§4 Заключительный этап: командная часть

Краткая справка

Финальный этап профиля проводился на специально разработанном стенде-тренажере “ИЭС” (Подробное устройство стенда представлено в Приложении 1). Участникам предстояло работать с моделью энергосистемы, в которой присутствует большое количество альтернативных источников энергии, накопителей энергии и широкие возможности для экономических действий. В результате участники изучали: физические и экономические параметры энергосетей; сильные и слабые стороны альтернативной энергетики, взаимосвязь инженерных и экономические решений и проблему качества и надежности электросетей.

В финальной задаче участникам требовались навыки и знания из школьной программы, а также самостоятельно приобретенные на хакатонах и из требуемых для решения задач второго этапа:

1. Умение программировать на языке Python.
2. Умение работать со случайными величинами в программных вычислениях.
3. Умение решать базовые оптимизационные задачи.
4. Навыки реализации решений математических задач в виде программ.
5. Законы электричества: закон Ома, закон Кирхгофа, работа электрического тока.
6. Понимание правил и решения закрытого аукциона первой цены.
7. Навыки командной работы над программным кодом.
8. Навыки работы с большими рядами данных в математических задачах.
9. Навыки работы с рядами данных в алгоритмах.
10. Понимание принципов работы и характеристик ветрогенераторов.
11. Теория вероятностей.

Успешное решение финальной задачи предполагает освоение и развитие следующих знаний и навыков:

1. Умение работы в команде.
2. Умение самостоятельно выделять и формулировать подзадачи.
3. Понимание принципов работы электрических сетей.
4. Понимание принципов оценки риска.
5. Навыки программирования, в том числе на языке Python.
6. Навыков работы с рядами данных, как аналитической, так и алгоритмической.
7. Понимания принципов физических измерений и оценки погрешностей.
8. Навыки работы с системами с инерцией.
9. Практического использования теории вероятностей, в том числе в программных вычислениях.

4.1 Финальная командная задача: Моделирование энергосистемы и разработка алгоритмов управления энергообеспечения

На стенде моделируется небольшой город, энергосистеме которого предстоит быть полностью перестроенной. В том числе, в энергосистему будет добавлено большое количество ветровой и солнечной генерации.

Будущая энергосистема будет «разделена» между четырьмя конкурирующими компаниями. Каждая команда станет одной из четырех конкурирующих энергокомпаний и будет строить свою собственную энергосистему и управлять ей.

Разделение будет происходить через цепочку аукционов, в которых участники изначально будут в принципиально одинаковых условиях. После этого команды проектируют собственные энергосистемы (из одних и тех же объектов можно составить очень разные по эффективности энергосистемы) и готовят скрипты для управления ими. Далее происходит натурное моделирование нескольких дней работы энергосистемы, в ходе которого участники управляют своими энергосистемами посредством скриптов; и происходит экспериментальное измерение эффективности построенных энергосистемы. Баллы, набранные командами во время моделирования, пересчитываются в баллы, которые их участники получат за командную часть олимпиады.

Проведение финала происходило в виде командного турнира. Цель команд участников — набрать наибольшее число баллов в турнире.

Регламент турнира

1. Подготовка.

1.1. Проводился уже сформированными командами. Это продолжительный по времени этап (3 дня), в течение которого участники знакомились с правилами, тренировались работать на стенде, изучали предоставленную систему и готовили заготовки (управляющие скрипты, стратегии и вспомогательные программы).
2. Два полуфинала турнира
2.1. Команды делились на две группы по четыре команды случайным образом. С каждой группой проводился отдельный полуфинал.
2.2. В полуфинале проводились две игры на разных погодных сценариях, но одинаковых конфигурациях сетей. Аукцион проводился один раз. Между играми участники не имели права подходить к стенду.
3. Финал турнира
3.1. В финал проходили две команды, набравшие наибольшее число очков (у.е.) по сумме двух игр.
3.2. В финале проходили шесть игр в следующей последовательности:

- Участникам выдавались прогнозы погоды на три игры
- Проводился один аукцион
- Участники составляли свои энергосистемы
- Три игры на основе составленных участниками конфигураций энергосистем. Таким образом, этапы 1–4 трёх игры были объединены, что позволило сократить время подготовки и увеличить число возможностей для исправления технических ошибок (и уменьшить их влияние на результат).
- Проводилось ещё три игры по такому же регламенту. Таким образом, аукционов было два, у каждой команды было две энергосистемы, а натурное моделирование проводилось шесть раз.

3.3. Победитель определялся по сумме очков (у.е.) за все шесть игр.
3.4. Финал за 5–8 место проводился по тем же правилам, что и полуфиналы.

Этапы одной игры

1. Анализ прогнозов погоды. Минимум за 20 минут до игры участникам выдавались прогнозы погоды и потребления в предстоящей игре. За это время участники должны были спроектировать энергосистему, наиболее отвечающую предстоящим условиям.

2. Основной аукцион. На этом этапе определялось, какой объект к чьей энергосистеме будет подключён. Аукцион закрытого типа, с продолжением в случае одинаковых ставок. Этот этап практически невозможно успешно пройти без глубокого предварительного анализа и заготовленных программ для поддержки принятия решений.

3. Дополнительный аукцион. Проводился через 1 минуту после основного. Каждая команда имела право повторно «выставить на торги» любой из приобретённых ранее объектов. Этот этап давал командам шанс на исправление одной ошибки, если она не слишком велика.
4. Монтаж энергосистемы и адаптация стратегии. Участникам было предоставлено на сборку сети 20 минут. За это время они находили оптимальную конфигурацию своей энергосистемы, монтировали её на стенде, включая оптимальную установку электростанций. В это же время команда адаптировала к получившейся энергосистеме составленные заранее управляющие скрипты.
5. Моделирование. Команды в полном составе находились возле собственных терминалов управления; к стенду запрещено было приближаться ближе, чем на 2 метра всем, включая обслуживающий персонал во избежание влияния на физические измерения. Участники наблюдали за работой своего скрипта и могли его заменить, например, в случае обнаружения ошибки. Число загружаемых скриптов правилами не было ограничено. Почти всё управление осуществлялось только скриптами, но небольшое подмножество управляющих воздействий было доступно игрокам через пользовательский интерфейс.

Аукцион

Аукцион проводился по закрытой схеме. Выигрывал предложивший наибольшую цену в аукционе на электростанции, и наименьшую — в аукционе на потребителей. Стартовая цена для электростанций составляла 1, для потребителей — 10. На ставку отводилось 30 секунд, ставки присылались администратору аукциона через личные сообщения в Telegram. В случае совпадения наилучших ставок проводился дополнительный тур аукциона, в котором участвовали только те, чьи ставки совпали. Порядок выставления лотов фиксирован и известен участникам заранее. В течение минуты после окончания аукциона каждая команда имела право выставить на торги один из своих объектов. Команда ничего с этого не приобретает, но получает возможность избавиться от лишнего объекта, нарушающего баланс энергосистемы. Команда не имела права делать ставки на объект, который выставила на торги. У каждой команды в наличии по умолчанию имелось два дома и одна солнечная электростанция.

Пример результатов аукциона

Адрес	Электросыч	Энерготех	D2AK	Ora et Labora
f0f1	6,2	**5,8**	7,1	6,7
s4	**16**	14,2	15,7	6
b0b1	—	—	7	**6**
s5	**16,7**	16,4	15,7	7
f2f3	6	7	6,2	**5,8**
s6	10	**17**	16	6
a0	10	2	**18**	14
b2b3	—	—	6,9	**6,5**
a1	7	2	**10**	5
b4b5	—	—	**5,9**	7,5
s7	13	**17,3**	12	8
f4f5	**5,3**	7	5,4	7
f6f7	6,3	6,5	**5,3**	7
a2	3	3	3,9	**4**
h9	**5**	6	5,2	—
a0	—	3	no	**5**

Легенда таблицы

- порядок лотов — сверху вниз;

- адреса потребителей начинаются с f(заводы), b(больницы) или h(дома);
- адреса электростанций начинаются с s(солнечная) или a(ветровая);
- число в поле означает ставку команды;
- «—» в поле означает пас;
- «no» в поле означает, что команда не имела права делать ставку в этом туре/лоте;
- полужирное число означает победившую ставку;
- пустая строка в конце таблицы отделяет основной аукцион от дополнительного.

Подзадачи финальной задачи

В ходе выполнения финальной задачи оценивалось комплексное решение финальной задачи профиля. Выделение и формулирование подзадачи является частью интегральной (главной) задачи профиля. Проверкой решения подзадачи являлся вклад её в результат игры. Описанные ниже задачи выделены разработчиками и было дано участником из описания. Распределение усилий между подзадачами принимали сами команды.

4.1.1 Физика

4.1.1.1. Оптимальное расположение ветряка

Задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Первая гипотеза «установить ветряк как можно ближе к анемометру» полностью разрушается двумя факторами:

1. При приближении к вентилятору создаваемое им ветровое поле становится всё более неравномерным. Например, на направлении его оси скорость ветра снижается, поскольку проталкивание воздуха осуществляется не всей плоскостью вентилятора, а только его лопастями. На следующем порядке малости на скорость ветра влияет расположение вентилятора относительно стенда и стен помещения, расположения других объектов на стенде (в особенности других ветряков).
2. Устройство анемометра, установленного на модели ветряка,- вертикально-осевое. На скорость его вращения, в большей степени, чем сама скорость ветра, влияет разность ветрового давления, создаваемого на правую и левую его часть. Получается, что анемометр нужно устанавливать не в зоне большего ветра, а в зоне, в которой силы, вращающие его в «правильную» сторону (против часовой стрелки), будут максимально превосходить силы, вращающие его в противоположную сторону.

Эта задача оптимального расположения весьма эффективно и просто решается при помощи простого наблюдения за скоростью вращения анемометра, однако те участники, которые продемонстрировали понимание физики происходящего, потратили на эту задачу в несколько раз меньше времени (и соответственно, смогли потратить больше времени на другие задачи).

4.1.1.2. Оптимальное расположение солнечной батареи

Эта задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Расположение солнечной батареи характеризуется четырьмя параметрами: ось вдоль стенда (ближе/дальше от светильников), ось поперёк стенда (ближе/дальше к краю светильников), углы наклона солнечной панели к плоскости стенда вдоль и поперёк стенда. Из общих соображений и соображений симметрии видно, что задача является частично вырожденной: по оси поперёк стенда солнечную батарею необходимо располагать как можно ближе к центру стенда, то же самое касается наклона солнечной батареи поперёк стенда — линия пересечения плоскостей солнечной панели и стенда должна быть перпендикулярна продольной оси стенда.

Светильники стенда являются светодиодными, и в первом приближении каждый светодиод можно считать точечным источником света. Из общих соображений солнечная батарея должна располагаться как можно ближе к продольной оси стенда. Это и то, что светодиоды расположены близко друг к другу (около 1 см), позволяет в модели заменить светильники точечными источниками света и исключить из модели положение светильника вдоль поперечной оси стенда.

Положение солнечной панели регулируется двумя параметрами: положением вдоль продольной оси стенда, и углом наклона солнечной панели вдоль поперечной оси стенда.

Для нахождения максимума светового потока, попадающего на солнечную батарею, достаточно найти угловой размер солнечной панели из «точки зрения» источников света.

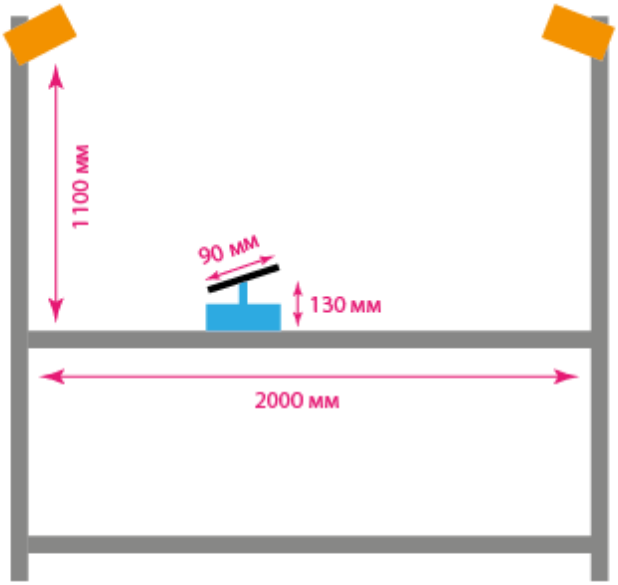


Рис 1. Габариты стенда, светильников и солнечной панели.

Рис. 2. Обозначения геометрической задачи нахождения углового размера солнечной батареи в поле зрения источника света

$$l = \sqrt{h^2 + x^2}$$

$$\gamma = \beta - \alpha$$

$$\phi_1 = arccos\frac{l - acos(\pi/2 + \gamma)}{\sqrt{l^2 + a^2 - 2alcos(\pi/2 + \gamma)}}$$

$$\phi_2 = arccos\frac{l - acos(\pi/2 - \gamma)}{\sqrt{l^2 + a^2 - 2alcos(\pi/2 - \gamma)}}$$

Отметим, что если солнечная панель развёрнута к источнику света «тыльной» стороной, то световой поток от него будет нулевым (например, мы наклонили её на 45° влево и максимально сдвинули влево, тогда второй светильник не освещает солнечную панель), однако угловой размер нулевым не будет. Это приводит к дополнительному ограничению:

$$\text{если} |\gamma| > \frac{\pi}{2}, \text{то} \phi_i = 0$$

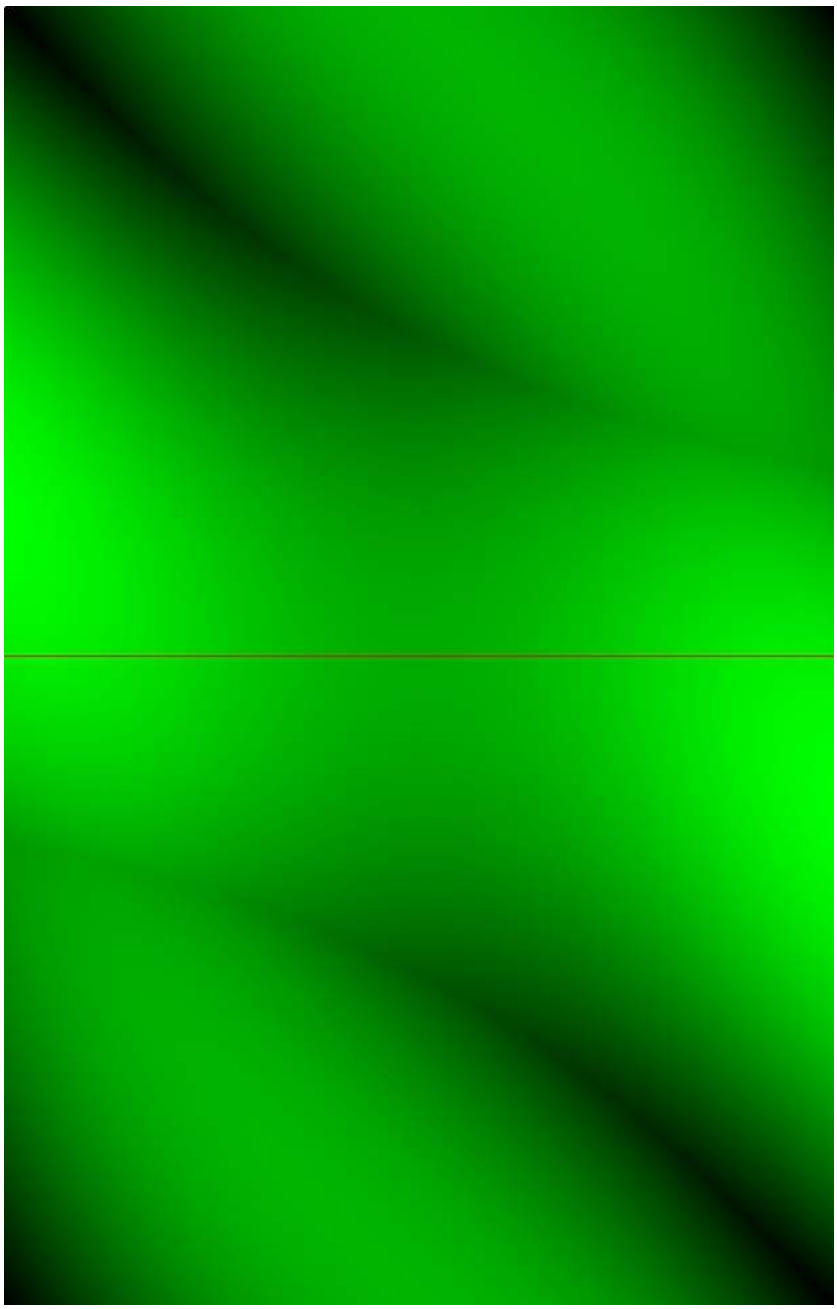
Для второго светильника угловой размер вычисляется аналогично, только x заменяется на l-x, а αна -α.

Точное решение полученной системы тригонометрических уравнений требует нахождения максимума функции нескольких переменных и выходит за рамки школьного курса. Однако приближенное решение является доступным.

Простым, но эффективным решением будет приблизительное нахождение максимумов функции численно.

Для примера приведём численное решение этой задачи:

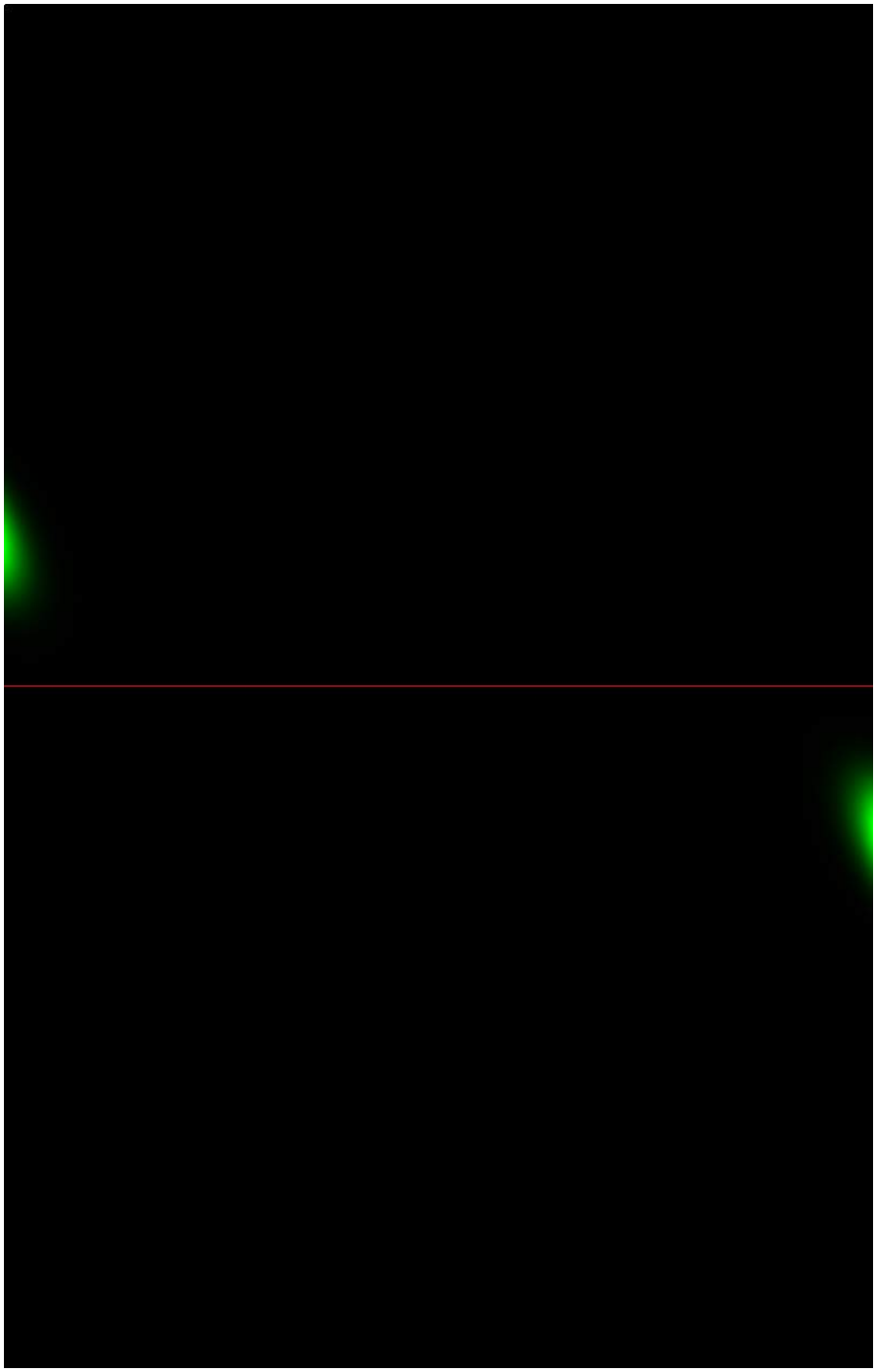
На рисунках изображена карта значений освещенности солнечной панели в зависимости от значений угла наклона (вертикальная ось) и расстояния от крайней левой точки стенда (горизонтальная ось). По оси абсцисс — положение солнечной батареи вдоль продольной оси стенда. По оси ординат — угол наклона солнечной батареи; самое верхнее значение соответствует наклону в 90° влево, самое нижнее — 90°. Горизонтальная красная линия в центре рисунка соответствует горизонтальному расположению солнечной панели.



Значение функции закодировано цветом: чем ярче зелёный цвет, тем оно больше.

Яркость цвета нормирована до единицы на размах функции освещённости в рассматриваемом диапазоне. Нулевая яркость (полностью чёрный цвет) соответствует минимальному значению функции, максимальная яркость (чистый зелёный цвет) — максимальному.

На втором рисунке яркость цвета возведена в степень 128, чтобы выделить только области вокруг максимумов функции.



Видно, что оптимально будет расположить солнечную панель как можно ближе к одному из светильников и повернуть примерно на 30° ко второму. С учётом физических погрешностей, присутствующих на стенде, решения такой точности уже достаточно.

Подобное решение можно получить не обладая навыками программирования, используя, например, средства Excel.

Такую карту значений можно использовать и для нахождения наилучшего расположения солнечной панели в условиях, когда оптимальное недоступно — например, если там уже расположена другая электростанция.

Следует заметить, что в реальности светодиоды обладают диаграммой направленности, которая к тому же искажается защитным стеклом светильника (последнее приводит к сложности вычисления диаграммы направленности светильника даже при использовании

диаграммы направленности, предоставленной производителем светодиодов в документации). Влияние её можно оценить экспериментально.

4.1.1.3. Вычисление потерь в сети

Эта задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при проектировании энергосистемы и при участии в аукционе.

На одной ветке сети могут располагаться только генераторы или только потребители. Исключением является миниподстанция, она может располагаться на одной ветке с любыми объектами. Ветки миниподстанции исключением не являются.

Потери вычисляются отдельно для каждой ветки, а затем суммируются. Потери для одной ветки растут как квадрат суммарной мощности на ветке. При мощности на ветке в 0 МВт потери равны нулю. При мощности на ветке в 18 МВт или более они составляют 20% от суммарной мощности на ветке.

Вычислять потери нужно как в скрипте на каждом такте игры, так и перед игрой для нахождения оптимальной конфигурации сети.

Пример решения

```
from powerbalance import *
import operator as o
#Импортируем наш же модуль
#Нам оттуда нужны функции по работе со случайными величинами
def getPower(code,tick):
    switcher = {
        'h' : [(-v,p) for (v,p) in fromForecast(houses[tick])],
        'b' : [(-v,p) for (v,p) in fromForecast(hospitals[tick])],
        'f' : [(-v,p) for (v,p) in fromForecast(factories[tick])],
        'w' : powerWind(tick),
        's' : powerSun(tick)
    }
    return switcher.get(code,[(0,1)])
def linePower(codes,tick):
    power=[(0,1)]
    for c in codes:
        power=fuzzyop(power,getPower(c,tick),o.add)
    return rough(power)
#Пример для отладки номер позиции в массиве = номер линии
#Закодирована конфигурация сети
exampleLines = ["h","f","s","ss","hf"]
def countLosses(lines, mini,tick):
    miniPower =
rough(fuzzyop(linePower(lines[3],tick),linePower(lines[4],tick),o.add))
    losses = [(0,1)]
    for i in range(0,5):
        power = linePower(lines[i],tick)
        if i==mini:
            power = rough(fuzzyop(power,miniPower,o.add))
        newLosses = rough([((0.2/(18^2)) * v**2,p) for (v,p) in power])
        losses = rough(fuzzyop(losses,newLosses,o.add))
    return losses
#Напечатать результат
#print(countLosses(exampleLines,3,55)
```

4.1.1.4. Определение взаимосвязи между яркостью освещения и генерацией солнечных батарей

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 1–3— при проектировании энергосистемы и при участии в аукционе.

Вырабатываемая мощность солнечных батарей линейно зависит от напряжения на солнечных панелях модели солнечной электростанции на стенде. Напряжение на солнечных панелях по отношению к яркости светильников, строго говоря, нелинейно. Однако характеристики солнечных панелей, измеряющих цепей и светильников подобраны так, что отклонение реальных значений от линейной их аппроксимации составляет не более 2%.

Однако в дальнейшем измерительная система осложняется несколькими факторами:

1. Внутри измерительных цепей солнечные панели подключены через балласт, такая измерительная цепь имеет инерцию: при резком изменении яркости светильников с максимальной до минимальной время полного разряда ёмкостей солнечных панелей составляет около 5 секунд. Для точной оценки генерации солнечных батарей нужно рассматривать солнечную батарею как RC-цепь.

2. Для нивелирования случайных ошибок измерения, измерения напряжения на солнечных панелях проходит через кольцевой буфер (это стандартная техника, используемая при программировании измерительных цепей микроконтроллеров). Его характеристики таковы:

- Глубина буфера — 20 измерений.
- Частота измерений — 100 мс.
- Значения в кольцевой буфер записываются непрерывно
- Алгоритм вычисления значения:
    1. Скопировать кольцевой буфер в «зеркало», чтобы новые значения, записываемые в буфер, не влияли на расчёт.
    2. Отсортировать значения.
    3. Отбросить 5 наименьших.
    4. Отбросить 5 наибольших.
    5. Вернуть среднее оставшихся 10 значений.

Точный расчёт требует построения из значений прогнозов и значений яркости солнца графика поведения RC-цепи, на котором вычисляется функция, описанная алгоритмом. Это решение, несмотря на значительное соответствие модели физической системе, является неэффективным: физические погрешности в системе очень велики и «забивают» все тонкости модели, такое решение будет вычислительно дорогим (а значит есть риск, что скрипт будет запаздывать, особенно при использовании в скрипте и других вычислительных алгоритмов), а главное — такое решение намного сложнее и дольше в корректной реализации, чем любое приближенное. В условиях олимпиады, когда у участников очень мало времени, эта задача является «ловушкой» для тех, у кого недостаточно навыков в оценке сложности задачи и планировании времени.

Рекомендуемым решением задачи вычисления генерации следующего хода будет линейная комбинация значений прогноза погоды на следующий ход и значений погоды на текущий и прошлый ход. Её можно вычислить на основании данных предыдущих игр.

«Правилом правой руки» будет просто использование для вычисления генерации следующего хода линейной функции от яркости солнца на текущем ходу: суммарные задержки в измерительной системе примерно равны длительности такта.

Например, вычисленные «задним числом» коэффициенты линейной комбинации от прогнозов погоды за все финальные игры составляют:

1. 0,12 от погоды на такт, для которого вычисляем прогноз.
2. 0,94 от погоды за предыдущий такт

3. 0,07 от погоды за пред-предыдущий такт
4. 0 для остальных тактов.

Сумма коэффициентов не равна единице, так как их сумма зависит от реально измеряемой эффективности конвертации солнечной энергии в электрическую.

Средняя величина ошибки такого набора коэффициентов между прогнозируемой и реальной генерацией на играх финала составила 19,2%, что даже несколько меньше заданной ошибки прогноза.

Пример решения

Программа написана на языке Haskell

```
module Main where
import System.Environment
import System.IO
import Data.Ord
import Data.List
--Файл с данными
testFile = "/home/user/graphs.txt"
main :: IO ()
main = do
  h <- openFile testFile ReadMode
  hSetEncoding h utf8
  contents <- hGetContents h
  let loglines = tail . lines $ contents
  let grs = map readOneLine loglines
  print grs
  print $ findBest grs
  --print $ searchBest 10 ( metrics grs ) [] $ allCombs 5 10
data Gr = Gr
  { sun
  , sunGen :: Float
  } deriving ( Show )
readOneLine :: String -> Gr
readOneLine str = Gr w wg
  where
  (_:_:w:wg:_) = map read $ words str
data LC = LC [ Float ]
  deriving ( Show )
rlc ( LC x ) = LC ( reverse x )
theoreticDiff :: [ Gr ] -> LC -> Maybe Float
theoreticDiff grs (LC lcs)
  | length grs < length lcs = Nothing
  | otherwise = Just $ (^2) $ sum sunny - sunGen ( last grs )
  where
  pairs = zip grs $ reverse lcs
  ony ( gr, coeff ) = sun gr * coeff
  sunny = map ony pairs
metrics :: [ Gr ] -> LC -> Float
metrics [] _ = 0
metrics grs lc = case theoreticDiff grs lc of
  Nothing -> 0
  Just val -> val + metrics ( tail grs ) lc
allCombs :: Int -> Int -> [ LC ]
allCombs len steps = map LC $ map normalize $ sequence $ replicate len list
  where
  normalize :: [ Float ] -> [ Float ]
  normalize fs | sum fs == 0 = fs
               | otherwise = map (/ sum fs ) fs
  list = map fromIntegral [0..steps-1]
findBest :: [ Gr ] -> LC
findBest grs = minimumBy ( comparing ( metrics grs )) $ allCombs 10 3
searchBest :: Int -> ( a -> Float ) -> [ a ] -> [ a ] -> [ a ]
searchBest _ _ accum [] = accum
```

```
searchBest limit f accum (lc:lcs)
  | length accum < limit = searchBest limit f (lc:accum) lcs
  | otherwise = searchBest limit f ( tail $ sortBy ( comparing f ) (lc:accum))
lcs
```

4.1.1.5. Определение взаимосвязи между скоростью ветра и генерацией ветряков

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 1–3 — при проектировании энергосистемы и при участии в аукционе.

Задача вычисления генерации ветровых электростанций по данным погоды похоже на такую же задачу для солнечных батарей, но является более сложной.

Для получения величины генерации используется скорость вращения анемометра, который установлен на модели ветровой электростанции. Устройство анемометра — вертикально-осевой; его лопасти устроены так, что скорость его вращения линейно пропорциональна скорости ветра (если считать поток ветра гомогенным). В случае постоянной негомогенности ветрового потока (когда анемометр не перемещается) скорость вращения анемометра также линейна по отношению к максимальной скорости ветра в потоке.

Измерительная система — инерциальная система вращения с трением. Её параметры нужно вычислять по данным прошедших игр. Время полной остановки анемометра с максимальной скорости вращения составляет около 12 тактов игры (оно зависит от максимальной скорости ветра в месте расположения анемометра). Время полного разгона аналогично составляет около 4 тактов игры.

Значения, измеряемые контроллером модели ветровой электростанции, также проходят через кольцевой буфер:

- Глубина буфера — 100 значений.
- Частота измерений — 10 Гц.
- Значения в кольцевой буфер записываются непрерывно.
- Результат измерения вычисляется из кольцевого буфера по следующему алгоритму:
    1. Скопировать кольцевой буфер в «зеркало», чтобы новые значения, записываемые в буфер, не влияли на расчёт.
    2. Отсортировать значения.
    3. Отбросить 25 наименьших.
    4. Отбросить 25 наибольших.
    5. Вернуть среднее оставшихся 50 значений.

Генерируемая мощность пропорциональная кубу скорости вращения анемометра. При достижении максимального уровня генерации (15 МВт) мощность далее не растёт.

Зависимость генерации от скорости ветра можно оценить либо эвристически, либо как линейную комбинацию от погоды за последние 10-15 тактов игры на основании данных прошедших игр.

Ветровую электростанцию возможно установить так, что максимальная мощность будет достигаться даже при сравнительно небольших скоростях ветра (по данным погоды). Вычисление коэффициента в зависимости генерации от скорости ветра необходимо делать экспериментально. Это можно оценить предварительными экспериментами, либо заложить процедуру оценки в управляющий скрипт, чтобы он делал её на каждом такте.

Например, вычисленные «задним числом» коэффициенты линейной комбинации от прогнозов погоды за все финальные игры составляют для наилучшего ветрогенератора:

1. 0,09 от погоды на такт, для которого вычисляем прогноз.

2. 0,32 от погоды за предыдущий такт
3. 0,49 от погоды за пред-предыдущий такт
4. 0,41 для следующего такта
5. 0,27 для следующего такта
6. 0,16 для следующего такта
7. Менее 0,05 для остальных тактов.

При вычислениях необходимо учитывать тот факт, что генерация ветровых электростанций ограничена правилами на уровне 15 МВт. Поэтому если прогнозируемая генерация превышает этот уровень, надо считать, что спрогнозировано именно 15.

Сумма коэффициентов не равна единице, так как их сумма зависит от реально измеряемой эффективности конвертации солнечной энергии в электрическую.

Средняя величина ошибки такого набора коэффициентов между прогнозируемой и реальной генерацией на играх финала составила 11,6%, что, с одной стороны, связано с большой инерциальностью физического анемометра. С другой стороны, из-за кубической зависимости генерации от силы ветра, любые погрешности «в середине» диапазона скоростей ветра очень сильно влияют на прогнозируемую мощность. Если скорость ветра мала или велика, погрешности влияют очень слабо.

Пример решения

Программа написана на языке Haskell

```
module Main where
import System.Environment
import System.IO
import Data.Ord
import Data.List
testFile          =          "/home/user/ips2018/semifinal-1/graphs-P2-2018-02-27
13:07:17.385870799 UTC.txt"
main :: IO ()
main = do
  h <- openFile testFile ReadMode
  hSetEncoding h utf8
  contents <- hGetContents h
  let loglines = tail . lines $ contents
  let grs = map readOneLine loglines
  print grs
  print $ findBest grs
  --print $ searchBest 10 ( metrics grs ) [] $ allCombs 5 10
data Gr = Gr
  { sun
  , sunGen :: Float
  } deriving ( Show )
readOneLine :: String -> Gr
readOneLine str = Gr w wg
  where
  (_:_:w:wg:_) = map read $ words str
data LC = LC [ Float ]
  deriving ( Show )
rlc ( LC x ) = LC ( reverse x )
theoreticDiff :: [ Gr ] -> LC -> Maybe Float
theoreticDiff grs (LC lcs)
  | length grs < length lcs = Nothing
  | otherwise = Just $ (^2) $ sum sunny - sunGen ( last grs )
  where
  pairs = zip grs $ reverse lcs
  ony ( gr, coeff ) = sun gr * coeff
  sunny = map ony pairs
metrics :: [ Gr ] -> LC -> Float
metrics [] _ = 0
metrics grs lc = case theoreticDiff grs lc of
```

```
  Nothing -> 0
  Just val -> val + metrics ( tail grs ) lc
allCombs :: Int -> Int -> [ LC ]
allCombs len steps = map LC $ map normalize $ sequence $ replicate len list
  where
  normalize :: [ Float ] -> [ Float ]
  normalize fs | sum fs == 0 = fs
               | otherwise = map (/ sum fs ) fs
  list = map fromIntegral [0..steps-1]
findBest :: [ Gr ] -> LC
findBest grs = minimumBy ( comparing ( metrics grs )) $ allCombs 10 3
searchBest :: Int -> ( a -> Float ) -> [ a ] -> [ a ] -> [ a ]
searchBest _ _ accum [] = accum
searchBest limit f accum (lc:lcs)
  | length accum < limit = searchBest limit f (lc:accum) lcs
  | otherwise = searchBest limit f ( tail $ sortBy ( comparing f ) (lc:accum))
lcs
```

4.1.1.6. Гравитационный накопитель

Гравитационный накопитель — система накопления гравитационной энергии, являлась частью стенда-тренажера 2017-2018 года. Вследствие того, что каждая команда приписана к своему устройству накопителя, для нивелирования погрешностей и ошибок измерения его поведение на аппаратном и программном уровнях максимально приближенно к линейному. Гравитационный накопитель дает возможность каждой команде работать с реальными алгоритмами накопления энергии. Точность алгоритмического использования накопителя дает достаточно большой вклад в уменьшение энергообмена с внешней сетью, балансирует разрыв между генерацией ВИЭ и потребителями для каждой команды эта задача решается отдельно.

4.1.2. Математика

4.1.2.1. Вычисление энергетического баланса на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы.

Всё время игры командам доступны все данные о прогнозах погоды и действующих контрактах. Из них можно вычислить прогноз дефицита или профицита мощности на каждый такт. Для этого нужно на основании составленных заранее игроками моделей вычислить из прогнозов погоды прогнозы генерации. Из результирующего множества случайных величин (вероятная генерация для каждой электростанции и вероятное потребления для каждого потребителя) нужно вычислить их сумму. Она будет представлять собой распределение вероятностей профицита/дефицита мощности в системе. Взаимодействие с гравитационным накопителем, внешней энергосистемой и биржей электроэнергии являются по отношению к этой случайной величине константами.

Пример решения

Приведённый здесь модуль называется powerbalance и будет использоваться почти во всех остальных решениях

```
import operator as o
#Этот модуль будет использоваться почти во всех остальных решениях
#Операция на «нечётких множествах»
def fuzzyop(fuz1, fuz2, op):
    result = []
    for (val1,prob1) in fuz1:
        for (val2,prob2) in fuz2:
```

```
            result.append((op(val1,val2),prob1*prob2))
    return result
#Оптимизировать представление случайной величины
def squash(d):
    d.sort(key=lambda x: x[0])
    l=len(d)
    newD = []
    acc = 0
    for i in range(len(d)-1):
        if d[i][0] != d[i+1][0]:
            x = (d[i][0], acc+d[i][1] )
            newD.append(x)
            acc = 0
        else:
            acc += d[i][1]
    x = (d[len(d)-1][0], acc+d[len(d)-1][1])
    newD.append(x)
    return newD
#Округление по произвольной базе
def myround(value,step):
    mod = value % step
    div = value // step
    if mod > step / 2:
        return step * ( div + 1 )
    else:
        return step * div
#Огрубить случайную величину. чтобы ускорить вычисления
#Желательно также убирать вероятности ниже порогового значения
roughstep = 0.2
def rough(fuz):
    result = [(myround(val,roughstep),prob) for (val,prob) in fuz]
    return squash(result)
#Получить численную случайную величину из прогноза
stepsize = 0.1
def fromForecast(forecast):
    lower = forecast / 1.2
    upper = forecast / 0.8
    stepsLower = []
    x = forecast
    while x >= lower:
        stepsLower.append(x)
        x -= stepsize
    lowers = []
    for l in stepsLower:
        lowers.append((l,1/len(stepsLower)))
    stepsUpper = []
    x = forecast + stepsize
    while x <= upper:
        stepsUpper.append(x)
        x += stepsize
    uppers = []
    for l in stepsUpper:
        uppers.append((l,1/len(stepsUpper)))
    result = lowers + uppers
    return result
#Прочитать прогнозы ветра
with open('wind.txt') as f:
    tmp = f.read().splitlines()
    wind = [ eval(x) for x in tmp ]
#Прочитать прогнозы солнца
with open('sun.txt') as f:
    tmp = f.read().splitlines()
    sun = [ eval(x) for x in tmp ]
#Прочитать прогнозы потребления домов
```

```
with open('houses.txt') as f:
    tmp = f.read().splitlines()
    houses = [ eval(x) for x in tmp ]
#Прочитать прогнозы потребления заводов
with open('factories.txt') as f:
    tmp = f.read().splitlines()
    factories = [ eval(x) for x in tmp ]
#Прочитать прогнозы потребления больниц
with open('hospitals.txt') as f:
    tmp = f.read().splitlines()
    hospitals = [ eval(x) for x in tmp ]
#Линейные коэффициенты генерации для ветра и солнца
coefSun = [0.12, 0.94, 0.7]
coefWind = [0.09, 0.32, 0.49, 0.41, 0.27, 0.16]
# Вычисление прогноза генерации
def powerForecast(coefficients,values,tick):
    power = [(0,1)]
    for i in range(0,len(coefficients)):
        if tick-i < 0:
            val=0
        else:
            val=values[tick-i]
        part = fuzzyop(fromForecast(val),[(coefficients[i],1)],o.mul)
        power = fuzzyop(power,part,o.add)
    return rough(power)
# Вычисление прогноза генерации
def powerSun(tick):
    return powerForecast(coefSun,sun,tick)
# Вычисление прогноза генерации
linear = 0.44 #коэффициент при x^3
def powerWind(tick):
    tmp = powerForecast(coefWind,wind,tick)
    return [(max(15,linear*(x**3)),p) for (x,p) in tmp ]
class Network:
    houses = 0
    hospitals = 0
    factories = 0
    wind = 0
    solar = 0
    def __init__(self,h,f,b,w,s):
        self.houses = h
        self.hospitals = b
        self.factories = f
        self.wind = w
        self.solar = s
# Задаём состав сети
network = Network(2,2,1,1,2)
def powerBalance(net):
    power = [(0,1)]
    for tick in range(0,3):
        for _ in range(0,net.houses):
            power=fuzzyop(power,fromForecast(houses[tick]),o.sub)
        power=rough(power)
        for _ in range(0,net.factories):
            power=fuzzyop(power,fromForecast(factories[tick]),o.sub)
        power=rough(power)
        for _ in range(0,net.hospitals):
            power=fuzzyop(power,fromForecast(hospitals[tick]),o.sub)
        power=rough(power)
        for _ in range(0,net.solar):
            power=fuzzyop(power,powerSun(tick),o.add)
        power=rough(power)
        for _ in range(0,net.wind):
            power=fuzzyop(power,powerWind(tick),o.add)
```

```
    return rough(power)
def expect(fuz):
    result = 0
    for (v,p) in fuz:
        result += v*p
    return result
#Напечатать математическое ожидание энергетического баланса
#print(expect(powerBalance(network)))
```

4.1.2.2. Вычисление экономического баланса энергосистемы на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 2–3 — участии в аукционе.

Далее нужно решить задачу нахождения вероятностного распределения экономических потерь. Это делается почти тривиально: каждый мегаватт избыточной мощности несёт убыток в 1 у.е., а каждый мегаватт недостаточной — 10 у.е. К этим значениям нужно добавить стоимость электроэнергии, закупленной/проданной во внешней энергосистеме. Далее математическое ожидание получившейся случайной величины нужно минимизировать, используя параметры направляемой/извлекаемой мощности из гравитационного накопителя и закупаемой/продаваемой мощности во внешней энергосистеме.

Эта задача немного проще, чем кажется из-за того, что гравитационный накопитель экономически намного выгоднее взаимодействия с внешней энергосистемой, имеет смысл закупать/продавать электроэнергию в ней только в случае невозможности сделать это через накопитель. Эти два параметра оказываются связаны в один — «балансирующее воздействие на энергосистему», которое можно разбить на гравитационный накопитель и внешнюю энергосистему «задним числом» после решения этой задачи.

Эту задачу (нахождение балансирующего воздействия, минимизирующего математическое ожидание экономических потерь) недостаточно решить аналитически, алгоритм решения нужно реализовать также в управляющем скрипте (и других вспомогательных программах), что для большинства школьников является нетривиальной и неустойчивой к ошибкам задачей. Немного проще её решить «перебором» — перебрав все значения балансирующего воздействия в размахе случайного распределения дефицита/профицита мощности в системе с фиксированным шагом, например, 0,1. Такой точности будет вполне достаточно, такое решение можно реализовать быстрее, чем точное, и впоследствии при наличии времени его можно точным заменить.

В дальнейшем на основании этих данных можно вычислить распределение вероятностей прибыли за всю игру.

Пример решения

Программа представляет собой модуль costbalance, который будет использоваться в примере решения задачи нахождения предельной стоимости объекта на аукционе.

```
import powerbalance as p
# Импорт нашего же модуля
# Константы их правил
buyCost = 5
buyCostFast = 10
sellCost = 2
sellCostFast = 1
#Вычисление стоимости одного исхода энергобаланса
def makeCost(value,adjust):
    result = 0
    if adjust > 0:
        result -= adjust * buyCost
```

```
    else:
        result += adjust * sellCost
    diff = value + adjust
    if diff < 0:
        result += diff * sellCostFast
    else:
        result -= diff * buyCostFast
    return result
#Вычисление распределения вероятностей расходов/прибылей
def costBalance(power,adjust):
    return [ (makeCost(v,adjust),p) for (v,p) in power]
#Простой и глупый алгоритм жадного спуска
def greed(a,b,c,x):
    if b < a and b < c:
        return 0
    if b > a:
        return -x
    if b < c:
        return x
    else:
        return -x
#Находим какой-то из минимумов коррекции баланса энергосистемы
def greedilyFindAdjust(net):
    adjStep = 0.1
    adj = 0
    power = p.powerBalance(p.network)
    print('Preparations are complete')
    while True:
        g                    =                    greed(costBalance(power,adj-
adjStep),costBalance(power,adj),costBalance(power,adj+adjStep),adjStep)
        if g == 0:
            return adj
        else:
            print(adj)
            adj += g
#Напечатать результат
print(greedilyFindAdjust(p.network))
```

4.1.2.3. Нахождение оптимальной конфигурации сети с точки зрения электрических потерь

Эта задача возникает на 1-м этапе игры, при анализе прогнозов. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 2–3 — участии в аукционе.

Сами по себе электрические потери в энергосистеме не важны — важны их экономические последствия. В разные моменты времени эти последствия могут быть различны. Если они ведут к увеличению закупок электроэнергии, то экономические потери от них — 5 у.е. за 1 МВт потерь. Если к уменьшению продаж электроэнергии — 2 у.е за 1 МВт потерь. Эти цифры будут таковыми, если:

1. Скрипт корректно управляет продажей и закупкой энергии во внешней энергосистеме — отсутствует или минимальна автоматическая балансировка.
2. Отсутствуют контракты с участием рассматриваемой команды.

В случае нарушения этих условий экономические потери от электрических потерь будут другими и могут вычисляться более сложно.

Важно заметить, что решение о конфигурации сети принимается командой один раз перед началом игры, поэтому расчёты делаются на основании прогнозов. Величина

экономических потерь в общем случае вычисляется из полного энергобаланса энергосистемы с учётом собственной оценки командой эффективности управляющего скрипта и прогноза экономического взаимодействия с другими командами.

Например, в энергосистеме из трёх объектов очевидно нужно располагать все на разных ветках.

В энергосистеме из трёх солнечных электростанций, двух заводов и двух домов решение следующее:

1. Две солнечные электростанции нужно расположить на одной из веток миниподстанции.
2. На второй ветке миниподстанции нужно расположить два потребителя. Это либо заводы, либо дома — их нужно подбирать исходя из конкретных прогнозов потребления, чтобы их потребление максимально совпадало с генерацией солнечных электростанций (детальные вычисления слишком массивны, чтобы приводить их в качестве примера, но тривиальны по сути). Как правило, это оказываются заводы — у домов пик потребления приходится на вечер, тогда как у заводов — на полдень, как и пик генерации солнечных электростанций.
3. Миниподстанцию нужно расположить на одной из веток основной подстанции. Для примера, пусть это будет первая. На этой же ветке нужно расположить солнечную электростанцию или одного из потребителей — это зависит от того, больше миниподстанция как объект энергосистемы потребляет или генерирует. Опять таки, вычисления тривиально проводятся на основании прогнозов погоды и генерации.
4. Оставшиеся два объекта нужно расположить на второй и третьей ветке основной подстанции. Задача решена.

В общем случае задачу можно решить даже перебором на компьютере, особенно если энергосистема невелика.

Пример решения

```
from powerbalance import *
import operator as o
#Импортируем наш же модуль
#Нам оттуда нужны функции по работе со случайными величинами
def getPower(code,tick):
    switcher = {
        'h' : [(-v,p) for (v,p) in fromForecast(houses[tick])],
        'b' : [(-v,p) for (v,p) in fromForecast(hospitals[tick])],
        'f' : [(-v,p) for (v,p) in fromForecast(factories[tick])],
        'w' : powerWind(tick),
        's' : powerSun(tick)
    }
    return switcher.get(code,[(0,1)])
def linePower(codes,tick):
    power=[(0,1)]
    for c in codes:
        power=fuzzyop(power,getPower(c,tick),o.add)
    return rough(power)
#Пример для отладки номер позиции в массиве = номер линии
#Закодирована конфигурация сети
exampleLines = ["h","f","s","ss","hf"]
def countLosses(lines, mini,tick):
    miniPower                                                                    =
rough(fuzzyop(linePower(lines[3],tick),linePower(lines[4],tick),o.add))
    losses = [(0,1)]
    for i in range(0,5):
        power = linePower(lines[i],tick)
        if i==mini:
            power = rough(fuzzyop(power,miniPower,o.add))
        newLosses = rough([((0.2/(18^2)) * v**2,p) for (v,p) in power])
        losses = rough(fuzzyop(losses,newLosses,o.add))
```

```
    return losses
def findConfig(net):
    best = 1e12
    bestlines = None
    tmp = ""
    for w in range(0,net.houses):
        tmp += 'h'
    for w in range(0,net.factories):
        tmp += 'f'
    for w in range(0,net.hospitals):
        tmp += 'b'
    for w in range(0,net.wind):
        tmp += 'w'
    for w in range(0,net.solar):
        tmp += 's'
    all = 5 ^ (len(tmp)+1)
    for x in range(0,all):
        var = x
        lines = ["","","","",""]
        for w in tmp:
            lines[var%5] += w
            var //= 5
        allLosses = 0
        for tick in range(0,168):
            print(tick)
            allLosses += expect(countLosses(lines,var,tick))
        if best > allLosses:
            best = allLosses
            bestlines = lines
    return bestlines
#Напечатать результат
#print(findConfig(network))
```

4.1.2.4. Расчёт предельной цены объекта

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Это задача нахождения предельной цены контракта объекта на аукционе — такой цены, приобретение контракта по которой ожидаемо не принесёт ни прибылей, ни убытков. Это расширение над задачей нахождения полного экономического баланса энергосистемы — достаточно вычислить суммарную прогнозируемую прибыль энергосистемы с этим объектом и без него. Для электростанций разницу между этими величинами нужно разделить на 168 (число тактов в игре). Для потребителей — разделить на суммарное потребление его за всю игру (это случайная величина; для примерной оценки её можно заменить на математическое ожидание, но на этом этапе у команд уже должно быть в избытке опыта вычислений со случайными величинами).

Получившееся число: для электростанций — максимальная осмысленная цена, для потребителей — минимальная.

Пример решения

```
#Импортируем собственный модуль расчёта энергетического баланса
from powerbalance import *
#Импортируем собственный модуль расчёта экономического баланса
from costbalance import *
import operator as o
networkBefore = Network(2,2,1,1,2)
networkAfter = Network(2,2,1,1,3)
def findCumulativeCost(net1,net2):
    before = costBalance(networkBefore)
    after = costBalance(networkAfter)
    powerBefore = powerBalance(networkBefore)
```

```
    powerAfter = powerBalance(networkAfter)
    if net1.solar + net1.wind != net2.solar + net2.wind:
        return
fuzzyop((fuzzyop(after,before,o.sub)),(fuzzyop(powerBefore,powerAfter,o.sub)),o.
div)
    else:
        return fuzzyop(fuzzyop(after,before,o.sub),[(168,1)],o.div)
print(findCumulativeCost(networkBefore,networkAfter))
```

4.1.2.5. Составление и адаптация стратегии для аукционов

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Хотя команды могут найти оптимальную энергосистему для любого набора прогнозов погоды, эту энергосистему им ещё нужно «отвоевать» у конкурентов. Эта игра в целом относится к рефлексивным, игры этого класса не имеют устойчивого решения, а использование смешанных стратегий едва ли оправдано на одном прогоне игры. Участникам нужно анализировать поведение оппонентов, предугадывать их цели, и искать способы помешать целям оппонентов и защитить свои. Для этого можно создать несколько вспомогательных инструментов:

1. Нахождение эффективных конфигураций энергосистем. Не только оптимальной, но всех, которые достаточно хороши. Все команды будут стремиться к какой-то из них, и для любой промежуточной ситуации на аукционе можно предположить, к какой из них будут стремиться оппоненты.

2. Нахождение оптимальной ставки на аукционе, исходя из ценности лота для себя и оппонентов. Ценности лота для оппонентов можно оценить из предположения об энергосистеме, к которой они стремятся, с использованием решения задачи расчёта прогноза рентабельности. В этих условиях оптимальной ставкой будет максимальная ставка всех оппонентов, при условии, что она не превышает собственной максимальной ставки. Далее участникам будет интересно от этой ставки отклониться, чтобы рискнуть и увеличить выгоду, либо нарушить стратегию оппонентов (от одного приобретённого по некорректной цене объекта, согласно правилам аукциона, можно избавиться в конце аукциона).

Также в течение всего времени аукциона командам нужно оценивать риск того, что целевую энергосистему составить не удастся, и при необходимости переходить к альтернативным вариантам. Такую задачу решить в общем случае в ограниченное время не представляется возможным, поэтому в ней на первое место выходит смекалка, скорость мышления и скорость принятия решений в команде.

4.1.2.6. Работа с биржей электроэнергии

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы.

Основная задача — составление предложений с выгодными энергетическими и временными параметрами, при привлекательных ценах.

Задача вычисления цены за 1 МВт энергии, которая будет выгодна другим игрокам, тривиальна и сепарабельна: цена должна попасть в «вилку», создаваемую ценами торговли с внешней энергосистемой, от 1 до 5 у.е. за 1 МВт. В случае конкуренции с другими игроками нужно устанавливать цены, более выгодные, чем у конкурентов. Для этого достаточно мониторить в автоматическом режиме действующие предложения на бирже электроэнергии.

Энергетические и временные параметры контракта нужно вычислять исходя из прогноза графика энергобаланса собственной энергосистемы, который, как правило, имеет как энергодефицитные области, так энергопрофицитные.

4.1.3. Информатика

4.1.3.1. Система поддержки принятия решений на аукционе

Это первая точка сборки решений предметных задач, возникающих в командном туре.

Основная задача таких систем — вычисление ценности лота при заданных параметрах энергосистемы.

Самый примитивный вариант такой системы может быть устроен следующим образом:

1. Имеются прогнозы погоды и потребления на следующую игру. Мы будем считать, что реальные значения будут точно соответствовать прогнозам.
2. Для каждого такта игры вычисляем генерацию. Например, солнечные батареи вычисляем из яркости солнца за предыдущий такт с коэффициентом 0,001 (1000 лк = 1 Мвт), а генерацию от ветряка считаем равной генерации за прошлый такт (поскольку за 1 такт она изменяется незначительно).
3. Вычисляем энергобаланс в каждый такт игры.
4. Вычисляем изменение счёта в каждый такт игры.
5. Добавляем в энергосистему интересующий нас объект.
6. Повторяем шаги 1–4.
7. Сравниваем результаты для энергосистемы при наличии и без наличия интересующего объекта.
    1. Для электростанций оптимальная цена есть их цена плюс разница результатов энергосистем, делённая на число тактов игры.
    2. Для потребителей оптимальная цена есть их цена плюс разница результатов игры, делённая на потреблённую потребителем энергию.

Хорошую систему от примитивной могут отличать следующие характеристики:

- Учёт погрешностей прогнозов. Расчёт наихудшего варианта, наилучшего, наиболее вероятного и других статистических характеристик.
- Использование более точных оценок генерации.
- Расчёт стоимости лота для энергосистем оппонентов — чтобы выиграть лот, нужно ставить не собственную цену, а цену, большую, чем у оппонентов. Для этого необходимо оценивать ценность лота для оппонентов.
- Прогноз эффективности задуманной энергосистемы.
- Расчёт величины риска в зависимости от размера ставки — для этого нужно использовать данные об энергосистемах оппонентов и опыт взаимодействия с ними.

Важно, что такая система является лишь **помогает** принимать решения, и не гарантирует того, что команда с наилучшей реализацией этой задачи наиболее эффективно проведёт аукцион.

Разработанные командами программы варьировались по сложности от простых таблиц Excel, до веб-сервисов с элементами ИИ.

4.1.3.2. Управляющий скрипт

Это вторая точка сборки предметных задач, возникающих в командном туре.

Задача управляющего скрипта — используя возможность управления продажей электроэнергии во внешнюю энергосистему, управления гравитационным накопителем и составления контрактов между игроками, максимизировать число очков, которое получит команда за командный тур.

Если эта задача не решена, то результаты команды будут плохими независимо от результатов аукциона и глубины решения предметных задач командного тура.

Самый примитивный вариант управляющего скрипта выглядит следующим образом

1. Команда во внешней программе на основании прогнозов вычисляет параметры управления (накопитель и внешняя энергосистема), например, в Excel.
2. Команда составляет скрипт, который состоит из двух команд: установка величины перетока энергии во внешнюю энергосистему и величины перетока энергии в гравитационный накопитель.
3. Команда загружает в систему составленный скрипт.
4. Пункты 1–3 повторяются каждый 2–4 такта, в зависимости от быстродействия команды.

Этот вариант «скрипта» наиболее просто реализовать, однако у него плохая точность и отсутствует потенциал роста. Многие команды прибегли к такому способу управления энергосистемой, и ни одна из них не показала хороших результатов в финалах или полуфиналах.

Самый простой вариант «настоящего» скрипта может действовать, например, по следующему алгоритму:

1. Оценить генерацию на следующем такте. Вычислить энергобаланс следующего такта.
2. ЕСЛИ энергобаланс положителен, ПЕРЕЙТИ к п. 6
3. Попытаться ликвидировать дефицит из гравитационного накопителя.
4. Ликвидировать дефицит из внешней энергосистемы.
5. ЗАВЕРШИТЬ РАБОТУ
6. Попытаться ликвидировать профицит, перенаправив мощность в гравитационный накопитель.
7. Ликвидировать профицит, продав мощность во внешнюю энергосистему.
8. ЗАВЕРШИТЬ РАБОТУ

Хороший скрипт может обладать следующими характеристиками:

- Более точные оценки генерации.
- Использование распределения вероятных значений энергобаланса, чтобы вычислять средневзвешенную величину его коррекции: дефицитный энергобаланс обходится в 5 раз дороже профицитного, соответственно нужно минимизировать не модуль энергобаланса, а математическое ожидание экономических потерь в результате ошибок прогнозов.
- Такое управление гравитационным накопителем, которое полностью разряжает его к последнему такту игры.
- Оценка выгодности предлагаемых контрактов.
- Вычисление параметров энергосистем оппонентов для составления привлекательных предложений контрактов.

- Управление риском: в ряде случаев осмысленно использование неоптимальных характеристик управления, которые несмотря на то, что они снижают математическое ожидание счёта в командном этапе, увеличивают вероятность обойти другую команду.

Пример управляющего скрипта

```
import math
VERSION = '14'
g_c = 0
eng = 0
debug = [0, 0, 0]
koef = [0, 0, 0]
def loss(x):
    a = 0.00325848
    b = -0.0107377
    c = - 0.00175687
    return a * x * x + b * x + c
def doTick():
    global g_c, debug, eng, koef
    forecast_req_sum = 0
    house_w = forecast.houses[tick]
    forecast_req_sum += len(houses) * house_w
    factories_w = forecast.factories[tick]
    forecast_req_sum += len(factories) * factories_w
    hospitals_w = forecast.hospitals[tick]
    forecast_req_sum += len(hospitals) * hospitals_w
    req_sum = sum([x.value for x in houses]) + sum([x.value for x in factories])
+ sum([x.value for x in hospitals])
    if forecast_req_sum == 0:
        forecast_req_sum += 0.1
    koef_req = req_sum / forecast_req_sum
    forecast_req_sum = 0
    house_w = forecast.houses[tick + 1]
    forecast_req_sum += len(houses) * house_w
    factories_w = forecast.factories[tick + 1]
    forecast_req_sum += len(factories) * factories_w
    hospitals_w = forecast.hospitals[tick + 1]
    forecast_req_sum += len(hospitals) * hospitals_w
    req_sum = koef_req * forecast_req_sum
    wind_w = forecast.wind[tick]
    forecast_gen_wind = len(wind_gens) * wind_w
    sun_w = forecast.sun[tick - 1]  # Здесь смысл с задержкой солнечной
    forecast_gen_sun = len(sun_gens) * sun_w
    gen_sum_wind = sum([x.generation for x in wind_gens])
    gen_sum_sun = sum([x.generation for x in sun_gens])
    if forecast_gen_wind == 0:
        forecast_gen_wind += 0.1
    koef_gen_wind = gen_sum_wind / forecast_gen_wind
    if forecast_gen_sun == 0:
        forecast_gen_sun += 0.1
    koef_gen_sun = gen_sum_sun / forecast_gen_sun
    koef = [koef_req, koef_gen_wind, koef_gen_sun]
    wind_w = forecast.wind[tick + 1]
    forecast_gen_wind = len(wind_gens) * wind_w
    sun_w = forecast.sun[tick]  # Здесь смысл с задержкой солнечной
    forecast_gen_sun = len(sun_gens) * sun_w
    gen_sum = forecast_gen_sun * koef_gen_sun + forecast_gen_wind * koef_gen_wind
    energy = abs(gen_sum) - abs(req_sum) - abs(loss(req_sum + gen_sum))
    debug = [gen_sum, req_sum, loss(req_sum + gen_sum)]
    g_c = grav.charge
    v_grav = 0
    v_external = 0
```

```
if energy < 0:
    energy = abs(energy)
    if grav.charge > 0:
        if grav.charge >= 10:
            grab = energy
            if grab > 10:
                grab = 10
                energy -= 10
            else:
                energy = 0
            v_grav = grab
        else:
            grab = grav.charge
            energy -= grav.charge
            v_grav = grab
    eng = energy
    v_external = energy
else:
    free_grav = abs(grav.capacity) - grav.charge
    put = energy
    if free_grav >= 10:
        if put > 10:
            put = 10
            energy -= 10
        else:
            energy = 0
    elif free_grav > 0:
        if free_grav >= put:
            energy = 0
        else:
            put = free_grav
            energy -= free_grav
    v_external = -energy
    eng = energy
    v_grav = -put
# v_grav = 0 # отключить гравицапу
# v_external = 0 # отключить внешнее
k_g = 0.8
# k_g = 0.1
k_e = 0.85
# k_e = k_g
# k_e = 0.1
setExternal(k_e * v_external + k_g * v_grav)
# setGrav()
```

4.2. Критерии оценивания

1. Для четырёх команд, которые в полуфиналах турнира заняли 1 и 2 места (попали в финал турнира) оцениваются следующим образом: команда, которая решила финальную задачу лучше всех остальных, получает максимальное количество баллов — 60. Команда, справившаяся хуже всех, получает 30 баллов.

Игровые очки, начисляемые в экономической модели игры и являющиеся интегральной оценкой всех логистических, энергетических, физических, математических решений принятых командой. Игровые очки могут колебаться в широких пределах. Поэтому в финальном подсчете для команд-победителей полуфиналов турнира игровые очки переводятся в баллы от 30 до 60 по формуле: **$x=30+(c-m)/(M-m)\times 30$**,
где M — число игровых очков, набранных за финальную задачу командой-победителем,
m — число игровых очков, набранное командой, занявшей последнее место,

c — число игровых очков, набранное рассматриваемой командой (т.е. результат которой считаем по этой формуле). По этой формуле победитель всегда получает 60, проигравший всегда получает 30, а остальные от 30 до 60, пропорционально результату в турнире.

2. Для четырех команд, которые в полуфиналах турнира заняли 3 и 4 места (и не попали в финал турнира) проводился дополнительный финал турнира за 5–8 место, чтобы честно проранжировать команды. Победитель финала турнира за 5–8 место получал 30 баллов. Занявший последнее место в финале турнира за 5–8 место получал 0 баллов.
Остальные команды получали баллы по формуле: $\mathbf{x=(c-m)/(M-m)\times 30}$,
где M — число игровых очков, набранных за финал командой-победителем финала турнира за 5–8 место,
m — число игровых очков, набранное командой, занявшей последнее место,
c — число игровых очков, набранное рассматриваемой командой. По этой формуле победитель всегда получает 30, проигравший всегда получает 0, а остальные от 0 до 30, пропорционально результату в игре.