Интеллектуальные энергетические системы

2022/23 учебный год

Второй отборочный этап

Второй этап позволяет очертить область предметных знаний, необходимую для участия в профиле. Второй этап профиля ИЭС включает в себя 6 командных задач, для решения которых достаточно школьных знаний и умений, навыков использовать школьные знания и информационный поиск для решения новых задач. Целью задач второго отборочного этапа является подготовка к практическому командному туру заключительного этапа.

Для участников и наставников разработаны методические рекомендации, которые 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 практического тура заключительного этапа олимпиады по профилю.

На втором отборочном этапе (дистанционном командном) участники формируют команды и загружают общее командное решение. Функций у этого этапа несколько: отбор команд из трёх-пяти участников, способных решать сложные задачи; ознакомление участников с математическими и физическими концепциями, на которых будет построена задача заключительного этапа; предъявление командам требований к уровню программирования, идентичных тем, что будут предъявлены им на заключительном этапе. Во время второго отборочного этапа участники решают задачи, которые отражают часть большой командной задачи заключительного этапа, и знакомятся с базовыми концепциями — работают с вероятностными прогнозами, графами, аукционами, задачами на оптимизацию.

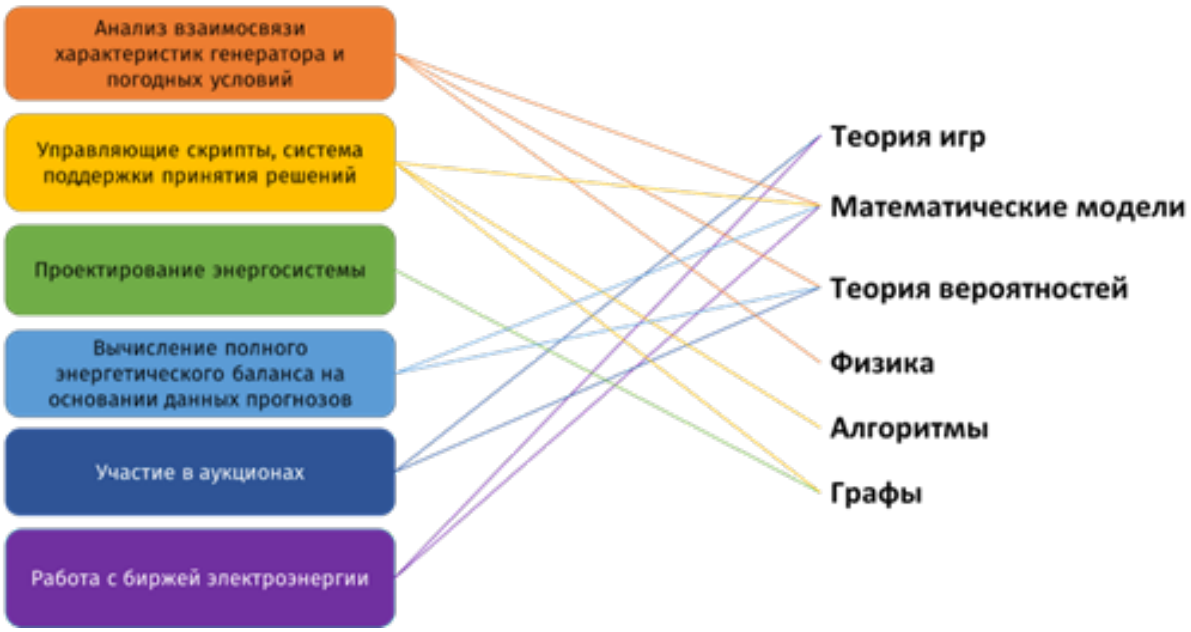


Рис. 1. Связь финальных тем и тем задач второго тура

Теория игр

На финале участникам предстоит на одном поле столкнуться с другими командами, и понимание основ теории игр позволит объективнее оценивать игровую ситуацию в условиях конкуренции за ресурсы.

Математические модели

Работа со сложными математическими моделями и системами — фундаментальный навык, в полной мере раскрывающийся при работе с финальной задачей.

Теория вероятностей

Мир сложен и неустойчив, и финальная задача моделирует в полной мере. Лучше заранее приготовиться и научиться работать с вероятностями, для чего второй этап содержит достаточно много задач по теории вероятностей. При этом для полноценной работы не потребуется погружаться в неё с головой — достаточно знания основ матстатистики и распределений случайных величин, но даже это даст преимущество в работе над финальной задачей.

Физика

В работе с энергетическими системами необходимо знать электротехнику в частности и физику в плане построения и работы с физическими моделями.

Алгоритмы

Финальная задача предполагает написание управляющего скрипта, и здесь важную роль играет навык разработки алгоритмов, равно как и поиска подходящих типовых. На проработку этих навыков и рассчитаны задачи раздела «Алгоритмы». При работе с ними важно делать акцент на информационном поиске и умении выявить типовую подзадачу.

Графы

Энергосети — это графы, и с ними нужно уметь работать. В работе с задачами этого раздела в первую очередь важно овладеть основным арсеналом — программные представления графов и базовые алгоритмы. Это может понадобиться при написании управляющего скрипта, не говоря уже о фундаментальном понимании сетей.

Аукционы

Каждая игровая сессия начинается с аукциона, знания о котором лежат в пересечении теории игр, экономики и психологии. Несмотря на то, что этот этап динамичен и иногда непредсказуем, его результаты влияют на всю остальную игру. Команда, обладающая основами теории аукционов, имеет значительное и глобальное преимущество перед другими.

№ задачи	Знания и навыки, на выявление и развитие которых направлена задача	Задачи инженерного тура заключительного этапа, в которых применимо
IV.1.	Нацелена на развитие навыков промышленного программирования и работы с предметными языками, а также алгоритмического мышления. Необходимы базовые знания по математике. Необходимы средний уровень навыков программирования и знания по информатике по следующим темам: циклы и ветвления, работа с массивами и строками.	Система поддержки принятия решений на аукционе, Написание управляющего скрипта.
IV.2.	Нацелена на развитие навыков работы со сложными математическими моделями. Необходимы знания по математике по следующим темам: иррациональные числа, возведение в степень и квадратный корень, среднее геометрическое. Необходимы базовые навыки программирования и знания по информатике по следующим темам: машинная арифметика, циклы, ветвления, чтение данных.	Проектирование энергосистемы, Анализ взаимосвязи характеристик генератора и погодных условий, Вычисление полного энергетического баланса.
IV.3.	Нацелена на развитие навыков программного анализа данных. Необходимы знания по математике по следующей теме: линейная интерполяция. Знание производных облегчит решение. Из информатики необходим средний уровень навыков программирования и знания по информатике по следующим темам: циклы и ветвления, работа с массивами, чтение данных.	Система поддержки принятия решений на аукционе, Анализ взаимосвязи характеристик генератора и погодных условий, Вычисление полного энергетического баланса на основании данных прогнозов.
IV.5.	Нацелена на развитие навыков работы с графами, а также информационного поиска. Необходимы базовые знания по математике. Из информатики необходимы средний уровень навыков программирования и знания по следующим темам: циклы, ветвления, алгоритмы на графах.	Проектирование энергосистемы, Вычисление полного энергетического баланса на основании данных прогнозов.
??.	Нацелена на развитие навыков работы с вероятностями и численного моделирования. Для решения задачи необходим раздел математики: теория вероятностей. Из информатики необходим средний уровень навыков программирования и знания по информатике по следующим темам: циклы и ветвления, чтение данных, работа с матрицами, численные оптимизационные алгоритмы.	Система поддержки принятия решений на аукционе

№ задачи	Знания и навыки, на выявление и развитие которых направлена задача	Задачи инженерного тура заключительного этапа, в которых применимо
IV.6.	Нацелена на развитие навыков работы с вероятностями и изучение ряда аспектов теории игр. Для решения задачи необходим раздел математики: теория вероятностей. Из информатики необходим базовый уровень навыков программирования и знания по информатике по следующим темам: ветвления, работа с массивами.	Система поддержки принятия решений на аукционе. Работа с биржей электроэнергии

Все задачи второго отборочного этапа решаются на платформе Stepik. В заданное время всем участникам открывается доступ к очередному набору задач (их условиям и проверочной системе). Участники должны написать программу на языке Python 3 и загрузить её текст на сервер. На сервере выполняется тестирование загруженной программы: её ответы сверяются с ответами эталонного решения, составленного авторами задачи. Задачи требуют базовые навыки программирования, поскольку это является необходимой частью финального задания. Ввод данных в программу осуществлется через стандартный ввод (`stdin`), вывод данных — через стандартный вывод (`stdout`). Случайные числа, используемые для генерации тестов, являются псевдослучайными. Все проверки программ всех участников производятся на одинаковых данных.

Командная работа начинается именно во время второго этапа. Помимо общих механизмов для профиля, создано условие на командное взаимодействие – ограниченное число попыток на каждую задачу для команды. Это приводит к необходимости уже на начальном этапе договариваться о стратегии сдачи решений общекомандно, и обозначать свою позицию по отношению к каждой задачи внутри команды — берешь ли ты ее на себя, пишешь варианты решений или передаешь ответственность за нее другим участникам команды. Так, задачи разделяются между участниками согласно их сильных и слабых сторон, что с учётом связи задач с финалом создаёт предпосылки к формированию ролей в работе над финальной задачей. Учёт общекомандных попыток ведётся посредством внешнего бота, предоставляющего текущую информацию о командных баллах и оставшихся командных попытках. Бот оживляет и делает осмысленным соревнование во время второго тура — оно становится проявленным и работает на повышение мотивации. На протяжении всего второго тура на вебинарах поднимаются темы совместных ресурсов (Trello, Яндекс.Диск, Github) для структурирования командной работы. Также на вебинарах не всегда возможно присутствие всех членов команды — и это порождает отдельный процесс по синхронизации информации и понимания. В конце второго тура командам предлагается совместная рефлексия с выявлением слабых и сильных сторон у каждого члена команды, определением белых пятен в знаниях и навыках у каждого и распределением нерешённых задач для дорешивания.

Задача IV.1. «Зделай брейнфук» (14 баллов)

Темы: программирование, работа с моделью.

Условие

Дан язык программирования Bublic, программы которого предназначены для одноимённого компьютера с памятью в виде **закольцованной** ленты из 10 ячеек, в которых может храниться любое **целое** число. Кроме этого, компьютер имеет **указатель** на текущую ячейку (перед запуском программы устанавливается на **нулевую** ячейку).

В языке всего 9 команд.

`Z(n)`	Занулить n-ю ячейку, считая вправо от активной.
`I(n)`	Прибавить единицу к n-й ячейке, считая вправо от активной.
`S(n,m)`	Поменять значения между ячейками (опять же, координаты относительные).
`J(n,m)`	Если значение в n-й ячейке больше значения в m-й ячейке, то ничего не делать (перейти к следующей команде), если нет — перескочить через следующую команду. Координаты ячеек относительные.
`G(n)`	Безусловно перейти к n-й строке программы, считая от текущей.
`L`	Сдвинуть **указатель** на ячейку влево.
`R`	Сдвинуть **указатель** на ячейку вправо.
`D(n)`	Удвоить значение в n-й ячейке, считая вправо от активной.
`M(k)`	Повторить следующую команду k раз. Если следующая команда — `J` или `G`, то `M` игнорируется (т. е. команда после перехода выполнится один раз). Если следующая команда — `M`, то число повторений умножается.

Числа в параметрах — **целые**, кроме команды `M`, где параметр — **натуральное** число. Положительное значение параметра соответствует перемещению вправо или вперёд. Шаг соответствует **выполнению** команды (в том числе её повторению).

Вам даны программа на языке Bublic и начальное состояние ленты. Определите значение **активной** ячейки ленты на момент **завершения** программы. Если программа выполняется **больше 5000** шагов, вывести «LONGCAT». Если произошёл переход на несуществующую строку программы, программа считается завершённой.

Формат входных данных

В нулевой строке — 10 целых чисел через пробел, начальное состояние ленты. Далее приводится текст программы в произвольное число строк. Пример входных данных:

```
35 15 0 0 0 0 0 0 0 0
Z(2)
I(2)
J(2,0)
G(3)
I(1)
G(-4)
R
G(-100)
S(-1,1)
L
```

```
M(10)
D(-1)
```

Формат выходных данных

Единственное целое число — значение активной ячейки ленты на момент завершения программы. Например, для примера выше нужно вывести 50.

Эта задача учит работе с предметными языками, что понадобится при работе с управляющими скриптами во время финала.

Решение

Для успешного решения задачи нужно внимательно реализовать интерпретатор языка согласно спецификации. Особое внимание следует обратить на механику повторов (команда `M`), относительность всех параметров-координат и то, что переход на индекс команды за пределами программы считается её завершением. Также необходимо не построчное чтение входных данных, а использование потока `stdin` (псевдофайл в библиотеке `sys`). В остальном реализация прямолинейная.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```python
MEMSIZE = 10
ERR = "LONGCAT"
MAXSTEP = 5000

def bake(x):
    i = x.find("#")
    if i > -1:
        x = x[:i]
    x = x.strip()
    if not x:
        return None
    cmd, param = x[0], x[1:]
    param = [int(v) for v in param[1:-1].split(",") ] if param else []
    return (cmd, param)

def run(prog, mem=None):
    prog = [bake(x) for x in prog.splitlines()]
    prog = list(filter(None, prog))
    mem = [0 for _ in range(MEMSIZE)] if mem is None else mem.copy()
    idx = 0
    i = 0
    steps = 0
    repeats = 1
    while 0 <= i < len(prog):
        if steps >= MAXSTEP:
            return None
        cmd, param = prog[i]
        # print(cmd, param)
        if cmd == "L": # slide left
            idx = (idx - 1) % MEMSIZE
        elif cmd == "R": # slide right
```

```
            idx = (idx + 1) % MEMSIZE
        elif cmd == "Z": # zero
            crd = (idx + param[0]) % MEMSIZE
            mem[crd] = 0
        elif cmd == "I": # increment
            crd = (idx + param[0]) % MEMSIZE
            mem[crd] += 1
        elif cmd == "D": # double
            crd = (idx + param[0]) % MEMSIZE
            mem[crd] *= 2
        elif cmd == "S": # swap
            crd1 = (idx + param[0]) % MEMSIZE
            crd2 = (idx + param[1]) % MEMSIZE
            mem[crd1], mem[crd2] = mem[crd2], mem[crd1]
        elif cmd == "J": # compare and jump over
            v1 = mem[(idx + param[0]) % MEMSIZE]
            v2 = mem[(idx + param[1]) % MEMSIZE]
            if v1 > v2:
                i += 1  # next operator
            else:
                i += 2  # jump over
            steps += 1
            repeats = 1
            continue
        elif cmd == "G":
            i = i + param[0]
            steps += 1
            repeats = 1
            continue
        elif cmd == "M":
            if repeats > 1:
                repeats *= param[0]
            else:
                repeats = param[0]
            i += 1
            steps += 1
            continue
        if repeats <= 1:
            steps += 1
            i += 1
            repeats = 1
        else:
            repeats -= 1
    return (idx, mem)

import sys
data = sys.stdin.read().splitlines()
mem = [int(x) for x in data[0].split()]
prog = "\n".join(data[1:])
s = run(prog, mem)
print(ERR if s is None else s[1][s[0]])
```

Задача IV.2. Пост-Стевин (20 баллов)

Темы: работа с моделью, математика.

Условие

Десятичные дроби, которыми мы пользуемся, не появились сами, а были **изобретены** — их придумал в конце 16 века математик **Симон Стевин**.

Чтобы найти десятичную дробь для числа, меньшего единицы, мы **делим** единичный отрезок на 10 одинаковых частей, и записываем номер отрезка (начиная с нуля), в который попало число. Затем этот отрезок снова делим на 10 частей, и так и далее, пока не получим достаточную точность. Если число больше единицы, то похожий процесс можно проводить и в обратную сторону, тогда мы найдём десятичные цифры слева от десятичной запятой.

В десятичных дробях, к которым мы привыкли, все цифры, стоящие на фиксированной позиции, отвечают за отрезки одинаковой длины, а позиция цифры отвечает за масштаб — при смещении вправо по записи числа масштаб увеличивается в 10 раз на один шаг.

Но числам не обязательно быть такими! Каждая цифра может быть ***особенной*** и отвечать за отрезок своей, **особой** длины! А ещё отрезки могут пересекаться, ведь это ни на что важное не влияет, зато красиво!

Например, единичный отрезок может быть разбит на цифры так:

- Цифра **0**: от начала единичного отрезка до $e/8$.
- Цифра **1**: e/π^3 правее цифры 0.
- Цифра **2**: $\pi/17$ левее цифры 3.
- Цифра **3**: $0,12$ левее цифры 4,
- Цифра **4**: $\sqrt{\pi}/12$ левее цифры 5.
- Цифра **5**: $1/7$ левее конца единичного отрезка.

В нашей системе счисления цифр всего шесть, и это самая непримечательная её особенность. Все не выделенные жирным цифры означают обычные **десятичные** числа.

Вам дан массив **особенных** дробей от нуля до единицы (исключая их самих). Нужно найти и вывести их **среднее геометрическое**. Естественно, в **особенной** форме. Но если особенная форма не получается, для начала можно и в обычной.

Формат входных данных

В первой строке — **десятичное** натуральное число N. Далее N строк, в каждой — одно **особенное** число. Например,

```
3
0.1234512345
0.0001001
0.54321
```

Формат выходных данных

Особенное число, среднее геометрическое. **Допускается** вывод ответа в десятичной форме, если число предваряется символом «`!`», например, «`!0.123456789`».

Задача оценивается исходя из числа пройденных тестов. Тесты поделены на блоки

по наборам данных. Если ваше решение находит среднее геометрическое и выводит его **хотя бы** в десятичной форме с погрешностью **не более** 10^{-7}, вы получаете 50% за блок. Если решение выводит особенное число, соответствующее десятичному с погрешностью **не более** 10^{-7}, вы получаете ещё 25%. Оставшиеся 25% требуют достичь погрешности **не более** 10^{-20}.

Эта задача проверяет навык работы со сложными математическими моделями, что является частью финальной задачи.

Решение

Задача состоит из трёх компонентов: декодирование «нестевиновской» дроби в вещественное число, обратное кодирование (для полного решения) и вычисление среднего геометрического. Последнее делается тривиально. Декодирование нестевиновской дроби осуществляется рекурсивно. Так, для вычисления значения дроби 0,354 мы прибавляем $1 - (0,12 + \sqrt{\pi}/12 + 1/7)$ (расстояние от нуля до точки 3 на единичном отрезке), после прибавляем $0,12 \cdot (1 - 1/7)$ (расстояние от нуля до точки 5 на отрезке между точками 3 и 4, взятыми на единичном отрезке), после — $0,12 \cdot 1/7 \cdot (1 - (\sqrt{\pi}/12 + 1/7))$ (расстояние от нуля до точки 4 на отрезке между точкой 5 и правым краем отрезка между точками 3 и 4...).

Таким образом, мы имеем рекурсивную процедуру. Единственная проблема — положение точки 2. С одной стороны, мы можем определить её на расстоянии e/π^3 правее точки 1, либо на расстоянии $\pi/17$ левее точки 3 (см. схему в авторском решении). Проверочная система допускает оба варианта, достаточно лишь соблюдать положение точки 2 консистентно во всём решении.

Для полного решения необходимо закодировать вещественное число обратно в нестевиновскую дробь. Это может быть реализовано путём последовательного приближения: подбираем цифру так, чтобы число попадало в отрезок этой цифры, после чего подбираем следующую цифру, и так до тех пор, как не будет достигнута необходимая точность. Авторское решение предлагает отталкиваться от готового кодировщика, постепенным подбором дроби с приближением и обратными шагами. Для большей точности используется модуль `decimal` из стандартной библиотеки.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
from decimal import Decimal as deci

"""
0 -- d0 -- 1 -- d1 -- 2?
                  2? -- d2 -- 3 -- d3 -- 4 -- d4 -- 5 -- d5 -- 6
"""

pi = deci("3.14159265358979323846")
e = deci("2.71828182845904523536")

d0 = e / 8
d1 = e / (pi ** 3)
d2 = pi / 17
d3 = deci("0.12")
d4 = (pi**deci("0.5")) / 12
```

```
d5 = deci("1") / 7

wds = [ d0, d1, d2, d3, d4, d5 ]

sts = [ deci(0), d0, 1 - (d2+d3+d4+d5), 1 - (d3+d4+d5)
      , 1 - (d4+d5), 1 - (d5), 1 ]

def geomean(xs):
    c = deci(1)
    for x in xs:
        c *= x
    return c ** (deci(1) / len(xs))

def decode(x):
    if x == "0":
        return deci(0)
    acc = deci(0)
    wi = deci(1)
    for c in x[2:]:
        if c in '012345':
            d = ord(c) - 48 # '0'
            acc += sts[d] * wi
            wi *= wds[d]
        else:
            raise ValueError()
    return acc

EPS = deci("1e-8")
NDIGITS = 50

next_num = dict(zip("01234", "12345"))
prev_num = dict(zip("12345", "01234"))

def encode(x):
    x = deci(x)
    if x == deci(0):
        return "0"
    ans = ["0.", "0"]
    added = False
    while len(ans) <= NDIGITS:
        v = decode("".join(ans))
        if v < x:
            if ans[-1] == "5":
                ans.append("0")
                added = False
            else:
                ans[-1] = next_num[ans[-1]]
                added = True
        if v > x:
            if ans[-1] == "0":
                ans[-2] = prev_num[ans[-2]]
                ans[-1] = "5"
            elif added:
                ans[-1] = prev_num[ans[-1]]
                ans.append("5")
            else:
                ans[-1] = prev_num[ans[-1]]
            added = False
        if v == x:
            break
```

```
    return "".join(ans)

n = int(input())
vs = [decode(input()) for _ in range(n)]
v = geomean(vs)
print(encode(v))
```

Задача IV.3. Фиксируем прибыль (9 баллов)

Темы: теория игр, анализ данных, работа с моделью.

Условие

Дауншифтер Джон Грин пережил особо холодную зиму, и теперь он продаёт накопившийся запас дров из сарая на **бирже**. Джон стремится сделать это как можно **выгоднее**, и для этого он отслеживает два **прогноза**: цена кубометра дров и их медианное потребление. Он хочет выбрать момент, когда **произведение** обеих величин будет **максимальным** за прогнозируемый период.

Прогнозы представлены в виде **рядов точек**, где X — это время в виде условных часов, Y — это значения соответствующих величин. Значения между точек интерполируются линейно. Найдите X, где произведение значений рядов **максимально**. Если в нескольких точках одинаково максимальное значение, выведите наиболее **раннюю**.

Формат входных данных

В первой строке два числа — количество точек (до 120) в первом (N) и втором (M) ряду данных соответственно. Далее N строк, в каждой точка первого ряда данных, представленная парой положительных вещественных чисел, X и Y соответственно. После идут M строк, в каждой точка второго ряда аналогичного формата. Точки приводятся в произвольном порядке. Пример формата:

```
2 3
1.5 2.2
3.4 1.2
1.0 1.1
2.0 5.2
4.12 2.3
```

Формат выходных данных

Единственное вещественное число, координата X, в которой произведение Y из обоих рядов **максимально**. Ответ считается корректным, если отличается от авторского не более, чем на 10^{-6}.

Эта задача проверяет навык программного анализа данных, что потребуется при решении финальной задачи.

Решение

Для решения необходимо найти максимум произведения ряда. Авторское решение реализовано более мягко, допуская в качестве ответа значения, полученные через точки ряда. Так, одновременно рассматривается по одной точке каждого ряда. Для отстающей точки одного ряда вычисляется линейная интерполяция соответствующей координаты другого ряда, после чего берётся следующая точка. Таким образом, для всех координат X (из обоих рядов) мы получаем значения Y для обоих рядов, что позволяет вычислить произведение в этих точках и определить максимальное.

При этом существует более качественное решение, основанное на рассмотрении отрезков двух графиков и экстремума первой производной их произведения: если отрезки сонаправлены, то максимум по отрезку соответствует одной из крайних X-координат, иначе вычисляется нулевая точка первой производной, и рассматривается значение в ней.

Пример программы-решения

Ниже представлено решение на языке Python 3.

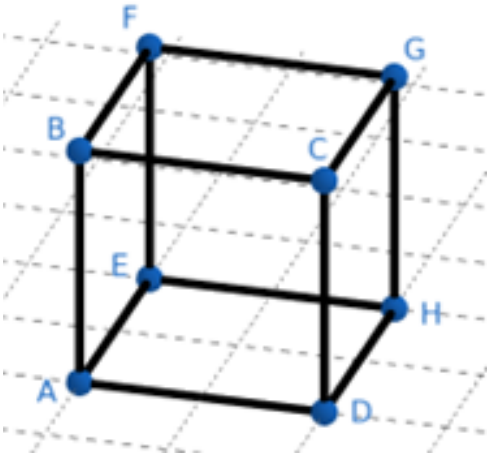
```
def interp(x1, x2, y1, y2, x):
    return y1 + (y2 - y1) * (x - x1) / (x2 - x1)

def prepare_row(r1, r2):
    r1, r2 = sorted(r1), sorted(r2)
    ri1, ri2 = len(r1), len(r2)
    i1, i2, ro = 0, 0, []
    while (i1 < ri1 and i2 < ri2):
        (x1, y1), (x2, y2) = r1[i1], r2[i2]
        if x1 == x2:
            ro.append((x1, y1 * y2))
            i1 += 1
            i2 += 1
            continue
        if x1 < x2:
            yi2 = interp(r2[i2-1][0], r2[i2][0], r2[i2-1][1],
                         r2[i2][1], x1)
            ro.append((x1, y1 * yi2))
            i1 += 1
            continue
        # if x1 > x2:
        yi1 = interp(r1[i1-1][0], r1[i1][0], r1[i1-1][1],
                     r1[i1][1], x2)
        ro.append((x2, yi1 * y2))
        i2 += 1
    return ro

n, m = map(int, input().split())
r1 = [tuple(map(float, input().split())) for _ in range(n)]
r2 = [tuple(map(float, input().split())) for _ in range(m)]
rr = prepare_row(r1, r2)
(_, ans) = max((y, -x) for (x, y) in rr)
print(-ans)
```

Задача IV.4. Путь к финалу (11 баллов)

Темы: программирование, работа с моделью.

Условие

Вам дан **куб** (см. рисунок, диагонали не построены для простоты понимания). Каждое ребро куба и каждая его диагональ, включая **внутренние**, обладает собственными фиксированными **энергозатратами** на перемещение по ним. Эти энергозатраты распределены **равномерно** по длине. Найдите путь **наименьших** энергозатрат из узла A в узел G, если в точках пересечения диагонали **соединены**.



Формат входных данных

28 строк, в которых через пробел приводится обозначение ребра/диагонали (две заглавных латинских буквы) и его энергозатраты. Порядок перечисления рёбер и диагоналей фиксирован. Пример формата:

```
AB 1234
BC 121
```

Формат выходных данных

Единственное вещественное число, энергозатраты оптимального пути из A в G. Ответ считается корректным, если отличается от авторского не более, чем на 10^{-6}.

Эта задача проверяет ваш навык работы со сложными моделями при помощи программирования, что является частью финальной задачи.

Решение

Задача представляет собой поиск кратчайшего пути в графе, для чего существует широкое множество различных алгоритмов. Авторское решение использует реализацию алгоритма Дейкстры из библиотеки `scipy`. Единственный нюанс заключается в необходимости ввести в граф дополнительные точки, соответствующие пересечениям диагоналей куба. Для этого написана вспомогательная функция, которая «делит» заданное ребро пополам новой точкой.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```python
import sys
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import dijkstra

VERT = "ABCDEFGH" + "IJKLMN" + "O"
NVERT = len(VERT)

def add_vert(graph, x, v):
    a, b = map(VERT.index, sorted(x))
    v = float(v)
    graph[a][b] = graph[b][a] = v

def del_vert(graph, x):
    a, b = map(VERT.index, sorted(x))
    x = graph[a][b]
    graph[a][b] = graph[b][a] = 0
    return x

def cut_half(gr, a, b, v):
    x = del_vert(gr, a + b) / 2
    add_vert(gr, a + v, x)
    add_vert(gr, v + b, x)

data = sys.stdin.read()
data = [x.split() for x in data.splitlines()]
gr = np.zeros((NVERT, NVERT))
for c, v in data:
    add_vert(gr, c, v)
cut_half(gr, "A", "C", "I"); cut_half(gr, "B", "D", "I")
cut_half(gr, "F", "H", "J"); cut_half(gr, "E", "G", "J")
cut_half(gr, "B", "G", "K"); cut_half(gr, "C", "F", "K")
cut_half(gr, "A", "H", "L"); cut_half(gr, "D", "E", "L")
cut_half(gr, "A", "F", "M"); cut_half(gr, "B", "E", "M")
cut_half(gr, "C", "H", "N"); cut_half(gr, "D", "G", "N")
cut_half(gr, "A", "G", "O"); cut_half(gr, "D", "F", "O")
cut_half(gr, "B", "H", "O"); cut_half(gr, "C", "E", "O")
gr = csr_matrix(gr)
dm = dijkstra(csgraph=gr, directed=False, indices=0)
print(dm[VERT.index("G")])
```

Задача IV.5. Невероятная дуэль (23 баллов)

Темы: теория вероятностей, программирование.

Условие

В перестрелке «все против всех» участвует N участников. Стрельба происходит **по очереди**, поражённые участники из перестрелки выбывают до тех пор, **пока не останется один участник**. Перестрелка идёт до последнего участника. Для каждого участника известна вероятность поражения им своей цели (меткость). Цель каждый участник выбирает случайно, но **пропорционально** её опасности: если у

участника A меткость в два раза выше, чем у участника Б (например, 80% и 40%), то участник A будет выбран с вдвое большей вероятностью, чем участник $Б$.

Стреляют участники в порядке **возрастания** меткости. Если меткости равны, порядок определяется позицией во входном списке (чем выше — тем раньше).

Каков шанс выжить у участника с **наибольшим** шансом выживания?

Формат входных данных

В первой строке целое N (до 15) — число участников дуэли. Далее N строк, в каждой вещественное число от нуля до единицы — меткость соответствующего участника. Пример формата:

```
3
0,12322
0,42145
0,12469993
```

Формат выходных данных

Единственное вещественное число, вероятность выжить у наиболее живучего участника. Ответ считается корректным, если отличается от авторского **не более**, чем на 15%.

Эта задача проверяет знание теории вероятностей и численных методов, что потребуется в работе над финальной задачей.

Решение

Для решения необходимо найти максимум произведения ряда. Авторское решение реализовано более мягко, допуская в качестве ответа значения, полученные через точки ряда. Так, одновременно рассматривается по одной точке каждого ряда. Для отстающей точки одного ряда вычисляется линейная интерполяция соответствующей координаты другого ряда, после чего берётся следующая точка. Таким образом, для всех координат X (из обоих рядов) мы получаем значения Y для обоих рядов, что позволяет вычислить произведение в этих точках и определить максимальное.

При этом существует более качественное решение, основанное на рассмотрении отрезков двух графиков и экстремума первой производной их произведения: если отрезки сонаправлены, то максимум по отрезку соответствует одной из крайних X-координат, иначе вычисляется нулевая точка первой производной, и рассматривается значение в ней.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
1  def interp(x1, x2, y1, y2, x):
2      return y1 + (y2 - y1) * (x - x1) / (x2 - x1)
3
4  def prepare_row(r1, r2):
```

```
    r1, r2 = sorted(r1), sorted(r2)
    ri1, ri2 = len(r1), len(r2)
    i1, i2, ro = 0, 0, []
    while (i1 < ri1 and i2 < ri2):
        (x1, y1), (x2, y2) = r1[i1], r2[i2]
        if x1 == x2:
            ro.append((x1, y1 * y2))
            i1 += 1
            i2 += 1
            continue
        if x1 < x2:
            yi2 = interp(r2[i2-1][0], r2[i2][0], r2[i2-1][1],
                         r2[i2][1], x1)
            ro.append((x1, y1 * yi2))
            i1 += 1
            continue
        # if x1 > x2:
        yi1 = interp(r1[i1-1][0], r1[i1][0], r1[i1-1][1],
                     r1[i1][1], x2)
        ro.append((x2, yi1 * y2))
        i2 += 1
    return ro

n, m = map(int, input().split())
r1 = [tuple(map(float, input().split())) for _ in range(n)]
r2 = [tuple(map(float, input().split())) for _ in range(m)]
rr = prepare_row(r1, r2)
(_, ans) = max((y, -x) for (x, y) in rr)
print(-ans)
```

Задача IV.6. Лесные гонки в лесных гонках (25 баллов)

Темы: теория игр, программирование.

Условие

Вам предлагается игра «**Лесные гонки в лесных гонках**». Незадачливый браконьер убегает от двух медведей по **ветвящейся** лесной тропинке. Вы играете за одного из медведей, и ваша задача — через **десять** развилок оказаться ближе к незадачливому браконьеру, чем ваш соперник.

Браконьер — хлипкий человечишка, и поэтому может бежать **только** по тропинке, на каждой развилке выбирая направление -1 (влево) или 1 (вправо), случайно с **одинаковой** вероятностью. Медведям тропинки не нужны, поэтому они могут бежать в любом направлении от -1 до 1. После развилки расстояние до браконьера изменяется как $\left(\frac{B-x}{2}\right)^2$, где B — выбор браконьера, а x — выбор медведя. Нетрудно заметить, что **минимум** математического ожидания изменения расстояния до браконьера за одну развилку соответствует выбору 0 (бежать прямо).

Вы соревнуетесь с медведем-чемпионом Михаилом Оптимусом, который всегда бежит **оптимально**, то есть всегда прямо. Вам нужно составить стратегию гонок, которая обгоняет Оптимуса **хотя бы** в 95 гонках из 100. Скорости всех медведей одинаковы. В момент начала гонки браконьер делает фальстарт и успевает отбежать от медведей на **неизвестное** расстояние. До окончания гонки браконьера **не догонят**. После окончания гонки браконьера загоняют на начало пути для следующего забега.

Для участия в игре нужно написать **бота** — функцию, на вход которой передаются данные о прошлых ходах в виде списка троек пар «собственное решение — решение медведя – решение браконьера» (`list[tuple[float,float,float]]`) и собственное текущее расстояние до браконьера (`float`). Функция выводит **единственное** число от -1 до $+1$ — направление бега. Сигнатура функции будет представлена в поле ввода решения.

Ваши функции **не должны** работать со стандартными потоками ввода, вывода и ошибок. Мы оставляем за собой право **обнулить** подобные решения, даже если они пройдут. В ограничение на время выполнения входит время работы вспомогательного кода, но оно крайне **незначительно**.

Эта задача учит аспектам теории игр, которые необходимы при решении финальной задачи.

Решение

Для успешного решения задачи необходимо «обогнать» матожидание. Самый надёжный способ сделать это — попытаться обогнать его хотя бы на небольшую долю расстояния. Для этого мы в первом раунде выдаём слегка отличное от нуля число. Если браконьер побежит в ту же сторону, и в следующем раунде расстояние до него окажется меньше, чем у Оптимуса — все оставшиеся раунды можно возвращать 0, т. к. выигрыш обеспечен. Иначе мы увеличиваем отклонение так, чтобы в случае успеха и компенсировать предыдущую неудачу, и обеспечить обгон. С большой вероятностью (более 95%) такая стратегия обеспечивает победу. Сравнение с Оптимусом можно производить как по прошлым попыткам (`stats`), так и по длине списка и расстоянию `dist`, но тогда необходимо учесть начальное отставание (`start_dist` в авторском решении).

Пример программы-решения

Ниже представлено решение на языке Python 3.

```python
start_dist = 0
def bot(stats, dist):
    global start_dist
    if not stats:
        start_dist = dist
        dist = 0
    else:
        dist -= start_dist
    moves = [0.001, -0.002, 0.004, -0.008, 0.016,
             -0.032, 0.064, -0.128, 0.256, -0.512]
    if dist < 0.25 * len(stats):
        return 0
    return moves[len(stats)]
```