**Отчёт лабораторной работы №3**

по курсу «Типы и структуры данных»

Отчёт выполнила:

Кондрашова Ольга

Группа ИУ7-35Б

Вариант 9

**Цель работы**:

Цель работы - реализовать алгоритмы обработки разреженных матриц, сравнить эффективность использования этих алгоритмов (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

**Задание 4**:

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 матрицы и вектора-столбца, 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.

**Входные данные:**

Файл с записанной заранее матрицей.

Номер пункта меню.

1 – для работы с матрицей и вектора.  
2 – для вывода оценки эффективности.

**Выходные данные:**

Матрица в стандартном или разреженном виде. Таблицы с результатами измерений времени и памяти при различных используемых форматах хранения и алгоритмах обработки матриц в их различном процентном заполнении нулями.

**Обращение к программе:**Через консоль

**Внутренняя структура данных:**

Данные считываются из файла или вводятся вручную. Создается матрица указанного размера (dim), затем по матрице создаются массивы A, IA и JA. При умножении создаются аналогичные массивы для вектора-строки и результирующего вектора-строки.

Создание матрицы

int \*\* matrix;

Создание массивов с данными

not\_zero – количество ненулевых элементов.

int \*A = (int \*)malloc(not\_zero \* sizeof(int));

int \*IA = (int \*)malloc(not\_zero \* sizeof(int));

int \*JA = (int \*)malloc((dim + 1) \* sizeof(int));

**Описание задачи, реализуемой программой**

1. Работа с данными: добавление записей в таблицу (из файла или из стандартного потока ввода) и удаление записей.
2. Вывод на экран основной таблицы и таблицы ключей.
3. Сортировка данных в таблице с использованием таблицы ключей.
4. Сравнение эффективности различных методов сортировки таблиц по времени и по памяти.

**Алгоритм**

1. Пока не выбран вариант создания матрицы:

1. Ввод матрицы вручную

2. Считывание матрицы из файла

2. Создание массивов A, IA, JA

3. Пока не выбран 0:

1. Если матрицы не нулевая

mult\_mtr\_raz(A, IA, JA, count,size); // умножение на вектор-строку с разреженной форме

get\_string( size); // получение вектора-строки

2. Если матрица не нулевая

mult\_mtr(matrix,size); // умножение матрицы обычным способом

get\_string(size); // получение вектора-строки

3: efficient()// вывод таблицы сравнения времени и памяти

4: если матрица не нулевая

print\_mtr\_raz(A,IA,JA,count,size); // вывод матрицы в разреженной форме

5: print\_mtr(matrix, size); вывод матрицы в порстой форме

**Функции**

int\* getColumn(int dim, int not\_zero);

Вход: размер матрицs, количество ненулевых элементов.

Выход: вектор-столбец.

int multiplyMatrix(int \*\*matr, int dim, int not\_zero);

Вход: матрица, размер матрицs, количество ненулевых элементов.

Выход: 0, если умножение прошло успешно; 1, иначе.

int multiplySparseMatrix(int dim, int not\_zero, int \*A, int \*IA, int \*JA);

Вход: размер матрицs, количество ненулевых элементов, вектора A, IA, JA.

Выход: 0, если умножение прошло успешно; 1, иначе.

void convertIntoSparseMatrix(int \*\*matr, int dim, int not\_zero, int \*A, int \*IA, int \*JA);

Вход: матрица, размер матрицs, количество ненулевых элементов, вектора A, IA, JA.

Выход: матрица в разреженном виде.

void freeMatrix(int \*\*matr, int dim);

Вход: матрица, размер матрицs.

Выход: освобождение памяти.

int\*\* manualInput(int \*dim, int \*not\_zero);

Вход: размер матрицы, количество ненулевых элементов.

Выход: матрица.

int\*\* fileInput(int \*dim, int \*not\_zero);

Вход: размер матрицы, количество ненулевых элементов.

Выход: матрица.

void printMatrix(int \*\*matr, int dim);

Вход: матрица, размер матрицы.

Выход: вывод на экран матрицы в стандартном виде.

void printSparseMatrix(int dim, int not\_zero, int \*A, int \*IA, int \*JA);

Вход: размер матрицs, количество ненулевых элементов, вектора A, IA, JA.

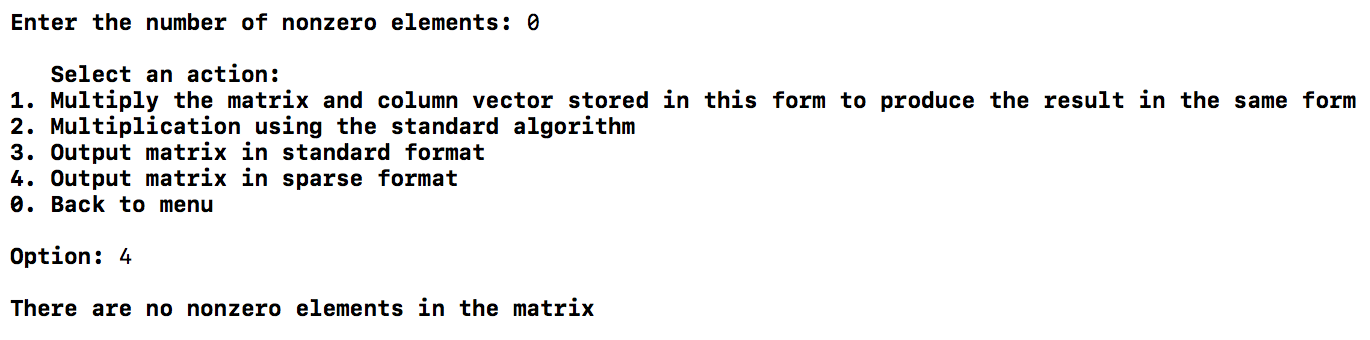
Выход: вывод на экран матрицы в разреженном виде.

**Аварийные ситуации и ошибки пользователя**

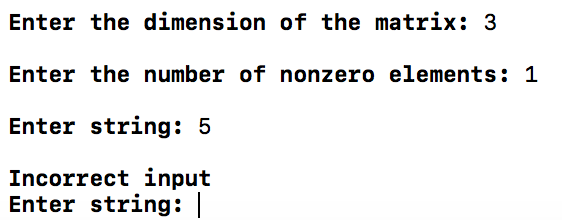
|  |  |
| --- | --- |
| Ошибка | Вывод |
| Ошибка ввода опции меню | “Unknown parameter ” |
| Ввод строки вместо числа | “Input error” |
| Ввод числа, выходящего за указанные пределы | “Incorrect size” |
| В результате опериции получилась нулевая матрица | “Error. Zero matrix” |

**Тесты**

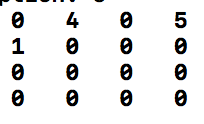
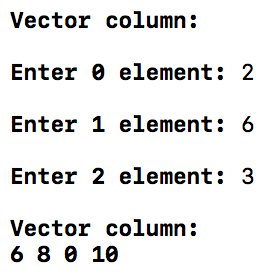
1. Вывод нулевой матрицы в разреженной форме.



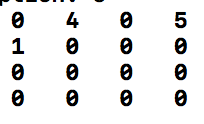
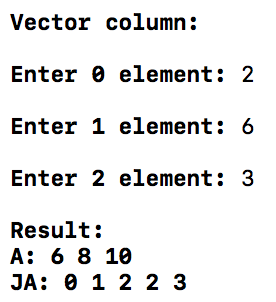
1. Ввод числа (индекса элемента), выходящего за пределы.



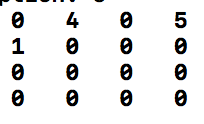
1. Умножение в стандартной форме.

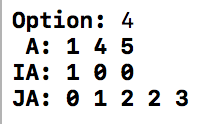
1. Умножение в разреженной форме.

1. Вывод матрицы в стандартной форме.



1. Вывод матрицы в разреженной форме.



**Оценка эффективности**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Matrix size | Fullness | Time, secs | | Memory, bytes | |
| Ordinaary | Sparse | Ordinary | Sparse |
| 100 | 1 | 0.000055 | 0.000006 | 40400 | 2008 |
| 5 | 0.000052 | 0.000004 | 5020 |
| 25 | 0.000053 | 0.000035 | 35140 |
| 26 | 0.000079 | 0.000037 | 37148 |
| 27 | 0.000076 | 0.000041 | 39156 |
| 28 | 0.000075 | 0.000044 | 41164 |
| 29 | 0.000055 | 0.000051 | 43172 |
| 30 | 0.000052 | 0.000054 | 45180 |
| 50 | 0.000051 | 0.000233 | 50200 |
| 100 | 0.000052 | 0.001784 | 100400 |
| 500 | 1 | 0.001001 | 0.000019 | 1002000 | 26008 |
| 5 | 0.001048 | 0.000142 | 130040 |
| 16 | 0.001927 | 0.001700 | 156268 |
| 17 | 0.002011 | 0.002590 | 182276 |
| 25 | 0.002108 | 0.003593 | 650200 |
| 30 | 0.002175 | 0.005969 | 910280 |
| 47 | 0.002229 | 0.022661 | 962296 |
| 48 | 0.002220 | 0.024083 | 988304 |
| 49 | 0.002312 | 0.025472 | 1014312 |
| 50 | 0.002385 | 0.027101 | 1300400 |
| 100 | 0.002511 | 0.223622 | 2600800 |
| 1000 | 1 | 0.026996 | 0.000024 | 4004004 | 92008 |
| 5 | 0.006353 | 0.000249 | 460040 |
| 20 | 0.006835 | 0.009894 | 552048 |
| 21 | 0.007435 | 0.010021 | 644068 |
| 25 | 0.008441 | 0.026996 | 2300200 |
| 43 | 0.006386 | 0.145412 | 3956344 |
| 44 | 0.006467 | 0.149931 | 4048352 |
| 45 | 0.006400 | 0.158795 | 2760240 |
| 50 | 0.006872 | 0.229384 | 4600400 |
| 100 | 0.006403 | 1.811114 | 9200800 |

В ходе тестирования были получены следующие результаты:

Эффективность использования разреженной матрицы по времени:

100 элементов: при заполненности до 30%

500 элементов: при заполненности до 17%

1000 элементов: при заполненности до 21%

Эффективность использования разреженной матрицы по памяти:

100 элементов: при заполненности до 28%

500 элементов: при заполненности до 49%

1000 элементов: при заполненности до 50%

**Вывод:**

Оценивая результаты сравнения времени и объема памяти, мы пришли к выводу, что использование разреженной матрицы выгодно при малом проценте заполнения и большой размерности матрицы. Мы получаем выигрыш по памяти примерно при заполении матрицы примерно до 25%, а выигрыш во времени примерно при 5-7%. При заполнении матрицы менее, чем на 7% мы используем памяти до 88% меньше, а также уменьшаем время выполнения операций.

**Ответы на вопросы.**

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

Разреженная матрица – матрица с преимущественно нулевыми элементами. Число ненулевых элементов при матрице порядка n может выражаться как n ^ (1 - g), где g < 1. Значения g лежат в интервале 0.2 ... 0.5.

Существуют различные методы хранения элементов матрицы в памяти. Например, линейный связный список, т.е. последовательность ячеек, связанных в определенном порядке. Каждая ячейка списка содержит элемент списка и указатель на положение следующей ячейки. Можно хранить матрицу, используя кольцевой связный список, двунаправленные стеки и очереди. Существует диагональная схема хранения симметричных матриц, а также связные схемы разреженного хранения. Связная схема хранения матриц, предложенная Кнутом, предлагает хранить в массиве (например, в AN) в произвольном порядке сами элементы, индексы строк и столбцов соответствующих элементов (например, в массивах I и J), номер (из массива AN) следующего ненулевого элемента, расположенного в матрице по строке (NR) и по столбцу (NC), а также номера элементов, с которых начинается строка (указатели для входа в строку – JR) и номера элементов, с которых начинается столбец (указатели для входа в столбец JC). Наиболее широко используемая схема хранения разреженных матриц — это схема, называемая: "разреженный строчный формат". В этом случае значения ненулевых элементов хранятся в массив е AN, соответствующие им столбцовые индексы — в массиве JA. Кроме того, используется массив указателей, например IA, отмечающих позиции AN и JA, с которых начинаются описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN.

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

Для обычной матрицы выделяется память N\*M\*sizeof(element), где N\*M – размер матрицы. В разреженной же таблице количество выделяемой памяти зависит от метода её хранения.

3. Каков принцип обработки разреженной матрицы?

Алгоритмы обработки разреженных матриц предусматривают действия только с ненулевыми элементами и, таким образом, количество операций будет пропорционально количеству ненулевых элементов.