Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО

Лабораторная работа №1

по дисциплине

“Линейная алгебра и анализ данных”

Семестр I

Выполнили:
студент
Тищенко Павел Валерьевич
гр. J3112
ИСУ 467726

Баранов Владимир Александрович
гр. J3112
ИСУ 407921
Отчет сдан:
99.99.9999

Санкт-Петербург
2024

Введение

В ходе нашего обучения на ИИИ на предмете *"Линейная алгебра и анализ данных"* мы узнали что такое матрицы, какие существуют действия с ними и их свойства, поэтому теперь перед нами стоит задача использовать все эти знания для успешного решения данной лабораторной работы :)

Цели и задачи

Цели

Нашей большой целью данной лабораторной работы является усвоение теории из нашего учебного курса, а также применение ее на практике с помощью создания класса реализующего множество различных операций с матрицами.

Задачи

В ходе выполнения лабораторной работы необходимо реализовать следующие задачи:

1. **Разработать класс для хранения матриц в разреженно-строчном формате, поддерживающий:**
   - Подсчёт следа матрицы.
   - Получение элемента матрицы по индексу строки и столбца.
2. **Дополнить функциональность работы с матрицами, реализовав:**
   - Сложение двух матриц.
   - Умножение матрицы на скаляр.
   - Умножение двух матриц.
3. **Реализовать функцию для вычисления определителя матрицы и проверки её обратимости.**

Ход решения

Задача 1

Разреженная матрица — это матрица, в которой большинство элементов равны нулю. Для эффективного хранения и обработки таких матриц используются специальные форматы, которые сохраняют только ненулевые элементы и информацию о их позициях. Один из наиболее распространенных форматов — разреженно-строчный формат (CSR, Compressed Sparse Row), его мы и используем для реализации.

В формате CSR матрица распределена на 3 массива:

1. `values`: список всех ненулевых элементов матрицы.
2. `col_index`: список номеров столбцов для каждого ненулевого элемента из `values`.
3. `row_pointers`: список индексов в `values`, указывающих на начало каждой строки в матрице.

Логика реализации кода

Класс `SparseMatrix` реализует хранение и обработку матрицы в разреженно-строчном формате. Основные шаги реализации:

1. **Инициализация матрицы.** При создании объекта класса `SparseMatrix` передается двухмерный список (`matrix`), представляющий исходную матрицу. В процессе инициализации происходит обход каждого элемента матрицы:
   - Если элемент ненулевой, то он добавляется в список `values`.
   - Номер столбца этого элемента сохраняется в `col_index`. Индексы начала каждой строки в списке `values` сохраняются в `row_pointers`.
2. **Метод `trace`.** Вычисляем след матрицы (сумму элементов главной диагонали).
   - Для каждой строки матрицы проверяется наличие ненулевого элемента на позиции, соответствующей главной диагонали (где номер строки равен номеру столбца).
   - Если такой элемент найден, его значение прибавляется к общему следу.
3. **Метод `get_element`.** Возвращает значение элемента матрицы по заданным индексу строки и столбца.
   - Проверяем корректность введенных индексов.
   - Обходим ненулевые элементы заданной строки и ищем элемент с нужным номером столбца.
   - Если элемент найден, возвращаем его значение, иначе возвращаем 0 (что означает, что элемент нулевой).
4. **Метод `return_full_matrix`.** Восстанавливает полную матрицу из разреженного формата.
   - Создаем двухмерный список, заполненный нулями.
   - Заполняем его ненулевыми элементами, используя информацию из `values` и `col_index`.

5. **Метод** `return_csr_matrix`**.** Возвращает матрицу в виде трех основных компонентов CSR формата: `values`, `col_index`, `row_pointers`.

Преимущество данного метода заключается в том, что мы задействуем меньше памяти и ускоряем время прохода вычислений за счет хранения только ненулевых элементов.

Задача 2

Чтобы не терять эффективность разреженно-строчных матриц, необходимо также реализовать операции сложения, перемножения и умножения на скаляр матриц без приведения к полной матрице.

Операции над разреженными матрицами

1. **Сложение матриц.** Сумма двух матриц той же размерности получается путем сложения соответствующих элементов.

   где A, B — исходные матрицы, а C — результат их сложения.]

   $$C_{ij} = A_{ij} + B_{ij},$$

   где A, B — исходные матрицы, а C — результат их сложения.

2. **Умножение матрицы на скаляр.** Каждый элемент матрицы умножается на скалярное значение. В разреженных матрицах умножаются только ненулевые элементы.

   $$B_{ij} = k \cdot A_{ij}.$$

3. **Умножение матриц.** Умножение двух матриц A размерности $N \times M$ и B размерности $M \times K$ возможно, если число столбцов матрицы A совпадает с числом строк матрицы B. Элемент результирующей матрицы C вычисляется как:

   $$C_{ij} = \sum_{k=1}^{M} A_{ik} \cdot B_{kj}.$$

Логика реализации кода

Для реализации операции были написаны 3 функции для каждой операции отдельно.

1. **Функция** `sum_matrix`**.** Суммирует две матрицы одинаковой размерности.
   - Проверяем, что матрицы имеют одинаковую размерность, иначе выдаем ошибку.
   - Создаем новый объект `result_matrix` → `SparseMatrix` для хранения результата.
   - Далее обрабатываем по строкам:
     - Извлекаем ненулевые элементы обеих матриц в словари с ключами — номерами столбцов и значениями — элементами матрицы.
     - Объединяем словари, суммируя значения элементов с одинаковыми номерами столбцов.
     - Отсортировываем элементы по номерам столбцов и добавляем их в `values` и `col_index` результирующей матрицы.
     - Обновляем `row_pointers` для указания начала следующей строки.
   - Возвращаем новую матрицу в CSR формате.
2. **Функция** `multiply_scalar_matrix`**.** Умножает разреженную матрицу на скаляр.
   - Создаем новый объект `result_matrix` → `SparseMatrix` для хранения результата.

- Проходимся по списку `values` и умножаем каждый элемент на скаляр.
- Списки `col_index` и `row_pointers` копируются без изменений, так как структура ненулевых элементов не меняется.
- Возвращаем новую матрицу в CSR формате, элементы которой умножены на заданный скаляр.

3. **Функция** `multiply_matrix`**.** Перемножает две разреженные матрицы.

- Проверяем возможность операции, количество столбцов первой матрицы должно быть равно количеству строк второй матрицы.
- Создаем новый объект `result_matrix` → `SparseMatrix` для хранения результата.
- Для облегчения доступа к столбцам второй матрицы, выполняем её транспонирование, сохраняя в разреженно-строчном формате.
- Обрабатываем по строкам первой матрицы и столбцам второй (теперь строкам транспонированной второй матрицы):
  - Для каждой строки первой матрицы:
    - Извлекаем ненулевые элементы строки.
    - Для каждого ненулевого элемента строки:
      - Находим соответствующие строки в транспонированной второй матрице.
      - Вычисляем произведения и собираем результаты в словаре, где ключ — номер столбца итоговой матрицы, значение — сумма произведений.
- Отсортировываем результаты по номерам столбцов и добавляем их в `values` и `col_index` итоговой матрицы.
- Обновляем `row_pointers`.
- Результат возвращаем новую матрицу в CSR формате, предоставляющую результат умножения.
- Реализованные функции позволяют эффективно выполнять основные операции над разреженными матрицами, сохраняя преимущества CSR-формата. Использование разреженного хранения и оптимизированных алгоритмов обработки обеспечивает экономию памяти и времени вычислений, особенно при работе с большими матрицами, где большинство элементов равны нулю.

Задача 3

Определитель матрицы и её обратимость

Определитель матрицы (determinant) — это численная характеристика квадратной матрицы, которая отображает её основные свойства, такие как возможность существования обратной матрицы и степень изменения объёма при линейных преобразованиях, задаваемых матрицей.

Для квадратной матрицы размера $n \times n$, определитель вычисляется рекурсивно с использованием разложения по строке или столбцу, либо через базовые формулы для небольших размеров матриц.

Определитель матрицы A обозначается как $\det(A)$ или $|A|$.

- Если определитель матрицы равен нулю, то матрица вырожденная, и обратная матрица не существует.
- Если определитель отличен от нуля, то матрица невырожденная, и обратная матрица существует.

Логика реализации

1. **Функция** `determinant`**.** Вычисляет определитель квадратной матрицы.
   - Проверяем, что матрица является квадратной, иначе выдаём ошибку.
   - Рассматриваем базовые случаи:
     - **Матрица 1x1:**
       * Если матрица содержит ненулевой элемент, возвращается его значение.
       * Если матрица пустая (нет ненулевых элементов), возвращается 0.
     - **Матрица 2x2:**
       * Извлекаются элементы матрицы с помощью метода `get_element`.
       * Определитель вычисляется по формуле $a \cdot d - b \cdot c$.
   - **Рекурсивный случай (для квадратных матриц,** $n > 2$**):**
     - Разложение по первой строке:
       * Итерируемся по ненулевым элементам первой строки.
       * Для каждого элемента a_{1j}:
         · Вычисляем минор M_{1j}:
         · Исключаем первую строку и j-й столбец.
         · Корректируем индексы столбцов оставшихся элементов.
         · Создаём новый объект `SparseMatrix` для минора.
         · Рекурсивно вычисляем определитель минора.
         · Добавляем вклад в общий определитель по формуле:

$$\det+ = a_{1j} \cdot (-1)^{1+j} \cdot \text{determinant}(M_{1j})$$

   - Возвращаем определитель.
2. **Функция** `has_inverse`**.** Определяет, есть ли обратная матрица.
   - Вычисляет определитель матрицы с помощью функции `determinant`.

- Если определитель не равен нулю, возвращает `True`.
- Если определитель равен нулю, возвращает `False`.

Выводы

В ходе лабораторной работы реализован эффективный класс для разреженных матриц с поддержкой базовых операций. Методы работают быстрее и потребляют меньше памяти, чем аналогичные операции с полными матрицами. Операции, такие как сложение и умножение матриц, адаптированы для CSR-формата, что сохраняет его преимущества. Также, лабораторная работа показала важность разреженных форматов для оптимизации работы с матрицами в задачах линейной алгебры.

Приложение

Листинг кода файла lab1

Приложение

Листинг кода файла lab1

```
# Задача 1
class SparseMatrix:
    def __init__(self, matrix):
      """
      Инициализирует разреженную матрицу.

      :param matrix: Матрица списком.
      """
      self.rows = len(matrix)
      self.cols = len(matrix[0])

      self.values = []
      self.col_index = []
      self.row_pointers = [0]

      for i in range(len(matrix)):
        for j in range(len(matrix[i])):
          if matrix[i][j] != 0:
            self.values.append(matrix[i][j])
            self.col_index.append(j)
        self.row_pointers.append(len(self.values))

    def trace(self):
      """
      Вычисляет след матрицы (сумма элементов главной диагонали).

      :return: след матрицы.
      """
      trace = 0

      for row_index in range(len(self.row_pointers) - 1):
        start = self.row_pointers[row_index]
        end = self.row_pointers[row_index + 1]

        for i in range(start, end):
          if self.col_index[i] == row_index:
            trace += self.values[i]

      return trace

    def get_element(self, row, col):
```

```
        """
        Находит элемент матрицы по списку

        :param row: Индекс строки
        :param col: Индекс столбца

        :return: Значение элемента.
        """
        if row < 1 or col < 1 or row > self.rows or col > self.cols:
            raise IndexError("Индекс(-ы) выходят за пределы матрицы")

        row -= 1
        col -= 1

        start = self.row_pointers[row]
        end = self.row_pointers[row + 1]

        for i in range(start, end):
            if self.col_index[i] == col:
                return self.values[i]

        return 0

    def return_full_matrix(self):
        """
        Возвращает плотную матрицу в виде списка.

        :return: Матрица в виде списка.
        """
        print_matrix = [[0 for _ in range(self.cols)] for _ in range(self.rows)]

        for row_index in range(len(self.row_pointers) - 1):
            start = self.row_pointers[row_index]
            end = self.row_pointers[row_index + 1]

            for i in range(start, end):
                col = self.col_index[i]
                print_matrix[row_index][col] = self.values[i]

        return print_matrix

    def return_csr_matrix(self):
        """
        Возвращает матрицу разреженно-строчном виде.

        :return: Матрица в разреженно-строчном виде.
        """
        return self.values, self.col_index, self.row_pointers

#Задача 2: Сложение Матриц
```

```python
def sum_matrix(matrix1: SparseMatrix, matrix2: SparseMatrix) -> SparseMatrix:
    """
    Суммирует две матрицы.

    :param matrix1: Первая матрица
    :param matrix2: Вторая матрица

    :return: Сумма матриц в разреженно-строчном виде.
    """
    if matrix1.rows != matrix2.rows or matrix1.cols != matrix2.cols:
        raise ValueError("Невозможно сложить матрицы разных размеров")

    result_matrix = SparseMatrix([[0 for _ in range(matrix1.cols)] for _ in range(matrix1.ro
    result_matrix.values = []
    result_matrix.col_index = []
    result_matrix.row_pointers = [0]

    for row in range(matrix1.rows):
        row_start1 = matrix1.row_pointers[row]
        row_end1 = matrix1.row_pointers[row + 1]
        row_start2 = matrix2.row_pointers[row]
        row_end2 = matrix2.row_pointers[row + 1]

        row_values1 = {matrix1.col_index[i]: matrix1.values[i] for i in range(row_start1, row_
        row_values2 = {matrix2.col_index[i]: matrix2.values[i] for i in range(row_start2, row_

        row_result = {}
        for col, value in row_values1.items():
            row_result[col] = row_result.get(col, 0) + value
        for col, value in row_values2.items():
            row_result[col] = row_result.get(col, 0) + value

        for col, value in sorted(row_result.items()):
            result_matrix.values.append(value)
            result_matrix.col_index.append(col)

        result_matrix.row_pointers.append(len(result_matrix.values))

    return result_matrix

#Задача 2: Умножение матрицы на скаляр
def multiply_scalar_matrix(matrix: SparseMatrix, scalar: float) -> SparseMatrix:
    """
    Умножает матрицу на скаляр.

    :param matrix: Матрица
    :param scalar: Скаляр

    :return: Умноженная матрица в разреженно-строчном виде.
    """
```

```
  result = SparseMatrix([[0 for _ in range(matrix.cols)] for _ in range(matrix.rows)])
  result.values = [value * scalar for value in matrix.values]
  result.col_index = matrix.col_index[:]
  result.row_pointers = matrix.row_pointers[:]

  return result

#Задача 2: Перемножение матриц
def multiply_matrix(matrix1: SparseMatrix, matrix2: SparseMatrix) -> SparseMatrix:
  """
  Перемножает две матрицы.

  :param matrix1: Первая матрица
  :param matrix2: Вторая матрица

  :return: Результат умножения в разреженно-строчном виде.
  """
  result_matrix = SparseMatrix([[0 for _ in range(matrix2.cols)] for _ in range(matrix1.ro
  result_matrix.values = []
  result_matrix.col_index = []
  result_matrix.row_pointers = [0]

  transposed_values = []
  transposed_indices = []
  transposed_row_pointers = [0]

  for col in range(matrix2.cols):
    column_data = {}

    for row in range(matrix2.rows):
      row_start = matrix2.row_pointers[row]
      row_end = matrix2.row_pointers[row + 1]

      for i in range(row_start, row_end):
        if matrix2.col_index[i] == col:
          column_data[row] = matrix2.values[i]
    for row, value in sorted(column_data.items()):
      transposed_values.append(value)
      transposed_indices.append(row)
    transposed_row_pointers.append(len(transposed_values))

  for row in range(matrix1.rows):
    row_start1 = matrix1.row_pointers[row]
    row_end1 = matrix1.row_pointers[row + 1]

    row_result = {}
    for i in range(row_start1, row_end1):
      col1 = matrix1.col_index[i]
      value1 = matrix1.values[i]
```

```
      col_start2 = transposed_row_pointers[col1]
      col_end2 = transposed_row_pointers[col1 + 1]

      for j in range(col_start2, col_end2):
        row2 = transposed_indices[j]
        value2 = transposed_values[j]
        row_result[row2] = row_result.get(row2, 0) + value1 * value2

    for col, value in sorted(row_result.items()):
      result_matrix.values.append(value)
      result_matrix.col_index.append(col)

    result_matrix.row_pointers.append(len(result_matrix.values))

  return result_matrix


#Задача 3: Поиск определителя матрицы
def determinant(matrix: SparseMatrix) -> float:
  """
  Считает определитель матрицы. Определяет, существует ли обратная матрица заданной.

  :param matrix: Матрица

  :return: Определитель матрицы.
  """
  if matrix.rows != matrix.cols:
    raise ValueError("Невозможно сложить матрицы разных размеров")

  det = 0
  if matrix.rows == 1:
    return matrix.values[0] if matrix.values else 0
  elif matrix.rows == 2:
    a = matrix.get_element(1, 1)
    b = matrix.get_element(1, 2)
    c = matrix.get_element(2, 1)
    d = matrix.get_element(2, 2)
    return a * d - b * c
  else:
    row_start = matrix.row_pointers[0]
    row_end = matrix.row_pointers[1]

    if row_start == row_end:
      return 0

    det = 0

    for i in range(row_start, row_end):
      col = matrix.col_index[i]
      value = matrix.values[i]
```

```
        minor_values = []
        minor_index = []
        minor_pointers = [0]

        for row in range(1, matrix.rows):
          row_start = matrix.row_pointers[row]
          row_end = matrix.row_pointers[row + 1]

          row_values = []
          row_index = []
          row_pointers = [0]

          for j in range(row_start, row_end):
            if matrix.col_index[j] != col:
              row_index.append(matrix.col_index[j] if matrix.col_index[j] < col else matrix.
              row_values.append(matrix.values[j])

          minor_values.extend(row_values)
          minor_index.extend(row_index)
          minor_pointers.append(len(minor_values))

        minor_matrix = SparseMatrix([[0 for _ in range(matrix.cols - 1)] for _ in range(matr
        minor_matrix.values = minor_values
        minor_matrix.col_index = minor_index
        minor_matrix.row_pointers = minor_pointers

        det += value * (-1) ** (col % 2) * determinant(minor_matrix)


    return det

def has_inverse(matrix: SparseMatrix) -> bool:
    """
    Проверяет, существует ли обратная матрица заданной.

    :param matrix: Матрица

    :return: True, если обратная матрица существует, иначе False.
    """
    det = determinant(matrix)

    return det != 0

if __name__ == "__main__":
    task = int(input("Введите номер задачи: "))

    if task == 1:
        num_rows = int(input("Введите количество строк: "))
        num_cols = int(input("Введите количество столбцов: "))
```

```
    matrix = []
    print("Введите элементы матрицы построчно, через пробел:")
    for _ in range(num_rows):
      row = list(map(float, input().split()))
      if len(row) != num_cols:
        raise ValueError("Неверное количество элементов в строке.")
      matrix.append(row)

    sparse_matrix = SparseMatrix(matrix)

    while True:
      print("Что вы хотите сделать с матрицей:")
      print("1. Вывести матрицу")
      print("2. Найти след матрицы")
      print("3. Найти элемент матрицы по индексу")
      print("4. Выход")

      choice = int(input("Введите номер действия: "))
      if choice == 1:
        print("Ваша матрица:")
        for row in matrix:
          print(" ".join(map(str, row)))

      elif choice == 2:
        print("След матрицы:", sparse_matrix.trace())

      elif choice == 3:
        row = int(input("Введите индекс строки: "))
        col = int(input("Введите индекс столбца: "))
        print(sparse_matrix.get_element(row, col))

      elif choice == 4:
        break

      else:
        print("Неверное значение")

  elif task == 2:
    num_rows1 = int(input("Введите количество строк матрицы 1: "))
    num_cols1 = int(input("Введите количество столбцов матрицы 1: "))

    num_rows2 = int(input("Введите количество строк матрицы 2: "))
    num_cols2 = int(input("Введите количество столбцов матрицы 2: "))

    matrix1 = []
    matrix2 = []
    print("Введите элементы матрицы 1 построчно, через пробел:")
    for _ in range(num_rows1):
      row = list(map(float, input().split()))
```

```
      if len(row) != num_cols1:
        raise ValueError("Неверное количество элементов в строке.")
      matrix1.append(row)

    sparse_matrix1 = SparseMatrix(matrix1)

    print("Введите элементы матрицы 2 построчно, через пробел:")
    for _ in range(num_rows2):
      row = list(map(float, input().split()))
      if len(row) != num_cols2:
        raise ValueError("Неверное количество элементов в строке.")
      matrix2.append(row)

    sparse_matrix2 = SparseMatrix(matrix2)

    while True:
      print("Что вы хотите сделать с матрицами")
      print("1. Сложить матрицы")
      print("2. Умножить матрицу на скаляр")
      print("3. Перемножить матрицы")
      print("4. Выход")

      choice = int(input("Введите номер действия: "))
      if choice == 1:
        result = sum_matrix(sparse_matrix1, sparse_matrix2)
        print("Результат сложения матриц:")
        print("Values:", result.values)
        print("Col_index:", result.col_index)
        print("Row_pointers:", result.row_pointers)
      elif choice == 2:
        scalar = float(input("Введите скаляр: "))
        result = multiply_scalar_matrix(sparse_matrix1, scalar)
        print("Результат умножения матрицы на скаляр:")
        print("Values:", result.values)
        print("Col_index:", result.col_index)
        print("Row_pointers:", result.row_pointers)
      elif choice == 3:
        result = multiply_matrix(sparse_matrix1, sparse_matrix2)
        print("Результат перемножения матриц:")
        print("Values:", result.values)
        print("Col_index:", result.col_index)
        print("Row_pointers:", result.row_pointers)
      elif choice == 4:
        break
      else:
        print("Неверное значение")

  elif task == 3:
    num_rows = int(input("Введите количество строк: "))
    num_cols = int(input("Введите количество столбцов: "))
```

```
    matrix = []
    print("Введите элементы матрицы построчно, через пробел:")
    for _ in range(num_rows):
      row = list(map(float, input().split()))
      if len(row) != num_cols:
        raise ValueError("Неверное количество элементов в строке.")
      matrix.append(row)

    sparse_matrix = SparseMatrix(matrix)

    det = determinant(sparse_matrix)
    print(f"Определитель матрицы: {det}")
    if has_inverse(sparse_matrix):
      print("Обратная матрица: да")
    else:
      print("Обратная матрица: нет")

# Тесты для задачи 1
import unittest

class TestSparseMatrix(unittest.TestCase):
    def test_init_with_sparse_matrix(self):
      matrix = [
          [0, 0, 3],
          [4, 0, 0],
          [0, 0, 0]
      ]
      sparse_matrix = SparseMatrix(matrix)
      self.assertEqual(sparse_matrix.rows, 3)
      self.assertEqual(sparse_matrix.cols, 3)
      self.assertEqual(sparse_matrix.values, [3, 4])
      self.assertEqual(sparse_matrix.col_index, [2, 0])
      self.assertEqual(sparse_matrix.row_pointers, [0, 1, 2, 2])

    def test_init_with_full_matrix(self):
      matrix = [
          [1, 2],
          [3, 4]
      ]
      sparse_matrix = SparseMatrix(matrix)
      self.assertEqual(sparse_matrix.rows, 2)
      self.assertEqual(sparse_matrix.cols, 2)
      self.assertEqual(sparse_matrix.values, [1, 2, 3, 4])
      self.assertEqual(sparse_matrix.col_index, [0, 1, 0, 1])
      self.assertEqual(sparse_matrix.row_pointers, [0, 2, 4])

    def test_trace_with_non_zero_diagonal(self):
      matrix = [
          [1, 0],
```

```
            [0, 2]
        ]
        sparse_matrix = SparseMatrix(matrix)
        self.assertEqual(sparse_matrix.trace(), 3)

    def test_trace_with_zero_diagonal(self):
        matrix = [
            [0, 1],
            [2, 0]
        ]
        sparse_matrix = SparseMatrix(matrix)
        self.assertEqual(sparse_matrix.trace(), 0)

    def test_get_element_non_zero(self):
        matrix = [
            [0, 1],
            [2, 0]
        ]
        sparse_matrix = SparseMatrix(matrix)
        self.assertEqual(sparse_matrix.get_element(1, 2), 1)

    def test_get_element_zero(self):
        matrix = [
            [0, 1],
            [2, 0]
        ]
        sparse_matrix = SparseMatrix(matrix)
        self.assertEqual(sparse_matrix.get_element(1, 1), 0)

    def test_get_element_out_of_bounds_low(self):
        matrix = [
            [0, 1],
            [2, 0]
        ]
        sparse_matrix = SparseMatrix(matrix)
        with self.assertRaises(IndexError):
            sparse_matrix.get_element(0, 1)

    def test_get_element_out_of_bounds_high(self):
        matrix = [
            [0, 1],
            [2, 0]
        ]
        sparse_matrix = SparseMatrix(matrix)
        with self.assertRaises(IndexError):
            sparse_matrix.get_element(3, 1)

if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(TestSparseMatrix)
    unittest.TextTestRunner().run(suite)
```

```
# Тесты для задачи 2
class TestSparseMatrixFunctions(unittest.TestCase):
    def test_sum_matrix_correct(self):
      matrix1 = SparseMatrix([
          [1, 0, 2],
          [0, 3, 0],
          [4, 0, 5]
      ])
      matrix2 = SparseMatrix([
          [0, 6, 0],
          [7, 0, 8],
          [0, 9, 0]
      ])
      result = sum_matrix(matrix1, matrix2)
      expected = [
          [1, 6, 2],
          [7, 3, 8],
          [4, 9, 5]
      ]
      self.assertEqual(result.return_full_matrix(), expected)

    def test_sum_matrix_incorrect_dimensions(self):
      matrix1 = SparseMatrix([
          [1, 0],
          [0, 2]
      ])
      matrix2 = SparseMatrix([
          [3, 4, 5],
          [6, 7, 8]
      ])
      with self.assertRaises(ValueError):
          sum_matrix(matrix1, matrix2)

    def test_multiply_scalar_matrix_positive_scalar(self):
      matrix = SparseMatrix([
          [1, 0],
          [0, 2]
      ])
      scalar = 5
      result = multiply_scalar_matrix(matrix, scalar)
      expected = [
          [5, 0],
          [0, 10]
      ]
      self.assertEqual(result.return_full_matrix(), expected)

    def test_multiply_scalar_matrix_zero_scalar(self):
      matrix = SparseMatrix([
          [1, 2],
```

```
            [3, 4]
        ])
        scalar = 0
        result = multiply_scalar_matrix(matrix, scalar)
        expected = [
            [0, 0],
            [0, 0]
        ]
        self.assertEqual(result.return_full_matrix(), expected)

    def test_multiply_matrix_correct(self):
        matrix1 = SparseMatrix([
            [1, 0, 2],
            [0, 3, 0]
        ])
        matrix2 = SparseMatrix([
            [4, 5],
            [6, 7],
            [8, 9]
        ])
        result = multiply_matrix(matrix1, matrix2)
        expected = [
            [1*4 + 0*6 + 2*8, 1*5 + 0*7 + 2*9],
            [0*4 + 3*6 + 0*8, 0*5 + 3*7 + 0*9]
        ]
        self.assertEqual(result.return_full_matrix(), expected)

    def test_multiply_matrix_incorrect_dimensions(self):
        matrix1 = SparseMatrix([
            [1, 2],
            [3, 4]
        ])
        matrix2 = SparseMatrix([
            [5, 6, 7]
        ])
        with self.assertRaises(ValueError):
            multiply_matrix(matrix1, matrix2)

if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(TestSparseMatrix)
    unittest.TextTestRunner().run(suite)

# Тесты для задачи 3
class TestMatrixFunctions(unittest.TestCase):

    def test_determinant_1x1(self):
        matrix = [[5]]
        self.assertEqual(determinant(matrix), 5)

    def test_determinant_2x2(self):
```

```
        matrix = [[1, 2], [3, 4]]
        self.assertEqual(determinant(matrix), -2)

    def test_determinant_3x3(self):
        matrix = [
            [6, 1, 1],
            [4, -2, 5],
            [2, 8, 7]
        ]
        self.assertEqual(determinant(matrix), -306)

    def test_determinant_zero(self):
        matrix = [
            [1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]
        ]
        self.assertEqual(determinant(matrix), 0)

    def test_non_square_matrix(self):
        matrix = [
            [1, 2, 3],
            [4, 5, 6]
        ]
        with self.assertRaises(ValueError):
            determinant(matrix)

    def test_inverse_exists(self):
        matrix = [[1, 2], [3, 4]]
        self.assertTrue(has_inverse(matrix))

    def test_inverse_not_exists(self):
        matrix = [
            [1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]
        ]
        self.assertFalse(has_inverse(matrix))

    def test_large_matrix(self):
        matrix = [
            [2, -3, 1],
            [2, 0, -1],
            [1, 4, 5]
        ]
        self.assertEqual(determinant(matrix), 49)

if __name__ == "__main__":
  suite = unittest.TestLoader().loadTestsFromTestCase(TestSparseMatrix)
  unittest.TextTestRunner().run(suite)
```

```python
# Задача 1
class SparseMatrix:
    def __init__(self, matrix):
        """
        Инициализирует разреженную матрицу.

        :param matrix: Матрица списком.
        """
        self.rows = len(matrix)
        self.cols = len(matrix[0])

        self.values = []
        self.col_index = []
        self.row_pointers = [0]

        for i in range(len(matrix)):
            for j in range(len(matrix[i])):
                if matrix[i][j] != 0:
                    self.values.append(matrix[i][j])
                    self.col_index.append(j)
            self.row_pointers.append(len(self.values))

    def trace(self):
        """
        Вычисляет след матрицы (сумма элементов главной диагонали).

        :return: след матрицы.
        """
        trace = 0

        for row_index in range(len(self.row_pointers) - 1):
            start = self.row_pointers[row_index]
            end = self.row_pointers[row_index + 1]

            for i in range(start, end):
                if self.col_index[i] == row_index:
                    trace += self.values[i]

        return trace

    def get_element(self, row, col):
        """
        Находит элемент матрицы по списку

        :param row: Индекс строки
        :param col: Индекс столбца

        :return: Значение элемента.
        """
```

```
    if row < 1 or col < 1 or row > self.rows or col > self.cols:
      raise IndexError("Индекс(-ы) выходят за пределы матрицы")

    row -= 1
    col -= 1

    start = self.row_pointers[row]
    end = self.row_pointers[row + 1]

    for i in range(start, end):
      if self.col_index[i] == col:
        return self.values[i]

    return 0

  def return_full_matrix(self):
    """
    Возвращает плотную матрицу в виде списка.

    :return: Матрица в виде списка.
    """
    print_matrix = [[0 for _ in range(self.cols)] for _ in range(self.rows)]

    for row_index in range(len(self.row_pointers) - 1):
      start = self.row_pointers[row_index]
      end = self.row_pointers[row_index + 1]

      for i in range(start, end):
        col = self.col_index[i]
        print_matrix[row_index][col] = self.values[i]

    return print_matrix

  def return_csr_matrix(self):
    """
    Возвращает матрицу разреженно-строчном виде.

    :return: Матрица в разреженно-строчном виде.
    """
    return self.values, self.col_index, self.row_pointers
```