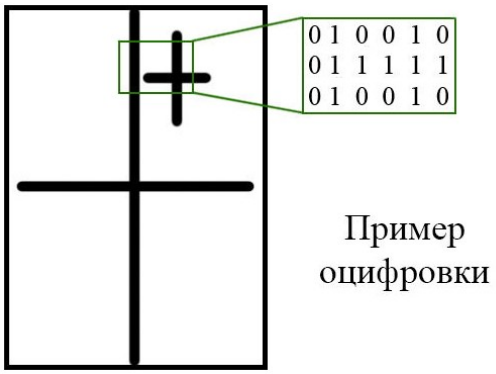
Второй отборочный этап

Индивидуальный модуль

Задача И1. Счет для уверенны (2,1 баллов)

В школе проходило психологическое тестирование. В рамках этого тестирования школьники отвечали на вопросы, выставляя в **произвольных местах** отдельного бланка **плюсы**, по одному на каждый положительный ответ.

После тестирования бланки оцифровываются и загружаются в компьютер, где каждый бланк представлен в виде **прямоугольной матрицы из нулей и единиц**, где единица означает **наличие** линии, а ноль — ее **отсутствие**.



Ваша задача: написать **программу**, сканирующую такую матрицу и подсчитывающую **количество** выставленных в ней плюсов. В рамках этой задачи условимся, что все плюсы **ровные** и их линии **параллельны** границам бланка, а их **толщина** ограничена одной единицей. При этом плюсы могут **касаться** друг друга, и их размеры **произвольные**, но для отдельного плюса **совпадают** во всех направлениях. В рамках этой задачи матрица **не содержит** иных фигур и шумов.

Входной формат: прямоугольная матрица размером 200 столбцов на 300 строк из нулей и единиц. Матрица будет передана в программу в следующем виде (строки слитные и разделены переносом):

010101010010101
100101010101010
101001010101001
101010101001010

Выходной формат: единственное целое число, количество плюсов в матрице.

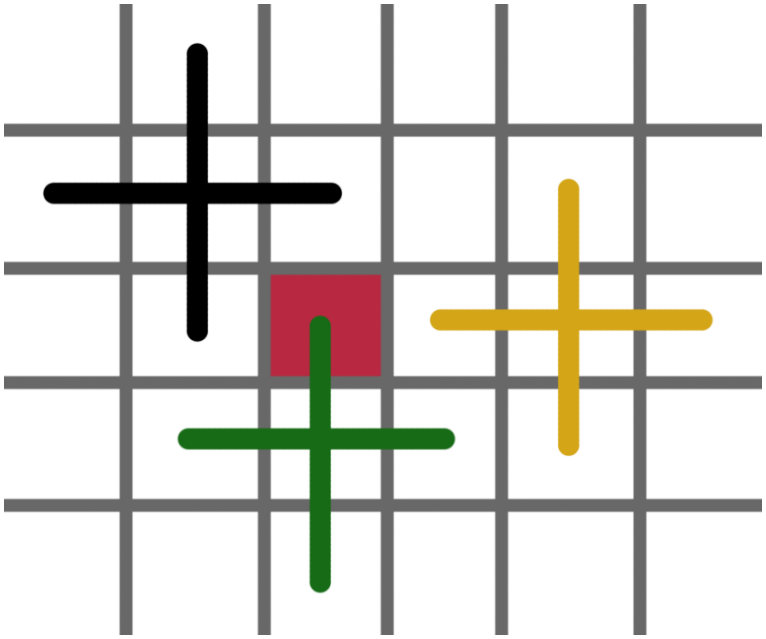
Ограничения работы программы: 2 секунды, 256 Мб.

Для решения этой задачи дается 15 попыток.

За написание программы, которая будет выдавать верный ответ, участник получает максимальный балл.

Алгоритм решения

1. Считываем данные в массив.
2. Перебираем ячейки массива, на предмет наличия единицы, являющейся центром плюса (когда слева, сверху, справа, снизу от нее также находятся единицы).
3. Считаем длину плюса.
4. Увеличиваем счетчик плюсов и удаляем найденный плюс, так как в дальнейшем могут быть сгенерированы ситуации, когда соприкасающиеся плюсы могут дать неверный центр.



Пример программы-решения

Ниже представлено решение на языке Python3.

```
buffer = [[int(c) for c in l] for l in data.splitlines(False)]
        n = 0 #счетчик
        h, w = len(buffer), len(buffer[0]) #высота и ширина массива
        for i in range(1, h-1):
                for j in range(1, w-1):
                if (buffer[i][j] and buffer[i][j+1] and buffer[i][j-1] and
                        buffer[i+1][j] and buffer[i-1][j]):
                        n += 1
                        buffer[i][j] = False
                        x = 1
                        while (0 <= i-x and i+x < h and
                               0 <= j-x and j+x < w and
                                buffer[i][j+x] and buffer[i][j-x] and
                                buffer[i+x][j] and buffer[i-x][j]):
                        buffer[i][j+x] = False
```

```
16                      buffer[i][j-x] = False
17                      buffer[i+x][j] = False
18                      buffer[i-x][j] = False
19                      x += 1
```

Задача И2. Воздушная передача (2,1 баллов)

Подзадача 1. Группа студентов собрала мини-ракету по типу CanSat со встроенным радиопередатчиком. Для оценки помехоустойчивости канала между ракетой и их приемником на земле, студенты решили использовать кодирование данных методом **контрольных сумм**.

Датчик непрерывно передает **произвольные строки** данных каждую **миллисекунду**. Строки состоят целиком из **цифр** и кодируются так, чтобы их **сумма** в строке была **равна** 16.

Вам предлагается набор пакетов, полученных с датчика. Необходимо посчитать **количество ошибочных пакетов** в этом наборе.

Входной формат: 100 **строк**, содержащих отдельные пакеты данных в виде последовательности цифр длиной до 16. Например,

808
22131194
702133
2005000000000009
38113

Выходной формат: **целое** число, **количество ошибок** в перечисленных пакетах.

За выполнение данной задачи участник получает 2/7 баллов от максимального количества баллов.

Подзадача 2. В боевых условиях данных значительно больше. Напишите **программу**, которая считает **количество ошибочных строк** в пределах каждого интервала **в одну десятую секунды**.

Входной формат: произвольное кратное 100 (до 800) число **строк**, содержащих отдельные пакеты данных в виде последовательности цифр длиной до 16. См. пример в подзадаче 1.

Выходной формат: по одной строке на каждые 100 входных строк, в которой записано время вывода статистики, затем через символ табуляции число ошибок за последние 100 пакетов. Например,

$t = 0.1\ N = 39$
$t = 0.2\ N = 49$

Ограничения работы программы: 1 секунда, 256 Мб.

Для решения этой задачи дается 15 попыток.

За выполнение данной задачи участник получает 5/7 баллов от максимального

количества баллов.

Алгоритм решения

1. Считываем данные.
2. Вводим внешний и внутренний счетчики.
3. Подсчитываем сумму цифр числа, если она не равна 16, то увеличиваем внешний счетчик.
4. Каждые 100 строк выводим ответ по указанному формату.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
summ_out = 0
msec = 0
time = 1
for line in sys.stdin:
        msec += 1
        if sum([int(j) for j in line.rstrip()]) != 16:
                summ_out += 1
        if msec == 100:
                print('t = ' + str(0.1 * time)[0:3] + '\t' + 'N = ' + str(summ_out))
                time += 1
                msec = 0
                 summ_out = 0
```

Задача И3. Умный код (4,95 баллов)

Несколько студентов-программистов решили создать свой мини-аналог Интернета вещей. Для этого им *пришлось* придумывать **индивидуальные двоичные коды** для каждого объекта в комнате.

Дело в том, что пронумеровав все предметы обычным способом, студенты столкнулись с проблемой. Их прибор для считывания кода начал вносить незначительные **помехи**, в результате которых **одна** из цифр в коде могла считаться неверно.

Вам нужно **придумать** такие **двоичные** коды для всех объектов, чтобы при **единичной** ошибке они продолжали считываться однозначно, и даже с помехами книга продолжала определяться, как книга. Для получения **максимального балла** за задачу, кодировка данных должна быть как можно **менее избыточной**, то есть использовать наименьшее число бит.

Входной формат: количество объектов в комнате N, натуральное число от 20 до 50.

Выходной формат: N строк одинаковой длины, в каждой из которых записан двоичный код объекта.

За решение данной задачи, при котором расстояние Хэмминга между всеми строками не менее 3, однако избыточность строк слишком велика, участник получает 10/33 от максимального количества баллов.

За решение данной задачи с минимальной избыточность, при котором расстояние Хэмминга между всеми строками не менее 3, участник получает максимальный балл.

Алгоритм решения

1. Определяем количество бит, необходимых для кодировки всех N чисел ($>$ или < 32).
2. Начинаем перебор всех чисел, вплоть до N в двоичном формате.
3. Кодируем двоичные числа с использованием кода Хэмминга с подходящими параметрами.

Задача И4. Полуавтомат (2,1 баллов)

Команда программистов разрабатывает прототип автопилота, который должен управлять машиной на местности с препятствиями и обходить их с **минимальными затратами**. Автопилот работает на основе изображений с видеорегистратора. Получая оцифрованное изображение, модуль машинного зрения распознает **преграды**, обозначая их как единицы в строке данных, а места с **возможностью проезда** — как нули.

Для проверки автопилота на вход подали ряд **отдельных** тестовых изображений, которые в данной задаче уже обработаны машинным зрением и представлены как **строки из нулей и единиц**. Далее алгоритм для каждого из предложенных снимков должен сформировать управляющую команду, обеспечивающую **оптимальный** объезд препятствия, то есть с **наименьшим** отклонением от курса. Управляющая команда содержит **дельту расстояния** от оси машины до свободного пути, причем движение влево кодируется отрицательным числом, по модулю равным необходимому расстоянию. То есть положительное число кодирует движение вправо, а ноль — сохранение исходного курса.

Габариты машины относительно снимка равны **трем** ячейкам, то есть для объезда препятствий она должна попасть в **центр** окна из трех нулей. При возможности **одинаково** оптимального объезде и справа, и слева, предпочтение отдается объезду **слева**.

Ваша задача: реализовать **алгоритм** объезда, формирующий необходимые управляющие команды. Гарантируется, что для каждого из снимков объезд **возможен**.

Входной формат: произвольное (до 150) число строк, содержащих **отдельные** тестовые снимки в виде двоичного кода из 9 разрядов. Например,

111100000
110000000
101010000
001011000
100010001

Выходной формат: такое же число строк, содержащих управляющие команды в описанном выше формате. Например,

-3
2
3
0
-2
0

Ограничения работы программы: 3 секунды, 256 Мб.

Для решения этой задачи дается 15 попыток.

За написание программы, которая будет выдавать верный ответ, участник получает максимальный балл.

Алгоритм решения

1. Считываем данные.
2. Проверяем центр на условие: слева, справа и в центре 0.
3. Если условие не выполняется, то начинаем перебор поиска ближайшего пути справа и слева от центра с запоминанием.
4. Выводим наименьший по модулю ответ, с приоритетом к повороту влево.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
for x in trade:
        left = 0
        right = 0
        i = 4
        while i > 0 and x[i-1] + x[i] + x[i+1] != "000":
                i -= 1
                left += 1
        i = 4
        while i < len(a) - 1 and a[i-1] + a[i] + a[i+1] != "000":
                i += 1
                right += 1
        if left <= right:
                print(-left)
        else:
                print(right)
```

Задача И5. Сломанный телеграф (3,75 баллов)

Вам в руки попал старый датчик для анализа электрических полей, передающий данные по радиоканалу, и вы решили его проверить. На самом датчике вы заметили надпись «ParityBitSuffix», из чего вы решили, что при отправке данных используется контроль целостности с помощью **контрольного бита четности**.

Для проверки вы прикрепили датчик к щитку электропитания и собрали набор данных. Просмотрев полученный набор, вы видите, что датчик передает отдельные пакеты в виде **непустых ASCII-строк**, где символы представляются **7-**

разрядными двоичными кодами (в порядке от старшего бита к младшему), а в конце строки присоединяется **контрольный** бит. Сама же передача происходит с большим количеством **помех**, на чем сказался возраст датчика. И глядя на полученные данные, вам не удалось понять, какое именно **значение четности** является **корректным** для этого кода.

Для правильной работы с датчиком напишите **программу**, которая будет определять **четность** кода, если известно, что количество помех **не больше** 40%, и двойная ошибка в одной строке **исключена**. Также программа должна выводить **декодированные строки** из ASCII-символов. Если в отдельной строке допущена **ошибка**, то **вся** строка должна быть заменена на символ *.

Входной формат: произвольное (до 500) число строк, содержащих отдельные пакеты в виде двоичного кода длиной до 50 разрядов. Например,

011000111000010
0101110111011110000111010110
111001001011001
011100010101111110110111100011011101101001111110011
11011110
1011010110011010001000110101110101000

Выходной формат: такое же число строк из ASCII-символов, полученных в результате декодирования пакетов. Например (не соответствует входному набору),

q?A)
L/RWa
**
-_82
**
/;

Ограничения работы программы: 1 секунда, 256 Мб.

Для решения этой задачи дается 15 попыток.

За написание программы, которая будет выдавать верный ответ, участник получает максимальный балл.

Алгоритм решения

1. Считываем данные.
2. Проверяем четность во всех строках, для выявления наибольшей.
3. Перебираем строки с проверкой четности.
   3.1. Если строка имеет преобладающую четность, тогда удаляем контрольный бит и переписываем сроку посимвольно.
   3.2. Если строка имеет подавляемую четность, то ставим символ *.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
even = odd = 0
lines = []
for line in sys.stdin:
        line = line.rstrip()
        kolvo_1 = line.count('1')
        if kolvo_1 % 2 == 1:
                even += 1
        else:
                odd += 1
        lines.append(line)
bt = 0 if even < odd else 1
for line in lines:
        if line.count('1') % 2 == bt:
                bin_text = line[:-1]
                print(''.join(chr(int(bin_text[i:i + 7], 2)) for i in range(0,
                ↪  len(bin_text), 7)))
        else:
                print('*')
```

Командный модуль

Основные задачи

Задача К1. Тройная устойчивость (7,73 баллов)

Тестируя новые датчики, ученые установили, что передаваемые датчиками **числовые** данные могут претерпевать искажение **одной** цифры из-за помех в используемых каналах передачи данных. Чтобы **изучить** помехи, при этом продолжая **получать** необходимые данные, ученые реализовали **кодировку** данных с датчиков при помощи **утроения** цифр исходного числа.

Так, исходное число 18 в результате кодировки становится 181818, и во время передачи через канал с помехами может приобретать вид 181918, 381818 и т. д.

Ваша задача: написать программу, которая будет **декодировать** принятые данные, **вести** статистику ошибок и **помечать** ошибочные пакеты символом *.

Входной формат: произвольное число строк до 1000, состоящих из чисел, количество цифр в которых кратно трем. Например,

101010
282928
393
241241271
550550550
999
787070

Выходной формат: первая строка — число строк с ошибками, далее построчно декодированные данные с пометкой * для пакетов, в которых была ошибка. Например,

29
*7**
673
*985**
*594**
*77**
4

Ограничения работы программы: 1 секунда, 256 Мб.

Для решения этой задачи команде дается 20 попыток.

За написание программы, которая будет выдавать верный ответ, участник получает максимальный балл.

Алгоритм решения

1. Считываем данные.
2. Определяем количество цифр закодированного числа n, для этого поделить общее количество цифр в строке на 3.
3. Выделяем первые n цифр в строке и проверяем, если при повторении схемы кодирования, получается исходная строка, то запоминаем эти n цифр.
4. Иначе, мы увеличиваем счетчик ошибок и начинаем проверку, какая пара из n цифр будет совпадать. Совпадающую пару мы также запоминаем.
5. Выводим сначала количество ошибок, а затем построчно все запомненные числа.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
err = 0
ans = []
for s in st:
        N = len(s)
        n = N // 3
        if s[:n] * 3 == s:
                ans.append(s[:n])
        else:
                err += 1
                p = 0
                while p < 3:
                        subs = s[p * n : (p + 1) * n]
                        if s.count(subs) == 2:
                                ans.append(subs + '*')
                                break
                        p += 1
print(err)
```

```
for s in ans:
        print(s)
```

Задача K2. Туннельный синдром (15,45 баллов)

Пользователь со своего рабочего места запросил у домашнего сервера данные через удаленный доступ. Эти данные передаются на рабочее место пользователя по туннелю **блоками** текстовых данных по 36 **символов** в каждом блоке.

Однако канал, по которому работает туннель, **нестабилен**, из-за чего блоки приходят на рабочий компьютер в **случайном** порядке, содержимое блоков при этом остается прежним (спасибо алгоритмам контроля ошибок).

Ваша задача: реализовать **кодировку** для стабильной передачи данных через туннель, а именно написать две функции: **кодировщик (encode)** и **декодировщик (decode)**. Первая функция **генерирует** блоки из входного объекта, вторая **воссоздает** из блоков исходный объект.

Формат реализации: две функции. Функция encode принимает на вход строку произвольной длины до 1048576 символов, и должна выдавать список строк длиной 36 символов каждая. Аналогичный список принимает на вход функция decode, которая должна возвращать восстановленную строку.

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 и глобальные переменные. Мы будем вынуждены не зачесть такие оригинальные решения. Также учитывайте, что в ограничение на время выполнения входит время работы **вспомогательного** кода (не более секунды).

Ограничения работы программы: 5 секунд, 256 Мб.

Для решения этой задачи команде дается 20 попыток.

За написание программы, которая будет правильной осуществлять передачу, участник получает максимальный балл.

Алгоритм решения

1. Принимаем строку в функции encode.
2. Выделяем из этой строки последовательные n байт, чтобы n было меньше 36.
3. В оставшиеся от 36 — n байт вставляем идентификатор блока.
4. Принимаем строки в функции decode.
5. Начинаем перебор строк, для поиска строки с наименьшим идентификатором.
6. После нахождения нужного блока, переписываем его n действительных байт в ответ и затираем саму строку.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
def encode(s: str) -> List[str]:
        encode list = []
```

```
        for i, x in enumerate(range(0, len(s), 32)):
                frag = s[x:x + 32]
                length = chr(len(frag) )
                idx = "".join(chr(x) for x in (i // 65536 % 256, i // 256 % 256, i %
                ↪  256))
                leftover = " " * (32 - len(frag))
                encode_list.append(idx + length + frag + leftover)
        return encode_list

def decode(chunks: List[str]) -> str:
        chunks.sort()
        return "".join(c[4:][:ord(c[3])] for c in chunks)
```

Задача К3. Звуковой беспилотник (19,32 баллов)

Автомобиль с беспилотной системой вождения едет по дороге с **постоянной** скоростью через автоматический пост ДПС. На пункте имеется автономная система слежения за беспилотными автомобилями. По приходящему от них слабому **акустическому** сигналу система слежения определяет **скорость** автомобиля и необходимость штрафа за превышение скорости или недостаток скорости движения.

Ваша задача: написать программу, которая по анализу приходящих данных **определяет скорость** автомобиля и **момент** (когда автомобиль проезжает ближе всего к посту ДПС) необходимости снимка.

Скорость звука считать **не зависящей** от частоты звука и равной 0,3 км/с. Спадание амплитуды принимаемого звука с расстоянием считать **отсутствующим**. Движение автомобиля начинается **задолго** до подхода к посту ДПС и заканчивается **спустя значительное время** после его проезда. Сигнал от автомобиля изменяется по закону $U = \cos(\omega \cdot t/20)$. Собственная частота ω_0 машины фиксирована и равна 0,5 рад/с.

Входной формат: 5000 строк данных, на каждой по два числа через пробел, **время** отсчета в секундах и **сигнал** от автомобиля.

0 1.0
1 0.984807753012208
2 0.9396926207859083
3 0.8660254037844387
4 0.766044443118978
5 0.6427876096865394
6 0.5000000000000001
7 0.3420201433256688

Выходной формат: два числа через пробел, скорость автомобиля в км/с и время, в которое автомобиль проехал ближе всего к посту ДПС в секундах.

*0.**035** 2391.**5***

Ограничения работы программы: 15 секунд, 256 Мб.

Для решения этой задачи на всю команду дается 20 попыток.

Ваше решение будет проверено на наборе **тестов**. Каждый тест оценивается **отдельно**, и балл за решение складывается из **соотношения** суммарного балла с мак-

симально возможным. Оценка за тест зависит от достигнутой **точности** вычисления. **Базовые** 1/3 от максимального балла ставятся, если скорость вычислена с точностью до 0,1 км/с, а время — до 500 с. Еще по 1/3 балла можно получить, если отдельно обеспечить точность вычисления скорости до 0,01 км/с, а времени — до 150 с. То есть если точность по обеим величинам будет достигнута **одновременно**, вы получите **полный** балл за тест. С другой стороны, при ошибке в коде или неудовлетворении «слабого» критерия, за тест вы не получите ничего.

Алгоритм решения

1. Считываем данные.
2. Находим истинную частоту сигнала, который приходит на автономную систему слежения в начале пути.
3. Подставляем найденную частоту в формулу скорости для отрезка пути **до** поста ДПС.
4. Начинаем перебирать все значения на поиск момента, когда частота сигнала изменится. Этот момент и будет временем, когда беспилотник проезжает мимо поста ДПС.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
baked_data = map(str.split, data.splitlines(False))
        x = [float(b) for _, b in baked_data]
        before = np.array(x[:2000])
        w = np.fft.fft(before)
        n = 2000 / len(before)
        freqs = np.fft.fftfreq(2000)
        idx = np.argmax(np.abs(w))
        freq = freqs[idx]
        freq_in_hertz = abs(freq * 20)
        pre = freq_in_hertz * n * 2 * math.pi
        V = 0.3 * (1 - 0.5 / pre)
        freq_curr = pre
        i = 0
        while i < 3000 and abs(pre-freq_curr) <= 0.016:
                before = np.array(x[i:i+2000])
                w = np.fft.fft(before)
                n = 2000 / len(before)
                 freqs = np.fft.fftfreq(len(before))
                idx = np.argmax(np.abs(w))
                freq = freqs[idx]
                 freq_curr = abs(freq * 20) * n * 2 * math.pi
                i += 1
        poz = i + 1000
```

Задача К4. Цифровой витраж (10,3 баллов)

Человек установил в своей комнате смарт-окно. Прогуливаясь по городу, человек захотел установить на свое умное окно рисунок, который он заснял на камеру своего телефона. Камера **оцифровала** изображение и отправила его на сервер дома в виде

массива из нулей и единиц, где **единица** означает **наличие** линии, а **ноль** — ее **отсутствие**.

Однако используемый для передачи рисунка канал связи оказался с **помехами**. В результате передачи данных по этому каналу на рисунке появился **шум**, из-за чего некоторые нули **превратились** в единицы и наоборот.

Ваша задача: зная, что передаваемый рисунок был **плюсом** с **одинаковой** длиной линий во всех направлениях, определить **центр** плюса и **перезаписать** файл рисунка **без помех**, исправив все нули, ставшие единицами, и наоборот. В рамках данной задачи условимся, что количество помех **не превышает** 1% от общего числа единиц рисунка, а линии плюса полностью **параллельны** границам рисунка. Гарантируется, что параметры плюса можно определить **однозначно**.

Входной формат: прямоугольная матрица размером 800 столбцов на 800 строк из нулей и единиц. Матрица будет передана в программу в следующем виде (строки слитные и разделены переносом). Пример формата:

01100010
01100100
01101110
00000000
10000001
01111110

Выходной формат: два целых числа через пробел, координаты центра плюса (**нумерация с 0**, порядок как в **растре**, то есть сначала столбец X, затем строка Y), затем перевод строки и далее строки из нулей и единиц без пробелов, новый рисунок без помех.

3 14
01100010
01100100
01101110
00000000
10000001
01111110

Ограничения работы программы: 3 секунды, 256 Мб.

Для решения этой задачи команде дается 20 попыток.

За написание программы, которая будет правильно осуществлять передачу, участник получает максимальный балл.

Алгоритм решения

1. Считываем данные.
2. Находим центр плюса.
3. Так как имеются помехи, искать центр, как единицу с четырех сторон которой находятся единицы, не имеет смыла.

4. Наиболее простой способ, найти строку и столбец с наибольшим количеством единиц.
5. Рассчитываем его размер. Так как в результате помех на его гранях могут возникнуть нули, необходимо вести подсчет с наличием двух или хотя бы трех единиц на одном расстоянии от центра.
6. Зная параметры плюса, необходимо просто добавить единиц на матрицу из нулей, чтобы считать помехи устраненными.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
for i in range(n):
        matrix.append([int(_) for _ in input()])
for i in range(n):
        gor.append((i, sum({matrix[i][j] for j in range(n)})))
        vert.append((i, sum([{matrix[j][i] for j in range(n)})))
max_gor = max(gor, key=lambda x: x[1])
max_vert = max(vert, key=lambda x: x[1])
center_x = max_vert[0]
center_y = max_gor[0]
len1 = len2 = len3 = len4 = 0
for k in range(center_y, -1, -1):
        if matrix[k][center_x]:
                len1 = abs(k - center_y) +1
for k in range(center_y, n):
        if matrix[k][center_x]:
                len2 = abs(k - center_y) +1
for k in range(center_x, -1, -1):
        if matrix[center_y][k]:
                len3 = abs(k - center_x) +1
for k in range(center_x, n):
        if matrix[center_y][k]:
                len4 = abs(k - center_x) + 1
res_matrix = [[0] * n for i in range(n)]
len1 = min(len1, len2, len3, len4)
for i in range(len1):
        res_matrix[center_y + i][center_x] = 1
        res_matrix[center_y - i][center_x] = 1
        res_matrix[center_y][center_x + i] = 1
        res_matrix[center_y][center_x - i] = 1
print(center_x, center_y)
for i in res_matrix:
        print(''.join([str(j) for j in i]))
```

Задача K5. Слепое прослушивание (11,59 баллов)

Занимаясь исследованиями на Северном полюсе, Вы решили наладить прямую связь с Новосибирском. Известно, что между вами есть **десять** каналов связи с неизвестными характеристиками. Попробовав передать по ним данные, вы обнаружили, что все каналы имеют **помехи**, возникающие с **фиксированной** частотой.

Вы решаете определить **наиболее устойчивый** из этих каналов. Для этого необходимо написать **две** функции. **Первая** должна сгенерировать такую последовательность байт, чтобы **вторая** функция по возвращении данных из Новосибирска

могла определить, каков процент ошибок в пришедшем файле и выбрать канал с **наименьшим числом ошибок**.

Формат реализации: две функции. Функция init возвращает список байт. Функция evaluate на вход принимает массив из десяти таких списков (результат передачи через соответствующий канал) и должна вернуть **номер оптимального канала (нумерация с 0)**.

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. Также учитывайте, что в ограничение на время выполнения входит время работы **вспомогательного** кода (зависит от возвращаемого списка байт, но в среднем не более секунды).

Ограничения работы программы: 5 секунд, 256 Мб.

Для решения этой задачи команде дается 20 попыток.

За написание программы, которая будет правильно осуществлять передачу, участник получает максимальный балл.

Алгоритм решения

1. Создаем данные для отправки.
2. Кодируем созданные данные.
3. Отправляем на выход функции init.
4. Считываем данные в функции evaluate.
5. Подсчитываем количество ошибок в каждом канале и выдаем номер самого оптимального.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
def init() -> bytes:
        return b"00000000000000000000000000000000"

def evaluate(outs: list) -> int:
        matchIndex, minErrors = None, None
        for index, binaryCode in enumerate(outs):
                countOfErrors = 0
                for byte in bytearray(binaryCode):
                        if byte != 48:
                                countOfErrors += 1
                if (matchIndex, minErrors) == (None, None) or countOfErrors <
                ↪  minErrors:
                    matchIndex = index
                    minErrors = countOferrors
        return matchIndex
```

Бонусные задачи

Задача Б1. Кабельное подключение (6,44 баллов)

Прогресс не стоит на месте, в одном городе потребовалось **заменить** устаревшую систему телевещания на цифровую. Для обеспечения наилучшего качества телевещания, было решено установить комплекс передающих антенн для полного обхвата территории города. Ретранслировать информацию по воздуху не очень рационально, поэтому антенны **соединяются** между собой кабелями. Но любое обеспечение связи в таких масштабах связано с ресурсами, а ресурсы нужно расходовать бережно.

Вам известны **координаты** точек расположения передающих антенн на условной декартовой плоскости (условимся, что первая точка соответствует главной передающей антенне). Необходимо **соединить все точки** в единую систему так, чтобы **минимизировать суммарную длину** проброшенных кабелей между антеннами и только ее. Линии протягиваются между **парами** точек, к одной точке может быть подключено **несколько** линий. В этой задаче мы **не учитываем** потери и прочие параметры.

Входной формат: список пар целых чисел, где каждая пара — **координаты** соответствующей точки расположения передающей антенны. Например,

[(-22, 42), (-18, 39), (-64, -10), (-77, 38), (-17, 62), (68, -44), (-23, 21), (71, 74), (-98, 67)]

Выходной формат: список пар целых чисел, где каждая пара описывает отдельную линию между точками расположения, а число в этой паре означает **порядковый номер** точки во входном списке (**нумерация с 0**). Например,

[(0, 1), (1, 0), (2, 13), (1, 5)]

Ограничения работы программы: 3 секунды, 256 Мб.

Для решения этой задачи команде дается 20 попыток.

За написание программы, которая будет правильной осуществлять передачу, участник получает максимальный балл.

В случае, если длина линий в полученной сети не больше, чем в «авторской», на 10^{-5}, ваше решение будет зачтено при условии, что ваша сеть **связна и оптимальна**.

Алгоритм решения

1. Считываем данные.
2. Находим расстояние между всеми антеннами.
3. Находим оптимальные расстояние для всех антенн и выявить наиболее оптимальные для каждой.
4. Создаем общую сеть из наиболее оптимальных путей у каждой антенны.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
def dist(a, b):
        return hypot(a[0]-b[0], a[1]-b[1])
def get(a):
        if p[a] == a:
                return p[a]
        else:
                p[a] = get(p[a])
                return p[a]
nodes = eval(data)
n = len(nodes)
p = [i for i in range(n)]
edges = []
for i in range(n):
        for j in range(i+1, n):
                edges.append((dist(nodes[i], nodes[j]), i, j))
edges.sort()
ans = []
for (_, a, b) in edges:
        da, db = get(a), get(b)
        if da != db:
                if p[da] != p[db]:
                        p[da] = db
                ans.append((a, b))
return str(ans)
```

Задача Б2. PatchMe (4,51 баллов)

Будучи широко известным в узких кругах инди-разработчиком, вы опубликовали свою очередную игру. Но вскоре вам стали приходить жалобы на определенного рода баги. Чтобы не потерять лояльную аудиторию, вам необходимо срочно **исправить** эти ошибки. Но игра имеет довольно большой размер, и мало кто согласится скачивать ее заново. Решением же будет программа-**патчер**, которая занимает мало места и содержит в себе лишь те участки, которые надо **переписать** в уже имеющихся файлах.

Ваша задача: **написать** такой патчер, который будет **заменять** в непрерывной последовательности байт их отдельные **цепочки**. Однако заменять их нужно **в порядке следования в исходной строке** с определенным **приоритетом**, чтобы на выходе получилось что-то играбельное. Более того, в рамках этой задачи каждая цепочка может встречаться **неоднократно**, и необходимо заменить **каждое** ее вхождение (повторимся, с учетом приоритета), причем та или иная цепочка может появиться **раньше**, чем та, что указана выше нее в списке замен.

Входной формат: несколько строк, на **первой** строке дается длина строки в **байтах**, как целое число M (до 1000). Далее на **второй** строке приводится байтовая строка из M байт. На **третьей** строке приводятся целые числа N (количество подстрок, от 1 до 3) и K (длина каждой подстроки). Последующие $2 \cdot N$ байтовых строк по K байт содержат: на **нечетных** местах — исходные подстроки, на **четных** — те, которые необходимо заменить. Четность определяется именно среди этих строк, **без учета** остальных. Сами пары строк приводятся в порядке **убывания** приоритета замены. Например,

```
10
abcdefghij
```

2 5
abcde
xyzzy
defgh
hello

Выходной формат: строка данных, пропатченная цепочка из M байт.

Ограничения работы программы: 2 секунды, 256 Мб.

Для решения этой задачи у вашей команды есть 20 попыток.

За написание программы, которая будет правильной осуществлять передачу, участник получает максимальный балл.

Алгоритм решения

1. Считать данные.
2. Запоминаем все подстроки.
3. Начинаем динамический перебор символов строки на предмет наличия запомненной нечетной подстроки.
4. При нахождении фрагмента подстроки заменяем его.
5. Повторно проделываем алгоритм проверки, на случай перебора не всей строки.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
m = int(input())
a = input()
n, k = map(int, input().split())  # число и длина подстрок
repa, repb = [], []
for _ in range(n):
        repa.append(input())
        repb.append(input())
na = []
i = 0
while i < m-k:
        for j in range(n):
                if a[i:i+k] == repa[j]:
                na.append(repb[j])
                i += k
                break
        else:
                na.append(a[i])
                i += 1
if i != m:
        for j in range(n):
                if a[i:i+k] == repa[j]:
                na.append(repb[j])
                break
        else:
                na.append(a[i:])
print("".join(na))
```

Задача Б3. Код для мотивации (9,66 баллов)

Подзадача 1. Пока вы работали в тяжелых условиях на Северном полюсе, вам пришел конверт от вашего друга по переписке. В конверте была записка, где написано лишь «**Try Hamming's (12,8), I've used plain ASCII**». Также в конверте была флешка с единственным **двоичным** файлом.

Так как Вам уже приходилось работать с беспроводной связью, Вы поняли, что на флешке находится **закодированное сообщение**. Известно, что письмо перевозилось в щадящих условиях, поэтому гарантируется **отсутствие ошибок** в сообщении.

Ваша задача: **декодировать** данные и перевести их в **читаемое** сообщение.

Входной формат: одна строка из нулей и единиц, соответствующая присланному двоичному **шифротексту**.

Выходной формат: одна строка, содержащая декодированное сообщение.

Для решения этой задачи на команде дается 20 попыток.

За выполнение данной задачи участник получает 2/5 от максимального количества баллов.

Подзадача 2. Вам удалось пробросить канал связи до своего товарища, но этот канал **неустойчив**, и в нем есть достаточно большая вероятность появления **одиночной** ошибки на каждый 12-битовый блок.

Задача остается той же, но теперь вам необходимо написать **программу**. Форматы данных идентичны предыдущей подзадаче.

Входной формат: одна строка из нулей и единиц, соответствующая присланному двоичному **шифротексту**.

Выходной формат: одна строка, содержащая декодированное сообщение.

Ограничения работы программы: 2 секунды, 256 Мб.

Для решения этой задачи на команде дается 20 попыток.

За написание программы, которая будет правильной осуществлять передачу, участник получает 3/5 от максимального балла.

Алгоритм решения

1. Считываем данные.
2. Отделяем блоки по 12 бит.
3. Исправляем ошибку, если она присутствует. Для этого подсчитываем контрольные суммы для всех контрольных бит. Место ошибки и последующей инверсии находится, как сумма неправильных контрольных бит.
4. Удаляем контрольные биты.
5. Превращаем получившийся 8 битный блок в соответствующий ASCII символ и присоединяем его к ответу.

Пример программы-решения

Ниже представлено решение на языке Python3.

```
def correction(parity_bit):
        cur_bit = parity_bit
        indexes = []
        while cur_bit <= 12:
                for i in range(parity_bit):
                        indexes += [cur_bit + i - 1]
                cur_bit += 2 * parity_bit
        while indexes[-1] > 11:
                del indexes[-1]
        return indexes[1:]
raw_data = input()
result = ''

for r in range(0, len(raw_data), 12):
        s = raw_data[r:r + 12]
        errors = []
        for _parity_bit in _parity_bits:
                indexes = correction(_parity_bit)
                parity_bit = 0
                for i in indexes:
                        if s[i] = '1':
                                parity_bit = (parity_bit + 1) % 2
                if parity_bit != int(s[_parity_bit - 1]):
                        errors += [_parity_bit]
        if len(errors) > 0:
                error = sum(errors) - 1
                s = s[:error] + ('1' if s[error] == '0' else '0') + s[error + 1:]
        s = s[2] + s[4:7] + s[8:12]
        result = result + chr(int(s, 2))
print(result)
```