Библиотека шаблонов классов для работы с разреженными матрицами и векторами

Алексей Сальников

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 форматов хранения данных таких как хэши и деревья, а также нестандартных алгоритмов реализации матричных операций. В большей степени это скорее работа со списками, нежели с двумерным массивом.

В рамках данного задания требуется написать написать 2 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. Решатели. Программы осуществляющие набор операций над матрицами. В основном предполагается, что это программы тестирующие библиотеку и иллюстрирующие её возможности.

Также необходимо создать библиотеку шаблонов и создать для неё документацию средством doxygen. Должны быть задокументированы классы, их поля, открытые методы (public). Документирование осуществляется путём написания специальных документирующих комментариев в коде. Имеет смысл посмотреть формат комментариев javadoc. Doxygen в документирующих комментариях допускает внутреннее представление текста в формате Markdown. К проверке задания необходимо сгенерировать каталог с документацией в формате html (Детали можно посмотреть в шаблоне файла конфигурации – Doxyfile (создаётся: doxygen -g)).

2. Структура библиотеки

2.1. Основные Классы

Библиотека должна включать в себя следующие классы:

- Шаблонный класс `Vector` – хранит разреженный вектор классов чисел. Числа собраны в абстрактную структуру данных словарь (dictionary) или упорядоченное по ключу множество: ключом в них является координата в векторе. Также хранит $\varepsilon \geq 0$ типа double, все числа модуль которых меньше данного числа считаются «нулём» и в векторе не должны быть сохранены. Значение $\varepsilon = 0$ означает, что вектор плотный и нужно хранить все значения. Тип контейнера для словаря или упорядоченного множества является параметром шаблона.

- Шаблонный класс `Matrix` – хранит разреженную матрицу. Данные хранятся аналогично вектору, за исключением того, что ключом является пара координат. Также хранит ε типа double, интерпретируемый по аналогии с ε для `Vector`.

- 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, для операции деления напечатать оба операнда.

Операции над матрицами и векторами должны быть реализованы над следующими структурами данных:

- Шаблонный 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 0, считать, что в числителе написан 0, а в знаменателе 1. Все целые числа должны в знаменателе хранить 1. Предусмотреть метод, позволяющий привести число к несократимой дроби. Числа, составляющие рациональное число, хранятся как protected поля класса типа, заданного параметром шаблона. Параметром шаблона должен быть знаковый целочисленный тип `(int8_t, int32_t и т.п.)`

- Шаблонный класс `Complex_number` с двумя параметрами – предназначен для хранения комплексных чисел в формате (type_real, type_imag). Если параметры шаблона не указаны, то специфицировать типом double, если второй параметр не указан, то он должен быть такой же как первый. Числа должны быть сравнимы на больше/меньше в соотвествии с их радиус-векторами.

- Тип данных bool.

Необходимо реализовать отдельные шаблоны для битовых векторов и матриц. Переменная такого типа определяется примерно как: `Vector<bool> v1.` В этом случае матрица и вектор должны храниться в памяти не в виде деревьев, а в виде данных, упакованных в 64-битные беззнаковые целые числа. Каждое содержит 64 элемента вектора или матрицы. Для организации внутреннего предствления нежелательно использовать `std::vector<bool>`, и запрещено использовать `std::bitset`. Операции умножения для битовых векторов и матриц понимаются как логическое «И»; операции сложения понимаются как логическое «ИЛИ». Операцию вычитания реализовывать для битовых матриц и векторов не нужно (должна возникать ошибка компиляции).

Если используется операция с рациональными и комплекными числами одновременно — результат должен получится комплексным (комплексные числа – объемлющий тип данных).

2.2. Собственным образом определённые числа: рациональные и комплексные

Для чисел должны быть переопределены корректным образом все стандартные операции: `+`, `-`, `*`, `/`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `++`, `-`,`+=`, `-=`, `*=`, `/=`. Операции с рациональными числами также должны работать, когда в качестве одного из операндов указан целочисленный базовый тип данных, например `unsigned short`. Также должно быть определено пользовательское преобразование в `int` путем отбрасывания дробной части числа. При этом, если преобразование таково что значение выходит за диапазон типа `int`, то необходимо генерировать исключение. Для рациональных чисел определить пользовательское преобразование к типу double. Для комплексных чисел операции, когда слева находится не комплексное число или не рациональное, считать недопустимыми: в этой ситуации компилятор может выдавать ошибку.

Для рациональных чисел определить методы: `round`, `floor`. Для всех чисел должны присутствовать оператор присваивания, а также конструкторы: по умолчанию, из числа типа `long int`, из строковых констант (типа `char*`), из одной строковой константы. Для приведения рационального числа к каноническому виду создать метод `make_canonical()`. Канонический вид рационального числа: несократимая дробь, в числителе целое число, в знаменателе – натуральное.

2.3. Вектора

Для векторов необходимо реализовать операции сложения/вычитания, умножения/деления на числа (комплексные, рациональные). А также умножения на матрицу. Необходимо реализовать операции сложения и вычитания двух векторов, а также унарный минус.

Для доступа к элементу вектора 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 `to_string`, который записывает вектор в строку. Необходимы конструкторы, которые заполняют вектор нулями, а также единицами. Не забыть про конструкторы копирования и перемещения, оператор присваивания и оператор перемещения.

2.4. Матрицы

Для матриц определить операции `+`,`-`,`*` и унарный минус. Операция `~` побитового отрицания должна генерировать транспонированную матрицу. Операция `[]` должна позволять получать срезы в матрице. Для этого должны быть определены несколько типов, которые могут быть аргументами:

- `Matrix_coords` – координаты, которые строятся по 4-м значениям координат: $(r_1, c_1), (r_2, c_2)$. Здесь $r_1 \leq r_2$ и $c_1 \leq c_2$, кроме особого случая, когда элементы равны -1. Если -1 находится в r_1 или c_1, то это понимается как «выбрать всё с начала матрицы до координаты r_2 или c_2». Если -1 во второй части координат, то понимается как «срез до конца матрицы». Если вторая часть параметров в конструкторе не указана, то считаем, что $r_2 = r_1$ и $c_2 = c_1$. При этом правая и нижняя координата понимаются как включительно. Срез $(1, 5), (1, 5)$ создаст как бы новую матрицу размера 1×1. Вообще, после доступа оператору [] для среза по координатам всегда создаётся как бы новая матрица, но мы всегда в ней обращаемся к элементам исходной материнской матрицы и не копируем элементы из материнской матрицы. В случае присвоения матрице из среза, должна быть физически создана новая матрица с копиями элементов исходной матрицы, входящих в срез. Для работы со срезами целесообразно переопределить оператор присваивания у матрицы и вектора. В случае операций, если один срез включает в себя другой срез, возвращать срез большего размера; если срезы не являются вкючением друг друга, операцию типа «сложить», «умножить» – не производить.

- `Matrix_row_coord` – возвращает «прокси-объект» (класс `Matrix_proxy`), который поддерживает операции доступа, такие же как для вектора, но он хранит указатель на матрицу. В случае удаления матрицы, на которую ссылается прокси-объект, указатель в прокси-объекте должен быть выставлен в `nullptr`. После этого, все операции доступа к прокси-объекту должны генерировать исключение. Прокси-объект должно быть можно присвоить вектору, в этом случае из строки матрицы, на которую указывает объект, должна быть создана копия и собственно копия записана в вектор.

- `Matrix_column_coord` – то же самое, что для строки матрицы, но только для столбца.

Доступ к элементам матрицы, который возвращает константную ссылку на элемент матрицы, должен быть реализован через оператор () с двумя аргументами.

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, оператор присваивания, конструктор перемещения, оператор перемещения.

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 с помощью переопределённого оператора []. Для этого можно использовать прокси-объекты и срезы.

Для каждого класса описанного выше должен быть определён метод `to_string()`, который превращает этот объект в текст (тип std::string с переносами строк внутри строки).

При реализации данного задания **рекомендуется пользоваться STL** в тех местах где это разумно и не запрещено условием.

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 тип элементов (complex, rational, bit). В случае complex, далее после ещё одного пробела идут два имени базовых типов (каждое может быть integer или float). Считывание должно проверять тип элементов (для complex – также базовые типы), и в случае несовместимости с типами в аргументах шаблона выдавать ошибку, что такой файл прочитать невозможно. Далее после пробела следует размерность вектора.

После чего следуют координаты и числа, составляющие собственно вектор, каждое число с координатой на своей строке. Сперва в строке идёт координата в векторе, затем, после произвольного количества пробельных символов, рациональное число. В файле могут быть пустые строки. Координаты нумеруются с единицы.

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

Для комплексных чисел, действительная и мнимая части находятся в круглых скобках и разделяются запятой.

В случае матриц вместо слова *vector* идёт слово *matrix*, тип элемента задаётся как для вектора, в конце строки вместо одного значения (число элементов в векторе) следуют через пробел два значения, задающих число строк матрицы, затем число столбцов матрицы. Далее с новой новой строки значения элементов матрицы. Формат значений следующий: сперва номер строки в матрице, потом номер столбца, далее число (как было описано ранее).

Пример файла с вектором:

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

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

Пример файла с матрицей рациональных чисел:

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

1     1   <100>
```

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

Пример файла с матрицей комплексных чисел:

```
#
# This file describes
# sparse matrix
#
matrix complex integer float 50000 5000

1      1    (100,0.201)
6000   2    (0,     2.0)
7      1    ( 2 ,   0.0 ) # spaces
22     2    ( 34,   1.1)
```

Для матрицы со значениями типа `bool` в строке в качестве значения записывается 1 в угловых скобках.

4. Баллы за задание

Звёздочкой помечены необязательные части задания. Чтобы получить ненулевой балл за задание, необходимо выполнить все его обязательные части.
При сдаче каждого пункта из таблицы необходимо предъявить тесты и doxygen-документацию по этому пункту, без них пункт не принимается. Тестироваться, помимо основной функциональности пункта, должно чтение данных из файла (и их обработка), формирование исключений с классами ошибок.

Рациональные числа	15	
Комплексные числа	10	*
Битовые матрицы	15	*
Разреженные матрицы	25	
Битовые векторы	10	*
Разреженные векторы	20	*
Срезы для битовых матриц	10	*
Срезы для матриц с рац. и компл. элементами	15	