

Задача 1: Статистика текста

Прежде чем нейросеть научится писать стихи или отвечать на вопросы, ей необходимо понять базовые закономерности языка. Анализ длины слов — первый шаг в подготовке данных для обучения языковых моделей. Эта задача погружает вас в фундаментальные принципы обработки естественного языка (NLP), которые лежат в основе таких систем, как ChatGPT.

Условие задачи

Дан текстовый файл `text.txt`, содержащий произвольный текст на русском и английском языках. Необходимо:

1. Извлечь все слова из текста согласно правилам ниже.
2. Рассчитать три статистические метрики для длин извлеченных слов:
 - Среднее арифметическое (с точностью 6 знаков после запятой)
 - Медиану (нижнюю медиану при четном числе элементов)
 - Моду (с разрешением конфликтов в пользу наименьшего значения)
3. Сохранить результаты в два файла: `words.txt` — список извлеченных слов - `result.txt` — вычисленные метрики
4. Заархивировать оба файла в `task1.zip`.



Теоретическая часть

Что такое слово?

Слово — непрерывная последовательность буквенных символов, где:

- **Буквенные символы:**

- Латинские буквы: a-z, A-Z
- Кириллические буквы: а-я, А-Я, включая ё и Ё
- **Разделители:** все остальные символы (цифры, знаки препинания, пробелы, спецсимволы).

Примеры:

- "Привет, world123! Это-тест." → Привет, world, Это, тест
- "Hello-world" → Hello, world
- "abc123def" → abc, def

Статистические метрики

1. Среднее арифметическое

Для списка длин слов $L = [l_1, l_2, \dots, l_n]$:

$$\bar{l} = \frac{1}{n} \sum_{i=1}^n l_i$$

Требование: результат выводится с точностью 6 знаков после запятой.

2. Медиана

Значение в середине упорядоченного списка. При четном n берется **нижняя медиана** (элемент с индексом $(n - 1) // 2$ в отсортированном списке).

Пример:

$[2, 3, 5, 8] \rightarrow n = 4$, индекс $(4 - 1) // 2 = 1 \rightarrow$ медиана = 3.

3. Мода

Наиболее частая длина. При равенстве частот выбирается **наименьшее** значение.

Пример:

$[2, 3, 2, 4, 3, 4] \rightarrow$ частоты: $2 \rightarrow 2, 3 \rightarrow 2, 4 \rightarrow 2 \rightarrow$ мода = 2.

Входные данные

- Файл `text.txt` в кодировке UTF-8 с произвольным текстом.

Выходные данные

Архив task1.zip, содержащий два файла:

1. words.txt
Каждая строка — слово из исходного текста в порядке появления.

Пример (для текста "Hello, world! Это тест123."):

```
Hello
world
Это
тест
```

2. result.txt
Три строки с вычисленными метриками:

```
3.833333
4
4
```

Критерии оценивания

№	Критерий	Баллы
1.1	Корректное извлечение слов (учет разделителей)	25
1.2	Правильное вычисление среднего и медианы	35
1.3	Корректный выбор моды (включая обработку равных частот)	40
	Итого	100

Технические требования

- Используйте только стандартные библиотеки Python (re, collections и др.).
- Среднее арифметическое должно иметь точность $\pm 1e-6$.
- Медиана и мода — целые числа.
- Все файлы должны быть в кодировке UTF-8.

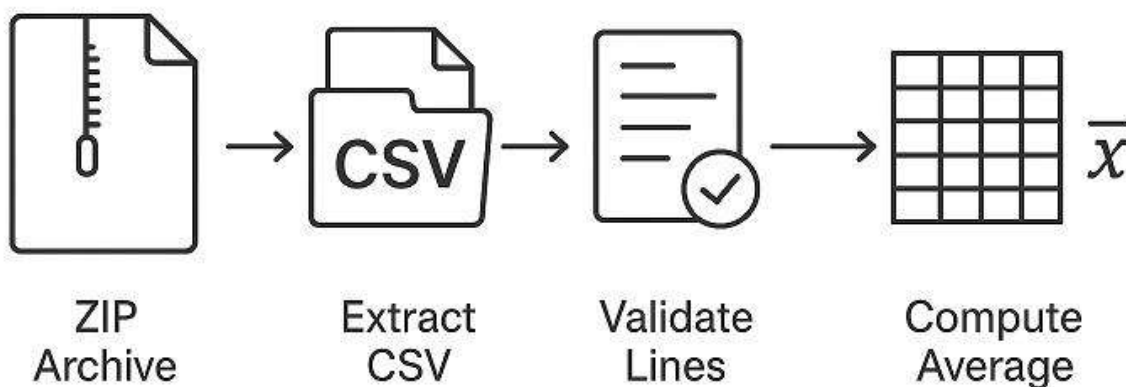
Задача 2: Чистильщик данных

В реальном мире данные редко бывают идеальными. Как археологи, вычищающие древние артефакты от пыли, дата-сайентисты тратят большую часть времени на очистку “грязных” данных. Пропущенные значения, ошибки датчиков, мусорные символы — всё это мешает анализу. Сегодня вы научитесь превращать хаотичные данные в чистые и пригодные для работы числа. Без этого ни одна модель искусственного интеллекта не сможет дать точный результат!

Условие задачи

Дан ZIP-архив `data.zip`, содержащий файлы различных форматов. Необходимо:

1. Найти все CSV-файлы в архиве (регистр расширения не важен: `.csv`, `.CSV` и т.д.).
2. Из каждой строки CSV-файлов извлечь корректные данные, удовлетворяющие правилам ниже.
3. Сохранить все корректные строки в файл `valid_lines.txt`.
4. Вычислить среднее арифметическое для каждого из трёх столбцов по всем корректным строкам и записать результаты в файл `result.txt` с точностью 6 знаков после запятой.
5. Заархивировать оба файла в `task2.zip`.



Теоретическая часть

Что такое CSV?

CSV (Comma-Separated Values) — текстовый формат хранения табличных данных, где значения разделены запятыми. Для этой задачи действуют строгие правила форматирования.

Правила корректной строки CSV:

- Каждая строка содержит ровно три вещественных числа
- Числа разделены запятыми **без пробелов**
- Все три значения должны быть корректными вещественными числами в десятичном формате

Примеры корректных строк:

1.5,2.3,4.1
-0.5,10,3.14
0.0,0.0,0.0

Примеры некорректных строк:

1.5, 2.3, 4.1	← пробелы вокруг запятых
1.5,2.3	← меньше трёх значений
1e3,2.3,4.1	← экспоненциальная запись
abc,2.3,4.1	← нечисловое значение
1.5,,4.1	← пропущенное значение

Что такое вещественное число в задаче?

Допустимые форматы:

- Целые числа: 42, -17, 0
- Дробные числа: 3.14, -0.5, 100.0
- Числа с нулевой дробной частью: 0.0, -0.0

Запрещено использовать:

- Экспоненциальную запись (1e3, 2.5e-2)
- Спецсимволы (inf, NaN, null)
- Числа с пробелами или другими разделителями

Среднее арифметическое по столбцу

Для каждого столбца j ($j = 1, 2, 3$) среднее вычисляется как:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{ij},$$

где N — общее количество корректных строк, x_{ij} — значение в i -й строке и j -м столбце.

Входные данные

Архив data.zip в кодировке UTF-8, содержащий: - Файлы с расширением .csv (регистр не важен) в корне архива - Другие файлы (.txt, .json, без расширения) и файлы во вложенных папках **игнорируются**

Пример структуры архива:

```
data.zip
├── file1.csv      ← обрабатываем
├── file2.txt      ← игнорируем
├── data.CSV       ← обрабатываем (регистр не важен)
├── subfolder/
│   └── more.csv   ← игнорируем (вложенные папки не обрабатываются)
```

Выходные данные

Архив task2.zip, содержащий два файла:

3. valid_lines.txt
 - Каждая строка: три корректных числа в формате {число1}, {число2}, {число3}
 - Порядок строк: сохраняется порядок обработки файлов и строк внутри них
 - Кодировка UTF-8
4. result.txt
 - Одна строка с тремя числами через пробел: {mean_col1} {mean_col2} {mean_col3}
 - Точность: ровно 6 знаков после запятой
 - Пример: -3.000000 1.000000 34.380000

Критерии оценивания

№	Критерий	Баллы
2.1	Корректная фильтрация CSV-файлов и отсеивание некорректных строк (формат файлов, количество столбцов, тип данных)	50
2.2	Правильное вычисление средних по столбцам с требуемой точностью	50
Итого		100

Технические требования

- Используйте только стандартные библиотеки Python (zipfile, csv, re и др.)
- Числовой допуск для средних: $\pm 10^{-6}$
- Порядок обработки файлов может быть произвольным
- Все файлы должны быть в кодировке UTF-8
- При обработке файлов учитывайте только файлы в корневой директории архива

Задача 3: Детектор аномалий во временном ряде

Каждую секунду миллионы датчиков собирают данные о работе критически важных систем: от сердечного ритма пациентов до состояния ядерных реакторов. Раннее обнаружение аномалий позволяет предотвратить катастрофы и спасти жизни. В основе таких систем лежит анализ локального контекста — сравнение текущего значения с характерным поведением системы в ближайшем окружении. Эта задача погружает вас в фундаментальные принципы временных рядов и статистического контроля качества, которые используются в медицине, промышленности и научных исследованиях.

Условие задачи

Дан временной ряд измерений в файле `series.txt`. Для каждого элемента ряда с индексом i (начиная с 0) необходимо определить, является ли он аномалией относительно своего локального окружения.

Аномалией считается точка x_i , если одновременно выполняются условия:

1. Для неё можно построить полное скользящее окно размером 11 точек (т.е. $5 \leq i \leq N - 6$, где N — общее количество точек).
2. Её отклонение от локального среднего превышает удвоенное стандартное отклонение: $|x_i - \mu_i| > 2 \cdot \sigma_i$.
3. Стандартное отклонение в окне положительно: $\sigma_i > 0$.

Требуется:

1. Для всех точек из диапазона $[5, N - 6]$ рассчитать μ_i (среднее) и σ_i (стандартное отклонение) для их окон.
2. Сформировать файл `checks.txt` с результатами вычислений.
3. Сформировать файл `result.txt` со списком индексов аномальных точек.
4. Заархивировать оба файла в `task3.zip`.

Теоретическая часть

Временной ряд

Временной ряд — последовательность значений, измеренных в последовательные моменты времени. Примеры: температура воздуха каждый час, пульс пациента каждую секунду, показания датчика вибрации.

Скользящее окно

Для анализа локального контекста вокруг точки x_i используется окно из 11 соседних значений: 5 точек слева, сама точка x_i и 5 точек справа. Индексы окна для точки i : $[i - 5, i - 4, \dots, i, \dots, i + 4, i + 5]$.

Важно: Окно можно построить только для индексов i , для которых существуют все соседи ($5 \leq i \leq N - 6$).

Среднее значение (μ)

Среднее арифметическое значений в окне:

$$\mu_i = \frac{1}{11} \sum_{k=i-5}^{i+5} x_k$$

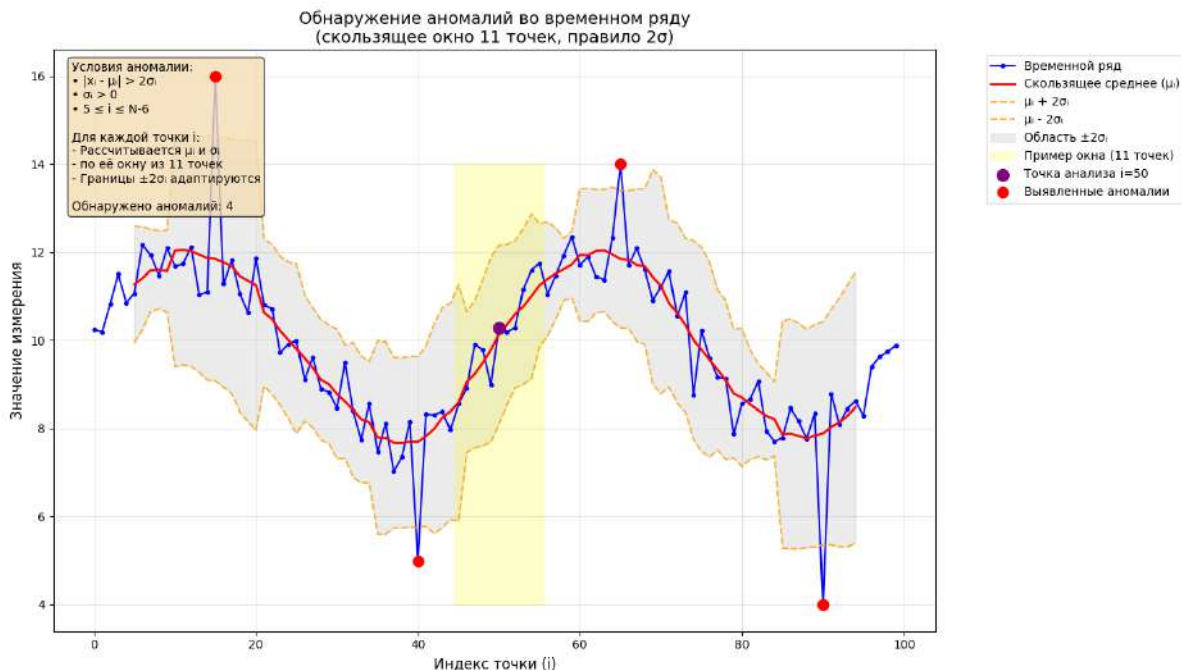
Стандартное отклонение (σ)

Мера разброса значений относительно среднего (популяционная формула):

$$\sigma_i = \sqrt{\frac{1}{11} \sum_{k=i-5}^{i+5} (x_k - \mu_i)^2}$$

Правило 2σ

В нормально распределённых данных $\sim 95\%$ значений лежат в пределах $\mu \pm 2\sigma$. Точки за этим интервалом считаются статистически значимыми аномалиями.



Входные данные

- Файл `series.txt` содержит N строк. Каждая строка — вещественное число x_i , представляющее измерение в момент времени i .
Файл в кодировке UTF-8.

Выходные данные

Архив `task3.zip`, содержащий два файла:

1. `checks.txt` — для каждого индекса i из диапазона $[5, N - 6]$:
 - Строка формата: `{i} {mu_i:.8f} {sigma_i:.8f} {is_anomaly}`
 - Где `is_anomaly` = 1, если точка аномальная, иначе 0.
2. `result.txt`
 - индексы всех обнаруженных аномалий (по одному на строку, без дубликатов, в порядке возрастания). Если аномалий нет
 - файл пустой.

Пример для входных данных из 16 точек:

```
# checks.txt
5 10.00000000 0.00000000 0
6 10.00000000 0.00000000 0
7 13.63636364 11.49912255 0
8 13.63636364 11.49912255 0
9 13.63636364 11.49912255 1
10 13.63636364 11.49912255 0
```

```
# result.txt
9
```

Файлы `checks.txt` и `result.txt` должны быть заархивированы в `task3.zip`.

Критерии оценивания

№	Критерий	Баллы
3.1	Корректная реализация алгоритма анализа локального контекста (скользящее окно и вычисление статистик)	60
3.2	Верное определение аномалий по правилу 2σ и формирование выходных файлов	40
Итого		100

Технические требования

- Для стандартного отклонения использовать **популяционную формулу**.
- Погрешность вычислений μ и σ не должна превышать 10^{-6} .
- Значения μ_i и σ_i выводить с точностью не менее 8 знаков после запятой.
- Все файлы должны быть в кодировке UTF-8.
- Использовать только стандартные библиотеки Python.

Задача 4: Простая рекомендательная система

Рекомендательные системы — невидимые помощники современности. Они подбирают фильмы, музыку, товары и даже друзей. В основе многих из них лежит простая идея: «Тебе понравится то, что нравится похожим людям». В этой задаче вы реализуете классический алгоритм коллаборативной фильтрации — тот самый, с которого начиналась эпоха персонализации. Возможно, именно такие расчёты однажды помогут найти кому-то идеальную книгу... или даже решение сложной проблемы.

Условие задачи

Дан файл `ratings.txt`, содержащий матрицу оценок пользователей товарам. Ваша задача:

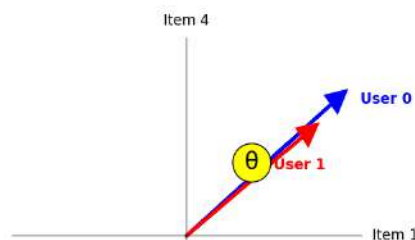
1. Для каждого пользователя $v \neq 0$ вычислить косинусную схожесть с пользователем 0 **только по общим оценённым товарам**.
2. Найти пользователя $v_{\max}$ с максимальной схожестью.
3. Среди товаров, которые пользователь 0 **не оценил** ($r_{0j} = 0$), выбрать товар, который получил **максимальную оценку** от $v_{\max}$. Если таких товаров несколько — выбрать товар с минимальным индексом.
4. Вывести результат в виде архива `task4.zip`, содержащего два файла:
 - `similarities.txt`
 - `result.txt`

User-Item Rating Matrix

User/Item	Item 1	Item 2	Item 3	Item 4	Item 5
User 0	5	3	0	4	0
User 1	4	0	5	3	2
User 2	0	4	3	5	1
User 3	3	5	4	0	3

0 = не оценено

Cosine Similarity



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

A = вектор оценок User 0
B = вектор оценок User 1

Similarity = 1.000

Collaborative Filtering: User-Based Similarity

Теоретическая часть

Что такое матрица оценок?

Матрица оценок $R \in \mathbb{R}^{U \times I}$ — двумерная таблица, где:

- U — количество пользователей (строки),
- I — количество товаров (столбцы),
- r_{ui} — оценка пользователя u товару i (целые значения от 0 до 5, где 0 означает отсутствие оценки).

Пример матрицы 4×6:

	Товар 0	Товар 1	Товар 2	Товар 3	Товар 4	Товар 5
User 0	5	0	4	0	0	3
User 1	4	0	5	0	0	4
User 2	0	3	0	2	5	0
User 3	5	0	3	0	0	4

Косинусная схожесть пользователей

Косинусная схожесть — мера сходства двух векторов, основанная на угле между ними. Для двух пользователей u и v вычисляется **только по общим оценённым товарам** I_{uv} :

$$\text{sim}(u, v) = \frac{\sum_{j \in I_{uv}} r_{uj} \cdot r_{vj}}{\sqrt{\sum_{j \in I_{uv}} r_{uj}^2} \cdot \sqrt{\sum_{j \in I_{uv}} r_{vj}^2}}$$

где $I_{uv} = \{j \mid r_{uj} > 0 \text{ и } r_{vj} > 0\}$ — множество индексов товаров, оценённых обоими пользователями.

Пример вычисления:

Для User 0 ([5,0,4,0,0,3]) и User 3 ([5,0,3,0,0,4]):

- Общие товары: индексы {0,2,5},
- Векторы оценок по общим товарам: [5,4,3] и [5,3,4],
- Числитель: $5 \cdot 5 + 4 \cdot 3 + 3 \cdot 4 = 25 + 12 + 12 = 49$,

- Знаменатель: $\sqrt{5^2 + 4^2 + 3^2} \cdot \sqrt{5^2 + 3^2 + 4^2} = \sqrt{50} \cdot \sqrt{50} = 50$,
- $\text{sim}(0,3) = 49/50 = 0.98$.

Входные данные

Файл `ratings.txt` в кодировке UTF-8:

- Каждая строка соответствует одному пользователю.
- Числа разделены пробелами.
- Первая строка — целевой пользователь (User 0).
- Пример содержимого:

```
5 0 4 0 0 3
4 0 5 0 0 4
0 3 0 2 5 0
5 0 3 0 0 4
```

Выходные данные

Архив `task4.zip` должен содержать два файла:

1. `similarities.txt`

- Для каждого пользователя $v \neq 0$, имеющего хотя бы один общий оценённый товар с User 0, строка в формате:
- `{user_id} {similarity}`
- `similarity` выводится с точностью 8 знаков после запятой.
- Порядок строк может быть произвольным.
- Пример:

```
1 0.97402464
3 0.98000000
```

2. `result.txt`

- Одна строка, содержащая целое число — индекс рекомендуемого товара.
- Пример:

```
1
```

Критерии оценивания

№	Критерий	Баллы
4.1	Корректное чтение матрицы и определение общих товаров	20
4.2	Правильный расчёт косинусной меры для всех пользователей	40
4.3	Верный выбор рекомендации (максимум оценки, при равенстве – минимум ID)	40
Итого		100

Технические требования

- Используйте только стандартные библиотеки Python (math, json и др.).
- Числовой допуск для схожести: $\pm 10^{-6}$.
- Гарантируется наличие хотя бы одной возможной рекомендации.
- Все файлы должны быть в кодировке UTF-8.
- При одинаковой максимальной схожести между несколькими пользователями выбирается любой из них.
- При одинаковых максимальных оценках у товаров выбирается товар с наименьшим индексом.

Задача 5: Анализ n-грамм

Нейросети не читают слова так, как это делают люди. Они анализируют последовательности символов и учатся на основе того, какие сочетания встречаются чаще всего. Такие пары и тройки символов как «пр», «ст», «ть» несут в себе ритм и структуру языка. Анализ -грамм — один из самых старых, но мощных методов для кодирования языка в машинном обучении. На его основе работают спам-фильтры, переводчики и системы анализа текстов. Ваша задача — расшифровать «ДНК языка», найдя наиболее частые последовательности символов.

Условие задачи

Дано: Архив `corpus.zip`, содержащий несколько текстовых файлов с расширением `.txt` (регистр расширения не важен).

Требуется:

1. Обработать все текстовые файлы в архиве в **алфавитном порядке имен файлов** (регистронезависимо), объединив их содержимое в указанном порядке.
2. Очистить объединенный текст согласно правилам предобработки.
3. Подсчитать частоты униграмм (1-грамм) и биграмм (2-грамм) в очищенном тексте.
4. Найти топ-10 наиболее частых униграмм и биграмм с учётом правил сортировки.
5. Подготовить два файла: `cleaned.txt` (очищенный текст) и `result.txt` (частоты -грамм).

Результат: Архив `task5.zip`, содержащий файлы `cleaned.txt` и `result.txt`.

Теоретическая часть

Что такое -грамма?

n -грамма — это последовательность из n подряд идущих символов в тексте. Например:

- **Униграмма** ($n = 1$): отдельные символы, например, “П”, “р”, “и”.
- **Биграмма** ($n = 2$): пары символов, например, “Пр”, “ри”, “ив”.

Из строки длины L можно получить:

- L униграмм,

- $L - 1$ биграмм.

Очистка текста

Оставить только буквы:

- Латинские буквы: a-z, A-Z
- Кириллические буквы: а-я, А-Я, ё, Ё
- Удалить всё остальное: цифры, знаки препинания, пробелы, специальные символы.

Пример очистки:

Исходный текст: "Привет, мир123!!! Это AI."

→ Только буквы: "ПриветмирЭтоAI"

Порядок обработки файлов

Для обеспечения воспроизводимости результата файлы обрабатываются **в алфавитном порядке имен файлов** (регистронезависимо). Например:

- file1.txt, file2.txt, notes.TXT, README.txt
- Порядок обработки: file1.txt → file2.txt → notes.TXT → README.txt

Подсчёт частот и сортировка

- **Частота** — количество вхождений -граммы в текст.
- **Правила сортировки топ-10:**
 1. По убыванию частоты.
 2. При равных частотах — по возрастанию ASCII-кода символов.

Порядок ASCII (важные символы):

'A' < 'B' < ... < 'Z' < 'a' < 'b' < ... < 'z' < 'А' < 'Б' < ... < 'Я' < 'а' < 'б' < ... < 'я'

Пример сортировки: Для униграмм с частотами:

'a': 10, 'б': 10, 'А': 10, 'в': 5, 'В': 5

Топ-3 после сортировки: А, а, б.

Входные данные

- Архив corpus.zip содержит файлы с расширением .txt (например, file1.txt, notes.TXT).

- В обработке участвуют **только** файлы с расширением .txt (регистронезависимо).
- Кодировка файлов: UTF-8.
- Порядок обработки: алфавитный порядок имен файлов (регистронезависимо).

Выходные данные

Архив task5.zip, содержащий два файла:

1. cleaned.txt:
 - Очищенный текст **без пробелов**.
 - Весь текст записывается в одну строку.
 - Пример содержимого: ПриветмирЭтотестHelloWorld.
2. result.txt:
 - Ровно 20 строк в формате: {n_грамма}<TAB>{частота}.
 - Первые 10 строк: топ-10 униграмм.
 - Следующие 10 строк: топ-10 биграмм.
 - Разделитель — символ табуляции (\t).
 - Пример содержимого:


```
т 4
л 3
о 3
...
He 1
ll 1
lo 1
...
```

Критерии оценивания

№	Критерий	Баллы
5.1	Корректное чтение файлов в алфавитном порядке, объединение	30
№	Критерий	Баллы
	содержимого и правильная очистка текста	
5.2	Топ-10 униграмм с правильной сортировкой	30
5.3	Топ-10 биграмм с правильной сортировкой	40
Итого		100

Технические требования

- Разрешённые библиотеки: os, re, collections.Counter, glob, zipfile.
- Кодировка всех файлов: UTF-8.

Задача 6: Обработка изображений фильтрами

Прежде чем нейросети смогут распознавать объекты на изображениях, им необходимо научиться извлекать из них полезные признаки. Фильтрация изображений — один из фундаментальных методов компьютерного зрения, позволяющий выделять границы, подавлять шумы и усиливать важные детали. Эта задача познакомит вас с базовыми операциями обработки изображений, которые лежат в основе современных систем компьютерного зрения.

Условие задачи

Дан файл `image.png`, содержащий полутоновое изображение. Необходимо последовательно применить к нему два фильтра:

1. **Гауссово размытие** с ядром 5×5 и $\sigma = 1.0$.
2. **Фильтр повышения четкости** методом “нерезкого маскирования” (unsharp masking).
3. **Инверсию изображения** для получения окончательного результата

Сохранить результаты всех этапов обработки в отдельные файлы и заархивировать их.

Требования к реализации

Этапы обработки и выходные файлы

Для проверки корректности выполнения каждого этапа необходимо сохранить промежуточные результаты:

1. `kernel.txt` — файл с вычисленным гауссовым ядром 5×5
 - Формат: 5 строк, в каждой строке 5 чисел с плавающей точкой (разделитель — пробел)
 - Точность: 6 знаков после запятой
 - Требование: сумма всех элементов ядра должна быть равна 1.0 с точностью $\pm 1e-6$
2. `blurred.png` — изображение после применения гауссова размытия
 - Формат: PNG, 8 бит на пиксель
 - Значения пикселей должны быть округлены до целых и ограничены диапазоном $[0, 255]$
3. `sharpened.png` — изображение после применения фильтра повышения четкости (до инверсии)
 - Формат: PNG, 8 бит на пиксель
 - Значения пикселей должны быть округлены до целых и ограничены диапазоном $[0, 255]$
4. `result.png` — финальное изображение после применения инверсии

- Формат: PNG, 8 бит на пиксель
- Формула инверсии: $I_{\text{final}}(x,y) = 255 - I_{\text{sharpened}}(x,y)$

Все файлы должны быть заархивированы в task6.zip.

Теоретическая часть

Гауссово размытие

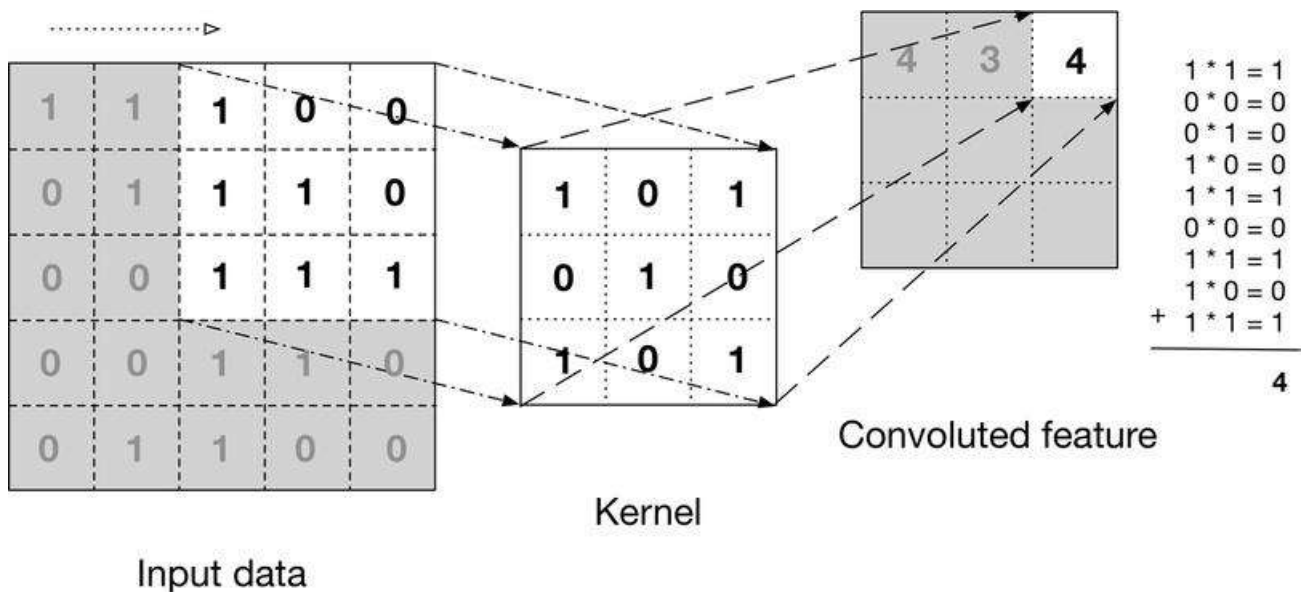
Гауссово ядро размера (k k) вычисляется по формуле:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

где (x, y) — координаты относительно центра ядра ((-2 x), (-2 y) для ядра 5×5).

Для применения фильтра выполняется операция свертки:

$$I_{\text{blurred}}(x, y) = \sum_{i=-2}^2 \sum_{j=-2}^2 I(x + i, y + j) \cdot G(i, j)$$



Пример вычисления ядра ($\sigma = 1.0$): Для центрального пикселя ($x=0, y=0$):

$$G(0,0) = \frac{1}{2\pi \cdot 1^2} e^{-\frac{0^2+0^2}{2 \cdot 1^2}} = \frac{1}{2\pi} \approx 0.159155$$

Фильтр повышения четкости (unsharp masking)

Метод основан на выделении высокочастотных компонент изображения. Алгоритм:

1. Исходное изображение: (I)
2. Размытое изображение: (B) (получено на предыдущем шаге)
3. Маска деталей: (M = I - B)
4. Усиленное изображение: ($I_{\text{sharp}} = I + M$)

где ($= 1.5$) — коэффициент усиления деталей.

Пример: Для пикселя со значением 100:

- Размытое значение: 95
- Маска деталей: $100 - 95 = 5$
- Усиленное значение: $100 + 1.5 \times 5 = 107.5 \rightarrow 108$ (после округления)

Инверсия изображения

Инверсия преобразует светлые области в темные и наоборот, что полезно для визуализации деталей:

$$I_{\text{inverted}}(x, y) = 255 - I_{\text{sharp}}(x, y)$$

Обработка границ

При применении фильтров для пикселей вблизи границ изображения используйте стратегию отражения (reflect):

$$I(x, y) = \begin{cases} I(0, y), & \text{если } x < 0 \\ I(\text{width} - 1, y), & \text{если } x \geq \text{width} \\ I(x, 0), & \text{если } y < 0 \\ I(x, \text{height} - 1), & \text{если } y \geq \text{height} \end{cases}$$

Пример отражения для изображения 3×3: Для координаты $(-1, 1) \rightarrow (0, 1)$ Для координаты $(3, 2) \rightarrow (2, 2)$

Входные данные

- Файл `image.png` в формате PNG с полутоновым изображением (8 бит на пиксель, значения 0-255)
- Размер изображения не менее 5×5 пикселей

Выходные данные

Четыре файла, заархивированных в `task6.zip`:

1. `kernel.txt` — матрица гауссова ядра 5×5 (5 строк по 5 чисел с точностью 6 знаков после запятой)
2. `blurred.png` — изображение после гауссова размытия

3. `sharpened.png` — изображение после повышения четкости
4. `result.png` — финальное изображение после инверсии

Критерии оценивания

№	Критерий	Баллы
6.1	Корректное вычисление гауссова ядра (точность, нормализация)	25
6.2	Правильная реализация свертки и обработки границ	35
6.3	Корректное применение фильтра повышения четкости и инверсии	40
	Итого	100

Технические требования

- **Разрешенные библиотеки:** `numpy`, `PIL (Pillow)`, `math`, `zipfile`
- **Точность вычислений:**
 - Гауссово ядро: не менее 6 знаков после запятой
 - Промежуточные вычисления: 32-битные числа с плавающей точкой
- **Ограничения значений:** все значения пикселей должны быть в диапазоне `[0, 255]`
- **Форматы файлов:**
 - `kernel.txt` — текстовый файл в кодировке UTF-8
 - Все изображения — PNG формат, 8 бит на пиксель
- **Структура архива:** `task2.zip` должен содержать ровно 4 файла в корневой директории

Примечания

- Гауссово ядро должно быть симметричным относительно центрального элемента
- Сумма всех элементов гауссова ядра должна быть строго равна 1.0
- При обработке границ обязательна стратегия отражения, а не заполнение нулями
- Все промежуточные результаты должны быть сохранены для проверки корректности каждого этапа
- Финальная инверсия применяется только к результату повышения четкости для создания контрастного представления деталей

Задача 7: Матричное представление свёртки с различными параметрами

Современные свёрточные нейронные сети (CNN) лежат в основе компьютерного зрения и обработки изображений. Понимание того, как операции свёртки могут быть представлены в виде матричных преобразований, является ключевым для оптимизации вычислений и глубокого понимания архитектуры нейросетей. Эта задача погружает вас в математические основы CNN, которые используются в таких системах, как ResNet, VGG и других архитектурах глубокого обучения.

Условие задачи

Дано входное изображение X размером 8×8 и три ядра свёртки разных размеров:

1. Ядро K_1 размером 2×2 со stride = 1
2. Ядро K_2 размером 3×3 со stride = 2
3. Ядро K_3 размером 4×4 со stride = 2

Для каждого ядра требуется построить соответствующую матрицу свёртки $K_{\text{mat}}^{(i)}$, которая преобразует векторизованное представление изображения $\text{vec}(X)$ в векторизованный результат свёртки $\text{vec}(Y_i)$:

$$\text{vec}(Y_i) = K_{\text{mat}}^{(i)} \cdot \text{vec}(X),$$

где Y_i — результат свёртки для i -го ядра (без padding).

Размеры выходных матриц $K_{\text{mat}}^{(i)}$ необходимо определить самостоятельно на основе параметров ядер и изображения.

Сохранить результаты в три файла и заархивировать их в task7.zip.

$$\underbrace{\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix}}_{\text{image}} * \underbrace{\begin{pmatrix} k_1 & k_2 \\ k_3 & k_4 \end{pmatrix}}_{\text{kernel}} \rightarrow \underbrace{\begin{pmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{pmatrix}}_{\text{transformed kernel}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix}}_{\text{transformed image}} = \underbrace{\begin{pmatrix} k_1x_1 + k_2x_2 + k_3x_4 + k_4x_5 \\ k_1x_2 + k_2x_3 + k_3x_5 + k_4x_6 \\ k_1x_4 + k_2x_5 + k_3x_7 + k_4x_8 \\ k_1x_5 + k_2x_6 + k_3x_8 + k_4x_9 \end{pmatrix}}_{\text{result}}$$

Теоретическая часть

Что такое свёртка?

Свёртка — фундаментальная операция в CNN, которая применяет ядро (фильтр) к локальным областям изображения для выявления признаков. Ключевые параметры:

- **Ядро (kernel)** — матрица весов, определяющая, какие признаки будут обнаружены
- **Stride** — шаг перемещения ядра по изображению
- **Padding** — добавление границ (в задаче padding = 0)

Формула размера выходного изображения

Размер выходного изображения при свёртке без padding определяется по формуле:

$$\text{output_size} = \left\lfloor \frac{\text{input_size} - \text{kernel_size}}{\text{stride}} \right\rfloor + 1$$

где $\lfloor \cdot \rfloor$ — округление вниз.

Примеры:

- Для 8×8 изображения и ядра 2×2 со stride = 1: $(8 - 2)/1 + 1 = 7$
- Для ядра 3×3 со stride = 2: $(8 - 3)/2 + 1 = 3$
- Для ядра 4×4 со stride = 2: $(8 - 4)/2 + 1 = 3$

Векторизация изображения

Векторизация в порядке **row-major** (по строкам) преобразует матрицу 8×8 в вектор длины 64:

$$X = \begin{bmatrix} x_{00} & x_{01} & \dots & x_{07} \\ x_{10} & x_{11} & \dots & x_{17} \\ \vdots & \vdots & \ddots & \vdots \\ x_{70} & x_{71} & \dots & x_{77} \end{bmatrix} \xrightarrow{\text{vec}} \text{vec}(X) = [x_{00}, x_{01}, \dots, x_{07}, x_{10}, \dots, x_{77}]^T$$

Индекс элемента $X[i, j]$ в векторе: $k = i \cdot 8 + j$.

Матрица свёртки

Матрица свёртки K_{mat} имеет размер $(\text{output_size}^2) \times 64$:

- Каждая строка соответствует одному пикселю выходного изображения
- Каждый столбец соответствует одному пикселю входного изображения
- Элемент $K_{\text{mat}}[p, q]$ содержит вес ядра, применяемый к пикселю q при вычислении пикселя p

Правила построения:

1. Определите размер выходного изображения для каждого ядра
2. Для каждой позиции выходного изображения:
 - Определите соответствующее окно во входном изображении с учетом stride
 - Разместите веса ядра в соответствующих позициях строки матрицы
3. Все остальные элементы матрицы равны нулю

Входные данные

Три файла с ядрами свёртки:

- kernel1.txt — ядро 2×2
- kernel2.txt — ядро 3×3
- kernel3.txt — ядро 4×4

Каждый файл содержит значения ядра в формате:

```
k00 k01 ... k0m
k10 k11 ... k1m
...
kn0 kn1 ... knm
```

Выходные данные

Архив task7.zip, содержащий три файла:

- result1.txt — матрица свёртки для первого ядра (2×2 , stride = 1)
- result2.txt — матрица свёртки для второго ядра (3×3 , stride = 2)
- result3.txt — матрица свёртки для третьего ядра (4×4 , stride = 2)

Формат каждого файла:

```
{row0_col0} {row0_col1} ... {row0_col63}
{row1_col0} {row1_col1} ... {row1_col63}
...
{rowM_col0} {rowM_col1} ... {rowM_col63}
```

где $M = \text{output_size}^2$.

Каждое число выводится с точностью до 3 знаков после запятой.

Пример для ядра 2×2 со $\text{stride} = 1$

Для ядра:

```
1.0 0.5
0.0 0.0
```

и изображения 8×8 , выход будет размером $7 \times 7 = 49$.

Первые две строки `result1.txt`:

```
1.000 0.500 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0
.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0
00 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000
0.000 1.000 0.500 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0
.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0
00 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
...

```

Объяснение: Первая строка соответствует вычислению верхнего левого пикселя выхода. Ядро 2×2 применяется к пикселям (0,0), (0,1), (1,0), (1,1) входного изображения. В векторизованном представлении эти пиксели имеют индексы 0, 1, 8, 9. Веса ядра 1.0 и 0.5 помещаются в позиции 0 и 1 соответственно.

Критерии оценивания

№	Критерий	Баллы
7.1	Правильный расчёт размеров выходных изображений для всех ядер	25
7.2	Корректное построение матрицы для одного ядра	25
7.3	Построение всех трёх матриц свёртки	35
7.4	Правильная обработка <code>stride</code> при построении матрицы	15
	Итого	100

Технические требования

- Разрешено использовать библиотеку `numpy`.
- Каждое число в выходных файлах должно быть выведено с точностью до 3 знаков после запятой.
- Порядок векторизации: `row-major` (по строкам).
- Все файлы должны быть в кодировке UTF-8.
- Числовой допуск при проверке: $\pm 10^{-6}$.

Задача 8: Оптимальная политика в решётке

Обучение с подкреплением формирует основу для создания автономных систем, способных принимать решения в сложных средах. От шахматных программ до систем управления роботами — все они используют принципы оценки состояний и поиска оптимальных стратегий. Алгоритм Value Iteration представляет собой фундаментальный метод динамического программирования, позволяющий агенту заранее оценивать ценность каждого состояния среды. Эта задача погружает вас в суть алгоритмов планирования, которые лежат в основе современных систем искусственного интеллекта.

Условие задачи

Дана прямоугольная сетка размером $H \times W$, где:

- S — стартовая позиция агента (ровно 1 клетка)
- T — сокровище (цель, даёт награду +10)
- X — ловушки (дают штраф -5)
- . — пустые клетки (штраф -0.1 за шаг)

Агент может выполнять действия в 4 направлениях: вверх, вниз, влево, вправо. При попытке выхода за границы сетки агент остаётся в текущей позиции.

Терминальными состояниями являются T и X, из которых дальнейшие переходы невозможны.

Требуется:

1. Реализовать алгоритм Value Iteration с коэффициентом дисконтирования $\gamma = 0.9$ для 100 итераций.
2. Вычислить ценность стартового состояния $V(S)$ после 100 итераций.
3. Подготовить три файла с результатами:
 - map.txt — координаты ключевых клеток в формате тип row col
 - v1.txt — значения функции ценности после 1-й итерации
 - result.txt — значение $V(S)$ после 100 итераций
4. Заархивировать все файлы в task8.zip.

Теоретическая часть

Основные понятия

Среда — прямоугольная сетка, где каждая клетка представляет собой состояние. Координатная система: (row, col) , где $row = 0$ соответствует верхней строке, $col = 0$ — левому столбцу.

Состояние s — текущая позиция агента (row, col).

Действие a — выбор направления движения из множества $A = \{Up, Down, Left, Right\}$.

Награда $R(s, a, s')$:

- +10 при переходе в состояние с сокровищем T
- −5 при переходе в состояние с ловушкой X
- −0.1 для всех остальных переходов (включая попытки выхода за границы)

Терминальные состояния — состояния с метками T и X. Для терминальных состояний $V(T) = V(X) = 0$, так как из них нет возможных переходов.

Функция ценности и алгоритм

Функция ценности $V(s)$ — ожидаемая суммарная дисконтированная награда при оптимальной стратегии, начинающейся из состояния s .

Уравнение Беллмана для оптимальной ценности:

$$V^*(s) = \max_{a \in A} [R(s, a) + \gamma \cdot V^*(s')]$$

Алгоритм Value Iteration:

1. **Инициализация:** $V_0(s) = 0$ для всех состояний s

2. **Итерация:** Для $k = 1$ до 100:

$$V_k(s) = \max_{a \in A} [R(s, a) + \gamma \cdot V_{k-1}(s')]$$

где s' — следующее состояние при действии a из состояния s

3. **Результат:** $V_{100}(S)$ — ценность стартового состояния после 100 итераций

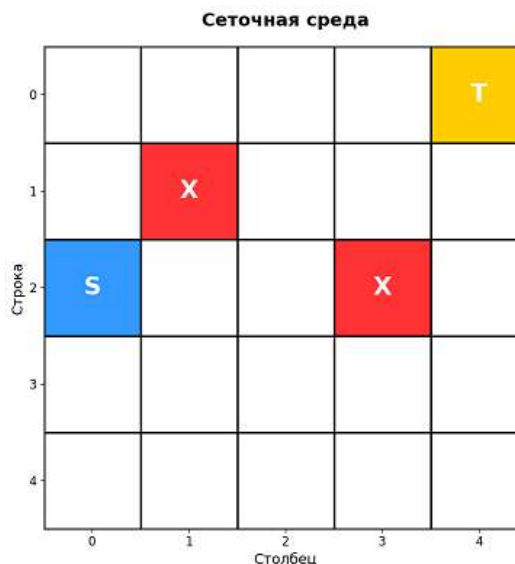
Пример

Для сетки:

....T
.X...
S..X.
.....

Координаты ключевых точек:

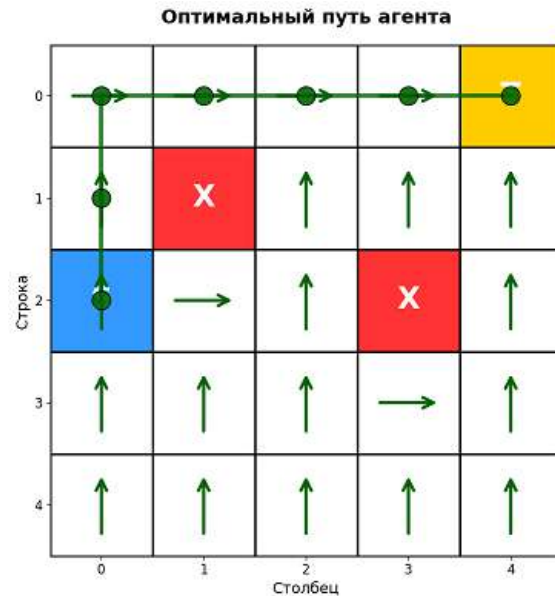
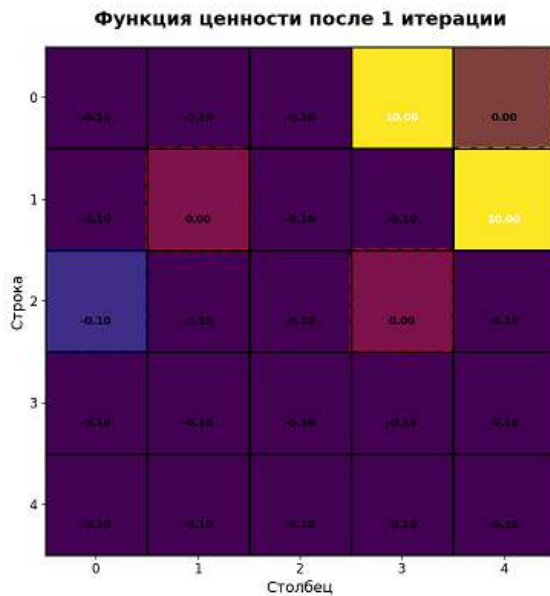
- S: (2,0)
- T: (0,4)



- $X: (1,1)$ и $(2,3)$

После 1-й итерации:

- $V(0,3) = R + \gamma \cdot V(T) = 10 + 0.9 \cdot 0 = 10.0$ (непосредственно перед сокровищем)
- $V(2,0) = \max_a [R + \gamma \cdot V(s')] = -0.1 + 0.9 \cdot 0 = -0.1$ (стартовая позиция)



Входные данные

Файл `grid.txt` в кодировке UTF-8, содержащий H строк по W символов (S, T, X, .).
Пример содержимого:

```
....T
.X...
S..X.
.....
```

Выходные данные

Архив `task8.zip`, содержащий три файла в кодировке UTF-8:

1. `map.txt`

Координаты ключевых точек в формате `row col` (каждая точка на новой строке):

```
S 2 0
T 0 4
X 1 1
X 2 3
```

2. v1.txt

Матрица значений функции ценности после 1-й итерации (H строк по W чисел с точностью 1 знак после запятой, разделённых пробелами):

```
-0.1 -0.1 -0.1 10.0 0.0  
-0.1 0.0 -0.1 -0.1 -0.1  
-0.1 -0.1 -0.1 0.0 -5.0  
-0.1 -0.1 -0.1 -0.1 -0.1
```

3. result.txt

Значение $V(S)$ после 100 итераций (6 знаков после запятой):

7.500000

Критерии оценивания

№	Критерий	Баллы
8.1	Корректный парсинг карты и определение координат	20
8.2	Корректная реализация одной итерации Value Iteration	40
8.3	Точное вычисление $V(S)$ после 100 итераций	40
Итого		100

Технические требования

- Параметры алгоритма: $\gamma = 0.9$, количество итераций = 100
- Точность вычислений: ± 0.01 для $V(S)$
- Все файлы должны быть в кодировке UTF-8
- Решение должно быть детерминированным и воспроизводимым