2. ВТОРОЙ ЭТАП

Задачи второго этапа

3.1. Задание 1

Задача 3.1.1. Мысленный аукцион (10 баллов)

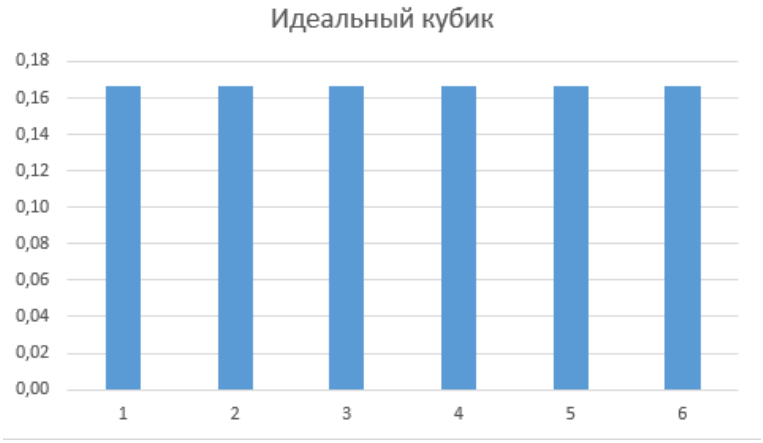
Голандский аукцион

Существует много схем аукционов. Как правило, они эквивалентны друг другу, но на практике отличия могут быть существенными. В схеме «голландского аукциона» когда объявляется лот, на специальном табло загорается цена, которая со временем снижается (и довольно быстро). У участников аукциона есть кнопка, нажав которую, они приобретут лот по той цене, которая в момент нажатия высвечивается на табло. Преимущества такой схемы аукциона в том, что розыгрыши идут очень быстро и за фиксированное время, и можно даже разыгрывать несколько лотов одновременно. Недостатком является то, что у участников очень мало времени на принятие решения, поэтому эта схема подходит в основном для большого числа однородных, но различных лотов. Например, цветов.

Вероятностные распределения

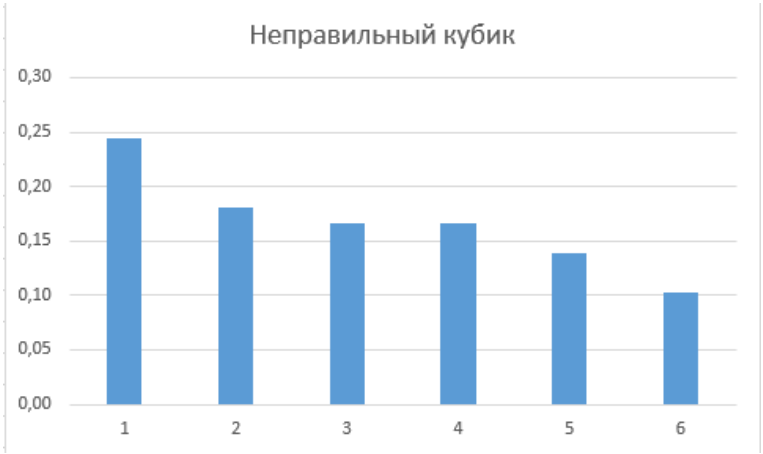
Вероятностные распределения — это характеристика случайной величины, по сути являющаяся её определением, которая говорит о том, какой исход случайной величины с какой вероятностью произойдёт.

Рассмотрим, например, случайную величину «число очков на шестигранном игральном кубике». Возможные исходы — 1, 2, 3, 4, 5 и 6; вероятности всех исходов одинаковы и равны $1/6$. В виде графика это распределение будет выглядеть вот так:



Рассмотрим в качестве второго примера не идеальный, а реальный кубик, такой, как на рисунке:

Из-за того, что очки на нём нанесены в виде больших углублений, у него смещён центр тяжести, и он имеет такое распределение вероятностей:



Вероятности исходов у такого кубика равны (по результатам 100 бросков): для 1 — 0,24, для 2 — 0,18, для 3 — 0,17, для 4 — 0,17, для 5 — 0,15, для 6 — 0,10.

Если мы вычислим среднее ожидаемое число очков на каждом их этих кубиков (математические ожидания соответствующих случайных величин), то получим следующее.

Для правильного кубика это

$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = 3,5$

Для нашего неправильного кубика это

$1 \times 0,24 + 2 \times 0,18 + 3 \times 0,17 + 4 \times 0,17 + 5 \times 0,15 + 6 \times 0,10 = 3,08$

Условие задачи

У вас есть три корзинки с тюльпанами, которые вы выставляете на голландский аукцион со стартовой ценой в 49 и шагом 1.

В аукционе участвует всего 5 участников, и у вас есть предположения о ценах, которую каждый участник готов заплатить за каждый лот.

Всего у вас, естественно, 15 таких предположений, и каждое из них представляет собой вероятностное распределение на ценах от 0 до 49.

Если два или более участников объявят одинаковую цену, то кто-то из них нажмёт кнопку чуть раньше.

Вероятность этого одинакова для всех участников.

Ноль — допустимая цена, и если все выбрали такую ставку, то кому-то корзинка достанется бесплатно.

Каждый участник купит не больше одной корзинки.

Исходя из ваших предположений об участниках, найдите, на сколько процентов можно поднять выручку от аукциона изменением порядка лотов по сравнению с начальным.

Формат входных данных

Трёхмерный массив, первый индекс которого — участник аукциона (его номер), второй — номер корзинки с цветами (в изначальной расстановке), третий — цена. Значение, которое хранится в массиве — вероятность того, что этот участник решит купить эту корзинку именно по этой цене.

Формат выходных данных

Число с плавающей точкой, например, 0.012182432410846942.

Обратите внимание, что ответ нужно выдать в процентах.

Требуемая точность — хотя бы 1e-10.

Пример входных данных

[[[0.0, 0.0, 0.00037280466184606246, 0.0006687013716653295, 0.0014910532789078324, 0.0022033598533922837, 0.0033672906128430487, 0.004827078289717003, 0.006056792296892899, 0.007235026230088179, 0.00894220596189969, ⋯]]]

Пример выходных данных

0.012182432410846942

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 30 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Для решения задачи нужно выделить и перебрать вероятности всех элементарных событий — действительных наборов ставок. В паре с фиксированной последовательностью лотов это позволяет получить вероятностное распределение результатов аукциона (при заданной последовательности лотов) и вычислить его математическое ожидание. Перебрав все 120 последовательностей, несложно получить ответ. Ограничения вычислительных ресурсов.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import copy
import itertools
import sys

totalPlayers = 5
totalFlowers = 3
totalPoints = 50
allP = [ x for x in range(totalPlayers)]
allF = [ x for x in range(totalFlowers)]
allPr = [ x for x in range(totalPoints)]

leftLim = -2
rightLim = 2

def patchLim(r):
    return r*(rightLim-leftLim) + leftLim

random.seed(883)

def smoothgen():
    num2 = []
    for _ in allPr:
        num2.append(patchLim(random.random()))
    num1 = [0]
    for i in range(totalPoints-1):
        new = num1[i] + num2[i]
        num1.append(new)
    num0 = [0]
    for i in range(totalPoints-1):
        new = num0[i] + num1[i]
        num0.append(new)
    adjustSign = min(num0)
    for i in range(totalPoints):
        num0[i] = num0[i]-adjustSign
    adjustScale = sum(num0)
    for i in range(totalPoints):
        num0[i] = num0[i]/adjustScale
```

```
    return num0

def genDistro():
    byPlayers = []
    for i in allP:
        byFlowers = []
        for j in allF:
            byFlowers.append(smoothgen())
        byPlayers.append(byFlowers)
    return byPlayers

distro = genDistro()

def d(player,flower,price):
    return distro[player][flower][price]

def probLessThanValue(value,player,flower):
    result = 0
    for i in range(value):
        result += d(player,flower,i)
    return result

def probOutcomeSucc(parts,plrs,flower,price,winner):
    result = 1
    for p in parts:
        if p == winner:
            result *= d(p,flower,price) / len(plrs)
        elif p in plrs:
            result *= d(p,flower,price)
        else:
            result *= probLessThanValue(price,p,flower)
    return result

def powerset(parts):
    result = [[]]
    for p in parts:
        new = []
        cp = copy.deepcopy(result)
        for x in cp:
            x.append(p)
            new.append(x)
        result += new
    return result

def outcomes(participants,who):
    parts = copy.deepcopy(participants)
    if who in parts:
        parts.remove(who)
        outs = powerset(parts)
        for o in outs:
            o.append(who)
        return outs
    else:
        return []

def mainFn(parts,winner,price,flower):
    if winner in parts:
        outs = outcomes(parts,winner)
        result = 0
        for o in outs:
```

```
            result += probOutcomeSucc(parts,o,flower,price,winner)
        return result
    else:
        return 0

def probPlayer(others,flower,who):
    result = 0
    for pr in allPr:
        result += mainFn(others,who,pr,flower)
    return result

def processChain(participants,flowers):
    if flowers == []:
        return 0
    else:
        result = 0
        f,*fs = flowers
        for w in participants:
            expect = 0
            for pr in allPr:
                expect += pr * mainFn(participants,w,pr,f)
            parts = copy.deepcopy(participants)
            parts.remove(w)
            result += expect + probPlayer(participants,f,w) * processChain(parts,fs)
    return result

def var(fs):
    return processChain(allP,fs)

def answer():
    fs = itertools.permutations(allF)
    exemplar = var(allF)
    best = 0
    for f in fs:
        v = var(f)
        if best < v:
            best = v
    result = (best-exemplar)/exemplar*100
    return result

def solu(values):
    global distro
    oldDistro = copy.deepcopy(distro)
    distro = values
    result = answer()
    distro = oldDistro
    return result

def testD():
    global distro
    oldDistro = copy.deepcopy(distro)
    distro = testDistMap2
    val = d(1,1,50)
    result = val == 8.283601731573895e-3
    if not result:
        print(val)
    distro = oldDistro
    return result

def testLessThan():
```

```
159     global distro
160     oldDistro = copy.deepcopy(distro)
161     distro = testDistMap2
162     val = probLessThanValue(50,0,0)
163     result =  (val == 0.7391552524635776)
164     if not result:
165         print(val)
166     distro = oldDistro
167     return result
168 
169 def testOutcomeSucc():
170     global distro
171     oldDistro = copy.deepcopy(distro)
172     distro = testDistMap2
173     val = probOutcomeSucc(allP,[1,2],0,50,1)
174     result = val == 5.269109353831852e-7
175     if not result:
176         print(val)
177     distro = oldDistro
178     return result
179 
180 def testResult():
181     global distro
182     oldDistro = copy.deepcopy(distro)
183     distro = testDistMap2
184     val = answer()
185     result = val == 8.9573356038474e-2
186     if not result:
187         print(val)
188     distro = oldDistro
189     return result
190 
191 def test():
192     print("testD",testD())
193     print("testLessThan",testLessThan())
194     print("testOutcomeSucc",testOutcomeSucc())
195     print("testResult",testResult())
196 
197 def generate():
198     return [ str(genDistro())+"\n" for _ in range(10) ]
199 
200 def solve(dataset):
201     return str(solu(eval(dataset)))
202 
203 def check(reply,clue):
204     their = eval(reply)
205     our = eval(clue)
206     if abs(our-their) < 1e-10:
207         return True
208     if our > their:
209         feedback = "Слишком мало, можно лучше!"
210     if our < their:
211         feedback = "Слишком много!"
212     if their < 0:
213         feedback = "Как минимум ноль!"
214     return False, feedback
215 
216 def testStepik():
217     tests = generate()
218     for raw in tests:
```

```
        sol = solve(raw)
        print(check(sol,solve(raw)))


print(solve(sys.stdin.read()))
```

3.2. Задание 2

Задача 3.2.1. Предначертанный отказ (4 балла)

Имеется энергосистема, в которой находится известное число (100) потребителей электроэнергии.

Мощность каждого потребителя известна и постоянна в течение всего периода моделирования.

Каждый месяц потребитель может сделать одно из следующих действий:

1. Установить солнечную панель на 2 кВт мощности за 10000 рублей.
2. Установить солнечную панель на 6 кВт мощности за 25000 рублей.
3. Установить дешёвый энергетический порт на 5 кВт установленной мощности за 15000 рублей.
4. Установить дорогой энергетический порт на 5 кВт установленной мощности за 20000 рублей.

- С помощью энергетического порта потребитель может продавать излишек своей электроэнергии другим потребителям за 50% цены от цены сетевой компании.
- Солнечные панели нужно считать постоянными источниками мощности (как будто их график генерации сглажен идеальными накопителями энергии).
- Пользователь принимает действие, если при текущей цене сетевой компании на электроэнергию покупка окупится не более чем через три года. Если пользователь может предпринять несколько действий, он предпринимает то из них, которое окупается быстрее (а лучше ещё и стоит меньше).
- Сетевая компания несёт фиксированные издержки на содержание энергосистемы в 1 млн. рублей в месяц (или 12 миллионов за 365 дней), плюс 1,5 рублей за каждый произведённый кВт·час.
- Тариф на электроэнергию сетевая компания устанавливает таким, чтобы при текущем уровне потребления сетевая компания имела прибыль в 5%.
- Новый тариф устанавливается каждый год (раз в 12 месяцев, в том числе и в самом начале моделирования).
- Если в системе установленная мощность дешёвых энергетических портов окажется больше 30% от суммарного потребления мощности из сети (суммарной мощности, которую объекты потребляют от сетевой компании и других потребителей, на не из собственных солнечных панелей), в ней произойдёт авария из-за низкого качества передаваемой ими электроэнергии.
- Если установка пользователем дешёвого порта приведёт превышению установленной мощности дешёвых энергетических портов в сети показателя в

10% от суммарного потребления мощности из сети, сетевая компания запретит установку, и пользователь придётся исключить это действие из рассмотрения.

- Потребители принимают решения независимо друг от друга в том порядке, в котором они перечислены в исходных данных.
- Компания устанавливает цену раньше, чем потребители принимают решения о покупках.

Обратите внимание, что сама по себе задача несложная, но её условие нужно читать очень внимательно!

Послесловие

Эта задача — модель энергосистемы, которая демонстрирует один из возможных путей коллапса энергетики. Для проектирования развития энергосистем очень важно такие пути знать, чтобы держаться от них подальше.

Модель в этой задаче нарочно простая, чтобы условие задачи влезало на полстраницы; в реальности и энергетические компании устроены по-другому, и потребители ведут себя намного сложнее, и их потребление есть величина не постоянная, и т.д.

Любая модель отражает лишь малую часть моделируемой системы, и даже если какую-то черту системы включить в модель технически легко, нужно ещё и утвердительно ответить на вопрос «Важна ли эта черта для интересующих нас эффектов»? И если ответ содержит сомнение, то влияние этой черты на модель нужно изучать отдельно. Особенно если вас интересуют качественные характеристики модели, а черта влияет только на количественные. Чем модель будет проще, тем легче её будет изучать и разбирать её поведение по косточкам. Развитие модели — это не доведение её до похожести на настоящую систему, а изучение того, что именно приводит к возникновению или исчезновению эффекта, и какие черты настоящей системы влияют на этот эффект в худшую или лучшую сторону.

Формат входных данных

Мощности потребителей в кВт·ч, массив из 100 чисел с плавающей точкой, от 5 до 500.

Формат выходных данных

Количество месяцев до аварии, целое число, например, 97.

Пример входных данных

[91.05173216589283, 346.59744306410516, 79.340611380049, 6.301441137920902, 34.63321051774173, 283.7442062865518, 462.85639563921933, 18.35969426088177, 114.42284118915013, 348.2511151684352, ⋯]

Пример выходных данных

97

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Для решения достаточно внимательно прочитать условие и реализовать описанный в нём алгоритм. Задача не требует применения специфических знаний, но имеет значительную когнитивную трудность в виде высоких требований к внимательности. Ограничения вычислительных ресурсов влияют только при выборе катастрофически неэффективных алгоритмов, например, самостоятельной сортировки вместо использования встроенных в язык функций.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import copy
import sys

sample = [91.05173216589283, 346.59744306410516, 79.340611380049,
6.301441137920902, 34.63321051774173, 283.7442062865518,
462.85639563921933, 18.35969426088177, 114.42284118915013,
348.2511151684352, 35.81719885313268, 49.63674917703884,
467.9423981733799, 238.65743487391796, 147.2073340259839,
312.48489408473336, 138.01039332211423, 355.1005034359866,
29.340271106787984, 99.15946433242176, 101.62375687795341,
141.7968745962582, 264.1127225211994, 390.7520075028461,
26.574805514157045, 403.57290262223626, 303.2238232129529,
26.84536690537845, 50.625385690392626, 352.5875266607501,
490.5712855832286, 416.65057322723743, 122.00634222950978,
335.9327639665609, 284.63060901729824, 160.79226719983157,
223.84570254433135, 413.4381031366249, 187.54717290087862,
128.90402815207682, 159.9803378175234, 14.171716784140491,
155.82255763644835, 304.81212895990535, 45.49900226772919,
102.1328582020983, 279.3144857150673, 17.220775870782532,
11.775763243956991, 420.3908810294901, 218.56532462565573,
114.08772495108836, 474.84396820456203, 23.42964393526904,
488.342275079058, 140.96501499579327, 298.9578766530911,
267.67125182920097, 354.611490764674, 226.22210041061237,
62.486648272655984, 165.4886048470618, 374.29038006374896,
451.369147364035, 77.13392481127703, 151.34737510649012,
32.727027013870156, 295.5116227866447, 226.4690930528016,
424.58867910418365, 69.47863785878867, 234.0665621852868,
378.7013351244484, 443.67743755922623, 431.7846439734082,
363.81894140612695, 440.6357464469665, 453.521156833,
494.61361579067295, 354.0936576752547, 403.9161844666748,
147.84681444651815, 411.5024834919147, 64.96641657683537,
435.45151943325885, 374.9547328567214, 418.8032864136387,
463.61590561438766, 474.47404786220955, 264.00569373194526,
72.32714571028217, 225.68351098136608, 494.647162066111,
294.1248402828908, 339.89532040440764, 91.82306688959734,
254.15497797507396, 19.47245737881847, 392.5312375132503,
23.220173991813397]
```

```
39
40  def timeme(fn):
41      from time import time
42      def inner(*args, **kwargs):
43          start = time()
44          r = fn(*args, **kwargs)
45          end = time()
46          print(fn.__name__, end-start)
47          return r
48      return inner
49
50  def testStepikOne(fn):
51      test = generate()[0]
52      your = timeme(fn)(test)
53      print("your", your)
54      clue = timeme(solve)(test)
55      print("mine", clue)
56      print(check(your,clue))
57  users = 100
58  userCount = -1
59
60  choiceSmallSolar = 0
61  choiceHugeSolar = 1
62  choiceBadInv = 2
63  choiceGoodInv = 3
64  choiceNone = 4
65
66  def traceChoice(c):
67      if c == 0:
68          return("Small")
69      if c == 1:
70          return("Huge")
71      if c == 2:
72          return("Bad")
73      if c == 3:
74          return("Good")
75      if c == 4:
76          return("None")
77
78  random.seed(532)
79
80  def patchRandom(r):
81      highest = 500
82      lowest = 5
83      return r*(highest-lowest)+lowest
84
85  def sampleGen():
86      result = []
87      for v in sample:
88          result.append(User(v))
89      return result
90
91  def genPowers():
92      result=[]
93      for _ in range(users):
94          result.append(patchRandom(random.random()))
95      return result
96
97  def makeUsers(powers):
98      result = []
```

```
    for p in powers:
        result.append(User(p))
    return result

def gen():
    result = []
    for _ in range(users):
        result.append(User(patchRandom(random.random())))
    return result

class User():
    consumption = 0
    generation = 0
    bandwidthGood = 0
    bandwidthBad = 0
    uid = 0
    def __init__(self,c):
        global userCount
        self.uid = userCount
        userCount += 1
        self.consumption = c
    def show(self):
        print("Cons",self.consumption,"Gen",self.generation,
              "Good",self.bandwidthGood,"Bad",self.bandwidthBad)
    def powerNeed(self):
        return max(0,self.consumption-self.generation)
    def sellingPower(self):
        fullBW = self.bandwidthBad + self.bandwidthGood
        fullG = self.generation - self.consumption
        if fullG <= 0:
            return 0
        else:
            return min(fullBW,fullG)
    def freeBandwidth(self):
        fullBW = self.bandwidthBad + self.bandwidthGood
        fullG = self.generation - self.consumption
        if fullG <= 0:
            return fullBW
        else:
            return max(0,fullBW - fullG)

def price(month,choice):
    if choice == choiceSmallSolar:
        pr = 10000
    elif choice == choiceHugeSolar:
        pr = 25000
    elif choice == choiceBadInv:
        pr = 15000
    elif choice == choiceGoodInv:
        pr = 20000
    else:
        pr = 0
    return pr #* 0.97 ** ( month // 12 )

def power(choice):
    if choice == choiceSmallSolar:
        return 2
    if choice == choiceHugeSolar:
        return 6
    if choice == choiceBadInv:
```

```
159         return 5
160     if choice == choiceGoodInv:
161         return 5
162     else:
163         return 0
164 
165 def payoffs(month,user,marketPrice,choice):
166     if choice == choiceSmallSolar or choice == choiceHugeSolar:
167         if user.powerNeed() >= power(choice):
168             pwr = power(choice)
169         elif user.powerNeed() <= 0:
170             pwr = min(user.freeBandwidth(),power(choice))/2
171         else:
172             pwr = user.powerNeed() + min(user.freeBandwidth(),
173             power(choice)-user.powerNeed())/2
174         if pwr == 0:
175             return float("inf")
176         return price(month,choice)/pwr/marketPrice
177     elif choice == choiceNone:
178         return float("inf")
179     elif choice == choiceBadInv or choice == choiceGoodInv:
180         if user.powerNeed() > 0 or user.freeBandwidth() > 0:
181             return float("inf")
182         else:
183             return price(month,choice) \
184                 / ( marketPrice / 2 ) \
185                 / min( power(choice),
186                       user.generation-user.consumption-user.bandwidthBad-user
187                                                                 .bandwidthGood)
188 
189 threshold = 24 * 365 * 3
190 
191 def userChoice(month,user,marketPrice,canBuyBad):
192     if canBuyBad:
193         opts = [ choiceSmallSolar, choiceHugeSolar, choiceBadInv, choiceGoodInv ]
194     else:
195         opts = [ choiceSmallSolar, choiceHugeSolar, choiceGoodInv ]
196     best = choiceNone
197     bestPayoff = float("inf")
198     for o in opts:
199         newPayoff = payoffs(month,user,marketPrice,o)
200         if newPayoff < bestPayoff:
201             best = o
202             bestPayoff = newPayoff
203     if payoffs(month,user,marketPrice,best) < threshold:
204         return best
205     return choiceNone
206 
207 def isSystemBroken(users):
208     cons = 0
209     badPower = 0
210     goodPower = 0
211     for u in users:
212         badPower += u.bandwidthBad
213         goodPower += u.bandwidthGood
214         cons += u.powerNeed()
215         #print(u.uid,u.consumption,u.generation,u.powerNeed())
216     result = badPower / cons >= 0.3
217     #print("Bad_power_and_cons_and_good_power",result,
218     #       badPower/cons,badPower,cons,goodPower)
```

```
219     return result
220     #return badPower / cons >= 0.3 #max(cons,badPower+goodPower) >= 0.3
221
222 def applyChoice(user,choice):
223     #traceChoice(choice)
224     if choice == choiceSmallSolar or choice == choiceHugeSolar:
225         user.generation += power(choice)
226     elif choice == choiceBadInv:
227         user.bandwidthBad += power(choice)
228     elif choice == choiceGoodInv:
229         user.bandwidthGood += power(choice)
230
231 def canAddBad(users):
232     cons = 0
233     badPower = power(choiceBadInv)
234     for u in users:
235         badPower += u.bandwidthBad
236         cons += u.powerNeed()
237     result = False
238     if cons > 0:
239         result = badPower / cons < 0.1
240     #print(result, badPower / cons)
241     return result
242
243 def fold(marketPrice,month,users):
244     for u in users:
245         oldUser = copy.deepcopy(u)
246         applyChoice(u,userChoice(month,oldUser,marketPrice,canAddBad(users)))
247         #u = applyChoice(oldUser,userChoice(month,oldUser,
248         #                marketPrice,canAddBad(users)))
249
250 def fixedPrice(users):
251     c = 1.5
252     operationalCosts = 1e6 * 12 / 365 / 24
253     userPower = 0
254     cons = 0
255     for u in users:
256         userPower += u.sellingPower()
257         cons += u.powerNeed()
258     soldPower = cons - userPower
259     if soldPower <= 0:
260         return float("inf")
261     return ( c + operationalCosts / soldPower ) * 1.05
262
263 def run(users):
264     month = 0
265     fixed = fixedPrice(users)
266     marketPr = fixedPrice(users)
267     while True:
268         if isSystemBroken(users):
269             #return users
270             return month
271         if month % 12 == 0:
272             fixed = fixedPrice(users)
273         marketPr = fixed
274         fold(marketPr,month,users)
275         if month == 7:
276             #print(marketPr)
277             for u in users:
278                 if u.bandwidthBad > 0:
```

```
                    pass#u.show()
        month += 1
    #return(users)
    return(month)

def test():
    dummyPrice = 2
    dummy = User(19.47245737881847)
    dummy.generation = 20.2
    dummy.bandwidthBad = 0
    dummy.bandwidthGood = 0
    #print(dummy.powerNeed())
    print(payoffs(1,dummy,dummyPrice,0))
    print(payoffs(1,dummy,dummyPrice,1))
    print(payoffs(1,dummy,dummyPrice,2))
    print(payoffs(1,dummy,dummyPrice,3))
    print("fs",traceChoice(userChoice(1,dummy,dummyPrice,False)))
    print("tr",traceChoice(userChoice(1,dummy,dummyPrice,True)))
    users = sampleGen()
    #users = gen()
    #for u in users:
        #print(u.uid,u.consumption)
    #stopped = run(users)
    #for s in stopped:
        #print(s.uid,s.consumption,s.generation,
        #      s.bandwidthGood,s.bandwidthBad)
    #print("Result:",run (users))
    #print(newMarketPrice(3,[dummy]))

def generate():
    return [ str(genPowers())+"\n" for _ in range(20) ]

def solve(dataset):
    return str(run(makeUsers(eval(dataset))))

def check(reply,clue):
    their = eval(reply.strip())
    our = eval(clue.strip())
    return our == their


print(solve(sys.stdin.read()))
```

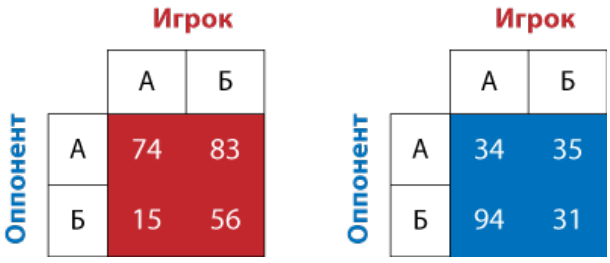
3.3. Задание 3

Задача 3.3.1. Мысленный аукцион (8 баллов)

Матрица игры

Основное понятие теории игр — это матрица игры. Это, по сути, просто таблица, в которой записаны выигрыши игроков. Чтобы разобраться, что это такое, и что такое в понимании математиков «игра», можно, например, прочитать статью Википедии «Дилемма заключённого».

В этой задаче игры всегда асимметричные, поэтому удобно для каждого игрока выписывать свою матрицу игры, например:



Здесь слева — выигрыши «Игрока», а справа — выигрыши «Оппонента»

Обратите внимание, что из-за асимметричности игры, если участников поменять местами, это повлияет на результат игры!

Турниры и стратегии

Очень часто в экономике и эволюционной биологии используется понятие итеративных игр. Это игры, которые повторяются помногу раз, и участники могут запоминать, что делал их «оппонент» в прошлый раз и адаптировать своё поведение. Выигрыш в таком турнире равен сумме выигрышей в каждой игре.

В нашей задаче присутствуют самые простые варианты таких стратегий. Мы для простоты будем использовать терминологию из дилеммы заключённого, и вариант «Б» в матрице игры мы будем считать «предательством». Но при этом неважно, является ли он «предательством» на самом деле, и является ли игра дилеммой заключённого вообще – мы это сделаем для простоты анализа ситуации.

Итак, наши стратегии:

«Всегда А»

Независимо от действий оппонента, выбираем «А»

«Всегда Б»

Независимо от действий оппонента, выбираем «Б»

«Око за око»

Если оппонент в прошлый раз выбрал «Б», выбираем «Б», иначе выбираем «А»

«Минимакс»

Выбираем такой вариант, который обеспечивает наибольший гарантированный выигрыш вне зависимости от действий оппонента. Если варианты одинаковы, выбираем «А».

Название «минимакс» происходит от её описания: сначала максимизируем свой выигрыш действиями оппонента, а затем минимизируем своими. Правда, в традиционной теории игры «выигрыш» — это штраф, который надо минимизировать. У нас же наоборот.

«Жадность»

Выбираем такой вариант, в котором может случиться наибольший выигрыш. Если варианты одинаковы, выбираем «А».

«Щедрость»

Так же, как «Жадность», но мы максимизируем выигрыш для оппонента. Если варианты одинаковы, выбираем «А».

«Око за два ока»

Если оппонент выбрал «Б» оба последних раза, выбираем «Б». Иначе выбираем «А».

«Реванш»

Выбираем такой вариант, который «побивает« предыдущий ход оппонента — мы максимизируем наш выигрыш в том случае, если оппонент повторит ход. Если варианты одинаковы, выбираем «А». В первый ход действуем как «Минимакс».

Все наши стратегии довольно «глупые» (на самом деле, «простые»), но даже такие, будучи посажены вместе, создают массу неожиданных эффектов.

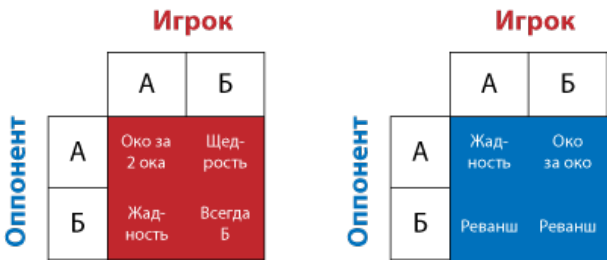
Условие задачи

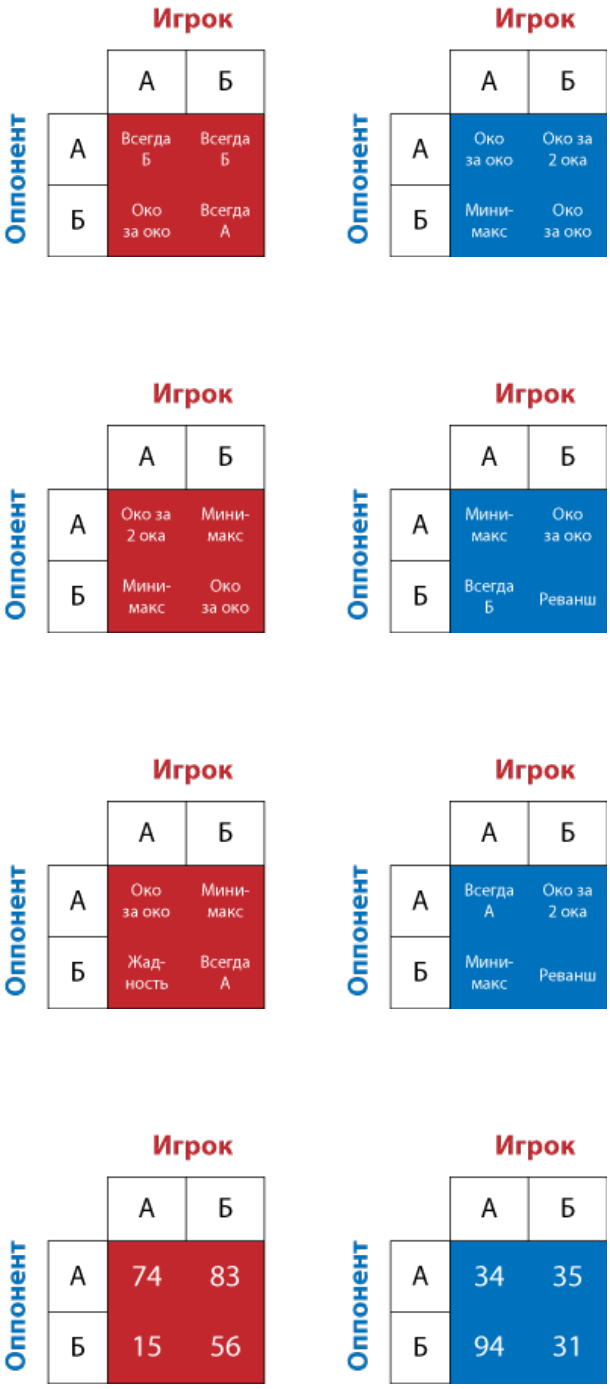
У вас есть несколько стратегий для итеративного турнира в несимметричную бинарную игру (а-ля дилемма заключённого):

1. Всегда А
2. Всегда Б
3. Око за око
4. Минимакс
5. Жадность
6. Щедрость
7. Око за 2 ока
8. Реванш

Один турнир всегда длится 20 ходов, и стратегии, естественно, не знают и не ожидают, когда оканчивается турнир. Второй вариант стратегии всегда считают «предательством» независимо от реального содержимого матрицы игры. Вам даётся матрица игры в которой «выигрыши» — это стратегии для использования в следующем турнире, в котором «выигрыши» — это стратегии для использования в следующем турнире...

Итого у вас есть пять турниров; в матрице последнего — нормальные очки, в матрицах остальных — стратегии из набора, например:





Чтобы играть в таком «метатурнире» можно использовать те же самые стратегии, и для каждой из них можно вычислить математическое ожидание размера её выигрыша в случае игры против случайной стратегии в случайной позиции («Оппонент» или «Игрок»). Играть сами с собой стратегии тоже могут.

Если мы попробуем на этом основании определить стратегию, которая зарабатывает больше всего, то выяснится, что очень часто она будет делить пьедестал с другими стратегиями.

Вам нужно по заданным характеристикам турниров определить, сколько стратегий будут делить первое место.

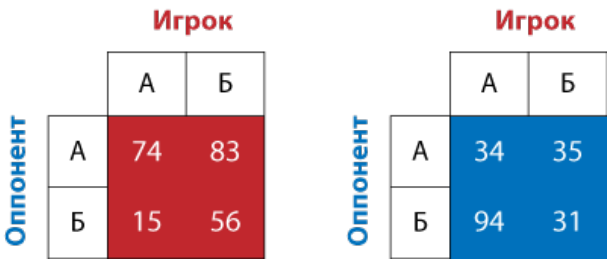
Формат входных данных

Пять массивов: игра 1-го уровня, 2-го, 3-го, 4-го и, наконец, игра с очками.

Одна игра — это массив из четырёх пар:

1. Выигрыши игрока и оппонента в случае, если оба выбрали «А»
2. Выигрыши игрока и оппонента в случае, если игрок выбрал «А», а оппонент — «Б»
3. Выигрыши игрока и оппонента в случае, если игрок выбрал «Б», а оппонент — «А»
4. Выигрыши игрока и оппонента в случае, если оба выбрали «Б»

Например, массив [(74, 34), (15, 94), (83, 35), (56, 31)] превращается в матрицы игры:



Стратегии закодированы числами от 0 до 7 в том же порядке, в котором они шли в условии:

- 0 — Всегда А
- 1 — Всегда Б
- 2 — Око за око
- 3 — Минимакс
- 4 — Жадность
- 5 — Щедрость
- 6 — Око за 2 ока
- 7 — Реванш

Пример входных данных

[[(6, 0), (7, 0), (0, 1), (0, 6)], [(2, 7), (5, 2), (1, 6), (5, 3)], [(0, 7), (6, 1), (3, 3), (2, 2)], [(4, 2), (5, 1), (6, 4), (0, 1)], [(74, 34), (15, 94), (83, 35), (56, 31)]]

Пример выходных данных

5

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Задача решается снизу вверх: результатом турнира «нижнего уровня» являются очки, которые являются наградой за один тур вышестоящего турнира. После вычисления всех возможных наград для него, можно вычислить и его результат, который снова является наградой за один тур следующего турнира. Таким образом, во всех матрицах, включая верхнюю, стратегии внутри матриц можно заменить на очки, после чего просто сравнить эффективность стратегий турнире по верхней матрице. Ограничения вычислительных ресурсов несущественны.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import copy
import sys

random.seed(796)

def genLow():
    result=[]
    for _ in range(4):
        a = round(100*random.random())
        b = round(100*random.random())
        result.append((a,b))
    return result

def genHigh():
    result=[]
    for _ in range(4):
        a = random.randrange(8)
        b = random.randrange(8)
        result.append((a,b))
    return result

def genLG():
    return game(genLow())

def genHG():
    return game(genHigh())

def genTourStr():
    l = genLow()
    h1 = genHigh()
    h2 = genHigh()
    h3 = genHigh()
    h4 = genHigh()
    return [h4,h3,h2,h1,l]

class game:
    aa = 0
    ab = 0
    ba = 0
```

```
    bb = 0
    def __init__(self,values):
        self.aa = values[0]
        self.ab = values[1]
        self.ba = values[2]
        self.bb = values[3]

zeroGame = game([(0,0),(0,0),(0,0),(0,0)])

class tour:
    highs = []
    lows = copy.deepcopy(zeroGame)
    def __init__(self,values):
        self.highs = []
        self.lows = copy.deepcopy(zeroGame)
        self.lows = values[0]
        for v in (values[1:]):
            self.highs.append(v)

def prettyTour(tour):
    print("TOURNAMENT")
    for x in range(len(tour.highs)):
        print("tour",x)
        prettyGame(tour.highs[x])
    print("low-level")
    prettyGame(tour.lows)

dummyLowGame = game([(61,90),(3,87),(97,14),(76,29)])

def prettyGame(g):
    print("AA",g.aa)
    print("AB",g.ab)
    print("BA",g.ba)
    print("BB",g.bb)

def flip(x):
    a,b = x
    return (b,a)

def flipGame(source):
    result = copy.deepcopy(zeroGame)
    result.aa = flip(source.aa)
    result.ab = flip(source.ba)
    result.ba = flip(source.ab)
    result.bb = flip(source.bb)
    return result

def scoreGame(game,s1,s2):
    if s1 and s2:
        return game.aa
    if s1 and (not s2):
        return game.ab
    if (not s1) and s2:
        return game.ba
    if (not s1) and (not s2):
        return game.bb

strDummyTour = str(
    [ [(2,1),(5,0),(4,2),(0,5)]
    , [(5,4),(3,0),(2,5),(3,5)]
```

```
    , [(6,0),(2,2),(2,0),(7,5)]
    , [(2,7),(6,0),(6,1),(1,2)]
    , [(61,90),(3,87),(97,14),(76,29)]
    ])
def fromStrTour(t):
    values = eval(t)
    #print("LEN1",len(values))
    games = [ game(v) for v in reversed(values) ]
    #print("LEN2",len(games))
    #for g in games:
        #print("####")
        #prettyGame(g)
        #print("####")
    result = tour(games)
    #print("LEN3",len(result.highs))
    return tour(games)

#dummyHighGame1 = Game {aa = (Eye,Rematch), ab = (Eye2,AllA),
#                       ba = (Eye2,AllB),   bb = (AllB,Eye)}
dummyHighGame1 = game([(2,7),(6,0),(6,1),(1,2)])
#dummyHighGame2 = Game {aa = (Eye2,AllA),   ab = (Eye,Eye),
#                       ba = (Eye,AllA),    bb = (Rematch,Generous)}
dummyHighGame2 = game([(6,0),(2,2),(2,0),(7,5)])
#dummyHighGame3 = Game {aa = (Generous,Greedy), ab = (MinMax,AllA),
#                       ba = (Eye,Generous),    bb = (MinMax,Generous)}
dummyHighGame3 = game([(5,4),(3,0),(2,5),(3,5)])
#dummyHighGame4 = Game {aa = (Eye,AllB),        ab = (Generous,AllA),
#                       ba = (Greedy,Eye),      bb = (AllA,Generous)}
dummyHighGame4 = game([(2,1),(5,0),(4,2),(0,5)])
dummyTour = tour([dummyLowGame,dummyHighGame1,dummyHighGame2,
                                dummyHighGame3,dummyHighGame4])

def choice(strat,game,history):
    if strat == 0: #All A
        return True
    if strat == 1: #All B
        return False
    if strat == 2: #Eye for eye
        if len(history) == 0:
            return True
        else:
            return history[-1]
    if strat == 3: #MinMax, most guaranteed
        worstA = min(game.aa[0],game.ab[0])
        worstB = min(game.ba[0],game.bb[0])
        return worstA >= worstB
    if strat == 4: #Greedy, max possible
        bestA = max(game.aa[0],game.ab[0])
        bestB = max(game.ba[0],game.bb[0])
        return bestA >= bestB
    if strat == 5: #Generous, max possible for opponent
        bestA = max(game.aa[1],game.ab[1])
        bestB = max(game.ba[1],game.bb[1])
        return bestA >= bestB
    if strat == 6: #Eye for two eyes
        if len(history) < 2:
            return True
        else:
            return history[-1] or history[-2]
    if strat == 7: #Rematch, maximum for last round
```

```
161         if len(history) == 0:
162             return choice(3,game,history)
163         else:
164             if history[-1]:
165                 return game.aa[0] >= game.ba[0]
166             else:
167                 return game.ab[0] >= game.bb[0]
168 
169 def simpleGame(game,s1,s2):
170     history1 = []
171     history2 = []
172     score1 = 0
173     score2 = 0
174     game1 = game
175     game2 = flipGame(game)
176     for _ in range(20):
177         c1 = choice(s1,game1,history2)
178         c2 = choice(s2,game2,history1)
179         sc1,sc2 = scoreGame(game1,c1,c2)
180         history1.append(c1)
181         history2.append(c2)
182         score1 += sc1
183         score2 += sc2
184     return(score1,score2)
185 
186 def foldGame(gameH,gameL):
187     result = copy.deepcopy(zeroGame)
188     result.aa = simpleGame(gameL,gameH.aa[0],gameH.aa[1])
189     result.ab = simpleGame(gameL,gameH.ab[0],gameH.ab[1])
190     result.ba = simpleGame(gameL,gameH.ba[0],gameH.ba[1])
191     result.bb = simpleGame(gameL,gameH.bb[0],gameH.bb[1])
192     return result
193 
194 def playTournament(t,s1,s2):
195     tour = copy.deepcopy(t)
196     effective = copy.deepcopy(tour.lows)
197     #prettyGame(effective)
198     for g in tour.highs:
199         #print("LEN",len(tour.highs))
200         effective = foldGame(g,effective)
201         #print("folding...")
202         #prettyGame(effective)
203     return simpleGame(effective,s1,s2)
204 
205 def fullTournament(tour,s):
206     t = copy.deepcopy(tour)
207     result1 = 0
208     result2 = 0
209     for o in range(8):
210         result1 += playTournament(t,s,o)[0]
211     #print(1,result1)
212     for o in range(8):
213         result2 += playTournament(t,o,s)[1]
214     #print(2,result2)
215     return result1 + result2
216 
217 def ans(tour):
218     t = copy.deepcopy(tour)
219     result = 0
220     best = 0
```

```
221     for x in range(8):
222         que = fullTournament(t,x)
223         #print("QUE",x,que)
224         if que > best:
225             best = que
226             result = 1
227         elif que == best:
228             result += 1
229     #print("-----",best)
230     return result
231 
232 def test():
233     g = dummyLowGame
234     print(simpleGame(g,2,7))
235     print("~~~~~~~~~~")
236     print(scoreGame(g,True,True))
237     print("~~~~~~~~~~")
238     prettyGame(flipGame(dummyLowGame))
239     print("~~~~~~~~~~")
240     prettyGame(foldGame(dummyHighGame1,dummyLowGame))
241     print("FF",simpleGame(foldGame(dummyHighGame4,dummyLowGame),3,0))
242     print("-")
243     prettyGame(foldGame(dummyHighGame2,dummyLowGame))
244     print("-")
245     prettyGame(foldGame(dummyHighGame3,dummyLowGame))
246     print("-")
247     prettyGame(foldGame(dummyHighGame4,dummyLowGame))
248     #print("~~~~~~~~~~")
249     #prettyTour(dummyTour)
250     print("~~~~~~~~~~")
251     test4_2=foldGame(dummyHighGame3,foldGame(dummyHighGame4,dummyLowGame))
252     prettyGame(test4_2)
253     print("~~~~~~~~~~")
254     for x in range(8):
255         for y in range(8):
256             print(x,y,playTournament(dummyTour,x,y))
257     print("~~~~~~~~~~")
258     print(fullTournament(dummyTour,2))
259     #prettyTour(flipTournament(dummyTour))
260     print(ans(dummyTour))
261     print(ans(fromStrTour(strDummyTour)))
262     for x in generate():
263         print(ans(fromStrTour(x)))
264 
265 def generate():
266     return [ str(genTourStr()) for _ in range(30) ]
267 
268 def solve(dataset):
269     return str(ans(fromStrTour(dataset)))
270 
271 def check(reply,clue):
272     their = eval(reply)
273     ours = eval(clue)
274     if their == ours:
275         return True
276     else:
277         return False
278 
279 
280 print(solve(sys.stdin.read()))
```

3.4. Задание 4

Задача 3.4.1. Светлое наследие Теслы (2 балла)

Никола Тесла мечтал передавать энергию по воздуху с помощью особых башен (посмотрите на башню Ворденклиф, чтобы было легче представить), создающих высокое напряжение, которое затем передаётся без проводов, прямо по воздуху в приёмники.

Вам поручены работы по проекту строительства подобной башни. Она должна обеспечивать электроэнергией небольшое поселение, каждое из зданий которого будет оборудовано приёмником. Эффективная площадь антенны каждого приёмника 2 квадратных метра (это площадь, которую бы имел идеальный приёмник для приёма той же мощности, что и реальный), и для простоты будем считать, что она не зависит от расстояния до приёмника. Планируемая высота башни обеспечивает подъём на 50 м над приёмниками.

Известны координаты будущих приёмников на плоскости (для удобства переведённые в локальную декартову систему). Нужно определить координаты для постройки башни, выполняющие следующие условия:

- Мощность излучателя минимальна.
- Все приёмники принимают не менее 1 кВт.

Принимаемая приёмниками мощность обратно пропорциональна квадрату расстояния до излучателя.

Часть 1

В этой части нужно вывести только координаты башни.

За эту часть начисляется 1,3 балла.

Часть 2

В этой части нужно вывести только мощность башни.

За эту часть начисляется 0,7 балла.

Формат входных данных

На первой строке целое число приёмников N, затем N строк с вещественными числами x_i и y_i через пробел. Это координаты приёмников. Единица измерения — метр.

Формат выходных данных

Часть 1

Два вещественных числа x_c, y_c через пробел — координаты башни.

Ответ считается верным, если вычисленные координаты отдалены от эталонных менее, чем на 1 условный метр

Число приёмников изменяется от 4 до 20.

Координаты изменяются от -1000 до 1000.

Стандартный ввод
4 1.2 -1.3 -501.33 1281.41 -1 -1 1 1
Стандартный вывод
0.0001 10.000

Часть 2

Вещественное число, мощность излучателя в кВт.

Стандартный ввод
4 1.2 -1.3 -501.33 1281.41 -1 -1 1 1
Стандартный вывод
250200.002

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Задачу можно решить как методом скорейшего спуска, так и алгебраически: достаточно, например, перебрать все тройки точек из множества входящих в выпуклую границу всех точке. Для каждой из трёх точек нужно проверить, что описанная вокруг их треугольника окружность удовлетворяет всем ограничениям условия, и найти минимальную. Её центр является ответом на первую часть. Для получения ответа ко второй части нужно найти расстояние от передатчика до самого крайнего приёмника — это гипотенуза прямоугольного треугольника с катетами, равными 50 метрам и радиусу найденной ранее окружности. Далее нужно найти площадь сферы с радиусом в найденное расстояние от передатчика до самого удалённого приёмника, и исходя из того, что 2 квадратных метра этой площади несут мощность в 1кВт, найти мощность, распределённую по всей сфере. Это и есть ответ второй части. Ограничения вычислительных ресурсов несущественны.

Пример программы-решения

Часть 1

Ниже представлено решение на языке Python3

```
import fileinput
import random
from math import sqrt, pi
import sys

k = 0.4
eff = 5

def part (d):
    sph = 4 * pi * d ** 2
    return eff / sph * k

def dist(X,Y,x,y):
    return sqrt((X-x)**2+(Y-y)**2)

def distH(X,Y,x,y):
    planar = dist(X,Y,x,y)
    spatial = dist(50,0,0,planar)
    return spatial

def powerPart(X,Y,x,y):
    return part((distH(X,Y,x,y)))

def circum1(Ax,Ay,Bx,By,Cx,Cy):
    a = sqrt((Ax-Bx)**2 + (Ay-By)**2)
    b = sqrt((Bx-Cx)**2 + (By-Cy)**2)
    c = sqrt((Cx-Ax)**2 + (Cy-Ay)**2)
    p = (a+b+c)/2
    r = a * b * c / 4 / sqrt ( p*(p-a)*(p-b)*(p-c) )
    x12 = Ax-Bx
    x23 = Bx-Cx
    x31 = Cx-Ax
    y12 = Ay-By
    y23 = By-Cy
    y31 = Cy-Ay
    z1 = Ax**2 + Ay**2
    z2 = Bx**2 + By**2
    z3 = Cx**2 + Cy**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 circum2(Ax,Ay,Bx,By,Cx,Cy):
    D = 2*(Ax*(By-Cy)+Bx*(Cy-Ay)+Cx*(Ay-By))
    Ux = ((Ax**2+Ay**2)*(By-Cy)+(Bx**2+By**2)*(Cy-Ay)+(Cx**2+Cy**2)*(Ay-By))/D
    Uy = ((Ax**2+Ay**2)*(Cx-Bx)+(Bx**2+By**2)*(Ax-Cx)+(Cx**2+Cy**2)*(Bx-Ax))/D
    return dist(Ux,Uy,Bx,By),Ux,Uy

def ctest():
    ax = random.random()
    ay = random.random()
    bx = random.random()
    by = random.random()
```

```
57      cx = random.random()
58      cy = random.random()
59      d1,rx,ry = circum1(ax,ay,bx,by,cx,cy)
60      d2,ux,uy = circum2(ax,ay,bx,by,cx,cy)
61      if rx == ux and ry == uy:
62          test()
63      else:
64          print("1da",dist(ax,ay,rx,ry))
65          print("1db",dist(bx,by,rx,ry))
66          print("1dc",dist(cx,cy,rx,ry))
67          print("2da",dist(ax,ay,ux,uy))
68          print("2db",dist(bx,by,ux,uy))
69          print("2dc",dist(cx,cy,ux,uy))
70          print("A",ax,ay)
71          print("B",bx,by)
72          print("C",cx,cy)
73          print("R:",rx,ry)
74          print("U:",ux,uy)
75
76  #ctest()
77
78  def covers(points,i1,i2,i3):
79      Ax,Ay = points[i1]
80      Bx,By = points[i2]
81      Cx,Cy = points[i3]
82      R,Ux,Uy = circum1(Ax,Ay,Bx,By,Cx,Cy)
83      for p in range(len(points)):
84          if p == i1 or p == i2 or p == i3:
85              pass
86          else:
87              x,y = points[p]
88              if dist(Ux,Uy,x,y) > R:
89                  return (False,None,None,None)
90      return (True,R,Ux,Uy)
91
92  def findSmallestCover(points):
93      best = float("inf")
94      bX = None
95      bY = None
96      for i in range(len(points)-2):
97          for j in range(i+1,len(points)-1):
98              for k in range(j+1,len(points)):
99                  que,d,x,y = covers(points,i,j,k)
100                 #print("-> ",que,d,x,y)
101                 if que:
102                     if best > d:
103                         best = d
104                         bX, bY = x,y
105     return bX,bY,best
106
107 def ans(points):
108     a,b,d = findSmallestCover(points)
109     power = 1e3 / powerPart(0,0,0,d)
110     return (a,b,power)
111
112 def readInput():
113     count = None
114     points = []
115     for line in fileinput.input():
116         if fileinput.isfirstline():
```

```
            count = eval(line)
        else:
            x,y = line.split()
            points.append((float(x),float(y)))
    return points

def test():
    points = readInput()
    x,y,d = findSmallestCover(points)
    print(x,y)

#test()

def readDataSet(dataset):
    count = None
    points = []
    for line in dataset.splitlines():
        #print(line)
        if count == None:
            count = eval(line)
        else:
            x,y = line.split()
            points.append((float(x),float(y)))
    return points

def generate():
    tests = []
    for _ in range(20):
        count = random.randint(4,20)
        result = "{}\n".format(count)
        for _ in range(count):
            x = random.random() * 2000 - 1000
            y = random.random() * 2000 - 1000
            result += "{} {}\n".format(x,y)
        tests.append(result)
    return tests

def solve(dataset):
    points = readDataSet(dataset)
    x,y,p = ans(points)
    return "{}".format(p)

def check(reply,clue):
    p = float(reply)
    P = float(clue)
    if abs(p-P)>=10:
        return False, "Мощность рассчитана неверно"
    return True

print(solve(sys.stdin.read()))
```

Часть 2

Ниже представлено решение на языке Python3

```
import fileinput
import random
from math import sqrt, pi
import sys
```

```

k = 0.4
eff = 5

def part (d):
    sph = 4 * pi * d ** 2
    return eff / sph * k

def dist(X,Y,x,y):
    return sqrt((X-x)**2+(Y-y)**2)

def distH(X,Y,x,y):
    planar = dist(X,Y,x,y)
    spatial = dist(50,0,0,planar)
    return spatial

def powerPart(X,Y,x,y):
    return part((distH(X,Y,x,y)))

def circum1(Ax,Ay,Bx,By,Cx,Cy):
    a = sqrt((Ax-Bx)**2 + (Ay-By)**2)
    b = sqrt((Bx-Cx)**2 + (By-Cy)**2)
    c = sqrt((Cx-Ax)**2 + (Cy-Ay)**2)
    p = (a+b+c)/2
    r = a * b * c / 4 / sqrt ( p*(p-a)*(p-b)*(p-c) )
    x12 = Ax-Bx
    x23 = Bx-Cx
    x31 = Cx-Ax
    y12 = Ay-By
    y23 = By-Cy
    y31 = Cy-Ay
    z1 = Ax**2 + Ay**2
    z2 = Bx**2 + By**2
    z3 = Cx**2 + Cy**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 circum2(Ax,Ay,Bx,By,Cx,Cy):
    D = 2*(Ax*(By-Cy)+Bx*(Cy-Ay)+Cx*(Ay-By))
    Ux = ((Ax**2+Ay**2)*(By-Cy)+(Bx**2+By**2)*(Cy-Ay)+(Cx**2+Cy**2)*(Ay-By))/D
    Uy = ((Ax**2+Ay**2)*(Cx-Bx)+(Bx**2+By**2)*(Ax-Cx)+(Cx**2+Cy**2)*(Bx-Ax))/D
    return dist(Ux,Uy,Bx,By),Ux,Uy

def ctest():
    ax = random.random()
    ay = random.random()
    bx = random.random()
    by = random.random()
    cx = random.random()
    cy = random.random()
    d1,rx,ry = circum1(ax,ay,bx,by,cx,cy)
    d2,ux,uy = circum2(ax,ay,bx,by,cx,cy)
    if rx == ux and ry == uy:
        test()
    else:
        print("1da",dist(ax,ay,rx,ry))
```

```
65          print("1db",dist(bx,by,rx,ry))
66          print("1dc",dist(cx,cy,rx,ry))
67          print("2da",dist(ax,ay,ux,uy))
68          print("2db",dist(bx,by,ux,uy))
69          print("2dc",dist(cx,cy,ux,uy))
70          print("A",ax,ay)
71          print("B",bx,by)
72          print("C",cx,cy)
73          print("R:",rx,ry)
74          print("U:",ux,uy)
75
76  #ctest()
77
78  def covers(points,i1,i2,i3):
79      Ax,Ay = points[i1]
80      Bx,By = points[i2]
81      Cx,Cy = points[i3]
82      R,Ux,Uy = circum1(Ax,Ay,Bx,By,Cx,Cy)
83      for p in range(len(points)):
84          if p == i1 or p == i2 or p == i3:
85              pass
86          else:
87              x,y = points[p]
88              if dist(Ux,Uy,x,y) > R:
89                  return (False,None,None,None)
90      return (True,R,Ux,Uy)
91
92  def findSmallestCover(points):
93      best = float("inf")
94      bX = None
95      bY = None
96      for i in range(len(points)-2):
97          for j in range(i+1,len(points)-1):
98              for k in range(j+1,len(points)):
99                  que,d,x,y = covers(points,i,j,k)
100                 #print("-> ",que,d,x,y)
101                 if que:
102                     if best > d:
103                         best = d
104                         bX, bY = x,y
105     return bX,bY,best
106
107 def ans(points):
108     a,b,d = findSmallestCover(points)
109     power = 1e3 / powerPart(0,0,0,d)
110     return (a,b,power)
111
112 def readInput():
113     count = None
114     points = []
115     for line in fileinput.input():
116         if fileinput.isfirstline():
117             count = eval(line)
118         else:
119             x,y = line.split()
120             points.append((float(x),float(y)))
121     return points
122
123 def test():
124     points = readInput()
```

```
125     x,y,d = findSmallestCover(points)
126     print(x,y)
127 
128 #test()
129 
130 def readDataSet(dataset):
131     count = None
132     points = []
133     for line in dataset.splitlines():
134         #print(line)
135         if count == None:
136             count = eval(line)
137         else:
138             x,y = line.split()
139             points.append((float(x),float(y)))
140     return points
141 
142 def generate():
143     tests = []
144     for _ in range(20):
145         count = random.randint(4,20)
146         result = "{}\n".format(count)
147         for _ in range(count):
148             x = random.random() * 2000 - 1000
149             y = random.random() * 2000 - 1000
150             result += "{} {}\n".format(x,y)
151         tests.append(result)
152     return tests
153 
154 def solve(dataset):
155     points = readDataSet(dataset)
156     x,y,p = ans(points)
157     return "{} {}".format(x,y)
158 
159 def check(reply,clue):
160     x,y = [ float(q) for q in reply.split()]
161     X,Y = [ float(q) for q in clue.split()]
162     if dist(X,Y,x,y) > 1:
163         return False, "Неверные координаты башни"
164     return True
165 
166 print(solve(sys.stdin.read()))
```

3.5. Задание 5

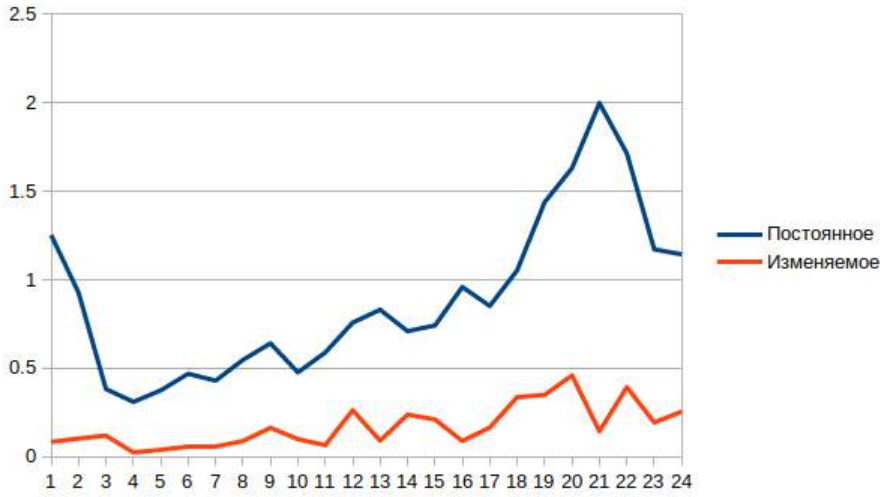
Задача 3.5.1. Принудительная добровольность (5 баллов)

У некоторой энергетической компании есть «головная боль» в виде загородного посёлка, сети в котором очень сильно перегружены, что приводит к большим издержкам на их обслуживание. Модернизация сетей в посёлке обойдётся очень дорого и в обозримом будущем не окупится. Компания пытается улучшить ситуацию, предложив потребителям в посёлке такие тарифы, при которых и пиковое потребление электроэнергии снизится, и потребители с радостью на них перейдут, потому что станут платить меньше. Вам предстоит рассчитать такие тарифы.

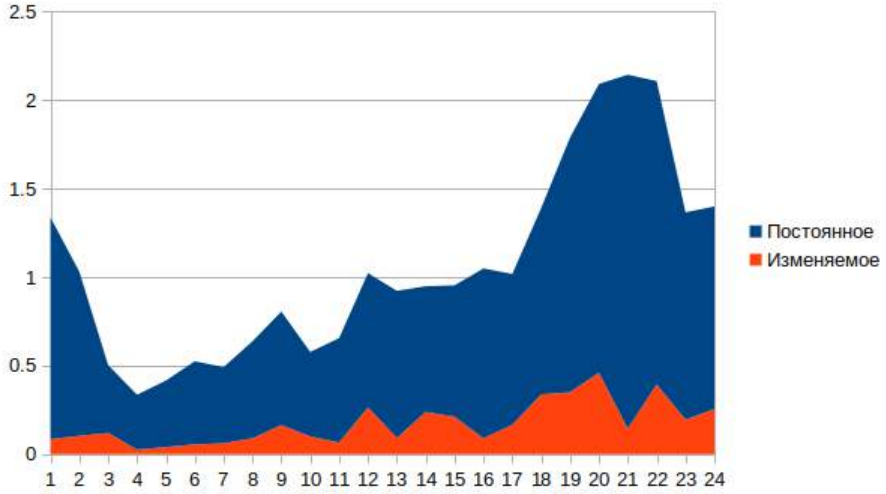
У вас имеется модель «типичного» потребителя в доме данного посёлка, у ко-

торого имеется «типичный» график почасового потребления электроэнергии.Часть потребления постоянна и от тарифа не зависит, а часть может изменяться. В нашей модели, для простоты, изменение состоит в том, что график изменяемого потребления может смещаться во времени на произвольное число часов.

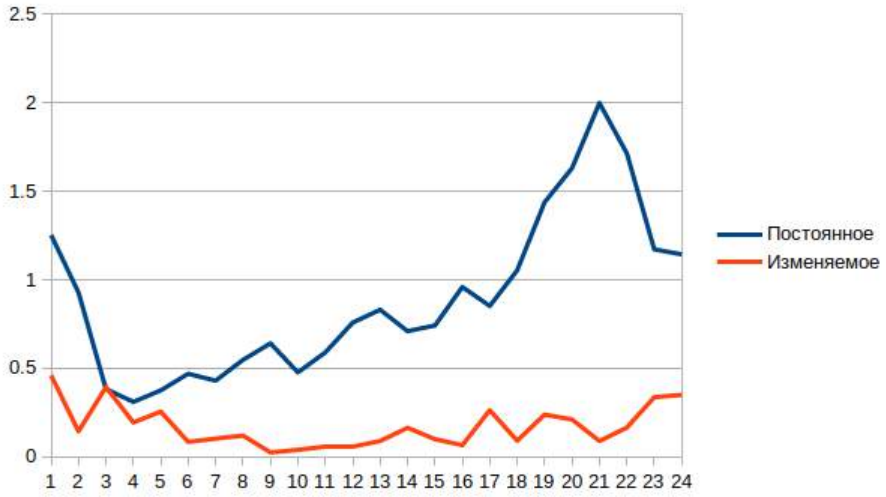
Например, если графики потребления выглядят так:

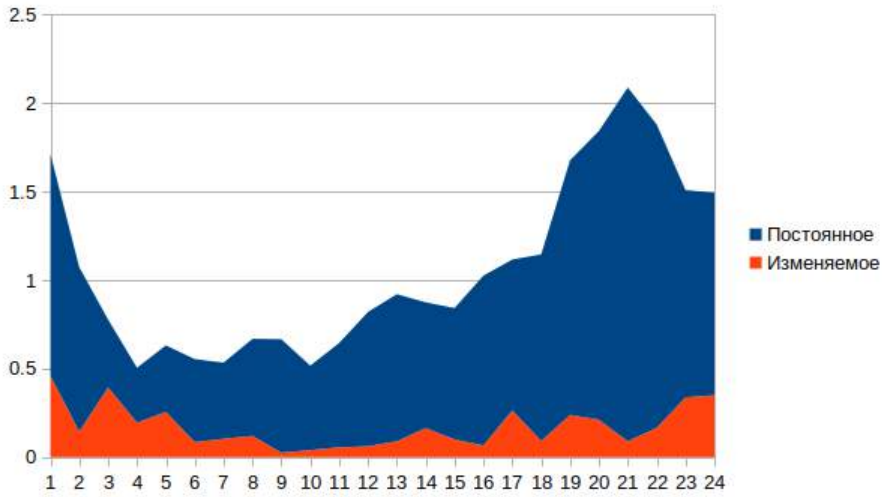


...и для суммарного потребления — так (здесь синий и красный графики с картинки выше сложены):



...то после смещения переменной составляющей потребления на 5 часов, эти графики превратятся в такие:





Владелец дома платит за электроэнергию по двухзонному тарифу: с 12 часов (включительно) до 16 (не включительно) и с 23 (включительно) до 6 (не включительно) часов цена составляет 3 р./кВт·ч, в остальное время — 6. По данным из примера (на следующей странице) можно подсчитать, что модельный потребитель платит за электроэнергию 45 546,89 р./год (за 365 дней).

- Каждый кВт пиковой мощности (максимальной мощности за день) приносит компании 40 000 р. убытков в год.
- Потребитель без проблем согласится перейти на новый тариф только в том случае, если после изменения графика переменной части потребления он будет платить хотя бы на 500 р./год меньше, чем он платит сейчас.
- Если при новых тарифах прибыль компании уменьшится, она на это изменение не пойдёт.
- Если по новому тарифу для потребителя без изменения графика потребления годовая стоимость энергии увеличится менее, чем на 1000 рублей, он не будет изменять своего поведения.
- Если потребитель может изменить график потребления несколькими способами, он сделает это наиболее удобным для себя способом; поскольку этот способ неизвестен, приходится исходить из того, что он будет вести себя наименее выгодным для энергокомпании образом.

Какие цены должна предложить энергокомпания, чтобы максимально увеличить свою прибыль? Если удовлетворяющего заданным ограничениям решения нет, то нужно вывести значения действующего тарифа. Решение состоит из двух чисел — цен за кВт·час ночью-днём и утром-вечером.

Задача может иметь несколько правильных ответов и ещё больше похожих на правильные. Для решения задачи вам нужно привести любой правильный ответ.

Формат входных данных

Два массива, каждый из 24 чисел (почасовые значения потребляемой энергии в кВт·ч за сутки).

Первый массив — постоянная составляющая потребления, второй — переменная. Отсчёт идёт с полуночи.

Формат выходных данных

Пара чисел — тарифы ночью-днём и утром-вечером в копейках за кВт·ч.

Пример входных данных

([1.2532064880926013, 0.9277330171025241, 0.3830082504508084, 0.310412919612098, 0.375401744425605, 0.46924600370281877, 0.43025390820572046, $\cdots$])

Пример выходных данных

Для тарифов в 2р. 53коп. и 8р. 47коп ответ выглядит вот так (это не ответ к примеру входных данных):

(253,847)

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Для решения сначала нужно найти пределы изменения тарифов, которые в принципе могут удовлетворять условию 2 — это изменение цены вечернего тарифа от 0 до действующего, а дневного — от действующего, до такого, при котором условие 2 выполняется, если вечерний тариф равен нулю. Затем для каждой пары тарифов нужно найти прибыль компании, для чего нужно выбирать смещение потребления таким образом, чтобы оно минимизировало эту прибыль (при соблюдении остальных условий задачи). Для выполнения ограничений вычислительных ресурсов нужно мемоизировать нахождение пиков потребления и потреблённых мощностей по разным тарифам в зависимости от смещения потребления.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import sys

content = [ 1.5935004
    , 1.0997878
    , 0.5595438
    , 0.3955478
    , 0.488944
    , 0.6068851
    , 0.62520367
    , 0.7163142
    , 0.78384954
    , 0.69044185
    , 0.82434213
    , 1.1544989
    , 1.193051
    , 1.0579418
    , 0.9862855
    , 1.2018712
    , 1.2106589
    , 1.5743363
    , 1.9298786
    , 2.4665824
    , 2.7173998
    , 2.3626094
    , 1.7456802
    , 1.5448046
    ]

random.seed(45235)
#random.seed(None)

def gen():
    constant = []
    variable = []
    for x in content:
        r1 = random.random() * 0.2 + 0.65
        r2 = random.random() * 0.2 + 0.05
        constant.append(x*r1)
        variable.append(x*r2)
    return constant, variable

defLower = 300
defUpper = 600
clause1Threshold = (500/365)
clause3Threshold = (1000/365)
powerProfit = (40000/365)
searchDepth = 2.5 # Ищем тарифы от 0 до 2.5 x текущий

def shifter(constant,variable,s):
    shifted = []
    for i in range(len(constant)):
        shifted.append(constant[i] + variable[(i+s)%24])
    return shifted

def hourLow(x):
```

```
57      h = x % 24
58      return (h<6) or (h>22) or ((h>11) and (h<16))
59
60  def cacheShifts(constant,variable):
61      result = []
62      for h in range(24):
63          lower = 0
64          higher = 0
65          hourly = shifter(constant,variable,h)
66          for hh in range (0,24):
67              if hourLow(hh):
68                  lower += hourly[hh]
69              else:
70                  higher += hourly[hh]
71          result.append((lower,higher))
72      return result
73
74  def cachePeaks(constant,variable):
75      result = []
76      for sh in range(24):
77          best = 0
78          values = shifter(constant,variable,sh)
79          for v in values:
80              if v > best:
81                  best = v
82          result.append(best)
83      return result
84
85  constant,variable = gen()
86
87  #cached = cacheShifts(constant,variable)
88  #cachedPeaks = cachePeaks(constant,variable)
89
90  def costsOnShift(cached,lower,upper,sh):
91      low,up = cached[sh]
92      return (low*lower + up*upper)
93
94  def currentDaily(cached):
95      return costsOnShift(cached,defLower,defUpper,0)
96  #print(currentDaily)
97
98  def clause1(cached,lower,upper):
99      old = currentDaily(cached)
100     new = old
101     for i in range(24):
102         que = costsOnShift(cached,lower,upper,i)
103         if que < new:
104             new = que
105     return old >= new + clause1Threshold
106
107 #def peak(sh):
108     #return cachedPeaks[sh]
109
110 def profits(cached,cachedPeaks,lower,upper,sh):
111     profitFromPower = ((cachedPeaks[0] - cachedPeaks[sh]) * powerProfit )
112     profitFromPay = costsOnShift(cached,lower,upper,sh) - currentDaily(cached)
113     #print(lower,upper,sh,profitFromPower,profitFromPay)
114     return profitFromPay + profitFromPower
115
116 def sureProfits(cached,cachedPeaks,lower,upper):
```

```
    if ( lower == defLower and upper == defUpper ):
        return 0
    result = 0
    worstProfit = 1000000000
    for i in userAcceptableShifts(cached,lower,upper):
        pr = profits(cached,cachedPeaks,lower,upper,i)
        if pr < worstProfit:
            result = pr
    return result

def clause3(cached,lower,upper):
    old = currentDaily(cached)
    new = costsOnShift(cached,lower,upper,0)
    return new > old + clause3Threshold

def userAcceptableShifts(cached,lower,upper):
    result = []
    for i in range(24):
        if costsOnShift(cached,lower,upper,i) + clause1Threshold \
                                                <= currentDaily(cached):
            result.append(i)
    return result

def clause2(cached,cachedPeaks,lower,upper):
    result = profits(cached,cachedPeaks,lower,upper,0) > 0
    for i in userAcceptableShifts(cached,lower,upper):
        result = result and profits(cached,cachedPeaks,lower,upper,i) > 0
    return result

def evaluateTariffs(cached,cachedPeaks,lower,upper):
    return clause1(cached,lower,upper) \
        and clause2(cached,cachedPeaks,lower,upper) \
        and clause3(cached,lower,upper)

def findTariffs(cached,cachedPeaks):
    result = []
    for lower in range(0,defLower):
        for upper in range (defUpper,defUpper * 3):
            if evaluateTariffs(cached,cachedPeaks,lower,upper):
                result.append((lower,upper))
    return result

def best(cached,cachedPeaks):
    result = (defLower,defUpper,0)
    bestProfit = 0
    for (l,u) in findTariffs(cached,cachedPeaks):
        pr = sureProfits(cached,cachedPeaks,l,u)
        if pr > bestProfit:
            bestProfit=pr
            result=(l,u,bestProfit)
    return result

def generate():
    tests = []
    for _ in range(20):
        q = gen()
        tests.append((str(q)+"\n",q))
    return tests

def solve(dataset):
```

```
177     constant,variable = eval(dataset)
178     cached = cacheShifts(constant,variable)
179     cachedPeaks = cachePeaks(constant,variable)
180     lower,upper,value = best(cached,cachedPeaks)
181     #print(lower, upper, value)
182     return str((lower,upper))
183 
184 def check(reply,clue):
185     theirLow, theirUpper = eval(reply)
186     constant,variable = clue
187     cached = cacheShifts(constant,variable)
188     cachedPeaks = cachePeaks(constant,variable)
189     _,_,ourValue = best(cached,cachedPeaks)
190     theirValue = sureProfits(cached,cachedPeaks,theirLow,theirUpper)
191     return abs(theirValue - ourValue) <= 3
192 
193 def test():
194     g = generate()
195     for i in range(len(g)):
196         print(g[i][0])
197         print(solve(g[i][0]))
198 
199 print(solve(sys.stdin.read()))
```

3.6. Задание 6

Задача 3.6.1. Золотое солнце (1 балл)

Вам даны почасовые данные за 30 дней по яркости солнца (в килолюксах) и потребляемой мощности (в кВт) для некоторого потребителя (всего 1440 чисел).

У потребителя установлена большая солнечная батарея, для которой зависимость мощности от яркости солнца линейна и составляет 1,5 кВт при 10 килолюксах.

Нужно выяснить, сколько денег пользователь сэкономил за счёт солнечной батареи за рассматриваемый месяц.

В течение часа яркость солнца нужно считать постоянной.

Стоимость электроэнергии 10 руб. за кВт·час.

Отдавать излишки энергии в сеть пользователь не может.

Пример входных данных

Это просто пара из двух массивов по 30*24 чисел каждый. Первый массив — данные по яркости солнца, в клк, второй — по потреблению, в кВт.

([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.3457030373754966, 3.0065015553528815, 4.26458504488992, 6.337972841901577, 6.398630888429304, 10.645502850939069, 13.663080234839168, 14.997035523241964, 15.265556118680628, 13.238674356373597, 16.509038655033837, 13.221606012902962, 11.691021062908755, 11.265662725768626, 6.511092267415828, 4.822169622939845, 3.1900099497706904, 0.7378545946770569, ···])

Пример выходных данных

4081.5

(не является ответом к примеру входных данных)

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Для каждого часа экономию можно найти независимо. Затем нужно только сложить экономию по всем часам в данных задачи. Ограничения вычислительных ресурсов несущественны.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import sys

patternS = [0.0,0.0,0.0,0.0,0.0,0.0,1.3,2.75,3.95,5.5,7.5,9.5,11.5,13.5,14.65,15.0,
15.0,13.55,10.75,9.4,7.5,5.5,3.15,0.8]
patternC = [2.05,1.2000000000000002,1.0499999999999998,1.1,1.35,1.55,
1.7000000000000002,2.0,2.3,2.8,3.1500000000000004,2.85,2.05,1.9,1.7000000000000002,
2.05,2.55,3.25,4.15,4.95,6.4,6.3,4.699999999999999,3.45]

random.seed(8457)

cost = 10

def gen():
    sun = []
    house = []
    for _ in range(30):
        sun += [ x * ( 0.8 + 0.4 * random.random()) for x in patternS ]
        house += [ x / 3 * ( 0.8 + 0.4 * random.random()) for x in patternC ]
    return sun, house

def scale(intensity):
    return intensity / 10 * 1.5

def hourEconomy(sun,cons):
    power = scale(sun)
    return cost * min(power,cons)

def ans(suns,houses):
    #print(sum([h * cost for h in houses]))
    q = list(zip(suns,houses))
    economies = [ hourEconomy(s,c) for s,c in q ]
    return sum(economies)

```

```
#ss,cc = gen()
#print(ans(ss,cc))

def generate():
    result = []
    for _ in range (20):
        test = gen()
        result.append((str(test),test))
    return result

def solve(dataset):
    suns,houses = eval(dataset)
    return str(ans(suns,houses))

def check(reply,clue):
    their = eval(reply)
    s,h = clue
    our = ans(s,h)
    return abs(our-their) <= 1

print(solve(sys.stdin.read()))
```

3.7. Задание 7

Задача 3.7.1. Умная сетка (2.2 баллов)

В жилом доме построена инновационная система энергопотребления, где все квартиры соединены в общую энергосеть и могут обмениваться электроэнергией (купленной, полученной с солнечных батарей, накопленной в энергонакопителе, и т.п.) таким образом, что каждая квартира получает необходимое энергоснабжение. Даже в том случае, когда какой-то квартире энергии не хватает, всегда можно взять энергию из внешней энергосистемы.

Пусть каждая квартира имеет на данный момент излишек или недостаток энергии.

Если у квартиры излишек электроэнергии, то она раздаёт энергию соседям в определённом владельцами квартиры соотношении, выражаемом максимальной долей излишка, которую квартира может передать каждому из соседей. Даже если кто-то из соседей не может принять переданный им излишек (им уже хватает энергии), то он всё равно не перераспределяется.

Если у квартиры недостаток электроэнергии, то она берёт энергию из выделенных её соседями долей, но в пределах, ограниченных соседями, таковы технические ограничения. Если энергии соседей тоже недостаточно, она берёт энергию из внешней энергосети.

По заданным излишкам и недостаткам, а также настройкам соотношений необходимо определить, сколько энергии придётся докупить с внешней сети, чтобы дорогие соседи смогли дальше пить чай и смотреть новости.

Формат входных данных

Два массива. В первом находятся текущие избытки или недостатки энергии у каждого из соседей. Во втором — установленные доли излишков, передаваемых сосе-

дям. Если этот массив назвать foo, то foo[3][4] == 0.5 означает, что сосед в квартире номер 3 передаст соседу в квартире номер 4 не больше половины своего избытка энергии (даже если остальной избыток передать будет некому).

Сумма значений по первому индексу массива всегда равна 1.

Размеры массивов согласованы: если первый имеет размер 100, то второй — 100 на 100.

Число квартир находится в диапазоне от 50 до 150.

Формат выходных данных

Число с плавающей точкой.

Пример входных данных

([25.764437694733104, -19.65574253924039, 23.252448869230683, -10.338138947636265, 12.335993891889927, 6.486647051427877, -0.9219876347859852, ···], [[0.16890061330053335, 0.18540184620782904, ···]])

Пример выходных данных

97.7 (не является ответом к примеру входных данных)

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

Для решения задачи достаточно однократно вычислить, сколько мощности соседи передают друг другу, а затем просуммировать недостатки (если есть) по всем квартирам. Ограничения вычислительных ресурсов несущественны.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import sys

random.seed(945591)

def genRow(count):
    row = [ 0 for _ in range(count) ]
    for i in range(count // 2):
        n = random.randint(0,count-1)
        if i != n:
            row[n] = random.random()
```

```
    total = sum(row)
    return [ x/total for x in row ]

def gen(count):
    affections = [ genRow(count) for _ in range(count) ]
    powers = [ -20 + 50*random.random() for _ in range(count) ]
    return powers, affections

class Power():
    influx = 0
    outflux = 0
    power = 0
    def __init__ ( self, power ):
        self.power = power

def makePower(rawValues):
    return [ Power(x) for x in rawValues ]

def balance(affections,pows):
    powers = pows.copy()
    for i in range(len(powers)):
        p = powers[i]
        if p.power > 0:
            p.outflux = p.power
            for j in range(len(powers)):
                q = powers[j]
                q.influx += affections[i][j] * p.outflux
    return powers

def missingPower(powers):
    return sum([ min(0,p.influx+p.power) for p in powers ])

def excessivePower(powers):
    return sum([ max(0,p.influx+p.power-p.outflux) for p in powers ])

def generate():
    num_tests = 20
    tests = []
    first = gen(10)
    tests.append((str(first)+'\n',first))
    for _ in range(num_tests-1):
        case = gen(random.randint(50,150))
        tests.append((str(case)+"\n",case))
    return tests

def solve(dataset):
    values,affections = eval(dataset)
    answer = missingPower(balance(affections,makePower(values)))
    return str(answer)

def check(reply,clue):
    ours = solve(str(clue))
    return abs(float(reply) - float(ours)) <= 1e10

print(solve(sys.stdin.read()))
```

3.8. Задание 8

Задача 3.8.1. PageRank на минималках (2.4 баллов)

Пользователь читает Википедию по следующему алгоритму:

1. Вначале у него открыта вкладка браузера с главной страницей.
2. Пользователь читает страницу на текущей вкладке и начисляет ей 10 очков рейтинга.
3. Он открывает все ссылки на страницы внутри Википедии, которые есть на странице в текущей вкладке браузера (кроме тех, что ведут на неё саму). Если пользователь натыкается на ссылку, которая уже открыта в какой-то вкладке, он не открывает эту ссылку, но странице, на которую она ведёт, начисляет 1 очко рейтинга. Ссылки открываются в том порядке, в котором они перечислены во входных данных. Затем он закрывает текущую вкладку.
4. Если открытых вкладок больше нет, он возвращается на главную страницу.
5. Если число открытых вкладок < 50, пользователь закрывает текущую вкладку и переходит к п.2
6. Если открыто $>= 50$ вкладок, то пользователь начисляет каждой вкладке, кроме последней, 5 очков и закрывает их.

Затем он переходит к п.2

По заданной структуре графа ссылок Википедии найдите страницу, которой пользователь присвоит наибольший рейтинг.

Если даже вы немного ошибётесь и укажете страницу, рейтинг которой отличается от рейтинга наилучшей меньше, чем на 1%, ответ будет засчитан как правильный.

Формат входных данных

В «Википедии» 500 страниц и примерно 11000 ссылок между ними (страницы могут иногда ссылаться сами на себя).

Каждая страница имеет номер от 0 до 499. Главная страница имеет номер 0.

Каждая ссылка представлена парой (a, b), где a — номер страницы, на которой ссылка расположена, а b — номер страницы, на которую она ведёт. Например, $(0, 1)$ — ссылка с главной страницы на страницу с номером 1.

Формат выходных данных

Номер самой «рейтинговой» страницы, например, 404.

Пример входных данных

[(0, 200), (0, 399), (0, 499), (0, 303), (0, 104), (0, 90), (0, 60), (0, 410), (0, 272), (0, 9), (336, 109), (387, 139), (78, 77), (480, 225), (273, 189), (58, 395), $\cdots$]

Пример выходных данных

404

Ограничения вычислительных ресурсов

Время выполнения программы на сервере не более 15 секунд.

Требуемая память на сервере не более 256 мегабайт.

Решение

В задаче описан алгоритм, который нужно реализовать. Нетривиальным является то, что в нём отсутствует критерий остановки. Его нужно подобрать самостоятельно, например, «на глаз» подобрать константу числа итераций алгоритма, после которых ранжировка лидеров не меняется. Для этого можно использовать самостоятельно сгенерированные примеры. Ограничения вычислительных ресурсов достаточно слабы, чтобы проходили решения, в которых критерий остановки выбран очень консервативно (допускаются решения в 10 раз медленнее авторского решения).

Пример программы-решения

Ниже представлено решение на языке Python3

```
import random
import sys

seed = 544234
random.seed(seed)

pageCount = 500

def genMain():
    result = []
    for _ in range(10):
        target = random.randint(0,pageCount-1)
        result.append((0,target))
    return result

def gen():
    result = genMain()
    for _ in range(10,round(pageCount**1.5)):
        source = random.randint(0,pageCount-1)
        target = random.randint(0,pageCount-1)
        result.append((source,target))
    return result

def viewOnePage(openPages,pagesRanks,wikiStructure):
    #print(openPages)
    pagesRanks[openPages[0]] += 10
    currentPage = openPages[0]
    for (source,destination) in wikiStructure:
        if source == currentPage:
            if ( destination in openPages[1:] ) and ( destination != currentPage ):
                pagesRanks[destination] += 1
```

```
            else:
                openPages.append(destination)
    if len(openPages) < 50:
        if openPages == []:
            resultPages = [0]
        else:
            resultPages = openPages[1:]
    else:
        rankers = openPages[:-1]
        for p in rankers:
            pagesRanks[p] += 5
        resultPages = [ openPages[-1]]
    return resultPages, pagesRanks

def surfNTimes(n,wikiStructure):
    openPages = [0]
    pagesRanks = [ 0 for _ in range(len(wikiStructure)) ]
    for i in range(n):
        opens,ranks = viewOnePage(openPages,pagesRanks,wikiStructure)
        openPages = opens
        pagesRanks = ranks
    return ranks

def findBestPages(ranks):
    tagged = [ (ranks[ix],ix) for ix in range(len(ranks)) ]
    tagged.sort()
    tagged.reverse()
    return tagged[:10]

def test(n):
    random.seed(seed)
    wiki = gen()
    wiki.sort()
    print(n)
    ranks = surfNTimes(n,wiki)
    bests = findBestPages(ranks)
    for b in range(len(bests)):
        print(b, bests[b])

#test(pageCount)
#test(pageCount*3)
#test(pageCount*10)
#test(pageCount*15)

def generate():
    result = []
    for _ in range(20):
        g = gen()
        result.append((str(g),g))
    return result

def solve(dataset):
    wiki = eval(dataset)
    ranks = surfNTimes(5000,wiki)
    bests = findBestPages(ranks)
    rating,best = bests[0]
    return str(best)

def check(reply,clue):
    their = eval(reply)
```

```
92      ranks = surfNTimes(10000,clue)
93      bests = findBestPages(ranks)
94      rating,_ = bests.pop()
95      if ranks[their] >= rating:
96          return True
97      else:
98          return ( ranks[their] / rating ) >= 0.99
99
100 print(solve(sys.stdin.read()))
```