. Инвестиционная траектория (16 баллов)

Условие

У вас есть генерирующая компания. Со своих электростанций в конце каждого месяца вы получаете прибыль, которую можете использовать для строительства новых электростанций. Для простоты будем считать, что электростанции строятся мгновенно и приносят полную прибыль уже в следующем месяце. Больше одной электростанции в месяц строить нельзя. Прибыльность каждой электростанции внутри задачи фиксирована, но генерируется случайно.

Вам доступны следующие виды электростанций:

- ЭС-1: стоимость 10 единиц, прибыль 0.95–1.05 единиц
- ЭС-2: стоимость 20 единиц, прибыль 2.2–2.4 единиц
- ЭС-3: стоимость 40 единиц, прибыль 5–5.4 единиц
- ЭС-4: стоимость 80 единиц, прибыль 11.2–12 единиц

Вам нужно написать программу, которая по входным параметрам электростанций определяет последовательность действий в каждом месяце так, чтобы к заданному месяцу получить максимальное количество наличных средств.

В начале у вас есть 10 единиц и постоянный источник прибыли в 1 единицу.

Описание формата данных	Входные данные: кортеж из двух элементов. Первый элемент — номер месяца, к которому нужно максимизировать очки, второй — массив ежемесячных прибылей с электростанций. Выходные данные: строка вида *"00010100200030001140 0"*, где каждый символ — это номер модели электростанции (ноль означает, что на этом ходу ничего строить не надо). Первый месяц — слева. Максимальное возможное значение месяца — 80. Пример входных данных является ответом к примеру выходных данных.
Пример входных данных	(31, [1.0261847744960386, 2.2894769356564626, 5.281709890879855, 11.352694895047318])
Пример выходных данных	10000010000002000200200000000000
Ограничения вычислительных ресурсов	Для получения 12 баллов: Время выполнения программы на сервере не более 27 секунд. Требуемая память на сервере не более 256 мегабайт. Для получения 16 баллов: Время выполнения программы на сервере не более 4 секунд. Требуемая память на сервере не более 256 мегабайт.

Решение

Предлагаемый авторами метод решения задачи — перебор с обрезанием дерева решений.

Пример решения позволяет получить только 12 баллов. Это связано с неоптимальной реализацией решения на языке Python. При использовании других языков программирования авторам удавалось добиться времени решения менее одной секунды в сравнимом вычислительном окружении.

Допустима погрешность в пределах 10^{-12}.

Проверяющий код

```
import random
import math
import time
random.seed(None)
maxtime = 0
winner = False
#mode = "Work"
#mode = "Test"
mode = "Task"
def income(fleet,code):
    if code.isalpha:
        pass
    else: print("I WANT CHAR!!!")
    if code == '0':
        return fleet[0].income
    if code == '1':
```

```
        return fleet[1].income
    if code == '2':
        return fleet[2].income
    if code == '3':
        return fleet[3].income
    if code == '4':
        return fleet[4].income
    print("BAD CODE INCOME " + code)
    raise Exception("INCOME " + str(code))
    return 0
def cost(fleet,code):
    if code.isalpha:
        pass
    else: print("I WANT CHAR!!!")
    if code == '0':
        return fleet[0].price
    if code == '1':
        return fleet[1].price
    if code == '2':
        return fleet[2].price
    if code == '3':
        return fleet[3].price
    if code == '4':
        return fleet[4].price
    print("BAD CODE cost " + code)
    raise Exception("COST " + str(code))
    return 0
class P():
    price = 0
    income = 0
    def __init__ ( self,c,i ):
        self.price = c
        self.income = i
vocab = list("01234")
class E():
    capital = 0
    income = 0
    def __init__ (self,c,i):
        self.capital = c
        self.income = i
def eco(fleet,evo):
    cap = 10
    inc = 1
    list(evo)
    for e in evo:
        cap += inc
        cap -= cost(fleet,e)
        inc += income(fleet,e)
    return E(cap,inc)
def validate(fleet,evo):
    cap = 10
    inc = 1
    list(evo)
    for e in evo:
        cap += inc
        cap -= cost(fleet,e)
        inc += income(fleet,e)
        if cap <= (-0.01):
            return False
    return True
def horizon(fleet,p):
    x = income(fleet,p)
    if x == 0:
        return 0
```

```
    else:
        return math.ceil(cost(fleet,p)/x)
def addable(fleet,evo,left,p):
    z = eco(fleet,evo)
    x = ( z.capital >= cost(fleet,p))
    y = ( left >= horizon(fleet,p))
    if z.income >= fleet[4].price:
        global winner
        winner = True
    return x and y
def homo(li,el):
    return(li+el)
def spread1(fleet,left,evo):
    ta = []
    ee = []
    q = []
    for x in vocab:
        if addable(fleet,evo,left,x):
            ta.append(x)
    for y in ta:
        q = homo(evo,[y])
        ee += [q]
    return ee
def spreadAll(fleet,left,evos):
    newEvos = []
    for e in evos:
        sheet = spread1(fleet,left,e)
        newEvos = newEvos + sheet
    return newEvos
def live(ecos,eco):
    alive = True
    for e in ecos:
        alive = alive and ( eco.capital >= e.capital or eco.income >= e.income )
    return alive
def diminish(fleet,evos):
    ecos = []
    ret = [[]]
    for e in evos:
        ecos = ecos + [eco(fleet,e)]
    for e in evos:
        if (live(ecos,eco(fleet,e))):
            ret += [e]
        else:
            pass
    del(ret[0])
    return ret
def step(fleet,left,evos):
    spr = spreadAll(fleet,left,evos)
    dim = diminish(fleet,spr)
    return dim
def patchWinners(fleet,evos,left):
    zeroes = horizon(fleet,'4')
    fours = left - zeroes - 1
    f=(repeat('4',fours))
    z=(repeat('0',zeroes))
    return evos + repeat('4',fours) + repeat('0',zeroes)
def run(fleet,count):
    counter = count
    evos = [[]]
    for i in range(int(count),int(0),int(-1)):
        evos = step(fleet,i,evos)
        if winner == True:
            out = []
            for w in evos:
```

```
                out.append(patchWinners(fleet,w,i))
            return out
    return evos
def repeat(x,count):
    out = []
    for i in range(count):
        out.append(x)
    return out
def best(fleet,evos):
    best = None
    bestCap = 0
    for e in evos:
        if eco(fleet,e).capital > bestCap:
            bestCap = eco(fleet,e).capital
            best=e
    return best
def generate():
    num_tests = 20
    tests = []
    for test in range(num_tests):
        if test == (num_tests-1):
            steps = 75
        else:
            steps = random.randint(20+2*test, 35+2*test)
        scale1 = 1*random.uniform(0.95,1.05)
        scale2 = 2*random.uniform(1.1,1.2)
        scale3 = 4*random.uniform(1.25,1.35)
        scale4 = 8*random.uniform(1.4,1.5)
        case = (steps,[scale1,scale2,scale3,scale4])
        tests.append((str(case)+"\n",case))
    return tests
def mkFleet(values):
    fleet = []
    fleet.append(P(0,0))
    for i in range(0,4):
        fleet.append(P(10*(2**i),values[i]))
    return(fleet)
def printFleet(fleet):
    for i in fleet:
        print(i.income)
def solve(dataset):
    global winner
    winner = False
    started = time.time()
    foo = eval(dataset)
    steps = foo[0]
    fleet = mkFleet(foo[1])
    answer=''.join(best(fleet,list(run(fleet,steps))))
    if mode=="Test":
        ran = (time.time()-started)
        global maxtime
        if maxtime < ran:
            maxtime = ran
    return str(answer)
def check(reply, clue):
    (steps,fl) = clue
    my = solve(str(clue))
    fleet = mkFleet(fl)
    resultMe = eco(fleet,list(my)).capital
    resultThem = eco(fleet,list(reply)).capital
    return len(my) <= len(reply) and (validate(fleet,reply)) and ( resultThem >=
resultMe - 0.000001 )
def test():
    while True:
```

```
        x = generate()
        for (y,z) in x:
            _ = solve(y)
        print(maxtime)
def work():
    print(solve(input()))
if mode=="Test":
    test()
if mode=="Work":
    work()
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.3. Как бы 50 Герц (10 баллов)

Условие

У вас имеется 10 генераторов сигналов с частотами, «близкими» к 50 Гц, каждый из которых определяется шестью параметрами $(\alpha, \beta, \gamma, \delta, \epsilon, \zeta)$:

$F_k(t) = (\alpha - \beta)sin\big(100\pi t + \gamma + \delta cos(\epsilon t + \zeta)\big), k \in [1{,}10]$

Каждый параметр может принимать значение от 0 до 1.

Нужно выбрать такие 5 генераторов, чтобы сумма их сигналов была наиболее похожа на синусоиду с частотой 50 Герц.

«Похожесть» некоторой функции на синусоиду мы будем измерять как сумму квадратов разности между нормированной изучаемой функцией f(t)

и синусоидой во всех интересующих нас моментах времени. Это означает следующее. Для начала введём «эталонный сигнал» — функцию $sin(100\pi t)$. В каждый интересующий нас момент времени мы будем вычислять величину $\big(Af(t_i) - sin(100\pi t_i)\big)^2$ (параметр A мы вычислим чуть позже). Затем эту величину для всех интересующих нас моментов времени мы сложим и получим $R(f, A) = \sum_i \big(Af(t_i) - sin(100\pi t_i)\big)^2$. Параметр A мы выбираем так, чтобы значение $R(f, A)$ было минимальным. Результат мы можем называть невязкой изучаемой функции к эталонному сигналу, и она будет обратной величиной к «похожести» — чем меньше функция похожа на синусоиду, тем невязка будет больше.

Точки для измерения расположены через каждую миллисекунду в течение одной секунды, начиная с момента 0. Момент времени «1 секунда с момента 0» не включаем.

Описание формата данных	Входные данные: массив кортежей, каждый из которых состоит из шести элементов: $(\alpha, \beta, \gamma, \delta, \epsilon, \zeta)$ Выходные данные: массив номеров генераторов, которые нужно включить (первый имеет номер "1", а не "0"). Порядок не важен. Пример входных данных является ответом к примеру выходных данных.
Пример входных данных	[(0.23985341450914743, 0.6299947148470596, 0.0355262962341194, 0.38331715711690806, 0.13830852620342016, 0.8506333294789787), (0.9972622635376275, 0.9687087928595056, 0.19693333116256784, 0.8404378530812737, 0.12801110106173041, 0.09275115115494803), (0.502917271604434, 0.565132701368741, 0.9452803976225446, 0.9149490570861714, 0.6828897771655124, 0.04449313891272633), (0.02294774152515444, 0.3315136155281351, 0.9493743508159547, 0.08084908626675924, 0.5529423951929902, 0.42500789277537476),

	(0.8478563844420818, 0.8014112998510504, 0.9624985911905132, 0.539917427488718, 0.5141874085563737, 0.1650646506747525), (0.7778838605503454, 0.744603817300623, 0.9501667030820351, 0.687354477988807, 0.20494764532580256, 0.2516549905578832), (0.6498360843839662, 0.06387737665456894, 0.5279190392631995, 0.48526244008371144, 0.48427422762913974, 0.8326376977220707), (0.3678561413474899, 0.31582875754291795, 0.4286040988098334, 0.21049707996598133, 0.41679220089706104, 0.41233432025868755), (0.027878701008625884, 0.9852770472762148, 0.2966846088179437, 0.5151134427747026, 0.3069188793704395, 0.052479615051166384), (0.32298193516241336, 0.6925431249904662, 0.24845981001237882, 0.4438381119862459, 0.622664132977857, 0.3676994562726448)]
Пример выходных данных	[1, 2, 5, 6, 8]
Ограничения вычислительных ресурсов	Время выполнения программы на сервере не более 15 секунд. Требуемая память на сервере не более 256 мегабайт.

Решение

Предлагаемый авторами метод решения задачи: полный перебор комбинаций функций, с вычислением невязки для каждого набора. Параметр A можно найти путём поиска минимума описанной функции через производную.

Проверяющий код

```
import random
import ast
from math import pi, sin, cos, sqrt, ceil, log
def makeVector():
   a = random.random()
   b = random.random()
   d = random.random()
   e = random.random()
   f = random.random()
   g = random.random()
   return (a,b,d,e,f,g)
def PROBLEM():
   vectors = []
   for i in range(10):
       vectors.append(makeVector())
   return vectors
def makeValues(params):
   val = []
   for t in range(1000):
       val.append(F(v,t/1000))
   return(val)
def F(params,t):
   (a,b,d,e,f,g) = params
   return (a-b)*sin(100*pi*t+d+e*cos(f*t+g))
def f(t):
   return sin(50*2*pi*t)
def aa(vectors,ixes):
    sum1 = 0
    sum2 = 0
    for t in range(1000):
        ff = 0
        for i in ixes:
            ff += F(vectors[i],t/1000)
        sum1 += ff**2
        sum2 += ff * f(t/1000)
```

```
    return sum2 / sum1
def RR(vectors,ixes):
   r = 0
   A = aa(vectors,ixes)
   for t in range(1000):
       ff = 0
       for i in ixes:
           ff += F(vectors[i],t/1000)
       r += (ff * A - f(t/1000))**2
   return sqrt(r/1000)
def bitPresent(bunch,ix):
   return(( bunch & ( 0b1 << ix )) != 0 )
def bits(bunch):
   bs = []
   size = 1+ceil ( log (bunch+1,2))
   for i in range(size):
       if (bitPresent(bunch,i)):
           bs.append(i)
   return bs
def solution(vectors):
   best = float("inf")
   bests = []
   for i in range(2**10):
       bs = bits(i)
       r = 0
       if len(bs) == 5:
           r += RR(vectors,bs)
           if r < best:
               best = r
               bests = bs
   out = []
   for b in bests:
       out.append(b+1)
   return out
def solve(dataset):
   a = ast.literal_eval(dataset)
   return str(solution(a))
def generate():
   num_tests = 10
   tests = []
   for i in range(num_tests):
       problem = PROBLEM()
       test = str(problem) + "\n"
       tests.append(test)
   return tests
def check(reply,clue):
   r = ast.literal_eval(reply)
   c = ast.literal_eval(clue)
   r.sort
   c.sort
   return r == c
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.4. Одеревенение (21 балл)

Условие

У вас имеется электросеть, в которой каждый узел может быть генератором или потребителем, в зависимости только от его желания.

Сеть состоит из 20 узлов и 23 линий между ними, сеть связна. Для каждого узла известна потребляемый(генерируемый) ток. Сумма потребления и генерации в системе, естественно, равна нулю. Для каждой линии известно её электрическое сопротивление.

Сеть немного избыточна: линий в ней больше, чем минимально необходимо для эффективного снабжения потребителей.

Вам нужно написать программу, выбирающую линии, которые необходимо отключить, с тем, чтобы:

1. Сеть была связной — существовал путь от любого узла к любому другому.
2. Сеть была минимальной — ещё одно отключение любой линии нарушит связность.
3. Электрические потери в сети были бы минимальными.
4.

Описание формата данных	Входные данные: кортеж из двух элементов: 1. Массив описаний узлов. Элемент массива — кортеж: (номер_узла, потребление_узла) 2. Массив описаний линий. Элемент массива — кортеж: (номер_узла_1, номер_узла_2, сопротивление) Выходные данные: массив описаний линий, которые нужно отключить. Элемент массива — кортеж (номер_узла_1, номер_узла_2) Пример входных данных является ответом к примеру выходных данных.
Пример входных данных	([(1, 6.457618749666315), (2, 2.34916382116208), (3, -0.740167552236842), (4, 6.234382106087031), (5, -3.1316369637865535), (6, 0.9897752777385684), (7, 0.24038606531962126), (8, 7.642214424430149), (9, -7.322509670549964), (10, 4.057923108024571), (11, -7.583180627399665), (12, 1.9181006271241192), (13, -4.997537117928276), (14, 1.7223675317322198), (15, 0.8690319317796793), (16, -7.221746031626173), (17, -0.19860624441088204), (18, 6.834536365495783), (19, -8.245282536012065), (20, 0.125166735390279)], [(1, 2, 1.3681083607992897), (1, 3, 1.6682686260929585), (2, 4, 1.645502600553864), (4, 5, 1.0229961608972262), (1, 6, 1.417562497800132), (3, 7, 1.2199285414300838), (2, 8, 1.704612736129983), (1, 9, 1.551127972497232), (9, 10, 1.6690915938314124), (10, 11, 1.236229798375231), (2, 12, 1.5423963597267267), (5, 13, 1.23329767231035), (6, 14, 1.5786233516601298), (9, 15, 1.3734895845961304), (2, 16, 1.9616574611483824), (7, 17, 1.005486137827392), (12, 18, 1.0784533475154987), (13, 19, 1.5823606002884856), (6, 20, 1.4396394718557581), (1, 17, 1.2200869423484038), (18, 15, 1.3275950982872524), (16, 13, 1.6224554572575487), (4, 14, 1.991519620183436)])
Пример выходных данных	[(1, 2), (1, 6), (2, 4), (7, 17)]
Ограничения вычислительных ресурсов	Для получения 18 баллов: Время выполнения программы на сервере не более 5 секунд. Требуемая память на сервере не более 256 мегабайт. Для получения 21 балла: Время выполнения программы на сервере не более 1 секунды. Требуемая память на сервере не более 256 мегабайт.

Решение

Авторами использовался следующий алгоритм решения.

1. Выкинуть из исходного графа 4 ребра

2. Проверить граф на связность. Если граф связен, продолжаем решение. Если нет — возвращаемся к п.1 и пробуем другие рёбра
3. Выбираем произвольную вершину в качестве корня дерева.
4. Для каждого листа дерева:
    1. Ток на ребре, ведущем к этому листу, равен потреблению этого листа.
    2. Из потребления вышестоящей вершины дерева нужно вычесть потребление листа.
    3. Лист из дерева можно удалить (потребление ребра нужно сохранить).
5. Если листья кончились, то мы нашли токи через все рёбра дерева (подграфа), идём дальше. Если нет — повторяем п.4.
6. Вычисляем потери на каждом ребре графа по формуле $P=I^2R$.
7. Суммируем потери.
8. Возвращаемся к п.1 до тех пор, пока не переберём все возможные остовные деревья графа. (Использовать более быстрый способ поиска остовных деревьев малоосмысленно, так как в худшем случае деревьями будут почти все подграфы).

Проверяющий код

```
import random
from copy import copy, deepcopy
import ast
random.seed(1)
def genTree(size):
    tree = []
    for i in range(1,size):
        to = random.randint(1,i)
        tree.append((to,i+1))
    return tree
def graphSize(graph):
    top = 0
    for (a,b) in graph:
        if a > top:
            top = a
        if b > top:
            top = b
    return (max(a,b))
def patchTree(tree,patches):
    graph = tree
    size = graphSize(tree)
    for i in range (patches):
        (a,b) = (0,0)
        while ( a == b or (a,b) in tree or (b,a) in tree ):
            a = random.randint(1,size)
            b = random.randint(1,size)
        graph.append((a,b))
    return graph
def addLosses(graph):
    new = []
    for (a,b) in graph:
        weight = random.uniform(1,2)
        new.append((a,b,weight))
    return new
def makeGraph(base,plus):
    return addLosses(patchTree(genTree(base),plus))
def makeNodes(size):
    nodes = []
    sum = 0
    for i in range (size):
        x = random.uniform(-10,10)
        sum += x
        nodes.append((i+1,x))
    sum /= size
```

```
    zeroed = []
    for (n,x) in nodes:
        zeroed.append((n,x-sum))
    return zeroed
def makeTask(base,plus):
    nodes = makeNodes(base)
    graph = makeGraph(base,plus)
    return ((nodes,graph))
def makeTable(nodes,graph):
    links = [[ None for i in range(len(nodes))] for j in range(len(nodes))]
    for (a,b,c) in graph:
        links[a-1][b-1] = c
        links[b-1][a-1] = c
    return links
def getType(table,dels):
    ty = "tree"
    every = range(len(table))
    visited = []
    shortList = [1]
    for step in every:
        seen = []
        for s in shortList:
            for c in every:
                if table[s-1][c] == None or (c+1) in visited or (s,c+1) in dels
or (c+1,s) in dels:
                    pass
                else:
                    #print(("SAW",c+1))
                    seen.append(c+1)
        stuffed = []
        #print(("seen",seen))
        for i in seen:
            if i in stuffed or i in shortList:
                #print(("CYCLE",i))
                ty = "cycled"
            else:
                #print(("stuffing",i))
                stuffed.append(i)
        for x in shortList:
            visited.append(x)
        shortList = stuffed.copy()
        #print(("ShortList",shortList))
        #print(("Visited",visited))
        #print("~~~~~~~~~~~~~~~")
    if len(visited) < len(table):
        return "unconnected"
    else:
        return ty
#    seen = []
#    for i in range(len(table)): #over all nodes
#        for j in range(i+1,len(table)):
#            if table[i][j] == None or (i+1,j+1) in dels or (j+1,i+1) in dels:
#                #print((i+1,j+1,"ignored"))
#                pass
#            else:
#                #print((i+1,j+1,"SPOTTED"))
#                if j+1 in seen and i+1 in seen:
#                    ty = "cycle" # Cycle detected
#                if not (j+1 in seen) or not (i+1 in seen):
#                    ty = "unconnected"
#                if not (i+1 in seen):
#                    seen.append(i+1)
#                if not (j+1 in seen):
#                    seen.append(j+1)
```

```
    #print(("SEEN",seen))
    return ty
def getType_(table):
    return getType(table,[])
def nubby(listoflists):
    out = []
    for l in listoflists:
        killme = l.copy()
        killme.sort()
        if not ( killme in out ):
            out.append(killme)
    return out
def spread1(li,elems):
    out = []
    for e in elems:
        if e in li:
            pass
        else:
            killme = li.copy()
            killme.append(e)
            out.append(killme)
    return nubby(out)
def spreadX(lis,elems):
    out = []
    for i in lis:
        killme = spread1(i,elems)
        out.append(killme)
    return nubby([ item for sub in out for item in sub ])
def makeDels(graph,count):
    li = []
    for l in graph:
        li.append((l[0],l[1]))
    out = []
    for e in li:
        out.append([e])
    for i in range(count-1):
        new = spreadX(out,li)
        out = new
    return out
def isLeaf(node,table,dels):
    edges = 0
    uplink = None
    for i in range(len(table)):
        if table[i][node-1] == None or (i+1,node) in dels or (node,i+1) in dels:
            pass
        else:
            edges += 1
            uplink = i+1
    if edges == 1:
        return uplink
    else:
        return None
def clearNodes(nodes,kill):
    alive = []
    mapping = []
    #nodes.sort()
    shift = 0
    for (a,b) in nodes:
        if a in kill:
            shift += 1
        else:
            alive.append((a-shift,b))
            mapping.append((a-shift,a))
    return(alive,mapping)
```

```
def mapMe(mapping,me):
    for (new,old) in mapping:
        if me == old:
            return new
    return me
def unMapMe(mapping,me):
    for (new,old) in mapping:
        if me == new:
            return old
    return me
def unMapEdges(mapping,edges):
    new = []
    for (a,b) in edges:
        aa = unMapMe(mapping,a)
        bb = unMapMe(mapping,b)
        #print(aa,bb)
        new.append((aa,bb))
    return new
def clearEdges(edges,kill,mapping):
    alive=[]
    for (a,b,c) in edges:
        if (a,b) in kill or (b,a) in kill:
            pass
        else:
            alive.append((mapMe(mapping,a),mapMe(mapping,b),c))
    return(alive)
def shrink(nodes_,graph_):
    nodes = deepcopy(nodes_)
    graph = deepcopy(graph_)
    table = makeTable(nodes,graph)
    killNodes = []
    killEdges = []
    losses = 0
    for whatever in range(len(table)):
        for i in range(len(table)):
            leaf = isLeaf(i+1,table,[])
            if leaf == None:
                pass
            else:
                killNodes.append(i+1)
                killEdges.append((i+1,leaf))
                killEdges.append((leaf,i+1))
                (a,b) = nodes[leaf-1]
                (c,d) = nodes[i]
                losses += d**2 * table[i][leaf-1]
                #print((i+1,leaf,"loss",losses))
                nodes[leaf-1] = (a,b+d)
                table[i][leaf-1] = None
                table[leaf-1][i] = None
    (clN,mapp) = clearNodes(nodes,killNodes)
    clE = clearEdges(graph,killEdges,mapp)
    return(losses,clN,clE,mapp)
def runTree(nodes_,table_,dels):
    table = deepcopy(table_)
    nodes = deepcopy(nodes_)
    losses = 0
    #print(("NODES",nodes))
    #print(("TABLE",table))
    #print(("DELS",dels))
    for whatever in range(len(table)):
        for i in range(len(table)):
            leaf = isLeaf(i+1,table,dels)
            if leaf == None:
                pass
```

```
            else:
                (a,b) = nodes[leaf-1]
                (c,d) = nodes[i]
                losses += d**2 * table[i][leaf-1]
                #print((i+1,leaf,"loss",losses))
                nodes[leaf-1] = (a,b+d)
                table[i][leaf-1] = None
                table[leaf-1][i] = None
    return losses
def solution(nodes_,edges_):
    (loss,nodes,edges,mapp) = shrink(nodes_,edges_)
    #loss = 0
    #nodes = deepcopy(nodes_)
    #edges = deepcopy(edges_)
    table = makeTable(nodes,edges)
    dels = makeDels(edges,len(edges)-len(nodes)+1)
    best = float("inf")
    guess = None
    for d in dels:
        #print(d)
        if (getType(table,d)) == "tree":
            que = runTree(nodes,table,d)
            if que < best:
                best = que
                guess = d
    return(unMapEdges(mapp,guess),best+loss)
    #return(guess,best+loss)
#(a,b) = makeTask(20,5)
#print(solution(a,b))
def generate():
    num_tests = 10
    tests = []
    for test in range(num_tests):
        task = makeTask(20,4)
        test_case = str(task)+"\n"
        tests.append((test_case,task))
    return tests
def solve(dataset):
    (nodes,edges) = ast.literal_eval(dataset)
    (remove,score) = solution(nodes,edges)
    return str(remove)
def check(reply, clue):
    (nodes,edges) = clue
    table = makeTable(nodes,edges)
    (my_remove,my_score) = solution(nodes,edges)
    remove = ast.literal_eval(reply)
    if getType(table,remove) == "tree":
        return runTree(nodes,table,remove) + 0.01 > my_score
    else:
        return False
#x = generate()
#(data,clue) = x[0]
#print(check(solve(data),clue))
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.5. Невероятная формулища (9 баллов)

Условие

Дана формула от восьми рациональных параметров $[a_0, .., a_7]$ и восьми рациональных случайных переменных $[x_0, .., x_7]$:

$$7a_0x_0 + \frac{148+x_7^3+22a_7}{3+0{,}93a_5} + a_1x_1^2 + \frac{a_1x_0x_1^3}{(0{,}1+x_2+a_5)^3} + \frac{(a_4a_5a_6)^2}{1+x_4+x_5+x_6} + \frac{(1+a_2)(1+x_2)(3+a_0x_2x_1)}{(42+x_2-x_1-x_0+a_7)(29-a_2-x_0)} + \sqrt{2^\pi - \frac{(x_1+x_2+x_0)}{3}} + a_0$$

Функции распределения случайных переменных дискретны. Их значения находятся в диапазоне от 0 до 10. Значения параметров также лежат в этом же диапазоне.

Вам нужно написать программу, которая по заданным значениям параметров и распределениям случайных переменных вычисляет число со следующими свойствами:

1. Это число целое.
2. Вероятность, того, что оно будет меньше, чем значение формулы, меньше, чем 50%.
3. Это минимальное из всех таких чисел.

Описание формата данных	Входные данные представлены в виде кортежа из двух элементов. Первый элемент — массив параметров $[a_0, .., a_7]$, второй — массив функций плотностей вероятности для каждой из величин $[x_0, .., x_7]$. Функции плотностей верояностей задаются так: это массив кортежей, первый элемент которого — значение случайной переменной, а второй — вероятность реализации этого значения. Выходные данные: целое число Обратите внимание, что пример входных данных не является ответом к примеру выходных данных!
Пример входных данных	([0.7310975430980204, 2.6571667962574907, 7.883201283868072, 3.4942875826161526, 4.2701527354698054, 1.407726988019492, 5.719482431132753, 0.9552188099243997], [[(5.798151945015613, 0.05105874720117737), (0.7617801350624043, 0.11562410620955695), (4.770577183812696, 0.13328294372332614), (1.1791546219122806, 0.1994748414206206), (5.116204108018508, 0.16758687740426434), (2.3715852863868503, 0.3329724840410547)], [(6.2517271800340835, 0.13827008266807558), (9.178436929439522, 0.2354043009154905), (3.3809924715652717, 0.1794324430385729), (5.560565845869336, 0.14820939816939205), (7.870746365454627, 0.08324591242966571), (2.152078646582715, 0.2154378627788034)], [(1.2080924395683557, 0.024673412412551228), (5.277858805530602, 0.08564143257957552), (1.1672947038609482, 0.25032417841681864), (3.161829341892507, 0.28838303521096653), (5.473228419407602, 0.19917371361373112), (1.9172548389881394, 0.15180422776635705)], [(9.055988016168287, 0.145086700867648), (1.5034578539658217, 0.24929307797876804), (3.488046842672148, 0.03407669257833888), (6.9674998756437905, 0.13720855023460976), (7.363525123262798, 0.23963283394340198), (3.285730533093031, 0.19470214439723327)], [(7.987129209926406, 0.21212170327732044), (1.469352092289995, 0.09703251986811512), (1.7417889708470813, 0.12957152554041798), (4.968010254127719, 0.20278289949914424), (3.183240185206744, 0.022324117942921197), (5.6316773461770175, 0.336167233872081)], [(1.7977131815774128, 0.12121404365141454), (7.5295460418904465, 0.19522170065335345), (7.978397905564866, 0.0801866393916841), (7.386311684998653, 0.2226124077412124), (3.1956985407579284, 0.21031435527653006), (9.735686308891179, 0.17045085328580534)], [(6.2086740367379765, 0.052271229095798684), (8.777432213320896, 0.26609259787125), (7.769576754539371, 0.1895503507908673), (9.123852015598189, 0.15689020865286335), (1.1133950258168257, 0.17320963936534145), (8.05798129294077, 0.16198597422387911)], [(4.310436223332143, 0.1750352603532333), (3.3794743945732697, 0.19166716236324008), (0.2715683576349248, 0.20978214620843333), (6.108694092322313, 0.03789652054629391), (0.4282190555235177, 0.1544386765503543), (4.07691330908113, 0.23118023397844512)]])
Пример выходных данных	583

Ограничения вычислительных ресурсов	Для получения 7 баллов: Время выполнения программы на сервере не более 10 секунд. Требуемая память на сервере не более 256 мегабайт. Для получения 9 баллов: Время выполнения программы на сервере не более 1 секунды. Требуемая память на сервере не более 128 мегабайт.

Решение

Фактически в задаче нужно найти медиану распределения случайной величины

Для устранения погрешностей расстояние от точного решения до ближайшего целого числа составляет не менее 0,2.

Одна из случайных величин в формуле отсутствует.

Метод решения — вычисление функции плотности вероятностей «в лоб», перебором всех исходов вектора случайных величин.

Формулу возможно разбить на сумму нескольких независимых случайных величин.

Проверяющий код

```
import random
from copy import deepcopy, copy
from functools import partial
import math
from time import time
def funDistr(fun,rvars,ixes):
    combos = [([],1)]
    for ix in ixes:
        newc = []
        for (var,prb) in combos:
            comb = []
            for (val,prob) in rvars[ix]:
                v = copy(var)
                v.append(val)
                item = (v,prb*prob)
                comb.append(item)
            newc += comb
        combos = copy(newc)
    out = []
    for (c,p) in combos:
        out.append((fun(c),p))
    return out
def f0(xs):
    return xs[0]+xs[1]+xs[2]
def tr(n):
    out = []
    for i in range(n):
        out.append((0.1*i,1/n))
    return out
testLen = 6
testVars = [ tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
           , tr(testLen)
```

```
          ]
testFix = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8]
def f1(fix,x):
    a = fix
    y1 = a[0]*x[0]*7 + a[1]*(x[1]**2) +
(1+a[2])*(1+x[2])*(3+a[0]*x[2]*x[1])/((42+x[2]-x[1]-x[0]+a[7])*(29-a[2]-x[0]))
    y2 = a[0] + a[1]*x[0]*(x[1]**3) / ((0.1+x[2]+a[5])**3)
    y3 = 2**math.pi-(x[1]+x[2]+x[0])/3
    if y3 > 0:
        y4 = math.sqrt(y3)
    else:
        y4 = float("-inf")
    return y1+y2+y4
    #return round(10*(x[0]+x[1]+x[2]))
def f2(fix,x):
    a = fix
    return ((a[4]*a[5]*a[6])**2)/(1+x[0]*x[1]*x[2])
    #return x[4]+x[5]+x[6]
def f3(fix,x):
    a = fix
    return ((148+x[0]**3+22*a[7])/(3+a[5]*0.93))
    #return round(10*x[7])
def fbrute(fix,x):
    z1 = f1(fix,x)
    z2 = f2(fix,x[3:])
    z3 = f3(fix,x[6:])
    #print(z1,z2,z3)
    return z1 + z2 + z3
def flatten(d):
    d.sort(key=lambda x: x[0])
    l=len(d)
    newD = []
    acc = 0
    for i in range(len(d)-1):
        if d[i][0] != d[i+1][0]:
            x = (d[i][0], acc+d[i][1] )
            newD.append(x)
            acc = 0
        else:
            acc += d[i][1]
    x = (d[len(d)-1][0], acc+d[len(d)-1][1])
    newD.append(x)
    return newD
def precise(rawdist):
    dist = sorted(rawdist, key=lambda tup: tup[0])
    prob = 0
    for i in range(len(dist)):
        prob += dist[i][1]
        if prob > 0.5:
            return dist[i][0]
        else:
            pass
    raise ValueError('Probabilities broke :(')
def solu(fix,ran):
    ff1 = partial(f1,fix)
    ff2 = partial(f2,fix)
    ff3 = partial(f3,fix)
    p1 = flatten(funDistr(ff1,ran,[0,1,2]))
    p2 = flatten(funDistr(ff2,ran,[4,5,6]))
    p3 = flatten(funDistr(ff3,ran,[7]))
    y = funDistr(f0,[p1,p2,p3],[0,1,2])
    return precise(y)
def mkTest():
    distLen = 6
```

```
    fixd = []
    rand = []
    for i in range(8):
        fixd.append(random.uniform(0,10))
        oneVar = []
        summy = 0
        for j in range(distLen):
            v = random.uniform(0,10)
            r = random.random()
            oneVar.append((v,r))
            summy += r
        for j in range(distLen):
            (oldV, oldP) = oneVar[j]
            oneVar[j] = ( oldV, oldP/summy )
        rand.append(oneVar)
    result = solu(fixd,rand)
    diff = math.ceil(result) - result
    if diff < 0.2:
        patch = random.uniform (0.2-diff,0.8-diff)
        fixd[0] -= patch
    if diff > 0.8:
        patch = random.uniform (diff-0.8,diff-0.2)
        fixd[0] += patch
    return (fixd,rand)
def generate():
    num_tests = 10
    tests = []
    for i in range(num_tests):
        case = mkTest()
        tests.append((str(case)+"\n",case))
    return tests
def solve(dataset):
    (fix,ran) = eval(dataset)
    return str(math.ceil(solu(fix,ran)))
def slowsolve(dataset):
    (fix,ran) = eval(dataset)
    ffbrute = partial(fbrute,fix)
    x=funDistr(ffbrute,ran,[0,1,2,4,5,6,7])
    y=precise(x)
    return str(math.ceil(y))
def test():
    (fix,ran) = mkTest()
    t0 = time()
    sl = (slowsolve(str((fix,ran))))
    print("BRUTE ",sl)
    t1 = time()
    print(t1-t0)
    t0 = time()
    sf = solve(str((fix,ran)))
    t2 = time()
    print(t2-t0)
    print("CLEVER",sf)
    #print(len(p))
def check(reply,clue):
    (fix,ran) = clue
    their = int(reply)
    our = math.ceil(solu(fix,ran))
    return our == their
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.6.1. Сейсмологическая одновременность (3 балла)

Условие

Имеется 10 сейсмографов, расположенных в известных координатах. Сколько на карте есть точек, таких, что:

1. Больше двух сейсмографов зарегистрируют событие, произошедшее в этой точке, одновременно.
2. Расстояние от эпицентра события до начала координат меньше, чем до любого из сейсмографов **из условия 1**.

Скорость распространения волн на изучаемой территории считать одинаковой во всех точках и направлениях.

Поверхность изучаемой территории считать плоской.

Изучаемую территорию считать ничем не ограниченной.

Описание формата данных	Входные данные: массив кортежей, каждый из которых содержит координаты одного сейсмографа. Выходные данные: целое число Обратите внимание, что пример входных данных не является ответом к примеру выходных данных!
Пример входных данных	[(-19.19831037099226, 91.29259948380525), (-31.404349516298964, -67.74354344337246), (33.82978301141941, -89.06745638194487), (-25.679014609987178, -25.632782661270625), (94.75862077704, -51.603687121929376), (25.301160932643455, -87.12989051411013), (26.66718015184975, 95.06090749451579), (-79.16682375010834, -94.1252168832308), (-35.51166674005286, -73.44924630571055), (94.23992056373254, -63.389901730938305)]
Пример выходных данных	99
Ограничения вычислительных ресурсов	Время выполнения программы на сервере не более 1 секунды. Требуемая память на сервере не более 128 мегабайт.

Решение

Альтернативная формулировка задачи: дано 10 точек. Найти сколько из них можно составить треугольников таких, что для каждого из них начало координат находится в центре описанной вокруг него окружности.

Проверяющий код

```
import random
from math import sqrt
def genPoint():
    x = random.uniform(-100,100)
    y = random.uniform(-100,100)
    return (x,y)
def independable(xs):
```

```
    vector = [ xs[0][0]-xs[1][0]
             , xs[0][1]-xs[1][1]
             ]
    if vector[0] == 0 and vector[1] == 0:
        return False
    if vector[0] == 0:
        if xs[0][0] == xs[2][0]:
            return False
        else:
            scaleX = float("inf")
    else:
        scaleX = xs[2][0]/vector[0]
    if vector[1] == 0:
        if xs[0][1] == xs[2][1]:
            return False
        else:
            scaleY = float("inf")
    else:
        scaleY = xs[2][1]/vector[1]
    return scaleX != scaleY
def generate():
    tests = []
    num_tests = 100
    for _ in range(num_tests):
        done = False
        while not done:
            done = True
            case = [ genPoint() for _ in range(10)]
            for (a,b,c) in cnk(case,3):
                if not independable([a,b,c]):
                    done = False
        tests.append((str(case)+"\n",case))
    return tests
def genTriple():
    que=[(0,0),(0,0),(0,0)]
    while not independable(que):
        que = [ genPoint() for _ in range(3) ]
    return que
def generate_():
    num_tests = 1
    tests = []
    for _ in range(num_tests):
        case = [ genTriple() for _ in range(10)]
        tests.append((str(case)+"\n",case))
    return tests
def cnk_(start,end,k):
    if k == 0:
        return [[]]
    out = []
    for take in range(start,end-k+1):
        for e in cnk_(take+1,end,k-1):
            out.append([take+1]+e)
    return out
def cnk(elems,k):
    combos = cnk_(0,len(elems),k)
    out = []
    for ixes in combos:
        tmp = [ elems[i-1] for i in ixes ]
        out.append(tmp)
    return out
def circle(p1,p2,p3):
    a = sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)
    b = sqrt((p2[0]-p3[0])**2 + (p2[1]-p3[1])**2)
    c = sqrt((p1[0]-p3[0])**2 + (p1[1]-p3[1])**2)
```

```
    p = ( a + b + c ) / 2
    r = a * b * c / 4 / sqrt ( p * (p-a) * (p-b) * (p-c) )
    x12 = p1[0]-p2[0]
    x23 = p2[0]-p3[0]
    x31 = p3[0]-p1[0]
    y12 = p1[1]-p2[1]
    y23 = p2[1]-p3[1]
    y31 = p3[1]-p1[1]
    z1 = p1[0]**2 + p1[1]**2
    z2 = p2[0]**2 + p2[1]**2
    z3 = p3[0]**2 + p3[1]**2
    z = x12 * y31 - y12 * x31
    zx = y12 * z3 + y23 * z1 + y31 * z2
    zy = x12 * z3 + x23 * z1 + x31 * z2
    rx = -zx / 2 / z
    ry = zy / 2 / z
    return (r,(rx,ry))
def isCircleGood(circle):
    r = circle[0]
    x = circle[1][0]
    y = circle[1][1]
    return x**2 + y**2 < r ** 2
def nub(elems):
    out = []
    for e in elems:
        inside = False
        for o in out:
            if e == o:
                inside = True
        if not inside:
            out.append(e)
    return out
def solu(points):
    good = 0
    for (a,b,c) in nub(cnk(points,3)):
        if isCircleGood(circle(a,b,c)):
            good += 1
    return good
def solve(dataset):
    return str(solu(eval(dataset)))
def check(reply,clue):
    their = int(reply)
    our = solu(clue)
    return our == their
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.6.2. Ветряки Беца (2 балла)

Условие

Мощность, вырабатываемая турбиной ветровой электростанции, выражается следующей формулой:

$P(u) = \mu \frac{1}{2} \rho u^3 \frac{\pi d^2}{4}$, где μ — коэффициент мощности турбины (он зависит от её конструкции), ρ — плотность воздуха, d — диаметр лопастей турбины. В реальности эта зависимость интереснее, с потерями на трение и вибрации, зависящими от скорости вращения турбины, но для нашей задачи мы эти сложности учитывать не будем.

Вам нужно решить, сколько нужно построить дополнительных турбин на электростанции, чтобы покрыть недостаток электроэнергии в энергосистеме.

Вам даны: почасовые данные о скорости ветра за 10 дней, почасовые данные о недостатке электроэнергии в системе.

Вам доступны небольшие, но очень эффективные турбины с коэффициентом мощности 59,2% и диаметром лопастей 20 метров.

Нужно вычислить, какое количество турбин надо построить, чтобы:

1. Ни в один момент времени (по имеющимся данным) дефицит **мощности** не превысил 500 кВт.
2. Дефицит электроэнергии за любые идущие подряд 24 часа составлял не более 1МВт*ч.
3. Число турбин должно быть минимальным, удовлетворяющим этим условиям

Скорость ветра одинакова для любого числа турбин.

Плотность воздуха считать равной 1,2466 кг/м³.

Описание формата данных	Обратите внимание, что пример выходных данных не является ответом к примеру входных данных! Входные данные: кортеж из двух массивов, в первом содержатся данные за 239 часов о недостатке электроэнергии в системе (в Ваттах), во втором — данные за 239 часов о скорости ветра. Выходные данные: число ветрогенераторов, которые нужно построить.
Пример входных данных	([0, 641152.5047465652, 576465.5670411313, 0, 0, 89577.95339977747, 18512.68835308484, 0, 0, 0, 521708.48311472853, 320858.659498753, 0, 422056.4611855966, 0, 0, 806900.2376077835, 0, 704100.3816498647, 0, 865213.7314178485, 729014.0364394245, 0, 0, 0, 691617.4664802587, 0, 0, 0, 0, 0, 0, 50286.03716309532, 708243.9219484641, 695733.6270480742, 0, 0, 0, 425422.9266193688, 0, 916616.629767943, 159406.34779288142, 0, 0, 0, 0, 974484.1809144919, 669031.178199078, 524267.63826771063, 0, 0, 0, 0, 0, 262256.110940563, 0, 590648.0052649293, 0, 39262.73349971687, 0, 0, 710994.4529334186, 637810.606599939, 513244.66590851656, 84220.63493245635, 185146.20383709256, 717545.9781915534, 431908.8718715649, 0, 0, 0, 564125.069537407, 0, 581952.8248449664, 0, 709352.9507263942, 0, 0, 149534.22886141288, 0, 0, 772469.4825745708, 791946.7851043598, 0, 697780.1935676287, 94093.63619430356, 0, 0, 0, 851126.0305314183, 568663.6690290319, 53717.35376840614, 0, 0, 257204.993122223, 141026.71430416414, 0, 303647.99893781455, 0, 0, 959693.3788930885, 0, 92941.74176055093, 0, 355548.1403382135, 0, 0, 880500.948819, 787143.4631740376, 44064.039448384996, 0, 0, 722339.2350421782, 595279.0137892447, 0, 925480.4790383207, 0, 0, 0, 156058.6066937427, 885344.161887555, 0, 13318.392764843478, 493973.2417113508, 799957.0626109367, 0, 0, 0, 844488.2623494187, 298110.98274720507, 594270.8454770402, 0, 912543.5970348665, 325308.4003224683, 0, 268927.18382202653, 400416.3352608071, 502097.53110999323, 0, 0, 502031.3667787676, 414519.093264178, 0, 17148.933745105176, 0, 0, 0, 0, 0, 143999.39538958162, 0, 957483.0829666953, 0, 0, 0, 806820.7060272121, 938594.0065393674, 0, 0, 0, 427104.5436087104, 440325.9346970002, 602512.5659492536, 864216.2866305338, 0, 0, 0, 449055.658488696, 0, 183710.5854197335, 0, 862192.4532781802, 67207.10220044712, 0, 0, 729578.4800237967, 0, 0, 0, 0, 0, 0, 559882.9428921429, 704636.6205462655, 0, 684278.5214254573, 306008.5691583601, 0, 860945.1582767079, 464117.48468657566, 0, 0, 0, 0, 0, 604845.4257730881, 771538.2077797155, 482216.1732171586, 0, 0, 241581.10605385862, 336524.83593223913, 0, 0, 0, 0, 462579.9880606555, 359443.31257912645, 0, 470192.5588973943, 601775.194071735, 0, 763142.4613142567, 0, 0, 0, 0, 0, 92993.73066794925, 883157.723236782, 478771.19277087477, 540891.9727926875, 641457.9645915705, 538990.3509426361, 862578.0733497974, 0, 0, 0, 0, 0, 383941.5478825527, 135713.64504849503, 0, 613779.3077404135, 386307.7127620814, 884474.9873598143, 546548.8560004401, 0, 0, 562455.7046793788, 0, 0, 571223.5674665384, 0], [7.622823318303678, 9.90426457035051, 4.995385486259914, 7.405932465277108, 8.628535143764616, 8.793510686746863, 5.92911427171197, 7.487841294589781, 11.359314492836218, 3.21161497746813, 4.402386622076857 5, 6.82376287188487, 7.3595566521231275, 5.486801668316119, 4.270325118359812, 11.633435476752567, 8.250462058176916, 4.479964425447194, 4.945199481924722, 7.684117099510488, 4.4787500303292, 5.592056549961818, 7.30402585050284, 3.9613045477486306, 11.905988031474097, 6.784651637206236, 6.98014911756112, 7.140480957218601, 3.597251201069475, 10.12077834302452, 5.6730117349501405, 6.085810171312441, 9.097373822687219, 6.5297496356475415, 9.343731500307513, 10.51465166084758, 7.836271132471337, 5.967540325131591, 8.301639258558781, 8.492875648363107, 10.132343621600016, 4.294467725384068, 4.301023895265296, 4.888299159959303, 4.232052907629321, 11.300007853082256, 9.52910839611831, 6.433511001448896,

	6.170290279779341, 5.85910566434713, 11.891447553061777, 4.636856140136828, 6.626972490909932, 7.283714426204532, 5.087397318953052, 9.36530613326012, 8.309188261706748, 10.3247310982121, 4.202035893870533, 6.578825090797857, 9.646430330613907, 5.329526160640645, 9.656267173726802, 6.999037811387847, 11.21878882763644, 9.436633039233323, 8.0588737009279, 4.563556629928886, 10.683959459463372, 5.0279961888733915, 11.679671461842482, 10.422356289851795, 3.9412402719053414, 5.297746114270298, 6.591213868688521, 3.2547311758031627, 6.538445540613232, 8.910689182599427, 5.2133926747764034, 7.713755877331305, 5.100962924216839, 9.57200210230188, 10.933727232797397, 8.58855650126503, 9.930119564172248, 11.373488730870626, 7.015041976201172, 7.523573234715622, 3.23542574296763, 11.828252007295786, 6.1341985075862775, 11.23620525959052, 10.612148707708036, 6.9771918752713376, 10.702319661385692, 6.012743098545478, 8.969735186443998, 3.3605171521321213, 6.493346789182116, 9.894721058123336, 5.6043285971133034, 5.271854601686886, 7.617876634900605, 10.922414970623917, 8.104045101967285, 7.118034603607885, 5.184202875713068, 11.211752109849035, 4.818954940147675, 6.025717315344813, 3.317325080550021, 11.353872153236258, 11.409095360920354, 5.546583699205675, 9.217249790470945, 7.243107668214879, 3.0807916227023844, 8.019337401661986, 4.785538234353881, 9.104502586187268, 8.661177733999992, 6.624459210254446, 7.262513941053421, 9.423442089662224, 9.626858733158393, 11.610162439224434, 7.075761687143298, 7.210280169042683, 10.967833061663038, 7.162760049208466, 6.008808263508216, 7.447575678685832, 10.956098762004713, 8.275807612095434, 4.050386328381153, 11.895043864130148, 7.912611604312824, 6.826072699901439, 10.133621137979448, 10.71180451043528, 11.680147066852449, 6.041502032727567, 10.040251345899744, 9.217095443156474, 10.352899870094603, 8.745264615901212, 6.1287019756276, 11.430447245769555, 9.892734526967516, 5.536954694565278, 5.567457913600048, 8.281506777784928, 7.64062885395502, 3.766554564707215, 6.967966804964262, 4.751549719899907, 6.959153946965211, 8.771967702705426, 4.449246948740138, 7.436283357979313, 10.551602344383518, 7.197255381148759, 10.818263615219937, 6.883801977076836, 7.391858826487197, 10.899279524080683, 10.12867854145053, 8.5615909959558, 10.718287230184451, 6.251277857541137, 9.328943448305681, 3.66551529040633, 8.853821217173566, 6.195624549262265, 5.6883911983511695, 5.5934666849213315, 8.333312087231192, 5.511571467548734, 6.323202346144242, 11.58332605124808, 3.914959517358648, 3.187292364478605, 9.548077738324555, 8.687270615977349, 5.567732393919439, 9.002596801068371, 4.747155276809428, 5.683431659543487, 4.164349359107765, 8.260960238293045, 10.464251439781343, 11.598190367622802, 9.043658192764932, 3.562762733365279, 7.875066926258507, 4.7043859681108255, 10.45012577890946, 3.5952521535540147, 11.773901639198332, 3.5411068820392053, 6.363245707868963, 9.47839137566936, 8.018547682054734, 9.752908076541273, 3.544716641448951, 10.264255434055585, 10.691471210425572, 9.311274792961234, 6.1593217282442945, 11.888261521703276, 8.263079153953228, 6.783835820510134, 10.725975957021266, 6.05301507813713, 6.967119996712591, 3.092998716023893, 4.255985076467285, 8.776977037217305, 8.371041078208687, 3.4390106495881447, 11.498279748777371, 3.8613051430197842, 10.024847342803954, 5.194970356917108, 6.719667270623727, 11.476578362417973, 8.630997173843905, 11.931650686561301, 3.5036546533457757, 11.02355376375226, 3.6646774013377756, 11.870917084191376, 6.205226451222023, 6.684580896855915, 5.331545778459204, 5.408362567863108, 9.726841682982183, 3.7606012070701396, 11.565689087362045])
Пример выходных данных	99
Ограничения вычислительных ресурсов	Время выполнения программы на сервере не более 1 секунды. Требуемая память на сервере не более 128 мегабайт.

Решение

Проверяющий код

```
from random import uniform, random
from math import pi
def power(u):
    return 0.592 * 1.2466 / 2 * u**3 * pi * 20**2 / 4
def deficits(cons,winds,tourbines):
    out = []
    for i in range(len(cons)):
        x = max (0, cons[i] - tourbines * power(winds[i]))
        out.append(x)
    return out
```

```
def maxDeficit(cons,winds,tourbines):
    return max(deficits(cons,winds,tourbines))
def sumDeficit(cons,winds,tourbines):
    acc = 0
    for i in range(len(cons)-24):
        tmp = [ deficits(cons,winds,tourbines)[i+j] for j in range(24) ]
        if acc < sum(tmp):
            acc = sum(tmp)
    return acc
def answer(cons,winds):
    tourbines = 0
    while True:
        if maxDeficit(cons,winds,tourbines) <= 500e3 and
sumDeficit(cons,winds,tourbines) <= 1e6:
            return tourbines
        else:
            tourbines += 1
def gen():
    outw = []
    outc = []
    for _ in range (239):
        w = uniform (3,12)
        if random() > 0.5:
            c = 0
        else:
            c = uniform(0,1e6)
        outw.append(w)
        outc.append(c)
    return (outc,outw)
def test():
    c,w = gen()
    print(answer(c,w))
#test()
def generate():
    num_tests = 10
    tests=[]
    for _ in range(num_tests):
        case = gen()
        tests.append((str(case)+"\n",case))
    return tests
def solve(dataset):
    c,w = eval(dataset)
    return str(answer(c,w))
def check(reply,clue):
    their = int(reply)
    our = answer(clue[0],clue[1])
    return their == our
#print(solve(input()))
```

Пример решения

Можно получить из проверяющего добавлением к нему строки

```
print(solve(input()))
```

Ответ

Правильным является любая программ, которая работает идентично приведенному выше примеру решения.

Задача 2.1.7.1. Недетерминированный конь (2,1 балла)

Условие

Шахматный конь находится на клетке кубика Рубика, прилегающей к вершине. Какова вероятность того, что через три случайных хода он окажется на противоположной грани? Конь ходит по клеткам кубика, как по шахматным.

Решение

Достаточно перебрать все 512 вариантов траектория движения коня. Это удобно сделать, например, склеив кубик рубика и разметив на нём все варианты.

Ответ

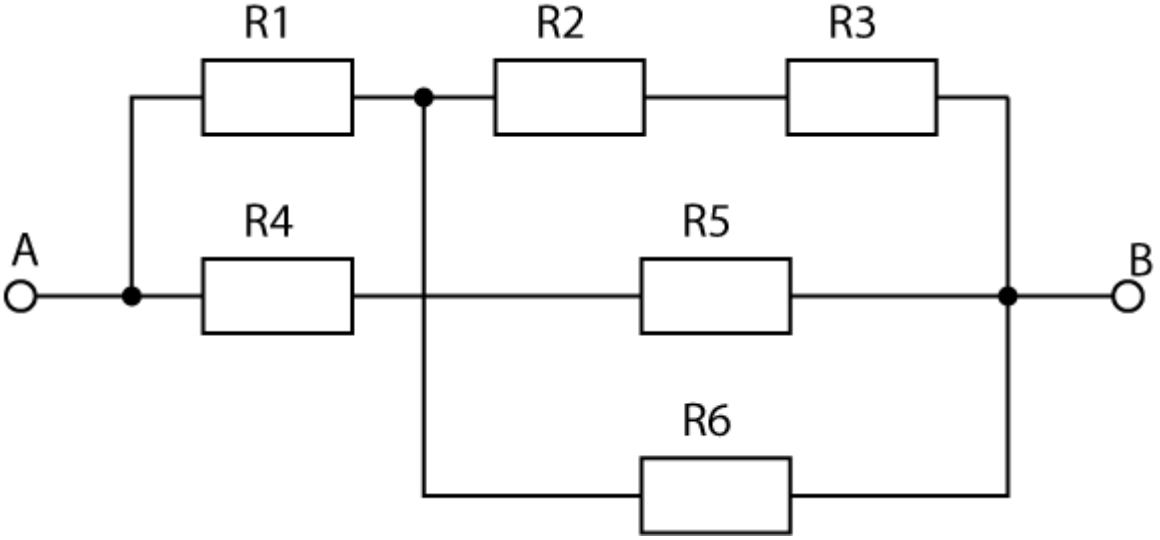
0.1328125 или 0.13671875 (в зависимости, то того, ходит конь «буквой Г» или «буквой L»).

Задача 2.1.7.2. Случайное сопротивление (3,5 балла)

Условие

Излишне решительный электронщик впаивает резисторы R1…R6 в цепь, изображённую на рисунке, выбирая их случайно из большой кучи.

В куче есть резисторы следующих номиналов:



- 100 Ом
- 120 Ом
- 150 Ом
- 180 Ом
- 220 Ом
- 270 Ом
- 330 Ом
- 390 Ом
- 470 Ом
- 560 Ом
- 680 Ом

- 820 Ом
- 1000 Ом

Куча очень большая, поэтому то, какие резисторы мы из неё уже взяли, не влияет на вероятность достать следующий — как будто мы проводим выборку с возвращением. Вероятность вытащить любой номинал такая же, как вероятность вытащить любой другой.

Каково математическое ожидание сопротивления между точками A и B.

Решение

«Вручную» задачу решить практически невозможно. Для нахождения ответа нужно написать программу, которая:

1. По заданным номиналам сопротивлений R1…R6 находит сопротивление всей цепи.
2. Перебирает все возможные значения номиналов и находит среднее значение сопротивления цепи.

Пример программы, вычисляющей ответ:

```
def R(R1,R2,R3,R4,R5,R6):
      D45 = 1/(R4+R5)
      D23 = 1/(R2+R3)
      D236 = D23 + (1/R6)
      R236 = 1 / D236
      D1236 = 1 / (R236+R1)
      return 1 / (D45+D1236)
noms = [ 100,120,150,180,220,270,330,390,470,560,680,820,1000 ]
def answer():
      out = 0
      for r1 in noms:
      for r2 in noms:
            for r3 in noms:
                  for r4 in noms:
                  for r5 in noms:
                        for r6 in noms:
                              out += R(r1,r2,r3,r4,r5,r6)
      return out / ( len(noms) ** 6 )
print(answer())
```

Ответ

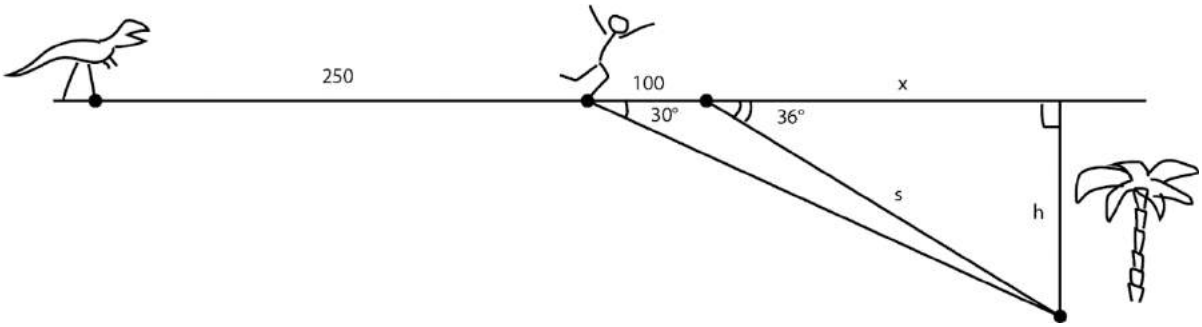
Число в диапазоне 321.492681±0.000001. Диапазон задан для нивелирования погрешностей округления.

Задача 2.1.7.3. Триангуляция на бегу (0,6 баллов)

Условие

Незадачливый путешественник во времени бежит по бескрайнему лугу, спасаясь от любопытного тираннозавра. Скорость путешественника 8 м/с, скорость тираннозавра 12 м/с. В тот момент, когда расстояние между ними составляет 250 м, путешественник замечает в направлении 30° по отношению к направлению его движения высокую пальму, на которой он, наверное, может спастись. Пробежав ещё 100 метров в прежнем направлении, он замечает, что направление на пальму теперь составляет 36°, и сворачивает к ней. На каком расстоянии от спасительной пальмы любопытный тираннозавр догонит незадачливого путешественника, если будет бежать в точности по его следам?

Решение



(x+100)/tg30° = x/tg36°, отсюда x = 100 * tg30° / (tg36°-tg30°) = 386,984065911021
h = x * tg36° = 281,1603815447865
s = 478,338611675281

Скорость сближения равна 4 м/с, расстояние между участниками гонки = 250 м, следовательно, гонка закончится не позднее, чем через 62,5 с. Путешественник за это время пробежит 500 м. Расстояние до пальмы составляет 100 + 478,338611675281 = 578,338611675281 м, следовательно, встреча произойдёт в 78,338611675281 метрах от пальмы.

Ответ

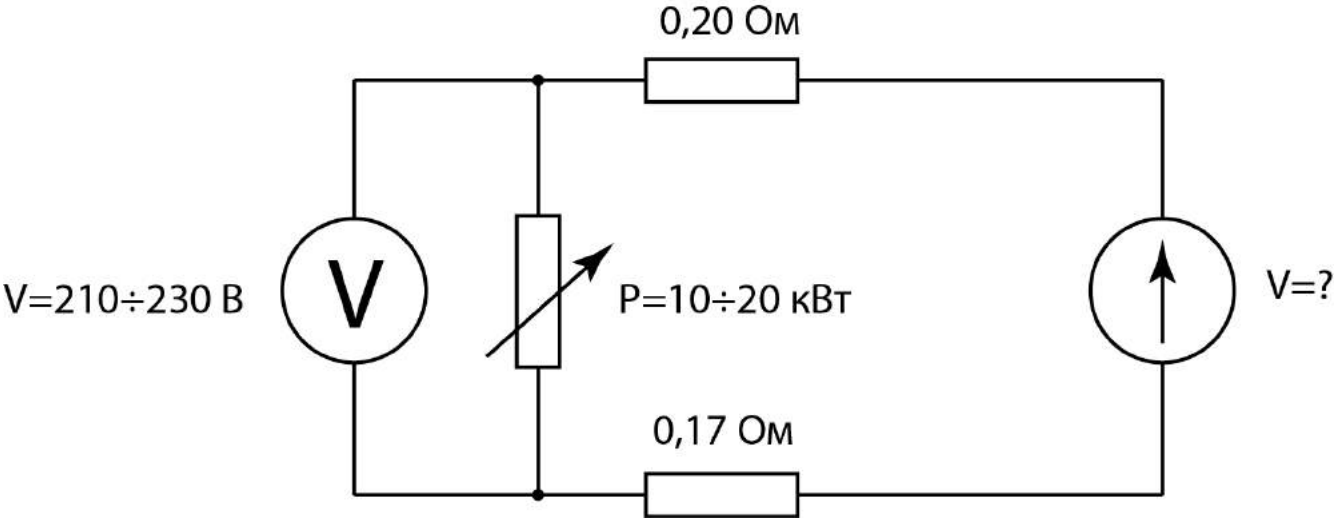
Число в диапазоне 78.338611±0.000001. Диапазон задан для нивелирования погрешностей округления.

Задача 2.1.7.4. Напряжённая вилка (2,7баллов)

Условие

От управляемого трансформатора к потребителю идёт два кабеля — «фаза» и «ноль». Сопротивление провода фазы составляет 0,20 Ом, сопротивление провода земли — 0,17 Ом. Какое напряжение нужно выставить на трансформаторе, чтобы любых изменениях нагрузки потребителя (которая может составлять от 10 до 20 кВт) напряжение между фазой и нулём в точке подключения потребителя оказалось в диапазоне от 210 до 230 В? Нагрузка потребителя имеет свойство потреблять постоянную мощность при любом напряжении, не выходящем за пределы этого диапазона.

Решение



Переменное напряжение во всей задаче можем заменить на постоянное действующее, так как реактивных нагрузок в цепи нет.

При уменьшении мощности нагрузки напряжение на ней будет возрастать, при увеличении — убывать.

Найдём предельные значения напряжения на трансформаторе (на схеме — источнике ЭДС):

При минимальной нагрузке напряжение не должно превысить 230 В:

Ток нагрузки в таких условиях равен 10000/230 = 43,478260869565

Падение напряжения на проводах составит 0,37 * (10000/230) = 16,086956521739 В.

Следовательно, максимально допустимое напряжение на трансформаторе составит 246,086956521739 В.

При максимальной нагрузке напряжение не должно стать меньше минимального. Отсюда аналогично получаем минимальное напряжение на трансформаторе равно 210 + (0,37 * 20000 / 210) = 245,238095238095

На трансформаторе можно выставить любое напряжение между полученными двумя значениями, например, 246 В.

Ответ

Число в диапазоне 245.66252587991718 ± 0.42443064182194234.

2.2 Критерии оценивания

Максимально возможное число баллов за тур — 70. Баллы автоматически считала программа на платформе Stepik за каждую задачу. В финал прошли команды, набравшие не менее 9,9 баллов (победители и призеры). Итого в финал прошло 8 команд, 32 участника.