Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет вычислительной математики и кибернетики

**Отчет по лабораторной работе**

**Упорядочивание массивов и методы поиска данных**

**Выполнил**:студент группы 8ххх

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.

Подпись

**Научный руководитель**:

Должность, уч. степень

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.

Подпись

Нижний Новгород

2011

**Содержание**

[Введение 3](#_Toc270962758)

[Постановка задачи 4](#_Toc270962759)

[Руководство пользователя 5](#_Toc270962760)

[Руководство программиста 6](#_Toc270962761)

[Описание структур данных 6](#_Toc270962762)

[Описание алгоритмов 6](#_Toc270962763)

[Описание структуры программы 6](#_Toc270962764)

[Заключение 7](#_Toc270962765)

[Литература 8](#_Toc270962766)

[Приложения 9](#_Toc270962767)

[Приложение 1 9](#_Toc270962768)

[Приложение 2 9](#_Toc270962769)

# Введение

## Методы решения систем с разреженной матрицей

Существует много понятий разреженной матрицы, основная идея в том, что в разреженной матрице "много" нулевых элементов. Обычно говорят, что матрица разрежена, если она содержит  отличных от нуля элементов. В противном случае матрица считается плотной. Типичным случаем разреженности является ограниченность числа ненулевых элементов в одной строке от 1 до k, где k\ll n. Задачи линейной алгебры с разреженными матрицами возникают во многих областях, например, при решении дифференциальных уравнений в частных производных, при решении многомерных задач локальной оптимизации. В п. 7.3 мы уже познакомились с методами решения СЛАУ с трехдиагональной матрицей, которая, являясь ленточной, относится также и к классу разреженных матриц.

Очевидно, что любую разреженную матрицу можно обрабатывать как плотную, и наоборот. При правильной реализации алгоритмов в обоих случаях будут получены правильные результаты, однако вычислительные затраты будут существенно отличаться. Поэтому приписывание матрице свойства разреженности эквивалентно утверждению о существовании алгоритма, использующего ее разреженность и делающего операции с ней эффективнее по сравнению со стандартными алгоритмами.

Многие алгоритмы, тривиальные для случая плотных матриц, в разреженном случае требуют более тщательного подхода. Во многих алгоритмах обработки разреженных матриц можно выделить два этапа: символический и численный. На символическом этапе формируется портрет результирующей матрицы (т.е. определяются места ненулевых элементов в структуре матрицы); на численном этапе определяются значения ненулевых элементов результирующей матрицы. В качестве примера типовых операций с разреженными матрицами в данной главе будет рассмотрен алгоритм умножения разреженной матрицы на плотный вектор, а также круг вопросов, возникающих при использовании метода Холецкого для решения систем линейных уравнений с разреженной матрицей.

## Хранение разреженной матрицы

Существуют различные форматы хранения разреженных матриц. Рассмотрим строчный и столбцовый форматы хранения.

### Разреженный строчный формат

Для хранения матрицы A требуется три одномерных массива:

* массив ненулевых элементов матрицы A, в котором они перечислены по строкам от первой до последней (обозначим его как values);
* массив номеров столбцов для соответствующих элементов массива values (обозначим его как cols);
* массив указателей позиций, с которых начинается описание очередной строки (обозначим его pointer). Описание k-й строки хранится в позициях с pointer[k] по (pointer[k+1]–1) массивов values и cols. Если pointer[k]=pointer[k+1], то k-я строка пустая. Если матрица A состоит из n строк, то длина массива pointer будет n+1.

Данный способ представления также является полным, и упорядоченным, поскольку элементы каждой строки хранятся в соответствии с возрастанием столбцовых индексов.

Для примера рассмотрим представление матрицы (7.40) в разреженном строчном формате:

values=(1, -1, -3, -2, 5, 4, 6, 4, -4, 2, 7, 8, -5);

cols=(1, 2, 4, 1, 2, 3, 4, 5, 1, 3, 4, 2, 5);

pointer=(1, 4, 6, 9, 12, 14).

Очевидно, что объем памяти, требуемый для хранения вектора pointer, значительно меньше, чем для хранения вектора rows. Более того, разреженный строчный формат обеспечивает эффективный доступ к строчкам матрицы; доступ к столбцам по прежнему затруднен. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают строчные операции.

Иногда бывает удобно использовано полный неупорядоченный способ хранения, при котором внутри каждой строки элементы могут храниться в произвольном порядке. Результаты многих матричных операций получаются неупорядоченными, и упорядочивание может быть весьма затратным. В то же время, многие алгоритмы для разреженных матриц не требуют, чтобы представление было упорядоченным.

После рассмотрения строчного формата хранения очевидным является и разреженный столбцовый формат. В этом случае ненулевые элементы матрицы A перечисляются в порядке их появления в столбцах матрицы, а не в строках. Все ненулевые элементы хранятся по столбцам в массиве values; индексы строк ненулевых элементов – в массиве rows; элементы массива pointer указывают на позиции, с которых начинается описание очередного столбца.

Столбцовые представления могут рассматриваться как строчные представления транспонированных матриц. Разреженный столбцовый формат обеспечивает эффективный доступ к столбцам матрицы; доступ к строкам затруднен. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают столбцовые операции.

В случае если обрабатываемая матрица симметрична, достаточно хранить лишь ее верхнюю треугольную подматрицу. При этом для хранения можно использовать любой из рассмотренных форматов. Например, симметричная матрица

A=\begin{bmatrix}
1 & -1 & 0 & -3 & 0\\
-1 & 5 & 0 & 0 & 0\\
0 & 0 & 4 & 6 & 4\\
-3 & 0 & 6 & 7 & 0\\
0 & 0 & 4 & 0 & -5
\end{bmatrix}

может быть представлена в разреженном строчном формате как

values=(1, -1, -3, 5, 4, 6, 4, 7, -5);

cols=(1, 2, 4, 2, 3, 4, 5, 4, 5);

pointer=(1, 4, 5, 8, 9, 10).

Если большинство диагональных элементов заданной симметричной матрицы отличны от нуля (например, у симметричной положительно определенной матрицы все диагональные элементы положительны), то они могут храниться в отдельном массиве BD, а разреженным форматом представляется только верхний треугольник В.

Завершая обзор форматов хранения разреженных матриц, можно еще раз отметить, что выбор способа хранения матрицы полностью определяется алгоритмами, которые планируется в дальнейшем использовать для обработки данной матрицы.

## Базовые алгоритмы обработки разреженных матриц

Реализация матричных операций является тривиальной в случае плотной или ленточной матрицы, но не столь очевидна для разреженных матриц, хранящихся в одном из экономичных форматов (здесь и далее мы будем рассматривать разреженный строчный формат хранения матрицы).

### **Умножение матрицы на вектор**

Рассмотрим операцию умножения разреженной матрицы на плотный вектор. Результатом этой операции будет заполненный вектор, а не разреженная матрица. Поэтому в алгоритмах умножения вычисления выполняются прямо, без символического этапа.

Среди алгоритмов, в которых встречается данная операция, можно отметить итерационные методы решения систем линейных уравнений. Достоинство данных методов, с вычислительной точки зрения, состоит в том, что единственная требуемая матричная операция, это повторное умножение матрицы на последовательность заполненных векторов; сама матрица не меняется.

Итак, рассмотрим умножение разреженной матрицы общего вида, хранимой в строчном формате посредством массивов values, cols, pointer на заполненный векторстолбец b, хранимый в одномерном массиве. Результатом будет новый заполненный вектор

c=Ab

также размещаемый в одномерном массиве.

Пусть n – число строк матрицы. Порядок, в котором накапливаются скалярные произведения, определяется порядком хранения элементов матрицы. Для каждой ее строки i мы находим с помощью массива индексов pointer значения первой pointer[i] и последней pointer[i+1]–1 позиций, занимаемых элементами строки i в массивах values и cols. Затем, чтобы вычислить скалярное произведение строки i и вектора b, мы просто просматриваем values и cols на отрезке от pointer[i] до pointer[i+1]–1; каждое значение, хранимое в cols[pointer[j]], есть столбцовый индекс и используется для извлечения из массива b элемента, который должен быть умножен на соответствующее число из массива values. Результат каждого умножения прибавляется к c[i].

Рассмотрим теперь параллельный алгоритм умножения разреженной матрицы на плотный вектор, матрица представлена в строчном формате. При таком способе представления данных в качестве базовой подзадачи может быть выбрана операция скалярного умножения одной строки матрицы на вектор. После завершения вычислений каждая базовая подзадача определяет один из элементов вектора результата c.

Как правило, количество элементов в одной строке разреженной матрицы размера n\times n ограничено некоторой константой k, k\ll n. Значит, в процессе умножения такой матрицы на вектор количество вычислительных операций для получения скалярного произведения примерно одинаково для всех базовых подзадач. При использовании систем с общей памятью число потоков p будет меньше числа базовых подзадач n, и мы можем объединить базовые подзадачи таким образом, чтобы каждый поток выполнял несколько таких задач, соответствующих непрерывной последовательности строк матрицы А. В этом случае по окончании вычислений каждая базовая подзадача определяет набор элементов результирующего вектора с. Распределение подзадач между потоками может быть выполнено произвольным образом.

### **Транспонирование матрицы**

Рассмотрим задачу транспонирования разреженной матрицы A. Формально матрица A^T может быть определена как

A^T[i,j]=A^T[j,i]

и простейшая реализация данной операции в "плотном" случае оказывается эффективной. Однако в случае разреженной матрицы ситуация не столь очевидна, и простейшая реализация может существенно ухудшить показатели эффективности.

Будем формировать результирующую матрицу построчно. Для этого можно брать столбцы исходной матрицы и создавать из них строки результирующей матрицы. Но операция выделения из CRS-матрицы столбца №i является трудоемкой, т.к. данные в векторе values хранятся по строкам и для выборки данных по столбцу нужно просмотреть всю матрицу, что приводит к квадратичной (от числа ненулевых элементов) трудоемкости алгоритма. Необходимо другое решение. Подробно проблема транспонирования разреженной матрицы обсуждается в книге [10], здесь же рассмотрим основные идеи описанного в [10] алгоритма.

1. Сформируем N одномерных векторов для хранения целых чисел (IntVectors), а также N векторов для хранения вещественных чисел (RealVectors). N в данном случае соответствует числу столбцов исходной матрицы.
2. В цикле просмотрим все строки исходной матрицы, для каждой строки – все ее элементы. Пусть текущий элемент находится в строке i, столбце j, его значение равно v. Тогда добавим числа i и v в j-ые вектора для хранения целых и вещественных чисел (соответственно). Тем самым в векторах мы сформируем строки транспонированной матрицы.
3. Последовательно скопируем данные из векторов в CRS структуру транспонированной матрицы (сols и values), попутно формируя массив pointer.

Рассмотрим пример (7.17).

При обходе исходной матрицы A формируются вектора IntVectors и RealVectors (число 3 расположено в строке 0 и столбце 1, следовательно, в IntVectors[1] добавляется 0, в RealVectors[1] добавляется 3 и т.д.). Далее вектора последовательно формируют структуру А^Т: обратите внимание на порядок следования элементов в массиве values матрицы А^Т и его соответствие порядку элементов в массиве RealVectors (аналогично cols и IntVectors). Для формирования pointer достаточно подсчитать количество элементов в каждом из N векторов. pointer[0] всегда равно нулю, pointer[i]=pointer[i-1] + "Количество элементов в векторе i-1".

Таким образом, алгоритм транспонирует матрицу за линейное время, что значительно лучше исходного тривиального алгоритма. Недостатком же алгоритма в том виде, как он изложен, является использование дополнительной памяти. В книге [10] приводится описание алгоритма, лишенного этого недостатка. Основная идея состоит в использовании структур данных матрицы А^Т для промежуточных результатов вычислений.

# Постановка задачи

Этот пункт содержит описание задачи, которую необходимо выполнить.

# Руководство пользователя

## Руководство по сборке

Сборка программы осуществляется с помощью Visual Studio. Основной проект matrix\_multiplication.sln каждой из версий программы находится в папке sln\matrix\_multiplication.

## Руководство по запуску

После того как проект собран в папке build должны появиться исполняемые файлы:

* checker.exe
* generator.exe
* binaryToTxt.exe
* matrix\_multiplication.exe

Для удобной работы с ними предусмотрены следующие скрипты, написанные на питоне:

# Руководство программиста

## Описание структуры программы

Программа состоит из трех частей:

1. test-version – тестовая версия с простыми хранением и умножением матриц.
2. openmp – столбцовое хранение матрицы и параллельное умножение в openmp версии.
3. tbb – столбцовое хранение матрицы и параллельное умножение в tbb версии.

Каждая часть программы состоит из следующих модулей:

* matrix – содержит классы матриц, алгоритмы работы с ними;
* matrix\_multiplication – консольное приложение позволяющее умножать матрицы и замерять время работы алгоритмов;
* generator – генератор тестов для разряженных матриц, генерирует исходные матрицы, по ним простым алгоритмом вычисляет правильный ответ;
* checker – сравнивает ответ полученный в модуле matrix\_multiplication с ответом полученным в generator;
* binaryToText – преобразует исходные бинарные матрицы в текстовый формат;
* tests – набор юнит тестов для проверки корректности алгоритмов и структур данных.

Каждая из трех частей программы разбита на следующие папки:

* sln – visual studio проекты
* src – файлы исходного кода
* include – заголовочные файлы
* build – скрипты и исполняемые файлы

## Описание структур данных

### class Matrix

поля:

vector<Element> values;

vector<int> rows;

vector<int> pointer;

int N;

методы:

Element\* operator [] (int j)

Matrix operator \* (Matrix& m)

void transpositionMatrix()

…

Класс описывает двумерную матрицу, которая хранится «одномерно» в vector<Element> vv, при этом элементы в матрицу добавляются «по столбцам», например матрица A = будет храниться так: . Изначально такой формат хранения казался более удобным для сравнения умножения разряженных матриц в столбцовом формате.

Доступ к ячейкам осуществляется следующим образом:

A[номер столбца][номер строки] = (&vv[номер столбца \*кол-во столбцов])[номер строки])

Умножение матриц выполняется тройным обычным тройным циклом по j, i, z и выполняется распараллеливание по первому циклу, для более быстрой работы.

### class MatrixCCS

поля:

vector<Element> values;

vector<int> rows;

vector<int> pointer;

int N;

методы:

void convertToMatrix(Matrix &A)

void transpositionMatrix()

void unite(const MatrixCCS &m)

MatrixCCS operator \* (const MatrixCCS &m)

…

Класс описывает разряженную матрицу, хранящуюся в столбцовом формате CCS. Структура хранения и алгоритм умножения подробно описаны во введении. В алгоритме умножения есть особенность в том, что в процессе подсчитывается оставшееся количество ненулевых элементов в текущем столбце и строке. Если в столбце или строке все оставшиеся элементы нулевые, выполняется continue. Данный вариант хорошо себя показывает на сильно разряженных матрицах, но значительно усложняет код (умножение занимает 70 строк).

## Описание алгоритмов

Описание алгоритмов, применяющихся в программе. Описание получается не путем копирования разработанного кода в текст отчета. Описание должно быть словесным, возможно с использованием псевдокода.

# Заключение

Этот пункт содержит перечисление тех результатов, которых вам удалось достигнуть.

# Литература

1. Курс Академия Intel: Intel Parallel Programming Professional (Introduction). Константин Баркалов, Владимир Воеводин, Виктор Гергель, Евгений Козинов, Александр Кудин, Валентина Кустикова, Алексей Линев, Иосиф Мееров, Алексей Сиднев, Александр Сысоев | Нижегородский государственный университет им. Н.И.Лобачевского
2. Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]

# Приложение. Фрагменты исходного кода программы