Министерство науки и высшего образования Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

ОБРАБОТКА РАЗРЯЖЕННЫХ МАТРИЦ

Методические указания по выполнению лабораторной работы
по курсу «Типы и структуры данных»

Калуга – 2019

УДК 004.62
ББК 32.972.5
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/5 от «23» января 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 7 от «28» 01 2019 г.

Председатель методической комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 4 от «5» 02 2019 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ _____ Ю.С. Белов
ассистент кафедры ИУ4-КФ _____ С.С. Гришунов

Аннотация

Методические указания к выполнению лабораторной работы по курсу «Типы и структуры данных» содержат сведения о разреженных матрицах и их хранении в памяти компьютера. Рассматриваются вопросы организации, хранения и использования разреженных матриц, различные алгоритмы сжатия разреженных матриц, практические примеры с использованием методов сжатия разреженных матриц.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Типы и структуры данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 2-го курса бакалавриата направления подготовки 09.03.04 «Программная инженерия», посвящены разреженным матрицам, занимающим особое место в разделе математического моделирования с использованием численных методов. В процессе построения задач, использующих численные методы, возникают большие системы линейных алгебраических уравнений (СЛАУ). Матрицы СЛАУ имеют ярко выраженную разреженную структуру. В данном пособии рассматриваются как виды разреженных матриц, так и способов оптимального хранения и обработки данных матриц.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является формирование практических навыков реализации хранения и обработки разреженных матриц.

Основными задачами выполнения лабораторной работы являются:

1. Познакомиться с понятиями разреженная/плотная матрица.
2. Изучить алгоритмы представления матриц в компактной форме.
3. Научиться реализовывать изученные алгоритмы средствами ООП-технологии.
4. Реализовать операцию над разреженными матрицами согласно варианту.

Результатами работы являются:

- Программа, реализующая индивидуальное задание
- Подготовленный отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

В настоящее время в связи с бурным развитием вычислительных средств широкое распространение получили информационные технологии, имеющие разнообразную теоретическую и прикладную направленность. Среди них особое место занимают системы автоматизированного проектирования (САПР), неотъемлемую часть которых составляют подсистемы математического моделирования различных физических процессов. Одним из самых перспективных направлений развития математического моделирования как составной части комплексной автоматизации сквозного цикла: проектирование – конструирование – изготовление, является широкое использование численных методов, таких как метод конечных элементов (МКЭ), метод граничных элементов (МГЭ) и некоторых других. В процессе построения дискретных аналогов краевых задач указанными методами возникают большие системы линейных, а в общем случае нелинейных, алгебраических уравнений. Нелинейные системы уравнений решаются в два этапа: на первом этапе они линеаризуются, а затем полученная система линейных уравнений решается с помощью какого-либо метода линейной алгебры. Если сходимость не достигнута, то процесс повторяется. Таким образом, каждый раз решается система линейных алгебраических уравнений (СЛАУ).

Матрицы СЛАУ, как правило, симметричны с выраженной разреженной структурой. При соответствующей нумерации узлов конечно-элементных моделей (КЭМ) эти матрицы могут иметь ленточную структуру. В связи с этим при использовании конечно-элементной технологии возникает проблема разработки эффективных алгоритмов формирования, хранения и использования разреженных матриц.

Память, используемая для хранения разреженных матриц, состоит из двух частей: основной памяти, в которой содержатся числовые значения элементов матриц, и дополнительной памяти, где хранятся указатели, индексы и другая информация, необходимая для формирования структуры матриц и обеспечивающая доступ к

числовым значениям их элементов. Способы хранения и использования данных, хранящихся в основной и дополнительной памяти, весьма разнообразны и определяются, главным образом, выбранным методом решения СЛАУ. Алгоритмы обработки разреженных матриц предусматривают действия только с ненулевыми элементами и, таким образом, количество операций будет пропорционально количеству ненулевых элементов.

Обычно говорят, что матрица *разрежена*, если количество нулевых элементов равно $n^{1+\gamma}$, где n –размерность матрицы, $\gamma < 1$.

В противном случае матрица считается *плотной*. Задачи линейной алгебры с разреженными матрицами возникают во многих областях, например, при решении дифференциальных уравнений в частных производных, при решении многомерных задач локальной оптимизации.

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.

Существуют различные форматы хранения разреженных матриц. Одни предназначены для хранения матриц специального вида (например, ленточных), другие обеспечивают работу с матрицами общего вида. Ниже рассмотрим некоторые весьма распространенные способы представления разреженных матриц.

АЛГОРИТМЫ СЖАТИЯ МАТРИЦ

Диагональная схема строения ленточных матриц

С ленточными матрицами связана простейшая и наиболее широко применяемая стратегия использования нулевых элементов матрицы. Матрицу A называют *ленточной*, если все ее ненулевые элементы заключены внутри ленты, образованной диагоналями, параллельными главной диагонали. Таким образом, $a_{ij} = 0$, если $|i - j| > \beta$, и $a_{k,k-\beta} \neq 0$, либо $a_{k,k+\beta} \neq 0$ хотя бы для одного значения k. Здесь β –*полуширина*, а $2\beta + 1$ –*ширина* ленты. *Лентой* матрицы A называется множество элементов, для которых $|i - j| \leq \beta$. Другими словами, для всякой строки i ленте принадлежат все элементы со столбцовыми индексами от $i - \beta$ до $i + \beta$, т. е. $2\beta + 1$ элементов. Это число может быть много меньше порядка матрицы.

Если матрица симметрична, то достаточно хранить ее *полуленту*. Верхняя полулента состоит из элементов, находящихся в верхней части ленты, т. е. таких, что $0 < j - i \leq \beta$; нижняя полулента состоит из элементов нижней части ленты, т. е. таких, что $0 < i - j \leq \beta$; в обоих случаях в строке β элементов.

$A =$

	1	*2*	*3*	*4*	*5*	*6*	*7*
1	1						
2		2	8	9			
3		8	3				
4		9		4	10		
5				10	5	11	12
6					11	6	
7					12		7

(a)

AN(I,J)=

		1
	0	2
0	8	3
9	0	4
0	10	5
0	11	6
12	0	7

(b)

Рис. 1. Симметричная ленточная матрица 7×7 с шириной ленты 5 (a); диагональная схема ее хранения в прямоугольном массиве $AN(I, J)$ с размерами 7×3 (b)

Диагональная схема хранения симметричной ленточной матрицы A посредством массива $AN(I,J)$ иллюстрируется примером на рис. 1. Для матрицы порядка n и полуширины ленты β массив имеет размеры $n\times(\beta+1)$. Главная диагональ хранится в последнем столбце, а нижние кодиагонали – в остальных столбцах со сдвигом на одну позицию вниз при каждом смещении влево. В нашем примере $n=7$, $\beta=2$ и для массива нужна 21 ячейка памяти. Для несимметричной матрицы A необходим массив размером $n\times(2\beta+1)$; нижняя полулента и главная диагональ хранятся прежним образом, а верхние кодиагонали – в правой части массива со сдвигом на одну позицию вверх при каждом смещении вправо. Диагональная схема удобна, если $\beta<<n$. Она является схемой прямого доступа в том смысле, что имеется простое взаимно однозначное соответствие между положением элемента в матрице A и его положением в массиве: a_{ij} хранится компонентой $A(i,j-i+\beta+1)$.

Ленточные матрицы имеют то важное свойство, что ширина ленты зависит от порядка, в каком расположены строки и столбцы. Поэтому можно искать перестановки строк и столбцов, приводящие к уменьшению ширины ленты. Малая ширина ленты означает снижение запросов к памяти; в вычислениях, связанных с матрицами, она обычно уменьшает и работу. В случае симметричной матрицы свойство симметрии будет сохранено, если для столбцов и строк используются одинаковые перестановки.

Если система линейных уравнений имеет ленточную матрицу коэффициентов и решается посредством гауссова исключения, причем главные элементы выбираются на главной диагонали, то вся арифметика ограничена лентой, вне которой никаких новых ненулевых элементов не возникает. Гауссово исключение можно проводить «на месте», поскольку для любого ненулевого элемента, если он появится, уже зарезервирована позиция в схеме хранения.

Профильная схема хранения симметричных матриц

Ленточная матрица высокого порядка может иметь широкую ленту и большое число нулей внутри ее. Для такой матрицы диагональная схема может быть очень неэкономной. Дженнингс предложил более эффективную схему хранения симметричных матриц, и она вследствие своей простоты стала весьма популярной. Называется она *профильной схемой*, или схемой переменной ленты. Для каждой строки i симметричной матрицы A положим

$$\beta_i = i - j_{\min}(i),$$

где $j_{\min}(i)$ – минимальный столбцовый индекс строки i, для которого $a_{ij} \neq 0$. Таким образом, первый ненулевой элемент строки i находится на β_i позиций левее главной диагонали. Определенная в диагональной схеме хранения полуширина ленты β есть просто $\max_i(\beta_i)$.

Оболочка матрицы A – это множество элементов a_{ij}, для которых $0 < i - j \le \beta_i$. В строке i оболочке принадлежат все элементы со столбцовыми индексами от $j_{\min}(i)$ до $i-1$, всего β_i элементов. Диагональные элементы не входят в оболочку. *Профиль* матрицы A определяется как число элементов в оболочке:

$$profile(A) = \sum_i \beta_i .$$

При использовании схемы Дженнингса все элементы оболочки, упорядоченные по строкам, хранятся, включая нули, в одномерном массиве, скажем AN. Диагональный элемент данной строки помещается в ее конец. Длина массива AN равна сумме профиля и порядка A. Необходим еще массив указателей, назовем его IA; элементы этого массива суть указатели расположения диагональных элементов в AN. Так, при $i > 1$ элементы строки i находятся в позициях от $IA(i-1)$ до $IA(i)$. Единственный элемент аи 1-й строки хранится в $AN(1)$. Элементы имеют последовательные легко вычисляемые столбцовые индексы. Например, для матрицы на рис. 1(а) профиль равен 7, а профильная схема выглядит так:

позиция	=	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
AN	=	1	2	8	3	9	0	4	10	5	11	6	12	0	7
IA	=	1	2	4	7	9	11	14							

Возможен вариант схемы Дженнингса с хранением оболочки по столбцам. Поскольку теперь столбцы матрицы сохраняют свои длины, эту схему часто называют *вертикальной*. Упомянем еще одно полезное понятие, используемое при конструировании схем хранения симметричных матриц. Рассмотрим строку i матрицы A. Будем говорить, что столбец $j, j > i$, *активен* в этой строке, если он содержит ненулевой элемент в строке i или выше ее. Пусть ω_i – число столбцов, активных в строке i. Тогда $\max_i \omega_i$ называют *волновым фронтом* или *шириной фронта* A. В примере на рис. 1 (а) столбец 4 активен в строке 3, а ширина фронта A равна 2.

Как и в случае ленточных матриц, профиль обычно меняется при перестановках строк и столбцов. Меньший профиль означает меньшую память и меньшее число операций в вычислениях, выполняемых с матрицей, поэтому алгоритмы минимизации профиля играют важную роль в технологии разреженных матриц.

Рассматривая элементы, заключенные в оболочке, можно заметить, что в гауссовом исключении значение β_i для каждой строки не изменится, если главные элементы выбирать на главной диагонали. Новые ненулевые элементы не могут возникнуть в позициях, внешних для оболочки. Они могут появиться внутри оболочки, где уже заготовлено место для них; отсюда следует, что исключение можно проводить при статической схеме хранения. Это важное свойство было исходной мотивировкой для схемы Дженнингса, но ту же схему можно с выгодой применять и в других случаях. Например, она очень хорошо приспособлена для итерационных алгоритмов, где требуется эффективность при вычислении произведений A на векторы, но сама

A не меняется. В данном контексте понятие оболочки можно очевидным образом распространить на несимметричные матрицы.

Схема переменной ленты строчно-ориентирована в том смысле, что любая строка матрицы сканируется эффективно; в то же время всякая попытка сканировать столбцы будет неэффективной. Кроме того, схема является статической, потому что включение нового элемента, лежащего вне оболочки, требует изменения всей структуры (если только не используются записи переменной длины).

Схема Кнута

Схемы хранения матриц, описанные ранее, дают прекрасные результаты во многих важных практических приложениях. Разреженная матрица A, которую можно переупорядочить так, чтобы она имела узкую ленту или малую оболочку, требует гораздо меньшей памяти, чем если бы она хранилась двумерным массивом, как это делают для плотных матриц. Хотя многие нули, находящиеся внутри ленты или оболочки, все же будут храниться и участвовать в операциях (или по крайней мере в проверках на нулевое значение), простота, свойственная этим схемам и связанному с ними программированию, может значительно перевесить это неудобство. Сравнение двух методов хранения показывает, что при использовании профильной схемы хранится меньшее число нулей; цена, которую приходится платить за это уменьшение – необходимость иметь массив IA, содержащий лишь вспомогательную информацию, и вследствие этого усложнение программы. Дополнительная память называется *накладной*.

И трудности программирования, и накладная память возрастают параллельно с усложнением схемы хранения. Высокосложные схемы требуют подлинно мастерского программирования, иначе их потенциальные преимущества будут утрачены. С этой точки зрения простая схема может быть более подходящей для задачи малого или среднего размера. С другой стороны, возможность применения высокосложной схемы может определять, разрешима ли вообще большая задача на данной машине.

Рассматриваемые далее схемы, в которых нули вообще не хранятся, либо хранятся только «некоторые» нули. Рассмотрим разреженную матрицу A; без потери общности и для большей ясности предположим, что A очень велика. Ее ненулевые элементы, которых очень мало в сравнении с числом нулей, рассеяны по всей матрице, образуя то, что называют *портретом матрицы* или шаблоном нулей-ненулей. Мы опишем прежде всего схему хранения разреженной матрицы, предложенную Кнутом. Ненулевые элементы хранятся в компактной форме и в произвольном порядке в одномерном массиве, скажем AN. Информация о положении ненулевых элементов может храниться двумя дополнительными параллельными одномерными массивами, скажем I и J; здесь для каждого ненулевого элемента содержатся его строчный и столбцовый индексы. Итак, для каждого $a_{ij} \neq 0$ в памяти находится тройка (a_{ij}, i, j) Далее, чтобы можно было легко отыскивать элементы произвольной строки или столбца матрицы, необходимы еще пара указателей для каждой тройки, а также указатели входа для строк и столбцов, сообщающие начало каждого строчного или столбцового списка. Пусть NR («next nonzero element in the same row» – «следующий ненулевой элемент той же строки») – массив, хранящий строчные указатели, а NC («next nonzero element in the same column» – «следующий ненулевой элемент того же столбца») – массив столбцовых указателей. Пять массивов AN, I, J, NR и NC имеют одинаковую длину, и их одноименные позиции соответствуют друг другу. Пусть JR и JC – массивы, содержащие указатели входа для строк и столбцов, расположенные в соответствии с порядком строк и столбцов матрицы. На рис. 2 (b) показан пример матрицы A(рис. 2(a)), хранимой согласно схеме Кнута. Если нам нужны, например, элементы 2-го столбца, то, заметив, что $JC(2) = 1$, начнем с 1-й позиции массивов AN, I и NC. Видим, что $AN(1) = 6$. И $I(1) = 1$, т. е. элемент 6 находится во 2-м столбце, 1-й строке. Поскольку $NC(1) = 3$, то затем мы переходим в позицию 3 и находим элемент 4. во 2-й строке. Так как $NC(3) = 6$, мы переходим в позицию 6 и находим элемент 2. в 4-й строке. Больше элементов во 2-м столбце нет, потому что $NC(6) = 0$.

Заметим, что $JC(3)=0$; это означает, что 3-й столбец пуст. При пользовании данной схемой элемент a_{ij} можно найти, лишь входя в список i-й строки и просматривая его до тех пор, пока не будет найден столбцовый индекс j, либо то же самое с переменой ролей строки и столбца. Обратная задача имеет очевидное решение: если задан номер позиции k элемента $AN(k)$, то его строчный и столбцовый индексы суть $I(k)$ и $J(k)$.

Схема Кнута требует пяти ячеек памяти для каждого ненулевого элемента A плюс к этому указатели входа для строк и столбцов. Вследствие большой накладной памяти такая схема весьма неэкономна. Достоинства ее в том, что в любом месте можно включить или исключить элемент, и можно эффективно сканировать строки и столбцы. Схема идеально приспособлена дня случаев, когда A строится каким-то алгоритмом, где нельзя предсказать конечное число и позиции ненулевых элементов. Наиболее важный пример такого алгоритма – гауссово исключение для матриц общего вида.

$A =$

	1	*2*	*3*	*4*
1		6		
2	9	4		7
3	5			
4		2		8

(a)

	1	*2*	*3*	*4*	*5*	*6*	*7*
AN =	6	9	4	7	5	2	8
I =	1	2	2	2	3	4	4
J =	2	1	2	4	1	2	4
NR =	0	3	4	0	0	7	0
NC =	3	5	6	7	0	0	0
JR =	1	2	5	6			
JC =	2	1	0	4			

(b)

Рис. 2. Схема Кнута для матрицы A

КРМ схема

Рейнболдт и Местеньи предложили модификацию схемы Кнута, сохраняющую ее ценные свойства, но использующую значительно меньше накладной памяти. Мы будем называть эту модификацию схемой Кнута-Рейн-болдта-Местеньи или *кольцевой КРМ-схемой*. Связные списки строк и столбцов закольцовываются, а начальные позиции списков включаются в указатели входа. Списки, ассоциированные со строками (столбцами), попарно не пересекаются и потому могут быть совместно хранимы одним массивом NR (для столбцов NC). Пример приведен на рис. 3. Эта схема более плотная по сравнению со схемой Кнута. Однако нужно уяснить, что если приходится просматривать элементы некоторой строки (или столбца), то мы не получаем никакой информации о столбцовых (строчных) индексах этих элементов. Рассмотрим теперь, как можно найти элемент a_{ij}. Сначала сканируется i-я строка и определяется множество S_i позиций элементов этой строки в массиве AN.

Подразумевается, что повторения отсутствуют, т. е. каждой паре p, q строчного и столбцового индексов соответствует самое большее одна позиция в каждом из массивов AN, NR и NC. Далее сканированием j-го столбцового списка определяется множество S_j позиций массива AN, где хранятся элементы j-го столбца. Наконец, находим $S_{ij} = S_i \cap S_j$ (символ $\cap$ означает пересечение; элемент принадлежит S_{ij} тогда и только тогда, когда он принадлежит и S_i, и S_j). S_{ij} либо пусто, либо содержит только один элемент. Если S_{ij} пусто, то $a_{ij} = 0$. Если S_{ij} содержит одно число, скажем, k, то $a_{ij} = AN(k)$. Обратная задача, т. е. по заданной позиции k в AN найти соответствующие строчный и столбцовый индексы i и j, не имеет иного решения как просмотр всей матрицы.

	1	2	3	4	5	6	7
AN =	6	9	4	7	5	2	8
NR =	1	3	4	2	5	7	6
NC =	3	5	6	7	2	1	4
JR =	1	2	5	6			
JC =	2	1	0	4			

Рис. 3. КРМ-схема для матрицы A

Разреженный строчный формат

Разреженный строчный формат – это одна из наиболее широко используемых схем хранения разреженных матриц. Эта схема предъявляет минимальные требования к памяти и в то же время оказывается очень удобной для нескольких важных операций над разреженными матрицами: сложения, умножения, перестановок строк и столбцов, транспонирования, решения линейных систем с разреженными матрицами коэффициентов как прямыми, так и итерационными методами и т.д. Значения ненулевых элементов матрицы и соответствующие столбцовые индексы хранятся в этой схеме по строкам в двух массивах; назовем их соответственно AN и JA. Используется также массив указателей (скажем, IA), отмечающих позиции массивов AN и JA, с которых начинается описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN. Рассмотрим матрицу:

$$A=\begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 1 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 5 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 1 & 0 & 0 \end{array}$$

A представляется следующим образом:

Позиция =	*1*	*2*	*3*	*4*	*5*	*6*
IA	1	4	4	6		
JA	3	4	8	6	8	
AN	1	3	5	7	1	

Описание 1-й строки A начинается с позиции $IA(1)=1$ массивов AN и JA. Поскольку описание 2-й строки начинается с $IA(2)=4$, это означает, что 1-й строке в AN и JA отвечают позиции 1, 2 и 3. Вообще в нашем примере:

$IA(1)=1$ первая строка начинается с $JA(1)$ и $AN(1)$.

$IA(2)=4$ вторая строка начинается с $JA(4)$ и $AN(4)$.

$IA(3)=4$ третья строка начинается с $JA(4)$ и $AN(4)$.

Поскольку это те же позиции, с которых начинается 2-я строка, это означает, что 2-я строка пуста.

$IA(4)=6$ это первая свободная позиция в JA и AN. Таким образом, описание 3-й строки заканчивается в позиции $6-1=5$ массивов JA и AN.

В общем случае описание r-й строки A хранится в позициях с $IA(r)$ до $IA(r+1)-1$ массивов JA и AN, за исключением равенства $IA(r+1)=IA(r)$, означающего, что r-ястрока пуста. Если матрица A имеет m строк, то IA содержит $m+1$ позиций.

Данный способ представления называют *полным,* поскольку представлена вся матрица A, и *упорядоченным,* поскольку элементы каждой строки хранятся в соответствии с возрастанием столбцовых индексов. Таким образом, это строчное представление, полное и упорядоченное.

Сжатие по Шерману

Схема Шермана используется для хранения разреженных треугольных матриц, вычисляемых в методе Гаусса. Чтобы объяснить схему, мы должны сперва определить *нижнюю треугольную матрицу* как матрицу, в которой ненулевые элементы могут стоять только в нижнем треугольнике и на главной диагонали, и *верхнюю треугольную матрицу* как матрицу с ненулевыми элементами только в верхнем треугольнике и на главной диагонали. Итак,

- нижняя треугольная матрица: $a_{ij} = 0$ при $j>i$,
- верхняя треугольная матрица: $a_{ij} = 0$ при $j<i$.

Наиболее важным их приложением является *треугольная факторизация* матрицы A:

$$A = LDU ,$$

где L — нижняя треугольная, U — верхняя треугольная, D — диагональная матрицы.

L, D и U получаются из A посредством некоторого процесса исключения, например гауссова исключения. Часто случается, что некоторые множества строк в U или L имеют сходную структуру. В частности, нередко будет так, что строки U с номерами i и j, $i < j$, имеют идентичную структуру справа от позиции j. Чтобы избежать повторной записи одинаковых последовательностей столбцовых индексов, Шерман предложил компактную схему хранения этих индексов, требующую дополнительного массива указателей. Идею схемы иллюстрирует следующий пример. Рассмотрим верхнюю треугольную матрицу:

		1	2	3	4	5	6	7	8	9	10
	1	1		11				12			13
	2		2				14			15	
	3			3				16			17
	4				4					18	
U=	5					5		19			20
	6						6		21	22	23
	7							7	24	25	
	8								8	26	
	9									9	27
	10										10

Эта матрица будет храниться так:

позиция	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
UD=	1	2	3	4	5	6	7	8	9	10								
UN=	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	
IU=	1	4	6	8	9	11	14	16	17	18	18							
JU=	3	7	10	6	9	8	9	10	8	9	10							
I=	1	4	2	5	2	6	9	10	11									

Диагональные элементы U хранятся в массиве UD, а внедиагональные элементы по строкам – в массиве UN. Массив IU содержит указатели для UN, определяемые обычным образом. Дополнительная компонента $IU(11)$ указывает первую свободную позицию в UN, в данном случае позицию 18. Столбцовые индексы, отвечающие находящимся в UN ненулевым элементам, хранятся в JU в компактной форме. Теперь между, UN и JU нет прямого соответствия. Столбцовые индексы выбираются из JU с помощью дополнительного

массива указателей IJ. Несколько строк описываются в JU одним и тем же набором столбцовых индексов. Например, строки 1, 3 и 5 имеют одинаковую структуру справа от столбца 5; строку 1 обслуживают индексы 3, 7, 10; в то же время индексы 7 и 10 обслуживают и строки 3, 5. Таким образом, $IJ(1)=1$, $IJ(3)=IJ(5)=2$. Если мы хотим выбрать из представления полную строку, скажем 1-ю, то действуем обычным образом. Численные значения элементов находятся в UN в позициях от $IU(1)=1$ до $IU(2)-1=3$. Столбцовые индексы находятся в JU, начиная с позиции $IJ(1)=1$. Поскольку уже известно, что нужны три столбцовых индекса, мы можем последовательно выбрать их из JU.

Схему хранения можно было бы сжать еще сильнее, если бы мы заметили, что строка 7 имеет столбцовые индексы 8 и 9, являющиеся одновременно первыми индексами строки 6. Обычно такое расположение индексов не используют, поскольку при небольшой экономии памяти затраты вычислительной работы здесь велики. Алгоритм исключения систематически порождает наборы строк, имеющих одинаковую структуру справа от некоторого столбца, тем самым предоставляя непосредственно информацию, необходимую для построения компактной схемы. Еще большее сжатие схемы потребовало бы дорогостоящего поиска.

РАБОТА С РАЗРЕЖЕННЫМИ МАТРИЦАМИ

Пример 1. Разреженная матрица $A(n \times m)$ хранится в разреженном строчном формате. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

Реализация на языке C++:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main() {
	int i, j, AN[16], JA[16], IA[4], k, t, p, st1, st2;
	int x = 0, BA[16], TA[16], a, b;
	int mas[4][4];
	cout << "BBedite Matrichu" << endl;
	for (i = 0; i <= 3; i++) {
		for (j = 0; j <= 3; j++)
			cin >> mas[i][j];
		cout << endl;
	}
	system("cls");

	for (i = 0; i <= 3; i++) {
		for (j = 0; j <= 3; j++)
			cout << mas[i][j] << ' ';
		cout << endl;
	}
	k = 0;
	for (i = 0; i <= 3; i++)
		for (j = 0; j <= 3; j++) {
			BA[x] = j;
			TA[x] = mas[i][j];
			x++;
			if (mas[i][j] != 0)
			{
				AN[k] = mas[i][j];
				JA[k] = j;
				k++;
			}
		}
	p = k;
	t = 1;
	cout << "AN=";
	for (i = 0; i<16; i++)
		if (AN[i]>0)
			cout << AN[i] << ' ';
	cout << endl;
	cout << "JA=";
```

```
for (i = 0; i<16; i++)
	if (AN[i]>0)
		cout << JA[i] << ' ';
cout << endl;
IA[0] = 1;
for (i = 0; i<16; i++)
	if (JA[i] <= JA[i - 1]) {
		IA[t] = i + 1;
		t++;
	}
cout << "IA=";
for (p = 0; p <= 4; p++)
	cout << IA[p] << ' ';
cout << endl;
cout << "BBedite Nomer 1 stolbcha" << endl;
cin >> st1;
cout << "BBedite Nomer 2 stolbcha" << endl;
cin >> st2;
a = st1;
for (i = 0; i<16; i++) {
	if (BA[i] == st1)
		a = i;
	if (BA[i] == st2)
		BA[i] = st1; b = TA[i]; TA[i] = TA[a]; TA[a] = b;
	BA[a] = st2;
}
cout << endl;
k = 0;
for (i = 0; i <= 3; i++)
	for (j = 0; j <= 3; j++) {
		mas[i][j] = TA[k];
		k++;
	}
k = 0;
for (i = 0; i <= 3; i++)
	for (j = 0; j <= 3; j++)
		if (mas[i][j] != 0) {
			AN[k] = mas[i][j];
			JA[k] = j;
			k++;
		}
cout << "AN=";
for (i = 0; i<16; i++)
	if (AN[i]>0)
		cout << AN[i] << ' ';
cout << endl;
cout << "JA=";
for (i = 0; i<16; i++)
	if (AN[i]>0)
		cout << JA[i] << ' ';
cout << endl;
IA[0] = 1;
for (i = 0; i<16; i++)
	if (JA[i] <= JA[i - 1]) {
		IA[t] = i + 1;
		t++;
	}
cout << "IA=";
```

```
        for (p = 0; p <= 4; p++)
                cout << IA[p] << ' ';
        cout << endl;
        system("pause");
        return 0;
}
```

Пример 2. Разреженная матрица $A(n \times m)$ хранится по схеме Шермана (сжатие по Шерману). Смоделировать операцию умножения двух матриц, хранящихся в этой форме, с получением результата в той же форме.

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>

using namespace std;

void print(int UD[], int UN[], int IU[], int JU[], int IJ[]) {
        int kolst = 0, znach = 0, j = 0; bool f2;
        for (int i = 0; i < 10; i++) {
                f2 = 0;
                kolst = IU[i + 1] - IU[i];
                if (kolst == 0)  f2 = 1;
                if (f2 == 0)  j = IJ[i];  else j = 0;
                for (int k = 0, l = 1; k < 10; k++)
                        if (k == i) cout << setw(4) << UD[k];
                        else if (k + 1 == JU[j - 1]
                             && l <= kolst && f2 == 0) {
                                cout << setw(4) << UN[znach]; znach++;
                                   j++; l++;
                        } else cout << setw(4) << 0;
                cout << endl;
        }
}

void task(int  UDA1[], int UNA1[], int IUA1[], int JUA1[], int
IJA1[], int UDB1[], int UNB1[], int IUB1[], int JUB1[], int IJB1[]) {
        int kolst, i = 0, znach, n, m, j, p = 0, s = 0, t = 0,
            kol = 0, d = 0, d1 = 0, ij;
        int kolsta = 0, znacha = 0, ja = 0;
        bool f, f2, f3 = 0, f5, f6;
        int UDC[10], UNC[29], IUC[50], JUC[50], IJC[50], ST[10];
        IUC[0] = 1;

        for (int u = 0; u<10; u++) {
                f2 = 0; kol = 0; f3 = 0; d1 = 0; if (u == 0) f3 = 1;
                kolsta = IUA1[u + 1] - IUA1[u];
                if (kolsta == 0)
                        f2 = 1;
```

```
for (int w = 0; w<10; w++) {
        s = 0; znacha = 0;
        if (f2 == 0) ja = IJA1[u]; else j = 0;
        for (int st = 0, la = 1; st<10; st++) {
                if (st == u) m = UDA1[st];
                else if (st + 1 == JUA1[ja - 1]
                   && la <= kolsta && f2 == 0) {
                      m = UNA1[znacha]; znacha++; ja++;
                      la++; }
                        else m = 0;
                znach = 0;
                for (int i = 0; i < 10; i++) {
                        kolst = IUB1[i + 1] - IUB1[i];
                        if (kolst == 0)
                                if (w == i && i == st) {
                                   n = UDB1[w]; f = 1; }
                                else if (i == st) {
                                   n = 0; f = 1; }
                                     else n = 0;
                                     else {
                                      j = IJB1[i];
                                      for (int k =
                                          0, l = 1;
                                          k<10; k++) {
                                            if (k == i)
                                            p = UDB1[k];
        else if (k + 1 == JUB1[j - 1] && l <= kolst) {
                p = UNB1[znach]; znach++; j++; l++;
        } else p = 0;
        if (k == w && i == st) {
                n = p;  f = 1;
        }
        }
        }
        if (f == 1) {
                s += n * m;  f = 0;
        }
        }
        }
        cout << setw(4) << s;
        if (u == w) UDC[u] = s; else if (s != 0) {
                UNC[t] = s; t++; kol++;
                if (f3 == 1) {
                        JUC[d] = w + 1; d++; IJC[0] = 1;
                }
                else {
                        ST[d1] = w + 1; d1++;
                }
        }
}

cout << endl;
IUC[u + 1] = IUC[u] + kol;
if (u>0) {
        f6 = 0;
        f5 = 1;
```

```
                        int r = 0;
                        while (r<d && f6 == 0) {
                                if (ST[0] == JUC[r]) {
                                        for (int q = 1; q<d1 - 1; q++)
                                                if (ST[q] != JUC[r + q])
                                                        f5 = 0;
                                        ij = r + 1;
                                        f6 = 1;
                                }
                                r++;
                        }
                        if (f5 == 0 || f6 == 0) {
                                for (int q = 0; q<d1; q++)
                                        JUC[d + q] = ST[q];
                                IJC[u] = d + 1;
                                d += d1;
                        } else if (f5 == 1 && f6 == 1)
                                IJC[u] = ij;
                }
                for (int r = 0; r<d1; r++)
                        ST[r] = 0;
        }
        IUC[10] = IUC[9];
        cout << "UDC" << endl;
        for (int r = 0; r < 10; r++)
                cout << setw(4) << UDC[r];
        cout << endl;
        cout << "UNC" << endl;
        for (int r = 0; r < t; r++)
                cout << setw(4) << UNC[r];
        cout << endl;
        cout << "IUC" << endl;
        for (int r = 0; r < 11; r++)
                cout << setw(4) << IUC[r];
        cout << endl;
        cout << "JUC" << endl;
        for (int r = 0; r < d; r++)
                cout << setw(4) << JUC[r];
        cout << endl;
        cout << "IJC" << endl;
        for (int r = 0; r < 9; r++)
                cout << setw(4) << IJC[r];
}

void main() {
        int UDA[] = { 1,2,3,4,5,6,7,8,9,10 };
        int UNA[] = { 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27};
        int IUA[] = { 1,4,6,8,9,11,14,16,17,18,18 };
        int JUA[] = { 3,7,10,6,9,8,9,10,8,9,10 };
        int IJA[] = { 1,4,2,5,2,6,9,10,11 };

        int UDB[] = { 10,9,8,7,6,5,4,3,2,1 };
        int UNB[] = { 27,15,5,8,4,30,1,9,3,7,21,19,2 };
        int IUB[] = { 1,2,5,7,9,11,12,13,14,14,14 };
        int JUB[] = { 4,8,10,6,9,7,8 };
        int IJB[] = { 4,1,4,6,2,5,3,3 };
```

```
        cout << "massiv A" << endl;
        print(UDA, UNA, IUA, JUA, IJA);
        cout << "massiv B" << endl;
        print(UDB, UNB, IUB, JUB, IJB);
        cout << "massiv C" << endl;
        task(UDA, UNA, IUA, JUA, IJA, UDB, UNB, IUB, JUB, IJB);
        cin.ignore().get();
}
```

ВАРИАНТЫ ЗАДАНИЙ

1. Разреженная матрица $A(n \times m)$ хранится в диагональной схеме. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

2. Разреженные матрицы $A(n \times m)$ и $B(n \times m)$ хранятся в диагональной схеме. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

3. Разреженная матрица $A(n \times m)$ хранится в профильной схеме. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

4. Разреженные матрицы $A(n \times m)$ и $B(n \times m)$ хранятся в профильной схеме. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

5. Разреженные матрицы $A(n \times m)$ и $B(m \times k)$ хранятся в диагональной схеме. Смоделировать операцию перемножения двух матриц с получением результата в том же формате.

6. Разреженные матрицы $A(n \times m)$ и $B(m \times k)$ хранятся в профильной схеме. Смоделировать операцию перемножения двух матриц с получением результата в том же формате.

7. Разреженная матрица $A(n \times m)$ хранится в схеме Кнута. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

8. Разреженные матрицы $A(n \times m)$ и $B(n \times m)$ хранятся в схеме Кнута. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

9. Разреженная матрица $A(n \times m)$ хранится в схеме Кнута. Смоделировать операцию перестановки двух строк матрицы с получением результата в том же формате.

10. Разреженная матрица $A(n \times m)$ хранится в схеме КРМ. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

11. Разреженные матрицы $A(n \times m)$ и $B(n \times m)$ хранятся в схеме КРМ. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

12. Разреженная матрица $A(n \times m)$ хранится в схеме КРМ. Смоделировать операцию перестановки двух строк матрицы с получением результата в том же формате.

13. Разреженные матрицы $A(n \times m)$ и $B(n \times m)$ хранятся в схеме Шермана. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

14. Разреженная матрица $A(n \times m)$ хранится в диагональной схеме. Привести матрицу к диагональному виду с получением результата в том же формате.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение понятию «разреженная матрица».
2. Раскройте понятие плотной матрицы.
3. Опишите алгоритмы представления матриц в компактной форме.
4. Перечислите основные алгоритмы сжатия матриц.
5. Опишите диагональную схему хранения ленточных матриц.
6. Опишите профильную схему хранения симметричных матриц.
7. Изложите концепцию схемы Кнута.
8. Охарактеризуйте различия схемы Кнута и КРМ-схемы.
9. Опишите разреженный строчный формат.
10. Опишите алгоритм сжатия по Шерману.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 4 занятия (8 академических часов: 7 часов на выполнение и сдачу практического задания и 1 час на подготовку отчета).

Номер варианта студента назначается индивидуально преподавателем.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), этапы выполнения работы, результаты выполнения работы, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 с. — Режим доступа: http://www.iprbookshop.ru/52186.html

2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт— Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с.— Режим доступа: http://www.iprbookshop.ru/63821.html

3. Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с.— Режим доступа: http://www.iprbookshop.ru/47275.html

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Костюкова Н.И. Графы и их применение [Электронный ресурс]/ Н.И. Костюкова. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 147 с. — Режим доступа: http://www.iprbookshop.ru/52185.html

5. Сундукова Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 с.— Режим доступа: http://www.iprbookshop.ru/57384.html

Электронные ресурсы:

6. Научная электронная библиотека http://elibrary.ru
7. Электронно-библиотечная система «ЛАНЬ» http://e.lanbook.com
8. Электронно-библиотечная система «IPRbooks» http://www.iprbookshop.ru
9. Электронно-библиотечная система «Юрайт» http://www.biblio-online.ru
10. Электронно-библиотечная система «Университетская библиотека ONLINE» http://www.biblioclub.ru