Технологии беспроводной связи

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 кодами, потоками и форматами данных, протоколами.

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

Анализ данных

Информация — главный ресурс, с которым мы работаем на этом профиле, и важно уметь анализировать и обрабатывать её в зависимости от задачи. При работе с этим разделом важно обращать внимание на формат данных, так как их представление может быть крайне разнообразно.

Кодирование-декодирование

Во время финальной задачи потребуется реализовать алгоритмы кодирования и декодирования, и особенности этих процессов проявлены в задачах этого раздела, начиная использованием уже готовых кодов, так и разработкой собственных алгоритмов для частных проблем передачи данных.

Алгоритмы

Финальная задача предполагает написание программного кода, 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.

Анализ кода

Правильный выбор кода в зависимости от задачи даёт половину решения. Эти задачи позволят рассмотреть коды в разных аспектах, увидеть типичные проблемы и, как следствие, более осознанно и осмысленно подойти к этой части финальной задачи.

Канал связи

Одна из составляющих процесса передачи данных — канал связи, обладающий своими характеристиками. Этот раздел содержит задачи, связанные с анализом этих характеристик, в основном отражённых в вероятностной форме. Поэтому при решении этих задач понадобятся как минимум базовые знания теории вероятностей.

Автономное управление

Часть финальной задачи составляет разработка алгоритма автономного управления спутником, качество которого напрямую влияет на общий результат, поэтому важно понимать приниципы автономности и обладать навыками создания самостоятельных алгоритмов. Хотя в некоторой степени это характерно для всех задач, задачи этого раздела направлены именно на автономику.

№ 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 работы со сложными математическими моделями. Необходимы знания по математике по следующим темам: параметрическое представление функции, тригонометрические функции, расстояние между точками. Необходимы базовые навыки программирования и знания по информатике по следующим темам: циклы, арифметика.	Анализ кодов на стенде ОВКС.
IV.2.	Нацелена на развитие навыков работы со сложными кодами и математическими моделями. Необходимы знания по математике по следующей теме: скалярное произведение векторов. Необходимы базовые навыки программирования и знания по информатике по следующим темам: циклы и ветвления, работа со строками.	Анализ кодов на стенде ОВКС, работа с потоком данных с помощью кодировщика и декодировщика на стенде УНКС.
IV.3.	Нацелена на развитие навыков алгоритмического мышления и работы с потоками данных. Из информатики необходим средний уровень навыков программирования и знания по информатике по следующим темам: циклы и ветвления, чтение данных, работа с массивами и текстовыми данными. Знание алгоритмов сжатия ускорит решение задачи.	Написание программ для стенда УНКС (в частности, кодировщика и декодировщика).
IV.4.	Нацелена на развитие навыков работы с двоичными и корректирующими кодами. Знание манчестерского кодирования и навык работы с корректирующими кодами ускорит решение задачи. Из информатики необходим базовый уровень навыков программирования и знания по информатике по следующим темам: циклы, чтение данных, работа с двоичными матрицами, бит паритета.	Анализ кодов на стенде ОВКС, работа с потоком данных с помощью кодировщика и декодировщика на стенде УНКС.
IV.5.	ННацелена на развитие навыков промышленного программирования и работы с текстовыми форматами данных. Из информатики требует базового уровня навыков программирования и знаний по следующим темам: циклы и ветвления, чтение данных, работа со строками и массивами.	Анализ кодов на стенде ОВКС, составление и анализ журнала слежения на стенде УНКС.

№ задачи	Знания и навыки, на выявление и развитие которых направлена задача	Задачи инженерного тура заключительного этапа, в которых применимо
IV.6.	Нацелена на развитие алгоритмики, а также информационного поиска. Необходимы базовые знания по математике. Из информатики необходимы базовый уровень навыков программирования и знания по следующим темам: циклы, ветвления, алгоритмы на графах.	Написание программ для стенда УНКС (в частности, алгоритма слежения).

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 одном из доступных языков (C, C++, Java, 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. Гиперштопор (12 баллов)

Темы: программирование, работа с моделью.

Условие

Во время путешествия в гиперпространстве космический молоковоз попал в четырёхмерный штопор, и его **траектория** по каждой из координат описывается уравнениями:

$$x(t) = \text{А}\cos(\text{Б}t + \text{В}) + \text{Г}\sin(\text{Д}t + \text{Е}) + \text{Ё}\cos(\text{Ж}t + \text{З})$$
$$y(t) = \text{И}\sin(\text{Й}t + \text{К}) + \text{Л}\cos(\text{М}t + \text{Н}) + \text{О}\sin(\text{П}t + \text{Р})$$
$$z(t) = \text{С}\sin(\text{Т}t + \text{У}) + \text{Ф}\cos(\text{Х}t + \text{Ц}) + \text{Ч}\cos(\text{Ш}t + \text{Щ})$$
$$w(t) = \text{Ъ}\cos(\text{Ы}t + \text{Ь}) + \text{Э}\sin(\text{Ю}t + \text{Я}) + \text{А}\sin(\text{Б}t + \text{В})$$

Все величины расстояния — в **световых годах**, времени — в **секундах**.

Чтобы выбраться из гиперпространства и спастись, молоковоз должен оказаться как можно **ближе** к Солнечной системе (её позиция соответствует **началу координат**). Для попытки выхода достаточно расстояния хотя бы в **световой год**, но чем ближе — тем **лучше**. Ваша задача — найти подходящий **момент времени**, учитывая, что в гиперпространстве молоковоз может провести **не более часа**. Если существует несколько подходящих моментов, выведите любое.

Формат входных данных

33 вещественных числа через пробел, константы уравнений соответственно буквам в русском алфавите.

Формат выходных данных

Единственное **вещественное** число — момент времени в секундах.

Задача оценивается исходя из числа пройденных тестов. Тесты поделены на блоки по наборам данных. Если ваше решение находит точку с **достаточным** расстоянием (не более 1 светового года), вы получаете 25% за блок. Гарантируется, что это возможно для всех блоков. Для оставшихся 75% необходимо найти точку с **минимальным** расстоянием, которое будет **не больше** авторского, чем на 5%.

Задача проверяет навык работы со сложными вычислительными моделями, что является частью финальной задачи.

Решение

Необходимо найти точку кривой, минимально отдалённую от начала координат. Формула достаточно сложная для аналитического решения, поэтому минимум реальнее всего искать численным путём. В авторском решении предлагается последовательный перебор значений с правильно подобранным шагом (чтобы получить достаточную точность, уложившись в ограничение по времени). Но также возможны решения, связанные с численной минимизацией. При решении задачи (особенно на Python) присутствует элемент низкоуровневой оптимизации. В общем случае необходимо перейти от символьных индексов к числовым. Также в Python для расчётов лучше использовать библиотеку `numpy`.

Пример программы-решения

Ниже представлено решение на языке Python 3.

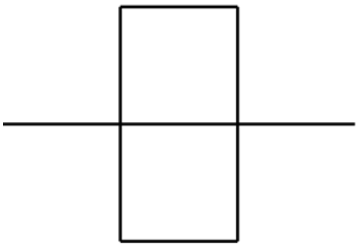
```
import numpy as np

HOUR = 3600

def dist(x):
    return (x[0]**2+x[1]**2+x[2]**2+x[3]**2) ** 0.5

def calc_n(cs):
    sin, cos = np.sin, np.cos
    def fn(t):
        return (
            cs[0 ]*cos(cs[1 ]*t+cs[2 ]) + cs[3 ]*sin(cs[4 ]*t+cs[5 ]) +
              cs[6 ]*cos(cs[7 ]*t+cs[8 ]),
            cs[9 ]*sin(cs[10]*t+cs[11]) + cs[12]*cos(cs[13]*t+cs[14]) +
              cs[15]*sin(cs[16]*t+cs[17]),
            cs[18]*sin(cs[19]*t+cs[20]) + cs[21]*cos(cs[22]*t+cs[23]) +
              cs[24]*cos(cs[25]*t+cs[26]),
            cs[27]*cos(cs[28]*t+cs[29]) + cs[30]*sin(cs[31]*t+cs[32]) +
              cs[0 ]*sin(cs[1 ]*t+cs[2 ]),
        )
    return fn

SEGN = 4
SEG = int(HOUR / SEGN)

def numpy_calc(cs, RES=1000):
    fn_n = calc_n(cs)
    mv, mt = None, None
    for i in range(SEGN):
        t = np.linspace(SEG*i, SEG*(i+1), SEG*RES)
        c = dist(fn_n(t))
        mi = np.argmin(c)
        if mv is None:
            mv, mt = c[mi], t[mi]
        elif mv > c[mi]:
            mv, mt = c[mi], t[mi]
    return mt

cs = [float(x) for x in input().split()]
ans = numpy_calc(cs, RES=500)
print(ans)
```

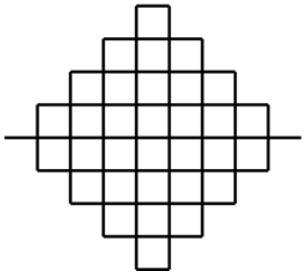
Задача IV.2. Тоже арифметика Пеано (19 баллов)

Темы: работа с моделью, программирование.

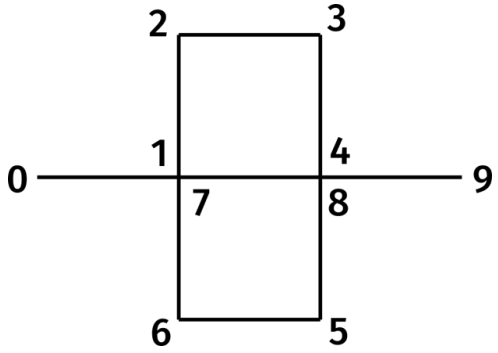
Условие

Интересное свойство кривых Пеано в том, что это линии, которые замощают собой фрагмент плоскости, что делает возможным каждой точке фрагмента плоскости сопоставить хотя бы одну точку прямой (во всяком случае с любой наперёд заданной точностью), и таким образом сопоставить точкам декартовой плоскости точки на числовой прямой.





На нашей кривой Пеано мы нумеруем точки так, как показано на рисунке.



Поскольку кривая Пеано **итерационна**, то мы можем любому узлу кривой сопоставить девятеричную дробь.

Вам даются две **девятеричные** дроби, которым соответствуют некоторые точки на кривой Пеано. Эти точки также расположены на плоскости. Нужно найти **скалярное** произведение векторов, задаваемых этими точками, если известно, что первая итерация нашей кривой Пеано — **единичный** отрезок на оси X.

Формат входных данных

Две строки, в каждой — девятиричная дробь. Например,

```
0,123456788
0,213124123
```

Формат выходных данных

Единственное вещественное **десятичное** число, скалярное произведение радиус-векторов. Ответ считается корректным, если отличается от авторского не более, чем

на 10^{-8}.

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

Решение

Задача требует декодирование девятеричной дроби в вектор. Для этого нужно обратить внимание на то, что цифры девятеричной дроби соответствуют не только точкам, но и отрезкам рекурсивно построенных кривых Пеано. Так, дробь 0,16 соответствует точке 6 кривой Пеано, построенной на отрезке 1–2 кривой Пеано, построенной на единичном отрезке. Таким образом, мы можем написать итерационную функцию, которая берёт соответствующий подотрезок кривой, отталкиваясь от представленного отрезка и выполнить последовательное взятие отрезков от первой цифры дроби до последней, и в конце взять начальную точку отрезка как результат. Скалярное произведение вычисляется тривиально.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import decimal
from decimal import Decimal as de

BASE_PEANO = ((de(0), de(0)), (de(1), de(0)))
THIRD = de(1) / 3

def ptint(a, b, v):
    return tuple(a[i] + (b[i] - a[i]) * v for i in (0, 1))

def subdivide(sg, digit):
    a, b = sg
    pt = ptint(a, b, THIRD)
    d = tuple(pt[i] - a[i] for i in (0, 1))

    if digit == '0':
        p1 = a
        p2 = pt
        return (p1, p2)
    if digit == '1':
        p1 = pt
        p2 = (pt[0] - d[1], pt[1] + d[0])
        return (p1, p2)
    if digit == '2':
        p1 = (pt[0] - d[1], pt[1] + d[0])
        p2 = (p1[0] + d[0], p1[1] + d[1])
        return (p1, p2)
    if digit == '3':
        p2 = ptint(a, b, 2*THIRD)
        p1 = (p2[0] - d[1], p2[1] + d[0])
        return (p1, p2)
    if digit == '4':
        p1 = ptint(a, b, 2*THIRD)
        p2 = (p1[0] + d[1], p1[1] - d[0])
        return (p1, p2)
    if digit == '5':
```

```
            p2 = (pt[0] + d[1], pt[1] - d[0])
            p1 = (p2[0] + d[0], p2[1] + d[1])
            return (p1, p2)
        if digit == '6':
            p2 = pt
            p1 = (pt[0] + d[1], pt[1] - d[0])
            return (p1, p2)
        if digit == '7':
            p1 = pt
            p2 = ptint(a, b, 2 * THIRD)
            return (p1, p2)
        if digit == '8':
            p1 = ptint(a, b, 2 * THIRD)
            p2 = b
            return (p1, p2)

    def unpeano(x):
        x = x.strip()[2:]
        sg = BASE_PEANO
        for c in x:
            sg = subdivide(sg, c)
        return sg[0]

    p1 = unpeano(input())
    p2 = unpeano(input())
    print(p1[0]*p2[0] + p1[1]*p2[1])
```

Задача IV.3. «Зделай винрар» (36 баллов)

Темы: каналы связи, программирование.

Условие

«Чем меньше передаёшь — тем меньше шанс допустить ошибку», — таким принципом руководствуется радиолюбитель Женя, придумывая очередной способ сжатия данных. Он передаёт данные через голосовой канал, кодируя их в последовательность из букв **13-тисимвольного** алфавита «thequickbrown». Например, «ntkcntrrthbowiictncniuebrckwkcteehtin». В таких данных легко допустить ошибку, поэтому нужно сократить объём передаваемой информации путём **сжатия**.

Ваша задача: написать две функции. Первая функция — `pack`, которая преобразует строку в сжатую. Вторая функция — `unpack`, которая должна восстановить сжатую строку в **идентичную** исходной. Заметьте, сжатая строка должна состоять из символов **того же алфавита**, что и исходная. Использование иных символов повлечёт **ошибку формата**.

Сигнатуры функций будут приведены в поле ввода решения после выбора нужного вам языка. Ваши функции **не должны** выводить что-либо в стандартные потоки вывода и ошибок, а также использовать статические или глобальные переменные. 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**.

Задача оценивается исходя из числа пройденных тестов, поделённых на блоки по наборам данных. В каждом блоке содержится по три теста, отличающихся критерием по **степени сжатия** (соотношению длины сжатого сообщения к длине ис-

ходного). Если ваше решение **успешно** выполняет архивацию и разархивацию, при этом длина сжатого сообщения **не более** 95%/90%/80% от исходного, вы получаете соответственно 1/3, 2/3 и 3/3 от стоимости блока. **Гарантируется**, что для всех представленных в задаче входных данных возможно сжатие на полный балл.

Эта задача проверяет навык работы с кодированными данными и протоколами, что является частью финальной задачи.

Решение

Ограничение, связанное с передачей через канал только в 13-ричной кодировке не позволяет напрямую использовать обычные алгоритмы сжатия на основе двоичных бит. В теории, можно реализовать сжатие за счёт кодирования повторений или по методу Хаффмана, но авторское решение предлагает более нестандартный метод, доступный на Python: сжать текст стандартной библиотекой `zlib`, после чего перекодировать блоки двоичных бит в 13-ричные «числа». Замечено, что 11-тибитные двоичные блоки успешно кодируются в 3-хсимвольные блоки алфавита, обеспечивая достаточную эффективность ($2^{11} = 2048$, $13^3 = 2197$, т. е. для кодирования используется 93,21% от всех 2197 чисел). Для соблюдения размерности дописываются недостающие до 13 нули, и их число «пишется» в начале передаваемых данных. Таким образом достигается уровень сжатия до 75% на предложенных входных наборах данных.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import zlib

# константы
alphabet = "thequickbrown"
alen = len(alphabet)
EBLL = 11
DBLL = 3

def chunks(x, n):
    for i in range(0, len(x), n):
        yield x[i:i+n]

def n_encode(x):
    x = int(x, 2)
    a = x % alen
    b = (x // alen) % alen
    c = x // (alen * alen)
    return (alphabet[a] + alphabet[b] + alphabet[c])

def n_decode(x):
    a = alphabet.index(x[0])
    b = alphabet.index(x[1])
    c = alphabet.index(x[2])
    v = a + b * alen + c * (alen * alen)
    vt = bin(v)[2:].zfill(EBLL)
    return vt

def pack(txt):
```

```
    r = "".join(bin(c)[2:].zfill(8) for c in zlib.compress(txt.encode()))
    ans = []
    align = 0
    for ch in chunks(r, EBLL):
        if len(ch) != EBLL:
            align = EBLL - len(ch)
            ch = ch + ('0' * align)
        ans.append(n_encode(ch))
    ans.insert(0, n_encode(bin(align)[2:].zfill(EBLL)))
    return "".join(ans)

def unpack(txt):
    chs = chunks(txt, DBLL)
    align = int(n_decode(next(chs)),2)
    ans = []
    for ch in chs:
        ans.append(n_decode(ch))
    if align:
        ans[-1] = ans[-1][:-align]
    raw = bytes(int(x, 2) for x in chunks("".join(ans), 8))
    r = zlib.decompress(raw)
    return r.decode()
```

Задача IV.4. Переводчик с двоичного II: Двойная угроза (10 баллов)

Темы: кодирование-декодирование, анализ кода.

Условие

Эта задача является продолжением задачи II.3.2 Инженерного тура. Если вы её решали, пропустите следующие два абзаца, это повтор условия для контекста.

Толя устроился на работу сисадмином на складе, где установлена СКУД (система контроля и управления доступом), работающая по RFID-меткам. Прошлый администратор оставил систему в неприглядном виде. В панели администрирования у каждой метки Толя увидел только два параметра: **шестнадцатиричный** идентификатор из 8 символов и надпись «**EM4100**».

Толя решился на эксперимент. Придя пораньше на работу вместе с кладовщиком Стёпой, он попытался выяснить, какая конкретно метка у Стёпы на руках. И тут обнаружилось, что декодер считывателя не работает, а новый считыватель приедет только через две недели. Начальник же требует, чтобы всё заработало вчера. Кладовщик подносит метку к считывателю, и вместо идентификатора считыватель выдаёт некую **двоичную** последовательность из 128 бит.

Известно, что ошибок при **чтении** считываемых данных не возникает, но сами метки долгое время лежали на магните с надписью «Для чистки дискет». Из-за этого зашитые в них данные могут содержать ошибки в виде искажения (т. е. **инверсии**) битов. Будем считать, что стартовый и конечные биты не повреждены, в пакете может возникнуть до двух ошибок, и при возможности коррекции данные восстанавливаются однозначным образом.

Напишите программу, которая определяет **версию** и **идентификатор** метки. Если это невозможно, программа должна вывести «ERROR».

Формат входных данных

Строка из 128 нулей и единиц.

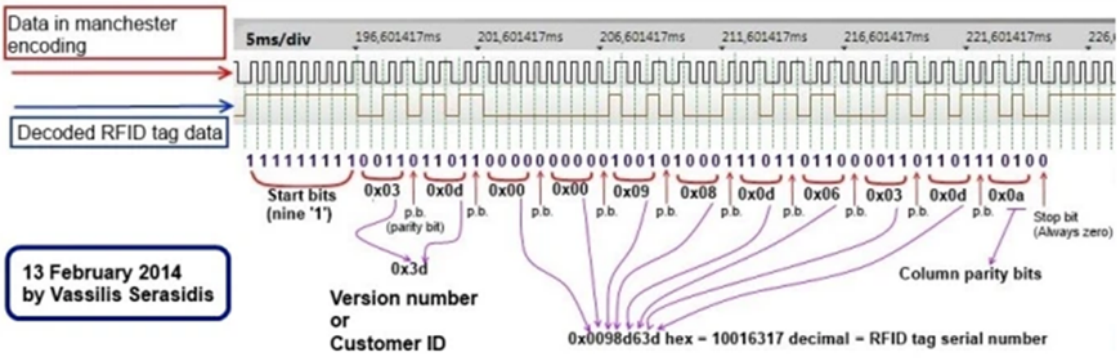
Формат выходных данных

Строка из **10** шестнадцатеричных цифр. Если код восстановить невозможно, вывести «ERROR».

Эта задача учит работе с двоичными и корректирующими кодами, что является частью финальной задачи.

Решение

Код представляет собой «сырой» двоичный сигнал с метки. Стандарт EM4100 предполагает передачу 64 бит данных с помощью амплитудной модуляции с манчестерским кодированием: если в течение такта происходит смена сигнала с низкого на высокий, это считается единицей, с высокого на низкий — нулём. Таким образом, из «сырого» кода образуется 64-битный пакет, в котором первые 9 бит — стартовые единицы, после идут 10 пятибитных блоков, состоящих из 4 бит данных и бита чётности, после 4 бита чётности столбцов и последний стоп-бит (всегда нуль). Первые два блока соответствуют номеру версии, остальные восемь — искомый идентификатор метки.



Для получения данных, как и в первой задаче (см. 1 этап),Так как ошибок в коде не предусмотрено, достаточно разбить строку на пары, заменить «01» на «1», а «10» на «0». Далее необходимо проверить блоки и столбцы на чётность (т. е. просуммировать биты по модулю 2 и сравнить с нулём).

Следует заметить, что восстановление пакета возможно только в том случае, когда несоответствия чётности однозначно указывают на искажённый бит данных. Таким образом, пакет подлежит восстановлению, только если имеется несовпадение в одном блоке-строке и одном столбце. В таком случае достаточно инвертировать бит в пересечении блока и столбца. При иных нечётностях пакет восстановлению не подлежит.

После проверки, отсечки невосстановимых пакетов и коррекции остаётся извлечь необходимые тетрады бит и преобразовать их в шестнадцатеричную систему.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
data = input()
unman = ["1" if data[i:i+2] == "01" else "0" for i in range(0, 128, 2)]
rows = [unman[9+i*5:14+i*5] for i in range(10)]
cols = list(zip(*(unman[9+i*5:13+i*5] for i in range(11))))
row_p = [sum(map(int, x)) % 2 for x in rows]
col_p = [sum(map(int, x)) % 2 for x in cols]
rp, cp = sum(row_p), sum(col_p)
if not ((rp == 0 and cp == 0) or (rp == 1 and cp == 1)):
    print("ERROR")
    exit()
if (rp == 1 and cp == 1):
    ri = row_p.index(1)
    ci = col_p.index(1)
    i = 9+5*ri+ci
    unman[i] = "1" if (unman[i] == "0") else "0"
unman = "".join(unman)
data = "".join(hex(int(unman[9+i*5:13+i*5],2))[2:] for i in range(10))
print(data)
```

Задача IV.5. Большая корректура больших данных (15 баллов)

Темы: анализ данных, программирование.

Условие

Современное машинное обучение полагается на большие массивы данных, среди которых есть и старые добрые **таблицы**. К сожалению (или счастью), машины не научились заполнять их за людей, а люди очень ненадёжны, и такие таблицы содержат массу **ошибок**, из-за которых моделям грустно и некомфортно учиться.

Искать эти ошибки вручную не менее грустно, поэтому мы предлагаем вам написать **программу**, которая ищет ошибки в таблице формата **Neverbroken Tabular Information** (NTI). Этот формат близок к CSV, таблица представляет собой набор строк с ячейками, разделяемыми символом табуляции (`\t`). В первой строке приводятся **заголовки** полей таблицы. Заголовок может содержать любые буквенные, числовые символы, пробел и подчёркивание.

Во второй строке — **типы** данных, а в следующих строках — **данные**, им соответствующие:

`number`	Вещественное число с десятичным разделителем — `точкой`
`text`	Строка (обязательно в **двойных** кавычках)
`natural`	**Натуральное** число
`char`	Одиночный **печатный** символ или пробел
`snils`	Строка вида ###-###-###-##, где # — это цифра
`phone`	Строка вида +7### ###-##-##, где # — это цифра
`time`	Время в формате ЧЧ:ММ (24-часовой)
`date`	Дата в формате ДД.ММ.ГГГГ

Вам предстоит искать ошибки в данных и выводить их описание в качестве ответа. Среди ошибок:

`Bad header name`	Некорректный символ в заголовке (указывает на этот символ)
`Empty header name`	Пустой заголовок (указывает на «начало» ячейки с этим заголовком)
`Bad type`	Некорректный или пустой тип данных (указывает на начало некорректного типа, соответствующее поле далее не проверяется)
`Bad data`	Некорректное значение (указывает на начало некорректной ячейки)
`Line overflow`	Лишние ячейки в строке (указывает на первую лишнюю табуляцию)
`Line underflow`	Недостаточно ячеек (указывает на символ переноса строки)

Ошибка выводится в формате: `Line` A: `Char` B: `<Error text>`, где A — номер строки в тексте (начиная с 0), B — номер символа в строке (начиная с 0). Например, `Line 0: Char 0: Bad data`. Если ошибок нет, программа должна вывести `OK`.

Число столбцов определяется **первой строкой**. По краям значений, разделённых табуляциями, могут стоять пробелы, при чтении значений они **отсекаются** (кроме типа `char`). Если поле по той или иной причине не имеет типа, его значения **не проверяются**. Гарантируется, что первая и вторая строки имеют **одинаковое** число ячеек, а в значениях **не встречаются** символы переноса и табуляции.

Формат входных данных

Произвольный текст, таблица в формате NTI. Гарантируется отсутствие непечатных символов, кроме пробелов, табуляций и переносов строк. Пример корректной таблицы (с выделенными символами табуляции):

```
name\tage\tchara
text\tnatural\tchar
"Anna"\t24\t!
"Bob"\t14\t+
"Charles"\t32\tP
```

Формат выходных данных

Произвольное число строк, в каждой — описание найденной ошибки. Вывод ошибок производится в порядке **увеличения** координат (строка-столбец). Если ошибок нет, выводится `OK`.

Эта задача проверяет ваш навык работы с форматированными текстовыми данными, что потребуется в работе над финальной задачей.

Решение

Задача требует аккуратную и внимательную реализацию всех перечисленных в условии проверок. Для проверки текстовых значений проще всего использовать регулярные выражения (библиотека `re` в Python). Алгоритм проверки даты с учётом

високосных лет тривиален и находится в открытом доступе. Особое внимание необходимо обратить на правильный расчёт координаты выявленной ошибки, она должна соответствовать месту в исходных входных данных. При использовании библиотечных функций разбиения на подстроки (`split`) возможна отсечка пробелов в подстроках, из-за чего координаты искажаются, и это являлось наиболее частой ошибкой в решении задачи.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
import re
from ast import literal_eval
import string
import sys

def make_err(v):
    x, y, err = v
    return f"Line {x}: Char {y}: {err}"

HEADER_ALPHABET = string.ascii_letters + string.digits + "_ "

NO_HEADER = "Empty header name"
BAD_HEADER = "Bad header name"
BAD_TYPE = "Bad type"
BAD_DATA = "Bad data"
OVERFLOW = "Line overflow"
UNDERFLOW = "Line underflow"
NO_ERRORS = "OK"

snils_re = re.compile("\d{3}-\d{3}-\d{3}-\d{2}")
phone_re = re.compile("\+\d{4} \d{3}-\d{2}-\d{2}")
time_re = re.compile("([01]?[0-9]|2[0-3]):[0-5][0-9]")
date_re = re.compile("(0?[1-9]|[12]\d|3[01])\.(0?[1-9]|1[0-2])\.(19|20)\d{2}")
date_months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

def ignore_me(x):
    return True

def verify_header(x):
    ans = []
    for i, c in enumerate(x):
        if c not in HEADER_ALPHABET:
            ans.append(i)
    return ans

def try_eval(x):
    try:
        t = literal_eval(x)
        return t
    except Exception:
        pass

def number_check(x):
    x = try_eval(x.strip())
    return isinstance(x, float)

def char_check(x):
```

```
    return len(x) == 1

def snils_check(x):
    return bool(snils_re.fullmatch(x.strip()))

def phone_check(x):
    return bool(phone_re.fullmatch(x.strip()))

def time_check(x):
    return bool(time_re.fullmatch(x.strip()))

def date_check(x):
    x = x.strip()
    if date_re.fullmatch(x.strip()):
        d, m, y = int(x[0:2]), int(x[3:5]), int(x[6:10])
        if m == 2 and (y % 400 == 0 or (y % 100 != 0 and y % 4 == 0)):
            if d > 29:
                return False
        elif d > date_months[m-1]:
            return False
        return True

def str_check(x):
    x = x.strip()
    if x[0] != '"' or x[-1] != '"':
        return False
    x = try_eval(x)
    return isinstance(x, str)

def natural_check(x):
    x = try_eval(x.strip())
    if isinstance(x, int):
        return x > 0
    return False

def eval_check(vals):
    def fn(x):
        x = try_eval(x.strip())
        if not isinstance(x, (int, str)):
            return False
        return (x in vals)
    return fn

TYPES = {
    "number": number_check, # Вещественное число с разделителем-точкой
    "text": str_check, # Строка (обязательно в двойных кавычках)
    "natural": natural_check, # Натуральное число
    "char": char_check, # Одиночный символ
    "snils": snils_check, # Строка вида ###-###-###-##, где # - это цифра
    "phone": phone_check, # Строка вида +7### ###-##-##, где # - это цифра
    "time": time_check, # Время в формате ЧЧ:ММ (24-часовой)
    "date": date_check, # Дата в формате ДД.ММ.ГГГГ
}

def verify_type(x):
    if x in TYPES:
        return TYPES[x]

def solve(data):
    errs = []
```

```
    data = data.splitlines()

    i = 0
    line = data[0]
    col_n = 0
    while i < len(line):
        j = line.find("\t", i)
        if j == -1:
            j = len(line)
        header = line[i:j].strip()
        col_n += 1
        if len(header) == 0:
            errs.append((0, i, NO_HEADER))
        else:
            x = verify_header(line[i:j])
            for lv in x:
                errs.append((0, i+lv, BAD_HEADER))
        i = j + 1

    checks = []
    i = 0
    line = data[1]
    while i < len(line):
        j = line.find("\t", i)
        if j == -1:
            j = len(line)
        tp = line[i:j].strip()
        ch = verify_type(tp)
        if ch is not None:
            checks.append(ch)
        else:
            errs.append((1, i, BAD_TYPE))
            checks.append(ignore_me)
        i = j + 1

    for line_n in range(2, len(data)):
        line = data[line_n]
        i = 0
        val_n = 0
        while i < len(line):
            if val_n == col_n:
                errs.append((line_n, i-1, OVERFLOW))
                break
            j = line.find("\t", i)
            if j == -1:
                j = len(line)
            va = line[i:j]
            if not checks[val_n](line[i:j]):
                errs.append((line_n, i, BAD_DATA))
            val_n += 1
            i = j + 1
        if val_n != col_n:
            errs.append((line_n, len(line), UNDERFLOW))

    if not errs:
        return NO_ERRORS

    return "\n".join(map(make_err, errs))

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

Задача IV.6. Микро-консенсус (8 баллов)

Темы: алгоритмы, работа с моделью.

Условие

Беспроводная сеть в офисе «Берляндия Телеком» построена на основе экспериментальной **mesh-системы**. Настройка её узлов производится с помощью конфигурационного файла. После настройки узла в **конфигурационный** файл заносится **метка**, отмечающая готовность данного узла к работе. В конце, головной узел проверяет все метки и запускает сеть. Следовательно, для запуска сети конфигурационный файл должен пройти **все** узлы и **вернуться** на головной.

Головной узел уже **настроен**, настройка каждого узла длится 2,14 секунд, задержек между принятием, настройкой и началом отправки конфиг-файла **нет**.

Зная задержки передачи, определите **наименьшее** время настройки всей сети.

Формат входных данных

В первой строке число N — количество узлов в системе (до 10). Далее N строк, в каждой N **целых** чисел — задержки (в мс) передачи файла от устройства A (номер строки) устройству B (номер столбца). Головное устройство имеет номер 0. Значения на главной диагонали всегда равны -1. Пример формата:

```
4
0 69 75 57
57 0 48 54
75 30 0 30
27 12 39 0
```

Формат выходных данных

Единственное число, наименьшее время настройки всей сети **в миллисекундах**. Ответ считается корректным, если отличается от авторского не более, чем на 10 мс.

Эта задача проверяет ваш навык работы с моделью и алгоритмики, что потребуется в работе над финальной задачей.

Решение

В общем виде эта задача представляет собой задачу коммивояжёра, цель которой — поиск минимального эйлерового цикла — замкнутого пути, проходящего по всем узлам сети ровно по одному разу. Способов её решения достаточно много, различных по вычислительной сложности. В качестве авторского решения выбран метод ветвей и границ.

Но учитывая ограничение в 10 элементов и ограничение в 3 секунды, возможно более тривиальное решение в виде перебора $(10-1)/2! = 181440$ перестановок и вычисления для каждой из них задержек передачи.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
from heapq import heappush, heappop
from copy import deepcopy

T_NODE = 2140
INF = 1e20

class Node:
    path = None
    reducedMatrix = None
    cost = 0
    vertex = 0
    level = 0
    def __init__(self):
        self.path = []
        self.reducedMatrix = None
    def __lt__(self, other):
        return self.cost < other.cost
    def __gt__(self, other):
        return self.cost > other.cost

def newNode(parentMatrix, path, level, i, j) -> Node:
    N = len(parentMatrix)
    node = Node()
    node.path = path.copy()
    if level != 0:
        node.path.append((i, j))
    node.reducedMatrix = deepcopy(parentMatrix)
    k = 0
    if level != 0:
        for k in range(N):
            node.reducedMatrix[i][k] = INF
            node.reducedMatrix[k][j] = INF
    node.reducedMatrix[j][0] = INF
    node.level = level
    node.vertex = j
    return node

def rowReduction(reducedMatrix):
    N = len(reducedMatrix)
    row = [INF for _ in range(N)]
    for i in range(N):
        for j in range(N):
            if reducedMatrix[i][j] < row[i]:
                row[i] = reducedMatrix[i][j]
    for i in range(N):
        for j in range(N):
            if reducedMatrix[i][j] != INF and row[i] != INF:
                reducedMatrix[i][j] -= row[i]
    return row

def columnReduction(reducedMatrix):
    N = len(reducedMatrix)
    col = [INF for _ in range(N)]
    for i in range(N):
        for j in range(N):
            if reducedMatrix[i][j] < col[j]:
```

```
                col[j] = reducedMatrix[i][j]
    for i in range(N):
        for j in range(N):
            if reducedMatrix[i][j] != INF and col[j] != INF:
                reducedMatrix[i][j] -= col[j]
    return col

def calculateCost(reducedMatrix) -> int:
    row = rowReduction(reducedMatrix)
    col = columnReduction(reducedMatrix)
    return (
        sum(x for x in row if x != INF) +
        sum(x for x in col if x != INF)
    )

def solveTSP(CostGraphMatrix):
    N = len(CostGraphMatrix)
    pq = []
    v = []
    root = newNode(CostGraphMatrix, v, 0, -1, 0)
    root.cost = calculateCost(root.reducedMatrix)
    pq.append(root)
    while pq:
        mi = heappop(pq)
        i = mi.vertex
        if mi.level == N-1:
            mi.path.append((i, 0))
            return mi.cost

        for j in range(N):
            if mi.reducedMatrix[i][j] != INF:
                child = newNode(mi.reducedMatrix, mi.path, mi.level + 1, i, j)
                child.cost = mi.cost + mi.reducedMatrix[i][j] +
                ↪  calculateCost(child.reducedMatrix)
                heappush(pq, child)
    return 0

n = int(input())
matrix = [
    [INF if x == "-1" else int(x) for x in input().split()]
    for _ in range(n)
]
tsp = solveTSP(matrix)
print(T_NODE * (n-1) + tsp)
```