Библиотека классов для работы с разреженными матрицами рациональных чисел

Алексей Сальников

2018

1. Введение

Разреженные матрицы — матрицы, где большая часть элементов имеет нулевые значения. В связи с этим, несмотря на то, что матричные операции универсальны, для своей работы они требуют несколько других форматов хранения данных и других алгоритмов реализации матричных операций. В большей степени это скорее работа со списками, нежели с двумерным массивом.

В рамках данного задания требуется написать написать 3 серии программ:

1. **Генераторы матриц и векторов**. Матрицы и векторы записываются в текстовые файлы в определённом формате. Генераторы должны уметь генерировать объекты удовлетворяющие нужным свойствам.

2. **Конвертеры**. Преобразуют матрицы из разреженного представвления в бинарное для плотных матриц. Матрица записывается как матрица double-ов. И наоборот, преобразуют из плотного, в разреженное (с указанием погрешности 0). Смысл коныерторов в том, чтобы можно было посмотреть матрицу в gnuplot, matplotlib, и. т. п.

3. **Решатели**. Программы осуществляющие набор операций над матрицами. В основном предполагается, что это программы тестирующие библиотеку и иллюстрирующие её возможности.

2. Структура библиотеки

2.1. Основные классы

Библиотека должна включать в себя следующие классы:

- Класс `Rational_number` – предназначен для хранения рациональных чисел и операций с ними. Напоминаю, что в рациональных числах числитель целый, а знаменатель натуральный. В случае необходимости хранения рационального 0, считать, что в числителе написан 0, а в знаменателе 1.

  Предусмотреть метод, позволяющий привести число к несократимой дроби.

  Предполагается, что число состоит из 2-х $uint32\_t$ и одного int, который задаёт знак числа.

- Класс `Vector` – хранит разреженный вектор рациональных чисел. Рациональные числа собраны в абстрактную структуру данных словарь (dictionary), ключом в ней является координата в векторе. (С точки зрения повышения скогрости доступа осмысленно хранить в виде дерева, по структуре близкого к сбалансированному, но можно и тупо списком).

- Класс `Matrix` – хранит разреженную матрицу. Данные хранятся аналогично вектору, за исключением того, что ключом является пара координат.

- Набор классов для обработки возникающих исключительных ситуаций, имеющих место при работе с векторами, матрицами и числами. Классы должны позволять печатать осмысленные сообщения об ошибках включая возможность напечатать объекты при работе с которыми возникли ошибки. например при операции деления напечатать оба операнда.

2.2. Рациональные числа

Для чисел должны быть переопределены корректным образом все стандартные операции: `+`, `-`, `*`, `/`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `++`, `-`,`+=`, `-=`, `*=`, `/=`. Операции с рациональными числами так же должны работать, когда в качестве одного из операндов указан целочисленных базовый тип данных например `unsigned short`. Также должно быть определено преобразование в `int`, `long int`, `short` — при этом если преобразование таково, что если значение выходит за диапазон, то необходимо генерировать исключение и не производить операцию. Дробная часть при таких операциях отбрасывается. Предусмотреть приведение к `double`

Также определить методы: `round`, `floor`. Должны присутствовать оператор присваивания, а также конструкторы: по умолчанию, из 2-х чисел типа `uint32_t`, из двух строковых констант (типа `char*`), из одной строковой константы (формат: “12/5”, “-13/12345”, “12”, “-64”).

Для приведения числа к каноническому виду создать метод `make_canonical()`. Предусмотреть методы `get_number_part()` и `get_fractional_part()` возвращающие целую и дробные части числа соответственно. при этом обе части должны оставаться рациональными числами.

2.3. Вектора

Для векторов необходимо реализовать операции сложения/вычитания, умножения/деления на рациональные числа. А также умножения на матрицу. Необходимо реализовать операции сложения и вычитания двух векторов, а также унарный минус. Операцию скалярного умнодения 2-х векторов, через операцию `*`.

Необходимо переопределить `[]` для доступа к элементу вектора (в случае выхода за диапазон генерировать исключение).

Должен присутствовать конструктор, позволяющий создать вектор по имени файла. Должен присутствовать метод `write`, который записывает вектор в файл по имени файла. Необходимы конструкторы которые заполняют вектор нулями, а так же единицами. Не забыть про конструктор копирования и оператор присваивания.

2.4. Матрицы

Для матриц определить операции `+`,`-`,`*` и унарный минус. Конструкция побитового отрицания должна генерировать транспонированную матрицу. Операция `^` должна возводить матрицу в нужную степень, при этом предполагается, что это операция “in place”, то есть в памяти создаётся только одна дополнительная матрица. Операция `[]` должна быть определена для нескольких типов (`Matrix_coords`, `Matrix_row_coord`, `Matrix_column_coord`), при этом для координат возвращается элемент, а для строки и столбца вектор. Для достура к элементу матрицы, без необходимости создавать отдельный класс координат предусмотреть перегруженный оператор `()` с двумя параметрами, который позволит доступ к элементу матрицы.

Матрица должна уметь читаться и записываться в файл. Предусмотреть конструкторы которые будут создавать матрицу заданной размерности заполненную нулями, заполненную единицами. Конструктор который создаёт единичную матрицу. Не забыть про конструктор копирования и оператор присваивания. Должен присутсвовать конструктор, который создаёт матрицу из вектора. Со вторым парметром по умолчанию задающим ориентацию. По умолчанию вектор будет иметь «вертикальную» ориентацию.

2.5. Некоторые общие моменты

Для всех классов должны быть предусмотрены методы позволяющие явно задавать поля. Например в матрице должен быть метод позволяющий в позицию (i, j) положить значение. Например конструкции в `[]` позволяют только посмотреть значение, а `()` позволяют записать значение. При этом для записи строк и столбцов предусмотреть специальный класс обёртку (итератор) над вектором, который будет содержать ссылку на матрицу и помнить столбец/строку в матрице, с которой ассоциирован. По вызову метода *sync_ to()* класс обёртка будет синхронизовать свои значения со значениями в матрице. По вызову метода *sync_from()* наоборот актуализировать содержимым матрицы.

Для матриц и векторов, с точки зрения экономии памяти рационально хранить счётчик ссылок на объект и в некоторых случаях вместо копирования содержимого, изменить счётчик ссылок.

Для каждого класса описанного выше должен быть определён метод *to_ string()*, который превращает этот объект в текст (тип `char *` с переносами строк внутри неё.) выделяя при этом для данного текста память.

При реализации данного задания **запрещается пользоваться STL**.

3. Форматы файлов

Файлы для хранения векторов и матриц – текстовые. В них могут встречаться комментарии. Комментарий начинается символом *'#'*.

Файл для хранения векторов начинается со слова *vector* далее пробел, далее размерность вектора. Затем следуют координаты и числа. каждое число с координатами на своей строке. сперва в строке идёт координата в векторе, а затем после, произвольного количества пробельных символов рациональное число. В файле могут быть пустые строки. Координаты нумеруются с единицы.

Рациональное число записано так: сперва идёт числитель, вместе со знаком, затем после символа *'/'* знаменатель. В случае если число целое, то символ *'/'* и знаменатель могут отсутствовать.

В случае матриц вместо слова *vector* идёт слово *matrix*, далее число задающее число строк матрицы, затем число столбцов матрицы. Далее с новой новой строчки значения матрицы. Формат значений следующий: сперва номер строки в матрице, потом номер столбца, далее число (как было описано ранее).

4. Примеры файлов

Пример файла с вектором:

```
#
# This file describes
# sparse vector
#
vector 50000

1       100
6000    23   / 5
7       -5/3
22      44  /1
```

Пример файла с матрицей:

```
#
# This file describes
# sparse matrix
#
matrix 50000 5000

1     1    100
6000  2    23   / 5
7     1    -5/3
22    2    44  /1
```