МГТУ им. Н. Э. Баумана

Отчет по лабораторной работе №3 по предмету «Типы и структуры данных»
«Обработка разреженных матриц»

Выполнила Параскун София,
ИУ7-34Б
Проверила
Никульшина Т. А.

Москва, 2020 г.

1. Описание условия задачи

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов:
- вектор A содержит значения ненулевых элементов;
- вектор IA содержит номера строк для элементов вектора A;
- связный список JA, в элементе Nk которого находится номер компонент в A и IA, с которых начинается описание столбца Nk матрицы A.

1. Смоделировать операцию умножения вектора-строки и матрицы, хранящихся в этой форме, с получением результата в той же форме.
2. Произвести операцию умножения, применяя стандартный алгоритм работы с матрицами.
3. Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц.

2. Описание ТЗ, включающего внешнюю спецификацию

а) описание исходных данных и результатов

Основная программа получает на вход выбранный пользователем пункт меню (целое число).

В соответствии с ним формат ввода исходных данных меняется. Сами из себя они представляют вектор-строку и матрицу, а также их размеры.

В случае ввода с консоли, у пользователя запрашивается размерность данных, после способ ввода — все значения или координатный метод. В первом случае пользователь вводит все элементы, во втором — количество ненулевых и соответствующие элементы в формате «столбец элемент».

В случае чтения из файла, данные берутся из файла «input.txt», находящегося в директории проекта. При таком способе ввода допускаем, что все числа являются целыми. Внутри файла данные находятся в стандартном формате.

Также предусмотрена генерация исходных данных с последующим выводом на экран.

Результат представляет из себя вектор-строку, который также хранится в разреженном формате.

b) Способ обращения к программе

Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 5 пунктов:
1 - Умножение разреженной матрицы(ввод с консоли),
2 - Умножение рандомной разреженной матрицы(с вводом процента нулевых элементов),
3 - Умножение разреженной матрицы(ввод из файла),
4 - Результаты эффективности,
0 - Выход из программы.

Все вводы и выводы также осуществляются в консоли.

c) Описание возможных аварийных ситуаций и ошибок пользователя

№ ситуации	Аварийная ситуация	Реакция программы
1	Ввод некорректных размеров вектора-строки или матрицы (в том числе если кол-во столбцов вектора не равен кол-ву строк матрицы)	Сообщение «Wrong values!», возврат в главное меню
2	Ввод некорректного целого элемента	Сообщение «Wrong values! Try again.», повторение ввода
3	При вводе/получении результата значения всех элементов равны 0	Сообщение «All elements equal zero!», возврат в главное меню
4	Ввод в главном меню не целого числа или целого числа, не соответствующего ни одному пункту	Сообщение «Its end of working the program!», выход из программы

3. Описание внутренних СД

Данные хранятся без специальных структур, нижепредставленные элементы являются динамическими массивами. Память из-под них высвобождается по ненадобности.

Исходные данные:
V – значения ненулевых элементов вектор-строки,
JV – номера элементов в V, с которых начинается описание столбца,
A – значения ненулевых элементов в матрице,
IA – номера строк для элементов A,
JA – номера элементов в A, с которых начинается описание столбца матрицы.

Результат:
R – значения ненулевых элементов результирующей вектор-строки,
JR – номера элементов в R, с которых начинается описание столбца.

4. Описание алгоритма

Умножение разреженных вектор-строки и матрицы происходит по следующему алгоритму.

Осуществляется проход по всем значениям массива JA кроме последнего. За каждый такой проход формируется элемент результирующей вектор-строки. Так как в JA хранятся индексы элементов от 0 до кол-ва ненулевых матрицы - 1, мы проходим от значения текущего до следующего JA. В соответствии с этим индексом, мы находим строку, которой он принадлежит и элемент из вектор-строки с соответствующим номером столбца. Перемножаем только что найденный элемент и тот, с которым мы работали. Полученную сумму мы прибавляем к результирующему вектору с индексом элемента JA.

В случае реализации стандартного умножения, осуществляется проход по столбцам вектор-строки с перемножением текущего элемента на каждый элемент соответствующего столбца матрицы, результаты по каждому элементу строки суммируются и помещаются в соответствующий элемент результирующей строки.

5. Описание используемых функций

void classic_mult() - реализует пункт меню, отвечающий за умножение данных, введенных пользователем;

void random_mult() - реализует пункт меню, отвечающий за умножение данных, сгенерированных автоматически;

void file_mult() - реализует пункт меню, отвечающий за умножение данных, полученных из файла;

void efficiency_table() - реализует пункт меню, отвечающий за вывод результатов эффективности разреженного формата хранения на разных размерах и заполненности матриц;

*void sparse_mult(int *V, int *JV, int n, int *A, int *IA, int *JA, int *non_zero_res, int *res)* – осуществляет умножение вектор-строки на матрицу, хранящихся в разреженном формате. Здесь V, JV, A, IA, JA – соответствующие элементы

хранения, n – количество столбцов матрицы, non_zero_res – количество ненулевых элементов результирующей вектор-строки, res – результирующая вектор-строка.

*void input_vector(int len, int *vec, int *vec_non_zero)* – производит ввод вектор-строки в выбранном пользователем формате. Здесь len – длина строки, vec – строка, vec_non_zero – количество ненулевых элементов строки.

void input_matrix(int m, int n, int *matrix, int *matrix_non_zero) – производит ввод вектор-строки в выбранном пользователем формате. Здесь m, n – размеры матрицы, matrix – матрицы, matrix_non_zero – количество ненулевых элементов матрицы.

void generate_vector(int len, int non_zero, int *vec) – генерирует значения вектора. Здесь len – длина строки, non_zero – количество ненулевых элементов строки, vec – строка.

void generate_matrix(int m, int n, int non_zero, int *matrix) – генерирует значения вектора. Здесь m, n – размеры матрицы, non_zero – количество ненулевых элементов матрица, matrix – матрица.

*void multiplication(int *vec, int m, int n, int *matrix, int *res)* – умножает матрицы, хранящие в стандартном формате. Здесь vec – строка, m, n – размеры матрицы (m соответствует длине строки, n – длине результирующей строки), matrix – матрица, res – результирующая строка.

6. Результаты эффективности

Результат сравнения эффективности на разных размерностях и разном проценте разреженности приведены ниже. Для усреднения времени производится 20 прогонов.

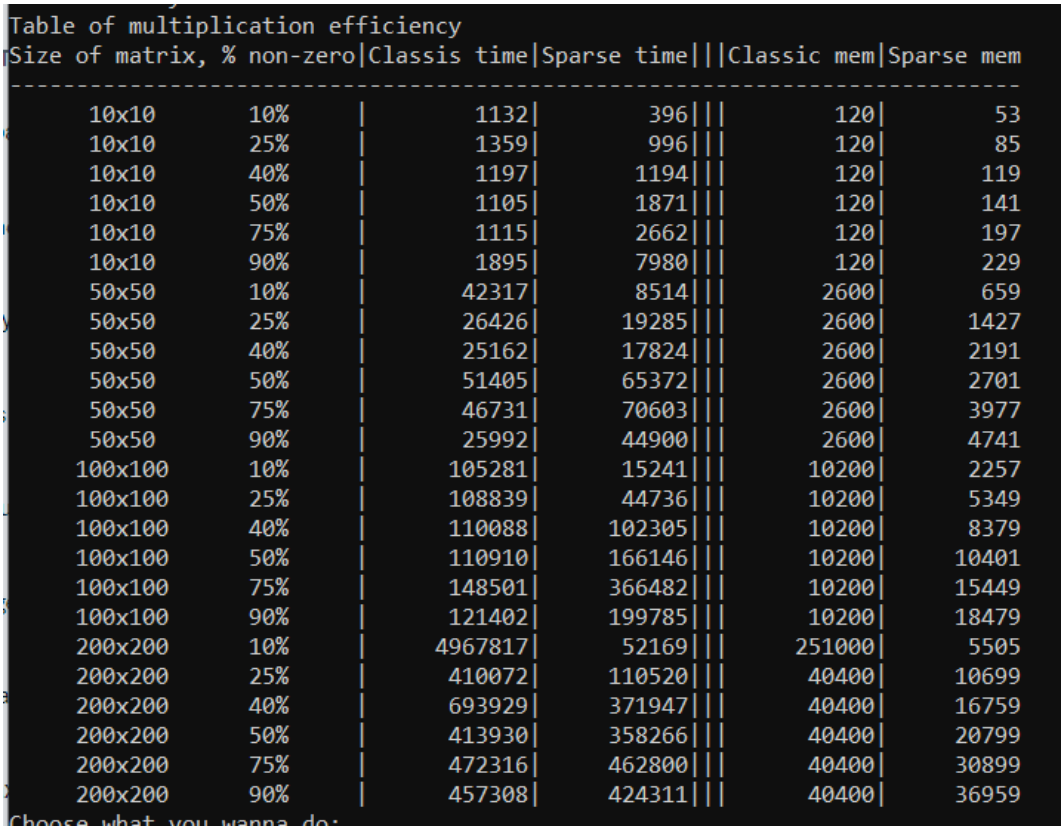

Table of multiplication efficiency

Size of matrix	% non-zero	Classis time	Sparse time	Classic mem	Sparse mem
10x10	10%	1132	396	120	53
10x10	25%	1359	996	120	85
10x10	40%	1197	1194	120	119
10x10	50%	1105	1871	120	141
10x10	75%	1115	2662	120	197
10x10	90%	1895	7980	120	229
50x50	10%	42317	8514	2600	659
50x50	25%	26426	19285	2600	1427
50x50	40%	25162	17824	2600	2191
50x50	50%	51405	65372	2600	2701
50x50	75%	46731	70603	2600	3977
50x50	90%	25992	44900	2600	4741
100x100	10%	105281	15241	10200	2257
100x100	25%	108839	44736	10200	5349
100x100	40%	110088	102305	10200	8379
100x100	50%	110910	166146	10200	10401
100x100	75%	148501	366482	10200	15449
100x100	90%	121402	199785	10200	18479
200x200	10%	4967817	52169	251000	5505
200x200	25%	410072	110520	40400	10699
200x200	40%	693929	371947	40400	16759
200x200	50%	413930	358266	40400	20799
200x200	75%	472316	462800	40400	30899
200x200	90%	457308	424311	40400	36959

Choose what you wanna do:

7. Выводы по проделанной работе

В ходе этой лабораторной удалось поработать с разреженными матрицами, а также реализовать алгоритмы их обработки.

Как мы видим, разреженный формат обеспечивает выигрыш по времени примерно до 40% ненулевых элементов. Из этого можно сделать вывод, что для достаточно больших матриц высокой разреженности алгоритм можно назвать крайне эффективным.

Очевидно, что по памяти же разреженный формат хранения выигрывает лишь до 45% ненулевых элементов, для сравнительно небольших матриц это не станет проблемой.

В целом, можно сделать вывод, что для достаточно больших и плотных матриц целесообразнее использовать стандартный алгоритм умножения, так как он создаст меньше потерь по времени и памяти. Однако, если количество ненулевых элементов в районе 40-45% и менее, разреженный формат хранения обеспечит более эффективную работу программы.

Ответы на контрольные вопросы

1. Что такое разреженная матрица, какие схемы хранения таких матриц Вы знаете?

Разреженная матрица — матрица с преимущественно нулевыми элементами. Способы хранения: сжатое хранение строкой (CSR – compressed sparse row, CRS – compressed row storage, Йельский формат), сжатое хранение столбцом (CSC – compressed sparse column, CCS – compressed column storage).

2. Каким образом и сколько по памяти выделяется под хранение разреженной и обычной матрицы?

Под обычную матрицу выделяется M * N ячеек памяти, где M – строки, N – столбцы. Для разреженной матрицы — зависит от способа хранения. В данной лабораторной потребовалось N + 1 + 2 * Matrix_non_Zero ячеек памяти, N – количество столбцов, Matrix_non_Zero – количество ненулевых элементов матрицы.

3. Каков принцип обработки разреженной матрицы?

Алгоритм обработки разреженной матрицы предусматривает действие только с ненулевыми элементами.

4. В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?

Исходя из экспериментальных данных, алгоритм рационально применять на больших матрицах с высокой разреженностью.